CANN
7.0.0

AscendCL 应用软件开发指南 (C&C++, 开放态)

文档版本 01

发布日期 2024-11-26

华为技术有限公司

版权所有 © 华为技术有限公司 2024。 保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明

HUAWEI和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

安全声明

产品生命周期政策

华为公司对产品生命周期的规定以“产品生命周期终止政策”为准，该政策的详细内容请参见如下网址：
https://support.huawei.com/ecolumnsweb/zh/warranty-policy

漏洞处理流程

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：
https://www.huawei.com/cn/psirt/vul-response-process
如企业客户须获取漏洞信息，请参见如下网址：
https://securitybulletin.huawei.com/enterprise/cn/security-advisory

华为初始证书权责说明

华为公司对随设备出厂的初始数字证书，发布了“华为设备初始数字证书权责说明”，该说明的详细内容请参见如下网址：
https://support.huawei.com/enterprise/zh/bulletins-service/ENEWS2000015766

华为企业业务最终用户许可协议(EULA)

本最终用户许可协议是最终用户（个人、公司或其他任何实体）与华为公司就华为软件的使用所缔结的协议。最终用户对华为软件的使用受本协议约束，该协议的详细内容请参见如下网址：
https://e.huawei.com/cn/about/eula

产品资料生命周期策略

华为公司针对随产品版本发布的售后客户资料（产品资料），发布了“产品资料生命周期策略”，该策略的详细内容请参见如下网址：
https://support.huawei.com/enterprise/zh/bulletins-website/ENEWS2000017760

目 录

1 学习向导.....1
2 快速入门.....3
3 AscendCL 概述.....13
3.1 AscendCL 架构及基本概念.....13
3.2 AscendCL 接口调用流程.....20
3.3 准备开发和运行环境.....26
4 基础推理应用.....28
4.1 推理应用开发流程.....28
4.2 模型构建.....30
4.3 AscendCL 初始化与去初始化.....31
4.4 运行管理资源申请与释放.....31
4.5 数据传输.....35
4.6 模型推理.....39
4.6.1 模型加载.....39
4.6.2 模型执行.....42
4.6.3 模型卸载.....49
5 媒体数据处理（含图像/视频等）.....50
5.1 媒体数据处理基础知识.....50
5.2 媒体数据处理 V1 与 V2 版本的差别.....56
5.3 各版本功能支持度说明.....56
5.4 DVPP 图像/视频处理（媒体数据处理 V1）.....57
5.4.1 VPC 图像处理典型功能.....57
5.4.2 JPEGD 图像解码.....69
5.4.3 JPEGE 图片编码.....72
5.4.4 PNGD 图片解码.....77
5.4.5 VDEC 视频解码.....80
5.4.6 VENC 视频编码.....90
5.5 DVPP 图像/视频处理（媒体数据处理 V2）.....95
5.5.1 VPC 图片处理典型功能.....95
5.5.2 JPEGD 图片解码.....101
5.5.3 JPEGE 图片编码.....105
5.5.4 PNGD 图片解码.....110

5.5.5 VDEC 视频解码 113
5.5.6 VENC 视频编码 118
5.6 精度提升建议 128
5.6.1 JPEGD+VPC+模型推理精度提升建议（Atlas 200/300/500 推理产品） 128
5.7 高性能编程建议 130
5.7.1 使用媒体数据处理 V1 版本接口 130
5.7.1.1 采用 VPC 多功能组合接口，减少系统调度压力，性能更优 130
5.7.1.2 采用 VPC 批处理接口，降低时延，性能更优 132
5.7.1.3 合理选择 VDEC 视频解码输出格式和分辨率，性能更优 133
5.7.1.4 合理使用 VDEC 解码跳帧，减少内存申请，减轻 VPC 压力，性能更优 134
5.7.1.5 VPC 处理时合理选择输出格式，降低内存申请，性能更优 134
5.7.2 使用媒体数据处理 V2 版本接口 135
5.7.2.1 采用 VPC 多功能组合接口，减少系统调度压力，性能更优 135
5.7.2.2 采用 VPC 批处理接口，降低时延，性能更优 136
5.7.2.3 合理选择 VDEC 视频解码输出格式和分辨率，性能更优 136
5.7.2.4 合理使用 VDEC 解码跳帧，减少内存申请，减轻 VPC 压力，性能更优 137
5.7.2.5 VPC 处理时合理选择输出格式，降低内存申请，性能更优 138
5.7.2.6 合理设置队列深度，减少硬件资源浪费，提升性能 138
6 单算子调用 140
6.1 单算子调用基础知识 140
6.2 单算子调用流程 143
6.3 单算子模型执行 146
6.3.1 调用 CBLAS 接口执行算子示例代码 146
6.3.2 执行固定 Shape 算子示例代码 149
6.3.3 执行动态 Shape 算子示例代码（不注册算子选择器） 150
6.3.4 执行动态 Shape 算子示例代码（注册算子选择器） 152
7 扩展更多特性 155
7.1 内存二次分配管理 155
7.2 多 Device 切换 158
7.3 多模型串联推理 159
7.4 多 Batch 模型推理 160
7.5 队列方式模型推理 161
7.6 异步模型推理 163
7.7 模型动态 Shape 输入推理 167
7.7.1 使用约束 167
7.7.2 动态 Batch/动态分辨率/动态维度（设置多档维度值） 168
7.7.3 动态 Shape 输入（设置 Shape 范围） 172
7.8 模型动态 AIPP 推理 173
7.8.1 使用约束 174
7.8.2 动态 AIPP（单个动态 AIPP 输入） 174
7.8.3 动态 AIPP（多个动态 AIPP 输入） 177
7.9 Stream 管理 178

7.10 同步等待 182
7.11 AI Core 异常信息获取 184
7.12 Profiling 性能数据采集 185
7.13 溢出算子数据采集及分析 191
7.14 特征向量检索 194
7.15 共享 Buffer 管理 198
8 应用调试 202
9 精度/性能优化 205
9.1 调优简介 205
9.2 模型推理精度提升建议 205
9.2.1 简介 205
9.2.2 DVPP+AIPP+模型推理精度提升建议（Atlas 200/300/500 推理产品） 206
9.2.3 算子精度导致推理结果不达标 207
9.2.3.1 问题描述 207
9.2.3.2 问题定位流程 207
9.2.3.3 配置精度模式 209
9.2.3.4 关闭数据缓存优化 210
9.2.3.5 关闭融合规则 210
9.2.3.6 检查数据处理或配置 211
9.2.3.7 案例介绍 211
9.3 模型推理性能调优建议 212
9.3.1 模型推理性能调优思路 212
9.3.2 使用 AOE 工具调优模型 213
9.3.3 采集&解析性能数据 215
9.3.4 使用 AMCT 工具压缩模型 218
9.3.4.1 基于精度的自动量化 218
9.3.4.2 关于增加校准集的建议 218
9.3.4.3 查找量化场景下的精度损失层 219
9.3.4.4 配置 skip_layers 跳过量化损失大的层 219
9.4 DVPP 数据处理高性能编程建议 220
10 应用样例参考 221
10.1 样例列表(Atlas 200/300/500 推理产品) 221
10.2 样例列表(Atlas 推理系列产品) 223
10.3 实现矩阵-矩阵乘运算 224
10.3.1 样例介绍 224
10.3.2 Ascend EP，编译及在 Device 上运行应用（Atlas 200/300/500 推理产品）（Atlas 推理系列产品） 226
10.4 基于 Caffe ResNet-50 网络实现图片分类（图片解码+缩放+同步推理） 228
10.4.1 样例介绍 228
10.4.2 Ascend EP，编译及在 Device 上运行应用（Atlas 200/300/500 推理产品）（Atlas 推理系列产品） 231
10.5 基于 Caffe ResNet-50 网络实现图片分类（图片解码+抠图缩放+图片编码+同步推理） 233
10.5.1 样例介绍 233

10.5.2 Ascend EP，编译及在 Device 上运行应用（Atlas 200/300/500 推理产品）（Atlas 推理系列产品）236
10.6 基于 Caffe ResNet-50 网络实现图片分类（视频解码+同步推理）240
10.6.1 样例介绍 240
10.6.2 Ascend EP，编译及在 Device 上运行应用（Atlas 200/300/500 推理产品）（Atlas 推理系列产品）242
10.7 基于 Caffe ResNet-50 网络实现图片分类（同步推理）245
10.7.1 样例介绍 245
10.7.2 Ascend EP，编译及在 Device 上运行应用（Atlas 200/300/500 推理产品）（Atlas 推理系列产品）247
10.8 基于 Caffe ResNet-50 网络实现图片分类（异步推理）250
10.8.1 样例介绍 250
10.8.2 Ascend EP，编译及在 Device 上运行应用（Atlas 200/300/500 推理产品）（Atlas 推理系列产品）252
10.9 媒体数据处理 V1（抠图，一图多框）255
10.9.1 样例介绍 255
10.9.2 Ascend EP，编译及在 Device 上运行应用（Atlas 200/300/500 推理产品）（Atlas 推理系列产品）257
10.10 媒体数据处理 V1（视频编码）258
10.10.1 样例介绍 258
10.10.2 Ascend EP，编译及在 Device 上运行应用（Atlas 200/300/500 推理产品）（Atlas 推理系列产品）259
10.11 媒体数据处理 V1（抠图贴图）261
10.11.1 样例介绍 261
10.11.2 Ascend EP，编译及在 Device 上运行应用（Atlas 200/300/500 推理产品）（Atlas 推理系列产品）262
10.12 基于 Caffe YOLOv3 网络实现目标检测（动态 Batch/动态分辨率）264
10.12.1 样例介绍 264
10.12.2 Ascend EP，编译及在 Device 上运行应用（Atlas 200/300/500 推理产品）（Atlas 推理系列产品）266
10.13 媒体数据处理 V2（VPC 抠图/贴图/缩放等）269
10.13.1 样例介绍 269
10.13.2 编译及运行应用 270
10.14 媒体数据处理 V2（JPEGD 图片解码）270
10.14.1 样例介绍 270
10.14.2 编译及运行应用 271
10.15 媒体数据处理 V2（JPEGE 图片编码）271
10.15.1 样例介绍 271
10.15.2 编译及运行应用 272
10.16 媒体数据处理 V2（VDEC 视频解码）272
10.16.1 样例介绍 273
10.16.2 编译及运行应用 273
10.17 媒体数据处理 V2（VENC 视频编码）274
10.17.1 样例介绍 274
10.17.2 编译及运行应用 275
10.18 媒体数据处理 V2（PNGD 图片解码）275
10.18.1 样例介绍 275
10.18.2 编译及运行应用 276

11 AscendCL API 参考 277
11.1 接口列表 278
11.1.1 接口版本说明 278
11.1.2 V1.1.0 及以上版本 278
11.1.3 V1.2.0 及以上版本 295
11.1.4 V1.3.0 及以上版本 296
11.1.5 V1.4.0 及以上版本 298
11.1.6 V1.5.0 及以上版本 299
11.1.6.1 接口列表（Atlas 200/300/500 推理产品&Atlas 推理系列产品） 300
11.1.7 V1.6.0 及以上版本 300
11.1.7.1 接口列表（Atlas 200/300/500 推理产品&Atlas 推理系列产品） 301
11.1.8 V1.7.0 及以上版本 303
11.2 废弃接口/返回码列表 305
11.3 同步&异步 API 说明 306
11.4 系统配置 306
11.4.1 aclInit 306
11.4.2 aclFinalize 310
11.4.3 aclrtGetVersion 311
11.4.4 aclSetCompileopt 311
11.4.5 aclGetCompileopt 312
11.4.6 aclGetCompileoptSize 312
11.4.7 aclrtGetSocName 313
11.4.8 aclGetRecentErrMsg 313
11.4.9 aclrtSetDeviceSatMode 314
11.4.10 aclrtGetDeviceSatMode 315
11.5 Device 管理 315
11.5.1 aclrtSetDevice 315
11.5.2 aclrtResetDevice 316
11.5.3 aclrtGetDevice 317
11.5.4 aclrtGetRunMode 317
11.5.5 aclrtSetTsDevice 318
11.5.6 aclrtGetDeviceCount 319
11.5.7 aclrtGetDeviceUtilizationRate 319
11.5.8 aclrtQueryDeviceStatus 320
11.5.9 aclrtSynchronizeDevice 321
11.6 Context 管理 321
11.6.1 aclrtCreateContext 321
11.6.2 aclrtDestroyContext 323
11.6.3 aclrtSetCurrentContext 323
11.6.4 aclrtGetCurrentContext 324
11.6.5 aclrtCtxSetSysParamOpt 324
11.6.6 aclrtCtxGetSysParamOpt 325

11.6.7 aclrtGetOverflowStatus 326
11.6.8 aclrtResetOverflowStatus 326
11.7 算力 Group 查询与设置 327
11.7.1 aclrtSetGroup 327
11.7.2 aclrtGetGroupCount 328
11.7.3 aclrtGetAllGroupInfo 329
11.7.4 aclrtGetGroupInfoDetail 330
11.7.5 aclrtCreateGroupInfo 331
11.7.6 aclrtDestroyGroupInfo 331
11.8 Stream 管理 332
11.8.1 aclrtCreateStream 332
11.8.2 aclrtCreateStreamV2 333
11.8.3 aclrtSetStreamConfigOpt 334
11.8.4 aclrtCreateStreamWithConfig 335
11.8.5 aclrtDestroyStream 336
11.8.6 aclrtDestroyStreamForce 337
11.8.7 aclrtSetStreamOverflowSwitch 338
11.8.8 aclrtGetStreamOverflowSwitch 338
11.8.9 aclrtSetStreamFailureMode 339
11.8.10 aclrtStreamQuery 340
11.8.11 aclrtSynchronizeStream 340
11.8.12 aclrtSynchronizeStreamWithTimeout 341
11.9 Event 管理 341
11.9.1 aclrtCreateEvent 341
11.9.2 aclrtCreateEventWithFlag 342
11.9.3 aclrtDestroyEvent 343
11.9.4 aclrtRecordEvent 344
11.9.5 aclrtResetEvent 345
11.9.6 aclrtQueryEvent 345
11.9.7 aclrtQueryEventStatus 346
11.9.8 aclrtQueryEventWaitStatus 347
11.9.9 aclrtSynchronizeEvent 347
11.9.10 aclrtSynchronizeEventWithTimeout 348
11.9.11 aclrtEventElapsedTime 348
11.9.12 aclrtStreamWaitEvent 349
11.9.13 aclrtSetOpWaitTimeout 350
11.10 执行控制 351
11.10.1 aclrtSubscribeReport 351
11.10.2 aclrtLaunchCallback 352
11.10.3 aclrtProcessReport 353
11.10.4 aclrtUnSubscribeReport 354
11.10.5 aclrtSubscribeHostFunc 355

11.10.6 aclrtProcessHostFunc.....356
11.10.7 aclrtUnSubscribeHostFunc.....356
11.11 内存管理.....357
11.11.1 总体说明.....357
11.11.2 aclrtMalloc.....358
11.11.3 aclrtMallocAlign32.....359
11.11.4 aclrtMallocCached.....360
11.11.5 aclrtMemFlush.....361
11.11.6 aclrtMemInvalidate.....361
11.11.7 aclrtFree.....362
11.11.8 aclrtMallocHost.....362
11.11.9 aclrtFreeHost.....363
11.11.10 aclrtMemset.....364
11.11.11 aclrtMemsetAsync.....365
11.11.12 aclrtMemcpy.....365
11.11.13 aclrtMemcpyAsync.....366
11.11.14 aclrtGetMemInfo.....368
11.11.15 aclrtDeviceCanAccessPeer.....369
11.11.16 aclrtDeviceEnablePeerAccess.....370
11.11.17 aclrtDeviceDisablePeerAccess.....370
11.11.18 aclrtMemcpy2d.....371
11.11.19 aclrtMemcpy2dAsync.....372
11.11.20 aclrtAllocatorRegister.....373
11.11.21 aclrtAllocatorUnregister.....374
11.11.22 aclrtReserveMemAddress.....375
11.11.23 aclrtReleaseMemAddress.....376
11.11.24 aclrtMallocPhysical.....377
11.11.25 aclrtFreePhysical.....378
11.11.26 aclrtMapMem.....378
11.11.27 aclrtUnmapMem.....379
11.12 模型加载与执行.....380
11.12.1 aclmdlLoadFromFile.....380
11.12.2 aclmdlLoadFromMem.....381
11.12.3 aclmdlLoadFromFileWithMem.....382
11.12.4 aclmdlLoadFromMemWithMem.....384
11.12.5 aclmdlLoadFromFileWithQ.....386
11.12.6 aclmdlLoadFromMemWithQ.....388
11.12.7 aclmdlExecute.....389
11.12.8 aclmdlExecuteAsync.....390
11.12.9 aclmdlSetExecConfigOpt.....391
11.12.10 aclmdlExecuteV2.....392
11.12.11 aclmdlExecuteAsyncV2.....394

11.12.12 aclmdlUnload 395
11.12.13 aclmdlQuerySize 395
11.12.14 aclmdlQuerySizeFromMem 396
11.12.15 aclmdlSetDynamicBatchSize 397
11.12.16 aclmdlSetDynamicHWSize 398
11.12.17 aclmdlSetInputAIPP 399
11.12.18 aclmdlGetFirstAippInfo 402
11.12.19 aclmdlGetAippType 403
11.12.20 aclmdlSetAIPPByInputIndex 403
11.12.21 aclmdlSetInputDynamicDims 406
11.12.22 aclmdlCreateAndGetOpDesc 407
11.12.23 aclmdlInitDump 409
11.12.24 aclmdlSetDump 410
11.12.25 aclmdlFinalizeDump 412
11.12.26 aclmdlSetConfigOpt 412
11.12.27 aclmdlLoadWithConfig 414
11.13 单算子 API 执行 415
11.13.1 总体说明 415
11.13.2 功能支持度说明 415
11.13.3 基本概念 415
11.13.3.1 两段式接口 415
11.13.3.2 非连续的 Tensor 416
11.13.3.3 broadcast 关系 417
11.13.3.4 互推导关系 417
11.13.3.5 互转换关系 419
11.13.3.6 数据格式 420
11.13.3.7 数据结构 420
11.13.4 公共接口 421
11.13.5 NN 类算子接口 427
11.13.6 DVPP 类算子接口 427
11.13.6.1 acldvppNormalize 427
11.13.6.2 acldvppDecodeJpeg 429
11.13.6.3 acldvppResize 430
11.13.6.4 acldvppImgToTensor 432
11.13.6.5 acldvppAdjustBrightness 433
11.13.6.6 acldvppAdjustSaturation 434
11.13.6.7 acldvppAdjustHue 436
11.13.6.8 acldvppAdjustContrast 437
11.13.6.9 acldvppHorizontalFlip 439
11.13.6.10 acldvppVerticalFlip 440
11.13.6.11 acldvppGaussianBlur 441
11.13.6.12 acldvppWarpAffine 443

11.13.6.13 acldvppWarpPerspective....444
11.13.6.14 acldvppCropAndResize....446
11.13.6.15 acldvppPad....448
11.13.6.16 acldvpp 返回码....450
11.13.7 融合类算子接口....450
11.13.7.1 aclnnPromptFlashAttention....451
11.13.7.2 aclnnPromptFlashAttentionV2....453
11.13.7.3 aclnnIncreFlashAttention....457
11.13.7.4 aclnnIncreFlashAttentionV2....460
11.13.7.5 aclnnFlashAttentionScore....465
11.13.7.6 aclnnFlashAttentionScoreGrad....468
11.13.7.7 aclnnFFN....472
11.13.7.8 aclnnMoeGatingTopKSoftmax....474
11.13.7.9 aclnnMoeInitRouting....484
11.13.7.10 aclnn 返回码....488
11.14 算子编译....490
11.14.1 aclopRegisterCompileFunc....490
11.14.2 aclopUnregisterCompileFunc....491
11.14.3 aclopCreateKernel....491
11.14.4 aclopSetKernelArgs....492
11.14.5 aclopSetKernelWorkspaceSizes....493
11.14.6 aclopUpdateParams....494
11.14.7 aclopCompile....495
11.14.8 aclopSetCompileFlag....497
11.14.9 aclGenGraphAndDumpForOp....497
11.15 算子加载与执行....499
11.15.1 aclopSetModelDir....499
11.15.2 aclopLoad....500
11.15.3 aclopExecute....501
11.15.4 aclopExecuteV2....503
11.15.5 aclopCompileAndExecute....505
11.15.6 aclopCompileAndExecuteV2....508
11.15.7 aclopExecWithHandle....510
11.15.8 aclopInferShape....511
11.15.9 aclrtSetOpExecuteTimeOut....513
11.15.10 aclopStartDumpArgs....514
11.15.11 aclopStopDumpArgs....515
11.16 CBLAS 接口....516
11.16.1 总体说明....516
11.16.2 关于 A、B、C 矩阵的 Shape 及内存大小计算公式....516
11.16.3 aclblasGemvEx....517
11.16.4 aclblasCreateHandleForGemvEx....519

11.16.5 aclblasHgemv 520
11.16.6 aclblasCreateHandleForHgemv 521
11.16.7 aclblasS8gemv 522
11.16.8 aclblasCreateHandleForS8gemv 523
11.16.9 aclblasGemmEx 524
11.16.10 aclblasCreateHandleForGemmEx 526
11.16.11 aclblasHgemm 528
11.16.12 aclblasCreateHandleForHgemm 529
11.16.13 aclblasS8gemm 530
11.16.14 aclblasCreateHandleForS8gemm 532
11.16.15 aclopCast 533
11.16.16 aclopCreateHandleForCast 534
11.17 媒体数据处理 V1 534
11.17.1 总体说明 535
11.17.2 基本概念 538
11.17.3 内存申请与释放 539
11.17.3.1 acldvppMalloc 539
11.17.3.2 acldvppFree 540
11.17.4 通道创建与释放 540
11.17.4.1 VPC/JPEGD/JPEGE/PNGD 图片处理通道 540
11.17.4.1.1 acldvppCreateChannel 541
11.17.4.1.2 acldvppDestroyChannel 542
11.17.4.2 VDEC 视频解码通道 542
11.17.4.2.1 aclvdecCreateChannel 542
11.17.4.2.2 aclvdecDestroyChannel 543
11.17.4.3 VENC 视频编码通道 544
11.17.4.3.1 aclvencCreateChannel 544
11.17.4.3.2 aclvencDestroyChannel 545
11.17.5 VPC 功能 546
11.17.5.1 功能说明 546
11.17.5.2 约束说明 549
11.17.5.3 性能指标说明 555
11.17.5.4 acldvppVpcResizeAsync 559
11.17.5.5 acldvppVpcCropAsync 561
11.17.5.6 acldvppVpcCropResizeAsync 564
11.17.5.7 acldvppVpcBatchCropAsync 566
11.17.5.8 acldvppVpcBatchCropResizeAsync 569
11.17.5.9 acldvppVpcCropAndPasteAsync 572
11.17.5.10 acldvppVpcCropResizePasteAsync 574
11.17.5.11 acldvppVpcBatchCropAndPasteAsync 577
11.17.5.12 acldvppVpcBatchCropResizePasteAsync 580
11.17.5.13 acldvppVpcBatchCropResizeMakeBorderAsync 583

11.17.5.14 acldvppVpcConvertColorAsync....587
11.17.5.15 acldvppVpcPyrDownAsync....588
11.17.5.16 acldvppVpcEqualizeHistAsync....590
11.17.5.17 acldvppVpcMakeBorderAsync....591
11.17.5.18 acldvppVpcCalcHistAsync....592
11.17.6 JPEGD 功能....594
11.17.6.1 功能及约束说明....594
11.17.6.2 性能指标说明....598
11.17.6.3 acldvppJpegDecodeAsync....599
11.17.6.4 acldvppJpegGetImageInfo....601
11.17.6.5 acldvppJpegGetImageInfoV2....601
11.17.6.6 acldvppJpegPredictDecSize....602
11.17.7 JPEGE 功能....603
11.17.7.1 功能及约束说明....603
11.17.7.2 性能指标说明....605
11.17.7.3 acldvppJpegEncodeAsync....605
11.17.7.4 acldvppJpegPredictEncSize....607
11.17.8 PNGD 功能....607
11.17.8.1 功能及约束说明....608
11.17.8.2 性能指标说明....609
11.17.8.3 acldvppPngDecodeAsync....610
11.17.8.4 acldvppPngGetImageInfo....612
11.17.8.5 acldvppPngPredictDecSize....613
11.17.9 VDEC 功能....613
11.17.9.1 功能及约束说明....614
11.17.9.2 性能指标说明....615
11.17.9.3 aclvdecSendFrame....619
11.17.9.4 aclvdecSendSkippedFrame....621
11.17.9.5 aclvdecCallback....622
11.17.10 VENC 功能....623
11.17.10.1 功能及约束说明....623
11.17.10.2 性能指标说明....625
11.17.10.3 aclvencSendFrame....625
11.17.10.4 aclvencCallback....627
11.18 媒体数据处理 V2....627
11.18.1 总体功能及约束说明....627
11.18.2 基本概念....632
11.18.3 公共接口....632
11.18.3.1 hi_mpi_sys_init....633
11.18.3.2 hi_mpi_sys_exit....633
11.18.3.3 hi_mpi_dvpp_malloc....634
11.18.3.4 hi_mpi_dvpp_free....635

11.18.3.5 hi_mpi_dvpp_get_image_info....636
11.18.3.6 hi_mpi_dvpp_get_version....636
11.18.3.7 hi_mpi_sys_create_epoll....637
11.18.3.8 hi_mpi_sys_ctl_epoll....638
11.18.3.9 hi_mpi_sys_wait_epoll....639
11.18.3.10 hi_mpi_sys_close_epoll....640
11.18.3.11 hi_mpi_sys_set_chn_csc_matrix....641
11.18.3.12 hi_mpi_sys_get_chn_csc_matrix....642
11.18.3.13 hi_mpi_sys_get_image_align_info....643
11.18.3.14 hi_mpi_sys_bind....644
11.18.3.15 hi_mpi_sys_unbind....645
11.18.4 AI 音频输入/AO 音频输出....646
11.18.4.1 功能及约束说明....646
11.18.4.2 hi_mpi_ai_set_pub_attr....647
11.18.4.3 hi_mpi_ai_enable....647
11.18.4.4 hi_mpi_ai_disable....648
11.18.4.5 hi_mpi_ai_set_chn_attr....648
11.18.4.6 hi_mpi_ai_enable_chn....649
11.18.4.7 hi_mpi_ai_disable_chn....650
11.18.4.8 hi_mpi_ai_get_frame....651
11.18.4.9 hi_mpi_ai_release_frame....652
11.18.4.10 hi_mpi_ai_enable_resample....653
11.18.4.11 hi_mpi_ai_disable_resample....654
11.18.4.12 hi_mpi_ao_set_pub_attr....654
11.18.4.13 hi_mpi_ao_enable....655
11.18.4.14 hi_mpi_ao_disable....656
11.18.4.15 hi_mpi_ao_enable_chn....656
11.18.4.16 hi_mpi_ao_disable_chn....657
11.18.4.17 hi_mpi_ao_send_frame....658
11.18.4.18 hi_mpi_ao_get_chn_delay....659
11.18.4.19 hi_mpi_ao_enable_resample....660
11.18.4.20 hi_mpi_ao_disable_resample....661
11.18.5 AENC 音频编码/ADEC 音频解码....661
11.18.5.1 功能及约束说明....661
11.18.5.2 hi_mpi_aenc_create_chn....663
11.18.5.3 hi_mpi_aenc_destroy_chn....664
11.18.5.4 hi_mpi_aenc_get_stream....664
11.18.5.5 hi_mpi_aenc_release_stream....665
11.18.5.6 hi_mpi_adec_create_chn....666
11.18.5.7 hi_mpi_adec_destroy_chn....666
11.18.5.8 hi_mpi_adec_send_stream....667
11.18.6 音量调节....668

11.18.6.1 功能描述 668
11.18.6.2 HI_ACODEC_SET_DACL_VOLUME 668
11.18.6.3 HI_ACODEC_SET_DACR_VOLUME 669
11.18.6.4 HI_ACODEC_GET_DACL_VOLUME 670
11.18.6.5 HI_ACODEC_GET_DACR_VOLUME 671
11.18.6.6 HI_ACODEC_SET_ADCL_VOLUME 671
11.18.6.7 HI_ACODEC_SET_ADCR_VOLUME 672
11.18.6.8 HI_ACODEC_GET_ADCL _VOLUME 673
11.18.6.9 HI_ACODEC_GET_ADCR _VOLUME 674
11.18.7 ISP 系统控制及 3A 算法注册 674
11.18.7.1 功能描述 674
11.18.7.2 基本概念 675
11.18.7.3 hi_mpi_isp_sensor_reg_callback 676
11.18.7.4 hi_mpi_isp_sensor_unreg_callback 676
11.18.7.5 hi_mpi_isp_mem_init 677
11.18.7.6 hi_mpi_isp_set_pub_attr 678
11.18.7.7 hi_mpi_isp_get_pub_attr 679
11.18.7.8 hi_mpi_isp_init 680
11.18.7.9 hi_mpi_isp_run 680
11.18.7.10 hi_mpi_isp_exit 681
11.18.7.11 hi_mpi_isp_set_sns_slave_attr 682
11.18.7.12 hi_mpi_isp_get_sns_slave_attr 683
11.18.7.13 hi_mpi_ae_register 683
11.18.7.14 hi_mpi_ae_unregister 684
11.18.7.15 hi_mpi_ae_sensor_reg_callback 685
11.18.7.16 hi_mpi_ae_sensor_unreg_callback 685
11.18.7.17 hi_mpi_awb_register 686
11.18.7.18 hi_mpi_awb_unregister 687
11.18.7.19 hi_mpi_awb_sensor_reg_callback 688
11.18.7.20 hi_mpi_awb_sensor_unreg_callback 689
11.18.8 MIPI Rx ioctl 命令字 689
11.18.8.1 功能描述 689
11.18.8.2 基本概念 691
11.18.8.3 HI_MIPI_SET_DEV_ATTR 693
11.18.8.4 HI_MIPI_SET_HS_MODE 693
11.18.8.5 HI_MIPI_ENABLE_MIPI_CLOCK 694
11.18.8.6 HI_MIPI_DISABLE_MIPI_CLOCK 694
11.18.8.7 HI_MIPI_RESET_MIPI 695
11.18.8.8 HI_MIPI_UNRESET_MIPI 695
11.18.8.9 HI_MIPI_ENABLE_SLVS_CLOCK 696
11.18.8.10 HI_MIPI_DISABLE_SLVS_CLOCK 696
11.18.8.11 HI_MIPI_RESET_SLVS 697

11.18.8.12 HI_MIPI_UNRESET_SLVS....697
11.18.8.13 HI_MIPI_CONFIG_SENSOR_CLOCK....698
11.18.8.14 HI_MIPI_ENABLE_SENSOR_CLOCK....698
11.18.8.15 HI_MIPI_DISABLE_SENSOR_CLOCK....699
11.18.8.16 HI_MIPI_RESET_SENSOR....699
11.18.8.17 HI_MIPI_UNRESET_SENSOR....700
11.18.9 VI 视频输入功能....700
11.18.9.1 功能说明....700
11.18.9.2 约束说明....701
11.18.9.3 基本概念....701
11.18.9.4 hi_mpi_vi_set_dev_bind_pipe....704
11.18.9.5 hi_mpi_vi_get_dev_bind_pipe....705
11.18.9.6 hi_mpi_vi_set_dev_attr....706
11.18.9.7 hi_mpi_vi_get_dev_attr....707
11.18.9.8 hi_mpi_vi_enable_dev....707
11.18.9.9 hi_mpi_vi_disable_dev....708
11.18.9.10 hi_mpi_vi_create_pipe....709
11.18.9.11 hi_mpi_vi_destroy_pipe....709
11.18.9.12 hi_mpi_vi_set_vi_vpss_mode....710
11.18.9.13 hi_mpi_vi_get_vi_vpss_mode....711
11.18.9.14 hi_mpi_vi_set_pipe_attr....712
11.18.9.15 hi_mpi_vi_get_pipe_attr....712
11.18.9.16 hi_mpi_vi_start_pipe....713
11.18.9.17 hi_mpi_vi_stop_pipe....714
11.18.9.18 hi_mpi_vi_set_pipe_pre_crop....714
11.18.9.19 hi_mpi_vi_get_pipe_pre_crop....715
11.18.9.20 hi_mpi_vi_set_pipe_pre_be_crop....716
11.18.9.21 hi_mpi_vi_set_pipe_bind_strobe....717
11.18.9.22 hi_mpi_vi_set_pipe_frame_dump_attr....717
11.18.9.23 hi_mpi_vi_get_pipe_frame_dump_attr....718
11.18.9.24 hi_mpi_vi_get_pipe_frame....719
11.18.9.25 hi_mpi_vi_release_pipe_frame....720
11.18.9.26 hi_mpi_vi_set_pipe_frame_source....720
11.18.9.27 hi_mpi_vi_get_pipe_frame_source....721
11.18.9.28 hi_mpi_vi_send_pipe_yuv....722
11.18.9.29 hi_mpi_vi_send_pipe_raw....723
11.18.9.30 hi_mpi_vi_pipe_get_buffer....724
11.18.9.31 hi_mpi_vi_pipe_release_buffer....725
11.18.9.32 hi_mpi_vi_set_pipe_vc_number....726
11.18.9.33 hi_mpi_vi_get_pipe_vc_number....726
11.18.9.34 hi_mpi_vi_get_pipe_fd....727
11.18.9.35 hi_mpi_vi_set_chn_attr....728

11.18.9.36 hi_mpi_vi_get_chn_attr.....728
11.18.9.37 hi_mpi_vi_enable_chn.....729
11.18.9.38 hi_mpi_vi_disable_chn.....730
11.18.9.39 hi_mpi_vi_set_chn_dis_config.....731
11.18.9.40 hi_mpi_vi_get_chn_dis_config.....731
11.18.9.41 hi_mpi_vi_set_chn_dis_attr.....732
11.18.9.42 hi_mpi_vi_get_chn_dis_attr.....733
11.18.9.43 hi_mpi_vi_set_chn_dis_param.....734
11.18.9.44 hi_mpi_vi_get_chn_dis_param.....734
11.18.9.45 hi_mpi_vi_set_chn_ldc_attr.....735
11.18.9.46 hi_mpi_vi_get_chn_ldc_attr.....736
11.18.9.47 hi_mpi_vi_get_chn_frame.....737
11.18.9.48 hi_mpi_vi_release_chn_frame.....738
11.18.9.49 hi_mpi_vi_get_chn_fd.....739
11.18.9.50 hi_mpi_vi_set_chn_low_delay_attr.....739
11.18.9.51 hi_mpi_vi_get_chn_low_delay_attr.....740
11.18.10 Region 区域管理.....741
11.18.10.1 功能及约束说明.....741
11.18.10.2 hi_mpi_rgn_create.....743
11.18.10.3 hi_mpi_rgn_destroy.....744
11.18.10.4 hi_mpi_rgn_attach_to_chn.....744
11.18.10.5 hi_mpi_rgn_detach_from_chn.....745
11.18.10.6 hi_mpi_rgn_set_display_attr.....746
11.18.10.7 hi_mpi_rgn_get_display_attr.....747
11.18.10.8 hi_mpi_rgn_get_canvas_info.....748
11.18.10.9 hi_mpi_rgn_update_canvas.....748
11.18.11 VPSS 视频处理功能.....749
11.18.11.1 功能说明.....749
11.18.11.2 约束说明.....750
11.18.11.3 hi_mpi_vpss_create_grp.....751
11.18.11.4 hi_mpi_vpss_destroy_grp.....752
11.18.11.5 hi_mpi_vpss_start_grp.....753
11.18.11.6 hi_mpi_vpss_stop_grp.....754
11.18.11.7 hi_mpi_vpss_set_grp_crop.....754
11.18.11.8 hi_mpi_vpss_get_grp_crop.....755
11.18.11.9 hi_mpi_vpss_set_grp_param.....756
11.18.11.10 hi_mpi_vpss_get_grp_param.....756
11.18.11.11 hi_mpi_vpss_set_grp_fisheye_cfg.....757
11.18.11.12 hi_mpi_vpss_get_grp_fisheye_cfg.....758
11.18.11.13 hi_mpi_vpss_set_chn_attr.....759
11.18.11.14 hi_mpi_vpss_get_chn_attr.....759
11.18.11.15 hi_mpi_vpss_enable_chn.....760

11.18.11.16 hi_mpi_vpss_disable_chn.....761
11.18.11.17 hi_mpi_vpss_get_chn_frame.....762
11.18.11.18 hi_mpi_vpss_release_chn_frame.....763
11.18.11.19 hi_mpi_vpss_set_chn_fisheye.....764
11.18.11.20 hi_mpi_vpss_get_chn_fisheye.....765
11.18.11.21 hi_mpi_vpss_get_chn_fd.....766
11.18.12 VO 视频输出功能.....766
11.18.12.1 hi_mpi_vo_set_pub_attr.....766
11.18.12.2 hi_mpi_vo_enable.....767
11.18.12.3 hi_mpi_vo_disable.....768
11.18.12.4 hi_mpi_vo_set_video_layer_attr.....769
11.18.12.5 hi_mpi_vo_get_video_layer_attr.....769
11.18.12.6 hi_mpi_vo_enable_video_layer.....770
11.18.12.7 hi_mpi_vo_disable_video_layer.....771
11.18.12.8 hi_mpi_vo_set_chn_attr.....771
11.18.12.9 hi_mpi_vo_enable_chn.....772
11.18.12.10 hi_mpi_vo_disable_chn.....773
11.18.12.11 hi_mpi_vo_set_chn_param.....774
11.18.12.12 hi_mpi_vo_set_chn_frame_rate.....775
11.18.12.13 hi_mpi_vo_pause_chn.....776
11.18.12.14 hi_mpi_vo_resume_chn.....777
11.18.12.15 hi_mpi_vo_hide_chn.....778
11.18.12.16 hi_mpi_vo_show_chn.....779
11.18.12.17 hi_mpi_vo_send_frame.....779
11.18.12.18 hi_mpi_vo_create_pool.....780
11.18.12.19 hi_mpi_vo_handle_to_phys_addr.....781
11.18.12.20 hi_mpi_vo_destroy_pool.....781
11.18.12.21 hi_mpi_vo_bind_layer.....782
11.18.12.22 hi_mpi_vo_unbind_layer.....783
11.18.13 TDE 图形绘制功能.....783
11.18.13.1 hi_tde_open.....783
11.18.13.2 hi_tde_close.....784
11.18.13.3 hi_tde_begin_job.....785
11.18.13.4 hi_tde_end_job.....786
11.18.13.5 hi_tde_cancel_job.....787
11.18.13.6 hi_tde_wait_for_done.....787
11.18.13.7 hi_tde_wait_all_done.....788
11.18.13.8 hi_tde_quick_copy.....788
11.18.13.9 hi_tde_quick_fill.....790
11.18.13.10 hi_tde_pattern_fill.....791
11.18.14 HIFB 叠加图形层管理功能.....793
11.18.14.1 功能及约束说明.....793

11.18.14.2 FBIOGET_VSCREENINFO....794
11.18.14.3 FBIOPUT_VSCREENINFO....795
11.18.14.4 FBIOGET_FSCREENINFO....796
11.18.14.5 FBIOPAN_DISPLAY....797
11.18.14.6 FBIOGET_CMAP....798
11.18.14.7 FBIOPUT_CMAP....799
11.18.14.8 FBIOGET_SCREEN_ORIGIN_HIFB....800
11.18.14.9 FBIOPUT_SCREEN_ORIGIN_HIFB....800
11.18.14.10 FBIOGET_SHOW_HIFB....801
11.18.14.11 FBIOPUT_SHOW_HIFB....802
11.18.14.12 FBIOGET_ALPHA_HIFB....803
11.18.14.13 FBIOPUT_ALPHA_HIFB....804
11.18.15 HDMI 外设....804
11.18.15.1 hi_mpi_hdmi_init....804
11.18.15.2 hi_mpi_hdmi_deinit....805
11.18.15.3 hi_mpi_hdmi_open....806
11.18.15.4 hi_mpi_hdmi_close....806
11.18.15.5 hi_mpi_hdmi_set_attr....807
11.18.15.6 hi_mpi_hdmi_get_attr....808
11.18.15.7 hi_mpi_hdmi_start....809
11.18.15.8 hi_mpi_hdmi_stop....809
11.18.15.9 hi_mpi_hdmi_set_infoframe....810
11.18.15.10 hi_mpi_hdmi_get_sink_capability....811
11.18.15.11 hi_mpi_hdmi_register_callback....812
11.18.16 VPC 图像处理功能....812
11.18.16.1 功能说明....813
11.18.16.2 约束说明....817
11.18.16.3 性能指标说明....821
11.18.16.4 hi_mpi_vpc_create_chn....823
11.18.16.5 hi_mpi_vpc_destroy_chn....824
11.18.16.6 hi_mpi_vpc_resize....824
11.18.16.7 hi_mpi_vpc_crop....827
11.18.16.8 hi_mpi_vpc_crop_resize....828
11.18.16.9 hi_mpi_vpc_crop_resize_paste....830
11.18.16.10 hi_mpi_vpc_convert_color....832
11.18.16.11 hi_mpi_vpc_convert_color_v2....834
11.18.16.12 hi_mpi_vpc_convert_color_to_yuv420....835
11.18.16.13 hi_mpi_vpc_copy_make_border....837
11.18.16.14 hi_mpi_vpc_pyrdown....839
11.18.16.15 hi_mpi_vpc_calc_hist....840
11.18.16.16 hi_mpi_vpc_equalize_hist....842
11.18.16.17 hi_mpi_vpc_crop_resize_make_border....843

11.18.16.18 hi_mpi_vpc_batch_crop_resize_paste....846
11.18.16.19 hi_mpi_vpc_batch_crop_resize_make_border....848
11.18.16.20 hi_mpi_vpc_get_process_result....851
11.18.16.21 hi_mpi_vpc_sys_create_chn....852
11.18.16.22 hi_mpi_vpc_set_roundview_stitching_param....852
11.18.16.23 hi_mpi_vpc_get_roundview_stitching_param....853
11.18.16.24 hi_mpi_vpc_roundview_stitching....854
11.18.16.25 hi_mpi_vpc_crop_resize_resize_paste....855
11.18.16.26 hi_mpi_vpc_set_chn_workspace....857
11.18.16.27 hi_mpi_vpc_get_lut_mem_size....857
11.18.16.28 hi_mpi_vpc_get_affine_lut....858
11.18.16.29 hi_mpi_vpc_get_perspective_lut....859
11.18.16.30 hi_mpi_vpc_get_remap_lut....860
11.18.16.31 hi_mpi_vpc_lut_remap....861
11.18.16.32 hi_mpi_vpc_median_blur....862
11.18.16.33 hi_mpi_vpc_erode....863
11.18.16.34 hi_mpi_vpc_dilate....864
11.18.16.35 hi_mpi_vpc_blur....864
11.18.16.36 hi_mpi_vpc_gaussian_blur....865
11.18.16.37 hi_mpi_vpc_filter2d....866
11.18.16.38 hi_mpi_vpc_rotate....867
11.18.16.39 hi_mpi_vpc_draw_mosaic....868
11.18.16.40 hi_mpi_vpc_draw_cover....869
11.18.16.41 hi_mpi_vpc_draw_line....870
11.18.16.42 hi_mpi_vpc_draw_osd....871
11.18.16.43 hi_mpi_vpc_get_affine_transform....872
11.18.16.44 hi_mpi_vpc_get_rotation_matrix....873
11.18.16.45 hi_mpi_vpc_warp_affine....874
11.18.16.46 hi_mpi_vpc_flip....875
11.18.16.47 hi_mpi_vpc_set_chn_opt_attr....875
11.18.17 VDEC 视频解码功能/JPEGD 图片解码功能....876
11.18.17.1 JPEGD 功能及约束说明....876
11.18.17.2 JPEGD 性能指标数据....880
11.18.17.3 VDEC 功能及约束说明....880
11.18.17.4 VDEC 性能指标数据....882
11.18.17.5 hi_mpi_vdec_create_chn....884
11.18.17.6 hi_mpi_vdec_destroy_chn....886
11.18.17.7 hi_mpi_vdec_get_chn_attr....887
11.18.17.8 hi_mpi_vdec_set_chn_attr....887
11.18.17.9 hi_mpi_vdec_start_recv_stream....888
11.18.17.10 hi_mpi_vdec_stop_recv_stream....889
11.18.17.11 hi_mpi_vdec_query_status....890

11.18.17.12 hi_mpi_vdec_reset_chn....890
11.18.17.13 hi_mpi_vdec_set_chn_param....891
11.18.17.14 hi_mpi_vdec_get_chn_param....892
11.18.17.15 hi_mpi_vdec_set_protocol_param....893
11.18.17.16 hi_mpi_vdec_get_protocol_param....894
11.18.17.17 hi_mpi_vdec_send_stream....894
11.18.17.18 hi_mpi_vdec_get_frame....896
11.18.17.19 hi_mpi_vdec_release_frame....898
11.18.17.20 hi_mpi_vdec_get_fd....899
11.18.17.21 hi_mpi_vdec_close_fd....899
11.18.17.22 hi_vdec_get_pic_buf_size....900
11.18.17.23 hi_vdec_get_tmv_buf_size....901
11.18.17.24 hi_mpi_vdec_set_jpegd_precision_mode....901
11.18.17.25 hi_mpi_vdec_get_jpegd_output_info....902
11.18.17.26 hi_mpi_vdec_set_display_mode....903
11.18.17.27 hi_mpi_vdec_get_display_mode....904
11.18.18 VENC 视频编码功能/JPEGE 图片编码功能....905
11.18.18.1 JPEGE 功能及约束说明....905
11.18.18.2 JPEGE 性能指标说明....907
11.18.18.3 VENC 功能及约束说明....908
11.18.18.4 VENC 性能指标说明....909
11.18.18.5 hi_mpi_venc_create_chn....910
11.18.18.6 hi_mpi_venc_destroy_chn....912
11.18.18.7 hi_mpi_venc_start_chn....913
11.18.18.8 hi_mpi_venc_stop_chn....914
11.18.18.9 hi_mpi_venc_query_status....915
11.18.18.10 hi_mpi_venc_get_stream....916
11.18.18.11 hi_mpi_venc_release_stream....919
11.18.18.12 hi_mpi_venc_send_frame....920
11.18.18.13 hi_mpi_venc_set_mod_param....921
11.18.18.14 hi_mpi_venc_get_mod_param....922
11.18.18.15 hi_mpi_venc_request_idr....923
11.18.18.16 hi_mpi_venc_get_fd....923
11.18.18.17 hi_mpi_venc_close_fd....924
11.18.18.18 hi_mpi_venc_set_jpeg_param....925
11.18.18.19 hi_mpi_venc_get_jpeg_param....926
11.18.18.20 hi_mpi_venc_set_chn_param....926
11.18.18.21 hi_mpi_venc_get_chn_param....927
11.18.18.22 hi_mpi_venc_set_scene_mode....928
11.18.18.23 hi_mpi_venc_get_scene_mode....929
11.18.18.24 hi_mpi_venc_set_rc_param....929
11.18.18.25 hi_mpi_venc_get_rc_param....930

11.18.18.26 hi_mpi_venc_set_jpeg_huffman_param....931
11.18.18.27 hi_mpi_venc_get_jpeg_huffman_param....932
11.18.18.28 hi_mpi_venc_compact_jpeg_tables....932
11.18.18.29 hi_mpi_venc_send_jpege_frame....933
11.18.18.30 hi_mpi_venc_get_jpege_predicted_size....934
11.18.18.31 hi_mpi_venc_set_roi_attr....935
11.18.18.32 hi_mpi_venc_get_roi_attr....937
11.18.18.33 hi_mpi_venc_set_ref_param....938
11.18.18.34 hi_mpi_venc_get_ref_param....941
11.18.18.35 hi_mpi_venc_set_h264_vui....942
11.18.18.36 hi_mpi_venc_get_h264_vui....943
11.18.18.37 hi_mpi_venc_set_h265_vui....944
11.18.18.38 hi_mpi_venc_get_h265_vui....945
11.18.18.39 hi_mpi_venc_set_cu_pred....946
11.18.18.40 hi_mpi_venc_get_cu_pred....947
11.18.18.41 hi_mpi_venc_set_intra_refresh....948
11.18.18.42 hi_mpi_venc_get_intra_refresh....949
11.18.18.43 hi_mpi_venc_set_chn_attr....950
11.18.18.44 hi_mpi_venc_get_chn_attr....951
11.18.18.45 hi_mpi_venc_set_slice_split....952
11.18.18.46 hi_mpi_venc_get_slice_split....952
11.18.19 PNGD 图片解码功能....953
11.18.19.1 功能及约束说明....953
11.18.19.2 性能指标说明....956
11.18.19.3 hi_mpi_pngd_create_chn....956
11.18.19.4 hi_mpi_pngd_destroy_chn....957
11.18.19.5 hi_mpi_pngd_send_stream....958
11.18.19.6 hi_mpi_pngd_get_image_data....959
11.18.19.7 hi_mpi_png_get_image_info....961
11.18.20 枚举值及数据结构....961
11.18.20.1 公共....961
11.18.20.1.1 基本数据类型说明....961
11.18.20.1.2 hi_pixel_format....962
11.18.20.1.3 hi_payload_type....966
11.18.20.1.4 hi_video_field....968
11.18.20.1.5 hi_dynamic_range....968
11.18.20.1.6 hi_color_gamut....969
11.18.20.1.7 hi_video_supplement....970
11.18.20.1.8 hi_video_frame....971
11.18.20.1.9 hi_video_frame_info....974
11.18.20.1.10 hi_vb_src....974
11.18.20.1.11 hi_mod_id....975

11.18.20.1.12 hi_data_bit_width....976
11.18.20.1.13 hi_video_format....976
11.18.20.1.14 hi_img_info....977
11.18.20.1.15 hi_pic_buf_attr....977
11.18.20.1.16 hi_jpeg_raw_format....978
11.18.20.1.17 hi_dvpp_epoll_ctl_op....978
11.18.20.1.18 hi_dvpp_epoll_event_type....979
11.18.20.1.19 hi_dvpp_epoll_event....979
11.18.20.1.20 hi_csc_matrix....980
11.18.20.1.21 hi_coefficient....982
11.18.20.1.22 hi_csc_coefficient....983
11.18.20.1.23 hi_video_size....983
11.18.20.1.24 hi_img_stream....984
11.18.20.1.25 hi_pic_info....984
11.18.20.1.26 hi_png_color_format....985
11.18.20.1.27 hi_rect....985
11.18.20.1.28 hi_op_mode....986
11.18.20.1.29 hi_point....986
11.18.20.1.30 hi_crop_info....986
11.18.20.1.31 hi_frame_rate_ctrl....987
11.18.20.1.32 hi_compress_mode....987
11.18.20.1.33 hi_mpp_chn....988
11.18.20.1.34 hi_size....989
11.18.20.1.35 hi_module_type....989
11.18.20.1.36 hi_img_base_info....989
11.18.20.1.37 hi_img_align_info....990
11.18.20.1.38 hi_coord....990
11.18.20.1.39 hi_aspect_ratio_type....991
11.18.20.1.40 hi_aspect_ratio....991
11.18.20.1.41 hi_video_display_mode....992
11.18.20.1.42 hi_wdr_mode....992
11.18.20.2 音频相关....993
11.18.20.2.1 基本数据类型....993
11.18.20.2.2 hi_audio_sample_rate....993
11.18.20.2.3 hi_aio_mode....994
11.18.20.2.4 hi_audio_bit_width....995
11.18.20.2.5 hi_audio_snd_mode....996
11.18.20.2.6 hi_aio_i2s_type....996
11.18.20.2.7 hi_aio_attr....997
11.18.20.2.8 hi_audio_frame....998
11.18.20.2.9 hi_audio_stream....999
11.18.20.2.10 hi_aec_frame....999

11.18.20.2.11 hi_audio_frame_info....1000
11.18.20.2.12 hi_acodec_volume_ctrl....1000
11.18.20.2.13 音量调节宏....1001
11.18.20.2.14 hi_aenc_chn_attr....1001
11.18.20.2.15 hi_adec_mode....1002
11.18.20.2.16 hi_adec_chn_attr....1003
11.18.20.2.17 hi_adec_attr_g711....1003
11.18.20.2.18 hi_ai_chn_mode....1004
11.18.20.2.19 hi_ai_chn_attr....1004
11.18.20.2.20 hi_aac_type....1004
11.18.20.2.21 hi_aac_bps....1005
11.18.20.2.22 hi_aac_transport_type....1006
11.18.20.2.23 hi_aenc_attr_aac....1006
11.18.20.2.24 hi_adec_attr_aac....1007
11.18.20.3 ISP 系统控制及 3A 算法注册....1007
11.18.20.3.1 hi_sensor_id....1007
11.18.20.3.2 hi_isp_sns_attr_info....1008
11.18.20.3.3 hi_isp_cmos_sensor_image_mode....1008
11.18.20.3.4 hi_isp_cmos_alg_key....1008
11.18.20.3.5 hi_isp_cmos_noise_calibration....1010
11.18.20.3.6 hi_isp_cmos_sensor_max_resolution....1010
11.18.20.3.7 hi_isp_cmos_sensor_mode....1011
11.18.20.3.8 hi_isp_cmos_wdr_switch_attr....1011
11.18.20.3.9 hi_isp_cmos_default....1012
11.18.20.3.10 hi_isp_cmos_black_level....1014
11.18.20.3.11 hi_isp_sns_type....1015
11.18.20.3.12 hi_isp_sns_commbus....1015
11.18.20.3.13 hi_isp_i2c_data....1016
11.18.20.3.14 hi_isp_ssp_data....1017
11.18.20.3.15 hi_isp_sns_regs_info....1018
11.18.20.3.16 hi_isp_sensor_exp_func....1020
11.18.20.3.17 hi_isp_sensor_register....1021
11.18.20.3.18 hi_isp_bayer_format....1022
11.18.20.3.19 hi_isp_pub_attr....1022
11.18.20.3.20 hi_isp_frame_info....1023
11.18.20.3.21 hi_slave_dev....1025
11.18.20.3.22 hi_isp_slave_sns_sync....1025
11.18.20.3.23 hi_isp_3a_alg_lib....1027
11.18.20.3.24 hi_isp_ae_accuracy_type....1028
11.18.20.3.25 hi_isp_ae_accuracy....1029
11.18.20.3.26 hi_isp_ae_sensor_default....1030
11.18.20.3.27 hi_isp_ae_sensor_exp_func....1036

11.18.20.3.28 hi_isp_ae_sensor_register....1037
11.18.20.3.29 hi_isp_awb_agc_table....1038
11.18.20.3.30 hi_isp_awb_ccm_tab....1038
11.18.20.3.31 hi_isp_awb_ccm....1039
11.18.20.3.32 hi_isp_awb_sensor_default....1039
11.18.20.3.33 hi_isp_awb_sensor_exp_func....1041
11.18.20.3.34 hi_isp_awb_sensor_register....1041
11.18.20.4 MIPI Rx ioctl 命令字参数....1041
11.18.20.4.1 combo_dev_t....1041
11.18.20.4.2 sns_rst_source_t....1042
11.18.20.4.3 sns_clk_source_t....1042
11.18.20.4.4 lane_divide_mode_t....1042
11.18.20.4.5 input_mode_t....1043
11.18.20.4.6 img_rect_t....1043
11.18.20.4.7 slvs_lane_rate_t....1043
11.18.20.4.8 slvs_err_check_mode_t....1044
11.18.20.4.9 data_type_t....1044
11.18.20.4.10 mipi_data_rate_t....1045
11.18.20.4.11 mipi_wdr_mode_t....1045
11.18.20.4.12 slvs_wdr_mode_t....1046
11.18.20.4.13 lvds_wdr_mode_t....1047
11.18.20.4.14 mipi_dev_attr_t....1047
11.18.20.4.15 lvds_sync_mode_t....1048
11.18.20.4.16 lvds_bit_endian_t....1048
11.18.20.4.17 lvds_vsync_type_t....1049
11.18.20.4.18 lvds_vsync_attr_t....1050
11.18.20.4.19 lvds_fid_type_t....1050
11.18.20.4.20 lvds_fid_attr_t....1051
11.18.20.4.21 lvds_dev_attr_t....1051
11.18.20.4.22 slvs_dev_attr_t....1053
11.18.20.4.23 combo_dev_attr_t....1053
11.18.20.4.24 sns_clk_freq_t....1054
11.18.20.4.25 sns_clk_cfg_t....1055
11.18.20.5 VI 视频输入....1056
11.18.20.5.1 hi_vi_dev....1056
11.18.20.5.2 hi_vi_pipe....1056
11.18.20.5.3 hi_vi_chn....1056
11.18.20.5.4 hi_vi_intf_mode....1056
11.18.20.5.5 hi_data_rate....1057
11.18.20.5.6 hi_vi_scan_mode....1058
11.18.20.5.7 hi_vi_data_seq....1058
11.18.20.5.8 hi_vi_data_type....1059

11.18.20.5.9 hi_vi_dev_attr.....1059
11.18.20.5.10 hi_vi_dev_bind_pipe.....1060
11.18.20.5.11 hi_vi_wdr_attr.....1061
11.18.20.5.12 hi_vi_pipe_attr.....1061
11.18.20.5.13 hi_vi_vpss_mode.....1063
11.18.20.5.14 hi_vi_vpss_mode_type.....1063
11.18.20.5.15 hi_vi_pipe_bypass_mode.....1064
11.18.20.5.16 hi_vi_frame_dump_attr.....1065
11.18.20.5.17 hi_vi_pipe_frame_source.....1065
11.18.20.5.18 hi_vi_chn_attr.....1066
11.18.20.5.19 hi_dis_config.....1067
11.18.20.5.20 hi_dis_attr.....1069
11.18.20.5.21 hi_dis_mode.....1070
11.18.20.5.22 hi_dis_motion_level.....1071
11.18.20.5.23 hi_dis_pdt_type.....1071
11.18.20.5.24 hi_dis_param.....1072
11.18.20.5.25 hi_vi_ldc_attr.....1072
11.18.20.5.26 hi_ldc_v1_attr.....1073
11.18.20.5.27 hi_ldc_v2_attr.....1074
11.18.20.5.28 hi_ldc_version.....1075
11.18.20.5.29 hi_vi_low_delay_info.....1075
11.18.20.6 Region 区域管理.....1076
11.18.20.6.1 HI_RGN_CLUT_NUM.....1076
11.18.20.6.2 HI_RGN_HANDLE_MAX.....1076
11.18.20.6.3 hi_rgn_handle.....1076
11.18.20.6.4 hi_phys_addr_t.....1076
11.18.20.6.5 hi_rgn_type.....1077
11.18.20.6.6 hi_rgn_overlayex_attr.....1077
11.18.20.6.7 hi_rgn_overlayex_chn_attr.....1079
11.18.20.6.8 hi_rgn_cover_chn_attr.....1079
11.18.20.6.9 hi_rgn_mosaic_chn_attr.....1080
11.18.20.6.10 hi_rgn_type_attr.....1081
11.18.20.6.11 hi_rgn_type_chn_attr.....1081
11.18.20.6.12 hi_rgn_attr.....1082
11.18.20.6.13 hi_rgn_chn_attr.....1082
11.18.20.6.14 hi_rgn_canvas_info.....1083
11.18.20.7 VPSS 视频处理.....1083
11.18.20.7.1 基本数据类型说明.....1083
11.18.20.7.2 hi_vpss_crop_info.....1084
11.18.20.7.3 hi_vpss_nr_type.....1085
11.18.20.7.4 hi_vpss_nr_motion_mode.....1085
11.18.20.7.5 hi_vpss_nr_attr.....1086

11.18.20.7.6 hi_vpss_grp_attr.... 1086
11.18.20.7.7 hi_vpss_nr_version.... 1087
11.18.20.7.8 hi_vpss_nrx_v3.... 1088
11.18.20.7.9 hi_v200_vpss_iey.... 1088
11.18.20.7.10 hi_v200_vpss_sfy.... 1089
11.18.20.7.11 hi_v200_vpss_mdy.... 1091
11.18.20.7.12 hi_v200_vpss_tfy.... 1092
11.18.20.7.13 hi_v200_vpss_nr_c.... 1094
11.18.20.7.14 hi_vpss_nrx_param_manual_v3.... 1094
11.18.20.7.15 hi_vpss_nrx_param_auto_v3.... 1095
11.18.20.7.16 hi_vpss_nrx_param_v3.... 1095
11.18.20.7.17 hi_vpss_grp_nrx_param.... 1096
11.18.20.7.18 hi_vpss_grp_param.... 1096
11.18.20.7.19 hi_fisheye_cfg.... 1097
11.18.20.7.20 hi_vpss_chn_mode.... 1097
11.18.20.7.21 hi_vpss_chn_attr.... 1098
11.18.20.7.22 hi_fisheye_correction_attr.... 1100
11.18.20.7.23 hi_fisheye_mount_mode.... 1101
11.18.20.7.24 hi_fisheye_view_mode.... 1101
11.18.20.7.25 hi_fisheye_rgn_attr.... 1102
11.18.20.7.26 hi_fisheye_attr.... 1103
11.18.20.8 VO 视频输出.... 1104
11.18.20.8.1 hi_vo_dev.... 1104
11.18.20.8.2 hi_vo_layer.... 1105
11.18.20.8.3 hi_vo_chn.... 1105
11.18.20.8.4 hi_vo_intf_type.... 1106
11.18.20.8.5 hi_vo_intf_sync.... 1106
11.18.20.8.6 hi_vo_sync_info.... 1109
11.18.20.8.7 hi_vo_partition_mode.... 1111
11.18.20.8.8 hi_vo_pub_attr.... 1111
11.18.20.8.9 hi_vo_video_layer_attr.... 1112
11.18.20.8.10 hi_vo_chn_attr.... 1113
11.18.20.8.11 hi_vo_chn_param.... 1114
11.18.20.9 TDE 图形绘制.... 1114
11.18.20.9.1 hi_tde_color_format.... 1114
11.18.20.9.2 hi_tde_surface.... 1118
11.18.20.9.3 hi_tde_rect.... 1119
11.18.20.9.4 hi_tde_none_src.... 1120
11.18.20.9.5 hi_tde_single_src.... 1120
11.18.20.9.6 hi_tde_double_src.... 1121
11.18.20.9.7 hi_tde_fill_color.... 1121
11.18.20.9.8 hi_tde_blend_cmd.... 1122

11.18.20.9.9 hi_tde_blend_mode....1123
11.18.20.9.10 hi_tde_rop_mode....1124
11.18.20.9.11 hi_tde_clip_mode....1126
11.18.20.9.12 hi_tde_colorkey_mode....1126
11.18.20.9.13 hi_tde_colorkey_component....1127
11.18.20.9.14 hi_tde_colorkey....1128
11.18.20.9.15 hi_tde_blend_opt....1128
11.18.20.9.16 hi_tde_alpha_blending....1129
11.18.20.9.17 hi_tde_out_alpha_from....1130
11.18.20.9.18 hi_tde_csc_opt....1130
11.18.20.9.19 hi_tde_pattern_fill_opt....1131
11.18.20.10 HIFB 叠加图形层管理....1132
11.18.20.10.1 fb_var_screeninfo....1132
11.18.20.10.2 fb_fix_screeninfo....1134
11.18.20.10.3 fb_cmap....1135
11.18.20.10.4 hi_fb_point....1136
11.18.20.10.5 hi_fb_alpha....1136
11.18.20.10.6 fb_bitfield....1137
11.18.20.11 HDMI 外设....1137
11.18.20.11.1 hi_hdmi_id....1138
11.18.20.11.2 hi_hdmi_video_format....1138
11.18.20.11.3 hi_hdmi_deep_color....1141
11.18.20.11.4 hi_hdmi_sample_rate....1142
11.18.20.11.5 hi_hdmi_bit_depth....1143
11.18.20.11.6 hi_hdmi_attr....1144
11.18.20.11.7 hi_hdmi_infoframe_type....1145
11.18.20.11.8 hi_hdmi_color_space....1146
11.18.20.11.9 hi_hdmi_bar_info....1147
11.18.20.11.10 hi_hdmi_scan_info....1147
11.18.20.11.11 hi_hdmi_colorimetry....1148
11.18.20.11.12 hi_hdmi_ex_colorimetry....1148
11.18.20.11.13 hi_pic_aspect_ratio....1149
11.18.20.11.14 hi_hdmi_active_aspect_ratio....1150
11.18.20.11.15 hi_hdmi_pic_scaline....1151
11.18.20.11.16 hi_hdmi_rgb_quant_range....1152
11.18.20.11.17 hi_hdmi_pixel_repetition....1152
11.18.20.11.18 hi_hdmi_content_type....1153
11.18.20.11.19 hi_hdmi_ycc_quant_range....1154
11.18.20.11.20 hi_hdmi_avi_infoframe....1155
11.18.20.11.21 hi_hdmi_audio_chn_cnt....1156
11.18.20.11.22 hi_hdmi_coding_type....1157
11.18.20.11.23 hi_hdmi_audio_sample_size....1159

11.18.20.11.24 hi_hdmi_audio_sample_freq....1159
11.18.20.11.25 hi_hdmi_level_shift_val....1160
11.18.20.11.26 hi_hdmi_lfe_playback_level....1162
11.18.20.11.27 hi_hdmi_audio_infoframe....1162
11.18.20.11.28 hi_hdmi_vendorspec_infoframe....1163
11.18.20.11.29 hi_hdmi_infoframe_unit....1164
11.18.20.11.30 hi_hdmi_infoframe....1164
11.18.20.11.31 hi_hdmi_sink_capability....1164
11.18.20.11.32 hi_hdmi_event_type....1165
11.18.20.11.33 hi_hdmi_callback....1166
11.18.20.11.34 hi_hdmi_callback_func....1166
11.18.20.12 VPC 图像处理....1166
11.18.20.12.1 hi_vpc_chn....1166
11.18.20.12.2 hi_vpc_pic_info....1167
11.18.20.12.3 hi_vpc_crop_region....1167
11.18.20.12.4 hi_vpc_resize_info....1168
11.18.20.12.5 hi_vpc_crop_region_info....1168
11.18.20.12.6 hi_vpc_crop_resize_region....1169
11.18.20.12.7 hi_vpc_crop_resize_paste_region....1169
11.18.20.12.8 hi_vpc_bord_type....1170
11.18.20.12.9 hi_vpc_make_border_info....1171
11.18.20.12.10 hi_vpc_histogram_config....1171
11.18.20.12.11 hi_vpc_lut_remap....1172
11.18.20.12.12 hi_vpc_scalar....1172
11.18.20.12.13 hi_vpc_chn_attr....1173
11.18.20.12.14 hi_vpc_crop_resize_border_region....1174
11.18.20.12.15 hi_csc_conf....1175
11.18.20.12.16 hi_roundview_stitching_param....1175
11.18.20.12.17 hi_roundview_stitching_pic_param....1176
11.18.20.12.18 hi_stiching_ipm_param....1176
11.18.20.12.19 hi_stiching_ipm_table....1177
11.18.20.12.20 hi_stitching_gain_param....1178
11.18.20.12.21 hi_stitching_histogram_type....1179
11.18.20.12.22 hi_stitching_gain_type....1179
11.18.20.12.23 hi_stitching_histogram_param....1180
11.18.20.12.24 hi_vpc_crop_resize_resize_paste_region....1180
11.18.20.12.25 hi_vpc_workspace_param....1181
11.18.20.12.26 hi_workspace_func....1182
11.18.20.12.27 hi_remap_lut....1182
11.18.20.12.28 hi_point_pair_info....1183
11.18.20.12.29 hi_float_point....1183
11.18.20.12.30 hi_map_param....1184

11.18.20.12.31 hi_warp_transform_param....1185
11.18.20.12.32 hi_transform_config....1185
11.18.20.12.33 hi_morph_shapes....1185
11.18.20.12.34 hi_rotation....1186
11.18.20.12.35 hi_blk_size....1187
11.18.20.12.36 hi_median_blur_param....1187
11.18.20.12.37 hi_median_blur_config....1188
11.18.20.12.38 hi_blur_param....1188
11.18.20.12.39 hi_blur_config....1189
11.18.20.12.40 hi_gaussian_blur_param....1189
11.18.20.12.41 hi_gaussian_blur_config....1190
11.18.20.12.42 hi_filter_2d_param....1190
11.18.20.12.43 hi_filter_2d_config....1191
11.18.20.12.44 hi_mosaic_param....1192
11.18.20.12.45 hi_rotate_param....1192
11.18.20.12.46 hi_mosaic....1193
11.18.20.12.47 hi_cover_type....1193
11.18.20.12.48 hi_cover_param....1194
11.18.20.12.49 hi_cover....1194
11.18.20.12.50 hi_quad_cover....1195
11.18.20.12.51 hi_line_param....1195
11.18.20.12.52 hi_line....1196
11.18.20.12.53 hi_osd_param....1196
11.18.20.12.54 hi_osd....1197
11.18.20.12.55 hi_osd_inverted_color....1198
11.18.20.12.56 hi_transform_matrix....1199
11.18.20.12.57 hi_matrix_type....1199
11.18.20.12.58 hi_flip_param....1200
11.18.20.12.59 hi_flip_mode....1200
11.18.20.12.60 hi_opt_attr....1201
11.18.20.12.61 hi_opt_attr_type....1201
11.18.20.13 VDEC 视频/JPEGD 图像解码....1201
11.18.20.13.1 hi_vdec_chn....1202
11.18.20.13.2 hi_vdec_send_mode....1202
11.18.20.13.3 hi_vdec_frame_type....1203
11.18.20.13.4 hi_quick_mark_mode....1203
11.18.20.13.5 hi_vdec_pic_info....1204
11.18.20.13.6 hi_vdec_video_attr....1206
11.18.20.13.7 hi_vdec_chn_attr....1207
11.18.20.13.8 hi_h264_protocol_param....1209
11.18.20.13.9 hi_h265_protocol_param....1209
11.18.20.13.10 hi_vdec_protocol_param....1210

11.18.20.13.11 hi_video_dec_mode.....1211
11.18.20.13.12 hi_video_out_order.....1211
11.18.20.13.13 hi_vdec_video_param.....1212
11.18.20.13.14 hi_vdec_pic_param.....1213
11.18.20.13.15 hi_vdec_chn_param.....1213
11.18.20.13.16 hi_vdec_dec_err.....1214
11.18.20.13.17 hi_vdec_chn_status.....1214
11.18.20.13.18 hi_vdec_stream.....1215
11.18.20.13.19 hi_vdec_supplement_info.....1216
11.18.20.13.20 hi_vdec_video_supplement_info.....1217
11.18.20.13.21 hi_jpegd_precision_mode.....1217
11.18.20.14 VENC 视频/JPEGE 图像编码.....1218
11.18.20.14.1 hi_venc_chn.....1218
11.18.20.14.2 hi_venc_stream.....1218
11.18.20.14.3 hi_venc_pack.....1219
11.18.20.14.4 hi_venc_chn_attr.....1220
11.18.20.14.5 hi_venc_attr.....1220
11.18.20.14.6 hi_venc_rc_attr.....1224
11.18.20.14.7 hi_venc_rc_mode.....1226
11.18.20.14.8 hi_venc_h264_cbr.....1228
11.18.20.14.9 hi_venc_h264_vbr.....1230
11.18.20.14.10 hi_venc_h264_avbr.....1231
11.18.20.14.11 hi_venc_h264_qvbr.....1232
11.18.20.14.12 hi_venc_h264_cvbr.....1234
11.18.20.14.13 hi_venc_h264_fixqp.....1235
11.18.20.14.14 hi_venc_h264_qpmap.....1236
11.18.20.14.15 hi_venc_mjpeg_cbr.....1237
11.18.20.14.16 hi_venc_mjpeg_vbr.....1237
11.18.20.14.17 hi_venc_mjpeg_fixqp.....1237
11.18.20.14.18 hi_venc_h265_cbr.....1237
11.18.20.14.19 hi_venc_h265_vbr.....1239
11.18.20.14.20 hi_venc_h265_avbr.....1241
11.18.20.14.21 hi_venc_h265_qvbr.....1242
11.18.20.14.22 hi_venc_h265_cvbr.....1244
11.18.20.14.23 hi_venc_h265_fixqp.....1245
11.18.20.14.24 hi_venc_h265_qpmap.....1246
11.18.20.14.25 hi_venc_gop_attr.....1247
11.18.20.14.26 hi_venc_gop_mode.....1247
11.18.20.14.27 hi_venc_gop_normal_p.....1248
11.18.20.14.28 hi_venc_gop_dual_p.....1249
11.18.20.14.29 hi_venc_gop_smart_p.....1249
11.18.20.14.30 hi_venc_gop_adv_smart_p.....1250

11.18.20.14.31 hi_venc_gop_bipred_b.... 1250
11.18.20.14.32 hi_venc_start_param.... 1250
11.18.20.14.33 hi_venc_chn_status.... 1251
11.18.20.14.34 hi_venc_h264_stream_info.... 1252
11.18.20.14.35 hi_venc_ref_type.... 1253
11.18.20.14.36 hi_venc_jpeg_stream_info.... 1253
11.18.20.14.37 hi_venc_h265_stream_info.... 1253
11.18.20.14.38 hi_venc_prores_stream_info.... 1254
11.18.20.14.39 hi_venc_mod_param.... 1255
11.18.20.14.40 hi_venc_mod_type.... 1255
11.18.20.14.41 hi_venc_venc_mod_param.... 1256
11.18.20.14.42 hi_venc_h264_mod_param.... 1257
11.18.20.14.43 hi_venc_h265_mod_param.... 1258
11.18.20.14.44 hi_venc_jpeg_mod_param.... 1259
11.18.20.14.45 hi_venc_jpeg_param.... 1259
11.18.20.14.46 hi_venc_h264_adv_stream_info.... 1260
11.18.20.14.47 hi_venc_h265_adv_stream_info.... 1261
11.18.20.14.48 hi_venc_data_type.... 1263
11.18.20.14.49 hi_venc_h264_nalu_type.... 1263
11.18.20.14.50 hi_venc_jpege_pack_type.... 1263
11.18.20.14.51 hi_venc_h265_nalu_type.... 1264
11.18.20.14.52 hi_venc_prores_pack_type.... 1264
11.18.20.14.53 hi_venc_chn_param.... 1264
11.18.20.14.54 hi_venc_stream_info.... 1265
11.18.20.14.55 hi_venc_pack_info.... 1266
11.18.20.14.56 hi_frame_rate_ctrl.... 1267
11.18.20.14.57 hi_venc_rc_qpmap_mode.... 1267
11.18.20.14.58 hi_venc_sse_info.... 1268
11.18.20.14.59 hi_venc_h264_attr.... 1268
11.18.20.14.60 hi_venc_h265_attr.... 1269
11.18.20.14.61 hi_venc_jpeg_attr.... 1270
11.18.20.14.62 hi_venc_prores_attr.... 1270
11.18.20.14.63 hi_venc_scene_mode.... 1271
11.18.20.14.64 hi_venc_rc_param.... 1271
11.18.20.14.65 hi_venc_h264_cbr_param.... 1274
11.18.20.14.66 hi_venc_h264_vbr_param.... 1275
11.18.20.14.67 hi_venc_h264_avbr_param.... 1276
11.18.20.14.68 hi_venc_h264_qvbr_param.... 1278
11.18.20.14.69 hi_venc_h264_cvbr_param.... 1280
11.18.20.14.70 hi_venc_h265_vbr_param.... 1282
11.18.20.14.71 hi_venc_h265_avbr_param.... 1283
11.18.20.14.72 hi_venc_h265_qvbr_param.... 1285

11.18.20.14.73 hi_venc_h265_cvbr_param....1287
11.18.20.14.74 hi_venc_h265_cbr_param....1289
11.18.20.14.75 hi_venc_scene_chg_detect....1290
11.18.20.14.76 hi_venc_huffman_dc_table....1291
11.18.20.14.77 hi_venc_huffman_ac_table....1291
11.18.20.14.78 hi_venc_jpeg_huffman_param....1291
11.18.20.14.79 hi_venc_mpf_cfg....1292
11.18.20.14.80 hi_venc_mjpeg_cbr_param....1292
11.18.20.14.81 hi_venc_mjpeg_vbr_param....1292
11.18.20.14.82 hi_venc_h264_ref_slice_type....1293
11.18.20.14.83 hi_venc_roi_attr....1293
11.18.20.14.84 hi_venc_ref_param....1294
11.18.20.14.85 hi_venc_h264_vui....1295
11.18.20.14.86 hi_venc_h265_vui....1295
11.18.20.14.87 hi_venc_vui_aspect_ratio....1296
11.18.20.14.88 hi_venc_vui_h264_time_info....1296
11.18.20.14.89 hi_venc_vui_h265_time_info....1297
11.18.20.14.90 hi_venc_vui_video_signal....1298
11.18.20.14.91 hi_venc_vui_bitstream_restric....1299
11.18.20.14.92 hi_venc_cu_prediction....1299
11.18.20.14.93 hi_venc_intra_refresh....1301
11.18.20.14.94 hi_venc_intra_refresh_mode....1303
11.18.20.14.95 hi_venc_slice_split....1303
11.18.20.15 PNGD 图像解码....1304
11.18.20.15.1 hi_pngd_chn....1304
11.18.20.15.2 hi_pngd_chn_attr....1304
11.18.21 返回码列表....1304
11.18.21.1 公共返回码....1304
11.18.21.2 音频相关返回码....1305
11.18.21.3 ISP 系统控制返回码....1309
11.18.21.4 VI 视频输入返回码....1309
11.18.21.5 区域管理返回码....1310
11.18.21.6 VPSS 视频处理子系统返回码....1311
11.18.21.7 VO 视频输出返回码....1311
11.18.21.8 TDE 图形绘制返回码....1312
11.18.21.9 HDMI 外设返回码....1313
11.18.21.10 VPC 图像处理返回码....1314
11.18.21.11 VDEC 视频解码/JPEGD 图片解码返回码....1315
11.18.21.12 VENC 视频编码/JPEGE 图片编码返回码....1316
11.18.21.13 PNGD 图像解码返回码....1318
11.19 异常处理....1319
11.19.1 aclrtSetExceptionInfoCallback....1319

11.19.2 aclrtGetTaskIdFromExceptionInfo.... 1320
11.19.3 aclrtGetStreamIdFromExceptionInfo.... 1321
11.19.4 aclrtGetThreadIdFromExceptionInfo.... 1321
11.19.5 aclrtGetDeviceIdFromExceptionInfo.... 1322
11.19.6 aclrtGetErrorCodeFromExceptionInfo.... 1323
11.20 日志管理.... 1323
11.20.1 aclAppLog.... 1323
11.21 特征向量检索.... 1324
11.21.1 总体说明.... 1324
11.21.2 aclfvInit.... 1325
11.21.3 aclfvRelease.... 1326
11.21.4 aclfvRepoAdd.... 1327
11.21.5 aclfvRepoDel.... 1328
11.21.6 aclfvDel.... 1328
11.21.7 aclfvModify.... 1329
11.21.8 aclfvSearch.... 1330
11.22 Profiling 数据采集.... 1331
11.22.1 Profiling AscendCL API（通过 Profiling AscendCL API 采集并落盘性能数据）.... 1331
11.22.1.1 功能介绍.... 1331
11.22.1.2 aclprofInit.... 1333
11.22.1.3 aclprofSetConfig.... 1333
11.22.1.4 aclprofStart.... 1335
11.22.1.5 aclprofStop.... 1335
11.22.1.6 aclprofFinalize.... 1336
11.22.2 Profiling AscendCL API for Extension（Profiling AscendCL API 扩展接口）.... 1336
11.22.2.1 功能介绍.... 1337
11.22.2.2 aclprofCreateStamp.... 1337
11.22.2.3 aclprofSetStampTraceMessage.... 1338
11.22.2.4 aclprofMark.... 1338
11.22.2.5 aclprofPush.... 1339
11.22.2.6 aclprofPop.... 1340
11.22.2.7 aclprofRangeStart.... 1340
11.22.2.8 aclprofRangeStop.... 1341
11.22.2.9 aclprofDestroyStamp.... 1342
11.22.3 Profiling AscendCL API for Subscription（订阅算子信息的 Profiling AscendCL API）.... 1343
11.22.3.1 功能介绍.... 1343
11.22.3.2 aclprofModelSubscribe.... 1344
11.22.3.3 aclprofModelUnSubscribe.... 1344
11.22.3.4 aclprofGetOpDescSize.... 1345
11.22.3.5 aclprofGetOpNum.... 1346
11.22.3.6 aclprofGetOpTypeLen.... 1346
11.22.3.7 aclprofGetOpType.... 1347

11.22.3.8 aclprofGetOpNameLen....1348
11.22.3.9 aclprofGetOpName....1349
11.22.3.10 aclprofGetOpStart....1349
11.22.3.11 aclprofGetOpEnd....1350
11.22.3.12 aclprofGetOpDuration....1351
11.22.3.13 aclprofGetModelId....1351
11.22.4 Profiling AscendCL API（PyTorch 场景标记迭代时间）....1352
11.22.4.1 aclprofGetStepTimestamp....1352
11.23 Tensor 数据传输....1353
11.23.1 acltdtCreateChannel....1353
11.23.2 acltdtSendTensor....1353
11.23.3 acltdtReceiveTensor....1354
11.23.4 acltdtStopChannel....1355
11.23.5 acltdtDestroyChannel....1356
11.23.6 acltdtCreateChannelWithCapacity....1357
11.23.7 acltdtQueryChannelSize....1357
11.24 数据类型转换及获取数据大小....1358
11.24.1 aclDataTypeSize....1358
11.24.2 aclFloat16ToFloat....1359
11.24.3 aclFloatToFloat16....1359
11.25 共享队列管理....1359
11.25.1 功能及约束说明....1360
11.25.2 acltdtCreateQueue....1360
11.25.3 acltdtDestroyQueue....1360
11.25.4 acltdtEnqueueData....1361
11.25.5 acltdtDequeueData....1362
11.25.6 acltdtEnqueue....1363
11.25.7 acltdtDequeue....1364
11.25.8 acltdtBindQueueRoutes....1365
11.25.9 acltdtUnbindQueueRoutes....1366
11.25.10 acltdtQueryQueueRoutes....1367
11.25.11 acltdtGrantQueue....1368
11.25.12 acltdtAttachQueue....1369
11.26 共享 Buffer 管理....1370
11.26.1 acltdtAllocBuf....1370
11.26.2 acltdtFreeBuf....1371
11.26.3 acltdtGetBufData....1372
11.26.4 acltdtSetBufUserData....1373
11.26.5 acltdtGetBufUserData....1373
11.26.6 acltdtSetBufDataLen....1374
11.26.7 acltdtGetBufDataLen....1375
11.26.8 acltdtCopyBufRef....1376

11.26.9 acltdtAppendBufChain.....1376
11.26.10 acltdtGetBufChainNum.....1377
11.26.11 acltdtGetBufFromChain.....1377
11.27 数据类型及其操作接口.....1378
11.27.1 aclError.....1378
11.27.2 aclDataType.....1398
11.27.3 aclFloat16.....1398
11.27.4 aclFormat.....1398
11.27.5 aclrtMemMallocPolicy.....1399
11.27.6 acldvppPixelFormat.....1400
11.27.7 acldvppStreamFormat.....1401
11.27.8 acldvppJpegFormat.....1401
11.27.9 acldvppCscMatrix.....1402
11.27.10 aclrtContext.....1403
11.27.11 aclrtStream.....1403
11.27.12 aclrtEvent.....1403
11.27.13 aclrtEventStatus.....1404
11.27.14 aclrtRunMode.....1404
11.27.15 aclTransType.....1404
11.27.16 aclComputeType.....1404
11.27.17 aclfvSearchType.....1404
11.27.18 aclAippInputFormat.....1404
11.27.19 aclAippInfo.....1405
11.27.20 aclAippDims.....1405
11.27.21 aclmdlIODims.....1406
11.27.22 aclrtGroupAttr.....1406
11.27.23 aclprofAicoreMetrics.....1406
11.27.24 acldvppBorderType.....1406
11.27.25 aclmdlInputAippType.....1407
11.27.26 acltdtTensorType.....1407
11.27.27 aclCompileOpt.....1407
11.27.28 acldvppChannelDescParamType.....1415
11.27.29 aclvdecChannelDescParamType.....1416
11.27.30 aclvencChannelDescParamType.....1418
11.27.31 aclmdlConfigAttr.....1420
11.27.32 aclMemType.....1422
11.27.33 aclOpCompileFlag.....1422
11.27.34 aclrtEventWaitStatus.....1422
11.27.35 aclprofStepTag.....1422
11.27.36 acltdtBuf.....1422
11.27.37 acltdtQueueAttrType.....1423
11.27.38 acltdtQueueRouteParamType.....1423

11.27.39 acltdtQueueRouteQueryMode....1423
11.27.40 acltdtQueueRouteQueryInfoParamType....1423
11.27.41 acldvppChannelMode....1423
11.27.42 aclopEngineType....1423
11.27.43 aclopCompileType....1423
11.27.44 aclrtEventRecordedStatus....1423
11.27.45 aclrtFloatOverflowMode....1424
11.27.46 acldvppJpegdPrecisionMode....1424
11.27.47 aclmdlExecConfigAttr....1424
11.27.48 aclrtStreamConfigAttr....1425
11.27.49 aclSysParamOpt....1426
11.27.50 aclrtUtilizationInfo....1427
11.27.51 aclrtStreamStatus....1427
11.27.52 aclrtMemHandleType....1427
11.27.53 aclrtMemAllocationType....1427
11.27.54 aclrtMemAttr....1427
11.27.55 aclrtMemLocationType....1427
11.27.56 aclrtMemLocation....1427
11.27.57 aclrtPhysicalMemProp....1427
11.27.58 aclrtDrvMemHandle....1428
11.27.59 aclrtDeviceStatus....1428
11.27.60 aclmdlAIPP....1428
11.27.60.1 aclmdlCreateAIPP....1428
11.27.60.2 aclmdlSetAIPPCscParams....1429
11.27.60.3 aclmdlSetAIPPInputFormat....1431
11.27.60.4 aclmdlSetAIPPRbuvSwapSwitch....1431
11.27.60.5 aclmdlSetAIPPAxSwapSwitch....1432
11.27.60.6 aclmdlSetAIPPSrcImageSize....1433
11.27.60.7 aclmdlSetAIPPScfParams....1433
11.27.60.8 aclmdlSetAIPPCropParams....1435
11.27.60.9 aclmdlSetAIPPPaddingParams....1436
11.27.60.10 aclmdlSetAIPPDtcPixelMean....1437
11.27.60.11 aclmdlSetAIPPDtcPixelMin....1438
11.27.60.12 aclmdlSetAIPPPixelVarReci....1439
11.27.60.13 aclmdlDestroyAIPP....1440
11.27.61 aclopHandle....1441
11.27.61.1 aclopCreateHandle....1441
11.27.61.2 aclopDestroyHandle....1442
11.27.62 aclDataBuffer....1442
11.27.62.1 aclCreateDataBuffer....1442
11.27.62.2 aclDestroyDataBuffer....1443
11.27.62.3 aclGetDataBufferAddr....1443

11.27.62.4 aclGetDataBufferSize....1444
11.27.62.5 aclGetDataBufferSizeV2....1444
11.27.62.6 aclUpdateDataBuffer....1445
11.27.63 aclmdlDataset....1446
11.27.63.1 aclmdlCreateDataset....1446
11.27.63.2 aclmdlDestroyDataset....1446
11.27.63.3 aclmdlAddDatasetBuffer....1446
11.27.63.4 aclmdlGetDatasetNumBuffers....1447
11.27.63.5 aclmdlGetDatasetBuffer....1448
11.27.63.6 aclmdlSetDatasetTensorDesc....1448
11.27.63.7 aclmdlGetDatasetTensorDesc....1449
11.27.64 aclmdlDesc....1450
11.27.64.1 aclmdlCreateDesc....1450
11.27.64.2 aclmdlDestroyDesc....1451
11.27.64.3 aclmdlGetDesc....1451
11.27.64.4 aclmdlGetNumInputs....1452
11.27.64.5 aclmdlGetNumOutputs....1452
11.27.64.6 aclmdlGetInputSizeByIndex....1453
11.27.64.7 aclmdlGetOutputSizeByIndex....1453
11.27.64.8 aclmdlGetInputDims....1454
11.27.64.9 aclmdlGetInputDimsV2....1456
11.27.64.10 aclmdlGetOutputDims....1458
11.27.64.11 aclmdlGetInputNameByIndex....1458
11.27.64.12 aclmdlGetOutputNameByIndex....1459
11.27.64.13 aclmdlGetInputFormat....1460
11.27.64.14 aclmdlGetOutputFormat....1461
11.27.64.15 aclmdlGetInputDataType....1461
11.27.64.16 aclmdlGetOutputDataType....1462
11.27.64.17 aclmdlGetInputIndexByName....1462
11.27.64.18 aclmdlGetOutputIndexByName....1463
11.27.64.19 aclmdlGetDynamicBatch....1463
11.27.64.20 aclmdlGetDynamicHW....1464
11.27.64.21 aclmdlGetCurOutputDims....1465
11.27.64.22 aclmdlGetInputDynamicGearCount....1466
11.27.64.23 aclmdlGetInputDynamicDims....1466
11.27.64.24 aclmdlGetTensorRealName....1468
11.27.64.25 aclmdlGetOpAttr....1468
11.27.64.26 aclmdlGetDescFromFile....1469
11.27.64.27 aclmdlGetDescFromMem....1470
11.27.65 aclTensorDesc....1471
11.27.65.1 aclCreateTensorDesc....1471
11.27.65.2 aclDestroyTensorDesc....1472

11.27.65.3 aclGetTensorDescType....1472
11.27.65.4 aclGetTensorDescFormat....1473
11.27.65.5 aclGetTensorDescSize....1473
11.27.65.6 aclGetTensorDescElementCount....1474
11.27.65.7 aclGetTensorDescNumDims....1474
11.27.65.8 aclGetTensorDescDim....1475
11.27.65.9 aclGetTensorDescDimV2....1476
11.27.65.10 aclGetTensorDescDimRange....1476
11.27.65.11 aclSetTensorDescName....1477
11.27.65.12 aclGetTensorDescName....1478
11.27.65.13 aclTransTensorDescFormat....1478
11.27.65.14 aclSetTensorStorageShape....1479
11.27.65.15 aclSetTensorStorageFormat....1480
11.27.65.16 aclSetTensorOriginShape....1480
11.27.65.17 aclSetTensorOriginFormat....1481
11.27.65.18 aclSetTensorShape....1482
11.27.65.19 aclSetTensorFormat....1482
11.27.65.20 aclSetTensorShapeRange....1483
11.27.65.21 aclSetTensorDynamicInput....1484
11.27.65.22 aclGetTensorDescByIndex....1485
11.27.65.23 aclGetTensorDescAddress....1485
11.27.65.24 aclSetTensorConst....1486
11.27.65.25 aclSetTensorPlaceMent....1487
11.27.65.26 aclSetTensorValueRange....1487
11.27.66 aclopAttr....1488
11.27.66.1 aclopCreateAttr....1488
11.27.66.2 aclopDestroyAttr....1488
11.27.66.3 aclopSetAttrBool....1489
11.27.66.4 aclopSetAttrInt....1489
11.27.66.5 aclopSetAttrFloat....1490
11.27.66.6 aclopSetAttrString....1490
11.27.66.7 aclopSetAttrListBool....1491
11.27.66.8 aclopSetAttrListInt....1492
11.27.66.9 aclopSetAttrListFloat....1492
11.27.66.10 aclopSetAttrListString....1493
11.27.66.11 aclopSetAttrListListInt....1493
11.27.66.12 aclopSetAttrDataType....1494
11.27.66.13 aclopSetAttrListDataType....1495
11.27.67 acldvppChannelDesc....1495
11.27.67.1 acldvppCreateChannelDesc....1495
11.27.67.2 acldvppDestroyChannelDesc....1496
11.27.67.3 acldvppGetChannelDescChannelId....1496

11.27.67.4 acldvppSetChannelDescMode.... 1497
11.27.67.5 acldvppSetChannelDescParam.... 1498
11.27.67.6 acldvppGetChannelDescParam.... 1498
11.27.68 acldvppPicDesc.... 1499
11.27.68.1 acldvppCreatePicDesc.... 1499
11.27.68.2 acldvppSetPicDesc 系列接口.... 1500
11.27.68.3 acldvppGetPicDesc 系列接口.... 1501
11.27.68.4 acldvppDestroyPicDesc.... 1502
11.27.69 acldvppRoiConfig.... 1502
11.27.69.1 acldvppCreateRoiConfig.... 1503
11.27.69.2 acldvppSetRoiConfig 系列接口.... 1504
11.27.69.3 acldvppDestroyRoiConfig.... 1505
11.27.70 acldvppResizeConfig.... 1505
11.27.70.1 acldvppCreateResizeConfig.... 1505
11.27.70.2 acldvppSetResizeConfigInterpolation.... 1506
11.27.70.3 acldvppGetResizeConfigInterpolation.... 1507
11.27.70.4 acldvppDestroyResizeConfig.... 1508
11.27.71 acldvppJpegeConfig.... 1508
11.27.71.1 acldvppCreateJpegeConfig.... 1508
11.27.71.2 acldvppSetJpegeConfigLevel.... 1509
11.27.71.3 acldvppGetJpegeConfigLevel.... 1510
11.27.71.4 acldvppDestroyJpegeConfig.... 1510
11.27.72 aclvdecChannelDesc.... 1511
11.27.72.1 aclvdecCreateChannelDesc.... 1511
11.27.72.2 aclvdecSetChannelDesc 系列接口.... 1511
11.27.72.3 aclvdecGetChannelDesc 系列接口.... 1513
11.27.72.4 aclvdecSetChannelDescParam.... 1515
11.27.72.5 aclvdecGetChannelDescParam.... 1516
11.27.72.6 aclvdecDestroyChannelDesc.... 1517
11.27.73 acldvppStreamDesc.... 1518
11.27.73.1 acldvppCreateStreamDesc.... 1518
11.27.73.2 acldvppSetStreamDesc 系列接口.... 1518
11.27.73.3 acldvppGetStreamDesc 系列接口.... 1519
11.27.73.4 acldvppDestroyStreamDesc.... 1520
11.27.74 aclvdecFrameConfig.... 1521
11.27.74.1 aclvdecCreateFrameConfig.... 1521
11.27.74.2 aclvdecDestroyFrameConfig.... 1521
11.27.75 acldvppBatchPicDesc.... 1521
11.27.75.1 acldvppCreateBatchPicDesc.... 1522
11.27.75.2 acldvppGetPicDesc.... 1522
11.27.75.3 acldvppDestroyBatchPicDesc.... 1523
11.27.76 aclvencChannelDesc.... 1523

11.27.76.1 aclvencCreateChannelDesc....1523
11.27.76.2 aclvencSetChannelDesc 系列接口....1524
11.27.76.3 aclvencGetChannelDesc 系列接口....1526
11.27.76.4 aclvencSetChannelDescParam....1528
11.27.76.5 aclvencGetChannelDescParam....1528
11.27.76.6 aclvencDestroyChannelDesc....1529
11.27.77 aclvencFrameConfig....1530
11.27.77.1 aclvencCreateFrameConfig....1530
11.27.77.2 aclvencSetFrameConfig 系列接口....1530
11.27.77.3 aclvencGetFrameConfig 系列接口....1531
11.27.77.4 aclvencDestroyFrameConfig....1532
11.27.78 acldvppLutMap....1532
11.27.78.1 acldvppCreateLutMap....1532
11.27.78.2 acldvppDestroyLutMap....1533
11.27.78.3 acldvppGetLutMapDims....1533
11.27.78.4 acldvppGetLutMapData....1534
11.27.79 acldvppBorderConfig....1535
11.27.79.1 acldvppCreateBorderConfig....1535
11.27.79.2 acldvppSetBorderConfig 系列接口....1535
11.27.79.3 acldvppGetBorderConfig 系列接口....1536
11.27.79.4 acldvppDestroyBorderConfig....1537
11.27.80 acldvppHist....1538
11.27.80.1 acldvppCreateHist....1538
11.27.80.2 acldvppDestroyHist....1538
11.27.80.3 acldvppGetHistDims....1539
11.27.80.4 acldvppGetHistData....1539
11.27.80.5 acldvppGetHistRetCode....1540
11.27.80.6 acldvppClearHist....1541
11.27.81 aclfvRepoRange....1541
11.27.81.1 aclfvCreateRepoRange....1541
11.27.81.2 aclfvDestroyRepoRange....1542
11.27.82 aclfvFeatureInfo....1542
11.27.82.1 aclfvCreateFeatureInfo....1543
11.27.82.2 aclfvDestroyFeatureInfo....1544
11.27.83 aclfvQueryTable....1544
11.27.83.1 aclfvCreateQueryTable....1544
11.27.83.2 aclfvDestroyQueryTable....1545
11.27.84 aclfvSearchInput....1545
11.27.84.1 aclfvCreateSearchInput....1545
11.27.84.2 aclfvDestroySearchInput....1546
11.27.85 aclfvSearchResult....1547
11.27.85.1 aclfvCreateSearchResult....1547

11.27.85.2 aclfvDestroySearchResult....1548
11.27.86 aclfvInitPara....1548
11.27.86.1 aclfvCreateInitPara....1548
11.27.86.2 aclfvDestroyInitPara....1549
11.27.86.3 aclfvSet1NTopNum....1549
11.27.86.4 aclfvSetNMTopNum....1550
11.27.87 aclprofConfig....1551
11.27.87.1 aclprofCreateConfig....1551
11.27.87.2 aclprofDestroyConfig....1553
11.27.88 aclprofSubscribeConfig....1553
11.27.88.1 aclprofCreateSubscribeConfig....1553
11.27.88.2 aclprofDestroySubscribeConfig....1554
11.27.89 aclprofStepInfo....1555
11.27.89.1 aclprofCreateStepInfo....1555
11.27.89.2 aclprofDestroyStepInfo....1555
11.27.90 acltdtDataItem....1556
11.27.90.1 acltdtCreateDataItem....1556
11.27.90.2 acltdtDestroyDataItem....1557
11.27.90.3 acltdtGetTensorTypeFromItem....1557
11.27.90.4 acltdtGetDataTypeFromItem....1558
11.27.90.5 acltdtGetDataAddrFromItem....1558
11.27.90.6 acltdtGetDataSizeFromItem....1559
11.27.90.7 acltdtGetDimNumFromItem....1559
11.27.90.8 acltdtGetDimsFromItem....1560
11.27.91 acltdtDataset....1561
11.27.91.1 acltdtCreateDataset....1561
11.27.91.2 acltdtDestroyDataset....1561
11.27.91.3 acltdtGetDataItem....1562
11.27.91.4 acltdtAddDataItem....1562
11.27.91.5 acltdtGetDatasetSize....1563
11.27.91.6 acltdtGetDatasetName....1563
11.27.92 aclmdlConfigHandle....1564
11.27.92.1 aclmdlCreateConfigHandle....1564
11.27.92.2 aclmdlDestroyConfigHandle....1564
11.27.93 acltdtQueueAttr....1565
11.27.93.1 acltdtCreateQueueAttr....1565
11.27.93.2 acltdtDestroyQueueAttr....1565
11.27.93.3 acltdtSetQueueAttr....1566
11.27.93.4 acltdtGetQueueAttr....1567
11.27.94 acltdtQueueRoute....1567
11.27.94.1 acltdtCreateQueueRoute....1567
11.27.94.2 acltdtDestroyQueueRoute....1568

11.27.94.3 acltdtGetQueueRouteParam....1568
11.27.95 acltdtQueueRouteList....1569
11.27.95.1 acltdtCreateQueueRouteList....1569
11.27.95.2 acltdtDestroyQueueRouteList....1570
11.27.95.3 acltdtAddQueueRoute....1570
11.27.95.4 acltdtGetQueueRoute....1571
11.27.95.5 acltdtGetQueueRouteNum....1571
11.27.96 acltdtQueueRouteQueryInfo....1572
11.27.96.1 acltdtCreateQueueRouteQueryInfo....1572
11.27.96.2 acltdtDestroyQueueRouteQueryInfo....1572
11.27.96.3 acltdtSetQueueRouteQueryInfo....1573
11.27.97 aclGraphDumpOption....1574
11.27.97.1 aclCreateGraphDumpOpt....1574
11.27.97.2 aclDestroyGraphDumpOpt....1574
11.27.98 aclmdlExecConfigHandle....1574
11.27.98.1 aclmdlCreateExecConfigHandle....1575
11.27.98.2 aclmdlDestroyExecConfigHandle....1575
11.27.99 aclrtStreamConfigHandle....1575
11.27.99.1 aclrtCreateStreamConfigHandle....1576
11.27.99.2 aclrtDestroyStreamConfigHandle....1576
11.27.100 aclrtAllocatorDesc....1577
11.27.100.1 aclrtAllocatorCreateDesc....1577
11.27.100.2 aclrtAllocatorDestroyDesc....1577
11.27.100.3 aclrtAllocatorSetObjToDesc....1577
11.27.100.4 aclrtAllocatorSetAllocFuncToDesc....1578
11.27.100.5 aclrtAllocatorSetAllocAdviseFuncToDesc....1579
11.27.100.6 aclrtAllocatorSetFreeFuncToDesc....1579
11.27.100.7 aclrtAllocatorSetGetAddrFromBlockFuncToDesc....1580

12 FAQ....1582

12.1 资源异常问题....1582
12.1.1 内存未释放导致内存泄露....1582
12.1.2 Event 数量超过上限导致 aclrtRecordEvent 接口返回失败....1583
12.1.3 进程异常退出后重新执行任务失败....1583
12.1.4 进程异常，下一次执行任务报错“unbind model stream failed”....1584
12.1.5 进程异常时资源清理的处理建议....1584
12.1.6 用户进程异常退出后重启进程失败....1585
12.1.7 AI 应用进程未退出，导致休眠唤醒失败....1585
12.1.8 算子插件未注册报错....1586
12.1.9 算子原型未注册报错....1587
12.1.10 动态 shape 模型用户输入和模型推导结果不匹配....1587
12.1.11 动态 shape 模型输入大小校验失败....1588
12.1.12 AI Core 算子执行报错....1589

12.1.13 AI CPU 算子执行报错.....1591
12.1.14 调用 Device 失败.....1592
12.1.15 内存申请失败，出现 OOM 情况.....1593
12.1.16 虚拟地址抢占导致 mmap 失败.....1593
12.2 推理问题.....1594
12.2.1 使用 dump 功能未获取 dump 结果.....1594
12.2.2 动态 shape 推理申请内存失败.....1595
12.2.3 异步拷贝调用查询接口报错.....1595
12.2.4 注册算子数超过最大规格.....1596
12.3 DVPP 处理数据问题.....1596
12.3.1 DVPP 驱动引擎异常返回码.....1597
12.3.2 VPC 图片处理.....1600
12.3.2.1 调用错误的内存申请接口，导致内存地址校验出错.....1600
12.3.2.2 使用正确的内存申请接口，但内存大小传值错误.....1601
12.3.2.3 读/写内存地址无效，导致异常中断.....1601
12.3.2.4 VPC 调用失败.....1602
12.3.2.5 VPC 参数校验失败.....1603
12.3.2.6 VPC 创建通道失败.....1604
12.3.2.7 VPC 获取结果失败.....1605
12.3.3 JPEGD 图片解码/VDEC 视频解码.....1605
12.3.3.1 调用错误的内存申请接口，导致内存地址校验出错.....1605
12.3.3.2 内存被提前释放，导致解码数据花屏.....1606
12.3.3.3 JPEGD 图像解码进程超时退出.....1606
12.3.3.4 JPEGD 图片解码失败.....1607
12.3.3.5 VDEC 视频解码丢帧/丢包.....1608
12.3.3.6 VDEC 视频解码花屏.....1610
12.3.3.7 VDEC 视频解码性能问题.....1611
12.3.3.8 VDEC 视频解码失败.....1612
12.3.3.9 VDEC 视频帧解码失败不触发回调函数.....1613
12.3.3.10 VDEC 视频解码异常导致进程卡死，无法退出.....1613
12.3.3.11 retCode 返回值设置错误，导致 VDEC 视频解码异常.....1615
12.3.3.12 VDEC 视频解码无报错，但无解码结果数据、CPU 占用率高.....1615
12.3.3.13 复用输出图片描述类型，VDEC 视频解码报错，提示有不支持的图片格式.....1617
12.3.3.14 异常码流导致 VDEC 视频解码失败.....1618
12.3.3.15 非 annex-B 格式的码流导致 VDEC 视频解码失败.....1619
12.3.4 JPEGE 图片编码/VENC 视频编码.....1620
12.3.4.1 调用错误的内存申请接口，导致内存地址校验出错.....1620
12.3.4.2 内存被提前释放，导致编码数据花屏.....1620
12.3.4.3 使用正确的内存申请接口，但内存大小传值错误.....1621
12.3.4.4 VENC 创建通道失败.....1621
12.3.4.5 VENC 送入待编码图像帧失败.....1622
12.3.4.6 buf_size 参数设置不合理导致视频编码异常.....1622

12.3.4.7 VENC 编码无输出.....1624
12.4 单算子调用问题.....1624
12.4.1 执行单算子产生 coredump 的定位处理.....1624
12.4.2 单算子匹配失败.....1625
12.4.3 调用 NN 类算子接口超时导致失败.....1625
12.5 编译运行问题.....1627
12.5.1 编译运行应用样例报错，提示找不到头文件或库文件.....1627
12.5.2 执行应用程序的权限不足导致 AscendCL 初始化报错.....1629
12.5.3 AscendCL 接口执行无输出无报错.....1630
12.5.4 APP 使用 dvpp 接口编译失败.....1631
13 附录.....1632
13.1 使用约束.....1632
13.2 表达约定.....1634
13.3 Atlas 200/300/500 推理产品->Atlas 推理系列产品的媒体数据处理接口迁移指引.....1635
13.3.1 功能支持度对比列表.....1635
13.3.2 Atlas 200/300/500 推理产品媒体数据处理 V1->Atlas 推理系列产品媒体数据处理 V1 迁移指引.....1637
13.3.2.1 使用场景说明.....1637
13.3.2.2 兼容性说明.....1638
13.3.2.3 优化建议.....1639
13.3.3 Atlas 200/300/500 推理产品媒体数据处理 V1->Atlas 推理系列产品媒体数据处理 V2 迁移指引.....1643
13.3.3.1 使用场景说明.....1643
13.3.3.2 头文件迁移.....1644
13.3.3.3 库文件迁移.....1644
13.3.3.4 VPC 功能迁移.....1644
13.3.3.4.1 媒体数据处理模块初始化.....1644
13.3.3.4.2 创建通道.....1645
13.3.3.4.3 内存申请.....1645
13.3.3.4.4 VPC 抠图缩放贴图功能（一图一框）.....1646
13.3.3.4.5 VPC 批量抠图缩放贴图功能.....1649
13.3.3.4.6 VPC 缩放功能.....1652
13.3.3.4.7 获取处理结果.....1654
13.3.3.4.8 销毁通道.....1655
13.3.3.4.9 媒体数据处理模块去初始化.....1655
13.3.3.5 VDEC 功能迁移.....1655
13.3.3.5.1 媒体数据处理模块初始化.....1655
13.3.3.5.2 创建通道.....1656
13.3.3.5.3 内存申请.....1658
13.3.3.5.4 发送码流.....1659
13.3.3.5.5 接收码流.....1660
13.3.3.5.6 销毁通道.....1661
13.3.3.5.7 媒体数据处理模块去初始化.....1661
13.3.3.6 JPEGD 功能迁移.....1661

13.3.3.6.1 媒体数据处理模块初始化.....1662
13.3.3.6.2 创建通道.....1662
13.3.3.6.3 预估输出内存大小.....1663
13.3.3.6.4 内存申请.....1663
13.3.3.6.5 发送图片.....1664
13.3.3.6.6 接收图片.....1665
13.3.3.6.7 销毁通道.....1665
13.3.3.6.8 媒体数据处理模块去初始化.....1666
13.3.3.7 JPEGE 功能迁移.....1666
13.3.3.7.1 媒体数据处理模块初始化.....1666
13.3.3.7.2 创建通道.....1666
13.3.3.7.3 内存申请.....1667
13.3.3.7.4 发送编码图片.....1668
13.3.3.7.5 获取编码图片.....1669
13.3.3.7.6 销毁通道.....1669
13.3.3.7.7 媒体数据处理模块去初始化.....1670
13.3.3.8 VENC 功能迁移.....1670
13.3.3.8.1 媒体数据处理模块初始化.....1670
13.3.3.8.2 创建通道.....1670
13.3.3.8.3 内存申请.....1671
13.3.3.8.4 发送编码视频帧.....1672
13.3.3.8.5 获取码流.....1673
13.3.3.8.6 销毁通道.....1674
13.3.3.8.7 媒体数据处理模块去初始化.....1674
13.3.3.9 PNGD 功能迁移.....1674
13.3.3.9.1 媒体数据处理模块初始化.....1674
13.3.3.9.2 创建通道.....1675
13.3.3.9.3 预估输出内存.....1675
13.3.3.9.4 内存申请.....1675
13.3.3.9.5 发送图片.....1676
13.3.3.9.6 接收图片.....1677
13.3.3.9.7 销毁通道.....1677
13.3.3.9.8 媒体数据处理模块去初始化.....1678

1 学习向导

本节介绍本文档的内容概要，并给出各知识点的学习顺序建议。

概述

本文用于指导开发人员基于现有模型、使用AscendCL（Ascend Computing Language）提供的C语言API库开发深度神经网络应用，用于实现目标识别、图像分类等功能。

通过本文档您可以学习以下内容：

- 了解AscendCL的功能架构、基本概念以及接口的典型调用流程。
- 使用AscendCL接口进行应用开发的基本流程和实现方法。
- 能够基于本文档中的样例，扩展进行其它应用的开发。

具备C++/C语言程序开发能力、对机器学习或深度学习有一定了解的开发者，可以更好地理解本文档。

文档使用建议

如果您是第一次使用本文档，或者还不清楚以下问题时，建议先**2 快速入门**了解下应用开发的整体过程，然后了解**概述**，再通过**开发基础推理应用**、**图像/视频数据处理**、**单算子调用**等章节的接口调用流程+示例代码来深入学习。

- AscendCL在CANN架构的什么位置?
- AscendCL中的Device、Stream、Context是用来做什么的?
- 使用AscendCL接口开发应用时，包含哪几个基本步骤?

如果您在使用本文档时，已了解使用AscendCL接口开发应用的基本步骤，想进一步学习时，可参照下图的应用开发向导。

准备环境
准备编译及调测环境
安装固件、驱动及CANN软件
CANN软件安装指南
构建模型
构建适配CANN的om模型
使用ATC工具转换模型
ATC工具使用指南
使用Ascend Graph接口构建模型
Ascend Graph构建指南
构建模型过程中，若提示有不支持的算子，需先开发该不支持的算子
算子开发指南
应用开发
调用AscendCL接口开发应用
在昇腾CANN平台上开发应用，实现深度学习推理计算、图形图像预处理、单算子加速计算等
应用开发指南
功能调测
定位应用执行过程中的报错
借助日志中的错误提示快速定位应用程序调试过程中出现的问题
日志参考
参考常见问题案例快速定位并解决应用程序调试过程中的问题
故障处理
性能调优
优化应用的性能
采集并解析应用运行过程中软硬件的性能数据
性能分析工具
借助AOE工具对模型中的算子进行调优
AOE工具使用指南
精度调优
优化应用推理的精度
借助精度比对工具分析、定位精度问题，结合经典案例，说明推理场景下的精度问题定位流程
精度提升专题
精度比对工具使用指南
参考资源
在线体验应用
获取更多参考样例
在线实验体验应用
https://www.hiascend.com/edu/experiment
在线课程学习应用
https://www.hiascend.com/edu/courses
获取更多参考样例
从Ascend samples开源仓获取更多样例

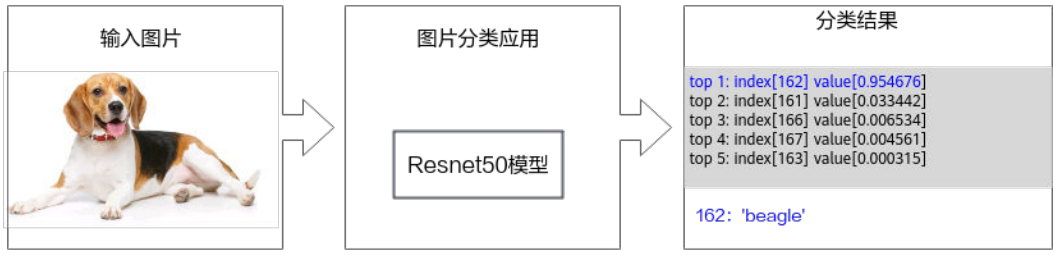
2 快速入门

在本节中，您可以通过一个简单的图片分类应用了解使用AscendCL接口（C语言接口）开发应用的基本过程以及开发过程中涉及的关键概念。

什么是图片分类应用?

“图片分类应用”，从名称上，我们也能直观地看出它的作用：标识图片所属的分类。

图 2-1 图片分类应用



“图片分类应用”是怎么做到这一点的呢？当然得先有一个能做到图片分类的模型，我们可以直接使用一些训练好的开源模型，也可以基于开源模型的源码进行修改、重新训练，还可以自己基于算法、框架构建适合自己的模型。

鉴于当前我们是入门内容，此处我们直接获取已训练好的开源模型，这种方式相对简单。此处我们选择的是ONNX框架的ResNet-50模型。

ResNet-50模型的基本介绍如下：

- 输入数据：BGR格式、224*224分辨率的输入图片
- 输出数据：图片的类别标签及其对应置信度

说明

- 置信度是指图片所属某个类别可能性。
- 类别标签和类别的对应关系与训练模型时使用的数据集有关，需要查阅对应数据集的标签及类别的对应关系。

前提条件

已在环境上部署昇腾AI软件栈。

安装环境，请参见**3.3 准备开发和运行环境**。

了解基本概念

- Host

  Host指与Device相连接的X86服务器、ARM服务器，会利用Device提供的NN（Neural-Network）计算能力，完成业务。

- Device

  Device指安装了昇腾AI处理器的硬件设备，利用PCIe接口与Host侧连接，提供NN计算能力。

- 开发环境、运行环境

  开发环境指编译开发代码的环境，运行环境指运行算子、推理或训练等程序的环境，运行环境上必须带昇腾AI处理器。

  开发环境和运行环境可以合设在同一台服务器上，也可以分设：

  - 合设场景下，登录到合设的服务器上，不用切换开发环境、运行环境。
  - 分设场景下，若开发环境和运行环境上的操作系统架构不同，在开发环境中需使用交叉编译，这样编译出来的可执行文件，才可以在运行环境中执行。

  说明

  您可以登录对应的环境，执行“uname -a”命令查询其操作系统的架构。

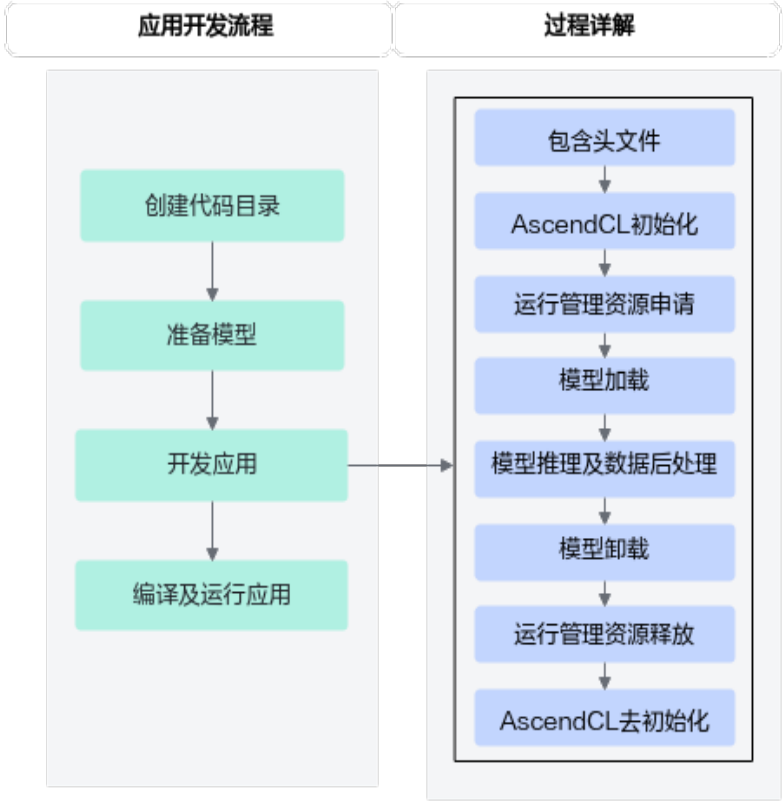
- 运行用户：

  运行驱动进程、推理业务或执行训练的用户。

了解开发过程

AscendCL（Ascend Computing Language）是一套用于在CANN（Compute Architecture for Neural Networks）上开发深度神经网络推理应用的C语言API库，提供模型加载与执行、媒体数据处理、算子加载与执行等API，能够实现在昇腾CANN平台上进行深度学习推理计算、图形图像预处理、单算子加速计算等能力。

图 2-2 开发流程



了解了这些大步骤后，下面我们再展开来说明开发应用具体涉及哪些关键功能？各功能又使用哪些AscendCL接口，这些AscendCL接口怎么串联？

虽然此时您可能不理解所有细节，但这也不影响，通过快速入门旨在先了解整体的代码逻辑，后续再深入学习，了解其它细节。

创建代码目录

您可以单击**Link**，获取基础的样例包MyFirstApp_ONNX.zip，以运行用户将该样例包**上传**至开发环境（例如，存放在/root/demo目录下），先执行“**chmod +x MyFirstApp_ONNX.zip**”命令增加执行权限，再执行“**unzip MyFirstApp_ONNX.zip**”命令解压样例包。

该样例包中包含测试图片、测试图片预处理脚本、编译脚本等，如下所示。

```
MyFirstApp_ONNX
├── data
│   ├── dog1_1024_683.jpg          // 测试图片
├── model                          // 用于存放ResNet-50模型文件
├── script
│   ├── transferPic.py             // 将测试图片预处理为符合模型要求的图片
                                   // 包括将*.jpg转换为*.bin，同时将图片从1024*683的分辨率缩放为224*224
├── src
│   ├── CMakeLists.txt             // cmake编译脚本
│   ├── main.cpp                   // 主函数，图片分类功能的实现文件
├── sample_build.sh                // 编译代码的脚本
├── sample_run.sh                  // 运行应用的脚本
```

准备模型

对于开源框架的网络模型，不能直接在昇腾AI处理器上做推理，需要先使用ATC（Ascend Tensor Compiler）工具将开源框架的网络模型转换为适配昇腾AI处理器的离线模型（*.om文件）。

步骤1 以运行用户登录开发环境。

步骤2 执行模型转换。

执行以下命令，将原始模型转换为昇腾AI处理器能识别的*.om模型文件。请注意，执行命令的用户需具有命令中相关路径的可读、可写权限。以下命令中的“***<SAMPLE_DIR>***”请根据实际样例包的存放目录替换、“***<soc_version>***”请根据实际昇腾AI处理器版本替换。

```
cd <SAMPLE_DIR>/MyFirstApp_ONNX/model
wget https://obs-9be7.obs.cn-east-2.myhuaweicloud.com/003_Atc_Models/resnet50/resnet50.onnx
atc --model=resnet50.onnx --framework=5 --output=resnet50 --input_shape="actual_input_1:1,3,224,224"  --soc_version=<soc_version>
```

各参数的解释如下，详细约束说明请参见《**ATC工具使用指南（开放态）**》。

- --model：ResNet-50网络的模型文件的路径。
- --framework：原始框架类型。5表示ONNX。
- --output：resnet50.om模型文件的路径。请注意，记录保存该om模型文件的路径，后续开发应用时需要使用。
- --input_shape：模型输入数据的shape。
- --soc_version：昇腾AI处理器的版本。

说明

如果无法确定当前设备的soc_version，则在安装NPU驱动包的服务器执行**npu-smi info**命令进行查询，在查询到的“Name”前增加Ascend信息，例如“Name”对应取值为*xxxyy*，实际配置的soc_version值为Ascend*xxxyy*。

----结束

开发应用

有了模型，接下来可以使用AscendCL接口加载、执行模型，开发AI应用了。

在开发环境中，打开“MyFirstApp_ONNX/src/main.cpp“文件，在该cpp中添加如下步骤中灰色底纹的代码。

步骤1 **定义一个main函数**，按照**了解开发过程**中的开发应用的过程详解，串联整个应用的代码逻辑。

```
int main()
{
    // 1.定义一个资源初始化的函数，用于AscendCL初始化、运行管理资源申请（指定计算设备）
  InitResource();

    // 2.定义一个模型加载的函数，加载图片分类的模型，用于后续推理使用
  const char *modelPath = "../model/resnet50.om";
  LoadModel(modelPath);

    // 3.定义一个读图片数据的函数，将测试图片数据读入内存，并传输到Device侧，用于后续推理使用
    const char *picturePath = "../data/dog1_1024_683.bin";
  LoadPicture(picturePath);

    // 4.定义一个推理的函数，用于执行推理
```

```
    Inference();

        // 5.定义一个推理结果数据处理的函数，用于在终端上屏显测试图片的top5置信度的类别编号
    PrintResult();

        // 6.定义一个模型卸载的函数，卸载图片分类的模型
    UnloadModel();

        // 7.定义一个函数，用于释放内存、销毁推理相关的数据类型，防止内存泄露
    UnloadPicture();

        // 8.定义一个资源去初始化的函数，用于AscendCL去初始化、运行管理资源释放（释放计算设备）
    DestroyResource();
}
```

了解总体的代码逻辑后，接下来开始写各自定义函数的实现，**以下函数的实现请按顺序添加在main函数之前**。

步骤2 include依赖的头文件，包括AscendCL、C或C++标准库的头文件。

在“MyFirstApp_ONNX/src/main.cpp”文件的开头增加如下代码。

```
#include "acl/acl.h"
#include <iostream>
#include <fstream>
#include <cstring>
#include <map>
#include <math.h>

using namespace std;
```

步骤3 资源初始化。

资源初始化包括2部分：

- 调用aclInit接口初始化AscendCL：

  使用AscendCL接口开发应用时，必须先初始化AscendCL ，否则可能会导致后续系统内部资源初始化出错，进而导致其它业务异常。

  在初始化时，还支持跟推理相关的可配置项（例如，性能相关的采集信息配置），以json格式的配置文件传入AscendCL初始化接口。如果当前的默认配置已满足需求（例如，默认不开启性能相关的采集信息配置），无需修改，可向AscendCL初始化接口中传入nullptr。

- 调用aclrtSetDevice接口指定计算设备。

```
int32_t deviceId = 0;
void InitResource()
{
    aclError ret = aclInit(nullptr);
    ret = aclrtSetDevice(deviceId);
}
```

有初始化就有去初始化，在确定完成了AscendCL的所有调用之后，或者进程退出之前，需执行资源去初始化，请参见**步骤10**。

步骤4 模型加载。

此处通过加载om模型文件来实现模型加载，如何获取om模型文件请参见**准备模型**。

```
uint32_t modelId;
void LoadModel(const char* modelPath)
{
    aclError ret = aclmdlLoadFromFile(modelPath, &modelId);
}
```

有加载就有卸载，模型推理结束后，需要卸载模型，请参见**步骤8**。

步骤5 将测试图片读入内存，再传输到Device侧，供推理使用。

```
size_t pictureDataSize = 0;
void *pictureHostData;
void *pictureDeviceData;

//申请内存，使用C/C++标准库的函数将测试图片读入内存
void ReadPictureTotHost(const char *picturePath)
{
    string fileName = picturePath;
    ifstream binFile(fileName, ifstream::binary);
    binFile.seekg(0, binFile.end);
    pictureDataSize = binFile.tellg();
    binFile.seekg(0, binFile.beg);
    aclError ret = aclrtMallocHost(&pictureHostData, pictureDataSize);
    binFile.read((char*)pictureHostData, pictureDataSize);
    binFile.close();
}

//申请Device侧的内存，再以内存复制的方式将内存中的图片数据传输到Device
void CopyDataFromHostToDevice()
{
    aclError ret = aclrtMalloc(&pictureDeviceData, pictureDataSize, ACL_MEM_MALLOC_HUGE_FIRST);
    ret = aclrtMemcpy(pictureDeviceData, pictureDataSize, pictureHostData, pictureDataSize,
ACL_MEMCPY_HOST_TO_DEVICE);
}

void LoadPicture(const char* picturePath)
{
    ReadPictureTotHost(picturePath);
    CopyDataFromHostToDevice();
}
```

步骤6 执行推理。

在调用AscendCL接口进行模型推理时，模型推理有输入、输出数据，输入、输出数据需要按照AscendCL规定的数据类型存放。相关数据类型如下：

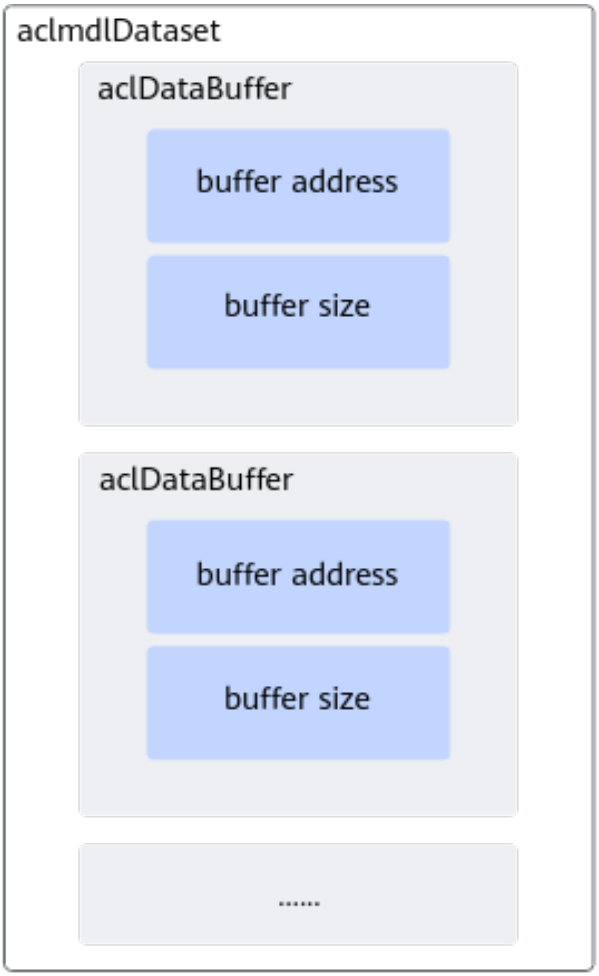
- 使用aclmdlDesc类型的数据描述模型基本信息（例如输入/输出的个数、名称、数据类型、Format、维度信息等）。

  模型加载成功后，用户可根据模型的ID，调用该数据类型下的操作接口获取该模型的描述信息，进而从模型的描述信息中获取模型输入/输出的个数、内存大小、维度信息、Format、数据类型等信息。
- 使用aclDataBuffer类型的数据来描述每个输入/输出的内存地址、内存大小。

  调用aclDataBuffer类型下的操作接口获取内存地址、内存大小等，便于向内存中存放输入数据、获取输出数据。
- 使用aclmdlDataset类型的数据描述模型的输入/输出数据。

  模型可能存在多个输入、多个输出，调用aclmdlDataset类型的操作接口添加多个aclDataBuffer类型的数据。

图 2-3 aclmdlDataset 类型与 aclDataBuffer 类型的关系




```
aclmdlDataset *inputDataSet;
aclDataBuffer *inputDataBuffer;
aclmdlDataset *outputDataSet;
aclDataBuffer *outputDataBuffer;
aclmdlDesc *modelDesc;
size_t outputDataSize = 0;
void *outputDeviceData;

// 准备模型推理的输入数据结构
void CreateModelInput()
{
        // 创建aclmdlDataset类型的数据，描述模型推理的输入
    inputDataSet = aclmdlCreateDataset();
    inputDataBuffer = aclCreateDataBuffer(pictureDeviceData, pictureDataSize);
    aclError ret = aclmdlAddDatasetBuffer(inputDataSet, inputDataBuffer);
}

// 准备模型推理的输出数据结构
void CreateModelOutput()
{
        // 创建模型描述信息
    modelDesc =  aclmdlCreateDesc();
    aclError ret = aclmdlGetDesc(modelDesc, modelId);
        // 创建aclmdlDataset类型的数据，描述模型推理的输出
    outputDataSet = aclmdlCreateDataset();
        // 获取模型输出数据需占用的内存大小，单位为Byte
    outputDataSize = aclmdlGetOutputSizeByIndex(modelDesc, 0);
        // 申请输出内存
    ret = aclrtMalloc(&outputDeviceData, outputDataSize, ACL_MEM_MALLOC_HUGE_FIRST);
```

```
    outputDataBuffer = aclCreateDataBuffer(outputDeviceData, outputDataSize);
    ret = aclmdlAddDatasetBuffer(outputDataSet, outputDataBuffer);
}

// 执行模型
void Inference()
{
        CreateModelInput();
    CreateModelOutput();
    aclError ret = aclmdlExecute(modelId, inputDataSet, outputDataSet);
}
```

步骤7 处理模型推理的结果数据，在屏幕上显示图片的top5置信度的类别编号。

```
void *outputHostData;

void PrintResult()
{
        // 获取推理结果数据
        aclError ret = aclrtMallocHost(&outputHostData, outputDataSize);
        ret = aclrtMemcpy(outputHostData, outputDataSize, outputDeviceData, outputDataSize,
ACL_MEMCPY_DEVICE_TO_HOST);
        // 将内存中的数据转换为float类型
        float* outFloatData = reinterpret_cast<float *>(outputHostData);

        // 屏显测试图片的top5置信度的类别编号
        map<float, unsigned int, greater<float>> resultMap;
        for (unsigned int j = 0; j < outputDataSize / sizeof(float);++j)
        {
                resultMap[*outFloatData] = j;
                outFloatData++;
        }

        // do data processing with softmax and print top 5 classes
        double totalValue=0.0;
        for (auto it = resultMap.begin(); it != resultMap.end(); ++it) {
            totalValue += exp(it->first);
        }

        int cnt = 0;
        for (auto it = resultMap.begin();it != resultMap.end();++it)
        {
                if(++cnt > 5)
                {
                        break;
                }
                printf("top %d: index[%d] value[%lf] \n", cnt, it->second, exp(it->first) /totalValue);
        }
}
```

步骤8 卸载模型，并释放模型描述信息。

推理结束，需及时释放模型描述信息、卸载模型。

```
void UnloadModel()
{
        // 释放模型描述信息
    aclmdlDestroyDesc(modelDesc);
        // 卸载模型
    aclmdlUnload(modelId);
}
```

步骤9 释放内存、销毁推理相关的数据类型。

```
void UnloadPicture()
{
    aclError ret = aclrtFreeHost(pictureHostData);
    pictureHostData = nullptr;
    ret = aclrtFree(pictureDeviceData);
    pictureDeviceData = nullptr;
    aclDestroyDataBuffer(inputDataBuffer);
    inputDataBuffer = nullptr;
```

```
    aclmdlDestroyDataset(inputDataSet);
    inputDataSet = nullptr;

    ret = aclrtFreeHost(outputHostData);
    outputHostData = nullptr;
    ret = aclrtFree(outputDeviceData);
    outputDeviceData = nullptr;
    aclDestroyDataBuffer(outputDataBuffer);
    outputDataBuffer = nullptr;
    aclmdlDestroyDataset(outputDataSet);
    outputDataSet = nullptr;
}
```

步骤10 资源释放。

在确定完成了AscendCL的所有调用之后，或者进程退出之前，需调用如下接口实现计算设备释放，AscendCL去初始化。

```
void DestroyResource()
{
    aclError ret = aclrtResetDevice(deviceId);
    aclFinalize();
}
```

----结束

编译及运行应用

步骤1 编译代码。

以运行用户登录开发环境，切换到MyFirstApp_ONNX目录下，执行以下命令。

如下为设置环境变量的示例，<SAMPLE_DIR>表示样例所在的目录，*$HOME/Ascend/ascend-toolkit*表示CANN软件包的安装路径，*<arch-os>*中*arch*表示操作系统架构（需根据运行环境的架构选择），*os*表示操作系统（需根据运行环境的操作系统选择）。

```
export APP_SOURCE_PATH=<SAMPLE_DIR>/MyFirstApp_ONNX
export DDK_PATH=$HOME/Ascend/ascend-toolkit/latest
export NPU_HOST_LIB=$HOME/Ascend/ascend-toolkit/latest/<arch-os>/devlib
chmod +x sample_build.sh
./sample_build.sh
```

说明

- 如果执行脚本报错“ModuleNotFoundError: No module named 'PIL'”，是由于开发环境中缺少Pillow库，需使用**pip3 install Pillow --user**命令安装Pillow库后，再执行脚本。
- 如果执行脚本报错“bash: ./sample_build.sh: /bin/bash^M: bad interpreter: No such file or directory”，是由于开发环境中没有安装dos2unix包，需使用**sudo apt-get install dos2unix**命令安装dos2unix包后，执行**dos2unix sample_build.sh**，然后再执行脚本。

步骤2 运行应用。

以运行用户将MyFirstApp_ONNX目录上传至运行环境，以运行用户登录运行环境，切换到MyFirstApp_ONNX目录下，执行以下命令。

```
chmod +x sample_run.sh
./sample_run.sh
```

终端上屏显的结果如下，index表示类别标识、value表示该分类的最大置信度：

```
top 1: index[162] value[0.954676]
top 2: index[161] value[0.033442]
top 3: index[166] value[0.006534]
top 4: index[167] value[0.004561]
top 5: index[163] value[0.000315]
```

说明

类别标签和类别的对应关系与训练模型时使用的数据集有关，本样例使用的模型是基于imagenet数据集进行训练的，您可以在互联网上查阅对应数据集的标签及类别的对应关系。

当前屏显信息中的类别标识与类别的对应关系如下：

"162": ["beagle"]

"161": ["basset", "basset hound"]

"166": ["Walker hound", "Walker foxhound"]

"167": ["English foxhound"]

"163": ["bloodhound", "sleuthhound"]

----结束

3 AscendCL 概述

AscendCL架构及基本概念
本节介绍AscendCL的主要功能、Device/Stream/Context等基本概念以及这些概念之间的关系。

AscendCL接口调用流程
本节介绍AscendCL接口调用流程以及调用AscendCL接口依赖的头文件和库文件。

准备开发和运行环境

3.1 AscendCL 架构及基本概念

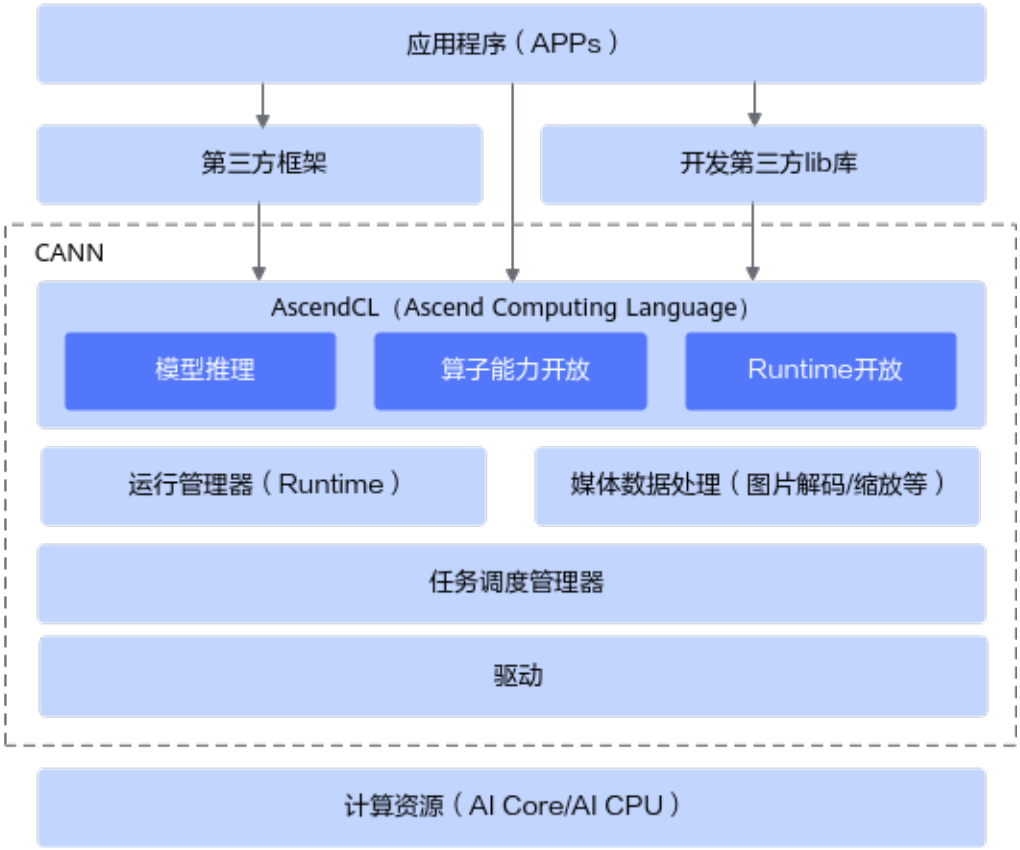
本节介绍AscendCL的主要功能、Device/Stream/Context等基本概念以及这些概念之间的关系。

AscendCL 是什么?

AscendCL（Ascend Computing Language）是一套用于在昇腾平台上开发深度神经网络应用的C语言API库，提供运行资源管理、内存管理、模型加载与执行、算子加载与执行、媒体数据处理等API，能够实现利用昇腾硬件计算资源、在昇腾CANN平台上进行**深度学习推理计算**、**图形图像预处理**、**单算子加速计算**等能力。简单来说，就是统一的API框架，实现对所有资源的调用。

计算资源层是昇腾AI处理器的硬件算力基础，主要完成神经网络的矩阵相关计算、完成控制算子/标量/向量等通用计算和执行控制功能、完成图像和视频数据的预处理，为深度神经网络计算提供了执行上的保障。

图 3-1 逻辑架构图



AscendCL的应用场景：

- **开发应用**：用户可以直接调用AscendCL提供的接口开发图片分类应用、目标识别应用等。
- **供第三方框架调用**：用户可以通过第三方框架调用AscendCL接口，以便使用昇腾AI处理器的计算能力。
- **供第三方开发lib库**：用户还可以使用AscendCL封装实现第三方lib库，以便提供昇腾AI处理器的运行管理、资源管理等能力。

AscendCL的优势：

- **高度抽象**：算子编译、加载、执行的API归一，相比每个算子一个API，AscendCL大幅减少API数量，降低复杂度。
- **向后兼容**：AscendCL具备向后兼容，确保软件升级后，基于旧版本编译的程序依然可以在新版本上运行。
- **零感知芯片**：一套AscendCL接口可以实现应用代码统一，多款昇腾AI处理器无差异。

基本概念

表 3-1 概念介绍

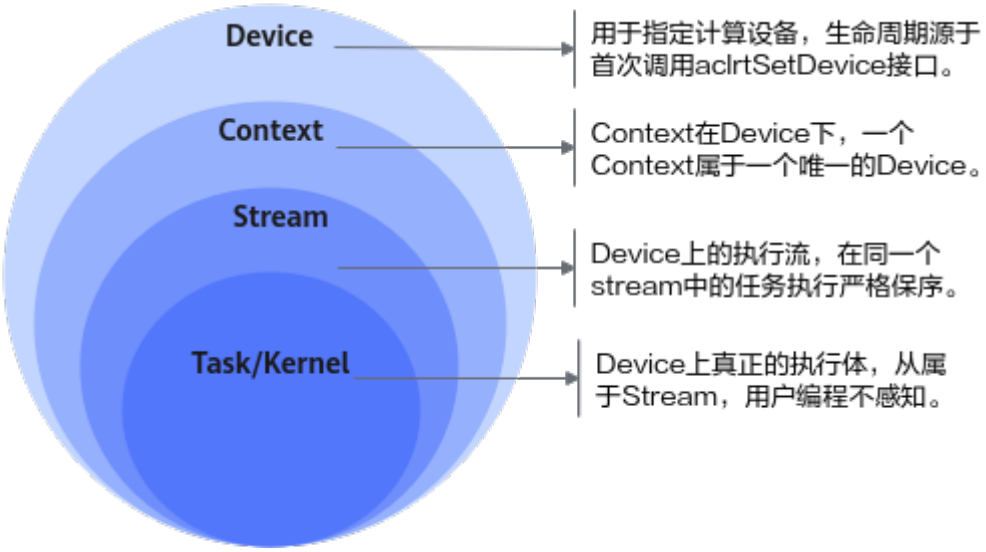
概念	描述
同步/异步	本文中提及的同步、异步是站在调用者和执行者的角度： ● 若在调用AscendCL接口后**不等待**Device侧的任务执行完成再返回，则表示调度是**异步**的； ● 若在调用AscendCL接口后**需等待**Device侧的任务执行完成再返回，则表示调度是**同步**的。
进程/线程	本文中提及的进程、线程，若无特别注明，则表示Host上的进程、线程。
Host	Host指与Device相连接的X86服务器、ARM服务器，会利用Device提供的NN（Neural-Network ）计算能力，完成业务。
Device	Device指安装了昇腾AI处理器的硬件设备，利用PCIe接口与Host侧连接，为Host提供NN计算能力。若存在多个Device，多个Device之间的内存资源不能共享。
Context	Context作为一个容器，管理了所有对象（包括Stream、Event、设备内存等）的生命周期。不同Context的Stream、不同Context的Event是完全隔离的，无法建立同步等待关系。 Context分为两种： ● 默认Context：调用**aclrtSetDevice**接口指定用于运算的Device时，系统会自动隐式创建一个默认Context，一个Device对应一个默认Context，默认Context不能通过**aclrtDestroyContext**接口来释放。 ● 显式创建的Context：**推荐**，在进程或线程中调用**aclrtCreateContext**接口显式创建一个Context。
Stream	Stream用于维护一些异步操作的执行顺序，确保按照应用程序中的代码调用顺序在Device上执行。 基于Stream的kernel执行和数据传输能够实现Host运算操作、Host与Device间的数据传输、Device内的运算并行。 Stream分两种： ● 默认Stream：调用**aclrtSetDevice**接口指定用于运算的Device时，系统会自动隐式创建一个默认Stream，一个Device对应一个默认Stream，默认Stream不能通过**aclrtDestroyStream**接口来释放。 ● 显式创建的Stream：**推荐**，在进程或线程中调用**aclrtCreateStream**接口显式创建一个Stream。

概念	描述
Event	支持调用AscendCL接口同步Stream之间的任务，例如同一个Device上的多个任务。 例如，若stream2的任务依赖stream1的任务，想保证stream1中的任务先完成，这时可创建一个Event，并将Event插入到stream1，在执行stream2的任务前，先同步等待Event完成。
AIPP	AIPP（Artificial Intelligence Pre-Processing）用于在AI Core上完成图像预处理，包括色域转换（转换图像格式）、图像归一化（减均值/乘系数）和抠图（指定抠图起始点，抠出神经网络需要大小的图片）等。 AIPP区分为静态AIPP和动态AIPP。您只能选择静态AIPP或动态AIPP方式来处理图片，不能同时配置静态AIPP和动态AIPP两种方式。 ● 静态AIPP：模型转换时设置AIPP模式为静态，同时设置AIPP参数，模型生成后，AIPP参数值被保存在离线模型（*.om）中，每次模型推理过程采用固定的AIPP预处理参数（无法修改）。 如果使用静态AIPP方式，多Batch情况下共用同一份AIPP参数。 ● 动态AIPP：模型转换时设置AIPP模式为动态，每次模型推理前，根据需求，在执行模型前设置动态AIPP参数值，然后在模型执行时可使用不同的AIPP参数。 如果使用动态AIPP方式，多Batch可使用不同的AIPP参数。
动态Batch/动态分辨率	在某些场景下，模型每次输入的batch size或分辨率是不固定的，如检测出目标后再执行目标识别网络，由于目标个数不固定导致目标识别网络输入BatchSize不固定。 ● 动态Batch：用户执行推理时，其batch size是动态可变的。 ● 动态分辨率: 用户执行推理时，每张图片的分辨率H*W是动态可变的。
动态维度（ND格式）	为了支持Transformer等网络在输入格式的维度不确定的场景，需要支持ND格式下任意维度的动态设置。 ND表示支持任意格式，当前N<=4。
通道	在RGB色彩模式下，图像通道就是指单独的红色R、绿色G、蓝色B部分。也就是说，一幅完整的图像，是由红色绿色蓝色三个通道组成的，它们共同作用产生了完整的图像。同样在HSV色系中指的是色调H，饱和度S，亮度V三个通道。
开放形态	开放Device上的Control CPU，用户可以直接在Device上运行应用，利用Device侧Control CPU的通用算力，降低Host侧CPU的负载，从而可降低对Host CPU的硬件要求，减少Host与Device之间的数据传输通信开销，提升整体处理性能。 Atlas 200/300/500 推理产品支持开放形态。 Atlas 推理系列产品支持开放形态。

概念	描述
EP模式	以昇腾 AI 处理器的PCIe的工作模式进行区分，如果PCIe工作在从模式，则称为EP模式。

Device、Context、Stream 之间的关系

图 3-2 Device、Context、Stream 之间的关系



- **Device**，用于指定计算设备。
  - Device的生命周期源于首次调用**aclrtSetDevice**接口。
  - 每次调用**aclrtSetDevice**接口，系统会进行引用计数加1；调用**aclrtResetdevice**接口，系统会进行引用计数减1。
  - 当引用计数减为零时，在本进程中Device上的资源不可用。
- **Context**，在Device下，一个Context一定属于一个唯一的Device。
  - Context分隐式创建和显式创建。
  - 隐式创建的Context（即默认Context），生命周期始于调用**aclrtSetDevice**接口，终结于调用**aclrtResetdevice**接口使引用计数为零时。隐式Context只会被创建一次，调用**aclrtSetDevice**接口重复指定同一个Device，只增加隐式创建的Context的引用计数。
  - 显式创建的Context，生命周期始于调用**aclrtCreateContext**接口，终结于调用**aclrtDestroyContext**接口。
  - 若在某一进程内创建多个Context（Context的数量与Stream相关，Stream数量有限制，请参见**11.8.1 aclrtCreateStream**），当前线程在同一时刻内只能使用其中一个Context，建议通过**aclrtSetCurrentContext**接口明确指定当前线程的Context，增加程序的可维护性。
  - 进程内的Context是共享的，可以通过**aclrtSetCurrentContext**进行切换。
- **Stream**，是Device上的执行流，在同一个stream中的任务执行严格保序。
  - Stream分隐式创建和显式创建。

  - 每个Context都会包含一个默认Stream，这个属于隐式创建，隐式创建的stream生命周期同归属的Context。
  - 用户可以显式创建stream，显式创建的stream生命周期始于调用**aclrtCreateStream**，终结于调用**aclrtDestroyStream**接口。显式创建的stream归属的Context被销毁或生命周期结束后，会影响该stream的使用，虽然此时stream没有被销毁，但不可再用。
- **Task/Kernel**，是Device上真正的任务执行体。

线程、Context、Stream 之间的关系

- 一个用户线程一定会绑定一个Context，所有Device的资源使用或调度，都必须基于Context。
- 一个线程中当前会有一个唯一的Context在用，Context中已经关联了本线程要使用的Device。
- 可以通过**aclrtSetCurrentContext**进行Device的快速切换。示例代码如下，仅供参考，不可以直接拷贝编译运行：

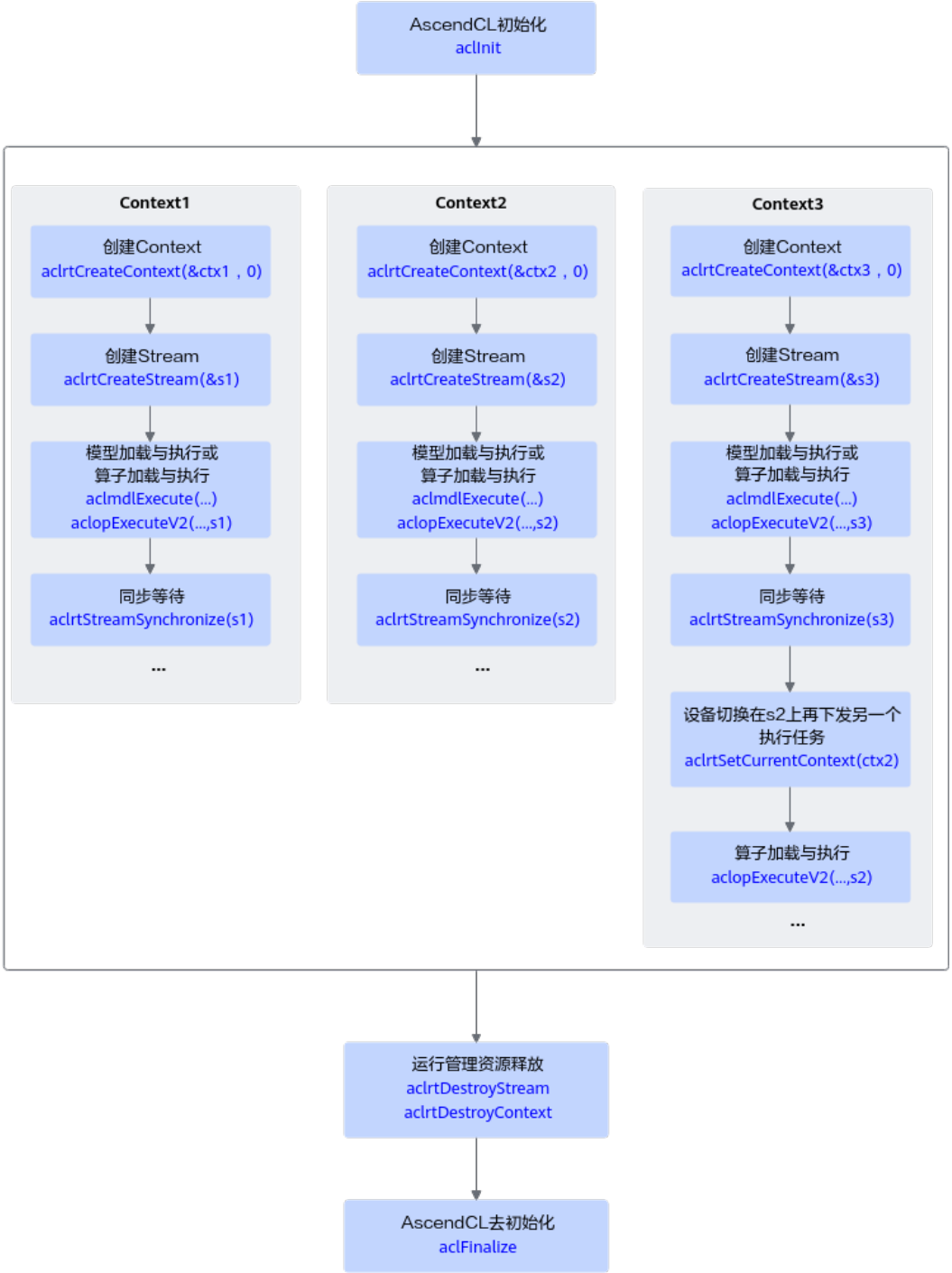
```
...
aclrtCreateContext(&ctx1,0);
aclrtCreateStream(&s1);
aclopExecuteV2(op1,...,s1);
aclrtCreateContext(&ctx2,1);

/*在当前线程中，创建ctx2后，当前线程对应的Context切换为ctx2，对应在Device 1进行后续的计算任务，本例中将在Device 1上进行op2的执行调用 */
aclrtCreateStream(&s2);
aclopExecuteV2(op2,...,s2);
aclrtSetCurrentContext(ctx1);

/*在当前线程中，通过Context切换，使后续计算任务在对应的Device 0上进行*/
aclopExecuteV2(op3,...,s1);
...
```

- 一个线程中可以创建多个Stream，不同的Stream上计算任务是可以并行执行；多线程场景下，推荐每个线程创建一个Stream，线程之间的Stream在Device上相互独立，每个Stream内部的任务是按照Stream下发的顺序执行。
- 多线程的调度依赖于运行应用的操作系统调度，多Stream在Device侧的调度，由Device上调度组件进行调度。

一个进程内多个线程间的 Context 迁移

- 一个进程中可以创建多个Context，但一个线程同一时刻只能使用一个Context。
- 线程中创建的多个Context，线程缺省使用最后一次创建的Context。
- 进程内创建的多个Context，可以通过**aclrtSetCurrentContext**设置当前需要使用的Context。

图 3-3 接口调用流程



默认 Context 和默认 Stream 的使用场景

- Device上执行操作下发前，必须有Context和Stream，这个Context、Stream可以显式创建，也可以隐式创建。**隐式创建**的Context、Stream就是默认Context、默认Stream。

  默认Stream作为接口入参时，直接传NULL。

- **默认Context**不允许用户执行**aclrtGetCurrentContext**或**aclrtSetCurrentContext**操作，也不允许执行**aclrtDestroyContext**操作。

- **默认Context、默认Stream**一般适用于简单应用，用户仅仅需要一个Device的计算场景下。多线程应用程序建议全部使用显式创建的Context和Stream。

示例代码如下，仅供参考，不可以直接拷贝编译运行：

```
...
aclInit(...);
aclrtSetDevice(0);

/*已经创建了一个default ctx，在default ctx中创建了一个default stream，并且在当前线程可用*/
...
aclopExecuteV2(op1,...,NULL);  //最后一个NULL表示在default stream上执行算子op1
aclopExecuteV2(op2,...,NULL); //最后一个NULL表示在default stream上执行算子op2
aclrtSynchronizeStream(NULL);

/*等待计算任务全部完成（op1、op2执行结束），用户根据需要获取计算任务的输出结果*/
...
aclrtResetDevice(0);  //释放计算设备0，对应的default ctx及default stream生命周期也终止。
```

多线程、多 stream 的性能说明

- 线程调度依赖运行的操作系统，Stream上下发了任务后，Stream的调度由Device的调度单元调度，但如果一个进程内的多Stream上的任务在Device存在资源争抢的时候，性能可能会比单Stream低。
- 当前昇腾AI处理器有不同的执行部件，如AI Core、AI CPU、Vector Core等，对应使用不同执行部件的任务，建议多Stream的创建按照算子执行引擎划分。
- 单线程多Stream与多线程多Stream（一个进程中可以包含多个线程，每个线程中一个Stream）性能上哪个更优，具体取决于应用本身的逻辑实现，一般来说前者性能略好，原因是相对后者，应用层少了线程调度开销。

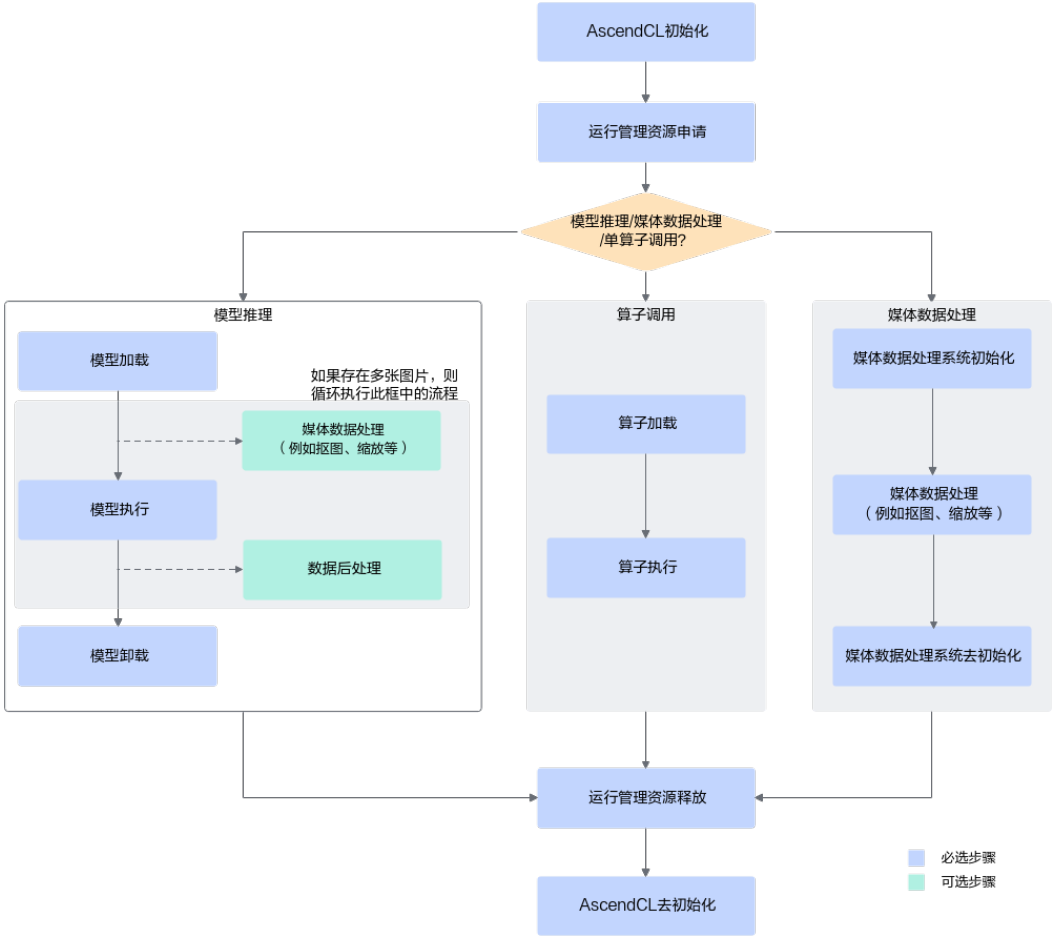
3.2 AscendCL 接口调用流程

本节介绍AscendCL接口调用流程以及调用AscendCL接口依赖的头文件和库文件。

接口调用流程

调用AscendCL接口，可开发包含模型推理、媒体数据处理、单算子调用等功能的应用，这些功能可以独立存在，也可以组合存在。下图给出了使用AscendCL接口开发AI应用的整体接口调用流程。

图 3-4 接口调用流程图



上图根据应用开发中的典型功能抽象出主要的接口调用流程，例如，如果模型对输入图片的宽高要求与用户提供的源图不一致，则需要媒体数据处理，将源图裁剪成符合模型的要求；如果需要实现模型推理的功能，则需要先加载模型，模型推理结束后，则需要卸载模型；如果模型推理后，需要从推理结果中查找最大置信度的类别标识对图片分类，则需要数据后处理。

1. AscendCL初始化。

   调用**aclInit**接口实现初始化AscendCL。

2. 运行管理资源申请。

   依次申请运行管理资源：**Device**、**Context**、**Stream**。

   具体流程，请参见**4.4 运行管理资源申请与释放**。

3. 模型推理/单算子调用/媒体数据处理。

   – **模型推理**

     i. 模型加载：模型推理前，需要先将对应的模型加载到系统中。

     接口调用流程，请参见**4.6.1 模型加载**。

     但加载模型前，必须要有适配昇腾AI处理器的离线模型，需提前构建模型，请参见**4.2 模型构建**。

     ii. （可选）**媒体数据处理**：可实现JPEG图片解码、视频解码、抠图/图片缩放/格式转换、JPEG图片编码等功能。

接口调用流程，请参见**5 媒体数据处理（含图像/视频等）**

iii. 模型执行：使用模型实现图片分类、目标识别等功能。

接口调用流程，请参见**4.6.2 模型执行**。

iv. （可选）数据后处理：处理模型推理的结果，此处根据用户的实际需求来处理推理结果，例如用户可以将获取到的推理结果写入文件、从推理结果中找到每张图片最大置信度的类别标识等。

v. 模型卸载：调用**aclmdlUnload**接口卸载模型。

– **算子调用**

如果AI应用中不仅仅包括模型推理，还有数学运算（例如BLAS基础线性代数运算）、数据类型转换等功能，也想使用昇腾的算力，直接通过AscendCL接口加载并执行单个算子，省去模型构建、训练的过程，相对轻量级，又可以使用昇腾的算力。另外，自定义的算子，也可以通过单算子调用的方式来验证算子的功能。

算子调用的接口调用流程，请参见**6.2 单算子调用流程**。

4. 运行管理资源释放。

所有数据处理都结束后，需要依次释放运行管理资源：**Stream**、**Context**、**Device**。

接口调用流程，请参见**4.4 运行管理资源申请与释放**。

5. AscendCL去初始化。

调用**aclFinalize**接口实现AscendCL去初始化。

说明

在应用开发过程中，各环节都涉及内存的申请与释放、数据传输（通过内存复制实现）、数据类型的创建与销毁，因此未在图中一一标识，关于内存申请与释放、内存复制的接口请参见**11.11 内存管理**，数据类型的创建与销毁的接口请参见**11.27 数据类型及其操作接口**。

调用接口依赖的头文件和库文件说明

您需要根据实际使用的接口来include依赖的文件，AscendCL中各头文件的用途如下表所示。

AscendCL头文件在“CANN软件安装后文件存储路径/include/”目录下，AscendCL库文件在“CANN软件安装后文件存储路径/lib64/”目录下。

须知

编译基于AscendCL接口的代码逻辑时，请按照include的头文件依赖对应的库文件，如果引用多余的so文件（例如libascendcl.a），可能导致版本功能异常或后续版本升级时存在兼容性问题。

表 3-2 头文件列表

定义接口的头文件	用途	对应的库文件
acl/acl_base.h	用于定义基本的数据类型（例如aclDataBuffer、aclTensorDesc等）及其操作接口、枚举值（例如aclFormat）、日志管理接口等。	libascendcl.so
acl/acl.h	该头文件中已包含acl/acl_mdl.h、acl/acl_rt.h、acl/acl_op.h。包含acl.h文件后，可以引用初始化/去初始化、Device管理、算力Group查询与设置、Context管理、Stream管理、同步等待、内存管理、模型加载与执行、算子编译（不包括aclopCompile接口）、算子加载与执行（不包括aclopCompileAndExecute接口）等接口。	libascendcl.so
acl/acl_prof.h	用于定义Profiling配置的接口。	libmsprofiler.so **说明** 为了兼容旧版本，旧版本中支持使用libascendcl.so，但后续版本这种方式会废弃，建议使用libmsprofiler.so，防止后续版本出现兼容性问题。
acl/ops/acl_cblas.h	用于定义CBLAS接口。	libacl_cblas.so
acl/ops/acl_dvpp.h	用于定义媒体数据处理V1版本的接口。	libacl_dvpp.so
acl/ops/acl_fv.h	用于定义特征向量检索的接口。 Atlas 200/300/500 推理产品，不支持引用该头文件中的接口。	libacl_retr.so
acl/acl_op_compiler.h	用于定义aclopCompile、aclopCompileAndExecute、aclSetCompileopt等算子在线编译相关的接口、数据类型、枚举值等。	libacl_op_compiler.so
acl/acl_tdt.h	用于定义Tensor数据传输接口。 Atlas 200/300/500 推理产品，不支持引用该头文件中的接口。 Atlas 推理系列产品，不支持引用该头文件中的接口。	libacl_tdt_channel.so
acl/acl_tdt_queue.h	用于定义共享队列管理、共享Buffer管理接口。	libacl_tdt_queue.so

定义接口的头文件	用途	对应的库文件
acl/dvpp/hi_dvpp.h	用于定义媒体数据处理V2版本的接口。	libacl_dvpp_mpi.so
hi_mpi_vi.h hi_common_vi.h hi_common_dis.h hi_common_gdc.h hi_media_common.h hi_media_type.h hi_mpi_sys.h	用于定义VI（Vedio Input）视频数据获取功能的接口。 Atlas 200/300/500 推理产品，不支持引用该头文件中的接口。 Atlas 推理系列产品，不支持引用该头文件中的接口。	libacl_vi_mpi.so libacl_dvpp_mpi.so
acl/media目录下： hi_mpi_isp.h hi_common_isp.h hi_common_3a.h hi_mpi_ae.h hi_common_ae.h hi_mpi_awb.h hi_common_awb.h hi_common_sns.h hi_media_common.h hi_media_type.h hi_mpi_sys.h	用于定义ISP（Image Signal Processing）系统控制功能的接口。 Atlas 200/300/500 推理产品，不支持引用该头文件中的接口。 Atlas 推理系列产品，不支持引用该头文件中的接口。	libacl_isp_ae_mpi.so libacl_isp_awb_mpi.so libacl_isp_mpi.so libacl_dvpp_mpi.so
acl/media目录下： hi_mpi_vpss.h hi_media_common.h hi_media_type.h hi_mpi_sys.h	用于定义VPSS（Video Process Sub-System）图像处理功能的接口。 Atlas 200/300/500 推理产品，不支持引用该头文件中的接口。 Atlas 推理系列产品，不支持引用该头文件中的接口。	libacl_vpss_mpi.so libacl_dvpp_mpi.so
acl/media/hi_mipi_rx.h	用于定义MIPI Rx ioctl命令字。 Atlas 200/300/500 推理产品，不支持引用该头文件中的接口。 Atlas 推理系列产品，不支持引用该头文件中的接口。	-

定义接口的头文件	用途	对应的库文件
acl/media目录下： hi_mpi_audio.h hi_common_aio.h	用于定义频输入、音频输出功能的接口。 Atlas 200/300/500 推理产品，不支持引用该头文件中的接口。 Atlas 推理系列产品，不支持引用该头文件中的接口。	libacl_audio_mpi.so
acl/media/ hi_acodec.h	用于定义音量调整的命令字。 Atlas 200/300/500 推理产品，不支持引用该头文件中的接口。 Atlas 推理系列产品，不支持引用该头文件中的接口。	-
acl/media目录下： hi_common_vo.h hi_mpi_vo.h	用于定义视频输出接口。 Atlas 200/300/500 推理产品，不支持引用该头文件中的接口。 Atlas 推理系列产品，不支持引用该头文件中的接口。	libacl_vo_mpi.so
acl/media/ hi_mpi_hdmi.h	用于定义对接外设的HDMI接口。 Atlas 200/300/500 推理产品，不支持引用该头文件中的接口。 Atlas 推理系列产品，不支持引用该头文件中的接口。	libacl_hdmi_mpi.so
acl/media/ hi_mpi_tde.h	用于定义TDE图形绘制接口。 Atlas 200/300/500 推理产品，不支持引用该头文件中的接口。 Atlas 推理系列产品，不支持引用该头文件中的接口。	libacl_tde_mpi.so
acl/media/hifb.h	用于定义叠加图形层管理接口。 Atlas 200/300/500 推理产品，不支持引用该头文件中的接口。 Atlas 推理系列产品，不支持引用该头文件中的接口。	-
aclnn/acl_meta.h	用于定义依赖的AscendCL meta接口，通过这些元接口可构建不同数据结构，如aclTensor、aclScalar、aclIntArray等。 Atlas 200/300/500 推理产品，不支持引用该头文件中的接口。 Atlas 推理系列产品，不支持引用该头文件中的接口。	libnnopbase.so

定义接口的头文件	用途	对应的库文件
aclnn/aclnn_base.h	用于定义NN类算子公共的base接口，即aclnnInit和aclnnFinalize。 Atlas 200/300/500 推理产品，不支持引用该头文件中的接口。 Atlas 推理系列产品，不支持引用该头文件中的接口。	libnnopbase.so
aclnnop/aclnn_*.h （*表示具体的算子名称） **说明** 为兼容旧版本，旧版本NN类算子头文件路径是aclnnop/level2/aclnn_*.h，但后续版本该路径会废弃，建议使用新路径aclnnop/aclnn_*.h，防止后续版本出现兼容性问题。	用于定义NN类算子的功能接口。 Atlas 200/300/500 推理产品，不支持引用该头文件中的接口。 Atlas 推理系列产品，不支持引用该头文件中的接口。	libopapi.so
acldvppop/acldvpp_base.h acldvppop/acldvpp_op_api.h	用于定义DVPP媒体数据处理类算子的功能接口。 Atlas 200/300/500 推理产品，不支持引用该头文件中的接口。 Atlas 推理系列产品，不支持引用该头文件中的接口。	libacl_dvpp_op.so

3.3 准备开发和运行环境

部署开发环境和运行环境，请参见《**CANN 软件安装指南 (开放态)**》对应Atlas产品的描述。

- 部署开发环境后，才能获取调用接口所需的头文件、编译运行接口所需的库文件。

  对于昇腾设备，已安装驱动、固件场景下，该环境可直接作为运行环境，执行编译生成的应用可执行文件。
- 部署运行环境后，才能在运行环境上执行编译生成的应用可执行文件。

说明

- 需要根据运行环境的安装包，确定引用的组件目录，否则会导致运行报错。安装方案请参见《**CANN 软件安装指南 (开放态)**》。

  安装CANN软件后，使用CANN运行用户编译、运行时，需要以CANN运行用户登录环境，执行**source** *${install_path}*/**set_env.sh**命令设置环境变量，其中${install_path}为CANN软件的安装目录。

  - 运行环境安装nnrt包，则开发过程中引用对应AscendCL目录。
    - 头文件路径：CANN软件安装后文件存储路径/nnrt/latest/include/acl
    - 库文件路径：CANN软件安装后文件存储路径/nnrt/latest/lib64
  - 运行环境安装nnae包，则开发过程中引用对应AscendCL目录。
    - 头文件路径：CANN软件安装后文件存储路径/nnae/latest/include/acl
    - 库文件路径：CANN软件安装后文件存储路径/nnae/latest/lib64
- 本文中的操作步骤（包括模型转换、编译代码、运行应用等）需以运行用户登录开发环境或运行环境后再执行，**请务必**获取各组件的运行用户，以便后续操作时使用。**开放形态**下，运行环境上的运行用户固定为HwHiAiUser。

- **（可选）通过环境变量ASCEND_CACHE_PATH、ASCEND_WORK_PATH设置AscendCL应用编译运行过程中产生的文件的落盘路径**，涉及ATC模型转换、AscendCL应用编译配置、AOE模型智能调优、性能数据采集、日志采集等功能，落盘文件包括算子编译缓存文件、知识库文件、调优结果文件、性能数据文件、日志文件等。

  配置示例如下，详细配置说明请参见《**环境变量参考**》：

```
export ASCEND_CACHE_PATH=/repo/task001/cache
export ASCEND_WORK_PATH=/repo/task001/172.16.1.12_01_03
```

说明

通过export命令，设置环境变量只在当前终端窗口生效，且只对设置环境变量之后启动的昇腾AI应用进程生效。

若将export命令写入~/.bashrc文件，使环境变量永久生效，则环境变量对该用户下的所有昇腾AI应用进程都生效。这种方式，可能会影响其它不需要调整Device ID的应用进程，**请谨慎使用**。将export命令写入~/.bashrc文件的方法如下：

1. 以安装用户在任意目录下执行**vi ~/.bashrc**，在该文件最后添加上述内容。
2. 执行**:wq!**命令保存文件并退出。
3. 执行**source ~/.bashrc**使环境变量生效。

4 基础推理应用

推理应用开发流程
本节介绍基于AscendCL接口开发基础推理应用的开发流程。

模型构建
对于开源框架的网络模型（如ONNX、TensorFlow等），不能直接在昇腾AI处理器上做推理，需要先使用ATC（Ascend Tensor Compiler）工具将开源框架的网络模型转换为适配昇腾AI处理器的离线模型（*.om文件）。

AscendCL初始化与去初始化
本节介绍AscendCL初始化与去初始化的相关接口、注意事项，并给出示例代码。

运行管理资源申请与释放
本节介绍运行管理资源包括哪些、如何申请&释放这些资源，并给出示例代码。

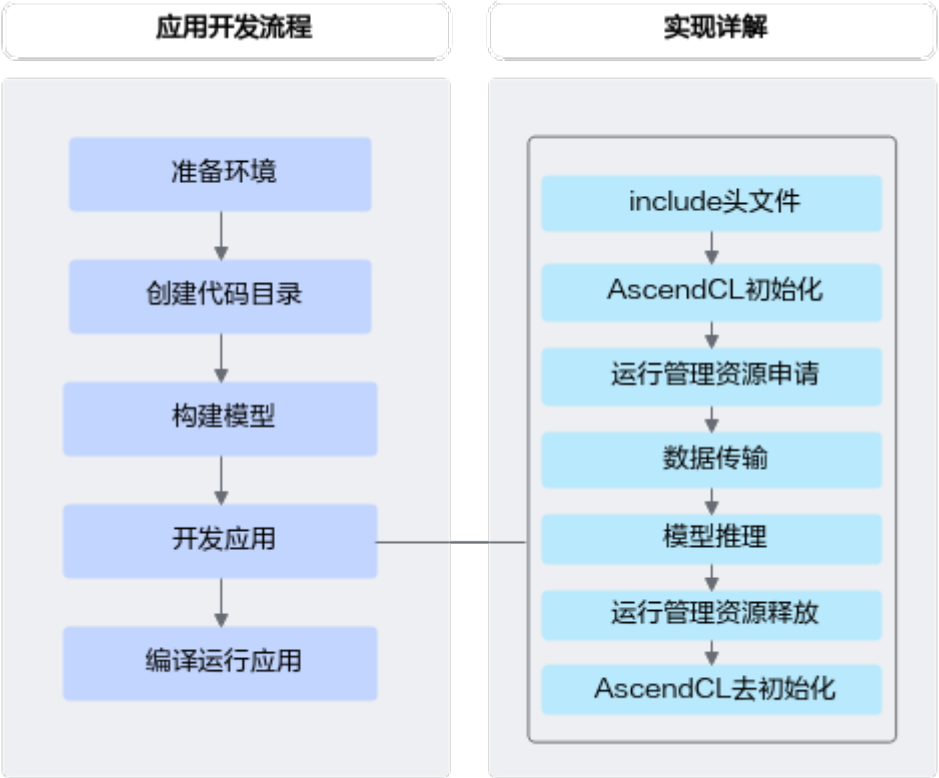
数据传输
本节介绍数据传输的相关接口、注意事项，并给出示例代码。

模型推理

4.1 推理应用开发流程

本节介绍基于AscendCL接口开发基础推理应用的开发流程。

图 4-1 开发流程



1. **准备环境**。
2. **创建代码目录**。

   在开发应用前，您需要先创建目录，存放代码文件、编译脚本、测试图片数据、模型文件等。

   如下仅是示例，供参考：

```
├App名称
├── model              // 该目录下存放模型文件
│   ├── xxxxxx

├── data
│   ├── xxx.jpg        // 测试数据

├── inc                // 该目录下存放声明函数的头文件
│   ├── xxx.h

├── out                // 该目录下存放输出结果

├── src
│   ├── xxx.json       // 系统初始化的配置文件
│   ├── CMakeLists.txt // 编译脚本
│   ├── xxx.cpp        // 实现文件
```

3. **构建模型**。

   模型推理场景下，必须要有适配昇腾AI处理器的离线模型（*.om文件），请参见**4.2 模型构建**。

4. **开发应用**。

   a. AscendCL初始化，请参见**4.3 AscendCL初始化与去初始化**。

      使用AscendCL接口开发应用时，必须先调用**aclInit**接口进行AscendCL初始化，否则可能会导致后续系统内部资源初始化出错，进而导致其它业务异常。

b. 运行管理资源申请，请参见**4.4 运行管理资源申请与释放**。
c. 数据传输，请参见**4.5 数据传输**。
d. 执行模型推理。请参见**4.6 模型推理**。

   若需要处理模型推理的结果，还需要进行数据后处理，例如对于图片分类应用，通过数据后处理从推理结果中查找最大置信度的类别标识。

   模型推理结束后，需及时释放推理相关资源。
e. 所有数据处理结束后，需及时释放运行管理资源，请参见**4.4 运行管理资源申请与释放**。
f. 执行AscendCL去初始化，请参见**4.3 AscendCL初始化与去初始化**。

5. **编译运行应用**，包括编译代码、运行应用，请参见**8 应用调试**。

4.2 模型构建

对于开源框架的网络模型（如ONNX、TensorFlow等），不能直接在昇腾AI处理器上做推理，需要先使用ATC（Ascend Tensor Compiler）工具将开源框架的网络模型转换为适配昇腾AI处理器的离线模型（*.om文件）。

此处以ONNX框架的ResNet-50网络为例，说明如何使用ATC工具进行模型转换，详细说明请参见《**ATC工具使用指南（开放态）**》。

步骤1 以运行用户登录开发环境。

步骤2 执行模型转换。

执行以下命令，将原始模型转换为昇腾AI处理器能识别的*.om模型文件。请注意，执行命令的用户需具有命令中相关路径的可读、可写权限。以下命令中的“***<SAMPLE_DIR>***”请根据实际样例包的存放目录替换、“***<soc_version>***”请根据实际昇腾AI处理器版本替换。

```
cd <SAMPLE_DIR>/MyFirstApp_ONNX/model
wget https://obs-9be7.obs.cn-east-2.myhuaweicloud.com/003_Atc_Models/resnet50/resnet50.onnx
atc --model=resnet50.onnx --framework=5 --output=resnet50 --input_shape="actual_input_1:1,3,224,224"  --soc_version=<soc_version>
```

各参数的解释如下，详细约束说明请参见《**ATC工具使用指南（开放态）**》。

- --model：ResNet-50网络的模型文件的路径。
- --framework：原始框架类型。5表示ONNX。
- --output：resnet50.om模型文件的路径。请注意，记录保存该om模型文件的路径，后续开发应用时需要使用。
- --input_shape：模型输入数据的shape。
- --soc_version：昇腾AI处理器的版本。

说明

如果无法确定当前设备的soc_version，则在安装NPU驱动包的服务器执行**npu-smi info**命令进行查询，在查询到的“Name”前增加Ascend信息，例如“Name”对应取值为*xxxyy*，实际配置的soc_version值为Ascend*xxxyy*。

步骤3 （后续处理）如果想快速体验直接使用转换后的om离线模型文件进行推理，请准备好环境、om模型文件、符合模型输入要求的*.bin格式的输入数据，单击**Link**，获取**msame工具**，参考该工具配套的README，进行体验。

----结束

说明

- 如果模型转换时，提示有不支持的算子，请先参见《**TBE&AI CPU算子开发指南（开放态）**》先完成自定义算子，再重新转换模型。
- 如果模型转换时，提示有算子编译相关问题，但根据报错信息无法定位问题、需要联系技术支持时（您可以获取日志后单击**Link**联系技术支持。），则需设置DUMP_GE_GRAPH、DUMP_GRAPH_LEVEL环境变量，再重新模型转换，收集模型转换过程中各个阶段的图描述信息。关于环境变量以及图描述信息的说明，请参见《**ATC工具使用指南（开放态）**》中的“参考>dump图详细信息”。
- 如果模型的输入Shape是动态，关于模型构建、模型推理的说明请参见**7.7 模型动态Shape输入推理**。
- 如果现有网络不满足您的需求，您可以使用昇腾AI处理器支持的算子、调用Ascend Graph接口自行构建自己的网络，再编译成om离线模型文件。详细说明请参见《**Ascend Graph开发指南**》。

4.3 AscendCL 初始化与去初始化

本节介绍AscendCL初始化与去初始化的相关接口、注意事项，并给出示例代码。

基本原理

您必须调用**aclInit**接口**初始化**AscendCL，配置文件内容为json格式。

如果当前的默认配置已满足需求，无需修改，可向aclInit接口中传入NULL，或者可将配置文件配置为空json串（即配置文件中只有{}）。向**aclInit**接口中传入空指针的示例如下：

```
aclError ret = aclInit(NULL);
```

有初始化就有去初始化，在确定完成了AscendCL的所有调用之后，或者进程退出之前，需调用**aclFinalize**接口实现AscendCL**去初始化**。

示例代码

您可以从**10.7.1 样例介绍**中获取完整样例代码。

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 初始化
// 此处的..表示相对路径，相对可执行文件所在的目录
// 例如，编译出来的可执行文件存放在out目录下，此处的..就表示out目录的上一级目录
const char *aclConfigPath = "../src/acl.json";
aclError ret = aclInit(aclConfigPath);

// ......

// 去初始化
ret = aclFinalize();
// ......
```

4.4 运行管理资源申请与释放

本节介绍运行管理资源包括哪些、如何申请&释放这些资源，并给出示例代码。

开发应用时，应用程序中必须包含运行管理资源申请的代码逻辑，关于运行管理资源申请的接口调用流程，请先参见**3.2 AscendCL接口调用流程**了解整体流程，再查看本节中的资源申请&释放流程说明、示例代码。

基本原理

您需要按顺序依次**申请**如下运行管理资源：Device、Context、Stream，确保可以使用这些资源执行运算、管理任务。所有数据处理都结束后，需要按顺序依次**释放**运行管理资源：Stream、Context、Device。

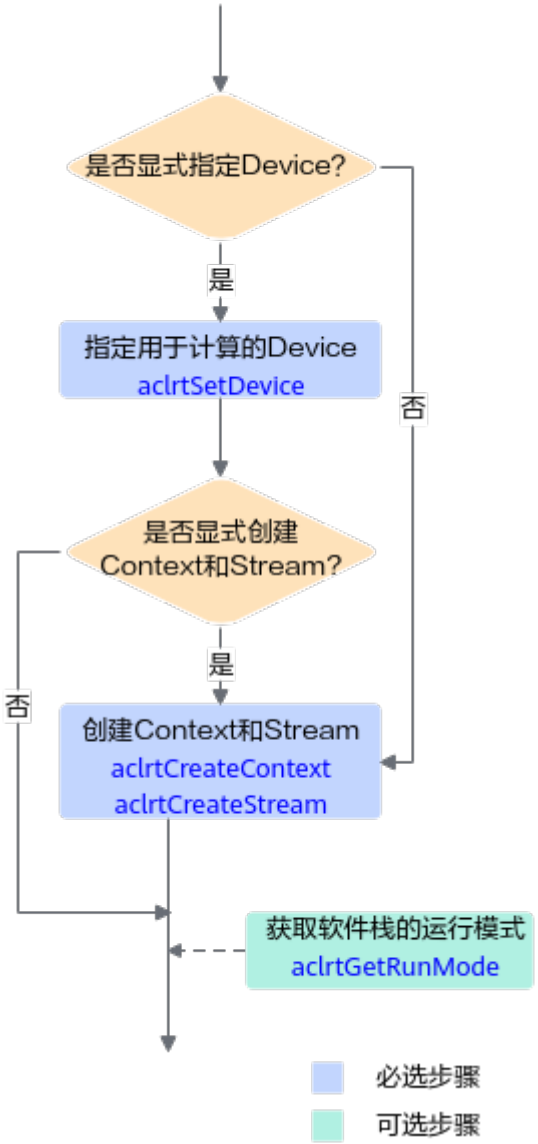
您需要按照Device、Context、Stream的顺序依次申请。其中，创建Context、Stream的方式分为隐式创建和显式创建，其**适用场景**有所不同：

- **隐式**创建Context和Stream：适合简单、无复杂交互逻辑的应用，但缺点在于，在多线程编程中，每个线程都使用默认Context或默认Stream，默认Stream中任务的执行顺序取决于操作系统线程调度的顺序。
- **显式**创建Context和Stream：**推荐显式**，适合大型、复杂交互逻辑的应用，且便于提高程序的可读性、可维护性。
- 关于单进程、单线程、单Stream场景如下所示：
  - 单进程：一个应用程序对应一个进程。
  - 单线程：不创建多个线程时，默认只有一个线程。
  - 单Stream：整个开发的过程中使用同一个Stream。

    对于同一个Stream中的异步任务，AscendCL会按照应用程序中任务的顺序执行任务，确保异步任务执行的顺序。
- 关于多线程、多Stream的场景请参见**7.9 Stream管理**。

运行管理资源申请流程

图 4-2 运行管理资源申请流程



关键接口的说明如下：

1. **申请**运行管理资源时，需按顺序依次申请：Device、Context、Stream。
   - 调用**aclrtSetDevice**接口**显式指定用于运算的Device**。
     - 调用**aclrtCreateContext**接口显式创建Context，调用**aclrtCreateStream**接口显式创建Stream。
     - 如果不显式创建Context和Stream，您可以使用**aclrtSetDevice**接口隐式创建的默认Context和默认Stream，但默认Context和默认Stream存在如下限制：
       - 一个Device对应一个默认Context，默认Context不能通过**aclrtDestroyContext**接口来释放。

    - 一个Device对应一个默认Stream，默认Stream不能通过**aclrtDestroyStream**接口来释放。默认Stream作为接口入参时，直接传NULL。
    - 默认Context、默认Stream，是在调用**aclrtResetDevice**接口后自动释放。
  - **隐式指定用于运算的Device**。

    调用**aclrtCreateContext**接口显式创建Context，调用**aclrtCreateStream**接口显式创建Stream。系统在显式创建Context时，系统内部会调用**aclrtSetDevice**接口指定运行的Device，Device ID通过**aclrtCreateContext**接口传入。
2. （可选）调用**aclrtGetRunMode**接口获取软件栈的运行模式，根据运行模式来判断后续的内存申请接口调用逻辑。

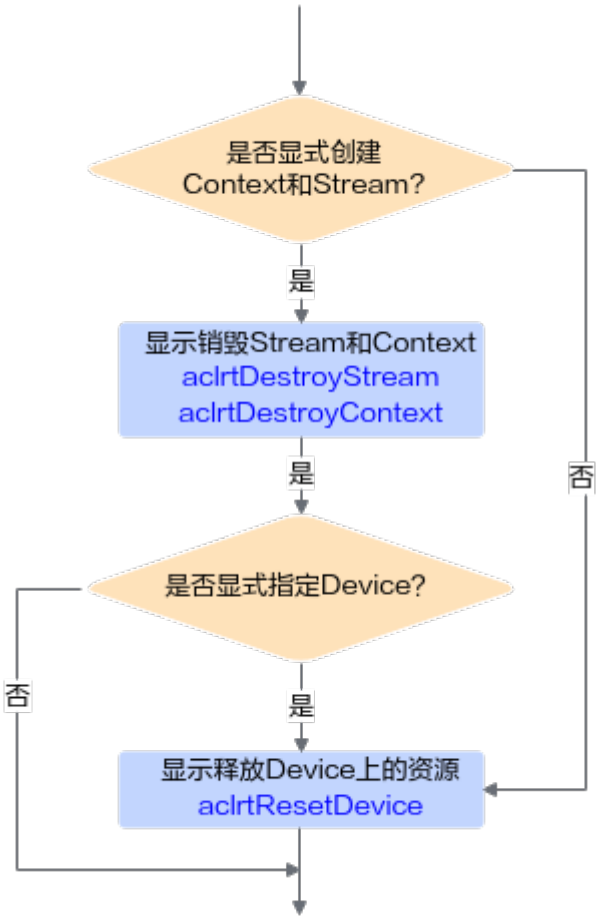
   如果查询结果为ACL_HOST，则数据传输时涉及申请Host上的内存。

   如果查询结果为ACL_DEVICE，则数据传输时仅需申请Device上的内存。

   数据传输的详细介绍请参见**4.5 数据传输**。

运行管理资源释放流程

图 4-3 运行管理资源释放流程



关键接口的说明如下：

- **释放**运行管理资源时，需按顺序依次释放：Stream、Context、Device。
- 显式创建Context和Stream时，需调用**aclrtDestroyStream**接口释放Stream，再调用**aclrtDestroyContext**接口释放Context。若显式调用**aclrtSetDevice**接口指定运算的Device时，还需调用**aclrtResetDevice**接口释放Device上的资源。
- 不显式创建Context和Stream时，仅需调用**aclrtResetDevice**接口释放Device上的资源。

示例代码

您可以从**10.7.1 样例介绍**中获取完整样例代码。

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 初始化变量
extern bool g_isDevice;

// =====运行管理资源申请=====
// 指定运算的Device
aclError ret = aclrtSetDevice(deviceId_);

// 显式创建一个Context，用于管理Stream对象
ret = aclrtCreateContext(&context_, deviceId_);

// 显式创建一个Stream
// 用于维护一些异步操作的执行顺序，确保按照应用程序中的代码调用顺序执行任务
ret = aclrtCreateStream(&stream_);

// 获取当前昇腾AI软件栈的运行模式，根据不同的运行模式，后续的接口调用方式不同

aclrtRunMode runMode;
ret = aclrtGetRunMode(&runMode);
g_isDevice = (runMode == ACL_DEVICE);
// =====运行管理资源申请=====

// ......

// =====运行管理资源释放=====
ret = aclrtDestroyStream(stream_);
ret = aclrtDestroyContext(context_);
ret = aclrtResetDevice(deviceId_);
// =====运行管理资源释放=====

// ......
```

4.5 数据传输

本节介绍数据传输的相关接口、注意事项，并给出示例代码。

接口调用流程

数据传输的关键接口调用流程如下：

1. **申请内存**。
   - Host上的内存，可以用C++标准库中的new、malloc接口申请内存，也可以使用AscendCL提供的**aclrtMallocHost**接口申请内存。
   - Device上的内存，使用AscendCL提供的**aclrtMalloc**或**aclrtMallocHost**接口申请内存。如果涉及媒体数据处理（例如，图片解码、缩放等）时，需使用**acldvppMalloc**或**hi_mpi_dvpp_malloc**接口申请内存。

2. **将数据读入内存**。

   由用户自行管理数据读入内存的实现逻辑。

3. 通过内存复制实现**数据传输**。

   数据传输可以通过内存复制的方式实现，分为同步内存复制、异步内存复制：

   - **同步**内存复制：调用**aclrtMemcpy**接口。
   - **异步**内存复制：调用**aclrtMemcpyAsync**接口，再调用**aclrtSynchronizeStream**接口实现Stream内任务的同步等待。
   - 对于Host内的数据传输、Device内的数据传输、Host与Device之间的数据传输，可以调用内存复制的接口实现，也可以直接通过指针传递数据。
   - 调用同步或异步内存复制接口时，支持以下类型的复制（可单击链接查看对应类型的内存复制示例代码）：
     - **Host内的数据传输**
     - **从Host到Device的数据传输**
     - **从Device到Host的数据传输**
     - **一个Device内的数据传输**

说明

如果当前版本支持多种**运行形态**，在这种场景下，若想实现相同的应用程序可支持在多种形态下运行，申请内存的方式不同，会影响数据传输时调用的接口：

- 若应用程序中区分申请Host内存或Device内存的接口，例如使用C++标准库的接口或**aclrtMallocHost**接口申请Host内存、使用**aclrtMalloc**接口申请Device内存时：

  需先调用**aclrtGetRunMode**接口获取软件栈的运行模式，当查询结果为ACL_HOST，则数据传输时涉及申请Host上的内存；当查询结果为ACL_DEVICE，则数据传输时不涉及申请Host上的内存，仅需申请Device上的内存。该种方式多一些代码逻辑的判断，不需要由用户处理Device上的内存对齐。在Device上运行应用的场景，该种方式少一些内存复制的步骤，性能较好。

- 若应用程序中不区分申请Host内存或Device内存的接口，统一使用**aclrtMallocHost**接口（该接口支持申请Host或Device内存），AscendCL内部会根据软件栈的运行模式自行判断运行时申请的是Host内存还是Device内存：

  无需调用**aclrtGetRunMode**接口获取软件栈的运行模式。该种方式代码逻辑相比前一种简单，但需由用户处理Device上的内存对齐。

Host 内的数据传输

当前支持调用aclrtMemcpy接口执行同步Host内的内存复制任务，不支持调用aclrtMemcpyAsync接口执行异步Host内的内存复制功能，若调用aclrtMemcpyAsync接口时选择ACL_MEMCPY_HOST_TO_HOST类型时，由于是异步接口，虽然接口调用成功，下发了内存复制任务，但在调用**aclrtSynchronizeStream**接口等待该任务执行时会返回失败。

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1. 申请内存
uint64_t size = 1 * 1024 * 1024;
void* hostPtrA = NULL;
void* hostPtrB = NULL;
aclrtMallocHost(&hostPtrA, size);
aclrtMallocHost(&hostPtrB, size);
```

```
// 2. 申请内存后，可向内存中读入数据，该自定义函数ReadFile由用户实现
ReadFile(fileName, hostPtrA, size);

// 3. 内存复制，可以选择同步
// 同步内存复制，hostPtrA表示Host上源内存地址指针，hostPtrB表示Host上目的内存地址指针，size表示内存大小
aclrtMemcpy(hostPtrB, size, hostPtrA, size, ACL_MEMCPY_HOST_TO_HOST);

// 4. 使用完内存中的数据后，需及时释放资源
aclrtFreeHost(hostPtrA);
aclrtFreeHost(hostPtrB);
// ......
```

从 Host 到 Device 的数据传输

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

- 同步内存复制

```
// 1. 申请内存
uint64_t size = 1 * 1024 * 1024;
void* hostPtrA = NULL;
void* devPtrB = NULL;
aclrtMallocHost(&hostPtrA, size);
aclrtMalloc(&devPtrB, size, ACL_MEM_MALLOC_NORMAL_ONLY);

// 2. 申请内存后，可向内存中读入数据，该自定义函数ReadFile由用户实现
ReadFile(fileName, hostPtrA, size);

// 3. 内存复制，可以选择同步
// 同步内存复制，hostPtrA表示Host上源内存地址指针，devPtrB表示Device上目的内存地址指针，size表示内存大小
aclrtMemcpy(devPtrB, size, hostPtrA, size, ACL_MEMCPY_HOST_TO_DEVICE);

// 4. 使用完内存中的数据后，需及时释放资源
aclrtFreeHost(hostPtrA);
aclrtFree(devPtrB);

// ......
```

- 异步内存复制

```
// 1. 申请内存
uint64_t size = 1 * 1024 * 1024;
void* hostAddr = NULL;
void* devAddr = NULL;
aclrtMallocHost(&hostAddr, size + 64);
aclrtMalloc(&devAddr, size, ACL_MEM_MALLOC_NORMAL_ONLY);

// 2. 异步内存复制
aclrtStream stream = NULL;
aclrtCreateStream(&stream);

// 申请内存后，可向内存中读入数据，该自定义函数ReadFile由用户实现
ReadFile(fileName, hostAddr, size);
aclrtMemcpyAsync(devAddr, size, hostAddr, size, ACL_MEMCPY_HOST_TO_DEVICE, stream);
aclrtSynchronizeStream(stream);

// 3. 释放资源
aclrtDestroyStream(stream);
aclrtFreeHost(hostAddr);
aclrtFree(devAddr);

// ......
```

从 Device 到 Host 的数据传输

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

- 同步内存复制

```
// 1. 申请内存
uint64_t size = 1 * 1024 * 1024;
void* devPtrA = NULL;
void* hostPtrB = NULL;
aclrtMalloc(&devPtrA, size, ACL_MEM_MALLOC_NORMAL_ONLY);
aclrtMallocHost(&hostPtrB, size);

// 2. 申请内存后，可向内存中读入数据，该自定义函数ReadFile由用户实现
ReadFile(fileName, devPtrA, size);

// 3. 内存复制，可以选择同步
// 同步内存复制，devPtrA表示Device上源内存地址指针，hostPtrB表示Host上目的内存地址指针，size表示内存大小
aclrtMemcpy(hostPtrB, size, devPtrA, size, ACL_MEMCPY_DEVICE_TO_HOST);

// 4. 使用完内存中的数据后，需及时释放资源
aclrtFree(devPtrA);
aclrtFreeHost(hostPtrB);

// ......
```

- 异步内存复制

```
// 1. 申请内存
uint64_t size = 1 * 1024 * 1024;
void* hostAddr = NULL;
void* devAddr = NULL;
aclrtMallocHost(&hostAddr, size + 64);
aclrtMalloc(&devAddr, size, ACL_MEM_MALLOC_NORMAL_ONLY);

// 2. 申请内存后，可向内存中读入数据，该自定义函数ReadFile由用户实现
ReadFile(fileName, devAddr, size);

// 3. 异步内存复制
aclrtStream stream = NULL;
aclrtCreateStream(&stream);
aclrtMemcpyAsync(hostAddr, size, devAddr, size, ACL_MEMCPY_DEVICE_TO_HOST, stream);
aclrtSynchronizeStream(stream);

// 4. 释放资源
aclrtDestroyStream(stream);
aclrtFreeHost(hostAddr);
aclrtFree(devAddr);

// ......
```

一个 Device 内的数据传输

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1. 申请内存
uint64_t size = 1 * 1024 * 1024;
void* devPtrA = NULL;
void* devPtrB = NULL;
aclrtMalloc(&devPtrA, size, ACL_MEM_MALLOC_NORMAL_ONLY);
aclrtMalloc(&devPtrB, size, ACL_MEM_MALLOC_NORMAL_ONLY);

// 2. 申请内存后，可向内存中读入数据，该自定义函数ReadFile由用户实现
ReadFile(fileName, devPtrA, size);

// 3. 内存复制，可以选择同步或异步
```

```
// 同步内存复制，devPtrA表示Device上源内存地址指针，devPtrB表示Device上目的内存地址指针，size表示内存大小
aclrtMemcpy(devPtrB, size, devPtrA, size, ACL_MEMCPY_DEVICE_TO_DEVICE);

// 异步内存复制
aclrtStream stream = NULL;
aclrtCreateStream(&stream);
aclrtMemcpyAsync(devPtrB, size, devPtrA, size, ACL_MEMCPY_DEVICE_TO_DEVICE, stream);
aclrtSynchronizeStream(stream);

// 4. 使用完内存中的数据后，需及时释放资源
aclrtDestroyStream(stream);
aclrtFree(devPtrA);
aclrtFree(devPtrB);

// ......
```

4.6 模型推理

4.6.1 模型加载

本节介绍如何加载模型，为模型执行做准备。

接口调用流程

开发应用时，如果涉及整网模型推理，则应用程序中必须包含模型加载的代码逻辑，关于模型加载的接口调用流程，请先参见**3.2 AscendCL接口调用流程**了解整体流程，再查看本节中的流程说明。本节描述的是**整网模型加载**的接口调用流程，对于算子模型加载与执行的详细说明请参见**6.2 单算子调用流程**。

AscendCL提供**两套模型加载的接口**，用户可根据编程习惯、使用场景选择对应的模型加载接口：

- 如**图4-4**所示，针对不同的加载方式（从文件加载、从内存加载等），只需**设置接口中的配置参数**，适用各种加载方式，但涉及**多个接口配合使用**，分别用于创建配置对象、设置对象中的属性值、加载模型。
- 如**图4-5**所示，根据不同的加载方式（从文件加载、从内存加载等）**选择不同的接口**，操作相对简单，但需要**记住各种方式的加载接口**。

图 4-4 模型加载流程（通过接口中的配置参数区分加载方式）

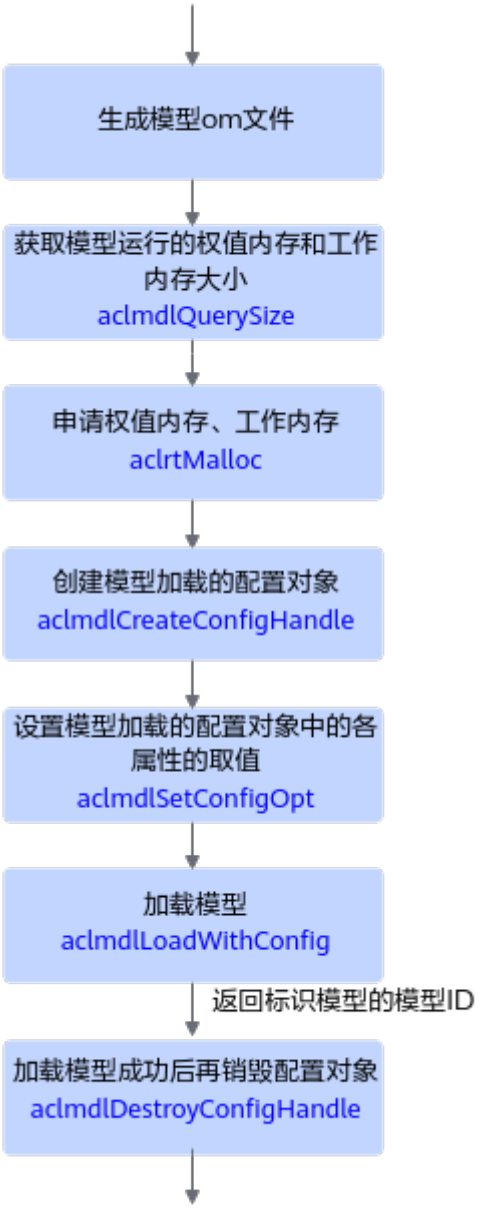
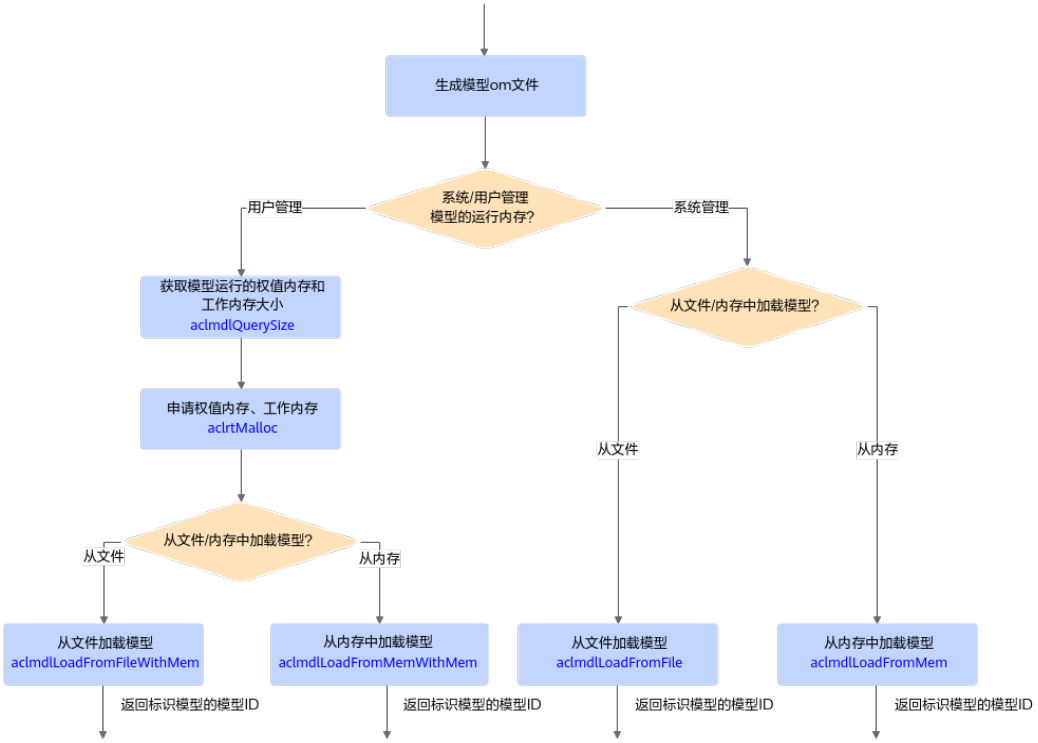


图 4-5 模型加载流程（通过不同接口区分加载方式）



关键接口的说明如下：

- **在模型加载前**，需要先**构建**出适配昇腾AI处理器的离线模型（***.om文件**），构建方式请参见**4.2 模型构建**。
- 当由用户管理内存时，为确保内存不浪费，在申请工作内存、权值内存前，需要调用**aclmdlQuerySize**接口**查询**模型运行时所需**工作内存**、**权值内存的大小**。

  如果模型输入数据的Shape不确定，则不能调用**aclmdlQuerySize**接口查询内存大小，在加载模型时，就无法由用户管理内存，因此需选择由系统管理内存的模型加载接口（例如，**aclmdlLoadFromFile**、**aclmdlLoadFromMem**）。
- 支持以下方式**加载模型**，模型加载成功后，返回标识模型的模型ID：
  - 使用**aclmdlSetConfigOpt**接口、**aclmdlLoadWithConfig**接口时，是通过配置对象中的属性来区分，在加载模型时是从文件加载，还是从内存加载，以及内存是由系统内部管理，还是由用户管理。
  - 使用以下接口时，是从使用的接口上区分从文件加载，还是从内存加载，以及内存是由系统内部管理，还是由用户管理。
    - **aclmdlLoadFromFile**：从文件加载离线模型数据，由系统内部管理内存。
    - **aclmdlLoadFromMem**：从内存加载离线模型数据，由系统内部管理内存。
    - **aclmdlLoadFromFileWithMem**：从文件加载离线模型数据，由用户自行管理模型运行的内存（包括工作内存和权值内存，工作内存用于模型执行过程中的临时数据，权值内存用于存放权值数据）。
    - **aclmdlLoadFromMemWithMem**：从内存加载离线模型数据，由用户自行管理模型运行的内存（包括工作内存和权值内存）。

示例代码

模型加载成功，会返回标识模型的ID，在**4.6.2 模型执行**时需要使用该ID。

您可以从**10.7.1 样例介绍**中获取完整样例代码。

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1.初始化变量。
// 此处的..表示相对路径，相对可执行文件所在的目录
// 例如，编译出来的可执行文件存放在out目录下，此处的..就表示out目录的上一级目录
const char* omModelPath = "../model/resnet50.om"
// ......

// 2.根据模型文件获取模型执行时所需的权值内存大小、工作内存大小。
aclError ret = aclmdlQuerySize(omModelPath, &modelMemSize_, &modelWeightSize_);

// 3.根据工作内存大小，申请Device上模型执行的工作内存。
ret = aclrtMalloc(&modelMemPtr_, modelMemSize_, ACL_MEM_MALLOC_HUGE_FIRST);

// 4.根据权值内存的大小，申请Device上模型执行的权值内存。
ret = aclrtMalloc(&modelWeightPtr_, modelWeightSize_, ACL_MEM_MALLOC_HUGE_FIRST);

// 5.加载离线模型文件，由用户自行管理模型运行的内存(包括权值内存、工作内存)。
// 模型加载成功，返回标识模型的ID。
ret = aclmdlLoadFromFileWithMem(modelPath, &modelId_, modelMemPtr_, modelMemSize_,
modelWeightPtr_, modelWeightSize_);

// ......
```

4.6.2 模型执行

本节结合接口调用流程、示例代码介绍模型执行前需要准备哪些数据、模型执行接口以及模型执行之后需要释放哪些资源。

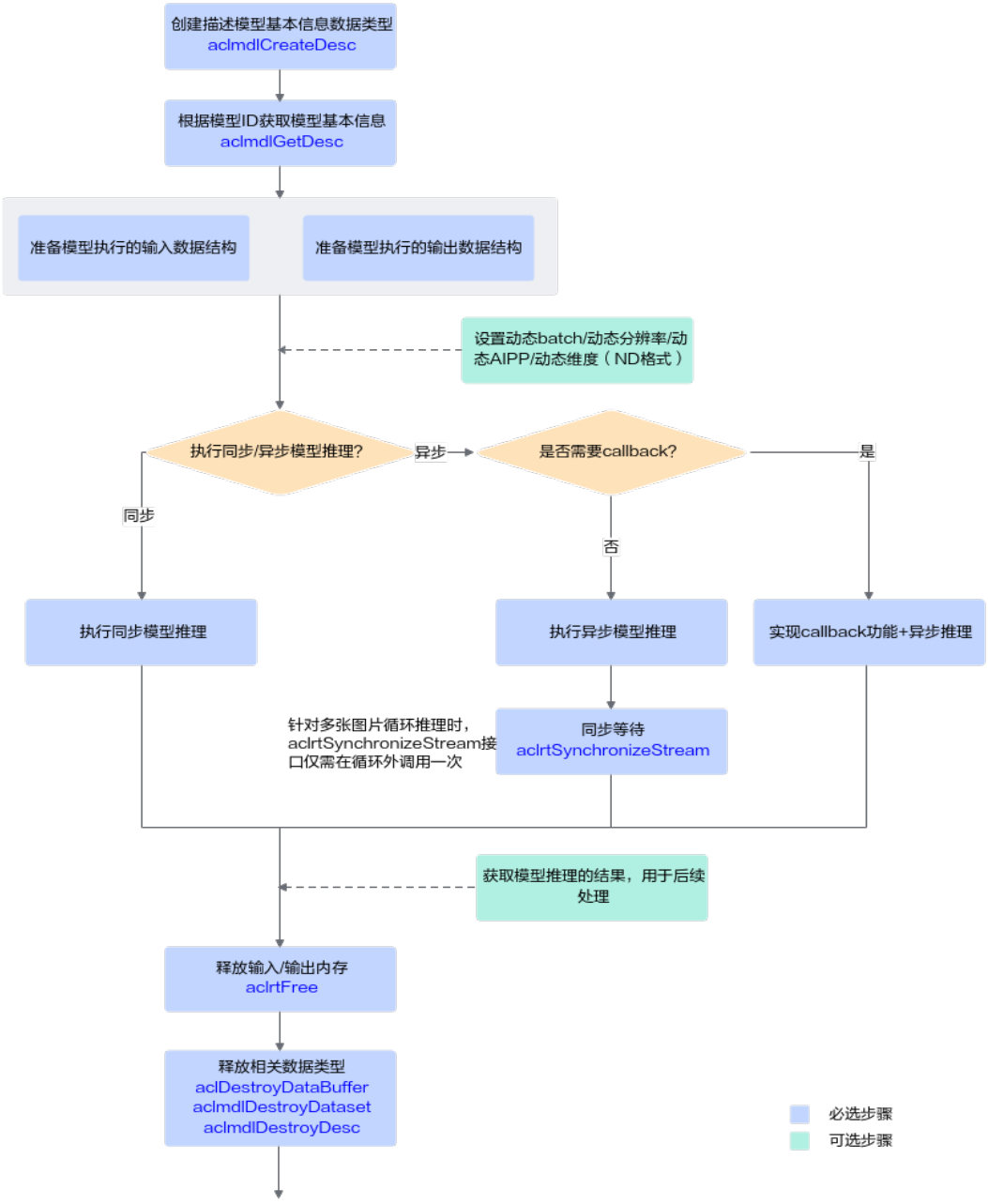
基本原理

开发应用时，如果涉及整网模型推理，则应用程序中必须包含模型执行的代码逻辑，关于模型执行的接口调用流程，请先参见**3.2 AscendCL接口调用流程**了解整体流程，再查看本节中的流程说明。本节描述的是**整网模型执行**的接口调用流程，对于算子模型加载与执行的详细说明请参见**6 单算子调用**。

- **在模型加载之后，模型执行之前**，需要准备输入、输出数据结构，将输入数据传输到模型输入数据结构的对应内存中。
- **模型执行结束后**，若无需使用输入数据、aclmdlDesc类型、aclmdlDataset类型、aclDataBuffer类型等相关资源，需及时释放内存、销毁对应的数据类型，防止内存异常。模型可能存在多个输入、多个输出，每个输入/输出的内存地址、内存大小用aclDataBuffer类型的数据来描述，针对每个输入/输出，需调用**aclDestroyDataBuffer**接口销毁相应的aclDataBuffer类型，并调用**aclrtFree**接口释放内存中的数据。

模型执行流程

图 4-6 基本的模型推理流程



关键接口的说明如下：

1. 调用**aclmdlCreateDesc**接口**创建描述模型基本信息**的数据类型。
2. 调用**aclmdlGetDesc**接口根据**4.6.1 模型加载**中返回的模型ID**获取模型基本信息**。
3. **准备模型执行的输入、输出数据结构**，具体流程，请参见**准备模型执行的输入/输出数据结构**。

   如果模型的输入涉及**动态Batch**、**动态分辨率**、**动态AIPP**、**动态维度（ND格式）**等特性，请参见**7.7 模型动态Shape输入推理**、**7.8 模型动态AIPP推理**。

4. **执行模型推理**。

   对于固定的多Batch场景，需要满足batch size后，才能将输入数据发送给模型进行推理。不满足batch size时，用户需根据自己的实际场景处理。

   当前系统支持模型的同步推理和异步推理：

   - 同步推理

     调用**aclmdlExecute**接口执行同步推理。

   - 异步推理

     调用**aclmdlExecuteAsync**接口执行异步推理。

     但对于异步接口，还需调用**aclrtSynchronizeStream**接口阻塞应用程序运行，直到指定Stream中的所有任务都完成。

     异步推理的详细介绍，请参见**7.6 异步模型推理**。

5. **获取模型推理的结果**，用于后续处理。

   - 对于同步推理，直接获取模型推理的输出数据即可。
   - 对于异步推理，在实现Callback功能时，在回调函数内获取模型推理的结果，供后续使用。

6. **释放内存**。

   调用**aclrtFree**接口释放Device上的内存。

7. **释放相关数据类型的数据**。

   在模型推理结束后，需依次调用**aclDestroyDataBuffer**接口、**aclmdlDestroyDataset**接口及时释放描述模型输入、输出数据类型的数据。如果存在多个输入、输出，需调用多次**aclDestroyDataBuffer**接口。

准备模型执行的输入/输出数据结构

AscendCL提供了以下数据类型来描述模型、描述其输入输出以及存放数据的内存，在模型执行前，需要构造好这些数据类型，作为模型执行的输入：

- 使用**aclmdlDesc**类型的数据描述模型基本信息（例如输入/输出的个数、名称、数据类型、Format、维度信息等）。

  模型加载成功后，用户可根据模型的ID，调用**aclmdlGetDesc**接口获取该模型的描述信息，进而从模型的描述信息中获取模型输入/输出的个数、内存大小、维度信息、Format、数据类型等信息，可参见**aclmdlDesc**类型下的操作接口。

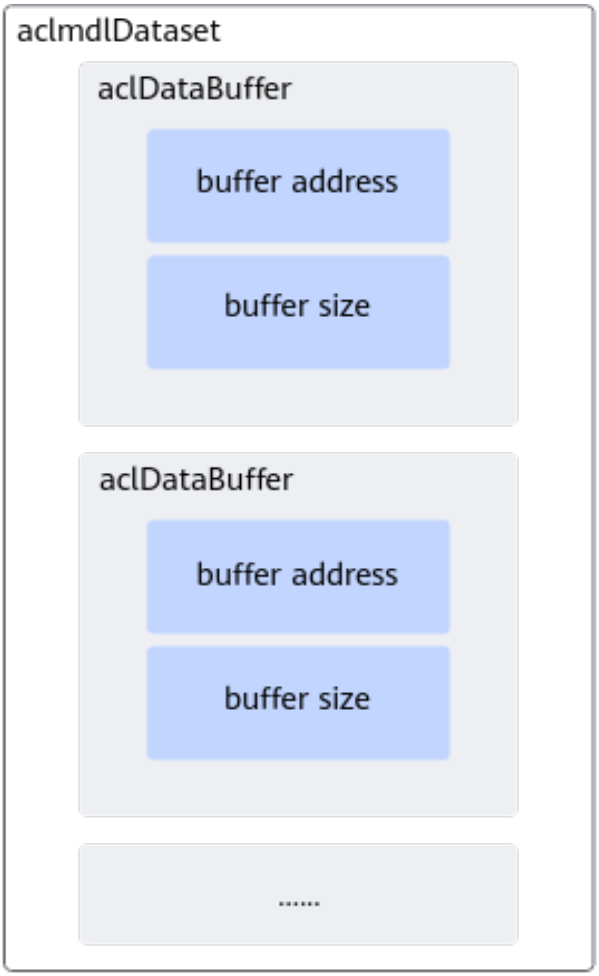
- 使用**aclmdlDataset**类型的数据描述模型的输入/输出数据，模型可能存在多个输入、多个输出。

  调用**aclmdlDataset**类型下的操作接口添加aclDataBuffer类型的数据、获取aclDataBuffer的个数等。

- 每个输入/输出的内存地址、内存大小用**aclDataBuffer**类型的数据来描述。

  调用**aclDataBuffer**类型下的操作接口获取内存地址、内存大小等。

图 4-7 aclmdlDataset 类型与 aclDataBuffer 类型的关系



了解相关的数据类型后，可以使用这些数据类型的操作接口准备模型的输入、输出数据结构，如下图所示。

图 4-8 模型执行的输入/输出数据结构的准备流程

关键说明如下：

- 模型存在多个输入、输出时，用户可调用**aclmdlGetNumInputs**、**aclmdlGetNumOutputs**接口获取输入、输出的个数。
- 模型每个输入、输出所需的内存大小，用户可调用**aclmdlGetInputSizeByIndex**、**aclmdlGetOutputSizeByIndex**接口获取。

  如果模型的输入涉及**动态Batch**、**动态分辨率**、**动态维度（ND格式）**等特性，输入tensor数据的Shape支持多种档位，在模型执行前才能确定，因此该输入所需的内存大小建议用户调用**aclmdlGetInputSizeByIndex**接口获取，该接口获取的是最大档位的内存，确保内存够用。

- 模型存在多个输入、输出时，用户在向aclmdlDataset中添加aclDataBuffer时，为避免顺序出错，可以先调用**aclmdlGetInputNameByIndex**、**aclmdlGetOutputNameByIndex**接口获取输入、输出的名称，根据输入、输出名称所对应的index的顺序添加。

示例代码

此处的示例代码是处理图片分类模型的输出结果，屏显每张图片的top5置信度的类别编号。用户可根据实际需求，自行实现模型推理输出数据的处理逻辑。

您可以从**10.7.1 样例介绍**中获取完整样例代码。

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1 根据模型的ID，获取该模型的描述信息。
// modelDesc_为aclmdlDesc类型。
modelDesc_ = aclmdlCreateDesc();
ret = aclmdlGetDesc(modelDesc_, modelId_);

// 2 准备模型推理的输入数据结构
// (1)申请输入内存
size_t modelInputSize;
void *modelInputBuffer = nullptr;
// 当前示例代码中的模型只有一个输入，所以index为0，如果模型有多个输入，则需要先调用
aclmdlGetNumInputs接口获取模型输入的数量
modelInputSize = aclmdlGetInputSizeByIndex(modelDesc_, 0);
aclRet = aclrtMalloc(&modelInputBuffer, modelInputSize, ACL_MEM_MALLOC_NORMAL_ONLY);

// (2)准备模型的输入数据结构
// 创建aclmdlDataset类型的数据，描述模型推理的输入，input_为aclmdlDataset类型
input_ = aclmdlCreateDataset();
aclDataBuffer *inputData = aclCreateDataBuffer(modelInputBuffer, modelInputSize);
ret = aclmdlAddDatasetBuffer(input_, inputData);

// 3 准备模型推理的输出数据结构
// (1)创建aclmdlDataset类型的数据，描述模型推理的输出，output_为aclmdlDataset类型
output_ = aclmdlCreateDataset();

// (2)获取模型的输出个数.
size_t outputSize = aclmdlGetNumOutputs(modelDesc_);

// (3)循环为每个输出申请内存，并将每个输出添加到aclmdlDataset类型的数据中.
for (size_t i = 0; i < outputSize; ++i) {
    size_t buffer_size = aclmdlGetOutputSizeByIndex(modelDesc_, i);
    void *outputBuffer = nullptr;
    aclError ret = aclrtMalloc(&outputBuffer, buffer_size, ACL_MEM_MALLOC_NORMAL_ONLY);
    aclDataBuffer* outputData = aclCreateDataBuffer(outputBuffer, buffer_size);
    ret = aclmdlAddDatasetBuffer(output_, outputData);
    }

// 4 模型执行
string testFile[] = {
        "../data/dog1_1024_683.bin",
        "../data/dog2_1024_683.bin"
    };

for (size_t index = 0; index < sizeof(testFile) / sizeof(testFile[0]); ++index) {
    // 4.1 自定义函数ReadBinFile，调用C++标准库std::ifstream中的函数读取图片文件，输出图片文件占用的内存
大小inputBuffSize以及图片文件存放在内存中的地址inputBuff
    void *inputBuff = nullptr;
    uint32_t inputBuffSize = 0;
    auto ret = Utils::ReadBinFile(fileName, inputBuff, inputBuffSize);

    // 4.2 准备模型推理的输入数据
    // 在申请运行管理资源时调用aclrtGetRunMode接口获取软件栈的运行模式
    // 如果运行模式为ACL_DEVICE，则g_isDevice参数值为true，表示软件栈运行在Device侧，无需传输图片数据
```

```
或在Device内传输数据；否则，需要调用内存复制接口将数据传输到Device
    if (!g_isDevice) {
        // if app is running in host, need copy data from host to device
        // modelInputBuffer、modelInputSize分别表示模型推理输入数据的内存地址、内存大小，在输入/输出数
据结构准备时申请该内存
        aclError aclRet = aclrtMemcpy(modelInputBuffer, modelInputSize, inputBuff, inputBuffSize,
ACL_MEMCPY_HOST_TO_DEVICE);
        (void)aclrtFreeHost(inputBuff);
    } else { // app is running in device
        aclError aclRet = aclrtMemcpy(modelInputBuffer, modelInputSize, inputBuff, inputBuffSize,
ACL_MEMCPY_DEVICE_TO_DEVICE);
        (void)aclrtFree(inputBuff);
    }

    // 4.3 执行模型推理
    // modelId_表示模型ID，在模型加载成功后，会返回标识模型的ID
    // input_、output_分别表示模型推理的输入、输出数据，在准备模型推理的输入、输出数据结构时已定义
    aclError ret = aclmdlExecute(modelId_, input_, output_)


    // 处理模型推理的输出数据，输出top5置信度的类别编号
    // output_表示模型执行的输出
    for (size_t i = 0; i < aclmdlGetDatasetNumBuffers(output_); ++i) {
    // 获取每个输出的内存地址和内存大小
        aclDataBuffer* dataBuffer = aclmdlGetDatasetBuffer(output_, i);
        void* data = aclGetDataBufferAddr(dataBuffer);

        size_t len = aclGetDataBufferSizeV2(dataBuffer);

        // 将内存中的数据转换为float类型
        float *outData = NULL;
        outData = reinterpret_cast<float*>(data);

        // 屏显每张图片的top5置信度的类别编号
        map<float, int, greater<float> > resultMap;
        for (int j = 0; j < len / sizeof(float); ++j) {
            resultMap[*outData] = j;
            outData++;
        }
        int cnt = 0;
        for (auto it = resultMap.begin(); it != resultMap.end(); ++it) {
            // print top 5
            if (++cnt > 5) {
                break;
            }

            INFO_LOG("top %d: index[%d] value[%lf]", cnt, it->second, it->first);
    }
}

// 5 释放模型推理的输入、输出资源
// 释放输入资源，包括数据结构和内存
for (size_t i = 0; i < aclmdlGetDatasetNumBuffers(input_); ++i) {
        aclDataBuffer *dataBuffer = aclmdlGetDatasetBuffer(input_, i);
        (void)aclDestroyDataBuffer(dataBuffer);
}
(void)aclmdlDestroyDataset(input_);
input_ = nullptr;
aclrtFree(modelInputBuffer);

// 释放输出资源，包括数据结构和内存
for (size_t i = 0; i < aclmdlGetDatasetNumBuffers(output_); ++i) {
    aclDataBuffer* dataBuffer = aclmdlGetDatasetBuffer(output_, i);
    void* data = aclGetDataBufferAddr(dataBuffer);
    (void)aclrtFree(data);
    (void)aclDestroyDataBuffer(dataBuffer);
}
```

```
(void)aclmdlDestroyDataset(output_);
output_ = nullptr;
```

4.6.3 模型卸载

模型执行结束后，需及时卸载模型，释放模型资源。

关于模型卸载的接口调用流程，请先参见**3.2 AscendCL接口调用流程**了解整体流程，再查看本节中的流程说明。

基本原理

在模型推理结束后，还需要通过**aclmdlUnload**接口卸载模型，并销毁aclmdlDesc类型的模型描述信息、释放模型运行的工作内存和权值内存。

示例代码

```
// 1. 卸载模型
aclError ret = aclmdlUnload(modelId_);

// 2. 释放模型描述信息
if (modelDesc_ != nullptr) {
    (void)aclmdlDestroyDesc(modelDesc_);
    modelDesc_ = nullptr;
}

// 3. 释放模型运行的工作内存
if (modelWorkPtr_ != nullptr) {
    (void)aclrtFree(modelWorkPtr_);
    modelWorkPtr_ = nullptr;
    modelWorkSize_ = 0;
}

// 4. 释放模型运行的权值内存
if (modelWeightPtr_ != nullptr) {
    (void)aclrtFree(modelWeightPtr_);
    modelWeightPtr_ = nullptr;
    modelWeightSize_ = 0;
}
```

5 媒体数据处理（含图像/视频等）

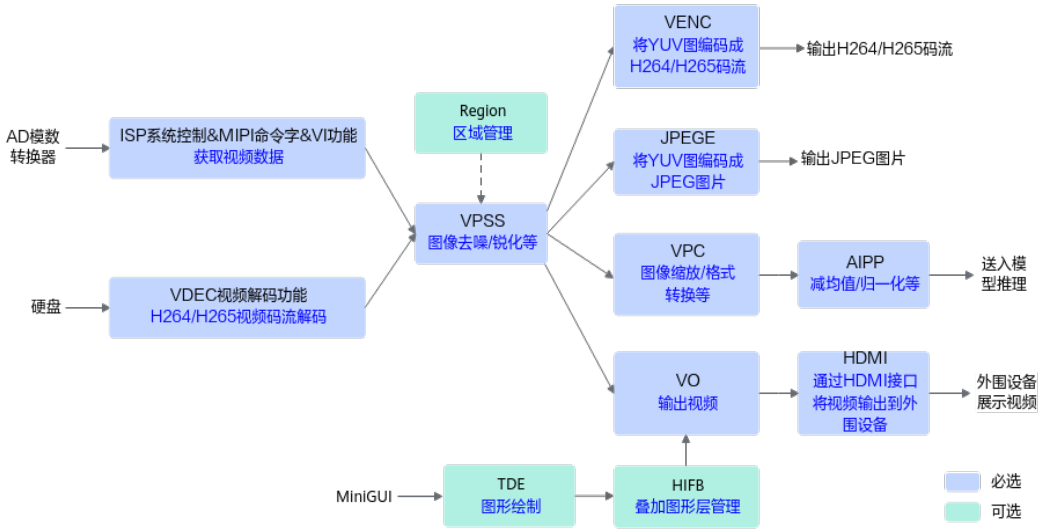
媒体数据处理基础知识
本章主要介绍图像/视频/音频数据处理的具体功能、接口调用流程以及示例代码。
媒体数据处理V1与V2版本的差别
媒体数据处理接口有V1、V2两套，本节介绍两套接口的共同点、差异点。
各版本功能支持度说明
DVPP图像/视频处理（媒体数据处理V1）
DVPP图像/视频处理（媒体数据处理V2）
精度提升建议
高性能编程建议

5.1 媒体数据处理基础知识

本章主要介绍图像/视频/音频数据处理的具体功能、接口调用流程以及示例代码。

图像/视频/音频数据处理的典型功能介绍

图 5-1 图像/视频数据处理



版权所有 © 华为技术有限公司

各功能的介绍如下表所示，昇腾AI处理器对这些功能的支持度请参见**5.3 各版本功能支持度说明**（AIPP当前各版本均支持）。

功能	子功能模块	描述
获取视频数据	**ISP**（Image Signal Processing）系统控制	系统控制部分用于注册3A算法、注册Sensor驱动、初始化ISP firmware、运行ISP firmware、退出ISP firmware、配置ISP属性等功能。
	MIPI Rx ioctl命令字	MIPI Rx是一个支持多种差分视频输入接口的采集单元，通过combo-PHY接收MIPI/LVDS/sub-LVDS/HiSPi接口的数据，通过不同的功能模式配置，MIPI Rx可以支持多种速度和分辨率的数据传输需求，支持多种外部输入设备。
	VI（Vedio Input）	VI模块捕获视频图像，可对其做裁剪、防抖、颜色优化、亮度优化、噪声去除等处理，并输出YUV或RAW格式的图像数据。
展示视频数据	**VO**（Vedio Output）	VO模块接收VPSS处理后的输出图像，可进行播放控制等处理，最后按用户配置的输出协议（当前仅支持HDMI）输出给外围视频设备。 VO可配合TDE（Two Dimensional Engine）模块、HIFB（Hisilicon Framebuffer）模块，利用硬件分别进行图形绘制、叠加图形层管理。
	HDMI（High Definition Multimedia Interfac）	HDMI是全数字化影像和声音发送接口，可以发送未压缩的音频及视频信号。
	TDE（Two Dimensional Engine）	TDE是图形二维加速引擎，它利用硬件为 OSD（On Screen Display）和 GUI（Graphics User Interface）提供快速的图形绘制功能，主要有快速拷贝、快速色彩填充、模式填充（当前仅支持Alpha Blending操作）。
	HIFB（Hisilicon Framebuffer）	HIFB用于管理叠加图形层，它不仅提供Linux Framebuffer的基本功能，还在Linux Framebuffer的基础上增加图层显示起始位置修改、层间Alpha等扩展功能。
区域管理	Region	叠加在视频上的OSD (On Screen Display)和遮挡在视频上的色块统称为区域。**区域管理模块**，用于统一管理这些区域资源，用于在视频上显示一些特定信息（如通道号、时间戳等）、或在视频中填充色块用于遮挡，当前该功能需配合VPSS一起使用。
图像/视频数据处理	**VPSS**（Video Process Sub-System）	VPSS模块支持对输入图像进行统一预处理，如去噪、去隔行、裁剪等，然后再对各通道分别进行处理，如缩放、加边框等。

功能	子功能模块	描述
	AIPP（Artificial Intelligence Pre-Processing）	AIPP人工智能预处理，在AI Core上完成数据预处理，主要功能包括改变图像尺寸（抠图、填充等）、色域转换（转换图像格式）、减均值/乘系数（改变图像像素）等。 AIPP区分为静态AIPP和动态AIPP。您只能选择静态AIPP或动态AIPP中的一种来处理图片，不能同时配置静态AIPP和动态AIPP两种方式。 ● 静态AIPP：模型转换时设置AIPP模式为静态，同时设置AIPP参数，模型生成后，AIPP参数值被保存在离线模型（*.om）中，每次模型推理过程采用固定的AIPP预处理参数（无法修改）。 如果使用静态AIPP方式，多Batch情况下共用同一份AIPP参数，AIPP参数值在使用ATC工具进行模型转换时设置，ATC工具的详细说明请《**ATC工具使用指南（开放态）**》。 ● 动态AIPP：模型转换时仅设置AIPP模式为动态，每次模型推理前，根据需求，在执行模型前设置动态AIPP参数值，然后在模型执行时可使用不同的AIPP参数。如果使用动态AIPP方式，多Batch可使用不同的AIPP参数，各Batch所使用的AIPP参数值通过AscendCL提供的接口来设置，请参见**7.8 模型动态AIPP推理**中的介绍。
	DVPP（Digital Vision Pre-Processing）	DVPP是昇腾AI处理器内置的图像处理单元，通过AscendCL媒体数据处理接口提供强大的媒体处理硬加速能力，主要功能包括以下功能： ● VPC（Vision Preprocessing Core）：处理YUV、RGB等格式的图片，包括缩放、抠图、图像金字塔、色域转换等。 ● JPEGD（JPEG Decoder）：JPEG压缩格式-->YUV格式的图片解码。 ● JPEGE（JPEG Encoder）：YUV格式-->JPEG压缩格式的图片编码。 ● VDEC（Video Decoder）：H264/H265格式-->YUV/RGB格式的视频码流解码。 ● VENC（Video Encoder）：YUV420SP格式-->H264/H265格式的视频码流编码。 ● PNGD（PNG Decoder）：PNG格式-->RGB格式的图片解码。 **说明** AIPP、DVPP可以分开独立使用，也可以组合使用。组合使用场景下，一般先使用DVPP对图片/视频进行解码、抠图、缩放等基本处理，但由于DVPP硬件上的约束，DVPP处理后的图片格式、分辨率有可能不满足模型的要求，因此还需要再经过AIPP进一步做色域转换、抠图、填充等处理。 例如，在Atlas 200/300/500 推理产品和Atlas 训练系列产品上，由于DVPP视频解码仅支持输出YUV格式的图片，如果模型需要RGB格式的图片，则需要再经过AIPP做色域转换的处理。

功能	子功能模块	描述
音频数据获取和输出	**AI**（Audio Input）	AI模块捕获音频数据 。
	AO（Audio Output）	通过ADEC模块解码后的音频数据，AO模块支持播放音频。
音频数据编解码	**AENC**（Audio Encoder）	通过AI模块获取的音频数据，AENC模块支持对其进行编码，输出音频码流。
	ADEC（Audio Decoder）	ADEC支持解码G.711a、G.711u等协议的音频码流，再通过AO模块播放音频。

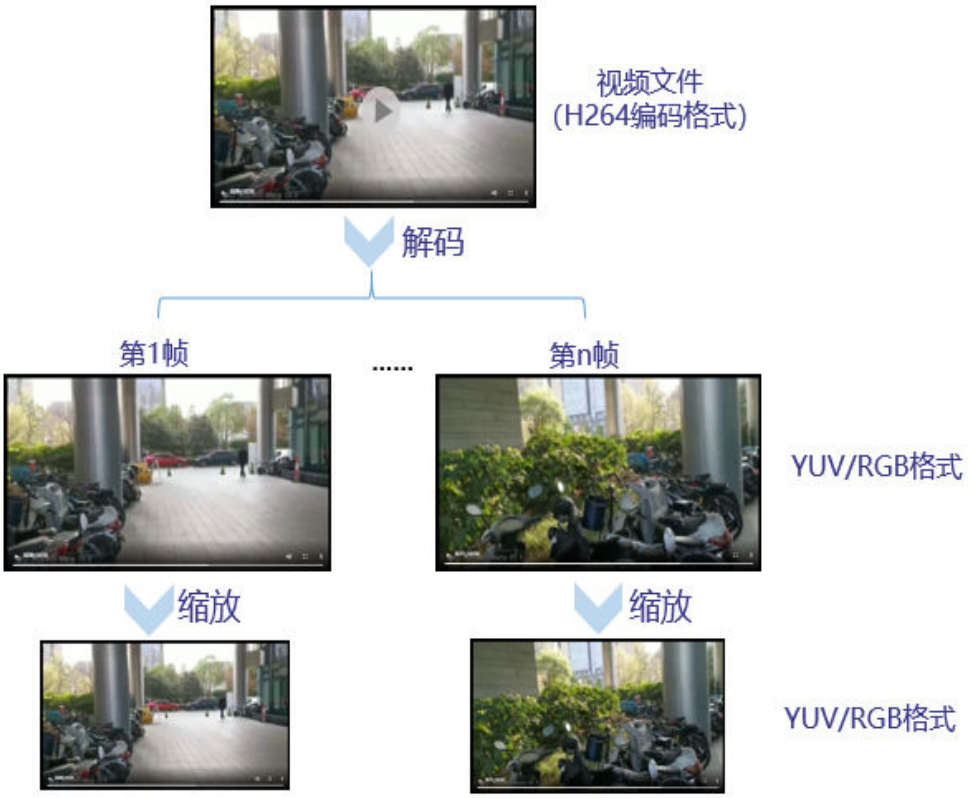
DVPP 图像/视频数据处理的典型使用场景

如果源图或视频的分辨率、格式等与模型的要求不一致时，我们可以将源图或视频处理成符合模型的要求。如下为**典型场景**的举例。

- **视频解码、缩放**

  使用Yolov3模型实现目标检测的场景下，用户提供的输入视频为H264/H265编码格式、分辨率为1920*1080，但Yolov3模型要求的输入图片格式为RGB/YUV、分辨率为416*416，两者不一致，此时可对视频执行以下一系列处理。

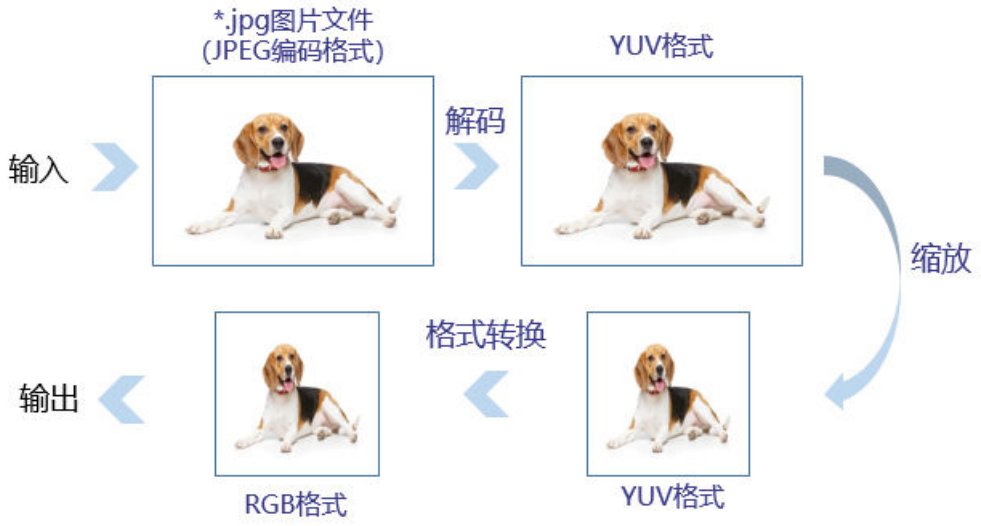
  图 5-2 视频解码、缩放使用场景图



- **图片解码、缩放、格式转换**

使用Resnet50模型实现图片分类的场景下，用户提供的输入图片为JPEG编码格式、分辨率为1280*720，但Resnet50模型要求的输入图片格式为RGB、分辨率为224*224，两者不一致，此时可对图片执行以下一系列处理。

图 5-3 图片解码、缩放、格式转换使用场景图



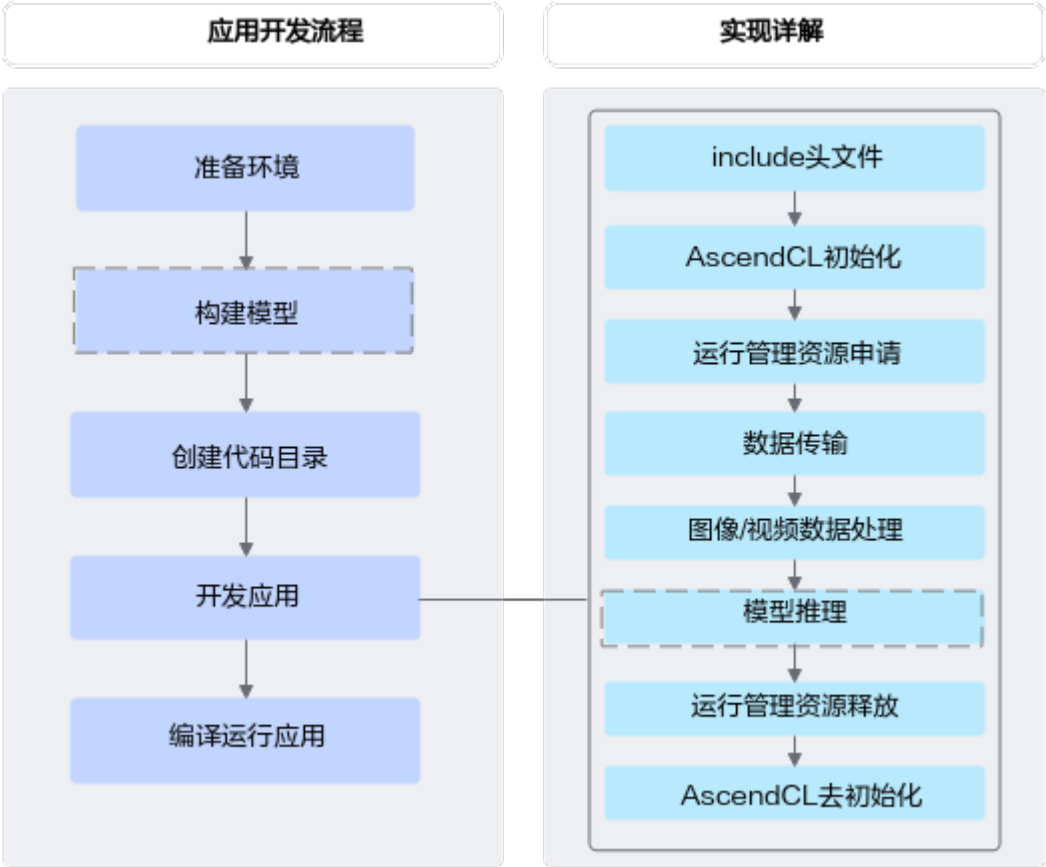
- **抠图、缩放、格式转换**

  使用Resnet50模型实现图片分类的场景下，用户提供的输入图片格式为YUV420SP、分辨率为1280*720，但Resnet50模型要求的输入图片格式为RGB、分辨率为224*224，两者不一致，此时对图片执行以下一系列处理。

图 5-4 抠图、缩放、格式转换使用场景图

媒体数据处理功能开发流程

图 5-5 开发流程



1. **准备环境**。

   请参见**3.3 准备开发和运行环境**。

2. **创建代码目录**。

   在开发应用前，您需要先创建目录，存放代码文件、编译脚本、测试图片数据、模型文件等

   如下仅是示例，供参考：

```
├App名称
├── model              // 该目录下存放模型文件
│   ├── xxx.json

├── data
│   ├── xxxxxx         // 测试数据

├── inc               // 该目录下存放声明函数的头文件
│   ├── xxx.h

├── out               // 该目录下存放输出结果

├── src
│   ├── xxx.json       // 系统初始化的配置文件
│   ├── CMakeLists.txt  // 编译脚本
│   ├── xxx.cpp        // 实现文件
```

3. **（可选）构建模型**。

模型推理场景下，必须要有适配昇腾AI处理器的离线模型（*.om文件），请参见**4.2 模型构建**。

说明

如果应用中涉及模型推理，则需要构建模型。

4. **开发应用**。

   依赖的头文件和库文件的说明请参见**调用接口依赖的头文件和库文件说明**。

   如果应用中涉及模型推理，请参见**4.6 模型推理**、**7 扩展更多特性**编写相应的代码。

5. **编译运行应用**，请参见**8 应用调试**。

5.2 媒体数据处理 V1 与 V2 版本的差别

媒体数据处理接口有V1、V2两套，本节介绍两套接口的共同点、差异点。

本手册中媒体数据处理V1版本与媒体数据处理V2版本的接口都是描述处理媒体数据的接口，用于实现抠图、图片缩放、格式转换等功能，但**V2版本的功能比V1版本更多，且两套接口不能混用**。

V2版本的功能比V1版本更多，如下：

- JPEGE：V2版本接口支持高级的参数配置，如huffman表配置。
- VENC：V2版本接口支持更加细化的码控参数配置和效果调优，如I/P帧QP、宏块码控等。
- VDEC：V2版本接口支持更细化的内存控制，如设置输入码流缓存。

须知

Atlas 200/300/500 推理产品上，当前仅支持V1版本的媒体数据处理接口。

Atlas 推理系列产品上，支持V1和V2两个版本的媒体数据处理接口，建议使用V2版本中的接口，保证后续版本接口功能以及业务的连续演进。

Atlas 200/300/500 推理产品和Atlas 推理系列产品都支持V1版本接口，但部分使用约束存在差异，如果涉及Atlas 200/300/500 推理产品媒体数据处理V1->Atlas 推理系列产品媒体数据处理V1迁移的场景，请参见**13.3.2 Atlas 200/300/500 推理产品媒体数据处理V1->Atlas 推理系列产品媒体数据处理V1迁移指引**。

V1、V2这两套媒体数据处理接口，在功能约束、接口调用流程上存在差异，如果涉及Atlas 200/300/500 推理产品媒体数据处理V1->Atlas 推理系列产品媒体数据处理V2迁移的场景，请参见**13.3.3 Atlas 200/300/500 推理产品媒体数据处理V1->Atlas 推理系列产品媒体数据处理V2迁移指引**。

5.3 各版本功能支持度说明

昇腾AI处理器对媒体数据处理各功能的支持度如下表所示。

版本	DVPP图像/视频处理（媒体数据处理V1）	DVPP图像/视频处理（媒体数据处理V2）
Atlas 200/300/500 推理产品	√	x
Atlas 推理系列产品	√	√

5.4 DVPP 图像/视频处理（媒体数据处理 V1）

5.4.1 VPC 图像处理典型功能

本节以抠图、缩放为例说明VPC图像处理时的接口调用流程，同时配合以下典型功能的示例代码辅助理解该接口调用流程。

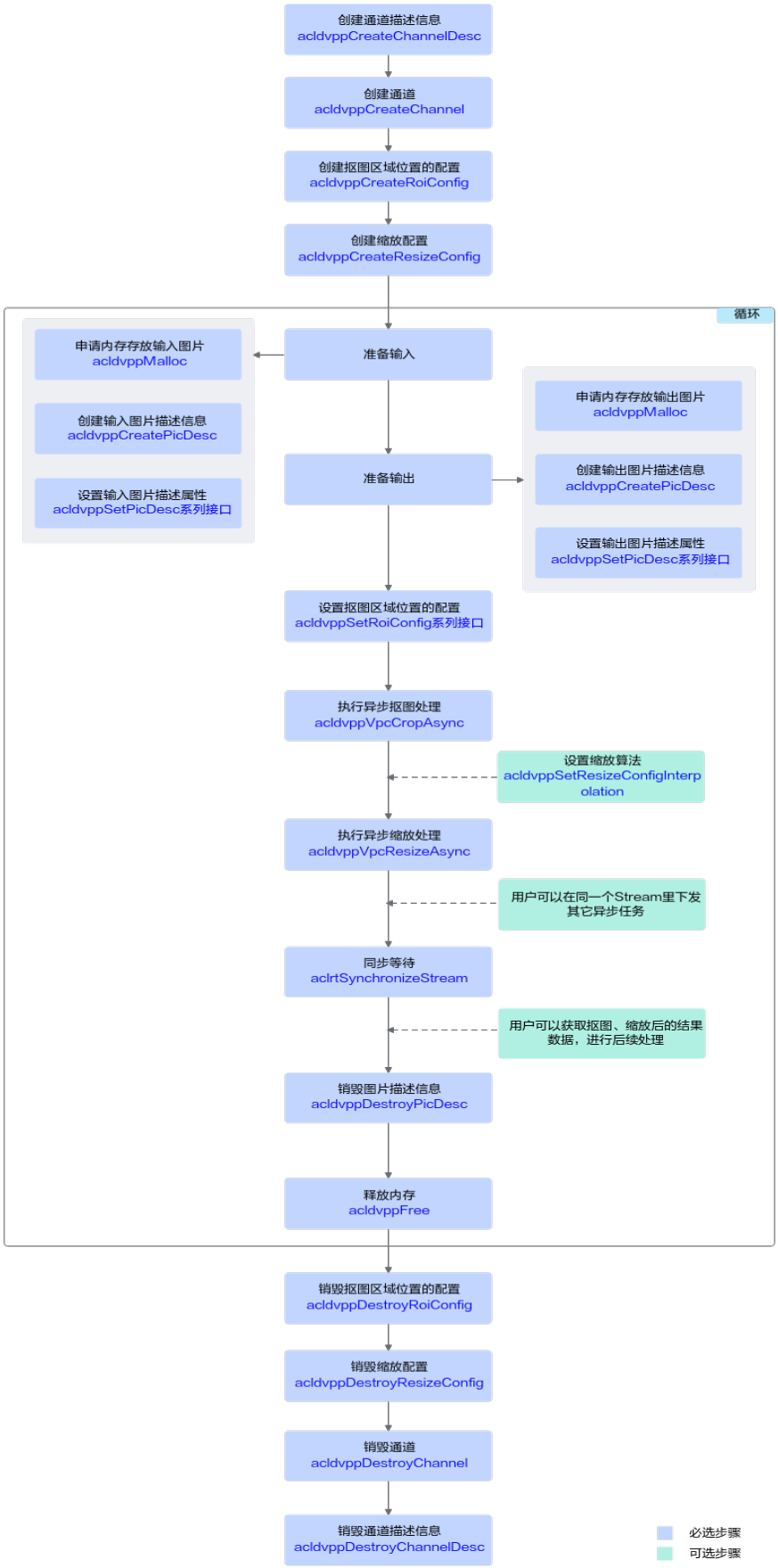
VPC（Vision Preprocessing Core）负责图像处理功能，支持对图片做抠图、缩放、格式转换等操作。关于VPC功能的详细介绍请参见**11.17.5.1 功能说明**，关于VPC功能对输入、输出的约束要求，请参见**11.17.5.2 约束说明**。

- 图片缩放示例代码
- 格式转换示例代码
- 抠图缩放（一图一框）示例代码
- 抠图贴图缩放（一图一框）示例代码
- 抠图贴图（一图多框）示例代码

接口调用流程（以抠图、缩放为例）

开发应用时，如果涉及抠图、缩放等图片处理，则应用程序中必须包含图片处理的代码逻辑，**关于图片处理的接口调用流程，请先参见3.2 AscendCL接口调用流程了解整体流程，再查看本节中的流程说明。**

图 5-6 抠图缩放流程



版权所有 © 华为技术有限公司

关键接口的说明如下（以抠图、缩放处理为例）：

1. 调用**acldvppCreateChannel**接口**创建图片数据处理的通道**。

   创建图片数据处理的通道前，需先调用**acldvppCreateChannelDesc**接口创建通道描述信息。
2. 调用**acldvppCreateRoiConfig**接口、**acldvppCreateResizeConfig**接口分别**创建抠图区域位置的配置、缩放配置**。
3. 实现抠图、缩放功能前，若需要**申请Device上的内存**存放输入或输出数据，需调用**acldvppMalloc**申请内存。
4. **执行抠图、缩放**。
   - 关于抠图：
     - 调用**acldvppVpcCropAsync**异步接口，按指定区域从输入图片中抠图，再将抠的图片存放到输出内存中，作为输出图片。

       输出图片区域与抠图区域cropArea不一致时会对图片再做一次缩放操作。
     - 当前系统还提供了**acldvppVpcCropAndPasteAsync**异步接口，支持按指定区域从输入图片中抠图，再将抠的图片贴到目标图片的指定位置，作为输出图片。
       - 抠图区域cropArea的宽高与贴图区域pasteArea宽高不一致时会对图片再做一次缩放操作。
       - 如果用户需要将目标图片读入内存用于存放输出图片，将贴图区域叠加在目标图片上，则需要编写代码逻辑：在申请输出内存后，将目标图片读入输出内存。
   - 关于缩放
     - 调用**acldvppVpcResizeAsync**异步接口，将输入图片缩放到输出图片大小。
     - 缩放后输出图片内存根据YUV420SP格式计算，计算公式：对齐后的宽*对齐后的高*3/2
   - 对于异步接口，还需调用**aclrtSynchronizeStream**接口阻塞程序运行，直到指定Stream中的所有任务都完成。
5. 调用**acldvppFree**接口**释放输入、输出内存**。
6. 调用**acldvppDestroyRoiConfig**接口、**acldvppDestroyResizeConfig**接口分别**销毁抠图区域位置的配置、缩放配置**。
7. 调用**acldvppDestroyChannel**接口**销毁图片数据处理的通道**。

   销毁图片数据处理的通道后，再调用**acldvppDestroyChannelDesc**接口销毁通道描述信息。

图片缩放示例代码

调用**acldvppSetResizeConfigInterpolation**接口指定缩放算法，再调用**acldvppVpcResizeAsync**异步接口，将输入图片缩放到输出图片大小。

您可以从10.4.1 样例介绍中获取完整样例代码。

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1.AscendCL初始化
aclRet = aclInit(nullptr);

// 2.运行管理资源申请（依次申请Device、Context、Stream）
aclrtContext context_;
aclrtStream stream_;
aclrtSetDevice(0);
aclrtCreateContext(&context_, 0);
aclrtCreateStream(&stream_);

// 3. 创建图片缩放配置数据、指定缩放算法
// resizeConfig_是acldvppResizeConfig类型
acldvppResizeConfig *resizeConfig_ = acldvppCreateResizeConfig();
aclError ret = acldvppSetResizeConfigInterpolation(resizeConfig, 0);

// 4. 创建图片数据处理通道时的通道描述信息，dvppChannelDesc_是acldvppChannelDesc类型
dvppChannelDesc_ = acldvppCreateChannelDesc();

// 5. 创建图片数据处理的通道。
ret = acldvppCreateChannel(dvppChannelDesc_);

// 6. 申请输入内存（区分运行状态）
// 调用aclrtGetRunMode接口获取软件栈的运行模式，如果调用aclrtGetRunMode接口获取软件栈的运行模式为ACL_HOST，则需要通过aclrtMemcpy接口将输入图片数据传输到Device，数据传输完成后，需及时释放内存；否则直接申请并使用Device的内存
aclrtRunMode runMode;
ret = aclrtGetRunMode(&runMode);
// inputPicWidth、inputPicHeight分别表示图片的对齐后宽、对齐后高，此处以YUV420SP格式的图片为例
uint32_t resizeInBufferSize = inputPicWidth * inputPicHeight * 3 / 2;
if（runMode == ACL_HOST）{
    // 申请Host内存vpcInHostBuffer
    void* vpcInHostBuffer = nullptr;
    vpcInHostBuffer = malloc(resizeInBufferSize);
    // 将输入图片读入内存中，该自定义函数ReadPicFile由用户实现
    ReadPicFile(picName, vpcInHostBuffer, resizeInBufferSize);
    // 申请Device内存resizeInDevBuffer_
    aclRet = acldvppMalloc(&resizeInDevBuffer_, resizeInBufferSize);
    // 通过aclrtMemcpy接口将输入图片数据传输到Device
    aclRet = aclrtMemcpy(resizeInDevBuffer_, resizeInBufferSize, vpcInHostBuffer, resizeInBufferSize,
ACL_MEMCPY_HOST_TO_DEVICE);
    // 数据传输完成后，及时释放内存
    free(vpcInHostBuffer);
} else {
    // 申请Device输入内存resizeInDevBuffer_
    ret = acldvppMalloc(&resizeInDevBuffer_, resizeInBufferSize);
    // 将输入图片读入内存中，该自定义函数ReadPicFile由用户实现
    ReadPicFile(picName, resizeInDevBuffer_, resizeInBufferSize);
}

// 7. 申请缩放输出内存resizeOutBufferDev_，内存大小resizeOutBufferSize_根据计算公式得出
// outputPicWidth、outputPicHeight分别表示图片的对齐后宽、对齐后高，此处以YUV420SP格式的图片为例
uint32_t resizeOutBufferSize_ = outputPicWidth * outputPicHeight * 3 / 2;
ret = acldvppMalloc(&resizeOutBufferDev_, resizeOutBufferSize_)

// 8. 创建缩放输入图片的描述信息，并设置各属性值
// resizeInputDesc_是acldvppPicDesc类型
resizeInputDesc_ = acldvppCreatePicDesc();
acldvppSetPicDescData(resizeInputDesc_, resizeInDevBuffer_);
acldvppSetPicDescFormat(resizeInputDesc_, PIXEL_FORMAT_YUV_SEMIPLANAR_420);
acldvppSetPicDescWidth(resizeInputDesc_, inputWidth_);
acldvppSetPicDescHeight(resizeInputDesc_, inputHeight_);
acldvppSetPicDescWidthStride(resizeInputDesc_, inputWidthStride);
acldvppSetPicDescHeightStride(resizeInputDesc_, inputHeightStride);
acldvppSetPicDescSize(resizeInputDesc_, resizeInBufferSize);

// 9. 创建缩放输出图片的描述信息，并设置各属性值
// 如果缩放的输出图片作为模型推理的输入，则输出图片的宽高要与模型要求的宽高保持一致
// resizeOutputDesc_是acldvppPicDesc类型
resizeOutputDesc_ = acldvppCreatePicDesc();
```

```
acldvppSetPicDescData(resizeOutputDesc_, resizeOutBufferDev_);
acldvppSetPicDescFormat(resizeOutputDesc_, PIXEL_FORMAT_YUV_SEMIPLANAR_420);
acldvppSetPicDescWidth(resizeOutputDesc_, resizeOutputWidth_);
acldvppSetPicDescHeight(resizeOutputDesc_, resizeOutputHeight_);
acldvppSetPicDescWidthStride(resizeOutputDesc_, resizeOutputWidthStride);
acldvppSetPicDescHeightStride(resizeOutputDesc_, resizeOutputHeightStride);
acldvppSetPicDescSize(resizeOutputDesc_, resizeOutBufferSize_);

// 10. 执行异步缩放，再调用aclrtSynchronizeStream接口阻塞程序运行，直到指定Stream中的所有任务都完成
ret = acldvppVpcResizeAsync(dvppChannelDesc_, resizeInputDesc_,
        resizeOutputDesc_, resizeConfig_, stream_);
ret = aclrtSynchronizeStream(stream_);

// 11. 缩放结束后，释放资源，包括缩放输入/输出图片的描述信息、缩放输入/输出内存
acldvppDestroyPicDesc(resizeInputDesc_);
acldvppDestroyPicDesc(resizeOutputDesc_);

if（runMode == ACL_HOST）{
    // 该模式下，由于处理结果在Device侧，因此需要调用内存复制接口传输结果数据后，再释放Device侧内存
    // 申请Host内存vpcOutHostBuffer
    void* vpcOutHostBuffer = nullptr;
    vpcOutHostBuffer = malloc(resizeOutBufferSize_);
    // 通过aclrtMemcpy接口将Device的处理结果数据传输到Host
    aclRet = aclrtMemcpy(vpcOutHostBuffer, resizeOutBufferSize_, resizeOutBufferDev_,
resizeOutBufferSize_, ACL_MEMCPY_DEVICE_TO_HOST);
    // 释放掉输入输出的device内存
    (void)acldvppFree(resizeInDevBuffer_);
    (void)acldvppFree(resizeOutBufferDev_);
    // 数据使用完成后，释放内存
    free(vpcOutHostBuffer);
} else {
    // 此时运行在device侧，处理结果也在Device侧，可以根据需要操作处理结果后，释放Device侧内存
    (void)acldvppFree(resizeInDevBuffer_);
    (void)acldvppFree(resizeOutBufferDev_);
}
acldvppDestroyChannel(dvppChannelDesc_);
(void)acldvppDestroyChannelDesc(dvppChannelDesc_);
dvppChannelDesc_ = nullptr;

// 12. 释放运行管理资源（依次释放Stream、Context、Device）
aclrtDestroyStream(stream_);
aclrtDestroyContext(context_);
aclrtResetDevice(0);

// 13.AscendCL去初始化
aclRet = aclFinalize();

// ....
```

格式转换示例代码

格式转换支持以下两种实现方式：

- 在实现抠图、缩放等功能时，调用对应的接口（例如**acldvppVpcCropAsync**接口）时，通过将输入图片和输出图片的格式设置成不同的，达到转换图片格式的目的。
- 如果仅仅做图片格式转换，也可以直接调用**acldvppVpcConvertColorAsync**接口。

  但Atlas 200/300/500 推理产品和Atlas 训练系列产品不支持调用该接口。

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1.AscendCL初始化
aclRet = aclInit(nullptr);
```

```
// 2.运行管理资源申请（依次申请Device、Context、Stream）
aclrtContext context_;
aclrtStream stream_;
aclrtSetDevice(0);
aclrtCreateContext(&context_, 0);
aclrtCreateStream(&stream_);

// 3. 创建图片数据处理通道时的通道描述信息，dvppChannelDesc_是acldvppChannelDesc类型
dvppChannelDesc_ = acldvppCreateChannelDesc();

// 4. 创建图片数据处理的通道。
aclError ret = acldvppCreateChannel(dvppChannelDesc_);

// 5. 申请输入内存（区分运行状态）
// 调用aclrtGetRunMode接口获取软件栈的运行模式，如果调用aclrtGetRunMode接口获取软件栈的运行模式为ACL_HOST，则需要通过aclrtMemcpy接口将输入图片数据传输到Device，数据传输完成后，需及时释放内存；否则直接申请并使用Device的内存
aclrtRunMode runMode;
ret = aclrtGetRunMode(&runMode);
// inputPicWidth、inputPicHeight分别表示图片的对齐后宽、对齐后高，此处以YUV420SP格式的图片为例
uint32_t inBufferSize = inputPicWidth * inputPicHeight * 3 / 2;
if（runMode == ACL_HOST）{
    // 申请Host内存vpcInHostBuffer
    void* vpcInHostBuffer = nullptr;
    vpcInHostBuffer = malloc(inBufferSize);
    // 将输入图片读入内存中，该自定义函数ReadPicFile由用户实现
    ReadPicFile(picName, vpcInHostBuffer, inBufferSize);
    // 申请Device内存inDevBuffer_
    aclRet = acldvppMalloc(&inDevBuffer_, inBufferSize);
    // 通过aclrtMemcpy接口将输入图片数据传输到Device
    aclRet = aclrtMemcpy(inDevBuffer_, inBufferSize, vpcInHostBuffer, inBufferSize,
ACL_MEMCPY_HOST_TO_DEVICE);
    // 数据传输完成后，及时释放内存
    free(vpcInHostBuffer);
} else {
    // 申请Device输入内存inDevBuffer_
    ret = acldvppMalloc(&inDevBuffer_, inBufferSize);
    // 将输入图片读入内存中，该自定义函数ReadPicFile由用户实现
    ReadPicFile(picName, inDevBuffer_, inBufferSize);
}

// 6. 申请色域转换输出内存outBufferDev_，内存大小outBufferSize_根据计算公式得出
// outputPicWidth、outputPicHeight分别表示图片的对齐后宽、对齐后高，此处以YUV420SP格式的图片为例
uint32_t outBufferSize_ = outputPicWidth * outputPicHeight * 3 / 2
ret = acldvppMalloc(&outBufferDev_, outBufferSize_)

// 7. 创建色域转换输入图片的描述信息，并设置各属性值
// inputDesc_是acldvppPicDesc类型
inputDesc_ = acldvppCreatePicDesc();
acldvppSetPicDescData(inputDesc_, inDevBuffer_);
acldvppSetPicDescFormat(inputDesc_, PIXEL_FORMAT_YUV_SEMIPLANAR_420);
acldvppSetPicDescWidth(inputDesc_, inputWidth_);
acldvppSetPicDescHeight(inputDesc_, inputHeight_);
acldvppSetPicDescWidthStride(inputDesc_, inputWidthStride);
acldvppSetPicDescHeightStride(inputDesc_, inputHeightStride);
acldvppSetPicDescSize(inputDesc_, inBufferSize);

// 8. 创建色域转换的输出图片的描述信息，并设置各属性值, 输出的宽和高要求和输入一致
// 如果色域转换的输出图片作为模型推理的输入，则输出图片的宽高要与模型要求的宽高保持一致
// outputDesc_是acldvppPicDesc类型
outputDesc_= acldvppCreatePicDesc();
acldvppSetPicDescData(outputDesc_, outBufferDev_);
acldvppSetPicDescFormat(outputDesc_, PIXEL_FORMAT_YUV_400);
acldvppSetPicDescWidth(outputDesc_, outputWidth_);
acldvppSetPicDescHeight(outputDesc_, outputHeight_);
acldvppSetPicDescWidthStride(outputDesc_, outputWidthStride);
acldvppSetPicDescHeightStride(outputDesc_, outputHeightStride);
acldvppSetPicDescSize(outputDesc_, outBufferSize_);
```

```
// 9. 执行异步色域转换，再调用aclrtSynchronizeStream接口阻塞程序运行，直到指定Stream中的所有任务都完成
ret = acldvppVpcConvertColorAsync(dvppChannelDesc_, inputDesc_, outputDesc_, stream_);
ret = aclrtSynchronizeStream(stream_);

// 10. 色域转换结束后，释放资源，包括输入/输出图片的描述信息、输入/输出内存
acldvppDestroyPicDesc(inputDesc_);
acldvppDestroyPicDesc(outputDesc_);

if（runMode == ACL_HOST）{
    // 该模式下，由于处理结果在Device侧，因此需要调用内存复制接口传输结果数据后，再释放Device侧内存
    // 申请Host内存vpcOutHostBuffer
    void* vpcOutHostBuffer = nullptr;
    vpcOutHostBuffer = malloc(outBufferSize_);
    // 通过aclrtMemcpy接口将Device的处理结果数据传输到Host
    aclRet = aclrtMemcpy(vpcOutHostBuffer, outBufferSize_, outBufferDev_, outBufferSize_,
ACL_MEMCPY_DEVICE_TO_HOST);
    // 释放掉输入输出的device内存
    (void)acldvppFree(inDevBuffer_);
    (void)acldvppFree(outBufferDev_);
    // 数据使用完成后，释放内存
    free(vpcOutHostBuffer);
} else {
    // 此时运行在device侧，处理结果也在Device侧，可以根据需要操作处理结果后，释放Device侧内存
    (void)acldvppFree(inDevBuffer_);
    (void)acldvppFree(outBufferDev_);
}
acldvppDestroyChannel(dvppChannelDesc_);
(void)acldvppDestroyChannelDesc(dvppChannelDesc_);
dvppChannelDesc_ = nullptr;

// 11. 释放运行管理资源（依次释放Stream、Context、Device）
aclrtDestroyStream(stream_);
aclrtDestroyContext(context_);
aclrtResetDevice(0);

// 12.AscendCL去初始化
aclRet = aclFinalize();

// ....
```

抠图缩放（一图一框）示例代码

调用acldvppVpcCropResizeAsync异步接口，按指定区域从输入图片中抠图，再将抠的图片存放到输出内存中，作为输出图片，此外，还支持指定缩放算法。当输出图片区域与抠图区域cropArea不一致时会对图片再做一次缩放操作。

您可以从10.5.1 样例介绍中获取完整样例代码。

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1.AscendCL初始化
aclRet = aclInit(nullptr);

// 2.运行管理资源申请（依次申请Device、Context、Stream）
aclrtContext context_;
aclrtStream stream_;
aclrtSetDevice(0);
aclrtCreateContext(&context_, 0);
aclrtCreateStream(&stream_);

// 3. 创建缩放配置数据，并指定抠图区域的位置
// resizeConfig_是acldvppResizeConfig类型
resizeConfig_ = acldvppCreateResizeConfig();
aclError aclRet = acldvppSetResizeConfigInterpolation(resizeConfig_, 0);
// cropArea_是acldvppRoiConfig类型
cropArea_ = acldvppCreateRoiConfig(550, 749, 480, 679);
```

```
// 4. 创建图片数据处理通道时的通道描述信息，dvppChannelDesc_是acldvppChannelDesc类型
dvppChannelDesc_ = acldvppCreateChannelDesc();

// 5. 创建图片数据处理的通道。
ret = acldvppCreateChannel(dvppChannelDesc_);

// 6. 申请输入内存（区分运行状态）
// 调用aclrtGetRunMode接口获取软件栈的运行模式，如果调用aclrtGetRunMode接口获取软件栈的运行模式为ACL_HOST，则需要通过aclrtMemcpy接口将输入图片数据传输到Device，数据传输完成后，需及时释放内存；否则直接申请并使用Device的内存
aclrtRunMode runMode;
ret = aclrtGetRunMode(&runMode);
// inputPicWidth、inputPicHeight分别表示图片的对齐后宽、对齐后高，此处以YUV420SP格式的图片为例
uint32_t cropInBufferSize = inputPicWidth * inputPicHeight * 3 / 2;
if（runMode == ACL_HOST）{
    // 申请Host内存cropInHostBuffer
    void* cropInHostBuffer = nullptr;
    cropInHostBuffer = malloc(cropInBufferSize);
    // 将输入图片读入内存中，该自定义函数ReadPicFile由用户实现
    ReadPicFile(picName, cropInHostBuffer, cropInBufferSize);
    // 申请Device内存cropInDevBuffer_
    aclRet = acldvppMalloc(&cropInDevBuffer_, cropInBufferSize);
    // 通过aclrtMemcpy接口图片数据传输到Device
    aclRet = aclrtMemcpy(cropInDevBuffer_, cropInBufferSize, cropInHostBuffer, cropInBufferSize,
ACL_MEMCPY_HOST_TO_DEVICE);
    // 数据传输完成后，及时释放内存
    free(cropInHostBuffer);
} else {
    // 申请Device输入内存cropInDevBuffer_
    ret = acldvppMalloc(&cropInDevBuffer_, cropInBufferSize);
    // 将输入图片读入内存中，该自定义函数ReadPicFile由用户实现
    ReadPicFile(picName, cropInDevBuffer_, cropInBufferSize);
}

// 7. 申请Device输出内存cropOutBufferDev_，内存大小cropOutBufferSize_根据计算公式得出
// outputPicWidth、outputPicHeight分别表示图片的对齐后宽、对齐后高，此处以YUV420SP格式的图片为例
uint32_t cropOutBufferSize_ = outputPicWidth * outputPicHeight * 3 / 2;
ret = acldvppMalloc(&cropOutBufferDev_, cropOutBufferSize_)

// 8. 创建输入图片的描述信息，并设置各属性值，cropInputDesc_是acldvppPicDesc类型
cropInputDesc_ = acldvppCreatePicDesc();
acldvppSetPicDescData(cropInputDesc_, cropInDevBuffer_);
acldvppSetPicDescFormat(cropInputDesc_, PIXEL_FORMAT_YUV_SEMIPLANAR_420);
acldvppSetPicDescWidth(cropInputDesc_, inputWidth_);
acldvppSetPicDescHeight(cropInputDesc_, inputHeight_);
acldvppSetPicDescWidthStride(cropInputDesc_, inputWidthStride);
acldvppSetPicDescHeightStride(cropInputDesc_, inputHeightStride);
acldvppSetPicDescSize(cropInputDesc_, cropInBufferSize);

// 9. 创建输出图片的描述信息，并设置各属性值，cropOutputDesc_是acldvppPicDesc类型
// 如果抠图的输出图片作为模型推理的输入，则输出图片的宽高要与模型要求的宽高保持一致
cropOutputDesc_ = acldvppCreatePicDesc();
acldvppSetPicDescData(cropOutputDesc_, cropOutBufferDev_);
acldvppSetPicDescFormat(cropOutputDesc_, PIXEL_FORMAT_YUV_SEMIPLANAR_420);
acldvppSetPicDescWidth(cropOutputDesc_, OutputWidth_);
acldvppSetPicDescHeight(cropOutputDesc_, OutputHeight_);
acldvppSetPicDescWidthStride(cropOutputDesc_, OutputWidthStride);
acldvppSetPicDescHeightStride(cropOutputDesc_, OutputHeightStride);
acldvppSetPicDescSize(cropOutputDesc_, cropOutBufferSize_);

// 10. 执行异步抠图缩放，再调用aclrtSynchronizeStream接口阻塞程序运行，直到指定Stream中的所有任务都完成
ret = acldvppVpcCropResizeAsync(dvppChannelDesc_, cropInputDesc_,
        cropOutputDesc_, cropArea_, resizeConfig_, stream_);
ret = aclrtSynchronizeStream(stream_);

// 11. 抠图贴图结束后，释放资源，包括输入/输出图片的描述信息、输入/输出内存、通道描述信息、通道等
acldvppDestroyRoiConfig(cropArea_);
```

```
acldvppDestroyResizeConfig(resizeConfig_);
acldvppDestroyPicDesc(cropInputDesc_);
acldvppDestroyPicDesc(cropOutputDesc_);
if（runMode == ACL_HOST）{
    // 该模式下，由于处理结果在Device侧，因此需要调用内存复制接口传输结果数据后，再释放Device侧内存
    // 申请Host内存cropOutHostBuffer
    void* cropOutHostBuffer = nullptr;
    cropOutHostBuffer = malloc(cropOutBufferSize_);
    // 通过aclrtMemcpy接口将Device的处理结果数据传输到Host
    aclRet = aclrtMemcpy(cropOutHostBuffer, cropOutBufferSize_, cropOutBufferDev_, cropOutBufferSize_,
ACL_MEMCPY_DEVICE_TO_HOST);
    // 释放掉输入输出的device内存
    (void)acldvppFree(cropInDevBuffer_);
    (void)acldvppFree(cropOutBufferDev_);
    // 数据使用完成后，释放内存
    free(cropOutHostBuffer);
} else {
    // 此时运行在device侧，处理结果也在Device侧，可以根据需要操作抠图结果后，释放Device侧内存
    (void)acldvppFree(cropInDevBuffer_);
    (void)acldvppFree(cropOutBufferDev_);
}
acldvppDestroyChannel(dvppChannelDesc_);
(void)acldvppDestroyChannelDesc(dvppChannelDesc_);
dvppChannelDesc_ = nullptr;

// 12. 释放运行管理资源（依次释放Stream、Context、Device）
aclrtDestroyStream(stream_);
aclrtDestroyContext(context_);
aclrtResetDevice(0);

// 13.AscendCL去初始化
aclRet = aclFinalize();

// ....
```

抠图贴图缩放（一图一框）示例代码

调用**acldvppVpcCropResizePasteAsync**异步接口，按指定区域从输入图片中抠图，再将抠的图片贴到目标图片的指定位置，作为输出图片，此外，还支持指定缩放算法。当抠图区域cropArea的宽高与贴图区域pasteArea宽高不一致时会对图片再做一次缩放操作。如果用户需要将目标图片读入内存用于存放输出图片，将贴图区域叠加在目标图片上，则需要编写代码逻辑：在申请输出内存后，将目标图片读入输出内存。

您可以从10.5.1 样例介绍中获取完整样例代码。

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1.AscendCL初始化
aclRet = aclInit(nullptr);

// 2.运行管理资源申请（依次申请Device、Context、Stream）
aclrtContext context_;
aclrtStream stream_;
aclrtSetDevice(0);
aclrtCreateContext(&context_, 0);
aclrtCreateStream(&stream_);

// 3.  创建缩放配置数据，并指定抠图区域的位置、指定贴图区域的位置
// resizeConfig_是acldvppResizeConfig类型
resizeConfig_ = acldvppCreateResizeConfig();
aclError aclRet = acldvppSetResizeConfigInterpolation(resizeConfig_, 0);
// cropArea_和pasteArea_是acldvppRoiConfig类型
cropArea_ = acldvppCreateRoiConfig(512, 711, 512, 711);
pasteArea_ = acldvppCreateRoiConfig(16, 215, 16, 215);

// 4. 创建图片数据处理通道时的通道描述信息，dvppChannelDesc_是acldvppChannelDesc类型
```

```
dvppChannelDesc_ = acldvppCreateChannelDesc();

// 5. 创建图片数据处理的通道。
ret = acldvppCreateChannel(dvppChannelDesc_);

// 6. 申请输入内存（区分运行状态）
// 调用aclrtGetRunMode接口获取软件栈的运行模式，如果调用aclrtGetRunMode接口获取软件栈的运行模式为ACL_HOST，则需要通过aclrtMemcpy接口将输入图片数据传输到Device，数据传输完成后，需及时释放内存；否则直接申请并使用Device的内存
aclrtRunMode runMode;
ret = aclrtGetRunMode(&runMode);
// inputPicWidth、inputPicHeight分别表示图片的对齐后宽、对齐后高，此处以YUV420SP格式的图片为例
uint32_t vpcInBufferSize = inputPicWidth * inputPicHeight * 3 / 2;
if（runMode == ACL_HOST）{
    // 申请Host内存vpcInHostBuffer
    void* vpcInHostBuffer = nullptr;
    vpcInHostBuffer = malloc(vpcInBufferSize);
    // 将输入图片读入内存中，该自定义函数ReadPicFile由用户实现
    ReadPicFile(picName, vpcInHostBuffer, vpcInBufferSize);
    // 申请Device内存vpcInDevBuffer_
    aclRet = acldvppMalloc(&vpcInDevBuffer_, vpcInBufferSize);
    // 通过aclrtMemcpy接口将输入图片数据传输到Device
    aclRet = aclrtMemcpy(vpcInDevBuffer_, vpcInBufferSize, vpcInHostBuffer, vpcInBufferSize,
ACL_MEMCPY_HOST_TO_DEVICE);
    // 数据传输完成后，及时释放内存
    free(vpcInHostBuffer);
} else {
    // 申请Device输入内存vpcInDevBuffer_
    ret = acldvppMalloc(&vpcInDevBuffer_, vpcInBufferSize);
    // 将输入图片读入内存中，该自定义函数ReadPicFile由用户实现
    ReadPicFile(picName, vpcInDevBuffer_, vpcInBufferSize);
}

// 7. 申请输出内存vpcOutBufferDev_，内存大小vpcOutBufferSize_根据计算公式得出
// outputPicWidth、outputPicHeight分别表示图片的对齐后宽、对齐后高，此处以YUV420SP格式的图片为例
uint32_t vpcOutBufferSize_ = outputPicWidth * outputPicHeight * 3 / 2;
ret = acldvppMalloc(&vpcOutBufferDev_, vpcOutBufferSize_)

// 8. 创建输入图片的描述信息，并设置各属性值
// 此处示例将解码的输出内存作为抠图贴图的输入，vpcInputDesc_是acldvppPicDesc类型
vpcInputDesc_ = acldvppCreatePicDesc();
acldvppSetPicDescData(vpcInputDesc_, decodeOutBufferDev_);
acldvppSetPicDescFormat(vpcInputDesc_, PIXEL_FORMAT_YUV_SEMIPLANAR_420);
acldvppSetPicDescWidth(vpcInputDesc_, inputWidth_);
acldvppSetPicDescHeight(vpcInputDesc_, inputHeight_);
acldvppSetPicDescWidthStride(vpcInputDesc_, jpegOutWidthStride);
acldvppSetPicDescHeightStride(vpcInputDesc_, jpegOutHeightStride);
acldvppSetPicDescSize(vpcInputDesc_, jpegOutBufferSize);

// 9. 创建输出图片的描述信息，并设置各属性值
// 如果抠图贴图的输出图片作为模型推理的输入，则输出图片的宽高要与模型要求的宽高保持一致
// vpcOutputDesc_是acldvppPicDesc类型
vpcOutputDesc_ = acldvppCreatePicDesc();
acldvppSetPicDescData(vpcOutputDesc_, vpcOutBufferDev_);
acldvppSetPicDescFormat(vpcOutputDesc_, PIXEL_FORMAT_YUV_SEMIPLANAR_420);
acldvppSetPicDescWidth(vpcOutputDesc_, dvppOutWidth);
acldvppSetPicDescHeight(vpcOutputDesc_, dvppOutHeight);
acldvppSetPicDescWidthStride(vpcOutputDesc_, dvppOutWidthStride);
acldvppSetPicDescHeightStride(vpcOutputDesc_, dvppOutHeightStride);
acldvppSetPicDescSize(vpcOutputDesc_, vpcOutBufferSize_);

// 10. 执行异步抠图贴图缩放，再调用aclrtSynchronizeStream接口阻塞程序运行，直到指定Stream中的所有任务都完成
ret = acldvppVpcCropResizePasteAsync(dvppChannelDesc_, vpcInputDesc_,
        vpcOutputDesc_, cropArea_, pasteArea_, resizeConfig_, stream_);
ret = aclrtSynchronizeStream(stream_);

// 11. 抠图贴图结束后，释放资源，包括输入/输出图片的描述信息、输入/输出内存、通道描述信息、通道等
acldvppDestroyRoiConfig(cropArea_);
```

```
acldvppDestroyRoiConfig(pasteArea_);
acldvppDestroyResizeConfig(resizeConfig_);
acldvppDestroyPicDesc(vpcInputDesc_);
acldvppDestroyPicDesc(vpcOutputDesc_);

if（runMode == ACL_HOST）{
    // 该模式下，由于处理结果在Device侧，因此需要调用内存复制接口传输结果数据后，再释放Device侧内存
    // 申请Host内存vpcOutHostBuffer
    void* vpcOutHostBuffer = nullptr;
    vpcOutHostBuffer = malloc(vpcOutBufferSize_);
    // 通过aclrtMemcpy接口将Device的处理结果数据传输到Host
    aclRet = aclrtMemcpy(vpcOutHostBuffer, vpcOutBufferSize_, vpcOutBufferDev_, vpcOutBufferSize_,
ACL_MEMCPY_DEVICE_TO_HOST);
    // 释放掉输入输出的device内存
    (void)acldvppFree(vpcInDevBuffer_);
    (void)acldvppFree(vpcOutBufferDev_);
    // 数据使用完成后，释放内存
    free(vpcOutHostBuffer);
} else {
    // 此时运行在device侧，处理结果也在Device侧，可以根据需要操作处理结果后，释放Device侧内存
    (void)acldvppFree(vpcInDevBuffer_);
    (void)acldvppFree(vpcOutBufferDev_);
}

acldvppDestroyChannel(dvppChannelDesc_);
(void)acldvppDestroyChannelDesc(dvppChannelDesc_);
dvppChannelDesc_ = nullptr;

// 12. 释放运行管理资源（依次释放Stream、Context、Device）
aclrtDestroyStream(stream_);
aclrtDestroyContext(context_);
aclrtResetDevice(0);

// 13.AscendCL去初始化
aclRet = aclFinalize();

// ....
```

抠图贴图（一图多框）示例代码

调用acldvppVpcBatchCropAndPasteAsync异步接口，按指定区域从输入图片中抠图，再将抠的图片贴到目标图片的指定位置，作为输出图片。当抠图区域cropArea的宽高与贴图区域pasteArea宽高不一致时会对图片再做一次缩放操作。如果用户需要将目标图片读入内存用于存放输出图片，将贴图区域叠加在目标图片上，则需要编写代码逻辑：在申请输出内存后，将目标图片读入输出内存。

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1.AscendCL初始化
aclRet = aclInit(nullptr);

// 2.运行管理资源申请（依次申请Device、Context、Stream）
aclrtContext context_;
aclrtStream stream_;
aclrtSetDevice(0);
aclrtCreateContext(&context_, 0);
aclrtCreateStream(&stream_);

// 3.指定批量抠图区域的位置、指定批量贴图区域的位置,cropAreas_和pasteAreas_是acldvppRoiConfig类型
acldvppRoiConfig *cropAreas_[2], pasteAreas_[2];
cropAreas_[0] = acldvppCreateRoiConfig(512, 711, 512, 711);
cropAreas_[1] = acldvppCreateRoiConfig(512, 711, 512, 711);
pasteAreas_[0] = acldvppCreateRoiConfig(16, 215, 16, 215);
pasteAreas_[1] = acldvppCreateRoiConfig(16, 215, 16, 215);
```

```
// 4. 创建图片数据处理通道时的通道描述信息，dvppChannelDesc_是acldvppChannelDesc类型
dvppChannelDesc_ = acldvppCreateChannelDesc();

// 5. 创建图片数据处理的通道。
aclError ret = acldvppCreateChannel(dvppChannelDesc_);

// 6. 申请输入内存（区分运行状态）
// 调用aclrtGetRunMode接口获取软件栈的运行模式，如果调用aclrtGetRunMode接口获取软件栈的运行模式为ACL_HOST，则需要通过aclrtMemcpy接口将输入图片数据传输到Device，数据传输完成后，需及时释放内存；否则直接申请并使用Device的内存
aclrtRunMode runMode;
ret = aclrtGetRunMode(&runMode);
// inputPicWidth、inputPicHeight分别表示图片的对齐后宽、对齐后高，此处以YUV420SP格式的图片为例
uint32_t vpcInBufferSize = inputPicWidth * inputPicHeight * 3 / 2;
if（runMode == ACL_HOST）{
    // 申请Host内存vpcInHostBuffer
    void* vpcInHostBuffer = nullptr;
    vpcInHostBuffer = malloc(vpcInBufferSize);
    // 将输入图片读入内存中，该自定义函数ReadPicFile由用户实现
    ReadPicFile(picName, vpcInHostBuffer, vpcInBufferSize);
    // 申请Device内存vpcInDevBuffer_
    aclRet = acldvppMalloc(&vpcInDevBuffer_, vpcInBufferSize);
    // 通过aclrtMemcpy接口将Host的图片数据传输到Device
    aclRet = aclrtMemcpy(vpcInDevBuffer_, vpcInBufferSize, vpcInHostBuffer, vpcInBufferSize,
ACL_MEMCPY_HOST_TO_DEVICE);
    // 数据传输完成后，及时释放内存
    free(vpcInHostBuffer);
} else {
    // 申请Device输入内存vpcInDevBuffer_
    ret = acldvppMalloc(&vpcInDevBuffer_, vpcInBufferSize);
    // 将输入图片读入内存中，该自定义函数ReadPicFile由用户实现
    ReadPicFile(picName, vpcInDevBuffer_, vpcInBufferSize);
}

// 7. 申请输出内存vpcOutBufferDev_，内存大小vpcOutBufferSize_根据计算公式得出
// outputPicWidth、outputPicHeight分别表示图片的对齐后宽、对齐后高，此处以YUV420SP格式的图片为例
uint32_t vpcOutBufferSize_ = outputPicWidth * outputPicHeight * 3 / 2;
ret = acldvppMalloc(&vpcOutBufferDev_, vpcOutBufferSize_)

// 8. 创建输入图片的描述信息，并设置各属性值
// 此处示例将解码的输出内存作为抠图贴图的输入，vpcInputDesc_是acldvppPicDesc类型
vpcInputBatchDesc_ = acldvppCreateBatchPicDesc(1);
vpcInputDesc_ = acldvppGetPicDesc(vpcInputBatchDesc_, 0);
acldvppSetPicDescData(vpcInputDesc_, decodeOutBufferDev_);
acldvppSetPicDescFormat(vpcInputDesc_, PIXEL_FORMAT_YUV_SEMIPLANAR_420);
acldvppSetPicDescWidth(vpcInputDesc_, inputWidth_);
acldvppSetPicDescHeight(vpcInputDesc_, inputHeight_);
acldvppSetPicDescWidthStride(vpcInputDesc_, jpegOutWidthStride);
acldvppSetPicDescHeightStride(vpcInputDesc_, jpegOutHeightStride);
acldvppSetPicDescSize(vpcInputDesc_, jpegOutBufferSize);

// 9. 创建批量输出图片的描述信息，并设置各属性值
// 如果抠图贴图的输出图片作为模型推理的输入，则输出图片的宽高要与模型要求的宽高保持一致
// vpcOutputDesc_是acldvppPicDesc类型
vpcOutputBatchDesc_ = acldvppCreateBatchPicDesc(2);
for (uint32_t index=0; index<2; ++index){
     vecOutPtr_.push_back(vpcOutBufferDev_);
     vpcOutputDesc_ = acldvppGetPicDesc(vpcInputBatchDesc_, index);
    acldvppSetPicDescData(vpcOutputDesc_, vpcOutBufferDev_);
    acldvppSetPicDescFormat(vpcOutputDesc_, PIXEL_FORMAT_YUV_SEMIPLANAR_420);
    acldvppSetPicDescWidth(vpcOutputDesc_, dvppOutWidth);
    acldvppSetPicDescHeight(vpcOutputDesc_, dvppOutHeight);
    acldvppSetPicDescWidthStride(vpcOutputDesc_, dvppOutWidthStride);
    acldvppSetPicDescHeightStride(vpcOutputDesc_, dvppOutHeightStride);
    acldvppSetPicDescSize(vpcOutputDesc_, vpcOutBufferSize_);
}

// 10. 创建roiNums,每张图对应需要抠图和贴图的数量
```

```
uint32_ttotalNum = 0;
std::unique_ptr<uint32_t[]> roiNums(new (std::nothrow) uint32_t[1]);
roiNums[0]=2;
// 11. 执行异步抠图贴图，再调用aclrtSynchronizeStream接口阻塞程序运行，直到指定Stream中的所有任务都完成
ret = acldvppVpcBatchCropAndPasteAsync(dvppChannelDesc_, vpcInputBatchDesc_, roiNums.get(), 1,
        vpcOutputBatchDesc_, cropAreas_, pasteAreas_, stream_);
ret = aclrtSynchronizeStream(stream_);

// 12. 抠图贴图结束后，释放资源，包括输入/输出图片的描述信息、输入/输出内存、通道描述信息、通道等
acldvppDestroyRoiConfig(cropAreas_[0]);
acldvppDestroyRoiConfig(cropAreas_[1]);
acldvppDestroyRoiConfig(pasteAreas_[0]);
acldvppDestroyRoiConfig(pasteAreas_[1]);
(void)acldvppFree(vpcInDevBuffer_);
for(uint32_t index=0; index<2; ++index){
if（runMode == ACL_HOST）{
    // 该模式下，由于处理结果在Device侧，因此需要调用内存复制接口传输结果数据后，再释放Device侧内存
    // 申请Host内存vpcOutHostBuffer
    void* vpcOutHostBuffer = nullptr;
    vpcOutHostBuffer = malloc(vpcOutBufferSize_);
    // 通过aclrtMemcpy接口将Device的处理结果数据传输到Host
    aclRet = aclrtMemcpy(vpcOutHostBuffer, vpcOutBufferSize_, vpcOutBufferDev_, vpcOutBufferSize_,
ACL_MEMCPY_DEVICE_TO_HOST);
    // 释放掉输入输出的device内存
    (void)acldvppFree(vpcOutBufferDev_);
    // 数据使用完成后，释放内存
    free(vpcOutHostBuffer);
} else {
    // 此时运行在device侧，处理结果也在Device侧，可以根据需要操作处理结果后，释放Device侧内存

    (void)acldvppFree(vpcOutBufferDev_);
}
}
acldvppDestroyBatchPicDesc(vpcInputDesc_);
acldvppDestroyBatchPicDesc(vpcOutputDesc_);
acldvppDestroyChannel(dvppChannelDesc_);
(void)acldvppDestroyChannelDesc(dvppChannelDesc_);
dvppChannelDesc_ = nullptr;

// 13. 释放运行管理资源（依次释放Stream、Context、Device）
aclrtDestroyStream(stream_);
aclrtDestroyContext(context_);
aclrtResetDevice(0);

// 14.AscendCL去初始化
aclRet = aclFinalize();

// ....
```

5.4.2 JPEGD 图像解码

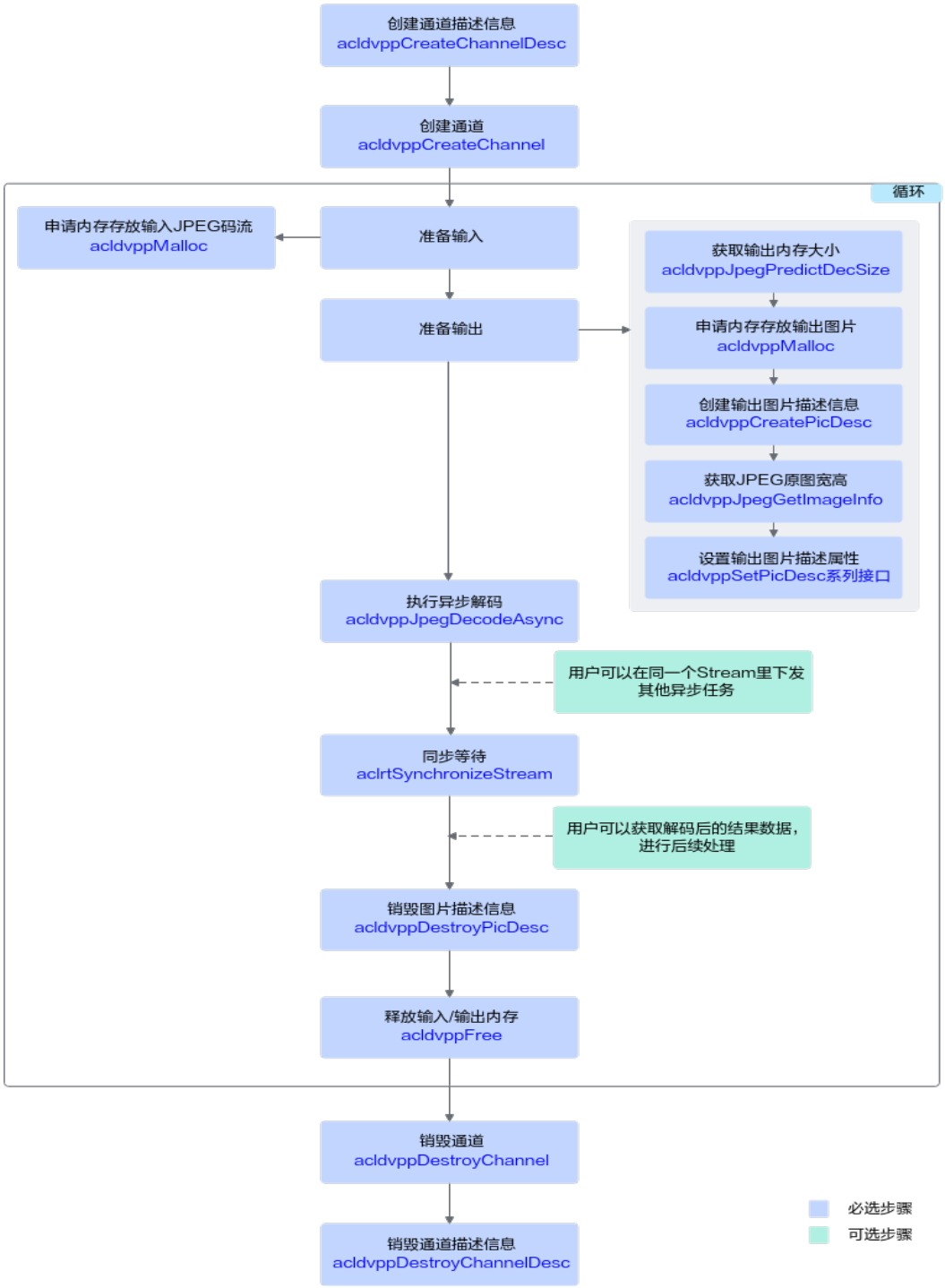
本节介绍JPEGD图片解码的接口调用流程，同时配合示例代码辅助理解该接口调用流程。

JPEGD（JPEG Decoder）负责完成图像解码功能，将.jpg、.jpeg、.JPG、.JPEG图片解码成YUV格式图片。关于JPEGD功能的详细介绍及约束请参见**11.17.6.1 功能及约束说明**。

接口调用流程

开发应用时，如果涉及对JPEG图片的解码，则应用程序中必须包含图片解码的代码逻辑，**关于图片解码的接口调用流程，请先参见3.2 AscendCL接口调用流程了解整体流程，再查看本节中的流程说明。**

图 5-7 JPEG 图片解码



当前系统支持.jpg、.jpeg、.JPG、.JPEG图片的解码，针对不同的源图编码格式，输出不同编码格式的图片，关键接口的说明如下：

1. 调用**acldvppCreateChannel**接口**创建图片数据处理的通道**。

   创建图片数据处理的通道前，需先调用**acldvppCreateChannelDesc**接口创建通道描述信息。

2. 实现JPEG图片解码功能前，若需要**申请Device上的内存**存放输入或输出数据，需调用**acldvppMalloc**申请内存。

在申请输出内存前，可根据存放JPEG图片数据的内存，调用**acldvppJpegPredictDecSize**接口预估JPEG图片解码后所需的输出内存的大小。

实际输出内存大小可能与调用**acldvppJpegPredictDecSize**接口预估的内存大小存在差异，如果用户需要获取解码后的实际输出内存大小，需调用**acldvppPicDesc**类型下的**acldvppGetPicDescSize**接口获取。

3. 调用**acldvppJpegDecodeAsync**异步接口进行**解码**。

   对于异步接口，还需调用**aclrtSynchronizeStream**接口阻塞程序运行，直到指定Stream中的所有任务都完成。

4. 在解码结束后，需及时调用**acldvppFree**接口**释放输入、输出内存**。
5. 调用**acldvppDestroyChannel**接口**销毁图片数据处理的通道**。

   销毁图片数据处理的通道后，再调用**acldvppDestroyChannelDesc**接口销毁通道描述信息。

示例代码

您可以从10.5.1 样例介绍中获取完整样例代码。

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1.AscendCL初始化
aclRet = aclInit(nullptr);

// 2.运行管理资源申请（依次申请Device、Context、Stream）
aclrtContext context_;
aclrtStream stream_;
aclrtSetDevice(0);
aclrtCreateContext(&context_, 0);
aclrtCreateStream(&stream_);

// 3.创建图片数据处理通道时的通道描述信息，dvppChannelDesc_是acldvppChannelDesc类型
dvppChannelDesc_ = acldvppCreateChannelDesc();

// 4.创建图片数据处理的通道。
aclError ret = acldvppCreateChannel(dvppChannelDesc_);

// 5. 申请输入内存（区分运行状态）
// 调用aclrtGetRunMode接口获取软件栈的运行模式，如果调用aclrtGetRunMode接口获取软件栈的运行模式为ACL_HOST，则需要通过aclrtMemcpy接口将输入图片数据传输到Device，数据传输完成后，需及时释放内存；否则直接申请并使用Device的内存
aclrtRunMode runMode;
ret = aclrtGetRunMode(&runMode);
if（runMode == ACL_HOST）{
    // 申请Host内存inputHostBuff，并将输入图片读入该地址，inDevBufferSize为读入图片大小
    void* inputHostBuff = nullptr;
    inputHostBuff = malloc(inDevBufferSize);
    // 将输入图片读入内存中，该自定义函数ReadPicFile由用户实现
    ReadPicFile(picName, inputHostBuff, inDevBufferSize);
    // 申请Device内存inDevBuffer_
    aclRet = acldvppMalloc(&inDevBuffer_, inDevBufferSize);
    // 通过aclrtMemcpy接口将输入图片数据传输到Device
    aclRet = aclrtMemcpy(inDevBuffer_, inDevBufferSize, inputHostBuff, inDevBufferSize,
ACL_MEMCPY_HOST_TO_DEVICE);

} else {
    // 申请Device输入内存inDevBuffer_
    ret = acldvppMalloc(&inDevBuffer_, inBufferSize);
    // 将输入图片读入内存中，该自定义函数ReadPicFile由用户实现
    ReadPicFile(picName, inDevBuffer_, inBufferSize);
}

// 6. 申请解码输出内存decodeOutDevBuffer_
```

```
// 预估JPEGD处理结果所需的内存大小
uint32_t decodeOutBufferSize = 0;
ret = acldvppJpegPredictDecSize(inputHostBuff, inDevBufferSize, PIXEL_FORMAT_YVU_SEMIPLANAR_420，
&decodeOutBufferSize)
ret = acldvppMalloc(&decodeOutDevBuffer_, decodeOutBufferSize)
// 及时释放内存
free(inputHostBuff);

// 7. 创建解码输出图片的描述信息，设置各属性值
// decodeOutputDesc是acldvppPicDesc类型
decodeOutputDesc_ = acldvppCreatePicDesc();
acldvppSetPicDescData(decodeOutputDesc_, decodeOutDevBuffer_);
acldvppSetPicDescFormat(decodeOutputDesc_, PIXEL_FORMAT_YUV_SEMIPLANAR_420);
acldvppSetPicDescWidth(decodeOutputDesc_, inputWidth_);
acldvppSetPicDescHeight(decodeOutputDesc_, inputHeight_);
acldvppSetPicDescWidthStride(decodeOutputDesc_, decodeOutWidthStride);
acldvppSetPicDescHeightStride(decodeOutputDesc_, decodeOutHeightStride);
acldvppSetPicDescSize(decodeOutputDesc_, decodeOutBufferSize);

// 8. 执行异步解码，再调用aclrtSynchronizeStream接口阻塞程序运行，直到指定Stream中的所有任务都完成
ret = acldvppJpegDecodeAsync(dvppChannelDesc_, inDevBuffer_, inDevBufferSize, decodeOutputDesc_,
stream_);
ret = aclrtSynchronizeStream(stream_);

// 9. 解码结束后，释放资源，包括解码输出图片的描述信息、解码输出内存、通道描述信息、通道等
acldvppDestroyPicDesc(decodeOutputDesc_);

if（runMode == ACL_HOST）{
    // 该模式下，由于处理结果在Device侧，因此需要调用内存复制接口传输结果数据后，再释放Device侧内存
    // 申请Host内存vpcOutHostBuffer
    void* vpcOutHostBuffer = nullptr;
    vpcOutHostBuffer = malloc(decodeOutBufferSize);
    // 通过aclrtMemcpy接口将Device的处理结果数据传输到Host
    aclRet = aclrtMemcpy(vpcOutHostBuffer, decodeOutBufferSize, decodeOutDevBuffer_,
decodeOutBufferSize, ACL_MEMCPY_DEVICE_TO_HOST);
    // 释放掉输入输出的device内存
    (void)acldvppFree(inDevBuffer_);
    (void)acldvppFree(decodeOutDevBuffer_);
    // 数据使用完成后，释放内存
    free(vpcOutHostBuffer);
} else {
    // 此时运行在device侧，处理结果也在Device侧，可以根据需要操作处理结果后，释放Device侧内存
    (void)acldvppFree(inDevBuffer_);
    (void)acldvppFree(decodeOutDevBuffer_);
}
acldvppDestroyChannel(dvppChannelDesc_);
(void)acldvppDestroyChannelDesc(dvppChannelDesc_);
dvppChannelDesc_ = nullptr;

// 10. 释放运行管理资源（依次释放Stream、Context、Device）
aclrtDestroyStream(stream_);
aclrtDestroyContext(context_);
aclrtResetDevice(0);

// 11.AscendCL去初始化
aclRet = aclFinalize();
// ....
```

5.4.3 JPEGE 图片编码

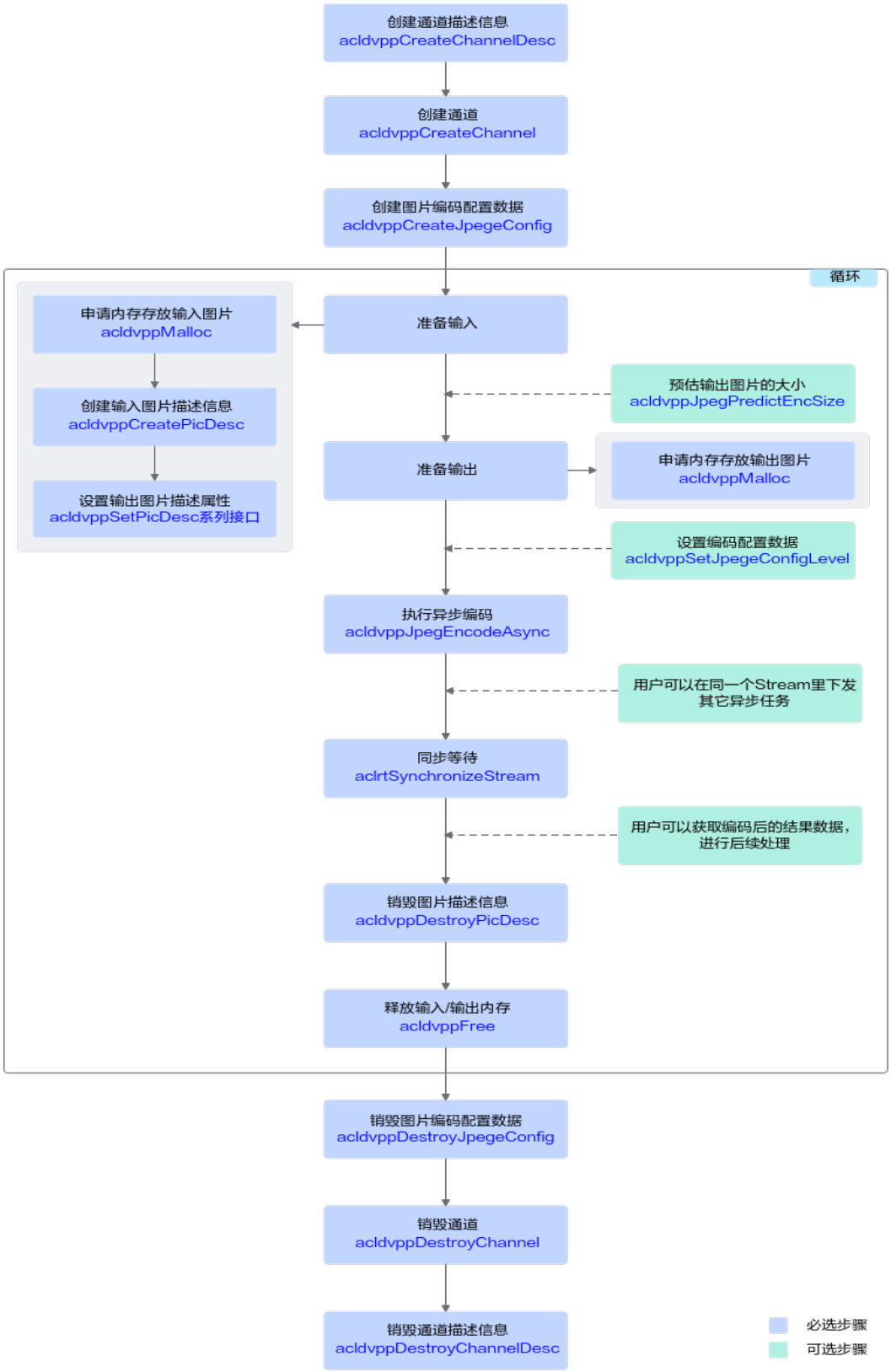
本节介绍JPEGE图片编码的接口调用流程，同时配合示例代码辅助理解该接口调用流程。

JPEGE（JPEG Encoder）负责完成图像编码功能，将YUV格式图片编码成.jpg图片。关于JPEGE功能的详细介绍及约束请参见**11.17.7.1 功能及约束说明**。

接口调用流程

开发应用时，如果涉及将YUV格式图片编码成JPEG压缩格式的图片文件，则应用程序中必须包含图片编码的代码逻辑，**关于图片编码的接口调用流程，请先参见3.2 AscendCL接口调用流程了解整体流程，再查看本节中的流程说明。**

图 5-8 JPEG 图片编码



当前系统支持将YUV格式图片编码成.jpg图片，关键接口的说明如下：

1. 调用**acldvppCreateChannel**接口**创建图片数据处理的通道**。

   创建图片数据处理的通道前，需先调用**acldvppCreateChannelDesc**接口创建通道描述信息。

2. 调用**acldvppCreateJpegeConfig**接口**创建图片编码配置数据**。
3. 实现JPEG图片编码功能前，若需要**申请Device上的内存**存放输入或输出数据，需调用**acldvppMalloc**申请内存。

   在申请输出内存前，可调用**acldvppJpegPredictEncSize**接口根据输入图片描述信息、图片编码配置数据可预估图片编码后所需的输出内存的大小。

   实际输出内存大小可能与调用**acldvppJpegPredictEncSize**接口预估的内存大小存在差异，如果用户需要获取编码后的实际输出内存大小，可通过**acldvppJpegEncodeAsync**接口的出参size获取。

4. 调用**acldvppJpegEncodeAsync**异步接口进行**编码**。

   对于异步接口，还需调用**aclrtSynchronizeStream**接口阻塞程序运行，直到指定Stream中的所有任务都完成。

5. 调用**acldvppDestroyJpegeConfig**接口**销毁图片编码配置数据**。
6. 在编码结束后，需及时调用**acldvppFree**接口**释放输入、输出内存**。
7. 调用**acldvppDestroyChannel**接口**销毁图片数据处理的通道**。

   销毁图片数据处理的通道后，再调用**acldvppDestroyChannelDesc**接口销毁通道描述信息。

示例代码

您可以从10.5.1 样例介绍中获取完整样例代码。

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1.AscendCL初始化
aclRet = aclInit(nullptr);

// 2.运行管理资源申请（依次申请Device、Context、Stream）
aclrtContext context_;
aclrtStream stream_;
aclrtSetDevice(0);
aclrtCreateContext(&context_, 0);
aclrtCreateStream(&stream_);

// 3.创建图片数据处理通道时的通道描述信息，dvppChannelDesc_是acldvppChannelDesc类型
dvppChannelDesc_ = acldvppCreateChannelDesc();

// 4.创建图片数据处理的通道
aclRet = acldvppCreateChannel(dvppChannelDesc_);

// 5.申请输入内存（区分运行状态）
// 调用aclrtGetRunMode接口获取软件栈的运行模式，如果调用aclrtGetRunMode接口获取软件栈的运行模式为ACL_HOST，则需要通过aclrtMemcpy接口将输入图片数据传输到Device，数据传输完成后，需及时释放内存；否则直接申请并使用Device的内存
aclrtRunMode runMode;
ret = aclrtGetRunMode(&runMode);
// inputPicWidth、inputPicHeight分别表示图片的对齐后宽、对齐后高，此处以YUV420SP格式的图片为例
uint32_t PicBufferSize = inputPicWidth * inputPicHeight * 3 / 2;
if（runMode == ACL_HOST）{
    // 申请Host内存vpcInHostBuffer
    void* vpcInHostBuffer = nullptr;
    vpcInHostBuffer = malloc(PicBufferSize);
    // 将输入图片读入内存中，该自定义函数ReadPicFile由用户实现
    ReadPicFile(picName, vpcInHostBuffer, PicBufferSize);
    // 申请Device内存inDevBuffer_
```

```
    aclRet = acldvppMalloc(&inDevBuffer_, PicBufferSize);
    // 通过aclrtMemcpy接口将输入图片数据传输到Device
    aclRet = aclrtMemcpy(inDevBuffer_, PicBufferSize, vpcInHostBuffer, PicBufferSize,
ACL_MEMCPY_HOST_TO_DEVICE);
    // 数据传输完成后，及时释放内存
    free(vpcInHostBuffer);
} else {
    // 申请Device输入内存inDevBuffer_
    ret = acldvppMalloc(&inDevBuffer_, PicBufferSize);
    // 将输入图片读入内存中，该自定义函数ReadPicFile由用户实现
    ReadPicFile(picName, inDevBuffer_, PicBufferSize);
}

// 6. 创建编码输入图片的描述信息，并设置各属性值
// encodeInputDesc_是acldvppPicDesc类型
encodeInputDesc_ = acldvppCreatePicDesc();
acldvppSetPicDescData(encodeInputDesc_, reinterpret_cast<void *>(inDevBuffer_));
acldvppSetPicDescFormat(encodeInputDesc_, PIXEL_FORMAT_YUV_SEMIPLANAR_420);
acldvppSetPicDescWidth(encodeInputDesc_, inputWidth_);
acldvppSetPicDescHeight(encodeInputDesc_, inputHeight_);
acldvppSetPicDescWidthStride(encodeInputDesc_, encodeInWidthStride);
acldvppSetPicDescHeightStride(encodeInputDesc_, encodeInHeightStride);
acldvppSetPicDescSize(encodeInputDesc_, inDevBufferSizeE_);

// 7. 创建图片编码配置数据，设置编码质量
// 编码质量范围[0, 100]，其中level 0编码质量与level 100差不多，而在[1, 100]内数值越小输出图片质量越差。
jpegeConfig_ = acldvppCreateJpegeConfig();
acldvppSetJpegeConfigLevel(jpegeConfig_, 100);

// 8. 申请输出内存，申请Device内存encodeOutBufferDev_,存放编码后的输出数据
uint32_t outBufferSize= 0;
ret = acldvppJpegPredictEncSize(encodeInputDesc_, jpegeConfig_, &outBufferSize);
ret = acldvppMalloc(&encodeOutBufferDev_, outBufferSize)

// 9. 执行异步编码，再调用aclrtSynchronizeStream接口阻塞程序运行，直到指定Stream中的所有任务都完成
aclRet = acldvppJpegEncodeAsync(dvppChannelDesc_, encodeInputDesc_, encodeOutBufferDev_,
          &outBufferSize, jpegeConfig_, stream_);
aclRet = aclrtSynchronizeStream(stream_);

// 10. 编码结束后，释放资源，包括编码输入/输出图片的描述信息、编码输入/输出内存、通道描述信息、通道等
acldvppDestroyPicDesc(encodeInputDesc_);

if（runMode == ACL_HOST）{
    // 该模式下，由于处理结果在Device侧，因此需要调用内存复制接口传输结果数据后，再释放Device侧内存
    // 申请Host内存outputHostBuffer
    void* outputHostBuffer = nullptr;
    outputHostBuffer = malloc(outBufferSize_);
    // 通过aclrtMemcpy接口将Device的处理结果数据传输到Host
    aclRet = aclrtMemcpy(outputHostBuffer, outBufferSize_, encodeOutBufferDev_, outBufferSize_,
ACL_MEMCPY_DEVICE_TO_HOST);
    // 释放掉输入输出的device内存
    (void)acldvppFree(inputDevBuff);
    (void)acldvppFree(encodeOutBufferDev_);
    // 数据使用完成后，释放内存
    free(outputHostBuffer);
} else {
    // 此时运行在device侧，处理结果也在Device侧，可以根据需要操作处理结果后，释放Device侧内存
    (void)acldvppFree(inputDevBuff);
    (void)acldvppFree(encodeOutBufferDev_);
}
acldvppDestroyChannel(dvppChannelDesc_);
(void)acldvppDestroyChannelDesc(dvppChannelDesc_);
dvppChannelDesc_ = nullptr;

// 11. 释放运行管理资源（依次释放Stream、Context、Device）
aclrtDestroyStream(stream_);
aclrtDestroyContext(context_);
aclrtResetDevice(0);
```

```
// 12.AscendCL去初始化
aclRet = aclFinalize();
// ....
```

5.4.4 PNGD 图片解码

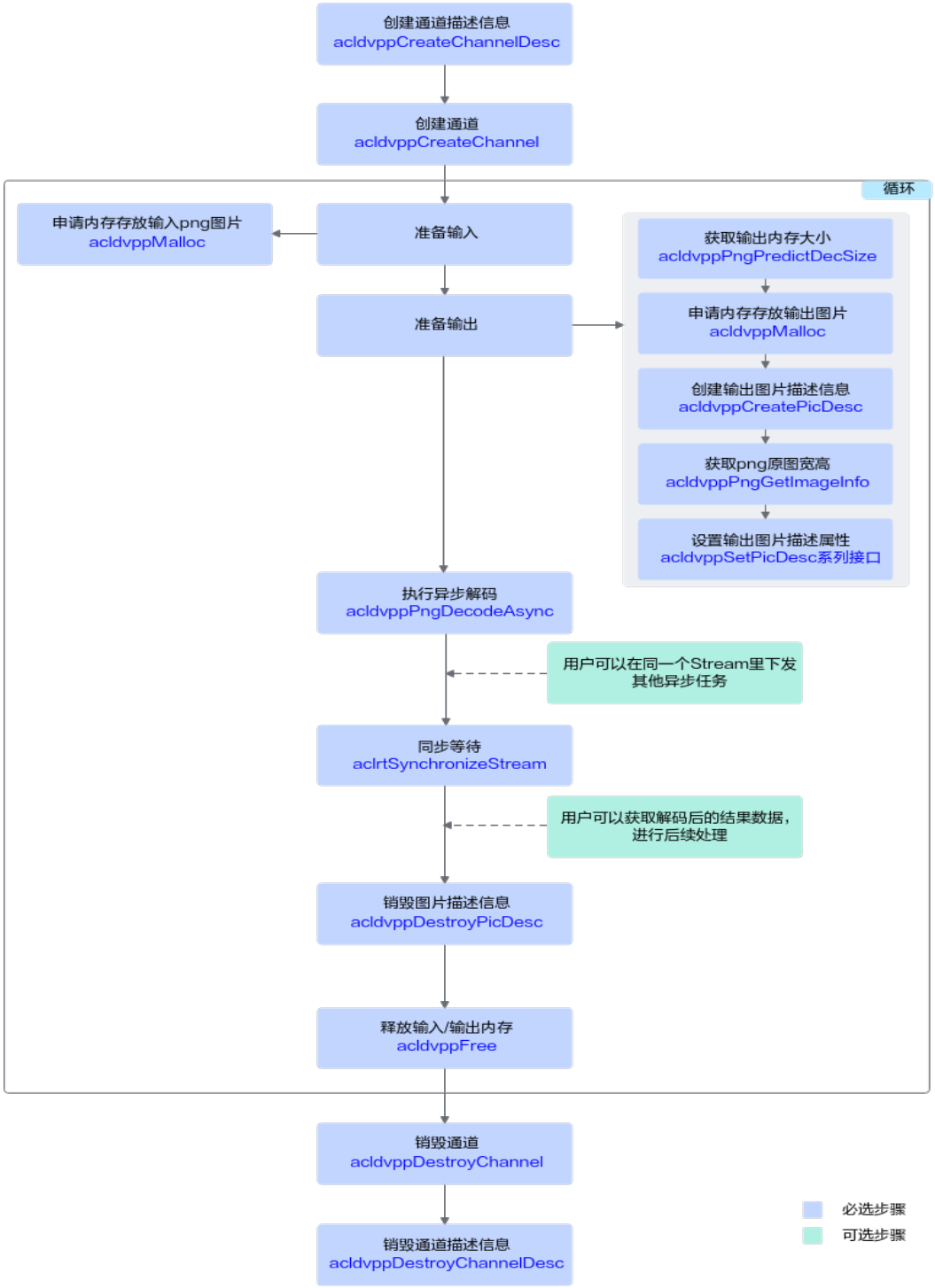
本节介绍PNGD图片编码的接口调用流程，同时配合示例代码辅助理解该接口调用流程。

PNGD（PNG decoder）负责PNG格式图片的解码。关于PNGD功能的详细介绍及约束请参见**11.17.8.1 功能及约束说明**。

接口调用流程

开发应用时，如果涉及对PNG图片的解码，则应用程序中必须包含图片解码的代码逻辑，**关于图片解码的接口调用流程，请先参见3.2 AscendCL接口调用流程了解整体流程，再查看本节中的流程说明。**

图 5-9 PNG 图片解码



当前系统支持png图片的解码，支持输出RGB、RGBA编码格式的图片，关键接口的说明如下：

1. 调用**acldvppCreateChannel**接口**创建图片数据处理的通道**。

   创建图片数据处理的通道前，需先调用**acldvppCreateChannelDesc**接口创建通道描述信息。

2. 实现PNG图片解码功能前，若需要**申请Device上的内存**存放输入或输出数据，需调用**acldvppMalloc**申请内存。

   在申请输出内存前，可调用**acldvppPngPredictDecSize**接口根据存放png图片数据的内存计算出png图片解码后所需的输出内存的大小。
3. 调用**acldvppPngDecodeAsync**异步接口进行**解码**。

   对于异步接口，还需调用**aclrtSynchronizeStream**接口阻塞程序运行，直到指定Stream中的所有任务都完成。
4. 在解码结束后，需及时调用**acldvppFree**接口**释放输入、输出内存**。
5. 调用**acldvppDestroyChannel**接口**销毁图片数据处理的通道**。

   销毁图片数据处理的通道后，再调用**acldvppDestroyChannelDesc**接口销毁通道描述信息。

示例代码

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1.AscendCL初始化
aclRet = aclInit(nullptr);

// 2.运行管理资源申请（依次申请Device、Context、Stream）
aclrtContext context_;
aclrtStream stream_;
aclrtSetDevice(0);
aclrtCreateContext(&context_, 0);
aclrtCreateStream(&stream_);

// 3.创建图片数据处理通道时的通道描述信息，dvppChannelDesc_是acldvppChannelDesc类型
dvppChannelDesc_ = acldvppCreateChannelDesc();

// 4.创建图片数据处理的通道。
aclError ret = acldvppCreateChannel(dvppChannelDesc_);

// 5. 申请输入内存（区分运行状态）
// 调用aclrtGetRunMode接口获取软件栈的运行模式，如果调用aclrtGetRunMode接口获取软件栈的运行模式为ACL_HOST，则需要通过aclrtMemcpy接口将输入图片数据传输到Device，数据传输完成后，需及时释放内存；否则直接申请并使用Device的内存
aclrtRunMode runMode;
ret = aclrtGetRunMode(&runMode);
if（runMode == ACL_HOST）{
    // 申请Host内存inputHostBuff，并将输入图片读入该地址，inDevBufferSize为读入图片大小
    void* inputHostBuff = nullptr;
    inputHostBuff = malloc(inDevBufferSize);
    // 将输入图片读入内存中，该自定义函数ReadPicFile由用户实现
    ReadPicFile(picName, inputHostBuff, inDevBufferSize);
    // 申请Device内存inDevBuffer_
    aclRet = acldvppMalloc(&inDevBuffer_, inDevBufferSize);
    // 通过aclrtMemcpy接口将输入图片数据传输到Device
    aclRet = aclrtMemcpy(inDevBuffer_, inDevBufferSize, inputHostBuff, inDevBufferSize,
ACL_MEMCPY_HOST_TO_DEVICE);

} else {
    // 申请Device输入内存inDevBuffer_
    ret = acldvppMalloc(&inDevBuffer_, inDevBufferSize);
    // 将输入图片读入内存中，该自定义函数ReadPicFile由用户实现
    ReadPicFile(picName, inDevBuffer_, inDevBufferSize);
}

// 6. 申请解码输出内存decodeOutDevBuffer_
// 计算PNGD处理结果所需的内存大小
uint32_t decodeOutBufferSize = 0;
ret = acldvppPngPredictDecSize(inputHostBuff, inDevBufferSize, PIXEL_FORMAT_RGB_888，
&decodeOutBufferSize)
```

```
ret = acldvppMalloc(&decodeOutDevBuffer_, decodeOutBufferSize)
// 及时释放内存
free(inputHostBuff);

// 7. 创建解码输出图片的描述信息，设置各属性值
// decodeOutputDesc是acldvppPicDesc类型
decodeOutputDesc_ = acldvppCreatePicDesc();
acldvppSetPicDescData(decodeOutputDesc_, decodeOutDevBuffer_);
acldvppSetPicDescFormat(decodeOutputDesc_, PIXEL_FORMAT_RGB_888);
acldvppSetPicDescSize(decodeOutputDesc_, decodeOutBufferSize);

// 8. 执行异步解码，再调用aclrtSynchronizeStream接口阻塞程序运行，直到指定Stream中的所有任务都完成
ret = acldvppPngDecodeAsync(dvppChannelDesc_, inDevBuffer_, inDevBufferSize, decodeOutputDesc_,
stream_);
ret = aclrtSynchronizeStream(stream_);

// 9. 解码结束后，释放资源，包括解码输出图片的描述信息、解码输出内存、通道描述信息、通道等
acldvppDestroyPicDesc(decodeOutputDesc_);

if（runMode == ACL_HOST）{
    // 该模式下，由于处理结果在Device侧，因此需要调用内存复制接口传输结果数据后，再释放Device侧内存
    // 申请Host内存OutHostBuffer
    void* OutHostBuffer = nullptr;
    OutHostBuffer = malloc(decodeOutBufferSize);
    // 通过aclrtMemcpy接口将Device的处理结果数据传输到Host
    aclRet = aclrtMemcpy(OutHostBuffer, decodeOutBufferSize, decodeOutDevBuffer_,
decodeOutBufferSize, ACL_MEMCPY_DEVICE_TO_HOST);
    // 释放掉输入输出的device内存
    (void)acldvppFree(inDevBuffer_);
    (void)acldvppFree(decodeOutDevBuffer_);
    // 数据使用完成后，释放内存
    free(OutHostBuffer);
} else {
    // 此时运行在device侧，处理结果也在Device侧，可以根据需要操作处理结果后，释放Device侧内存
    (void)acldvppFree(inDevBuffer_);
    (void)acldvppFree(decodeOutDevBuffer_);
}
acldvppDestroyChannel(dvppChannelDesc_);
(void)acldvppDestroyChannelDesc(dvppChannelDesc_);
dvppChannelDesc_ = nullptr;

// 10. 释放运行管理资源（依次释放Stream、Context、Device）
aclrtDestroyStream(stream_);
aclrtDestroyContext(context_);
aclrtResetDevice(0);

// 11.AscendCL去初始化
aclRet = aclFinalize();
// ....
```

5.4.5 VDEC 视频解码

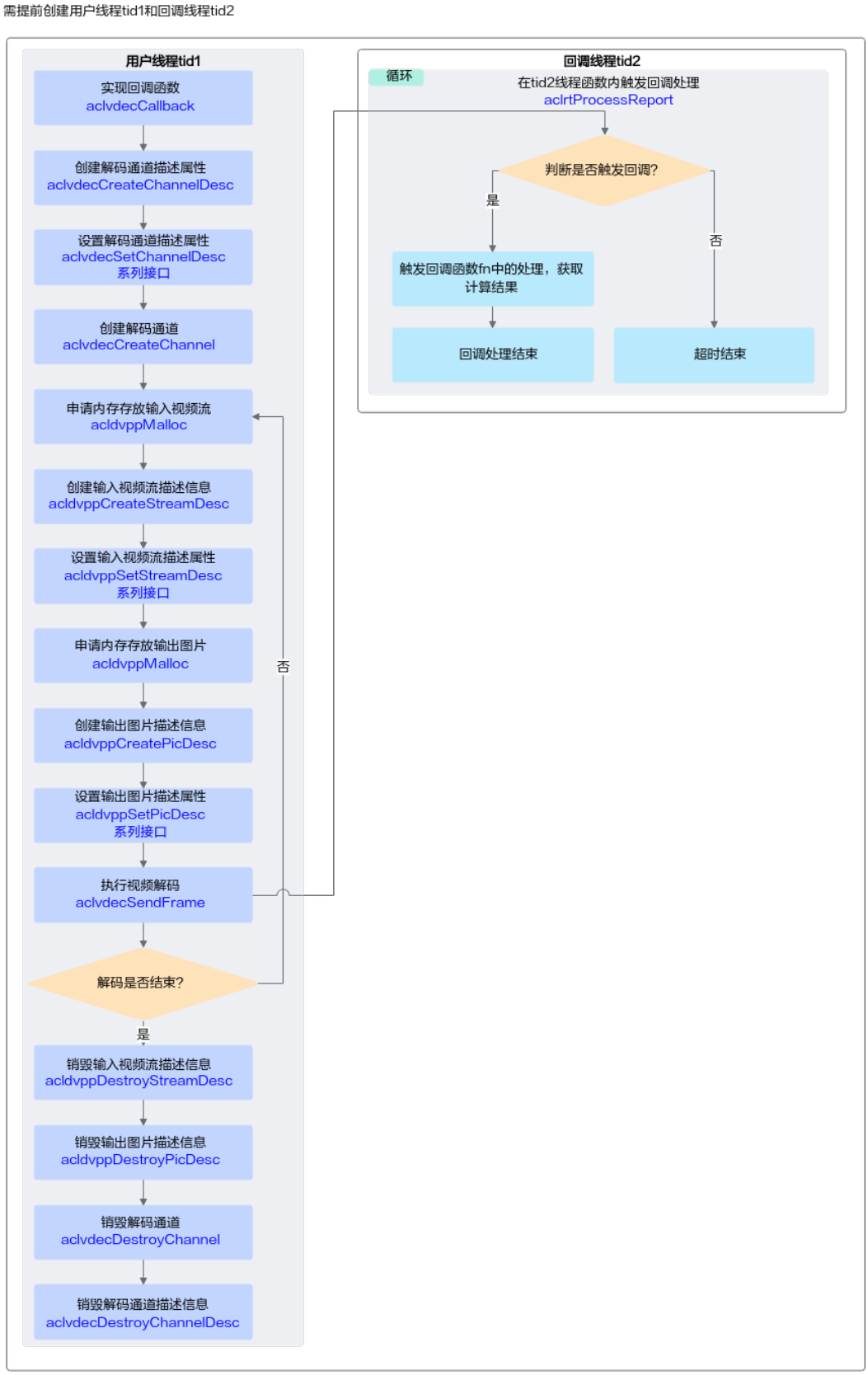
本节介绍VDEC视频编码的接口调用流程，同时配合示例代码辅助理解该接口调用流程。

VDEC（Video Decoder）负责将H264/H265格式的视频码流解码为YUV/RGB格式的图片。关于VDEC功能的详细介绍及约束请参见**11.17.9.1 功能及约束说明**。

接口调用流程

开发应用时，如果涉及视频解码，则应用程序中必须包含视频解码的代码逻辑，**关于视频解码的接口调用流程，请先参见3.2 AscendCL接口调用流程了解整体流程，再查看本节中的流程说明。**

图 5-10 视频解码流程



实现视频的解码，关键接口的说明如下：

版权所有 © 华为技术有限公司

1. **调用aclvdecCreateChannel接口创建视频解码处理的通道。**
   - 创建视频解码处理通道前，需先执行以下操作：
     i. 调用**aclvdecCreateChannelDesc**接口**创建通道描述信息**。
     ii. 调用**aclvdecSetChannelDesc系列接口设置通道描述信息的属性**，包括**解码通道号、线程、回调函数、视频编码协议**等，其中：
       1) 回调函数需由用户提前创建，用于在视频解码后，获取解码数据，并及时释放相关资源，回调函数的原型前参见**11.17.9.5 aclvdecCallback**。

       在回调函数内，用户需调用**acldvppGetPicDescRetCode**接口获取retCode返回码判断是否解码成功，retCode为0表示解码成功，为1表示解码失败。如果解码失败，需要根据日志中的返回码判断具体的问题，返回码请参见**返回码说明**。

       解码结束后，建议用户在回调函数内及时释放VDEC的输入码流内存、输出图片内存以及相应的视频码流描述信息、图片描述信息。
       2) 线程需由用户提前创建，并自定义线程函数，在线程函数内调用**aclrtProcessReport**接口，等待指定时间后，触发**1.ii.1)**中的回调函数。

       说明

       如果不调用**aclvdecSetChannelDescOutPicFormat**接口设置输出格式，则默认使用YUV420SP NV12。
   - **aclvdecCreateChannel**接口内部封装了如下接口，无需用户单独调用：
     i. **aclrtCreateStream**接口：显式创建Stream，VDEC内部使用。
     ii. **aclrtSubscribeReport**接口：指定处理Stream上回调函数的线程，回调函数和线程是由用户调用**aclvdecSetChannelDesc系列接口**时指定的。
2. **调用aclvdecSendFrame接口将视频码流解码成YUV420SP格式的图片。**
   - **视频解码前**，需先执行以下操作：
     - 调用**acldvppCreateStreamDesc**接口创建输入视频码流描述信息，并调用**acldvppSetStreamDesc系列接口**设置输入视频的内存地址、内存大小、码流格式等属性。
     - 调用**acldvppCreatePicDesc**接口创建输出图片描述信息，并调用**acldvppSetPicDesc系列接口**设置输出图片的内存地址、内存大小、图片格式等属性。
   - **视频解码时：**

     aclvdecSendFrame接口内部封装了**aclrtLaunchCallback**接口，用于在Stream的任务队列中增加一个需要执行的回调函数。用户无需单独调用**aclrtLaunchCallback**接口。
   - **视频解码后**，视频解码的结果数据通过回调函数获取：

     获取解码数据前，先获取retCode的值，判断解码是否成功，0表示解码成功，1表示解码失败。如果解码失败，需要根据日志中的返回码判断具体的问题，返回码请参见**返回码说明**。

     如果用户需要获取解码的帧序号，则可以在**aclvdecSendFrame**接口的userData参数处定义，然后解码的帧序号可以通过userData参数传递给VDEC的回调函数，用于确定回调函数中处理的是第几帧数据。

如果不想获取某一帧的解码结果， 可以调用**aclvdecSendSkippedFrame**接口， 将待解码的码流（输入内存）传到解码器进行解码，此时，解码结果最终不会输出，解码完成的回调函数中返回的output为nullptr。

3. **调用aclvdecDestroyChannel接口销毁视频处理的通道。**
   - 系统会等待已发送帧解码完成且用户的回调函数处理完成后再销毁通道。
   - **aclvdecDestroyChannel**接口内部封装了如下接口，无需用户单独调用：
     - **aclrtUnSubscribeReport**接口：取消线程注册（Stream上的回调函数不再由指定线程处理）。
     - **aclrtDestroyStream**接口：销毁Stream。
   - 销毁通道后，需调用**aclvdecDestroyChannelDesc**接口销毁通道描述信息。
   - 销毁通道描述信息后，用户才可以销毁**1.ii.2)**中创建的线程。

示例代码

您可以从10.6.1 样例介绍中获取完整样例代码。

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1.AscendCL初始化
aclRet = aclInit(nullptr);

// 2.运行管理资源申请（依次申请Device、Context、Stream）
aclrtContext context_;
aclrtStream stream_;
aclrtSetDevice(0);
aclrtCreateContext(&context_, 0);
aclrtCreateStream(&stream_);

// 3.创建回调函数
void callback(acldvppStreamDesc *input, acldvppPicDesc *output, void *userdata)
{
    static int count = 1;
    if (output != nullptr) {
        // 获取VDEC解码的输出内存，调用自定义函数WriteToFile将输出内存中的数据写入文件后，再调用
acldvppFree接口释放输出内存
        void *vdecOutBufferDev = acldvppGetPicDescData(output);
        if (vdecOutBufferDev != nullptr) {
            // 0: vdec success; others, vdec failed
            // retCode为0表示解码成功，为1表示解码失败
            int retCode = acldvppGetPicDescRetCode(output);
            if (retCode == 0) {
                // process task: write file
                uint32_t size = acldvppGetPicDescSize(output);
                std::string fileNameSave = "outdir/image" + std::to_string(count);
                // vdec输出结果在device侧，在WriteToFile方法中进行下述处理
                if (!Utils::WriteToFile(fileNameSave.c_str(), vdecOutBufferDev, size)) {
                    ERROR_LOG("write file failed.");
                }
            } else {
                ERROR_LOG("vdec decode frame failed.");
            }

            // free output vdecOutBufferDev
            aclError ret = acldvppFree(vdecOutBufferDev);
        }
        // 释放acldvppPicDesc类型的数据，表示解码后输出图片描述数据
        aclError ret = acldvppDestroyPicDesc(output);
    }

    // free input vdecInBufferDev and destroy stream desc
```

```
    if (input != nullptr) {
        void *vdecInBufferDev = acldvppGetStreamDescData(input);
        if (vdecInBufferDev != nullptr) {
            aclError ret = acldvppFree(vdecInBufferDev);
        }
        // 释放acldvppStreamDesc类型的数据，表示解码的输入码流描述数据
        aclError ret = acldvppDestroyStreamDesc(input);
    }

    INFO_LOG("success to callback %d.", count);
    count++;
}

// 4.创建视频码流处理通道时的通道描述信息，设置视频处理通道描述信息的属性，其中线程、callback回调函数
需要用户提前创建。
// vdecChannelDesc_是aclvdecChannelDesc类型
vdecChannelDesc_ = aclvdecCreateChannelDesc();
ret = aclvdecSetChannelDescChannelId(vdecChannelDesc_, 10);
ret = aclvdecSetChannelDescThreadId(vdecChannelDesc_, threadId_);
ret = aclvdecSetChannelDescCallback(vdecChannelDesc_, callback);
// 示例中使用的是H265_MAIN_LEVEL视频编码协议
ret = aclvdecSetChannelDescEnType(vdecChannelDesc_, static_cast<acldvppStreamFormat>(enType_));
// 示例中使用的是PIXEL_FORMAT_YVU_SEMIPLANAR_420
ret = aclvdecSetChannelDescOutPicFormat(vdecChannelDesc_, static_cast<acldvppPixelFormat>(format_));

// 5.创建视频码流处理的通道
ret = aclvdecCreateChannel(vdecChannelDesc_);

// 6.申请输入码流内存（区分运行状态）
// 调用aclrtGetRunMode接口获取软件栈的运行模式，如果调用aclrtGetRunMode接口获取软件栈的运行模式
为ACL_HOST，则需要通过aclrtMemcpy接口将输入图片数据传输到Device，数据传输完成后，需及时释放内
存；否则直接申请并使用Device的内存
aclrtRunMode runMode;
ret = aclrtGetRunMode(&runMode);
if（runMode == ACL_HOST）{
    // 申请Host内存inputHostBuff
    void* inputHostBuff= nullptr;
    // inBufferSize_为输入码流大小
    inputHostBuff= malloc(inBufferSize_);
    // 将输入图片读入内存中，该自定义函数ReadPicFile由用户实现
    ReadPicFile(picName, inputHostBuff, inBufferSize_);
    // 申请Device内存inBufferDev_
    aclRet = acldvppMalloc(&inBufferDev_, inBufferSize_);
    // 通过aclrtMemcpy接口将输入图片数据传输到Device
    aclRet = aclrtMemcpy(inBufferDev_, inBufferSize_, inputHostBuff, inBufferSize_,
ACL_MEMCPY_HOST_TO_DEVICE);
    // 数据传输完成后，及时释放内存
    free(inputHostBuff);
} else {
    // 申请Device输入内存dataDev， StreamBufferSize为输入码流大小
    ret = acldvppMalloc(&inBufferDev_, inBufferSize_);
    // 将输入图片读入内存中，该自定义函数ReadPicFile由用户实现
    ReadPicFile(picName, inBufferDev_, inBufferSize_);
}

// 7.循环10次执行视频解码，输出10张YUV420SP NV12格式的图片
int rest_len = 10;
int32_t count = 0;
while (rest_len > 0) {
    // 7.1 创建输入视频码流描述信息，设置码流信息的属性
    streamInputDesc_ = acldvppCreateStreamDesc();
    // inBufferDev_表示Device存放输入视频数据的内存，inBufferSize_表示内存大小
    ret = acldvppSetStreamDescData(streamInputDesc_, inBufferDev_);
    ret = acldvppSetStreamDescSize(streamInputDesc_, inBufferSize_);

    // 7.2 申请Device内存picOutBufferDev_，用于存放VDEC解码后的输出数据
    ret = acldvppMalloc(&picOutBufferDev_, size);

    // 7.3 创建输出图片描述信息，设置图片描述信息的属性
```

```
    // picOutputDesc_是acldvppPicDesc类型
    picOutputDesc_ = acldvppCreatePicDesc();
    ret = acldvppSetPicDescData(picOutputDesc_, picOutBufferDev_);
    ret = acldvppSetPicDescSize(picOutputDesc_, size);
    ret = acldvppSetPicDescFormat(picOutputDesc_, static_cast<acldvppPixelFormat>(format_));

    // 7.4 执行视频码流解码，解码每帧数据后，系统自动调用callback回调函数将解码后的数据写入文件，再及时释放相关资源
    ret = aclvdecSendFrame(vdecChannelDesc_, streamInputDesc_, picOutputDesc_, nullptr, nullptr);
    // ......
    ++count;
    rest_len = rest_len - 1;
    // ......
}

// 8.释放图片处理通道、图片描述信息
ret = aclvdecDestroyChannel(vdecChannelDesc_);
aclvdecDestroyChannelDesc(vdecChannelDesc_);

// 9. 释放运行管理资源（依次释放Stream、Context、Device）
aclrtDestroyStream(stream_);
aclrtDestroyContext(context_);
aclrtResetDevice(0);

// 10.AscendCL去初始化
aclRet = aclFinalize();

// ......
```

返回码说明

表 5-1 返回码列表

返回码	含义	可能原因及解决方法
AICPU_DVPP_KERNEL_STATE_SUCCESS = 0	解码成功。	-
AICPU_DVPP_KERNEL_STATE_FAILED = 1	其它错误。	-
AICPU_DVPP_KERNEL_STATE_DVPP_ERROR = 2	AscendCL内部调用其它模块的接口失败。	-
AICPU_DVPP_KERNEL_STATE_PARAM_INVALID = 3	参数校验失败。	请检查接口的参数是否符合接口要求。
AICPU_DVPP_KERNEL_STATE_OUTPUT_SIZE_INVALID = 4	输出内存大小校验失败。	请检查输出内存大小是否符合接口要求。
AICPU_DVPP_KERNEL_STATE_INTERNAL_ERROR = 5	系统内部错误。	-
AICPU_DVPP_KERNEL_STATE_QUEUE_FULL = 6	系统内部队列满。	-
AICPU_DVPP_KERNEL_STATE_QUEUE_EMPTY = 7	系统内部队列空。	-

返回码	含义	可能原因及解决方法
AICPU_DVPP_KERNEL_STATE_QUEUE_NOT_EXIST = 8	系统内部队列不存在。	-
AICPU_DVPP_KERNEL_STATE_GET_CONTEX_FAILED = 9	获取系统内部上下文失败。	-
AICPU_DVPP_KERNEL_STATE_SUBMIT_EVENT_FAILED = 10	提交系统内部事件失败。	-
AICPU_DVPP_KERNEL_STATE_MEMORY_FAILED = 11	系统内部申请内存失败。	请检查系统是否有可用内存。
AICPU_DVPP_KERNEL_STATE_SEND_NOTIFY_FAILED = 12	发送系统内部通知失败。	-
AICPU_DVPP_KERNEL_STATE_VPC_OPERATE_FAILED = 13	系统内部接口处理失败。	-
AICPU_DVPP_KERNEL_STATE_CHANNEL_ABNORMAL = 14	当前通道异常。	
ERR_INVALID_STATE = 0x10001	VDEC解码器状态异常错误。	-
ERR_HARDWARE = 0x10002	硬件错误，包含解码器启动、执行、停止等异常。	-
ERR_SCD_CUT_FAIL = 0x10003	将视频码流分解成多帧图片异常。	请检查输入的视频流数据是否正确。
ERR_VDM_DECODE_FAIL = 0x10004	解码某一帧异常。	请检查输入的视频流数据是否正确。
ERR_ALLOC_MEM_FAIL = 0x10005	内部申请内存失败。	请检查系统是否有可用内存，**若忽略错误，则可能导致用户持续送入码流却无任何解码结果输出**。
ERR_ALLOC_DYNAMIC_MEM_FAIL = 0x10006	包括输入视频分辨率超范围、内部动态申请内存失败等异常。	请检查输入视频流的分辨率、系统是否有可用内存，**若忽略错误，则可能导致用户持续送入码流却无任何解码结果输出**。

返回码	含义	可能原因及解决方法
ERR_ALLOC_IN_OR_OUT_PORT_MEM_FAIL = 0x10007	系统内部申请VDEC的输入、输出buffer异常。	请检查系统是否有可用内存，**若忽略错误，则可能导致用户持续送入码流却无任何解码结果输出**。
ERR_BITSTREAM = 0x10008	码流错误（如语法解析失败、重发eos或首帧即发送eos）。	请检查输入的视频流数据是否正确。
ERR_VIDEO_FORMAT = 0x10009	输入视频格式错误。	请检查输入视频的格式是否为h264或h265。
ERR_IMAGE_FORMAT = 0x1000a	输出格式配置错误。	请检查输出图像的格式是否为nv12或nv21。
ERR_CALLBACK = 0x1000b	回调函数为空。	请检查配置的回调函数是否为空。
ERR_INPUT_BUFFER = 0x1000c	输入内存为空。	请检输入内存是否为空。
ERR_INBUF_SIZE = 0x1000d	输入内存大小<=0。	请检查输入内存大小是否小于等于0。
ERR_THREAD_CREATE_FBD_FAIL = 0x1000e	系统内部将解码结果通过回调函数返回给用户的线程异常。	请检查系统中资源（例如：线程、内存等）是否可用。
ERR_CREATE_INSTANCE_FAIL = 0x1000f	创建解码实例失败。	-
ERR_INIT_DECODER_FAIL = 0x10010	初始化解码器失败，例如解码实例个数超出范围（最大16）。	-
ERR_GET_CHANNEL_HANDLE_FAIL = 0x10011	系统内部获取某路视频流的解码句柄失败。	-
ERR_COMPONENT_SET_FAIL = 0x10012	系统内部设置解码实例异常。	请检查解码的入参值是否正确，例如输入视频格式video_format、输出帧格式image_format等。

返回码	含义	可能原因及解决方法
ERR_COMPARE_NAME_FAIL = 0x10013	系统内部设置解码实例名称异常。	请检查解码的入参值是否正确，例如输入视频格式video_format、输出帧格式image_format等。
ERR_OTHER = 0x10014	其它错误。	-
ERR_DECODE_NOPIC = 0x20000	隔行码流场景下使用，隔行码流每帧发送两场，解码时其中一块无图像输出，属于正常现象，会返回该错误码；隔行码流的解码输出数据都在奇数场对应的输出buffer中。	-
0x20001	参考帧个数设置错误。	请检查码流实际参考帧个数与用户设置的参考帧个数是否一致。Atlas 推理系列产品，默认参考帧个数为8。
0x20002	VDEC解码帧存大小设置错误。	请检查输入码流实际宽、高与用户设置的宽、高是否一致。
0xA0058001	无效的Device ID。	请检查Device ID。
0xA0058002	无效的channel ID。	请检查传入接口的通道号是否正确，或者检查通道总数是否达到上限。
0xA0058003	参数不合法，例如不合法的枚举值。	请根据日志检查出错的参数。
0xA0058004	资源已存在。	请检查是否重复创建通道。
0xA0058005	通道资源不存在。	请检查是否使用了不存在的通道号或通道句柄。
0xA0058006	函数参数中有空指针。	请根据日志检查接口的入参。

返回码	含义	可能原因及解决方法
0xA0058007	使能系统、Device或通道前未配置对应的参数。	请根据日志检查接口的入参。
0xA0058008	不支持的参数或者功能。	请根据日志检查接口的入参。
0xA0058009	该操作不允许，如试图修改静态配置参数。	-
0xA005800C	分配内存失败，如系统内存不足。	请检查系统是否有可用内存，**若忽略错误，则可能导致用户持续送入码流却无任何解码结果输出**。
0xA005800D	分配缓存失败，如申请的数据缓冲区太大。	-
0xA005800E	缓冲区中无数据。	系统未完成解码，缓冲区中无解码结果数据，需等待缓冲区中有数据后，再尝试获取数据。
0xA005800F	缓冲区中数据满。	用户发送输入码流数据的速度太快，导致输入缓冲区数据满，请尝试降低发送输入码流数据的速度，或者在创建通道时将缓冲区大小设置为较大值。
0xA0058010	系统没有初始化或者相关依赖的模块没有加载。	请检查是否调用系统初始化接口。
0xA0058011	地址错误。	-
0xA0058012	系统忙。	请检查VDEC解码总路数是否达到上限。
0xA0058013	缓存小于实际需要的大小。	-
0xA0058014	硬件或软件处理超时。	-
0xA0058015	内部系统错误。	-

返回码	含义	可能原因及解决方法
0xA005803F	最大的返回码，该模块的错误码必须小于该值。	-

5.4.6 VENC 视频编码

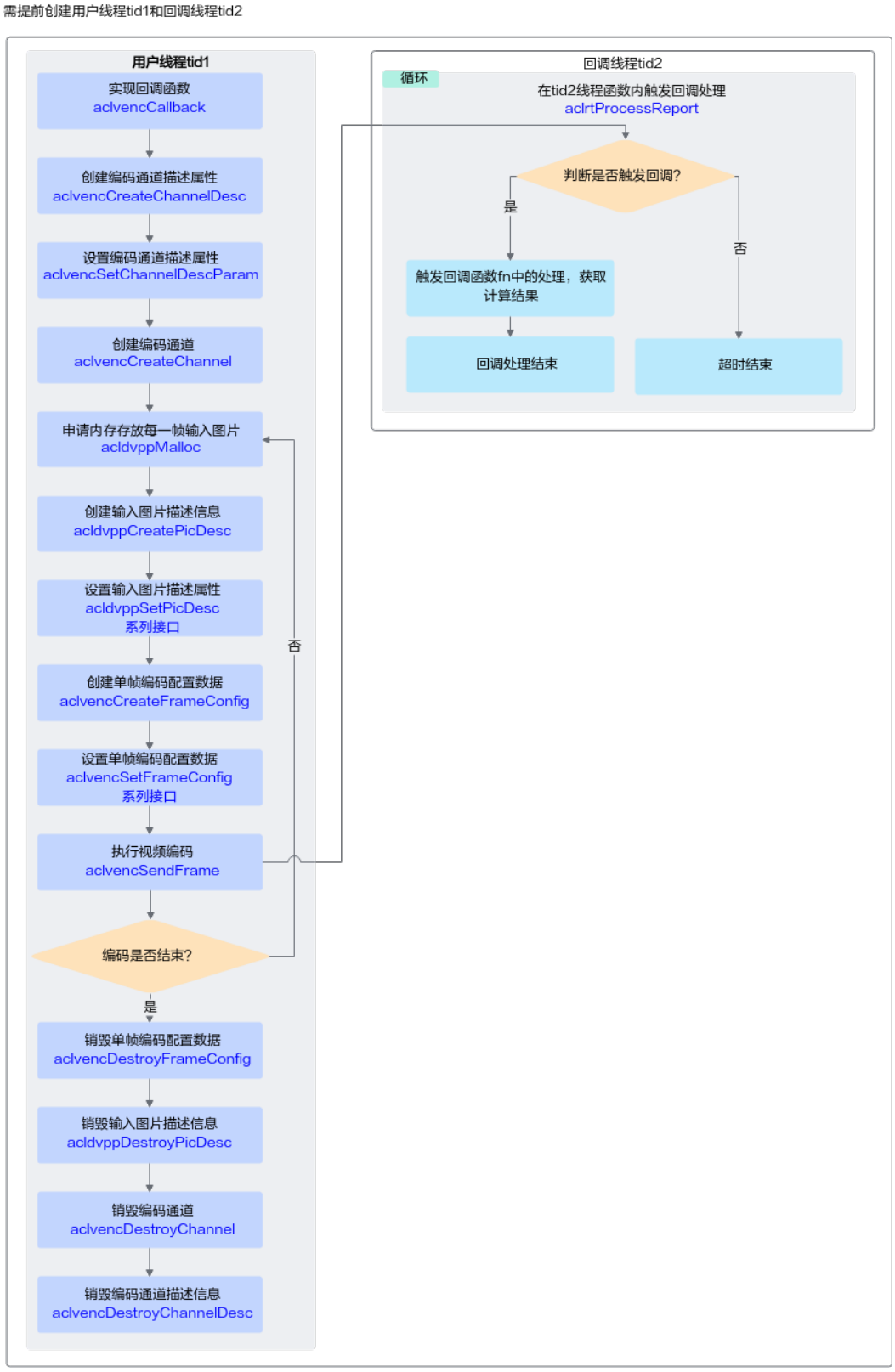
本节介绍VENC视频编码的接口调用流程，同时配合示例代码辅助理解该接口调用流程。

VENC（Video Encoder）将YUV420SP格式的图片编码成H264/H265格式的视频码流。关于VENC功能的详细介绍及约束请参见**11.17.10.1 功能及约束说明**。

接口调用流程

开发应用时，如果涉及视频编码，则应用程序中必须包含视频编码的代码逻辑，**关于视频编码的接口调用流程，请先参见3.2 AscendCL接口调用流程了解整体流程，再查看本节中的流程说明**。

图 5-11 视频编码流程



实现视频的编码，关键接口的说明如下：

版权所有 © 华为技术有限公司

1. **调用aclvencCreateChannel接口创建视频编码处理的通道。**
   - 创建视频编码处理通道前，需先执行以下操作：
     i. 调用aclvencCreateChannelDesc接口**创建通道描述信息**。
     ii. 调用aclvencSetChannelDescParam接口**设置通道描述信息的属性**，包括**线程、回调函数、视频编码协议、输入图片格式**等，其中：
       1) 回调函数需由用户提前创建，用于在视频编码后，获取编码数据，并及时释放相关资源，回调函数的原型请参见11.17.10.4 aclvencCallback。

       视频编码结束后，建议用户在回调函数内及时释放输入图片内存、以及相应的图片描述信息。视频编码的输出内存由系统管理，不由用户管理，因此无需用户释放。
       2) 线程需由用户提前创建，并自定义线程函数，在线程函数内调用aclrtProcessReport接口，等待指定时间后，触发1.ii.1)中的回调函数。

       说明

       推荐使用aclvencSetChannelDescParam接口设置通道描述信息的属性，通过枚举值来选择通过该接口设置某一个属性的值。

       但为兼容旧版本，也可以调用aclvencSetChannelDesc系列接口设置通道描述信息的属性，每个属性的设置对应一个set接口。
   - aclvencCreateChannel接口内部封装了如下接口，无需用户单独调用：
     i. aclrtCreateStream接口：显式创建Stream，VENC内部使用。
     ii. aclrtSubscribeReport接口：指定处理Stream上回调函数的线程，回调函数和线程是由用户调用aclvencSetChannelDescParam接口时指定的。
2. **调用aclvencSendFrame接口将YUV420SP格式的图片编码成H264/H265格式的视频码流。**
   - **视频编码前**，需先执行以下操作：
     - 调用acldvppCreatePicDesc接口创建输入图片描述信息，并调用acldvppSetPicDesc系列接口设置输入图片的内存地址、内存大小、图片格式等属性。
     - 调用aclvencCreateFrameConfig接口创建单帧编码配置数据，并调用aclvencSetFrameConfig系列接口设置是否强制重新开始I帧间隔、是否结束帧。
   - **视频编码时**，aclvencSendFrame接口内部封装了aclrtLaunchCallback接口，用于在Stream的任务队列中增加一个需要执行的回调函数。用户无需单独调用aclrtLaunchCallback接口。
   - **视频编码后**，视频编码的结果数据通过回调函数获取。
3. **调用aclvencDestroyChannel接口销毁视频处理的通道。**
   - 系统会等待已发送帧编码完成且用户的回调函数处理完成后再销毁通道。
   - aclvencDestroyChannel接口内部封装了如下接口，无需用户单独调用：
     - aclrtUnSubscribeReport接口：取消线程注册（Stream上的回调函数不再由指定线程处理）。
     - aclrtDestroyStream接口：销毁Stream。

- 销毁通道后，需调用**aclvencDestroyChannelDesc**接口销毁通道描述信息。
- 销毁通道描述信息后，用户才可以销毁**1.ii.2)**中创建的线程。

示例代码

您可以从10.10.1 样例介绍中获取完整样例代码。

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1.AscendCL初始化
aclRet = aclInit(nullptr);

// 2.运行管理资源申请（依次申请Device、Context、Stream）
aclrtContext context_;
aclrtStream stream_;
aclrtSetDevice(0);
aclrtCreateContext(&context_, 0);
aclrtCreateStream(&stream_);

// 3.创建执行回调函数的线程及线程函数
static bool runFlag = true;
void *ThreadFunc(void *arg)
{
    // Notice: create context for this thread
    int deviceId = 0;
    aclrtContext context = nullptr;
    aclError ret = aclrtCreateContext(&context, deviceId);
    INFO_LOG("process callback thread start ");
    while (runFlag) {
        // Notice: timeout 1000ms
        (void)aclrtProcessReport(1000);
    }
    // ......
    ret = aclrtDestroyContext(context);
    return (void*)0;
}

int createThreadErr = pthread_create(&threadId_, nullptr, ThreadFunc, nullptr);

// 4.创建回调函数
void callback(acldvppPicDesc *input, acldvppStreamDesc *outputStreamDesc, void *userdata)
{
    // 获取视频编码结果数据，并写入文件
    void *outputDev = acldvppGetStreamDescData(outputStreamDesc);
    uint32_t streamDescSize = acldvppGetStreamDescSize(outputStreamDesc);
    if (!Utils::WriteToFile(g_outFileFp, outputDev, streamDescSize)) {
        ERROR_LOG("write file:%s failed.", g_outFile.c_str());
    }
    INFO_LOG("success to callback, stream size:%u", streamDescSize);
}

// 5.创建视频编码处理通道时的通道描述信息，设置通道描述信息的属性，其中线程、callback回调函数需要用户提前创建
// vencChannelDesc_是aclvdecChannelDesc类型
vencChannelDesc_ = aclvencCreateChannelDesc();
auto ret = aclvencSetChannelDescThreadId(vencChannelDesc_, threadId_);
ret = aclvencSetChannelDescCallback(vencChannelDesc_, callback);
ret = aclvencSetChannelDescEnType(vencChannelDesc_, enType_);
ret = aclvencSetChannelDescPicFormat(vencChannelDesc_, format_);
ret = aclvencSetChannelDescPicWidth(vencChannelDesc_, 128);
ret = aclvencSetChannelDescPicHeight(vencChannelDesc_, 128);
ret = aclvencSetChannelDescKeyFrameInterval(vencChannelDesc_, 16);

// 6.创建视频码流处理的通道、单帧编码配置数据
ret = aclvencCreateChannel(vencChannelDesc_);
// vencFrameConfig_是aclvencFrameConfig类型
```

```
vencFrameConfig_ = aclvencCreateFrameConfig();

// 7.申请Device内存dataDev，存放视频编码的输入数据
// 7.1 读入图片数据
const char *fileName = "../dvpp_venc_128x128_nv12.yuv";
FILE *fp = fopen(fileName, "rb+");
fseek(fp, 0, SEEK_END);
long fileLenLong = ftell(fp);
fseek(fp, 0, SEEK_SET);
auto fileLen = static_cast<uint32_t>(fileLenLong);
uint32_t dataSize = fileLen;

// 调用aclrtGetRunMode接口获取软件栈的运行模式，如果调用aclrtGetRunMode接口获取软件栈的运行模式为ACL_HOST，则需要通过aclrtMemcpy接口将输入图片数据传输到Device，数据传输完成后，需及时释放内存；否则直接申请并使用Device的内存
aclrtRunMode runMode;
ret = aclrtGetRunMode(&runMode);
if（runMode == ACL_HOST）{
    void *dataHost = malloc(fileLen);
    size_t readSize = fread(dataHost, 1, fileLen, fp)
    void *dataDev = nullptr;
    ret = acldvppMalloc(&dataDev, dataSize);
    ret = aclrtMemcpy(dataDev, dataSize, dataHost, fileLen, ACL_MEMCPY_HOST_TO_DEVICE);
    // 完成数据传输后，需及时释放内存
    free(dataHost);
}
else {
    ret = acldvppMalloc(&dataDev, dataSize);
}

// 8.执行视频编码
size_t g_vencCnt = 16;
// 8.1 创建输入图片描述信息，设置输入图片数据的内存地址和内存大小
inputPicputDesc_ = acldvppCreatePicDesc();
ret = acldvppSetPicDescData(inputPicputDesc_, dataDev);
ret = acldvppSetPicDescSize(inputPicputDesc_, dataSize);
// 8.2 设置单帧编码配置数据，不是结束帧
ret = aclvencSetFrameConfigEos(vencFrameConfig_, 0);
ret = aclvencSetFrameConfigForceIFrame(vencFrameConfig_, 0)
// 8.3 创建输出码流描述信息
acldvppStreamDesc *outputStreamDesc = nullptr;
// 8.4 执行视频编码
while (g_vencCnt > 0) {
        ret = aclvencSendFrame(vencChannelDesc_, inputPicputDesc_,
            static_cast<void *>(outputStreamDesc), vencFrameConfig_, nullptr);
        g_vencCnt--;
    }
// 8.5 设置单帧编码配置数据，是结束帧
ret = aclvencSetFrameConfigEos(vencFrameConfig_, 1);
ret = aclvencSetFrameConfigForceIFrame(vencFrameConfig_, 0)
// 8.6 执行最后一帧的视频编码
ret = aclvencSendFrame(vencChannelDesc_, nullptr, nullptr, vencFrameConfig_, nullptr);

// 9.释放资源
(void)aclvencDestroyChannel(vencChannelDesc_);
(void)aclvencDestroyChannelDesc(vencChannelDesc_);
(void)acldvppDestroyPicDesc(inputPicputDesc_);
(void)aclvencDestroyFrameConfig(vencFrameConfig_);
// 释放内存和销毁线程
(void)acldvppFree(inBufferDev_);
void *res = nullptr;
int joinThreadErr = pthread_join(threadId_, &res);

// 10. 释放运行管理资源（依次释放Stream、Context、Device）
aclrtDestroyStream(stream_);
aclrtDestroyContext(context_);
aclrtResetDevice(0);

// 11.AscendCL去初始化
```

```
aclRet = aclFinalize();
// ......
```

如果调用aclvencSetChannelDescParam接口设置通道描述信息的属性，调用aclvencGetChannelDescParam接口获取通道描述信息中的属性值，示例代码如下：

```
// 设置回调函数
void *func = (void *)callback;
aclvencSetChannelDescParam(vencChannelDesc_, ACL_VENC_CALLBACK_PTR, 8, &func);
// 获取回调函数
void *func1 = nullptr;
aclvencGetChannelDescParam(vencChannelDesc_, ACL_VENC_CALLBACK_PTR, 8, &len, &func1);

// 设置输入图片格式
acldvppPixelFormat format = PIXEL_FORMAT_YUV_SEMIPLANAR_420;
aclvencSetChannelDescParam(vencChannelDesc_, ACL_VENC_PIXEL_FORMAT_UINT32, 4, &format);
// 获取输入图片格式
acldvppPixelFormat format1 = PIXEL_FORMAT_YUV_SEMIPLANAR_420;
aclvencGetChannelDescParam(vencChannelDesc_, ACL_VENC_PIXEL_FORMAT_UINT32, 4, &len, &format1);

// 设置图片宽度
uint32_t width = 128;
aclvencSetChannelDescParam(vencChannelDesc_, ACL_VENC_PIC_WIDTH_UINT32, 4, &width);
// 获取图片高度
uint32_t width1 = 0;
aclvencGetChannelDescParam(vencChannelDesc_, ACL_VENC_PIC_WIDTH_UINT32, 4, &len, &width1);

// 设置图片高度
uint32_t height = 128;
aclvencSetChannelDescParam(vencChannelDesc_, ACL_VENC_PIC_HEIGHT_UINT32, 4, &height);
// 获取图片高度
uint32_t height1 = 0;
aclvencGetChannelDescParam(vencChannelDesc_, ACL_VENC_PIC_HEIGHT_UINT32, 4, &len, &height1);

// 设置编码输出缓存地址
ret = acldvppMalloc(&buf, bufSize);
aclvencSetChannelDescParam(vencChannelDesc_, ACL_VENC_BUF_ADDR_PTR, 8, &buf);
// 获取编码输出缓存地址
void *buf1 = nullptr;
aclvencGetChannelDescParam(vencChannelDesc_, ACL_VENC_BUF_ADDR_PTR, 8, &len, &buf1);
```

5.5 DVPP 图像/视频处理（媒体数据处理 V2）

5.5.1 VPC 图片处理典型功能

本节以抠图、缩放为例说明VPC图像处理时的接口调用流程，同时配合以下典型功能的示例代码辅助理解该接口调用流程。

VPC（Vision Preprocessing Core）负责图像处理功能，支持对图片做抠图、缩放、格式转换等操作。关于VPC功能的详细介绍请参见11.18.16.1 功能说明，关于VPC功能对输入、输出的约束要求，请参见11.18.16.2 约束说明。

- 图片缩放示例代码
- 抠图示例代码

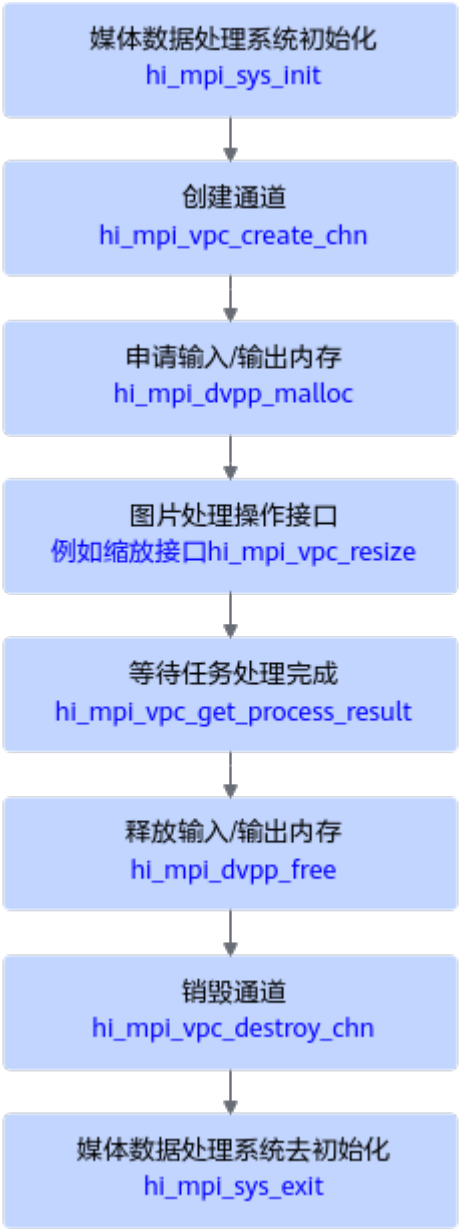
须知

Atlas 200/300/500 推理产品上，当前版本不支持该功能。

典型功能接口调用流程（以缩放为例）

开发应用时，如果涉及抠图、缩放等图片处理，则应用程序中必须包含图片处理的代码逻辑，**关于图片处理的接口调用流程，请先参见3.2 AscendCL接口调用流程了解整体流程，再查看本节中的流程说明。**

图 5-12 主要接口调用流程



当前系统支持对输入图片做抠图、缩放等处理，关键接口的说明如下：

1. 调用**hi_mpi_sys_init**接口进行**媒体数据处理系统初始化**。
2. 调用**hi_mpi_vpc_create_chn**接口**创建通道**。
3. 调用**hi_mpi_dvpp_malloc**接口**申请Device上的内存**，存放输入或输出数据。
4. **执行抠图、缩放**等，此步骤以缩放为例说明，其它功能（例如格式转换、金字塔等）请参见**11.17.5 VPC功能**下的接口说明。

调用**hi_mpi_vpc_resize**接口，对图片进行缩放。**hi_mpi_vpc_resize**接口是异步接口，调用该接口成功仅表示任务下发成功，还需要调用**hi_mpi_vpc_get_process_result**接口等待任务完成。

- 可以跟**hi_mpi_vpc_resize**接口在同一个线程中调用**hi_mpi_vpc_get_process_result**接口，也可以新起一个线程调用**hi_mpi_vpc_get_process_result**接口，后者多线程并行，提高效率，但用户需自行实现线程间同步。
- 在实现抠图、缩放等功能时，通过将输入图片和输出图片的格式设置成不同的，达到转换图片格式的目的。
- 调用**aclrtGetRunMode**接口获取软件栈的运行模式，如果运行模式为ACL_HOST，且Host上需要展示VPC输出的图片数据，则需要申请Host内存，通过**aclrtMemcpy**接口将Device的输出图片数据传输到Host；如果Host上不需要展示VPC输出的图片数据，则VPC的输出图片数据可以直接作为模型推理的输入，模型推理的相关介绍请参见**4.6 模型推理**、**7.8 模型动态AIPP推理**。

5. 调用**hi_mpi_dvpp_free**接口**释放输入、输出内存**。
6. 调用**hi_mpi_vpc_destroy_chn**接口**销毁通道**。
7. 调用**hi_mpi_sys_exit**接口进行**媒体数据处理系统去初始化**。

图片缩放示例代码

您可以从10.13.1 样例介绍中获取完整样例代码。

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1.AscendCL初始化
aclRet = aclInit(nullptr);

// 2.运行管理资源申请（依次申请Device、Context）
aclrtContext g_context;
aclRet = aclrtSetDevice(0);
aclRet = aclrtCreateContext(&g_context, 0);
// 获取软件栈的运行模式，不同运行模式影响后续的接口调用流程（例如是否进行数据传输等）
aclrtRunMode runMode;
aclError aclRet = aclrtGetRunMode(&runMode);

// 3.初始化媒体数据处理系统
int32_t ret = hi_mpi_sys_init();

// 4.创建通道
hi_vpc_chn chnId;
hi_vpc_chn_attr stChnAttr;
ret = hi_mpi_vpc_sys_create_chn(&chnId, &stChnAttr);

// 5.执行缩放
// 5.1 构造存放输入图片信息的结构体
hi_vpc_pic_info inputPic;
inputPic.picture_width = 1920;
inputPic.picture_height = 1080;
inputPic.picture_format = HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420;
inputPic.picture_width_stride = 1920;
inputPic.picture_height_stride = 1080;
inputPic.picture_buffer_size = inputPic.picture_width_stride * inputPic.picture_height_stride * 3 / 2;

// 5.2 准备输入图片数据
// 申请Device内存，用于媒体数据处理
ret = hi_mpi_dvpp_malloc(0, &inputPic.picture_address, inputPic.picture_buffer_size);

// 如果运行模式为ACL_HOST，则需要申请Host内存，将输入图片数据读入Host内存，再通过aclrtMemcpy接口
```

```
将Host的图片数据传输到Device，数据传输完成后，需及时释放Host内存；否则直接将输入图片数据读入Device
内存
if (runMode == ACL_HOST) {
    void* inputAddr = nullptr;
    // 申请Host内存
    aclRet = aclrtMallocHost(&inputAddr, inputPic.picture_buffer_size);
    // 将输入图片读入内存中，该自定义函数ReadPicFile由用户实现
    ReadPicFile(picName, inputAddr, inputPic.picture_buffer_size);
    // 数据传输
    aclRet = aclrtMemcpy(inputPic.picture_address, inputPic.picture_buffer_size, inputAddr,
                         inputPic.picture_buffer_size, ACL_MEMCPY_HOST_TO_DEVICE);
    // 完成数据传输后，需及时释放内存
    aclrtFreeHost(inputAddr);
    inputAddr = nullptr;
}
else {
    // 将输入图片读入内存中，该自定义函数ReadPicFile由用户实现
    ReadPicFile(picName, inputPic.picture_address, inputPic.picture_buffer_size);
    }

// 5.3 构造存放输出图片信息的结构体
hi_vpc_pic_info outputPic;
outputPic.picture_width = 960;
outputPic.picture_height  = 540;
outputPic.picture_format = HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420;
outputPic.picture_width_stride = 960;
outputPic.picture_height_stride = 540;
outputPic.picture_buffer_size = outputPic.picture_width_stride * outputPic.picture_height_stride * 3 / 2;
ret = hi_mpi_dvpp_malloc(0, &outputPic.picture_address, outputPic.picture_buffer_size);

// 初始化内存
if (runMode == ACL_HOST) {
    aclRet = aclrtMemset(outputPic.picture_address, outputPic.picture_buffer_size, 0,
outputPic.picture_buffer_size);
} else {
     memset(outputPic.picture_address, 0, outputPic.picture_buffer_size);
}

// 5.4 调用缩放接口
uint32_t taskID = 0;
ret = hi_mpi_vpc_resize(chnId, &inputPic, &outputPic, 0, 0, 0, &taskID, -1);

// 5.5 等待任务处理结束，任务处理结束后，输出图片数据在outputPic.picture_address指向的内存中
uint32_t taskIDResult = taskID;
ret = hi_mpi_vpc_get_process_result(chnId, taskIDResult, -1);

// 5.6 如果运行模式为ACL_HOST，且Host上需要展示VPC输出的图片数据，则需要申请Host内存，通过
aclrtMemcpy接口将Device的输出图片数据传输到Host；如果Host上不需要展示VPC输出的图片数据，则VPC的
输出图片数据可以直接作为模型推理的输入
if (g_runMode == ACL_HOST) {
    hi_vpc_pic_info outputPicHost = outputPic;
    aclRet = aclrtMallocHost(&outputPicHost.picture_address, outputPic.picture_buffer_size);
    aclRet = aclrtMemcpy(outputPicHost.picture_address, outputPic.picture_buffer_size,
outputPic.picture_address,outputPic.picture_buffer_size, ACL_MEMCPY_DEVICE_TO_HOST);
    // ......
    // Host侧的数据使用完成后，释放Host内存
    aclrtFreeHost(outputPicHost.picture_address);
    outputPicHost.picture_address = nullptr;
} else {
    // 可以直接使用VPC的输出图片数据，在outputPic.picture_address指向的内存中
    // TODO：推理相关的代码逻辑
}

// 5.7 释放输入、输出内存
ret = hi_mpi_dvpp_free(inputPic.picture_address);
ret = hi_mpi_dvpp_free(outputPic.picture_address);

// 6.销毁通道
ret = hi_mpi_vpc_destroy_chn(chnId);
```

```
// 7. 媒体数据处理系统去初始化
ret = hi_mpi_sys_exit();

// 8. 释放运行管理资源（依次释放Context、Device）
aclRet = aclrtDestroyContext(g_context);
aclRet = aclrtResetDevice(0);

// 9.AscendCL去初始化
aclRet = aclFinalize();

// ....
```

抠图示例代码

您可以从10.13.1 样例介绍中获取完整样例代码。

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1.AscendCL初始化
aclRet = aclInit(nullptr);

// 2.运行管理资源申请（依次申请Device、Context）
aclrtContext g_context;
aclRet = aclrtSetDevice(0);
aclRet = aclrtCreateContext(&g_context, 0);
获取软件栈的运行模式，不同运行模式影响后续的接口调用流程（例如是否进行数据传输等）
aclrtRunMode runMode;
aclError aclRet = aclrtGetRunMode(&runMode);

// 3.初始化媒体数据处理系统
int32_t ret = hi_mpi_sys_init();

// 4.创建通道
hi_vpc_chn chnId;
hi_vpc_chn_attr stChnAttr;
ret = hi_mpi_vpc_sys_create_chn(&chnId, &stChnAttr);

// 5.执行抠图
// 5.1 构造存放输入图片信息的结构体
hi_vpc_pic_info inputPic;
inputPic.picture_width = 1920;
inputPic.picture_height = 1080;
inputPic.picture_format = HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420;
inputPic.picture_width_stride = 1920;
inputPic.picture_height_stride = 1080;
inputPic.picture_buffer_size = inputPic.picture_width_stride * inputPic.picture_height_stride * 3 / 2;

// 5.2 准备输入图片数据
// 申请Device内存，用于媒体数据处理
ret = hi_mpi_dvpp_malloc(0, &inputPic.picture_address, inputPic.picture_buffer_size);

// 如果运行模式为ACL_HOST，则需要申请Host内存，将输入图片数据读入Host内存，再通过aclrtMemcpy接口将Host的图片数据传输到Device，数据传输完成后，需及时释放Host内存；否则直接将输入图片数据读入Device内存
if (runMode == ACL_HOST) {
    void* inputAddr = nullptr;
    // 申请Host内存
    aclRet = aclrtMallocHost(&inputAddr, inputPic.picture_buffer_size);
    // 将输入图片读入内存中，该自定义函数ReadPicFile由用户实现
    ReadPicFile(picName, inputAddr, inputPic.picture_buffer_size);
    // 数据传输
    aclRet = aclrtMemcpy(inputPic.picture_address, inputPic.picture_buffer_size, inputAddr,
                         inputPic.picture_buffer_size, ACL_MEMCPY_HOST_TO_DEVICE);
    // 完成数据传输后，需及时释放内存
    aclrtFreeHost(inputAddr);
    inputAddr = nullptr;
```

```
} else {
    // 将输入图片读入内存中，该自定义函数ReadPicFile由用户实现
    ReadPicFile(picName, inputPic.picture_address, inputPic.picture_buffer_size);
}

// 5.3 构造存放输出图片信息的结构体
// 该参数表示抠图数量
uint32_t multiCount = 1;
// cropRegionInfos数组的大小与抠图数量保持一致
hi_vpc_crop_region_info cropRegionInfos[1];
hi_vpc_pic_info outputPic;
for (uint32_t i = 0; i < multiCount; i++) {
    outputPic.picture_width = 960;
    outputPic.picture_height  = 540;
    outputPic.picture_format = HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420;
    outputPic.picture_width_stride = 960;
    outputPic.picture_height_stride = 540;
    outputPic.picture_buffer_size = outputPic.picture_width_stride * outputPic.picture_height_stride * 3 / 2;
    ret = hi_mpi_dvpp_malloc(0, &outputPic.picture_address, outputPic.picture_buffer_size);

    // 初始化内存
    if (runMode == ACL_HOST) {
        aclRet = aclrtMemset(outputPic.picture_address, outputPic.picture_buffer_size, 0,
outputPic.picture_buffer_size);
    } else {
        memset(outputPic.picture_address, 0, outputPic.picture_buffer_size);
    }

    // 表示从输入图片中抠出以左上角为原点、分辨率960*540的子图
    cropRegionInfos[i].dest_pic_info = outputPic;
    cropRegionInfos[i].crop_region.left_offset = 0;
    cropRegionInfos[i].crop_region.top_offset = 0;
    cropRegionInfos[i].crop_region.crop_width = 960;
    cropRegionInfos[i].crop_region.crop_height = 540;
}

// 5.4 调用抠图接口
uint32_t taskID = 0;
ret = hi_mpi_vpc_crop(chnId, &inputPic, cropRegionInfos, 1, &taskID, -1);

// 5.5 等待任务处理结束，任务处理结束后，输出图片数据在outputPic.picture_address指向的内存中
uint32_t taskIDResult = taskID;
ret = hi_mpi_vpc_get_process_result(chnId, taskIDResult, -1);

// 5.6 如果运行模式为ACL_HOST，且Host上需要展示VPC输出的图片数据，则需要申请Host内存，通过
aclrtMemcpy接口将Device的输出图片数据传输到Host；如果Host上不需要展示VPC输出的图片数据，则VPC的
输出图片数据可以直接作为模型推理的输入
if (g_runMode == ACL_HOST) {
    hi_vpc_pic_info outputPicHost = outputPic;
    aclRet = aclrtMallocHost(&outputPicHost.picture_address, outputPic.picture_buffer_size);
    aclRet = aclrtMemcpy(outputPicHost.picture_address, outputPic.picture_buffer_size,
outputPic.picture_address,outputPic.picture_buffer_size, ACL_MEMCPY_DEVICE_TO_HOST);
    // ......
    // Host侧的数据使用完成后，释放Host内存
    aclrtFreeHost(outputPicHost.picture_address);
    outputPicHost.picture_address = nullptr;
} else {
    // 可以直接使用VPC的输出图片数据，在outputPic.picture_address指向的内存中
    // TODO：推理相关的代码逻辑

}

// 5.7 释放输入、输出内存
ret = hi_mpi_dvpp_free(inputPic.picture_address);
inputPic.picture_address = nullptr;
for (uint32_t i = 0; i < multiCount; i++) {
     hi_mpi_dvpp_free(cropRegionInfos[i].dest_pic_info.picture_address);
     cropRegionInfos[i].dest_pic_info.picture_address = nullptr;
}
```

```
// 6.销毁通道
ret = hi_mpi_vpc_destroy_chn(chnId);

// 7. 媒体数据处理系统去初始化
ret = hi_mpi_sys_exit();

// 8. 释放运行管理资源（依次释放Context、Device）
aclRet = aclrtDestroyContext(g_context);
aclRet = aclrtResetDevice(0);

// 9.AscendCL去初始化
aclRet = aclFinalize();

// ....
```

5.5.2 JPEGD 图片解码

本节介绍JPEGD图片解码的接口调用流程，同时配合示例代码辅助理解该接口调用流程。

JPEGD（JPEG Decoder）负责完成图像解码功能，将.jpg、.jpeg、.JPG、.JPEG图片解码成YUV格式图片。关于JPEGD功能的详细介绍及约束请参见**11.18.17.1 JPEGD功能及约束说明**。

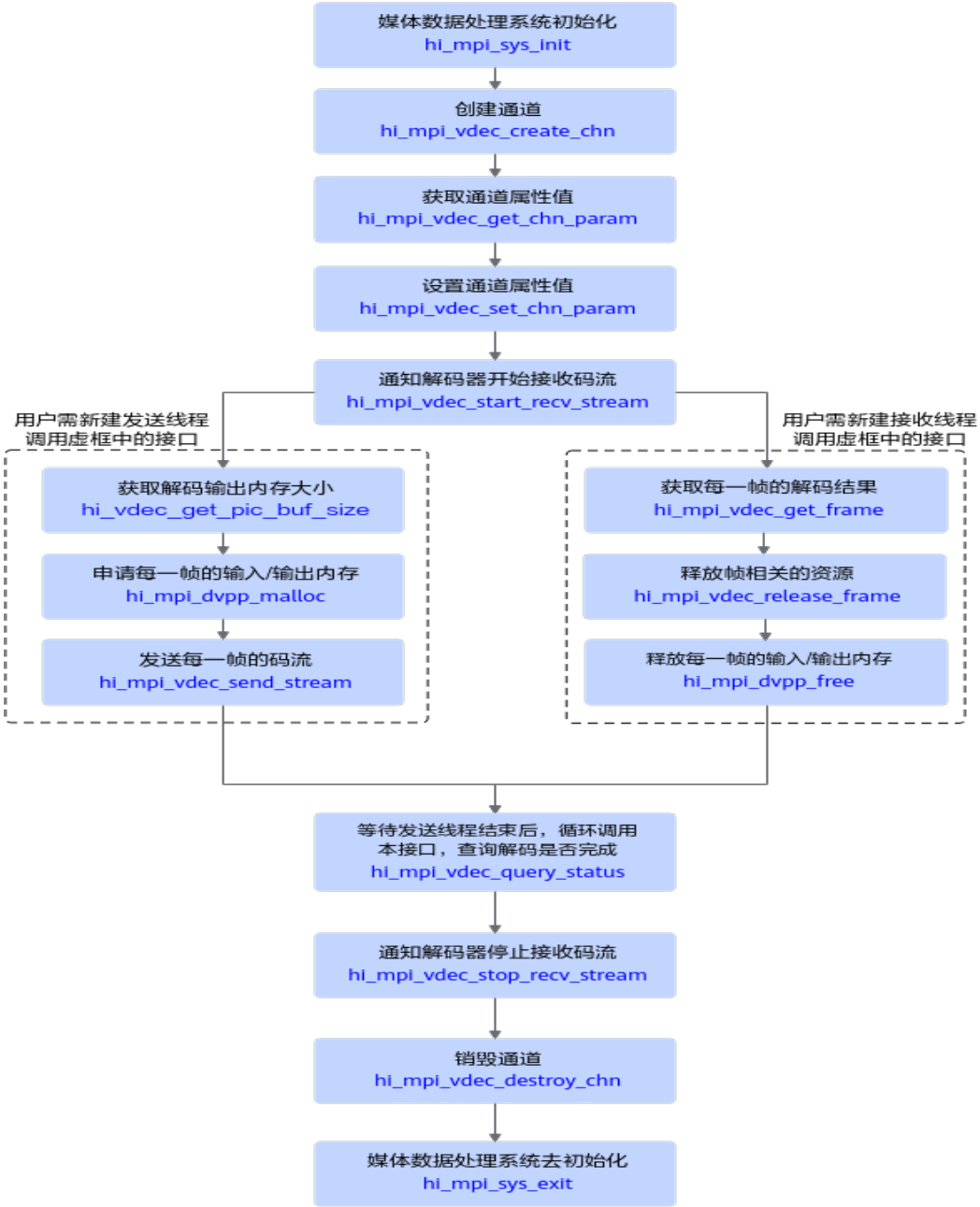
须知

Atlas 200/300/500 推理产品上，当前版本不支持该功能。

接口调用流程

开发应用时，如果涉及对JPEG图片的解码，则应用程序中必须包含解码的代码逻辑，**关于图片解码的接口调用流程，请先参见3.2 AscendCL接口调用流程了解整体流程，再查看本节中的流程说明。**

图 5-13 接口调用的流程



当前系统支持解码JPEG图片，关键接口的说明如下：

1. 调用**hi_mpi_sys_init**接口进行媒体数据处理系统初始化。
2. 调用**hi_mpi_vdec_create_chn**接口创建通道。
3. 调用**hi_mpi_dvpp_malloc**接口申请Device上的内存，存放输入或输出数据。
4. 解码前，需调用**hi_mpi_vdec_start_recv_stream**接口通知解码器启动接收码流，再调用**hi_mpi_vdec_send_stream**接口发送解码码流，**hi_mpi_vdec_send_stream**接口是异步接口，调用该接口仅表示任务下发成功，还需要调**hi_mpi_vdec_get_frame**接口获取解码结果数据，成功获取解码数据后，可以调用**hi_mpi_vdec_release_frame**接口释放帧相关的资源。解码结束后，需调用**hi_mpi_vdec_stop_recv_stream**接口通知解码器停止接收码流。

5. 调用**hi_mpi_dvpp_free**接口释放输入、输出内存。
6. 调用**hi_mpi_vdec_destroy_chn**接口销毁通道。
7. 调用**hi_mpi_sys_exit**接口进行媒体数据处理系统去初始化。

示例代码

您可以从10.14.1 样例介绍中获取完整样例代码。

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1.AscendCL初始化
aclRet = aclInit(nullptr);

// 2.运行管理资源申请（依次申请Device、Context）
aclrtContext g_context;
aclRet = aclrtSetDevice(0);
aclRet = aclrtCreateContext(&g_context, 0);
// 获取软件栈的运行模式，不同运行模式影响后续的接口调用流程（例如是否进行数据传输等）
aclrtRunMode runMode;
aclError aclRet = aclrtGetRunMode(&runMode);

// 3.初始化媒体数据处理系统
int32_t ret = hi_mpi_sys_init();

// 4.创建通道
hi_vdec_chn chnId;
hi_vdec_chn_attr chnAttr;

chnAttr.type = HI_PT_JPEG;
chnAttr.mode = HI_VDEC_SEND_MODE_FRAME;
chnAttr.pic_width = 1920;
chnAttr.pic_height = 1080;
chnAttr.stream_buf_size = 1920 * 1080;

ret = hi_mpi_vdec_create_chn(chnId, &chnAttr);

// 5.设置通道属性
hi_vdec_chn_param chnParam;
ret = hi_mpi_vdec_get_chn_param(chnId, &chnParam);

chnParam.pic_param.pixel_format = HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420;
chnParam.pic_param.alpha = 255;
chnParam.display_frame_num = 0;
ret = hi_mpi_vdec_set_chn_param(chnId, &chnParam);

// 6.解码器启动接收码流
ret = hi_mpi_vdec_start_recv_stream(chnId);

// 7.发送码流
// 7.1 申请输入内存
uint8_t* inputAddr = nullptr;
// inputsize表示输入图片占用的内存大小，此处以1024 Byte为例，用户需根据实际情况计算内存大小
int32_t inputSize = 1024;
ret = hi_mpi_dvpp_malloc(0, &inputAddr, inputSize);

// 如果运行模式为ACL_HOST，则需要申请Host内存，将输入图片数据读入Host内存，再通过aclrtMemcpy接口将Host的图片数据传输到Device，数据传输完成后，需及时释放Host内存；否则直接将输入图片数据读入Device内存
if (runMode == ACL_HOST) {
    void* hostInputAddr = nullptr;
    // 申请Host内存
    aclRet = aclrtMallocHost(&hostInputAddr, inputSize);
    // 将输入图片读入内存中，该自定义函数ReadStreamFile由用户实现
    ReadStreamFile(fileName, hostInputAddr, inputSize);
    // 数据传输
    aclRet = aclrtMemcpy(inputAddr, inputSize, hostInputAddr, inputSize,
```

```
ACL_MEMCPY_HOST_TO_DEVICE);
} else {
    // 将输入图片读入内存中，该自定义函数ReadStreamFile由用户实现
    ReadStreamFile(fileName, inputAddr, inputSize);
}

// 7.2 构造存放输入图片信息的结构体
hi_vdec_stream stStream{};
hi_img_info stImgInfo{};
stStream.pts = 0;
if (g_runMode == ACL_HOST) {
    stStream.addr  = (uint8_t *)hostInputAddr;
} else {
    stStream.addr  = (uint8_t *)inputAddr;
}
stStream.len = inputSize;
stStream.end_of_frame = HI_TRUE;
stStream.end_of_stream = HI_FALSE;
stStream.need_display  = HI_TRUE;

ret = hi_mpi_dvpp_get_image_info(HI_PT_JPEG, &stStream, &stImgInfo);
if (g_runMode == ACL_HOST) {
    // 如果不使用Host上的数据，需及时释放
   aclrtFreeHost(hostInputAddr);
    hostInputAddr = nullptr;
}
stStream.addr = (uint8_t *)inputAddr;

// 7.3 构造存放输出图片信息的结构体，并申请输出内存
hi_vdec_pic_info outPicInfo{};
void *outBuffer = nullptr;
outPicInfo.width  = stImgInfo.width;
outPicInfo.height = stImgInfo.height;
outPicInfo.width_stride  = stImgInfo.width_stride;
outPicInfo.height_stride = stImgInfo.height_stride;
outPicInfo.buffer_size   = stImgInfo.img_buf_size;
outPicInfo.pixel_format  = HI_PIXEL_FORMAT_UNKNOWN;

ret = hi_mpi_dvpp_malloc(0, &outBuffer, outPicInfo.buffer_size);

outPicInfo.vir_addr = (uint64_t)outBuffer;

// 7.4 发送需解码的输入图片
ret = hi_mpi_vdec_send_stream(chnId, &stStream, &outPicInfo, 0);

// 8.接收解码结果
// 8.1 获取解码结果
hi_video_frame_info frame;
hi_vdec_stream stream;
hi_vdec_supplement_info stSupplement;
ret = hi_mpi_vdec_get_frame(chnId, &frame, &stSupplement, &stream, 0);
if (ret == HI_SUCCESS) {
   decResult = frame.v_frame.frame_flag;
   if (decResult == 0) { // 0: Decode success
       printf("[%s][%d] Chn %u GetFrame Success, Decode Success \n",__FUNCTION__, __LINE__, chnId);
   } else { // Decode fail
       printf("[%s][%d] Chn %u GetFrame Success, Decode Fail \n",__FUNCTION__, __LINE__, chnId);
   }
}

// 8.2 如果运行模式为ACL_HOST，且Host上需要展示JPEGD输出的图片数据，则需要申请Host内存，通过
aclrtMemcpy接口将Device的输出图片数据传输到Host
if (g_runMode == ACL_HOST) {
    void* hostOutputAddr = nullptr;
    aclRet = aclrtMallocHost(&hostOutputAddr, outputSize);
    aclRet = aclrtMemcpy(hostOutputAddr, outputSize, frame.v_frame.virt_addr[0], outputSize,
ACL_MEMCPY_DEVICE_TO_HOST);
    // ......
    // 数据使用完成后，及时释放不使用的内存
```

```
    aclrtFreeHost(hostOutputAddr);
    hostOutputAddr = nullptr;
} else {
    // 可以直接使用JPEGD的输出图片数据，在frame.v_frame.virt_addr[0]指向的内存中
    // ......
}


// 8.3 释放输入、输出内存
ret = hi_mpi_dvpp_free(frame.v_frame.virt_addr[0]);
ret = hi_mpi_dvpp_free(stream.addr);

// 8.4 释放资源
ret = hi_mpi_vdec_release_frame(chnId, &frame);

// 9.解码器停止接收码流
ret = hi_mpi_vdec_stop_recv_stream(chnId);

// 10.销毁通道
ret = hi_mpi_vdec_destroy_chn(chnId);

// 11. 媒体数据处理系统去初始化
ret = hi_mpi_sys_exit();

// 12. 释放运行管理资源（依次释放Context、Device）
aclRet = aclrtDestroyContext(g_context);
aclRet = aclrtResetDevice(0);

// 13.AscendCL去初始化
aclRet = aclFinalize();

// ....
```

5.5.3 JPEGE 图片编码

本节介绍JPEGE图片编码的接口调用流程，同时配合示例代码辅助理解该接口调用流程。

JPEGE（JPEG Encoder）负责完成图像编码功能，将YUV格式图片编码成.jpg图片。关于JPEGE功能的详细介绍及约束请参见**11.18.18.1 JPEGE功能及约束说明**。

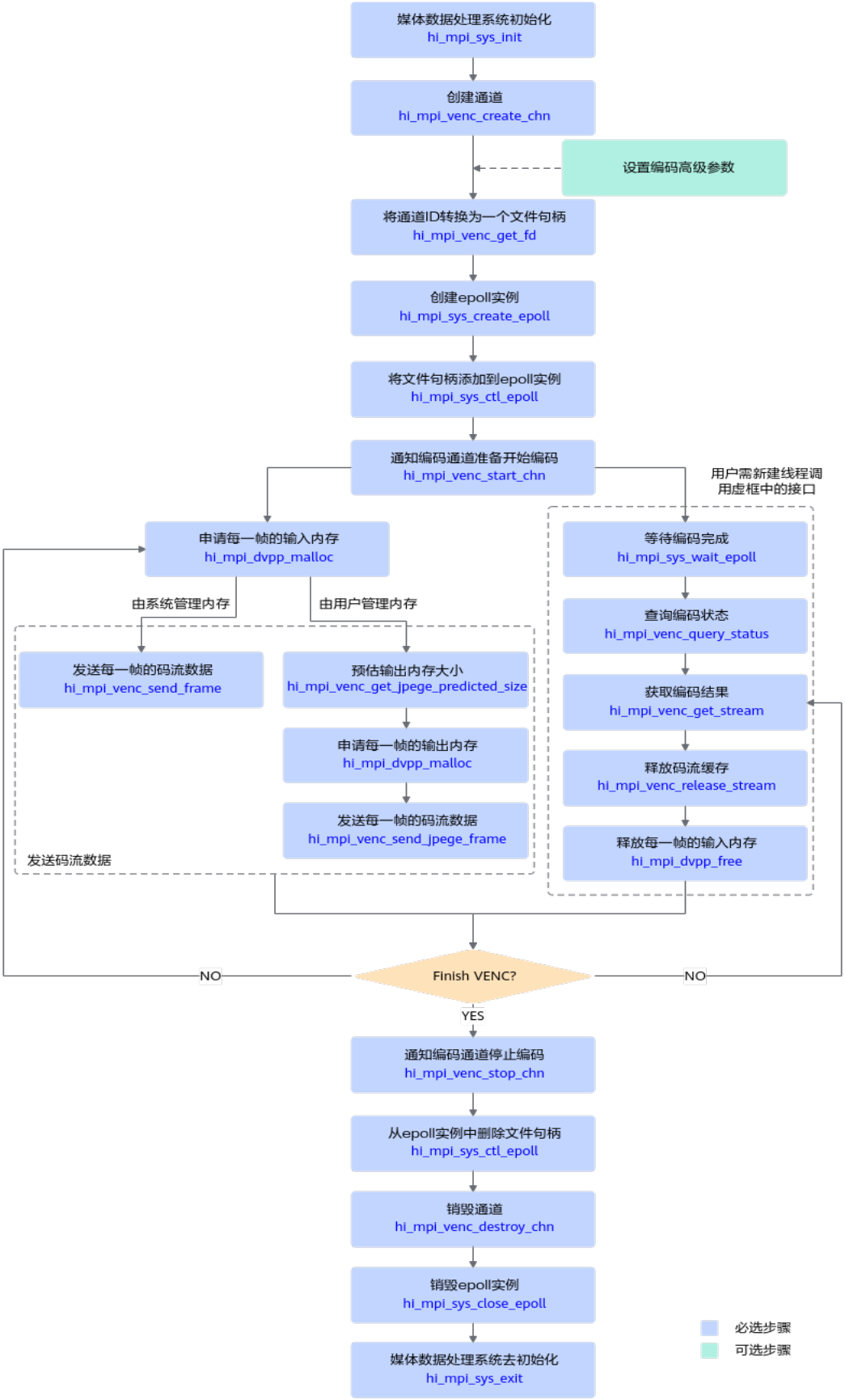
须知

Atlas 200/300/500 推理产品上，当前版本不支持该功能。

接口调用流程

开发应用时，如果涉及将YUV格式图片编码成JPEG压缩格式的图片文件，则应用程序中必须包含编码的代码逻辑，**关于编码的接口调用流程，请先参见3.2 AscendCL接口调用流程了解整体流程，再查看本节中的流程说明**。

图 5-14 接口调用流程


媒体数据处理系统初始化
hi_mpi_sys_init
创建通道
hi_mpi_venc_create_chn
设置编码高级参数
将通道ID转换为一个文件句柄
hi_mpi_venc_get_fd
创建epoll实例
hi_mpi_sys_create_epoll
将文件句柄添加到epoll实例
hi_mpi_sys_ctl_epoll
通知编码通道准备开始编码
hi_mpi_venc_start_chn
用户需新建线程调用虚框中的接口
申请每一帧的输入内存
hi_mpi_dvpp_malloc
由系统管理内存
由用户管理内存
发送每一帧的码流数据
hi_mpi_venc_send_frame
预估输出内存大小
hi_mpi_venc_get_jpege_predicted_size
申请每一帧的输出内存
hi_mpi_dvpp_malloc
发送每一帧的码流数据
hi_mpi_venc_send_jpege_frame
发送码流数据
等待编码完成
hi_mpi_sys_wait_epoll
查询编码状态
hi_mpi_venc_query_status
获取编码结果
hi_mpi_venc_get_stream
释放码流缓存
hi_mpi_venc_release_stream
释放每一帧的输入内存
hi_mpi_dvpp_free
NO
Finish VENC?
NO
YES
通知编码通道停止编码
hi_mpi_venc_stop_chn
从epoll实例中删除文件句柄
hi_mpi_sys_ctl_epoll
销毁通道
hi_mpi_venc_destroy_chn
销毁epoll实例
hi_mpi_sys_close_epoll
媒体数据处理系统去初始化
hi_mpi_sys_exit
必选步骤
可选步骤

当前系统支持将YUV格式图片编码成JPEG压缩格式的图片文件，关键接口的说明如下：

1. **资源初始化**：
   a. 调用**hi_mpi_sys_init**接口进行媒体数据处理系统初始化；
   b. 调用**hi_mpi_venc_create_chn**函数创建完通道;

      成功创建通道之后，您可以根据实际需求设置编码的高级参数，例如场景模式、码流控制器的高级参数等，请参见**11.18.18.18 hi_mpi_venc_set_jpeg_param**~**11.18.18.28 hi_mpi_venc_compact_jpeg_tables**章节中的接口说明。
   c. 调用**hi_mpi_venc_get_fd**将通道ID转换为一个文件句柄;
   d. 调用**hi_mpi_sys_create_epoll**函数创建DVPP epoll实例；
   e. 调用**hi_mpi_sys_ctl_epoll**函数将编码通道的文件句柄添加到epoll实例中，由epoll实例处理；

      select或者poll方式，不需要执行该步骤。
2. **图片编码**：
   a. 调用**hi_mpi_venc_start_chn**函数通知通道准备开始编码；
   b. 调用**hi_mpi_dvpp_malloc**接口申请存放Device上输入数据的内存。
   c. 启动一个用户态线程，调用**hi_mpi_sys_wait_epoll**函数等待编码完成；
   d. 之后用户就可以调用**hi_mpi_venc_send_frame**函数发送待编码的码流；
   e. 一旦编码完成，**hi_mpi_sys_wait_epoll**函数或select函数或poll函数就会返回，用户就可以调用**hi_mpi_venc_query_status**接口查询编码状态，再调用**hi_mpi_venc_get_stream**函数获取编码结果;
   f. 用户需要注意的是，编码结果数据使用完成之后，需要及时调用**hi_mpi_venc_release_stream**函数释放buffer。否则会因编码buffer用完导致后续编码无法进行；
   g. 调用**hi_mpi_dvpp_free**接口释放输入内存；
   h. 当用户不需发送图像到目的通道继续编码时，需要调用**hi_mpi_venc_stop_chn**函数通知该通道不再接收新的输入图片；
3. **资源释放**：
   a. 调用**hi_mpi_sys_ctl_epoll**函数从epoll实例中删除编码通道的文件句柄；
   b. 当用户完成所有编码之后，需要调用**hi_mpi_venc_destroy_chn**释放编码通道以及内部内存资源；
   c. 调用**hi_mpi_sys_close_epoll**函数销毁DVPP epoll实例；
   d. 调用**hi_mpi_sys_exit**接口进行媒体数据处理系统去初始化。

说明

Control CPU开放形态下，为了兼容旧版本，用户在等待编码完成时，旧版本的应用程序中调用Linux操作系统的select或者poll函数的方式仍然可用。建议使用上图中的接口调用流程，保证后续版本的演进。

支持DVPP内部管理输出内存，或用户自行管理输出内存两种方式：

- 不需要用户管理、由DVPP内部管理输出内存时，调用hi_mpi_venc_send_frame接口发送原始图像进行图像编码。

  在调用hi_mpi_venc_create_chn接口创建通道时，必须正确设置hi_venc_chn_attr.venc_attr.buf_size参数值（参数描述请参见11.18.20.14.5 hi_venc_attr）。

  该方式下，相比由用户管理内存，输出结果数据的JPEG头中不存在COM注释字段，数据长度会短一点，但需要用户从DVPP返回的内存中拷贝输出结果数据到指定内存。

- 由用户自行管理输出内存、管理内存的生命周期，调用hi_mpi_venc_send_jpege_frame接口发送原始图像进行图像编码。

  在调用hi_mpi_venc_create_chn接口创建通道时，需将hi_venc_chn_attr.venc_attr.buf_size参数值设置为0（参数描述请参见11.18.20.14.5 hi_venc_attr），然后调用hi_mpi_venc_get_jpege_predicted_size接口预估输出内存大小，调用hi_mpi_dvpp_malloc/hi_mpi_dvpp_free接口申请/释放输出内存。

  该方式下，直接在调用hi_mpi_venc_send_jpege_frame接口时，设置输出内存地址，输出结果数据直接存放到用户设置的内存中，**相比由系统管理内存的方式，用户可减少一次“从DVPP返回的内存中拷贝输出结果数据到指定内存”的操作，但输出结果数据的JPEG头中可能会存在COM注释字段（字段长度范围4~19Byte），数据长度会长一点。**

示例代码

您可以从10.15.1 样例介绍中获取完整样例代码。

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1.AscendCL初始化
aclRet = aclInit(nullptr);

// 2.运行管理资源申请（依次申请Device、Context）
aclrtContext g_context;
aclRet = aclrtSetDevice(0);
aclRet = aclrtCreateContext(&g_context, 0);
// 获取软件栈的运行模式，不同运行模式影响后续的接口调用流程（例如是否进行数据传输等）
aclrtRunMode runMode;
aclError aclRet = aclrtGetRunMode(&runMode);

// 3.初始化媒体数据处理系统
int32_t ret = hi_mpi_sys_init();

// 4.创建通道
hi_venc_chn chn = 0;
hi_venc_chn_attr attr{};
attr.venc_attr.type = HI_PT_JPEG;
attr.venc_attr.profile = 0;
attr.venc_attr.max_pic_width = 128;
attr.venc_attr.max_pic_height = 128;
attr.venc_attr.pic_width = 128;
attr.venc_attr.pic_height = 128;
attr.venc_attr.buf_size = 2 * 1024 * 1024;
attr.venc_attr.is_by_frame = HI_TRUE;
attr.venc_attr.jpeg_attr.dcf_en = HI_FALSE;
attr.venc_attr.jpeg_attr.recv_mode = HI_VENC_PIC_RECV_SINGLE;
attr.venc_attr.jpeg_attr.mpf_cfg.large_thumbnail_num = 0;
ret = hi_mpi_venc_create_chn(chn, &attr);

// 5.通知编码器开始接收输入数据
```

```
hi_venc_start_param recv_param{};
recv_param.recv_pic_num = -1;
ret = hi_mpi_venc_start_chn(chn, &recv_param);

// 6.发送输入数据
// 6.1 申请输入内存
uint8_t* inputAddr = nullptr;
int32_t inputSize = 128 * 128 * 3 / 2;
ret = hi_mpi_dvpp_malloc(0, &inputAddr, inputSize);

// 如果运行模式为ACL_HOST，则需要申请Host内存，将输入图片数据读入Host内存，再通过aclrtMemcpy接口
将Host的图片数据传输到Device，数据传输完成后，需及时释放Host内存；否则直接将输入图片数据读入Device
内存
if (runMode == ACL_HOST) {
    void* hostInputAddr = nullptr;
    // 申请Host内存
    aclRet = aclrtMallocHost(&hostInputAddr, inputSize);
    // 将输入数据读入内存中，该自定义函数JpegeReadYuvFile由用户实现
    JpegeReadYuvFile(streamName, hostInputAddr, inputSize);
    // 数据传输
    aclRet = aclrtMemcpy(inputAddr, inputSize, hostInputAddr, inputSize, ACL_MEMCPY_HOST_TO_DEVICE);
    // 完成数据传输后，需及时释放不使用的内存
    aclrtFreeHost(hostInputAddr );
    hostInputAddr = nullptr;
} else {
    // 将输入数据读入内存中，该自定义函数JpegeReadYuvFile由用户实现
    JpegeReadYuvFile(streamName, inputAddr, inputSize);
}

// 6.2 发送输入数据，开始编码
hi_video_frame_info frame{};
frame.mod_id = HI_ID_VENC;
frame.v_frame.width = 128;
frame.v_frame.height = 128;
frame.v_frame.field = HI_VIDEO_FIELD_FRAME;
frame.v_frame.pixel_format = HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420;
frame.v_frame.video_format = HI_VIDEO_FORMAT_LINEAR;
frame.v_frame.compress_mode = HI_COMPRESS_MODE_NONE;
frame.v_frame.dynamic_range = HI_DYNAMIC_RANGE_SDR8;
frame.v_frame.color_gamut = HI_COLOR_GAMUT_BT709;
frame.v_frame.width_stride[0] = 128;
frame.v_frame.width_stride[1] = 128;
frame.v_frame.width_stride[2] = 128;
frame.v_frame.virt_addr[0] = inputAddr;
frame.v_frame.virt_addr[1] = (hi_void *)((uintptr_t)frame.v_frame.virt_addr[0] + 128 * 128);
frame.v_frame.frame_flag = 0;
frame.v_frame.time_ref = 0;
frame.v_frame.pts = 0;
ret = hi_mpi_venc_send_frame(chn, &frame, 0);

// 7.获取编码结果
// 7.1 创建EPOLL实例
int32_t epollFd = 0;
int32_t fd = hi_mpi_venc_get_fd(chn);
ret = hi_mpi_sys_create_epoll(10, &epollFd);

hi_dvpp_epoll_event event;
event.events = HI_DVPP_EPOLL_IN;
event.data = (void*)(unsigned long)(fd);
ret = hi_mpi_sys_ctl_epoll(epollFd, HI_DVPP_EPOLL_CTL_ADD, fd, &event);

int32_t eventCount = 0;
// 编码完成前，会超时阻塞在这里，一旦完成，才会往下执行
ret = hi_mpi_sys_wait_epoll(epollFd, event, 3, 1000, &eventCount);

// 7.2 获取编码结果
hi_venc_chn_status stat;
ret = hi_mpi_venc_query_status(chn, &stat);
hi_venc_stream stream;
```

```
stream.pack_cnt = stat.cur_packs;
stream.pack = new hi_venc_pack[stream.pack_cnt];
ret = hi_mpi_venc_get_stream(chn, &stream, 1000);

// 7.3 如果运行模式为ACL_HOST，且Host上需要使用编码输出的码流，则需要申请Host内存，通过
aclrtMemcpy接口将Device的输出码流传输到Host；否则直接使用编码输出码流数据
if (g_runMode == ACL_HOST) {
    void* hostOutputAddr = nullptr;
    aclRet = aclrtMallocHost(&hostOutputAddr, outputSize);
    aclRet = aclrtMemcpy(hostOutputAddr, outputSize, stream.pack[0].addr, outputSize,
ACL_MEMCPY_DEVICE_TO_HOST);
    // ......
    // 数据使用完成后，需及时释放不使用的内存
    aclrtFreeHost(hostOutputAddr);
    hostOutputAddr = nullptr;
} else {
    // 可以直接使用编码输出码流数据，在stream.pack[0].addr指向的内存中
    // ......
}

// 8.释放输入内存和输出码流
ret = hi_mpi_dvpp_free(inputAddr);
ret = hi_mpi_venc_release_stream(chn, &stream);
delete[] stream.pack;

// 9.通知编码器停止接收输入数据
ret = hi_mpi_venc_stop_chn(chn);
ret = hi_mpi_sys_ctl_epoll(epollFd, HI_DVPP_EPOLL_CTL_DEL, fd, NULL);
ret = hi_mpi_sys_close_epoll(epollFd);

// 10.销毁通道
ret = hi_mpi_venc_destroy_chn(chn);

// 11.媒体数据处理系统去初始化
ret = hi_mpi_sys_exit();

// 12.释放运行管理资源（依次释放Context、Device）
aclRet = aclrtDestroyContext(g_context);
aclRet = aclrtResetDevice(0);

// 13.AscendCL去初始化
aclRet = aclFinalize();

// ....
```

5.5.4 PNGD 图片解码

本节介绍PNGD图片编码的接口调用流程，同时配合示例代码辅助理解该接口调用流程。

PNGD（PNG decoder）负责PNG格式图片的解码。关于PNGD功能的详细介绍及约束请参见**11.18.19 PNGD图片解码功能**。

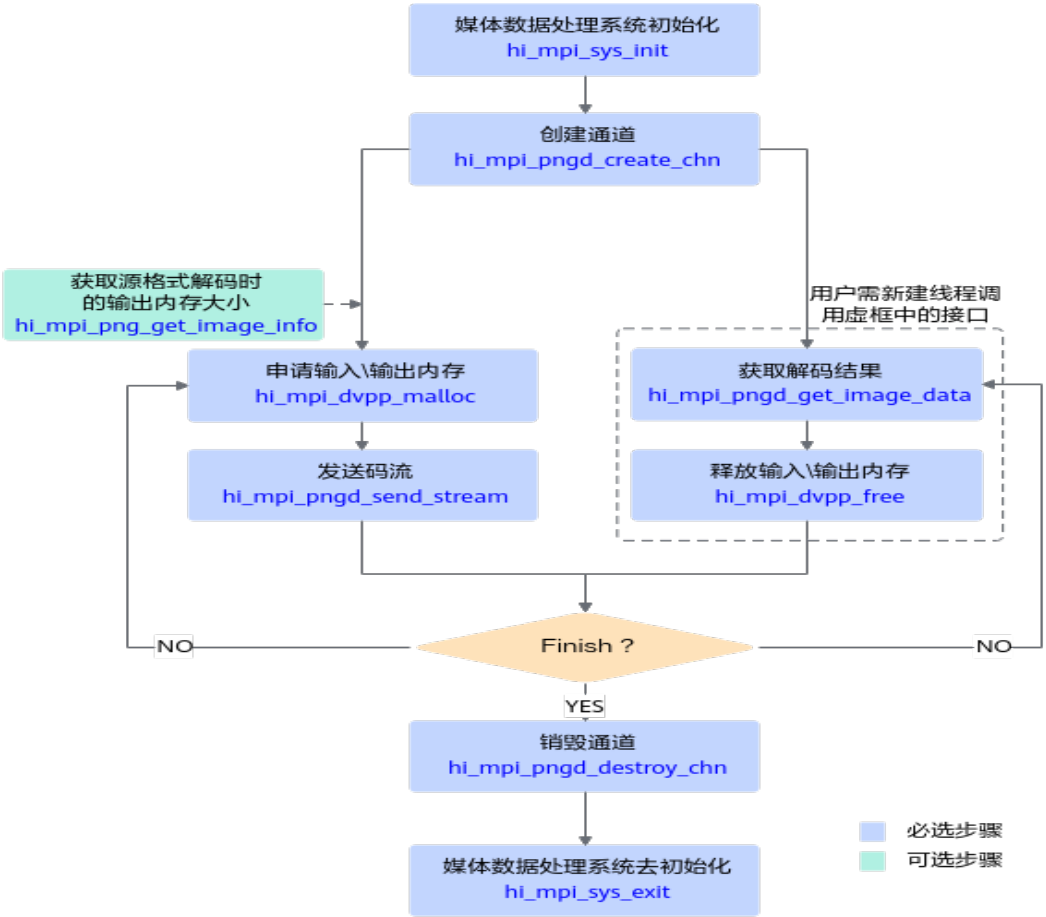
须知

Atlas 200/300/500 推理产品上，当前版本不支持该功能。

接口调用流程

开发应用时，如果涉及对PNG图片的解码，则应用程序中必须包含解码的代码逻辑，**关于图片解码的接口调用流程，请先参见3.2 AscendCL接口调用流程了解整体流程，再查看本节中的流程说明。**

图 5-15 接口调用流程



当前系统支持解码PNG图片，关键接口的说明如下：

1. 调用**hi_mpi_sys_init**接口进行媒体数据处理系统初始化。
2. 调用**hi_mpi_pngd_create_chn**接口创建通道。
3. 调用**hi_mpi_dvpp_malloc**接口申请Device上的内存，存放输入或输出数据。
4. 调用**hi_mpi_pngd_send_stream**接口发送解码码流，**hi_mpi_pngd_send_stream**接口是异步接口，调用该接口仅表示任务下发成功，还需要调**hi_mpi_pngd_get_image_data**接口获取解码结果数据。
5. 调用**hi_mpi_dvpp_free**接口释放输入、输出内存。
6. 调用**hi_mpi_pngd_destroy_chn**接口销毁通道。
7. 调用**hi_mpi_sys_exit**接口进行媒体数据处理系统去初始化。

示例代码

您可以从10.18.1 样例介绍中获取完整样例代码。

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1.获取软件栈的运行模式，不同运行模式影响后续的接口调用流程（例如是否进行数据传输等）
aclrtRunMode runMode;
aclError aclRet = aclrtGetRunMode(&runMode);
```

```
// 2.AscendCL初始化
aclRet = aclInit(nullptr);

// 3.运行管理资源申请（依次申请Device、Context）
aclrtContext g_context;
aclRet = aclrtSetDevice(0);
aclRet = aclrtCreateContext(&g_context, 0);

// 4.初始化媒体数据处理系统
int32_t ret = hi_mpi_sys_init();

// 5.创建通道
hi_pngd_chn chnId;
hi_pngd_chn_attr chnAttr; // hi_pngd_chn_attr都是保留参数,无需设置
ret = hi_mpi_pngd_create_chn(chnId, &chnAttr);

// 6.发送码流
// 6.1 申请输入内存
uint8_t* inputAddr = nullptr;
// inputsize表示输入图片占用的内存大小，此处以1024 byte为例，用户需根据实际情况计算内存大小
int32_t inputSize = 1024;
ret = hi_mpi_dvpp_malloc(0, &inputAddr, inputSize);

// 如果运行模式为ACL_HOST，则需要申请Host内存，将输入图片数据读入Host内存，再通过aclrtMemcpy接口
将Host的图片数据传输到Device，数据传输完成后，需及时释放Host内存；否则直接将输入图片数据读入Device
内存
if (runMode == ACL_HOST) {
    void* hostInputAddr = nullptr;
    // 申请Host内存
    aclRet = aclrtMallocHost(&hostInputAddr, inputSize);
    // 将输入图片读入内存中，该自定义函数ReadStreamFile由用户实现
    ReadStreamFile(fileName, hostInputAddr, inputSize);
    // 数据传输
    aclRet = aclrtMemcpy(inputAddr, inputSize, hostInputAddr, inputSize,
ACL_MEMCPY_HOST_TO_DEVICE);
} else {
    // 将输入图片读入内存中，该自定义函数ReadStreamFile由用户实现
    ReadStreamFile(fileName, inputAddr, inputSize);
}

// 6.2 构造存放输入图片信息的结构体
hi_img_stream stStream{};
hi_img_info stImgInfo{};
stStream.pts = 0;
if (g_runMode == ACL_HOST) {
    stStream.addr  = (uint8_t *)hostInputAddr;
} else {
    stStream.addr  = (uint8_t *)inputAddr;
}
stStream.len  = inputSize;
stStream.type = HI_PT_PNG;

ret = hi_mpi_png_get_image_info(&stStream, &stImgInfo);
if (g_runMode == ACL_HOST) {
    // 如果不使用Host上的数据，需及时释放
   aclrtFreeHost(hostInputAddr);
    hostInputAddr = nullptr;
}
stStream.addr = (uint8_t *)inputAddr;

// 6.3 构造存放输出图片信息的结构体，并申请输出内存
hi_pic_info outPicInfo{};
void *outBuffer = nullptr;
outPicInfo.picture_width  = stImgInfo.width;
outPicInfo.picture_height = stImgInfo.height;
outPicInfo.picture_width_stride  = stImgInfo.width_stride;
outPicInfo.picture_height_stride = stImgInfo.height_stride;
outPicInfo.picture_buffer_size   = stImgInfo.img_buf_size;
outPicInfo.picture_format = HI_PIXEL_FORMAT_UNKNOWN;
```

```
ret = hi_mpi_dvpp_malloc(0, &outBuffer, outPicInfo.buffer_size);

outPicInfo.picture_address = (uint64_t)outBuffer;

// 6.4 发送需解码的输入图片
ret = hi_mpi_pngd_send_stream(chnId, &stream, &outPicInfo, 0);

// 7.接收解码结果
// 7.1 获取解码结果
hi_pic_info picInfo;
hi_img_stream stream;
ret = hi_mpi_pngd_get_image_data(chnId, &picInfo, &stream, 0);
if (ret == HI_SUCCESS) { // Decode success
    printf("[%s][%d] Chn %u GetFrame Success, Decode Success \n",__FUNCTION__, __LINE__, chnId);
} else if (ret == HI_ERR_PNGD_BUF_EMPTY){ // Decoding
    printf("[%s][%d] Chn %u Decoding, try again \n",__FUNCTION__, __LINE__, chnId);
} else { // Decode fail
    printf("[%s][%d] Chn %u GetFrame Success, Decode Fail \n",__FUNCTION__, __LINE__, chnId);
}

// 7.2 如果运行模式为ACL_HOST，且Host上需要展示PNGD输出的图片数据，则需要申请Host内存，通过
aclrtMemcpy接口将Device的输出图片数据传输到Host
if (g_runMode == ACL_HOST) {
    void* hostOutputAddr = nullptr;
    aclRet = aclrtMallocHost(&hostOutputAddr, outputSize);
    aclRet = aclrtMemcpy(hostOutputAddr, outputSize, frame.v_frame.virt_addr[0], outputSize,
ACL_MEMCPY_DEVICE_TO_HOST);
    // ......
    // 数据使用完成后，及时释放不使用的内存
    aclrtFreeHost(hostOutputAddr);
    hostOutputAddr = nullptr;
} else {
    // 可以直接使用PNGD的输出图片数据，在outputPic.picture_address指向的内存中
    // ......
}

// 7.3 释放输入、输出内存
ret = hi_mpi_dvpp_free(frame.v_frame.virt_addr[0]);
ret = hi_mpi_dvpp_free(stream.addr);

// 8.销毁通道
ret = hi_mpi_pngd_destroy_chn(chnId);

// 9. 媒体数据处理系统去初始化
ret = hi_mpi_sys_exit();

// 10. 释放运行管理资源（依次释放Context、Device）
aclRet = aclrtDestroyContext(g_context);
aclRet = aclrtResetDevice(0);

// 11.AscendCL去初始化
aclRet = aclFinalize();

// ....
```

5.5.5 VDEC 视频解码

本节介绍VDEC视频编码的接口调用流程，同时配合示例代码辅助理解该接口调用流程。

VDEC（Video Decoder）负责将H264/H265格式的视频码流解码为YUV/RGB格式的图片。关于VDEC功能的详细介绍及约束请参见**11.18.17.3 VDEC功能及约束说明**。

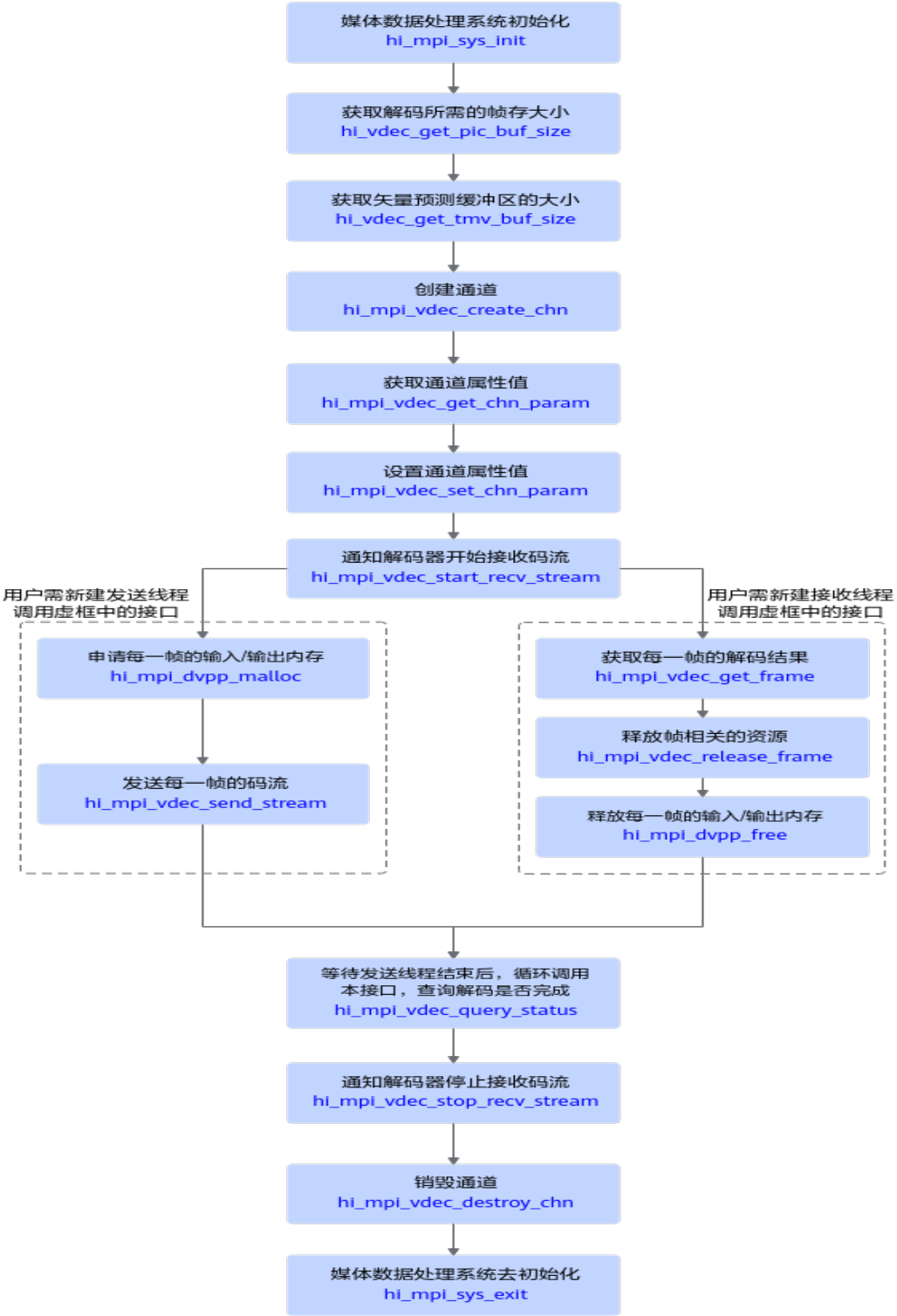
须知

Atlas 200/300/500 推理产品上，当前版本不支持该功能。

接口调用流程

开发应用时，如果涉及视频解码，则应用程序中必须包含解码的代码逻辑，**关于视频解码的接口调用流程，请先参见3.2 AscendCL接口调用流程了解整体流程，再查看本节中的流程说明。**

图 5-16 接口调用的流程



当前系统支持解码H264/H265的视频码流，关键接口的说明如下：

1. 调用**hi_mpi_sys_init**接口进行媒体数据处理系统初始化。
2. 调用**hi_vdec_get_pic_buf_size**接口获取解码所需的帧存大小、调用**hi_vdec_get_tmv_buf_size**接口获取矢量预测缓冲区的大小，在创建通道时需要使用这些数据。

3. 调用**hi_mpi_vdec_create_chn**接口创建通道。
4. 调用**hi_mpi_dvpp_malloc**接口申请Device上的内存，存放输入或输出数据。
5. 解码前，需调用**hi_mpi_vdec_start_recv_stream**接口通知解码器启动接收码流，再调用**hi_mpi_vdec_send_stream**接口发送解码码流，**hi_mpi_vdec_send_stream**接口是异步接口，调用该接口仅表示任务下发成功，还需要调**hi_mpi_vdec_get_frame**接口获取解码结果数据，成功获取解码数据后，可以调用**hi_mpi_vdec_release_frame**接口释放帧相关的资源。解码结束后，需调用**hi_mpi_vdec_stop_recv_stream**接口通知解码器停止接收码流。
6. 调用**hi_mpi_dvpp_free**接口释放输入、输出内存。
7. 调用**hi_mpi_vdec_destroy_chn**接口销毁通道。
8. 调用**hi_mpi_sys_exit**接口进行媒体数据处理系统去初始化。

示例代码

您可以从10.16.1 样例介绍中获取完整样例代码。

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1.获取软件栈的运行模式，不同运行模式影响后续的接口调用流程（例如是否进行数据传输等）
aclrtRunMode runMode;
aclError aclRet = aclrtGetRunMode(&runMode);

// 2.AscendCL初始化
aclRet = aclInit(nullptr);

// 3.运行管理资源申请（依次申请Device、Context）
aclrtContext g_context;
aclRet = aclrtSetDevice(0);
aclRet = aclrtCreateContext(&g_context, 0);

// 4.初始化媒体数据处理系统
int32_t ret = hi_mpi_sys_init();

// 5.创建通道
hi_vdec_chn chnId;
hi_vdec_chn_attr chnAttr;
hi_pic_buf_attr buf_attr{1920, 1080, 0, 8, HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420,
HI_COMPRESS_MODE_NONE};

chnAttr.type = HI_PT_H264;
chnAttr.mode = HI_VDEC_SEND_MODE_FRAME;
chnAttr.pic_width = 1920;
chnAttr.pic_height = 1080;
chnAttr.stream_buf_size = 1920 * 1080 * 3 / 2;
chnAttr.frame_buf_cnt = 16;
chnAttr.frame_buf_size = hi_vdec_get_pic_buf_size(HI_PT_H264, &buf_attr);
chnAttr.video_attr.ref_frame_num = 12;
chnAttr.video_attr.temporal_mvp_en = HI_TRUE;
chnAttr.video_attr.tmv_buf_size = hi_vdec_get_tmv_buf_size(HI_PT_H264, 1920, 1080);

ret = hi_mpi_vdec_create_chn(chnId, &chnAttr);

// 6.设置通道属性
hi_vdec_chn_param chnParam;
ret = hi_mpi_vdec_get_chn_param(chnId, &chnParam);

chnParam.video_param.dec_mode = HI_VIDEO_DEC_MODE_IPB;
chnParam.video_param.compress_mode = HI_COMPRESS_MODE_HFBC;
chnParam.video_param.video_format = HI_VIDEO_FORMAT_TILE_64x16;
chnParam.display_frame_num = 3;
chnParam.video_param.out_order = HI_VIDEO_OUT_ORDER_DISPLAY;
```

```
ret = hi_mpi_vdec_set_chn_param(chnId, &chnParam);

// 7.解码器启动接收码流
ret = hi_mpi_vdec_start_recv_stream(chnId);

// 8.发送码流
// 8.1 申请输入内存
uint8_t* inputAddr = nullptr;
// inputsize表示每一帧码流占用的内存大小，此处以1024 Byte为例，用户需根据实际情况计算内存大小
int32_t inputSize = 1024;
ret = hi_mpi_dvpp_malloc(0, &inputAddr, inputSize);

// 如果运行模式为ACL_HOST，则需要申请Host内存，将输入码流数据读入Host内存，再通过aclrtMemcpy接口
将Host的码流数据传输到Device，数据传输完成后，需及时释放Host内存；否则直接将输入图片数据读入Device
内存
if (runMode == ACL_HOST) {
    void* hostInputAddr = nullptr;
    // 申请Host内存
    aclRet = aclrtMallocHost(&hostInputAddr, inputSize);
    // 将输入码流读入内存中，该自定义函数ReadStreamFile由用户实现
    ReadStreamFile(streamName, hostInputAddr, inputSize);
    // 数据传输
    aclRet = aclrtMemcpy(inputAddr, inputSize, hostInputAddr, inputSize, ACL_MEMCPY_HOST_TO_DEVICE);
    // 完成数据传输后，需及时释放不使用的内存
    aclrtFreeHost(hostInputAddr );
    hostInputAddr = nullptr;
} else {
   // 将输入码流读入内存中，该自定义函数ReadStreamFile由用户实现
    ReadStreamFile(streamName, inputAddr, inputSize);
}

// 8.2 申请输出内存
uint8_t* outputAddr = nullptr;
// 输出图片数据占用的内存大小与输出图片格式有关
int32_t outputSize = 1920 * 1080 * 3 / 2;
ret = hi_mpi_dvpp_malloc(0, &outputAddr, outputSize);

// 8.3 构造存放一帧输入码流信息的结构体
hi_vdec_stream stream;
stream.addr = inputAddr;
stream.len = inputSize;
stream.end_of_frame = HI_TRUE;
stream.end_of_stream = HI_FALSE;
stream.need_display = HI_TRUE;

// 8.4 构造存放一帧输出结果信息的结构体
hi_vdec_pic_info outPicInfo;
outPicInfo.width = 1920;
outPicInfo.height = 1080;
outPicInfo.width_stride = 1920;
outPicInfo.height_stride = 1080;
outPicInfo.pixel_format = HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420;
outPicInfo.vir_addr = outputAddr;
outPicInfo.buffer_size = outputSize;

// 8.5 发送一帧码流
ret = hi_mpi_vdec_send_stream(chnId, &stream, &outPicInfo, 0);

// 9.接收解码结果
// 9.1 获取解码结果
hi_video_frame_info frame;
hi_vdec_stream stream;
hi_vdec_supplement_info stSupplement;
ret = hi_mpi_vdec_get_frame(chnId, &frame, &stSupplement, &stream, 0);
if (ret == HI_SUCCESS) {
   decResult = frame.v_frame.frame_flag;
   if (decResult == 0) { // 0: Decode success
       printf("[%s][%d] Chn %u GetFrame Success, Decode Success \n",__FUNCTION__, __LINE__, chnId);
   } elseif (decResult == 1) { // 1: Decode fail
```

```
        printf("[%s][%d] Chn %u GetFrame Success, Decode Fail \n",__FUNCTION__, __LINE__, chnId);
    } elseif (decResult == 2) { // 2:This result is returned for the second field of
        printf("[%s][%d] Chn %u GetFrame Success, No Picture \n", __FUNCTION__, __LINE__, chnId);
    } elseif (decResult == 3) { // 3: Reference frame number set error
        printf("[%s][%d] Chn %u GetFrame Success, RefFrame Num Error \n",__FUNCTION__, __LINE__, chnId);
    } elseif (decResult == 4) { // 4: Reference frame size set error
        printf("[%s][%d] Chn %u GetFrame Success, RefFrame Size Error \n",__FUNCTION__, __LINE__, chnId);
  }
}

// 9.2 如果运行模式为ACL_HOST，且Host上需要展示VDEC输出的图片数据，则需要申请Host内存，通过
aclrtMemcpy接口将Device的输出图片数据传输到Host
if (g_runMode == ACL_HOST) {
    void* hostOutputAddr = nullptr;
    aclRet = aclrtMallocHost(&hostOutputAddr, outputSize);
    aclRet = aclrtMemcpy(hostOutputAddr, outputSize, frame.v_frame.virt_addr[0], outputSize,
ACL_MEMCPY_DEVICE_TO_HOST);
    // ......
    // 数据使用完成后，需及时释放不需要使用的内存
    aclrtFreeHost(hostOutputAddr);
    hostOutputAddr = nullptr;
} else {
    // 可以直接使用VDEC的输出图片数据，在frame.v_frame.virt_addr[0]指向的内存中
    // TODO：推理相关的代码逻辑
}

// 9.3 释放输入、输出内存
ret = hi_mpi_dvpp_free(frame.v_frame.virt_addr[0]);
ret = hi_mpi_dvpp_free(stream.addr);

// 9.4 释放资源
ret = hi_mpi_vdec_release_frame(chnId, &frame);

// 10.解码器停止接收码流
ret = hi_mpi_vdec_stop_recv_stream(chnId);

// 11.销毁通道
ret = hi_mpi_vdec_destroy_chn(chnId);

// 12. 媒体数据处理系统去初始化
ret = hi_mpi_sys_exit();

// 13. 释放运行管理资源（依次释放Context、Device）
aclRet = aclrtDestroyContext(g_context);
aclRet = aclrtResetDevice(0);

// 14.AscendCL去初始化
aclRet = aclFinalize();

// ....
```

5.5.6 VENC 视频编码

本节介绍VENC视频编码的接口调用流程，同时配合示例代码辅助理解该接口调用流程。

VENC（Video Encoder）将YUV420SP格式的图片编码成H264/H265格式的视频码流。关于VENC功能的详细介绍及约束请参见**11.18.18.3 VENC功能及约束说明**。

在实现VENC视频编码功能时，可在创建通道时设置基本参数、或调用对应的set接口设置高级参数，优化视频编码质量，请参见**优化视频编码质量**。

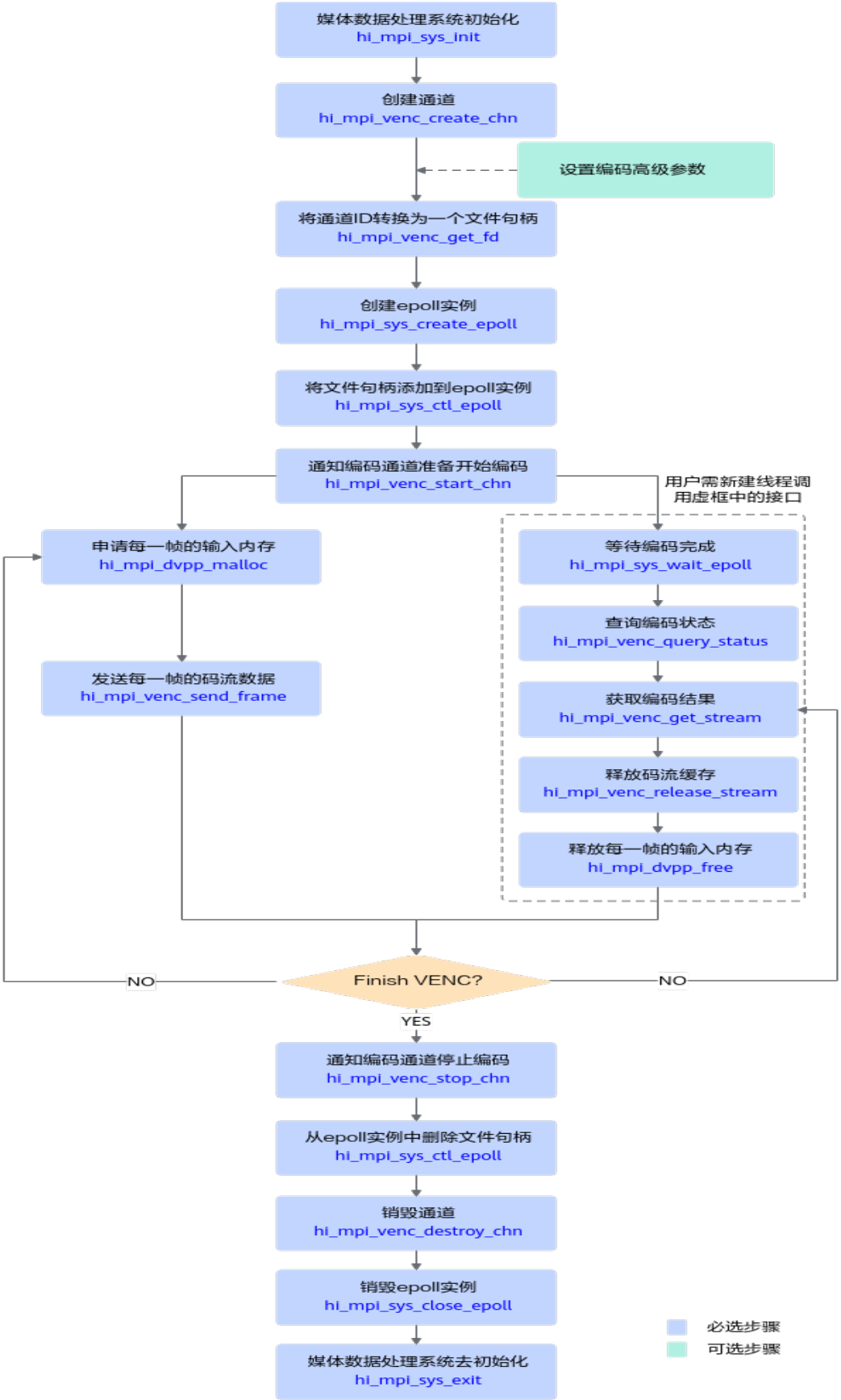
须知

Atlas 200/300/500 推理产品上，当前版本不支持该功能。

接口调用流程

开发应用时，如果涉及视频编码，则应用程序中必须包含编码的代码逻辑，**关于编码的接口调用流程，请先参见3.2 AscendCL接口调用流程了解整体流程，再查看本节中的流程说明。**

图 5-17 接口调用流程


媒体数据处理系统初始化
hi_mpi_sys_init
创建通道
hi_mpi_venc_create_chn
设置编码高级参数
将通道ID转换为一个文件句柄
hi_mpi_venc_get_fd
创建epoll实例
hi_mpi_sys_create_epoll
将文件句柄添加到epoll实例
hi_mpi_sys_ctl_epoll
通知编码通道准备开始编码
hi_mpi_venc_start_chn
用户需新建线程调用虚框中的接口
申请每一帧的输入内存
hi_mpi_dvpp_malloc
发送每一帧的码流数据
hi_mpi_venc_send_frame
等待编码完成
hi_mpi_sys_wait_epoll
查询编码状态
hi_mpi_venc_query_status
获取编码结果
hi_mpi_venc_get_stream
释放码流缓存
hi_mpi_venc_release_stream
释放每一帧的输入内存
hi_mpi_dvpp_free
Finish VENC?
NO
NO
YES
通知编码通道停止编码
hi_mpi_venc_stop_chn
从epoll实例中删除文件句柄
hi_mpi_sys_ctl_epoll
销毁通道
hi_mpi_venc_destroy_chn
销毁epoll实例
hi_mpi_sys_close_epoll
媒体数据处理系统去初始化
hi_mpi_sys_exit
必选步骤
可选步骤

当前系统支持H264/H265格式的视频码流，关键接口的说明如下：

1. **资源初始化**：
   a. 用**hi_mpi_sys_init**接口进行媒体数据处理系统初始化；
   b. 调用**hi_mpi_venc_create_chn**函数创建完通道;

      成功创建通道之后，您可以根据实际需求设置编码的高级参数，例如场景模式、码流控制器的高级参数等，请参见**11.18.18.18 hi_mpi_venc_set_jpeg_param**~**11.18.18.28 hi_mpi_venc_compact_jpeg_tables**章节中的接口说明。
   c. 调用**hi_mpi_venc_get_fd**将通道ID转换为一个文件句柄;
   d. 调用**hi_mpi_sys_create_epoll**函数创建DVPP epoll实例；
   e. 调用**hi_mpi_sys_ctl_epoll**函数将编码通道的文件句柄添加到epoll实例中，由epoll实例处理；

      select或者poll方式，不需要执行该步骤。
2. **视频编码**：
   a. 调用**hi_mpi_venc_start_chn**函数通知通道准备开始编码；
   b. 调用**hi_mpi_dvpp_malloc**接口申请存放Device上输入数据的内存。
   c. 启动一个用户态线程，调用**hi_mpi_sys_wait_epoll**函数等待编码完成；
   d. 之后用户就可以调用**hi_mpi_venc_send_frame**函数发送待编码的码流；
   e. 一旦编码完成，**hi_mpi_sys_wait_epoll**函数或select函数或poll函数就会返回，用户就可以调用**hi_mpi_venc_query_status**接口查询编码状态，再调用**hi_mpi_venc_get_stream**函数获取编码结果;
   f. 用户需要注意的是，编码结果数据使用完成之后，需要及时调用**hi_mpi_venc_release_stream**函数释放buffer。否则会因编码buffer用完导致后续编码无法进行；
   g. 调用**hi_mpi_dvpp_free**接口释放输入内存；
   h. 当用户不需发送图像到目的通道继续编码时，需要调用**hi_mpi_venc_stop_chn**函数通知该通道不再接收新的输入图片；
3. **资源释放**：
   a. 调用**hi_mpi_sys_ctl_epoll**函数从epoll实例中删除编码通道的文件句柄；
   b. 当用户完成所有编码之后，需要调用**hi_mpi_venc_destroy_chn**释放编码通道以及内部内存资源；
   c. 调用**hi_mpi_sys_close_epoll**函数销毁DVPP epoll实例；
   d. 调用**hi_mpi_sys_exit**接口进行媒体数据处理系统去初始化。

说明

Control CPU开放形态下，为了兼容旧版本，用户在等待编码完成时，旧版本的应用程序中调用Linux操作系统的select或者poll函数的方式仍然可用。建议使用上图中的接口调用流程，保证后续版本的演进。

优化视频编码质量

在实现VENC视频编码功能时，可在创建通道时设置基本参数、或调用对应的set接口设置高级参数，优化视频编码质量，以下调整手段可以叠加使用，效果是叠加的，例如：

- H264视频数据获取场景，分辨率720P，gop = 60，帧率30fps，码率1M需要提升编码质量，可以使用如下优化手段组合：CBR模式、HI_VENC_SCENE_0、stats_time等于2、profile等于2、关闭宏块级码控。
- H265电影场景，分辨率1080P，gop=30，帧率25fps，码率2M需要提升编码质量，可以使用如下优化手段组合：CBR模式、HI_VENC_SCENE_1、stats_time等于1、关闭宏块级码控。

当前支持以下方式优化视频编码质量：

- **设置基本参数，优化视频编码质量**

  不同分辨率的视频，其编码质量与视频的帧率、GOP（Group of pictures）、码率有关，在调用**hi_mpi_venc_create_chn**接口创建通道时，可设置编码的等级、设置H.264/H.265协议编码场景下CBR/VBR/AVBR/CVBR/QVBR模式的帧率、GOP、码率等参数，来调整视频编码质量：

  - 编码等级，通过hi_venc_chn_attr.venc_attr结构内的profile参数来设置；
  - 帧率，通过hi_venc_chn_attr.rc_attr结构体内的src_frame_rate输入帧率参数、dst_frame_rate输出帧率参数来设置；
  - GOP，通过hi_venc_chn_attr.rc_attr结构体内的gop参数来设置；
  - 码率，通过hi_venc_chn_attr.rc_attr结构体内的bit_rate或max_bit_rate或target_bit_rate参数来设置。

表 5-2 典型场景下帧率、GOP、码率的取值

画质/分辨率	帧率	GOP	码率（Mbps）
4K 3840*2160/4096*2160	25或30	建议GOP为帧率的整数倍，例如帧率为25时，GOP建议25或50。	● 视频数据获取场景 H264/H265码流，码率取值8~12。 ● 秀场/主播/短视频场景 H265码流，码率取值6~12。 H264码流，不涉及。 ● 游戏视频场景 H264/H265码流，码率取值10~16。
2K 2560*1440	25或30	建议GOP为帧率的整数倍，例如帧率为25时，GOP建议25或50。	● 视频数据获取场景 H264/H265码流，码率取值6~10。 ● 秀场/主播/短视频场景 H265码流，码率取值4.8~8。 H264码流，不涉及。 ● 游戏视频场景 H264/H265码流，码率取值6~10。

画质/分辨率	帧率	GOP	码率（Mbps）
1080P（蓝光） 1920*1080	25或30	建议GOP为帧率的整数倍，例如帧率为25时，GOP建议25或50。	● 视频数据获取场景 H265码流，码率取值1~4。 H264码流，码率取值2~6。 ● 秀场/主播/短视频场景 H265码流，码率取值1.4~3.6。 H264码流，码率取值2~4.8。 ● 游戏视频场景 H264/H265码流，码率取值3~6。
720P（高清） 1280*720	25或30	建议GOP为帧率的整数倍，例如帧率为25时，GOP建议25或50。	● 视频数据获取场景 H265码流，码率取值0.8~2。 H264码流，码率取值1~3。 ● 秀场/主播/短视频场景 H265码流，码率取值1~2。 H264码流，码率取值1~3。 ● 游戏视频场景 H264/H265码流，码率取值2~4。
480P/D1_N（标清） 854*480/720*480	25或30	建议GOP为帧率的整数倍，例如帧率为25时，GOP建议25或50。	● 视频数据获取场景 H265码流，码率取值0.3~0.7。 H264码流，码率取值0.6~1.4。 ● 秀场/主播/短视频场景 H265码流，码率取值0.25~0.6。 H264码流，码率取值0.3~0.7。 ● 游戏视频场景 不涉及。
576P/D1（标清） 720*576	25或30	建议GOP为帧率的整数倍，例如帧率为25时，GOP建议25或50。	● 视频数据获取场景 H265码流，码率取值0.3~0.7。 H264码流，码率取值0.6~1.4。 ● 秀场/主播/短视频场景 H265码流，码率取值0.25~0.6。 H264码流，码率取值0.3~0.7。 ● 游戏视频场景 不涉及。

画质/分辨率	帧率	GOP	码率（Mbps）
270P（流畅） 480*270	25或30	建议GOP为帧率的整数倍，例如帧率为25时，GOP建议25或50。	• 视频数据获取场景 不涉及。 • 秀场/主播/短视频场景 H265码流，码率取值0.2。 H264码流，码率取值0.3。 • 游戏视频场景 不涉及。
CIF P/N 352*288/320*240	25或30	建议GOP为帧率的整数倍，例如帧率为25时，GOP建议25或50。	• 视频数据获取场景 H264/H265码流，码率取值0.25。 • 秀场/主播/短视频场景 不涉及。 • 游戏视频场景 不涉及。

- **设置高级参数，调整视频编码细节**

  您可以调用接口设置码控模式、宏块级码率控制参数、编码场景模式等，来调整视频编码的细节，进一步改善编码质量。

表 5-3 高级配置项列表

配置项	接口	参数名	说明
码控模式	**hi_mpi_venc_create_chn**	hi_venc_chn_attr.rc_attr结构体内的rc_mode参数	追求码率平稳或追求PSNR大且码率符合目标值，配置为CBR； 追求节省码率，对主观编码质量有一定要求，配置为VBR； 追求节省码率，对主观编码质量有一定要求，且场景中有较多静止画面，配置为AVBR； 追求PSNR且对码率上浮没有严格要求，配置为QVBR； 追求节省码率，对主观编码质量有一定要求，且可以根据带宽、存储空间要求进行更多调整，配置为CVBR；

配置项	接口	参数名	说明
码率控制模型统计时间	hi_mpi_venc_create_chn	hi_venc_chn_attr.rc_attr内各模式属性值结构体内的stats_time参数	关注长期码率稳定，短期波动不在意的可以设置大一些，例：DVR存盘。设大可以提高重编码判决的门槛，重编码次数会减少，但是码率波动会加大。
宏块级码率控制参数	hi_mpi_venc_set_rc_param	hi_venc_rc_param结构内的threshold_i、threshold_p、threshold_b、direction、row_qp_delta参数	如果图像内容复杂、细节较多或用户关注PSNR等客观指标时，需关闭宏块级码率控制。
第一帧的起始Qp值	hi_mpi_venc_create_chn	hi_venc_rc_param结构内的first_frame_start_qp参数	典型场景下，用户配置的码率小于表5-2中给的参考值，且编码后的视频第一帧明显模糊，则建议配置first_frame_start_qp参数，参数值取[min_i_qp, max_i_qp]的中间值，例如，[min_i_qp, max_i_qp]为[30, 40]，则first_frame_start_qp参数配置为35，同时将max_reencode_times参数配置为0，会获得较好的编码质量。
编码场景模式	hi_mpi_venc_set_scene_mode	hi_venc_scene_mode	安防场景配置为HI_VENC_SCENE_0；自动驾驶、直播、游戏、动画、电影配置为HI_VENC_SCENE_1。

示例代码

您可以从10.17.1 样例介绍中获取完整样例代码。

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1.获取软件栈的运行模式，不同运行模式影响后续的接口调用流程（例如是否进行数据传输等）
aclrtRunMode runMode;
aclError aclRet = aclrtGetRunMode(&runMode);

// 2.AscendCL初始化
aclRet = aclInit(nullptr);

// 3.运行管理资源申请（依次申请Device、Context）
aclrtContext g_context;
```

```
aclRet = aclrtSetDevice(0);
aclRet = aclrtCreateContext(&g_context, 0);

// 4.初始化媒体数据处理系统
int32_t ret = hi_mpi_sys_init();

// 5.创建通道
hi_venc_chn chn = 0;
hi_venc_chn_attr attr{};
attr.venc_attr.type = HI_PT_H265;
attr.venc_attr.profile = 0;
attr.venc_attr.max_pic_width = 128;
attr.venc_attr.max_pic_height = 128;
attr.venc_attr.pic_width = 128;
attr.venc_attr.pic_height = 128;
attr.venc_attr.buf_size = 2 * 1024 * 1024;
attr.venc_attr.is_by_frame = HI_TRUE;
attr.rc_attr.rc_mode = HI_VENC_RC_MODE_H265_VBR;
attr.rc_attr.h265_vbr.gop = 30;
attr.rc_attr.h265_vbr.stats_time = 1;
attr.rc_attr.h265_vbr.src_frame_rate = 30;
attr.rc_attr.h265_vbr.dst_frame_rate = 30;
attr.rc_attr.h265_vbr.max_bit_rate = 4000;
attr.gop_attr.gop_mode = HI_VENC_GOP_MODE_NORMAL_P;
attr.gop_attr.normal_p.ip_qp_delta = 3;
ret = hi_mpi_venc_create_chn(chn, &attr);

// 6.通知编码器启动接收输入数据
hi_venc_start_param recv_param{};
recv_param.recv_pic_num = -1;
ret = hi_mpi_venc_start_chn(chn, &recv_param);

// 7.发送输入数据
// 7.1 申请输入内存
uint8_t* inputAddr = nullptr;
int32_t inputSize = 128 * 128 * 3 / 2;
ret = hi_mpi_dvpp_malloc(0, &inputAddr, inputSize);

// 如果运行模式为ACL_HOST，则需要申请Host内存，将输入数据读入Host内存，再通过aclrtMemcpy接口将
Host的数据传输到Device，数据传输完成后，需及时释放Host内存；否则直接将输入数据读入Device内存
if (runMode == ACL_HOST) {
    void* hostInputAddr = nullptr;
    // 申请Host内存
    aclRet = aclrtMallocHost(&hostInputAddr, inputSize);
    // 将输入数据读入内存中，该自定义函数VencReadYuvFile由用户实现
    VencReadYuvFile(streamName, hostInputAddr, inputSize);
    // 数据传输
    aclRet = aclrtMemcpy(inputAddr, inputSize, hostInputAddr, inputSize, ACL_MEMCPY_HOST_TO_DEVICE);
    // 完成数据传输后，需及时释放不使用的内存
    aclrtFreeHost(hostInputAddr );
    hostInputAddr = nullptr;
} else {
    // 将输入数据读入内存中，该自定义函数VencReadYuvFile由用户实现
    VencReadYuvFile(streamName, inputAddr, inputSize);
}

// 7.2 发送输入数据，开始编码
hi_video_frame_info frame{};
frame.mod_id = HI_ID_VENC;
frame.v_frame.width = 128;
frame.v_frame.height = 128;
frame.v_frame.field = HI_VIDEO_FIELD_FRAME;
frame.v_frame.pixel_format = HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420;
frame.v_frame.video_format = HI_VIDEO_FORMAT_LINEAR;
frame.v_frame.compress_mode = HI_COMPRESS_MODE_NONE;
frame.v_frame.dynamic_range = HI_DYNAMIC_RANGE_SDR8;
frame.v_frame.color_gamut = HI_COLOR_GAMUT_BT709;
frame.v_frame.width_stride[0] = 128;
frame.v_frame.width_stride[1] = 128;
```

```
frame.v_frame.width_stride[2] = 128;
frame.v_frame.virt_addr[0] = inputAddr;
frame.v_frame.virt_addr[1] = (hi_void *)((uintptr_t)frame.v_frame.virt_addr[0] + 128 * 128);
frame.v_frame.frame_flag = 0;
frame.v_frame.time_ref = 0;
frame.v_frame.pts = 0;
ret = hi_mpi_venc_send_frame(chn, &frame, 0);

// 8.获取编码结果
// 8.1 创建EPOLL实例
int32_t epollFd = 0;
int32_t fd = hi_mpi_venc_get_fd(chn);
ret = hi_mpi_sys_create_epoll(10, &epollFd);

hi_dvpp_epoll_event event;
event.events = HI_DVPP_EPOLL_IN;
event.data = (void*)(unsigned long)(fd);
ret = hi_mpi_sys_ctl_epoll(epollFd, HI_DVPP_EPOLL_CTL_ADD, fd, &event);

int32_t eventCount = 0;
// 编码完成前，会超时阻塞在这里，一旦完成，才会往下执行
ret = hi_mpi_sys_wait_epoll(epollFd, event, 3, 1000, &eventCount);

// 8.2 获取编码结果
hi_venc_chn_status stat;
ret = hi_mpi_venc_query_status(chn, &stat);
hi_venc_stream stream;
stream.pack_cnt = stat.cur_packs;
stream.pack = new hi_venc_pack[stream.pack_cnt];
ret = hi_mpi_venc_get_stream(chn, &stream, 1000);

// 8.3 如果运行模式为ACL_HOST，且Host上需要使用编码输出的码流，则需要申请Host内存，通过
aclrtMemcpy接口将Device的输出码流传输到Host
if (g_runMode == ACL_HOST) {
    void* hostOutputAddr = nullptr;
    aclRet = aclrtMallocHost(&hostOutputAddr, outputSize);
    aclRet = aclrtMemcpy(hostOutputAddr, outputSize, stream.pack[0].addr, outputSize,
ACL_MEMCPY_DEVICE_TO_HOST);
    // ......
    // 数据使用完成后，需及时释放不使用的内存
    aclrtFreeHost(hostOutputAddr);
    hostOutputAddr = nullptr;
} else {
    // 可以直接使用编码输出码流数据，在stream.pack[0].addr指向的内存中
    // TODO: 推理相关的代码逻辑
}

// 9.释放输入内存和输出码流
ret = hi_mpi_dvpp_free(inputAddr);
ret = hi_mpi_venc_release_stream(chn, &stream);
delete[] stream.pack;

// 10.通知编码器停止接收输入数据
ret = hi_mpi_venc_stop_chn(chn);
ret = hi_mpi_sys_ctl_epoll(epollFd, HI_DVPP_EPOLL_CTL_DEL, fd, NULL);
ret = hi_mpi_sys_close_epoll(epollFd);

// 11.销毁通道
ret = hi_mpi_venc_destroy_chn(chn);

// 12.媒体数据处理系统去初始化
ret = hi_mpi_sys_exit();

// 13.释放运行管理资源（依次释放Context、Device）
aclRet = aclrtDestroyContext(g_context);
aclRet = aclrtResetDevice(0);

// 14.AscendCL去初始化
aclRet = aclFinalize();
```

```
// ....
```

5.6 精度提升建议

5.6.1 JPEGD+VPC+模型推理精度提升建议（Atlas 200/300/500 推理产品）

图片解码、图片抠图/缩放以及模型推理功能串联使用时，可能由于接口调用或配置导致推理精度存在偏差，本节基于该场景给出一些建议。

问题描述

JPEGD+VPC+模型推理串联使用，由于宽/高的对齐、输出图片格式等配置问题，可能会导致JPEGD与VPC之间、或VPC与模型推理之间的衔接存在偏差，进而影响整网的推理精度。

精度提升建议

关于JPEGD+VPC+模型推理多个功能串联使用时的一些问题及精度提升建议如下：

1. **JPEGD+VPC串联使用时**：

   由于JPEGD解码后的输出图片的宽stride*高stride有128*16对齐的约束，因此解码后的输出图片的宽、高有一些补边的无效数据，在调用VPC的缩放接口acldvppVpcResizeAsync时，需先调用acldvppSetPicDescWidth和acldvppSetPicDescHeight接口正确设置输入图片的原图宽高，VPC内部会根据原图宽高自行抠图，然后再执行缩放，实现去除无效数据对图像精度的影响。

   为保证模型推理的精度，建议参见**图5-18**、**图5-19**中的正例来编写代码逻辑。

   通过**图5-18**中的反例也可以看出，jpegd解码后，直接将对齐后的宽高作为原图宽高送入VPC，导致模型推理的输入图片存在无效数据，最终可能影响精度。

   JPEGD和VPC的接口调用流程及详细介绍请分别参见**JPEGD接口调用流程**、**VPC接口调用流程**。

2. **使用VPC时**：
   - VPC的抠图、缩放两个功能可以通过acldvppVpcCropAsync或acldvppVpcBatchCropAsync接口来完成，该接口的输出图片的宽stride、高stride必须满足16*2对齐，否则接口返回报错。

     AscendCL当前提供了acldvppVpcCropAsync接口进行抠图、acldvppVpcResizeAsync接口进行缩放，抠图和缩放串联使用时，可以直接用acldvppVpcCropAsync接口，性能更优。
   - VPC的抠图、缩放、贴图三个功能可以通过acldvppVpcCropAndPasteAsync或acldvppVpcBatchCropAndPasteAsync接口来完成，该接口的输出图片的宽stride、高stride必须满足16*2对齐，否则接口返回报错。
   - VPC缩放+贴图串联使用时，如果缩放后的图片宽不是16对齐，在贴图时，vpc会增加无效数据（见**图5-19**中的反例），使其16对齐，为防止无效数据对后续的推理精度有影响，此处建议用户将宽*高按照16*2对齐的要求进行缩放，见**图5-19**中的正例。vpc缩放时，如果完全按vpc等比例缩放，应该输出238*416分辨率的图片，不满足16对齐，存在无效数据，为了使无效数据不影响精度，建议将图片缩放至240*416分辨率。

– VPC贴图时，贴图与输出图片的左边界距离必须满足16对齐，见**图5-19**中的正例。检测网络中、等比例缩放场景下，用户在实际使用时，如果贴图与输出图片的左边界距离d3满足16对齐后，可能导致贴图区域不在输出图片的中心位置，这时需注意，检测框与贴图区域左边界的距离d1=检测框与输出图片左边界的距离d2-d3。

  例如，**图5-19**中，在最后的vpc贴图操作中，输出图片的分辨率为416*416，贴图区域的宽是240*416，贴图相对输出图片的左偏移=（416-240）/2=88，但88不是16对齐的，如果想继续贴图，得确保贴图相对输出图片的左偏移为16对齐，例如96。这时，计算检测框与贴图区域左边界的距离d1=d2-96，而不是d2-88。

3. **模型推理时：**

   如果需要硬件AIPP进行色域转换，AIPP色域转换配置中的源图片格式要与VPC的输出图片格式保持一致，如果不一致，例如，VPC的输出格式是yuv420sp，色域转换的配置是yvu420sp（源图片格式）-->rgb888（目标图片格式），uv分量的顺序不同也会影响最后模型推理的精度。

   模型推理的接口调用流程及详细介绍请参见**4 基础推理应用**。

   色域转换的配置请参见《**ATC工具使用指南（开放态）**》。

说明

用户训练模型后，如果在昇腾上验证精度是否达标，建议模型推理前的数据预处理建议与训练模型时的数据预处理过程保持一致。可参考**Modelzoo仓**中样例代码。

典型案例

图 5-18 典型分类网络示例

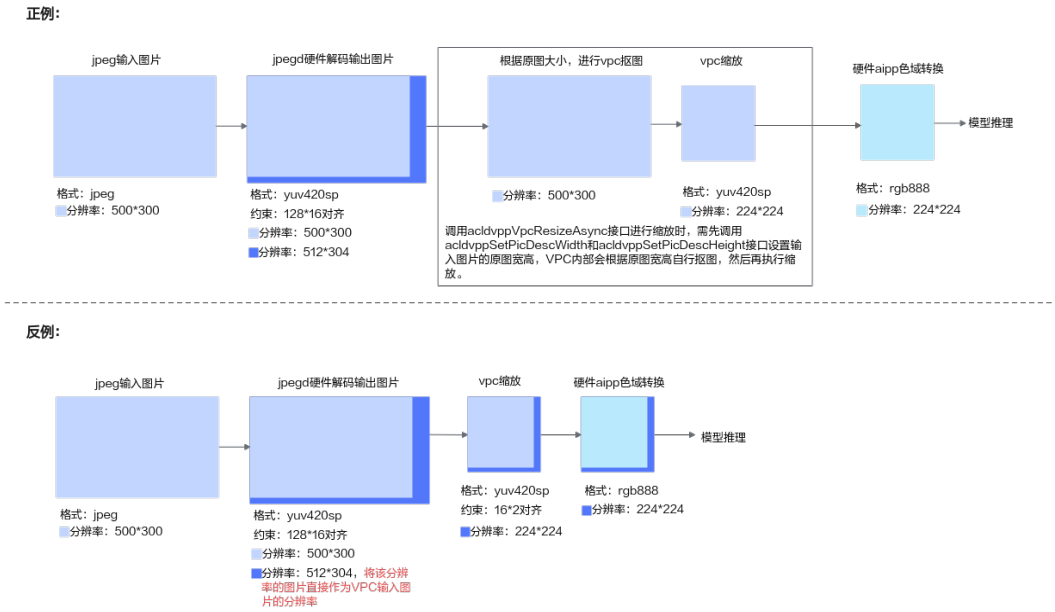
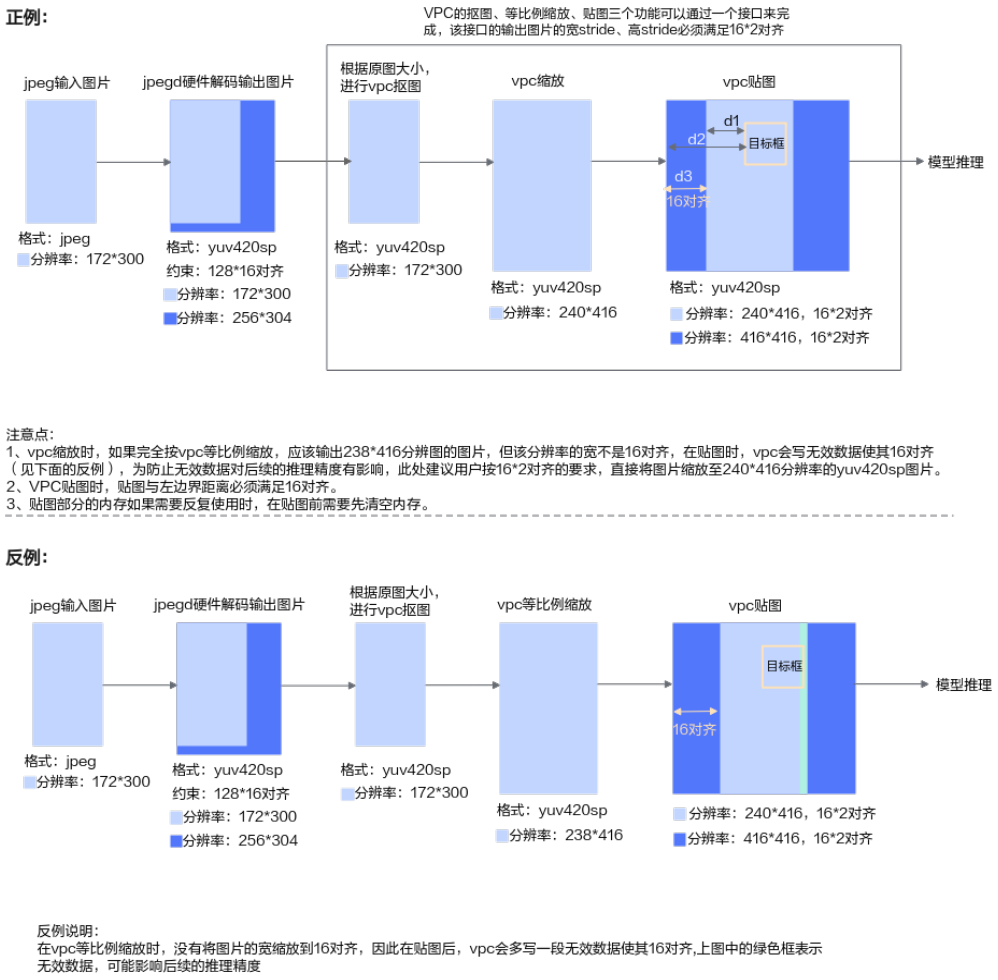


图 5-19 典型检测网络示例



示例代码

- 典型分类网络的示例代码请单击**Link**获取。
- 典型检测网络的示例代码请单击**Link**获取。

5.7 高性能编程建议

5.7.1 使用媒体数据处理 V1 版本接口

5.7.1.1 采用 VPC 多功能组合接口，减少系统调度压力，性能更优

背景说明

在对图像进行抠图、缩放、贴图、填充等处理时，AscendCL媒体数据处理部分提供了以下实现功能的接口：

- 一个接口只做一次操作（即单功能接口），例如**acldvppVpcCropAsync**、**acldvppVpcResizeAsync**、**acldvppVpcMakeBorderAsync**接口

该方式下，如果想实现多个功能，例如抠图+缩放+填充，您需要调用以上3个接口。

- 一个接口做多个操作（即多功能组合接口），例如：**acldvppVpcBatchCropResizePasteAsync**、**acldvppVpcBatchCropResizeMakeBorderAsync**接口

  该方式下，如果想实现多个功能，例如抠图+缩放+填充，您仅需要调用1个接口**acldvppVpcBatchCropResizeMakeBorderAsync**。

单功能接口与多功能组合接口的对应关系如下。

单功能接口	多功能组合接口
• **acldvppVpcCropAsync**（抠图） • **acldvppVpcResizeAsync**（缩放）	• **acldvppVpcCropResizeAsync**（抠图缩放）或**acldvppVpcBatchCropResizeAsync**（批量抠图缩放） • **acldvppVpcCropAndPasteAsync**（抠图贴图）或**acldvppVpcBatchCropAndPasteAsync**（批量抠图贴图） • **acldvppVpcCropResizePasteAsync**（抠图缩放贴图）、**acldvppVpcBatchCropResizePasteAsync**（批量抠图缩放贴图）
• **acldvppVpcCropAsync**（抠图） • **acldvppVpcResizeAsync**（缩放） • **acldvppVpcMakeBorderAsync**（填充）	**acldvppVpcBatchCropResizeMakeBorderAsync**（批量抠图缩放填充）

基本原理

一个接口内部会有多次Host和Device的任务交互，每次交互有时延，若对于抠图、缩放等多个功能，调用多次接口，Host和Device的任务交互次数就会增加，时延自然也会随之增加。

采用多个功能组合接口，调用一个接口完成多个功能，虽然是多个功能，但对于Device来说都是一次处理（一个多功能组合接口和一个单功能接口的硬件执行时间相同），相对调用多个单功能接口，能够减少Host和Device的调度次数，减少Device的处理次数，对调度和性能有较多的提升，在性能优化时可以考虑。

使用示例

此处以批量抠图、缩放为例说明如何调用多功能组合接口**acldvppVpcBatchCropResizeAsync**，完整代码请单击**Link**获取。

5.7.1.2 采用 VPC 批处理接口，降低时延，性能更优

背景说明

在对图像进行抠图、缩放等处理时，AscendCL媒体数据处理部分提供了以下两类接口：

- 一次处理一张图片，例如**acldvppVpcCropAsync**接口
  该方式下，如果存在多张输入图片，一般都采用for循环的方式，针对每张图片，都调用一次**acldvppVpcCropAsync**接口。
- 一次处理多张图片（即批处理接口），例如**acldvppVpcBatchCropAsync**接口
  该方式，如果存在多张输入图片，只需调用一次**acldvppVpcBatchCropAsync**接口。

以上两类接口的对应关系表如下。

单张图片处理接口	批量图片处理接口
acldvppVpcCropAsync（抠图）	**acldvppVpcBatchCropAsync**（批量抠图）
acldvppVpcCropResizeAsync（抠图缩放）	**acldvppVpcBatchCropResizeAsync**（批量抠图缩放）
acldvppVpcCropAndPasteAsync（抠图贴图）	**acldvppVpcBatchCropAndPasteAsync**（批量抠图贴图）
acldvppVpcCropResizePasteAsync（抠图缩放贴图）	**acldvppVpcBatchCropResizePasteAsync**（批量抠图缩放贴图）
-	**acldvppVpcBatchCropResizeMakeBorderAsync**（批量抠图缩放填充）

基本原理

昇腾AI处理器内置图像处理单元DVPP（Digital Video Pre-Processing），在DVPP中，有多个VPC（Vision Preprocessing Core）模块，处理图片的抠图、缩放、格式转换等任务。

在调用批处理接口时，批量任务会被均分到多个VPC模块、并行处理，批量接口的处理时延会降低，性能提升。

使用示例

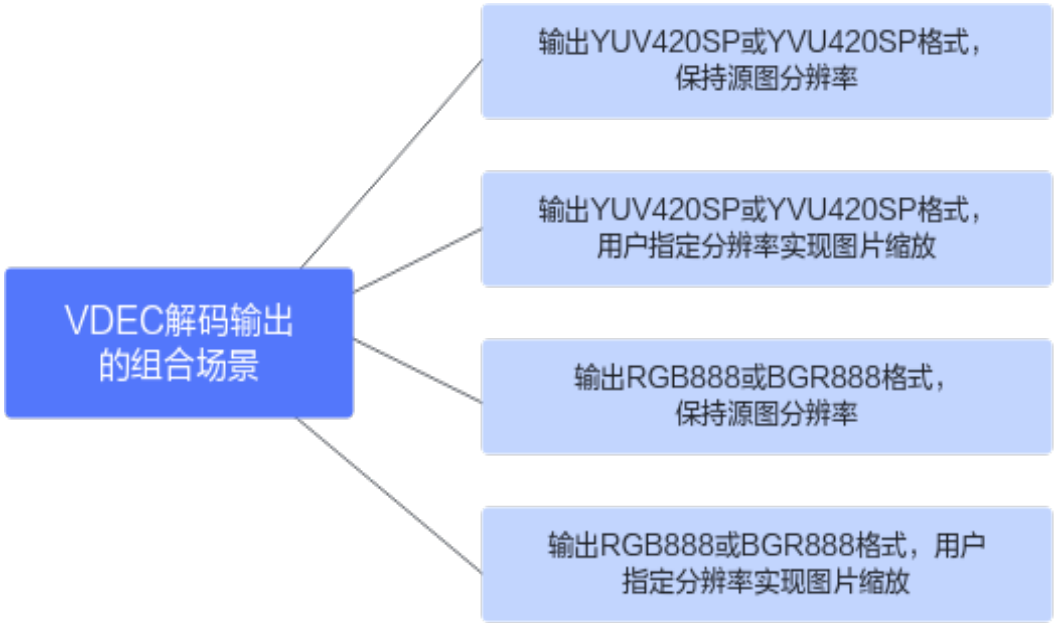
此处以批量抠图、缩放为例说明如何调用批处理接口**acldvppVpcBatchCropResizeAsync**，完整代码请单击**Link**获取。

5.7.1.3 合理选择 VDEC 视频解码输出格式和分辨率，性能更优

背景说明

如果用户需要借助DVPP进行视频解码，想要得到RGB格式图片，在Atlas 推理系列产品上，当前视频解码接口**aclvdecSendFrame**支持输出YUV420SP格式或RGB888格式，且支持在解码时对图片进行缩放，所以从提升性能的角度，可以优化代码逻辑，直接调用视频解码接口**aclvdecSendFrame**输出RGB888格式。

图 5-20 VDEC 解码输出的组合场景



基本原理

视频解码接口**aclvdecSendFrame**支持输出YUV420SP格式或RGB888格式，可设置接口参数输出不同的格式，省去调用**acldvppVpcConvertColorAsync**进行格式转换的步骤，减少接口调用。

若视频码流分辨率与模型输入图片的分辨率不一致，需要对解码后的图片进行缩放处理，也可以在视频解码接口**aclvdecSendFrame**中设置输出图片的分辨率，在解码的同时完成图片的缩放，省去单独调用缩放接口的步骤，减少接口调用。

总结下来，可以在视频解码接口**aclvdecSendFrame**中完成解码+缩放+色域转换三个功能，减少调用接口的数量，提升性能。

使用示例

视频解码+推理的完整代码请单击**Link**获取。

如果视频解码需要输出RGB888格式，并且同时实现缩放功能，您需要设置**aclvdecSendFrame**接口的output参数（表示输出图片描述信息）中的图片格式、宽、高、宽Stride、高Stride，例如：

```
//创建输出图片描述信息
acldvppPicDesc picOutputDesc = acldvppCreatePicDesc();
```

```
//定义输出图片格式、宽、高、宽Stride、高Stride以及存放输出图片数据的内存
uint32_t width = 224;
uint32_t height = 224;
uint32_t widthStride = 224 * 3;
uint32_t heightStride = 224;
uint32_t size = widthStride * heightStride;
void *picOutBufferDev = nullptr;
acldvppMalloc(&picOutBufferDev, size);

//设置输出图片格式、宽、高、宽Stride、高Stride以及存放输出图片数据的内存
acldvppSetPicDescData(picOutputDesc, picOutBufferDev);
acldvppSetPicDescSize(picOutputDesc, size);
acldvppSetPicDescFormat(picOutputDesc, PIXEL_FORMAT_RGB_888);
acldvppSetPicDescWidth(picOutputDesc, width);
acldvppSetPicDescHeight(picOutputDesc, height);
acldvppSetPicDescWidthStride(picOutputDesc, widthStride);
acldvppSetPicDescHeightStride(picOutputDesc, heightStride);

//调用解码接口
aclvdecSendFrame(channelDesc, picInputDesc, picOutputDesc, nullptr, nullptr);
```

5.7.1.4 合理使用 VDEC 解码跳帧，减少内存申请，减轻 VPC 压力，性能更优

背景说明

在视频解码+模型推理的场景下，若视频的帧数比较多，且不是每一帧都需要进行推理，对于不需要推理的帧，推荐用户使用**aclvdecSendSkippedFrame**接口进行解码，不输出解码结果。

基本原理

视频解码是需要连续的数据，解码后的数据需要输出YUV格式，则每帧数据解码后，VDEC内部还需要通过VPC进行解压缩、缩放、格式转换的流程。

如果不想获取某一帧的解码结果， 可以调用**aclvdecSendSkippedFrame**接口，不需要申请输出内存，且VDEC内部也不通过VPC模块进行解压缩、缩放、格式转换的流程。减少内存申请，也减轻了VPC的处理压力，达到提升性能的目标。

使用说明

请参见**11.17.9.4 aclvdecSendSkippedFrame**处的说明。

5.7.1.5 VPC 处理时合理选择输出格式，降低内存申请，性能更优

背景说明

在Atlas 推理系列产品上，VPC图像处理功能支持输出YUV400格式（灰度图像），如果模型推理的输入图像是灰度图像，就直接使用VPC功能，无需再使用AIPP色域转换功能。

基本原理

在Atlas 推理系列产品上，直接使用VPC功能输出YUV400格式（灰度图像），省去使用AIPP色域转换功能，降低AI Core单元的负载，从硬件上提升性能。

使用说明

参见**13.3.2.3 优化建议**中的“YUV400格式图像处理”功能说明。

关于VPC功能的示例代码，请参见5.4.1 VPC图像处理典型功能。

5.7.2 使用媒体数据处理 V2 版本接口

5.7.2.1 采用 VPC 多功能组合接口，减少系统调度压力，性能更优

背景说明

在对图像进行抠图、缩放、贴图、填充等处理时，AscendCL媒体数据处理部分提供了以下实现功能的接口：

- 一个接口只做一次操作（即单功能接口），例如hi_mpi_vpc_crop、hi_mpi_vpc_resize、hi_mpi_vpc_copy_make_border接口

  该方式下，如果想实现多个功能，例如抠图+缩放+填充，您需要调用以上3个接口。
- 一个接口做多个操作（即多功能组合接口），例如：hi_mpi_vpc_batch_crop_resize_paste、hi_mpi_vpc_batch_crop_resize_make_border接口

  该方式下，如果想实现多个功能，例如抠图+缩放+填充，您仅需要调用1个接口hi_mpi_vpc_batch_crop_resize_make_border。

单功能接口与多功能组合接口的对应关系如下。

单功能接口	多功能组合接口
• hi_mpi_vpc_crop（抠图） • hi_mpi_vpc_resize（缩放）	• hi_mpi_vpc_crop_resize（抠图缩放） • hi_mpi_vpc_crop_resize_paste（抠图缩放贴图）、hi_mpi_vpc_batch_crop_resize_paste（批量抠图缩放贴图）
• hi_mpi_vpc_crop（抠图） • hi_mpi_vpc_resize（缩放） • hi_mpi_vpc_copy_make_border（填充）	hi_mpi_vpc_crop_resize_make_border（抠图缩放填充）或hi_mpi_vpc_batch_crop_resize_make_border（批量抠图缩放填充）

基本原理

一个接口内部会有多次Host和Device的任务交互，每次交互有时延，若对于抠图、缩放等多个功能，调用多次接口，Host和Device的任务交互次数就会增加，时延自然也会随之增加。

采用多个功能组合接口，调用一个接口完成多个功能，虽然是多个功能，但对于Device来说都是一次处理（一个多功能组合接口和一个单功能接口的硬件执行时间相同），相对调用多个单功能接口，能够减少Host和Device的调度次数，减少Device的处理次数，对调度和性能有较多的提升，在性能优化时可以考虑。

使用示例

此处以批量抠图、缩放、填充为例说明如何调用多功能组合接口hi_mpi_vpc_batch_crop_resize_make_border，完整代码请单击**Link**获取。

5.7.2.2 采用 VPC 批处理接口，降低时延，性能更优

背景说明

在对图像进行抠图、缩放等处理时，AscendCL媒体数据处理部分提供了以下两类接口：

- 一次处理一张图片，例如**hi_mpi_vpc_crop_resize_make_border**接口

  该方式下，如果存在多张输入图片，一般都采用for循环的方式，针对每张图片，都调用一次**hi_mpi_vpc_crop_resize_make_border**接口。
- 一次处理多张图片（即批处理接口），例如**hi_mpi_vpc_batch_crop_resize_make_border**接口

  该方式，如果存在多张输入图片，只需调用一次**hi_mpi_vpc_batch_crop_resize_make_border**接口。

以上两类接口的对应关系表如下。

单张图片处理接口	批量图片处理接口
hi_mpi_vpc_crop_resize_paste（抠图缩放贴图）	**hi_mpi_vpc_batch_crop_resize_paste**（批量抠图缩放贴图）
hi_mpi_vpc_crop_resize_make_border（抠图缩放填充）	**hi_mpi_vpc_batch_crop_resize_make_border**（批量抠图缩放填充）

基本原理

昇腾AI处理器内置图像处理单元DVPP（Digital Video Pre-Processing），在DVPP中，有多个VPC（Vision Preprocessing Core）模块，处理图片的抠图、缩放、格式转换等任务。

在调用批处理接口时，批量任务会被均分到多个VPC模块、并行处理，批量接口的处理时延会降低，性能提升。

使用示例

此处以批量抠图、缩放、填充为例说明如何调用多功能组合接口**hi_mpi_vpc_batch_crop_resize_make_border**，完整代码请单击**Link**获取。

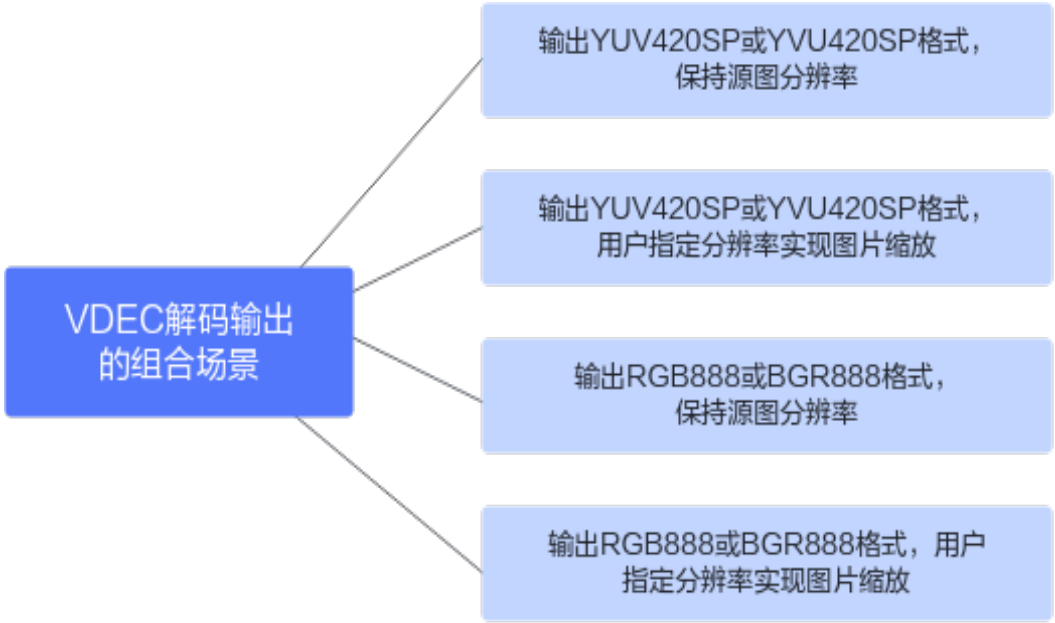
5.7.2.3 合理选择 VDEC 视频解码输出格式和分辨率，性能更优

背景说明

如果用户需要借助DVPP进行视频解码，想要得到RGB格式图片，在Atlas 推理系列产品上，当前视频解码接口**hi_mpi_vdec_send_stream**支持输出YUV420SP格式或

RGB888格式，且支持在解码时对图片进行缩放，所以从提升性能的角度，可以优化代码逻辑，直接调用视频解码接口**hi_mpi_vdec_send_stream**输出RGB888格式。

图 5-21 VDEC 解码输出的组合场景



基本原理

视频解码接口**hi_mpi_vdec_send_stream**支持输出YUV420SP格式或RGB888格式，可设置接口参数输出不同的格式，省去调用**hi_mpi_vpc_convert_color**进行格式转换的步骤，减少接口调用。

若视频码流分辨率与模型输入图片的分辨率不一致，需要对解码后的图片进行缩放处理，也可以在视频解码接口**hi_mpi_vdec_send_stream**中设置输出图片的分辨率，在解码的同时完成图片的缩放，省去单独调用缩放接口的步骤，减少接口调用。

总结下来，可以在视频解码接口**hi_mpi_vdec_send_stream**中完成解码+缩放+色域转换三个功能，减少调用接口的数量，提升性能。

使用示例

视频解码的示例代码请单击**Link**获取。

5.7.2.4 合理使用 VDEC 解码跳帧，减少内存申请，减轻 VPC 压力，性能更优

背景说明

在视频解码+模型推理的场景下，若视频的帧数比较多，且不是每一帧都需要进行推理，对于不需要推理的帧，推荐用户在调用**hi_mpi_vdec_send_stream**接口解码时设置当前帧输出不显示，不输出解码结果。

基本原理

视频解码是需要连续的数据，解码后的数据需要输出YUV格式，则每帧数据解码后，VDEC内部还需要通过VPC进行解压缩、缩放、格式转换的流程。

如果不想获取某一帧的解码结果， 在调用**hi_mpi_vdec_send_stream**接口解码时设置当前帧输出不显示，这样，就不需要申请输出内存，且VDEC内部也不通过VPC模块进行解压缩、缩放、格式转换的流程。减少内存申请，也减轻了VPC的处理压力，达到提升性能的目标。

使用说明

请参见**hi_mpi_vdec_send_stream**接口的**hi_vdec_stream**结构体内的need_display参数说明。

5.7.2.5 VPC 处理时合理选择输出格式，降低内存申请，性能更优

在Atlas 推理系列产品上，VPC图像处理功能支持输出YUV400格式（灰度图像），如果模型推理的输入图像是灰度图像，就直接使用VPC功能，无需再使用AIPP色域转换功能。

基本原理

在Atlas 推理系列产品上，直接使用VPC功能输出YUV400格式（灰度图像），省去使用AIPP色域转换功能，降低AI Core单元的负载，从硬件上提升性能。

使用说明

关于VPC功能的示例代码，请参见**5.5.1 VPC图片处理典型功能**。

5.7.2.6 合理设置队列深度，减少硬件资源浪费，提升性能

背景说明

在Atlas 推理系列产品上，可使用媒体数据处理V2版本接口，配置VPC队列深度，队列深度范围[10，350]之间。用户业务下发任务时，若VPC队列满后，会反压阻塞任务下发，影响性能。

基本原理

适当加大VPC队列深度，可以缓解用户业务下发任务与VPC内部任务处理相互之间性能波动的影响。

- 如果用户短时间内下发任务多于VPC通道处理速度，则多出来的任务会缓存在队列中，不会阻塞用户继续下发任务；
- 如果短时间内用户下发任务较少，则VPC可以处理队列中缓存的任务，不会造成空闲，导致硬件资源浪费。

用户可基于业务下发任务的速度与VPC单通道处理的性能适当调整队列任务深度：

- 用户业务创建一个通道，起多个线程往同一通道下发任务，且下发任务数超过上面单通道参考性能，则建议用户将队列深度设大；
- 用户业务创建一个通道，只有一个线程往该通道下发任务，且为下发一次任务，获取一次结果，串行执行。该情况通道任务不会堆积，不必设大，保持默认即可。

说明

VPC单通道处理的性能，请参见**11.18.16.3 性能指标说明**。

使用说明

调用**hi_mpi_vpc_create_chn**接口或**hi_mpi_vpc_sys_create_chn**接口创建VPC通道时，通过**hi_vpc_chn_attr**结构体内的attr参数设置队列深度。

关于VPC队列深度设置示例代码，参见**sample_vpc.cpp**中queue_len参数。

6 单算子调用

单算子调用基础知识
本节介绍基于AscendCL接口调用单个算子的使用场景、开发流程等。

单算子调用流程
本节介绍调用单算子的两种方式、以及这两种方式下的接口调用流程。

单算子模型执行

6.1 单算子调用基础知识

本节介绍基于AscendCL接口调用单个算子的使用场景、开发流程等。

单算子调用的使用场景

如果AI应用中不仅仅包括模型推理，还有数学运算（例如BLAS基础线性代数运算）、数据类型转换等功能，也想使用昇腾的算力，昇腾CANN还能支持吗?

答案是肯定的，昇腾CANN提供了单算子调用的方式，直接通过AscendCL接口加载并执行单个算子，省去模型构建、训练的过程，相对轻量级，又可以使用昇腾的算力。

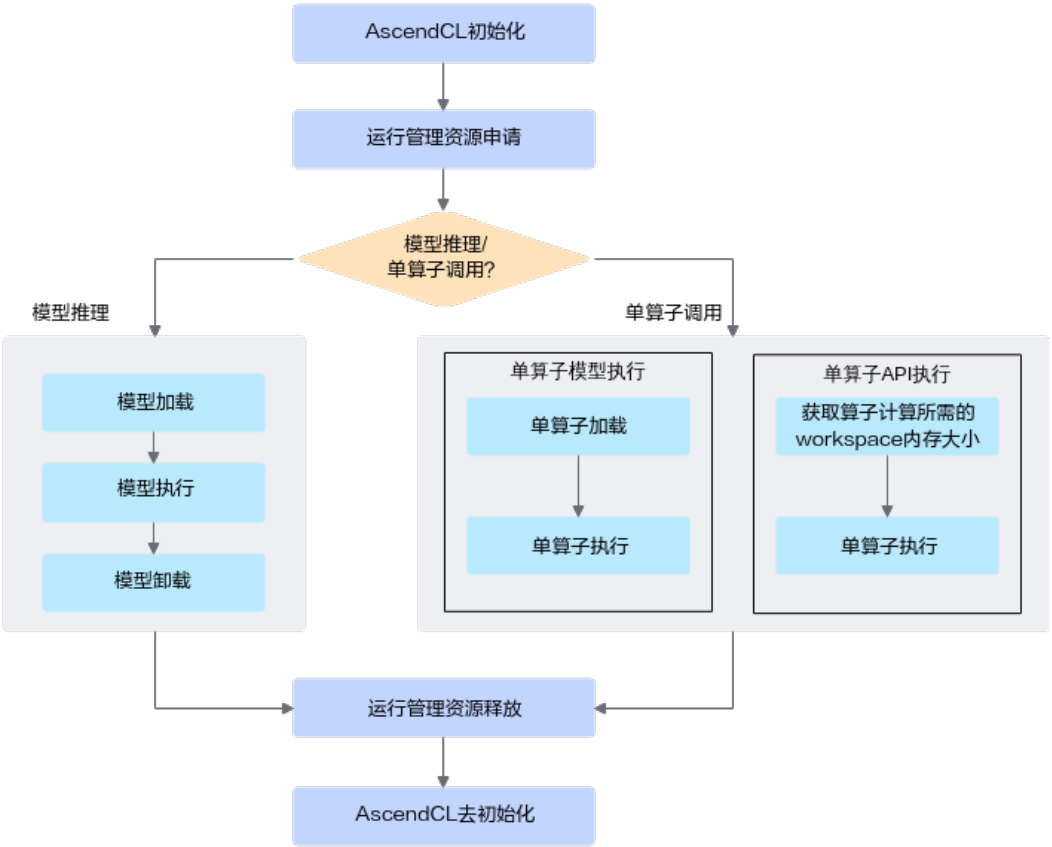
另外，自定义的算子，也可以通过单算子调用的方式来验证算子的功能。

单算子调用与模型推理的差别

在解释单算子调用与模型推理的差别前，我们先观察下面这个开发流程图，先找出基本的共同点、不同点。

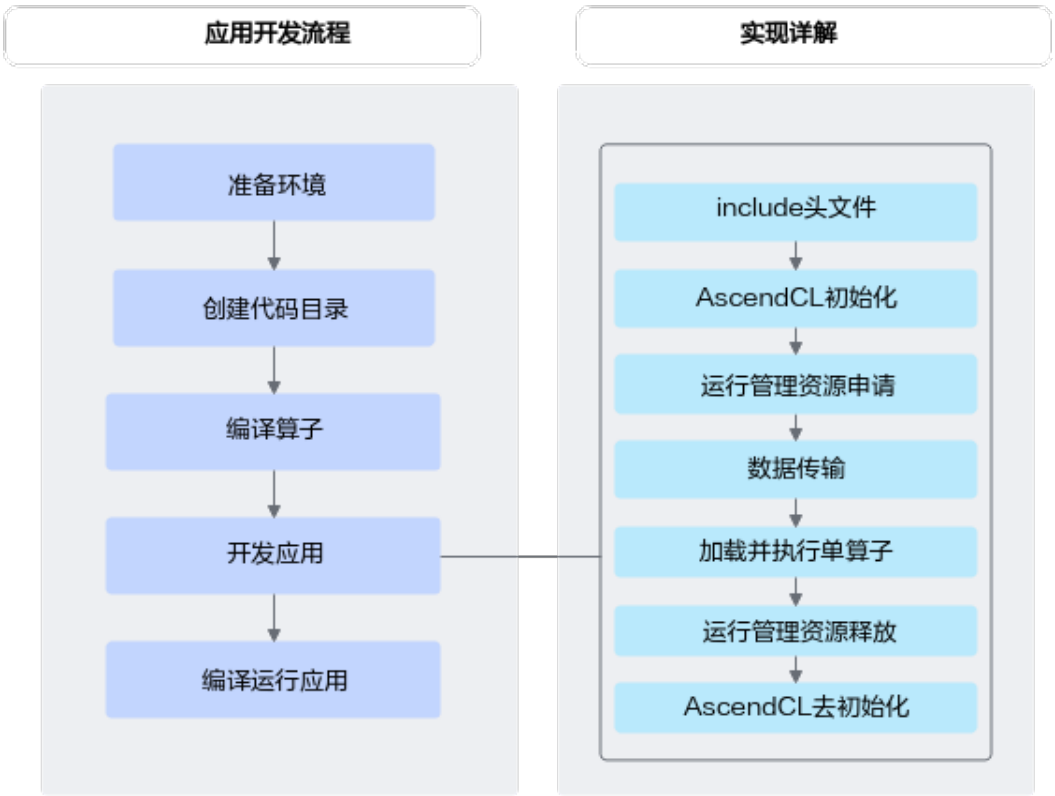
- **共同点**:
  - 不管是模型推理，还是单算子调用，**都需要**AscendCL初始化和去初始化、运行管理资源申请和释放。
  - 不管是模型推理，还是单算子调用，**都涉及**执行的步骤，但是要注意，两者执行是调用不同的AscendCL接口。
- **不同点**:
  - 模型推理涉及**模型卸载**的步骤，单算子调用不涉及。

图 6-1 单算子调用与模型推理的流程对比



单算子调用功能开发流程

图 6-2 开发流程



1. **准备环境**。

   请参见**3.3 准备开发和运行环境**。

2. **创建代码目录**。

   在开发应用前，您需要先创建目录，存放代码文件、编译脚本、测试图片数据、模型文件等。

   如下仅是示例，供参考：

```
├App名称
├── op_model            // 该目录下存放编译算子的算子描述文件
│   ├── xxx.json

├── data
│   ├── xxxxxx          // 测试数据

├── inc                 // 该目录下存放声明函数的头文件
│   ├── xxx.h

├── out                 // 该目录下存放输出结果

├── src
│   ├── xxx.json        // 系统初始化的配置文件
│   ├── CMakeLists.txt  // 编译脚本
│   ├── xxx.cpp         // 实现文件
```

3. **编译算子**。

   若基于“**6.3 单算子模型执行**”方式调用算子，则需编译算子，编译算子有以下两种方式：

- 使用ATC工具编译算子生成om模型文件

  该种方式，需要先构造*.json格式单算子描述文件（描述算子的输入、输出及属性等信息），借助ATC工具，将单算子描述文件编译成om模型文件；再分别调用AscendCL接口加载om模型文件、执行算子。

  关于ATC工具的使用说明，请参见《**ATC工具使用指南（开放态）**》。

- 也可以调用AscendCL提供的编译算子接口

  该种方式，直接调用AscendCL接口编译、执行算子。

说明

关于单算子模型执行详细的接口调用流程请参见**6.2 单算子调用流程**。

4. **开发应用**。

   依赖的头文件和库文件的说明请参见**调用接口依赖的头文件和库文件说明**。

   单算子调用的流程请参见**6.2 单算子调用流程**及相关的示例代码。

5. **编译运行应用**，请参见**8 应用调试**。

6.2 单算子调用流程

本节介绍调用单算子的两种方式、以及这两种方式下的接口调用流程。

开发应用时，如果涉及执行单个算子，请先参见**3.2 AscendCL接口调用流程**了解整体流程，再查看本节中的流程说明。

系统支持的算子请参见《**算子清单（开放态）**》。

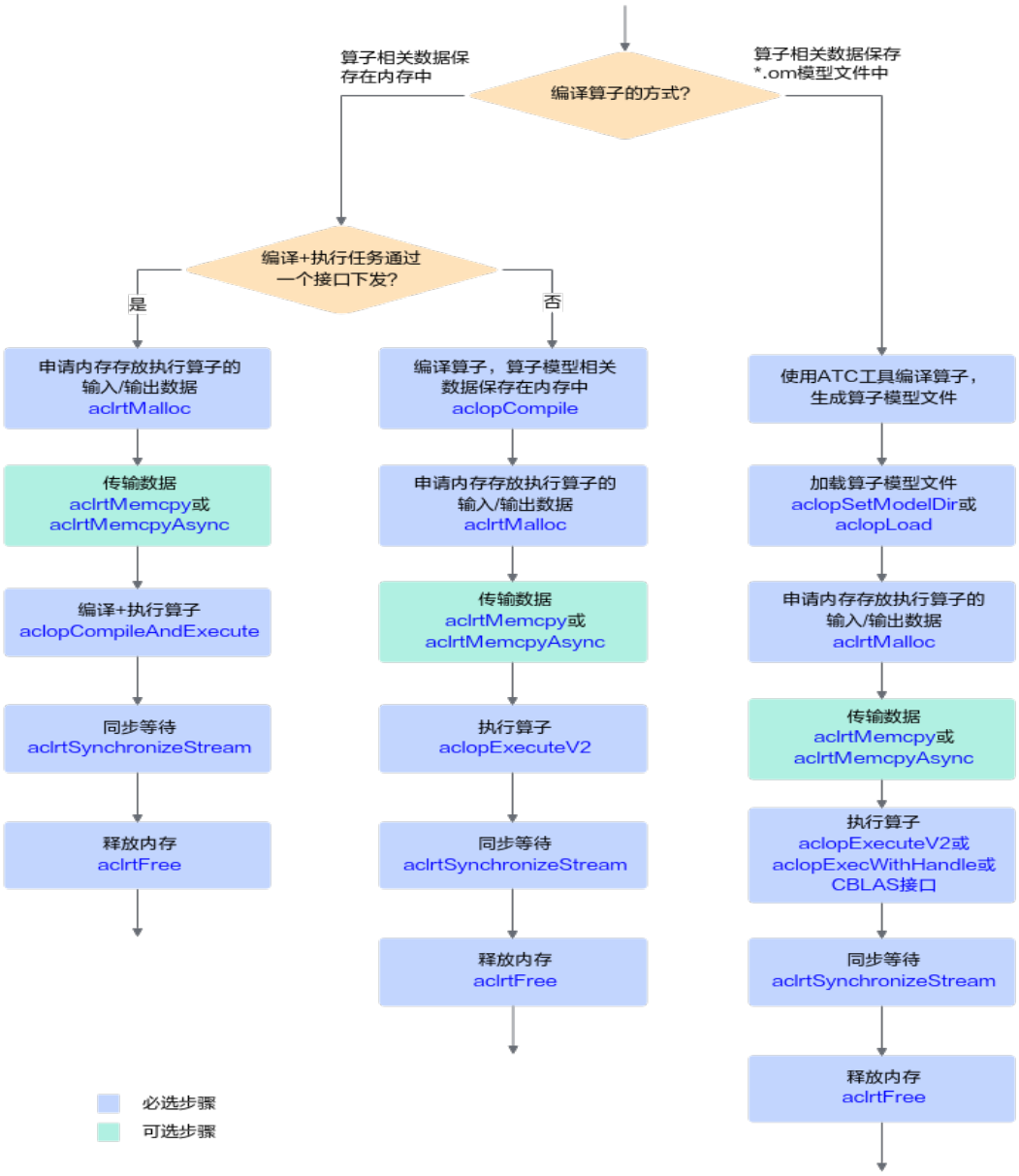
对于系统不支持的算子，用户需先参见《**TBE&AI CPU算子开发指南（开放态）**》完成自定义算子开发。

说明

对于TIK自定义动态Shape算子，需要先注册算子选择器，请参见**6.3.4 执行动态Shape算子示例代码（注册算子选择器）**。

单算子模型执行接口调用流程

图 6-3 单算子模型执行接口调用流程



关键接口的说明如下：

1. **编译算子**。

   根据算子编译的方式，可分为以下两种：

   - 编译算子后，算子相关数据保存在*.om模型文件中

     该种方式下编译算子，需使用ATC工具，详细描述请参见《**ATC工具使用指南（开放态）**》，将单算子定义文件（*.json）编译成适配昇腾AI处理器的离线模型（*.om文件）。

     编译算子后，依次进行**2**、**3**、**4**、**5**、**6**、**7**。

   - 编译算子后，算子相关数据保存在内存中

该种方式下编译算子，需调用AscendCL提供的接口，根据不同场景调用不同的接口：

- 对于同一个算子，编译一次，多次执行的场景，建议调用**aclopCompile**接口编译算子。编译算子后，依次进行**3**、**4**、**5**、**6**、**7**。
- 对于编译算子、执行算子次数相同的场景，建议先执行**3**，再调用**aclopCompileAndExecute**接口编译算子。编译算子后，再依次进行**6**、**7**。

2. **加载算子模型文件**。

   支持以下2种方式中的一种加载单算子模型文件：

   - 调用**aclopSetModelDir**接口，设置加载模型文件的目录，目录下存放单算子模型文件（*.om文件）。
   - 调用**aclopLoad**接口，从内存中加载单算子模型数据，由用户管理内存。单算子模型数据是指“单算子编译成*.om文件后，再将om文件读取到内存中”的数据。

3. 调用**aclrtMalloc**接口**申请Device上的内存**，存放执行算子的输入、输出数据。

   如果需要将Host上数据传输到Device，则需要调用**aclrtMemcpy**接口（同步接口）或**aclrtMemcpyAsync**接口（异步接口）通过内存复制的方式实现数据传输。

4. 动态Shape场景，如果无法明确算子的输出Shape时，在执行算子前，还需**推导或预估算子的输出Shape**。

   需用户调用**aclopInferShape**接口、**aclGetTensorDescNumDims**接口、**aclGetTensorDescDimV2**接口、**aclGetTensorDescDimRange**等接口，推导或预估算子的输出Shape，作为算子执行接口**aclopExecuteV2**的输入。

5. **执行算子**。

   - 对于被封装成AscendCL接口的算子（参见**CBLAS接口**），包括GEMM算子、Cast算子，目前支持以下两种执行方式：
     - 不以handle方式执行算子，接口名称中不包含“Handle”关键字，例如，调用**aclblasGemmEx**接口（封装GEMM算子）、**aclopCast**接口（封装Cast算子）等执行算子。
     - 以handle方式执行算子，接口名称中包含“Handle”关键字，例如，调用**aclblasCreateHandleForGemmEx**接口、**aclopCreateHandleForCast**接口等创建handle后，还需要调用**aclopExecWithHandle**接口执行算子。
   - 对于未被封装成AscendCL接口的算子，目前支持以下两种执行方式：
     - 不以handle方式执行算子，调用**aclopExecuteV2**接口执行算子。
     - 以handle方式执行算子，调用**aclopCreateHandle**接口创建handle，再调用**aclopExecWithHandle**接口执行算子。

说明

不以handle方式执行算子时，每次执行算子时，系统内部都会根据算子描述信息匹配内存中的模型。

以handle方式执行算子时，系统内部将算子描述信息匹配到内存中的模型，并缓存在Handle中，每次执行算子时，无需重复匹配算子与模型，因此在涉及多次执行同一个算子时，效率更高，但该方式不支持动态Shape算子，且Handle使用结束后，需调用**aclopDestroyHandle**接口释放。

6. 调用**aclrtSynchronizeStream**接口**阻塞应用运行**，直到指定Stream中的所有任务都完成。
7. 调用**aclrtFree**接口**释放内存**。

   如果需要将Device上的算子执行结果数据传输到Host，则需要调用**aclrtMemcpy**接口（同步接口）或**aclrtMemcpyAsync**接口（异步接口）通过内存复制的方式实现数据传输，然后再释放内存。

6.3 单算子模型执行

6.3.1 调用 CBLAS 接口执行算子示例代码

本节介绍基于单算子模型执行的方式调用CBLAS算子的关键接口、示例代码。

基本原理

接口调用流程，请参见**6.2 单算子调用流程**。

目前，AscendCL已将GEMM算子（用于矩阵-向量乘、矩阵-矩阵乘）、Cast算子（用于转换数据类型）封装成AscendCL的CBLAS接口，可参见**CBLAS接口**，目前支持以下两种执行方式：

- 不以handle方式执行算子，接口名称中不包含“Handle”关键字，例如，调用**aclblasGemmEx**接口（封装GEMM算子）、**aclopCast**接口（封装Cast算子）等执行算子。
- 以handle方式执行算子，接口名称中包含“Handle”关键字，例如，调用**aclblasCreateHandleForGemmEx**接口、**aclopCreateHandleForCast**接口等创建handle后，还需要调用**aclopExecWithHandle**接口执行算子。

说明

不以handle方式执行算子时，每次执行算子时，系统内部都会根据算子描述信息匹配内存中的模型。

以handle方式执行算子时，系统内部将算子描述信息匹配到内存中的模型，并缓存在Handle中，每次执行算子时，无需重复匹配算子与模型，因此在涉及多次执行同一个算子时，效率更高，但该方式不支持动态Shape算子，且Handle使用结束后，需调用**aclopDestroyHandle**接口释放。

示例代码

本章以**aclblasGemmEx**接口为例，该接口封装的是GEMM算子，该接口中矩阵乘的计算公式为：C = αAB + βC。您可以单击**acl_execute_gemm**，查看样例。

调用CBLAS接口（封装GEMM算子）分为以下几步：

1. 准备GEMM算子的模型文件。
   a. 构造GEMM算子的描述文件（*.json文件，描述输入输出Tensor描述、算子属性等）。

      GEMM算子的描述文件示例如下：

```
[
{
  "op": "GEMM",
  "input_desc": [
    {
```

```
      "format": "ND",
      "shape": [16, 16],
      "type": "float16"
    },
    {
      "format": "ND",
      "shape": [16, 16],
      "type": "float16"
    },
    {
      "format": "ND",
      "shape": [16, 16],
      "type": "float16"
    },
    {
      "format": "ND",
      "shape": [],
      "type": "float16"
    },
    {
      "format": "ND",
      "shape": [],
      "type": "float16"
    }
  ],
  "output_desc": [
    {
      "format": "ND",
      "shape": [16, 16],
      "type": "float16"
    }
  ],
  "attr": [
  {
    "name": "transpose_a",
    "type": "bool",
    "value": false
  },
  {
    "name": "transpose_b",
    "type": "bool",
    "value": false
    }
  ]
}
]
```

b. 借助ATC工具，将该算子描述文件编译成单算子模型文件（*.om文件），再分别调用AscendCL接口加载om模型文件、执行算子。

ATC工具的命令示例如下：

```
atc --singleop=$HOME/singleop/gemm.json --output=$HOME/singleop/out/op_model --soc_version=<soc_version>
```

关键参数解释如下（详细参数说明，请参见《**ATC工具使用指南（开放态）**》。）：

- --singleop：单算子描述文件（json格式）的路径。
- --output：存放单算子模型文件的目录。
- --soc_version：昇腾AI处理器的版本。

  进入“CANN软件安装目录/compiler/data/platform_config”目录，".ini"文件的文件名即为昇腾AI处理器的版本，请根据实际情况选择。

2. 编写调用CBLAS的代码逻辑。

以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考，调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。完整代码，您可以从**acl_execute_gemm**样例中查看。

```
// 1.AscendCL初始化
aclRet = aclInit(nullptr);

// 2.运行管理资源申请（使用默认Context、默认Stream，默认Stream在作为其它接口入参时，可传空指针）
aclRet = aclrtSetDevice(0);
获取软件栈的运行模式，不同运行模式影响后续的接口调用流程（例如是否进行数据传输等）
aclrtRunMode runMode;
bool g_isDevice = false;
aclError aclRet = aclrtGetRunMode(&runMode);
g_isDevice = (runMode == ACL_DEVICE);

// 3. 设置单算子模型文件所在的目录
// 该目录相对可执行文件所在的目录，例如，编译出来的可执行文件存放在run/out目录下，此处就表示run/out/op_models目录
aclopSetModelDir("op_models");

// 4. 申请内存
// 申请Device上的内存存放执行算子的输入数据
// 对于该矩阵乘示例，依次申请存放矩阵A数据、矩阵B数据、矩阵C数据、标量α数据、标量β数据的内存
aclrtMalloc((void **) &devMatrixA_, sizeA_, ACL_MEM_MALLOC_NORMAL_ONLY);
aclrtMalloc((void **) &devMatrixB_, sizeB_, ACL_MEM_MALLOC_NORMAL_ONLY);
aclrtMalloc((void **) &devMatrixC_, sizeC_, ACL_MEM_MALLOC_NORMAL_ONLY);
aclrtMalloc((void **) &devAlpha_, sizeAlphaBeta_, ACL_MEM_MALLOC_NORMAL_ONLY);
aclrtMalloc((void **) &devBeta_, sizeAlphaBeta_, ACL_MEM_MALLOC_NORMAL_ONLY);

// 申请Host上的内存，此处根据软件栈的运行模式判断是否需要申请Host上的内存
// 如果运行模式为ACL_DEVICE，则g_isDevice参数值为true，表示软件栈运行在Device侧，无需申请Host内存，无需传输图片数据或在Device内传输数据
// 如果运行模式为ACL_HOST，则g_isDevice参数值为false，表示软件栈运行在Host侧，需要申请Host内存，涉及Host和Device之间的数据传输
if (g_isDevice) {
        hostMatrixA_ = devMatrixA_;
        hostMatrixB_ = devMatrixB_;
        hostMatrixC_ = devMatrixC_;
    } else {
        aclrtMallocHost((void **) &hostMatrixA_, sizeA_);
        aclrtMallocHost((void **) &hostMatrixB_, sizeB_);
        aclrtMallocHost((void **) &hostMatrixC_, sizeC_);
    }

// 5. 准备输入数据，ReadFile为自定义函数，由用户自行管理，从文件中读入数据到内存中
size_t fileSize;
// Read matrix A
char *fileData = ReadFile("test_data/data/matrix_a.bin", fileSize, hostMatrixA_, sizeA_);
// Read matrix B
fileData = ReadFile("test_data/data/matrix_b.bin", fileSize, hostMatrixB_, sizeB_);
// Read matrix C
fileData = ReadFile("test_data/data/matrix_c.bin", fileSize, hostMatrixC_, sizeC_);
// 根据软件栈的运行模式判断是否涉及Host与Device之间的数据传输
if (!g_isDevice) {
    aclError ret = aclrtMemcpy(devMatrixA_, sizeA_, hostMatrixA_, sizeA_,
ACL_MEMCPY_HOST_TO_DEVICE);
    ret = aclrtMemcpy(devMatrixB_, sizeB_, hostMatrixB_, sizeB_, ACL_MEMCPY_HOST_TO_DEVICE);
    ret = aclrtMemcpy(devMatrixC_, sizeC_, hostMatrixC_, sizeC_, ACL_MEMCPY_HOST_TO_DEVICE);
}

aclrtMemcpyKind kind = g_isDevice ? ACL_MEMCPY_DEVICE_TO_DEVICE :
ACL_MEMCPY_HOST_TO_DEVICE;
ret = aclrtMemcpy(devAlpha_, sizeAlphaBeta_, hostAlpha_, sizeAlphaBeta_, kind);
ret = aclrtMemcpy(devBeta_, sizeAlphaBeta_, hostBeta_, sizeAlphaBeta_, kind);

// 6. 执行单算子
// 对于该示例，调用aclblasGemmEx接口（异步接口）实现矩阵-矩阵的乘法
aclblasGemmEx(ACL_TRANS_N, ACL_TRANS_N, ACL_TRANS_N, m_, n_, k_,
                devAlpha_, devMatrixA_, k_, inputType_, devMatrixB_, n_, inputType_,
```

```
                    devBeta_, devMatrixC_, n_, outputType_, ACL_COMPUTE_HIGH_PRECISION,
                    stream);
// 调用aclrtSynchronizeStream接口阻塞Host运行，直到指定Stream中的所有任务都完成
aclrtSynchronizeStream(nullptr);

// 7. 传输算子执行结果，根据软件栈的运行模式判断是否涉及Host与Device之间的数据传输
if (!g_isDevice) {
        auto ret = aclrtMemcpy(hostMatrixC_, sizeC_, devMatrixC_, sizeC_,
ACL_MEMCPY_DEVICE_TO_HOST);
}

// 8. 是否直接在终端屏幕上显示算子执行结果，由用户自行管理代码逻辑

// 9. 释放运行管理资源（默认Context、Stream无需用户释放，调用aclrtResetDevice接口后自动释放）
aclRet = aclrtResetDevice(0);

// 10.AscendCL去初始化
aclRet = aclFinalize();

// ......
```

6.3.2 执行固定 Shape 算子示例代码

本节介绍基于单算子模型执行的方式调用固定Shape算子的关键接口、示例代码。

前提条件

在调用AscendCL接口执行固定Shape算子前，需提前编译算子。此处借助ATC工具编译Add算子的模型文件为例：

1. 先构造该算子的描述文件（*.json文件，描述输入输出Tensor描述、算子属性等）。

   Add算子的描述文件示例如下：

```
[
  {
    "op": "Add",
    "input_desc": [
      {
        "format": "ND",
        "shape": [8, 16],
        "type": "int32"
      },
      {
        "format": "ND",
        "shape": [8, 16],
        "type": "int32"
      }
    ],
    "output_desc": [
      {
        "format": "ND",
        "shape": [8, 16],
        "type": "int32"
      }
    ]
  }
]
```

2. 借助ATC工具，将该算子描述文件编译成单算子模型文件（*.om文件），再分别调用AscendCL接口加载om模型文件、执行算子。

   ATC工具的命令示例如下：

```
atc --singleop=$HOME/singleop/add.json --output=$HOME/singleop/out/op_model --
soc_version=<soc_version>
```

   关键参数解释如下（详细参数说明，请参见《**ATC工具使用指南（开放态）**》。）：

- --singleop：单算子描述文件（json格式）的路径。
- --output：存放单算子模型文件的目录。
- --soc_version：昇腾AI处理器的版本。

  进入“CANN软件安装目录/compiler/data/platform_config”目录，".ini"文件的文件名即为昇腾AI处理器的版本，请根据实际情况选择。

示例代码

以下是单算子加载、执行关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考，调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。完整代码，您可以从**acl_execute_add**样例中查看。

```
// 1.AscendCL初始化
aclRet = aclInit(nullptr);

// 2.运行管理资源申请（使用默认Context、默认Stream，默认Stream在作为其它接口入参时，可传空指针）
aclRet = aclrtSetDevice(0);
获取软件栈的运行模式，不同运行模式影响后续的接口调用流程（例如是否进行数据传输等）
aclrtRunMode runMode;
bool g_isDevice = false;
aclError aclRet = aclrtGetRunMode(&runMode);
g_isDevice = (runMode == ACL_DEVICE);

// 3.加载单算子模型文件（*.om文件）
// 该目录相对可执行文件所在的目录，例如，编译出来的可执行文件存放在out目录下，此处就表示out/
op_models目录
aclRet = aclopSetModelDir("op_models");

// 4.执行算子
// opType表示算子类型名称，例如Add
// numInputs表示算子输入个数，例如Add算子是2个输入
// inputDesc表示算子输入tensor描述的数组，描述每个输入的format、shape、数据类型
// inputs表示算子输入tensor数据
// numOutputs表示算子输出个数，例如Add算子是1个输出
// outputDesc表示算子输出tensor描述的数组，描述每个输出的format、shape、数据类型
// outputs表示算子输出tensor数据
// attr表示算子属性，如果算子没有属性，也需要调用aclopCreateAttr接口创建aclopAttr类型的数据
// stream用于维护一些异步操作的执行顺序

aclopExecuteV2(opType, numInputs, inputDesc, inputs,
               numOutputs, outputDesc, outputs, attr, nullptr);

// 处理执行算子后的输出数据，例如在屏幕上显示、写入文件等，由用户根据实际情况自行实现
// ......

// 阻塞应用运行，直到指定Stream中的所有任务都完成
aclrtSynchronizeStream(nullptr);

// 5. 释放运行管理资源（默认Context、Stream无需用户释放，调用aclrtResetDevice接口后自动释放）
aclRet = aclrtResetDevice(0);

// 6.AscendCL去初始化
aclRet = aclFinalize();

// ....
```

6.3.3 执行动态 Shape 算子示例代码（不注册算子选择器）

本节介绍基于单算子模型执行的方式调用动态Shape算子的关键接口、示例代码。

基本原理

对于支持动态Shape的算子：

- 如果算子输出Shape明确时，该类算子执行的基本流程与固定Shape算子执行类似，接口调用流程请参见**6.2 单算子调用流程**，执行固定Shape算子的示例代码请参见**6.3.2 执行固定Shape算子示例代码**。
- 如果无法明确算子的输出Shape时，在调用**aclopExecuteV2**接口前，需用户调用**aclopInferShape**接口、**aclGetTensorDescNumDims**接口、**aclGetTensorDescDimV2**接口、**aclGetTensorDescDimRange**等接口，推导或预估算子的输出Shape，作为算子执行接口**aclopExecuteV2**的输入。

示例代码

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// ......

const char *opType;
int numInputs;
aclTensorDesc *inputDesc[2];
aclDataBuffer *inputs[2];
int numOutputs;
aclTensorDesc *outputDesc[1];
aclopAttr *attr;

// 此处算子输入tensor数据的内存必须根据应用运行模式来确定，应用运行在Host时，此处需申请Host上的内存；应用运行在Device时，此处需申请Device上的内存。
aclError ret = aclopInferShape(opType, numInputs, inputDesc, inputs，numOutputs, outputDesc, attr);

std::vector<std::vector<int64_t>> tensorDims; // inferShape之后的输出tensor的Shape
// 循环算子的每一个输出，推导或预估Shape值：
for (int index = 0; index < numOutputs; ++index) {
    std::vector<int64_t> dimSize; // 表示执行算子时，输出的shape
    size_t dimNums = aclGetTensorDescNumDims(outputDesc[index]);
        // 表示动态Shape场景下维度个数未知，该场景预留
    if (dimNums == ACL_UNKNOWN_RANK)  {
        // 由用户预估最大Shape值max shape
        dimSize.push_back(max_shape);
    } else {
        for (size_t i = 0; i < dimNums; ++i) {
            int64_t dim;
            ret = aclGetTensorDescDimV2(outputDesc[index], i, &dim);

                        // 表示动态Shape场景下维度值是动态的
            if(dim == -1) {
                int64_t dimRange[2];
                            // 获取Shape范围，使用该范围中的Shape最大值来构造输出tensorDesc，作为
aclopExecuteV2的输入
                ret = aclGetTensorDescDimRange(outputDesc[index], i, 2, dimRange);
                dim = dimRange[1];
            }
            dimSize.push_back(dim);
        }
    }
    tensorDims.push_back(dimSize);
}

// 构造算子输入tensorDesc和输入tensor，作为aclOpExecuteV2的输入
aclTensorDesc *inputDescNew[2];
aclDataBuffer *inputsNew[2];
aclDataBuffer *outputsNew[1];
// 以上给出了执行算子时输出的shape, 根据tensorDims中的dims构造输出tensorDesc（即outputDescNew参数值）, 用于调用aclopExecuteV2
ret = aclopExecuteV2(opType, numInputs, inputDescNew, inputsNew, numOutputs, outputDescNew,
outputsNew, attr, stream);

// 针对上面用户预估Shape值以及使用Shape范围中的最大Shape的场景，在算子执行结束后，需增加下面的调用，获取准确的shape：
```

```
// for 循环每一个输出的tensorDesc
std::vector<std::vector<int64_t>> outTensorDims; // 准确的输出tensorShape
for (int index = 0; index < numOutputs; ++index) {
    std::vector<int64_t> dimSize;
    int dimNums = aclGetTensorDescNumDims(outputDescNew[index]);
    for (int i = 0; i < dimNums; i++){
        int64_t dim;
        ret = aclGetTensorDescDimV2(outputDescNew[index], i, &dim);
        dimSize.push_back(dim);
    }
    outTensorDims.push_back(dimSize);
}
// ......
```

6.3.4 执行动态 Shape 算子示例代码（注册算子选择器）

本节介绍基于单算子模型执行的方式调用动态Shape算子的关键接口、示例代码，注册算子选择器的使用场景有限，主要针对TIK自定义算子。

前提条件

在加载与执行动态Shape算子前，您需要参见《**TBE&AI CPU算子开发指南（开放态）**》中的“专题 > TIK自定义算子动态Shape专题”中的说明开发自定义算子以及生成对应的二进制文件。

基本原理

动态Shape、注册算子选择器场景下，算子加载与执行的流程如下：

1. 资源初始化，包括AscendCL初始化、设置单算子模型文件的加载目录、指定用于运算的Device等。
   - 调用**aclInit**接口实现AscendCL初始化。
   - 调用AscendCL接口注册要编译的自定义算子：
     - 调用**aclopRegisterCompileFunc**接口注册算子选择器（即选择Tiling策略的函数），用于在算子执行时，能针对不同Shape，选择相应的Tiling策略。

       算子选择器需由用户提前定义并实现：
       - 函数原型：
         ```
         typedef aclError (*aclopCompileFunc)(int numInputs, const aclTensorDesc *const inputDesc[], int numOutputs, const aclTensorDesc *const outputDesc[], const aclopAttr *opAttr, aclopKernelDesc *aclopKernelDesc);
         ```
       - 函数实现：

         用户自行编写代码逻辑实现Tiling策略选择、Tiling参数生成，并调用**aclopSetKernelArgs**接口，设置算子Tiling参数、执行并发数等。
     - 调用**aclopCreateKernel**接口将算子注册到系统内部，用于在算子执行时，查找到算子实现代码。
   - 调用**aclrtSetDevice**接口指定运算的Device。
   - 调用**aclrtCreateContext**接口显式创建一个Context，调用**aclrtCreateStream**接口显式创建一个Stream。

     若没有显式创建Stream，则使用默认Stream，默认Stream是在调用**aclrtSetDevice**接口时隐式创建的，默认Stream作为接口入参时，直接传NULL。

2. 用户自行构造算子描述信息（输入输出Tensor描述、算子属性等）、申请存放算子输入输出数据的内存。
3. 将算子输入数据复制到Device上。
   - 调用**aclrtMemcpy**接口实现同步内存复制，内存使用结束后需及时释放。
   - 调用**aclrtMemcpyAsync**接口实现异步内存复制，内存使用结束后需及时释放。
4. 编译单算子。

   调用**aclopUpdateParams**接口编译指定算子，触发算子选择器的调用逻辑。
5. 执行单算子。

   调用**aclopExecuteV2**接口加载并执行算子。
6. 获取算子运算的输出数据。
   - 调用**aclrtMemcpy**接口实现同步内存复制，内存使用结束后需及时释放。
   - 调用**aclrtMemcpyAsync**接口实现异步内存复制，内存使用结束后需及时释放。
7. 按顺序先释放Stream资源，再释放Context资源，最后释放Device资源。
   - 调用**aclrtDestroyStream**接口释放Stream。

     不涉及显式创建Stream，使用默认Stream时，无需调用**aclrtDestroyStream**接口释放Stream。
   - 调用**aclrtDestroyContext**接口释放Context。

     不涉及显式创建Context，使用默认Context时，无需调用**aclrtDestroyContext**接口释放Context。
   - 调用**aclrtResetDevice**接口释放Device。
8. 调用**aclFinalize**接口实现AscendCL去初始化。

示例代码

在样例代码中，调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

动态Shape算子（注册算子选择器）样例的获取、编译运行请参见《**TBE&AI CPU算子开发指南（开放态）**》中的“专题 > TIK自定义算子动态Shape专题>样例使用”。

```
#include "acl/acl.h"
// ......

// 1.资源初始化
// 此处是相对路径，相对可执行文件所在的目录
aclError ret = aclInit(NULL);
aclopRegisterCompileFunc("BatchNorm", SelectAclopBatchNorm);
// 将算子Kernel的*.o文件需要用户提前编译好，并调用用户自定义函数该文件加载到内存buffer中，length表示内存大小，如果有多个算子Kernel的*.o文件，需要多次调用该接口
aclopCreateKernel("BatchNorm", "tiling_mode_1__kernel0", "tiling_mode_1__kernel0",
                  buffer, length, ACL_ENGINE_AICORE, Deallocator);

// -----自定义函数BatchNormTest(n, c, h, w)，执行以下操作-----

// 2.构造BatchNorm算子的输入输出Tensor、输入输出Tensor描述，并申请存放算子输入数据、输出数据的内存
aclTensorDesc *input_desc[3];
aclTensorDesc *output_desc[1];
input_desc[0] = aclCreateTensorDesc(ACL_FLOAT16, 4, shape_input, ACL_FORMAT_NCHW);
input_desc[1] = aclCreateTensorDesc(ACL_FLOAT16, 1, shape_gamma, ACL_FORMAT_ND);
input_desc[2] = aclCreateTensorDesc(ACL_FLOAT16, 1, shape_beta, ACL_FORMAT_ND);
```

```
output_desc[0] = aclCreateTensorDesc(ACL_FLOAT16, 4, shape_out, ACL_FORMAT_NCHW);

for (int i = 0; i < n * c * h * w; ++i) {
        input[i] = aclFloatToFloat16(1.0f);
    }

    for (int i = 0; i < c; ++i) {
        gamma[i] = aclFloatToFloat16(0.5f);
        beta[i] = aclFloatToFloat16(0.1f);
    }

aclrtMalloc(&devInput, size_input, ACL_MEM_MALLOC_NORMAL_ONLY);
aclrtMalloc(&devInput_gamma, size_gamma, ACL_MEM_MALLOC_NORMAL_ONLY);
aclrtMalloc(&devInput_beta, size_beta, ACL_MEM_MALLOC_NORMAL_ONLY);
aclrtMalloc(&devOutput, size_output, ACL_MEM_MALLOC_NORMAL_ONLY);

// 3.将算子输入数据复制到Device上
aclrtMemcpy(devInput, size_input, input, size_input, ACL_MEMCPY_HOST_TO_DEVICE);
aclrtMemcpy(devInput_gamma, size_gamma, gamma, size_gamma, ACL_MEMCPY_HOST_TO_DEVICE);
aclrtMemcpy(devInput_beta, size_beta, beta, size_beta, ACL_MEMCPY_HOST_TO_DEVICE);

// 4.调用aclopUpdateParams接口编译算子
aclopUpdateParams("BatchNorm", 3, input_desc, 1, output_desc, nullptr, ACL_ENGINE_AICORE,
ACL_COMPILE_UNREGISTERED, nullptr));

// 5.调用aclopExecuteV2接口加载并执行算子

aclopExecuteV2("BatchNorm", 3, input_desc, inputs, 1, output_desc, outputs, nullptr, stream);

// -----自定义函数BatchNormTest(n, c, h, w)，执行以上操作-----

// 6.获取算子运算的输出数据
aclrtMemcpy(output, size_output, devOutput, size_output, ACL_MEMCPY_DEVICE_TO_HOST);

// 7.按顺序释放资源
// 7.1 释放算子的输入输出Tensor描述
for (auto desc : input_desc) {
        aclDestroyTensorDesc(desc);
    }

for (auto desc : output_desc) {
        aclDestroyTensorDesc(desc);
    }
// 7.2 及时释放不使用的内存
delete[]input;
delete[]gamma;
delete[]beta;
delete[]output;
// 7.3 释放Device上的内存
aclrtFree(devInput);
aclrtFree(devInput_gamma);
aclrtFree(devInput_beta));
aclrtFree(devOutput);
// 7.4 依次释放Stream、Context、Device资源，如果未显式创建Stream、Context，则无需释放
aclrtDestroyStream(stream);
aclrtDestroyContext(context);
aclrtResetDevice(deviceId);
aclFinalize();
```

7 扩展更多特性

内存二次分配管理
用户通过AscendCL提供的内存管理接口申请内存后，若需二次分配管理，需关注各内存接口的约束，防止出现内存越界。

多Device切换
本节介绍多Device切换场景的关键接口及接口调用流程。

多模型串联推理

多Batch模型推理

队列方式模型推理
本节介绍如何基于队列加载模型、准备模型输入数据、获取模型推理输出数据。

异步模型推理
本节介绍异步推理接口如何与Callback配合使用，每隔一段时间下发一次Callback任务，获取前一段时间内的异步推理结果。

模型动态Shape输入推理

模型动态AIPP推理

Stream管理
本节介绍单Stream、多Stream的创建、销毁流程，以及多Stream同步等待的流程。

同步等待
本节介绍Device、Stream、Event在异步场景下的使用示例及关键接口。

AI Core异常信息获取

Profiling性能数据采集

溢出算子数据采集及分析

特征向量检索

共享Buffer管理

7.1 内存二次分配管理

用户通过AscendCL提供的内存管理接口申请内存后，若需二次分配管理，需关注各内存接口的约束，防止出现内存越界。

用户内存管理有两种管理方式：

- 独立内存管理，根据需要单独申请所需的内存，内存不做拆分或者二次分配。
- 内存池管理内存，用户一次性申请一块较大内存，并在使用时从这块较大内存中二次分配所需内存。

在内存二次分配时，使用如下接口从内存池申请对应内存，由于接口对申请的内存地址、大小有约束，在内存池管理时，需要关注，否则容易出现内存越界。

内存管理的总体说明请参见**11.11.1 总体说明**。

接口	用途	输入内存/输出内存
aclrtMemcpyAsync	实现内存复制，异步接口。	• 调用本接口进行内存复制时，源地址和目的地址都必须64字节对齐。
aclrtMalloc	在Device上分配size大小的线性内存，并通过*devPtr返回已分配内存的指针。本接口分配的内存会进行字节对齐，会对用户申请的size向上对齐成32字节整数倍后再多加32字节。	• 若用户需申请大块内存并自行划分、管理内存时，建议使用aclrtMallocAlign32接口，该接口相比aclrtMalloc接口，只会对用户申请的size向上对齐成32字节整数倍，不会再多加32字节。 不管是aclrtMalloc接口，还是aclrtMallocAlign32接口，若用户使用本接口申请大块内存并自行划分、管理内存时，每段内存需同时满足以下需求： – 内存大小向上对齐成32整数倍+32字节（m=ALIGN_UP[len,32]+32字节）； – 内存起始地址需满足64字节对齐（ALIGN_UP[m,64]）。 **说明** len表示某段内存的大小，ALIGN_UP[len,k]表示向上按k字节对齐：((len-1)/k+1)*k。
acldvppMalloc	该接口主要用于分配内存给Device侧媒体数据处理时使用，申请的大页内存满足数据处理的要求（例如，内存首地址128字节对齐），同步接口。 媒体数据处理各功能的详细介绍请参见**11.17 媒体数据处理V1**。	媒体数据处理的输出作为模型推理的输入时，若用户使用本接口申请大块内存并自行划分、管理内存时，每段内存需同时满足以下要求： • 内存大小向上对齐成32整数倍+32字节（m=ALIGN_UP[len,32]+32字节）； • 内存起始地址需满足128字节对齐（ALIGN_UP[m,128]）。 **说明** len表示某段内存的大小，ALIGN_UP[len,k]表示向上按k字节对齐：((len-1)/k+1)*k。

接口	用途	输入内存/输出内存
hi_mpi_dvpp_malloc	申请Device上的内存，申请的内存满足媒体数据处理的要求（例如，内存首地址128字节对齐）。 媒体数据处理各功能的详细介绍请参见11.18 媒体数据处理V2。	媒体数据处理的输出作为模型推理的输入时，若用户使用本接口申请大块内存并自行划分、管理时，每段内存需同时满足以下要求： ● 内存大小向上对齐成32整数倍+32字节（m=ALIGN_UP[len,32]+32字节）； ● 内存起始地址需满足128字节对齐（ALIGN_UP[m,128]）。 **说明** len表示某段内存的大小，ALIGN_UP[len,k]表示向上按k字节对齐：((len-1)/k+1)*k。
aclrtMallocHost	AscendCL应用程序在Host上运行时，调用该接口申请的是Host内存（该内存是锁页内存），由系统保证内存首地址64字节对齐。 AscendCL应用程序在Device上运行时，调用该接口申请的是Device内存，且Device上的内存按普通页申请，如需首地址64字节对齐，需要用户自行处理对齐。	● 若用户使用本接口申请大块内存并自行划分、管理内存时，每段内存需同时满足以下需求： – 内存大小向上对齐成32整数倍+32字节（m=ALIGN_UP[len,32]+32字节）； – 内存起始地址需满足64字节对齐（ALIGN_UP[m,64]）。 **说明** len表示某段内存的大小，ALIGN_UP[len,k]表示向上按k字节对齐：((len-1)/k+1)*k。
aclrtMallocCached	在Device上申请size大小的线性内存，通过*devPtr返回已分配内存的指针，该接口在任何场景下申请的内存都是支持cache缓存。	其它约束与aclrtMalloc接口相同。

在计算机视觉领域，一般涉及使用媒体数据处理功能，因此会涉及以上多种内存申请接口，内存首地址涉及64字节或128字节对齐，为方便统一管理，内存首地址对齐值建议选取较大的，比如内存首地址128字节对齐。

关于媒体数据处理时自行管理内存时的典型场景如下，媒体数据处理的功能点介绍请参见**5.4 DVPP图像/视频处理（媒体数据处理V1）**。

图 7-1 VDEC 场景

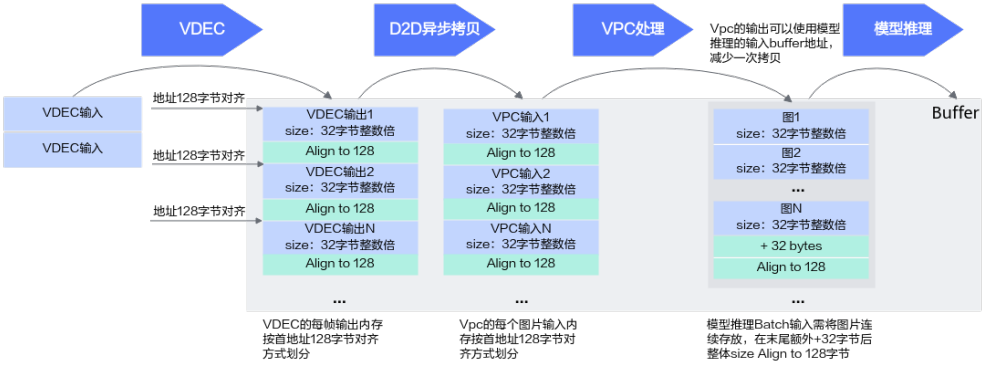
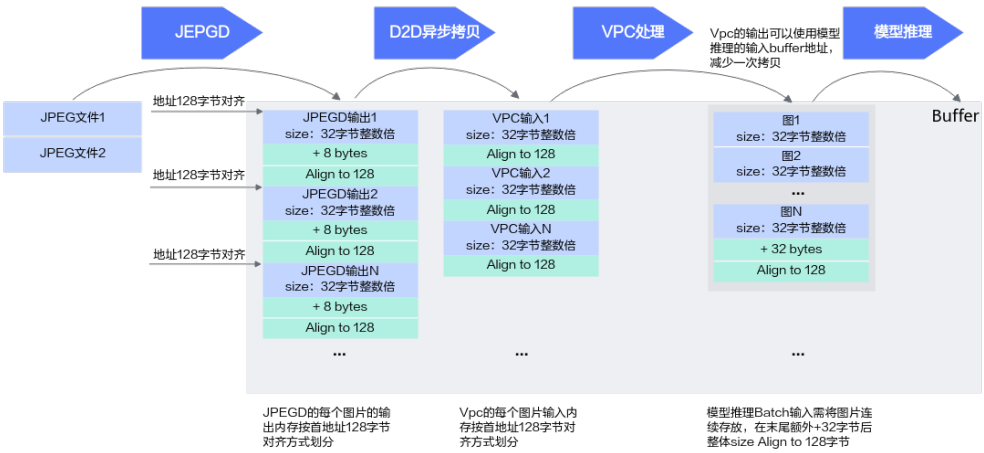


图 7-2 JPEGD 场景

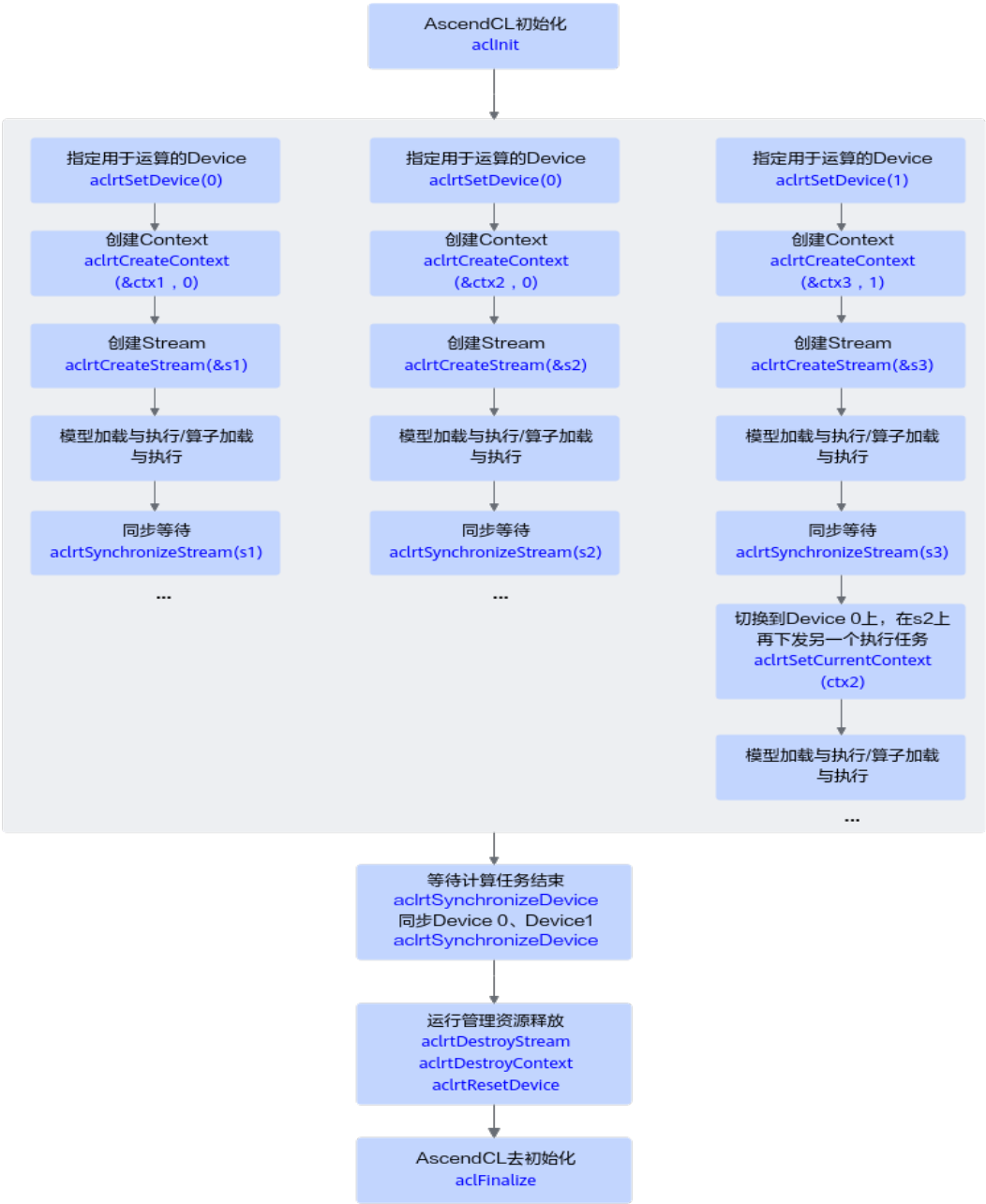


7.2 多 Device 切换

本节介绍多Device切换场景的关键接口及接口调用流程。

开发应用时，如果涉及多Device之间的任务等待，则应用程序中必须包含相关的代码逻辑，关于该场景的接口调用流程，请依次参见**3.2 AscendCL接口调用流程**以及本节中的说明。

图 7-3 同步等待流程_多 Device 场景



- 在多Device时，利用Context切换（调用**aclrtSetCurrentContext**接口）来切换Device，比使用**aclrtSetDevice**接口效率高。
- 调用**aclrtSynchronizeDevice**接口等待Device上的计算任务结束。
- 模型加载与执行的流程请参见**4 基础推理应用**。
- 算子加载与执行的流程请参见**6 单算子调用**。

7.3 多模型串联推理

多模型推理的基本流程与单模型类似，请参见**4 基础推理应用**。

多模型推理与单模型推理在AscendCL接口使用上的不同点如下：

- 关于模型加载，如果涉及多个模型，需调用多次模型加载接口。模型加载请参见**4.6.1 模型加载**。
- 关于模型执行，如果涉及多个模型，需调用多次模型执行接口。模型执行请参见**4.6.2 模型执行**。

  例如，调用**aclmdlExecute**接口实现同步模型推理。

7.4 多 Batch 模型推理

多Batch推理的基本流程与单Batch类似，请参见**4 基础推理应用**。

多Batch推理与单Batch推理的不同点在于：

- 多Batch场景下，在构建模型时，使用ATC工具的input_shape参数需根据实际的Batch数设置batch size值，详细说明请参见《**ATC工具使用指南（开放态）**》。
- 在推理前，需要编写一段代码，实现逻辑为：等输入数据满足多Batch（例如：8Batch）的要求，申请Device上的内存存放多Batch的数据，作为模型推理的输入。如果最后循环遍历所有的输入数据后，仍不满足多Batch的要求，则直接将剩余数据作为模型推理的输入。

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考，此处以8Batch为例：

```
uint32_t batchSize = 8;
uint32_t deviceNum = 1;
uint32_t deviceId = 0;

// 获取模型第一个输入的大小
uint32_t modelInputSize = aclmdlGetInputSizeByIndex(modelDesc, 0);
// 获取每个Batch输入数据的大小
uint32_t singleBuffSize = modelInputSize / batchSize;

// 定义该变量，用于累加batch size是否达到8Batch
uint32_t cnt = 0;
// 定义该变量，用于描述每个文件读入内存时的位置偏移
uint32_t pos = 0;

void* p_batchDst = NULL;
std::vector<std::string>inferFile_vec;

for (int i = 0; i < files.size(); ++i)
    {
        // 每8个文件，申请一次Device上的内存，存放8Batch的输入数据
        if (cnt % batchSize == 0)
        {
            pos = 0;
            inferFile_vec.clear();
            // 申请Device上的内存
            ret = aclrtMalloc(&p_batchDst, modelInputSize, ACL_MEM_MALLOC_NORMAL_ONLY);
        }

        // TODO: 从某个目录下读入文件，计算文件大小fileSize

        // 根据文件大小，申请内存，存放文件数据
        ret = aclrtMallocHost(&p_imgBuf, fileSize);

        // 将数据传输到Device的内存
        ret = aclrtMemcpy((uint8_t *)p_batchDst + pos, fileSize, p_imgBuf, fileSize,
ACL_MEMCPY_HOST_TO_DEVICE);
        pos += fileSize;
        // 及时释放不使用的内存
        aclrtFreeHost(p_imgBuf);
```

```
        // 将第i个文件存入vector中，同时cnt+1
        inferFile_vec.push_back(files[i]);
        cnt++;

        // 每8Batch的输入数据送给模型推理进行推理
        if (cnt % batchSize == 0)
        {
            // TODO: 创建aclmdlDataset、aclDataBuffer类型的数据，用于描述模型的输入、输出数据
            // TODO: 调用aclmdlExecute接口执行模型推理
            // TODO: 推理结束后，调用aclrtFree接口释放Device上的内存
        }
    }

// 如果最后循环遍历所有的输入数据后，仍不满足多Batch的要求，则直接将剩余数据作为模型推理的输入。
if (cnt % batchSize != 0)
  {
        // TODO: 创建aclmdlDataset、aclDataBuffer类型的数据，用于描述模型的输入、输出数据
        // TODO: 调用aclmdlExecute接口执行模型推理
        // TODO: 推理结束后，调用aclrtFree接口释放Device上的内存
  }
```

7.5 队列方式模型推理

本节介绍如何基于队列加载模型、准备模型输入数据、获取模型推理输出数据。

基本原理

- 调用**aclmdlLoadFromFileWithQ**或**aclmdlLoadFromMemWithQ**接口以队列方式加载模型。
- 调用**acltdtEnqueueData**接口将模型的输入数据传入队列，由AscendCL内部根据队列中的输入数据进行推理，无需调用模型执行的接口。
- 调用**acltdtDequeueData**接口等待模型推理执行完毕，再由用户从输出内存中获取结果数据。

说明

如果涉及多线程，当模型有多个输入时，多个输入数据的入队任务（即调用acltdtEnqueueData接口）必须在同一个线程中；当模型有多个输出时，多个输出数据的出队任务（即调用acltdtDequeueData接口）必须在同一个线程中。

示例代码

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
#include "acl/acl.h"
// ......
// 1. AscendCL初始化
// 此处的..表示相对路径，相对可执行文件所在的目录
// 例如，编译出来的可执行文件存放在out目录下，此处的..就表示out目录的上一级目录
const char *aclConfigPath = "../src/acl.json";
aclInit(aclConfigPath);

// 2. 申请运行管理资源
extern bool g_isDevice;
aclrtSetDevice(deviceId_);
aclrtCreateContext(&context_, deviceId_);
aclrtCreateStream(&stream_);
// 获取当前昇腾AI软件栈的运行模式，根据不同的运行模式，后续的内存申请、内存复制等接口调用方式不同
aclrtRunMode runMode;
aclError ret = aclrtGetRunMode(&runMode);
g_isDevice = (runMode == ACL_DEVICE);
```

```
// 3. 加载并执行模型
// 此处的..表示相对路径，相对可执行文件所在的目录
// 例如，编译出来的可执行文件存放在out目录下，此处的..就表示out目录的上一级目录
const char* omModelPath = "../model/resnet50.om"

// 3.1 创建模型的输入队列，如果模型有多个输入，则创建多个输入队列，此处以一个输入为例
acltdtQueueAttr *attr = acltdtCreateQueueAttr();
uint32_t *inputQueueList = new (nothrow) uint32_t[num];
int32_t inputNum = 1;

for (int n = 0; n < inputNum; n++) {
        uint32_t inputQid;
        ret = acltdtCreateQueue(attr, &inputQid);
        inputQueueList[n] = inputQid;
    }

// 3.2 创建模型的输出队列，如果模型有多个输出，则创建多个输出队列，此处以一个输出为例
uint32_t *outputQueueList = new (nothrow) uint32_t[num];
int32_t outputNum = 1;

for (int n = 0; n < outputNum; n++) {
        uint32_t outputQid;
        ret = acltdtCreateQueue(attr, &outputQid);
        outputQueueList[n] = outputQid;
    }

// 3.3 加载模型
uint32_t modelId;
ret= aclmdlLoadFromFileWithQ(modelPath, &modelId,
                             inputQueueList, inputNum, outputQueueList, outputNum);

// 3.4 根据模型的ID，获取该模型的描述信息
aclmdlDesc *modelDesc = aclmdlCreateDesc();
ret = aclmdlGetDesc(modelDesc, modelId);

// 3.5 获取模型的输入内存大小，如果模型有多个输入，则需要获取每个输入的内存大小，此处以一个输入为例
size_t inputSize = aclmdlGetInputSizeByIndex(modelDesc, 0);

// 3.6 加载测试图片数据，进行推理，并对推理结果数据进行后处理
string testFile[] = {
        "../data/dog1_1024_683.bin",
        "../data/dog2_1024_683.bin"
};

for (size_t index = 0; index < sizeof(testFile) / sizeof(testFile[0]); ++index) {
        uint32_t devBufferSize;
        void *picDevBuffer = nullptr;
        // 自定义函数ReadBinFile，根据昇腾AI软件栈的运行模式，申请对应的内存，再调用C++标准库中的函数将
图片数据读入内存
        ret = Utils::ReadBinFile(testFile[index], picDevBuffer, devBufferSize);

        // 将模型输入数据传入队列中，执行模型推理，-1是表示阻塞程序直到输入数据入队完成
        ret = acltdtEnqueueData(inputQid, picDevBuffer, devBufferSize, nullptr, 0, -1, 0);
        // 获取每个输出的大小
    size_t dataSize = aclmdlGetOutputSizeByIndex(modelDesc, 0);
    void *data = nullptr;
        size_t retDataSize = 0;
        // 为模型输出数据申请内存
    if (!g_isDevice) {
            aclError aclRet = aclrtMallocHost(&data, dataSize);
     } else {
        aclError aclRet = aclrtMalloc(&data, dataSize, ACL_MEM_MALLOC_NORMAL_ONLY);
     }
         // 等待模型推理执行完毕，从输出内存中获取结果数据，-1是表示阻塞程序直到推理输出数据入队完成
         ret = acltdtDequeueData(outputQid, data, dataSize, &retDataSize, nullptr, 0, -1);
         //将输出内存中的数据转换为float类型
         float *outData = NULL;
         outData = reinterpret_cast<float*>(data);
```

```
        //屏显每张图片的top5置信度的类别编号
        map<float, int, greater<float> > resultMap;
        for (int j = 0; j < len / sizeof(float); ++j) {
            resultMap[*outData] = j;
            outData++;
        }
        int cnt = 0;
        for (auto it = resultMap.begin(); it != resultMap.end(); ++it) {
            // print top 5
            if (++cnt > 5) {
                break;
            }
            INFO_LOG("top %d: index[%d] value[%lf]", cnt, it->second, it->first);
        }
        if (!g_isDevice) {
            aclError aclRet = aclrtFreeHost(picDevBuffer);
            aclrtFreeHost(data);
    } else {
        aclError aclRet = aclrtFree(picDevBuffer);
            aclrtFree(data);
    }
}

// 4. 卸载模型，并释放模型推理相关资源
aclmdlUnload(modelId);
aclmdlDestroyDesc(modelDesc);
acltdtDestroyQueue(inputQid);
acltdtDestroyQueue(outputQid);
acltdtDestroyQueueAttr(attr);

// 6. 释放运行管理资源
aclrtDestroyStream(stream_);
aclrtDestroyContext(context_);
aclrtResetDevice(deviceId_);

// 7. AscendCL去初始化
aclFinalize();
// ......
```

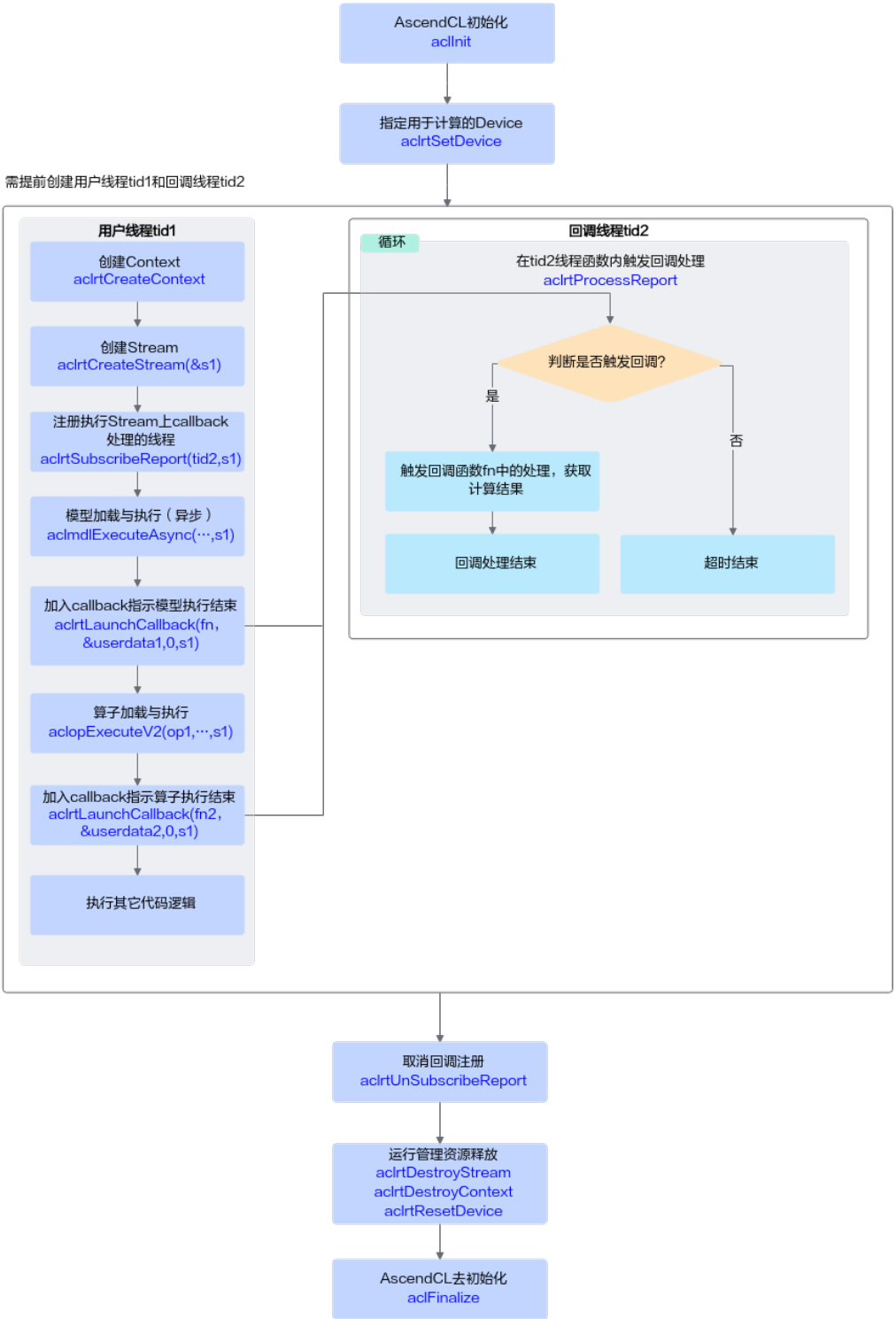
7.6 异步模型推理

本节介绍异步推理接口如何与Callback配合使用，每隔一段时间下发一次Callback任务，获取前一段时间内的异步推理结果。

接口调用流程

开发应用时，如果涉及异步场景下的同步等待，则应用程序中必须包含相关的代码逻辑，关于该场景的接口调用流程，请参见下图。

图 7-4 同步等待流程_Callback 场景



关键接口说明如下：

1. 回调函数需由用户提前创建，用于获取并处理模型推理或算子执行的结果。
2. 线程需由用户提前创建，并自定义线程函数，在线程函数内调用**aclrtProcessReport**接口，设置超时时间，等待**aclrtLaunchCallback**接口下发的回调任务执行。

3. 调用**aclrtSubscribeReport**接口：指定处理Stream上回调函数的线程，线程与**2**中创建的线程保持一致。
4. 异步推理时调用**aclmdlExecuteAsync**接口。

   对于异步接口，还需调用**aclrtSynchronizeStream**接口阻塞应用程序运行，直到指定Stream中的所有任务都完成。

   用户可以在**aclrtSynchronizeStream**接口之后一次性获取所有图片的异步推理结果，但如果图片数据量较大的情况下，需要等待的时间比较长，这时可以使用Callback功能，每隔一段时间下发一次Callback任务，获取前一段时间内的异步推理结果。
5. 调用**aclrtLaunchCallback**接口：在Stream的任务队列中下发一个回调任务，系统内部在执行到该回调任务时，会在Stream上注册的线程（通过**aclrtSubscribeReport**接口注册的线程）中执行回调函数，回调函数与**1**中的回调函数保持一致。

   每调用一次本接口，就会触发一次回调函数的执行。
6. 调用**aclrtUnSubscribeReport**接口：取消线程注册（Stream上的回调函数不再由指定线程处理）。

示例代码

您可以从**10.8.1 样例介绍**中获取完整样例代码。

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
#include "acl/acl.h"
// ......
// 1. AscendCL初始化
// 此处的..表示相对路径，相对可执行文件所在的目录
// 例如，编译出来的可执行文件存放在out目录下，此处的..就表示out目录的上一级目录
const char *aclConfigPath = "../src/acl.json";
aclError ret = aclInit(aclConfigPath);

// 2. 申请运行管理资源
extern bool g_isDevice;
ret = aclrtSetDevice(deviceId_);
ret = aclrtCreateContext(&context_, deviceId_);
ret = aclrtCreateStream(&stream_);
// 获取当前昇腾AI软件栈的运行模式，根据不同的运行模式，后续的内存申请、内存复制等接口调用方式不同
aclrtRunMode runMode;
ret = aclrtGetRunMode(&runMode);
g_isDevice = (runMode == ACL_DEVICE);

// 3. 申请模型推理资源

// 此处的..表示相对路径，相对可执行文件所在的目录
// 例如，编译出来的可执行文件存放在out目录下，此处的..就表示out目录的上一级目录
const char* omModelPath = "../model/resnet50.om"

// 3.1 加载模型
// 根据模型文件获取模型执行时所需的权值内存大小、工作内存大小，并申请权值内存、工作内存
ret = aclmdlQuerySize(omModelPath, &modelMemSize_, &modelWeightSize_);
ret = aclrtMalloc(&modelMemPtr_, modelMemSize_, ACL_MEM_MALLOC_NORMAL_ONLY);
ret = aclrtMalloc(&modelWeightPtr_, modelWeightSize_, ACL_MEM_MALLOC_NORMAL_ONLY);

// 加载离线模型文件，模型加载成功，返回标识模型的ID。
ret = aclmdlLoadFromFileWithMem(modelPath, &modelId_, modelMemPtr_,
        modelMemSize_, modelWeightPtr_, modelWeightSize_);

// 3.2 根据模型的ID，获取该模型的描述信息
modelDesc_ = aclmdlCreateDesc();
ret = aclmdlGetDesc(modelDesc_, modelId_);
```

```
// 3.3 自定义函数InitMemPool，初始化内存池，存放模型推理的输入数据、输出数据
// -----自定义函数InitMemPool内部的关键实现-----
string testFile[] = {
        "../data/dog1_1024_683.bin",
        "../data/dog2_1024_683.bin"
    };
size_t fileNum = sizeof(testFile) / sizeof(testFile[0]);
// g_memoryPoolSize表示内存池中的内存块的个数默认为100个
for (size_t i = 0; i < g_memoryPoolSize; ++i) {
        size_t index = i % (sizeof(testFile) / sizeof(testFile[0]));
        // model process
        uint32_t devBufferSize;
        // 自定义函数GetDeviceBufferOfFile，完成以下功能：
        // 获取存放输入图片数据的内存及内存大小、将图片数据传输到Device
        void *picDevBuffer = Utils::GetDeviceBufferOfFile(testFile[index], devBufferSize);
        aclmdlDataset *input = nullptr;
        // 自定义函数CreateInput，创建aclmdlDataset类型的数据input，用于存放模型推理的输入数据
        Result ret = CreateInput(picDevBuffer, devBufferSize, input);
        aclmdlDataset *output = nullptr;
        // 自定义函数CreateOutput，创建aclmdlDataset类型的数据output，用于存放模型推理的输出数据，
modelDesc表示模型的描述信息
        CreateOutput(output, modelDesc);
        {
            std::lock_guard<std::recursive_mutex> lk(freePoolMutex_);
            freeMemoryPool_[input] = output;
        }
}
// -----自定义函数InitMemPool内部的关键实现-----

// 4 模型推理
// 4.1 创建线程tid，并将该tid线程指定为处理Stream上回调函数的线程
// 其中ProcessCallback为线程函数，在该函数内调用aclrtProcessReport接口，等待指定时间后，触发回调函数
处理
pthread_t tid;
(void)pthread_create(&tid, nullptr, ProcessCallback, &s_isExit);

// 4.2 指定处理Stream上回调函数的线程
aclError aclRt = aclrtSubscribeReport(tid, stream_);

// 4.2 创建回调函数，用户处理模型推理的结果，由用户自行定义
void ModelProcess::CallBackFunc(void *arg)
{
    std::map<aclmdlDataset *, aclmdlDataset *> *dataMap =
        (std::map<aclmdlDataset *, aclmdlDataset *> *)arg;

    aclmdlDataset *input = nullptr;
    aclmdlDataset *output = nullptr;
    MemoryPool *memPool = MemoryPool::Instance();

    for (auto& data : *dataMap) {
        ModelProcess::OutputModelResult(data.second);
        memPool->FreeMemory(data.first, data.second);
    }

    delete dataMap;
}

// 4.3 自定义函数ExecuteAsync，执行模型推理
// -----自定义函数ExecuteAsync内部的关键实现开始-----
// g_callbackInterval表示callback间隔，默认为1，表示1次异步推理后，下发一次callback任务
bool isCallback = (g_callbackInterval != 0);
    size_t callbackCnt = 0;
    std::map<aclmdlDataset *, aclmdlDataset *> *dataMap = nullptr;
    aclmdlDataset *input = nullptr;
    aclmdlDataset *output = nullptr;
    MemoryPool *memPool = MemoryPool::Instance();
    // g_executeTimes表示执行模型异步推理的次数，默认为100次
    for (uint32_t cnt = 0; cnt < g_executeTimes; ++cnt) {
```

```
        if (memPool->mallocMemory(input, output) != SUCCESS) {
            ERROR_LOG("get free memory failed");
            return FAILED;
        }
        // 执行异步推理
        aclError ret = aclmdlExecuteAsync(modelId_, input, output, stream_);

        if (isCallback) {
            if (dataMap == nullptr) {
                dataMap = new std::map<aclmdlDataset *, aclmdlDataset *>;
                if (dataMap == nullptr) {
                    ERROR_LOG("malloc list failed, modelId is %u", modelId_);
                    memPool->FreeMemory(input, output);
                    return FAILED;
                }
            }
            (*dataMap)[input] = output;
            callbackCnt++;
            if ((callbackCnt % g_callbackInterval) == 0) {
                // 在Stream的任务队列中增加一个需要执行的回调函数
                ret = aclrtLaunchCallback(CallBackFunc, (void *)dataMap, ACL_CALLBACK_BLOCK, stream_);
                if (ret != ACL_SUCCESS) {
                    ERROR_LOG("launch callback failed, index=%zu", callbackCnt);
                    memPool->FreeMemory(input, output);
                    delete dataMap;
                    return FAILED;
                }
                dataMap = nullptr;
            }

        }
    }
// -----自定义函数ExecuteAsync内部的关键实现结束-----

// 4.4 对于异步推理，需阻塞应用程序运行，直到指定Stream中的所有任务都完成
aclrtSynchronizeStream(stream_);

// 4.5 取消线程注册，Stream上的回调函数不再由指定线程处理
aclRt = aclrtUnSubscribeReport(static_cast<uint64_t>(tid), stream_);
s_isExit = true;
(void)pthread_join(tid, nullptr);

// 5 释放运行管理资源
aclError ret = aclrtDestroyStream(stream_);
ret = aclrtDestroyContext(context_);
ret = aclrtResetDevice(deviceId_);

// 6 AscendCL去初始化
ret = aclFinalize();
// ......
```

7.7 模型动态 Shape 输入推理

7.7.1 使用约束

- 对同一个模型，**AIPP**（包括静态AIPP和动态AIPP）与动态维度（ND格式）不能同时使用。
- 对同一个模型，以下方式，**只能选择其中一种**：
  - 调用**aclmdlSetDatasetTensorDesc**接口设置Shape范围
  - 调用**aclmdlSetDynamicBatchSize**接口设置动态Batch
  - 调用**aclmdlSetDynamicHWSize**接口设置动态分辨率

  - 调用**aclmdlSetInputDynamicDims**接口设置动态维度的维度值
- **申请模型推理的输出内存时**，可以按照各档位的实际大小申请内存，也可以调用**aclmdlGetOutputSizeByIndex**接口获取内存大小后再申请内存（建议使用该方式，确保内存足够）。
- **动态AIPP和动态Batch同时使用时**：
  - 调用**aclmdlCreateAIPP**接口设置batchSize时，batchSize要设置为最大batch size。
  - 模型中需要进行动态AIPP处理的data节点，其对应的输入内存大小需按照最大Batch来申请。
- **动态AIPP和动态分辨率同时使用时**：
  - 若在设置动态AIPP参数时，开启了抠图或缩放或补边功能，则不能与动态分辨率同时使用。
  - 若在设置动态AIPP参数时，未开启抠图或缩放或补边功能，在与动态分辨率同时使用时，需确保通过**aclmdlSetAIPPSrcImageSize**接口设置的宽、高与通过**aclmdlSetDynamicHWSize**接口设置的宽、高相等，都必须设置成模型转换时动态分辨率最大档位的宽、高。
  - 模型中需要进行动态AIPP处理的data节点，其对应的输入内存大小需按照最大分辨率（宽、高）来申请。
- **动态AIPP和动态Shape输入（设置Shape范围）同时使用时**，动态AIPP的输出图片宽、高要在所设置的Shape范围内。
- **静态AIPP和动态分辨率同时使用时**，由于动态分辨率场景下输入图片的宽和高不确定，因此在使用ATC工具的insert_op_conf参数传入AIPP配置文件时，AIPP配置文件中不能开启Crop和Padding功能，并且需要将配置文件中的src_image_size_w和src_image_size_h取值设置为0。

7.7.2 动态 Batch/动态分辨率/动态维度（设置多档维度值）

本节介绍动态Batch/动态分辨率/动态维度功能涉及的关键接口、接口调用流程及示例代码。

接口调用流程

动态Shape输入场景下模型推理与**4 基础推理应用**的流程类似，都涉及AscendCL初始化与去初始化、运行管理资源申请与释放、模型构建、模型加载、模型执行、模型卸载等。

本节中重点描述动态Shape输入场景下模型推理与**4 基础推理应用**的不同之处：

1. **构建模型时**，需配置动态Batch、动态分辨率、动态维度（ND格式）相关的信息：

   若模型推理时包含动态Batch特性，在模型推理时，需调用AscendCL提供的接口设置模型推理时需使用的batch size，模型支持的batch size已提前在构建模型时配置（使用ATC工具的dynamic_batch_size参数）。

   若模型推理时包含动态分辨率特性，在模型推理时，需调用AscendCL提供的接口设置模型推理时需使用的分辨率，模型支持的分辨率已提前在构建模型时配置（使用ATC工具的dynamic_image_size参数）。

   若模型推理时包含动态维度（ND格式）特性，在模型推理时，需调用AscendCL提供的接口设置模型推理时需使用的维度值，模型支持哪些维度值已提前在构建模型时配置（使用ATC工具的dynamic_dims参数）。

构建模型成功后，在生成的om模型中，会新增相应的输入（下文简称动态Batch/动态分辨率/动态维度输入），在模型推理时通过该新增的输入提供具体的Batch值/分辨率/维度值。

例如，a输入的batch size是动态的，在om模型中，会新增与a对应的b输入来描述a的batch信息。在模型执行时，准备a输入的数据结构请参见**准备模型执行的输入/输出数据结构**，准备b输入的数据结构、设置b输入的数据请参见**2**。

ATC工具的参数说明请参见《**ATC工具使用指南（开放态）**》。

2. **在执行模型推理前**：
   - 需准备动态Batch/动态分辨率/动态维度输入的数据结构：
     i. 申请动态Batch/动态分辨率/动态维度输入对应的内存前，需要先调用**aclmdlGetInputIndexByName**接口根据输入名称（固定为ACL_DYNAMIC_TENSOR_NAME）获取模型中标识该输入的index。

     说明

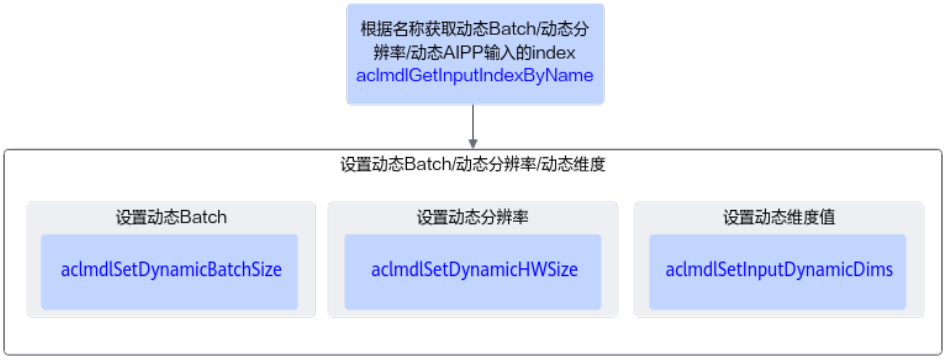
     ACL_DYNAMIC_TENSOR_NAME是一个宏，宏的定义如下：
     ```
     #define ACL_DYNAMIC_TENSOR_NAME "ascend_mbatch_shape_data"
     ```

     ii. 调用**aclmdlGetInputSizeByIndex**根据index获取输入内存大小。
     iii. 调用**aclrtMalloc**接口根据**2.ii**中的大小申请内存。

     申请动态Batch/动态分辨率/动态AIPP/动态维度输入对应的内存后，无需用户设置该内存中的数据（否则可能会导致业务异常），用户调用**2.ii**中的接口后，系统会自动向该内存中填入数据。

     iv. 调用**aclCreateDataBuffer**接口创建aclDataBuffer类型的数据，用于存放动态Batch/动态分辨率/动态维度输入数据的内存地址、内存大小。
     v. 调用**aclmdlCreateDataset**接口创建aclmdlDataset类型的数据，并调用**aclmdlAddDatasetBuffer**接口向aclmdlDataset类型的数据中增加aclDataBuffer类型的数据。
   - 需设置动态Batch/动态分辨率/动态维度参数值：

     图 7-5 接口调用流程

     i. 调用**aclmdlGetInputIndexByName**接口根据输入名称（固定为ACL_DYNAMIC_TENSOR_NAME）获取模型中标识该输入的index。
     ii. 设置动态Batch/动态分辨率/动态维度参数值。
       - 调用**aclmdlSetDynamicBatchSize**接口设置动态Batch。

         此处设置的batch size只能是构建模型时设置的Batch档位中的某一个。

也可以调用**aclmdlGetDynamicBatch**接口获取指定模型支持的Batch档位数以及每一档中的batch size。

- 调用**aclmdlSetDynamicHWSize**接口设置动态分辨率。

  此处设置的分辨率只能是构建模型时设置的分辨率档位中的某一个。

  也可以调用**aclmdlGetDynamicHW**接口获取指定模型支持的分辨率档位数以及每一档中的宽、高。

- 调用**aclmdlSetInputDynamicDims**接口设置动态维度的维度值。

  此处设置的动态维度的值只能是构建模型时设置的档位中的某一档。

  也可以调用**aclmdlGetInputDynamicDims**接口获取指定模型支持的动态维度档位数以及每一档中的值。

动态 Batch 示例代码

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1.模型加载，加载成功后，再设置动态Batch
// ......

// 2.准备模型描述信息modelDesc_，准备模型的输入数据input_和模型的输出数据output_
// ......

// 3.自定义函数，设置动态Batch
int  ModelSetDynamicInfo()
{
        size_t index;
        // 3.1 获取动态Batch输入的index，标识动态Batch输入的输入名称固定为ACL_DYNAMIC_TENSOR_NAME
        aclError ret = aclmdlGetInputIndexByName(modelDesc_, ACL_DYNAMIC_TENSOR_NAME, &index);
        // 3.2 设置Batch
        // modelId_表示加载成功的模型的ID，input_表示aclmdlDataset类型的数据，index表示标识动态Batch输入的输入index，batchSize表示Batch数（此处以8为例）
        uint64_t batchSize = 8;
        ret = aclmdlSetDynamicBatchSize(modelId_, input_, index, batchSize);
        // ......
}

// 4.自定义函数，执行模型
int ModelExecute(int index)
{
        aclError ret;
        // 4.1 调用自定义函数，设置动态Batch
    ret = ModelSetDynamicInfo();
        // 4.2 执行模型，modelId_表示加载成功的模型的ID，input_和output_分别表示模型的输入和输出
        ret = aclmdlExecute(modelId_, input_, output_);
        // ......
}
// 5.处理模型推理结果
// TODO
```

动态分辨率示例代码

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1.模型加载，加载成功后，再设置动态分辨率
// ......

// 2.准备模型描述信息modelDesc_，准备模型的输入数据input_和模型的输出数据output_
// ......
```

```
// 3.自定义函数，设置动态分辨率
int  ModelSetDynamicInfo()
{
        size_t index;
                // 3.1 获取动态分辨率输入的index，标识动态分辨率输入的输入名称固定为
ACL_DYNAMIC_TENSOR_NAME
        aclError ret = aclmdlGetInputIndexByName(modelDesc_, ACL_DYNAMIC_TENSOR_NAME, &index);
                // 3.2 设置输入图片分辨率，modelId_表示加载成功的模型的ID，input_表示aclmdlDataset类型的数
据，index表示标识动态分辨率输入的输入index
                uint64_t height = 224;
        uint64_t width = 224;
        ret = aclmdlSetDynamicHWSize(modelId_, input_, index, height, width);
                // ......
}

// 4.自定义函数，执行模型
int ModelExecute(int index)
{
        aclError ret;
        // 4.1 调用自定义函数，设置动态分辨率
    ret = ModelSetDynamicInfo();
        // 4.2 执行模型，modelId_表示加载成功的模型的ID，input_和output_分别表示模型的输入和输出
        ret = aclmdlExecute(modelId_, input_, output_);
        // ......
}

// 5.处理模型推理结果
// TODO
```

ND 格式，动态维度示例代码

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1.模型加载，加载成功后，再设置动态维度
// ......

// 2.准备模型描述信息modelDesc_，准备模型的输入数据input_和模型的输出数据output_
// ......

// 3.自定义函数，设置动态维度
int  ModelSetDynamicInfo()
{
        size_t index;
        // 3.1 获取动态维度输入的index，标识动态维度输入的输入名称固定为ACL_DYNAMIC_TENSOR_NAME
        aclError ret = aclmdlGetInputIndexByName(modelDesc_, ACL_DYNAMIC_TENSOR_NAME, &index);
        // 3.2 设置具体档位信息，包括维度数dimCount和各个维度的数值，modelId_表示加载成功的模型的ID，
input_表示aclmdlDataset类型的数据，index表示标识动态维度输入的输入index
        aclmdlIODims currentDims;
        currentDims.dimCount = 4;
        currentDims.dims[0] = 8;
        currentDims.dims[1] = 3;
        currentDims.dims[2] = 224;
        currentDims.dims[3] = 224;
        ret = aclmdlSetInputDynamicDims(modelId_, input_, index, &currentDims);
        // ......
}

// 4.自定义函数，执行模型
int ModelExecute(int index)
{
        aclError ret;
        // 4.1 调用自定义函数，设置动态维度
    ret = ModelSetDynamicInfo();
        // 4.2 执行模型，modelId_表示加载成功的模型的ID，input_和output_分别表示模型的输入和输出
        ret = aclmdlExecute(modelId_, input_, output_);
        // ......
}
```

```
// 5.处理模型推理结果
// TODO
```

7.7.3 动态 Shape 输入（设置 Shape 范围）

本节介绍动态Shape输入场景下，如何设置Shape范围，介绍其关键接口、接口调用流程及示例代码。

Atlas 200/300/500 推理产品**不支持该特性**。

接口调用流程

如果模型输入Shape是动态的，在模型执行之前调**aclmdlSetDatasetTensorDesc**设置该输入的tensor描述信息（主要是设置Shape信息），在模型执行之后，调用**aclmdlGetDatasetTensorDesc**接口获取模型动态输出的Tensor描述信息，再进一步调用**aclTensorDesc**下的操作接口获取输出Tensor数据占用的内存大小、Tensor的Format信息、Tensor的维度信息等。

关键原理说明如下：

1. 构建模型。

   模型推理场景下，对于动态Shape的输入数据，使用ATC工具转换模型时，通过input_shape参数设置输入Shape范围。

   ATC工具的参数说明请参见《**ATC工具使用指南（开放态）**》。

2. 加载模型。

   模型加载的详细流程，请参见**4.6.1 模型加载**，模型加载成功后，返回标识模型的ID。

3. 创建aclmdlDataset类型的数据，用于描述模型执行的输入、输出。

   详细调用流程请参见**准备模型执行的输入/输出数据结构**。

   注意点如下：

   - 当调用**aclmdlGetInputSizeByIndex**获取到的size大小为0时，表示该输入的Shape是动态的，用户可根据实际情况预估一块较大的输入内存。
   - 当调用**aclmdlGetOutputSizeByIndex**获取到的size大小为0时，表示该输出的Shape是动态的，用户可根据实际情况预估一块较大的输出内存。

4. 在成功加载模型之后，执行模型之前，调用**aclmdlSetDatasetTensorDesc**接口设置动态Shape输入的Tensor描述信息（主要是设置Shape信息）。

   在调用**aclCreateTensorDesc**接口，创建Tensor描述信息时，设置Shape信息，包括维度个数、维度大小，此处设置的维度个数、维度大小必须在模型构建时设置的输入Shape范围内，模型构建的详细说明请参见**4.2 模型构建**。

5. 执行模型。

   例如，调用**aclmdlExecute**接口（同步接口）执行模型。

6. 获取模型执行的结果数据。

   调用**aclmdlGetDatasetTensorDesc**接口获取动态Shape输出的Tensor描述信息，再利用**aclTensorDesc**数据类型的操作接口获取Tensor描述信息的属性，此处以获取size（表示Tensor数据占用的空间大小）为例，然后从内存中读取对应size的数据。

示例代码

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 此例中假设模型的第一个输入为动态输入，其index为0；模型的第一个输出为动态输出，其index为0。

// 1.模型加载，加载成功后，再设置动态输入的Tensor描述信息，主要设置动态输入的Shape信息
// ......

// 2.准备模型描述信息modelDesc_，准备模型的输入数据input_和模型的输出数据output_
// 此处需注意：
// 当利用aclmdlGetInputSizeByIndex获取到的size大小为0时，表示该输入的Shape是动态的，用户可根据实际情况预估一块较大的输入内存
// 当利用aclmdlGetOutputSizeByIndex获取到的size大小为0时，表示该输出的Shape是动态的，用户可根据实际情况预估一块较大的输出内存
// ......

// 3.自定义函数，设置动态输入的Tensor描述信息
void SetTensorDesc()
{
    // ......
    // 创建Tensor描述信息，其中dataType和format不需要设置，AscendCL直接从模型中获取，此处使用的默认值
    shape需要和给定的输入数据的shape一致
    int64_t shapes = {1, 3, 224, 224};
    aclTensorDesc *inputDesc = aclCreateTensorDesc(ACL_FLOAT, 4, shapes, ACL_FORMAT_NCHW);
    // 设置index为0的动态输入的Tensor描述信息
    aclError ret = aclmdlSetDatasetTensorDesc(input_, inputDesc, 0);
    // ......
}
// 4.自定义函数，执行模型，并获取动态输出的Tensor描述信息
void ModelExecute()
{
    aclError ret;
    // 调用自定义接口，设置动态输入的Tensor描述信息
    SetTensorDesc();
        // 执行模型
    ret = aclmdlExecute(modelId, input_, output_);
    // 获取index为0的动态输出的Tensor描述信息
        aclTensorDesc *outputDesc = aclmdlGetDatasetTensorDesc(output_, 0);
    // 利用aclTensorDesc数据类型的操作接口获取outputDesc的属性，此处需要获取size（表示Tensor数据占用的空间大小），然后从内存中读取对应size的数据
    string outputFileName = ss.str();
        FILE *outputFile = fopen(outputFileName.c_str(), "wb");
    size_t outputDesc_size = aclGetTensorDescSize(outputDesc);
        aclDataBuffer *dataBuffer = aclmdlGetDatasetBuffer(output_, 0);
        void *data = aclGetDataBufferAddr(dataBuffer);
    void *outHostData = nullptr;
        // 调用aclrtGetRunMode接口获取软件栈的运行模式，并根据运行模式判断是否进行数据传输
        aclrtRunMode runMode;
        ret = aclrtGetRunMode(&runMode);
        if (runMode == ACL_HOST) {
            ret = aclrtMallocHost(&outHostData, outputDesc_size);
       // 由于动态shape申请的内存比较大，而真实数据的大小是outputDesc_size，所以此处用真实数据大小去拷贝内存
        ret = aclrtMemcpy(outHostData, outputDesc_size, data, outputDesc_size, ACL_MEMCPY_DEVICE_TO_HOST);
        fwrite(outHostData, outputDesc_size, sizeof(char), outputFile);
            ret = aclrtFreeHost(outHostData);
    } else {
        // if app is running in host, write model output data into result file
            fwrite(data, outputDesc_size, sizeof(char), outputFile);
    }
    fclose(outputFile);
        // ......
}

// 5.处理模型推理结果
// TODO
```

7.8 模型动态 AIPP 推理

7.8.1 使用约束

- **动态AIPP和动态Batch同时使用时**：
  - 调用**aclmdlCreateAIPP**接口设置batchSize时，batchSize要设置为最大batch size。
  - 模型中需要进行动态AIPP处理的data节点，其对应的输入内存大小需按照最大Batch来申请。
- **动态AIPP和动态分辨率同时使用时**：
  - 若在设置动态AIPP参数时，开启了抠图或缩放或补边功能，则不能与动态分辨率同时使用。
  - 若在设置动态AIPP参数时，未开启抠图或缩放或补边功能，在与动态分辨率同时使用时，需确保通过**aclmdlSetAIPPSrcImageSize**接口设置的宽、高与通过**aclmdlSetDynamicHWSize**接口设置的宽、高相等，都必须设置成模型转换时动态分辨率最大档位的宽、高。
  - 模型中需要进行动态AIPP处理的data节点，其对应的输入内存大小需按照最大分辨率（宽、高）来申请。
- **动态AIPP和动态Shape输入（设置Shape范围）同时使用时**，动态AIPP的输出图片宽、高要在所设置的Shape范围内。
- 对同一个模型，AIPP（包括静态AIPP和动态AIPP）与动态维度（ND格式）不能同时使用。
- AscendCL还提供了基于DVPP（Digital Vision Pre-Processing）硬件进行媒体数据处理的功能，包括缩放、抠图、格式转换、图片编解码、视频编解码等，功能比AIPP丰富，但对于输入/输出图片、内存有一定的约束。

  基于DVPP的媒体数据处理接口介绍，请参见**5 媒体数据处理（含图像/视频等）**。

7.8.2 动态 AIPP（单个动态 AIPP 输入）

本节介绍单个动态AIPP输入的模型，在执行模型推理时的关键接口、示例代码。

接口调用流程

动态AIPP场景下模型推理与**4 基础推理应用**的流程类似，都涉及AscendCL初始化与去初始化、运行管理资源申请与释放、模型构建、模型加载、模型执行、模型卸载等。

本节中重点描述动态AIPP场景下模型推理与**4 基础推理应用**的不同之处：

- **模型构建时**，需配置动态AIPP相关参数：

  构建模型时，需通过ATC工具的insert_op_conf参数配置动态AIPP模式。ATC工具的参数说明请参见《**ATC工具使用指南（开放态）**》。

  构建模型成功后，在生成的om模型中，会新增相应的输入（下文简称动态AIPP输入），在模型推理时通过该新增的输入提供具体的AIPP配置值。

  例如，a输入的AIPP配置是动态的，在om模型中，会有与a对应的b输入来描述a的AIPP配置信息。在模型执行时，准备a输入的数据结构请参见**准备模型执行的输入/输出数据结构**，准备b输入的数据结构、设置b输入的数据请参见以下内容。
- **在执行模型推理前**：
  - 准备动态AIPP输入的数据结构：

i. 申请动态AIPP输入对应的内存前，需要先调用**aclmdlGetInputIndexByName**接口根据输入名称（固定为ACL_DYNAMIC_AIPP_NAME）获取模型中标识该输入的index。

说明

ACL_DYNAMIC_AIPP_NAME是一个宏，宏的定义如下：

```
#define ACL_DYNAMIC_AIPP_NAME "ascend_dynamic_aipp_data"
```

ii. 调用**aclmdlGetInputSizeByIndex**根据index获取输入内存大小。

iii. 调用**aclrtMalloc**接口根据**ii**中的大小申请内存。

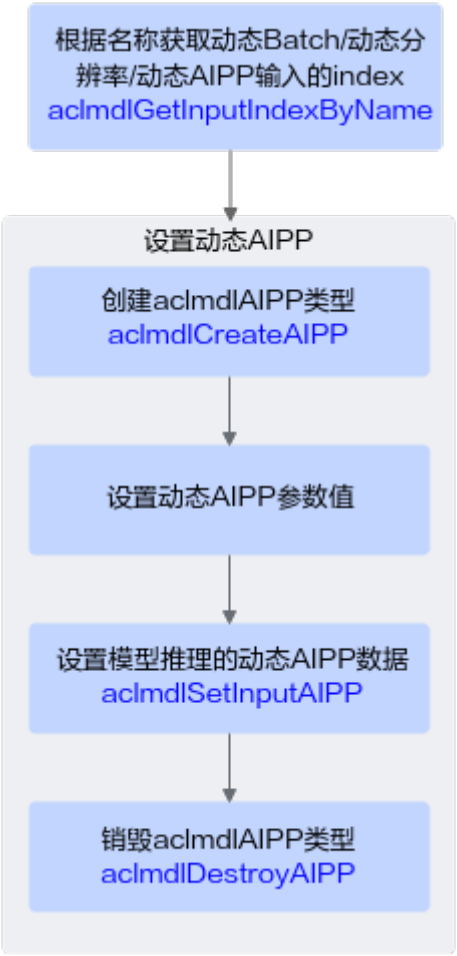
申请动态AIPP输入对应的内存后，无需用户设置该内存中的数据（否则可能会导致业务异常），用户调用**ii**中的接口后，系统会自动向该内存中填入数据。

iv. 调用**aclCreateDataBuffer**接口创建aclDataBuffer类型的数据，用于存放动态AIPP输入数据的内存地址、内存大小。

v. 调用**aclmdlCreateDataset**接口创建aclmdlDataset类型的数据，并调用**aclmdlAddDatasetBuffer**接口向aclmdlDataset类型的数据中增加aclDataBuffer类型的数据。

– 设置动态AIPP参数值：

图 7-6 接口调用流程



i. 调用**aclmdlGetInputIndexByName**接口根据输入名称（固定为ACL_DYNAMIC_AIPP_NAME）获取模型中标识该输入的index。

ii. 设置动态AIPP参数值。

1) 调用**aclmdlCreateAIPP**接口创建aclmdlAIPP类型。

2) 根据实际需求，调用**11.27.60 aclmdlAIPP**数据类型下的操作接口设置动态AIPP参数值。

3) 动态AIPP场景下，**aclmdlSetAIPPSrcImageSize**接口（设置原始图片的宽和高）必须调用。

4) 调用**aclmdlSetInputAIPP**接口设置模型推理时的动态AIPP数据。

5) 及时调用**aclmdlDestroyAIPP**接口销毁aclmdlAIPP类型。

示例代码

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1.模型加载，加载成功后，再设置动态AIPP参数值
// ......

// 2.准备模型描述信息modelDesc_，准备模型的输入数据input_和模型的输出数据output_

// 3.自定义函数，设置动态AIPP参数值
int  ModelSetDynamicAIPP()
{
    // 3.1 获取标识动态AIPP输入的index
    size_t index;
    // modelDesc_为aclmdlCreateDesc表示模型描述信息，根据1中加载成功的模型的ID，获取该模型的描述信息
    aclError ret = aclmdlGetInputIndexByName(modelDesc_, ACL_DYNAMIC_AIPP_NAME, &index);

    // 3.2 设置动态AIPP参数值
    uint64_t batchNumber = 1;
    aclmdlAIPP *aippDynamicSet = aclmdlCreateAIPP(batchNumber);
    ret = aclmdlSetAIPPSrcImageSize(aippDynamicSet, 256, 224);
    ret = aclmdlSetAIPPInputFormat(aippDynamicSet, ACL_YUV420SP_U8);
    ret = aclmdlSetAIPPCscParams(aippDynamicSet, 1, 256, 443, 0, 256, -86, -178, 256, 0, 350, 0, 0, 0, 0,
128, 128);
    ret = aclmdlSetAIPPRbuvSwapSwitch(aippDynamicSet, 0);
    ret = aclmdlSetAIPPDtcPixelMean(aippDynamicSet, 0, 0, 0, 0, 0);
    ret = aclmdlSetAIPPDtcPixelMin(aippDynamicSet, 0, 0, 0, 0, 0);
    ret = aclmdlSetAIPPPixelVarReci(aippDynamicSet, 1.0, 1.0, 1.0, 1.0, 0);
    ret = aclmdlSetAIPPCropParams(aippDynamicSet, 1, 2, 2, 224, 224, 0);
    ret = aclmdlSetInputAIPP(modelId_, input_, index, aippDynamicSet);
    ret = aclmdlDestroyAIPP(aippDynamicSet);

    // ......
}
// 4.自定义函数，执行模型
int ModelExecute(int index)
{
        aclError ret;
        // 4.1 调用自定义函数，设置动态AIPP参数值
    ret = ModelSetDynamicAIPP();
        // 4.2 执行模型，modelId_表示加载成功的模型的ID，input_和output_分别表示模型的输入和输出
        ret = aclmdlExecute(modelId_, input_, output_);
        // ......
}

// 5.处理模型推理结果
// TODO
```

7.8.3 动态 AIPP（多个动态 AIPP 输入）

本节介绍多个动态AIPP输入的模型，在执行模型推理时的关键接口、示例代码。

接口调用流程

模型有多个动态AIPP输入时的推理基本流程与单个动态AIPP输入类似，请参见**7.8.2 动态AIPP（单个动态AIPP输入）**。

多个动态AIPP输入与单个动态AIPP输入的不同点如下：

- 需调用**aclmdlGetAippType**接口查询指定模型的指定输入是否有关联的动态AIPP输入，若有，则输出标识动态AIPP输入的index，该参数值可作为**aclmdlSetAIPPByInputIndex**接口的入参之一，来设置对应输入上的动态AIPP参数值。
- 为避免在错误的输入上设置动态AIPP参数，用户可调用**aclmdlGetInputNameByIndex**接口先获取指定输入index的输入名称，根据输入名称所对应的index设置动态AIPP参数。

示例代码

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1.模型加载，加载成功后，再设置动态AIPP参数值
// ......

// 2.准备模型描述信息modelDesc_，准备模型的输入数据input_和模型的输出数据output_
// ......

// 3.自定义函数，设置动态AIPP参数值
int  ModelSetDynamicAIPP()
{
    // 3.1 获取标识动态AIPP输入的index
    std::vector<size_t> dataNeedDynamicAipp;
    for (size_t index = 0; index < aclmdlGetNumInputs(modelDesc_); ++index) {
        aclmdlInputAippType aippType;
        size_t dynamicAttachedDataIndex;
        aclError ret = aclmdlGetAippType(modelId_, index, &aippType, &dynamicAttachedDataIndex);
        if (aippType == ACL_DATA_WITH_DYNAMIC_AIPP) {
            dataNeedDynamicAipp.push_back(index);
        }
    }

    // 3.2 当前示例中以2个动态AIPP输入为例，用户可根据实际情况修改
    if (dataNeedDynamicAipp.size() != 2) {
        return -1;
    }
    // 创建第一个动态aipp配置参数
    uint64_t batchNumber1 = 1;
    aclmdlAIPP *aippDynamicSet1 = aclmdlCreateAIPP(batchNumber1);
    ret = aclmdlSetAIPPSrcImageSize(aippDynamicSet1, 256, 224);
    ret = aclmdlSetAIPPInputFormat(aippDynamicSet1, ACL_YUV420SP_U8);
    ret = aclmdlSetAIPPCscParams(aippDynamicSet1, 1, 256, 443, 0, 256, -86, -178, 256, 0, 350, 0, 0, 0, 0,
128, 128);
    ret = aclmdlSetAIPPRbuvSwapSwitch(aippDynamicSet1, 0);
    ret = aclmdlSetAIPPDtcPixelMean(aippDynamicSet1, 0, 0, 0, 0, 0);
    ret = aclmdlSetAIPPDtcPixelMin(aippDynamicSet1, 0, 0, 0, 0, 0);
    ret = aclmdlSetAIPPPixelVarReci(aippDynamicSet1, 1.0, 1.0, 1.0, 1.0, 0);
    ret = aclmdlSetAIPPCropParams(aippDynamicSet1, 1, 2, 2, 224, 224, 0);
    // 设置模型推理时的动态aipp参数值
    ret = aclmdlSetAIPPByInputIndex(modelId_, input_, dataNeedDynamicAipp[0], aippDynamicSet1);
    ret = aclmdlDestroyAIPP(aippDynamicSet1);
```

```
    // 创建第二个动态aipp配置参数
    uint64_t batchNumber2 = 2;
    aclmdlAIPP *aippDynamicSet2 = aclmdlCreateAIPP(batchNumber2);
    ret = aclmdlSetAIPPSrcImageSize(aippDynamicSet2, 224, 224);
    // 此处可以继续调用其它AIPP参数设置接口
    // 设置模型推理时的动态aipp参数值
    ret = aclmdlSetAIPPByInputIndex(modelId_, input_, dataNeedDynamicAipp[1], aippDynamicSet2);
    ret = aclmdlDestroyAIPP(aippDynamicSet2);
}

// 4.自定义函数，执行模型
int ModelExecute(int index)
{
        aclError ret;
        // 4.1 调用自定义函数，设置动态AIPP参数值
    ret = ModelSetDynamicAIPP();
        // 4.2 执行模型，modelId_表示加载成功的模型的ID，input_和output_分别表示模型的输入和输出
        ret = aclmdlExecute(modelId_, input_, output_);
        // ......
}

// 5.处理模型推理结果
// TODO
```

7.9 Stream 管理

本节介绍单Stream、多Stream的创建、销毁流程，以及多Stream同步等待的流程。

在AscendCL中，Stream是一个任务队列，应用程序通过Stream来管理任务的并行，一个Stream内部的任务保序执行，即Stream根据发送过来的任务依次执行；不同Stream中的任务并行执行。一个默认Context下会挂一个默认Stream，如果不显式创建Stream，可使用默认Stream。默认Stream作为接口入参时，直接传NULL。

AscendCL提供以下几种Stream管理机制，其中，多Stream场景下的同步等待流程请参见**多Stream时的同步等待流程**：

- **单线程单Stream**
- **单线程多Stream**
- **多线程多Stream**

单线程单 Stream

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
#include "acl/acl.h"
// ......
// 显式创建一个Stream
aclrtStream stream;
aclrtCreateStream(&stream);

// 调用触发任务的接口，传入stream参数
aclrtMemcpyAsync(dstPtr, dstSize, srcPtr, srcSize, ACL_MEMCPY_HOST_TO_DEVICE, stream);
// 调用aclrtSynchronizeStream接口，阻塞应用程序运行，直到指定Stream中的所有任务都完成。
aclrtSynchronizeStream(stream);

// Stream使用结束后，显式销毁Stream
aclrtDestroyStream(stream);
// ......
```

单线程多 Stream

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
#include "acl/acl.h"
// ......
int32_t deviceId = 0;
uint32_t modelId1 = 0;
uint32_t modelId2 = 1;
aclrtContext context;
aclrtStream stream1;
aclrtStream stream2;

// 如果只创建了一个Context，线程默认将这个Context作为线程当前的Context；
// 如果是多个Context，则需要调用aclrtSetCurrentContext接口设置当前线程的Context
 aclrtCreateContext(&context, deviceId);

 aclrtCreateStream(&stream1);
// 调用触发任务的接口，例如异步模型推理，任务下发在stream1
aclmdlDataset *input1;
aclmdlDataset *output1;
aclmdlExecuteAsync(modelId1, input1, output1, stream1);

aclrtCreateStream(&stream2);
// 调用触发任务的接口，例如异步模型推理， 任务下发在stream2
aclmdlDataset *input2;
aclmdlDataset *output2;
aclmdlExecuteAsync(modelId2, input1, output2, stream2);

// 流同步
aclrtSynchronizeStream(stream1);
aclrtSynchronizeStream(stream2);

// 释放资源
aclrtDestroyStream(stream1);
aclrtDestroyStream(stream2);
aclrtDestroyContext(context);
// ....
```

多线程多 Stream

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

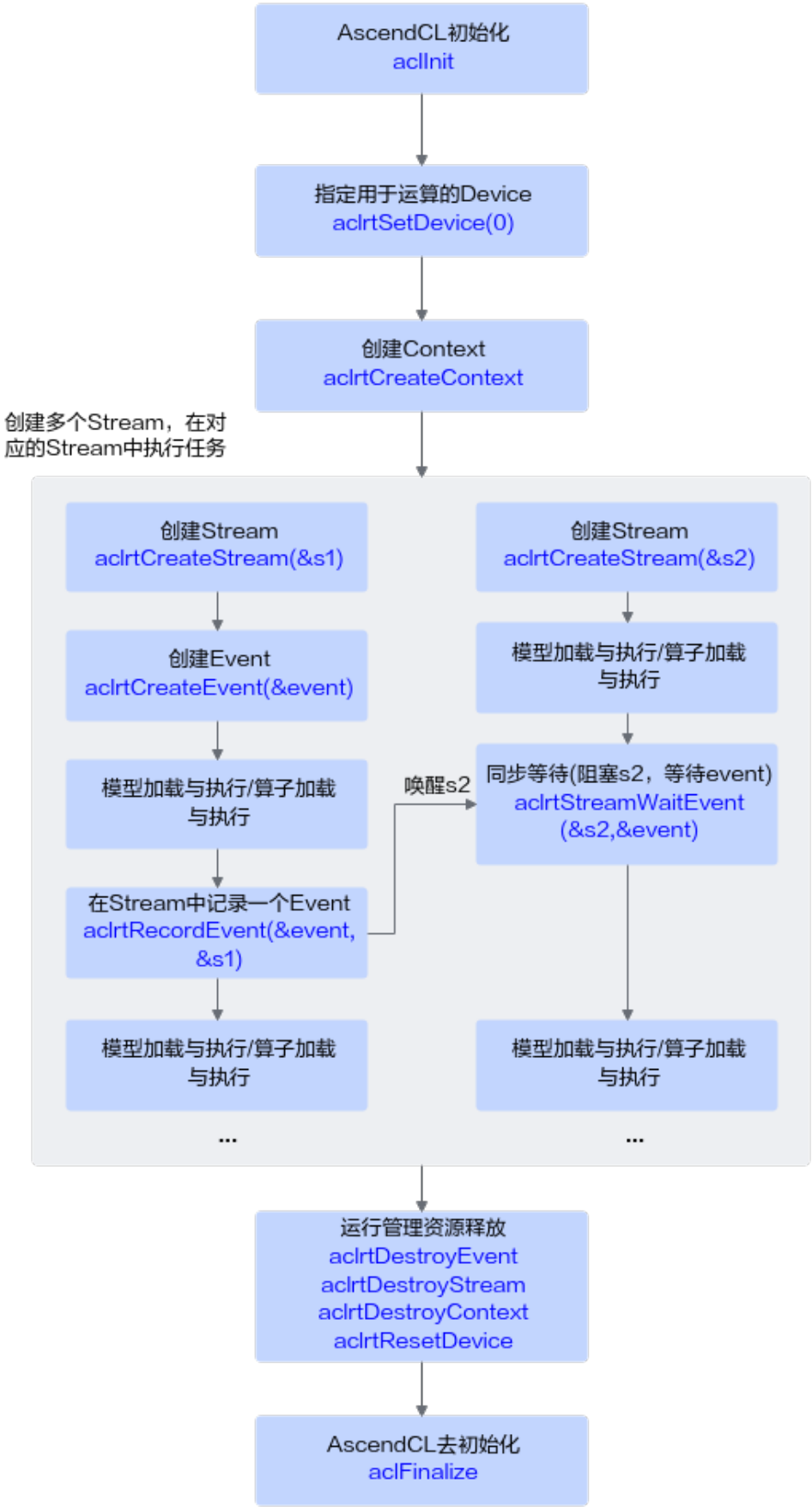
```
#include "acl/acl.h"
// ......
void runThread(aclrtStream stream) {
    int32_t deviceId =0;
    aclrtContext context;

    // 如果只创建了一个Context，线程默认将这个Context作为线程当前的Context；
    // 如果是多个Context，则需要调用aclrtSetCurrentContext接口设置当前线程的Context
    aclrtCreateContext(&context, deviceId);
    aclrtCreateStream(&stream);

    // 调用触发任务的接口
    // ....

    // 释放资源
    aclrtDestroyStream(stream);
    aclrtDestroyContext(context);

}

aclrtStream stream1;
aclrtStream stream2;
// 创建2个线程，每个线程对应一个Stream
```

```
std::thread t1(runThread, stream1);
std::thread t2(runThread, stream2);
// 显式调用join函数确保结束线程
t1.join();
t2.join();
```

多 Stream 时的同步等待流程

开发应用时，如果涉及多Stream之间的任务等待，则应用程序中必须包含相关的代码逻辑，关于该场景的接口调用流程，请依次参见**3.2 AscendCL接口调用流程**以及本节中的说明。

图 7-7 同步等待流程_多 Stream 场景



多Stream之间任务的同步等待可以利用Event实现，调用**aclrtStreamWaitEvent**接口阻塞指定Stream的运行，直到指定的Event完成。需在调用**aclrtStreamWaitEvent**接口前，先调用**aclrtRecordEvent**接口。调用示例请参见**Stream间任务的同步等待示例代码**。

模型加载与执行的流程请参见**4 基础推理应用**。

算子加载与执行的流程请参见**6 单算子调用**。

7.10 同步等待

本节介绍Device、Stream、Event在异步场景下的使用示例及关键接口。

同步机制

AscendCL提供以下几种同步机制：

- **Event的同步等待示例代码**：调用**aclrtSynchronizeEvent**接口，阻塞应用程序运行，等待Event完成。
- **Stream内任务的同步等待示例代码**：调用**aclrtSynchronizeStream**接口，阻塞应用程序运行，直到指定Stream中的所有任务都完成。
- **Stream间任务的同步等待示例代码**：调用**aclrtStreamWaitEvent**接口，阻塞指定Stream的运行，直到指定的Event完成。支持多个Stream等待同一个Event的场景。接口调用流程请参见**多Stream时的同步等待流程**。
- **Device的同步等待示例代码**：调用**aclrtSynchronizeDevice**接口，阻塞应用程序运行，直到正在运算中的Device完成运算。多Device场景下，调用该接口等待的是当前Context对应的Device。接口调用流程请参见**7.2 多Device切换**。

Event 的同步等待示例代码

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
#include "acl/acl.h"
// ......
// 创建一个Event
aclrtEvent event;
aclrtCreateEvent(&event);

// 创建一个Stream
aclrtStream stream;
aclrtCreateStream(&stream);

// stream末尾添加了一个event
aclrtRecordEvent(event, stream);

// 阻塞应用程序运行，等待event发生，也就是stream执行完成
// stream完成后产生event，唤醒执行应用程序的控制流，开始执行程序
aclrtSynchronizeEvent(event);

// 显式销毁资源
aclrtDestroyStream(stream);
aclrtDestroyEvent(event);
// ......
```

Stream 内任务的同步等待示例代码

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
#include "acl/acl.h"
// ......
// 显式创建一个Stream
aclrtStream stream;
aclrtCreateStream(&stream);

// 调用触发任务的接口，传入stream参数
aclrtMemcpyAsync(dstPtr, dstSize, srcPtr, srcSize, ACL_MEMCPY_HOST_TO_DEVICE, stream);
// 调用aclrtSynchronizeStream接口，阻塞应用程序运行，直到指定Stream中的所有任务都完成。
aclrtSynchronizeStream(stream);

// Stream使用结束后，显式销毁Stream
aclrtDestroyStream(stream);
// ......
```

Stream 间任务的同步等待示例代码

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
#include "acl/acl.h"
// ......
// 创建一个Event
aclrtEvent event;
aclrtCreateEvent(&event);

// 创建两个Stream
aclrtStream s1;
aclrtStream s2;
aclrtCreateStream(&s1);
aclrtCreateStream(&s2);

// 在s1末尾添加了一个event
aclrtRecordEvent(event, s1);

// 阻塞s2运行，直到指定event发生，也就是s1执行完成
// s1完成后，唤醒s2，继续执行s2的任务
aclrtStreamWaitEvent(s2, event);

// 显式销毁资源
aclrtDestroyStream(s2);
aclrtDestroyStream(s1);
aclrtDestroyEvent(event);
// ......
```

Device 的同步等待示例代码

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
#include "acl/acl.h"
// ......
// 指定device
aclrtSetDevice(0);

// 创建context
aclrtContext ctx;
aclrtCreateContext(&ctx, 0);

// 创建stream
aclrtStream stream;
```

```
aclrtCreateStream(&stream);

// 阻塞应用程序运行，直到正在运算中的Device完成运算
aclrtSynchronizeDevice();

// 资源销毁
aclrtDestroyStream(stream);
aclrtDestroyContext(ctx);
aclrtResetDevice(0);
```

7.11 AI Core 异常信息获取

基本原理

使用场景举例：执行整网模型推理时（**不支持动态Shape场景**），如果产生AI Core报错，可以调用本接口获取报错算子的描述信息，再做进一步错误排查。

推荐的接口调用顺序如下：

1. 定义并实现异常回调函数fn(aclrtExceptionInfoCallback类型)，回调函数原型请参见**11.19.1 aclrtSetExceptionInfoCallback** 。

   实现回调函数的关键步骤如下：

   a. 在异常回调函数fn内调用**aclrtGetDeviceIdFromExceptionInfo**、**aclrtGetStreamIdFromExceptionInfo**、**aclrtGetTaskIdFromExceptionInfo**接口分别获取Device ID、Stream ID、Task ID。

   b. 在异常回调函数fn内调用**aclmdlCreateAndGetOpDesc**接口获取算子的描述信息。

   c. 在异常回调函数fn内调用**aclGetTensorDescByIndex**接口获取指定算子输入/输出的tensor描述。

   d. 在异常回调函数fn内如下接口获取tensor描述中的数据，进行进一步分析。

      例如，调用**aclGetTensorDescAddress**接口获取tensor数据的内存地址（用户可从该内存地址中获取tensor数据）、调用**aclGetTensorDescType**接口获取tensor描述中的数据类型、调用**aclGetTensorDescFormat**接口获取tensor描述中的Format、调用**aclGetTensorDescNumDims**接口获取tensor描述中的Shape维度个数、调用**aclGetTensorDescDimV2**接口获取Shape中指定维度的大小。

2. 调用**aclrtSetExceptionInfoCallback**接口设置异常回调函数。
3. 执行模型推理。

   如果存在AI Core报错，则触发回调函数fn，获取算子的信息，进行进一步分析。

示例代码

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

本节中的示例重点介绍AI Core异常信息获取的代码逻辑，AscendCL初始化和去初始化请参见**4.3 AscendCL初始化与去初始化**，运行管理资源申请与释放请参见**4.4 运行管理资源申请与释放**，模型加载、准备模型推理的输入/输出数据的接口调用流程、模型执行、模型卸载等请参见**4.6 模型推理**。

```
// 1.AscendCL初始化

// 2.申请运行管理资源，包括设置用于计算的Device、创建Context、创建Stream
```

```
// 3.模型加载，加载成功后，返回标识模型的modelId

// 4.创建aclmdlDataset类型的数据，用于描述模型的输入数据input、输出数据output

// 5.实现异常回调函数
void callback(aclrtExceptionInfo *exceptionInfo)
{
    deviceId = aclrtGetDeviceIdFromExceptionInfo(exceptionInfo);
    streamId = aclrtGetStreamIdFromExceptionInfo(exceptionInfo);
    taskId = aclrtGetTaskIdFromExceptionInfo(exceptionInfo);

    char opName[256];
    aclTensorDesc *inputDesc = nullptr;
    aclTensorDesc *outputDesc = nullptr;
    size_t inputCnt = 0;
    size_t outputCnt = 0;
    // 用户可以将获取的算子信息写入到文件，或者另起线程，当发生异常回调时触发线程处理函数，在线程处理
函数中将算子信息在屏幕上显示
    aclmdlCreateAndGetOpDesc(deviceId, streamId, taskId, opName, 256,  &inputDesc, &inputCnt,
&outputDesc, &outputCnt);
    // 可以调用AscendCL tensor的相关接口，获取算子的相关信息，用户可以根据自己需要调用
    for (size_t i = 0; i < inputCnt; ++i) {
        const aclTensorDesc *desc = aclGetTensorDescByIndex(inputDesc, i);
        aclGetTensorDescAddress(desc);
        aclGetTensorDescFormat(desc);
    }
    for (size_t i = 0; i < outputCnt; ++i) {
        const aclTensorDesc *desc = aclGetTensorDescByIndex(outputDesc, i);
        aclGetTensorDescAddress(desc);
        aclGetTensorDescFormat(desc);
    }
    aclDestroyTensorDesc(inputDesc);
    aclDestroyTensorDesc(outputDesc);
}

// 6.设置异常回调
aclrtSetExceptionInfoCallback(callback);

// 7.执行模型
ret = aclmdlExecute(modelId, input, output);

// 8.处理模型推理结果

// 9.释放描述模型输入/输出信息、内存等资源，卸载模型

// 10.释放运行管理资源

// 11. AscendCL去初始化

// ......
```

7.12 Profiling 性能数据采集

基本原理

该章节下的接口用于Profiling采集性能数据，实现方式支持以下三种：

- **Profiling AscendCL API（通过Profiling AscendCL API采集并落盘性能数据）**

  实现将采集到的Profiling数据写入文件，再使用Profiling工具解析该文件（请参见《CANN 开发工具指南（开放态）》中的“性能分析工具”章节下的“数据解析与导出”），并展示性能分析数据。

  包括以下**两种接口调用方式**：

- **aclprofInit**接口、**aclprofStart**接口、**aclprofStop**接口、**aclprofFinalize**接口配合使用，实现该方式的性能数据采集。该方式可获取AscendCL的接口性能数据、AI Core上算子的执行时间、AI Core性能指标数据等。目前这些接口为进程级控制，表示在进程内任意线程调用该接口，其它线程都会生效。

  一个进程内，可以根据需求多次调用这些接口，基于不同的Profiling采集配置，采集数据。
- 调用**aclInit**接口，在AscendCL初始化阶段，通过*.json 文件传入要采集的Profiling数据。该方式可获取AscendCL的接口性能数据、AI Core上算子的执行时间、AI Core性能指标数据等。

  一个进程内，只能调用一次**aclInit**接口，如果要修改Profiling采集配置，需修改*.json文件中的配置。详细使用说明请参见**aclInit**接口处的说明，不在本章节描述。

- **Profiling AscendCL API for Extension（Profiling AscendCL API扩展接口）**

  当用户需要定位应用程序或上层框架程序的性能瓶颈时，可在Profiling采集进程内（**aclprofStart**接口与**aclprofStop**接口之间）调用Profiling AscendCL API扩展接口（统称为msproftx功能），开启记录应用程序执行期间特定事件发生的时间跨度，并将数据写入Profiling数据文件，再使用Profiling工具解析该文件，并导出展示性能分析数据。

  一个进程内，可以根据需求多次调用这些接口，**接口调用方式如下**：在**aclprofStart**和**aclprofStop**接口之间调用**aclprofCreateStamp**、**aclprofPush**、**aclprofPop**、**aclprofRangeStart**、**aclprofRangeStop**、**aclprofDestroyStamp**接口。该方式可获取应用程序执行期间特定时间发生的事件并记录事件发生的时间跨度。

  Profiling工具解析导出操作请参见《**CANN 开发工具指南（开放态）**》中的“性能分析工具”章节下的“Profiling数据解析”和“Profiling数据导出”。

- **Profiling AscendCL API for Subscription（订阅算子信息的Profiling AscendCL API）**

  实现将采集到的Profiling数据解析后写入管道，由用户读入内存，再由用户调用AscendCL的接口获取性能数据。

  接口调用方式：**aclprofModelSubscribe**接口、aclprofGet*接口、**aclprofModelUnSubscribe**接口配合使用，实现该方式的性能数据采集，当前支持获取网络模型中算子的性能数据，包括算子名称、算子类型名称、算子执行时间等。

Profiling AscendCL API 示例代码

调用接口后，需增加异常处理的分支，示例代码中不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

本节中的示例重点介绍Profiling性能数据采集的代码逻辑，AscendCL初始化和去初始化请参见**4.3 AscendCL初始化与去初始化**，运行管理资源申请与释放请参见**4.4 运行管理资源申请与释放**，模型加载、准备模型推理的输入/输出数据的接口调用流程、模型执行、模型卸载等请参见**4.6 模型推理**。

```
// 1.AscendCL初始化

// 2.申请运行管理资源，包括设置用于计算的Device、创建Context、创建Stream

// 3.profiling初始化
// 设置数据落盘路径
const char *aclProfPath = "...";
aclprofInit(aclProfPath, strlen(aclProfPath));

// 4.进行profiling配置
```

```
uint32_t deviceIdList[1] = {0};
// 创建配置结构体
aclprofConfig *config = aclprofCreateConfig(deviceIdList, 1, ACL_AICORE_ARITHMETIC_UTILIZATION,
    nullptr,ACL_PROF_ACL_API | ACL_PROF_TASK_TIME | ACL_PROF_AICORE_METRICS | ACL_PROF_AICPU |
ACL_PROF_L2CACHE | ACL_PROF_HCCL_TRACE | ACL_PROF_MSPROFTX | ACL_PROF_RUNTIME_API);
const char *memFreq = "15";
ret = aclprofSetConfig(ACL_PROF_SYS_HARDWARE_MEM_FREQ, memFreq, strlen(memFreq));
aclprofStart(config);

// 5.模型加载，加载成功后，返回标识模型的modelId

// 6.创建aclmdlDataset类型的数据，用于描述模型的输入数据input、输出数据output

// 7.执行模型
ret = aclmdlExecute(modelId, input, output);

// 8.处理模型推理结果

// 9.释放描述模型输入/输出信息、内存等资源，卸载模型

// 10.关闭profiling配置, 释放配置资源, 释放profiling组件资源
aclprofStop(config);
aclprofDestroyConfig(config);
aclprofFinalize();

// 11.释放运行管理资源

// 12. AscendCL去初始化
// ......
```

Profiling AscendCL API for Extension 示例代码

调用接口后，需增加异常处理的分支，示例代码中不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

Profiling msproftx接口，请参见如下示例中的加粗部分代码。。

示例一（aclprofMark示例）：

```
//1.AscendCL初始化

//2.申请运行管理资源，包括设置用于计算的Device、创建Context、创建Stream
stamp = aclprofCreateStamp();
aclprofSetStampTraceMessage(stamp, "aclrtCreateStream_mark", strlen("AscendCL_Init_Mark"));
aclprofMark(stamp);    //标记Create Stream事件
aclprofDestroyStamp(stamp);
ret = aclrtCreateStream(&stream_);

//3..Profiling初始化
//设置数据落盘路径
const char *aclProfPath = "...";
aclprofInit(aclProfPath, strlen(aclProfPath));

//4.进行Profiling配置
uint32_t deviceIdList[1] = {0};
//创建配置结构体
aclprofConfig *config = aclprofCreateConfig(deviceIdList, 1, ACL_AICORE_ARITHMETIC_UTILIZATION,
    nullptr,ACL_PROF_ACL_API | ACL_PROF_TASK_TIME);
const char *memFreq = "15";
ret = aclprofSetConfig(ACL_PROF_SYS_HARDWARE_MEM_FREQ, memFreq, strlen(memFreq));
aclprofStart(config);

aclprofStepInfo *stepInfo = aclprofCreateStepInfo();
int ret = aclprofGetStepTimestamp(stepInfo, ACL_STEP_START, stream_);

//5.模型加载，加载成功后，返回标识模型的modelId
stamp = aclprofCreateStamp();
aclprofSetStampTraceMessage(stamp, "model_load_mark", strlen("model_load_mark"));
```

```
aclprofMark(stamp);     //标记模型加载事件
aclprofDestroyStamp(stamp);

//6.创建aclmdlDataset类型的数据，用于描述模型的输入数据input、输出数据output

//7.执行模型
ret = aclmdlExecute(modelId, input, output);

//8.处理模型推理结果

//9.释放描述模型输入/输出信息、内存等资源，卸载模型
int ret = aclprofGetStepTimestamp(stepInfo, ACL_STEP_END, stream_);
aclprofDestroyStepInfo(stepInfo);

//10.关闭Profiling配置, 释放配置资源, 释放Profiling组件资源
aclprofStop(config);
aclprofDestroyConfig(config);
aclprofFinalize();

//11.释放运行管理资源

//12. AscendCL去初始化
//......
```

示例二（aclprofPush/aclprofPop示例，适用于单线程）：

```
//1.AscendCL初始化

//2.申请运行管理资源，包括设置用于计算的Device、创建Context、创建Stream
ret = aclrtCreateStream(&stream_);

//3..Profiling初始化
//设置数据落盘路径
const char *aclProfPath = "...";
aclprofInit(aclProfPath, strlen(aclProfPath));

//4.进行Profiling配置
uint32_t deviceIdList[1] = {0};
//创建配置结构体
aclprofConfig *config = aclprofCreateConfig(deviceIdList, 1, ACL_AICORE_ARITHMETIC_UTILIZATION,
    nullptr,ACL_PROF_ACL_API | ACL_PROF_TASK_TIME);
const char *memFreq = "15";
ret = aclprofSetConfig(ACL_PROF_SYS_HARDWARE_MEM_FREQ, memFreq, strlen(memFreq));
aclprofStart(config);

aclprofStepInfo *stepInfo = aclprofCreateStepInfo();
int ret = aclprofGetStepTimestamp(stepInfo, ACL_STEP_START, stream_);

//5.模型加载，加载成功后，返回标识模型的modelId

//6.创建aclmdlDataset类型的数据，用于描述模型的输入数据input、输出数据output

//7.执行模型（模型仅在单线程执行）
stamp = aclprofCreateStamp();
aclprofSetStampTraceMessage(stamp, "aclmdlExecute_duration", strlen("aclmdlExecute_duration"));
aclprofPush(stamp);
ret = aclmdlExecute(modelId, input, output);
aclprofPop(stamp);
aclprofDestroyStamp(stamp);

//8.处理模型推理结果

//9.释放描述模型输入/输出信息、内存等资源，卸载模型
int ret = aclprofGetStepTimestamp(stepInfo, ACL_STEP_END, stream_);
aclprofDestroyStepInfo(stepInfo);

//10.关闭Profiling配置, 释放配置资源, 释放Profiling组件资源
aclprofStop(config);
aclprofDestroyConfig(config);
aclprofFinalize();
```

```
//11.释放运行管理资源

//12. AscendCL去初始化
//......
```

示例三（aclprofRangeStart/aclprofRangeStop示例，适用于单线程或跨线程）：

```
//1.AscendCL初始化

//2.申请运行管理资源，包括设置用于计算的Device、创建Context、创建Stream
ret = aclrtCreateStream(&stream_);

//3..Profiling初始化
//设置数据落盘路径
const char *aclProfPath = "...";
aclprofInit(aclProfPath, strlen(aclProfPath));

//4.进行Profiling配置
uint32_t deviceIdList[1] = {0};
//创建配置结构体
aclprofConfig *config = aclprofCreateConfig(deviceIdList, 1, ACL_AICORE_ARITHMETIC_UTILIZATION,
    nullptr,ACL_PROF_ACL_API | ACL_PROF_TASK_TIME);
const char *memFreq = "15";
ret = aclprofSetConfig(ACL_PROF_SYS_HARDWARE_MEM_FREQ, memFreq, strlen(memFreq));
aclprofStart(config);

aclprofStepInfo *stepInfo = aclprofCreateStepInfo();
int ret = aclprofGetStepTimestamp(stepInfo, ACL_STEP_START, stream_);

//5.模型加载，加载成功后，返回标识模型的modelId

//6.创建aclmdlDataset类型的数据，用于描述模型的输入数据input、输出数据output

//7.执行模型（模型在跨线程执行）
stamp = aclprofCreateStamp();
aclprofSetStampTraceMessage(stamp, "aclmdlExecute_duration", strlen("aclmdlExecute_duration"));
aclprofRangeStart(stamp, &rangeId);
ret = aclmdlExecute(modelId, input, output);
aclprofRangeStop(rangeId);
aclprofDestroyStamp(stamp);

//8.处理模型推理结果

//9.释放描述模型输入/输出信息、内存等资源，卸载模型
int ret = aclprofGetStepTimestamp(stepInfo, ACL_STEP_END, stream_);
aclprofDestroyStepInfo(stepInfo);

//10.关闭Profiling配置, 释放配置资源, 释放Profiling组件资源
aclprofStop(config);
aclprofDestroyConfig(config);
aclprofFinalize();

//11.释放运行管理资源

//12. AscendCL去初始化
//......
```

Profiling AscendCL API for Subscription 示例代码

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

本节中的示例重点介绍Profiling性能数据采集的代码逻辑，AscendCL初始化和去初始化请参见**4.3 AscendCL初始化与去初始化**，运行管理资源申请与释放请参见**4.4 运行管理资源申请与释放**，模型加载、准备模型推理的输入/输出数据的接口调用流程、模型执行、模型卸载等请参见**4.6 模型推理**。

```
// 1.AscendCL初始化

// 2.申请运行管理资源，包括设置用于计算的Device、创建Context、创建Stream

// 3.模型加载，加载成功后，返回标识模型的modelId

// 4.创建aclmdlDataset类型的数据，用于描述模型的输入数据input、输出数据output

// 5.创建管道（UNIX操作系统下需要引用C++标准库头文件unistd.h），用于读取以及写入模型订阅的数据
int subFd[2];
// 读管道指针指向subFd[0]，写管道指针指向subFd[1]
pipe(subFd);

// 6.创建模型订阅的配置并且进行模型订阅
aclprofSubscribeConfig *config = aclprofCreateSubscribeConfig(1, ACL_AICORE_NONE, &subFd[1]);
// 模型订阅需要传入模型的modelId
aclprofModelSubscribe(modelId, config);

// 7.实现管道读取订阅数据的函数
// 7.1 自定义函数，实现从用户内存中读取订阅数据的函数
void getModelInfo(void *data, uint32_t len) {
    uint32_t opNumber = 0;
    uint32_t dataLen = 0;
    // 通过AscendCL接口读取算子信息个数
    aclprofGetOpNum(data, len, &opNumber);
    // 遍历用户内存的算子信息
    for (int32_t i = 0; i < opNumber; i++){
        // 获取算子的模型id
        uint32_t modelId = aclprofGetModelId(data,len, i);
        // 获取算子的类型名称长度
        size_t opTypeLen = 0;
        aclprofGetOpTypeLen(data, len, i, &opTypeLen);
        // 获取算子的类型名称
        char opType[opTypeLen];
        aclprofGetOpType(data, len, i, opType, opTypeLen);
        // 获取算子的详细名称长度
        size_t opNameLen = 0;
        aclprofGetOpNameLen(data, len, i, &opNameLen);
        // 获取算子的详细名称
        char opName[opNameLen];
        aclprofGetOpName(data, len, i, opName, opNameLen);
        // 获取算子的执行开始时间
        uint64_t opStart = aclprofGetOpStart(data, len, i);
        // 获取算子的执行结束时间
        uint64_t opEnd = aclprofGetOpEnd(data, len, i);
        uint64_t opDuration = aclprofGetOpDuration(data, len, i);
    }
}

// 7.2 自定义函数，实现从管道中读取数据到用户内存的函数
void *profDataRead(void *fd) {
    // 设置每次从管道中读取的算子信息个数
    uint64_t N = 10;
    // 获取单位算子信息的大小（Byte）
    uint64_t bufferSize = 0;
    aclprofGetOpDescSize(&bufferSize);
    // 计算存储算子信息的内存的大小，并且申请内存
    uint64_t readbufLen = bufferSize * N;
    char *readbuf = new char[readbufLen];
    // 从管道中读取数据到申请的内存中，读取到的实际数据大小dataLen可能小于bufferSize * N，如果管道中没有数据，默认会阻塞直到读取到数据为止
    auto dataLen = read(*(int*)fd, readbuf, readbufLen);
    // 读取数据到readbuf成功
    while (dataLen > 0) {
        // 调用5.1实现的函数解析内存中的数据
        getModelInfo(readbuf, dataLen);
        memset(readbuf, 0, bufferSize);
        dataLen = read(*(int*)fd, readbuf, readbufLen);
    }
```

```
    delete []readbuf;
}

// 8. 启动线程读取管道数据并解析
pthread_t subTid = 0;
pthread_create(&subTid, NULL, profDataRead, &subFd[0]);

// 9.执行模型
ret = aclmdlExecute(modelId, input, output);

// 10.处理模型推理结果

// 11.释放描述模型输入/输出信息、内存等资源，卸载模型

// 12.取消订阅，释放订阅相关资源
aclprofModelUnSubscribe(modelId);
pthread_join(subTid, NULL);
// 关闭读管道指针
close(subFd[0]);
// 释放config指针
aclprofDestroySubscribeConfig(config);

// 13.释放运行管理资源

// 14. AscendCL去初始化
// ......
```

7.13 溢出算子数据采集及分析

前提条件

使用ATC工具转换模型时，需在转换命令中增加--status_check参数，并将参数值设置为1，表示在编译算子时添加溢出检测逻辑。

关于ATC工具及其参数的详细说明，请参见《**ATC工具使用指南（开放态）**》。

采集溢出算子信息

在调用**aclInit**接口初始化AscendCL时，在json配置文件中增加溢出算子Dump配置。

json配置文件中的示例内容如下，示例中的dump_path以相对路径为例：

```
{
    "dump":{
        "dump_path":"output",
        "dump_debug":"on"
    }
}
```

当dump_path配置为相对路径时，您可以在“应用可执行文件的目录/{dump_path}”下查看导出的数据文件，针对每个溢出算子，会导出两个数据文件：

- 溢出算子的dump文件：命名规则如{op_type}.{op_name}.{taskid}.{stream_id}.{timestamp}，如果op_type、op_name出现了“.”、“/”、“\”、空格时，会转换为下划线表示。

  用户可通过该信息知道具体出现溢出错误的算子，并通过解析溢出算子的dump文件获取该算子的输入和输出信息。

- 算子溢出数据文件：命名规则如OpDebug.Node_Opdebug.{taskid}.{stream_id}.{timestamp}，其中taskid不是溢出算子的taskid，用户不需要关注taskid的实际含义。

用户可通过解析算子溢出数据文件获取溢出相关信息，包括溢出算子所在的模型、AICore的status寄存器状态等。

解析溢出算子的 dump 文件

步骤1 请根据实际情况，将{op_type}.{op_name}.{taskid}.{stream_id}.{timestamp}上传到安装有Toolkit软件包的环境。

步骤2 进入解析脚本所在目录，例如Toolkit软件包安装目录为：/home/HwHiAiUser/Ascend/ascend-toolkit/latest。

```
cd /home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/operator_cmp/compare
```

步骤3 执行msaccucmp.py脚本，转换dump文件为numpy文件。举例：

```
python3 msaccucmp.py convert -d /home/HwHiAiUser/dump -out /home/HwHiAiUser/dumptonumpy -v 2
```

说明

-d参数支持传入单个文件，对单个dump文件进行转换，也支持传入目录，对整个path下所有的dump文件进行转换。

步骤4 调用Python，转换numpy文件为txt文件。举例：

```
$ python3
>>> import numpy as np
>>> a = np.load("/home/HwHiAiUser/dumptonumpy/Pooling.pool1.1147.1589195081588018.output.0.npy")
>>> b = a.flatten()
>>> np.savetxt("/home/HwHiAiUser/dumptonumpy/Pooling.pool1.1147.1589195081588018.output.0.txt", b)
```

转换为.txt格式文件后，维度信息、Dtype均不存在。详细的使用方法请参考numpy官网介绍。

----结束

解析算子溢出数据文件

由于生成的溢出数据是二进制格式，可读性较差，需要通过工具将bin文件解析为用户可读性好的json文件。

步骤1 请根据实际情况，将溢出数据文件OpDebug.Node_Opdebug.{taskid}.{timestamp}上传到安装有Toolkit软件包的环境。

步骤2 进入解析脚本所在路径，例如Toolkit软件包安装目录为：/home/HwHiAiUser/Ascend/ascend-toolkit/latest。

```
cd /home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/operator_cmp/compare
```

步骤3 执行解析命令，例如：

python3 msaccucmp.py convert -d /home/HwHiAiUser/opdebug/Opdebug.Node_OpDebug.59.1597922031178434 -out /home/HwHiAiUser/result

关键参数：

- -d：溢出数据文件所在目录，包括文件名。
- -out：解析结果待存储目录，如果不指定，默认生成在当前目录下。

步骤4 解析结果文件内容如下所示。

```
{
    "DHA Atomic Add": {
        "model_id": 0,
        "stream_id": 0,
        "task_id": 0,
        "task_type": 0,
        "pc_start": "0x0",
        "para_base": "0x0",
        "status": 0
    },
    "L2 Atomic Add": {
        "model_id": 0,
        "stream_id": 0,
        "task_id": 0,
        "task_type": 0,
        "pc_start": "0x0",
        "para_base": "0x0",
        "status": 0
    },
    "AI Core": {
        "model_id": 514,
        "stream_id": 563,
        "task_id": 57,
        "task_type": 0,
        "pc_start": "0x1008005b0000",
        "para_base": "0x100800297000",
        "kernel_code": "0x1008005ae000",
        "block_idx": 1,
        "status": 32
    }
}
```

参数解释：

- model_id：标识溢出算子所在的模型id。
- stream_id：标识溢出算子所在的streamid。
- task_id：标识溢出算子的taskid。
- task_type：标识溢出算子的task类型。
- pc_start：标识溢出算子的代码程序的内存起始地址。
- para_base：标识溢出算子的参数的内存起始地址。
- kernel_code：标识溢出算子的代码程序的内存起始地址，和pc_start相同。
- block_idx：标识溢出算子的blockid参数。
- status：AICore的status寄存器状态，用户可以从status值分析得到具体溢出错误。status为10进制表示，需要转换成16进制，然后定位到具体错误。

  例如：status为272，转换成16进制为0x00000110，则可以判定出可能原因为0x00000010+0x00000100。

  - 0x00000008: 符号整数最小负数NEG符号位取反溢出

- 0x00000010: 整数加法、减法、乘法或乘加操作计算有溢出
- 0x00000020: 浮点计算有溢出
- 0x00000080: 浮点数转无符号数的输入是负数
- 0x00000100: FP32转FP16或32位符号整数转FP16中出现溢出
- 0x00000400: CUBE累加出现溢出

----结束

7.14 特征向量检索

须知

Atlas 200/300/500 推理产品上，不支持该功能。

基本原理

该部分主要实现了对特征检索的功能验证，生成随机底库，随机生成特征数据进行特征检索（当前支持1:N、M：N两种检索模式，下文的示例代码以1：N为例）。大致可分为初始化、添加特征到底库、底库搜索、精准修改或删除底库特征、去初始化几个主要步骤，具体接口调用方式如下：

- **初始化**：调用aclInit接口进行AscendCL初始化、运行管理资源申请，调用**aclfvCreateInitPara**接口创建aclfvInitPara类型的数据来指定特征向量检索的初始化参数。
- **添加特征到底库**：主要调用**aclfvCreateFeatureInfo**接口创建aclfvFeatureInfo类型数据来表示创建特征的描述信息，然后调用**aclfvRepoAdd**添加底库。
- **底库搜索**：调用**aclfvSearch**接口来实现检索。
- **精准修改或删除底库特征**：调用**aclfvDel**和**aclfvModify**接口来实现删除或修改底库中某个特征。下文的代码以删除底库特征为例。
- **去初始化**：主要包括释放运行管理资源、调用**aclfvDestroyInitPara**接口销毁aclfvInitPara类型的数据、调用**aclfvRelease**接口特征检索模块去初始化，释放内存空间。

示例代码

本节中的示例重点介绍特征向量检索的代码逻辑，AscendCL初始化和去初始化请参见**4.3 AscendCL初始化与去初始化**，运行管理资源申请与释放请参见**4.4 运行管理资源申请与释放**。

示例代码如下，可以从**Link**中查看完整样例代码。

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
// 1.AscendCL初始化

// 2.申请运行管理资源，包括设置用于计算的Device、创建Context
// 设置默认的运行模式为HOST
aclrtRunMode runMode = ACL_HOST;

// 3.初始化
```

```
// 3.1 初始化特征检索模块，此处以底库特征数100000为例
size_t fsNum = 100000;
fvInitPara = aclfvCreateInitPara(fsNum);

// 3.2 指定特征向量检索的初始化参数
ret = aclfvInit(fvInitPara);

// 4.添加底库和特征向量
// 4.1 增加第一个特征，创建特征描述信息时，偏移量offset参数值为0
uint32_t offset = 0;
uint32_t featureCount = 1000;
uint32_t featureLen = 36;

// 此处的自定义函数BaseShortFeaAlloc用于生成特征随机数据，由用户自行实现
void *featureData = BaseShortFeaAlloc(1000, static_cast<size_t>(featureCount), 0);
std::shared_ptr<void> feaBufPtr(featureData, [](void *p){(void)aclrtFreeHost(p);});
void *inputData = featureData;
std::shared_ptr<void> inputDataPtr = nullptr;

// 如果运行模式为ACL_HOST，则需要申请内存，再通过aclrtMemcpy接口将Host的随机特征数据传输到
Device；否则直接将随机特征数据读入Device内存
if (aclrtGetRunMode(&runMode) == ACL_HOST) {
    // 为inputData申请内存
    ret = aclrtMalloc(&inputData, featureLen * featureCount,
                      ACL_MEM_MALLOC_NORMAL_ONLY);
    // 将随机特征数据读入到Device内存中
    inputDataPtr.reset(inputData, [](void *p) {(void)aclrtFree(p);});
    // 将featureData从Host侧拷贝到Device侧
    ret = aclrtMemcpy(inputData,
                      featureLen * featureCount,
                      featureData,
                      featureLen * featureCount,
                      ACL_MEMCPY_HOST_TO_DEVICE);
}

// 创建特征描述信息，inputData表示前一步的特征随机数据
auto featureInfo = aclfvCreateFeatureInfo(id0, id1, offset, featureLen, featureCount,
                      reinterpret_cast<uint8_t *>(inputData), featureLen * featureCount);

// 添加底库并向底库中添加特征，featureInfo表示前一步的特征描述信息
aclError ret = aclfvRepoAdd(SEARCH_1_N, featureInfo);

// 销毁aclfvFeatureInfo特征描述信息
aclfvDestroyFeatureInfo(featureInfo);

// 4.2增加第二个特征，创建特征描述信息时，偏移值offset需要与库中已添加特征个数一致，并精确删除或修改
底库中的某个特征
offset += featureCount;

// 增加特征到底库的步骤，参考4.1中的代码
// ....

uint8_t featureData[36];
for (size_t i = 0; i < 36; i++) {
    featureData[i] = static_cast<uint8_t>(i);
}

// 创建内存并传输特征数据
void *inputData = nullptr;
aclrtMalloc(&inputData, 36, ACL_MEM_MALLOC_NORMAL_ONLY);
std::shared_ptr<void> inputDataPtr(inputData, [](void *p){(void)aclrtFree(p);});
aclrtMemcpyKind kind = ACL_MEMCPY_DEVICE_TO_DEVICE;

// 如果运行模式是ACL_HOST,将特征数据拷贝到Device侧，否则无需拷贝，其中dataLen为featureData指针申请
的内存长度
if (aclrtGetRunMode(&runMode) == ACL_HOST) {
    kind = ACL_MEMCPY_HOST_TO_DEVICE;
}
aclrtMemcpy(inputData, 36, featureData, dataLen, kind);
```

```
// 创建特征描述信息
uint32_t id0 = 0;
uint32_t id1 = 0;
auto featureInfo1 = aclfvCreateFeatureInfo(id0, id1, offset, 36, 1,
                                    reinterpret_cast<uint8_t *>(inputData), 36);
std::shared_ptr<aclfvFeatureInfo> featureInfoPtr(featureInfo1,
                                    [](aclfvFeatureInfo *p){(void)aclfvDestroyFeatureInfo(p);});

// 删除1个特征
aclfvDel(featureInfo1);

// 4.3 增加特征到其它底库,其中一级底库为1，二级底库为1
id0 = 1;
id1 = 1;
offset = 0;

// 增加特征到底库步骤，参考4.1中的代码
// ....

// 5 底库检索（以1： N检索为例），主要包括特征检索预处理，特征1:N检索，特征检索结果处理三个部分
// 5.1 特征检索预处理，对于1:N来说, queryCnt必须为1
uint32_t queryCnt = 1;
uint32_t topK = 5;
uint32_t dataLen = queryCnt * topK * sizeof(uint32_t);
uint32_t resultNumDataLen = queryCnt * sizeof(uint32_t);
const uint32_t tableLen = 32 * 1024;
uint32_t tableDataLen = queryCnt * tableLen;

// 生成数据表，用户通过数据表进行检索比对，此处的自定义函数AdcTabInit用于初始化特征检索输入Adc表，由
用户自行实现
uint8_t *tableDataTmp = (uint8_t *)AdcTabInit(1000, queryCnt * 1024);
std::shared_ptr<void> tableDataTmpPtr(tableDataTmp,[](void *p){(void)aclrtFreeHost(p);});

// 为数据表分配内存，tableDataDev用于创建检索输入表信息
void *devPtr = nullptr;
aclrtMalloc(&devPtr, tableDataLen, ACL_MEM_MALLOC_NORMAL_ONLY);
tableDataDev.reset(devPtr, [](void *p) {(void)aclrtFree(p);});

// 拷贝表数据到Device侧
uint8_t *devPtrTmp = reinterpret_cast<uint8_t *>(devPtr);
for (uint32_t i = 0; i < queryCnt; ++i) {
    for (uint32_t j = 0; j < 32; ++j) {
        uint8_t *dst = devPtrTmp + i * 32 * 1024 + j * 1024;
        uint8_t *src = tableDataTmp + i * 1024;
        aclrtMemcpy(dst, 1024, src, 1024, ACL_MEMCPY_HOST_TO_DEVICE);
    }
}

// 为检索结果resultNumDev,id0Dev,id1Dev,resultOffsetDev,resultDistanceDev分配内存
aclrtMalloc(&devPtr, resultNumDataLen, ACL_MEM_MALLOC_NORMAL_ONLY);
resultNumDev.reset(devPtr, [](void *p) {(void)aclrtFree(p);});
aclrtMalloc(&devPtr, dataLen, ACL_MEM_MALLOC_NORMAL_ONLY);
id0Dev.reset(devPtr, [](void *p) {(void)aclrtFree(p);});
aclrtMalloc(&devPtr, dataLen, ACL_MEM_MALLOC_NORMAL_ONLY);
id1Dev.reset(devPtr, [](void *p) {(void)aclrtFree(p);});
aclrtMalloc(&devPtr, dataLen, ACL_MEM_MALLOC_NORMAL_ONLY);
resultOffsetDev.reset(devPtr, [](void *p) {(void)aclrtFree(p);});
aclrtMalloc(&devPtr, dataLen, ACL_MEM_MALLOC_NORMAL_ONLY);
resultDistanceDev.reset(devPtr, [](void *p) {(void)aclrtFree(p);});

// 创建检索输入表信息，结果用于创建检索任务输入信息
aclfvQueryTable *searchQueryTable = aclfvCreateQueryTable(queryCnt, tableLen, reinterpret_cast<uint8_t
*>
                                (tableDataDev.get()), tableDataLen);
searchQueryTable.reset(searchQueryTable, [](aclfvQueryTable *p){(void)aclfvDestroyQueryTable(p);});

// 创建特征库范围参数，结果用于创建检索任务输入信息
aclfvRepoRange *searchRange = aclfvCreateRepoRange(0, 1023, 0, 1023);
```

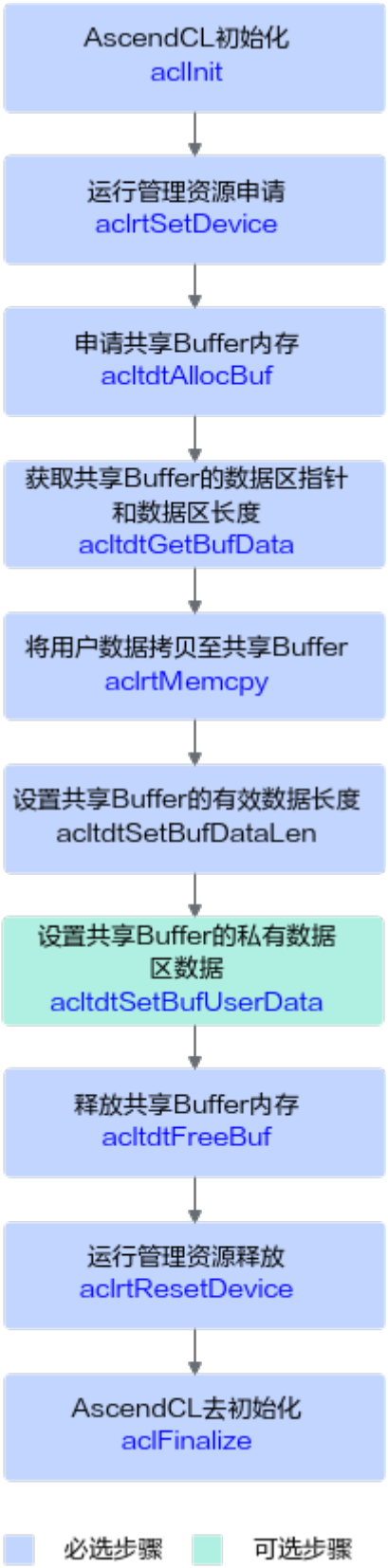
```
searchRange.reset(searchRange, [](aclfvRepoRange *p){(void)aclfvDestroyRepoRange(p);});

// 创建检索任务输入信息，结果用于特征1:N检索
aclfvSearchInput *searchInput = aclfvCreateSearchInput(searchQueryTable, searchRange, topK);
searchInput.reset(searchInput, [](aclfvSearchInput *p){(void)aclfvDestroySearchInput(p);});

// 创建检索结果信息，结果用于特征1:N检索
aclfvSearchResult *searchResult = aclfvCreateSearchResult(queryCnt,
                                  reinterpret_cast<uint32_t *>(resultNumDev.get()),
                                  resultNumDataLen,
                                  reinterpret_cast<uint32_t *>(id0Dev.get()),
                                  reinterpret_cast<uint32_t *>(id1Dev.get()),
                                  reinterpret_cast<uint32_t *>(resultOffsetDev.get()),
                                  reinterpret_cast<float *>(resultDistanceDev.get()),
                                  dataLen);
searchResult.reset(searchResult, [](aclfvSearchResult *p){(void)aclfvDestroySearchResult(p);});

// 5.2 特征1:N检索
aclfvSearch(SEARCH_1_N, searchInput.get(), searchResult.get());

// 5.3 特征检索结果处理
// 获取检索结果
uint32_t dataLen = queryCnt * topK * sizeof(uint32_t);
uint32_t *id0 = (uint32_t *)id0Dev.get();
uint32_t *id1 = (uint32_t *)id1Dev.get();
uint32_t *resultOffset= (uint32_t *)resultOffsetDev.get();
float *resultDistance = (float *)resultDistanceDev.get();

// 如果运行模式为ACL_HOST，则需要通过aclrtMemcpy接口将Device的检索结果回传到Host侧；否则无需回传
if (aclrtGetRunMode(&runMode) == ACL_HOST) {
        // 从Device侧拷贝数据到Host侧
        id0 = (uint32_t *)malloc(dataLen);
        id0Ptr.reset(id0);
        id1 = (uint32_t *)malloc(dataLen);
        id1Ptr.reset(id1);
        resultOffset = (uint32_t *)malloc(dataLen);
        resultOffsetPtr.reset(resultOffset);
        resultDistance = (float *)malloc(dataLen);
        resultDistancePtr.reset(resultDistance);
        aclrtMemcpy(id0, dataLen, id0Dev.get(), dataLen, ACL_MEMCPY_DEVICE_TO_HOST);
        aclrtMemcpy(id1, dataLen, id0Dev.get(), dataLen, ACL_MEMCPY_DEVICE_TO_HOST);
        aclrtMemcpy(resultOffset, dataLen, resultOffsetDev.get(), dataLen, ACL_MEMCPY_DEVICE_TO_HOST);
        aclrtMemcpy(resultDistance, dataLen, resultDistanceDev.get(), dataLen,
ACL_MEMCPY_DEVICE_TO_HOST);
}

// 展示底库中的数据
for (uint32_t i = 0; i < queryCnt; i++) {
     for (uint32_t j = 0; j < topK; ++j) {
          uint32_t i0 = id0[i * topK + j];
          uint32_t i1 = id1[i * topK + j];
          uint32_t offset = resultOffset[i * topK + j];
          float distance = resultDistance[i * topK + j];
    }
}

// 6. 删除底库和数据
// 创建特征库范围并删除指定范围内的底库
uint32_t id0Min = 0;
uint32_t id0Max = 1023;
uint32_t id1Min = 0;
uint32_t id1Max = 1023;
aclfvRepoRange *repoRange = aclfvCreateRepoRange(id0Min, id0Max, id1Min, id1Max);
aclfvRepoDel(SEARCH_1_N, repoRange)
// 销毁aclfvInitPara类型的数据
aclfvDestroyInitPara(fvInitPara);

// 7. 释放运行管理资源
```

```
// 8. AscendCL去初始化
// ......
```

7.15 共享 Buffer 管理

本特性提供了跨进程共享Buffer管理能力，可配合**共享队列**一起使用。共享Buffer的共享范围由共享队列的授权确定，共享范围包括“一主多从”的这些进程。

接口调用流程



关键接口的调用流程如上图所示，流程说明如下：

1. **AscendCL初始化。**

   调用**aclInit**接口实现初始化AscendCL。

2. **运行管理资源申请。**

   具体流程，请参见**4.4 运行管理资源申请与释放**。

3. **申请共享Buffer内存。**

   调用**acltdtAllocBuf**接口申请共享Buffer内存，此处需根据实际情况选择内存类型。

4. **向共享Buffer中填充有效数据。**

   需先调用**acltdtGetBufData**接口获取共享Buffer的数据区指针和数据区长度，接着调用**aclrtMemcpy**接口把用户数据复制到共享Buffer中，最后可调用**acltdtSetBufDataLen**接口设置有效数据长度，以便在多进程场景下调用**acltdtGetBufDataLen**接口获取有效数据长度，再根据有效数据长度获取共享Buffer中的数据。

5. **设置共享Buffer的私有数据区数据。**

   调用**acltdtSetBufUserData**接口设置共享Buffer的私有数据区数据，以便在多进程场景下调用**acltdtGetBufUserData**接口获取共享Buffer的私有数据区数据。

6. **释放共享Buffer内存。**

   调用**acltdtFreeBuf**接口来释放共享Buffer。

7. **运行管理资源释放。**

   接口调用流程，请参见**4.4 运行管理资源申请与释放**。

8. **AscendCL去初始化。**

   调用**aclFinalize**接口实现AscendCL去初始化。

示例代码

调用接口后，需增加异常处理的分支，并记录报错日志、提示日志，此处不一一列举。以下是关键步骤的代码示例，不可以直接拷贝编译运行，仅供参考。

```
#include "acl/acl.h"

// 1. AscendCL初始化
// 此处的..表示相对路径，相对可执行文件所在的目录
// 例如，编译出来的可执行文件存放在out目录下，此处的..就表示out目录的上一级目录
const char *aclConfigPath = "../src/acl.json";
aclError ret = aclInit(aclConfigPath);

// 2. 运行管理资源申请，指定计算设备，此处以deviceId = 0为例
ret = aclrtSetDevice(0);

// 3. 申请mbuf内存并进行内存管理，此处以申请DVPP内存、内存大小1024U为例
size_t size = 1024U;
acltdtBuf buf;
ret = acltdtAllocBuf(size, ACL_TDT_DVPP_MEM, &buf);

// 4. 向共享Buffer中填充有效数据
// 4.1 获取共享Buffer的数据区指针和数据区长度
void *dataPtr = nullptr;
size_t dataSize = 0U;
ret = acltdtGetBufData(buf, &dataPtr, &dataSize);

// 4.2 把用户数据拷贝到共享Buffer中
size_t len = 512U; // 用户数据长度
void *ptr = new (std::nothrow) char_t[len];  // 用户申请自己的内存
// 用户对自己申请的数据进行处理
//……
```

```
ret = aclrtMemcpy(dataPtr, size, ptr, len, ACL_MEMCPY_HOST_TO_DEVICE);
// 5. 释放内存
delete[] ptr;
ptr = nullptr;
delete[] newPtr;
newPtr = nullptr;
ret = acltdtFreeBuf(buf);

// 6. 运行管理资源释放
ret = aclrtResetDevice(0);

// 7. AscendCL去初始化
ret = aclFinalize();
```

8 应用调试

完成程序代码编写后，可按照本节中的指导编译程序、执行应用。

编译及运行应用

1. **编译代码**。

   可参考**10.7 基于Caffe ResNet-50网络实现图片分类（同步推理）**中的说明，主要步骤如下：

   a. 创建编译脚本。

      您可以从该**10.7 基于Caffe ResNet-50网络实现图片分类（同步推理）**样例中获取编译脚本CMakeLists.txt，在该编译脚本的基础上修改如下参数。

      - include_directories：添加头文件所在的目录。依赖的头文件请参见**调用接口依赖的头文件和库文件说明**。

        示例如下：

        ```
        include_directories(
            directoryPath1
            directoryPath2
        )
        ```

      - link_directories：添加库文件所在的目录。

        示例如下：

        ```
        link_directories(
            directoryPath3
            directoryPath4
        )
        ```

      - add_executable：修改可执行文件的名称（例如：*main*）、添加*.cpp文件所在的目录。

        示例如下：

        ```
        add_executable(main
            directoryPath5
            directoryPath6)
        ```

      - target_link_libraries：修改可执行文件的名称（与add_executable中设置的名称保持一致）、添加可执行文件依赖的库文件。依赖的库文件与接口所在的头文件有关，请参见**调用接口依赖的头文件和库文件说明**。

        示例如下：

        ```
        target_link_libraries(main
            ascendcl
        ```

```
    libName1
    libName2)
```

说明

编译基于AscendCL接口的代码逻辑时，请按照include的头文件依赖对应的库文件，如果引用多余的so文件（例如libascendcl.a），可能导致版本功能异常或后续版本升级时存在兼容性问题。

- 编译选项，修改可执行文件的名称（与与add_executable中设置的名称保持一致），以及可执行文件的安装目录。

  示例如下，表示*main*安装在${CMAKE_INSTALL_PREFIX}/out目录下，${CMAKE_INSTALL_PREFIX}变量定义的路径是相对路径，相对cmake命令执行的路径：

```
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY  "../../../out")

install(TARGETS main DESTINATION ${CMAKE_RUNTIME_OUTPUT_DIRECTORY})
```

b. **设置环境变量**。

如果用户是自行开发的编译脚本，未使用DDK_PATH、NPU_HOST_LIB环境变量，则无需设置该环境变量，用户根据编辑脚本逻辑自行增加相关配置即可。

如果用户是基于**10.7 基于Caffe ResNet-50网络实现图片分类（同步推理）**样例中的CMakeLists.txt编译脚本进行修改时，则需设置DDK_PATH、NPU_HOST_LIB环境变量。

DDK_PATH、NPU_HOST_LIB环境变量分别用于指向AscendCL头文件目录、库文件目录。如下为设置环境变量的示例，${INSTALL_DIR}表示CANN软件安装目录，例如，$HOME/Ascend/ascend-toolkit/latest，*arch*表示操作系统架构，*{os}*表示操作系统。

```
export DDK_PATH=${INSTALL_DIR}
export NPU_HOST_LIB=${INSTALL_DIR}/{arch-os}/devlib
```

须知

使用“${INSTALL_DIR}/*{arch-os}*/devlib”目录下的*.so库，是为了编译基于AscendCL接口的代码逻辑时，不依赖其它组件（例如Driver）的任何*.so库。因此在使用cmake编译命令时，请务必将DCMAKE_SKIP_RPATH设置为TRUE，代表不会将rpath路径（即“${INSTALL_DIR}/{arch-os}/devlib”）添加到编译生成的可执行文件中去。

编译通过后，运行应用时，通过配置环境变量，应用会链接到运行环境上“lib64”目录下的*.so库，运行时会自动链接到依赖其它组件的*.so库。

c. 执行cmake命令编译代码。

示例命令如下，其中，“../../../src”表示CMakeLists.txt文件所在的目录，请根据实际目录层级修改；通过-DCMAKE_CXX_COMPILER参数指定的编译器，请根据实际版本要求修改。

- 当开发环境与运行环境操作系统架构相同时，执行如下命令编译：

```
cmake ../../../src -DCMAKE_CXX_COMPILER=g++ -DCMAKE_SKIP_RPATH=TRUE
```

- 当开发环境与运行环境操作系统架构不同时，执行以下命令进行交叉编译：

  例如，当开发环境为X86架构，运行环境为AArch64架构时，执行以下命令进行交叉编译。

```
cmake ../../../src -DCMAKE_CXX_COMPILER=aarch64-linux-gnu-g++ -
DCMAKE_SKIP_RPATH=TRUE
```

说明

关于cmake参数的详细介绍，请参见**https://cmake.org/cmake/help/latest/guide/tutorial/index.html**，选择对应的版本后再查看参数。

d. 执行make命令生成可执行文件。

```
make
```

2. **运行可执行文件**。

将AscendCL初始化配置文件（acl.json）所在的目录、可执行文件所在的目录、测试图片所在的目录、*.om文件所在的目录都上传到运行环境的同一个目录下，再登录到运行环境，切换到可执行文件所在的目录，运行可执行文件，示例命令如下：

```
./main
```

如果在**AscendCL初始化**阶段，在**aclInit**接口中传入空指针，则无需将AscendCL初始化配置文件（acl.json）所在的目录上传到运行环境。

问题定位

运行应用时如果出错，您可以参见《**日志参考（开放态）**》获取日志文件，以便查看日志文件中详细报错。根据报错初步定位后：

- 如果是接口约束导致接口调用逻辑不对，需查看总体的**使用约束**以及各接口本身的约束，再调整接口调用逻辑。
- 如果是算子在AI Core上运行报错，需要进一步定位算子报错的原因，调用AscendCL提供的接口，获取出错算子的描述信息，用于进一步分析时使用，可参见**7.11 AI Core异常信息获取**，查看原理及调用示例。
- 对于Atlas 200/300/500 推理产品，典型的案例及其解决方法请参见《**故障处理（开放态）**》。

9 精度/性能优化

调优简介
模型推理精度提升建议
模型推理性能调优建议
DVPP数据处理高性能编程建议

9.1 调优简介

本章重点介绍推理应用的精度、性能调优，由于是调优，因此在调优前，请确保已经完成了整网推理功能调测，功能不阻塞，只是推理精度错误、推理精度与标杆数据存在少量差距、模型推理性能不符合预期或待提升等问题。

- **应用的精度问题**可能由于推理功能与其它功能之间的串接问题、整网中算子本身的精度问题等，可参考本章中的建议排查功能串接时的接口参数配置问题、借助工具获取详细数据定位分析问题。
- **应用的性能问题**可能由于模型在昇腾AI处理器上的算子适配或数据读写问题、DVPP接口使用问题等，可参考本章中的建议排查接口使用问题、借助工具优化模型、借助工具获取详细数据定位分析问题。

9.2 模型推理精度提升建议

9.2.1 简介

本文介绍整网推理场景下的精度调优流程、相关配置及典型案例等。由于是调优，因此在调优前，请确保已经完成了整网推理功能调测，功能不阻塞，只是推理精度错误，或推理精度与标杆数据存在少量差距。

在整网推理时，可能由于以下原因导致推理精度错误或者推理精度不达标：

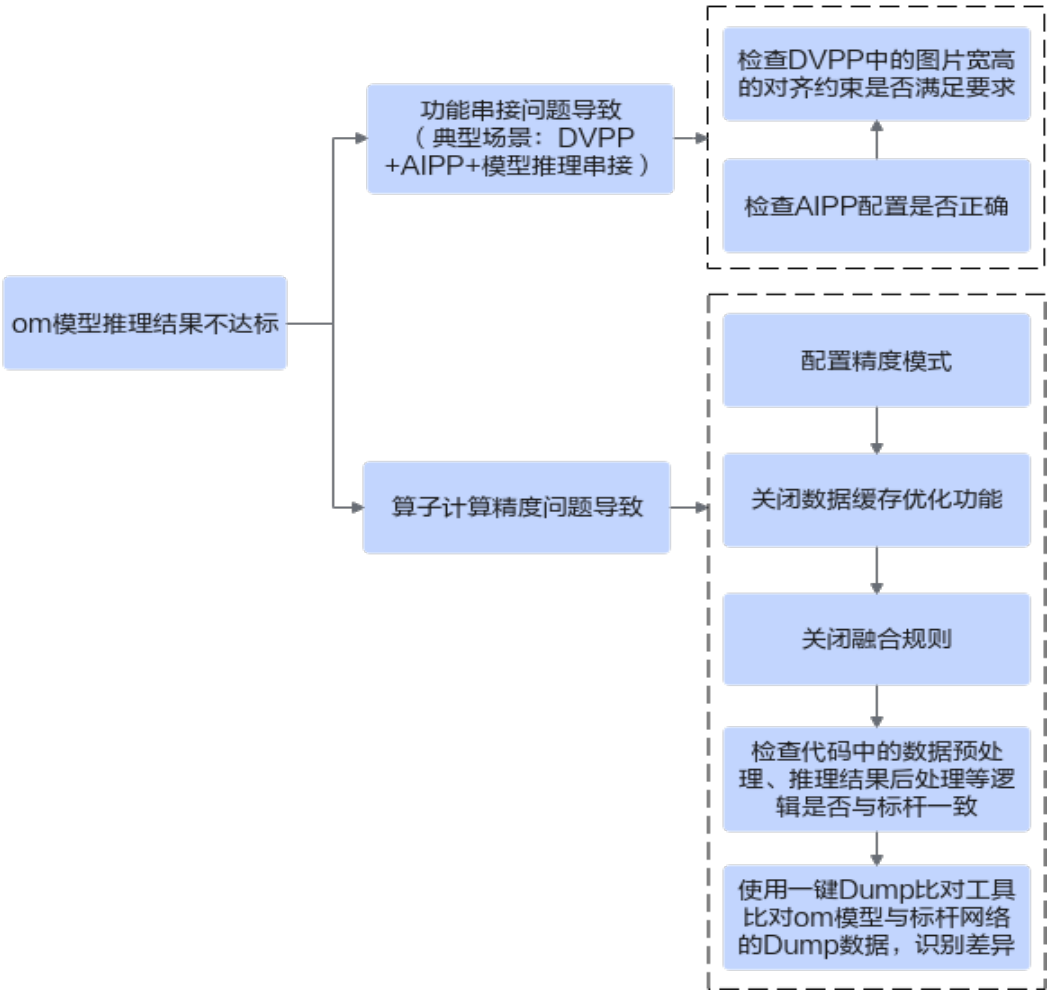
- 推理功能与其它功能之间的串接问题，当前比较典型的是DVPP+AIPP+模型推理串联使用的场景，该场景主要是因为接口中参数的配置问题、模型转换时AIPP的配置问题导致。

  如果存在DVPP、AIPP功能、模型推理串联使用的场景，建议先排查这部分问题；否则，可以跳过该排查，直接排查算子本身的精度问题。

本文也结合具体的正、反例给出了配置建议，供参考。

- 整网中算子本身的精度问题，该类问题可以借助精度比对工具，根据下文中具体的问题定位流程获取各类数据后，再进行比对、分析，确认是配置问题，还是算子实现问题，再逐一解决问题。本文中会结合具体的比对、分析的案例，介绍如何比对、分析。

图 9-1 推理精度问题



说明

- 本文中的“推理”，当前限定为使用om离线模型文件进行推理的场景。
- 本文中的算子精度问题，是指整网中算子的精度问题，对于自定义的TBE算子，运行验证时的发现精度问题，请参见《**TBE&AI CPU算子开发指南（开放态）**》。

9.2.2 DVPP+AIPP+模型推理精度提升建议（Atlas 200/300/500 推理产品）

DVPP+AIPP+模型推理串联使用，由于接口中参数的配置问题、模型转换时AIPP的配置问题可能会导致模块之间输入、输出的衔接存在问题，最后影响整网的推理精度。

请参考**5.6.1 JPEGD+VPC+模型推理精度提升建议（Atlas 200/300/500 推理产品）**中内容了解精度提升建议、典型案例以及示例代码。

9.2.3 算子精度导致推理结果不达标

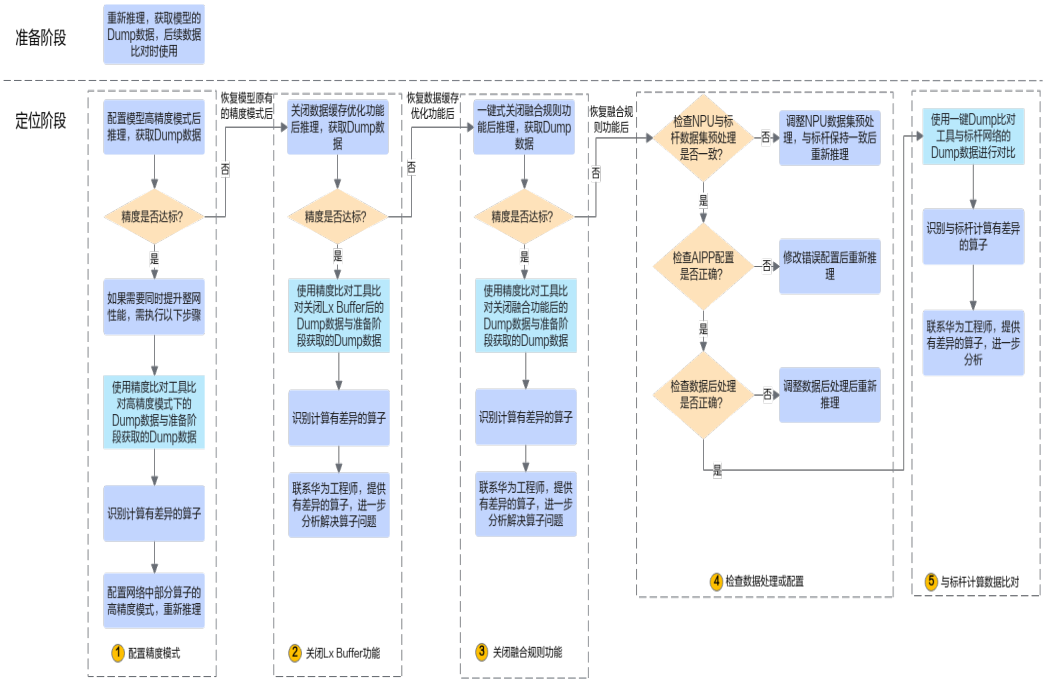
9.2.3.1 问题描述

推理结果不达标，包括以下两种情况：

- 算子精度导致推理结果错误，是指整网推理的功能已调通，但推理结果错误，例如目标检测网络MAP结果全0、昇腾om模型的推理结果与标杆网络的推理结果比对时余弦相似度为0。
- 算子精度导致推理精度不达标，是指整网推理的功能已调通，单次昇腾om模型的推理结果与标杆网络的推理结果比对时余弦相似度在95%以上，但数据集推理精度与标杆数据存在少量差距，例如：
  - 分类网络昇腾om模型，Top1/Top5分别为：0.90/0.70,；标杆网络Top1/Top5分别为：0.92/0.71。
  - 检测网络昇腾om模型MAP精度：0.54；标杆网络MAP精度：0.55。

9.2.3.2 问题定位流程

图 9-2 定位流程



步骤1 推理结果错误，为了后续定位问题，需要重新执行推理，用于获取模型的Dump数据。

获取模型的Dump数据，需要调用AscendCL接口打开Dump开关，详细描述请参见《**CANN 开发工具指南（开放态）**》中的“精度比对工具”章节。

步骤2 配置精度模式。

1. 配置模型高精度模式后推理，获取模型的Dump数据。推理后，如果精度达标，则进行步骤**步骤2.2**；如果精度不达标，则进行步骤**步骤3**。

配置模型高精度模式后推理，可能会影响推理性能，如果在精度达标的同时，需要保持性能，则执行**步骤2.2**~**步骤2.4**，配置部分算子保持原始网络中的数据类型。

配置模型高精度模式，请参见**配置网络模型的高精度模式**。

2. 使用精度比对工具比对高精度模式下的Dump数据与**步骤1**获取的Dump数据。

   工具的使用请参见《**CANN 开发工具指南（开放态）**》中的“精度比对工具”章节。

3. 根据**步骤2.2**中的比对结果识别计算有差异的算子。

   一般来说，每次识别一个差异算子（首个余弦相似度较低的算子，例如低于0.95），找到差异算子后，执行**步骤2.4**推理，推理的同时获取Dump数据，用来与高精度模式下的Dump数据比对，继续找到下一个差异算子。

   需要循环执行该步骤，直至没有差异算子。

4. 对于有差异的算子，配置该部分算子保持原始网络中的数据类型，再重新推理。

   配置部分算子的高精度模式，请参见**配置部分算子保持原始网络中的数据类型**。

步骤3 关闭数据缓存优化功能。

1. 恢复模型的原有精度模式后，关闭数据缓存优化功能后推理，如果精度达标，则进行步骤**步骤3.2**；如果精度不达标，则进行步骤**步骤4**。

   当前默认开启数据缓存优化，开启数据缓存优化可提高计算效率、提升性能，但由于部分算子在实现上可能存在未考虑的场景，导致影响精度，因此在出现精度问题时可以尝试关闭数据缓存优化。如果关闭数据缓存优化功能后，精度达标，则还是需要识别出问题算子，反馈给技术支持进一步分析、解决算子问题，解决算子问题后，建议保持开启数据缓存优化。

   关闭数据缓存优化功能，请参见**9.2.3.4 关闭数据缓存优化**。

2. 使用精度比对工具比对关闭数据缓存优化功能后的Dump数据与**步骤1**获取的Dump数据。

   工具的使用请参见《**CANN 开发工具指南（开放态）**》中的“精度比对工具”章节。

3. 根据**步骤3.2**中的比对结果识别计算有差异的算子。
4. 联系技术支持（您可以获取日志后单击**Link**联系技术支持。），提供有差异的算子，进一步分析。

步骤4 关闭融合规则功能。

1. 恢复启用数据缓存优化功能，关闭融合规则功能后推理，如果精度达标，则进行步骤**步骤4.2**；如果精度不达标，则进行步骤**步骤5**。

   当前默认开启融合规则，开启融合规则可提高计算效率、提升性能，但算子之间可能会融合，融合后的部分算子在实现上可能存在未考虑的场景，导致影响精度，因此在出现精度问题时可以尝试关闭融合规则。如果关闭融合规则功能后，精度达标，则还是需要识别出问题算子，反馈给技术支持进一步分析、解决算子问题，解决算子问题后，建议保持开启融合规则功能。

   关闭融合规则功能，请参见**9.2.3.5 关闭融合规则**。关闭某些融合规则可能会导致功能问题，因此在配置关闭融合规则后，系统在不影响功能的前提下关闭部分融合规则，而不是全部融合规则。

2. 使用精度比对工具比对关闭融合规则后的Dump数据与**步骤1**获取的Dump数据。

   工具的使用请参见《**CANN 开发工具指南（开放态）**》中的“精度比对工具”章节。

3. 根据**步骤4.2**中的比对结果识别计算有差异的算子。

4. 联系技术支持（您可以获取日志后单击**Link**联系技术支持。），提供有差异的算子，进一步分析。

步骤5 检查数据处理或配置。

推理精度不达标可能是由于数据集、AIPP、后处理方式的差异导致，需逐步进行排查，恢复启用融合规则功能后，请检查数据处理或配置，参见**9.2.3.6 检查数据处理或配置**。

如果数据处理逻辑或数据配置有问题，则需修改后重新推理；如果数据处理逻辑或数据配置没有问题，则进行**步骤6**。

步骤6 与标杆计算数据比对。

1. 使用一键Dump比对工具与标杆网络的Dump数据进行对比。

   一键Dump比对工具及其使用指导请单击**Link**获取。
2. 根据**步骤6.1**中的比对结果识别计算有差异的算子。
3. 联系技术支持（您可以获取日志后单击**Link**联系技术支持。），提供有差异的算子，进一步分析。

----结束

9.2.3.3 配置精度模式

如果在模式转换时不指定网络模型或算子的精度模式，默认采用fp16（float16）数据类型进行计算。

配置模型高精度模式后推理，可提升精度，但可能会影响推理性能，如果在精度达标的同时，需要保持性能，则可以配置部分算子保持原始网络中的数据类型。关于ATC参数的详细说明请参见《**ATC工具使用指南（开放态）**》。

配置网络模型的高精度模式

步骤1 使用ATC工具转换模型时，增加高级参数--precision_mode，用于指定精度模式。

参数设置如下所示，表示如果网络模型中算子支持fp32（float32），则使用fp32；如果网络模型中算子不支持fp32，则使用fp16（float16）。

```
--precision_mode=allow_fp32_to_fp16
```

步骤2 使用转换后的om模型重新推理。

----结束

配置部分算子保持原始网络中的数据类型

步骤1 使用ATC工具转换模型时，增加高级参数--keep_dtype（指定部分算子计算时保持原始网络的数据类型）和--precision_mode（指定网络模型的精度模式）。

参数使用示例如下：

```
--keep_dtype=$HOME/execeptionlist.cfg --precision_mode=force_fp16
```

配置文件名举例为*execeptionlist.cfg*，配置文件样例如下，文件中每一行是一个算子的名称，将配置好的*execeptionlist.cfg*文件上传到ATC工具所在服务器任意目录：

```
Opname1
Opname2
...
```

步骤2 使用转换后的om模型重新推理。

----结束

9.2.3.4 关闭数据缓存优化

如果在模型转换时不指定关闭数据缓存优化功能，当前默认开启数据缓存优化，开启数据缓存优化可提高计算效率、提升性能，但由于部分算子在实现上可能存在未考虑的场景，导致影响精度，因此在出现精度问题时可以尝试关闭数据缓存优化。

如果关闭数据缓存优化功能后，精度达标，则还是需要识别出问题算子，反馈给技术支持进一步分析、解决算子问题，解决算子问题后，建议保持开启数据缓存优化。

步骤1 使用ATC工具转换模型时，增加高级参数：--buffer_optimize，用于关闭数据缓存优化。

参数设置如下所示，：

```
--buffer_optimize=off_optimize
```

参数的详细说明请参见《**ATC工具使用指南（开放态）**》。

步骤2 使用转换后的om模型重新推理。

----结束

说明

在联系技术支持前（您可以获取日志后单击**Link**联系技术支持。），设置DUMP_GE_GRAPH、DUMP_GRAPH_LEVEL环境变量，重新模型转换，打印模型转换过程中各个阶段的图描述信息。关于环境变量以及图描述信息的说明，请参见《**ATC工具使用指南（开放态）**》中的“参考>dump图详细信息”。

9.2.3.5 关闭融合规则

如果在模型转换时不指定关闭融合规则，当前默认开启融合规则，开启融合规则可提高计算效率、提升性能，但算子之间可能会融合，融合后的部分算子在实现上可能存在未考虑的场景，导致影响精度，因此在出现精度问题时可以尝试关闭融合规则。

如果关闭融合规则功能后，精度达标，则还是需要识别出问题算子，反馈给技术支持进一步分析、解决算子问题，解决算子问题后，建议保持开启融合规则功能。

步骤1 使用ATC工具转换模型时，增加高级参数：--fusion_switch_file

参数使用示例如下：

```
--fusion_switch_file=$HOME/module/fusion_switch.cfg
```

配置文件名举例为*fusion_switch.cfg*，配置文件样例如下，将配置好的*fusion_switch.cfg*文件上传到ATC工具所在服务器任意目录：

```
{
   "Switch":{
      "GraphFusion":{
         "ALL":"off"
      },
      "UBFusion":{
         "ALL":"off"
       }
   }
}
```

参数的详细说明请参见《**ATC工具使用指南（开放态）**》。

步骤2 使用转换后的om模型重新推理。

----结束

说明

在联系技术支持前（您可以获取日志后单击**Link**联系技术支持。），设置DUMP_GE_GRAPH、DUMP_GRAPH_LEVEL环境变量，重新模型转换，打印模型转换过程中各个阶段的图描述信息。关于环境变量以及图描述信息的说明，请参见《**ATC工具使用指南（开放态）**》中的“参考>dump图详细信息”。

9.2.3.6 检查数据处理或配置

步骤1 检查昇腾om模型与标杆网络推理的输入数据以及输入数据的处理是否一致，如果不一致，需调整成一致。

步骤2 检查AIPP配置。

AIPP（Artificial Intelligence Pre-Processing），用于在AI Core上完成图像预处理，包括改变图像尺寸、色域转换（转换图像格式）、减均值/乘系数（改变图像像素），数据处理之后再进行真正的模型推理。

如果AIPP配置错误可能导致模型推理的输入数据不准确，需要参见《**ATC工具使用指南（开放态）**》中的“高级功能>AIPP使能”章节检查AIPP配置，如有不正确的AIPP配置，修改正确后，重新转换模型，再重新推理。

步骤3 检查昇腾om模型与标杆网络推理结果的后处理方式是否一致，如果不一致，需调整成一致。

----结束

9.2.3.7 案例介绍

案例描述

Fastrcnn网络，模型转换时，保持默认高性能模式、force_fp16精度模式，推理出来的精度错误，MAP结果为0。

然后，在模型转换时，设置模型的高精度模式（precision_mode=allow_fp32_to_fp16），推理出来的精度正确。

案例分析

1. 模型转换时保持默认高性能模式、force_fp16精度模式，进行推理，获取该模式下的Dump数据文件。
2. 再次模型转换，设置模型的高精度模式（precision_mode=allow_fp32_to_fp16），再次进行推理，获取该模式下的Dump数据文件。
3. 使用精度比对工具，比对**1**与**2**中的Dump数据。

   比对结果示例如下：

Index	LeftOp	RightOp	TensorIndex	CosineSim	MaxAbsolu	AccumulatI	RelativeE	KullbackL	StandardD	CompareFailReason
844	multilevel_crop_an	multilevel	multilevel_crop_and_resize/AddN:input:1	NaN	NaN	NaN	NaN	NaN	NaN	[multilevel_crop_and_resize/
844	multilevel_crop_an	multilevel	multilevel_crop_and_resize/AddN:input:2	NaN	NaN	NaN	NaN	NaN	NaN	[multilevel_crop_and_resize/
844	multilevel_crop_an	multilevel	multilevel_crop_and_resize/AddN:input:3	NaN	NaN	NaN	NaN	NaN	NaN	[multilevel_crop_and_resize/
844	multilevel_crop_an	multilevel	multilevel_crop_and_resize/AddN:output:0	0.722928	130050	460707.5	1.057259	inf	(84351.02	Cannot compare by KL Diverge
845	multilevel_crop_an	multilevel	multilevel_crop_and_resize/Cast_2:input:0	0.722928	130050	460707.5	1.057259	inf	(84351.02	Cannot compare by KL Diverge
845	multilevel_crop_an	multilevel	multilevel_crop_and_resize/Cast_2:output:0	0.722928	130050	460707.5	1.057259	inf	(84351.02	Cannot compare by KL Diverge
846	multilevel_crop_an	multilevel	multilevel_crop_and_resize/GatherV2:input:0	0.999999	0.073951	343481.8	0.001661	0	(-0.025,0.937)	(-0.025,0.937)
846	multilevel_crop_an	multilevel	multilevel_crop_and_resize/GatherV2:input:1	0.722928	130050	460707.5	1.057259	inf	(84351.02	Cannot compare by KL Diverge
846	multilevel_crop_an	multilevel	multilevel_crop_and_resize/GatherV2:output:0	0.015732	16.76621	3.4E+08	1.304415	inf	(-0.060,1	Cannot compare by KL Diverge

4. 从图中可以看CosineSimilarity这一列，余弦相似度算法比对出来的结果，范围是[-1,1]，比对的结果如果越接近1，表示两者的值越相近，越接近-1意味着两者的值越相反。对于大部分算子，值低于0.95就说明存在精度问题。

   上图中AddN算子第0个输出的余弦相似度只有0.72，说明这个算子可能存在精度问题，因此需要进一步分析该算子在高精度模式下的第0个输出的Dump数据文件（2中获取的Dump数据文件）。

5. 由于Dump数据文件无法通过文本工具直接查阅，因此在分析该Dump数据文件前，请参考《**CANN 开发工具指南（开放态）**》中的“精度比对工具”章节的“附录>如何查看dump数据文件”章节，先将dump数据文件转换为numpy格式，再将numpy格式文件为txt格式文件。

   在将numpy格式文件为txt格式文件的过程中，可以获取AddN算子第0个输出的最大值、最小值，命令示例如下（*******.npy***表示numpy格式文件的路径）：

```
$ python3
Python 3 (default, Mar  5 2020, 16:07:54) [GCC 5.4.0 20160609] on linuxType "help", "copyright", "credits" or "license" for more information.
>>> import numpy as np
>>> a = np.load("****.npy")
>>> a.max()
>>> 109508.0
>>> a.min()
>>> 70683.0
```

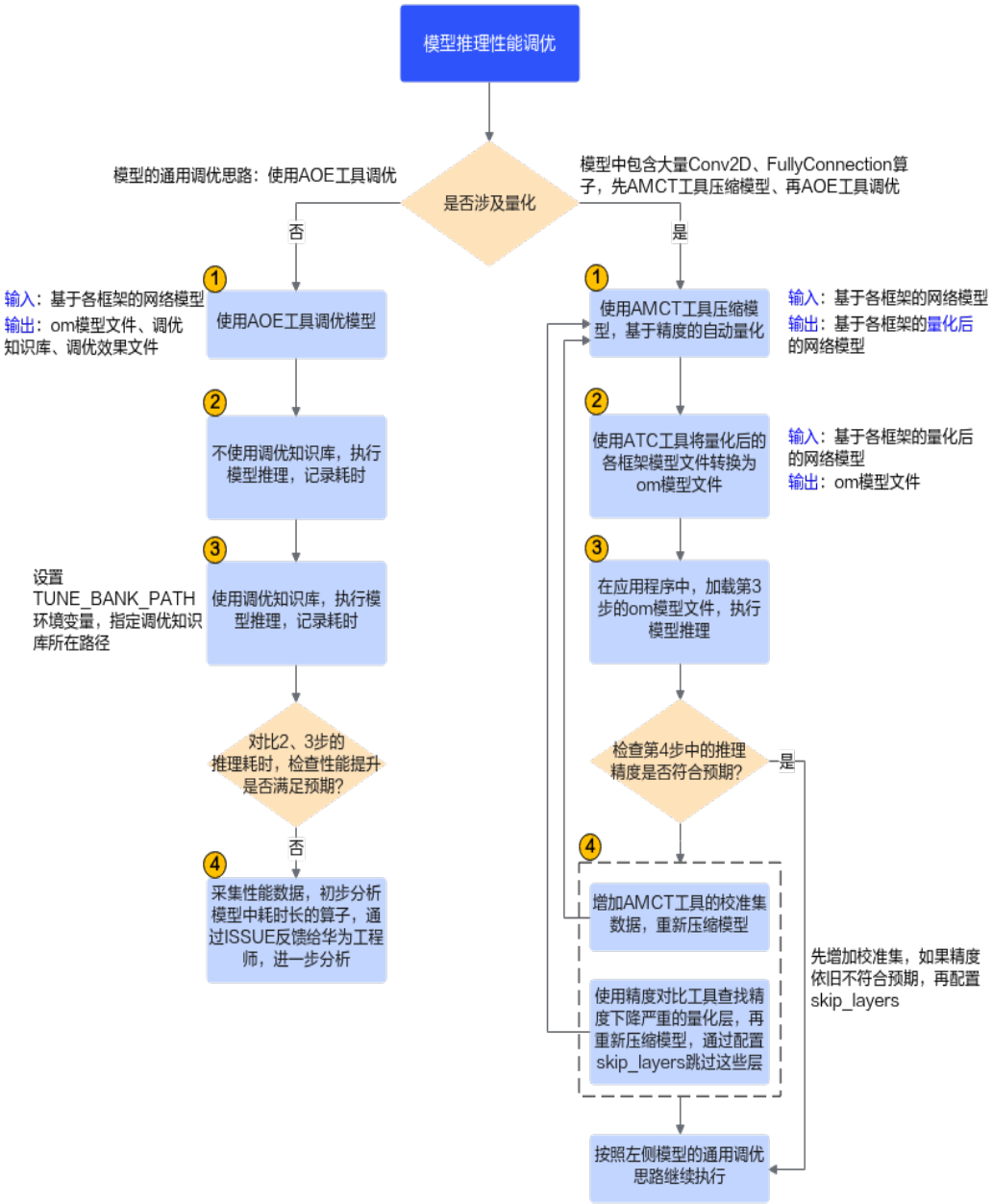
6. 从5获取到的AddN算子第0个输出的最大值、最小值，可以看出高精度模式下AddN算子输出tensor的最大值为109508.0，而高性能模式（fp16）下，输出tensor的最大值为65504.0（FP16能表达的最大值域范围为（-65505~65504），由此可以得出高精度模式下AddN算子的输出值大于fp16类型域表达范围，因此需要配置该算子走高精度模式，参见**配置部分算子保持原始网络中的数据类型**。

9.3 模型推理性能调优建议

9.3.1 模型推理性能调优思路

在整网推理时，可能由于模型在昇腾AI处理器上的算子适配、数据读写等问题，导致模型推理的性能不符合预期，您可以查阅本节介绍的内容，了解模型推理时的性能调优流程。由于是调优，因此在调优前，请确保已经完成了整网推理功能调测，功能不阻塞，只是模型推理性能不符合预期、待提升。

在下图的性能调优流程中，涉及调优的关键工具为：模型调优工具AOE（Ascend Optimization Engine）、模型压缩工具AMCT（Ascend Model Compression Toolkit）。在调优过程中，涉及转换模型、记录模型推理耗时、分析性能瓶颈点等操作时，还会辅助使用模型转换工具ATC、性能数据采集工具、精度比对工具。



9.3.2 使用 AOE 工具调优模型

使用aoe命令调优时，会生成：适配昇腾AI处理器的om模型文件、基于昇腾AI处理器的调优知识库、调优效果文件（通过效果文件可查阅模型中各算子的性能提升率）。由于生成om模型文件，所以aoe命令是包含模型转换的功能，因此aoe命令的参数，一般来说，是在atc命令参数的基础上增加了**--job_type**参数，增加**子图调优**、**算子调优**的功能。

先进行子图调优，再进行算子调优，原因是：先进行子图调优会生成图的切分方式，子图调优后算子已经被切分成最终的shape了，再进行算子调优，会基于这个最终shape去做算子调优。如果优先算子调优，这时调优的算子shape不是最终切分后的算子shape，可能会影响最终的调优效果。

本节介绍**推理**场景下使用**aoe命令**进行子图调优、算子调优的基本方法，关于AOE工具的其它参数、使用约束介绍请参见《**AOE工具使用指南**》。

操作步骤

1. **使用aoe命令依次执行子图调优、算子调优。**
   - **子图调优命令示例如下所示：**
     ```
     aoe --model=${HOME}/module/resnet50_pytorch_1.4.onnx --framework=5 --job_type=1
     ```
   - **算子调优命令示例如下所示：**
     ```
     aoe --model=${HOME}/module/resnet50_pytorch_1.4.onnx --framework=5 --job_type=2
     ```
2. **查看调优结果。**

   执行调优结果如下所示，代表调优完成并且性能有提升。调优后，同时生成自定义知识库、om模型文件以及调优效果文件。

   ```
   <xxxx> process finished. Performance improved by xx%    //xxxx：调优任务名称,xx%：性能提升的比例。
   ```

   调优结果文件如下：
   - **基于昇腾AI处理器的调优知识库**
     - 针对子图调优，默认存储到${HOME}/Ascend/latest/data/aoe/custom/graph/${soc_version}目录下。
     - 针对算子调优，默认存储到${HOME}/Ascend/latest/data/aoe/custom/op/${soc_version}路径下。
   - **适配昇腾AI处理器的om模型文件**

     调优后的om模型默认存放在执行aoe命令的当前目录下，具体路径如下：${model_name}_${timestamp}/tunespace/result/${model_name}_${timestamp}_tune.om（或者${model_name}_${timestamp}_tune_${os}_${arch}.om）。
   - **调优效果文件**

     执行aoe命令的当前目录下生成命名为“aoe_result_opat_{timestamp}_{pid*xxx*}.json”的文件，记录调优过程中被调优的算子信息。

     json文件中的内容片段示例如下：
     ```
     {
             "op_name": "Conv_125",
             "op_type": "Conv2D",
             "tune_performance": {
               "Schedule": {
                 "performance_after_tune(us)": 72.046,
                 "performance_before_tune(us)": 72.055,
                 "performance_improvement": "0.01%",
                 "update_mode": "add"
               }
             }
     }
     ```
3. **指定调优知识库，再执行模型推理。**
   a. 设置TUNE_BANK_PATH环境变量，指定为AOE调优后的自定义知识库存放路径，该路径下的graph目录下为子图调优知识库、op目录下为算子调优知识库。示例如下：
      ```
      export TUNE_BANK_PATH=/home/HwHiAiUser/custom
      ```
   b. AscendCL模型推理。

      使用**1**中执行算子调优命令生成的om模型文件，执行模型推理。

      或者可以使用**msame工具**快速推理，查看推理耗时数据。

说明

支持“一次AOE调优+多次ATC模型转换”的场景，使用AOE工具调优模型后，若由于其它业务需求需要重新转换模型，可通过环境变量指定AOE调优知识库的路径，再使用ATC工具重新转换模型，这样就可以基于知识库中的调优策略编译模型、转换出调优后的模型。

9.3.3 采集&解析性能数据

本节介绍**推理**场景下使用**msprof命令行**方式采集和解析性能数据、并通过生成的结果文件分析性能瓶颈的基本方法，关于msprof命令行的详细参数介绍、以及其它性能数据采集分析方式请参见《**CANN 开发工具指南（开放态）**》中的“性能分析工具”章节。

采集、解析并导出性能数据

步骤1 执行msprof命令一键式采集、解析并导出性能数据。

```
msprof --application=/home/HwHiAiUser/HIAI_PROJECTS/MyAppname/out/main --output=/home/HwHiAiUser/profiling_output
```

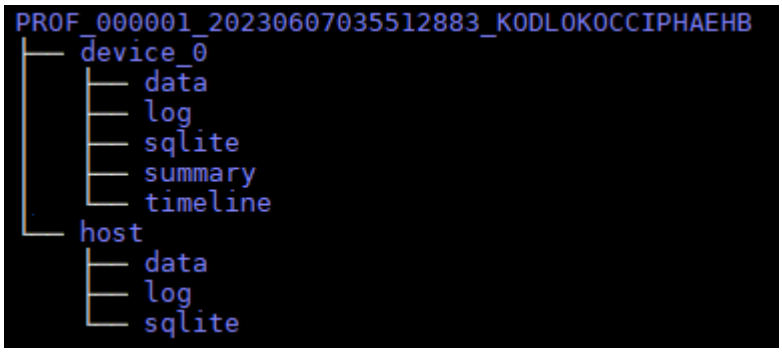
表 9-1 常用参数说明

参数	描述	可选/必选
--application	配置为运行环境上AI任务文件。 不建议配置其他用户目录下的AI任务，避免提权风险。 不建议使用此参数进行有安全风险的高危操作，如删除文件或目录、修改密码、提权命令等。	必选
--output	收集到的Profiling数据的存放路径，默认为AI任务文件所在目录。	可选

默认情况下，导出迭代数最多的模型ID（Model ID）对应的第一轮迭代的性能数据。

步骤2 命令执行完成后，在output指定的目录下生成**PROF_XXX目录，存放采集并解析后的性能数据**，目录结构如图9-3所示。

图 9-3 性能数据目录结构（仅为示例）



- data/sqlite文件夹为采集和解析的过程数据，一般无需关注。
- log文件夹为日志文件，一般无需关注。
- summary文件夹汇总了AI任务运行时的软硬件数据。
- timeline文件夹汇总了AI任务运行的时序信息。

步骤3 进入summary和timeline目录，**查看性能数据文件**。

默认情况下采集到的文件请参考**表9-2**。

表 9-2 msprof 默认配置采集的性能数据文件

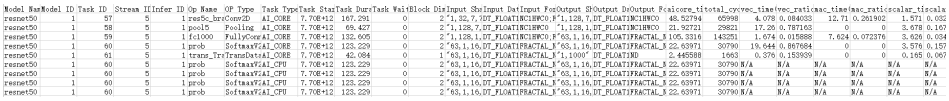
文件夹	文件名	说明
timeline	msprof*.json	timeline数据总表。
	acl_*.json	AscendCL接口调用时序。训练场景不生成。
	ai_stack_time_*.json	昇腾AI软件栈各组件（AscendCL，GE，Runtime，Task Scheduler等）运行时序。
	ge_*.json	GE接口耗时数据。
	step_trace_*.json	迭代轨迹数据，每轮迭代的耗时。
	task_time_*.json	Task Scheduler任务调度时序。
	thread_group_*.json	AscendCL，GE，Runtime组件耗时数据。
	ge_op_execute_*.json	算子下发各阶段耗时数据。当模型为动态Shape时自动采集并生成该文件。
summary	acl_*.csv	AscendCL API调用过程。训练场景不生成。
	acl_statistic_*.csv	AscendCL API数据统计。训练场景不生成。
	op_summary_*.csv	AI Core和AI CPU算子数据。
	op_statistic _*.csv	AI Core和AI CPU算子调用次数及耗时统计。
	step_trace_*.csv	迭代轨迹数据。
	task_time_*.csv	Task Scheduler任务调度信息。
	ai_stack_time_*.csv	昇腾AI软件栈各组件（AscendCL，GE，Runtime，Task Scheduler等）信息。
	fusion_op_*.csv	模型中算子融合前后信息。
	ge_op_execute_*.csv	算子下发各阶段耗时数据。当模型为动态Shape时自动采集并生成该文件。
	prof_rule_0.json	调优建议。
注：“*”表示{device_id}_{model_id}_{iter_id}，其中{device_id}表示设备ID，{model_id}表示模型ID，{iter_id}表示某轮迭代的ID。		

- timeline文件后缀为json，需要在Chrome浏览器中输入**chrome://tracing**，将文件拖到空白处进行打开，通过键盘上的快捷键（w：放大，s：缩小，a：左移，d：右移）。通过该文件可查看当前AI任务运行的时序信息，比如运行过程中接口调用时间线，如图9-4所示。

  图 9-4 查看 timeline 文件

- summary文件后缀为csv，可直接打开查看。通过该文件可以看到AI任务运行时的软硬件数据，比如各算子在AI处理器软硬件上的运行耗时，通过字段排序等可以快速找出需要的信息，如图9-5所示。

  图 9-5 查看 summary 文件

Model Name	Model ID	Task ID	Stream ID	Infer ID	Op Name	OP Type	Task Type	Task Start	Task Duration	Wait Block Dim	Input Shape	Input Data	Input Format	Output Shape	Output Data	Output Format	Aicore_time(us)	total_cycle	vec_time(us)	vec_ratio	mac_time(us)	mac_ratio	scalar_time(us)	scalar_ratio
resnet50	1	57	5	1	res5c_branch2c	Conv2D	AI_CORE	7.70E+12	167.291	0	2	"1,32,7,7"	DT_FLOAT16	C1HWC0	"1,128,7,7"	DT_FLOAT16	C1HWC0	48.52794	65998	4.078	0.084033	12.71	0.261902	1.571	0.032
resnet50	1	58	5	1	pool5	Pooling	AI_CORE	7.70E+12	69.427	0	2	"1,128,7,7"	DT_FLOAT16	C1HWC0	"1,128,1,1"	DT_FLOAT16	C1HWC0	21.92721	29821	17.26	0.787163	0	0	3.678	0.167
resnet50	1	59	5	1	fc1000	FullyConv	AI_CORE	7.70E+12	132.605	0	2	"1,128,1,1"	DT_FLOAT16	C1HWC0	"63,1,16,16"	DT_FLOAT16	FRACTAL_Z	105.3316	143251	1.674	0.015888	7.624	0.072376	3.626	0.034
resnet50	1	60	5	1	prob	SoftmaxV2	AI_CORE	7.70E+12	123.229	0	2	"63,1,16,16"	DT_FLOAT16	FRACTAL_Z	"63,1,16,16"	DT_FLOAT16	FRACTAL_Z	22.63971	30790	19.644	0.867684	0	0	3.576	0.157
resnet50	1	61	5	1	trans_TransData	AI_CORE	7.70E+12	42.084	0	1	"63,1,16,16"	DT_FLOAT16	FRACTAL_Z	"1,1000"	DT_FLOAT16	ND	2.445588	1963	0.376	0.153839	0	0	0.165	0.067
resnet50	1	60	5	1	prob	SoftmaxV2	AI_CPU	7.70E+12	123.229	0	2	"63,1,16,16"	DT_FLOAT16	FRACTAL_Z	"63,1,16,16"	DT_FLOAT16	FRACTAL_Z	22.63971	30790	N/A	N/A	N/A	N/A	N/A	N/A
resnet50	1	60	5	1	prob	SoftmaxV2	AI_CPU	7.70E+12	123.229	0	2	"63,1,16,16"	DT_FLOAT16	FRACTAL_Z	"63,1,16,16"	DT_FLOAT16	FRACTAL_Z	22.63971	30790	N/A	N/A	N/A	N/A	N/A	N/A
resnet50	1	60	5	1	prob	SoftmaxV2	AI_CPU	7.70E+12	123.229	0	2	"63,1,16,16"	DT_FLOAT16	FRACTAL_Z	"63,1,16,16"	DT_FLOAT16	FRACTAL_Z	22.63971	30790	N/A	N/A	N/A	N/A	N/A	N/A
resnet50	1	60	5	1	prob	SoftmaxV2	AI_CPU	7.70E+12	123.229	0	2	"63,1,16,16"	DT_FLOAT16	FRACTAL_Z	"63,1,16,16"	DT_FLOAT16	FRACTAL_Z	22.63971	30790	N/A	N/A	N/A	N/A	N/A	N/A

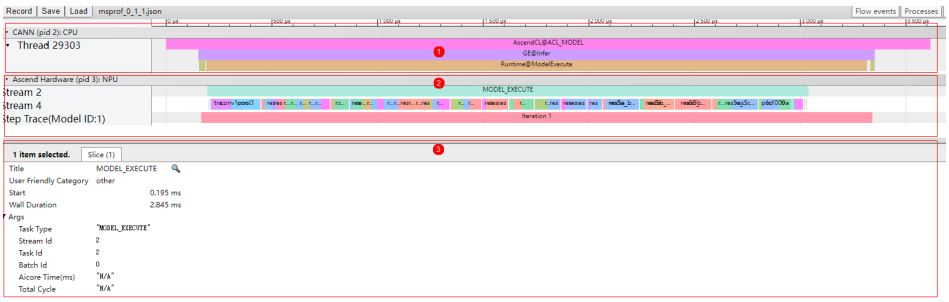
----结束

性能分析

从上文我们可以看到，性能数据文件较多，分析方法也较灵活，以下介绍几个重要文件及分析方法。

- 通过**msprof*.json**文件从整体角度查看AI任务运行的时序信息，进而分析出可能存在的瓶颈点。

  图 9-6 msprof*.json 文件示例

  - 区域1：CANN层数据，主要包含AscendCL、GE和Runtime组件的耗时数据。
  - 区域2：底层NPU数据，主要包含Task Schduler组件耗时数据、迭代轨迹数据。
  - 区域3：展示timeline中各算子、接口的详细信息，单击区域1和区域2中各timeline时展示。

  从上图可以大致分析出AI任务在哪个阶段耗时较多，比如发现区域1的AscendCL aclmdlExecute接口执行阶段耗时较多，可以继续查看区域2的Task Schduler任务调度信息，分析执行推理过程中具体耗时较长的任务，查看区域3的耗时较长的接口和算子，再结合summary文件进行量化分析，定位出具体的性能瓶颈。

- 通过**op_statistic_*.csv**文件分析各类算子的调用总时间、总次数等，排查是否某类算子总耗时较长，进而分析这类算子是否有优化空间。

  图 9-7 op_statistic_*.csv 文件示例

Model Nar	OP Type	Core Type	Count	Total Time(us)	Min Time(u	Avg Time(u	Max Time(	Ratio(%)
resnet50	Conv2D	AI_CORE	50	5618.09	39.11	112.361	330.37	84.1293
resnet50	Pooling	AI_CORE	2	263.65	80.84	131.825	182.81	3.9481
resnet50	ReadSelec	AI_CORE	3	209.36	68.23	69.786	71.4	3.1351
resnet50	FullyConne	AI_CORE	1	204.37	204.37	204.37	204.37	3.0604
resnet50	SoftmaxV2	AI_CORE	1	154.42	154.42	154.42	154.42	2.3124
resnet50	Cast	AI_CORE	2	115.9	37.61	57.95	78.29	1.7356
resnet50	TransData	AI_CORE	2	112.13	39.94	56.065	72.19	1.6791

  可以按照Total Time排序，找出哪类算子耗时较长。

- 通过**op_summary_*.csv**文件分析具体某个算子的信息和耗时情况，从而找出高耗时算子，进而分析该算子是否有优化空间。

  图 9-8 op_summary*.csv 文件示例

Model Nar	Model ID	Task ID	Stream ID	Infer ID	Op Name	OP Type	Task Type	Task Start	Task Duration(us)
resnet50	1	53	1	1	res5b_bran	Conv2D	AI_CORE	1.19E+15	330.37
resnet50	1	50	1	1	res5a_bran	Conv2D	AI_CORE	1.19E+15	325.1
resnet50	1	54	1	1	res5b_bran	Conv2D	AI_CORE	1.19E+15	281.66
resnet50	1	57	1	1	res5c_bran	Conv2D	AI_CORE	1.19E+15	281.1
resnet50	1	55	1	1	res5b_bran	Conv2D	AI_CORE	1.19E+15	273.23
resnet50	1	52	1	1	res5a_bran	Conv2D	AI_CORE	1.19E+15	238.59
resnet50	1	51	1	1	res5a_bran	Conv2D	AI_CORE	1.19E+15	213.7
resnet50	1	60	1	1	fc1000	FullyConne	AI_CORE	1.19E+15	204.37

  Task Duration字段为算子耗时信息，可以按照Task Duration排序，找出高耗时算子；也可以按照Task Type排序，查看不同核（AI Core和AI CPU）上运行的高耗时算子。

9.3.4 使用 AMCT 工具压缩模型

9.3.4.1 基于精度的自动量化

基于精度的自动量化是为了方便用户在对量化精度有一定要求时所使用的功能，是借助AMCT工具提供的Python接口实现。该方法能够在保证用户所需的模型精度前提下，自动搜索模型的量化配置并执行**训练后量化**的流程，最终生成满足精度要求的量化模型。

当前仅如下框架支持使用基于精度的自动量化：

- **PyTorch**框架，详细说明请参见《**CANN 开发工具指南（开放态）**》中的“AMCT工具（PyTorch）”章节。
- **TensorFlow**框架，详细说明请参见《**CANN 开发工具指南（开放态）**》中的“AMCT工具（TensorFlow）”章节。
- **Caffe**框架，详细说明请参见《**CANN 开发工具指南（开放态）**》中的“AMCT工具（Caffe）”章节。

9.3.4.2 关于增加校准集的建议

在基于精度的自动量化时，手动修改量化配置中的batch_num，可根据batch大小以及量化需要使用的图片数量调整，以便调整校准使用的数据量。batch_num控制量化使

用数据的batch数目，batch_num = total_nums / batch_size，其中，total_nums表示总的图片数量，batch_size表示每个batch使用的图片数量。

通常情况下：

batch_num越大，量化过程中使用的数据样本越多，量化后精度损失越小；但过多的数据并不会带来精度的提升，反而会占用较多的内存，降低量化的速度，并可能引起内存、显存、线程资源不足等情况；因此，建议batch_num*batch_size为16或32。

9.3.4.3 查找量化场景下的精度损失层

当前**仅支持Caffe**框架模型，使用精度比对工具查找量化场景下的精度损失层，主要包含以下两步的比对：

1. **定位量化阶段的精度问题**。

   执行非量化原始模型（GPU/CPU） vs 量化原始模型（GPU/CPU）。

2. **定位模型转换阶段产生的精度问题**，即量化离线模型在NPU上运行时的精度问题。

   执行量化原始模型（GPU/CPU） vs 量化离线模型（关闭融合规则）（NPU）。

详细操作请参见《**CANN 开发工具指南（开放态）**》中的“精度比对工具”章节中的“GPU/CPU vs Ascend NPU精度比对（Caffe推理）”。

9.3.4.4 配置 skip_layers 跳过量化损失大的层

使用精度比对工具比对原始模型、量化后模型各层的精度，查找精度损失大的层，在基于精度的自动量化时通过配置skip_layers跳过量化损失大的层。

步骤1 修改基于精度的自动量化代码，增加skip_layers配置，跳过不支持量化的层。

代码示例如下，表示跳过某一层。如果涉及跳过多层，则每一层之间用逗号分隔，例如：skip_layers=['op1','op2','op3']

```
config_json_file = './config.json'
skip_layers = ['Default/network-DeepLabV3/resnet-Resnet/layer4-SequentialCell/0-Bottleneck/downsample-SequentialCell/0-Conv2d/Conv2D-op311']
batch_num = 1
amct.create_quant_config(config_json_file, model, input_data,
                         skip_layers, batch_num)

scale_offset_record_file = os.path.join(TMP, 'scale_offset_record.txt')
result_path = os.path.join(RESULT, 'model')
```

步骤2 执行精度的自动量化。

步骤3 重新执行推理。

如果跳过不支持量化的层影响模型推理的结果数据，则需要用户自行调整模型，再重新量化模型。

----结束

9.4 DVPP 数据处理高性能编程建议

目前在处理图像、视频数据时，可以使用VPC多功能组合接口、VPC批处理接口、合理选择VDEC的输出格式等编码方式，来提升应用的性能，具体编程建议请参见**5.7 高性能编程建议**。

10 应用样例参考

样例列表(Atlas 200/300/500 推理产品)
样例列表(Atlas 推理系列产品)
实现矩阵-矩阵乘运算
基于Caffe ResNet-50网络实现图片分类（图片解码+缩放+同步推理）
基于Caffe ResNet-50网络实现图片分类（图片解码+抠图缩放+图片编码+同步推理）
基于Caffe ResNet-50网络实现图片分类（视频解码+同步推理）
基于Caffe ResNet-50网络实现图片分类（同步推理）
基于Caffe ResNet-50网络实现图片分类（异步推理）
媒体数据处理V1（抠图，一图多框）
媒体数据处理V1（视频编码）
媒体数据处理V1（抠图贴图）
基于Caffe YOLOv3网络实现目标检测（动态Batch/动态分辨率）
媒体数据处理V2（VPC抠图/贴图/缩放等）
媒体数据处理V2（JPEGD图片解码）
媒体数据处理V2（JPEGE图片编码）
媒体数据处理V2（VDEC视频解码）
媒体数据处理V2（VENC视频编码）
媒体数据处理V2（PNGD图片解码）

10.1 样例列表(Atlas 200/300/500 推理产品)

本文中提及的样例如下表所示。单击**Gitee**或**Github**获取更多样例。

表 10-1 Sample 列表

Sample名称	Sample获取	基本功能	编译运行指导（Ctrl CPU开放形态）
gemm	单击**gemm**获取样例	实现矩阵-矩阵乘运算	**10.3 实现矩阵-矩阵乘运算**
vpc_resnet50_imagenet_classification	单击**vpc_resnet50_imagenet_classification**获取样例	基于Caffe ResNet-50网络实现图片分类（图片解码+缩放+同步推理）	**10.4 基于Caffe ResNet-50网络实现图片分类（图片解码+缩放+同步推理）**
vpc_jpeg_resnet50_imagenet_classification	单击**vpc_jpeg_resnet50_imagenet_classification**获取样例	基于Caffe ResNet-50网络实现图片分类（图片解码+抠图缩放+图片编码+同步推理）	**10.5 基于Caffe ResNet-50网络实现图片分类（图片解码+抠图缩放+图片编码+同步推理）**
vdec_resnet50_classification	单击**vdec_resnet50_classification**获取样例	基于Caffe ResNet-50网络实现图片分类（视频解码+同步推理）	**10.6 基于Caffe ResNet-50网络实现图片分类（视频解码+同步推理）**
resnet50_imagenet_classification	单击**resnet50_imagenet_classification**获取样例	基于Caffe ResNet-50网络实现图片分类（同步推理）	**10.7 基于Caffe ResNet-50网络实现图片分类（同步推理）**
resnet50_async_imagenet_classification	单击**resnet50_async_imagenet_classification**获取样例	基于Caffe ResNet-50网络实现图片分类（异步推理）	**10.8 基于Caffe ResNet-50网络实现图片分类（异步推理）**
batchcrop	单击**batchcrop**获取样例	媒体数据处理V1（抠图，一图多框）	**10.9 媒体数据处理V1（抠图，一图多框）**
venc_image	单击**venc_image**获取样例	媒体数据处理V1（视频编码）	**10.10 媒体数据处理V1（视频编码）**
smallResolution_cropandpaste	单击**smallResolution_cropandpaste**获取样例	媒体数据处理V1（抠图贴图）	**10.11 媒体数据处理V1（抠图贴图）**
YOLOV3_dynamic_batch_detection_picture	单击**YOLOV3_dynamic_batch_detection_picture**获取样例	基于Caffe YOLOv3网络实现目标检测（动态Batch/动态分辨率）	**10.12 基于Caffe YOLOv3网络实现目标检测（动态Batch/动态分辨率）**

Sample名称	Sample获取	基本功能	编译运行指导（Ctrl CPU开放形态）
API Samples	API Samples	基于AscendCL架构开发的一系列样例代码，包含简单的编译演示样例	下载样例后查看readme
图像目标检测	图像目标检测	基于AscendCL架构的目标检测开发Demo	下载样例后查看readme

10.2 样例列表(Atlas 推理系列产品)

本文中提及的样例如下表所示。单击Gitee或Github获取更多样例。

表 10-2 Sample 列表

Sample名称	Sample获取	基本功能	编译运行指导（Ctrl CPU开放形态）
gemm	单击gemm获取样例	实现矩阵-矩阵乘运算	10.3 实现矩阵-矩阵乘运算
vpc_resnet50_imagenet_classification	单击vpc_resnet50_imagenet_classification获取样例	基于Caffe ResNet-50网络实现图片分类（图片解码+缩放+同步推理）	10.4 基于Caffe ResNet-50网络实现图片分类（图片解码+缩放+同步推理）
vpc_jpeg_resnet50_imagenet_classification	单击vpc_jpeg_resnet50_imagenet_classification获取样例	基于Caffe ResNet-50网络实现图片分类（图片解码+抠图缩放+图片编码+同步推理）	10.5 基于Caffe ResNet-50网络实现图片分类（图片解码+抠图缩放+图片编码+同步推理）
vdec_resnet50_classification	单击vdec_resnet50_classification获取样例	基于Caffe ResNet-50网络实现图片分类（视频解码+同步推理）	10.6 基于Caffe ResNet-50网络实现图片分类（视频解码+同步推理）
resnet50_imagenet_classification	单击resnet50_imagenet_classification获取样例	基于Caffe ResNet-50网络实现图片分类（同步推理）	10.7 基于Caffe ResNet-50网络实现图片分类（同步推理）
resnet50_async_imagenet_classification	单击resnet50_async_imagenet_classification获取样例	基于Caffe ResNet-50网络实现图片分类（异步推理）	10.8 基于Caffe ResNet-50网络实现图片分类（异步推理）

Sample名称	Sample获取	基本功能	编译运行指导（Ctrl CPU开放形态）
batchcrop	单击**batchcrop**获取样例	媒体数据处理V1（抠图，一图多框）	**10.9 媒体数据处理V1（抠图，一图多框）**
venc_image	单击**venc_image**获取样例	媒体数据处理V1（视频编码）	**10.10 媒体数据处理V1（视频编码）**
smallResolution_cropandpaste	单击**smallResolution_cropandpaste**获取样例	媒体数据处理V1（抠图贴图）	**10.11 媒体数据处理V1（抠图贴图）**
YOLOV3_dynamic_batch_detection_picture	单击**YOLOV3_dynamic_batch_detection_picture**获取样例	基于Caffe YOLOv3网络实现目标检测（动态Batch/动态分辨率）	**10.12 基于Caffe YOLOv3网络实现目标检测（动态Batch/动态分辨率）**
vpc_sample	单击**vpc_sample**获取样例	媒体数据处理V2（VPC抠图\贴图\缩放等）	请参见样例工程中的README
jpegd_sample	单击**jpegd_sample**获取样例	媒体数据处理V2（JPEG图片解码）	请参见样例工程中的README
jpege_sample	单击**jpege_sample**获取样例	媒体数据处理V2（JPEG图片编码）	请参见样例工程中的README
vdec_sample	单击**vdec_sample**获取样例	媒体数据处理V2（VDEC视频解码）	请参见样例工程中的README
venc_sample	单击**venc_sample**获取样例	媒体数据处理V2（VENC视频编码）	请参见样例工程中的README
pngd_sample	单击**pngd_sample**获取样例	媒体数据处理V2（PNGD图片解码）	请参见样例工程中的README

10.3 实现矩阵-矩阵乘运算

10.3.1 样例介绍

获取样例

单击**gemm**获取样例

功能描述

该样例主要实现矩阵-矩阵相乘的运算：C = αAB + βC，A、B、C都是16*16的矩阵，即：m=16，n=16，k=16。矩阵乘的结果是一个16*16的矩阵。

图 10-1 Sample 示例

$$\text{矩阵 A}=\begin{pmatrix} a11 & a12 & a13 & \dots & a1k \\ a21 & a22 & a23 & \dots & a2k \\ a31 & a32 & a33 & \dots & a3k \\ \dots & \dots & \dots & & \dots \\ am1 & am2 & am3 & \dots & amk \end{pmatrix}, B=\begin{pmatrix} b11 & b12 & b13 & \dots & b1n \\ b21 & b22 & b23 & \dots & b2n \\ b31 & b32 & b33 & \dots & b3n \\ \dots & \dots & \dots & & \dots \\ bk1 & bk2 & bk3 & \dots & bkn \end{pmatrix}, C=\begin{pmatrix} C11 & C12 & C13 & \dots & C1n \\ C21 & C22 & C23 & \dots & C2n \\ C31 & C32 & C33 & \dots & C3n \\ \dots & \dots & \dots & & \dots \\ Cm1 & Cm2 & Cm3 & \dots & Cmn \end{pmatrix}$$

原理介绍

在该样例中，涉及的关键功能点，如下表所示。

表 10-3 关键功能点

初始化	● 调用aclInit接口初始化AscendCL配置。 ● 调用aclFinalize接口实现AscendCL去初始化。
Device管理	● 调用aclrtSetDevice接口指定用于运算的Device。 ● 调用aclrtGetRunMode接口获取软件栈的运行模式，根据运行模式的不同，内部处理流程不同。 ● 调用aclrtResetDevice接口复位当前运算的Device，回收Device上的资源。
Stream管理	● 调用aclrtCreateStream接口创建Stream。 ● 调用aclrtDestroyStream接口销毁Stream。 ● 调用aclrtSynchronizeStream接口阻塞程序运行，直到指定Stream中的所有任务都完成。
内存管理	● 调用aclrtMallocHost接口申请Host上内存。 ● 调用aclrtFreeHost释放Host上的内存。 ● 调用aclrtMalloc接口申请Device上的内存。 ● 调用aclrtFree接口释放Device上的内存。
数据传输	如果在Device上运行应用，则无需进行数据传输。

单算子调用	• 调用aclblasGemmEx接口实现矩阵-矩阵相乘的运算，由用户指定矩阵中元素的数据类型。在aclblasGemmEx接口内部封装了系统内置的矩阵乘算子GEMM。 • 使用ATC（Ascend Tensor Compiler）工具将内置的矩阵乘算子GEMM的算子描述信息（包括输入输出Tensor描述、算子属性等）编译成适配昇腾AI处理器的离线模型（*.om文件），用于验证矩阵乘算子GEMM的运行结果。

目录结构

样例代码结构如下所示。

```
├── inc
│   ├── common.h                //定义公共函数（例如：文件读取函数）的头文件
│   ├── gemm_runner.h             //定义矩阵乘运算相关函数的头文件

├── run
│   ├── out
│   │   ├──test_data
│   │   │   ├── config
│   │   │   │   ├── acl.json          //系统初始化的配置文件
│   │   │   │   ├── gemm.json           //矩阵乘算子的算子描述信息
│   │   │   ├── data
│   │   │   │   ├── generate_data.py   //用于生成矩阵A、矩阵B的数据

├── src
│   ├── CMakeLists.txt            //编译脚本
│   ├── common.cpp                //公共函数（例如：文件读取函数）的实现文件
│   ├── gemm_main.cpp             //主函数的实现文件
│   ├── gemm_runner.cpp           //执行矩阵乘运算相关函数的实现文件
```

10.3.2 Ascend EP，编译及在 Device 上运行应用（Atlas 200/300/500 推理产品）（Atlas 推理系列产品）

步骤1 模型转换。

1. 以运行用户登录开发环境。
2. 参考《**ATC工具使用指南（开放态）**》，准备环境，包括获取工具、设置环境变量。
3. 将矩阵乘算子的算子描述信息（*.json文件）编译成适配昇腾AI处理器的离线模型（*.om文件），运行矩阵乘算子时使用。

   切换到gemm样例目录，执行如下命令：

   ```
   atc --singleop=run/out/test_data/config/gemm.json --soc_version=${soc_version} --output=run/out/op_models
   ```

   – --soc_version：昇腾AI处理器的版本，请根据实际情况替换。

     可从ATC安装路径的“atc/data/platform_config”目录下查看昇腾AI处理器的版本，对应“*.ini”文件的名字（不带.ini后缀）即为soc_version。

   – --output：生成的om文件必须放在“run/out/op_models”目录下。

   关于各参数的详细解释，请参见《**ATC工具使用指南（开放态）**》。

步骤2 以运行用户登录开发环境，并切换到gemm样例所在的目录。

步骤3 编译代码。

1. 设置环境变量，编译脚本src/CMakeLists.txt通过环境变量所设置的头文件、库文件的路径来编译代码。

   如下为示例，$HOME/AscendMiniOs表示Runtime开放形态安装包的安装路径，请根据实际情况替换。

   ```
   export DDK_PATH=$HOME/AscendMiniOs
   export NPU_HOST_LIB=$HOME/AscendMiniOs/runtime/lib64/stub
   ```

   说明

   使用“$HOME/AscendMiniOs/runtime/lib64/stub”目录下的*.so库，是为了编译基于AscendCL接口的代码逻辑时，不依赖其它组件（例如Driver）的任何*.so库。

   编译通过后，在Device上运行应用时，应用会链接到Device上“Runtime开放形态安装包的安装路径/runtime/lib64”目录下的*.so库，运行时会自动链接到依赖其它组件的*.so库。

2. 在样例目录下，创建目录用于存放编译文件，例如，本文中，创建的目录为“build/intermediates/device”。

   ```
   mkdir -p build/intermediates/device
   ```

3. 切换到“build/intermediates/device”目录，执行**cmake**生成编译文件。
   - “../../../src”表示CMakeLists.txt文件所在的目录，请根据实际目录层级修改。
   - 在-DCMAKE_CXX_COMPILER参数处设置交叉编译器的路径，“$HOME/Ascend/ascend-toolkit/latest”表示Toolkit组件的默认安装路径，请根据实际情况替换。

   ```
   cd build/intermediates/device
   cmake ../../../src -DCMAKE_CXX_COMPILER=$HOME/Ascend/ascend-toolkit/latest/toolkit/
   toolchain/hcc/bin/aarch64-target-linux-gnu-g++ -DCMAKE_SKIP_RPATH=TRUE
   ```

4. 执行**make**命令，生成的可执行文件main在“样例目录/out”目录下。

   ```
   make
   ```

步骤4 准备测试数据。

1. 以运行用户登录开发环境。
2. 切换到“gemm样例目录/run/out/test_data/data”目录下，执行generate_data.py脚本，在“run/out/test_data/data”目录下生成矩阵A的数据文件matrix_a.bin、矩阵B的数据文件matrix_b.bin、矩阵C的数据文件matrix_c.bin。

   ```
   python3 generate_data.py
   ```

   说明

   “run/out/test_data/data”目录下生成的output.bin文件中的数据是generate_data.py脚本中直接通过公式计算出来的矩阵乘结果，不作为该样例的输入数据。

步骤5 运行应用。

1. 以运行用户将开发环境的gemm样例目录及目录下的文件上传到运行环境（Device），例如“$HOME/acl_execute_gemm”。
2. 以运行用户登录运行环境（Device）。
3. 切换到可执行文件execute_gemm_op所在的目录，例如“acl_execute_gemm/run/out”，给该目录下的execute_gemm_op文件加执行权限。

   ```
   chmod +x execute_gemm_op
   ```

4. 切换到可执行文件execute_gemm_op所在的目录，例如“acl_execute_gemm/run/out”，运行可执行文件。

   ```
   ./execute_gemm_op
   ```

执行成功后，会直接在终端窗口显示矩阵A的数据、矩阵B的数据以及矩阵乘的结果，同时在“result_files”目录下，生成存放矩阵乘结果的matrix_c.bin文件。

----结束

10.4 基于 Caffe ResNet-50 网络实现图片分类（图片解码+缩放+同步推理）

10.4.1 样例介绍

获取样例

单击**vpc_resnet50_imagenet_classification**获取样例

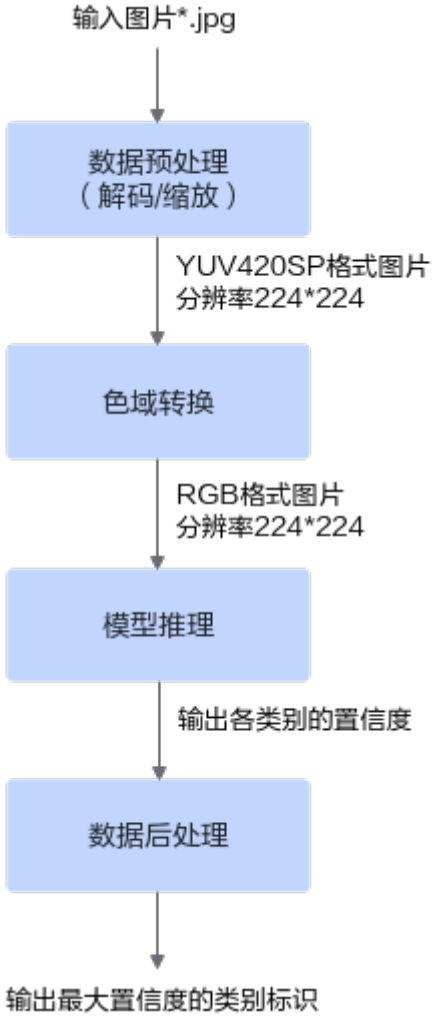
功能描述

该样例主要是基于Caffe ResNet-50网络（单输入、单Batch）实现图片分类的功能。

将Caffe ResNet-50网络的模型文件转换为适配昇腾AI处理器的离线模型（*.om文件），在样例中，加载该om文件，对2张*.jpg图片进行解码、缩放、推理，分别得到推理结果后，再对推理结果进行处理，输出最大置信度的类别标识。

转换模型时，需配置色域转换参数，用于将YUV420SP格式的图片转换为RGB格式的图片，才能符合模型的输入要求。

图 10-2 Sample 示例



原理介绍

在该样例中，涉及的关键功能点，如下表所示。

初始化	• 调用aclInit接口初始化AscendCL配置。 • 调用aclFinalize接口实现AscendCL去初始化。
Device管理	• 调用aclrtSetDevice接口指定用于运算的Device。 • 调用aclrtGetRunMode接口获取软件栈的运行模式，根据运行模式的不同，内部处理流程不同。 • 调用aclrtResetDevice接口复位当前运算的Device，回收Device上的资源。
Context管理	• 调用aclrtCreateContext接口创建Context。 • 调用aclrtDestroyContext接口销毁Context。

Stream管理	● 调用aclrtCreateStream接口创建Stream。 ● 调用aclrtDestroyStream接口销毁Stream。 ● 调用aclrtSynchronizeStream接口阻塞程序运行，直到指定Stream中的所有任务都完成。
内存管理	● 调用aclrtMallocHost接口申请Host上内存。 ● 调用aclrtFreeHost释放Host上的内存。 ● 调用aclrtMalloc接口申请Device上的内存。 ● 调用aclrtFree接口释放Device上的内存。 执行媒体数据处理时，若需要申请Device上的内存存放输入或输出数据，需调用acldvppMalloc申请内存、调用acldvppFree接口释放内存。
数据传输	如果在Device上运行应用，则无需进行数据传输。
媒体数据处理V1	● 调用acldvppJpegDecodeAsync接口将*.jpg图片解码成YUV420SP格式图片。 ● 调用acldvppVpcResizeAsync接口将YUV420SP格式图片缩小成分辨率为224*224的图片。
模型推理	● 调用aclmdlLoadFromFileWithMem接口从*.om文件加载模型。 ● 调用aclmdlExecute接口执行模型推理。 推理前，通过*.om文件中的色域转换参数将YUV420SP格式的图片转换为RGB格式的图片。 ● 调用aclmdlUnload接口卸载模型。
数据后处理（单算子调用）	处理模型推理的结果，通过调用算子Cast将推理结果的数据类型从float32转成float16，再调用ArgMaxD算子从推理结果中查找最大置信度的类别标识。 通过aclopSetModelDir接口加载单算子模型文件，通过aclopCast接口执行Cast算子、通过aclopExecuteV2接口执行ArgMaxD算子。

目录结构

样例代码结构如下所示。

```
├── caffe_model
│   ├── aipp.cfg        //带色域转换参数的配置文件，模型转换时使用

├── data
│   ├── dog1_1024_683.jpg          //测试数据,需要按指导获取测试图片，放到data目录下
│   ├── dog2_1024_683.jpg          //测试数据,需要按指导获取测试图片，放到data目录下

├── inc
│   ├── dvpp_process.h             //声明媒体数据处理相关函数的头文件
│   ├── model_process.h            //声明模型处理相关函数的头文件
│   ├── sample_process.h           //声明资源初始化/销毁相关函数的头文件
│   ├── singleOp_process.h         //声明单算子执行相关函数的头文件
│   ├── utils.h                    //声明公共函数（例如：文件读取函数）的头文件

├── out
│   ├── op_models
│   ├──├──op_list.json             //Cast算子和ArgMaxD算子的算子描述信息
```

```
├── src
│   ├── acl.json          //系统初始化的配置文件
│   ├── CMakeLists.txt          //编译脚本
│   ├── dvpp_process.cpp        //媒体数据处理相关函数的实现文件
│   ├── main.cpp              //主函数，图片分类功能的实现文件
│   ├── model_process.cpp       //模型处理相关函数的实现文件
│   ├── sample_process.cpp      //资源初始化/销毁相关函数的实现文件
│   ├── singleOp_process.cpp    //单算子执行相关函数的实现文件
│   ├── utils.cpp             //公共函数（例如：文件读取函数）的实现文件

├── .project     //工程信息文件，包含工程类型、工程描述、运行目标设备类型等
├── CMakeLists.txt    //编译脚本，调用src目录下的CMakeLists文件
```

10.4.2 Ascend EP，编译及在 Device 上运行应用（Atlas 200/300/500 推理产品）（Atlas 推理系列产品）

步骤1 模型转换。

1. 以运行用户登录开发环境。
2. 参考《**ATC工具使用指南（开放态）**》，准备环境，包括获取工具、设置环境变量。
3. 准备数据。

   您可以从以下链接中获取ResNet-50网络的模型文件（*.prototxt）、权重文件（*.caffemodel），并以运行用户将获取的文件上传至开发环境的“vpc_resnet50_imagenet_classification样例目录/caffe_model”目录下。如果目录不存在，需要自行创建。

   – ResNet-50网络的模型文件（*.prototxt）：单击**Link**下载该文件。
   – ResNet-50网络的权重文件（*.caffemodel）：单击**Link**下载该文件。
4. 将ResNet-50网络转换为适配昇腾AI处理器的离线模型（*.om文件），转换模型时，需配置色域转换参数，用于将YUV420SP格式的图片转换为RGB格式的图片。

   切换到vpc_resnet50_imagenet_classification样例所在的目录，执行如下命令：

   ```
   atc --model=caffe_model/resnet50.prototxt --weight=caffe_model/resnet50.caffemodel --framework=0 --output=model/resnet50_aipp --soc_version=${soc_version} --insert_op_conf=caffe_model/aipp.cfg
   ```

   – --soc_version：昇腾AI处理器的版本，请根据实际情况替换。

     可从ATC安装路径的“atc/data/platform_config”目录下查看昇腾AI处理器的版本，对应“*.ini”文件的名字（不带.ini后缀）即为soc_version。
   – --output：生成的resnet50_aipp.om文件存放在“样例目录/model”目录下。建议使用命令中的默认设置，否则在编译代码前，您还需要修改sample_process.cpp中的omModelPath参数值。

     ```
     const char* omModelPath = "../model/resnet50_aipp.om";
     ```

   关于各参数的详细解释，请参见《**ATC工具使用指南（开放态）**》。
5. 将Cast和ArgMaxD两个算子的算子描述信息（*.json文件）编译成适配昇腾AI处理器的离线模型（*.om文件），用于运行算子时使用。

   切换到vpc_resnet50_imagenet_classification样例所在的目录，执行如下命令：

   ```
   atc --singleop=out/op_models/op_list.json --soc_version=${soc_version} --output=out/op_models
   ```

   – --soc_version：昇腾AI处理器的版本，请根据实际情况替换。

     可从ATC安装路径的“atc/data/platform_config”目录下查看昇腾AI处理器的版本，对应“*.ini”文件的名字（不带.ini后缀）即为soc_version。
   – --output参数：生成的om文件必须放在“out/op_models”目录下。

   关于各参数的详细解释，请参见《**ATC工具使用指南（开放态）**》。

步骤2 以运行用户登录开发环境，并切换到vpc_resnet50_imagenet_classification样例所在的目录。

步骤3 编译代码。

1. 设置环境变量，编译脚本src/CMakeLists.txt通过环境变量所设置的头文件、库文件的路径来编译代码。

   如下为示例，$HOME/AscendMiniOs表示Runtime开放形态安装包的安装路径，请根据实际情况替换。

   ```
   export DDK_PATH=$HOME/AscendMiniOs
   export NPU_HOST_LIB=$HOME/AscendMiniOs/runtime/lib64/stub
   ```

   说明

   使用“$HOME/AscendMiniOs/runtime/lib64/stub”目录下的*.so库，是为了编译基于AscendCL接口的代码逻辑时，不依赖其它组件（例如Driver）的任何*.so库。

   编译通过后，在Device上运行应用时，应用会链接到Device上“Runtime开放形态安装包的安装路径/runtime/lib64”目录下的*.so库，运行时会自动链接到依赖其它组件的*.so库。

2. 在样例目录下，创建目录用于存放编译文件，例如，本文中，创建的目录为“build/intermediates/device”。

   ```
   mkdir -p build/intermediates/device
   ```

3. 切换到“build/intermediates/device”目录，执行**cmake**生成编译文件。
   - “../../../src”表示CMakeLists.txt文件所在的目录，请根据实际目录层级修改。
   - 在-DCMAKE_CXX_COMPILER参数处设置交叉编译器的路径，“$HOME/Ascend/ascend-toolkit/latest”表示Toolkit组件的默认安装路径，请根据实际情况替换。

   ```
   cd build/intermediates/device
   cmake ../../../src -DCMAKE_CXX_COMPILER=$HOME/Ascend/ascend-toolkit/latest/toolkit/
   toolchain/hcc/bin/aarch64-target-linux-gnu-g++ -DCMAKE_SKIP_RPATH=TRUE
   ```

4. 执行**make**命令，生成的可执行文件main在“样例目录/out”目录下。

   ```
   make
   ```

步骤4 准备输入图片。

请从以下链接获取该样例的输入图片，并以运行用户将获取的文件上传至开发环境的“vpc_resnet50_imagenet_classification样例目录/data”目录下。如果目录不存在，需自行创建。

https://obs-9be7.obs.cn-east-2.myhuaweicloud.com/models/aclsample/dog1_1024_683.jpg

https://obs-9be7.obs.cn-east-2.myhuaweicloud.com/models/aclsample/dog2_1024_683.jpg

步骤5 运行应用。

1. 以运行用户将开发环境的vpc_resnet50_imagenet_classification样例目录及目录下的文件上传到运行环境（Device），例如“$HOME/acl_dvpp_resnet50”。
2. 以运行用户登录运行环境（Device）。
3. 切换到可执行文件main所在的目录，例如“acl_dvpp_resnet50/out”，给该目录下的main文件加执行权限。

   ```
   chmod +x main
   ```

4. 切换到可执行文件main所在的目录，例如“acl_dvpp_resnet50/out”，运行可执行文件。

```
./main
```

执行成功后，在屏幕上显示最大置信度的类别标识，示例如下，类别编号161表示["basset", "basset hound"]，类别编号267表示["standard poodle"]。

```
[INFO] acl init success
[INFO] open device 0 success
[INFO] create context success
[INFO] create stream success
[INFO] dvpp init resource success
[INFO] load model ../model/resnet50_aipp.om success
[INFO] create model description success
[INFO] create model output success
[INFO] start to process picture:../data/dog1_1024_683.jpg
[INFO] Process dvpp success
[INFO] model execute success
[INFO] execute singleOp Cast success
[INFO] execute ArgMaxD success
[INFO] singleOp process success
[INFO] ---> index of classification result is 161
[INFO] start to process picture:../data/dog2_1024_683.jpg
[INFO] Process dvpp success
[INFO] model execute success
[INFO] execute singleOp Cast success
[INFO] execute ArgMaxD success
[INFO] singleOp process success
[INFO] ---> index of classification result is 267
[INFO] Unload model success, modelId is 1
[INFO] execute sample success
[INFO] end to destroy stream
[INFO] end to destroy context
[INFO] end to reset device is 0
```

----结束

10.5 基于 Caffe ResNet-50 网络实现图片分类（图片解码+抠图缩放+图片编码+同步推理）

10.5.1 样例介绍

获取样例

单击**vpc_jpeg_resnet50_imagenet_classification**获取样例

功能描述

该样例主要是基于Caffe ResNet-50网络（单输入、单Batch）实现图片分类的功能。

根据运行应用的入参，该样例可实现以下功能：

- 将一张YUV420SP格式的图片编码为*.jpg格式的图片。
- 将两张*.jpg格式的解码成两张YUV420SP NV12格式的图片，缩放，再进行模型推理，分别得到两张图片的推理结果后，处理推理结果，输出最大置信度的类别标识以及top5置信度的总和。
- 将两张*.jpg格式的解码成两张YUV420SP NV12格式的图片，抠图，再进行模型推理，分别得到两张图片的推理结果后，处理推理结果，输出最大置信度的类别标识以及top5置信度的总和。

- 将两张*.jpg格式的解码成两张YUV420SP NV12格式的图片，抠图贴图，再进行模型推理，分别得到两张图片的推理结果后，处理推理结果，输出最大置信度的类别标识以及top5置信度的总和。
- 将YUV420SP NV12格式的图片（分辨率8192*8192）缩放，得到4000*4000。

该样例中用于推理的模型文件是*.om文件（适配昇腾AI处理器的离线模型），转换模型时，需配置色域转换参数，用于将YUV420SP格式的图片转换为RGB格式的图片，才能符合模型的输入要求。

原理介绍

在该样例中，涉及的关键功能点，如下表所示。

初始化	● 调用aclInit接口初始化AscendCL配置。 ● 调用aclFinalize接口实现AscendCL去初始化。
Device管理	● 调用aclrtSetDevice接口指定用于运算的Device。 ● 调用aclrtGetRunMode接口获取软件栈的运行模式，根据运行模式的不同，内部处理流程不同。 ● 调用aclrtResetDevice接口复位当前运算的Device，回收Device上的资源。
Context管理	● 调用aclrtCreateContext接口创建Context。 ● 调用aclrtDestroyContext接口销毁Context。
Stream管理	● 调用aclrtCreateStream接口创建Stream。 ● 调用aclrtDestroyStream接口销毁Stream。 ● 调用aclrtSynchronizeStream接口阻塞程序运行，直到指定Stream中的所有任务都完成。
内存管理	● 调用aclrtMallocHost接口申请Host上内存。 ● 调用aclrtFreeHost释放Host上的内存。 ● 调用aclrtMalloc接口申请Device上的内存。 ● 调用aclrtFree接口释放Device上的内存。 执行媒体数据处理时，若需要申请Device上的内存存放输入或输出数据，需调用acldvppMalloc申请内存、调用acldvppFree接口释放内存。
数据传输	如果在Device上运行应用，则无需进行数据传输。

媒体数据处理V1	• 图片编码 调用acldvppJpegEncodeAsync接口将YUV420SP格式的图片编码为*.jpg格式的图片。 • 图片解码 调用acldvppJpegDecodeAsync接口将*.jpg图片解码成YUV420SP格式图片。 • 缩放 调用acldvppVpcResizeAsync接口对YUV420SP格式的输入图片进行缩放。 • 抠图 调用acldvppVpcCropResizeAsync接口按指定区域从输入图片中抠图，再将抠的图片存放到输出内存中，作为输出图片。 • 抠图贴图 调用acldvppVpcCropResizePasteAsync接口按指定区域从输入图片中抠图，再将抠的图片贴到目标图片的指定位置，作为输出图片。
模型推理	• 调用aclmdlLoadFromFileWithMem接口从*.om文件加载模型。 • 调用aclmdlExecute接口执行模型推理。 推理前，通过*.om文件中的色域转换参数将YUV420SP格式的图片转换为RGB格式的图片。 • 调用aclmdlUnload接口卸载模型。

目录结构

样例代码结构如下所示。

```
├── caffe_model
│   ├── aipp.cfg        //带色域转换参数的配置文件，模型转换时使用

├── data
│   ├── persian_cat_1024_1536_283.jpg          //测试数据,需要按指导获取测试图片，放到data目录下
│   ├── wood_rabbit_1024_1061_330.jpg          //测试数据,需要按指导获取测试图片，放到data目录下
│   ├── wood_rabbit_1024_1068_nv12.yuv         //测试数据,需要按指导获取测试图片，放到data目录下
│   ├── dvpp_vpc_8192x8192_nv12.yuv            //测试数据,需要按指导获取测试图片，放到data目录下

├── inc
│   ├── dvpp_process.h          //声明媒体数据处理相关函数的头文件
│   ├── model_process.h         //声明模型处理相关函数的头文件
│   ├── sample_process.h        //声明资源初始化/销毁相关函数的头文件
│   ├── utils.h                 //声明公共函数（例如：文件读取函数）的头文件

├── src
│   ├── acl.json        //系统初始化的配置文件
│   ├── CMakeLists.txt        //编译脚本
│   ├── dvpp_process.cpp        //媒体数据处理相关函数的实现文件
│   ├── main.cpp        //主函数，图片分类功能的实现文件
│   ├── model_process.cpp        //模型处理相关函数的实现文件
│   ├── sample_process.cpp        //资源初始化/销毁相关函数的实现文件
│   ├── utils.cpp        //公共函数（例如：文件读取函数）的实现文件

├── .project        //工程信息文件，包含工程类型、工程描述、运行目标设备类型等
├── CMakeLists.txt        //编译脚本，调用src目录下的CMakeLists文件
```

10.5.2 Ascend EP，编译及在 Device 上运行应用（Atlas 200/300/500 推理产品）（Atlas 推理系列产品）

步骤1 模型转换。

1. 以运行用户登录开发环境。
2. 参考《**ATC工具使用指南（开放态）**》，准备环境，包括获取工具、设置环境变量。
3. 准备数据。

   您可以从以下链接中获取ResNet-50网络的模型文件（*.prototxt）、权重文件（*.caffemodel），并以运行用户将获取的文件上传至开发环境的“vpc_jpeg_resnet50_imagenet_classification样例目录/caffe_model”目录下。如果目录不存在，需要自行创建。

   – ResNet-50网络的模型文件（*.prototxt）：单击**Link**下载该文件。
   – ResNet-50网络的权重文件（*.caffemodel）：单击**Link**下载该文件。

4. 将ResNet-50网络转换为适配昇腾AI处理器的离线模型（*.om文件），转换模型时，需配置色域转换参数，用于将YUV420SP格式的图片转换为RGB格式的图片。

   切换到vpc_jpeg_resnet50_imagenet_classification样例所在目录，执行如下命令：

   ```
   atc --model=caffe_model/resnet50.prototxt --weight=caffe_model/resnet50.caffemodel --framework=0 --output=model/resnet50_aipp --soc_version=${soc_version} --insert_op_conf=caffe_model/aipp.cfg
   ```

   – --soc_version：昇腾AI处理器的版本，请根据实际情况替换。

     可从ATC安装路径的“atc/data/platform_config”目录下查看昇腾AI处理器的版本，对应“*.ini”文件的名字（不带.ini后缀）即为soc_version。

   – --output：生成的resnet50_aipp.om文件存放在“样例目录/model”目录下。建议使用命令中的默认设置，否则在编译代码前，您还需要修改sample_process.cpp中的omModelPath参数值。

     ```
     const char* omModelPath = "../model/resnet50_aipp.om";
     ```

   关于各参数的详细解释，请参见《**ATC工具使用指南（开放态）**》。

步骤2 以运行用户登录开发环境，并切换到vpc_jpeg_resnet50_imagenet_classification样例所在的目录。

步骤3 编译代码。

1. 设置环境变量，编译脚本src/CMakeLists.txt通过环境变量所设置的头文件、库文件的路径来编译代码。

   如下为示例，$HOME/AscendMiniOs表示Runtime开放形态安装包的安装路径，请根据实际情况替换。

   ```
   export DDK_PATH=$HOME/AscendMiniOs
   export NPU_HOST_LIB=$HOME/AscendMiniOs/runtime/lib64/stub
   ```

   说明

   使用“$HOME/AscendMiniOs/runtime/lib64/stub”目录下的*.so库，是为了编译基于AscendCL接口的代码逻辑时，不依赖其它组件（例如Driver）的任何*.so库。

   编译通过后，在Device上运行应用时，应用会链接到Device上“Runtime开放形态安装包的安装路径/runtime/lib64”目录下的*.so库，运行时会自动链接到依赖其它组件的*.so库。

2. 在样例目录下，创建目录用于存放编译文件，例如，本文中，创建的目录为“build/intermediates/device”。

```
mkdir -p build/intermediates/device
```

3. 切换到“build/intermediates/device”目录，执行**cmake**生成编译文件。
   - “../../../src”表示CMakeLists.txt文件所在的目录，请根据实际目录层级修改。
   - 在-DCMAKE_CXX_COMPILER参数处设置交叉编译器的路径，“$HOME/Ascend/ascend-toolkit/latest”表示Toolkit组件的默认安装路径，请根据实际情况替换。

```
cd build/intermediates/device
cmake ../../../src -DCMAKE_CXX_COMPILER=$HOME/Ascend/ascend-toolkit/latest/toolkit/toolchain/hcc/bin/aarch64-target-linux-gnu-g++ -DCMAKE_SKIP_RPATH=TRUE
```

4. 执行**make**命令，生成的可执行文件main在“样例目录/out”目录下。

```
make
```

步骤4 准备输入图片。

请从以下链接获取该样例的输入图片，并以运行用户将获取的文件上传至开发环境的“vpc_jpeg_resnet50_imagenet_classification样例目录/data”目录下。如果目录不存在，需自行创建。

https://obs-9be7.obs.cn-east-2.myhuaweicloud.com/models/aclsample/dvpp_vpc_8192x8192_nv12.yuv

https://obs-9be7.obs.cn-east-2.myhuaweicloud.com/models/aclsample/persian_cat_1024_1536_283.jpg

https://obs-9be7.obs.cn-east-2.myhuaweicloud.com/models/aclsample/wood_rabbit_1024_1061_330.jpg

https://obs-9be7.obs.cn-east-2.myhuaweicloud.com/models/aclsample/wood_rabbit_1024_1068_nv12.yuv

步骤5 运行应用。

1. 以运行用户将开发环境的vpc_jpeg_resnet50_imagenet_classification样例目录及目录下的文件上传到运行环境（Device），例如“$HOME/acl_vpc_jpege_resnet50”。
2. 以运行用户登录运行环境（Device）。
3. 切换到可执行文件main所在的目录，例如“acl_vpc_jpege_resnet50/out”，给该目录下的main文件加执行权限。

```
chmod +x main
```

4. 切换到可执行文件main所在的目录，例如“acl_vpc_jpege_resnet50/out”，运行可执行文件。
   a. 将两张*.jpg格式的解码成两张YUV420SP NV12格式的图片，缩放，再进行模型推理，分别得到两张图片的推理结果。

```
./main 0
```

   执行结果示例如下，提示信息中的classType表示类别标识、top1表示该分类的最大置信度、top5表示最大的5个置信度之和，这些值可能会根据版本、环境有所不同，请以实际情况为准（类别编号283表示["Persian cat"]，类别编号330表示["wood rabbit", "cottontail", "cottontail rabbit"]）：

```
[INFO] acl init success
[INFO] open device 0 success
[INFO] create context success
[INFO] create stream success
[INFO] dvpp init resource success
[INFO] load model ../model/resnet50_aipp.om success
[INFO] create model description success
```

```
[INFO] create model output success
[INFO]--------------------------------------------
[INFO] start to process picture:../data/persian_cat_1024_1536_283.jpg
[INFO] call JpegD
[INFO] call vpcResize
[INFO] Process dvpp success
[INFO] model execute success
[INFO] result : classType[283], top1[xxxxxxx], top5[xxxxxx]
[INFO]--------------------------------------------
[INFO] start to process picture:../data/wood_rabbit_1024_1061_330.jpg
[INFO] call JpegD
[INFO] call vpcResize
[INFO] Process dvpp success
[INFO] model execute success
[INFO] result : classType[330], top1[xxxxxxx], top5[xxxxxx]
[INFO]--------------------------------------------
[INFO] Unload model success, modelId is 1
[INFO] execute sample success
[INFO] end to destroy stream
[INFO] end to destroy context
[INFO] end to reset device is 0
```

b. 将两张*.jpg格式的解码成两张YUV420SP NV12格式的图片，抠图，再进行模型推理，分别得到两张图片的推理结果。

```
./main 1
```

执行结果示例如下，提示信息中的classType表示类别标识、top1表示该分类的最大置信度、top5表示最大的5个置信度之和，这些值可能会根据版本、环境有所不同，请以实际情况为准（类别编号284表示["Siamese cat", "Siamese"]，类别编号330表示["wood rabbit", "cottontail", "cottontail rabbit"]）：

```
[INFO] acl init success
[INFO] open device 0 success
[INFO] create context success
[INFO] create stream success
[INFO] dvpp init resource success
[INFO] load model ../model/resnet50_aipp.om success
[INFO] create model description success
[INFO] create model output success
[INFO]--------------------------------------------
[INFO] start to process picture:../data/persian_cat_1024_1536_283.jpg
[INFO] call JpegD
[INFO] call vpcCrop
[INFO] Process dvpp success
[INFO] model execute success
[INFO] result : classType[284], top1[xxxxxx], top5[xxxxxx]
[INFO]--------------------------------------------
[INFO] start to process picture:../data/wood_rabbit_1024_1061_330.jpg
[INFO] call JpegD
[INFO] call vpcCrop
[INFO] Process dvpp success
[INFO] model execute success
[INFO] result : classType[330], top1[xxxxxx], top5[xxxxxx]
[INFO]--------------------------------------------
[INFO] Unload model success, modelId is 1
[INFO] execute sample success
[INFO] end to destroy stream
[INFO] end to destroy context
[INFO] end to reset device is 0
```

c. 将两张*.jpg格式的解码成两张YUV420SP NV12格式的图片，抠图贴图，再进行模型推理，分别得到两张图片的推理结果。

```
./main 2
```

执行结果示例如下，提示信息中的classType表示类别标识、top1表示该分类的最大置信度、top5表示最大的5个置信度之和，这些值可能会根据版本、环境有所不同，请以实际情况为准（类别编号284表示["Siamese cat", "Siamese"]，类别编号331表示["hare"]）：

```
[INFO] acl init success
[INFO] open device 0 success
[INFO] create context success
[INFO] create stream success
[INFO] dvpp init resource success
[INFO] load model ../model/resnet50_aipp.om success
[INFO] create model description success
[INFO] create model output success
[INFO]--------------------------------------------
[INFO] start to process picture:../data/persian_cat_1024_1536_283.jpg
[INFO] call JpegD
[INFO] call vpcCropAndPaste
[INFO] Process dvpp success
[INFO] model execute success
[INFO] result : classType[284], top1[xxxxxx], top5[xxxxxx]
[INFO]--------------------------------------------
[INFO] start to process picture:../data/wood_rabbit_1024_1061_330.jpg
[INFO] call JpegD
[INFO] call vpcCropAndPaste
[INFO] Process dvpp success
[INFO] model execute success
[INFO] result : classType[331], top1[xxxxxx], top5[xxxxxx]
[INFO]--------------------------------------------
[INFO] Unload model success, modelId is 1
[INFO] execute sample success
[INFO] end to destroy stream
[INFO] end to destroy context
[INFO] end to reset device is 0
```

d. 将一张YUV420SP格式的图片编码为*.jpg格式的图片。

```
./main 3
```

执行结果示例如下：

```
[INFO] acl init success
[INFO] open device 0 success
[INFO] create context success
[INFO] create stream success
[INFO] dvpp init resource success
[INFO] start to process picture:../data/wood_rabbit_1024_1068_nv12.yuv
[INFO] call JpegE
[INFO] end to destroy stream
[INFO] end to destroy context
[INFO] end to reset device is 0
```

e. 将一张分辨率为8192*8192的YUV420SP格式的图片缩放至4000*4000。

```
./main 4
```

执行结果示例如下：

```
[INFO] acl init success
[INFO] open device 0 success
[INFO] create context success
[INFO] create stream success
[INFO] get run mode success
[INFO] dvpp process 8k resize begin
[INFO] dvpp init resource success
[INFO] dvpp process 8k resize success
[INFO] end to destroy stream
[INFO] end to destroy context
[INFO] end to reset device is 0
[INFO] end to finalize acl
```

说明

执行可执行文件成功后，同时会在main文件同级的result目录下生成结果文件，便于后期查看。结果文件如下：

- dvpp_output_0：persian_cat_1024_1536_283.jpg图片经过缩放或抠图或抠图贴图之后的结果图片。
- dvpp_output_1：wood_rabbit_1024_1061_330.jpg图片经过缩放或抠图或抠图贴图之后的结果图片。
- model_output_0：persian_cat_1024_1536_283.jpg图片的模型推理结果，二进制文件。
- model_output_0.txt：persian_cat_1024_1536_283.jpg图片的模型推理结果，txt文件。
- model_output_1：wood_rabbit_1024_1061_330.jpg图片的模型推理结果，二进制文件。
- model_output_1.txt：wood_rabbit_1024_1061_330.jpg图片的模型推理结果，txt文件。
- jpege_output_0.jpg：wood_rabbit_1024_1068_nv12.yuv图片结果编码后的结果图片。
- dvpp_vpc_4000x4000_nv12.yuv：dvpp_vpc_8192x8192_nv12.yuv图片缩放后的结果图片。

----结束

10.6 基于 Caffe ResNet-50 网络实现图片分类（视频解码+同步推理）

10.6.1 样例介绍

获取样例

单击**vdec_resnet50_classification**获取样例

功能描述

该样例主要是基于Caffe ResNet-50网络（单输入、单Batch）实现图片分类的功能。

将Caffe ResNet-50网络的模型文件转换为适配昇腾AI处理器的离线模型（*.om文件），在样例中，加载该om文件，将1个*.h265格式的视频码流（仅包含一帧）循环10次解码出10张YUV420SP NV12格式的图片，对该10张图片做缩放，再对10张YUV420SP NV12格式的图片进行推理，分别得到推理结果后，再对推理结果进行处理，输出最大置信度的类别标识以及top5置信度的总和。

转换模型时，需配置色域转换参数，用于将YUV420SP格式的图片转换为RGB格式的图片，才能符合模型的输入要求。

原理介绍

在该样例中，涉及的关键功能点，如下表所示。

初始化	● 调用aclInit接口初始化AscendCL配置。 ● 调用aclFinalize接口实现AscendCL去初始化。
Device管理	● 调用aclrtSetDevice接口指定用于运算的Device。 ● 调用aclrtGetRunMode接口获取软件栈的运行模式，根据运行模式的不同，内部处理流程不同。 ● 调用aclrtResetDevice接口复位当前运算的Device，回收Device上的资源。
Context管理	● 调用aclrtCreateContext接口创建Context。 ● 调用aclrtDestroyContext接口销毁Context。
Stream管理	● 调用aclrtCreateStream接口创建Stream。 ● 调用aclrtDestroyStream接口销毁Stream。 ● 调用aclrtSynchronizeStream接口阻塞程序运行，直到指定Stream中的所有任务都完成。
内存管理	调用aclrtMallocHost接口申请Host上内存。 ● 调用aclrtFreeHost释放Host上的内存。 ● 调用aclrtMalloc接口申请Device上的内存。 ● 调用aclrtFree接口释放Device上的内存。 执行媒体数据处理时，若需要申请Device上的内存存放输入或输出数据，需调用acldvppMalloc申请内存、调用acldvppFree接口释放内存。
数据传输	如果在Device上运行应用，则无需进行数据传输。
媒体数据处理V1	● 视频解码 调用aclvdecSendFrame接口将视频码流解码成YUV420SP格式图片。 ● 缩放 调用acldvppVpcResizeAsync接口将YUV420SP NV12格式图片缩小成分辨率为224*224的图片。
模型推理	● 调用aclmdlLoadFromFileWithMem接口从*.om文件加载模型。 ● 调用aclmdlExecute接口执行模型推理。 推理前，通过*.om文件中的色域转换参数将YUV420SP格式的图片转换为RGB格式的图片。 ● 调用aclmdlUnload接口卸载模型。

目录结构

样例代码结构如下所示。

```
├── caffe_model
│   ├── aipp.cfg        //带色域转换参数的配置文件，模型转换时使用

├── data
│   ├── vdec_h265_1frame_rabbit_1280x720.h265          //测试数据,需要按指导获取测试图片，放到data目录下
```

```
├── inc
│   ├── dvpp_process.h          //声明媒体数据处理相关函数的头文件
│   ├── model_process.h         //声明模型处理相关函数的头文件
│   ├── sample_process.h        //声明资源初始化/销毁相关函数的头文件
│   ├── utils.h                 //声明公共函数（例如：文件读取函数）的头文件
│   ├── vdec_process.h          //声明视频处理函数的头文件

├── src
│   ├── acl.json          //系统初始化的配置文件
│   ├── CMakeLists.txt    //编译脚本
│   ├── dvpp_process.cpp  //媒体数据处理相关函数的实现文件
│   ├── main.cpp          //主函数，图片分类功能的实现文件
│   ├── model_process.cpp //模型处理相关函数的实现文件
│   ├── sample_process.cpp //资源初始化/销毁相关函数的实现文件
│   ├── utils.cpp         //公共函数（例如：文件读取函数）的实现文件
│   ├── vdec_process.cpp  //声明视频处理函数的实现文件

├── .project      //工程信息文件，包含工程类型、工程描述、运行目标设备类型等
├── CMakeLists.txt  //编译脚本，调用src目录下的CMakeLists文件
```

10.6.2 Ascend EP，编译及在 Device 上运行应用（Atlas 200/300/500 推理产品）（Atlas 推理系列产品）

步骤1 模型转换。

1. 以运行用户登录开发环境。
2. 参考《**ATC工具使用指南（开放态）**》，准备环境，包括获取工具、设置环境变量。
3. 准备数据。

   您可以从以下链接中获取ResNet-50网络的模型文件（*.prototxt）、权重文件（*.caffemodel），并以运行用户将获取的文件上传至开发环境的“vdec_resnet50_classification样例目录/caffe_model”目录下。如果目录不存在，需要自行创建。

   – ResNet-50网络的模型文件（*.prototxt）：单击**Link**下载该文件。
   – ResNet-50网络的权重文件（*.caffemodel）：单击**Link**下载该文件。
4. 将ResNet-50网络转换为适配昇腾AI处理器的离线模型（*.om文件），转换模型时，需配置色域转换参数，用于将YUV420SP格式的图片转换为RGB格式的图片。

   切换到vdec_resnet50_classification样例目录，执行如下命令：

   ```
   atc --model=caffe_model/resnet50.prototxt --weight=caffe_model/resnet50.caffemodel --framework=0 --output=model/resnet50_aipp --soc_version=${soc_version} --insert_op_conf=caffe_model/aipp.cfg
   ```

   – --soc_version：昇腾AI处理器的版本，请根据实际情况替换。

     可从ATC安装路径的“atc/data/platform_config”目录下查看昇腾AI处理器的版本，对应“*.ini”文件的名字（不带.ini后缀）即为soc_version。
   – --output：生成的resnet50_aipp.om文件存放在“样例目录/model”目录下。建议使用命令中的默认设置，否则在编译代码前，您还需要修改sample_process.cpp中的omModelPath参数值。

     ```
     const char* omModelPath = "../model/resnet50_aipp.om";
     ```

   关于各参数的详细解释，请参见《**ATC工具使用指南（开放态）**》。

步骤2 以运行用户登录开发环境，并切换到vdec_resnet50_classification样例所在的目录。

步骤3 编译代码。

1. 设置环境变量，编译脚本src/CMakeLists.txt通过环境变量所设置的头文件、库文件的路径来编译代码。

如下为示例，$HOME/AscendMiniOs表示Runtime开放形态安装包的安装路径，请根据实际情况替换。

```
export DDK_PATH=$HOME/AscendMiniOs
export NPU_HOST_LIB=$HOME/AscendMiniOs/runtime/lib64/stub
```

说明

使用“$HOME/AscendMiniOs/runtime/lib64/stub”目录下的*.so库，是为了编译基于AscendCL接口的代码逻辑时，不依赖其它组件（例如Driver）的任何*.so库。

编译通过后，在Device上运行应用时，应用会链接到Device上“Runtime开放形态安装包的安装路径/runtime/lib64”目录下的*.so库，运行时会自动链接到依赖其它组件的*.so库。

2. 在样例目录下，创建目录用于存放编译文件，例如，本文中，创建的目录为“build/intermediates/device”。

```
mkdir -p build/intermediates/device
```

3. 切换到“build/intermediates/device”目录，执行**cmake**生成编译文件。
   - “../../../src”表示CMakeLists.txt文件所在的目录，请根据实际目录层级修改。
   - 在-DCMAKE_CXX_COMPILER参数处设置交叉编译器的路径，“$HOME/Ascend/ascend-toolkit/latest”表示Toolkit组件的默认安装路径，请根据实际情况替换。

```
cd build/intermediates/device
cmake ../../../src -DCMAKE_CXX_COMPILER=$HOME/Ascend/ascend-toolkit/latest/toolkit/
toolchain/hcc/bin/aarch64-target-linux-gnu-g++ -DCMAKE_SKIP_RPATH=TRUE
```

4. 执行**make**命令，生成的可执行文件main在“样例目录/out”目录下。

```
make
```

步骤4 准备输入视频码流。

从以下链接获取输入视频码流文件，并以运行用户上传至开发环境的“vdec_resnet50_classification样例目录/data”目录下。如果目录不存在，需自行创建。

https://obs-9be7.obs.cn-east-2.myhuaweicloud.com/models/aclsample/vdec_h265_1frame_rabbit_1280x720.h265

步骤5 运行应用。

1. 以运行用户将开发环境的vdec_resnet50_classification样例目录及目录下的文件上传到运行环境（Device），例如“$HOME/acl_vdec_resnet50”。
2. 以运行用户登录运行环境（Device）。
3. 切换到可执行文件main所在的目录，例如“acl_vdec_resnet50/out”，给该目录下的main文件加执行权限。

```
chmod +x main
```

4. 切换到可执行文件main所在的目录，例如“acl_vdec_resnet50/out”，运行可执行文件。

```
./main
```

可执行文件执行成功后，在屏幕上的关键提示信息示例如下，提示信息中的classType表示类别标识、top1表示该分类的最大置信度、top5表示最大的5个置信度之和，这些值可能会根据版本、环境有所不同，请以实际情况为准（类别编号331表示["hare"]）：

```
......
[INFO]  output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO]  memcopy output data from device to host buffer success.
[INFO]  create output file success, filename=./result/model_output_0, size=4000
[INFO]  start check result file:./result/model_output_0
```

```
[INFO]  check result success, file exist
[INFO]  reselut file: [./result/model_output_0.txt]
[INFO]  result:classType[331],top1[xxxxxx],top5[xxxxxx]
[INFO]  Process dvpp success
[INFO]  model execute success
[INFO]  output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO]  memcopy output data from device to host buffer success.
[INFO]  create output file success, filename=./result/model_output_1, size=4000
[INFO]  start check result file:./result/model_output_1
[INFO]  check result success, file exist
[INFO]  reselut file: [./result/model_output_1.txt]
[INFO]  result:classType[331],top1[xxxxxx],top5[xxxxxx]
[INFO]  Process dvpp success
[INFO]  model execute success
[INFO]  output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO]  memcopy output data from device to host buffer success.
[INFO]  create output file success, filename=./result/model_output_2, size=4000
[INFO]  start check result file:./result/model_output_2
[INFO]  check result success, file exist
[INFO]  reselut file: [./result/model_output_2.txt]
[INFO]  result:classType[331],top1[xxxxxx],top5[xxxxxx]
[INFO]  Process dvpp success
[INFO]  model execute success
[INFO]  output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO]  memcopy output data from device to host buffer success.
[INFO]  create output file success, filename=./result/model_output_3, size=4000
[INFO]  start check result file:./result/model_output_3
[INFO]  check result success, file exist
[INFO]  reselut file: [./result/model_output_3.txt]
[INFO]  result:classType[331],top1[xxxxxx],top5[xxxxxx]
[INFO]  Process dvpp success
[INFO]  model execute success
[INFO]  output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO]  memcopy output data from device to host buffer success.
[INFO]  create output file success, filename=./result/model_output_4, size=4000
[INFO]  start check result file:./result/model_output_4
[INFO]  check result success, file exist
[INFO]  reselut file: [./result/model_output_4.txt]
[INFO]  result:classType[331],top1[xxxxxx],top5[xxxxxx]
[INFO]  Process dvpp success
[INFO]  model execute success
[INFO]  output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO]  memcopy output data from device to host buffer success.
[INFO]  create output file success, filename=./result/model_output_5, size=4000
[INFO]  start check result file:./result/model_output_5
[INFO]  check result success, file exist
[INFO]  reselut file: [./result/model_output_5.txt]
[INFO]  result:classType[331],top1[xxxxxx],top5[xxxxxx]
[INFO]  Process dvpp success
[INFO]  model execute success
[INFO]  output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO]  memcopy output data from device to host buffer success.
[INFO]  create output file success, filename=./result/model_output_6, size=4000
[INFO]  start check result file:./result/model_output_6
[INFO]  check result success, file exist
[INFO]  reselut file: [./result/model_output_6.txt]
[INFO]  result:classType[331],top1[xxxxxx],top5[xxxxxx]
[INFO]  Process dvpp success
[INFO]  model execute success
[INFO]  output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO]  memcopy output data from device to host buffer success.
[INFO]  create output file success, filename=./result/model_output_7, size=4000
[INFO]  start check result file:./result/model_output_7
[INFO]  check result success, file exist
[INFO]  reselut file: [./result/model_output_7.txt]
[INFO]  result:classType[331],top1[xxxxxx],top5[xxxxxx]
[INFO]  Process dvpp success
[INFO]  model execute success
[INFO]  output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
```

```
[INFO]  memcopy output data from device to host buffer success.
[INFO]  create output file success, filename=./result/model_output_8, size=4000
[INFO]  start check result file:./result/model_output_8
[INFO]  check result success, file exist
[INFO]  reselut file: [./result/model_output_8.txt]
[INFO]  result:classType[331],top1[xxxxxx],top5[xxxxxx]
[INFO]  Process dvpp success
[INFO]  model execute success
[INFO]  output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO]  memcopy output data from device to host buffer success.
[INFO]  create output file success, filename=./result/model_output_9, size=4000
[INFO]  start check result file:./result/model_output_9
[INFO]  check result success, file exist
[INFO]  reselut file: [./result/model_output_9.txt]
[INFO]  result:classType[331],top1[xxxxxx],top5[xxxxxx]
[INFO]  unload model success, modelId is 1
[INFO]  execute sample success
......
```

说明

- 可执行文件运行过程中，VDEC解码成功后，模型推理前，会在main文件同级生成outdir目录，用于保存解码后的YUV420SP NV12格式的图片，推理结束后，系统会自动清理outdir目录。
- 可执行文件执行成功后，同时会在main文件同级的result目录下生成结果文件，便于后期查看。*.h265格式的视频码流（仅包含一帧）被解码出一张YUV420SP NV12格式的图片，对该张图片做十次循环处理，包括缩放图片和模型推理，分别得到十张图片的推理结果：
  - model_output_*：模型推理结果的二进制文件。
  - model_output_*.txt：模型推理结果的txt文件。

----结束

10.7 基于 Caffe ResNet-50 网络实现图片分类（同步推理）

10.7.1 样例介绍

获取样例

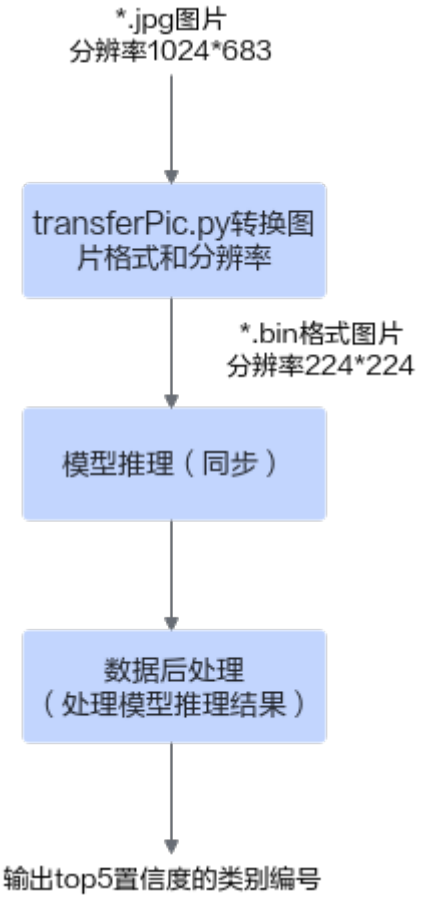
单击**resnet50_imagenet_classification**获取样例

功能描述

该样例主要是基于Caffe ResNet-50网络（单输入、单Batch）实现图片分类的功能。

将Caffe ResNet-50网络的模型文件转换为适配昇腾AI处理器的离线模型（*.om文件），在样例中，加载该om文件，对2张*.jpg图片进行同步推理，分别得到推理结果后，再对推理结果进行处理，输出top5置信度的类别标识。

图 10-3 Sample 示例



原理介绍

在该Sample中，涉及的关键功能点，如下表所示。

初始化	• 调用aclInit接口初始化AscendCL配置。 • 调用aclFinalize接口实现AscendCL去初始化。
Device管理	• 调用aclrtSetDevice接口指定用于运算的Device。 • 调用aclrtGetRunMode接口获取软件栈的运行模式，根据运行模式的不同，内部处理流程不同。 • 调用aclrtResetDevice接口复位当前运算的Device，回收Device上的资源。
Context管理	• 调用aclrtCreateContext接口创建Context。 • 调用aclrtDestroyContext接口销毁Context。
Stream管理	• 调用aclrtCreateStream接口创建Stream。 • 调用aclrtDestroyStream接口销毁Stream。
内存管理	• 调用aclrtMalloc接口申请Device上的内存。 • 调用aclrtFree接口释放Device上的内存。

数据传输	如果在Device上运行应用，则无需进行数据传输。
模型推理	● 调用aclmdlLoadFromFileWithMem接口从*.om文件加载模型。 ● 调用aclmdlExecute接口执行模型推理，同步接口。 ● 调用aclmdlUnload接口卸载模型。

数据后处理	提供样例代码，处理模型推理的结果，直接在终端上显示top5置信度的类别编号。
	另外，样例中提供了自定义接口DumpModelOutputResult，用于将模型推理的结果写入文件（运行可执行文件后，推理结果文件在运行环境上的应用可执行文件的同级目录下），默认未调用该接口，用户可在sample_process.cpp中，在调用OutputModelResult接口前，增加如下代码调用DumpModelOutputResult接口：
	// print the top 5 confidence values with indexes.use function DumpModelOutputResult
	// if want to dump output result to file in the current directory
	modelProcess.DumpModelOutputResult();
	modelProcess.OutputModelResult();

目录结构

样例代码结构如下所示。

```
├── data
│   ├── dog1_1024_683.jpg          //测试数据,需要按指导获取测试图片，放到data目录下
│   ├── dog2_1024_683.jpg          //测试数据,需要按指导获取测试图片，放到data目录下

├── inc
│   ├── model_process.h            //声明模型处理相关函数的头文件
│   ├── sample_process.h           //声明资源初始化/销毁相关函数的头文件
│   ├── utils.h                    //声明公共函数（例如：文件读取函数）的头文件

├── script
│   ├── transferPic.py             //将*.jpg转换为*.bin，同时将图片从1024*683的分辨率缩放为224*224

├── src
│   ├── acl.json          //系统初始化的配置文件
│   ├── CMakeLists.txt    //编译脚本
│   ├── main.cpp          //主函数，图片分类功能的实现文件
│   ├── model_process.cpp //模型处理相关函数的实现文件
│   ├── sample_process.cpp //资源初始化/销毁相关函数的实现文件
│   ├── utils.cpp         //公共函数（例如：文件读取函数）的实现文件

├── .project      //工程信息文件，包含工程类型、工程描述、运行目标设备类型等
├── CMakeLists.txt //编译脚本，调用src目录下的CMakeLists文件
```

10.7.2 Ascend EP，编译及在 Device 上运行应用（Atlas 200/300/500 推理产品）（Atlas 推理系列产品）

步骤1 模型转换。

1. 以运行用户登录开发环境。
2. 参考《**ATC工具使用指南（开放态）**》，准备环境，包括获取工具、设置环境变量。
3. 准备数据。

   您可以从以下链接中获取ResNet-50网络的模型文件（*.prototxt）、权重文件（*.caffemodel），并以运行用户将获取的文件上传至开发环境的

“resnet50_imagenet_classification样例目录/caffe_model”目录下。如果目录不存在，需要自行创建。

- ResNet-50网络的模型文件（*.prototxt）：单击**Link**下载该文件。
- ResNet-50网络的权重文件（*.caffemodel）：单击**Link**下载该文件。

4. 将ResNet-50网络转换为适配昇腾AI处理器的离线模型（*.om文件）。

   切换到resnet50_imagenet_classification样例目录，执行如下命令：

```
atc --model=caffe_model/resnet50.prototxt --weight=caffe_model/resnet50.caffemodel --framework=0 --output=model/resnet50 --soc_version=${soc_version} --input_format=NCHW --input_fp16_nodes=data --output_type=FP32 --out_nodes=prob:0
```

   - --soc_version：昇腾AI处理器的版本，请根据实际情况替换。

     可从ATC安装路径的“atc/data/platform_config”目录下查看昇腾AI处理器的版本，对应“*.ini”文件的名字（不带.ini后缀）即为soc_version。
   - --output：生成的resnet50.om文件存放在“样例目录/model”目录下。建议使用命令中的默认设置，否则在编译代码前，您还需要修改sample_process.cpp中的omModelPath参数值。

     ```
     const char* omModelPath = "../model/resnet50.om";
     ```

   关于各参数的详细解释，请参见《**ATC工具使用指南（开放态）**》。

步骤2 以运行用户登录开发环境，并切换到resnet50_imagenet_classification样例所在的目录。

步骤3 编译代码。

1. 设置环境变量，编译脚本src/CMakeLists.txt通过环境变量所设置的头文件、库文件的路径来编译代码。

   如下为示例，$HOME/AscendMiniOs表示Runtime开放形态安装包的安装路径，请根据实际情况替换。

```
export DDK_PATH=$HOME/AscendMiniOs
export NPU_HOST_LIB=$HOME/AscendMiniOs/runtime/lib64/stub
```

   说明

   使用“$HOME/AscendMiniOs/runtime/lib64/stub”目录下的*.so库，是为了编译基于AscendCL接口的代码逻辑时，不依赖其它组件（例如Driver）的任何*.so库。

   编译通过后，在Device上运行应用时，应用会链接到Device上“Runtime开放形态安装包的安装路径/runtime/lib64”目录下的*.so库，运行时会自动链接到依赖其它组件的*.so库。

2. 在样例目录下，创建目录用于存放编译文件，例如，本文中，创建的目录为“build/intermediates/device”。

```
mkdir -p build/intermediates/device
```

3. 切换到“build/intermediates/device”目录，执行**cmake**生成编译文件。
   - “../../../src”表示CMakeLists.txt文件所在的目录，请根据实际目录层级修改。
   - 在-DCMAKE_CXX_COMPILER参数处设置交叉编译器的路径，“$HOME/Ascend/ascend-toolkit/latest”表示Toolkit组件的默认安装路径，请根据实际情况替换。

```
cd build/intermediates/device
cmake ../../../src -DCMAKE_CXX_COMPILER=$HOME/Ascend/ascend-toolkit/latest/toolkit/toolchain/hcc/bin/aarch64-target-linux-gnu-g++ -DCMAKE_SKIP_RPATH=TRUE
```

4. 执行**make**命令，生成的可执行文件main在“样例目录/out”目录下。

```
make
```

步骤4 准备测试数据。

1. 请从以下链接获取该样例的输入图片，并以运行用户将获取的文件上传至开发环境的“resnet50_imagenet_classification样例目录/data”目录下。如果目录不存在，需自行创建。

   https://obs-9be7.obs.cn-east-2.myhuaweicloud.com/models/aclsample/dog1_1024_683.jpg

   https://obs-9be7.obs.cn-east-2.myhuaweicloud.com/models/aclsample/dog2_1024_683.jpg
2. 以运行用户登录开发环境。
3. 切换到“resnet50_imagenet_classification样例目录/data”目录下，执行transferPic.py脚本，将*.jpg转换为*.bin，同时将图片从1024*683的分辨率缩放为224*224。在“resnet50_imagenet_classification样例目录/data”目录下生成2个*.bin文件。
   ```
   python3 ../script/transferPic.py
   ```

   说明

   如果执行脚本报错“ModuleNotFoundError: No module named 'PIL'”，则表示缺少Pillow库，请使用**pip3 install Pillow --user**命令安装Pillow库。

步骤5 运行应用。

1. 以运行用户将开发环境的resnet50_imagenet_classification样例目录及目录下的文件上传到运行环境（Device），例如“$HOME/acl_resnet50”。
2. 以运行用户登录运行环境（Device）。
3. 切换到可执行文件main所在的目录，例如“acl_resnet50/out”，给该目录下的main文件加执行权限。
   ```
   chmod +x main
   ```
4. 切换到可执行文件main所在的目录，例如“acl_resnet50/out”，运行可执行文件。
   ```
   ./main
   ```

   执行成功后，在屏幕上的关键提示信息示例如下，提示信息中的index表示类别标识、value表示该分类的最大置信度，index和value值可能会根据版本、环境有所不同，请以实际情况为准（类别编号161表示["basset", "basset hound"]，类别编号267表示["standard poodle"]）：

   ```
   [INFO] acl init success
   [INFO] open device 0 success
   [INFO] create context success
   [INFO] create stream success
   [INFO] load model ../model/resnet50.om success
   [INFO] create model description success
   [INFO] create model output success
   [INFO] start to process file:../data/dog1_1024_683.bin
   [INFO] model execute success
   [INFO] top 1: index[161] value[xxxxxx]
   [INFO] top 2: index[xxx] value[xxxxxx]
   [INFO] top 3: index[xxx] value[xxxxxx]
   [INFO] top 4: index[xxx] value[xxxxxx]
   [INFO] top 5: index[xxx] value[xxxxxx]
   [INFO] output data success
   [INFO] start to process file:../data/dog2_1024_683.bin
   [INFO] model execute success
   [INFO] top 1: index[267] value[xxxxxx]
   [INFO] top 2: index[xxx] value[xxxxxx]
   [INFO] top 3: index[xxx] value[xxxxxx]
   [INFO] top 4: index[xxx] value[xxxxxx]
   [INFO] top 5: index[xxx] value[xxxxxx]
   [INFO] output data success
   [INFO] Unload model success, modelId is 1
   [INFO] execute sample success
   ```

```
[INFO] end to destroy stream
[INFO] end to destroy context
[INFO] end to reset device is 0
```

----结束

10.8 基于 Caffe ResNet-50 网络实现图片分类（异步推理）

10.8.1 样例介绍

获取样例

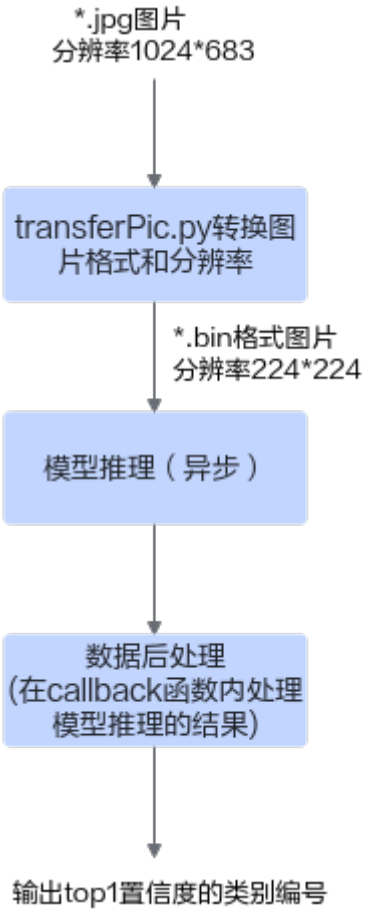
单击**resnet50_async_imagenet_classification**获取样例

功能描述

该样例主要是基于Caffe ResNet-50网络（单输入、单Batch）实现图片分类的功能。

将Caffe ResNet-50网络的模型文件转换为适配昇腾AI处理器的离线模型（*.om文件），在样例中，加载该om文件，对2张*.jpg图片进行n次异步推理（n作为运行应用的参数，由用户配置），分别得到n次推理结果后，再对推理结果进行处理，输出top1置信度的类别标识。

图 10-4 Sample 示例



原理介绍

在该Sample中，涉及的关键功能点，如下表所示。

初始化	● 调用aclInit接口初始化AscendCL配置。 ● 调用aclFinalize接口实现AscendCL去初始化。
Device管理	● 调用aclrtSetDevice接口指定用于运算的Device。 ● 调用aclrtGetRunMode接口获取软件栈的运行模式，根据运行模式的不同，内部处理流程不同。 ● 调用aclrtResetDevice接口复位当前运算的Device，回收Device上的资源。
Context管理	● 调用aclrtCreateContext接口创建Context。 ● 调用aclrtSetCurrentContext接口设置线程的Context。 ● 调用aclrtDestroyContext接口销毁Context。
Stream管理	● 调用aclrtCreateStream接口创建Stream。 ● 调用aclrtDestroyStream接口销毁Stream。
内存管理	● 调用aclrtMallocHost接口申请Host上内存。 ● 调用aclrtFreeHost释放Host上的内存。 ● 调用aclrtMalloc接口申请Device上的内存。 ● 调用aclrtFree接口释放Device上的内存。
数据传输	如果在Device上运行应用，则无需进行数据传输。
模型推理	● 调用aclmdlLoadFromFileWithMem接口从*.om文件加载模型。 ● 创建新线程（例如t1），在线程函数内调用aclrtProcessReport接口，等待指定时间后，触发回调函数（例如CallBackFunc，用于处理模型推理结果）。 ● 调用aclrtSubscribeReport接口，指定处理Stream上回调函数（CallBackFunc）的线程（t1）。 ● 调用aclmdlExecuteAsync接口执行模型推理，异步接口。 ● 调用aclrtLaunchCallback接口，在Stream的任务队列中增加一个需要在Host/Device上执行的回调函数（CallBackFunc）。 ● 调用aclrtSynchronizeStream接口，阻塞应用程序运行，直到指定Stream中的所有任务都完成。 ● 调用aclrtUnSubscribeReport接口，取消线程注册，Stream上的回调函数（CallBackFunc）不再由指定线程（t1）处理。 ● 模型推理结束后，调用aclmdlUnload接口卸载模型。

数据后处理	提供样例代码，处理模型推理的结果，直接在终端上显示top1置信度的类别编号。
	另外，样例中提供了自定义接口DumpModelOutputResult，用于将模型推理的结果写入文件（运行可执行文件后，推理结果文件在运行环境上的应用可执行文件的同级目录下），默认未调用该接口，用户可在model_process.cpp中，在调用OutputModelResult接口前，增加如下代码调用DumpModelOutputResult接口：
	// OutputModelResult prints the top 1 confidence value with index.
	// If want to dump output result to file in the current directory,
	// use function DumpModelOutputResult.
	ModelProcess::DumpModelOutputResult(data.second);
	ModelProcess::OutputModelResult(data.second);

目录结构

样例代码结构如下所示。

```
├── data
│   ├── dog1_1024_683.jpg          //测试数据,需要按指导获取测试图片，放到data目录下
│   ├── dog2_1024_683.jpg          //测试数据,需要按指导获取测试图片，放到data目录下

├── inc
│   ├── memory_pool.h              //声明内存池处理相关函数的头文件
│   ├── model_process.h            //声明模型处理相关函数的头文件
│   ├── sample_process.h           //声明资源初始化/销毁相关函数的头文件
│   ├── utils.h                    //声明公共函数（例如：文件读取函数）的头文件

├── script
│   ├── transferPic.py             //将*.jpg转换为*.bin，同时将图片从1024*683的分辨率缩放为224*224

├── src
│   ├── acl.json          //系统初始化的配置文件
│   ├── CMakeLists.txt    //编译脚本
│   ├── main.cpp          //主函数，图片分类功能的实现文件
│   ├── memory_pool.cpp   //内存池处理相关函数的实现文件
│   ├── model_process.cpp //模型处理相关函数的实现文件
│   ├── sample_process.cpp //资源初始化/销毁相关函数的实现文件
│   ├── utils.cpp         //公共函数（例如：文件读取函数）的实现文件

├── .project     //工程信息文件，包含工程类型、工程描述、运行目标设备类型等
├── CMakeLists.txt    //编译脚本，调用src目录下的CMakeLists文件
```

10.8.2 Ascend EP，编译及在 Device 上运行应用（Atlas 200/300/500 推理产品）（Atlas 推理系列产品）

步骤1 模型转换。

1. 以运行用户登录开发环境。
2. 参考《**ATC工具使用指南（开放态）**》，准备环境，包括获取工具、设置环境变量。
3. 准备数据。

   您可以从以下链接中获取ResNet-50网络的模型文件（*.prototxt）、权重文件（*.caffemodel），并以运行用户将获取的文件上传至开发环境的“resnet50_async_imagenet_classification样例目录/caffe_model”目录下。如果目录不存在，需要自行创建。

   – ResNet-50网络的模型文件（*.prototxt）：单击**Link**下载该文件。

– ResNet-50网络的权重文件（*.caffemodel）：单击**Link**下载该文件。

4. 将ResNet-50网络转换为适配昇腾AI处理器的离线模型（*.om文件）。

   切换到resnet50_async_imagenet_classification样例目录，执行如下命令：

```
atc --model=caffe_model/resnet50.prototxt --weight=caffe_model/resnet50.caffemodel --framework=0 --output=model/resnet50 --soc_version=${soc_version} --input_format=NCHW --input_fp16_nodes=data -output_type=FP32 --out_nodes=prob:0
```

   – --soc_version：昇腾AI处理器的版本，请根据实际情况替换。

     可从ATC安装路径的“atc/data/platform_config”目录下查看昇腾AI处理器的版本，对应“*.ini”文件的名字（不带.ini后缀）即为soc_version。

   – --output：生成的resnet50.om文件存放在“样例目录/model”目录下。建议使用命令中的默认设置，否则在编译代码前，您还需要修改sample_process.cpp中的omModelPath参数值。

```
const char* omModelPath = "../model/resnet50.om";
```

   关于各参数的详细解释，请参见《**ATC工具使用指南（开放态）**》。

步骤2 以运行用户登录开发环境，并切换到resnet50_async_imagenet_classification样例所在的目录。

步骤3 编译代码。

1. 设置环境变量，编译脚本src/CMakeLists.txt通过环境变量所设置的头文件、库文件的路径来编译代码。

   如下为示例，$HOME/AscendMiniOs表示Runtime开放形态安装包的安装路径，请根据实际情况替换。

```
export DDK_PATH=$HOME/AscendMiniOs
export NPU_HOST_LIB=$HOME/AscendMiniOs/runtime/lib64/stub
```

   说明

   使用“$HOME/AscendMiniOs/runtime/lib64/stub”目录下的*.so库，是为了编译基于AscendCL接口的代码逻辑时，不依赖其它组件（例如Driver）的任何*.so库。

   编译通过后，在Device上运行应用时，应用会链接到Device上“Runtime开放形态安装包的安装路径/runtime/lib64”目录下的*.so库，运行时会自动链接到依赖其它组件的*.so库。

2. 在样例目录下，创建目录用于存放编译文件，例如，本文中，创建的目录为“build/intermediates/device”。

```
mkdir -p build/intermediates/device
```

3. 切换到“build/intermediates/device”目录，执行**cmake**生成编译文件。

   – “../../../src”表示CMakeLists.txt文件所在的目录，请根据实际目录层级修改。

   – 在-DCMAKE_CXX_COMPILER参数处设置交叉编译器的路径，“$HOME/Ascend/ascend-toolkit/latest”表示Toolkit组件的默认安装路径，请根据实际情况替换。

```
cd build/intermediates/device
cmake ../../../src -DCMAKE_CXX_COMPILER=$HOME/Ascend/ascend-toolkit/latest/toolkit/toolchain/hcc/bin/aarch64-target-linux-gnu-g++ -DCMAKE_SKIP_RPATH=TRUE
```

4. 执行**make**命令，生成的可执行文件main在“样例目录/out”目录下。

```
make
```

步骤4 准备测试数据。

1. 请从以下链接获取该样例的输入图片，并以运行用户将获取的文件上传至开发环境的“resnet50_async_imagenet_classification样例目录/data”目录下。如果目录不存在，需自行创建。

https://obs-9be7.obs.cn-east-2.myhuaweicloud.com/models/aclsample/dog1_1024_683.jpg

https://obs-9be7.obs.cn-east-2.myhuaweicloud.com/models/aclsample/dog2_1024_683.jpg

2. 以运行用户登录开发环境。
3. 切换到“resnet50_async_imagenet_classification样例目录/data”目录下，执行transferPic.py脚本，将*.jpg转换为*.bin，同时将图片从1024*683的分辨率缩放为224*224。在“resnet50_async_imagenet_classification样例目录/data”目录下生成2个*.bin文件。

```
python3 ../script/transferPic.py
```

说明

如果执行脚本报错“ModuleNotFoundError: No module named 'PIL'”，则表示缺少Pillow库，请使用**pip3 install Pillow --user**命令安装Pillow库。

步骤5 运行应用。

1. 以运行用户将开发环境的resnet50_async_imagenet_classification样例目录及目录下的文件上传到运行环境（Device），例如“$HOME/acl_resnet50_async”。
2. 以运行用户登录运行环境（Device）。
3. 切换到可执行文件main所在的目录，例如“acl_resnet50_async/out”，给该目录下的main文件加执行权限。

```
chmod +x main
```

4. 切换到可执行文件main所在的目录，例如“acl_resnet50_async/out”，运行可执行文件。
   - 运行可执行文件，不带参数时：
     - 执行模型异步推理的次数默认为100次；
     - callback间隔默认为1，表示1次异步推理后，下发一次callback任务；
     - 内存池中的内存块的个数默认为100个。
   - 运行可执行文件，带参数时：
     - 第一个参数表示执行模型异步推理的次数；
     - 第二个参数表示下发callback间隔，参数值为0时表示不下发callback任务，参数值为非0值（例如m）时表示m次异步推理后下发一次callback任务；
     - 第三个参数表示内存池中内存块的个数，内存块个数需大于等于模型异步推理的次数。用户可根据输入图片数量，来调整内存块的个数，例如有2张输入图片、内存块个数为2时，则1张图片1个内存块；例如有3张输入图片、内存块个数为10时，则执行10次循环，每（10/3取整）个内存块对应同一张图片，剩下1个内存块随机对应1张图片。内存块中存放是模型推理的输入数据、输出数据，若多个内存块对应的是同一张图片，则多个内存块中存放的是相同的输入数据、输出数据，用于输入图片少但又想模拟大量图片数据的场景。

```
./main
```

执行成功后，在屏幕上的关键提示信息示例如下，提示信息中的index表示类别标识、value表示该分类的最大置信度，index和value值可能会根据版本、环境有所不同，请以实际情况为准（类别编号161表示["basset", "basset hound"]，类别编号267表示["standard poodle"]）：

```
[INFO]  ./main param1 param2 param3, param1 is execute model times(default 100), param2 is
callback interval(default 1), param3 is memory pool size(default 100)
[INFO]  execute times = 100
[INFO]  callback interval = 1
[INFO]  memory pool size = 100
[INFO]  acl init success
[INFO]  open device 0 success
[INFO]  create context success
[INFO]  create stream success
[INFO]  get run mode success
[INFO]  load model ../model/resnet50.om success
[INFO]  create model description success
[INFO]  init memory pool success
[INFO]  subscribe report success
[INFO]  top 1: index[267] value[xxxxxx]
[INFO]  top 1: index[161] value[xxxxxx]
[INFO]  top 1: index[267] value[xxxxxx]
[INFO]  top 1: index[161] value[xxxxxx]
[INFO]  top 1: index[161] value[xxxxxx]
[INFO]  top 1: index[267] value[xxxxxx]
[INFO]  top 1: index[161] value[xxxxxx]
[INFO]  top 1: index[267] value[xxxxxx]
[INFO]  top 1: index[161] value[xxxxxx]
[INFO]  top 1: index[267] value[xxxxxx]
......
[INFO]  top 1: index[161] value[xxxxxx]
[INFO]  top 1: index[267] value[xxxxxx]
[INFO]  model execute success
[INFO]  unsubscribe report success
[INFO]  unload model success, modelId is 1
[INFO]  execute sample success
[INFO]  end to destroy stream
[INFO]  end to destroy context
[INFO]  end to reset device is 0
[INFO]  end to finalize acl
```

----结束

10.9 媒体数据处理 V1（抠图，一图多框）

10.9.1 样例介绍

获取样例

单击**batchcrop**获取样例

功能描述

该样例从一张YUV420SP NV12格式的输入图片中按指定区域分别抠出八张224*224子图（YUV420SP NV12）。

原理介绍

在该样例中，涉及的关键功能点，如下表所示。

初始化	• 调用aclInit接口初始化AscendCL配置。 • 调用aclFinalize接口实现AscendCL去初始化。

Device管理	● 调用aclrtSetDevice接口指定用于运算的Device。 ● 调用aclrtGetRunMode接口获取软件栈的运行模式，根据运行模式的不同，内部处理流程不同。 ● 调用aclrtResetDevice接口复位当前运算的Device，回收Device上的资源。
Context管理	● 调用aclrtCreateContext接口创建Context。 ● 调用aclrtDestroyContext接口销毁Context。
Stream管理	● 调用aclrtCreateStream接口创建Stream。 ● 调用aclrtDestroyStream接口销毁Stream。 ● 调用aclrtSynchronizeStream接口阻塞程序运行，直到指定Stream中的所有任务都完成。
内存管理	● 调用aclrtMallocHost接口申请Host上内存。 ● 调用aclrtFreeHost释放Host上的内存。 ● 调用aclrtMalloc接口申请Device上的内存。 ● 调用aclrtFree接口释放Device上的内存。 执行媒体数据处理时，若需要申请Device上的内存存放输入或输出数据，需调用acldvppMalloc申请内存、调用acldvppFree接口释放内存。
数据传输	如果在Device上运行应用，则无需进行数据传输。
媒体数据处理	● 抠图 调用acldvppVpcBatchCropResizeAsync接口按指定区域从输入图片中抠图，再将抠的图片存放到输出内存中，作为输出图片。

目录结构

样例代码结构如下所示。

```
├── data
│   ├── dvpp_vpc_1920x1080_nv12.yuv          //测试数据,需要按指导获取测试图片，放到data目录下

├── inc
│   ├── dvpp_process.h            //声明媒体数据处理相关函数的头文件
│   ├── sample_process.h            //声明模型处理相关函数的头文件
│   ├── utils.h                   //声明公共函数（例如：文件读取函数）的头文件

├── src
│   ├── acl.json              //系统初始化的配置文件
│   ├── CMakeLists.txt        //编译脚本
│   ├── dvpp_process.cpp      //媒体数据处理相关函数的实现文件
│   ├── main.cpp              //主函数，一图多框抠图功能的实现文件
│   ├── sample_process.cpp    //资源初始化/销毁相关函数的实现文件
│   ├── utils.cpp             //公共函数（例如：文件读取函数）的实现文件

├── CMakeLists.txt    //编译脚本，调用src目录下的CMakeLists文件
```

10.9.2 Ascend EP，编译及在 Device 上运行应用（Atlas 200/300/500 推理产品）（Atlas 推理系列产品）

步骤1 以运行用户登录开发环境，并切换到batchcrop样例所在目录。

步骤2 编译代码。

1. 设置环境变量，编译脚本src/CMakeLists.txt通过环境变量所设置的头文件、库文件的路径来编译代码。

   如下为示例，$HOME/AscendMiniOs表示Runtime开放形态安装包的安装路径，请根据实际情况替换。

   ```
   export DDK_PATH=$HOME/AscendMiniOs
   export NPU_HOST_LIB=$HOME/AscendMiniOs/runtime/lib64/stub
   ```

   说明

   使用“$HOME/AscendMiniOs/runtime/lib64/stub”目录下的*.so库，是为了编译基于AscendCL接口的代码逻辑时，不依赖其它组件（例如Driver）的任何*.so库。

   编译通过后，在Device上运行应用时，应用会链接到Device上“Runtime开放形态安装包的安装路径/runtime/lib64”目录下的*.so库，运行时会自动链接到依赖其它组件的*.so库。

2. 在样例目录下，创建目录用于存放编译文件，例如，本文中，创建的目录为“build/intermediates/device”。

   ```
   mkdir -p build/intermediates/device
   ```

3. 切换到“build/intermediates/device”目录，执行**cmake**生成编译文件。
   - “../../../src”表示CMakeLists.txt文件所在的目录，请根据实际目录层级修改。
   - 在-DCMAKE_CXX_COMPILER参数处设置交叉编译器的路径，“$HOME/Ascend/ascend-toolkit/latest”表示Toolkit组件的默认安装路径，请根据实际情况替换。

   ```
   cd build/intermediates/device
   cmake ../../../src -DCMAKE_CXX_COMPILER=$HOME/Ascend/ascend-toolkit/latest/toolkit/
   toolchain/hcc/bin/aarch64-target-linux-gnu-g++ -DCMAKE_SKIP_RPATH=TRUE
   ```

4. 执行**make**命令，生成的可执行文件main在“样例目录/out”目录下。

   ```
   make
   ```

步骤3 准备输入图片。

请从以下链接获取该样例的输入图片，并以运行用户将获取的文件上传至开发环境的“batchcrop样例目录/data”目录下。如果目录不存在，需自行创建。

https://obs-9be7.obs.cn-east-2.myhuaweicloud.com/models/aclsample/dvpp_vpc_1920x1080_nv12.yuv

步骤4 运行应用。

1. 以运行用户将开发环境的batchcrop样例目录及目录下的文件上传到运行环境（Device），例如“$HOME/acl_vpc_batchcrop”。
2. 以运行用户登录运行环境（Device）。
3. 切换到可执行文件main所在的目录，例如“acl_vpc_batchcrop/out”，给该目录下的main文件加执行权限。

   ```
   chmod +x main
   ```

4. 切换到可执行文件main所在的目录，例如“acl_vpc_batchcrop/out”，运行可执行文件。

   ```
   ./main
   ```

执行成功后，在屏幕上的关键提示信息示例如下。

```
[INFO]  aclInit success, ret = 0.
[INFO]  open device 0 success
[INFO]  create context success
[INFO]  create stream success
[INFO]  dvpp init resource success
[INFO]  open file = ./dvpp_vpc_1920x1080_nv12.yuv success.
[INFO]  start set inputDesc success.
[INFO]  write out to file ./cropName0 success.
[INFO]  write out to file ./cropName1 success.
[INFO]  write out to file ./cropName2 success.
[INFO]  write out to file ./cropName3 success.
[INFO]  write out to file ./cropName4 success.
[INFO]  write out to file ./cropName5 success.
[INFO]  write out to file ./cropName6 success.
[INFO]  write out to file ./cropName7 success.
[INFO]  ProcessBatchCrop success.
[INFO]  ProcessBatchCrop success.
[INFO]  DestroyBatchCropResource start
[INFO]  DestroyBatchCropResource end
[INFO]  SampleProcess DestroyResource start.
[INFO]  end to destroy stream
[INFO]  end to destroy context
[INFO]  0 deviceID
[INFO]  end to reset device is 0
[INFO]  SampleProcess DestroyResource success.
[INFO]  end to finalize acl
......
```

说明

执行可执行文件成功后，同时会在main文件同级的目录下生成结果文件，便于后期查看。

cropName0、cropName1、cropName2、cropName3、cropName4、cropName5、cropName6、cropName7这八张图片是从输入图片dvpp_vpc_1920×1980_nv12.yuv中抠出的子图。

----结束

10.10 媒体数据处理 V1（视频编码）

10.10.1 样例介绍

获取样例

单击**venc_image**获取样例

功能描述

该样例将一张YUV420SP NV12格式的图片连续编码n次，生成一个H265格式的视频码流（n可配，通过运行应用时设置入参来配置，默认为16次）。

原理介绍

在该样例中，涉及的关键功能点，如下表所示。

初始化	• 调用aclInit接口初始化AscendCL配置。 • 调用aclFinalize接口实现AscendCL去初始化。

Device管理	● 调用aclrtSetDevice接口指定用于运算的Device。 ● 调用aclrtGetRunMode接口获取软件栈的运行模式，根据运行模式的不同，内部处理流程不同。 ● 调用aclrtResetDevice接口复位当前运算的Device，回收Device上的资源。
Context管理	● 调用aclrtCreateContext接口创建Context。 ● 调用aclrtDestroyContext接口销毁Context。
内存管理	● 调用acldvppMalloc接口申请device上内存。 ● 调用acldvppFree释放device上的内存。
数据传输	如果在Device上运行应用，则无需进行数据传输。
媒体数据处理	视频编码，调用aclvencSendFrame接口将将待编码的图片传到编码器进行编码。

目录结构

样例代码结构如下所示。

```
├── data
│   ├── dvpp_venc_128x128_nv12.yuv          //测试数据,需要按指导获取测试图片，放到data目录下

├── inc
│   ├── venc_process.h              //声明媒体数据处理相关函数的头文件
│   ├── sample_process.h              //声明模型处理相关函数的头文件
│   ├── utils.h                     //声明公共函数（例如：文件读取函数）的头文件

├── src
│   ├── acl.json              //系统初始化的配置文件
│   ├── CMakeLists.txt        //编译脚本
│   ├── main.cpp              //主函数，多图多框抠图功能的实现文件
│   ├── sample_process.cpp    //资源初始化/销毁相关函数的实现文件
│   ├── utils.cpp             //公共函数（例如：文件读取函数）的实现文件
│   ├── venc_process.cpp      //媒体数据处理相关函数的实现文件

├── CMakeLists.txt    //编译脚本，调用src目录下的CMakeLists文件
```

10.10.2 Ascend EP，编译及在 Device 上运行应用（Atlas 200/300/500 推理产品）（Atlas 推理系列产品）

步骤1 以运行用户登录开发环境，并切换到venc_image样例所在目录。

步骤2 编译代码。

1. 设置环境变量，编译脚本src/CMakeLists.txt通过环境变量所设置的头文件、库文件的路径来编译代码。

   如下为示例，$HOME/AscendMiniOs表示Runtime开放形态安装包的安装路径，请根据实际情况替换。

```
export DDK_PATH=$HOME/AscendMiniOs
export NPU_HOST_LIB=$HOME/AscendMiniOs/runtime/lib64/stub
```

说明

使用“$HOME/AscendMiniOs/runtime/lib64/stub”目录下的*.so库，是为了编译基于AscendCL接口的代码逻辑时，不依赖其它组件（例如Driver）的任何*.so库。

编译通过后，在Device上运行应用时，应用会链接到Device上“Runtime开放形态安装包的安装路径/runtime/lib64”目录下的*.so库，运行时会自动链接到依赖其它组件的*.so库。

2. 在样例目录下，创建目录用于存放编译文件，例如，本文中，创建的目录为“build/intermediates/device”。
```
mkdir -p build/intermediates/device
```
3. 切换到“build/intermediates/device”目录，执行**cmake**生成编译文件。
   - “../../../src”表示CMakeLists.txt文件所在的目录，请根据实际目录层级修改。
   - 在-DCMAKE_CXX_COMPILER参数处设置交叉编译器的路径，“$HOME/Ascend/ascend-toolkit/latest”表示Toolkit组件的默认安装路径，请根据实际情况替换。
```
cd build/intermediates/device
cmake ../../../src -DCMAKE_CXX_COMPILER=$HOME/Ascend/ascend-toolkit/latest/toolkit/toolchain/hcc/bin/aarch64-target-linux-gnu-g++ -DCMAKE_SKIP_RPATH=TRUE
```
4. 执行**make**命令，生成的可执行文件main在“样例目录/out”目录下。
```
make
```

步骤3 准备输入图片。

请从以下链接获取该样例的输入图片，并以运行用户将获取的文件上传至开发环境的“venc_image样例目录/data”目录下。如果目录不存在，需自行创建。

https://obs-9be7.obs.cn-east-2.myhuaweicloud.com/models/aclsample/dvpp_venc_128x128_nv12.yuv

步骤4 运行应用。

1. 以运行用户将开发环境的venc_image样例目录及目录下的文件上传到运行环境（Device），例如“$HOME/acl_venc”。
2. 以运行用户登录运行环境（Device）。
3. 切换到可执行文件main所在的目录，例如“acl_venc/out”，给该目录下的main文件加执行权限。
```
chmod +x main
```
4. 切换到可执行文件main所在的目录，例如“acl_venc/out”，运行可执行文件。

   执行可执行文件成功后，同时会在main文件同级output目录下生成编码结果文件dvpp_venc_128x128.h265，便于后期查看。

   执行可执行文件时，可以增加入参来配置编码的次数，例如“./main 20”，如果不增加入参，则默认编码次数为16次。
```
./main
```
   执行成功后，在屏幕上的关键提示信息示例如下。
```
[INFO]  ./main param, param is execute venc times(default 16)
[INFO]  acl init success
[INFO]  open device 0 success
[INFO]  create context success
[INFO]  get run mode success
[INFO]  create process callback thread successfully, threadId = 139926343980800
[INFO]  start check result fold:output/
[INFO]  make directory successfully.
[INFO]  check result success, fold exist
[INFO]  process callback thread start
[INFO]  venc init resource success
```

```
[INFO]  success to callback, stream size:5431
[INFO]  success to callback, stream size:704
[INFO]  success to callback, stream size:693
[INFO]  success to callback, stream size:922
[INFO]  success to callback, stream size:1153
[INFO]  success to callback, stream size:689
[INFO]  success to callback, stream size:857
[INFO]  success to callback, stream size:1273
[INFO]  success to callback, stream size:785
[INFO]  success to callback, stream size:976
[INFO]  success to callback, stream size:869
[INFO]  success to callback, stream size:1445
[INFO]  success to callback, stream size:845
[INFO]  success to callback, stream size:1003
[INFO]  success to callback, stream size:922
[INFO]  success to callback, stream size:905
[INFO]  venc process success
[INFO]  execute sample success
[INFO]  destory process callback thread success
[INFO]  end to destroy context
[INFO]  end to reset device is 0
[INFO]  end to finalize acl
```

----结束

10.11 媒体数据处理 V1（抠图贴图）

10.11.1 样例介绍

获取样例

单击**smallResolution_cropandpaste**获取样例

功能描述

该样例主要实现以下三种场景的抠图贴图：

1、抠图区域小于10*6。

2、贴图区域与抠图区域的缩放比例在[1/32, 16]范围内。

3、满足VPC分辨率要求的输入图片的抠图贴图，分辨率的要求，请参见**11.17.5.2 约束说明**。

原理介绍

在该样例中，涉及的关键功能点，如下表所示。

初始化	• 调用aclInit接口初始化AscendCL配置。 • 调用aclFinalize接口实现AscendCL去初始化。
Device管理	• 调用aclrtSetDevice接口指定用于运算的Device。 • 调用aclrtGetRunMode接口获取软件栈的运行模式，根据运行模式的不同，内部处理流程不同。 • 调用aclrtResetDevice接口复位当前运算的Device，回收Device上的资源。

Context管理	● 调用aclrtCreateContext接口创建Context。 ● 调用aclrtDestroyContext接口销毁Context。
Stream管理	● 调用aclrtCreateStream接口创建Stream。 ● 调用aclrtDestroyStream接口销毁Stream。 ● 调用aclrtSynchronizeStream接口阻塞程序运行，直到指定Stream中的所有任务都完成。
内存管理	● 调用aclrtMallocHost接口申请Host上内存。 ● 调用aclrtFreeHost释放Host上的内存。 ● 调用aclrtMalloc接口申请Device上的内存。 ● 调用aclrtFree接口释放Device上的内存。 执行媒体数据处理时，若需要申请Device上的内存存放输入或输出数据，需调用acldvppMalloc申请内存、调用acldvppFree接口释放内存。
数据传输	如果在Device上运行应用，则无需进行数据传输。
媒体数据处理	抠图贴图： 调用acldvppVpcCropAndPasteAsync接口按指定区域从输入图片中抠图，再将抠的图片贴到目标图片的指定位置，作为输出图片。

目录结构

样例代码结构如下所示。

```
├── data
│   ├── dvpp_vpc_1920x1080_nv12.yuv          //测试数据,需要按指导获取测试图片，放到data目录下

├── inc
│   ├── dvpp_process.h              //声明媒体数据处理相关函数的头文件
│   ├── sample_process.h              //声明模型处理相关函数的头文件
│   ├── utils.h                     //声明公共函数（例如：文件读取函数）的头文件

├── src
│   ├── acl.json              //系统初始化的配置文件
│   ├── CMakeLists.txt        //编译脚本
│   ├── dvpp_process.cpp      //媒体数据处理相关函数的实现文件
│   ├── main.cpp              //主函数，抠图贴图功能的实现文件
│   ├── sample_process.cpp    //资源初始化/销毁相关函数的实现文件
│   ├── utils.cpp             //公共函数（例如：文件读取函数）的实现文件

├── CMakeLists.txt    //编译脚本，调用src目录下的CMakeLists文件
```

10.11.2 Ascend EP，编译及在 Device 上运行应用（Atlas 200/300/500 推理产品）（Atlas 推理系列产品）

步骤1 以运行用户登录开发环境，并切换到smallResolution_cropandpaste样例所在目录。

步骤2 编译代码。

1. 设置环境变量，编译脚本src/CMakeLists.txt通过环境变量所设置的头文件、库文件的路径来编译代码。

如下为示例，$HOME/AscendMiniOs表示Runtime开放形态安装包的安装路径，请根据实际情况替换。

```
export DDK_PATH=$HOME/AscendMiniOs
export NPU_HOST_LIB=$HOME/AscendMiniOs/runtime/lib64/stub
```

说明

使用“$HOME/AscendMiniOs/runtime/lib64/stub”目录下的*.so库，是为了编译基于AscendCL接口的代码逻辑时，不依赖其它组件（例如Driver）的任何*.so库。

编译通过后，在Device上运行应用时，应用会链接到Device上“Runtime开放形态安装包的安装路径/runtime/lib64”目录下的*.so库，运行时会自动链接到依赖其它组件的*.so库。

2. 在样例目录下，创建目录用于存放编译文件，例如，本文中，创建的目录为“build/intermediates/device”。

```
mkdir -p build/intermediates/device
```

3. 切换到“build/intermediates/device”目录，执行**cmake**生成编译文件。
   - “../../../src”表示CMakeLists.txt文件所在的目录，请根据实际目录层级修改。
   - 在-DCMAKE_CXX_COMPILER参数处设置交叉编译器的路径，“$HOME/Ascend/ascend-toolkit/latest”表示Toolkit组件的默认安装路径，请根据实际情况替换。

```
cd build/intermediates/device
cmake ../../../src -DCMAKE_CXX_COMPILER=$HOME/Ascend/ascend-toolkit/latest/toolkit/toolchain/hcc/bin/aarch64-target-linux-gnu-g++ -DCMAKE_SKIP_RPATH=TRUE
```

4. 执行**make**命令，生成的可执行文件main在“样例目录/out”目录下。

```
make
```

步骤3 准备输入图片。

请从以下链接获取该样例的输入图片，并以运行用户将获取的文件上传至开发环境的“smallResolution_cropandpaste样例目录/data”目录下。如果目录不存在，需自行创建。

https://obs-9be7.obs.cn-east-2.myhuaweicloud.com/models/aclsample/dvpp_vpc_1920x1080_nv12.yuv

步骤4 运行应用。

1. 以运行用户将开发环境的smallResolution_cropandpaste样例目录及目录下的文件上传到运行环境（Device），例如“$HOME/acl_vpc_smallResolution_crop”。
2. 以运行用户登录运行环境（Device）。
3. 切换到可执行文件main所在的目录，例如“acl_vpc_smallResolution_crop/out”，给该目录下的main文件加执行权限。

```
chmod +x main
```

4. 切换到可执行文件main所在的目录，例如“acl_vpc_smallResolution_crop/out”，如下示例命令可实现：2次调用VPC进行抠图、缩放，将输入图片中指定区域分辨率为6*6的子图抠出，并放大到分辨率为224*224的图片。

```
./main --inImgName dvpp_vpc_1920x1080_nv12.yuv --inFormat 1 --inWidth 1920 --inHeight 1080 --cLeftOffset 0 --cRightOffset 5 --cTopOffset 0 --cBottomOffset 5 --outImgName output_224_224.yuv --outFormat 1 --outWidth 224 --outHeight 224 --pLeftOffset 0 --pRightOffset 223 --pTopOffset 0 --pBottomOffset 223
```

参数说明如下，您可以根据实际需求修改：

- inImgName：输入图像文件的路径，包含文件名。
- inFormat：输入图片的格式，请参见**11.17.5.2 约束说明**。

- inWidth：输入图片的宽。
- inHeight：输入图片的高。
- cLeftOffset：抠图左偏移，必须为偶数。
- cRightOffset：抠图右偏移，必须为奇数。
- cTopOffset：抠图上偏移，必须为偶数。
- cBottomOffset：抠图下偏移，必须为奇数。
- outImgName：输出图像文件的路径，包含文件名。
- outFormat：输出图片的格式，请参见**11.17.5.2 约束说明**。
- outWidth：输出图片的宽。
- outHeight：输出图片的高。
- pLeftOffset：贴图左偏移，必须为偶数，需要16对齐。
- pRightOffset：贴图右偏移，必须为奇数。
- pTopOffset：贴图上偏移，必须为偶数。
- pBottomOffset：贴图下偏移，必须为奇数。

执行成功后，在屏幕上的关键提示信息示例如下。

```
[INFO]  acl init success
[INFO]  set device 0 success
[INFO]  create context success
[INFO]  create stream success
[INFO]  get run mode success
[INFO]  dvpp init resource success
[INFO]  call SplitProcessCropAndPaste
[INFO]  vpc crop and paste success
[INFO]  execute sample success
[INFO]  end to destroy stream
[INFO]  end to destroy context
[INFO]  end to reset device 0
```

----结束

10.12 基于 Caffe YOLOv3 网络实现目标检测（动态 Batch/动态分辨率）

10.12.1 样例介绍

获取样例

单击**YOLOV3_dynamic_batch_detection_picture**获取样例

功能描述

该样例主要是基于Caffe YOLOv3网络、在动态Batch或动态多分辨率场景下实现目标检测的功能。

将Caffe YOLOv3网络的模型文件转换为适配昇腾AI处理器的离线模型（*.om文件），转换命令中需要设置不同档位的batch size（样例中batch档位分为1，2，4，8）或不同档位的分辨率（样例中分辨率档位分为416, 416；832，832；1248，1248），在应用中加载该om文件，通过传参设置选择不同档位的batch size或者分辨率进行推理，并将推理结果保存到文件中。

原理介绍

在该Sample中，涉及的关键功能点，如下表所示。

初始化	● 调用aclInit接口初始化AscendCL配置。 ● 调用aclFinalize接口实现AscendCL去初始化。
Device管理	● 调用aclrtSetDevice接口指定用于运算的Device。 ● 调用aclrtGetRunMode接口获取软件栈的运行模式，根据运行模式的不同，内部处理流程不同。 ● 调用aclrtResetDevice接口复位当前运算的Device，回收Device上的资源。
Context管理	● 调用aclrtCreateContext接口创建Context。 ● 调用aclrtDestroyContext接口销毁Context。
Stream管理	● 调用aclrtCreateStream接口创建Stream。 ● 调用aclrtDestroyStream接口销毁Stream。
内存管理	● 调用aclrtMalloc接口申请Device上的内存。 ● 调用aclrtFree接口释放Device上的内存。
数据传输	如果在Device上运行应用，则无需进行数据传输。
模型推理	● 调用aclmdlLoadFromFileWithMem接口从*.om文件加载模型。 ● 调用aclmdlSetDynamicBatchSize设置batch size或者调用aclmdlSetDynamicHWSize设置分辨率。 ● 调用aclmdlExecute接口执行模型推理，同步接口。 ● 调用aclmdlUnload接口卸载模型。
数据后处理	样例中提供了自定义接口DumpModelOutputResult，用于将模型推理的结果写入文件（运行可执行文件后，推理结果文件在运行环境上的应用可执行文件的同级目录下）： **processModel.DumpModelOutputResult();**

目录结构

样例代码结构如下所示。

```
├── data
│   ├── tools_generate_data.py          //测试数据生成脚本

├── inc
│   ├── model_process.h              //声明模型处理相关函数的头文件
│   ├── sample_process.h             //声明资源初始化/销毁相关函数的头文件
│   ├── utils.h                    //声明公共函数（例如：文件读取函数）的头文件

├── src
│   ├── acl.json        //系统初始化的配置文件
│   ├── CMakeLists.txt        //编译脚本
│   ├── main.cpp             //主函数，图片分类功能的实现文件
│   ├── model_process.cpp      //模型处理相关函数的实现文件
│   ├── sample_process.cpp     //资源初始化/销毁相关函数的实现文件
│   ├── utils.cpp            //公共函数（例如：文件读取函数）的实现文件
```

```
├── .project     //工程信息文件，包含工程类型、工程描述、运行目标设备类型等
├── CMakeLists.txt    //编译脚本，调用src目录下的CMakeLists文件
```

10.12.2 Ascend EP，编译及在 Device 上运行应用（Atlas 200/300/500 推理产品）（Atlas 推理系列产品）

步骤1 模型转换。

1. 以运行用户登录开发环境。
2. 参考《**ATC工具使用指南（开放态）**》，准备环境，包括获取工具、设置环境变量。
3. 准备数据。

   您可以从以下链接中获取yolov3网络的模型文件（*.prototxt）、权重文件（*.caffemodel），并以运行用户将获取的文件上传至开发环境的“样例目录/caffe_model”目录下。如果目录不存在，需要自行创建。

   – YOLOv3网络的模型文件（*.prototxt）：单击**Link**下载该文件。
   – YOLOv3网络的权重文件（*.caffemodel）：单击**Link**下载该文件。
4. 切换到YOLOV3_dynamic_batch_detection_picture样例目录，将yolov3网络转换为适配昇腾AI处理器的离线模型（*.om文件）。

   如果模型推理的输入数据是动态Batch的，执行如下命令转换模型：

   ```
   atc --framework=0 --model=caffe_model/yolov3.prototxt --weight=caffe_model/yolov3.caffemodel --output=model/yolov3_dynamic_batch --input_shape="data:-1,3,416,416" --input_format=NCHW --dynamic_batch_size="1,2,4,8" --soc_version=${soc_version}
   ```

   如果模型推理的输入数据是动态分辨率的，执行如下命令转换模型：

   ```
   atc --framework=0 --model=caffe_model/yolov3.prototxt --weight=caffe_model/yolov3.caffemodel --output=model/yolov3_dynamic_hw --input_shape="data:1,3,-1,-1" --input_format=NCHW --dynamic_image_size="416,416;832,832;1248,1248" --soc_version=${soc_version}
   ```

   – --soc_version：昇腾AI处理器的版本，请根据实际情况替换。

     可从ATC安装路径的“atc/data/platform_config”目录下查看昇腾AI处理器的版本，对应“*.ini”文件的名字（不带.ini后缀）即为soc_version。
   – --output：生成的yolov3_dynamic_batch.om或者yolov3_dynamic_hw.om文件存放在“样例目录/model”目录下。建议使用命令中的默认设置，否则在编译代码前，您还需要修改sample_process.cpp中的omModelPath参数值。

     ```
     string omModelPath = "../model/yolov3_dynamic_batch.om";
     ......
     string omModelPath = "../model/yolov3_dynamic_hw.om";
     ```

   关于各参数的详细解释，请参见《**ATC工具使用指南（开放态）**》。

步骤2 以运行用户登录开发环境，并切换到YOLOV3_dynamic_batch_detection_picture样例所在目录。

步骤3 编译代码。

1. 设置环境变量，编译脚本src/CMakeLists.txt通过环境变量所设置的头文件、库文件的路径来编译代码。

   如下为示例，$HOME/AscendMiniOs表示Runtime开放形态安装包的安装路径，请根据实际情况替换。

   ```
   export DDK_PATH=$HOME/AscendMiniOs
   export NPU_HOST_LIB=$HOME/AscendMiniOs/runtime/lib64/stub
   ```

说明

使用“$HOME/AscendMiniOs/runtime/lib64/stub”目录下的*.so库，是为了编译基于AscendCL接口的代码逻辑时，不依赖其它组件（例如Driver）的任何*.so库。

编译通过后，在Device上运行应用时，应用会链接到Device上“Runtime开放形态安装包的安装路径/runtime/lib64”目录下的*.so库，运行时会自动链接到依赖其它组件的*.so库。

2. 在样例目录下，创建目录用于存放编译文件，例如，本文中，创建的目录为“build/intermediates/device”。
```
mkdir -p build/intermediates/device
```
3. 切换到“build/intermediates/device”目录，执行**cmake**生成编译文件。
   - “../../../src”表示CMakeLists.txt文件所在的目录，请根据实际目录层级修改。
   - 在-DCMAKE_CXX_COMPILER参数处设置交叉编译器的路径，“$HOME/Ascend/ascend-toolkit/latest”表示Toolkit组件的默认安装路径，请根据实际情况替换。
```
cd build/intermediates/device
cmake ../../../src -DCMAKE_CXX_COMPILER=$HOME/Ascend/ascend-toolkit/latest/toolkit/toolchain/hcc/bin/aarch64-target-linux-gnu-g++ -DCMAKE_SKIP_RPATH=TRUE
```
4. 执行**make**命令，生成的可执行文件main在“样例目录/out”目录下。
```
make
```

步骤4 准备测试数据。

切换到“YOLOV3_dynamic_batch_detection_picture样例目录/data”目录下，执行tools_generate_data.py脚本，可以在“YOLOV3_dynamic_batch_detection_picture样例目录/data”目录下生成各种不同Batch或者不同分辨率场景下的测试bin文件。

示例命令如下，可以生成batch size为1，通道数为3，宽、高都为416像素，数据类型是float32的bin文件input_float32_1x3x416x416.bin.in。
```
python3.7 tools_generate_data.py input -s [1,3,416,416] -r [2,3] -d float32
```

说明

tools_generate_data.py脚本参数说明如下：

- input：bin文件前缀名。
- s：输入数据的Shape信息。
- r：每个通道像素取值范围，该取值范围中的最小值、最大值必须在[0,255]范围。执行tools_generate_data.py脚本，会在-r参数指定的取值范围内随机生成每个通道的像素值。
- d：数据格式，支持int8、uint8、float16、float32、int32、uint32。

步骤5 运行应用。

1. 以运行用户将开发环境的YOLOV3_dynamic_batch_detection_picture样例目录及目录下的文件上传到运行环境（Device），例如“$HOME/acl_yolov3_dynamic_batch”。
2. 以运行用户登录运行环境（Device）。
3. 切换到可执行文件main所在的目录，例如“acl_yolov3_dynamic_batch/out”，给该目录下的main文件加执行权限。
```
chmod +x main
```
4. 切换到可执行文件main所在的目录，例如“acl_yolov3_dynamic_batch/out”，运行可执行文件。
   - 动态Batch场景下，执行如下命令，可执行程序的入参需替换为实际的batch size，必须为模型转换时通过--dynamic_batch_size参数指定的其中一档batch size：

```
./main 1
```

执行成功后，在屏幕上的关键提示信息示例如下：

```
[INFO]  1: ./main [param], [param] is dynamic batch. It should be 1,2,4 or 8. For example: ./
main 8;
[INFO]  2: ./main [param1] [param2], [param1] is dynamic height. [param1] is dynamic width.
It should be 416, 416; 832, 832; 1248, 1248. For example: ./main 416 416
[INFO]  acl init success.
[INFO]  set device 0 success.
[INFO]  create context success.
[INFO]  create stream success.
[INFO]  get run mode success.
[INFO]  load model ../model/yolov3_dynamic_batch.om success.
[INFO]  create model description success.
[INFO]  start to process file: ../data/input_float32_1x3x416x416.bin.in
[INFO]  create model output success.
[INFO]  set dynamic batch size[1] success.
[INFO]  model input num[2], output num[3].
[INFO]  start to print input tensor desc:
[INFO]  index[0]: name[data], inputSize[16613376], fotmat[0], dataType[0]
[INFO]  dimcount:[4],dims:[-1][3][416][416]
[INFO]  index[1]: name[ascend_mbatch_shape_data], inputSize[8], fotmat[2], dataType[9]
[INFO]  dimcount:[1],dims:[1]
[INFO]  start to print output tensor desc:
[INFO]  index[0]: name[layer82-conv:0:layer82-conv], outputSize[1379040], fotmat[0],
dataType[0]
[INFO]  dimcount:[4],dims:[8][255][13][13]
[INFO]  index[1]: name[layer94-conv:0:layer94-conv], outputSize[5516160], fotmat[0],
dataType[0]
[INFO]  dimcount:[4],dims:[8][255][26][26]
[INFO]  index[2]: name[layer106-conv:0:layer106-conv], outputSize[22064640], fotmat[0],
dataType[0]
[INFO]  dimcount:[4],dims:[8][255][52][52]
[INFO]  start to print model dynamic batch info:
[INFO]  dynamic batch count:[4],dims:{[1][2][4][8]}
[INFO]  start to print model current output shape info:
[INFO]  index:0,dims:[1][255][13][13]
[INFO]  index:1,dims:[1][255][26][26]
[INFO]  index:2,dims:[1][255][52][52]
[INFO]  model execute success.
[INFO]  dump data success.
[INFO]  unload model success, modelId is 1.
[INFO]  execute sample success.
[INFO]  end to destroy stream.
[INFO]  end to destroy context.
[INFO]  end to reset device: 0.
[INFO]  end to finalize acl.
```

– 动态分辨率场景下，可执行程序的第一个、第二个入参需要分别替换为实际的高、宽，必须为模型转换时通过--dynamic_image_size参数指定的其中一档分辨率：

```
./main 416 416
```

执行成功后，在屏幕上的关键提示信息示例如下：

```
[INFO]  1: ./main [param], [param] is dynamic batch. It should be 1,2,4 or 8. For example: ./
main 8;

[INFO]  2: ./main [param1] [param2], [param1] is dynamic height. [param1] is dynamic width.
It should be 416, 416; 832, 832; 1248, 1248. For example: ./main 416 416
[INFO]  acl init success.
[INFO]  set device 0 success.
[INFO]  create context success.
[INFO]  create stream success.
[INFO]  get run mode success.
[INFO]  load model ../model/yolov3_dynamic_hw.om success.
[INFO]  create model description success.
[INFO]  start to process file: ../data/input_float32_1x3x416x416.bin.in
[INFO]  create model input success.
[INFO]  create model output success.
```

```
[INFO]  set dynamic hw[416, 416] success.
[INFO]  model input num[2], output num[3].
[INFO]  start to print input tensor desc:
[INFO]  index[0]: name[data], inputSize[18690048], fotmat[0], dataType[0]
[INFO]  dimcount:[4],dims:[1][3][-1][-1]
[INFO]  index[1]: name[ascend_mbatch_shape_data], inputSize[16], fotmat[2], dataType[9]
[INFO]  dimcount:[1],dims:[2]
[INFO]  start to print output tensor desc:
[INFO]  index[0]: name[layer82-conv:0:layer82-conv], outputSize[1551420], fotmat[0],
dataType[0]
[INFO]  dimcount:[4],dims:[1][255][39][39]
[INFO]  index[1]: name[layer94-conv:0:layer94-conv], outputSize[6205680], fotmat[0],
dataType[0]
[INFO]  dimcount:[4],dims:[1][255][78][78]
[INFO]  index[2]: name[layer106-conv:0:layer106-conv], outputSize[24822720], fotmat[0],
dataType[0]
[INFO]  dimcount:[4],dims:[1][255][156][156]
[INFO]  start to print model dynamic hw info:
[INFO]  dynamic hw count:[3],dims:{[416, 416][832, 832][1248, 1248]}
[INFO]  start to print model current output shape info:
[INFO]  index:0,dims:[1][255][13][13]
[INFO]  index:1,dims:[1][255][26][26]
[INFO]  index:2,dims:[1][255][52][52]
[INFO]  model execute success.
[INFO]  dump data success.
[INFO]  unload model success, modelId is 1.
[INFO]  destroy model description success.
[INFO]  destroy model input success.
[INFO]  destroy model output success.
[INFO]  execute sample success.
[INFO]  end to destroy stream.
[INFO]  end to destroy context.
[INFO]  end to reset device: 0.
```

----结束

10.13 媒体数据处理 V2（VPC 抠图/贴图/缩放等）

10.13.1 样例介绍

获取样例

单击**vpc_sample**获取样例

功能描述

该样例实现图片的抠图、缩放、边界填充、色域转换、金字塔、LUT重映射、直方图统计等功能。

原理介绍

在该样例中，涉及的关键功能点，如下表所示。

初始化	• 调用aclInit接口初始化AscendCL配置。 • 调用aclFinalize接口实现AscendCL去初始化。

Device管理	• 调用aclrtSetDevice接口指定用于运算的Device。 • 调用aclrtGetRunMode接口获取软件栈的运行模式，根据运行模式的不同，内部处理流程不同。 • 调用aclrtResetDevice接口复位当前运算的Device，回收Device上的资源。
Context管理	• 调用aclrtCreateContext接口创建Context。 • 调用aclrtDestroyContext接口销毁Context。
内存管理	• 调用aclrtMallocHost接口/aclrtFreeHost接口申请/释放Host上内存。 • 执行媒体数据处理时，若需要申请Device上的内存存放输入或输出数据，需调用hi_mpi_dvpp_malloc接口/hi_mpi_dvpp_free接口申请/释放内存。
数据传输	如果在Device上运行应用，则无需进行数据传输。
媒体数据处理V2	参见**11.18.16 VPC图像处理功能**中的接口。

目录结构

样例代码结构如下所示。

```
├── common                    //示例代码文件所在的目录
│   ├── sample_comm.cpp         //公共函数的实现文件
│   ├── sample_comm.h           //声明公共函数的头文件
│   ├── sample_comm_vpc_*.cpp   //抠图、缩放等函数的实现文件

├── smoke_vpc                 //示例代码文件所在的目录
│   ├── sample_vpc.cpp          //main函数的实现文件

├── CMakeLists.txt            //编译脚本
```

10.13.2 编译及运行应用

单击**vpc_sample**获取样例，查看该样例下的README。

10.14 媒体数据处理 V2（JPEGD 图片解码）

10.14.1 样例介绍

获取样例

单击**jpegd_sample**获取样例

功能描述

该样例实现.jpg、.jpeg、.JPG、.JPEG图片的解码。

原理介绍

在该样例中，涉及的关键功能点，如下表所示。

初始化	• 调用aclInit接口初始化AscendCL配置。 • 调用aclFinalize接口实现AscendCL去初始化。
Device管理	• 调用aclrtSetDevice接口指定用于运算的Device。 • 调用aclrtGetRunMode接口获取软件栈的运行模式，根据运行模式的不同，内部处理流程不同。 • 调用aclrtResetDevice接口复位当前运算的Device，回收Device上的资源。
Context管理	• 调用aclrtCreateContext接口创建Context。 • 调用aclrtDestroyContext接口销毁Context。
内存管理	• 调用aclrtMallocHost接口/aclrtFreeHost接口申请/释放Host上内存。 • 执行媒体数据处理时，若需要申请Device上的内存存放输入或输出数据，需调用hi_mpi_dvpp_malloc接口/hi_mpi_dvpp_free接口申请/释放内存。
数据传输	如果在Device上运行应用，则无需进行数据传输。
媒体数据处理V2	参见**11.18.17 VDEC视频解码功能/JPEGD图片解码功能**中的接口。

目录结构

样例代码结构如下所示。

```
├── common                    //示例代码文件所在的目录
│   ├── sample_comm.h           //声明解码函数的头文件
│   ├── sample_comm_jpegd.cpp   //解码函数的实现文件

├── smoke_vpc                 //示例代码文件所在的目录
│   ├── sample_jpegd.cpp        //main函数的实现文件

├── CMakeLists.txt            //编译脚本
```

10.14.2 编译及运行应用

单击**jpegd_sample**获取样例，查看该样例下的README。

10.15 媒体数据处理 V2（JPEGE 图片编码）

10.15.1 样例介绍

获取样例

单击**jpege_sample**获取样例

功能描述

该样例实现将YUV格式图片编码成.jpg图片。

原理介绍

在该样例中，涉及的关键功能点，如下表所示。

初始化	• 调用aclInit接口初始化AscendCL配置。 • 调用aclFinalize接口实现AscendCL去初始化。
Device管理	• 调用aclrtSetDevice接口指定用于运算的Device。 • 调用aclrtGetRunMode接口获取软件栈的运行模式，根据运行模式的不同，内部处理流程不同。 • 调用aclrtResetDevice接口复位当前运算的Device，回收Device上的资源。
Context管理	• 调用aclrtCreateContext接口创建Context。 • 调用aclrtDestroyContext接口销毁Context。
内存管理	• 调用aclrtMallocHost接口/aclrtFreeHost接口申请/释放Host上内存。 • 执行媒体数据处理时，若需要申请Device上的内存存放输入或输出数据，需调用hi_mpi_dvpp_malloc接口/hi_mpi_dvpp_free接口申请/释放内存。
数据传输	如果在Device上运行应用，则无需进行数据传输。
媒体数据处理V2	参见**11.18.18 VENC视频编码功能/JPEGE图片编码功能**中的接口。

目录结构

样例代码结构如下所示。

```
├── common                    //示例代码文件所在的目录
│   ├── sample_comm.h           //声明编码函数的头文件
│   ├── sample_comm_venc.cpp    //编码函数的实现文件

├── smoke_vpc                 //示例代码文件所在的目录
│   ├── sample_jpege.cpp        //main函数的实现文件

├── CMakeLists.txt            //编译脚本
```

10.15.2 编译及运行应用

单击**jpege_sample**获取样例，查看该样例下的README。

10.16 媒体数据处理 V2（VDEC 视频解码）

10.16.1 样例介绍

获取样例

单击vdec_sample获取样例

功能描述

该样例实现H264码流格式的视频解码。

原理介绍

在该样例中，涉及的关键功能点，如下表所示。

初始化	• 调用aclInit接口初始化AscendCL配置。 • 调用aclFinalize接口实现AscendCL去初始化。
Device管理	• 调用aclrtSetDevice接口指定用于运算的Device。 • 调用aclrtGetRunMode接口获取软件栈的运行模式，根据运行模式的不同，内部处理流程不同。 • 调用aclrtResetDevice接口复位当前运算的Device，回收Device上的资源。
Context管理	• 调用aclrtCreateContext接口创建Context。 • 调用aclrtDestroyContext接口销毁Context。
内存管理	• 调用aclrtMallocHost接口/aclrtFreeHost接口申请/释放Host上内存。 • 执行媒体数据处理时，若需要申请Device上的内存存放输入或输出数据，需调用hi_mpi_dvpp_malloc接口/hi_mpi_dvpp_free接口申请/释放内存。
数据传输	如果在Device上运行应用，则无需进行数据传输。
媒体数据处理V2	参见**11.18.17 VDEC视频解码功能/JPEGD图片解码功能**中的接口。

目录结构

样例代码结构如下所示。

```
├── Vdec.h            //声明解码函数的头文件
├── Vdec.cpp          //解码函数的实现文件
├── VdecDemo.cpp      //main函数的实现文件
├── CMakeLists.txt    //编译脚本
```

10.16.2 编译及运行应用

单击vdec_sample获取样例，查看该样例下的README。

10.17 媒体数据处理 V2（VENC 视频编码）

10.17.1 样例介绍

获取样例

单击**venc_sample**获取样例

功能描述

该样例实现将YUV420SP、YVU420SP格式的视频编码成H264、H265格式的码流。

原理介绍

在该样例中，涉及的关键功能点，如下表所示。

初始化	• 调用aclInit接口初始化AscendCL配置。 • 调用aclFinalize接口实现AscendCL去初始化。
Device管理	• 调用aclrtSetDevice接口指定用于运算的Device。 • 调用aclrtGetRunMode接口获取软件栈的运行模式，根据运行模式的不同，内部处理流程不同。 • 调用aclrtResetDevice接口复位当前运算的Device，回收Device上的资源。
Context管理	• 调用aclrtCreateContext接口创建Context。 • 调用aclrtDestroyContext接口销毁Context。
内存管理	• 调用aclrtMallocHost接口/aclrtFreeHost接口申请/释放Host上内存。 • 执行媒体数据处理时，若需要申请Device上的内存存放输入或输出数据，需调用hi_mpi_dvpp_malloc接口/hi_mpi_dvpp_free接口申请/释放内存。
数据传输	如果在Device上运行应用，则无需进行数据传输。
媒体数据处理V2	参见**11.18.18 VENC视频编码功能/JPEGE图片编码功能**中的接口。

目录结构

样例代码结构如下所示。

```
├── libHisiSdkCodec            //示例代码文件所在的目录
│   ├── hmev_hisi_sdk_*.h      //声明编码函数的头文件
│   ├── hmev_hisi_sdk_*.cpp    //编码函数的实现文件

├── CMakeLists.txt             //编译脚本
```

10.17.2 编译及运行应用

单击venc_sample获取样例，查看该样例下的README。

10.18 媒体数据处理 V2（PNGD 图片解码）

10.18.1 样例介绍

获取样例

单击pngd_sample获取样例

功能描述

该样例实现.png、.PNG图片的解码。

原理介绍

在该样例中，涉及的关键功能点，如下表所示。

初始化	• 调用aclInit接口初始化AscendCL配置。 • 调用aclFinalize接口实现AscendCL去初始化。
Device管理	• 调用aclrtSetDevice接口指定用于运算的Device。 • 调用aclrtGetRunMode接口获取软件栈的运行模式，根据运行模式的不同，内部处理流程不同。 • 调用aclrtResetDevice接口复位当前运算的Device，回收Device上的资源。
Context管理	• 调用aclrtCreateContext接口创建Context。 • 调用aclrtDestroyContext接口销毁Context。
内存管理	• 调用aclrtMallocHost接口/aclrtFreeHost接口申请/释放Host上内存。 • 执行媒体数据处理时，若需要申请Device上的内存存放输入或输出数据，需调用hi_mpi_dvpp_malloc接口/hi_mpi_dvpp_free接口申请/释放内存。
数据传输	如果在Device上运行应用，则无需进行数据传输。
媒体数据处理V2	参见11.18.19 PNGD图片解码功能中的接口。

目录结构

样例代码结构如下所示。

```
├── src                    //示例代码文件所在的目录
│   ├── sample_comm.h      //声明解码函数的头文件
```

```
│   ├── sample_comm_pngd.cpp    //解码函数的实现文件
│   ├── sample_pngd.cpp         //解码函数的实现文件

├── CMakeLists.txt              //编译脚本
```

10.18.2 编译及运行应用

单击**pngd_sample**获取样例，查看该样例下的README。

11 AscendCL API 参考

接口列表
废弃接口/返回码列表
同步&异步API说明
系统配置
Device管理
Context管理
算力Group查询与设置
Stream管理
Event管理
执行控制
内存管理
模型加载与执行
单算子API执行
算子编译
算子加载与执行
CBLAS接口
媒体数据处理V1
媒体数据处理V2
异常处理
日志管理
特征向量检索
Profiling数据采集
Tensor数据传输

数据类型转换及获取数据大小

共享队列管理

共享Buffer管理

数据类型及其操作接口

11.1 接口列表

11.1.1 接口版本说明

V1.1.0为第一个AscendCL接口版本，根据AscendCL接口的变更，该版本号会随之改变，您可以调用**aclrtGetVersion**接口获取AscendCL接口版本号、再通过本章的接口列表查阅变更接口。

AscendCL接口版本号命名采用：A.B.C模式，其中：

- A表示有不兼容修改，从1开始，如果出现接口的不兼容变更时，加1；
- B表示新增接口，按照迭代周期，有新增接口时加1；
- C表示bug修复，从0开始，表示本版本仅仅解决了问题。在A、B不变的情况下，如果本版本修复了问题，则C加1；但A、B增加的时候，C一般为0。

说明

昇腾AI处理器对枚举值、数据类型及其操作接口的支持情况请参见**11.18.20 枚举值及数据结构**、**11.27 数据类型及其操作接口**。

接口的返回码说明请参见**11.27.1 aclError**，如果接口调用失败，则接口的出参无效，不可继续使用。

接口命名规则的说明请参见**13.2 表达约定**。

11.1.2 V1.1.0 及以上版本

各版本对AscendCL接口的支持情况如下所示，各标识的含义如下：

- √：支持。
- x：不支持。
- 废弃：后续版本待废弃，具体废弃接口说明请参见**11.2 废弃接口/返回码列表**。

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
系统配置	aclInit		√	√
	aclFinalize		√	√
	aclrtGetVersion		√	√
	aclSetCompileopt		x	√
	aclrtGetSocName		√	√

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
	aclGetRecentErrMsg		√	√
Device管理	aclrtSetDevice		√	√
	aclrtResetDevice		√	√
	aclrtGetDevice		√	√
	aclrtGetRunMode		√	√
	aclrtSetTsDevice		√	√
	aclrtGetDeviceCount		√	√
Context管理	aclrtCreateContext		√	√
	aclrtDestroyContext		√	√
	aclrtSetCurrentContext		√	√
	aclrtGetCurrentContext		√	√
算力Group查询与设置	aclrtSetGroup		x	x
	aclrtGetGroupCount		x	x
	aclrtCreateGroupInfo		x	x
	aclrtDestroyGroupInfo		x	x
	aclrtGetAllGroupInfo		x	x
	aclrtGetGroupInfoDetail		x	x
Stream管理	aclrtCreateStream		√	√
	aclrtDestroyStream		√	√
同步等待	aclrtCreateEvent		√	√
	aclrtCreateEventWithFlag		√	√

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
	aclrtDestroyEvent		√	√
	aclrtRecordEvent		√	√
	aclrtResetEvent		√	√
	aclrtQueryEvent		废弃	废弃
	aclrtQueryEventStatus		√	√
	aclrtQueryEventWaitStatus		√	√
	aclrtSynchronizeEvent		√	√
	aclrtEventElapsedTime		√	√
	aclrtStreamWaitEvent		√	√
	aclrtSynchronizeDevice		√	√
	aclrtSynchronizeStream		√	√
	aclrtSubscribeReport		√	√
	aclrtLaunchCallback		√	√
	aclrtProcessReport		√	√
	aclrtUnSubscribeReport		√	√
	aclrtSetExceptionInfoCallback		√	√
	aclrtGetTaskIdFromExceptionInfo		√	√
	aclrtGetStreamIdFromExceptionInfo		√	√
	aclrtGetThreadIdFromExceptionInfo		√	√

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
	aclrtGetDeviceIdFromExceptionInfo		√	√
	aclrtSetOpWaitTimeout		√	x
内存管理	aclrtMalloc		√	√
	aclrtMallocCached		√	√
	aclrtMemFlush		√	√
	aclrtMemInvalidate		√	√
	aclrtFree		√	√
	aclrtMallocHost		√	√
	aclrtFreeHost		√	√
	aclrtMemset		√	√
	aclrtMemsetAsync		√	√
	aclrtMemcpy		√	√
	aclrtMemcpyAsync		√	√
	aclrtGetMemInfo		√	√
	aclrtDeviceCanAccessPeer		x	√
	aclrtDeviceEnablePeerAccess		x	√
	aclrtDeviceDisablePeerAccess		x	√
	aclrtMemcpy2d		√	仅Atlas 推理系列加速卡产品支持该接口，Atlas 推理系列加速模块产品不支持该接口。

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
	aclrtMemcpy2dAsync		√	仅Atlas推理系列加速卡产品支持该接口，Atlas 推理系列加速模块产品不支持该接口。
模型加载与执行	aclmdlLoadFromFile		√	√
	aclmdlLoadFromMem		√	√
	aclmdlLoadFromFileWithMem		√	√
	aclmdlLoadFromMemWithMem		√	√
	aclmdlLoadFromFileWithQ		x	x
	aclmdlLoadFromMemWithQ		x	x
	aclmdlExecute		√	√
	aclmdlExecuteAsync		√	√
	aclmdlUnload		√	√
	aclmdlQuerySize		√	√
	aclmdlQuerySizeFromMem		√	√
	aclmdlSetDynamicBatchSize		√	√
	aclmdlSetDynamicHWSize		√	√
	aclmdlSetInputAIPP		√	√
	aclmdlGetFirstAippInfo		√	√

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
	aclmdlGetAippType		√	√
	aclmdlSetAIPPByInputIndex		√	√
	aclmdlSetInputDynamicDims		√	√
	aclmdlCreateAndGetOpDesc		√	√
	aclmdlInitDump		√	√
	aclmdlSetDump		√	√
	aclmdlFinalizeDump		√	√
	aclmdlSetConfigOpt		√	√
	aclmdlLoadWithConfig		√	√
算子编译	aclopRegisterCompileFunc		√	√
	aclopUnregisterCompileFunc		√	√
	aclopCreateKernel		√	√
	aclopSetKernelArgs		√	√
	aclopSetKernelWorkspaceSizes		√	√
	aclopUpdateParams		√	√
	aclopCompile		x	√
	aclopSetCompileFlag		x	√
	aclGenGraphAndDumpForOp		√	√
算子加载与执行	aclopSetModelDir		√	√
	aclopLoad		√	√
	aclopExecute		废弃	废弃

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
	aclopExecuteV2		√	√
	aclopCompileAndExecute		√	√
	aclopExecWithHandle		√	√
	aclopInferShape		√	√
CBLAS接口	aclblasGemvEx		√	√
	aclblasCreateHandleForGemvEx		√	√
	aclblasHgemv		√	√
	aclblasCreateHandleForHgemv		√	√
	aclblasS8gemv		√	√
	aclblasCreateHandleForS8gemv		√	√
	aclblasGemmEx		√	√
	aclblasCreateHandleForGemmEx		√	√
	aclblasHgemm		√	√
	aclblasCreateHandleForHgemm		√	√
	aclblasS8gemm		√	√
	aclblasCreateHandleForS8gemm		√	√
	aclopCast		√	√
	aclopCreateHandleForCast		√	√
媒体数据处理V1	内存申请与释放	acldvppMalloc	√	√
		acldvppFree	√	√
	通道创建与释放	acldvppCreateChannel	√	√
		acldvppDestroyChannel	√	√

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
		aclvdecCreateChannel	√	√
		aclvdecDestroyChannel	√	√
		aclvencCreateChannel	√	√
		aclvencDestroyChannel	√	√
	VPC功能	acldvppVpcResizeAsync	√	√
		acldvppVpcCropAsync	√	√
		acldvppVpcCropResizeAsync	√	√
		acldvppVpcBatchCropAsync	√	√
		acldvppVpcBatchCropResizeAsync	√	√
		acldvppVpcCropAndPasteAsync	√	√
		acldvppVpcCropResizePasteAsync	√	√
		acldvppVpcBatchCropAndPasteAsync	√	√
		acldvppVpcBatchCropResizePasteAsync	√	√
		acldvppVpcBatchCropResizeMakeBorderAsync	x	√
		acldvppVpcConvertColorAsync	x	√

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
		acldvppVpcPyrDownAsync	x	√
		acldvppVpcEqualizeHistAsync	x	√
		acldvppVpcMakeBorderAsync	x	√
		acldvppVpcCalcHistAsync	x	√
	JPEGD功能	acldvppJpegDecodeAsync	√	√
		acldvppJpegGetImageInfo	√	√
		acldvppJpegGetImageInfoV2	√	√
		acldvppJpegPredictDecSize	√	√
	JPEGE功能	acldvppJpegEncodeAsync	√	√
		acldvppJpegPredictEncSize	√	√
	PNGD功能	acldvppPngDecodeAsync	√	x
		acldvppPngGetImageInfo	√	x
		acldvppPngPredictDecSize	√	x
	VDEC功能	aclvdecSendFrame	√	√
		aclvdecSendSkippedFrame	√	√
		aclvdecCallback	√	√
	VENC功能	aclvencSendFrame	√	√
		aclvencCallback	√	√

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
媒体数据处理V2	公共接口	hi_mpi_sys_init	x	√
		hi_mpi_sys_exit	x	√
		hi_mpi_dvpp_malloc	x	√
		hi_mpi_dvpp_free	x	√
		hi_mpi_dvpp_get_image_info	x	√
		hi_mpi_dvpp_get_version	x	√
		hi_mpi_sys_create_epoll	x	√
		hi_mpi_sys_ctl_epoll	x	√
		hi_mpi_sys_wait_epoll	x	√
		hi_mpi_sys_close_epoll	x	√
		hi_mpi_sys_set_chn_csc_matrix	x	√
		hi_mpi_sys_get_chn_csc_matrix	x	√
	VPC功能	hi_mpi_vpc_create_chn	x	√
		hi_mpi_vpc_destroy_chn	x	√
		hi_mpi_vpc_resize	x	√
		hi_mpi_vpc_crop	x	√
		hi_mpi_vpc_crop_resize	x	√
		hi_mpi_vpc_crop_resize_paste	x	√

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
		hi_mpi_vpc_convert_color	x	√
		hi_mpi_vpc_convert_color_to_yuv420	x	√
		hi_mpi_vpc_copy_make_border	x	√
		hi_mpi_vpc_pyrdown	x	√
		hi_mpi_vpc_calc_hist	x	√
		hi_mpi_vpc_equalize_hist	x	√
		hi_mpi_vpc_crop_resize_make_border	x	√
		hi_mpi_vpc_batch_crop_resize_paste	x	√
		hi_mpi_vpc_batch_crop_resize_make_border	x	√
		hi_mpi_vpc_get_process_result	x	√
		hi_mpi_vpc_sys_create_chn	x	√
	VDEC功能/JPEGD功能	hi_mpi_vdec_create_chn	x	√
		hi_mpi_vdec_destroy_chn	x	√
		hi_mpi_vdec_get_chn_attr	x	√
		hi_mpi_vdec_set_chn_attr	x	x
		hi_mpi_vdec_start_recv_stream	x	√

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
		hi_mpi_vdec_stop_recv_stream	x	√
		hi_mpi_vdec_query_status	x	√
		hi_mpi_vdec_reset_chn	x	√
		hi_mpi_vdec_set_chn_param	x	√
		hi_mpi_vdec_get_chn_param	x	√
		hi_mpi_vdec_set_protocol_param	x	x
		hi_mpi_vdec_get_protocol_param	x	√
		hi_mpi_vdec_send_stream	x	√
		hi_mpi_vdec_get_frame	x	√
		hi_mpi_vdec_release_frame	x	√
		hi_mpi_vdec_get_fd	x	√
		hi_mpi_vdec_close_fd	x	√
		hi_vdec_get_pic_buf_size	x	√
		hi_vdec_get_tmv_buf_size	x	√
	VENC功能/JPEGE功能	hi_mpi_venc_create_chn	x	√
		hi_mpi_venc_destroy_chn	x	√
		hi_mpi_venc_start_chn	x	√

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
		hi_mpi_venc_stop_chn	x	√
		hi_mpi_venc_query_status	x	√
		hi_mpi_venc_get_stream	x	√
		hi_mpi_venc_release_stream	x	√
		hi_mpi_venc_send_frame	x	√
		hi_mpi_venc_set_mod_param	x	√
		hi_mpi_venc_get_mod_param	x	√
		hi_mpi_venc_request_idr	x	√
		hi_mpi_venc_get_fd	x	√
		hi_mpi_venc_close_fd	x	x
		hi_mpi_venc_set_jpeg_param	x	√
		hi_mpi_venc_get_jpeg_param	x	√
		hi_mpi_venc_set_chn_param	x	√
		hi_mpi_venc_set_scene_mode	x	√
		hi_mpi_venc_set_rc_param	x	√
		hi_mpi_venc_get_rc_param	x	√
		hi_mpi_venc_set_jpeg_huffman_param	x	√

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
		hi_mpi_venc_get_jpeg_huffman_param	x	√
		hi_mpi_venc_compact_jpeg_tables	x	√
	PNGD功能	hi_mpi_pngd_create_chn	x	√
		hi_mpi_pngd_destroy_chn	x	√
		hi_mpi_pngd_send_stream	x	√
		hi_mpi_pngd_get_image_data	x	√
		hi_mpi_png_get_image_info	x	√
日志管理	aclAppLog		√	√
特征向量检索	aclfvInit		x	仅Atlas 推理系列加速卡产品支持该接口，Atlas 推理系列加速模块产品不支持该接口。
	aclfvRelease		x	仅Atlas 推理系列加速卡产品支持该接口，Atlas 推理系列加速模块产品不支持该接口。

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
	aclfvRepoAdd		x	仅Atlas 推理系列加速卡产品支持该接口，Atlas 推理系列加速模块产品不支持该接口。
	aclfvRepoDel		x	仅Atlas 推理系列加速卡产品支持该接口，Atlas 推理系列加速模块产品不支持该接口。
	aclfvDel		x	仅Atlas 推理系列加速卡产品支持该接口，Atlas 推理系列加速模块产品不支持该接口。
	aclfvModify		x	仅Atlas 推理系列加速卡产品支持该接口，Atlas 推理系列加速模块产品不支持该接口。

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
	aclfvSearch		x	仅Atlas推理系列加速卡产品支持该接口，Atlas 推理系列加速模块产品不支持该接口。
Profiling配置	aclprofInit		√	√
	aclprofStart		√	√
	aclprofStop		√	√
	aclprofFinalize		√	√
	aclprofCreateStamp		√	√
	aclprofSetStampTraceMessage		√	√
	aclprofMark		√	√
	aclprofPush		√	√
	aclprofPop		√	√
	aclprofRangeStart		√	√
	aclprofRangeStop		√	√
	aclprofDestroyStamp		√	√
	aclprofModelSubscribe		√	√
	aclprofModelUnSubscribe		√	√
	aclprofGetOpDescSize		√	√
	aclprofGetOpNum		√	√
	aclprofGetOpTypeLen		√	√
	aclprofGetOpType		√	√

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
	aclprofGetOpNameLen		√	√
	aclprofGetOpName		√	√
	aclprofGetOpStart		√	√
	aclprofGetOpEnd		√	√
	aclprofGetOpDuration		√	√
	aclprofGetModelId		√	√
	aclprofGetStepTimestamp		√	√
Tensor数据传输接口	acltdtCreateChannel		x	x
	acltdtSendTensor		x	x
	acltdtReceiveTensor		x	x
	acltdtStopChannel		x	x
	acltdtDestroyChannel		x	x
	acltdtCreateChannelWithCapacity		x	x
	acltdtQueryChannelSize		x	x
数据类型转换及获取数据大小	aclDataTypeSize		√	√
	aclFloat16ToFloat		√	√
	aclFloatToFloat16		√	√
扩展接口	aclextProcessAicpuTask		x	x
共享队列管理	acltdtCreateQueue		√	√
	acltdtDestroyQueue		√	√

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
	acltdtEnqueueData		√	√
	acltdtDequeueData		√	√
	acltdtEnqueue		x	x
	acltdtDequeue		x	x
	acltdtBindQueueRoutes		√	√
	acltdtUnbindQueueRoutes		√	√
	acltdtQueryQueueRoutes		√	√
	acltdtGrantQueue		x	x
	acltdtAttachQueue		x	x
共享Buffer管理	acltdtAllocBuf		√	√
	acltdtFreeBuf		√	√
	acltdtGetBufData		√	√

11.1.3 V1.2.0 及以上版本

在V1.1.0基础上，新增了如下接口。

各版本对AscendCL接口的支持情况如下所示，各标识的含义如下：

- √：支持。
- x：不支持。
- 废弃：后续版本待废弃，具体废弃接口说明请参见**11.2 废弃接口/返回码列表**。

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
算子加载与执行	aclopCompileAndExecuteV2		√	√

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
媒体数据处理V2	VDEC功能/JPEGD功能	hi_mpi_vdec_set_jpegd_precision_mode	x	√
媒体数据处理V2	VPC功能	hi_mpi_vpc_convert_color_v2	x	√
数据类型及其操作接口	acltdtDataset	acltdtGetDatasetName	x	x
模型加载与执行	aclmdlLoadFromFileWithQ		√	√
模型加载与执行	aclmdlLoadFromMemWithQ		√	√

11.1.4 V1.3.0 及以上版本

在V1.2.0基础上，新增了如下接口。

各版本对AscendCL接口的支持情况如下所示，各标识的含义如下：

- √：支持。
- x：不支持。
- 废弃：后续版本待废弃，具体废弃接口说明请参见**11.2 废弃接口/返回码列表**。

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
媒体数据处理V2	VPC功能	hi_mpi_vpc_set_roundview_stitching_param	x	x
		hi_mpi_vpc_get_roundview_stitching_param	x	x
		hi_mpi_vpc_roundview_stitching	x	x
		hi_vpc_crop_resize_resize_paste	x	x
		hi_mpi_vpc_set_chn_workspace	x	x

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
		hi_mpi_vpc_get_lut_mem_size	x	x
		hi_mpi_vpc_get_affine_lut	x	x
		hi_mpi_vpc_get_perspective_lut	x	x
		hi_mpi_vpc_get_remap_lut	x	x
		hi_mpi_vpc_lut_remap	x	x
		hi_mpi_vpc_median_blur	x	x
		hi_mpi_vpc_erode	x	x
		hi_mpi_vpc_dilate	x	x
		hi_mpi_vpc_blur	x	x
		hi_mpi_vpc_gaussian_blur	x	x
		hi_mpi_vpc_filter2d	x	x
		hi_mpi_vpc_rotate	x	x
		hi_mpi_vpc_draw_mosaic	x	x
		hi_mpi_vpc_draw_cover	x	x
		hi_mpi_vpc_draw_line	x	x
		hi_mpi_vpc_draw_osd	x	x
媒体数据处理V2	VENC功能/JPEGE功能	hi_mpi_venc_send_jpege_frame	x	√
		hi_mpi_venc_get_jpege_predicted_size	x	√
共享Buffer管理	acltdtSetBufUserData		√	√

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
	acltdtGetBufUserData		√	√
	acltdtSetBufDataLen		√	√
	acltdtGetBufDataLen		√	√
	acltdtCopyBufRef		√	√
	acltdtAppendBufChain		√	√
	acltdtGetBufChainNum		√	√
	acltdtGetBufFromChain		√	√

11.1.5 V1.4.0 及以上版本

在V1.3.0基础上，新增了如下接口。

各版本对AscendCL接口的支持情况如下所示，各标识的含义如下：

- √：支持。
- x：不支持。
- 废弃：后续版本待废弃，具体废弃接口说明请参见**11.2 废弃接口/返回码列表**。

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
Profiling数据采集	Profiling AscendCL API（通过Profiling AscendCL API采集并落盘性能数据）	aclprofSetConfig	√	√
媒体数据处理V2	VPC功能	hi_mpi_vpc_get_affine_transform	x	x
		hi_mpi_vpc_get_rotation_matrix	x	x

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
		hi_mpi_vpc_warp_affine	x	x
		hi_mpi_vpc_flip	x	x
媒体数据处理V2	VENC功能/JPEGE功能	hi_mpi_venc_set_h264_vui	x	√
		hi_mpi_venc_get_h264_vui	x	√
		hi_mpi_venc_set_h265_vui	x	√
		hi_mpi_venc_get_h265_vui	x	√
		hi_mpi_venc_get_scene_mode	x	x
		hi_mpi_venc_set_roi_attr	x	x
		hi_mpi_venc_get_roi_attr	x	x
		hi_mpi_venc_set_ref_param	x	x
		hi_mpi_venc_get_ref_param	x	x
系统配置	aclrtSetDeviceSatMode		x	x
	aclrtGetDeviceSatMode		x	x
Stream管理	aclrtSetStreamOverflowSwitch		x	x
	aclrtGetStreamOverflowSwitch		x	x
同步等待	aclrtSynchronizeStreamWithTimeout		√	√

11.1.6 V1.5.0 及以上版本

11.1.6.1 接口列表（Atlas 200/300/500 推理产品&Atlas 推理系列产品）

针对以下产品，在V1.4.0基础上，新增了接口：

- Atlas 200/300/500 推理产品
- Atlas 推理系列产品

各版本对AscendCL接口的支持情况如下所示，各标识的含义如下：

- √：支持。
- x：不支持。
- 废弃：后续版本待废弃，具体废弃接口说明请参见11.2 废弃接口/返回码列表。

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
媒体数据处理V2	VENC功能/JPEGE功能	hi_mpi_venc_set_cu_pred	x	√
		hi_mpi_venc_get_cu_pred	x	√
		hi_mpi_venc_set_intra_refresh	x	√
		hi_mpi_venc_get_intra_refresh	x	√
	公共接口	hi_mpi_sys_bind	x	x
		hi_mpi_sys_unbind	x	x
	ISP系统控制	该章节下的所有接口	x	x
	MIPI Rx ioctl命令字	该章节下的所有接口	x	x
	VI功能	该章节下的所有接口	x	x
	VPSS功能	该章节下的所有接口	x	x
Stream管理	aclrtCreateStreamWithConfig		√	√
	aclrtDestroyStreamForce		√	√

11.1.7 V1.6.0 及以上版本

11.1.7.1 接口列表（Atlas 200/300/500 推理产品&Atlas 推理系列产品）

针对以下产品，在V1.5.0基础上，新增了接口：

- Atlas 200/300/500 推理产品
- Atlas 推理系列产品

各版本对AscendCL接口的支持情况如下所示，各标识的含义如下：

- √：支持。
- x：不支持。
- 废弃：后续版本待废弃，具体废弃接口说明请参见**11.2 废弃接口/返回码列表**。

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
媒体数据处理V2	VDEC视频解码功能/JPEGD图片解码功能	hi_mpi_vdec_set_display_mode	x	x
		hi_mpi_vdec_get_display_mode	x	x
	AI音频输入/AO音频输出	该章节下的所有接口	x	x
	音量调节	该章节下的所有接口	x	x
	AENC音频编码/ADEC音频解码	该章节下的所有接口	x	x
	Region区域管理	该章节下的所有接口	x	x
	VO视频输出功能	该章节下的所有接口	x	x
	TDE图形绘制功能	该章节下的所有接口	x	x
	HIFB叠加图形层管理功能	该章节下的所有接口	x	x
	HDMI外设	该章节下的所有接口	x	x
	VENC视频编码功能/JPEGE图片编码功能	hi_mpi_venc_set_chn_attr	x	√
		hi_mpi_venc_get_chn_attr	x	√
	VI视频输入功能	hi_mpi_vi_set_pipe_attr	x	x
		hi_mpi_vi_get_pipe_attr	x	x
		hi_mpi_vi_set_pipe_vc_number	x	x

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
		hi_mpi_vi_get_pipe_vc_number	x	x
		hi_mpi_vi_set_chn_low_delay_attr	x	x
		hi_mpi_vi_get_chn_low_delay_attr	x	x
		hi_mpi_vi_set_vi_vpss_mode	x	x
		hi_mpi_vi_get_vi_vpss_mode	x	x
		hi_mpi_vi_set_pipe_pre_be_crop	x	x
		hi_mpi_vi_set_pipe_bind_strobe	x	x
算子加载与执行	aclrtSetOpExecuteTimeOut		x	√
系统配置	aclGetCompileopt		√	√
	aclGetCompileoptSize		√	√
Stream管理	aclrtSetStreamFailureMode		x	√
	aclrtSetStreamConfigOpt		x	x
	aclrtCreateStreamV2		x	x
同步等待	aclrtSynchronizeEventWithTimeout		√	√
模型加载与执行	aclmdlSetExecConfigOpt		√	√
	aclmdlExecuteV2		√	√
	aclmdlExecuteAsyncV2		√	√
内存管理	aclrtAllocatorRegister		x	√
	aclrtAllocatorUnregister		x	√
	aclrtMallocAlign32		√	√

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
Context管理	aclrtCtxSetSysParamOpt		√	√
	aclrtCtxGetSysParamOpt		√	√
	aclrtGetOverflowStatus		√	√
	aclrtResetOverflowStatus		√	√
Device管理	aclrtGetDeviceUtilizationRate		√	√
单算子API执行	公共接口（AscendCL meta） **说明** 定义算子时依赖的一些AscendCL元接口，通过这些元接口可以构建所需的数据结构，如aclTensor、aclScalar、aclIntArray等。		√	√
	NN类算子接口（aclnn） **说明** aclnn提供的核心功能接口，包括张量基础计算（加/减/乘/除等）以及NN相关的如卷积、池化、激活函数等操作。	该章节下的所有接口	x	x
算力Group查询与设置	该章节下的所有接口		x	√ Control CPU开放形态支持调用该接口

11.1.8 V1.7.0 及以上版本

针对以下产品，在V1.6.0基础上，新增了接口：

- Atlas 200/300/500 推理产品
- Atlas 推理系列产品

各版本对AscendCL接口的支持情况如下所示，各标识的含义如下：

- √：支持。
- x：不支持。
- 废弃：后续版本待废弃，具体废弃接口说明请参见**11.2 废弃接口/返回码列表**。

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
算子加载与执行	aclopStartDumpArgs		x	x
	aclopStopDumpArgs		x	x
同步等待	aclrtSubscribeHostFunc		x	x
	aclrtProcessHostFunc		x	x
	aclrtUnSubscribeHostFunc		x	x
单算子API执行	DVPP类算子接口	acldvppNormalize	x	x
		acldvppDecodeJpeg	x	x
		acldvppResize	x	x
		acldvppImgToTensor	x	x
		acldvppAdjustBrightness	x	x
		acldvppAdjustHue	x	x
		acldvppAdjustSaturation	x	x
		acldvppAdjustContrast	x	x
	融合类算子接口	aclnnPromptFlashAttention	x	x
		aclnnIncreFlashAttention	x	x
		aclnnFlashAttentionScore	x	x
		aclnnFlashAttentionScoreGrad	x	x
		aclnnMoeFFN	x	x
媒体数据处理V2	VPC图像处理功能	hi_mpi_vpc_set_chn_opt_attr	x	x

一级节点	二级节点	三级节点	Atlas 200/300/500 推理产品	Atlas 推理系列产品
Stream管理	aclrtStreamQuery		√	√
内存管理	aclrtReserveMemAddress		√	√
	aclrtReleaseMemAddress		√	√
	aclrtMallocPhysical		√	√
	aclrtFreePhysical		√	√
	aclrtMapMem		√	√
	aclrtUnmapMem		√	√
Device管理	aclrtQueryDeviceStatus		x	√

11.2 废弃接口/返回码列表

接口

- **aclopExecute**接口
  此接口后续版本会废弃，请使用**aclopExecuteV2**接口。
- **aclGetTensorDescDim**接口
  此接口后续版本会废弃，请使用**aclGetTensorDescDimV2**接口。
- **aclGetDataBufferSize**接口
  此接口后续版本会废弃，请使用**aclGetDataBufferSizeV2**接口。
- **aclSetTensorStorageShape**接口
  此接口后续版本会废弃，请使用**aclSetTensorShape**接口。
- **aclSetTensorStorageFormat**接口
  此接口后续版本会废弃，请使用**aclSetTensorFormat**接口。
- **aclrtQueryEvent**接口
  此接口后续版本会废弃，请使用**aclrtQueryEventStatus**接口。
- **aclopSetCompileFlag**接口
  此接口后续版本会废弃，请使用**aclSetCompileopt**接口设置编译选项。

返回码

- **ACL_ERROR_NONE**返回码

此返回码后续版本会废弃，请使用**ACL_SUCCESS**返回码。

- **ACL_ERROR_NOT_STATIC_AIPP**

  此返回码后续版本会废弃，请使用**ACL_ERROR_GE_AIPP_NOT_EXIST**返回码。
- **ACL_ERROR_STREAM_NOT_SUBSCRIBE**

  此返回码后续版本会废弃，请使用**ACL_ERROR_RT_STREAM_NO_CB_REG**返回码。
- **ACL_ERROR_THREAD_NOT_SUBSCRIBE**

  此返回码后续版本会废弃，请使用**ACL_ERROR_RT_THREAD_SUBSCRIBE**返回码。
- **ACL_ERROR_WAIT_CALLBACK_TIMEOUT**

  此返回码后续版本会废弃，请使用**ACL_ERROR_RT_REPORT_TIMEOUT**返回码。
- **ACL_ERROR_INVALID_DEVICE**

  此返回码后续版本会废弃，请使用**ACL_ERROR_RT_INVALID_DEVICEID**返回码。
- **ACL_ERROR_GROUP_NOT_SET**

  此返回码后续版本会废弃，请使用**ACL_ERROR_RT_GROUP_NOT_SET**返回码。
- **ACL_ERROR_GROUP_NOT_CREATE**

  此返回码后续版本会废弃，请使用**ACL_ERROR_RT_GROUP_NOT_CREATE**返回码。

11.3 同步&异步 API 说明

CANN支持以下几类显式同步，调用此类接口后，主机线程会阻塞直到相关的任务执行完成。

- **设备同步：aclrtSynchronizeDevice**

  阻塞当前主机线程直到device上所有显式或隐式创建的stream都完成所有先前下发的任务。应该尽量少使用该函数，以免拖延主机运行。
- **流同步：aclrtSynchronizeStream**

  阻塞当前主机线程直到指定的stream中完成所有下发的任务。
- **事件同步：aclrtSynchronizeEvent**

  阻塞当前主机线程直到指定的Event事件完成。属于更细粒度的同步。

对于异步接口，主机线程调用异步接口后仅代表下发任务，在任务未完成前，异步接口已向主机线程返回成功。用户需要调用上面的显式同步接口阻塞主机线程，等待任务完成。

11.4 系统配置

11.4.1 aclInit

函数功能

AscendCL初始化函数。

约束说明

- 一个进程内只能调用一次aclInit接口。
- 使用AscendCL接口开发应用时，必须先调用aclInit接口，否则可能会导致后续系统内部资源初始化出错，进而导致其它业务异常。

函数原型

aclError aclInit(const char *configPath)

参数说明

参数名	输入/输出	说明
configPath	输入	配置文件所在路径的指针，包含文件名，配置文件内容为json格式（json文件内的“{”的层级最多为10，“[”的层级最多为10）。如果以下的默认配置已满足需求，无需修改，可向aclInit接口中传入NULL，或者可将配置文件配置为空json串（即配置文件中只有{}）。 配置文件格式为json格式，当前支持以下配置： ● **Dump信息配置**，包括以下配置： – **模型Dump配置**（用于导出模型中每一层算子输入和输出数据）、**单算子Dump配置**（用于导出单个算子的输入和输出数据），导出的数据用于与指定模型或算子进行比对，定位精度问题，配置示例、说明及约束请参见《**CANN 开发工具指南（开放态）**》中的“精度比对工具”章节。默认不启用该dump配置。 – **异常算子Dump配置**（用于导出异常算子的输入输出数据、workspace信息），导出的数据用于分析AI Core Error问题，配置示例请参见**配置文件示例（异常算子Dump配置）**。默认不启用该dump配置。 – **溢出算子Dump配置**（用于导出模型中溢出算子的输入和输出数据），导出的数据用于分析溢出原因，定位模型精度的问题，配置示例、说明及约束请参见**配置文件示例（溢出算子Dump配置）**。默认不启用该dump配置。**预留功能，暂不支持**。 ● **Profiling采集信息配置**，示例、配置说明及约束请参见《**CANN 开发工具指南（开放态）**》中的“性能分析工具”章节。默认不启用Profiling采集信息配置。 ● **算子缓存信息老化配置**，为节约内存和平衡调用性能，可通过max_opqueue_num参数配置“算子类型-单算子模型”映射队列的最大长度，如果长度达到最大，则会先删除长期未使用的映射信息以及缓存中的单算子模型，再加载最新的映射信息以及对应的单算子模型。如果不配置映射队列的最大长度，则默认最大长度为20000。示例及约束说明请参见**配置文件示例（算子缓存信息老化配置）**。 建议不要同时配置dump信息和Profiling采集信息，否则dump操作会影响系统性能，导致Profiling采集的性能数据指标不准确。

返回值说明

返回0表示成功，返回其它值表示失败。

配置文件示例（异常算子 Dump 配置）

将dump_scene参数设置为lite_exception，启用异常算子Dump功能，同时可配合使用ASCEND_WORK_PATH环境变量配置落盘路径，否则落盘在应用程序的当前执行目录下。

配置文件中的示例内容如下：

```
{
  "dump":{
        "dump_scene":"lite_exception"
  }
}
```

配置文件示例（溢出算子 Dump 配置）

溢出算子Dump配置的相关约束说明如下：

- 将dump_debug配置为on表示开启溢出算子配置，不配置dump_debug或将dump_debug配置为off表示不开启溢出算子配置。
- 若开启溢出算子配置，则dump_path必须配置，表示导出数据文件的存储路径。

  支持配置绝对路径或相对路径：
  - 绝对路径配置以“/”开头，例如：/home。
  - 相对路径配置直接以目录名开始，例如：output。

  获取导出的数据文件后，如何解析请参见**7.13 溢出算子数据采集及分析**。
- 溢出算子Dump配置，不能与模型Dump配置或单算子Dump配置同时开启，否则会返回报错。

配置文件中的示例内容如下：

```
{
   "dump":{
      "dump_path":"output",
      "dump_debug":"on"
   }
}
```

配置文件示例（算子缓存信息老化配置）

算子缓存信息老化配置的相关约束说明如下：

- 对于静态加载的算子，调用**aclopSetModelDir**接口加载指定目录下的单算子模型或调用**aclopLoad**接口加载指定单算子模型时，老化配置无效，不会对该部分的算子信息做老化。
- 在线编译算子的场景下，调用**aclopCompile**接口编译算子或调用**aclopCompileAndExecute**接口编译执行算子时，接口内部会按照入参加载单算子模型，老化配置有效。

  如果用户调用**aclopCompile**接口编译算子、调用**aclopExecuteV2**接口执行算子，则在编译算子后需及时执行算子，否则可能导致执行算子时，算子信息已被老化，需要重新编译。建议调用**aclopCompileAndExecuteV2**接口编译执行算子。
- AscendCL内部分开维护固定Shape和动态Shape算子的映射队列，最大长度都为max_opqueue_num参数值。
- max_opqueue_num参数值为静态加载算子的单算子模型个数和在线编译算子的单算子模型个数的总和，因此max_opqueue_num参数值应大于当前进程中可用

的、静态加载算子的单算子模型个数，否则会导致在线编译算子的信息无法老化。

配置文件中的示例内容如下：

```
{
    "max_opqueue_num": "10000"
}
```

相关接口

AscendCL还提供了其它使能Dump或Profiling的接口，如下，与aclInit不同的是，以下这些接口相对灵活，可以在一个进程内调用多次接口，每次调用接口时可以基于不同的Dump配置或Profiling配置。

- 获取Dump数据，参见**11.12.23 aclmdlInitDump**、**11.12.24 aclmdlSetDump**、**11.12.25 aclmdlFinalizeDump**。
- 获取Profiling数据，参见**11.22 Profiling数据采集**。

参考资源

接口调用示例，参见**4.3 AscendCL初始化与去初始化**。

11.4.2 aclFinalize

函数功能

AscendCL去初始化函数，用于释放进程内的AscendCL相关资源。

在该接口内，默认增加2000ms延时（实际最大延时可达2000ms），用于Device业务日志回传，保证ERROR级别和EVENT级别日志不丢失，您可以将ASCEND_LOG_DEVICE_FLUSH_TIMEOUT环境变量设置为0（命令示例：export ASCEND_LOG_DEVICE_FLUSH_TIMEOUT=0），去除该默认延时。关于ASCEND_LOG_DEVICE_FLUSH_TIMEOUT环境变量的详细描述请参见《**日志参考（开放态）**》。

约束说明

应用进程退出前，应显式调用该接口实现AscendCL去初始化，否则可能会导致异常，例如应用进程退出时有异常报错。

不建议在析构函数中调用aclFinalize接口，否则在进程退出时可能由于单例析构顺序未知而导致进程异常退出的问题。

函数原型

aclError aclFinalize()

参数说明

无

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见4.3 AscendCL初始化与去初始化。

11.4.3 aclrtGetVersion

函数功能

查询接口版本号，AscendCL接口版本号命名采用：A.B.C模式，其中，A表示有不兼容修改，B表示新增接口，C表示bug修复。

函数原型

aclError aclrtGetVersion(int32_t *majorVersion, int32_t *minorVersion, int32_t *patchVersion)

参数说明

参数名	输入/输出	说明
majorVersion	输出	主版本号的指针，从1开始，如果出现接口的不兼容变更时，加1。
minorVersion	输出	次版本号的指针，从0开始，按照迭代周期，有新增接口时加1。
patchVersion	输出	补丁版本号的指针，从0开始，表示本版本仅仅解决了问题，在majorVersion、minorVersion不变的情况下加1；但majorVersion、minorVersion增加的时候，patchVersion一般为0。

返回值说明

返回0表示成功，返回其它值表示失败。

11.4.4 aclSetCompileopt

函数功能

用户可调用本接口设置对应的编译选项，该选项为进程级全局共享，算子或模型开始编译时，以当前的编译选项为准，一次编译过程中不会变更。

用户可以在调用aclopCompile接口编译算子或aclopCompileAndExecute接口编译执行算子前，调用本接口设置编译选项。

函数原型

aclError aclSetCompileopt(aclCompileOpt opt, const char *value)

参数说明

参数名	输入/输出	说明
opt	输入	编译选项。
value	输入	编译选项具体值的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.4.5 aclGetCompileopt

函数功能

用户可调用本接口获取编译选项的值。

用户可以调用本接口获取当前编译选项的值，在调用**aclSetCompileopt**前，本接口返回的是各编译选项的默认值；在调用**aclSetCompileopt**后，本接口返回的是用户设置的值。

函数原型

aclError aclGetCompileopt(aclCompileOpt opt, char *value, size_t length)

参数说明

参数名	输入/输出	说明
opt	输入	编译选项。
value	输出	存放编译选项值的内存的指针。 用户在调用本前需要申请一块内存来保存编译选项的值，内存大小可以通过**aclGetCompileoptSize**获得。
length	输入	value指向内存的大小，单位：Byte。

返回值说明

返回0表示成功，返回其它值表示失败。

11.4.6 aclGetCompileoptSize

函数功能

用户可调用本接口获取存放编译选项值的内存大小。

用户在调用aclGetCompileopt前需要申请一块内存来保存编译选项的值，内存大小可以通过调用本接口获得。

函数原型

size_t aclGetCompileoptSize(aclCompileOpt opt)

参数说明

参数名	输入/输出	说明
opt	输入	编译选项。

返回值说明

返回存放该编译选项值的内存大小，单位：Byte。

11.4.7 aclrtGetSocName

函数功能

查询当前运行环境的芯片版本。

函数原型

const char *aclrtGetSocName()

参数说明

无

返回值说明

返回芯片版本字符串的指针。如果通过该接口获取芯片版本失败，则返回空指针。

11.4.8 aclGetRecentErrMsg

函数功能

获取并清空与本接口在同一个线程中的其它AscendCL接口调用失败时的错误描述信息。

约束说明

建议在每次调用AscendCL接口失败时都调用aclGetRecentErrMsg接口，以便获取调用AscendCL接口异常时的错误描述信息，用于定位问题。同一个线程中多次调用aclGetRecentErrMsg接口后，只有最后一次调用aclGetRecentErrMsg接口返回的错误描述字符串的指针有效，之前aclGetRecentErrMsg接口返回的错误描述字符串指针不能使用，否则可能导致内存非法访问。

如果未在每次调用AscendCL接口失败时调用aclGetRecentErrMsg接口，则可能导致错误信息堆积、丢失。

函数原型

const char *aclGetRecentErrMsg()

参数说明

无

返回值说明

返回错误描述字符串的指针。如果通过本接口获取到多条错误描述信息，最上面的错误描述信息为最新的。

获取错误描述信息失败时，返回nullptr。

11.4.9 aclrtSetDeviceSatMode

函数功能

设置当前Context对应的Device的浮点计算结果输出模式。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

调用该接口成功后，后续在该Device上新创建的Stream按设置的模式生效，对之前已创建的Stream不生效。

函数原型

aclError aclrtSetDeviceSatMode(aclrtFloatOverflowMode mode)

参数说明

参数名	输入/输出	说明
mode	输入	设置浮点计算结果输出模式。

返回值说明

返回0表示成功，返回其它值表示失败。

11.4.10 aclrtGetDeviceSatMode

函数功能

查询当前Context对应的Device的浮点计算结果输出模式。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

aclError aclrtGetDeviceSatMode(aclrtFloatOverflowMode *mode)

参数说明

参数名	输入/输出	说明
mode	输出	获取浮点计算结果输出模式。

返回值说明

返回0表示成功，返回其它值表示失败。

11.5 Device 管理

11.5.1 aclrtSetDevice

函数功能

指定当前线程中用于运算的Device，同时隐式创建默认Context。

对于Atlas 200/300/500 推理产品，该默认Context中包含2个Stream，1个默认Stream和1个执行内部同步的Stream。

对于Atlas 推理系列产品，在**Control CPU开放形态**下，该默认Context中包含1个默认Stream。

约束说明

如果多次调用aclrtSetDevice接口而不调用**aclrtResetDevice**接口释放本线程使用的Device资源，功能上不会有问题，因为在进程退出时也会释放本进程使用的Device资源。建议aclrtSetDevice接口和**aclrtResetDevice**接口配对使用，在不使用Device上资源时，通过调用**aclrtResetDevice**接口及时释放本进程使用的Device资源。

支持以下使用场景：

- 在不同进程或线程中支持调用aclrtSetDevice接口指定同一个Device用于运算。在同一个进程中的多个线程中，如果调用aclrtSetDevice接口指定同一个Device用于运算，这时隐式创建的默认Context是同一个。
- 多Device场景下，可在进程中通过aclrtSetDevice接口切换到其它Device。但利用Context切换（调用**aclrtSetCurrentContext**接口）来切换Device，比使用**aclrtSetDevice**接口效率高。

函数原型

aclError aclrtSetDevice(int32_t deviceId)

参数说明

参数名	输入/输出	说明
deviceId	输入	Device ID。 用户调用**aclrtGetDeviceCount**接口获取可用的Device数量后，这个Device ID的取值范围：[0, (可用的Device数量-1)]

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**4.4 运行管理资源申请与释放**。

11.5.2 aclrtResetDevice

函数功能

复位当前运算的Device，释放Device上的资源，包括默认Context、默认Stream以及默认Context下创建的所有Stream。若默认Context或默认Stream下的任务还未完成，系统会等待任务完成后再释放。

约束说明

若要复位的Device上存在显式创建的Context、Stream、Event，在复位前，建议遵循如下接口调用顺序，否则可能会导致业务异常。

接口调用顺序：调用**aclrtDestroyEvent**接口释放Event/调用**aclrtDestroyStream**接口释放显式创建的Stream-->调用**aclrtDestroyContext**释放显式创建的Context-->调用aclrtResetDevice接口

函数原型

aclError aclrtResetDevice(int32_t deviceId)

参数说明

参数名	输入/输出	说明
deviceId	输入	Device ID。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**4.4 运行管理资源申请与释放**。

11.5.3 aclrtGetDevice

函数功能

获取当前正在使用的Device的ID。

约束说明

如果没有调用**aclrtSetDevice**接口显式指定Device或没有调用**aclrtCreateContext**接口隐式指定Device，则调用aclrtGetDevice接口时，返回错误。

函数原型

aclError aclrtGetDevice(int32_t *deviceId)

参数说明

参数名	输入/输出	说明
deviceId	输出	Device ID的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.5.4 aclrtGetRunMode

函数功能

获取当前昇腾AI软件栈的运行模式。

函数原型

aclError aclrtGetRunMode(aclrtRunMode *runMode)

参数说明

参数名	输入/输出	说明
runMode	输出	运行模式的指针。 ● ACL_DEVICE：昇腾AI软件栈运行在Device的Control CPU或板端环境上。 ● ACL_HOST：昇腾AI软件栈运行在Host CPU上。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例代码，参见**4.4 运行管理资源申请与释放**。

11.5.5 aclrtSetTsDevice

函数功能

设置本次计算需要使用的Task Schedule。

约束说明

Atlas 200/300/500 推理产品中只有AICORE Task Schedule，无需使用本接口。

Atlas 推理系列产品中只有AICORE Task Schedule，无需使用本接口。

函数原型

aclError aclrtSetTsDevice(aclrtTsId tsId)

参数说明

参数名	输入/输出	说明

tsId	输入	指定本次计算需要使用的Task Schedule。如果昇腾AI软件栈中只有AICORE Task Schedule，没有AIVECTOR Task Schedule，则设置该参数无效，默认使用AICORE Task Schedule。
		typedef enum aclrtTsId {
		ACL_TS_ID_AICORE = 0, //使用AICORE Task Schedule
		ACL_TS_ID_AIVECTOR = 1, //使用AIVECTOR Task Schedule
		ACL_TS_ID_RESERVED = 2,
		} aclrtTsId;

返回值说明

返回0表示成功，返回其它值表示失败。

11.5.6 aclrtGetDeviceCount

函数功能

获取可用Device的数量。

函数原型

aclError aclrtGetDeviceCount(uint32_t *count)

参数说明

参数名	输入/输出	说明
count	输出	Device数量的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.5.7 aclrtGetDeviceUtilizationRate

函数功能

查询Device上Cube、Vector、AI CPU等的利用率。

约束说明

- 昇腾虚拟化实例场景下，不支持调用本接口查询利用率，接口返回值无实际意义。
- 开启Profiling功能时，不支持调用本接口查询利用率，接口返回值无实际意义。
- 查询Device内存利用率为预留功能，当前版本不支持，若调用本接口查询内存利用率，查询到的利用率为-1。
- Atlas 200/300/500 推理产品上没有Vector，调用本接口查询Vector利用率时，查询到的利用率为-1。

函数原型

aclError aclrtGetDeviceUtilizationRate(int32_t deviceId, aclrtUtilizationInfo *utilizationInfo)

参数说明

参数名	输入/输出	说明
deviceId	输入	Device ID。 用户调用aclrtGetDeviceCount接口获取可用的Device数量后，这个Device ID的取值范围：[0, (可用的Device数量-1)]
utilizationInfo	输出	利用率信息结构体指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.5.8 aclrtQueryDeviceStatus

函数功能

查询Device状态是正常可用、还是异常不可用。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

aclError aclrtQueryDeviceStatus(int32_t deviceId, aclrtDeviceStatus *deviceStatus)

参数说明

参数名	输入/输出	说明
deviceId	输入	Device ID。 用户调用aclrtGetDeviceCount接口获取可用的Device数量后，这个Device ID的取值范围：[0, (可用的Device数量-1)]
deviceStatus	输出	Device状态。

返回值说明

返回0表示成功，返回其它值表示失败。

11.5.9 aclrtSynchronizeDevice

函数功能

阻塞应用程序运行，直到正在运算中的Device完成运算。

函数原型

aclError aclrtSynchronizeDevice(void)

参数说明

无

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

- 接口调用流程，参见7.2 多Device切换。
- 接口调用示例，参见Device的同步等待示例代码。

11.6 Context 管理

11.6.1 aclrtCreateContext

函数功能

在当前进程或线程中显式创建一个Context。

对于Atlas 200/300/500 推理产品，该Context中包含2个Stream，1个默认Stream和1个执行内部同步的Stream。

对于Atlas 推理系列产品，在**Control CPU开放形态**下，该Context中包含1个默认Stream。

约束说明

支持以下使用场景：

- 若不调用aclrtCreateContext接口显式创建Context，那系统会使用默认Context，该默认Context是在调用**aclrtSetDevice**接口时隐式创建的。
  - 隐式创建Context：适合简单、无复杂交互逻辑的应用，但缺点在于，在多线程编程中，执行结果取决于线程调度的顺序。
  - 显式创建Context：**推荐显式**，适合大型、复杂交互逻辑的应用，且便于提高程序的可读性、可维护性。
- 在某一进程中指定Device，该进程内的多个线程可共用在此Device上显式创建的Context（调用**aclrtCreateContext**接口显式创建Context）。
- 若在某一进程内创建多个Context（Context的数量与Stream相关，Stream数量有限制，请参见**11.8.1 aclrtCreateStream**），当前线程在同一时刻内只能使用其中一个Context，建议通过**aclrtSetCurrentContext**接口明确指定当前线程的Context，增加程序的可维护性。

说明

对于Atlas 200/300/500 推理产品，如果在程序中没有调用**aclrtSetDevice**接口，那么在首次调用aclrtCreateContext接口时，系统内部会根据该接口传入的Device ID，为该Device绑定一个默认Stream（一个Device仅绑定一个默认Stream），因此仅在首次调用aclrtCreateContext接口时，会占用3个Stream：Device上绑定的默认Stream、Context内的默认Stream、Context内的用于执行内部同步的Stream。

对于Atlas 推理系列产品，如果在程序中没有调用**aclrtSetDevice**接口，那么在首次调用aclrtCreateContext接口时，系统内部会根据该接口传入的Device ID，为该Device绑定一个默认Stream（一个Device仅绑定一个默认Stream），因此仅在首次调用aclrtCreateContext接口时，会占用2个Stream：Device上绑定的默认Stream、Context内的默认Stream。

函数原型

aclError aclrtCreateContext(aclrtContext *context, int32_t deviceId)

参数说明

参数名	输入/输出	说明
deviceId	输入	需创建Context的Device的ID。
context	输出	Context的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**4.4 运行管理资源申请与释放**。

11.6.2 aclrtDestroyContext

函数功能

销毁一个Context，释放Context的资源。只能销毁通过**aclrtCreateContext**接口创建的Context。

函数原型

aclError aclrtDestroyContext(aclrtContext context)

参数说明

参数名	输入/输出	说明
context	输入	需销毁的Context。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**4.4 运行管理资源申请与释放**。

11.6.3 aclrtSetCurrentContext

函数功能

设置线程的Context。

约束说明

- 支持以下场景：
  - 如果在某线程（例如：thread1）中调用**aclrtCreateContext**接口显式创建一个Context（例如：ctx1），则可以不调用aclrtSetCurrentContext接口指定该线程的Context，系统默认将ctx1作为thread1的Context。
  - 如果没有调用**aclrtCreateContext**接口显式创建Context，则系统将默认Context作为线程的Context，此时，不能通过**aclrtDestroyContext**接口来释放默认Context。
  - 如果多次调用aclrtSetCurrentContext接口设置线程的Context，以最后一次为准。
- 若给线程设置的Context所对应的Device已经被复位，则不能将该Context设置为线程的Context，否则会导致业务异常。
- 推荐在某一线程中创建的Context，在该线程中使用。若在线程A中调用**aclrtCreateContext**接口创建Context，在线程B中使用该Context，则需由用户自行保证两个线程中同一个Context下同一个Stream中任务执行的顺序。

函数原型

aclError aclrtSetCurrentContext(aclrtContext context)

参数说明

参数名	输入/输出	说明
context	输入	指定线程当前的Context。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程，参见7.2 多Device切换。

11.6.4 aclrtGetCurrentContext

函数功能

获取线程的Context。

约束说明

如果用户多次调用aclrtSetCurrentContext接口设置当前线程的Context，则获取的是最后一次设置的Context。

函数原型

aclError aclrtGetCurrentContext(aclrtContext *context)

参数说明

参数名	输入/输出	说明
context	输出	线程当前Context的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.6.5 aclrtCtxSetSysParamOpt

函数功能

设置当前Context中的系统参数值。

约束说明

多次调用本接口，以最后一次设置的值为准。

函数原型

aclError aclrtCtxSetSysParamOpt(aclSysParamOpt opt, int64_t value)

参数说明

参数名	输入/输出	说明
opt	输入	系统参数。
value	输入	系统参数值。

返回值说明

返回0表示成功，返回其它值表示失败。

11.6.6 aclrtCtxGetSysParamOpt

函数功能

获取当前Context中的系统参数值。

约束说明

系统参数无默认值，如果不调用aclrtCtxSetSysParamOpt接口设置系统参数的值，直接调用本接口获取系统参数的值，接口会返回失败。

函数原型

aclError aclrtCtxGetSysParamOpt(aclSysParamOpt opt, int64_t *value)

参数说明

参数名	输入/输出	说明
opt	输入	系统参数。
value	输出	存放系统参数值的内存的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.6.7 aclrtGetOverflowStatus

函数功能

获取当前Context下所有Stream上任务的溢出状态，并将状态值拷贝到用户申请的Device内存中。异步接口。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 该接口是异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功。调用该接口后，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保任务已执行完成。

函数原型

aclError aclrtGetOverflowStatus(void *outputAddr, size_t outputSize, aclrtStream stream)

参数说明

参数名	输入/输出	说明
outputAddr	输入&输出	用户申请的Device内存，需通过aclrtMalloc接口申请。 如果需要在Host侧查看数据，可调用**aclrtMemcpy**或**aclrtMemcpyAsync**接口，将Device侧的数据传输到Host侧。
outputSize	输入	需申请的Device内存大小，单位Byte，固定大小为64Byte。
stream	输入	指定Stream，用于下发溢出状态查询任务。

返回值说明

返回0表示成功，返回其它值表示失败。

11.6.8 aclrtResetOverflowStatus

函数功能

清除当前Context下所有Stream上任务的溢出状态。异步接口。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 该接口是异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功。调用该接口后，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保任务已执行完成。

函数原型

aclError aclrtResetOverflowStatus(aclrtStream stream)

参数说明

参数名	输入/输出	说明
stream	输入	指定Stream，用于下发溢出状态复位任务。

返回值说明

返回0表示成功，返回其它值表示失败。

11.7 算力 Group 查询与设置

11.7.1 aclrtSetGroup

函数功能

指定当前运算使用哪个Group的算力，该接口必须在指定Context后调用。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- Atlas 推理系列产品，**Control CPU开放形态**下，支持调用本接口。
- 调用DSMI提供的dsmi_create_capability_group接口创建分组后，再调用AscendCL接口进行算力Group查询与设置。

  AscendCL接口调用顺序：调用**aclrtSetDevice**接口指定计算设备-->调用**aclrtGetAllGroupInfo**接口获取所有Group信息-->调用**aclrtGetGroupCount**接口获取Group数量-->调用**aclrtGetGroupInfoDetail**接口获取指定Group信息-->调用**aclrtSetGroup**接口设置分组-->执行其它任务-->调用**aclrtResetDevice**接口释放计算设备。

说明

若刷新分组（例如调用dsmi_create_capability_group接口新增分组、调用dsmi_delete_capability_group接口删除分组等），需重启业务AscendCL应用进程。

dsmi_create_capability_group接口和dsmi_delete_capability_group接口的详细说明请参见《DSMI API参考》。

函数原型

aclError aclrtSetGroup(int32_t groupId)

参数说明

参数名	输入/输出	说明
groupId	输入	表示Group的ID，用于指定当前计算要使用的Group。 您需要提前调用**aclrtGetGroupInfoDetail**接口获取Group的ID。

返回值说明

返回0表示成功，返回其它值表示失败。

11.7.2 aclrtGetGroupCount

函数功能

查询当前Context下可以使用的Group个数。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- Atlas 推理系列产品，**Control CPU开放形态**下，支持调用本接口。
- 调用DSMI提供的dsmi_create_capability_group接口创建分组后，再调用AscendCL接口进行算力Group查询与设置。

  AscendCL接口调用顺序：调用**aclrtSetDevice**接口指定计算设备-->调用**aclrtGetAllGroupInfo**接口获取所有Group信息-->调用**aclrtGetGroupCount**接口获取Group数量-->调用**aclrtGetGroupInfoDetail**接口获取指定Group信息-->调用**aclrtSetGroup**接口设置分组-->执行其它任务-->调用**aclrtResetDevice**接口释放计算设备。

  说明

  若刷新分组（例如调用dsmi_create_capability_group接口新增分组、调用dsmi_delete_capability_group接口删除分组等），需重启业务AscendCL应用进程。

  dsmi_create_capability_group接口和dsmi_delete_capability_group接口的详细说明请参见《DSMI API参考》。

函数原型

aclError aclrtGetGroupCount(uint32_t *count)

参数说明

参数名	输入/输出	说明
count	输出	当前Context下可用Group个数的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.7.3 aclrtGetAllGroupInfo

函数功能

查询当前Context下可以使用的所有Group的详细算力信息。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- Atlas 推理系列产品，**Control CPU开放形态**下，支持调用本接口。
- 调用DSMI提供的dsmi_create_capability_group接口创建分组后，再调用AscendCL接口进行算力Group查询与设置。

  AscendCL接口调用顺序：调用**aclrtSetDevice**接口指定计算设备-->调用**aclrtGetAllGroupInfo**接口获取所有Group信息-->调用**aclrtGetGroupCount**接口获取Group数量-->调用**aclrtGetGroupInfoDetail**接口获取指定Group信息-->调用**aclrtSetGroup**接口设置分组-->执行其它任务-->调用**aclrtResetDevice**接口释放计算设备。

说明

若刷新分组（例如调用dsmi_create_capability_group接口新增分组、调用dsmi_delete_capability_group接口删除分组等），需重启业务AscendCL应用进程。

dsmi_create_capability_group接口和dsmi_delete_capability_group接口的详细说明请参见《DSMI API参考》。

函数原型

aclError aclrtGetAllGroupInfo(aclrtGroupInfo *groupInfo)

参数说明

参数名	输入/输出	说明
groupInfo	输出	获取所有Group对应的详细算力信息的指针。 需提前调用**aclrtCreateGroupInfo**接口创建aclrtGroupInfo类型的数据。

返回值说明

返回0表示成功，返回其它值表示失败。

11.7.4 aclrtGetGroupInfoDetail

函数功能

查询当前Context下指定Group的算力信息。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- Atlas 推理系列产品，**Control CPU开放形态**下，支持调用本接口。
- 调用DSMI提供的dsmi_create_capability_group接口创建分组后，再调用AscendCL接口进行算力Group查询与设置。

  AscendCL接口调用顺序：调用**aclrtSetDevice**接口指定计算设备-->调用**aclrtGetAllGroupInfo**接口获取所有Group信息-->调用**aclrtGetGroupCount**接口获取Group数量-->调用**aclrtGetGroupInfoDetail**接口获取指定Group信息-->调用**aclrtSetGroup**接口设置分组-->执行其它任务-->调用**aclrtResetDevice**接口释放计算设备。

说明

若刷新分组（例如调用dsmi_create_capability_group接口新增分组、调用dsmi_delete_capability_group接口删除分组等），需重启业务AscendCL应用进程。

dsmi_create_capability_group接口和dsmi_delete_capability_group接口的详细说明请参见《DSMI API参考》。

函数原型

aclError aclrtGetGroupInfoDetail(const aclrtGroupInfo *groupInfo, int32_t groupIndex, aclrtGroupAttr attr, void *attrValue, size_t valueLen, size_t *paramRetSize)

参数说明

参数名	输入/输出	说明
groupInfo	输入	指定算力详细信息的首地址的指针。 需提前调用**aclrtGetAllGroupInfo**接口创获取所有Group的算力信息。
groupIndex	输入	访问groupInfo连续内存块的Group索引。 Group索引的取值范围：[0, (Group数量-1)]，用户可调用**aclrtGetGroupCount**接口获取Group数量。
attr	输入	指定要获取其算力值的算力属性。

参数名	输入/输出	说明
attrValue	输出	获取指定算力属性所对应的算力值的指针。 用户需根据每个属性的属性值数据类型申请对应大小的内存，用于存放属性值。
valueLen	输入	表示attrValue的最大长度，单位为Byte。
paramRetSize	输出	实际返回的attrValue大小的指针，单位为Byte。

返回值说明

返回0表示成功，返回其它值表示失败。

11.7.5 aclrtCreateGroupInfo

函数功能

根据实际支持的Group数量创建aclrtGroupInfo类型的连续内存块，并返回对应指针。

如需销毁aclrtGroupInfo类型的数据，请参见**aclrtDestroyGroupInfo**。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

Atlas 推理系列产品，**Control CPU开放形态**下，支持调用本接口。

函数原型

aclrtGroupInfo *aclrtCreateGroupInfo()

参数说明

无

返回值说明

返回aclrtGroupInfo类型的指针，如果无Group或不支持Group则返回nullptr。

11.7.6 aclrtDestroyGroupInfo

函数功能

销毁aclrtGroupInfo类型的数据，释放相关的内存。只能销毁通过**aclrtCreateGroupInfo**接口创建的aclrtGroupInfo类型。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

Atlas 推理系列产品，**Control CPU开放形态**下，支持调用本接口。

函数原型

aclError aclrtDestroyGroupInfo(aclrtGroupInfo *groupInfo)

参数说明

参数名	输入/输出	说明
groupInfo	输入	待销毁的aclrtGroupInfo类型数据的指针。

返回值说明

返回0表示成功，非零表示失败。

11.8 Stream 管理

11.8.1 aclrtCreateStream

函数功能

在当前进程或线程中创建一个Stream。

约束说明

- 每个Context对应一个默认Stream，该默认Stream是调用**aclrtSetDevice**接口或**aclrtCreateContext**接口隐式创建的。推荐调用aclrtCreateStream接口显式创建Stream。
  - 隐式创建Stream：适合简单、无复杂交互逻辑的应用，但缺点在于，在多线程编程中，执行结果取决于线程调度的顺序。
  - 显式创建Stream：**推荐显式**，适合大型、复杂交互逻辑的应用，且便于提高程序的可读性、可维护性。
- 对于Atlas 200/300/500 推理产品，硬件资源最多支持1024个Stream，如果已存在多个默认Stream，只能显式创建N个Stream（N=1024-默认Stream个数-执行内部同步的Stream个数），例如，若已存在一个默认Stream和一个执行内部同步的Stream，则只能显式创建1022个Stream。
- 对于Atlas 推理系列产品，硬件资源最多支持1024个Stream，
  - **Control CPU开放形态**下，如果已存在多个默认Stream，只能显式创建N个Stream（N=1024-默认Stream个数），例如，若已存在一个默认Stream，则只能显式创建1023个Stream。

函数原型

aclError aclrtCreateStream(aclrtStream *stream)

参数说明

参数名	输入/输出	说明
stream	输出	Stream的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**4.4 运行管理资源申请与释放**。

11.8.2 aclrtCreateStreamV2

函数功能

在当前进程或线程中创建一个Stream，支持创建Stream时增加Stream配置。

Atlas 200/300/500 推理产品不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 本接口需要配合其它接口一起使用，创建Stream，接口调用顺序如下：
  a. 调用**aclrtCreateStreamConfigHandle**接口创建Stream配置对象。
  b. 多次调用**aclrtSetStreamConfigOpt**接口设置配置对象中每个属性的值。
  c. 调用aclrtCreateStreamV2接口创建Stream。
  d. Stream使用完成后，调用**aclrtDestroyStreamConfigHandle**接口销毁Stream配置对象，调用**aclrtDestroyStream**接口销毁Stream。

函数原型

aclError aclrtCreateStreamV2(aclrtStream *stream, const aclrtStreamConfigHandle *handle)

参数说明

参数名	输入/输出	说明
stream	输出	Stream的指针。

参数名	输入/输出	说明
handle	输入	Stream配置对象的指针。与aclrtSetStreamConfigOpt中的handle保持一致。

返回值说明

返回0表示成功，返回其它值表示失败。

11.8.3 aclrtSetStreamConfigOpt

函数功能

设置Stream配置对象中的各属性的取值。

Atlas 200/300/500 推理产品不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

本接口需要配合其它接口一起使用，创建Stream，接口调用顺序如下：

1. 调用aclrtCreateStreamConfigHandle接口创建Stream配置对象。
2. 多次调用aclrtSetStreamConfigOpt接口设置配置对象中每个属性的值。
3. 调用aclrtCreateStreamV2接口创建Stream。
4. Stream使用完成后，调用aclrtDestroyStreamConfigHandle接口销毁Stream配置对象，调用aclrtDestroyStream接口销毁Stream。

函数原型

aclError aclrtSetStreamConfigOpt(aclrtStreamConfigHandle *handle, aclrtStreamConfigAttr attr, const void *attrValue, size_t valueSize)

参数说明

参数名	输入/输出	说明
handle	输出	Stream配置对象的指针。需提前调用aclrtCreateStreamConfigHandle接口创建该对象。
attr	输入	指定需设置的属性。
attrValue	输入	指向属性值的指针，attr对应的属性取值。 如果属性值本身是指针，则传入该指针的地址。

参数名	输入/输出	说明
valueSize	输入	attrValue部分的数据长度。 用户可使用C/C++标准库的函数sizeof(*attrValue)查询数据长度。

返回值说明

返回0表示成功，返回其它值表示失败。

11.8.4 aclrtCreateStreamWithConfig

函数功能

在当前进程或线程中创建一个Stream。

在Atlas 200/300/500 推理产品上，使用本接口与**aclrtCreateStream**接口是等价的。

在Atlas 推理系列产品上，相比**aclrtCreateStream**接口，使用本接口可以创建一个快速下发任务的Stream，但会增加内存消耗或CPU的性能消耗。

函数原型

aclError aclrtCreateStreamWithConfig(aclrtStream *stream, uint32_t priority, uint32_t flag)

参数说明

参数名	输入/输出	说明
stream	输出	Stream的指针。
priority	输入	优先级。 Atlas 200/300/500 推理产品上，当前固定设置为0，预留参数，暂不使用。 Atlas 推理系列产品上，该参数取值范围：[0, 7]，总共最多支持8个优先级，数字越小代表优先级越高，其中，0的优先级最高，7的优先级最低。配置取值范围以外的值，本接口返回报错。

参数名	输入/输出	说明
flag	输入	Stream指针的flag。 取值范围： ● **ACL_STREAM_FAST_LAUNCH**：使用该模式创建出来的Stream，在使用Stream时，下发任务的速度更快。相比**aclrtCreateStream**接口创建出来的Stream，在使用Stream时才会申请系统内部资源，导致下发任务的时长增加，使用本接口的**ACL_STREAM_FAST_LAUNCH**模式创建Stream时，会在创建Stream时预申请系统内部资源，因此创建Stream的时长增加，下发任务的时长缩短，总体来说，创建一次Stream，使用多次的场景下，总时长缩短，但创建Stream时预申请内部资源会增加内存消耗。 ● **ACL_STREAM_FAST_SYNC**：使用该模式创建出来的Stream，在调用**aclrtSynchronizeStream**接口时，会阻塞当前线程，主动查询任务的执行状态，一旦任务完成，立即返回。 相比**aclrtCreateStream**接口创建出来的Stream，在调用**aclrtSynchronizeStream**接口时，会一直被动等待Device上任务执行完成的通知，等待时间长，使用本接口的**ACL_STREAM_FAST_SYNC**模式创建的Stream，没有被动等待，总时长缩短，但主动查询的操作会增加CPU的性能消耗。 **说明** 配置取值范围之外的值，本接口创建出来的Stream等同于**aclrtCreateStream**接口。

返回值说明

返回0表示成功，返回其它值表示失败。

11.8.5 aclrtDestroyStream

函数功能

销毁指定Stream，销毁通过**aclrtCreateStream**或**aclrtCreateStreamWithConfig**或**aclrtCreateStreamV2**接口创建的Stream，若Stream上有未完成的任务，会等待任务完成后再销毁Stream。

约束说明

- 在调用aclrtDestroyStream接口销毁指定Stream前，需要先调用**aclrtSynchronizeStream**接口确保Stream中的任务都已完成。
- 调用aclrtDestroyStream接口销毁指定Stream时，需确保该Stream在当前Context下。
- 在调用aclrtDestroyStream接口销毁指定Stream时，需确保其它接口没有正在使用该Stream。

函数原型

aclError aclrtDestroyStream(aclrtStream stream)

参数说明

参数名	输入/输出	说明
stream	输入	待销毁的Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**4.4 运行管理资源申请与释放**。

11.8.6 aclrtDestroyStreamForce

函数功能

销毁指定Stream，销毁通过**aclrtCreateStream**或**aclrtCreateStreamWithConfig**接口创建的Stream，若Stream上有未完成的任务，不会等待任务完成，直接强制销毁Stream。

约束说明

调用本接口销毁指定Stream时，需确保该Stream在当前Context下。

函数原型

aclError aclrtDestroyStreamForce(aclrtStream stream)

参数说明

参数名	输入/输出	说明
stream	输入	待销毁的Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

11.8.7 aclrtSetStreamOverflowSwitch

函数功能

饱和模式下，对接上层训练框架时（例如PyTorch），针对指定Stream，打开或关闭溢出检测开关，关闭后无法通过溢出检测算子获取任务是否溢出。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 在调用本接口前，可调用**aclrtSetDeviceSatMode**接口设置饱和模式。
- 调用该接口打开或关闭溢出检测开关后，仅对后续新下的任务生效，已下发的任务仍维持原样。

函数原型

aclError aclrtSetStreamOverflowSwitch(aclrtStream stream, uint32_t flag)

参数说明

参数名	输入/输出	说明
stream	输入	待操作Stream，若传入NULL，则操作默认Stream。
flag	输入	溢出检测开关，取值范围如下： ● 0：关闭 ● 1：打开

返回值说明

返回0表示成功，返回其它值表示失败。

11.8.8 aclrtGetStreamOverflowSwitch

函数功能

针对指定Stream，获取其当前溢出检测开关是否打开。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

aclError aclrtGetStreamOverflowSwitch(aclrtStream stream, uint32_t *flag)

参数说明

参数名	输入/输出	说明
stream	输入	待操作Stream，若传入NULL，则操作默认Stream。
flag	输出	溢出检测开关，取值范围如下： ● 0：关闭 ● 1：打开

返回值说明

返回0表示成功，返回其它值表示失败。

11.8.9 aclrtSetStreamFailureMode

函数功能

当一个Stream上下发了多个任务时，可通过本接口指定任务调度模式，以便控制某个任务失败后是否继续执行下一个任务。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

aclError aclrtSetStreamFailureMode(aclrtStream stream, uint64_t mode)

参数说明

参数名	输入/输出	说明
stream	输入	待操作Stream。 不支持指定默认Stream（即该参数传入NULL）的任务调度模式。
mode	输入	当一个Stream上下发了多个任务时，可通过本参数指定任务调度模式，以便控制某个任务失败后是否继续执行下一个任务。 取值范围如下： ● ACL_CONTINUE_ON_FAILURE：默认值，某个任务失败后，继续执行下一个任务； ● ACL_STOP_ON_FAILURE：某个任务失败后，停止执行后续的任务。

返回值说明

返回0表示成功，返回其它值表示失败。

11.8.10 aclrtStreamQuery

函数功能

查询指定Stream上的所有任务是否已经执行完成。

函数原型

aclError aclrtStreamQuery(aclrtStream stream, aclrtStreamStatus *status)

参数说明

参数名	输入/输出	说明
stream	输入	Stream的指针。
status	输出	Stream上的任务状态。

返回值说明

返回0表示成功，返回其它值表示失败。

11.8.11 aclrtSynchronizeStream

函数功能

阻塞应用程序运行，直到指定Stream中的所有任务都完成。

函数原型

aclError aclrtSynchronizeStream(aclrtStream stream)

参数说明

参数名	输入/输出	说明
stream	输入	指定需要完成所有任务的Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见**Stream内任务的同步等待示例代码**。

11.8.12 aclrtSynchronizeStreamWithTimeout

函数功能

阻塞应用程序运行，直到指定Stream中的所有任务都完成，该接口是在aclrtSynchronizeStream接口基础上进行了增强，支持用户设置超时时间，当应用程序异常时可根据所设置的超时时间自行退出。

函数原型

aclError aclrtSynchronizeStreamWithTimeout(aclrtStream stream, int32_t timeout)

参数说明

参数名	输入/输出	说明
stream	输入	指定需要完成所有任务的Stream。
timeout	输入	接口的超时时间。 取值说明如下： ● -1：表示永久等待，和接口aclrtSynchronizeStream功能一样； ● >0：配置具体的超时时间，单位是毫秒。

返回值说明

返回0表示成功，返回其它值表示失败。

11.9 Event 管理

11.9.1 aclrtCreateEvent

函数功能

创建一个Event，创建出来的Event可用于统计两个Event之间的耗时、多Stream场景下的同步等待等场景。

函数原型

aclError aclrtCreateEvent(aclrtEvent *event)

参数说明

参数名	输入/输出	说明
event	输出	Event的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

相关接口

您也可以使用**aclrtCreateEventWithFlag**接口创建一个带标识的Event，不同标识的Event用于不同的功能。

参考资源

- 接口调用流程，参见**多Stream时的同步等待流程**。
- 接口调用示例，参见**Event的同步等待示例代码**、**Stream间任务的同步等待示例代码**。

11.9.2 aclrtCreateEventWithFlag

函数功能

创建一个带flag的Event，不同flag的Event用于不同的功能。支持创建Event时携带多个flag（按位进行或操作），从而同时使能对应flag的功能。

约束说明

调用本接口创建Event时，flag为bitmap，支持将flag设置为单个宏、或者对多个宏进行或操作，若将flag参数值包含ACL_EVENT_SYNC宏，则创建出来的Event数量受限，具体如下：

Atlas 200/300/500 推理产品，单个Device上最多支持1023个Event。

Atlas 推理系列产品，单个Device上最多支持1023个Event。

函数原型

aclError aclrtCreateEventWithFlag(aclrtEvent *event, uint32_t flag)

参数说明

参数名	输入/输出	说明
event	输出	Event的指针。

参数名	输入/输出	说明

flag	输入	Event指针的flag。
		当前支持将flag设置为如下宏：
		● ACL_EVENT_TIME_LINE
		使能该bit表示创建的Event需要记录时间戳信息。注意：使能时间戳功能会影响Event相关接口的性能。
		`#define ACL_EVENT_TIME_LINE 0x00000008u`
		● ACL_EVENT_SYNC
		使能该bit表示创建的Event支持多Stream间的同步。
		`#define ACL_EVENT_SYNC 0x00000001u`

返回值说明

返回0表示成功，返回其它值表示失败。

相关接口

您也可以使用**aclrtCreateEvent**接口创建Event，Event数量有限，创建出来的Event可用于统计两个Event之间的耗时、多Stream场景下的同步等待、Event内的同步等待等场景。

11.9.3 aclrtDestroyEvent

函数功能

销毁一个Event，只能销毁通过**aclrtCreateEvent**/**aclrtCreateEventWithFlag**接口创建的Event。调用**aclrtCreateEvent**接口创建的Event，在销毁时，用户需确保等待**aclrtSynchronizeEvent**接口或**aclrtStreamWaitEvent**接口涉及的任务都结束后，再销毁。

约束说明

在调用aclrtDestroyEvent接口销毁指定Event时，需确保其它接口没有正在使用该Event。

函数原型

aclError aclrtDestroyEvent(aclrtEvent event)

参数说明

参数名	输入/输出	说明
event	输入	待销毁的Event。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

- 接口调用流程，参见**多Stream时的同步等待流程**。
- 接口调用示例，参见**Event的同步等待示例代码**、**Stream间任务的同步等待示例代码**。

11.9.4 aclrtRecordEvent

函数功能

在Stream中记录一个Event。本接口被调用时，AscendCL会捕获当前Stream上已下发的任务，并记录到Event事件中，因此后续若调用**aclrtQueryEventStatus**或**aclrtStreamWaitEvent**接口时，会检查或等待该Event事件中所捕获的任务都已经完成。

约束说明

- 该接口是异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功。调用该接口后，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保任务已执行完成。
- **aclrtRecordEvent**接口与**aclrtStreamWaitEvent**接口配合使用时，主要用于多Stream之间同步的场景，在调用aclrtRecordEvent接口时，系统内部会申请Event资源，在调用**aclrtStreamWaitEvent**接口之后，请及时调用**aclrtResetEvent**接口释放Event资源。

  接口调用顺序：**aclrtCreateEvent-->aclrtRecordEvent-->aclrtStreamWaitEvent-->aclrtResetEvent**
- 由于在调用aclrtRecordEvent接口，系统内部会申请Event资源，因此会受Event数量的限制，具体如下：

  Atlas 200/300/500 推理产品，单个Device上最多支持1023个Event。

  Atlas 推理系列产品，单个Device上最多支持1023个Event。

函数原型

aclError aclrtRecordEvent(aclrtEvent event, aclrtStream stream)

参数说明

参数名	输入/输出	说明
event	输入	待记录的Event。
stream	输入	将指定Event记录在指定的Stream中。 如果使用默认Stream，此处设置为NULL。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

- 接口调用流程，参见**多Stream时的同步等待流程**。
- 接口调用示例，参见**Event的同步等待示例代码**、**Stream间任务的同步等待示例代码**。

11.9.5 aclrtResetEvent

函数功能

复位一个Event。用户需确保等待Stream中的任务都完成后，再复位Event，异步接口。

约束说明

- 该接口是异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功。调用该接口后，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保任务已执行完成。
- 在复位Event时，涉及重置Event的状态，Event的状态是在调用**aclrtRecordEvent**接口时记录，因此在调用aclrtResetEvent接口前，需要先调用**aclrtRecordEvent**接口。

  接口调用顺序：aclrtCreateEvent-->aclrtRecordEvent-->aclrtResetEvent**-->aclrtSynchronizeStream**

函数原型

aclError aclrtResetEvent(aclrtEvent event, aclrtStream stream)

参数说明

参数名	输入/输出	说明
event	输入	待复位的Event。
stream	输入	指定Event所在的Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

11.9.6 aclrtQueryEvent

须知

此接口后续版本会废弃，请使用**aclrtQueryEventStatus**接口。

函数功能

查询与本接口在同一线程中的**aclrtRecordEvent**接口所记录的Event是否执行完成。

函数原型

aclError aclrtQueryEvent(aclrtEvent event, aclrtEventStatus *status)

参数说明

参数名	输入/输出	说明
event	输入	指定待查询的Event。
status	输出	Event状态的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.9.7 aclrtQueryEventStatus

函数功能

查询该Event捕获的所有任务的执行状态。具体见**aclrtRecordEvent**接口参考Event捕获的细节。Event和Stream必须在同一个Device中，可调用aclrtQueryEventStatus或aclrtStreamWaitEvent检查或等待Event捕获的任务是否已完成。

函数原型

aclError aclrtQueryEventStatus(aclrtEvent event, aclrtEventRecordedStatus *status)

参数说明

参数名	输入/输出	说明
event	输入	指定待查询的Event。
status	输出	Event状态的指针。 如果该Event捕获的所有任务都已经执行完成则返回ACL_EVENT_RECORDED_STATUS_COMPLETE，如果有任何一个任务未执行完成则返回ACL_EVENT_RECORDED_STATUS_NOT_READY。

返回值说明

返回0表示成功，返回其它值表示失败。

11.9.8 aclrtQueryEventWaitStatus

函数功能

调用**aclrtStreamWaitEvent**接口后查询该Event对应的等待任务是否都执行完成。

函数原型

aclError aclrtQueryEventWaitStatus(aclrtEvent event, aclrtEventWaitStatus *status)

参数说明

参数名	输入/输出	说明
event	输入	指定待查询的Event。
status	输出	Event状态的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.9.9 aclrtSynchronizeEvent

函数功能

阻塞当前线程运行直到Event捕获的所有任务都执行完成。具体见**aclrtRecordEvent**接口参考Event捕获的细节。

函数原型

aclError aclrtSynchronizeEvent(aclrtEvent event)

参数说明

参数名	输入/输出	说明
event	输入	需等待的Event。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见**Event的同步等待示例代码**。

11.9.10 aclrtSynchronizeEventWithTimeout

函数功能

阻塞当前线程运行直到Event捕获的所有任务都执行完成（具体见**aclrtRecordEvent**接口参考Event捕获的细节），该接口是在接口**aclrtSynchronizeEvent**基础上进行了增强，支持用户设置永久等待、或配置具体的超时时间，若配置具体的超时时间，则当应用程序异常时可根据所设置的超时时间自行退出。

约束说明

Atlas 200/300/500 推理产品，不支持调用接口配置具体的超时时间，仅支持设置永久等待。

函数原型

aclError aclrtSynchronizeEventWithTimeout(aclrtEvent event, int32_t timeout)

参数说明

参数名	输入/输出	说明
event	输入	需等待的Event。
timeout	输入	接口的超时时间。 取值说明如下： ● -1：表示永久等待，和接口**aclrtSynchronizeEvent**功能一样； ● >0：配置具体的超时时间，单位是毫秒。

返回值说明

返回0表示成功，返回其它值表示失败。

11.9.11 aclrtEventElapsedTime

函数功能

统计两个Event之间的耗时。

约束说明

接口调用顺序：调用**aclrtCreateEvent**/**aclrtCreateEventWithFlag**接口创建Event-->调用**aclrtRecordEvent**接口在同一个Stream中记录起始Event、结尾Event-->调用

aclrtSynchronizeStream接口阻塞应用程序运行，直到指定Stream中的所有任务都完成-->调用aclrtEventElapsedTime接口统计两个Event之间的耗时

函数原型

aclError aclrtEventElapsedTime(float *ms, aclrtEvent startEvent, aclrtEvent endEvent)

参数说明

参数名	输入/输出	说明
ms	输出	表示两个Event之间耗时的指针，单位为毫秒。
start	输入	起始Event。
end	输入	结尾Event。

返回值说明

返回0表示成功，返回其它值表示失败。

11.9.12 aclrtStreamWaitEvent

函数功能

阻塞指定Stream的运行，直到指定的Event完成，支持多个Stream等待同一个Event的场景。异步接口。

约束说明

- 该接口是异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功。调用该接口后，需调用同步等待接口（例如，aclrtSynchronizeStream）确保任务已执行完成。
- aclrtRecordEvent接口与aclrtStreamWaitEvent接口配合使用时，主要用于多Stream之间同步的场景，在调用aclrtRecordEvent接口时，系统内部会申请Event资源，在调用aclrtStreamWaitEvent接口之后，请及时调用aclrtResetEvent接口释放Event资源。

  接口调用顺序：aclrtCreateEvent-->aclrtRecordEvent-->aclrtStreamWaitEvent-->aclrtResetEvent
- 一个进程内，调用aclInit接口初始化AscendCL后，调用aclrtSetOpWaitTimeout接口设置超时时间，本进程内后续调用aclrtStreamWaitEvent接口下发的任务支持在所设置的超时时间内等待，若等待的时间超过所设置的超时时间，则AscendCL会返回报错。

  由于aclrtStreamWaitEvent接口是异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功，因此若等待的时间超过所设置的超时时间，则在调用aclrtSynchronizeStream接口后，会返回报错。

函数原型

aclError aclrtStreamWaitEvent(aclrtStream stream, aclrtEvent event)

参数说明

参数名	输入/输出	说明
stream	输入	指定需要等待Event完成的Stream。 如果使用默认Stream，此处设置为NULL。
event	输入	需等待的Event。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

- 接口调用流程，参见**多Stream时的同步等待流程**。
- 接口调用示例，参见**Event的同步等待示例代码**、**Stream间任务的同步等待示例代码**。

11.9.13 aclrtSetOpWaitTimeout

函数功能

本接口用于设置等待Event完成的超时时间。

Atlas 推理系列产品，不支持该接口。

约束说明

- 不调用本接口，则默认不超时；一个进程内多次调用本接口，则以最后一次设置的时间为准。
- 一个进程内，调用**aclInit**接口初始化AscendCL后，调用本接口设置超时时间，本进程内后续调用**aclrtStreamWaitEvent**接口下发的任务支持在所设置的超时时间内等待。

  由于aclrtStreamWaitEvent接口是异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功，因此若等待的时间超过所设置的超时时间，则在调用**aclrtSynchronizeStream**接口后，会返回报错。

函数原型

aclError aclrtSetOpWaitTimeout(uint32_t timeout)

参数说明

参数名	输入/输出	说明
timeout	输入	设置超时时间，单位为秒。 将该参数设置为0时，表示不超时。

返回值说明

返回0表示成功，返回其它值表示失败。

11.10 执行控制

11.10.1 aclrtSubscribeReport

函数功能

异步任务场景下，注册处理Stream上回调函数的线程。

本接口需与以下其它接口配合使用，以便实现异步场景下的callback功能：

1. 定义并实现回调函数，函数原型为：typedef void (*aclrtCallback)(void *userData)；
2. 新建线程，在线程函数内，调用**aclrtProcessReport**接口设置超时时间（需循环调用），等待回调任务执行；
3. 调用**aclrtSubscribeReport**接口建立第2步中的线程和Stream的绑定关系，该Stream下发的回调函数将在绑定的线程中执行；
4. 在指定Stream上执行异步任务（例如异步推理任务）；
5. 调用**aclrtLaunchCallback**接口在Stream的任务队列中下发回调任务，触发第2步中注册的线程处理回调函数，每调用一次aclrtLaunchCallback接口，就会触发一次回调函数的执行；
6. 异步任务全部执行完成后，取消线程注册（**aclrtUnSubscribeReport**接口）。

约束说明

- 支持多次调用aclrtSubscribeReport接口给多个Stream（仅支持同一Device内的多个Stream）注册同一个处理回调函数的线程；
- 为确保Stream内的任务按调用顺序执行，不支持调用aclrtSubscribeReport接口给同一个Stream注册多个处理回调函数的线程；
- 在Atlas 200/300/500 推理产品上，单进程内调用aclrtSubscribeReport接口注册的线程数量如果超过128个，则接口返回失败；
- 在Atlas 推理系列产品上，单进程内调用aclrtSubscribeReport接口注册的线程数量如果超过1024个，则接口返回失败；
- 考虑操作系统的线程切换性能开销，建议调用aclrtSubscribeReport接口注册的线程数量控制在32个以下（包括32）；
- 同一个进程内，在不同的Device上注册回调函数的线程时，不能指定同一个线程ID。

函数原型

aclError aclrtSubscribeReport(uint64_t threadId, aclrtStream stream)

参数说明

参数名	输入/输出	说明
threadId	输入	指定线程的ID。
stream	输入	指定Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例代码，参见**7.6 异步模型推理**。

11.10.2 aclrtLaunchCallback

函数功能

异步任务场景下，在Stream的任务队列中下发一个回调任务，系统内部在执行到该回调任务时，会在Stream上注册的线程（通过**aclrtSubscribeReport**接口注册的线程）中执行回调函数。异步接口。

本接口需与以下其它接口配合使用，以便实现异步场景下的callback功能：

1. 定义并实现回调函数，函数原型为：typedef void (*aclrtCallback)(void *userData)；
2. 新建线程，在线程函数内，调用**aclrtProcessReport**接口设置超时时间（需循环调用），等待回调任务执行；
3. 调用**aclrtSubscribeReport**接口建立第2步中的线程和Stream的绑定关系，该Stream下发的回调函数将在绑定的线程中执行；
4. 在指定Stream上执行异步任务（例如异步推理任务）；
5. 调用**aclrtLaunchCallback**接口在Stream的任务队列中下发回调任务，触发第2步中注册的线程处理回调函数，每调用一次aclrtLaunchCallback接口，就会触发一次回调函数的执行；
6. 异步任务全部执行完成后，取消线程注册（**aclrtUnSubscribeReport**接口）。

约束说明

该接口是异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功。调用该接口后，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保任务已执行完成。

函数原型

aclError aclrtLaunchCallback(aclrtCallback fn, void *userData, aclrtCallbackBlockType blockType, aclrtStream stream)

参数说明

参数名	输入/输出	说明
fn	输入	指定要增加的回调函数。 回调函数的函数原型为： typedef void (*aclrtCallback)(void *userData)
userData	输入	待传递给回调函数的用户数据的指针。
blockType	输入	指定回调函数调用是否阻塞Device运行。 typedef enum aclrtCallbackBlockType { ACL_CALLBACK_NO_BLOCK, //非阻塞，Atlas 200/300/500 推理产品不支持该选项，该选项预留 ACL_CALLBACK_BLOCK, //阻塞 } aclrtCallbackBlockType;
stream	输入	指定Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例代码，参见**7.6 异步模型推理**。

11.10.3 aclrtProcessReport

函数功能

异步任务场景下，调用本接口设置超时时间，等待**aclrtLaunchCallback**接口下发的回调任务执行。

本接口需与以下其它接口配合使用，以便实现异步场景下的callback功能：

1. 定义并实现回调函数，函数原型为：typedef void (*aclrtCallback)(void *userData)；
2. 新建线程，在线程函数内，调用**aclrtProcessReport**接口设置超时时间（需循环调用），等待回调任务执行；
3. 调用**aclrtSubscribeReport**接口建立第2步中的线程和Stream的绑定关系，该Stream下发的回调函数将在绑定的线程中执行；
4. 在指定Stream上执行异步任务（例如异步推理任务）；
5. 调用**aclrtLaunchCallback**接口在Stream的任务队列中下发回调任务，触发第2步中注册的线程处理回调函数，每调用一次aclrtLaunchCallback接口，就会触发一次回调函数的执行；

6. 异步任务全部执行完成后，取消线程注册（**aclrtUnSubscribeReport**接口）。

函数原型

aclError aclrtProcessReport(int32_t timeout)

参数说明

参数名	输入/输出	说明
timeout	输入	超时时间，单位为ms。 取值范围： ● -1：表示无限等待 ● 大于0（不包含0）：表示等待的时间

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例代码，参见**7.6 异步模型推理**。

11.10.4 aclrtUnSubscribeReport

函数功能

异步任务场景下，取消线程注册，Stream上的回调函数不再由指定线程处理。

本接口需与以下其它接口配合使用，以便实现异步场景下的callback功能：

1. 定义并实现回调函数，函数原型为：typedef void (*aclrtCallback)(void *userData)；
2. 新建线程，在线程函数内，调用**aclrtProcessReport**接口设置超时时间（需循环调用），等待回调任务执行；
3. 调用**aclrtSubscribeReport**接口建立第2步中的线程和Stream的绑定关系，该Stream下发的回调函数将在绑定的线程中执行；
4. 在指定Stream上执行异步任务（例如异步推理任务）；
5. 调用**aclrtLaunchCallback**接口在Stream的任务队列中下发回调任务，触发第2步中注册的线程处理回调函数，每调用一次aclrtLaunchCallback接口，就会触发一次回调函数的执行；
6. 异步任务全部执行完成后，取消线程注册（**aclrtUnSubscribeReport**接口）。

函数原型

aclError aclrtUnSubscribeReport(uint64_t threadId, aclrtStream stream)

参数说明

参数名	输入/输出	说明
threadId	输入	指定线程的ID。
stream	输入	指定Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例代码，参见**7.6 异步模型推理**。

11.10.5 aclrtSubscribeHostFunc

函数功能

模型中有CPU算子、且调用aclmdlExecuteV2或aclmdlExecuteAsyncV2接口执行模型推理时，调用本接口注册处理Stream上回调函数的线程（线程需由用户自行创建），再配合调用**aclrtProcessHostFunc**接口触发回调函数、在模型执行之后调用**aclrtUnSubscribeHostFunc**接口取消注册。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 支持多次调用aclrtSubscribeHostFunc接口给多个Stream（仅支持同一Device内的多个Stream）注册同一个处理回调函数的线程。
- 为确保Stream内的任务按调用顺序执行，不支持调用aclrtSubscribeHostFunc接口给同一个Stream注册多个处理回调函数的线程。

函数原型

aclError aclrtSubscribeHostFunc(uint64_t hostFuncThreadId, aclrtStream exeStream)

参数说明

参数名	输入/输出	说明
hostFuncThreadId	输入	指定线程的ID。
exeStream	输入	指定Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

11.10.6 aclrtProcessHostFunc

函数功能

等待指定时间后，触发回调处理，由aclrtSubscribeHostFunc接口指定的线程处理回调。

线程需由用户提前自行创建，并自定义线程函数，在线程函数内调用本接口，等待指定时间后通过系统内部进行算子计算。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

aclError aclrtProcessHostFunc(int32_t timeout)

参数说明

参数名	输入/输出	说明
timeout	输入	超时时间，单位为ms。 取值范围： ● -1：表示无限等待 ● 大于0（不包含0）：表示等待的时间

返回值说明

返回0表示成功，返回其它值表示失败。

11.10.7 aclrtUnSubscribeHostFunc

函数功能

与aclrtSubscribeHostFunc接口配合使用，调用模型执行接口后，调用本接口取消线程注册，Stream上的回调函数不再由指定线程处理。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

aclError aclrtUnSubscribeHostFunc(uint64_t hostFuncThreadId, aclrtStream exeStream)

参数说明

参数名	输入/输出	说明
hostFuncThreadId	输入	指定线程的ID。
exeStream	输入	指定Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

11.11 内存管理

11.11.1 总体说明

- 对于媒体数据处理功能，关于内存使用，有以下注意事项：
  a. 若涉及媒体数据处理功能，由于媒体数据处理功能对存放输入、输出数据的内存有更高的要求（例如，内存首地址128字节对齐），因此需调用专用的内存申请接口，如下：
     - 调用**媒体数据处理V1**版本的接口对图片进行抠图、缩放等操作时，调用**acldvppMalloc**接口申请内存。
     - 调用**媒体数据处理V2**版本的接口对图片进行抠图、缩放等操作时，调用**hi_mpi_dvpp_malloc**接口申请内存。
  b. 调用**a**申请出来的内存可以满足媒体数据处理的要求，也可以在其它任务中使用，例如，从性能角度，为了减少拷贝，媒体数据处理的输出作为模型推理的输入，实现内存复用。
  c. 但由于媒体数据处理访问的地址空间有限，为确保媒体数据处理时内存足够，除媒体数据处理功能外的其它功能（例如，模型加载），建议调用**内存管理**章节下的**aclrtMalloc**接口、或**aclrtMallocHost**接口、或**aclrtMallocCached**接口申请内存。

     Atlas 推理系列产品上，媒体数据处理功能在每个进程内，可访问的地址空间最大为16GB。
- 对于Atlas 200/300/500 推理产品，若涉及申请Device上的大页内存，需注意，当前版本为防止大页内存不够用，系统内部已提前预留（Device操作系统上的可用内存*25%≈1940M）大页内存，用户在申请大页内存时优先使用这部分预留内存。Device操作系统上的可用内存指使用free命令查询得到的total内存。
  – 在**Control CPU开放形态**下，使用完预留的大页内存后，除**acldvppMalloc**和**hi_mpi_dvpp_malloc**接口外（这两个接口内部会根据Device上的小页内存进行拼接），其它接口申请大页会返回失败。

11.11.2 aclrtMalloc

函数功能

在Device上分配size大小的线性内存，并通过*devPtr返回已分配内存的指针。本接口分配的内存会进行字节对齐，会对用户申请的size向上对齐成32字节整数倍后再多加32字节。

约束说明

- 若涉及媒体数据处理功能，由于媒体数据处理功能对存放输入、输出数据的内存有更高的要求（例如，内存首地址128字节对齐），因此需调用专用的内存申请接口，如下：
  - 调用**媒体数据处理V1**版本的接口对图片进行抠图、缩放等操作时，调用**acldvppMalloc**接口申请内存。
  - 调用**媒体数据处理V2**版本的接口对图片进行抠图、缩放等操作时，调用**hi_mpi_dvpp_malloc**接口申请内存。
- 本接口分配的内存不会进行对内容进行初始化。
- 本接口内部不会进行隐式的Device同步或流同步。如果申请内存成功或申请内存失败会立刻返回结果。
- 使用aclrtMalloc接口申请的内存，需要通过**aclrtFree**接口释放内存。
- 频繁调用aclrtMalloc接口申请内存、调用**aclrtFree**接口释放内存，会损耗性能，建议用户提前做内存预先分配或二次管理，避免频繁申请/释放内存。
- 若用户需申请大块内存并自行划分、管理内存时，建议使用aclrtMallocAlign32接口，该接口相比aclrtMalloc接口，只会对用户申请的size向上对齐成32字节整数倍，不会再多加32字节。

  不管是aclrtMalloc接口，还是aclrtMallocAlign32接口，若用户使用本接口申请大块内存并自行划分、管理内存时，每段内存需同时满足以下需求：
  - 内存大小向上对齐成32整数倍+32字节（m=ALIGN_UP[len,32]+32字节）；
  - 内存起始地址需满足64字节对齐（ALIGN_UP[m,64]）。

  说明

  len表示某段内存的大小，ALIGN_UP[len,k]表示向上按k字节对齐：((len-1)/k+1)*k。

函数原型

aclError aclrtMalloc(void **devPtr, size_t size, aclrtMemMallocPolicy policy)

参数说明

参数名	输入/输出	说明
devPtr	输出	“Device上已分配内存的指针”的指针。
size	输入	申请内存的大小，单位Byte。 size不能为0。
policy	输入	内存分配规则。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见**4.5 数据传输**。

11.11.3 aclrtMallocAlign32

函数功能

在Device上分配size大小的线性内存，并通过*devPtr返回已分配内存的指针。本接口分配的内存会进行字节对齐，会对用户申请的size向上对齐成32字节整数倍。

与aclrtMalloc接口相比，本接口只会对用户申请的size向上对齐成32字节整数倍，不会再多加32字节。

约束说明

- 若涉及媒体数据处理功能，由于媒体数据处理功能对存放输入、输出数据的内存有更高的要求（例如，内存首地址128字节对齐），因此需调用专用的内存申请接口，如下：
  - 调用**媒体数据处理V1**版本的接口对图片进行抠图、缩放等操作时，调用**acldvppMalloc**接口申请内存。
  - 调用**媒体数据处理V2**版本的接口对图片进行抠图、缩放等操作时，调用**hi_mpi_dvpp_malloc**接口申请内存。
- 本接口分配的内存不会进行对内容进行初始化。
- 本接口内部不会进行隐式的Device同步或流同步。如果申请内存成功或申请内存失败会立刻返回结果。
- 使用aclrtMallocAlign32接口申请的内存，需要通过**aclrtFree**接口释放内存。
- 频繁调用aclrtMallocAlign32接口申请内存、调用**aclrtFree**接口释放内存，会损耗性能，建议用户提前做内存预先分配或二次管理，避免频繁申请/释放内存。
- 若用户使用本接口申请大块内存并自行划分、管理内存时，每段内存需同时满足以下需求：
  - 内存大小向上对齐成32整数倍+32字节（m=ALIGN_UP[len,32]+32字节）；
  - 内存起始地址需满足64字节对齐（ALIGN_UP[m,64]）。

  说明

  len表示某段内存的大小，ALIGN_UP[len,k]表示向上按k字节对齐：((len-1)/k+1)*k。

函数原型

aclError aclrtMallocAlign32(void **devPtr, size_t size, aclrtMemMallocPolicy policy)

参数说明

参数名	输入/输出	说明
devPtr	输出	“Device上已分配内存的指针”的指针。
size	输入	申请内存的大小，单位Byte。 size不能为0。
policy	输入	内存分配规则。

返回值说明

返回0表示成功，返回其它值表示失败。

11.11.4 aclrtMallocCached

函数功能

在Device上申请size大小的线性内存，通过*devPtr返回已分配内存的指针，该接口在任何场景下申请的内存都是支持cache缓存。

使用aclrtMallocCached接口申请的内存与使用**aclrtMalloc**接口申请的内存是等价的，都支持cache缓存，不需要用户处理cpu与npu之间的cache一致性。

调用**媒体数据处理**的接口前，若需要申请Device上的内存存放输入或输出数据，需调用**acldvppMalloc**申请内存。

约束说明

其它约束与**aclrtMalloc**接口相同。

函数原型

aclError aclrtMallocCached(void **devPtr, size_t size, aclrtMemMallocPolicy policy)

参数说明

参数名	输入/输出	说明
devPtr	输出	“Device上已分配内存的指针”的指针。
size	输入	申请内存的大小，单位Byte。 size不能为0。
policy	输入	内存分配规则。

返回值说明

返回0表示成功，返回其它值表示失败。

11.11.5 aclrtMemFlush

函数功能

将cache中的数据刷新到ddr中，并将cache中的内容设置成无效。

该版本不需要用户处理cpu与npu之间的cache一致性，无需调用该接口。

函数原型

aclError aclrtMemFlush(void *devPtr, size_t size)

参数说明

参数名	输入/输出	说明
devPtr	输入	要flush的ddr内存起始地址指针。
size	输入	要flush的ddr内存大小，单位Byte。 size不能为0。

返回值说明

返回0表示成功，返回其它值表示失败。

11.11.6 aclrtMemInvalidate

函数功能

将cache中的数据设置成无效。

该版本不需要用户处理cpu与npu之间的cache一致性，无需调用该接口。

函数原型

aclError aclrtMemInvalidate(void *devPtr, size_t size)

参数说明

参数名	输入/输出	说明
devPtr	输入	需要将其中cache数据置为无效的ddr内存起始地址指针。
size	输入	ddr内存大小，单位Byte。 size不能为0。

返回值说明

返回0表示成功，返回其它值表示失败。

11.11.7 aclrtFree

函数功能

释放Device上的内存。

约束说明

- **aclrtFree**接口只能释放通过**aclrtMalloc**接口或**aclrtMallocCached**接口或**aclrtMallocAlign32**接口申请的内存。
- 本接口会立刻释放传入的内存，函数内部不会进行隐式的Device同步或流同步。用户需要确保调用完本接口后不再对该内存指针进行访问。

函数原型

aclError aclrtFree(void *devPtr)

参数说明

参数名	输入/输出	说明
devPtr	输入	待释放内存的指针。 如果传入的devPtr为空指针，本接口会返回报错。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见**4.5 数据传输**。

11.11.8 aclrtMallocHost

函数功能

AscendCL应用程序在Host上运行时，调用该接口申请的是Host内存（该内存是锁页内存），由系统保证内存首地址64字节对齐。AscendCL应用程序在Device上运行时，调用该接口申请的是Device内存，且Device上的内存按普通页申请，如需首地址64字节对齐，需要用户自行处理对齐。

约束说明

- 本接口内部不会进行隐式的device同步或流同步。如果申请内存成功或申请内存失败会立刻返回结果。

- 使用aclrtMallocHost接口申请的内存，需要通过**aclrtFreeHost**接口释放内存。
- 使用aclrtMallocHost接口分配过多的锁页内存，将导致操作系统用于分页的物理内存变少，从而降低系统整体的性能。
- 频繁调用aclrtMallocHost接口申请内存、调用**aclrtFreeHost**接口释放内存，会损耗性能，建议用户提前做内存预先分配或二次管理，避免频繁申请/释放内存。
- 若用户使用本接口申请大块内存并自行划分、管理内存时，每段内存需同时满足以下需求：
  - 内存大小向上对齐成32整数倍+32字节（m=ALIGN_UP[len,32]+32字节）；
  - 内存起始地址需满足64字节对齐（ALIGN_UP[m,64]）。

说明

len表示某段内存的大小，ALIGN_UP[len,k]表示向上按k字节对齐：((len-1)/k+1)*k。

函数原型

aclError aclrtMallocHost(void **hostPtr, size_t size)

参数说明

参数名	输入/输出	说明
size	输入	申请内存的大小，单位Byte。 size不能为0。
hostPtr	输出	“已分配内存的指针”的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见**4.5 数据传输**。

11.11.9 aclrtFreeHost

函数功能

释放内存。

约束说明

- 本接口会立刻释放传入的内存，函数内部不会进行隐式的Device同步或流同步。用户需要确保调用完本接口后不再对该内存指针进行访问。
- **aclrtFreeHost**接口只能释放通过**aclrtMallocHost**接口申请的内存。

函数原型

aclError aclrtFreeHost(void *hostPtr)

参数说明

参数名	输入/输出	说明
hostPtr	输入	待释放内存的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见4.5 数据传输。

11.11.10 aclrtMemset

函数功能

初始化内存，将内存中的内容设置为指定的值。

要初始化的内存支持在Host侧或Device侧，系统根据地址判定是Host还是Device。

约束说明

本接口会立刻进行内存初始化，函数内部不会进行隐式的device同步或流同步。

函数原型

aclError aclrtMemset (void *devPtr, size_t maxCount, int32_t value, size_t count)

参数说明

参数名	输入/输出	说明
devPtr	输入	内存起始地址的指针。
maxCount	输入	内存的最大长度，单位Byte。
value	输入	设置的值。
count	输入	需要设置为指定值的内存长度，单位Byte。

返回值说明

返回0表示成功，返回其它值表示失败。

11.11.11 aclrtMemsetAsync

函数功能

初始化内存，将内存中的内容设置为指定的值，异步接口。

要初始化的内存支持在Host侧或Device侧，系统根据地址判定是Host还是Device。

约束说明

该接口是异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功。调用该接口后，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保内存初始化的任务已执行完成。

函数原型

aclError aclrtMemsetAsync(void *devPtr, size_t maxCount, int32_t value, size_t count, aclrtStream stream)

参数说明

参数名	输入/输出	说明
devPtr	输入	内存起始地址的指针。
maxCount	输入	内存的最大长度，单位Byte。
value	输入	设置的值。
count	输入	需要设置为指定值的内存长度，单位Byte。
stream	输入	指定stream。

返回值说明

返回0表示成功，返回其它值表示失败。

11.11.12 aclrtMemcpy

函数功能

实现内存复制。

约束说明

- 本接口会立刻进行内存复制，函数内部不会进行隐式的device同步或流同步。
- Atlas 推理系列产品，如果执行Device间的内存复制，需先调用**aclrtDeviceCanAccessPeer**接口查询两个Device间是否支持内存复制、调用**aclrtDeviceEnablePeerAccess**接口使用两个Device间的内存复制，再调用本接口进行内存复制。

函数原型

aclError aclrtMemcpy(void *dst, size_t destMax, const void *src, size_t count, aclrtMemcpyKind kind)

参数说明

参数名	输入/输出	说明
dst	输入	目的内存地址指针。
destMax	输入	目的内存地址的最大内存长度，单位Byte。
src	输入	源内存地址指针。
count	输入	内存复制的长度，单位Byte。
kind	输入	内存复制的类型，预留参数，配置枚举值中的值无效，系统内部会根据源内存地址指针、目的内存地址指针判断是否可以将源地址的数据复制到目的地址，如果不可以，则系统会返回报错。 typedef enum aclrtMemcpyKind { ACL_MEMCPY_HOST_TO_HOST, // Host内的内存复制 ACL_MEMCPY_HOST_TO_DEVICE, // Host到Device的内存复制 ACL_MEMCPY_DEVICE_TO_HOST, // Device到Host的内存复制 ACL_MEMCPY_DEVICE_TO_DEVICE, // Device内或Device间的内存复制 } aclrtMemcpyKind;

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见4.5 数据传输。

11.11.13 aclrtMemcpyAsync

函数功能

实现内存复制，异步接口。

约束说明

- 调用本接口进行内存复制时，源地址和目的地址都必须64字节对齐。
- 本接口是异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功。调用该接口后，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保内存复制的任务已执行完成。

- 若调用本接口传入的kind为ACL_MEMCPY_HOST_TO_DEVICE或ACL_MEMCPY_DEVICE_TO_HOST，并且Host内存不是通过aclrtMallocHost申请得到的，本接口不会报错，但是在实际执行到该任务时会发生不可预知的错误。
- 本接口不支持异步Host内的内存复制功能，因此传入的kind为ACL_MEMCPY_HOST_TO_HOST时，任务下发成功，但系统内部处理该任务时会返回失败。
- 在Atlas 200/300/500 推理产品上：

  Control CPU开放形态下，在Device上运行应用时，选择涉及Host的内存复制类型后，系统内部会默认使用Device内的内存复制。
- 在Atlas 推理系列产品上：

  如果执行Device间的内存复制，需先调用**aclrtDeviceCanAccessPeer**接口查询两个Device间是否支持内存复制、调用**aclrtDeviceEnablePeerAccess**接口使用两个Device间的内存复制，再调用本接口进行内存复制。

  Control CPU开放形态下，在Device上运行应用时，选择涉及Host的内存复制类型后，系统内部会默认使用Device内的内存复制。

函数原型

aclError aclrtMemcpyAsync(void *dst, size_t destMax, const void *src, size_t count, aclrtMemcpyKind kind, aclrtStream stream)

参数说明

参数名	输入/输出	说明
dst	输入	目的内存地址指针。
destMax	输入	目的内存地址的最大内存长度，单位Byte。
src	输入	源内存地址指针。
count	输入	内存复制的长度，单位Byte。
kind	输入	内存复制的类型。 typedef enum aclrtMemcpyKind { ACL_MEMCPY_HOST_TO_HOST, // Host内的内存复制 ACL_MEMCPY_HOST_TO_DEVICE, // Host到Device的内存复制 ACL_MEMCPY_DEVICE_TO_HOST, // Device到Host的内存复制 ACL_MEMCPY_DEVICE_TO_DEVICE, // Device内或Device间的内存复制 } aclrtMemcpyKind;
stream	输入	指定stream。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见4.5 数据传输。

11.11.14 aclrtGetMemInfo

函数功能

根据指定属性，获取Device上AscendCL应用可用内存的空闲大小和总大小，不包括系统预留内存大小。

约束说明

- 调用本接口前是必须先指定用于计算的Device（例如调用aclrtSetDevice接口指定用于计算的Device），因此本接口中不体现Device ID。
- 请根据实际硬件支持的情况，选择对应属性的内存，否则调用本接口获取到的空闲大小和总大小都为0。

  该功能适用以下项：

  Atlas 200/300/500 推理产品
- 由于Device上没有HBM相关的内存，若指定查询HBM相关内存，则AscendCL系统内部自动转换为查询DDR相关内存，例如，若查询ACL_HBM_MEM，则AscendCL系统内部实际查询ACL_DDR_MEM；若查询ACL_HBM_MEM_HUGE，则AscendCL系统内部实际查询ACL_DDR_MEM_HUGE。

  该功能适用于以下项：

  Atlas 推理系列产品

函数原型

aclError aclrtGetMemInfo(aclrtMemAttr attr, size_t *free, size_t *total)

参数说明

参数名	输入/输出	说明
attr	输入	需要查询的内存的属性值。
free	输出	对应属性内存空闲大小的指针，单位Byte。
total	输出	对应属性内存总大小的指针，单位Byte。

返回值说明

返回0表示成功，返回其它值表示失败。

11.11.15 aclrtDeviceCanAccessPeer

函数功能

查询Device之间是否支持内存复制。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 仅支持物理机和容器场景；
- 仅支持同一个PCIe Switch内Device之间的内存复制。AI Server场景下，虽然是跨PCIe Switch，但也支持Device之间的内存复制。
- 仅支持同一个物理机或容器内的Device之间的内存复制操作；
- 仅支持同一个进程内、线程间的Device之间的内存复制，不支持不同进程间Device之间的内存复制。
- Atlas 推理系列产品，**Control CPU开放形态**下，应用程序运行在Device的Control CPU上时，该接口不支持Device之间的内存复制。

函数原型

aclError aclrtDeviceCanAccessPeer(int32_t *canAccessPeer, int32_t deviceId, int32_t peerDeviceId)

参数说明

参数名	输入/输出	说明
canAccessPeer	输出	通过deviceId参数指定的Device和通过peerDeviceId参数指定的Device之间是否支持内存复制，1表示支持，0表示不支持。
deviceId	输入	指定Device的ID，不能与peerDeviceId参数值相同。 用户调用**aclrtGetDeviceCount**接口获取可用的Device数量后，这个Device ID的取值范围：[0, (可用的Device数量-1)]
peerDeviceId	输入	指定Device的ID，不能与deviceId参数值相同。 用户调用**aclrtGetDeviceCount**接口获取可用的Device数量后，这个Device ID的取值范围：[0, (可用的Device数量-1)]

返回值说明

返回0表示成功，返回其它值表示失败。

11.11.16 aclrtDeviceEnablePeerAccess

函数功能

使能当前Device与指定Device之间的内存复制。使能内存复制是Device级的。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

可提前调用**aclrtDeviceCanAccessPeer**接口显查询当前Device与指定Device之间能否进行内存复制。使能Device间的内存复制功能后，若想关闭该功能，可调用**aclrtDeviceDisablePeerAccess**接口。

Atlas 推理系列产品，**Control CPU开放形态**下，应用程序运行在Device的Control CPU上时，该接口不支持Device之间的内存复制。

函数原型

aclError aclrtDeviceEnablePeerAccess(int32_t peerDeviceId, uint32_t flags)

参数说明

参数名	输入/输出	说明
peerDeviceId	输入	Device ID，该ID不能与当前Device的ID相同。 用户调用**aclrtGetDeviceCount**接口获取可用的Device数量后，这个Device ID的取值范围：[0, (可用的Device数量-1)]
flags	输入	保留参数，当前必须设置为0。

返回值说明

返回0表示成功，返回其它值表示失败。

11.11.17 aclrtDeviceDisablePeerAccess

函数功能

关闭当前Device与指定Device之间的内存复制功能。关闭内存复制功能是Device级的。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

调用**aclrtDeviceEnablePeerAccess**接口使能当前Device与指定Device之间的内存复制后，可调用aclrtDeviceDisablePeerAccess接口关闭内存复制功能。

Atlas 推理系列产品，**Control CPU开放形态**下，应用程序运行在Device的Control CPU上时，该接口不支持Device之间的内存复制。

函数原型

aclError aclrtDeviceDisablePeerAccess(int32_t peerDeviceId)

参数说明

参数名	输入/输出	说明
peerDeviceId	输入	Device ID，该ID不能与当前Device的ID相同。 用户调用**aclrtGetDeviceCount**接口获取可用的Device数量后，这个Device ID的取值范围：[0, (可用的Device数量-1)]

返回值说明

返回0表示成功，返回其它值表示失败。

11.11.18 aclrtMemcpy2d

函数功能

实现同步内存复制，主要用于矩阵数据的复制。

Atlas 推理系列加速模块产品不支持该接口。

函数原型

aclError aclrtMemcpy2d(void *dst, size_t dpitch, const void *src, size_t spitch, size_t width, size_t height, aclrtMemcpyKind kind)

约束说明

- 当前仅支持ACL_MEMCPY_HOST_TO_DEVICE类型和ACL_MEMCPY_DEVICE_TO_HOST类型的内存复制。
- Atlas 推理系列产品，**Control CPU开放形态**下，不支持调用本接口。
- Atlas 200/300/500 推理产品，**Control CPU开放形态**下，不支持调用本接口。

参数说明

参数名	输入/输出	说明
dst	输入	目的内存地址指针。
dpitch	输入	目的内存中相邻两列向量的地址距离。

参数名	输入/输出	说明
src	输入	源内存地址指针。
spitch	输入	源内存中相邻两列向量的地址距离。
width	输入	待复制的矩阵宽度。
height	输入	待复制的矩阵高度。
kind	输入	内存复制的类型。 typedef enum aclrtMemcpyKind { ACL_MEMCPY_HOST_TO_HOST, // Host内的内存复制 ACL_MEMCPY_HOST_TO_DEVICE, // Host到Device的内存复制 ACL_MEMCPY_DEVICE_TO_HOST, // Device到Host的内存复制 ACL_MEMCPY_DEVICE_TO_DEVICE, // Device内或Device间的内存复制 } aclrtMemcpyKind;

返回值说明

返回0表示成功，返回其它值表示失败。

11.11.19 aclrtMemcpy2dAsync

函数功能

实现异步内存复制，主要用于矩阵数据的复制。异步接口。

Atlas 推理系列加速模块产品不支持该接口。

约束说明

- 该接口是异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功。调用该接口后，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保内存复制的任务已执行完成。
- 当前仅支持ACL_MEMCPY_HOST_TO_DEVICE类型和ACL_MEMCPY_DEVICE_TO_HOST类型的内存复制。
- Atlas 推理系列产品，**Control CPU开放形态**下，不支持调用本接口。
- Atlas 200/300/500 推理产品，**Control CPU开放形态**下，不支持调用本接口。

函数原型

aclError aclrtMemcpy2dAsync(void *dst, size_t dpitch, const void *src, size_t spitch, size_t width, size_t height, aclrtMemcpyKind kind, aclrtStream stream)

参数说明

参数名	输入/输出	说明
dst	输入	目的内存地址指针。
dpitch	输入	目的内存中相邻两列向量的地址距离。
src	输入	源内存地址指针。
spitch	输入	源内存中相邻两列向量的地址距离。
width	输入	待复制的矩阵宽度。
height	输入	待复制的矩阵高度。 height最大设置为5*1024*1024=5242880，否则接口返回失败。
kind	输入	内存复制的类型。 typedef enum aclrtMemcpyKind { ACL_MEMCPY_HOST_TO_HOST, // Host内的内存复制 ACL_MEMCPY_HOST_TO_DEVICE, // Host到Device的内存复制 ACL_MEMCPY_DEVICE_TO_HOST, // Device到Host的内存复制 ACL_MEMCPY_DEVICE_TO_DEVICE, // Device内或Device间的内存复制 } aclrtMemcpyKind;
stream	输入	指定stream。

返回值说明

返回0表示成功，返回其它值表示失败。

11.11.20 aclrtAllocatorRegister

函数功能

调用该接口注册用户提供的Allocator以及Allocator对应的回调函数，用于使用用户提供的Allocator。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 当前仅支持在执行单算子场景下使用本接口，且需在调用单算子执行接口（例如：**aclopExecuteV2**、**aclopCompileAndExecuteV2**等）之前调用本接口。
- 调用本接口前，需要先调用**aclrtAllocatorCreateDesc**创建Allocator描述符，再分别调用**aclrtAllocatorSetObjToDesc**、**aclrtAllocatorSetAllocFuncToDesc**、**aclrtAllocatorSetGetAddrFromBlockFuncToDesc**、**aclrtAllocatorSetFreeFuncToDesc**设置Allocator对象及回调函数。调用本接口完成注册后，可调用**aclrtAllocatorDestroyDesc**接口销毁Allocator描述符。

- 对于同一条流，多次调用本接口，以最后一次注册为准。
- 对于不同流，如果用户使用同一个Allocator，不可以多条流并发执行，在执行下一条Stream前，需要对上一Stream做流同步。
- 将Allocator中的内存释放给操作系统前，需要先调用**aclrtSynchronizeStream**接口执行流同步，确保Stream中的任务已执行完成。

函数原型

aclError aclrtAllocatorRegister(aclrtStream stream, aclrtAllocatorDesc allocatorDesc)

参数说明

参数名	输入/输出	说明
stream	输入	该Allocator需要注册的Stream。
allocatorDesc	输入	Allocator描述符指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.11.21 aclrtAllocatorUnregister

函数功能

取消注册用户提供的Allocator以及Allocator对应的回调函数，用于取消使用用户提供的Allocator。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 用户销毁Allocator前，调用本接口取消注册。
- 待取消注册的Stream不存在，或多次调用本接口取消注册，本接口内部不做任何操作，返回成功。

函数原型

aclError aclrtAllocatorUnregister(aclrtStream stream)

参数说明

参数名	输入/输出	说明
stream	输入	该Allocator对应的stream。

返回值说明

返回0表示成功，返回其它值表示失败。

11.11.22 aclrtReserveMemAddress

函数功能

预留虚拟内存。

本接口需与以下其它接口配合使用，以便申请地址连续的虚拟内存、最大化利用物理内存地址的目的：

1. 申请虚拟内存（**aclrtReserveMemAddress**接口）；
2. 申请物理内存（**aclrtMallocPhysical**接口）；
3. 将虚拟内存映射到物理内存（**aclrtMapMem**接口）；
4. 执行任务（调用具体的任务接口）；
5. 取消虚拟内存与物理内存的映射（**aclrtUnmapMem**接口）；
6. 释放物理内存（**aclrtFreePhysical**接口）；
7. 释放虚拟内存（**aclrtReleaseMemAddress**接口）。

约束说明

Ascend RC形态不支持调用本接口。

函数原型

aclError aclrtReserveMemAddress(void **virPtr, size_t size, size_t alignment, void *expectPtr, uint64_t flags)

参数说明

参数名	输入/输出	说明
virPtr	输出	“已分配的虚拟内存地址的指针”的指针。
size	输入	虚拟内存大小，单位Byte。 size不能为0。
alignment	输入	虚拟内存地址对齐值，预留，当前只能设置为0。
expectPtr	输入	指定期望返回的虚拟内存起始地址.，预留，当前只能传nullptr。

参数名	输入/输出	说明
flags	输入	大页/普通页标志，，此处的标志需与aclrtMallocPhysical接口的内存类型保持一致。 参数取值如下： ● 0：普通页 ● 1：大页

返回值说明

返回0表示成功，返回其它值表示失败。

11.11.23 aclrtReleaseMemAddress

函数功能

释放通过aclrtReserveMemAddress接口申请的虚拟内存。

本接口需与以下其它接口配合使用，以便申请地址连续的虚拟内存、最大化利用物理内存地址的目的：

1. 申请虚拟内存（aclrtReserveMemAddress接口）；
2. 申请物理内存（aclrtMallocPhysical接口）；
3. 将虚拟内存映射到物理内存（aclrtMapMem接口）；
4. 执行任务（调用具体的任务接口）；
5. 取消虚拟内存与物理内存的映射（aclrtUnmapMem接口）；
6. 释放物理内存（aclrtFreePhysical接口）；
7. 释放虚拟内存（aclrtReleaseMemAddress接口）。

约束说明

- Ascend RC形态不支持调用本接口。
- 若该虚拟内存与物理内存存在映射关系，则释放虚拟内存前，需调用aclrtUnmapMem接口取消该虚拟内存与物理内存的映射。

函数原型

aclError aclrtReleaseMemAddress(void *virPtr)

参数说明

参数名	输入/输出	说明
virPtr	输入	待释放的虚拟内存地址指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.11.24 aclrtMallocPhysical

函数功能

申请物理内存，并返回一个物理内存handle。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

本接口需与以下其它接口配合使用，以便申请地址连续的虚拟内存、最大化利用物理内存地址的目的：

1. 申请虚拟内存（**aclrtReserveMemAddress**接口）；
2. 申请物理内存（**aclrtMallocPhysical**接口）；
3. 将虚拟内存映射到物理内存（**aclrtMapMem**接口）；
4. 执行任务（调用具体的任务接口）；
5. 取消虚拟内存与物理内存的映射（**aclrtUnmapMem**接口）；
6. 释放物理内存（**aclrtFreePhysical**接口）；
7. 释放虚拟内存（**aclrtReleaseMemAddress**接口）。

约束说明

- Ascend RC形态不支持调用本接口。
- 当前版本仅支持ACL_HBM_MEM_HUGE（表示HBM大页内存）、ACL_HBM_MEM_NORMAL（表示HBM普通页内存）两种内存类型，若传入ACL_HBM_MEM_NORMAL类型，系统内部当前也按HBM大页内存申请。

  在Atlas 推理系列产品上，由于Device上没有HBM内存，使用本接口申请ACL_HBM_MEM_HUGE类型的内存时，AscendCL系统内部自动转换为申请ACL_DDR_MEM_HUGE类型的DDR内存。

函数原型

aclError aclrtMallocPhysical(aclrtDrvMemHandle *handle, size_t size, const aclrtPhysicalMemProp *prop, uint64_t flags)

参数说明

参数名	输入/输出	说明
handle	输出	存放物理内存信息的handle。
size	输入	物理内存大小，单位Byte。 size当前需按照2M对齐。
prop	输入	物理内存属性信息。
flags	输入	预留，当前只能设置为0。

返回值说明

返回0表示成功，返回其它值表示失败。

11.11.25 aclrtFreePhysical

函数功能

释放通过**aclrtMallocPhysical**接口申请的物理内存。

本接口需与以下其它接口配合使用，以便申请地址连续的虚拟内存、最大化利用物理内存地址的目的：

1. 申请虚拟内存（**aclrtReserveMemAddress**接口）；
2. 申请物理内存（**aclrtMallocPhysical**接口）；
3. 将虚拟内存映射到物理内存（**aclrtMapMem**接口）；
4. 执行任务（调用具体的任务接口）；
5. 取消虚拟内存与物理内存的映射（**aclrtUnmapMem**接口）；
6. 释放物理内存（**aclrtFreePhysical**接口）；
7. 释放虚拟内存（**aclrtReleaseMemAddress**接口）。

约束说明

- Ascend RC形态不支持调用本接口。
- 若该物理内存与虚拟内存存在映射关系，则释放物理内存前，需调用**aclrtUnmapMem**接口取消该物理内存与虚拟内存的映射。

函数原型

aclError aclrtFreePhysical(aclrtDrvMemHandle handle)

参数说明

参数名	输入/输出	说明
handle	输入	待释放的物理内存信息handle。

返回值说明

返回0表示成功，返回其它值表示失败。

11.11.26 aclrtMapMem

函数功能

将虚拟内存映射到物理内存。

本接口需与以下其它接口配合使用，以便申请地址连续的虚拟内存、最大化利用物理内存地址的目的：

1. 申请虚拟内存（**aclrtReserveMemAddress**接口）；
2. 申请物理内存（**aclrtMallocPhysical**接口）；
3. 将虚拟内存映射到物理内存（**aclrtMapMem**接口）；
4. 执行任务（调用具体的任务接口）；
5. 取消虚拟内存与物理内存的映射（**aclrtUnmapMem**接口）；
6. 释放物理内存（**aclrtFreePhysical**接口）；
7. 释放虚拟内存（**aclrtReleaseMemAddress**接口）。

约束说明

Ascend RC形态不支持调用本接口。

函数原型

aclError aclrtMapMem(void *virPtr, size_t size, size_t offset, aclrtDrvMemHandle handle, uint64_t flags)

参数说明

参数名	输入/输出	说明
virPtr	输入	待映射的虚拟内存地址指针。 这个地址不一定是起始地址，用户也可以根据起始地址自行偏移后，再映射。 虚拟地址当前需按照2M对齐。
size	输入	待映射的内存大小，单位Byte。 size当前需按照2M对齐。
offset	输入	物理内存偏移值，当前只能设置为0。
handle	输入	物理内存信息handle。
flags	输入	预留，当前只能设置为0。

返回值说明

返回0表示成功，返回其它值表示失败。

11.11.27 aclrtUnmapMem

函数功能

取消虚拟内存与物理内存之间的映射关系。

本接口需与以下其它接口配合使用，以便申请地址连续的虚拟内存、最大化利用物理内存地址的目的：

1. 申请虚拟内存（**aclrtReserveMemAddress**接口）；
2. 申请物理内存（**aclrtMallocPhysical**接口）；
3. 将虚拟内存映射到物理内存（**aclrtMapMem**接口）；
4. 执行任务（调用具体的任务接口）；
5. 取消虚拟内存与物理内存的映射（**aclrtUnmapMem**接口）；
6. 释放物理内存（**aclrtFreePhysical**接口）；
7. 释放虚拟内存（**aclrtReleaseMemAddress**接口）。

约束说明

Ascend RC形态不支持调用本接口。

函数原型

aclError aclrtUnmapMem(void *virPtr)

参数说明

参数名	输入/输出	说明
virPtr	输入	待取消映射的虚拟内存地址指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.12 模型加载与执行

11.12.1 aclmdlLoadFromFile

函数功能

从文件加载离线模型数据（适配昇腾AI处理器的离线模型），由系统内部管理内存。

系统完成模型加载后，返回的模型ID，作为后续操作时用于识别模型的标志。

约束说明

模型加载、模型执行、模型卸载的操作必须在同一个Context下（关于Context的创建请参见**11.6.1 aclrtCreateContext**）。在加载前，请先根据模型文件的大小评估内存空间是否足够，内存空间不足，会导致应用程序异常。

函数原型

aclError aclmdlLoadFromFile(const char *modelPath, uint32_t *modelId)

参数说明

参数名	输入/输出	说明
modelPath	输入	离线模型文件路径的指针，路径中包含文件名。运行程序（APP）的用户需要对该存储路径有访问权限。 此处的离线模型文件是适配昇腾AI处理器的离线模型，即*.om文件。 **说明** ● 关于如何获取om文件，请参见《ATC工具使用指南（开放态）》。 ● 对om模型文件大小有限制的场景下，如果使用ATC工具生成om文件时，将--external_weight参数设置为1（1表示将原始网络中的Const/Constant节点的权重保存在单独的文件中，且该文件保存在与om文件同级的weight目录下），那么在使用本接口加载om文件时，需将weight目录与om文件放在同级目录下，这时AscendCL会自行到weight目录下查找权重文件，否则可能会导致单独的权重文件加载不成功。
modelId	输出	模型ID的指针。 系统成功加载模型后会返回的模型ID。

返回值说明

返回0表示成功，返回其它值表示失败。

相关接口

当前AscendCL还提供了**aclmdlSetConfigOpt**接口、**aclmdlLoadWithConfig**接口来实现模型加载，通过配置对象中的属性来区分，在加载模型时是从文件加载，还是从内存加载，以及内存是由系统内部管理，还是由用户管理。

参考资源

接口调用流程，参见**4.6.1 模型加载**。

11.12.2 aclmdlLoadFromMem

函数功能

从内存加载离线模型数据，由系统内部管理模型运行的内存。

系统完成模型加载后，返回的模型ID，作为后续操作时用于识别模型的标志。

约束说明

模型加载、模型执行、模型卸载的操作必须在同一个Context下（关于Context的创建请参见**11.6.1 aclrtCreateContext**）。在加载前，请先根据模型文件的大小评估内存空间是否足够，内存空间不足，会导致应用程序异常。

函数原型

aclError aclmdlLoadFromMem(const void* model, size_t modelSize, uint32_t* modelId)

参数说明

参数名	输入/输出	说明
model	输入	存放模型数据的内存地址指针。 应用运行在Host时，此处需申请Host上的内存；应用运行在Device时，此处需申请Device上的内存。内存申请接口请参见11.11 内存管理。
modelSize	输入	内存中的模型数据长度，单位Byte。
modelId	输出	模型ID的指针。 系统成功加载模型后会返回的模型ID。

返回值说明

返回0表示成功，返回其它值表示失败。

相关接口

当前AscendCL还提供了aclmdlSetConfigOpt接口、aclmdlLoadWithConfig接口来实现模型加载，通过配置对象中的属性来区分，在加载模型时是从文件加载，还是从内存加载，以及内存是由系统内部管理，还是由用户管理。

参考资源

接口调用流程，参见4.6.1 模型加载。

11.12.3 aclmdlLoadFromFileWithMem

函数功能

从文件加载离线模型数据（适配昇腾AI处理器的离线模型），由用户自行管理模型运行的内存。

系统完成模型加载后，返回的模型ID，作为后续操作时用于识别模型的标志。

约束说明

模型加载、模型执行、模型卸载的操作必须在同一个Context下（关于Context的创建请参见11.6.1 aclrtCreateContext）。

函数原型

aclError aclmdlLoadFromFileWithMem(const char *modelPath, uint32_t *modelId, void *workPtr, size_t workSize, void *weightPtr, size_t weightSize)

参数说明

参数名	输入/输出	说明
modelPath	输入	离线模型文件路径的指针，路径中包含文件名。运行程序（APP）的用户需要对该存储路径有访问权限。 此处的离线模型文件是适配昇腾AI处理器的离线模型，即*.om文件。 **说明** • 关于如何获取om文件，请参见《**ATC工具使用指南（开放态）**》。
modelId	输出	模型ID的指针。 系统成功加载模型后会返回的模型ID。
workPtr	输入	Device上模型所需工作内存（存放模型执行过程中的临时数据）的地址指针，由用户自行管理，模型执行过程中不能释放该内存。 如果在workPtr参数处传入空指针，表示由系统管理内存。 **说明** 由用户自行管理工作内存时，如果多个模型串行执行，支持共用同一个工作内存，但用户需确保模型的串行执行顺序、且工作内存的大小需按多个模型中最大工作内存的大小来申请，例如通过以下方式保证串行： • 同步模型执行时，加锁，保证执行任务串行。 • 异步模型执行时，使用同一个stream，保证执行任务串行。
workSize	输入	模型所需工作内存的大小，单位Byte。workPtr为空指针时无效。

参数名	输入/输出	说明
weightPtr	输入	Device上模型权值内存（存放权值数据）的地址指针，由用户自行管理，模型执行过程中不能释放该内存。 如果在weightPtr参数处传入空指针，表示由系统管理内存。 **说明** 由用户自行管理权值内存时，在多线程场景下，对于同一个模型，如果在每个线程中都加载了一次，支持共用weightPtr，因为weightPtr内存在推理过程中是只读的。此处需注意，在共用weightPtr期间，不能释放weightPtr。
weightSize	输入	模型所需权值内存的大小，单位Byte。weightPtr为空指针时无效。

返回值说明

返回0表示成功，返回其它值表示失败。

相关接口

当前AscendCL还提供了**aclmdlSetConfigOpt**接口、**aclmdlLoadWithConfig**接口来实现模型加载，通过配置对象中的属性来区分，在加载模型时是从文件加载，还是从内存加载，以及内存是由系统内部管理，还是由用户管理。

参考资源

接口调用流程及示例代码，参见**4.6.1 模型加载**。

11.12.4 aclmdlLoadFromMemWithMem

函数功能

从内存加载离线模型数据，由用户自行管理模型运行的内存。

系统完成模型加载后，返回的模型ID，作为后续操作时用于识别模型的标志。

约束说明

模型加载、模型执行、模型卸载的操作必须在同一个Context下（关于Context的创建请参见**11.6.1 aclrtCreateContext**）。在加载前，请先根据模型文件的大小评估内存空间是否足够，内存空间不足，会导致应用程序异常。

函数原型

aclError aclmdlLoadFromMemWithMem(const void *model, size_t modelSize, uint32_t *modelId, void *workPtr, size_t workSize, void *weightPtr, size_t weightSize)

参数说明

参数名	输入/输出	说明
model	输入	存放模型数据的内存地址指针。 应用运行在Host时，此处需申请Host上的内存；应用运行在Device时，此处需申请Device上的内存。内存申请接口请参见**11.11 内存管理**。
modelSize	输入	模型数据长度，单位Byte。
modelId	输出	模型ID的指针。 系统成功加载模型后会返回的模型ID。
workPtr	输入	Device上模型所需工作内存（存放模型执行过程中的临时数据）的地址指针，由用户自行管理，模型执行过程中不能释放该内存。 如果在workPtr参数处传入空指针，表示由系统管理内存。 **说明** 由用户自行管理工作内存时，如果多个模型串行执行，支持共用同一个工作内存，但用户需确保模型的串行执行顺序、且工作内存的大小需按多个模型中最大工作内存的大小来申请，例如通过以下方式保证串行： ● 同步模型执行时，加锁，保证执行任务串行。 ● 异步模型执行时，使用同一个stream，保证执行任务串行。
workSize	输入	模型所需工作内存的大小，单位Byte。workPtr为空指针时无效。
weightPtr	输入	Device上模型权值内存（存放权值数据）的地址指针，由用户自行管理，模型执行过程中不能释放该内存。 如果在weightPtr参数处传入空指针，表示由系统管理内存。 **说明** 由用户自行管理权值内存时，在多线程场景下，对于同一个模型，如果在每个线程中都加载了一次，支持共用weightPtr，因为weightPtr内存在推理过程中是只读的。此处需注意，在共用weightPtr期间，不能释放weightPtr。
weightSize	输入	模型所需权值内存的大小，单位Byte。weightPtr为空指针时无效。

返回值说明

返回0表示成功，返回其它值表示失败。

相关接口

当前AscendCL还提供了**aclmdlSetConfigOpt**接口、**aclmdlLoadWithConfig**接口来实现模型加载，通过配置对象中的属性来区分，在加载模型时是从文件加载，还是从内存加载，以及内存是由系统内部管理，还是由用户管理。

参考资源

接口调用流程，参见**4.6.1 模型加载**。

11.12.5 aclmdlLoadFromFileWithQ

函数功能

从文件加载离线模型数据（适配昇腾AI处理器的离线模型），模型的输入、输出数据都存放在队列中。

系统完成模型加载后，返回的模型ID，作为后续操作时用于识别模型的标志。

约束说明

- 本接口只支持加载固定Shape输入的模型。
- 模型加载、模型执行、模型卸载的操作必须在同一个Context下（关于Context的创建请参见**11.6.1 aclrtCreateContext**）。

函数原型

aclError aclmdlLoadFromFileWithQ(const char *modelPath, uint32_t *modelId, const uint32_t *inputQ, size_t inputQNum, const uint32_t *outputQ, size_t outputQNum)

参数说明

参数名	输入/输出	说明
modelPath	输入	离线模型文件路径的指针，路径中包含文件名。运行程序（APP）的用户需要对该存储路径有访问权限。 此处的离线模型文件是适配昇腾AI处理器的离线模型，即*.om文件。 **说明** • 关于如何获取om文件，请参见《**ATC工具使用指南（开放态）**》。 • 对om模型文件大小有限制的场景下，如果使用ATC工具生成om文件时，将--external_weight参数设置为1（1表示将原始网络中的Const/Constant节点的权重保存在单独的文件中，且该文件保存在与om文件同级的weight目录下），那么在使用本接口加载om文件时，需将weight目录与om文件放在同级目录下，这时AscendCL会自行到weight目录下查找权重文件，否则可能会导致单独的权重文件加载不成功。
modelId	输出	模型ID的指针。 系统成功加载模型后会返回的模型ID。
inputQ	输入	队列ID的指针，一个模型的输入对应一个队列ID。
inputQNum	输入	输入队列大小。
outputQ	输入	队列ID的指针，一个模型的输出对应一个队列ID。
outputQNum	输入	输出队列大小。

返回值说明

返回0表示成功，返回其它值表示失败。

相关接口

当前AscendCL还提供了**aclmdlSetConfigOpt**接口、**aclmdlLoadWithConfig**接口来实现模型加载，通过配置对象中的属性来区分，在加载模型时是从文件加载，还是从内存加载，以及内存是由系统内部管理，还是由用户管理。

11.12.6 aclmdlLoadFromMemWithQ

函数功能

从内存加载离线模型数据（适配昇腾AI处理器的离线模型），模型的输入、输出数据都存放在队列中。

系统完成模型加载后，返回的模型ID，作为后续操作时用于识别模型的标志。

约束说明

- 本接口只支持加载固定Shape输入的模型。
- 模型加载、模型执行、模型卸载的操作必须在同一个Context下（关于Context的创建请参见**11.6.1 aclrtCreateContext**）。

函数原型

aclError aclmdlLoadFromMemWithQ(const void *model, size_t modelSize, uint32_t *modelId, const uint32_t *inputQ, size_t inputQNum, const uint32_t *outputQ, size_t outputQNum)

参数说明

参数名	输入/输出	说明
model	输入	存放模型数据的内存地址指针。
modelSize	输入	内存中的模型数据长度，单位Byte。
modelId	输出	模型ID的指针。 系统成功加载模型后会返回的模型ID。
inputQ	输入	队列ID的指针，一个模型的输入对应一个队列ID。
inputQNum	输入	输入队列大小。
outputQ	输入	队列ID的指针，一个模型的输出对应一个队列ID。
outputQNum	输入	输出队列大小。

返回值说明

返回0表示成功，返回其它值表示失败。

相关接口

当前AscendCL还提供了**aclmdlSetConfigOpt**接口、**aclmdlLoadWithConfig**接口来实现模型加载，通过配置对象中的属性来区分，在加载模型时是从文件加载，还是从内存加载，以及内存是由系统内部管理，还是由用户管理。

11.12.7 aclmdlExecute

函数功能

执行模型推理，直到返回推理结果。

约束说明

- 若由于业务需求，必须在多线程中使用同一个modelId，则用户线程间需加锁，保证刷新输入输出内存、保证执行是连续操作，例如：

```
// 线程A的接口调用顺序：
lock(handle1) -> aclrtMemcpy刷新输入输出内存 -> alcmdlExecute执行推理 -> unlock(handle1)

// 线程B的接口调用顺序：
lock(handle1) -> aclrtMemcpy刷新输入输出内存 -> alcmdlExecute执行推理 -> unlock(handle1)
```

- 模型加载、模型执行、模型卸载的操作必须在同一个Context下（关于Context的创建请参见**11.6.1 aclrtCreateContext**）。
- 存放模型输入/输出数据的Device内存，可以使用以下接口申请：**aclrtMalloc**、或**aclrtMallocHost**、或**aclrtMallocCached**接口、或**acldvppMalloc**接口、或**hi_mpi_dvpp_malloc**接口。

  其中：
  - 各内存申请接口的使用场景、使用约束请参见各内存申请接口的说明。
  - **aclrtMallocHost**接口仅在如下场景才能申请Device内存：仅当AscendCL应用程序在Device上运行时，申请的才是Device内存。
  - **acldvppMalloc**接口和**hi_mpi_dvpp_malloc**接口是媒体数据处理功能专用的内存申请接口，一般从性能角度，为了减少拷贝，媒体数据处理的输出作为模型推理的输入，实现内存复用。
  - 由于硬件对内存有对齐和补齐要求，若用户使用这些接口申请大块内存并自行划分、管理内存时，需满足对应接口的对齐和补齐约束，请参见**7.1 内存二次分配管理**。

函数原型

aclError aclmdlExecute(uint32_t modelId, const aclmdlDataset *input, aclmdlDataset *output)

参数说明

参数名	输入/输出	说明
modelId	输入	指定需要执行推理的模型的ID。 调用**aclmdlLoadFromFile**接口/**aclmdlLoadFromMem**接口/**aclmdlLoadFromFileWithMem**接口/**aclmdlLoadFromMemWithMem**接口加载模型成功后，会返回模型ID。
input	输入	模型推理的输入数据的指针。

参数名	输入/输出	说明

output	输出	模型推理的输出数据的指针。
		调用aclCreateDataBuffer接口创建存放对应index**输出数据**的aclDataBuffer类型时，**支持在data参数处传入nullptr，同时size需设置为0**，表示创建一个空的aclDataBuffer类型，然后在模型执行过程中，系统**内部自行计算并申请**该index输出的内存。使用该方式可节省内存，但内存数据使用结束后，需由用户释放内存并重置aclDataBuffer，同时，系统内部申请内存时涉及内存拷贝，可能涉及性能损耗。
		释放内存并重置aclDataBuffer的示例代码如下：
		aclDataBuffer *dataBuffer = aclmdlGetDatasetBuffer(output, 0); // 根据index获取对应的dataBuffer
		void *data = aclGetDataBufferAddr(dataBuffer); // 获取data的Device指针
		aclrtFree(data); // 释放Device内存
		aclUpdateDataBuffer(dataBuffer, nullptr, 0); // 重置dataBuffer里面内容，以便下次推理

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例代码，参见**4.6.2 模型执行**。

11.12.8 aclmdlExecuteAsync

函数功能

执行模型推理，异步接口。

约束说明

- 该接口是异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功。调用该接口后，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保任务已执行完成。
- 对同一个modelId的模型，不能调用aclmdlExecuteAsync接口执行多Stream并发场景下的模型推理。错误示例如下，该示例中，两次aclmdlExecuteAsync接口多Stream并发执行，导致报错：
```
//......
aclmdlExecuteAsync(modelId1, input, output, stream1);
aclmdlExecuteAsync(modelId1, input, output, stream2);
aclrtSynchronizeStream(stream1);
aclrtSynchronizeStream(stream2);
//......
```
- 若由于业务需求，必须在多线程中使用同一个modelId，则用户线程间需加锁，保证刷新输入输出内存、保证执行是连续操作，例如：
```
// 线程A的接口调用顺序：
lock(handle1) -> aclrtMemcpyAsync(stream1)刷新输入输出内存 ->
aclmdlExecuteAsync(modelId1,stream1)执行推理 -> unlock(handle1)

// 线程B的接口调用顺序：
```

```
lock(handle1) -> aclrtMemcpyAsync(stream1)刷新输入输出内存 ->
aclmdlExecuteAsync(modelId1,stream1)执行推理 -> unlock(handle1)
```

- 模型加载、模型执行、模型卸载的操作必须在同一个Context下（关于Context的创建请参见**11.6.1 aclrtCreateContext**）。
- 存放模型输入/输出数据的内存，可以使用以下接口申请：**aclrtMalloc**、或**aclrtMallocHost**、或**aclrtMallocCached**接口、或**acldvppMalloc**接口、或**hi_mpi_dvpp_malloc**接口。

  其中，**acldvppMalloc**接口和**hi_mpi_dvpp_malloc**接口是媒体数据处理功能专用的内存申请接口，一般从性能角度，为了减少拷贝，媒体数据处理的输出作为模型推理的输入，实现内存复用。

  由于硬件对内存有对齐和补齐要求，若用户使用这些接口申请大块内存并自行划分、管理内存时，需满足对应接口的对齐和补齐约束，请参见**7.1 内存二次分配管理**。

函数原型

aclError aclmdlExecuteAsync(uint32_t modelId, const aclmdlDataset *input, aclmdlDataset *output, aclrtStream stream)

参数说明

参数名	输入/输出	说明
modelId	输入	指定需要执行推理的模型的ID。 调用**aclmdlLoadFromFile**接口/**aclmdlLoadFromMem**接口/**aclmdlLoadFromFileWithMem**接口/**aclmdlLoadFromMemWithMem**接口加载模型成功后，会返回模型ID。
input	输入	模型推理的输入数据的指针。
output	输出	模型推理的输出数据的指针。
stream	输入	指定Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例代码，参见**7.6 异步模型推理**。

11.12.9 aclmdlSetExecConfigOpt

函数功能

设置模型执行的配置对象中的各属性的取值。

约束说明

本接口需要配合其它接口一起使用，实现模型执行，接口调用顺序如下：

1. 调用**aclmdlCreateExecConfigHandle**接口创建模型执行的配置对象。
2. 多次调用**aclmdlSetExecConfigOpt**接口设置配置对象中每个属性的值。
3. 调用**aclmdlExecuteV2**或**aclmdlExecuteAsyncV2**接口指定模型执行时需要的配置信息，并进行模型执行。
4. 模型加载成功后，调用**aclmdlDestroyExecConfigHandle**接口销毁。

函数原型

aclError aclmdlSetExecConfigOpt(aclmdlExecConfigHandle *handle, aclmdlExecConfigAttr attr, const void *attrValue, size_t valueSize)

参数说明

参数名	输入/输出	说明
handle	输出	模型执行的配置对象的指针。需提前调用**aclmdlCreateExecConfigHandle**接口创建该对象。
attr	输入	指定需设置的属性。
attrValue	输入	指向属性值的指针，attr对应的属性取值。 如果属性值本身是指针，则传入该指针的地址。
valueSize	输入	attrValue部分的数据长度。 用户可使用C/C++标准库的函数sizeof(*attrValue)查询数据长度。

返回值说明

返回0表示成功，返回其它值表示失败。

11.12.10 aclmdlExecuteV2

函数功能

根据**aclmdlSetExecConfigOpt**接口所配置的属性，执行模型推理，直到返回推理结果。该接口是在**aclmdlExecute**接口基础上进行了增强，支持在执行模型推理时控制Stream任务的超时时间、Event任务的超时时间。

约束说明

- 本接口需要配合其它接口一起使用，实现模型执行，接口调用顺序如下：

a. 调用**aclmdlCreateExecConfigHandle**接口创建模型执行的配置对象。
b. 多次调用**aclmdlSetExecConfigOpt**接口设置配置对象中每个属性的值。
c. 调用aclmdlExecuteV2接口指定模型执行时需要的配置信息，并进行模型执行。
d. 模型加载成功后，调用**aclmdlDestroyExecConfigHandle**接口销毁。

- 其他约束与**aclmdlExecute**一致。

函数原型

aclError aclmdlExecuteV2(uint32_t modelId, const aclmdlDataset *input, aclmdlDataset *output, aclrtStream stream, const aclmdlExecConfigHandle *handle)

参数说明

参数名	输入/输出	说明
modelId	输入	指定需要执行推理的模型的ID。 调用**aclmdlLoadFromFile**接口/**aclmdlLoadFromMem**接口/**aclmdlLoadFromFileWithMem**接口/**aclmdlLoadFromMemWithMem**接口加载模型成功后，会返回模型ID。
input	输入	模型推理的输入数据的指针。

output	输出	模型推理的输出数据的指针。
		调用aclCreateDataBuffer接口创建存放对应index**输出数据**的aclDataBuffer类型时，**支持在data参数处传入nullptr，同时size需设置为0**，表示创建一个空的aclDataBuffer类型，然后在模型执行过程中，系统**内部自行计算并申请**该index输出的内存。使用该方式可节省内存，但内存数据使用结束后，需由用户释放内存并重置aclDataBuffer，同时，系统内部申请内存时涉及内存拷贝，可能涉及性能损耗。
		释放内存并重置aclDataBuffer的示例代码如下：
		`aclDataBuffer *dataBuffer = aclmdlGetDatasetBuffer(output, 0); // 根据index获取对应的dataBuffer`
		`void *data = aclGetDataBufferAddr(dataBuffer);  // 获取data的Device指针`
		`aclrtFree(data ); // 释放Device内存`
		`aclUpdateDataBuffer(dataBuffer, nullptr, 0); // 重置dataBuffer里面内容，以便下次推理`

stream	输入	指定Stream。
handle	输入	模型执行的配置对象的指针。与**aclmdlSetExecConfigOpt**中的handle保持一致。

返回值说明

返回0表示成功，返回其它值表示失败。

11.12.11 aclmdlExecuteAsyncV2

函数功能

根据**aclmdlSetExecConfigOpt**所配置的属性，执行模型推理，直到返回推理结果。该接口是在**aclmdlExecuteAsync**接口基础上进行了增强，支持在执行模型推理时控制Stream任务的超时时间、Event任务的超时时间。异步接口。

Atlas 200/300/500 推理产品不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 本接口需要配合其它接口一起使用，实现模型执行，接口调用顺序如下：
  a. 调用**aclmdlCreateExecConfigHandle**接口创建模型执行的配置对象。
  b. 多次调用**aclmdlSetExecConfigOpt**接口设置配置对象中每个属性的值。
  c. 调用aclmdlExecuteAsyncV2接口指定模型执行时需要的配置信息，并进行模型执行。
  d. 模型加载成功后，调用**aclmdlDestroyExecConfigHandle**接口销毁。
- 该接口是异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功。调用该接口后，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保任务已执行完成。
- 其他约束与**aclmdlExecuteAsync**一致。

函数原型

aclError aclmdlExecuteAsyncV2(uint32_t modelId, const aclmdlDataset *input, aclmdlDataset *output, aclrtStream stream, const aclmdlExecConfigHandle *handle)

参数说明

参数名	输入/输出	说明
modelId	输入	指定需要执行推理的模型的ID。 调用**aclmdlLoadFromFile**接口/**aclmdlLoadFromMem**接口/**aclmdlLoadFromFileWithMem**接口/**aclmdlLoadFromMemWithMem**接口加载模型成功后，会返回模型ID。
input	输入	模型推理的输入数据的指针。
output	输出	模型推理的输出数据的指针。
stream	输入	指定Stream。

参数名	输入/输出	说明
handle	输入	模型执行的配置对象的指针。与aclmdlSetExecConfigOpt中的handle保持一致。

返回值说明

返回0表示成功，返回其它值表示失败。

11.12.12 aclmdlUnload

函数功能

系统完成模型推理后，可调用该接口卸载模型，释放资源。

约束说明

- 在调用aclmdlUnload接口卸载指定模型时，需确保其它接口没有正在使用该模型。
- 模型加载、模型执行、模型卸载的操作必须在同一个Context下（关于Context的创建请参见11.6.1 aclrtCreateContext）。

函数原型

aclError aclmdlUnload(uint32_t modelId)

参数说明

参数名	输入/输出	说明
modelId	输入	需卸载的模型的ID。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程，参见3.2 AscendCL接口调用流程。

11.12.13 aclmdlQuerySize

函数功能

根据模型文件获取模型执行时所需的权值内存大小、工作内存大小。

约束说明

当由用户管理内存时，为确保内存不浪费，在申请工作内存、权值内存前，需要调用**aclmdlQuerySize**接口查询模型运行时所需工作内存、权值内存的大小。

如果模型输入数据的Shape不确定，则不能调用**aclmdlQuerySize**接口查询内存大小，在加载模型时，就无法由用户管理内存，因此需选择由系统管理内存的模型加载接口（例如，**aclmdlLoadFromFile**、**aclmdlLoadFromMem**）。

函数原型

aclError aclmdlQuerySize(const char *fileName, size_t *workSize, size_t *weightSize)

参数说明

参数名	输入/输出	说明
fileName	输入	模型文件路径的指针，路径中包含文件名。运行程序（APP）的用户需要对该路径有访问权限。
workSize	输出	模型执行时所需的工作内存大小的指针，单位Byte。
weightSize	输出	模型执行时所需权值内存大小的指针，单位Byte。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例代码，参见**4.6.1 模型加载**。

11.12.14 aclmdlQuerySizeFromMem

函数功能

根据内存中的模型数据获取模型执行时所需的权值内存大小、内存大小。

约束说明

工作和权重内存为Device内存，而且需要用户申请和释放。

函数原型

aclError aclmdlQuerySizeFromMem(const void *model, size_t modelSize, size_t *workSize, size_t *weightSize)

参数说明

参数名	输入/输出	说明
model	输入	模型数据的指针。
modelSize	输入	模型数据长度，单位Byte。
workSize	输出	模型执行时所需的工作内存大小的指针，单位Byte。
weightSize	输出	模型执行时所需权值内存大小的指针，单位Byte。

返回值说明

返回0表示成功，返回其它值表示失败。

11.12.15 aclmdlSetDynamicBatchSize

函数功能

在动态Batch场景下，在模型执行前调用本接口设置模型推理时的批量大小Batch（每次处理图片的数量）。

函数原型

aclError aclmdlSetDynamicBatchSize(uint32_t modelId, aclmdlDataset *dataset, size_t index, uint64_t batchSize)

参数说明

参数名	输入/输出	说明
modelId	输入	模型ID。 调用aclmdlLoadFromFile接口/aclmdlLoadFromMem接口/aclmdlLoadFromFileWithMem接口/aclmdlLoadFromMemWithMem接口加载模型成功后，会返回模型ID。
dataset	输入&输出	模型推理的输入数据的指针。 使用aclmdlDataset类型的数据描述模型推理时的输入数据，输入的内存地址、内存大小用aclDataBuffer类型的数据来描述。

参数名	输入/输出	说明
index	输入	标识动态Batch输入的输入index，需调用**aclmdlGetInputIndexByName**接口获取，输入名称固定为ACL_DYNAMIC_TENSOR_NAME。
batchSize	输入	模型推理时的批量大小Batch。 此处设置的batch size只能是模型构建时设置的Batch档位中的某一档。 如果不清楚模型构建时的Batch档位设置，也可以调用**aclmdlGetDynamicBatch**接口获取指定模型支持的Batch档位数以及每一档中的batch size。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**7.7 模型动态Shape输入推理**。

11.12.16 aclmdlSetDynamicHWSize

函数功能

动态分辨率场景下，在模型执行前调用本接口设置模型推理时输入图片的高和宽。

函数原型

aclError aclmdlSetDynamicHWSize(uint32_t modelId, aclmdlDataset *dataset, size_t index, uint64_t height, uint64_t width)

参数说明

参数名	输入/输出	说明
modelId	输入	模型ID。 调用**aclmdlLoadFromFile**接口/**aclmdlLoadFromMem**接口/**aclmdlLoadFromFileWithMem**接口/**aclmdlLoadFromMemWithMem**接口加载模型成功后，会返回模型ID。

参数名	输入/输出	说明
dataset	输入&输出	模型推理的输入数据的指针。 使用aclmdlDataset类型的数据描述模型推理时的输入数据，输入的内存地址、内存大小用**aclDataBuffer**类型的数据来描述。
index	输入	标识动态分辨率输入的输入index，需调用**aclmdlGetInputIndexByName**接口获取，输入名称固定为ACL_DYNAMIC_TENSOR_NAME。
height	输入	需设置的H值。 此处设置的分辨率只能是模型构建时设置的分辨率档位中的某一档。 如果不清楚模型构建时的分辨率档位，也可以调用**aclmdlGetDynamicHW**接口获取指定模型支持的分辨率档位数以及每一档中的宽、高。
width	输入	需设置的W值。 此处设置的分辨率只能是模型构建时设置的分辨率档位中的某一档。 如果不清楚模型构建时的分辨率档位，也可以调用**aclmdlGetDynamicHW**接口获取指定模型支持的分辨率档位数以及每一档中的宽、高。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**7.7 模型动态Shape输入推理**。

11.12.17 aclmdlSetInputAIPP

函数功能

动态AIPP场景下，根据指定的动态AIPP输入的输入index，设置模型推理时的AIPP参数值。

动态AIPP支持的几种操作的计算方式及其计算顺序如下：抠图->色域转换->缩放（当前版本不支持缩放）->减均值/归一化->padding。

约束说明

- 经过动态AIPP处理后的图像的宽、高必须与原始模型中输入Shape中的宽、高保持一致。
- 多Batch场景下，根据每个Batch的配置计算出动态AIPP后输出图片的宽、高，经过动态AIPP后每个Batch的输出图片宽、高必须是一致的。计算输出图片宽、高的计算公式如**表11-1**所示。
- 抠图或者缩放或者padding之后，对图片宽、高的校验规则如下，其中，aippOutputW、aippOutputH分别表示AIPP输出图片的宽、高，其它参数是**aclmdlSetAIPPSrcImageSize**、**aclmdlSetAIPPScfParams**、**aclmdlSetAIPPCropParams**、**aclmdlSetAIPPPaddingParams**接口的入参：

表 11-1 输出图片宽、高计算公式

抠图	缩放	补边（padding）	动态AIPP输出图片的宽、高
否	否	否	aippOutputW=srcImageSizeW，aippOutputH=srcImageSizeH
是	否	否	aippOutputW=cropSizeW，aippOutputH=cropSizeH
是	是	否	aippOutputW=scfOutputSizeW，aippOutputH=scfOutputSizeH
是	否	是	aippOutputW=cropSizeW + paddingSizeLeft + paddingSizeRight，aippOutputH=cropSizeH + paddingSizeTop + paddingSizeBottom
否	否	是	aippOutputW=srcImageSizeW + paddingSizeLeft + paddingSizeRight，aippOutputH=srcImageSizeH + paddingSizeTop + paddingSizeBottom
否	是	是	aippOutputW=scfOutputSizeW + paddingSizeLeft + paddingSizeRight，aippOutputH=scfOutputSizeH + paddingSizeTop + paddingSizeBottom
否	是	否	aippOutputW=scfOutputSizeW，aippOutputH=scfOutputSizeH

抠图	缩放	补边（padding）	动态AIPP输出图片的宽、高
是	是	是	aippOutputW=scfOutputSizeW + paddingSizeLeft + paddingSizeRight，aippOutputH=scfOutputSizeH + paddingSizeTop + paddingSizeBottom

函数原型

aclError aclmdlSetInputAIPP(uint32_t modelId, aclmdlDataset *dataset, size_t index, const aclmdlAIPP *aippParmsSet)

参数说明

参数名	输入/输出	说明
modelId	输入	模型的ID。 调用**aclmdlLoadFromFile**接口/**aclmdlLoadFromMem**接口/**aclmdlLoadFromFileWithMem**接口/**aclmdlLoadFromMemWithMem**接口加载模型成功后，会返回模型ID。
dataset	输入	模型推理的输入数据的指针。 使用aclmdlDataset类型的数据描述模型推理时的输入数据，输入的内存地址、内存大小用**aclDataBuffer**类型的数据来描述。
index	输入	标识动态AIPP输入的输入index。 多个动态AIPP输入的场景下，用户可调用**aclmdlGetAippType**接口获取指定模型输入所关联的动态AIPP输入的输入index。 为保证向前兼容，如果明确只有一个动态AIPP输入，可调用**aclmdlGetInputIndexByName**接口获取，输入名称固定为ACL_DYNAMIC_AIPP_NAME。
aippParmsSet	输入	动态AIPP参数对象的指针。 提前调用**aclmdlCreateAIPP**接口创建aclmdlAIPP类型的数据。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见7.8 模型动态AIPP推理。

11.12.18 aclmdlGetFirstAippInfo

函数功能

获取模型AIPP（包括静态AIPP和动态AIPP）的配置信息。

AIPP支持的几种操作的计算方式及其计算顺序如下：抠图->色域转换->缩放（当前版本不支持缩放）->减均值/归一化->padding。

约束说明

如果使用本接口获取模型中动态AIPP的信息，只能获取aclAippInfo结构体中如下参数的值：srcFormat、srcDatatype、srcDimNum、shapeCount、outDims，其它参数值无效。

函数原型

aclError aclmdlGetFirstAippInfo(uint32_t modelId, size_t index, aclAippInfo *aippInfo)

参数说明

参数名	输入/输出	说明
modelId	输入	模型ID。 调用aclmdlLoadFromFile接口/aclmdlLoadFromMem接口/aclmdlLoadFromFileWithMem接口/aclmdlLoadFromMemWithMem接口加载模型成功后，会返回模型ID。
index	输入	模型中输入的index。
aippInfo	输出	获取指定输入上AIPP配置信息的指针。 详细说明及参数解释，请参考《ATC工具使用指南（开放态）》。

返回值说明

返回0表示成功，返回非0表示失败。

11.12.19 aclmdlGetAippType

函数功能

获取指定模型的指定输入所支持的AIPP类型（动态AIPP或静态AIPP）及动态AIPP输入对应的index值。

函数原型

aclError aclmdlGetAippType(uint32_t modelId, size_t index, aclmdlInputAippType *type, size_t *dynamicAttachedDataIndex)

参数说明

参数名	输入/输出	说明
modelId	输入	指定模型的ID。 调用aclmdlLoadFromFile接口/aclmdlLoadFromMem接口/aclmdlLoadFromFileWithMem接口/aclmdlLoadFromMemWithMem接口加载模型成功后，会返回模型ID。
index	输入	模型中输入的index。
type	输出	指定模型输入的AIPP类型的指针。
dynamicAttachedDataIndex	输出	当type不为ACL_DATA_WITH_DYNAMIC_AIPP时，该值返回0xFFFFFFFF，表示无效。 当type为ACL_DATA_WITH_DYNAMIC_AIPP时，该值返回动态AIPP输入（用于配置动态AIPP参数）的index。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见7.8.3 动态AIPP（多个动态AIPP输入）。

11.12.20 aclmdlSetAIPPByInputIndex

函数功能

动态AIPP场景下，根据指定的模型输入的输入index，设置模型推理时的AIPP参数值。

动态AIPP支持的几种操作的计算方式及其计算顺序如下：抠图->色域转换->缩放（当前版本不支持缩放）->减均值/归一化->padding。

约束说明

- 经过动态AIPP处理后的图像的宽、高必须与原始模型中输入Shape中的宽、高保持一致。
- 多Batch场景下，根据每个Batch的配置计算出动态AIPP后输出图片的宽、高，经过动态AIPP后每个Batch的输出图片宽、高必须是一致的。计算输出图片宽、高的计算公式如**表11-2**所示。
- 抠图或者缩放或者padding之后，对图片宽、高的校验规则如下，其中，aippOutputW、aippOutputH分别表示AIPP输出图片的宽、高，其它参数是**aclmdlSetAIPPSrcImageSize**、**aclmdlSetAIPPScfParams**、**aclmdlSetAIPPCropParams**、**aclmdlSetAIPPPaddingParams**接口的入参：

表 11-2 输出图片宽、高计算公式

抠图	缩放	补边（padding）	动态AIPP输出图片的宽、高
否	否	否	aippOutputW=srcImageSizeW，aippOutputH=srcImageSizeH
是	否	否	aippOutputW=cropSizeW，aippOutputH=cropSizeH
是	是	否	aippOutputW=scfOutputSizeW，aippOutputH=scfOutputSizeH
是	否	是	aippOutputW=cropSizeW + paddingSizeLeft + paddingSizeRight，aippOutputH=cropSizeH + paddingSizeTop + paddingSizeBottom
否	否	是	aippOutputW=srcImageSizeW + paddingSizeLeft + paddingSizeRight，aippOutputH=srcImageSizeH + paddingSizeTop + paddingSizeBottom
否	是	是	aippOutputW=scfOutputSizeW + paddingSizeLeft + paddingSizeRight，aippOutputH=scfOutputSizeH + paddingSizeTop + paddingSizeBottom

抠图	缩放	补边（padding）	动态AIPP输出图片的宽、高
否	是	否	aippOutputW=scfOutputSizeW，aippOutputH=scfOutputSizeH
是	是	是	aippOutputW=scfOutputSizeW + paddingSizeLeft + paddingSizeRight，aippOutputH=scfOutputSizeH + paddingSizeTop + paddingSizeBottom

函数原型

aclError aclmdlSetAIPPByInputIndex(uint32_t modelId, aclmdlDataset *dataset, size_t index, const aclmdlAIPP *aippParmsSet)

参数说明

参数名	输入/输出	说明
modelId	输入	模型的ID。 调用**aclmdlLoadFromFile**接口/**aclmdlLoadFromMem**接口/**aclmdlLoadFromFileWithMem**接口/**aclmdlLoadFromMemWithMem**接口加载模型成功后，会返回模型ID。
dataset	输入	模型推理的输入数据的指针。 使用aclmdlDataset类型的数据描述模型推理时的输入数据，输入的内存地址、内存大小用**aclDataBuffer**类型的数据来描述。
index	输入	表示在第几个输入上设置动态AIPP参数。如果该输入没有关联动态AIPP输入，则返回报错。 可调用**aclmdlGetAippType**查询指定模型的指定输入是否有关联的动态AIPP输入，若有，则本接口的index参数值与**aclmdlGetAippType**接口的index参数值保持一致；若无，则无需设置动态AIPP参数。
aippParmsSet	输入	动态AIPP参数对象的指针。 提前调用**aclmdlCreateAIPP**接口创建aclmdlAIPP类型的数据。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见7.8.3 动态AIPP（多个动态AIPP输入）。

11.12.21 aclmdlSetInputDynamicDims

函数功能

如果模型输入的Shape是动态的、输入数据Format为ND格式（ND表示支持任意格式，N<=4），在模型执行前调用本接口设置模型推理时具体维度的值。

函数原型

aclError aclmdlSetInputDynamicDims(uint32_t modelId, aclmdlDataset *dataset, size_t index, const aclmdlIODims *dims)

参数说明

参数名	输入/输出	说明
modelId	输入	模型ID。 调用aclmdlLoadFromFile接口/aclmdlLoadFromMem接口/aclmdlLoadFromFileWithMem接口/aclmdlLoadFromMemWithMem接口加载模型成功后，会返回模型ID。
dataset	输入&输出	模型推理的输入数据的指针。 使用aclmdlDataset类型的数据描述模型推理时的输入数据，输入的内存地址、内存大小用aclDataBuffer类型的数据来描述。
index	输入	标识动态维度的输入index。 需调用aclmdlGetInputIndexByName接口获取，输入名称固定为ACL_DYNAMIC_TENSOR_NAME。

参数名	输入/输出	说明
dims	输入	具体某一档上的所有维度信息的指针。 此处设置的动态维度的值只能是模型构建时设置的档位中的某一档。 如果不清楚模型构建时的动态维度档位，也可以调用**aclmdlGetInputDynamicDims**接口获取指定模型支持的动态维度档位数以及每一档中的值。 **例如：**使用ATC工具进行模型转换时，input_shape="data:1,1,40,-1;label:1,-1;mask:-1,-1" ，dynamic_dims="20,20,1,1; 40,40,2,2; 80,60,4,4"，若输入数据的真实维度为（1,1,40,20,1,20,1,1），则dims结构体信息的填充示例如下（name暂不使用）： dims.dimCount = 8 dims.dims[0] = 1 dims.dims[1] = 1 dims.dims[2] = 40 dims.dims[3] = 20 dims.dims[4] = 1 dims.dims[5] = 20 dims.dims[6] = 1 dims.dims[7] = 1

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**7.7 模型动态Shape输入推理**。

11.12.22 aclmdlCreateAndGetOpDesc

函数功能

获取指定算子的描述信息，包括算子名、输入tensor描述、输出tensor描述，如果查询不到指定算子，则返回报错。该接口不支持动态Shape场景。

约束说明

使用场景举例：执行整网模型推理时（**不支持动态Shape场景**），如果产生AI Core报错，可以调用本接口获取报错算子的描述信息，再做进一步错误排查。

推荐的接口调用顺序如下：

1. 定义并实现异常回调函数fn(aclrtExceptionInfoCallback类型)，回调函数原型请参见**11.19.1 aclrtSetExceptionInfoCallback** 。

   实现回调函数的关键步骤如下：

   a. 在异常回调函数fn内调用**aclrtGetDeviceIdFromExceptionInfo**、**aclrtGetStreamIdFromExceptionInfo**、**aclrtGetTaskIdFromExceptionInfo**接口分别获取Device ID、Stream ID、Task ID。

   b. 在异常回调函数fn内调用**aclmdlCreateAndGetOpDesc**接口获取算子的描述信息。

   c. 在异常回调函数fn内调用**aclGetTensorDescByIndex**接口获取指定算子输入/输出的tensor描述。

   d. 在异常回调函数fn内如下接口获取tensor描述中的数据，进行进一步分析。

      例如，调用**aclGetTensorDescAddress**接口获取tensor数据的内存地址（用户可从该内存地址中获取tensor数据）、调用**aclGetTensorDescType**接口获取tensor描述中的数据类型、调用**aclGetTensorDescFormat**接口获取tensor描述中的Format、调用**aclGetTensorDescNumDims**接口获取tensor描述中的Shape维度个数、调用**aclGetTensorDescDimV2**接口获取Shape中指定维度的大小。

2. 调用**aclrtSetExceptionInfoCallback**接口设置异常回调函数。
3. 执行模型推理。

   如果存在AI Core报错，则触发回调函数fn，获取算子的信息，进行进一步分析。

函数原型

aclError aclmdlCreateAndGetOpDesc(uint32_t deviceId, uint32_t streamId, uint32_t taskId, char *opName, size_t opNameLen, aclTensorDesc **inputDesc, size_t *numInputs, aclTensorDesc **outputDesc, size_t *numOutputs)

参数说明

参数名	输入/输出	说明
deviceId	输入	Device ID。 调用**aclrtGetDeviceIdFromExceptionInfo**接口获取异常信息中的Device ID，作为本接口的输入。
streamId	输入	Stream ID。 调用**aclrtGetStreamIdFromExceptionInfo**接口获取异常信息中的Stream ID，作为本接口的输入。

参数名	输入/输出	说明
taskId	输入	Task ID。 调用**aclrtGetTaskIdFromExceptionInfo**接口获取异常信息中的Task ID，作为本接口的输入。
opName	输出	算子名称字符串的指针。
opNameLen	输入	算子名称字符串长度。 若用户指定的长度比实际算子名称的长度短，则返回报错。
inputDesc	输出	算子所有输入的tensor描述的指针，指向一块连续内存的首地址。
numInputs	输出	输入个数的指针。
outputDesc	输出	算子所有输出的tensor描述的指针，指向一块连续内存的首地址。
numOutputs	输出	输出个数的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见**7.11 AI Core异常信息获取**。

11.12.23 aclmdlInitDump

函数功能

Dump初始化。

约束说明

- **aclmdlInitDump**接口需要与**aclmdlSetDump**接口、**aclmdlFinalizeDump**接口配合使用，用于将Dump数据记录到文件中。一个进程内，可以根据需求多次调用这些接口，基于不同的Dump配置信息，获取Dump数据。

  场景举例：

  – 两次模型执行，需要设置不同的Dump配置信息，接口调用顺序：**aclInit**接口-->**aclmdlInitDump**接口-->**aclmdlSetDump**接口-->模型加载-->模型执行-->**aclmdlFinalizeDump**接口-->模型卸载-->**aclmdlInitDump**接口-->**aclmdlSetDump**接口-->模型加载-->模型执行-->**aclmdlFinalizeDump**接口-->模型卸载-->执行其它任务-->**aclFinalize**接口
  – 同一个模型执行两次，第一次需要Dump，第二次无需Dump，接口调用顺序：**aclInit**接口-->**aclmdlInitDump**接口-->**aclmdlSetDump**接口-->模型加

载-->模型执行-->**aclmdlFinalizeDump**接口-->模型卸载-->模型加载-->模型执行-->执行其它任务-->**aclFinalize**接口

- 如果已经通过**aclInit**接口配置了dump信息，则调用aclmdlInitDump接口时会返回失败。
- 必须在调用**aclInit**接口之后、模型加载接口之前调用aclmdlInitDump接口。

函数原型

aclError aclmdlInitDump()

参数说明

无

返回值说明

返回0表示成功，返回其它值表示失败。

相关接口

AscendCL还提供了**aclInit**接口，在AscendCL初始化阶段，通过*.json文件传入Dump配置信息，运行应用后获取Dump数据的方式。该种方式，一个进程内，只能调用一次**aclInit**接口，如果要修改Dump配置信息，需修改*.json文件中的配置。

11.12.24 aclmdlSetDump

函数功能

设置dump参数。

约束说明

- **aclmdlInitDump**接口需要与**aclmdlSetDump**接口、**aclmdlFinalizeDump**接口配合使用，用于将Dump数据记录到文件中。一个进程内，可以根据需求多次调用这些接口，基于不同的Dump配置信息，获取Dump数据。

  场景举例：

  - 两次模型执行，需要设置不同的Dump配置信息，接口调用顺序：**aclInit**接口-->**aclmdlInitDump**接口-->**aclmdlSetDump**接口-->模型加载-->模型执行-->**aclmdlFinalizeDump**接口-->模型卸载-->**aclmdlInitDump**接口-->**aclmdlSetDump**接口-->模型加载-->模型执行-->**aclmdlFinalizeDump**接口-->模型卸载-->执行其它任务-->**aclFinalize**接口
  - 同一个模型执行两次，第一次需要Dump，第二次无需Dump，接口调用顺序：**aclInit**接口-->**aclmdlInitDump**接口-->**aclmdlSetDump**接口-->模型加载-->模型执行-->**aclmdlFinalizeDump**接口-->模型卸载-->模型加载-->模型执行-->执行其它任务-->**aclFinalize**接口
- 只有在调用本接口之后加载模型，配置的Dump信息有效。在调用本接口之前已经加载的模型不受影响，除非用户在调用本接口后重新加载该模型。

  例如以下接口调用顺序中，加载的模型1不受影响，配置的Dump信息仅对加载的模型2有效：

  aclmdlInitDump接口-->模型1加载-->aclmdlSetDump接口-->模型2加载-->**aclmdlFinalizeDump**接口

- 多次调用本接口对同一个模型配置了Dump信息，系统内处理时会采用覆盖策略。

  例如以下接口调用顺序中，第二次调用本接口配置的Dump信息会覆盖第一次配置的Dump信息：

  aclmdlInitDump接口-->aclmdlSetDump接口-->aclmdlSetDump接口-->模型1加载-->**aclmdlFinalizeDump**接口

函数原型

aclError aclmdlSetDump(const char *dumpCfgPath)

参数说明

参数名	输入/输出	说明
dumpCfgPath	输入	配置文件路径的指针，包含文件名。 配置文件格式为json格式，当前可配置dump数据的相关信息，示例请参见**配置文件示例**，详细配置说明请参见《**CANN 开发工具指南（开放态）**》中的“精度比对工具”章节中的“比对数据准备>推理场景数据准备>准备离线模型dump数据文件（AscendCL接口方式）”。

配置文件示例

以Caffe ResNet-50网络为例，若需要比对Caffe ResNet-50网络与基于Caffe ResNet-50转换成的适配昇腾AI处理器的离线模型中某些层算子的输出结果，可以在配置文件中配置如下内容：

```
{
   "dump":{
      "dump_list":[
         {
            "model_name":"ResNet-50",
            "layer":[
                  "conv1conv1_relu",
                  "res2a_branch2ares2a_branch2a_relu",
                  "res2a_branch1",
                  "pool1"
            ]
         }
      ],
      "dump_path":"/MyApp20/dump",
            "dump_mode":"output"
   }

}
```

返回值说明

返回0表示成功，返回其它值表示失败。

相关接口

AscendCL还提供了**aclInit**接口，在AscendCL初始化阶段，通过*.json文件传入Dump配置信息，运行应用后获取Dump数据的方式。该种方式，一个进程内，只能调用一次**aclInit**接口，如果要修改Dump配置信息，需修改*.json文件中的配置。

11.12.25 aclmdlFinalizeDump

函数功能

Dump去初始化。

约束说明

- **aclmdlInitDump**接口需要与**aclmdlSetDump**接口、**aclmdlFinalizeDump**接口配合使用，用于将Dump数据记录到文件中。一个进程内，可以根据需求多次调用这些接口，基于不同的Dump配置信息，获取Dump数据。

  场景举例：

  - 两次模型执行，需要设置不同的Dump配置信息，接口调用顺序：**aclInit**接口-->**aclmdlInitDump**接口-->**aclmdlSetDump**接口-->模型加载-->模型执行-->**aclmdlFinalizeDump**接口-->模型卸载-->**aclmdlInitDump**接口-->**aclmdlSetDump**接口-->模型加载-->模型执行-->**aclmdlFinalizeDump**接口-->模型卸载-->执行其它任务-->**aclFinalize**接口
  - 同一个模型执行两次，第一次需要Dump，第二次无需Dump，接口调用顺序：**aclInit**接口-->**aclmdlInitDump**接口-->**aclmdlSetDump**接口-->模型加载-->模型执行-->**aclmdlFinalizeDump**接口-->模型卸载-->模型加载-->模型执行-->执行其它任务-->**aclFinalize**接口

函数原型

aclError aclmdlFinalizeDump()

参数说明

无

返回值说明

返回0表示成功，返回其它值表示失败。

相关接口

AscendCL还提供了**aclInit**接口，在AscendCL初始化阶段，通过*.json文件传入Dump配置信息，运行应用后获取Dump数据的方式。该种方式，一个进程内，只能调用一次**aclInit**接口，如果要修改Dump配置信息，需修改*.json文件中的配置。

11.12.26 aclmdlSetConfigOpt

函数功能

设置模型加载的配置对象中的各属性的取值，包括模型执行的优先级、模型的文件路径或内存地址、内存大小等。

本接口需要与以下其它接口配合，实现模型加载功能：

1. 调用**aclmdlCreateConfigHandle**接口创建模型加载的配置对象。
2. 多次调用**aclmdlSetConfigOpt**接口设置配置对象中每个属性的值。
3. 调用**aclmdlLoadWithConfig**接口指定模型加载时需要的配置信息，并进行模型加载。
4. 模型加载成功后，调用**aclmdlDestroyConfigHandle**接口销毁。

函数原型

aclError aclmdlSetConfigOpt(aclmdlConfigHandle *handle, aclmdlConfigAttr attr, const void *attrValue, size_t valueSize)

参数说明

参数名	输入/输出	说明
handle	输出	模型加载的配置对象的指针。需提前调用**aclmdlCreateConfigHandle**接口创建该对象。
attr	输入	指定需设置的属性。
attrValue	输入	指向属性值的指针，attr对应的属性取值。 如果属性值本身是指针，则传入该指针的地址。
valueSize	输入	attrValue部分的数据长度。 用户可使用C/C++标准库的函数sizeof(*attrValue)查询数据长度。

返回值说明

返回0表示成功，返回其它值表示失败。

相关接口

使用**aclmdlSetConfigOpt**接口、**aclmdlLoadWithConfig**接口时，是通过配置对象中的属性来区分，在加载模型时是从文件加载，还是从内存加载，以及内存是由系统内部管理，还是由用户管理。

当前AscendCL还提供了以下接口实现模型加载的功能，从使用的接口上区分从文件加载，还是从内存加载，以及内存是由系统内部管理，还是由用户管理。

- **aclmdlLoadFromFile**接口
- **aclmdlLoadFromMem**接口
- **aclmdlLoadFromFileWithMem**接口
- **aclmdlLoadFromMemWithMem**接口
- **aclmdlLoadFromFileWithQ**接口

- **aclmdlLoadFromMemWithQ**接口

11.12.27 aclmdlLoadWithConfig

函数功能

指定模型加载时需要的配置信息，并进行模型加载。

本接口需要与以下其它接口配合，实现模型加载功能：

1. 调用**aclmdlCreateConfigHandle**接口创建模型加载的配置对象。
2. 多次调用**aclmdlSetConfigOpt**接口设置配置对象中每个属性的值。
3. 调用**aclmdlLoadWithConfig**接口指定模型加载时需要的配置信息，并进行模型加载。
4. 模型加载成功后，调用**aclmdlDestroyConfigHandle**接口销毁。

约束说明

在加载前，请先根据模型文件的大小评估内存空间是否足够，内存空间不足，会导致应用程序异常。

函数原型

aclError aclmdlLoadWithConfig(const aclmdlConfigHandle *handle, uint32_t *modelId)

参数说明

参数名	输入/输出	说明
handle	输入	模型加载的配置对象的指针。需提前调用**aclmdlCreateConfigHandle**接口创建该对象，与aclmdlSetConfigOpt中的handle保持一致。
modelId	输出	模型ID的指针。 系统成功加载模型后会返回的模型ID。

返回值说明

返回0表示成功，返回其它值表示失败。

相关接口

使用**aclmdlSetConfigOpt**接口、**aclmdlLoadWithConfig**接口时，是通过配置对象中的属性来区分，在加载模型时是从文件加载，还是从内存加载，以及内存是由系统内部管理，还是由用户管理。

当前AscendCL还提供了以下接口实现模型加载的功能，从使用的接口上区分从文件加载，还是从内存加载，以及内存是由系统内部管理，还是由用户管理。

- **aclmdlLoadFromFile**接口
- **aclmdlLoadFromMem**接口
- **aclmdlLoadFromFileWithMem**接口
- **aclmdlLoadFromMemWithMem**接口
- **aclmdlLoadFromFileWithQ**接口
- **aclmdlLoadFromMemWithQ**接口

11.13 单算子 API 执行

11.13.1 总体说明

本章主要介绍了如下几类算子API，提供了详细的接口介绍，包括接口原型、功能描述、参数说明、调用示例等，适合开发人员、测试人员。

- **NN类算子**：主要包含实现数学基础运算（如加、减、乘、除等）、CNN相关操作（如卷积、池化、激活函数）等功能的算子接口。
- **DVPP类算子**：主要包含实现图像缩放、JPEG图像解码、图像归一化等功能的算子接口。
- **融合类算子**：主要将多个独立的“小算子”（向量Vector、矩阵Cube等）融合成一个“大算子”，达到提升算子性能的目的。

11.13.2 功能支持度说明

通过单算子API执行方式调用算子时，昇腾AI处理器对各类算子接口的支持度如下表所示。

版本	NN类算子接口（aclnn）	DVPP类算子接口（acldvpp）	融合类算子接口（aclnn）
Atlas 200/300/500 推理产品	x	x	x
Atlas 推理系列产品	x	x	x

11.13.3 基本概念

11.13.3.1 两段式接口

单算子API执行的算子接口一般定义为“两段式接口”，以NN类算子接口定义为例：

```
aclnnStatus aclnnXxxGetWorkspaceSize(const aclTensor *src, ..., aclTensor *out, ..., uint64_t workspaceSize,
aclOpExecutor **executor);
aclnnStatus aclnnXxx(void* workspace, int64 workspaceSize, aclOpExecutor* executor, aclrtStream stream);
```

其中aclnn*Xxx*GetWorkspaceSize为第一段接口，主要用于计算本次API调用计算过程中需要多少的workspace内存。获取到本次API计算需要的workspace大小后，按照workspaceSize大小申请昇腾AI处理器内存，然后调用第二段接口aclnn*Xxx*。

说明

- workspace是指除输入/输出外，API在昇腾AI处理器上完成计算所需要的临时内存。
- 第二段接口aclnn*Xxx*(...)不能重复调用，如下调用方式会出现异常：
```
aclnnXxxGetWorkspaceSize(...)
aclnnXxx(...)
aclnnXxx(...)
```

11.13.3.2 非连续的 Tensor

目前大部分算子API的输入Tensor支持“**非连续的Tensor**”，即一个Tensor可以通过(shape, strides, offset)表示。

示例 1

例如现有一个shape=(6, 5)、strides=(10, 1)、offset=22的Tensor，其内存排布如下：

$a_{0,0}$, $a_{0,1}$, $a_{0,2}$, $a_{0,3}$, $a_{0,4}$, $a_{0,5}$, $a_{0,6}$, $a_{0,7}$, $a_{0,8}$, $a_{0,9}$
$a_{1,0}$, $a_{1,1}$, $a_{1,2}$, $a_{1,3}$, $a_{1,4}$, $a_{1,5}$, $a_{1,6}$, $a_{1,7}$, $a_{1,8}$, $a_{1,9}$
$a_{2,0}$, $a_{2,1}$, $\mathbf{a_{2,2}}$, $\mathbf{a_{2,3}}$, $\mathbf{a_{2,4}}$, $\mathbf{a_{2,5}}$, $\mathbf{a_{2,6}}$, $a_{2,7}$, $a_{2,8}$, $a_{2,9}$
$a_{3,0}$, $a_{3,1}$, $\mathbf{a_{3,2}}$, $\mathbf{a_{3,3}}$, $\mathbf{a_{3,4}}$, $\mathbf{a_{3,5}}$, $\mathbf{a_{3,6}}$, $a_{3,7}$, $a_{3,8}$, $a_{3,9}$
$a_{4,0}$, $a_{4,1}$, $\mathbf{a_{4,2}}$, $\mathbf{a_{4,3}}$, $\mathbf{a_{4,4}}$, $\mathbf{a_{4,5}}$, $\mathbf{a_{4,6}}$, $a_{4,7}$, $a_{4,8}$, $a_{4,9}$
$a_{5,0}$, $a_{5,1}$, $\mathbf{a_{5,2}}$, $\mathbf{a_{5,3}}$, $\mathbf{a_{5,4}}$, $\mathbf{a_{5,5}}$, $\mathbf{a_{5,6}}$, $a_{5,7}$, $a_{5,8}$, $a_{5,9}$
$a_{6,0}$, $a_{6,1}$, $\mathbf{a_{6,2}}$, $\mathbf{a_{6,3}}$, $\mathbf{a_{6,4}}$, $\mathbf{a_{6,5}}$, $\mathbf{a_{6,6}}$, $a_{6,7}$, $a_{6,8}$, $a_{6,9}$
$a_{7,0}$, $a_{7,1}$, $\mathbf{a_{7,2}}$, $\mathbf{a_{7,3}}$, $\mathbf{a_{7,4}}$, $\mathbf{a_{7,5}}$, $\mathbf{a_{7,6}}$, $a_{7,7}$, $a_{7,8}$, $a_{7,9}$
$a_{8,0}$, $a_{8,1}$, $a_{8,2}$, $a_{8,3}$, $a_{8,4}$, $a_{8,5}$, $a_{8,6}$, $a_{8,7}$, $a_{8,8}$, $a_{8,9}$
$a_{9,0}$, $a_{9,1}$, $a_{9,2}$, $a_{9,3}$, $a_{9,4}$, $a_{9,5}$, $a_{9,6}$, $a_{9,7}$, $a_{9,8}$, $a_{9,9}$

即该Tensor排布如上图的深色位置。这个完整的Tensor在内存排布上是不连续的，strides描述Tensor维度上相邻两个元素的间隔，如果在维度1上的stride为1， 该维度是连续的；如果在维度0上的stride为10，那么相邻的元素间隔10个元素，即非连续。offset表示这个Tensor的首元素相对addr的偏移。

示例 2

例如现有一个shape=(4, 3)、strides=(20, 2)、offset=22的Tensor，其内存排布如下：

$a_{0,0}$, $a_{0,1}$, $a_{0,2}$, $a_{0,3}$, $a_{0,4}$, $a_{0,5}$, $a_{0,6}$, $a_{0,7}$, $a_{0,8}$, $a_{0,9}$
$a_{1,0}$, $a_{1,1}$, $a_{1,2}$, $a_{1,3}$, $a_{1,4}$, $a_{1,5}$, $a_{1,6}$, $a_{1,7}$, $a_{1,8}$, $a_{1,9}$
$a_{2,0}$, $a_{2,1}$, $\mathbf{a_{2,2}}$, $a_{2,3}$, $\mathbf{a_{2,4}}$, $a_{2,5}$, $\mathbf{a_{2,6}}$, $a_{2,7}$, $a_{2,8}$, $a_{2,9}$
$a_{3,0}$, $a_{3,1}$, $a_{3,2}$, $a_{3,3}$, $a_{3,4}$, $a_{3,5}$, $a_{3,6}$, $a_{3,7}$, $a_{3,8}$, $a_{3,9}$
$a_{4,0}$, $a_{4,1}$, $\mathbf{a_{4,2}}$, $a_{4,3}$, $\mathbf{a_{4,4}}$, $a_{4,5}$, $\mathbf{a_{4,6}}$, $a_{4,7}$, $a_{4,8}$, $a_{4,9}$
$a_{5,0}$, $a_{5,1}$, $a_{5,2}$, $a_{5,3}$, $a_{5,4}$, $a_{5,5}$, $a_{5,6}$, $a_{5,7}$, $a_{5,8}$, $a_{5,9}$
$a_{6,0}$, $a_{6,1}$, $\mathbf{a_{6,2}}$, $a_{6,3}$, $\mathbf{a_{6,4}}$, $a_{6,5}$, $\mathbf{a_{6,6}}$, $a_{6,7}$, $a_{6,8}$, $a_{6,9}$
$a_{7,0}$, $a_{7,1}$, $a_{7,2}$, $a_{7,3}$, $a_{7,4}$, $a_{7,5}$, $a_{7,6}$, $a_{7,7}$, $a_{7,8}$, $a_{7,9}$
$a_{8,0}$, $a_{8,1}$, $\mathbf{a_{8,2}}$, $a_{8,3}$, $\mathbf{a_{8,4}}$, $a_{8,5}$, $\mathbf{a_{8,6}}$, $a_{8,7}$, $a_{8,8}$, $a_{8,9}$
$a_{9,0}$, $a_{9,1}$, $a_{9,2}$, $a_{9,3}$, $a_{9,4}$, $a_{9,5}$, $a_{9,6}$, $a_{9,7}$, $a_{9,8}$, $a_{9,9}$

即该Tensor排布如上图的深色位置。这个完整的Tensor在内存排布上是不连续的，strides描述Tensor维度上相邻两个元素的间隔，如果在维度1上的stride为2， 该维度上间隔1个元素；如果在维度0上的stride为20，那么相邻的元素间隔20个元素，即非连续。offset表示这个Tensor的首元素相对addr的偏移。

11.13.3.3 broadcast 关系

广播概念

目前大部分算子API支持**NumPy**的broadcast特征。“broadcast”（广播）描述了NumPy在算术运算期间如何处理具有不同形状的数组。在某些情况下，较小的数组可以“广播至”较大的数组，使两者shape互相兼容。

广播提供了一种数组向量化操作，从而使得循环在C而不是Python中发生。它无需复制不必要的数据即可完成，通常算法的效率较高。然而在某些情况下，广播并不是一种好方法，因为它会导致内存使用效率降低，从而减缓计算速度。

广播规则

一般进行广播计算时，需要理解以下规则：

- 规则1：让所有输入数组都向形状最长的数组看齐。形状不足的部分通过在前面（左侧）填充1。

  说明

  形状其实就是指the number of dimensions。比如计算a+b，其中a.shape=(2, 2, 3)、b.shape=(2, 3)，那么数组b将被broadcast为b.shape=(1, 2, 3)。

- 规则2：如果两个数组的形状在任何维度上均不匹配，但是某个数组中某一个维度为1，则该维度中形状为1的数组将被拉伸以匹配另一个数组对应维度形状。

  说明

  本场景下，只需保证能在某一个维度做broadcast即可。比如计算a+b，其中a.shape=(1, 3)、b.shape=(3, 1)，那么两个数组会boradcast为a.shape=(3, 3)、b.shape=(3, 3)。

- 规则3：如果两个数组的形状在任何维度上均不匹配，且均没有等于1的维度，则会报错。

11.13.3.4 互推导关系

推导规则

当一个API（如aclnnAdd、aclnnMul等）**输入的Tensor数据类型**不一致时，API内部会推导出一个数据类型，将输入数据转换成该数据类型进行计算。

类型推导的规则如**表11-3**和**表11-4**所示。

说明

- 为方便描述，表格中使用的数据类型是简写形式，代表的含义：ACL_FLOAT(f32)、ACL_FLOAT16(f16)、ACL_DOUBLE(f64)、ACL_BF16(bf16)、ACL_INT8(s8)、ACL_UINT8(u8)、ACL_INT16(s16)、ACL_UINT16(u16)、ACL_INT32(s32)、ACL_UINT32(u32)、ACL_INT64(s64)、ACL_UINT64(u64)、ACL_BOOL(bool)、ACL_COMPLEX32(c32)、ACL_COMPLEX64(c64)、ACL_COMPLEX128(c128)。
- 表格里表头和最左侧一列分别表示待推导的两个输入数据的类型，表格中对应位置表示推导出的数据类型。
- NA表示这两种类型不能进行推导计算。
- 当输入数据类型为bool与其他任意数据类型（非u16、u32、u64），其推导出的数据类型为其他任意数据类型。当输入数据类型为bool与u16、u32、u64其中一个，均不支持数据类型推导。
- 当两个输入的数据类型都为u16，其推导出的数据类型仍为u16。当两个输入的数据类型分别为u16、非u16的任意数据类型，不支持数据类型推导。
- 当两个输入的数据类型都为u32，其推导出的数据类型仍为u32。当两个输入的数据类型分别为u32、非u32的任意数据类型，不支持数据类型推导。
- 当两个输入的数据类型都为u64，其推导出的数据类型仍为u64。当两个输入的数据类型分别为u64、非u64的任意数据类型，不支持数据类型推导。

表 11-3 常见数据类型推导关系

数据类型	f32	f16	f64	bf16	s8	u8	s16	s32	s64	c32	c64	c128
f32	f32	f32	f64	f32	f32	f32	f32	f32	f32	c64	c64	c128
f16	f32	f16	f64	f32	f16	f16	f16	f16	f16	c32	c64	c128
f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	c64	c64	c128
bf16	f32	f64	f64	bf16	bf16	bf16	bf16	bf16	bf16	c32	c64	c128
s8	f32	f16	f64	bf16	s8	s16	s16	s32	s64	c32	c64	c128
u8	f32	f16	f64	bf16	s16	u8	s16	s32	s64	c32	c64	c128
s16	f32	f16	f64	bf16	s16	s16	s16	s32	s64	c32	c64	c128
s32	f32	f16	f64	bf16	s32	s32	s32	s32	s64	c32	c64	c128
s64	f32	f16	f64	bf16	s64	s64	s64	s64	s64	c32	c64	c128
c32	c64	c32	c64	c32	c32	c32	c32	c32	c32	c32	c64	c128

数据类型	f32	f16	f64	bf16	s8	u8	s16	s32	s64	c32	c64	c128
c64	c64	c64	c64	c64	c64	c64	c64	c64	c64	c64	c64	c128
c128	c128	c128	c128	c128	c128	c128	c128	c128	c128	c128	c128	c128

表 11-4 特殊数据类型推导关系

数据类型	f32	f16	f64	bf16	s8	u8	s16	s32	s64	c32	c64	c128
u16	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
u32	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
u64	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

推导示例

- 调用aclnnAdd接口时，如果两个输入参数的数据类型不一致，一个为float16，一个为float32，那么API内部就会将float16的数据类型转换成float32的数据类型然后进行计算。
- 调用aclnnAdd接口时，如果两个输入参数的数据类型不一致，一个为float32，一个为bool，那么API内部就会将bool的数据类型转换成float32的数据类型然后进行计算。

11.13.3.5 互转换关系

当一个API（如aclnnAdd、aclnnMul等）**输出的Tensor数据类型**与**输入的数据类型**推导后的计算类型不一致时，API内部就会将计算结果转换成输出类型对应的数据类型。

数据类型转换需要满足以下规则，不满足规则的将不能进行转换，调用API时会出现参数校验失败。

- 浮点数据类型：ACL_FLOAT16、ACL_FLOAT、ACL_DOUBLE、ACL_BF16间可以互相转换。
- 整型数据类型：ACL_INT8、ACL_UINT8、ACL_INT16、ACL_UINT16、ACL_INT32、ACL_UINT32、ACL_INT64、ACL_UINT64间可以互相转换。
- 整型数据类型可以往浮点类型转换。
- 浮点数据类型不可以向整型转换。
- 非BOOL型不可以往BOOL型转换。
- 复数类型：ACL_COMPLEX64、ACL_COMPLEX128间可以转换，复数类型不可以和非复数类型互转。

11.13.3.6 数据格式

数据格式（format）是用于描述一个多维Tensor的轴的业务语义，表示数据的物理排布格式，定义了解读数据的维度，比如1D、2D、3D、4D、5D等。一般在CNN（Convolutional Neural Network）类的API中需要给出具体的数据排布格式。

说明

- 数据格式一般形式为“ACL_FORMAT_*XXXX*”，为方便描述，可简写为“*XXXX*”。
- 数据格式中维度含义：N（Batch）表示批量大小、H（Height）表示特征图高度、W（Width）表示特征图宽度、C（Channels）表示特征图通道、D（Depth）表示特征图深度。

常见数据格式

目前大部分算子API都是支持ND数据格式的。例如aclnnAdd接口，均标明支持的数据格式是ND（即多维Tensor的，低维度优先连续排布的规则）。

对于aclnnConvolution，该API要求输入的aclTensor设置具有业务语义的格式，而不是ND格式。此类算子，在计算过程中需要知道Tensor中的业务语义才可以进行对应的计算。例如2D卷积中，需要知道Batch维度、Channel维度、Height维度、Width维度与Tensor维度的对应关系是什么。

当调用aclCreateTensor接口的时候，根据API要求设置Tensor的数据格式，当前支持的数据格式有：

ACL_FORMAT_ND、ACL_FORMAT_NCHW、ACL_FORMAT_NHWC、ACL_FORMAT_HWCN、ACL_FORMAT_NDHWC、ACL_FORMAT_NCDHW、ACL_FORMAT_NC、ACL_FORMAT_NCL。

对于非ND的Tensor，Tensor的维度要求与format表述是一致的。例如：

- 5D的Tensor：要求为ACL_FORMAT_NCDHW、ACL_FORMAT_NDHWC或者ACL_FORMAT_ND（如果API参数说明中未标明支持ND，设置ND格式后API将会校验报错）。
- 4D的Tensor：要求为ACL_FORMAT_NCHW、ACL_FORMAT_NHWC、ACL_FORMAT_HWCN或者ACL_FORMAT_ND。
- 3D的Tensor：要求为ACL_FORMAT_NCL或者ACL_FORMAT_ND。
- 2D的Tensor：要求为ACL_FORMAT_NC或者ACL_FORMAT_ND。
- 其他维度的Tensor：要求为ACL_FORMAT_ND。

私有数据格式

除了常见的数据格式，还存在其他的数据格式，比如ACL_FORMAT_NC1HWC0、ACL_FORMAT_FRACTAL_Z、ACL_FORMAT_NC1HWC0_C04、ACL_FORMAT_FRACTAL_NZ、ACL_FORMAT_NDC1HWC0、ACL_FORMAT_FRACTAL_Z_3D等。

这些格式，属于NPU的一些私有格式，当前绝大多数的aclnn API还不支持这些格式。如果有个别API申明了支持的数据格式，请以该API实际描述为准。

11.13.3.7 数据结构

常用的数据结构如下，可以通过**11.13.4 公共接口**创建。

- 定义AscendCL张量数据结构，用来管理和存储类似向量、矩阵等数据。
  typedef struct aclTensor aclTensor
- 定义AscendCL标量数据结构，用来管理和存储标量数据（比如一个单独的数）。
  typedef struct aclScalar aclScalar
- 定义AscendCL整型数组数据结构，用来管理和存储由int类型数据构成的数组。
  typedef struct aclIntArray aclIntArray
- 定义AscendCL浮点型数组数据结构，用来管理和存储由float类型数据构成的数组。
  typedef struct aclFloatArray aclFloatArray
- 定义AscendCL布尔型数组数据结构，用来管理和存储由bool类型数据构成的数组。
  typedef struct aclBoolArray aclBoolArray
- 定义AscendCL张量列表数据结构，用来管理和存储由tensor类型数据构成的列表。
  typedef struct aclTensorList aclTensorList
- 定义AscendCL执行器数据结构，用来执行算子计算的容器。
  typedef struct aclOpExecutor aclOpExecutor
- 定义AscendCL流处理数据结构，用来管理和维护一些异步操作的执行顺序。
  typedef void *aclrtStream

11.13.4 公共接口

本节罗列了单算子API执行依赖的一些AscendCL元接口，通过这些元接口可以构建所需的**11.13.3.7 数据结构**，如aclTensor、aclScalar、aclIntArray等。

表 11-5 AscendCL meta 接口

接口名	接口定义	功能说明	参数说明	返回值
aclCreateTensor	**aclTensor *aclCreateTensor(const int64_t *viewDims, uint64_t viewDimsNum, aclDataType dataType, const int64_t *stride, int64_t offset, aclFormat format, const int64_t *storageDims, uint64_t storageDimsNum, void *tensorData)**	创建aclTensor。	• viewDims（输入）：view shape的维度值。 • viewDimsNum（输入）：view shape的维度数。 • dataType（输入）：tensor的数据类型。 • stride（输入）：tensor的view的各个维度步长。 • offset（输入）：tensor首元素相对于storage的偏移。 • format（输入）：tensor的数据排布格式。 • storageDims（输入）：tensor的存储的shape的维度值。 • storageDimsNum（输入）：tensor的存储的shape的维度数。 • tensorData（输入）：tensor在Device侧的存储地址。	成功则返回创建好的aclTensor，否则返回nullptr。
aclDestroyTensor	**aclnnStatus aclDestroyTensor(const aclTensor *tensor)**	销毁aclTensor。	tensor（输入）：需要销毁的tensor指针。	返回0表示成功，返回其它值表示失败。
aclCreateScalar	**aclScalar *aclCreateScalar(void *value, aclDataType dataType)**	创建aclScalar。	• value（输入）：Host侧的scalar类型的指针，其指向的值会作为scalar。 • dataType（输入）：scalar的数据类型。	成功则返回创建好的aclScalar，否则返回nullptr。

接口名	接口定义	功能说明	参数说明	返回值
aclDestroyScalar	**aclnnStatus aclDestroyScalar(const aclScalar *scalar)**	销毁aclScalar。	scalar（输入）：需要销毁的scalar。	返回0表示成功，返回其它值表示失败。
aclCreateIntArray	**aclIntArray *aclCreateIntArray(const int64_t *value, uint64_t size)**	创建aclIntArray。	● value（输入）：Host侧的int64_t类型的指针，其指向的值会赋给Array。 ● size（输入）：整型数组的长度。	成功则返回创建好的aclIntArray，否则返回nullptr。
aclDestroyIntArray	**aclnnStatus aclDestroyIntArray(const aclIntArray *array)**	销毁aclIntArray。	array（输入）：需要销毁的aclIntArray。	返回0表示成功，返回其它值表示失败。
aclCreateFloatArray	**aclFloatArray *aclCreateFloatArray(const float *value, uint64_t size)**	创建aclFloatArray。	● value（输入）：Host侧的float类型的指针，其指向的值会赋给Array。 ● size（输入）：浮点型数组的长度。	成功则返回创建好的aclFloatArray，否则返回nullptr。
aclDestroyFloatArray	**aclnnStatus aclDestroyFloatArray(const aclFloatArray *array)**	销毁aclFloatArray。	array（输入）：需要销毁的aclFloatArray	返回0表示成功，返回其它值表示失败。
aclCreateBoolArray	**aclBoolArray *aclCreateBoolArray(const bool *value, uint64_t size)**	创建aclBoolArray。	● value（输入）：Host侧的bool类型的指针，其指向的值会赋给Array。 ● size（输入）：布尔型数组的长度。	成功则返回创建好的aclBoolArray，否则返回nullptr。
aclDestroyBoolArray	**aclnnStatus aclDestroyBoolArray(const aclBoolArray *array)**	销毁aclBoolArray。	array（输入）：需要销毁的aclBoolArray。	返回0表示成功，返回其它值表示失败。

接口名	接口定义	功能说明	参数说明	返回值
aclCreateTensorList	**aclTensorList *aclCreateTensorList(const aclTensor *const *value, uint64_t size)**	创建aclTensorList。	value（输入）：Device侧的aclTensor类型的指针，其指向的值会赋给TensorList。 size（输入）：张量列表的长度。	成功则返回创建好的aclTensorList，否则返回nullptr。
aclDestroyTensorList	**aclnnStatus aclDestroyTensorList(const aclTensorList *array)**	销毁aclTensorList。	array（输入）：需要销毁的aclTensorList。	返回0表示成功，返回其它值表示失败。
aclGetViewShape	**aclnnStatus aclGetViewShape(const aclTensor *tensor, int64_t **viewDims, uint64_t *viewDimsNum)**	获取aclTensor的ViewShape。	• tensor（输入）：输入的tensor。 • viewDims（输出）：返回的viewShape的维度值。 • viewDimsNum（输出）：返回的viewShape的维度数。	返回0表示成功，返回其它值表示失败。
aclGetViewStrides	**aclnnStatus aclGetViewStrides(const aclTensor *tensor, int64_t **stridesValue, uint64_t *stridesNum)**	获取aclTensor的viewStrides。	• tensor（输入）：输入的tensor。 • stridesValue（输出）：返回的viewStrides的值。 • stridesNum（输出）：返回的viewStrides的strides值个数。	返回0表示成功，返回其它值表示失败。
aclGetViewOffset	**aclnnStatus aclGetViewOffset(const aclTensor *tensor, int64_t *offset)**	获取aclTensor的viewOffset。	• tensor（输入）：输入的tensor。 • offset（输出）：返回的offset值。	返回0表示成功，返回其它值表示失败。
aclGetFormat	**aclnnStatus aclGetFormat(const aclTensor *tensor, aclFormat *format)**	获取aclTensor的format。	• tensor（输入）：输入的tensor。 • format（输出）：返回的aclTensor的数据格式。	返回0表示成功，返回其它值表示失败。

接口名	接口定义	功能说明	参数说明	返回值
aclGetDataType	**aclnnStatus aclGetDataType(const aclTensor *tensor, aclDataType *dataType)**	获取aclTensor的DataType。	• tensor（输入）：输入的tensor。 • dataType（输出）：返回的aclTensor的数据类型。	返回0表示成功，返回其它值表示失败。
aclGetIntArraySize	**aclnnStatus aclGetIntArraySize(const aclIntArray *array, uint64_t *size)**	获取aclIntArray的大小。	• array（输入）：输入的aclIntArray。 • size（输出）：返回的长度。	返回0表示成功，返回其它值表示失败。
aclGetFloatArraySize	**aclnnStatus aclGetFloatArraySize(const aclFloatArray *array, uint64_t *size)**	获取aclFloatArray的大小。	• array（输入）：输入的aclFloatArray。 • size（输出）：返回的长度。	返回0表示成功，返回其它值表示失败。
aclGetBoolArraySize	**aclnnStatus aclGetBoolArraySize(const aclBoolArray *array, uint64_t *size)**	获取aclBoolArray的大小。	• array（输入）：输入的aclBoolArray。 • size（输出）：返回的长度。	返回0表示成功，返回其它值表示失败。
aclGetTensorListSize	**aclnnStatus aclGetTensorListSize(const aclTensorList *tensorList, uint64_t *size)**	获取aclTensorList的大小。	• tensorList（输入）：输入的aclTensorList。 • size（输出）：返回的长度。	返回0表示成功，返回其它值表示失败。
aclGetStorageShape	**aclnnStatus aclGetStorageShape(const aclTensor *tensor, int64_t **storageDims, uint64_t *storageDimsNum)**	或取Tensor在内存上的存储shape。	• tensor（输入）：输入的tensor对应的地址。 • storageDims（输出）：表示存储shape的维度值。 • storageDimsNum（输出）：表示存储shape的维度数。	返回0表示成功，返回其它值表示失败。

接口名	接口定义	功能说明	参数说明	返回值
aclCreateScalarList	**aclScalarList *aclCreateScalarList(const aclScalar *const *value, uint64_t size)**	创建aclScalarList。	value（输入）：Device侧的aclScalar类型的指针，其指向的值会赋给ScalarList。 size（输入）：标量列表的长度。	成功则返回创建好的aclScalarList，否则返回nullptr。
aclDestroyScalarList	**aclnnStatus aclDestroyScalarList(const aclScalarList *array)**	销毁aclScalarList。	array（输入）：需要销毁的aclScalarList。	返回0表示成功，返回其它值表示失败。
aclGetScalarListSize	**aclnnStatus aclGetScalarListSize(const aclScalarList *scalarList, uint64_t *size)**	获取aclScalarList的大小。	● scalarList（输入）：输入的aclScalarList。 ● size（输出）：返回的长度。	返回0表示成功，返回其它值表示失败。

接口名	接口定义	功能说明	参数说明	返回值
aclInitTensor	**aclnnStatus aclInitTensor(aclTensor *tensor, const int64_t *viewDims, uint64_t viewDimsNum, aclDataType dataType, const int64_t *stride, int64_t offset, aclFormat format, const int64_t *storageDims, uint64_t storageDimsNum, void *tensorDataAddr)**	初始化给定tensor的参数。 当用户想复用已创建aclTensor时，可使用该接口先重置aclTensor的各项属性。	• tensor（输入）：待初始化参数的tensor，通过aclCreateTensor创建。 • viewDims（输入）：view shape的维度值。 • viewDimsNum（输入）：view shape的维度数。 • dataType（输入）：tensor的数据类型。 • stride（输入）：tensor的view的各个维度步长。 • offset（输入）：tensor首元素相对于storage的偏移。 • format（输入）：tensor的数据排布格式。 • storageDims（输入）：tensor的存储shape的维度值。 • storageDimsNum（输入）：tensor的存储shape的维度数。 • tensorDataAddr（输入）：tensor在Device侧的存储地址。	返回0表示成功，返回其它值表示失败。

11.13.5 NN 类算子接口

请通过附件获取NN类算子API参考（aclnn）压缩包，解压该文件后，查看readme.md，了解各算子的使用说明。

11.13.6 DVPP 类算子接口

11.13.6.1 acldvppNormalize

Atlas 200/300/500 推理产品不支持该算子。

Atlas 推理系列产品不支持该算子。

接口原型

每个算子有两段接口，必须先调用“**acldvppNormalizeGetWorkspaceSize**”接口获取入参并根据计算流程计算所需workspace大小，再调用“**acldvppNormalize**”接口执行计算。两段式接口如下：

- **acldvppStatus acldvppNormalizeGetWorkspaceSize(const aclTensor *self, const aclFloatArray* mean, const aclFloatArray* std, aclTensor *out, uint64_t *workspaceSize, aclOpExecutor **executor)**
- **acldvppStatus acldvppNormalize(void *workspace, uint64_t workspaceSize, aclOpExecutor *executor, aclrtStream stream)**

功能描述

- 算子功能：图像归一化，减均值、除标准差，由用户指定均值、标准差。
- 计算公式：

  out = (self - mean) / std

acldvppNormalizeGetWorkspaceSize

- **接口定义：**

  acldvppStatus acldvppNormalizeGetWorkspaceSize(const aclTensor *self, const aclFloatArray* mean, const aclFloatArray* std, aclTensor *out, uint64_t *workspaceSize, aclOpExecutor **executor)
- **参数说明：**
  - self：表示算子输入Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输入Tensor的dataType支持UINT8/FLOAT、Format支持NCHW/NHWC、不支持**非连续的Tensor**，同时N支持1或空、C支持1或3（1表示输入数据是GRAY格式的图，3表示输入数据是按R、G、B顺序排布的图）。
  - mean：表示均值数据，需调用**aclCreateFloatArray**接口创建aclFloatArray类型的数据（数据存放在Host侧，dataType为FLOAT），该接口的入参size需与C轴大小保持一致。
  - std：表示标准差数据，需调用**aclCreateFloatArray**接口创建aclFloatArray类型的数据（数据存放在Host侧，dataType为FLOAT），该接口的入参size需与C轴大小保持一致。
  - out：表示算子输出Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输出Tensor的dataType支持FLOAT、不支持**非连续的Tensor**，Format、Shape需要和self一致。
  - workspaceSize：返回用户需要在Device侧申请的workspace大小。
  - executor：返回op执行器，包含了算子计算流程。
- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

acldvppNormalize

- **接口定义：**

acldvppStatus acldvppNormalize(void *workspace, uint64_t workspaceSize, aclOpExecutor *executor, aclrtStream stream)

- **参数说明：**
  - workspace：需调用**aclrtMalloc**接口申请Device内存，内存大小为workspaceSize，**aclrtMalloc**接口输出的内存地址在此处传入。
  - workspaceSize：与acldvppNormalizeGetWorkspaceSize接口获取的workspaceSize保持一致。
  - executor：op执行器，包含了算子计算流程，与acldvppNormalizeGetWorkspaceSize接口的executor保持一致。
  - stream：指定执行任务的Stream，可复用已创建的Stream节省资源或调用**aclrtCreateStream**接口创建Stream，再作为入参在此处传入。
- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

约束与限制

支持图像分辨率范围在[6*4~4096*8192]。

11.13.6.2 acldvppDecodeJpeg

Atlas 200/300/500 推理产品不支持该算子。

Atlas 推理系列产品不支持该算子。

接口原型

每个算子有两段接口，必须先调用“**acldvppDecodeJpegGetWorkspaceSize**”接口获取入参并根据计算流程计算所需workspace大小，再调用“**acldvppDecodeJpeg**”接口执行计算。两段式接口如下：

- **第一段接口：acldvppStatus acldvppDecodeJpegGetWorkspaceSize(const aclTensor *self, uint32_t channels, bool tryRecoverTruncated, aclTensor *out, uint64_t *workspaceSize, aclOpExecutor **executor)**
- **第二段接口：acldvppStatus acldvppDecodeJpeg(void *workspace, uint64_t workspaceSize, aclOpExecutor *executor, aclrtStream stream)**

功能描述

算子功能：将.jpg、.jpeg、.JPG、.JPEG图片文件解码为单通道（GRAY）或三通道（RGB）图像。

acldvppDecodeJpegGetWorkspaceSize

- **接口定义：**

  acldvppStatus acldvppDecodeJpegGetWorkspaceSize(const aclTensor *self, uint32_t channels, bool tryRecoverTruncated, aclTensor *out, uint64_t *workspaceSize, aclOpExecutor **executor)
- **参数说明：**
  - self：表示算子输入Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输入Tensor的dataType支持UINT8、Format支持ND、不支持**非连续的Tensor**。

  - channels：指定输出通道数量，1或者3，配置1输出GRAY图，配置3输出RGB图。
  - tryRecoverTruncated：是否尝试解码损坏码流。该参数为预留参数，暂不支持。当前用户需配置为true。
  - out：表示算子输出Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输出Tensor的dataType支持UINT8、Format支持NCHW/NHWC、不支持**非连续的Tensor**。
  - workspaceSize：返回用户需要在Device侧申请的workspace大小。
  - executor：返回op执行器，包含了算子计算流程。
- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

acldvppDecodeJpeg

- **接口定义：**

  acldvppStatus acldvppDecodeJpeg(void *workspace, uint64_t workspaceSize, aclOpExecutor *executor, aclrtStream stream)
- **参数说明：**
  - workspace：需调用**aclrtMalloc**接口申请Device内存，内存大小为workspaceSize，**aclrtMalloc**接口输出的内存地址在此处传入。
  - workspaceSize：与acldvppDecodeJpegGetWorkspaceSize接口获取的workspaceSize保持一致。
  - executor：op执行器，包含了算子计算流程，与acldvppDecodeJpegGetWorkspaceSize接口的executor保持一致。
  - stream：指定执行任务的Stream，可复用已创建的Stream节省资源或调用**aclrtCreateStream**接口创建Stream，再作为入参在此处传入。
- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

约束与限制

支持分辨率范围在[6*4, 32768*32768]内的JPEG图像解码，分辨率在[32*32, 16384*16384]范围内的图片会走专门的硬件加速解码，其它场景（包含硬件解码失败的场景）使用开源软件libjpeg-turbo在AI CPU上解码。

在硬件解码场景下，本算子的解码结果与OpenCV 3.4.2版本保持一致，OpenCV 3.4.2版本中默认使用libjpeg-turbo 1.5.3版本。

11.13.6.3 acldvppResize

Atlas 200/300/500 推理产品不支持该算子。

Atlas 推理系列产品不支持该算子。

接口原型

每个算子有两段接口，必须先调用“**acldvppResizeGetWorkspaceSize**”接口获取入参并根据计算流程计算所需workspace大小，再调用“**acldvppResize**”接口执行计算。两段式接口如下：

- **acldvppStatus acldvppResizeGetWorkspaceSize(const aclTensor* self, uint32_t interpolationMode, aclTensor* out, uint64_t* workspaceSize, aclOpExecutor** executor)**
- **acldvppStatus acldvppResize(void* workspace, uint64_t workspaceSize, aclOpExecutor* executor, aclrtStream stream)**

函数功能

算子功能：图像缩放，支持bilinear、nearest、bicubic插值算法。

acldvppResizeGetWorkspaceSize

- **接口定义：**

  acldvppStatus acldvppResizeGetWorkspaceSize(const aclTensor* self, uint32_t interpolationMode, aclTensor* out, uint64_t* workspaceSize, aclOpExecutor** executor)

- **参数说明：**
  - self：表示算子输入Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输入Tensor的dataType支持UINT8/FLOAT、Format支持NCHW/NHWC、不支持**非连续的Tensor**，同时N支持1或空、C支持1或3（1表示输入GRAY图，3表示输入RGB图）。
  - interpolationMode：表示缩放算法，该参数的取值范围如下：
    - 0：bilinear算法
    - 1：nearest算法
    - 2：bicubic算法
  - out：表示算子输出Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输出Tensor的dataType支持UINT8/FLOAT、不支持**非连续的Tensor**，dataType、Format需要和self一致，Shape中的N轴、C轴大小需要和self一致，Shape中的W轴、H轴大小需和缩放后的宽、高保持一致。
  - workspaceSize：返回用户需要在Device侧申请的workspace大小。
  - executor：返回op执行器，包含了算子计算流程。
- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

acldvppResize

- **接口定义：**

  acldvppStatus acldvppResize(void* workspace, uint64_t workspaceSize, aclOpExecutor* executor, aclrtStream stream)

- **参数说明：**
  - workspace：需调用**aclrtMalloc**接口申请Device内存，内存大小为workspaceSize，**aclrtMalloc**接口输出的内存地址在此处传入。
  - workspaceSize：与acldvppResizeGetWorkspaceSize接口获取的workspaceSize保持一致。
  - executor：op执行器，包含了算子计算流程，与acldvppResizeGetWorkspaceSize接口的executor保持一致。

  - stream：指定执行任务的Stream，可复用已创建的Stream节省资源或调用**aclrtCreateStream**接口创建Stream，再作为入参在此处传入。
- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

约束与限制

1. 支持图像分辨率范围在[6*4~32768*32768]。
2. 当输出out的shape的宽度超过4096时，输入self的shape的宽*高需要大于128*16。

11.13.6.4 acldvppImgToTensor

Atlas 200/300/500 推理产品不支持该算子。

Atlas 推理系列产品不支持该算子。

接口原型

每个算子有两段接口，必须先调用“**acldvppImgToTensorGetWorkspaceSize**”接口获取入参并根据计算流程计算所需workspace大小，再调用“**acldvppImgToTensor**”接口执行计算。两段式接口如下：

- **acldvppStatus acldvppImgToTensorGetWorkspaceSize(const aclTensor *self, aclTensor *out, uint64_t *workspaceSize, aclOpExecutor **executor)**
- **acldvppStatus acldvppImgToTensor(void *workspace, uint64_t workspaceSize, aclOpExecutor *executor, aclrtStream stream)**

函数功能

- 算子功能：图像归一化，减均值、除标准差，但均值固定为0，标准差固定为255，仅用于将[0, 255]的UINT8图像归一化到[0.0, 1.0]的FLOAT图像。
- 计算公式：

  out = (self - mean) / std

acldvppImgToTensorGetWorkspaceSize

- **接口定义：**

  acldvppStatus acldvppImgToTensorGetWorkspaceSize(const aclTensor *self, aclTensor *out, uint64_t *workspaceSize, aclOpExecutor **executor)
- **参数说明：**
  - self：表示算子输入Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输入Tensor的dataType支持UINT8、Format支持NCHW/NHWC、不支持**非连续的Tensor**，同时N支持1或空、C支持1或3（1表示输入GRAY图，3表示输入RGB图）。
  - out：表示算子输出Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输出Tensor的dataType支持FLOAT、不支持**非连续的Tensor**，Format、Shape需要和self一致。
  - workspaceSize：返回用户需要在Device侧申请的workspace大小。
  - executor：返回op执行器，包含了算子计算流程。

- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

acldvppImgToTensor

- **接口定义：**

  acldvppStatus acldvppImgToTensor(void *workspace, uint64_t workspaceSize, aclOpExecutor *executor, aclrtStream stream)

- **参数说明：**
  - workspace：需调用**aclrtMalloc**接口申请Device内存，内存大小为workspaceSize，**aclrtMalloc**接口输出的内存地址在此处传入。
  - workspaceSize：与acldvppImgToTensorGetWorkspaceSize接口获取的workspaceSize保持一致。
  - executor：op执行器，包含了算子计算流程，与acldvppImgToTensorGetWorkspaceSize接口的executor保持一致。
  - stream：指定执行任务的Stream，可复用已创建的Stream节省资源或调用**aclrtCreateStream**接口创建Stream，再作为入参在此处传入。

- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

约束与限制

支持图像分辨率范围在[6*4~4096*8192]。

11.13.6.5 acldvppAdjustBrightness

Atlas 200/300/500 推理产品不支持该算子。

Atlas 推理系列产品不支持该算子。

接口原型

每个算子有两段接口，必须先调用“**acldvppAdjustBrightnessGetWorkspaceSize**”接口获取入参并根据计算流程计算所需workspace大小，再调用“**acldvppAdjustBrightness**”接口执行计算。两段式接口如下：

- **acldvppStatus acldvppAdjustBrightnessGetWorkspaceSize(const aclTensor *self, float factor, aclTensor *out, uint64_t *workspaceSize, aclOpExecutor **executor)**
- **acldvppStatus acldvppAdjustBrightness(void *workspace, uint64_t workspaceSize, aclOpExecutor *executor, aclrtStream stream)**

函数功能

- 算子功能：调节图像亮度。
- 计算公式：

  out = self * factor

acldvppAdjustBrightnessGetWorkspaceSize

- **接口定义：**

  aclnnStatus acldvppAdjustBrightnessGetWorkspaceSize(const aclTensor *self, float factor, aclTensor *out, uint64_t *workspaceSize, aclOpExecutor **executor)

- **参数说明：**
  - self：表示算子输入Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输入Tensor的dataType支持UINT8/FLOAT、Format支持NCHW/NHWC、不支持**非连续的Tensor**，同时N支持1或空、C支持1或3（1表示输入GRAY图，3表示输入RGB图）。
  - factor：图像亮度调节参数，非负数。
  - out：表示算子输出Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输出Tensor的dataType支持UINT8/FLOAT、Format支持NCHW/NHWC、不支持**非连续的Tensor**，同时N只支持1、C支持1或3（1表示GRAY图，3表示RGB图），dataType、Format、Shape需要和self一致。
  - workspaceSize：返回用户需要在Device侧申请的workspace大小。
  - executor：返回op执行器，包含了算子计算流程。

- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

acldvppAdjustBrightness

- **接口定义：**

  acldvppStatus acldvppAdjustBrightness(void *workspace, uint64_t workspaceSize, aclOpExecutor *executor, aclrtStream stream)

- **参数说明：**
  - workspace：需调用**aclrtMalloc**接口申请Device内存，内存大小为workspaceSize，**aclrtMalloc**接口输出的内存地址在此处传入。
  - workspaceSize：与acldvppAdjustBrightnessGetWorkspaceSize接口获取的workspaceSize保持一致。
  - executor：op执行器，包含了算子计算流程，与acldvppAdjustBrightnessGetWorkspaceSize接口的executor保持一致。
  - stream：指定执行任务的Stream，可复用已创建的Stream节省资源或调用**aclrtCreateStream**接口创建Stream，再作为入参在此处传入。

- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

约束与限制

支持图像分辨率范围在[6*4~4096*8192]。

11.13.6.6 acldvppAdjustSaturation

Atlas 200/300/500 推理产品不支持该算子。

Atlas 推理系列产品不支持该算子。

接口原型

每个算子有两段接口，必须先调用“**acldvppAdjustSaturationGetWorkspaceSize**”接口获取入参并根据计算流程计算所需workspace大小，再调用“**acldvppAdjustSaturation**”接口执行计算。两段式接口如下：

- **acldvppStatus acldvppAdjustSaturationGetWorkspaceSize(const aclTensor *self, float factor, aclTensor *out, uint64_t *workspaceSize, aclOpExecutor **executor)**
- **acldvppStatus acldvppAdjustSaturation(void *workspace, uint64_t workspaceSize, aclOpExecutor *executor, aclrtStream stream)**

功能描述

- 算子功能：调节图像的饱和度。
- 计算公式：

  out = self * factor + (self->Y8) * (1-factor)

acldvppAdjustSaturationGetWorkspaceSize

- **接口定义：**

  acldvppStatus acldvppAdjustSaturationGetWorkspaceSize(const aclTensor *self, float factor, aclTensor *out, uint64_t *workspaceSize, aclOpExecutor **executor)
- **参数说明：**
  - self：表示算子输入Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输入Tensor的dataType支持UINT8/FLOAT、Format支持NCHW/NHWC、不支持**非连续的Tensor**，同时N支持1或空、C支持1或3（1表示输入数据是GRAY格式的图，3表示输入数据是按R、G、B顺序排布的图）。
  - factor：表示调节饱和度系数。dataType支持FLOAT。支持factor为大于0的值。
  - out：表示算子输出Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输出Tensor的dataType支持UINT8/FLOAT、不支持**非连续的Tensor**，dataType、Format、Shape需要和self一致。
  - workspaceSize：返回用户需要在Device侧申请的workspace大小。
  - executor：返回op执行器，包含了算子计算流程。
- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

acldvppAdjustSaturation

- **接口定义：**

  acldvppStatus acldvppAdjustSaturation(void *workspace, uint64_t workspaceSize, aclOpExecutor *executor, aclrtStream stream)
- **参数说明：**
  - workspace：需调用**aclrtMalloc**接口申请Device内存，内存大小为workspaceSize，**aclrtMalloc**接口输出的内存地址在此处传入。

  - workspaceSize：与acldvppAdjustSaturationGetWorkspaceSize接口获取的workspaceSize保持一致。
  - executor：op执行器，包含了算子计算流程，与acldvppAdjustSaturationGetWorkspaceSize接口的executor保持一致。
  - stream：指定执行任务的Stream，可复用已创建的Stream节省资源或调用**aclrtCreateStream**接口创建Stream，再作为入参在此处传入。
- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

约束与限制

支持图像分辨率范围在[6*4~4096*8192]。

当factor参数值为小数、且self和out数据的dataType为UINT8时，与第三方软件Torchvision相比，单个像素值最大差异为正负1。

11.13.6.7 acldvppAdjustHue

Atlas 200/300/500 推理产品不支持该算子。

Atlas 推理系列产品不支持该算子。

接口原型

每个算子有两段接口，必须先调用“**acldvppAdjustHueGetWorkspaceSize**”接口获取入参并根据计算流程计算所需workspace大小，再调用“**acldvppAdjustHue**”接口执行计算。两段式接口如下：

- **acldvppStatus acldvppAdjustHueGetWorkspaceSize(const aclTensor* self, float factor, aclTensor* out, uint64_t* workspaceSize, aclOpExecutor** executor)**
- **acldvppStatus acldvppAdjustHue(void* workspace, uint64_t workspaceSize, aclOpExecutor* executor, aclrtStream stream)**

功能描述

- 算子功能：调节图像色度。
- 计算公式：

  out = self + factor

  RGB图像转HSV（Hue, Saturation, Value），H通道+factor，图像再转成RGB。

acldvppAdjustHueGetWorkspaceSize

- **接口定义：**

  acldvppStatus acldvppAdjustHueGetWorkspaceSize(const aclTensor* self, float factor, aclTensor* out, uint64_t* workspaceSize, aclOpExecutor** executor)
- **参数说明：**
- self：表示算子输入Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输入Tensor的dataType支持UINT8/FLOAT、Format支持NCHW/NHWC、不支持**非连续的Tensor**，同时N支持1或空、C支持1或3（1表示输入数据是GRAY格式的图，3表示输入数据是按R、G、B顺序排布的图）。

- factor：表示调节色度系数。dataType支持FLOAT。支持factor值范围为[-0.5, 0.5]。
- out：表示算子输出Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输出Tensor的dataType支持UINT8/FLOAT、不支持**非连续的Tensor**，dataType、Format、Shape需要和self一致。
- workspaceSize：返回用户需要在Device侧申请的workspace大小。
- executor：返回op执行器，包含了算子计算流程。
- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

acldvppAdjustHue

- **接口定义：**

  acldvppStatus acldvppAdjustHue(void* workspace, uint64_t workspaceSize, aclOpExecutor* executor, aclrtStream stream)
- **参数说明：**
  - workspace：需调用**aclrtMalloc**接口申请Device内存，内存大小为workspaceSize，**aclrtMalloc**接口输出的内存地址在此处传入。
  - workspaceSize：与acldvppAdjustHueGetWorkspaceSize接口获取的workspaceSize保持一致。。
  - executor：op执行器，包含了算子计算流程，与acldvppAdjustHueGetWorkspaceSize接口的executor保持一致。。
  - stream：指定执行任务的Stream，可复用已创建的Stream节省资源或调用**aclrtCreateStream**接口创建Stream，再作为入参在此处传入。
- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

约束与限制

支持图像分辨率范围在[6*4~4096*8192]

11.13.6.8 acldvppAdjustContrast

Atlas 200/300/500 推理产品不支持该算子。

Atlas 推理系列产品不支持该算子。

接口原型

每个算子有两段接口，必须先调用“**acldvppAdjustContrastGetWorkspaceSize**”接口获取入参并根据计算流程计算所需workspace大小，再调用“**acldvppAdjustContrast**”接口执行计算。两段式接口如下：

- **acldvppStatus acldvppAdjustContrastGetWorkspaceSize(const aclTensor *self, float factor, aclTensor *out, uint64_t *workspaceSize, aclOpExecutor **executor)**
- **acldvppStatus acldvppAdjustContrast(void *workspace, uint64_t workspaceSize, aclOpExecutor *executor, aclrtStream stream)**

函数功能

- 算子功能：调节图像对比度。
- 计算公式：

  out = (self - mean) * factor + mean，其中mean为图片灰度图的均值。

acldvppAdjustContrastGetWorkspaceSize

- **接口定义：**

  aclnnStatus acldvppAdjustContrastGetWorkspaceSize(const aclTensor *self, float factor, aclTensor *out, uint64_t *workspaceSize, aclOpExecutor **executor)

- **参数说明：**
  - self：表示算子输入Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输入Tensor的dataType支持UINT8/FLOAT、Format支持NCHW/NHWC、不支持**非连续的Tensor**，同时N支持1或空、C支持1或3（1表示输入GRAY图，3表示输入RGB图）。
  - factor：图像对比度调节参数，非负数。
  - out：表示算子输出Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输出Tensor的dataType支持UINT8/FLOAT、Format支持NCHW/NHWC、不支持**非连续的Tensor**，同时N只支持1、C支持1或3（1表示GRAY图，3表示RGB图），dataType、Format、Shape需要和self一致。
  - workspaceSize：返回用户需要在Device侧申请的workspace大小。
  - executor：返回op执行器，包含了算子计算流程。
- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

acldvppAdjustContrast

- **接口定义：**

  acldvppStatus acldvppAdjustContrast(void *workspace, uint64_t workspaceSize, aclOpExecutor *executor, aclrtStream stream)

- **参数说明：**
  - workspace：需调用**aclrtMalloc**接口申请Device内存，内存大小为workspaceSize，**aclrtMalloc**接口输出的内存地址在此处传入。
  - workspaceSize：与acldvppAdjustContrastGetWorkspaceSize接口获取的workspaceSize保持一致。
  - executor：op执行器，包含了算子计算流程，与acldvppAdjustContrastGetWorkspaceSize接口的executor保持一致。
  - stream：指定执行任务的Stream，可复用已创建的Stream节省资源或调用**aclrtCreateStream**接口创建Stream，再作为入参在此处传入。
- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

约束与限制

支持图像分辨率范围在[6*4~4096*8192]。

11.13.6.9 acldvppHorizontalFlip

Atlas 200/300/500 推理产品不支持该算子。

Atlas 推理系列产品不支持该算子。

接口原型

每个算子有两段接口，必须先调用“**acldvppHorizontalFlipGetWorkspaceSize**”接口获取入参并根据计算流程计算所需workspace大小，再调用“**acldvppHorizontalFlip**”接口执行计算。两段式接口如下：

- **acldvppStatus acldvppHorizontalFlipGetWorkspaceSize(const aclTensor* self, aclTensor* out, uint64_t* workspaceSize, aclOpExecutor** executor)**
- **acldvppStatus acldvppHorizontalFlip(void* workspace, uint64_t workspaceSize, aclOpExecutor* executor, aclrtStream stream)**

函数功能

- 算子功能：对图像做水平翻转。

acldvppHorizontalFlipGetWorkspaceSize

- **接口定义：**

  aclnnStatus acldvppHorizontalFlipGetWorkspaceSize(const aclTensor* self, aclTensor* out, uint64_t* workspaceSize, aclOpExecutor** executor)

- **参数说明：**
  - self：表示算子输入Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输入Tensor的dataType支持UINT8/FLOAT、Format支持NCHW/NHWC、不支持**非连续的Tensor**，同时N支持1或空、C支持1或3（1表示输入GRAY图，3表示输入RGB图）。
  - out：表示算子输出Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输出Tensor的dataType支持UINT8/FLOAT、Format支持NCHW/NHWC、不支持**非连续的Tensor**，同时N只支持1、C支持1或3（1表示GRAY图，3表示RGB图），dataType、Format、Shape需要和self一致。
  - workspaceSize：返回用户需要在Device侧申请的workspace大小。
  - executor：返回op执行器，包含了算子计算流程。

- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

acldvppHorizontalFlip

- **接口定义：**

  acldvppStatus acldvppHorizontalFlip(void* workspace, uint64_t workspaceSize, aclOpExecutor* executor, aclrtStream stream)

- **参数说明：**
  - workspace：需调用**aclrtMalloc**接口申请Device内存，内存大小为workspaceSize，**aclrtMalloc**接口输出的内存地址在此处传入。
  - workspaceSize：与acldvppHorizontalFlipGetWorkspaceSize接口获取的workspaceSize保持一致。

  - executor：op执行器，包含了算子计算流程，与acldvppHorizontalFlipGetWorkspaceSize接口的executor保持一致。
  - stream：指定执行任务的Stream，可复用已创建的Stream节省资源或调用**aclrtCreateStream**接口创建Stream，再作为入参在此处传入。
- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

约束与限制

支持图像分辨率范围在[6*4~4096*8192]。

11.13.6.10 acldvppVerticalFlip

Atlas 200/300/500 推理产品不支持该算子。

Atlas 推理系列产品不支持该算子。

接口原型

每个算子有两段接口，必须先调用“**acldvppVerticalFlipGetWorkspaceSize**”接口获取入参并根据计算流程计算所需workspace大小，再调用“**acldvppVerticalFlip**”接口执行计算。两段式接口如下：

- **acldvppStatus acldvppVerticalFlipGetWorkspaceSize(const aclTensor* self, aclTensor* out, uint64_t* workspaceSize, aclOpExecutor** executor)**
- **acldvppStatus acldvppVerticalFlip(void* workspace, uint64_t workspaceSize, aclOpExecutor* executor, aclrtStream stream)**

函数功能

- 算子功能：对图像做垂直翻转。

acldvppVerticalFlipGetWorkspaceSize

- **接口定义：**

  aclnnStatus acldvppVerticalFlipGetWorkspaceSize(const aclTensor* self, aclTensor* out, uint64_t* workspaceSize, aclOpExecutor** executor)
- **参数说明：**
  - self：表示算子输入Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输入Tensor的dataType支持UINT8/FLOAT、Format支持NCHW/NHWC、不支持**非连续的Tensor**，同时N支持1或空、C支持1或3（1表示输入GRAY图，3表示输入RGB图）。
  - out：表示算子输出Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输出Tensor的dataType支持UINT8/FLOAT、Format支持NCHW/NHWC、不支持**非连续的Tensor**，同时N只支持1、C支持1或3（1表示GRAY图，3表示RGB图），dataType、Format、Shape需要和self一致。
  - workspaceSize：返回用户需要在Device侧申请的workspace大小。
  - executor：返回op执行器，包含了算子计算流程。
- **返回值：**

返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

acldvppVerticalFlip

- **接口定义：**

  acldvppStatus acldvppVerticalFlip(void* workspace, uint64_t workspaceSize, aclOpExecutor* executor, aclrtStream stream)

- **参数说明：**
  - workspace：需调用**aclrtMalloc**接口申请Device内存，内存大小为workspaceSize，**aclrtMalloc**接口输出的内存地址在此处传入。
  - workspaceSize：与acldvppVerticalFlipGetWorkspaceSize接口获取的workspaceSize保持一致。
  - executor：op执行器，包含了算子计算流程，与acldvppVerticalFlipGetWorkspaceSize接口的executor保持一致。
  - stream：指定执行任务的Stream，可复用已创建的Stream节省资源或调用**aclrtCreateStream**接口创建Stream，再作为入参在此处传入。

- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

约束与限制

支持图像分辨率范围在[6*4~4096*8192]。

11.13.6.11 acldvppGaussianBlur

Atlas 200/300/500 推理产品不支持该算子。

Atlas 推理系列产品不支持该算子。

接口原型

每个算子有两段接口，必须先调用“**acldvppGaussianBlurGetWorkspaceSize**”接口获取入参并根据计算流程计算所需workspace大小，再调用“**acldvppGaussianBlur**”接口执行计算。两段式接口如下：

- **acldvppStatus acldvppGaussianBlurGetWorkspaceSize(const aclTensor* self, const aclIntArray* kernelSize, const aclFloatArray* sigma, uint32_t paddingMode, aclTensor* out, uint64_t* workspaceSize, aclOpExecutor** executor)**
- **acldvppStatus acldvppGaussianBlur(void* workspace, uint64_t workspaceSize, aclOpExecutor* executor, aclrtStream stream)**

函数功能

- 算子功能：对图像做高斯模糊。

acldvppGaussianBlurGetWorkspaceSize

- **接口定义：**

  aclnnStatus acldvppGaussianBlurGetWorkspaceSize(const aclTensor* self, const aclIntArray* kernelSize, const aclFloatArray* sigma, uint32_t

paddingMode, aclTensor* out, uint64_t* workspaceSize, aclOpExecutor** executor)

- **参数说明：**
  - self：表示算子输入Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输入Tensor的dataType支持UINT8/FLOAT、Format支持NCHW/NHWC、不支持**非连续的Tensor**，同时N支持1或空、C支持1或3（1表示输入GRAY图，3表示输入RGB图）。
  - kernelSize: 高斯核的尺寸，该参数是一个长度为2的数组，数组内的第一个元素是宽、第二个元素是高，宽和高只能设置为1，3或者5。
  - sigma：高斯核的标准差，该值必须是正数。该参数是一个长度为2的数组，数组内的第一个元素是x方向的标准差、第二个元素是y方向的标准差。
  - paddingMode：填充模式，该参数取值如下：
    - 0：CONSTANT，填充固定值
    - 2：REFLECT，边界元素的镜像，镜像不包括边界元素。举例，其中*表示任意图像元素：cb|abc****fgh|gf
  - out：表示算子输出Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输出Tensor的dataType支持UINT8/FLOAT、Format支持NCHW/NHWC、不支持**非连续的Tensor**，同时N只支持1、C支持1和3（1表示GRAY图，3表示RGB图），dataType、Format、Shape需要和self一致。
  - workspaceSize：返回用户需要在Device侧申请的workspace大小。
  - executor：返回op执行器，包含了算子计算流程。
- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

acldvppGaussianBlur

- **接口定义：**

  acldvppStatus acldvppGaussianBlur(void* workspace, uint64_t workspaceSize, aclOpExecutor* executor, aclrtStream stream)
- **参数说明：**
  - workspace：需调用**aclrtMalloc**接口申请Device内存，内存大小为workspaceSize，**aclrtMalloc**接口输出的内存地址在此处传入。
  - workspaceSize：与acldvppGaussianBlurGetWorkspaceSize接口获取的workspaceSize保持一致。
  - executor：op执行器，包含了算子计算流程，与acldvppGaussianBlurGetWorkspaceSize接口的executor保持一致。
  - stream：指定执行任务的Stream，可复用已创建的Stream节省资源或调用**aclrtCreateStream**接口创建Stream，再作为入参在此处传入。
- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

约束与限制

支持图像分辨率范围在[6*4~4096*8192]。

11.13.6.12 acldvppWarpAffine

Atlas 200/300/500 推理产品不支持该算子。

Atlas 推理系列产品不支持该算子。

接口原型

每个算子有两段接口，必须先调用“**acldvppWarpAffineGetWorkspaceSize**”接口获取入参并根据计算流程计算所需workspace大小，再调用“**acldvppWarpAffine**”接口执行计算。两段式接口如下：

- **acldvppStatus acldvppWarpAffineGetWorkspaceSize(const aclTensor *self, const aclFloatArray* matrix, uint32_t interpolationMode, uint32_t paddingMode, const aclFloatArray* fill, aclTensor *out, uint64_t *workspaceSize, aclOpExecutor **executor)**
- **acldvppStatus acldvppWarpAffine(void *workspace, uint64_t workspaceSize, aclOpExecutor *executor, aclrtStream stream)**

函数功能

- 算子功能：对图像做仿射变换。

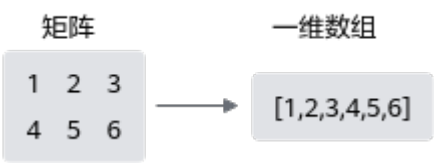
acldvppWarpAffineWorkspaceSize

- **接口定义：**

  acldvppStatus acldvppWarpAffineGetWorkspaceSize(const aclTensor *self, const aclFloatArray* matrix, uint32_t interpolationMode, uint32_t paddingMode,const aclFloatArray* fill, aclTensor *out, uint64_t *workspaceSize, aclOpExecutor **executor)

- **参数说明：**
  - self：表示算子输入Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输入Tensor的dataType支持UINT8/FLOAT、Format支持NCHW/NHWC、不支持**非连续的Tensor**，同时N支持1或空、C支持1或3（1表示输入GRAY图，3表示输入RGB图）。
  - matrix：长度为6的一维数组，数组中依次存放2*3的仿射变换矩阵数据。

    数据存放示例如下：

  - interpolationMode：表示缩放算法，该参数的取值范围如下：
    - 0：bilinear算法
    - 1：nearest算法
  - paddingMode：填充模式，该参数取值如下：
    - 0：CONSTANT，填充固定值

    - 1：EDGE，重复最后一个元素。举例，其中*表示任意图像元素: aaaaaa|a*****h|hhhhhhh
  - fill: fill是一个长度为3的数组，用于设置每个通道上填充的值，需调用**aclCreateFloatArray**接口创建参数aclFloatArray类型的数据，仅在paddingMode为CONSTANT时fill参数有效。如果C为1（表示GRAY图），填充fill[0]；如果C为3（表示RGB图），按照R、G、B顺序依次填写fill[0]、fill[1]、fill[2]。
  - out：表示算子输出Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输出Tensor的dataType支持UINT8/FLOAT、Format支持NCHW/NHWC、不支持**非连续的Tensor**，同时N只支持1、C支持1和3（1表示GRAY图，3表示RGB图），dataType、Format、Shape需要和self一致。
  - workspaceSize：返回用户需要在Device侧申请的workspace大小。
  - executor：返回op执行器，包含了算子计算流程。
- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

acldvppWarpAffine

- **接口定义：**

  acldvppStatus acldvppWarpAffine(void* workspace, uint64_t workspaceSize, aclOpExecutor* executor, aclrtStream stream);
- **参数说明：**
  - workspace：需调用**aclrtMalloc**接口申请Device内存，内存大小为workspaceSize，**aclrtMalloc**接口输出的内存地址在此处传入。
  - workspaceSize：与acldvppWarpAffineGetWorkspaceSize接口获取的workspaceSize保持一致。
  - executor：op执行器，包含了算子计算流程，与acldvppWarpAffineGetWorkspaceSize接口的executor保持一致。
  - stream：指定执行任务的Stream，可复用已创建的Stream节省资源或调用**aclrtCreateStream**接口创建Stream，再作为入参在此处传入。
- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

约束与限制

- 支持图像分辨率范围在[6*4~32768*32768]。
- 变换矩阵大小是2*3。
- 输入输出格式宽高保持一致。

11.13.6.13 acldvppWarpPerspective

Atlas 200/300/500 推理产品不支持该算子。

Atlas 推理系列产品不支持该算子。

接口原型

每个算子有两段接口，必须先调用“**acldvppWarpPerspectiveGetWorkspaceSize**”接口获取入参并根据计算流程计算所需workspace大小，再调用“**acldvppWarpPerspective**”接口执行计算。两段式接口如下：

- **acldvppStatus acldvppWarpPerspectiveGetWorkspaceSize(const aclTensor *self, const aclFloatArray* matrix, uint32_t interpolationMode, uint32_t paddingMode, const aclFloatArray* fill, aclTensor *out, uint64_t *workspaceSize, aclOpExecutor **executor)**
- **acldvppStatus acldvppWarpPerspective(void *workspace, uint64_t workspaceSize, aclOpExecutor *executor, aclrtStream stream)**

函数功能

- 算子功能：对图像做透射变换。

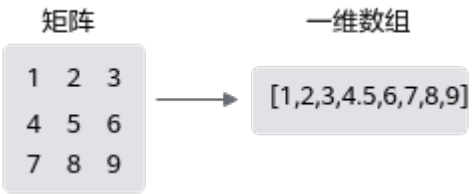
acldvppWarpPerspectiveWorkspaceSize

- **接口定义：**

  acldvppStatus acldvppWarpPerspectiveGetWorkspaceSize(const aclTensor *self, const aclFloatArray* matrix, uint32_t interpolationMode, uint32_t paddingMode,const aclFloatArray* fill, aclTensor *out, uint64_t *workspaceSize, aclOpExecutor **executor)

- **参数说明：**
  - self：表示算子输入Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输入Tensor的dataType支持UINT8/FLOAT、Format支持NCHW/NHWC、不支持**非连续的Tensor**，同时N支持1或空、C支持1和3（1表示输入GRAY图，3表示输入RGB图）。
  - matrix：长度为9的一维数组，数组中依次存放3*3的透射变换矩阵数据。

    数据存放示例如下：

  - interpolationMode：表示缩放算法，该参数的取值范围如下：
    - 0：bilinear算法
    - 1：nearest算法
  - paddingMode：填充模式，该参数取值如下：
    - 0：CONSTANT，填充固定值
    - 1：EDGE，重复最后一个元素。举例，其中*表示任意图像元素: aaaaaa|a*****h|hhhhhhh
  - fill: fill是一个长度为3的数组，用于设置每个通道上填充的值，需调用**aclCreateFloatArray**接口创建参数aclFloatArray类型的数据，仅在

paddingMode为CONSTANT时fill参数有效。如果C为1（表示GRAY图），填充fill[0]；如果C为3（表示RGB图），按照R、G、B顺序依次填写fill[0]、fill[1]、fill[2]。

- out：表示算子输出Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输出Tensor的dataType支持UINT8/FLOAT、Format支持NCHW/NHWC、不支持**非连续的Tensor**，同时N只支持1、C支持1和3（1表示GRAY图，3表示RGB图），dataType、Format、Shape需要和self一致。
- workspaceSize：返回用户需要在Device侧申请的workspace大小。
- executor：返回op执行器，包含了算子计算流程。

- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

acldvppWarpPerspective

- **接口定义：**

  acldvppStatus acldvppWarpPerspective(void* workspace, uint64_t workspaceSize, aclOpExecutor* executor, aclrtStream stream);

- **参数说明：**
  - workspace：需调用**aclrtMalloc**接口申请Device内存，内存大小为workspaceSize，**aclrtMalloc**接口输出的内存地址在此处传入。
  - workspaceSize：与acldvppWarpPerspectiveGetWorkspaceSize接口获取的workspaceSize保持一致。
  - executor：op执行器，包含了算子计算流程，与acldvppWarpPerspectiveGetWorkspaceSize接口的executor保持一致。
  - stream：指定执行任务的Stream，可复用已创建的Stream节省资源或调用**aclrtCreateStream**接口创建Stream，再作为入参在此处传入。

- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

约束与限制

- 支持图像分辨率范围在[6*4~4096*8192]。
- 变换矩阵大小是3*4。
- 输入输出格式宽高保持一致。
- 由于硬件限制，本算子接口内部计算耗时较长，性能可能不符合用户预期，后续版本进一步优化。

11.13.6.14 acldvppCropAndResize

Atlas 200/300/500 推理产品不支持该算子。

Atlas 推理系列产品不支持该算子。

接口原型

每个算子有两段接口，必须先调用“**acldvppCropAndResizeGetWorkspaceSize**”接口获取入参并根据计算流程计算所需workspace大小，再调用“**acldvppCropAndResize**”接口执行计算。两段式接口如下：

- **acldvppStatus acldvppCropAndResizeGetWorkspaceSize(const aclTensor *self, uint32_t top, uint32_t left, uint32_t height, uint32_t width, const aclIntArray* size, uint32_t interpolationMode, aclTensor *out, uint64_t *workspaceSize, aclOpExecutor **executor)**
- **acldvppStatus acldvppCropAndResize(void *workspace, uint64_t workspaceSize, aclOpExecutor *executor, aclrtStream stream)**

函数功能

- 算子功能：对图像做抠图缩放。

acldvppCropAndResizeGetWorkspaceSize

- **接口定义：**

  acldvppStatus acldvppCropAndResizeGetWorkspaceSize(const aclTensor *self, uint32_t top, uint32_t left, uint32_t height, uint32_t width, const aclIntArray* size, uint32_t interpolationMode, aclTensor *out, uint64_t *workspaceSize, aclOpExecutor **executor)

- **参数说明：**
  - self：表示算子输入Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输入Tensor的dataType支持UINT8/FLOAT、Format支持NCHW/NHWC、不支持**非连续的Tensor**，同时N支持1或空、C支持1或3（1表示输入GRAY图，3表示输入RGB图）。
  - top：抠图的上边界位置。
  - left：抠图的左边界位置。
  - height：抠图的高度，取值范围[4, 32768]。
  - width：抠图的宽度，取值范围[6, 32768]。
  - size：表示缩放之后的高、宽。需调用aclCreateIntArray接口创建参数aclIntArray类型的数据，数组长度为2，第一个元素是高、第二个元素是宽。缩放后宽高必须与输出宽高一致。
  - interpolationMode：表示缩放算法，该参数的取值范围如下：
    - 0：bilinear算法
    - 1：nearest算法
    - 2：bicubic算法
  - out：表示算子输出Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输出Tensor的dataType支持UINT8/FLOAT、Format支持NCHW/NHWC、不支持**非连续的Tensor**，同时N只支持1、C支持1或3（1表示GRAY图，3表示RGB图），dataType、Format需要和self一致，Shape中的N轴、C轴大小需要和self一致，Shape中的W轴、H轴大小需和缩放后的宽、高保持一致。
  - workspaceSize：返回用户需要在Device侧申请的workspace大小。

  - executor：返回op执行器，包含了算子计算流程。
- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

acldvppCropAndResize

- **接口定义：**

  acldvppStatus acldvppCropAndResize(void *workspace, uint64_t workspaceSize, aclOpExecutor *executor, aclrtStream stream)
- **参数说明：**
  - workspace：需调用**aclrtMalloc**接口申请Device内存，内存大小为workspaceSize，**aclrtMalloc**接口输出的内存地址在此处传入。
  - workspaceSize：与acldvppCropAndResizeGetWorkspaceSize接口获取的workspaceSize保持一致。
  - executor：op执行器，包含了算子计算流程，与acldvppCropAndResizeGetWorkspaceSize接口的executor保持一致。
  - stream：指定执行任务的Stream，可复用已创建的Stream节省资源或调用**aclrtCreateStream**接口创建Stream，再作为入参在此处传入。
- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

约束与限制

- 支持图像分辨率范围在[6*4~32768*32768]。
- 当输出out的shape的宽度超过4096时，输入self的shape的宽高不能小于128*16。

11.13.6.15 acldvppPad

Atlas 200/300/500 推理产品不支持该算子。

Atlas 推理系列产品不支持该算子。

接口原型

每个算子有两段接口，必须先调用“**acldvppPadGetWorkspaceSize**”接口获取入参并根据计算流程计算所需workspace大小，再调用“**acldvppPad**”接口执行计算。两段式接口如下：

- **acldvppStatus acldvppPadGetWorkspaceSize(const aclTensor *self, const aclIntArray* padding, uint32_t paddingMode, const aclFloatArray* fill, aclTensor *out, uint64_t *workspaceSize, aclOpExecutor **executor)**
- **acldvppStatus acldvppPad(void* workspace, uint64_t workspaceSize, aclOpExecutor* executor, aclrtStream stream)**

函数功能

- 算子功能：对图像做边界填充。

acldvppPadGetWorkspaceSize

- **接口定义：**

  acldvppStatus acldvppPadGetWorkspaceSize(const aclTensor *self, const aclIntArray* padding, uint32_t paddingMode, const aclFloatArray* fill, aclTensor *out, uint64_t *workspaceSize, aclOpExecutor **executor)

- **参数说明：**
  - self：表示算子输入Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输入Tensor的dataType支持UINT8/FLOAT、Format支持NCHW/NHWC、不支持**非连续的Tensor**，同时N支持1或空、C支持1和3（1表示输入GRAY图，3表示输入RGB图）。
  - padding: 表示四周边框填充宽度。需调用aclCreateIntArray接口创建参数aclIntArray类型的数据，长度为4，分别表示左、上、右、下四个方向上填充宽度，宽度范围[0, 2048]。
  - paddingMode：填充模式，该参数取值如下：
    - 0：CONSTANT，填充固定值
    - 1：EDGE，重复最后一个元素。举例，其中*表示任意图像元素: aaaaaa|a*****h|hhhhhhh
    - 2：REFLECT，边界元素的镜像，镜像不包括边界元素。举例，其中*表示任意图像元素：cb|abc****fgh|gf
    - 3：SYMMETRIC，边界元素的镜像，镜像包括边界元素。举例，其中*表示任意图像元素：ba|abc****fgh|hg
  - fill: fill是一个长度为3的数组，用于设置每个通道上填充的值，需调用**aclCreateFloatArray**接口创建参数aclFloatArray类型的数据，仅在paddingMode为CONSTANT时fill参数有效。如果C为1（表示GRAY图），填充fill[0]；如果C为3（表示RGB图），按照R、G、B顺序依次填写fill[0]、fill[1]、fill[2]。
  - out：表示算子输出Tensor，需调用**aclCreateTensor**接口创建aclTensor类型的数据（数据存放在Device侧），输出Tensor的dataType支持UINT8/FLOAT、Format支持NCHW/NHWC、不支持**非连续的Tensor**，同时N只支持1、C支持1和3（1表示GRAY图，3表示RGB图），dataType、Format需要和self一致，Shape中的N轴、C轴大小需要和self一致，Shape中的W轴、H轴大小需和填充后的宽、高保持一致。
  - workspaceSize：返回用户需要在Device侧申请的workspace大小。
  - executor：返回op执行器，包含了算子计算流程。

- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

acldvppPad

- **接口定义：**

  acldvppStatus acldvppPad(void* workspace, uint64_t workspaceSize, aclOpExecutor* executor, aclrtStream stream)

- **参数说明：**
  - workspace：需调用**aclrtMalloc**接口申请Device内存，内存大小为workspaceSize，**aclrtMalloc**接口输出的内存地址在此处传入。

  - workspaceSize：与acldvppPadGetWorkspaceSize接口获取的workspaceSize保持一致。
  - executor：op执行器，包含了算子计算流程，与acldvppPadGetWorkspaceSize接口的executor保持一致。
  - stream：指定执行任务的Stream，可复用已创建的Stream节省资源或调用**aclrtCreateStream**接口创建Stream，再作为入参在此处传入。
- **返回值：**

  返回acldvppStatus状态码，具体请参见**11.13.6.16 acldvpp返回码**。

约束与限制

支持图像分辨率范围在[6*4~32768*32768]。

11.13.6.16 acldvpp 返回码

调用DVPP类算子API时，常见的接口返回码如**表11-6**所示。

表 11-6 返回状态码

状态码名称	状态码值	状态码说明
ACLDVPP_SUCCESS	0	成功。
ACLDVPP_ERR_PARAM_NULLPTR	106001	参数校验错误，参数中存在非法的nullptr。
ACLDVPP_ERR_PARAM_INVALID	106002	参数校验错误，如输入的两个数据类型不满足输入类型推导关系。 详细的错误消息，可以通过**aclGetRecentErrMsg**接口获取（该接口在acl.h中）。
ACLDVPP_ERR_RUNTIME_ERROR	306001	API内存调用npu runtime的接口异常。
ACLDVPP_ERR_INNER_XXX	506xxx	API内部发生内部异常，详细的错误消息，可以通过**aclGetRecentErrMsg**接口获取。 请根据报错排查问题，或联系技术支持。（您可以获取日志后单击**Link**联系技术支持。）
ACLDVPP_ERR_INNER	506000	
ACLNN_ERR_INNER_CREATE_EXECUTOR	561101	
ACLNN_ERR_INNER_NOT_TRANS_EXECUTOR	561102	
ACLNN_ERR_INNER_NULLPTR	561103	

11.13.7 融合类算子接口

11.13.7.1 aclnnPromptFlashAttention

接口原型

每个算子分为两段式接口，必须先调用“aclnnPromptFlashAttentionGetWorkspaceSize”接口获取入参并根据计算流程计算所需workspace大小，再调用“aclnnPromptFlashAttention”接口执行计算。

- **aclnnStatus aclnnPromptFlashAttentionGetWorkspaceSize(const aclTensor *query, const aclTensor *key, const aclTensor *value, const aclTensor *paddingMask, const aclTensor *attenMask, const aclIntArray *actualSeqLengths, int64_t numHeads, double scaleValue, int64_t preTokens, int64_t nextTokens, char* inputLayout, int64_t numKeyValueHeads, const aclTensor *attentionOut, uint64_t *workspaceSize, aclOpExecutor **executor)**
- **aclnnStatus aclnnPromptFlashAttention(void *workspace, uint64_t workspaceSize, aclOpExecutor *executor, aclrtStream stream)**

功能描述

- **算子功能**：全量推理场景的FlashAttention算子。
- **计算公式**：

  self-attention（自注意力）利用输入样本自身的关系构建了一种注意力模型。其原理是假设有一个长度为n的输入样本序列x，x的每个元素都是一个d维向量，可以将每个d维向量看作一个token embedding，将这样一条序列经过3个权重矩阵变换得到3个维度为n*d的矩阵。

  self-attention的计算公式一般定义如下，其中Q、K、V为输入样本的重要属性元素，是输入样本经过空间变换得到，且可以统一到一个特征空间中。公式及算子名称中的"Attention"为"self-attention"的简写。

$$Attention(Q, K, V)=Score(Q, K)V$$

  本算子中Score函数采用Softmax函数，self-attention计算公式为：

$$Attention(Q, K, V)=Softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$$

  其中Q和K^T的乘积代表输入x的注意力，为避免该值变得过大，通常除以d的开根号进行缩放，并对每行进行softmax归一化，与V相乘后得到一个n*d的矩阵。

aclnnPromptFlashAttentionGetWorkspaceSize

- **参数说明：**
  - query（aclTensor*，计算输入）：Device侧的aclTensor，公式中的输入Q，数据类型支持FLOAT16、BFLOAT16、INT8，数据类型与key的数据类型需满足数据类型推导规则，即保持与key、value的数据类型一致。不支持非连续的Tensor，数据格式支持ND。
  - key（aclTensor*，计算输入）：Device侧的aclTensor，公式中的输入K，数据类型支持FLOAT16、BFLOAT16、INT8，数据类型与query的数据类型需满足数据类型推导规则，即保持与query、value的数据类型一致。不支持非连续的Tensor，数据格式支持ND。

- value（aclTensor*，计算输入）：Device侧的aclTensor，公式中的输入V，数据类型支持FLOAT16、BFLOAT16、INT8，数据类型与query的数据类型需满足数据类型推导规则，即保持与query、key的数据类型一致。不支持非连续的Tensor，数据格式支持ND。
- paddingMask（aclTensor*，计算输入）：。Device侧的aclTensor，数据类型支持FLOAT16、BOOL，数据类型与query的数据类型需满足数据类型推导规则。不支持非连续的Tensor，数据格式支持ND。**当前版本暂不支持该参数**，如果不使用该功能可传入nullptr。
- attenMask（aclTensor*，计算输入）：Device侧的aclTensor，代表下三角全为0上三角全为负无穷的倒三角mask矩阵，不支持非连续的Tensor，数据类型支持BOOL。数据格式支持ND。如果不使用该功能可传入nullptr。限制：其shape支持S,S;B,S,S;1,S,S;B,1,S,S；1,1,S,S；attenMask只支持数据类型BOOL。综合约束请见**约束与限制**。
- actualSeqLengths（aclIntArray*，计算输入）：Host侧的aclIntArray，代表不同Batch中query的有效Sequence Length，数据类型支持INT64。如果不指定seqlen可以传入nullptr，表示和query的shape的s长度相同。限制：该入参中每个batch中的有效Sequence Length应该不大于query中对应batch的Sequence Length；inputLayout为SH格式时，该入参各Batch的有效Sequence Length之和需要等于query的shape中的S。
- numHeads（int64_t，计算输入）：Host侧的int，代表query的head个数，数据类型支持INT64。限制：在BNSD/NSD场景下，需要与shape中的query的N轴shape值相同。
- scaleValue（double，计算输入）：Host侧的double，公式中d开根号的倒数，代表缩放系数，作为计算流中Muls的scalar值，数据类型支持DOUBLE。数据类型与query的数据类型需满足数据类型推导规则。用户不特意指定时可传入默认值1.0。
- preTokens（int64_t，计算输入）：Host侧的int，用于稀疏计算，表示attention需要和前几个Token计算关联，数据类型支持INT64。用户不特意指定时可传入默认值214748647，当前不支持负数。
- nextTokens（int64_t，计算输入）：Host侧的int，用于稀疏计算，表示attention需要和后几个Token计算关联。数据类型支持INT64。用户不特意指定时可传入默认值0，当前不支持负数。
- inputLayout（char*，计算输入）：Host侧的字符指针CHAR*，用于标识输入query、key、value的数据排布格式，当前支持BSH、BNSD、NSD、SH。用户不特意指定时可传入默认值"BSH”。限制：inputLayout为SH格式时，不支持S不等长场景（即不支持query与key/value的S不相等）。

  说明

  query、key、value数据排布格式支持从多种维度解读，其中B（Batch）表示输入样本批量大小、S（Seq-Length）表示输入样本序列长度、H（Head-Size）表示隐藏层的大小、N（Head-Num）表示多头数、D（Head-Dim）表示隐藏层最小的单元尺寸，且满足D=H/N。
- numKeyValueHeads（int64_t，计算输入）：Host侧的int，代表key、value中head个数，用于支持GQA（Grouped-Query Attention，分组查询注意力）场景，数据类型支持INT64。用户不特意指定时可传入默认值:0，表示和key/value的head个数相等。限制：在BNSD/NSD场景下，需要与shape中的key/value的N轴shape值相同，且需要满足numHeads整除numKeyValueHeads，否则报错。
- attentionOut（aclTensor*，计算输出）：Device侧的aclTensor，公式中的输出，数据类型支持FLOAT16、BFLOAT16、INT8。数据格式支持ND。限制：该入参的shape需要与入参query的shape保持一致。

  - workspaceSize（uint64_t*，出参）：返回用户需要在Device侧申请的workspace大小。
  - executor（aclOpExecutor**，出参）：返回op执行器，包含了算子计算流程。
- **返回值：**

  返回aclnnStatus状态码，具体参见**11.13.7.10 aclnn返回码**。

  说明

  第一段接口完成入参校验，若出现以下错误码，则对应原因为：
  - 返回161001（ACLNN_ERR_PARAM_NULLPTR）：如果传入参数是必选输入、输出或者必选属性，且是空指针，则返回161001。
  - 返回161002（ACLNN_ERR_PARAM_INVALID）：query、key、value、paddingMask、attenMask、attentionOut的数据类型和数据格式不在支持的范围内。

aclnnPromptFlashAttention

- **参数说明：**
  - workspace（void*，入参）：在Device侧申请的workspace内存起址。
  - workspaceSize（uint64_t，入参）：在Device侧申请的workspace大小，由**aclnnPromptFlashAttentionGetWorkspaceSize**获取。
  - executor（aclOpExecutor*，入参）：op执行器，包含了算子计算流程。
  - stream（aclrtStream，入参）：指定执行任务的AscendCL stream流。
- **返回值：**

  返回aclnnStatus状态码，具体参见**11.13.7.10 aclnn返回码**。

约束与限制

- 参数inputLayout为"B,S,H"时，Q的实际输入维度可以为2或者3。Q的实际输入维度为2时，Q按照"S,H"处理，actualSeqLengths未传入则默认B=1。
- 该接口与PyTorch配合使用时，需要保证CANN相关包与PyTorch相关包的版本匹配。
- 入参为空的处理：算子内部需要判断参数query是否为空，如果是空则直接返回。参数query不为空Tensor，参数key、value为空tensor（即S2为0），则填充全零的对应shape的输出（填充attention_out）。attention_out为空Tensor时，aclnn框架会处理。其余在上述参数说明中标注了"可传入nullptr"的入参为空指针时，不进行处理。
- D（Head-Dim：表示隐藏层最小的单元尺寸）不以16对齐时，入参query、key、value的数据类型只支持FLOAT16；入参query、key、value的数据类型为INT8时，D必须以32对齐。

11.13.7.2 aclnnPromptFlashAttentionV2

接口原型

每个算子分为两段式接口，必须先调用“aclnnPromptFlashAttentionV2GetWorkspaceSize”接口获取入参并根据计算流程计算所需workspace大小，再调用“aclnnPromptFlashAttentionV2”接口执行计算。

- **aclnnStatus aclnnPromptFlashAttentionV2GetWorkspaceSize(const aclTensor *query, const aclTensor *key, const aclTensor *value, const**

aclTensor *paddingMask, const aclTensor *attenMask, const aclIntArray *actualSeqLengths, const aclIntArray *actualSeqLengthsKv, const aclTensor *deqScale1, const aclTensor *quantScale1, const aclTensor *deqScale2, const aclTensor *quantScale2, const aclTensor *quantOffset2, int64_t numHeads, double scaleValue, int64_t preTokens, int64_t nextTokens, char *inputLayout, int64_t numKeyValueHeads, int64_t sparseMode, const aclTensor *attentionOut, uint64_t *workspaceSize, aclOpExecutor **executor)

- **aclnnStatus aclnnPromptFlashAttentionV2(void *workspace, uint64_t workspaceSize, aclOpExecutor *executor, const aclrtStream stream)**

功能描述

- 算子功能：全量推理场景的FlashAttention算子，相较于**11.13.7.1 aclnnPromptFlashAttention**接口，**此接口新增**了支持sparse优化、支持actualSeqLengthsKv优化、支持int8量化功能。
- 计算公式：

  self-attention（自注意力）利用输入样本自身的关系构建了一种注意力模型。其原理是假设有一个长度为n的输入样本序列x，x的每个元素都是一个d维向量，可以将每个d维向量看作一个token embedding，将这样一条序列经过3个权重矩阵变换得到3个维度为n*d的矩阵。

  self-attention的计算公式一般定义如下，其中Q、K、V为输入样本的重要属性元素，是输入样本经过空间变换得到，且可以统一到一个特征空间中。公式及算子名称中的"Attention"为"self-attention"的简写。

$$Attention(Q, K, V)=Score(Q, K)V$$

  本算子中Score函数采用Softmax函数，self-attention计算公式为：

$$Attention(Q, K, V)=Softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$$

  其中Q和K^T的乘积代表输入x的注意力，为避免该值变得过大，通常除以d的开根号进行缩放，并对每行进行softmax归一化，与V相乘后得到一个n*d的矩阵。

aclnnPromptFlashAttentionV2GetWorkspaceSize

- **参数说明：**
  - query（aclTensor*，计算输入）：Device侧的aclTensor，公式中的输入Q，数据类型支持FLOAT16、BFLOAT16、INT8，数据类型与key的数据类型需满足数据类型推导规则，即保持与key、value的数据类型一致。不支持非连续的Tensor，数据格式支持ND。
  - key（aclTensor*，计算输入）：Device侧的aclTensor，公式中的输入K，数据类型支持FLOAT16、BFLOAT16、INT8，数据类型与query的数据类型需满足数据类型推导规则，即保持与query、value的数据类型一致。不支持非连续的Tensor，数据格式支持ND。
  - value（aclTensor*，计算输入）：Device侧的aclTensor，公式中的输入V，数据类型支持FLOAT16、BFLOAT16、INT8，数据类型与query的数据类型需满足数据类型推导规则,，即保持与query、key的数据类型一致。不支持非连续的Tensor，数据格式支持ND。

- paddingMask（aclTensor*，计算输入）：该参数暂不支持。Device侧的aclTensor，数据类型支持FLOAT16、BOOL，数据类型与query的数据类型需满足数据类型推导规则。不支持非连续的Tensor，数据格式支持ND。如不使用该功能时可传入nullptr。
- attenMask（aclTensor*，计算输入）：Device侧的aclTensor，代表下三角全为0上三角全为负无穷的倒三角mask矩阵，不支持非连续的Tensor，数据类型支持BOOL。数据格式支持ND。如不使用该功能时可传入nullptr。限制：其shape支持S,S;B,S,S;1,S,S;B,1,S,S；1,1,S,S；attenMask只支持数据类型BOOL。综合约束请见**约束与限制**章节。
- actualSeqLengths（aclIntArray*，计算输入）：Host侧的aclIntArray，代表不同Batch中query的有效Sequence Length，数据类型支持INT64。如果不指定seqlen可以转入nullptr，表示和query的shape的s长度相同。限制：该入参中每个batch的有效Sequence Length应该不大于query中对应batch的Sequence Length；inputLayout为SH格式时，该入参各Batch的有效Sequence Length之和需要等于query的shape中的S。
- actualSeqLengthsKv（aclIntArray*，计算输入）：Host侧的aclIntArray，可传入nullptr，代表不同batch中key/value的有效Sequence Length。数据类型支持：INT64。限制：该入参中每个batch的有效Sequence Length应该不大于key/value中对应batch的Sequence Length。限制：inputLayout为SH格式时，actualSeqLengthsKv参数无效（若传入非空指针，则告警且忽略此参数）。
- deqScale1（aclTensor*，计算输入）：Device侧的aclTensor，数据类型支持：UINT64。数据格式支持ND（参考），表示BMM1后面反量化的量化因子，支持pre-tensor（scalar）。 如不使用该功能时可传入nullptr。
- quantScale1（aclTensor*，计算输入）：Device侧的aclTensor，数据类型支持：FLOAT。数据格式支持ND（参考），表示BMM2前面量化的量化因子，支持pre-tensor（scalar）。 如不使用该功能时可传入nullptr。
- deqScale2（aclTensor*，计算输入）：Device侧的aclTensor，数据类型支持：UINT64。数据格式支持ND（参考），表示BMM2后面量化的量化因子，支持pre-tensor（scalar）。 如不使用该功能时可传入nullptr。
- quantScale2（aclTensor*，计算输入）：Device侧的aclTensor，数据类型支持：FLOAT。数据格式支持ND（参考），表示输出量化的量化因子，支持pre-tensor（scalar）。 如不使用该功能时可传入nullptr。
- quantOffset2（aclTensor*，计算输入）：Device侧的aclTensor，数据类型支持：FLOAT。数据格式支持ND（参考），表示输出量化的量化偏移，支持pre-tensor（scalar）。 如不使用该功能时可传入nullptr。
- numHeads（int64_t，计算输入）：Host侧的int，代表query的head个数，数据类型支持INT64。
- scaleValue（double，计算输入）：Host侧的double，公式中d开根号的倒数，代表缩放系数，作为计算流中Muls的scalar值，数据类型支持DOUBLE。数据类型与query的数据类型需满足数据类型推导规则。用户不特意指定时可传入默认值:1.0。
- preTokens（int64_t，计算输入）：Host侧的int，用于稀疏计算，表示attention需要和前几个Token计算关联，数据类型支持INT64。用户不特意指定时可传入默认值:214748647，当前不支持负数。
- nextTokens（int64_t，计算输入）：Host侧的int，用于稀疏计算，表示attention需要和后几个Token计算关联。数据类型支持INT64。用户不特意指定时可传入默认值:0，当前不支持负数。
- inputLayout（char*，计算输入）：Host侧的字符指针CHAR*，用于标识输入query、key、value的数据排布格式，当前支持BSH、BNSD、NSD、SH。

用户不特意指定时可传入默认值:"BSH”。限制：inputLayout为SH格式时，不支持S不等长场景（即不支持query与key/value的S不相等）。

说明

query、key、value数据排布格式支持从多种维度解读，其中B（Batch）表示输入样本批量大小、S（Seq-Length）表示输入样本序列长度、H（Head-Size）表示隐藏层的大小、N（Head-Num）表示多头数、D（Head-Dim）表示隐藏层最小的单元尺寸，且满足D=H/N。

- numKeyValueHeads（int64_t，计算输入）：Host侧的int，代表key、value中head个数，用于支持GQA（Grouped-Query Attention，分组查询注意力）场景，数据类型支持INT64。用户不特意指定时可传入默认值:0，表示和key/value的head个数相等。限制：在BNSD/NSD场景下，需要与shape中的key/value的N轴shape值相同，且需要满足numHeads整除numKeyValueHeads，否则报错。
- sparseMode（int64_t，计算输入）：Host侧的int，表示sparse的模式。数据类型支持：INT64。sparseMode为0时，代表defaultMask模式，如果attenmask未传入则不做mask操作，忽略preTokens和nextTokens(内部赋值为INT_MAX)；如果传入，则需要传入完整的attenmask矩阵（S1 * S2），表示preTokens和nextTokens之间的部分需要计算。
  - sparseMode为为1时，代表allMask，暂不支持。
  - sparseMode为2时，代表leftUpCausal模式的mask，对应以左顶点为划分的下三角场景，需要传入优化后的attenmask矩阵（2048*2048）。
  - sparseMode为为3、4、5、6、7、8时，分别代表rightDownCausal、band、prefix、global、dilated、block_local，均暂不支持。用户不特意指定时可传入默认值:0。综合约束请见约束与限制章节。
- attentionOut（aclTensor*，计算输出）：Device侧的aclTensor，公式中的输出，数据类型支持FLOAT16、BFLOAT16、INT8。数据格式支持ND。限制：该入参的shape需要与入参query的shape保持一致。
- workspaceSize（uint64_t*，出参）：返回用户需要在Device侧申请的workspace大小。
- executor（aclOpExecutor**，出参）：返回op执行器，包含了算子计算流程。

- **返回值：**

  返回aclnnStatus状态码，具体参见**11.13.7.10 aclnn返回码**。

  说明

  第一段接口完成入参校验，若出现以下错误码，则对应原因为：
  - 返回161001（ACLNN_ERR_PARAM_NULLPTR）：如果传入参数是必选输入、输出或者必选属性，且是空指针，则返回161001。
  - 返回161002（ACLNN_ERR_PARAM_INVALID）：query、key、value、paddingMask、attenMask、attentionOut的数据类型和数据格式不在支持的范围内。

aclnnPromptFlashAttentionV2

- **参数说明：**
  - workspace（void*，入参）：在Device侧申请的workspace内存起址。
  - workspaceSize（uint64_t，入参）：在Device侧申请的workspace大小，由**aclnnPromptFlashAttentionV2GetWorkspaceSize**获取。

  - executor（aclOpExecutor*，入参）：op执行器，包含了算子计算流程。
  - stream（aclrtStream，入参）：指定执行任务的AscendCL stream流。
- **返回值：**

  返回aclnnStatus状态码，具体参见**11.13.7.10 aclnn返回码**。

约束与限制

- 参数inputLayout为"B,S,H"时，Q的实际输入维度可以为2或者3。Q的实际输入维度为2时，Q按照"S,H"处理，actualSeqLengths未传入则默认B=1。actualSeqLengthsKv未传入则默认B=1,传入则需要和actualSeqLengths的dim一致。
- 该接口与pytorch配合使用时，需要保证CANN相关包与PyTorch相关包的版本匹配。
- 入参为空的处理：算子内部需要判断参数query是否为空，如果是空则直接返回。参数query不为空Tensor，参数key、value为空tensor(即S2为0),则填充全零的对应shape的输出(填充attention_out)。attention_out为空Tensor时，AscendCLNN框架会处理。其余在上述参数说明中标注了"可传入nullptr"的入参为空指针时，不进行处理。
- 参数sparseMode当前仅支持值为0或2的场景，取其它值时会报错。sparseMode =0时，attenMask如果为空指针，则忽略入参preTokens、nextTokens(内部赋值为INT_MAX)。sparseMode = 2时，attenMask的shape需要为S,S或1,S,S或1,1,S,S,其中S的值需要固定为2048，且需要用户保证传入的attenMask为下三角，不传入attenMask或者传入的shape不正确报错，此场景忽略入参preTokens、nextTokens并按照相关规则赋值。
- D（Head-Dim：表示隐藏层最小的单元尺寸)不以16对齐时，入参query、key、value的数据类型只支持FLOAT16。入参query、key、value的数据类型为INT8时，D必须以32对齐。
- int8量化相关入参数量与输入、输出数据格式的综合限制：
  a. 输入为INT8，输出为INT8的场景：入参deqScale1、quantScale1、deqScale2、quantScale2、quantOffset2需要同时存在。
  b. 输入为INT8，输出为FLOAT16的场景：入参deqScale1、quantScale1、deqScale2需要同时存在，若存在入参quantScale2、quantOffset2时（即不为nullptr），则忽略入参quantScale2、quantOffset2。

11.13.7.3 aclnnIncreFlashAttention

接口原型

每个算子分为两段式接口，必须先调用“aclnnIncreFlashAttentionGetWorkspaceSize”接口获取入参并根据计算流程计算所需workspace大小，再调用“aclnnIncreFlashAttention”接口执行计算。

- **aclnnStatus aclnnIncreFlashAttentionGetWorkspaceSize(const aclTensor *query, const aclTensorList *key, const aclTensorList *value, const aclTensor *paddingMask, const aclTensor *attenMask, const aclIntArray *actualSeqLengths, int64_t numHeads, double scaleValue, char *inputLayout, int64_t numKeyValueHeads, const aclTensor *attentionOut, uint64_t *workspaceSize, aclOpExecutor **executor)**
- **aclnnstatus aclnnIncreFlashAttention(void *workspace, uint64_t workspaceSize, aclOpExecutor *executor, aclrtStream stream)**

功能描述

- 算子功能：对于自回归（Auto-regressive）的语言模型，随着新词的生成，推理输入长度不断增大。在原来全量推理的基础上**实现增量推理**，query的S轴固定为1，key和value是经过KV Cache后，将之前推理过的state信息，叠加在一起，每个Batch对应S轴的实际长度可能不一样，输入的数据是经过padding后的固定长度数据。

  相比全量场景的FlashAttention算子（**11.13.7.1 aclnnPromptFlashAttention**），增量推理的流程与正常全量推理并不完全等价，不过增量推理的精度并无明显劣化。

  说明

  KV Cache是大模型推理性能优化的一个常用技术。采样时，Transformer模型会以给定的prompt/context作为初始输入进行推理（可以并行处理），随后逐一生成额外的token来继续完善生成的序列（体现了模型的自回归性质）。在采样过程中，Transformer会执行自注意力操作，为此需要给当前序列中的每个项目（无论是prompt/context还是生成的token）提取键值（KV）向量。这些向量存储在一个矩阵中，通常被称为kv缓存（KV Cache）。

- 计算公式：

  self-attention（自注意力）利用输入样本自身的关系构建了一种注意力模型。其原理是假设有一个长度为n的输入样本序列x，x的每个元素都是一个d维向量，可以将每个d维向量看作一个token embedding，将这样一条序列经过3个权重矩阵变换得到3个维度为n*d的矩阵。

  self-attention的计算公式一般定义如下，其中Q、K、V为输入样本的重要属性元素，是输入样本经过空间变换得到，且可以统一到一个特征空间中。

$$Attention(Q, K, V)=Score(Q, K)V$$

  本算子中Score函数采用Softmax函数，self-attention计算公式为

$$Attention(Q, K, V)=Softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$$

  其中Q和K^T的乘积代表输入x的注意力，为避免该值变得过大，通常除以d的开根号进行缩放，并对每行进行softmax归一化，与V相乘后得到一个n*d的矩阵。

aclnnIncreFlashAttentionGetWorkspaceSize

- **参数说明：**
  - query（aclTensor*，计算输入）：Device侧的aclTensor，公式中的输入Q，数据类型支持FLOAT16，数据格式支持ND。
  - key（aclTensorList*，计算输入）：Device侧的aclTensorList，公式中的输入K，数据类型支持FLOAT16，数据格式支持ND。
  - value（aclTensorList*，计算输入）：Device侧的aclTensorList，公式中的输入V，数据类型支持FLOAT16，数据格式支持ND。
  - paddingMask（aclTensor*，计算输入）：Device侧的aclTensor，可选参数，数据类型支持FLOAT16，数据格式支持ND。
  - attenMask（aclTensor*，计算输入）：Device侧的aclTensor，可选参数，数据类型支持BOOL、FLOAT16，数据格式支持ND。
  - actualSeqLengths（aclIntArray*，计算输入）：Host侧的aclIntArray，可选参数，数据类型支持INT64。

- numHeads（int64_t，计算输入）：Host侧的int64_t，代表head个数，数据类型支持INT64。
- scaleValue（double，计算输入）：Host侧的double，公式中d开根号的倒数，代表缩放系数，作为计算流中Muls的scalar值，数据类型支持DOUBLE。
- inputLayout（char*，计算输入）：Host侧的字符指针，用于标识输入query、key、value的数据排布格式，当前支持BSH、BNSD。

  说明

  query、key、value数据排布格式支持从多种维度解读，其中B（Batch）表示输入样本批量大小、S（Seq-Length）表示输入样本序列长度、H（Head-Size）表示隐藏层的大小、N（Head-Num）表示多头数、D（Head-Dim）表示隐藏层最小的单元尺寸，且满足D=H/N。
- numKeyValueHeads（int64_t，计算输入）：Host侧的int64_t，代表key、value中head个数，用于支持GQA（Grouped-Query Attention，分组查询注意力）场景，默认为0，表示和query的head个数相等，数据类型支持INT64。
- attentionOut（aclTensor*，计算输出）：Device侧的aclTensor，公式中的输出，数据类型支持FLOAT16，数据格式支持ND。
- workspaceSize（uint64_t*，出参）：返回用户需要在Device侧申请的workspace大小。
- executor（aclOpExecutor**，出参）：返回op执行器，包含了算子计算流程。

- **返回值：**

  返回aclnnStatus状态码，具体参见**11.13.7.10 aclnn返回码**。

  说明

  第一段接口完成入参校验，若出现以下错误码，则对应原因为：
  - 返回161001（ACLNN_ERR_PARAM_NULLPTR）：如果传入参数是必选输入、输出或者必选属性，且是空指针，则返回161001。
  - 返回161002（ACLNN_ERR_PARAM_INVALID）：query、key、value、paddingMask、attenMask、attentionOut的数据类型和数据格式不在支持的范围内。

aclnnIncreFlashAttention

- **参数说明：**
  - workspace（void*，入参）：在Device侧申请的workspace内存起址。
  - workspaceSize（uint64_t，入参）：在Device侧申请的workspace大小，由**aclnnIncreFlashAttentionGetWorkspaceSize**获取。
  - executor（aclOpExecutor*，入参）：op执行器，包含了算子计算流程。
  - stream（aclrtStream，入参）：指定执行任务的AscendCL stream流。
- **返回值：**

  返回aclnnStatus状态码，具体参见**11.13.7.10 aclnn返回码**。

约束与限制

- 参数key、value 中对应tensor的shape需要完全一致；非连续场景下 key、value的tensorlist中的tensor的shape也需要完全一致，且batch只能为1。
- 参数query和attentionOut的shape需要完全一致。

- 参数query中的N和numHeads值相等，key、value的N和numKeyValueHeads值相等，并且numHeads是numKeyValueHeads的倍数关系。
- 非连续场景下，参数key、value的tensorlist中tensor的个数等于query的B，shape需要完全一致。

11.13.7.4 aclnnIncreFlashAttentionV2

接口原型

每个算子分为两段式接口，必须先调用“aclnnIncreFlashAttentionV2GetWorkspaceSize”接口获取入参并根据计算流程计算所需workspace大小，再调用“aclnnIncreFlashAttentionV2”接口执行计算。

- **aclnnStatus aclnnIncreFlashAttentionV2GetWorkspaceSize(const aclTensor *query, const aclTensorList *key, const aclTensorList *value, const aclTensor *paddingMask, const aclTensor *attenMask, const aclIntArray *actualSeqLengths, const aclTensor *dequantScale1, const aclTensor *quantScale1, const aclTensor *dequantScale2, const aclTensor *quantScale2, const aclTensor *quantOffset2, int64_t numHeads, double scaleValue, char *inputLayout, int64_t numKeyValueHeads, const aclTensor *attentionOut, uint64_t *workspaceSize, aclOpExecutor **executor)**
- **aclnnstatus aclnnIncreFlashAttentionV2(void *workspace, uint64_t workspaceSize, aclOpExecutor *executor, aclrtStream stream)**

功能描述

- **算子功能**：兼容**11.13.7.3 aclnnIncreFlashAttention**接口功能，在其基础上**新增量化特性**。

  对于自回归（Auto-regressive）的语言模型，随着新词的生成，推理输入长度不断增大。在原来全量推理的基础上**实现增量推理**，query的S轴固定为1，key和value是经过KV Cache后，将之前推理过的state信息，叠加在一起，每个Batch对应S轴的实际长度可能不一样，输入的数据是经过padding后的固定长度数据。

  相比全量场景的FlashAttention算子（**11.13.7.1 aclnnPromptFlashAttention**），增量推理的流程与正常全量推理并不完全等价，不过增量推理的精度并无明显劣化。

  说明

  KV Cache是大模型推理性能优化的一个常用技术。采样时，Transformer模型会以给定的prompt/context作为初始输入进行推理（可以并行处理），随后逐一生成额外的token来继续完善生成的序列（体现了模型的自回归性质）。在采样过程中，Transformer会执行自注意力操作，为此需要给当前序列中的每个项目（无论是prompt/context还是生成的token）提取键值（KV）向量。这些向量存储在一个矩阵中，通常被称为kv缓存（KV Cache）。

- **计算公式**：

  self-attention（自注意力）利用输入样本自身的关系构建了一种注意力模型。其原理是假设有一个长度为n的输入样本序列x，x的每个元素都是一个d维向量，可以将每个d维向量看作一个token embedding，将这样一条序列经过3个权重矩阵变换得到3个维度为n*d的矩阵。

  self-attention的计算公式一般定义如下，其中Q、K、V为输入样本的重要属性元素，是输入样本经过空间变换得到，且可以统一到一个特征空间中。

$$Attention(Q, K, V)=Score(Q, K)V$$

本算子中Score函数采用Softmax函数，self-attention计算公式为

$$Attention(Q, K, V)=Softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$$

其中Q和K^T的乘积代表输入x的注意力，为避免该值变得过大，通常除以d的开根号进行缩放，并对每行进行softmax归一化，与V相乘后得到一个n*d的矩阵。

aclnnIncreFlashAttentionV2GetWorkspaceSize

- **参数说明：**
  - query（aclTensor*，计算输入）：Device侧的aclTensor，公式中的输入Q，数据类型支持FLOAT16/INT8/BFLOAT16，数据格式支持ND。
  - key（aclTensorList*，计算输入）：Device侧的aclTensorList，公式中的输入K，数据类型支持FLOAT16/INT8/BFLOAT16，数据格式支持ND。
  - value（aclTensorList*，计算输入）：Device侧的aclTensorList，公式中的输入V，数据类型支持FLOAT16/INT8/BFLOAT16 ，数据格式支持ND。
  - paddingMask（aclTensor*，计算输入）：Device侧的aclTensor，暂不生效，数据类型支持FLOAT16，数据格式支持ND。
  - attenMask（aclTensor*，计算输入）：Device侧的aclTensor，可选参数，数据类型支持BOOL、FLOAT16，数据格式支持ND。
  - actualSeqLengths（aclIntArray*，计算输入）：Host侧的aclIntArray，可选参数，数据类型支持INT64。
  - dequantScale1（aclTensor*，计算输入）：Device侧的aclTensor，数据类型支持：UINT64。数据格式支持ND，表示BMM1后面反量化的量化因子，支持pre-tensor（scalar）。 如不使用该功能时可传入nullptr。
  - quantScale1（aclTensor*，计算输入）：Device侧的aclTensor，数据类型支持：FLOAT。数据格式支持ND，表示BMM2前面量化的量化因子，支持pre-tensor（scalar）。 如不使用该功能时可传入nullptr。
  - dequantScale2（aclTensor*，计算输入）：Device侧的aclTensor，数据类型支持：UINT64。数据格式支持ND，表示BMM2后面量化的量化因子，支持pre-tensor（scalar）。 如不使用该功能时可传入nullptr。
  - quantScale2（aclTensor*，计算输入）：Device侧的aclTensor，数据类型支持：FLOAT。数据格式支持ND，表示输出量化的量化因子，支持pre-tensor（scalar）。 如不使用该功能时可传入nullptr。
  - quantOffset2（aclTensor*，计算输入）：Device侧的aclTensor，数据类型支持：FLOAT。数据格式支持ND，表示输出量化的量化偏移，支持pre-tensor（scalar）。 如不使用该功能时可传入nullptr。
  - numHeads（int64_t，计算输入 ）：Host侧的int64_t，代表head个数，数据类型支持INT64。
  - scaleValue（double，计算输入）：Host侧的double，公式中d开根号的倒数，代表缩放系数，作为计算流中Muls的scalar值，数据类型支持DOUBLE。
  - inputLayout（char*，计算输入）：Host侧的字符指针，用于标识输入query、key、value的数据排布格式，当前支持BSH、BNSD。

说明

query、key、value数据排布格式支持从多种维度解读，其中B（Batch）表示输入样本批量大小、S（Seq-Length）表示输入样本序列长度、H（Head-Size）表示隐藏层的大小、N（Head-Num）表示多头数、D（Head-Dim）表示隐藏层最小的单元尺寸，且满足D=H/N。

- numKeyValueHeads（int64_t，计算输入）：Host侧的int64_t，代表key、value中head个数，用于支持GQA（Grouped-Query Attention，分组查询注意力）场景，默认为0，表示和query的head个数相等，数据类型支持INT64。
- attentionOut（aclTensor*，计算输出）：Device侧的aclTensor，公式中的输出，数据类型支持FLOAT16，数据格式支持ND。
- workspaceSize（uint64_t*，出参）：返回用户需要在Device侧申请的workspace大小。
- executor（aclOpExecutor**，出参）：返回op执行器，包含了算子计算流程。

- **返回值：**

  返回aclnnStatus状态码，具体参见**11.13.7.10 aclnn返回码**。

  说明

  第一段接口完成入参校验，出现以下场景时报错：

  - 返回161001（ACLNN_ERR_PARAM_NULLPTR）：如果传入参数是必选输入、输出或者必选属性，且是空指针，则返回161001。
  - 返回161002（ACLNN_ERR_PARAM_INVALID）：query、key、value、paddingMask、attenMask、attentionOut的数据类型和数据格式不在支持的范围内。

aclnnIncreFlashAttentionV2

- **参数说明：**
  - workspace（void*，入参）：在Device侧申请的workspace内存起址。
  - workspaceSize（uint64_t，入参）：在Device侧申请的workspace大小，由**aclnnIncreFlashAttentionV2GetWorkspaceSize**获取。
  - executor（aclOpExecutor*，入参）：op执行器，包含了算子计算流程。
  - stream（aclrtStream，入参）：指定执行任务的AscendCL stream流。
- **返回值：**

  返回aclnnStatus状态码，具体参见**11.13.7.10 aclnn返回码**。

约束与限制

- 参数key、value 中对应tensor的shape需要完全一致；非连续场景下 key、value的tensorlist中的tensor的shape也需要完全一致，且batch只能为1。
- 参数query和attentionOut的shape需要完全一致。
- 参数query中的N和numHeads值相等，key、value的N和numKeyValueHeads值相等，并且numHeads是numKeyValueHeads的倍数关系。
- 非连续场景下，参数key、value的tensorlist中tensor的个数等于query的B，shape需要完全一致。
- D的限制为16k，大于16k会报错拦截。

- int8量化相关入参数量与输入、输出数据格式的综合限制：
  a. 输入为INT8，输出为INT8的场景：入参dequantScale1、quantScale1、dequantScale2、quantScale2、quantOffset2需要同时存在。
  b. 输入为INT8，输出为FLOAT16的场景：入参dequantScale1、quantScale1、dequantScale2需要同时存在，不能传入quantScale2、quantOffset2（传nullptr）参数。
  c. 输入为FLOAT16，输出为INT8的场景：入参quantScale2、quantOffset2需要同时存在，不能传入dequantScale1、quantScale1、dequantScale2（传nullptr）参数。

```
#include <iostream>
#include <vector>
#include "acl/acl.h"
#include "aclnnop/aclnn_add.h"
#include "aclnn_incre_flash_attention_v2.h"

#define CHECK_RET(cond, return_expr) \
  do {                              \
    if (!(cond)) {                    \
      return_expr;                    \
    }                               \
  } while (0)

#define LOG_PRINT(message, ...)     \
  do {                              \
    printf(message, ##__VA_ARGS__); \
  } while (0)

int64_t GetShapeSize(const std::vector<int64_t>& shape) {
  int64_t shape_size = 1;
  for (auto i : shape) {
    shape_size *= i;
  }
  return shape_size;
}

int Init(int32_t deviceId, aclrtContext* context, aclrtStream* stream) {
  // 固定写法，AscendCL初始化
  auto ret = aclInit(nullptr);
  CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclInit failed. ERROR: %d\n", ret); return ret);
  ret = aclrtSetDevice(deviceId);
  CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtSetDevice failed. ERROR: %d\n", ret); return ret);
  ret = aclrtCreateContext(context, deviceId);
  CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtCreateContext failed. ERROR: %d\n", ret); return ret);
  ret = aclrtSetCurrentContext(*context);
  CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtSetCurrentContext failed. ERROR: %d\n", ret); return
ret);
  ret = aclrtCreateStream(stream);
  CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtCreateStream failed. ERROR: %d\n", ret); return ret);
  return 0;
}

template <typename T>
int CreateAclTensor(const std::vector<T>& hostData, const std::vector<int64_t>& shape, void** deviceAddr,
                    aclDataType dataType, aclTensor** tensor) {
  auto size = GetShapeSize(shape) * sizeof(T);
  // 调用aclrtMalloc申请device侧内存
  auto ret = aclrtMalloc(deviceAddr, size, ACL_MEM_MALLOC_HUGE_FIRST);
  CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtMalloc failed. ERROR: %d\n", ret); return ret);
  // 调用aclrtMemcpy将Host侧数据拷贝到device侧内存上
  ret = aclrtMemcpy(*deviceAddr, size, hostData.data(), size, ACL_MEMCPY_HOST_TO_DEVICE);
  CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtMemcpy failed. ERROR: %d\n", ret); return ret);
  // 计算连续tensor的strides
  std::vector<int64_t> strides(shape.size(), 1);
  for (int64_t i = shape.size() - 2; i >= 0; i--) {
    strides[i] = shape[i + 1] * strides[i + 1];
```

```
  }
  // 调用aclCreateTensor接口创建aclTensor
  *tensor = aclCreateTensor(shape.data(), shape.size(), dataType, strides.data(), 0,
aclFormat::ACL_FORMAT_ND,
                          shape.data(), shape.size(), *deviceAddr);
  return 0;
}

int main() {
  // 1.（固定写法）device/context/stream初始化, 参考AscendCL对外接口列表
  // 根据自己的实际device填写deviceId
  int32_t deviceId = 0;
  aclrtContext context;
  aclrtStream stream;
  auto ret = Init(deviceId, &context, &stream);
  // check根据自己的需要处理
  CHECK_RET(ret == 0, LOG_PRINT("Init acl failed. ERROR: %d\n", ret); return ret);

  // 2. 构造输入与输出，需要根据API的接口自定义构造
  std::vector<int64_t> selfShape = {4, 2};
  std::vector<int64_t> otherShape = {4, 2};
  std::vector<int64_t> outShape = {4, 2};
  void* selfDeviceAddr = nullptr;
  void* otherDeviceAddr = nullptr;
  void* outDeviceAddr = nullptr;
  aclTensor* self = nullptr;
  aclTensor* other = nullptr;
  aclScalar* alpha = nullptr;
  aclTensor* out = nullptr;
  std::vector<float> selfHostData = {0, 1, 2, 3, 4, 5, 6, 7};
  std::vector<float> otherHostData = {1, 1, 1, 2, 2, 2, 3, 3};
  std::vector<float> outHostData = {0, 0, 0, 0, 0, 0, 0, 0};
  float alphaValue = 1.2f;
  // 创建self aclTensor
  ret = CreateAclTensor(selfHostData, selfShape, &selfDeviceAddr, aclDataType::ACL_FLOAT, &self);
  CHECK_RET(ret == ACL_SUCCESS, return ret);
  // 创建other aclTensor
  ret = CreateAclTensor(otherHostData, otherShape, &otherDeviceAddr, aclDataType::ACL_FLOAT, &other);
  CHECK_RET(ret == ACL_SUCCESS, return ret);
  // 创建alpha aclScalar
  alpha = aclCreateScalar(&alphaValue, aclDataType::ACL_FLOAT);
  CHECK_RET(alpha != nullptr, return ret);
  // 创建out aclTensor
  ret = CreateAclTensor(outHostData, outShape, &outDeviceAddr, aclDataType::ACL_FLOAT, &out);
  CHECK_RET(ret == ACL_SUCCESS, return ret);

  // 3.调用CANN算子库API，需要修改为具体的算子接口
  uint64_t workspaceSize = 0;
  aclOpExecutor* executor;
  // 调用aclnnIncreFlashAttentionV2第一段接口
  ret = aclnnIncreFlashAttentionV2GetWorkspaceSize(tensors[q_idx], tensor_key_list, tensor_value_list,
                                                   nullptr, tensors[attenMask_idx], actual_seq_lengths,
                                                   tensors[dequantScale1_idx],
                                                   tensors[quantScale1_idx],
                                                   tensors[dequantScale2_idx],
                                                   tensors[quantScale2_idx],
                                                   tensors[quantOffset2_idx],
                                                   num_heads,
                                                   scale_value,
                                                   layer_out,
                                                   numKeyValueHeads,
                                                   tensors[out_idx], &workspace_size, &handle);
  CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclnnAddGetWorkspaceSize failed. ERROR: %d\n", ret);
return ret);
  // 根据第一段接口计算出的workspaceSize申请device内存
  void* workspaceAddr = nullptr;
  if (workspaceSize > 0) {
    ret = aclrtMalloc(&workspaceAddr, workspaceSize, ACL_MEM_MALLOC_HUGE_FIRST);
    CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("allocate workspace failed. ERROR: %d\n", ret); return
```

```
ret;);
  }
  // 调用aclnnIncreFlashAttentionV2第二段接口
  ret = aclnnIncreFlashAttentionV2(workspace, workspace_size, handle, stream);
  CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclnnAdd failed. ERROR: %d\n", ret); return ret);

  // 4.（ 固定写法 ）同步等待任务执行结束
  ret = aclrtSynchronizeStream(stream);
  CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtSynchronizeStream failed. ERROR: %d\n", ret); return
ret);

  // 5. 获取输出的值，将device侧内存上的结果拷贝至Host侧，需要根据具体API的接口定义修改
  auto size = GetShapeSize(outShape);
  std::vector<float> resultData(size, 0);
  ret = aclrtMemcpy(resultData.data(), resultData.size() * sizeof(resultData[0]), outDeviceAddr, size *
sizeof(float),
                    ACL_MEMCPY_DEVICE_TO_HOST);
  CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("copy result from device to host failed. ERROR: %d\n", ret);
return ret);
  for (int64_t i = 0; i < size; i++) {
    LOG_PRINT("result[%ld] is: %f\n", i, resultData[i]);
  }

  // 6. 释放aclTensor和aclScalar，需要根据具体API的接口定义修改
  aclDestroyTensor(self);
  aclDestroyTensor(other);
  aclDestroyScalar(alpha);
  aclDestroyTensor(out);

  // 7. 释放device资源，需要根据具体API的接口定义修改
  aclrtFree(selfDeviceAddr);
  aclrtFree(otherDeviceAddr);
  aclrtFree(outDeviceAddr);
  if (workspaceSize > 0) {
    aclrtFree(workspaceAddr);
  }
  aclrtDestroyStream(stream);
  aclrtDestroyContext(context);
  aclrtResetDevice(deviceId);
  aclFinalize();
  return 0;
}
```

11.13.7.5 aclnnFlashAttentionScore

接口原型

每个算子分为两段式接口，必须先调用“aclnnFlashAttentionScoreGetWorkspaceSize”接口获取入参并根据计算流程计算所需workspace大小，再调用“aclnnFlashAttentionScore”接口执行计算。

- **aclnnStatus aclnnFlashAttentionScoreGetWorkspaceSize(const aclTensor *query, const aclTensor *key, const aclTensor *value, const aclTensor *realShiftOptional, const aclTensor *dropMaskOptional, const aclTensor *paddingMaskOptional, const aclTensor *attenMaskOptional, const aclIntArray *prefixOptional, double scaleValueOptional, double keepProbOptional, int64_t preTockensOptional, int64_t nextTockensOptional, int64_t headNum, char *inputLayout, int64_t innerPreciseOptional, int64_t sparseModeOptional, const aclTensor *softmaxMaxOut, const aclTensor *softmaxSumOut, const aclTensor *softmaxOutOut, const aclTensor *attentionOutOut, uint64_t *workspaceSize, aclOpExecutor **executor)**

- **aclnnStatus aclnnFlashAttentionScore(void *workspace, uint64_t workspaceSize, aclOpExecutor *executor, const aclrtStream stream)**

功能描述

- 算子功能：训练场景下，使用FlashAttention算法实现self-attention（自注意力）的计算。
- 计算公式：

  注意力的正向计算公式如下：

$$Attention(Q,K,V)=Dropout\left(Softmax\left(Mask\left(\frac{QK^T+pse}{\sqrt{d}}\right)\right)\right)V$$

aclnnFlashAttentionScoreGetWorkspaceSize

- **参数说明：**
  - query（(aclTensor*，计算输入)：即公式中的输入Q。数据类型支持：FLOAT、FLOAT16、BFLOAT16。数据类型与key/value的数据类型一致。数据格式支持ND。
  - key（aclTensor*，计算输入）：即公式中的输入K。数据类型支持：FLOAT、FLOAT16、BFLOAT16。数据类型与query/value的数据类型一致。数据格式支持ND。
  - value（aclTensor*，计算输入）：即公式中的输入V。数据类型支持：FLOAT、FLOAT16、BFLOAT16。数据类型与query/key的数据类型一致。数据格式支持ND。
  - realShiftOptional（aclTensor*，计算输入）：即公式中的输入pse。数据类型支持：FLOAT、FLOAT16、BFLOAT16。数据类型与query的数据类型一致。数据格式支持ND。

    alibi位置编码场景下，数据格式每个batch参数不同[B N S S]/每个batch参数相同[1 N S S]。
  - dropMaskOptional（aclTensor*，计算输入）：数据类型支持：UINT8。数据格式支持ND。
  - paddingMaskOptional（aclTensor*，计算输入）：保留输入，暂未使用。
  - attenMaskOptional（aclTensor*，计算输入）：数据类型支持：BOOL。数据格式支持ND。
  - prefixOptional（aclIntArray*，计算输入）：数据类型支持：FLOAT、FLOAT16、BFLOAT16。数据类型与query/key/value的数据类型一致。数据格式支持ND。
  - scaleValueOptional（double，计算输入）：公式中d开根号的倒数，数据类型支持：DOUBLE，代表缩放系数，作为计算流中Muls的scalar值。
  - keepProbOptional（double，计算输入）：数据类型支持：DOUBLE，代表dropMask中1的比例。
  - preTockensOptional（int64_t，计算输入）：数据类型支持：INT64，用于稀疏计算 ，表示slides window的左边界。
  - nextTockensOptional（int64_t，计算输入）：数据类型支持：INT64，用于稀疏计算，表示slides window的右边界。
  - headNum（int64_t，计算输入）：数据类型支持：INT64，代表单卡的head个数。

- inputLayout（char*，计算输入）：数据类型支持：String，代表输入query、key、value的数据排布格式，支持BSH、SBH、BSND、BNSD。

  说明

  query、key、value数据排布格式支持从多种维度解读，其中B（Batch）表示输入样本批量大小、S（Seq-Length）表示输入样本序列长度、H（Head-Size）表示隐藏层的大小、N（Head-Num）表示多头数、D（Head-Dim）表示隐藏层最小的单元尺寸，且满足D=H/N。

- innerPreciseOptional（int64_t，计算输入）：保留参数，暂未使用。
- sparseModeOptional（int64_t，计算输入）：数据类型支持：INT64。用户不特意指定时可传入默认值：0。
  - sparseMode为0时，代表defaultMask模式，如果attenmask未传入则不做mask操作，忽略preTokens和nextTokens(内部赋值为INT_MAX)；如果传入，则需要传入完整的attenmask矩阵（S1 * S2），表示preTokens和nextTokens之间的部分需要计算。
  - sparseMode为为1时，代表allMask，即传入完整的attenmask矩阵。
  - sparseMode为2时，代表leftUpCausal模式的mask，对应以左上顶点为划分的下三角场景，需要传入shape为[2048, 2048]的下三角attenmask矩阵。
  - sparseMode为3时，代表rightDownCausal模式的mask，对应以右下顶点为划分的下三角场景，需要传入shape为[2048, 2048]的下三角attenmask矩阵。
  - sparseMode为为4时，代表band场景，即计算preTokens和nextTokens之间的部分。
  - sparseMode为为5时，代表prefix场景，即在rightDownCasual的基础上，左侧加上一个长为S1，宽为N的矩阵，N的值由新增的输入prefix获取，且每个Batch轴的N值不一样。
  - sparseMode为为6、7、8时，分别代表global、dilated、block_local，**均暂不支持**。

  注意：当所有的attenmask的shape小于2048且相同的时候，建议使用default模式，来减少内存使用量；sparse_mode配置为2或3时，不能配置preTokens、nextTokens。
- softmaxMaxOut（aclTensor*，计算输出）：数据类型支持：FLOAT。数据格式支持ND。
- softmaxSumOut（aclTensor*，计算输出）：数据类型支持：FLOAT。数据格式支持ND。
- softmaxOutOut（aclTensor*，计算输出）：保留输出，暂未使用。
- attentionOutOut（aclTensor*，计算输出）：数据类型支持：FLOAT、FLOAT16、BFLOAT16。数据类型query的数据类型一致。数据格式支持ND。
- workspaceSize（uint64_t*，出参）：返回需要在npu device侧申请的workspace大小。
- executor（aclOpExecutor**，出参）：返回op执行器，包含了算子计算流程。

- **返回值：**

  返回aclnnStatus状态码，具体参见**11.13.7.10 aclnn返回码**。

  说明

  第一段接口完成入参校验，若出现以下错误码，则对应原因为：

  - 返回161001（ACLNN_ERR_PARAM_NULLPTR）：如果传入参数是必选输入、输出或者必选属性，且是空指针，则返回161001。
  - 返回161002（ACLNN_ERR_PARAM_INVALID）：query、key、value、realShift、dropMask、paddingMask、attenMask、softmaxMax、softmaxSum、softmaxOut、attentionOut的数据类型和数据格式不在支持的范围内。

aclnnFlashAttentionScore

- **参数说明：**
  - workspace（void*，入参）：在Device侧申请的workspace内存起址。
  - workspaceSize（uint64_t，入参）：在Device侧申请的workspace大小，由**aclnnFlashAttentionScoreGetWorkspaceSize**获取。
  - executor（aclOpExecutor*，入参）：op执行器，包含了算子计算流程。
  - stream（aclrtStream，入参）：指定执行任务的AscendCL stream流。
- **返回值：**

  返回aclnnStatus状态码，具体参见**11.13.7.10 aclnn返回码**。

约束与限制

关于数据shape的约束，以inputLayout的BSND、BNSD为例（BSH、SBH下H=N*D），其中：

- B：取值范围为1~256。
- N：取值范围为1~256。
- S：取值范围为1~32K，且为16的倍数。
- D：取值为64、96、128、256。

11.13.7.6 aclnnFlashAttentionScoreGrad

接口原型

每个算子分为两段式接口，必须先调用“aclnnFlashAttentionScoreGradGetWorkspaceSize”接口获取入参并根据计算流程计算所需workspace大小，再调用“aclnnFlashAttentionScoreGrad”接口执行计算。

- **aclnnStatus aclnnFlashAttentionScoreGradGetWorkspaceSize(const aclTensor *query, const aclTensor *key, const aclTensor *value, const aclTensor *dy, const aclTensor *pseShift, const aclTensor *dropMask, const aclTensor *paddingMask, const aclTensor *attenMask, const aclTensor *softmaxMax, const aclTensor *softmaxSum, const aclTensor *softmaxIn, const aclTensor *attentionIn, const aclTensor *prefix, const aclTensor *dq, const aclTensor *dk, const aclTensor *dv, const aclTensor *dpse, double scaleValue, double keepProb, int64_t preTockens, int64_t nextTockens, int64_t headNum, string *inputLayout, int32_t innerPrecise, int64_t sparseMode, uint64_t *workspaceSize, aclOpExecutor **executor)**

- **aclnnStatus aclnnFlashAttentionScoreGrad(void *workspace, uint64_t workspaceSize, aclOpExecutor *executor, aclrtStream stream)**

功能描述

- 算子功能：训练场景下计算注意力的反向输出，即**11.13.7.5 aclnnFlashAttentionScore**的反向计算。
- 计算公式：

  已知注意力的正向计算公式为：

$$Y=Dropout\left(Softmax\left(Mask\left(\frac{QK^T+pse}{\sqrt{d}}\right)\right)\right)V$$

  为方便表达，以变量S和P表示计算公式：

$$S=Mask\left(\frac{QK^T+pse}{\sqrt{d}}\right)$$

$$P=Dropout(Softmax(S))$$

$$Y=PV$$

  则注意力的反向计算公式为：

$$dV=P^TdY$$

$$dQ=\frac{((dS)*K)}{\sqrt{d}}$$

$$dK=\frac{((dS)^T*Q)}{\sqrt{d}}$$

$$d(pse)=dS*\sqrt{d}$$

aclnnFlashAttentionGradGetWorkspaceSize

- **参数说明：**
  - query（aclTensor*，计算输入）：Device侧的aclTensor，公式中的输入Q，数据类型支持FLOAT16、BFLOAT16，数据格式支持ND。
  - key（aclTensor*，计算输入）：Device侧的aclTensor，公式中的输入K，数据类型支持FLOAT16、BFLOAT16，数据格式支持ND。
  - value（aclTensor*，计算输入）：Device侧的aclTensor，公式中的输入V，数据类型支持FLOAT16、BFLOAT16，数据格式支持ND。
  - dy（aclTensor*，计算输入）：Device侧的aclTensor，公式中的输入dY，数据类型支持、FLOAT16、BFLOAT16，数据格式支持ND。

- pseShift（aclTensor*，计算输入）：Device侧的aclTensor，公式中的输入pse，可选参数，表示位置编码。数据类型支持FLOAT16、BFLOAT16，数据格式支持ND，alibi位置编码场景下，每个batch参数不同[B N S S], 每个batch参数相同[B 1 S S]
- dropMask（aclTensor*，计算输入）：Device侧的aclTensor，可选属性，数据类型支持UINT8(标识8个1bit BOOL)，数据格式支持ND。
- paddingMask（aclTensor*，计算输入）：Device侧的aclTensor，暂不支持该传参。。
- attenMask（aclTensor*，计算输入）：Device侧的aclTensor，可选属性，代表下三角全为0上三角全为负无穷的倒三角mask矩阵，数据类型支持BOOL、UINT8，数据格式支持ND。
- softmaxMax（aclTensor*，计算输入）：Device侧的aclTensor，注意力正向计算的中间输出，数据类型支持FLOAT，数据格式支持ND。
- softmaxSum（aclTensor*，计算输入）：Device侧的aclTensor，注意力正向计算的中间输出，数据类型支持FLOAT，数据格式支持ND。
- softmaxIn（aclTensor*，计算输入）：Device侧的aclTensor，注意力正向计算的中间输出，数据类型支持FLOAT16，数据格式支持ND。
- attentionIn（aclTensor*，计算输入）：Device侧的aclTensor，注意力正向计算的最终输出attentionOut，数据类型支持FLOAT16、BFLOAT16，数据格式支持ND。
- prefix（aclTensor*，计算输入）：Device侧的aclTensor，可选属性，代表prefix稀疏计算场景每个Batch的N值。数据类型支持INT64，数据格式支持ND。
- dq（aclTensor*，计算输出）：Device侧的aclTensor，公式中的dQ，表示query的梯度，计算输出，数据类型支持FLOAT16、BFLOAT16，数据格式支持ND。
- dk（aclTensor*，计算输出）：Device侧的aclTensor，公式中的dK，表示key的梯度，计算输出，数据类型支持FLOAT16、BFLOAT16，数据格式支持ND。
- dv（aclTensor*，计算输出）：Device侧的aclTensor，公式中的dV，表示value的梯度，计算输出，数据类型支持FLOAT16、BFLOAT16，数据格式支持ND。
- dpse（aclTensor*，计算输出）：Device侧的aclTensor，公式中的d(pse)，表示pse的梯度，计算输出，数据类型支持FLOAT16、BFLOAT16，数据格式支持ND。
- scaleValue（double，计算输入）：Host侧的double，公式中d开根号的倒数，代表缩放系数，作为计算流中Muls的scalar值，数据类型支持DOUBLE。
- keepProb（double，计算输入）：Host侧的double，可选参数，代表dropMask中1的比例，数据类型支持FLOAT32。
- preTokens（int64_t，计算输入）：Host侧的int64_t，用于稀疏计算的参数，可选参数，数据类型支持INT64。
- nextTokens（int64_t，计算输入）：Host侧的int64_t，用于稀疏计算的参数，可选参数，数据类型支持INT64。
- headNum（int64_t，计算输入）：Host侧的int64_t，代表head个数，数据类型支持INT64。
- inputLayout（string*，计算输入）：Host侧的string，代表输入query、key、value的数据排布格式，支持BSH、SBH、BSND、BNSD。

说明

query、key、value数据排布格式支持从多种维度解读，其中B（Batch）表示输入样本批量大小、S（Seq-Length）表示输入样本序列长度、H（Head-Size）表示隐藏层的大小、N（Head-Num）表示多头数、D（Head-Dim）表示隐藏层最小的单元尺寸，且满足D=H/N。

- innerPrecise（int32_t，计算输入）：保留参数，暂未使用。
- sparseMode（int64_t，计算输入）：Host侧的int，表示sparse的模式。数据类型支持：INT64。
  - sparseMode为0时，代表defaultMask模式，如果attenmask未传入则不做mask操作，忽略preTokens和nextTokens(内部赋值为INT_MAX)；如果传入，则需要传入完整的attenmask矩阵（S1 * S2），表示preTokens和nextTokens之间的部分需要计算。
  - sparseMode为为1时，代表allMask，即传入完整的attenmask矩阵。。
  - sparseMode为2时，代表leftUpCausal模式的mask，对应以左顶点为划分的下三角场景，需要传入优化后的attenmask矩阵（2048*2048）。
  - sparseMode为3时，代表rightDownCausal模式的mask，对应以右下顶点为划分的下三角场景，需要传入优化后的attenmask矩阵（2048*2048）。
  - sparseMode为为4时，代表band场景，即计算preTokens和nextTokens之间的部分。
  - sparseMode为为5时，代表prefix场景，即在rightDownCasual的基础上，左侧加上一个长为S1，宽为N的矩阵，N的值由新增的输入prefix获取，且每个Batch轴的N值不一样。
  - sparseMode为为6、7、8时，分别代表global、dilated、block_local，均暂不支持。

  用户不特意指定时可传入默认值0。**注意：**当所有的attenmask的shape小于2048且相同的时候，建议使用default模式，来减少内存使用量；sparse_mode配置为2或3时，不能配置preTokens、nextTokens。
- workspaceSize（uint64_t*，出参）：返回用户需要在Device侧申请的workspace大小。
- executor（aclOpExecutor**，出参）：返回op执行器，包含了算子计算流程。

- **返回值：**

  返回aclnnStatus状态码，具体参见**11.13.7.10 aclnn返回码**。

  说明

  第一段接口完成入参校验，若出现以下错误码，则对应原因为：

  - 返回161001（ACLNN_ERR_PARAM_NULLPTR）：如果传入参数是必选输入、输出或者必选属性，且是空指针，则返回161001。
  - 返回161002（ACLNN_ERR_PARAM_INVALID）：query、key、value、dy、pseShift、dropMask、paddingMask、attenMask、softmaxMax、softmaxSum、softmaxIn、attentionIn、dq、dk、dv的数据类型和数据格式不在支持的范围内。

aclnnFlashAttentionScoreGrad

- **参数说明：**
  - workspace（void*，入参）：在Device侧申请的workspace内存起址。
  - workspaceSize（uint64_t，入参）：在Device侧申请的workspace大小，由**aclnnFlashAttentionGradGetWorkspaceSize**获取。
  - executor（aclOpExecutor*，入参）：op执行器，包含了算子计算流程。
  - stream（aclrtStream，入参）：指定执行任务的AscendCL stream流。
- **返回值：**

  返回aclnnStatus状态码，具体参见**11.13.7.10 aclnn返回码**。

约束与限制

关于数据shape的约束，以inputLayout的BSND、BNSD为例（BSH、SBH下H=N*D），其中

- B：取值范围为1~256。
- N：取值范围为1~256。
- S：取值范围为1~32K，且为16的倍数。
- D：取值为64、96、128、256。

11.13.7.7 aclnnFFN

接口原型

每个算子分为两段式接口，必须先调用“aclnnFFNGetWorkspaceSize”接口获取入参并根据计算流程计算所需workspace大小，再调用“aclnnFFN”接口执行计算。

- **aclnnStatus aclnnFFNGetWorkspaceSize(const aclTensor *x, const aclTensor *weight1, const aclTensor *weight2, const aclIntArray *expertTokens, const aclTensor *bias1, const aclTensor *bias2, const aclTensor *scale, const aclTensor *offset, const aclTensor *deqScale1, const aclTensor *deqScale2, char *activation, int64_t innerPrecise, const aclTensor *y, uint64_t *workspaceSize, aclOpExecutor **executor)**
- **aclnnStatus aclnnFFN(void *workspace, uint64_t workspaceSize, aclOpExecutor *executor, aclrtStream stream)**

功能描述

- 算子功能：该FFN算子兼容MoeFFN和FFN。该算子在没有专家分组（expert_tokens为空）时是FFN，有专家分组时是MoeFFN，统称为FFN，属于Moe结构。MoE（Mixture-of-Experts，混合专家系统）是一种用于训练万亿参数量级模型的技术。MoE将预测建模任务分解为若干子任务，在每个子任务上训练一个专家模型（Expert Model），开发一个门控模型（Gating Model），该模型根据要预测的输入来学习信任哪个专家，最终综合多个专家计算结果作为预测结果。
- 使用约束：**当前仅支持expert_tokens不为空的场景，即仅支持有专家分组的MoeFFN。**
- 计算公式：FFN类计算公式为

$$y=activation(xW1+b1)W2+b2$$

aclnnFFNGetWorkspaceSize

- **参数说明：**
  - x（aclTensor*，计算输入）：必选，Device侧的aclTensor，公式中的输入x，数据类型支持FLOAT16、INT8，数据格式支持ND，支持输入为2维~8维。
  - weight1（aclTensor*，计算输入）：必选，Device侧的aclTensor，专家的评估数据，公式中的W1，数据类型支持FLOAT16、INT8，数据格式支持ND，输入在有/无专家时分别为3维/2维。
  - weight2（aclTensor*，计算输入）：必选，Device侧的aclTensor，专家的评估数据，公式中的W2，数据类型支持FLOAT16、INT8，数据格式支持ND，输入在有/无专家时分别为3维/2维。
  - expertTokens（aclIntArray*，计算输入）：可选，Host侧的aclIntArray类型，代表各专家的评估权重，数据类型支持INT64，数据格式支持ND，若不为空时可支持的最大长度为256个。
  - bias1（aclTensor*，计算输入）：可选，Device侧的aclTensor，评估数据修正值，公式中的b1，数据类型支持FLOAT16、INT32，数据格式支持ND，输入在有/无专家时分别为2维/1维。
  - bias2（aclTensor*，计算输入）：可选，Device侧的aclTensor，评估数据修正值，公式中的b2，数据类型支持FLOAT16、INT32，数据格式支持ND，输入在有/无专家时分别为2维/1维。
  - scale（aclTensor*，计算输入）：可选，Device侧的aclTensor，量化参数，可选参数，数据类型支持FLOAT，数据格式支持ND，一维向量，输入元素个数在有/无专家时分别为expertTokens长度/1个。
  - offset（aclTensor*，计算输入）：可选，Device侧的aclTensor，量化参数，可选参数，数据类型支持FLOAT，数据格式支持ND，一维向量，输入元素个数在有/无专家时分别为expertTokens长度/1个。
  - deqScale1（aclTensor*，计算输入）：可选，Device侧的aclTensor，第一个分组matmul的反量化参数，数据类型支持UINT64，数据格式支持ND，输入在有/无专家时分别为2维/1维。
  - deqScale2（aclTensor*，计算输入）：可选，Device侧的aclTensor，第二个分组matmul的反量化参数，数据类型支持UINT64，数据格式支持ND，输入在有/无专家时分别为2维/1维。
  - activation（char*，计算输入）：必选，Host侧的属性值，代表使用的激活函数，公式中的activation，数据类型支持string，当前支持fastgelu/gelu/relu/silu。
  - innerPrecise（int64_t，计算输入）：可选，Host侧的属性值，代表内部精度模式，数据类型支持int，0和1分别代表高精度和高性能，默认值为高精度，当前仅支持高性能。
  - y（aclTensor*，计算输出）：Device侧的aclTensor，公式中的输出y，数据类型支持FLOAT16，数据格式支持ND。
  - workspaceSize（uint64_t*，出参）：返回用户需要在Device侧申请的workspace大小。
  - executor（aclOpExecutor**，出参）：返回op执行器，包含了算子计算流程。

- **返回值：**

  返回aclnnStatus状态码，具体参见**11.13.7.10 aclnn返回码**。

  说明

  第一段接口完成入参校验，若出现以下错误码，则对应原因为：

  - 返回161001（ACLNN_ERR_PARAM_NULLPTR）：如果传入参数是必选输入、输出或者必选属性，且是空指针，则返回161001。
  - 返回161002（ACLNN_ERR_PARAM_INVALID）：x、weight1、weight2、expertTokens、bias1、bias2、y的数据类型和数据格式不在支持的范围内。

aclnnFFN

- **参数说明：**
  - workspace（void*，入参）：在Device侧申请的workspace内存起址。
  - workspaceSize（uint64_t，入参）：在Device侧申请的workspace大小，由**aclnnFFNGetWorkspaceSize**获取。
  - executor（aclOpExecutor*，入参）：op执行器，包含了算子计算流程。
  - stream（aclrtStream，入参）：指定执行任务的AscendCL stream流。
- **返回值：**

  返回aclnnStatus状态码，具体参见**11.13.7.10 aclnn返回码**。

约束与限制

- 有专家时，专家数据的总数需要与BS（其中B表示输入样本批量大小、S表示输入样本序列长度）长度保持一致。
- 专家个数不超过256个。

11.13.7.8 aclnnMoeGatingTopKSoftmax

该接口为试验版本，当前不支持应用于商用产品中，后续版本会作为正式功能更新发布。

接口原型

每个算子分为两段式接口，必须先调用“aclnnMoeGatingTopKSoftmaxGetWorkspaceSize”接口获取入参并根据计算流程计算所需workspace大小，再调用“aclnnMoeGatingTopKSoftmax”接口执行计算。

- **aclnnStatus aclnnMoeGatingTopKSoftmaxGetWorkspaceSize(const aclTensor *x, const aclTensor *finishedOptional, int64_t k, const aclTensor *yOut, const aclTensor *expertIdxOut, const aclTensor *rowIdxOut, uint64_t *workspaceSize, aclOpExecutor **executor)**
- **aclnnStatus aclnnMoeGatingTopKSoftmax(void* workspace, uint64_t workspaceSize, aclOpExecutor* executor, const aclrtStream stream)**

功能描述

- 算子功能：MoE计算中，对x的输出做Softmax计算，取topk操作。其中yOut为softmax的topk结果；expertIdxOut为topk的indices结果即对应的专家序号；rowIdxOut为与expertIdxOu相同shape的列取值结果。如果对应的行finished为True，则expert序号直接填num_expert值（即x的最后一个轴大小）。

- 计算公式：

$$softmaxOut = softmax(x, axis)=-1$$

$$yOut, expertIdxOut = topK(softmaxOut, k=k)$$

$$rowIdxOut = arange(expertIdxOut.shape)[0] * expertIdxOut.shape[1].reshape([expertIdxOut.shape[1], expertIdxOut.shape[0]]).transpose(1, 0)$$

aclnnMoeGatingTopKSoftmaxGetWorkspaceSize

- **参数说明：**
  - x（aclTensor*，计算输入）：待计算的输入，要求是一个2D/3D的Tensor，数据类型支持FLOAT16、BFLOAT16、FLOAT32，数据格式要求为ND。
  - finishedOptional（aclTensor*，可选计算输入）：要求是一个1D/2D的Tensor，数据类型支持bool，shape为x_shape[:-1]，数据格式要求为ND。
  - k（int64_t，计算输入）：topk的k值，大小为0 <= k <= gating的-1轴大小。
  - yOut（aclTensor*, 计算输出）：对x做softmax后取的topk值，要求是一个2D/3D的Tensor，数据类型与x需要保持一致，其非-1轴要求与x的对应轴大小一致，其-1轴要求其大小同k值。数据格式要求为ND。
  - expertIdxOut（aclTensor*, 计算输出）：对x做softmax后取topk值的索引，即专家的序号。shape要求与yOut一致。数据类型支持int32，数据格式要求为ND。
  - rowIdxOut（aclTensor*, 计算输出）：指示每个位置对应的原始行位置（见示例），shape要求与yOut一致。数据类型支持int32，数据格式要求为ND。
  - workspaceSize（uint64_t*，出参）：Device侧的整型，返回需要在Device侧申请的workspace大小。
  - executor（aclOpExecutor**，出参）：Device侧的aclOpExecutor，返回op执行器，包含了算子计算流程。
- **返回值：**

  返回aclnnStatus状态码，具体参见**11.13.7.10 aclnn返回码**。

aclnnMoeGatingTopKSoftmax

- **参数说明：**
  - workspace（void*，入参）：在Device侧申请的workspace内存起址。
  - workspaceSize（uint64_t，入参）：在Device侧申请的workspace大小，由**aclnnMoeGatingTopKSoftmaxGetWorkspaceSize**获取。
  - executor（aclOpExecutor*，入参）：op执行器，包含了算子计算流程。
  - stream（aclrtStream，入参）：指定执行任务的AscendCL stream流。
- **返回值：**

  返回aclnnStatus状态码，具体参见**11.13.7.10 aclnn返回码**。

约束与限制

专家个数不超过1024个。

调用示例

```
constexpr int opt_case_id = 1000;
constexpr int opt_op_type = 1001;

//通用参数
constexpr int op_gating_dim0 = 1011;
constexpr int op_gating_dim1 = 1012;
constexpr int op_finished_dim0 = 1013;
constexpr int op_device_id = 1014;

constexpr int op_tensor_format = 1050;
constexpr int op_tensor_type = 1051;
constexpr int op_thread_num = 1052;
constexpr int op_input_path = 1053;
constexpr int op_output_path = 1054;
constexpr int op_top_k = 1055;
constexpr int op_num_experts=1056;
constexpr int op_device_num = 1057;
constexpr int op_sat_mode = 1058;

const char * tmp_err_msg = NULL;
//其他参数
constexpr int op_loop_times = 10;
constexpr int DEVICE_NUM_MAX = 8;

//全局变量定义
int64_t gCaseId;
std::string gOpType;
int64_t g_gating_dim0 = 1;
int64_t g_gating_dim1 = 1;
int64_t g_finished_dim0 = 1024;
int64_t g_poolSize = 1;
int64_t g_loop_times = 1;
int64_t g_device_num = 1;
int64_t g_device_id = 0;
int64_t g_thread_num = 1;
int64_t g_topk = 2;
int64_t g_num_experts =16;
uint64_t g_fisrt = 0;
uint64_t g_last =0;
int g_sat_mode = 0;

std::string g_str_input_format;
std::string g_str_output_format;
std::string g_input_path;
std::string g_output_path;

aclDataType g_tensor_type[16]={aclDataType::ACL_DT_UNDEFINED};
aclFormat   g_tensor_format[16] ={ACL_FORMAT_ND};

std::map<std::string,aclDataType>g_data_type_map =
{
{"FLOAT"    , ACL_FLOAT},
{"FP16"        , ACL_FLOAT16},
{"INT8"     , ACL_INT8},
{"INT16"    , ACL_INT16},
{"UINT16"    , ACL_UINT16},
{"UINT8"    , ACL_UINT8},
{"INT32"    , ACL_INT32},
{"INT64"    , ACL_INT64},
{"UINT32"    , ACL_UINT32},
{"BOOL"     , ACL_BOOL},
{"DOUBLE"    , ACL_DOUBLE},
{"BF16"     , ACL_BF16},
{"UNDEFINED", ACL_DT_UNDEFINED}
};


int Init(int32_t deviceId, aclrtContext* context, aclrtStream* stream) {
```

```
  // 固定写法，acl初始化
  auto ret = aclInit(nullptr); // profiling需要aclint,如果不需要profiling则不需要此步骤
  CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclInit failed. ERROR: %d\n", ret); return ret);
  ret = aclrtSetDevice(deviceId);
  CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtSetDevice failed. ERROR: %d\n", ret); return ret);
  ret = aclrtCreateContext(context, deviceId);
  CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtCreateContext failed. ERROR: %d\n", ret); return ret);
  ret = aclrtSetCurrentContext(*context);
  CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtSetCurrentContext failed. ERROR: %d\n", ret); return
ret);
  ret = aclrtCreateStream(stream);
  CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtCreateStream failed. ERROR: %d\n", ret); return ret);
  return 0;
}

template <typename T>
int CreateAclTensor(const std::vector<T>& hostData, const std::vector<int64_t>& shape, void** deviceAddr,
                    aclDataType dataType, aclTensor** tensor, aclFormat tensorFormat) {
  auto size = GetShapeSize(shape) * sizeof(T);
  printf("Alloc size [%zu] for tensor\n", size);
  // 调用aclrtMalloc申请device侧内存
  auto ret = aclrtMalloc(deviceAddr, size, ACL_MEM_MALLOC_HUGE_FIRST);
  CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtMalloc failed. ERROR: %d\n", ret); return ret);
  // 调用aclrtMemcpy将host侧数据拷贝到device侧内存上
  ret = aclrtMemcpy(*deviceAddr, size, hostData.data(), size, ACL_MEMCPY_HOST_TO_DEVICE);
  CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtMemcpy failed. ERROR: %d\n", ret); return ret);
  // 计算连续tensor的strides
  std::vector<int64_t> strides(shape.size(), 1);
  for (int64_t i = shape.size() - 2; i >= 0; i--)
  {
    strides[i] = shape[i + 1] * strides[i + 1];
  }
  // 调用aclCreateTensor接口创建aclTensor
  *tensor = aclCreateTensor(shape.data(), shape.size(), dataType, strides.data(), 0, tensorFormat,
                            shape.data(), shape.size(), *deviceAddr);
  return 0;
}

void FreeDeviceMem(void* addr)
{
    if (addr != nullptr)
    {
        aclrtFree(addr);
    }
}

void DestroyAclTensor(aclTensor* tensor, void* addr)
{
    FreeDeviceMem(addr);
    aclDestroyTensor(tensor);
    tensor = nullptr;
}

void MoEGatingTopKSoftmax(void *args)
{
    ThreadData *threadData;
    threadData = (ThreadData *) args;
    int threadId = threadData->threadId;
    int device_id = threadData->device_id;
    auto tensors = threadData->tensors;
    auto out_size = threadData->tensor_size[2];
    auto indicesOut_size = threadData->tensor_size[3];
    auto sourceRowOut_size = threadData->tensor_size[4];
    auto dev_mem = threadData->dev_mem;
    auto expert_tokens = threadData->expert_tokens;

    aclrtContext context = threadData->context;
    aclrtStream stream;
    auto ret = aclrtSetCurrentContext(context);
```

```
    CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("threadId %d aclrtSetCurrentContext failed. ERROR: %d
\n",threadId, ret); return ;);

    ret= aclrtSetDeviceSatMode((aclrtFloatOverflowMode)g_sat_mode);
    CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("threadId %d aclrtSetDeviceSatMode failed. ERROR: %d
\n",threadId, ret); return ;);

    ret = aclrtCreateStream(&stream);
    CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("threadId %d aclrtCreateStream failed. ERROR: %d
\n",threadId, ret); return ;);

    aclOpExecutor *executor;
    uint64_t workspaceSize;
    aclrtMemAttr attr = ACL_HBM_MEM;

    size_t mem_free;
    size_t mem_total;
    size_t aa = 1024;
    for(int i = 0;i < g_loop_times; i++)
    {
       ret = aclnnMoeGatingTopKSoftmaxGetWorkspaceSize(tensors[0], tensors[1], g_topk, tensors[2],
tensors[3], tensors[4], &workspaceSize, &executor);
       CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclnnMoeGatingTopKSoftmaxGetWorkspaceSize failed.
ERROR: %d\n", ret); return ;);
       printf("aclnnMoeGatingTopKSoftmaxGetWorkspaceSize workspace size is %ld \n",workspaceSize);

       void *workspace = NULL;
       if (workspaceSize != 0)
       {
           ret = aclrtMalloc(&workspace, workspaceSize, ACL_MEM_MALLOC_HUGE_FIRST);
           CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtMalloc failed. ERROR: %d\n", ret);
return ;);
       }

       ret = aclnnMoeGatingTopKSoftmax(&workspace, workspaceSize, executor, stream);
       CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclnnMoEGatingTopKSoftmax failed. ERROR: %d\n",
ret); return ;);

       ret = aclrtSynchronizeStream(stream);
       if( i == g_loop_times -1 )
       {
          ret = aclrtGetMemInfo(attr, &mem_free, &mem_total);
          size_t mem_used_after = (mem_total-mem_free)/aa/aa;
          printf(" after run : free :%ld M,  total:%ld M, used :%ld M, ret :%d \n", mem_free/aa/aa,
mem_total/aa/aa, mem_used_after, ret);
          g_last = mem_free/aa/aa;
          printf(" op cost memory :%ld M \n", g_fisrt-g_last);
       }

       tmp_err_msg = aclGetRecentErrMsg();
       if(tmp_err_msg !=NULL)
       { printf(" ERROR Message : %s \n " ,tmp_err_msg);}
       CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtSynchronizeStream failed. ERROR: %d\n", ret);
return ;);
       if(workspaceSize != 0)
       {
         aclrtFree(workspace);
       }
    }

    tmp_err_msg = aclGetRecentErrMsg();
    if(tmp_err_msg != NULL)
    {
      printf(" ERROR Message : %s \n " ,tmp_err_msg);
    }

    void *host_data1 = malloc(out_size);
    aclrtMemcpy(host_data1, out_size, dev_mem[2], out_size, ACL_MEMCPY_DEVICE_TO_HOST);
```

```
    WriteFile(g_output_path +"_"+ std::to_string(device_id) + "_"+std::to_string(threadId) + "out.bin",
host_data1, out_size);
    free(host_data1);

    void *host_data2 = malloc(indicesOut_size);
    aclrtMemcpy(host_data2, indicesOut_size, dev_mem[3], indicesOut_size,
ACL_MEMCPY_DEVICE_TO_HOST);
    WriteFile(g_output_path +"_"+ std::to_string(device_id) + "_"+std::to_string(threadId) + "indices_out.bin",
host_data2, indicesOut_size);
    free(host_data2);

    void *host_data3 = malloc(sourceRowOut_size);
    aclrtMemcpy(host_data3, sourceRowOut_size, dev_mem[4], sourceRowOut_size,
ACL_MEMCPY_DEVICE_TO_HOST);
    WriteFile(g_output_path +"_"+ std::to_string(device_id) + "_"+std::to_string(threadId) +
"source_row_out.bin", host_data3, sourceRowOut_size);
    free(host_data3);

    ret = aclrtDestroyStream(stream);
    CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("threadId %d aclrtDestroyStream failed. ERROR: %d
\n",threadId, ret); return ;);
}

struct option long_options[] =
{
     // 图像类参数
     {"case_id", 1, nullptr, opt_case_id},
     {"op_type", 1, nullptr, opt_op_type},
     {"gating_dim0", 1, nullptr, op_gating_dim0},
     {"gating_dim1", 1, nullptr, op_gating_dim1},
     {"finished_dim0", 1, nullptr, op_finished_dim0},
     {"num_experts",1, nullptr, op_num_experts},
     {"thread_num", 1, nullptr, op_thread_num},
     {"device_num", 1, nullptr, op_device_num},
     {"device_id", 1, nullptr, op_device_id},
     {"tensor_type", 1, nullptr, op_tensor_type},
     {"loop", 1, nullptr, op_loop_times},
     {"tensor_format",1, nullptr, op_tensor_format},
     {"topK", 1, nullptr, op_top_k},
     {"input_path", 1, nullptr, op_input_path},
     {"output_path", 1, nullptr, op_output_path},
     {"sat_mode", 1, nullptr, op_sat_mode},
};

bool is_argv_in_options(std::string &argv_str)
{
    for (int option_index = 0; option_index < sizeof(long_options) / sizeof(struct option); option_index++) {
        std::string option_str = std::string("--") + std::string(long_options[option_index].name);
        if (argv_str == option_str) {
            return true;
        }
    }
    return false;
}

int32_t check_argv(int argc, char *argv[])
{
    for (int argc_index = 0; argc_index < argc; argc_index++)
    {
        std::string argv_str(argv[argc_index]);
        if (argv_str.find(std::string("--")) != std::string::npos)
        {
            bool find_flag = is_argv_in_options(argv_str);
            if (find_flag == false) {
                printf("argv:%s not support.\n", argv_str.c_str());
                return -1;
            }
        }
    }
```

```
    return 0;
}

int32_t get_option(int argc, char **argv) {

    if (check_argv(argc, argv) != 0) {
        return -1;
    }
    while (1) {
        int32_t option_index = 0;
        int32_t option_value  = getopt_long(argc, argv, "", long_options, &option_index);
        if (option_value  == -1)
        {
            break;
        }
        switch (option_value) {
            case opt_case_id:
                gCaseId = atoi(optarg);
                break;
            case opt_op_type:
                gOpType = optarg;
                break;
            case op_gating_dim0:
                g_gating_dim0 = atoi(optarg);
                break;
            case op_gating_dim1:
                g_gating_dim1 = atoi(optarg);
                break;
            case op_finished_dim0:
                g_finished_dim0 = atoi(optarg);
                break;
            case op_thread_num:
                g_thread_num = atoi(optarg);
                break;
            case op_loop_times:
                g_loop_times = atoi(optarg);
                break;
            case op_device_num:
                g_device_num = atoi(optarg);
                break;
            case op_input_path:
                g_input_path = optarg;
                break;
            case op_output_path:
                g_output_path = optarg;
                break;
            case op_device_id:
                g_device_id = atoi(optarg);
                break;
            case op_top_k:
                g_topk = atoi(optarg);
                break;
            case op_tensor_type:
                {
                  std::vector<std::string> tokens;
                  split(optarg,tokens,':');
                  for(int i = 0;i<tokens.size();i++)
                  {
                    g_tensor_type[i] = g_data_type_map[tokens[i]];
                    printf(" token is %s, tensor_type is %d \n ",tokens[i].c_str(),g_tensor_type[i]);
                  }
                }
            break;
            case op_tensor_format:
                {
                  std::vector<std::string> tokens;
                  split(optarg,tokens,':');
                  for(int i = 0;i<tokens.size();i++)
                  {
```

```
                g_tensor_format[i] =(aclFormat)atoi(tokens[i].c_str());
              }
            }
        break;
        case op_num_experts:
            g_num_experts = atoi(optarg);
            break;
        case op_sat_mode:
            g_sat_mode = atoi(optarg);
            break;
        default:
            printf("invalid para %d", option_value);
            break;
    }
  }
  return 0;
}


int main(int argc, char **argv)
{
  get_option(argc,&(*argv));
  aclError ret;
  ret = aclInit(NULL);
  std::vector<std::thread>vec_thread;
  aclTensor *v_tensors[DEVICE_NUM_MAX][g_thread_num][15];
  void *v_dev_mem[DEVICE_NUM_MAX][g_thread_num][15];
  size_t v_tensors_size[DEVICE_NUM_MAX][g_thread_num][15]={0};
  ThreadData threadData[DEVICE_NUM_MAX][g_thread_num];

  std::vector<aclrtContext> vec_context;
  int64_t shape1[] =  {g_gating_dim0,g_gating_dim1}; //gating
  int64_t shape2[] =  {g_finished_dim0};//finishedOptional
  int64_t shape3[] =  {g_gating_dim0,g_topk};//out
  int64_t shape4[] =  {g_gating_dim0,g_topk};//indicesOut
  int64_t shape5[] =  {g_gating_dim0,g_topk};//sourceRowOut

  tensor_info tensor_desc[]=
  {
      {shape1,2,g_tensor_type[0],g_tensor_format[0]},
      {shape2,1,g_tensor_type[1],g_tensor_format[1]},
      {shape3,2,g_tensor_type[2],g_tensor_format[2]},
      {shape4,2,g_tensor_type[3],g_tensor_format[3]},
      {shape5,2,g_tensor_type[4],g_tensor_format[4]},
  };

  std::string bin_path[] = {"gating.bin", "finished.bin"};

  if(g_device_num == 1)//指定某一个device上运行
  {
      ret = aclrtSetDevice(g_device_id);
      CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtSetDevice failed. ERROR: %d\n", ret); return ret);
      aclrtMemAttr attr = ACL_HBM_MEM;
      size_t mem_free;
      size_t mem_total;
      size_t aa = 1024;
      ret = aclrtGetMemInfo(attr, &mem_free, &mem_total);
      size_t mem_used_after = (mem_total-mem_free)/aa/aa;
      printf(" after run : free :%ld M,  total:%ld M, used :%ld M, ret :%d \n", mem_free/aa/aa,
mem_total/aa/aa, mem_used_after, ret);
      g_fisrt = mem_free/aa/aa;

      aclrtContext context;
      ret = (aclrtCreateContext(&context, g_device_id) != ACL_SUCCESS);
      CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtCreateContext  failed. ERROR: %d\n", ret); return
ret);


      for(int threadId = 0;threadId< g_thread_num; threadId++)
```

```
        {
            int len = sizeof(tensor_desc)/sizeof(tensor_info);
            printf("len is %d \n",len);
            for(auto i = 0; i < len ;i++)
            {
                uint8_t *host;
                void *data;
                struct tensor_info *info = &(tensor_desc[i]);
                size_t size = get_data_size(info);
                std::cout<<"size :"<<size<<std::endl;
                if (size == 0)
                {
                    v_tensors[g_device_id][threadId][i] = NULL;
                    v_dev_mem[g_device_id][threadId][i] = NULL;
                    continue;
                }

                if(aclrtMallocHost((void**)(&host), size)!= ACL_SUCCESS)
                {
                    printf("Malloc host memory failed, error message is:\n%s\n", aclGetRecentErrMsg());
                   return -1;
                }

                if (aclrtMalloc(&data, size, ACL_MEM_MALLOC_HUGE_FIRST) != ACL_SUCCESS)
                {
                    printf("Malloc device memory failed, error message is:\n%s\n", aclGetRecentErrMsg());
                    return -1;
                }
                v_tensors_size[g_device_id][threadId][i] = size;
               if(i<2)
               {
                    ReadFile(g_input_path +"/"+ bin_path[i], size, host, size);
                    if(aclrtMemcpy(data, size, host, size, ACL_MEMCPY_HOST_TO_DEVICE) !=ACL_SUCCESS)
                    {
                     printf("Malloc device memory failed, error message is:\n%s\n", aclGetRecentErrMsg());
                     return -1;
                    }
               }

               v_dev_mem[g_device_id][threadId][i] = data;
               v_tensors[g_device_id][threadId][i] = aclCreateTensor(info->dims, info->dim_cnt, info->dtype,
NULL, 0, info->fmt, info->dims, info->dim_cnt, data);
            }
            threadData[g_device_id][threadId].threadId = threadId;
            threadData[g_device_id][threadId].device_id = g_device_id;
            for(int num= 0 ;num < len; num ++)
            {
              threadData[g_device_id][threadId].tensors[num]  = v_tensors[g_device_id][threadId][num];
              threadData[g_device_id][threadId].tensor_size[num] = v_tensors_size[g_device_id][threadId][num];
              threadData[g_device_id][threadId].dev_mem[num] =  v_dev_mem[g_device_id][threadId][num];
            }
            threadData[g_device_id][threadId].context = context;
           vec_thread.emplace_back(MoEGatingTopKSoftmax, &threadData[g_device_id][threadId]);
        }
    }
    else
    {
        for(int deviceId = 0;deviceId < g_device_num; deviceId++ )
        {
            auto ret = aclrtSetDevice(deviceId);
            CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtSetDevice failed. ERROR: %d\n", ret); return
ret);
            aclrtContext context;
            ret = ( aclrtCreateContext(&context, deviceId) != ACL_SUCCESS);
            CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtCreateContext failed. ERROR: %d\n", ret);
return ret);
            for(int threadId = 0;threadId< g_thread_num; threadId++)
            {
                int len = sizeof(tensor_desc)/sizeof(tensor_info);
```

```
            printf("len is %d \n",len);
            for(auto i = 0; i < len ;i++)
            {
            uint8_t *host;
            void *data;
            struct tensor_info *info = &(tensor_desc[i]);
            size_t size = get_data_size(info);
            std::cout<<"size :"<<size<<std::endl;
            if (size == 0)
            {
               v_tensors[deviceId][threadId][i] = NULL;
               v_dev_mem[deviceId][threadId][i] = NULL;
               continue;
            }
            if(aclrtMallocHost((void**)(&host), size)!= ACL_SUCCESS)
            {
               printf("Malloc host memory failed, error message is:\n%s\n", aclGetRecentErrMsg());
              return -1;
            }

            if (aclrtMalloc(&data, size, ACL_MEM_MALLOC_HUGE_FIRST) != ACL_SUCCESS)
            {
               printf("Malloc device memory failed, error message is:\n%s\n", aclGetRecentErrMsg());
               return -1;
            }
            v_tensors_size[deviceId][threadId][i] = size;
             if(i<2)
             {
               ReadFile(g_input_path +"/"+ bin_path[i], size, host, size);
               if(aclrtMemcpy(data, size, host, size, ACL_MEMCPY_HOST_TO_DEVICE) !=ACL_SUCCESS)
               {
                 printf("Malloc device memory failed, error message is:\n%s\n", aclGetRecentErrMsg());
                 return -1;
               }
             }

           v_dev_mem[deviceId][threadId][i] = data;
           v_tensors[deviceId][threadId][i] = aclCreateTensor(info->dims, info->dim_cnt, info->dtype, NULL,
0, info->fmt, info->dims, info->dim_cnt, data);
           }
          threadData[deviceId][threadId].threadId = threadId;
          threadData[deviceId][threadId].device_id = deviceId;
         for(int num = 0;num<len;num++)
         {
          threadData[deviceId][threadId].tensors[num]  = v_tensors[deviceId][threadId][num];
          threadData[deviceId][threadId].tensor_size[num] = v_tensors_size[deviceId][threadId][num];
          threadData[deviceId][threadId].dev_mem[num] =  v_dev_mem[deviceId][threadId][num];
         }
          threadData[deviceId][threadId].context = context;
          vec_thread.emplace_back(MoEGatingTopKSoftmax, &threadData[deviceId][threadId]);
        }
      }

   }
   //资源初始化
   for (auto &tmp:vec_thread)
   {
       if (tmp.joinable())
       {
          tmp.join();
       }
   }

   if(g_device_num == 1)
   {
        ret = aclrtResetDevice(g_device_id);
   }
   else
   {
```

```
        for(int i = 0;i < g_device_num;i++ )
        {
            ret = aclrtResetDevice(i);
        }
    }

    (void) aclFinalize();

}
```

11.13.7.9 aclnnMoeInitRouting

该接口为试验版本，当前不支持应用于商用产品中，后续版本会作为正式功能更新发布。

接口原型

每个算子分为两段式接口，必须先调用 “aclnnMoeInitRoutingGetWorkspaceSize” 接口获取入参并根据计算流程计算所需workspace大小，再调用 “aclnnMoeInitRoutingn” 接口执行计算。

- **aclnnStatus aclnnMoeInitRoutingGetWorkspaceSize(const aclTensor *x, const aclTensor *rowIdx, const aclTensor *expertIdx, int64_t activeNum, const aclTensor *expandedXOut, const aclTensor *expandedRowIdxOut, const aclTensor *expandedExpertIdxOut, uint64_t *workspaceSize, aclOpExecutor **executor)**
- **aclnnStatus aclnnMoeInitRouting(void *workspace, uint64_t workspaceSize, aclOpExecutor *executor, aclrtStream stream)**

功能描述

- **算子功能：**MoE的routing计算，根据**11.13.7.8 aclnnMoeGatingTopKSoftmax**的计算结果做routing处理。
- **计算公式：**

$$expandedExpertIdx,sortedRowIdx=keyValueSort(expertIdx,rowIdx)$$

$$expandedRowIdx[sortedRowIdx[i]]=i$$

$$expandedX[i]=x[sortedRowIdx[i]\%numRows]$$

aclnnMoeInitRoutingGetWorkspaceSize

- **参数说明：**
  - x（aclTensor*，计算输入）：MOE的输入即token特征输入，要求为一个2D的Tensor，shape为 (NUM_ROWS, H)。数据类型支持FLOAT16、BFLOAT16、FLOAT32，数据格式要求为ND。
  - rowIdx（aclTensor*，计算输入）：指示每个位置对应的原始行位置，shape要求与expertForSourceRow 一致。数据类型支持int32，数据格式要求为ND。

- expertIdx（aclTensor*，计算输入）：**11.13.7.8 aclnnMoeGatingTopKSoftmax**的输出每一行特征对应的K个处理专家，要求是一个2D的shape (NUM_ROWS, K)。数据类型支持int32，数据格式要求为ND。
- activeNum（int64_t，计算输入）：表示总的最大处理row数，expandedXOut只有这么多行是有效的。
- expandedXOut（aclTensor*，计算输出）：根据expertIdx进行扩展过的特征，要求是一个2D的Tensor，shape (min(NUM_ROWS, activeNum) * k, H)。数据类型同x, 数据格式要求为ND。
- expandedRowIdxOut（aclTensor*，计算输出）：expandedX和x的映射关系， 要求是一个1D的Tensor，Shape为(NUM_ROWS*K,)，数据类型支持int32，数据格式要求为ND。
- expandedExpertIdxOut（aclTensor*，计算输出）：输出expertIdx排序后的结果。
- workspaceSize（uint64_t*，出参）：返回用户需要在Device侧申请的workspace大小。
- executor（aclOpExecutor**，出参）：返回op执行器，包含了算子计算流程。

- **返回值**

  返回aclnnStatus状态码，具体参见**11.13.7.10 aclnn返回码**。

aclnnMoeInitRouting

- **参数说明：**
  - workspace（void*，入参）：在Device侧申请的workspace内存起址。
  - workspaceSize（uint64_t，入参）：在Device侧申请的workspace大小，由**aclnnMoeInitRoutingGetWorkspaceSize**获取。
  - executor（aclOpExecutor*，入参）：op执行器，包含了算子计算流程。
  - stream（aclrtStream，入参）：指定执行任务的AscendCL stream流。
- **返回值：**

  返回aclnnStatus状态码，具体参见**11.13.7.10 aclnn返回码**。

约束与限制

x的shape大小需要小于2^24。

调用示例

```
#include "acl/acl.h"
#include "aclnnop/aclnn_moe_init_routing.h"
#include <iostream>
#include <vector>
#define CHECK_RET(cond, return_expr) \
  do {                              \
    if (!(cond)) {                  \
      return_expr;                  \
    }                               \
  } while (0)
#define LOG_PRINT(message, ...)     \
  do {                              \
    printf(message, ##__VA_ARGS__); \
  } while (0)
```

```
int64_t GetShapeSize(const std::vector<int64_t>& shape) {
    int64_t shape_size = 1;
    for (auto i : shape) {
        shape_size *= i;
    }
    return shape_size;
}
int Init(int32_t deviceId, aclrtContext* context, aclrtStream* stream) {
    // 固定写法，acl初始化
    auto ret = aclInit(nullptr);
    CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclInit failed. ERROR: %d\n", ret); return ret);
    ret = aclrtSetDevice(deviceId);
    CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtSetDevice failed. ERROR: %d\n", ret); return ret);
    ret = aclrtCreateContext(context, deviceId);
    CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtCreateContext failed. ERROR: %d\n", ret); return ret);
    ret = aclrtSetCurrentContext(*context);
    CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtSetCurrentContext failed. ERROR: %d\n", ret); return
ret);
    ret = aclrtCreateStream(stream);
    CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtCreateStream failed. ERROR: %d\n", ret); return ret);
    return 0;
}
template <typename T>
int CreateAclTensor(const std::vector<T>& hostData, const std::vector<int64_t>& shape, void** deviceAddr,
                    aclDataType dataType, aclTensor** tensor) {
    auto size = GetShapeSize(shape) * sizeof(T);
    // 调用aclrtMalloc申请device侧内存
    auto ret = aclrtMalloc(deviceAddr, size, ACL_MEM_MALLOC_HUGE_FIRST);
    CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtMalloc failed. ERROR: %d\n", ret); return ret);
    // 调用aclrtMemcpy将host侧数据拷贝到device侧内存上
    ret = aclrtMemcpy(*deviceAddr, size, hostData.data(), size, ACL_MEMCPY_HOST_TO_DEVICE);
    CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtMemcpy failed. ERROR: %d\n", ret); return ret);
    // 计算连续tensor的strides
    std::vector<int64_t> strides(shape.size(), 1);
    for (int64_t i = shape.size() - 2; i >= 0; i--) {
        strides[i] = shape[i + 1] * strides[i + 1];
    }
    // 调用aclCreateTensor接口创建aclTensor
    *tensor = aclCreateTensor(shape.data(), shape.size(), dataType, strides.data(), 0,
aclFormat::ACL_FORMAT_ND,
                              shape.data(), shape.size(), *deviceAddr);
    return 0;
}
int main() {
    // 1. 固定写法，device/context/stream初始化, 参考acl对外接口列表
    // 根据自己的实际device填写deviceId
    int32_t deviceId = 0;
    aclrtContext context;
    aclrtStream stream;
    auto ret = Init(deviceId, &context, &stream);
    // check根据自己的需要处理
    CHECK_RET(ret == 0, LOG_PRINT("Init acl failed. ERROR: %d\n", ret); return ret);
    // 2. 构造输入与输出，需要根据API的接口定义构造
    std::vector<int64_t> xShape = {3, 4};
    std::vector<int64_t> idxShape = {3, 2};
    std::vector<int64_t> expandedXOutShape = {6, 4};
    std::vector<int64_t> idxOutShape = {1, 6};
    void* xDeviceAddr = nullptr;
    void* rowIdxDeviceAddr = nullptr;
    void* expertIdxDeviceAddr = nullptr;
    void* expandedXOutDeviceAddr = nullptr;
    void* expandedRowIdxOutDeviceAddr = nullptr;
    void* expandedExpertIdxOutDeviceAddr = nullptr;
    aclTensor* x = nullptr;
    aclTensor* rowIdx = nullptr;
    aclTensor* expertIdx = nullptr;
    int64_t activeNum = 3;
    aclTensor* expandedXOut = nullptr;
    aclTensor* expandedRowIdxOut = nullptr;
```

```
    aclTensor* expandedExpertIdxOut = nullptr;
    std::vector<float> xHostData = {0.1, 0.1, 0.1, 0.1, 0.2, 0.2, 0.2, 0.2, 0.3, 0.3, 0.3, 0.3};
    std::vector<int> expertIdxHostData = {1, 2, 0, 1, 0, 2};
    std::vector<int> rowIdxHostData = {0, 3, 1, 4, 2, 5};
    std::vector<float> expandedXOutHostData = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    std::vector<int> expandedRowIdxOutHostData = {0, 0, 0, 0, 0, 0};
    std::vector<int> expandedExpertIdxOutHostData = {0, 0, 0, 0, 0, 0};
    // 创建self aclTensor
    ret = CreateAclTensor(xHostData, xShape, &xDeviceAddr, aclDataType::ACL_FLOAT, &x);
    CHECK_RET(ret == ACL_SUCCESS, return ret);
    ret = CreateAclTensor(rowIdxHostData, idxShape, &rowIdxDeviceAddr, aclDataType::ACL_INT32, &rowIdx);
    CHECK_RET(ret == ACL_SUCCESS, return ret);
    ret = CreateAclTensor(expertIdxHostData, idxShape, &expertIdxDeviceAddr, aclDataType::ACL_INT32,
&expertIdx);
    CHECK_RET(ret == ACL_SUCCESS, return ret);
    // 创建out aclTensor
    ret = CreateAclTensor(expandedXOutHostData, expandedXOutShape, &expandedXOutDeviceAddr,
aclDataType::ACL_FLOAT, &expandedXOut);
    CHECK_RET(ret == ACL_SUCCESS, return ret);
    ret = CreateAclTensor(expandedRowIdxOutHostData, idxOutShape, &expandedRowIdxOutDeviceAddr,
aclDataType::ACL_INT32, &expandedRowIdxOut);
    CHECK_RET(ret == ACL_SUCCESS, return ret);
    ret = CreateAclTensor(expandedExpertIdxOutHostData, idxOutShape,
&expandedExpertIdxOutDeviceAddr, aclDataType::ACL_INT32, &expandedExpertIdxOut);
    CHECK_RET(ret == ACL_SUCCESS, return ret);
    // 3. 调用CANN算子库API，需要修改为具体的HostApi
    uint64_t workspaceSize = 0;
    aclOpExecutor* executor;
    // 调用aclnnAny第一段接口
    ret = aclnnMoeInitRoutingGetWorkspaceSize(x, rowIdx, expertIdx, activeNum, expandedXOut,
expandedRowIdxOut, expandedExpertIdxOut, &workspaceSize, &executor);
    CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclnnMoeInitRoutingGetWorkspaceSize failed. ERROR: %d
\n", ret); return ret);
    // 根据第一段接口计算出的workspaceSize申请device内存
    void* workspaceAddr = nullptr;
    if (workspaceSize > 0) {
        ret = aclrtMalloc(&workspaceAddr, workspaceSize, ACL_MEM_MALLOC_HUGE_FIRST);
        CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("allocate workspace failed. ERROR: %d\n", ret); return
ret;);
    }
    // 调用aclnnAny第二段接口
    ret = aclnnMoeInitRouting(workspaceAddr, workspaceSize, executor, stream);
    CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclnnMoeInitRouting failed. ERROR: %d\n", ret); return
ret);
    // 4. 固定写法，同步等待任务执行结束
    ret = aclrtSynchronizeStream(stream);
    CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("aclrtSynchronizeStream failed. ERROR: %d\n", ret); return
ret);
    // 5. 获取输出的值，将device侧内存上的结果拷贝至host侧，需要根据具体API的接口定义修改
    auto expandedXSize = GetShapeSize(expandedXOutShape);
    std::vector<float> expandedXData(expandedXSize, 0);
    ret = aclrtMemcpy(expandedXData.data(), expandedXData.size() * sizeof(expandedXData[0]),
expandedXOutDeviceAddr, expandedXSize * sizeof(float),
                      ACL_MEMCPY_DEVICE_TO_HOST);
    CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("copy result from device to host failed. ERROR: %d\n",
ret); return ret);
    for (int64_t i = 0; i < expandedXSize; i++) {
        LOG_PRINT("expandedXData[%ld] is: %f\n", i, expandedXData[i]);
    }
    auto expandedRowIdxSize = GetShapeSize(idxOutShape);
    std::vector<int> expandedRowIdxData(expandedRowIdxSize, 0);
    ret = aclrtMemcpy(expandedRowIdxData.data(), expandedRowIdxData.size() *
sizeof(expandedRowIdxData[0]), expandedRowIdxOutDeviceAddr, expandedRowIdxSize * sizeof(int32_t),
                      ACL_MEMCPY_DEVICE_TO_HOST);
    CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("copy result from device to host failed. ERROR: %d\n",
ret); return ret);
    for (int64_t i = 0; i < expandedRowIdxSize; i++) {
        LOG_PRINT("expandedRowIdxData[%ld] is: %d\n", i, expandedRowIdxData[i]);
    }
```

```
    auto expandedExpertIdxSize = GetShapeSize(idxOutShape);
    std::vector<int> expandedExpertIdxData(expandedExpertIdxSize, 0);
    ret = aclrtMemcpy(expandedExpertIdxData.data(), expandedExpertIdxData.size() *
sizeof(expandedExpertIdxData[0]), expandedExpertIdxOutDeviceAddr, expandedExpertIdxSize *
sizeof(int32_t),
                      ACL_MEMCPY_DEVICE_TO_HOST);
    CHECK_RET(ret == ACL_SUCCESS, LOG_PRINT("copy result from device to host failed. ERROR: %d\n",
ret); return ret);
    for (int64_t i = 0; i < expandedExpertIdxSize; i++) {
        LOG_PRINT("expandedExpertIdxData[%ld] is: %d\n", i, expandedExpertIdxData[i]);
    }
    // 6. 释放aclTensor和aclScalar，需要根据具体API的接口定义修改
    aclDestroyTensor(x);
    aclDestroyTensor(rowIdx);
    aclDestroyTensor(expertIdx);
    aclDestroyTensor(expandedXOut);
    aclDestroyTensor(expandedRowIdxOut);
    aclDestroyTensor(expandedExpertIdxOut);
    return 0;
}
```

11.13.7.10 aclnn 返回码

调用算子接口时，常见的接口返回码如表11-7所示。

表 11-7 返回状态码

状态码名称	状态码值	状态码说明
ACLNN_SUCCESS	0	成功。
ACLNN_ERR_PARAM_NULLPTR	161001	参数校验错误，参数中存在非法的nullptr。
ACLNN_ERR_PARAM_INVALID	161002	参数校验错误，如输入的两个数据类型不满足输入类型推导关系。 详细的错误消息，可以通过**aclGetRecentErrMsg**接口获取（该接口在acl.h中）。
ACLNN_ERR_RUNTIME_ERROR	361001	API内存调用npu runtime的接口异常。
ACLNN_ERR_INNER_XXX	561xxx	API发生内部异常，详细的错误消息，可以通过**aclGetRecentErrMsg**接口获取。请根据报错排查问题，或者联系技术支持。（您可以获取日志后单击**Link**联系技术支持。）

更多关于ACLNN_ERR_INNER_XXX类状态码的说明如表11-8所示。

表 11-8 异常状态码

状态码名称	状态码值	状态码说明
ACLNN_ERR_INNER	561000	内部异常：API发生内部异常。

状态码名称	状态码值	状态码说明
ACLNN_ERR_INNER_INFERSHAPE_ERROR	561001	内部异常：API内部进行输出shape推导发生错误。
ACLNN_ERR_INNER_TILING_ERROR	561002	内部异常：API内部做npu kernel的tiling时发生异常。
ACLNN_ERR_INNER_FIND_KERNEL_ERROR	561003	内部异常：API内部做查找npu kernel异常（可能因为opp kernel的包未安装）。
ACLNN_ERR_INNER_CREATE_EXECUTOR	561101	内部异常：API内部创建aclOpExecutor失败（可能因为操作系统异常）。
ACLNN_ERR_INNER_NOT_TRANS_EXECUTOR	561102	内部异常：API内部未调用uniqueExecutor ReleaseTo。
ACLNN_ERR_INNER_NULLPTR	561103	内部异常：aclnn API内部发生异常，出现了nullptr的异常。
ACLNN_ERR_INNER_WRONG_ATTR_INFO_SIZE	561104	内部异常：aclnn API内部发生异常，算子的属性个数异常。
ACLNN_ERR_INNER_KEY_CONFILICT	561105	内部异常：aclnn API内部发生异常，算子的kernel匹配的hash key发生冲突。
ACLNN_ERR_INNER_INVALID_IMPL_MODE	561106	内部异常：aclnn API内部发生异常，算子的实现模式参数错误。
ACLNN_ERR_INNER_OPP_PATH_NOT_FOUND	561107	内部异常：aclnn API内部发生异常，没有检测到需要配置的环境变量ASCEND_OPP_PATH。
ACLNN_ERR_INNER_LOAD_JSON_FAILED	561108	内部异常：aclnn API内部发生异常，加载算子kernel库中算子信息json文件失败。
ACLNN_ERR_INNER_JSON_VALUE_NOT_FOUND	561109	内部异常：aclnn API内部发生异常，加载算子kernel库中算子信息json文件的某个字段失败。
ACLNN_ERR_INNER_JSON_FORMAT_INVALID	561110	内部异常：aclnn API内部发生异常，算子kernel库中算子信息json文件的format填写为非法值。
ACLNN_ERR_INNER_JSON_DTYPE_INVALID	561111	内部异常：aclnn API内部发生异常，算子kernel库中算子信息json文件的dtype填写为非法值。
ACLNN_ERR_INNER_OPP_KERNEL_PKG_NOT_FOUND	561112	内部异常：aclnn API内部发生异常，没有加载到算子的二进制kernel库。
ACLNN_ERR_INNER_OP_FILE_INVALID	561113	内部异常：aclnn API内部发生异常，加载算子json文件字段时，发生异常。

状态码名称	状态码值	状态码说明
ACLNN_ERR_INNER_ATTR_NUM_OUT_OF_BOUND	561114	内部异常：aclnn API内部发生异常，算子的属性个数与算子信息json中不一致，超过了json中指定的attr个数。
ACLNN_ERR_INNER_ATTR_LEN_NOT_ENOUGH	561115	内部异常：aclnn API内部发生异常，算子的属性个数与算子信息json中不一致，少于json中指定的attr个数。
ACLNN_ERR_INNER_INPUT_NUM_IN_JSON_TOO_LARGE	561116	内部异常：aclnn API内部发生异常，算子的输入个数超出32的限制。
ACLNN_ERR_INNER_INPUT_JSON_IS_NULL	561117	内部异常：aclnn API内部发生异常，算子信息json文件中，不存在对输出的信息描述。
ACLNN_ERR_INNER_STATIC_WORKSPACE_INVALID	561118	内部异常：aclnn API内部发生异常，解析静态二进制json文件中的workspace信息时，发生异常。
ACLNN_ERR_INNER_STATIC_BLOCK_DIM_INVALID	561119	内部异常：aclnn API内部发生异常，解析静态二进制json文件中的核数使用信息时，发生异常。

11.14 算子编译

11.14.1 aclopRegisterCompileFunc

函数功能

动态Shape场景下，注册算子选择器，用于在算子执行时，能针对不同shape，选择相应的Tiling策略。

约束说明

如果某算子已注册算子选择器，则不允许重新注册，如果需要变更算子选择器，必须先调用**aclopUnregisterCompileFunc**接口取消注册，然后再调用aclopRegisterCompileFunc接口重新注册。

函数原型

aclError aclopRegisterCompileFunc(const char *opType, aclopCompileFunc func)

参数说明

参数名	输入/输出	说明
opType	输入	算子类型的指针。
func	输入	算子选择器回调函数，函数定义： typedef aclError (*aclopCompileFunc)(int numInputs, const aclTensorDesc *const inputDesc[], int numOutputs, const aclTensorDesc *const outputDesc[], const aclopAttr *opAttr, aclopKernelDesc *aclopKernelDesc);

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见6.3.4 执行动态Shape算子示例代码（注册算子选择器）。

11.14.2 aclopUnregisterCompileFunc

函数功能

动态Shape场景下，取消注册算子选择器。

函数原型

aclError aclopUnregisterCompileFunc(const char *opType)

参数说明

参数名	输入/输出	说明
opType	输入	算子类型的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.14.3 aclopCreateKernel

函数功能

动态Shape场景下，将算子注册到系统内部，运行算子时使用。

函数原型

aclError aclopCreateKernel(const char *opType,

const char *kernelId,

const char *kernelName,

void *binData,

int binSize,

aclopEngineType enginetype,

aclDataDeallocator deallocator)

参数说明

参数名	输入/输出	说明
opType	输入	算子类型的指针。
kernelId	输入	算子执行时要指定的Kernel ID的指针。
kernelName	输入	算子Kernel名称的指针，和算子二进制文件中的kernelName保持一致。
binData	输入	算子Kernel文件的内存地址指针。
binSize	输入	算子Kernel文件的内存大小，单位为Byte。
enginetype	输入	表示算子执行引擎。 typedef enum aclEngineType { ACL_ENGINE_SYS, //不关心具体执行引擎时填写 ACL_ENGINE_AICORE, //将算子编译成AI Core算子 ACL_ENGINE_VECTOR, //将算子编译成Vector Core算子 } aclopEngineType;
deallocator	输入	释放binData内存的回调函数，调用者根据binData的构造方法设置对应的释放函数。如果数据由调用者释放则设置为空。 函数签名为： typedef void (*aclDataDeallocator)(void *data, size_t length);

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**6.3.4 执行动态Shape算子示例代码（注册算子选择器）**。

11.14.4 aclopSetKernelArgs

函数功能

动态Shape场景下，设置算子Tiling参数、执行并发数。

函数原型

```
aclError aclopSetKernelArgs(aclopKernelDesc *kernelDesc,
const char *kernelId,
uint32_t blockDim,
const void *args,
uint32_t argSize)
```

参数说明

参数名	输入/输出	说明
kernelDesc	输入	Kernel描述缓存，aclopKernelDesc类型的指针。 typedef struct aclopKernelDesc aclopKernelDesc;
kernelId	输入	算子执行时要指定的Kernel ID的指针，与调用**aclopCreateKernel**时传递的kernelId一致。
blockDim	输入	Kernel执行的并发AI Core核数。 建议此处设置的blockDim和TIK算子实现时的使用的AI Core核数保持一致。
args	输入	Tiling参数的指针。
argSize	输入	Tiling参数内存大小，单位为Byte。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**6.3.4 执行动态Shape算子示例代码（注册算子选择器）**。

11.14.5 aclopSetKernelWorkspaceSizes

函数功能

动态Shape场景下，设置算子Workspace参数。为非必须接口，根据算子情况可选。

函数原型

```
aclError aclopSetKernelWorkspaceSizes(aclopKernelDesc *kernelDesc, int
numWorkspaces, size_t *workspaceSizes)
```

参数说明

参数名	输入/输出	说明
kernelDesc	输入	Kernel描述缓存，aclopKernelDesc类型的指针。 typedef struct aclopKernelDesc aclopKernelDesc
numWorkspaces	输入	Workspaces个数。
workspaceSizes	输入	Workspaces大小的数组地址指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.14.6 aclopUpdateParams

函数功能

在算子动态Shape场景下，根据指定算子，触发调用算子选择器。

函数原型

```
aclError aclopUpdateParams(const char *opType,
int numInputs,
const aclTensorDesc *const inputDesc[],
int numOutputs,
const aclTensorDesc *const outputDesc[],
const aclopAttr *attr)
```

参数说明

参数名	输入/输出	说明
opType	输入	算子类型名称的指针。
numInputs	输入	算子输入tensor的数量。
inputDesc	输入	算子输入tensor的描述的指针数组。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。 inputDesc数组中的元素个数必须与numInputs参数值保持一致。

参数名	输入/输出	说明
numOutputs	输入	算子输出tensor的数量。
outputDesc	输入	算子输出tensor的描述的指针数组。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。 outputDesc数组中的元素个数必须与numOutputs参数值保持一致。
attr	输入	算子属性的指针。 需提前调用**aclopCreateAttr**接口创建aclopAttr类型。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**6.3.4 执行动态Shape算子示例代码（注册算子选择器）**。

11.14.7 aclopCompile

函数功能

编译指定算子。在编译算子前，可以调用**aclSetCompileopt**接口设置编译选项。

约束说明

- 每个算子的输入、输出组织不同，需要应用在调用时严格按照算子输入、输出参数来组织算子，用户在调用aclopCompile时，AscendCL根据optype、输入tensor的描述、输出tensor的描述、attr等信息查找对应的任务，再编译算子。
- 编译动态Shape的算子时，如果Shape具体值不明确，但Shape范围明确，则在调用aclCreateTensorDesc接口创建**aclTensorDesc**类型时，需要将dims数组的表示动态维度的元素值设置为-1，再调用**aclSetTensorShapeRange**接口设置tensor的各个维度的取值范围；如果Shape具体值以及Shape范围都不明确（该场景预留），则在调用aclCreateTensorDesc接口创建**aclTensorDesc**类型时，需要将dims数组的值设置为-2，例如：int64_t dims[] = {-2}。
- 编译有可选输入的算子时，如果可选输入不使用，则需按此种方式创建**aclTensorDesc**类型的数据：**aclCreateTensorDesc**(ACL_DT_UNDEFINED, 0, nullptr, ACL_FORMAT_UNDEFINED)，表示数据类型设置为ACL_DT_UNDEFINED，Format设置为ACL_FORMAT_UNDEFINED，Shape信息为nullptr。
- 编译有constant输入的算子时，需要先调用**aclSetTensorConst**接口设置constant输入。调用aclopCompile接口、**aclopExecuteV2**接口前，设置的constant输入数据必须保持一致。

函数原型

```
aclError aclopCompile(const char *opType,
int numInputs,
const aclTensorDesc *const inputDesc[],
int numOutputs,
const aclTensorDesc *const outputDesc[],
const aclopAttr *attr,
aclopEngineType engineType,
aclopCompileType compileFlag,
const char* opPath)
```

参数说明

参数名	输入/输出	说明
opType	输入	算子类型名称的指针。
numInputs	输入	算子输入tensor的数量。
inputDesc	输入	算子输入tensor的描述的指针数组。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。 inputDesc数组中的元素个数必须与numInputs参数值保持一致。
numOutputs	输入	算子输出tensor的数量。
outputDesc	输入	算子输出tensor的描述的指针数组。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。 outputDesc数组中的元素个数必须与numOutputs参数值保持一致。
attr	输入	算子属性的指针。 需提前调用**aclopCreateAttr**接口创建aclopAttr类型。
engineType	输入	算子执行引擎。
compileFlag	输入	算子编译标识。
opPath	输入	算子实现文件（*.py）的路径的指针，不包含文件名。 预留参数，当前仅支持设置为nullptr。

返回值说明

返回0表示成功，返回其它值表示失败。

11.14.8 aclopSetCompileFlag

须知

此接口后续版本会废弃，请使用aclSetCompileopt接口设置编译选项。

函数功能

在编译算子前调用本接口设置算子编译标识，该设置进程级全局共享。

函数原型

aclError aclopSetCompileFlag(aclOpCompileFlag flag)

参数说明

参数名	输入/输出	说明
flag	输入	用于设置算子编译标识。 如果没有设置flag，则默认按照精确编译。

返回值说明

返回0表示成功，返回其它值表示失败。

11.14.9 aclGenGraphAndDumpForOp

函数功能

构建单个算子的图并Dump，生成.txt文件。此处的图是指表达算子IR（Intermediate Representation）的结构，图编译过程中由于融合、拆分等原因会导致图不断变换、进而生成新的图，这些变换的目的是为了将用户表达的图适配到昇腾AI处理器上，获得更高性能。

使用场景：算子调优场景下，调用本接口构建单个算子的图并Dump，生成.txt文件，作为算子调优的输入数据之一。

函数原型

```
aclError aclGenGraphAndDumpForOp(const char *opType,
int numInputs,
const aclTensorDesc *const inputDesc[],
```

```
const aclDataBuffer *const inputs[],
int numOutputs,
const aclTensorDesc *const outputDesc[],
aclDataBuffer *const outputs[],
const aclopAttr *attr,
aclopEngineType engineType,
const char *graphDumpPath,
const aclGraphDumpOption *graphDumpOpt)
```

参数说明

参数名	输入/输出	说明
opType	输入	指定算子类型名称的指针。
numInputs	输入	算子输入tensor的数量。
inputDesc	输入	算子输入tensor的描述的指针数组。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。 inputDesc数组中的元素个数必须与numInputs参数值保持一致，且inputs数组与inputDesc数组中的元素必须一一对应。
inputs	输入	算子输入tensor的指针数组。 需提前调用**aclCreateDataBuffer**接口创建aclDataBuffer类型的数据。 inputs数组中的元素个数必须与numInputs参数值保持一致，且inputs数组与inputDesc数组中的元素必须一一对应。
numOutputs	输入	算子输出tensor的数量。
outputDesc	输入	算子输出tensor的描述的指针数组。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。 outputDesc数组中的元素个数必须与numOutputs参数值保持一致，且outputs数组与outputDesc数组中的元素必须一一对应。
outputs	输入	算子输出tensor的指针数组。 需提前调用**aclCreateDataBuffer**接口创建aclDataBuffer类型的数据。 outputs数组中的元素个数必须与numOutputs参数值保持一致，且outputs数组与outputDesc数组中的元素必须一一对应。

参数名	输入/输出	说明
attr	输入	算子属性的指针。 需提前调用aclopCreateAttr接口创建aclopAttr类型。
engineType	输入	算子执行引擎。
graphDumpPath	输入	Dump单算子图所生成的文件（文件名后缀为.txt）的保存路径的指针。 路径需要由用户提前创建，路径中如果不指定文件名，则由AscendCL内部指定文件名。 如果多次Dump时，指定同一个路径，且文件名相同，则最新一次的文件会覆盖之前的文件。
graphDumpOpt	输入	单算子图的dump选项的指针。 当前该参数预留，仅支持传入nullptr。

返回值说明

返回0表示成功，返回其它值表示失败。

11.15 算子加载与执行

11.15.1 aclopSetModelDir

函数功能

设置加载模型文件的目录，该模型文件是由单算子编译成的*.om文件。

约束说明

在加载前，请先根据单算子om文件的大小评估内存空间是否足够，内存空间不足，会导致应用程序异常。

Atlas 200/300/500 推理产品，在Device上运行应用时，一个进程内正在执行的算子的最大个数上限是2000000。

Atlas 推理系列产品，在Device上运行应用时，一个进程内正在执行的算子的最大个数上限是2000000。

函数原型

aclError aclopSetModelDir(const char *modelDir)

参数说明

参数名	输入/输出	说明
modelDir	输入	模型文件路径的指针。 此处可设置多级目录，但系统最多从多级目录的最后一级开始，读取三级目录下的模型文件。 例如，将modelDir设置为"dir0/dir1"，dir1下的目录层级为“dir2/dir3/dir4”，这时系统只支持读取“dir1”、“dir1/dir2”、“dir1/dir2/dir3”目录下的模型文件。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**6 单算子调用**。

11.15.2 aclopLoad

函数功能

从内存中加载单算子模型数据（单算子模型数据是指“单算子编译成*.om文件后，再将om文件读取到内存中”的数据），由用户管理内存。

约束说明

Atlas 200/300/500 推理产品，在Device上运行应用时，一个进程内正在执行的算子的最大个数上限是2000000。

Atlas 推理系列产品，在Device上运行应用时，一个进程内正在执行的算子的最大个数上限是2000000。

函数原型

aclError aclopLoad(const void *model, size_t modelSize)

参数说明

参数名	输入/输出	说明
model	输入	单算子模型数据的内存地址指针。
modelSize	输入	内存中的模型数据长度，单位Byte。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**6 单算子调用**。

11.15.3 aclopExecute

须知

此接口后续版本会废弃，请使用**aclopExecuteV2**接口。

函数功能

执行指定的算子，异步接口。

约束说明

- 该接口是异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功。调用该接口后，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保任务已执行完成。
- 多线程场景下，不支持调用本接口时指定同一个Stream或使用默认Stream，否则可能任务执行异常。
- 每个算子的输入、输出组织不同，需要应用在调用时严格按照算子输入、输出参数来组织算子。用户在调用aclopExecute接口时，AscendCL根据optype、输入tensor的描述、输出tensor的描述、attr等信息查找对应的任务，并下发执行。

函数原型

```
aclError aclopExecute(const char *opType,
int numInputs,
const aclTensorDesc *const inputDesc[],
const aclDataBuffer *const inputs[],
int numOutputs,
const aclTensorDesc *const outputDesc[],
aclDataBuffer *const outputs[],
const aclopAttr *attr,
aclrtStream stream)
```

参数说明

参数名	输入/输出	说明
opType	输入	算子类型名称的指针。
numInputs	输入	算子输入tensor的数量。
inputDesc	输入	算子输入tensor描述的指针数组。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。 inputDesc数组中的元素个数必须与numInputs参数值保持一致，且inputs数组与inputDesc数组中的元素必须一一对应。
inputs	输入	算子输入tensor的指针数组。 需提前调用**aclCreateDataBuffer**接口创建aclDataBuffer类型的数据。 inputs数组中的元素个数必须与numInputs参数值保持一致，且inputs数组与inputDesc数组中的元素必须一一对应。
numOutputs	输入	算子输出tensor的数量。 outputDesc数组中的元素个数必须与numOutputs参数值保持一致。
outputDesc	输入	算子输出tensor描述的指针数组。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。 outputDesc数组中的元素个数必须与numOutputs参数值保持一致，且outputs数组与outputDesc数组中的元素必须一一对应。
outputs	输出	算子输出tensor的指针数组。 需提前调用**aclCreateDataBuffer**接口创建aclDataBuffer类型的数据。 outputs数组中的元素个数必须与numOutputs参数值保持一致，且outputs数组与outputDesc数组中的元素必须一一对应。
attr	输入	算子属性的指针。 需提前调用**aclopCreateAttr**接口创建aclopAttr类型。
stream	输入	该算子需要加载的stream。

返回值说明

返回0表示成功，返回其它值表示失败。

11.15.4 aclopExecuteV2

函数功能

执行指定的算子。

约束说明

- 该接口是异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功。调用该接口后，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保任务已执行完成。
- 多线程场景下，不支持调用本接口时指定同一个Stream或使用默认Stream，否则可能任务执行异常。
- 每个算子的输入、输出组织不同，需要应用在调用时严格按照算子输入、输出参数来组织算子。用户在调用aclopExecuteV2接口时，AscendCL根据optype、输入tensor的描述、输出tensor的描述、attr等信息查找对应的任务，并下发执行。
- 对于支持动态Shape的算子，无法明确算子输出Shape时，可调用**aclopInferShape**接口获取算子的输出Shape：
  - 若可获取到准确的输出Shape，则使用准确的输出Shape来构造outputDesc参数，作为aclopExecuteV2的输入。该场景下，aclopExecuteV2接口是**异步接口**，对于异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功。调用该接口后，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保任务已执行完成。
  - 若无法获取准确的输出Shape，仅能获取输出Shape范围，则使用Shape最大值来构造outputDesc参数，作为aclopExecuteV2的输入。该场景下，在调用aclopExecuteV2接口执行算子后，系统内部会计算出准确的输出Shape，通过aclopExecuteV2接口的outputDesc参数输出，此时aclopExecuteV2接口是**同步接口**。
  - （该场景预留）若无法获取准确的输出Shape以及Shape范围，则需由用户预估一个最大的Shape来构造outputDesc参数，作为aclopExecuteV2的输入。该场景下，在调用aclopExecuteV2接口执行算子后，系统内部会计算出准确的输出Shape，通过aclopExecuteV2接口的outputDesc参数输出，此时aclopExecuteV2接口是**同步接口**。
- 执行有可选输入的算子时，如果可选输入不使用：
  - 则需按此种方式创建**aclTensorDesc**类型的数据：**aclCreateTensorDesc**(ACL_DT_UNDEFINED, 0, nullptr, ACL_FORMAT_UNDEFINED)，表示数据类型设置为ACL_DT_UNDEFINED，Format设置为ACL_FORMAT_UNDEFINED，Shape信息为nullptr。
  - 则需按此种方式创建**aclDataBuffer**类型的数据：aclCreateDataBuffer(nullptr, 0)，同时aclDataBuffer中的数据不需要释放，因为是空指针。
- 执行有constant输入的算子时，需要先调用**aclSetTensorConst**接口设置constant输入。

  调用**aclopCompile**接口、aclopExecuteV2接口时，设置的constant输入数据必须保持一致。

  对于算子有constant输入的场景，如果未调用**aclSetTensorConst**接口设置constant输入，则需调用**aclSetTensorPlaceMent**设置TensorDesc的placement属性，将memType设置为Host内存。

- 在执行单算子时，一般要求用户传入的输入/输出tensor数据是存放在Device内存的，比如两个tensor相加的场景。但是，也存在部分算子，除了将featureMap、weight等Device内存中的tensor数据作为输入外，还需tensor shape信息、learningRate、tensor的dims信息等通常在Host内存中的tensor数据也作为输入，此时，用户不需要额外将这些Host内存中的tensor数据拷贝到Device上作为输入，只需要调用**aclSetTensorPlaceMent**接口设置对应TensorDesc的placement属性为Host内存，在执行单算子时，设置了placement属性为Host内存的TensorDesc，其对应的tensor数据必须存放在Host内存中，AscendCL内部会自动将Host上的tensor数据拷贝到Device上。

函数原型

```
aclError aclopExecuteV2(const char *opType,
int numInputs,
aclTensorDesc *inputDesc[],
aclDataBuffer *inputs[],
int numOutputs,
aclTensorDesc *outputDesc[],
aclDataBuffer *outputs[],
aclopAttr *attr,
aclrtStream stream)
```

参数说明

参数名	输入/输出	说明
opType	输入	算子类型名称的指针。
numInputs	输入	算子输入tensor的数量。
inputDesc	输入	算子输入tensor描述的指针数组。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。 inputDesc数组中的元素个数必须与numInputs参数值保持一致，且inputs数组与inputDesc数组中的元素必须一一对应。
inputs	输入	算子输入tensor的指针数组。 需提前调用**aclCreateDataBuffer**接口创建aclDataBuffer类型的数据。 inputs数组中的元素个数必须与numInputs参数值保持一致，且inputs数组与inputDesc数组中的元素必须一一对应。
numOutputs	输入	算子输出tensor的数量。

参数名	输入/输出	说明
outputDesc	输入&输出	算子输出tensor描述的指针数组。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。 outputDesc数组中的元素个数必须与numOutputs参数值保持一致，且outputs数组与outputDesc数组中的元素必须一一对应。
outputs	输出	算子输出tensor的指针数组。 需提前调用**aclCreateDataBuffer**接口创建aclDataBuffer类型的数据。 outputs数组中的元素个数必须与numOutputs参数值保持一致，且outputs数组与outputDesc数组中的元素必须一一对应。
attr	输入	算子属性的指针。 需提前调用**aclopCreateAttr**接口创建aclopAttr类型。
stream	输入	该算子需要加载的stream。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**6 单算子调用**。

11.15.5 aclopCompileAndExecute

函数功能

编译并执行指定的算子，当前只支持固定Shape的算子，异步接口。在编译执行算子前，可以调用**aclSetCompileopt**接口设置编译选项。

约束说明

- 该接口是异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功。调用该接口后，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保任务已执行完成。
- 多线程场景下，不支持调用本接口时指定同一个Stream或使用默认Stream，否则可能任务执行异常。
- 每个算子的输入、输出组织不同，需要应用在调用时严格按照算子输入、输出参数来组织算子，用户在调用aclopCompileAndExecute时，AscendCL根据optype、输入tensor的描述、输出tensor的描述、attr等信息查找对应的任务，再编译、执行算子。

- 编译、执行有可选输入的算子时，如果可选输入不使用，则：
  - 需按此种方式创建aclTensorDesc类型的数据：**aclCreateTensorDesc**(ACL_DT_UNDEFINED, 0, nullptr, ACL_FORMAT_UNDEFINED)，表示数据类型设置为ACL_DT_UNDEFINED，Format设置为ACL_FORMAT_UNDEFINED，Shape信息为nullptr；
  - 需按此种方式创建aclDataBuffer类型的数据：**aclCreateDataBuffer**(nullptr, 0)，同时aclDataBuffer中的数据不需要释放，因为是空指针。
- 编译、执行有constant输入的算子时，需要先调用**aclSetTensorConst**接口设置constant输入。

  对于算子有constant输入的场景，如果未调用**aclSetTensorConst**接口设置constant输入，则需调用**aclSetTensorPlaceMent**设置TensorDesc的placement属性，将memType设置为Host内存。
- 在执行单算子时，一般要求用户传入的输入/输出tensor数据是存放在Device内存的，比如两个tensor相加的场景。但是，也存在部分算子，除了将featureMap、weight等Device内存中的tensor数据作为输入外，还需tensor shape信息、learningRate、tensor的dims信息等通常在Host内存中的tensor数据也作为输入，此时，用户不需要额外将这些Host内存中的tensor数据拷贝到Device上作为输入，只需要调用**aclSetTensorPlaceMent**接口设置对应TensorDesc的placement属性为Host内存，在执行单算子时，设置了placement属性为Host内存的TensorDesc，其对应的tensor数据必须存放在Host内存中，AscendCL内部会自动将Host上的tensor数据拷贝到Device上。

函数原型

```
aclError aclopCompileAndExecute(const char *opType,
int numInputs,
const aclTensorDesc *const inputDesc[],
const aclDataBuffer *const inputs[],
int numOutputs,
const aclTensorDesc *const outputDesc[],
aclDataBuffer *const outputs[],
const aclopAttr *attr,
aclopEngineType engineType,
aclopCompileType compileFlag,
const char *opPath,
aclrtStream stream)
```

参数说明

参数名	输入/输出	说明
opType	输入	算子类型名称的指针。

参数名	输入/输出	说明
numInputs	输入	算子输入tensor的数量。
inputDesc	输入	算子输入tensor描述的指针数组。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。 inputDesc数组中的元素个数必须与numInputs参数值保持一致，且inputs数组与inputDesc数组中的元素必须一一对应。
inputs	输入	算子输入tensor的指针数组。 需提前调用**aclCreateDataBuffer**接口创建aclDataBuffer类型的数据。 inputs数组中的元素个数必须与numInputs参数值保持一致，且inputs数组与inputDesc数组中的元素必须一一对应。
numOutputs	输入	算子输出tensor的数量。
outputDesc	输入	算子输出tensor描述的指针数组。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。 outputDesc数组中的元素个数必须与numOutputs参数值保持一致，且outputs数组与outputDesc数组中的元素必须一一对应。
outputs	输入&输出	算子输出tensor的指针数组。 需提前调用**aclCreateDataBuffer**接口创建aclDataBuffer类型的数据。 outputs数组中的元素个数必须与numOutputs参数值保持一致，且outputs数组与outputDesc数组中的元素必须一一对应。
attr	输入	算子属性的指针。 需提前调用**aclopCreateAttr**接口创建aclopAttr类型。
engineType	输入	算子执行引擎。
compileFlag	输入	算子编译标识。
opPath	输入	算子实现文件（*.py）路径的指针，不包含文件名。 预留参数，当前仅支持设置为nullptr。
stream	输入	该算子需要加载的stream。

返回值说明

返回0表示成功，返回其它值表示失败。

11.15.6 aclopCompileAndExecuteV2

函数功能

编译并执行指定的算子，当前只支持固定Shape的算子。在编译执行算子前，可以调用**aclSetCompileopt**接口设置编译选项。

约束说明

- 多线程场景下，不支持调用本接口时指定同一个Stream或使用默认Stream，否则可能任务执行异常。
- 每个算子的输入、输出组织不同，需要应用在调用时严格按照算子输入、输出参数来组织算子，用户在调用aclopCompileAndExecuteV2时，AscendCL根据optype、输入tensor的描述、输出tensor的描述、attr等信息查找对应的任务，再编译、执行算子。
- 编译、执行有可选输入的算子时，如果可选输入不使用，则：
  - 需按此种方式创建aclTensorDesc类型的数据：**aclCreateTensorDesc**(ACL_DT_UNDEFINED, 0, nullptr, ACL_FORMAT_UNDEFINED)，表示数据类型设置为ACL_DT_UNDEFINED，Format设置为ACL_FORMAT_UNDEFINED，Shape信息为nullptr；
  - 需按此种方式创建aclDataBuffer类型的数据：**aclCreateDataBuffer**(nullptr, 0)，同时aclDataBuffer中的数据不需要释放，因为是空指针。
- 编译、执行有constant输入的算子时，需要先调用**aclSetTensorConst**接口设置constant输入。

  对于算子有constant输入的场景，如果未调用**aclSetTensorConst**接口设置constant输入，则需调用**aclSetTensorPlaceMent**设置TensorDesc的placement属性，将memType设置为Host内存。
- 在执行单算子时，一般要求用户传入的输入/输出tensor数据是存放在Device内存的，比如两个tensor相加的场景。但是，也存在部分算子，除了将featureMap、weight等Device内存中的tensor数据作为输入外，还需tensor shape信息、learningRate、tensor的dims信息等通常在Host内存中的tensor数据也作为输入，此时，用户不需要额外将这些Host内存中的tensor数据拷贝到Device上作为输入，只需要调用**aclSetTensorPlaceMent**接口设置对应TensorDesc的placement属性为Host内存，在执行单算子时，设置了placement属性为Host内存的TensorDesc，其对应的tensor数据必须存放在Host内存中，AscendCL内部会自动将Host上的tensor数据拷贝到Device上。
- 对于动态Shape的算子，无法明确算子输出Shape时，可调用**aclopInferShape**接口获取算子的输出Shape：
  - 若可获取到准确的输出Shape，则使用准确的输出Shape来构造outputDesc参数，作为aclopCompileAndExecuteV2的输入。该场景下，aclopCompileAndExecuteV2接口是**异步接口**，对于异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功。调用该接口后，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保任务已执行完成。
  - 若无法获取准确的输出Shape，仅能获取输出Shape范围，则使用Shape最大值来构造outputDesc参数，作为aclopCompileAndExecuteV2的输入。该场景下，在调用aclopCompileAndExecuteV2接口执行算子后，系统内部会计算出准确的输出Shape，通过aclopCompileAndExecuteV2接口的outputDesc参数输出，此时aclopCompileAndExecuteV2接口是**同步接口**。

- （该场景预留）若无法获取准确的输出Shape以及Shape范围，则需由用户预估一个最大的Shape来构造outputDesc参数，作为aclopCompileAndExecuteV2的输入。该场景下，在调用aclopCompileAndExecuteV2接口执行算子后，系统内部会计算出准确的输出Shape，通过aclopCompileAndExecuteV2接口的outputDesc参数输出，此时aclopCompileAndExecuteV2接口是**同步接口**。

函数原型

```
aclError aclopCompileAndExecuteV2(const char *opType,
int numInputs,
aclTensorDesc *inputDesc[],
aclDataBuffer *inputs[],
int numOutputs,
aclTensorDesc *outputDesc[],
aclDataBuffer *outputs[],
aclopAttr *attr,
aclopEngineType engineType,
aclopCompileType compileFlag,
const char *opPath,
aclrtStream stream)
```

参数说明

参数名	输入/输出	说明
opType	输入	算子类型名称的指针。
numInputs	输入	算子输入tensor的数量。
inputDesc	输入	算子输入tensor描述的指针数组。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。 inputDesc数组中的元素个数必须与numInputs参数值保持一致，且inputs数组与inputDesc数组中的元素必须一一对应。
inputs	输入	算子输入tensor的指针数组。 需提前调用**aclCreateDataBuffer**接口创建aclDataBuffer类型的数据。 inputs数组中的元素个数必须与numInputs参数值保持一致，且inputs数组与inputDesc数组中的元素必须一一对应。
numOutputs	输入	算子输出tensor的数量。

参数名	输入/输出	说明
outputDesc	输入&输出	算子输出tensor描述的指针数组。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。 outputDesc数组中的元素个数必须与numOutputs参数值保持一致，且outputs数组与outputDesc数组中的元素必须一一对应。
outputs	输入&输出	算子输出tensor的指针数组。 需提前调用**aclCreateDataBuffer**接口创建aclDataBuffer类型的数据。 outputs数组中的元素个数必须与numOutputs参数值保持一致，且outputs数组与outputDesc数组中的元素必须一一对应。
attr	输入	算子属性的指针。 需提前调用**aclopCreateAttr**接口创建aclopAttr类型。
engineType	输入	算子执行引擎。
compileFlag	输入	算子编译标识。
opPath	输入	算子实现文件（*.py）路径的指针，不包含文件名。 预留参数，当前仅支持设置为nullptr。
stream	输入	该算子需要加载的stream。

返回值说明

返回0表示成功，返回其它值表示失败。

11.15.7 aclopExecWithHandle

函数功能

以Handle方式调用一个算子，不支持动态Shape算子，动态Shape算子请使用**aclopExecuteV2**。异步接口。

约束说明

- 该接口是异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功。调用该接口后，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保任务已执行完成。
- 多线程场景下，不支持调用本接口时指定同一个Stream或使用默认Stream，否则可能任务执行异常。
- 执行有可选输入的算子时，如果可选输入不使用，则需按此种方式创建**aclDataBuffer**类型的数据：aclCreateDataBuffer(nullptr, 0)，同时aclDataBuffer中的数据不需要释放，因为是空指针。

函数原型

```
aclError aclopExecWithHandle(aclopHandle *handle,
int numInputs,
const aclDataBuffer *const inputs[],
int numOutputs,
aclDataBuffer *const outputs[],
aclrtStream stream);
```

参数说明

参数名	输入/输出	说明
handle	输入	算子执行算子handle的指针。 需提前调用**aclopCreateHandle**接口创建aclopHandle类型的数据。
numInputs	输入	算子输入tensor的数量。
inputs	输入	算子输入tensor的指针数组。 需提前调用**aclCreateDataBuffer**接口创建aclDataBuffer类型的数据。 inputs数组中的元素个数必须与numInputs参数值保持一致。
numOutputs	输入	算子输出tensor的数量。
outputs	输出	算子输出tensor的指针数组。 需提前调用**aclCreateDataBuffer**接口创建aclDataBuffer类型的数据。 outputs数组中的元素个数必须与numOutputs参数值保持一致
stream	输入	该算子需要加载的stream。

返回值说明

返回0表示成功，返回其它值表示失败。

11.15.8 aclopInferShape

函数功能

根据算子的输入Shape、输入值推导出算子的输出Shape。

约束说明

- 推导算子的输出Shape包含三种场景：
  - 根据Shape推导，可以得到算子的准确输出Shape，则返回准确输出Shape；
  - 根据Shape推导，无法得到算子的准确输出Shape，但可以得到输出Shape的范围，则在输出参数outputDesc中将算子输出tensor描述中的动态维度的维度值记为-1。该场景下，用户可调用**aclGetTensorDescDimRange**接口获取tensor描述中指定维度的范围值。
  - （该场景预留）根据Shape推导，无法得到算子的准确输出Shape以及Shape范围，则在输出参数outputDesc中将算子输出tensor描述中的动态维度的维度值记为-2。
- 如果算子有动态输入DYNAMIC_INPUT或动态输出DYNAMIC_OUTPUT，在调用aclopInferShape接口推导算子的输出Shape前，需先调用**aclSetTensorDescName**接口设置所有输入和输出的tensor描述的名称，且名称必须按照如下要求（算子IR原型中定义的输入/输出名称请参见《**算子清单（开放态）**》。）
  - 对于必选输入、可选输入、必选输出，名称必须与算子IR原型中定义的输入/输出名称保持一致。
  - 对于动态输入、动态输出，名称必须是：算子IR原型中定义的输入/输出名称+*编号*。编号根据动态输入/输出的个数确定，从0开始，0对应第一个动态输入/输出，1对应第二个动态输入/输出，以此类推。

  例如某个算子有2个输入（第1个是必选输入x，第二个是动态输入y且输入个数为2）、1个必选输出z，则调用**aclSetTensorDescName**接口设置名称的代码示例如下：

```
aclSetTensorDescName(inputTensorDesc[0], "x");
aclSetTensorDescName(inputTensorDesc[1], "y0");
aclSetTensorDescName(inputTensorDesc[2], "y1");
aclSetTensorDescName(outputTensorDesc[0], "z");
```

函数原型

aclError aclopInferShape(const char *opType,

int numInputs,

aclTensorDesc *inputDesc[],

aclDataBuffer *inputs[],

int numOutputs,

aclTensorDesc *outputDesc[],

aclopAttr *attr)

参数说明

参数名	输入/输出	说明
opType	输入	算子类型名称的指针。
numInputs	输入	算子输入tensor的数量。

参数名	输入/输出	说明
inputDesc	输入	算子输入tensor描述的指针数组。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。 inputDesc数组中的元素个数必须与numInputs参数值保持一致。
inputs	输入	算子输入tensor的指针数组。 需提前调用**aclCreateDataBuffer**接口创建aclDataBuffer类型的数据。 此处算子输入tensor数据的内存必须根据应用运行模式来确定，应用运行在Host时，此处需申请Host上的内存；应用运行在Device时，此处需申请Device上的内存。内存申请接口请参见**11.11 内存管理**。
numOutputs	输入	算子输出tensor的数量。
outputDesc	输出	算子输出tensor描述的指针数组。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。 outputDesc数组中的元素个数必须与numOutputs参数值保持一致
attr	输入	算子属性。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见**6.3.3 执行动态Shape算子示例代码（不注册算子选择器）**。

11.15.9 aclrtSetOpExecuteTimeOut

函数功能

设置算子执行的超时时间。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 不调用本接口，算子执行的默认超时时间如下：

版本	AI Core算子的默认超时时间	AI CPU算子的默认超时时间
Atlas 推理系列产品	547秒	30秒

- 一个进程内多次调用本接口，则以最后一次设置的时间为准，允许设置的最大时间如下：

版本	AI Core算子的最大超时时间	AI CPU算子的最大超时时间
Atlas 推理系列产品	547秒	不支持设置

函数原型

aclError aclrtSetOpExecuteTimeOut(uint32_t timeout)

参数说明

参数名	输入/输出	说明
timeout	输入	设置超时时间，单位为秒。 将该参数设置为0时，表示使用默认超时时间。

返回值说明

返回0表示成功，返回其它值表示失败。

11.15.10 aclopStartDumpArgs

函数功能

调用本接口开启算子信息统计功能，并需与**aclopStopDumpArgs**接口配合使用，将算子信息文件输出到path参数指定的目录，一个shape对应一个算子信息文件，文件中包含算子类型、算子属性、算子输入&输出的format/数据类型/shape等信息。

使用场景：例如要统计某个模型执行涉及哪些算子，可在模型执行之前调用aclopStartDumpArgs接口，在模型执行之后调用aclopStopDumpArgs接口，接口调用成功后，在path参数指定的目录下生成每个算子shape的算子信息文件。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

aclError aclopStartDumpArgs(uint32_t dumpType, const char *path)

参数说明

参数名	输入/输出	说明
dumpType	输入	指定dump信息的类型。 当前仅支持ACL_OP_DUMP_OP_AICORE_ARGS，表示统计所有算子信息。 #define ACL_OP_DUMP_OP_AICORE_ARGS 0x00000001U
path	输入	指定dump文件的保存路径，支持绝对路径（指相对应用可执行文件所在的目录）或相对路径，但用户需确保路径存在或者该路径可以被创建。

返回值说明

返回0表示成功，返回其它值表示失败。

11.15.11 aclopStopDumpArgs

函数功能

调用本接口关闭算子信息统计功能，并需与**aclopStartDumpArgs**接口配合使用，将算子信息文件输出到path参数指定的目录，一个shape对应一个算子信息文件，文件中包含算子类型、算子属性、算子输入&输出的format/数据类型/shape等信息。

使用场景：例如要统计某个模型执行涉及哪些算子，可在模型执行之前调用aclopStartDumpArgs接口，在模型执行之后调用aclopStopDumpArgs接口，接口调用成功后，在path参数指定的目录下生成每个算子shape的算子信息文件。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

aclError aclopStopDumpArgs(uint32_t dumpType)

参数说明

参数名	输入/输出	说明
dumpType	输入	指定dump信息的类型。 当前仅支持ACL_OP_DUMP_OP_AICORE_ARGS，表示统计所有算子信息。 #define ACL_OP_DUMP_OP_AICORE_ARGS 0x00000001U

返回值说明

返回0表示成功，返回其它值表示失败。

11.16 CBLAS 接口

11.16.1 总体说明

- 调用本章节接口实现“矩阵-矩阵”或“矩阵-向量”相乘的运算，本章中的接口内部封装了系统内置的矩阵乘算子GEMM，接口调用流程参见**6.2 单算子调用流程**。

  按照**6.2 单算子调用流程**，在使用本章节的接口前，需要先使用ATC（Ascend Tensor Compiler）工具将内置的矩阵乘算子GEMM的算子描述信息（包括输入输出Tensor描述、算子属性等）编译成适配昇腾AI处理器的离线模型（*.om文件），用于验证矩阵乘算子GEMM的运行结果。
- 本章的CBLAS接口都是异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功。调用该接口后，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保任务已执行完成。

11.16.2 关于 A、B、C 矩阵的 Shape 及内存大小计算公式

- A、B、C矩阵是否转置的标记如果设置为ACL_TRANS_N或ACL_TRANS_T，则用户在申请内存存放A、B、C矩阵数据时，实际申请的内存要和实际数据大小匹配，Shape及内存大小的计算公式如下：
  - A矩阵：shape = (m, k)；内存大小 = m*k*sizeof(dataTypeA)
  - B矩阵：shape = (k, n)；内存大小 = k*n*sizeof(dataTypeB)
  - C矩阵：shape = (m, n)；内存大小 = m*n*sizeof(dataTypeC)
- A、B、C矩阵是否转置的标记如果设置为ACL_TRANS_NZ，表示采用内部数据格式，矩阵Shape为4维，Shape及内存大小的计算公式如下（假设m,k,n分别为原始轴）：
  - 当矩阵A和矩阵B中数据的类型为**aclFloat16**，在计算实际内存大小时，m、k、n均按16对齐向上取整计算：
    - A矩阵：shape = (⌈k/16⌉, ⌈m/16⌉, 16, 16)；内存大小 = ⌈m/16⌉*16*⌈k/16⌉*16*sizeof(dataTypeA)
    - B矩阵：shape = (⌈n/16⌉, ⌈k/16⌉, 16, 16)；内存大小 = ⌈k/16⌉*16*⌈n/16⌉*16*sizeof(dataTypeB)
    - C矩阵：shape = (⌈n/16⌉, ⌈m/16⌉, 16, 16)；内存大小 = ⌈m/16⌉*16*⌈n/16⌉*16*sizeof(dataTypeC)
  - 当矩阵A和矩阵B中数据的类型为int8_t，在计算实际内存大小时，reduce轴按32对齐向上取整计算，非reduce轴按16对齐向上取整计算：
    - A矩阵：shape = (⌈k/32⌉, ⌈m/16⌉, 16, 32)；内存大小 = ⌈m/16⌉*16*⌈k/32⌉*32*sizeof(dataTypeA)
    - B矩阵：shape = (⌈k/32⌉, ⌈n/16⌉, 32, 16)；内存大小 = ⌈k/32⌉*32*⌈n/16⌉*16*sizeof(dataTypeB)

- C矩阵：shape = (⌈n/16⌉, ⌈m/16⌉, 16, 16)；内存大小 = ⌈m/16⌉*16*⌈n/16⌉*16*sizeof(dataTypeC)

说明

⌈ ⌉表示向上对齐。

11.16.3 aclblasGemvEx

函数功能

执行矩阵-向量的乘法，y = αAx + βy，输入数据、输出数据的数据类型通过入参设置，异步接口。

约束说明

A、x、y的数据类型支持仅支持以下组合， α和β的数据类型与y一致。

A的数据类型	x的数据类型	y的数据类型
aclFloat16	aclFloat16	aclFloat16
aclFloat16	aclFloat16	float(float32)
int8_t	int8_t	float(float32)
int8_t	int8_t	int32_t

函数原型

```
aclError aclblasGemvEx(aclTransType transA,
int m,
int n,
const void *alpha,
const void *a,
int lda,
aclDataType dataTypeA,
const void *x,
int incx,
aclDataType dataTypeX,
const void *beta,
void *y,
int incy,
aclDataType dataTypeY,
```

```
aclComputeType type,
aclrtStream stream);
```

参数说明

参数名	输入/输出	说明
transA	输入	A矩阵是否转置的标记。
m	输入	矩阵A的行数，存储矩阵乘数据时，行优先。
n	输入	矩阵A的列数。
alpha	输入	用于执行乘操作的标量α的指针。
a	输入	矩阵A的指针。
lda	输入	A矩阵的主维，此时选择转置，按行优先，则lda为A的列数。预留参数，当前只能设置为-1。
dataTypeA	输入	矩阵A的数据类型。
x	输入	向量x的指针。
incx	输入	x连续元素之间的步长。 预留参数，当前只能设置为-1。
dataTypeX	输入	向量x的数据类型。
beta	输入	用于执行乘操作的标量β的指针。
y	输入&输出	向量y的指针。
incy	输入	y连续元素之间的步长。 预留参数，当前只能设置为-1。
dataTypeY	输入	向量y的数据类型。
type	输入	计算精度，默认高精度。
stream	输入	执行算子所在的Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程，参见6.2 单算子调用流程。

11.16.4 aclblasCreateHandleForGemvEx

函数功能

创建矩阵-向量乘的handle，输入数据、输出数据的数据类型通过入参设置，同步接口。

创建handle成功后，需调用**aclopExecWithHandle**接口执行算子。

约束说明

A、x、y的数据类型支持仅支持以下组合：

A的数据类型	x的数据类型	y的数据类型
aclFloat16	**aclFloat16**	**aclFloat16**
aclFloat16	**aclFloat16**	float(float32)
int8_t	int8_t	float(float32)
int8_t	int8_t	int32_t

函数原型

```
aclError aclblasCreateHandleForGemvEx(aclTransType transA,
int m,
int n,
aclDataType dataTypeA,
aclDataType dataTypeX,
aclDataType dataTypeY,
aclComputeType type,
aclopHandle **handle)
```

参数说明

参数名	输入/输出	说明
transA	输入	A矩阵是否转置的标记。
m	输入	矩阵A的行数，存储矩阵乘数据时，行优先。
n	输入	矩阵A的列数。
dataTypeA	输入	矩阵A的数据类型。
dataTypeX	输入	向量x的数据类型。

参数名	输入/输出	说明
dataTypeY	输入	向量y的数据类型。
type	输入	计算精度，默认高精度。
handle	输出	“算子执行handle的指针”的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.16.5 aclblasHgemv

函数功能

执行矩阵-向量的乘法，y = αAx + βy，输入数据和输出数据的数据类型为aclFloat16，异步接口：

函数原型

```
aclError aclblasHgemv(aclTransType transA,
int m,
int n,
const aclFloat16 *alpha,
const aclFloat16 *a,
int lda,
const aclFloat16 *x,
int incx,
const aclFloat16 *beta,
aclFloat16 *y,
int incy,
aclComputeType type,
aclrtStream stream)
```

参数说明

参数名	输入/输出	说明
transA	输入	A矩阵是否转置的标记。

参数名	输入/输出	说明
m	输入	矩阵A的行数，存储矩阵乘数据时，行优先。
n	输入	矩阵A的列数。
alpha	输入	用于执行乘操作的标量α的指针。
a	输入	矩阵A的指针。
lda	输入	A矩阵的主维，此时选择转置，按行优先，则lda为A的列数。预留参数，当前只能设置为-1。
x	输入	向量x的指针。
incx	输入	x连续元素之间的步长。 预留参数，当前只能设置为-1。
beta	输入	用于执行乘操作的标量β的指针。
y	输入&输出	向量y的指针。
incy	输入	y连续元素之间的步长。 预留参数，当前只能设置为-1。
type	输入	计算精度，默认高精度。
stream	输入	执行算子所在的Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程，参见**6.2 单算子调用流程**。

11.16.6 aclblasCreateHandleForHgemv

函数功能

创建矩阵-向量乘的handle，输入数据和输出数据的数据类型为**aclFloat16**，同步接口。

创建handle成功后，需调用**aclopExecWithHandle**接口执行算子。

函数原型

aclError aclblasCreateHandleForHgemv(aclTransType transA,

int m,

```
int n,
aclComputeType type,
aclopHandle **handle)
```

参数说明

参数名	输入/输出	说明
transA	输入	A矩阵是否转置的标记。
m	输入	矩阵A的行数，存储矩阵乘数据时，行优先。
n	输入	矩阵A的列数。
type	输入	计算精度，默认高精度。
handle	输出	“算子执行handle的指针”的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.16.7 aclblasS8gemv

函数功能

执行矩阵-向量的乘法，y = αAx + βy，输入数据的数据类型为int8_t，输出数据的数据类型为int32_t，异步接口：

函数原型

```
aclError aclblasS8gemv(aclTransType transA,
int m,
int n,
const int32_t *alpha,
const int8_t *a,
int lda,
const int8_t *x,
int incx,
const int32_t *beta,
int32_t *y,
int incy,
aclComputeType type,
```

aclrtStream stream)

参数说明

参数名	输入/输出	说明
transA	输入	A矩阵是否转置的标记。
m	输入	矩阵A的行数，存储矩阵乘数据时，行优先。
n	输入	矩阵A的列数。
alpha	输入	用于执行乘操作的标量α的指针。
a	输入	矩阵A的指针。
lda	输入	A矩阵的主维，此时选择转置，按行优先，则lda为A的列数。预留参数，当前只能设置为-1。
x	输入	向量x的指针。
incx	输入	x连续元素之间的步长。 预留参数，当前只能设置为-1。
beta	输入	用于执行乘操作的标量β的指针。
y	输入&输出	向量y的指针。
incy	输入	y连续元素之间的步长。 预留参数，当前只能设置为-1。
type	输入	计算精度，默认高精度。
stream	输入	执行算子所在的Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程，参见**6.2 单算子调用流程**。

11.16.8 aclblasCreateHandleForS8gemv

函数功能

创建矩阵-向量乘的handle，输入数据的数据类型为int8_t，输出数据的数据类型为int32_t，同步接口。

创建handle成功后，需调用**aclopExecWithHandle**接口执行算子。

函数原型

```
aclError aclblasCreateHandleForS8gemv(aclTransType transA,
int m,
int n,
aclComputeType type,
aclopHandle **handle)
```

参数说明

参数名	输入/输出	说明
transA	输入	A矩阵是否转置的标记。
m	输入	矩阵A的行数，存储矩阵乘数据时，行优先。
n	输入	矩阵A的列数。
type	输入	计算精度，默认高精度。
handle	输出	“算子执行handle的指针”的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.16.9 aclblasGemmEx

函数功能

执行矩阵-矩阵的乘法，C = αAB + βC，输入数据、输出数据的数据类型通过入参设置，异步接口：

约束说明

A、B、C的数据类型支持仅支持以下组合， α和β的数据类型与C一致。

A的数据类型	B的数据类型	C的数据类型
aclFloat16	aclFloat16	aclFloat16
aclFloat16	aclFloat16	float(float32)
int8_t	int8_t	float(float32)
int8_t	int8_t	int32_t

函数原型

```
aclError aclblasGemmEx(aclTransType transA,
aclTransType transB,
aclTransType transC,
int m,
int n,
int k,
const void *alpha,
const void *matrixA,
int lda,
aclDataType dataTypeA,
const void *matrixB,
int ldb,
aclDataType dataTypeB,
const void *beta,
void *matrixC,
int ldc,
aclDataType dataTypeC,
aclComputeType type,
aclrtStream stream)
```

参数说明

参数名	输入/输出	说明
transA	输入	A矩阵是否转置的标记。
transB	输入	B矩阵是否转置的标记。
transC	输入	C矩阵的标记，当前仅支持ACL_TRANS_N。
m	输入	矩阵A的行数与矩阵C的行数。
n	输入	矩阵B的列数与矩阵C的列数。
k	输入	矩阵A的列数与矩阵B的行数。
alpha	输入	用于执行乘操作的标量α的指针。
matrixA	输入	矩阵A的指针。

参数名	输入/输出	说明
lda	输入	A矩阵的主维，此时选择转置，按行优先，则lda为A的列数。预留参数，当前只能设置为-1。
dataTypeA	输入	矩阵A的数据类型。
matrixB	输入	矩阵B的指针。
ldb	输入	B矩阵的主维，此时选择转置，按行优先，则ldb为B的列数。预留参数，当前只能设置为-1。
dataTypeB	输入	矩阵B的数据类型。
beta	输入	用于执行乘操作的标量β的指针。
matrixC	输入&输出	矩阵C的指针。
ldc	输入	C矩阵的主维，预留参数，当前只能设置为-1。
dataTypeC	输入	矩阵C的数据类型。
type	输入	计算精度，默认高精度。
stream	输入	执行算子所在的Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

- 接口调用及示例流程，参见**6 单算子调用**。

11.16.10 aclblasCreateHandleForGemmEx

函数功能

创建矩阵-矩阵乘的handle，输入数据、输出数据的数据类型通过入参设置，同步接口。

创建handle成功后，需调用**aclopExecWithHandle**接口执行算子。

约束说明

A、B、C的数据类型支持仅支持以下组合：

A的数据类型	B的数据类型	C的数据类型
aclFloat16	**aclFloat16**	**aclFloat16**

A的数据类型	B的数据类型	C的数据类型
aclFloat16	aclFloat16	float(float32)
int8_t	int8_t	float(float32)
int8_t	int8_t	int32_t

函数原型

```
aclError aclblasCreateHandleForGemmEx(aclTransType transA,
aclTransType transB,
aclTransType transC,
int m,
int n,
int k,
aclDataType dataTypeA,
aclDataType dataTypeB,
aclDataType dataTypeC,
aclComputeType type,
aclopHandle **handle)
```

参数说明

参数名	输入/输出	说明
transA	输入	A矩阵是否转置的标记。
transB	输入	B矩阵是否转置的标记。
transC	输入	C矩阵的标记，当前仅支持ACL_TRANS_N。
m	输入	矩阵A的行数与矩阵C的行数。
n	输入	矩阵B的列数与矩阵C的列数。
k	输入	矩阵A的列数与矩阵B的行数。
dataTypeA	输入	矩阵A的数据类型。
dataTypeB	输入	矩阵B的数据类型。
dataTypeC	输入	矩阵C的数据类型。
type	输入	计算精度，默认高精度。

参数名	输入/输出	说明
handle	输出	“算子执行handle的指针”的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.16.11 aclblasHgemm

函数功能

执行矩阵-矩阵的乘法，C = αAB + βC，输入数据和输出数据的数据类型为aclFloat16，异步接口：

函数原型

```
aclError aclblasHgemm(aclTransType transA,
aclTransType transB,
aclTransType transC,
int m,
int n,
int k,
const aclFloat16 *alpha,
const aclFloat16 *matrixA,
int lda,
const aclFloat16 *matrixB,
int ldb,
const aclFloat16 *beta,
aclFloat16 *matrixC,
int ldc,
aclComputeType type,
aclrtStream stream)
```

参数说明

参数名	输入/输出	说明
transA	输入	A矩阵是否转置的标记。
transB	输入	B矩阵是否转置的标记。
transC	输入	C矩阵的标记，当前仅支持ACL_TRANS_N。
m	输入	矩阵A的行数与矩阵C的行数。
n	输入	矩阵B的列数与矩阵C的列数。
k	输入	矩阵A的列数与矩阵B的行数。
alpha	输入	用于执行乘操作的标量α的指针。
matrixA	输入	矩阵A的指针。
lda	输入	A矩阵的主维，此时选择转置，按行优先，则lda为A的列数。预留参数，当前只能设置为-1。
matrixB	输入	矩阵B的指针。
ldb	输入	B矩阵的主维，此时选择转置，按行优先，则ldb为B的列数。预留参数，当前只能设置为-1。
beta	输入	用于执行乘操作的标量β的指针。
matrixC	输入&输出	矩阵C的指针。
ldc	输入	C矩阵的主维，预留参数，当前只能设置为-1。
type	输入	计算精度，默认高精度。
stream	输入	执行算子所在的Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程，参见**6.2 单算子调用流程**。

11.16.12 aclblasCreateHandleForHgemm

函数功能

创建矩阵-矩阵乘的handle，输入数据和输出数据的数据类型为**aclFloat16**，同步接口。

创建handle成功后，需调用**aclopExecWithHandle**接口执行算子。

函数原型

```
aclError aclblasCreateHandleForHgemm(aclTransType transA,
aclTransType transB,
aclTransType transC,
int m,
int n,
int k,
aclComputeType type,
aclopHandle **handle)
```

参数说明

参数名	输入/输出	说明
transA	输入	A矩阵是否转置的标记。
transB	输入	B矩阵是否转置的标记。
transC	输入	C矩阵的标记，当前仅支持ACL_TRANS_N。
m	输入	矩阵A的行数与矩阵C的行数。
n	输入	矩阵B的列数与矩阵C的列数。
k	输入	矩阵A的列数与矩阵B的行数。
type	输入	计算精度，默认高精度。
handle	输出	“算子执行handle的指针”的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.16.13 aclblasS8gemm

函数功能

执行矩阵-矩阵的乘法，C = αAB + βC，输入数据的数据类型为int8_t，输出数据的数据类型为int32_t，异步接口：

函数原型

```
aclError aclblasS8gemm(aclTransType transA,
aclTransType transB,
```

```
aclTransType transC,
int m,
int n,
int k,
const int32_t *alpha,
const int8_t *matrixA,
int lda,
const int8_t *matrixB,
int ldb,
const int32_t *beta,
int32_t *matrixC,
int ldc,
aclComputeType type,
aclrtStream stream)
```

参数说明

参数名	输入/输出	说明
transA	输入	A矩阵是否转置的标记。
transB	输入	B矩阵是否转置的标记。
transC	输入	C矩阵的标记，当前仅支持ACL_TRANS_N。
m	输入	矩阵A的行数与矩阵C的行数。
n	输入	矩阵B的列数与矩阵C的列数。
k	输入	矩阵A的列数与矩阵B的行数。
alpha	输入	用于执行乘操作的标量α的指针。
matrixA	输入	矩阵A的指针。
lda	输入	A矩阵的主维，此时选择转置，按行优先，则lda为A的列数。预留参数，当前只能设置为-1。
matrixB	输入	矩阵B的指针。
ldb	输入	B矩阵的主维，此时选择转置，按行优先，则ldb为B的列数。预留参数，当前只能设置为-1。
beta	输入	用于执行乘操作的标量β的指针。
matrixC	输入&输出	矩阵C的指针。

参数名	输入/输出	说明
ldc	输入	C矩阵的主维，预留参数，当前只能设置为-1。
type	输入	计算精度，默认高精度。
stream	输入	执行算子所在的Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程，参见6.2 单算子调用流程。

11.16.14 aclblasCreateHandleForS8gemm

函数功能

创建矩阵-矩阵乘的handle，输入数据的数据类型为int8_t，输出数据的数据类型为int32_t，同步接口。

创建handle成功后，需调用aclopExecWithHandle接口执行算子。

函数原型

```
aclError aclblasCreateHandleForS8gemm(aclTransType transA,
aclTransType transB,
aclTransType transC,
int m,
int n,
int k,
aclComputeType type,
aclopHandle **handle)
```

参数说明

参数名	输入/输出	说明
transA	输入	A矩阵是否转置的标记。
transB	输入	B矩阵是否转置的标记。
transC	输入	C矩阵的标记，当前仅支持ACL_TRANS_N。

参数名	输入/输出	说明
m	输入	矩阵A的行数与矩阵C的行数。
n	输入	矩阵B的列数与矩阵C的列数。
k	输入	矩阵A的列数与矩阵B的行数。
type	输入	计算精度，默认高精度。
handle	输出	“算子执行handle的指针”的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.16.15 aclopCast

函数功能

转换输入数据的数据类型，异步接口。

函数原型

```
aclError aclopCast(const aclTensorDesc *srcDesc,
const aclDataBuffer *srcBuffer,
const aclTensorDesc *dstDesc,
aclDataBuffer *dstBuffer,
uint8_t truncate,
aclrtStream stream)
```

参数说明

参数名	输入/输出	说明
srcDesc	输入	输入tensor描述的指针。
srcBuffer	输入	输入tensor的指针。
dstDesc	输入	输出tensor描述的指针。
dstBuffer	输出	输出tensor的指针。
truncate	输入	预留。
stream	输入	执行算子所在的Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程，参见6.2 单算子调用流程。

11.16.16 aclopCreateHandleForCast

函数功能

创建数据类型转换的handle，同步接口。

创建handle成功后，需调用aclopExecWithHandle接口执行算子。

函数原型

```
aclError aclopCreateHandleForCast(aclTensorDesc *srcDesc,
aclTensorDesc *dstDesc,
uint8_t truncate,
aclopHandle **handle)
```

参数说明

参数名	输入/输出	说明
srcDesc	输入	输入tensor描述的指针。
dstDesc	输入	输出tensor描述的指针。
truncate	输入	预留。
handle	输出	“算子执行handle的指针”的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.17 媒体数据处理 V1

11.17.1 总体说明

多版本接口差异

- 本手册中媒体数据处理V1版本与媒体数据处理V2版本的接口都是描述处理媒体数据的接口，用于实现视频编解码、图像编解码、图像处理等功能，但两套接口不能混用。
  - V2版本的功能比V1版本更多，如下：
    - JPEGE：V2版本接口支持高级的参数配置，如huffman表配置。
    - VENC：V2版本接口支持更加细化的码控参数配置和效果调优，如I/P帧QP、宏块码控等。
    - VDEC：V2版本接口支持更细化的内存控制，如设置输入码流缓存。
  - 建议使用V2版本中的接口，保证后续版本接口功能以及业务的连续演进。
  - V1版本中的接口是为了兼容旧版本，保证使用该部分接口的用户能继续使用，后续版本不再演进。
- 在Atlas 200/300/500 推理产品上，本手册中的媒体数据处理接口与《**DVPP API参考**》中的接口都是描述处理媒体数据的接口，用于实现抠图、图片缩放、格式转换等功能，但两套接口不能混用，两者的区别在于，本手册中媒体数据处理接口支持在**标准形态**和**Control CPU开放形态**下调用，《**DVPP API参考**》中接口仅支持在**Control CPU开放形态**下调用。
  - 建议使用本手册中的媒体数据处理V2版本接口，保证后续版本接口功能以及业务的连续演进。
  - 《**DVPP API参考**》中的接口是为了兼容旧版本，保证使用该部分接口的用户能继续使用。

功能说明

本章节介绍媒体数据处理V1版本（DVPP，Digital Vision Pre-Processing）的功能，如**表11-9**所示。

表 11-9 功能说明

功能	功能
VPC（Vision Preprocessing Core）	负责图像处理功能，支持对图片做抠图、缩放、格式转换等操作，详细描述请参见**11.17.5.2 约束说明**。
JPEGD（JPEG Decoder）	负责完成图像解码功能，将.jpg、.jpeg、.JPG、.JPEG图片解码成YUV格式图片，详细描述请参见**11.17.6.1 功能及约束说明**。
JPEGE（JPEG Encoder）	负责完成图像编码功能，将YUV格式图片编码成.jpg图片，详细描述请参见**11.17.7.1 功能及约束说明**。
VDEC（Video Decoder）	负责视频解码，详细描述请参见**11.17.9.1 功能及约束说明**。

功能	功能
VENC（Video Encoder）	负责视频编码，详细描述请参见11.17.10.1 功能及约束说明。
PNGD（PNG Decoder）	负责PNG格式图片的解码，详细描述请参见11.17.8.1 功能及约束说明。

功能支持度说明

昇腾AI处理器对媒体数据处理V1版本各功能的支持度如表11-10所示。

表 11-10 功能支持度说明

昇腾AI处理器	VPC	JPEGD	JPEGE	PNGD	VDEC	VENC
Atlas 200/300/500 推理产品	√	√	√	√	√	√
Atlas 推理系列产品	√	√	√	√	√	√

整体约束说明

使用本章中介绍的接口，有以下注意点：

- 关于异步接口：

  对于本章介绍的异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功，对于有依赖的接口，为确保能按序执行任务，建议用户在多个接口中指定同一个stream，因为同一个stream中的任务按接口调用顺序执行。

  在调用异步口对图片进行解码、抠图、缩放等操作时，如果任务之间有依赖，一定要调用**aclrtSynchronizeStream**接口确保在同一个Stream中的任务按序执行。

  从性能角度考虑，建议一个stream上下发多个异步媒体数据处理任务后，执行一次**aclrtSynchronizeStream**接口。

  调用异步接口后，不能马上释放资源，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保Device侧任务执行完成后才能释放。

- 关于内存申请/释放：

  a. 实现媒体数据处理的VPC、JPEGD、JPEGE等功能前，若需要申请Device上的内存存放输入或输出数据，需调用**acldvppMalloc**申请内存、调用**acldvppFree**接口释放内存。如果多个功能串联使用的场景，需要复用同一段内存，则按最大内存要求来申请内存。

  b. 调用**a**申请出来的内存可以满足媒体数据处理的要求，也可以在其它任务中使用，例如，从性能角度，为了减少拷贝，媒体数据处理的输出作为模型推理的输入，实现内存复用。

  c. 但由于媒体数据处理访问的地址空间有限，为确保媒体数据处理时内存足够，除媒体数据处理功能外的其它功能（例如，模型加载），建议调用内存

管理章节下的aclrtMalloc接口、或aclrtMallocHost接口、或aclrtMallocCached接口申请内存。

Atlas 推理系列产品上，媒体数据处理功能在每个进程内，可访问的地址空间最大为16GB。

- 关于通道的要求：

  实现媒体数据处理的各功能前，必须调用接口创建对应功能的通道，创建通道的接口请参见11.17.4 通道创建与释放。通道的创建与销毁会涉及资源的申请与释放，反复创建与销毁通道会影响业务性能，因此建议根据实际场景管理通道，例如，如果有持续VPC图片处理，则创建VPC的通道后，等到所有VPC功能调用完成后，再销毁该VPC通道。

  通道数量多，会影响Device的CPU占用率和内存占用，通道数量建议参考各功能章节下的的性能指标的路数。

  在Atlas 推理系列产品上，各功能的通道数有一定的限制，VPC的通道数最多256，JPEGD与VDEC共用通道且通道数最多256，JPEGE与VENC共用通道且通道数最多256，PNGD的通道数最多128。创建通道的接口请参见11.17.4 通道创建与释放。

  说明

  对于Atlas 推理系列产品，昇腾虚拟化实例（Ascend Virtual Instance）场景下，如果通道总数不为整数，则向下取整：

  - VPC通道总数 =（被分配的VPC硬件单元数量 / VPC硬件单元总数）* 256
  - VDEC通道和JPEGD通道总数 =（被分配的VDEC硬件单元+JPEGD硬件单元数量之和 / VDEC硬件单元+JPEGD硬件单元总数之和）* 256
  - VENC通道和JPEGE通道总数 =（被分配的VENC硬件单元+JPEGE硬件单元数量之和 / VENC硬件单元+JPEGE硬件单元总数之和）* 256
  - PNGD通道总数 = 切分规格 * 128

    对于PNGD功能，若使用以下昇腾虚拟化实例（Ascend Virtual Instance）模板，则对通道数的限制不同：

    - 使用vir04_4c_dvpp模板时，通道总数固定为128。
    - 使用vir04_3c_ndvpp模板时，表示不使用DVPP功能，因此通道总数为0。

  昇腾虚拟化实例（Ascend Virtual Instance）的详细描述，以及各场景下的算力资源规格请参见《昇腾虚拟化实例（AVI）用户指南》。

11.17.2 基本概念

表 11-11 关键概念列表

概念	描述
宽stride（widthStride）	指一行图像跨距，表示输入/输出图片对齐后的宽，RGB格式和YUV格式的宽stride计算方式不一样。 各功能对宽stride的要求不同，请参见对应功能的约束说明： ● VPC功能，请参见**图片格式、宽高对齐、内存约束**。 ● JPEGD功能，请参见**11.17.6.1 功能及约束说明**。 ● JPEGE功能，请参见**11.17.7.1 功能及约束说明**。 ● PNGD功能，请参见**11.17.8.1 功能及约束说明**。 ● VDEC功能，请参见**11.17.9.1 功能及约束说明**。 ● VENC功能，请参见**11.17.10.1 功能及约束说明**。
高stride（height Stride）	指图像在内存中的行数，表示输入/输出图片对齐后的高。 各功能对高stride的要求不同，请参见对应功能的约束说明： ● VPC功能，请参见**图片格式、宽高对齐、内存约束**。 ● JPEGD功能，请参见**11.17.6.1 功能及约束说明**。 ● JPEGE功能，请参见**11.17.7.1 功能及约束说明**。 ● PNGD功能，请参见**11.17.8.1 功能及约束说明**。 ● VDEC功能，请参见**11.17.9.1 功能及约束说明**。 ● VENC功能，请参见**11.17.10.1 功能及约束说明**。
上/下/左/右偏移	通过配置上偏移、下偏移、左偏移、右偏移可以实现两个功能：指定抠图区域或贴图区域的位置，控制抠图或贴图区域的宽、高，右偏移-左偏移+1=宽，下偏移-上偏移+1=高。参见**图11-1**。
抠图区域	指用户指定的需抠出的图片区域。 抠图区域最小分辨率为10*6，最大分辨率为4096*4096。 抠图区域的约束，请参见**抠图、贴图约束**。
贴图区域	指在输出图片中用户指定的区域，贴图区域最小分辨率为10*6，最大分辨率为4096*4096。 贴图区域的约束，请参见**抠图、贴图约束**。
非8K	非8K表示10*6~4096*4096（包括4096）分辨率范围的输入图片。 非8K场景下，输入图片宽stride、高stride的取值范围如下：（宽stride/系数）在32~4096（包括4096）范围内，高stride在6~4096。 系数与输入图片的格式有关： ● YUV400、YUV420SP、YUV422SP、YUV444SP、YUV440SP：系数为1。 ● YUV422packed：系数为2。 ● YUV444packed、RGB888：系数为3。 ● ARGB8888：系数为4。

概念	描述
8K	8K表示宽或高在4096~8192（不包括4096）范围内的输入图片。 8K场景下，输入图片宽stride、高stride的取值范围如下： Atlas 200/300/500 推理产品上，宽stride在4096~8192范围内或高stride在4096~8192范围内（不包括4096）。 Atlas 推理系列产品上，宽stride在4096~16384范围内（不包括4096）。

11.17.3 内存申请与释放

11.17.3.1 acldvppMalloc

函数功能

该接口主要用于分配内存给Device侧媒体数据处理时使用，申请的大页内存满足数据处理的要求（例如，内存首地址128字节对齐），同步接口。

约束说明

- 调用该接口申请内存后，如果内存不使用，需及时调用**acldvppFree**接口释放内存。
- 频繁调用acldvppMalloc接口申请内存、调用**acldvppFree**接口释放内存，会损耗性能，建议用户提前做内存预先分配或二次管理，避免频繁申请/释放内存。
- 调用acldvppMalloc接口申请内存时，会对用户输入的size按向上对齐成32整数倍后，再多加32字节。
- 媒体数据处理的输出作为模型推理的输入时，若用户使用本接口申请大块内存并自行划分、管理内存时，每段内存需同时满足以下要求：
  - 内存大小向上对齐成32整数倍+32字节（m=ALIGN_UP[len,32]+32字节）；
  - 内存起始地址需满足128字节对齐（ALIGN_UP[m,128]）。

  说明

  len表示某段内存的大小，ALIGN_UP[len,k]表示向上按k字节对齐：((len-1)/k+1)*k。
- 调用该接口申请大页内存失败，仅表示系统内的大页内存不够。关于系统内预留大页的说明请参见**11.11.1 总体说明**。

函数原型

aclError acldvppMalloc(void **devPtr, size_t size)

参数说明

参数名	输入/输出	说明
devPtr	输出	“Device上已分配内存的指针”的指针。
size	输入	申请内存的大小，单位Byte。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**5.4 DVPP图像/视频处理（媒体数据处理V1）**。

11.17.3.2 acldvppFree

函数功能

释放通过**acldvppMalloc**接口申请的内存，同步接口。

函数原型

aclError acldvppFree(void *devPtr)

参数说明

参数名	输入/输出	说明
devPtr	输入	待释放内存的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**5.4 DVPP图像/视频处理（媒体数据处理V1）**。

11.17.4 通道创建与释放

11.17.4.1 VPC/JPEGD/JPEGE/PNGD 图片处理通道

11.17.4.1.1 acldvppCreateChannel

函数功能

创建图片数据处理的通道，同一个通道可以重复使用，销毁后不再可用，同步接口。

约束说明

- 通道为非线程安全，即不同线程要求创建不同的通道。
- 同一个通道不能同时在多个Stream中并发使用。
- acldvppCreateChannel接口内部调用1次**aclrtCreateStream**接口创建了1个Stream，该Stream用于创建通道的任务，创建通道的任务结束后，该Stream资源自动释放。

  用户在创建通道时，**aclrtCreateStream**接口的约束需要遵循，例如Stream的数量等。

函数原型

aclError acldvppCreateChannel(acldvppChannelDesc *channelDesc)

参数说明

参数名	输入/输出	说明
channelDesc	输入&输出	通道描述信息的指针。 需提前调用**acldvppCreateChannelDesc**接口创建acldvppChannelDesc类型的数据。 同时，对于Atlas 推理系列产品，您还需要调用**acldvppSetChannelDescMode**接口指定通道描述信息中的通道模式，明确图片数据处理通道用于实现哪种功能，目前支持**VPC**、**JPEGD**、**JPEGE**、**PNGD**功能。若不指定通道模式，则系统默认4种模式的通道都创建，可能会占用通道资源（通道数的限制，请参见**11.17.1 总体说明**）。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**5.4 DVPP图像/视频处理（媒体数据处理V1）**。

11.17.4.1.2 acldvppDestroyChannel

函数功能

销毁图片数据处理的通道，只能销毁通过**acldvppCreateChannel**接口创建的通道，同步接口。

约束说明

如果在调用acldvppDestroyChannel接口销毁通道前，已调用**acldvppDestroyChannelDesc**接口销毁了通道描述信息，那么在调用acldvppDestroyChannel接口销毁通道时会报错。

函数原型

aclError acldvppDestroyChannel(acldvppChannelDesc *channelDesc)

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针。 与调用**acldvppCreateChannel**接口创建通道时指定的channelDesc保持一致。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**5.4 DVPP图像/视频处理（媒体数据处理V1）**。

11.17.4.2 VDEC 视频解码通道

11.17.4.2.1 aclvdecCreateChannel

函数功能

创建视频解码处理的通道，同一个通道可以重复使用，销毁后不再可用，同步接口。

约束说明

- 通道为非线程安全，即不同线程要求创建不同的通道。
- 通道数的限制，请参见**11.17.1 总体说明**。
- aclvdecCreateChannel接口内部封装了**aclrtCreateStream**接口和**aclrtSubscribeReport**：

- 调用2次**aclrtCreateStream**接口创建了2个Stream，一个Stream用于视频码流解码任务，另一个用于回调函数的处理。这些Stream资源只有在调用**aclvdecDestroyChannel**接口销毁通道后才能自动释放。
- 调用**aclrtSubscribeReport**接口指定处理Stream上回调函数的线程，回调函数和线程是由用户调用**aclvdecSetChannelDesc系列接口**时指定的。

用户在实现VDEC功能时，无需再单独调用**aclrtCreateStream**接口、**aclrtSubscribeReport**接口，但**aclrtCreateStream**接口、**aclrtSubscribeReport**接口的约束需要遵循，例如Stream的数量、处理Stream上回调函数的线程数量等。

函数原型

aclError aclvdecCreateChannel(aclvdecChannelDesc *channelDesc)

参数说明

参数名	输入/输出	说明
channelDesc	输入&输出	通道描述信息的指针。 需提前调用**aclvdecCreateChannelDesc**接口创建aclvdecChannelDesc类型的数据，再调用**aclvdecSetChannelDesc系列接口**设置通道描述信息的属性。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**5.4 DVPP图像/视频处理（媒体数据处理V1）**。

11.17.4.2.2 aclvdecDestroyChannel

函数功能

销毁视频解码处理的通道，只能销毁通过**aclvdecCreateChannel**接口创建的通道，同步接口。

约束说明

- 销毁通道接口会等待已发送帧解码完成且用户的回调函数处理完成后再销毁通道。
- aclvdecDestroyChannel接口内部封装了**aclrtUnSubscribeReport**接口取消线程注册（Stream上的回调函数不再由指定线程处理）、**aclrtDestroyStream**接口销毁Stream。用户在实现VDEC功能时，无需再单独调用**aclrtUnSubscribeReport**接口、**aclrtDestroyStream**接口。

- 如果在调用aclvdecDestroyChannel接口销毁通道前，已调用**aclvdecDestroyChannelDesc**接口销毁了通道描述信息，那么在调用aclvdecDestroyChannel接口销毁通道时会报错。

函数原型

aclError aclvdecDestroyChannel (aclvdecChannelDesc *channelDesc)

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针。 与调用**aclvdecCreateChannel**接口创建通道时指定的channelDesc保持一致。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**5.4 DVPP图像/视频处理（媒体数据处理V1）**。

11.17.4.3 VENC 视频编码通道

11.17.4.3.1 aclvencCreateChannel

函数功能

创建视频编码处理的通道，同一个通道可以重复使用，销毁后不再可用，同步接口。

约束说明

- 通道为非线程安全，即不同线程要求创建不同的通道。
- 通道数的限制，请参见**11.17.1 总体说明**。
- aclvencCreateChannel接口内部调用2次**aclrtCreateStream**接口创建了2个Stream，一个Stream用于视频码流编码任务，另一个Stream用于回调函数的处理。这些Stream资源只有在调用**aclvencDestroyChannel**接口销毁通道后才能自动释放。

  用户在实现VENC功能时，**aclrtCreateStream**接口的约束需要遵循，例如Stream的数量等。

函数原型

aclError aclvencCreateChannel(aclvencChannelDesc *channelDesc)

参数说明

参数名	输入/输出	说明
channelDesc	输入&输出	通道描述信息的指针。 需提前调用**aclvencCreateChannelDesc**接口创建aclvencChannelDesc类型的数据，再调用**aclvencSetChannelDesc系列接口**设置通道描述信息的属性。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**5.4 DVPP图像/视频处理（媒体数据处理V1）**。

11.17.4.3.2 aclvencDestroyChannel

函数功能

销毁视频编码处理的通道，只能销毁通过**aclvencCreateChannel**接口创建的通道，同步接口。

约束说明

- 销毁通道接口会等待用户的回调函数处理完成后再销毁。
- 如果在调用aclvencDestroyChannel接口销毁通道前，已调用**aclvencDestroyChannelDesc**接口销毁了通道描述信息，那么在调用aclvencDestroyChannel接口销毁通道时会报错。

函数原型

aclError aclvencDestroyChannel(aclvencChannelDesc *channelDesc)

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针 与调用**aclvencCreateChannel**接口创建通道时指定的channelDesc保持一致。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**5.4 DVPP图像/视频处理（媒体数据处理V1）**。

11.17.5 VPC 功能

11.17.5.1 功能说明

功能说明

VPC（Vision Preprocessing Core）各功能的介绍、各版本的功能支持度如下表所示。本章的接口都是异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功。调用该接口后，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保任务已执行完成。

功能	描述	Atlas 200/300/500 推理产品	Atlas 推理系列产品
抠图	从输入图片中抠出需要用的图片区域，支持一图多框和多图多框。	√	√
缩放	• 针对不同分辨率的图像，支持**8K**缩放、**非8K**缩放。 • 支持单图裁剪缩放（支持非压缩格式）、一图多框裁剪缩放（支持非压缩格式）。 • 其它缩放方式，如：原图缩放、等比例缩放（缩放前后图片的宽高比例相同）。	√	√
叠加	从输入图片中抠出来的图，对抠出的图进行缩放后，放在用户输出图片的指定区域，输出图片可以是空白图片（由用户申请的空输出内存产生的），也可以是已有图片（由用户申请输出内存后将已有图片读入输出内存），只有当输出图片是已有图片时，才表示叠加。	√	√
拼接	从输入图片中抠多张图片，对抠出的图进行缩放后，放到输出图片的指定区域。	√	√
图像金字塔	对原图进行多次高斯滤波之后向下采样产生不同尺寸的图像。	x	√

功能	描述	Atlas 200/300/500 推理产品	Atlas 推理系列产品
直方图统计	统计图像每个通道（RGB/YUV）的像素值分布。	x	√
色彩重映射	根据配置信息将图片从原图映射为另一张图。	x	√
边界填充	对图像进行边界填充。	x	√
格式转换	支持RGB格式、YUV格式之间的格式转换。	√	√
图像灰度化	将彩色图像转化为灰度图像。需注意，输入为灰度图像、输出只能为灰度图像。	√ 从YUV420SP格式的输出图像数据中只取Y分量的数据	√ 实现图像灰度化的操作是输出YUV400格式的输出图片

功能示意图(Atlas 200/300/500 推理产品)(Atlas 推理系列产品)

图 11-1 VPC 功能示意图（抠图+缩放+叠加）

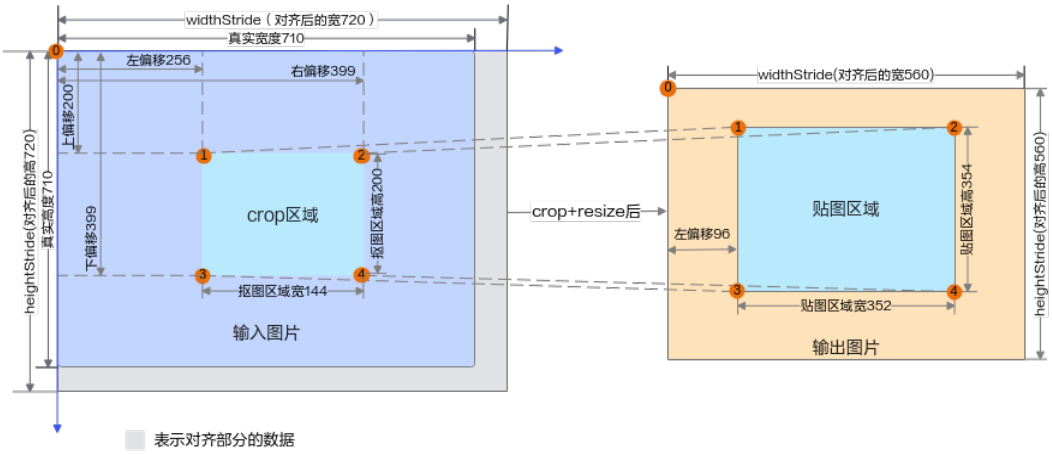


图 11-2 VPC 功能示意图（拼接）

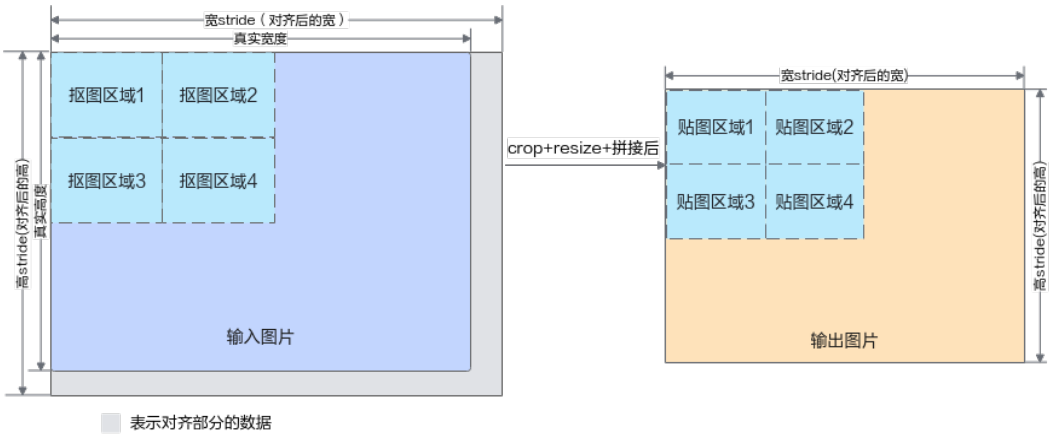


图 11-3 等比例缩放（贴图区域在输出图片的中心位置），即缩放前后图片的宽高比例相同

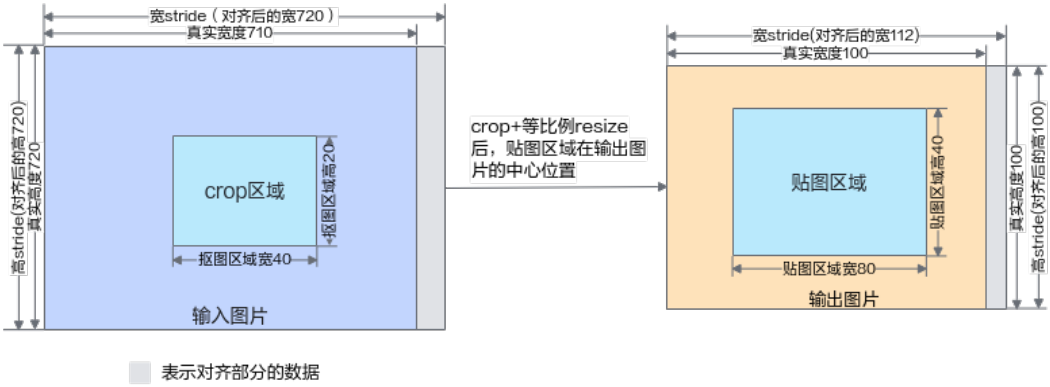


图 11-4 等比例缩放（贴图区域在输出图片的上下居中位置）

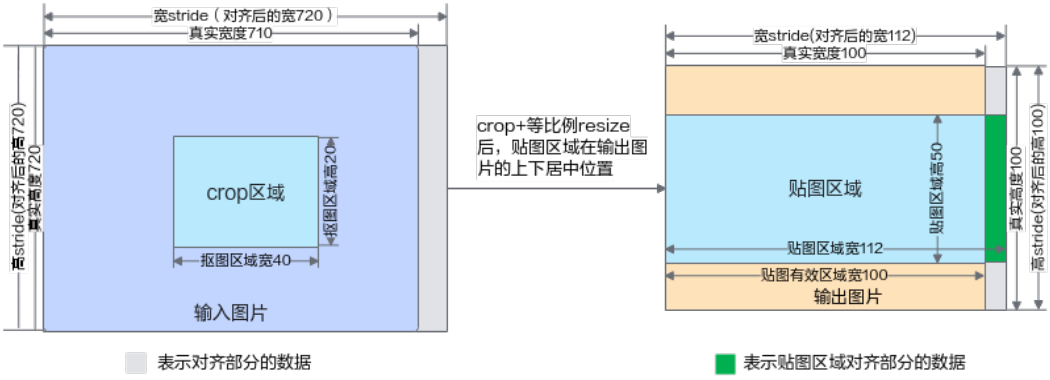
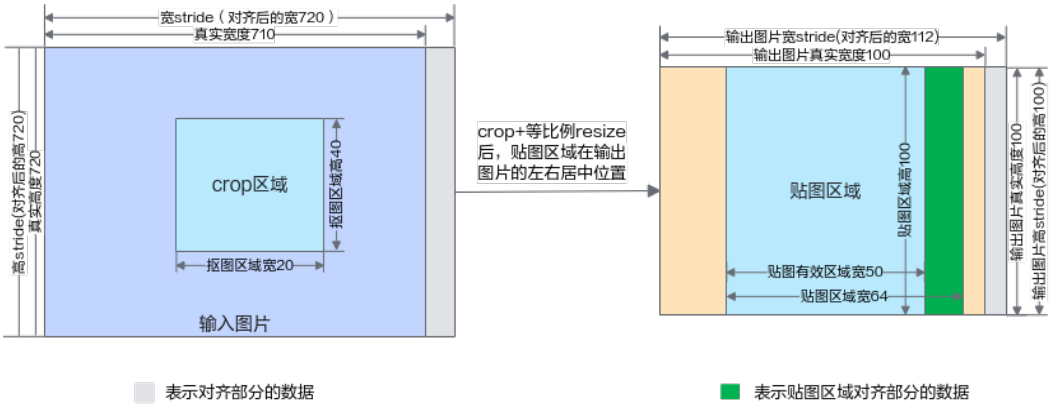


图 11-5 等比例缩放（贴图区域在输出图片的左右居中位置）



参考资源

RGB、YUV格式图像的各分量排布示意图。示例：SP图像以YUV420SP为例，Packed和RGB图像以ARGB图像为例。

格式	分辨率	宽stride	高stride	buffer大小					
yuv420sp	4*4	4	4	24					
内存排列									
y11	y12	y13	y14						
y21	y22	y23	y24						
y31	y32	y33	y34						
y41	y42	y43	y44						
u11	v11	u13	v13						
u31	v31	u33	v33						
格式	分辨率	宽stride	高stride	buffer大小					
yuv420sp	4*4	6	6	54					
内存排列	x为无效数据								
y11	y12	y13	y14	x	x				
y21	y22	y23	y24	x	x				
y31	y32	y33	y34	x	x				
y41	y42	y43	y44	x	x				
x	x	x	x	x	x				
x	x	x	x	x	x				
u11	v11	u13	v13	x	x				
u31	v31	u33	v33	x	x				
x	x	x	x	x	x				
格式	分辨率	宽stride	高stride	buffer大小					
argb	2*2	10	4	40					
内存排列	x为无效数据								
a11	r11	g11	b11	a12	r12	g12	b12	x	x
a21	r21	g21	b21	a22	r22	g22	b22	x	x
x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x

11.17.5.2 约束说明

本节中8K、非8K的概念如下：

- **非8K**：宽*高在10*6~4096*4096范围内，包括4096。
- **8K**：宽或高在4096~8192，不包括4096。

图片分辨率约束

- 输入图片分辨率

  10*6~8192*8192

  在调用接口实现VPC功能时，各接口对分辨率的要求可能不同，请参见**11.17.5 VPC功能**下各接口的说明。其中，当输入图片格式为YUV440SP、YUV440P时，输入图片的宽最大值为4096。
- 输出图片分辨率

版本	分辨率范围
Atlas 200/300/500 推理产品 Atlas 推理系列产品	10*6~4096*4096

图片格式、宽高对齐、内存约束

VPC在处理图片时，需调用**acldvppMalloc**接口申请Device上的输入、输出内存，调用**acldvppFree**接口释放输入、输出内存，这部分内存的生命周期由用户自行管理。

图片格式、宽高对齐约束请参见**表11-12**、**表11-13**。

Atlas 200/300/500 推理产品和Atlas 训练系列产品，在8K场景下，输入图片格式仅支持YUV420SP NV12 8bit和YUV420SP NV21 8bit。

Atlas 推理系列产品，在8K场景下，输入图片格式支持如下格式：

- YUV400 8bit
- YUV420SP NV12 8bit、YUV420SP NV21 8bit
- YUV422SP 8bit、YVU422SP 8bit
- YUV444SP 8bit、YVU444SP 8bit
- YUV422P YUYV 8bit、YUV422P UYVY 8bit、YUV422P YVYU 8bit、YUV422P VYUY 8bit

说明

在调用接口实现VPC功能时：

- 图片格式的定义请参见**acldvppPixelFormat**，宽stride、高stride、8K等概念请参见**11.17.2 基本概念**。
- 不同版本中，宽stride、高stride的最小值、最大值不同，如下：
  - Atlas 200/300/500 推理产品上，宽stride最小32、最大16384（16384=4096*4，宽是4096的argb格式的图像，1个像素占用4个字节，一行像素就占用4096*4，即宽stride）；高stride最小6、最大16384。
  - Atlas 推理系列产品上，宽stride最小32、最大16384（16384=4096*4，宽是4096的argb格式的图像，1个像素占用4个字节，一行像素就占用4096*4，即宽stride）；高stride最小6、最大16384。

表 11-12 输入图片格式、宽高对齐、内存大小约束

版本	输入图片格式	输入图片宽、高对齐要求	输入图片宽stride、高stride、内存大小要求 Atlas 200/300/500 推理产品	输入图片宽stride、高stride、内存大小要求 Atlas 推理系列产品
通用格式，各版本都支持	YUV400 8bit	无对齐要求	宽stride为宽16对齐后的值。 高stride为高2对齐后的值。 内存大小（单位Byte）= 宽stride * 高stride * 3/2	宽stride为宽16对齐后的值。 高stride无对齐要求，与高相同即可。 内存大小（单位Byte）= 宽stride * 高stride
	YUV420SP NV12 8bit	宽2对齐 高2对齐	宽stride为宽16对齐后的值。 高stride为高2对齐后的值。 内存大小（单位Byte）= 宽stride * 高stride * 3/2	宽stride为宽16对齐后的值。 高stride为高2对齐后的值。 内存大小（单位Byte）= 宽stride * 高stride * 3/2
	YUV420SP NV21 8bit			
	YUV422SP 8bit	宽2对齐 高无对齐要求	宽stride为宽16对齐后的值。 高stride为高2对齐后的值。 内存大小（单位Byte）= 宽stride * 高stride * 2	宽stride为宽16对齐后的值。 高stride无对齐要求，与高相同即可。 内存大小（单位Byte）= 宽stride * 高stride * 2
	YVU422SP 8bit			
	YUV444SP 8bit	无对齐要求	宽stride为宽16对齐后的值。 高stride为高2对齐后的值。 内存大小（单位Byte）= 宽stride * 高stride * 3	宽stride为宽16对齐后的值。 高stride无对齐要求，与高相同即可。 内存大小（单位Byte）= 宽stride * 高stride * 3
	YVU444SP 8bit			
	YUV422Packed YUYV 8bit	宽2对齐 高无对齐要求	宽stride为宽16对齐后、再乘以2的值。 高stride为高2对齐后的值。 内存大小（单位Byte）= 宽stride * 高stride	宽stride为宽16对齐后、再乘以2的值。 高stride无对齐要求，与高相同即可。 内存大小（单位Byte）= 宽stride * 高stride
	YUV422Packed UYVY 8bit			

版本	输入图片格式	输入图片宽、高对齐要求	输入图片宽stride、高stride、内存大小要求 Atlas 200/300/500 推理产品	输入图片宽stride、高stride、内存大小要求 Atlas 推理系列产品
	YUV422Packed YVYU 8bit			
	YUV422Packed VYUY 8bit			
	YUV444Packed 8bit	无对齐要求	宽stride为宽16对齐后、再乘以3的值。 高stride为高2对齐后的值。 内存大小（单位Byte）= 宽stride * 高stride	宽stride为宽16对齐后、再乘以3的值。 高stride无对齐要求，与高相同即可。 内存大小（单位Byte）= 宽stride * 高stride
	RGB888			
	BGR888			
	ARGB8888	无对齐要求	宽stride为宽16对齐后、再乘以4的值。 高stride为高2对齐后的值。 内存大小（单位Byte）= 宽stride * 高stride	宽stride为宽16对齐后、再乘以4的值。 高stride无对齐要求，与高相同即可。 内存大小（单位Byte）= 宽stride * 高stride
	ABGR8888			
	RGBA8888			
	BGRA8888			
该部分格式仅以下版本支持： Atlas 推理系列产品	YUV440SP 8bit	宽无对齐要求 高2对齐	-	宽stride为宽16对齐后的值。 高stride为高2对齐后的值。 内存大小（单位Byte）= 宽stride * 高stride * 2
	YVU440SP 8bit			

表 11-13 输出图片格式、宽高对齐、内存大小约束

版本	输出图片格式	输出图片宽、高对齐要求	输出图片宽stride、高stride、内存大小要求 Atlas 200/300/500 推理产品	输出图片宽stride、高stride、内存大小要求 Atlas 推理系列产品
通用格式，各版本都支持	YUV420SP NV12 8bit	宽2对齐 高2对齐	宽stride为宽16对齐后的值。 高stride为高2对齐后的值。 内存大小（单位Byte）= 宽stride * 高stride * 3/2	宽stride为宽16对齐后的值。 高stride为高2对齐后的值。 内存大小（单位Byte）= 宽stride * 高stride * 3/2
	YUV420SP NV21 8bit			
该部分格式仅以下版本支持： Atlas 推理系列产品	YUV400 8bit	无对齐要求	-	宽stride为宽16对齐后的值。 高stride无对齐要求，与高相同即可。 内存大小（单位Byte）= 宽stride * 高stride
	YUV422SP 8bit	宽2对齐 高无对齐要求		宽stride为宽16对齐后的值。 高stride无对齐要求，与高相同即可。 内存大小（单位Byte）= 宽stride * 高stride * 2
	YVU422SP 8bit			
	YUV444Packed 8bit	无对齐要求		宽stride为宽16对齐后、再乘以3的值。 高stride无对齐要求，与高相同即可。 内存大小（单位Byte）= 宽stride * 高stride
	RGB888			
	BGR888			
	ARGB8888	无对齐要求		宽stride为宽16对齐后、再乘以4的值。 高stride无对齐要求，与高相同即可。 内存大小（单位Byte）= 宽stride * 高stride
	ABGR8888			
	RGBA8888			
	BGRA8888			

抠图、贴图约束

- 抠图区域不超出输入图片区域。

- 贴图区域不超出输出图片区域，最大贴图个数256个。

  贴图区域相对输出图片的左偏移16对齐，贴图时可直接放置在输出图片的最左侧，即相对输出图片的左偏移为0，0也满足16对齐的要求。
- 输出图片的贴图宽度建议16对齐，如果不是16对齐，会多写一段无效数据使其16对齐。见**图11-4**、**图11-5**，贴图区域旁边的绿色框就表示无效数据。
- 抠图、贴图区域的奇数、偶数限制为：

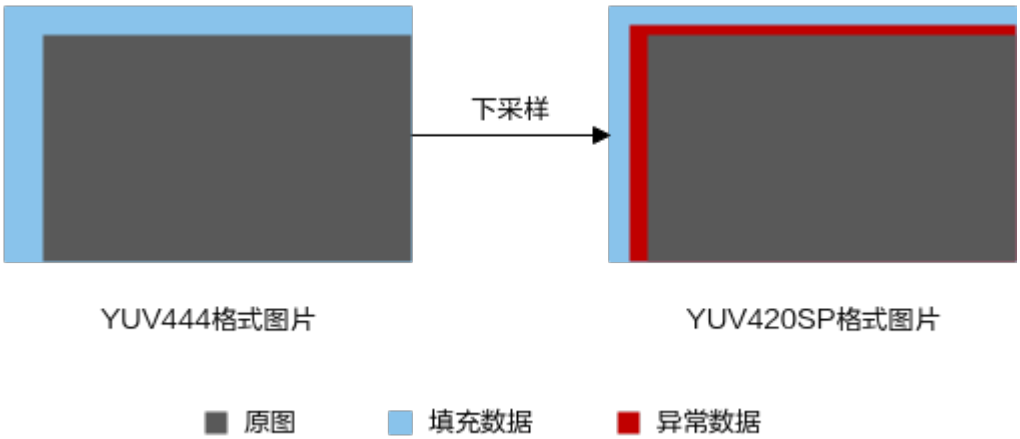
昇腾AI处理器	约束
Atlas 200/300/500 推理产品	输出图片格式为YUV420SP，贴图区域奇数、偶数限制为：左偏移和上偏移为偶数、右偏移和下偏移为奇数。
Atlas 推理系列产品	输出图片格式为YUV420SP，贴图区域奇数、偶数限制为：左偏移和上偏移为偶数、右偏移和下偏移为奇数。 输出图片格式为YUV422SP，贴图区域奇数、偶数限制为：左偏移为偶数、右偏移为奇数。 其它格式，贴图区域没有偏移奇偶数的限制。

- Atlas 200/300/500 推理产品上，缩放场景下，贴图/抠图的宽或高缩放比例范围：[1/32, 16]。
- 在等比例缩放场景下，由于贴图左偏移16对齐的限制，如果直接调用**acldvppVpcCropResizePasteAsync**接口实现缩放功能，贴图的位置可能不在输出图片的正中心位置，此时可通过抠图、缩放、填充几个功能配合使用（调用**acldvppVpcBatchCropResizeMakeBorderAsync**接口），实现等比例缩放。

YUV 格式图像下采样约束

VPC在处理图片时，会根据输入或输出图片格式，将输入图片格式转换为YUV444或RGB用于内部处理，YUV444或RGB没有宽高奇偶数的限制，但当输出图片格式为YUV420SP或YUV422SP格式时，会进行下采样处理，由于YUV420SP或YUV422SP格式本身的数据排布导致宽高存在奇偶数限制，因此输出图片的边缘可能存在异常数据。

图 11-6 异常效果图片举例



出现异常数据的根因在于，在计算过程中，出现输出图片位置处于奇数起始点时，此时YUV444格式的图片是正确的，但是下采样到YUV420SP格式时，由于奇数行和偶数

行是共用同一个uv的，导致图片起始行的Y与上一行的UV组成新的像素，产生异常数据。

精度相关约束

VPC的输出作为模型推理的输入时：

在Atlas 200/300/500 推理产品和Atlas 训练系列产品上，VPC功能中，贴图区域的宽*高有16*2对齐的约束，贴图有效区域的宽*高不满足16*2对齐时，贴图区域中会存在一些补边的无效数据，为保证推理精度，贴图有效区域的宽*高不满足16*2对齐时，用户需借助AIPP抠出有效区域，去除无效数据。

在Atlas 推理系列产品上，VPC功能中，贴图区域的宽有16对齐的约束，贴图有效区域的宽不满足16对齐时，贴图区域中会存在一些补边的无效数据，为保证推理精度，贴图有效区域的宽不满足16对齐时，用户需借助AIPP抠出有效区域，去除无效数据。

11.17.5.3 性能指标说明

本节仅描述各产品对应的昇腾AI处理器的性能指标参考数据，各产品由于其内存、CPU等配置不同，在性能规格上会存在差异，详细产品的性能规格请查阅昇腾社区的官方说明。

本节以单个昇腾AI处理器为例，描述基本场景性能指标参考，如下表所示，其它分辨率可以等量估算。

性能指标说明（Atlas 200/300/500 推理产品）

以下性能数据，是基于一个stream上下发10个异步媒体数据处理任务后，执行一次**aclrtSynchronizeStream**接口。下表中的1路对应一个通道，一个通道对应一个线程。

- 对于**非8K**图像：

  例如1080p的图像，若存在Host->Device的图片数据拷贝，由于拷贝带宽限制，最大总帧率约为1000fps。

  例如4K的图像，若存在Host->Device的图片数据拷贝，由于拷贝带宽限制，最大总帧率约为250fps。

表 11-14 Atlas 200/300/500 推理产品的昇腾 AI 处理器性能指标参考数据

场景举例	总帧率
• 输入图像分辨率：1080p（1920*1080） • 输出图像分辨率：1080p（1920*1080） • 输入/输出图片格式：YUV420SP • n路（n<4）	n*360fps

场景举例	总帧率
● 输入图像分辨率：1080p（1920*1080） ● 输出图像分辨率：1080p（1920*1080） ● 输入/输出图片格式：YUV420SP ● n路（n≥4）	1440fps
● 输入图像分辨率：4K图像（3840*2160） ● 输出图像分辨率：4K图像（3840*2160） ● 输入/输出图片格式：YUV420SP ● n路（n<4）	n*90fps
● 输入图像分辨率：4K图像（3840*2160） ● 输出图像分辨率：4K图像（3840*2160） ● 输入/输出图片格式：YUV420SP ● n路（n≥4）	360fps

- 对于**8K**图像：

表 11-15 Atlas 200/300/500 推理产品的昇腾 AI 处理器性能指标参考数据

场景举例	总帧率
● 输入图像分辨率：8K图像（7680*4320） ● 输出图像分辨率：1080p（1920*1080） ● 输入/输出图片格式：YUV420SP ● n路（n<4）	n*25fps
● 输入图像分辨率：8K图像（7680*4320） ● 输出图像分辨率：1080p（1920*1080） ● 输入/输出图片格式：YUV420SP ● n路（n≥4）	100fps

场景举例	总帧率
• 输入图像分辨率：8K图像（7680*4320） • 输出图像分辨率：4K图像（3840*2160） • 输入/输出图片格式：YUV420SP • n路（n<4）	n*25fps
• 输入图像分辨率：8K图像（7680*4320） • 输出图像分辨率：4K图像（3840*2160） • 输入/输出图片格式：YUV420SP • n路（n≥4）	100fps

说明

- VPC处理性能与处理过程中的图像分辨率强相关，以输入图像和输出图像中的最大分辨率作为基准分辨率，基准分辨率越大，处理耗时越久，性能越低。
- 调用VPC批处理接口（接口命名中包含Batch，例如acldvppVpcBatchCropAsync接口）时，由于图像处理单元DVPP（Digital Video Pre-Processing）内部多个VPC硬件单元会并行处理图片任务，因此单路就可以达到最大总帧率。

性能指标说明（Atlas 推理系列产品）

以下性能数据，是基于一个stream上下发64个异步媒体数据处理任务后，执行一次**aclrtSynchronizeStream**接口，且Host服务器配置为：CPU主频为Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz，CPU核心数为14。

下表中的1路对应一个通道，一个通道对应一个线程。

表 11-16 Atlas 推理系列产品的昇腾 AI 处理器性能指标参考数据

场景举例	总帧率
• 输入图片分辨率：1080p（1920*1080） • 输出图片分辨率：1080p（1920*1080） • 输入/输出图片格式：YUV420SP • n路（1≤n≤12）	n*360fps

场景举例	总帧率
● 输入图片分辨率：1080p（1920*1080） ● 输出图片分辨率：1080p（1920*1080） ● 输入/输出图片格式：YUV420SP ● n路（n>12）	4320fps
● 输入图片分辨率：4K图像（3840*2160） ● 输出图片分辨率：4K图像（3840*2160） ● 输入/输出图片格式：YUV420SP ● n路（1≤n≤12）	n*90fps
● 输入图片分辨率：4K图像（3840*2160） ● 输出图片分辨率：4K图像（3840*2160） ● 输入/输出图片格式：YUV420SP ● n路（n>12）	1080fps
● 输入图片分辨率：8K图像（7680*4320） ● 输出图片分辨率：4K图像（3840*2160） ● 输入/输出图片格式：YUV420SP ● n路（1≤n≤12）	n*22.5fps
● 输入图片分辨率：8K图像（7680*4320） ● 输出图片分辨率：4K图像（3840*2160） ● 输入/输出图片格式：YUV420SP ● n路（n>12）	270fps

说明

调用VPC批处理接口（接口命名中包含Batch，例如acldvppVpcBatchCropAsync接口）时，由于图像处理单元DVPP（Digital Video Pre-Processing）内部多个VPC硬件单元会并行处理图片任务，因此单路就可以达到最大总帧率。

11.17.5.4 acldvppVpcResizeAsync

函数功能

将输入图片缩放到输出图片大小，异步接口。

约束说明

版本	约束

Atlas 200/300/500 推理产品	● 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**acldvppPixelFormat**枚举值中的如下枚举项：
	PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	● 关于输出图片分辨率、图片格式请参见**图片格式、宽高对齐、内存约束**。
	● 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。

版本	约束

Atlas 推理系列产品	● 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**acldvppPixelFormat**枚举值中的如下枚举项：
	PIXEL_FORMAT_YUV_400 = 0, // YUV400 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3, // YUV422SP 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4, // YVU422SP 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5, // YUV444SP 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6, // YVU444SP 8bit
	PIXEL_FORMAT_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	PIXEL_FORMAT_UYVY_PACKED_422 = 8, // YUV422Packed UYVY 8bit
	PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 8bit
	PIXEL_FORMAT_VYUY_PACKED_422 = 10, // YUV422Packed VYUY 8bit
	● 关于输出图片分辨率、图片格式请参见**图片格式、宽高对齐、内存约束**。
	● 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。
	● 当输入、输出图片格式都为ARGB8888、ABGR8888、RGBA8888、BGRA8888格式时，在经过本接口缩放处理后，A分量值会变为0。

函数原型

aclError acldvppVpcResizeAsync(acldvppChannelDesc *channelDesc,

acldvppPicDesc *inputDesc,

acldvppPicDesc *outputDesc,

acldvppResizeConfig *resizeConfig,

aclrtStream stream)

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针。 与调用**acldvppCreateChannel**接口创建通道时指定的channelDesc保持一致。

参数名	输入/输出	说明
inputDesc	输入	输入图片信息的指针。 ● 调用**acldvppCreatePicDesc**接口创建图片描述信息。 ● 调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。 **说明** 如果输入图片的宽高不是2对齐，VPC会自动处理成向下2对齐，例如输入图片的宽高为15*15，VPC会先处理成14*14，再进行缩放。
outputDesc	输入&输出	输出图片信息的指针。 outputDesc参数作为输入时，需要用户调用如下接口： ● 调用**acldvppCreatePicDesc**接口创建图片描述信息。 ● 调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。
resizeConfig	输入	图片缩放配置数据的指针。 调用**acldvppCreateResizeConfig**接口创建图片缩放配置数据。
stream	输入	指定Stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

参考资源

接口调用流程及示例，参见**5.4.1 VPC图像处理典型功能**

11.17.5.5 acldvppVpcCropAsync

函数功能

按指定区域从一张输入图片中抠出一张子图，再将抠的图片存放到输出内存中，作为输出图片，异步接口。

输出图片区域与cropArea不一致时会对图片再做一次缩放操作。

Atlas 200/300/500 推理产品，默认缩放算法为“自研的高阶滤波算法”。

Atlas 推理系列产品，默认缩放算法为“业界通用的Bilinear算法”。

约束说明

版本	约束

Atlas 200/300/500 推理产品	● 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**acldvppPixelFormat**枚举值中的如下枚举项：
	PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	● 关于输出图片分辨率、图片格式请参见**图片格式、宽高对齐、内存约束**。
	● 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。
	● 由于通过本接口抠的图直接作为输出图片，所以需满足输出图片最大分辨率4096*4096的限制，因此本接口中抠图区域最大分辨率为4096*4096。

Atlas 推理系列产品	● 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**acldvppPixelFormat**枚举值中的如下枚举项：
	PIXEL_FORMAT_YUV_400 = 0, // YUV400 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3, // YUV422SP 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4, // YVU422SP 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5, // YUV444SP 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6, // YVU444SP 8bit
	PIXEL_FORMAT_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	PIXEL_FORMAT_UYVY_PACKED_422 = 8, // YUV422Packed UYVY 8bit
	PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 8bit
	PIXEL_FORMAT_VYUY_PACKED_422 = 10, // YUV422Packed VYUY 8bit
	● 关于输出图片分辨率、图片格式请参见**图片格式、宽高对齐、内存约束**。
	● 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。
	● 由于通过本接口抠的图直接作为输出图片，所以需满足输出图片最大分辨率4096*4096的限制，因此本接口中抠图区域最大分辨率为4096*4096。

函数原型

```
aclError acldvppVpcCropAsync(acldvppChannelDesc *channelDesc,
acldvppPicDesc *inputDesc,
acldvppPicDesc *outputDesc,
acldvppRoiConfig *cropArea,
aclrtStream stream)
```

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针。 与调用acldvppCreateChannel接口创建通道时指定的channelDesc保持一致。
inputDesc	输入	输入图片信息的指针。 ● 调用acldvppCreatePicDesc接口创建图片描述信息。 ● 调用acldvppSetPicDesc系列接口设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。
outputDesc	输入&输出	输出图片信息的指针。 outputDesc参数作为输入时，需要用户调用如下接口： ● 调用acldvppCreatePicDesc接口创建图片描述信息。 ● 调用acldvppSetPicDesc系列接口设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。
cropArea	输入	抠图区域位置的指针。 调用acldvppCreateRoiConfig接口创建区域位置数据。
stream	输入	指定Stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

参考资源

接口调用流程及示例，参见5.4.1 VPC图像处理典型功能

11.17.5.6 acldvppVpcCropResizeAsync

函数功能

按指定区域从一张输入图片中抠出一张子图，再将抠的图片存放到输出内存中，作为输出图片，异步接口。

输出图片区域与cropArea不一致时会对图片再做一次缩放操作，按照resizeConfig处配置的缩放算法。

约束说明

版本	约束

Atlas 200/300/500 推理产品	● 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**acldvppPixelFormat**枚举值中的如下枚举项：
	PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	● 关于输出图片分辨率、图片格式请参见**图片格式、宽高对齐、内存约束**。
	● 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。

版本	约束

Atlas 推理系列产品	● 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**acldvppPixelFormat**枚举值中的如下枚举项：
	PIXEL_FORMAT_YUV_400 = 0, // YUV400 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3, // YUV422SP 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4, // YVU422SP 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5, // YUV444SP 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6, // YVU444SP 8bit
	PIXEL_FORMAT_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	PIXEL_FORMAT_UYVY_PACKED_422 = 8, // YUV422Packed UYVY 8bit
	PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 8bit
	PIXEL_FORMAT_VYUY_PACKED_422 = 10, // YUV422Packed VYUY 8bit
	● 关于输出图片分辨率、图片格式请参见**图片格式、宽高对齐、内存约束**。
	● 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。
	● 当输入、输出图片格式都为ARGB8888、ABGR8888、RGBA8888、BGRA8888格式时，在经过本接口缩放处理后，A分量值会变为0。

函数原型

aclError acldvppVpcCropResizeAsync(acldvppChannelDesc *channelDesc,

acldvppPicDesc *inputDesc,

acldvppPicDesc *outputDesc,

acldvppRoiConfig *cropArea,

acldvppResizeConfig *resizeConfig,

aclrtStream stream)

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针。 与调用**acldvppCreateChannel**接口创建通道时指定的channelDesc保持一致。
inputDesc	输入	输入图片信息的指针。 ● 调用**acldvppCreatePicDesc**接口创建图片描述信息。 ● 调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。
outputDesc	输入&输出	输出图片信息的指针。 outputDesc参数作为输入时，需要用户调用如下接口： ● 调用**acldvppCreatePicDesc**接口创建图片描述信息。 ● 调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。
cropArea	输入	抠图区域位置的指针。 调用**acldvppCreateRoiConfig**接口创建区域位置数据。
resizeConfig	输入	图片缩放配置数据的指针。 需提前调用**acldvppCreateResizeConfig**接口创建图片缩放配置数据。
stream	输入	指定Stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

参考资源

接口调用示例，参见**抠图缩放（一图一框）示例代码**。

11.17.5.7 acldvppVpcBatchCropAsync

函数功能

对于一张或多张输入图片，支持从每个输入图片中抠出一张或多张子图并一一对应存放到多个输出区域，输出图片区域与抠图区域不一致时会对图片再做一次缩放操作。异步接口。

Atlas 200/300/500 推理产品，默认缩放算法为“自研的高阶滤波算法”。

Atlas 推理系列产品，默认缩放算法为“业界通用的Bilinear算法”。

约束说明

版本	约束

Atlas 200/300/500 推理产品	● 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**acldvppPixelFormat**枚举值中的如下枚举项：
	PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	● 关于输出图片分辨率、图片格式请参见**图片格式、宽高对齐、内存约束**。
	● 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。

Atlas 推理系列产品	● 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**acldvppPixelFormat**枚举值中的如下枚举项：
	PIXEL_FORMAT_YUV_400 = 0, // YUV400 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3, // YUV422SP 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4, // YVU422SP 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5, // YUV444SP 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6, // YVU444SP 8bit
	PIXEL_FORMAT_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	PIXEL_FORMAT_UYVY_PACKED_422 = 8, // YUV422Packed UYVY 8bit
	PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 8bit
	PIXEL_FORMAT_VYUY_PACKED_422 = 10, // YUV422Packed VYUY 8bit
	● 关于输出图片分辨率、图片格式请参见**图片格式、宽高对齐、内存约束**。
	● 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。

函数原型

aclError acldvppVpcBatchCropAsync(acldvppChannelDesc *channelDesc,

acldvppBatchPicDesc *srcBatchPicDescs,

uint32_t *roiNums,

uint32_t size,

acldvppBatchPicDesc *dstBatchPicDescs,

acldvppRoiConfig *cropAreas[],

aclrtStream stream)

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针。 与调用**acldvppCreateChannel**接口创建通道时指定的channelDesc保持一致。
srcBatchPicDescs	输入	批量输入图片描述信息的指针。 ● 调用**acldvppCreateBatchPicDesc**接口创建批量图片描述信息。 ● 调用**acldvppGetPicDesc**接口获取指定图片的图片描述信息。 ● 调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。
roiNums	输入	抠图数量的指针，当前最大抠图数量为256。 roiNums为数组，数组总和小于等于256，与dstBatchPicDescs结构体中的batchSize值保持一致。 roiNums[0]+...+roiNums[size-1] <= 256
size	输入	表示roiNums数组中的元素个数，个数小于等于256。
dstBatchPicDescs	输入&输出	批量输出图片描述信息的指针。 dstBatchPicDescs参数作为输入时，需要用户调用如下接口： ● 调用**acldvppCreateBatchPicDesc**接口创建批量图片描述信息。 ● 调用**acldvppGetPicDesc**接口获取指定图片的图片描述信息。 ● 调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。
cropAreas	输入	抠图区域位置的指针数组。 调用**acldvppCreateRoiConfig**接口创建区域位置数据。

参数名	输入/输出	说明
stream	输入	指定Stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

11.17.5.8 acldvppVpcBatchCropResizeAsync

函数功能

对于一张或多张输入图片，支持从每个输入图片中抠出一张或多张子图并一一对应存放到多个输出区域，输出图片区域与抠图区域不一致时会对图片再做一次缩放操作，按照resizeConfig处配置的缩放算法。异步接口。

约束说明

版本	约束

Atlas 200/300/500 推理产品	• 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**acldvppPixelFormat**枚举值中的如下枚举项：
	PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	• 关于输出图片分辨率、图片格式请参见**图片格式、宽高对齐、内存约束**。
	• 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。

版本	约束

Atlas 推理系列产品	● 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**acldvppPixelFormat**枚举值中的如下枚举项：
	PIXEL_FORMAT_YUV_400 = 0, // YUV400 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3, // YUV422SP 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4, // YVU422SP 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5, // YUV444SP 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6, // YVU444SP 8bit
	PIXEL_FORMAT_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	PIXEL_FORMAT_UYVY_PACKED_422 = 8, // YUV422Packed UYVY 8bit
	PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 8bit
	PIXEL_FORMAT_VYUY_PACKED_422 = 10, // YUV422Packed VYUY 8bit
	● 关于输出图片分辨率、图片格式请参见**图片格式、宽高对齐、内存约束**。
	● 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。
	● 当输入、输出图片格式都为ARGB8888、ABGR8888、RGBA8888、BGRA8888格式时，在经过本接口缩放处理后，A分量值会变为0。

函数原型

```
aclError acldvppVpcBatchCropResizeAsync(acldvppChannelDesc *channelDesc,
acldvppBatchPicDesc *srcBatchPicDescs,
uint32_t *roiNums,
uint32_t size,
acldvppBatchPicDesc *dstBatchPicDescs,
acldvppRoiConfig *cropAreas[],
acldvppResizeConfig *resizeConfig,
aclrtStream stream)
```

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针。 与调用**acldvppCreateChannel**接口创建通道时指定的channelDesc保持一致。
srcBatchPicDescs	输入	批量输入图片描述信息的指针。 ● 调用**acldvppCreateBatchPicDesc**接口创建批量图片描述信息。 ● 调用**acldvppGetPicDesc**接口获取指定图片的图片描述信息。 ● 调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。
roiNums	输入	抠图数量的指针，当前最大抠图数量为256。 roiNums为数组，数组总和小于等于256，与dstBatchPicDescs结构体中的batchSize值保持一致。 roiNums[0]+...+roiNums[size-1] <= 256
size	输入	表示roiNums数组中的元素个数，个数小于等于256。
dstBatchPicDescs	输入&输出	批量输出图片描述信息的指针。 dstBatchPicDescs参数作为输入时，需要用户调用如下接口： ● 调用**acldvppCreateBatchPicDesc**接口创建批量图片描述信息。 ● 调用**acldvppGetPicDesc**接口获取指定图片的图片描述信息。 ● 调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。
cropAreas	输入	抠图区域位置的指针数组。 调用**acldvppCreateRoiConfig**接口创建区域位置数据。
resizeConfig	输入	图片缩放配置数据的指针。 需提前调用**acldvppCreateResizeConfig**接口创建图片缩放配置数据。
stream	输入	指定Stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

11.17.5.9 acldvppVpcCropAndPasteAsync

函数功能

按指定区域从一个输入图片中抠出一张子图，再将抠的子图贴到目标图片的指定位置，作为输出图片。异步接口。

当pasteArea的宽高与cropArea的宽高不一致时会对图片做一次缩放操作。

Atlas 200/300/500 推理产品，默认缩放算法为“自研的高阶滤波算法”。

Atlas 推理系列产品，默认缩放算法为“业界通用的Bilinear算法”。

约束说明

版本	约束

Atlas 200/300/500 推理产品	● 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**acldvppPixelFormat**枚举值中的如下枚举项：
	PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	● 关于输出图片分辨率、图片格式请参见**图片格式、宽高对齐、内存约束**。
	● 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。

版本	约束

Atlas 推理系列产品	● 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**acldvppPixelFormat**枚举值中的如下枚举项：
	PIXEL_FORMAT_YUV_400 = 0, // YUV400 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3, // YUV422SP 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4, // YVU422SP 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5, // YUV444SP 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6, // YVU444SP 8bit
	PIXEL_FORMAT_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	PIXEL_FORMAT_UYVY_PACKED_422 = 8, // YUV422Packed UYVY 8bit
	PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 8bit
	PIXEL_FORMAT_VYUY_PACKED_422 = 10, // YUV422Packed VYUY 8bit
	● 关于输出图片分辨率、图片格式请参见**图片格式、宽高对齐、内存约束**。
	● 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。

函数原型

aclError acldvppVpcCropAndPasteAsync(acldvppChannelDesc *channelDesc,

acldvppPicDesc *inputDesc,

acldvppPicDesc *outputDesc,

acldvppRoiConfig *cropArea,

acldvppRoiConfig *pasteArea,

aclrtStream stream)

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针。 与调用**acldvppCreateChannel**接口创建通道时指定的channelDesc保持一致。

参数名	输入/输出	说明
inputDesc	输入	输入图片信息的指针。 ● 调用**acldvppCreatePicDesc**接口创建图片描述信息。 ● 调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。
outputDesc	输入&输出	输出图片信息的指针。 outputDesc参数作为输入时，需要用户调用如下接口： ● 调用**acldvppCreatePicDesc**接口创建图片描述信息。 ● 调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。
cropArea	输入	抠图区域位置的指针。 调用**acldvppCreateRoiConfig**接口创建区域位置数据。
pasteArea	输入	贴图区域位置的指针。 ● 调用**acldvppCreateRoiConfig**接口创建区域位置数据。 ● 贴图区域左偏移需要16对齐。
stream	输入	指定Stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

11.17.5.10 acldvppVpcCropResizePasteAsync

函数功能

按指定区域从一个输入图片中抠出一张子图，再将抠的子图贴到目标图片的指定位置，作为输出图片。异步接口。

当pasteArea的宽高与cropArea的宽高不一致时会对图片做一次缩放操作，按照resizeConfig处配置的缩放算法。

约束说明

版本	约束

Atlas 200/300/500 推理产品	● 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**acldvppPixelFormat**枚举值中的如下枚举项：
	PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	● 关于输出图片分辨率、图片格式请参见**图片格式、宽高对齐、内存约束**。
	● 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。

Atlas 推理系列产品	● 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**acldvppPixelFormat**枚举值中的如下枚举项：
	PIXEL_FORMAT_YUV_400 = 0, // YUV400 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3, // YUV422SP 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4, // YVU422SP 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5, // YUV444SP 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6, // YVU444SP 8bit
	PIXEL_FORMAT_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	PIXEL_FORMAT_UYVY_PACKED_422 = 8, // YUV422Packed UYVY 8bit
	PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 8bit
	PIXEL_FORMAT_VYUY_PACKED_422 = 10, // YUV422Packed VYUY 8bit
	● 关于输出图片分辨率、图片格式请参见**图片格式、宽高对齐、内存约束**。
	● 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。
	● 当输入、输出图片格式都为ARGB8888、ABGR8888、RGBA8888、BGRA8888格式时，在经过本接口缩放处理后，A分量值会变为0。

函数原型

```
aclError acldvppVpcCropResizePasteAsync(acldvppChannelDesc *channelDesc,
acldvppPicDesc *inputDesc,
acldvppPicDesc *outputDesc,
acldvppRoiConfig *cropArea,
acldvppRoiConfig *pasteArea,
acldvppResizeConfig *resizeConfig,
aclrtStream stream)
```

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针。 与调用**acldvppCreateChannel**接口创建通道时指定的channelDesc保持一致。
inputDesc	输入	输入图片信息的指针。 ● 调用**acldvppCreatePicDesc**接口创建图片描述信息。 ● 调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。
outputDesc	输入&输出	输出图片信息的指针。 outputDesc参数作为输入时，需要用户调用如下接口： ● 调用**acldvppCreatePicDesc**接口创建图片描述信息。 ● 调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。
cropArea	输入	抠图区域位置的指针。 调用**acldvppCreateRoiConfig**接口创建区域位置数据。
pasteArea	输入	贴图区域位置的指针。 ● 调用**acldvppCreateRoiConfig**接口创建区域位置数据。 ● 贴图区域左偏移需要16对齐。
resizeConfig	输入	图片缩放配置数据的指针。 需提前调用**acldvppCreateResizeConfig**接口创建图片缩放配置数据。
stream	输入	指定Stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

参考资源

接口调用示例，参见**抠图缩放（一图一框）示例代码**。

11.17.5.11 acldvppVpcBatchCropAndPasteAsync

函数功能

按指定区域从一张或多张输入图片中抠出一个或多个子图，再将抠的子图贴到目标图片的指定位置，作为输出图片。异步接口。

当pasteArea的宽高与cropArea的宽高不一致时会对图片做一次缩放操作。

Atlas 200/300/500 推理产品，默认缩放算法为“自研的高阶滤波算法”。

Atlas 推理系列产品，默认缩放算法为“业界通用的Bilinear算法”。

约束说明

版本	约束

Atlas 200/300/500 推理产品	• 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**acldvppPixelFormat**枚举值中的如下枚举项：
	PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	• 关于输出图片分辨率、图片格式请参见**图片格式、宽高对齐、内存约束**。
	• 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。

版本	约束

Atlas 推理系列产品	● 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**acldvppPixelFormat**枚举值中的如下枚举项：
	PIXEL_FORMAT_YUV_400 = 0, // YUV400 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3, // YUV422SP 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4, // YVU422SP 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5, // YUV444SP 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6, // YVU444SP 8bit
	PIXEL_FORMAT_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	PIXEL_FORMAT_UYVY_PACKED_422 = 8, // YUV422Packed UYVY 8bit
	PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 8bit
	PIXEL_FORMAT_VYUY_PACKED_422 = 10, // YUV422Packed VYUY 8bit
	● 关于输出图片分辨率、图片格式请参见**图片格式、宽高对齐、内存约束**。
	● 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。

函数原型

aclError acldvppVpcBatchCropAndPasteAsync (acldvppChannelDesc *channelDesc,

acldvppBatchPicDesc *srcBatchPicDescs,

uint32_t *roiNums,

uint32_t size,

acldvppBatchPicDesc *dstBatchPicDescs,

acldvppRoiConfig *cropAreas[],

acldvppRoiConfig *pasteAreas[],

aclrtStream stream)

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针。 与调用**acldvppCreateChannel**接口创建通道时指定的channelDesc保持一致。
srcBatchPicDescs	输入	指定批量输入图片描述信息的指针。 ● 调用**acldvppCreateBatchPicDesc**接口创建批量图片描述信息。 ● 调用**acldvppGetPicDesc**接口获取指定图片的图片描述信息。 ● 调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。
roiNums	输入	每个输入图片抠图数量的指针，当前最大抠图数量为256。 roiNums为数组，数组总和小于等于256，与dstBatchPicDescs结构体中的batchSize值保持一致。 roiNums[0]+...+roiNums[size-1] <= 256
size	输入	表示roiNums数组中的元素个数，个数小于等于256。
dstBatchPicDescs	输入&输出	批量输出图片描述信息的指针。 dstBatchPicDescs参数作为输入时，需要用户调用如下接口： ● 调用**acldvppCreateBatchPicDesc**接口创建批量图片描述信息。 ● 调用**acldvppGetPicDesc**接口获取指定图片的图片描述信息。 ● 调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。
cropAreas	输入	抠图区域位置的指针数组。 ● 调用**acldvppCreateRoiConfig**接口创建区域位置数据。 ● cropAreas、pasteAreas数组中的元素个数与dstBatchPicDescs结构体中的batchSize值相等。

参数名	输入/输出	说明
pasteAreas	输入	贴图区域位置的指针数组。 ● 调用**acldvppCreateRoiConfig**接口创建区域位置数据。 ● 贴图区域左偏移需要16对齐。 ● cropAreas、pasteAreas数组中的元素个数与dstBatchPicDescs参数中的batchSize值相等。
stream	输入	指定Stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

参考资源

接口调用示例，参见**抠图贴图（一图多框）示例代码**。

11.17.5.12 acldvppVpcBatchCropResizePasteAsync

函数功能

按指定区域从一张或多张输入图片中抠出一个或多个子图，再将抠的子图贴到目标图片的指定位置，作为输出图片。异步接口。

当pasteArea的宽高与cropArea的宽高不一致时会对图片做一次缩放操作，按照resizeConfig处配置的缩放算法。

约束说明

版本	约束

Atlas 200/300/500 推理产品	● 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**acldvppPixelFormat**枚举值中的如下枚举项：
	PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	● 关于输出图片分辨率、图片格式请参见**图片格式、宽高对齐、内存约束**。
	● 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。

Atlas 推理系列产品	● 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**acldvppPixelFormat**枚举值中的如下枚举项：
	PIXEL_FORMAT_YUV_400 = 0, // YUV400 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3, // YUV422SP 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4, // YVU422SP 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5, // YUV444SP 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6, // YVU444SP 8bit
	PIXEL_FORMAT_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	PIXEL_FORMAT_UYVY_PACKED_422 = 8, // YUV422Packed UYVY 8bit
	PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 8bit
	PIXEL_FORMAT_VYUY_PACKED_422 = 10, // YUV422Packed VYUY 8bit
	● 关于输出图片分辨率、图片格式请参见**图片格式、宽高对齐、内存约束**。
	● 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。
	● 当输入、输出图片格式都为ARGB8888、ABGR8888、RGBA8888、BGRA8888格式时，在经过本接口缩放处理后，A分量值会变为0。

函数原型

```
aclError acldvppVpcBatchCropResizePasteAsync(acldvppChannelDesc *channelDesc,
acldvppBatchPicDesc *srcBatchPicDescs,
uint32_t *roiNums,
uint32_t size,
acldvppBatchPicDesc *dstBatchPicDescs,
acldvppRoiConfig *cropAreas[],
acldvppRoiConfig *pasteAreas[],
acldvppResizeConfig *resizeConfig,
aclrtStream stream)
```

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针。 与调用acldvppCreateChannel接口创建通道时指定的channelDesc保持一致。
srcBatchPicDescs	输入	批量输入图片描述信息的指针。 ● 调用acldvppCreateBatchPicDesc接口创建批量图片描述信息。 ● 调用acldvppGetPicDesc接口获取指定图片的图片描述信息。 ● 调用acldvppSetPicDesc系列接口设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。
roiNums	输入	每个输入图片抠图数量的指针，当前最大抠图数量为256。 roiNums为数组，数组总和小于等于256，与dstBatchPicDescs结构体中的batchSize值保持一致。 roiNums[0]+...+roiNums[size-1] <= 256
size	输入	表示roiNums数组中的元素个数，个数小于等于256。

参数名	输入/输出	说明
dstBatchPicDescs	输入&输出	批量输出图片描述信息的指针。 dstBatchPicDescs参数作为输入时，需要用户调用如下接口： ● 调用**acldvppCreateBatchPicDesc**接口创建批量图片描述信息。 ● 调用**acldvppGetPicDesc**接口获取指定图片的图片描述信息。 ● 调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。
cropAreas	输入	抠图区域位置的指针数组。 ● 调用**acldvppCreateRoiConfig**接口创建区域位置数据。 ● cropAreas、pasteAreas数组中的元素个数与dstBatchPicDescs结构体中的batchSize值相等。
pasteAreas	输入	贴图区域位置的指针数组。 ● 调用**acldvppCreateRoiConfig**接口创建区域位置数据。 ● 贴图区域左偏移需要16对齐。 ● cropAreas、pasteAreas数组中的元素个数与dstBatchPicDescs参数中的batchSize值相等。
resizeConfig	输入	图片缩放配置数据的指针。 需提前调用**acldvppCreateResizeConfig**接口创建图片缩放配置数据。
stream	输入	指定Stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

11.17.5.13 acldvppVpcBatchCropResizeMakeBorderAsync

函数功能

支持从一张或多张输入图片中抠出一张或多张子图，对每个子图进行缩放，扩边填充后输出至目标图像内存地址。异步接口。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

版本	约束

Atlas 200/300/500 推理产品	● 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**acldvppPixelFormat**枚举值中的如下枚举项：
	PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	● 关于输出图片分辨率、图片格式请参见**图片格式、宽高对齐、内存约束**。
	● 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。
	● 使用本接口实现等比例缩放功能时，相关的描述及约束请参见**抠图、贴图约束**。
	● 由于**YUV格式图像下采样约束**，当输出图片格式为YUV420SP或YUV422SP格式时，需注意：
	– 对于YUV420SP输出格式，上下左右填充的尺寸建议为偶数；
	– 对于YUV422SP输出格式，左右填充的尺寸建议为偶数。

版本	约束

Atlas 推理系列产品	● 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**acldvppPixelFormat**枚举值中的如下枚举项：
	PIXEL_FORMAT_YUV_400 = 0, // YUV400 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3, // YUV422SP 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4, // YVU422SP 8bit
	PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5, // YUV444SP 8bit
	PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6, // YVU444SP 8bit
	PIXEL_FORMAT_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	PIXEL_FORMAT_UYVY_PACKED_422 = 8, // YUV422Packed UYVY 8bit
	PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 8bit
	PIXEL_FORMAT_VYUY_PACKED_422 = 10, // YUV422Packed VYUY 8bit
	● 关于输出图片分辨率、图片格式请参见**图片格式、宽高对齐、内存约束**。
	● 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。
	● 当输入、输出图片格式都为ARGB8888、ABGR8888、RGBA8888、BGRA8888格式时，在经过本接口缩放处理后，A分量值会变为0。
	● 使用本接口实现等比例缩放功能时，相关的描述及约束请参见**抠图、贴图约束**。
	● 由于**YUV格式图像下采样约束**，当输出图片格式为YUV420SP或YUV422SP格式时，需注意：
	– 对于YUV420SP输出格式，上下左右填充的尺寸建议为偶数；
	– 对于YUV422SP输出格式，左右填充的尺寸建议为偶数。

函数原型

aclError acldvppVpcBatchCropResizeMakeBorderAsync(acldvppChannelDesc *channelDesc,

acldvppBatchPicDesc *srcBatchPicDescs,

uint32_t *roiNums,

uint32_t size,

acldvppBatchPicDesc *dstBatchPicDescs,

acldvppRoiConfig *cropAreas[],

acldvppBorderConfig *borderCfgs[],

acldvppResizeConfig *resizeConfig,

aclrtStream stream)

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针。 与调用**acldvppCreateChannel**接口创建通道时指定的channelDesc保持一致。
srcBatchPicDescs	输入	批量输入图片描述信息的指针。 ● 调用**acldvppCreateBatchPicDesc**接口创建批量图片描述信息。 ● 调用**acldvppGetPicDesc**接口获取指定图片的图片描述信息。 ● 调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。
roiNums	输入	每个输入图片抠图数量的指针，当前最大抠图数量为256。 roiNums为数组，数组总和小于等于256，与dstBatchPicDescs结构体中的batchSize值保持一致。 roiNums[0]+...+roiNums[size-1] <= 256
size	输入	表示roiNums数组中的元素个数，个数小于等于256。
dstBatchPicDescs	输入&输出	批量输出图片描述信息的指针。 dstBatchPicDescs参数作为输入时，需要用户调用如下接口： ● 调用**acldvppCreateBatchPicDesc**接口创建批量图片描述信息。 ● 调用**acldvppGetPicDesc**接口获取指定图片的图片描述信息。 ● 调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。

参数名	输入/输出	说明
cropAreas	输入	抠图区域位置的指针数组。 需提前调用acldvppCreateRoiConfig接口创建区域位置数据。 cropAreas、borderCfgs数组中的元素个数与dstBatchPicDescs结构体中的batchSize值相等。
borderCfgs	输入	扩边填充配置的指针数组。 borderCfgs数组中的元素与cropAreas数组中的元素一一对应。 扩边填充配置参数中的borderType参数值仅支持BORDER_CONSTANT和BORDER_REPLICATE。
resizeConfig	输入	图片缩放配置数据的指针。 需提前调用acldvppCreateResizeConfig接口创建图片缩放配置数据。 VPC进行缩放处理时，缩放后的宽/高是输出图片的宽/高减去扩边的值。
stream	输入	指定Stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

11.17.5.14 acldvppVpcConvertColorAsync

函数功能

转换图片的格式，输出图片的宽高须与输入图片的宽高一致。异步接口。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 输入、输出图片分辨率为10*6~4096*4096（包括4096）。
- 输入、输出图片的格式请参见图片格式、宽高对齐、内存约束。
- Atlas 推理系列产品上，当输入格式是YUV400时，输出格式只能是YUV400。

函数原型

```
aclError acldvppVpcConvertColorAsync(acldvppChannelDesc *channelDesc,
acldvppPicDesc *inputDesc,
acldvppPicDesc *outputDesc,
aclrtStream stream)
```

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针。 与调用**acldvppCreateChannel**接口创建通道时指定的channelDesc保持一致。
inputDesc	输入	输入图片信息的指针。 ● 调用**acldvppCreatePicDesc**接口创建图片描述信息。 ● 调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。
outputDesc	输入&输出	输出图片信息的指针。 outputDesc参数作为输入时，需要用户调用如下接口： ● 调用**acldvppCreatePicDesc**接口创建图片描述信息。 ● 调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。 需指定输出图片的宽、高、宽stride、高stride、内存地址、内存大小、格式等。 – 配置输出图片的宽、高时，需与输入图片的宽、高保持一致，否则会返回报错。 如果将输出图片的宽或高配置为0，则VPC内部会将输入图片的宽、高分别作为输出图片的宽、高；同时，VPC内部会按对齐要求计算宽stride、高stride，不同图片格式的对齐要求不同，请参见**图片格式、宽高对齐、内存约束**。 – 配置输出图片的宽stride、高stride时，满足对齐要求即可，不同图片格式的对齐要求不同，请参见**图片格式、宽高对齐、内存约束**。
stream	输入	指定Stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

参考资源

接口调用示例，参见**格式转换示例代码**。

11.17.5.15 acldvppVpcPyrDownAsync

函数功能

对图像进行金字塔缩放，将输出图片缩放为输入图片的一半。异步接口。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 关于输入图片
  - 输入图片分辨率：20*12~2048*2048
  - 输入图片格式：PIXEL_FORMAT_YUV_400
- 关于输出图片
  - 输出图像分辨率：输入图像分辨率/2
  - 图像格式：PIXEL_FORMAT_YUV_400
- PIXEL_FORMAT_YUV_400图片格式的对齐要求请参见**图片格式、宽高对齐、内存约束**

函数原型

aclError acldvppVpcPyrDownAsync(acldvppChannelDesc *channelDesc,

acldvppPicDesc *inputDesc,

acldvppPicDesc *outputDesc,

void *reserve,

aclrtStream stream)

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针。 与调用**acldvppCreateChannel**接口创建通道时指定的channelDesc保持一致。
inputDesc	输入	输入图片信息的指针。 ● 调用**acldvppCreatePicDesc**接口创建图片描述信息。 ● 调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。
outputDesc	输入&输出	输出图片信息的指针。 outputDesc参数作为输入时，需要用户调用如下接口： ● 调用**acldvppCreatePicDesc**接口创建图片描述信息。 ● 调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。
reserve	输入	金字塔缩放配置参数的指针，预留参数。
stream	输入	指定Stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

11.17.5.16 acldvppVpcEqualizeHistAsync

函数功能

实现图片的色彩重映射。异步接口。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 输入图片分辨率为10*6~4096*4096（包括4096）。
- 输入图片格式支持acldvppPixelFormat枚举值中的如下枚举项：

```
PIXEL_FORMAT_YUV_400 = 0,            // YUV400 8bit
PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1,   // YUV420SP NV12 8bit
PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2,   // YUV420SP NV21 8bit
PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3,   // YUV422SP 8bit
PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4,   // YVU422SP 8bit
PIXEL_FORMAT_YUV_PACKED_444 = 11,      // YUV444P 8bit
PIXEL_FORMAT_RGB_888 = 12,           // RGB888
PIXEL_FORMAT_BGR_888 = 13,           // BGR888
```

- 输出图片分辨率、输出图片格式、widthStride对齐要求、heightStride对齐要求等，必须与输入图片格的要求保持一致。
- 不同图片格式的对齐要求请参见图片格式、宽高对齐、内存约束。

函数原型

aclError acldvppVpcEqualizeHistAsync(const acldvppChannelDesc *channelDesc,

const acldvppPicDesc *inputDesc,

acldvppPicDesc *outputDesc,

const acldvppLutMap *lutMap,

aclrtStream stream)

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针。 与调用acldvppCreateChannel接口创建通道时指定的channelDesc保持一致。

参数名	输入/输出	说明
inputDesc	输入	输入图片信息的指针。 ● 调用acldvppCreatePicDesc接口创建图片描述信息。 ● 调用acldvppSetPicDesc系列接口设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。
outputDesc	输入&输出	输出图片信息的指针。 outputDesc参数作为输入时，需要用户调用如下接口： ● 调用acldvppCreatePicDesc接口创建图片描述信息。 ● 调用acldvppSetPicDesc系列接口设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。
lutMap	输入	色彩重映射表的指针。 调用acldvppCreateLutMap接口创建acldvppLutMap类型的数据（表示色彩重映射表），再调用acldvppGetLutMapData获取内存指针，由用户自行向该内存中写入色彩重映射表数据，写完数据后，再将acldvppLutMap类型的数据作为入参传入acldvppVpcEqualizeHistAsync。
stream	输入	指定Stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

11.17.5.17 acldvppVpcMakeBorderAsync

函数功能

实现图片边界填充功能。异步接口。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 输入图片分辨率为10*6~4096*4096（包括4096），输入图片格式、对齐、内存等约束，请参见图片格式、宽高对齐、内存约束。
- 输出图片分辨率、图片格式、对齐、内存等约束，请参见11.17.5.2 约束说明。
- 由于YUV格式图像下采样约束，当输出图片格式为YUV420SP或YUV422SP格式时，需注意：
  - 对于YUV420SP输出格式，上下左右填充的尺寸建议为偶数；
  - 对于YUV422SP输出格式，左右填充的尺寸建议为偶数。

函数原型

```
aclError acldvppVpcMakeBorderAsync(const acldvppChannelDesc *channelDesc,
const acldvppPicDesc *inputDesc,
acldvppPicDesc *outputDesc,
const acldvppBorderConfig *borderConfig,
aclrtStream stream)
```

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针。 与调用acldvppCreateChannel接口创建通道时指定的channelDesc保持一致。
inputDesc	输入	输入图片信息的指针。 ● 调用acldvppCreatePicDesc接口创建图片描述信息。 ● 调用acldvppSetPicDesc系列接口设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。
outputDesc	输入&输出	输出图片信息的指针。 outputDesc参数作为输入时，需要用户调用如下接口： ● 调用acldvppCreatePicDesc接口创建图片描述信息。 ● 调用acldvppSetPicDesc系列接口设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。
borderConfig	输入	填充边界配置的指针。 调用acldvppCreateBorderConfig接口创建填充边界配置。
stream	输入	指定Stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

11.17.5.18 acldvppVpcCalcHistAsync

函数功能

统计图像中每个像素的各颜色分量的分布。异步接口。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 关于输入图片分辨率的要求：

版本	约束
Atlas 推理系列产品	输入图片分辨率：10*6~4096*4096（包括4096）。

- 输入图片格式支持**acldvppPixelFormat**枚举值中的如下枚举项：

```
PIXEL_FORMAT_YUV_400 = 0,            // YUV400 8bit
PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1,    // YUV420SP NV12 8bit
PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2,    // YUV420SP NV21 8bit
PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3,    // YUV422SP 8bit
PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4,    // YVU422SP 8bit
PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5,    // YUV444SP 8bit
PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6,    // YVU444SP 8bit
PIXEL_FORMAT_YUYV_PACKED_422 = 7,       // YUV422Packed YUYV 8bit
PIXEL_FORMAT_UYVY_PACKED_422 = 8,       // YUV422Packed UYVY 8bit
PIXEL_FORMAT_YVYU_PACKED_422 = 9,       // YUV422Packed YVYU 8bit
PIXEL_FORMAT_VYUY_PACKED_422 = 10,      // YUV422Packed VYUY 8bit
PIXEL_FORMAT_YUV_PACKED_444 = 11,       // YUV444P 8bit
PIXEL_FORMAT_RGB_888 = 12,            // RGB888
PIXEL_FORMAT_BGR_888 = 13,            // BGR888
PIXEL_FORMAT_YUV_SEMIPLANAR_440 = 1000, // YUV440SP 8bit
PIXEL_FORMAT_YVU_SEMIPLANAR_440 = 1001, // YVU440SP 8bit
```

- 关于图片格式、宽高对齐、内存等总体约束，请参见**图片格式、宽高对齐、内存约束**。

函数原型

aclError acldvppVpcCalcHistAsync(acldvppChannelDesc *channelDesc,

acldvppPicDesc *srcPicDesc,

acldvppHist *hist,

void *reserve,

aclrtStream stream)

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针。 与调用**acldvppCreateChannel**接口创建通道时指定的channelDesc保持一致。

参数名	输入/输出	说明
srcPicDesc	输入	输入图片信息的指针。 ● 调用acldvppCreatePicDesc接口创建图片描述信息。 ● 调用acldvppSetPicDesc系列接口设置图片描述参数（例如，内存地址、内存大小、图片格式、宽、高等）。 **说明** 当图片格式为YUV440SP 8bit、YVU440SP 8bit时，Y分量统计准确，UV分量统计上存在误差。
hist	输入&输出	直方图描述信息的指针。 调用acldvppCreateHist接口创建直方图描述信息作为输入。 调用本接口后，输出每张图像各颜色分量的像素值（0-255）分布情况。
reserve	输入	直方图统计配置的指针，预留，当前可填NULL。
stream	输入	指定Stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

11.17.6 JPEGD 功能

11.17.6.1 功能及约束说明

功能说明

JPEGD（JPEG Decoder）实现.jpg、.jpeg、.JPG、.JPEG图片文件的解码。

- **JPEGD在解码图片时，支持对图片进行旋转。**

  如果输入图片的码流中包含Orientation信息（代表捕获图像时摄像机相对于场景的方向），则JPEGD在解码时会解析Orientation信息，将图片进行90度、180度、270度或镜像旋转。旋转后输出图片的宽stride、高stride、输出内存仍需满足图片格式、宽高对齐、内存约束中的要求。

  如果输入图片的码流异常，导致JPEGD解码时无法读取Orientation信息，则不能实现图片旋转的功能。

  说明

  Atlas 200/300/500 推理产品上，多路并发解码JPEG图片场景下，如果解码的JPEG图片中，包含带旋转信息的大分辨率图片，例如3840*2160分辨率及以上的图片，则可能导致图片解码时间过长，从而导致用户进程超时退出。详细描述请参见12.3.3.3 JPEGD图像解码进程超时退出。

- **JPEGD在解码图片时，支持按源图片格式解码。**

源图片格式解码是指解码前后图片的编码格式保持一致，例如解码前输入图片为jpeg(440)，解码后输出图片为YUV440SP V在前U在后或YUV440SP U在前V在后。

使用源图片格式解码，有以下方式：

- 调用**acldvppJpegGetImageInfoV2**接口，获取JPEGD解码前输入图片的编码格式，在调用JPEGD解码接口时，将输出图片格式设置的与输入图片编码格式一致。
- 在调用JPEGD解码接口时，将输出图片格式配置为PIXEL_FORMAT_UNKNOWN，输出格式默认按源图片格式输出、且是V在前U在后的Semi-Planar格式。例如，JPEG输入图片编码格式为jpeg(440)，输出图片格式配置为PIXEL_FORMAT_UNKNOWN，JPEGD解码后，实际输出图片格式为YUV440SP V在前U在后。

说明

JPEGD解码后的输出图片，如果要直接作为模型推理的输入，建议将输出图片格式配置为PIXEL_FORMAT_UNKNOWN，这时JPEGD使用源图片格式解码（但这里要确保解码后的图片格式模型是支持的），保证模型推理的精度。

JPEGD解码后的输出图片，如果直接作为VPC的输入，该场景下若使用源图片格式解码时，则需要关注解码后的输出图片格式VPC是否支持（VPC输入图片的格式请参见**11.17.5.2 约束说明**），如果VPC不支持，则用户需按VPC支持的情况指定JPEGD的输出图片格式。

图片分辨率约束

- 输入图片分辨率

版本	分辨率范围
Atlas 200/300/500 推理产品 Atlas 推理系列产品	最大分辨率：8192*8192，最小分辨率：32*32。

- 输出图片分辨率

  JPEGD只对图片解码，不会改变图片分辨率，因此输出与输入的图片分辨率保持一致。

图片格式、宽高对齐、内存约束

实现JPEGD图片解码功能时，需调用**acldvppMalloc**接口申请Device上的输入、输出内存，调用**acldvppFree**接口释放输入、输出内存，这部分内存的生命周期由用户自行管理。

- 输入内存的大小就是指实际的输入图片所占用的大小。
- 输出内存的大小，可调用**acldvppJpegPredictDecSize**接口预估，计算公式参见下表。

实现JPEGD图片解码功能时，仅支持Huffman编码，压缩前的原图像色彩空间为YUV，像素的各分量比例为4:4:4或4:2:2或4:2:0或4:0:0或4:4:0，不支持算术编码、不支持渐进JPEG格式、不支持JPEG2000格式。

说明

输出图片格式的定义请参见**acldvppPixelFormat**，宽stride、高stride等概念请参见**11.17.2 基本概念**。

表 11-17 图片格式、宽高对齐、内存大小约束

输入图片格式（YUV分量比例）	输出图片格式	输出图片宽、高对齐要求	输出图片宽stride、高stride、内存大小要求 Atlas 200/300/500推理产品	输出图片宽stride、高stride、内存大小要求 Atlas 推理系列产品
jpeg(444)	YVU444SP 8bit	无对齐要求	宽stride为宽128对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 3	宽stride为宽64对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 3
	YUV444SP 8bit			
	YUV420SP NV12 8bit	宽2对齐 高2对齐	宽stride为宽128对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 3/2	宽stride为宽64对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 3/2
	YUV420SP NV21 8bit			
jpeg(422)	YVU422SP 8bit	宽2对齐 高2对齐	宽stride为宽128对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 2	宽stride为宽64对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 2
	YUV422SP 8bit			
	YUV420SP NV12 8bit	宽2对齐 高2对齐	宽stride为宽128对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 3/2	宽stride为宽64对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 3/2
	YUV420SP NV21 8bit			

输入图片格式（YUV分量比例）	输出图片格式	输出图片宽、高对齐要求	输出图片宽stride、高stride、内存大小要求 Atlas 200/300/500推理产品	输出图片宽stride、高stride、内存大小要求 Atlas 推理系列产品
jpeg(420)	YUV420SP NV12 8bit	宽2对齐 高2对齐	宽stride为宽128对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 3/2	宽stride为宽64对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 3/2
	YUV420SP NV21 8bit			
jpeg(400)	YUV420SP NV12 8bit	宽2对齐 高2对齐	宽stride为宽128对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 3/2	宽stride为宽64对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 3/2
	YUV420SP NV21 8bit			
	YUV400 8bit Atlas 200/300/500推理产品和Atlas 训练系列产品不支持该格式	无对齐要求	宽stride为宽128对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 2	宽stride为宽64对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride
jpeg(440)	YVU440SP 8bit	宽2对齐 高2对齐	宽stride为宽128对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 2	宽stride为宽64对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 2
	YUV440SP 8bit			
	YUV420SP NV12 8bit	宽2对齐 高2对齐	宽stride为宽128对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 3/2	宽stride为宽64对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 3/2
	YUV420SP NV21 8bit			

软、硬件约束

- **硬件约束**：

  - 最多支持4张Huffman表，其中包括2 张DC（直流）表和2 张AC（交流）表。
  - 最多支持3张量化表。
  - 只支持8bit采样精度。
  - 只支持对顺序式编码的图片进行解码。
  - 只支持基于DCT（Discrete Cosine Transform）变换的JPEG 格式解码。
  - 只支持一个SOS（Start of Scan）标志的图片解码。
- **软件约束**：
  - 支持3个SOS标志的图片解码。
  - 支持mcu（Minimum Coded Unit）数据不足的异常图片解码。

精度相关约束

在Atlas 200/300/500 推理产品和Atlas 训练系列产品上，JPEGD+VPC串联使用时，由于JPEGD解码后的输出图片的宽stride*高stride有128*16对齐的约束，因此解码后的输出图片的宽、高有一些补边的无效数据，所以在执行VPC功能时（例如缩放时），需调用acldvppSetPicDescWidth和acldvppSetPicDescHeight接口正确设置输入图片的原图宽高，这样VPC在缩放图片前会先根据原图宽高自行抠图，目的是去除无效数据对图像精度的影响。

在Atlas 推理系列产品上，JPEGD+VPC串联使用时，由于JPEGD解码后的输出图片的宽stride*高stride有64*16对齐的约束，因此解码后的输出图片的宽、高有一些补边的无效数据，所以在执行VPC功能时（例如缩放时），需调用acldvppSetPicDescWidth和acldvppSetPicDescHeight接口正确设置输入图片的原图宽高，这样VPC在缩放图片前会先根据原图宽高自行抠图，目的是去除无效数据对图像精度的影响。

11.17.6.2 性能指标说明

本节仅描述各产品对应的昇腾AI处理器的性能指标参考数据，各产品由于其内存、CPU等配置不同，在性能规格上会存在差异，详细产品的性能规格请查阅昇腾社区的官方说明。

本节以单个昇腾AI处理器为例，描述基本场景性能指标参考，如下表所示，其它分辨率可以等量估算。

下表中的1路对应一个通道，一个通道对应一个线程；各分辨率的含义为：1080p指分辨率为1920*1080的图片，4K指分辨率为3840*2160的图片。

JPEGD性能指标是基于硬件解码的性能，JPEGD硬件解码不支持3个SOS的图片解码，对于硬件不支持的格式，会使用软件解码，软件解码性能参考为1080P 15fps。JPEGD解码的输出图片如果涉及旋转，则性能指标低于软件解码的参考值，例如对于1080P的图片，性能指标低于15fps。

性能指标说明（Atlas 200/300/500 推理产品）

以下性能数据，是基于一个stream上下发1个异步媒体数据处理任务后，执行一次**aclrtSynchronizeStream**接口。

表 11-18 Atlas 200/300/500 推理产品的昇腾 AI 处理器性能指标参考数据

场景举例	总帧率
1080p*1路	128fps
1080p*n路（n≥2）	256fps
4k*1路	32fps
4k*n路（n≥2）	64fps

性能指标说明（Atlas 推理系列产品）

以下性能数据，是基于一个stream上下发10个异步媒体数据处理任务后，执行一次**aclrtSynchronizeStream**接口，且Host服务器配置为：CPU主频为Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz，CPU核心数为14。

表 11-19 Atlas 推理系列产品的昇腾 AI 处理器性能指标参考数据

场景举例	总帧率
1080p*n路（1≤n≤16）	n*128fps
1080p*n路（n>16）	2048fps
4k*n路（1≤n≤16）	n*32fps
4k*n路（n>16）	512fps

11.17.6.3 acldvppJpegDecodeAsync

函数功能

解码.jpg、.jpeg、.JPG、.JPEG图片，异步接口。

约束说明

本接口是异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功。调用该接口后，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保任务已执行完成。

函数原型

```
aclError acldvppJpegDecodeAsync(acldvppChannelDesc *channelDesc,
const void *data,
uint32_t size,
acldvppPicDesc *outputDesc,
aclrtStream stream)
```

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针。 与调用**acldvppCreateChannel**接口创建通道时指定的channelDesc保持一致。
data	输入	输入图片内存地址的指针。
size	输入	输入图片的实际数据大小，单位Byte。
outputDesc	输入&输出	输出图片信息的指针。 outputDesc参数作为输入时，需要用户调用如下接口： ● 调用**acldvppCreatePicDesc**接口创建图片描述信息。 ● 调用**acldvppSetPicDesc系列接口**设置输出图片的内存地址、内存大小、格式、widthStride、heightStride。 – 输出图片的内存大小可提前调用**acldvppJpegPredictDecSize**接口预估。 – JPEG原图的宽、高可通过**acldvppJpegGetImageInfo**/**acldvppJpegGetImageInfoV2**接口获取。 – 输出图片格式、输出图片的宽高对齐要求，请参见**11.17.6.1 功能及约束说明**。 outputDesc参数作为输出时，用户可以从内存地址中获取解码后的输出图片数据、可以调用**acldvppGetPicDesc系列接口**获取输出图片的宽/高。 **说明** Atlas 推理系列产品上，如果解码后的输出图片数据需要在后续操作（例如，使用VPC实现抠图、缩放等功能）中使用，建议在解码后调用**acldvppGetPicDesc系列接口**获取输出图片的宽/高，因为解码过程中可能会对输出图片的宽高执行对齐操作，例如，jpeg(444) 源码的图片解码成YUV420SP格式的输出图片时，当jpeg(444) 源码图片的宽/高为奇数时，解码出来的YUV420SP格式的输出图片的宽/高理论上应该为奇数，但是YUV420SP格式本身要求图片的宽/高都为偶数，这时JPEGD会对奇数宽/高做向下2对齐的操作。
stream	输入	指定Stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

参考资源

接口调用流程及示例，参见**5.4.2 JPEGD图像解码**。

11.17.6.4 acldvppJpegGetImageInfo

函数功能

从存放JPEG图片数据的内存中读取JPEG图片的宽、高、颜色通道个数。

该接口会检查图片格式是否支持解码，如果遇到不支持的码流则该接口会返回ACL_ERROR_UNSUPPORTED_JPEG错误码，该错误码的详细描述请参见**aclError**。

函数原型

aclError acldvppJpegGetImageInfo(const void *data,

uint32_t size,

uint32_t *width,

uint32_t *height,

int32_t *components)

参数说明

参数名	输入/输出	说明
data	输入	存放JPEG图片数据的内存地址指针。
size	输入	内存大小，单位为Byte。
width	输出	图片宽的指针。
height	输出	图片高的指针。
components	输出	颜色通道个数的指针。

返回值说明

返回0表示获取信息成功，返回非0表示获取信息失败。

11.17.6.5 acldvppJpegGetImageInfoV2

函数功能

从存放JPEG图片数据的内存中读取JPEG图片的宽、高、颜色通道个数、JPEG原图编码格式。

该接口会检查图片格式是否支持解码，如果遇到不支持的码流则该接口会返回ACL_ERROR_UNSUPPORTED_JPEG错误码，该错误码的详细描述请参见**aclError**。

函数原型

aclError acldvppJpegGetImageInfoV2(const void *data,

uint32_t size,

uint32_t *width,

uint32_t *height,

int32_t *components,

acldvppJpegFormat *format)

参数说明

参数名	输入/输出	说明
data	输入	存放JPEG图片数据的内存地址指针。
size	输入	内存大小，单位为Byte。
width	输出	图片宽的指针。
height	输出	图片高的指针。
components	输出	颜色通道个数的指针。
format	输出	JPEG原图编码格式的指针。

返回值说明

返回0表示获取信息成功，返回非0表示获取信息失败。

11.17.6.6 acldvppJpegPredictDecSize

函数功能

根据存放JPEG图片数据的内存预估JPEG图片解码后所需的输出内存的大小。

函数原型

aclError acldvppJpegPredictDecSize(const void *data,

uint32_t dataSize,

acldvppPixelFormat outputPixelFormat,

uint32_t *decSize)

参数说明

参数名	输入/输出	说明
data	输入	存放JPEG图片数据的内存地址指针。
size	输入	内存大小，单位为Byte。

参数名	输入/输出	说明
outputPixelFormat	输入	解码后的输出图片的格式。
decSize	输出	预估JPEG图片解码后所需输出内存大小的指针，单位为Byte。

返回值说明

返回0表示获取信息成功，返回非0表示获取信息失败。

参考资源

接口调用流程及示例，参见**5.4.2 JPEGD图像解码**。

11.17.7 JPEGE 功能

11.17.7.1 功能及约束说明

功能说明

JPEGE（JPEG Encoder）将YUV格式图片编码成JPEG压缩格式的图片文件，例如*.jpg。

图片分辨率约束

- 输入图片分辨率：

版本	分辨率范围
Atlas 200/300/500 推理产品 Atlas 推理系列产品	最大分辨率：8192*8192，最小分辨率：32*32。

- 输出图片分辨率

  JPEGE只对图片编码，不会改变图片分辨率，因此输出与输入的图片分辨率保持一致。

图片格式、宽高对齐、内存约束

实现JPEGE图片编码功能时，需调用**acldvppMalloc**接口申请Device上的输入、输出内存，调用**acldvppFree**接口释放输入、输出内存，这部分内存的生命周期由用户自行管理。

- 输入内存的大小请参见下表中的计算公式。
- 输出内存的大小就是指实际的编码后图片所占用的大小，可调用**acldvppJpegPredictEncSize**接口预估输出内存大小。

说明

输入图片格式的定义请参见**acldvppPixelFormat**，宽stride、高stride等概念请参见**11.17.2 基本概念**。

表 11-20 图片格式、宽高对齐、内存大小约束

输入图片格式	输入图片宽、高对齐要求	输入图片宽stride、高stride、内存大小要求	输出图片格式
YUV420SP NV12 8bit	宽2对齐 高2对齐	宽stride为宽16对齐后的值，兼容对齐到16的倍数如128（按128对齐时，性能更优）。 高stride支持以下两种配置： ● 配置为与输入图片的高度相同的数值； ● 或配置为输入图片的高度向上对齐到16的数值（最小为32）。该方式的使用场景举例：JPEGD的输出图片直接作为JPEGE的输入（JPEGD输出图片高度是向上对齐到16的）。 内存大小（单位Byte）= 宽stride * 高stride * 3/2	JPEG压缩格式的图片文件，例如*.jpg。 只支持huffman编码，不支持算术编码，不支持渐进编码。
YUV420SP NV21 8bit			
YUV422Packed YUYV 8bit	宽2对齐 高无对齐要求	宽stride为宽的两倍再16对齐后的值。 高stride支持以下两种配置： ● 配置为与输入图片的高度相同的数值； ● 或配置为输入图片的高度向上对齐到16的数值（最小为32）。该方式的使用场景举例：JPEGD的输出图片直接作为JPEGE的输入（JPEGD输出图片高度是向上对齐到16的）。 内存大小（单位Byte）= 宽stride * 高stride	
YUV422Packed UYVY 8bit			
YUV422Packed YVYU 8bit			
YUV422Packed VYUY 8bit			

11.17.7.2 性能指标说明

本节仅描述各产品对应的昇腾AI处理器的性能指标参考数据，各产品由于其内存、CPU等配置不同，在性能规格上会存在差异，详细产品的性能规格请查阅昇腾社区的官方说明。

本节以单个昇腾AI处理器为例，描述基本场景性能指标参考，如下表所示，其它分辨率可以等量估算。

下表中的1路对应一个通道，一个通道对应一个线程；各分辨率的含义为：1080p指分辨率为1920*1080的图片，4K指分辨率为3840*2160的图片。

性能指标说明（Atlas 200/300/500 推理产品）

以下性能数据，是基于一个stream上下发10个异步媒体数据处理任务后，执行一次**aclrtSynchronizeStream**接口。

表 11-21 Atlas 200/300/500 推理产品的昇腾 AI 处理器性能指标参考数据

场景举例	总帧率
1080p*n路（n≥1）	64fps
4k*n路（n≥1）	16fps

性能指标说明（Atlas 推理系列产品）

以下性能数据，是基于一个stream上下发10个异步媒体数据处理任务后，执行一次**aclrtSynchronizeStream**接口，且Host服务器配置为：CPU主频为Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz，CPU核心数为14。

表 11-22 Atlas 推理系列产品的昇腾 AI 处理器性能指标参考数据

场景举例（输入图片格式：YUV420 8bit）	总帧率
1080p*n路（1≤n≤8）	n*128fps
1080p*n路（n>8）	1024fps
4k*n路（1≤n≤8）	n*32fps
4k*n路（n>8）	256fps

11.17.7.3 acldvppJpegEncodeAsync

函数功能

将YUV格式图片编码成.jpg图片，异步接口。

约束说明

本接口是异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功。调用该接口后，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保任务已执行完成。

函数原型

aclError acldvppJpegEncodeAsync(acldvppChannelDesc *channelDesc,

acldvppPicDesc *inputDesc,

const void *data,

uint32_t *size,

acldvppJpegeConfig *config,

aclrtStream stream);

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针。 与调用**acldvppCreateChannel**接口创建通道时指定的channelDesc保持一致。
inputDesc	输入	输入图片描述信息的指针。 ● 调用**acldvppCreatePicDesc**接口创建图片描述信息。 ● 调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，图片格式、宽、高等）。 输入图片分辨率（≤通道的最大宽度*高度）、图像格式的要求，请参见**11.17.7.1 功能及约束说明**。
data	输出	输出内存地址的指针，存放编码后的数据。
size	输入&输出	输出内存大小的指针，单位Byte。 size作为输入参数时，可提前调用**acldvppJpegPredictEncSize**接口预估输出内存大小。 size作为输出参数时，表示实际输出内存大小，可能与调用**acldvppJpegPredictEncSize**接口预估的内存大小存在差异，如果用户需要取用编码后的数据，请使用实际输出内存大小。
config	输入	图片编码配置数据的指针。 调用**acldvppCreateJpegeConfig**接口创建图片编码配置数据，调用**acldvppSetJpegeConfigLevel**接口设置编码配置数据。
stream	输入	指定Stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

参考资源

接口调用流程及示例，参见5.4.3 JPEGE图片编码。

11.17.7.4 acldvppJpegPredictEncSize

函数功能

根据输入图片描述信息、图片编码配置数据预估图片编码后所需的输出内存的大小。

函数原型

aclError acldvppJpegPredictEncSize(const acldvppPicDesc *inputDesc, const acldvppJpegeConfig *config, uint32_t *size)

参数说明

参数名	输入/输出	说明
inputDesc	输入	输入图片描述信息的指针。 调用acldvppCreatePicDesc接口创建图片描述信息，调用acldvppSetPicDesc系列接口设置图片描述参数（例如，图片格式、宽、高等）。
config	输入	图片编码配置数据的指针。 调用acldvppCreateJpegeConfig接口创建图片编码配置数据。
size	输出	预估JPEG图片编码后所需输出内存大小的指针，单位为Byte。 预估的输出内存会大于实际输出内存，实际输出内存需从acldvppJpegEncodeAsync接口的输出参数size获取。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见5.4.3 JPEGE图片编码。

11.17.8 PNGD 功能

11.17.8.1 功能及约束说明

功能说明

PNGD（PNG Decoder）功能：实现PNG格式图片的解码。

PNGD在解码图片时，支持按源图片格式解码。在调用PNGD解码接口时，直接将输出图片格式配置为PIXEL_FORMAT_UNKNOWN，输出格式默认按源图片格式输出。

源图片格式解码是指解码前后图片的编码格式保持一致，例如解码前输入图片格式为RGB，解码后输出图片格式为RGB888。

图片分辨率约束

- 输入图片分辨率：

  最大分辨率4096*4096，最小分辨率32*32。
- 输出图片分辨率

  PNGD只对图片解码，不会改变图片分辨率，因此输出与输入的图片分辨率保持一致。

图片格式、宽高对齐、内存约束

实现图片解码功能时，需调用**acldvppMalloc**接口申请Device上的输入、输出内存，调用**acldvppFree**接口释放输入、输出内存，这部分内存的生命周期由用户自行管理。

- 输入内存的大小就是指实际的输入图片所占用的大小。
- 输出内存的大小可调用**acldvppPngPredictDecSize**接口预估。

说明

输出图片格式的定义请参见**acldvppPixelFormat**，宽stride、高stride等概念请参见**11.17.2 基本概念**。

表 11-23 图片格式、宽高对齐、内存大小约束

输入图片格式	输出图片格式	输出图片宽、高对齐要求	输出图片宽stride、高stride、内存大小要求
RGB	RGB888	无对齐要求	宽stride为宽128对齐后再乘以3的值。 高stride为高16对齐后的值。 输出内存的大小可调用**acldvppPngPredictDecSize**接口预估。
GRAY	RGB888	无对齐要求	宽stride为宽128对齐后再乘以3的值。 高stride为高16对齐后的值。 输出内存的大小可调用**acldvppPngPredictDecSize**接口预估。

输入图片格式	输出图片格式	输出图片宽、高对齐要求	输出图片宽stride、高stride、内存大小要求
RGBA	RGB888	无对齐要求	宽stride为宽128对齐后再乘以3的值。 高stride为高16对齐后的值。 输出内存的大小可调用**acldvppPngPredictDecSize**接口预估。
	RGBA8888 8bit	无对齐要求	宽stride为宽128对齐后再乘以4的值。 高stride为高16对齐后的值。 输出内存的大小可调用**acldvppPngPredictDecSize**接口预估。
AGRAY	RGB888	无对齐要求	宽stride为宽128对齐后再乘以3的值。 高stride为高16对齐后的值。 输出内存的大小可调用**acldvppPngPredictDecSize**接口预估。
	RGBA8888 8bit	无对齐要求	宽stride为宽128对齐后再乘以4的值。 高stride为高16对齐后的值。 输出内存的大小可调用**acldvppPngPredictDecSize**接口预估。

11.17.8.2 性能指标说明

本节仅描述各产品对应的昇腾AI处理器的性能指标参考数据，各产品由于其内存、CPU等配置不同，在性能规格上会存在差异，详细产品的性能规格请查阅昇腾社区的官方说明。

本节以单个昇腾AI处理器为例，描述基本场景性能指标参考，如下表所示，其它分辨率可以等量估算。

下表中的1路对应一个通道，一个通道对应一个线程；各分辨率的含义为：1080p指分辨率为1920*1080的图片，4K指分辨率为3840*2160的图片。

性能指标说明（Atlas 200/300/500 推理产品）

以下性能数据，是基于一个stream上下发10个异步媒体数据处理任务后，执行一次**aclrtSynchronizeStream**接口。

表 11-24 Atlas 200/300/500 推理产品的昇腾 AI 处理器性能指标参考数据

场景举例	总帧率
1080p*n路（1≤n≤5）	n*4fps
1080p*n路（n≥ 6）	24fps
4k*n路（1≤n≤5）	n*1fps
4k*n路（n≥ 6）	6fps

性能指标说明（Atlas 推理系列产品）

以下性能数据，是基于一个stream上下发10个异步媒体数据处理任务后，执行一次**aclrtSynchronizeStream**接口，且Host服务器配置为：CPU主频为Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz，CPU核心数为14。

表 11-25 Atlas 推理系列产品的昇腾 AI 处理器性能指标参考数据

场景举例	总帧率
1080p*n路（1≤n≤5）	n*4fps
1080p*n路（n≥ 6）	24fps
4k*n路（1≤n≤5）	n*1fps
4k*n路（n≥ 6）	6fps

11.17.8.3 acldvppPngDecodeAsync

函数功能

解码png图片，异步接口。

功能	输入图片格式	输出图片格式
调用本接口时将输出图片格式设置为PIXEL_FORMAT_UNKNOWN 表示不指定输出图片格式，输出图片格式与输入图片格式一致	RGB/GRAY	RGB
调用本接口时将输出图片格式设置为PIXEL_FORMAT_UNKNOWN 表示不指定输出图片格式，输出图片格式与输入图片格式一致	RGBA/AGRAY	RGBA

功能	输入图片格式	输出图片格式
调用本接口时将输出图片格式设置为PIXEL_FORMAT_RGB_888，表示指定输出图片格式，将输入图片格式转换为输出图片格式	RGB/RGBA/GRAY/AGRAY	RGB

约束说明

本接口是异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功。调用该接口后，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保任务已执行完成。

函数原型

aclError acldvppPngDecodeAsync(acldvppChannelDesc *channelDesc,

const void *data,

uint32_t size,

acldvppPicDesc *outputDesc,

aclrtStream stream)

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针。 与调用**acldvppCreateChannel**接口创建通道时指定的channelDesc保持一致。
data	输入	输入图片内存地址的指针。
size	输入	输入图片的实际数据大小，单位Byte。
outputDesc	输入&输出	输出图片信息的指针。 outputDesc参数作为输入时，需要用户调用如下接口： ● 调用**acldvppCreatePicDesc**接口创建图片描述信息。 ● 调用**acldvppSetPicDesc系列接口**设置输出图片的内存地址、内存大小、格式。 – 输出图片的内存大小可提前调用**acldvppPngPredictDecSize**接口获取。 – png原图信息（例如宽、高）可通过**acldvppPngGetImageInfo**接口获取。 – 输出图片格式、内存等要求，请参见**11.17.8.1 功能及约束说明**。 outputDesc参数作为输出时，用户可以从内存地址中获取解码后的输出图片数据。

参数名	输入/输出	说明
stream	输入	指定Stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

参考资源

接口调用流程及示例，参见5.4.4 PNGD图片解码。

11.17.8.4 acldvppPngGetImageInfo

函数功能

从存放png图片数据的内存中读取png图片的宽、高。

函数原型

```
aclError acldvppPngGetImageInfo(const void *data,
uint32_t dataSize,
uint32_t *width,
uint32_t *height,
int32_t *components)
```

参数说明

参数名	输入/输出	说明
data	输入	存放png图片数据内存地址的指针。
dataSize	输入	内存大小，单位为Byte。
width	输出	图片宽的指针。
height	输出	图片高的指针。
components	输出	颜色通道个数的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.17.8.5 acldvppPngPredictDecSize

函数功能

根据存放png图片数据的内存计算出png图片解码后所需的输出内存的大小。

函数原型

aclError acldvppPngPredictDecSize(const void *data,

uint32_t dataSize,

acldvppPixelFormat outputPixelFormat,

uint32_t *decSize)

参数说明

参数名	输入/输出	说明
data	输入	存放png图片数据内存地址的指针。
size	输入	内存大小，单位为Byte。
outputPixelFormat	输入	解码后的输出图片的格式，支持设置如下格式： ● PIXEL_FORMAT_UNKNOWN：按占用内存大小最大的PNG解码输出格式计算内存大小。 ● PIXEL_FORMAT_RGB_888 ● PIXEL_FORMAT_RGBA_8888
decSize	输出	png图片解码后所需输出内存大小的指针，单位为Byte。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**5.4.4 PNGD图片解码**。

11.17.9 VDEC 功能

11.17.9.1 功能及约束说明

功能说明

VDEC（Video Decoder）实现视频的解码，VDEC内部经过VPC处理后，输出指定格式的图片，参见**码流/图片格式、宽高对齐、内存约束**。

分辨率约束

- 输入码流分辨率：

版本	分辨率范围
Atlas 200/300/500 推理产品 Atlas 推理系列产品	最大分辨率4096*4096，最小分辨率128*128。

- 输出图片分辨率：

版本	分辨率范围
Atlas 200/300/500 推理产品 Atlas 推理系列产品	最大分辨率4096*4096，最小分辨率10*6。

码流/图片格式、宽高对齐、内存约束

实现VDEC视频解码功能时，输入或输出内存的生命周期由用户自行管理：

- 输入内存

  调用**acldvppMalloc**/**acldvppFree**接口申请/释放内存、或调用**aclrtMalloc**/**aclrtFree**接口申请/释放内存，内存大小就是指实际的输入码流所占用的大小。

  调用**acldvppMalloc**接口申请的内存为媒体数据处理的专用内存，但专用内存的地址空间有限，若关注内存规划或内存资源有限时，使用VDEC视频解码功能时，建议调用**aclrtMalloc**接口申请内存。

- 输出内存

  需调用**acldvppMalloc**接口申请Device上的输出内存，调用**acldvppFree**接口释放输出内存，内存大小请参见下表中的计算公式。

说明

输出图片格式的定义请参见**acldvppPixelFormat**，宽stride、高stride等概念请参见**11.17.2 基本概念**。

VDEC视频解码时，输入码流支持如下格式：

- H264 bp/mp/hp level5.1 YUV420编码的码流，当前只支持annex-B的裸码流。
- H265 8/10bit level5.1 YUV420编码的码流，当前只支持annex-B的裸码流。

VDEC视频解码时，输出图片格式、宽高对齐要求等如下表所示：

表 11-26 码流/图片格式、宽高对齐、内存大小约束

版本	输出图片格式	输出图片宽、高对齐要求	输出图片宽stride、高stride、内存大小要求 Atlas 200/300/500 推理产品 Atlas 推理系列产品
通用格式，各版本都支持	YUV420SP NV12 8bit	宽2对齐 高2对齐	宽stride为宽16对齐后的值。如果用户设置的输出图片宽度小于16，则宽stride最小为32。 高stride为高2对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 3/2
	YUV420SP NV21 8bit		
该部分格式仅以下版本支持： Atlas 推理系列产品	RGB888	无对齐要求	宽stride为宽16对齐后再乘以3的值。 高stride无对齐要求。 内存大小（单位Byte）≥ 宽stride * 高stride
	BGR888		

其它约束

- VDEC只支持按帧输入码流进行解码。
- 若码流中有坏帧、缺帧等情况，解码器VDEC可能会丢帧。
- 通过隔行扫描方式编码出来的码流，VDEC仅支持解码H264 8bit编码的码流。

11.17.9.2 性能指标说明

本节仅描述各产品对应的昇腾AI处理器的性能指标参考数据，各产品由于其内存、CPU等配置不同，在性能规格上会存在差异，详细产品的性能规格请查阅昇腾社区的官方说明。

本节以单个昇腾AI处理器为例，描述基本场景性能指标参考，如下表所示，其它分辨率可以等量估算。

下表中的1路对应一个通道，一个通道对应一个线程；各分辨率的含义为：720p指分辨率为1280*720的图片，1080p指分辨率为1920*1080的图片，4K指分辨率为3840*2160的图片。

性能指标说明（Atlas 200/300/500 推理产品）

- 针对如下典型场景下各分辨率的码流，单个Device的VDEC解码的路数推荐如下值：

表 11-27 Atlas 200/300/500 推理产品的昇腾 AI 处理器性能指标参考数据

分辨率	总性能	单路性能（n路）	每路VDEC解码的最大内存消耗（此处以H265格式的输入码流、参考帧数量9个为例，作为参考值）
4k	120fps	120fps/n（推荐n=4，每路30fps）	约182MB
1080p	480fps	480fps/n（推荐n=16，每路30fps）	约56MB
≤720p	960fps	960fps/n（推荐n=32，每路30fps）	约35MB

下表以1080P分辨率的输入码流为例，说明每路VDEC解码的最大内存消耗的计算公式，在计算公式中：

- 输入码流缓存大小：1080P分辨率及以下的输入码流，输入码流缓存大小默认为6M；1080P分辨率以上的输入码流，输入码流缓存大小默认为9M。
- 解码图像帧存大小：1080P分辨率的输入码流，该参数值为3MB。其它分辨率时参数值可等量折算。
- 视频解码图像Tmv缓存大小：H264格式、1080P分辨率的输入码流，该参数值为0.5MB；H265格式、1080P分辨率的输入码流，该参数值为1MB。其它分辨率时参数值可等量折算。
- 参考帧数量：以最大参考帧个数为例，系统内部会根据实际码流的参考帧个数自适应。

每路VDEC解码的内存消耗计算公式	场景举例	内存消耗（单位为MB）
4MB + *输入码流缓存大小* * 2 + (*解码图像帧存大小* + *视频解码图像Tmv缓存大小*) * (*参考帧数量* + 1)	● 输入码流格式H264 ● 输入码流分辨率1080P ● 输入码流缓存大小为6M ● 解码图像帧存大小为3MB ● 视频解码图像Tmv缓存大小为0.5MB	51MB（参考帧数量9个） 26.5MB（参考帧数量2个）

每路VDEC解码的内存消耗计算公式	场景举例	内存消耗（单位为MB）
	● 输入码流格式H265 ● 输入码流分辨率1080P ● 输入码流缓存大小为6M ● 解码图像帧存大小为3MB ● 视频解码图像Tmv缓存大小为1MB	56MB（参考帧数量9个） 28MB（参考帧数量2个）

- 关于解码路数与帧率的建议

  注意：下表中给出的规格建议供参考，如果单进程内启动的路数超过下表中的建议，则可能出现内存不足或性能不够的问题，进而导致创建解码通道失败或执行解码缓慢。

典型分辨率	单进程内各启动路数时的规格建议（依据输入帧率得出）				
-	输入帧率≥25fps	20fps<输入帧率<25fps	15fps<输入帧率≤20fps	10fps<输入帧率≤15fps	输入帧率≤10fps
≤720p	32路	32路	32路	32路	32路
1080p	16路	19路	24路	32路	32路
4K	4路	4路	6路	8路	12路

性能指标说明（Atlas 推理系列产品）

以下性能数据，基于以下Host服务器配置：CPU主频为Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz，CPU核心数为14。

表 11-28 Atlas 推理系列产品的昇腾 AI 处理器性能指标参考数据

场景举例	总帧率
720p*n路(1≤n≤12)	n*480fps **说明** 针对 Atlas 300V 视频解析卡，当1<n≤12时，总帧率都为n*400fps。
720p*n路(n>12)	5760fps **说明** 针对 Atlas 300V 视频解析卡，当n>12时，总帧率都为4800fps。
1080p*n路(1≤n≤12)	n*240fps **说明** 针对 Atlas 300V 视频解析卡，当1<n≤12时，总帧率都为n*200fps。

场景举例	总帧率
1080p*n路(n>12)	2880fps **说明** 针对 Atlas 300V 视频解析卡，当n>12时，总帧率都为2400fps。
4k*n路(1≤n≤12)	n*60fps
4k*n路(n>12)	720fps

下表说明了典型场景下每路VDEC解码的内存消耗，在内存消耗的计算公式中：

- 输入码流缓存大小：等于输入码流的宽*高*2。
- 解码图像帧存大小：1080P分辨率的输入码流，该参数值为3MB。其它分辨率时参数值可等量折算。
- 视频解码图像Tmv缓存大小：H264格式、1080P分辨率的输入码流，该参数值为0.5MB；H265格式、1080P分辨率的输入码流，该参数值为1MB。其它分辨率时参数值可等量折算。
- 参考帧数量可由用户调用**aclvdecSetChannelDescRefFrameNum**接口设置。
- 解码后缓存图像帧数固定为2个。

每路VDEC解码的内存消耗计算公式	场景举例	内存消耗（单位为MB）
6MB + *输入码流缓存大小* * 2 + (*解码图像帧存大小* + *视频解码图像Tmv缓存大小*) * (*参考帧数量* + *解码后缓存图像帧数* +1)	• 输入码流格式H264 • 输入码流分辨率1080P • 输入码流缓存大小为4MB • 解码图像帧存大小为3MB • 视频解码图像Tmv缓存大小为0.5MB • 解码后缓存图像帧数2个	52.5MB（参考帧数量8个） 31.5MB（参考帧数量2个）
	• 输入码流格式H265 • 输入码流分辨率1080P • 输入码流缓存大小为4MB • 解码图像帧存大小为3MB • 视频解码图像Tmv缓存大小为1MB • 解码后缓存图像帧数2个	58MB（参考帧数量8个） 34MB（参考帧数量2个）

11.17.9.3 aclvdecSendFrame

函数功能

将待解码的输入内存和解码输出内存一起传到解码器进行解码，异步接口。

约束说明

- 本接口是异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功。调用该接口后，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保任务已执行完成。
- 发送数据前必须保证通道已经被创建，否则返回错误。
- 发送码流时须按帧发送，一次只发送完整的一帧码流。
- 不能发送eos为0的空码流包（码流长度为0或码流地址为空）。
- 发送完所有码流后，可以发送eos为1的帧，表示当前码流文件结束。发送eos后，本接口会等待已发送帧全部解码并且用户的回调函数处理完成后才返回。
- 由于码流异常、解码超时等原因，可能导致aclvdecSendFrame接口发送帧或发送eos失败，建议用户在编写代码时，获取该接口的返回码，当该接口调用失败时，进行异常处理。
- aclvdecSendFrame接口内部封装了**aclrtLaunchCallback**接口，用于在Stream的任务队列中增加一个需要执行的回调函数。用户在实现VDEC功能时，无需再单独调用**aclrtLaunchCallback**接口。

函数原型

```
aclError aclvdecSendFrame(aclvdecChannelDesc *channelDesc,
acldvppStreamDesc *input,
acldvppPicDesc *output,
aclvdecFrameConfig *config,
void* userData)
```

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针。 与调用**aclvdecCreateChannel**接口创建通道时指定的channelDesc保持一致。 调用**aclvdecSetChannelDesc系列接口**设置通道描述信息的属性，包括解码通道号、线程、回调函数、视频编码协议等。 **说明** 在Atlas 200/300/500 推理产品上，由于软件约束，不支持调用**aclvdecSetChannelDescRefFrameNum**接口设置通道描述信息的对应属性，若调用，则会返回报错。 在Atlas 推理系列产品上，支持调用**aclvdecSetChannelDescRefFrameNum**接口设置通道描述信息的对应属性。
input	输入	输入码流描述信息的指针，输入内存用户需提前申请。 ● 调用**acldvppCreateStreamDesc**接口创建码流描述信息，调用**acldvppSetStreamDesc系列接口**设置视频码流信息的属性（例如，码流数据的内存地址）。输入码流格式以通道描述信息中设置的格式为准，调用**acldvppSetStreamDescFormat**接口设置输入码流格式无效。 ● 输入视频码流的分辨率、视频格式的要求，请参见**11.17.9.1 功能及约束说明**。
output	输入&输出	输出图片描述信息的指针，输出内存用户需提前申请。 output参数作为输入时，需要用户调用如下接口： ● 调用**acldvppCreatePicDesc**接口创建图片描述信息。 ● 调用**acldvppSetPicDesc系列接口**设置输出图片的数据内存、内存大小、宽、高、格式。 – 如果此处设置输出图片的宽高，则需要同时设置width、height、widthStride、heightStride这4个参数，且当width<16时，widthStride必须对齐到32，不能对齐到16；如果这4个参数都不设置，则以输入码流的宽高为准；如果设置部分参数，则接口返回错误。 – 如果此处设置输出图片的格式，则以此处为准；如果此处不设置，则以通道描述信息中设置的为准。 – 输出图像格式的要求，请参见**11.17.9.1 功能及约束说明**。 output参数作为输出时，用户需在回调函数中调用**acldvppGetPicDesc系列接口**获取解码后的输出图片数据。隔行码流场景下，隔行码流每帧发送两场，解码时其中一场无图像输出，属于正常现象，会返回ERR_DECODE_NOPIC = 0x20000错误码；隔行码流的解码输出数据都在奇数场对应的输出buffer中。

参数名	输入/输出	说明
config	输入	解码配置的指针，预留，当前可填NULL。
userData	输入	用户自定义数据的指针。 如果用户需要获取解码的帧序号，则可以在userData参数处定义，然后解码的帧序号可以通过userData参数传递给VDEC的回调函数，用于确定回调函数中处理的是第几帧数据。

返回值说明

返回0表示成功，返回非0表示失败。

参考资源

- 接口调用流程及示例，参见**5.4.5 VDEC视频解码**。
- Device内存不足，VDEC解码任务异常的案例请参见**12.3.3.10 VDEC视频解码异常导致进程卡死，无法退出**。

11.17.9.4 aclvdecSendSkippedFrame

函数功能

如果不想获取某一帧的解码结果， 可以调用本接口， 将待解码的码流（输入内存)传到解码器进行解码，此时，解码结果最终不会输出，解码完成的回调函数中返回的output为nullptr。异步接口。

约束说明

- 本接口是异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功。调用该接口后，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保任务已执行完成。
- 发送数据前必须保证通道已经被创建，否则返回错误。
- 发送码流时须按帧发送，一次只发送完整的一帧码流。
- 不能发送eos为0的空码流包（码流长度为0或码流地址为空）。
- 不可以发送eos为1的帧。
- 由于码流异常、解码超时等原因，可能导致aclvdecSendFrame接口发送帧或发送eos失败，建议用户在编写代码时，获取该接口的返回码，当该接口调用失败时，进行异常处理。
- aclvdecSendSkippedFrame接口内部封装了**aclrtLaunchCallback**接口，用于在Stream的任务队列中增加一个需要执行的回调函数。用户在实现VDEC功能时，无需再单独调用**aclrtLaunchCallback**接口。

函数原型

aclError aclvdecSendSkippedFrame(aclvdecChannelDesc *channelDesc,

acldvppStreamDesc *input,

aclvdecFrameConfig *config,

void *userData)

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针。 与调用**aclvdecCreateChannel**接口创建通道时指定的channelDesc保持一致。 调用**aclvdecSetChannelDesc系列接口**设置通道描述信息的属性，包括解码通道号、线程、回调函数、视频编码协议等。 **说明** 在Atlas 200/300/500 推理产品上，由于软件约束，不支持调用**aclvdecSetChannelDescRefFrameNum**接口设置通道描述信息的对应属性，若调用，则会返回报错。 在Atlas 推理系列产品上，支持调用**aclvdecSetChannelDescRefFrameNum**接口设置通道描述信息的对应属性。
input	输入	输入码流描述信息的指针，输入内存用户需提前申请。 ● 调用**acldvppCreateStreamDesc**接口创建码流描述信息，调用**acldvppSetStreamDesc系列接口**设置视频码流信息的属性（例如，码流格式）。 输入码流格式以通道描述信息中设置的格式为准，调用**acldvppSetStreamDescFormat**接口设置输入码流格式无效。 ● 输入视频码流的分辨率、视频格式的要求，请参见**11.17.9.1 功能及约束说明**。
config	输入	解码配置的指针，预留，当前可填NULL。
userData	输入	用户自定义数据的指针。 如果用户需要获取解码的帧序号，则可以在userData参数处定义，然后解码的帧序号可以通过userData参数传递给VDEC的回调函数，用于确定回调函数中处理的是第几帧数据。

返回值说明

返回0表示成功，返回非0表示失败。

11.17.9.5 aclvdecCallback

函数功能

视频解码回调函数，同步接口。用户需自定义该回调函数。

约束说明

- 在回调函数中不能执行销毁通道操作，否则会导致程序执行死锁。
- 回调函数处理的时延应满足发帧帧率要求，否则会影响**aclvdecSendFrame**接口进行视频帧处理的实时性。
- 注销回调线程要在所有Callback执行完成后（即注销送码流线程之后）。
- 当使用了抽帧功能后（调用**aclvdecSendSkippedFrame**接口），回调相应帧的output为nullptr。

  对于抽帧的情况，输出Buffer无解码结果，且**抽掉的帧**按照顺序输出。

  对于不抽帧的情况，输出Buffer中才会有解码结果，且**不抽的帧**按照顺序输出。

  如果解码码流中同时存在抽帧、不抽帧的情况，则系统不保证抽掉的帧与不抽的帧之间按照顺序输出。

函数原型

void (* aclvdecCallback) (acldvppStreamDesc * input, acldvppPicDesc * output, void* userData)

参数说明

参数名	输入/输出	说明
input	输入	输入码流描述信息的指针。与**aclvdecSendFrame**接口中的input一致。
output	输入	VDEC解码后的输出图片描述信息的指针，输出内存用户需提前申请。
userData	输入	用户自定义数据的指针。

返回值说明

无

参考资源

接口调用流程及示例，参见**5.4.5 VDEC视频解码**。

11.17.10 VENC 功能

11.17.10.1 功能及约束说明

功能说明

VENC（Video Encoder）将YUV420SP NV12/NV21-8bit图片数据编码成H264/H265格式的视频码流。

在Atlas 200/300/500 推理产品上，不支持单进程多线程场景。

分辨率约束

- 输入图片分辨率：

版本	分辨率范围
Atlas 200/300/500 推理产品	最大分辨率1920*1920，最小分辨率128*128。
Atlas 推理系列产品	宽和高的范围都是128~4096，对于H264格式的码流，最大分辨率的乘积不超过4096*2304。

- 输出码流分辨率

  VENC只对图片编码，不会改变图片分辨率，因此输出与输入的图片分辨率保持一致。

码流/图片格式、宽高对齐、内存约束

实现VENC视频编码功能时：

- 输入内存

  需调用**acldvppMalloc**接口申请Device上的输入内存，调用**acldvppFree**接口释放输入内存，这部分内存的生命周期由用户自行管理。内存大小参见下表中的计算公式。

- 输出内存

  不需要用户管理输出内存，由系统管理内存。

说明

输入图片格式的定义请参见**acldvppPixelFormat**，宽stride、高stride等概念请参见**11.17.2 基本概念**。

表 11-29 图片格式、宽高对齐、内存大小约束

输入图片格式	输入图片宽、高对齐要求	输入图片宽stride、高stride、内存大小要求	输出码流格式
YUV420SP NV12 8bit	宽2对齐 高2对齐	宽stride为宽16对齐后的值。 高stride无需设置。 内存大小（单位Byte）= 宽stride * 高 * 3/2	• H264 BP/MP/HP • H265 MP（仅支持Slice码流）
YUV420SP NV21 8bit			

11.17.10.2 性能指标说明

本节仅描述各产品对应的昇腾AI处理器的性能指标参考数据，各产品由于其内存、CPU等配置不同，在性能规格上会存在差异，详细产品的性能规格请查阅昇腾社区的官方说明。

本节以单个昇腾AI处理器为例，描述基本场景性能指标参考，如下表所示，其它分辨率可以等量估算。

下表中的1路对应一个通道，一个通道对应一个线程；各分辨率的含义为：720p指分辨率为1280*720的图片，1080p指分辨率为1920*1080的图片，4K指分辨率为3840*2160的图片。

性能指标说明（Atlas 200/300/500 推理产品）

表 11-30 Atlas 200/300/500 推理产品的昇腾 AI 处理器性能指标参考数据

场景举例	总帧率
1080p*n路（一个进程对应一路）	30fps

性能指标说明（Atlas 推理系列产品）

以下性能数据，基于以下Host服务器配置：CPU主频为Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz，CPU核心数为14。

表 11-31 Atlas 推理系列产品的昇腾 AI 处理器性能指标参考数据

场景举例	总帧率
720p*n路(1≤n≤3)	n*480fps
720p*n路(n>3)	1440fps
1080p*n路(1≤n≤3)	n*240fps
1080p*n路(n>3)	720fps
4k*n路(1≤n≤3)	n*60fps
4k*n路(n>3)	180fps

11.17.10.3 aclvencSendFrame

函数功能

将待编码的图片传到编码器进行编码。异步接口。

约束说明

- 本接口是异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功。调用该接口后，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保任务已执行完成。
- 发送数据前必须保证通道已经被创建，否则返回错误。
- 不能发送eos为0的空码流包（码流长度为0或码流地址为空）。
- 结束视频编码时可以发送eos为1的空图片，表示当前编码结束。
- 编码时，输入内存需要在**aclvencCallback**回调函数内获取结果之后才能进行释放。

函数原型

aclError aclvencSendFrame(aclvencChannelDesc *channelDesc,

acldvppPicDesc *input,

void *reserve,

aclvencFrameConfig *config,

void *userdata)

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针。 与调用**aclvencCreateChannel**接口创建通道时指定的channelDesc保持一致。 在通道描述信息中指定视频编码的回调函数。
input	输入	输入图片描述信息的指针，输入内存用户需提前申请。 ● 调用**acldvppCreatePicDesc**接口创建图片描述信息，调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，宽、高等）。 输入图片格式以通道描述信息中设置的格式为准，调用**acldvppSetPicDescFormat**接口设置输入图片格式无效。 ● 输入图片的分辨率、格式的要求，请参见**11.17.10.1 功能及约束说明**。
reserve	输入	预留参数，暂不使用。
config	输入	单帧配置数据的指针。
userdata	输入	用户自定义数据的指针。

返回值说明

返回0表示成功，返回非0表示失败。

参考资源

接口调用流程及示例，参见5.4.6 VENC视频编码。

11.17.10.4 aclvencCallback

函数功能

视频编码回调函数，同步接口。用户需自定义该回调函数。

约束说明

- 在回调函数中不能执行销毁通道操作，否则会导致程序执行死锁。
- 回调函数处理的时延应满足发帧帧率要求，否则会影响aclvencSendFrame接口进行视频帧处理的实时性。
- 注销回调线程要在所有Callback执行完成后。

函数原型

void (*aclvencCallback)(acldvppPicDesc *input, acldvppStreamDesc *output, void *userdata)

参数说明

参数名	输入/输出	说明
input	输入	输入图片描述信息的指针。
output	输入	VENC编码后的输出帧码流描述信息的指针。
userdata	输入	用户自定义数据的指针。

返回值说明

无

参考资源

接口调用流程及示例，参见5.4.6 VENC视频编码。

11.18 媒体数据处理 V2

11.18.1 总体功能及约束说明

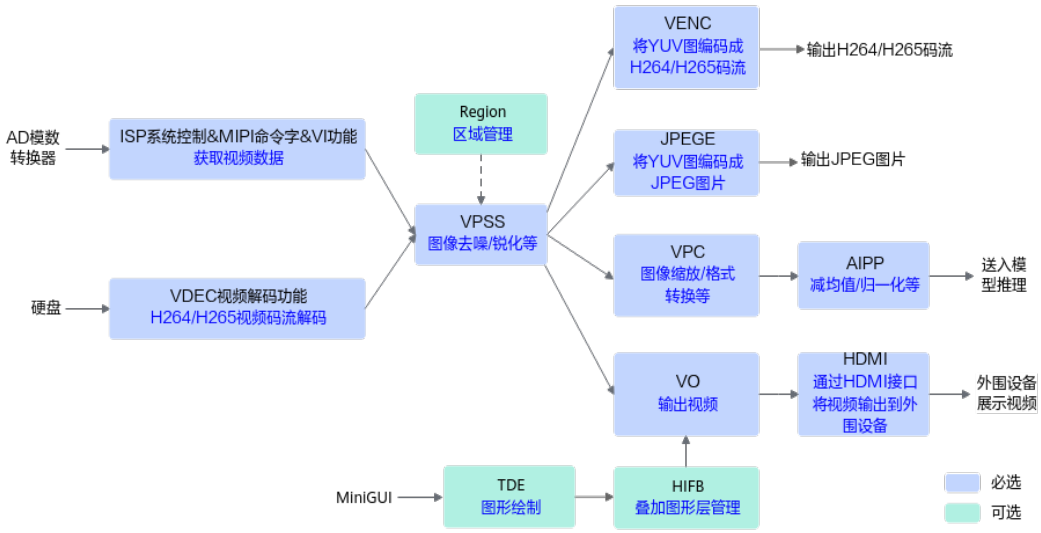
多版本接口差异

本手册中媒体数据处理V1版本与媒体数据处理V2版本的接口都是描述处理媒体数据的接口，用于实现视频编解码、图像编解码、图像处理等功能，但两套接口不能混用。

- V2版本的功能比V1版本更多，如下：
  - JPEGE：V2版本接口支持高级的参数配置，如huffman表配置。
  - VENC：V2版本接口支持更加细化的码控参数配置和效果调优，如I/P帧QP、宏块码控等。
  - VDEC：V2版本接口支持更细化的内存控制，如设置输入码流缓存。
- 建议使用V2版本中的接口，保证后续版本接口功能以及业务的连续演进。
- V1版本中的接口是为了兼容旧版本，保证使用该部分接口的用户能继续使用，后续版本不再演进。

图像/视频/音频数据处理的典型功能介绍

图 11-7 图像/视频数据处理



各功能的介绍如下表所示，昇腾AI处理器对这些功能的支持度请参见**5.3 各版本功能支持度说明**（AIPP当前各版本均支持）。

功能	子功能模块	描述
获取视频数据	**ISP**（Image Signal Processing）系统控制	系统控制部分用于注册3A算法、注册Sensor驱动、初始化ISP firmware、运行ISP firmware、退出ISP firmware、配置ISP属性等功能。
	MIPI Rx ioctl命令字	MIPI Rx是一个支持多种差分视频输入接口的采集单元，通过combo-PHY接收MIPI/LVDS/sub-LVDS/HiSPi接口的数据，通过不同的功能模式配置，MIPI Rx可以支持多种速度和分辨率的数据传输需求，支持多种外部输入设备。
	VI（Vedio Input）	VI模块捕获视频图像，可对其做裁剪、防抖、颜色优化、亮度优化、噪声去除等处理，并输出YUV或RAW格式的图像数据。

功能	子功能模块	描述
展示视频数据	**VO**（Vedio Output）	VO模块接收VPSS处理后的输出图像，可进行播放控制等处理，最后按用户配置的输出协议（当前仅支持HDMI）输出给外围视频设备。 VO可配合TDE（Two Dimensional Engine）模块、HIFB（Hisilicon Framebuffer）模块，利用硬件分别进行图形绘制、叠加图形层管理。
	HDMI（High Definition Multimedia Interfac）	HDMI是全数字化影像和声音发送接口，可以发送未压缩的音频及视频信号。
	TDE（Two Dimensional Engine）	TDE是图形二维加速引擎，它利用硬件为 OSD（On Screen Display）和 GUI（Graphics User Interface）提供快速的图形绘制功能，主要有快速拷贝、快速色彩填充、模式填充（当前仅支持Alpha Blending操作）。
	HIFB（Hisilicon Framebuffer）	HIFB用于管理叠加图形层，它不仅提供Linux Framebuffer的基本功能，还在Linux Framebuffer的基础上增加图层显示起始位置修改、层间Alpha等扩展功能。
区域管理	Region	叠加在视频上的OSD (On Screen Display)和遮挡在视频上的色块统称为区域。**区域管理模块**，用于统一管理这些区域资源，用于在视频上显示一些特定信息（如通道号、时间戳等）、或在视频中填充色块用于遮挡，当前该功能需配合VPSS一起使用。
图像/视频数据处理	**VPSS**（Video Process Sub-System）	VPSS模块支持对输入图像进行统一预处理，如去噪、去隔行、裁剪等，然后再对各通道分别进行处理，如缩放、加边框等。

功能	子功能模块	描述
	AIPP（Artificial Intelligence Pre-Processing）	AIPP人工智能预处理，在AI Core上完成数据预处理，主要功能包括改变图像尺寸（抠图、填充等）、色域转换（转换图像格式）、减均值/乘系数（改变图像像素）等。 AIPP区分为静态AIPP和动态AIPP。您只能选择静态AIPP或动态AIPP中的一种来处理图片，不能同时配置静态AIPP和动态AIPP两种方式。 ● 静态AIPP：模型转换时设置AIPP模式为静态，同时设置AIPP参数，模型生成后，AIPP参数值被保存在离线模型（*.om）中，每次模型推理过程采用固定的AIPP预处理参数（无法修改）。 如果使用静态AIPP方式，多Batch情况下共用同一份AIPP参数，AIPP参数值在使用ATC工具进行模型转换时设置，ATC工具的详细说明请《**ATC工具使用指南（开放态）**》。 ● 动态AIPP：模型转换时仅设置AIPP模式为动态，每次模型推理前，根据需求，在执行模型前设置动态AIPP参数值，然后在模型执行时可使用不同的AIPP参数。如果使用动态AIPP方式，多Batch可使用不同的AIPP参数，各Batch所使用的AIPP参数值通过AscendCL提供的接口来设置，请参见**7.8 模型动态AIPP推理**中的介绍。
	DVPP（Digital Vision Pre-Processing）	DVPP是昇腾AI处理器内置的图像处理单元，通过AscendCL媒体数据处理接口提供强大的媒体处理硬加速能力，主要功能包括以下功能： ● VPC（Vision Preprocessing Core）：处理YUV、RGB等格式的图片，包括缩放、抠图、图像金字塔、色域转换等。 ● JPEGD（JPEG Decoder）：JPEG压缩格式-->YUV格式的图片解码。 ● JPEGE（JPEG Encoder）：YUV格式-->JPEG压缩格式的图片编码。 ● VDEC（Video Decoder）：H264/H265格式-->YUV/RGB格式的视频码流解码。 ● VENC（Video Encoder）：YUV420SP格式-->H264/H265格式的视频码流编码。 ● PNGD（PNG Decoder）：PNG格式-->RGB格式的图片解码。 **说明** AIPP、DVPP可以分开独立使用，也可以组合使用。组合使用场景下，一般先使用DVPP对图片/视频进行解码、抠图、缩放等基本处理，但由于DVPP硬件上的约束，DVPP处理后的图片格式、分辨率有可能不满足模型的要求，因此还需要再经过AIPP进一步做色域转换、抠图、填充等处理。 例如，在Atlas 200/300/500 推理产品和Atlas 训练系列产品上，由于DVPP视频解码仅支持输出YUV格式的图片，如果模型需要RGB格式的图片，则需要再经过AIPP做色域转换的处理。

功能	子功能模块	描述
音频数据获取和输出	**AI**（Audio Input）	AI模块捕获音频数据 。
	AO（Audio Output）	通过ADEC模块解码后的音频数据，AO模块支持播放音频。
音频数据编解码	**AENC**（Audio Encoder）	通过AI模块获取的音频数据，AENC模块支持对其进行编码，输出音频码流。
	ADEC（Audio Decoder）	ADEC支持解码G.711a、G.711u等协议的音频码流，再通过AO模块播放音频。

功能支持度说明

昇腾AI处理器对媒体数据处理V2版本各功能的支持度如下表所示。

昇腾AI处理器	VPC	JPEGD	JPEGE	PNGD	VDEC	VENC
Atlas 推理系列产品	√	√	√	√	√	√

整体约束说明

使用本章中介绍的接口，有以下注意点：

- 关于内存申请/释放：
  a. 实现媒体数据处理的VPC、JPEGD、JPEGE等功能前，若需要申请内存存放输入或输出数据，需调用**hi_mpi_dvpp_malloc**申请内存、调用**hi_mpi_dvpp_free**接口释放内存。如果多个功能串联使用的场景，需要复用同一段内存，则按最大内存要求来申请内存。
  b. 调用**a**申请出来的内存可以满足媒体数据处理的要求，也可以在其它任务中使用，例如，从性能角度，为了减少拷贝，媒体数据处理的输出作为模型推理的输入，实现内存复用。
  c. 但由于媒体数据处理访问的地址空间有限，为确保媒体数据处理时内存足够，除媒体数据处理功能外的其它功能（例如，模型加载），建议调用**内存管理**章节下的**aclrtMalloc**接口、或**aclrtMallocHost**接口、或**aclrtMallocCached**接口申请内存。

     针对Atlas 推理系列产品：媒体数据处理功能在每个进程内，可访问的地址空间最大为16GB。
- 关于通道的要求：

  实现媒体数据处理的各功能前，必须调用接口创建对应功能的通道，请分别参见**11.18.16 VPC图像处理功能**、**11.18.17 VDEC视频解码功能/JPEGD图片解码功能**、**11.18.18 VENC视频编码功能/JPEGE图片编码功能**、**11.18.19 PNGD图片解码功能**章节下的通道创建与销毁接口，查看接口说明以及通道数的最大限制。

  通道的创建与销毁会涉及资源的申请与释放，反复创建与销毁通道会影响业务性能，因此建议根据实际场景管理通道，例如，如果有持续VPC图片处理，则创建VPC的通道后，等到所有VPC功能调用完成后，再销毁该VPC通道。

通道数量多，会影响Device的CPU占用率和内存占用，通道数量建议参考各功能章节下的性能指标的路数。

- 本章节描述结构体、枚举值等，预留的字段需要手动设置为0，避免后续版本的兼容性问题。

  结构体或枚举值中的预留字段，其名称中包含*_BUTT字符串的，例如HI_COMPRESS_MODE_BUTT。

11.18.2 基本概念

表 11-32 概念解释

概念	描述
宽stride（widthStride）	指一行图像跨距，表示输入图片对齐后的宽，RGB格式和YUV格式的宽stride计算方式不一样。 各功能对宽stride的要求不同，请参见对应功能的约束说明： ● VPC功能，请参见**图片格式、宽高对齐、内存约束**。 ● JPEGD功能，请参见**11.18.17.1 JPEGD功能及约束说明**。 ● JPEGE功能，请参见**11.18.18.1 JPEGE功能及约束说明**。 ● PNGD功能，请参见**11.18.19.1 功能及约束说明**。 ● VDEC功能，请参见**11.18.17.3 VDEC功能及约束说明**。 ● VENC功能，请参见**11.18.18.3 VENC功能及约束说明**。
高stride（height Stride）	指图像在内存中的行数，表示输入图片对齐后的高。 各功能对高stride的要求不同，请参见对应功能的约束说明： ● VPC功能，请参见**图片格式、宽高对齐、内存约束**。 ● JPEGD功能，请参见**11.18.17.1 JPEGD功能及约束说明**。 ● JPEGE功能，请参见**11.18.18.1 JPEGE功能及约束说明**。 ● PNGD功能，请参见**11.18.19.1 功能及约束说明**。 ● VDEC功能，请参见**11.18.17.3 VDEC功能及约束说明**。 ● VENC功能，请参见**11.18.18.3 VENC功能及约束说明**。
抠图区域	指用户指定的需抠出的图片区域。抠图起始坐标无奇数、偶数限制。 抠图区域最小分辨率为10*6，最大分辨率为8192*8192。 抠图区域的约束请参见**抠图、贴图约束**。
贴图区域	指在输出图片中用户指定的区域。 贴图区域最小分辨率为10*6，最大分辨率为4096*4096。 贴图区域的约束请参见**抠图、贴图约束**。
上/左偏移	通过配置上偏移、左偏移、抠图/贴图区域的宽和高可以指定抠图区域或贴图区域的位置。参见**图11-14**。

11.18.3 公共接口

11.18.3.1 hi_mpi_sys_init

函数功能

用户处理图像、视频等媒体数据之前，需先调用本接口对媒体数据处理系统底层进行初始化。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 需在申请AscendCL运行管理资源后调用本接口。AscendCL运行管理资源申请流程请参见**4.4 运行管理资源申请与释放**。
- 多个进程同时运行情况下，每个进程都必须调用本接口初始化。
- 一个进程内，可以调用本接口反复初始化，不返回失败。
- 多Device场景下，针对每一个Device，都需要调用hi_mpi_sys_init接口进行初始化。

函数原型

hi_s32 hi_mpi_sys_init(hi_void)

参数说明

无

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.1 公共返回码**

参考资源

接口调用流程及示例，参见**5.5 DVPP图像/视频处理（媒体数据处理V2）**。

11.18.3.2 hi_mpi_sys_exit

函数功能

用户处理完图像、视频等媒体数据之后，需先调用本接口对媒体数据处理系统底层进行去初始化。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 可以调用本接口反复去初始化，不返回失败。
- 调用本接口去初始化之后，会释放系统内部媒体数据处理的资源，因此不能再调用媒体数据处理V2章节下的其它接口（除**hi_mpi_sys_init**接口外），否则可能出现接口调用异常、返回码错误等情况。
- 需在释放AscendCL运行管理资源前调用本接口。AscendCL运行管理资源申请流程请参见**4.4 运行管理资源申请与释放**。

函数原型

hi_s32 hi_mpi_sys_exit(hi_void)

参数说明

无

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.1 公共返回码**

参考资源

接口调用流程及示例，参见**5.5 DVPP图像/视频处理（媒体数据处理V2）**。

11.18.3.3 hi_mpi_dvpp_malloc

函数功能

申请Device上的内存，申请的内存满足媒体数据处理的要求（例如，内存首地址128字节对齐）。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 调用该接口申请内存后，这部分内存的生命周期由用户自行管理，如果不使用，必须调用**hi_mpi_dvpp_free**接口及时释放内存。
- 通过该接口申请的内存仅支持在单个进程内使用，不能在多个进程之间共享。
- **Control CPU开放形态**下，若配置cgroup，应用程序在Device上运行时，当前默认在内存不足时，应用程序可能会挂起，等待内存资源，用户可以根据实际需求选择启用操作系统提供的一些配置（例如，enable_oom_killer），这样在内存不足时，应用程序会自动退出，不会一直等待。

  若启用enable_oom_killer，您需登录Device，在“/proc/sys/vm”目录下，以root用户启用enable_oom_killer，命令示例如下，1表示启用，0表示禁用：
  ```
  echo 1 > enable_oom_killer
  ```

  说明

  cgroup的配置请参见《**CANN 软件安装指南 (开放态)**》。
- 调用hi_mpi_dvpp_malloc接口申请内存时，会对用户输入的size按向上对齐成32整数倍后，再多加32字节。
- 媒体数据处理的输出作为模型推理的输入时，若用户使用本接口申请大块内存并自行划分、管理内存时，每段内存需同时满足以下要求：
  - 内存大小向上对齐成32整数倍+32字节（m=ALIGN_UP[len,32]+32字节）；
  - 内存起始地址需满足128字节对齐（ALIGN_UP[m,128]）。

  说明

  len表示某段内存的大小，ALIGN_UP[len,k]表示向上按k字节对齐：((len-1)/k+1)*k。

函数原型

hi_s32 hi_mpi_dvpp_malloc(hi_u32 dev_id, hi_void **dev_ptr, hi_u64 size)

参数说明

参数名	输入/输出	说明
dev_id	输入	Device ID，预留参数，该参数配置无效。
dev_ptr	输出	“Device上已分配内存的指针”的指针。
size	输入	申请内存的大小，单位Byte。

返回值说明

- 0：成功
- 非0：失败

参考资源

接口调用流程及示例，参见5.5 DVPP图像/视频处理（媒体数据处理V2）。

11.18.3.4 hi_mpi_dvpp_free

函数功能

释放Device上的内存。

hi_mpi_dvpp_free接口只能释放通过hi_mpi_dvpp_malloc接口申请的内存。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

hi_s32 hi_mpi_dvpp_free(hi_void *dev_ptr)

参数说明

参数名	输入/输出	说明
dev_ptr	输入	待释放内存的指针。

返回值说明

- 0：成功
- 非0：失败

参考资源

接口调用流程及示例，参见5.5 DVPP图像/视频处理（媒体数据处理V2）。

11.18.3.5 hi_mpi_dvpp_get_image_info

函数功能

根据输入码流，获取按源图格式输出时的图片宽、高、宽stride、高stride、解码后占用的内存大小、源图片格式等信息。该接口会检查图片格式是否支持解码，如果遇到不支持的码流则该接口会返回HI_ERR_VDEC_NOT_SUPPORT错误码，该错误码的详细描述请参见11.18.21.11 VDEC视频解码/JPEGD图片解码返回码。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

hi_s32 hi_mpi_dvpp_get_image_info(hi_payload_type img_type, const hi_vdec_stream *stream, hi_img_info *img_info);

参数说明

参数名	输入/输出	说明
img_type	输入	要解析的码流类型。 当前仅支持码流类型为HI_PT_JPEG。
stream	输入	输入码流信息的指针。 Atlas 推理系列产品，Control CPU开放形态下，在Device上调用本接口时，该结构体内的addr参数配置的地址必须是Device上的内存地址。
img_info	输出	图片信息的指针，包含图片宽、高、宽stride、高stride、解码后占用的内存大小、源图片格式等信息。 其中，源图格式通过pixel_format参数返回。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.11 VDEC视频解码/JPEGD图片解码返回码。

11.18.3.6 hi_mpi_dvpp_get_version

函数功能

查询接口版本号，版本号命名采用：A.B.C模式，其中，A表示有不兼容修改，B表示新增接口，C表示bug修复。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

hi_s32 hi_mpi_dvpp_get_version(hi_s32 *major_version, hi_s32 *minor_version, hi_s32 *patch_version)

参数说明

参数名	输入/输出	说明
major_version	输出	主版本号指针，从1开始，如果出现接口的不兼容变更时，加1。
minor_version	输出	次版本号指针，从0开始，按照迭代周期，有新增接口时加1。
patch_version	输出	补丁版本号指针，从0开始，表示本版本仅仅解决了问题，在major_version、minor_version不变的情况下加1；但major_version、minor_version增加的时候，patch_version一般为0。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.1 公共返回码**

11.18.3.7 hi_mpi_sys_create_epoll

函数功能

创建新的媒体数据处理的Epoll实例，使用Epoll编程模型操作媒体数据处理通道的输入/输出数据。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- **hi_mpi_sys_create_epoll**、**hi_mpi_sys_ctl_epoll**、**hi_mpi_sys_wait_epoll**、**hi_mpi_sys_close_epoll**这几个接口要配合使用，实现使用Epoll编程模型操作媒体数据处理通道的输入/输出数据。
- Atlas 推理系列产品，**Control CPU开放形态**下，使用AscendCL提供的Epoll接口（例如，**hi_mpi_sys_create_epoll**）与使用Linux操作系统的Epoll函数等价。
- VPC功能不支持使用该接口。

函数原型

hi_s32 hi_mpi_sys_create_epoll(hi_s32 size, hi_s32 *epoll_fd)

参数说明

参数名	输入/输出	说明
size	输入	Epoll实例最大可处理的DVPP通道文件句柄个数，当前该字段为预留，传入一个正数即可。
epoll_fd	输出	Epoll实例句柄的指针。句柄个数受限于Linux操作系统的open files（一个任务最多可以同时打开的文件个数）资源限制，您可以使用ulimit -al命令查看open files。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.1 公共返回码**

参考资源

接口调用流程及示例，参见**5.5.3 JPEGE图片编码**、**5.5.6 VENC视频编码**。

11.18.3.8 hi_mpi_sys_ctl_epoll

函数功能

在媒体数据处理Epoll实例中，对对应的媒体数据处理通道进行控制操作（增、删、改）。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- **hi_mpi_sys_create_epoll**、**hi_mpi_sys_ctl_epoll**、**hi_mpi_sys_wait_epoll**、**hi_mpi_sys_close_epoll**这几个接口要配合使用，实现使用epoll编程模型操作媒体数据处理通道的输入/输出数据。
- VPC功能不支持使用该接口。

函数原型

hi_s32 hi_mpi_sys_ctl_epoll(hi_s32 epoll_fd, hi_s32 operation, hi_s32 fd, hi_dvpp_epoll_event *event)

参数说明

参数名	输入/输出	说明
epoll_fd	输入	Epoll实例句柄，句柄需提前通过**hi_mpi_sys_create_epoll**接口创建。

参数名	输入/输出	说明
operation	输入	操作类型（增、删、改），参见hi_dvpp_epoll_ctl_op。
fd	输入	媒体数据处理通道的文件句柄。VENC场景下，您可以调用hi_mpi_venc_get_fd接口提前获取该文件句柄。VDEC场景下，您可以调用hi_mpi_vdec_get_fd接口提前获取该文件句柄。
event	输入	Epoll事件信息的指针。 该参数用于描述对应媒体数据处理通道事件类型以及关联的用户数据。 进行删除操作时，该字段不用，可直接传入NULL。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.1 公共返回码

参考资源

接口调用流程及示例，参见5.5.3 JPEGE图片编码、5.5.6 VENC视频编码。

11.18.3.9 hi_mpi_sys_wait_epoll

函数功能

使用Epoll模型进行媒体数据处理通道输入输出操作，等待媒体数据处理Epoll实例产生事件。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- hi_mpi_sys_create_epoll、hi_mpi_sys_ctl_epoll、hi_mpi_sys_wait_epoll、hi_mpi_sys_close_epoll这几个接口要配合使用，实现使用Epoll编程模型操作媒体数据处理通道的输入/输出数据。
- VPC功能不支持使用该接口。

函数原型

hi_s32 hi_mpi_sys_wait_epoll(hi_s32 epoll_fd, hi_dvpp_epoll_event *events, hi_s32 max_events, hi_s32 timeout, hi_s32 *event_num)

参数说明

参数名	输入/输出	说明
epoll_fd	输入	媒体数据处理Epoll实例句柄的指针。
events	输出	hi_dvpp_epoll_event数组，保存产生的事件信息。
max_events	输入	最大可返回的事件数，不应超过events数组的元素个数。
timeout	输入	超时时间，单位是毫秒。 ● -1：永久阻塞 ● 0：立即返回 ● >0：配置具体的超时时间
event_num	输出	Epoll实际产生的事件数的指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.1 公共返回码**

参考资源

接口调用流程及示例，参见**5.5.3 JPEGE图片编码**、**5.5.6 VENC视频编码**。

11.18.3.10 hi_mpi_sys_close_epoll

函数功能

使用Epoll模型进行媒体数据处理通道输入输出操作，销毁媒体数据处理Epoll实例。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- **hi_mpi_sys_create_epoll**、**hi_mpi_sys_ctl_epoll**、**hi_mpi_sys_wait_epoll**、**hi_mpi_sys_close_epoll**这几个接口要配合使用，实现使用epoll编程模型操作媒体数据处理通道的输入/输出数据。
- VPC功能不支持使用该接口。

函数原型

hi_s32 hi_mpi_sys_close_epoll(hi_s32 epoll_fd)

参数说明

参数名	输入/输出	说明
epoll_fd	输入	待销毁媒体数据处理Epoll实例句柄，句柄通过**hi_mpi_sys_create_epoll**接口创建。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.1 公共返回码**

参考资源

接口调用流程及示例，参见**5.5.3 JPEGE图片编码**、**5.5.6 VENC视频编码**。

11.18.3.11 hi_mpi_sys_set_chn_csc_matrix

函数功能

在处理图像或视频前，YUV与RGB之间格式转换场景下，如果需要调整色域转换参数时，在创建对应图像或视频处理通道后，可调用该接口设置色域转换参数。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

如果不调用本接口设置色域转换参数，默认使用**BT601 wide标准的色域转换矩阵参数值**。

如果默认的色域转换参数不适用，仅需修改一次，该通道内的其它任务都使用修改后的色域转换参数，则在创建对应图像或视频处理通道后，可调用本接口设置色域转换参数，再调用具体的图像或视频处理接口，然后再调用对应的图像或视频处理结果获取接口。

如果默认的色域转换参数不适用，不同任务使用不同的色域转换参数，需多次修改，则在创建对应图像或视频处理通道后，在上一次任务执行结束后、下一次任务执行前调用本接口设置色域转换参数（如果上一次任务没有执行结束，就再次调用本接口设置色域转换参数，可能会导致任务处理的结果不正确），再调用具体的图像或视频处理接口，然后再调用对应的图像或视频处理结果获取接口。

调用顺序示例如下：

1. **hi_mpi_sys_set_chn_csc_matrix**（第一次修改色域转换参数）
2. **hi_mpi_vpc_crop**（抠图）
3. **hi_mpi_vpc_get_process_result**（等待抠图任务结束）
4. **hi_mpi_sys_set_chn_csc_matrix**（第二次修改色域转换参数）
5. **hi_mpi_vpc_resize**（缩放）
6. **hi_mpi_vpc_get_process_result**（等待缩放任务结束）

函数原型

hi_s32 hi_mpi_sys_set_chn_csc_matrix(hi_mod_id mode, hi_s32 chn, hi_csc_matrix csc_matrix, hi_csc_coefficient *csc_coefficient)

参数说明

参数名	输入/输出	说明
mode	输入	模块ID，当前只支持HI_ID_VDEC和HI_ID_VPC。
chn	输入	通道号。当mode设置为HI_ID_VDEC时，通道号范围请参见**11.18.17.5 hi_mpi_vdec_create_chn**处chn参数的取值范围；当mode设置为HI_ID_VPC时，通道号范围请参见**11.18.16.4 hi_mpi_vpc_create_chn**处chn参数的取值范围。
csc_matrix	输入	色域转换矩阵的数据标准。
csc_coefficient	输入	色域转换矩阵参数值的指针。 如果将csc_matrix参数设置为HI_CSC_MATRIX_USER，则需要使用csc_coefficient参数设置色域转换矩阵参数；否则，此处传任意值。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.1 公共返回码**

11.18.3.12 hi_mpi_sys_get_chn_csc_matrix

函数功能

用户使用该接口获取通道的色域转换矩阵参数值。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 在创建通道成功后，才可以调用本接口获取通道的色域转换矩阵参数值。
- 不调用**hi_mpi_sys_set_chn_csc_matrix**接口设置色域转换矩阵参数时，返回默认的**BT601 wide标准的色域转换矩阵参数值**。

函数原型

hi_s32 hi_mpi_sys_get_chn_csc_matrix(hi_mod_id mode, hi_s32 chn, hi_csc_matrix *csc_matrix, hi_csc_coefficient *csc_coefficient)

参数说明

参数名	输入/输出	说明
mode	输入	模块ID，当前只支持HI_ID_VDEC和HI_ID_VPC。
chn	输入	通道号。 当mode设置为HI_ID_VDEC时，通道号范围请参见11.18.17.5 hi_mpi_vdec_create_chn处chn参数的取值范围； 当mode设置为HI_ID_VPC时，通道号范围请参见11.18.16.4 hi_mpi_vpc_create_chn处chn参数的取值范围。
csc_matrix	输出	色域转换矩阵的数据标准的指针。
csc_coefficient	输出	色域转换矩阵参数值的指针。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.1 公共返回码

11.18.3.13 hi_mpi_sys_get_image_align_info

函数功能

根据模块、图片宽高计算出宽高对齐后的值以及存放图片数据的内存大小。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 图片宽、高不能小于该模块最小宽高、不能大于最大宽高，图片格式必须是该模块所支持的。
- 各模块支持的图片格式、对齐要求请参见11.18.16 VPC图像处理功能、11.18.17 VDEC视频解码功能/JPEGD图片解码功能、11.18.18 VENC视频编码功能/JPEGE图片编码功能、11.18.19 PNGD图片解码功能下的约束说明。

  图片格式不支持HI_PIXEL_FORMAT_UNKNOWN。

函数原型

```
hi_s32 hi_mpi_sys_get_image_align_info(const hi_module_type mod_type[],
const hi_u32 mod_num, const hi_img_base_info *img_base_info,
hi_img_align_info *img_align_info)
```

参数说明

参数名	输入/输出	说明
mod_type	输入	模块类型数组。 如果数组中只有一个模块，则系统内部按该模块的对齐值来计算；如果数组中有多个模块，则系统内部按多个模块中的最大对齐值来计算；如果将模块类型设置为HI_MOD_ALL，则系统内部会从所有模块中选取最大对齐值来计算，便于简化理解各模块的对齐差异，操作简便，但可能占用内存较大。
mod_num	输入	模块类型数组长度。
img_base_info	输入	图片基本信息结构体。
img_align_info	输出	图片对齐信息、内存大小的结构体。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.1 公共返回码**

11.18.3.14 hi_mpi_sys_bind

函数功能

调用本接口建立数据接收者与数据源之间的关联关系，同一个数据接收者只能绑定一个数据源，绑定后，数据源生成的数据将自动发送给接收者。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 当前支持如下数据源、数据接收者之间的绑定关系：

数据源	数据接收者	说明
视频输入VI	视频处理VPSS	VPSS Group ID取值在[256, 268)范围内。
视频解码VDEC	视频处理VPSS	VPSS Group ID取值在[0, 256)范围内。
视频处理VPSS	视频输出VO	VPSS Group ID取值在[0, 256)范围内。
音频解码ADEC	音频输出AO	-

数据源	数据接收者	说明
音频输入AI	音频编码AENC	-

- 除以下场景可将设备号或通道号设置为0，其它场景均需根据实际使用的设备号、通道号来设置。
  - VI和VDEC作为数据源，是以通道为发送者，向其他模块发送数据，用户将设备号src_chn.dev_id参数设置为0，SDK不检查输入的设备号。
  - VPSS作为数据接收者时，是以设备（GROUP）为接收者，接收其他模块发来的数据，用户将通道号dst_chn.chn_id参数设置为0。

函数原型

hi_s32 hi_mpi_sys_bind(const hi_mpp_chn *src_chn, const hi_mpp_chn *dest_chn)

参数说明

参数名	输入/输出	说明
src_chn	输入	源通道指针。
dest_chn	输入	目的通道指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.1 公共返回码**

11.18.3.15 hi_mpi_sys_unbind

函数功能

调用本接口解除数据源与数据接收者之间的关联关系。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- dst_chn如果找不到绑定的源通道，则直接返回成功。如果找到了绑定的源通道，但是绑定的源通道和src_chn不匹配，则返回失败。

函数原型

hi_s32 hi_mpi_sys_unbind(const hi_mpp_chn *src_chn, const hi_mpp_chn *dest_chn)

参数说明

参数名	输入/输出	说明
src_chn	输入	源通道指针。
dest_chn	输入	目的通道指针。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.1 公共返回码

11.18.4 AI 音频输入/AO 音频输出

11.18.4.1 功能及约束说明

- AI（Audio Input）音频输入、AO（Audio Output）音频输出API主要实现配置及启用音频输入设备、获取音频帧数据、播放等功能，**本章中的接口均不支持多进程，且同一个设备ID不支持在多线程中使用**。
- 对于内置Audio Codec，主要通过ioctl命令字实现对硬件设备的操作。
- 在录音起始阶段，由于硬件存在资源准备的过程，因此可能在起始阶段瞬间存在音频数据波动现象，但并不影响后续录音功能。
- 关于AI、AO设备的说明如下：

表 11-33 AO 设备

AO dev	输出对接器件
0	对接外置codec0
1	对接外置codec1
2	内置codec
3	HDMI0
4	HDMI1

表 11-34 AI 设备

AI dev	输入对接器件
0	对接外置codec0
1	对接外置codec1
2	内置codec

11.18.4.2 hi_mpi_ai_set_pub_attr

函数功能

设置AI设备属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 仅支持I2S主模式。
- 在双声道和右声道模式下，chn_cnt需为2；在左声道模式下，chn_cnt需为1。

函数原型

hi_s32 hi_mpi_ai_set_pub_attr(hi_audio_dev ai_dev, hi_aio_attr *attr);

参数说明

参数名	输入/输出	说明。
ai_dev	输入	音频设备号，取值范围：[0, 2]。
attr	输入	AI设备属性。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.2 音频相关返回码。

11.18.4.3 hi_mpi_ai_enable

函数功能

启用AI设备。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 启用AI设备前，必须先调用hi_mpi_ai_set_pub_attr接口配置AI设备属性，否则返回属性未配置错误。
- 如果AI设备已经处于enable状态，则直接返回成功。

函数原型

hi_s32 hi_mpi_ai_enable(hi_audio_dev ai_dev);

参数说明

参数名	输入/输出	说明。
ai_dev	输入	音频设备号，取值范围：[0, 2]。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.2 音频相关返回码**。

11.18.4.4 hi_mpi_ai_disable

函数功能

禁用AI设备。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 如果AI设备已经处于禁用状态，则直接返回成功。
- 禁用AI设备前，必须先调用**hi_mpi_ai_disable_chn**接口禁用该设备下已启用的所有AI通道。

函数原型

hi_s32 hi_mpi_ai_disable(hi_audio_dev ai_dev);

参数说明

参数名	输入/输出	说明。
ai_dev	输入	音频设备号，取值范围：[0, 2]。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.2 音频相关返回码**。

11.18.4.5 hi_mpi_ai_set_chn_attr

函数功能

设置AI通道属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_ai_set_chn_attr(hi_audio_dev ai_dev, hi_ai_chn ai_chn, const hi_ai_chn_attr *chn_attr);

参数说明

参数名	输入/输出	说明。
ai_dev	输入	音频设备号，取值范围：[0, 2]。 NVR（Network Video Recorder）场景中仅涉及dev2。
ai_chn	输入	音频输入通道号。 目前只支持单声道，所以ai_chn取值固定为0。
chn_attr	输入	音频通道属性结构体指针。 ● 通道属性结构体中目前只有一个成员变量，用于设置通道的工作模式，默认工作模式为HI_AI_CHN_MODE_FAST。 ● NVR（Network Video Recorder）场景中需要是使用HI_AI_CHN_MODE_NORMAL标准模式，支持调用**hi_mpi_ai_enable_resample**接口开启重采样功能。 ● HI_AI_CHN_MODE_FAST模式下不支持重采样功能，不支持绑定AI与AENC，通过AI获取的音频数据，用户需调用**hi_mpi_ai_get_frame**接口获取音频数据。

返回值说明

- 0：成功
- 非0：失败，参考**11.18.21.2 音频相关返回码**。

11.18.4.6 hi_mpi_ai_enable_chn

函数功能

启用AI通道。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 启用AI通道前，必须先调用hi_mpi_ai_set_pub_attr接口配置AI设备属性，否则返回属性未配置错误。
- 启用AI通道前，必须先调用hi_mpi_ai_enable接口启用其所属的AI设备，否则返回设备未启动的错误码。
- 录音为左声道和双声道时，ai_chn固定为0；右声道时，ai_chn固定为1。
- 如果AI设备下该通道已经处于enable状态，则直接返回成功。

函数原型

hi_s32 hi_mpi_ai_enable_chn(hi_audio_dev ai_dev, hi_ai_chn ai_chn);

参数说明

参数名	输入/输出	说明
ai_dev	输入	音频设备号，取值范围：[0, 2]。
ai_chn	输入	录音通道号。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.2 音频相关返回码。

11.18.4.7 hi_mpi_ai_disable_chn

函数功能

禁用AI通道。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 录音为左声道和双声道时，ai_chn固定为0；右声道时，ai_chn固定为1。
- 如果已经设置属性，但AI chn已经处于禁用状态，则直接返回成功；如果未设置属性，此时调用该接口会返回属性未配置的错误。

函数原型

hi_s32 hi_mpi_ai_disable_chn(hi_audio_dev ai_dev, hi_ai_chn ai_chn);

参数说明

参数名	输入/输出	说明
ai_dev	输入	音频设备号，取值范围：[0, 2]。
ai_chn	输入	录音通道号。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.2 音频相关返回码。

11.18.4.8 hi_mpi_ai_get_frame

函数功能

获取音频帧。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 获取音频帧数据前，必须先调用hi_mpi_ai_enable接口使能对应的AI设备。
- 录音为左声道和双声道时，ai_chn需固定为0；右声道时，ai_chn为1。
- 本接口与hi_mpi_ai_release_frame接口需成对出现，调用本接口获取音频帧，音频帧使用完毕之后应该及时调用hi_mpi_ai_release_frame接口释放音频帧。

函数原型

```
hi_s32 hi_mpi_ai_get_frame(hi_audio_dev ai_dev, hi_ai_chn ai_chn,
hi_audio_frame *frame, hi_aec_frame *aec_frame, hi_s32 milli_sec);
```

参数说明

参数名	输入/输出	说明
ai_dev	输入	音频设备号，取值范围：[0, 2]。
ai_chn	输入	音频输入通道号。
frame	输出	音频帧结构体指针。
aec_frame	输出	回声抵消参考帧结构体指针。 预留参数，当前设置为NULL。

参数名	输入/输出	说明
milli_sec	输入	配置获取数据的超时时间，单位是毫秒，取值范围如下： ● -1：表示阻塞模式，无数据时一直等待； ● 0：表示非阻塞模式，无数据时则报错返回； ● >0：需要配置具体的超时时间，表示超过指定时间后无数据，则报错返回。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.2 音频相关返回码。

11.18.4.9 hi_mpi_ai_release_frame

函数功能

释放音频帧。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 录音为左声道和双声道时，ai_chn需固定为0；右声道时，ai_chn为1。
- 本接口与hi_mpi_ai_get_frame接口需成对出现，调用hi_mpi_ai_get_frame接口获取音频帧，音频帧使用完毕之后应该及时调用本接口释放音频帧。
- 不支持重复调用本接口。

函数原型

hi_s32 hi_mpi_ai_release_frame(hi_audio_dev ai_dev, hi_ai_chn ai_chn, const hi_audio_frame *frame, const hi_aec_frame *aec_frame);

参数说明

参数名	输入/输出	描述
ai_dev	输入	音频设备号，取值范围：[0, 2]。
ai_chn	输入	音频输入通道号。
frame	输入	音频帧结构体指针。 预留参数，当前设置为NULL。

参数名	输入/输出	描述
aec_frame	输入	设置为NULL。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.2 音频相关返回码。

11.18.4.10 hi_mpi_ai_enable_resample

函数功能

启用AI重采样。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用hi_mpi_ai_enable_chn接口启用AI通道后，调用hi_mpi_sys_bind接口绑定VENC与AI前，调用本接口启用重采样功能。
- 调用hi_mpi_ai_disable_chn接口禁用AI通道后，如果重新启用AI通道，并使用重采样功能，需调用本接口再次启用重采样。
- 由于奈奎斯特采样定理的限制，在音频采样率为8kHz时，则所支持的音频频率小于4kHz。
- 由于AI的采样率固定为48kHz，G.711a、G.711u协议的采样率仅支持8kHz，因此需启用重采样功能；而48kHz在AAC协议采样率支持的范围内，因此使用AAC协议时，在AI时无需重采样。

函数原型

hi_s32 hi_mpi_ai_enable_resample(hi_audio_dev ai_dev, hi_ai_chn ai_chn, hi_audio_sample_rate out_sample_rate);

参数说明

参数名	输入/输出	说明。
ai_dev	输入	音频设备号，取值范围：[0, 2]。 NVR（Network Video Recorder）场景中仅涉及dev2。
ai_chn	输入	音频输入通道号。 目前只支持单声道（左声道），所以ai_chn取值固定为0。
hi_audio_sample_rate	输入	音频重采样的输出采样率，当前只支持8k。

返回值说明

- 0：成功
- 非0：失败，参考**11.18.21.2 音频相关返回码**。

11.18.4.11 hi_mpi_ai_disable_resample

函数功能

禁用AI重采样。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_ai_disable_resample(hi_audio_dev ai_dev, hi_ai_chn ai_chn);

参数说明

参数名	输入/输出	说明。
ai_dev	输入	音频设备号，取值范围：[0, 2]。 NVR（Network Video Recorder）场景中仅涉及dev2。
ai_chn	输入	音频输入通道号。 目前只支持单声道（左声道），所以ai_chn取值固定为0。

返回值说明

- 0：成功
- 非0：失败，参考**11.18.21.2 音频相关返回码**。

11.18.4.12 hi_mpi_ao_set_pub_attr

函数功能

设置AO设备属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 在设置属性之前，需要先调用**hi_mpi_ao_disable**接口禁用AO设备。

- 在双声道和右声道模式下，chn_cnt需为2；在左声道模式下，chn_cnt需为1。

函数原型

hi_s32 hi_mpi_ao_set_pub_attr(hi_audio_dev ao_dev, hi_aio_attr *attr);

参数说明

参数名	输入/输出	描述
ao_dev	输入	音频设备号，取值范围：[0, 4]。
attr	输入	AO设备属性。

返回值说明

- 0：成功。
- 非0：失败，参见11.18.21.2 音频相关返回码。

11.18.4.13 hi_mpi_ao_enable

函数功能

启用AO设备。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 在启用AO设备前，必须先调用hi_mpi_ao_set_pub_attr接口配置AO设备属性，否则会返回属性未配置的错误。
- 如果AO设备已经处于enable状态，则直接返回成功。

函数原型

hi_s32 hi_mpi_ao_enable(hi_audio_dev ao_dev);

参数说明

参数名	输入/输出	描述
ao_dev	输入	音频设备号，取值范围：[0, 4]。

返回值说明

- 0：成功。
- 非0：失败，参见11.18.21.2 音频相关返回码。

11.18.4.14 hi_mpi_ao_disable

函数功能

禁用AO设备。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 如果AO设备已经处于disable状态，则直接返回成功。
- 禁用AO设备前，必须先调用**hi_mpi_ao_disable_chn**接口禁用该设备下已启用的所有AO通道。

函数原型

hi_s32 hi_mpi_ao_disable(hi_audio_dev ao_dev);

参数说明

参数名	输入/输出	描述
ao_dev	输入	音频设备号，取值范围：[0, 4]。

返回值说明

- 0：成功。
- 非0：失败，参见**11.18.21.2 音频相关返回码**。

11.18.4.15 hi_mpi_ao_enable_chn

函数功能

启用AO通道。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 启用AO通道前，必须先调用**hi_mpi_ao_set_pub_attr**接口配置AO设备属性，否则返回属性未配置错误。
- 启用AO通道前，必须先调用**hi_mpi_ao_enable**接口启用其所属的AO设备，否则返回设备未启动的错误码。
- 录音为左声道和双声道时，ao_chn固定为0；右声道时，ao_chn固定为1。
- 如果AO设备已经处于enable状态，则直接返回成功。

函数原型

hi_s32 hi_mpi_ao_enable_chn(hi_audio_dev ao_dev, hi_ao_chn ao_chn);

参数说明

参数名	输入/输出	描述
ao_dev	输入	音频设备号，取值范围：[0, 4]。
ao_chn	输入	音频输出通道号。

返回值说明

- 0：成功。
- 非0：失败，参见**11.18.21.2 音频相关返回码**。

11.18.4.16 hi_mpi_ao_disable_chn

函数功能

禁用AO通道。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 录音为左声道和双声道时，ao_chn固定为0；右声道时，ao_chn固定为1。
- 如果已经设置属性，但AO chn已经处于禁用状态，则直接返回成功；如果未设置属性，此时调用该接口会报属性未配置的错误。

函数原型

hi_s32 hi_mpi_ao_disable_chn(hi_audio_dev ao_dev, hi_ao_chn ao_chn);

参数说明

参数名	输入/输出	描述
ao_dev	输入	音频设备号，取值范围：[0, 4]。
ao_chn	输入	音频输出通道号。

返回值说明

- 0：成功。
- 非0：失败，参见**11.18.21.2 音频相关返回码**。

11.18.4.17 hi_mpi_ao_send_frame

函数功能

发送AO音频帧。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 该接口用于用户主动发送音频帧至AO输出。
- 调用该接口发送音频帧到AO输出时，必须先调用**hi_mpi_ao_enable_chn**接口使能对应的AO通道。
- 播音为左声道和双声道时，ai_chn需固定为0；右声道时，ai_chn为1。
- 音频帧结构体的成员变量len和virt_addr需要匹配，len代表帧长（单位：字节），virt_addr为传入的音频数据的地址。**建议**每次传入的数据为一帧数据，即长度len为point_num_per_frame与位宽（16bit为2字节，24bit为3字节）的乘积。
- 对长度len限制如下：首先根据len与位宽计算出数据点数（数据点数=len/位宽对应的字节数），数据点数的范围为[1, **frame_num** * **point_num_per_frame**]，最大上限为4096。
  - 举例1：如果frame_num=2，point_num_per_frame=480（frame_num和point_num_per_frame的范围限制可以参见结构体**hi_aio_attr**），此时frame_num * point_num_per_frame=960，且960<4096，那么数据点数的范围为[1, 960]。
  - 举例2：如果frame_num=30，point_num_per_frame=4096，此时frame_num * point_num_per_frame=122880，且122880>4096，那么数据点数的范围为[1, 4096]。
  - 举例3：如果根据frame_num和point_num_per_frame确定的数据点数范围为[1, 960]。若此时len=960，位宽为16位（即2字节），计算得到数据点数为960/2=480，在[1, 960]范围之内，即符合要求；若此时len=3840，位宽为16位（即2字节），计算得到数据点数为3840/2=1920，在[1, 960]范围之外，则会返回报错。

函数原型

hi_s32 hi_mpi_ao_send_frame(hi_audio_dev ao_dev, hi_ao_chn ao_chn, const hi_audio_frame *data, hi_s32 milli_sec);

参数说明

参数名	输入/输出	说明
ao_dev	输入	音频设备号，取值范围：[0, 4]。
ao_chn	输入	音频输出通道号。
data	输入	音频帧结构体。

参数名	输入/输出	说明
milli_sec	输入	配置发送数据的超时时间，单位是毫秒，取值范围如下： ● -1：表示阻塞模式，无数据时一直等待； ● 0：表示非阻塞模式，无数据时则报错返回； ● >0：需要配置具体的超时时间，表示超过指定时间后无数据，则报错返回。

返回值说明

- 0：成功。
- 非0：失败，参见11.18.21.2 音频相关返回码。

11.18.4.18 hi_mpi_ao_get_chn_delay

函数功能

获取AO通道中当前的音频延时大小。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 该接口要在AO通道成功启用之后调用。
- 获取AO通道中当前的音频延时大小，当延时查询到的延时为0时停止等待，进入播音停止流程。

函数原型

hi_s32 hi_mpi_ao_get_chn_delay(hi_audio_dev ao_dev, hi_ao_chn ao_chn, hi_u32 *milli_sec);

参数说明

参数名	输入/输出	说明
ao_dev	输入	音频设备号，取值范围：[0, 4]。
ao_chn	输入	音频输出通道号。
milli_sec	输出	音频延时的指针。音频延时大小以毫秒为单位。

返回值说明

- 0：成功。
- 非0：失败，参见11.18.21.2 音频相关返回码。

11.18.4.19 hi_mpi_ao_enable_resample

函数功能

启用AO重采样。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用hi_mpi_ao_enable_chn接口启用AO通道之后，调用hi_mpi_sys_bind接口绑定ADEC与AO之前，调用本接口启用重采样功能。
- 调用hi_mpi_ao_disable_chn接口禁用AO通道后，如果重新启用AO通道，并使用重采样功能，需调用本接口再次启用重采样。
- AO重采样的输入采样率必须与AO设备属性配置的采样率不相同。
- 由于奈奎斯特采样定理的限制，在音频采样率为8kHz时，则所支持的音频频率小于4kHz。
- 由于AO的采样率固定为48kHz，G.711a、G.711u协议的采样率仅支持8kHz，因此需启用重采样功能；而48kHz在AAC协议采样率支持的范围内，因此使用AAC协议时，在AO时无需重采样。

函数原型

hi_s32 hi_mpi_ao_enable_resample(hi_audio_dev ao_dev, hi_ao_chn ao_chn, hi_audio_sample_rate in_sample_rate);

参数说明

参数名	输入/输出	说明。
ao_dev	输入	音频设备号，取值范围：[0, 4]。 NVR（Network Video Recorder）场景中仅涉及dev2、dev3。
ao_chn	输入	音频输出通道号。目前只支持单声道（左声道），所以ao_chn取值固定为0。
hi_audio_sample_rate	输入	音频重采样的输入采样率,目前只支持8k。

返回值说明

- 0：成功
- 非0：失败，参考11.18.21.2 音频相关返回码。

11.18.4.20 hi_mpi_ao_disable_resample

函数功能

禁用 AO 重采样。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_ao_disable_resample(hi_audio_dev ao_dev, hi_ao_chn ao_chn);

参数说明

参数名	输入/输出	说明。
ao_dev	输入	音频设备号，取值范围：[0, 4]。 NVR（Network Video Recorder）场景中仅涉及dev2、dev3。
ao_chn	输入	音频输出通道号。 目前只支持单声道（左声道），所以ao_chn取值固定为0。

返回值说明

- 0：成功
- 非0：失败，参考11.18.21.2 音频相关返回码。

11.18.5 AENC 音频编码/ADEC 音频解码

11.18.5.1 功能及约束说明

当前仅支持G.711a、G.711u、AAC（Advanced Audio Coding）协议、单声道数据进行编码、解码，且为软件编码、解码。**本章中的接口均不支持多进程，且同一个设备ID不支持在多线程中使用。**

音频编码时，G.711a、G.711u码流遵循以下表格中描述的帧结构，**即在每帧码流数据的净荷数据之前填充有4个字节的帧头**；音频解码时，需要读取相应的帧头信息。在G.711a、G.711u码流中，目前帧头中的数据净荷长度需为40，80，120，160，240（单位：hi_s16），所以对应的每帧采样点数即为80，160，240，320，480。但在VQE（Voice Quality Enhancement）框架中，编解码操作是与AI、AO绑定在一起，所以支持的每帧采样点数范围缩小。

参数位置(单位：hi_s16)	参数比特位说明	参数含义
0	[15:8]	数据帧类型标志位。 01：语音帧； 其他：保留。
	[7:0]	保留。
1	[15:8]	帧循环计数器：0～255。
	[7:0]	数据净荷长度(单位：hi_s16)。
2	[15:0]	净荷数据。
3	[15:0]	净荷数据。
……	[15:0]	净荷数据。
n+1	[15:0]	净荷数据。
n+2	[15:0]	净荷数据。

G.711a、G.711u、AAC协议的相关规格及优缺点如下：

协议	采样率	每帧采样点数	码率（kbps）	压缩率	CPU消耗	描述
G.711a、G.711u	8kHz	80/160/240/320/480	64	2	1 MHz	● 优点：语音质量最好；CPU消耗小；支持广泛，协议免费。 ● 缺点：压缩效率低。

协议	采样率	每帧采样点数	码率（kbps）	压缩率	CPU消耗	描述
AAC 仅支持MPEG-4版本、无CRC（Cyclic Redundancy Check）校验的音频码流	48kHz	1024	[48, 256]	与码率相关，码率越小，压缩率越大	-	• 优点：持更高采样率的编解码，能够保留原始音频的更多高频信息，同时拥有更多的码率范围选择，压缩率更高。 • 缺点：算法复杂度高。

11.18.5.2 hi_mpi_aenc_create_chn

函数功能

创建音频编码通道。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

通道如果已经被创建，则会返回通道已创建的错误。

函数原型

hi_s32 hi_mpi_aenc_create_chn(hi_aenc_chn aenc_chn, const hi_aenc_chn_attr *attr);

参数说明

参数名	输入/输出	说明。
aenc_chn	输入	通道号，取值范围[0，6)
attr	输入	音频编码通道属性指针。

返回值说明

- 0：成功
- 非0：失败，参考11.18.21.2 音频相关返回码。

11.18.5.3 hi_mpi_aenc_destroy_chn

函数功能

销毁音频编码通道。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_aenc_destroy_chn(hi_aenc_chn aenc_chn);

参数说明

参数名	输入/输出	说明。
aenc_chn	输入	通道号，取值范围[0，6)

返回值说明

- 0：成功
- 非0：失败，参考11.18.21.2 音频相关返回码。

11.18.5.4 hi_mpi_aenc_get_stream

函数功能

获取音频编码后的码流。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 必须调用hi_mpi_aenc_create_chn接口创建通道后才可能获取码流，否则直接返回失败。
- 本接口与hi_mpi_aenc_release_stream接口需成对出现，调用本接口获取码流，码流使用完毕之后应该及时调用hi_mpi_aenc_release_stream接口释放码流。

函数原型

hi_s32 hi_mpi_aenc_get_stream(hi_aenc_chn aenc_chn, hi_audio_stream *stream, hi_s32 milli_sec);

参数说明

参数名	输入/输出	说明。
aenc_chn	输入	编码通道号，取值范围[0，6)
stream	输出	获取的音频码流。
milli_sec	输入	配置获取数据的超时时间，单位是毫秒，取值范围如下： ● -1：表示阻塞模式，无数据时一直等待； ● 0：表示非阻塞模式，无数据时则报错返回； ● >0：需要配置具体的超时时间，表示超过指定时间后无数据，则报错返回。

返回值说明

- 0：成功
- 非0：失败，参考**11.18.21.2 音频相关返回码**。

11.18.5.5 hi_mpi_aenc_release_stream

函数功能

释放音频编码后的码流。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 必须调用**hi_mpi_aenc_create_chn**接口创建通道后才可能获取码流，否则直接返回失败。
- 本接口与hi_mpi_aenc_get_stream接口需成对出现，调用hi_mpi_aenc_get_stream接口获取码流，码流使用完毕之后应该及时调用本接口释放码流。
- 不支持重复调用本接口。

函数原型

hi_s32 hi_mpi_aenc_release_stream(hi_aenc_chn aenc_chn, const hi_audio_stream *stream);

参数说明

参数名	输入/输出	说明。
aenc_chn	输入	编码通道号，取值范围[0，6)
stream	输出	待释放的音频码流。

返回值说明

- 0：成功
- 非0：失败，参考11.18.21.2 音频相关返回码。

11.18.5.6 hi_mpi_adec_create_chn

函数功能

创建音频解码通道。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

通道如果已经被创建，则会返回通道已创建的错误。

函数原型

hi_s32 hi_mpi_adec_create_chn(hi_adec_chn adec_chn, const hi_adec_chn_attr *attr);

参数说明

参数名	输入/输出	说明。
adec_chn	输入	通道号，取值范围[0，6)
attr	输入	音频解码通道属性指针。

返回值说明

- 0：成功
- 非0：失败，参考11.18.21.2 音频相关返回码。

11.18.5.7 hi_mpi_adec_destroy_chn

函数功能

销毁音频解码通道。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_adec_destroy_chn(hi_adec_chn adec_chn);

参数说明

参数名	输入/输出	说明。
adec_chn	输入	通道号，取值范围[0，6)

返回值说明

- 0：成功
- 非0：失败，参考**11.18.21.2 音频相关返回码**。

11.18.5.8 hi_mpi_adec_send_stream

函数功能

向音频解码通道发送码流。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 创建音频解码通道时（**hi_mpi_adec_create_chn**）可以通过hi_adec_mode变量指定解码方式为pack方式或stream方式，**推荐使用pack方式**。
  - **pack方式**：用于用户确认当前码流包为一帧数据编码结果的情况下，解码器会直接进行对其解码，如果不是一帧，解码器会出错。

    例如，对于G.711a文件来说，帧头中有数据长度信息，假设此时的数据长度为240（单位：s16），即为240×2=480字节，由于帧头长度为4字节，所以完整的一帧长度为480+4=484字节，即每次数据应该读484字节。
  - **stream方式**：用于不确定码流包为是否为一帧的情况，该方式效率较低，且可能会有延迟或者缓存满的情况。使用场景适合程序长时间运行不退出、发送端和接收端无匹配要求、效率不高的场景。
- 发送数据时必须保证音频解码通道已经被创建，否则直接返回失败。
- 支持阻塞或非阻塞方式发送码流，当阻塞方式发送码流时，如果用于缓存解码后的音频帧的buffer满，则此接口调用会被阻塞，直至解码后的音频帧数据被取走。
- 该接口只支持单声道、且带有符合要求的语音帧头的G.711a、G.711u音源进行解码。

函数原型

```
hi_s32 hi_mpi_adec_send_stream(hi_adec_chn adec_chn, const
hi_audio_stream *stream, hi_bool block);
```

参数说明

参数名	输入/输出	说明。
adec_chn	输入	通道号，取值范围[0，6)
stream	输入	音频码流。
block	输入	阻塞标识： ● HI_TRUE：阻塞。 ● HI_FALSE：非阻塞。

返回值说明

- 0：成功
- 非0：失败，参考11.18.21.2 音频相关返回码。

11.18.6 音量调节

11.18.6.1 功能描述

- 音量调节可以实现对在内置codec（即 device 2）播音、录音时的音量调节。
- 调节音量共涉及DACL（播音左声道）、DACR（播音右声道）、ADCL（录音左声道）、ADCR（录音右声道）四种情况。
- 需在播音录音开始之前调节音量，播音录音结束之后音量不会回到默认大小，而是保持在调节之后的状态。
- 多进程调节音量时，音量的调节效果以最后一次调节的大小为准。
- 播音时如果出现削波现象、声音失真，可能是音量过大，需要调小音量，或者调整音源。
- 录音时如果出现削波现象、声音失真，可能外部输入的信号过大，用户需自行调整外部输入的信号。

11.18.6.2 HI_ACODEC_SET_DACL_VOLUME

函数功能

左声道输出音量控制。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 左声道输出音量范围：[0, 127]，赋值越大，音量越小。赋值为 0 时，音量最大，为6db，赋值为 127 时，音量最小，为静音。
- 例如hi_acodec_volume_ctrl结构体中volume_ctrl变量被配置为x，音量会被调节为（6-x）db。

函数原型

int ioctl(int fd, HI_ACODEC_SET_DACL_VOLUME, hi_acodec_volume_ctrl *arg);

参数说明

参数名	输入/输出	说明
fd	输入	Audio Codec设备文件描述符。
HI_ACODEC_SET_DACL_VOLUME	输入	ioctl号。
arg	输入	内置Audio Codec音量控制参数。

返回值说明

- 0：成功。
- 非0：失败。

11.18.6.3 HI_ACODEC_SET_DACR_VOLUME

函数功能

右声道输出音量控制。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 右声道输出音量范围：[0, 127]，赋值越大，音量越小。赋值为 0 时，音量最大，为6db，赋值为 127 时，音量最小，为静音。
- 例如hi_acodec_volume_ctrl结构体中volume_ctrl变量被配置为x，音量会被调节为 (6-x) db。

函数原型

int ioctl(int fd, HI_ACODEC_SET_DACR_VOLUME, hi_acodec_volume_ctrl*arg);

参数说明

参数名	输入/输出	说明
fd	输入	Audio Codec设备文件描述符。
HI_ACODEC_SET_DACR_VOLUME	输入	ioctl号。
arg	输入	内置Audio Codec音量控制参数。

返回值说明

- 0：成功。
- 非0：失败。

11.18.6.4 HI_ACODEC_GET_DACL_VOLUME

函数功能

获取左声道输出的音量。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 左声道输出音量范围：[0, 127]，赋值越大，音量越小。赋值为 0 时，音量最大，为6db，赋值为 127 时，音量最小，为静音。
- 例如获取hi_acodec_volume_ctrl结构体中volume_ctrl变量的值为x，则代表音量被调节为（6-x）db。

函数原型

int ioctl(int fd, HI_ACODEC_GET_DACL_VOLUME, hi_acodec_volume_ctrl*arg);

参数说明

参数名	输入/输出	说明
fd	输入	Audio Codec设备文件描述符。
HI_ACODEC_GET_DACL_VOLUME	输入	ioctl号。
arg	输出	内置Audio Codec音量控制参数。

返回值说明

- 0：成功。

- 非0：失败。

11.18.6.5 HI_ACODEC_GET_DACR_VOLUME

函数功能

获取右声道输出的音量控制。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 右声道输出音量范围：[0, 127]，赋值越大，音量越小。赋值为 0 时，音量最大，为6db，赋值为 127 时，音量最小，为静音。
- 例如获取hi_acodec_volume_ctrl结构体中volume_ctrl变量的值为x，则代表音量被调节为（6-x）db。

函数原型

int ioctl(int fd, HI_ACODEC_GET_DACR_VOLUME, hi_acodec_volume_ctrl*arg);

参数说明

参数名	输入/输出	说明
fd	输入	Audio Codec设备文件描述符。
HI_ACODEC_GET_DACR_VOLUME	输入	ioctl号。
arg	输出	内置Audio Codec音量控制参数。

返回值说明

- 0：成功。
- 非0：失败。

11.18.6.6 HI_ACODEC_SET_ADCL_VOLUME

函数功能

左声道输入音量控制。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 左声道输入音量范围：[0, 127]，赋值越大，音量越小。赋值为 0 时，音量最大，为30db，赋值为 127 时，音量最小，为-97db。

- 例如hi_acodec_volume_ctrl结构体中volume_ctrl变量被配置为y，音量会被调节为 (30-y) db。

函数原型

int ioctl(int fd, HI_ACODEC_SET_ADCL_VOLUME, hi_acodec_volume_ctrl*arg);

参数说明

参数名	输入/输出	说明
fd	输入	Audio Codec设备文件描述符。
HI_ACODEC_SET_ADCL_VOLUME	输入	ioctl号。
arg	输入	内置Audio Codec音量控制参数。

返回值说明

- 0：成功。
- 非0：失败。

11.18.6.7 HI_ACODEC_SET_ADCR_VOLUME

函数功能

右声道输入音量控制。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 右声道输入音量范围：[0, 127]，赋值越大，音量越小。赋值为 0 时，音量最大，为30db，赋值为 127 时，音量最小，为-97db。
- 例如hi_acodec_volume_ctrl结构体中volume_ctrl变量被配置为y，音量会被调节为 (30-y) db。

函数原型

int ioctl(int fd, HI_ACODEC_SET_ADCR_VOLUME, hi_acodec_volume_ctrl*arg);

参数说明

参数名	输入/输出	说明
fd	输入	Audio Codec设备文件描述符。

参数名	输入/输出	说明
HI_ACODEC_SET_ADCR_VOLUME	输入	ioctl号。
arg	输入	内置Audio Codec音量控制参数。

返回值说明

- 0：成功。
- 非0：失败。

11.18.6.8 HI_ACODEC_GET_ADCL _VOLUME

函数功能

获取左声道输入的音量控制。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 左声道输入音量范围：[0, 127]，赋值越大，音量越小。赋值为 0 时，音量最大，为30db，赋值为 127 时，音量最小，为-97db。
- 例如获取hi_acodec_volume_ctrl结构体中volume_ctrl变量的值为y，则代表音量被调节为（30-y）db。

函数原型

int ioctl(int fd, HI_ACODEC_GET_ ADCL _VOLUME, hi_acodec_volume_ctrl*arg);

参数说明

参数名	输入/输出	说明
fd	输入	Audio Codec设备文件描述符。
HI_ACODEC_GET_ ADCL _VOLUME	输入	ioctl号。
arg	输出	内置Audio Codec音量控制参数。

返回值说明

- 0：成功。
- 非0：失败。

11.18.6.9 HI_ACODEC_GET_ADCR _VOLUME

函数功能

获取右声道输入的音量控制。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 右声道输入音量范围：[0, 127]，赋值越大，音量越小。赋值为 0 时，音量最大，为30db，赋值为 127 时，音量最小，为-97db。
- 例如获取hi_acodec_volume_ctrl结构体中volume_ctrl变量的值为y，则代表音量被调节为（30-y）db。

函数原型

int ioctl(int fd, HI_ACODEC_GET_ADCR _VOLUME, hi_acodec_volume_ctrl*arg);

参数说明

参数名	输入/输出	说明
fd	输入	Audio Codec设备文件描述符。
HI_ACODEC_GET_ADCR _VOLUME	输入	ioctl号。
arg	输出	内置Audio Codec音量控制参数。

返回值说明

- 0：成功。
- 非0：失败。

11.18.7 ISP 系统控制及 3A 算法注册

本文档中接口，如无特殊说明，不支持多进程。

11.18.7.1 功能描述

ISP（Image Signal Processing）系统控制

系统控制部分用于注册3A算法、注册Sensor驱动、初始化ISP firmware、运行ISP firmware、退出ISP firmware、配置ISP属性等功能。 ISP的Firmware包含三部分，一部分是ISP控制单元和基础算法库，一部分是AE/AWB/AF算法库，一部分是Sensor库。

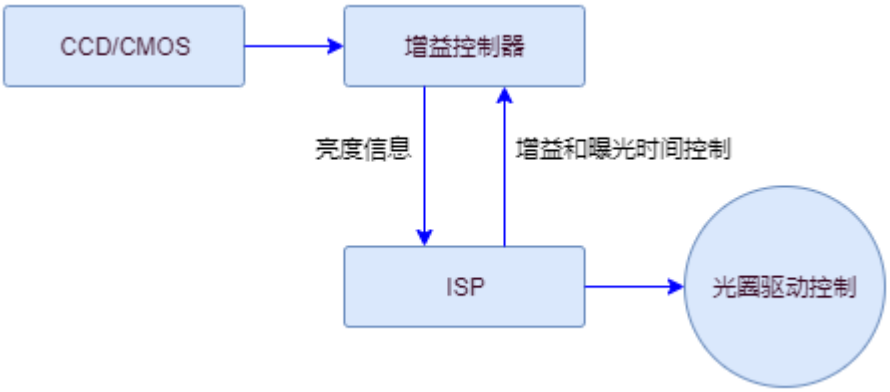
AE（Auto Exposure）自动曝光

AE模块实现的功能是：根据自动测光系统获得当前图像的曝光量，再自动配置镜头光圈、Sensor快门及增益来获得最佳的图像质量。

自动曝光的算法主要分：

- 光圈优先，优先调整光圈到合适的位置，再分配曝光时间和增益，只适合p-iris 镜头，这样能均衡噪声和景深。
- 快门优先，优先分配曝光时间，再分配Sensor增益和ISP增益，这样拍摄的图像噪声会比较小。
- 增益优先，优先分配Sensor增益和ISP增益，再分配曝光时间，适合拍摄运动物体的场景。当前AE算法也支持客户设定更灵活的曝光分配策略，AE模块的工作流程下图所示。

图 11-8 AE 模块工作流程图



AWB（Automatic White Balance）自动白平衡

色温随可见光的光谱成分变化而变化，在低色温光源下，白色物体偏红，在高色温光源下，白色物体偏蓝，人眼可根据大脑的记忆判断，识别物体的真实颜色。AWB算法的功能是降低外界光源对物体真实颜色的影响，使得我们采集的颜色信息转变为在理想日光光源下的无偏色信息。

11.18.7.2 基本概念

- 曝光时间：sensor积累电荷的时间，是sensor pixel从开始曝光到电量被读出的这段时间。
- 曝光增益：对sensor的输出电荷的总的放大系数，一般有数字增益和模拟增益，模拟增益引入的噪声会稍小，所以一般优先用模拟增益。
- 光圈：光圈是镜头中可以改变中间孔大小的机械装置。
- 抗闪烁：由于电灯的电源工频与sensor的帧率不匹配而导致的画面闪烁，一般通过限定曝光时间和修改sensor的帧率来达到抗闪烁的效果。
- 色温：色温是按**绝对黑体**定义的，**光源**辐射在可见区和绝对黑体的辐射完全相同时，此时黑体的**温度**称此光源的色温。
- 白平衡：在不同色温的光源下，白色在传感器中的响应会偏蓝或偏红。白平衡算法通过调整 R, G, B 三个颜色通道的强度，使白色真实呈现。

11.18.7.3 hi_mpi_isp_sensor_reg_callback

函数功能

ISP提供的Sensor注册的回调接口，用于获取差异化的初始化参数，并控制sensor。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_isp_sensor_reg_callback(hi_vi_pipe vi_pipe, const hi_isp_sns_attr_info *sns_attr_info, const hi_isp_sensor_register *sns_register)

约束说明

此接口不支持多进程操作。

参数说明

参数名	输入/输出	说明
vi_pipe	输入	VI PIPE号。 取值范围：[0, 12)。
sns_attr_info	输入	向ISP注册的Sensor的属性。 该结构体内的sensor_id是sensor库中自定义的值，需确保唯一，主要用于校对向ISP注册的sensor和向3A注册的sensor是否为同一个sensor。
sns_register	输入	Sensor注册结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见**ISP返回码**。

11.18.7.4 hi_mpi_isp_sensor_unreg_callback

函数功能

ISP提供的取消Sensor注册的回调接口。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_isp_sensor_unreg_callback(hi_vi_pipe vi_pipe, hi_sensor_id sensor_id)

约束说明

此接口不支持多进程操作。

参数说明

参数名	输入/输出	说明
vi_pipe	输入	VI PIPE号。 取值范围：[0, 12)。
sensor_id	输入	向ISP注册的Sensor的Id。 sensor_id是sensor库中自定义的值，需确保唯一，主要用于校对向ISP反注册的sensor和向3A反注册的sensor是否为同一个sensor。

返回值说明

- 0：成功
- 非0：失败，参见**ISP返回码**。

11.18.7.5 hi_mpi_isp_mem_init

函数功能

初始化ISP内部资源。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_isp_mem_init(hi_vi_pipe vi_pipe)

约束说明

- 调用该接口前，必须先调用**hi_mpi_isp_sensor_reg_callback**接口向ISP注册Sensor。
- 不支持多进程，必须要与hi_mpi_isp_sensor_reg_callback、hi_mpi_isp_ae_lib_reg_callback、hi_mpi_isp_awb_lib_reg_callback、hi_mpi_isp_init、hi_mpi_isp_run、hi_mpi_isp_exit、hi_mpi_ae_register、hi_mpi_awb_register接口在同一个进程调用。
- 当前业务正在运行hi_mpi_isp_run接口时，不能调用本接口。

- 推荐调用hi_mpi_isp_exit接口后，再调用本接口重新初始化。
- 不支持相同vi_pipe时，多线程执行ISP创建和销毁（即多线程同时调用hi_mpi_isp_sensor_reg_callback、hi_mpi_isp_ae_lib_reg_callback、hi_mpi_isp_awb_lib_reg_callback、hi_mpi_isp_init、hi_mpi_isp_run、hi_mpi_isp_exit、hi_mpi_ae_register、 hi_mpi_awb_register接口）

参数说明

参数名	输入/输出	说明
vi_pipe	输入	VI PIPE号。 取值范围：[0, 12)。

返回值说明

- 0：成功
- 非0：失败，参见**ISP返回码**。

11.18.7.6 hi_mpi_isp_set_pub_attr

函数功能

设置ISP公共属性，例如裁剪分辨率、帧率等。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_isp_set_pub_attr(hi_vi_pipe vi_pipe, const hi_isp_pub_attr *pub_attr)

约束说明

- 调用本接口前，必须先调用**hi_mpi_isp_mem_init**接口初始化ISP内部资源。
- ISP支持运行过程中动态裁剪图像的起始位置。
- 调用本接口后ISP内的处理流程：

  ISP firmware判断图像WDR模式、分辨率、帧率是否变化，若都不变则直接返回；否则，ISP firmware会调用通过**hi_mpi_isp_sensor_reg_callback**接口注册的回调函数**pfn_cmos_set_wdr_mode**、**pfn_cmos_set_image_mode**改变Sensor模式；

  - 若sensor模式不改变（回调函数返回值-2），判断ISP的裁剪宽高是否变化，若有变化，ISP firmware切换分辨率，并调用通过**hi_mpi_isp_sensor_reg_callback**接口注册的回调函数**pfn_cmos_sensor_init**重新配置sensor；若无变化，就不变。
  - 若sensor模式改变（回调函数返回值为0），则ISP firmware会调用通过**hi_mpi_isp_sensor_reg_callback**接口注册的回调函数**pfn_cmos_sensor_init**重新配置Sensor；

- 用户可以修改回调函数**pfn_cmos_set_image_mode**的实现来调整sensor模式切换的顺序。如只提供了5M30fps和1080P60fps初始化序列的Sensor，若要运行1080P30fps，可以从5M30fps裁剪得到，也可以从1080P60fps降帧得到， 修改**pfn_cmos_set_image_mode**函数实现即可。
- 切换线性模式和帧WDR模式时，同样会判断**pfn_cmos_set_image_mode**的返回值， 因此线性模式和帧WDR模式应该采用不同的 image_mode，才能保证切换成功。

参数说明

参数名	输入/输出	说明
vi_pipe	输入	VI PIPE号。 取值范围：[0, 12)。
pub_attr	输入	ISP公共属性。

返回值说明

- 0：成功
- 非0：失败，参见**ISP返回码**。

11.18.7.7 hi_mpi_isp_get_pub_attr

函数功能

获取ISP公共属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_isp_get_pub_attr(hi_vi_pipe vi_pipe, hi_isp_pub_attr *pub_attr)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	VI PIPE号。 取值范围：[0, 12)。
pub_attr	输出	ISP公共属性。

返回值说明

- 0：成功
- 非0：失败，参见**ISP返回码**。

11.18.7.8 hi_mpi_isp_init

函数功能

初始化ISP firmware。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_isp_init(hi_vi_pipe vi_pipe)

约束说明

- 调用本接口前，必须先调用**hi_mpi_isp_set_pub_attr**接口设置图像公共属性。
- 为确保前一次进程退出前资源已完全清理，推荐先调用hi_mpi_isp_exit接口，再调用本接口初始化ISP firmware。
- 不支持多进程，必须要与**hi_mpi_isp_sensor_reg_callback**、**hi_mpi_isp_run**、**hi_mpi_isp_exit**、**hi_mpi_ae_register**、 **hi_mpi_awb_register**接口在同一个进程调用。
- 不支持重复调用本接口。
- 不支持相同vi_pipe时，多线程执行ISP创建和销毁（即多线程同时调用**hi_mpi_isp_sensor_reg_callback**、**hi_mpi_isp_run**、**hi_mpi_isp_exit**、**hi_mpi_ae_register**、 **hi_mpi_awb_register**接口）。
- ISP初始化后，需要一帧时间给硬件读取算法系数表。所以调用本接口后一 帧时间内，不能调用hi_mpi_vi_stop_pipe接口停止VI PIPE。

参数说明

参数名	输入/输出	说明
vi_pipe	输入	VI PIPE号。 取值范围：[0, 12)。

返回值说明

- 0：成功
- 非0：失败，参见**ISP返回码**。

11.18.7.9 hi_mpi_isp_run

函数功能

运行ISP firmware。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_isp_run(hi_vi_pipe vi_pipe)

约束说明

- 调用本接口前，必须先调用hi_mpi_isp_init接口初始化ISP firmware。
- 不支持多进程，必须要与hi_mpi_isp_sensor_reg_callback、hi_mpi_isp_init、hi_mpi_isp_exit、hi_mpi_ae_register、 hi_mpi_awb_register接口在同一个进程调用。
- 该接口是阻塞接口，建议用户采用单独的线程调用本接口。

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。

返回值说明

- 0：成功
- 非0：失败，参见ISP返回码。

11.18.7.10 hi_mpi_isp_exit

函数功能

退出ISP firmware。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_isp_exit(hi_vi_pipe vi_pipe)

约束说明

- 调用hi_mpi_isp_run之后，再调用本接口退出ISP firmware。
- 不支持多进程，必须要与hi_mpi_isp_sensor_reg_callback、hi_mpi_isp_init、hi_mpi_isp_run、hi_mpi_ae_register、 hi_mpi_awb_register接口在同一个进程调用。
- 支持重复调用本接口。
- 不支持相同vi_pipe时，多线程执行ISP创建和销毁（即多线程同时调用hi_mpi_isp_sensor_reg_callback、hi_mpi_isp_run、hi_mpi_isp_exit、hi_mpi_ae_register、 hi_mpi_awb_register接口）。

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。

返回值说明

- 0：成功
- 非0：失败，参见**ISP返回码**。

11.18.7.11 hi_mpi_isp_set_sns_slave_attr

函数功能

设置从模式Sensor行场同步信号。

从模式sensor需要其提供行同步信号XHS和场同步信号XVS，在这两个信号的控制下进行曝光与数据读出，这个接口主要配置同步信号发生模块，使其输出sensor要求的行场时序。此接口一般在sensor库里调用。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_isp_set_sns_slave_attr(hi_slave_dev slave_dev, const hi_isp_slave_sns_sync* sns_sync)

约束说明

从信号有两组绑定关系，首先pipe 和vsync之间有绑定关系，其次vsync内部还可选择不同的信号源slave，即不同pipe可以选择不同的从信号设备vsync，不同的从信号设备vsync又可以选择不同的信号源slave。

参数说明

参数名	输入/输出	说明
slave_dev	输入	从信号设备号。
sns_sync	输入	同步信号配置。

返回值说明

- 0：成功
- 非0：失败，参见**ISP返回码**。

11.18.7.12 hi_mpi_isp_get_sns_slave_attr

函数功能

获取从模式sensor行场同步信号。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_isp_get_sns_slave_attr(hi_slave_dev slave_dev, hi_isp_slave_sns_sync *sns_sync)

参数说明

参数名	输入/输出	说明
slave_dev	输入	从信号设备号。
sns_sync	输出	同步信号配置。

返回值说明

- 0：成功
- 非0：失败，参见**ISP返回码**。

11.18.7.13 hi_mpi_ae_register

函数功能

向ISP注册昇腾AE库。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_ae_register(hi_vi_pipe vi_pipe, hi_isp_3a_alg_lib *ae_lib)

约束说明

- AE库可以注册多个实例。
- 此接口不支持多进程操作。

参数说明

参数名	输入/输出	说明
vi_pipe	输入	VI PIPE号。 取值范围：[0, 12)。
ae_lib	输入	AE算法库结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见**ISP返回码**。

11.18.7.14 hi_mpi_ae_unregister

函数功能

向ISP取消注册昇腾AE库。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_ae_unregister(hi_vi_pipe vi_pipe, hi_isp_3a_alg_lib *ae_lib)

约束说明

此接口不支持多进程操作。

参数说明

参数名	输入/输出	说明
vi_pipe	输入	VI PIPE号。 取值范围：[0, 12)。
ae_lib	输入	AE 算法库结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见**ISP返回码**。

11.18.7.15 hi_mpi_ae_sensor_reg_callback

函数功能

AE库提供的sensor注册的回调接口。AE通过sensor注册的一系列回调接口，获取差异化的初始化参数，并控制sensor。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_ae_sensor_reg_callback(hi_vi_pipe vi_pipe, hi_isp_3a_alg_lib *ae_lib, const hi_isp_sns_attr_info *sns_attr_info, const hi_isp_ae_sensor_register *pregister)

约束说明

此接口不支持多进程操作。

参数说明

参数名	输入/输出	说明
vi_pipe	输入	VI PIPE号。 取值范围：[0, 12)。
ae_lib	输入	AE算法库结构体指针。
sns_attr_info	输入	向AE注册的Sensor的属性。 该结构体内的sensor_id是sensor 库中自定义的值，需确保唯一，主要用于校对向ISP注册的sensor和向3A注册的sensor是否为同一个sensor。
pregister	输入	Sensor注册结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见**ISP返回码**。

11.18.7.16 hi_mpi_ae_sensor_unreg_callback

函数功能

AE库提供的sensor注销的回调接口。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_ae_sensor_unreg_callback(hi_vi_pipe vi_pipe, hi_isp_3a_alg_lib *ae_lib, hi_sensor_id sensor_id)

约束说明

此接口不支持多进程操作。

参数说明

参数名	输入/输出	说明
vi_pipe	输入	VI PIPE号。 取值范围：[0, 12)。
ae_lib	输入	AE算法库结构体指针。
sensor_id	输入	向AE反注册的Sensor的Id。 sensor_id是sensor库中自定义的值，主要用于校对向ISP反注册的sensor和向3A反注册的sensor是否为同一个sensor。

返回值说明

- 0：成功
- 非0：失败，参见**ISP返回码**

11.18.7.17 hi_mpi_awb_register

函数功能

向ISP注册昇腾AWB库。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_awb_register(hi_vi_pipe vi_pipe, hi_isp_3a_alg_lib *awb_lib)

约束说明

- AWB库可以注册多个实例。
- 此接口不支持多进程操作。

参数说明

参数名	输入/输出	说明
vi_pipe	输入	VI PIPE号。 取值范围：[0, 12)。
awb_lib	输入	AWB算法库结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见**ISP返回码**。

11.18.7.18 hi_mpi_awb_unregister

函数功能

向ISP取消注册昇腾AWB库。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_awb_unregister(hi_vi_pipe vi_pipe, hi_isp_3a_alg_lib *awb_lib)

约束说明

此接口不支持多进程操作。

参数说明

参数名	输入/输出	说明
vi_pipe	输入	VI PIPE号。 取值范围：[0, 12)。
awb_lib	输入	AWB算法库结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见**ISP返回码**。

11.18.7.19 hi_mpi_awb_sensor_reg_callback

函数功能

AWB库提供的Sensor注册的回调接口。AWB通过Sensor注册的一系列回调接口，获取差异化的初始化参数，并控制Sensor。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_awb_sensor_reg_callback(hi_vi_pipe vi_pipe, hi_isp_3a_alg_lib *awb_lib, const hi_isp_sns_attr_info *sns_attr_info, const hi_isp_awb_sensor_register * awb_sns_register)

约束说明

此接口不支持多进程操作。

参数说明

参数名	输入/输出	说明
vi_pipe	输入	VI PIPE号。 取值范围：[0, 12)。
awb_lib	输入	AWB算法库结构体指针。
sns_attr_info	输入	向AWB注册的Sensor的属性。 该结构体内的sensor_id是sensor库中自定义的值，需确保唯一，主要用于校对向ISP注册的sensor和向3A注册的sensor是否为同一个sensor。
awb_sns_register	输入	Sensor注册结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见**ISP返回码**。

参考样例

```
hi_isp_3a_alg_lib awb_lib;
hi_isp_awb_sensor_register awb_register;
hi_isp_sns_attr_info sns_attr_info;
hi_isp_awb_sensor_exp_func *exp_funcs = &awb_register.sns_exp;
memset(exp_funcs, 0, sizeof(hi_isp_awb_sensor_exp_func));
exp_funcs-> pfn_cmos_get_awb_default = cmos_get_awb_default;
hi_vi_pipe vi_pipe = 0;
awb_lib.id = 0;
sns_attr_info.sensor_id = IMX178_ID;
```

```
strcpy(awb_lib. lib_name, HI_AWB_LIB_NAME);
ret = hi_mpi_awb_sensor_reg_callback(vi_pipe, &awb_lib, &sns_attr_info, &awb_register);
if (ret) {
    printf("sensor register callback function to awb lib failed!\n");
return ret;
}
```

11.18.7.20 hi_mpi_awb_sensor_unreg_callback

函数功能

AWB库提供的sensor注销的回调接口。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_awb_sensor_unreg_callback(hi_vi_pipe vi_pipe, hi_isp_3a_alg_lib *awb_lib, hi_sensor_id sensor_id)

约束说明

- 此接口不支持多进程操作。

参数说明

参数名	输入/输出	说明
vi_pipe	输入	VI PIPE号。 取值范围：[0, 12)。
awb_lib	输入	AWB算法库结构体指针。
sensor_id	输入	向AWB注册的sensor的Id。

返回值说明

- 0：成功
- 非0：失败，参见**ISP返回码**。

11.18.8 MIPI Rx ioctl 命令字

11.18.8.1 功能描述

MIPI Rx通过低电压差分信号接收原始视频数据，将接收到的串行差分信号（serial differential signal）转化为DC（Digital Camera）时序后传递给下一级模块VICAP（Video Capture）。MIPI Rx支持MIPI D-PHY、sub-LVDS、HiSPi（High-Speed Serial Pixel Interface）等串行视频信号输入。

SLVS-EC接口由SONY公司定义，用于高帧率和高分辨率图像采集，它可以将高速串行的数据转化为DC（Digital Camera）时序后传递给下一级模块 VICAP（Video

Capture ）。SLVS-EC串行视频接口可以提供更高的传输带宽，更低的功耗，在组包方式上，数据的冗余度也更低。在应用中SLVS-EC接口提供了更加可靠和稳定的传输。

说明

当前版本仅配套支持MIPI CSI接口以及SLVS-EC接口。

MIPI Rx

MIPI Rx是一个支持多种差分视频输入接口的采集单元，通过combo-PHY接收MIPI/LVDS/sub-LVDS/HiSPi接口的数据，通过不同的功能模式配置，MIPI Rx可以支持多种速度和分辨率的数据传输需求，支持多种外部输入设备。

- MIPI Rx最大支持Lane个数：

  MIPI Rx最大支持8个Lane MIPI输入或8个Lane sub-LVDS输入或8个Lane HiSPI输入。
- MIPI Rx能同时对接多个Sensor，最多对接4个Sensor。
- MIPI Rx最大能同时对接不同数量的sensor，每个sensor需要的Lane也不尽相同。因此用户需要确定MIPI Rx 的LANE分布模式。具体的Lane 分布模式请参见下表：

表 11-35 MIPI Rx Lane 分布模式

Mode	DEV0	DEV1	DEV2	DEV3
0	L0~L7	N	N	N
1	L0~L3	N	L4~L7	N
2	L0~L3	N	L4、L6	L5、L7
3	L0、L2	L1、L3	L4、L6	L5、L7

SLVS-EC

SLVS-EC接口支持更高帧率更大分辨率图像的采集，通过SLVS-EC的PHY接收高速串行的数据转化为DC（Digital Camera）时序，通过不同的功能模式配置，SLVS-EC可以支持多种速度和分辨率的数据传输需求，支持多种外部输入设备。

SLVS-EC 最大支持 8Lane SLVS 输入。

Lane 管脚及复用说明

MIPI Rx与SLVS-EC的Lane复用管脚，同一时刻同一个Lane只能被MIPI Rx和SLVS-EC中的一个使用。

具体的Lane 管脚连接请参见下表。

表 11-36 MIPI Rx 与 SLVS-EC 的 Lane 复用关系图

LANE	MIPI0	MIPI1	MIPI2	MIPI3	SLVS0	SLVS1
Lane0	√	-	-	-	√	√

LANE	MIPI0	MIPI1	MIPI2	MIPI3	SLVS0	SLVS1
Lane1	√	√	-	-	√	√
Lane2	√	-	-	-	√	√
Lane3	√	√	-	-	√	√
Lane4	√	-	√	-	√	√
Lane5	√	-	√	√	√	√
Lane6	√	-	√	-	√	√
Lane7	√	-	√	√	√	√

- MIPI0对应VICAP的DEV 0，MIPI1对应VICAP的DEV1，依次类推。
- SLVS0对应VICAP的DEV0，SLVS1对应VICAP的DEV1。

11.18.8.2 基本概念

- MIPI

  MIPI的全称是Mobile Industry Processor Interface(移动行业处理器接口)，本文描述的MIPI接口特指物理层使用D-PHY传输规范，协议层使用CSI-2的通信接口。
- sub-LVDS

  LVDS的全称是Low Voltage differential Signaling(低压差分信号)，通过同步码区分消隐区和有效数据。sub-LVDS是LVDS的低压变种，常见于sony sensor。
- SLVS-EC

  SLVS-EC的全称是Scalable Low Voltage Signaling Embedded Clock，是与MIPI并列的接口，用于高帧率和高分辨率图像采集。
- Lane

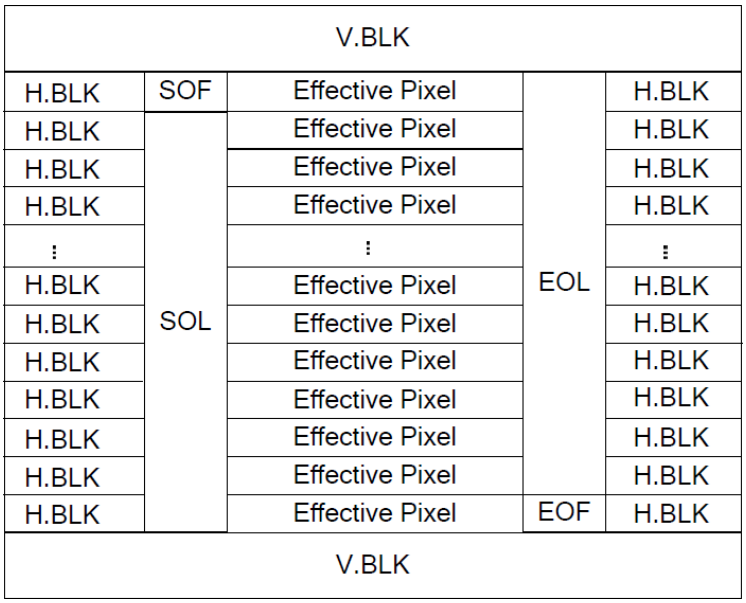
  用于连接发送端和接收端的一对高速差分线，既可以是时钟Lane，也可以是数据Lane。
- Link

  发送端和接收端之间的时钟Lane和至少一个数据Lane组成一个 Link，本文中的link 是一个软件概念，每一个 link 包括两个数据 lane。
- 同步码

  MIPI接口使用CSI-2 里面的短包进行同步，sub-LVDS 使用同步码区分有效数据和消隐区。sub-LVDS 有两种同步方式：

  – 使用 SOF/EOF 表示帧起始和结束，使用SOL/EOL 表示行的起始和结束。同步方式如下图所示。

图 11-9 SOF/EOF/SOL/EOL 同步方式

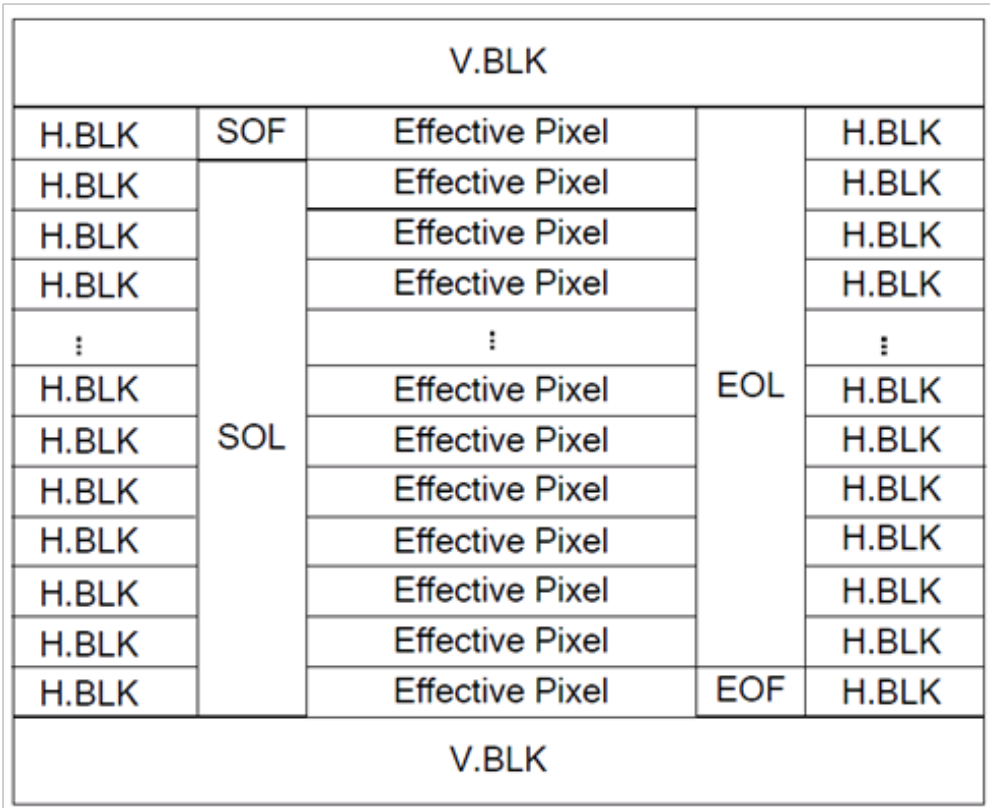


- 使用SAV(invalid) EAV(invalid)表示消隐区的无效数据开始和结束，使用SAV(valid) EAV(valid)表示有效像素数据的开始和结束。

  每个同步码由4个字段组成，每个字段的位宽与像素数据位宽保持一致。前3个字段为固定基准码字，第4个字段由sensor厂家确定。

  由于不同的sensor可能会有不同的同步码，所以需要根据sensor配置同步码。同步方式如下图所示。

图 11-10 SAV/EAV 同步方式



- DOL
  DOL的全称是 Digital Overlap，指SONY的WDR 功能。

11.18.8.3 HI_MIPI_SET_DEV_ATTR

功能描述

设置MIPI Rx、SLVS和并口设备属性。

Atlas 200/300/500 推理产品，不支持该命令字。

Atlas 推理系列产品，不支持该命令字。

定义

```
#define HI_MIPI_SET_DEV_ATTR _IOW(HI_MIPI_IOC_MAGIC, 0x01, combo_dev_attr_t)
```

参数

- HI_MIPI_IOC_MAGIC：MIPI Rx ioctl命令的幻数。
  ```
  #define HI_MIPI_IOC_MAGIC 'm'
  ```
- 0x01：标识命令字功能，用于设置MIPI Rx、SLVS和并口设备属性。
- combo_dev_attr_t：combo设备属性。

返回值

- 0：成功
- -1：失败

注意事项

操作SENSOR复位信号线和时钟信号线会对所连接到该信号线的所有SENSOR都产生效果。

11.18.8.4 HI_MIPI_SET_HS_MODE

功能描述

设置MIPI Rx的Lane分布模式，对SLVS无作用。

Atlas 200/300/500 推理产品，不支持该命令字。

Atlas 推理系列产品，不支持该命令字。

定义

```
#define HI_MIPI_SET_HS_MODE _IOW(HI_MIPI_IOC_MAGIC, 0x0b, lane_divide_mode_t)
```

参数

- HI_MIPI_IOC_MAGIC：MIPI Rx ioctl命令的幻数。
  ```
  #define HI_MIPI_IOC_MAGIC 'm'
  ```
- 0x0b：标识命令字功能，用于设置MIPI Rx的Lane分布模式。

- lane_divide_mode_t：MIPI Rx的Lane分布模式。

返回值

- 0：成功
- -1：失败

注意事项

- 执行本命令字后，系统内部会对MIPI Rx子系统进行复位和解复位动作，因此在调用其它命令字前需先调用本命令字，否则会导致其它命令字的配置被复位。
- 在接多路sensor输入时，建议在初始时根据硬件连接设置Lane的分布模式，在之后的多路sensor采集过程中不能再使用本命令字，否则会导致本命令字之前的其它配置被复位。

11.18.8.5 HI_MIPI_ENABLE_MIPI_CLOCK

功能描述

打开MIPI设备的时钟。

Atlas 200/300/500 推理产品，不支持该命令字。

Atlas 推理系列产品，不支持该命令字。

定义

```
#define HI_MIPI_ENABLE_MIPI_CLOCK _IOW(HI_MIPI_IOC_MAGIC, 0x0c, combo_dev_t)
```

参数

- HI_MIPI_IOC_MAGIC：MIPI Rx ioctl命令的幻数。
  ```
  #define HI_MIPI_IOC_MAGIC 'm'
  ```
- 0x0c：标识命令字功能，用于打开MIPI设备的时钟。
- combo_dev_t：设备号。

返回值

- 0：成功
- -1：失败

11.18.8.6 HI_MIPI_DISABLE_MIPI_CLOCK

功能描述

关闭MIPI设备的时钟。

Atlas 200/300/500 推理产品，不支持该命令字。

Atlas 推理系列产品，不支持该命令字。

定义

```
#define HI_MIPI_DISABLE_MIPI_CLOCK _IOW(HI_MIPI_IOC_MAGIC, 0x0d, combo_dev_t)
```

参数

- HI_MIPI_IOC_MAGIC：MIPI Rx ioctl命令的幻数。
  ```
  #define HI_MIPI_IOC_MAGIC 'm'
  ```
- 0x0d：标识命令字功能，用于关闭MIPI设备的时钟。
- combo_dev_t：设备号。

返回值

- 0：成功
- -1：失败

11.18.8.7 HI_MIPI_RESET_MIPI

功能描述

复位MIPI Rx。

Atlas 200/300/500 推理产品，不支持该命令字。

Atlas 推理系列产品，不支持该命令字。

定义

```
#define HI_MIPI_RESET_MIPI _IOW(HI_MIPI_IOC_MAGIC, 0x07, combo_dev_t)
```

参数

- HI_MIPI_IOC_MAGIC：MIPI Rx ioctl命令的幻数。
  ```
  #define HI_MIPI_IOC_MAGIC 'm'
  ```
- 0x07：标识命令字功能，用于复位MIPI Rx。
- combo_dev_t：设备号。

返回值

- 0：成功
- -1：失败

11.18.8.8 HI_MIPI_UNRESET_MIPI

功能描述

撤销复位MIPI Rx。

Atlas 200/300/500 推理产品，不支持该命令字。

Atlas 推理系列产品，不支持该命令字。

功能定义

```
#define HI_MIPI_UNRESET_MIPI _IOW(HI_MIPI_IOC_MAGIC, 0x08, combo_dev_t)
```

参数

- HI_MIPI_IOC_MAGIC：MIPI Rx ioctl命令的幻数。
  ```
  #define HI_MIPI_IOC_MAGIC 'm'
  ```
- 0x08：标识命令字功能，用于撤销复位MIPI Rx。
- combo_dev_t：设备号。

返回值

- 0：成功
- -1：失败

11.18.8.9 HI_MIPI_ENABLE_SLVS_CLOCK

功能描述

打开SLVS设备的时钟。

Atlas 200/300/500 推理产品，不支持该命令字。

Atlas 推理系列产品，不支持该命令字。

定义

```
#define HI_MIPI_ENABLE_SLVS_CLOCK _IOW(HI_MIPI_IOC_MAGIC, 0x0e, combo_dev_t)
```

参数

- HI_MIPI_IOC_MAGIC：MIPI Rx ioctl命令的幻数。
  ```
  #define HI_MIPI_IOC_MAGIC 'm'
  ```
- 0x0e：标识命令字功能，用于打开SLVS设备的时钟。
- combo_dev_t：设备号。

返回值

- 0：成功
- -1：失败

11.18.8.10 HI_MIPI_DISABLE_SLVS_CLOCK

功能描述

关闭SLVS设备的时钟。

Atlas 200/300/500 推理产品，不支持该命令字。

Atlas 推理系列产品，不支持该命令字。

定义

```
#define HI_MIPI_DISABLE_SLVS_CLOCK _IOW(HI_MIPI_IOC_MAGIC, 0x0f, combo_dev_t)
```

参数

- HI_MIPI_IOC_MAGIC：MIPI Rx ioctl命令的幻数。
  ```
  #define HI_MIPI_IOC_MAGIC 'm'
  ```
- 0x0f：标识命令字功能，用于关闭SLVS设备的时钟。
- combo_dev_t：设备号。

返回值

- 0：成功
- -1：失败

11.18.8.11 HI_MIPI_RESET_SLVS

功能描述

复位SLVS。

Atlas 200/300/500 推理产品，不支持该命令字。

Atlas 推理系列产品，不支持该命令字。

定义

```
#define HI_MIPI_RESET_SLVS _IOW(HI_MIPI_IOC_MAGIC, 0x09, combo_dev_t)
```

参数

- HI_MIPI_IOC_MAGIC：MIPI Rx ioctl命令的幻数。
  ```
  #define HI_MIPI_IOC_MAGIC 'm'
  ```
- 0x09：标识命令字功能，用于复位SLVS。
- combo_dev_t：设备号。

返回值

- 0：成功
- -1：失败

11.18.8.12 HI_MIPI_UNRESET_SLVS

功能描述

撤销复位SLVS。

Atlas 200/300/500 推理产品，不支持该命令字。

Atlas 推理系列产品，不支持该命令字。

定义

```
#define HI_MIPI_UNRESET_SLVS _IOW(HI_MIPI_IOC_MAGIC, 0x0a, combo_dev_t)
```

参数

- HI_MIPI_IOC_MAGIC：MIPI Rx ioctl命令的幻数。
  ```
  #define HI_MIPI_IOC_MAGIC 'm'
  ```
- 0x0a：标识命令字功能，用于撤销复位SLVS。
- combo_dev_t：设备号。

返回值

- 0：成功
- -1：失败

11.18.8.13 HI_MIPI_CONFIG_SENSOR_CLOCK

功能描述

配置Sensor的输入时钟频率。

Atlas 200/300/500 推理产品，不支持该命令字。

Atlas 推理系列产品，不支持该命令字。

定义

```
#define HI_MIPI_CONFIG_SENSOR_CLOCK _IOW(HI_MIPI_IOC_MAGIC, 0x17, sns_clk_cfg_t)
```

参数

- HI_MIPI_IOC_MAGIC：MIPI Rx ioctl命令的幻数。
  ```
  #define HI_MIPI_IOC_MAGIC 'm'
  ```
- 0x17：标识命令字功能，用于配置输入给Sensor的时钟频率。
- sns_clk_cfg_t：Sensor从模式下输入给Sensor的时钟配置信息。
  在Sensor工作在从模式或者需要芯片给Sensor提供时钟时，需要在流程中调用此命令字配置输入给Sensor的时钟配置信息。

返回值

- 0：成功
- -1：失败

注意事项

在配置SENSOR时钟时，必须先使用**HI_MIPI_DISABLE_SENSOR_CLOCK**命令字禁用SENSOR时钟。

11.18.8.14 HI_MIPI_ENABLE_SENSOR_CLOCK

功能描述

打开SENSOR的时钟。

Atlas 200/300/500 推理产品，不支持该命令字。

Atlas 推理系列产品，不支持该命令字。

定义

```
#define HI_MIPI_ENABLE_SENSOR_CLOCK _IOW(HI_MIPI_IOC_MAGIC, 0x10, sns_clk_source_t)
```

参数

- HI_MIPI_IOC_MAGIC：MIPI Rx ioctl命令的幻数。
  ```
  #define HI_MIPI_IOC_MAGIC 'm'
  ```
- 0x10：标识命令字功能，用于打开SENSOR的时钟。
- sns_clk_source_t：SENSOR的时钟源。

返回值

- 0：成功
- -1：失败

11.18.8.15 HI_MIPI_DISABLE_SENSOR_CLOCK

功能描述

关闭SENSOR的时钟。

Atlas 200/300/500 推理产品，不支持该命令字。

Atlas 推理系列产品，不支持该命令字。

定义

```
#define HI_MIPI_DISABLE_SENSOR_CLOCK _IOW(HI_MIPI_IOC_MAGIC, 0x11, sns_clk_source_t)
```

参数

- HI_MIPI_IOC_MAGIC：MIPI Rx ioctl命令的幻数。
  ```
  #define HI_MIPI_IOC_MAGIC 'm'
  ```
- 0x11：标识命令字功能，用于关闭SENSOR的时钟。
- sns_clk_source_t：SENSOR的时钟源。

返回值

- 0：成功
- -1：失败

11.18.8.16 HI_MIPI_RESET_SENSOR

功能描述

复位Sensor。

Atlas 200/300/500 推理产品，不支持该命令字。

Atlas 推理系列产品，不支持该命令字。

定义

```
#define HI_MIPI_RESET_SENSOR _IOW(HI_MIPI_IOC_MAGIC, 0x05, sns_rst_source_t)
```

参数

- HI_MIPI_IOC_MAGIC：MIPI Rx ioctl命令的幻数。
  ```
  #define HI_MIPI_IOC_MAGIC 'm'
  ```
- 0x05：标识命令字功能，用于复位Sensor。
- sns_rst_source_t：Sensor复位信号线编号。

返回值

- 0：成功
- -1：失败

注意事项

部分Sensor对复位时间有要求，可以在撤销复位前增加延时操作。

11.18.8.17 HI_MIPI_UNRESET_SENSOR

功能描述

撤销复位Sensor。

Atlas 200/300/500 推理产品，不支持该命令字。

Atlas 推理系列产品，不支持该命令字。

定义

```
#define HI_MIPI_UNRESET_SENSOR _IOW(HI_MIPI_IOC_MAGIC, 0x06, sns_rst_source_t)
```

参数

- HI_MIPI_IOC_MAGIC：MIPI Rx ioctl命令的幻数。
  ```
  #define HI_MIPI_IOC_MAGIC 'm'
  ```
- 0x06：标识命令字功能，用于复位Sensor。
- sns_rst_source_t：Sensor复位信号线编号。

返回值

- 0：成功
- -1：失败

11.18.9 VI 视频输入功能

11.18.9.1 功能说明

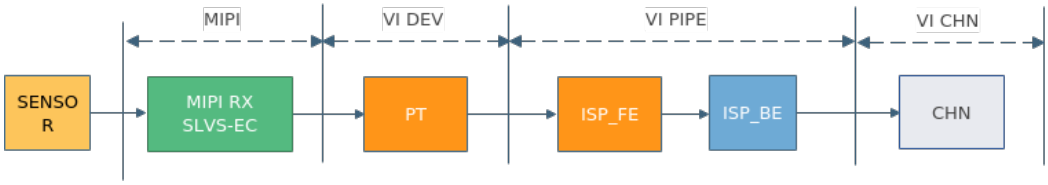
视频输入（VI）模块实现的功能：通过MIPI Rx(含MIPI CSI接口、LVDS接口和HISPI接口)，SLVS-EC等接口接收视频数据。VI将接收到的数据存入到指定的内存区域，在此

过程中，VI可以对接收到的原始视频图像数据进行裁剪、防抖、颜色优化、亮度优化、噪声去除等处理，并输出YUV或RAW格式的图像数据，实现视频数据的获取。

说明

当前版本仅配套支持MIPI CSI接口以及SLVS-EC接口。

图 11-11 VI 功能框图



VI从功能上划分了VI DEV（输入设备）、VI PIPE（图示为物理PIPE，虚拟PIPE只包含ISP_BE）、VI CHN（图示为物理通道）三个层级。

11.18.9.2 约束说明

- 最大并发4路接入，总体性能规格为4096W*2160H@45fps。
- 单路视频输入最大分辨率为16384*16384，最小分辨率为120*120。在开启不同功能后，分辨率规格会有额外约束，具体如下：
  - 开启视频防抖后，最小分辨率为1280*720，最大分辨率为4096*2160。
  - 开启畸变矫正后，最小分辨率为640*480，最大分辨率为8192*8192。
  - 开启RAW图压缩功能后，最大分辨率规格为4096*4096。
  - 开启RAW图帧压缩功能后，最小分辨率为640*480。
  - 当sensor模组直出YUV图像时，仅pipe2和pipe3支持YUV422 8bit位宽图像接入，宽度取值范围[560, 3840]，高度取值范围[300, 2160]。
- 所有输入和输出分辨率，宽度必须以4像素对齐，高度必须4像素对齐。
- 支持WDR，最多支持2帧WDR，支持多种WDR时序。
- 支持处理的数据位宽为：8bit、10bit、12bit、14bit。
- VI的接口，除pipe、chn取流相关的接口（hi_mpi_vi_get_pipe_frame、hi_mpi_vi_release_pipe_frame、hi_mpi_vi_pipe_get_buffer、hi_mpi_vi_pipe_release_buffer、hi_mpi_vi_get_chn_frame、hi_mpi_vi_release_chn_frame）外，其他接口都不支持多线程的并发调用；所有接口，都不支持多进程调用。

11.18.9.3 基本概念

- VI设备/PIPE/通道
  - 视频输入设备VI DEV

    视频输入设备支持若干种时序输入，负责对时序进行解析。
  - 视频输入处理VI PIPE
    - 视频输入处理物理PIPE

      视频输入处理PIPE绑定在视频输入设备后端，负责设备解析后的图像算法处理的ISP FE和ISP BE部分，对图像数据进行流水线处理，输出YUV图像格式给通道。

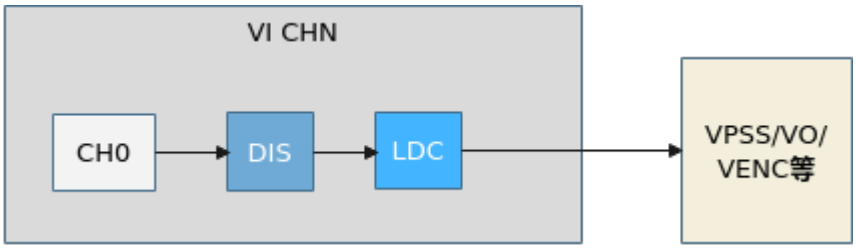
- 视频输入处理虚拟PIPE

  视频输入处理虚拟PIPE不绑定设备，负责其他模块或用户发送过来的数据再处理，不支持PIPE中的ISP_FE，直接从PIPE中的ISP_BE开始。

– 视频通道VI CHN

- 视频物理通道PHY_CHN

  物理通道负责通过不同的硬件加速单元，实现DIS、LDC等算法处理，并将最终处理后的数据输出到内存中。通道功能框图如下：



说明

开启DIS、LDC等功能后，需要在调用接口**hi_mpi_vi_set_chn_attr**时，每个功能额外分配1块VB资源，即chn属性的depth要额外+1。

- 视频扩展通道EXT_CHN

  当前版本不支持。

● 绑定关系

为了将前端的Camera模组视频流通过MIPI接口接入到VI中处理，在芯片板级连线上，首先要满足DEV设备和MIPI 设备之间的绑定约束关系，其次为了软件侧的灵活调度编排，用户可自行配置调整DEV设备和PIPE之间的绑定关系。

– DEV和MIPI设备的绑定关系

  VI DEV与MIPI RX/SLVS-EC设备的绑定关系固定，不可配置。对应关系如下：

表 11-37 VI DEV 与 MIPI RX/SLVS-EC 接口的绑定关系

VI DEV	MIPI RX	SLVS-EC
0	0	0
1	1	1
2	2	X
3	3	X

– DEV和PIPE的绑定关系

- 每个PIPE都可以与任意Dev通过接口**hi_mpi_vi_set_dev_bind_pipe**设置绑定关系，且必须在调用接口**hi_mpi_vi_create_pipe**创建PIPE前绑定，不能动态修改绑定关系。

- 线性模式（非WDR模式）下只需要将一个物理PIPE绑定一个Dev；WDR模式下需要多个物理PIPE绑定同一个Dev，且只有绑定到Dev的第1条物理PIPE的通道有数据输出，其他绑定的物理PIPE的通道没有图像数据输

出。例如2合1WDR模式，我们需要将PIPE0、PIPE1绑定到同一个Dev0设备上，只有PIPE0上的通道有数据输出，PIPE1上的通道无数据输出。

- 镜头畸变校正（LDC）

  镜头畸变校正，一些低端镜头容易产生图像畸变，需要根据畸变程度对其图像进行校正。

- 视频防抖（DIS）

  DIS模块通过比较当前图像与前两帧图像采用不同自由度的防抖算法计算出当前图像在各个轴方向上的抖动偏移向量，然后根据抖动偏移向量对当前图像进行校正，从而起到防抖的效果。

- 低延时

  低延时指图像写出指定的行数到DDR后，VI上报一个中断，把图像发给后端模块处理，可以减少延时，且硬件会有机制保证图像是先写后读，不会出现读图像错误。ISP_BE后接3DNR、DIS、LDC支持低延时功能。

- 从模式

  部分SENSOR需要工作在从模式，此时需要由外部提供行场曝光同步信息，需要使用VI的从模式模块。用户需要根据SENSOR管脚的连线和下表确定使用哪个从模式模块，然后选择对应的物理PIPE号创建物理PIPE，否则会没有数据。

表 11-38 从模式与 PIPE 的对应关系表

从模式编号	可连接Sensor数目	管脚名称	建议使用PIPE的编号
0	2	SENSOR_VS0 SENSOR_HS0	0、1
1	2	SENSOR_VS1 SENSOR_HS1	2、3

- VI功能涉及的缩略语如下：

表 11-39 缩略语列表

缩略语	全称	含义
VI	Video Input	视频输入
WDR	Wide Dynamic Range	宽动态
BE	Backend	ISP pipeline中FPN及之后的部分
FE	Frontend	ISP pipeline中FPN算法之前的部分
DIS	Digital Image Stabilization	数字防抖
GME	Global Motion Estimation	全局运动估计

缩略语	全称	含义
GYRO	Gyroscope	陀螺仪
LDC	Lens Distortion Correction	镜头畸变矫正
MIPI	Mobile Industry Processor Interface	移动行业处理器界面
LVDS	Low-Voltage Differential Signal	低压差分信号
HISPI	High-Speed Serial Pixel Interface	高速串行像素接口
SLVS-EC	Scalable Low-Voltage Signaling interface with an Embedded Clock	可伸缩的、含嵌入式时钟的低电压信号接口
IPC	IP camera	网络摄像机
DV	Digital Visual	数字摄像
PMF	Perspective Mapping Function	投影变换映射函数
MPP	Media Processing Platform	媒体处理平台
ISP	Image Signal Processor	图像信号处理器
Gyro DIS	Gyroscope Digital Image Stabilization	陀螺仪数字防抖

11.18.9.4 hi_mpi_vi_set_dev_bind_pipe

函数功能

一对多设置VI设备与物理PIPE的绑定关系。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 必须在调用**hi_mpi_vi_enable_dev**接口使用VI设备后、调用**hi_mpi_vi_create_pipe**接口创建PIPE之前才能调用本接口绑定物理PIPE。
- 不支持绑定已经创建的PIPE。
- 建议将带宽高的VI设备绑定到pipe0或pipe1上。

函数原型

hi_s32 hi_mpi_vi_set_dev_bind_pipe(hi_vi_dev vi_dev, const hi_vi_dev_bind_pipe *dev_bind_pipe)

参数说明

参数名	输入/输出	说明
vi_dev	输入	VI设备号。 取值范围：[0, 4)。
dev_bind_pipe	输入	VI物理PIPE号。 取值范围：[0, 4)。 WDR模式下，设置绑定关系时，必须pipe0和pipe1一组，pipe2和pipe3一组，即将hi_vi_dev_bind_pipe结构体内的num参数设置为2，将hi_vi_dev_bind_pipe结构体内的pipe_id数组内的元素设置为0、1两个值，或者2、3两个值。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.4 VI视频输入返回码**。

11.18.9.5 hi_mpi_vi_get_dev_bind_pipe

函数功能

获取VI设备与物理PIPE的绑定关系。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vi_get_dev_bind_pipe(hi_vi_dev vi_dev, hi_vi_dev_bind_pipe *dev_bind_pipe)

参数说明

参数名	输入/输出	说明
vi_dev	输入	VI设备号。 取值范围：[0, 4)。

参数名	输入/输出	说明
dev_bind_pipe	输出	VI 物理PIPE号。 取值范围：[0, 4)。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.6 hi_mpi_vi_set_dev_attr

函数功能

设置VI设备属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

在调用前要保证VI设备处于禁用状态。如果VI设备已处于开启状态，可以使用hi_mpi_vi_disable_dev来禁用设备。

函数原型

hi_s32 hi_mpi_vi_set_dev_attr(hi_vi_dev vi_dev, const hi_vi_dev_attr *dev_attr)

参数说明

参数名	输入/输出	说明
vi_dev	输入	VI设备号。 取值范围：[0, 4)。
dev_attr	输入	VI设备属性指针。静态属性。 参数dev_attr主要用来配置指定VI设备的视频接口模式、数据类型、图像宽高等属性，用于与外围sensor对接。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.7 hi_mpi_vi_get_dev_attr

函数功能

获取VI设备属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

如果未设置VI设备属性，该接口将返回失败。

函数原型

hi_s32 hi_mpi_vi_get_dev_attr(hi_vi_dev vi_dev, hi_vi_dev_attr *dev_attr)

参数说明

参数名	输入/输出	说明
vi_dev	输入	VI设备号。 取值范围：[0, 4)。
dev_attr	输出	VI设备属性指针。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.8 hi_mpi_vi_enable_dev

函数功能

启用VI设备。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 启用VI设备前，必须先调用hi_mpi_vi_set_dev_attr接口设置设备属性，否则返回失败。
- 可重复启用VI设备，不返回失败。

函数原型

hi_s32 hi_mpi_vi_enable_dev(hi_vi_dev vi_dev)

参数说明

参数名	输入/输出	说明
vi_dev	输入	VI设备号。 取值范围：[0, 4)。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.4 VI视频输入返回码**。

11.18.9.9 hi_mpi_vi_disable_dev

函数功能

禁用VI设备。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 需先调用**hi_mpi_vi_destroy_pipe**接口销毁所有与该VI设备绑定的物理PIPE后，再禁用VI设备。
- 可重复禁用，不返回失败。
- 禁用VI设备后，VI设备处理完全关闭的状态，在下次调用**hi_mpi_vi_enable_dev**接口启用VI设备前，需要重新调用**hi_mpi_vi_set_dev_attr**接口设置该设备的属性。

函数原型

hi_s32 hi_mpi_vi_disable_dev(hi_vi_dev vi_dev)

参数说明

参数名	输入/输出	说明
vi_dev	输入	VI设备号。 取值范围：[0, 4)。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.4 VI视频输入返回码**。

11.18.9.10 hi_mpi_vi_create_pipe

函数功能

创建一个VI PIPE。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 物理PIPE属性中的size、pixel_format、bit_width等必须与VI设备的属性保持一致，否则会出现错误。
- 同一个PIPE号，不支持重复创建。
- 不管输入图像宽度多大，只要在PIPE所支持宽度规格内，都可支持数据压缩；仅pipe0和pipe1支持压缩；当输入图像大于4096时，不支持压缩。
- WDR模式下，需要创建多个物理PIPE绑定到同一个开了WDR的设备上，当进行切换时，需要把所有绑定到该设备的物理PIPE销毁再重建。不能使用上次用过但未销毁的物理PIPE，否则可能造成错误。

函数原型

hi_s32 hi_mpi_vi_create_pipe(hi_vi_pipe vi_pipe, hi_vi_pipe_attr *pipe_attr)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。
pipe_attr	输入	PIPE的属性结构体指针。 静态属性。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.4 VI视频输入返回码**。

11.18.9.11 hi_mpi_vi_destroy_pipe

函数功能

销毁一个VI PIPE。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 使用本接口前，需先调用hi_mpi_vi_stop_pipe停止PIPE。
- 在未创建PIPE或重复销毁PIPE时，调用本接口，将提示PIPE不存在。

函数原型

hi_s32 hi_mpi_vi_destroy_pipe(hi_vi_pipe vi_pipe)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.12 hi_mpi_vi_set_vi_vpss_mode

函数功能

设置VI模块、VPSS模块的工作模式 ，包括在线模式、离线模式。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

必须在调用hi_mpi_sys_init接口后、创建VI PIPE和VPSS组前调用本接口设置工作模式。各模式的说明如下表所示。

表 11-40 VI 和 VPSS 的工作模式说明

模式	VI工作模式（VI内部FE和BE之间）	VPSS工作模式（VI与VPSS之间）
在线模式	VI内部FE处理完成后，视频流不写出到DDR，直接送给BE继续处理。 VI在线模式下，并发规格降低为1路。**当前版本不支持。**	VI处理完成后，视频流不写出到DDR，直接送给VPSS继续处理。该模式下，VI CHN不会产生图像数据，最终图像数据需要从VPSS CHN中获取。

模式	VI工作模式（VI内部FE和BE之间）	VPSS工作模式（VI与VPSS之间）
离线模式	VI内部FE处理完成后，视频流输出写出到DDR，BE从FE写出的DDR中读取图像继续处理。	VI完成后，视频流输出写出到DDR，并可从VI CHN获取图像数据。 离线模式下，用户也可通过hi_mpi_sys_bind接口设置VI和VPSS的绑定关系，此时被绑定的VPSS会自动从VI写出的DDR中读取图像继续处理。

函数原型

hi_s32 hi_mpi_vi_set_vi_vpss_mode(const hi_vi_vpss_mode *vi_vpss_mode)

参数说明

参数名	输入/输出	说明
vi_vpss_mode	输入	VI、VPSS工作模式。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.13 hi_mpi_vi_get_vi_vpss_mode

函数功能

获取VI、VPSS当前的工作模式 。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vi_get_vi_vpss_mode(hi_vi_vpss_mode *vi_vpss_mode)

参数说明

参数名	输入/输出	说明
vi_vpss_mode	输出	VI、VPSS工作模式。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.14 hi_mpi_vi_set_pipe_attr

函数功能

设置VI PIPE属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 必须先调用hi_mpi_vi_create_pipe接口，创建VI PIPE后，才能调用该接口。

函数原型

hi_s32 hi_mpi_vi_set_pipe_attr(hi_vi_pipe vi_pipe, const hi_vi_pipe_attr *pipe_attr)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。
pipe_attr	输入	PIPE的属性结构体指针。 静态属性。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.15 hi_mpi_vi_get_pipe_attr

函数功能

获取VI PIPE的属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 必须先调用hi_mpi_vi_create_pipe接口，创建VI PIPE后，才能调用该接口。
- PIPE属性必须合法，其中部分静态属性不可动态设置，具体请参见hi_vi_pipe_attr。

函数原型

hi_s32 hi_mpi_vi_get_pipe_attr(hi_vi_pipe vi_pipe, hi_vi_pipe_attr *pipe_attr)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。
pipe_attr	输入	PIPE的属性结构体指针。 静态属性。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.16 hi_mpi_vi_start_pipe

函数功能

启用VI PIPE。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用hi_mpi_vi_create_pipe接口创建PIPE。
- 支持重复调用本接口操作同一个PIPE。

函数原型

hi_s32 hi_mpi_vi_start_pipe(hi_vi_pipe vi_pipe)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.17 hi_mpi_vi_stop_pipe

函数功能

禁用VI PIPE。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用hi_mpi_vi_create_pipe接口创建PIPE。
- 支持重复调用该接口操作同一个PIPE。

函数原型

hi_s32 hi_mpi_vi_stop_pipe(hi_vi_pipe vi_pipe)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.18 hi_mpi_vi_set_pipe_pre_crop

函数功能

设置VI物理PIPE输入端的裁剪功能属性，在VI DEV输出给VI PIPE处理前进行裁剪。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用hi_mpi_vi_create_pipe接口创建PIPE。
- 需在调用hi_mpi_vi_start_pipe接口启动PIPE前调用本接口 ，启动PIPE后 ，不可调用本接口 。

函数原型

hi_s32 hi_mpi_vi_set_pipe_pre_crop(hi_vi_pipe vi_pipe, const hi_crop_info *crop_info)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。
crop_info	输入	裁剪区域属性。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.19 hi_mpi_vi_get_pipe_pre_crop

函数功能

获取VI 物理PIPE输入端的裁剪功能属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用hi_mpi_vi_create_pipe接口创建PIPE。
- 不支持虚拟PIPE。

函数原型

hi_s32 hi_mpi_vi_get_pipe_pre_crop(hi_vi_pipe vi_pipe, hi_crop_info *crop_info)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。

参数名	输入/输出	说明
crop_info	输出	裁剪区域属性。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.20 hi_mpi_vi_set_pipe_pre_be_crop

函数功能

设置VI PIPE内部BE输入端的裁剪功能属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用hi_mpi_vi_create_pipe接口创建PIPE。
- 需在调用hi_mpi_vi_start_pipe接口启动PIPE前调用本接口 ，启动PIPE后 ，不可调用本接口 。
- 仅支持物理pipe，不支持虚拟pipe。

函数原型

hi_s32 hi_mpi_vi_set_pipe_pre_be_crop(hi_vi_pipe vi_pipe, const hi_crop_info *crop_info)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 4)。
crop_info	输入	裁剪区域属性。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.21 hi_mpi_vi_set_pipe_bind_strobe

函数功能

设置VI PIPE与对外strobe输入信号的绑定关系。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用**hi_mpi_vi_create_pipe**接口创建PIPE。
- 必须在调用**hi_mpi_vi_start_pipe**启动pipe前调用本接口。

函数原型

hi_s32 hi_mpi_vi_set_pipe_bind_strobe(hi_vi_pipe vi_pipe, hi_u32 strobe_id)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	物理PIPE号。 取值范围：[0, 4)。
strobe_id	输入	strobe管脚编号。 取值范围：[0, 3)。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.4 VI视频输入返回码**。

11.18.9.22 hi_mpi_vi_set_pipe_frame_dump_attr

函数功能

设置VI 物理PIPE dump属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用**hi_mpi_vi_create_pipe**接口创建PIPE。
- 需要dump不同的像素格式数据，可以在dump前调用**hi_mpi_vi_create_pipe**接口创建pipe时指定像素格式，像素格式需要与sensor模组实际输出的格式对齐。
- 需要dump裁剪后的图像，可以在dump前调用**hi_mpi_vi_set_pipe_pre_crop**接口设置pipe的裁剪。

- 需要dump压缩或者非压图像，可以在dump前调用hi_mpi_vi_create_pipe接口创建pipe时指定压缩类型。
- 不支持虚拟PIPE。
- 当pipe已经通过hi_mpi_vi_start_pipe接口启动后再调用当前接口，会清除内部缓存的用户未通过hi_mpi_vi_get_pipe_frame接口取走的图像。

函数原型

hi_s32 hi_mpi_vi_set_pipe_frame_dump_attr(hi_vi_pipe vi_pipe, const hi_vi_frame_dump_attr *dump_attr)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	物理PIPE号。 取值范围：[0, 4)。
dump_attr	输入	VI 物理PIPE dump的属性。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.23 hi_mpi_vi_get_pipe_frame_dump_attr

函数功能

获取VI物理PIPE dump属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用hi_mpi_vi_create_pipe接口创建PIPE。
- 不支持虚拟PIPE不支持。

函数原型

hi_s32 hi_mpi_vi_get_pipe_frame_dump_attr(hi_vi_pipe vi_pipe, hi_vi_frame_dump_attr *dump_attr)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	物理PIPE号。 取值范围：[0, 4)。
dump_attr	输出	VI 物理PIPE dump的属性。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.4 VI视频输入返回码**。

11.18.9.24 hi_mpi_vi_get_pipe_frame

函数功能

获取VI物理PIPE的数据。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用**hi_mpi_vi_create_pipe**接口创建PIPE。
- 调用**hi_mpi_vi_set_pipe_frame_dump_attr**接口开启dump、设置depth， 否则内部不会预留临时的内存空间，最终导致获取不到图像数据。
- 不支持虚拟PIPE。

函数原型

hi_s32 hi_mpi_vi_get_pipe_frame(hi_vi_pipe vi_pipe, hi_video_frame_info *frame_info, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	物理PIPE号。 取值范围：[0, 4)
frame_info	输出	VI PIPE 数据信息的指针。
milli_sec	输入	超时参数 ，取值范围： ● -1：表示阻塞模式； ● 0：表示非阻塞模式； ● >0： 配置具体的超时时间，单位为毫秒（ms）。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.25 hi_mpi_vi_release_pipe_frame

函数功能

释放VI PIPE的数据。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 本接口需与 hi_mpi_vi_get_pipe_frame接口配对使用，接口调用次数保持一致，否则返回报错。
- 用户需先调用hi_mpi_vi_get_pipe_frame接口获取frame_info，再将获取到的frame_info传入本接口，否则可能造成释放不成功。

函数原型

hi_s32 hi_mpi_vi_release_pipe_frame(hi_vi_pipe vi_pipe, const hi_video_frame_info *frame_info)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	物理PIPE号。 取值范围：[0, 4)
frame_info	输入	VI PIPE数据信息的指针。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.26 hi_mpi_vi_set_pipe_frame_source

函数功能

设置VI PIPE数据的来源。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用hi_mpi_vi_create_pipe接口创建PIPE。

函数原型

hi_s32 hi_mpi_vi_set_pipe_frame_source(hi_vi_pipe vi_pipe, const hi_vi_pipe_frame_source source)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE 号。 取值范围：[0, 12)。
source	输入	PIPE 的数据来源。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.27 hi_mpi_vi_get_pipe_frame_source

函数功能

获取VI PIPE数据的来源。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用hi_mpi_vi_create_pipe接口创建PIPE。

函数原型

hi_s32 hi_mpi_vi_get_pipe_frame_source(hi_vi_pipe vi_pipe, hi_vi_pipe_frame_source *source)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE 号。 取值范围：[0, 12)。

参数名	输入/输出	说明
source	输出	PIPE 的数据来源。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.28 hi_mpi_vi_send_pipe_yuv

函数功能

通过VI PIPE发送YUV数据。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用hi_mpi_vi_start_pipe接口启动PIPE。
- 发送YUV数据前，需要先调用hi_mpi_vi_set_pipe_frame_source接口将PIPE的数据来源设置为HI_VI_PIPE_FRAME_SOURCE_USER，此时来自SENSOR的数据不再会送给ISP BE处理，ISP BE只会处理用户送下来的帧数据。
- 通过本接口发送YUV数据时，对YUV数据有如下要求：
  - YUV数据的宽高必须与hi_mpi_vi_create_pipe接口创建PIPE时设置的size宽高保持一致。
  - 送下来的YUV帧信息必须为真实有效的通过hi_mpi_vi_pipe_get_buffer接口获取的帧信息。
  - YUV数据像素格式必须与hi_mpi_vi_create_pipe接口创建PIPE时设置的像素格式保持一致。
  - YUV数据field必须为HI_VIDEO_FIELD_FRAME。
  - YUV数据视频格式video_format必须为HI_VIDEO_FORMAT_LINEAR。
  - YUV数据压缩模式compress_mode必须为不压缩HI_COMPRESS_MODE_NONE。
  - YUV数据动态范围dynamic_range必须为HI_DYNAMIC_RANGE_SDR8、HI_DYNAMIC_RANGE_SDR10、HI_DYNAMIC_RANGE_XDR。
- WDR模式下，不支持调用本接口。
- 只能从BE送，不支持从FE送。

函数原型

hi_s32 hi_mpi_vi_send_pipe_yuv(hi_vi_pipe vi_pipe, const hi_video_frame_info *frame_info, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE 号。 取值范围：[0, 12)。
frame_info	输入	VI 帧信息结构指针。
milli_sec	输入	超时参数 ，取值范围： ● -1：表示阻塞模式； ● 0：表示非阻塞模式； ● >0： 配置具体的超时时间，单位为毫秒（ms）。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.4 VI视频输入返回码**。

11.18.9.29 hi_mpi_vi_send_pipe_raw

函数功能

通过VI PIPE发送RAW数据。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先依次调用**hi_mpi_vi_create_pipe**接口创建PIPE、调用**hi_mpi_vi_start_pipe**接口启动PIPE，在调用**hi_mpi_vi_create_pipe**接口创建PIPE时，PIPE属性的isp_bypass必须为HI_FALSE。
- 发送RAW数据前，需要先调用**hi_mpi_vi_set_pipe_frame_source**接口将PIPE的数据来源设置为HI_VI_PIPE_FRAME_SOURCE_USER，此时来自SENSOR的数据不再会送给ISP BE处理，ISP BE只会处理用户送下来的帧数据。
- WDR模式下，每个PIPE的RAW在frame_info存放的顺序必须与**hi_mpi_vi_set_dev_bind_pipe**接口绑定PIPE时的顺序保持一致，目的是保证长短曝光帧的顺序正确，每个PIPE的RAW的属性(宽高，像素格式，压缩等等)要与**hi_mpi_vi_create_pipe**接口创建时保持一致。
- 通过本接口发送RAW数据时，对RAW数据有如下要求：
  - 送下来的RAW帧信息必须为真实有效的通过**hi_mpi_vi_pipe_get_buffer**接口获取的帧信息。
  - RAW数据像素格式及图像大小必须与**hi_mpi_vi_create_pipe**接口创建PIPE时设置的像素格式及图像大小保持一致。
  - RAW数据field必须为HI_VIDEO_FIELD_FRAME。

- RAW数据video_format必须为HI_VIDEO_FORMAT_LINEAR。
- RAW数据compress_mode仅支持HI_COMPRESS_MODE_NONE、HI_COMPRESS_MODE_LINE、HI_COMPRESS_MODE_FRAME。
- RAW数据动态范围dynamic_range必须为HI_DYNAMIC_RANGE_SDR8。

函数原型

hi_s32 hi_mpi_vi_send_pipe_raw(hi_u32 pipe_num, hi_vi_pipe pipe_id[], const hi_video_frame_info *frame_info[], hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
pipe_num	输入	PIPE数量。 当工作模式为线性模式时，该参数取值为 1；当工作模式为WDR模式时，该参数取值需要与WDR的PIPE数相同。具体模式可通过hi_mpi_vi_set_dev_attr接口设置WDR属性配置。
pipe_id	输入	PIPE号数组，数组内的元素个数与pipe_num参数值保证一致。
frame_info	输入	RAW数据信息。
milli_sec	输入	超时参数 ，取值范围： ● -1：表示阻塞模式； ● 0：表示非阻塞模式； ● >0： 配置具体的超时时间，单位为毫秒（ms）。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.30 hi_mpi_vi_pipe_get_buffer

函数功能

获取内部空闲的内存块，用于用户指定输入数据。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用hi_mpi_vi_create_pipe接口创建PIPE。

函数原型

hi_s32 hi_mpi_vi_pipe_get_buffer(hi_vi_pipe vi_pipe, hi_video_frame_info *frame_info)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE 号。 取值范围：[0, 12)。
frame_info	输出	视频信息结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.31 hi_mpi_vi_pipe_release_buffer

函数功能

释放通过hi_mpi_vi_pipe_get_buffer接口获取的视频帧相关资源。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vi_pipe_release_buffer(hi_vi_pipe vi_pipe, const hi_video_frame_info *frame_info)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE 号。 取值范围：[0, 12)。
frame_info	输入	视频帧信息。

返回值说明

- 0：成功

- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.32 hi_mpi_vi_set_pipe_vc_number

函数功能

设置VI物理PIPE对接前端sensor的VC号。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 必须调用hi_mpi_vi_create_pipe接口创建PIPE后、调用hi_mpi_vi_start_pipe接口前调用本接口 。
- 有部分sensor在2合1的WDR场景下输出的VC号为1、2等，如果不单独通过本接口指定VC号，则会与系统内部默认的VC号0、1不匹配，导致获取数据失败。
- 不支持虚拟PIPE。

函数原型

hi_s32 hi_mpi_vi_set_pipe_vc_number(hi_vi_pipe vi_pipe, hi_u32 vc_number)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	物理PIPE号。 取值范围：[0, 4)
vc_number	输入	VC号。 取值范围：[0, 4)

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.33 hi_mpi_vi_get_pipe_vc_number

函数功能

获取VI物理PIPE对接前端sensor的VC号。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 必须在PIPE创建后，开启PIPE之前调用 。
- 虚拟PIPE不支持。

函数原型

hi_s32 hi_mpi_vi_get_pipe_vc_number(hi_vi_pipe vi_pipe, hi_u32 *vc_number)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	物理PIPE号。 取值范围：[0, 4)
vc_number	输出	VC号。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.4 VI视频输入返回码**。

11.18.9.34 hi_mpi_vi_get_pipe_fd

函数功能

获取VI PIPE对应的文件句柄，可用于通过select或epoll方式等待可用的PIPE输出结果。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vi_get_pipe_fd(hi_vi_pipe vi_pipe)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。

返回值说明

当大于或等于0时，接口调用成功，返回对应句柄，否则表示接口调用失败。

11.18.9.35 hi_mpi_vi_set_chn_attr

函数功能

设置VI通道属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用hi_mpi_vi_create_pipe接口创建PIPE。

函数原型

hi_s32 hi_mpi_vi_set_chn_attr(hi_vi_pipe vi_pipe, hi_vi_chn vi_chn, const hi_vi_chn_attr *chn_attr)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。
vi_chn	输入	VI通道号。 取值范围：[0, 1)
chn_attr	输入	VI通道属性结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.36 hi_mpi_vi_get_chn_attr

函数功能

获取VI通道属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用hi_mpi_vi_create_pipe接口创建PIPE。
- 必须先调用hi_mpi_vi_set_chn_attr接口设置通道属性。

函数原型

hi_s32 hi_mpi_vi_get_chn_attr(hi_vi_pipe vi_pipe, hi_vi_chn vi_chn, hi_vi_chn_attr *chn_attr)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。
vi_chn	输入	VI通道号。 取值范围：[0, 1)
chn_attr	输出	VI通道属性结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.4 VI视频输入返回码**。

11.18.9.37 hi_mpi_vi_enable_chn

函数功能

启用VI通道。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用**hi_mpi_vi_create_pipe**接口创建PIPE。
- 必须先调用**hi_mpi_vi_set_chn_attr**接口设置通道属性。
- 支持重复启用VI通道，不返回失败。

函数原型

hi_s32 hi_mpi_vi_enable_chn(hi_vi_pipe vi_pipe, hi_vi_chn vi_chn)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。

参数名	输入/输出	说明
vi_chn	输入	VI通道号。 取值范围：[0, 1)

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.38 hi_mpi_vi_disable_chn

函数功能

禁用VI通道。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用hi_mpi_vi_create_pipe接口创建PIPE。
- 支持重复禁用VI通道，不返回失败。

函数原型

hi_s32 hi_mpi_vi_disable_chn(hi_vi_pipe vi_pipe, hi_vi_chn vi_chn)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。
vi_chn	输入	VI通道号。 取值范围：[0, 1)

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.39 hi_mpi_vi_set_chn_dis_config

函数功能

设置VI通道的DIS视频防抖功能配置信息。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用hi_mpi_vi_create_pipe接口创建PIPE。
- 调用本接口将hi_dis_config.scale设置为HI_TRUE且调用hi_mpi_vi_set_chn_dis_attr接口开启DIS功能时，DIS配置信息中的crop_ratio参数可以在DIS功能保持开启的情况下动态修改；其它情况要修改DIS配置信息参数时，必须先调用hi_mpi_vi_set_chn_dis_attr接口关闭DIS功能再修改参数，不可动态修改。

函数原型

hi_s32 hi_mpi_vi_set_chn_dis_config(hi_vi_pipe vi_pipe, hi_vi_chn vi_chn, const hi_dis_config *dis_config)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。
vi_chn	输入	VI通道号。 取值范围：[0, 1)
dis_config	输入	DIS配置信息结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.40 hi_mpi_vi_get_chn_dis_config

函数功能

获取VI通道的DIS配置信息。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用hi_mpi_vi_create_pipe接口创建PIPE。

函数原型

hi_s32 hi_mpi_vi_get_chn_dis_config(hi_vi_pipe vi_pipe, hi_vi_chn vi_chn, hi_dis_config *dis_config)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。
vi_chn	输入	VI通道号。 取值范围：[0, 1)
dis_config	输出	DIS配置信息结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.41 hi_mpi_vi_set_chn_dis_attr

函数功能

设置VI通道的DIS属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用hi_mpi_vi_create_pipe接口创建PIPE。
- 必须先调用hi_mpi_vi_enable_chn接口开启VI通道、调用hi_mpi_vi_set_chn_dis_config接口设置DIS配置后，才能设置DIS属性。
- 调用本接口开启DIS功能，则不能调用hi_mpi_vi_set_pipe_pre_crop接口开启通道裁剪功能，如果要裁剪后的图像做DIS，建议在前端MIPI或者PIPE前端做裁剪。
- 修改DIS配置或修改通道属性，必须先关闭DIS。

函数原型

hi_s32 hi_mpi_vi_set_chn_dis_attr(hi_vi_pipe vi_pipe, hi_vi_chn vi_chn, const hi_dis_attr *dis_attr)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。
vi_chn	输入	VI通道号。 取值范围：[0, 1)
dis_attr	输入	DIS属性结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.42 hi_mpi_vi_get_chn_dis_attr

函数功能

获取VI通道的DIS属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用hi_mpi_vi_create_pipe接口创建PIPE。
- 建议先调用hi_mpi_vi_set_chn_dis_attr接口设置DIS属性后，再使用该接口。

函数原型

hi_s32 hi_mpi_vi_get_chn_dis_attr(hi_vi_pipe vi_pipe, hi_vi_chn vi_chn, hi_dis_attr *dis_attr)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。
vi_chn	输入	VI通道号。 取值范围：[0, 1)
dis_attr	输出	DIS属性结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.43 hi_mpi_vi_set_chn_dis_param

函数功能

设置VI通道的DIS可选参数，仅可调整GME模式下的防抖效果，GYRO模式下无效。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用hi_mpi_vi_create_pipe接口创建PIPE。

函数原型

hi_s32 hi_mpi_vi_set_chn_dis_param(hi_vi_pipe vi_pipe, hi_vi_chn vi_chn, const hi_dis_param *dis_param)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。
vi_chn	输入	VI通道号。 取值范围：[0, 1)
dis_param	输入	DIS可选参数结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.44 hi_mpi_vi_get_chn_dis_param

函数功能

获取VI通道的DIS可选参数，仅在GME模式下有效，GYRO模式下无效。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用hi_mpi_vi_create_pipe接口创建PIPE。

函数原型

hi_s32 hi_mpi_vi_get_chn_dis_param(hi_vi_pipe vi_pipe, hi_vi_chn vi_chn, hi_dis_param *dis_param)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。
vi_chn	输入	VI通道号。 取值范围：[0, 1)
dis_param	输出	DIS可选参数属性结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.45 hi_mpi_vi_set_chn_ldc_attr

函数功能

设置VI镜头畸变校正（ LDC ）属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用hi_mpi_vi_create_pipe接口创建PIPE。
- 必须调用hi_mpi_vi_set_chn_attr接口设置通道属性后，再调用本接口。

函数原型

hi_s32 hi_mpi_vi_set_chn_ldc_attr(hi_vi_pipe vi_pipe, hi_vi_chn vi_chn, const hi_vi_ldc_attr *ldc_attr)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。
vi_chn	输入	VI通道号。 取值范围：[0, 1)
ldc_attr	输入	LDC属性结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.4 VI视频输入返回码**。

11.18.9.46 hi_mpi_vi_get_chn_ldc_attr

函数功能

获取VI镜头畸变校正（ LDC ）属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用**hi_mpi_vi_create_pipe**接口创建PIPE。

函数原型

hi_s32 hi_mpi_vi_get_chn_ldc_attr(hi_vi_pipe vi_pipe, hi_vi_chn vi_chn, hi_vi_ldc_attr *ldc_attr)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。
vi_chn	输入	VI通道号。 取值范围：[0, 1)
ldc_attr	输入&输出	LDC属性结构体指针。 ldc_attr参数作为输入时，需要指定该参数结构体内的ldc_version，否则会返回报错。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.47 hi_mpi_vi_get_chn_frame

函数功能

从指定VI通道获取的视频图像信息，图像信息主要包括：图像的宽度、高度、像素格式、时间戳以及YUV各分量的地址等等。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用hi_mpi_vi_create_pipe接口创建PIPE。
- 通道已启用且通道队列深度不为0时，才能调用本接口获取到图像。
- 支持多次获取后再释放，但建议获取和释放接口配对使用。
- 获取的地址信息来自模块内部使用的临时内存，因此使用完之后，必须要调用hi_mpi_vi_release_chn_frame接口释放其内存。
- 用户可结合hi_mpi_vi_get_chn_fd接口，采用select/epoll方式等待可读图像信息，等到有可用的视频输入数据后再通过非阻塞方式使用hi_mpi_vi_get_chn_frame获取图像。

函数原型

hi_s32 hi_mpi_vi_get_chn_frame(hi_vi_pipe vi_pipe, hi_vi_chn vi_chn, hi_video_frame_info *frame_info, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。
vi_chn	输入	VI通道号。 取值范围：[0, 1)
frame_info	输入	VI帧信息结构体。

参数名	输入/输出	说明
milli_sec	输入	超时参数 。取值范围： ● -1：表示阻塞模式，程序一直等待，直到获取到图像才返回； ● 0：表示非阻塞模式； ● >0： 配置具体的超时时间，单位为毫秒（ms），在配置的超时时间内如果没有获取到图像，则超时返回。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.4 VI视频输入返回码**。

11.18.9.48 hi_mpi_vi_release_chn_frame

函数功能

释放一帧从VI通道获取的图像。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 本接口需与**hi_mpi_vi_get_chn_frame**接口配对使用，接口调用次数保持一致，否则返回报错。
- 用户需先调用**hi_mpi_vi_get_chn_frame**接口获取frame_info，再将获取到的frame_info传入本接口，否则可能造成释放不成功。

函数原型

hi_s32 hi_mpi_vi_release_chn_frame(hi_vi_pipe vi_pipe, hi_vi_chn vi_chn, const hi_video_frame_info *frame_info)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。
vi_chn	输入	VI通道号。 取值范围：[0, 1)
frame_info	输入	VI帧信息结构体。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.9.49 hi_mpi_vi_get_chn_fd

函数功能

获取VI通道文件描述符 。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vi_get_chn_fd(hi_vi_pipe vi_pipe, hi_vi_chn vi_chn)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	PIPE号。 取值范围：[0, 12)。
vi_chn	输入	VI通道号。 取值范围：[0, 1)

返回值说明

当大于或等于0时，接口调用成功，返回对应句柄，否则表示接口调用失败。

11.18.9.50 hi_mpi_vi_set_chn_low_delay_attr

函数功能

设置VI PIPE低延时属性 。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用hi_mpi_vi_create_pipe接口创建PIPE。
- 为确保设置的低延时生效，需：
  - 调用hi_mpi_sys_bind接口绑定VI与VPSS。
  - 已开启视频防抖（DIS）、镜头畸变校正（LDC）中的任意一个功能或组合功能。

函数原型

hi_s32 hi_mpi_vi_set_chn_low_delay_attr(hi_vi_pipe vi_pipe, hi_vi_chn vi_chn, const hi_vi_low_delay_info *delay_info)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	VI PIPE号。 取值范围：[0, 12)。
vi_chn	输入	VI通道号。 取值范围：[0, 1)
delay_info	输入	低延时功能参数结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.4 VI视频输入返回码**。

11.18.9.51 hi_mpi_vi_get_chn_low_delay_attr

函数功能

获取VI通道低延时属性 。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口前，必须先调用**hi_mpi_vi_create_pipe**接口创建PIPE。

函数原型

hi_s32 hi_mpi_vi_get_chn_low_delay_attr(hi_vi_pipe vi_pipe, hi_vi_chn vi_chn, hi_vi_low_delay_info *delay_info)

参数说明

参数名	输入/输出	说明
vi_pipe	输入	VI PIPE号。 取值范围：[0, 12)。
vi_chn	输入	VI通道号。 取值范围：[0, 1)

参数名	输入/输出	说明
delay_info	输出	低延时功能参数结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.4 VI视频输入返回码。

11.18.10 Region 区域管理

11.18.10.1 功能及约束说明

功能说明

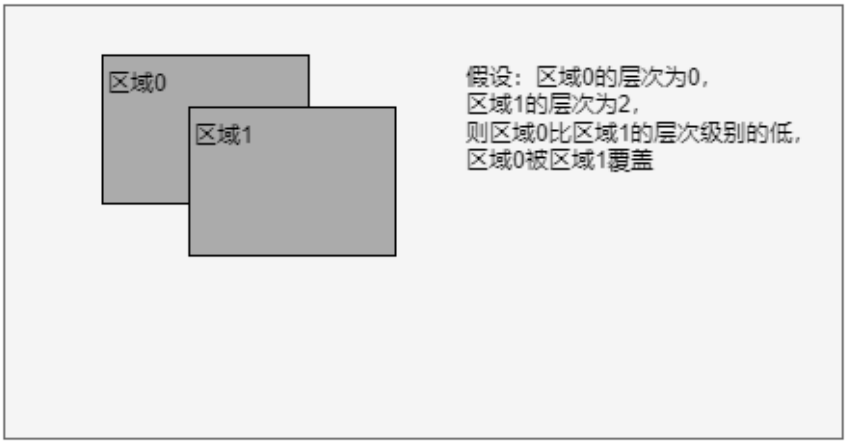
叠加在视频上的OSD (On Screen Display)和遮挡在视频上的色块统称为区域。**区域管理模块**，用于统一管理这些区域资源，用于在视频上显示一些特定信息（如通道号、时间戳等）、或在视频中填充色块用于遮挡。

例如， 实际应用中，用户通过创建一个区域，通过调用hi_mpi_rgn_attach_to_chn接口，将该区域叠加到指定的一个或多个VPSS通道中，同时支持在每个通道中指定区域的显示属性，如位置、层次、透明度等。

基本概念

表 11-41 概念解释

概念	描述
区域类型	区域类型包括以下： ● OVERLAY ： 视频叠加区域 ，其中区域支持位图的加载 、 背景色更新 等功能。 ● OVERLAYEX ： 扩展视频叠加区域，功能与 OVERLAY 类似，支持位图加载、背景色更新等。 ● COVER ： 视频遮挡区域，其中区域支持纯色块遮挡。 ● COVEREX ： 扩展视频遮挡区域，功能与 COVER 类似，支持纯色块遮挡。 ● LINE ： 扩展的线条视频叠加区域，支持颜色、粗细等调节。 ● MOSAIC：马赛克遮挡区域，支持精度调节。 ● MOSAICEX：扩展的马赛克遮挡区域，支持精度调节。 ● CORNER_RECTEX：角框区域，支持大小，颜色，粗细等调节。 其中，OVERLAYEX 类型的区域会绑定到VPSS的Channel上；MOSAIC、COVER 类型的区域会绑定到VPSS的Group上。VPSS的Group和Channel区别详见11.18.11.1 功能说明

概念	描述
区域层次	指区域的叠加级别，层次值越大，表示区域的显示级别越高。当发生重叠时，层次值大的将会覆盖层次值小的。 **图 11-12** 层次叠加示意图 
位图填充	指将位图的内存值填充到区域内存空间中，位图将会从区域的左上角开始填充。当位图小于区域时，只能填充一部分内存，剩余部分保持原有值；位图大小等于区域时，将刚好全部填充；当位图大于区域时，位图只能将自身和区域一样大小的内存信息填充到区域中。**位图填充仅针对OVERLAYEX区域类型有效。**
区域属性	用户创建一个区域时，需要设置该区域的属性信息，包含OVERLAYEX包含像素格式、大小、背景色等。
通道显示属性	指区域在某通道的显示特征 。 例如，MOSAIC的通道显示属性包含显示位置、区域大小、层次、mosaic块大小。 当区域的通道显示属性中的is_show（区域是否显示）为TRUE时，表示区域会显示在该通道中；反之，表示区域在该通道中存在，但处于隐藏状态。

约束说明

表 11-42 region 支持的模块信息

类型	支持模块（mod_id）	设备号（dev_id）取值范围	通道号（chn_id）取值范围
OVERLAY	当前不支持	-	-
COVER	VPSS（HI_ID_VPSS）	VPSS Group ID取值范围：[0,255]	0
OVERLAYEX	VPSS（HI_ID_VPSS）	VPSS Group ID取值范围：[0,255]	[0,1]

类型	支持模块（mod_id）	设备号（dev_id）取值范围	通道号（chn_id）取值范围
COVEREX	当前不支持	-	-
LINE	当前不支持	-	-
MOSAIC	VPSS（HI_ID_VPSS）	VPSS Group ID取值范围：[0,255]	0
MOSAICEX	当前不支持	-	-
CORNER_RECTEX	当前不支持	-	-

11.18.10.2 hi_mpi_rgn_create

函数功能

创建不同类型的区域。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 不支持重复创建。
- 创建区域时，本接口只进行基本的参数的检查，譬如：最小宽高，最大宽高等；当调用hi_mpi_rgn_attach_to_chn接口将区域叠加到通道上时，根据各通道模块支持类型的约束条件进行更加有针对性的参数检查，譬如支持的像素格式等。

函数原型

hi_s32 hi_mpi_rgn_create(hi_rgn_handle handle, const hi_rgn_attr *rgn_attr)

参数说明

参数名	输入/输出	说明
handle	输入	区域句柄号。 必须是未使用的handle号，由用户指定，意义等同于ID号。 取值范围：[0, 1024)。
rgn_attr	输入	区域属性指针，不能为空。 当前仅支持创建COVER、MOSAIC、OVERLAYEX类型的区域。其它区域属性，如区域位置、层次等，需调用hi_mpi_rgn_attach_to_chn接口时指定。

返回值说明

- 0：获取数据成功
- 非0：失败，参见19.18.5-区域管理返回码

11.18.10.3 hi_mpi_rgn_destroy

函数功能

销毁区域。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

区域必须已创建。

函数原型

hi_s32 hi_mpi_rgn_destroy(hi_rgn_handle handle)

参数说明

参数名	输入/输出	说明
handle	输入	区域句柄号。 取值范围：[0, 1024)。

返回值说明

- 0：获取数据成功
- 非0：失败，参见19.18.5-区域管理返回码

11.18.10.4 hi_mpi_rgn_attach_to_chn

函数功能

将区域叠加到通道上。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 区域必须已创建。
- 支持多次调用本接口，但此接口不用于改变hi_mpi_rgn_create接口创建的区域属性。

- 如果将某个区域重复叠加到指定通道上，接口返回成功，但不会改变之前设置的区域通道显示属性。
- 最多支持将4个不同的MOSAIC类型的区域绑定到同一个通道上。
- 最多支持将8个不同的COVER类型的区域绑定到同一个通道上。
- 最多支持将8个不同的OVERLAYEX类型的区域绑定到同一个通道上。

函数原型

hi_s32 hi_mpi_rgn_attach_to_chn(hi_rgn_handle handle, const hi_mpp_chn *chn, const hi_rgn_chn_attr *chn_attr)

参数说明

参数名	输入/输出	说明
handle	输入	区域句柄号。 取值范围：[0, 1024)。
chn	输入	通道结构体指针，不能为空。 该参数的详细配置请参见**表2 region支持的模块信息**。
chn_attr	输入	区域通道显示属性指针，不能为空。

返回值说明

- 0：获取数据成功
- 非0：失败，参见**19.18.5-区域管理返回码**

11.18.10.5 hi_mpi_rgn_detach_from_chn

函数功能

从通道中删除叠加区域。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 区域必须已创建。
- 支持多次调用本接口。

函数原型

hi_s32 hi_mpi_rgn_detach_from_chn(hi_rgn_handle handle, const hi_mpp_chn *chn)

参数说明

参数名	输入/输出	说明
handle	输入	区域句柄号。 取值范围：[0, 1024)。
chn	输入	通道结构体指针，不能为空。 该参数的详细配置请参见**表2 region支持的模块信息**。

返回值说明

- 0：获取数据成功
- 非0：失败，参见**19.18.5-区域管理返回码**

11.18.10.6 hi_mpi_rgn_set_display_attr

函数功能

设置区域的通道显示属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 区域必须已创建。
- 建议先获取属性，再设置。
- 必须先调用**hi_mpi_rgn_attach_to_chn**接口将区域叠加到通道上。

函数原型

hi_s32 hi_mpi_rgn_set_display_attr(hi_rgn_handle handle, const hi_mpp_chn *chn, const hi_rgn_chn_attr *chn_attr)

参数说明

参数名	输入/输出	说明
handle	输入	区域句柄号。 取值范围：[0, 1024)。
chn	输入	通道结构体指针，不能为空。 该参数的详细配置请参见**表2 region支持的模块信息**。

参数名	输入/输出	说明
chn_attr	输入	区域通道显示属性指针，不能为空。 其中，静态属性不能修改，动态属性可以被修改。

返回值说明

- 0：获取数据成功
- 非0：失败，参见19.18.5-区域管理返回码

11.18.10.7 hi_mpi_rgn_get_display_attr

函数功能

获取区域的通道显示属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

区域必须已创建。

函数原型

hi_s32 hi_mpi_rgn_get_display_attr(hi_rgn_handle handle, const hi_mpp_chn *chn, hi_rgn_chn_attr *chn_attr)

参数说明

参数名	输入/输出	说明
handle	输入	区域句柄号。 取值范围：[0, 1024)。
chn	输入	通道结构体指针，不能为空。 该参数的详细配置请参见表2 region支持的模块信息。
chn_attr	输出	区域通道显示属性指针，不能为空。

返回值说明

- 0：获取数据成功
- 非0：失败，参见19.18.5-区域管理返回码

11.18.10.8 hi_mpi_rgn_get_canvas_info

函数功能

获取区域的显示画布信息。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 区域必须已创建。
- 主要用于OVERLAYEX类型导入位图数据 。
- 本接口用于获取区域对应的画布信息， 在得到画布地址之后，用户可直接对画布进行操作，譬如：将bmp数据直接填写到该画布中。 然后通过调用hi_mpi_rgn_update_canvas接口更新显示画布数据。

函数原型

hi_s32 hi_mpi_rgn_get_canvas_info(hi_rgn_handle handle, hi_rgn_canvas_info *canvas_info)

参数说明

参数名	输入/输出	说明
handle	输入	区域句柄号。 取值范围：[0, 1024)。
canvas_info	输出	区域显示画布信息。

返回值说明

- 0：获取数据成功
- 非0：失败，参见19.18.5-区域管理返回码

11.18.10.9 hi_mpi_rgn_update_canvas

函数功能

更新显示画布。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 区域必须已创建。
- 本接口配合hi_mpi_rgn_get_canvas_info使用 。 主要用于画布内存数据更新之后，进行画布切换显示。

- 每次调用本接口前，都必须先获取画布信息，然后在调用本接口进行更新。
- 本接口仅支持OVERLAYEX类型的区域 。

函数原型

hi_s32 hi_mpi_rgn_update_canvas(hi_rgn_handle handle)

参数说明

参数名	输入/输出	说明
handle	输入	区域句柄号。 取值范围：[0, 1024)。

返回值说明

- 0：获取数据成功
- 非0：失败，参见19.18.5-区域管理返回码

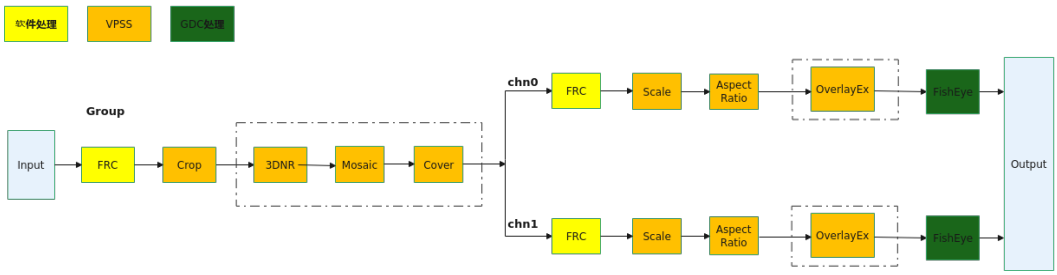
11.18.11 VPSS 视频处理功能

11.18.11.1 功能说明

VPSS模块接收VI/VDEC模块发送过来的图像，可对图像进行抠图、缩放、鱼眼矫正等处理，并实现同源输出两路不同分辨率的图像数据用于编码、预览或抓拍。

用户可调用hi_mpi_sys_bind接口，将VI/VDEC的视频流绑定到VPSS上，送给VPSS做进一步的处理，VPSS通过组Group以及组上的通道Channel来提供对应的处理能力，最终将图像输出给用户或下级模块(例如VO)。

图 11-13 功能框图



VPSS相关概念说明如下：

- Group

  VPSS对用户提供组（ Group ）的概念，各个组通过分时复用方式共享VPSS硬件资源。

- Channel

  VPSS组的通道，一个VPSS组提供多个通道，每个通道具有独立的缩放、鱼眼矫正等功能。

- FRC（Frame Rate Control）

  帧率控制，按控制点分为组帧率控制和通道帧率控制。

  - 组帧率控制，在**hi_mpi_vpss_create_grp**创建组时，可通过帧率控制属性控制各个Group对输入图像的接收。
  - 通道帧率控制，在**hi_mpi_vpss_set_chn_attr**设置组通道属性时，可通过帧率控制属性控制各个通道图像的处理。
- CROP

  从图像中裁剪指定区域。
- 3DNR（Three-dimensional Noise Reduction）

  3D去噪。通过参数配置，把图像中的高斯噪声去除，使得图像变得平滑，有助于降低编码码率。
- Mosaic

  马赛克，对 VPSS 输出图像在指定区域填充马赛克块。该功能需要配合区域管理功能一起使用。
- Cover

  视频遮挡区域，对 VPSS的输出图像填充纯色块。 该功能需要配合区域管理功能一起使用。

  遮挡区域坐标类型分为绝对坐标遮挡和相对坐标比例遮挡。
- Aspect Ratio

  幅形比，指定输出画面相对于输入画面的宽高纵横比。
- OverlayEx

  视频叠加区域，对 VPSS物理通道的输出图像叠加位图。该功能需要配合区域管理功能一起使用。
- FISHEYE

  鱼眼矫正功能。
- Scale

  缩放，对图像进行缩小放大 。 缩放倍数指水平、垂直各缩放多少倍。

11.18.11.2 约束说明

- 所有输入和输出分辨率的宽度必须以4像素对齐，高度必须4像素对齐。
- 视频防抖、畸变矫正下，鱼眼单矫正使用的是内部同一个硬件加速器，该加速器整体规格为4096*2160@45帧，所以仅开启鱼眼单区域矫正时，才能按4096*2160@45满规格运行，否则如果同时开启VI的视频防抖、畸变矫正功能，鱼眼规格降低，预计降一半。
- VPSS的接口，除chn取流相关的接口(**hi_mpi_vpss_get_chn_frame**、**hi_mpi_vpss_release_chn_frame**)外，其他接口都不支持多线程并发调用。
- 在不同Group ID范围内的VPSS Group支持能力存在差别，相关的能力如下表所示：

VPSS Group ID范围	能力说明
Group ID取值在[0, 256)范围内	• 不支持3DNR功能。 • 与区域管理模块配合使用，支持马赛克、Cover、OSD等功能。 • 支持与VDEC、VO模块绑定，不支持与VI模块绑定(如何绑定请参考**hi_mpi_sys_bind**接口)。 • 输入、输出图像所支持处理的最小分辨率为64*64，最大分辨率为4096*8192。 • 若开启鱼眼矫正，输入图像分辨率约束为：宽在[1920,4096]范围内，高在[1080,8192]范围内。 • 不开启鱼眼矫正下，VPSS相关Group的整体处理性能规格为1200FPS@1080P。若开启OverlayEx视频叠加区域功能，则对整体性能规格有影响，影响程度取决于该功能的输出图片分辨率。 • 适用NVR场景，该场景支持多进程调用。
Group ID取值在[256, 268)范围内	• 支持3DNR功能。 • 不支持与区域管理模块配合使用。 • 支持与VI模块绑定，不支持与VDEC、VO模块绑定(如何绑定请参考**hi_mpi_sys_bind**接口)。 • 图像所支持处理的宽高有效范围为[64, 16384]，包括通过hi_mpi_vpss_set_grp_crop设置的裁剪后的宽高或者通过hi_mpi_vpss_set_chn_attr接口设置的图像输出宽高。当用户调用hi_mpi_vpss_create_grp接口，开启3DNR后，图像宽高需满足如下约束： – 不开启3DNR压缩：宽度[64, 16384]；高度[64, 8192]。 – 开启3DNR压缩：宽度[640, 16384]；高度[480, 4096]。 • 若开启鱼眼矫正，输入图像分辨率约束为：宽在[1920,8192]范围内，高在[1080,8192]范围内。 • VPSS整体处理性能规格为4096*2160@45帧。 • 适用Camera场景，该场景下VPSS需配套VI接口一起使用，VPSS相关业务必须和VI业务归属在同一个进程中实现，即只允许有一个进程实例在运行。

11.18.11.3 hi_mpi_vpss_create_grp

函数功能

创建一个VPSS组。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 不支持重复创建。
- 如需使能3DNR，则高度不能超过8192，否则将自动bypass 3dnr算法。

函数原型

hi_s32 hi_mpi_vpss_create_grp(hi_vpss_grp grp, const hi_vpss_grp_attr *grp_attr);

参数说明

参数名	输入/输出	说明
grp	输入	VPSS组ID。 取值范围：[0, 264)。如果通过hi_mpi_vpss_create_grp接口的hi_vpss_grp_attr.nr_en参数开启3DNR功能，则VPSS组ID的取值范围为：[256, 268)。
grp_attr	输入	VPSS组属性指针。

返回值说明

- 0：成功
- 非0：失败，参见6.10.7-VPSS视频处理子系统返回码

11.18.11.4 hi_mpi_vpss_destroy_grp

函数功能

销毁一个VPSS组。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- VPSS组必须已创建，请参见hi_mpi_vpss_create_grp接口。
- 调用此接口之前，必须先调用hi_mpi_vpss_stop_grp接口禁用此组。
- 调用此接口时，会一直等待此VPSS组中的当前任务处理结束，才会真正销毁该VPSS组。

函数原型

hi_s32 hi_mpi_vpss_destroy_grp(hi_vpss_grp grp);

参数说明

参数名	输入/输出	说明
grp	输入	VPSS组ID。 取值范围：[0, 264)。如果通过**hi_mpi_vpss_create_grp**接口的hi_vpss_grp_attr.nr_en参数开启3DNR功能，则VPSS组ID的取值范围为：[256, 268)。

返回值说明

- 0：成功
- 非0：失败，参见**6.10.7-VPSS视频处理子系统返回码**

11.18.11.5 hi_mpi_vpss_start_grp

函数功能

启用VPSS组。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- VPSS组必须已创建，请参见**hi_mpi_vpss_create_grp**接口。
- 重复调用该接口启用同一个组返回成功。

函数原型

hi_s32 hi_mpi_vpss_start_grp(hi_vpss_grp grp);

参数说明

参数名	输入/输出	说明
grp	输入	VPSS组ID。 取值范围：[0, 264)。如果通过**hi_mpi_vpss_create_grp**接口的hi_vpss_grp_attr.nr_en参数开启3DNR功能，则VPSS组ID的取值范围为：[256, 268)。

返回值说明

- 0：成功

- 非0：失败，参见6.10.7-VPSS视频处理子系统返回码

11.18.11.6 hi_mpi_vpss_stop_grp

函数功能

禁用VPSS组。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- VPSS组必须已创建，请参见hi_mpi_vpss_create_grp接口。
- 重复调用本接口禁用同一个组返回成功。

函数原型

hi_s32 hi_mpi_vpss_stop_grp(hi_vpss_grp grp);

参数说明

参数名	输入/输出	说明
grp	输入	VPSS组ID。 取值范围：[0, 264)。如果通过hi_mpi_vpss_create_grp接口的hi_vpss_grp_attr.nr_en参数开启3DNR功能，则VPSS组ID的取值范围为：[256, 268)。

返回值说明

- 0：成功
- 非0：失败，参见6.10.7-VPSS视频处理子系统返回码

11.18.11.7 hi_mpi_vpss_set_grp_crop

函数功能

设置VPSS组的裁剪参数。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

VPSS组必须已创建，请参见hi_mpi_vpss_create_grp接口。

函数原型

hi_s32 hi_mpi_vpss_set_grp_crop(hi_vpss_grp grp, const hi_vpss_crop_info *crop_info);

参数说明

参数名	输入/输出	说明
grp	输入	VPSS组ID。 取值范围：[0, 264)。如果通过hi_mpi_vpss_create_grp接口的hi_vpss_grp_attr.nr_en参数开启3DNR功能，则VPSS组ID的取值范围为：[256, 268)。
crop_info	输入	裁剪区域信息指针。

返回值说明

- 0：成功
- 非0：失败，参见6.10.7-VPSS视频处理子系统返回码

11.18.11.8 hi_mpi_vpss_get_grp_crop

函数功能

获取VPSS组的裁剪参数。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpss_get_grp_crop(hi_vpss_grp grp, hi_vpss_crop_info *crop_info);

参数说明

参数名	输入/输出	说明
grp	输入	VPSS组ID。 取值范围：[0, 264)。如果通过hi_mpi_vpss_create_grp接口的hi_vpss_grp_attr.nr_en参数开启3DNR功能，则VPSS组ID的取值范围为：[256, 268)。
crop_info	输出	裁剪区域信息指针。

返回值说明

- 0：成功
- 非0：失败，参见6.10.7-VPSS视频处理子系统返回码

11.18.11.9 hi_mpi_vpss_set_grp_param

函数功能

设置VPSS组参数。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- VPSS组必须已创建，请参见hi_mpi_vpss_create_grp接口。
- 请务必先调用hi_mpi_vpss_get_grp_param获取当前参数，再进行设置操作。

函数原型

hi_s32 hi_mpi_vpss_set_grp_param(hi_vpss_grp grp, const hi_vpss_grp_param *grp_param);

参数说明

参数名	输入/输出	说明
grp	输入	VPSS组ID。 取值范围：[256, 268)
grp_param	输入	组参数设置。

返回值说明

- 0：成功
- 非0：失败，参见6.10.7-VPSS视频处理子系统返回码

11.18.11.10 hi_mpi_vpss_get_grp_param

函数功能

获取VPSS组参数。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

VPSS组必须已创建，请参见hi_mpi_vpss_create_grp接口。

函数原型

hi_s32 hi_mpi_vpss_get_grp_param(hi_vpss_grp grp, hi_vpss_grp_param *grp_param);

参数说明

参数名	输入/输出	说明
grp	输入	VPSS组ID。 取值范围：[256, 268)
grp_param	输入&输出	组参数设置。 grp_param参数作为输入时，需要指定该参数结构体内的nrx_param.nr_version，否则会返回报错。

返回值说明

- 0：成功
- 非0：失败，参见6.10.7-VPSS视频处理子系统返回码

11.18.11.11 hi_mpi_vpss_set_grp_fisheye_cfg

函数功能

设置VPSS组鱼眼镜头LMF（Lens Map Function）参数。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- VPSS组必须已创建，请参见hi_mpi_vpss_create_grp接口。
- LMF参数需要按照镜头厂商的推荐参数进行转换后再配置，如果镜头厂商未提供，则需要关闭LMF功能。
- 正确的LMF参数符合lmf_coef[i + 1] ≥ lmf_coef[i] + 5 && lmf_coef[i + 1] <= lmf_coef[i] + 31 && lmf_coef[57] < 1024 < lmf_coef[85] && lmf_coef[0] = 0的规律，如果配置的参数不满足此规律则会报错，如果配置的参数有误则可能导致不出图等异常现象。

函数原型

hi_s32 hi_mpi_vpss_set_grp_fisheye_cfg(hi_vpss_grp grp, const hi_fisheye_cfg *fisheye_cfg);

参数说明

参数名	输入/输出	说明
grp	输入	VPSS组ID。 取值范围：[256, 268)。
fisheye_cfg	输入	鱼眼镜头LMF参数结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见**6.10.7-VPSS视频处理子系统返回码**

11.18.11.12 hi_mpi_vpss_get_grp_fisheye_cfg

函数功能

获取VPSS组鱼眼镜头LMF参数。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

VPSS组必须已创建，请参见**hi_mpi_vpss_create_grp**接口。

函数原型

hi_s32 hi_mpi_vpss_get_grp_fisheye_cfg(hi_vpss_grp grp, hi_fisheye_cfg *fisheye_cfg);

参数说明

参数名	输入/输出	说明
grp	输入	VPSS组ID。 取值范围：[256, 268)。
fisheye_cfg	输出	鱼眼镜头LMF参数结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见**6.10.7-VPSS视频处理子系统返回码**

11.18.11.13 hi_mpi_vpss_set_chn_attr

函数功能

设置VPSS物理通道属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

VPSS组必须已创建，请参见**hi_mpi_vpss_create_grp**接口。

函数原型

hi_s32 hi_mpi_vpss_set_chn_attr(hi_vpss_grp grp, hi_vpss_chn chn, const hi_vpss_chn_attr *chn_attr);

参数说明

参数名	输入/输出	说明
grp	输入	VPSS组ID。 取值范围：[0, 264)。如果通过**hi_mpi_vpss_create_grp**接口的hi_vpss_grp_attr.nr_en参数开启3DNR功能，则VPSS组ID的取值范围为：[256, 268)。
chn	输入	VPSS 通道号。 取值范围：[0, 2)
chn_attr	输入	VPSS通道属性。

返回值说明

- 0：成功
- 非0：失败，参见**6.10.7-VPSS视频处理子系统返回码**

11.18.11.14 hi_mpi_vpss_get_chn_attr

函数功能

获取VPSS物理通道属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

VPSS组必须已创建，请参见hi_mpi_vpss_create_grp接口。

函数原型

hi_s32 hi_mpi_vpss_get_chn_attr(hi_vpss_grp grp, hi_vpss_chn chn, hi_vpss_chn_attr *chn_attr);

参数说明

参数名	输入/输出	说明
grp	输入	VPSS组ID。 取值范围：[0, 264)。如果通过hi_mpi_vpss_create_grp接口的hi_vpss_grp_attr.nr_en参数开启3DNR功能，则VPSS组ID的取值范围为：[256, 268)。
chn	输入	VPSS 通道号。 取值范围：[0, 2)
chn_attr	输出	VPSS通道属性。

返回值说明

- 0：成功
- 非0：失败，参见6.10.7-VPSS视频处理子系统返回码

11.18.11.15 hi_mpi_vpss_enable_chn

函数功能

启用VPSS通道。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 需先调用hi_mpi_vpss_set_chn_attr接口设置通道属性后，再调用本接口启用通道。
- 调用本接口重复启用同一个通道返回成功
- VPSS组必须已创建，请参见hi_mpi_vpss_create_grp接口。

函数原型

hi_s32 hi_mpi_vpss_enable_chn(hi_vpss_grp grp, hi_vpss_chn chn);

参数说明

参数名	输入/输出	说明
grp	输入	VPSS组ID。 取值范围：[0, 264)。如果通过hi_mpi_vpss_create_grp接口的hi_vpss_grp_attr.nr_en参数开启3DNR功能，则VPSS组ID的取值范围为：[256, 268)。
chn	输入	VPSS 通道号。 取值范围：[0, 2)

返回值说明

- 0：成功
- 非0：失败，参见6.10.7-VPSS视频处理子系统返回码

11.18.11.16 hi_mpi_vpss_disable_chn

函数功能

禁用VPSS通道。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用本接口重复禁用同一个通道返回成功
- VPSS组必须已创建，请参见hi_mpi_vpss_create_grp接口。

函数原型

hi_s32 hi_mpi_vpss_disable_chn(hi_vpss_grp grp, hi_vpss_chn chn);

参数说明

参数名	输入/输出	说明
grp	输入	VPSS组ID。 取值范围：[0, 264)。如果通过hi_mpi_vpss_create_grp接口的hi_vpss_grp_attr.nr_en参数开启3DNR功能，则VPSS组ID的取值范围为：[256, 268)。

参数名	输入/输出	说明
chn	输入	VPSS 通道号。 取值范围：[0, 2)

返回值说明

- 0：成功
- 非0：失败，参见6.10.7-VPSS视频处理子系统返回码

11.18.11.17 hi_mpi_vpss_get_chn_frame

函数功能

用户从通道获取一帧处理完成的图像。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- VPSS组必须已创建，请参见hi_mpi_vpss_create_grp接口。
- 只有在通过hi_mpi_vpss_get_chn_attr接口设置为USER模式下，并且队列深度不为0，才能获取到图像。
- 调用该接口获取图像，不会对后端绑定的模块有影响。如后端绑定VO显示，可以在显示过程中获取图像，VO仍正常显示，不会受到影响。
- 解码回放场景，由于不允许出现丢帧，VPSS只要有一个通道不处理新图像（通道已开启），则整个VPSS不处理新图像。例如说开启了通道0和通道1，两者都不绑定后端，通道图像队列长度都设为2，此时从通道0中最多获取出2帧已缓存的图像，因为通道1缓存2帧后未处理新图像，所以VPSS不会再处理新图像。

函数原型

```
hi_s32 hi_mpi_vpss_get_chn_frame(hi_vpss_grp grp, hi_vpss_chn chn,
hi_video_frame_info *frame_info, hi_s32 milli_sec);
```

参数说明

参数名	输入/输出	说明
grp	输入	VPSS组ID。 取值范围：[0, 264)。如果通过hi_mpi_vpss_create_grp接口的hi_vpss_grp_attr.nr_en参数开启3DNR功能，则VPSS组ID的取值范围为：[256, 268)。

参数名	输入/输出	说明
chn	输入	VPSS通道号。 取值范围：[0, 2)
frame_info	输出	图像信息。
milli_sec	输入	超时时间。 取值范围： ● -1：表示阻塞模式，程序一直等待，直到获取到图像才返回； ● 0：表示非阻塞模式； ● >0： 配置具体的超时时间，单位为毫秒（ms）。在此指定时间内如果没有获取到图像，则超时返回。

返回值说明

- 0：成功
- 非0：失败，参见6.10.7-VPSS视频处理子系统返回码

11.18.11.18 hi_mpi_vpss_release_chn_frame

函数功能

用户释放一帧通道图像。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

调用hi_mpi_vpss_get_chn_frame接口获取到图像数据后，需及时调用本接口释放资源。

函数原型

hi_s32 hi_mpi_vpss_release_chn_frame(hi_vpss_grp grp, hi_vpss_chn chn, const hi_video_frame_info *frame_info);

参数说明

参数名	输入/输出	说明
grp	输入	VPSS组ID。 取值范围：[0, 264)。如果通过hi_mpi_vpss_create_grp接口的hi_vpss_grp_attr.nr_en参数开启3DNR功能，则VPSS组ID的取值范围为：[256, 268)。
chn	输入	VPSS 通道号。 取值范围：[0, 2)
frame_info	输入	图像信息。

返回值说明

- 0：成功
- 非0：失败，参见6.10.7-VPSS视频处理子系统返回码

11.18.11.19 hi_mpi_vpss_set_chn_fisheye

函数功能

设置VPSS物理通道鱼眼属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- VPSS组必须已创建，请参见hi_mpi_vpss_create_grp接口。
- 调用本接口配置鱼眼属性前必须先调用hi_mpi_vpss_set_chn_attr接口配置对应物理通道的属性。
- 物理通道如果需要开启鱼眼矫正，建议在启用通道前配置鱼眼属性。
- 动态改变鱼眼属性时，鱼眼矫正后的输出图像大小应小于启用通道前配置的大小，如果未进行配置则应小于对应通道的宽高属性
- 物理通道开启了鱼眼矫正后，不支持动态改变通道Crop属性以及通道宽高属性。
- 输出图像的非校正区域是随机数据，可能会出现异常图像。
- 通道输出VIDEO_FORMAT_TILE_16x8格式时，不支持鱼眼。
- 如果在本接口中，hi_fisheye_attr属性开启了LMF参数功能，则必须调用VPSS的hi_mpi_vpss_set_grp_fisheye_cfg设置LMF（Lens Map Function）参数，否则进行鱼眼处理时会因为获取不到用户配置的LMF参数而报错。

函数原型

```
hi_s32 hi_mpi_vpss_set_chn_fisheye(hi_vpss_grp grp, hi_vpss_chn chn, const hi_fisheye_correction_attr *correction_attr);
```

参数说明

参数名	输入/输出	说明
grp	输入	VPSS组ID。 取值范围：[0, 264)。
chn	输入	VPSS 通道号。 取值范围：[0, 2)。
correction_attr	输入	VPSS物理通道的鱼眼属性结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见6.10.7-VPSS视频处理子系统返回码

11.18.11.20 hi_mpi_vpss_get_chn_fisheye

函数功能

获取用户前一次调用hi_mpi_vpss_set_chn_fisheye接口设置的VPSS物理通道鱼眼属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- VPSS组必须已创建，请参见hi_mpi_vpss_create_grp接口。
- 必须先设置才能获取，否则会获取失败。

函数原型

hi_s32 hi_mpi_vpss_get_chn_fisheye(hi_vpss_grp grp, hi_vpss_chn chn, hi_fisheye_correction_attr *correction_attr);

参数说明

参数名	输入/输出	说明
grp	输入	VPSS组ID。 取值范围：[0, 264)。
chn	输入	VPSS 通道号。 取值范围：[0, 2)。
correction_attr	输出	VPSS物理通道的鱼眼属性结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见6.10.7-VPSS视频处理子系统返回码

11.18.11.21 hi_mpi_vpss_get_chn_fd

函数功能

获取VPSS通道对应的设备文件句柄。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpss_get_chn_fd(hi_vpss_grp grp, hi_vpss_chn chn);

参数说明

参数名	输入/输出	说明
grp	输入	VPSS组ID。 取值范围：[0, 264)。如果通过hi_mpi_vpss_create_grp接口的hi_vpss_grp_attr.nr_en参数开启3DNR功能，则VPSS组ID的取值范围为：[256, 268)。
chn	输入	VPSS 通道号。 取值范围：[0, 2)。

返回值说明

当大于或等于0时，接口调用成功，返回对应句柄，否则表示接口调用失败。

11.18.12 VO 视频输出功能

11.18.12.1 hi_mpi_vo_set_pub_attr

函数功能

配置视频输出设备的公共属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 视频输出设备属性，必须在执行hi_mpi_vo_enable前配置。
- 该接口暂不支持多进程。

函数原型

hi_s32 hi_mpi_vo_set_pub_attr(hi_vo_dev dev, const hi_vo_pub_attr*pub_attr)

参数说明

参数名	输入/输出	说明
dev	输入	视频输出设备号。 取值范围：[0, 2)。
pub_attr	输入	视频输出设备公共属性结构体指针。 静态属性。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.7 VO视频输出返回码

11.18.12.2 hi_mpi_vo_enable

函数功能

使用视频输出功能前必须先调用本接口使能视频输出设备。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 在调用设备使能前，必须对设备公共属性进行配置，否则返回设备未配置错误。
- 如果设备已经使能，调用此接口则返回未禁用错误，不支持重复使能。
- 为适应开机画面与正常操作界面间顺畅切换，此处需要检查VO（Video Output）硬件是否已经使能，如果已使能则返回成功，且沿用已有接口和时序配置。
- 如果希望更改VO的接口或时序配置，则需要先调用hi_mpi_vo_disable接口，强制禁用VO硬件后再使能。
- 该接口暂不支持多进程。

函数原型

hi_s32 hi_mpi_vo_enable(hi_vo_dev dev)

参数说明

参数名	输入/输出	说明
dev	输入	视频输出设备号。 取值范围：[0, 2)。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.7 VO视频输出返回码

11.18.12.3 hi_mpi_vo_disable

函数功能

禁用视频输出设备。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 设备禁用前必须先禁用该设备上的视频层。
- 设备禁用前，如果有使能回写功能，则必须先禁用回写功能。
- 设备禁用后需要重新调用hi_mpi_vo_set_pub_attr设置设备公共属性，才可使能设备。
- 该接口暂不支持多进程。

函数原型

hi_s32 hi_mpi_vo_disable(hi_vo_dev dev)

参数说明

参数名	输入/输出	说明
dev	输入	视频输出设备号。 取值范围：[0, 2)。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.7 VO视频输出返回码

11.18.12.4 hi_mpi_vo_set_video_layer_attr

函数功能

设置视频层属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 需要在视频层所绑定的设备处于使能状态时才能设置视频层属性。
- 设置视频层属性（SINGLE模式下除了layer_attr中display_rect的x,y）必须在视频层禁用的情况下进行。
- 该接口暂不支持多进程。

函数原型

hi_s32 hi_mpi_vo_set_video_layer_attr(hi_vo_layer layer, const hi_vo_video_layer_attr *layer_attr)

参数说明

参数名	输入/输出	说明
layer	输入	视频输出视频层号。 取值范围：[0, 9)。 不支持图形层和级联视频层。
layer_attr	输入	视频层属性结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.7 VO视频输出返回码**

11.18.12.5 hi_mpi_vo_get_video_layer_attr

函数功能

获取视频层属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 建议在设置视频层属性前先调用此接口获取视频层属性。

- 该接口暂不支持多进程。

函数原型

hi_s32 hi_mpi_vo_get_video_layer_attr(hi_vo_layer layer, hi_vo_video_layer_attr *layer_attr)

参数说明

参数名	输入/输出	说明
layer	输入	视频输出视频层号。 取值范围：[0, 9)。 不支持图形层和级联视频层。
layer_attr	输出	视频层属性结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.7 VO视频输出返回码**

11.18.12.6 hi_mpi_vo_enable_video_layer

函数功能

使能视频层。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 视频层使能前必须保证该视频层所绑定的设备处于使能状态。
- 视频层使能前必须保证该视频层已经配置。
- 该接口暂不支持多进程。

函数原型

hi_s32 hi_mpi_vo_enable_video_layer(hi_vo_layer layer)

参数说明

参数名	输入/输出	说明
layer	输入	视频输出视频层号。 取值范围：[0, 9)。 不支持图形层和级联视频层。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.7 VO视频输出返回码

11.18.12.7 hi_mpi_vo_disable_video_layer

函数功能

禁用视频层。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 视频层禁用前必须保证其上的通道全部禁用。
- 在禁用视频层时，非直通情况下，如果用户没有释放从VO获取的图像buffer资源，该接口会返回HI_ERR_VO_BUSY的错误码，表示VO创建的VB资源没有释放。多见于用户调用获取屏幕图像未释放的情况下。
- 在禁用视频层时，必须保证将该设备的回写功能禁用。
- 该接口暂不支持多进程。

函数原型

hi_s32 hi_mpi_vo_disable_video_layer(hi_vo_layer layer)

参数说明

参数名	输入/输出	说明
layer	输入	视频输出视频层号。 取值范围：[0, 9)。 不支持图形层和级联视频层。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.7 VO视频输出返回码

11.18.12.8 hi_mpi_vo_set_chn_attr

函数功能

配置指定视频输出通道的属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 通道显示区域不能超过视频层属性中设定的画布大小(img_size大小)。
- 该接口为动态设置接口，可在VO设备使能且相应视频层已配置的情况下调用。
- 设备的通道属性rect需2对齐，且宽高必须不小于32。起始点的纵坐标建议按照4对齐配置，否则可能会出现不可预料的图像质量问题。
- 如果设备有视频层放大的情况，rect是放大前视频层上的通道原始的起始位置和宽高，放大后的通道的显示起始位置和宽高会按视频层放大的比例偏移或放大。
- 该接口暂不支持多进程。

函数原型

hi_s32 hi_mpi_vo_set_chn_attr(hi_vo_layer layer, hi_vo_chn chn, const hi_vo_chn_attr *chn_attr)

参数说明

参数名	输入/输出	说明
layer	输入	视频输出视频层号。 取值范围：[0, 9)。 不支持图形层和级联视频层。
chn	输入	视频输出通道的通道号。 取值范围：[0, 64)。
chn_attr	输入	视频通道属性指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.7 VO视频输出返回码**

11.18.12.9 hi_mpi_vo_enable_chn

函数功能

使能指定的视频输出通道。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用前必须保证视频层绑定关系存在，否则将返回失败。
- 调用前必须使能相应设备上的视频层。
- 通道使能前必须进行通道配置，否则返回通道未配置的错误。
- 允许重复使能同一视频输出通道，不返回失败。

- 该接口暂不支持多进程。

函数原型

hi_s32 hi_mpi_vo_enable_chn (hi_vo_layer layer, hi_vo_chn chn)

参数说明

参数名	输入/输出	说明
layer	输入	视频输出视频层号。 取值范围：[0, 9)。 不支持图形层和级联视频层。
chn	输入	视频输出通道的通道号。 取值范围：[0, 64)。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.7 VO视频输出返回码**

11.18.12.10 hi_mpi_vo_disable_chn

函数功能

禁用指定的视频输出通道。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 允许重复禁用同一视频输出通道，不返回失败。
- 当高清设备的通道绑定VPSS（Video Processing Subsystem）时，建议先调用本接口停止通道后，再解绑定VO通道与VPSS通道的绑定关系，否则可能出现HI_ERR_VO_BUSY的超时返回错误。
- 该接口暂不支持多进程。

函数原型

hi_s32 hi_mpi_vo_disable_chn(hi_vo_layer layer, hi_vo_chn chn)

参数说明

参数名	输入/输出	说明
layer	输入	视频输出视频层号。 取值范围：[0, 9)。 不支持图形层和级联视频层。
chn	输入	视频输出通道的通道号。 取值范围：[0, 64)。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.7 VO视频输出返回码**

11.18.12.11 hi_mpi_vo_set_chn_param

函数功能

本接口需与VPSS功能配合使用（调用**hi_mpi_sys_bind**接口绑定VO与VPSS），通过本接口配置幅型比参数，VPSS根据本接口配置的参数实现幅型比功能。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 本接口为动态设置接口，需在VO设备使能且相应视频层和通道属性已配置的情况下调用。
- 调用**hi_mpi_vo_set_video_layer_attr**接口将视频层的分割模式设置为HI_VO_PARTITION_MODE_MULTI，且调用VPSS的**hi_mpi_vpss_set_chn_attr**接口将VPSS通道的工作模式设置为HI_VPSS_CHN_MODE_AUTO模式、输入图像宽高比与通道宽高比不相等时，可通过本接口设置参数，否则参数设置无效。
- 该接口暂不支持多进程。

函数原型

hi_s32 hi_mpi_vo_set_chn_param(hi_vo_layer layer, hi_vo_chn chn, const hi_vo_chn_param *chn_param)

参数说明

参数名	输入/输出	说明
layer	输入	视频输出视频层号。 取值范围：[0, 9)。 不支持图形层和级联视频层。

参数名	输入/输出	说明
chn	输入	视频输出通道的通道号。 取值范围：[0, 64)。
chn_param	输入	视频通道参数指针

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.7 VO视频输出返回码**

11.18.12.12 hi_mpi_vo_set_chn_frame_rate

函数功能

设置指定视频输出通道的显示帧率。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 需先调用**hi_mpi_vo_set_chn_attr**接口设置VO相应视频输出通道属性 ，才能调用本接口，否则返回失败。
- 调用**hi_mpi_vo_enable_chn**接口会使通道重置为正常速度播放，因此，必须在通道使能后再调用本接口进行播放控制。
- 在通道禁用的情况下，调用**hi_mpi_vo_set_chn_attr**接口会将通道帧率重置为显示帧率。

  通道帧率为通道处理图像数据的帧率，显示帧率为显示设备的帧率，显示帧率可以在调用**hi_mpi_vo_set_video_layer_attr**接口设置视频层属性时配置。
- VO与VPSS绑定场景下，该接口设置的帧率生效。
- 该接口暂不支持多进程。

函数原型

hi_s32 hi_mpi_vo_set_chn_frame_rate(hi_vo_layer layer, hi_vo_layer chn, hi_s32 frame_rate)

参数说明

参数名	输入/输出	说明
layer	输入	视频输出视频层号。 取值范围：[0, 9)。 layer参数设置为0、1时有效，设置其它值无效。

参数名	输入/输出	说明
chn	输入	视频输出通道的通道号。 取值范围：[0, 64)。
frame_rate	输入	通道帧率，帧率可设置为Nx，其中N为[1,64]的任意整数，x为设备帧率。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.7 VO视频输出返回码**

11.18.12.13 hi_mpi_vo_pause_chn

函数功能

暂停指定的视频输出通道。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用**hi_mpi_vo_enable_chn**接口启用通道后，才可以调用本接口。
- 若调用**hi_mpi_vo_set_video_layer_attr**接口将视频层的分割模式设置为HI_VO_PARTITION_MODE_SINGLE时，则不支持暂停指定通道，调用本接口设置参数无效，但不影响视频显示结果。
- 调用**hi_mpi_vo_send_frame**接口将视频图像送入指定输出通道显示时，需至少2个图像帧轮转（即定义2个hi_video_frame_info结构体变量），否则暂停功能失效。
- 允许重复暂停同一通道，不返回失败。
- 该接口暂不支持多进程。

函数原型

hi_s32 hi_mpi_vo_pause_chn(int layer, int chn)

参数说明

参数名	输入/输出	说明
layer	输入	视频输出视频层号。 取值范围：[0, 9)。 layer参数设置为0、1时有效，设置其它值无效。

参数名	输入/输出	说明
chn	输入	视频输出通道的通道号。 取值范围：[0, 64)。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.7 VO视频输出返回码

11.18.12.14 hi_mpi_vo_resume_chn

函数功能

恢复指定的视频输出通道。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用hi_mpi_vo_enable_chn接口启用通道后，才可以调用本接口。
- 若调用hi_mpi_vo_set_video_layer_attr接口将视频层的分割模式设置为HI_VO_PARTITION_MODE_SINGLE时，则不支持恢复指定通道，调用本接口设置参数无效，但不影响视频显示结果。
- 调用hi_mpi_vo_send_frame接口将视频图像送入指定输出通道显示时，需至少2个图像帧轮转（即定义2个hi_video_frame_info结构体变量），否则恢复功能失效。
- 允许重复恢复同一通道，不返回失败。
- 该接口暂不支持多进程。

函数原型

hi_s32 hi_mpi_vo_resume_chn(int layer, int chn)

参数说明

参数名	输入/输出	说明
layer	输入	视频输出视频层号。 取值范围：[0, 9)。 layer参数设置为0、1时有效，设置其它值无效。
chn	输入	视频输出通道的通道号。 取值范围：[0, 64)。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.7 VO视频输出返回码

11.18.12.15 hi_mpi_vo_hide_chn

函数功能

隐藏指定通道。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用该接口前需要由用户保证通道已经使能。
- 当VO与VPSS配合使用场景下，在资源释放时，建议先调用本接口隐藏通道，再调用hi_mpi_sys_unbind接口解除VO与VPSS的绑定关系，否则可能出现HI_ERR_VO_BUSY的错误。
- 隐藏通道后对该通道的送图/暂停/恢复等通道操作无效。必须在通道恢复后重新对通道进行送图操作，通道才可重新显示画面
- 该接口暂不支持多进程。

函数原型

hi_s32 hi_mpi_vo_hide_chn(hi_vo_layer layer, hi_vo_chn chn)

参数说明

参数名	输入/输出	说明
layer	输入	视频输出视频层号。 取值范围：[0, 9)。 layer参数设置为0、1时有效，设置其它值无效。
chn	输入	视频输出通道的通道号。 取值范围：[0, 64)。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.7 VO视频输出返回码

11.18.12.16 hi_mpi_vo_show_chn

函数功能

显示指定通道。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用前需保证视频输出通道已使能，且视频输出通道所在的视频层已经使能。
- 默认情况下通道处于显示状态。
- VO在SINGLE模式下不支持显示指定通道，接口参数配置无效，不影响显示结果。
- 使能通道后，程序退出时必须由用户保证正确销毁全部申请的通道资源、vb相关内存资源，否则属于未定义行为，不保证后续送显。
- 该接口暂不支持多进程。

函数原型

hi_s32 hi_mpi_vo_show_chn(hi_vo_layer layer, hi_vo_chn chn)

参数说明

参数名	输入/输出	说明
layer	输入	视频输出视频层号。 取值范围：[0, 9)。 layer参数设置为0、1时有效，设置其它值无效。
chn	输入	视频输出通道的通道号。 取值范围：[0, 64)。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.7 VO视频输出返回码**

11.18.12.17 hi_mpi_vo_send_frame

函数功能

将视频图像送入指定输出通道显示。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用该接口前必须保证通道已经使能。
- 输入视频数据信息要符合VO数据的要求。宽和高需要与实际图像宽高相符，且均不能小于32，宽高要求以2对齐。
- single模式下只有一个通道可被显示，故调用hi_mpi_vo_set_chn_attr接口设置通道优先级后，只显示优先级最高的通道。
- 该接口暂不支持多进程。

函数原型

hi_s32 hi_mpi_vo_send_frame(hi_vo_layer layer, hi_vo_chn chn, const hi_video_frame_info *frame_info, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
layer	输入	视频输出视频层号。 取值范围：[0, 9)。 不支持图形层和级联视频层。
chn	输入	视频输出通道的通道号。 取值范围：[0, 64)。
frame_info	输入	视频数据信息指针。 single模式下用户读取的图片对应宽高不可大于目标通道的宽高。
milli_sec	输入	超时参数 ，取值范围如下（当前仅支持0）： ● -1：表示阻塞模式，一直等到有图像发送为止； ● 0：表示非阻塞模式； ● >0： 配置具体的超时时间，超过该时间则不再等待，单位为毫秒（ms）。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.7 VO视频输出返回码

11.18.12.18 hi_mpi_vo_create_pool

函数功能

创建VO显示内存池。预留接口，暂不支持。

函数原型

hi_s32 hi_mpi_vo_create_pool(const hi_u64 size)

参数说明

参数名	输入/输出	说明
size	输入	申请内存池大小

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.7 VO视频输出返回码**

11.18.12.19 hi_mpi_vo_handle_to_phys_addr

函数功能

根据内存池句柄。预留接口，暂不支持。

函数原型

hi_u64 hi_mpi_vo_handle_to_phys_addr(hi_s32 handle)

参数说明

参数名	输入/输出	说明
handle	输入	内存池句柄

返回值说明

- 0：失败，地址为NULL
- 非0：成功，申请的内存地址。

11.18.12.20 hi_mpi_vo_destroy_pool

函数功能

销毁VO显示内存池。预留接口，暂不支持。

函数原型

hi_s32 hi_mpi_vo_destroy_pool(hi_s32 handle)

参数说明

参数名	输入/输出	说明
handle	输入	内存池句柄

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.7 VO视频输出返回码

11.18.12.21 hi_mpi_vo_bind_layer

函数功能

绑定视频层或图形层到某个设备，当前版本不支持绑定视频层。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 对图形层已经存在的绑定关系，不支持重复绑定。需要先解除绑定，再重新绑定。
- 不能在图形层使能的情况下，调用该接口设置绑定关系。
- 图形层仅支持G2、G3、G4调用本接口，G0、G1为固定绑定关系，不可修改绑定。
- 图形层G2、G3支持绑定到DHD0或DHD1，G4支持绑定到DHD1或不绑定，G3与G4不支持同时绑定到DHD1。
- 该接口暂不支持多进程。

函数原型

hi_s32 hi _mpi_vo_bind_layer(hi_vo_layer layer, hi_vo_dev dev)

参数说明

参数名	输入/输出	说明
layer	输入	视频/图形层号。 取值范围：[0,9)。
dev	输入	视频输出设备号。 取值范围：[0, 2)。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.7 VO视频输出返回码

11.18.12.22 hi_mpi_vo_unbind_layer

函数功能

从某个设备解绑定视频层或图形层。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 将视频层或图形层从设备解除绑定前，必须禁用视频层或图形层 。
- 重复解绑定返回成功。
- 不强制要求传入的形参dev必须是之前layer层绑定的设备号用户调用此接口就视为解除layer层的绑定关系。
- 该接口暂不支持多进程。

函数原型

hi_s32 hi_mpi_vo_unbind_layer(hi_vo_layer layer, hi_vo_dev dev)

参数说明

参数名	输入/输出	说明
layer	输入	视频/图形层号。 取值范围：[0,9)。
dev	输入	视频输出设备号。 取值范围：[0, 2)。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.7 VO视频输出返回码

11.18.13 TDE 图形绘制功能

11.18.13.1 hi_tde_open

函数功能

打开TDE（Two Dimension Engine）设备。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 在进行TDE的相关接口操作前必须进行打开操作，保证TDE设备处于打开状态。
- 该接口不支持多进程。

函数原型

hi_s32 hi_tde_open(hi_void)

参数说明

无

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.8 TDE图形绘制返回码**

11.18.13.2 hi_tde_close

函数功能

关闭TDE设备。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用**hi_tde_open**接口打开TDE设备后才可调用本接口。
- 调用**hi_tde_open**与**hi_tde_close**的次数需要对应。
- 该接口不支持多进程。

函数原型

hi_s32 hi_tde_close(hi_void)

参数说明

无

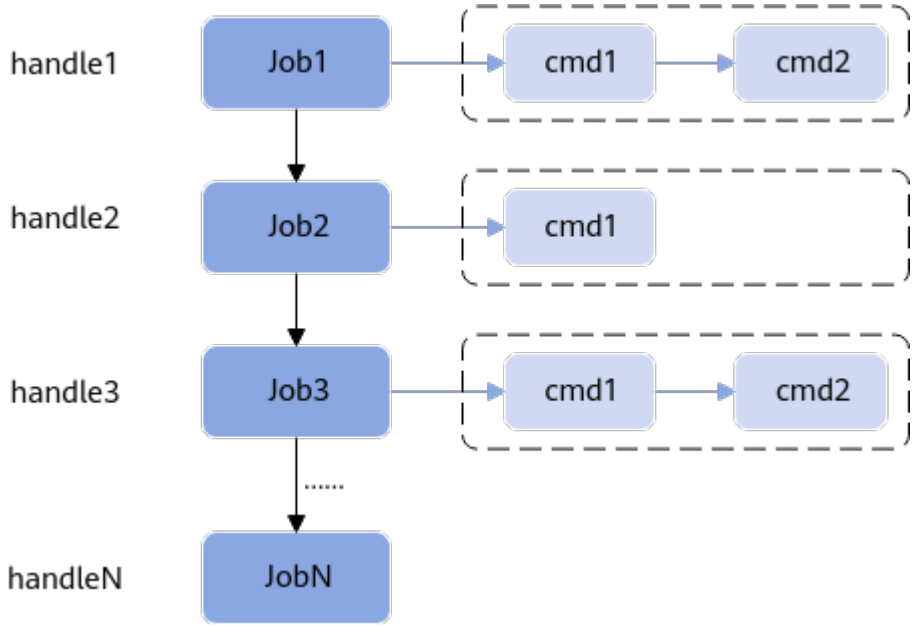
返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.8 TDE图形绘制返回码**

11.18.13.3 hi_tde_begin_job

函数功能

创建TDE任务，获取任务句柄。TDE以任务形式管理TDE命令，一个TDE任务对应一个job，是一个或多个TDE命令的集合；一个TDE任务内的命令是按顺序执行的，如下图所示。成功创建TDE任务、添加TDE命令后，通过**hi_tde_end_job**提交任务。



Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用**hi_tde_open**接口打开TDE设备后才可调用本接口。
- TDE能够同时执行的任务数、命令数最大均不超过200。
- **hi_tde_begin_job**需要和**hi_tde_end_job**配对使用。
- 该接口不支持多进程。

函数原型

hi_s32 hi_tde_begin_job(hi_void)

参数说明

无

返回值说明

- 正数：获取的任务句柄
- 负数：失败，参见**11.18.21.8 TDE图形绘制返回码**

11.18.13.4 hi_tde_end_job

函数功能

提交已创建的TDE任务。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 在调用此接口前应保证调用hi_tde_open打开TDE设备，并且调用hi_tde_begin_job获得了有效的任务句柄。
- 提交任务后，此任务对应的handle会变为无效，再次提交会出现错误码HI_ERR_TDE_INVALID_HANDLE。
- 该接口不支持多进程。

函数原型

hi_s32 hi_tde_end_job(hi_s32 handle, hi_bool is_sync, hi_bool is_block, hi_u32 time_out);

参数说明

参数名	输入/输出	说明
handle	输入	tde任务句柄。
is_sync	输入	同步标志，预留参数，当前固定设置为HI_FALSE。
is_block	输入	阻塞标志。 ● HI_TRUE：阻塞，提交的TDE任务完成或等待指定超时时间后返回。 ● HI_FALSE：非阻塞，直接返回，但需要配合hi_tde_wait_for_done或hi_tde_wait_all_done接口使用。
time_out	输入	超时时间，单位:ms。 当is_block参数设置为阻塞时，可通过本参数设置超时时间；当is_block参数设置为非阻塞时，设置本参数无效。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.8 TDE图形绘制返回码

11.18.13.5 hi_tde_cancel_job

函数功能

在调用**hi_tde_begin_job**创建任务之后、调用hi_tde_end_job提交任务之前，可调用本接口取消对应的TDE任务及任务中的命令。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 在调用此接口前应保证调用**hi_tde_open**打开TDE设备，并且调用**hi_tde_begin_job**获得了有效的任务句柄。
- 调用hi_tde_end_job接口提交的任务，无法调用本接口取消。
- 取消后的任务不再有效，不能再向该任务添加命令操作，也不能提交该任务。
- 该接口不支持多进程。

函数原型

hi_s32 hi_tde_cancel_job(hi_s32 handle);

参数说明

参数名	输入/输出	说明
handle	输入	tde任务句柄。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.8 TDE图形绘制返回码**

11.18.13.6 hi_tde_wait_for_done

函数功能

调用hi_tde_end_job接口、且使用非阻塞方式提交TDE任务后，调用本接口等待指定TDE任务完成。本接口为阻塞接口，会阻塞到指定TDE任务完成后退出本接口。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 在调用此接口前应保证调用**hi_tde_open**打开TDE设备，并且调用**hi_tde_begin_job**获得了有效的任务句柄。
- 该接口不支持多进程。

函数原型

hi_s32 hi_tde_wait_for_done(hi_s32 handle);

参数说明

参数名	输入/输出	说明
handle	输入	tde任务句柄。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.8 TDE图形绘制返回码**

11.18.13.7 hi_tde_wait_all_done

函数功能

调用hi_tde_end_job接口、且使用非阻塞方式提交TDE任务后，等待当前TDE设备上所有任务完成。本接口为阻塞接口，会阻塞到所有TDE任务完成后退出本接口。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 在调用此接口前应保证调用**hi_tde_open**打开TDE设备，并且调用**hi_tde_begin_job**获得了有效的任务句柄。
- 该接口不支持多进程。

函数原型

hi_s32 hi_tde_wait_all_done(hi_void)

参数说明

无

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.8 TDE图形绘制返回码**

11.18.13.8 hi_tde_quick_copy

函数功能

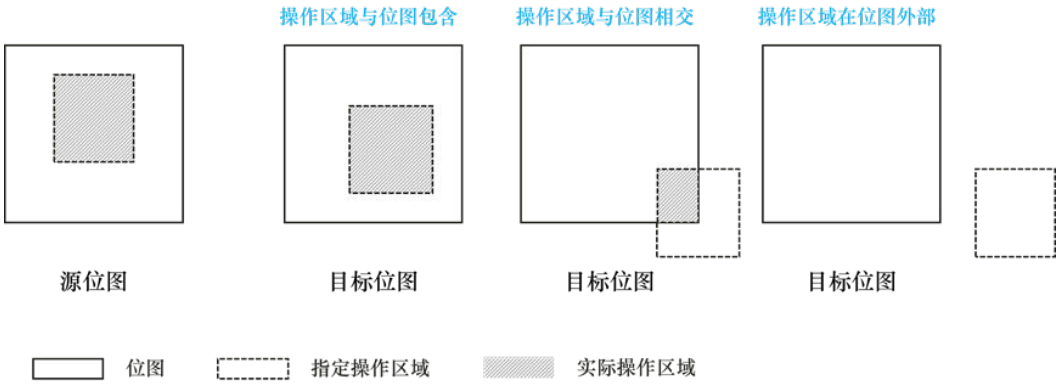
向指定TDE任务中添加快速拷贝命令，实现将源位图的操作区域拷贝到目标位图的操作区域。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 在调用此接口前应保证调用hi_tde_open打开TDE设备，并且调用hi_tde_begin_job获得了有效的任务句柄。
- 区域大小长度及宽度支持的区间为[1,4096]。
- 如果目标位图包含目标位图的操作区域，则无需其他操作。如果操作区域与目标位图相交，则裁剪操作区域，有效的操作区域为灰色相交部分。若指定的操作区域与位图不相交，位图及位图操作区域的关系如下图所示。



- 快速拷贝不支持格式转换，源位图和目标位图格式必须一致，当前仅支持ARGB8888格式。
- 快速拷贝不支持缩放功能，如果源和目的的操作区域尺寸不一致，则按照两者最小公共区域进行拷贝搬移。
- 指定操作区域要和指定的位图有公共区域，否则会返回错误。
- 该接口不支持多进程。

函数原型

hi_s32 hi_tde_quick_copy(hi_s32 handle, const hi_tde_single_src*single_src)

参数说明

参数名	输入/输出	说明
handle	输入	tde任务句柄。
single_src	输入	单源位图区域信息和目标位图区域信息结构体。

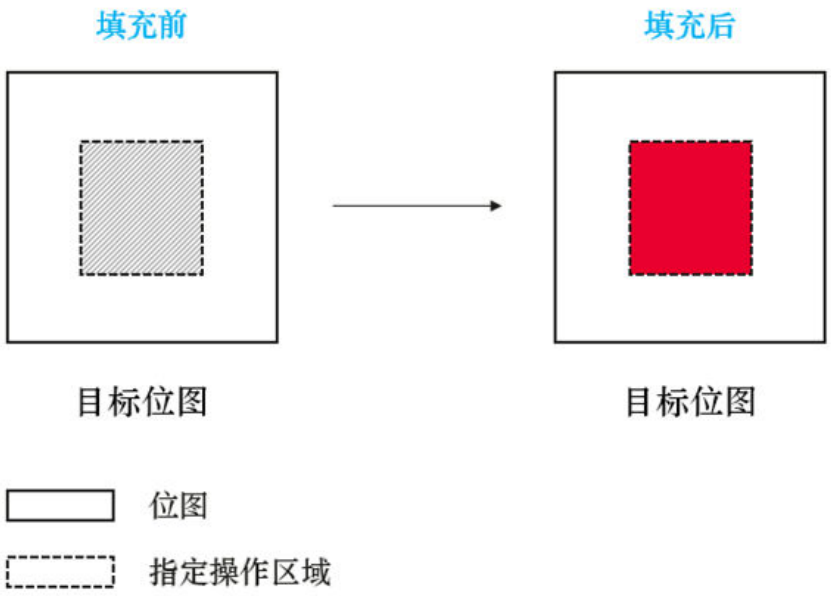
返回值说明

- 0：成功
- 非0：失败，参见11.18.21.8 TDE图形绘制返回码

11.18.13.9 hi_tde_quick_fill

函数功能

向指定TDE任务中添加快速填充命令，实现将指定的颜色数值纯色填充到目标位图的操作区域中，具体如下图所示。



Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 在调用此接口前应保证调用hi_tde_open打开TDE设备，并且调用hi_tde_begin_job获得了有效的任务句柄。
- 区域大小长度及宽度支持的区间为[1,4096]。
- 指定操作区域要和指定的位图有公共区域，否则会返回错误。
- 当前仅支持ARGB8888格式。
- 该接口不支持多进程。

函数原型

hi_s32 hi_tde_quick_fill(hi_s32 handle, const hi_tde_none_src *none_src, hi_u32 fill_data)

参数说明

参数名	输入/输出	说明
handle	输入	tde任务句柄。
none_src	输入	无源位图操作区域位图信息结构体。

参数名	输入/输出	说明
fill_data	输入	填充颜色。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.8 TDE图形绘制返回码**

11.18.13.10 hi_tde_pattern_fill

函数功能

向指定TDE任务中添加模式填充命令，当前仅支持Alpha Blending操作。

模式填充：输入前景位图和背景位图，将前景位图和背景位图的操作区域做附加运算后，平铺在背景位图的操作区域，若前景位图的操作区域大于背景位图的操作区域，则自动进行裁减。

Alpha Blending：通过通过计算前景和背景重叠后的混合颜色，将前景色和背景色结合获得混合后的新颜色。前景色的透明度不限，如果前景色完全透明，混合后的颜色就是背景色，如果前景色完全不透明，混合后的颜色就是前景色。介于(0, 1)中间的透明度，混合后的颜色，需要通过前景色和背景色的加权公式来计算，混合计算必须使用平直Alpha颜色值，以下是blending的前景、背景组合示例：

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 在调用此接口前应保证调用**hi_tde_open**打开TDE设备，并且调用**hi_tde_begin_job**获得了有效的任务句柄。
- 区域大小长度及宽度支持的区间为[1, 4096]；前景位图宽度支持区间为[1, 256]，高度为[1, 4096]。
- 若前景位图操作区域大于目标位图操作区域，则会自动进行裁剪；若前景位图操作区域大于背景位图操作区域，则自动裁减。
- 模式填充背景位图操作区域长度和宽度要和目标位图操作区域长度和宽度保持一致。
- 该操作不能做缩放。
- 操作区域要和指定的位图有公共区域，否则会返回错误。
- 当前仅支持ARGB8888格式。
- 该接口不支持多进程。

函数原型

hi_s32 hi_tde_pattern_fill(hi_s32 handle, const hi_tde_double_src *double_src, const hi_tde_pattern_fill_opt *fill_opt)

参数说明

参数名	输入/输出	说明
handle	输入	tde任务句柄。
double_src	输入	双源位图区域信息和目标位图区域信息结构体。
fill_opt	输入	附加操作，当前只支持Alpha Bending。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.8 TDE图形绘制返回码**

11.18.14 HIFB 叠加图形层管理功能

11.18.14.1 功能及约束说明

HIFB（Hisilicon FrameBuffer）支持的图形层如下表所示，其中G0、G1为超高清图形层，G2图形层为鼠标层，G3、G4图形层为标清图形层。

表 11-43 各图层基本信息说明

图形层	类型	HIFB名称（fb设备）	默认绑定VO设备	支持颜色格式	分辨率	显存
G0	超高清图形层	fb0	0	• ARGB8888 • ARGB1555 • ARGB4444	• 最小：32*32 • 最大（默认）：1920*1080	8100KB
G1	超高清图形层	fb1	1	• ARGB8888 • ARGB1555 • ARGB4444	• 最小：32*32 • 最大（默认）：1920*1080	8100KB

图形层	类型	HIFB名称（fb设备）	默认绑定VO设备	支持颜色格式	分辨率	显存
G2	鼠标层	fb2	0	• ARGB8888 • ARGB1555 • ARGB4444	• 最小：8*8 • 最大（默认）：256*256	256KB
G3	标清图形层	fb3	0	• ARGB8888 • ARGB1555 • ARGB4444 • CLUT4 • CLUT2	• 最小：8*8 • 最大（默认）：720*576	1620KB
G4	标清图形层	fb4	1	• ARGB8888 • ARGB1555 • ARGB4444 • CLUT4 • CLUT2	• 最小：8*8 • 最大（默认）：720*576	1620KB

11.18.14.2 FBIOGET_VSCREENINFO

函数功能

获取Framebuffer屏幕的可变信息，包括分辨率、像素格式等信息。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 高清设备的图形层默认分辨率为1920*1080，鼠标层的默认分辨率为256*256，标清设备的图形层的默认分辨率为720*576，像素格式默认为ARGB8888。
- 使用HIFB接口需先调用**hi_mpi_vo_enable**启用VO设备。
- 该接口不支持多进程，多线程。

函数原型

int ioctl (int fd, FBIOGET_VSCREENINFO, fb_var_screeninfo *var)

参数说明

参数名	输入/输出	说明
fd	输入	Framebuffer设备文件描述符。 应用程序中调用open("/dev/fbx")，会返回当前VO设备对应的Framebuffer设备文件描述符。
FBIOGET_VSCREENINFO	输入	ioctl号。
var	输出	可变信息结构体指针。

返回值说明

- 0：成功
- 非0：失败，返回Linux内核原生错误码，请单击**Link**查看详细说明

11.18.14.3 FBIOPUT_VSCREENINFO

功能描述

设置Framebuffer屏幕的可变信息，包括分辨率、像素格式等信息。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 可见屏幕宽度（xres）与宽度方向的偏移（xoffset）的和不能超过虚拟分辨率宽度（xres_virtual），yres同理。
- 虚拟分辨率的设置需要在当前图层的显存约束范围内（各个图层的最大/最小分辨率及显存限制如**表11-43**所示）。虚拟分辨率所需显存的计算公式如下：

$$M_G = xres_virtual \times yres_virtual \times (bits_per_pixel / 8)$$

- 必须保证实际分辨率与偏移的和在虚拟分辨率范围内，否则系统会自动调整实际分辨率的大小让其在虚拟分辨率范围内。
- 使用HIFB接口需先调用**hi_mpi_vo_enable**启用VO设备。
- 该接口不支持多进程，多线程。

函数原型

int ioctl (int fd, FBIOPUT_VSCREENINFO, fb_var_screeninfo *var)

参数说明

参数名	输入/输出	说明
fd	输入	Framebuffer设备文件描述符。 应用程序中调用open("/dev/fbx")，会返回当前VO设备对应的Framebuffer设备文件描述符。
FBIOPUT_VSCREENINFO	输入	ioctl号。
var	输入	可变信息结构体指针。

返回值说明

- 0：成功
- 非0：失败

11.18.14.4 FBIOGET_FSCREENINFO

功能描述

获取Framebuffer的固定信息，包括显存用户态地址、显存大小等信息。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 使用HIFB接口需先调用**hi_mpi_vo_enable**启用VO设备。
- 显存通过内核mmap函数获取用户态虚拟地址；注意该用户态虚拟地址无法被加速器访问，若需配置给加速器的用户态地址，则需使用**fb_fix_screeninfo**中mmio_start参数。
- 无需显示图形层时，用户需调用内核munmap接口释放用户态虚拟地址，否则会导致该图形层不可用，且驱动无法退出。
- 该接口不支持多进程、多线程。

函数原型

int ioctl (int fd, FBIOGET_FSCREENINFO, fb_fix_screeninfo *fix)

参数说明

参数名	输入/输出	说明
fd	输入	Framebuffer设备文件描述符。 应用程序中调用open("/dev/fbx")，会返回当前VO设备对应的Framebuffer设备文件描述符。
FBIOGET_FSCREENINFO	输入	ioctl号。
fix	输出	固定信息结构体指针。

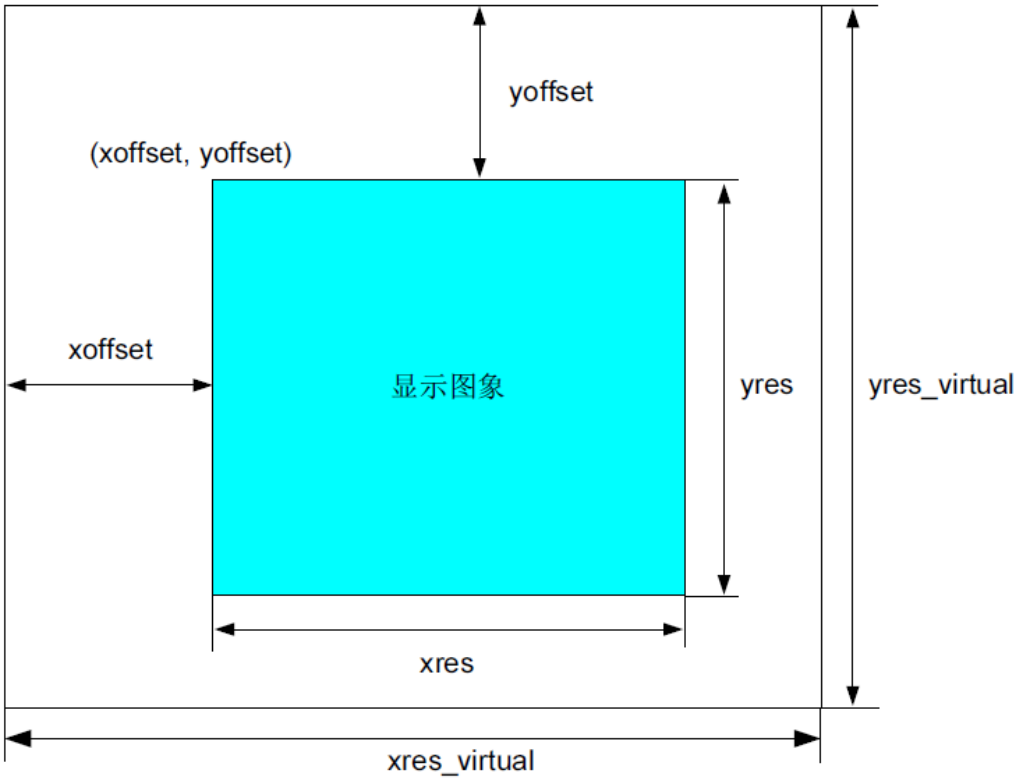
返回值说明

- 0：成功
- 非0：失败

11.18.14.5 FBIOPAN_DISPLAY

功能描述

设置从虚拟分辨率中的不同偏移处开始显示，实际分辨率不变，如下图所示，（xres_virtual, yres_virtual）是虚拟分辨率，（xres, yres）是实际显示的分辨率，(xoffset, yoffset)是显示的偏移。



Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 使用HIFB接口需先调用hi_mpi_vo_enable启用VO设备。
- 必须保证实际分辨率与偏移的和在虚拟分辨率范围内，否则设置不成功。
- 该接口不支持多进程、多线程。

函数原型

int ioctl (int fd, FBIOPAN_DISPLAY, fb_var_screeninfo *var)

参数说明

参数名	输入/输出	说明
fd	输入	Framebuffer设备文件描述符。 应用程序中调用open("/dev/fbx")，会返回当前VO设备对应的Framebuffer设备文件描述符。
FBIOPAN_DISPLAY	输入	ioctl号。
var	输入	可变信息结构体指针。

返回值说明

- 0：成功
- 非0：失败

11.18.14.6 FBIOGET_CMAP

功能描述

获取颜色表信息。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 使用HIFB接口需先调用hi_mpi_vo_enable启用VO设备。
- 只有G3和G4图形层才支持颜色表格式，支持格式为CLUT4/CLUT2。
- 该接口不支持多进程、多线程。

函数原型

int ioctl (int fd, FBIOGET_CMAP, fb_cmap *cmap)

参数说明

参数名	输入/输出	说明
fd	输入	Framebuffer设备文件描述符。 应用程序中调用open("/dev/fbx")，会返回当前VO设备对应的Framebuffer设备文件描述符。
FBIOGET_CMAP	输入	ioctl号。
cmap	输出	颜色表结构体指针。

返回值说明

- 0：成功
- 非0：失败

11.18.14.7 FBIOPUT_CMAP

功能描述

设置颜色表信息。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 使用HIFB接口需先调用**hi_mpi_vo_enable**启用VO设备。
- 只有G3和G4图形层才支持颜色表格式，支持格式为CLUT4/CLUT2。
- 该接口不支持多进程、多线程。

函数原型

int ioctl (int fd, FBIOPUT_CMAP, fb_cmap *cmap)

参数说明

参数名	输入/输出	说明
fd	输入	Framebuffer设备文件描述符。 应用程序中调用open("/dev/fbx")，会返回当前VO设备对应的Framebuffer设备文件描述符。
FBIOPUT_CMAP	输入	ioctl号。
cmap	输入	颜色表结构体指针。

返回值说明

- 0：成功
- 非0：失败

11.18.14.8 FBIOGET_SCREEN_ORIGIN_HIFB

功能描述

获取叠加层在屏幕上显示的起始点坐标。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 使用HIFB接口需先调用**hi_mpi_vo_enable**启用VO设备。
- 该接口不支持多进程、多线程。

函数原型

int ioctl (int fd, FBIOGET_SCREEN_ORIGIN_HIFB, hi_fb_point *point)

参数说明

参数名	输入/输出	说明
fd	输入	Framebuffer设备文件描述符。 应用程序中调用open("/dev/fbx")，会返回当前VO设备对应的Framebuffer设备文件描述符。
FBIOGET_SCREEN_ORIGIN_HIFB	输入	ioctl号。
point	输出	起始点坐标结构体指针。

返回值说明

- 0：成功
- 非0：失败

11.18.14.9 FBIOPUT_SCREEN_ORIGIN_HIFB

功能描述

设置叠加层在屏幕上显示的起始点坐标。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 使用HIFB接口需先调用hi_mpi_vo_enable启用VO设备。
- 起始点坐标必须非负。
- 该接口不支持多进程、多线程。

函数原型

int ioctl (int fd, FBIOPUT_SCREEN_ORIGIN_HIFB, hi_fb_point *point)

参数说明

参数名	输入/输出	说明
fd	输入	Framebuffer设备文件描述符。 应用程序中调用open("/dev/fbx")，会返回当前VO设备对应的Framebuffer设备文件描述符。
FBIOPUT_SCREEN_ORIGIN_HIFB	输入	ioctl号。
point	输入	坐标结构体指针。

返回值说明

- 0：成功
- 非0：失败

11.18.14.10 FBIOGET_SHOW_HIFB

功能描述

获取当前叠加层的显示状态。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 使用HIFB接口需先调用hi_mpi_vo_enable启用VO设备。
- 该接口不支持多进程、多线程。

函数原型

int ioctl (int fd, FBIOGET_SHOW_HIFB, hi_bool *bshow)

参数说明

参数名	输入/输出	说明
fd	输入	Framebuffer设备文件描述符。 应用程序中调用open("/dev/fbx")，会返回当前VO设备对应的Framebuffer设备文件描述符。
FBIOGET_SHOW_HIFB	输入	ioctl号。
bshow	输出	显示状态指针。 ● HI_TRUE：代表叠加层当前状态为显示。 ● HI_FALSE：代表叠加层当前状态为隐藏。

返回值说明

- 0：成功
- 非0：失败

11.18.14.11 FBIOPUT_SHOW_HIFB

功能描述

显示或隐藏该叠加层。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 使用HIFB接口需先调用**hi_mpi_vo_enable**启用VO设备。
- 该接口不支持多进程、多线程。

函数原型

int ioctl (int fd, FBIOPUT_SHOW_HIFB, hi_bool *bshow)

参数说明

参数名	输入/输出	说明
fd	输入	Framebuffer设备文件描述符。 应用程序中调用open("/dev/fbx")，会返回当前VO设备对应的Framebuffer设备文件描述符。

参数名	输入/输出	说明
FBIOPUT_SHOW_HIFB	输入	ioctl号。
bshow	输入	显示状态指针。 ● HI_TRUE：代表设置叠加层状态为显示。 ● HI_FALSE：代表设置叠加层状态为隐藏。

返回值说明

- 0：成功
- 非0：失败

11.18.14.12 FBIOGET_ALPHA_HIFB

功能描述

获取叠加层的Alpha信息。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 使用HIFB接口需先调用**hi_mpi_vo_enable**启用VO设备。
- 该接口不支持多进程、多线程。

函数原型

int ioctl (int fd, FBIOGET_ALPHA_HIFB, hi_fb_alpha *alpha)

参数说明

参数名	输入/输出	说明
fd	输入	Framebuffer设备文件描述符。 应用程序中调用open("/dev/fbx")，会返回当前VO设备对应的Framebuffer设备文件描述符。
FBIOGET_ALPHA_HIFB	输入	ioctl号。
alpha	输出	Alpha属性指针。

返回值说明

- 0：成功
- 非0：失败

11.18.14.13 FBIOPUT_ALPHA_HIFB

功能描述

设置叠加层的Alpha信息。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 使用HIFB接口需先调用**hi_mpi_vo_enable**启用VO设备。
- 该接口不支持多进程、多线程。

函数原型

int ioctl (int fd, FBIOPUT_ALPHA_HIFB, hi_fb_alpha *alpha)

参数说明

参数名	输入/输出	说明
fd	输入	Framebuffer设备文件描述符。 应用程序中调用open("/dev/fbx")，会返回当前VO设备对应的Framebuffer设备文件描述符。
FBIOPUT_ALPHA_HIFB	输入	ioctl号。
alpha	输出	Alpha属性指针。

返回值说明

- 0：成功
- 非0：失败

11.18.15 HDMI 外设

11.18.15.1 hi_mpi_hdmi_init

函数功能

初始化HDMI。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 该函数接口需要与**hi_mpi_hdmi_deinit**接口成对调用，即本接口被调用后，需要调用**hi_mpi_hdmi_deinit**接口，才允许再次调用**hi_mpi_hdmi_init**接口。
- 该接口暂不支持多进程。

函数原型

hi_s32 hi_mpi_hdmi_init(void)

参数说明

无

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.9 HDMI外设返回码**

11.18.15.2 hi_mpi_hdmi_deinit

函数功能

去初始化HDMI。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 该函数接口需要与**hi_mpi_hdmi_init**接口成对调用，即**hi_mpi_hdmi_init**函数被调用后，才允许调用**hi_mpi_hdmi_deinit**接口，且保证二者调用次数相同。
- 该接口暂不支持多进程。

函数原型

hi_s32 hi_mpi_hdmi_deinit(void)

参数说明

无

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.9 HDMI外设返回码**

11.18.15.3 hi_mpi_hdmi_open

函数功能

打开指定的HDMI接口。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- hi_mpi_hdmi_open之前必须先调用hi_mpi_hdmi_init。
- 重复打开HDMI返回成功。
- 该接口暂时不支持多进程场景。
- 该函数接口需要与hi_mpi_hdmi_close接口成对调用，即hi_mpi_hdmi_open函数被调用后，退出时需要调用hi_mpi_hdmi_close接口，且需要保持两个接口入参中的hi_hdmi_id参数值一致。

函数原型

hi_s32 hi_mpi_hdmi_open(hi_hdmi_id hdmi)

参数说明

参数名	输入/输出	说明
hdmi	输入	HDMI接口号。 取值范围：[0, 2)。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.9 HDMI外设返回码

11.18.15.4 hi_mpi_hdmi_close

函数功能

关闭指定的HDMI接口。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用hi_mpi_hdmi_close接口之前必须先调用hi_mpi_hdmi_open接口，且需要保持两个接口入参中的hi_hdmi_id参数值一致。
- 该接口暂时不支持多进程场景。

- 该函数接口需要与hi_mpi_hdmi_open接口成对调用，即hi_mpi_hdmi_open函数被调用后，才允许调用hi_mpi_hdmi_close接口，且保证二者调用次数相同。

函数原型

hi_s32 hi_mpi_hdmi_close(hi_hdmi_id hdmi);

参数说明

参数名	输入/输出	说明
hdmi	输入	HDMI接口号。 取值范围：[0, 2)。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.9 HDMI外设返回码

11.18.15.5 hi_mpi_hdmi_set_attr

函数功能

设置HDMI属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 在调用hi_mpi_hdmi_open接口打开HDMI之后、调用hi_mpi_hdmi_start接口启用HDMI之前，调用本接口设置HDMI属性，但需保证hi_mpi_hdmi_open接口与本接口中的hi_hdmi_id参数值保持一致。

  若已调用hi_mpi_hdmi_start启动HDMI，则应先调用hi_mpi_hdmi_stop停止HDMI，设置属性后再重新启动。不遵循该流程使用的行为是未定义的，暂不支持。
- 若只设置部分属性，设置前应先调用hi_mpi_hdmi_get_attr接口获取属性，赋值该部分属性后再设置。
- 该接口暂时不支持多进程场景。

函数原型

hi_s32 hi_mpi_hdmi_set_attr(hi_hdmi_id hdmi, const hi_hdmi_attr *attr)

参数说明

参数名	输入/输出	说明
hdmi	输入	HDMI接口号。 取值范围：[0, 2)。
attr	输入	HDMI属性结构体指针。部分属性暂时不支持，见数据类型hi_hdmi_attr说明。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.9 HDMI外设返回码

11.18.15.6 hi_mpi_hdmi_get_attr

函数功能

获取HDMI属性。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 在调用hi_mpi_hdmi_open接口打开HDMI之后，调用本接口获取HDMI属性，但需保证hi_mpi_hdmi_open接口与本接口中的hi_hdmi_id参数值保持一致。
- 该接口暂时不支持多进程场景。

函数原型

hi_s32 hi_mpi_hdmi_get_attr(hi_hdmi_id hdmi, hi_hdmi_attr *attr)

参数说明

参数名	输入/输出	说明
hdmi	输入	HDMI接口号。 取值范围：[0, 2)。
attr	输入	HDMI属性结构体指针。部分属性暂时不支持，见数据类型hi_hdmi_attr说明。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.9 HDMI外设返回码

11.18.15.7 hi_mpi_hdmi_start

函数功能

启动指定的HDMI，使能其送显能力。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- hi_mpi_hdmi_start接口被调用前必须先调用hi_mpi_hdmi_open接口来打开对应的HDMI，且需要保持两个接口入参中的hi_hdmi_id参数值一致
- 该接口暂时不支持多进程场景。

函数原型

hi_s32 hi_mpi_hdmi_start(hi_hdmi_id hdmi)

参数说明

参数名	输入/输出	说明
hdmi	输入	HDMI接口号。 取值范围：[0, 2)。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.9 HDMI外设返回码

11.18.15.8 hi_mpi_hdmi_stop

函数功能

停止指定的HDMI接口，不使能其送显能力。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- hi_mpi_hdmi_stop接口被调用前必须先调用hi_mpi_hdmi_open接口来打开对应的HDMI，且需要保持两个接口入参中的hi_hdmi_id参数值一致
- 该接口暂时不支持多进程场景。

函数原型

hi_s32 hi_mpi_hdmi_stop(hi_hdmi_id hdmi)

参数说明

参数名	输入/输出	说明
hdmi	输入	HDMI接口号。 取值范围：[0, 2)。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.9 HDMI外设返回码**

11.18.15.9 hi_mpi_hdmi_set_infoframe

函数功能

设置信息帧。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 调用该接口前，前必须先打开HDMI（调用**hi_mpi_hdmi_open**），再调用**hi_mpi_hdmi_set_attr**接口设置基本属性，最后调用本接口补充设置输出颜色空间、宽高比、时序属性（若无需指定上述参数，使用默认值，可不调用本接口），且保持三个接口入参中的**hi_hdmi_id**值一致。
- 该API 目前只支持HI_INFOFRAME_TYPE_AUDIO和HI_INFOFRAME_TYPE_AVI信息帧。
- HI_INFOFRAME_TYPE_AVI信息帧中只支持修改**hi_hdmi_infoframe**结构体的timing_mode参数（必须与VO输入的具体时序保持一致）、color_space参数（用户送图时需要设置为YUV444）、aspect_ratio参数（仅支持设置为16比9、4比3）为非默认值，
- HI_INFOFRAME_TYPE_AUDIO信息帧中均需要设置为默认参数，暂不支持修改，详细赋值要求请见**hi_hdmi_audio_infoframe**数据类型描述。
- 该接口的部分属性不支持设置，详情请见**hi_hdmi_infoframe**结构体及其成员说明。
- 调用该接口设置信息帧相关属性后，会生效成attr属性。
- 该接口属于高级接口，一般不需要调用。若用户使用，则应根据已设置的音视频相关属性（如enVideoFmt），及遵从《High-Definition Multimedia Interface Specification Version 1.4b》、《High-Definition Multimedia Interface SpecificationVersion 2.0》、《CEA-861-D 》与《CEA-861-F》标准基础上设置信息帧，不依据音视频属性遵从标准发送信息帧的行为是未定义的，暂不支持。
- 该接口暂时不支持多进程场景。

函数原型

hi_s32 hi_mpi_hdmi_set_infoframe(hi_hdmi_id hdmi, const hi_hdmi_infoframe *infoframe)

参数说明

参数名	输入/输出	说明
hdmi	输入	HDMI接口号。 取值范围：[0, 2)。
infoframe	输入	HDMI信息帧结构体指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.9 HDMI外设返回码**

11.18.15.10 hi_mpi_hdmi_get_sink_capability

函数功能

获取指定端口的能力集。

约束说明

若在打开HDMI设备（通过**hi_mpi_hdmi_open**接口打开）之后调用本接口，可获取设备是否连接的信息；若在打开HDMI设备（通过**hi_mpi_hdmi_open**接口打开）且插入线缆之后调用本接口才可以获取Sink端能力集的所有信息。

函数原型

hi_s32 hi_mpi_hdmi_get_sink_capability(hi_hdmi_id hdmi, hi_hdmi_sink_capability *capability)

参数说明

参数名	输入/输出	说明
hdmi	输入	HDMI接口号。 取值范围：[0, 2)。
capability	输入	HDMI Sink端能力集结构体指针。

返回值说明

- 0：成功

- 非0：失败，参见**11.18.21.9 HDMI外设返回码**

11.18.15.11 hi_mpi_hdmi_register_callback

函数功能

用户态注册HDMI事件回调函数。

约束说明

- 调用前必须先调用**hi_mpi_hdmi_open**；
- 建议用户注册HDMI事件回调函数；

  例如，当产生热插拔事件时，可以通过注册的回调函数读取热插拔后产生的能力集信息为依据更改HDMI的属性，然后重新启动HDMI，使HDMI的属性适配新插入的对端显示器/电视。

  若用户不注册事件回调函数，则当事件产生时，HDMI内部会采取默认的处理方式，默认支持原屏幕热插拔；若用户注册事件回调函数，则直接按用户注册的函数处理，系统内部的默认热插拔函数会失效，需由用户自行保证对热插拔的支持。
- 回调函数注册只支持插入和拔出两个事件；

  一般在插入事件处理函数中使用**hi_mpi_hdmi_get_sink_capability**获取sink端能力集。
- 回调函数注册需要由用户保证入参中的函数指针不存在内存泄漏、函数本身运行挂死等问题；
- 若本接口被调用时，HDMI显示设备已连接，则第一次拔出事件不响应，第二次才开始响应。

函数原型

hi_s32 hi_mpi_hdmi_register_callback(hi_hdmi_id hdmi, const hi_hdmi_callback_func *callback_func)

参数说明

参数名	输入/输出	说明
hdmi	输入	HDMI接口号。 取值范围：[0, 2)。
callback_func	输入	HDMI回调函数结构体。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.9 HDMI外设返回码**

11.18.16 VPC 图像处理功能

11.18.16.1 功能说明

Atlas 200/300/500 推理产品上，该功能为预留功能，暂不支持。

功能说明

VPC（Vision Preprocessing Core）各功能的介绍、各版本的功能支持度如下表所示。

功能	描述	Atlas 推理系列产品
抠图	从输入图片中抠出需要用的图片区域，支持一图多框和多图多框。	√
缩放	● 针对不同分辨率的图像，支持8K缩放、非8K缩放。 ● 支持单图裁剪缩放（支持非压缩格式）、一图多框裁剪缩放（支持非压缩格式）。 ● 其它缩放方式，如：原图缩放、等比例缩放（缩放前后图片的宽高比例相同）。	√
叠加	从输入图片中抠出来的图，对抠出的图进行缩放后，放在用户输出图片的指定区域，输出图片可以是空白图片（由用户申请的空输出内存产生的），也可以是已有图片（由用户申请输出内存后将已有图片读入输出内存），只有当输出图片是已有图片时，才表示叠加。	√
拼接	从输入图片中抠多张图片，对抠出的图进行缩放后，放到输出图片的指定区域。	√
图像金字塔	对原图进行多次高斯滤波之后向下采样产生不同尺寸的图像。	√
直方图统计	统计图像每个通道（RGB/YUV）的像素值分布。	√
色彩重映射	根据配置信息将图片从原图映射为另一张图。	√
边界填充	对图像进行边界填充。	√
格式转换	支持RGB格式、YUV格式之间的格式转换。	√
图像灰度化	将彩色图像转化为灰度图像。需注意，输入为灰度图像、输出只能为灰度图像。 实现图像灰度化的操作是输出YUV400格式的输出图片。	√

功能	描述	Atlas 推理系列产品
环视拼接	主要用于车载场景生成全景图。 使用该功能时，输入图片为4张，一般为前、左、后、右四个方向的图像，经过畸变矫正、增益补偿、图像融合，最终将4张图片融合成一张全景图片。其中，增益补偿是可选步骤，可在调用参数设置接口时选择是否进行增益补偿，通过增益补偿可平衡不同摄像头光照，达到更好的效果；图像融合当前采用的是加权融合方式，即像素加权平均法。	x
Remap变换	根据像素位置LUT对输入图像进行几何形变，典型的用途包括：镜头畸变校正、仿射变换、透视变换。 形变方式可以用下述公式描述：dst(x,y)=src(LUT(x,y))。 其中，dst(x,y)为输出图像在坐标(x,y)处的像素值，像素位置LUT(x,y)是输出图像在(x,y)处的像素对应输入图像中的横、纵坐标值，src(LUT(x,y))为输入图像在坐标LUT(x,y)处的像素值。通用的像素位置LUT可通过用户提供的map1（横坐标映射矩阵）和map2（纵坐标映射矩阵）生成，仿射变换或透视变换的像素位置LUT也可通过用户提供的3个点对或4个点对信息计算生成。	x
滤波	对输入图片做滤波处理，当前可支持中值滤波/腐蚀/膨胀/高斯滤波/均值滤波/卷积滤波。	x
旋转	对输入图片做固定角度的旋转，支持90度/180度/270度。	x
马赛克	对输入图片做马赛克处理。	x
覆盖	对输入图片做局部覆盖操作。	x
画线	对输入图片做画线处理。	x
添加水印	对输入图片做添加水印的处理。	x

功能示意图(Atlas 推理系列产品)

图 11-14 VPC 功能示意图（抠图+缩放+叠加）

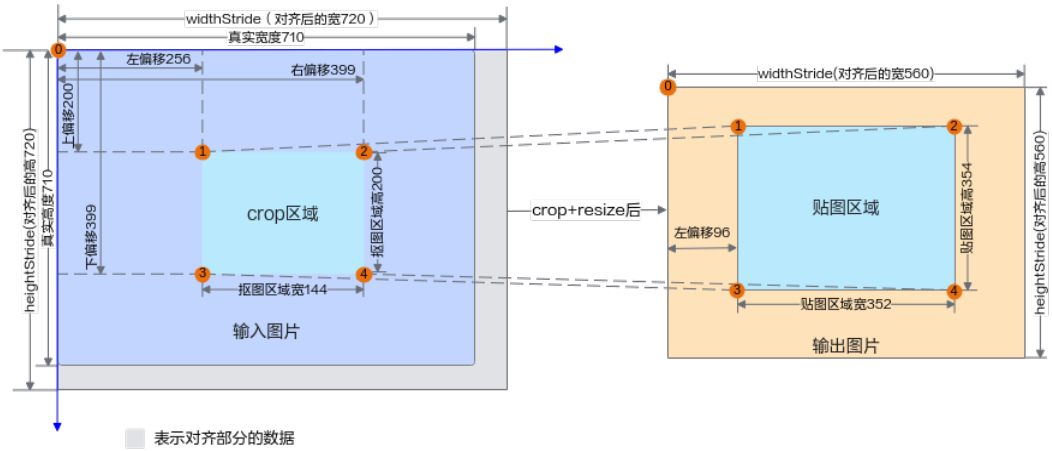


图 11-15 VPC 功能示意图（拼接）

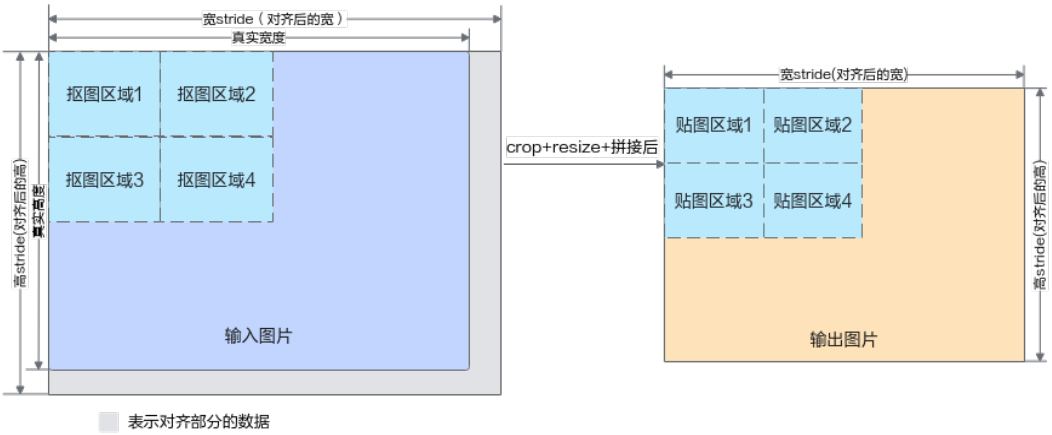


图 11-16 等比例缩放（贴图区域在输出图片的中心位置），即缩放前后图片的宽高比例相同

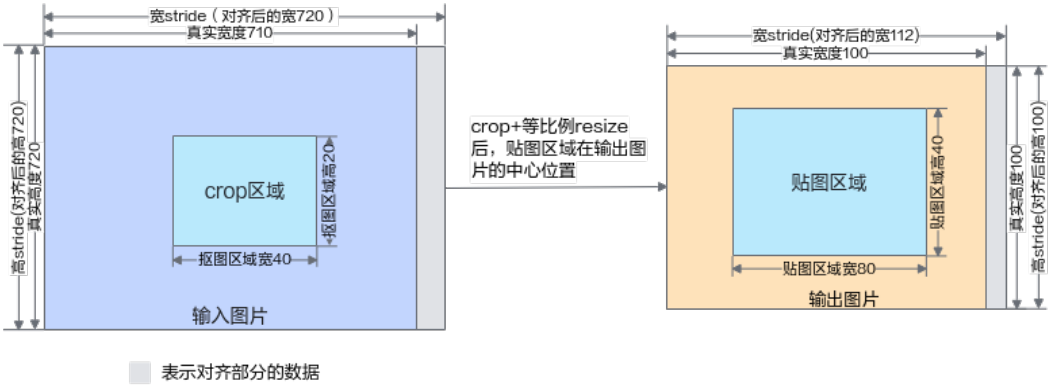


图 11-17 等比例缩放（贴图区域在输出图片的上下居中位置）

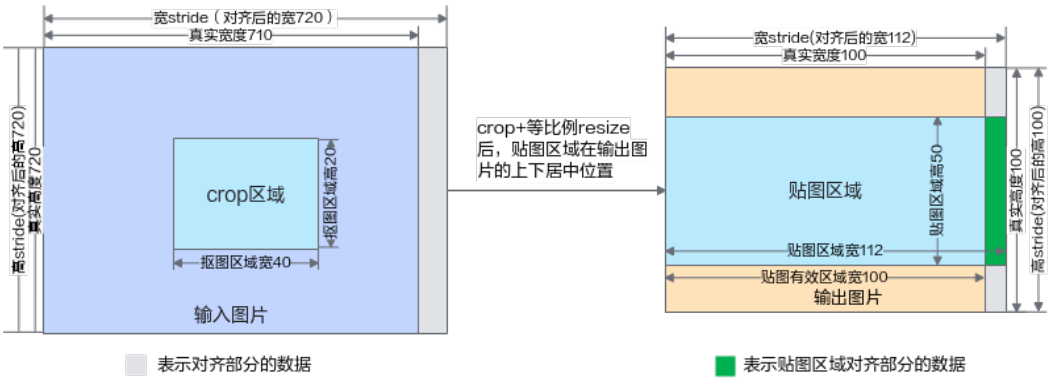
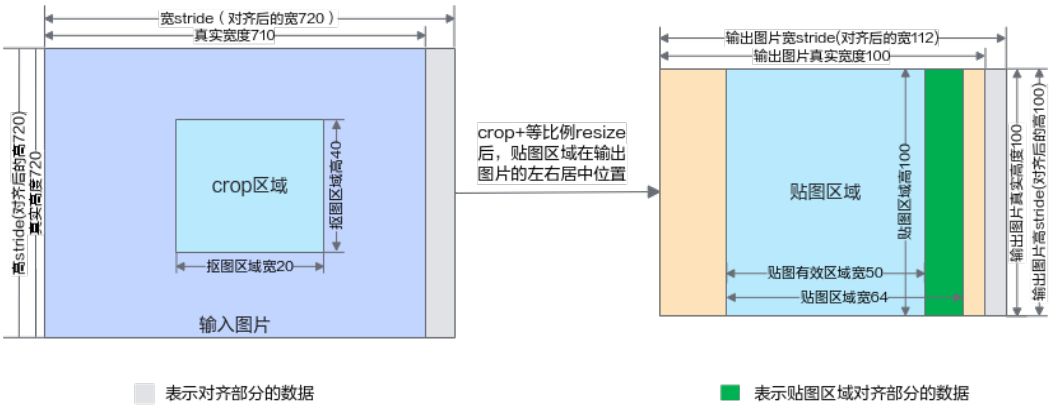


图 11-18 等比例缩放（贴图区域在输出图片的左右居中位置）



参考说明

RGB、YUV格式图像的各分量排布示意图。示例：SP图像以YUV420SP为例，Packed和RGB图像以ARGB图像为例。

格式	分辨率	宽stride	高stride	buffer大小					
yuv420sp	4*4	4	4	24					
内存排列									
y11	y12	y13	y14						
y21	y22	y23	y24						
y31	y32	y33	y34						
y41	y42	y43	y44						
u11	v11	u13	v13						
u31	v31	u33	v33						
格式	分辨率	宽stride	高stride	buffer大小					
yuv420sp	4*4	6	6	54					
内存排列	x为无效数据								
y11	y12	y13	y14	x	x				
y21	y22	y23	y24	x	x				
y31	y32	y33	y34	x	x				
y41	y42	y43	y44	x	x				
x	x	x	x	x	x				
x	x	x	x	x	x				
u11	v11	u13	v13	x	x				
u31	v31	u33	v33	x	x				
x	x	x	x	x	x				
格式	分辨率	宽stride	高stride	buffer大小					
argb	2*2	10	4	40					
内存排列	x为无效数据								
a11	r11	g11	b11	a12	r12	g12	b12	x	x
a21	r21	g21	b21	a22	r22	g22	b22	x	x
x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x

11.18.16.2 约束说明

Atlas 200/300/500 推理产品上，该功能为预留功能，暂不支持。

图片分辨率约束

- 输入图片分辨率

  VPC处理图片时，通用的输入分辨率范围为10*6~8192*8192，其中部分接口的分辨率限制可能不同，请以具体接口内的说明为准，请参见**11.18.16 VPC图像处理功能**下各接口的说明。其中，当输入图片格式为YUV440SP、YUV440P时，输入图片的宽最大值为4096。

- 输出图片分辨率

版本	分辨率范围
Atlas 推理系列产品	10*6~4096*4096

内存申请/释放接口说明

输入、输出内存大小要求请参见**图片格式、宽高对齐、内存约束**中的计算公式。

版本	内存申请/释放接口
Atlas 推理系列产品	VPC在处理图片时，需调用hi_mpi_dvpp_malloc接口申请Device上的输入、输出内存，调用hi_mpi_dvpp_free接口释放输入、输出内存，这部分内存的生命周期由用户自行管理。

图片格式、宽高对齐、内存约束

说明

在调用接口实现VPC功能时：

- 图片格式的定义请参见hi_pixel_format，宽stride、高stride等概念请参见11.18.2 基本概念。
- 不同版本中，宽stride、高stride的最小值、最大值不同，如下：
  - Atlas 推理系列产品上，宽stride最小32、最大16384（16384=4096*4，宽是4096的argb格式的图像，1个像素占用4个字节，一行像素就占用4096*4，即宽stride）；高stride最小6、最大16384。
- 各接口对图片格式的要求可能不同，请参见11.18.16 VPC图像处理功能下各接口的说明。

表 11-44 输入图片格式、宽高对齐、内存大小约束

版本	图片格式	图片宽、高对齐要求	图片宽stride、高stride对齐要求 Atlas 推理系列产品	内存大小要求（单位Byte）
通用格式，各版本都支持	YUV400 8bit	无对齐要求	宽stride为宽16对齐后的值； 高stride无对齐要求，与高相同即可。	宽stride * 高stride
	YUV420SP NV12 8bit YUV420SP NV21 8bit	宽2对齐 高2对齐	宽stride为宽16对齐后的值； 高stride为高2对齐后的值。	宽stride * 高stride * 3/2
	YUV422SP 8bit YVU422SP 8bit	宽2对齐 高无对齐要求	宽stride为宽16对齐后的值； 高stride无对齐要求，与高相同即可。	宽stride * 高stride * 2
	YUV444SP 8bit YVU444SP 8bit	无对齐要求	宽stride为宽16对齐后的值； 高stride无对齐要求，与高相同即可。	宽stride * 高stride * 3

版本	图片格式	图片宽、高对齐要求	图片宽stride、高stride对齐要求 Atlas 推理系列产品	内存大小要求（单位Byte）
	YUV422Packed YUYV 8bit YUV422Packed UYVY 8bit YUV422Packed YVYU 8bit YUV422Packed VYUY 8bit	宽2对齐 高无对齐要求	宽stride为宽16对齐后、再乘以2的值； 高stride无对齐要求，与高相同即可。	宽stride * 高stride
	YUV444Packed 8bit RGB888 BGR888	无对齐要求	宽stride为宽16对齐后、再乘以3的值； 高stride无对齐要求，与高相同即可。	宽stride * 高stride
	ARGB8888 ABGR8888 RGBA8888 BGRA8888	无对齐要求	宽stride为宽16对齐后、再乘以4的值； 高stride无对齐要求，与高相同即可。	宽stride * 高stride
	YUV440SP 8bit YVU440SP 8bit	宽无对齐要求，但宽≤4096 高2对齐	宽stride为宽16对齐后的值； 高stride为高2对齐后的值。	宽stride * 高stride * 2

表 11-45 输出图片格式、宽高对齐、内存大小约束

-	图片格式	图片宽、高对齐要求	图片宽stride、高stride对齐要求 Atlas 推理系列产品	内存大小要求（单位Byte）
通用格式，各版本都支持	YUV400 8bit	无对齐要求	宽stride为宽16对齐后的值； 高stride无对齐要求，与高相同即可。	宽stride * 高stride

-	图片格式	图片宽、高对齐要求	图片宽stride、高stride对齐要求 Atlas 推理系列产品	内存大小要求（单位Byte）
	YUV420SP NV12 8bit YUV420SP NV21 8bit	宽2对齐 高2对齐	宽stride为宽16对齐后的值； 高stride为高2对齐后的值。	宽stride * 高stride * 3/2
	YUV422SP 8bit YVU422SP 8bit	宽2对齐 高无对齐要求	宽stride为宽16对齐后的值； 高stride无对齐要求，与高相同即可。	宽stride * 高stride * 2
	YUV444Packed 8bit RGB888 BGR888	无对齐要求	宽stride为宽16对齐后、再乘以3的值； 高stride无对齐要求，与高相同即可。	宽stride * 高stride
	ARGB8888 ABGR8888 RGBA8888 BGRA8888	无对齐要求	宽stride为宽16对齐后、再乘以4的值； 高stride无对齐要求，与高相同即可。	宽stride * 高stride

抠图、贴图约束

- 抠图区域不超出输入图片区域。
- 抠图、贴图区域的奇数、偶数限制为：

  输出图片格式为YUV420SP，贴图区域奇数、偶数限制为：左偏移和上偏移为偶数、右偏移和下偏移为奇数。

  输出图片格式为YUV422SP，贴图区域奇数、偶数限制为：左偏移为偶数、右偏移为奇数。

  其它格式，贴图区域没有偏移奇偶数的限制。
- 贴图区域不超出输出图片区域，最大贴图个数256个。

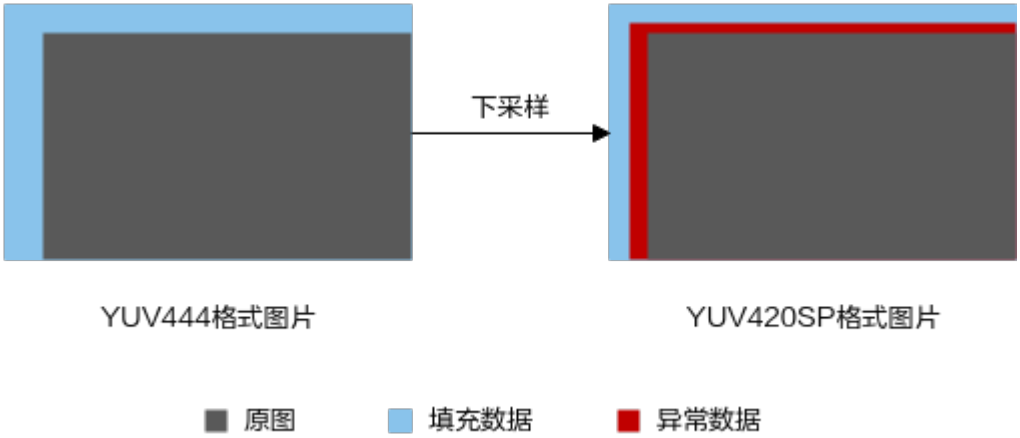
  在Atlas 推理系列产品上，由于16对齐的要求，还存在以下约束：
  - 贴图区域相对输出图片的左偏移16对齐。贴图时可直接放置在输出图片的最左侧，即相对输出图片的左偏移为0，0也满足16对齐的要求。
  - 输出图片的贴图宽度建议16对齐，如果不是16对齐，会多写一段无效数据使其16对齐，如**图11-17**、**图11-18**所示，贴图区域旁边的绿色框就表示无效数据。
  - 等比例缩放场景下，由于贴图左偏移16对齐的限制，如果直接调用**hi_mpi_vpc_crop_resize_paste**接口实现缩放功能，贴图的位置可能不在输

出图片的正中心位置，此时可通过抠图、缩放、填充几个功能配合使用（调用hi_mpi_vpc_crop_resize_make_border或hi_mpi_vpc_batch_crop_resize_make_border接口），实现等比例缩放。

YUV 格式图像下采样约束

VPC在处理图片时，会根据输入或输出图片格式，将输入图片格式转换为YUV444或RGB用于内部处理，YUV444或RGB没有宽高奇偶数的限制，但当输出图片格式为YUV420SP或YUV422SP格式时，会进行下采样处理，由于YUV420SP或YUV422SP格式本身的数据排布导致宽高存在奇偶数限制，因此输出图片的边缘可能存在异常数据。

图 11-19 异常效果图片举例



出现异常数据的根因在于，在计算过程中，出现输出图片位置处于奇数起始点时，此时YUV444格式的图片是正确的，但是下采样到YUV420SP格式时，由于奇数行和偶数行是共用同一个uv的，导致图片起始行的Y与上一行的UV组成新的像素，产生异常数据。

精度相关约束（Atlas 推理系列产品）

VPC功能中，贴图区域的宽有16对齐的约束，贴图有效区域的宽不满足16对齐时，贴图区域中会存在一些补边的无效数据，为保证推理精度，贴图有效区域的宽不满足16对齐时，在贴图前用户可先将贴图有效区域的宽缩放成16对齐，或在推理前借助AIPP抠出有效区域，去除无效数据。

11.18.16.3 性能指标说明

Atlas 200/300/500 推理产品上，该功能为预留功能，暂不支持。

性能指标说明

本节仅描述各产品对应的昇腾AI处理器的性能指标参考数据，各产品由于其内存、CPU等配置不同，在性能规格上会存在差异，详细产品的性能规格请查阅昇腾社区的官方说明。

本节以单个昇腾AI处理器为例，描述基本场景性能指标参考，如下表所示，其它分辨率可以等量估算。

下表中的1路对应一个通道，一个通道对应一个线程；各分辨率的含义为：720p指分辨率为1280*720的图片，1080p指分辨率为1920*1080的图片，4K指分辨率为3840*2160的图片。

表 11-46 Atlas 推理系列产品的昇腾 AI 处理器性能指标参考数据

场景举例	总帧率
● 输入图片分辨率：1080p（1920*1080） ● 输出图片分辨率：1080p（1920*1080） ● 输入/输出图片格式：YUV420SP ● n路（1≤n≤12）	n*360fps
● 输入图片分辨率：1080p（1920*1080） ● 输出图片分辨率：1080p（1920*1080） ● 输入/输出图片格式：YUV420SP ● n路（n>12）	4320fps
● 输入图片分辨率：4K图像（3840*2160） ● 输出图片分辨率：4K图像（3840*2160） ● 输入/输出图片格式：YUV420SP ● n路（1≤n≤12）	n*90fps
● 输入图片分辨率：4K图像（3840*2160） ● 输出图片分辨率：4K图像（3840*2160） ● 输入/输出图片格式：YUV420SP ● n路（n>12）	1080fps
● 输入图片分辨率：8K图像（7680*4320） ● 输出图片分辨率：4K图像（3840*2160） ● 输入/输出图片格式：YUV420SP ● n路（1≤n≤12）	n*22.5fps
● 输入图片分辨率：8K图像（7680*4320） ● 输出图片分辨率：4K图像（3840*2160） ● 输入/输出图片格式：YUV420SP ● n路（n>12）	270fps

说明

- VPC处理性能与处理过程中的图像分辨率强相关，以输入图像和输出图像中的最大分辨率作为基准分辨率，基准分辨率越大，处理耗时越久，性能越低。
- 调用VPC批处理接口（接口命名中包含batch，例如hi_mpi_vpc_batch_crop_resize_paste接口）时，由于图像处理单元DVPP（Digital Video Pre-Processing）内部多个VPC硬件单元会并行处理图片任务，因此单路就可以达到最大总帧率。

11.18.16.4 hi_mpi_vpc_create_chn

函数功能

根据设置的通道属性创建图像处理通道，由用户指定通道号。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- Atlas 推理系列产品上，单个Device上的通道号不能超出最大的通道号范围。用户在使用VPC功能或VDEC功能（内部会使用VPC功能）时，都会占用VPC通道号，为防止VPC功能与VDEC功能的通道号冲突，用户在使用VPC功能时，通道号的取值范围为[0,255]；在使用VDEC功能时，VDEC内部使用的VPC通道号范围为[256,511]。
- 如果参数attr为空，会返回错误码HI_ERR_VPC_NULL_PTR。
- 在创建图像处理通道之前必须保证通道未创建（或者已经销毁），否则会直接返回失败。

函数原型

hi_s32 hi_mpi_vpc_create_chn(hi_vpc_chn chn, const hi_vpc_chn_attr *attr)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。 **说明** **对于Atlas 推理系列产品，**昇腾虚拟化实例（Ascend Virtual Instance）场景下，通道总数 =（被分配的VPC硬件单元 / VPC硬件单元总数）* 256，如果通道总数不为整数，则向下取整。 昇腾虚拟化实例（Ascend Virtual Instance）的详细描述，以及各场景下的算力资源规格请参见《昇腾虚拟化实例（AVI）用户指南》。
attr	输入	图片处理通道属性指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.10 VPC图像处理返回码**

参考资源

接口调用流程及示例，参见**5.5.1 VPC图片处理典型功能**。

11.18.16.5 hi_mpi_vpc_destroy_chn

函数功能

销毁创建的图像处理通道。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

通道号不能超出最大的通道号范围。

函数原型

hi_s32 hi_mpi_vpc_destroy_chn(hi_vpc_chn chn)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.10 VPC图像处理返回码**

参考资源

接口调用流程及示例，参见**5.5.1 VPC图片处理典型功能**。

11.18.16.6 hi_mpi_vpc_resize

函数功能

对图像进行按照一定比例进行缩放或者缩放到固定尺寸。异步接口。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

版本	约束

Atlas 推理系列产品	● 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**hi_pixel_format**枚举值中的如下枚举项：
	HI_PIXEL_FORMAT_YUV_400 = 0, // YUV400 8bit
	HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	HI_PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	HI_PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3, // YUV422SP 8bit
	HI_PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4, // YVU422SP 8bit
	HI_PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5, // YUV444SP 8bit
	HI_PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6, // YVU444SP 8bit
	HI_PIXEL_FORMAT_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	HI_PIXEL_FORMAT_UYVY_PACKED_422 = 8, // YUV422Packed UYVY 8bit
	HI_PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 8bit
	HI_PIXEL_FORMAT_VYUY_PACKED_422 = 10, // YUV422Packed VYUY 8bit
	– 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。
	● 关于输出图片分辨率、图片格式、宽高对齐、内存大小约束请参见**11.18.16.2 约束说明**。
	● 当输入、输出图片格式都为ARGB8888、ABGR8888、RGBA8888、BGRA8888格式时，在经过本接口缩放处理后，A分量值会变为0。
	● 设置fx/fy之后，缩放后的图片的分辨率必须在[10*6~4096*4096]范围内，否则返回错误。
	● 缩放可以通过fx/fy或者目标图片宽高两种方式来指定，但是两者均配置以目标尺寸缩放优先。两者不可同时为0。

函数原型

hi_s32 hi_mpi_vpc_resize(hi_vpc_chn chn, const hi_vpc_pic_info *source_pic, hi_vpc_pic_info *dest_pic, hi_double fx, hi_double fy, hi_u32 interpolation, hi_u32 *task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
source_pic	输入	原始图片信息的指针。
dest_pic	输入	目标图片信息的指针。
fx	输入	宽的缩放比例。
fy	输入	高的缩放比例。
interpolation	输入	缩放算法。此处配置的缩放算法建议与训练模型时的缩放算法保持一致。 Atlas 推理系列产品，支持如下缩放算法： ● 0：业界通用的Bilinear算法（与OpenCV算法的计算过程类似，当输入和输出图片格式都为RGB时，在[1/32, 512]的缩放范围内，与OpenCV的单个像素值最大差异为正负1） ● 1：业界通用的Nearest neighbor 算法（与OpenCV算法的计算过程类似）
task_id	输出	任务ID的指针，用来区分任务。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.10 VPC图像处理返回码**

参考资源

接口调用流程及示例，参见**5.5.1 VPC图片处理典型功能**。

11.18.16.7 hi_mpi_vpc_crop

函数功能

按照指定区域从一张输入图片中抠出一张或多张子图。异步接口。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

版本	约束

Atlas 推理系列产品	● 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**hi_pixel_format**枚举值中的如下枚举项：
	HI_PIXEL_FORMAT_YUV_400 = 0, // YUV400 8bit
	HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	HI_PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	HI_PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3, // YUV422SP 8bit
	HI_PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4, // YVU422SP 8bit
	HI_PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5, // YUV444SP 8bit
	HI_PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6, // YVU444SP 8bit
	HI_PIXEL_FORMAT_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	HI_PIXEL_FORMAT_UYVY_PACKED_422 = 8, // YUV422Packed UYVY 8bit
	HI_PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 8bit
	HI_PIXEL_FORMAT_VYUY_PACKED_422 = 10, // YUV422Packed VYUY 8bit
	– 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。
	● 关于输出图片分辨率、图片格式、宽高对齐、内存大小约束请参见**11.18.16.2 约束说明**。
	● 单张1080P的图像处理超时时间建议设置3ms，以此类推，如果是2张1080P的图像，建议设置超时间为6ms；如果是3张1080P图片，建议设置超时时间为9ms。其他分辨率，按照图像尺寸进行近似的等价折算，例如，3840*2160分辨率的图像建议超时时间设置为12ms。

函数原型

hi_s32 hi_mpi_vpc_crop(hi_vpc_chn chn, const hi_vpc_pic_info *source_pic, hi_vpc_crop_region_info crop_info[], hi_u32 count, hi_u32 *task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
source_pic	输入	原始图片信息的指针。
crop_info	输入	抠图图片信息数组，该数组长度与count参数值保持一致。
count	输入	抠图区域的数量，取值范围[1,256]。
task_id	输出	任务ID的指针，用来区分任务。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.10 VPC图像处理返回码**

参考资源

接口调用流程及示例，参见**5.5.1 VPC图片处理典型功能**。

11.18.16.8 hi_mpi_vpc_crop_resize

函数功能

从一张大图中扣出一张或多张子图，并缩放到指定尺寸。异步接口。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

版本	约束

Atlas 推理系列产品	● 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**hi_pixel_format**枚举值中的如下枚举项：
	HI_PIXEL_FORMAT_YUV_400 = 0, // YUV400 8bit
	HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	HI_PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	HI_PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3, // YUV422SP 8bit
	HI_PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4, // YVU422SP 8bit
	HI_PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5, // YUV444SP 8bit
	HI_PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6, // YVU444SP 8bit
	HI_PIXEL_FORMAT_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	HI_PIXEL_FORMAT_UYVY_PACKED_422 = 8, // YUV422Packed UYVY 8bit
	HI_PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 8bit
	HI_PIXEL_FORMAT_VYUY_PACKED_422 = 10, // YUV422Packed VYUY 8bit
	– 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。
	● 关于输出图片分辨率、图片格式、宽高对齐、内存大小约束请参见**11.18.16.2 约束说明**。
	● 当输入、输出图片格式都为ARGB8888、ABGR8888、RGBA8888、BGRA8888格式时，在经过本接口缩放处理后，A分量值会变为0。
	● 若是crop的宽高与resize之后的宽高一致，则不进行缩放；resize宽高必须与输出宽高一致。

函数原型

hi_s32 hi_mpi_vpc_crop_resize(hi_vpc_chn chn, const hi_vpc_pic_info *source_pic, hi_vpc_crop_resize_region crop_resize_info[], hi_u32 count, hi_u32 *task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。

参数名	输入/输出	说明
source_pic	输入	原始图片信息的指针。
crop_resize_info	输入	抠图并缩放的图片信息数组，该数组长度与count参数值保持一致。
count	输入	抠图并缩放的图片数量，取值范围[1,256]。
task_id	输出	任务ID的指针，用来区分任务。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.10 VPC图像处理返回码**

11.18.16.9 hi_mpi_vpc_crop_resize_paste

函数功能

从输入图片中抠图，对抠出的子图进行缩放后，贴在用户输出图片的指定区域。异步接口。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

版本	约束

Atlas 推理系列产品	● 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**hi_pixel_format**枚举值中的如下枚举项：
	HI_PIXEL_FORMAT_YUV_400 = 0, // YUV400 8bit
	HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	HI_PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	HI_PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3, // YUV422SP 8bit
	HI_PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4, // YVU422SP 8bit
	HI_PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5, // YUV444SP 8bit
	HI_PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6, // YVU444SP 8bit
	HI_PIXEL_FORMAT_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	HI_PIXEL_FORMAT_UYVY_PACKED_422 = 8, // YUV422Packed UYVY 8bit
	HI_PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 8bit
	HI_PIXEL_FORMAT_VYUY_PACKED_422 = 10, // YUV422Packed VYUY 8bit
	– 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。
	● 关于输出图片分辨率、图片格式、宽高对齐、内存大小约束请参见**11.18.16.2 约束说明**。
	● 当输入、输出图片格式都为ARGB8888、ABGR8888、RGBA8888、BGRA8888格式时，在经过本接口缩放处理后，A分量值会变为0。
	● 输出图片的数量最多等于抠图区域的数量，最少是一张图片。也就是说，如果从输入图片中扣n张子图，那么贴图的目标图片可以是同一张大图，也可以是多张大图。取决于hi_vpc_crop_resize_paste_region结构体中指定的地址。
	● 由于hi_vpc_crop_resize_paste_region结构体内容比较丰富，使用者可以自由组合功能。例如：若是crop的宽高与resize之后的宽高一致，则不进行缩放。

函数原型

hi_s32 hi_mpi_vpc_crop_resize_paste(hi_vpc_chn chn, const hi_vpc_pic_info *source_pic, hi_vpc_crop_resize_paste_region crop_resize_paste_info[], hi_u32 count, hi_u32 *task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
source_pic	输入	原始图片信息的指针。
crop_resize_paste_info	输入	抠图缩放并贴图的图片信息数组，该数组长度与count参数值保持一致。
count	输入	抠图缩放贴图的图片数量，取值范围[1,256]。
task_id	输出	任务ID的指针，用来区分任务。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.10 VPC图像处理返回码**

11.18.16.10 hi_mpi_vpc_convert_color

函数功能

转换图片的格式。异步接口。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

版本	约束
Atlas 推理系列产品	● 输入、输出图片分辨率：10*6~4096*4096（包括4096*4096）。 ● 输入图片格式、输出图片格式、宽高对齐约束、内存约束请参见**图片格式、宽高对齐、内存约束**。当输入格式是YUV400时，输出格式只能是YUV400。

函数原型

hi_s32 hi_mpi_vpc_convert_color(hi_vpc_chn chn, const hi_vpc_pic_info *source_pic, hi_vpc_pic_info *dest_pic, hi_u32 *task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
source_pic	输入	原始图片信息的指针。
dest_pic	输入	目标图片信息的指针。 需指定输出图片的宽、高、宽stride、高stride、内存地址、内存大小、格式等。 ● 配置输出图片的宽、高时，需与输入图片的宽、高保持一致，否则会返回报错。 如果将输出图片的宽或高配置为0，则VPC内部会将输入图片的宽、高分别作为输出图片的宽、高；同时，VPC内部会按对齐要求计算宽stride、高stride，不同图片格式的对齐要求不同，请参见**表11-45**。 ● 配置输出图片的宽stride、高stride时，满足对齐要求即可，不同图片格式的对齐要求不同，请参见**表11-45**。
task_id	输出	任务ID的指针，用来区分任务。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.10 VPC图像处理返回码**

11.18.16.11 hi_mpi_vpc_convert_color_v2

函数功能

转换图片的格式，在hi_mpi_vpc_convert_color接口的基础上扩展功能，支持设置RGBA格式图片的透明度。异步接口。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

版本	约束
Atlas 推理系列产品	• 输入、输出图片分辨率：10*6~4096*4096（包括4096*4096）。 • 输入图片格式、输出图片格式、宽高对齐约束、内存约束请参见图片格式、宽高对齐、内存约束。当输入格式是YUV400时，输出格式只能是YUV400。

函数原型

hi_s32 hi_mpi_vpc_convert_color_v2(hi_vpc_chn chn, const hi_vpc_pic_info *source_pic, hi_vpc_pic_info *dest_pic, hi_csc_conf *conf, hi_u32 *task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
source_pic	输入	原始图片信息的指针。
dest_pic	输入	目标图片信息的指针。 需指定输出图片的宽、高、宽stride、高stride、内存地址、内存大小、格式等。 • 配置输出图片的宽、高时，需与输入图片的宽、高保持一致，否则会返回报错。 如果将输出图片的宽或高配置为0，则VPC内部会将输入图片的宽、高分别作为输出图片的宽、高；同时，VPC内部会按对齐要求计算宽stride、高stride，不同图片格式的对齐要求不同，请参见表11-45。 • 配置输出图片的宽stride、高stride时，满足对齐要求即可，不同图片格式的对齐要求不同，请参见表11-45。

参数名	输入/输出	说明
conf	输入	输出图片透明度值的指针。 透明度只针对以下图像格式有效： PIXEL_FORMAT_ARGB_8888 = 14, PIXEL_FORMAT_ABGR_8888 = 15, PIXEL_FORMAT_RGBA_8888 = 16, PIXEL_FORMAT_BGRA_8888 = 17,
task_id	输出	任务ID的指针，用来区分任务。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.10 VPC图像处理返回码**

11.18.16.12 hi_mpi_vpc_convert_color_to_yuv420

函数功能

转换图片的格式，且转换后的格式仅支持YUV420 semi-planar/YVU420 semi-planar。异步接口。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 本接口兼容旧昇腾AI处理器版本上的格式转换接口，兼容场景下的输入、输出图片格式如下，兼容场景下输出的二进制相同：
  - 输入图片格式：YUV420 semi-planar/YVU420 semi-planar/YUV422 semi-planar/YVU422 semi-planar/YUV444 packed/YUV444 semi-planar/YVU444 semi-planar
  - 输出图片格式：YUV420 semi-planar/YVU420 semi-planar
  - 在以上输入图片格式、输出图片格式场景下，本接口与**hi_mpi_vpc_convert_color**接口输出的二进制不一致。
- 关于输入分辨率的要求：

版本	约束
Atlas 推理系列产品	输入图片分辨率：10*6~4096*4096（包括4096*4096）。

- 输入图片格式支持hi_pixel_format枚举值中的如下枚举项：
```
HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1,      // YUV420SP NV12 8bit
HI_PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2,      // YUV420SP NV21 8bit
HI_PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3,      // YUV422SP 8bit
HI_PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4,      // YVU422SP 8bit
HI_PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5,      // YUV444SP 8bit
HI_PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6,      // YVU444SP 8bit
HI_PIXEL_FORMAT_YUYV_PACKED_422 = 7,         // YUV422Packed YUYV 8bit
HI_PIXEL_FORMAT_UYVY_PACKED_422 = 8,         // YUV422Packed UYVY 8bit
HI_PIXEL_FORMAT_YVYU_PACKED_422 = 9,         // YUV422Packed YVYU 8bit
HI_PIXEL_FORMAT_VYUY_PACKED_422 = 10,        // YUV422Packed VYUY 8bit
HI_PIXEL_FORMAT_YUV_PACKED_444 = 11,         // YUV444Packed 8bit
HI_PIXEL_FORMAT_RGB_888 = 12,                // RGB888
HI_PIXEL_FORMAT_BGR_888 = 13,                // BGR888
HI_PIXEL_FORMAT_ARGB_8888 = 14,              // ARGB8888
HI_PIXEL_FORMAT_ABGR_8888 = 15,              // ABGR8888
HI_PIXEL_FORMAT_RGBA_8888 = 16,              // RGBA8888
HI_PIXEL_FORMAT_BGRA_8888 = 17,              // BGRA8888
HI_PIXEL_FORMAT_YUV_SEMIPLANAR_440 = 1000,   // YUV440SP 8bit
HI_PIXEL_FORMAT_YVU_SEMIPLANAR_440 = 1001,   // YVU440SP 8bit
```
- 输出图片格式支持hi_pixel_format枚举值中的如下枚举项，设置其它图片格式无效，接口会返回失败：
```
HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1,      // YUV420SP NV12 8bit
HI_PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2,      // YUV420SP NV21 8bit
```
- 关于各图片格式的宽高对齐、内存大小约束，请参见图片格式、宽高对齐、内存约束。

函数原型

hi_s32 hi_mpi_vpc_convert_color_to_yuv420(hi_vpc_chn chn, const hi_vpc_pic_info *source_pic, hi_vpc_pic_info *dest_pic, hi_u32 *task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
source_pic	输入	原始图片信息的指针。

参数名	输入/输出	说明
dest_pic	输入	目标图片信息的指针。 需指定输出图片的宽、高、宽stride、高stride、内存地址、内存大小、格式等。 ● 配置输出图片的宽、高时，需与输入图片的宽、高保持一致，否则会返回报错。 如果将输出图片的宽或高配置为0，则VPC内部会将输入图片的宽、高分别作为输出图片的宽、高；同时，VPC内部会按对齐要求计算宽stride、高stride，不同图片格式的对齐要求不同，请参见表11-45。 ● 配置输出图片的宽stride、高stride时，满足对齐要求即可，不同图片格式的对齐要求不同，请参见表11-45。
task_id	输出	任务ID的指针，用来区分任务。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.10 VPC图像处理返回码

11.18.16.13 hi_mpi_vpc_copy_make_border

函数功能

对输入图像进行边界填充。异步接口。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

版本	约束
Atlas 推理系列产品	● 输入、输出图片分辨率为10*6~4096*4096。 ● 输入、输出图片的格式、各图片格式的宽高对齐、内存大小约束，请参见图片格式、宽高对齐、内存约束。扩充图像的宽没有16对齐的要求，但是内存申请仍然要按照16的对齐要求进行申请，即Stride要按照16对齐。 ● 由于YUV格式图像下采样约束，当输出图片格式为YUV420SP或YUV422SP格式时，需注意： – 对于YUV420SP输出格式，上下左右填充的尺寸建议为偶数； – 对于YUV422SP输出格式，左右填充的尺寸建议为偶数。

函数原型

hi_s32 hi_mpi_vpc_copy_make_border(hi_vpc_chn chn, const hi_vpc_pic_info *source_pic, hi_vpc_pic_info *dest_pic, hi_vpc_make_border_info make_border_info, hi_u32 *task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
source_pic	输入	原始图片信息的指针。
dest_pic	输入	目标图片信息的指针。
make_border_info	输入	边界填充信息。
task_id	输出	任务ID的指针，用来区分任务。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.10 VPC图像处理返回码

11.18.16.14 hi_mpi_vpc_pyrdown

函数功能

对图像进行金字塔缩放，当前仅支持YUV400的图片格式。异步接口。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpc_pyrdown(hi_vpc_chn chn, const hi_vpc_pic_info *source_pic, hi_vpc_pic_info dest_pic[], hi_u32 filter_level, hi_s8 gaussian_filter[][5], hi_u16 divisor, hi_vpc_make_border_info make_border_info, hi_u32 *task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
source_pic	输入	输入图片信息的指针。输入图片分辨率最大支持2048*2048，仅支持YUV 400格式输入。根据filter_level, 最小输入分辨率限制如下： filter_level = 1，最小输入分辨率为20*12 filter_level = 2，最小输入分辨率为40*24 filter_level = 3，最小输入分辨率为80*48 filter_level = 4，最小输入分辨率为160*96
dest_pic	输入	目标图片信息，该数组长度与filter_level参数值保持一致。 dest_pic[0]保存的是原图宽高都缩小为1/2的图，dest_pic[1]保存的是原图宽高都缩小为1/4的图，dest_pic[2]保存的是原图宽高都缩小为1/8的图，dest_pic[3]保存的是原图宽高都缩小为1/16的图。
filter_level	输入	参数有效范围是1-4，指定金字塔图像层数以及图像数量。

参数名	输入/输出	说明
gaussian_filter	输入	高斯滤波参数。 将该参数设置为空指针，表示使用如下默认值： {{1, 4, 6, 4, 1}, {4, 16, 24, 16, 4}, {6, 24, 36, 24, 6}, {4, 16, 24, 16, 4}, {1, 4, 6, 4, 1}}
divisor	输入	滤波器除数，必须是2的幂次方。
make_border_info	输入	边界填充信息。 固定填充2个像素，支持的填充类型为：HI_BORDER_CONSTANT、HI_BOARD_REPLICATE和HI_BOARD_REFLECT。
task_id	输出	任务ID的指针，用来区分任务。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.10 VPC图像处理返回码**

11.18.16.15 hi_mpi_vpc_calc_hist

函数功能

统计图像每个通道的像素值分布。异步接口。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpc_calc_hist(hi_vpc_chn chn, const hi_vpc_pic_info *source_pic, hi_vpc_histogram_config *hist_config, hi_u32 *task_id, hi_s32 milli_sec)

约束说明

- 关于输入分辨率的要求：

版本	约束
Atlas 推理系列产品	输入图片分辨率：10*6~4096*4096（包括4096*4096）。

- 输入图片格式支持**hi_pixel_format**枚举值中的如下枚举项：

```
HI_PIXEL_FORMAT_YUV_400 = 0,            // YUV400 8bit
HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1,    // YUV420SP NV12 8bit
HI_PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2,    // YUV420SP NV21 8bit
HI_PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3,    // YUV422SP 8bit
HI_PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4,    // YVU422SP 8bit
HI_PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5,    // YUV444SP 8bit
HI_PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6,    // YVU444SP 8bit
HI_PIXEL_FORMAT_YUYV_PACKED_422 = 7,       // YUV422Packed YUYV 8bit
HI_PIXEL_FORMAT_UYVY_PACKED_422 = 8,       // YUV422Packed UYVY 8bit
HI_PIXEL_FORMAT_YVYU_PACKED_422 = 9,       // YUV422Packed YVYU 8bit
HI_PIXEL_FORMAT_VYUY_PACKED_422 = 10,      // YUV422Packed VYUY 8bit
HI_PIXEL_FORMAT_YUV_PACKED_444 = 11,       // YUV444P 8bit
HI_PIXEL_FORMAT_RGB_888 = 12,            // RGB888
HI_PIXEL_FORMAT_BGR_888 = 13,            // BGR888
HI_PIXEL_FORMAT_YUV_SEMIPLANAR_440 = 1000, // YUV440SP 8bit
HI_PIXEL_FORMAT_YVU_SEMIPLANAR_440 = 1001, // YVU440SP 8bit
```

- 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
source_pic	输入	输入图片信息的指针。 **说明** 当图片格式为YUV440SP 8bit、YVU440SP 8bit时，Y分量统计准确，UV分量统计上存在误差。
hist_config	输出	统计结果的指针。 hist_config是一个包含3个数组的结构体，用来存放每张图像3个分量的0-255的分布情况。
task_id	输出	任务ID的指针，用来区分任务。

参数名	输入/输出	说明
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

- hist_config：统计结果。
- task_id：此次任务分配的ID，用来区分任务。

11.18.16.16 hi_mpi_vpc_equalize_hist

函数功能

将输入图片中指定分量的指定像素值重映射成其它像素值，作为输出图片，主要用于图像增强的场景。通过图片的色彩重映射以达到图片对比度更好，清晰度更好的效果。异步接口。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 输入图片分辨率为10*6~4096*4096（包括4096）。
- 输入图片格式支持hi_pixel_format枚举值中的如下枚举项：
```
HI_PIXEL_FORMAT_YUV_400 = 0,            // YUV400 8bit
HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1,   // YUV420SP NV12 8bit
HI_PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2,   // YUV420SP NV21 8bit
HI_PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3,   // YUV422SP 8bit
HI_PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4,   // YVU422SP 8bit
HI_PIXEL_FORMAT_YUV_PACKED_444 = 11,      // YUV444P 8bit
HI_PIXEL_FORMAT_RGB_888 = 12,           // RGB888
HI_PIXEL_FORMAT_BGR_888 = 13,           // BGR888
```
- 输出图片格式与输入图片格式保持一致。
- 关于各图片格式的宽高对齐、内存大小约束，请参见图片格式、宽高对齐、内存约束。

函数原型

hi_s32 hi_mpi_vpc_equalize_hist(hi_vpc_chn chn, const hi_vpc_pic_info* source_pic, hi_vpc_pic_info *dest_pic, const hi_vpc_lut_remap *lut_remap, hi_u32 *task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
source_pic	输入	输入图片信息的指针。
dest_pic	输入	目标图片信息的指针。
lut_remap	输入	图像lut_remap的配置的指针。 lut_remap是一个768 Byte的数组，配置给硬件进行图片像素重映射。
task_id	输出	任务ID的指针，用来区分任务。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.10 VPC图像处理返回码**

11.18.16.17 hi_mpi_vpc_crop_resize_make_border

函数功能

按指定区域从一张输入图片中抠出一个或多个子图，对子图缩放后，再将每个子图按指定类型填充，作为一张或多张目标图片输出，主要用于等比例缩放场景。异步接口。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

版本	约束

Atlas 推理系列产品	● 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**hi_pixel_format**枚举值中的如下枚举项：
	HI_PIXEL_FORMAT_YUV_400 = 0, // YUV400 8bit
	HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	HI_PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	HI_PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3, // YUV422SP 8bit
	HI_PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4, // YVU422SP 8bit
	HI_PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5, // YUV444SP 8bit
	HI_PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6, // YVU444SP 8bit
	HI_PIXEL_FORMAT_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	HI_PIXEL_FORMAT_UYVY_PACKED_422 = 8, // YUV422Packed UYVY 8bit
	HI_PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 8bit
	HI_PIXEL_FORMAT_VYUY_PACKED_422 = 10, // YUV422Packed VYUY 8bit
	– 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。
	● 关于输出图片分辨率、图片格式、宽高对齐、内存大小约束请参见**11.18.16.2 约束说明**。
	● 当输入、输出图片格式都为ARGB8888、ABGR8888、RGBA8888、BGRA8888格式时，在经过本接口缩放处理后，A分量值会变为0。
	● 由于hi_vpc_crop_resize_border_region结构体内容比较丰富，使用者可以自由组合功能。但存在一些约束，例如：若是crop的宽高与resize之后的宽高一致，则不进行缩放；若偏移值为0，则不进行填充。
	● 使用本接口实现等比例缩放功能时，相关的描述及约束请参见**抠图、贴图约束**。
	● 由于**YUV格式图像下采样约束**，当输出图片格式为YUV420SP或YUV422SP格式时，需注意：
	– 对于YUV420SP输出格式，上下左右填充的尺寸建议为偶数；
	– 对于YUV422SP输出格式，左右填充的尺寸建议为偶数。

函数原型

hi_s32 hi_mpi_vpc_crop_resize_make_border(hi_vpc_chn chn, const hi_vpc_pic_info *source_pic, hi_vpc_crop_resize_border_region

crop_resize_make_border_info[], hi_u32 count, hi_u32 *task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
source_pic	输入	原始图片信息的指针。
crop_resize_make_border_info	输入	抠图缩放填充的图片信息数组，需要用户设置抠图、缩放、填充信息以及目标图片的内存地址，VPC将抠图缩放填充后的结果数据存放在用户指定的内存地址中。如果多个填充结果的数据都指向同一个内存地址，后一次的结果数据会覆盖前一次的结果数据，导致输出图片跟预期不一致。 当前填充类型仅支持HI_BORDER_CONSTANT和HI_BORDER_REPLICATE，可以Padding到4096*4096。crop_resize_make_border_info数组的长度与count参数值保持一致。
count	输入	抠图缩放填充的图片数量，取值范围[1,256]。
task_id	输出	任务ID的指针，用来区分任务。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.10 VPC图像处理返回码**

11.18.16.18 hi_mpi_vpc_batch_crop_resize_paste

函数功能

hi_mpi_vpc_crop_resize_paste的扩展接口，支持一次处理多张输入图片。可输入多张原图，从每张原图中抠一张或多张子图缩放并贴图到每张目标图上，目标图片的数量与子图数量保持一致。异步接口。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

版本	约束

Atlas 推理系列产品	● 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**hi_pixel_format**枚举值中的如下枚举项：
	HI_PIXEL_FORMAT_YUV_400 = 0, // YUV400 8bit
	HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	HI_PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	HI_PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3, // YUV422SP 8bit
	HI_PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4, // YVU422SP 8bit
	HI_PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5, // YUV444SP 8bit
	HI_PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6, // YVU444SP 8bit
	HI_PIXEL_FORMAT_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	HI_PIXEL_FORMAT_UYVY_PACKED_422 = 8, // YUV422Packed UYVY 8bit
	HI_PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 8bit
	HI_PIXEL_FORMAT_VYUY_PACKED_422 = 10, // YUV422Packed VYUY 8bit
	– 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。
	● 关于输出图片分辨率、图片格式、宽高对齐、内存大小约束请参见**11.18.16.2 约束说明**。
	● 当输入、输出图片格式都为ARGB8888、ABGR8888、RGBA8888、BGRA8888格式时，在经过本接口缩放处理后，A分量值会变为0。
	● 由于hi_vpc_crop_resize_paste_region结构体内容比较丰富，使用者可以自由组合功能。但存在一些通用约束，例如：
	– 若是crop的宽高和原图宽高一致，则不进行crop。
	– 若是crop的宽高与resize之后的宽高一致，则不进行缩放。
	– 若是crop、resize之后的宽高和目标图片一致，并且贴图左上角坐标等于(0,0)，则不进行贴图。

函数原型

hi_s32 hi_mpi_vpc_batch_crop_resize_paste(hi_vpc_chn chn, const hi_vpc_pic_info *source_pic[], hi_u32 pic_num, hi_vpc_crop_resize_paste_region crop_resize_paste_info[], hi_u32 count[], hi_u32 *task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
source_pic	输入	原始图片信息的指针数组。
pic_num	输入	原始图片数量，与source_pic数组长度、count数组长度保持一致。
crop_resize_paste_info	输入	抠图缩放并贴图信息结构体数组，需要用户设置抠图、缩放、贴图信息以及目标图片的内存地址，VPC将抠图缩放贴图后的结果数据存放在用户指定的内存地址中。
count	输入	每张原图抠图缩放贴图的图片数量，数组内第一个元素的值表示第一张原图的抠图缩放贴图的图片数量，第二个元素的值表示第二张原图的抠图缩放贴图的图片数量，以此类推。count数组内元素值之和等于crop_resize_paste_info数组的长度。count数组内元素值之和的取值范围[1,256]。
task_id	输出	任务ID的指针，用来区分任务。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.10 VPC图像处理返回码**

11.18.16.19 hi_mpi_vpc_batch_crop_resize_make_border

函数功能

hi_mpi_vpc_crop_resize_make_border的扩展接口，支持批处理模式。可输入多张原图，按指定区域从每张输入图片中抠出一个或多个子图，对子图缩放后，再将子图贴到每张目标图片（目标图片的数量与子图数量保持一致）的指定位置，区域之外按指定类型填充。异步接口。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

版本	约束

Atlas 推理系列产品	● 关于输入图片分辨率、图片格式：
	– 输入图片分辨率在10*6~4096*4096（包括4096）范围内时，支持**图片格式、宽高对齐、内存约束**处说明的输入图片格式。
	– 输入图片的宽或高在4096~8192（包括8192）范围内时，支持格式如下：
	支持**hi_pixel_format**枚举值中的如下枚举项：
	HI_PIXEL_FORMAT_YUV_400 = 0, // YUV400 8bit
	HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // YUV420SP NV12 8bit
	HI_PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // YUV420SP NV21 8bit
	HI_PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3, // YUV422SP 8bit
	HI_PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4, // YVU422SP 8bit
	HI_PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5, // YUV444SP 8bit
	HI_PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6, // YVU444SP 8bit
	HI_PIXEL_FORMAT_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	HI_PIXEL_FORMAT_UYVY_PACKED_422 = 8, // YUV422Packed UYVY 8bit
	HI_PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 8bit
	HI_PIXEL_FORMAT_VYUY_PACKED_422 = 10, // YUV422Packed VYUY 8bit
	– 关于各图片格式的宽高对齐、内存大小约束，请参见**图片格式、宽高对齐、内存约束**。
	● 关于输出图片分辨率、图片格式、宽高对齐、内存大小约束请参见**11.18.16.2 约束说明**。
	● 当输入、输出图片格式都为ARGB8888、ABGR8888、RGBA8888、BGRA8888格式时，在经过本接口缩放处理后，A分量值会变为0。
	● 由于hi_vpc_crop_resize_border_region结构体内容比较丰富，使用者可以自由组合功能。但存在一些通用约束，例如：
	– 若是crop的宽高和原图宽高一致，则不进行crop。
	– 若是crop的宽高与resize之后的宽高一致，则不进行缩放。
	– 若是crop、resize之后的宽高和目标图片的宽高一致，并且左上角坐标等于(0,0)，则不进行填充。
	● 使用本接口实现等比例缩放功能时，相关的描述及约束请参见**抠图、贴图约束**。
	● 由于**YUV格式图像下采样约束**，当输出图片格式为YUV420SP或YUV422SP格式时，需注意：
	– 对于YUV420SP输出格式，上下左右填充的尺寸建议为偶数；
	– 对于YUV422SP输出格式，左右填充的尺寸建议为偶数。

函数原型

hi_s32 hi_mpi_vpc_batch_crop_resize_make_border(hi_vpc_chn chn, const hi_vpc_pic_info *source_pic[], hi_u32 pic_num, hi_vpc_crop_resize_border_region crop_resize_make_border_info[], hi_u32 count[], hi_u32 *task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
source_pic	输入	原始图片信息的指针数组。
pic_num	输入	原始图片数量，与source_pic数组长度、count数组长度保持一致。
crop_resize_make_border_info	输入	抠图缩放填充的信息结构体，需要用户设置抠图、缩放、填充信息以及目标图片的内存地址，VPC将抠图缩放填充后的结果数据存放在用户指定的内存地址中。当前填充类型仅支持HI_BORDER_CONSTANT和HI_BORDER_REPLICATE，可以填充到4096*4096。
count	输入	每张原图抠图缩放填充的图片数量，count数组内元素值之和等于crop_resize_make_border_info数组的长度。count数组内元素值之和的取值范围[1,256]。
task_id	输出	任务ID的指针，用来区分任务。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.10 VPC图像处理返回码**

11.18.16.20 hi_mpi_vpc_get_process_result

函数功能

等待任务处理完成，再获取图片处理结果。

由于VPC功能接口是异步接口（例如：hi_mpi_vpc_crop），因此在调用VPC功能接口之后，还需要调用本接口等待任务处理完成，然后再获取处理结果。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpc_get_process_result(hi_vpc_chn chn, hi_u32 task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
task_id	输入	任务ID。 此处通过任务ID来明确需要获取哪次任务执行后的结果数据。 任务ID可以在调用抠图、缩放等接口后获取，请参见**11.18.16.6 hi_mpi_vpc_resize**~**11.18.16.19 hi_mpi_vpc_batch_crop_resize_make_border**。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.10 VPC图像处理返回码**

参考资源

接口调用流程及示例，参见5.5.1 VPC图片处理典型功能。

11.18.16.21 hi_mpi_vpc_sys_create_chn

函数功能

根据设置的通道属性创建图像处理通道，由系统返回可用通道号给用户。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpc_sys_create_chn(hi_vpc_chn *chnl, const hi_vpc_chn_attr *attr)

参数说明

参数名	输入/输出	说明
chnl	输出	图片处理通道号的指针，调用该接口后，由系统返回可用通道号给用户。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
attr	输入	图片处理通道属性指针。 如果参数attr为空，会返回错误码HI_ERR_VPC_NULL_PTR。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.10 VPC图像处理返回码

11.18.16.22 hi_mpi_vpc_set_roundview_stitching_param

函数功能

设置环视拼接参数，包括畸变矫正、增益补偿参数。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 输入、输出图片宽高必须准确配置，分辨率为[10, 6]到[4096,4096]。
- 针对一个通道，只能调用本接口设置一次环视拼接参数。
- 设置拼接参数时，拼接表中的图片编号需设置成功，否则可能出现拼接后的输出图片异常，具体图片编号的设置请参见11.18.20.12.19 hi_stiching_ipm_table。

函数原型

hi_s32 hi_mpi_vpc_set_roundview_stitching_param(hi_vpc_chn chn, const hi_roundview_stitching_param *stitch_param)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
stitch_param	输入	环视拼接参数信息，包括畸变矫正、增益补偿参数。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.10 VPC图像处理返回码**

11.18.16.23 hi_mpi_vpc_get_roundview_stitching_param

函数功能

获取环视拼接参数，包括畸变矫正、增益补偿参数。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

调用本接口前，必须正确设置ipm_table_address、dest_pic_width、dest_pic_height和ipm_table_size参数。

函数原型

hi_s32 hi_mpi_vpc_get_roundview_stitching_param(hi_vpc_chn chn, hi_roundview_stitching_param *stitch_param)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。

参数名	输入/输出	说明
stitch_param	输出	环视拼接参数。 输出结果中的imp_param.ipm_table_address中的float类型参数值，与用户设置输入时的参数存在浮点精度差异（不超过0.01）。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.10 VPC图像处理返回码**

11.18.16.24 hi_mpi_vpc_roundview_stitching

函数功能

下发环视拼接任务。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

不能在一个通道内并发下发多个环视拼接任务。需要**hi_mpi_vpc_get_process_result**后，再通过hi_mpi_vpc_roundview_stitching接口下发环视拼接任务。

函数原型

hi_s32 hi_mpi_vpc_roundview_stitching(hi_vpc_chn chn, hi_roundview_stitching_pic_param *roundview_stitching_pic_param, hi_u32 *task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。

参数名	输入/输出	说明
roundview_stitching_pic_param	输入	输入和输出图片信息。 ● 输入： – 输入图片数量：当前环视拼接功能仅支持对4张输入图片； – 输入图片格式：支持YUV420SP、YVU420SP； – 输入图片分辨率：取值范围在10*6~4096*4096之间，包含10*6和4096*4096。 ● 输出： – 输出图片数量：1张图片； – 输出图片格式：支持YUV420SP、YVU420SP、RGB888、BGR888； – 输出图片分辨率：取值范围在10*6~4096*4096之间，包含10*6和4096*4096。
milli_sec	输入	超时时间，单位是毫秒。 ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。
task_id	输出	此次任务分配的ID，用来区分任务。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.10 VPC图像处理返回码**

11.18.16.25 hi_mpi_vpc_crop_resize_resize_paste

函数功能

从输入图片中抠图，对抠出的子图进行两次缩放后（两次缩放目的是对敏感图像信息模糊脱敏），贴在用户输出图片的指定区域。异步接口。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 输出图片的数量最多等于抠图区域的数量，最少是一张图片。也就是说，如果从输入图片中扣n张子图，那么贴图的目标图片可以是同一张大图，也可以是多张大图。取决于hi_vpc_crop_resize_resize_paste_region结构体中指定的地址。

- 当输入、输出图片格式都为ARGB8888、ABGR8888、RGBA8888、BGRA8888格式时，在经过本接口缩放处理后，A分量值会变为0。
- VPC功能的约束请参见**11.18.16.2 约束说明**。

函数原型

hi_s32 hi_mpi_vpc_crop_resize_resize_paste(hi_vpc_chn chn, const hi_vpc_pic_info *source_pic, hi_vpc_crop_resize_resize_paste_region crop_resize_resize_paste_info[], hi_u32 count, hi_u32 *task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
source_pic	输入	原始图片信息的指针。
crop_resize_resize_paste_info	输入	抠图缩放并贴图的图片信息数组，该数组长度与count参数值保持一致。
count	输入	抠图缩放贴图的图片数量，取值范围[1,256]。
task_id	输出	任务ID的指针，用来区分任务。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.10 VPC图像处理返回码**

11.18.16.26 hi_mpi_vpc_set_chn_workspace

函数功能

调用本接口后，系统内部会根据输入图片、输出图片的最大宽高信息申请需使用的内部临时缓存，通过该接口申请的临时缓存，在销毁通道时会自动释放。同步接口。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpc_set_chn_workspace(hi_vpc_chn chn, hi_vpc_workspace_param *workspace_param, hi_u32 param_cnt)

约束说明

对于每个通道，该接口只能设置一次，后续不能修改。

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
workspace_param	输入	指向workspace_param类型数组的指针。
param_cnt	输入	数组长度，需与workspace_param数组长度保持一致。

返回值说明

- 0：成功
- 非0：失败，**11.18.21.10 VPC图像处理返回码**

11.18.16.27 hi_mpi_vpc_get_lut_mem_size

函数功能

根据输出图片宽高信息获取像素位置重映射表所需的内存大小。同步接口。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpc_get_lut_mem_size(hi_u32 width, hi_u32 height, hi_u32 *lut_size)

参数说明

参数名	输入/输出	说明
width	输入	输出图像宽度。
height	输入	输出图像高度。
lut_size	输出	像素位置重映射表的内存大小，单位Byte。

返回值说明

- 0：成功
- 非0：失败，11.18.21.10 VPC图像处理返回码

11.18.16.28 hi_mpi_vpc_get_affine_lut

函数功能

根据输入、输出图片中的像素位置坐标点，获取仿射变换像素位置重映射表。同步接口。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpc_get_affine_lut(hi_point_pair_info *point_pair_info, hi_u32 interpolation, hi_remap_lut *remap_lut)

参数说明

参数名	输入/输出	说明
point_pair_info	输入	三对输入、输出图片中的像素点坐标信息，每对中分别包含一个输入图片像素点坐标、一个输出图片像素点坐标。
interpolation	输入	缩放算法。 支持如下缩放算法： ● 0：业界通用的Bilinear算法（与OpenCV算法的计算过程类似） ● 1：业界通用的Nearest neighbor算法（与OpenCV算法的计算过程类似）

参数名	输入/输出	说明
remap_lut	输入&输出	仿射变换像素位置重映射表信息。 在hi_remap_lut结构体内： ● lut参数作为输入时，需由用户提前申请该内存，内存大小lut_size可调用hi_mpi_vpc_get_lut_mem_size接口获取。 ● lut参数作为输出时，在成功调用本接口后，用户可从该内存中获取仿射变换LUT表信息。

返回值说明

- 0：成功
- 非0：失败，11.18.21.10 VPC图像处理返回码

11.18.16.29 hi_mpi_vpc_get_perspective_lut

函数功能

根据输入、输出图片中的像素位置坐标点，获取透视变换像素位置重映射表。同步接口。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpc_get_perspective_lut(hi_point_pair_info *point_pair_info, hi_u32 interpolation, hi_remap_lut *remap_lut)

参数说明

参数名	输入/输出	说明
point_pair_info	输入	四对输入、输出图片中的像素点坐标信息，每对中分别包含一个输入图片像素点坐标、一个输出图片像素点坐标。
interpolation	输入	支持如下缩放算法： ● 0：业界通用的Bilinear算法（与OpenCV算法的计算过程类似） ● 1：业界通用的Nearest neighbor算法（与OpenCV算法的计算过程类似）

参数名	输入/输出	说明
remap_lut	输入输出	透视变换像素位置重映射表。 在hi_remap_lut结构体内： ● lut参数作为输入时，需由用户提前申请该内存，内存大小lut_size可调用hi_mpi_vpc_get_lut_mem_size接口获取。 ● lut参数作为输出时，在成功调用本接口后，用户可从该内存中获取仿射变换LUT表信息。

返回值说明

- 0：成功
- 非0：失败，11.18.21.10 VPC图像处理返回码

11.18.16.30 hi_mpi_vpc_get_remap_lut

函数功能

根据用户输入的Remap表（例如仿射、透视、鱼眼等），采用业界通用的Bilinear算法（与OpenCV算法的计算过程类似），获取系统使用的像素位置重映射表。同步接口。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpc_get_remap_lut(hi_map_param *map_param, hi_remap_lut *remap_lut)

参数说明

参数名	输入/输出	说明
map_param	输入	用户原始map表信息。
remap_lut	输入&输出	像素位置重映射表。 在hi_remap_lut结构体内： ● lut参数作为输入时，需由用户提前申请该内存，内存大小lut_size可调用hi_mpi_vpc_get_lut_mem_size接口获取。 ● lut参数作为输出时，在成功调用本接口后，用户可从该内存中获取仿射变换LUT表信息。

返回值说明

- 0：成功
- 非0：失败，11.18.21.10 VPC图像处理返回码

11.18.16.31 hi_mpi_vpc_lut_remap

函数功能

基于输入像素位置重映射表的方式对图片进行Remap处理。异步接口。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpc_lut_remap(hi_vpc_chn chn, hi_warp_transform_param *transform_param, hi_remap_lut *remap_lut, hi_u32 *task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
transform_param	输入	Remap变换参数。
remap_lut	输入	像素位置重映射表信息。 需提前调用以下get接口获取像素位置重映射表信息：hi_mpi_vpc_get_affine_lut、或hi_mpi_vpc_get_perspective_lut、或hi_mpi_vpc_get_remap_lut，但本接口的缩放算法需与get接口的缩放算法保持一致，否则可能导致remap结果不符合预期。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。
task_id	输出	任务ID的指针，用来区分任务。

返回值说明

- 0：成功
- 非0：失败，11.18.21.10 VPC图像处理返回码

11.18.16.32 hi_mpi_vpc_median_blur

函数功能

对输入图片做中值滤波处理。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpc_median_blur(hi_vpc_chn chn, const hi_median_blur_param* median_blur_param, hi_u32* task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
median_blur_param	输入	中值滤波信息 在hi_median_blur_param结构体内配置输入、输出图片信息，以及中值滤波参数。其中，中值滤波参数在hi_median_blur_config结构体内配置。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。
task_id	输出	任务ID的指针，用来区分任务。

返回值说明

- 0：成功
- 非0：失败，11.18.21.10 VPC图像处理返回码

11.18.16.33 hi_mpi_vpc_erode

函数功能

对输入图片做腐蚀处理。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpc_erode(hi_vpc_chn chn, const hi_blur_param* erode_param, hi_u32* task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
erode_param	输入	腐蚀信息。 在hi_blur_param结构体内配置输入、输出图片信息，以及腐蚀参数。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。
task_id	输出	任务ID的指针，用来区分任务。

返回值说明

- 0：成功
- 非0：失败，11.18.21.10 VPC图像处理返回码

11.18.16.34 hi_mpi_vpc_dilate

函数功能

对输入图片做膨胀处理。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpc_dilate(hi_vpc_chn chn, const hi_blur_param* dilate_param, hi_u32* task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
dilate_param	输入	膨胀信息。 在hi_blur_param结构体内配置输入、输出图片信息，以及膨胀参数。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。
task_id	输出	任务ID的指针，用来区分任务。

返回值说明

- 0：成功
- 非0：失败，11.18.21.10 VPC图像处理返回码

11.18.16.35 hi_mpi_vpc_blur

函数功能

对输入图片做均值滤波处理。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpc_blur(hi_vpc_chn chn, const hi_blur_param* blur_param, hi_u32* task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
blur_param	输入	滤波信息。 在hi_blur_param结构体内配置输入、输出图片信息，以及均值滤波参数。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。
task_id	输出	任务ID的指针，用来区分任务。

返回值说明

- 0：成功
- 非0：失败，11.18.21.10 VPC图像处理返回码

11.18.16.36 hi_mpi_vpc_gaussian_blur

函数功能

对输入图片做高斯滤波处理。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpc_gaussian_blur(hi_vpc_chn chn, const hi_gaussian_blur_param* gaussian_blur_param, hi_u32* task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
gaussian_blur_param	输入	高斯滤波信息。 在hi_gaussian_blur_param结构体内配置输入、输出图片信息，以及高斯滤波参数。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。
task_id	输出	任务ID的指针，用来区分任务。

返回值说明

- 0：成功
- 非0：失败，11.18.21.10 VPC图像处理返回码

11.18.16.37 hi_mpi_vpc_filter2d

函数功能

对输入图片做2D卷积滤波处理。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpc_filter2d(hi_vpc_chn chn, const hi_filter_2d_param* filter_2d_param, hi_u32* task_id, hi_s32 milli_sec);

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
filter_2d_param	输入	2D卷积滤波信息。 在hi_filter_2d_param结构体内配置输入、输出图片信息，以及卷积滤波参数。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。
task_id	输出	任务ID的指针，用来区分任务。

返回值说明

- 0：成功
- 非0：失败，11.18.21.10 VPC图像处理返回码

11.18.16.38 hi_mpi_vpc_rotate

函数功能

对输入图片做90度/180度/270度操作，输出图片宽高也要随旋转角度置换。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpc_rotate(hi_vpc_chn chn, const hi_rotate_param* rotate_param, hi_u32* task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
rotate_param	输入	旋转信息。 在hi_rotate_param结构体内配置输入、输出图片信息，以及旋转参数。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。
task_id	输出	任务ID的指针，用来区分任务。

返回值说明

- 0：成功
- 非0：失败，11.18.21.10 VPC图像处理返回码

11.18.16.39 hi_mpi_vpc_draw_mosaic

函数功能

对输入图片的部分区域做马赛克处理。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpc_draw_mosaic(hi_vpc_chn chn, const hi_mosaic_param* mosaic_param, hi_u32* task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
mosaic_param	输入	马赛克信息。 在hi_mosaic_param结构体内配置输入、输出图片信息，以及马赛克参数。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。
task_id	输出	任务ID的指针，用来区分任务。

返回值说明

- 0：成功
- 非0：失败，11.18.21.10 VPC图像处理返回码

11.18.16.40 hi_mpi_vpc_draw_cover

函数功能

对输入图片的部分区域做覆盖处理。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpc_draw_cover(hi_vpc_chn chn, const hi_cover_param* cover_param, hi_u32* task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
cover_param	输入	覆盖信息。 在hi_cover_param结构体内配置输入、输出图片信息，以及覆盖参数。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。
task_id	输出	任务ID的指针，用来区分任务。

返回值说明

- 0：成功
- 非0：失败，11.18.21.10 VPC图像处理返回码

11.18.16.41 hi_mpi_vpc_draw_line

函数功能

对输入图片做画线处理。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpc_draw_line(hi_vpc_chn chn, const hi_line_param* line_param, hi_u32* task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
line_param	输入	画线信息。 在hi_line_param结构体内配置输入、输出图片信息，以及画线参数。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。
task_id	输出	任务ID的指针，用来区分任务。

返回值说明

- 0：成功
- 非0：失败，11.18.21.10 VPC图像处理返回码

11.18.16.42 hi_mpi_vpc_draw_osd

函数功能

对输入图片做叠加处理，可用于添加水印的场景。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpc_draw_osd(hi_vpc_chn chn, hi_osd_param* osd_param, hi_u32* task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
osd_param	输入	叠加信息。 在hi_osd_param结构体内配置输入、输出图片信息，以及叠加参数。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。
task_id	输出	任务ID的指针，用来区分任务。

返回值说明

- 0：成功
- 非0：失败，11.18.21.10 VPC图像处理返回码

11.18.16.43 hi_mpi_vpc_get_affine_transform

函数功能

获取仿射变换需要的变换矩阵。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpc_get_affine_transform(hi_point_pair_info *point_pair_info, hi_transform_matrix *matrix)

参数说明

参数名	输入/输出	说明
point_pair_info	输入	三对输入、输出图片中的像素点坐标信息，每对中分别包含一个输入图片像素点坐标、一个输出图片像素点坐标。
matrix	输出	仿射变换需要的变换矩阵。

返回值说明

- 0：成功
- 非0：失败，11.18.21.10 VPC图像处理返回码

11.18.16.44 hi_mpi_vpc_get_rotation_matrix

函数功能

获取仿射变换需要的旋转矩阵。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpc_get_rotation_matrix(hi_float_point center_point, hi_double angle, hi_double scale, hi_transform_matrix *matrix)

参数说明

参数名	输入/输出	说明
center_point	输入	输入图像的旋转中心点。
angle	输入	旋转角度，正值表示逆时针旋转。
scale	输入	缩放参数。 scale参数值大于1表示放大图片，参数值小于1表示缩小图片。
matrix	输出	仿射变换的旋转矩阵。

返回值说明

- 0：成功
- 非0：失败，11.18.21.10 VPC图像处理返回码

11.18.16.45 hi_mpi_vpc_warp_affine

函数功能

对输入图片做仿射变换。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpc_warp_affine(hi_vpc_chn chn, hi_transform_matrix *matrix, hi_warp_transform_param *transform_param, hi_u32 *task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
matrix	输入	使用hi_mpi_vpc_get_affine_transform或者hi_mpi_vpc_get_rotation_matrix接口获取的仿射变换矩阵。
transform_param	输入	仿射变换参数，在hi_warp_transform_param结构体内配置输入、输出图片信息。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。
task_id	输出	任务ID的指针，用来区分任务。

返回值说明

- 0：成功
- 非0：失败，11.18.21.10 VPC图像处理返回码

11.18.16.46 hi_mpi_vpc_flip

函数功能

对输入图片做翻转处理，支持水平方向翻转、竖直方向翻转和水平竖直方向同时翻转。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vpc_flip(hi_vpc_chn chn, hi_flip_param* flip_param, hi_u32* task_id, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
flip_param	输入	翻转信息。 在hi_flip_param结构体内配置输入、输出图片信息以及翻转模式。
milli_sec	输入	超时时间配置，单位是毫秒，取值范围如下： ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。
task_id	输出	任务ID的指针，用来区分任务。

返回值说明

- 0：成功
- 非0：失败，11.18.21.10 VPC图像处理返回码

11.18.16.47 hi_mpi_vpc_set_chn_opt_attr

函数功能

设置通道可选属性，目前仅支持设置通道优先级。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 针对同一通道，本接口只允许在创建通道后调用一次，对同一通道重复调用将会返回失败；
- 通道优先级只能将通道设置为高优先级，不调用本接口则默认为低优先级；

  针对不同通道，如果都设置为高优先级，则这些通道的任务按下发顺序调用。
- 高优先级通道中的任务会被优先调度，但不会影响已下发的低优先级通道的任务；

  建议在高优先级通道中只下发需要处理时延敏感的关键任务，避免高优先级通道中的任务长时间占用VPC资源而导致低优先级通道中任务一直排队等待。
- VDEC任务中也会调用VPC资源，其默认使用低优先级，设置VPC的通道优先级也会对VDEC任务中的VPC资源调度产生影响；

函数原型

hi_s32 hi_mpi_vpc_set_chn_opt_attr(hi_vpc_chn chn, const hi_opt_attr* opt_attr)

参数说明

参数名	输入/输出	说明
chn	输入	图片处理通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，通道总数最多256。
opt_attr	输入	通道可选属性信息。

返回值说明

- 0：成功
- 非0：失败，**11.18.21.10 VPC图像处理返回码**

11.18.17 VDEC 视频解码功能/JPEGD 图片解码功能

11.18.17.1 JPEGD 功能及约束说明

Atlas 200/300/500 推理产品上，该功能为预留功能，暂不支持。

功能说明

JPEGD（JPEG Decoder）实现.jpg、.jpeg、.JPG、.JPEG图片文件的解码。

- **JPEGD在解码图片时，支持对图片进行旋转。**

如果输入图片的码流中包含Orientation信息（代表捕获图像时摄像机相对于场景的方向），则JPEGD在解码时会解析Orientation信息，将图片进行90度、180度、270度或镜像旋转。旋转后输出图片的宽stride、高stride、输出内存仍需满足**图片格式、宽高对齐、内存约束**中的要求。如果输入图片的码流异常，导致JPEGD解码时无法读取Orientation信息，则不能实现图片旋转的功能。

- **JPEGD在解码图片时，支持按源图片格式解码。**

  源图片格式解码是指解码前后图片的编码格式保持一致，例如解码前输入图片为jpeg(440)，解码后输出图片为YUV440SP V在前U在后或YUV440SP U在前V在后。

  使用源图片格式解码，有以下方式：

  – 在调用JPEGD解码接口时，直接将输出图片格式配置为HI_PIXEL_FORMAT_UNKNOWN，输出格式默认按源图片格式输出、且是V在前U在后的Semi-Planar格式。例如，JPEG源图片格式为jpeg(440)，输出图片格式配置为HI_PIXEL_FORMAT_UNKNOWN，JPEGD解码后，实际输出图片格式为YUV440SP V在前U在后。

    此种方式，由于不知道输出图片格式，因此需要用户申请尽量大的内存或调用**hi_mpi_dvpp_get_image_info**接口获取解码输出内存大小，防止内存不够，无法存放输出图片。

  – 先调用**hi_mpi_dvpp_get_image_info**接口根据传入的jpeg源图片，获取按源图解码时的输出图片的宽、高、宽stride、高stride、解码输出内存大小、图片格式等信息后，再调用JPEGD解码接口，使用通过**hi_mpi_dvpp_get_image_info**接口获取的图片格式来设置输出图片格式。

说明

JPEGD解码后的输出图片，如果要直接作为模型推理的输入，建议将输出图片格式配置为HI_PIXEL_FORMAT_UNKNOWN，这时JPEGD使用源图片格式解码（但这里要确保解码后的图片格式模型是支持的），保证模型推理的精度。

JPEGD解码后的输出图片，如果直接作为VPC的输入，该场景下若使用源图片格式解码时，则需要关注解码后的输出图片格式VPC是否支持（VPC输入图片的格式请参见**11.18.16.2 约束说明**），如果VPC不支持，则用户需按VPC支持的情况指定JPEGD的输出图片格式。

图片分辨率约束

- 输入图片分辨率

版本	分辨率范围
Atlas 推理系列产品	最大分辨率：8192*8192，最小分辨率：32*32。

- 输出图片分辨率

  JPEGD只对图片解码，不会改变图片分辨率，因此输出与输入的图片分辨率保持一致。

内存申请/释放接口说明

输入内存的大小就是指实际的输入图片所占用的大小，输出内存的大小参见**表11-47**中的计算公式。

版本	内存申请/释放接口
Atlas 推理系列产品	JPEGD图片解码时，调用**hi_mpi_dvpp_malloc****hi_mpi_dvpp_free**接口申请\释放Device上的输入、输出内存，这部分内存的生命周期由用户自行管理。

图片格式、宽高对齐、内存约束

实现JPEGD图片解码功能时，仅支持Huffman编码，压缩前的原图像色彩空间为YUV，像素的各分量比例为4:4:4或4:2:2或4:2:0或4:0:0或4:4:0，不支持算术编码、不支持渐进JPEG格式、不支持JPEG2000格式。

说明

输出图片格式的定义请参见**hi_pixel_format**，宽stride、高stride等概念请参见**11.18.2 基本概念**。

表 11-47 图片格式、宽高对齐、内存大小约束

输入图片格式（YUV分量比例）	输出图片格式	输出图片宽、高对齐要求	输出图片宽stride、高stride、内存大小要求
jpeg(444)	YVU444SP 8bit	无对齐要求	宽stride为宽64对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 3
	YUV444SP 8bit		
	YUV420SP NV12 8bit	宽2对齐 高2对齐	宽stride为宽64对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 3/2
	YUV420SP NV21 8bit		
jpeg(422)	YVU422SP 8bit	宽2对齐 高无对齐要求	宽stride为宽64对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 2
	YUV422SP 8bit		
	YUV420SP NV12 8bit	宽2对齐 高2对齐	宽stride为宽64对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 3/2
	YUV420SP NV21 8bit		
jpeg(420)	YUV420SP NV12 8bit	宽2对齐 高2对齐	宽stride为宽64对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 3/2
	YUV420SP NV21 8bit		

输入图片格式（YUV分量比例）	输出图片格式	输出图片宽、高对齐要求	输出图片宽stride、高stride、内存大小要求
jpeg(400)	YUV420SP NV12 8bit	宽2对齐 高2对齐	宽stride为宽64对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 3/2
	YUV420SP NV21 8bit		
	YUV400 8bit	无对齐要求	宽stride为宽64对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride
jpeg(440)	YVU440SP 8bit	宽无对齐要求 高2对齐	宽stride为宽64对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 2
	YUV440SP 8bit		
	YUV420SP NV12 8bit	宽2对齐 高2对齐	宽stride为宽64对齐后的值。 高stride为高16对齐后的值。 内存大小（单位Byte）≥ 宽stride * 高stride * 3/2
	YUV420SP NV21 8bit		

软、硬件约束

- **硬件约束：**
  - 最多支持4张Huffman表，其中包括2 张DC（Direct Current）（直流）表和2 张AC（Alternating Current）（交流）表；
  - 最多支持3张量化表；
  - 只支持8bit采样精度；
  - 只支持对顺序式编码的图片进行解码；
  - 只支持基于DCT（Discrete Cosine Transform）变换的JPEG 格式解码；
  - 只支持一个SOS（Start of Scan）标志的图片解码。
- **软件约束：**
  - 支持3个SOS标志的图片解码；
  - 支持mcu（Minimum Coded Unit）数据不足的异常图片解码。

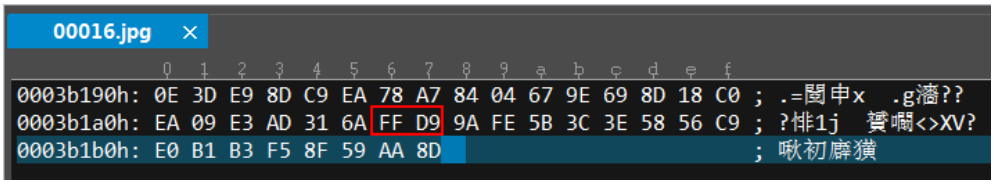
精度相关约束

JPEGD+VPC串联使用时，由于JPEGD解码后的输出图片的宽stride*高stride有64*16对齐的约束，因此解码后的输出图片的宽、高有一些补边的无效数据，所以在执行VPC功能时（例如缩放时），需在输入图片的hi_vpc_pic_info.picture_width、hi_vpc_pic_info.picture_height参数处正确设置输入图片的原图宽高，这样VPC在缩放图片前会先根据原图宽高自行抠图，目的是去除无效数据对图像精度的影响。

其它注意事项

若图片内EOI（End Of Image，标记代码为0XFFD9）之后，还有用户自定义的数据，则JPEGD在对图片进行解码时，会直接清零EOI之后的8字节数据，若用户需要保留这些自定义的数据，则将图片数据读入内存之后，需要提前备份这部分数据，再传给JPEGD处理。

若需要查看图片内EOI之后是否存在自定义数据，可以使用二进制查看工具打开图片查看，例如下图中的FFD9标记符之后就存在自定义数据。



11.18.17.2 JPEGD 性能指标数据

Atlas 200/300/500 推理产品上，该功能为预留功能，暂不支持。

性能指标说明

本节仅描述各产品对应的昇腾AI处理器的性能指标参考数据，各产品由于其内存、CPU等配置不同，在性能规格上会存在差异，详细产品的性能规格请查阅昇腾社区的官方说明。

本节以单个昇腾AI处理器为例，描述基本场景性能指标参考，如下表所示，其它分辨率可以等量估算。

下表中的1路对应一个通道，一个通道对应一个线程；各分辨率的含义为：720p指分辨率为1280*720的图片，1080p指分辨率为1920*1080的图片，4K指分辨率为3840*2160的图片。

JPEGD性能指标是基于硬件解码的性能，JPEGD硬件解码不支持3个SOS的图片解码，对于硬件不支持的格式，会使用软件解码，软件解码性能参考为1080P 15fps。JPEGD解码的输出图片如果涉及旋转，则性能指标低于软件解码的参考值，例如对于1080P的图片，性能指标低于15fps。

表 11-48 Atlas 推理系列产品的昇腾 AI 处理器性能指标参考数据

场景举例	总帧率
1080p*n路（1≤n≤16）	n*128fps
1080p*n路（n>16）	2048fps
4k*n路（1≤n≤16）	n*32fps
4k*n路（n>16）	512fps

11.18.17.3 VDEC 功能及约束说明

Atlas 200/300/500 推理产品上，该功能为预留功能，暂不支持。

功能说明

VDEC（Video Decoder）实现视频的解码。

分辨率约束

- 输入码流分辨率：

版本	分辨率范围
Atlas 推理系列产品	最大分辨率4096*4096，最小分辨率128*128。

- 输出图片分辨率：

版本	分辨率范围
Atlas 推理系列产品	最大分辨率4096*4096，最小分辨率10*6。

内存申请/释放接口说明

输入内存的大小就是指实际的输入码流所占用的大小，输出内存的大小参见**表11-49**中的计算公式。

版本	内存申请/释放接口
Atlas 推理系列产品	VDEC视频解码时，调用**hi_mpi_dvpp_malloc****hi_mpi_dvpp_free**接口申请\释放Device上的输入、输出内存，这部分内存的生命周期由用户自行管理。

码流/图片格式、宽高对齐、内存约束

- 输入码流格式
  - H264 bp/mp/hp level5.1 YUV420编码的码流，当前只支持annex-B的裸码流。
  - H265 8/10bit level5.1 YUV420编码的码流，当前只支持annex-B的裸码流。
- 输出图片要求：

表 11-49 码流/图片格式、宽高对齐、内存大小约束

输出图片格式	输出图片宽、高对齐要求	输出图片宽stride、高stride、内存大小要求 Atlas 推理系列产品
YUV420SP NV12 8bit	宽2对齐 高2对齐	宽stride为宽16对齐后的值，最小32，最大16384。 高stride为高2对齐后的值，最小6，最大16384。 内存大小（单位Byte）≥ 宽stride * 高stride * 3/2
YUV420SP NV21 8bit	宽2对齐 高2对齐	
RGB888	无对齐要求	宽stride为宽16对齐后再乘以3的值，最小48，最大16384。 高stride无对齐要求，最小6，最大16384。 内存大小（单位Byte）≥ 宽stride * 高stride
BGR888	无对齐要求	

说明

输出图片格式的定义请参见**hi_pixel_format**，宽stride、高stride等概念请参见**11.18.2 基本概念**。

其它约束

- VDEC只支持对按帧输入码流进行解码。
- 若码流中有坏帧、缺帧等情况，解码器VDEC解码时会将该帧标记为解码失败，并上报异常。
- 通过隔行扫描方式编码出来的码流，VDEC仅支持解码H264 8bit编码的码流。
- 同时配置按帧发送码流与按解码序输出，可以达到快速解码和快速输出的目的，但这种场景不支持解码含有B帧的码流。

11.18.17.4 VDEC 性能指标数据

Atlas 200/300/500 推理产品上，该功能为预留功能，暂不支持。

性能指标说明

本节仅描述各产品对应的昇腾AI处理器的性能指标参考数据，各产品由于其内存、CPU等配置不同，在性能规格上会存在差异，详细产品的性能规格请查阅昇腾社区的官方说明。

本节以单个昇腾AI处理器为例，描述基本场景性能指标参考，如下表所示，其它分辨率可以等量估算。

下表中的1路对应一个通道，一个通道对应一个线程；各分辨率的含义为：720p指分辨率为1280*720的图片，1080p指分辨率为1920*1080的图片，4K指分辨率为3840*2160的图片。

表 11-50 Atlas 推理系列产品的昇腾 AI 处理器性能指标参考数据

场景举例	总帧率
720p*n路(1≤n≤12)	n*480fps **说明** 针对 Atlas 300V 视频解析卡，当1<n≤12时，总帧率都为n*400fps。
720p*n路(n>12)	5760fps **说明** 针对 Atlas 300V 视频解析卡，当n>12时，总帧率都为4800fps。
1080p*n路(1≤n≤12)	n*240fps **说明** 针对 Atlas 300V 视频解析卡，当1<n≤12时，总帧率都为n*200fps。
1080p*n路(n>12)	2880fps **说明** 针对 Atlas 300V 视频解析卡，当n>12时，总帧率都为2400fps。
4k*n路(1≤n≤12)	n*60fps
4k*n路(n>12)	720fps

下表以1080P分辨率的输入码流为例，说明每路VDEC解码的最大内存消耗的计算公式，在计算公式中：

- 输入码流缓存大小：大于或等于解码通道大小（宽*高）的 3/4倍。
- 解码图像帧存大小：可调用**hi_vdec_get_pic_buf_size**接口获取解码图像帧存大小，该参数值跟输入码流分辨率相关。
- 视频解码图像Tmv缓存大小：可调用**hi_vdec_get_tmv_buf_size**接口获取视频解码图像Tmv缓存大小，该参数值跟输入码流分辨率相关。
- 输入码流缓存大小、解码图像帧存大小、视频解码图像Tmv缓存大小、参考帧数量均由用户调用**hi_mpi_vdec_create_chn**接口创建通道时设置。
- 解码后缓存图像帧数由用户在调用**hi_mpi_vdec_set_chn_param**接口时设置。

每路VDEC解码的内存消耗计算公式	场景举例	内存消耗（单位为MB）
6MB + *输入码流缓存大小* * 2 + (*解码图像帧存大小* + *视频解码图像Tmv缓存大小*) * (*参考帧数量* + *解码后缓存图像帧数* +1)	● 输入码流格式H264 ● 输入码流分辨率1080P ● 输入码流缓存大小为4MB ● 解码图像帧存大小为3MB ● 视频解码图像Tmv缓存大小为0.5MB ● 解码后缓存图像帧数2个	52.5MB（参考帧数量为8） 31.5MB（参考帧数量为2）
	● 输入码流格式H265 ● 输入码流分辨率1080P ● 输入码流缓存大小为4MB ● 解码图像帧存大小为3MB ● 视频解码图像Tmv缓存大小为1MB ● 解码后缓存图像帧数2个	58MB（参考帧数量为8） 34MB（参考帧数量为2）

11.18.17.5 hi_mpi_vdec_create_chn

函数功能

根据设置的通道属性创建解码通道。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 单个Device上的通道号不能超出最大的通道号范围。
- 在创建解码通道之前必须保证通道未创建（或者已经销毁），否则会直接返回错误。
- 系统内存不足时会返回HI_ERR_VDEC_NO_MEM的错误码，可考虑扩展操作系统内存。
- 如果需要解码的H.264码流有B帧，或者需要解码的H.265码流支持时域运动矢量预测（sps_temporal_mvp_enabled_flag = 1），则创建通道时需要设置此通道支持时域运动矢量预测（temporal_mvp_en 设置为 1），还需要为其分配输出每一帧Tmv（Temporal Motion Vector）信息的VB（Vedio Buffer）块，该VB块的大

小比图像VB块小很多，所需个数为RefFrameNum+1，具体大小通过调用函数**hi_vdec_get_tmv_buf_size**获取，否则会导致解码出现花屏等错误。

- H264、H265解码每个解码通道所需VB个数至少为参考帧+显示帧+1，JPEG解码每个解码通道所需VB个数至少为显示帧+1。不同协议解码所需的图像VB块大小不同，具体大小通过调用函数**hi_vdec_get_pic_buf_size**获取。
- 如果 H.264解码不需要解码B帧，或者 H.265解码不需要解码支持时域运动矢量预测（sps_temporal_mvp_enabled_flag = 1）的码流，则创建通道时可设置此通道不支持时域运动矢量预测（temporal_mvp_en 设置为 0），此种情况不输出Tmv信息，可以不用创建Tmv VB池，节省内存。
- 用户需要根据解码码流配置解码所需的帧存VB大小 frame_buf_size 和个数 frame_buf_cnt，以及存放Tmv信息的VB大小 tmv_buf_size，解码器内部按此配置创建相应的私有VB池。
- 只要帧存buffer和Tmvbuffer足够大，解码通道能解码在最大最小分辨率范围内的任意分辨率码流。通道宽高当前只与码流buffer、SCDbuffer（start code detector buffer）大小有关。

函数原型

hi_s32 hi_mpi_vdec_create_chn(hi_vdec_chn chn, const hi_vdec_chn_attr *attr)

参数说明

参数名	输入/输出	说明
chn	输入	解码通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，JPEGD功能和VDEC功能共用通道，且通道总数最多256。 **说明** **对于Atlas 推理系列产品，**昇腾虚拟化实例（Ascend Virtual Instance）场景下，VDEC通道和JPEGD通道总数 =（被分配的VDEC硬件单元+JPEGD硬件单元数量之和 / VDEC硬件单元+JPEGD硬件单元总数之和）* 256，如果通道总数不为整数，则向下取整。 昇腾虚拟化实例（Ascend Virtual Instance）的详细描述，以及各场景下的算力资源规格请参见《昇腾虚拟化实例（AVI）用户指南》。
attr	输入	解码通道属性指针。 如果参数attr为空，会返回错误码HI_ERR_VDEC_NULL_PTR。 当通道属性attr中的值超过解码能力集时，会返回HI_ERR_VDEC_ILLEGAL_PARAM的错误码。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.11 VDEC视频解码/JPEGD图片解码返回码**

参考资源

接口调用流程及示例，参见**5.5.2 JPEGD图片解码**、**5.5.5 VDEC视频解码**。

11.18.17.6 hi_mpi_vdec_destroy_chn

函数功能

销毁解码通道，以释放相关资源。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 销毁前必须保证通道已创建，否则会返回通道未创建的错误。
- 销毁前必须停止接收码流（或者尚未开始接收码流），否则返回错误码HI_ERR_VDEC_NOT_PERM。

函数原型

hi_s32 hi_mpi_vdec_destroy_chn(hi_vdec_chn chn)

参数说明

参数名	输入/输出	说明
chn	输入	解码通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，JPEGD功能和VDEC功能共用通道，且通道总数最多256。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.11 VDEC视频解码/JPEGD图片解码返回码**

参考资源

接口调用流程及示例，参见**5.5.2 JPEGD图片解码**、**5.5.5 VDEC视频解码**。

11.18.17.7 hi_mpi_vdec_get_chn_attr

函数功能

获取视频解码通道属性。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

获取属性前必须保证通道已创建，否则会返回通道未创建的错误码HI_ERR_VDEC_UNEXIST。

函数原型

hi_s32 hi_mpi_vdec_get_chn_attr(hi_vdec_chn chn, hi_vdec_chn_attr *attr)

参数说明

参数名	输入/输出	说明
chn	输入	解码通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，JPEGD功能和VDEC功能共用通道，且通道总数最多256。
attr	输出	解码通道属性指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.11 VDEC视频解码/JPEGD图片解码返回码**

11.18.17.8 hi_mpi_vdec_set_chn_attr

函数功能

设置解码通道属性。不支持该接口。

约束说明

- 获取属性前必须保证通道已创建，否则会返回通道未创建的错误码HI_ERR_VDEC_UNEXIST。
- 可以改变的属性有type、mode、ref_frame_num，其它属性不能改变。其中，type只支持HI_PT_H264、HI_PT_H265。
- 解码器内部会在改变属性时会自动复位解码通道。

函数原型

hi_s32 hi_mpi_vdec_set_chn_attr(hi_vdec_chn chn, const hi_vdec_chn_attr *attr)

参数说明

参数名	输入/输出	说明
chn	输入	解码通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，JPEGD功能和VDEC功能共用通道，且通道总数最多256。
attr	输入	解码通道属性指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.11 VDEC视频解码/JPEGD图片解码返回码**

11.18.17.9 hi_mpi_vdec_start_recv_stream

函数功能

创建通道之后，启动解码之前，解码器开始接收用户发送的码流。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 启动接收码流前必须保证通道已创建，否则会返回通道未创建的错误码HI_ERR_VDEC_UNEXIST。
- 启动接收码流之后，才能调用**hi_mpi_vdec_send_stream**发送码流。
- 重复调用启动接收码流接口时，返回成功。

函数原型

hi_s32 hi_mpi_vdec_start_recv_stream(hi_vdec_chn chn)

参数说明

参数名	输入/输出	说明
chn	输入	解码通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，JPEGD功能和VDEC功能共用通道，且通道总数最多256。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.11 VDEC视频解码/JPEGD图片解码返回码**

参考资源

接口调用流程及示例，参见**5.5.2 JPEGD图片解码**、**5.5.5 VDEC视频解码**。

11.18.17.10 hi_mpi_vdec_stop_recv_stream

函数功能

码流发送结束之后，解码器停止接收用户发送的码流，在销毁解码通道之前**必须**调用此接口。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 停止接收码流前必须保证通道已创建，否则会返回通道未创建的错误码HI_ERR_VDEC_UNEXIST。
- 调用此接口后，再调用发送码流接口**hi_mpi_vdec_send_stream**会返回失败。
- 重复调用停止接收码流接口时，返回成功。

函数原型

hi_s32 hi_mpi_vdec_stop_recv_stream(hi_vdec_chn chn)

参数说明

参数名	输入/输出	说明
chn	输入	解码通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，JPEGD功能和VDEC功能共用通道，且通道总数最多256。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.11 VDEC视频解码/JPEGD图片解码返回码**

参考资源

接口调用流程及示例，参见**5.5.2 JPEGD图片解码**、**5.5.5 VDEC视频解码**。

11.18.17.11 hi_mpi_vdec_query_status

函数功能

解码过程中通过此接口来查询解码通道状态。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

查询通道状态前必须保证通道已创建，否则会返回通道未创建的错误码HI_ERR_VDEC_UNEXIST。

函数原型

hi_s32 hi_mpi_vdec_query_status(hi_vdec_chn chn, hi_vdec_chn_status *status)

参数说明

参数名	输入/输出	说明
chn	输入	解码通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，JPEGD功能和VDEC功能共用通道，且通道总数最多256。
status	输出	通道状态的指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.11 VDEC视频解码/JPEGD图片解码返回码**

11.18.17.12 hi_mpi_vdec_reset_chn

函数功能

解码通道长时间没有反应的情况下或者解码失败等情况下，可以调用此接口复位通道，尝试恢复。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 复位前必须保证通道已创建，否则会返回错误码HI_ERR_VDEC_UNEXIST。
- 复位前必须停止接收码流，否则返回错误码HI_ERR_VDEC_NOT_PERM。

函数原型

hi_s32 hi_mpi_vdec_reset_chn(hi_vdec_chn chn)

参数说明

参数名	输入/输出	说明
chn	输入	解码通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，JPEGD功能和VDEC功能共用通道，且通道总数最多256。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.11 VDEC视频解码/JPEGD图片解码返回码

11.18.17.13 hi_mpi_vdec_set_chn_param

函数功能

改变通道参数的时候调用此接口，设置解码通道参数。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 必须保证通道已创建，否则会返回错误码HI_ERR_VDEC_UNEXIST。
- 该接口可设置通道的一些高级属性，创建通道后有默认值，默认值见数据类型hi_vdec_chn_param的说明。
- 各项参数如超过合法范围，会返回 HI_ERR_VDEC_ILLEGAL_PARAM 的错误码，各参数范围见数据类型hi_vdec_chn_param的说明。
- 调用hi_mpi_vdec_create_chn接口创建通道时设置按帧发送码流（目前仅支持按帧发送），同时，调用本接口设置解码序输出码流，即将本接口的chn_param->video_param.out_order参数设置为HI_VIDEO_OUT_ORDER_DEC，可以达到快速解码和快速输出的目的，但这种场景不支持解码含有B帧的码流。

函数原型

hi_s32 hi_mpi_vdec_set_chn_param(hi_vdec_chn chn, const hi_vdec_chn_param *chn_param)

参数说明

参数名	输入/输出	说明
chn	输入	解码通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，JPEGD功能和VDEC功能共用通道，且通道总数最多256。

参数名	输入/输出	说明
chn_param	输入	解码通道参数指针。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.11 VDEC视频解码/JPEGD图片解码返回码

参考资源

接口调用流程及示例，参见5.5.2 JPEGD图片解码、5.5.5 VDEC视频解码。

11.18.17.14 hi_mpi_vdec_get_chn_param

函数功能

修改通道参数前可先调用此接口获取通道参数。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

必须保证通道已创建，否则会返回错误码HI_ERR_VDEC_UNEXIST。

函数原型

hi_s32 hi_mpi_vdec_get_chn_param(hi_vdec_chn chn, hi_vdec_chn_param *chn_param)

参数说明

参数名	输入/输出	说明
chn	输入	解码通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，JPEGD功能和VDEC功能共用通道，且通道总数最多256。
chn_param	输出	解码通道参数指针。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.11 VDEC视频解码/JPEGD图片解码返回码

参考资源

接口调用流程及示例，参见5.5.2 JPEGD图片解码、5.5.5 VDEC视频解码。

11.18.17.15 hi_mpi_vdec_set_protocol_param

函数功能

设置协议相关的内存分配通道参数，如slice、pps（Picture Parameter Set）、sps（Sequence Parameter Set）等。

不支持该接口。

约束说明

- 必须保证通道已创建，否则会返回错误码HI_ERR_VDEC_UNEXIST。
- 设置协议参数前必须先停止接收码流，否则返回错误码HI_ERR_VDEC_NOT_PERM。
- 切换协议参数时，解码器内部会复位解码通道。
- 协议参数所需的内存是解码器内部根据当前码流实际vps（Video Parameter Set）/sps/pps/slice个数使用vmalloc动态分配的，最大上限不超过此接口设置的最大vps/sps/pps/slice个数。

函数原型

hi_s32 hi_mpi_vdec_set_protocol_param(hi_vdec_chn chn, const hi_vdec_protocol_param *protocol_param)

参数说明

参数名	输入/输出	说明
chn	输入	解码通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，JPEGD功能和VDEC功能共用通道，且通道总数最多256。
protocol_param	输入	解码通道参数指针。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.11 VDEC视频解码/JPEGD图片解码返回码

11.18.17.16 hi_mpi_vdec_get_protocol_param

函数功能

获取协议相关的内存分配通道参数，如slice、pps（Picture Parameter Set）、sps（Sequence Parameter Set）等。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

必须保证通道已创建，否则会返回错误码HI_ERR_VDEC_UNEXIST。

函数原型

hi_s32 hi_mpi_vdec_get_protocol_param(hi_vdec_chn chn, hi_vdec_protocol_param *protocol_param)

参数说明

参数名	输入/输出	说明
chn	输入	解码通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，JPEGD功能和VDEC功能共用通道，且通道总数最多256。
protocol_param	输出	解码通道参数指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.11 VDEC视频解码/JPEGD图片解码返回码**

11.18.17.17 hi_mpi_vdec_send_stream

函数功能

解码前，向解码通道发送码流数据及存放解码结果的buffer。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 发送数据前必须保证已经调用**hi_mpi_vdec_start_recv_stream**接口启动接收码流，否则直接返回该操作不允许的错误码HI_ERR_VDEC_NOT_PERM。如果在发送数据过程中停止接收码流，就会立刻返回HI_ERR_VDEC_NOT_PERM。
- 发送数据前必须保证通道已经被创建，否则直接返回通道未创建的错误码HI_ERR_VDEC_UNEXIST。如果在发送码流过程中复位通道或者销毁通道，就会立刻返回错误码HI_ERR_VDEC_UNEXIST。

- 发送码流时需要按照创建解码通道时设置的发送方式（仅支持按帧发送）进行发送。按帧发送时，调用此接口一次，必须发送完整的一帧码流，否则，解码会出现错误。
- 不能发送end_of_stream为0的空码流包（码流长度为0或码流地址为空），否则返回错误码。发送end_of_stream为1的空码流包时，解码器会把所有码流全部解完并输出全部图像。除此之外，其它情况应该把end_of_stream置为0。
- 当码流buffer为空且装不下当前包码流时，会返回参数超出范围的错误码HI_ERR_VDEC_ILLEGAL_PARAM。
- 以非阻塞方式发送码流，如果码流缓冲区已满，会立刻返回错误码HI_ERR_VDEC_BUF_FULL。
- 以超时方式发送码流，到达设定的超时时间还不能成功发送码流会返回错误码HI_ERR_VDEC_BUF_FULL。
- 视频解码时支持设置是否将解码结果写入输出内存中，hi_vdec_stream结构体中将need_display设置为0时，此帧码流的解码结果将不会写入到hi_vdec_pic_info结构体中配置的输出内存中，此时允许将输出内存地址配置为NULL。**hi_mpi_vdec_get_frame**接口仍然能获取到结构hi_video_frame_info和hi_vdec_stream的信息，但是此时输出内存中无解码结果数据，需要用户释放输出内存（如果用户传入的输出内存地址是NULL，则无需释放）。
- 解码时，输入输出内存均需要在调用**hi_mpi_vdec_get_frame**接口获取结果之后才能进行释放。
- 视频解码，输入输出内存要求、输入码流格式、输出图片格式请参见**11.18.17.3 VDEC功能及约束说明**。
- 图像解码，输入输出内存要求、输入图片格式、输出图片格式请参见**11.18.17.1 JPEGD功能及约束说明**。

函数原型

hi_s32 hi_mpi_vdec_send_stream(hi_vdec_chn chn, const hi_vdec_stream *stream, hi_vdec_pic_info *vdec_pic_info, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	解码通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，JPEGD功能和VDEC功能共用通道，且通道总数最多256。
stream	输入	输入码流信息的指针。 该结构体内的addr参数配置的地址为Device上的内存地址。
vdec_pic_info	输入	输出图片信息的指针。

参数名	输入/输出	说明
milli_sec	输入	超时时间，单位是毫秒。 ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.11 VDEC视频解码/JPEGD图片解码返回码**

参考资源

接口调用流程及示例，参见**5.5.2 JPEGD图片解码**、**5.5.5 VDEC视频解码**。

11.18.17.18 hi_mpi_vdec_get_frame

函数功能

解码后，获取解码通道的解码图像及输入Stream。

JPEGD图像解码场景下，在调用本接口时，如果硬件解码失败，则会在本接口内部触发软件解码。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 向VDEC获取解码结果，输入buffer和输出buffer一起获取。
- JPEGD图片解码时，默认对于jpeg(420)、jpeg(422)、jpeg(440)格式的输入图片，如果输入图片的宽高为奇数，JPEGD解码输出图片的宽高为向下2对齐；默认对于jpeg(444)、jpeg(400)格式的输入图片，JPEGD解码输出图片的宽高与输入图片保持一致。若需调整输出图片的宽、高对齐模式，可调用**hi_mpi_vdec_set_jpegd_precision_mode**接口。

函数原型

hi_s32 hi_mpi_vdec_get_frame(hi_vdec_chn chn, hi_video_frame_info *frame_info, hi_vdec_supplement_info *supplement, hi_vdec_stream *stream, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	解码通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，JPEGD功能和VDEC功能共用通道，且通道总数最多256。
frame_info	输出	已解码的图像信息的指针。解码后的数据存放在Device内存中。
supplement	输出	解码图像补充信息的指针。预留参数，暂未使用，建议用户设置为NULL。
stream	输出	已解码的输入码流信息指针。
milli_sec	输入	超时时间，单位是毫秒。 ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。 **说明** JPEGD图像解码场景下，此处配置的超时时间仅指获取硬件解码结果的等待时间，不包含软件解码的时间。

返回值说明

- 0：成功，获取解码数据成功后，需调用**hi_mpi_vdec_release_frame**接口释放解码相关资源。

  本接口虽返回成功，但有可能获取到的是解码失败的数据，解码是否成功，需要查看hi_video_frame_info结构体的v_frame成员下的**frame_flag**参数值说明。
- 非0：失败，参见**11.18.21.11 VDEC视频解码/JPEGD图片解码返回码**

  注意：
  - 如果在解码图像时未创建通道，则会返回错误码HI_ERR_VDEC_UNEXIST。
  - 如果在获取图像的过程中销毁通道，则会返回错误码HI_ERR_VDEC_UNEXIST。
  - 如果在获取解码图像的过程中复位通道，则会返回错误码HI_ERR_VDEC_UNEXIST。
  - 以非阻塞方式获取解码图像，如果缓冲区内无图像，则会返回错误码HI_ERR_VDEC_BUF_EMPTY。

- 以超时方式获取解码图像，到达设定的超时时间还不能获取到硬件解码结果则会返回错误码HI_ERR_VDEC_BUF_EMPTY，此时则可增大超时时间，或循环调用本接口等待解码结果。

参考资源

接口调用流程及示例，参见5.5.2 JPEGD图片解码、5.5.5 VDEC视频解码。

11.18.17.19 hi_mpi_vdec_release_frame

函数功能

解码之后，释放资源。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 此接口需与hi_mpi_vdec_get_frame配对使用，获取的数据应当在使用完之后立即释放，如果不及时释放，会导致解码过程阻塞等待资源。
- 释放的数据必须是调用hi_mpi_vdec_get_frame接口从该通道获取的数据，不得对数据信息结构体进行任何修改。

函数原型

hi_s32 hi_mpi_vdec_release_frame(hi_vdec_chn chn, const hi_video_frame_info *frame_info)

参数说明

参数名	输入/输出	说明
chn	输入	解码通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，JPEGD功能和VDEC功能共用通道，且通道总数最多256。
frame_info	输入	解码后的图像信息指针，调用hi_mpi_vdec_get_frame接口获取。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.11 VDEC视频解码/JPEGD图片解码返回码

参考资源

接口调用流程及示例，参见5.5.2 JPEGD图片解码、5.5.5 VDEC视频解码。

11.18.17.20 hi_mpi_vdec_get_fd

函数功能

获取视频解码通道的设备文件句柄。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

在进程退出时会自动关闭文件句柄，回收资源。

函数原型

hi_s32 hi_mpi_vdec_get_fd(hi_vdec_chn chn)

参数说明

参数名	输入/输出	说明
chn	输入	解码通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，JPEGD功能和VDEC功能共用通道，且通道总数最多256。

返回值说明

- 0或正数值：有效返回值
- 非正数值：无效返回值

11.18.17.21 hi_mpi_vdec_close_fd

函数功能

关闭视频解码的设备文件句柄。

不支持该接口。

函数原型

hi_s32 hi_mpi_vdec_close_fd(hi_vdec_chn chn)

参数说明

参数名	输入/输出	说明
chn	输入	解码通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，JPEGD功能和VDEC功能共用通道，且通道总数最多256。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.11 VDEC视频解码/JPEGD图片解码返回码**

11.18.17.22 hi_vdec_get_pic_buf_size

函数功能

获取解码图像需要的Buffer大小。用户在两个地方有使用这个函数：计算视频解码需要的内部缓冲区的大小，在初始化的时候配置对应的通道参数；计算图像解码的输出内存。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

提供给用户来计算输出图片缓冲区大小。

函数原型

hi_u32 hi_vdec_get_pic_buf_size(hi_payload_type type, hi_pic_buf_attr *buf_attr)

参数说明

参数名	输入/输出	说明
type	输入	解码视频类型，当前支持HI_PT_H264、HI_PT_H265、HI_PT_JPEG。
buf_attr	输入	图片缓冲区信息的指针。

返回值说明

图像存储需要的图像大小。

解码视频类型为HI_PT_JPEG（表示JPEGD解码）时，pixel_format传入不支持的格式或者HI_PIXEL_FORMAT_UNKNOWN，返回值为0。

参考资源

接口调用流程及示例，参见5.5.5 VDEC视频解码。

11.18.17.23 hi_vdec_get_tmv_buf_size

函数功能

获取解码图像需要的矢量预测缓冲区的大小。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

提供给用户来计算矢量预测缓冲区大小，在创建通道的时候初始化对应的参数。

函数原型

hi_u32 hi_vdec_get_tmv_buf_size(hi_payload_type type, hi_u32 width, hi_u32 height)

参数说明

参数名	输入/输出	说明
type	输入	解码视频类型。
width	输入	解码图像宽。
height	输入	解码图像高。

返回值说明

图像存储需要矢量预测缓冲区大小。

参考资源

接口调用流程及示例，参见5.5.5 VDEC视频解码。

11.18.17.24 hi_mpi_vdec_set_jpegd_precision_mode

函数功能

设置JPEGD解码输出图片的宽、高对齐模式。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 调用本接口设置对齐模式前，必须保证通道已创建，否则会返回通道未创建的错误码HI_ERR_VDEC_UNEXIST。

- 调用本接口设置对齐模式，仅对JPEGD解码通道有效，其它类型通道返回操作不支持。
- 在调用hi_mpi_vdec_start_recv_stream接口之前调用本接口设置对齐模式，在调用hi_mpi_vdec_start_recv_stream接口之后调用本接口，系统会返回操作不允许的错误提示。

函数原型

hi_s32 hi_mpi_vdec_set_jpegd_precision_mode(hi_vdec_chn chn, const hi_jpegd_precision_mode mode)

参数说明

参数名	输入/输出	说明
chn	输入	解码通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，JPEGD功能和VDEC功能共用通道，且通道总数最多256。
mode	输入	设置JPEGD解码输出图片的宽、高对齐模式。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.11 VDEC视频解码/JPEGD图片解码返回码**

11.18.17.25 hi_mpi_vdec_get_jpegd_output_info

函数功能

图片解码场景下，根据输入码流，获取按源图格式或指定图片格式输出时的图片宽、高、宽stride、高stride、解码后占用的内存大小、图片格式等信息。

该接口会检查图片格式是否支持解码，如果遇到不支持的码流则该接口会返回HI_ERR_VDEC_NOT_SUPPORT错误码，该错误码的详细描述请参见**11.18.21.1 公共返回码**。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

hi_s32 hi_mpi_vdec_get_jpegd_output_info(const hi_vdec_stream *stream, const hi_pixel_format output_format, hi_img_info *img_info)

参数说明

参数名	输入/输出	说明
stream	输入	输入码流信息的指针。 Atlas 推理系列产品，**Control CPU开放形态**下，在Device上调用本接口时，该结构体内的addr参数配置的地址必须是Device上的内存地址。
output_format	输入	输出图片格式。 若按源图格式获取相关信息，则将该参数值设置为HI_PIXEL_FORMAT_UNKNOWN；若按指定格式获取相关信息，则将该参数值设置为对应的图片格式，支持的图片格式请参见**图片格式、宽高对齐、内存约束**。
img_info	输出	输出图片信息结构体，包含图片宽、高、宽stride、高stride、解码后占用的内存大小、图片格式等信息。 其中，输出图片格式，通过img_pixel_format参数返回。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.1 公共返回码**、**11.18.21.11 VDEC视频解码/JPEGD图片解码返回码**。

11.18.17.26 hi_mpi_vdec_set_display_mode

函数功能

若调用**hi_mpi_sys_bind**接口绑定数据接收者VPSS与数据源VDEC后，可调用本接口设置显示模式。不设置显示模式时，默认为回放模式。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

- 设置显示模式前必须保证通道已创建，否则会返回通道未创建的错误码HI_ERR_VDEC_UNEXIT。
- 预览模式（HI_VIDEO_DISPLAY_MODE_PREVIEW）：预览模式下VDEC绑定的直接后级模块（比如 VPSS）以非阻塞方式接收解码图像，即当VPSS的图像Buffer满时（解码帧存个数比 VPSS 缓存队列个数多）， VPSS丢弃VDEC发送过来的图像，以达到不反压VDEC 解码的目的，实现实时预览。需要注意的是， 当解码帧存个数比VPSS缓存队列个数少时，即使开启预览模式， VPSS还是会反压解码。

- 回放模式（HI_VIDEO_DISPLAY_MODE_PLAYBACK）：回放模式下VDEC绑定的直接后级模块（比如 VPSS）以阻塞方式接收解码图像， 即当VPSS的图像Buffer满时，拒绝接收VDEC发送过来的图像， VDEC发现当前图像发送失败后启动图像重新发送机制，直到图像发送成功为止。回放模式下VDEC绑定的直接后级模块能够反压VDE 解码，以达到不丢弃任何一帧解码图像的回放效果。

函数原型

hi_s32 hi_mpi_vdec_set_display_mode(hi_vdec_chn chn, hi_video_display_mode display_mode)

参数说明

参数名	输入/输出	说明
chn	输入	解码通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，JPEGD功能和VDEC功能共用通道，且通道总数最多256。
display_mode	输入	显示模式枚举。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.1 公共返回码**、**11.18.21.11 VDEC视频解码/JPEGD图片解码返回码**。

11.18.17.27 hi_mpi_vdec_get_display_mode

函数功能

获取显示模式。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

获取显示模式前必须保证通道已创建，否则会返回通道未创建的错误码HI_ERR_VDEC_UNEXIT。

函数原型

hi_s32 hi_mpi_vdec_get_display_mode(hi_vdec_chn chn, hi_video_display_mode *display_mode)

参数说明

参数名	输入/输出	说明
chn	输入	解码通道号。 Atlas 推理系列产品，该参数的取值范围：[0, 256)，JPEGD功能和VDEC功能共用通道，且通道总数最多256。
display_mode	输入	显示模式枚举指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.1 公共返回码**、**11.18.21.11 VDEC视频解码/JPEGD图片解码返回码**。

11.18.18 VENC 视频编码功能/JPEGE 图片编码功能

11.18.18.1 JPEGE 功能及约束说明

Atlas 200/300/500 推理产品上，该功能为预留功能，暂不支持。

功能说明

JPEGE（JPEG Encoder）将YUV格式图片编码成JPEG压缩格式的图片文件，例如*.jpg。

图片分辨率约束

- 输入图片分辨率：

版本	分辨率范围
Atlas 推理系列产品	最大分辨率：8192*8192，最小分辨率：32*32。

- 输出图片分辨率

  JPEGE只对图片编码，不会改变图片分辨率，因此输出与输入的图片分辨率保持一致。

内存申请/释放接口说明

输入内存的大小就是指实际的输入码流所占用的大小，输出内存的大小参见**表11-51**中的计算公式。

版本	输入内存申请/释放接口	输出内存申请/释放接口
Atlas 推理系列产品	JPEGE图片编码功能时，调用**hi_mpi_dvpp_malloc****hi_mpi_dvpp_free**接口申请\释放Device上的输入内存，这部分内存的生命周期由用户自行管理。	支持以下两种方式： ● 不需要用户管理、**由DVPP内部管理输出内存**时，调用**hi_mpi_venc_send_frame**接口发送原始图像进行图像编码。在调用**hi_mpi_venc_create_chn**接口创建通道时，必须正确设置hi_venc_chn_attr.venc_attr.buf_size参数值（参数描述请参见**11.18.20.14.5 hi_venc_attr**）。 **该方式下，相比由用户管理内存，输出结果数据的JPEG头中不存在COM注释字段，数据长度会短一点，但需要用户从DVPP返回的内存中拷贝输出结果数据到指定内存。** ● **由用户自行管理输出内存**、管理内存的生命周期，调用**hi_mpi_venc_send_jpege_frame**接口发送原始图像进行图像编码。在调用**hi_mpi_venc_create_chn**接口创建通道时，需将hi_venc_chn_attr.venc_attr.buf_size参数值设置为0（参数描述请参见**11.18.20.14.5 hi_venc_attr**），然后调用**hi_mpi_venc_get_jpege_predicted_size**接口预估输出内存大小，调用**hi_mpi_dvpp_malloc****hi_mpi_dvpp_free**接口申请\释放输出内存。 **该方式下，**直接在调用**hi_mpi_venc_send_jpege_frame**接口时，设置输出内存地址，输出结果数据直接存放到用户设置的内存中，**相比由系统管理内存的方式，用户可减少一次“从DVPP返回的内存中拷贝输出结果数据到指定内存”的操作，但输出结果数据的JPEG头中可能会存在COM注释字段（字段长度范围4~19Byte），数据长度会长一点。**

图片格式、宽高对齐、内存约束

表 11-51 图片格式、宽高对齐、内存大小约束

输入图片格式	输入图片宽、高对齐要求	输入图片宽stride、高stride、内存大小要求	输出图片格式
YUV420SP NV12 8bit	宽2对齐 高2对齐	宽stride为宽16对齐后的值，兼容对齐到16的倍数如128（按128对齐时，性能更优）。 高stride无需设置。 内存大小（单位Byte）= 宽stride * 高 * 3/2	JPEG压缩格式的图片文件，例如*.jpg。 只支持huffman编码，不支持算术编码，不支持渐进编码。
YUV420SP NV21 8bit			
YUV422Packed YUYV 8bit	宽2对齐 高无对齐要求	宽stride为宽的两倍再16对齐后的值。 高stride无需设置。 内存大小（单位Byte）= 宽stride * 高	
YUV422Packed UYVY 8bit			
YUV422Packed YVYU 8bit			
YUV422Packed VYUY 8bit			

说明

输入图片格式的定义请参见**hi_pixel_format**，宽stride、高stride等概念请参见**11.18.2 基本概念**。

11.18.18.2 JPEGE 性能指标说明

Atlas 200/300/500 推理产品上，该功能为预留功能，暂不支持。

性能指标说明

本节仅描述各产品对应的昇腾AI处理器的性能指标参考数据，各产品由于其内存、CPU等配置不同，在性能规格上会存在差异，详细产品的性能规格请查阅昇腾社区的官方说明。

本节以单个昇腾AI处理器为例，描述基本场景性能指标参考，如下表所示，其它分辨率可以等量估算。

下表中的1路对应一个通道，一个通道对应一个线程；各分辨率的含义为：1080p指分辨率为1920*1080的图片，4K指分辨率为3840*2160的图片。

表 11-52 Atlas 推理系列产品的昇腾 AI 处理器性能指标参考数据

场景举例（输入图片格式：YUV420 8bit）	总帧率
1080p*n路（1≤n≤8）	n*128fps
1080p*n路（n>8）	1024fps
4k*n路（1≤n≤8）	n*32fps
4k*n路（n>8）	256fps

11.18.18.3 VENC 功能及约束说明

Atlas 200/300/500 推理产品上，该功能为预留功能，暂不支持。

功能说明

VENC（Video Encoder）将YUV420SP NV12/NV21-8bit图片数据编码成H264/H265格式的视频码流。

分辨率约束

- 输入图片分辨率：

版本	分辨率范围
Atlas 推理系列产品	宽和高的范围都是128~4096，对于H264格式的码流，最大分辨率的乘积不超过4096*2304。

- 输出码流分辨率

  VENC只对图片编码，不会改变图片分辨率，因此输出与输入的图片分辨率保持一致。

内存申请/释放接口说明

输入内存的大小参见**表11-53**中的计算公式，输出内存无需用户管理、由系统管理。

版本	内存申请/释放接口
Atlas 推理系列产品	VENC视频编码时，调用**hi_mpi_dvpp_malloc****hi_mpi_dvpp_free**接口申请\释放Device上的输入内存，这部分内存的生命周期由用户自行管理。

码流/图片格式、宽高对齐、内存约束

表 11-53 图片格式、宽高对齐、内存大小约束

输入图片格式	输入图片宽、高对齐要求	输入图片宽stride、高stride、内存大小要求	输出码流格式
YUV420SP NV12 8bit	宽2对齐 高2对齐	宽stride为宽16对齐后的值。 高stride无需设置。 内存大小（单位Byte）= 宽stride * 高 * 3/2	• H264 BP/MP/HP • H265 MP（仅支持Slice码流）
YUV420SP NV21 8bit			

说明

输入图片格式的定义请参见**hi_pixel_format**，宽stride、高stride等概念请参见**11.18.2 基本概念**。

11.18.18.4 VENC 性能指标说明

Atlas 200/300/500 推理产品上，该功能为预留功能，暂不支持。

性能指标说明

本节仅描述各产品对应的昇腾AI处理器的性能指标参考数据，各产品由于其内存、CPU等配置不同，在性能规格上会存在差异，详细产品的性能规格请查阅昇腾社区的官方说明。

本节以单个昇腾AI处理器为例，描述基本场景性能指标参考，如下表所示，其它分辨率可以等量估算。

下表中的1路对应一个通道，一个通道对应一个线程；各分辨率的含义为：720p指分辨率为1280*720的图片，1080p指分辨率为1920*1080的图片，4K指分辨率为3840*2160的图片。

表 11-54 Atlas 推理系列产品的昇腾 AI 处理器性能指标参考数据

场景举例	总帧率
720p*n路(1≤n≤3)	n*480fps
720p*n路(n>3)	1440fps
1080p*n路(1≤n≤3)	n*240fps
1080p*n路(n>3)	720fps
4k*n路(1≤n≤3)	n*60fps
4k*n路(n>3)	180fps

11.18.18.5 hi_mpi_venc_create_chn

函数功能

创建编码通道。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 单个Device上的通道号不能超出最大的通道号范围。
- 编码通道属性由三部分组成，编码器属性、码率控制器属性和帧结构类型属性，帧结构类型属性简称GOP（Group of Pictures ）类型属性。
- 编码器属性必须设置编码码流buffer深度、获取码流方式、编码profile等，**表11-55**详细描述了各种协议的各项属性的特性。
- 推荐的编码宽高为：3840*2160(4k)、1920*1080(1080P)、1280*720(720P)。
- H.264/H.265编码帧存由YHeaderSize、CHeaderSize、YSize、CSize、PmeSize、PmeInfoSize和TmvSize共同构成，编码器默认根据最大宽高计算帧存进行内存分配，在设置通道宽高时需保证根据通道宽高计算的每一部分帧存大小都不能大于最大宽高计算的帧存大小。
- 码率控制器属性中的输入/输出帧率（src_frame_rate/dst_frame_rate）是用于配合编码过程中的码率（bit_rate）分配，不是最终的播放器显示帧率，若想要设置显示帧率请参考**hi_mpi_venc_set_h264_vui**或**hi_mpi_venc_set_h265_vui**接口。

  建议用户将码控帧率与显示帧率设置一致，否则播放时的码率可能与用户设置的编码码率不同，播放时的码率会随着所设置的显示帧率而变化。

函数原型

hi_s32 hi_mpi_venc_create_chn(hi_venc_chn chn, const hi_venc_chn_attr *attr)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。 **说明** **对于Atlas 推理系列产品，**昇腾虚拟化实例（Ascend Virtual Instance）场景下，VENC通道和JPEGE通道总数 =（被分配的VENC硬件单元+JPEGE硬件单元数量之和 / VENC硬件单元+JPEGE硬件单元总数之和）* 256，如果通道总数不为整数，则向下取整。 昇腾虚拟化实例（Ascend Virtual Instance）的详细描述，以及各场景下的算力资源规格请参见《昇腾虚拟化实例（AVI）用户指南》。
attr	输入	编码通道属性的指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

参考资源

接口调用流程及示例，参见**5.5.3 JPEGE图片编码**、**5.5.6 VENC视频编码**。

参考信息

表 11-55 编码器属性的约束

编码协议	编码方式	码流 buffer 深度	获取码流模式	编码 profile
H.264	Frame	• 当 mini_buf_mode=0时，Buffer≥max_pic_width*max_pic_height*3/4； • 当 mini_buf_mode=1时，Buffer≥32*1024 Byte；	Frame/Slice	Baseline Mainprofile Highprofile

编码协议	编码方式	码流 buffer 深度	获取码流模式	编码 profile
JPEG	Frame	• mini_buf_mode=0 时，如果原图格式为YUV420，Buffer最小值为原图宽16对齐*原图高16对齐*3/2 ；如果原图格式为YUV422 Packed，Buffer最小值为原图宽16对齐*原图高16对齐*2 • mini_buf_mode=1 时，Buffer≥32*1024 Byte；	Frame/Ecs	Baseline
H.265	Frame	• mini_buf_mode=0时，Buffer≥ max_pic_width*max_pic_height*3/4； • mini_buf_mode=1时，Buffer≥32*1024 Byte	Frame/ Slice	Main profile Main 10 profile（暂不支持）
		• mini_buf_mode=0时，Buffer≥ max_pic_width*max_pic_height*(3/4)*(5/4）； • mini_buf_mode=1时，Buffer≥32*1024 Byte	Frame/ Slice	Main 10 profile（暂不支持）

11.18.18.6 hi_mpi_venc_destroy_chn

函数功能

用户退出编码流程时，需要关闭之前打开的编码通道，以释放相关资源。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 销毁并不存在的通道，返回失败。
- 销毁前必须停止接收图像，否则返回失败。

函数原型

hi_s32 hi_mpi_venc_destroy_chn(hi_venc_chn chn)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

参考资源

接口调用流程及示例，参见**5.5.3 JPEGE图片编码**、**5.5.6 VENC视频编码**。

11.18.18.7 hi_mpi_venc_start_chn

函数功能

开启编码通道接收输入图片，允许指定接收帧数，超出指定的帧数后自动停止接收图像。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 如果通道未创建，则返回错误码HI_ERR_VENC_UNEXIST。
- 需要接收的图像帧数设置为-1时表示不指定帧数。
- 如果通道已经开始接收图像，没有停止接收图像前再一次调用此接口指定接收帧数，返回操作不允许。
- 如果通道已经开始接收图像，没有停止接收图像前允许再一次调用此接口不指定接收帧数。
- 只有开启接收之后编码器才开始接收图像编码。

函数原型

hi_s32 hi_mpi_venc_start_chn(hi_venc_chn chn, const hi_venc_start_param *recv_param)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
recv_param	输入	接收图像参数的指针，用于指定需要接收的图像帧数。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

参考资源

接口调用流程及示例，参见**5.5.3 JPEGE图片编码**、**5.5.6 VENC视频编码**。

11.18.18.8 hi_mpi_venc_stop_chn

函数功能

停止编码通道接收输入图片。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 如果通道未创建，则返回失败。
- 此接口并不判断当前是否停止接收，即允许重复停止接收不返回错误。
- 此接口用于编码通道停止接收图像来编码，在编码通道销毁或复位前必须停止接收图像。
- 调用此接口仅停止接收原始数据编码，码流buffer并不会被清除。

函数原型

hi_s32 hi_mpi_venc_stop_chn(hi_venc_chn chn)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

参考资源

接口调用流程及示例，参见**5.5.3 JPEGE图片编码**、**5.5.6 VENC视频编码**。

11.18.18.9 hi_mpi_venc_query_status

函数功能

查询编码通道状态。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 如果通道未创建，则返回失败。
- 此接口用于查询此函数调用时刻的编码器状态，status包含以下信息：
  - 在编码通道状态结构体中，left_pics表示待编码的帧个数。在复位通道前，可以通过查询是否还有图像待编码来决定复位时机，防止复位时将可能需要编码的帧清理出去。
  - 在编码通道状态结构体中，left_stream_bytes表示码流buffer中用户暂未取走的byte数目。

    在复位通道前，可以通过查询是否还有码流没有被处理来决定复位时机，防止复位时将可能需要的码流清理出去。
  - 在编码通道状态结构体中，left_stream_frames表示码流buffer中用户暂未取走的帧数目。

    在复位通道前，可以通过查询是否还有图像的码流没有被取走来决定复位时机，防止复位时将可能需要的码流清理出去。
  - 在编码通道状态结构体中，cur_packs表示当前帧的码流包个数。在调用**hi_mpi_venc_get_stream**之前应确保cur_packs大于0。

    在按包获取时当前帧可能不是一个完整帧（被取走一部分），按帧获取时表示当前一个完整帧的包个数（如果没有一帧数据则为 0）。用户在需要按帧获取码流时，需要查询一个完整帧的包个数，在这种情况下，通常可以在select成功后执行query操作，此时，cur_packs是当前完整帧中包的个数。

- 在编码通道状态结构体中，left_recv_pics表示调用hi_mpi_venc_start_chn接口后剩余等待接收的帧数目。
- 在编码通道状态结构体中，left_enc_pics表示调用hi_mpi_venc_start_chn接口后剩余等待编码的帧数目。
- 如果调用hi_mpi_venc_start_chn但没有指定接收帧数，left_recv_pics和left_enc_pics数目始终为0。

函数原型

hi_s32 hi_mpi_venc_query_status(hi_venc_chn chn, hi_venc_chn_status *status)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
status	输出	编码通道状态的指针。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.12 VENC视频编码/JPEGE图片编码返回码

参考资源

接口调用流程及示例，参见5.5.3 JPEGE图片编码、5.5.6 VENC视频编码。

11.18.18.10 hi_mpi_venc_get_stream

函数功能

获取编码的码流。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 如果通道未创建，返回失败。
- 如果stream为空，返回错误码HI_ERR_VENC_NULL_PTR。
- 如果milli_sec小于-1，返回错误码HI_ERR_VENC_ILLEGAL_PARAM。
- 支持超时方式获取。支持select/poll系统调用。
  - milli_sec = 0 时，则为非阻塞获取，即如果缓冲无数据，则返回错误码HI_ERR_VENC_BUF_EMPTY。

  - milli_sec = -1 时，则为阻塞，即如果缓冲无数据，则会等待有数据时才返回获取成功。
  - milli_sec > 0 时，则为超时，即如果缓冲无数据，则会等待用户设定的超时时间，若在设定的时间内有数据则返回获取成功，否则返回超时失败。
- 支持按包或按帧方式获取码流。如果按包获取，则：
  - 对于JPEG编码协议， 每次获取的是一个参数包或数据包。
  - 对于H.265编码协议，每次获取的是一个 NAL 单元。
- 码流结构体hi_venc_stream包含以下部分：
  - 码流包信息指针pack

    指向一组hi_venc_pack结构体的连续的内存空间，该空间由调用者分配。如果是按包获取，则此空间不小于sizeof（hi_venc_pack）的大小；如果按帧获取，则此空间不小于N*sizeof（hi_venc_pack）的大小，其中，N代表当前帧之中的包的个数，可以在 select之后通过**hi_mpi_venc_query_status**接口获得。
  - 码流包个数pack_cnt

    在输入时，此值指定pack中hi_venc_pack结构体内存的个数。按包获取时，pack_cnt必须不小于 1；按帧获取时，pack_cnt必须不小于当前帧的包个数。在函数调用成功后，pack_cnt返回实际填充pack的包的个数。
  - 序列号seq按帧获取时是帧序列号；按包获取时为包序列号。
  - 码流特征信息，数据类型为联合体，包含了不同编码协议对应的码流特征信息h264_info/jpeg_info/h265_info，码流特征信息的输出用于支持用户的上层应用。
  - 码流高级特征信息，数据类型为联合体，包含了不同编码协议对应的码流高级特征信息h264_adv_info/h265_adv_info，码流高级特征信息的输出用于支持用户的上层应用。
- 如果用户长时间不获取码流，码流缓冲区就会满。一个编码通道如果发生码流缓冲区满，就会不再启动编码，直到用户获取码流并释放码流，从而有足够的码流缓冲可以用于编码时，才开始继续编码。
- 建议用户获取码流接口调用与释放码流的接口调用成对出现，且尽快释放码流，防止出现由于用户获取码流，释放不及时而导致的码流buffer满，停止编码。
- 建议用户使用select方式获取码流，且按照如下的流程：
  - 调用**hi_mpi_venc_query_status** 函数查询编码通道状态；
  - 确保cur_packs和left_stream_frames同时大于0；
  - 调用malloc分配cur_packs个包信息结构体；
  - 调用**hi_mpi_venc_get_stream**获取编码码流；
  - 调用**hi_mpi_venc_release_stream**释放码流缓存。
- 当one_stream_buf为1时，用户获取一帧码流时只会得到一个码流包（不包含用户数据）的地址，即pack_cnt为1，地址为pack[0].addr。而在hi_venc_pack结构体中增加offset，用来指出一帧码流中有效数据的地址和pack[0].addr的偏移。如**图11-20**所示（以 H.264 单包为例），pack[0].data_num = 3,即3种NAL包，分别为SPS、PPS、SEI：
  - pack[0].pack_info[0]. pack_type. h264_type = HI_VENC_H264_NALU_SPS
  - pack[0].pack_info[1]. pack_type. h264_type = HI_VENC_H264_NALU_PPS
  - pack[0].pack_info[2]. pack_type. h264_type = HI_VENC_H264_NALU_SEI

  其他协议以此类推，data_num和 pack_info只在单包模式才有意义。

- 两种模式可通过调用接口hi_mpi_venc_set_mod_param设置h.265e/h264e/jpege模块参数one_stream_buf来选择。
  - one_stream_buf=1 表示单包模式；注意，JPEGE目前只支持单帧单包模式。
  - one_stream_buf=0 表示非单包模式，系统默认one_stream_buf=0。

函数原型

hi_s32 hi_mpi_venc_get_stream(hi_venc_chn chn, hi_venc_stream *stream, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
stream	输出	输出码流数据的指针。 该结构体内的addr参数配置的地址为Device上的内存地址，表示输出的码流存放在Device的内存中。
milli_sec	输入	超时时间，单位是毫秒。 ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

- 0：成功，获取编码结果数据成功后，需调用hi_mpi_venc_release_stream接口释放码流缓存资源。

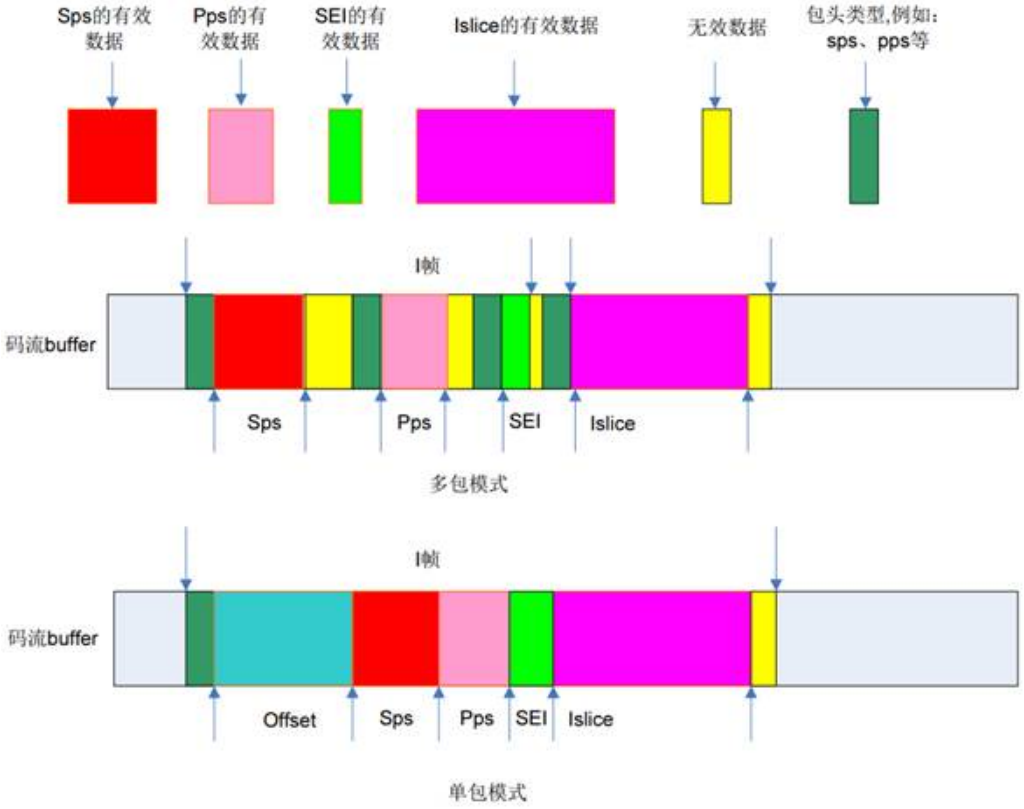
  本接口虽返回成功，但有可能获取到的是编码失败的数据，编码是否成功，需要查看hi_venc_stream结构体的pack成员下的len参数值，len=0表示编码失败。
- 非0：失败，参见11.18.21.12 VENC视频编码/JPEGE图片编码返回码

参考资源

接口调用流程及示例，参见5.5.3 JPEGE图片编码、5.5.6 VENC视频编码。

其它参考信息

图 11-20 码流包结构图



11.18.18.11 hi_mpi_venc_release_stream

函数功能

释放码流缓存。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 如果通道未创建，则返回错误码HI_ERR_VENC_UNEXIST。
- 此接口应当和 **hi_mpi_venc_get_stream** 配对起来使用，用户获取码流后必须及时释放已经获取的码流缓存，否则可能会导致码流buffer满，影响编码器编码，并且用户必须按先获取先释放的顺序释放已经获取的码流缓存。
- 在编码通道复位以后，所有未释放的码流包均无效，不能再使用或者释放这部分无效的码流缓存。
- 释放无效的码流会返回失败HI_ERR_VENC_ILLEGAL_PARAM。

函数原型

hi_s32 hi_mpi_venc_release_stream(hi_venc_chn chn, hi_venc_stream *stream)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
stream	输入	编码后的输出码流数据的指针。 如果stream为空，则返回错误码HI_ERR_VENC_NULL_PTR。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

参考资源

接口调用流程及示例，参见**5.5.3 JPEGE图片编码**、**5.5.6 VENC视频编码**。

11.18.18.12 hi_mpi_venc_send_frame

函数功能

发送原始图像进行视频或图像编码。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 视频输入的原始图像大小必须与编码通道的大小保持一致；JPEGE输入的原始图像大小小于或等于编码通道的大小。
- 调用该接口发送图像，用户需要保证编码通道已创建且开启接收输入图片。
- 编码时，输入内存需要在调用**hi_mpi_venc_get_stream**接口获取结果之后才能进行释放。
- 视频编码，对输入、输出的约束请参见**11.18.18.3 VENC功能及约束说明**。
- 图像编码，对输入、输出的约束请参见**11.18.18.1 JPEGE功能及约束说明**。

函数原型

hi_s32 hi_mpi_venc_send_frame(hi_venc_chn chn, const hi_video_frame_info *frame, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
frame	输入	原始图像信息的指针。
milli_sec	输入	超时时间，单位是毫秒。 ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

参考资源

接口调用流程及示例，参见**5.5.3 JPEGE图片编码**、**5.5.6 VENC视频编码**。

11.18.18.13 hi_mpi_venc_set_mod_param

函数功能

设置编码相关的模块参数。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

此接口在通道创建前调用，如果通道已经创建，则返回错误码HI_ERR_VENC_NOT_PERM。建议用户在调用此接口之前先调用hi_mpi_venc_get_mod_param接口，获取当前通道的编码模块参数配置，然后再根据需求设置相应参数。

函数原型

hi_s32 hi_mpi_venc_set_mod_param(const hi_venc_mod_param *mod_param)

参数说明

参数名	输入/输出	说明
mod_param	输入	编码模块参数指针。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.12 VENC视频编码/JPEGE图片编码返回码

参考资源

接口调用流程及示例，参见5.5.3 JPEGE图片编码、5.5.6 VENC视频编码。

11.18.18.14 hi_mpi_venc_get_mod_param

函数功能

获取编码相关的模块参数。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

建议用户在调用此接口之前先设置hi_venc_mod_param结构体中的mod_type参数，接口会根据不同的模块类型去获取相应模块参数。

函数原型

hi_s32 hi_mpi_venc_get_mod_param(hi_venc_mod_param *mod_param)

参数说明

参数名	输入/输出	说明
mod_param	输出	编码模块参数指针。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.12 VENC视频编码/JPEGE图片编码返回码

参考资源

接口调用流程及示例，参见5.5.3 JPEGE图片编码、5.5.6 VENC视频编码。

11.18.18.15 hi_mpi_venc_request_idr

函数功能

请求IDR帧。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 如果通道未创建，则返回失败。
- 接受IDR（Instantaneous Decoder Refresh）帧请求后，当instant=0时，则在帧率控制的下一帧编出IDR帧，当instant=1时，则立即编出IDR帧，不受帧率控制约束。
- IDR帧请求，只支持H.264/H.265编码协议。
- 由于目前的使用场景目标帧率与源帧率的值一致，所以此接口不受帧率控制影响当instant设置为0或1时，都是每调用一次接口即编出一个IDR，调用频繁会影响码流帧率和码率的稳定，使用时需要注意。
- 当GOP模式为Smartp或B帧模式（暂不支持这两种模式）下，请求IDR帧会延时生效。

函数原型

hi_s32 hi_mpi_venc_request_idr(hi_venc_chn chn, hi_bool instant)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
instant	输入	是否使能立即编码IDR帧。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

11.18.18.16 hi_mpi_venc_get_fd

函数功能

获取编码通道对应的文件句柄。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

在进程退出时会自动关闭文件句柄，回收资源。

函数原型

hi_s32 hi_mpi_venc_get_fd(hi_venc_chn chn)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。

返回值说明

- 0或正数值：有效返回值
- 非正数值：无效返回值

参考资源

接口调用流程及示例，参见**5.5.3 JPEGE图片编码**、**5.5.6 VENC视频编码**。

11.18.18.17 hi_mpi_venc_close_fd

函数功能

关闭编码通道对应的设备文件句柄。

不支持该接口。

约束说明

用户可以不调用本接口，在应用程序执行完成，对应的进程退出后，系统会自动清理句柄；如果用户主动调用本接口，必须确保在调用**hi_mpi_venc_destroy_chn**接口之后再调用。此接口不能与其它 MPI 接口同时调用，用户必须保证此接口与其它接口在时间上是串行调用的。

函数原型

hi_s32 hi_mpi_venc_close_fd(hi_venc_chn chn)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。

返回值说明

- 0：关闭成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

11.18.18.18 hi_mpi_venc_set_jpeg_param

函数功能

设置 JPEG 协议编码通道的量化表高级配置参数，用户需要调整JPEGE量化参数表时才需要使用该接口。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

本接口在编码通道创建之后、编码通道销毁之前调用。

用户可以先调用**hi_mpi_venc_get_jpeg_param**获取当前参数，然后对该参数进行修改后再调用本接口设置。当hi_venc_jpeg_param结构体内的qfactor参数值配置为0xFFFFFFFF，QT（Quantisation Table）表的Cb分量（Cb表示蓝色分量）和Cr分量（Cr表示红色分量）需保持一致。

函数原型

hi_s32 hi_mpi_venc_set_jpeg_param(hi_venc_chn chn, const hi_venc_jpeg_param *jpeg_param)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
jpeg_param	输入	JPEG协议编码通道的高级参数集合的指针。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.12 VENC视频编码/JPEGE图片编码返回码

11.18.18.19 hi_mpi_venc_get_jpeg_param

函数功能

获取JPEG协议编码通道的量化表高级配置参数。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

本接口在编码通道创建之后、编码通道销毁之前调用。

调用hi_mpi_venc_set_jpeg_param接口设置参数前，可以调用该接口获取当前JPEGE的量化表参数。

函数原型

hi_s32 hi_mpi_venc_get_jpeg_param(hi_venc_chn chn, hi_venc_jpeg_param *jpeg_param)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
jpeg_param	输出	JPEG协议编码通道的高级参数集合的指针。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.12 VENC视频编码/JPEGE图片编码返回码

11.18.18.20 hi_mpi_venc_set_chn_param

函数功能

设置通道参数。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 本接口在编码通道创建之后、编码通道销毁之前调用。如果针对未创建的通道，调用本接口设置参数，则返回失败。

函数原型

hi_s32 hi_mpi_venc_set_chn_param(hi_venc_chn chn, const hi_venc_chn_param *chn_param)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
chn_param	输入	VENC通道参数的指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

11.18.18.21 hi_mpi_venc_get_chn_param

函数功能

获取通道参数。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 本接口在编码通道创建之后、编码通道销毁之前调用。如果针对未创建的通道，调用本接口获取参数，则返回失败。

函数原型

hi_s32 hi_mpi_venc_get_chn_param(hi_venc_chn chn, hi_venc_chn_param *chn_param)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
chn_param	输出	VENC通道参数的指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

11.18.18.22 hi_mpi_venc_set_scene_mode

函数功能

设置场景模式。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 本接口在编码通道创建之后、编码通道销毁之前调用。
- 不同的编码模式场景对图像质量有影响，默认场景模式为HI_VENC_SCENE_0。

函数原型

hi_s32 hi_mpi_venc_set_scene_mode(hi_venc_chn chn, const hi_venc_scene_mode scene_mode)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
scene_mode	输入	场景模式。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.12 VENC视频编码/JPEGE图片编码返回码

11.18.18.23 hi_mpi_venc_get_scene_mode

函数功能

获取场景模式。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

本接口在编码通道创建之后、编码通道销毁之前调用。

函数原型

hi_s32 hi_mpi_venc_get_scene_mode(hi_venc_chn chn, hi_venc_scene_mode *scene_mode)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
scene_mode	输出	场景模式指针。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.12 VENC视频编码/JPEGE图片编码返回码

11.18.18.24 hi_mpi_venc_set_rc_param

函数功能

设置视频编码通道码率控制器的高级参数的值。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 本接口在编码通道创建之后、编码通道销毁之前调用。

- 如果不调用本接口设置编码通道码率控制器的高级参数的值，就会默认采用这些高级参数的默认值。
- 建议用户先调用hi_mpi_venc_get_rc_param接口获取编码通道码率控制器的高级参数的值，再调用本接口对高级参数进行设置。

函数原型

hi_s32 hi_mpi_venc_set_rc_param(hi_venc_chn chn, const hi_venc_rc_param *rc_param)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
rc_param	输入	编码通道码率控制器的高级参数的指针。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.12 VENC视频编码/JPEGE图片编码返回码

11.18.18.25 hi_mpi_venc_get_rc_param

函数功能

获取视频编码通道码率控制器的高级参数的值。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

本接口在编码通道创建之后、编码通道销毁之前调用。

函数原型

hi_s32 hi_mpi_venc_get_rc_param(hi_venc_chn chn, hi_venc_rc_param *rc_param)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
rc_param	输出	编码通道码率控制器的高级参数的指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

11.18.18.26 hi_mpi_venc_set_jpeg_huffman_param

函数功能

设置编码通道的JPEGE huffman编码高级参数。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

本接口在编码通道创建之后、编码通道销毁之前调用。

函数原型

hi_s32 hi_mpi_venc_set_jpeg_huffman_param(hi_venc_chn chn, const hi_venc_jpeg_huffman_param *jpeg_huffman_param)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
jpeg_huffman_param	输入	huffman编码参数的指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

11.18.18.27 hi_mpi_venc_get_jpeg_huffman_param

函数功能

获取编码通道的JPEGE huffman编码高级参数。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 本接口在编码通道创建之后、编码通道销毁之前调用。
- 不调用**hi_mpi_venc_set_jpeg_huffman_param**接口设置huffman编码高级参数时，返回默认的标准huffman编码参数。

函数原型

hi_s32 hi_mpi_venc_get_jpeg_huffman_param(hi_venc_chn chn, hi_venc_jpeg_huffman_param *jpeg_huffman_data)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
jpeg_huffman_data	输出	huffman高级参数的指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

11.18.18.28 hi_mpi_venc_compact_jpeg_tables

函数功能

配置是否对高级参数huffman或QT表进行压缩。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

本接口在编码通道创建之后、编码通道销毁之前调用。

调用此接口后，如果huffman表或QT表的Y、Cb、Cr分量内容一致时，支持压缩。

函数原型

hi_s32 hi_mpi_venc_compact_jpeg_tables(hi_venc_chn chn, hi_u32 table_type, hi_bool enable)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
table_type	输入	压缩类型，取值范围：仅支持0，表示huffman表数据和QT量化表数据都压缩。
enable	输入	是否压缩，0表示不压缩，1表示压缩。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

11.18.18.29 hi_mpi_venc_send_jpege_frame

函数功能

发送原始图像进行图像编码，并支持配置编码结果的输出内存地址。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 在调用**hi_mpi_venc_create_chn**接口创建通道时，将参数hi_venc_chn_attr.venc_attr.buf_size参数值设置为0，本接口才有效，否则返回错误码HI_ERR_VENC_NOT_SUPPORT。
- JPEGE输入图像分辨率必须小于或等于编码通道的宽、高。
- 调用该接口发送图像，用户需要保证编码通道已创建且开启接收输入图片。
- 编码时，输入、输出内存均需要在调用**hi_mpi_venc_get_stream**接口获取结果之后才能进行释放。

- 图像编码，对输入、输出的约束请参见11.18.18.1 JPEGE功能及约束说明。

函数原型

hi_s32 hi_mpi_venc_send_jpege_frame(hi_venc_chn chn, const hi_video_frame_info *frame, const hi_img_stream* jpege_stream, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
frame	输入	原始图像信息的指针。
jpege_stream	输入	用于指定编码结果的输出内存地址和长度。 输出内存首地址需满足16字节对齐，内存大小可以调用hi_mpi_venc_get_jpege_predicted_size获取，再可调用hi_mpi_dvpp_malloc接口申请输出内存。
milli_sec	输入	超时时间，单位是毫秒。 ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.12 VENC视频编码/JPEGE图片编码返回码

参考资源

接口调用流程及示例，参见5.5.3 JPEGE图片编码、5.5.6 VENC视频编码。

11.18.18.30 hi_mpi_venc_get_jpege_predicted_size

函数功能

预估JPEGE编码一帧图片所需的输出缓冲区大小。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

hi_s32 hi_mpi_venc_get_jpege_predicted_size(const hi_video_frame_info* frame, const hi_venc_jpeg_param* jpeg_param, hi_u32 *size)

参数说明

参数名	输入/输出	说明
frame	输入	原始图像信息的指针。
jpeg_param	输入	JPEG协议编码通道的高级参数集合的指针，预留参数。
size	输出	输出缓冲区大小的指针，单位为Byte。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

11.18.18.31 hi_mpi_venc_set_roi_attr

函数功能

设置H.264/H.265通道的ROI属性，用户可以通过配置ROI区域，对该区域的图像Qp进行限制，从而实现图像中该区域的Qp与其他图像区域的差异化。关于ROI编码功能的详细说明请参见**ROI编码功能说明**。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 本接口用于设置H.264/H.265通道的ROI属性。
- 本接口在编码通道创建之后，编码通道销毁之前设置。此接口在编码过程中被调用时，等到下一个帧时生效。
- 建议用户在创建通道之后，启动编码之前调用此接口，减少在编码过程中调用的次数。
- 建议用户在调用此接口之前先调用**hi_mpi_venc_get_roi_attr**接口，获取当前通道的ROI配置，然后再进行设置。
- 设置该接口后，如果当前帧判断编码为pskip 帧，以pskip帧效果优先。

函数原型

hi_s32 hi_mpi_venc_set_roi_attr(hi_venc_chn chn, const hi_venc_roi_attr *roi_attr)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
roi_attr	输入	ROI 区域参数指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

ROI 编码功能说明

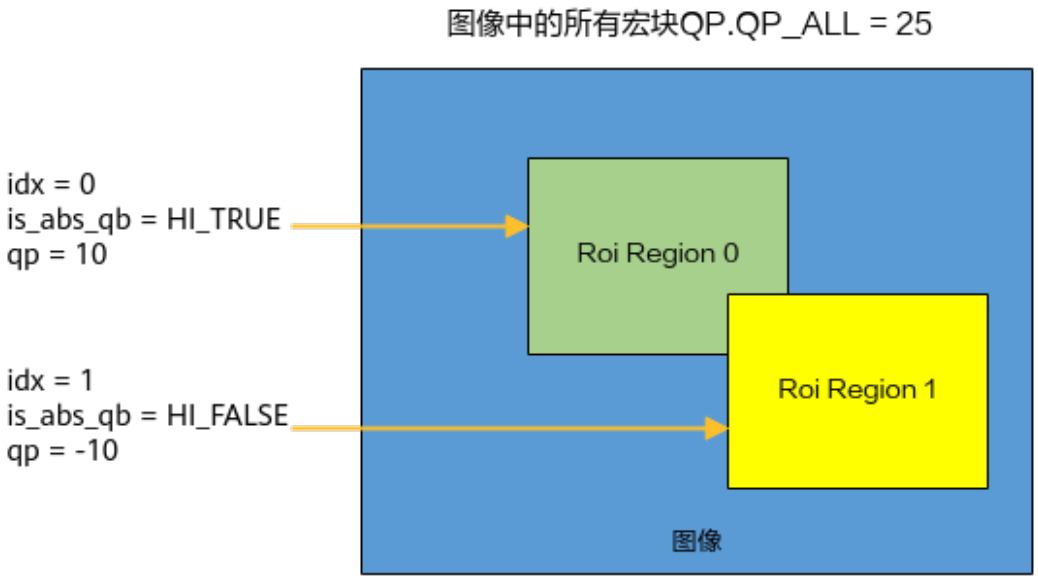
ROI（Region Of Interest）编码：对感兴趣区域编码。

用户可以通过配置ROI区域，对该区域的图像Qp进行限制，从而实现图像中该区域的Qp与其它图像区域的差异化。系统现仅支持对H.264/H.265通道进行ROI设置。系统提供了8个感兴趣区域，可供用户同时使用。

8个区域可以互相叠加，且叠加时的优先级按照0～7的索引号依次提高，这里，叠加优先级是指发生叠加时，图像区域的最终Qp值的判定，最终的区域Qp值按照优先级最高的区域设定。ROI区域可配置绝对Qp和相对Qp两种模式。

- 绝对Qp：ROI区域的Qp为用户设定的Qp 值。
- 相对Qp：ROI区域的Qp为码率控制产生的Qp与用户设定的Qp偏移值的和。

以下图为例，假如图像Qp为25，即图像中所有宏块Qp值为25。Roi区域0设置为绝对Qp模式，Qp值为10，索引为0；Roi区域1设置为相对Qp模式，Qp为-10，索引为1。区域0的index小于区域1的index，所以在发生互相重叠的图像区域按高优先级的区域（区域1）Qp设置。区域0除了发生重叠的部分的Qp值等于10。区域1 的Qp值为25-10=15。



11.18.18.32 hi_mpi_venc_get_roi_attr

函数功能

获取H.264/H.265通道的ROI配置属性。关于ROI编码功能的详细说明请参见**ROI编码功能说明**。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 本接口用于获取H.264/H.265协议编码通道的指定索引号的ROI配置。
- 本接口在编码通道创建之后，编码通道销毁之前调用。
- 建议用户在创建通道之后，启动编码之前调用此接口，减少在编码过程中调用的次数。

函数原型

hi_s32 hi_mpi_venc_get_roi_attr(hi_venc_chn chn, hi_u32 idx, hi_venc_roi_attr *roi_attr)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
idx	输入	H.264/H.265协议编码通道ROI区域索引。

参数名	输入/输出	说明
roi_attr	输出	ROI区域参数指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

11.18.18.33 hi_mpi_venc_set_ref_param

函数功能

设置H.264/H.265编码通道高级跳帧参考参数，用于改变帧间参考关系。关于高级跳帧模式的详细说明请参见**高级跳帧模式说明**。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 创建H.264/H.265协议编码通道时，默认跳帧参考模式是1倍跳帧参考模式。如果用户需要修改编码通道的跳帧参考，建议在创建通道之后，启动编码之前调用此接口，减少在编码过程中调用的次数。
- 本接口在编码过程中被调用时，等到下一个I帧时生效。
- 如果用户设置1倍跳帧参考模式，对应的配置为：pred_en=HI_TRUE，enhance=0 ，base =1；如果设置2倍跳帧参考模式，对应的配置为： pred_en= HI_TRUE，enhance=1， base = 1；如果设置 4 倍跳帧参考模式，对应的配置为： pred_en =HI_TRUE， enhance = 1， base = 2。
- 如果用户设置通道的Gop Mode为HI_VENC_GOP_MODE_DUAL_P时，sp_interval只支持设置为sp_interval = （ enhance+1)*base，其中，当sp_interval 不为0时， pred _en只支持设置为 HI_TRUE。
- 如果用户设置通道的Gop Mode为HI_VENC_GOP_MODE_SMART_P时，pred_en只支持设置为HI_TRUE 。
- 如果用户设置通道的Gop Mode为HI_VENC_GOP_MODE_BIPREDB时，enhance必须等于b_frame_num 。

函数原型

hi_s32 hi_mpi_venc_set_ref_param(hi_venc_chn chn, const hi_venc_ref_param *ref_param)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
ref_param	输入	H.264/H.265编码通道高级跳帧参考参数指针。

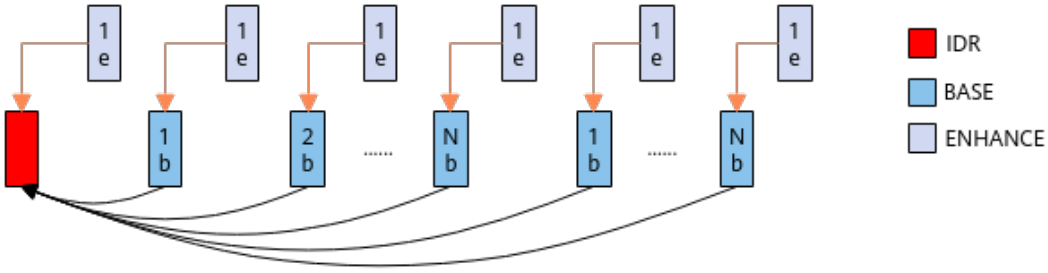
返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

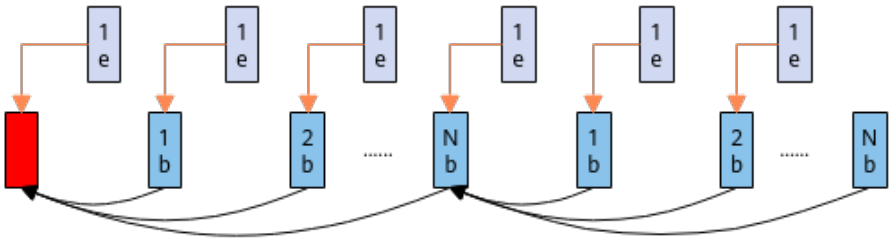
高级跳帧模式说明

高级跳帧参考模式涉及3 个参数：base、enhance 和pred_en，其含义请参考hi_venc_ref_param。高级跳帧参考模式示意图如下图所示。

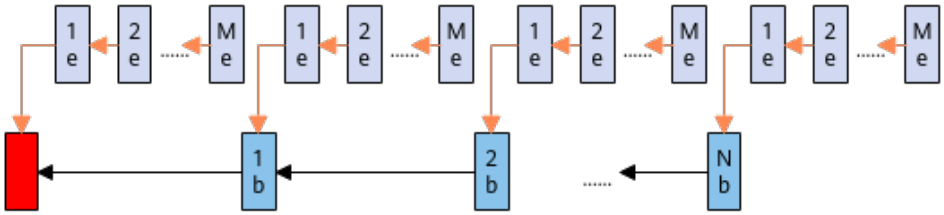
- 注意：dual_p 模式且sp_interval不等于0下配置高级跳帧参考需要满足如下条件：base *(enhance+1)= sp_interval，例如：4 倍高级跳帧参考（base=2，enhance=1），那么dual_p 的sp_interval =2*（1+1）=4。
- normal_p高级跳帧参考示意图



(base = N, enhance = 1, pred_en = HI_FALSE)

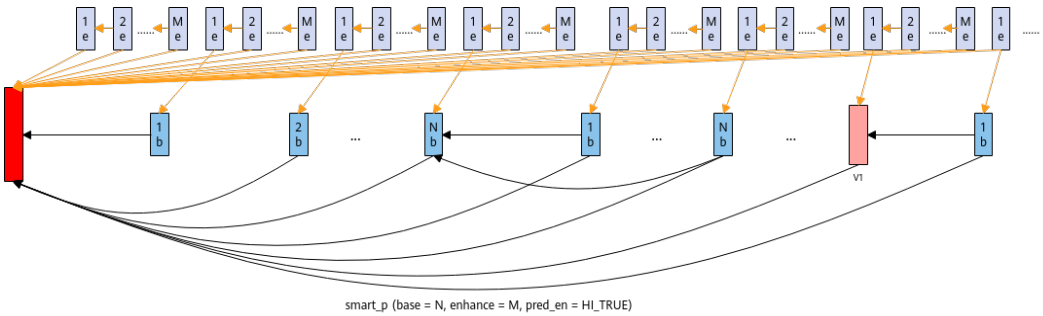


(base = N, enhance = 1, pred_en = HI_TRUE)

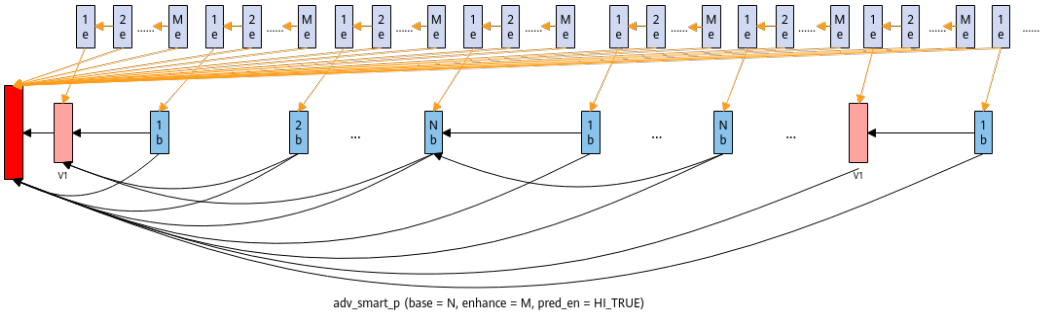


(base = 1, enhance = M, pred_en = HI_TRUE)

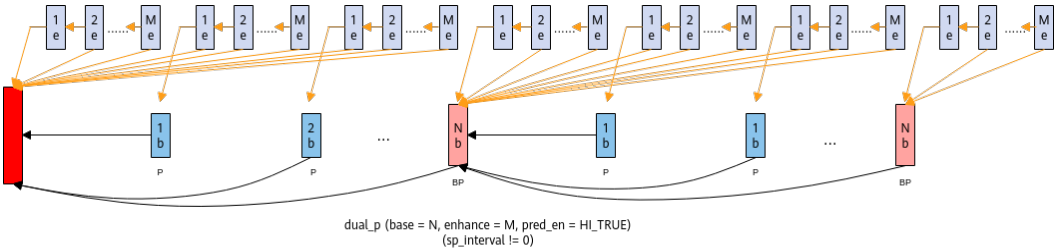
- smart_p高级跳帧参考示意图



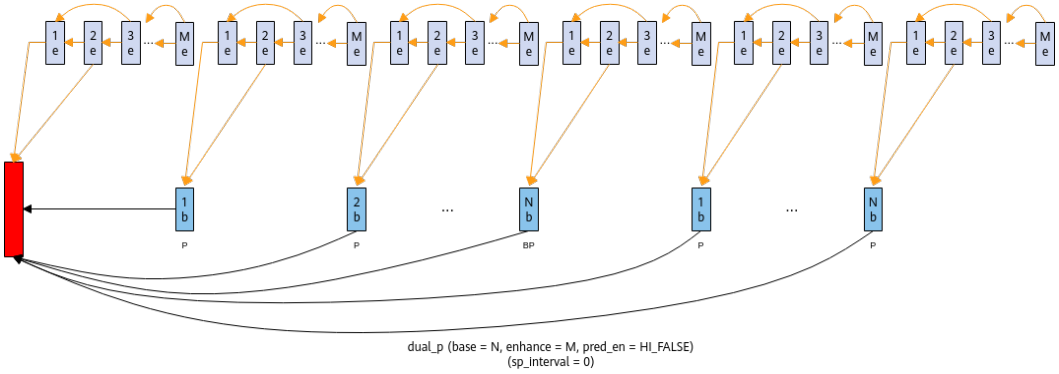
- adv_smart_p高级跳帧参考示意图



- dual_p (sp_interval!=0)高级跳帧参考示意图



- dual_p (sp_interval=0)高级跳帧参考示意图



11.18.18.34 hi_mpi_venc_get_ref_param

函数功能

获取H.264/H.265编码通道高级跳帧参考参数。关于高级跳帧模式的详细说明请参见**高级跳帧模式说明**。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

本接口在编码通道创建之后、编码通道销毁之前调用。

函数原型

hi_s32 hi_mpi_venc_get_ref_param(hi_venc_chn chn, hi_venc_ref_param *ref_param)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
ref_param	输出	H.264/H.265编码通道高级跳帧参考参数指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

11.18.18.35 hi_mpi_venc_set_h264_vui

函数功能

设置H.264协议编码通道的Vui参数。

Vui（Video usability information）参数主要包含解码图像的格式信息，比如采样横纵比、光电转换特性、颜色空间等。Vui参数在协议中属于可选参数，不影响视频编码过程，但会建议解码器/播放器做一些矫正处理。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 本接口用于设置H.264协议编码通道的Vui配置。
- 本接口在编码通道创建之后，编码通道销毁之前设置。此接口在编码过程中被调用时，等到下一个I帧时生效。
- 建议用户在创建通道之后，启动编码之前调用此接口，减少在编码过程中调用的次数。
- 用户应当仅在H.264协议编码通道中调用本接口。
- 目前仅开放hi_venc_h264_vui中的hi_venc_vui_video_signal参数和hi_venc_vui_h264_time_info参数供用户配置，其它参数预留，均需配置为默认值，否则会导致编码失败。建议用户在调用此接口之前先调用**hi_mpi_venc_get_h264_vui**接口，获取当前编码通道的Vui默认配置，然后再进行设置，避免预留参数出错。
- hi_venc_vui_h264_time_info参数包含时钟计数相关参数，默认情况下为不生效状态，用户想要控制显示帧率时需要主动配置相关参数。注意，显示帧率只影响播放器的播放速度，若用户想设置编码过程中的码控帧率需配置**hi_mpi_venc_create_chn**接口中的输入/输出帧率参数。

  建议用户将码控帧率与显示帧率设置一致，否则播放时的码率可能与用户设置的编码码率不同，播放时的码率会随着所设置的显示帧率而变化。

如果用户需要调整播放速度，则建议在合理范围内调整显示帧率，一般正常视频播放速度为30fps，表示显示帧率为30fps，若将显示帧率调大，则播放速度加快，例如3倍速播放，则显示帧率设置为90fps；若将显示帧率调小，则播放速度减慢，例如0.5倍速播放，则显示帧率设置为15fps。

- 由于Vui只是协议可选参数而非强制参数，调用本接口设置参数（如显示帧率、像素值域范围等）的最终效果取决于解码器/播放器行为，或者用户自定义的解码/播放行为。

函数原型

hi_s32 hi_mpi_venc_set_h264_vui(hi_venc_chn chn, const hi_venc_h264_vui *h264_vui)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
h264_vui	输入	H.264协议编码通道的Vui参数。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

11.18.18.36 hi_mpi_venc_get_h264_vui

函数功能

获取H.264协议编码通道的Vui配置。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 本接口用于获取H.264协议编码通道的Vui配置。
- 本接口在编码通道创建之后，编码通道销毁之前调用。
- 建议用户在创建通道之后，启动编码之前调用此接口，减少在编码过程中调用的次数。
- 用户应当仅在H.264协议编码通道中调用本接口。

函数原型

hi_s32 hi_mpi_venc_get_h264_vui(hi_venc_chn chn, hi_venc_h264_vui *h264_vui)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
h264_vui	输出	H.264协议编码通道的Vui属性。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

11.18.18.37 hi_mpi_venc_set_h265_vui

函数功能

设置H.265协议编码通道的Vui参数。

Vui（Video usability information）参数主要包含解码图像的格式信息，比如采样横纵比、光电转换特性、颜色空间等。Vui参数在协议中属于可选参数，不影响视频编码过程，但会建议解码器/播放器做一些矫正处理。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 本接口用于设置H.265协议编码通道的Vui配置。
- 本接口在编码通道创建之后，编码通道销毁之前设置。此接口在编码过程中被调用时，等到下一个I帧时生效。
- 建议用户在创建通道之后，启动编码之前调用此接口，减少在编码过程中调用的次数。
- 用户应当仅在H.265协议编码通道中调用本接口。
- 目前仅开放hi_venc_h265_vui中的hi_venc_vui_video_signal参数和hi_venc_vui_h265_time_info参数供用户配置，其它参数预留，均需配置为默认值，否则会导致编码失败。建议用户在调用此接口之前先调用**hi_mpi_venc_get_h265_vui**接口，获取当前编码通道的Vui默认配置，然后再进行设置，避免预留参数出错。
- hi_venc_vui_h265_time_info参数包含时钟计数相关参数，默认情况下为不生效状态，用户想要控制显示帧率时需要主动配置相关参数。注意，此处的显示帧率只影响播放器的播放速度，用户想控制编码过程中的码控帧率需配置**hi_mpi_venc_create_chn**接口中的hi_venc_rc_attr参数。

  建议用户将码控帧率与显示帧率设置一致，否则播放时的码率可能与用户设置的编码码率不同，播放时的码率会随着所设置的显示帧率而变化。

  如果用户需要调整播放速度，则建议在合理范围内调整显示帧率，一般正常视频播放速度为30fps，表示显示帧率为30fps，若将显示帧率调大，则播放速度加

快，例如3倍速播放，则显示帧率设置为90fps；若将显示帧率调小，则播放速度减慢，例如0.5倍速播放，则显示帧率设置为15fps。

- 由于Vui只是协议可选参数而非强制参数，本接口设置参数（如显示帧率、像素值域范围等）的最终效果取决于解码器/播放器行为，或者用户自定义的解码/播放行为。

函数原型

hi_s32 hi_mpi_venc_set_h265_vui(hi_venc_chn chn, const hi_venc_h265_vui *h265_vui)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
h265_vui	输入	H.265协议编码通道的Vui参数。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

11.18.18.38 hi_mpi_venc_get_h265_vui

函数功能

获取H.265协议编码通道的Vui配置。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 本接口用于获取H.265协议编码通道的Vui配置。
- 本接口在编码通道创建之后，编码通道销毁之前调用。
- 建议用户在创建通道之后，启动编码之前调用此接口，减少在编码过程中调用的次数。
- 用户应当仅在H.265协议编码通道中调用本接口。

函数原型

hi_s32 hi_mpi_venc_get_h265_vui(hi_venc_chn chn, hi_venc_h265_vui *h265_vui)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
h265_vui	输出	H.265协议编码通道的Vui属性。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

11.18.18.39 hi_mpi_venc_set_cu_pred

函数功能

本接口用于调节通道CU/MB的模式选择的倾向性，支持调节帧间预测（Inter）和帧内预测（Intra）的倾向性配置以及编码块大小的倾向性配置。

编码协议以像素块作为基本处理单元，H.265/H.264中分别称为CU(Coding Unit)/MB(MacroBlock)。本接口中的CU模式指Inter/Intra预测模式中对不同尺寸像素块的选择。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 仅H.265/H.264编码通道支持。
- 建议用户在创建通道之后，启动编码之前调用本接口，减少编码过程中调用本接口的次数。若在编码过程中调用本接口，则下一帧编码时配置生效。
- 建议用户在调用此接口之前先调用**hi_mpi_venc_get_cu_pred**接口，获取CU模式选择的倾向性配置，然后再进行设置。
- 注意某些场景下特定模式不使能（例如，H.264中最大MB尺寸为16*16，且baseline和main profile下inter16_cost不使能，high profile下inter8_cost不使能），即使用户设置了这些特定模式的倾向性参数也不生效。

函数原型

hi_s32 hi_mpi_venc_set_cu_pred(hi_venc_chn chn, const hi_venc_cu_prediction *cu_pred)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
cu_pred	输入	CU模式选择的倾向性参数。 该结构体内包含以下参数： ● pred_mode：倾向性选择模式，支持auto和manual两种模式。auto模式下由驱动内部完成倾向性配置，即用户的配置参数不生效；manual模式允许用户根据实际场景完成自定义的倾向性配置。 ● intra32_cost/intra16_cost/intra8_cost/intra4_cost/inter64_cost/inter32_cost/inter16_cost/inter8_cost分别设置各自模式的倾向性大小。默认每个cost值为8，即不做倾向性配置。每个模式的cost设置越大意味着选择该模式的倾向性越小。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

11.18.18.40 hi_mpi_venc_get_cu_pred

函数功能

获取通道CU/MB的模式选择的倾向性设置。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 本接口用于获取CU模式选择的倾向性配置。
- 本接口在编码通道创建之后，编码通道销毁之前调用。
- 仅H.265/H.264编码通道支持。

函数原型

hi_s32 hi_mpi_venc_get_cu_pred(hi_venc_chn chn, hi_venc_cu_prediction *cu_pred)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
cu_pred	输出	CU模式选择的倾向性参数。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

11.18.18.41 hi_mpi_venc_set_intra_refresh

函数功能

设置P帧刷新Islice的参数。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 仅支持H.265/H.264、GOP模式为HI_VENC_GOP_MODE_NORMAL_P的编码通道。
- 调用**hi_mpi_venc_set_ref_param**接口设置高级跳帧参考后需要重新进行设置，不支持高级跳帧参考pred_en为0。
- 建议在创建通道后、启动编码前先调用**hi_mpi_venc_get_intra_refresh**接口获取P帧刷Islice的设置参数后，再调用本接口设置P帧刷Islice的参数。
- 调用本接口使能P帧刷Islice时，会生成一个I帧，且需等下一个GOP的IDR帧才能生效；关闭P帧刷Islice参数时会立即生效。

函数原型

```
hi_s32 hi_mpi_venc_set_intra_refresh(hi_venc_chn chn, const
hi_venc_intra_refresh *intra_refresh);
```

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
intra_refresh	输入	刷Islice的参数。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.12 VENC视频编码/JPEGE图片编码返回码

11.18.18.42 hi_mpi_venc_get_intra_refresh

函数功能

获取P帧刷新Islice的设置参数。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 本接口必须在编码通道创建之后，编码通道销毁之前调用。
- 切换GOP模式时，获取的属性是已设置的最新的属性。

函数原型

hi_s32 hi_mpi_venc_get_intra_refresh(hi_venc_chn chn, hi_venc_intra_refresh *intra_refresh)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
intra_refresh	输出	刷Islice的参数。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

11.18.18.43 hi_mpi_venc_set_chn_attr

函数功能

本接口用于设置编码通道的属性。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 仅H.265/H.264编码通道支持。
- 建议用户在调用此接口之前先调用**hi_mpi_venc_get_chn_attr**接口，获取当前通道属性，然后再进行设置。
- 编码通道属性包括了编码器属性、帧结构GOP属性和码率控制器RC属性三部分。编码通道属性分为动态属性和静态属性两种。其中，静态属性的属性值在通道创建时配置，在通道创建之后不能被修改；动态属性的属性值在通道创建时配置，有部分参数不支持修改，支持修改的参数在通道销毁之前可以被修改。如果设置静态属性或者不支持修改的参数，则返回失败。静态属性、属性是否支持修改、属性取值范围具体请参见**hi_venc_chn_attr**。
- 设置编码通道属性，未修改gop_mode 时，则通道属性立即生效；修改gop_mode时，则通道属性延时生效，最大延时生效时间如下表所示（**当前不支持HI_VENC_GOP_MODE_LOW_DELAY_B模式**）：

gop_mode修改前	gop_mode修改后	最大延时（以编码帧的个数为单位）
HI_VENC_GOP_MODE_LOW_DELAY_B	非HI_VENC_GOP_MODE_LOW_DELAY_B	1+b_frame_num
非HI_VENC_GOP_MODE_LOW_DELAY_B	任意不同于改变前的gop_mode	1

- 修改gop_mode时，通道的P帧刷Islice、跳帧参考的属性（ H.264/H.265 ）、pic_order_cnt_type（ H.264 ）会恢复默认值，如需继续使用需要重新配置。
- 如果修改GOP、RC属性中的一种，RC模块会重新初始化，所有RC高级参数都恢复到默认值。如果客户程序中有对RC高级参数的调整，需要重新设置。 RC高级参数请参考**hi_mpi_venc_set_rc_param**等。
- 如果同时修改gop_mode和rc_mode时，由于存在延迟生效（延时参见上表），为了防止调用**hi_mpi_venc_get_rc_param**获取的RC参数值不准确，建议等待延迟参数生效后再执行相应操作。

函数原型

hi_s32 hi_mpi_venc_set_chn_attr(hi_venc_chn chn, const hi_venc_chn_attr *chn_attr)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
chn_attr	输入	编码通道属性的指针。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.12 VENC视频编码/JPEGE图片编码返回码

11.18.18.44 hi_mpi_venc_get_chn_attr

函数功能

获取编码通道当前的属性。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 本接口在编码通道创建之后、编码通道销毁之前调用。
- 仅H.265/H.264编码通道支持。

函数原型

hi_s32 hi_mpi_venc_get_chn_attr(hi_venc_chn chn, hi_venc_chn_attr *chn_attr)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
chn_attr	输出	编码通道属性的指针。

返回值说明

- 0：成功

- 非0：失败，参见11.18.21.12 VENC视频编码/JPEGE图片编码返回码

11.18.18.45 hi_mpi_venc_set_slice_split

函数功能

本接口用于设置通道的slice分割配置。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 仅H.265/H.264编码通道支持。
- 本接口在编码通道创建之后，编码通道销毁之前设置。若在编码过程中调用本接口，则下一帧编码时配置生效。
- 建议用户在调用此接口之前先调用hi_mpi_venc_get_slice_split接口，获取当前通道的slice_split配置，然后再进行设置。

函数原型

hi_s32 hi_mpi_venc_set_slice_split(hi_venc_chn chn, const hi_venc_slice_split *slice_split)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
slice_split	输入	H.264/H.265协议编码的码流分割参数。

返回值说明

- 0：成功
- 非0：失败，参见11.18.21.12 VENC视频编码/JPEGE图片编码返回码

11.18.18.46 hi_mpi_venc_get_slice_split

函数功能

本接口用于获取通道的slice分割配置。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 仅H.265/H.264编码通道支持。
- 本接口在编码通道创建之后、编码通道销毁之前调用。

函数原型

hi_s32 hi_mpi_venc_get_slice_split(hi_venc_chn chn, hi_venc_slice_split *slice_split)

参数说明

参数名	输入/输出	说明
chn	输入	编码通道号。 Atlas 推理系列产品，编码通道号的取值范围：[0, 256)，JPEGE功能和VENC功能共用通道，且通道总数最多256。
slice_split	输出	H.264/H.265协议编码的码流分割参数。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.12 VENC视频编码/JPEGE图片编码返回码**

11.18.19 PNGD 图片解码功能

11.18.19.1 功能及约束说明

Atlas 200/300/500 推理产品上，当前版本不支持该功能。

功能说明

PNGD（PNG Decoder）功能：实现PNG格式图片的解码。

PNGD在解码图片时，支持按源图片格式解码。源图片格式解码是指解码前后图片的编码格式保持一致，例如解码前输入图片格式为RGB，解码后输出图片格式为RGB888。

如果不清楚PNG源图片格式、但想使用源格式解码时，有以下两种方式：

- 在调用PNGD解码接口时，直接将输出图片格式配置为HI_PIXEL_FORMAT_UNKNOWN，输出格式默认按源图片格式输出。此种方式，由于不知道输出图片格式，因此需要用户申请尽量大的内存，防止内存不够，无法存放输出图片。

输入图片格式	操作	实际输出图片格式
RGB/GRAY	将输出图片格式设置为HI_PIXEL_FORMAT_UNKNOWN	RGB888
RGBA/AGRAY	将输出图片格式设置为HI_PIXEL_FORMAT_UNKNOWN	RGBA8888

- 先调用**hi_mpi_png_get_image_info**接口根据传入的PNG源图片，获取按源图解码时的输出图片的宽、高、宽stride、高stride、解码输出内存大小、图片格式等信息后，再调用PNGD解码接口，使用通过hi_mpi_png_get_image_info接口获取的图片格式来设置输出图片格式。

图片分辨率约束

- 输入图片分辨率：

  最大分辨率4096*4096，最小分辨率32*32。
- 输出图片分辨率

  PNGD只对图片解码，不会改变图片分辨率，因此输出与输入的图片分辨率保持一致。

内存申请/释放接口说明

输入内存的大小就是指实际的输入图片所占用的大小，输出内存的大小参见**表11-56**中的计算公式。

版本	内存申请/释放接口
Atlas 推理系列产品	PNG图片解码时，调用**hi_mpi_dvpp_malloc****hi_mpi_dvpp_free**接口申请\释放Device上的输入、输出内存，这部分内存的生命周期由用户自行管理。

图片格式、宽高对齐、内存约束

若不确定输出格式、将输出图片格式设置为HI_PIXEL_FORMAT_UNKNOWN时：

- 宽stride：

  如果输入图片是RGB或GRAY格式，则宽stride为输出图片的宽先向上128对齐后再乘以3的值；如果输入图片是RGBA或AGRAY格式，则宽stride为输出图片的宽先向上128对齐后再乘以4的值。
- 高stride：为输出图片的高向上16对齐后的值。

表 11-56 图片格式、宽高对齐、内存大小约束

输入图片格式	输出图片格式	输出图片宽、高对齐要求	输出图片宽stride、高stride、内存大小要求
RGB	RGB888	无对齐要求	宽stride为宽向上1对齐、或16对齐、或128对齐后再乘以3的值。 高stride的取值范围：[输出图片的高, 输出图片的高向上128对齐]。 内存大小（单位Byte）= 宽stride * 高stride
GRAY	RGB888		
RGBA	RGB888		宽stride为宽向上1对齐、或16对齐、或128对齐后再乘以3的值。 高stride的取值范围：[输出图片的高, 输出图片的高向上128对齐]。 内存大小（单位Byte）= 宽stride * 高stride
	RGBA8888 8bit		宽stride为宽向上1对齐、或16对齐、或128对齐后再乘以4的值。 高stride的取值范围：[输出图片的高, 输出图片的高向上128对齐]。 内存大小（单位Byte）= 宽stride * 高stride
AGRAY	RGB888		宽stride为宽向上1对齐、或16对齐、或128对齐后再乘以3的值。 高stride的取值范围：[输出图片的高, 输出图片的高向上128对齐]。 内存大小（单位Byte）= 宽stride * 高stride
	RGBA8888 8bit		宽stride为宽向上1对齐、或16对齐、或128对齐后再乘以4的值。 高stride的取值范围：[输出图片的高, 输出图片的高向上128对齐]。 内存大小（单位Byte）= 宽stride * 高stride

说明

- 输出图片格式的定义请参见**hi_pixel_format**，宽stride、高stride等概念请参见**11.18.2 基本概念**。
- 如果用户将宽stride、高stride设置为0，传入对应的接口，PNGD内部在处理时，会默认根据输出图片格式将宽stride向上128对齐、高stride向上16对齐。

其它约束

PNGD只支持对完整PNG图片进行解码，不支持将一张PNG图片分割成多个数据包后，由PNGD解码。

11.18.19.2 性能指标说明

Atlas 200/300/500 推理产品上，当前版本不支持该功能。

性能指标说明

本节以单个昇腾AI处理器为例，描述基本场景性能指标参考，如下表所示，其它分辨率可以等量估算。

下表中的1路对应一个通道，一个通道对应一个线程；各分辨率的含义为：1080p指分辨率为1920*1080的图片，4K指分辨率为3840*2160的图片。

场景举例	总帧率
1080p*n路（1≤n≤5）	n*4fps
1080p*n路（n≥ 6）	24fps
4k*n路（1≤n≤5）	n*1fps
4k*n路（n≥ 6）	6fps

11.18.19.3 hi_mpi_pngd_create_chn

函数功能

根据设置的通道属性创建解码通道

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 单个Device上的通道号不能超出最大的通道号范围。
- 如果参数attr为空，会返回错误码HI_ERR_PNGD_NULL_PTR。
- 在创建PNG解码通道之前必须保证通道未创建（或者已经销毁），否则会直接返回错误。
- 系统内存不足时会返回HI_ERR_PNGD_NO_MEM的错误码，可考虑扩展OS内存。
- 当通道属性attr中的值超过解码能力集时，会返回HI_ERR_PNGD_ILLEGAL_PARAM的错误码。

函数原型

hi_s32 hi_mpi_pngd_create_chn(hi_pngd_chn chn, const hi_pngd_chn_attr *attr)

参数说明

参数名	输入/输出	说明
chn	输入	解码通道号。 该参数的取值范围：[0, 128)，通道总数最多128。 **说明** **对于Atlas 推理系列产品**，昇腾虚拟化实例（Ascend Virtual Instance）场景下，使用不同的昇腾虚拟化实例（Ascend Virtual Instance）模板，对通道数的限制不同： ● 使用vir04_4c_dvpp模板时，通道总数固定为128。 ● 使用vir04_3c_ndvpp模板时，表示不使用DVPP功能，因此通道总数为0。 ● 使用其它模板时，通道总数 =切分规格 * 128，如果通道总数不为整数，则向下取整。 昇腾虚拟化实例（Ascend Virtual Instance）的详细描述，以及各场景下的算力资源规格请参见《昇腾虚拟化实例（AVI）用户指南》。
attr	输入	解码通道属性指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.13 PNGD图像解码返回码**

参考资源

接口调用流程及示例，参见**5.5.4 PNGD图片解码**。

11.18.19.4 hi_mpi_pngd_destroy_chn

函数功能

用户退出流程时，需要关闭之前创建的解码通道，以释放相关资源。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

销毁前必须保证通道已创建，否则会返回通道未创建的错误。

函数原型

hi_s32 hi_mpi_pngd_destroy_chn(hi_pngd_chn chn)

参数说明

参数名	输入/输出	说明
chn	输入	解码通道号。 该参数的取值范围：[0, 128)，通道总数最多128。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.13 PNGD图像解码返回码**

参考资源

接口调用流程及示例，参见**5.5.4 PNGD图片解码**。

11.18.19.5 hi_mpi_pngd_send_stream

函数功能

解码前，向解码通道发送码流数据及存放解码结果的buffer。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 发送数据前必须保证通道已经被创建，否则直接返回通道未创建的错误码HI_ERR_PNGD_UNEXIST。如果在发送码流过程中销毁通道，就会立刻返回错误码HI_ERR_PNGD_UNEXIST。
- 发送码流时需要送入整张png图片。否则，解码会出现错误。
- 以非阻塞方式发送码流，如果码流缓冲区已满，会立刻返回错误码HI_ERR_PNGD_BUF_FULL。
- 以超时方式发送码流，到达设定的超时时间还不能成功发送码流会返回错误码HI_ERR_PNGD_BUF_FULL。
- 图像解码时，对输入、输出图片的要求请参见**11.18.19.1 功能及约束说明**。
- 解码时，输入输出内存必须使用**hi_mpi_dvpp_malloc**接口和**hi_mpi_dvpp_free**接口进行申请和释放，输入输出内存均需要在调用**hi_mpi_pngd_get_image_data**接口获取结果之后才能进行释放。

函数原型

hi_s32 hi_mpi_pngd_send_stream(hi_pngd_chn chn, const hi_img_stream *stream, hi_pic_info *png_pic_info, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	解码通道号。 该参数的取值范围：[0, 128)，通道总数最多128。
stream	输入	输入码流信息的指针。 该结构体内的addr参数配置的地址为Device上的内存地址。
png_pic_info	输入	输出图片信息的指针。
milli_sec	输入	超时时间，单位是毫秒。 ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.13 PNGD图像解码返回码**

参考资源

接口调用流程及示例，参见**5.5.4 PNGD图片解码**。

11.18.19.6 hi_mpi_pngd_get_image_data

函数功能

解码后，获取解码通道的解码图像及输入Stream。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

- 此接口通过改变milli_sec值支持阻塞方式、非阻塞方式、超时方式获取解码图像。
- 获取解码图像时必须保证通道已经被创建，否则直接返回通道未创建的错误码HI_ERR_PNGD_UNEXIST。如果在获取图像的过程中销毁通道，就会立刻返回错误码 HI_ERR_PNGD_UNEXIST。

- 以非阻塞方式获取解码图像，如果缓冲区内无图像，会立刻返回错误码HI_ERR_PNGD_BUF_EMPTY。
- 以超时方式获取解码图像，到达设定的超时时间还不能获取到图像则会返回错误码 HI_ERR_PNGD_BUF_EMPTY。
- 向PNGD获取解码结果，输入buffer和输出buffer一起获取。
- PNGD图片解码时，关于输入、输出图片的要求请参见**11.18.19.1 功能及约束说明**。

函数原型

hi_s32 hi_mpi_pngd_get_image_data(hi_pngd_chn chn, hi_pic_info *png_pic_info, hi_img_stream *stream, hi_s32 milli_sec)

参数说明

参数名	输入/输出	说明
chn	输入	解码通道号。 该参数的取值范围：[0, 128)，通道总数最多128。
png_pic_info	输出	已解码的图像信息的指针。解码后的数据存放在Device内存中。
stream	输出	已解码的输入码流信息的指针。
milli_sec	输入	超时时间，单位是毫秒。 ● -1：阻塞方式 ● 0：非阻塞方式 ● >0：超时方式，配置具体的超时时间。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的milli_sec参数值为1，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

- 0：获取数据成功
- 非0：失败，参见**11.18.21.13 PNGD图像解码返回码**。HI_ERR_PNGD_BUF_EMPTY表示无数据，其他错误码表示解码异常

参考资源

接口调用流程及示例，参见**5.5.4 PNGD图片解码**。

11.18.19.7 hi_mpi_png_get_image_info

函数功能

根据输入码流，获取png源图片的宽、高、源格式及源格式解码时输出图片的宽stride、高stride、以及所需的内存大小等信息。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

提供给用户来计算解码后输出图片占用的内存大小。当前仅支持解析png图片。通过本接口获取的widthStride向上128对齐、heightStride向上16对齐。

函数原型

hi_s32 hi_mpi_png_get_image_info(const hi_img_stream *png_stream, hi_img_info *img_info)

参数说明

参数名	输入/输出	说明
png_stream	输入	输入码流信息的指针。 **Control CPU开放形态**下，在Device上调用本接口时，该结构体内的addr参数配置的地址必须是Device上的内存地址。
img_info	输出	图片信息的指针。

返回值说明

- 0：成功
- 非0：失败，参见**11.18.21.13 PNGD图像解码返回码**

11.18.20 枚举值及数据结构

11.18.20.1 公共

此部分作为公共部分，描述了本模块所支持以及未来可能支持的图片格式，以及支持的Payload（数据内容）类型等等。

11.18.20.1.1 基本数据类型说明

```
typedef unsigned char hi_u8;
typedef signed char hi_s8;
typedef unsigned short hi_u16;
typedef short hi_s16;
typedef unsigned int hi_u32;
typedef int hi_s32;
typedef unsigned long long hi_u64;
```

```
typedef long long hi_s64;
typedef char hi_char;
typedef double hi_double;
typedef hi_u32 hi_fr32;
typedef float hi_float;

typedef enum {
    HI_FALSE = 0,
    HI_TRUE = 1,
} hi_bool;

#define hi_void void
#define HI_NULL 0L
#define HI_SUCCESS 0
#define HI_FAILURE (-1)
```

11.18.20.1.2 hi_pixel_format

说明

DVPP所支持的各种图片格式的枚举。

定义

```
typedef enum {
    HI_PIXEL_FORMAT_YUV_400 = 0,              // YUV400 8bit
    HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1,   // YUV420SP NV12 8bit
    HI_PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2,   // YUV420SP NV21 8bit
    HI_PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3,   // YUV422SP 8bit
    HI_PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4,   // YVU422SP 8bit
    HI_PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5,   // YUV444SP 8bit
    HI_PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6,   // YVU444SP 8bit
    HI_PIXEL_FORMAT_YUYV_PACKED_422 = 7,      // YUV422 Package YUYV 8bit
    HI_PIXEL_FORMAT_UYVY_PACKED_422 = 8,      // YUV422 Package  UYVY 8bit
    HI_PIXEL_FORMAT_YVYU_PACKED_422 = 9,      // YUV422 Package  YVYU 8bit
    HI_PIXEL_FORMAT_VYUY_PACKED_422 = 10,     // YUV422 Package  VYUY 8bit
    HI_PIXEL_FORMAT_YUV_PACKED_444 = 11,      // YUV444 Package  8bit
    HI_PIXEL_FORMAT_RGB_888 = 12,             // RGB888
    HI_PIXEL_FORMAT_BGR_888 = 13,             // BGR888
    HI_PIXEL_FORMAT_ARGB_8888 = 14,           // ARGB8888
    HI_PIXEL_FORMAT_ABGR_8888 = 15,           // ABGR8888
    HI_PIXEL_FORMAT_RGBA_8888 = 16,           // RGBA8888
    HI_PIXEL_FORMAT_BGRA_8888 = 17,           // BGRA8888
    HI_PIXEL_FORMAT_YUV_SEMI_PLANNER_420_10BIT = 18,    // YUV420SP 10bit
    HI_PIXEL_FORMAT_YVU_SEMI_PLANNER_420_10BIT = 19,    // YVU420sp 10bit
    HI_PIXEL_FORMAT_YVU_PLANAR_420 = 20,       // YVU420P 8bit
    HI_PIXEL_FORMAT_YVU_PLANAR_422 = 21,       // YVU422P 8bit
    HI_PIXEL_FORMAT_YVU_PLANAR_444 = 22,       // YVU444P 8bit
    HI_PIXEL_FORMAT_RGB_444 = 23,             // RGB444  R:4bit G:4bit B:4bit，当前不支持该格式
    HI_PIXEL_FORMAT_BGR_444 = 24,             // BGR444  R:4bit G:4bit B:4bit，当前不支持该格式
    HI_PIXEL_FORMAT_ARGB_4444 = 25,           // ARGB4444 A:4bit R:4bit G:4bit B:4bit
    HI_PIXEL_FORMAT_ABGR_4444 = 26,           // ABGR4444 A:4bit B:4bit G:4bit R:4bit，当前不支持该格式
    HI_PIXEL_FORMAT_RGBA_4444 = 27,           // RGBA4444 R:4bit G:4bit B:4bit A:4bit，当前不支持该格式
    HI_PIXEL_FORMAT_BGRA_4444 = 28,           // BGRA4444 B:4bit G:4bit R:4bit A:4bit，当前不支持该格式
    HI_PIXEL_FORMAT_RGB_555 = 29,             // RGB555 R:5bit G:5bit B:5bit，当前不支持该格式
    HI_PIXEL_FORMAT_BGR_555 = 30,             // BGR555 B:5bit G:5bit R:5bit，当前不支持该格式
    HI_PIXEL_FORMAT_RGB_565 = 31,             // RGB565 R:5bit G:6bit B:5bit，当前不支持该格式
    HI_PIXEL_FORMAT_BGR_565 = 32,             // BGR565 B:5bit G:6bit R:5bit，当前不支持该格式
    HI_PIXEL_FORMAT_ARGB_1555 = 33,           // ARGB1555 A:1bit R:5bit G:6bit B:5bit
    HI_PIXEL_FORMAT_ABGR_1555 = 34,           // ABGR1555 A:1bit B:5bit G:6bit R:5bit，当前不支持该格式
    HI_PIXEL_FORMAT_RGBA_1555 = 35,           // RGBA1555 A:1bit B:5bit G:6bit R:5bit，当前不支持该格式
    HI_PIXEL_FORMAT_BGRA_1555 = 36,           // BGRA1555 A:1bit B:5bit G:6bit R:5bit，当前不支持该格式
    HI_PIXEL_FORMAT_ARGB_8565 = 37,           // ARGB8565 A:8bit R:5bit G:6bit B:5bit，当前不支持该格式
    HI_PIXEL_FORMAT_ABGR_8565 = 38,           // ABGR8565 A:8bit B:5bit G:6bit R:5bit，当前不支持该格式
    HI_PIXEL_FORMAT_RGBA_8565 = 39,           // RGBA8565 A:8bit R:5bit G:6bit B:5bit，当前不支持该格式
    HI_PIXEL_FORMAT_BGRA_8565 = 40,           // BGRA8565 A:8bit B:5bit G:6bit R:5bit，当前不支持该格式
    HI_PIXEL_FORMAT_ARGB_CLUT2 = 41,          // ARGB Color Lookup Table 2bit
```

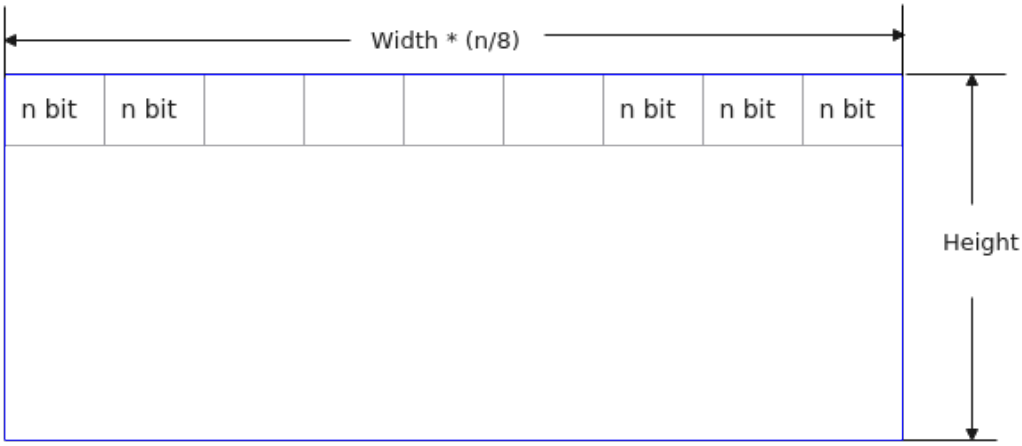
```
    HI_PIXEL_FORMAT_ARGB_CLUT4 = 42,          // ARGB Color Lookup Table 4bit

    HI_PIXEL_FORMAT_RGB_BAYER_8BPP = 50,
    HI_PIXEL_FORMAT_RGB_BAYER_10BPP = 51,
    HI_PIXEL_FORMAT_RGB_BAYER_12BPP = 52,
    HI_PIXEL_FORMAT_RGB_BAYER_14BPP = 53,
    HI_PIXEL_FORMAT_RGB_BAYER_16BPP = 54,      // RGB Bayer 16bit，Bayer图像，当前不支持该格式
    HI_PIXEL_FORMAT_YUV_PLANAR_420 = 55,       // YUV420P 8bit
    HI_PIXEL_FORMAT_YUV_PLANAR_422 = 56,       // YUV422P 8bit
    HI_PIXEL_FORMAT_YUV_PLANAR_444 = 57,       // YUV444P 8bit
    HI_PIXEL_FORMAT_YVU_PACKED_444 = 58,       // YVU444 Package 8bit
    HI_PIXEL_FORMAT_XYUV_PACKED_444 = 59,      // AYUV444 Package 8bit
    HI_PIXEL_FORMAT_XYVU_PACKED_444 = 60,      // AYVU444 Package 8bit
    HI_PIXEL_FORMAT_YUV_SEMIPLANAR_411 = 61,   // YUV411SP 8bit
    HI_PIXEL_FORMAT_YVU_SEMIPLANAR_411 = 62,   // YVU411SP 8bit
    HI_PIXEL_FORMAT_YUV_PLANAR_411 = 63,       // YUV411P 8bit
    HI_PIXEL_FORMAT_YVU_PLANAR_411 = 64,       // YVU411P 8bit
    HI_PIXEL_FORMAT_YUV_PLANAR_440 = 65,       // YUV440P 8bit
    HI_PIXEL_FORMAT_YVU_PLANAR_440 = 66,       // YVU440P 8bit

    HI_PIXEL_FORMAT_RGB_888_PLANAR = 69,       // RGB888 Planar
    HI_PIXEL_FORMAT_BGR_888_PLANAR = 70,       // BGR888 Planar
    HI_PIXEL_FORMAT_HSV_888_PACKAGE = 71,      // HSV Package，HSV图像package格式，当前不支持该格
式
    HI_PIXEL_FORMAT_HSV_888_PLANAR = 72,       // HSV Planar，HSV图像Planar格式，当前不支持该格式
    HI_PIXEL_FORMAT_LAB_888_PACKAGE = 73,      // LAB Package，LAB图像package格式，当前不支持该格
式
    HI_PIXEL_FORMAT_LAB_888_PLANAR = 74,       // LAB Planar，LAB图像Planar格式，当前不支持该格式
    HI_PIXEL_FORMAT_S8C1 = 75,                 // Signed 8bit for 1pixel 1Channel，每个像素用1个8bit有符号数
据表示的单通道图像，当前不支持该格式
    HI_PIXEL_FORMAT_S8C2_PACKAGE = 76,         // Signed 8bit for 1pixel 2Channel Package，每个像素用2个
8bit有符号数表示的双通道图像Package格式，当前不支持该格式
    HI_PIXEL_FORMAT_S8C2_PLANAR = 77,          // Signed 8bit for 1pixel 2Channel Planar，每个像素用2个
8bit有符号数据表的双通道图像Planar格式，当前不支持该格式
    HI_PIXEL_FORMAT_S16C1 = 78,                // Signed 16bit 1pixel 1Channel，每个像素用1个16bit有符号数
据表示的单通道图像，当前不支持该格式
    HI_PIXEL_FORMAT_U8C1 = 79,                 // Unsigned 8bit 1pixel 1Channel，每个像素用1个8bit无符号数
据表示的单通道图像，当前不支持该格式
    HI_PIXEL_FORMAT_U16C1 = 80,                // Unsigned 16bit 1pixel 1Channel，每个像素用1个16bit无符号
数据表示的单通道图像，当前不支持该格式
    HI_PIXEL_FORMAT_S32C1 = 81,                // Signed 32bit 1pixel 1Channel，每个像素用1个32bit有符号数
据表示的单通道图像，当前不支持该格式
    HI_PIXEL_FORMAT_U32C1 = 82,                // Unsigned 32bit 1pixel 1Channel，每个像素用1个32bit无符号
数据表示的单通道图像，当前不支持该格式
    HI_PIXEL_FORMAT_U64C1 = 83,                // Unsigned 64bit 1pixel 1Channel，每个像素用1个64bit无符号
数据表示的单通道图像，当前不支持该格式
    HI_PIXEL_FORMAT_S64C1 = 84,                // Signed 64bit 1pixel 1Channel，每个像素用1个64bit有符号数
据表示的单通道图像，当前不支持该格式


    HI_PIXEL_FORMAT_RGB_888_INT8 = 110,        // RGB888 Package 每个像素的单分量占用1个8bit有符号数
    HI_PIXEL_FORMAT_BGR_888_INT8 = 111,        // BGR888 Package 每个像素的单分量占用1个8bit有符号数
    HI_PIXEL_FORMAT_RGB_888_INT16 = 112,       // RGB888 Package 每个像素的单分量占用1个16bit有符号
数
    HI_PIXEL_FORMAT_BGR_888_INT16 = 113,       // BGR888 Package 每个像素的单分量占用1个16bit有符号
数
    HI_PIXEL_FORMAT_RGB_888_INT32 = 114,       // RGB888 Package 每个像素的单分量占用1个32bit有符号
数
    HI_PIXEL_FORMAT_BGR_888_INT32 = 115,       // BGR888 Package 每个像素的单分量占用1个32bit有符号
数
    HI_PIXEL_FORMAT_RGB_888_UINT16 = 116,      // RGB888 Package 每个像素的单分量占用1个16bit无符号
数
    HI_PIXEL_FORMAT_BGR_888_UINT16 = 117,      // BGR888 Package 每个像素的单分量占用1个16bit无符号
数
    HI_PIXEL_FORMAT_RGB_888_UINT32 = 118,      // RGB888 Package 每个像素的单分量占用1个32bit无符号
数
    HI_PIXEL_FORMAT_BGR_888_UINT32 = 119,      // BGR888 Package 每个像素的单分量占用1个32bit无符号
数
    HI_PIXEL_FORMAT_RGB_888_PLANAR_INT8  = 120,// RGB888 Planar 每个像素的单分量占用1个8bit有符号
```

```
数
    HI_PIXEL_FORMAT_BGR_888_PLANAR_INT8  = 121,// BGR888 Planar 每个像素的单分量占用1个8bit有符号
数
    HI_PIXEL_FORMAT_RGB_888_PLANAR_INT16 = 122,// RGB888 Planar 每个像素的单分量占用1个16bit有符
号数
    HI_PIXEL_FORMAT_BGR_888_PLANAR_INT16 = 123,// BGR888 Planar 每个像素的单分量占用1个16bit有符
号数
    HI_PIXEL_FORMAT_RGB_888_PLANAR_INT32 = 124,// RGB888 Planar 每个像素的单分量占用1个32bit有符
号数
    HI_PIXEL_FORMAT_BGR_888_PLANAR_INT32 = 125,// BGR888 Planar 每个像素的单分量占用1个32bit有符
号数
    HI_PIXEL_FORMAT_RGB_888_PLANAR_UINT16 = 126,// RGB888 Planar 每个像素的单分量占用1个16bit无
符号数
    HI_PIXEL_FORMAT_BGR_888_PLANAR_UINT16 = 127,// BGR888 Planar 每个像素的单分量占用1个16bit无
符号数
    HI_PIXEL_FORMAT_RGB_888_PLANAR_UINT32 = 128,// RGB888 Planar 每个像素的单分量占用1个32bit无
符号数
    HI_PIXEL_FORMAT_BGR_888_PLANAR_UINT32 = 129,// BGR888 Planar 每个像素的单分量占用1个32bit无
符号数
    HI_PIXEL_FORMAT_YUV400_UINT16 = 130,       // YUV400 Package 每个像素的单分量占用1个16bit无符号
数
    HI_PIXEL_FORMAT_YUV400_UINT32 = 131,       // YUV400 Package 每个像素的单分量占用1个32bit无符号
数
    HI_PIXEL_FORMAT_YUV400_UINT64 = 132,       // YUV400 Package 每个像素的单分量占用1个64bit无符号
数
    HI_PIXEL_FORMAT_YUV400_INT8   = 133,       // YUV400 Package 每个像素的单分量占用1个8bit有符号数
    HI_PIXEL_FORMAT_YUV400_INT16  = 134,       // YUV400 Package 每个像素的单分量占用1个16bit有符号数
    HI_PIXEL_FORMAT_YUV400_INT32  = 135,       // YUV400 Package 每个像素的单分量占用1个32bit有符号数
    HI_PIXEL_FORMAT_YUV400_INT64  = 136,       // YUV400 Package 每个像素的单分量占用1个64bit有符号数
    HI_PIXEL_FORMAT_YUV400_FP16 = 137,         // YUV400 Package 每个像素用1个float16数据表示
    HI_PIXEL_FORMAT_YUV400_FP32 = 138,         // YUV400 Package 每个像素用1个float32数据表示
    HI_PIXEL_FORMAT_YUV400_FP64 = 139,         // YUV400 Package 每个像素用1个float64数据表示
    HI_PIXEL_FORMAT_YUV400_BF16 = 140,         // YUV400 Package 每个像素用1个BFloat16数据表示，当前
不支持该格式

    HI_PIXEL_FORMAT_YUV_SEMIPLANAR_440 = 1000, // YUV440SP 8bit
    HI_PIXEL_FORMAT_YVU_SEMIPLANAR_440 = 1001, // YVU440SP 8bit
    HI_PIXEL_FORMAT_FLOAT32 = 1002,            // Float 32bit for 1pixel，每个像素用1个float32数据表示，当
前不支持该格式
    HI_PIXEL_FORMAT_BUTT = 1003,

    HI_PIXEL_FORMAT_RGB_888_PLANAR_FP16 = 1004,// RGB888 Planar 每个像素用1个float16数据表示
    HI_PIXEL_FORMAT_BGR_888_PLANAR_FP16 = 1005,// BGR888 Planar 每个像素用1个float16数据表示
    HI_PIXEL_FORMAT_RGB_888_PLANAR_FP32 = 1006,// RGB888 Planar 每个像素用1个float32数据表示
    HI_PIXEL_FORMAT_BGR_888_PLANAR_FP32 = 1007,// BGR888 Planar 每个像素用1个float32数据表示
    HI_PIXEL_FORMAT_RGB_888_PLANAR_BF16 = 1008,// RGB888 Planar 每个像素用1个BFloat16数据表示，
当前不支持该格式
    HI_PIXEL_FORMAT_BGR_888_PLANAR_BF16 = 1009,// BGR888 Planar 每个像素用1个BFloat16数据表示，
当前不支持该格式
    HI_PIXEL_FORMAT_RGB_888_FP16 = 1010,       // RGB888 Package，每个像素用1个float16数据表示
    HI_PIXEL_FORMAT_BGR_888_FP16 = 1011,       // BGR888 Package，每个像素用1个float16数据表示
    HI_PIXEL_FORMAT_RGB_888_FP32 = 1012,       // RGB888 Package，每个像素用1个float32数据表示
    HI_PIXEL_FORMAT_BGR_888_FP32 = 1013,       // BGR888 Package，每个像素用1个float32数据表示
    HI_PIXEL_FORMAT_RGB_888_BF16 = 1014,       // RGB888 Package 每个像素用1个BFloat16数据表示，当前
不支持该格式
    HI_PIXEL_FORMAT_BGR_888_BF16 = 1015,       // BGR888 Package 每个像素用1个BFloat16数据表示，当前
不支持该格式

    HI_PIXEL_FORMAT_UNKNOWN = 10000
} hi_pixel_format;
```

参考信息

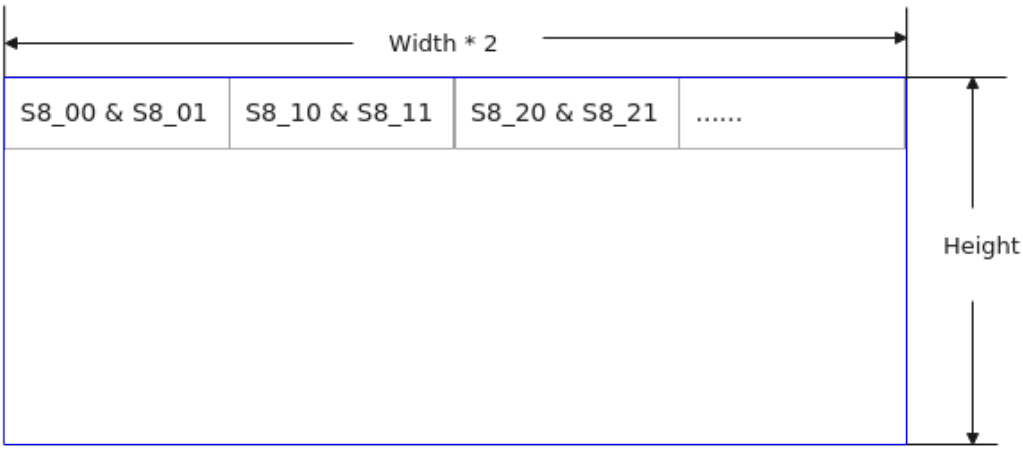
下文以HI_PIXEL_FORMAT_S8C1、HI_PIXEL_FORMAT_S8C2_PACKAGE、HI_PIXEL_FORMAT_S8C2_PLANAR为例说明排布格式，供参考。

图 11-21 HI_PIXEL_FORMAT_S8C1



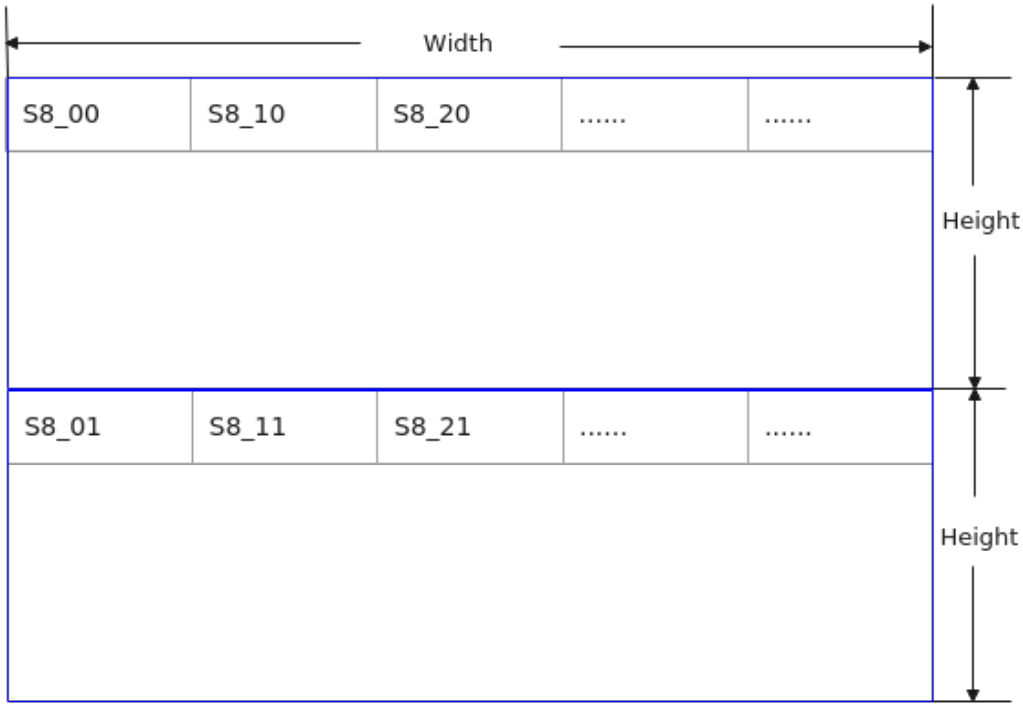
n表示bit位数
Width表示图片宽
Height表示图片高

图 11-22 HI_PIXEL_FORMAT_S8C2_PACKAGE



Width表示图片宽
Height表示图片高
S8_00中的第一个数字0表示像素点，第二个数字0表示通道号

图 11-23 HI_PIXEL_FORMAT_S8C2_PLANAR



Width表示图片宽
Height表示图片高
S8_00中的第一个数字0表示像素点，第二个数字0表示通道号

11.18.20.1.3 hi_payload_type

说明

定义图像或视频类型枚举。

目前DVPP模块仅支持HI_PT_H264、HI_PT_H265、HI_PT_JPEG、HI_PT_VPC，HI_PT_PNG，其中各功能支持的图像或视频类型如下：

- VPC：HI_PT_VPC
- JPEGD：HI_PT_JPEG
- JPEGE：HI_PT_JPEG
- VDEC：HI_PT_H264、HI_PT_H265
- VENC：HI_PT_H264、HI_PT_H265
- PNGD：HI_PT_PNG

定义

```
typedef enum {
    HI_PT_PCMU          = 0,
    HI_PT_1016          = 1,
    HI_PT_G721          = 2,
    HI_PT_GSM           = 3,
    HI_PT_G723          = 4,
```

```
HI_PT_DVI4_8K        = 5,
HI_PT_DVI4_16K       = 6,
HI_PT_LPC            = 7,
HI_PT_PCMA           = 8,
HI_PT_G722           = 9,
HI_PT_S16BE_STEREO   = 10,
HI_PT_S16BE_MONO     = 11,
HI_PT_QCELP          = 12,
HI_PT_CN             = 13,
HI_PT_MPEGAUDIO      = 14,
HI_PT_G728           = 15,
HI_PT_DVI4_3         = 16,
HI_PT_DVI4_4         = 17,
HI_PT_G729           = 18,
HI_PT_G711A          = 19,
HI_PT_G711U          = 20,
HI_PT_G726           = 21,
HI_PT_G729A          = 22,
HI_PT_LPCM           = 23,
HI_PT_CelB           = 25,
HI_PT_JPEG           = 26,
HI_PT_CUSM           = 27,
HI_PT_NV             = 28,
HI_PT_PICW           = 29,
HI_PT_CPV            = 30,
HI_PT_H261           = 31,
HI_PT_MPEGVIDEO      = 32,
HI_PT_MPEG2TS        = 33,
HI_PT_H263           = 34,
HI_PT_SPEG           = 35,
HI_PT_MPEG2VIDEO     = 36,
HI_PT_AAC            = 37,
HI_PT_WMA9STD        = 38,
HI_PT_HEAAC          = 39,
HI_PT_PCM_VOICE      = 40,
HI_PT_PCM_AUDIO      = 41,
HI_PT_MP3            = 43,
HI_PT_ADPCMA         = 49,
HI_PT_AEC            = 50,
HI_PT_X_LD           = 95,
HI_PT_H264           = 96,
HI_PT_D_GSM_HR       = 200,
HI_PT_D_GSM_EFR      = 201,
HI_PT_D_L8           = 202,
HI_PT_D_RED          = 203,
HI_PT_D_VDVI         = 204,
HI_PT_D_BT656        = 220,
HI_PT_D_H263_1998    = 221,
HI_PT_D_MP1S         = 222,
HI_PT_D_MP2P         = 223,
HI_PT_D_BMPEG        = 224,
HI_PT_MP4VIDEO       = 230,
HI_PT_MP4AUDIO       = 237,
HI_PT_VC1            = 238,
HI_PT_JVC_ASF        = 255,
HI_PT_D_AVI          = 256,
HI_PT_DIVX3          = 257,
HI_PT_AVS            = 258,
HI_PT_REAL8          = 259,
HI_PT_REAL9          = 260,
HI_PT_VP6            = 261,
HI_PT_VP6F           = 262,
HI_PT_VP6A           = 263,
HI_PT_SORENSON       = 264,
HI_PT_H265           = 265,
HI_PT_VP8            = 266,
HI_PT_MVC            = 267,
HI_PT_PNG            = 268,
HI_PT_AMR            = 1001,
```

```
    HI_PT_MJPEG          = 1002,
    HI_PT_AMRWB          = 1003,
    HI_PT_PRORES         = 1006,
    HI_PT_OPUS           = 1007,
    HI_PT_VPC            = 2000,
    HI_PT_BUTT
} hi_payload_type;
```

11.18.20.1.4 hi_video_field

说明

定义视频图像帧场类型。

定义

```
typedef enum {
    HI_VIDEO_FIELD_TOP = 0x1,
    HI_VIDEO_FIELD_BOTTOM = 0x2,
    HI_VIDEO_FIELD_INTERLACED = 0x3,
    HI_VIDEO_FIELD_FRAME = 0x4,
    HI_VIDEO_FIELD_BUTT
} hi_video_field;
```

成员

成员名称	描述
HI_VIDEO_FIELD_TOP	顶场类型。
HI_VIDEO_FIELD_BOTTOM	底场类型。
HI_VIDEO_FIELD_INTERLACED	两场间插类型。
HI_VIDEO_FIELD_FRAME	帧类型。
HI_VIDEO_FIELD_BUTT	保留值。

11.18.20.1.5 hi_dynamic_range

说明

定义动态范围枚举。

定义

```
typedef enum {
    HI_DYNAMIC_RANGE_SDR8 = 0,
    HI_DYNAMIC_RANGE_SDR10,
    HI_DYNAMIC_RANGE_HDR10,
    HI_DYNAMIC_RANGE_HLG,
    HI_DYNAMIC_RANGE_SLF,
    HI_DYNAMIC_RANGE_XDR,
    HI_DYNAMIC_RANGE_BUTT
} hi_dynamic_range;
```

成员

成员名称	描述
HI_DYNAMIC_RANGE_SDR8	8bit 数据的标准动态范围。
HI_DYNAMIC_RANGE_SDR10	10bit 数据的标准动态范围。
HI_DYNAMIC_RANGE_HDR10	10bit 数据的高动态范围。
HI_DYNAMIC_RANGE_HLG	带有混合对数的Gamma。
HI_DYNAMIC_RANGE_SLF	暂时无效。
HI_DYNAMIC_RANGE_XDR	10bit 数据，算法处理的一个中间类型数据，用户无需关心。
HI_DYNAMIC_RANGE_BUTT	保留值。

注意事项

各个动态范围对应的曲线如下：

Dynamic Range	Transfer Characteristic
SDR8/ SDR10	$V = \alpha * Lc^{0.45} - (\alpha - 1)$for $1 >= L_c >= \beta$ $V = 4.500 * L_c$for $\beta > L_c >= 0$
HDR10	对于L_c的所有值，$V = ((c_1 + c_2 * L_c^n) \div (1 + c_3 * L_c^n))$ $c_1 = c_3 - c_2 + 1 = 3424 \div 4096 = 0.8359375$ $c_2 = 32 * 2413 \div 4096 = 18.8515625$ $c_3 = 32 * 2392 \div 4096 = 18.6875$ $m = 128 * 2523 \div 4096 = 78.84375$ $n = 0.25 * 2610 \div 4096 = 0.1593017578125$ 其中，峰白区L_c等于1通常是为了表示参考输出发光亮度，即10000坎德拉每平方米。
HLG	$V = a * Ln(12 * L_c - b) + c$ for $1 >= L_c > 1 \div 12$ $V = Sqrt(3) * L_c^{0.5}$ for $1 \div 12 >= L_c >= 0$ $a = 0.17883277, b = 0.28466892, c = 0.55991073$

11.18.20.1.6 hi_color_gamut

说明

定义色域范围枚举。

定义

```
typedef enum {
    HI_COLOR_GAMUT_BT601 = 0,
    HI_COLOR_GAMUT_BT709,
    HI_COLOR_GAMUT_BT2020,
    HI_COLOR_GAMUT_USER,
    HI_COLOR_GAMUT_BUTT
} hi_color_gamut;
```

成员

成员名称	描述
HI_COLOR_GAMUT_BT601	BT601色域
HI_COLOR_GAMUT_BT709	BT709色域。
HI_COLOR_GAMUT_BT2020	BT2020色域。
HI_COLOR_GAMUT_USER	用户自定义的色域，非标准色域。
HI_COLOR_GAMUT_BUTT	保留值。

11.18.20.1.7 hi_video_supplement

说明

定义视频图像帧补充信息。

定义

```
typedef struct {
    hi_u64 misc_info_phys_addr;
    hi_u64 jpeg_dcf_phys_addr;
    hi_u64 isp_info_phys_addr;
    hi_u64 low_delay_phys_addr;
    hi_u64 bnr_rnt_phys_addr;
    hi_u64 motion_data_phys_addr;
    union {
        hi_u64 frame_dng_phys_addr;
        hi_u64 offset;
    }
    hi_void* ATTRIBUTE misc_info_virt_addr;
    hi_void* ATTRIBUTE jpeg_dcf_virt_addr;
    hi_void* ATTRIBUTE isp_info_virt_addr;
    hi_void* ATTRIBUTE low_delay_virt_addr;
    hi_void* ATTRIBUTE bnr_mot_virt_addr;
    hi_void* ATTRIBUTE motion_data_virt_addr;
    hi_void* ATTRIBUTE frame_dng_virt_addr;
} hi_video_supplement;
```

成员

成员名称	描述
misc_info_phys_addr	视频扩展信息的物理地址。暂不支持，用户不能配置修改。

成员名称	描述
jpeg_dcf_phys_addr	Jpeg DCF（Design rule for Camera File system）信息的物理地址。暂不支持，用户不能配置，系统使用默认值0。
isp_info_phys_addr	ISP（Image Signal Processor）辅助信息的物理地址，用户不能配置修改。
low_delay_phys_addr	低延时信息物理地址。暂不支持。
bnr_rnt_phys_addr	预留参数，暂不支持，用户不能配置修改。
motion_data_phys_addr	运动数据物理地址。暂不支持，用户不能配置修改。
frame_dng_phys_addr	预留参数，暂不支持，用户不能配置修改。
offset	图像偏移信息，仅配套hi_mpi_vi_send_pipe_raw接口使用，其它场景用户不能配置修改。
misc_info_virt_addr	视频扩展信息的虚拟地址。暂不支持，用户不能配置修改。
jpeg_dcf_virt_addr	Jpeg DCF信息的虚拟地址。暂不支持，用户不能配置修改。
isp_info_virt_addr	ISP辅助信息的内核态虚拟地址。
low_delay_virt_addr	低延时信息虚拟地址。暂不支持，用户不能配置修改。
bnr_mot_virt_addr	预留参数，暂不支持，用户不能配置修改。
motion_data_virt_addr	运动信息虚拟地址。暂不支持，用户不能配置修改。
frame_dng_virt_addr	预留参数，暂不支持，用户不能配置修改。

11.18.20.1.8 hi_video_frame

说明

定义视频原始图像帧结构。

定义

```
typedef struct {
    hi_u32              width;
    hi_u32              height;
    hi_video_field      field;
    hi_pixel_format     pixel_format;
    hi_video_format     video_format;
    hi_compress_mode    compress_mode;
    hi_dynamic_range    dynamic_range;
    hi_color_gamut      color_gamut;

    hi_u32              header_stride[HI_MAX_COLOR_COMPONENT];
    hi_u32              width_stride[HI_MAX_COLOR_COMPONENT];
    hi_u32              height_stride[HI_MAX_COLOR_COMPONENT];
```

```
    hi_u64          header_phys_addr[HI_MAX_COLOR_COMPONENT];
    hi_u64          phys_addr[HI_MAX_COLOR_COMPONENT];
    hi_void* ATTRIBUTE  header_virt_addr[HI_MAX_COLOR_COMPONENT];
    hi_void* ATTRIBUTE  virt_addr[HI_MAX_COLOR_COMPONENT];

    hi_u32          time_ref;
    hi_u64          pts;

    hi_u64          user_data[HI_MAX_USER_DATA_NUM];
    hi_u32          frame_flag;
    hi_video_supplement supplement;
} hi_video_frame;
```

成员

成员名称	描述
width	图像宽度。
height	图像高度。
field	帧场模式。预留参数。
pixel_format	视频图像像素格式。
video_format	视频图像格式。
compress_mode	视频压缩模式。
dynamic_range	动态范围。预留参数。
color_gamut	色域范围。预留参数。
header_stride	图像压缩头跨距。预留参数。
width_stride	输出图像分量的宽度数据跨距。YUV图像则为Y、U、V分量的数据跨距。RGB图像则为R、G、B分量的数据跨距。
height_stride	输出图像分量的高度数据跨距。YUV图像则为Y、U、V分量的数据跨距。RGB图像则为R、G、B分量的数据跨距。 若涉及VI视频采集、ISP系统控制、VPSS视频处理、VENC视频编码、JPEGE图片编码功能时，该参数作为预留参数，暂不支持。
header_phys_addr	压缩头物理地址。预留参数。
phys_addr	物理地址。预留参数。
header_virt_addr	压缩头虚拟地址。预留参数。
virt_addr	图像在Device内存中的起始虚拟地址。 作为输入图像时，对于YUV图像，virt_addr[0]为图像起始地址，也是Y分量起始地址，virt_addr[1]为U分量起始地址，virt_addr[2]为V分量起始地址； 作为输出图像时，对于YUV和RGB图像，virt_addr[0]为图像在device上的起始地址，virt_addr[1]和virt_addr[2]为保留字段。

成员名称	描述
time_ref	图像帧序列号。 解码场景下，该参数预留。 JPEGE编码场景下，该参数预留。 VENC编码场景下，针对每一帧设置该参数时，按照帧的顺序，该参数值需递增且为偶数，例如，第一帧时该参数值设置为2，第二帧时该参数值设置为4，以此类推。
pts	图像时间戳。
user_data	私有数据。预留参数。
frame_flag	• **在VDEC解码场景下**，该参数表示该帧解码是否成功，不管成功或失败，用户都需及时释放输入、输出内存，对于解码失败的帧，系统不保证按照显示序或解码序按顺序输出，本参数取值范围： – 0：解码成功。 – 1：解码失败，可能由于输入码流异常、设置的码流格式与实际码流不一致等问题导致。 – 2：隔行码流场景下使用，隔行码流每帧发送两场，带两块输出内存；解码时其中一块内存无图像输出，属于正常现象，用户需及时释放内存；隔行码流的解码输出数据都在奇数场对应的输出buffer中。 – 3：参考帧个数设置错误导致解码失败。 – 4：VDEC解码帧存大小设置错误导致解码失败。 • **在JPEGD解码场景下**，该参数表示该帧解码是否成功，不管成功或失败，用户都需及时释放输入、输出内存，本参数取值范围： – 0：解码成功。 – 1：解码失败，可能由于输入码流异常、设置的码流格式与实际码流不一致等问题导致。 – 4：JPEGD解码输出内存大小过小导致解码失败，可提前调用**hi_mpi_vdec_get_jpegd_output_info**接口获取输出内存大小。 – 5：仅JPEGD解码场景使用，表示JPEGD解码配置参数设置错误，例如，设置的宽高超出限制、设置的目标解码格式不支持等。 • **在VENC编码场景下**，该参数表示是否使能slice_temporal_mvp_enabled_flag语法（表示帧间预测是否使用时域MV（MotionVector）预测），取值范围如下，但**仅H265协议下该参数生效**： – 0：slice_temporal_mvp_enabled_flag为True，表示使用时域MV预测。 – 1：slice_temporal_mvp_enabled_flag为False，表示不使用时域MV预测。 • **在JPEGE编码场景下**，取值0或1，**但当前版本暂未使用该参数，参数值无效**。

成员名称	描述
supplement	图像的补充信息。 Atlas 推理系列产品上，该参数为透传参数。

注意事项

10bit数据不压缩时在内存中的存储方式是紧凑排列的。

10bit YUV段压缩数据存储是前8bit和后2bit分开存储。

10bit tile 64*16 数据（即VDH（video decoder hardware）解码之后的数据）不管是压缩还是非压缩，在内存中的存储方式都是前8bit和后2bit分开存储。

11.18.20.1.9 hi_video_frame_info

说明

定义视频图像帧信息结构体。

定义

```
typedef struct {
    hi_video_frame v_frame;
    hi_u32         pool_id;
    hi_mod_id      mod_id;
} hi_video_frame_info;
```

成员

成员名称	描述
v_frame	视频图像帧属性结构体。
pool_id	视频缓存池ID，预留参数。
mod_id	DVPP内的子模块ID，预留参数。

11.18.20.1.10 hi_vb_src

说明

定义VB来源选择。

定义

```
typedef enum  {
    HI_VB_SRC_COMMON  = 0,  //公共VB
    HI_VB_SRC_MOD     = 1, //模块VB
    HI_VB_SRC_PRIVATE = 2, //私有VB
    HI_VB_SRC_USER    = 3, //用户VB
    HI_VB_SRC_BUTT
} hi_vb_src;
```

11.18.20.1.11 hi_mod_id

说明

定义模块ID枚举类型。

定义

```
typedef enum {
    HI_ID_CMPI = 0,
    HI_ID_VB = 1,
    HI_ID_SYS = 2,
    HI_ID_RGN = 3,
    HI_ID_CHNL = 4,
    HI_ID_VDEC = 5,
    HI_ID_AVS = 6,
    HI_ID_VPC = 7,
    HI_ID_VENC = 8,
    HI_ID_SVP = 9,
    HI_ID_H264E = 10,
    HI_ID_JPEGE = 11,
    HI_ID_MPEG4E = 12,
    HI_ID_H265E = 13,
    HI_ID_JPEGD = 14,
    HI_ID_VO = 15,
    HI_ID_VI = 16,
    HI_ID_DIS = 17,
    HI_ID_VALG = 18,
    HI_ID_RC = 19,
    HI_ID_AIO = 20,
    HI_ID_AI = 21,
    HI_ID_AO = 22,
    HI_ID_AENC = 23,
    HI_ID_ADEC = 24,
    HI_ID_VPU = 25,
    HI_ID_PCIV = 26,
    HI_ID_PCIVFMW = 27,
    HI_ID_ISP = 28,
    HI_ID_IVE = 29,
    HI_ID_USER = 30,
    HI_ID_DCCM = 31,
    HI_ID_DCCS = 32,
    HI_ID_PROC = 33,
    HI_ID_LOG = 34,
    HI_ID_VFMW = 35,
    HI_ID_H264D = 36,
    HI_ID_GDC = 37,
    HI_ID_PHOTO = 38,
    HI_ID_FB = 39,
    HI_ID_HDMI = 40,
    HI_ID_VOIE = 41,
    HI_ID_TDE = 42,
    HI_ID_HDR = 43,
    HI_ID_PRORES = 44,
    HI_ID_VGS = 45,

    HI_ID_FD = 47,
    HI_ID_ODT = 48,
    HI_ID_VQA = 49,
    HI_ID_LPR = 50,
    HI_ID_SVP_NNIE = 51,
    HI_ID_SVP_DSP = 52,
    HI_ID_DPU_RECT = 53,
    HI_ID_DPU_MATCH = 54,

    HI_ID_MOTIONSENSOR = 55,
    HI_ID_MOTIONFUSION = 56,
```

```
    HI_ID_GYRODIS = 57,
    HI_ID_PM = 58,
    HI_ID_SVP_ALG = 59,
    HI_ID_IVP = 60,
    HI_ID_MCF = 61,
    HI_ID_VPSS = 62,
    HI_ID_DRV_VPC = 63,
    HI_ID_PNGD = 64,
    HI_ID_BUTT = 0x100,
} hi_mod_id;
```

11.18.20.1.12 hi_data_bit_width

说明

定义数据比特位宽类型。

目前DVPP各功能仅支持HI_DATA_BIT_WIDTH_8和HI_DATA_BIT_WIDTH_10两个数据位宽格式。

目前VI功能支持HI_DATA_BIT_WIDTH_8、HI_DATA_BIT_WIDTH_10、HI_DATA_BIT_WIDTH_12、HI_DATA_BIT_WIDTH_14几种数据位宽格式。

定义

```
typedef enum  {
    HI_DATA_BIT_WIDTH_8 = 0,
    HI_DATA_BIT_WIDTH_10,
    HI_DATA_BIT_WIDTH_12,
    HI_DATA_BIT_WIDTH_14,
    HI_DATA_BIT_WIDTH_16,
    HI_DATA_BIT_WIDTH_BUTT
} hi_data_bit_width;
```

11.18.20.1.13 hi_video_format

说明

定义视频格式枚举。

定义

```
typedef enum {
    HI_VIDEO_FORMAT_LINEAR = 0,
    HI_VIDEO_FORMAT_TILE_64x16,
    HI_VIDEO_FORMAT_BUTT
} hi_video_format;
```

成员

成员名称	描述
HI_VIDEO_FORMAT_LINEAR	线性存储的视频格式。
HI_VIDEO_FORMAT_TILE_64x16	TILE 格式存储的视频格式，其中，tile的块大小为64像素宽，16行像素高，视频/图像编码暂不支持。
HI_VIDEO_FORMAT_BUTT	保留值。

11.18.20.1.14 hi_img_info

说明

定义图片信息的结构体。

定义

```
typedef struct {
    hi_u32 width;
    hi_u32 height;
    hi_u32 width_stride;
    hi_u32 height_stride;
    hi_u32 img_buf_size;
    union {
        hi_jpeg_raw_format pixel_format;
        hi_png_color_format png_pixel_format;
        hi_pixel_format img_pixel_format;
    };
    hi_u32 reserved[4];
} hi_img_info;
```

成员

成员名称	描述
width	图片宽。
height	图片高。
width_stride	图片宽stride。
height_stride	图片高stride。
img_buf_size	存放图片的内存大小，单位是Byte。
pixel_format	JPEGD图片格式。
png_pixel_format	PNGD图片格式。
img_pixel_format	通用图片格式。
reserved	预留参数，为保证后续版本兼容性，请务必使用memset结构体方式进行清零初始化，在代码中必须避免显式对reserved字段进行访问。

11.18.20.1.15 hi_pic_buf_attr

说明

定义图片缓冲区信息的结构体。

定义

```
typedef struct {
    hi_u32 width;
    hi_u32 height;
```

```
    hi_u32 align;
    hi_data_bit_width bit_width;
    hi_pixel_format pixel_format;
    hi_compress_mode compress_mode;
} hi_pic_buf_attr;
```

成员

成员名称	描述
width	图片宽。
height	图片高。
align	对齐参数，预留参数，暂不支持。
bit_width	图片位宽。
pixel_format	图片格式。 VDEC视频解码时，暂不使用该参数；JPEGD图像解码时，表示解码输出图像格式，当前支持的输出图像格式请参见**11.18.17.1 JPEGD功能及约束说明**。
compress_mode	压缩模式。

11.18.20.1.16 hi_jpeg_raw_format

说明

定义JPEG源图片格式的枚举。

定义

```
typedef enum {
    HI_JPEG_RAW_FORMAT_YUV444 = 0,
    HI_JPEG_RAW_FORMAT_YUV422 = 1,
    HI_JPEG_RAW_FORMAT_YUV420 = 2,
    HI_JPEG_RAW_FORMAT_YUV440 = 3,
    HI_JPEG_RAW_FORMAT_YUV400 = 4,
    HI_JPEG_RAW_FORMAT_YUV411 = 5,
    HI_JPEG_RAW_FORMAT_MAX = 100
} hi_jpeg_raw_format;
```

11.18.20.1.17 hi_dvpp_epoll_ctl_op

说明

定义DVPP epoll操作类型。

定义

```
typedef enum {
    HI_DVPP_EPOLL_CTL_ADD = 1,
    HI_DVPP_EPOLL_CTL_MOD = 2,
    HI_DVPP_EPOLL_CTL_DEL = 3,
    HI_DVPP_EPOLL_CTL_BUTT
} hi_dvpp_epoll_ctl_op;
```

成员

成员名称	描述
HI_DVPP_EPOLL_CTL_ADD	向epoll实例中添加新的DVPP通道的文件句柄。
HI_DVPP_EPOLL_CTL_MOD	修改epoll实例中已有DVPP通道的事件类型或对应的用户数据。
HI_DVPP_EPOLL_CTL_DEL	从epoll实例中删除DVPP通道的文件句柄。
HI_DVPP_EPOLL_CTL_BUTT	保留值。

11.18.20.1.18 hi_dvpp_epoll_event_type

说明

定义DVPP epoll事件类型。

定义

```
typedef enum {
    HI_DVPP_EPOLL_IN = 1u,
    HI_DVPP_EPOLL_OUT = 1u << 1u,
    HI_DVPP_EPOLL_ET = (hi_u32)1u << 31u
} hi_dvpp_epoll_event_type;
```

成员

成员名称	描述
HI_DVPP_EPOLL_IN	可读事件，标识DVPP通道中存在输出数据。
HI_DVPP_EPOLL_OUT	可写事件，标识DVPP通道可以进行新的数据输入。预留功能，暂不支持。
HI_DVPP_EPOLL_ET	边缘触发方式，不设定则DVPP epoll默认为水平触发。

11.18.20.1.19 hi_dvpp_epoll_event

说明

定义DVPP epoll事件信息。

定义

```
typedef struct {
    hi_u32 events;
```

```
    hi_void *data;
} hi_dvpp_epoll_event;
```

成员

成员名称	描述
events	DVPP epoll事件类型 ，由hi_dvpp_epoll_event_type中定义的类型组成的位掩码（bitmask），例如处理可读和可写两种事件时，此处配置为：HI_DVPP_EPOLL_IN | HI_DVPP_EPOLL_OUT。
data	关联到DVPP通道的用户自定义数据。

11.18.20.1.20 hi_csc_matrix

说明

定义色域转换矩阵的数据标准。

基于各标准的色域转换矩阵，其各分量的转换公式如下：

```
# YUV转RGB：
# | R |   | matrix_r0c0 matrix_r0c1 matrix_r0c2 |   | Y - input_bias_0 |
# | G | = | matrix_r1c0 matrix_r1c1 matrix_r1c2 | * | U - input_bias_1 |
# | B |   | matrix_r2c0 matrix_r2c1 matrix_r2c2 |   | V - input_bias_2 |
# BGR转YUV：
# | Y |   | output_bias_0 |   | matrix_r0c0 matrix_r0c1 matrix_r0c2 |   | R |
# | U | = | output_bias_1 | + | matrix_r1c0 matrix_r1c1 matrix_r1c2 | * | G |
# | V |   | output_bias_2 |   | matrix_r2c0 matrix_r2c1 matrix_r2c2 |   | B |
```

定义

```
typedef enum {
   HI_CSC_MATRIX_BT601_WIDE = 0,
   HI_CSC_MATRIX_BT601_NARROW,
   HI_CSC_MATRIX_BT709_WIDE,
   HI_CSC_MATRIX_BT709_NARROW,
   HI_CSC_MATRIX_BT2020_WIDE,
   HI_CSC_MATRIX_BT2020_NARROW,
   HI_CSC_MATRIX_USER = 100,
   HI_CSC_MATRIX_BUTT
} hi_csc_matrix;
```

成员

表 11-57 色域转换矩阵的数据标准

成员名称	描述
HI_CSC_MATRIX_BT601_WIDE	基于BT601 wide标准的色域转换矩阵。默认为HI_CSC_MATRIX_BT601_WIDE。 基于该标准的色域转换矩阵，各参数值如下： # YUV转RGB： # | R | | 1.000 0.000 1.402 | | Y - 0 | # | G | = | 1.000 -0.344 -0.714 | * | U - 128 | # | B | | 1.000 1.772 0.000 | | V - 128 | # RGB转YUV： # | Y | |-0.5 | | 0.299 0.587 0.114 | | R | # | U | = |127.5 | + | -0.168 -0.331 0.500 | * | G | # | V | |127.5 | | 0.500 -0.419 -0.081 | | B |
HI_CSC_MATRIX_BT601_NARROW	基于BT601 narrow标准的色域转换矩阵。 基于该标准的色域转换矩阵，各参数值如下： # YUV转RGB： # | R | | 1.16438 0.00000 1.59602 | | Y - 16 | # | G | = | 1.16438 -0.39176 -0.81297 | * | U - 128 | # | B | | 1.16438 2.01723 0.00000 | | V - 128 | # RGB转YUV： # | Y | |16 | | 0.25679 0.51564 0.10014 | | R | # | U | = |128| + | -0.14491 -0.29099 0.43922 | * | G | # | V | |128| | 0.42941 -0.36779 -0.07143 | | B |
HI_CSC_MATRIX_BT709_WIDE	基于BT709 wide标准的色域转换矩阵。 基于该标准的色域转换矩阵，各参数值如下： # YUV转RGB： # | R | | 1.00000 0.00000 1.57480 | | Y - 0 | # | G | = | 1.00000 -0.18732 -0.46812 | * | U - 128 | # | B | | 1.00000 1.85560 0.00000 | | V - 128 | # RGB转YUV： # | Y | |0 | | 0.21260 0.71520 0.07220 | | R | # | U | = |128| + | -0.11457 -0.38543 0.50000 | * | G | # | V | |128| | 0.50000 -0.45415 -0.04585 | | B |
HI_CSC_MATRIX_BT709_NARROW	基于BT709 narrow标准的色域转换矩阵。 基于该标准的色域转换矩阵，各参数值如下： # YUV转RGB： # | R | | 1.16438 0.00000 1.79274 | | Y - 16 | # | G | = | 1.16438 -0.21325 -0.53291 | * | U - 128 | # | B | | 1.16438 2.11240 0.00000 | | V - 128 | # RGB转YUV： # | Y | |16 | | 0.18259 0.62825 0.06342 | | R | # | U | = |128| + | -0.09840 -0.33857 0.43922 | * | G | # | V | |128| | 0.42941 -0.39894 -0.04027 | | B |
HI_CSC_MATRIX_BT2020_WIDE	基于BT2020 wide标准的色域转换矩阵。 基于该标准的色域转换矩阵，各参数值如下： # YUV转RGB： # | R | | 1.00000 0.00000 1.47460 | | Y - 0 | # | G | = | 1.00000 -0.16455 -0.57135 | * | U - 128 | # | B | | 1.00000 1.88140 0.00000 | | V - 128 | # RGB转YUV： # | Y | |0 | | 0.26270 0.67800 0.05930 | | R | # | U | = |128| + | -0.13963 -0.36037 0.50000 | * | G | # | V | |128| | 0.50000 -0.45979 -0.04021 | | B |

成员名称	描述
HI_CSC_MATRIX_BT2020_NARROW	基于BT2020 narrow标准的色域转换矩阵。 基于该标准的色域转换矩阵，各参数值如下： # YUV转RGB： # | R | | 1.16438 0.00000 1.67868 | | Y - 16 | # | G | = | 1.16438 -0.18733 -0.65042 | * | U - 128 | # | B | | 1.16438 2.14177 0.00000 | | V - 128 | # RGB转YUV： # | Y | |16 | | 0.22564 0.59558 0.05209 | | R | # | U | = |128| + | -0.11992 -0.31656 0.43922 | * | G | # | V | |128| | 0.42941 -0.40389 -0.03533 | | B |
HI_CSC_MATRIX_USER	用户自定义色域转换矩阵。
HI_CSC_MATRIX_BUTT	保留值。

11.18.20.1.21 hi_coefficient

说明

定义色域转换矩阵参数。

定义

```
typedef struct {
    hi_double csc_matrix_r0_c0;
    hi_double csc_matrix_r0_c1;
    hi_double csc_matrix_r0_c2;
    hi_double csc_matrix_r1_c0;
    hi_double csc_matrix_r1_c1;
    hi_double csc_matrix_r1_c2;
    hi_double csc_matrix_r2_c0;
    hi_double csc_matrix_r2_c1;
    hi_double csc_matrix_r2_c2;
    hi_double csc_bias_r0;
    hi_double csc_bias_r1;
    hi_double csc_bias_r2;
} hi_coefficient;
```

成员

成员名称	描述
csc_matrix_r0_c0	设置色域转换矩阵参数R0C0。
csc_matrix_r0_c1	设置色域转换矩阵参数R0C1。
csc_matrix_r0_c2	设置色域转换矩阵参数R0C2。
csc_matrix_r1_c0	设置色域转换矩阵参数R1C0。
csc_matrix_r1_c1	设置色域转换矩阵参数R1C1。
csc_matrix_r1_c2	设置色域转换矩阵参数R1C2。

成员名称	描述
csc_matrix_r2_c0	设置色域转换矩阵参数R2C0。
csc_matrix_r2_c1	设置色域转换矩阵参数R2C1。
csc_matrix_r2_c2	设置色域转换矩阵参数R2C2。
csc_bias_r0	设置偏移R0。
csc_bias_r1	设置偏移R1。
csc_bias_r2	设置偏移R2。

11.18.20.1.22 hi_csc_coefficient

说明

定义色域转换矩阵参数的结构体。

定义

```
typedef struct {
    hi_coefficient yuv_to_rgb_coefficient;
    hi_coefficient rgb_to_yuv_coefficient;
} hi_csc_coefficient;
```

成员

成员名称	描述
yuv_to_rgb_coefficient	yuv转rgb的色域转换参数。
rgb_to_yuv_coefficient	rgb转yuv的色域转换参数。

11.18.20.1.23 hi_video_size

说明

预留结构体，暂不支持。

定义

```
typedef struct {
    hi_u32      width;
    hi_u32      height;
} hi_video_size;
```

11.18.20.1.24 hi_img_stream

说明

定义图片码流信息。

定义

```
typedef struct {
    hi_payload_type    type;
    hi_u8 *ATTRIBUTE   addr;
    hi_u32    len;
    hi_u64    pts;
    hi_s32    reserved[2];
} hi_img_stream;
```

成员

成员名称	描述
type	图片类型。
addr	图片在内存中的起始虚拟地址。
len	存放图片的内存大小，单位Byte。
pts	图片时间戳，预留参数。
reserved	预留参数，为保证后续版本兼容性，请务必使用memset结构体方式进行清零初始化，在代码中必须避免显式对reserved字段进行访问。

11.18.20.1.25 hi_pic_info

说明

定义图片信息。

定义

```
typedef struct {
    hi_void* picture_address;
    hi_u32    picture_buffer_size;
    hi_u32    picture_width;
    hi_u32    picture_height;
    hi_u32    picture_width_stride;
    hi_u32    picture_height_stride;
    hi_pixel_format picture_format;
} hi_pic_info;
```

成员

成员名称	描述
picture_address	图片在Device内存中的起始虚拟地址。

成员名称	描述
picture_buffer_size	存放图片的内存大小，单位Byte。
picture_width	图片宽。
picture_height	图片高。
picture_width_stride	图片宽stride。
picture_height_stride	图片高stride。
picture_format	图片格式。

11.18.20.1.26 hi_png_color_format

说明

定义PNG源图片格式的枚举。

定义

```
typedef enum {
    HI_PNG_COLOR_FORMAT_GRAY  = 0x0,   // gray bitmap
    HI_PNG_COLOR_FORMAT_RGB   = 0x2,   // RGB bitmap
    HI_PNG_COLOR_FORMAT_CLUT  = 0x3,   // PLTE调色板数据
    HI_PNG_COLOR_FORMAT_AGRAY = 0x4,   // gray bitmap with alpha
    HI_PNG_COLOR_FORMAT_ARGB  = 0x6,   // RGB bitmap with alpha
    HI_PNG_COLOR_FORMAT_BUTT  = 0x100
} hi_png_color_format;
```

11.18.20.1.27 hi_rect

说明

定义矩形宽高和左上角坐标

定义

```
typedef struct {
    hi_s32 x;
    hi_s32 y;
    hi_s32 width;
    hi_s32 height;
} hi_rect;
```

成员

成员名称	描述
x	左上角x轴
y	左上角y轴
width	矩形宽度

成员名称	描述
height	矩形高度

11.18.20.1.28 hi_op_mode

说明

定义操作模式枚举。

定义

```
typedef enum {
    HI_OP_MODE_AUTO = 0,   // 自动模式，一般此模式使用程序内部的默认参数
    HI_OP_MODE_MANUAL = 1    // 手动模式，一般此模式使用用户配置的参数
} hi_op_mode;
```

11.18.20.1.29 hi_point

说明

定义坐标点结构体。

定义

```
typedef struct {
    hi_s32 x;
    hi_s32 y;
} hi_point;
```

成员

成员名称	描述
x	x轴坐标值。
y	y轴坐标值。

11.18.20.1.30 hi_crop_info

说明

裁剪CROP 属性结构体。

定义

```
typedef struct {
    hi_bool enable;
    hi_rect  rect;
} hi_crop_info;
```

成员

成员名称	描述
enable	是否开启裁剪功能。 取值范围： ● HI_FALSE：不开启。 ● HI_TRUE：开启。
rect	裁剪起始位置与宽高信息。

11.18.20.1.31 hi_frame_rate_ctrl

说明

定义帧率控制结构体 。

定义

```
typedef struct {
    hi_s32 src_frame_rate;
    hi_s32 dst_frame_rate;
} hi_frame_rate_ctrl;
```

成员

成员名称	描述
src_frame_rate	源帧率，即输入帧率。 取值范围-1或[1, 240]。
dst_frame_rate	目的帧率，即输出帧率。当源帧率为-1时，目标帧率必须为-1，表示不进行帧率控制，其它情况下，目标帧率不能大于源帧率。 取值范围[-1, src_frame_rate]。

11.18.20.1.32 hi_compress_mode

说明

定义视频压缩数据格式枚举。

定义

```
typedef enum {
    HI_COMPRESS_MODE_NONE = 0,
    HI_COMPRESS_MODE_SEG,
    HI_COMPRESS_MODE_TILE,
    HI_COMPRESS_MODE_HFBC,
    HI_COMPRESS_MODE_LINE,
    HI_COMPRESS_MODE_FRAME,
```

```
    HI_COMPRESS_MODE_BUTT
} hi_compress_mode;
```

成员

成员名称	描述
HI_COMPRESS_MODE_NONE	不压缩。
HI_COMPRESS_MODE_SEG	段压缩，以256字节为一个压缩片段。
HI_COMPRESS_MODE_TILE	Tile压缩，以一个tile为一个压缩片段。
HI_COMPRESS_MODE_HFBC	本昇腾AI处理器特有的HFBC压缩模式，压缩单位是64*8。
HI_COMPRESS_MODE_LINE	行压缩，以一整行为一个压缩片段进行压缩。如ISP WDR模式下内部的RAW图像采用行压缩。
HI_COMPRESS_MODE_FRAME	帧压缩，以一整帧为一个压缩片段进行压缩。如ISP线性模式下内部的RAW图像采用帧压缩，或者3DNR内部参考帧和重构帧采用帧压缩。
HI_COMPRESS_MODE_BUTT	预留值。

11.18.20.1.33 hi_mpp_chn

说明

定义模块设备通道结构体。

定义

```
typedef struct {
    hi_mod_id    mod_id;
    hi_s32       dev_id;
    hi_s32       chn_id;
} hi_mpp_chn;
```

成员

成员名称	描述
mod_id	模块号。
dev_id	设备号。
chn_id	通道号。

11.18.20.1.34 hi_size

说明

定于矩形尺寸的结构体。

定义

```
typedef struct {
    hi_u32 width;
    hi_u32 height;
} hi_size;
```

成员

成员名称	描述
width	宽度
height	高度

11.18.20.1.35 hi_module_type

说明

定义模块类型。

定义

```
typedef enum {
    HI_MOD_VPC = 0,
    HI_MOD_VENC = 1,
    HI_MOD_VDEC = 2,
    HI_MOD_JPEGE = 3,
    HI_MOD_JPEGD = 4,
    HI_MOD_PNGD = 5,
    HI_MOD_ALL = 100,
    HI_MOD_BUTT = 255  // 预留值
} hi_module_type;
```

11.18.20.1.36 hi_img_base_info

说明

定义图片基本信息。

定义

```
typedef struct {
    hi_u32 width;
    hi_u32 height;
    hi_pixel_format pixel_format;
    hi_u64 reserved[2];
} hi_img_base_info;
```

成员

成员名称	描述
width	图片宽度。
height	图片高度。
pixel_format	图片格式。
reserved	预留参数，为保证后续版本兼容性，请务必使用memset结构体方式进行清零初始化，在代码中必须避免显式对reserved字段进行访问。

11.18.20.1.37 hi_img_align_info

说明

定义图片对齐信息。

定义

```
typedef struct {
    hi_u32 width_stride;
    hi_u32 height_stride;
    hi_u64 img_buf_size;
    hi_u64 reserved[2];
} hi_img_align_info;
```

成员

成员名称	描述
width_stride	图片宽度对齐后的值。
height_stride	图片高度对齐后的值。
img_buf_size	存放图片数据所需的内存大小，单位Byte。
reserved	预留参数，为保证后续版本兼容性，请务必使用memset结构体方式进行清零初始化，在代码中必须避免显式对reserved字段进行访问。

11.18.20.1.38 hi_coord

说明

定义矩形区域信息结构体。

定义

```
typedef enum {
    HI_COORD_ABS = 0,
```

```
    HI_COORD_RATIO
} hi_coord;
```

成员

成员名称	描述
HI_COORD_ABS	绝对坐标。
HI_COORD_RATIO	相对坐标，即坐标值是以与当前图像宽高的比率来表示，具体比率转换与实际使用接口为准。

11.18.20.1.39 hi_aspect_ratio_type

说明

定义幅型比类型。

定义

```
typedef enum  {
    HI_ASPECT_RATIO_NONE   = 0,        /* full screen */
    HI_ASPECT_RATIO_AUTO   = 1,        /* ratio no change, 1:1*/
    HI_ASPECT_RATIO_MANUAL = 2
} hi_aspect_ratio_type;
```

成员

成员名称	描述
HI_ASPECT_RATIO_NONE	无幅型比。
HI_ASPECT_RATIO_AUTO	自动模式。
HI_ASPECT_RATIO_MANUAL	手动模式。

11.18.20.1.40 hi_aspect_ratio

说明

定义坐标点信息结构体。

定义

```
typedef struct {
    hi_aspect_ratio_type mode;    /* aspect ratio mode: none/auto/manual */
    hi_u32          bg_color;      /* background color, RGB 888 */
    hi_rect         video_rect;    /* valid in ASPECT_RATIO_MANUAL mode */
} hi_aspect_ratio;
```

成员

成员名称	描述
mode	幅型比类型。
bg_color	幅型比背景颜色。
video_rect	幅型比视频区域。

11.18.20.1.41 hi_video_display_mode

说明

显示模式枚举。

定义

```
typedef enum {
    HI_VIDEO_DISPLAY_MODE_PREVIEW = 0,
    HI_VIDEO_DISPLAY_MODE_PLAYBACK = 1
 }hi_video_display_mode;
```

成员

成员名称	描述
HI_VIDEO_DISPLAY_MODE_PREVIEW	预览模式。
HI_VIDEO_DISPLAY_MODE_PLAYBACK	回放模式。

11.18.20.1.42 hi_wdr_mode

说明

定义宽动态模式。

定义

```
typedef enum {
    HI_WDR_MODE_NONE = 0,
    HI_WDR_MODE_BUILT_IN,
    HI_WDR_MODE_QUDRA,
    HI_WDR_MODE_2To1_LINE,
    HI_WDR_MODE_2To1_FRAME,
    HI_WDR_MODE_3To1_LINE,
    HI_WDR_MODE_3To1_FRAME,
    HI_WDR_MODE_4To1_LINE,
    HI_WDR_MODE_4To1_FRAME,
    HI_WDR_MODE_BUTT,
} hi_wdr_mode;
```

成员

成员名称	描述
HI_WDR_MODE_NONE	线性模式。
HI_WDR_MODE_BUILT_IN	Sensor合成WDR模式。
HI_WDR_MODE_QUDRA	QUDRA WDR模式。
HI_WDR_MODE_2To1_LINE	2 帧合成行WDR模式。
HI_WDR_MODE_2To1_FRAME	2 帧合成帧WDR模式。
HI_WDR_MODE_3To1_LINE	3 帧合成行WDR模式。
HI_WDR_MODE_3To1_FRAME	3 帧合成帧WDR模式。
HI_WDR_MODE_4To1_LINE	4 帧合成行WDR模式。
HI_WDR_MODE_4To1_FRAME	4 帧合成帧WDR模式。
HI_WDR_MODE_BUTT	预留值。

说明

此处的WDR模式配置，需要与VI、Sensor、MIPI RX配置保持一致，否则会导致出图异常。

11.18.20.2 音频相关

11.18.20.2.1 基本数据类型

```
typedef hi_s32 hi_audio_dev;
typedef hi_s32 hi_ai_chn;
typedef hi_s32 hi_ao_chn;
typedef hi_s32 hi_aenc_chn;
typedef hi_s32 hi_adec_chn;
```

11.18.20.2.2 hi_audio_sample_rate

说明

定义音频采样率。

定义

```
typedef enum {
    HI_AUDIO_SAMPLE_RATE_8000   = 8000,    /* 8kHz sample rate */
    HI_AUDIO_SAMPLE_RATE_12000  = 12000,   /* 12kHz sample rate */
    HI_AUDIO_SAMPLE_RATE_11025  = 11025,   /* 11.025kHz sample rate */
    HI_AUDIO_SAMPLE_RATE_16000  = 16000,   /* 16kHz sample rate */
    HI_AUDIO_SAMPLE_RATE_22050  = 22050,   /* 22.05kHz sample rate */
    HI_AUDIO_SAMPLE_RATE_24000  = 24000,   /* 24kHz sample rate */
    HI_AUDIO_SAMPLE_RATE_32000  = 32000,   /* 32kHz sample rate */
    HI_AUDIO_SAMPLE_RATE_44100  = 44100,   /* 44.1kHz sample rate */
    HI_AUDIO_SAMPLE_RATE_48000  = 48000,   /* 48kHz sample rate */
    HI_AUDIO_SAMPLE_RATE_64000  = 64000,   /* 64kHz sample rate */
    HI_AUDIO_SAMPLE_RATE_96000  = 96000,   /* 96kHz sample rate */
```

```
    HI_AUDIO_SAMPLE_RATE_BUTT
} hi_audio_sample_rate;
```

成员

成员名称	描述
HI_AUDIO_SAMPLE_RATE_8000	8kHz采样率。
HI_AUDIO_SAMPLE_RATE_12000	12kHz采样率。
HI_AUDIO_SAMPLE_RATE_11025	11.025kHz采样率。
HI_AUDIO_SAMPLE_RATE_16000	16kHz采样率。
HI_AUDIO_SAMPLE_RATE_22050	22.050kHz采样率。
HI_AUDIO_SAMPLE_RATE_24000	24kHz采样率。
HI_AUDIO_SAMPLE_RATE_32000	32kHz采样率。
HI_AUDIO_SAMPLE_RATE_44100	44.1kHz采样率。
HI_AUDIO_SAMPLE_RATE_48000	48kHz采样率。
HI_AUDIO_SAMPLE_RATE_64000	64kHz采样率。。
HI_AUDIO_SAMPLE_RATE_96000	96kHz采样率。
HI_AUDIO_SAMPLE_RATE_BUTT	预留值。

11.18.20.2.3 hi_aio_mode

说明

定义音频输入输出设备工作模式。

定义

```
typedef enum {
    HI_AIO_MODE_I2S_MASTER  = 0,    /* AIO I2S master mode */
    HI_AIO_MODE_I2S_SLAVE,          /* AIO I2S slave mode */
    HI_AIO_MODE_PCM_SLAVE_STD,      /* AIO PCM slave standard mode */
    HI_AIO_MODE_PCM_SLAVE_NON_STD,  /* AIO PCM slave non-standard mode */
    HI_AIO_MODE_PCM_MASTER_STD,     /* AIO PCM master standard mode */
```

```
    HI_AIO_MODE_PCM_MASTER_NON_STD, /* AIO PCM master non-standard mode */
    HI_AIO_MODE_BUTT
} hi_aio_mode;
```

成员

成员名称	描述
HI_AIO_MODE_I2S_MASTER	I2S(Integrated Interchip Sound)主模式。
HI_AIO_MODE_I2S_SLAVE	I2S从模式。
HI_AIO_MODE_PCM_SLAVE_STD	PCM(Pulse Code Modulation)从模式（标准协议）。
HI_AIO_MODE_PCM_SLAVE_NON_STD	PCM从模式（自定义协议）。
HI_AIO_MODE_PCM_MASTER_STD	PCM主模式（标准协议）。
HI_AIO_MODE_PCM_MASTER_NON_STD	PCM主模式（自定义协议）。
HI_AIO_MODE_BUTT	预留值。

11.18.20.2.4 hi_audio_bit_width

说明

定义音频采样位宽。

定义

```
typedef enum {
    HI_AUDIO_BIT_WIDTH_8   = 0,   /* 8bit width */
    HI_AUDIO_BIT_WIDTH_16  = 1,   /* 16bit width */
    HI_AUDIO_BIT_WIDTH_24  = 2,   /* 24bit width */
    HI_AUDIO_BIT_WIDTH_BUTT
} hi_audio_bit_width;
```

成员

成员名称	描述
HI_AUDIO_BIT_WIDTH_8	采样位宽为8bit。
HI_AUDIO_BIT_WIDTH_16	采样位宽为16bit。
HI_AUDIO_BIT_WIDTH_24	采样位宽为24bit。

11.18.20.2.5 hi_audio_snd_mode

说明

定义音频声道模式。

定义

```
typedef enum {
    HI_AUDIO_SOUND_MODE_MONO   = 0, /* mono */
    HI_AUDIO_SOUND_MODE_STEREO = 1, /* stereo */
    HI_AUDIO_SOUND_MODE_BUTT
} hi_audio_snd_mode;
```

成员

成员名称	描述
HI_AUDIO_SOUND_MODE_MONO	单声道。
HI_AUDIO_SOUND_MODE_STEREO	双声道。

11.18.20.2.6 hi_aio_i2s_type

说明

定义设备 I2S 对接设备类型。

定义

```
typedef enum {
    HI_AIO_I2STYPE_INNERCODEC = 0,
    HI_AIO_I2STYPE_INNERHDMI,
    HI_AIO_I2STYPE_EXTERN
} hi_aio_i2s_type;
```

成员

成员名称	描述
HI_AIO_I2STYPE_INNERCODEC	对接内置编解码器CODEC。
HI_AIO_I2STYPE_INNERHDMI	对接内置HDMI。
HI_AIO_I2STYPE_EXTERN	对接外接设备（例如编解码器）。

11.18.20.2.7 hi_aio_attr

说明

定义音频输入输出设备属性结构体。

定义

```
typedef struct {
    hi_audio_sample_rate sample_rate;
    hi_audio_bit_width   bit_width;
    hi_aio_mode          work_mode;
    hi_audio_snd_mode    snd_mode;
    hi_u32               expand_flag;

    hi_u32               frame_num;
    hi_u32               point_num_per_frame;
    hi_u32               chn_cnt;
    hi_u32               clk_share;
    hi_aio_i2s_type      i2s_type;
} hi_aio_attr;
```

成员

成员名称	描述
sample_rate	音频采样率（从模式下，此参数不起作用），仅支持48k。
bit_width	音频采样精度（从模式下，此参数必须和音频AD/DA的采样精度匹配），只支持16bit和24bit。
work_mode	音频输入输出工作模式。 仅支持0（master模式），如果配置其它值，系统自动设置为0。
snd_mode	音频声道模式，0表示单声道，1表示立体声。
expand_flag	音频位宽扩展标识。 预留参数，设置无效，配置其它值，系统自动设置为0。
frame_num	缓存帧数目。取值范围：[2, 300]。
point_num_per_frame	每帧的采样点个数。AI取值范围为：[480, 2048]，AO取值范围为：[480, 4096]。
chn_cnt	每路I2S支持的通道数目，最大支持2个通道。双声道时固定为2，单声道时固定为1。
clk_share	配置AI设备0是否复用AO设备0的帧同步时钟及位流时钟。 预留参数，设置无效，配置其它值，系统自动设置为1。

成员名称	描述
i2s_type	配置设备I2S类型。 仅支持0（对接内置编解码器CODEC），如果配置其它值，系统自动设置为0。

11.18.20.2.8 hi_audio_frame

说明

定义音频帧结构体。

定义

```
typedef struct {
    hi_audio_bit_width   bit_width;     /* audio frame bit_width */
    hi_audio_snd_mode    snd_mode;      /* audio frame momo or stereo mode */
    hi_u8 ATTRIBUTE *virt_addr[HI_AUDIO_FRAME_CHN_NUM];
    hi_u64 ATTRIBUTE phys_addr[HI_AUDIO_FRAME_CHN_NUM];
    hi_u64  time_stamp;                 /* audio frame time stamp */
    hi_u32  seq;                        /* audio frame seq */
    hi_u32  len;                        /* data length per channel in frame */
    hi_u32  pool_id[HI_AUDIO_FRAME_CHN_NUM];
} hi_audio_frame;
```

成员

成员名称	描述
bit_width	音频采样精度。
snd_mode	音频声道模式。
virt_addr	音频帧数据虚拟地址。
phys_addr	音频帧数据物理地址。
time_stamp	音频帧时间戳（单位：μs）。
seq	音频帧序号。
len	单个声道音频帧长度（单位：byte）。
pool_id	音频帧缓存池ID。

说明

len（音频帧长度）指单个声道的数据长度。

单声道数据存放的虚拟地址为 virt_addr [0]，长度为 len；立体声数据按左右声道分开存放，虚拟地址 virt_addr [0]存放长度为 len 的左声道数据，虚拟地址virt_addr [1]存放长度为 len 的右声道数据。

11.18.20.2.9 hi_audio_stream

说明

定义音频码流结构体。

定义

```
typedef struct {
    hi_u8 ATTRIBUTE *stream;    /* the virtual address of stream */
    hi_u64 ATTRIBUTE phys_addr;  /* the physics address of stream */
    hi_u32 len;                 /* stream length, by bytes */
    hi_u64 time_stamp;          /* frame time stamp */
    hi_u32 seq;                 /* frame seq, if stream is not a valid frame,seq is 0 */
} hi_audio_stream;
```

成员

成员名称	描述
stream	音频码流数据指针。
phys_addr	音频码流的物理地址。
len	音频码流长度（单位：byte）。 ● 调用hi_mpi_adec_create_chn接口设置pack方式解码，必须为一帧音频码流的长度（指码流数据+头长度），且一帧码流最大长度为16384。 ● 调用hi_mpi_adec_create_chn接口设置stream方式解码： – 调用hi_mpi_adec_send_stream接口发送码流时，设置非阻塞方式时，len参数取值范围：(0, 16384]。 – 调用hi_mpi_adec_send_stream接口发送码流时，设置阻塞方式时，len参数取值范围在hi_u32数据类型的取值范围内，但不包括0。
time_stamp	音频码流时间戳。
seq	音频码流序号，如果stream无效，seq为0。

11.18.20.2.10 hi_aec_frame

说明

定义音频回声抵消参考帧信息结构体。

定义

```
typedef struct {
    hi_audio_frame ref_frame;      /* AEC reference audio frame */
    hi_bool     valid;             /* whether frame is valid */
    hi_bool     sys_bind;          /* whether is sysbind */
} hi_aec_frame;
```

成员

成员名称	描述
ref_frame	回声抵消参考帧结构体。
valid	参考帧有效的标志。 HI_TRUE：参考帧有效。 HI_FALSE：参考帧无效，无效时不能使用此参考帧进行回声抵消。
sys_bind	AI和AENC是否系统绑定。

说明

目前不支持回声抵消功能，该结构体设置为空即可。

11.18.20.2.11 hi_audio_frame_info

说明

定义解码后的音频帧信息结构体。

定义

```
typedef struct {
    hi_audio_frame *frame;          /* frame pointer */
    hi_u32          id;             /* frame id */
} hi_audio_frame_info;
```

成员名称	描述
frame	音频帧指针。
id	音频帧的索引。

11.18.20.2.12 hi_acodec_volume_ctrl

说明

定义内置 Audio Codec 音量控制结构体。

定义

```
typedef struct {
    /* volume control, 0x00~0x7e, 0x7F:mute */
    hi_u32 volume_ctrl;
    /* adc/dac mute control, 1:mute, 0:unmute */
    hi_u32 volume_ctrl_mute;
} hi_acodec_volume_ctrl;
```

成员

成员名称	描述
volume_ctrl	音量大小。 取值范围：0~0x7F，在0x7F时为静音。
volume_ctrl_mute	静音控制。 取值范围： ● 0：不静音 ● 1：静音

11.18.20.2.13 音量调节宏

宏定义	取值
HI_ACODEC_SET_DACL_VOLUME	0xc008410d
HI_ACODEC_SET_DACR_VOLUME	0xc008410e
HI_ACODEC_GET_DACL_VOLUME	0xc0084111
HI_ACODEC_GET_DACR_VOLUME	0xc0084112
HI_ACODEC_SET_ADCL_VOLUME	0xc008410f
HI_ACODEC_SET_ADCR_VOLUME	0xc0084110
HI_ACODEC_GET_ADCL_VOLUME	0xc0084113
HI_ACODEC_GET_ADCR_VOLUME	0xc0084114

11.18.20.2.14 hi_aenc_chn_attr

说明

定义音频编码通道属性结构体。

定义

```
typedef struct {
    hi_payload_type type;
    hi_u32 point_num_per_frame;
    hi_u32 buf_size;
```

```
    hi_void ATTRIBUTE *value;
} hi_aenc_chn_attr;
```

成员

成员名称	描述
type	音频编码协议类型，目前支持HI_PT_G711A、HI_PT_G711U、HI_PT_AAC。 静态属性。
point_num_per_frame	音频编码协议对应的帧长（编码时收到的音频帧长小于等于该帧长都可以进行编码）。 G.711a/G.711u编码时支持80/160/240/320/480，AAC编码只支持1024。
buf_size	音频编码缓存大小，以帧为单位。 G.711a/G.711u编码时，该参数取值范围：[2, 100]；AAC编码时，该参数取值范围：[10, 80]。 静态属性。
value	具体协议属性指针。 G.711a/G.711u编码时，该参数值当前无效，非空指针即可；AAC编码时，需传入**hi_aenc_attr_aac**结构体变量。 静态属性。

11.18.20.2.15 hi_adec_mode

说明

定义解码方式。

定义

```
typedef enum {
    HI_ADEC_MODE_PACK = 0,
    HI_ADEC_MODE_STREAM,
    HI_ADEC_MODE_BUTT
} hi_adec_mode;
```

成员名称	描述
HI_ADEC_MODE_PACK	pack方式解码。
HI_ADEC_MODE_STREAM	stream方式解码。
HI_ADEC_MODE_BUTT	预留值。

11.18.20.2.16 hi_adec_chn_attr

说明

定义音频解码通道属性结构体。

定义

```
typedef struct {
    hi_payload_type type;
    hi_u32          buf_size;
    hi_adec_mode    mode;
    hi_void ATTRIBUTE *value;
} hi_adec_chn_attr;
```

成员名称	描述
type	音频解码协议类型，当前仅支持HI_PT_G711A、HI_PT_G711U、HI_PT_AAC。
buf_size	音频解码缓存大小, 以帧为单位。 G.711a/G.711u解码时，该参数取值范围：[2, 100]；AAC解码时，该参数取值范围：[10, 80]。
mode	解码方式。 G.711a/G.711u解码时，支持stream和pack解码方式；AAC解码时，仅支持stream模式。
value	具体协议属性指针。 G.711a/G.711u解码时该参数值当前无效，非空指针即可；AAC解码时，需传入**hi_adec_attr_aac**结构体变量。 静态属性。

11.18.20.2.17 hi_adec_attr_g711

说明

定义G.711解码协议属性结构体。预留。

定义

```
typedef struct {
    hi_u32 reserved;
}hi_adec_attr_g711;
```

成员

成员名称	描述
reserved	待扩展用（目前暂未使用）。

11.18.20.2.18 hi_ai_chn_mode

说明

定义AI通道的工作模式。

定义

```
typedef enum {
    HI_AI_CHN_MODE_NORMAL = 0,
    HI_AI_CHN_MODE_FAST,
    HI_AI_CHN_MODE_BUTT
} hi_ai_chn_mode;
```

成员名称	描述
HI_AI_CHN_MODE_NORMAL	标准模式。
HI_AI_CHN_MODE_FAST	快速模式。
HI_AI_CHN_MODE_BUTT	预留值。

11.18.20.2.19 hi_ai_chn_attr

说明

定义AI通道属性结构体。

定义

```
typedef struct {
    hi_ai_chn_mode mode;
} hi_ai_chn_attr;
```

成员名称	描述
mode	工作模式。

11.18.20.2.20 hi_aac_type

说明

AAC编解码协议类型。

定义

```
typedef enum {
    HI_AAC_TYPE_AACLC       = 0,
    HI_AAC_TYPE_EAAC        = 1,
    HI_AAC_TYPE_EAACPLUS    = 2,
    HI_AAC_TYPE_AACLD       = 3,
    HI_AAC_TYPE_AACELD      = 4,
    HI_AAC_TYPE_BUTT,
} hi_aac_type;
```

成员名称	描述
HI_AAC_TYPE_AACLC	AACLC格式，当前只支持单声道，数据位宽16bit，采样率48k的AAC编解码。
HI_AAC_TYPE_EAAC	eAAC格式（也称为HEAAC、AAC+或 aacPlusV1）。当前不支持。
HI_AAC_TYPE_EAACPLUS	eAACPLUS格式（也称为AAC++或 aacPlusV2）。当前不支持。
HI_AAC_TYPE_AACLD	ACLD格式。当前不支持。
HI_AAC_TYPE_AACELD	AACELD格式。当前不支持。
HI_AAC_TYPE_BUTT	预留值。

11.18.20.2.21 hi_aac_bps

说明

AAC编解码协议码率。

定义

```
typedef enum {
    HI_AAC_BPS_8K      = 8000,
    HI_AAC_BPS_16K     = 16000,
    HI_AAC_BPS_22K     = 22000,
    HI_AAC_BPS_24K     = 24000,
    HI_AAC_BPS_32K     = 32000,
    HI_AAC_BPS_48K     = 48000,
    HI_AAC_BPS_64K     = 64000,
    HI_AAC_BPS_96K     = 96000,
    HI_AAC_BPS_128K    = 128000,
    HI_AAC_BPS_256K    = 256000,
    HI_AAC_BPS_320K    = 320000,
    HI_AAC_BPS_BUTT
} hi_aac_bps;
```

成员名称	描述
HI_AAC_BPS_8K	码率8kbit/s。当前不支持。
HI_AAC_BPS_16K	码率16kbit/s。当前不支持。
HI_AAC_BPS_22K	码率22kbit/s。当前不支持。
HI_AAC_BPS_24K	码率24kbit/s。当前不支持。
HI_AAC_BPS_32K	码率32kbit/s。当前不支持。
HI_AAC_BPS_48K	码率48kbit/s。
HI_AAC_BPS_64K	码率64kbit/s。
HI_AAC_BPS_96K	码率96kbit/s。

成员名称	描述
HI_AAC_BPS_128K	码率128kbit/s。
HI_AAC_BPS_256K	码率256kbit/s。
HI_AAC_BPS_320K	码率320kbit/s。当前不支持。
HI_AAC_BPS_BUTT	预留值。

11.18.20.2.22 hi_aac_transport_type

说明

AAC编解码协议传输封装类型。

定义

```
typedef enum {
    HI_AAC_TRANSPORT_TYPE_ADTS = 0,
    HI_AAC_TRANSPORT_TYPE_LOAS = 1,
    HI_AAC_TRANSPORT_TYPE_LATM_MCP1 = 2,
    HI_AAC_TRANSPORT_TYPE_BUTT
} hi_aac_transport_type;
```

成员名称	描述
AAC_TRANS_TYPE_ADTS	ADTS封装格式。当前仅支持AACLC。
AAC_TRANS_TYPE_LOAS	LOAS封装格式。当前不支持。
AAC_TRANS_TYPE_LATM_MCP1	LATM1封装格式。当前不支持。
HI_AAC_TRANSPORT_TYPE_BUTT	预留值。

11.18.20.2.23 hi_aenc_attr_aac

说明

AAC编码协议属性结构体。

定义

```
typedef struct {
    hi_aac_type      aac_type;
    hi_aac_bps       bit_rate;
    hi_audio_sample_rate sample_rate;
    hi_audio_bit_width   bit_width;
    hi_audio_snd_mode    snd_mode;
    hi_aac_transport_type   transport_type;
    hi_s16           band_width;
} hi_aenc_attr_aac;
```

成员

成员名称	描述
aac_type	AAC编解码协议类型。当前仅支持HI_AAC_TYPE_AACLC。
bit_rate	AAC编解码协议码率。目前由于采样率只支持48kHz，在单声道的情况下，支持的码率范围为48~256 kbps。
sample_rate	采样率。当前仅支持48kHz。
bit_width	位宽。当前仅支持16 bit。
snd_mode	声道。当前仅支持单声道。
transport_type	AAC编解码协议传输封装类型。当前仅支持HI_AAC_TRANSPORT_TYPE_ADTS类型。
band_width	频段范围，取值为0或[1000, 24000]。

11.18.20.2.24 hi_adec_attr_aac

说明

AAC解码协议属性结构体。

定义

```
typedef struct {
    hi_aac_transport_type    transport_type;
} hi_adec_attr_aac;
```

成员

成员名称	描述
transport_type	AAC编解码协议传输封装类型。当前仅支持HI_AAC_TRANSPORT_TYPE_ADTS类型。

11.18.20.3 ISP 系统控制及 3A 算法注册

11.18.20.3.1 hi_sensor_id

说明

定义Sensor ID。

定义

```
typedef hi_s32 hi_sensor_id;
```

11.18.20.3.2 hi_isp_sns_attr_info

说明

定义ISP Sensor属性。

定义

```
typedef struct {
    hi_sensor_id sensor_id;
} hi_isp_sns_attr_info;
```

成员

成员名称	描述
sensor_id	Sensor ID号。

11.18.20.3.3 hi_isp_cmos_sensor_image_mode

说明

定义sensor输出的宽高和帧率属性。

定义

```
typedef struct {
    hi_u16      width;
    hi_u16      height;
    hi_float    fps;
    hi_u8       sns_mode;
} hi_isp_cmos_sensor_image_mode;
```

成员

成员名称	描述
width	Sensor输出的宽度。
height	Sensor输出的高度。
fps	Sensor输出的帧率。
sns_mode	用于进行Sensor初始化序列的选择，在分辨率和帧率相同时，配置不同的sns_mode对应不同的初始化序列；其他情况，sns_mode默认为0。

11.18.20.3.4 hi_isp_cmos_alg_key

说明

定义 ISP 的各算法是否采用 cmos 中的默认配置的标志位。

定义

```
typedef union {
    hi_u64  key;
    struct {
        hi_u64  bit1_drc              : 1;   /* [0] */
        hi_u64  bit1_demosaic         : 1;   /* [1] */
        hi_u64  bit1_pregamma         : 1;   /* [2] */
        hi_u64  bit1_gamma            : 1;   /* [3] */
        hi_u64  bit1_sharpen          : 1;   /* [4] */
        hi_u64  bit1_edge_mark        : 1;   /* [5] */
        hi_u64  bit1_hlc              : 1;   /* [6] */
        hi_u64  bit1_ldci             : 1;   /* [7] */
        hi_u64  bit1_dpc              : 1;   /* [8] */
        hi_u64  bit1_lsc              : 1;   /* [9] */
        hi_u64  bit1_ge               : 1;   /* [10] */
        hi_u64  bit1_anti_false_color : 1;   /* [11] */
        hi_u64  bit1_bayer_nr         : 1;   /* [12] */
        hi_u64  bit1_detail           : 1;   /* [13] */
        hi_u64  bit1_ca               : 1;   /* [14] */
        hi_u64  bit1_expander         : 1;   /* [15] */
        hi_u64  bit1_clut             : 1;   /* [16] */
        hi_u64  bit1_wdr              : 1;   /* [17] */
        hi_u64  bit1_dehaze           : 1;   /* [18] */
        hi_u64  bit1_lcac             : 1;   /* [19] */
        hi_u64  bit1_acs              : 1;   /* [20] */
        hi_u64  bit44_rsv             : 43;  /* [21:63] */
    };
} hi_isp_cmos_alg_key;
```

成员

成员名称	描述
bit1_drc	drc 模块是否采用 cmos 默认配置的标志位。
bit1_demosaic	demosaic 模块是否采用 cmos 默认配置的标志位。
bit1_pregamma	pregamma 模块是否采用 cmos 默认配置的标志位。
bit1_gamma	gamma 模块是否采用 cmos 默认配置的标志位。
bit1_sharpen	sharpen 模块是否采用 cmos 默认配置的标志位。
bit1_edge_mark	edge mark模块是否采用cmos默认配置的标志位
bit1_hlc	hlc模块是否采用cmos默认配置的标志位。
bit1_ldci	ldci 模块是否采用 cmos 默认配置的标志位。
bit1_dpc	dpc 模块是否采用 cmos 默认配置的标志位。
bit1_lsc	lsc 模块是否采用 cmos 默认配置的标志位。
bit1_ge	ge 模块是否采用 cmos 默认配置的标志位。
bit1_anti_false_color	anti false color 模块是否采用 cmos 默认配置的标志位。
bit1_bayer_nr	bayer nr 模块是否采用 cmos 默认配置的标志位。
bit1_detail	detail enhance模块是否采用cmos默认配置的标志位
bit1_ca	ca 模块是否采用 cmos 默认配置的标志位。

成员名称	描述
bit1_expander	expander 模块是否采用 cmos 默认配置的标志位。 仅 sensor built-in 模式有效。
bit1_clut	clut 模块是否采用 cmos 默认配置的标志位。
bit1_wdr	wdr 模块是否采用 cmos 默认配置的标志位。
bit1_dehaze	dehaze 模块是否采用 cmos 默认配置的标志位。
bit1_lcac	Local cac 模块是否采用 cmos 默认配置的标志位。
bit1_acs	acs 模块是否采用 cmos 默认配置的标志位。

注意事项

如果ISP的某个算法模块要使用cmos中的配置，要将对应的标志位置为1，否则采用的是算法内部的默认配置。

11.18.20.3.5 hi_isp_cmos_noise_calibration

说明

定义噪声校正参数。

定义

```
typedef struct {
    hi_u16   calibration_lut_num;
    hi_float calibration_coef[HI_ISP_BAYER_CALIBTAION_MAX_NUM]
[HI_ISP_BAYER_CALIBRATION_PARA_NUM];
} hi_isp_cmos_noise_calibration;
```

成员

成员名称	描述
calibration_lut_num	标定序列数目。 取值范围：[0, 49]。
calibration_coef	噪声标定表。 HI_ISP_BAYER_CALIBTAION_MAX_NUM用于定义标定噪声模型参数的iso档位个数的最大值，当前为50。 HI_ISP_BAYER_CALIBRATION_PARA_NUM用于定义标定噪声模型参数的最大个数，当前为16。

11.18.20.3.6 hi_isp_cmos_sensor_max_resolution

说明

定义Sensor最大分辨率结构体。

定义

```
typedef struct {
    hi_u32    max_width;
    hi_u32    max_height;
} hi_isp_cmos_sensor_max_resolution;
```

成员

成员名称	描述
max_width	最大宽度。
max_height	最大高度。

11.18.20.3.7 hi_isp_cmos_sensor_mode

说明

定义sensor模式寄存器。

定义

```
typedef struct {
    hi_u32  sensor_id;
    hi_u8   sensor_mode;
} hi_isp_cmos_sensor_mode;
```

成员

成员名称	描述
sensor_id	Sensor ID。
sensor_mode	Sensor自定义工作模式，不同分辨率与帧率对应不同的工作模式。

11.18.20.3.8 hi_isp_cmos_wdr_switch_attr

说明

定义WDR切换属性。

定义

```
typedef struct {
    hi_u32    exp_ratio[HI_ISP_EXP_RATIO_NUM];
} hi_isp_cmos_wdr_switch_attr;
```

成员

成员名称	描述
exp_ratio	曝光比期望值

注意事项

在 cmos_get_isp_default 函数里根据 WDR 模式给 exp_ratio 赋默认值，所赋值要与 cmos_get_ae_default 中 AE 初始化的曝光比保持一致性：

- 若 ae_sns_dft->man_ratio_enable 为 TRUE，exp_ratio 与 ae_sns_dft->arr_ratio 的值相同；
- 若 ae_sns_dft->man_ratio_enable 为 FALSE，exp_ratio 为 0x40。

11.18.20.3.9 hi_isp_cmos_default

说明

定义ISP基础算法库的初始化参数结构体。

定义

```
typedefstruct {
    hi_isp_cmos_alg_key              key;
    const hi_isp_cmos_drc            *drc;
    const hi_isp_cmos_demosaic       *demosaic;
    const hi_isp_cmos_pregamma       *pregamma;
    const hi_isp_cmos_gamma          *gamma;
    const hi_isp_cmos_sharpen        *sharpen;
    const hi_isp_cmos_edgemark       *edge_mark;
    const hi_isp_cmos_hlc            *hlc;
    const hi_isp_cmos_ldci           *ldci;
    const hi_isp_cmos_dpc            *dpc;
    const hi_isp_cmos_lsc            *lsc;
    const hi_isp_cmos_ge             *ge;
    const hi_isp_cmos_afc            *anti_false_color;
    const hi_isp_cmos_bayernr        *bayer_nr;
    const hi_isp_cmos_ca             *ca;
    const hi_isp_expander_attr       *expander;
    const hi_isp_cmos_clut           *clut;
    const hi_isp_cmos_wdr            *wdr;
    const hi_isp_cmos_dehaze         *dehaze;
    const hi_isp_cmos_lcac           *lcac;
    const hi_isp_cmos_acs            *acs;
    hi_isp_cmos_noise_calibration    noise_calibration;
    hi_isp_cmos_sensor_max_resolution sensor_max_resolution;
    hi_isp_cmos_sensor_mode          sensor_mode;
    hi_isp_cmos_wdr_switch_attr      wdr_switch_attr;
} hi_isp_cmos_default;
```

成员

成员名称	描述
key	标识各算法是否采用 cmos 中默认配置的 key。

成员名称	描述
*drc	DRC结构体指针。 结构体说明请参见《**ISP图像调优指南**》。
*demosaic	Demosaic结构体指针。 结构体说明请参见《**ISP图像调优指南**》。
*pregamma	PreGamma结构体指针。 结构体说明请参见《**ISP图像调优指南**》。
*gamma	Gamma结构体指针。 结构体说明请参见《**ISP图像调优指南**》。
*sharpen	Sharpen结构体指针。 结构体说明请参见《**ISP图像调优指南**》。
*edge_mark	EdgeMark结构体指针。 结构体说明请参见《**ISP图像调优指南**》。
*hlc	HLC结构体指针。 结构体说明请参见《**ISP图像调优指南**》。
*ldci	LDCI结构体指针。 结构体说明请参见《**ISP图像调优指南**》。
*dpc	DPC结构体指针。 结构体说明请参见《**ISP图像调优指南**》。
*lsc	LSC结构体指针。 结构体说明请参见《**ISP图像调优指南**》。
*ge	GE模块结构体指针。 结构体说明请参见《**ISP图像调优指南**》。
*anti_false_color	AntiFalse结构体指针。 结构体说明请参见《**ISP图像调优指南**》。
*bayer_nr	BayerNR结构体指针。 结构体说明请参见《**ISP图像调优指南**》。
*ca	CA模块结构体指针。 结构体说明请参见《**ISP图像调优指南**》。
*expander	Expander结构体指针。仅Sensor built-in模式有效。 结构体说明请参见《**ISP图像调优指南**》。
*clut	Clut结构体指针。 结构体说明请参见《**ISP图像调优指南**》。
*wdr	WDR模式结构体指针。 结构体说明请参见《**ISP图像调优指南**》。

成员名称	描述
*dehaze	Dehaze结构体指针。 结构体说明请参见《ISP图像调优指南》。
*lcac	Local cac结构体指针。 结构体说明请参见《ISP图像调优指南》。
*acs	ACS结构体指针。 结构体说明请参见《ISP图像调优指南》。
noise_calibration	Noise校正结构体。
sensor_max_resolution	Sensor最大宽高结构体。
sensor_mode	Sensor模式结构体。
wdr_switch_attr	WDR切换属性。

注意事项

- ISP的各算法模块如果要采用cmos的默认配置，要在回调函数**pfn_cmos_get_isp_default**中将对应的标志位置1，并给该算法模块的cmos结构体指针赋值。如果cmos默认值配置的不合法，会导致算法初始化失败，同时run起来之后算法不能正常调节。

11.18.20.3.10 hi_isp_cmos_black_level

说明

定义sensor的黑电平结构体。

定义

```
typedef struct {
    hi_bool update;
    hi_u16  black_level[HI_ISP_BAYER_CHN_NUM];
} hi_isp_cmos_black_level;
```

成员

成员名称	描述
update	Sensor的黑电平是否会动态根据增益改变。 取值范围： ● HI_TRUE：是 ● HI_FALSE：否 如果sensor的黑电平不会动态根据增益改变，update配置为HI_FALSE 即可。

成员名称	描述
black_level	Sensor的黑电平数组，取值范围：[0, 4095]。 HI_ISP_BAYER_CHN_NUM用于定义Bayer数据的通道数目，当前为4。

11.18.20.3.11 hi_isp_sns_type

说明

定义Sensor与ISP的通信接口类型。

定义

```
typedef enum {
    HI_ISP_SNS_I2C_TYPE = 0,
    HI_ISP_SNS_SSP_TYPE,
    HI_ISP_SNS_TYPE_BUTT,
} hi_isp_sns_type;
```

成员

成员名称	描述
HI_ISP_SNS_I2C_TYPE	Sensor与ISP使用I2C接口通信。
HI_ISP_SNS_SSP_TYPE	Sensor与ISP使用SSP接口通信。
HI_ISP_SNS_TYPE_BUTT	预留。

11.18.20.3.12 hi_isp_sns_commbus

说明

定义与Sensor绑定的设备号信息。

定义

```
typedef union {
    hi_s8   i2c_dev;
    struct {
        hi_s8  bit4_ssp_dev       : 4;
        hi_s8  bit4_ssp_cs        : 4;
    } ssp_dev;
} hi_isp_sns_commbus;
```

成员

成员名称	描述
i2c_dev	Sensor绑定的I2C设备号。

成员名称	描述
bit4_ssp_dev	Sensor绑定的SPI设备号。
bit4_ssp_cs	Sensor绑定的SPI片选信号。

11.18.20.3.13 hi_isp_i2c_data

说明

定义I2C数据的参数。

定义

```
typedef struct {
    hi_bool update;        /* RW; Range: [0x0, 0x1]; Format:1.0;
                           HI_TRUE: The sensor registers are written,
                           HI_FALSE: The sensor registers are not written */
    hi_u8   delay_frm_num; /* RW; Number of delayed frames for the sensor register */
    hi_u8   int_pos;       /* RW;Position where the configuration of the sensor register takes effect */
    hi_u8   dev_addr;      /* RW;Sensor device address */
    hi_u32  reg_addr;      /* RW;Sensor register address */
    hi_u32  addr_byte_num; /* RW;Bit width of the sensor register address */
    hi_u32  data;          /* RW;Sensor register data */
    hi_u32  data_byte_num; /* RW;Bit width of sensor register data */
} hi_isp_i2c_data;
```

成员

成员名称	描述
update	数据是否会配置sensor寄存器。 HI_TRUE：数据会配置sensor寄存器； HI_FALSE：数据不会配置sensor寄存器。
delay_frame_num	Sensor寄存器延迟配置的帧数。此变量的目的是保证曝光时间和增益同时生效。

成员名称	描述
interrupt_pos	Sensor寄存器的配置生效的位置。 ● 设置为0x0时表示寄存器在超短帧起始中断配置生效，设置为0x 1时表示寄存器在超短帧结束中断配置生效。 ● 设置为0x10时表示寄存器在短帧起始中断配置生效，设置为0x11时表示寄存器在短帧结束中断配置生效。 ● 设置为0x20时表示c寄存器在中帧起始中断配置生效，设置为0x21时表示寄存器在中帧结束中断配置生效。 ● 设置为0x30时表示寄存器在长帧起始中断配置生效，设置为0x31时表示寄存器在长帧结束中断配置生效。
dev_addr	Sensor设备地址。
reg_addr	Sensor寄存器地址。
addr_byte_num	Sensor寄存器地址位宽。
data	Sensor寄存器数据。
data_byte_num	Sensor寄存器数据位宽。

11.18.20.3.14 hi_isp_ssp_data

说明

定义SSP数据的参数。

定义

```
typedef struct {
    hi_bool update;             /* RW; Range: [0x0, 0x1]; Format:1.0;
                                HI_TRUE: The sensor registers are written,
                                HI_FALSE: The sensor registers are not written */
    hi_u8   delay_frm_num;      /* RW; Number of delayed frames for the sensor register */
    hi_u8   int_pos;            /* RW;Position where the configuration of the sensor register takes effect */
    hi_u32  dev_addr;           /* RW;Sensor device address */
    hi_u32  dev_addr_byte_num;  /* RW;Bit width of the sensor device address */
    hi_u32  reg_addr;           /* RW;Sensor register address */
    hi_u32  reg_addr_byte_num;  /* RW;Bit width of the sensor register address */
    hi_u32  data;               /* RW;Sensor register data */
    hi_u32  data_byte_num;      /* RW;Bit width of sensor register data */
} hi_isp_ssp_data;
```

成员

成员名称	描述
update	数据是否会配置sensor寄存器。 HI_TRUE：数据会配置sensor寄存器； HI_FALSE：数据不会配置sensor寄存器。
delay_frame_num	sensor寄存器延迟配置的帧数。此变量的目的是保证曝光时间和增益同时生效。
interrupt_pos	Sensor寄存器的配置生效的位置。 ● 设置为0x0时表示寄存器在帧起始中断配置生效，设置为1时表示寄存器在AF中断配置生效。 ● 设置为0x10时表示寄存器在短帧起始中断配置生效，设置为0x11时表示寄存器在短帧结束中断配置生效。 ● 设置为0x20时表示寄存器在中帧起始中断配置生效，设置为0x21时表示寄存器在中帧结束中断配置生效。 ● 设置为0x30时表示寄存器在长帧起始中断配置生效，设置为0x31时表示寄存器在长帧结束中断配置生效。
dev_addr	Sensor设备地址。
dev_addr_byte_num	Sensor设备地址位宽。
reg_addr	Sensor寄存器地址。
reg_addr_byte_num	Sensor寄存器地址位宽。
data	Sensor寄存器数据。
data_byte_num	Sensor寄存器数据位宽。

11.18.20.3.15 hi_isp_sns_regs_info

说明

定义sensor的寄存器信息。

定义

```
typedef struct {
    hi_isp_sns_type sns_type;
```

```
    hi_u32  reg_num;
    hi_u8   cfg2_valid_delay_max;
    hi_u32  exp_distance[HI_ISP_WDR_MAX_FRAME_NUM - 1]
    hi_isp_sns_commbus  com_bus;
    union {
        hi_isp_i2c_data i2c_data[HI_ISP_MAX_SNS_REGS];
        hi_isp_ssp_data ssp_data[HI_ISP_MAX_SNS_REGS];
    };
    struct {
        hi_bool update;
        hi_u8   delay_frame_num;
        hi_u32  slave_vs_time;
        hi_u32  slave_bind_dev;
    } slv_sync;
    hi_bool config;
} hi_isp_sns_regs_info;
```

成员

成员名称	描述	
sns_type	Sensor与ISP的通信接口类型。	
reg_num	曝光结果写到sensor时需要配置的寄存器个数，不支持动态修改。	
cfg2_valid_delay_max	所有Sensor寄存器从配置到生效延迟的帧数的最大值，单位为帧，用于保证sensor寄存器和ISP寄存器的同步。一般情况下，cmos sensor的曝光时间寄存器的延迟最大，为1~2帧，因此配置一般为1或2。	
exp_distance	Sensor在wdr模式下曝光长帧与中帧的行差，中帧与短帧的行差，短帧与短短帧的行差。预留属性，暂不支持。	
com_bus	与Sensor绑定的设备号信息	
i2c_data	I2C数据参数。	
ssp_data	SSP数据参数。	
update	数据是否会配置sensor寄存器。 HI_TRUE：数据会配置sensor寄存器； HI_FALSE：数据不会配置sensor寄存器。	
delay_frame_num	sensor寄存器延迟配置的帧数。此变量的目的是保证曝光时间和增益同时生效。	
slave_vs_time	XVS信号周期，单位：sensor输入时钟周期。	
slave_bind_dev	Slave设备号与vi_pipe绑定关系。	

成员名称	描述
config	Sensor寄存器数据配置完成标志。 ● TD_TRUE：完成配置。 ● TD_FALSE：还未配置。

注意事项

无

相关数据类型及接口

hi_isp_sensor_exp_func

11.18.20.3.16 hi_isp_sensor_exp_func

说明

定义sensor回调函数结构体。

定义

```
typedef struct {
    hi_void (*pfn_cmos_sensor_init)(hi_vi_pipe vi_pipe);
    hi_void (*pfn_cmos_sensor_exit)(hi_vi_pipe vi_pipe);
    hi_void (*pfn_cmos_sensor_global_init)(hi_vi_pipe vi_pipe);
    hi_s32 (*pfn_cmos_set_image_mode)(hi_vi_pipe vi_pipe, hi_isp_cmos_sensor_image_mode
*sensor_image_mode);
    hi_s32 (*pfn_cmos_set_wdr_mode)(hi_vi_pipe vi_pipe, hi_u8 mode);
    hi_s32 (*pfn_cmos_get_isp_default)(hi_vi_pipe vi_pipe, hi_isp_cmos_default *def);
    hi_s32 (*pfn_cmos_get_isp_black_level)(hi_vi_pipe vi_pipe, hi_isp_cmos_black_level *black_level);
    hi_s32 (*pfn_cmos_get_sns_reg_info)(hi_vi_pipe vi_pipe, hi_isp_sns_regs_info *sns_regs_info);
    hi_void (*pfn_cmos_set_pixel_detect)(hi_vi_pipe vi_pipe, hi_bool enable);
    hi_void (*pfn_cmos_notify_event)(hi_vi_pipe vi_pipe, hi_u32 event_id, hi_void *value);
} hi_isp_sensor_exp_func;
```

成员

成员名称	描述
pfn_cmos_sensor_init	初始化sensor的回调函数指针。
pfn_cmos_sensor_exit	sensor的回调退出函数指针。
pfn_cmos_sensor_global_init	初始化全局变量的回调函数指针。
pfn_cmos_set_image_mode	设置分辨率和帧率切换的回调函数指针。 返回值0表示sensor模式发生改变，ISP会调用pfn_cmos_sensor_init 重新配置sensor； 返回值-2表示sensor模式没有变化，ISP不会重新配置sensor。
pfn_cmos_set_wdr_mode	设置wdr模式的回调函数指针。

成员名称	描述
pfn_cmos_get_isp_default	获取ISP基础算法的初始值的回调函数指针。
pfn_cmos_get_isp_black_level	获取sensor的黑电平值的回调函数指针，支持根据sensor增益动态调整黑电平值。若此处动态调整黑电平值，则外部只能通过接口hi_mpi_isp_set_black_level_attr的手动模式设置黑电平。
pfn_cmos_get_sns_reg_info	获取sensor寄存器信息的回调函数指针，用于实现内核态配置AE信息。
pfn_cmos_set_pixel_detect	设置坏点校正开关的回调函数指针。
pfn_cmos_notify_event	通知模组驱动特殊事件，如休眠后唤醒事件，便于模组做部分逻辑处理。 ● event_id：必须为0，只支持1个事件，即休眠唤醒事件。 ● value：必须为null，预留字段。

注意事项

- pfn_cmos_sensor_init, pfn_cmos_get_isp_default, pfn_cmos_get_isp_black_level, pfn_cmos_set_pixel_detect 和 pfn_cmos_get_sns_reg_info 必须赋值，其他回调函数指针如果不需要赋值，应置为 NULL。例如有的 sensor 不支持切换分辨率， 那么 pfn_cmos_set_image_mode 需要置为 NULL。
- 不支持切换AWB增益配置位置。

11.18.20.3.17 hi_isp_sensor_register

说明

定义sensor注册结构体。

定义

```
typedef struct {
    hi_isp_sensor_exp_func    sns_exp;
} hi_isp_sensor_register;
```

成员

成员名称	描述
sns_exp	Sensor注册的回调函数结构体。

11.18.20.3.18 hi_isp_bayer_format

说明

定义输入Bayer图像数据格式。

定义

```
typedef enum {
    HI_ISP_BAYER_RGGB    = 0,
    HI_ISP_BAYER_GRBG    = 1,
    HI_ISP_BAYER_GBRG    = 2,
    HI_ISP_BAYER_BGGR    = 3,
    HI_ISP_BAYER_BUTT
} hi_isp_bayer_format;
```

成员

成员名称	描述
HI_ISP_BAYER_RGGB	RGGB排列方式。
HI_ISP_BAYER_GRBG	GRGB排列方式。
HI_ISP_BAYER_GBRG	GBRG排列方式。
HI_ISP_BAYER_BGGR	BGGR排列方式。

11.18.20.3.19 hi_isp_pub_attr

说明

定义 ISP 公共属性。

定义

```
typedef struct {
    hi_rect            wnd_rect;
    hi_size            sns_size;
    hi_float           frame_rate;
    hi_isp_bayer_format        bayer_format;
    hi_wdr_mode                wdr_mode;
    hi_u8              sns_mode;
} hi_isp_pub_attr;
```

成员

成员名称	描述
wnd_rect	裁剪窗口起始位置和图像宽高，必须设置为与vi pipe属性中的size保持一致，且不能比所绑定的dev属性中的宽高大，如果配置不符，会导致出图异常。 ● wnd_rect中设置的水平方向起始位置与图像宽度之和应小于sensor输出的图像宽度。 ● wnd_rect中设置的垂直方向起始位置与图像高度之和应小于sensor输出的图像高度，由于无法检测sensor实际输出的宽高，当不满足该条件时会返回不报错。
sns_size	Sensor输出的图像宽高。 宽或高的取值范围：[120, 16384]。 设置sns_size时，需与对应的sensor采集属性值保持一致。
frame_rate	输入图像帧率，取值范围为(0.00, 65535.00]。
bayer_format	Bayer数据格式。
wdr_mode	WDR模式选择。
sns_mode	用于进行Sensor初始化序列的选择，在分辨率和帧率相同时，配置不同的sns_mode对应不同的初始化序列；其他情况，sns_mode默认配置为0，可通过sns_size和frame_rate进行初始化序列的选择。

11.18.20.3.20 hi_isp_frame_info

说明

定义 ISP 实时信息。

定义

```
typedef struct {
    hi_u32      iso;
    hi_u32      exposure_time;
    hi_u32      isp_dgain;
    hi_u32      again;
    hi_u32      dgain;
    hi_u32      ratio[3];
    hi_u32      isp_nr_strength;
    hi_u32      f_number;
    hi_u32      sensor_id;
    hi_u32      sensor_mode;
    hi_u32      hmax_times;
    hi_u32      vmax;
    hi_u32      vc_num;
    hi_u64      fsync_timestamp;
    hi_u64      fs_timestamp;
    hi_u64      fe_timestamp;
    hi_u64      pe_timestamp;
    hi_s64      motion_filter[9];
    hi_u32      reserved[10];
} hi_isp_frame_info;
```

成员

成员名称	描述
iso	当前sensor模拟增益*sensor数字增益*ISP数字增益*100。仅作PQ调测参考使用。
exposure_time	曝光时间，单位是微秒（us）。仅作PQ调测参考使用。
isp_dgain	ISP数字增益。仅作PQ调测参考使用。
again	Sensor的模拟增益。仅作PQ调测参考使用。
dgain	Sensor的数字增益。仅作PQ调测参考使用。
ratio	多帧合成WDR相邻2帧默认曝光比。仅作PQ调测参考使用。
isp_nr_strength	ISP的NR强度。当前未支持，默认值为0。
f_number	当前使用的镜头的F值。仅作PQ调测参考使用。
sensor_id	当前使用的sensorID。仅作PQ调测参考使用。
sensor_mode	当前使用的sensor序列模式。仅作PQ调测参考使用。
hmax_times	当前使用的sensor对应读出一行的时间，单位是纳秒（ns）。仅作PQ调测参考使用。
vmax	sensor每帧实际生效的总行数，单位是行。仅作PQ调测参考使用。
vc_num	当前未支持，默认值为0。当前未支持，默认值为0。
fsync_timestamp	发出fsync曝光同步信号的时间。
fs_timestamp	收到sensor模组FrameStart信号的时间，仅做参考，会受CPU负载以及调度而存在误差。
fe_timestamp	收到sensor模组FrameEnd信号的时间，仅做参考，会受CPU负载以及调度而存在误差。
pe_timestamp	ISP处理完成后输出图像的时间，仅做维测参考使用。

成员名称	描述
motion_filter	数字防抖的3*3仿射变换矩阵，15bit 小数精度。该值无需用户设置，是开启数字防抖后自动生成的值。 用户不能直接使用该矩阵作为仿射变化矩阵，完成以下步骤后可得到仿射变换矩阵： （1）转换为浮点类型做归一化处理 $m = float([x_1, x_2, x_3, x_4, x_5 x_6, x_7, x_8, x_9])$ $m_1 = \begin{bmatrix} \frac{x_1}{x_9} & \frac{x_2}{x_9} & \frac{x_3}{x_9} \\ \frac{x_4}{x_9} & \frac{x_5}{x_9} & \frac{x_6}{x_9} \\ \frac{x_7}{x_9} & \frac{x_8}{x_9} & \frac{x_9}{x_9} \end{bmatrix}$ （2）求m1的逆矩阵，最终得到m2矩阵，m2矩阵可与图像直接进行运算 $m_2 = inv(m_1)$ （3）m2矩阵与输入图像进行相乘可得到校正后的图像 $img_{dst} = m_2 \otimes img_{src}$
reserved	预留字段，默认输出为0。

11.18.20.3.21 hi_slave_dev

说明

定义从信号设备号。

定义

```
typedef hi_s32 hi_slave_dev;
```

11.18.20.3.22 hi_isp_slave_sns_sync

说明

定义从模式 sensor 同步信号配置。

定义

```
typedef struct {
    union {
        struct {
            hi_u32    bit16_reserved : 16;
            hi_u32    bit_h_inv : 1;
            hi_u32    bit_v_inv : 1;
            hi_u32    bit12_reserved : 12;
            hi_u32    bit_h_enable : 1;
            hi_u32    bit_v_enable : 1;
        } bits;
        hi_u32 bytes;
    } cfg;
    hi_u32    vs_time;
    hi_u32    hs_time;
    hi_u32    vs_cyc;
    hi_u32    hs_cyc;
    hi_u32    hs_dly_cyc;
    hi_u32    slave_mode_time;
} hi_isp_slave_sns_sync;
```

成员

成员名称	描述
bit16_reserved	保留字段。
bit_h_inv	XHS极性配置。 ● 0：正极； ● 1 ： 负极。
bit_v_inv	XVS极性配置。 ● 0：正极； ● 1 ： 负极。
bit12_reserved	保留字段。
bit_h_enable	XHS输出使能。
bit_v_enable	XVS输出使能。
vs_time	XVS信号周期，单位：sensor输入时钟周期。
hs_time	XHS信号周期，单位：sensor输入时钟周期。
vs_cyc	XVS有效电平宽度，单位：sensor输入时钟周期。
hs_cyc	XHS有效电平宽度，单位：sensor 输入时钟周期。
hs_dly_cyc	XHS脉冲输出相对XVS脉冲的延迟周期配置，单位：sensor输入时钟周期。
slave_mode_time	Sensor从模式时序配置选择寄存器： ● 0：选择SENSOR0 timing配置； ● 1：选择SENSOR1 timing配置； ● 2：选择SENSOR2 timing配置； ● 3：选择SENSOR3 timing配置。

注意事项

如图 2-4~图 2-6 所示，说明了同步信号发生模块配置参数的含义。

图2-4 同步信号配置时序图

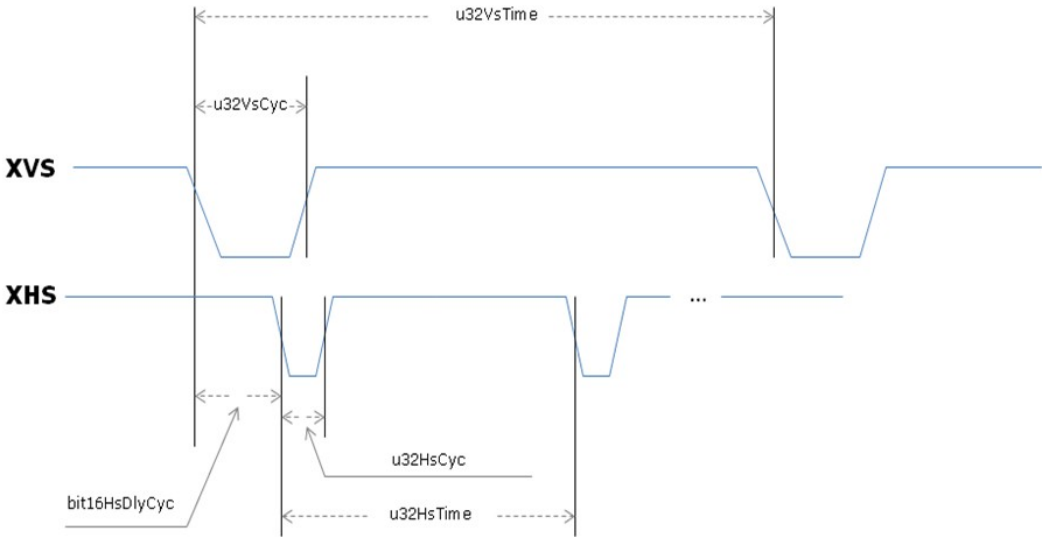


图2-5 同步信号极性翻转

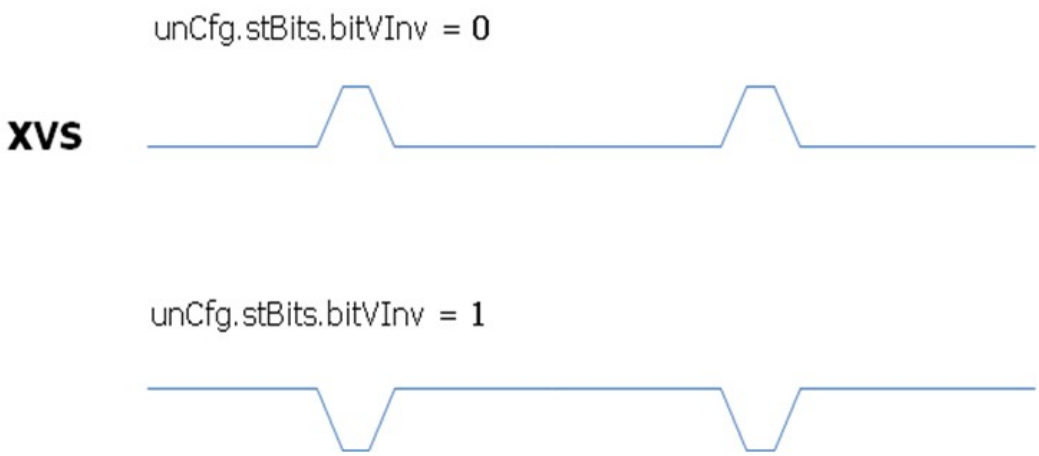
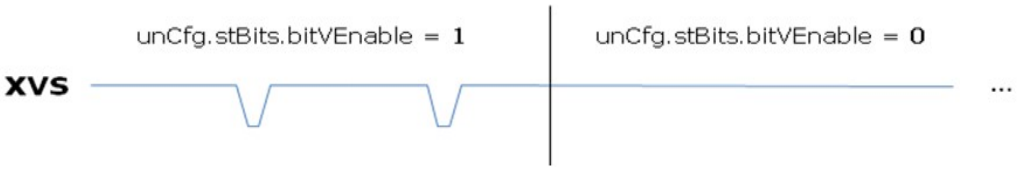


图2-6 同步信号使能



11.18.20.3.23 hi_isp_3a_alg_lib

说明

定义AE/AWB算法库结构体。

定义

```
typedef struct {
    hi_s32    id;
    hi_char lib_name[HI_ISP_ALG_LIB_NAME_SIZE_MAX];
} hi_isp_3a_alg_lib;
```

成员

成员名称	描述
id	算法库实例的Id，用以支持运行同一个算法库的多个实例
lib_name	标识算法库名称的字符数组，用以区分不同的算法库。 HI_ISP_ALG_LIB_NAME_SIZE_MAX用于定义3A算法库名称的最大字符数，表示20。

11.18.20.3.24 hi_isp_ae_accuracy_type

说明

定义曝光时间、增益的精度类型的枚举。

定义

```
typedef enum {
    HI_ISP_AE_ACCURACY_DB = 0,
    HI_ISP_AE_ACCURACY_LINEAR,
    HI_ISP_AE_ACCURACY_TABLE,
    HI_ISP_AE_ACCURACY_BUTT,
} hi_isp_ae_accuracy_type;
```

成员

成员名称	描述
HI_ISP_AE_ACCURACY_DB	DB精度类型。 DB精度指的是增益的倍数以倍增的形式增加。例如，sensor支持的again 为0db，0.3db，0.6db……这种情况下，精度类型可以设定为HI_ISP_AE_ACCURACY_DB，精度值设置为0.3，在这种精度下，again为 80，则表示80*0.3db=24db，24db是16倍增益；再例如，sensor支持的again为1倍，2倍，4倍，8倍……这种情况下，对应为db则为0db，6db，12db，18db，所以精度类型设定为HI_ISP_AE_ACCURACY_DB，精度值设置为6。由于AE算法内部实现线性转DB时计算精度存在误差，当DB精度较高(小于1)时，建议采用表格精度来实现。

成员名称	描述
HI_ISP_AE_ACCURACY_LINEAR	线性精度类型。 线性精度指的是增益的倍数均匀增加。例如，sensor能支持的again为 1 倍，1(1+1/16)倍，1(1+2/16)倍……这种情况下，精度类型可以设定为HI_ISP_AE_ACCURACY_LINEAR，精度值设置为0.0625，在这种精度下，again为32则表示32*0.0625= 2倍增益。
HI_ISP_AE_ACCURACY_TABLE	表格类型。 表格精度指增益的倍数是通过查表的形式获得，表格统一使用10bit精度，即1024表示1倍增益。当某些sensor的增益值的增加规律不线性或者DB精度较高时，可以使用表格精度方式，AE算法计算出需要的模拟增益/数字增益的数值，用查询sensor增益表格中最接近的值作为数字增益/模拟增益的值。 使用该模式时，需要相应的初始化回调结构体hi_isp_ae_sensor_exp_func 中的回调函数。

11.18.20.3.25 hi_isp_ae_accuracy

说明

定义曝光时间、增益的精度的结构体。

定义

```
typedef struct {
    hi_isp_ae_accuracy_type accu_type;
    float accuracy;
    float offset;
} hi_isp_ae_accuracy;
```

成员

成员名称	描述
accu_type	精度类型。
accuracy	精度值。
offset	曝光时间的偏移量，支持正负偏移量设置，以行为单位，该值配置与sensor强相关。

注意事项

- 曝光时间的精度通常都是线性的，都是以行为单位。若曝光时间为线性精度，精度accuracy为大于1的整数，则意味着AE计算出来的曝光行数只能为最小曝光时间加上accuracy的整数倍。该功能可用于计算某些sensor WDR模式下的长/ 短帧曝

光时间，如某些sensor曝光行数必须为2n+1，则可以配置最小曝光时间为1或3，配置accuracy为2。

- 曝光时间与sensor增益保持线性关系是AE曝光量分配的前提，即认为曝光量一定的情况下，曝光时间和sensor增益可以相互转换而保持图像亮度基本不变。比如说曝光量为4096，那么分配为曝光时间(2)*增益(2048)，也可以分配为曝光时间(4)*增益(1024)，这两种情况下图像亮度应该基本不变。若不满足这种线性关系，在高亮环境下，由于曝光时间很短，1行曝光时间的变化也容易使画面发生闪烁。某些sensor，如Panasoni MN34220采用1080p@30fps的初始化序列配置时，曝光时间必须偏移0.8018 行之后才与sensor增益有线性关系，因此增加了offset这个变量，它为float型变量，以行为单位，将offset设置成 0.8018， AE 算法内部即可完成偏移处理。一般sensor的曝光时间和增益不存在这种偏移关系，需在cmos.c中将该值配为0。

11.18.20.3.26 hi_isp_ae_sensor_default

说明

定义AE算法库的初始化参数结构体。

定义

```
typedef struct {
    hi_u8   hist_thresh[HI_ISP_HIST_THRESH_NUM];
    hi_u8 ae_compensation;
    hi_u32  lines_per500ms;
    hi_u32  flicker_freq;
    hi_float fps;
    hi_u32  hmax_times;
    hi_u32  init_exposure;
    hi_u32  init_ae_speed;
    hi_u32  init_ae_tolerance;
    hi_u32  full_lines_std;
    hi_u32  full_lines_max;
    hi_u32  full_lines;
    hi_u32  binning_full_lines;
    hi_u32  max_int_time;
    hi_u32  min_int_time;
    hi_u32  max_int_time_target;
    hi_u32  min_int_time_target;
    hi_isp_ae_accuracy int_time_accu;
    hi_u32  max_again;
    hi_u32  min_again;
    hi_u32  max_again_target;
    hi_u32  min_again_target;
    hi_isp_ae_accuracy again_accu;
    hi_u32  max_dgain;
    hi_u32  min_dgain;
    hi_u32  max_dgain_target;
    hi_u32  min_dgain_target;
    hi_isp_ae_accuracy dgain_accu;
    hi_u32  max_isp_dgain_target;
    hi_u32  min_isp_dgain_target;
    hi_u32  isp_dgain_shift;
    hi_u32  max_int_time_step;
    hi_bool  max_time_step_enable;
    hi_u32  max_inc_time_step[HI_ISP_WDR_MAX_FRAME_NUM];
    hi_u32  max_dec_time_step[HI_ISP_WDR_MAX_FRAME_NUM];
    hi_u32  lf_max_short_time;
    hi_u32  lf_min_exposure;
    hi_isp_ae_route ae_route_attr;
    hi_bool ae_route_ex_valid;
    hi_isp_ae_route_ex ae_route_attr_ex;
```

```
    hi_isp_ae_route ae_route_sf_attr;
    hi_isp_ae_route_ex ae_route_sf_attr_ex;
    hi_u16 man_ratio_enable;
    hi_u32 arr_ratio[HI_ISP_EXP_RATIO_NUM];
    hi_isp_iris_type  iris_type;
    hi_isp_piris_attr piris_attr;
    hi_isp_iris_f_no  max_iris_fno;
    hi_isp_iris_f_no  min_iris_fno;
    hi_isp_ae_strategy ae_exp_mode;
    hi_u16 iso_cal_coef;
    hi_u8  ae_run_interval;
    hi_u32 exp_ratio_max;
    hi_u32 exp_ratio_min;
    hi_bool diff_gain_support;
    hi_isp_quick_start_param quick_start;
    hi_isp_prior_frame prior_frame;
    hi_bool ae_gain_sep_cfg;
    hi_bool lhcg_support;
    hi_u32 sns_lhcg_exp_ratio;
} hi_isp_ae_sensor_default;
```

成员

成员名称	描述
hist_thresh	五段直方图的分割门限值数组，数组内各元素的取值范围为[0,0xff]。推荐使用默认值，线性模式为{0xd,0x28,0x60,0x80}，BUILT_IN WDR模式为{0x20,0x40,0x60,0x80}，帧合成WDR模式为{0xc,0x18,0x60,0x80}。 `#define HI_ISP_HIST_THRESH_NUM 4`
ae_compensation	AE 亮度目标值，取值范围为[0,255]，建议用0x38~0x40。
lines_per500ms	每 500ms 的总行数。
flicker_freq	抗闪频率，数值为当前电源频率的 256 倍。
fps	基准帧率。
hmax_times	Sensor 读出一行的时间，单位：ns。
init_exposure	默认初始曝光量，等于曝光时间与曝光增益的乘积，其中曝光时间的单位为微秒(us)。AE 算法采用该值作为初始 5 帧的曝光量，可用于运动 DV 加速启动。 建议关注快速启动的产品形态根据常用场景配置一个合适的初始曝光量，以达到 AE 快速收敛。线性/WDR 模式切换时，也会采用该值作为切换后第一帧的曝光量。合理修改 sensor 初始化序列的曝光时间和增益，将曝光时间(行数)*增益(6bit 小数精度)计算得到曝光量，并赋值给该变量，可使得切换更加平滑。若不设置，该值默认为 0。
init_ae_speed	默认初始 AE 调节速度，海思 AE 算法采用该值作为初始 100 帧的调节速度，可用于运动 DV 加速启动。 建议关注快速启动的产品形态可将该值配置为128。若不设置，该值默认为0。AE算法内部会将该值钳位至[64,255]。

成员名称	描述
init_ae_tolerance	默认初始 AE 曝光容忍偏差，海思 AE 算法采用该值作为初始 100 帧的曝光容忍偏差值，可用于设置得到首次AE收敛稳定标志的亮度范围。AE 统计亮度第一次落在 [目标亮度- init_ae_tolerance，目标亮度+init_ae_tolerance]范围内时，得到首次 AE 收敛稳定标志。若不设置，该值默认为 0。若在 cmos.c 不对该值赋值或赋值为 0，则 AE 算法内部将其置为 ae_compensation /10。AE 算法内部会将该值钳位至(0,255]。
full_lines_std	基准帧率时 1 帧的有效行数。
full_lines_max	sensor 降帧后 1 帧所能达到的最大行数，一般设为 sensor 所支持的最大行数。
full_lines	sensor 每帧实际生效的总行数。使用海思 AE 算法时，回调 cmos_fps_set 和 cmos_slow_framerate_set 时必须给该值赋值，用于返回每帧实际生效的总行数。
binning_full_lines	sensor 在 binning 模式下的等效曝光总行数，以行为单位。
max_int_time	最大曝光时间，以行为单位。
min_int_time	最小曝光时间，以行为单位。
max_int_time_target	最大曝光时间目标值，以行为单位。
min_int_time_target	最小曝光时间目标值，以行为单位。
int_time_accu	曝光时间精度。
max_again	最大模拟增益，以倍为单位。
min_again	最小模拟增益，以倍为单位。
max_again_target	最大模拟增益目标值，以倍为单位。
min_again_target	最小模拟增益目标值，以倍为单位。
again_accu	模拟增益精度。
max_dgain	最大数字增益，以倍为单位。
min_dgain	最小数字增益，以倍为单位。
max_dgain_target	最大数字增益目标值，以倍为单位。
min_dgain_target	最小数字增益目标值，以倍为单位。
dgain_accu	数字增益精度。
max_isp_dgain_target	最大 ISP 数字增益目标值，以倍为单位。
min_isp_dgain_target	最小 ISP 数字增益目标值，以倍为单位。
isp_dgain_shift	ISP 数字增益精度。

成员名称	描述
max_int_time_step	自动长帧模式下普通模式和长帧模式之前切换过程中，对短帧曝光时间减小的最大调整步长，以行数为单位。仅在自动长帧模式下有效。
max_time_step_enable	短帧曝光时间最大步长限定开关，该值为 HI_TRUE 时，短帧曝光时间最大步长限制功能生效。仅在 2 帧行合成 WDR 模式下生效。
max_inc_time_step	目前只开放 max_inc_time_step [0]，max_inc_time_step[0]为短帧曝光时间增加的最大步长，以行为单位。即下一帧的曝光行数相比当前帧增加的最大曝光行数不超过 max_inc_time_step [0]。仅在 2 帧行合成 WDR 模式下 max_time_step_enable 开关使能后生效。 HI_ISP_WDR_MAX_FRAME_NUM用于定义WDR合成的最大帧数，表示4。
max_dec_time_step	目前只开放 max_dec_time_step [0]，max_dec_time_step [0]为短帧曝光时间减少的最大步长，以行为单位。即下一帧的曝光行数相比当前帧减少的最大曝光行数不超过 max_dec_time_step [0]。仅在 2 帧行合成 WDR 模式下 max_time_step_enable 开关使能后生效。 HI_ISP_WDR_MAX_FRAME_NUM用于定义WDR合成的最大帧数，表示4。
lf_max_short_time	自动长帧模式下短帧最大曝光时间。
lf_min_exposure	自动长帧模式下强制输出长帧的最小曝光量。
ae_route_attr	AE 默认分配路线。
ae_route_ex_valid	AE 扩展分配路线是否生效开关，该值为 HI_TRUE 时使用扩展分配路线，否则使用普通 AE 分配路线。
ae_route_attr_ex	AE 默认扩展分配路线，合理配置可用于优化 WDR 模式图像效果。
ae_route_sf_attr	AE 短帧默认分配路线。
ae_route_sf_attr_ex	AE 短帧默认扩展分配路线，合理配置可用于优化 WDR 模式图像效果。
man_ratio_enable	多帧合成 WDR 手动曝光比使能，该值为 HI_TRUE 时，曝光比固定，不会根据场景自动更新。

成员名称	描述
arr_ratio	多帧合成WDR相邻2帧默认曝光比。当man_ratio_enable为HI_TRUE时，采用 arr_ratio[0]作为S/VS 默认曝光比，arr_ratio[1]作为M/S默认曝光比，arr_ratio[2]作为 L/M 默认曝光比。上述VS，S，M，L 分别表示极短帧，短帧，中帧和长帧的曝光时间，2 帧合成时只有VS和S；3帧合成时采用VS，S 和 M；4帧合成时采用VS，S，M 和 L。6bit 小数精度。 数组内各元素取值范围：[0x40, 0xFFF]。 HI_ISP_EXP_RATIO_NUM用于定义WDR曝光比的数目，表示3。
iris_type	默认镜头类型，DC-Iris 或 P-Iris。默认镜头类型与实际对接镜头类型不一致时，AI 算法无法正常工作。
piris_attr	P-Iris 参数，与具体镜头相关。只有默认镜头类型是 P-Iris 时，才需要用该结构体参数。
max_iris_fno	P-Iris 可用的最大 F 值，与具体使用镜头相关，使用 P-Iris 时必须设置，用于限制实际生效的 max_iris_fno_target 和 min_iris_fno_target，避免光圈目标值落到不合理的范围导致 AE 分配异常。 取值范围为[HI_ISP_IRIS_F_NO_32_0, HI_ISP_IRIS_F_NO_1_0]。
min_iris_fno	P-Iris 可用的最小 F 值，与具体使用镜头相关，使用 P-Iris 时必须设置，用于限制实际生效的 max_iris_fno_target 和 min_iris_fno_target，避免光圈目标值落到不合理的范围导致 AE 分配异常。 取值范围为[HI_ISP_IRIS_F_NO_32_0, HI_ISP_IRIS_F_NO_1_0]。
ae_exp_mode	默认曝光策略，高光优先或低光优先。建议 FSWDR 模式设置为低光优先，线性模式及 BuiltIn WDR 设置为高光优先。若不设置该值，默认为高光优先。
iso_cal_coef	ISO 标定系数，用于保证拍照所需 DCF 信息中显示的 ISO 是标准的，8bit 精度。 取值范围：[0x0, 0xFFFF]，默认值为 0x100。
ae_run_interval	默认 AE 算法运行间隔，以帧为单位。若不设置，则默认 AE 每帧执行一次。 取值范围：(0x0, 0xFF]

成员名称	描述
exp_ratio_max	默认最大曝光比，仅在多帧合成 WDR 模式下有效。 ● 当 man_ratio_enable 为 HI_FALSE 时，exp_ratio_max 表示最长帧与最短帧曝光时间比值的最大值。即 2 帧合成时表示 S/VS 曝光比的最大值，3 帧合成时表示 M/VS 曝光比的最大值，4 帧合成时表示 L/VS 曝光比的最大值。 ● 当 man_ratio_enable 为 HI_TRUE 时，exp_ratio_max 无效。 6bit 小数精度，0x40 表示曝光比为 1 倍。 取值范围：[0x40, 0x4000]
exp_ratio_min	默认最小曝光比，仅在多帧合成 WDR 模式下有效。 ● 当 man_ratio_enable 为 HI_FALSE 时，exp_ratio_min 表示长帧曝光时间与短帧曝光时间比值的最小值。 ● 当 man_ratio_enable 为 HI_TRUE 时，exp_ratio_min 无效。 6bit 小数精度，0x40 表示曝光比为 1 倍。 取值范围：[0x40, exp_ratio_max]
diff_gain_support	sensor 是否支持长短帧不同增益配置，该值为 HI_TRUE 时，sensor 支持长短帧不同 senor 模拟增益、sensor 数字增益配置，该值为 HI_FALSE 时，sensor 不支持长短帧不同 senor 模拟增益、sensor 数字增益配置。
quick_start	设置不带光敏快启相关参数。
prior_frame	WDR 模式下曝光路线生效的优先帧, 线性模式下此参数需配置为 LONG_FRAME。
ae_gain_sep_cfg	长短帧增益是否分开分配。
lhcg_support	Sensor 是否使用 LCG+HCG 模式，该值为 HI_TRUE 时， Sensor 使用 LCG+HCG 模式，该值为 HI_FALSE 时，Sensor 使用普通模式。
sns_lhcg_exp_ratio	LCG+HCG 模式的基础曝光比。

注意事项

- 线性/WDR 模式切换时，会回调 pfn_cmos_get_ae_default 函数更新 AE 相关默认参数。若 WDR 模式要使用 AE 扩展分配路线而线性模式不需要，建议在 cmos_get_ae_default 函数里面先对 AE 路线清零：ae_route_ex_valid= HI_FALSE，ae_route_attr. total_num= 0，ae_route_attr_ex. total_num= 0，然后视需要在 WDR 分支赋值。
- 海思 AE 算法采用 init_exposure 作为初始 5 帧的曝光量，可用于运动 DV 加速启动。建议关注快速启动的产品形态根据常用场景配置一个合适的初始曝光量，以达到 AE 快速收敛。FSWDR 模式该值对应的是长帧曝光量，FSWDR 模式若要快速启动，最好在 cmos.c 设置为固定曝光比，以减少曝光比调整的时间，待 AE 稳

定后再根据需要设置为自动曝光比。若在 cmos.c 未给 init_exposure 赋值或将 init_exposure 赋值为 0，则 AE 算法内部按起始曝光量为 1024 开始计算。

WDR 模式切换时，海思 AE 算法内部会计算切换曝光量以保证切换平滑，如果希望在 WDR 模式切换时init_exposure 生效，可以通过配置 init_ae_speed 为 0xFFFFFFFF 实现。

- 对于特定 Sensor，如 OV2718 DCG 模式，内部仅支持固定曝光比，需要配置 man_ratio_enable= HI_TRUE，并把 ratio 配置为 Sensor 支持的固定曝光比，如果使用自动曝光比或手动配置为不支持的曝光比，会使图像效果不正常。
- max_int_time_step 自动长帧模式下普通模式和长帧模式之前切换过程中, 对短帧曝光时间减小的最大调整步长，以行数为单位。仅在自动长帧模式下有效。此参数针对短帧曝光时间减小过快会出现坏帧的 sensor 开放，一般 sensor 可以设定为较大值，则可以受此参数限制。
- max_time_step_enable，max_inc_time_step [0]，max_dec_time_step [0]，这部分参数针对 2 合 1 行合成 WDR 模式下，短帧曝光时间前后 2 帧之间增加或者减少有限制的 sensor 开放。
- lf_max_short_time 为自动长帧模式下短帧曝光时间的最大值，如果此参数设置过小会导致自动长帧模式下亮区噪声表现变差。
- 对于支持 LCG+HCG 模式的 Sensor，如 OV2775，如果使用该模式，需正确配置基础曝光比 sns_lhcg_exp_ratio，并且必须配置 prior_frame=HI_ISP_LONG_FRAME，ae_gain_sep_cfg= HI_TRUE，否则会使图像效果不正常。

11.18.20.3.27 hi_isp_ae_sensor_exp_func

说明定义

sensor回调函数结构体。

定义

```
typedefstruct {
    hi_s32 (*pfn_cmos_get_ae_default)(hi_vi_pipe vi_pipe, hi_isp_ae_sensor_default*ae_sns_dft);
    hi_void (*pfn_cmos_fps_set)(hi_vi_pipe vi_pipe, hi_floatf32_fps, hi_isp_ae_sensor_default*ae_sns_dft);
    hi_void (*pfn_cmos_slow_framerate_set)(hi_vi_pipe vi_pipe, hi_u32full_lines,
hi_isp_ae_sensor_default*ae_sns_dft);
    hi_void (*pfn_cmos_inttime_update)(hi_vi_pipe vi_pipe, hi_u32int_time);
    hi_void (*pfn_cmos_gains_update)(hi_vi_pipe vi_pipe, hi_u32again, hi_u32dgain);
    hi_void (*pfn_cmos_again_calc_table)(hi_vi_pipe vi_pipe, hi_u32*again_lin, hi_u32*again_db);
    hi_void (*pfn_cmos_dgain_calc_table)(hi_vi_pipe vi_pipe, hi_u32*dgain_lin, hi_u32*dgain_db);
    hi_void (*pfn_cmos_get_inttime_max)(hi_vi_pipe vi_pipe, hi_u16man_ratio_enable,
    hi_u32*ratio, hi_u32*int_time_max, hi_u32*int_time_min, hi_u32*lf_max_int_time);
    hi_void (*pfn_cmos_ae_fswdr_attr_set)(hi_vi_pipe vi_pipe, hi_isp_ae_fswdr_attr*ae_fswdr_attr);
    hi_void (*pfn_cmos_ae_quick_start_status_set)(hi_vi_pipe vi_pipe, hi_boolquick_start_status);
    hi_void (*pfn_cmos_exp_param_convert)(hi_vi_pipe vi_pipe, hi_isp_ae_convert_param*exp_param);
} hi_isp_ae_sensor_exp_func;
```

成员

成员名称	描述
pfn_cmos_get_ae_default	获取AE算法库的初始值的回调函数指针。
pfn_cmos_fps_set	设置sensor的帧率。
pfn_cmos_slow_framerate_set	设置sensor的降帧。

成员名称	描述
pfn_cmos_inttime_update	设置sensor的曝光时间。
pfn_cmos_gains_update	设置sensor的模拟增益和数字增益。
pfn_cmos_again_calc_table	计算TABLE类型sensor模拟增益。
pfn_cmos_dgain_calc_table	计算TABLE类型sensor数字增益。
pfn_cmos_get_inttime_max	WDR模式下，计算短帧最大曝光时间的回调函数指针，与sensor强相关。
pfn_cmos_ae_fswdr_attr_set	2to1LineWDR模式下，设置长帧模式。
pfn_cmos_ae_quick_start_status_set	设置AE无光敏快启收敛状态。
pfn_cmos_exp_param_convert	完成不同帧率等曝光量转换曝光参数属性，配套hi_mpi_isp_get_exp_convert提供。

注意事项

- 如果回调函数指针不需要赋值，需要置为NULL。
- 在hi_isp_ae_sensor_default中定义了曝光时间和增益的精度，pfn_cmos_inttime_update和pfn_cmos_gains_update中设置的曝光时间和增益都是带精度的值，如何转换成sensor的配置值与sensor强相关，请参阅sensor手册。
- 不使用无光敏快启功能时，pfn_cmos_ae_quick_start_status_set需要设置为NULL。
- quick_start_status是AE无光敏快启收敛状态的标志位。该值为HI_TRUE时，无光敏快启收敛完成。

11.18.20.3.28 hi_isp_ae_sensor_register

说明

定义sensor注册结构体。

定义

```
typedef struct {
    hi_isp_ae_sensor_exp_func sns_exp;
} hi_isp_ae_sensor_register;
```

成员

成员名称	描述
sns_exp	Sensor注册的回调函数结构体。

11.18.20.3.29 hi_isp_awb_agc_table

说明

定义饱和度初始化参数结构体。

定义

```
typedef struct {
    hi_bool valid;
    hi_u8  saturation[HI_ISP_AUTO_ISO_NUM];
} hi_isp_awb_agc_table;
```

成员

成员名称	描述
valid	该结构体的数据是否有效，取值范围为 0、1。
saturation	根据增益动态调节饱和度的插值数组，数组内元素的取值范围为[0, 255]。 saturation为非单调递减序列。 HI_ISP_AUTO_ISO_NUM用于定义ISO档位数，表示16。

11.18.20.3.30 hi_isp_awb_ccm_tab

说明

定义不同色温下自动颜色校正矩阵系数。

定义

```
typedef struct {
    hi_u16 color_temp;
    hi_u16 ccm[HI_ISP_CCM_MATRIX_SIZE];
} hi_isp_awb_ccm_tab;
```

成员

成员名称	描述
color_temp	当前配置的CCM对应的色温。 取值范围：[2000, 10000]
ccm	不同色温下的颜色校正矩阵，8bit小数精度。Bit[15]是符号位，0表示正数，1 表示负数，例如0x8010表示-16。 Bit[12],Bit[13],Bit[14]是无效的，即正数从0x0000到0x0FFF，负数从0x8000到0x8FFF，是CCM的合理参数。 取值范围：[0x0, 0xFFFF]。 HI_ISP_CCM_MATRIX_SIZE用于定义CCM矩阵参数个数，表示9。

11.18.20.3.31 hi_isp_awb_ccm

说明

定义CCM颜色校正矩阵属性。

定义

```
typedef struct {
    hi_u16  ccm_tab_num;
    hi_isp_awb_ccm_tab ccm_tab[HI_ISP_CCM_MATRIX_NUM];
} hi_isp_awb_ccm;
```

成员

成员名称	描述
ccm_tab_num	当前配置的 CCM 的组数。 取值范围：[3, 7]。
ccm_tab	不同色温下的颜色校正矩阵和对应的色温值。 HI_ISP_CCM_MATRIX_NUM用于定义昇腾AWB可配置的CCM矩阵组数，表示7。

注意事项

如果选择三组CCM，推荐高色温CCM在D50光源标定，中色温CCM在TL84光源标定，低色温CCM在A光源标定。

11.18.20.3.32 hi_isp_awb_sensor_default

说明

定义AWB算法库的初始化参数结构体。

定义

```
typedef struct {
    hi_u16  wb_ref_temp;
    hi_u16  gain_offset[HI_ISP_BAYER_CHN_NUM];
    hi_s32  wb_para[HI_ISP_AWB_CURVE_PARA_NUM];
    hi_u16  golden_rgain;
    hi_u16  golden_bgain;
    hi_u16  sample_rgain;
    hi_u16  sample_bgain;
    hi_isp_awb_agc_table agc_tbl;
    hi_isp_awb_ccm ccm;
    hi_u16    init_rgain;
    hi_u16    init_ggain;
    hi_u16    init_bgain;
    hi_u8     awb_run_interval;
    hi_u16    init_ccm[HI_ISP_CCM_MATRIX_SIZE];
} hi_isp_awb_sensor_default;
```

成员

成员名称	描述
wb_ref_temp	静态白平衡校正色温，取值范围为[0,0xFFFF]。
gain_offset	静态白平衡的R、Gr、Gb、B颜色通道的增益值数组，数组内元素的取值范围为[0, 0xFFFF]。 HI_ISP_BAYER_CHN_NUM用于定义Bayer数据的通道数目，表示4。
wb_para	校正工具给出的白平衡参数数组，取值范围为[0, 0xFFFFFFFF]。 HI_ISP_AWB_CURVE_PARA_NUM用于定义昇腾AWB标定的Planck 曲线参数个数，表示6。
golden_rgain	Golden样机在线标定得到的G/R值。
golden_bgain	Golden样机在线标定得到的G/B值。
sample_rgain	当前样机在线标定得到的G/R值。
sample_bgain	当前样机在线标定得到的G/B值。
agc_tbl	该结构体的数据是否有效，取值范围为[0,1]。
	根据增益动态调节饱和度的插值数组，取值范围为[0, 255]。
ccm	当前配置的CCM的组数。 取值范围：[3, 7]
	不同色温下的颜色校正矩阵和对应的色温值。
init_rgain	ISP启动时R通道白平衡增益初始值。
init_ggain	ISP启动时G通道白平衡增益初始值。
init_bgain	ISP启动时B通道白平衡增益初始值。
awb_run_interval	ISP启动时AWB工作频率。 取值范围：[0x1, 0xFF]
init_ccm	ISP启动时CCM初始值数组。 HI_ISP_CCM_MATRIX_SIZE用于定义CCM矩阵参数个数，表示9。

注意事项

参考色温即静态白平衡校正的环境色温，需要提供色度计测量的实际值。

11.18.20.3.33 hi_isp_awb_sensor_exp_func

说明

定义sensor回调函数结构体。

定义

```
typedef struct {
    hi_s32 (*pfn_cmos_get_awb_default)(hi_vi_pipe vi_pipe, hi_isp_awb_sensor_default *awb_sns_dft);
} hi_isp_awb_sensor_exp_func;
```

成员

成员名称	描述
vi_pipe	vi_pipe号。
pfn_cmos_get_awb_default	获取 AWB 算法库的初始值的回调函数指针。

11.18.20.3.34 hi_isp_awb_sensor_register

说明

定义 sensor 注册结构体。

定义

```
typedef struct {
    hi_isp_awb_sensor_exp_func sns_exp;
} hi_isp_awb_sensor_register;
```

成员

成员名称	描述
sns_exp	sensor 注册的回调函数结构体。

11.18.20.4 MIPI Rx ioctl 命令字参数

11.18.20.4.1 combo_dev_t

说明

定义MIPI Rx、SLVS设备号。

定义

```
typedef unsigned int combo_dev_t;
```

注意事项

- MIPI RX设备有效范围为[0, 4)。
- SLVS-EC设备有效范围为[0, 2)。

11.18.20.4.2 sns_rst_source_t

说明

Sensor的复位信号线编号，软件上称为Sensor的复位源。

定义

```
typedef unsigned int sns_rst_source_t;
```

注意事项

- 有效值范围为[0, 2)。
- 每条Sensor复位信号线可以接多个Sensor，用户需要根据板子的连线确认Sensor复位信号线编号。

11.18.20.4.3 sns_clk_source_t

说明

Sensor的时钟信号线编号，软件上称为Sensor的时钟源。

定义

```
typedef unsigned int sns_clk_source_t;
```

注意事项

- 有效值范围为[0, 4)。
- 每条Sensor时钟信号线可以接多个Sensor，用户需要根据板子的连线确认Sensor时钟信号线编号。

11.18.20.4.4 lane_divide_mode_t

说明

定义MIPI Rx的LANE分布模式。各模式的含义请参见**表11-35**。

定义

```
typedef enum {
    LANE_DIVIDE_MODE_0= 0,
    LANE_DIVIDE_MODE_1     = 1,
    LANE_DIVIDE_MODE_2     = 2,
    LANE_DIVIDE_MODE_3     = 3,
    LANE_DIVIDE_MODE_BUTT
} lane_divide_mode_t;
```

注意事项

只有MIPI Rx需要设置LANE的分布模式。

11.18.20.4.5 input_mode_t

说明

定义MIPI Rx输入接口类型。

定义

```
typedef enum {
    INPUT_MODE_MIPI = 0x0,    /* mipi */
    INPUT_MODE_SUBLVDS = 0x1, /* SUB_LVDS */
    INPUT_MODE_LVDS = 0x2,    /* LVDS */
    INPUT_MODE_HISPI = 0x3,   /* HISPI */
    INPUT_MODE_SLVS = 0x4,
    INPUT_MODE_BUTT
} input_mode_t;
```

11.18.20.4.6 img_rect_t

说明

定义MIPI的裁剪属性。

定义

```
typedef struct {
    int x;
    int y;
    unsigned int width;
    unsigned int height;
} img_rect_t;
```

成员

成员名称	描述
x	裁剪起始位置x。
y	裁剪起始位置y。 SLV-EC的裁剪Y坐标必须要大于等于Sensor输出有效行的行号。
width	裁剪宽度，单位：像素。
height	裁剪高度，单位：像素。

11.18.20.4.7 slvs_lane_rate_t

说明

定义SLVS LANE的输入速率。

定义

```
typedef enum {
    SLVS_LANE_RATE_LOW = 0,         /* 1152Mbps */
```

```
    SLVS_LANE_RATE_HIGH = 1,        /* 2304Mbps */
    SLVS_LANE_RATE_BUTT
} slvs_lane_rate_t;
```

11.18.20.4.8 slvs_err_check_mode_t

说明

SLVS的CRC（Cyclic Redundancy Check）和ECC（Error Control Coding）模式。

定义

```
typedef enum {
    SLVS_ERR_CHECK_MODE_NONE = 0x0,       /* disable ECC & CRC */
    SLVS_ERR_CHECK_MODE_CRC = 0x1,        /* enable   CRC */
    SLVS_ERR_CHECK_MODE_ECC_2BYTE = 0x2,  /* enable   2 Byte ECC */
    SLVS_ERR_CHECK_MODE_ECC_4BYTE = 0x3,  /* enable   4 Byte ECC */
    SLVS_ERR_CHECK_MODE_BUTT
} slvs_err_check_mode_t;
```

成员

成员名称	描述
SLVS_ERR_CHECK_MODE_NONE	ECC与CRC都不启用。
SLVS_ERR_CHECK_MODE_CRC	启用CRC。
SLVS_ERR_CHECK_MODE_ECC_2BYTE	启用2 Byte ECC。
SLVS_ERR_CHECK_MODE_ECC_4BYTE	启用4 Byte ECC。

注意事项

SLVS ECC与CRC功能是互斥的，不能同时打开。

11.18.20.4.9 data_type_t

说明

定义传输的数据类型。

定义

```
typedef enum {
    DATA_TYPE_RAW_8BIT = 0,
    DATA_TYPE_RAW_10BIT = 1,
    DATA_TYPE_RAW_12BIT = 2,
    DATA_TYPE_RAW_14BIT = 3,
    DATA_TYPE_YUV420_8BIT_NORMAL = 5,
    DATA_TYPE_YUV422_8BIT = 7,
    DATA_TYPE_YUV422_PACKED = 8
    DATA_TYPE_BUTT
} data_type_t;
```

成员

成员名称	描述
DATA_TYPE_RAW_8BIT	8BIT的RAW数据。
DATA_TYPE_RAW_10BIT	10BIT的RAW数据。
DATA_TYPE_RAW_12BIT	12BIT的RAW数据。
DATA_TYPE_RAW_14BIT	14BIT的RAW数据。
DATA_TYPE_YUV420_8BIT_NORMAL	8BIT的YUV420数据，NORMAL模式。
DATA_TYPE_YUV422_8BIT	8BIT的YUV422数据。
DATA_TYPE_YUV422_PACKED	YUV422数据在MIPI按16bit raw打包接收。

11.18.20.4.10 mipi_data_rate_t

说明

MIPI Rx输入速率。

定义

```
typedef enum
{
    MIPI_DATA_RATE_X1 = 0,        /* output 1 pixel per clock */
    MIPI_DATA_RATE_X2 = 1,        /* output 2 pixel per clock */
    MIPI_DATA_RATE_BUTT
} mipi_data_rate_t;
```

成员

成员名称	描述
MIPI_DATA_RATE_X1	1拍1像素。
MIPI_DATA_RATE_X2	1拍2像素。当前不支持该选项。

11.18.20.4.11 mipi_wdr_mode_t

说明

定义MIPI WDR模式。

定义

```
typedef enum {
    HI_MIPI_WDR_MODE_NONE = 0x0,
    HI_MIPI_WDR_MODE_VC = 0x1,  /* Virtual Channel */
    HI_MIPI_WDR_MODE_DT = 0x2,  /* Data Type */
```

```
    HI_MIPI_WDR_MODE_DOL = 0x3, /* DOL Mode */
    HI_MIPI_WDR_MODE_BUTT
} mipi_wdr_mode_t;
```

成员

成员名称	描述
HI_MIPI_WDR_MODE_NONE	线性模式。
HI_MIPI_WDR_MODE_VC	使用Packet header中的Virtual Channel区分长短曝光帧。
HI_MIPI_WDR_MODE_DT	使用Packet header中的自定义Data type区分长短曝光帧。
HI_MIPI_WDR_MODE_DOL	表示DOL模式WDR，使用Packet header之后的一个像素识别长短曝光帧

注意事项

需与当前对接的Sensor模组配置对齐，且需与后端VI、ISP的模式配置对齐。

11.18.20.4.12 slvs_wdr_mode_t

说明

定义SLVS的WDR模式。

定义

```
typedef enum {
    HI_SLVS_WDR_MODE_NONE      = 0x0,
    HI_SLVS_WDR_MODE_2F        = 0x1,
    HI_SLVS_WDR_MODE_DOL_2F  = 0x4
} slvs_wdr_mode_t;
```

成员

成员名称	描述
HI_SLVS_WDR_MODE_NONE	线性模式。
HI_SLVS_WDR_MODE_2F	2合1WDR。 SLVS接口不支持该选项。
HI_SLVS_WDR_MODE_DOL_2F	DOL模式2合1WDR。

注意事项

- Built-in WDR模式和帧合成WDR模式都需要配置为HI_SLVS_WDR_MODE_NONE。
- DOL WDR模式需要配置为HI_SLVS_WDR_MODE_DOL_2F。
- 需与当前对接的Sensor模组配置对齐，且需与后端VI、ISP的模式配置对齐。

11.18.20.4.13 lvds_wdr_mode_t

说明

LVDS的WDR模式。

定义

```
typedef enum {
    HI_LVDS_WDR_MODE_NONE     = 0x0,
    HI_LVDS_WDR_MODE_2F       = 0x1,
    HI_LVDS_WDR_MODE_DOL_2F  = 0x4
} lvds_wdr_mode_t;
```

成员

成员名称	描述
HI_LVDS_WDR_MODE_NONE	线性模式。
HI_LVDS_WDR_MODE_2F	2合1WDR。 SLVS接口不支持该选项。
HI_LVDS_WDR_MODE_DOL_2F	DOL模式2合1WDR。

注意事项

- Built-in WDR 模式和帧合成WDR 模式都需要配置为HI_LVDS_WDR_MODE_NONE。
- DOL WDR 模式需要配置为 HI_LVDS_WDR_MODE_DOL_2F。
- 需与当前对接的Sensor模组配置对齐，且需与后端VI、ISP的模式配置对齐。

11.18.20.4.14 mipi_dev_attr_t

说明

定义MIPI设备属性。

定义

```
typedef struct {
    data_type_t input_data_type;  /* data type: 8/10/12/14/16 bit */
    mipi_wdr_mode_t wdr_mode;     /* MIPI WDR mode */
    short lane_id[MIPI_LANE_NUM]; /* lane_id: -1 - disable */
```

```
    union {
        short data_type[WDR_VC_NUM]; /* attribute of MIPI WDR mode.
AUTO:mipi_wdr_mode_t:HI_MIPI_WDR_MODE_DT; */
    };
} mipi_dev_attr_t;
```

成员

成员名称	描述
input_data_type	传输的数据类型。
lane_id	发送端(Sensor)和接收端(MIPI Rx) Lane的对应关系，未使用的Lane设置为-1。 MIPI_LANE_NUM表示8。
wdr_mode	MIPI WDR模式。
data_type	当wdr_mode为 HI_MIPI_WDR_MODE_DT时，需要设置data_type，表示不同曝光长度数据对应的Data Type。 WDR_VC_NUM表示4。

11.18.20.4.15 lvds_sync_mode_t

说明

定义LVDS同步方式。

定义

```
typedef enum {
    LVDS_SYNC_MODE_SOF = 0, /* Sensor SOL, EOL, SOF, EOF */
    LVDS_SYNC_MODE_SAV,     /* SAV, EAV */
    LVDS_SYNC_MODE_BUTT
} lvds_sync_mode_t;
```

成员

成员名称	同步方式
LVDS_SYNC_MODE_SOF	SOF、EOF、SOL、EOL。
LVDS_SYNC_MODE_SAV	invalid SAV、invalid EAV、valid SAV、valid EAV。

11.18.20.4.16 lvds_bit_endian_t

说明

定义比特位大小端模式。

定义

```
typedef enum {
    LVDS_ENDIAN_LITTLE = 0x0,
    LVDS_ENDIAN_BIG = 0x1,
    LVDS_ENDIAN_BUTT
} lvds_bit_endian_t;
```

11.18.20.4.17 lvds_vsync_type_t

说明

定义LVDS VSync（垂直同步）类型。

定义

```
typedef enum {
    LVDS_VSYNC_NORMAL = 0x00,
    LVDS_VSYNC_SHARE = 0x01,
    LVDS_VSYNC_HCONNECT = 0x02,
    LVDS_VSYNC_BUTT
} lvds_vsync_type_t;
```

成员

成员名称	描述
LVDS_VSYNC_NORMAL	长短曝光帧有独立的SOF-EOF、SOL-EOL或者invalid SAV-invalid EAV, valid SAV-valid EAV 。
LVDS_VSYNC_SHARE	长短曝光帧共用一对SOF-EOF标识，短曝光的起始几行用固定值填充。
LVDS_VSYNC_HCONNECT	长短曝光帧共用一对SAV-EAV标识，长短曝光帧之间是固定周期的消隐。

参考信息

- LVDS_VSYNC_SHARE同步方式：

<table>
<tr><td>SOF</td><td>Long Exposure</td><td rowspan="2">EOL</td><td rowspan="2">Horizontal Blanking</td><td rowspan="2">SOL</td><td>Padding</td><td>EOL</td><td rowspan="2">Horizontal Blanking</td></tr>
<tr><td>SOL</td><td>Padding</td><td>Short Exposure</td><td>EOF</td></tr>
<tr><td>SOV</td><td>V.BLK</td><td>EOV</td><td>-</td><td>SOV</td><td>V.BLK</td><td>EOV</td><td>-</td></tr>
</table>

- LVDS_VSYNC_HCONNECT同步方式：

<table>
<tr><td rowspan="4">SAV</td><td rowspan="2">Long Exposure frame</td><td rowspan="4">Horizontal Blanking(fix period)</td><td>V.BLK</td><td rowspan="4">Horizontal Blanking(fix period)</td><td>V.BLK</td><td rowspan="4">EAV</td><td rowspan="4">Horizontal Blanking</td></tr>
<tr><td rowspan="2">Short ExposureFrame1</td><td rowspan="2">Short Exposure Frame2</td></tr>
<tr><td>V.BLK</td></tr>
<tr><td>V.BLK</td></tr>
<tr><td></td><td colspan="5">V.BLK</td><td>-</td></tr>
</table>

11.18.20.4.18 lvds_vsync_attr_t

说明

定义LVDS VSync（垂直同步）参数。

定义

```
typedef struct {
    lvds_vsync_type_t sync_type;
    unsigned short hblank1;
    unsigned short hblank2;
} lvds_vsync_attr_t;
```

成员

成员名称	描述
sync_type	长短曝光帧同步方式。
hblank1	当同步方式为LVDS_VSYNC_HCONNECT，需要配置，表示Hconnect得消隐区长度。
hblank2	当同步方式为LVDS_VSYNC_HCONNECT，需要配置，表示Hconnect得消隐区长度。

11.18.20.4.19 lvds_fid_type_t

说明

定义FID（Frame Identification Id）的类型。

定义

```
typedef enum {
    LVDS_FID_NONE = 0x00,
    LVDS_FID_IN_SAV = 0x01,  /* frame identification id in SAV 4th */
    LVDS_FID_IN_DATA = 0x02, /* frame identification id in first data */
    LVDS_FID_BUTT
} lvds_fid_type_t;
```

成员

成员名称	描述
LVDS_FID_NONE	不使用FID。
LVDS_FID_IN_SAV	FID插入在SAV第4个字段中，DOL 4个字段的同步码需要将lvds_fid_attr_t.fid_type配置为LVDS_FID_IN_SAV。
LVDS_FID_IN_DATA	FID作为Frame Information Column插入在同步码之后的第一个像素之前，DOL 5 个字段的同步码需要将lvds_fid_attr_t.fid_type配置为LVDS_FID_IN_DATA。

11.18.20.4.20 lvds_fid_attr_t

说明

定义FID（Frame Identification Id）属性的配置信息。

定义

```
typedef struct {
    lvds_fid_type_t fid_type;
    unsigned char output_fil;
} lvds_fid_attr_t;
```

成员

成员名称	描述
fid_type	LVDS DOL模式下的FID类型
output_fil	DOL模式中的Frame information line紧接在 V-Blanking 之后输出，Frame ID是Frame information line中的第一个像素值。 Frame information line中并不包含有效的视频数据： ● 如果output_fil设置为HI_TRUE，Frame information line会输出到后端设备。 ● 如果output_fil设置为HI_FALSE，MIPI Rx将丢弃这一行数据。

11.18.20.4.21 lvds_dev_attr_t

说明

定义LVDS/SubLVDS/HiSPi设备属性。

定义

```
typedef struct {
    data_type_t input_data_type; /* data type: 8/10/12/14 bit */
```

```
    lvds_wdr_mode_t wdr_mode;         /* WDR mode */
    lvds_sync_mode_t sync_mode;   /* sync mode: SOF, SAV */
    lvds_vsync_attr_t vsync_attr; /* normal, share, hconnect */
    lvds_fid_attr_t fid_attr;     /* frame identification code */
    lvds_bit_endian_t data_endian;      /* data endian: little/big */
    lvds_bit_endian_t sync_code_endian; /* sync code endian: little/big */
    short lane_id[LVDS_LANE_NUM];       /* lane_id: -1 - disable */
    /* each vc has 4 params, sync_code[i]:
       sync_mode is SYNC_MODE_SOF: SOF, EOF, SOL, EOL
       sync_mode is SYNC_MODE_SAV: invalid sav, invalid eav, valid sav, valid eav  */
    unsigned short sync_code[LVDS_LANE_NUM][WDR_VC_NUM][SYNC_CODE_NUM];
} lvds_dev_attr_t;
```

成员

成员名称	描述
input_data_type	传输的数据类型。
wdr_mode	WDR模式。
sync_mode	LVDS同步模式。
vsync_ attr	vsync类型，当wdr_mod为DOL模式并且sync_mode 为LVDS_SYNC_MODE_SAV 时，需要配置vsync的类型。
fid_attr	frame identification类型，当wdr_mode为DOL模式，并且sync_mode为 LVDS_SYNC_MODE_SAV时，需要配置。
data_endian	数据大小端模式。
sync_code_endian	同步码大小端模式。
lane_id	发送端(Sensor)和接收端(MIPI Rx) lane的对应关系，未使用的lane设置为-1。 LVDS_LANE_NUM表示8。
sync_code	同步码共4个，前3个同步码固定为全1、全0、全0，不需要配置，只需配置第 4 个同步码。每lane有16个配置项，共四行四列。每一行代表一个Virtual Channel，WDR模式时 4行的配置需要根据发送端配置不同的同步码，线性模式时4行应配置为相同的同步码。根据不同的同步模式，4列的同步码分别表示SOF/EOF/ SOL/EOL 的同步码或者invalid SAV/invalid EAV/valid SAV/valid EAV的同步码。 LVDS_LANE_NUM表示8。 WDR_VC_NUM表示4。 SYNC_CODE_NUM表示4。

注意事项

使用该结构体，LVDS的LANE同步码同步到达。

11.18.20.4.22 slvs_dev_attr_t

说明

定义SLVS设备属性。

定义

```
typedef struct {
    data_type_t          input_data_type;
    slvs_wdr_mode_t      wdr_mode;
    slvs_lane_rate_t     lane_rate;
    int                  sensor_valid_width;
    short                lane_id[SLVS_LANE_NUM];   /* lane_id: -1 - disable */
    slvs_err_check_mode_t err_check_mode;          /* ECC CRC mode */
} slvs_dev_attr_t;
```

成员

成员名称	描述
input_data_type	传输的数据类型。
wdr_mode	WDR模式。
lane_rate	SLVS Lane速率。
sensor_valid_width	一行数据包的像素个数，仅低14bit有效，即最大16383像素。
lane_id	发送端(Sensor)和接收端(SLVS) Lane的对应关系。未使用的Lane设置为-1。 SLVS_LANE_NUM表示8。
err_check_mode	SLVS的CRC、ECC 模式，必须与Sensor端匹配。

注意事项

- SLVS只能支持线性模式和2合1WDR模式。
- SLVS接口不受**HI_MIPI_SET_HS_MODE**的限制。
- SLVS 的设备0和1的Lane可以在lane0-lane7之间任意选择，两个设备的Lane不能重复。

11.18.20.4.23 combo_dev_attr_t

说明

combo设备属性，由于MIPI Rx能够对接CSI-2、LVDS、HiSPi等时序，所以将MIPI Rx称为combo设备。

定义

```
typedef struct {
    combo_dev_t devno;
    input_mode_t input_mode;
```

```
    mipi_data_rate_t date_rate;
    img_rect_t img_rect;
    union {
        mipi_dev_attr_t mipi_attr;
        lvds_dev_attr_t lvds_attr;
        slvs_dev_attr_t slvs_attr;
    };
} combo_dev_attr_t;
```

成员

成员名称	描述
devno	MIPI Rx，SLVS设备号。
input_mode	输入接口类型。
img_rect	图像裁剪区域。
date_rate	接口传输速率。当前版本只支持1拍1像素，所以必须设置为MIPI_DATA_RATE_X1。
mipi_attr	如果input_mode配置为INPUT_MODE_MIPI，则必须配置mipi_attr。
lvds_attr	如果input_mode配置为INPUT_MODE_SUBLVDS/ INPUT_MODE_LVDS/ INPUT_MODE_HISPI，则必须配置lvds_attr。
slvs_attr	如果input_mode配置为INPUT_MODE_SLVS，则必须配置slvs_attr。
lvds_attr_ex	如果input_mode配置为INPUT_MODE_LVDS_EX，则必须配置lvds_attr_ex。

11.18.20.4.24 sns_clk_freq_t

说明

Sensor从模式下的时钟输入频率。

定义

```
typedef enum {
    SENSOR_CLK_74P25MHz  = 0x00,
    SENSOR_CLK_72MHz     = 0x01,
    SENSOR_CLK_54MHz     = 0x02,
    SENSOR_CLK_50MHz     = 0x03,
    SENSOR_CLK_24MHz     = 0x04,
    SENSOR_CLK_37P125MHz = 0x05,
    SENSOR_CLK_36MHz     = 0x06,
    SENSOR_CLK_27MHz     = 0x07,
    SENSOR_CLK_25MHz     = 0x08,
    SENSOR_CLK_12MHz     = 0x09,
    SENSOR_CLK_FREQ_BUTT
} sns_clk_freq_t;
```

成员

成员名称	描述
SENSOR_CLK_74P25MHz	时钟频率为74.25MHz
SENSOR_CLK_72MHz	时钟频率为72MHz
SENSOR_CLK_54MHz	时钟频率为54MHz
SENSOR_CLK_50MHz	时钟频率为50MHz
SENSOR_CLK_24MHz	时钟频率为24MHz
SENSOR_CLK_37P125MHz	时钟频率为37.125MHz
SENSOR_CLK_36MHz	时钟频率为36MHz
SENSOR_CLK_27MHz	时钟频率为27MHz
SENSOR_CLK_25MHz	时钟频率为25MHz
SENSOR_CLK_12MHz	时钟频率为12MHz

11.18.20.4.25 sns_clk_cfg_t

说明

Sensor从模式下的Sensor时钟输入配置。

定义

```
typedef struct {
    sns_clk_source_t clk_source;
    sns_clk_freq_t clk_freq;
} sns_clk_cfg_t;
```

成员

成员名称	描述
clk_source	时钟信号源序号。
clk_freq	时钟频率。

11.18.20.5 VI 视频输入

11.18.20.5.1 hi_vi_dev

说明

定义VI设备号。

定义

```
typedef hi_s32 hi_vi_dev;
```

11.18.20.5.2 hi_vi_pipe

说明

定义VI PIPE号。

定义

```
typedef hi_s32 hi_vi_pipe;
```

11.18.20.5.3 hi_vi_chn

说明

定义VI通道号。

定义

```
typedef hi_s32 hi_vi_chn;
```

11.18.20.5.4 hi_vi_intf_mode

说明

定义视频设备的接口模式 。

定义

```
typedef enum {
    HI_VI_MODE_MIPI = 4,
    HI_VI_MODE_MIPI_YUV420_NORM,
    HI_VI_MODE_MIPI_YUV420_LEGACY,
    HI_VI_MODE_MIPI_YUV422,
    HI_VI_MODE_LVDS,
    HI_VI_MODE_HISPI,
    HI_VI_MODE_SLVS,
    HI_VI_MODE_BUTT
} hi_vi_intf_mode;
```

成员

表 11-58 视频设备接口模式

成名名称	描述
HI_VI_INTF_MODE_MIPI	输入数据符合MIPI协议，用于传输RAW数据。
HI_VI_INTF_MODE_MIPI_YUV420_NORM	输入数据符合MIPI协议，用于传输YUV420 normal模式的数据。
HI_VI_INTF_MODE_MIPI_YUV420_LEGACY	输入数据符合MIPI协议，用于传输YUV420 legacy模式的数据。当前版本不支持该模式。
HI_VI_INTF_MODE_MIPI_YUV422	输入数据符合MIPI协议，用于传输YUV422数据。
HI_VI_MODE_LVDS	输入数据符合LVDS协议。当前版本不支持该模式。
HI_VI_MODE_HISPI	输入数据符合HISPI协议。当前版本不支持该模式。
HI_VI_MODE_SLVS	输入数据符合SLVS-EC协议。
HI_VI_MODE_BUTT	预留值。

11.18.20.5.5 hi_data_rate

说明

定义设备的速率。

定义

```
typedef enum {
    HI_DATA_RATE_X1 = 0,
    HI_DATA_RATE_X2 = 1,
    HI_DATA_RATE_BUTT
} hi_data_rate;
```

成员

成员名称	描述
HI_DATA_RATE_X1	一拍一像素。
HI_DATA_RATE_X2	一拍两像素，当前版本不支持。
HI_DATA_RATE_BUTT	预留值。

11.18.20.5.6 hi_vi_scan_mode

说明

定义视频设备接收的是隔行或逐行输入图像。

定义

```
typedef enum {
    HI_VI_SCAN_PROGRESSIVE = 0,
    HI_VI_SCAN_INTERLACED,
    HI_VI_SCAN_BUTT
} hi_vi_scan_mode;
```

成员

表 11-59 视频设备接收扫描模式

成名名称	描述
HI_VI_SCAN_PROGRESSIVE	VI输入为逐行图像。
HI_VI_SCAN_INTERLACED	VI输入为隔行图像。
HI_VI_SCAN_BUTT	预留值。

11.18.20.5.7 hi_vi_data_seq

说明

定义视频设备接收的YUV数据的数据排列顺序 。

定义

```
typedef enum {
    HI_VI_DATA_SEQ_VUVU = 0,
    HI_VI_DATA_SEQ_UVUV,
    HI_VI_DATA_SEQ_UYVY,
    HI_VI_DATA_SEQ_VYUY,
    HI_VI_DATA_SEQ_YUYV,
    HI_VI_DATA_SEQ_YVYU,
    HI_VI_DATA_SEQ_BUTT
} hi_vi_data_seq;
```

成员

成员名称	描述
HI_VI_DATA_SEQ_VUVU	YUV数据通过分离模式输入时，C分量的输入排列顺序为VUVU。

成员名称	描述
HI_VI_DATA_SEQ_UVUV	YUV数据通过分离模式输入时，C分量的输入排列顺序为UVUV。
HI_VI_DATA_SEQ_UYVY	YUV数据通过复合模式输入时，顺序为UYVY。当前版本不支持。
HI_VI_DATA_SEQ_VYUY	YUV数据通过复合模式输入时，顺序为VYUY。当前版本不支持。
HI_VI_DATA_SEQ_YUYV	YUV数据通过复合模式输入时，顺序为YUYV。当前版本不支持。
HI_VI_DATA_SEQ_YVYU	YUV数据通过复合模式输入时，顺序为YVYU。当前版本不支持。
HI_VI_DATA_SEQ_BUTT	预留值。

11.18.20.5.8 hi_vi_data_type

说明

VI输入数据类型枚举。

定义

```
typedef enum {
    HI_VI_DATA_TYPE_RAW = 0,
    HI_VI_DATA_TYPE_YUV,
    HI_VI_DATA_TYPE_BUTT
} hi_vi_data_type;
```

成员

成员名称	描述
HI_VI_DATA_TYPE_RAW	输入数据类型为 RGB，VI 前端一般接的是Sensor。
HI_VI_DATA_TYPE_YUV	输入数据类型为 YUV，VI 前端一般接的是 AD。
HI_VI_DATA_TYPE_BUTT	预留值。

11.18.20.5.9 hi_vi_dev_attr

说明

定义视频输入设备的属性。

定义

```
typedef struct {
    hi_vi_intf_mode      intf_mode;
```

```
    hi_vi_scan_mode      scan_mode;
    hi_vi_data_seq       data_seq;
    hi_vi_data_type      data_type;
    hi_size              in_size;
    hi_vi_wdr_attr       wdr_attr;
    hi_data_rate         data_rate;
    hi_u32               reserved[30];
} hi_vi_dev_attr;
```

成员

成员名称	描述
intf_mode	接口模式。
scan_mode	输入扫描模式 (逐行、隔行)。**当前仅支持逐行。**
data_seq	输入YUV数据的排列顺序 (仅支持YUV格式，如VUVU或UVUV等)。 当data_type为HI_VI_DATA_TYPE_YUV时，需要配置data_seq。
data_type	输入数据类型，Sensor输入一般为Raw，部分内置ISP的Sensor输入也可以为YUV。
in_size	VI设备可设置要捕获图像的高宽。 捕获图像的最小宽高与最大宽高： 宽度：[120, 16384] 高度：[120, 16384]
data_rate	设备的速率，可选择是1拍1像素还是1拍2像素。**当前仅支持1拍1像素。**
wdr_addr	WDR属性。
reserved	预留参数，为保证后续版本兼容性，请务必使用memset结构体方式进行清零初始化，在代码中必须避免显式对reserved字段进行访问。

11.18.20.5.10 hi_vi_dev_bind_pipe

说明

定义VI DEV与PIPE 的绑定关系。

定义

```
typedef struct {
    hi_u32  num;
    hi_vi_pipe pipe_id[HI_VI_MAX_PHYS_PIPE_NUM];
} hi_vi_dev_bind_pipe;
```

成员

成员名称	描述
num	该VI Dev所绑定的PIPE数量，取值范围[1, 4]。
pipe_id	该VI Dev绑定的PIPE号。 HI_VI_MAX_PHYS_PIPE_NUM表示4。

11.18.20.5.11 hi_vi_wdr_attr

说明

定义VI WDR融合组的属性 。

定义

```
typedef struct {
    hi_wdr_mode wdr_mode;
    hi_u32 cache_line;
} hi_vi_wdr_attr;
```

成员

成员名称	描述
wdr_mode	WDR工作模式，分为帧模式、行模式、非WDR等三大类。
cache_line	预留参数。

说明

WDR模式配置需与Sensor模组配置、Mipi RX配置对齐，否则会导致出图异常。

11.18.20.5.12 hi_vi_pipe_attr

说明

定义VI PIPE属性。

定义

```
typedef struct {
    hi_vi_pipe_bypass_mode pipe_bypass_mode;
    hi_bool                isp_bypass;
    hi_size                size;
    hi_pixel_format        pixel_format;
    hi_compress_mode       compress_mode;
    hi_data_bit_width      bit_width;
    hi_frame_rate_ctrl     frame_rate_ctrl;
    hi_u32                 depth;
    hi_u32                 reserved[10];
} hi_vi_pipe_attr;
```

成员

成员名称	描述
pipe_bypass_mode	VI PIPE的bypass模式。静态属性，创建PIPE时设定，不可更改。
isp_bypass	ISP是否运行。静态属性，创建PIPE时设定，不可更改。 ● HI_FALSE：ISP正常运行。 ● HI_TRUE：ISP不运行。 当输入为Raw数据时，必须设置为HI_FALSE；当输入为YUV时则可以设置为HI_TRUE或者HI_FALSE。
size	输入图像宽高。需根据所绑定的VI设备属性的宽高或调用hi_mpi_vi_set_pipe_pre_crop接口配置的宽高，保证大小一致，需与ISP公共属性所设置的宽高需保持一致。静态属性，创建PIPE时设定，不可更改。 取值范围：[120, 16384]。
pixel_format	像素格式。静态属性，创建PIPE时设定，不可更改。 如果是YUV图像，则仅pipe2和pipe3支持，且仅支持输出HI_PIXEL_FORMAT_YVU_SEMIPLANAR_422、HI_PIXEL_FORMAT_YVU_SEMIPLANAR_420、HI_PIXEL_FORMAT_YUV_400。 如果是RAW图像，则仅支持位宽为8、10、12、14的RGB BAYER格式。
compress_mode	数据压缩格式。静态属性，创建PIPE时设定，不可更改。 ● 仅支持帧压缩HI_COMPRESS_MODE_FRAME或行压缩HI_COMPRESS_MODE_LINE，且开启帧压缩时，分辨率需满足：总像素个数不少于307200个。 ● 当输入数据格式为HI_PIXEL_FORMAT_RGB_BAYER_10BPP、HI_PIXEL_FORMAT_RGB_BAYER_12BPP、HI_PIXEL_FORMAT_RGB_BAYER_14BPP时，线性模式、WDR帧模式和行模式都支持HI_COMPRESS_MODE_LINE、HI_COMPRESS_MODE_FRAME；当输入数据格式为格式为HI_PIXEL_FORMAT_RGB_BAYER_8BPP时，不支持压缩；当输入数据格式为YUV时，不支持压缩。
bit_width	输入图像的bit位宽。仅当像素格式pixel_format为YUV像素格式时有效。静态属性，创建PIPE时设定，不可更改。
frame_rate_ctrl	帧率控制。
depth	PIPE队列深度，用于缓存VI PIPE的临时数据、RAW/YUV图输入和输出数据。静态属性，创建 PIPE 时设定，不可更改。 临时缓存实际所能使用的最大队列深度为：hi_vi_pipe_attr.depth - hi_vi_frame_dump_attr.depth 取值范围：[0, 16]。

成员名称	描述
reserved	预留参数，为保证后续版本兼容性，请务必使用memset结构体方式进行清零初始化，在代码中必须避免显式对reserved字段进行访问。

11.18.20.5.13 hi_vi_vpss_mode

说明

定义VI各个PIPE和VPSS各个组的工作模式。

定义

```
typedef struct {
    hi_vi_vpss_mode_type mode[VI_MAX_PIPE_NUM];
} hi_vi_vpss_mode;
```

成员

成员名称	描述
mode	VI各个PIPE和VPSS各个组的工作模式。 数组内每个元素对应每个VI PIPE，例如数组内index为0的元素对应PIPE 0。 `#define VI_MAX_PIPE_NUM  12`

11.18.20.5.14 hi_vi_vpss_mode_type

说明

定义VI PIPE和VPSS组的工作模式。

定义

```
typedef enum {
    HI_VI_OFFLINE_VPSS_OFFLINE = 0,
    HI_VI_OFFLINE_VPSS_ONLINE,
    HI_VI_ONLINE_VPSS_OFFLINE,
    HI_VI_ONLINE_VPSS_ONLINE,
    HI_VI_PARALLEL_VPSS_OFFLINE,
    HI_VI_PARALLEL_VPSS_PARALLEL,
    HI_VI_VPSS_MODE_BUTT
} hi_vi_vpss_mode_type;
```

成员

成员名称	描述
HI_VI_OFFLINE_VPSS_OFFLINE	VI离线，VPSS离线。 使用该模式时，如果需要让视频流自动流转到VPSS继续处理，则需要调用**hi_mpi_sys_bind**接口设定VI、VPSS绑定关系，并完成VI PIPE以及被绑VPSS Group的初始化。
HI_VI_OFFLINE_VPSS_ONLINE	VI离线，VPSS在线。 当使用该模式时，所有VI PIPE都需要使用该模式。该模式下，VI PIPE编号与VPSS GROUP号一一对应，数据从VI PIPE流动到VPSS GROUP，不需要通过**hi_mpi_sys_bind**接口设定VI、VPSS绑定关系，绑定关系内部自动维护为：VPSS GROUP ID = VI PIPE ID + 256。 该模式下，VI PIPE以及内部自动绑定的VPSS Group必须都完成初始化，否则没法正常出图。
HI_VI_ONLINE_VPSS_OFFLINE	VI在线，VPSS离线。当前版本不支持。
HI_VI_ONLINE_VPSS_ONLINE	VI在线，VPSS在线。当前版本不支持。
HI_VI_PARALLEL_VPSS_OFFLINE	VI并行，VPSS离线。当前版本不支持。
HI_VI_PARALLEL_VPSS_PARALLEL	VI并行，VPSS并行。当前版本不支持。

11.18.20.5.15 hi_vi_pipe_bypass_mode

说明

定义VI PIPE的bypass模式。

定义

```
typedef enum {
    HI_VI_PIPE_BYPASS_NONE,
    HI_VI_PIPE_BYPASS_FE,
    HI_VI_PIPE_BYPASS_BE,
    HI_VI_PIPE_BYPASS_BUTT
} hi_vi_pipe_bypass_mode;
```

成员

成员名称	描述
HI_VI_PIPE_BYPASS_NONE	VI 的数据经过FE与BE处理。
HI_VI_PIPE_BYPASS_FE	VI 的数据不经过FE处理，只经过BE处理。当前版本不支持。

成员名称	描述
HI_VI_PIPE_BYPASS_BE	VI 的数据经过FE处理，不经过BE处理。
HI_VI_PIPE_BYPASS_BUTT	预留值。

11.18.20.5.16 hi_vi_frame_dump_attr

说明

定义VI PIPE dump属性。

定义

```
typedef struct {
    hi_bool         enable;
    hi_u32          depth;
} hi_vi_frame_dump_attr;
```

成员

成员名称	描述
enable	是否开启dump。 ● HI_FALSE：不开启。 ● HI_TRUE：开启。
depth	Dump数据的队列深度，该值不允许超过对应PIPE属性里的depth值。 取值范围：[0, 8]。

11.18.20.5.17 hi_vi_pipe_frame_source

说明

定义视频输入设备的属性。

定义

```
typedef enum {
    HI_VI_PIPE_FRAME_SOURCE_FE = 0,
    HI_VI_PIPE_FRAME_SOURCE_USER,
    HI_VI_PIPE_FRAME_SOURCE_BUTT
} hi_vi_pipe_frame_source;
```

成员

成员名称	描述
HI_VI_PIPE_FRAME_SOURCE_FE	数据来自于前端FE，默认模式。

成员名称	描述
HI_VI_PIPE_FRAME_SOURCE_USER	数据来自用户从BE送进来的数据。
HI_VI_PIPE_FRAME_SOURCE_BUTT	预留值。

11.18.20.5.18 hi_vi_chn_attr

说明

定义VI通道属性。

定义

```
typedef struct {
    hi_size            size;
    hi_pixel_format    pixel_format;
    hi_dynamic_range   dynamic_range;
    hi_video_format    video_format;
    hi_compress_mode   compress_mode;
    hi_u32             depth;
    hi_frame_rate_ctrl frame_rate_ctrl;
    hi_u32             reserved[10];
} hi_vi_chn_attr;
```

成员

成员名称	描述
size	目标图像大小。静态属性，设置通道时设定，不可更改。 ● 必须配置，且大小需要在VI支持的范围，详细范围说明参考**11.18.9.2 约束说明**。 ● 通道不支持缩放，通道size必须和pipe属性配置的size一致 。
pixel_format	目标图像像素格式。静态属性，设置通道时设定，不可更改。 当未开启DIS或LDC时，支持的像素格式为：HI_PIXEL_FORMAT_YVU_SEMIPLANAR_422、HI_PIXEL_FORMAT_YVU_SEMIPLANAR_420、HI_PIXEL_FORMAT_YUV_400。 当开启DIS或LDC时，支持的像素格式为：HI_PIXEL_FORMAT_YVU_SEMIPLANAR_420、HI_PIXEL_FORMAT_YUV_400。
dynamic_range	目标图像动态范围。静态属性，设置通道时设定，不可更改。

成员名称	描述
video_format	目标图像视频数据格式。当前版本仅支持HI_VIDEO_FORMAT_LINEAR格式。
compress_mode	目标图像压缩格式。静态属性，设置通道时设定，不可更改。当前版本不支持。
depth	用户获取图像的队列深度。静态属性，设置通道时设定，不可更改。 取值范围：[0, 8]。
frame_rate_ctrl	帧率控制。
reserved	预留参数，为保证后续版本兼容性，请务必使用memset结构体方式进行清零初始化，在代码中必须避免显式对reserved字段进行访问。

说明

- 当前版本，VI CHN通道不支持缩放，所以请确保将chn属性中的size大小与pipe属性中的size大小保持一致。
- 请根据实际负载和抖动，设置合理的depth值，内部实际申请的内存块数量为：调用**hi_mpi_vi_enable_chn**接口开启通道时的hi_vi_pipe_attr属性中的depth - hi_vi_frame_dump_attr属性中的depth + chn属性中的depth值。所以建议用户在开启通道前，确保已完成pipe dump属性的相关配置。

11.18.20.5.19 hi_dis_config

说明

定义DIS的配置信息。

定义

```
typedef struct {
    hi_dis_mode         mode;
    hi_dis_motion_level motion_level;
    hi_dis_pdt_type     pdt_type;
    hi_u32              buf_num;
    hi_u32              crop_ratio;
    hi_u32              frame_rate;
    hi_bool camera_steady;
    hi_bool scale;
} hi_dis_config;
```

成员

成员名称	描述
mode	DIS中使用不同自由度防抖算法，共有3种不同算法。

成员名称	描述
motion_level	Camera的运动级别。
pdt_type	使用DIS的产品形态，当前支持IPC、DV和无人机。
buf_num	DIS用于缓存图像的内存块数目，在DIS输出帧率偶尔出现丢帧时，可以增加缓存buf数。 取值范围：[5, 10]。
crop_ratio	DIS输出图像的裁剪比例。 取值范围：[50, 98]。
frame_rate	设置为VI输出的实际帧率。取值范围：(0, 60]。 ● 当防抖模式为HI_DIS_MODE_4_DOF_GME/HI_DIS_MODE_6_DOF_GME时，低帧率会导致两帧间隔变大，两帧之间抖动幅度增加，同时，果冻效应（rolling shutter）也更加严重，预期可能会出现防抖效果下降，特别是在抖动幅度大的场景； ● 当防抖模式为HI_DIS_MODE_GYRO时， 除了防抖效果的影响和GME一致，会导致算法支持的陀螺仪的最高采样率下降。当前3.5KHz的规格是基于最小25fps设计，如果下降， 对应的支持的陀螺仪的最大采样率也会下降。
camera_steady	镜头是否固定静止的。 取值范围： ● HI_FALSE：不固定静止。 ● HI_TRUE：固定静止。
scale	Crop后的输出图像是否进行放大。当前DIS提供Crop 后的输出图像是否进行放大操作选择。如果用户不想采用在DIS放大功能，可以选择在后端的VPSS进行放大操作。 取值范围： ● HI_FALSE：不放大。 ● HI_TRUE：放大。 scale为HI_FALSE的情况下，输出的图像宽高为输入图像宽高乘以crop_ratio设置的裁剪比例，且值为2对齐。

11.18.20.5.20 hi_dis_attr

说明

定义DIS属性。

定义

```
typedef struct {
    hi_bool enable;
    hi_bool gdc_bypass;
    hi_u32  moving_subject_level;
    hi_s32  rolling_shutter_coef;
    hi_s32  timelag;
    hi_u32  view_angle;
    hi_u32  horizontal_limit;
    hi_u32  vertical_limit;
    hi_bool still_crop;
    hi_u32  strength;
} hi_dis_attr;
```

成员

成员名称	描述
enable	是否开启DIS功能。 ● HI_FALSE：不开启； ● HI_TRUE：开启。
gdc_bypass	DIS功能中是否开启bypass GDC的开关。 ● HI_FALSE：不开启bypass GDC功能，即正常调用GDC做PMF矫正； ● HI_TRUE：开启bypass GDC功能，即不调用GDC做PMF矫正。
moving_subject_level	用于判断物体是否是运动的级别，值越小，运动过程中越稳定，但更容易出现偏移情况。值较大时，防抖效果越弱，运动过程中背景抖动幅度大，但能够改善图像偏移现象。 取值范围：[0, 6]。
rolling_shutter_coef	校正果冻效应（rolling shutter）现象的校正参数。 取值范围：[0, 1000]。
timelag	帧起始时间和陀螺仪数据采集时间的时间差，单位为微秒。 取值范围：[-2000000, 2000000]。
view_angle	预留参数。

成员名称	描述
horizontal_limit	水平偏移限制。当大面积物体经过引起背景拖拽的水平偏移超过一定幅度时就不进行防抖。偏移幅度计算：2047*horizontal_limit/1000。 取值范围：[0,1000]。
vertical_limit	垂直偏移限制。当大面积物体经过引起背景拖拽的垂直偏移超过一定幅度时就不进行防抖。偏移幅度计算 ： 2047* vertical_limit/1000。 取值范围：[0,1000]。
still_crop	关闭DIS防抖效果，但图像依旧按照所配置的比例裁剪。
strength	陀螺仪防抖的强度控制。仅对GYRO模式有效，对GME模式无效，默认强度为1024。 取值范围：[0,1024]。

11.18.20.5.21 hi_dis_mode

说明

定义DIS防抖算法模式。

定义

```
typedef enum {
    HI_DIS_MODE_4_DOF_GME = 0,
    HI_DIS_MODE_6_DOF_GME,
    HI_DIS_MODE_GYRO,
    HI_DIS_MODE_DOF_BUTT,
} hi_dis_mode;
```

成员

成员名称	描述
HI_DIS_MODE_4_DOF_GME	四自由度GME算法，不使用陀螺仪。
HI_DIS_MODE_6_DOF_GME	六自由度GME算法，不使用陀螺仪。
HI_DIS_MODE_GYRO	陀螺仪算法，不使用GME。
HI_DIS_MODE_DOF_BUTT	预留值。

11.18.20.5.22 hi_dis_motion_level

说明

定义镜头的运动级别。

定义

```
typedef enum {
    HI_DIS_MOTION_LEVEL_LOW = 0,
    HI_DIS_MOTION_LEVEL_NORMAL,
    HI_DIS_MOTION_LEVEL_HIGH,
    HI_DIS_MOTION_LEVEL_BUTT
} hi_dis_motion_level;
```

成员

成员名称	描述
HI_DIS_MOTION_LEVEL_LOW	低级别运动，镜头小幅度运动。当前版本不支持。
HI_DIS_MOTION_LEVEL_NORMAL	正常级别运动，镜头正常幅度运动。
HI_DIS_MOTION_LEVEL_HIGH	高级别运动，镜头大幅度运动。
HI_DIS_MOTION_LEVEL_BUTT	预留值。

11.18.20.5.23 hi_dis_pdt_type

说明

定义使用DIS的产品形态。

定义

```
typedef enum {
    HI_DIS_PDT_TYPE_IPC = 0,
    HI_DIS_PDT_TYPE_DV,
    HI_DIS_PDT_TYPE_DRONE,
    HI_DIS_PDT_TYPE_BUTT
} hi_dis_pdt_type;
```

成员

成员名称	描述
HI_DIS_PDT_TYPE_IPC	IPC产品形态。
HI_DIS_PDT_TYPE_DV	DV产品形态。
HI_DIS_PDT_TYPE_DRONE	无人机产品形态。
HI_DIS_PDT_TYPE_BUTT	预留值。

11.18.20.5.24 hi_dis_param

说明

定义DIS可选参数。

定义

```
typedef struct {
    hi_u32 large_motion_stable_coef;
    hi_u32 low_freq_motion_preserve;
    hi_u32 low_freq_motion_freq;
} hi_dis_param;
```

成员

成员名称	描述
large_motion_stable_coef	大幅度运动的防抖衰减参数。 取值范围：[0, 100]。
low_freq_motion_preserve	低频运动的保留程度。 取值范围：[0, 100]。
low_freq_motion_freq	低频运动的截止频率。 取值范围：[0, 100]。

11.18.20.5.25 hi_vi_ldc_attr

说明

定义镜头畸变校正属性结构体 。

定义

```
typedef struct {
    hi_bool enable;
    hi_ldc_version ldc_version;
    union {
        hi_ldc_v1_attr ldc_v1_attr;
        hi_ldc_v2_attr ldc_v2_attr;
    }
} hi_vi_ldc_attr;
```

成员

成员名称	描述
enable	控制LDC是否开启。 ● HI_FALSE：不开启； ● HI_TRUE：开启。
ldc_version	LDC版本。

成员名称	描述
ldc_v1_attr	镜头畸变校正属性，ldc_version为HI_LDC_V1时才生效。
ldc_v2_attr	镜头畸变校正属性，ldc_version为HI_LDC_V2时才生效，必须跟Gyro DIS一起使用。

11.18.20.5.26 hi_ldc_v1_attr

说明

定义HI_LDC_V1 版本的 镜头畸变校正属性 。

定义

```
typedef struct {
    hi_bool    aspect;
    hi_s32     x_ratio;
    hi_s32     y_ratio;
    hi_s32     xy_ratio;
    hi_s32 center_x_offset;
    hi_s32 center_y_offset;
    hi_s32 distortion_ratio;
} hi_ldc_v1_attr;
```

成员

成员名称	描述
aspect	畸变校正类型。 取值范围： ● HI_FALSE：不保持幅形比。 ● HI_TRUE：保持幅形比。
x_ratio	水平视场角大小，取值范围：[0, 100]，aspect=HI_FALSE时本参数有效。 其值为水平视场角介于最小视场角和最大视场角之间的一个比例。
y_ratio	垂直视场角大小，取值范围：[0, 100]，aspect=HI_FALSE时本参数有效； 其值为垂直视场角介于最小视场角和最大视场角之间的一个比例。
xy_ratio	整体视场角大小，包括水平视场角和垂直视场角，取值范围：[0, 100]，aspect=HI_FALSE时本参数有效； 其值为整体视场角介于最小视场角和最大视场角之间的一个比例。

成员名称	描述
center_x_offset	畸变中心点相对图象中心点水平偏移。 取值范围：[-511, 511]。
center_y_offset	畸变中心点相对图象中心点垂直偏移。 取值范围：[-511, 511]。
distortion_ratio	畸变程度，有效范围为[-300, 500]，负数为枕型，正数为桶型。

11.18.20.5.27 hi_ldc_v2_attr

说明

定义HI_LDC_V2版本的镜头畸变校正属性。

定义

```
typedef struct {
    hi_s32 focal_len_x;
    hi_s32 focal_len_y;
    hi_s32 coor_shift_x;
    hi_s32 coor_shift_y;
    hi_s32 src_cali_ratio[HI_SRC_LENS_COEF_SEG][HI_SRC_LENS_COEF_NUM];
    hi_s32 src_jun_pt;
    hi_s32 dst_cali_ratio[HI_DST_LENS_COEF_SEG][HI_DST_LENS_COEF_NUM];
    hi_s32 dst_jun_pt[HI_DST_LENS_COEF_SEG_POINT];
    hi_s32 max_du;
} hi_ldc_v2_attr;
```

成员

成员名称	描述
focal_len_x	水平方向镜头有效焦距。 取值范围：[6400, 117341700]。
focal_len_y	垂直方向镜头有效焦距。 取值范围：[6400, 117341700]。
coor_shift_x	光心X坐标。 取值范围：[35* Width，65* Width]，Width为图像宽。
coor_shift_y	光心Y坐标。 取值范围：[35* Height，65* Height]，Height为图像高。
src_cali_ratio	镜头畸变系数，最多支持分2 段。其中[0][0]固定为1*10^5，其他成员范围均为[-16*10^5, 16*10^5]。

成员名称	描述
src_jun_pt	镜头畸变系数。 取值范围：[0, 32*10^5]。
dst_cali_ratio	镜头畸变系数，最多支持分3段。每个成员范围均为[-16*10^5, 16*10^5]。
dst_jun_pt	镜头畸变系数。 取值范围：[0,16*10^5]。
max_du	镜头畸变系数。 取值范围：[0,1<<20]。

11.18.20.5.28 hi_ldc_version

说明

定义LDC 版本 。

定义

```
typedef enum {
    HI_LDC_V1 = 1,
    HI_LDC_V2 = 2,
    HI_LDC_VERSION_BUTT
} hi_ldc_version
```

成员

成员名称	描述
HI_LDC_V1	HI_LDC_V1版本。
HI_LDC_V2	HI_LDC_V2版本。 该版本的畸变矫正必须配合DIS陀螺仪算法使用。
HI_LDC_VERSION_BUTT	预留值。

11.18.20.5.29 hi_vi_low_delay_info

说明

定义低延时属性 。

定义

```
typedef struct {
  hi_bool enable;
  hi_u32  line_cnt;
} hi_vi_low_delay_info;
```

成员

成员名称	描述
enable	是否开启低延时开关。 取值范围： ● HI_FALSE：不开启。 ● HI_TRUE：开启。
line_cnt	低延时输出行号，最小值为32，最大值不超过图像高度，不建议设置行号太接近图像高度。

11.18.20.6 Region 区域管理

11.18.20.6.1 HI_RGN_CLUT_NUM

说明

定义RGB 颜色查表个数。

定义

```
#define HI_RGN_CLUT_NUM  16
```

11.18.20.6.2 HI_RGN_HANDLE_MAX

说明

定义区域的最大句柄个数。

定义

```
#define HI_RGN_HANDLE_MAX 1024
```

11.18.20.6.3 hi_rgn_handle

说明

定义区域句柄。

定义

```
typedef hi_u32 hi_rgn_handle
```

11.18.20.6.4 hi_phys_addr_t

说明

定义物理地址类型。

定义

```
typedef hi_u64 hi_phys_addr_t;
```

11.18.20.6.5 hi_rgn_type

说明

定义区域类型。

定义

```
typedef enum {
    HI_RGN_OVERLAY = 0,
    HI_RGN_COVER,
    HI_RGN_OVERLAYEX,
    HI_RGN_COVEREX,
    HI_RGN_LINE,
    HI_RGN_MOSAIC,
    HI_RGN_MOSAICEX,
    HI_RGN_CORNER_RECTEX,
} hi_rgn_type;
```

成员

成员名称	描述
HI_RGN_OVERLAY	视频叠加区域。
HI_RGN_COVER	视频遮挡区域。
HI_RGN_OVERLAYEX	扩展视频叠加区域。
HI_RGN_COVEREX	扩展视频遮挡区域。
HI_RGN_LINE	扩展线形遮挡区域。
HI_RGN_MOSAIC	MOSAIC视频区域。
HI_RGN_MOSAICEX	扩展MOSAICEX视频区域。
HI_RGN_CORNER_RECTEX	角框区域。

11.18.20.6.6 hi_rgn_overlayex_attr

说明

定义扩展叠加区域属性结构体。

定义

```
typedef struct {
    hi_pixel_format   pixel_format;
    hi_u32            bg_color;
    hi_size           size;
    hi_u32            canvas_num;
    hi_u32            clut[HI_RGN_CLUT_NUM];
} hi_rgn_overlayex_attr;
```

成员

成员名称	描述
pixel_format	像素格式 取值范围： HI_PIXEL_FORMAT_ARGB_1555 = 33； HI_PIXEL_FORMAT_ARGB_4444 = 25； HI_PIXEL_FORMAT_ARGB_8888 = 14； HI_PIXEL_FORMAT_ARGB_CLUT2 = 41; HI_PIXEL_FORMAT_ARGB_CLUT4 = 42。
bg_color	区域背景色。 当像素格式为HI_PIXEL_FORMAT_ARGB_1555 = 33，取值范围： [0, 0xffff]。 当像素格式为HI_PIXEL_FORMAT_ARGB_4444 = 25，取值范围： [0, 0xffff]。 当像素格式为HI_PIXEL_FORMAT_ARGB_8888 = 14，取值范围：[0, 0xffffffff]。 当像素格式为HI_PIXEL_FORMAT_ARGB_CLUT2 = 41，取值范围：[0, 0x3]。 当像素格式为HI_PIXEL_FORMAT_ARGB_CLUT4 = 42，取值范围：[0,0xf]。
size	区域的高宽 取值范围： 宽度：[2, 8192]，要求以2对齐。 高度：[2, 8192]，要求以2对齐。
canvas_num	区域的内存数量 取值范围：[1, 2]。
clut	定义RGB颜色值。HI_RGN_CLUT_NUM用于定义RGB颜色查表个数，当前表示16。 颜色值取值范围：[0x0, 0xffffffff]。 像素为HI_PIXEL_FORMAT_ARGB_CLUT2 = 41或者HI_PIXEL_FORMAT_ARGB_CLUT4 = 42时，才进行参数范围检查。

11.18.20.6.7 hi_rgn_overlayex_chn_attr

说明

定义扩展叠加区域的通道显示属性。

定义

```
typedef struct {
    hi_point      point;
    hi_u32        fg_alpha;
    hi_u32        bg_alpha;
    hi_u32        layer;
} hi_rgn_overlayex_chn_attr;
```

成员

成员名称	描述
point	区域位置。动态属性。 区域位置坐标取值范围： 水平位置X：[0, 16382]，要求以2对齐。 垂直位置Y：[0, 16382]，要求以2对齐。
fg_alpha	Alpha位为1的像素点的透明度。也称前景Alpha。动态属性。 取值范围：[0, 255]。取值越小，越透明。 仅当被叠加图片的格式为HI_PIXEL_FORMAT_ARGB_1555格式时，需设置该参数。
bg_alpha	Alpha位为0的像素点的透明度。也称背景Alpha。动态属性。 取值范围：[0, 255]。取值越小，越透明。 仅当被叠加图片的格式为HI_PIXEL_FORMAT_ARGB_1555格式时，需设置该参数。
layer	区域层次。动态属性。 取值范围与各模块最多能支持的OVERLAYEX个数相关，从0开始到通道支持区域最大个数减1。值越大，层次越高。

11.18.20.6.8 hi_rgn_cover_chn_attr

说明

定义遮挡区域的通道显示属性。

定义

```
typedef struct {
    hi_cover        cover;
    hi_u32          layer;
    hi_coord        coord;
} hi_rgn_cover_chn_attr;
```

成员

成员名称	描述
cover	COVER属性。动态属性。 其中rect表示区域坐标位置和宽高 绝对坐标（coord为0的情况）： ● 区域位置坐标取值范围： 水平位置X：[-16384, 16382]，要求以2对齐。 垂直位置Y：[-16384, 16382]，要求以2对齐。 ● 宽高取值范围： 宽度：[2, 16384]，要求以2对齐。 高度：[2, 16384]，要求以2对齐。
layer	区域层次。动态属性。 取值范围与各模块最多能支持的COVER个数相关，从0开始到通道支持区域最大个数减1。值越大，层次越高。
coord	区域坐标类型，当前只支持配置为0，表示绝对坐标。动态属性。

11.18.20.6.9 hi_rgn_mosaic_chn_attr

说明

定义mosaic区域属性结构体。

定义

```
typedef struct {
    hi_rect         rect;
    hi_blk_size     blk_size;
    hi_u32          layer;
} hi_rgn_mosaic_chn_attr;
```

成员

成员名称	描述
rect	区域位置、宽高。动态属性。 ● 区域位置坐标取值范围： 水平位置X：[-16384, 16384] 垂直位置Y：[-16384, 16384] ● 宽高取值范围： 宽度：[8, 16384] 高度：[8, 16384]
blk_size	区域块大小。动态属性。 只支持HI_BLK_SIZE_8、HI_BLK_SIZE_16、HI_BLK_SIZE_32。
layer	MOSAIC区域层次。动态属性。 取值范围与各模块最多能支持的MOSIC个数相关，从0开始到通道支持区域最大个数减1。值越大，层次越高。

11.18.20.6.10 hi_rgn_type_attr

说明

定义区域属性联合体。

定义

```
typedef union {
    hi_rgn_overlayex_attr        overlayex;
} hi_rgn_type_attr;
```

成员

成员名称	描述
overlayex	扩展叠加区域属性。

11.18.20.6.11 hi_rgn_type_chn_attr

说明

定义区域通道显示属性联合体。

定义

```
typedef union {
    hi_rgn_cover_chn_attr       cover_chn;
    hi_rgn_overlayex_chn_attr   overlayex_chn;
```

```
    hi_rgn_mosaic_chn_attr      mosaic_chn;
} hi_rgn_type_chn_attr;
```

成员

成员名称	描述
cover_chn	遮挡区域通道显示属性。
overlayex_chn	扩展叠加区域通道显示属性。
mosaic_chn	MOSAIC显示属性。

11.18.20.6.12 hi_rgn_attr

说明

定义区域属性结构体。

定义

```
typedef struct {
    hi_rgn_type            type;
    hi_rgn_type_attr        attr;
} hi_rgn_attr;
```

成员

成员名称	描述
type	区域类型。
attr	区域属性。

11.18.20.6.13 hi_rgn_chn_attr

说明

定义区域通道显示属性结构体 。

定义

```
typedef struct {
    hi_bool                    is_show;
    hi_rgn_type                  type;
    hi_rgn_type_chn_attr           attr;
} hi_rgn_chn_attr;
```

成员

成员名称	描述
is_show	区域是否显示。动态属性。 取值范围： ● HI_TRUE：显示 ● HI_FALSE：不显示
type	区域类型。静态属性。
attr	区域通道显示属性。

11.18.20.6.14 hi_rgn_canvas_info

说明

定义画布信息结构体 。

定义

```
typedef struct {
    hi_phys_addr_t          phys_addr;
    hi_size                 size;
    hi_u32                  stride;
    hi_pixel_format         pixel_format;
    hi_void  ATTRIBUTE      *virt_addr;
} hi_rgn_canvas_info;
```

成员

成员名称	描述
phys_addr	画布物理地址。
size	画布尺寸。
stride	画布的stride。
pixel_format	画布的像素格式。
virt_addr	画布虚拟地址。 用户的位图数据拷贝到该地址指向的内存中，用于更新画布内容。

11.18.20.7 VPSS 视频处理

11.18.20.7.1 基本数据类型说明

```
typedef hi_s32 hi_vpss_grp;
typedef hi_s32 hi_vpss_chn;
```

11.18.20.7.2 hi_vpss_crop_info

说明

定义裁剪功能所需信息。

定义

```
typedef struct {
    hi_bool     enable;
    hi_coord    crop_mode;
    hi_rect     crop_rect;
}hi_vpss_crop_info;
```

成员

成员名称	描述
enable	是否开启裁剪功能。 取值范围： ● HI_FALSE：不开启。 ● HI_TRUE：开启。
crop_mode	裁剪区域起始点坐标模式。
crop_rect	裁剪的矩形区域。 ● 将crop_mode参数设置为相对坐标时，裁剪区域坐标取值范围为 [0, 999]，裁剪区域宽高取值范围为[1, 1000] 。 ● 裁剪区域的尺寸不能小于VPSS支持的最小输入分辨率，不能超过VPSS支持的最大输入分辨率；如果裁剪宽度大于输入图像宽度，则裁剪输出宽度调整为输入图像宽度；如果裁剪高度大于输入图像高度，则裁剪输出高度调整为输入图像高度；裁剪区域起始点不支持负坐标，裁剪区域右边界不能超出VPSS支持的最大输入宽度，裁剪区域下边界不能超出VPSS支持的最大输入高度。 ● 如果裁剪区域超出图像范围，裁剪坐标向原点方向移动，优先保证裁剪出的宽高与所设置的参数相同。

说明

- 相对坐标模式下：
  - 横坐标计算公式： X = crop_rect.x * 实际图像宽度/1000，计算完成后会进行取整操作和对齐操作。公式同样适用于纵坐标计算。
  - 区域宽度计算公式：Width=crop_rect.width * 实际图像宽度/1000，计算完成后会进行取整操作和对齐操作。公式同样适用于区域高度计算。
  - 如果经过计算后，裁剪区域宽高小于VPSS支持的最小分辨率，则裁剪输出宽高调整为输入图像宽高，接口返回成功。
- 绝对坐标模式下：
  - 如果裁剪宽高小于VPSS支持的最小输入分辨率，则crop设置不生效，接口返回失败。
  - 如果裁剪宽高大于VPSS支持的最大输入分辨率，则crop设置不生效，接口返回失败。
  - 裁剪区域坐标和裁剪宽高之和超过VPSS支持的最大输入分辨率，则crop设置不生效，接口返回失败。

11.18.20.7.3 hi_vpss_nr_type

说明

NR（Noise Reduction）类型枚举。

定义

```
typedef enum {
    HI_VPSS_NR_TYPE_VIDEO_NORM   = 0,
    HI_VPSS_NR_TYPE_BUTT
} hi_vpss_nr_type;
```

成员

成员名称	描述
HI_VPSS_NR_TYPE_VIDEO_NORM	视频NR类型。
HI_VPSS_NR_TYPE_BUTT	预留值。

11.18.20.7.4 hi_vpss_nr_motion_mode

说明

NR的运动矢量模式。

定义

```
typedef enum {
    HI_VPSS_NR_MOTION_MODE_NORM  = 0,
    HI_VPSS_NR_MOTION_BUTT
} hi_vpss_nr_motion_mode;
```

成员

成员名称	描述
HI_VPSS_NR_MOTION_MODE_NORM	普通模式。
HI_VPSS_NR_MOTION_BUTT	预留值。

11.18.20.7.5 hi_vpss_nr_attr

说明

定义VPSS GRP的3DNR属性。开3DNR时，压缩格式只支持帧压缩和非压，而且压缩方式不能动态改变。

定义

```
typedef struct {
    hi_vpss_nr_type      nr_type;
    hi_compress_mode     compress_mode;
    hi_vpss_nr_motion_mode    nr_motion_mode;
} hi_vpss_nr_attr;
```

成员

成员名称	描述
nr_type	NR类型。
compress_mode	重构帧压缩类型，仅支持帧压缩模式HI_COMPRESS_MODE_FRAME，且开启帧压缩模式时，宽度必须大于等于640，高度必须大于等于480并小于等于4096。
nr_motion_mode	NR运动矢量模式。

11.18.20.7.6 hi_vpss_grp_attr

说明

定义VPSS Group属性。

定义

```
typedef struct {
    hi_u32                  max_width;
    hi_u32                  max_height;
    hi_pixel_format         pixel_format;
    hi_dynamic_range        dynamic_range;
    hi_frame_rate_ctrl      frame_rate;
    hi_bool                 nr_en;
    hi_vpss_nr_attr         nr_attr;
    hi_u32                  reserved[20];
} hi_vpss_grp_attr;
```

成员

成员名称	描述
max_width	最大输入图像宽度。 静态属性，创建Group时设定，不可更改。
max_height	最大输入图像高度。 静态属性，创建Group时设定，不可更改。
pixel_format	输入图像像素格式。 静态属性，创建Group时设定，不可更改。 ● VPSS Group ID在[0,256)范围内，支持以下格式：HI_PIXEL_FORMAT_YVU_SEMIPLANAR_420、HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420。 ● VPSS Group ID在[256,264)范围内，支持以下格式：HI_PIXEL_FORMAT_YVU_SEMIPLANAR_422、HI_PIXEL_FORMAT_YUV_SEMIPLANAR_422、HI_PIXEL_FORMAT_YVU_SEMIPLANAR_420、HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420、HI_PIXEL_FORMAT_YUV_400。
dynamic_range	保留参数，必须赋值为0。
frame_rate	组帧率。
nr_en	是否开启NR。 取值范围： ● HI_FALSE：不开启。 ● HI_TRUE：开启。
nr_attr	NR属性。
reserved	预留参数，为保证后续版本兼容性，请务必使用memset结构体方式进行清零初始化，在代码中必须避免显式对reserved字段进行访问。

11.18.20.7.7 hi_vpss_nr_version

说明

NR版本枚举。

定义

```
typedef enum {
    HI_VPSS_NR_V3 = 3,
    HI_VPSS_NR_BUTT
} hi_vpss_nr_version;
```

成员

成员名称	描述
HI_VPSS_NR_V3	3DNR参数版本3。

11.18.20.7.8 hi_vpss_nrx_v3

说明

定义3DNR X接口版本V3的参数。

定义

```
typedef struct {
    hi_v200_vpss_iey  iey[5];
    hi_v200_vpss_sfy  sfy[5];
    hi_v200_vpss_mdy  mdy[2];
    hi_v200_vpss_tfy  tfy[3];
    hi_v200_vpss_nr_c  nr_c;
} hi_vpss_nrx_v3;
```

成员

成名名称	描述
iey	增强模块参数。
sfy	空域滤波参数。
mdy	运动检测参数。
tfy	时域滤波参数。
nrc	色度滤波参数。

11.18.20.7.9 hi_v200_vpss_iey

说明

定义3DNR增强模块参数。

定义

```
typedef struct {
    hi_u8  ies0, ies1, ies2, ies3;
    hi_u16  iedz : 10, ie_en : 1, reserved : 5;
} hi_v200_vpss_iey;
```

成员

成名名称	描述
ies0、ies1、ies2、ies3	边缘的绝对增强强度，0~3对应不同频段。 取值范围：[0, 255]。
iedz	噪声控制阈值。 取值范围：[0, 999]。
ie_en	ie模块开关。
reserved	预留参数，为保证后续版本兼容性，请务必使用memset结构体方式进行清零初始化，在代码中必须避免显式对reserved字段进行访问。

11.18.20.7.10 hi_v200_vpss_sfy

说明

定义3DNR空域滤波参数。

定义

```
typedef struct {
    hi_u8  spn6 : 3, sfr  : 5;
    hi_u8  sbn6 : 3, pbr6 : 5;
    hi_u16  srt0 : 5, srt1 : 5, j_mode : 3, de_idx : 3;
    hi_u8  sfr6[4], sbr6[2], de_rate;
    hi_u8  sfs1,  sft1,  sbr1;
    hi_u8  sfs2,  sft2,  sbr2;
    hi_u8  sfs4,  sft4,  sbr4;
    hi_u16  sth1 : 9,  sfn1 : 3, sfn0  : 3, nr_y_en   : 1;
    hi_u16  sth_d1 : 9, reserved : 7;
    hi_u16  sth2 : 9,  sfn2 : 3, k_mode : 3, reserved_1   : 1;
    hi_u16  sth_d2 : 9, reserved_2 : 7;
    hi_u16  sbs_k[32], sds_k[32];
} hi_v200_vpss_sfy;
```

成员

成名名称	描述
j_mode	空域混合模式。 取值范围：[0,4]。
spn6、sbn6	混合模式滤波器选择。 取值范围：[0, 5]。
pbr6	表示spn6和sbn6滤波结果的混合比例，当混合模式j_mode为1时生效。 取值范围 [0,15]。

成名名称	描述
sfr6[4]	表示 由sbn6选择的滤波器产生的结果和spn6融合后的相对强度。 取值范围 [0, 31]。
de_idx	预留参数，当前只允许取默认值4。
de_rate	预留参数，当前只允许取默认值0。
srt0，srt1	预留参数，sfy[0]和sfy[1]下只允许取默认值16，sfy[2]~sfy[4]下只允许取0。
sfr	纯空域滤波器在SFi或者SFk模式下空域滤波结果通过4种滤波器审查后所能发挥的相对强度。 取值范围：[0,31]。
sfs1、sfs2、sfs4	表示1~4号滤波器强度（3和4号滤波器强度一样）。 取值范围：[0,255]。
sft1、sft2、sft4	表示1~4号滤波器附加强度。 取值范围：[0,255]。
sbr1、sbr2、sbr4、sbr6[2]	表示1~4、6号滤波器的滤波的不对称强度。 sbr1、sbr2、sbr4取值范围：[0,255]。 sbr6取值范围：[0,15]。
sth1、sth2、 sth_d1、sth_d2	保边阈值上限和下限。值越小，越多的边缘被保留，噪声也会越大；值越大，保留的边缘越少，只有很强的边缘被保留住。 取值范围：[0,511]。
sfn0、sfn1、sfn2	对应 sth1，sth2不同图像特性选择不同滤波器的类型（编号）。 取值范围：[0,6]。
k_mode	选择根据绝对亮度调整去噪强度的模式。第2级和第3级有效。 取值范围：[0,3]。
nr_y_en	每一级去噪的使能开关。 0：关闭； 1：打开。
sbs_k[32]、 sds_k[32]	根据画面的绝对亮度阈值划分的去相对暗区的去噪强度表。第2级和第3级有效。 取值范围 [0, 8191]。
reserved、 reserved_1、 reserved_2	预留参数，为保证后续版本兼容性，请务必使用memset结构体方式进行清零初始化，在代码中必须避免显式对reserved字段进行访问。

11.18.20.7.11 hi_v200_vpss_mdy

说明

定义3DNR运动检测参数。

定义

```
typedef struct {
    hi_u16  madz0   : 9,   mai00 : 2,  mai01  : 2, mai02 : 2, reserved : 1;
    hi_u16  madz1   : 9,   mai10 : 2,  mai11  : 2, mai12 : 2, reserved_1 : 1;
    hi_u8   mabr0, mabr1;
    hi_u16  math0   : 10,  mate0 : 4,  matw   : 2;
    hi_u16  math_d0  : 10,  reserved_2 : 6;
    hi_u16  math1   : 10,  reserved_3 : 6;
    hi_u16  math_d1  : 10,  reserved_4 : 6;
    hi_u8   masw    :  4,  mate1 : 4;
    hi_u8   mabw0   :  4,  mabw1 : 4;
    hi_u16  adv_math : 1,   adv_th : 12, reserved_5  : 3;
} hi_v200_vpss_mdy;
```

成员

成名名称	描述
mai00、mai01、mai02	选择采用哪种时域滤波器和空域的滤波器进行混合（通路0）。 取值范围 [0,3]。
mai10、mai11、mai12	选择采用哪种时域滤波器和空域的滤波器进行混合（通路1）。 取值范围 [0,3]。
madz0	控制根据图像的方差信息（madz0表示方差阈值的上限），在mai02中混入mai01的结果。 取值范围[0,511]。
madz1	控制根据图像的方差信息（madz1表示方差阈值的上限），在mai12中混入mai11的结果。 取值范围[0,511]。
mabr0	控制根据图像的亮度信息（mabr0表示亮度阈值），在mai02中混入mai01的结果。 取值范围[0, 255]。
mabr1	控制根据图像的亮度信息（mabr1表示亮度阈值），在mai12中混入mai11的结果。 取值范围[0, 255]。
math0、math_d0	动静判决阈值上下限（通路0），math0表示上限、math_d0表示下限，math0、math_d0的取值范围为[0, 999]，且要求math0必须大于math_d0，否则math0会默认与math_d0相等。

成名名称	描述
math1、math_d1	动静判决阈值上下限（通路1），math1表示上限、math_d1表示下限，math1、math_d1的取值范围为[0, 999]，且要求math1必须大于math_d1，否则math1会默认与math_d1相等。
mate0、mate1	表示通路0，1中平坦区域运动检测指数。 取值范围[0, 8]。
matw	表示时域滤波防运动拖尾指数，该值越大，运动拖尾收敛越快，反之，该值越小，运动拖尾收敛越慢，一般不建议调试，建议设置为默认值2。 取值范围[0, 3]。
masw	表示时域滤波防雨点指数，该值越大，有助于降低雨点噪声出现的概率，一般不建议调试，设置为默认值12。 取值范围[0, 15]。
mabw0、mabw1	表示通路0，1中运动检测内容窗口大小的选择。 取值范围[0, 9]。
adv_math	增强型MATH模式开关。0：关闭，1：打开。
adv_th	控制增强型MATH的效果。取值范围[0, 999]。
reserved、reserved_1、reserved_2、reserved_3、reserved_4、reserved_5	预留参数，为保证后续版本兼容性，请务必使用memset结构体方式进行清零初始化，在代码中必须避免显式对reserved字段进行访问。

11.18.20.7.12 hi_v200_vpss_tfy

说明

定义3DNR时域滤波参数。

定义

```
typedef struct {
    hi_u16  tfs0 : 4,   tdz0 : 10,  tdx0    : 2;
    hi_u16  tfs1 : 4,   tdz1 : 10,  tdx1    : 2;
    hi_u16  sdz0 : 10,  str0 : 5,   dz_mode0 : 1;
    hi_u16  sdz1 : 10,  str1 : 5,   dz_mode1 : 1;
    hi_u8   tss0 : 4,   tsi0 : 4,   tfr0[6];
    hi_u8   tss1 : 4,   tsi1 : 4,   tfr1[6];
    hi_u8   tfrs : 4,   ted  : 2,   ref     : 1,  reserved : 1;
} hi_v200_vpss_tfy;
```

成员

成名名称	描述
tfs0、tfs1	通路0，1中时域滤波绝对强度。
dz_mode0、dz_mode1	通路0，1中tdz的模式选择。 取值范围[0,1]
tdz0、tdz1	当dz_mode0， dz_mode1为0时，保护运动区域的纹理不受时域滤波影响，将tdz调大时，运动区域的纹理可以得到保护，同时也会带来时域滤波强度的削弱；当dz_mode0， dz_mode1为1时，增加运动区域的时域滤波强度，将tdz调大时，运动区域时域去噪能力加强。 取值范围[0, 999]。
tdx0、tdx1	该参数建议使用默认值为2，不推荐调试。 取值范围：[0, 3]。
str0、str1	通路0，1中滤波器作用后叠加在结果的比例，值越大比例越高。 取值范围：[0, 31]。
tfr0[6]、tfr1[6]	通路0，1中静止区域时域滤波的相对强度。 取值范围[0, 31]。
tss0、tss1	通路0，1中时域静止区域混入空域的比例。 取值范围[0, 15]。
tsi0、tsi1	通路0，1中时混入空域滤波器的选择。 取值范围[0, 1]
ref	参考帧开关。 0：关闭 1：打开
ted	仅在时域生效时有效，用于控制运动后新内容的方法，0表示关闭，不做处理，1、2表示使用空域方法处理，3表示使用时域方法处理。 取值范围 [0,3]。
tfrs	tfr强度控制模式。 取值范围 [0, 15]。
sdz0、sdsz1	通路0，1中约束滤波器作用强度，值越小，作用强度越低。 取值范围：[0, 999]。
reserved	预留参数，为保证后续版本兼容性，请务必使用memset结构体方式进行清零初始化，在代码中必须避免显式对reserved字段进行访问。

11.18.20.7.13 hi_v200_vpss_nr_c

说明

定义3DNR视频色度滤波参数。

定义

```
typedef struct {
    hi_u8  sfc, tfc : 6, reserved : 2;
    hi_u8  trc, tpc : 6, reserved_1 : 2;
    hi_u8 mode : 1, reserved_2 : 7;
    hi_u8  presfc : 6, reserved_3 : 2;
} hi_v200_vpss_nr_c;
```

成员

成名名称	描述
sfc	色度空域相对强度参数。 取值范围[0, 255]。
tpc	时域色噪滤波类型。 取值范围[0, 32]。
tfc	色度时域相对强度参数。 取值范围[0, 32]。
trc	用于抑制运动区域的色彩侵染现象. 取值范围[0, 255]。
mode	色度滤波模式。 取值范围[0,1]
presfc	色度空域预处理滤波器的强度。 取值范围[0,32]。
reserved、reserved_1、reserved_2、reserved_3	预留参数，为保证后续版本兼容性，请务必使用memset结构体方式进行清零初始化，在代码中必须避免显式对reserved字段进行访问。

11.18.20.7.14 hi_vpss_nrx_param_manual_v3

说明

定义3DNR X接口版本V3的手动参数。

定义

```
typedef struct {
    hi_vpss_nrx_v3 nrx_param;
} hi_vpss_nrx_param_manual_v3;
```

成员

成员名称	描述
nrx_param	3DNR接口的标准参数。

11.18.20.7.15 hi_vpss_nrx_param_auto_v3

说明

定义3DNR X接口版本V3的自动参数。

定义

```
typedef struct {
    hi_u32 param_num;
    hi_u32 *iso;
    hi_vpss_nrx_v3 *nrx_param;
} hi_vpss_nrx_param_auto_v3;
```

成员

成员名称	描述
param_num	NR自动模式配置标准参数个数。 取值范围：[1, 16]。 约束说明：自动模式下的参数需要申请内存空间，自动参数的组数为param_num，需要为iso申请不小于param_num * sizeof(hi_u32)长度的内存，需要为nrx_param申请不小于param_num * sizeof(hi_vpss_nrx_v3)长度的内存。
iso	NR自动模式ISO数组指针。 ISO值的取值范围：[100, 3276800]，且数组中的ISO值是递增的。
nrx_param	NR 自动模式标准参数数组指针。

11.18.20.7.16 hi_vpss_nrx_param_v3

说明

定义3DNR X接口版本V3的属性。

定义

```
typedef struct {
    hi_op_mode                opt_mode;
    hi_vpss_nrx_param_manual_v3 nrx_manual;
    hi_vpss_nrx_param_auto_v3   nrx_auto;
} hi_vpss_nrx_param_v3;
```

成员

成员名称	描述
opt_mode	NR选择模式。
nrx_manual	NR手动模式的标准参数。
nrx_auto	NR自动模式的标准参数。

11.18.20.7.17 hi_vpss_grp_nrx_param

说明

定义3DNR标准参数属性。

定义

```
typedef struct {
    hi_vpss_nr_version nr_version;
    union {
        hi_vpss_nrx_param_v3 nrx_param_v3;
    };
} hi_vpss_grp_nrx_param;
```

成员

成名名称	描述
nr_version	NR接口版本枚举。
nrx_param_v3	当前芯片NR接口标准参数。

11.18.20.7.18 hi_vpss_grp_param

说明

定义VPSS的3DNR参数。

定义

```
typedef struct{
    hi_vpss_grp_nrx_param    nrx_param;
    hi_u32                   reserved[4];
}hi_vpss_grp_param;
```

成员

成名名称	描述
nrx_param	3DNR标准参数。

成名名称	描述
reserved	预留参数，为保证后续版本兼容性，请务必使用memset结构体方式进行清零初始化，在代码中必须避免显式对reserved字段进行访问。

11.18.20.7.19 hi_fisheye_cfg

说明

定义FISHEYE的鱼眼镜头LMF参数配置。

定义

```
typedef struct {
    hi_u16 lmf_coef[HI_FISHEYE_LMF_COEF_NUM]; /* RW;  LMF coefficient of gdc len */
} hi_fisheye_cfg;
```

成员

成员名称	描述
lmf_coef	鱼眼镜头LMF参数。 #define HI_FISHEYE_LMF_COEF_NUM 128

须知

LMF参数要按照镜头厂商的推荐参数进行转换后再配置（正确的LMF参数符合lmf_coef[i+1] ≥lmf_coef[i]+5&& lmf_coef [i + 1] <= lmf_coef [i] + 31 && lmf_coef [57] <1024< lmf_coef [85] &&lmf_coef[0] = 0的规律），如果配置的参数不满足此规律则会报错，如果配置的参数有误则可能出现总线错误等异常现象，如果没有镜头厂商提供的参数建议关闭LMF功能。

11.18.20.7.20 hi_vpss_chn_mode

说明

定义VPSS通道的工作模式 。

定义

```
typedef enum {
    HI_VPSS_CHN_MODE_AUTO = 0,
    HI_VPSS_CHN_MODE_USER,
    HI_VPSS_CHN_MODE_BUTT
} hi_vpss_chn_mode;
```

成员

成员名称	描述
HI_VPSS_CHN_MODE_USER	用户设置模式。 如果需要支持从VPSS通道获取处理的图像数据，则必须将模式设置为HI_VPSS_CHN_MODE_USER。 若使用鱼眼矫正功能，则必须将模式设置为HI_VPSS_CHN_MODE_USER。
HI_VPSS_CHN_MODE_AUTO	自动模式。 ● AUTO模式下，hi_vpss_chn_attr中的width、height、aspect_ratio、compress_mode、pixel_format、frame_rate、video_format、depth参数设置不生效，相应设置由后端模块确定。 ● AUTO模式下，用户无法获取图像。 ● AUTO模式下，各通道仅可与一个接收者绑定，主要用于预览和回放场景下做播放控制。
HI_VPSS_CHN_MODE_BUTT	预留值。

11.18.20.7.21 hi_vpss_chn_attr

说明

定义VPSS物理通道的属性。

定义

```
typedef struct {
    hi_bool            mirror_en;
    hi_bool            flip_en;
    hi_vpss_chn_mode   chn_mode;        /* RW; vpss channel's work mode. */
    hi_u32             width;
    hi_u32             height;
    hi_video_format    video_format;    /* RW; video format of target image. */
    hi_pixel_format    pixel_format;    /* RW; pixel format of target image. */
    hi_dynamic_range   dynamic_range;   /* RW; dynamic_range of target image. */
    hi_compress_mode   compress_mode;   /* RW; compression mode of the output. */
    hi_frame_rate_ctrl frame_rate;      /* frame rate control info */
    hi_u32             depth;           /* RW; range: [0, 8]; user get list depth. */
    hi_aspect_ratio    aspect_ratio;    /* aspect ratio info. */
    hi_u32             reserved[10];
} hi_vpss_chn_attr;
```

成员

成员名称	描述
mirror_en	预留参数，当前版本不支持。 需要用户手动设置为0，避免后续版本的兼容性问题。

成员名称	描述
flip_en	预留参数，当前版本不支持。 需要用户手动设置为0，避免后续版本的兼容性问题。
chn_mode	通道工作模式。
width	目标图像宽度。 ● VPSS Group ID在[0,256)范围内，通道0和通道1都支持放大和缩小。 ● VPSS Group ID在[256,264)范围内，不同通道支持的功能不同，约束也不同，如下： – 通道0仅支持放大，最大支持16倍放大。当缩放前图像宽度小于4096时，输出宽度不允许超过8192；当缩放前图像宽度大于4096时，最大放大倍数为2。 – 通道1仅支持缩小，最大支持15倍缩小。
height	目标图像高度。 ● VPSS Group ID在[0,256)范围内，通道0和通道1都支持放大和缩小 ● VPSS Group ID在[256,264)范围内，不同通道支持的功能不同，约束也不同，如下： – 通道0仅支持放大，最大支持16倍放大。 – 通道1仅支持缩小，最大支持15倍缩小。
video_format	视频格式，仅支持HI_VIDEO_FORMAT_LINEAR。
pixel_format	目标图像像素格式。 ● VPSS Group ID在[0,256)范围内，支持以下格式：支持HI_PIXEL_FORMAT_YVU_SEMIPLANAR_420、HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420。 如果开启鱼眼矫正，则只支持HI_PIXEL_FORMAT_YVU_SEMIPLANAR_420。 ● VPSS Group ID在[256,264)范围内，支持以下格式： – 支持HI_PIXEL_FORMAT_YVU_SEMIPLANAR_422、HI_PIXEL_FORMAT_YUV_SEMIPLANAR_422、HI_PIXEL_FORMAT_YVU_SEMIPLANAR_420、HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420、HI_PIXEL_FORMAT_YUV_400。 如果开启鱼眼矫正，则不支持HI_PIXEL_FORMAT_YVU_SEMIPLANAR_422、HI_PIXEL_FORMAT_YUV_SEMIPLANAR_422、HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420。 – 支持semi-planar 422转semi-planar 420，支持通过设置通道输出像素格式实现UV反转。
dynamic_range	预留参数。
compress_mode	仅支持不压缩HI_COMPRESS_MODE_NONE。

成员名称	描述
frame_rate	帧率控制信息。 ● 源帧率与目标帧率都为-1，则不进行帧率控制，在通道AUTO模式下通道帧率控制不生效。 ● 目标帧率不能大于源帧率。
depth	用户获取通道图像的队列长度。若队列长度为0，HI_VPSS_CHN_MODE_USER模式下用户无法获取图像。 开启鱼眼矫正功能时，需要额外多分配1个图像内存资源。 取值范围：[0, 8]。
aspect_ratio	幅形比参数。幅形比处理时，先做缩放，再加黑边，缩放比例越大，芯片处理时间越长，可能会导致帧率下降。 ● 开启幅形比后，非视频区域的填充颜色，值不大于0xFFFFFF。 ● 如果输出图像存在缩放，则图像会缩放至幅形比视频区域的大小，缩放比例不能超过 VPSS 支持的缩放比例。 ● ASPECT_RATIO_MANUAL 模式下，坐标值必须满足 x：大于等于 0，小于通道图像宽； y：大于等于 0，小于通道图像高；宽高满足 w：大于等于64，小于等于通道图像宽度；h：大于等于 64,小于等于通道图像高度，并且 x+w 不大于通道图像宽，y+h 不大于通道图像高。
reserved	预留参数，为保证后续版本兼容性，请务必使用memset结构体方式进行清零初始化，在代码中必须避免显式对reserved字段进行访问。

11.18.20.7.22 hi_fisheye_correction_attr

说明

定义鱼眼矫正属性的相关配置。

定义

```
typedef struct {
    hi_bool enable;
    hi_fisheye_attr fisheye_attr;
    hi_size dst_size;
} hi_fisheye_correction_attr;
```

成员

成员名称	描述
enable	控制鱼眼矫正是否开启。若开启鱼眼矫正，输入图像分辨率增加额外约束为：宽在[1920,8192]范围内，高在[1080,8192]范围内。 取值范围： ● HI_FALSE：不开启。 ● HI_TRUE：开启。
fisheye_attr	鱼眼属性。
dst_size	鱼眼矫正后的输出图像大小。范围：[640*360, 8192*8192]

11.18.20.7.23 hi_fisheye_mount_mode

说明

定义FISHEYE属性中的安装模式。

定义

```
typedef enum {
    HI_FISHEYE_MOUNT_MODE_DESKTOP = 0, /* Desktop mount mode */
    HI_FISHEYE_MOUNT_MODE_CEILING = 1, /* Ceiling mount mode */
    HI_FISHEYE_MOUNT_MODE_WALL = 2, /* wall mount mode */
    HI_FISHEYE_MOUNT_MODE_BUTT
} hi_fisheye_mount_mode;
```

成员

成员名称	描述
HI_FISHEYE_MOUNT_MODE_DESKTOP	地装模式。
HI_FISHEYE_MOUNT_MODE_CEILING	顶装模式。
HI_FISHEYE_MOUNT_MODE_WALL	壁装模式。
HI_FISHEYE_MOUNT_MODE_BUTT	预留值。

11.18.20.7.24 hi_fisheye_view_mode

说明

定义FISHEYE属性中的校正模式。

定义

```
typedef enum {
    HI_FISHEYE_VIEW_MODE_360_PANORAMA = 0, /* 360 panorama mode of gdc correction */
```

```
    HI_FISHEYE_VIEW_MODE_180_PANORAMA = 1, /* 180 panorama mode of gdc correction */
    HI_FISHEYE_VIEW_MODE_NORM = 2, /* normal mode of gdc correction */
    HI_FISHEYE_VIEW_MODE_NO_TRANS = 3, /* no gdc correction */
    HI_FISHEYE_VIEW_MODE_BUTT
} hi_fisheye_view_mode;
```

成员

成员名称	描述
HI_FISHEYE_VIEW_MODE_360_PANORAMA	360全景模式。
HI_FISHEYE_VIEW_MODE_180_PANORAMA	180全景模式。
HI_FISHEYE_VIEW_MODE_NORM	Normal校正模式。
HI_FISHEYE_VIEW_MODE_NO_TRANS	不做校正模式，输出原图。
HI_FISHEYE_VIEW_MODE_BUTT	预留值。

11.18.20.7.25 hi_fisheye_rgn_attr

说明

定义FISHEYE每个校正区域的属性配置。

定义

```
typedef struct {
    hi_fisheye_view_mode view_mode;
    hi_u32 in_radius;
    hi_u32 out_radius;
    hi_u32 pan;
    hi_u32 tilt;
    hi_u32 hor_zoom;
    hi_u32 ver_zoom;
    hi_rect out_rect;
} hi_fisheye_rgn_attr;
```

成员

成员名称	描述
view_mode	该校正区域的校正模式。目前不支持设置为HI_FISHEYE_VIEW_MODE_NO_TRANS。
in_radius	360全景模式表示该校正区域所对应原图的内半径，其他模式无效。 取值范围：[0, out_radius)。
out_radius	360全景模式表示该校正区域所对应原图的外半径，其他模式为校正区域的可视半径。 取值范围：[1, 3 x max(width of input picture/4, height of input picture/4)]。

成员名称	描述
pan	该校正区域PTZ参数的Pan值。 取值范围：[0, 360]。
tilt	该校正区域PTZ参数的Tilt值。 取值范围：[0, 360]。
hor_zoom	该校正区域PTZ参数的水平Zoom值。 取值范围：[1, 4095]。
ver_zoom	该校正区域PTZ参数的垂直Zoom值。 取值范围：[1, 4095]。
out_rect	该校正区域的输出位置及宽高。 取值范围：位置及宽高不能超出整个鱼眼矫正输出图的范围；且起始x坐标需16像素的对齐，起始y坐标需2像素对齐。

11.18.20.7.26 hi_fisheye_attr

说明

定义FISHEYE属性的相关配置。

定义

```
typedef struct {
    hi_bool lmf_en;
    hi_bool bg_color_en;
    hi_u32  bg_color;
    hi_s32  hor_offset;
    hi_s32  ver_offset;
    hi_u32  trapezoid_coef;
    hi_s32  fan_strength;
    hi_fisheye_mount_mode mount_mode;
    hi_u32 rgn_num;
    hi_fisheye_rgn_attr fisheye_rgn_attr[HI_FISHEYE_MAX_RGN_NUM];
} hi_fisheye_attr;
```

成员

成员名称	描述
lmf_en	是否使用用户设置的鱼眼镜头LMF参数。取值范围： ● HI_FALSE：不开启。 ● HI_TRUE：开启。仅在VPSS组ID的取值范围满足[256, 268)下才可开启。

成员名称	描述
bg_color_en	是否在输出图像打上背景色。目前仅支持设置为HI_FALSE。 取值范围： ● HI_FALSE：不开启。 ● HI_TRUE：开启。
bg_color	背景色的颜色RGB888格式，取值范围[0, 0xFFFFFF]。
hor_offset	镜头中心点相对于SENSOR中心点的水平偏移，取值范围[-511, 511]，单位为像素。镜头中心点必须为图像中的有效像素点。
ver_offset	镜头中心点相对于SENSOR中心点的垂直偏移，取值范围[-511, 511]，单位为像素。镜头中心点必须为图像中的有效像素点。
trapezoid_coef	梯形校正强度系数。用于壁装时的俯仰角校正，取值范围[0, 32]。
fan_strength	扇形校正强度系数，仅在180模式时有效，取值范围[-760, 760]。
mount_mode	鱼眼校正安装模式。
rgn_num	一幅图像的校正区域数目，最多支持4个区域。
fisheye_rgn_attr	每个校正区域各自的属性配置。

11.18.20.8 VO 视频输出

11.18.20.8.1 hi_vo_dev

说明

定义设备号。

定义

```
typedef hi_s32 hi_vo_dev;
```

成员

成员名称	描述
hi_vo_dev	视频输出模块有2个视频输出设备，如下定义： 0：DHD0设备，即高清显示设备0。 1：DHD1设备，即高清显示设备1。

11.18.20.8.2 hi_vo_layer

说明

定义视频层号。

定义

```
typedef hi_s32 hi_vo_layer;
```

成员

成员名称	描述
hi_vo_layer	视频输出模块有4个视频层和5个图形层，如下定义： ● 0：HI_VO_LAYER_V0，即视频层 0； ● 1：HI_VO_LAYER_V1，即视频层 1； ● 2：HI_VO_LAYER_V2，即视频层 2，用作 PIP 层； ● 3：HI_VO_LAYER_V3，即视频层 3； ● 4：HI_VO_LAYER_G0，即图形层 0； ● 5：HI_VO_LAYER_G1，即图形层 1； ● 6：HI_VO_LAYER_G2，即图形层 2，用作鼠标层； ● 7：HI_VO_LAYER_G3，即图形层 3； ● 8：HI_VO_LAYER_G4，即图形层 4；

说明

当前版本接口HI_VO_LAYER_G0到HI_VO_LAYER_G4配置无效。

11.18.20.8.3 hi_vo_chn

说明

定义通道备号。

定义

```
typedef hi_s32 hi_vo_chn;
```

成员

成员名称	描述
hi_vo_chn	支持最大64路通道。

11.18.20.8.4 hi_vo_intf_type

说明

定义接口类型。

定义

```
typedef hi_u32 hi_vo_intf_type;
```

成员

成员名称	描述
hi_vo_intf_type	接口类型支持： HI_VO_INTF_VGA：VGA接口输出，定义值为（0x01L << 1）。 HI_VO_INTF_HDMI：HDMI0接口输出，定义值为（0x01L << 4）。 HI_VO_INTF_HDMI1：HDMI1接口输出，定义值为（0x01L << 12）。 HI_VO_INTF_MIPI：MIPITX接口输出，定义值为（0x01L << 10）。

11.18.20.8.5 hi_vo_intf_sync

说明

定义显示输出时序。

定义

```
typedef enum {
    HI_VO_OUT_PAL = 0, /* PAL standard */
    HI_VO_OUT_NTSC = 1, /* NTSC standard */
    HI_VO_OUT_960H_PAL = 2, /* ITU-R BT.1302 960 x 576 at 50 Hz (interlaced) */
    HI_VO_OUT_960H_NTSC = 3, /* ITU-R BT.1302 960 x 480 at 60 Hz (interlaced) */
    HI_VO_OUT_640x480_60 = 4, /* VESA 640 x 480 at 60 Hz (non-interlaced) CVT */
    HI_VO_OUT_480P60 = 5, /* 720 x 480 at 60 Hz. */
    HI_VO_OUT_576P50 = 6, /* 720 x 576 at 50 Hz. */
    HI_VO_OUT_800x600_60 = 7, /* VESA 800 x 600 at 60 Hz (non-interlaced) */
    HI_VO_OUT_1024x768_60 = 8, /* VESA 1024 x 768 at 60 Hz (non-interlaced) */
    HI_VO_OUT_720P50 = 9, /* 1280 x 720 at 50 Hz. */
    HI_VO_OUT_720P60 = 10, /* 1280 x 720 at 60 Hz. */
    HI_VO_OUT_1280x800_60 = 11, /* 1280*800@60Hz VGA@60Hz */
    HI_VO_OUT_1280x1024_60 = 12, /* VESA 1280 x 1024 at 60 Hz (non-interlaced) */
    HI_VO_OUT_1366x768_60 = 13, /* VESA 1366 x 768 at 60 Hz (non-interlaced) */
    HI_VO_OUT_1400x1050_60 = 14, /* VESA 1400 x 1050 at 60 Hz (non-interlaced) CVT */
    HI_VO_OUT_1440x900_60 = 15, /* VESA 1440 x 900 at 60 Hz (non-interlaced) CVT Compliant */
    HI_VO_OUT_1680x1050_60 = 16, /* VESA 1680 x 1050 at 60 Hz (non-interlaced) */
    HI_VO_OUT_1080P24 = 17, /* 1920 x 1080 at 24 Hz. */
    HI_VO_OUT_1080P25 = 18, /* 1920 x 1080 at 25 Hz. */
    HI_VO_OUT_1080P30 = 19, /* 1920 x 1080 at 30 Hz. */
    HI_VO_OUT_1080I50 = 20, /* 1920 x 1080 at 50 Hz, interlaced. */
    HI_VO_OUT_1080I60 = 21, /* 1920 x 1080 at 60 Hz, interlaced. */
    HI_VO_OUT_1080P50 = 22, /* 1920 x 1080 at 50 Hz. */
```

```
    HI_VO_OUT_1080P60 = 23, /* 1920 x 1080 at 60 Hz. */
    HI_VO_OUT_1600x1200_60 = 24, /* VESA 1600 x 1200 at 60 Hz (non-interlaced) */
    HI_VO_OUT_1920x1200_60 = 25, /* VESA 1920 x 1200 at 60 Hz (non-interlaced) CVT (Reduced Blanking)
*/
    HI_VO_OUT_1920x2160_30 = 26, /* 1920x2160_30 */
    HI_VO_OUT_2560x1440_30 = 27, /* 2560x1440_30 */
    HI_VO_OUT_2560x1440_60 = 28, /* 2560x1440_60 */
    HI_VO_OUT_2560x1600_60 = 29, /* 2560x1600_60 */
    HI_VO_OUT_3840x2160_24 = 30, /* 3840x2160_24 */
    HI_VO_OUT_3840x2160_25 = 31, /* 3840x2160_25 */
    HI_VO_OUT_3840x2160_30 = 32, /* 3840x2160_30 */
    HI_VO_OUT_3840x2160_50 = 33, /* 3840x2160_50 */
    HI_VO_OUT_3840x2160_60 = 34, /* 3840x2160_60 */
    HI_VO_OUT_4096x2160_24 = 35, /* 4096x2160_24 */
    HI_VO_OUT_4096x2160_25 = 36, /* 4096x2160_25 */
    HI_VO_OUT_4096x2160_30 = 37, /* 4096x2160_30 */
    HI_VO_OUT_4096x2160_50 = 38, /* 4096x2160_50 */
    HI_VO_OUT_4096x2160_60 = 39, /* 4096x2160_60 */
    HI_VO_OUT_7680x4320_30 = 40, /* 7680x4320_30 */
    HI_VO_OUT_240x320_50 = 41, /* 240x320_50 */
    HI_VO_OUT_320x240_50 = 42, /* 320x240_50 */
    HI_VO_OUT_240x320_60 = 43, /* 240x320_60 */
    HI_VO_OUT_320x240_60 = 44, /* 320x240_60 */
    HI_VO_OUT_800x600_50 = 45, /* 800x600_50 */
    HI_VO_OUT_720x1280_60 = 46, /* For MIPI DSI Tx 720 x1280 at 60 Hz */
    HI_VO_OUT_1080x1920_60 = 47, /* For MIPI DSI Tx 1080x1920 at 60 Hz */
    HI_VO_OUT_800x480_60 = 48, /* For MIPI DSI Tx 800x480 at 60 Hz */
    HI_VO_OUT_1080x1920_60_VIDEO = 49, /* For MIPI DSI Tx 1080x1920 at 60 Hz in video mode */
    HI_VO_OUT_USER = 50, /* User timing. */
    HI_VO_OUT_BUTT
} hi_vo_intf_sync;
```

成员

成员名称	描述
HI_VO_OUT_PAL	PAL（Phase Alternating Line）制式。
HI_VO_OUT_NTSC	NTSC（National Television System Committee）制式。
HI_VO_OUT_960H_PAL	PAL 960H制式。
HI_VO_OUT_960H_NTSC	NTSC 960H制式。
HI_VO_OUT_640x480_60	640x480 60帧 。
HI_VO_OUT_480P60	480P 60帧， HDMI/VGA支持。
HI_VO_OUT_576P50	576P 50帧。
HI_VO_OUT_800x600_60	800x600 60帧 ，HDMI/VGA支持。
HI_VO_OUT_1024x768_60	1024x768 60帧。
HI_VO_OUT_720P50	720P 50帧。
HI_VO_OUT_720P60	720P 60帧，HDMI/VGA支持。
HI_VO_OUT_1280x800_60	1280x800 60帧。
HI_VO_OUT_1280x1024_60	1280x1024 60帧。

成员名称	描述
HI_VO_OUT_1366x768_60	1366x768 60帧。
HI_VO_OUT_1400x1050_60	1400x1050 60帧。
HI_VO_OUT_1440x900_60	1440x900 60帧。
HI_VO_OUT_1680x1050_60	1680x1050 60帧。
HI_VO_OUT_1080P24	1080P 24帧。
HI_VO_OUT_1080P25	1080P 25帧。
HI_VO_OUT_1080P30	1080P 30帧，HDMI/VGA支持。
HI_VO_OUT_1080I50	1080I 50帧。
HI_VO_OUT_1080I60	1080I 60帧。
HI_VO_OUT_1080P50	1080P50帧，HDMI/VGA支持。
HI_VO_OUT_1080P60	1080P60帧，HDMI/VGA支持。
HI_VO_OUT_1600x1200_60	1600x1200 60帧，HDMI支持。
HI_VO_OUT_1920x1200_60	1920x1200 60帧，HDMI支持。
HI_VO_OUT_1920x2160_30	1920x2160 30帧。
HI_VO_OUT_2560x1440_30	2560x1440 30帧。
HI_VO_OUT_2560x1440_60	2560x1440 60帧。
HI_VO_OUT_2560x1600_60	2560x1600 60帧。
HI_VO_OUT_3840x2160_24	3840x2160 24帧，HDMI支持。
HI_VO_OUT_3840x2160_25	3840x2160 25帧，HDMI支持。
HI_VO_OUT_3840x2160_30	3840x2160 30帧，HDMI支持。
HI_VO_OUT_3840x2160_50	3840x2160 50帧，HDMI支持。
HI_VO_OUT_3840x2160_60	3840x2160 60帧，HDMI支持。

成员名称	描述
HI_VO_OUT_4096x2160_24	4096x2160 24帧。
HI_VO_OUT_4096x2160_25	4096x2160 25帧。
HI_VO_OUT_4096x2160_30	4096x2160 30帧。
HI_VO_OUT_4096x2160_50	4096x2160 50帧。
HI_VO_OUT_4096x2160_60	4096x2160 60帧。
HI_VO_OUT_7680x4320_30	7680x4320 30帧。
HI_VO_OUT_240x320_50	240x320 50帧。
HI_VO_OUT_320x240_50	320x240 50帧。
HI_VO_OUT_240x320_60	240x320 60帧。
HI_VO_OUT_320x240_60	320x240 60帧。
HI_VO_OUT_800x600_50	800x600 50帧。
HI_VO_OUT_720x1280_60	720x1280 60帧。
HI_VO_OUT_1080x1920_60	1080x1920 60帧，MIPITX Mate9屏cmd模式支持。
HI_VO_OUT_800x480_60	800x480 60帧，MIPITX 树莓派屏支持。
HI_VO_OUT_1080x1920_60_VIDEO	1080x1920 60帧，MIPITX Mate9屏video模式支持。
HI_VO_OUT_USER	用户配置规格。
HI_VO_OUT_BUTT	预留值。

11.18.20.8.6 hi_vo_sync_info

说明

定义时序信息结构体。

说明

当前版本目前该结构体未使用，仅在**hi_vo_intf_sync**设置为HI_VO_OUT_USER时用户才感知。

定义

```
typedef struct {
    hi_bool syncm;
```

```
    hi_bool iop;
    hi_u8 intfb;
    hi_u16 vact;
    hi_u16 vbb;
    hi_u16 vfb;
    hi_u16 hact;
    hi_u16 hbb;
    hi_u16 hfb;
    hi_u16 hmid;
    hi_u16 bvact;
    hi_u16 bvbb;
    hi_u16 bvfb;
    hi_u16 hpw;
    hi_u16 vpw;
    hi_bool idv;
    hi_bool ihs;
    hi_bool ivs;
} hi_vo_sync_info;
```

成员

成员名称	描述
syncm	同步模式，RGB接口选择1，表示信号同步。取值范围：[0, 1]
iop	0为隔行，1为逐行，RGB接口一般配置为 1。取值范围：[0, 1]
intfb	无效参数，可以忽略。取值范围：[0, 255]
vact	垂直有效区，隔行输出时表示顶场垂直有效区。单位：行。取值范围：[100, 4096]
vbb	垂直消隐后肩，隔行输出时表示顶场垂直消隐后肩。单位：行。取值范围：[1, 256]
vfb	垂直消隐前肩，隔行输出时表示顶场垂直消隐前肩。单位：行。取值范围：[1, 256]
hact	水平有效区。单位：像素。取值范围：[1, 4096]
hbb	水平消隐后肩。单位：像素。取值范围：[1, 65535]
hfb	水平消隐前肩。单位：像素。取值范围：[1, 65535]
hmid	底场垂直同步有效像素值。取值范围：[0, 65535]
bvact	底场垂直有效区，隔行时有效。单位：行。取值范围：[0, 4096]
bvbb	底场垂直消隐后肩，隔行时有效。单位：行。取值范围：[0, 256]
bvfb	底场垂直消隐前肩，隔行时有效。单位：行。取值范围：[0, 256]。
hpw	水平同步信号的宽度。单位：像素。取值范围：[1, 65535]
vpw	垂直同步信号的宽度。单位：行。取值范围：[1, 256]

成员名称	描述
idv	数据有效信号的极性。配置0为高有效，配置1为低有效。取值范围：[0, 1]
ihs	水平有效信号的极性，配置0为高有效，配置1为低有效。取值范围：[0, 1]
ivs	垂直有效信号的极性，配置0为高有效，配置1为低有效。取值范围：[0, 1]

11.18.20.8.7 hi_vo_partition_mode

说明

定义视频层分割处理模式。

定义

```
typedef enum {
    HI_VO_PARTITION_MODE_SINGLE = 0,
    HI_VO_PARTITION_MODE_MULTI = 1,
    HI_VO_PARTITION_MODE_BUTT
} hi_vo_partition_mode;
```

成员

成员名称	描述
HI_VO_PARTITION_MODE_SINGLE	视频层按照单区域配置给硬件显示。
HI_VO_PARTITION_MODE_MULTI	视频层按照多区域配置给硬件显示。
HI_VO_PARTITION_MODE_BUTT	预留值。

11.18.20.8.8 hi_vo_pub_attr

说明

定义视频输出公共属性结构体。

定义

```
typedef struct {
    hi_u32 bg_color;
    hi_vo_intf_type intf_type;
    hi_vo_intf_sync intf_sync;
    hi_vo_sync_info sync_info;
} hi_vo_pub_attr;
```

成员

成员名称	描述
bg_color	设备背景色，表示方法为RGB888。
intf_type	接口类型，可选择一种类型或多种类型的组合。
intf_sync	接口时序。
sync_info	接口时序信息。 intf_sync配置用户时序HI_VO_OUT_USER时，该结构体生效。

说明

当前版本支持组合场景如下：

HI_VO_INTF_HDMI绑定DHD0；不支持与HI_VO_INTF_HDMI1，HI_VO_INTF_VGA，HI_VO_INTF_MIPI同时送显

HI_VO_INTF_VGA绑定DHD0；不支持与HI_VO_INTF_HDMI，HI_VO_INTF_HDMI1，HI_VO_INTF_MIPI同时送显

HI_VO_INTF_MIPI绑定DHD0；不支持与HI_VO_INTF_HDMI，HI_VO_INTF_HDMI1，HI_VO_INTF_VGA同时送显

HI_VO_INTF_HDMI1绑定DHD1；不支持与HI_VO_INTF_HDMI，HI_VO_INTF_VGA，HI_VO_INTF_MIPI同时送显

11.18.20.8.9 hi_vo_video_layer_attr

说明

定义视频层属性。

定义

```
typedef struct {
    hi_rect display_rect;
    hi_size img_size;
    hi_u32 display_frame_rate;
    hi_pixel_format pixel_format;
    hi_bool double_frame_en;
    hi_bool cluster_mode_en;
    hi_dynamic_range dst_dynamic_range;
    hi_u32 display_buf_len;
    hi_vo_partition_mode partition_mode;
    hi_compress_mode compress_mode;
} hi_vo_video_layer_attr;
```

成员

成员名称	描述
display_rect	视频显示区域矩形结构体， ● 当partition_mode参数设置为HI_VO_PARTITION_MODE_SINGLE时，display_rect.x、display_rect.y为动态属性，display_rect.width、display_rect.height为静态属性。 ● 当partition_mode参数设置为HI_VO_PARTITION_MODE_MULTI时，display_rect为静态属性。
img_size;	图像分辨率结构体，即合成画面尺寸，静态属性。
display_frame_rate	视频显示帧率，静态属性。
pixel_format	视频层输入像素格式，静态属性。 支持以下格式：YVU420 SEMIPLANAR、YVU422 SEMIPLANAR、YUV400。
double_frame_en	视频层倍帧开关，静态属性。
cluster_mode_en	视频层内存聚集使能开关，静态属性。
dst_dynamic_range	视频层输出动态范围类型，静态属性。
display_buf_len	视频层显示缓存的长度，静态属性。 当前场景无需关注。
partition_mode	视频层的分割模式，静态属性。
compress_mode	视频层支持压缩或解压模式，静态属性。

11.18.20.8.10 hi_vo_chn_attr

说明

定义视频输出通道属性。

定义

```
typedef struct {
    hi_u32 priority;
    hi_rect rect;
    hi_bool deflicker_en;
} hi_vo_chn_attr;
```

成员

成员名称	描述
priority	视频通道叠加优先级，数值越大优先级越高，优先级高的在上层。 hi_vo_video_layer_attr.partition_mode参数设置为HI_VO_PARTITION_MODE_SINGLE时，priority参数值有效，且取值范围为：[0, 64 ）。 动态属性。
rect	通道矩形显示区域。 以屏幕的左上角为原点，其取值必须是2对齐，且该矩形区域必须在屏幕范围之内、矩形区域的宽高要求2对齐、矩形区域的宽高必须不小于64。 动态属性。
deflicker_en	是否使能抗闪烁，当前不支持。 ● HI_TRUE：使能 ● HI_FALSE：禁用

11.18.20.8.11 hi_vo_chn_param

说明

定义视频输出通道参数，用于幅型比功能。

定义

```
typedef struct {
    hi_aspect_ratio aspect_ratio
} hi_vo_chn_param;
```

成员

成员名称	描述
aspect_ratio	幅型比参数。

11.18.20.9 TDE 图形绘制

11.18.20.9.1 hi_tde_color_format

说明

颜色格式枚举。

定义

```
typedef enum {
    HI_TDE_COLOR_FORMAT_RGB444 = 0,
    HI_TDE_COLOR_FORMAT_BGR444,
    HI_TDE_COLOR_FORMAT_RGB555,
    HI_TDE_COLOR_FORMAT_BGR555,
    HI_TDE_COLOR_FORMAT_RGB565,
    HI_TDE_COLOR_FORMAT_BGR565,
    HI_TDE_COLOR_FORMAT_RGB888,
    HI_TDE_COLOR_FORMAT_BGR888,
    HI_TDE_COLOR_FORMAT_ARGB4444,
    HI_TDE_COLOR_FORMAT_ABGR4444,
    HI_TDE_COLOR_FORMAT_RGBA4444,
    HI_TDE_COLOR_FORMAT_BGRA4444,
    HI_TDE_COLOR_FORMAT_ARGB1555,
    HI_TDE_COLOR_FORMAT_ABGR1555,
    HI_TDE_COLOR_FORMAT_RGBA1555,
    HI_TDE_COLOR_FORMAT_BGRA1555,
    HI_TDE_COLOR_FORMAT_ARGB8565,
    HI_TDE_COLOR_FORMAT_ABGR8565,
    HI_TDE_COLOR_FORMAT_RGBA8565,
    HI_TDE_COLOR_FORMAT_BGRA8565,
    HI_TDE_COLOR_FORMAT_ARGB8888,
    HI_TDE_COLOR_FORMAT_ABGR8888,
    HI_TDE_COLOR_FORMAT_RGBA8888,
    HI_TDE_COLOR_FORMAT_BGRA8888,
    HI_TDE_COLOR_FORMAT_RABG8888,
    HI_TDE_COLOR_FORMAT_CLUT1,
    HI_TDE_COLOR_FORMAT_CLUT2,
    HI_TDE_COLOR_FORMAT_CLUT4,
    HI_TDE_COLOR_FORMAT_CLUT8,
    HI_TDE_COLOR_FORMAT_ACLUT44,
    HI_TDE_COLOR_FORMAT_ACLUT88,
    HI_TDE_COLOR_FORMAT_A1,
    HI_TDE_COLOR_FORMAT_A8,
    HI_TDE_COLOR_FORMAT_YCbCr888,
    HI_TDE_COLOR_FORMAT_AYCbCr8888,
    HI_TDE_COLOR_FORMAT_YCbCr422,
    HI_TDE_COLOR_FORMAT_PKGVYUY,
    HI_TDE_COLOR_FORMAT_BYTE,
    HI_TDE_COLOR_FORMAT_HALFWORD,
    HI_TDE_COLOR_FORMAT_JPG_YCbCr400MBP,
    HI_TDE_COLOR_FORMAT_JPG_YCbCr422MBHP,
    HI_TDE_COLOR_FORMAT_JPG_YCbCr422MBVP,
    HI_TDE_COLOR_FORMAT_MP1_YCbCr420MBP,
    HI_TDE_COLOR_FORMAT_MP2_YCbCr420MBP,
    HI_TDE_COLOR_FORMAT_MP2_YCbCr420MBI,
    HI_TDE_COLOR_FORMAT_JPG_YCbCr4,
    HI_TDE_COLOR_FORMAT_JPG_YCbCr444MBP,
    HI_TDE_COLOR_FORMAT_MAX
} hi_tde_color_format;
```

成员

成员名称	描述
HI_TDE_COLOR_FORMAT_RGB444	RGB444颜色格式。
HI_TDE_COLOR_FORMAT_BGR444	BGR444颜色格式。
HI_TDE_COLOR_FORMAT_RGB555	RGB555颜色格式。

成员名称	描述
HI_TDE_COLOR_FORMAT_BGR555	BGR555颜色格式。
HI_TDE_COLOR_FORMAT_RGB565	RGB565颜色格式。
HI_TDE_COLOR_FORMAT_BGR565	BGR565颜色格式。
HI_TDE_COLOR_FORMAT_RGB888	RGB888颜色格式。
HI_TDE_COLOR_FORMAT_BGR888	BGR888颜色格式。
HI_TDE_COLOR_FORMAT_ARGB4444	ARGB4444颜色格式。
HI_TDE_COLOR_FORMAT_ABGR4444	ABGR4444颜色格式。
HI_TDE_COLOR_FORMAT_RGBA4444	RGBA4444颜色格式。
HI_TDE_COLOR_FORMAT_BGRA4444	BGRA4444颜色格式。
HI_TDE_COLOR_FORMAT_ARGB1555	ARGB1555颜色格式。
HI_TDE_COLOR_FORMAT_ABGR1555	ABGR1555颜色格式。
HI_TDE_COLOR_FORMAT_RGBA1555	RGBA1555颜色格式。
HI_TDE_COLOR_FORMAT_BGRA1555	BGRA1555颜色格式。
HI_TDE_COLOR_FORMAT_ARGB8565	ARGB8565颜色格式。
HI_TDE_COLOR_FORMAT_ABGR8565	ABGR8565颜色格式。
HI_TDE_COLOR_FORMAT_RGBA8565	RGBA8565颜色格式。
HI_TDE_COLOR_FORMAT_BGRA8565	BGRA8565颜色格式。
HI_TDE_COLOR_FORMAT_ARGB8888	ARGB8888颜色格式
HI_TDE_COLOR_FORMAT_ABGR8888	ABGR8888颜色格式。

成员名称	描述
HI_TDE_COLOR_FORMAT_RGBA8888	RGBA8888颜色格式。
HI_TDE_COLOR_FORMAT_BGRA8888	BGRA8888颜色格式。
HI_TDE_COLOR_FORMAT_RABG8888	RABG8888颜色格式。
HI_TDE_COLOR_FORMAT_CLUT1	CLUT1颜色格式。
HI_TDE_COLOR_FORMAT_CLUT2	CLUT2颜色格式。
HI_TDE_COLOR_FORMAT_CLUT4	CLUT4颜色格式。
HI_TDE_COLOR_FORMAT_CLUT8	CLUT8颜色格式。
HI_TDE_COLOR_FORMAT_ACLUT44	ACLUT44颜色格式。
HI_TDE_COLOR_FORMAT_ACLUT88	ACLUT88颜色格式。
HI_TDE_COLOR_FORMAT_A1	A1颜色格式。
HI_TDE_COLOR_FORMAT_A8	A8颜色格式。
HI_TDE_COLOR_FORMAT_YCbCr888	YCbCr888颜色格式。
HI_TDE_COLOR_FORMAT_AYCbCr8888	AYCbCr8888颜色格式。
HI_TDE_COLOR_FORMAT_YCbCr422	YCbCr422颜色格式，YVYU排列。
HI_TDE_COLOR_FORMAT_PKGVYUY	YCbCr422颜色格式，VYUY排列。
HI_TDE_COLOR_FORMAT_BYTE	Byte格式。
HI_TDE_COLOR_FORMAT_HALFWORD	halfword格式。
HI_TDE_COLOR_FORMAT_JPG_YCbCr400MBP	YCbCr400MBP颜色格式。
HI_TDE_COLOR_FORMAT_JPG_YCbCr422MBHP	YCbCr422MBHP颜色格式。

成员名称	描述
HI_TDE_COLOR_FORMAT_JPG_YCbCr422MBVP	YCbCr422MBVP颜色格式。
HI_TDE_COLOR_FORMAT_MP1_YCbCr420MBP	YCbCr420MBP颜色格式。
HI_TDE_COLOR_FORMAT_MP2_YCbCr420MBP	YCbCr420MBP颜色格式。
HI_TDE_COLOR_FORMAT_MP2_YCbCr420MBI	YCbCr420MBI颜色格式。
HI_TDE_COLOR_FORMAT_JPG_YCbCr420MBP	YCbCr420MBP颜色格式。
HI_TDE_COLOR_FORMAT_JPG_YCbCr444MBP	YCbCr444MBP颜色格式。
TDE_COLOR_FMT_BUTT	无效的像素格式。

11.18.20.9.2 hi_tde_surface

说明

位图信息结构体。

定义

```
typedef struct {
    hi_u64 phys_addr;
    hi_u32 phys_len;
    hi_tde_color_format color_format;
    hi_u32 height;
    hi_u32 width;
    hi_u32 stride;
    hi_bool is_ycbcr_clut;
    hi_bool alpha_max_is_255;
    hi_u8 alpha0;
    hi_u8 alpha1;
    hi_u64 cbcr_phys_addr;
    hi_u32 cbcr_phys_len;
    hi_u32 cbcr_stride;
    hi_u64 clut_phys_addr;
    hi_u32 clut_phys_len;
} hi_tde_surface;
```

成员

成员名称	描述
phys_addr	位图或Y分量的起始地址。
phys_len	物理地址长度。
color_format	颜色格式。

成员名称	描述
height	位图高度。
width	位图宽度。
stride	位图或Y分量的步长。
is_ycbcr_clut	CLUT是否在YCbCr空间中。
alpha_max_is_255	alpha最大值是128还是255。
support_alpha_ex_1555	ARGB1555位图中是否支持alpha扩展。
alpha0	alpha0，用于ARGB1555 format。
alpha1	alpha1，用于ARGB1555 format。
cbcr_phys_addr	CbCr分量的地址。
cbcr_phys_len	CbCr分量的地址长度。
cbcr_stride	CbCr分量的步长。
clut_phys_addr	Clut表的地址。
clut_phys_len	Clut表的地址长度。

11.18.20.9.3 hi_tde_rect

说明

矩形信息结构体。

定义

```
typedef struct {
    hi_s32 pos_x;
    hi_s32 pos_y;
    hi_u32 width;
    hi_u32 height;
} hi_tde_rect;
```

成员

成员名称	描述
pos_x	水平坐标。
pos_y	垂直坐标。
width	宽度。
height	高度。

11.18.20.9.4 hi_tde_none_src

说明

无源位图操作区域位图信息结构体，当前在快速填充功能中作为入参使用。

定义

```
typedef struct {
    hi_tde_surface *dst_surface;
    hi_tde_rect *dst_rect;
} hi_tde_none_src;
```

成员

成员名称	描述
dst_surface	目标位图。
dst_rect	目标位图操作区域。

11.18.20.9.5 hi_tde_single_src

说明

单源位图区域信息和目标位图区域信息结构体，当前在快速拷贝功能中作为入参使用。

定义

```
typedef struct {
    hi_tde_surface *src_surface;
    hi_tde_surface *dst_surface;
    hi_tde_rect *src_rect;
    hi_tde_rect *dst_rect;
} hi_tde_single_src;
```

成员

成员名称	描述
src_surface	源位图。
dst_surface	目标位图。
src_rect	源位图操作区域。
dst_rect	目标位图操作区域。

11.18.20.9.6 hi_tde_double_src

说明

双源位图区域信息和目标位图区域信息结构体，当前在模式填充中使用。

定义

```
typedef struct {
    hi_tde_surface *bg_surface;
    hi_tde_surface *fg_surface;
    hi_tde_surface *dst_surface;
    hi_tde_rect *bg_rect;
    hi_tde_rect *fg_rect;
    hi_tde_rect *dst_rect;
} hi_tde_double_src;
```

成员

成员名称	描述
bg_surface	背景位图。
fg_surface	前景位图。
dst_surface	目标位图。
bg_rect	背景位图操作区域。
fg_rect	前景位图操作区域。
dst_rect	目标位图操作区域。

11.18.20.9.7 hi_tde_fill_color

说明

填充颜色信息结构体。

定义

```
typedef struct {
    hi_tde_color_format color_format;
    hi_u32 color_value;
} hi_tde_fill_color;
```

成员

成员名称	描述
color_format	颜色格式。
color_value	颜色数据。

11.18.20.9.8 hi_tde_blend_cmd

说明

混合选项命令。

定义

```
typedef enum {
    HI_TDE_BLEND_CMD_NONE = 0x0,
    HI_TDE_BLEND_CMD_CLEAR,
    HI_TDE_BLEND_CMD_SRC,
    HI_TDE_BLEND_CMD_SRCOVER,
    HI_TDE_BLEND_CMD_DSTOVER,
    HI_TDE_BLEND_CMD_SRCIN,
    HI_TDE_BLEND_CMD_DSTIN,
    HI_TDE_BLEND_CMD_SRCOUT,
    HI_TDE_BLEND_CMD_DSTOUT,
    HI_TDE_BLEND_CMD_SRCATOP,
    HI_TDE_BLEND_CMD_DSTATOP,
    HI_TDE_BLEND_CMD_ADD,
    HI_TDE_BLEND_CMD_XOR,
    HI_TDE_BLEND_CMD_DST,
    HI_TDE_BLEND_CMD_CONFIG,
    HI_TDE_BLEND_CMD_MAX
} hi_tde_blend_cmd;
```

成员

描述中有以下定义

ff：前景图混合系数，范围[0, 1]

bf：背景图混合系数，范围[0, 1]

fa：前景图透明度, 范围[0, 1]

ba：背景图透明度，范围[0, 1]

成员名称	描述
HI_TDE_BLEND_CMD_NONE	ff取fa，bf取1.0 - fa。
HI_TDE_BLEND_CMD_CLEAR	ff取0.0，fd取0.0。
HI_TDE_BLEND_CMD_SRC	ff取1.0，fd取0.0。
HI_TDE_BLEND_CMD_SRCOVER	ff取1.0，bf取1.0 - fa。
HI_TDE_BLEND_CMD_DSTOVER	ff取1.0 – ba，bf取1.0。
HI_TDE_BLEND_CMD_SRCIN	ff取ba，bf取0.0。
HI_TDE_BLEND_CMD_DSTIN	ff取0.0，bf取fa。

成员名称	描述
HI_TDE_BLEND_CMD_SRCOUT	ff取1.0 – ba，bf取0.0。
HI_TDE_BLEND_CMD_DSTOUT	ff取0.0, bf取1.0 - fa。
HI_TDE_BLEND_CMD_SRCATOP	ff取ba, bf取1.0 - fa。
HI_TDE_BLEND_CMD_DSTATOP	ff取1.0 - ba, bf取fa。
HI_TDE_BLEND_CMD_ADD	ff取1.0, bf取1.0。
HI_TDE_BLEND_CMD_XOR	ff取1.0 - ba, bf取1.0 - fa。
HI_TDE_BLEND_CMD_DST	ff取0.0, bf取1.0。
HI_TDE_BLEND_CMD_CONFIG	用户自定义参数规则。
HI_TDE_BLEND_CMD_MAX	无效混合选项。

11.18.20.9.9 hi_tde_blend_mode

说明

混合选项命令。

定义

```
typedef enum {
    HI_TDE_BLEND_ZERO = 0x0,
    HI_TDE_BLEND_ONE,
    HI_TDE_BLEND_SRC2COLOR,
    HI_TDE_BLEND_INVSRC2COLOR,
    HI_TDE_BLEND_SRC2ALPHA,
    HI_TDE_BLEND_INVSRC2ALPHA,
    HI_TDE_BLEND_SRC1COLOR,
    HI_TDE_BLEND_INVSRC1COLOR,
    HI_TDE_BLEND_SRC1ALPHA,
    HI_TDE_BLEND_INVSRC1ALPHA,
    HI_TDE_BLEND_SRC2ALPHASAT,
    HI_TDE_BLEND_MAX
} hi_tde_blend_mode;
```

成员

描述中有以下定义

ff：前景图混合系数，范围[0, 1]

bf：背景图混合系数，范围[0, 1]

fa：前景图透明度, 范围[0, 1]

ba：背景图透明度，范围[0, 1]

成员名称	描述
HI_TDE_BLEND_ZERO	混合系数取0。
HI_TDE_BLEND_ONE	混合系数取1。
HI_TDE_BLEND_SRC2COLOR	混合系数取fc。
HI_TDE_BLEND_INVSRC2COLOR	混合系数取1-fc。
HI_TDE_BLEND_SRC2ALPHA	混合系数取fa。
HI_TDE_BLEND_INVSRC2ALPHA	混合系数取1- fa。
HI_TDE_BLEND_SRC1COLOR	混合系数取bc。
HI_TDE_BLEND_INVSRC1COLOR	混合系数取1-bc。
HI_TDE_BLEND_SRC1ALPHA	混合系数取ba。
HI_TDE_BLEND_INVSRC1ALPHA	混合系数取1-ba。
HI_TDE_BLEND_SRC2ALPHASAT	混合系数取min(1 – ba，fa) + 1。
HI_TDE_BLEND_MAX	无效的alpha混合模式。

11.18.20.9.10 hi_tde_rop_mode

说明

TDE支持的ROP操作类型。

定义

```
typedef enum {
    HI_TDE_ROP_BLACK = 0,    /* Blackness */
    HI_TDE_ROP_NOTMERGEPEN, /* ~(S2 | S1) */
    HI_TDE_ROP_MASKNOTPEN,   /* ~S2&S1 */
    HI_TDE_ROP_NOTCOPYPEN,   /* ~S2 */
    HI_TDE_ROP_MASKPENNOT,   /* S2&~S1 */
    HI_TDE_ROP_NOT,          /* ~S1 */
    HI_TDE_ROP_XORPEN,       /* S2^S1 */
    HI_TDE_ROP_NOTMASKPEN,   /* ~(S2 & S1) */
```

```
    HI_TDE_ROP_MASKPEN,     /* S2&S1 */
    HI_TDE_ROP_NOTXORPEN,   /* ~(S2^S1) */
    HI_TDE_ROP_NOP,         /* S1 */
    HI_TDE_ROP_MERGENOTPEN, /* ~S2|S1 */
    HI_TDE_ROP_COPYPEN,     /* S2 */
    HI_TDE_ROP_MERGEPENNOT, /* S2|~S1 */
    HI_TDE_ROP_MERGEPEN,    /* S2|S1 */
    HI_TDE_ROP_WHITE,       /* Whiteness */
    HI_TDE_ROP_MAX
} hi_tde_rop_mode;
```

成员

描述中有以下定义：S1表示位图1，S2表示位图2。

成员名称	描述
HI_TDE_ROP_BLACK	黑色。
HI_TDE_ROP_NOTMERGEPEN	~(S1+S2) 。
HI_TDE_ROP_MASKNOTPEN	~S2&S1。
HI_TDE_ROP_NOTCOPYPEN	~S2。
HI_TDE_ROP_MASKPENNOT	S2&~S1。
HI_TDE_ROP_NOT	~S1。
HI_TDE_ROP_XORPEN	S2^S1。
HI_TDE_ROP_NOTMASKPEN	~(S2^S1) 。
HI_TDE_ROP_MASKPEN	S2&S1。
HI_TDE_ROP_NOTXORPEN	~(S2^S1)。
HI_TDE_ROP_NOP	S1。
HI_TDE_ROP_MERGENOTPEN	~S2 + S1。
HI_TDE_ROP_COPYPEN	S2。
HI_TDE_ROP_MERGEPENNOT	S2 + ~S1。
HI_TDE_ROP_MERGEPEN	S2 + S1 。
HI_TDE_ROP_WHITE	白色。
HI_TDE_ROP_MAX	无效ROP类型。

11.18.20.9.11 hi_tde_clip_mode

说明

TDE裁剪的配置属性。

定义

```
typedef enum {
    HI_TDE_CLIP_MODE_NONE = 0,
    HI_TDE_CLIP_MODE_INSIDE,
    HI_TDE_CLIP_MODE_OUTSIDE,
    HI_TDE_CLIP_MODE_MAX
} hi_tde_clip_mode;
```

成员

描述中有以下定义：S1表示位图1，S2表示位图2。

成员名称	描述
HI_TDE_CLIP_MODE_NONE	输出结果不做剪切。
HI_TDE_CLIP_MODE_INSIDE	区域内剪切模式 。
HI_TDE_CLIP_MODE_OUTSIDE	区域外剪切模式。
HI_TDE_CLIP_MODE_MAX	无效剪切模式。

11.18.20.9.12 hi_tde_colorkey_mode

说明

TDE colorkey的配置属性。

定义

```
typedef enum {
    HI_TDE_COLORKEY_MODE_NONE = 0,
    HI_TDE_COLORKEY_MODE_FG,
    HI_TDE_COLORKEY_MODE_BG,
    HI_TDE_COLORKEY_MODE_MAX
} hi_tde_colorkey_mode;
```

成员

成员名称	描述
HI_TDE_COLORKEY_MODE_NONE	不做colorkey操作。

成员名称	描述
HI_TDE_COLORKEY_MODE_FG	对前景位图进行colorkey操作。
HI_TDE_COLORKEY_MODE_BG	对背景位图进行colorkey操作。
HI_TDE_COLORKEY_MODE_MAX	无效的colorkey模式 。

11.18.20.9.13 hi_tde_colorkey_component

说明

TDE单个颜色分量的关键色属性。

定义

```
typedef struct {
    hi_u8 min_component;
    hi_u8 max_component;
    hi_u8 is_component_out;
    hi_u8 is_component_ignore;
    hi_u8 component_mask;
    hi_u8 component_reserved;
    hi_u8 component_reserved1;
    hi_u8 component_reserved2;
} hi_tde_colorkey_component;
```

成员

成员名称	描述
min_component	分量最小值。
max_component	分量最大值。
is_component_out	分量关键色在范围内或范围外。
is_component_ignore	分量是否忽略。
component_mask	分量掩码。
component_reserved	保留值。
component_reserved1	保留值。
component_reserved2	保留值。

11.18.20.9.14 hi_tde_colorkey

说明

TDE关键色的配置属性。

定义

```
typedef union {
    struct {
        hi_tde_colorkey_component alpha;
        hi_tde_colorkey_component red;
        hi_tde_colorkey_component green;
        hi_tde_colorkey_component blue;
    } argb_colorkey;
    struct {
        hi_tde_colorkey_component alpha;
        hi_tde_colorkey_component y;
        hi_tde_colorkey_component cb;
        hi_tde_colorkey_component cr;
    } ycbcr_colorkey;
    struct {
        hi_tde_colorkey_component alpha;
        hi_tde_colorkey_component clut;
    } clut_colorkey;
} hi_tde_colorkey;
```

成员

成员名称	描述
argb_colorkey.alpha	argb alpha 分量关键色属性。
argb_colorkey.red	argb red 分量关键色属性。
argb_colorkey.green	argb green 分量关键色属性。
argb _colorkey.blue	argb r blue 分量关键色属性。
ycbcr_colorkey.y	YCbCr Y 分量关键色属性。
ycbcr_colorkey.cb	YCbCr Cb 分量关键色属性。
ycbcr_colorkey.cr	YCbCr Cr 分量关键色属性。
clut_colorke.alpha	CLUT alpha 分量关键色属性。
clut_colorkey.clut	CLUT CLUT 分量关键色属性。

11.18.20.9.15 hi_tde_blend_opt

说明

模式填充混合选项结构体。

定义

```
typedef struct {
    hi_bool global_alpha_en;
```

```
    hi_bool pixel_alpha_en;
    hi_bool src1_alpha_premulti;
    hi_bool src2_alpha_premulti;
    hi_tde_blend_cmd blend_cmd;
    hi_tde_blend_mode src1_blend_mode;
    hi_tde_blend_mode src2_blend_mode;
} hi_tde_blend_opt;
```

成员

成员名称	描述
global_alpha_en	全局alpha是否使能。
pixel_alpha_en	像素alpha是否使能。
src1_alpha_premulti	src1 alpha预乘是否使能。
src2_alpha_premulti	src2 alpha预乘是否使能。
blend_cmd	Alpha 叠加指令。
src1_blend_mode	设置src1_blend模式。
src2_blend_mode	设置src2_blend模式。

11.18.20.9.16 hi_tde_alpha_blending

说明

透明度混合枚举。

定义

```
typedef enum {
    HI_TDE_ALPHA_BLENDING_NONE = 0x0,
    HI_TDE_ALPHA_BLENDING_BLEND = 0x1,
    HI_TDE_ALPHA_BLENDING_ROP = 0x2,
    HI_TDE_ALPHA_BLENDING_COLORIZE = 0x4,
    HI_TDE_ALPHA_BLENDING_MAX = 0x8
} hi_tde_alpha_blending;
```

成员

成员名称	描述
HI_TDE_ALPHA_BLENDING_NONE	不使用混合操作。
HI_TDE_ALPHA_BLENDING_BLEND	Alpha通道混合操作选项。
HI_TDE_ALPHA_BLENDING_ROP	混合ROP选项。
HI_TDE_ALPHA_BLENDING_COLORIZE	混合colorize选项。

11.18.20.9.17 hi_tde_out_alpha_from

说明

输出图像的alpha来源类型。

定义

```
typedef enum {
    HI_TDE_OUT_ALPHA_FROM_NORM = 0,    /* Output from the result of alpha blending or anti-flicker */
    HI_TDE_OUT_ALPHA_FROM_BG,          /* Output from the background bitmap */
    HI_TDE_OUT_ALPHA_FROM_FG,          /* Output from the foreground bitmap */
    HI_TDE_OUT_ALPHA_FROM_GLOBALALPHA, /* Output from the global alpha */
    HI_TDE_OUT_ALPHA_FROM_MAX
} hi_tde_out_alpha_from;
```

成员

成员名称	描述
HI_TDE_OUT_ALPHA_FROM_NORM	输出图像的alpha来源于alpha blending结果。
HI_TDE_OUT_ALPHA_FROM_BG	输出图像alpha来源于背景位图。
HI_TDE_OUT_ALPHA_FROM_FG	输出图像alpha来源于前景位图。
HI_TDE_OUT_ALPHA_FROM_GLOBALALPHA	输出图像alpha来源于全局alpha。

11.18.20.9.18 hi_tde_csc_opt

说明

CSC参数选项。

定义

```
typedef struct {
    hi_bool src_csc_user_en;          /* User-defined ICSC parameter enable */
    hi_bool src_csc_param_reload_en; /* User-defined ICSC parameter reload enable */
    hi_bool dst_csc_user_en;          /* User-defined OCSC parameter enable */
    hi_bool dst_csc_param_reload_en; /* User-defined OCSC parameter reload enable */
    hi_u64 src_csc_param_addr;        /* ICSC parameter address. The address must be 128-bit aligned. */
    hi_s32 src_csc_param_len;
    hi_u64 dst_csc_param_addr; /* OCSC parameter address. The address must be 128-bit aligned. */
    hi_s32 dst_csc_param_len;
} hi_tde_csc_opt;
```

成员

成员名称	描述
src_csc_user_en	用户自定义ICSC参数使能。
src_csc_param_reload_en	重新加载用户自定义ICSC参数使能。
dst_csc_user_en	用户自定义OCSC参数使能。
dst_csc_param_reload_en	重新加载用户自定义OCSC参数使能。
src_csc_param_addr	ICSC参数地址，要求128bit对齐。
src_csc_param_len	ICSC参数长度。
dst_csc_param_addr	OCSC参数地址，要求128bit对齐。
dst_csc_param_len	OCSC参数长度。

11.18.20.9.19 hi_tde_pattern_fill_opt

说明

模式填充选项操作，在模式填充功能结构中使用。

定义

```
typedef struct {
    hi_tde_alpha_blending alpha_blending_cmd;
    hi_tde_rop_mode rop_color;
    hi_tde_rop_mode rop_alpha;
    hi_tde_colorkey_mode colorkey_mode;
    hi_tde_colorkey colorkey_value;
    hi_tde_clip_mode clip_mode;
    hi_tde_rect clip_rect;
    hi_bool clut_reload;
    hi_u8 global_alpha;
    hi_tde_out_alpha_from out_alpha_from;
    hi_u32 color_resize;
    hi_tde_blend_opt blend_opt;
    hi_tde_csc_opt csc_opt;
} hi_tde_pattern_fill_opt;
```

成员

成员名称	描述
alpha_blending_cmd	逻辑操作类型。
rop_color	颜色空间的rop类型。
rop_alpha	alpha分量的rop类型。
colorkey_mode	Colorkey模式。
colorkey_value	Colorkey值。

成员名称	描述
clip_mode	裁剪模式。
clip_rect	裁剪区域。
clut_reload	是否重载clut。
global_alpha	全局alpha值。
out_alpha_from	输出alpha的源。
color_resize	Colorize的值。
blend_opt	混合选项。
csc_opt	csc选项。

11.18.20.10 HIFB 叠加图形层管理

11.18.20.10.1 fb_var_screeninfo

说明

定义HIFB的可变屏幕信息，此为Linux FrameBuffer原生结构体。

定义

```
struct fb_var_screeninfo {
    __u32 xres;             /* visible resolution       */
    __u32 yres;
    __u32 xres_virtual;     /* virtual resolution       */
    __u32 yres_virtual;
    __u32 xoffset;          /* offset from virtual to visible */
    __u32 yoffset;          /* resolution               */
    __u32 bits_per_pixel;
    __u32 grayscale;        /* 0 = color, 1 = grayscale,    */
                            /* >1 = FOURCC              */
    struct fb_bitfield red;     /* bitfield in fb mem if true color, */
    struct fb_bitfield green;   /* else only length is significant */
    struct fb_bitfield blue;
    struct fb_bitfield transp;  /* transparency         */
    __u32 nonstd;           /* != 0 Non standard pixel format */
    __u32 activate;         /* see FB_ACTIVATE_*        */
    __u32 height;           /* height of picture in mm    */
    __u32 width;            /* width of picture in mm     */
    __u32 accel_flags;      /* (OBSOLETE) see fb_info.flags */

    /* Timing: All values in pixclocks, except pixclock (of course) */
    __u32 pixclock;         /* pixel clock in ps (pico seconds) */
    __u32 left_margin;      /* time from sync to picture    */
    __u32 right_margin;     /* time from picture to sync    */
    __u32 upper_margin;     /* time from sync to picture    */
    __u32 lower_margin;
    __u32 hsync_len;        /* length of horizontal sync    */
    __u32 vsync_len;        /* length of vertical sync  */
    __u32 sync;             /* see FB_SYNC_*        */
    __u32 vmode;            /* see FB_VMODE_*       */
    __u32 rotate;           /* angle we rotate counter clockwise */
    __u32 colorspace;       /* colorspace for FOURCC-based modes */
```

```
    __u32 reserved[4];      /* Reserved for future compatibility */
};
```

成员

G0、G1图层，下表中的Gx_min、Gy_min为32；G2、G3、G4图层，下表中的Gx_min、Gy_min为8。Gx_min表示当前图层支持的最小宽度，Gy_min表示当前图层支持的最小高度。

成员名称	描述
xres	可见屏幕宽度（像素数），需要为偶数。 取值范围：[Gx_min, xres_virtual]。
yres	可见屏幕高度（像素数），需要为偶数。 取值范围：[Gy_min, yres_virtual]。
xres_virtual	虚拟屏幕宽度（显存中图像宽度）。 取值范围：大于或等于Gx_min。
yres_virtual	虚拟屏幕高度（显存中图像高度）。 取值范围：大于或等于Gy_min。
xoffset	在x方向上的偏移像素数。 取值范围：[0, xres_virtual - xres]。
yoffset	在y方向上的偏移像素数。 取值范围：[0, yres_virtual - yres]
bits_per_pixel	每个像素所占的比特数。 取值范围：32/16/4/2 注：4和2分别对应CLUT4和CLUT2模式，需要使能CMAP功能。
red	颜色分量中红色的位域信息。
green	颜色分量中绿色的位域信息。
blue	颜色分量中蓝色的位域信息。
transp	颜色分量中alpha分量的位域信息。
grayscale	颜色模式，当前版本不支持。
nonstd	是否为标准像素格式，当前版本不支持。 ● 0：是 ● 1：否
activate	设置生效的时刻，当前版本不支持。
height	屏幕高，单位为mm，当前版本不支持。
width	屏幕宽，单位为mm，当前版本不支持。
accel_flags	加速标志，当前版本不支持。

成员名称	描述
pixclock	显示一个点需要的时间，单位为ns，当前版本不支持。
left_margin	左消隐信号，当前版本不支持。
right_margin	右消隐信号，当前版本不支持。
upper_margin	上消隐信号，当前版本不支持。
lower_margin	下消隐信号，当前版本不支持。
hsync_len	水平同步时长，当前版本不支持。
vsync_len	垂直同步时长，当前版本不支持。
sync	同步信号方式，当前版本不支持。
vmode	扫描模式，当前版本不支持。
rotate	顺时针旋转的角度，当前版本不支持。
colorspace	FOURCC模式的色彩空间， 当前版本不支持。
reserved[4]	预留参数，当前版本不支持。

11.18.20.10.2 fb_fix_screeninfo

说明

定义HIFB的固定屏幕信息，此为Linux FrameBuffer原生结构体。

定义

```
struct fb_fix_screeninfo {
    char id[16];            /* identification string eg "TT Builtin" */
    unsigned long smem_start;   /* Start of frame buffer mem */
                            /* (physical address) */
    __u32 smem_len;             /* Length of frame buffer mem */
    __u32 type;             /* see FB_TYPE_*        */
    __u32 type_aux;         /* Interleave for interleaved Planes */
    __u32 visual;           /* see FB_VISUAL_*      */
    __u16 xpanstep;         /* zero if no hardware panning  */
    __u16 ypanstep;         /* zero if no hardware panning  */
    __u16 ywrapstep;        /* zero if no hardware ywrap    */
    __u32 line_length;      /* length of a line in bytes    */
    unsigned long mmio_start;   /* Start of Memory Mapped I/O   */
                        /* (physical address) */
    __u32 mmio_len;         /* Length of Memory Mapped I/O  */
    __u32 accel;            /* Indicate to driver which */
                        /*  specific chip/card we have  */
    __u16 capabilities;     /* see FB_CAP_*         */
    __u16 reserved[2];      /* Reserved for future compatibility */
};
```

成员

成员名称	描述
id[16]	设备驱动名称。
smem_start	显存起始内核态虚拟地址。
smem_len	显存大小，单位：字节。
type	显存类型，当前版本不支持。
type_aux	附加类型，当前版本不支持。
visual	色彩模式，当前版本不支持。
xpanstep	支持水平方向上的PAN显示： ● 0：不支持。 ● 非0：支持，此时该值用于表示在水平方向上每步进的像素值。 当前版本不支持。
ypanstep	支持垂直方向上的PAN显示： ● 0：不支持。 ● 非0：支持，此时该值用于表示在垂直方向上每步进的像素值。 当前版本不支持。
ywrapstep	类似于ypanstep，不同之处在于：当起显示当底部时，能回到显存的开始处进行显示。 当前版本不支持。
line_length	每行字节数，当前版本不支持。
mmio_start	显存起始用户态虚拟地址。
mmio_len	显存大小，单位：字节。
accel	显示所支持的硬件加速设备，当前版本不支持。
capabilities	预留参数，当前版本不支持。
reserved[2]	预留参数，当前版本不支持。

11.18.20.10.3 fb_cmap

说明

定义HIFB的颜色表信息，此为Linux FrameBuffer原生结构体。

定义

```
struct fb_cmap {
    __u32 start;          /* First entry  */
```

```
    __u32 len;            /* Number of entries */
    __u16 *red;           /* Red values   */
    __u16 *green;
    __u16 *blue;
    __u16 *transp;        /* transparency, can be NULL */
};
```

成员

成员名称	描述
start	颜色表入口索引。
len	颜色表长度（单分量颜色表数量）。 最大长度为256。
red	红色分量表。
green	绿色分量表。
blue	蓝色分量表。
transp	Alpha分量表。

11.18.20.10.4 hi_fb_point

说明

定义HIFB的起始坐标信息。

定义

```
typedef struct {
    hi_s32 x_pos;
    hi_s32 y_pos;
} hi_fb_point;
```

成员

成员名称	描述
x_pos	起始点水平坐标。
y_pos	起始点垂直坐标。

11.18.20.10.5 hi_fb_alpha

说明

定义HIFB Alpha结构体。

定义

```
typedef struct {
    hi_bool alpha_en;
```

```
    hi_bool alpha_chn_en;
    hi_u8 alpha0;
    hi_u8 alpha1;
    hi_u8 global_alpha;
    hi_u8 reserved;
} hi_fb_alpha;
```

成员

成员名称	描述
alpha_en	Alpha叠加使能。
alpha_chn_en	Alpha通道叠加使能。
alpha0	Alpha通道为0时选取的Alpha值，当前版本不支持。
alpha1	Alpha通道为1时选取的Alpha值，当前版本不支持。
global_alpha	全局Alpha值，范围为0～255，默认为255。在Alpha通道使能时起作用。
reserved	预留参数，当前版本不支持。

11.18.20.10.6 fb_bitfield

说明

位域信息，用于设置像素格式，此为Linux FrameBuffer原生结构体。

定义

```
struct fb_bitfield {
    __u32 offset;           /* beginning of bitfield    */
    __u32 length;           /* length of bitfield       */
    __u32 msb_right;        /* != 0 : Most significant bit is */
                            /* right */
};
```

成员

成员名称	描述
offset	颜色分量的偏移。
length	颜色分量所占bit位数，
msb_right	● 0：最高位在左侧。 ● 非0：最高位在右侧。

11.18.20.11 HDMI 外设

11.18.20.11.1 hi_hdmi_id

说明

定义HDMI设备号。

定义

```
typedef enum {
    HI_HDMI_ID_0,
    HI_HDMI_ID_1,
    HI_HDMI_ID_BUTT
} hi_hdmi_id;
```

成员

成员名称	描述
HI_HDMI_ID_0	HDMI设备0。
HI_HDMI_ID_1	HDMI设备1。
HI_HDMI_ID_BUTT	预留值。

11.18.20.11.2 hi_hdmi_video_format

说明

定义HDMI视频格式。

定义

```
typedef enum {
    HI_HDMI_VIDEO_FORMAT_1080P_60,
    HI_HDMI_VIDEO_FORMAT_1080P_50,
    HI_HDMI_VIDEO_FORMAT_1080P_30,
    HI_HDMI_VIDEO_FORMAT_1080P_25,
    HI_HDMI_VIDEO_FORMAT_1080P_24,
    HI_HDMI_VIDEO_FORMAT_1080i_60,
    HI_HDMI_VIDEO_FORMAT_1080i_50,
    HI_HDMI_VIDEO_FORMAT_720P_60,
    HI_HDMI_VIDEO_FORMAT_720P_50,
    HI_HDMI_VIDEO_FORMAT_576P_50,
    HI_HDMI_VIDEO_FORMAT_480P_60,
    HI_HDMI_VIDEO_FORMAT_PAL,
    HI_HDMI_VIDEO_FORMAT_NTSC,
    HI_HDMI_VIDEO_FORMAT_861D_640X480_60,
    HI_HDMI_VIDEO_FORMAT_VESA_800X600_60,
    HI_HDMI_VIDEO_FORMAT_VESA_1024X768_60,
    HI_HDMI_VIDEO_FORMAT_VESA_1280X800_60,
    HI_HDMI_VIDEO_FORMAT_VESA_1280X1024_60,
    HI_HDMI_VIDEO_FORMAT_VESA_1366X768_60,
    HI_HDMI_VIDEO_FORMAT_VESA_1440X900_60,
    HI_HDMI_VIDEO_FORMAT_VESA_1400X1050_60,
    HI_HDMI_VIDEO_FORMAT_VESA_1600X1200_60,
    HI_HDMI_VIDEO_FORMAT_VESA_1680X1050_60,
    HI_HDMI_VIDEO_FORMAT_VESA_1920X1200_60,
    HI_HDMI_VIDEO_FORMAT_2560x1440_30,
    HI_HDMI_VIDEO_FORMAT_2560x1440_60,
    HI_HDMI_VIDEO_FORMAT_2560x1600_60,
```

```
    HI_HDMI_VIDEO_FORMAT_1920x2160_30,
    HI_HDMI_VIDEO_FORMAT_3840X2160P_24,
    HI_HDMI_VIDEO_FORMAT_3840X2160P_25,
    HI_HDMI_VIDEO_FORMAT_3840X2160P_30,
    HI_HDMI_VIDEO_FORMAT_3840X2160P_50,
    HI_HDMI_VIDEO_FORMAT_3840X2160P_60,
    HI_HDMI_VIDEO_FORMAT_4096X2160P_24,
    HI_HDMI_VIDEO_FORMAT_4096X2160P_25,
    HI_HDMI_VIDEO_FORMAT_4096X2160P_30,
    HI_HDMI_VIDEO_FORMAT_4096X2160P_50,
    HI_HDMI_VIDEO_FORMAT_4096X2160P_60,
    HI_HDMI_VIDEO_FORMAT_3840X2160P_120,
    HI_HDMI_VIDEO_FORMAT_4096X2160P_120,
    HI_HDMI_VIDEO_FORMAT_7680X4320P_30,
    HI_HDMI_VIDEO_FORMAT_VESA_CUSTOMER_DEFINE,
    HI_HDMI_VIDEO_FORMAT_BUTT
} hi_hdmi_video_format;
```

成员

成员名称	描述
HI_HDMI_VIDEO_FORMAT_1080P_60	1080P60帧。
HI_HDMI_VIDEO_FORMAT_1080P_50	1080P50帧。
HI_HDMI_VIDEO_FORMAT_1080P_30	1080P 30帧。
HI_HDMI_VIDEO_FORMAT_1080P_25	1080P 25帧。
HI_HDMI_VIDEO_FORMAT_1080P_24	1080P 24帧。
HI_HDMI_VIDEO_FORMAT_1080i_60	1080I 60帧。
HI_HDMI_VIDEO_FORMAT_1080i_50	1080I 50帧。
HI_HDMI_VIDEO_FORMAT_720P_60	720P 60帧。
HI_HDMI_VIDEO_FORMAT_720P_50	720P 50帧。
HI_HDMI_VIDEO_FORMAT_576P_50	576P 50帧。
HI_HDMI_VIDEO_FORMAT_480P_60	480P 60帧。
HI_HDMI_VIDEO_FORMAT_PAL	PAL（Phase Alternating Line）制式。
HI_HDMI_VIDEO_FORMAT_NTSC	NTSC（National Television System Committee）制式。

成员名称	描述
HI_HDMI_VIDEO_FORMAT_861D_640X480_60	640x480 60帧 。
HI_HDMI_VIDEO_FORMAT_VESA_800X600_60	800x600 60帧 。
HI_HDMI_VIDEO_FORMAT_VESA_1024X768_60	1024x768 60帧。
HI_HDMI_VIDEO_FORMAT_VESA_1280X800_60	1280x800 60帧。
HI_HDMI_VIDEO_FORMAT_VESA_1280X1024_60	1280x1024 60帧。
HI_HDMI_VIDEO_FORMAT_VESA_1366X768_60	1366x768 60帧。
HI_HDMI_VIDEO_FORMAT_VESA_1440X900_60	1440x900 60帧。
HI_HDMI_VIDEO_FORMAT_VESA_1400X1050_60	1400x1050 60帧。
HI_HDMI_VIDEO_FORMAT_VESA_1600X1200_60	1600x1200 60帧。
HI_HDMI_VIDEO_FORMAT_VESA_1680X1050_60	1680x1050 60帧。
HI_HDMI_VIDEO_FORMAT_VESA_1920X1200_60	1920x1200 60帧。
HI_HDMI_VIDEO_FORMAT_2560x1440_30	2560x1440 30帧。
HI_HDMI_VIDEO_FORMAT_2560x1440_60	2560x1440 60帧。
HI_HDMI_VIDEO_FORMAT_2560x1600_60	2560x1600 30帧。
HI_HDMI_VIDEO_FORMAT_1920x2160_30	1920x2160 30帧。
HI_HDMI_VIDEO_FORMAT_3840X2160P_24	3840x2160 24帧。
HI_HDMI_VIDEO_FORMAT_3840X2160P_25	3840x2160 25帧。
HI_HDMI_VIDEO_FORMAT_3840X2160P_30	3840x2160 30帧。
HI_HDMI_VIDEO_FORMAT_3840X2160P_50	3840x2160 50帧。

成员名称	描述
HI_HDMI_VIDEO_FORMAT_3840X2160P_60	3840x2160 60帧。
HI_HDMI_VIDEO_FORMAT_4096X2160P_24	4096x2160 24帧。
HI_HDMI_VIDEO_FORMAT_4096X2160P_25	4096x2160 25帧。
HI_HDMI_VIDEO_FORMAT_4096X2160P_30	4096x2160 30帧。
HI_HDMI_VIDEO_FORMAT_4096X2160P_50	4096x2160 50帧。
HI_HDMI_VIDEO_FORMAT_4096X2160P_60	4096x2160 60帧。
HI_HDMI_VIDEO_FORMAT_3840X2160P_120	3840X2160 120帧。
HI_HDMI_VIDEO_FORMAT_4096X2160P_120	4096x2160 120帧。
HI_HDMI_VIDEO_FORMAT_7680X4320P_30	7680x4320 30帧。
HI_HDMI_VIDEO_FORMAT_VESA_CUSTOMER_DEFINE	用户配置规格。
HI_HDMI_VIDEO_FORMAT_BUTT	预留值。

11.18.20.11.3 hi_hdmi_deep_color

说明

定义HDMI深色模式。

定义

```
typedef enum {
    HI_HDMI_DEEP_COLOR_24BIT,
    HI_HDMI_DEEP_COLOR_30BIT,
    HI_HDMI_DEEP_COLOR_36BIT,
    HI_HDMI_DEEP_COLOR_BUTT
} hi_hdmi_deep_color;
```

成员

成员名称	描述
HI_HDMI_DEEP_COLOR_24BIT	HDMI Deep Color 24bit 模式
HI_HDMI_DEEP_COLOR_30BIT	HDMI Deep Color 30bit 模式。
HI_HDMI_DEEP_COLOR_36BIT	HDMI Deep Color 36bit 模式。
HI_HDMI_DEEP_COLOR_BUTT	预留值。

11.18.20.11.4 hi_hdmi_sample_rate

说明

定义 HDMI 音频输出采样率。

定义

```
typedef enum {
    HI_HDMI_SAMPLE_RATE_UNKNOWN,
    HI_HDMI_SAMPLE_RATE_8K,
    HI_HDMI_SAMPLE_RATE_11K,
    HI_HDMI_SAMPLE_RATE_12K,
    HI_HDMI_SAMPLE_RATE_16K,
    HI_HDMI_SAMPLE_RATE_22K,
    HI_HDMI_SAMPLE_RATE_24K,
    HI_HDMI_SAMPLE_RATE_32K,
    HI_HDMI_SAMPLE_RATE_44K,
    HI_HDMI_SAMPLE_RATE_48K,
    HI_HDMI_SAMPLE_RATE_88K,
    HI_HDMI_SAMPLE_RATE_96K,
    HI_HDMI_SAMPLE_RATE_176K,
    HI_HDMI_SAMPLE_RATE_192K,
    HI_HDMI_SAMPLE_RATE_768K,
    HI_HDMI_SAMPLE_RATE_BUTT
} hi_hdmi_sample_rate;
```

成员

成员名称	描述
HI_HDMI_SAMPLE_RATE_8K	8K采样率。
HI_HDMI_SAMPLE_RATE_11K	11K采样率。
HI_HDMI_SAMPLE_RATE_12K	12K采样率。

成员名称	描述
HI_HDMI_SAMPLE_RATE_16K	16K采样率。
HI_HDMI_SAMPLE_RATE_22K	22K采样率。
HI_HDMI_SAMPLE_RATE_24K	24K采样率。
HI_HDMI_SAMPLE_RATE_32K	32K采样率。
HI_HDMI_SAMPLE_RATE_44K	44K采样率。
HI_HDMI_SAMPLE_RATE_48K	48K采样率
HI_HDMI_SAMPLE_RATE_88K	88K采样率。
HI_HDMI_SAMPLE_RATE_96K	96K采样率。
HI_HDMI_SAMPLE_RATE_176K	176K采样率。
HI_HDMI_SAMPLE_RATE_192K	192K采样率。
HI_HDMI_SAMPLE_RATE_768K	768K采样率。
HI_HDMI_SAMPLE_RATE_BUTT	预留值

11.18.20.11.5 hi_hdmi_bit_depth

说明

定义HDMI音频输出采样位宽。

定义

```
typedef enum {
    HI_HDMI_BIT_DEPTH_UNKNOWN,
    HI_HDMI_BIT_DEPTH_8,
    HI_HDMI_BIT_DEPTH_16,
    HI_HDMI_BIT_DEPTH_18,
    HI_HDMI_BIT_DEPTH_20,
    HI_HDMI_BIT_DEPTH_24,
    HI_HDMI_BIT_DEPTH_32,
    HI_HDMI_BIT_DEPTH_BUTT
} hi_hdmi_bit_depth;
```

成员

成员名称	描述
HI_HDMI_BIT_DEPTH_8	8bit采样位宽。
HI_HDMI_BIT_DEPTH_16	16bit采样位宽
HI_HDMI_BIT_DEPTH_18	18bit采样位宽。
HI_HDMI_BIT_DEPTH_20	20bit采样位宽。
HI_HDMI_BIT_DEPTH_24	24bit采样位宽。
HI_HDMI_BIT_DEPTH_32	32bit采样位宽。
HI_HDMI_BIT_DEPTH_BUTT	预留值。

11.18.20.11.6 hi_hdmi_attr

说明

定义HDMI输出属性结构体。

定义

```
typedef struct {
    hi_bool hdmi_en;
    hi_hdmi_video_format video_format;
    hi_hdmi_deep_color deep_color_mode;
    hi_bool audio_en;
    hi_hdmi_sample_rate sample_rate;
    hi_hdmi_bit_depth bit_depth;
    hi_bool auth_mode_en;
    hi_bool deep_color_adapt_en;
    hi_u32 pix_clk;
} hi_hdmi_attr;
```

成员

成员名称	描述
hdmi_en	是否强制HDMI输出。 ● HI_TRUE：强制HDMI输出 ● HI_FALSE：DVI输出 当前版本需设置为HI_TRUE
video_format	视频制式。建议用户设置为Sink能力集支持的制式。当前版本中此参数仅支持设置为HI_HDMI_VIDEO_FORMAT_VESA_CUSTOMER_DEFINE

成员名称	描述
deep_color_mode	DeepColor输出模式。 默认为HI_HDMI_DEEP_COLOR_24BIT（HI_HDMI_DEEP_COLOR_OFF）。 部分Sink 不支持HI_HDMI_DEEP_COLOR_30BIT 和HI_HDMI_DEEP_COLOR_36BIT，设置此DeepColor可能引起异常，需要设置为默认值。
audio_en	是否使能音频：使用音频输出时必须设置为HI_TRUE;
sample_rate	音频采样率，此参数需要与AO的配置保持一致。当前版本目前不支持除48KHz 外的采样率，同时建议用户设置为Sink能力集支持的采样率。
bit_depth	音频位宽，默认为16，当前版本不支持更改为非默认值（此参数需要与AO的配置保持一致）。
auth_mode_en	使能该模式，HDMI强制输出，不再去根据显示或认证设备的EDID信息来自适应调整，主要针对认证场景。 ● HI_TRUE：使能bAuthMode ● HI_FALSE：不能使bAuthMode 当前版本不支持更改此参数为非默认值。
deep_color_adapt_en	当前版本中此参数仅支持设置为：HI_TRUE。
pix_clk	像素时钟，此参数需要与VO配置的输入时序制式对应的像素时钟保持一致，具体值可通过查询VESA对应标准获取。 当前版本不支持更改此参数为与VO输入时序对应的像素时钟不一致的值。

11.18.20.11.7 hi_hdmi_infoframe_type

说明

定义HDMI信息帧类型。

定义

```
typedef enum {
    HI_INFOFRAME_TYPE_AVI,
    HI_INFOFRAME_TYPE_AUDIO,
    HI_INFOFRAME_TYPE_VENDORSPEC,
    HI_INFOFRAME_TYPE_BUTT
} hi_hdmi_infoframe_type;
```

成员

成员名称	描述
HI_INFOFRAME_TYPE_AVI	显示信息帧类型。

成员名称	描述
HI_INFOFRAME_TYPE_AUDIO	音频信息帧类型。
HI_INFOFRAME_TYPE_VENDORSPEC	供应商定制信息帧类型。
HI_INFOFRAME_TYPE_BUTT	预留值。

说明

当前版本对该结构体的部分参数赋值存在如上表所述的限制，用例中需要按照上述限制配置相关接口的入参值，否则属于未定义行为，影响送显、音频输出。

11.18.20.11.8 hi_hdmi_color_space

说明

定义HDMI颜色空间。

定义

```
typedef enum {
    HI_HDMI_COLOR_SPACE_RGB444,
    HI_HDMI_COLOR_SPACE_YCBCR422,
    HI_HDMI_COLOR_SPACE_YCBCR444,
    HI_HDMI_COLOR_SPACE_YCBCR420,
    HI_HDMI_COLOR_SPACE_BUTT
} hi_hdmi_color_space;
```

成员

成员名称	描述
HI_HDMI_COLOR_SPACE_RGB444	RGB444。
HI_HDMI_COLOR_SPACE_YCBCR422	YUV422。
HI_HDMI_COLOR_SPACE_YCBCR444	YUV444
HI_HDMI_COLOR_SPACE_YCBCR420	YUV420。
HI_HDMI_COLOR_SPACE_BUTT	预留值。

说明

当前版本HDMI输入来自VO，VO输出格式仅支持YUV444。

11.18.20.11.9 hi_hdmi_bar_info

说明

定义HDMI面板信息。

定义

```
typedef enum {
    HI_HDMI_BAR_INFO_NOT_VALID,
    HI_HDMI_BAR_INFO_V,
    HI_HDMI_BAR_INFO_H,
    HI_HDMI_BAR_INFO_VH,
    HI_HDMI_BAR_INFO_BUTT
} hi_hdmi_bar_info;
```

成员

成员名称	描述
HI_HDMI_BAR_INFO_NOT_VALID	当前版本不支持HDMI 面板信息配置，需配置此项。
HI_HDMI_BAR_INFO_V	预留值。
HI_HDMI_BAR_INFO_H	预留值。
HI_HDMI_BAR_INFO_VH	预留值。
HI_HDMI_BAR_INFO_BUTT	预留值。

11.18.20.11.10 hi_hdmi_scan_info

说明

定义HDMI扫描信息。

定义

```
typedef enum {
    HI_HDMI_SCAN_INFO_NO_DATA,
    HI_HDMI_SCAN_INFO_OVERSCANNED,
    HI_HDMI_SCAN_INFO_UNDERSCANNED,
    HI_HDMI_SCAN_INFO_BUTT
} hi_hdmi_scan_info;
```

成员

成员名称	描述
HI_HDMI_SCAN_INFO_NO_DATA	当前版本不支持HDMI 扫描信息配置，需配置此项。
HI_HDMI_SCAN_INFO_OVERSCANNED	预留值。
HI_HDMI_SCAN_INFO_UNDERSCANNED	预留值。
HI_HDMI_SCAN_INFO_BUTT	预留值。

11.18.20.11.11 hi_hdmi_colorimetry

说明

定义HDMI音频输出信号色域标准。

定义

```
typedef enum {
    HI_HDMI_COMMON_COLORIMETRY_NO_DATA,
    HI_HDMI_COMMON_COLORIMETRY_ITU601,
    HI_HDMI_COMMON_COLORIMETRY_ITU709,
    HI_HDMI_COMMON_COLORIMETRY_BUTT
} hi_hdmi_colorimetry;
```

成员

成员名称	描述
HI_HDMI_COMMON_COLORIMETRY_NO_DATA	不配置色域。
HI_HDMI_COMMON_COLORIMETRY_ITU601	国际电信联盟601标准。
HI_HDMI_COMMON_COLORIMETRY_ITU709	国际电信联盟709标准。
HI_HDMI_COMMON_COLORIMETRY_BUTT	预留值。

11.18.20.11.12 hi_hdmi_ex_colorimetry

说明

定义HDMI音频输出信号色域标准扩展。

定义

```
typedef enum {
    HI_HDMI_COMMON_COLORIMETRY_XVYCC_601,
    HI_HDMI_COMMON_COLORIMETRY_XVYCC_709,
    HI_HDMI_COMMON_COLORIMETRY_S_YCC_601,
    HI_HDMI_COMMON_COLORIMETRY_ADOBE_YCC_601,
    HI_HDMI_COMMON_COLORIMETRY_ADOBE_RGB,
    HI_HDMI_COMMON_COLORIMETRY_2020_CONST_LUMINOUS,
    HI_HDMI_COMMON_COLORIMETRY_2020_NON_CONST_LUMINOUS,
    HI_HDMI_COMMON_COLORIMETRY_EXT_BUTT
} hi_hdmi_ex_colorimetry;
```

成员

成员名称	描述
HI_HDMI_COMMON_COLORIMETRY_XVYCC_601	新一代601色域标准扩展
HI_HDMI_COMMON_COLORIMETRY_XVYCC_709	预留值。
HI_HDMI_COMMON_COLORIMETRY_S_YCC_601	预留值。
HI_HDMI_COMMON_COLORIMETRY_ADOBE_YCC_601	预留值。
HI_HDMI_COMMON_COLORIMETRY_ADOBE_RGB	预留值。
HI_HDMI_COMMON_COLORIMETRY_2020_CONST_LUMINOUS	预留值。
HI_HDMI_COMMON_COLORIMETRY_2020_NON_CONST_LUMINOUS	预留值。
HI_HDMI_COMMON_COLORIMETRY_EXT_BUTT	预留值。

11.18.20.11.13 hi_pic_aspect_ratio

说明

定义HDMI图像宽高比。

定义

```
typedef enum {
    HI_HDMI_PIC_ASPECT_RATIO_NO_DATA,
    HI_HDMI_PIC_ASPECT_RATIO_4TO3,
    HI_HDMI_PIC_ASPECT_RATIO_16TO9,
    HI_HDMI_PIC_ASPECT_RATIO_64TO27,
    HI_HDMI_PIC_ASPECT_RATIO_256TO135,
```

```
    HI_HDMI_PIC_ASPECT_RATIO_BUTT
} hi_pic_aspect_ratio;
```

成员

成员名称	描述
HI_HDMI_PIC_ASPECT_RATIO_4TO3	宽高比4：3。
HI_HDMI_PIC_ASPECT_RATIO_16TO9	宽高比16：9。
HI_HDMI_PIC_ASPECT_RATIO_64TO27	宽高比64：27。
HI_HDMI_PIC_ASPECT_RATIO_256TO135	宽高比256：135。
HI_HDMI_PIC_ASPECT_RATIO_BUTT	预留值。

11.18.20.11.14 hi_hdmi_active_aspect_ratio

说明

定义HDMI实际图像有效宽高比。

定义

```
typedef enum {
    HI_HDMI_ACTIVE_ASPECT_RATIO_16TO9_TOP = 2,
    HI_HDMI_ACTIVE_ASPECT_RATIO_14TO9_TOP,
    HI_HDMI_ACTIVE_ASPECT_RATIO_16TO9_BOX_CENTER,
    HI_HDMI_ACTIVE_ASPECT_RATIO_SAME_PIC = 8,
    HI_HDMI_ACTIVE_ASPECT_RATIO_4TO3_CENTER,
    HI_HDMI_ACTIVE_ASPECT_RATIO_16TO9_CENTER,
    HI_HDMI_ACTIVE_ASPECT_RATIO_14TO9_CENTER,
    HI_HDMI_ACTIVE_ASPECT_RATIO_4TO3_14_9 = 13,
    HI_HDMI_ACTIVE_ASPECT_RATIO_16TO9_14_9,
    HI_HDMI_ACTIVE_ASPECT_RATIO_16TO9_4_3,
    HI_HDMI_ACTIVE_ASPECT_RATIO_BUTT
} hi_hdmi_active_aspect_ratio;
```

成员

成员名称	描述
HI_HDMI_ACTIVE_ASPECT_RATIO_16TO9_TOP	实际图像宽高比16：9，顶部区域。。
HI_HDMI_ACTIVE_ASPECT_RATIO_14TO9_TOP	实际图像宽高比14：9，顶部区域。

成员名称	描述
HI_HDMI_ACTIVE_ASPECT_RATIO_16TO9_BOX_CENTER	实际图像宽高比16：9，盒型中心区域。
HI_HDMI_ACTIVE_ASPECT_RATIO_SAME_PIC	实际图像宽高比与图形一致。
HI_HDMI_ACTIVE_ASPECT_RATIO_4TO3_CENTER	实际图像宽高比4：3，中心区域。
HI_HDMI_ACTIVE_ASPECT_RATIO_16TO9_CENTER	实际图像宽高比16：9，中心区域。
HI_HDMI_ACTIVE_ASPECT_RATIO_14TO9_CENTER	实际图像宽高比14：9，中心区域。
HI_HDMI_ACTIVE_ASPECT_RATIO_4TO3_14_9	实际图像宽高比4：3，拉升为14：9。
HI_HDMI_ACTIVE_ASPECT_RATIO_16TO9_14_9	实际图像宽高比16：9，拉升为14：9。
HI_HDMI_ACTIVE_ASPECT_RATIO_16TO9_4_3	实际图像宽高比16：9，拉升为4：3。
HI_HDMI_ACTIVE_ASPECT_RATIO_BUTT	预留值。

11.18.20.11.15 hi_hdmi_pic_scaline

说明

定义HDMI图像扫描信息。

定义

```
typedef enum {
    HI_HDMI_PIC_NON_UNIFORM_SCALING,
    HI_HDMI_PIC_SCALING_H,
    HI_HDMI_PIC_SCALING_V,
    HI_HDMI_PIC_SCALING_HV,
    HI_HDMI_PIC_SCALING_BUTT
} hi_hdmi_pic_scaline;
```

成员

成员名称	描述
HI_HDMI_PIC_NON_UNIFORM_SCALING	不做特殊扫描处理。
HI_HDMI_PIC_SCALING_H	水平扫描。

成员名称	描述
HI_HDMI_PIC_SCALING_V	垂直扫描。
HI_HDMI_PIC_SCALING_HV	水平垂直同时扫描。
HI_HDMI_PIC_SCALING_BUTT	预留值。

11.18.20.11.16 hi_hdmi_rgb_quant_range

说明

定义HDMI RGB量化范围。

定义

```
typedef enum {
    HI_HDMI_RGB_QUANT_DEFAULT_RANGE
    HI_HDMI_RGB_QUANT_LIMITED_RANGE
    HI_HDMI_RGB_QUANT_FULL_RANGE
    HI_HDMI_RGB_QUANT_FULL_BUTT
} hi_hdmi_rgb_quant_range;
```

成员

成员名称	描述
HI_HDMI_RGB_QUANT_DEFAULT_RANGE	RGB默认范围。
HI_HDMI_RGB_QUANT_LIMITED_RANGE	RGB有限范围。
HI_HDMI_RGB_QUANT_FULL_RANGE	RGB全量范围。
HI_HDMI_RGB_QUANT_FULL_BUTT	预留值。

11.18.20.11.17 hi_hdmi_pixel_repetition

说明

定义HDMI像素复制次数。

定义

```
typedef enum {
    HI_HDMI_PIXEL_REPET_NO,
    HI_HDMI_PIXEL_REPET_2_TIMES,
    HI_HDMI_PIXEL_REPET_3_TIMES,
```

```
    HI_HDMI_PIXEL_REPET_4_TIMES,
    HI_HDMI_PIXEL_REPET_5_TIMES,
    HI_HDMI_PIXEL_REPET_6_TIMES,
    HI_HDMI_PIXEL_REPET_7_TIMES,
    HI_HDMI_PIXEL_REPET_8_TIMES,
    HI_HDMI_PIXEL_REPET_9_TIMES,
    HI_HDMI_PIXEL_REPET_10_TIMES,
    HI_HDMI_PIXEL_REPET_BUTT
} hi_hdmi_pixel_repetition;
```

成员

成员名称	描述
HI_HDMI_PIXEL_REPET_NO	不做像素复制。
HI_HDMI_PIXEL_REPET_2_TIMES	像素复制2次。
HI_HDMI_PIXEL_REPET_3_TIMES	像素复制3次。
HI_HDMI_PIXEL_REPET_4_TIMES	像素复制4次。
HI_HDMI_PIXEL_REPET_5_TIMES	像素复制5次。
HI_HDMI_PIXEL_REPET_6_TIMES	像素复制6次。
HI_HDMI_PIXEL_REPET_7_TIMES	像素复制7次。
HI_HDMI_PIXEL_REPET_8_TIMES	像素复制8次。
HI_HDMI_PIXEL_REPET_9_TIMES	像素复制9次。
HI_HDMI_PIXEL_REPET_10_TIMES	像素复制10次。
HI_HDMI_PIXEL_REPET_BUTT	预留值。

11.18.20.11.18 hi_hdmi_content_type

说明

定义HDMI内容信息。

定义

```
typedef enum {
    HI_HDMI_CONTNET_GRAPHIC,
```

```
    HI_HDMI_CONTNET_PHOTO,
    HI_HDMI_CONTNET_CINEMA,
    HI_HDMI_CONTNET_GAME,
    HI_HDMI_CONTNET_BUTT
} hi_hdmi_content_type;
```

成员

成员名称	描述
HI_HDMI_CONTNET_GRAPHIC	图像信息。
HI_HDMI_CONTNET_PHOTO	照片信息。
HI_HDMI_CONTNET_CINEMA	相机信息。
HI_HDMI_CONTNET_GAME	游戏信息。
HI_HDMI_CONTNET_BUTT	预留值。

11.18.20.11.19 hi_hdmi_ycc_quant_range

说明

定义HDMI YUV量化范围。

定义

```
typedef enum {
    HI_HDMI_YCC_QUANT_LIMITED_RANGE,
    HI_HDMI_YCC_QUANT_FULL_RANGE,
    HI_HDMI_YCC_QUANT_BUTT
} hi_hdmi_ycc_quant_range;
```

成员

成员名称	描述
HI_HDMI_YCC_QUANT_LIMITED_RANGE	YUV分量有限范围。
HI_HDMI_YCC_QUANT_FULL_RANGE	YUV全量有限范围。
HI_HDMI_YCC_QUANT_BUTT	预留值。

11.18.20.11.20 hi_hdmi_avi_infoframe

说明

定义HDMI输出AVI（Audio Video Interleave）信息帧。

定义

```
typedef struct {
    hi_hdmi_video_format timing_mode;
    hi_hdmi_color_space color_space;
    hi_bool active_info_present;
    hi_hdmi_bar_info bar_info;
    hi_hdmi_scan_info scan_info;
    hi_hdmi_colorimetry colorimetry;
    hi_hdmi_ex_colorimetry ex_colorimetry;
    hi_pic_aspect_ratio aspect_ratio;
    hi_hdmi_active_aspect_ratio active_aspect_ratio;
    hi_hdmi_pic_scaline pic_scaling;
    hi_hdmi_rgb_quant_range rgb_quant;
    hi_bool is_it_content;
    hi_hdmi_pixel_repetition pixel_repetition;
    hi_hdmi_content_type content_type;
    hi_hdmi_ycc_quant_range ycc_quant;
    hi_u16 line_n_end_of_top_bar;
    hi_u16 line_n_start_of_bot_bar;
    hi_u16 pixel_n_end_of_left_bar;
    hi_u16 pixel_n_start_of_right_bar;
} hi_hdmi_avi_infoframe;
```

成员

成员名称	描述
timing_mode	视频时序，必须与vo输入时序对应,否则属于未定义行为，影响送显。
color_space	颜色空间，必须与vo输入颜色空间对应，用户送图时必须设置为HI_HDMI_COLOR_SPACE_YCBCR444。
active_info_present	信息是否有效：当前版本仅支持设置为HI_FALSE。
bar_info	面板信息：当前版本仅支持设置为HI_HDMI_BAR_INFO_NOT_VALID。
scan_info	扫描信息：当前版本仅支持设置为HI_HDMI_SCAN_INFO_NO_DATA。
colorimetry	色域：当前版本仅支持设置为HI_HDMI_COMMON_COLORIMETRY_ITU601。
ex_colorimetry	扩展色域：当前版本仅支持设置为HI_HDMI_COMMON_COLORIMETRY_XVYCC_601。
aspect_ratio;	图像宽高比：当前版本仅支持设置为HI_HDMI_PIC_ASPECT_RATIO_4TO3或者HI_HDMI_PIC_ASPECT_RATIO_16TO9。
active_aspect_ratio	实际图像有效宽高比：当前版本仅支持设置为HI_HDMI_ACTIVE_ASPECT_RATIO_SAME_PIC。

成员名称	描述
pic_scaling	图像扫描：当前版本仅支持设置为HI_HDMI_PIC_NON_UNIFORM_SCALING。
rgb_quant	RGB量化：当前版本仅支持设置为HI_HDMI_RGB_QUANT_FULL_RANGE。
is_it_content	判断内容是否有效：当前版本仅支持设置为HI_FALSE。
pixel_repetition	像素加倍：当前版本仅支持设置为HI_HDMI_PIXEL_REPET_NO。
content_type	内容信息类型：当前版本仅支持设置为HI_HDMI_CONTNET_PHOTO。
ycc_quant	YUV分量量化：当前版本仅支持设置为HI_HDMI_YCC_QUANT_FULL_RANGE。
line_n_end_of_top_bar	顶部终止行数：当前版本仅支持设置为0。
line_n_start_of_bot_bar	底部开始行数：当前版本仅支持设置为0。
pixel_n_end_of_left_bar	左侧终止像素数：当前版本仅支持设置为0。
pixel_n_start_of_right_bar	右侧开始像素数：当前版本仅支持设置为0。

11.18.20.11.21 hi_hdmi_audio_chn_cnt

说明

定义音频通道数。

定义

```
typedef enum {
    HI_HDMI_AUDIO_CHN_CNT_STREAM,
    HI_HDMI_AUDIO_CHN_CNT_2,
    HI_HDMI_AUDIO_CHN_CNT_3,
    HI_HDMI_AUDIO_CHN_CNT_4,
    HI_HDMI_AUDIO_CHN_CNT_5,
    HI_HDMI_AUDIO_CHN_CNT_6,
    HI_HDMI_AUDIO_CHN_CNT_7,
    HI_HDMI_AUDIO_CHN_CNT_8,
    HI_HDMI_AUDIO_CHN_CNT_BUTT
} hi_hdmi_audio_chn_cnt;
```

成员

成员名称	描述
HI_HDMI_AUDIO_CHN_CNT_STREAM	STREAM模式的音频通道形态。

成员名称	描述
HI_HDMI_AUDIO_CHN_CNT_2	2声道。
HI_HDMI_AUDIO_CHN_CNT_3	3声道。
HI_HDMI_AUDIO_CHN_CNT_4	4声道。
HI_HDMI_AUDIO_CHN_CNT_5	5声道。
HI_HDMI_AUDIO_CHN_CNT_6	6声道。
HI_HDMI_AUDIO_CHN_CNT_7	7声道。
HI_HDMI_AUDIO_CHN_CNT_8	8声道。
HI_HDMI_AUDIO_CHN_CNT_BUTT	预留值。

11.18.20.11.22 hi_hdmi_coding_type

说明

定义HDMI音频输出采样位宽。

定义

```
typedef enum {
    HI_HDMI_AUDIO_CODING_REFER_STREAM_HEAD,
    HI_HDMI_AUDIO_CODING_PCM,
    HI_HDMI_AUDIO_CODING_AC3,
    HI_HDMI_AUDIO_CODING_MPEG1,
    HI_HDMI_AUDIO_CODING_MP3,
    HI_HDMI_AUDIO_CODING_MPEG2,
    HI_HDMI_AUDIO_CODING_AACLC,
    HI_HDMI_AUDIO_CODING_DTS,
    HI_HDMI_AUDIO_CODING_ATRAC,
    HI_HDMI_AUDIO_CODIND_ONE_BIT_AUDIO,
    HI_HDMI_AUDIO_CODING_ENAHNCED_AC3,
    HI_HDMI_AUDIO_CODING_DTS_HD,
    HI_HDMI_AUDIO_CODING_MAT,
    HI_HDMI_AUDIO_CODING_DST,
    HI_HDMI_AUDIO_CODING_WMA_PRO,
    HI_HDMI_AUDIO_CODING_BUTT
} hi_hdmi_coding_type;
```

成员

成员名称	描述
HI_HDMI_AUDIO_CODING_REFER_STREAM_HEAD	REFER_STREAM_HEAD格式音频。
HI_HDMI_AUDIO_CODING_PCM	PCM格式音频。
HI_HDMI_AUDIO_CODING_AC3	AC3格式音频。
HI_HDMI_AUDIO_CODING_MPEG1	MPEG1格式音频。
HI_HDMI_AUDIO_CODING_MP3	MP3格式音频。
HI_HDMI_AUDIO_CODING_MPEG2	MPEG2格式音频。
HI_HDMI_AUDIO_CODING_AACLC	AACLC格式音频。
HI_HDMI_AUDIO_CODING_DTS	DTS格式音频。
HI_HDMI_AUDIO_CODING_ATRAC	ATRAC格式音频。
HI_HDMI_AUDIO_CODIND_ONE_BIT_AUDIO	ONE_BIT_AUDIO格式音频。
HI_HDMI_AUDIO_CODING_ENAHNCED_AC3	ENAHNCED_AC3格式音频。
HI_HDMI_AUDIO_CODING_DTS_HD	DTS_HD格式音频。
HI_HDMI_AUDIO_CODING_MAT	MAT格式音频。
HI_HDMI_AUDIO_CODING_DST	DST格式音频。
HI_HDMI_AUDIO_CODING_WMA_PRO	WMA_PRO格式音频。
HI_HDMI_AUDIO_CODING_BUTT	预留值。

11.18.20.11.23 hi_hdmi_audio_sample_size

说明

定义音频采样大小。

定义

```
typedef enum {
    HI_HDMI_AUDIO_SAMPLE_SIZE_STREAM,
    HI_HDMI_AUDIO_SAMPLE_SIZE_16,
    HI_HDMI_AUDIO_SAMPLE_SIZE_20,
    HI_HDMI_AUDIO_SAMPLE_SIZE_24,
    HI_HDMI_AUDIO_SAMPLE_SIZE_BUTT
} hi_hdmi_audio_sample_size;
```

成员

成员名称	描述
HI_HDMI_AUDIO_SAMPLE_SIZE_STREAM	STREAM格式采样率。
HI_HDMI_AUDIO_SAMPLE_SIZE_16	音频采样深度为16。
HI_HDMI_AUDIO_SAMPLE_SIZE_20	音频采样深度为20。
HI_HDMI_AUDIO_SAMPLE_SIZE_24	音频采样深度为24。
HI_HDMI_AUDIO_SAMPLE_SIZE_BUTT	预留值。

11.18.20.11.24 hi_hdmi_audio_sample_freq

说明

定义HDMI 音频采样率。

定义

```
typedef enum {
    HI_HDMI_AUDIO_SAMPLE_FREQ_STREAM,
    HI_HDMI_AUDIO_SAMPLE_FREQ_32000,
    HI_HDMI_AUDIO_SAMPLE_FREQ_44100,
    HI_HDMI_AUDIO_SAMPLE_FREQ_48000,
    HI_HDMI_AUDIO_SAMPLE_FREQ_88200,
    HI_HDMI_AUDIO_SAMPLE_FREQ_96000,
    HI_HDMI_AUDIO_SAMPLE_FREQ_176400,
    HI_HDMI_AUDIO_SAMPLE_FREQ_192000,
    HI_HDMI_AUDIO_SAMPLE_FREQ_BUTT
} hi_hdmi_audio_sample_freq;
```

成员

成员名称	描述
HI_HDMI_AUDIO_SAMPLE_FREQ_STREAM	定义音频采样频率枚举类型：流控制。
HI_HDMI_AUDIO_SAMPLE_FREQ_32000	定义音频采样频率枚举类型：32000hz。
HI_HDMI_AUDIO_SAMPLE_FREQ_44100	定义音频采样频率枚举类型：32000hz。
HI_HDMI_AUDIO_SAMPLE_FREQ_48000	定义音频采样频率枚举类型：48000hz。
HI_HDMI_AUDIO_SAMPLE_FREQ_88200	定义音频采样频率枚举类型：88200hz。
HI_HDMI_AUDIO_SAMPLE_FREQ_96000	定义音频采样频率枚举类型：96000hz。
HI_HDMI_AUDIO_SAMPLE_FREQ_176400	定义音频采样频率枚举类型：176400hz。
HI_HDMI_AUDIO_SAMPLE_FREQ_192000	定义音频采样频率枚举类型：192000hz。
HI_HDMI_AUDIO_SAMPLE_FREQ_BUTT	预留值。

11.18.20.11.25 hi_hdmi_level_shift_val

说明

定义HDMI音频调测等级。

定义

```
typedef enum {
    HI_HDMI_LEVEL_SHIFT_VAL_0_DB,
    HI_HDMI_LEVEL_SHIFT_VAL_1_DB,
    HI_HDMI_LEVEL_SHIFT_VAL_2_DB,
    HI_HDMI_LEVEL_SHIFT_VAL_3_DB,
    HI_HDMI_LEVEL_SHIFT_VAL_4_DB,
    HI_HDMI_LEVEL_SHIFT_VAL_5_DB,
    HI_HDMI_LEVEL_SHIFT_VAL_6_DB,
    HI_HDMI_LEVEL_SHIFT_VAL_7_DB,
    HI_HDMI_LEVEL_SHIFT_VAL_8_DB,
    HI_HDMI_LEVEL_SHIFT_VAL_9_DB,
    HI_HDMI_LEVEL_SHIFT_VAL_10_DB,
    HI_HDMI_LEVEL_SHIFT_VAL_11_DB,
    HI_HDMI_LEVEL_SHIFT_VAL_12_DB,
    HI_HDMI_LEVEL_SHIFT_VAL_13_DB,
    HI_HDMI_LEVEL_SHIFT_VAL_14_DB,
    HI_HDMI_LEVEL_SHIFT_VAL_15_DB,
    HI_HDMI_LEVEL_SHIFT_VAL_BUTT
} hi_hdmi_level_shift_val;
```

成员

成员名称	描述
HI_HDMI_LEVEL_SHIFT_VAL_0_DB	左移值：0db。
HI_HDMI_LEVEL_SHIFT_VAL_1_DB	左移值：1db。
HI_HDMI_LEVEL_SHIFT_VAL_2_DB	左移值：2db。
HI_HDMI_LEVEL_SHIFT_VAL_3_DB	左移值：3db。
HI_HDMI_LEVEL_SHIFT_VAL_4_DB	左移值：4db。
HI_HDMI_LEVEL_SHIFT_VAL_5_DB	左移值：5db。
HI_HDMI_LEVEL_SHIFT_VAL_6_DB	左移值：6db。
HI_HDMI_LEVEL_SHIFT_VAL_7_DB	左移值：7db。
HI_HDMI_LEVEL_SHIFT_VAL_8_DB	左移值：8db。
HI_HDMI_LEVEL_SHIFT_VAL_9_DB	左移值：9db。
HI_HDMI_LEVEL_SHIFT_VAL_10_DB	左移值：10db。
HI_HDMI_LEVEL_SHIFT_VAL_11_DB	左移值：11db。
HI_HDMI_LEVEL_SHIFT_VAL_12_DB	左移值：12db。
HI_HDMI_LEVEL_SHIFT_VAL_13_DB	左移值：13db。
HI_HDMI_LEVEL_SHIFT_VAL_14_DB	左移值：14db。
HI_HDMI_LEVEL_SHIFT_VAL_15_DB	左移值：15db。
HI_HDMI_LEVEL_SHIFT_VAL_BUTT	预留值。

11.18.20.11.26 hi_hdmi_lfe_playback_level

说明

定义HDMI音频回放等级。

定义

```
typedef enum {
    HI_HDMI_LFE_PLAYBACK_NO,
    HI_HDMI_LFE_PLAYBACK_0_DB,
    HI_HDMI_LFE_PLAYBACK_10_DB,
    HI_HDMI_LFE_PLAYBACK_BUTT
} hi_hdmi_lfe_playback_level;
```

成员

成员名称	描述
HI_HDMI_LFE_PLAYBACK_NO	回放功能关闭。
HI_HDMI_LFE_PLAYBACK_0_DB	回放功能等级为0db。
HI_HDMI_LFE_PLAYBACK_10_DB	回放功能等级为10db。
HI_HDMI_LFE_PLAYBACK_BUTT	预留值。

11.18.20.11.27 hi_hdmi_audio_infoframe

说明

定义HDMI AUDIO信息帧单元。

定义

```
typedef struct {
    hi_hdmi_audio_chn_cnt chn_cnt;
    hi_hdmi_coding_type coding_type;
    hi_hdmi_audio_sample_size sample_size;
    hi_hdmi_audio_sample_freq sampling_freq;
    hi_u8 chn_alloc;
    hi_hdmi_level_shift_val level_shift;
    hi_hdmi_lfe_playback_level lfe_playback_level;
    hi_bool down_mix_inhibit;
} hi_hdmi_audio_infoframe;
```

成员

成员名称	描述
chn_cnt	音频通道个数：当前版本仅支持设置为HI_HDMI_AUDIO_CHN_CNT_2。
coding_type	音频格式：当前版本仅支持设置为HI_HDMI_AUDIO_CODING_PCM。
sample_size	音频采样深度（位宽）：当前版本仅支持设置为HI_HDMI_AUDIO_SAMPLE_SIZE_16。
sampling_freq	enSamplingFrequency：当前版本仅支持设置为HI_HDMI_AUDIO_SAMPLE_FREQ_48000。
chn_alloc	Channel/Speaker分配：当前版本仅支持设置为0。
level_shift	Level Shift Value，左移值：当前版本仅支持设置为HI_HDMI_LEVEL_SHIFT_VAL_0_DB。
lfe_playback_level	LFE playback level information ，LFE 播放等级信息。当前版本仅支持设置为HI_HDMI_LFE_PLAYBACK_NO。
down_mix_inhibit	Down-mix Inhibit 标志位：当前版本仅支持设置为HI_FALSE。

11.18.20.11.28 hi_hdmi_vendorspec_infoframe

说明

定义HDMI定制化信息帧单元结构体。

定义

```
typedef struct {
    hi_u8 data_len;
    hi_u8 user_data[HI_HDMI_VENDOR_USER_DATA_MAX_LEN];
} hi_hdmi_vendorspec_infoframe;
```

成员

成员名称	描述
data_len	IEEE（Institute of Electrical and Electronics Engineers）注册码数组长度，最大值为22。
user_data	IEEE注册码，当前版本暂不支持使用。 #define HI_HDMI_VENDOR_USER_DATA_MAX_LEN 22

说明

当前版本中该结构体暂未使用，不可赋值，否则属于未定义行为。

11.18.20.11.29 hi_hdmi_infoframe_unit

说明

定义HDMI信息帧类型结构。

定义

```
typedef union {
    hi_hdmi_avi_infoframe avi_infoframe;
    hi_hdmi_audio_infoframe audio_infoframe;
    hi_hdmi_vendorspec_infoframe vendor_spec_infoframe;
} hi_hdmi_infoframe_unit;
```

成员

成员名称	描述
avi_infoframe	显示信息帧单元。
audio_infoframe	音频信息帧单元。
vendor_spec_infoframe	供应商定制信息帧单元。

11.18.20.11.30 hi_hdmi_infoframe

说明

定义HDMI输出信息帧结构体。

定义

```
typedef struct {
    hi_hdmi_infoframe_type infoframe_type;
    hi_hdmi_infoframe_unit infoframe_unit;
} hi_hdmi_infoframe;
```

成员

成员名称	描述
infoframe_type	信息帧类型，仅支持HI_INFOFRAME_TYPE_AUDIO和HI_INFOFRAME_TYPE_AVI信息帧。
infoframe_unit	信息帧单元(内容)结构体。

11.18.20.11.31 hi_hdmi_sink_capability

说明

定义HDMI Sink端能力集结构体。

定义

```
typedef struct {
    hi_bool is_connected;
    hi_bool support_hdmi;
    hi_bool support_hdmi_2_0;
} hi_hdmi_sink_capability;
```

成员

成员名称	描述
is_connected	设备是否连接。 ● HI_TRUE：已连接 ● HI_FALSE：未连接
support_hdmi	设备是否支持HDMI1.4。 ● HI_TRUE：支持 ● HI_FALSE：不支持
support_hdmi_2_0	设备是否支持HDMI2.0。 ● HI_TRUE：支持 ● HI_FALSE：不支持

11.18.20.11.32 hi_hdmi_event_type

说明

定义HDMI事件通知枚举。

定义

```
typedef enum {
    HI_HDMI_EVENT_HOTPLUG = 0x10,
    HI_HDMI_EVENT_NO_PLUG,
    HI_HDMI_EVENT_EDID_FAIL,
    HI_HDMI_EVENT_BUTT
} hi_hdmi_event_type;
```

成员

成员名称	描述
HI_HDMI_EVENT_HOTPLUG	HDMI Cable插入事件。
HI_HDMI_EVENT_NO_PLUG	HDMI Cable拔出事件。
HI_HDMI_EVENT_EDID_FAIL	HDMI EDID读取失败事件，当前预留，未使能。

11.18.20.11.33 hi_hdmi_callback

说明

定义HDMI回调函数指针类型。

定义

```
typedef void (*hi_hdmi_callback)(hi_hdmi_event_type event, void *private_data);
```

成员

成员名称	描述
event	HDMI事件通知类型。
private_data	事件私有数据。

11.18.20.11.34 hi_hdmi_callback_func

说明

定义HDMI回调函数结构体。

定义

```
typedef struct {
    hi_hdmi_callback hdmi_event_callback;
    void *private_data;
} hi_hdmi_callback_func;
```

成员

成员名称	描述
hdmi_event_callback	事件处理回调函数。
private_data	回调函数参数私有数据。

11.18.20.12 VPC 图像处理

11.18.20.12.1 hi_vpc_chn

说明

描述图片处理通道号。

定义

```
typedef hi_s32 hi_vpc_chn;
```

11.18.20.12.2 hi_vpc_pic_info

说明

图像信息结构体，用来描述VPC功能输入和输出图片信息。

定义

```
typedef struct {
    hi_void* picture_address;
    hi_u32 picture_buffer_size;
    hi_u32 picture_width;
    hi_u32 picture_height;
    hi_u32 picture_width_stride;
    hi_u32 picture_height_stride;
    hi_pixel_format picture_format;
} hi_vpc_pic_info;
```

成员

成员名称	描述
picture_address	存放图片数据的Device地址。
picture_buffer_size	存放图片数据的缓冲区大小。
picture_width	图片真实宽。
picture_height	图片真实高。
picture_width_stride	图片宽stride。
picture_height_stride	图片高stride。
picture_format	目标图片的格式。

11.18.20.12.3 hi_vpc_crop_region

说明

VPC抠图功能需要的参数。

定义

```
typedef struct {
    hi_u32 top_offset;
    hi_u32 left_offset;
    hi_u32 crop_width;
    hi_u32 crop_height;
} hi_vpc_crop_region;
```

成员

成员名称	描述
top_offset	上偏移。
left_offset	左偏移。
crop_width	抠图区域宽。
crop_height	抠图区域高。

11.18.20.12.4 hi_vpc_resize_info

说明

VPC缩放功能需要的参数。

定义

```
typedef struct {
    hi_u32 resize_width;
    hi_u32 resize_height;
    hi_u32 interpolation;
} hi_vpc_resize_info;
```

成员

成员名称	描述
resize_width	缩放之后的宽。
resize_height	缩放之后的高。
interpolation	缩放算法。 Atlas 推理系列产品，支持如下缩放算法： ● 0：业界通用的Bilinear算法（与OpenCV算法的计算过程类似，当输入和输出图片格式都为RGB时，在[1/32, 512]的缩放范围内，与OpenCV的单个像素值最大差异为正负1） ● 1：业界通用的Nearest neighbor 算法（与OpenCV算法的计算过程类似）

11.18.20.12.5 hi_vpc_crop_region_info

说明

VPC抠图功能需要的参数。

定义

```
typedef struct {
    hi_vpc_pic_info dest_pic_info;
```

```
    hi_vpc_crop_region crop_region;
} hi_vpc_crop_region_info;
```

成员

成员名称	描述
dest_pic_info	抠图目标图片信息。
crop_region	抠图区域信息。

11.18.20.12.6 hi_vpc_crop_resize_region

说明

VPC抠图并缩放功能需要的参数。

定义

```
typedef struct {
    hi_vpc_pic_info dest_pic_info;
    hi_vpc_crop_region crop_region;
    hi_vpc_resize_info resize_info;
} hi_vpc_crop_resize_region;
```

成员

成员名称	描述
dest_pic_info	目标图片信息。
crop_region	抠图区域信息。
resize_info	图片缩放信息。

注意事项

处理图片的数量不能大于256个。缩放宽高必须与目标图片宽高一致。

11.18.20.12.7 hi_vpc_crop_resize_paste_region

说明

VPC抠图然后缩放并贴图到特定区域功能需要的参数。

定义

```
typedef struct {
    hi_vpc_pic_info dest_pic_info;
    hi_vpc_crop_region crop_region;
    hi_vpc_resize_info resize_info;
    hi_u32 dest_top_offset;
    hi_u32 dest_left_offset;
} hi_vpc_crop_resize_paste_region;
```

成员

成员名称	描述
dest_pic_info	目标图片信息。
crop_region	抠图区域信息。
resize_info	图片缩放信息。
dest_top_offset	贴图到另外一张图的顶部偏移。
dest_left_offset	贴图到另外一张图的左侧偏移。 注：Atlas 推理系列产品上，左偏移16像素对齐。

11.18.20.12.8 hi_vpc_bord_type

说明

定义边界填充的类型。

定义

```
typedef enum {
    HI_BORDER_CONSTANT = 0,
    HI_BORDER_REPLICATE,
    HI_BORDER_REFLECT,
    HI_BORDER_REFLECT_101
} hi_vpc_bord_type;
```

成员

成员名称	描述
HI_BORDER_CONSTANT	添加有颜色的常数值边界。
HI_BORDER_REPLICATE	重复最后一个元素。 举例，其中*表示任意图像元素: aaaaaa|a*****h|hhhhhhh;
HI_BORDER_REFLECT	边界元素的镜像，镜像包括边界元素。 举例，其中*表示任意图像元素：ba|abc*******fgh|hg;
HI_BORDER_REFLECT_101	边界元素的镜像，镜像不包括边界元素。 举例，其中*表示任意图像元素：cb|abc****fgh|gf;

注意事项

版本	约束
Atlas 推理系列产品	支持HI_BORDER_CONSTANT和HI_BORDER_REPLICATE可以填充到4096*4096

11.18.20.12.9 hi_vpc_make_border_info

说明

VPC图像填充需要用的信息。

定义

```
typedef struct {
    hi_u32 top;
    hi_u32 bottom;
    hi_u32 left;
    hi_u32 right;
    hi_vpc_bord_type border_type;
    hi_vpc_scalar scalar_value;
} hi_vpc_make_border_info;
```

成员

成员名称	描述
top	顶部填充像素数。
bottom	底部填充像素数。
left	左侧填充像素数。
right	右侧填充像素数。
border_type	扩充边缘像素的类型。
scalar_value	仅在填充类型为HI_BORDER_CONSTANT的时候有效，指定填充的像素值。

11.18.20.12.10 hi_vpc_histogram_config

说明

存放直方图统计结果的结构体。

定义

```
typedef struct {
    hi_u32 histogram_y_or_r[256];
    hi_u32 histogram_u_or_g[256];
    hi_u32 histogram_v_or_b[256];
} hi_vpc_histogram_config;
```

成员

成员名称	描述
histogram_y_or_r	Y或者R分量，像素分布情况。
histogram_u_or_g	U或者G分量，像素分布情况。
histogram_v_or_b	V或者B分量，像素分布情况。

11.18.20.12.11 hi_vpc_lut_remap

说明

配置图像各分量重映射信息的结构体。

定义

```
typedef struct {
    hi_u8 map_value_y_or_r[256];
    hi_u8 map_value_u_or_g[256];
    hi_u8 map_value_v_or_b[256];
} hi_vpc_lut_remap;
```

成员

成员名称	描述
map_value_y_or_r	Y或者R分量的重映射配置信息。
map_value_u_or_g	U或者G分量的重映射配置信息。
map_value_v_or_b	V或者B分量的重映射配置信息。

11.18.20.12.12 hi_vpc_scalar

说明

定义图像处理像素各分量结构体。

定义

```
typedef struct {
    hi_double val[4];
} hi_vpc_scalar;
```

成员

成员名称	描述
val	存放颜色分量的值，取值范围[0,255]。 当前支持用户手动按顺序存放R、G、B分量或Y、U、V分量的值，即val[0]存放R分量或Y分量，val[1]存放G分量或U分量，val[2]存放B分量或V分量，val[3]预留。 若输入、输出图片格式都是YUV格式，填充YUV分量的值；反之，填充的是RGB分量的值。

11.18.20.12.13 hi_vpc_chn_attr

说明

定义图像处理通道属性结构体。

定义

```
typedef struct {
    hi_s32 attr;
    hi_u32 pic_width;
    hi_u32 pic_height;
} hi_vpc_chn_attr;
```

成员

成员名称	描述
attr	任务队列深度，取值范围：[0,350]，取值越大，队列深度越深，在一个通道内可下发的任务数量越多，但系统内部内存消耗线性增加。 attr参数取值为[0,350]之间，具体说明如下： ● 取值为0，表示默认使用32。 ● 取值为[1, 10)，VPC内部考虑批量任务的处理（例如hi_mpi_vpc_batch_crop_resize_paste），自动将队列深度设置为10。 ● 取值为[10, 350]，取用户所配置的值。 **说明** 任务队列深度取值大于350时，当前版本会自行采用350的取值，为避免后续版本的兼容性问题，建议用户在[队列深度最小值,350]范围内取值。 在一个VPC处理通道关联n个线程的场景下，防止通道队列满而阻塞任务下发，建议增加任务队列深度以保证性能，当线程数≤32时，可使用默认队列深度32；当线程数>32时，建议队列深度大于或等于线程数。
pic_width	通道支持的处理图像最大宽（以像素为单位）。预留字段，暂不支持。需要用户手动设置为0，避免后续版本的兼容性问题。 静态属性。

成员名称	描述
pic_height	通道支持的处理图像最大高（以像素为单位）。预留字段，暂不支持。需要用户手动设置为0，避免后续版本的兼容性问题。 静态属性。

11.18.20.12.14 hi_vpc_crop_resize_border_region

说明

VPC抠图缩放填充信息结构体。

定义

```
typedef struct {
    hi_vpc_pic_info dest_pic_info;
    hi_vpc_crop_region crop_region;
    hi_vpc_resize_info resize_info;
    hi_u32 dest_top_offset;
    hi_u32 dest_left_offset;
    hi_vpc_bord_type border_type;
    hi_vpc_scalar scalar_value;
} hi_vpc_crop_resize_border_region;
```

成员

成员名称	描述
dest_pic_info	目标图片信息。
crop_region	原图抠图区域信息。
resize_info	抠图缩放信息。
dest_top_offset	贴图到另外一张图的顶部偏移。
dest_left_offset	贴图到另外一张图的左侧偏移。
border_type	扩充边缘像素的类型。
scalar_value	仅在填充类型为HI_BORDER_CONSTANT的时候有效，指定填充的像素值。 当前支持用户手动按顺序存放R、G、B分量或Y、U、V分量的值。 若输入输出都是YUV格式，填充YUV分量的值；反之，填充的是RGB分量的值。

注意事项

默认抠图区域的图片格式会转成目标图片的格式，然后再缩放、填充至目标图片的宽高。

11.18.20.12.15 hi_csc_conf

说明

定义色域配置信息。

定义

```
type struct{
    hi_u32 alpha;
    hi_u32 attr[4];
};
```

成员

成员名称	描述
alpha	用于设置图像的透明度，取值范围：[0, 255]，取值越小，图像越透明。 在查看图像时，无法直接感知透明度，一般在对图像进行融合、渲染等处理时才会使用透明度信息。
attr	预留字段，暂不支持。需要用户手动将数组中的每个元素值设置为0，避免后续版本的兼容性问题。

11.18.20.12.16 hi_roundview_stitching_param

说明

定义环视拼接参数的结构体。

定义

```
typedef struct  {
    hi_stiching_ipm_param imp_conf;
    hi_stitching_gain_param gain_conf;
    hi_u32 reserved[8];
} hi_roundview_stitching_param;
```

成员

成员名称	描述
imp_conf	畸变矫正参数结构体。
gain_conf	增益补偿参数结构体。 若无需增益补偿，则需要给该结构体内的gain_type成员设置为GAIN_NONE。
reserved	预留参数，为保证后续版本兼容性，请务必使用memset结构体方式进行清零初始化，在代码中必须避免显式对reserved字段进行访问。

11.18.20.12.17 hi_roundview_stitching_pic_param

说明

定义环视拼接输入、输出图片信息的结构体。

定义

```
typedef struct  {
    hi_vpc_pic_info *dest_pic;
    hi_u32 count;
    hi_vpc_pic_info *source_pic;
} hi_roundview_stitching_pic_param;
```

成员

成员名称	描述
dest_pic	输出图片指针。
count	输入图片数量，当前固定配置为4，否则会返回报错。
source_pic	输入图片数组。 source_pic[0]存放第一张输入图片数据，source_pic[1]存放第二张输入图片数据，source_pic[2]存放第三张输入图片数据，source_pic[3]存放第四张输入图片数据。 4张输入图片的宽保持一致、高保持一致。

11.18.20.12.18 hi_stiching_ipm_param

说明

定义畸变矫正参数的结构体。

定义

```
typedef struct {
    hi_u32 src_pic_width;
    hi_u32 src_pic_height;
    hi_u32 dest_pic_width;
    hi_u32 dest_pic_height;
    hi_u32 ipm_table_len;
    hi_stiching_ipm_table *ipm_table_address;
} hi_stiching_ipm_param;
```

成员

成员名称	描述
src_pic_width	输入图片宽。 4张输入图片的宽保持一致。

成员名称	描述
src_pic_height	输入图片高。 4张输入图片的高保持一致。
dest_pic_width	输出图片宽。
dest_pic_height	输出图片高。
ipm_table_len	数组长度，必须为dest_pic_width * dest_pic_height
ipm_table_address	畸变矫正结构体数组。 由用户提前申请内存存放数组中的数据，内存大小（单位为Byte） = sizeof(hi_stiching_ipm_table) * ipm_table_len，该内存大小可能比较大，如果直接使用C标准库中的malloc接口申请内存，会导致内存碎片化，大页个数减少，进而影响使用**hi_mpi_dvpp_malloc**接口申请DVPP各功能的输入/输出内存。因此**推荐**直接使用**hi_mpi_dvpp_malloc**接口申请存放ipm_table_address数组数据的内存。

11.18.20.12.19 hi_stiching_ipm_table

说明

定义畸变矫正表的结构体。即每个输出图片像素点，是怎么由2幅输入图片融合成的。

定义

```
typedef struct {
    hi_u32 pic1_index;
    hi_u32 pic2_index;
    hi_float weight1;
    hi_float weight2;
    hi_float offset1_x;
    hi_float offset1_y;
    hi_float offset2_x;
    hi_float offset2_y;
}hi_stiching_ipm_table;
```

成员

设置输入图片编号时，如果pic1_index、pic2_index设置为同一个编号，则表示图像融合时参考的是同一张输入图片。

设置图像融合的比重时，要求weight2 + weight1 = 1。

如果weight1、weight2都设置为0 ，无论pic1_index、pic2_index配置什么，硬件都不会进行融合。

成员名称	描述
pic1_index	对应输入图片1的编号，取值范围[0, 3]。
pic2_index	对应输入图片2的编号，取值范围[0, 3]。

成员名称	描述
weight1	图像融合时图片1的比重，取值范围[0, 1]。
weight2	图像融合时图片2的比重，取值范围[0, 1]。
offset1_x	图片1插值点x坐标。
offset1_y	图片1插值点y坐标。
offset2_x	图片2插值点x坐标。
offset2_y	图片2插值点y坐标。

11.18.20.12.20 hi_stitching_gain_param

说明

定义增益补偿参数的结构体。

定义

```
typedef struct {
    hi_stitching_histogram_type hist_type;
    hi_stitching_gain_type gain_type;
    hi_void *user_data;
    hi_s32 (*calulate_gain_callback)(
        hi_void *user_data,
        hi_void *hist_config,
        hi_u32 hist_count,
        void *gain,
        hi_u32 gain_size);
} hi_stitching_gain_param;
```

成员

成员名称	描述
hist_type	直方图统计类型。
gain_type	增益补偿类型。
user_data	用户自定义数据。

成员名称	描述
calulate_gain_callback	回调函数指针。 gain_type选择GAIN_LUT时，必须配置回调函数指针。 回调函数内的参数说明如下： ● user_data 系统将hi_stitching_gain_param.user_data用户自定义数据赋值给回调函数内的user_data。 ● hist_config 回调函数中的hist_config参数，由系统内部管理，用户可以在下发环视拼接任务之后，释放环视拼接内部资源前，通过该参数获取直方图信息。 ● hist_count hi_stitching_histogram_param结构体对象的个数。 当直方图统计类型hi_stitching_histogram_type为OVERLAP_HISTOGRAM 时，hist_count固定为8；当直方图统计类型hi_stitching_histogram_type为GLOBAL_HISTOGRAM时，hist_count固定为4。 ● gain 回调函数中的gain参数，需由用户转换为hi_vpc_lut_remap结构体数组，数组长度为4： `hi_vpc_lut_remap* lutRemap = (hi_vpc_lut_remap*)gain` ● gain_size hi_vpc_lut_remap结构体数组的长度，当前为4。

11.18.20.12.21 hi_stitching_histogram_type

说明

环视拼接的直方图统计类型。

定义

```
typedef enum {
    OVERLAP_HISTOGRAM = 0,    // 对8个重叠区域做直方图统计
    GLOBAL_HISTOGRAM,         // 对4张输入图做直方图统计
} hi_stitching_histogram_type;
```

11.18.20.12.22 hi_stitching_gain_type

说明

定义增益补偿类型的枚举。

定义

```
typedef enum  {
    GAIN_NONE = 0,  // 无需增益补偿
    GAIN_LUT,       // 由用户配置图像各分量重映射信息来实现增益补偿
} hi_stitching_gain_type;
```

11.18.20.12.23 hi_stitching_histogram_param

说明

在用于增益补偿的回调函数中，将hi_stitching_gain_param.calulate_gain_callback.hist_config字段转换为hi_stitching_histogram_param结构体，可参见**转换示例代码**。

定义

```
typedef struct {
    hi_u32 pic_index;
    hi_u32 overlap_pic_index;
    hi_vpc_histogram_config histogram_info;
} hi_stitching_histogram_param;
```

成员

成员名称	描述
pic_index	输入图片编号。
overlap_pic_index	当直方图统计类型hi_stitching_histogram_type为OVERLAP_HISTOGRAM 时，overlap_pic_index参数表示与输入图片有重叠区域的图片的编号。
histogram_info	直方图统计结果。

转换示例代码

```
hi_stitching_histogram_param *hisData = (hi_stitching_histogram_param *)hist_config;
```

11.18.20.12.24 hi_vpc_crop_resize_resize_paste_region

说明

VPC抠图然后缩放并贴图到特定区域功能需要的参数，支持两次缩放。

定义

```
typedef struct {
    hi_vpc_pic_info dest_pic_info;
    hi_vpc_crop_region crop_region;
    hi_vpc_resize_info resize_info1;
    hi_vpc_resize_info resize_info2;
    hi_u32 dest_top_offset;
    hi_u32 dest_left_offset;
    hi_u32 reserved[2];
} hi_vpc_crop_resize_resize_paste_region;
```

成员

成员名称	描述
dest_pic_info	目标图片信息。

成员名称	描述
crop_region	抠图区域信息。
resize_info1	第一次图片缩放的信息。 缩放后的图片宽高需与输出图片的宽高对齐要求保持一致，请参见表11-45。
resize_info2	第二次图片缩放的信息。 缩放后的图片宽高需与输出图片的宽高对齐要求保持一致，请参见表11-45。
dest_top_offset	贴图到另外一张图的顶部偏移。
dest_left_offset	贴图到另外一张图的左侧偏移。 注：Atlas 推理系列产品上，左偏移16像素对齐。
reserved	预留参数，为保证后续版本兼容性，请务必使用memset结构体方式进行清零初始化，在代码中必须避免显式对reserved字段进行访问。

11.18.20.12.25 hi_vpc_workspace_param

说明

定义设置系统内部临时缓存相关的参数结构体。

定义

```
type struct {
    hi_workspace_func func;
    hi_u32 max_source_pic_width;
    hi_u32 max_source_pic_height;
    hi_u32 max_dest_pic_width;
    hi_u32 max_dest_pic_height;
    hi_u64 reserved;
} hi_vpc_workspace_param;
```

成员

参数名	输入/输出	说明
func	输入	指定为哪种功能申请临时缓存。
max_source_pic_width	输入	输入图片的最大宽度。
max_source_pic_height	输入	输入图片的最大高度。
max_dest_pic_width	输入	输出图片的最大宽度。
max_dest_pic_height	输入	输出图片的最大高度。

参数名	输入/输出	说明
reserved	输入	预留参数，为保证后续版本兼容性，请务必使用memset结构体方式进行清零初始化，在代码中必须避免显式对reserved字段进行访问。

11.18.20.12.26 hi_workspace_func

说明

定义需申请临时缓存的功能枚举。

定义

```
typedef enum {
    REMAP = 0,
    AFFINE,
} hi_workspace_func;
```

11.18.20.12.27 hi_remap_lut

说明

定义像素位置重映射表信息。

定义

```
typedef struct {
    hi_u32 src_pic_width;
    hi_u32 src_pic_height;
    hi_u32 dest_pic_width;
    hi_u32 dest_pic_height;
    hi_void *lut;
    hi_u32 lut_size;
    hi_u64 reserved;
} hi_remap_lut;
```

成员

成员名称	描述
src_pic_width	输入图片宽度。
src_pic_height	输入图片高度。
dest_pic_width	输出图片宽度。
dest_pic_height	输出图片高度。

成员名称	描述
lut_size	像素位置重映射表的内存大小，单位Byte。 可调用hi_mpi_vpc_get_lut_mem_size接口获取内存大小。
lut	像素位置重映射表的内存地址指针。
reserved	预留参数，为保证后续版本兼容性，请务必使用memset结构体方式进行清零初始化，在代码中必须避免显式对reserved字段进行访问。

11.18.20.12.28 hi_point_pair_info

说明

定义输入、输出图片中的像素点坐标信息的结构体。

定义

```
typedef struct {
    hi_float_point src_point[4];
    hi_u32 src_point_count;
    hi_float_point dest_point[4];
    hi_u32 dest_point_count;
} hi_point_pair_info;
```

成员

成员名称	描述
src_point	输入图片像素点坐标。
src_point_count	输入图片像素点坐标数量。
dest_point	输出图片像素点坐标。
dest_point_count	输出图片像素点坐标数量。

11.18.20.12.29 hi_float_point

说明

定义坐标信息的结构体。

定义

```
typedef struct {
    hi_float x;
    hi_float y;
} hi_float_point;
```

成员

成员名称	描述
x	像素点x坐标。
y	像素点y坐标。

11.18.20.12.30 hi_map_param

说明

定义用户原始map表信息的结构体。

定义

```
typedef struct {
    hi_void *map1;
    hi_void *map2;
    hi_u32 map_size;
    hi_u32 src_pic_width;
    hi_u32 src_pic_height;
    hi_u32 dest_pic_width;
    hi_u32 dest_pic_height;
    hi_u64 reserved;
} hi_map_param;
```

成员

成员名称	描述
map1	Remap变换使用的map1表, 表示(x,y)坐标的x，类型为hi_float, 大小为: dest_pic_width * dest_pic_height *sizeof(hi_float)
map2	Remap变换使用的map2表, 表示(x,y)坐标的y，类型为hi_float, 大小为: dest_pic_width * dest_pic_height *sizeof(hi_float)
map_size	map1或map2的内存大小，map1表和map2表的内存大小一样，单位Byte。
src_pic_width	map表对应的输入图片的宽度。
src_pic_height	map表对应的输入图片的高度。
dest_pic_width	map表对应的输出图像的宽度。
dest_pic_height	map表对应的输出图像的高度。
reserved	预留参数，为保证后续版本兼容性，请务必使用memset结构体方式进行清零初始化，在代码中必须避免显式对reserved字段进行访问。

11.18.20.12.31 hi_warp_transform_param

说明

定义Remap变换参数结构体。

定义

```
typedef struct {
    hi_vpc_pic_info       src;
    hi_vpc_pic_info       dst;
    hi_transform_config   transform_config;
} hi_warp_transform_param;
```

成员

成员名称	描述
src	输入图片信息
dst	输出图片信息。
transform_config	仿射/透视/Remap变换配置信息。

11.18.20.12.32 hi_transform_config

说明

定义仿射/透视/Remap变换配置结构体。

定义

```
typedef struct {
    hi_u32 interpolation;
    hi_vpc_bord_type  border_type;
    hi_vpc_scalar     scalar_value;
} hi_transform_config;
```

成员

成员名称	描述
interpolation	缩放算法。
border_type	边界填充的类型。
scalar_value	图像填充的像素值，只支持3通道。

11.18.20.12.33 hi_morph_shapes

说明

定义滤波卷积形状。

定义

```
typedef enum {
    MORPH_RECT = 0,
    MORPH_CROSS = 1,
    MORPH_ELLIPSE = 2,
    MORPH_MAX = 100
} hi_morph_shapes;
```

成员

成员名称	描述
MORPH_RECT	矩形。
MORPH_CROSS	交叉形，预留字段，暂不支持。
MORPH_ELLIPSE	椭圆形，预留字段，暂不支持。
MORPH_MAX	最大可支持的卷积滤波形状，预留字段，暂不支持。

11.18.20.12.34 hi_rotation

说明

定义旋转角度。

定义

```
typedef enum {
    HI_ROTATION_90 = 0,
    HI_ROTATION_180 = 1,
    HI_ROTATION_270 = 2,
} hi_rotation;
```

成员

成员名称	描述
HI_ROTATION_90	顺时针旋转90度。
HI_ROTATION_180	顺时针旋转180度。
HI_ROTATION_270	顺时针旋转270度。

11.18.20.12.35 hi_blk_size

说明

定义单个马赛克框的大小。

定义

```
typedefenum {
    HI_BLK_SIZE_4 = 0,
    HI_BLK_SIZE_8,
    HI_BLK_SIZE_16,
    HI_BLK_SIZE_32,
    HI_BLK_SIZE_64,
    HI_BLK_SIZE_128,
} hi_blk_size;
```

成员

成员名称	描述
HI_BLK_SIZE_4	预留字段，暂不支持。
HI_BLK_SIZE_8	单个马赛克是8*8的正方形。
HI_BLK_SIZE_16	单个马赛克是16*16的正方形。
HI_BLK_SIZE_32	单个马赛克是32*32的正方形。
HI_BLK_SIZE_64	预留字段，暂不支持。
HI_BLK_SIZE_128	预留字段，暂不支持。

11.18.20.12.36 hi_median_blur_param

说明

定义中值滤波信息结构体。

定义

```
typedef struct {
    hi_vpc_pic_info src;
    hi_vpc_pic_info dst;
    hi_median_blur_config median_blur_cfg;
} hi_median_blur_param;
```

成员

成员名称	描述
src	输入图片信息。
dst	输出图片信息。
median_blur_cfg	中值滤波配置参数。

11.18.20.12.37 hi_median_blur_config

说明

定义中值滤波参数结构体。

定义

```
typedef struct {
    hi_u32 kernel_size;
} hi_median_blur_config;
```

成员

成员名称	描述
kernel_size	中值滤波卷积核尺寸，只支持1，3或者5。

11.18.20.12.38 hi_blur_param

说明

定义滤波信息结构体。

定义

```
typedef struct {
    hi_vpc_pic_info src;
    hi_vpc_pic_info dst;
    hi_blur_config blur_cfg;
} hi_blur_param;
```

成员

成员名称	描述
src	输入图片信息。
dst	输出图片信息。
blur_cfg	滤波配置参数。

11.18.20.12.39 hi_blur_config

说明

定义滤波参数结构体。

定义

```
typedef struct {
    hi_size kernel_size;
    hi_morph_shapes morph_shapes;
    hi_point anchor;
    hi_u32 iterations;
    hi_vpc_bord_type border_type;
    hi_vpc_scalar scalar_value;
} hi_blur_config;
```

成员

成员名称	描述
kernel_size	卷积核的尺寸，kernel_size的宽和高只能设置为1，3或者5。
morph_shapes	滤波的卷积形状。
anchor	卷积核锚点，需要设置成(-1, -1)，预留字段。
iterations	卷积迭代次数，范围[1, 100]， 均值滤波处理只支持设置成1，腐蚀或膨胀处理时支持取[1, 100]范围内的值。
border_type	边界填充类型，只支持HI_BORDER_CONSTANT和HI_BORDER_REPLICATE两种填充模式。
scalar_value	边界填充值。

11.18.20.12.40 hi_gaussian_blur_param

说明

定义高斯滤波信息结构体。

定义

```
typedef struct {
    hi_vpc_pic_info src;
    hi_vpc_pic_info dst;
    hi_gaussian_blur_config gaussian_blur_cfg;
} hi_gaussian_blur_param;
```

成员

成员名称	描述
src	输入图片信息。
dst	输出图片信息。

成员名称	描述
gaussian_blur_cfg	高斯滤波配置参数。

11.18.20.12.41 hi_gaussian_blur_config

说明

定义高斯滤波参数结构体。

定义

```
typedef struct {
    hi_size kernel_size;
    hi_double sigma_x;
    hi_double sigma_y;
    hi_vpc_bord_type border_type;
    hi_vpc_scalar scalar_value;
    hi_u32 reserved[2];
} hi_gaussian_blur_config;
```

成员

成员名称	描述
kernel_size	卷积核尺寸，kernel_size的宽和高只能设置为1，3或者5。
sigma_x	高斯滤波x轴sigma值。
sigma_y	高斯滤波y轴sigma值，如果sigma_y=0，则sigma_y=sigma_x。
border_type	边界填充类型，只支持HI_BORDER_CONSTANT和HI_BORDER_REPLICATE两种填充模式。
scalar_value	边界填充值。
reserved[2]	预留参数，为保证后续版本兼容性，请务必使用memset结构体方式进行清零初始化，在代码中必须避免显式对reserved字段进行访问。

11.18.20.12.42 hi_filter_2d_param

说明

定义2D卷积滤波信息结构体。

定义

```
typedef struct {
    hi_vpc_pic_info src;
    hi_vpc_pic_info dst;
    hi_filter_2d_config filter_2d_cfg;
} hi_filter_2d_param;
```

成员

成员名称	描述
src	输入图片信息。
dst	输出图片信息。
filter_2d_cfg	2D卷积滤波配置参数。

11.18.20.12.43 hi_filter_2d_config

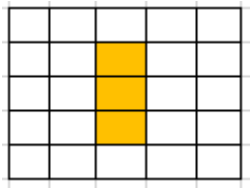
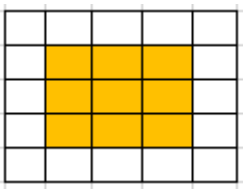
说明

定义2D卷积滤波参数结构体。

定义

```
typedef struct {
    hi_double filter[5][5];
    hi_size kernel_size;
    hi_point anchor;
    hi_double delta;
    hi_vpc_bord_type border_type;
    hi_vpc_scalar scalar_value;
    hi_u32 reserved[2];
} hi_filter_2d_config;
```

成员

成员名称	描述
filter	自定义卷积核。 由于filter是5*5的数组，当kernel_size内的宽、高不同时为5时，二维数组内的中间数据有效，例如： ● 宽为1、高为3时，下图中的黄区区域数据有效：  ● 宽为3、高为3时，下图中的黄色区域数据有效： 
kernel_size	卷积核尺寸，kernel_size的宽和高只能设置为1，3或者5。
anchor	卷积核锚点，预留字段，需要设置成(-1, -1)。

成员名称	描述
delta	增量值，会被加到执行卷积之后的每个像素值之上。
border_type	边界填充类型，只支持HI_BORDER_CONSTANT和HI_BORDER_REPLICATE两种填充模式。
scalar_value	边界填充值。
reserved[2]	预留参数，为保证后续版本兼容性，请务必使用memset结构体方式进行清零初始化，在代码中必须避免显式对reserved字段进行访问。

11.18.20.12.44 hi_mosaic_param

说明

定义马赛克信息结构体。

定义

```
typedef struct {
    hi_vpc_pic_info src;
    hi_vpc_pic_info dst;
    hi_u32 count;
    hi_mosaic *mosaic;
} hi_mosaic_param;
```

成员

成员名称	描述
src	输入图片信息。
dst	输出图片信息。
count	马赛克区域个数，取值范围[1, 100]。
mosaic	马赛克区域配置参数数组指针。

11.18.20.12.45 hi_rotate_param

说明

定义固定角度旋转信息结构体。

定义

```
typedef struct {
    hi_vpc_pic_info src;
    hi_vpc_pic_info dst;
    hi_rotation angle;
} hi_rotate_param;
```

成员

成员名称	描述
src	输入图片信息。
dst	输出图片信息。
angle	旋转角度。

11.18.20.12.46 hi_mosaic

说明

定义马赛克参数结构体。

定义

```
typedef struct {
    hi_blk_size blk_size;
    hi_rect rect;
} hi_mosaic;
```

成员

成员名称	描述
blk_size	单个马赛克的大小，只支持HI_BLK_SIZE_8、HI_BLK_SIZE_16、HI_BLK_SIZE_32。
rect	马赛克区域，其宽、高最小为8，可以部分区域在图片外。

11.18.20.12.47 hi_cover_type

说明

定义覆盖类型。

定义

```
typedef enum {
    HI_COVER_RECT = 0,
    HI_COVER_QUAD,
} hi_cover_type;
```

成员

成员名称	描述
HI_COVER_RECT	矩形覆盖。

成员名称	描述
HI_COVER_QUAD	任意四边形覆盖。

11.18.20.12.48 hi_cover_param

说明

定义覆盖信息结构体。

定义

```
typedef struct {
    hi_vpc_pic_info src;
    hi_vpc_pic_info dst;
    hi_u32 count;
    hi_cover *cover;
} hi_cover_param;
```

成员

成员名称	描述
src	输入图片信息。
dst	输出图片信息。
count	覆盖区域个数，取值范围[1, 100]。
cover	覆盖区域配置参数数组指针。

11.18.20.12.49 hi_cover

说明

定义覆盖参数结构体。

定义

```
typedef struct {
    hi_cover_type type;
    union {
        hi_rect rect;
        hi_quad_cover quad;
    };
    hi_u32 color;
} hi_cover;
```

成员

成员名称	描述
type	覆盖类型。
rect	覆盖区域。 当图片格式为YUV422SP、YUV420SP时，要求x、y、width、height值均为偶数。
quad	自定义覆盖区域。
color	覆盖区域的颜色值，低24位有效。

11.18.20.12.50 hi_quad_cover

说明

定义自定义覆盖区域结构体。

定义

```
typedef struct {
    hi_bool is_solid;
    hi_u32 thick;
    hi_point point[4];
} hi_quad_cover;
```

成员

成员名称	描述
is_solid	只支持is_solid设置为1。
thick	预留字段，暂不支持。
point[4]	四边形4个点的坐标。

11.18.20.12.51 hi_line_param

说明

定义画线信息结构体。

定义

```
typedef struct {
    hi_vpc_pic_info src;
    hi_vpc_pic_info dst;
    hi_u32 count;
    hi_line *line;
} hi_line_param;
```

成员

成员名称	描述
src	输入图片信息。
dst	输出图片信息。
count	画线条数，取值范围[1, 100]。
line	画线配置参数数组指针。 当图片格式为YUV422SP、YUV420SP时，要求起始点、结束点坐标值为偶数。

11.18.20.12.52 hi_line

说明

定义画线参数结构体。

定义

```
typedef struct {
    hi_point start_point;
    hi_point end_point;
    hi_u32 thick;
    hi_u32 color;
} hi_line;
```

成员

成员名称	描述
start_point	起始点坐标。
end_point	结束点坐标。 结束点坐标不能与起始点坐标相同。
thick	线的粗细值，取值需要2对齐。
color	线的颜色值，低24位有效。

11.18.20.12.53 hi_osd_param

说明

定义叠加信息结构体。

定义

```
typedef struct {
    hi_vpc_pic_info src;
    hi_vpc_pic_info dst;
```

```
    hi_u32 count;
    hi_osd *osd;
    hi_u32 clut[16];
} hi_osd_param;
```

成员

成员名称	描述
src	输入图片信息。
dst	输出图片信息。
count	叠加区域个数，取值范围[1, 100]。
osd	叠加区域配置参数数组指针。
clut	clut查找表系数。

11.18.20.12.54 hi_osd

说明

定义叠加参数结构体。

定义

```
typedef struct {
    hi_rect rect;
    hi_pixel_format pixel_format;
    hi_void* picture_address;
    hi_u32 stride;
    hi_u32 bg_alpha;
    hi_u32 fg_alpha;
    hi_osd_inverted_color osd_inverted_color;
} hi_osd;
```

成员

成员名称	描述
rect	叠加区域坐标。

pixel_format	被叠加图片的格式。
	支持如下格式：
	HI_PIXEL_FORMAT_ARGB_8888 = 14， // ARGB8888
	HI_PIXEL_FORMAT_ARGB_1555 = 33， // ARGB1555 A:1bit R:5bit G:5bit B:5bit
	HI_PIXEL_FORMAT_ARGB_4444 = 25， // ARGB4444 A:4bit R:4bit G:4bit B:4bit
	HI_PIXEL_FORMAT_ARGB_CLUT2 = 41， // ARGB Color Lookup Table 2bit
	HI_PIXEL_FORMAT_ARGB_CLUT4 = 42, // ARGB Color Lookup Table 4bit

picture_address	被叠加图片的地址。

成员名称	描述
stride	被叠加图片的stride。 ● 对于HI_PIXEL_FORMAT_ARGB_8888格式，stride值为rect.width的4倍 ● 对于HI_PIXEL_FORMAT_ARGB_1555/HI_PIXEL_FORMAT_ARGB_4444格式， stride值为rect.width的2倍 ● 对于HI_PIXEL_FORMAT_ARGB_CLUT2格式，stride值为rect.width的1/4倍 ● 对于HI_PIXEL_FORMAT_ARGB_CLUT4格式，stride值为rect.width的1/2倍
bg_alpha	背景alpha。 取值范围[0,255]。 仅当被叠加图片的格式为HI_PIXEL_FORMAT_ARGB_1555格式时，需设置该参数。
fg_alpha	前景alpha。 取值范围[0,255]。 仅当被叠加图片的格式为HI_PIXEL_FORMAT_ARGB_1555格式时，需设置该参数。
osd_inverted_color	反色类型。

11.18.20.12.55 hi_osd_inverted_color

说明

定义OSD反色类型。

定义

```
typedef enum {
    HI_OSD_INVERTED_COLOR_NONE = 0,
    HI_OSD_INVERTED_COLOR_RGB,
    HI_OSD_INVERTED_COLOR_ALPHA,
    HI_OSD_INVERTED_COLOR_ALL,
} hi_osd_inverted_color;
```

成员

成员名称	描述
HI_OSD_INVERTED_COLOR_NONE	不反色。

成员名称	描述
HI_OSD_INVERTED_COLOR_RGB	RGB通道反色。
HI_OSD_INVERTED_COLOR_ALPHA	ALPHA通道反色。
HI_OSD_INVERTED_COLOR_ALL	所有通道反色。

11.18.20.12.56 hi_transform_matrix

说明

定义仿射变换中的结构体。

定义

```
typedef struct {
    hi_matrix_type matrix_type;
    hi_float matrix[3][4];
} hi_transform_matrix;
```

成员

成员名称	描述
matrix_type	矩阵类型。
matrix	仿射变换矩阵。

11.18.20.12.57 hi_matrix_type

说明

定义仿射变换中的矩阵类型。

定义

```
typedef enum {
    HI_MATRIX_2_PLUS_3 = 0,
    HI_MATRIX_3_PLUS_3,
    HI_MATRIX_3_PLUS_4,
} hi_matrix_type;
```

成员

成员名称	描述
HI_MATRIX_2_PLUS_3	2*3 Matrix。

成员名称	描述
HI_MATRIX_3_PLUS_3	3*3 Matrix。
HI_MATRIX_3_PLUS_4	3*4 Matrix。

11.18.20.12.58 hi_flip_param

说明

定义翻转信息结构体。

定义

```
typedef struct {
    hi_vpc_pic_info src;
    hi_vpc_pic_info dst;
    hi_flip_mode flip_mode;
} hi_flip_param;
```

成员

成员名称	描述
src	输入图片信息。
dst	输出图片信息。
flip_mode	翻转模式参数。

11.18.20.12.59 hi_flip_mode

说明

定义翻转模式类型。

定义

```
typedef enum {
    HI_FLIP_HOR = 0,
    HI_FLIP_VER = 1,
    HI_FLIP_BOTH = 2
} hi_flip_mode;
```

成员

成员名称	描述
HI_FLIP_HOR	进行水平方向翻转。
HI_FLIP_VER	进行竖直方向翻转。

成员名称	描述
HI_FLIP_BOTH	同时进行水平方向翻转和竖直方向翻转，相当于180度旋转。

11.18.20.12.60 hi_opt_attr

说明

定义通道属性信息。

定义

```
typedef struct {
    hi_opt_attr_type attr_type;
    hi_u32 len;
    hi_void* value;
} hi_opt_attr;
```

成员

成员名称	描述
attr_type	属性类型。
len	属性值数据的长度，当前固定为4。
value	属性值，当前仅支持设置为0。

11.18.20.12.61 hi_opt_attr_type

说明

定义属性类型。

定义

```
typedef enum {
    HI_VPC_CHN_PRIORITY = 0
} hi_opt_attr_type;
```

成员

成员名称	描述
HI_VPC_CHN_PRIORITY	VPC通道优先级属性，当前仅支持设置为0，表示将通道设置为高优先级

11.18.20.13 VDEC 视频/JPEGD 图像解码

11.18.20.13.1 hi_vdec_chn

说明

描述解码通道。

定义

```
typedef hi_s32 hi_vdec_chn;
```

11.18.20.13.2 hi_vdec_send_mode

说明

定义码流发送方式。

定义

```
typedef enum {
    HI_VDEC_SEND_MODE_STREAM = 0,
    HI_VDEC_SEND_MODE_FRAME ,
    HI_VDEC_SEND_MODE_COMPAT,
    HI_VDEC_SEND_MODE_BUTT
}hi_vdec_send_mode;
```

成员

成员名称	描述
HI_VDEC_SEND_MODE_STREAM	流模式，当前不支持。
HI_VDEC_SEND_MODE_FRAME	帧模式，当前仅支持按帧发送。
HI_VDEC_SEND_MODE_COMPAT	兼容模式，当前不支持。
HI_VDEC_SEND_MODE_BUTT	保留值。

注意事项

1. 仅支持HI_VDEC_SEND_MODE_FRAME；
2. 当一帧码流的最后一包没有把end_of_frame为HI_TRUE时，还可以再发送一个end_of_frame为HI_TRUE的空包（注意带上当前帧的PTS（Presentation Time Stamp））给解码器内部标识当前帧码流已发送完毕。
3. 不支持一帧图像分多包进行发送的模式；

11.18.20.13.3 hi_vdec_frame_type

说明

定义码流发送方式。

定义

```
typedef enum {
    HI_VDEC_FRAME_TYPE_I = 0,
    HI_VDEC_FRAME_TYPE_P = 1,
    HI_VDEC_FRAME_TYPE_B = 2,
    HI_VDEC_FRAME_TYPE_BUTT
}hi_vdec_frame_type;
```

成员

成员名称	描述
HI_VDEC_FRAME_TYPE_I	IDR帧。
HI_VDEC_FRAME_TYPE_P	P帧。
HI_VDEC_FRAME_TYPE_B	B帧。
HI_VDEC_FRAME_TYPE_BUTT	保留值。

11.18.20.13.4 hi_quick_mark_mode

说明

定义快速释放参考帧模式枚举。

定义

```
typedef enum {
    HI_QUICK_MARK_ADAPT = 0,
    HI_QUICK_MARK_FORCE,
    HI_QUICK_MARK_NONE,
    HI_QUICK_MARK_BUTT
}hi_quick_mark_mode;
```

成员

成员名称	描述
HI_QUICK_MARK_ADAPT	自适应模式。预留参数，暂未支持。
HI_QUICK_MARK_FORCE	强制模式。预留参数，暂未支持。

成员名称	描述
HI_QUICK_MARK_NONE	普通模式。预留参数，暂未支持。
HI_QUICK_MARK_BUTT	保留值。

11.18.20.13.5 hi_vdec_pic_info

说明

定义视频原始图像帧结构。

定义

```
typedef struct {
    hi_u32 width;
    hi_u32 height;
    hi_u32 width_stride;
    hi_u32 height_stride;
    hi_pixel_format pixel_format;
    hi_u64 vir_addr;
    hi_u32 buffer_size;
    hi_s16 offset_top;
    hi_s16 offset_bottom;
    hi_s16 offset_left;
    hi_s16 offset_right;
}hi_vdec_pic_info;
```

成员

成员名称	描述
width	图像宽度，视频解码时使用该参数。 ● Atlas 推理系列产品，视频解码场景下，支持对输出图片进行缩放，缩放算法默认为业界通用的Bilinear算法（与OpenCV算法的计算过程类似），解码图像经过缩放后的输出宽取值范围为[10, 4096]。 以下场景，VDEC不对输出图片进行缩放：在hi_vdec_pic_info结构体内，将width参数值和height参数值同时设置为0。
height	图像高度，视频解码时使用该参数。 ● Atlas 推理系列产品，视频解码场景下，支持对输出图片进行缩放，缩放算法默认为业界通用的Bilinear算法（与OpenCV算法的计算过程类似），解码图像经过缩放后的输出高度范围为[6, 4096]。
width_stride	图像宽度步长，视频解码时，宽stride的配置约束请参见**11.18.17.3 VDEC功能及约束说明**。 取值范围：[32, 16384]。

成员名称	描述
height_stride	图像高度步长，视频解码时，高stride的配置约束请参见**11.18.17.3 VDEC功能及约束说明**。 取值范围：[6, 16384]。
pixel_format	图像像素格式。图像或视频解码时使用该参数。VDEC视频解码输出图片格式参见**11.18.17.3 VDEC功能及约束说明**，JPEGD图片解码输出图片格式参见**11.18.17.1 JPEGD功能及约束说明**。
vir_addr	解码输出数据在内存中的起始地址。 隔行码流场景下，隔行码流每帧发送两场，带两块输出内存；解码时其中一块内存无图像输出，属于正常现象，用户需及时释放内存；隔行码流的解码输出数据都在奇数场对应的输出buffer中。
buffer_size	存放解码图像的缓冲区大小。
offset_top	解码区域相对原图的顶部偏移，JPEG图片解码场景下按指定区域解码时配置，如**图11-24**所示，若无需按指定区域解码，需将此值初始化为0。 Atlas 推理系列产品不支持该参数，为保证后续版本兼容性，建议此参数配置为0。
offset_bottom	解码区域相对原图的底部偏移，JPEG图片解码场景下按指定区域解码时配置，如**图11-24**所示，若无需按指定区域解码，需将此值初始化为0。 Atlas 推理系列产品不支持该参数，为保证后续版本兼容性，建议此参数配置为0。
offset_left	解码区域相对原图的左侧偏移，JPEG图片解码场景下按指定区域解码时配置，如**图11-24**所示，若无需按指定区域解码，需将此值初始化为0。 Atlas 推理系列产品不支持该参数，为保证后续版本兼容性，建议此参数配置为0。
offset_right	解码区域相对原图的右侧偏移，JPEG图片解码场景下按指定区域解码时配置，如**图11-24**所示，若无需按指定区域解码，需将此值初始化为0。 Atlas 推理系列产品不支持该参数，为保证后续版本兼容性，建议此参数配置为0。

注意事项

10bit的数据经过VPC处理（图像处理）之后均会变成8bit。

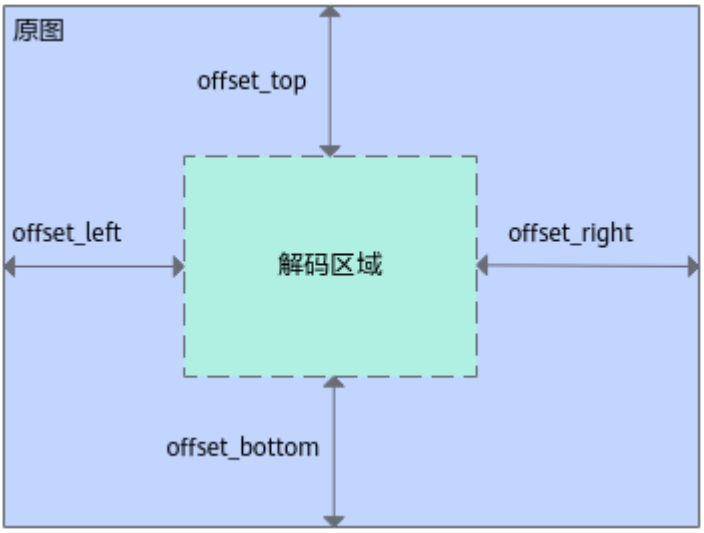
参考信息

JPEG图片解码场景下，按区域解码的功能示意图如下。

使用区域解码存在以下约束：

- 对于jpeg图片解码进行区域解码，不支持旋转、软件解码图片；
- 区域解码的范围不能超出原图大小，并且最小支持32*32，最大支持16384*16384；
- 使用以下图片格式时，对各偏移值的约束如下：
  - HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420或HI_PIXEL_FORMAT_YVU_SEMIPLANAR_420格式：offset_top和offset_left必须为偶数，（height - offset_bottom）和（width - offset_right）必须为偶数。
  - HI_PIXEL_FORMAT_YUV_SEMIPLANAR_422或HI_PIXEL_FORMAT_YVU_SEMIPLANAR_422格式：offset_left必须为偶数，（width - offset_right）必须为偶数。
  - HI_PIXEL_FORMAT_YUV_SEMIPLANAR_440或HI_PIXEL_FORMAT_YVU_SEMIPLANAR_440：offset_top必须为偶数，（height - offset_bottom）必须为偶数。

图 11-24 按区域解码功能示意图



11.18.20.13.6 hi_vdec_video_attr

说明

定义视频解码视频属性。

定义

```
typedef struct {
    hi_u32 ref_frame_num;
    hi_bool temporal_mvp_en;
    hi_u32 tmv_buf_size;
}hi_vdec_video_attr;
```

成员

成员名称	描述
ref_frame_num	参考帧的数目，用于决定解码时需要的参考帧个数，会较大的影响内存VB（Vedio Buffer）块占用，根据实际情况设置合适的值。如果此处设置的值大于实际码流中的参考帧数量，系统内部会根据实际码流的参考帧数量进行调整，以节省内存，但不超过当前所设置的参数值。 取值范围：[0, 16]，以帧为单位。 不知情的情况下，其他码流：推荐设为5。 测试码流：推荐设为16。 静态属性。
temporal_mvp_en	是否支持时域运动矢量预测。 取值范围：[0, 1]。 如果H.264解码不需要解码B帧，或者H.265解码不需要解码“支持时域运动矢量预测（sps_temporal_mvp_enabled_flag =1）的码流”，则配置temporal_mvp_en为0，否则配置为1。当配置为0时，可不分配输出TMV信息的VB块，节省内存。 **注意**：调用**hi_mpi_vdec_set_chn_param**接口设置通道参数时，只支持将hi_video_dec_mode设置为HI_VIDEO_DEC_MODE_IPB（表示解码I、P、B帧），因此解码H.264码流时只能将temporal_mvp_en配置为1。
tmv_buf_size	视频解码图像TMV（Temporal Motion Vector）缓存大小，以Byte为单位，temporal_mvp_en为1时有效。用户可先调用hi_vdec_get_tmv_buf_size接口获取视频解码图像TMV缓存大小。

11.18.20.13.7 hi_vdec_chn_attr

说明

定义解码通道属性结构体。

定义

```
typedef struct {
    hi_payload_type type;
    hi_vdec_send_mode mode;
    hi_u32 pic_width;
    hi_u32 pic_height;
    hi_u32 stream_buf_size;
    hi_u32 frame_buf_size;
    hi_u32 frame_buf_cnt;

    union {
        hi_vdec_video_attr video_attr;
    };
}hi_vdec_chn_attr;
```

成员

成员名称	描述
type	解码协议类型枚举值。 静态属性。
mode	码流发送方式。静态属性。 Atlas 推理系列产品上，H.264/H.265/JPEG仅支持按帧发送码流。
pic_width	通道支持的解码图像最大宽（以像素为单位）。静态属性。
pic_height	通道支持的解码图像最大高（以像素为单位）。静态属性。
stream_buf_size	输入码流缓存大小。 取值范围：大于或等于解码通道大小（宽*高）的 3/4倍，即YUV420图像大小的一半（宽*高*3/2*1/2），小于或等于1024*1024*1024，以 Byte 为单位。 推荐值：一幅YUV420解码图像大小。即：宽*高*1.5。 静态属性。
frame_buf_size	解码图像帧存大小，以Byte为单位。JPEGD图像解码不需要设置该参数。 取值范围：大于或等于0，取值为0时，VDEC内部将根据实际码流信息自动分配合适的帧存大小；取值大于0时，用户需保证（帧存大小*参考帧个数 >= 实际所需最小帧存大小*实际所需最小参考帧个数），否则不能正常解码。实际所需最小帧存大小，用户需先根据码流中的SPS（Sequence Parameter Set）信息获取输入码流的宽、高、比特位宽，再调用hi_vdec_get_pic_buf_size接口获取实际所需最小帧存大小。
frame_buf_cnt	解码图像帧存个数。JPEGD图像解码不需要设置该参数。 取值范围：大于或等于0。 ● 一般情况下，设置为0，VDEC内部将根据实际码流信息自动分配合适的帧存个数； ● 如果客户需要限制VDEC解码帧存大小，则设置为大于0的值，用户需保证（帧存大小*参考帧个数 >= 实际需要最小帧存大小*实际需要最小参考帧个数），否则不能正常解码。实际所需最小参考帧个数，用户需先根据码流中的SPS（Sequence Parameter Set）信息获取码流参考帧个数，H.264/H.265解码所需帧存个数=参考帧+显示帧+1。
video_attr	视频(H.264/H.265)解码通道属性。

注意事项

设置通道属性的图像宽和高，若用户清楚码流的具体宽高，建议以实际值来设置。若是不清楚，建议以此码流的最大规格来创建。但是最大规格是4096*4096，不能超过此值。

11.18.20.13.8 hi_h264_protocol_param

说明

与 H.264 协议相关的内存分配参数。

定义

```
typedef struct {
    hi_s32 max_slice_num;
    hi_s32 max_sps_num;
    hi_s32 max_pps_num;
}hi_h264_protocol_param;
```

成员

成员名称	描述
max_slice_num	该通道解码支持的最大Slice个数。 取值范围：[1, 300]。 Default：16。
max_sps_num	该通道解码支持的最大SPS个数。 取值范围：[1, 32]。 Default：2。
max_pps_num	该通道解码支持的最大PPS个数。 取值范围：[1, 256]。 Default：2。

11.18.20.13.9 hi_h265_protocol_param

说明

定义与 H.265 协议相关的内存分配参数。

定义

```
typedef struct {
    hi_s32 max_slice_segment_num;
    hi_s32 max_vps_num;
    hi_s32 max_sps_num;
    hi_s32 max_pps_num;
}hi_h265_protocol_param;
```

成员

成员名称	描述
max_slice_segment_num	该通道解码支持的最大SliceSegment个数。 取值范围：[1, 600] Default：16。
max_vps_num	该通道解码支持的最大VPS个数。 取值范围：[1, 16]。 Default：2。
max_sps_num	该通道解码支持的最大SPS个数。 取值范围：[1, 16]。 Default：2。
max_pps_num	该通道解码支持的最大PPS个数。 取值范围：[1, 64]。 Default：2。

11.18.20.13.10 hi_vdec_protocol_param

说明

定义与协议相关的内存分配参数。

定义

```
typedef struct {
    hi_payload_type type;
    union
    {
        hi_h264_protocol_param h264_param;
        hi_h265_protocol_param h265_param;
    };
}hi_vdec_protocol_param;
```

成员

成员名称	描述
type	解码通道支持的协议。
h264_param	H.264协议参数。
h265_param	H.265协议参数。

11.18.20.13.11 hi_video_dec_mode

说明

定义视频解码模式枚举。

定义

```
typedef enum {
    HI_VIDEO_DEC_MODE_IPB = 0,
    HI_VIDEO_DEC_MODE_IP,
    HI_VIDEO_DEC_MODE_I,
    HI_VIDEO_DEC_MODE_BUTT
}hi_video_dec_mode;
```

成员

成员名称	描述
HI_VIDEO_DEC_MODE_IPB	IPB模式，即I、P、B帧都解码。
HI_VIDEO_DEC_MODE_IP	IP模式，即只解码I帧和P帧。暂不支持
HI_VIDEO_DEC_MODE_I	I模式，即只解码I帧。暂不支持
HI_VIDEO_DEC_MODE_BUTT	保留值。

11.18.20.13.12 hi_video_out_order

说明

定义视频解码输出顺序枚举。

定义

```
typedef enum {
    HI_VIDEO_OUT_ORDER_DISPLAY = 0,
    HI_VIDEO_OUT_ORDER_DEC,
    HI_VIDEO_OUT_ORDER_BUTT
}hi_video_out_order;
```

成员

成员名称	描述
HI_VIDEO_OUT_ORDER_DISPLAY	显示序输出。
HI_VIDEO_OUT_ORDER_DEC	解码序输出。
HI_VIDEO_OUT_ORDER_BUTT	保留值。

注意事项

解码有 B 帧的码流应设置为显示序输出。

11.18.20.13.13 hi_vdec_video_param

说明

定义视频解码高级参数。

定义

```
typedef struct {
    hi_bool composite_dec_en;
    hi_bool slice_input_en;
    hi_s32 err_threshold;
    hi_video_dec_mode dec_mode;
    hi_video_out_order out_order;
    hi_compress_mode compress_mode;
    hi_video_format video_format;
    hi_quick_mark_mode quick_mark_mode;
} hi_vdec_video_param;
```

成员

成员名称	描述
composite_dec_en	复合码流使能开关。预留参数，暂未支持。
slice_input_en	预留参数，暂未支持。
err_threshold	预留参数，暂未支持。
dec_mode	解码模式。 Default：HI_VIDEO_DEC_MODE_IPB。
out_order	解码图像输出顺序。 Default：HI_VIDEO_OUT_ORDER_DISPLAY
compress_mode	解码图像压缩模式。支持HFBC压缩HI_COMPRESS_MODE_HFBC。 Default：HI_COMPRESS_MODE_HFBC
video_format	解码图像数据格式。 当前仅支持HI_VIDEO_FORMAT_TILE_64x16和HI_VIDEO_FORMAT_LINEAR。 Default：HI_VIDEO_FORMAT_TILE_64x16
quick_mark_mode	快速释放参考帧模式。预留参数，暂未支持。

11.18.20.13.14 hi_vdec_pic_param

说明

定义图像解码高级参数。

定义

```
typedef struct {
    hi_pixel_format  pixel_format;
    hi_u32 alpha;
}hi_vdec_pic_param;
```

成员

成员名称	描述
pixel_format	图片像素格式。预留参数。
alpha	RGB解码透明度。预留参数。

11.18.20.13.15 hi_vdec_chn_param

说明

定义解码通道高级参数。

定义

```
typedef struct {
    hi_payload_type type;
    hi_u32 display_frame_num;
    union{
        hi_vdec_video_param video_param;
        hi_vdec_pic_param pic_param;
    };
}hi_vdec_chn_param;
```

成员

成员名称	描述
type	解码协议。
display_frame_num	解码后缓存图像帧数。 取值范围：[0, 16]。 Default：2。
video_param	视频(H.264/H.265)解码高级参数。
pic_param	图片(JPEG)解码高级参数。

11.18.20.13.16 hi_vdec_dec_err

说明

定义解码错误信息结构体。

定义

```
typedef struct {
    hi_s32 set_pic_size_err;
    hi_s32 set_protocol_num_err;
    hi_s32 set_ref_num_err;
    hi_s32 set_pic_buf_size_err;
    hi_s32 format_err;
    hi_s32 stream_unsupport;
    hi_s32 pack_err;
    hi_s32 stream_size_over;
    hi_s32 stream_not_release;
} hi_vdec_dec_err;
```

成员

成员	描述
set_pic_size_err	图像的宽（或高）比通道的宽（或高）大，预留参数。
set_protocol_num_err	设置的协议参数个数不够。比如：Slice/PPS/SPS个数。此参数仅对H.264/H.265解码有效。
set_ref_num_err	设置的参考帧个数不够。此参数仅对H.264/H.265解码有效。
set_pic_buf_size_err	图像buffer内存大小不够。
format_err	不支持的格式。JPEG的Progressive码流。
stream_unsupport	不支持的规格（码流规格与昇腾AI处理器宣称支持的规格不一致）。
pack_err	码流有错。
stream_size_over	一帧码流太大了，当整个SCD（Start Code Detection） buffer都装不下一帧码流时，强制清空SCDbuffer。 此参数仅对H.264/H.265解码有效。
stream_not_release	VFMW（video firmware）内部管理码流错误，出现长时间不释放码流的情况，预留参数。

11.18.20.13.17 hi_vdec_chn_status

说明

定义通道状态结构体。

定义

```
typedef struct {
    hi_payload_type type;
```

```
    hi_u32 left_stream_bytes;
    hi_u32 left_stream_frames;
    hi_u32 left_decoded_frames;
    hi_bool is_started;
    hi_u32 recv_stream_frames;
    hi_u32 dec_stream_frames;
    hi_vdec_dec_err dec_err;
    hi_u32 width;
    hi_u32 height;
    hi_u64 latest_frame_pts;
} hi_vdec_chn_status;
```

成员

成员名称	描述
type	解码协议。
left_stream_bytes	码流buffer中待解码的byte数，包括正在解码的当前帧中未解码的byte数。
left_stream_frames	码流buffer中待解码的帧数，不包括正在解码的当前帧。-1表示无效。仅按帧发送时有效。
left_decoded_frames	图像buffer中剩余的图片数目。
is_started	解码器是否已经启动接收码流。
recv_stream_frames	码流buffer中已接收码流帧数。-1表示无效。仅按帧发送时有效。
dec_stream_frames	码流buffer中已解码帧数。仅按帧发送时有效。
dec_err	解码错误信息。
width	解码后输出图片的宽。
height	解码后输出图片的高。
latest_frame_pts	最新解码图像的时间戳。预留参数，暂不支持。

11.18.20.13.18 hi_vdec_stream

说明

定义视频/图像解码的码流结构体。

定义

```
typedef struct {
    hi_bool end_of_frame;
    hi_bool end_of_stream;
    hi_bool need_display;
    hi_u64 pts;
    hi_u64 private_data;
    hi_u32 len;
    hi_u8 *ATTRIBUTE addr;
} hi_vdec_stream;
```

成员

成员	描述
end_of_frame	当前帧是否结束（此值目前暂无效）。
end_of_stream	是否发完所有码流。
need_display	当前帧输出是否显示。 ● 0：不显示，设置为0，输出Buffer无解码结果，且**不显示帧**按照显示序或解码序输出。 ● 1：显示，设置为1，才能输出解码结果，且**显示帧**按照显示序或解码序输出。 **但**如果解码码流中同时存在不显示帧、显示帧，则系统不保证不显示帧与显示帧之间按照显示序或解码序输出。
pts	码流包的时间戳，以us为单位。仅按帧发送时有效。
private_data	私有数据。预留参数，暂不支持。
len	码流包的长度，以Byte为单位。
addr	码流包的地址。

注意事项

1. 按帧发送时，解码图像的时间戳等于码流包中的时间戳。
2. 按流发送时，解码图像的时间戳等于0。
3. 当发完所有码流后，把end_of_stream设置为1，表示码流文件结束，这时解码器会解完发送下来的所有码流并输出所有图像。如果发完所有码流后把end_of_stream设置为0，解码器内部可能残余大于等于一帧的图像未解码输出，因为解码器必须等到下一帧码流到来才能知道当前帧已经结束，送入解码。
4. VDEC支持发送一包end_of_stream为1的空码流包（地址为空或长度为 0）。

11.18.20.13.19 hi_vdec_supplement_info

说明

定义输出帧补充信息， 暂不支持。

定义

```
typedef struct {
    hi_payload_type type;
    union {
        hi_vdec_video_supplement_info video_supplement_info;
    };
} hi_vdec_supplement_info;
```

11.18.20.13.20 hi_vdec_video_supplement_info

说明

定义输出视频帧补充信息， 暂不支持。

定义

```
typedef struct {
    hi_vdec_frame_type frame_type;
    hi_u32 err_rate;
    hi_u32 poc;
}hi_vdec_video_supplement_info;
```

成员

成员名称	描述
frame_type	解码帧类型。预留参数，暂不支持。
err_rate	错误率。预留参数，暂不支持。
poc	poc值。预留参数，暂不支持。

11.18.20.13.21 hi_jpegd_precision_mode

说明

定义JPEGD解码输出图片的宽、高对齐模式。

定义

```
typedef enum {
    YUVOUT_ALIGN_DOWN = 0,
    YUVOUT_ALIGN_UP = 1,
    YUVOUT_ALIGN_DOWN_COMPAT = 2,
} hi_jpegd_precision_mode;
```

对齐模式取值的含义如下：

- YUVOUT_ALIGN_DOWN：默认值
  - 解码后的输出图片格式为YUV420SP时，若源图的宽、高为奇数时，将输出图片的宽、高向下2对齐
  - 解码后的输出图片格式为YUV422SP时，若源图的宽为奇数时，将输出图片的宽向下2对齐
  - 解码后的输出图片格式为YUV440SP时，若源图的高为奇数时，将输出图片高向下2对齐
- YUVOUT_ALIGN_UP：
  - 解码后的输出图片格式为YUV420SP时，若源图的宽、高为奇数时，将输出图片的宽、高向上2对齐
  - 解码后的输出图片格式为YUV422SP时，若源图的宽为奇数时，将输出图片的宽向上2对齐

  - 解码后的输出图片格式为YUV440SP时，若源图的高为奇数时，将输出图片高向上2对齐
- YUVOUT_ALIGN_DOWN_COMPAT：

  兼容旧版本，解码后的输出图片格式为YUV420SP/YUV422SP/YUV440SP时，若源图的宽、高为奇数时，将输出图片的宽、高向下2对齐。

11.18.20.14 VENC 视频/JPEGE 图像编码

11.18.20.14.1 hi_venc_chn

说明

描述编码通道。

定义

```
typedef hi_s32 hi_venc_chn;
```

11.18.20.14.2 hi_venc_stream

说明

定义帧码流类型结构体。

定义

```
typedef struct {
    hi_venc_pack ATTRIBUTE* pack;
    hi_u32       ATTRIBUTE pack_cnt;
    hi_u32       seq;

    union {
        hi_venc_h264_stream_info h264_info;
        hi_venc_jpeg_stream_info jpeg_info;
        hi_venc_h265_stream_info h265_info;
        hi_venc_prores_stream_info prores_info;
    };

    union {
        hi_venc_h264_adv_stream_info h264_adv_info;
        hi_venc_h265_adv_stream_info h265_adv_info;
    };
} hi_venc_stream;
```

成员

成员名称	描述
pack	帧码流包结构。
pack_cnt	一帧码流的所有包的个数。
seq	码流序列号。 按帧获取时，此处为帧序号；按包获取时，此处为包序号。

成员名称	描述
h264_info/h265_info	码流特征信息。 Atlas 推理系列产品不支持该参数。
jpeg_info/prores_info	码流特征信息，暂不支持。
h264_adv_info/ h265_adv_info	码流高级特征信息。 Atlas 推理系列产品不支持该参数。

11.18.20.14.3 hi_venc_pack

说明

定义帧码流包结构体。

定义

```
typedef struct {
    union {
        hi_u64 phys_addr;
        hi_u64 input_addr;
    };
    hi_u8 ATTRIBUTE* addr;
    hi_u32 ATTRIBUTE len;
    hi_u64 pts;
    hi_bool is_frame_end;
    hi_venc_data_type data_type;
    hi_u32 offset;
    hi_u32 data_num;
    hi_venc_pack_info pack_info[HI_VENC_MAX_PACK_INFO_NUM];
} hi_venc_pack;
```

成员

成员名称	描述
phys_addr	码流包物理地址。
input_addr	用户输入码流数据的内存地址。
addr	码流包首地址。
len	码流包长度。 当len=0，表示视频/图像编码失败。
pts	时间戳。单位：us。
is_frame_end	帧结束标识。 取值范围： ● HI_TRUE：该码流包是该帧的最后一个包。 ● HI_FALSE：该码流包不是该帧的最后一个包。
data_type	码流类型，支持 H.265/H.264/JPEG协议类型的数据包。

成员名称	描述
offset	码流包中有效数据与码流包首地址addr的偏移。
data_num	当前码流包（当前包的类型由 data_type 指定）数据中包含码流包的个数。
pack_info[HI_VENC_MAX_PACK_INFO_NUM]	当前码流包数据中包含码流包数据信息。 #define HI_VENC_MAX_PACK_INFO_NUM 8

11.18.20.14.4 hi_venc_chn_attr

说明

定义编码通道属性结构体。

定义

```
typedef struct {
    hi_venc_attr venc_attr;
    hi_venc_rc_attr rc_attr;
    hi_venc_gop_attr gop_attr;
}hi_venc_chn_attr;
```

成员

成员名称	描述
venc_attr	编码器属性。 编码器属性中除通道宽高（pic_width和pic_height）外都是静态属性，一旦创建编码通道成功，静态属性不支持被修改，除非该通道被销毁，重新创建。
rc_attr	码率控制器属性。 码率控制器属性首先需要配置RC（Rate Control）模式，JPEG不需要配置码率控制器属性，其它协议类型通道（H.264/H.265）都必须配置。码率控制器属性RC模式必须与编码器属性协议类型匹配。
gop_attr	GOP Mode类型的结构体。 对于H264、H265协议，需要配置该参数。

11.18.20.14.5 hi_venc_attr

说明

定义编码器属性结构体。

定义

```
typedef struct {
    hi_payload_type type;
    hi_u32 max_pic_width;
    hi_u32 max_pic_height;
    hi_u32 buf_size;
    hi_u32 profile;
    hi_bool is_by_frame;
    hi_u32 pic_width;
    hi_u32 pic_height;
    union {
        hi_venc_h264_attr h264_attr;
        hi_venc_h265_attr h265_attr;
        hi_venc_jpeg_attr jpeg_attr;
        hi_venc_prores_attr prores_attr;
    };
}hi_venc_attr;
```

成员

成员名称	描述
type	编码协议类型。
max_pic_width	编码图像最大宽度，静态属性，必须是2的整数倍。 取值范围：[MIN_WIDTH, MAX_WIDTH]，以像素为单位。MIN_WIDTH、MAX_WIDTH，分别表示编码通道支持的最小宽度、最大宽度，参见**表11-60**。
max_pic_height	除jpeg协议外，编码图像最大高度，静态属性，必须是2的整数倍。 取值范围：[MIN_HEIGHT, MAX_HEIGHT]，以像素为单位。MIN_HEIGHT、MAX_HEIGHT分别表示编码通道支持的最小高度、最大高度，参见**表11-60**。

成员名称	描述
buf_size	码流buffer大小，单位为Byte，静态属性。 ● 调用**hi_mpi_venc_send_jpege_frame**接口发送原始图像进行图像编码时，在创建通道时，buf_size参数值必须设置为0。 ● 调用**hi_mpi_venc_send_frame**接口发送原始图像进行视频或图像编码时，在创建通道时，buf_size参数值必须设置为64的整数倍，取值范围：[Min, Max]，以Byte为单位。 （1）**视频编码场景下**，Min当前表示32*1024，Max当前表示1*1024*1024*1024。 用户可以在创建通道时，合理设置每个通道的buf_size，以节省内存使用开销，**推荐将buf_size设置为：原图宽*原图高*3/2后再64对齐**，但buf_size依然需要在[Min, Max]范围内。 视频编码场景下，若buf_size设置的不合理，可能出现以下异常情况： - 反复重编，进而导致编码时延变长、性能下降、图像质量下降； - 编码失败，获取不到编码结果。 （2）**图像编码场景下**，如果原图格式为YUV420，Min值为原图宽16对齐*原图高16对齐*3/2；如果原图格式为YUV422 Packed，Min值为原图宽16对齐*原图高16对齐*2。Max当前表示1*1024*1024*1024。 用户可以在创建通道时根据业务场景中图片的实际最大分辨率，合理设置每个通道的buf_size，以节省内存使用开销，**推荐将buf_size设置为：原图宽*原图高*倍数**，倍数推荐为5。 例如：用户图片的分辨率有720p（1280*720）、1080p（1080*1920）、4K（3840*2160）大小的图片，需要的buf_size应按照4k配置：3840*2160*5。 图片编码场景下，buf_size最小值为320*240*5，防止小分辨率场景下，buffer容易被占满，影响编码功能执行，例如：用户图片的最大分辨率为128*128，推荐用户将buf_size设置为320*240*5。
profile	编码的等级，数字越大编码质量越好，静态属性。 ● H.264 取值范围：[0, 3]。 – 0：Baseline。当取值为该值时，不支持编码B帧。 – 1：Main Profile。 – 2：High Profile。 ● H.265 取值范围：[0,1]。 – 0：Main Profile。 – 1：Main 10 Profile，预留值，暂不支持。 ● Jpeg 取值范围：0（表示Baseline）。
is_by_frame	帧/包模式获取码流，静态属性。 ● HI_TRUE：按帧获取。 ● HI_FALSE：按包获取。

成员名称	描述
pic_width	编码通道宽度，必须是MIN_ALIGN的整数倍。通道创建后，不支持修改。 取值范围：[MIN_WIDTH, MAX_WIDTH]，以像素为单位。MIN_WIDTH、MAX_WIDTH、MIN_ALIGN分别表示编码通道支持的最小宽度、最大宽度、最小对齐单元（像素），参见**表11-60**。 JPEG通道宽高的设置要满足：pic_width*pic_height<= max_pic_width*max_pic_height。
pic_height	除jpeg协议外，编码通道高度，必须是MIN_ALIGN的整数倍。通道创建后，不支持修改。 取值范围：[MIN_HEIGHT, MAX_HEIGHT]，以像素为单位。MIN_HEIGHT、MAX_HEIGHT、MIN_ALIGN分别表示编码通道支持的最小高度、最大高度、最小对齐单元（像素），参见**表11-60**。 JPEG通道宽高的设置要满足：pic_width*pic_height<= max_pic_width*max_pic_height。
h264_attr	编码协议的属性。 Atlas 推理系列产品，预留参数，暂不支持。
h265_attr	编码协议的属性。 Atlas 推理系列产品，预留参数，暂不支持。
jpeg_attr	编码协议的属性。 预留参数，暂不支持。
prores_attr	编码协议的属性。 预留参数，暂不支持。

参考信息

MIN_WIDTH、MAX_WIDTH、MIN_HEIGHT、MAX_HEIGHT、MIN_ALIGN分别表示编码通道支持的最小宽度、最大宽度、最小高度、最大高度、最小对齐单元（像素）。

表 11-60 编码通道宽高

W/H (pixel)	H.264	H.265	JPEG
MIN_WIDTH	128	128	32
MAX_WIDTH	4096	4096	8192
MIN_HEIGHT	128	128	32
MAX_HEIGHT	4096	4096	8192
MIN_ALIGN	2	2	2

11.18.20.14.6 hi_venc_rc_attr

说明

定义编码通道码率控制器属性。

定义

```
typedef struct {
    hi_venc_rc_mode rc_mode;
    union {
        hi_venc_h264_cbr h264_cbr;
        hi_venc_h264_vbr h264_vbr;
        hi_venc_h264_avbr h264_avbr;
        hi_venc_h264_qvbr h264_qvbr;
        hi_venc_h264_cvbr h264_cvbr;
        hi_venc_h264_fixqp h264_fixqp;
        hi_venc_h264_qpmap h264_qpmap;

        hi_venc_mjpeg_cbr mjpeg_cbr;
        hi_venc_mjpeg_vbr mjpeg_vbr;
        hi_venc_mjpeg_fixqp mjpeg_fixqp;

        hi_venc_h265_cbr h265_cbr;
        hi_venc_h265_vbr h265_vbr;
        hi_venc_h265_avbr h265_avbr;
        hi_venc_h265_qvbr h265_qvbr;
        hi_venc_h265_cvbr h265_cvbr;
        hi_venc_h265_fixqp h265_fixqp;
        hi_venc_h265_qpmap h265_qpmap;
    };
} hi_venc_rc_attr;
```

成员

成员名称	描述
rc_mode	RC模式。 ● CBR（Constant Bit Rate）固定比特率。即在码率统计时间内保证编码码率平稳。 ● VBR（Variable Bit Rate）可变比特率，即允许在码率统计时间内编码码率波动，从而保证编码图像质量平稳。 ● AVBR（Adaptive Variable Bit Rate） 自适应可变比特率，即允许在码率统计时间内编码码率波动，从而保证编码图像质量平稳。码率控制内部会检测当前场景的运动静止状态，在运动时用较高码率编码，在静止时主动降低目标码率。 ● QVBR (Quality Variable Bit Rate)基于主观图像质量的可变比特率，该码控调节方式是利用实时统计的 PSNR（图像质量客观评价指标）的大小来动态调整码率，从而保证编码图像质量平稳。在PSNR较小时主动升高目标码率，PSNR较大时主动降低目标码率。 ● CVBR (Constrained Variable Bit Rate) 是以VBR为基础，旨在提供平稳的图像质量的码控算法，同时对VBR的码率进行限制，以满足传输带宽以及存储空间的要求。具体来说， CVBR设置了瞬时，短期与长期码率的限制。其中，瞬时码率的限制保证了网络带宽对传输的要求；长期码率限制保证了在长时间视频录制时，存储设备有足够的空间储存数据；同时，短期码率会根据长期码率的设置和实际使用情况进行调节，以在场景复杂的情况下提供更加平稳的图像质量，并在场景简单时节省码率。 ● FIXQP（Fix Quantisation Parameter）固定QP值。在码率统计时间内，编码图像所有宏块QP值相同，采用用户设定的图像QP值，I帧和P帧的QP值可以分别设置。 并且对于不同协议，相同 RC 模式的属性变量基本一致。**表11-61**介绍了几种RC模式的公共的属性。
h264_cbr	H.264协议编码通道CBR模式属性。
h264_vbr	H.264协议编码通道VBR模式属性。
h264_avbr	H.264协议编码通道AVBR模式属性。
h264_qvbr	H.264协议编码通道QVBR模式属性。
h264_cvbr	H.264协议编码通道CVBR模式属性。
h264_fixqp	H.264协议编码通道FIXQP模式属性。 Atlas 推理系列产品，预留参数，暂不支持。

成员名称	描述
h264_qpmap	H.264协议编码通道QPMAP模式属性。预留参数，暂不支持。
mjpeg_cbr	Mjpeg协议编码通道CBR模式属性。预留参数，暂不支持。
mjpeg_vbr	Mjpeg协议编码通道VBR模式属性。预留参数，暂不支持。
mjpeg_fixqp	Mjpeg（Motion Joint Photographic Experts Group）协议编码通道FIXQP模式属性。预留参数，暂不支持。
h265_cbr	H.265协议编码通道CBR模式属性。
h265_vbr	H.265协议编码通道VBR模式属性。
h265_avbr	H.265协议编码通道AVBR模式属性。
h265_qvbr	H.265协议编码通道QVBR模式属性。
h265_cvbr	H.265协议编码通道CVBR模式属性。
h265_fixqp	H.265协议编码通道FIXQP模式属性。 Atlas 推理系列产品，预留参数，暂不支持。
h265_qpmap	H.265协议编码通道QPMAP模式属性。预留参数，暂不支持。

参考信息

表 11-61 码率控制器属性的约束

RC 模式	GOP	StatTime(s)	FrmRate(src_frame_rate/dst_frame_rate）
CBR	≥1	≥1	src_frame_rate ≥ dst_frame_rate
VBR	≥1	≥1	src_frame_rate ≥ dst_frame_rate
AVBR	≥1	≥1	src_frame_rate ≥ dst_frame_rate
FIXQP	≥1	≥1	src_frame_rate ≥ dst_frame_rate
QPMAP	≥1	≥1	src_frame_rate ≥ dst_frame_rate

11.18.20.14.7 hi_venc_rc_mode

说明

定义编码通道码率控制器模式。

定义

```
typedef enum {
    HI_VENC_RC_MODE_H264_CBR = 1,
    HI_VENC_RC_MODE_H264_VBR,
    HI_VENC_RC_MODE_H264_AVBR,
    HI_VENC_RC_MODE_H264_QVBR,
    HI_VENC_RC_MODE_H264_CVBR,
    HI_VENC_RC_MODE_H264_FIXQP,
    HI_VENC_RC_MODE_H264_QPMAP,

    HI_VENC_RC_MODE_MJPEG_CBR,
    HI_VENC_RC_MODE_MJPEG_VBR,
    HI_VENC_RC_MODE_MJPEG_FIXQP,

    HI_VENC_RC_MODE_H265_CBR,
    HI_VENC_RC_MODE_H265_VBR,
    HI_VENC_RC_MODE_H265_AVBR,
    HI_VENC_RC_MODE_H265_QVBR,
    HI_VENC_RC_MODE_H265_CVBR,
    HI_VENC_RC_MODE_H265_FIXQP,
    HI_VENC_RC_MODE_H265_QPMAP,

    HI_VENC_RC_MODE_BUTT,
} hi_venc_rc_mode;
```

成员

成员名称	描述
HI_VENC_RC_MODE_H264_CBR	H.264 CBR模式。
HI_VENC_RC_MODE_H264_VBR	H.264 VBR模式。
HI_VENC_RC_MODE_H264_AVBR	H.264 AVBR模式。
HI_VENC_RC_MODE_H264_QVBR	H.264 QVBR模式。
HI_VENC_RC_MODE_H264_CVBR	H.264 CVBR模式。
HI_VENC_RC_MODE_H264_FIXQP	H.264 FIXQP模式。 Atlas 推理系列产品，预留参数，暂不支持。
HI_VENC_RC_MODE_H264_QPMAP	H.264 QPMAP模式。预留参数，暂不支持。
HI_VENC_RC_MODE_MJPEG_CBR	MJPEG CBR模式。预留参数，暂不支持。
HI_VENC_RC_MODE_MJPEG_VBR	MJPEG VBR模式。预留参数，暂不支持。
HI_VENC_RC_MODE_MJPEG_FIXQP	MJPEG FIXQP模式。预留参数，暂不支持。
HI_VENC_RC_MODE_H265_CBR	H.265 CBR模式。
HI_VENC_RC_MODE_H265_VBR	H.265 VBR模式。
HI_VENC_RC_MODE_H265_AVBR	H.265 AVBR模式。

成员名称	描述
HI_VENC_RC_MODE_H265_QVBR	H.265 QVBR模式。
HI_VENC_RC_MODE_H265_CVBR	H.265 CVBR模式。
HI_VENC_RC_MODE_H265_FIXQP	H.265 FIXQP模式。 Atlas 推理系列产品，预留参数，暂不支持。
HI_VENC_RC_MODE_H265_QPMAP	H.265 QPMAP模式。预留参数，暂不支持。

11.18.20.14.8 hi_venc_h264_cbr

说明

定义H.264编码通道CBR属性结构。

定义

```
typedef struct {
    hi_u32 gop;
    hi_u32 stats_time;
    hi_u32 src_frame_rate;
    hi_u32 dst_frame_rate;
    hi_u32 bit_rate;
} hi_venc_h264_cbr;
```

成员

成员名称	描述
gop	Group Of Pictures， 视频序列由若干时间连续的图像构成，在对其进行编码时，先将该视频序列分割成若干个小的图像组，该参数即指一个图像组的帧数；由于GOP结构仅支持Normalp（第一帧为I帧，其余帧为P帧），此参数也为I帧间隔，默认值65535；由于编码I帧不需参考之前的帧，之前帧的编码质量影响从此帧开始不再传递，故该值越小，编码质量越好；但不建议小于帧率，因为I帧仅有帧内预测块，大部分场景需要有一定的P帧（有帧间预测块）才能保证编码质量。 取值范围：[1, 65536]。

成员名称	描述
stats_time	CBR码率统计时间，以秒为单位，一般场景下，stats_time= gop / dst_frame_rate，编码质量较好。 码率统计时间越长，每帧图像的码率波动对于码率调节的影响越弱，码率的调节会更缓慢，图像质量的波动会更轻微；码率统计时间越短，每帧图像的码率波动对于码率调节的影响越强，图像码率的调节会更灵敏，图像质量的波动会更剧烈。 取值范围：[1, 60]。
src_frame_rate	编码器输入帧率，以fps为单位，即每秒进入编码器的帧数，默认值为30；码率一定的情况下，该值越小，编码质量越好，但小于25播放时画面会不连贯。 该参数取值范围：[1, 240]。
dst_frame_rate	编码器输出帧率，以fps为单位，即每秒编码器输出的帧数，默认值为30；码率一定的情况下，该值越小，编码质量越好，但小于25播放时画面会不连贯。 取值范围：[1, src_frame_rate]。 暂不支持分数帧率。
bit_rate	编码器输出平均码率，以kbps为单位，即编码码流每秒比特数，该值越大，编码质量越好；默认值2000。 该参数取值范围：[2, 614400]。 平均比特率的设置与编码通道宽高以及图像帧率都有关系。典型的平均比特率的设置如**表11-62**所示。注意，**表11-62**中的平均比特率的设置是在通道编码帧率为满帧率（30fps）时的设置。当用户设置编码输出帧率不为满帧率时，可以对下表中的码率按用户设置帧率与满帧率（30fps）的比例进行换算。

表 11-62 典型平均比特率配置

图像宽高/码率等级	1080p(1920x1080)	4K(3840x2160)
低码率	＜2Mbps	＜8Mbps
中码率	2Mbps～8Mbps	8Mbps～30Mbps
高码率	>8Mbps	>30Mbps

11.18.20.14.9 hi_venc_h264_vbr

说明

定义H.264编码通道VBR属性结构。

定义

```
typedef struct {
    hi_u32 gop;
    hi_u32 stats_time;
    hi_u32 src_frame_rate;
    hi_u32 dst_frame_rate;
    hi_u32 max_bit_rate;
}hi_venc_h264_vbr;
```

成员

成员名称	描述
gop	Group Of Pictures， 视频序列由若干时间连续的图像构成，在对其进行编码时，先将该视频序列分割成若干个小的图像组，该参数即指一个图像组的帧数；由于GOP结构仅支持Normalp（第一帧为I帧，其余帧为P帧），此参数也为I帧间隔，默认值65535；由于编码I帧不需参考之前的帧，之前帧的编码质量影响从此帧开始不再传递，故该值越小，编码质量越好；但不建议小于帧率，因为I帧仅有帧内预测块，大部分场景需要有一定的P帧（有帧间预测块）才能保证编码质量。 取值范围：[1, 65536]。
stats_time	VBR码率统计时间，以秒为单位，一般场景下，stats_time= gop / dst_frame_rate，编码质量较好。 码率统计时间越长，每帧图像的码率波动对于码率调节的影响越弱，码率的调节会更缓慢，图像质量的波动会更轻微；码率统计时间越短，每帧图像的码率波动对于码率调节的影响越强，图像码率的调节会更灵敏，图像质量的波动会更剧烈。 取值范围：[1, 60]。
src_frame_rate	编码器输入帧率，以fps为单位，即每秒进入编码器的帧数，默认值为30；码率一定的情况下，该值越小，编码质量越好，但小于25播放时画面会不连贯。 该参数取值范围：[1, 240]。

成员名称	描述
dst_frame_rate	编码器输出帧率，以fps为单位，即每秒编码器输出的帧数，默认值为30；码率一定的情况下，该值越小，编码质量越好，但小于25播放时画面会不连贯。 取值范围：[1, src_frame_rate]。 暂不支持分数帧率。
max_bit_rate	编码器输出最大码率，以kbps为单位，即编码码流每秒比特数，该值越大，编码质量越好；默认值2000。 该参数取值范围：[2, 614400]。

11.18.20.14.10 hi_venc_h264_avbr

说明

定义 H.264 编码通道AVBR属性结构。

定义

```
typedef struct {
    hi_u32 gop;
    hi_u32 stats_time;
    hi_u32 src_frame_rate;
    hi_u32 dst_frame_rate;
    hi_u32 max_bit_rate;
}hi_venc_h264_avbr;
```

成员

成员名称	描述
gop	Group Of Pictures， 视频序列由若干时间连续的图像构成，在对其进行编码时，先将该视频序列分割成若干个小的图像组，该参数即指一个图像组的帧数；由于GOP结构仅支持Normalp（第一帧为I帧，其余帧为P帧），此参数也为I帧间隔，默认值65535；由于编码I帧不需参考之前的帧，之前帧的编码质量影响从此帧开始不再传递，故该值越小，编码质量越好；但不建议小于帧率，因为I帧仅有帧内预测块，大部分场景需要有一定的P帧（有帧间预测块块）才能保证编码质量。 取值范围：[1, 65536]。

成员名称	描述
stats_time	AVBR码率统计时间，以秒为单位，一般场景下，stats_time= gop / dst_frame_rate，编码质量较好。 码率统计时间越长，每帧图像的码率波动对于码率调节的影响越弱，码率的调节会更缓慢，图像质量的波动会更轻微；码率统计时间越短，每帧图像的码率波动对于码率调节的影响越强，图像码率的调节会更灵敏，图像质量的波动会更剧烈。 取值范围：[1, 60]。
src_frame_rate	编码器输入帧率，以fps为单位，即每秒进入编码器的帧数，默认值为30；码率一定的情况下，该值越小，编码质量越好，但小于25播放时画面会不连贯。 取值范围：[1, 240]。
dst_frame_rate	编码器输出帧率，以fps为单位，即每秒编码器输出的帧数，默认值为30；码率一定的情况下，该值越小，编码质量越好，但小于25播放时画面会不连贯。 取值范围：[1, src_frame_rate]。 暂不支持分数帧率。
max_bit_rate	编码器输出最大码率，以kbps为单位，即编码码流每秒比特数，该值越大，编码质量越好；默认值2000。 取值范围：[2, 614400]

11.18.20.14.11 hi_venc_h264_qvbr

说明

定义H.264协议编码通道QVBR模式属性结构。

定义

```
typedef struct {
    hi_u32 gop;
    hi_u32 stats_time;
    hi_u32 src_frame_rate;
    hi_u32 dst_frame_rate;
    hi_u32 target_bit_rate;
}hi_venc_h264_qvbr;
```

成员

成员名称	描述
gop	Group Of Pictures， 视频序列由若干时间连续的图像构成，在对其进行编码时，先将该视频序列分割成若干个小的图像组，该参数即指一个图像组的帧数；由于GOP结构仅支持Normalp（第一帧为I帧，其余帧为P帧），此参数也为I帧间隔，默认值65535；由于编码I帧不需参考之前的帧，之前帧的编码质量影响从此帧开始不再传递，故该值越小，编码质量越好；但不建议小于帧率，因为I帧仅有帧内预测块，大部分场景需要有一定的P帧（有帧间预测块块）才能保证编码质量。 取值范围：[1, 65536]。
stats_time	QVBR码率统计时间，以秒为单位，一般场景下，stats_time= gop / dst_frame_rate，编码质量较好。 码率统计时间越长，每帧图像的码率波动对于码率调节的影响越弱，码率的调节会更缓慢，图像质量的波动会更轻微；码率统计时间越短，每帧图像的码率波动对于码率调节的影响越强，图像码率的调节会更灵敏，图像质量的波动会更剧烈。 取值范围：[1, 60]。
src_frame_rate	编码器输入帧率，以fps为单位，即每秒进入编码器的帧数，默认值为30；码率一定的情况下，该值越小，编码质量越好，但小于25播放时画面会不连贯。 取值范围：[1, 240]。
dst_frame_rate	编码器输出帧率，以fps为单位，即每秒编码器输出的帧数，默认值为30；码率一定的情况下，该值越小，编码质量越好，但小于25播放时画面会不连贯。 取值范围：[1, src_frame_rate]。 暂不支持分数帧率。
target_bit_rate	编码器输出最大码率，以kbps为单位，即编码码流每秒比特数，该值越大，编码质量越好；默认值2000。 取值范围：[2, 614400]

11.18.20.14.12 hi_venc_h264_cvbr

说明

定义H.264协议编码通道CVBR模式属性结构。

定义

```
typedef struct {
    hi_u32 gop;
    hi_u32 stats_time;
    hi_u32 src_frame_rate;
    hi_u32 dst_frame_rate;
    hi_u32 max_bit_rate;
    hi_u32 short_term_stats_time;
    hi_u32 long_term_stats_time;
    hi_u32 long_term_max_bit_rate;
    hi_u32 long_term_min_bit_rate;
} hi_venc_h264_cvbr;
```

成员

成员名称	描述
gop	Group Of Pictures， 视频序列由若干时间连续的图像构成，在对其进行编码时，先将该视频序列分割成若干个小的图像组，该参数即指一个图像组的帧数；由于GOP结构仅支持Normalp（第一帧为I帧，其余帧为P帧），此参数也为I帧间隔，默认值65535；由于编码I帧不需参考之前的帧，之前帧的编码质量影响从此帧开始不再传递，故该值越小，编码质量越好；但不建议小于帧率，因为I帧仅有帧内预测块，大部分场景需要有一定的P帧（有帧间预测块块）才能保证编码质量。 取值范围：[1, 65536]。
stats_time	CVBR码率统计时间，以秒为单位，一般场景下，stats_time= gop / dst_frame_rate，编码质量较好。 码率统计时间越长，每帧图像的码率波动对于码率调节的影响越弱，码率的调节会更缓慢，图像质量的波动会更轻微；码率统计时间越短，每帧图像的码率波动对于码率调节的影响越强，图像码率的调节会更灵敏，图像质量的波动会更剧烈。 取值范围：[1, 60]。

成员名称	描述
src_frame_rate	编码器输入帧率，以fps为单位，即每秒进入编码器的帧数，默认值为30；码率一定的情况下，该值越小，编码质量越好，但小于25播放时画面会不连贯。 取值范围：[1, 240]。
dst_frame_rate	编码器输出帧率，以fps为单位，即每秒编码器输出的帧数，默认值为30；码率一定的情况下，该值越小，编码质量越好，但小于25播放时画面会不连贯。 取值范围：[1, src_frame_rate]。 暂不支持分数帧率。
max_bit_rate	编码器输出最大码率，以kbps为单位，即编码码流每秒比特数，该值越大，编码质量越好；默认值2000。 取值范围：[2, 614400]
short_term_stats_time	码率短期统计时间，以秒为单位。 取值范围：[1, 120]。
long_term_stats_time	码率长期统计时间，默认为分钟。 取值范围：[1, 1440]。
long_term_max_bit_rate	编码器输出长期最大码率，以kbps为单位。 取值范围：[2, max_bit_rate]。
long_term_min_bit_rate	编码器输出长期最小码率，以kbps为单位。 取值范围：[0, long_term_max_bit_rate] 。

11.18.20.14.13 hi_venc_h264_fixqp

说明

定义 H.264 编码通道FIXQP属性结构。暂不支持。

定义

```
typedef struct {
    hi_u32 gop;
    hi_u32 src_frame_rate;
    hi_u32 dst_frame_rate;
    hi_u32 i_qp;
    hi_u32 p_qp;
    hi_u32 b_qp;
} hi_venc_h264_fixqp;
```

成员

成员名称	描述
gop	Group Of Pictures， 视频序列由若干时间连续的图像构成，在对其进行编码时，先将该视频序列分割成若干个小的图像组，该参数即指一个图像组的帧数；由于GOP结构仅支持Normalp（第一帧为I帧，其余帧为P帧），此参数也为I帧间隔，默认值65535；由于编码I帧不需参考之前的帧，之前帧的编码质量影响从此帧开始不再传递，故该值越小，编码质量越好；但不建议小于帧率，因为I帧仅有帧内预测块，大部分场景需要有一定的P帧（有帧间预测块块）才能保证编码质量。 取值范围：[1, 65536]。
src_frame_rate	编码器输入帧率，以fps为单位，即每秒进入编码器的帧数，默认值为30；码率一定的情况下，该值越小，编码质量越好，但小于25播放时画面会不连贯。 该参数取值范围：[1, 240]。
dst_frame_rate	编码器输出帧率，以fps为单位，即每秒编码器输出的帧数，默认值为30；码率一定的情况下，该值越小，编码质量越好，但小于25播放时画面会不连贯。 取值范围：[1, src_frame_rate]。 暂不支持分数帧率。
i_qp	I帧所有宏块Qp值。 取值范围：[0, 51]。
p_qp	P帧所有宏块Qp值。 取值范围：[0, 51]。
b_qp	B帧所有宏块Qp值，当前仅支持配置为0。

11.18.20.14.14 hi_venc_h264_qpmap

说明

定义 H.264 编码通道QPMAP属性结构。暂不支持。

定义

```
typedef struct {
    hi_u32 gop;
    hi_u32 stats_time;
```

```
    hi_u32 src_frame_rate;
    hi_u32 dst_frame_rate;
}hi_venc_h264_qpmap;
```

11.18.20.14.15 hi_venc_mjpeg_cbr

说明

定义MJPEG编码通道CBR属性结构。暂不支持。

定义

```
typedef struct {
    hi_u32 stats_time; // RW; Range:[1, 60]; the rate statistic time, the unit is senconds(s)
    hi_u32 src_frame_rate; // RW; Range:[1, 240]; the input frame rate of the venc channel
    hi_u32 dst_frame_rate; // RW; Range:[1, 240]; the target frame rate of the venc channel,
                           // can not be larger than src_frame_rate
    hi_u32 bit_rate; // RW; Range:[2, 614400]; average bitrate
} hi_venc_mjpeg_cbr;
```

11.18.20.14.16 hi_venc_mjpeg_vbr

说明

定义MJPEG编码通道VBR属性结构。暂不支持。

定义

```
typedef struct {
    hi_u32 stats_time; // RW; Range:[1, 60]; the rate statistic time, the unit is senconds(s)
    hi_u32 src_frame_rate; // RW; Range:[1, 240]; the input frame rate of the venc channel
    hi_u32 dst_frame_rate; // RW; Range:[1, 240]; the target frame rate of the venc channel,
                           // can not be larger than src_frame_rate
    hi_u32 max_bit_rate; // RW; Range:[2, 614400];the max bitrate
}hi_venc_mjpeg_vbr;
```

11.18.20.14.17 hi_venc_mjpeg_fixqp

说明

定义MJPEG编码通道FIXQP属性结构。暂不支持。

定义

```
typedef struct {
    hi_u32 src_frame_rate; // RW; Range:[1, 240]; the input frame rate of the venc channel
    hi_u32 dst_frame_rate; // RW; Range:[1, 240]; the target frame rate of the venc channel,
                           // can not be larger than src_frame_rate
    hi_u32 qfactor; // RW; Range:[1,99];image quality.
} hi_venc_mjpeg_fixqp;
```

11.18.20.14.18 hi_venc_h265_cbr

说明

定义 H.265编码通道CBR属性结构。

定义

```
typedef struct {
    hi_u32 gop;
```

```
    hi_u32 stats_time;
    hi_u32 src_frame_rate;
    hi_u32 dst_frame_rate;
    hi_u32 bit_rate;
} hi_venc_h265_cbr;
```

成员

成员名称	描述
gop	Group Of Pictures， 视频序列由若干时间连续的图像构成，在对其进行编码时，先将该视频序列分割成若干个小的图像组，该参数即指一个图像组的帧数；由于GOP结构仅支持Normalp（第一帧为I帧，其余帧为P帧），此参数也为I帧间隔，默认值65535；由于编码I帧不需参考之前的帧，之前帧的编码质量影响从此帧开始不再传递，故该值越小，编码质量越好；但不建议小于帧率，因为I帧仅有帧内预测块，大部分场景需要有一定的P帧（有帧间预测块块）才能保证编码质量。 取值范围：[1, 65536]。
stats_time	CBR码率统计时间，以秒为单位，一般场景下，stats_time= gop / dst_frame_rate，编码质量较好。 码率统计时间越长，每帧图像的码率波动对于码率调节的影响越弱，码率的调节会更缓慢，图像质量的波动会更轻微；码率统计时间越短，每帧图像的码率波动对于码率调节的影响越强，图像码率的调节会更灵敏，图像质量的波动会更剧烈。 取值范围：[1, 60]。
src_frame_rate	编码器输入帧率，以fps为单位，即每秒进入编码器的帧数，默认值为30；码率一定的情况下，该值越小，编码质量越好，但小于25播放时画面会不连贯。 该参数取值范围：[1, 240]。
dst_frame_rate	编码器输出帧率，以fps为单位，即每秒编码器输出的帧数，默认值为30；码率一定的情况下，该值越小，编码质量越好，但小于25播放时画面会不连贯。 取值范围：[1, src_frame_rate]。 暂不支持分数帧率。

成员名称	描述
bit_rate	编码器输出平均码率，以kbps为单位，即编码码流每秒比特数，该值越大，编码质量越好；默认值2000。 该参数取值范围：[2, 614400]。 平均比特率的设置与编码通道宽高以及图像帧率都有关系。典型的平均比特率的设置如**表11-63**所示。注意，**表11-63**中的平均比特率的设置是在通道编码帧率为满帧率（30fps）时的设置。当用户设置编码输出帧率不为满帧率时，可以对下表中的码率按用户设置帧率与满帧率（30fps）的比例进行换算。

表 11-63 典型平均比特率配置

图像宽高/码率等级	1080p(1920x1080)	4K(3840x2160)
低码率	＜2Mbps	＜8Mbps
中码率	2Mbps～8Mbps	8Mbps～30Mbps
高码率	>8Mbps	>30Mbps

11.18.20.14.19 hi_venc_h265_vbr

说明

定义 H.265编码通道VBR属性结构。

定义

```
typedef struct {
    hi_u32 gop;
    hi_u32 stats_time;
    hi_u32 src_frame_rate;
    hi_u32 dst_frame_rate;
    hi_u32 max_bit_rate;
}hi_venc_h265_vbr;
```

成员

成员名称	描述
gop	Group Of Pictures， 视频序列由若干时间连续的图像构成，在对其进行编码时，先将该视频序列分割成若干个小的图像组，该参数即指一个图像组的帧数；由于GOP结构仅支持Normalp（第一帧为I帧，其余帧为P帧），此参数也为I帧间隔，默认值65535；由于编码I帧不需参考之前的帧，之前帧的编码质量影响从此帧开始不再传递，故该值越小，编码质量越好；但不建议小于帧率，因为I帧仅有帧内预测块，大部分场景需要有一定的P帧（有帧间预测块块）才能保证编码质量。 取值范围：[1, 65536]。
stats_time	VBR 码率统计时间，以秒为单位，一般场景下，stats_time= gop / dst_frame_rate，编码质量较好。 码率统计时间越长，每帧图像的码率波动对于码率调节的影响越弱，码率的调节会更缓慢，图像质量的波动会更轻微；码率统计时间越短，每帧图像的码率波动对于码率调节的影响越强，图像码率的调节会更灵敏，图像质量的波动会更剧烈。 取值范围：[1, 60]。
src_frame_rate	编码器输入帧率，以fps为单位，即每秒进入编码器的帧数，默认值为30；码率一定的情况下，该值越小，编码质量越好，但小于25播放时画面会不连贯。 该参数取值范围：[1, 240]。
dst_frame_rate	编码器输出帧率，以fps为单位，即每秒编码器输出的帧数，默认值为30；码率一定的情况下，该值越小，编码质量越好，但小于25播放时画面会不连贯。 取值范围：[1, src_frame_rate]。 暂不支持分数帧率。
max_bit_rate	编码器输出最大码率，以kbps为单位，即编码码流每秒比特数，该值越大，编码质量越好；默认值2000。 该参数取值范围：[2, 614400]。

注意事项

请参见hi_venc_h264_cbr中关于src_frame_rate和dst_frame_rate的说明。

11.18.20.14.20 hi_venc_h265_avbr

说明

定义H.265编码通道AVBR属性结构。

定义

```
typedef struct {
    hi_u32 gop;
    hi_u32 stats_time;
    hi_u32 src_frame_rate;
    hi_u32 dst_frame_rate;
    hi_u32 max_bit_rate;
}hi_venc_h265_avbr;
```

成员

成员名称	描述
gop	Group Of Pictures， 视频序列由若干时间连续的图像构成，在对其进行编码时，先将该视频序列分割成若干个小的图像组，该参数即指一个图像组的帧数；由于GOP结构仅支持Normalp（第一帧为I帧，其余帧为P帧），此参数也为I帧间隔，默认值65535；由于编码I帧不需参考之前的帧，之前帧的编码质量影响从此帧开始不再传递，故该值越小，编码质量越好；但不建议小于帧率，因为I帧仅有帧内预测块，大部分场景需要有一定的P帧（有帧间预测块块）才能保证编码质量。 取值范围：[1, 65536]。
stats_time	AVBR码率统计时间，以秒为单位，一般场景下，stats_time= gop / dst_frame_rate，编码质量较好。 码率统计时间越长，每帧图像的码率波动对于码率调节的影响越弱，码率的调节会更缓慢，图像质量的波动会更轻微；码率统计时间越短，每帧图像的码率波动对于码率调节的影响越强，图像码率的调节会更灵敏，图像质量的波动会更剧烈。 取值范围：[1, 60]。

成员名称	描述
src_frame_rate	编码器输入帧率，以fps为单位，即每秒进入编码器的帧数，默认值为30；码率一定的情况下，该值越小，编码质量越好，但小于25播放时画面会不连贯。 该参数取值范围：[1, 240]。
dst_frame_rate	编码器输出帧率，以fps为单位，即每秒编码器输出的帧数，默认值为30；码率一定的情况下，该值越小，编码质量越好，但小于25播放时画面会不连贯。 取值范围：[1, src_frame_rate]。 暂不支持分数帧率。
max_bit_rate	编码器输出最大码率，以kbps为单位，即编码码流每秒比特数，该值越大，编码质量越好；默认值2000。 该参数取值范围：[2, 614400]。

11.18.20.14.21 hi_venc_h265_qvbr

说明

定义H.265协议编码通道QVBR模式属性结构。

定义

```
typedef struct {
    hi_u32 gop;
    hi_u32 stats_time;
    hi_u32 src_frame_rate;
    hi_u32 dst_frame_rate;
    hi_u32 target_bit_rate;
}hi_venc_h265_qvbr;
```

成员

成员名称	描述
gop	Group Of Pictures， 视频序列由若干时间连续的图像构成，在对其进行编码时，先将该视频序列分割成若干个小的图像组，该参数即指一个图像组的帧数；由于GOP结构仅支持Normalp（第一帧为I帧，其余帧为P帧），此参数也为I帧间隔，默认值65535；由于编码I帧不需参考之前的帧，之前帧的编码质量影响从此帧开始不再传递，故该值越小，编码质量越好；但不建议小于帧率，因为I帧仅有帧内预测块，大部分场景需要有一定的P帧（有帧间预测块块）才能保证编码质量。 取值范围：[1, 65536]。
stats_time	QVBR码率统计时间，以秒为单位，一般场景下，stats_time= gop / dst_frame_rate，编码质量较好。 码率统计时间越长，每帧图像的码率波动对于码率调节的影响越弱，码率的调节会更缓慢，图像质量的波动会更轻微；码率统计时间越短，每帧图像的码率波动对于码率调节的影响越强，图像码率的调节会更灵敏，图像质量的波动会更剧烈。 取值范围：[1, 60]。
src_frame_rate	编码器输入帧率，以fps为单位，即每秒进入编码器的帧数，默认值为30；码率一定的情况下，该值越小，编码质量越好，但小于25播放时画面会不连贯。 取值范围：[1, 240]。
dst_frame_rate	编码器输出帧率，以fps为单位，即每秒编码器输出的帧数，默认值为30；码率一定的情况下，该值越小，编码质量越好，但小于25播放时画面会不连贯。 取值范围：[1, src_frame_rate]。 暂不支持分数帧率。
target_bit_rate	编码器输出最大码率，以kbps为单位，即编码码流每秒比特数，该值越大，编码质量越好；默认值2000。 取值范围：[2, 614400]。

11.18.20.14.22 hi_venc_h265_cvbr

说明

定义H.265协议编码通道CVBR模式属性结构。

定义

```
typedef struct {
    hi_u32 gop;
    hi_u32 stats_time;
    hi_u32 src_frame_rate;
    hi_u32 dst_frame_rate;
    hi_u32 max_bit_rate;
    hi_u32 short_term_stats_time;
    hi_u32 long_term_stats_time;
    hi_u32 long_term_max_bit_rate;
    hi_u32 long_term_min_bit_rate;
} hi_venc_h265_cvbr;
```

成员

成员名称	描述
gop	Group Of Pictures， 视频序列由若干时间连续的图像构成，在对其进行编码时，先将该视频序列分割成若干个小的图像组，该参数即指一个图像组的帧数；由于GOP结构仅支持Normalp（第一帧为I帧，其余帧为P帧），此参数也为I帧间隔，默认值65535；由于编码I帧不需参考之前的帧，之前帧的编码质量影响从此帧开始不再传递，故该值越小，编码质量越好；但不建议小于帧率，因为I帧仅有帧内预测块，大部分场景需要有一定的P帧（有帧间预测块块）才能保证编码质量。 取值范围：[1, 65536]。
stats_time	CVBR码率统计时间，以秒为单位，一般场景下，stats_time= gop / dst_frame_rate，编码质量较好。 码率统计时间越长，每帧图像的码率波动对于码率调节的影响越弱，码率的调节会更缓慢，图像质量的波动会更轻微；码率统计时间越短，每帧图像的码率波动对于码率调节的影响越强，图像码率的调节会更灵敏，图像质量的波动会更剧烈。 取值范围：[1, 60]。

成员名称	描述
src_frame_rate	编码器输入帧率，以fps为单位，即每秒进入编码器的帧数，默认值为30；码率一定的情况下，该值越小，编码质量越好，但小于25播放时画面会不连贯。 取值范围：[1, 240]。
dst_frame_rate	编码器输出帧率，以fps为单位，即每秒编码器输出的帧数，默认值为30；码率一定的情况下，该值越小，编码质量越好，但小于25播放时画面会不连贯。 取值范围：[1, src_frame_rate]。 暂不支持分数帧率。
max_bit_rate	编码器输出最大码率，以kbps为单位，即编码码流每秒比特数，该值越大，编码质量越好；默认值2000。 取值范围：[2, 614400]
short_term_stats_time	码率短期统计时间，以秒为单位。 取值范围：[1, 120]。
long_term_stats_time	码率长期统计时间，默认为分钟。 取值范围：[1, 1440]。
long_term_max_bit_rate	编码器输出长期最大码率，以kbps为单位。 取值范围：[2, max_bit_rate]。
long_term_min_bit_rate	编码器输出长期最小码率，以kbps为单位。 取值范围：[0, long_term_max_bit_rate] 。

11.18.20.14.23 hi_venc_h265_fixqp

说明

定义H.265编码通道FIXQP属性结构。暂不支持。

定义

```
typedef struct {
    hi_u32 gop;
    hi_u32 src_frame_rate;
    hi_u32 dst_frame_rate;
    hi_u32 i_qp;
    hi_u32 p_qp;
    hi_u32 b_qp;
} hi_venc_h265_fixqp;
```

成员

成员名称	描述
gop	Group Of Pictures， 视频序列由若干时间连续的图像构成，在对其进行编码时，先将该视频序列分割成若干个小的图像组，该参数即指一个图像组的帧数；由于GOP结构仅支持Normalp（第一帧为I帧，其余帧为P帧），此参数也为I帧间隔，默认值65535；由于编码I帧不需参考之前的帧，之前帧的编码质量影响从此帧开始不再传递，故该值越小，编码质量越好；但不建议小于帧率，因为I帧仅有帧内预测块，大部分场景需要有一定的P帧（有帧间预测块块）才能保证编码质量。 取值范围：[1, 65536]。
src_frame_rate	编码器输入帧率，以fps为单位，即每秒进入编码器的帧数，默认值为30；码率一定的情况下，该值越小，编码质量越好，但小于25播放时画面会不连贯。 该参数取值范围：[1, 240]。
dst_frame_rate	编码器输出帧率，以fps为单位，即每秒编码器输出的帧数，默认值为30；码率一定的情况下，该值越小，编码质量越好，但小于25播放时画面会不连贯。 取值范围：[1, src_frame_rate]。 暂不支持分数帧率。
i_qp	I帧所有宏块Qp值。 取值范围：[0, 51]。
p_qp	P帧所有宏块Qp值。 取值范围：[0, 51]。
b_qp	B帧所有宏块Qp值，当前仅支持配置为0。

11.18.20.14.24 hi_venc_h265_qpmap

说明

定义H.265编码通道QPMAP属性结构。暂不支持。

定义

```
typedef struct {
    hi_u32 gop;
    hi_u32 stats_time;
```

```
    hi_u32 src_frame_rate;
    hi_u32 dst_frame_rate;
    hi_venc_rc_qpmap_mode qpmap_mode;
}hi_venc_h265_qpmap;
```

11.18.20.14.25 hi_venc_gop_attr

说明

定义编码器GOP属性结构体。

定义

```
typedef struct {
    hi_venc_gop_mode gop_mode;
    union {
        hi_venc_gop_normal_p normal_p;
        hi_venc_gop_dual_p dual_p;
        hi_venc_gop_smart_p smart_p;
        hi_venc_gop_adv_smart_p adv_smart_p;
        hi_venc_gop_bipred_b bipred_b;
    };
} hi_venc_gop_attr;
```

成员

成员名称	描述
gop_mode	编码GOP类型。
normal_p	编码单参考帧P帧GOP属性结构体。
dual_p	编码双参考帧P帧GOP属性结构体。 Atlas 推理系列产品，预留参数，暂不支持。
smart_p	编码智能P帧GOP属性结构体。 Atlas 推理系列产品，预留参数，暂不支持。
adv_smart_p	编码高级智能P帧GOP属性结构体，暂不支持。
bipred_b	编码B帧GOP属性结构体，暂不支持。

11.18.20.14.26 hi_venc_gop_mode

说明

定义 H.264/H.265 GOP类型。

定义

```
typedef enum {
    HI_VENC_GOP_MODE_NORMAL_P     = 0,
    HI_VENC_GOP_MODE_DUAL_P       = 1,
    HI_VENC_GOP_MODE_SMART_P      = 2,
```

```
    HI_VENC_GOP_MODE_ADV_SMART_P  = 3,
    HI_VENC_GOP_MODE_BIPRED_B     = 4,
    HI_VENC_GOP_MODE_LOW_DELAY_B  = 5,

    HI_VENC_GOP_MODE_BUTT,
} hi_venc_gop_mode;
```

成员

成员名称	描述
HI_VENC_GOP_MODE_NORMAL_P	编码单参考帧P帧GOP类型。
HI_VENC_GOP_MODE_DUAL_P	编码双参考帧P帧GOP类型。 Atlas 推理系列产品，预留参数，暂不支持。
HI_VENC_GOP_MODE_SMART_P	编码智能P帧GOP类型。 Atlas 推理系列产品，预留参数，暂不支持。
HI_VENC_GOP_MODE_ADV_SMART_P	编码高级智能P帧GOP类型，暂不支持。
HI_VENC_GOP_MODE_BIPRED_B	编码B帧GOP类型，暂不支持。
HI_VENC_GOP_MODE_LOW_DELAY_B	编码B帧GOP类型，其中B帧只有前向参考帧，暂不支持。
HI_VENC_GOP_MODE_BUTT	保留值

11.18.20.14.27 hi_venc_gop_normal_p

说明

定义编码单参考帧P帧GOP属性结构体。

定义

```
typedef struct {
    hi_s32 ip_qp_delta;
}hi_venc_gop_normal_p;
```

成员

成员名称	描述
ip_qp_delta	I帧相对P帧的QP（Quantisation Parameter）差值。 取值范围：[-10, 30]。

11.18.20.14.28 hi_venc_gop_dual_p

说明

定义编码双参考帧P帧GOP属性结构体。

Atlas 推理系列产品，预留结构体，暂不支持。

定义

```
typedef struct {
    hi_u32 sp_interval;
    hi_s32 sp_qp_delta;
    hi_s32 ip_qp_delta;
}hi_venc_gop_dual_p;
```

成员

成员名称	描述
sp_interval	Special P帧的间隔。 取值范围：[0, 1) U (1, gop-1]，gop是I帧间隔。
sp_qp_delta	Special P帧相对普通P帧的QP差值。 取值范围：[-10, 30]。
ip_qp_delta	I帧相对普通P帧的QP差值。 取值范围：[-10, 30]。

11.18.20.14.29 hi_venc_gop_smart_p

说明

定义编码智能P帧GOP属性结构体。

Atlas 推理系列产品，预留结构体，暂不支持。

定义

```
typedef struct {
    hi_u32 bg_interval;
    hi_s32 bg_qp_delta;
    hi_s32 vi_qp_delta;
}hi_venc_gop_smart_p;
```

成员

成员名称	描述
bg_interval	长期参考帧的间隔。 取值范围：[gop, 65536]，且必须是gop的整数倍，gop是I帧间隔。

成员名称	描述
bg_qp_delta	长期参考帧相对P帧的QP差值。 取值范围：[-10, 30]。 在调试信息中，该参数等同其它码控模式的ip_qp_delta。
vi_qp_delta	虚拟I帧相对普通P帧的QP差值。 取值范围：[-10, 30]。

11.18.20.14.30 hi_venc_gop_adv_smart_p

说明

定义编码高级智能P帧GOP属性结构体，暂不支持。

定义

```
typedef struct {
    hi_u32 bg_interval;
    hi_s32 bg_qp_delta;
    hi_s32 vi_qp_delta;
}hi_venc_gop_adv_smart_p;
```

11.18.20.14.31 hi_venc_gop_bipred_b

说明

定义编码B帧GOP属性结构体，暂不支持。

定义

```
typedef struct {
    hi_u32 b_frame_num; // RW; Range:[1, 3]; Number of B frames
    hi_s32 b_qp_delta; // RW; Range:[-10, 30]; QP variance between P frame and B frame
    hi_s32 ip_qp_delta; // RW; Range:[-10, 30]; QP variance between P frame and I frame
} hi_venc_gop_bipred_b;
```

11.18.20.14.32 hi_venc_start_param

说明

定义编码通道连续接收并编码的帧数结构体。

定义

```
typedef struct {
    hi_s32 recv_pic_num;
} hi_venc_start_param;
```

成员

成员名称	描述
recv_pic_num	编码通道连续接收并编码的帧数。 取值范围：-1或正整数。

11.18.20.14.33 hi_venc_chn_status

说明

定义编码通道的状态结构体。

定义

```
typedef struct {
    hi_u32 left_pics;
    hi_u32 left_stream_bytes;
    hi_u32 left_stream_frames;
    hi_u32 cur_packs;
    hi_u32 left_recv_pics;
    hi_u32 left_enc_pics;
    hi_bool is_jpeg_snap_end;
    hi_u64 release_pic_pts;
    hi_venc_stream_info stream_info;
} hi_venc_chn_status;
```

成员

成员名称	描述
left_pics	待编码的图像数，暂不支持。
left_stream_bytes	码流buffer中用户暂未取走的byte数目，暂不支持。
left_stream_frames	码流buffer中用户暂未取走的帧数目，暂不支持。
cur_packs	当前帧的码流包个数。
left_recv_pics	剩余待接收的帧数，在用户调用接口hi_mpi_venc_start_chn设置接收帧数后有效，暂不支持。
left_enc_pics	剩余待编码的帧数，在用户调用接口hi_mpi_venc_start_chn设置接收帧数后有效，暂不支持。
is_jpeg_snap_end	Jpege抓拍模式下指示抓拍过程是否结束，暂不支持。
release_pic_pts	预留参数，暂不支持。
stream_info	码流信息。 Atlas 推理系列产品，预留参数，暂不支持。

注意事项

如果venc 接收的抓拍帧有丢帧出现，is_jpeg_snap_end会存在不能正确标记抓拍结束的情况。

11.18.20.14.34 hi_venc_h264_stream_info

说明

定义H.264协议码流特征信息。

Atlas 推理系列产品，预留结构体，暂不支持。

定义

```
typedef struct {
    hi_u32 pic_bytes;
    hi_u32 inter16x16_mb_num;
    hi_u32 inter8x8_mb_num;
    hi_u32 intra16_mb_num;
    hi_u32 intra8_mb_num;
    hi_u32 intra4_mb_num;
    hi_venc_ref_type ref_type;
    hi_u32 update_attr_cnt;
    hi_u32 start_qp;
    hi_u32 mean_qp;
    hi_bool is_p_skip;
} hi_venc_h264_stream_info;
```

成员

成员名称	描述
pic_bytes	编码当前帧的字节数。
inter16x16_mb_num	编码当前帧中采用Inter16*16预测模式的宏块数。
inter8x8_mb_num	编码当前帧中采用Inter8*8预测模式的宏块数。
intra16_mb_num	编码当前帧中采用Intra16预测模式的宏块数。
intra8_mb_num	编码当前帧中采用Intra8预测模式的宏块数。
intra4_mb_num	编码当前帧中采用Intra4预测模式的宏块数。
ref_type	高级跳帧参考下的编码帧类型。
update_attr_cnt	通道属性或参数(包含RC参数)被设置的次数。
start_qp	编码当前帧的startqp值。
mean_qp	编码当前帧的平均QP帧。
is_p_skip	标识当前帧是否为pskip帧。

11.18.20.14.35 hi_venc_ref_type

说明

定义H.264跳帧参考码流的帧类型和参考属性，暂不支持。

定义

```
typedef enum {
    HI_VENC_BASE_IDR_SLICE = 0,
    HI_VENC_BASE_P_SLICE_REF_TO_IDR,
    HI_VENC_BASE_P_SLICE_REF_BY_BASE,
    HI_VENC_BASE_P_SLICE_REF_BY_ENHANCE,
    HI_VENC_ENHANCE_P_SLICE_REF_BY_ENHANCE,
    HI_VENC_ENHANCE_P_SLICE_NOT_FOR_REF,
    HI_VENC_P_SLICE_BUTT
} hi_venc_ref_type;
```

11.18.20.14.36 hi_venc_jpeg_stream_info

说明

定义JPEG协议码流特征信息， 暂不支持。

定义

```
typedef struct {
    hi_u32 pic_bytes;
    hi_u32 update_attr_cnt;
    hi_u32 qfactor;
}hi_venc_jpeg_stream_info;
```

成员

成员名称	描述
pic_bytes	一帧JPEG码流大小，以字节为单位。
update_attr_cnt	通道属性或参数(包含RC参数)被设置的次数。
qfactor	编码当前帧的Qfactor。

11.18.20.14.37 hi_venc_h265_stream_info

说明

定义H.265协议码流特征信息。

Atlas 推理系列产品，预留结构体，暂不支持。

定义

```
typedef struct {
    hi_u32 pic_bytes;
    hi_u32 inter64x64_cu_num;
    hi_u32 inter32x32_cu_num;
    hi_u32 inter16x16_cu_num;
```

```
    hi_u32 inter8x8_cu_num;
    hi_u32 intra32x32_cu_num;
    hi_u32 intra16x16_cu_num;
    hi_u32 intra8x8_cu_num;
    hi_u32 intra4x4_cu_num;
    hi_venc_ref_type ref_type;
    hi_u32 update_attr_cnt;
    hi_u32 start_qp;
    hi_u32 mean_qp;
    hi_bool is_p_skip;
}hi_venc_h265_stream_info;
```

成员

成员名称	描述
pic_bytes	编码当前帧的字节数。
inter64x64_cu_num	编码当前帧中采用Inter64*64预测模式的CU块数。
inter32x32_cu_num	编码当前帧中采用Inter32*32预测模式的CU块数。
inter16x16_cu_num	编码当前帧中采用Inter16*16预测模式的CU块数。
inter8x8_cu_num	编码当前帧中采用Inter8*8预测模式的CU块数。
intra32x32_cu_num	编码当前帧中采用Intra32*32预测模式的CU块数。
intra16x16_cu_num	编码当前帧中采用Intra16*16预测模式的CU块数。
intra8x8_cu_num	编码当前帧中采用Intra8*8预测模式的CU块数。
intra4x4_cu_num	编码当前帧中采用Intra4*4预测模式的CU块数。
ref_type	高级跳帧参考下的编码帧类型。
update_attr_cnt	通道属性或参数(包含RC参数)被设置的次数。
start_qp	编码当前帧的startqp值。
mean_qp	编码当前帧的平均QP值。
is_p_skip	标识当前帧是否为pskip帧。

注意事项

ref_type请参见hi_venc_h264_stream_info中关于ref_type变量的说明。

11.18.20.14.38 hi_venc_prores_stream_info

说明

定义PRORES协议码流特征信息， 暂不支持。

定义

```
typedef struct {
    hi_u32 pic_bytes;
    hi_u32 update_attr_cnt;
} hi_venc_prores_stream_info;
```

成员

成员名称	描述
pic_bytes	编码当前帧的字节数。
update_attr_cnt	通道属性或参数被设置的次数。

11.18.20.14.39 hi_venc_mod_param

说明

编码相关模块参数。

定义

```
typedef struct {
    hi_venc_mod_type mod_type;
    union {
        hi_venc_venc_mod_param venc_mod_param;
        hi_venc_h264_mod_param h264_mod_param;
        hi_venc_h265_mod_param h265_mod_param;
        hi_venc_jpeg_mod_param jpeg_mod_param;
    };
} hi_venc_mod_param;
```

成员

成员名称	描述
mod_type	设置或者获取模块参数的类型。
venc_mod_param/ h264_mod_param/ h265_mod_param/ jpeg_mod_param	VENC内部各个模块的工作参数。

11.18.20.14.40 hi_venc_mod_type

说明

编码相关模块参数类型。

定义

```
typedef enum {
    HI_VENC_MOD_VENC = 1,
```

```
    HI_VENC_MOD_H264,
    HI_VENC_MOD_H265,
    HI_VENC_MOD_JPEG,
    HI_VENC_MOD_RC,
    HI_VENC_MOD_BUTT
} hi_venc_mod_type;
```

成员

成员名称	描述
HI_VENC_MOD_VENC	VENC内公共模块参数类型。
HI_VENC_MOD_H264	VENC内h264e模块参数类型。
HI_VENC_MOD_H265	VENC内h265e模块参数类型。
HI_VENC_MOD_JPEG	VENC内jpege模块参数类型。
HI_VENC_MOD_RC	VENC内码控模块参数类型。暂不支持设置。
HI_VENC_MOD_BUTT	保留值。

11.18.20.14.41 hi_venc_venc_mod_param

说明

Device上的venc内公共模块参数。

定义

```
typedef struct {
    hi_u32 buf_cache;
    hi_u32 frame_buf_recycle;
} hi_venc_venc_mod_param;
```

成员

成员名称	描述
buf_cache	码流获取是否支持cache方式，暂不支持，默认值为0。 ● 0：关闭码流Buffer Cache。 ● 1：打开码流Buffer Cache。
frame_buf_recycle	帧存是否回收，暂不支持，默认值为0。 ● 0：关闭编码帧存回收。 ● 1：打开编码帧存回收。

11.18.20.14.42 hi_venc_h264_mod_param

说明

Device上的venc内h264e模块参数。

定义

```
typedef struct {
    hi_u32          one_stream_buf;
    hi_u32          mini_buf_mode;
    hi_u32          low_power_mode;
    hi_vb_src       vb_src;
    hi_bool         qp_hist_en;
    hi_u32          max_user_data_len;
} hi_venc_h264_mod_param;
```

成员

成员名称	描述
one_stream_buf	编码码流帧配置模式，默认值为1。 ● 0：多包模式。 ● 1：单包模式。
mini_buf_mode	编码码流buffer配置模式，暂不支持, 默认值为1。 0：码流buffer根据分辨率分配。 1：码流buffer下限为32k，用户保证合理
low_power_mode	低功耗模式，暂不支持，默认值为0。 ● 0：关闭低功耗模式。 ● 1：使能低功耗模式。 注意：低功耗或者极低功耗使能后会导致图像质量损失。
vb_src	参考帧和重构帧的帧存分配方式，默认值为HI_VB_SRC_PRIVATE。 ● HI_VB_SRC_PRIVATE：私有VB方式，驱动内部完成帧存分配。 ● HI_VB_SRC_USER：用户VB，需由用户分配帧存，暂不支持
qp_hist_en	Qp 直方图输出控制模式，暂不支持，默认值为1。 ● 0：Qp 直方图不输出，即hi_venc_h264_adv_stream_info中的成员变量qp_hist为全0。 ● 1：Qp 直方图输出，即hi_venc_h264_adv_stream_info中的成员变量qp_hist为实际的当前帧的QP直方图 。
max_user_data_len	用户数据所占内存的最大大小，单位Byte，暂不支持，默认值为1024。

11.18.20.14.43 hi_venc_h265_mod_param

说明

Device上的venc内h265e模块参数。

定义

```
typedef struct {
    hi_u32 one_stream_buf;
    hi_u32 mini_buf_mode;
    hi_u32 low_power_mode;
    hi_vb_src vb_src;
    hi_bool qp_hist_en;
    hi_u32 max_user_data_len;
} hi_venc_h265_mod_param;
```

成员

成员名称	描述
one_stream_buf	编码码流帧配置模式，默认值为1。 ● 0：多包模式。 ● 1：单包模式。
mini_buf_mode	编码码流buffer配置模式，暂不支持，默认值为1。 ● 0：码流buffer 根据分辨率分配。 ● 1：码流buffer下限为32k，用户保证合理。
low_power_mode	低功耗模式，暂不支持，默认值为0。 ● 0：关闭低功耗模式。 ● 1：使能低功耗模式。 ● 2：使能极低功耗模式。 注意：低功耗或者极低功耗使能后会导致图像质量损失。
vb_src	参考帧和重构帧的帧存分配方式，默认值为HI_VB_SRC_PRIVATE。 ● HI_VB_SRC_PRIVATE：私有VB方式，驱动内部完成帧存分配。 ● HI_VB_SRC_USER：用户VB，需由用户分配帧存，暂不支持。
qp_hist_en	Qp 直方图输出控制模式，暂不支持，默认值为1。 ● 0：Qp 直方图不输出，即hi_venc_h265_adv_stream_info中的成员变量qp_hist为全0。 ● 1：Qp 直方图输出，即hi_venc_h265_adv_stream_info中的成员变量qp_hist为实际的当前帧的QP直方图 。

成员名称	描述
max_user_data_len	用户数据所占内存的最大大小，单位Byte，暂不支持，默认值为1024。

11.18.20.14.44 hi_venc_jpeg_mod_param

说明

Device上的venc内jpege模块中参数。

定义

```
typedef struct {
    hi_u32 one_stream_buf;
    hi_u32 mini_buf_mode;
    hi_u32 clear_stream_buf;
    hi_u32 dering_mode;
} hi_venc_jpeg_mod_param;
```

成员

成员名称	描述
one_stream_buf	编码码流帧配置模式，默认值为1。 ● 0：多包模式，暂不支持 ● 1：单包模式
mini_buf_mode	编码码流buffer配置模式，暂不支持，默认值为0。 ● 0：码流buffer根据分辨率分配。 ● 1：码流buffer下限为32k，用户保证合理。
clear_stream_buf	JPEG 编码通道设置属性时是否清空码流buffer，暂不支持，默认值为1。 ● 0：设置通道属性时保留码流buffer 和上下文计数。 ● 1：设置通道属性时清空码流buffer。
dering_mode	预留参数，暂不使用，默认值为0。

11.18.20.14.45 hi_venc_jpeg_param

说明

用于设置或者获取指定JPEGE编码通道的量化参数的数据结构。

定义

```
typedef struct {
    hi_u32 qfactor;
```

```
    hi_u8 y_qt[64];
    hi_u8 cb_qt[64];
    hi_u8 cr_qt[64];
    hi_u32 mcu_per_ecs;
    hi_bool ecs_output_en;
} hi_venc_jpeg_param;
```

成员

成员名称	描述
qfactor	编码质量范围0xFFFFFFFF或[1, 100]，数值越小图片质量越差，默认100。 ● 0xFFFFFFFF：配置成该值，则用户需要自行配置y_qt、cb_qt、cr_qt，量化表的配置会影响最后的编码结果。 ● [1, 100]：配置成该范围内的值，表示y_qt、cb_qt、cr_qt参数不生效，以系统默认的为准。
y_qt	Y量化表。 取值范围：[1, 255]。
cb_qt	Cb量化表。 取值范围：[1, 255]。
cr_qt	Cr量化表。 取值范围：[1, 255]。
mcu_per_ecs	每个ECS中包含多少个MCU，系统默认为 0，表示不划分Ecs，预留参数，当前不支持。 mcu_per_ecs：[0, (picwidth +15)>>4*(picheight+15)>>4*2]
ecs_output_en	预留参数，暂不支持。

11.18.20.14.46 hi_venc_h264_adv_stream_info

说明

定义H.264协议码流高级特征信息。

Atlas 推理系列产品，预留结构体，暂不支持。

定义

```
typedef struct {
    hi_u32 residual_bits;
    hi_u32 head_bits;
    hi_u32 madi_val;
    hi_u32 madp_val;
    hi_double psnr_val;
    hi_u32 sse_lcu_cnt;
    hi_u64 sse_sum;
    hi_venc_sse_info sse_info[HI_VENC_MAX_SSE_NUM];
```

```
    hi_u32 qp_hist[HI_VENC_QP_HIST_NUM];
    hi_u32 move_scene16x16_num;
    hi_u32 move_scene_bits;
} hi_venc_h264_adv_stream_info;
```

成员

成员名称	描述
residual_bits	编码当前帧残差（bit）数 。
head_bits	编码当前帧头信息的（bit）数 。
madi_val	编码当前帧空域纹理复杂度Madi值 。
madp_val	编码当前帧时域运动复杂度Madp值。
psnr_val	编码当前帧的PSNR（Peak signal-to-noise ratio）（峰值信噪比）值 。 不同软件中关于峰值信噪比的计算公式略有不同，因此峰值信噪比值可能存在差异。 仅支持输入图片分辨率的乘积小于4096*2032时获取psnr_val值，否则psnr_val值无效。
sse_lcu_cnt	编码当前帧中LCU（Largist Coding Unit）个数 。
sse_sum	编码当前帧中SSE（sum of squared errors ）（和方差）值。 不同软件中关于和方差的计算公式略有不同，因此和方差值可能存在差异。
sse_info	编码当前帧中8个区域的SSE（和方差）值。 不同软件中关于和方差的计算公式略有不同，因此和方差值可能存在差异。 `#define HI_VENC_MAX_SSE_NUM 8`
qp_hist	编码当前帧Qp直方图。 `#define HI_VENC_QP_HIST_NUM 52`
move_scene16x16_num	判断为图像前景的16*16块的数目，需要开启前景宏块级码控制。
move_scene_bits	判断为图像前景区域编码bit数，需要开启前景宏块级码控制。

11.18.20.14.47 hi_venc_h265_adv_stream_info

说明

定义H.265协议码流高级特征信息。

Atlas 推理系列产品，预留结构体，暂不支持。

定义

```
typedef struct {
    hi_u32 residual_bits;
    hi_u32 head_bits;
    hi_u32 madi_val;
    hi_u32 madp_val;
    hi_double psnr_val;
    hi_u32 sse_lcu_cnt;
    hi_u64 sse_sum;
    hi_venc_sse_info sse_info[HI_VENC_MAX_SSE_NUM];
    hi_u32 qp_hist[HI_VENC_QP_HIST_NUM];
    hi_u32 move_scene32x32_num;
    hi_u32 move_scene_bits;
} hi_venc_h265_adv_stream_info;
```

成员

成员名称	描述
residual_bits	编码当前帧残差（bit）数 。
head_bits	编码当前帧头信息的（bit）数 。
madi_val	编码当前帧空域纹理复杂度Madi值 。
madp_val	编码当前帧时域运动复杂度Madp值。
psnr_val	编码当前帧的PSNR（峰值信噪比）值 。 不同软件中关于峰值信噪比的计算公式略有不同，因此峰值信噪比值可能存在差异。 仅支持输入图片分辨率的乘积小于8192*4064时获取psnr_val值，否则psnr_val值无效。
sse_lcu_cnt	编码当前帧中LCU个数 。
sse_sum	编码当前帧中SSE（和方差）值。 不同软件中关于和方差的计算公式略有不同，因此和方差值可能存在差异。
sse_info	编码当前帧中8个区域的SSE（和方差）值。 不同软件中关于和方差的计算公式略有不同，因此和方差值可能存在差异。 `#define HI_VENC_MAX_SSE_NUM 8`
qp_hist	编码当前帧Qp直方图。 `#define HI_VENC_QP_HIST_NUM 52`
move_scene32x32_num	判断为图像前景的32*32块的数目，需要开启前景宏块级码控制。
move_scene_bits	判断为图像前景区域编码bit数，需要开启前景宏块级码控制。

11.18.20.14.48 hi_venc_data_type

说明

定义码流结果类型。

定义

```
typedef union {
    hi_venc_h264_nalu_type h264_type;
    hi_venc_jpege_pack_type jpeg_type;
    hi_venc_h265_nalu_type h265_type;
    hi_venc_prores_pack_type prores_type;
} hi_venc_data_type;
```

成员

成员名称	描述
h264_type	H.264码流包类型。
jpeg_type	JPEG码流包类型。
h265_type	H.265码流包类型。
prores_type	PRORES码流包类型。

11.18.20.14.49 hi_venc_h264_nalu_type

说明

定义H.264码流NALU类型。

定义

```
typedef enum {
    HI_VENC_H264_NALU_B_SLICE = 0, // B SLICE types
    HI_VENC_H264_NALU_P_SLICE = 1, // P SLICE types
    HI_VENC_H264_NALU_I_SLICE = 2, // I SLICE types
    HI_VENC_H264_NALU_IDR_SLICE = 5, // IDR SLICE types
    HI_VENC_H264_NALU_SEI     = 6, // SEI types
    HI_VENC_H264_NALU_SPS     = 7, // SPS types
    HI_VENC_H264_NALU_PPS     = 8, // PPS types
    HI_VENC_H264_NALU_BUTT
} hi_venc_h264_nalu_type;
```

11.18.20.14.50 hi_venc_jpege_pack_type

说明

定义JPEG码流的PACK类型。

定义

```
typedef enum {
    HI_VENC_JPEG_PACK_ECS = 5, // ECS types
    HI_VENC_JPEG_PACK_APP = 6, // APP types
```

```
    HI_VENC_JPEG_PACK_VDO = 7, // VDO types
    HI_VENC_JPEG_PACK_PIC = 8, // PIC types
    HI_VENC_JPEG_PACK_DCF = 9, // DCF types
    HI_VENC_JPEG_PACK_DCF_PIC = 10, // DCF PIC types
    HI_VENC_JPEG_PACK_BUTT
} hi_venc_jpege_pack_type;
```

11.18.20.14.51 hi_venc_h265_nalu_type

说明

定义H.265码流NALU类型。

定义

```
typedef enum {
    HI_VENC_H265_NALU_B_SLICE = 0, // B SLICE types
    HI_VENC_H265_NALU_P_SLICE = 1, // P SLICE types
    HI_VENC_H265_NALU_I_SLICE = 2, // I SLICE types
    HI_VENC_H265_NALU_IDR_SLICE = 19, // IDR SLICE types
    HI_VENC_H265_NALU_VPS     = 32, // VPS types
    HI_VENC_H265_NALU_SPS     = 33, // SPS types
    HI_VENC_H265_NALU_PPS     = 34, // PPS types
    HI_VENC_H265_NALU_SEI     = 39, // SEI types
    HI_VENC_H265_NALU_ENHANCE   = 64, // ENHANCE typesHI_VENC_H265_NALU_BUTT
} hi_venc_h265_nalu_type;
```

11.18.20.14.52 hi_venc_prores_pack_type

说明

定义PRORES码流的PACK类型，暂不支持。

定义

```
typedef enum {
    HI_VENC_PRORES_PACK_PIC = 1, // PIC types
    HI_VENC_PRORES_PACK_BUTT
} hi_venc_prores_pack_type;
```

11.18.20.14.53 hi_venc_chn_param

说明

定义Venc通道参数结构体。

定义

```
typedef struct {
    hi_bool color_to_grey_en;
    hi_u32 priority;
    hi_u32 max_stream_cnt;
    hi_u32 poll_wake_up_frame_cnt;
    hi_crop_info crop_info;
    hi_frame_rate_ctrl frame_rate;
} hi_venc_chn_param;
```

成员

成员名称	描述
color_to_grey_en	开启或关闭一个通道的彩转灰功能。 Atlas 推理系列产品，预留参数，暂不支持。
priority	编码通道优先级参数，默认为0，数字越大优先级越高。 取值范围：[0, 2)。
max_stream_cnt	最大码流缓存帧数，暂不支持。
poll_wake_up_frame_cnt	当通道使用超时或阻塞获取码流，编码指定的帧poll_wake_up_frame_cnt之后唤醒阻塞接口，暂不支持。 取值范围：大于0 默认值：1
crop_info	预留参数，暂不支持。
frame_rate	通道帧率控制参数，暂不支持。

11.18.20.14.54 hi_venc_stream_info

说明

定义码流信息结构体。

Atlas 推理系列产品，预留结构体，暂不支持。

定义

```
typedef struct {
    hi_venc_ref_type ref_type;
    hi_u32 pic_bytes;
    hi_u32 pic_cnt;
    hi_u32 start_qp;
    hi_u32 mean_qp;
    hi_bool is_p_skip;

    hi_u32 residual_bits;
    hi_u32 head_bits;
    hi_u32 madi_val;
    hi_u32 madp_val;
    hi_u64 sse_sum;
    hi_u32 sse_lcu_cnt;
    hi_double psnr_val;
} hi_venc_stream_info;
```

成员

成员名称	描述
ref_type	高级跳帧参考下的编码帧类型。

成员名称	描述
pic_bytes	已编码码流的字节数。
pic_cnt	当通道属性is_by_frame == 1时，表示帧数；当is_by_frame == 0时，表示包数。
start_qp	编码帧的起始QP。
mean_qp	编码帧的平均QP。
is_p_skip	是否P帧是skip帧。
residual_bits	编码当前帧残差bit数。
head_bits	编码当前帧头信息bit数。
madi_val	编码当前帧空域纹理复杂度Madi值。
madp_val	编码当前帧时域运动复杂度Madp值。
sse_sum	编码当前帧中SSE（和方差）值。 不同软件中关于和方差的计算公式略有不同，因此和方差值可能存在差异。
sse_lcu_cnt	编码当前帧中LCU个数。
psnr_val	编码当前帧的PSNR（峰值信噪比）值。 不同软件中关于峰值信噪比的计算公式略有不同，因此峰值信噪比值可能存在差异。 对于H264，仅支持输入图片分辨率的乘积小于4096*2032时获取psnr_val值，否则psnr_val值无效。 对于H265，仅支持输入图片分辨率的乘积小于8192*4064时获取psnr_val值，否则psnr_val值无效。

11.18.20.14.55 hi_venc_pack_info

说明

定义当前码流包数据中包含的其他类型码流包数据的结构体。

定义

```
typedef struct {
    hi_venc_data_type pack_type;
    hi_u32 pack_offset;
    hi_u32 pack_len;
} hi_venc_pack_info;
```

成员

成员名称	描述
pack_type	当前码流包数据包含其他码流包的类型。
pack_offset	当前码流包数据包含其他码流包数据的偏移。
pack_len	当前码流包数据包含其他码流包数据的大小。

11.18.20.14.56 hi_frame_rate_ctrl

说明

定义通道帧率控制参数，暂不支持。

定义

```
typedef struct {
    hi_s32 src_frame_rate;
    hi_s32 dst_frame_rate;
} hi_frame_rate_ctrl;
```

成员

成员名称	描述
src_frame_rate	进入通道的帧率，以fps为单位。 取值范围：-1 或[1, 240]。
dst_frame_rate	通道输出的帧率，以fps为单位。 取值范围： ● Jpeg：-1 或[1, src_frame_rate]； ● H.264/H.265：-1 或[1, 240]。

11.18.20.14.57 hi_venc_rc_qpmap_mode

说明

定义编码通道QPMAP模式下CU32、CU64 QP值的取值方式，暂不支持。

定义

```
typedef enum {
    HI_VENC_RC_QPMAP_MODE_MEAN_QP = 0,
    HI_VENC_RC_QPMAP_MODE_MIN_QP,
    HI_VENC_RC_QPMAP_MODE_MAX_QP,

    HI_VENC_RC_QPMAP_MODE_BUTT,
} hi_venc_rc_qpmap_mode;
```

成员

成员名称	描述
HI_VENC_RC_QPMAP_MODE_MEAN_QP	CU32 QP值取4个16*16块Qp值的平均值； CU64 QP值取16个16*16块Qp值的平均值。
HI_VENC_RC_QPMAP_MODE_MIN_QP	CU32 QP值取4个16*16块Qp值的最小值； CU64 QP值取16个16*16块Qp值的最小值。
HI_VENC_RC_QPMAP_MODE_MAX_QP	CU32 QP值取4个16*16块Qp值的最大值； CU64 QP值取16个16*16块Qp值的最大值。
HI_VENC_RC_QPMAP_MODE_BUTT	保留值。

11.18.20.14.58 hi_venc_sse_info

说明

定义H264/H265协议SSE信息，暂不支持。

定义

```
typedef struct {
    hi_bool enable;
    hi_u64 sse_val;
} hi_venc_sse_info;
```

成员

成员名称	描述
enable	区域SSE使能。1表示使能，0表示不使能。
sse_val	区域SSE值。

11.18.20.14.59 hi_venc_h264_attr

说明

定义H.264编码属性结构体。

Atlas 推理系列产品，预留结构体，暂不支持。

定义

```
typedef struct {
    hi_bool rcn_ref_share_buf_en;
    hi_u32 frame_buf_ratio;
} hi_venc_h264_attr;
```

成员

成员名称	描述
rcn_ref_share_buf_en	是否使能参考帧和回写帧复用buffer。预留参数，暂不支持。 取值范围：[0,1]。1表示使能，0表示不使能。 静态属性。
frame_buf_ratio	帧buffer比例，设置该比例后，系统内部会根据该比例压缩帧buffer。 取值范围：[70,100]。比如将frame_buf_ratio设置为80，则表示帧buffer为原始大小的80%。 ● 默认值为100，表示不压缩帧buffer； ● 非取值范围内的值，表示不压缩帧buffer； ● [70,100)范围内的值，表示需按比例压缩帧buffer，可能会影响编码质量。建议在图片纹理不复杂、分辨率大于640*480等场景下，对于内存带宽有要求时，可以设置成[70,100)范围内的值。

11.18.20.14.60 hi_venc_h265_attr

说明

定义H.265编码属性结构体。

Atlas 推理系列产品，预留结构体，暂不支持。

定义

```
typedef struct {
    hi_bool rcn_ref_share_buf_en;
    hi_u32 frame_buf_ratio;
} hi_venc_h265_attr;
```

成员

成员名称	描述
rcn_ref_share_buf_en	是否使能参考帧和回写帧复用buffer。预留参数，暂不支持。 取值范围：[0,1]。1表示使能，0表示不使能。 静态属性。

成员名称	描述
frame_buf_ratio	帧buffer比例，设置该比例后，系统内部会根据该比例压缩帧buffer。 取值范围：[70,100]。比如将frame_buf_ratio设置为80，则表示帧buffer为原始大小的80%。 ● 默认值为100，表示不压缩帧buffer； ● 非取值范围内的值，表示不压缩帧buffer； ● [70,100)范围内的值，表示需按比例压缩帧buffer，可能会影响编码质量。建议在图片纹理不复杂、分辨率大于640*480等场景下，对于内存带宽有要求时，可以设置成[70,100)范围内的值。

11.18.20.14.61 hi_venc_jpeg_attr

说明

定义JPEG编码属性结构体，暂不支持。

定义

```
typedef enum {
    HI_VENC_PIC_RECV_SINGLE = 0,
    HI_VENC_PIC_RECV_MULTI,

    HI_VENC_PIC_RECV_BUTT
} hi_venc_pic_recv_mode;

typedef struct {
    hi_bool dcf_en;
    hi_venc_mpf_cfg mpf_cfg;
    hi_venc_pic_recv_mode recv_mode;
} hi_venc_jpeg_attr;
```

11.18.20.14.62 hi_venc_prores_attr

说明

定义PRORES属性结构体，暂不支持。

定义

```
typedef enum {
    HI_VENC_PRORES_FRAME_RATE_UNKNOWN = 0,
    HI_VENC_PRORES_FRAME_RATE_23_976,
    HI_VENC_PRORES_FRAME_RATE_24,
    HI_VENC_PRORES_FRAME_RATE_25,
    HI_VENC_PRORES_FRAME_RATE_29_97,
    HI_VENC_PRORES_FRAME_RATE_30,
    HI_VENC_PRORES_FRAME_RATE_50,
    HI_VENC_PRORES_FRAME_RATE_59_94,
    HI_VENC_PRORES_FRAME_RATE_60,
    HI_VENC_PRORES_FRAME_RATE_100,
```

```
    HI_VENC_PRORES_FRAME_RATE_119_88,
    HI_VENC_PRORES_FRAME_RATE_120,
    HI_VENC_PRORES_FRAME_RATE_BUTT
} hi_venc_prores_frame_rate;

// the aspect ratio of PRORES
typedef enum {
    HI_VENC_PRORES_ASPECT_RATIO_UNKNOWN = 0,
    HI_VENC_PRORES_ASPECT_RATIO_SQUARE,
    HI_VENC_PRORES_ASPECT_RATIO_4_3,
    HI_VENC_PRORES_ASPECT_RATIO_16_9,
    HI_VENC_PRORES_ASPECT_RATIO_BUTT
} hi_venc_prores_aspect_ratio;


// the attribute of PRORES
typedef struct {
    hi_char identifier[HI_VENC_PRORES_MAX_ID_CHAR_NUM];
    hi_venc_prores_frame_rate frame_rate_code;
    hi_venc_prores_aspect_ratio aspect_ratio;
} hi_venc_prores_attr;
```

11.18.20.14.63 hi_venc_scene_mode

说明

定义编码模式场景。

定义

```
typedef enum {
    HI_VENC_SCENE_0 = 0,  // 摄像机不运动或周期性连续运动的场景，支持h.264/h.265
    HI_VENC_SCENE_1 = 1,  // 高码率下运动场景，支持h.265
    HI_VENC_SCENE_2 = 2,  // 暂不支持
    HI_VENC_SCENE_BUTT
}hi_venc_scene_mode;
```

安防场景配置为HI_VENC_SCENE_0；自动驾驶、直播、游戏、动画、电影配置为HI_VENC_SCENE_1。

11.18.20.14.64 hi_venc_rc_param

说明

定义编码通道码率控制器的高级参数。

定义

```
typedef struct {
    hi_u32 threshold_i[HI_VENC_TEXTURE_THRESHOLD_SIZE];
    hi_u32 threshold_p[HI_VENC_TEXTURE_THRESHOLD_SIZE];
    hi_u32 threshold_b[HI_VENC_TEXTURE_THRESHOLD_SIZE];
    hi_u32 direction;
    hi_u32 row_qp_delta;
    hi_s32 first_frame_start_qp;
    hi_venc_scene_chg_detect scene_chg_detect;
    union {
        hi_venc_h264_cbr_param h264_cbr_param;
        hi_venc_h264_vbr_param h264_vbr_param;
        hi_venc_h264_avbr_param h264_avbr_param;
        hi_venc_h264_qvbr_param h264_qvbr_param;
        hi_venc_h264_cvbr_param h264_cvbr_param;
        hi_venc_h265_cbr_param h265_cbr_param;
        hi_venc_h265_vbr_param h265_vbr_param;
```

```
        hi_venc_h265_avbr_param h265_avbr_param;
        hi_venc_h265_qvbr_param h265_qvbr_param;
        hi_venc_h265_cvbr_param h265_cvbr_param;
        hi_venc_mjpeg_cbr_param mjpeg_cbr_param;
        hi_venc_mjpeg_vbr_param mjpeg_vbr_param;
    };
}hi_venc_rc_param;
```

成员

成员名称	描述
threshold_i	I帧宏块级码率控制的Madi（用于度量当前帧的空域纹理复杂度）门限。 取值范围：[0, 255]。 默认值：[0,0,0,0,3,3,5,5,8,8,8,15,15,20,25,25]。 减方向的数值设置为0，表示关闭当前级；加方向的数值设置为255，表示关闭当前级。
threshold_p	P帧宏块级码率控制的Madi门限。 取值范围：[0, 255]。 默认值：[0,0,0,0,3,3,5,5,8,8,8,15,15,20,25,25]。 减方向的数值设置为0，表示关闭当前级；加方向的数值设置为255，表示关闭当前级。
threshold_b	B帧宏块级码率控制的Madi门限。 取值范围：[0, 255]。 默认值：[0,0,0,0,3,3,5,5,8,8,8,15,15,20,25,25]。 减方向的数值设置为0，表示关闭当前级；加方向的数值设置为255，表示关闭当前级。
direction	在基于纹理宏块级码率控制时，用于控制加减方向。 取值范围：[0, 16]。 默认值：8。 举例，direction=7 表示threshold_i、threshold_p、threshold_b的前7个参数用于QP减方向；后9个参数用于QP加方向。
row_qp_delta	行级码率控制调节幅度是一帧内行级调节的最大范围，其中行级以宏块行为单位。调节幅度越大，允许行级调整的QP范围越大，码率越平稳。对于图像复杂度分布不均匀的场景，行级码率控制调节幅度设置过大会带来图像质量不均匀。 取值范围：[0, 10]，设置为0表示关闭基于行的宏块级码率控制。 默认值：1。

成员名称	描述
first_frame_start_qp	设置第一帧的起始Qp值，CBR/VBR/AVBR/QVBR/CVBR模式下，该参数有效。此处第一帧的含义是：通道创建，GOP模式切换，RC模式切换，或分辨率切换后，序列的第一个IDR帧。需要注意如果第一帧编码完成后判定为重编，重编的帧不是第一帧，不受first_frame_start_qp的约束。 取值范围：[MinIQP, MaxIQP]和-1。MinIQP参数值、MaxIQP参数值在码率控制模式的高级参数结构体内设置。 默认值：-1。如果为-1，则第一帧的起始QP由编码器内部计算，如果为其它合法值则由用户通过该参数设置第一帧的起始QP。 在低码率、少量运动场景下，如果编码效果不好，可以通过降低该参数值来提升图像质量和效果，例如，在“352*288分辨率，帧率25fps，GOP=50”场景下，将first_frame_start_qp设置为32，同时将max_reencode_times参数设置为0（表示不重编码，该参数是以下码率控制模式高级参数之一）。
scene_chg_detect	场景检测相关的控制参数，自适应检测当前编码场景是否发生变化，使能后会改善场景切换时的码率波动，VBR/AVBR/QVBR/CVBR 有效。使能场景切换检测时可使能自适应插入IDR帧，在场景切换时编码IDR帧，需要注意关闭重编后检测场景切换会失效。该参数建议与编码场景模式的SCENE_0模式配合使用。
h264_cbr_param	H.264通道CBR（Constant Bit Rate）码率控制模式高级参数。
h264_vbr_param	H.264通道VBR（Variable Bit Rate）码率控制模式高级参数。
h264_avbr_param	H.264通道AVBR（Adaptive Variable Bit Rate）码率控制模式高级参数。
h264_qvbr_param	H.264通道QVBR (Quality Variable Bit Rate)码率控制模式高级参数。
h264_cvbr_param	H.264通道CVBR (Constrained Variable Bit Rate)码率控制模式高级参数。
h265_cbr_param	H.265通道CBR（Constant Bit Rate）码率控制模式高级参数。
h265_vbr_param	H.265通道VBR（Variable Bit Rate）码率控制模式高级参数。

成员名称	描述
h265_avbr_param	H.265通道AVBR（Adaptive Variable Bit Rate）码率控制模式高级参数。
h265_qvbr_param	H.265通道QVBR (Quality Variable Bit Rate) 码率控制模式高级参数。
h265_cvbr_param	H.265通道CVBR (Constrained Variable Bit Rate)码率控制模式高级参数。
mjpeg_cbr_param	MJPEG通道CBR（Constant Bit Rate）码率控制模式高级参数。预留参数，暂不支持。
mjpeg_vbr_param	MJPEG通道VBR（Variable Bit Rate）码率控制模式高级参数。预留参数，暂不支持。

11.18.20.14.65 hi_venc_h264_cbr_param

说明

定义H264协议编码通道CBR码率控制模式的高级参数。

定义

```
typedef struct {
    hi_u32 max_i_proportion;
    hi_u32 min_i_proportion;
    hi_u32 max_qp;
    hi_u32 min_qp;
    hi_u32 max_i_qp;
    hi_u32 min_i_qp;
    hi_s32 max_reencode_times;
    hi_bool qpmap_en;
}hi_venc_h264_cbr_param;
```

成员

成员名称	描述
min_i_proportion	最小IP帧比例，预留参数，暂不支持。
max_i_proportion	最大IP帧比例。 取值范围：[1, 100]。 该参数默认值为100。
max_qp	帧最大QP，用于钳位质量。 取值范围：[0, 51]。 默认值：51。
min_qp	帧最小QP，用于钳位码率波动。 取值范围：[0, max_qp]。 默认值：10。

成员名称	描述
min_i_qp	I帧的最小QP。用于控制I帧的最大bits数。 取值范围：[0, max_i_qp]。 默认值：10。
max_i_qp	I帧的最大QP。用于控制I帧的最小bits数。 取值范围：[0, 51]。 默认值：51。
max_reencode_times	每帧重编码次数。0表示不进行重编码。 取值范围：[0, 3]。 默认值：2。
qpmap_en	预留参数，暂不支持。

11.18.20.14.66 hi_venc_h264_vbr_param

说明

定义H264协议编码通道VBR码率控制模式的高级参数。

定义

```
typedef struct {
    hi_s32 chg_pos;
    hi_u32 max_i_proportion;
    hi_u32 min_i_proportion;
    hi_s32 max_reencode_times;
    hi_bool qpmap_en;
    hi_u32 max_qp;
    hi_u32 min_qp;
    hi_u32 max_i_qp;
    hi_u32 min_i_qp;
} hi_venc_h264_vbr_param;
```

成员

max_qp、min_qp用于控制图像的质量范围，max_bit_rate用于钳位码率统计时间内的最大编码码率，chg_pos用于控制开始调整QP的码率基准线。

当编码码率大于max_bit_rate*chg_pos时，图像QP会逐步向max_qp调整，如果图像QP达到max_qp，QP会被钳位到最大值，max_bit_rate的钳位效果失效，编码码率有可能会超出max_bit_rate。

当编码码率小于max_bit_rate*chg_pos时，图像QP会逐步向min_qp调整，如果图像QP达到min_qp，此时编码的码率已经达到最大值，而且图像质量最好。

说明

max_bit_rate参数在创建VENC通道时设置，是hi_venc_chn_attr.rc_attr结构体内的成员变量。

成员名称	描述
chg_pos	VBR开始调整Qp时的码率相对于最大码率的比例。 取值范围：[50, 100]。 默认值：90。
min_i_proportion	预留参数，暂不支持。
max_i_proportion	最大IP帧码率的比值。 取值范围：[1,100]。 该参数默认值为100。
max_reencode_times	每帧重编码次数。0 表示不进行重编码。 取值范围：[0, 3]。 默认值：2。
qpmap_en	预留参数，暂不支持。
max_qp	P、B帧的最大QP。取值范围：[0, 51]。 默认值：51。
min_qp	P、B帧的最小QP。 取值范围：[0, max_qp]。 默认值：24。
max_i_qp	I帧的最大QP。 取值范围：[0, 51]。 默认值：51。
min_i_qp	I帧的最小QP。 取值范围：[0, max_i_qp]。 默认值：24。

11.18.20.14.67 hi_venc_h264_avbr_param

说明

定义H264协议编码通道AVBR码率控制模式的高级参数。

定义

```
typedef struct {
    hi_s32 chg_pos;
    hi_u32 max_i_proportion;
    hi_u32 min_i_proportion;
    hi_s32 max_reencode_times;
    hi_bool qpmap_en;
    hi_s32 min_still_percent;
    hi_u32 max_still_qp;
    hi_u32 min_still_psnr;
    hi_u32 max_qp;
```

```
    hi_u32 min_qp;
    hi_u32 max_i_qp;
    hi_u32 min_i_qp;
    hi_u32 min_qp_delta;
    hi_u32 motion_sensitivity;
    hi_bool save_bitrate_en;
}hi_venc_h264_avbr_param;
```

成员

max_bit_rate表示运动场景下的最大码率，max_bit_rate*chg_pos*min_still_percent表示静止情况下的最小码率。

根据运动程度的不同，目标码率会在最大码率和最小码率间调整。max_qp，min_qp用于控制图像的质量范围，码率控制以QP钳位为最高优先级，超出min_qp，max_qp范围内码率控制将失效。

说明

max_bit_rate参数在创建VENC通道时设置，是hi_venc_chn_attr.rc_attr结构体内的成员变量。

成员名称	描述
chg_pos	AVBR开始调整Qp时的码率相对于最大码率的比例。 取值范围：[50, 100]。 默认值：65。
min_i_proportion	预留参数，暂不支持。
max_i_proportion	最大IP帧码率的比值。 取值范围：[1,100]。 默认值：100。
max_reencode_times	每帧重编码次数。0 表示不进行重编码。 取值范围：[0, 3]。 默认值：2。
qpmap_en	预留参数，暂不支持。
min_still_percent	静止状态下目标码率的最小百分比。此变量设置为100，AVBR将不会在判别为静止时主动调低目标码率。 取值范围：[5, 100]。 默认值：25。
max_still_qp	静止场景I帧QP的最大值。 取值范围：[min_i_qp, max_i_qp]。 默认值：35。
min_still_psnr	预留参数，暂不支持。

成员名称	描述
max_qp	P、B帧的最大QP。 取值范围：[0, 51]。 默认值：51。
min_qp	P、B帧的最小QP。 取值范围：[0, max_qp]。 默认值：24。
max_i_qp	I帧的最大QP。 取值范围：[0, 51]。 默认值：51。
min_i_qp	I帧的最小QP。 取值范围：[0, max_i_qp]。 默认值：24。
min_qp_delta	帧级QP最小值和CU级QP最小值的差值。 I帧，帧级QP最小值= min_qp_delta + min_i_qp P、B帧，帧级QP最小值= min_qp_delta +min_qp 取值范围：[0, 4]； 默认值：0。
motion_sensitivity	运动敏感度。 取值范围：[0, 100]。 默认值：100。
save_bitrate_en	预留参数，暂不支持。

11.18.20.14.68 hi_venc_h264_qvbr_param

说明

定义H264协议编码通道QVBR码率控制模式的高级参数。

定义

```
typedef struct {
    hi_u32 max_i_proportion;
    hi_u32 min_i_proportion;
    hi_s32 max_reencode_times;
    hi_bool qpmap_en;
    hi_u32 max_qp;
    hi_u32 min_qp;
    hi_u32 max_i_qp;
    hi_u32 min_i_qp;
    hi_s32 max_bit_percent;
    hi_s32 min_bit_percent;
```

```
    hi_s32 max_psnr_fluctuate;
    hi_s32 min_psnr_fluctuate;
} hi_venc_h264_qvbr_param;
```

成员

当实时统计的PSNR小于max_psnr_fluctuate时，适当增加目标码率，最大码率=target_bit_rate* max_bit_percent。

当实时统计的PSNR大于max_psnr_fluctuate时，适当减小目标码率，最小码率=target_bit_rate* min_bit_percent。

根据当前PSNR的不同，目标码率会在最大码率和最小码率间调整，当PSNR值范围超过[min_psnr_fluctuate-4,max_psnr_fluctuate+4]∩[20,40]时，PSNR不再起作用，码率会在最大码率和最小码率间调整。

max_qp，min_qp用于控制图像的质量范围，码率控制以QP钳位为最高优先级，超出min_qp，max_qp范围内码率控制将失效。

码率浮动上下限的优先级高于PSNR的优先级，例如，当码率浮动到上限依然不能满足PSNR要求，则码率不会再继续上调。

说明

target_bit_rate参数在创建VENC通道时设置，是hi_venc_chn_attr.rc_attr结构体内的成员变量。

成员名称	描述
min_i_proportion	预留参数，暂不支持。
max_i_proportion	最大IP帧码率的比值。 取值范围：[1,100]。 默认值：100。
max_reencode_times	每帧重编码次数。0表示不进行重编码。 取值范围：[0, 3]。 默认值：2。
qpmap_en	预留参数，暂不支持。
max_qp	P、B帧的最大QP。 取值范围：[min_qp, 51]。 默认值：51。
min_qp	P、B帧的最小QP。 取值范围：[0, 51]。 默认值：16。
max_i_qp	I帧的最大QP。 取值范围：[min_i_qp, 51]。 默认值：51。

成员名称	描述
min_i_qp	I帧的最小QP。 取值范围：[0, 51]。 默认值：16。
max_bit_percent	码率百分比上限。 取值范围：[min_bit_percent, 180]。 默认值：110。
min_bit_percent	码率百分比下限。 取值范围：[30, 180]。 默认值：45。
max_psnr_fluctuate	Psnr上限。 取值范围：[min_psnr_fluctuate, 40]。 默认值：40。
min_psnr_fluctuate	Psnr下限。 取值范围：[18, 40]。 默认值：23。

11.18.20.14.69 hi_venc_h264_cvbr_param

说明

定义H264协议编码通道CVBR码率控制模式的高级参数。

定义

```
typedef struct {
    hi_u32 max_i_proportion;
    hi_u32 min_i_proportion;
    hi_s32 max_reencode_times;
    hi_bool qpmap_en;
    hi_u32 max_qp;
    hi_u32 min_qp;
    hi_u32 max_i_qp;
    hi_u32 min_i_qp;
    hi_u32 min_qp_delta;
    hi_u32 max_qp_delta;
    hi_u32 extra_bit_percent;
    hi_u32 long_term_stats_time_unit;
    hi_bool save_bitrate_en;
} hi_venc_h264_cvbr_param;
```

成员

成员名称	描述
min_i_proportion	预留参数，暂不支持。

成员名称	描述
max_i_proportion	最大IP帧码率的比值。 取值范围：[1,100]。 默认值：100。
max_reencode_times	每帧重编码次数。0 表示不进行重编码。 取值范围：[0, 3]。 默认值：2。
qpmap_en	预留参数，暂不支持。
max_qp	P、B帧的最大QP。 取值范围：[min_qp, 51]。 默认值：47。
min_qp	P、B帧的最小QP。 取值范围：[0, 51]。 默认值：22。
max_i_qp	I帧的最大QP。 取值范围：[min_i_qp, 51]。 默认值：47。
min_i_qp	I帧的最小QP。 取值范围：[0, 51]。 默认值：20。
min_qp_delta	帧级QP最小值和CU级QP最小值的差值。 I帧，帧级QP最小值 = min_qp_delta + min_i_qp P、B帧，帧级QP最小值= min_qp_delta +min_qp 取值范围：[0, 4]。 默认值：0。
max_qp_delta	帧级QP最大值和CU级QP最大值的差值。 I帧，帧级QP最大值 = max_i_qp - max_qp_delta P、B 帧：帧级QP最大值= max_qp - max_qp_delta 取值范围：[0, 4]。 默认值：0。

成员名称	描述
extra_bit_percent	编码器输出码流最大透支bit 数百分比。 在码率不足时，为保证图像质量，编码器会通过透支一定的bit数以提升图像质量，这部分透支的码率会在编码压力较小时进行偿还。 取值范围：[0, 1000]。 默认值：5。
long_term_stats_time_unit	码率长期统计时间参数（long_term_stats_time）的单位，本变量单位是秒（s）,如配置long_term_stats_time_unit = 60，long_term_stats_time =3，代表长期统计时间为3 分钟。 取值范围：[1, 1800]。 默认值：60。
save_bitrate_en	预留参数，暂不支持。

11.18.20.14.70 hi_venc_h265_vbr_param

说明

定义H265协议编码通道VBR码率控制模式的高级参数。

定义

```
typedef struct {
    hi_s32 chg_pos;
    hi_u32 max_i_proportion;
    hi_u32 min_i_proportion;
    hi_s32 max_reencode_times;
    hi_u32 max_qp;
    hi_u32 min_qp;
    hi_u32 max_i_qp;
    hi_u32 min_i_qp;
    hi_bool qpmap_en;
    hi_venc_rc_qpmap_mode qpmap_mode;
}hi_venc_h265_vbr_param;
```

成员

max_qp、min_qp用于控制图像的质量范围，max_bit_rate用于钳位码率统计时间内的最大编码码率，chg_pos用于控制开始调整QP的码率基准线。

当编码码率大于max_bit_rate*chg_pos时，图像QP会逐步向max_qp调整，如果图像QP达到max_qp，QP会被钳位到最大值，max_bit_rate的钳位效果失效，编码码率有可能会超出max_bit_rate。

当编码码率小于max_bit_rate*chg_pos时，图像QP会逐步向min_qp调整，如果图像QP达到min_qp，此时编码的码率已经达到最大值，而且图像质量最好。

说明

max_bit_rate参数在创建VENC通道时设置，是hi_venc_chn_attr.rc_attr结构体内的成员变量。

成员名称	描述
chg_pos	开始调整Qp时的码率相对于最大码率的比例。 取值范围：[50, 100]。 默认值：90。
max_i_proportion	最大IP 帧码率的比值。 取值范围：[1,100]。 该参数默认值为100。
min_i_proportion	预留参数，暂不支持。
max_reencode_times	每帧重编码次数。0 表示不进行重编码。 取值范围：[0, 3]。 默认值：2。
max_qp	P帧的最大QP。 取值范围：[0, 51]。 默认值：51。
min_qp	P帧的最小QP。 取值范围：[0, max_qp]。 默认值：24。
max_i_qp	I帧的最大QP。 取值范围：[0, 51]。 默认值：51。
min_i_qp	I帧的最小QP。 取值范围：[0, max_i_qp]。 默认值：24。
qpmap_en	预留参数，暂不支持。
qpmap_mode	预留参数，暂不支持。

11.18.20.14.71 hi_venc_h265_avbr_param

说明

定义H265协议编码通道AVBR码率控制模式的高级参数。

定义

```
typedef struct {
    hi_s32 chg_pos;
    hi_u32 max_i_proportion;
```

```
    hi_u32 min_i_proportion;
    hi_s32 max_reencode_times;
    hi_s32 min_still_percent;
    hi_u32 max_still_qp;
    hi_u32 min_still_psnr;
    hi_u32 max_qp;
    hi_u32 min_qp;
    hi_u32 max_i_qp;
    hi_u32 min_i_qp;
    hi_u32 min_qp_delta;
    hi_u32 motion_sensitivity;
    hi_bool qpmap_en;
    hi_venc_rc_qpmap_mode qpmap_mode;
}hi_venc_h265_avbr_param;
```

成员

max_bit_rate表示运动场景下的最大码率，max_bit_rate*chg_pos*min_still_percent表示静止情况下的最小码率。

根据运动程度的不同，目标码率会在最大码率和最小码率间调整。max_qp，min_qp用于控制图像的质量范围，码率控制以QP钳位为最高优先级，超出min_qp，max_qp范围内码率控制将失效。

说明

max_bit_rate参数在创建VENC通道时设置，是hi_venc_chn_attr.rc_attr结构体内的成员变量。

成员名称	描述
chg_pos	开始调整Qp 时的码率相对于最大码率的比例。 取值范围：[50, 100]。 默认值：65。
min_i_proportion	预留参数，暂不支持。
max_i_proportion	最大IP帧码率的比值。 取值范围：[1,100]。 默认值：100。
max_reencode_times	每帧重编码次数。0 表示不进行重编码。 取值范围：[0, 3]。 默认值：2。
min_still_percent	静止状态下目标码率的最小百分比。此变量设置为100，AVBR 将不会在判别为静止时主动调低目标码率。 取值范围：[5, 100]。 默认值：25。
max_still_qp	静止场景I帧QP的最大值。 取值范围：[min_i_qp, max_i_qp]。 默认值：35。
min_still_psnr	预留参数，暂不支持。

成员名称	描述
max_qp	P帧的最大QP。 取值范围：[0, 51]。 默认值：51。
min_qp	P帧的最小QP。 取值范围：[0, max_qp]。 默认值：24。
max_i_qp	I帧的最大QP。 取值范围：[0, 51]。 默认值：51。
min_i_qp	I帧的最小QP。 取值范围：[0, max_i_qp]。 默认值：24。
min_qp_delta	帧级QP最小值和CU级QP最小值的差值。 I帧：FrameLevelMinQp = min_qp_delta + min_i_qp P、B 帧：FrameLevelMinQp= min_qp_delta +min_qp 取值范围：[0, 4]； 默认值：0。
motion_sensitivity	运动敏感度。 取值范围：[0, 100]。 默认值：100。
qpmap_en	预留参数，暂不支持。
qpmap_mode	预留参数，暂不支持。

11.18.20.14.72 hi_venc_h265_qvbr_param

说明

定义H265协议编码通道QVBR码率控制模式的高级参数。

定义

```
typedef struct {
    hi_u32 max_i_proportion;
    hi_u32 min_i_proportion;
    hi_s32 max_reencode_times;
    hi_bool qpmap_en;
    hi_venc_rc_qpmap_mode qpmap_mode;
    hi_u32 max_qp;
    hi_u32 min_qp;
    hi_u32 max_i_qp;
```

```
    hi_u32 min_i_qp;
    hi_s32 max_bit_percent;
    hi_s32 min_bit_percent;
    hi_s32 max_psnr_fluctuate;
    hi_s32 min_psnr_fluctuate;
} hi_venc_h265_qvbr_param;
```

成员

当实时统计的PSNR小于max_psnr_fluctuate时，适当增加目标码率，最大码率=target_bit_rate* max_bit_percent。

当实时统计的PSNR大于max_psnr_fluctuate时，适当减小目标码率，最小码率=target_bit_rate* min_bit_percent。

根据当前PSNR的不同，目标码率会在最大码率和最小码率间调整，当PSNR值范围超过[min_psnr_fluctuate-4,max_psnr_fluctuate+4]∩[20,40]时，PSNR不再起作用，码率会在最大码率和最小码率间调整。

max_qp，min_qp用于控制图像的质量范围，码率控制以QP钳位为最高优先级，超出min_qp，max_qp范围内码率控制将失效。

码率浮动上下限的优先级高于PSNR的优先级，例如，当码率浮动到上限依然不能满足PSNR要求，则码率不会再继续上调。

说明

target_bit_rate参数在创建VENC通道时设置，是hi_venc_chn_attr.rc_attr结构体内的成员变量。

成员名称	描述
min_i_proportion	预留参数，暂不支持。
max_i_proportion	最大IP 帧码率的比值。 取值范围：[1,100]。 默认值：100。
max_reencode_times	每帧重编码次数。0 表示不进行重编码。 取值范围：[0, 3]。 默认值：2。
qpmap_en	预留参数，暂不支持。
qpmap_mode	预留参数，暂不支持。
max_qp	P、B帧的最大QP。 取值范围：[min_qp, 51]。 默认值：51。
min_qp	P、B帧的最小QP。 取值范围：[0, 51]。 默认值：16。

成员名称	描述
max_i_qp	I帧的最大QP。 取值范围：[min_i_qp, 51]。 默认值：51。
min_i_qp	I帧的最小QP。 取值范围：[0, 51]。 默认值：16。
max_bit_percent	码率百分比上限。 取值范围：[min_bit_percent, 180]。 默认值：110。
min_bit_percent	码率百分比下限。 取值范围：[30, 180]。 默认值：45。
max_psnr_fluctuate	Psnr上限。 取值范围：[min_psnr_fluctuate, 40]。 默认值：40。
min_psnr_fluctuate	Psnr下限。 取值范围：[18, 40]。 默认值：23。

11.18.20.14.73 hi_venc_h265_cvbr_param

说明

定义H265协议编码通道CVBR码率控制模式的高级参数。

定义

```
typedef struct {
    hi_u32 max_i_proportion;
    hi_u32 min_i_proportion;
    hi_s32 max_reencode_times;
    hi_bool qpmap_en;
    hi_venc_rc_qpmap_mode qpmap_mode;
    hi_u32 max_qp;
    hi_u32 min_qp;
    hi_u32 max_i_qp;
    hi_u32 min_i_qp;
    hi_u32 min_qp_delta;
    hi_u32 max_qp_delta;
    hi_u32 extra_bit_percent;
    hi_u32 long_term_stats_time_unit;
} hi_venc_h265_cvbr_param;
```

成员

成员名称	描述
min_i_proportion	预留参数，暂不支持。
max_i_proportion	最大IP 帧码率的比值。 取值范围：[1,100]。 默认值：100。
max_reencode_times	每帧重编码次数。0 表示不进行重编码。 取值范围：[0, 3]。 默认值：2。
qpmap_en	预留参数，暂不支持。
qpmap_mode	预留参数，暂不支持。
max_qp	P、B帧的最大QP。 取值范围：[min_qp, 51]。 默认值：47。
min_qp	P、B帧的最小QP。 取值范围：[0, 51]。 默认值：22。
max_i_qp	I帧的最大QP。 取值范围：[min_i_qp, 51]。 默认值：47。
min_i_qp	I帧的最小QP。 取值范围：[0, 51]。 默认值：20。
min_qp_delta	帧级QP最小值和CU级QP最小值的差值。 I 帧：FrameLevelMinQp = min_qp_delta + min_i_qp P、B帧：FrameLevelMinQp= min_qp_delta +min_qp 取值范围：[0, 4]。 默认值：0。
max_qp_delta	帧级QP最大值和CU级QP最大值的差值。 I帧，帧级QP最大值 = max_i_qp - max_qp_delta P、B帧，帧级QP最大值= max_qp - max_qp_delta 取值范围：[0, 4]。 默认值：0。

成员名称	描述
extra_bit_percent	编码器输出码流最大透支bit 数百分比。 在码率不足时，为保证图像质量，编码器会通过透支一定的bit 数以提升图像质量，这部分透支的码率会在编码压力较小时进行偿还。 取值范围[0, 1000]。 默认值：5。
long_term_stats_time_unit	码率长期统计时间参数（long_term_stats_time）的单位，本变量单位是秒（s）,如配置long_term_stats_time_unit = 60，long_term_stats_time =3，代表长期统计时间为3 分钟。 取值范围：[1, 1800]。 默认值：60。

11.18.20.14.74 hi_venc_h265_cbr_param

说明

定义H265协议编码通道CBR码率控制模式的高级参数。

定义

```
typedef struct {
    hi_u32 max_i_proportion;
    hi_u32 min_i_proportion;
    hi_u32 max_qp;
    hi_u32 min_qp;
    hi_u32 max_i_qp;
    hi_u32 min_i_qp;
    hi_s32 max_reencode_times;
    hi_bool qpmap_en;
    hi_venc_rc_qpmap_mode qpmap_mode;
}hi_venc_h265_cbr_param;
```

成员

成员名称	描述
min_i_proportion	预留参数，暂不支持。
max_i_proportion	最大IP帧比例。 取值范围：[1, 100]。 该参数默认值为100。
max_qp	帧最大QP，用于钳位质量。 取值范围：[0, 51]。 默认值：51。

成员名称	描述
min_qp	帧最小QP，用于钳位码率波动。 取值范围：[0, max_qp]。 默认值：10。
max_i_qp	I帧的最大QP。用于控制I帧的最小bits数。 取值范围：[0, 51]。 默认值：51。
min_i_qp	I帧的最小QP。用于控制I帧的最大bits数。 取值范围：[0, max_i_qp]。 默认值：10。
max_reencode_times	每帧重编码次数。0表示不进行重编码。 取值范围：[0, 3]。 默认值：2。
qpmap_en	预留参数，暂不支持。
qpmap_mode	预留参数，暂不支持

11.18.20.14.75 hi_venc_scene_chg_detect

说明

定义编码场景检测控制参数。

定义

```
typedef struct {
    hi_bool detect_scene_chg_en;
    hi_bool adapt_insert_idr_frame_en;
}hi_venc_scene_chg_detect;
```

成员

成员名称	描述
detect_scene_chg_en	是否使能场景切换检测。 取值范围如下： ● HI_TRUE：是，默认为该值 ● HI_FALSE：否
adapt_insert_idr_frame_en	是否使能自适应插入IDR帧。 取值范围如下： ● HI_TRUE：是 ● HI_FALSE：否，默认为该值

11.18.20.14.76 hi_venc_huffman_dc_table

说明

用于定义指定JPEGE编码通道的亮度DC huffman编码表参数的数据结构。

定义

```
typedef struct {
    hi_u8 dc_bits[16];
    hi_u8 dc_value[12];
} hi_venc_huffman_dc_table;
```

成员

成员名称	描述
dc_bits[]	dc_bits数组存放的是长度从1bit到16bit的码字分别对应的符号个数。
dc_value[]	dc_value数据存放的是码字对应的符号值。

11.18.20.14.77 hi_venc_huffman_ac_table

说明

用于定义指定JPEGE编码通道的色度AC huffman编码表参数的数据结构。

定义

```
typedef struct {
    hi_u8 ac_bits[16];
    hi_u8 ac_value[162];
} hi_venc_huffman_ac_table;
```

成员

成员名称	描述
ac_bits[]	ac_bits数组存放的是长度从1bit到16bit的码字分别对应的符号个数。
ac_value[]	ac_value数据存放的是码字对应的符号值。

11.18.20.14.78 hi_venc_jpeg_huffman_param

说明

用于定义指定JPEGE编码通道的huffman编码高级参数的数据结构。

定义

```
typedef struct {
    hi_venc_huffman_dc_table dc_tables[3];
    hi_venc_huffman_ac_table ac_tables[3];
    hi_u32 reserved[2];
} hi_venc_jpeg_huffman_param;
```

成员

成员名称	描述
dc_tables[]	亮度DC huffman编码表数据。 Cb分量和Cr分量需保持一致。
ac_tables[]	色度AC huffman编码表数据。 Cb分量和Cr分量需保持一致。
reserved[]	预留参数，为保证后续版本兼容性，请务必使用memset结构体方式进行清零初始化，在代码中必须避免显式对reserved字段进行访问。

11.18.20.14.79 hi_venc_mpf_cfg

说明

预留结构体，暂不支持。

定义

```
typedef struct {
    hi_u8   large_thumbnail_num;
    hi_video_size  large_thumbnail_size[HI_VENC_MAX_MPF_NUM];
} hi_venc_mpf_cfg;
```

11.18.20.14.80 hi_venc_mjpeg_cbr_param

说明

预留结构体，暂不支持。

定义

```
typedef struct {
    hi_u32      max_qfactor;
    hi_u32      min_qfactor;
} hi_venc_mjpeg_cbr_param;
```

11.18.20.14.81 hi_venc_mjpeg_vbr_param

说明

预留结构体，暂不支持。

定义

```
typedef struct {
    hi_s32      chg_pos;
    hi_u32      max_qfactor;
    hi_u32      min_qfactor;
} hi_venc_mjpeg_vbr_param;
```

11.18.20.14.82 hi_venc_h264_ref_slice_type

说明

定义H.264 Slice码流的参考类型，预留，暂不支持。

定义

```
typedef enum {
    HI_VENC_H264_REF_SLICE_FOR_1X = 1, // Reference slice for H264E_REF_MODE_1X
    HI_VENC_H264_REF_SLICE_FOR_2X = 2, // Reference slice for H264E_REF_MODE_2X
    HI_VENC_H264_REF_SLICE_FOR_4X = 5, // Reference slice for H264E_REF_MODE_4X
    HI_VENC_H264_REF_SLICE_FOR_BUTT // slice not for reference
} hi_venc_h264_ref_slice_type;
```

11.18.20.14.83 hi_venc_roi_attr

说明

定义编码感兴趣区域信息。

定义

```
typedef struct {
    hi_u32   idx;
    hi_bool  enable;
    hi_bool  is_abs_qp;
    hi_s32   qp;
    hi_rect  rect;
 } hi_venc_roi_attr;
```

成员

成员名称	描述
idx	ROI区域的索引，系统支持的索引范围为[0,7]，不支持超出这个范围的索引。 系统支持每个通道可设置8个ROI区域，系统内部按照0~7的索引号对ROI区域进行管理。ROI区域之间可以互相叠加，且当发生叠加时，ROI区域之间的优先级按照索引号0~7依次提高。
enable	是否使能这个ROI区域。 Cb分量和Cr分量需保持一致。

成员名称	描述
is_abs_qp	表示ROI区域的QP模式，指定当前的ROI区域采用绝对QP方式或是相对QP 。 ● HI_FALSE：相对QP ● HI_TURE：绝对QP
qp	QP值，当QP模式为HI_FALSE时，qp为QP偏移，qp范围[-51,51]，当QP模式为HI_TRUE时，qp为宏块QP值，qp范围[0,51]。
rect	指定当前的ROI区域的位置坐标和区域的大小。 ROI区域的起始点坐标必须在图像范围内，且必须16对齐；ROI区域必须在图像范围内，ROI区域的长宽必须是16对齐并且大于0。即，x、y、width、height必须是16对齐。

11.18.20.14.84 hi_venc_ref_param

说明

定义H.264/H.265编码的高级跳帧参考参数。

定义

```
typedef struct {
    hi_u32   base;
    hi_u32   enhance;
    hi_bool  pred_en;
 } hi_venc_ref_param;
```

成员

成员名称	描述
base	base层的周期。 取值范围：（0，+∞ ）。
enhance	enhance层的周期。 取值范围：[0, 255]。
pred_en	代表base层的帧是否被base层其他帧用作参考。当pred_en设置为HI_FALSE时，等同于base设置为无限大，base 层的所有帧都参考IDR 帧。

11.18.20.14.85 hi_venc_h264_vui

说明

定义H.264协议编码通道Vui结构体。

定义

```
typedef struct {
    hi_venc_vui_aspect_ratio          vui_aspect_ratio;
    hi_venc_vui_h264_time_info        vui_time_info;
    hi_venc_vui_video_signal          vui_video_signal;
    hi_venc_vui_bitstream_restric     vui_bitstream_restric;
} hi_venc_h264_vui;
```

成员

成员名称	描述
vui_aspect_ratio	预留参数，具体含义请参见H.264协议。
vui_time_info	Vui字段中时钟计数相关参数。
vui_video_signal	Vui字段中视频信号相关参数。
vui_bitstream_restric	预留参数，具体含义请参见H.264协议。

11.18.20.14.86 hi_venc_h265_vui

说明

定义H.265协议编码通道Vui结构体。

定义

```
typedef struct {
    hi_venc_vui_aspect_ratio        vui_aspect_ratio;
    hi_venc_vui_h265_time_info      vui_time_info;
    hi_venc_vui_video_signal        vui_video_signal;
    hi_venc_vui_bitstream_restric   vui_bitstream_restric;
} hi_venc_h265_vui;
```

成员

成员名称	描述
vui_aspect_ratio	预留参数，具体含义请参见H.265协议。
vui_time_info	Vui字段中时钟计数相关参数。
vui_video_signal	Vui字段中视频信号相关参数。
vui_bitstream_restric	预留参数，具体含义请参见H.265协议。

11.18.20.14.87 hi_venc_vui_aspect_ratio

说明

定义H.264/H.265协议编码通道Vui中AspectRatio信息的结构体。预留结构体，暂不支持用户配置。

定义

```
typedef struct {
    hi_u8  aspect_ratio_info_present_flag;
    hi_u8  aspect_ratio_idc;
    hi_u8  overscan_info_present_flag;
    hi_u8  overscan_appropriate_flag;
    hi_u16 sar_width;
    hi_u16 sar_height;
} hi_venc_vui_aspect_ratio;
```

成员

成员名称	描述
aspect_ratio_info_present_flag	具体含义请参见H.264/H.265协议，系统默认值为0。
aspect_ratio_idc	具体含义请参见H.264/H.265协议，系统默认值为1。
overscan_info_present_flag	具体含义请参见H.264/H.265协议，系统默认值为0。
overscan_appropriate_flag	具体含义请参见H.264/H.265协议，系统默认值为0。
sar_width	具体含义请参见H.264/H.265协议，系统默认值为1。
sar_height	具体含义请参见H.264/H.265协议，系统默认值为1。

11.18.20.14.88 hi_venc_vui_h264_time_info

说明

定义H.264协议编码通道Vui中时钟计数信息的结构体。

定义

```
typedef struct {
    hi_u8  timing_info_present_flag;
    hi_u8  fixed_frame_rate_flag;
    hi_u32 num_units_in_tick;
    hi_u32 time_scale;
} hi_venc_vui_h264_time_info;
```

成员

成员名称	描述
timing_info_present_flag	Vui字段中时钟计数信息是否生效的标志位，系统默认值为0。取值范围：0或1。表格中其它参数只有在该标志为1时才生效。
fixed_frame_rate_flag	具体含义请参见H.264协议，系统默认值为1。取值范围：0或1。
num_units_in_tick	时钟计数器加1所经过的时间单元数，系统默认值为1。取值范围：大于0。
time_scale	一秒钟时长所经过的时间单元数，系统默认值为60。取值范围：大于0。

注意事项

时钟计数用于标记视频帧信号，在本H.264编码器中，视频显示帧率可由根据以下公式计算：display_rate = time_scale / num_units_in_tick / 2

由于显示帧率只与time_scale和num_units_in_tick的比值有关，建议用户设置时保持num_units_in_tick值为1，通过time_scale参数调整显示帧率，此时显示帧率 = time_scale / 2。

显示帧率的最终决定权在于播放器，播放器也可以选择不参考TimeInfo信息。

11.18.20.14.89 hi_venc_vui_h265_time_info

说明

定义H.265协议编码通道Vui中时钟计数信息的结构体。

定义

```
typedef struct {
    hi_u32 timing_info_present_flag;
    hi_u32 num_units_in_tick;
    hi_u32 time_scale;
    hi_u32 num_ticks_poc_diff_one_minus1;
 } hi_venc_vui_h265_time_info;
```

成员

成员名称	描述
timing_info_present_flag	Vui字段中时钟计数信息是否生效的标志位，系统默认值为0。取值范围：0或1。表格中其它参数只有在该标志为1时才生效。
num_units_in_tick	时钟计数器加1所经过的时间单元数，系统默认值为1。取值范围：大于0。

成员名称	描述
time_scale	一秒钟时长所经过的时间单元数，系统默认值为60。取值范围：大于0。
num_ticks_poc_diff_one_minus1	具体含义请参见H.265协议，系统默认值为1。取值范围：[0, 2^32-2].

注意事项

时钟计数用于标记视频帧信号，在本H.265编码器中，视频显示帧率可由以下公式计算：display_rate = time_scale / num_units_in_tick

由于显示帧率只与time_scale和num_units_in_tick的比值有关，建议用户设置时保持num_units_in_tick值为1，通过time_scale参数调整显示帧率，此时显示帧率 = time_scale。

显示帧率的最终决定权在于播放器，播放器也可以选择不参考TimeInfo信息。

11.18.20.14.90 hi_venc_vui_video_signal

说明

定义H.264/H.265协议编码通道Vui中视频信号信息的结构体。

定义

```
typedef struct {
    hi_u8  video_signal_type_present_flag;
    hi_u8  video_format;
    hi_u8  video_full_range_flag;
    hi_u8  colour_description_present_flag;
    hi_u8  colour_primaries;
    hi_u8  transfer_characteristics;
    hi_u8  matrix_coefficients;
 } hi_venc_vui_video_signal;
```

成员

成员名称	描述
video_signal_type_present_flag	Vui字段中视频信号信息是否生效的标志位，系统默认值为1。取值范围：0或1。表格中其它参数只有在该标志为1时才生效。
video_format	具体含义请参见H.264/H.265协议，系统默认值为5。取值范围：H.264:[0, 7]，H.265:[0, 5]。
video_full_range_flag	具体含义请参见H.264/H.265协议，系统默认值为0。取值范围：0或1。置0表示像素值域受限，亮度取值范围是16～235，色度的取值范围是16～240。

成员名称	描述
colour_description_present_flag	具体含义请参见H.264/H.265协议，系统默认值为1，用户无需配置，系统内部根据当前图像实际信息填写。
colour_primaries	具体含义请参见H.264/H.265协议，系统默认值为1，用户无需配置，系统内部根据当前图像实际信息填写。
transfer_characteristics	具体含义请参见H.264/H.265协议，系统默认值为1，用户无需配置，系统内部根据当前图像实际信息填写。
matrix_coefficients	具体含义请参见H.264/H.265协议，系统默认值为1，用户无需配置，系统内部根据当前图像实际信息填写。

11.18.20.14.91 hi_venc_vui_bitstream_restric

说明

定义H.264/H.265协议编码通道Vui中Bitstream_Restriction信息的结构体。预留结构体，暂不支持用户配置。

定义

```
typedef struct {
    hi_u8  bitstream_restriction_flag;
 } hi_venc_vui_bitstream_restric;
```

成员

成员名称	描述
bitstream_restriction_flag	具体含义请参见H.265协议，系统默认值为0。

11.18.20.14.92 hi_venc_cu_prediction

说明

定义CU模式选择的倾向性配置结构体。

定义

```
typedef struct {
    hi_op_mode    pred_mode;
    hi_u32    intra32_cost;
    hi_u32    intra16_cost;
    hi_u32    intra8_cost;
    hi_u32    intra4_cost;
    hi_u32    inter64_cost;
    hi_u32    inter32_cost;
```

```
    hi_u32    inter16_cost;
    hi_u32    inter8_cost;
} hi_venc_cu_prediction;
```

成员

成员名称	描述
pred_mode	倾向性选择模式，支持自动模式、手动模式两种。
intra32_cost	Intra32倾向性调节，对应帧内预测32*32像素块尺寸模式，该值增大表示选择该模式的倾向性越小。 取值范围：[0, 15]。 默认值：8。 H.264该值无效。
intra16_cost	Intra16倾向性调节，对应帧内预测16*16像素块尺寸模式，该值增大表示选择该模式的倾向性越小。 取值范围：[0, 15]。 默认值：8。
intra8_cost	Intra8倾向性调节，对应帧内预测8*8像素块尺寸模式，该值增大表示选择该模式的倾向性越小。 取值范围：[0, 15]。 默认值：8。
intra4_cost	Intra4倾向性调节，对应帧内预测4*4像素块尺寸模式，该值增大表示选择该模式的倾向性越小。 取值范围：[0, 15]。 默认值：8。
inter64_cost	Inter64倾向性调节，对应帧间预测64*64像素块尺寸模式，该值增大表示选择该模式的倾向性越小。 取值范围：[0, 15]。 默认值：8。 H.264该值无效。
inter32_cost	Inter32倾向性调节，对应帧间预测32*32像素块尺寸模式，该值增大表示选择该模式的倾向性越小。 取值范围：[0, 15]。 默认值：8。 H.264该值无效。

成员名称	描述
inter16_cost	Inter16倾向性调节，对应帧间预测16*16像素块尺寸模式，该值增大表示选择该模式的倾向性越小。 取值范围：[0, 15]。 默认值：8。
inter8_cost	Inter8倾向性调节，对应帧间预测8*8像素块尺寸模式，该值增大表示选择该模式的倾向性越小。 取值范围：[0, 15]。 默认值：8。

11.18.20.14.93 hi_venc_intra_refresh

说明

P帧刷Islice控制参数。

定义

```
typedef struct {
    hi_bool                 refresh_enable;
    hi_venc_intra_refresh_mode intra_refresh_mode;
    hi_u32                  refresh_num;
    hi_u32                  req_i_qp;
} hi_venc_intra_refresh;
```

成员

成员名称	描述
refresh_enable	是否使能刷Islice功能。 ● 0：不使能，默认值； ● 1：使能 。
intra_refresh_mode	I宏块刷新模式，分为按行刷新和按列刷新。默认按行刷新。 Atlas 推理系列产品，仅支持按行刷新。

成员名称	描述
refresh_num	每次I宏块刷新行数或者列数，可以通过这个变量控制刷新的速度及码流的平稳程度。刷新行数或者列数越多，刷新的速度越快，但是码流平稳度越差；刷新的行数或者列数越少，刷新的速度越慢，但是码流平稳度越好。 需保证设置的refresh_num可以在一个GOP内完成Islice刷新，注意高级跳帧参考时只会在base层中的P帧（可被base层中其他帧的参考）进行刷新。refresh_num需满足**表11-64**中的计算公式。
req_i_qp	I帧QP值。 **hi_mpi_venc_set_intra_refresh**接口与**hi_mpi_venc_request_idr**接口配合使用时，该值用于控制插入的IDR帧的质量，该值越小，质量越好，插入的IDR帧大小越大。 取值范围：[0, 51]，默认值51。

参考信息

表 11-64 refresh_num 参数值计算公式

-	计算公式	备注
H.264，refresh_num默认值为： (pic_height +lcu_size-1)>>6 **H.265**，refresh_num默认值为： ((pic_height + lcu_size - 1)/lcu_size) >> 2	• 行：refresh_num*(max_refresh_frame_in_gop - 1) >= (pic_height + lcu_size - 1) / lcu_size • 列：refresh_num*(max_refresh_frame_in_gop - 1) >= (pic_width + lcu_size - 1) / lcu_size 计算公式中的/表示向下取整。	• GOP内P帧刷Islice最大帧数max_refresh_frame_in_gop参数值取值说明： – 无高级跳帧参考时：max_refresh_frame_in_gop = gop; – 有高级跳帧参考时：max_refresh_frame_in_gop = (gop + (base*(enhance+1) -1))/(base*(enhance+1))。 • 编码单元尺寸lcu_size参数值取值说明： – H.264时，该参数值固定为16； – H265时： – Atlas 推理系列产品，该参数值固定为64

11.18.20.14.94 hi_venc_intra_refresh_mode

说明

P帧刷Islice模式定义。

定义

```
typedef enum{
    INTRA_REFRESH_ROW = 0,                  /* 按行刷新 */
    INTRA_REFRESH_COLUMN,                    /* 按列刷新 */
    INTRA_REFRESH_BUTT
} hi_venc_intra_refresh_mode;
```

11.18.20.14.95 hi_venc_slice_split

说明

P帧刷Islice控制参数。

定义

```
typedef struct {
    hi_bool         split_enable;
    hi_u32          split_mode;
    hi_u32          split_size;
    hi_bool         slice_output_en;
    hi_u32          reserved[2];
} hi_venc_slice_split;
```

成员

成员名称	描述
split_enable	Slice分割是否使能。 ● 0：不使能，默认值； ● 1：使能 。
split_mode	Slice分割模式。 当前仅支持设置为1，表示按宏块行或者LCU行分割。
split_size	● H.264表示每个slice占的宏块行数 最大值：(图像高+15)/16 最小值：(最大值+15)/16 ● H.265表示每个slice占的LCU行数 最大值：(图像高+LCU -1)/LCU 最小值：(最大值+15)/16
slice_output_en	当前仅支持设置为false。
reserved[2]	预留参数，当前仅支持设置为0。

注意事项

split_mode=1时，对H265协议需要关注当前芯片的LCU大小。

- Atlas 推理系列产品，该参数值固定为64

11.18.20.15 PNGD 图像解码

11.18.20.15.1 hi_pngd_chn

说明

描述PNG图片解码通道。

定义

```
typedef hi_s32 hi_pngd_chn;
```

11.18.20.15.2 hi_pngd_chn_attr

说明

定义PNGD解码通道属性结构体。

定义

```
typedef struct {
    hi_u32   stream_que_cnt;
    hi_u64   reserved[4];
 } hi_pngd_chn_attr;
```

成员

成员名称	描述
stream_que_cnt	PNGD图片缓存队列大小。预留参数。
reserved	预留参数，为保证后续版本兼容性，请务必使用memset结构体方式进行清零初始化，在代码中必须避免显式对reserved字段进行访问。

11.18.21 返回码列表

11.18.21.1 公共返回码

错误代码	宏定义	描述
0xA0028001	-	无效的Device ID。
0xA0028002	-	无效的channel ID。

错误代码	宏定义	描述
0xA0028003	HI_ERR_SYS_ILLEGAL_PARAM	参数设置无效
0xA0028004	HI_ERR_SYS_EXIST	通道或资源已存在
0xA0028005	HI_ERR_SYS_UNEXIST	通道或资源不存在
0xA0028006	HI_ERR_SYS_NULL_PTR	空指针错误
0xA0028007	-	使能系统、Device或通道前未配置对应的参数。
0xA0028008	HI_ERR_SYS_NOT_SUPPORT	不支持的功能
0xA0028009	HI_ERR_SYS_NOT_PERM	操作不允许
0xA002800C	HI_ERR_SYS_NO_MEM	分配内存失败，如系统内存不足
0xA002800D	-	分配缓存失败，如申请的数据缓冲区太大。
0xA002800E	-	缓冲区中无数据。
0xA002800F	-	缓冲区中数据满。
0xA0028010	HI_ERR_SYS_NOT_READY	系统控制属性未配置
0xA0028011	-	地址错误。
0xA0028012	HI_ERR_SYS_BUSY	系统忙
0xA0028013	-	缓存小于实际需要的大小。
0xA0028014	-	硬件或软件处理超时。
0xA0028015	HI_ERR_SYS_ERR	内部系统错误。
0xA002803F	-	最大的返回码，该模块的错误码必须小于该值。

11.18.21.2 音频相关返回码

表 11-65 音频基础属性.API 错误码

错误代码	宏定义	描述
0xa0148003	HI_ERR_AIO_ILLEGAL_PARAM	音频基础属性参数设置无效。
0xa0148006	HI_ERR_AIO_NULL_PTR	输入参数空指针错误。
0xa0148009	HI_ERR_AIO_NOT_PERM	操作不允许。

错误代码	宏定义	描述
0xa0148010	HI_ERR_AIO_NOT_READY	音频基础系统未初始化。
0xa0148012	HI_ERR_AIO_BUSY	音频基础系统忙。
0xa014801d	HI_ERR_AIO_REGISTER_ERR	注册失败。

表 11-66 音频输入错误码

错误代码	宏定义	描述
0xa0158001	HI_ERR_AI_INVALID_DEV_ID	音频输入设备号无效。
0xa0158002	HI_ERR_AI_INVALID_CHN_ID	音频输入通道号无效。
0xa0158003	HI_ERR_AI_ILLEGAL_PARAM	音频输入参数设置无效。
0xa0158006	HI_ERR_AI_NULL_PTR	输入参数空指针错误。
0xa0158007	HI_ERR_AI_NOT_CFG	音频输入设备属性未设置。
0xa0158009	HI_ERR_AI_NOT_PERM	操作不允许。
0xa015800c	HI_ERR_AI_NO_MEM	分配内存失败。
0xa015800d	HI_ERR_AI_NO_BUF	音频输入缓存不足。
0xa015800e	HI_ERR_AI_BUF_EMPTY	音频输入缓存为空。
0xa015800f	HI_ERR_AI_BUF_FULL	音频输入缓存为满。
0xa0158010	HI_ERR_AI_NOT_READY	音频输入系统未初始化。
0xa0158012	HI_ERR_AI_BUSY	音频输入系统忙。
0xa0158017	HI_ERR_AI_NOT_ENABLED	音频输入设备或通道没有使能。

表 11-67 音频输出错误码

错误代码	宏定义	描述
0xa0168001	HI_ERR_AO_INVALID_DEV_ID	音频输出设备号无效。
0xa0168002	HI_ERR_AO_INVALID_CHN_ID	音频输出通道号无效。

错误代码	宏定义	描述
0xa0168003	HI_ERR_AO_ILLEGAL_PARAM	音频输出参数设置无效。
0xa0168006	HI_ERR_AO_NULL_PTR	输出空指针错误。
0xa0168007	HI_ERR_AO_NOT_ CFG	音频输出设备属性未设置。
0xa0168009	HI_ERR_AO_NOT_PERM	操作不允许。
0xa016800c	HI_ERR_AO_NO_MEM	统内存不足。
0xa016800d	HI_ERR_AO_NO_BUF	音频输出缓存不足。
0xa016800e	HI_ERR_AO_BUF_EMPTY	音频输出缓存为空。
0xa016800f	HI_ERR_AO_BUF_FULL	音频输出缓存为满。
0xa0168010	HI_ERR_AO_NOT_READY	音频输出系统未初始化。
0xa0168012	HI_ERR_AO_BUSY	音频输出系统忙。
0xa0168017	HI_ERR_AO_NOT_ENABLED	音频输出设备或通道没使能。

表 11-68 音频编码错误码

错误代码	宏定义	描述
0xa0178001	HI_ERR_AENC_INVALID_DEV_ID	音频设备号无效。
0xa0178002	HI_ERR_AENC_INVALID_CHN_ID	音频编码通道号无效。
0xa0178003	HI_ERR_AENC_ILLEGAL_PARAM	音频编码参数设置无效。
0xa0178004	HI_ERR_AENC_EXIST	音频编码通道已经创建。
0xa0178005	HI_ERR_AENC_UNEXIST	音频编码通道未创建。
0xa0178006	HI_ERR_AENC_NULL_PTR	输入参数空指针错误。
0xa0178007	HI_ERR_AENC_NOT_CFG	编码通道未配置。
0xa0178008	HI_ERR_AENC_NOT_SUPPORT	操作不被支持。
0xa0178009	HI_ERR_AENC_NOT_PERM	操作不允许。
0xa017800c	HI_ERR_AENC_NO_MEM	系统内存不足。
0xa017800d	HI_ERR_AENC_NO_BUF	编码通道缓存分配失败。

错误代码	宏定义	描述
0xa017800e	HI_ERR_AENC_BUF_EMPTY	编码通道缓存空。
0xa017800f	HI_ERR_AENC_BUF_FULL	编码通道缓存满。
0xa0178010	HI_ERR_AENC_NOT_READY	系统没有初始化。
0xa0178040	HI_ERR_AENC_ENCODER_ERR	音频编码数据错误。

表 11-69 音频解码错误码

错误代码	宏定义	描述
0xa0188001	HI_ERR_ADEC_INVALID_DEV_ID	音频解码设备号无效。
0xa0188002	HI_ERR_ADEC_INVALID_CHN_ID	音频解码通道号无效。
0xa0188003	HI_ERR_ADEC_ILLEGAL_PARAM	音频解码参数设置无效。
0xa0188004	HI_ERR_ADEC_EXIST	音频解码通道已经创建。
0xa0188005	HI_ERR_ADEC_UNEXIST	音频解码通道未创建。
0xa0188006	HI_ERR_ADEC_NULL_PTR	输入参数空指针错误。
0xa0188007	HI_ERR_ADEC_NOT_CFG	解码通道属性未配置。
0xa0188008	HI_ERR_ADEC_NOT_SUPPORT	操作不被支持。
0xa0188009	HI_ERR_ADEC_NOT_PERM	操作不允许。
0xa018800c	HI_ERR_ADEC_NO_MEM	系统内存不足。
0xa018800d	HI_ERR_ADEC_NO_BUF	解码通道缓存分配失败。
0xa018800e	HI_ERR_ADEC_BUF_EMPTY	解码通道缓存空。
0xa018800f	HI_ERR_ADEC_BUF_FULL	解码通道缓存满。
0xa0188010	HI_ERR_ADEC_NOT_READY	系统没有初始化。
0xa0188040	HI_ERR_ADEC_DECODER_ERR	音频解码数据错误。
0xa0188041	HI_ERR_ADEC_BUF_LACK	解码输入缓存空间不够。

11.18.21.3 ISP 系统控制返回码

表 11-70 ISP API 错误码

错误代码	宏定义	描述
0xA01C8003	HI_ERR_ISP_ILLEGAL_PARAM	输入参数无效。
0xA01C8006	HI_ERR_ISP_NULL_PTR	空指针错误。
0xA01C8008	HI_ERR_ISP_NOT_SUPPORT	当前ISP不支持。
0xA01C800C	HI_ERR_ISP_NO_MEM	内存不足。
0xA01C8040	HI_ERR_ISP_NOT_INIT	ISP没有初始化。
0xA01C8041	HI_ERR_ISP_MEM_NOT_INIT	外部寄存器没有初始化。
0xA01C8042	HI_ERR_ISP_ATTR_NOT_CFG	属性未配置。
0xA01C8043	HI_ERR_ISP_SNS_UNREGISTER	Sensor未注册。
0xA01C8044	HI_ERR_ISP_INVALID_ADDR	无效地址。
0xA01C8046	HI_ERR_ISP_NO_INT	ISP无中断。

11.18.21.4 VI 视频输入返回码

错误代码	宏定义	描述
0xA0108001	HI_ERR_VI_INVALID_DEV_ID	视频输入设备号无效。
0xA0108002	HI_ERR_VI_INVALID_CHN_ID	视频输入通道号无效。
0xA0108003	HI_ERR_VI_ILLEGAL_PARAM	视频输入参数设置无效。
0xA0108004	HI_ERR_VI_PIPE_EXIST	PIPE已存在。
0xA0108005	HI_ERR_VI_PIPE_UNEXIST	PIPE不存在。
0xA0108006	HI_ERR_VI_NULL_PTR	输入参数空指针错误。
0xA0108007	HI_ERR_VI_NOT_CFG	视频设备或通道属性未配置。
0xA0108008	HI_ERR_VI_NOT_SUPPORT	操作不支持。
0xA0108009	HI_ERR_VI_NOT_PERM	操作不允许。
0xA010800A	HI_ERR_VI_INVALID_PIPE_ID	PIPE号无效。
0xA010800C	HI_ERR_VI_NO_MEM	分配内存失败。
0xA010800E	HI_ERR_VI_BUF_EMPTY	视频输入缓存为空。
0xA010800F	HI_ERR_VI_BUF_FULL	视频输入缓存为满。

错误代码	宏定义	描述
0xA0108010	HI_ERR_VI_SYS_NOT_READY	视频输入系统未初始化。
0xA0108012	HI_ERR_VI_BUSY	视频输入系统忙。
0xA0108040	HI_ERR_VI_NOT_ENABLE	视频输入设备或通道未启用。
0xA0108041	HI_ERR_VI_NOT_DISABLE	视频输入设备或通道未禁用。
0xA0108043	HI_ERR_VI_TIME_OUT	视频配置属性超时。
0xA0108047	HI_ERR_VI_NOT_BINDED	视频通道未绑定。
0xA0108048	HI_ERR_VI_BINDED	视频通道已绑定。

11.18.21.5 区域管理返回码

错误代码	宏定义	描述
0xA0038001	HI_ERR_RGN_INVALID_DEV_ID	设备ID超出合法范围。
0xA0038002	HI_ERR_RGN_INVALID_CHN_ID	通道组号错误或无效区域句柄。
0xA0038003	HI_ERR_RGN_ILLEGAL_PARAM	参数超出合法范围。
0xA0038004	HI_ERR_RGN_EXIST	重复创建已存在的设备、通道或资源。
0xA0038005	HI_ERR_RGN_UNEXIST	试图使用或者销毁不存在的设备、通道或者资源。
0xA0038006	HI_ERR_RGN_NULL_PTR	函数参数中有空指针。
0xA0038007	HI_ERR_RGN_NOT_CFG	模块没有配置。
0xA0038008	HI_ERR_RGN_NOT_SUPPORT	不支持的参数或者功能。
0xA0038009	HI_ERR_RGN_NOT_PERM	该操作不允许，如试图修改静态配置参数。
0xA003800C	HI_ERR_RGN_NO_MEM	分配内存失败，如系统内存不足。
0xA003800D	HI_ERR_RGN_NO_BUF	分配缓存失败，如申请的数据缓冲区太大。
0xA003800E	HI_ERR_RGN_BUF_EMPTY	缓冲区中无数据。
0xA003800F	HI_ERR_RGN_BUF_FULL	缓冲区中数据满。
0xA0038010	HI_ERR_RGN_NOT_READY	系统没有初始化或没有加载相应模块。

错误代码	宏定义	描述
0xA0038011	HI_ERR_RGN_BAD_ADDR	地址非法。
0xA0038012	HI_ERR_RGN_BUSY	系统忙。

11.18.21.6 VPSS 视频处理子系统返回码

错误代码	宏定义	描述
0xA03E8001	HI_ERR_VPSS_INVALID_DEV_ID	VPSS组ID无效
0xA03E8002	HI_ERR_VPSS_INVALID_CHN_ID	VPSS通道号无效
0xA03E8003	HI_ERR_VPSS_ILLEGAL_PARAM	VPSS参数设置无效
0xA03E8004	HI_ERR_VPSS_EXIST	VPSS Group已创建
0xA03E8005	HI_ERR_VPSS_UNEXIST	VPSS Group未创建
0xA03E8006	HI_ERR_VPSS_NULL_PTR	输入参数空指针错误
0xA03E8008	HI_ERR_VPSS_NOT_SUPPORT	操作不支持
0xA03E8009	HI_ERR_VPSS_NOT_PERM	操作不允许
0xA03E800C	HI_ERR_VPSS_NO_MEM	分配内存失败
0xA03E800D	HI_ERR_VPSS_NO_BUF	分配BUF池失败
0xA03E800E	HI_ERR_VPSS_BUF_EMPTY	图像队列为空
0xA03E8010	HI_ERR_VPSS_NOT_READY	VPSS系统未初始化
0xA03E8012	HI_ERR_VPSS_BUSY	VPSS系统忙
0xA03E8013	HI_ERR_VPSS_SIZE_NOT_ENOUGH	VB块大小不够

11.18.21.7 VO 视频输出返回码

表 11-71 视频输出错误码

错误码	定义	描述
0xa00f8001	HI_ERR_VO_INVALID_DEV_ID	设备ID超出合法范围
0xa00f8003	HI_ERR_VO_INVALID_CHN_ID	通道ID超出合法范围
0xa00f8004	HI_ERR_VO_INVALID_LAYER_ID	层ID超出合法范围

错误码	定义	描述
0xa00f8007	HI_ERR_VO_ILLEGAL_PARAM	参数超出合法范围
0xa00f800a	HI_ERR_VO_NULL_PTR	函数参数中有空指针
0xa00f800b	HI_ERR_VO_NOT_CFG	未配置
0xa00f800c	HI_ERR_VO_NOT_SUPPORT	不支持的操作
0xa00f800d	HI_ERR_VO_NOT_PERM	操作不允许
0xa00f8010	HI_ERR_VO_NOT_ENABLE	未使能
0xa00f8011	HI_ERR_VO_NOT_DISABLE	未禁用
0xa00f8014	HI_ERR_VO_NO_MEM	分配内存失败
0xa00f8018	HI_ERR_VO_NOT_READY	系统未初始化
0xa00f8020	HI_ERR_VO_TIMEOUT	等待超时
0xa00f8022	HI_ERR_VO_BUSY	资源忙
0xa00f8024	HI_ERR_VO_NOT_BINDED	未被绑定
0xa00f8025	HI_ERR_VO_BINDED	已被绑

11.18.21.8 TDE 图形绘制返回码

表 11-72 图像处理错误码

错误码	定义	描述
0xA0648001	HI_ERR_TDE_DEV_NOT_OPEN	TDE设备未打开
0xA0648002	HI_ERR_TDE_DEV_OPEN_FAILED	TDE设备打开失败
0xA0648003	HI_ERR_TDE_NULL_PTR	输入参数包含空指针
0xA0648004	HI_ERR_TDE_NO_MEM	内存分配失败
0xA0648005	HI_ERR_TDE_INVALID_HANDLE	非法任务句柄
0xA0648006	HI_ERR_TDE_INVALID_PARAM	无效参数设置

错误码	定义	描述
0xA0648007	HI_ERR_TDE_NOT_ALIGNED	参数值没有对齐
0xA0648008	HI_ERR_TDE_MINIFICATION	缩小倍数过大
0xA0648009	HI_ERR_TDE_CLIP_AREA	操作区域与clip区域没有交集，显示不会有更新
0xA0648010	HI_ERR_TDE_JOB_TIMEOUT	阻塞任务等待超时
0xA0648011	HI_ERR_TDE_UNSUPPORTED_OPERATION	不支持的操作
0xA0648012	HI_ERR_TDE_QUERY_TIMEOUT	指定的任务超时未完成
0xA0648013	HI_ERR_TDE_INTERRUPT	阻塞任务被打断

11.18.21.9 HDMI 外设返回码

表 11-73 HDMI 错误码

错误码	定义	描述
0xA0288001	HI_ERR_HDMI_NOT_INIT	HDMI未初始化
0xA0288002	HI_ERR_HDMI_INVALID_PARA	参数非法
0xA0288003	HI_ERR_HDMI_NUL_PTR	空指针
0xA0288004	HI_ERR_HDMI_DEV_NOT_OPEN	HDMI未打开
0xA0288005	HI_ERR_HDMI_DEV_NOT_CONNECT	HDMI设备未连接
0xA0288006	HI_ERR_HDMI_READ_SINK_FAILED	HDMI读取Sink 端失败
0xA0288007	HI_ERR_HDMI_INIT_ALREADY	HDMI已经初始化
0xA0288008	HI_ERR_HDMI_CALLBACK_ALREADY	HDMI回调已注册
0xA0288009	HI_ERR_HDMI_INVALID_CALLBACK	回调函数无效
0xA028800A	HI_ERR_HDMI_FEATURE_NO_USPPORT	功能不支持

错误码	定义	描述
0xA028800B	HI_ERR_HDMI_BUS_BUSY	总线忙
0xA028800C	HI_ERR_HDMI_READ_EVENT_FAILED	读取事件失败
0xA028800D	HI_ERR_HDMI_NOT_START	HDMI未工作
0xA028800E	HI_ERR_HDMI_READ_EDID_FAILED	EDID（Extended Display Identification Data）读取失败
0xA028800F	HI_ERR_HDMI_INIT_FAILED	初始化失败
0xA0288010	HI_ERR_HDMI_CREATE_TESK_FAILED	创建任务失败
0xA0288011	HI_ERR_HDMI_MALLOC_FAILED	分配内存失败
0xA0288012	HI_ERR_HDMI_FREE_FAILED	释放内存失败
0xA0288013	HI_ERR_HDMI_PTHREAD_CREATE_FAILED	创建线程失败
0xA0288014	HI_ERR_HDMI_PTHREAD_JOIN_FAILED	加入线程失败
0xA0288015	HI_ERR_HDMI_STRATEGY_FAILED	工作策略失败
0xA0288016	HI_ERR_HDMI_SET_ATTR_FAILED	设置参数失败
0xA0288017	HI_ERR_HDMI_CALLBACK_NOT_REGISTER	回调未注册
0xA0288018	HI_ERR_HDMI_UNKNOWN_COMMAND	未知命令
0xA0288019	HI_ERR_HDMI_MUTEX_LOCK_FAILED	锁失败

11.18.21.10 VPC 图像处理返回码

错误代码	宏定义	描述
0xA0078001	-	无效的Device ID。
0xA0078002	HI_ERR_VPC_INVALID_CHN_ID	通道 ID 超出合法范围。

错误代码	宏定义	描述
0xA0078003	HI_ERR_VPC_ILLEGAL_PARAM	参数超出合法范围。
0xA0078004	HI_ERR_VPC_EXIST	试图创建已经存在的通道。
0xA0078005	HI_ERR_VPC_UNEXIST	通道未创建或已销毁。
0xA0078006	HI_ERR_VPC_NULL_PTR	函数参数中有空指针。
0xA0078007	HI_ERR_VPC_NOT_CFG	使用前未配置。
0xA0078008	HI_ERR_VPC_NOT_SUPPORT	不支持的参数或者功能。
0xA0078009	HI_ERR_VPC_NOT_PERM	该操作不允许。
0xA007800C	HI_ERR_VPC_NO_MEM	分配内存失败，如系统内存不足。
0xA007800D	HI_ERR_VPC_NO_BUF	分配缓存失败，如申请的数据缓冲区太大。
0xA007800E	HI_ERR_VPC_BUF_EMPTY	缓冲区中无数据。
0xA007800F	HI_ERR_VPC_BUF_FULL	缓冲区中数据满。
0xA0078010	HI_ERR_VPC_SYS_NOT_READY	系统没有初始化或者相关依赖的模块没有加载。
0xA0078011	HI_ERR_VPC_BAD_ADDR	内存地址错误。 媒体数据处理功能涉及的输入、输出内存需调用hi_mpi_dvpp_malloc接口申请，或者内存大小配置错误，请检查代码逻辑。
0xA0078012	HI_ERR_VPC_BUSY	系统忙。
0xA0078013	-	缓存小于实际需要的大小。
0xA0078014	-	硬件或软件处理超时。
0xA0078015	HI_ERR_VPC_SYS_ERROR	内部系统错误。
0xA007803F	-	最大的返回码，该模块的错误码必须小于该值。

11.18.21.11 VDEC 视频解码/JPEGD 图片解码返回码

表11-74为VDEC视频解码和JPEGD图片解码共用的返回码。

表 11-74 共用返回码

错误代码	宏定义	描述
0xA0058001	-	无效的Device ID。
0xA0058002	HI_ERR_VDEC_INVALID_CHN_ID	通道 ID 超出合法范围。
0xA0058003	HI_ERR_VDEC_ILLEGAL_PARAM	参数超出合法范围。
0xA0058004	HI_ERR_VDEC_EXIST	试图创建已经存在的通道。
0xA0058005	HI_ERR_VDEC_UNEXIST	通道未创建或已销毁。
0xA0058006	HI_ERR_VDEC_NULL_PTR	函数参数中有空指针。
0xA0058007	HI_ERR_VDEC_NOT_CFG	使用前未配置。
0xA0058008	HI_ERR_VDEC_NOT_SUPPORT	不支持的参数或者功能。
0xA0058009	HI_ERR_VDEC_NOT_PERM	该操作不允许。
0xA005800C	HI_ERR_VDEC_NO_MEM	分配内存失败，如系统内存不足。
0xA005800D	HI_ERR_VDEC_NO_BUF	分配缓存失败，如申请的数据缓冲区太大。
0xA005800E	HI_ERR_VDEC_BUF_EMPTY	缓冲区中无数据。
0xA005800F	HI_ERR_VDEC_BUF_FULL	缓冲区中数据满。
0xA0058010	HI_ERR_VDEC_SYS_NOT_READY	系统没有初始化或者相关依赖的模块没有加载。
0xA0058011	HI_ERR_VDEC_BAD_ADDR	内存地址错误。 媒体数据处理功能涉及的输入、输出内存需调用hi_mpi_dvpp_malloc接口申请，请检查代码逻辑。
0xA0058012	HI_ERR_VDEC_BUSY	系统忙。
0xA0058013	-	缓存小于实际需要的大小。
0xA0058014	-	硬件或软件处理超时。
0xA0058015	HI_ERR_VDEC_SYS_ERROR	内部系统错误。
0xA005803F	-	最大的返回码，该模块的错误码必须小于该值。

11.18.21.12 VENC 视频编码/JPEGE 图片编码返回码

表11-75为VENC视频编码和JPEGE图片编码共用的返回码。

表 11-75 共用返回码

错误代码	宏定义	描述
0xA0088001	-	无效的Device ID。
0xA0088002	HI_ERR_VENC_INVALID_CHN_ID	通道 ID 超出合法范围。
0xA0088003	HI_ERR_VENC_ILLEGAL_PARAM	参数超出合法范围。
0xA0088004	HI_ERR_VENC_EXIST	试图申请或者创建已经存在的设备、通道或者资源。
0xA0088005	HI_ERR_VENC_UNEXIST	试图使用或者销毁不存在的设备、通道或者资源。
0xA0088006	HI_ERR_VENC_NULL_PTR	函数参数中有空指针。
0xA0088007	HI_ERR_VENC_NOT_CFG	使用前未配置。
0xA0088008	HI_ERR_VENC_NOT_SUPPORT	不支持的参数或者功能。
0xA0088009	HI_ERR_VENC_NOT_PERM	该操作不允许，如试图修改静态配置参数。
0xA008800C	HI_ERR_VENC_NO_MEM	分配内存失败，如系统内存不足。
0xA008800D	HI_ERR_VENC_NO_BUF	分配缓存失败，如申请的数据缓冲区太大、频繁传入输入帧导致缓存区满等。
0xA008800E	HI_ERR_VENC_BUF_EMPTY	缓冲区中无数据。
0xA008800F	HI_ERR_VENC_BUF_FULL	缓冲区中数据满。
0xA0088010	HI_ERR_VENC_SYS_NOT_READY	系统没有初始化或没有加载相应模块。
0xA0088011	HI_ERR_VENC_BAD_ADDR	内存地址错误。 媒体数据处理功能涉及的输入、输出内存需调用hi_mpi_dvpp_malloc接口申请，请检查代码逻辑。
0xA0088012	HI_ERR_VENC_BUSY	系统忙。
0xA0088013	-	缓存小于实际需要的大小。

错误代码	宏定义	描述
0xA0088014	-	硬件或软件处理超时。
0xA0088015	HI_ERR_VENC_SYS_ERROR	内部系统错误。
0xA008803F	-	最大的返回码，该模块的错误码必须小于该值。

11.18.21.13 PNGD 图像解码返回码

错误代码	宏定义	描述
0xA0408001	-	无效的Device ID。
0xA0408002	HI_ERR_PNGD_INVALID_CHN_ID	通道 ID 超出合法范围。
0xA0408003	HI_ERR_PNGD_ILLEGAL_PARAM	参数超出合法范围。
0xA0408004	HI_ERR_PNGD_EXIST	试图创建已经存在的通道。
0xA0408005	HI_ERR_PNGD_UNEXIST	通道未创建或已销毁。
0xA0408006	HI_ERR_PNGD_NULL_PTR	函数参数中有空指针。
0xA0408007	HI_ERR_PNGD_NOT_CFG	使用前未配置。
0xA0408008	HI_ERR_PNGD_NOT_SUPPORT	不支持的参数或者功能。
0xA0408009	HI_ERR_PNGD_NOT_PERM	该操作不允许。
0xA040800C	HI_ERR_PNGD_NO_MEM	分配内存失败，如系统内存不足。
0xA040800D	HI_ERR_PNGD_NO_BUF	分配缓存失败，如申请的数据缓冲区太大。
0xA040800E	HI_ERR_PNGD_BUF_EMPTY	缓冲区中无数据。
0xA040800F	HI_ERR_PNGD_BUF_FULL	缓冲区中数据满。
0xA0408010	HI_ERR_PNGD_SYS_NOT_READY	系统没有初始化或者相关依赖的模块没有加载。
0xA0408011	HI_ERR_PNGD_BAD_ADDR	内存地址错误。 媒体数据处理功能涉及的输入、输出内存需调用hi_mpi_dvpp_malloc接口申请，请检查代码逻辑。
0xA0408012	HI_ERR_PNGD_BUSY	系统忙。
0xA0408013	-	缓存小于实际需要的大小。

错误代码	宏定义	描述
0xA0408014	HI_ERR_PNGD_TIMEOUT	硬件或软件处理超时。
0xA0408015	HI_ERR_PNGD_SYS_ERROR	内部系统错误。
0xA040803F	-	最大的返回码，该模块的错误码必须小于该值。

11.19 异常处理

11.19.1 aclrtSetExceptionInfoCallback

函数功能

设置异常回调函数。

约束说明

- 您需要在执行异步任务之前，设置异常回调函数，当Device上的任务执行异常时，系统会向用户设置的异常回调函数中传入一个包含任务ID、Stream ID、线程ID、Device ID以及错误码的aclrtExceptionInfo结构体指针，并执行回调函数，用户可以再分别调用**aclrtGetTaskIdFromExceptionInfo**、**aclrtGetStreamIdFromExceptionInfo**、**aclrtGetThreadIdFromExceptionInfo**、**aclrtGetDeviceIdFromExceptionInfo**、**aclrtGetErrorCodeFromExceptionInfo**接口获取产生异常的任务ID、Stream ID、线程ID、Device ID以及错误码，便于定位问题。

  使用场景举例：例如，在调用**aclopExecuteV2**接口前，调用aclrtSetExceptionInfoCallback接口设置异常回调函数，当算子在Device执行异常时，系统会向用户设置的异常回调函数中传入一个包含任务ID、Stream ID、线程ID、Device ID以及错误码的aclrtExceptionInfo结构体指针，并执行回调函数。
- 如果多次设置异常回调函数，以最后一次设置为准。
- 如果想清空回调函数，可调用aclrtSetExceptionInfoCallback接口，将入参设置为空指针。

函数原型

aclError aclrtSetExceptionInfoCallback(aclrtExceptionInfoCallback callback)

参数说明

参数名	输入/输出	说明
callback	输入	指定要注册的回调函数。 回调函数的函数原型为： typedef void (*aclrtExceptionInfoCallback) (aclrtExceptionInfo *exceptionInfo);

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见**7.11 AI Core异常信息获取**。

11.19.2 aclrtGetTaskIdFromExceptionInfo

函数功能

获取异常信息中的任务ID。该接口与**aclrtSetExceptionInfoCallback**接口配合使用。

函数原型

uint32_t aclrtGetTaskIdFromExceptionInfo(const aclrtExceptionInfo *info)

参数说明

参数名	输入/输出	说明
info	输入	异常信息的指针。 在执行任务之前调用**aclrtSetExceptionInfoCallback**接口，系统会将产生异常的任务ID、Stream ID、线程ID、Device ID存放在aclExceptionInfo结构体中。

返回值说明

返回异常信息中的任务ID，返回值为0xFFFFFFFF（以十六进制为例）时表示Device异常。

参考资源

接口调用示例，参见**7.11 AI Core异常信息获取**。

11.19.3 aclrtGetStreamIdFromExceptionInfo

函数功能

获取异常信息中的Stream ID。该接口与**aclrtSetExceptionInfoCallback**接口配合使用。

函数原型

uint32_t aclrtGetStreamIdFromExceptionInfo(const aclrtExceptionInfo *info)

参数说明

参数名	输入/输出	说明
info	输入	异常信息的指针。 在执行任务之前调用**aclrtSetExceptionInfoCallback**接口，系统会将产生异常的任务ID、Stream ID、线程ID、Device ID存放在aclExceptionInfo结构体中。

返回值说明

返回异常信息中的Stream ID，返回值为0xFFFFFFFF（以十六进制为例）时表示Device异常。

参考资源

接口调用示例，参见**7.11 AI Core异常信息获取**。

11.19.4 aclrtGetThreadIdFromExceptionInfo

函数功能

获取异常信息中的线程ID。该接口与**aclrtSetExceptionInfoCallback**接口配合使用。

函数原型

uint32_t aclrtGetThreadIdFromExceptionInfo(const aclrtExceptionInfo *info)

参数说明

参数名	输入/输出	说明
info	输入	异常信息的指针。 在执行任务之前调用**aclrtSetExceptionInfoCallback**接口，系统会将产生异常的任务ID、Stream ID、线程ID、Device ID存放在aclExceptionInfo结构体中。

返回值说明

返回异常信息中的线程ID，返回值为0xFFFFFFFF（以十六进制为例）时表示Device异常。

11.19.5 aclrtGetDeviceIdFromExceptionInfo

函数功能

获取异常信息中的Device ID。该接口与**aclrtSetExceptionInfoCallback**接口配合使用。

函数原型

uint32_t aclrtGetDeviceIdFromExceptionInfo(const aclrtExceptionInfo *info)

参数说明

参数名	输入/输出	说明
info	输入	异常信息的指针。 在执行任务之前调用**aclrtSetExceptionInfoCallback**接口，系统会将产生异常的任务ID、Stream ID、线程ID、Device ID存放在aclExceptionInfo结构体中。

返回值说明

返回异常信息中的Device ID，返回值为0xFFFFFFFF（以十六进制为例）时表示Device异常。

参考资源

接口调用示例，参见**7.11 AI Core异常信息获取**。

11.19.6 aclrtGetErrorCodeFromExceptionInfo

函数功能

获取异常信息中的错误码。该接口与**aclrtSetExceptionInfoCallback**接口配合使用。

函数原型

uint32_t aclrtGetErrorCodeFromExceptionInfo(const aclrtExceptionInfo *info)

参数说明

参数名	输入/输出	说明
info	输入	异常信息的指针。 在执行任务之前调用**aclrtSetExceptionInfoCallback**接口，系统会将产生异常的任务ID、Stream ID、线程ID、Device ID、错误码存放在aclExceptionInfo结构体中。

返回值说明

返回异常信息中的错误码，返回值为0xFFFFFFFF（以十六进制为例）时表示Device异常。

11.20 日志管理

11.20.1 aclAppLog

函数功能

将日志记录到日志文件中。

AscendCL提供ACL_APP_LOG宏，封装了aclAppLog接口，推荐用户调用ACL_APP_LOG宏，传入日志级别、日志描述、fmt中的可变参数。日志文件的详细说明，请参见《**日志参考（开放态）**》。

```
#define ACL_APP_LOG(level, fmt, ...) \
    aclAppLog(level, __FUNCTION__, __FILE__, __LINE__, fmt, ##__VA_ARGS__)
```

函数原型

void aclAppLog(aclLogLevel logLevel, const char *func, const char *file, uint32_t line, const char *fmt, ...)

参数说明

参数名	输入/输出	说明
logLevel	输入	日志级别。 typedef enum { ACL_DEBUG = 0, ACL_INFO = 1, ACL_WARNING = 2, ACL_ERROR = 3, } aclLogLevel;
func	输入	表示用户在哪个接口中调用aclAppLog接口，固定配置为__FUNCTION__
file	输入	表示用户在哪个文件中调用aclAppLog接口，固定配置为__FILE__
line	输入	表示用户在哪一行中调用aclAppLog接口，固定配置为__LINE__
fmt	输入	日志描述。 在调用格式化函数时，fmt中参数的类型、个数必须与实际参数类型、个数保持一致。
...	输入	fmt中的可变参数，根据日志内容添加。

返回值说明

无

调用示例

```
//若fmt中存在可变参数，需提前定义
uint32_t modelId = 1;
ACL_APP_LOG(ACL_INFO, "load model success, modelId is %u", modelId);
```

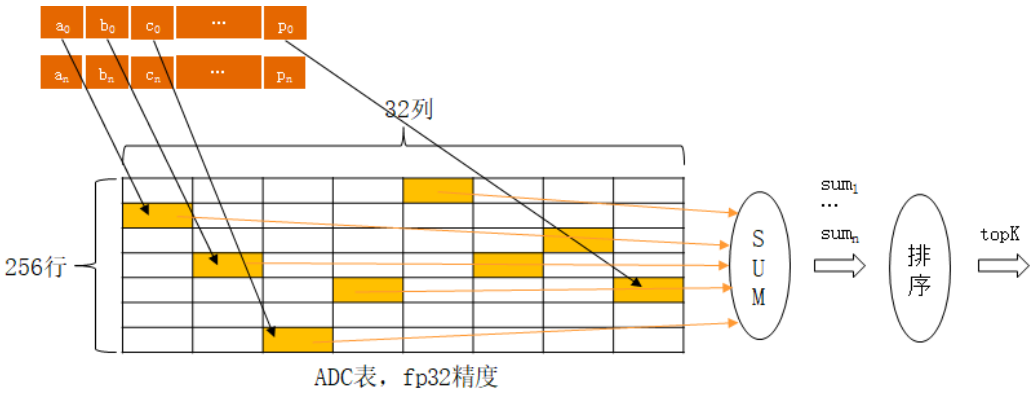
11.21 特征向量检索

11.21.1 总体说明

在图像检索场景，通常使用深度学习算法处理感兴趣的图像时，提取出高维特征向量；并将一组感兴趣的图像提取得到的特征向量作为底库，待搜索的特征向量与底库中的向量依次进行相似度计算，按相似度排序返回检索结果。为了进一步降低存储和计算的开销，对浮点型长特征向量进行量化处理、转换为短特征再进行检索比对的技术被广泛应用。

昇腾AI处理器支持基于硬件加速的PQ短特征检索，提供特征底库的添加/删除接口，特征向量的修改/删除接口、以及执行检索接口；支持长度为32 Byte的短特征向量，支持设置返回TOP结果的数量，支持1:N和N:M检索模式。用户需要自行准备短特征底库，以及生成待检索向量的ADC表，流程如下：

检索比对过程示意图如下所示：



ADC表是大小为32KByte的查找表，使用时需要将表的内容以无符号字符的格式传入对应接口。

Atlas 推理系列产品的特征检索规格如下表所示：

表 11-76 规格说明

类型	规格说明
特征长度	32字节（因为数据结构管理开销，每条特征的平均内存开销约为36字节）
底库容量	最大5亿条（实际值取决于板卡的内存容量）
检索比对类型	支持1:N检索，支持N:M检索
底库数量	1:N底库最大100万个 N:M底库最大1个
底库容量	1:N每个特征库最大100万条特征 N:M底库最大1000万条特征
检索比对性能	每秒50亿次； 5亿底库下可达到10QPS，时延<1秒
检索返回候选特征数量	2~4800

11.21.2 aclfvInit

函数功能

初始化特征检索模块。

与**aclfvRelease**接口配套使用。

Atlas 200/300/500 推理产品不支持该接口。

Atlas 推理系列加速模块产品不支持该接口。

约束说明

一个进程内只能调用一次aclfvInit接口。

函数原型

aclError aclfvInit(aclfvInitPara *initPara)

参数说明

参数名	输入/输出	说明
initPara	输入	特征向量检索初始化参数的指针。 ● 调用**aclfvCreateInitPara**接口创建**aclfvInitPara**类型的数据。 ● 1:N场景下调用**aclfvSet1NTopNum**接口设置返回的topK结果数量的最大值，不设置时，默认返回的topK结果数量的最大值为4800。 ● N:M场景下调用**aclfvSetNMTopNum**接口设置返回的topK结果数量的最大值，不设置时，默认返回的topK结果数量的最大值为500。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例代码，参见**7.14 特征向量检索**。

11.21.3 aclfvRelease

函数功能

特征检索模块去初始化，释放内存空间，与**aclfvInit**接口配套使用。

Atlas 200/300/500 推理产品不支持该接口。

Atlas 推理系列加速模块产品不支持该接口。

函数原型

aclError aclfvRelease()

参数说明

无

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例代码，参见7.14 特征向量检索。

11.21.4 aclfvRepoAdd

函数功能

添加底库或向已存在底库中添加特征。该章节描述的1:N表示“检索请求的个数:底库的数量”，N:M用于碰撞两个库的相似性。

Atlas 200/300/500 推理产品不支持该接口。

Atlas 推理系列加速模块产品不支持该接口。

约束说明

- 1:N添加时，对于每个库，用户需要保证aclfvFeatureInfo结构体中的offset连续；N:M添加时，offset不为0会报错。
- 非线程安全，N:M场景不允许多线程同时添加\检索\删除。
- 调用本接口，内部会默认创建一个Stream，用于接口内任务同步执行，在接口执行完成后，该Stream资源会被自动释放。

  受硬件限制，Stream总数有限，请参见11.8.1 aclrtCreateStream处的说明。

函数原型

aclError aclfvRepoAdd(aclfvSearchType type, aclfvFeatureInfo *featureInfo)

参数说明

参数名	输入/输出	说明
type	输入	检索类型。
featureInfo	输入	特征描述信息的指针。 需提前调用aclfvCreateFeatureInfo接口创建aclfvFeatureInfo类型的数据。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例代码，参见7.14 特征向量检索。

11.21.5 aclfvRepoDel

函数功能

删除底库。

Atlas 200/300/500 推理产品不支持该接口。

Atlas 推理系列加速模块产品不支持该接口。

约束说明

- 非线程安全，N:M场景不允许多线程同时添加\检索\删除。
- 调用本接口，内部会默认创建一个Stream，用于接口内任务同步执行，在接口执行完成后，该Stream资源会被自动释放。

  受硬件限制，Stream总数有限，请参见**11.8.1 aclrtCreateStream**处的说明。

函数原型

aclError aclfvRepoDel(aclfvSearchType type, aclfvRepoRange *repoRange)

参数说明

参数名	输入/输出	说明
type	输入	检索类型。
repoRange	输入	特征删除范围的指针。 需提前调用**aclfvCreateRepoRange**接口创建**aclfvRepoRange**类型的数据。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例代码，参见**7.14 特征向量检索**。

11.21.6 aclfvDel

函数功能

精确删除底库中某个特征，一次只能删除底库中一个特征。N:M场景下不涉及这个接口。

Atlas 200/300/500 推理产品不支持该接口。

Atlas 推理系列加速模块产品不支持该接口。

约束说明

- 调用本接口，内部会默认创建一个Stream，用于接口内任务同步执行，在接口执行完成后，该Stream资源会被自动释放。

  受硬件限制，Stream总数有限，请参见**11.8.1 aclrtCreateStream**处的说明。

函数原型

aclError aclfvDel(aclfvFeatureInfo *featureInfo)

参数说明

参数名	输入/输出	说明
featureInfo	输入	特征描述信息的指针。 需提前调用**aclfvCreateFeatureInfo**接口创建**aclfvFeatureInfo**类型的数据。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例代码，参见**7.14 特征向量检索**。

11.21.7 aclfvModify

函数功能

修改底库中某个特征，一次只能修改底库中一个特征，不允许修改底库中不存在的特征。N:M场景下不涉及这个接口。

Atlas 200/300/500 推理产品不支持该接口。

Atlas 推理系列加速模块产品不支持该接口。

约束说明

- 调用本接口，内部会默认创建一个Stream，用于接口内任务同步执行，在接口执行完成后，该Stream资源会被自动释放。

  受硬件限制，Stream总数有限，请参见**11.8.1 aclrtCreateStream**处的说明。

函数原型

aclError aclfvModify(aclfvFeatureInfo *featureInfo)

参数说明

参数名	输入/输出	说明
featureInfo	输入	特征描述信息的指针。 需提前调用aclfvCreateFeatureInfo接口创建aclfvFeatureInfo类型的数据。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例代码，参见7.14 特征向量检索。

11.21.8 aclfvSearch

函数功能

特征1:N或N:M检索。

Atlas 200/300/500 推理产品不支持该接口。

Atlas 推理系列加速模块产品不支持该接口。

约束说明

- 非线程安全，N:M场景不允许多线程同时添加\检索\删除。
- N:M场景或1:N场景，不添加底库，调用aclfvSearch接口检索时，返回结果数量为0。
- 调用本接口，内部会默认创建一个Stream，用于接口内任务同步执行，在接口执行完成后，该Stream资源会被自动释放。

  受硬件限制，Stream总数有限，请参见11.8.1 aclrtCreateStream处的说明。

函数原型

aclError aclfvSearch(aclfvSearchType type, aclfvSearchInput *searchInput, aclfvSearchResult *searchRst)

参数说明

参数名	输入/输出	说明
type	输入	特征检索场景类型。

参数名	输入/输出	说明
searchInput	输入	检索输入信息的指针。 需提前调用**aclfvCreateSearchInput**接口创建**aclfvSearchInput**类型的数据。
searchRst	输出	检索输出结果的指针。 需提前调用**aclfvCreateSearchResult**接口创建**aclfvSearchResult**类型的数据，该数据中queryCnt参数应与检索输入信息searchInput中的queryCnt参数相同。 检索输出返回的结果数量通过**aclfvCreateSearchInput**接口的topk参数设置，但与调用**aclfvInit**接口进行初始化时设置的topK结果数量最大值有关，若调用**aclfvCreateSearchInput**接口设置的topk参数值大于或等于初始化时设置的topK结果数量最大值，则在返回检索输出结果时，按初始化时设置的topK结果数量最大值返回检索结果；若调用**aclfvCreateSearchInput**接口设置的topk参数值小于初始化时设置的topK结果数量最大值，则在返回检索输出结果时，按实际通过**aclfvCreateSearchInput**接口设置的topk参数值返回检索结果。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例代码，参见**7.14 特征向量检索**。

11.22 Profiling 数据采集

11.22.1 Profiling AscendCL API（通过 Profiling AscendCL API 采集并落盘性能数据）

11.22.1.1 功能介绍

功能

该章节描述通过Profiling AscendCL API采集并落盘性能数据，实现将采集到的Profiling数据写入文件，再使用Profiling工具解析该文件（请参见《**CANN 开发工具指南（开放态）**》中的“性能分析工具”章节下的“高级功能>数据解析与导出”），并展示性能分析数据。

包括以下两种接口调用方式：

- **aclprofInit**接口、**aclprofStart**接口、**aclprofStop**接口、**aclprofFinalize**接口配合使用，实现该方式的性能数据采集。该方式可获取AscendCL的接口性能数据、AI Core上算子的执行时间、AI Core性能指标数据等。目前这些接口为进程级控制，表示在进程内任意线程调用该接口，其它线程都会生效。

  一个进程内，可以根据需求多次调用这些接口，基于不同的Profiling采集配置，采集数据。

- 调用**aclInit**接口，在AscendCL初始化阶段，通过*.json 文件传入要采集的Profiling数据。该方式可获取AscendCL的接口性能数据、AI Core上算子的执行时间、AI Core性能指标数据等。

  一个进程内，只能调用一次**aclInit**接口，如果要修改Profiling采集配置，需修改*.json文件中的配置。详细使用说明请参见**aclInit**接口处的说明，不在本章节描述。

总体约束

不能与**11.22.3 Profiling AscendCL API for Subscription（订阅算子信息的Profiling AscendCL API）**的接口交叉调用：**aclprofInit**接口和**aclprofFinalize**接口之间不能调用**aclprofModelSubscribe**接口、aclprofGet*接口、**aclprofModelUnSubscribe**接口。

接口约束说明

- **调用接口要求**：
  - **aclprofInit**接口必须在aclInit接口之后、模型加载之前调用。

    如果已经通过**aclInit**接口配置了Profiling信息，则调用**aclprofInit**接口、**aclprofStart**接口、**aclprofStop**接口、**aclprofFinalize**时，会返回报错。

    如果没有调用**aclprofInit**接口，调用**aclprofStart**接口、**aclprofStop**接口、**aclprofFinalize**时，会返回报错。
  - **aclprofStart**接口在模型执行之前调用，若在模型执行过程中调用**aclprofStart**接口，Profling采集到的数据为调用**aclprofStart**接口之后的数据，可能导致数据不完整。

    调用**aclprofStart**接口时，可以指定从一个Device上采集性能数据，也可以指定从多个Device上采集性能数据。

    一个用户APP进程内，如果连续调用多次**aclprofStart**接口，指定重复的Profiling配置，或指定的Device重复，会返回报错。
  - 在用户APP的进程生命周期内，**aclprofInit**接口与**aclprofFinalize**接口配对使用，建议只调用一次，如该组合多次调用可以改变保存性能数据的文件的路径。
  - **aclprofStart**接口与**aclprofStop**接口需配对使用。
  - aclprofSetConfig接口必须在aclprofStart接口之前调用。一个APP进程内，可以根据需要选择一次或多次调用aclprofSetConfig接口。
  - 调用aclFinalize并接收到正常退出码后为执行完毕，其他情况为非正常。由于性能数据采集不支持多进程并发执行，为确保驱动关闭正常，需要在前一个性能数据采集用例完全执行完毕之后再执行下一轮采集。建议在aclFinalize接口返回值上加入异常处理操作，方便展示执行状态与问题定位。
- **接口调用顺序**：
  - **建议的接口调用顺序如下**，以“一个用户APP进程内采集多个模型推理时的性能数据”为例：

aclInit接口-->aclprofInit接口-->aclprofStart接口(指定Device 0和Device 1)-->模型1加载-->模型1执行-->aclprofStop接口(与aclprofStart接口的aclprofConfig数据保持一致)-->aclprofStart接口(指定Device 1和Device 2)-->模型2加载-->模型2执行-->aclprofStop接口(与aclprofStart接口的aclprofConfig数据保持一致)-->aclprofFinalize接口-->执行其它任务-->模型卸载-->aclFinalize接口

- **错误的接口调用顺序示例如下**，以“一个用户APP进程内，如果连续调用多次aclprofStart接口，指定的Device重复”为例：

  aclInit接口-->aclprofInit接口-->aclprofStart接口(指定Device 0和**Device 1**)-->aclprofStart接口(指定**Device 1**和Device 2)-->模型1加载-->模型1执行-->模型2加载-->模型2执行-->aclprofStop接口-->aclprofStop接口-->aclprofFinalize-->执行其它任务-->模型卸载-->aclFinalize接口

11.22.1.2 aclprofInit

函数功能

初始化Profiling，目前用于设置保存性能数据的文件的路径。

约束说明

与aclprofFinalize接口配对使用，先调用aclprofInit接口再调用aclprofFinalize接口。

函数原型

aclError aclprofInit(const char *profilerResultPath, size_t length)

参数说明

参数名	输入/输出	说明
profilerResultPath	输入	指定保存性能数据的文件的路径，此路径为绝对路径。
length	输入	profilerResultPath的长度，单位为Byte，最大长度不超过4096字节。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见7.12 Profiling性能数据采集。

11.22.1.3 aclprofSetConfig

函数功能

aclprofCreateConfig接口的扩展接口，用于设置性能数据采集参数。

该接口支持多次调用，用户需要保证数据的一致性和准确性。

约束说明

先调用aclprofSetConfig接口再调用**aclprofStart**接口，可根据需求选择调用该接口。

函数原型

aclError aclprofSetConfig(aclprofConfigType configType, const char *config, size_t configLength)

参数说明

参数名	输入/输出	说明
configType	输入	作为configType参数值。每个枚举表示不同采集配置，若要使用该接口下不同的选项采集多种性能数据，则需要多次调用该接口，详细说明如下： ● ACL_PROF_STORAGE_LIMIT ：指定落盘目录允许存放的最大文件容量，有效取值范围为[200, 4294967295]，单位为MB。 ● ACL_PROF_SYS_HARDWARE_MEM_FREQ：片上内存的带宽及内存信息采集频率、LLC的读写带宽数据采集频率、组件内存采集频率，范围[1,100]。 不同产品支持情况不同，请以实际实现为准。 ● ACL_PROF_LLC_MODE：LLC Profiling采集事件。要求同时设置ACL_PROF_SYS_HARDWARE_MEM_FREQ。可以设置为： – Atlas 200/300/500 推理产品： – capacity：采集AI CPU和Control CPU的LLC capacity数据。 – bandwidth：采集LLC bandwidth。默认值为capacity。 – Atlas 推理系列产品： – read：读事件，三级缓存读速率。 – write：写事件，三级缓存写速率。默认为read。 ● ACL_PROF_SYS_IO_FREQ：NIC（Atlas 200/300/500 推理产品）采集频率，范围[1,100]。 ● ACL_PROF_SYS_INTERCONNECTION_FREQ：PCIe数据（Atlas 推理系列产品）采集频率，范围[1,50]。 ● ACL_PROF_DVPP_FREQ：DVPP采集频率，范围[1,100]。
config	输入	指定配置项参数值。
configLength	输入	config的长度，单位为Byte，最大长度不超过256字节。

返回值说明

返回0表示成功，返回其它值表示失败。

11.22.1.4 aclprofStart

函数功能

下发Profiling请求，使能对应数据的采集。

用户可根据需要，在模型执行过程中按需调用aclprofStart接口，Profling采集到的数据为调用该接口之后的数据。

约束说明

与**aclprofStop**接口配对使用，先调用aclprofStart接口再调用aclprofStop接口。

函数原型

aclError aclprofStart(const aclprofConfig *profilerConfig)

参数说明

参数名	输入/输出	说明
profilerConfig	输入	指定Profiling配置数据。 需提前调用**11.27.87.1 aclprofCreateConfig**接口创建aclprofConfig类型的数据。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见**7.12 Profiling性能数据采集**。

11.22.1.5 aclprofStop

函数功能

停止Profiling数据采集。

约束说明

与**aclprofStart**接口配对使用，先调用aclprofStart接口再调用aclprofStop接口。

函数原型

aclError aclprofStop(const aclprofConfig *profilerConfig)

参数说明

参数名	输入/输出	说明
profilerConfig	输入	指定停止Profiling数据采集的配置。 与**aclprofStart**接口中的**aclprofConfig**类型数据保持一致。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见**7.12 Profiling性能数据采集**。

11.22.1.6 aclprofFinalize

函数功能

结束Profiling。

约束说明

与**aclprofInit**接口配对使用，先调用aclprofInit接口再调用aclprofFinalize接口。

函数原型

aclError aclprofFinalize()

参数说明

无

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见**7.12 Profiling性能数据采集**。

11.22.2 Profiling AscendCL API for Extension（Profiling AscendCL API 扩展接口）

11.22.2.1 功能介绍

功能

当用户需要定位应用程序或上层框架程序的性能瓶颈时，可在Profiling采集进程内（**aclprofStart**接口、**aclprofStop**接口之间）调用Profiling AscendCL API扩展接口（统称为msproftx功能），开启记录应用程序执行期间特定事件发生的时间跨度，并将数据写入Profiling数据文件，再使用Profiling工具解析该文件，并导出展示性能分析数据。

Profiling工具解析导出操作请参见《**CANN 开发工具指南（开放态）**》中的“性能分析工具”章节下的“高级功能>数据解析与导出”。

接口调用方式：在**aclprofStart**和**aclprofStop**接口之间调用**aclprofCreateStamp**、**11.22.2.3 aclprofSetStampTraceMessage**、**11.22.2.4 aclprofMark**、**aclprofPush**、**aclprofPop**、**aclprofRangeStart**、**aclprofRangeStop**、**aclprofDestroyStamp**接口。该方式可获取应用程序执行期间特定时间发生的事件并记录事件发生的时间跨度。

一个进程内，可以根据需求多次调用这些接口。

接口约束说明

- **调用接口要求**：msproftx功能相关接口须在**aclprofStart**接口与**aclprofStop**接口之间调用。其中配对使用的接口有：**aclprofCreateStamp**/**aclprofDestroyStamp**、**aclprofPush**/**aclprofPop**、**aclprofRangeStart**/**aclprofRangeStop**。
- **接口调用顺序**：**aclprofStart接口**(指定Device 0和Device 1)-->**aclprofCreateStamp**接口-->**aclprofSetStampTraceMessage**接口-->**aclprofMark**接口-->(**aclprofPush**接口-->**aclprofPop**接口)或(**aclprofRangeStart**接口-->**aclprofRangeStop**接口)-->**aclprofDestroyStamp**接口-->**aclprofStop接口**(与**aclprofStart**接口的**aclprofConfig**数据保持一致)。

11.22.2.2 aclprofCreateStamp

函数功能

创建msproftx事件标记。后续调用**aclprofMark**、**aclprofSetStampTraceMessage**、**aclprofPush**和**aclprofRangeStart**接口时需要以描述该事件的指针作为输入，表示记录该事件发生的时间跨度。

约束说明

与**11.22.2.9 aclprofDestroyStamp**接口配对使用，需提前调用**11.22.1.4 aclprofStart**接口。

函数原型

void *aclprofCreateStamp(void)

返回值说明

- 返回void类型的指针，表示成功。

- 返回nullptr，表示失败。

参考资源

接口调用示例，参见7.12 Profiling性能数据采集。

11.22.2.3 aclprofSetStampTraceMessage

函数功能

为msproftx事件标记携带字符串描述，在Profiling解析并导出结果中msprof_tx summary数据展示。

Profiling解析并导出结果请参见《CANN 开发工具指南（开放态）》中的“性能分析工具”章节下的“Profiling数据说明>msproftx数据说明”。

约束说明

在11.22.2.2 aclprofCreateStamp接口和11.22.2.9 aclprofDestroyStamp接口之间调用。

函数原型

aclError aclprofSetStampTraceMessage(void *stamp, const char *msg, uint32_t msgLen)

参数说明

参数名	输入/输出	说明
stamp	输入	Stamp指针，指代msproftx事件标记。 指定aclprofCreateStamp接口的指针。
msg	输入	字符串内容。
msgLen	输入	字符串长度。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见7.12 Profiling性能数据采集。

11.22.2.4 aclprofMark

函数功能

msproftx标记瞬时事件。

调用此接口后，Profiling自动在Stamp指针中加上当前时间戳，将Event type设置为Mark，表示开始一次msproftx采集。

约束说明

在11.22.2.2 aclprofCreateStamp接口和11.22.2.9 aclprofDestroyStamp接口之间调用。

函数原型

aclError aclprofMark(void *stamp)

参数说明

参数名	输入/输出	说明
stamp	输入	Stamp指针，指代msproftx事件标记。 指定aclprofCreateStamp接口的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见7.12 Profiling性能数据采集。

11.22.2.5 aclprofPush

函数功能

msproftx用于记录事件发生的时间跨度的开始时间。

调用此接口后，Profiling自动在Stamp指针中记录开始的时间戳，将Event type设置为Push/Pop。

约束说明

- 与11.22.2.6 aclprofPop接口成对使用，表示时间跨度的开始和结束。
- 在11.22.2.2 aclprofCreateStamp接口和11.22.2.9 aclprofDestroyStamp接口之间调用。
- 不能跨线程调用，若需要跨线程可使用11.22.2.7 aclprofRangeStart/11.22.2.8 aclprofRangeStop接口。

函数原型

aclError aclprofPush(void *stamp)

参数说明

参数名	输入/输出	说明
stamp	输入	Stamp指针，指代msproftx事件标记。 指定aclprofCreateStamp接口的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见7.12 Profiling性能数据采集。

11.22.2.6 aclprofPop

函数功能

msproftx用于记录事件发生的时间跨度的结束时间。

调用此接口后，Profiling自动在Stamp指针中记录采集结束的时间戳。

约束说明

- 与11.22.2.5 aclprofPush接口成对使用，表示时间跨度的开始和结束。
- 在11.22.2.2 aclprofCreateStamp接口和11.22.2.9 aclprofDestroyStamp接口之间调用。
- 不能跨线程调用。若需要跨线程可使用11.22.2.7 aclprofRangeStart/11.22.2.8 aclprofRangeStop接口。

函数原型

aclError aclprofPop()

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见7.12 Profiling性能数据采集。

11.22.2.7 aclprofRangeStart

函数功能

msproftx用于记录事件发生的时间跨度的开始时间。

调用此接口后，Profiling自动在Stamp指针记录采集开始的时间戳，将Event type设置为Start/Stop，生成一个进程唯一的id，并将Stamp保存在以进程粒度维护的一个map中。

约束说明

- 与**11.22.2.8 aclprofRangeStop**接口成对使用，表示时间跨度的开始和结束。
- 在**11.22.2.2 aclprofCreateStamp**接口和**11.22.2.9 aclprofDestroyStamp**接口之间调用。
- 可以跨线程调用。

函数原型

aclError aclprofRangeStart(void *stamp, uint32_t *rangeId)

参数说明

参数名	输入/输出	说明
stamp	输入	Stamp指针，指代msproftx事件标记。 指定**aclprofCreateStamp**接口的指针。
rangeId	输出	msproftx事件标记的唯一标识。用于在跨线程时区分。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见**7.12 Profiling性能数据采集**。

11.22.2.8 aclprofRangeStop

函数功能

msproftx用于记录事件发生的时间跨度的结束时间。

调用此接口后，Profiling自动在Stamp指针中记录采集结束的时间戳。

约束说明

- 与**11.22.2.7 aclprofRangeStart**接口成对使用，表示时间跨度的开始和结束。
- 在**11.22.2.2 aclprofCreateStamp**接口和**11.22.2.9 aclprofDestroyStamp**接口之间调用。
- 可以跨线程调用。

函数原型

aclError aclprofRangeStop(uint32_t rangeId)

参数说明

参数名	输入/输出	说明
rangeId	输出	msproftx事件标记的唯一标识。用于在跨线程时区分。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见**7.12 Profiling性能数据采集**。

11.22.2.9 aclprofDestroyStamp

函数功能

释放msproftx事件标记。

约束说明

与**11.22.2.2 aclprofCreateStamp**接口配对使用，在**11.22.1.5 aclprofStop**接口前调用。

函数原型

void aclprofDestroyStamp(void *stamp)

参数说明

参数名	输入/输出	说明
stamp	输入	Stamp指针，指代msproftx事件标记。 指定**aclprofCreateStamp**接口的指针。

参考资源

接口调用示例，参见**7.12 Profiling性能数据采集**。

11.22.3 Profiling AscendCL API for Subscription（订阅算子信息的 Profiling AscendCL API）

11.22.3.1 功能介绍

功能

该章节描述使用订阅算子信息的Profiling AscendCL API采集性能数据，实现将采集到的Profiling数据解析后写入管道，由用户读入内存，再由用户调用AscendCL的接口获取性能数据。

接口调用方式：**aclprofModelSubscribe**接口、aclprofGet*接口、**aclprofModelUnSubscribe**接口配合使用，实现该方式的性能数据采集，当前支持获取网络模型中算子的性能数据，包括算子名称、算子类型名称、算子执行时间等。

总体约束

不能与**11.22.1 Profiling AscendCL API（通过Profiling AscendCL API采集并落盘性能数据）**的接口交叉调用：**aclprofModelSubscribe**接口和**aclprofModelUnSubscribe**接口之间不能调用**aclprofInit**接口、**aclprofStart**接口、**aclprofStop**接口和**aclprofFinalize**接口。

接口约束说明

- **接口调用要求**：
  - **aclprofModelSubscribe**接口在模型执行之前调用，若在模型执行过程中调用**aclprofModelSubscribe**接口，Profling采集到的数据为调用**aclprofModelSubscribe**接口之后的数据，可能导致数据不完整。
  - **aclprofModelSubscribe**接口需与**aclprofModelUnSubscribe**接口配对使用，不能在调用**aclprofModelUnSubscribe**接口前，多次调用**aclprofModelSubscribe**接口重复订阅相同的模型。
  - 不能调用**aclprofModelSubscribe**接口订阅不存在的模型ID。
  - 不能调用**aclprofModelUnSubscribe**接口取消订阅不存在的模型ID或未订阅过的模型ID。
  - 如果在同一个Device上加载了多个模型，只能对多个模型下发同样的订阅配置。
- **接口调用顺序**：
  - **建议的接口调用顺序如下**：

    模型加载-->**aclprofModelSubscribe**接口-->**aclprofGetOpDescSize**接口-->**aclprofGetOpNum**接口-->**aclprofGetOpType**/**aclprofGetOpName**/**aclprofGetOpStart**/**aclprofGetOpEnd**/**aclprofGetOpDuration**/**aclprofGetModelId**接口-->**aclprofModelUnSubscribe**接口
  - **错误的接口调用顺序示例如下**，以重复定义同一个模型为例：

    模型1加载-->**aclprofModelSubscribe**接口(指定模型1)-->**aclprofModelSubscribe**接口(指定模型1)-->**aclprofModelUnSubscribe**接口

11.22.3.2 aclprofModelSubscribe

函数功能

网络场景下，订阅算子的基本信息，包括算子名称、算子类型、算子执行耗时等。

约束说明

与**aclprofModelUnSubscribe**接口配对使用。

函数原型

aclError aclprofModelSubscribe(uint32_t modelId, const aclprofSubscribeConfig *profSubscribeConfig)

参数说明

参数名	输入/输出	说明
modelId	输入	待订阅的网络模型的ID。 调用**aclmdlLoadFromFile**接口/**aclmdlLoadFromMem**接口/**aclmdlLoadFromFileWithMem**接口/**aclmdlLoadFromMemWithMem**接口加载模型成功后，会返回模型ID。
profSubscribeConfig	输入	待订阅的配置信息。 需提前调用**aclprofCreateSubscribeConfig**接口创建aclprofSubscribeConfig类型的数据。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见**7.12 Profiling性能数据采集**。

11.22.3.3 aclprofModelUnSubscribe

函数功能

网络场景下，取消订阅算子的基本信息，包括算子名称、算子类型、算子执行耗时等。

约束说明

与**aclprofModelSubscribe**接口配对使用。

函数原型

aclError aclprofModelUnSubscribe(uint32_t modelId)

参数说明

参数名	输入/输出	说明
modelId	输入	已订阅的模型的ID。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见**7.12 Profiling性能数据采集**。

11.22.3.4 aclprofGetOpDescSize

函数功能

获取单个算子数据结构的大小，单位为Byte。当前版本中约定每个算子数据结构的大小是一样的。

建议用户新建一个线程，在新线程内调用该接口，否则可能阻塞主线程中的其它任务调度。

函数原型

aclError aclprofGetOpDescSize(size_t *opDescSize)

参数说明

参数名	输入/输出	说明
opDescSize	输出	算子数据结构的大小。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见**7.12 Profiling性能数据采集**。

11.22.3.5 aclprofGetOpNum

函数功能

获取指定内存中算子的数量。

建议用户新建一个线程，在新线程内调用该接口，否则可能阻塞主线程中的其它任务调度。

函数原型

aclError aclprofGetOpNum(const void *opInfo, size_t opInfoLen, uint32_t *opNumber)

参数说明

参数名	输入/输出	说明
opInfo	输入	指定算子信息的内存地址。 调用**aclprofGetOpDescSize**接口获取到单个算子数据结构的大小后，用户需按照“单个算子数据结构的大小*整数系数”得到的数值申请内存，用于存放Profiling采集到的算子信息数据，作为本接口的输入。
opInfoLen	输入	算子信息的长度。
opNumber	输出	算子的数量。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见**7.12 Profiling性能数据采集**。

11.22.3.6 aclprofGetOpTypeLen

函数功能

获取算子类型的字符串长度，用于内存申请。

建议用户新建一个线程，在新线程内调用该接口，否则可能阻塞主线程中的其它任务调度。

函数原型

aclError aclprofGetOpTypeLen(const void *opInfo, size_t opInfoLen, uint32_t index, size_t *opTypeLen)

参数说明

参数名	输入/输出	说明
opInfo	输入	包含算子信息的地址。
opInfoLen	输入	算子信息的长度。
index	输入	指定获取第几个算子的算子类型名称。 用户调用aclprofGetOpNum接口获取算子数量后，这个index的取值范围：[0, (算子数量-1)]。
opTypeLen	输出	opType的长度。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见7.12 Profiling性能数据采集。

11.22.3.7 aclprofGetOpType

函数功能

获取指定算子的算子类型名称。

建议用户新建一个线程，在新线程内调用该接口，否则可能阻塞主线程中的其它任务调度。

函数原型

aclError aclprofGetOpType(const void *opInfo, size_t opInfoLen, uint32_t index, char *opType, size_t opTypeLen)

参数说明

参数名	输入/输出	说明
opInfo	输入	包含算子信息的地址。
opInfoLen	输入	算子信息的长度。
index	输入	指定获取第几个算子的算子类型名称。 用户调用aclprofGetOpNum接口获取算子数量后，这个index的取值范围：[0, (算子数量-1)]。
opType	输出	算子类型名称。

参数名	输入/输出	说明
opTypeLen	输入	opType的实际内存申请长度。取值范围建议不小于aclprofGetOpTypeLen，否则内容会有截断。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见7.12 Profiling性能数据采集。

11.22.3.8 aclprofGetOpNameLen

函数功能

获取算子名称的字符串长度，用于内存申请。

建议用户新建一个线程，在新线程内调用该接口，否则可能阻塞主线程中的其它任务调度。

函数原型

aclError aclprofGetOpNameLen(const void *opInfo, size_t opInfoLen, uint32_t index, size_t *opNameLen)

参数说明

参数名	输入/输出	说明
opInfo	输入	包含算子信息的地址。
opInfoLen	输入	算子信息的长度。
index	输入	指定获取第几个算子的算子名称长度。 用户调用aclprofGetOpNum接口获取算子数量后，这个index的取值范围：[0, (算子数量-1)]。
opNameLen	输出	opName的实际内存申请长度。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见7.12 Profiling性能数据采集。

11.22.3.9 aclprofGetOpName

函数功能

获取指定算子的算子名称。

建议用户新建一个线程，在新线程内调用该接口，否则可能阻塞主线程中的其它任务调度。

函数原型

aclError aclprofGetOpName(const void *opInfo, size_t opInfoLen, uint32_t index, char *opName, size_t opNameLen)

参数说明

参数名	输入/输出	说明
opInfo	输入	包含算子信息的地址。
opInfoLen	输入	算子信息的长度。
index	输入	指定获取第几个算子的算子名称。 用户调用**aclprofGetOpNum**接口获取算子数量后，这个index的取值范围：[0, (算子数量-1)]。
opName	输出	算子名称。
opNameLen	输入	opName的实际内存申请长度。取值范围建议不小于aclprofGetOpNameLen，否则内容会有截断。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用示例，参见**7.12 Profiling性能数据采集**。

11.22.3.10 aclprofGetOpStart

函数功能

获取算子执行的开始时间，单位为ns。

建议用户新建一个线程，在新线程内调用该接口，否则可能阻塞主线程中的其它任务调度。

函数原型

uint64_t aclprofGetOpStart(const void *opInfo, size_t opInfoLen, uint32_t index)

参数说明

参数名	输入/输出	说明
opInfo	输入	包含算子信息的地址。
opInfoLen	输入	算子信息的长度。
index	输入	指定获取第几个算子执行的开始时间。 用户调用**aclprofGetOpNum**接口获取算子数量后，这个index的取值范围：[0, (算子数量-1)]。

返回值说明

算子执行的开始时间。

参考资源

接口调用示例，参见**7.12 Profiling性能数据采集**。

11.22.3.11 aclprofGetOpEnd

函数功能

获取算子执行的结束时间，单位为ns。

建议用户新建一个线程，在新线程内调用该接口，否则可能阻塞主线程中的其它任务调度。

函数原型

uint64_t aclprofGetOpEnd(const void *opInfo, size_t opInfoLen, uint32_t index)

参数说明

参数名	输入/输出	说明
opInfo	输入	包含算子信息的地址。
opInfoLen	输入	算子信息的长度。
index	输入	指定获取第几个算子执行的结束时间。 用户调用**aclprofGetOpNum**接口获取算子数量后，这个index的取值范围：[0, (算子数量-1)]。

返回值说明

算子执行结束时间。

参考资源

接口调用示例，参见7.12 Profiling性能数据采集。

11.22.3.12 aclprofGetOpDuration

函数功能

获取算子执行的耗时时间，单位为ns。

建议用户新建一个线程，在新线程内调用该接口，否则可能阻塞主线程中的其它任务调度。

函数原型

uint64_t aclprofGetOpDuration(const void *opInfo, size_t opInfoLen, uint32_t index)

参数说明

参数名	输入/输出	说明
opInfo	输入	包含算子信息的地址。
opInfoLen	输入	算子信息的长度。
index	输入	指定获取第几个算子执行的耗时时间。 用户调用aclprofGetOpNum接口获取算子数量后，这个index的取值范围：[0, (算子数量-1)]。

返回值说明

算子执行的耗时时间。

参考资源

接口调用示例，参见7.12 Profiling性能数据采集。

11.22.3.13 aclprofGetModelId

函数功能

获取指定算子所在模型的ID。

建议用户新建一个线程，在新线程内调用该接口，否则可能阻塞主线程中的其它任务调度。

函数原型

size_t aclprofGetModelId(const void *opInfo, size_t opInfoLen, uint32_t index)

参数说明

参数名	输入/输出	说明
opInfo	输入	包含算子信息的地址。
opInfoLen	输入	算子信息的长度。
index	输入	指定获取第几个算子所在模型的ID。 用户调用**aclprofGetOpNum**接口获取算子数量后，这个index的取值范围：[0, (算子数量-1)]。

返回值说明

模型的ID。

参考资源

接口调用示例，参见**7.12 Profiling性能数据采集**。

11.22.4 Profiling AscendCL API（PyTorch 场景标记迭代时间）

11.22.4.1 aclprofGetStepTimestamp

函数功能

利用**单算子加载与执行接口**实现训练的场景下，使用本接口用于标记迭代开始与结束时间，为后续Profiling解析提供迭代标识，以便以迭代为粒度展示性能数据。

函数原型

aclError aclprofGetStepTimestamp(aclprofStepInfo* stepInfo, aclprofStepTag tag, aclrtStream stream)

参数说明

参数名	输入/输出	说明
stepinfo	输入	指定迭代信息。需提前调用**aclprofCreateStepInfo**接口创建**aclprofStepInfo**类型的数据。
tag	输入	用于标记迭代开始或结束。在迭代开始时传入枚举值ACL_STEP_START，迭代结束时需传入枚举值ACL_STEP_END。
stream	输入	指定Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

11.23 Tensor 数据传输

TDT（Tensor Data Transfer ）用于Host与Device之间传递Tensor数据，比如训练的预处理后数据，比如NN计算时的loss数据。

11.23.1 acltdtCreateChannel

函数功能

创建acltdtChannelHandle类型的数据，表示可以用于向Device发送数据或是从Device接收数据的通道。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

acltdtChannelHandle *acltdtCreateChannel(uint32_t deviceId, const char *name)

参数说明

参数名	输入/输出	说明
deviceId	输入	Device ID。 用户调用**aclrtGetDeviceCount**接口获取可用的Device数量后，这个Device ID的取值范围：[0, (可用的Device数量-1)]
name	输入	队列通道名称的指针。

返回值说明

- 返回acltdtChannelHandle类型的指针，表示成功。
- 返回nullptr，表示失败。

11.23.2 acltdtSendTensor

函数功能

从Host向Device发送预处理好的数据。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

```
aclError acltdtSendTensor(const acltdtChannelHandle *handle,
const acltdtDataset *dataset,
int32_t timeout)
```

参数说明

参数名	输入/输出	说明
handle	输入	指定通道。 需提前调用acltdtCreateChannel接口或acltdtCreateChannelWithCapacity接口创建acltdtChannelHandle类型的数据。
dataset	输入	向Device发送的数据的指针。
timeout	输入	等待超时时间。 该参数取值范围如下： ● -1：阻塞方式，一直等待直到数据发送完成。 ● 0：非阻塞方式，当通道满时，直接返回通道满这个错误，这时由用户自行设定重试间隔。 ● >0：配置具体的超时时间，单位为毫秒。通道满时，等待达到超时时间后返回报错。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的超时时间为1ms，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

返回0表示成功，返回其它值表示失败。

11.23.3 acltdtReceiveTensor

函数功能

在Host接收Device发过来的数据。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

aclError acltdtReceiveTensor(const acltdtChannelHandle *handle,

acltdtDataset *dataset,

int32_t timeout)

参数说明

参数名	输入/输出	说明
handle	输入	指定通道。 需提前调用acltdtCreateChannel接口或acltdtCreateChannelWithCapacity接口创建acltdtChannelHandle类型的数据。
dataset	输出	接收到的Device数据的指针。
timeout	输入	等待超时时间。 该参数取值范围如下： ● -1：阻塞方式，一直等待直到数据接收完成。 ● 0：非阻塞方式，当通道空时，直接返回通道空这个错误，这时由用户自行设定重试间隔。 ● >0：配置具体的超时时间，单位为毫秒。通道空时，等待达到超时时间后返回报错。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的超时时间为1ms，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

返回0表示成功，返回其它值表示失败。

11.23.4 acltdtStopChannel

函数功能

调用acltdtSendTensor接口发送数据时或调用acltdtReceiveTensor接口接收数据时，用户线程可能在没有数据时会卡住，此时如果需要退出的话，需要先将线程唤醒，该接

口就是用来唤醒被卡住阻塞的线程用的。需要用户在发送、接收线程之外的一个线程里调用这个函数，来唤醒处于阻塞状态的发送/接收线程。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

aclError acltdtStopChannel(acltdtChannelHandle *handle)

参数说明

参数名	输入/输出	说明
handle	输入	指定通道。 需提前调用**acltdtCreateChannel**接口或**acltdtCreateChannelWithCapacity**接口创建acltdtChannelHandle类型的数据。

返回值说明

返回0表示成功，返回其它值表示失败。

11.23.5 acltdtDestroyChannel

函数功能

销毁acltdtChannelHandle类型的数据，只能销毁通过**acltdtCreateChannel**接口或**acltdtCreateChannelWithCapacity**接口创建的acltdtChannelHandle类型。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

aclError acltdtDestroyChannel(acltdtChannelHandle *handle)

参数说明

参数名	输入/输出	说明
handle	输入	待销毁的acltdtChannelHandle类型的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.23.6 acltdtCreateChannelWithCapacity

函数功能

创建acltdtChannelHandle类型的数据，表示可以用于向Device发送数据或是从Device接收数据的通道，通道带容量。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

acltdtChannelHandle *acltdtCreateChannelWithCapacity(uint32_t deviceId, const char *name, size_t capacity)

参数说明

参数名	输入/输出	说明
deviceId	输入	Device ID。 用户调用**aclrtGetDeviceCount**接口获取可用的Device数量后，这个Device ID的取值范围：[0, (可用的Device数量-1)]
name	输入	队列通道名称的指针。
capacity	输入	队列通道容量。

返回值说明

- 返回acltdtChannelHandle类型的指针，表示成功。
- 返回nullptr，表示失败。

11.23.7 acltdtQueryChannelSize

函数功能

查询队列通道内的消息数量。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

aclError acltdtQueryChannelSize(const acltdtChannelHandle *handle, size_t *size)

参数说明

参数名	输入/输出	说明
handle	输入	指定通道。 需提前调用acltdtCreateChannelWithCapacity接口创建acltdtChannelHandle类型的数据。
size	输出	消息数量的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.24 数据类型转换及获取数据大小

11.24.1 aclDataTypeSize

函数功能

获取aclDataType数据的大小，单位Byte。

函数原型

size_t aclDataTypeSize(aclDataType dataType)

参数说明

参数名	输入/输出	说明
dataType	输入	指定要获取大小的aclDataType数据。

返回值说明

返回aclDataType数据的大小，单位Byte。

11.24.2 aclFloat16ToFloat

函数功能

将**aclFloat16**类型的数据转换为float（指float32）类型的数据。

函数原型

float aclFloat16ToFloat(aclFloat16 value)

参数说明

参数名	输入/输出	说明
value	输入	待转换的数据。

返回值说明

转换后的数据。

11.24.3 aclFloatToFloat16

函数功能

将float（指float32）类型的数据转换为**aclFloat16**类型的数据。

函数原型

aclFloat16 aclFloatToFloat16(float value)

参数说明

参数名	输入/输出	说明
value	输入	待转换的数据。

返回值说明

转换后的数据。

注意，由于C语言无float16类型，此处返回值使用uint16_t对数据进行存储。若用户需要打印，需自行将uint16_t解释成float16进行打印，或者转成float进行打印。

11.25 共享队列管理

11.25.1 功能及约束说明

功能说明

本章提供的接口用于实现Device侧的数据队列和数据转发功能，用户可以结合本章的队列管理和共享Buffer机制以实现数据驱动编程。

约束说明

队列只允许单个消费者或单个生产者操作，Enqueue（指向队列中添加数据）或Dequeue（指从队列中获取数据）时应单线程操作，不支持并发。

11.25.2 acltdtCreateQueue

函数功能

创建队列。

函数原型

aclError acltdtCreateQueue(const acltdtQueueAttr *attr, uint32_t *qid)

参数说明

参数名	输入/输出	说明
attr	输入	队列属性配置信息的指针。 需提前调用**acltdtCreateQueueAttr**接口创建acltdtQueueAttr类型的数据，再调用**acltdtSetQueueAttr**接口设置队列属性值（例如，队列名）。 若该参数为nullptr时，则使用队列属性默认值，即队列名系统自动分配，队列深度默认为8。
qid	输出	队列ID的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.25.3 acltdtDestroyQueue

函数功能

销毁通过acltdtCreateQueue接口创建的队列。

函数原型

aclError acltdtDestroyQueue(uint32_t qid)

参数说明

参数名	输入/输出	说明
qid	输入	指定需销毁的队列。

返回值说明

返回0表示成功，返回其它值表示失败。

11.25.4 acltdtEnqueueData

函数功能

向队列中添加数据。

函数原型

aclError acltdtEnqueueData(uint32_t qid, const void *data, size_t dataSize, const void *userData, size_t userDataSize, int32_t timeout, uint32_t rsv);

参数说明

参数名	输入/输出	说明
qid	输入	需要添加数据的队列。 队列需提前调用acltdtCreateQueue接口创建。
data	输入	内存数据指针，支持Host侧或Device侧的内存。
dataSize	输入	内存数据大小，单位为Byte。
userData	输入	用户自定义数据指针。 若用户没有自定义数据，则传nullptr。
userDataSize	输入	用户自定义数据大小（<=96Byte）。 若用户没有自定义数据，则传0。

参数名	输入/输出	说明
timeout	输入	等待超时时间。当队列满时，如果向队列中添加数据，系统内部会根据设置的等待超时时间来决定如何处理。 该参数取值范围如下： ● -1：阻塞方式，一直等待直到数据成功加入队列。 ● 0：非阻塞方式（仅支持device场景，host场景无效），当队列满时，直接返回队列满这个错误，这时由用户自行设定重试间隔。 ● >0：配置具体的超时时间，单位为毫秒。队列满时，等待达到超时时间后返回报错。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的超时时间为1ms，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。
rsv	输入	预留参数，暂不支持。当前可设置为0。

返回值说明

返回0表示成功，返回其它值表示失败。

11.25.5 acltdtDequeueData

函数功能

从队列中获取数据。

函数原型

aclError acltdtDequeueData(uint32_t qid, void *data, size_t dataSize, size_t *retDataSize, void *userData, size_t userDataSize, int32_t timeout);

参数说明

参数名	输入/输出	说明
qid	输入	需要从哪个队列中获取数据。 队列需提前调用acltdtCreateQueue接口创建。

参数名	输入/输出	说明
data	输入	内存数据指针，支持Host侧或Device侧的内存。
dataSize	输入	内存数据大小，单位为Byte。
retDataSize	输入&输出	返回实际数据大小，单位为Byte。
userData	输入	用户自定义数据指针。 若用户没有自定义数据，则传nullptr。
userDataSize	输入	用户自定义数据大小（<=96Byte）。 若用户没有自定义数据，则传0。
timeout	输入	等待超时时间。当队列空时，如果从队列中获取数据，系统内部会根据设置的等待超时时间来决定如何处理。 该参数取值范围如下： ● -1：阻塞方式，一直等待直到队列有数据后返回。 ● 0：非阻塞方式（仅支持device场景，host场景无效），当队列空时，直接返回队列空这个错误，这时由用户自行设定重试间隔。 ● >0：配置具体的超时时间，单位为毫秒。队列空时，等待达到超时时间后返回报错。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的超时时间为1ms，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

返回0表示成功，返回其它值表示失败。

11.25.6 acltdtEnqueue

函数功能

向队列中添加数据，存放数据的内存必须调用**acltdtAllocBuf**接口申请。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

aclError acltdtEnqueue(uint32_t qid, acltdtBuf buf, int32_t timeout)

参数说明

参数名	输入/输出	说明
qid	输入	需要添加数据的队列。 队列需提前调用**acltdtCreateQueue**接口创建。
buf	输入	共享Buffer指针。 该内存必须提前调用**acltdtAllocBuf**接口申请。
timeout	输入	等待超时时间。当队列满时，如果向队列中添加数据，系统内部会根据设置的等待超时时间来决定如何处理。 该参数取值范围如下： ● -1：阻塞方式，一直等待直到数据成功加入队列。 ● 0：非阻塞方式，当队列满时，直接返回队列满这个错误，这时由用户自行设定重试间隔。 ● >0：配置具体的超时时间，单位为毫秒。队列满时，等待达到超时时间后返回报错。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的超时时间为1ms，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

返回0表示成功，返回其它值表示失败。

11.25.7 acltdtDequeue

函数功能

从队列中获取数据。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

aclError acltdtDequeue(uint32_t qid, acltdtBuf *buf, int32_t timeout)

参数说明

参数名	输入/输出	说明
qid	输入	需要从哪个队列中获取数据。 队列需提前调用**acltdtCreateQueue**接口创建。
buf	输出	共享Buffer指针。
timeout	输入	等待超时时间。当队列空时，如果从队列中获取数据，系统内部会根据设置的等待超时时间来决定如何处理。 该参数取值范围如下： ● -1：阻塞方式，一直等待直到队列有数据后返回。 ● 0：非阻塞方式，当队列空时，直接返回队列空这个错误，这时由用户自行设定重试间隔。 ● >0：配置具体的超时时间，单位为毫秒。队列空时，等待达到超时时间后返回报错。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的超时时间为1ms，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

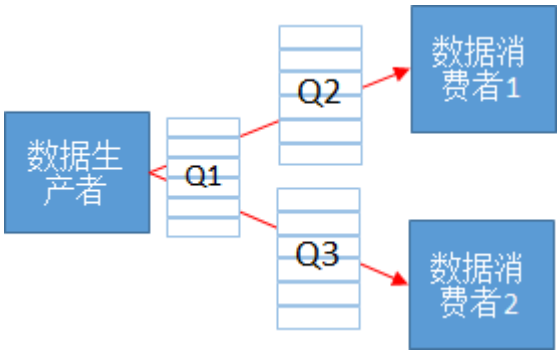
返回0表示成功，返回其它值表示失败。

11.25.8 acltdtBindQueueRoutes

函数功能

当应用存在数据一对多分发时，通过本接口绑定队列间数据转发路由关系。

如下图所示，可以建立两条路由关系Q1->Q2, Q1->Q3。



约束说明

- 数据一对多分发时，传递的共享Buffer数据是无锁的，消费者不能对数据进行inplace操作，如上图所示消费者1对共享数据的修改会导致消费者2访问的数据发生变化。
- 会对添加的队列路由关系的进行是否成环校验，不允许成环。
- 不支持多线程并发调用。

函数原型

aclError acltdtBindQueueRoutes(acltdtQueueRouteList *qRouteList)

参数说明

参数名	输入/输出	说明
qRouteList	输入/输出	路由关系数组的指针，接口调用完成后返回路由绑定结果。 需提前调用acltdtCreateQueueRouteList接口创建acltdtQueueRouteList类型的数据，再调用acltdtAddQueueRoute接口添加路由关系。

返回值说明

返回0表示成功，返回其它值表示失败。

只有当所有队列关系绑定成功且路由状态正常时，本接口才会返回成功；任何一条绑定失败，本接口返回失败，如果您需要知道具体哪个路由关系绑定失败，您可以先调用acltdtGetQueueRoute接口从路由关系数组中获取每一个路由关系，再调用acltdtGetQueueRouteParam接口查询绑定关系状态。

11.25.9 acltdtUnbindQueueRoutes

函数功能

解绑定数据队列路由关系。

函数原型

aclError acltdtUnbindQueueRoutes(acltdtQueueRouteList *qRouteList)

参数说明

参数名	输入/输出	说明
qRouteList	输入/输出	路由关系数组的指针，接口调用完成后返回路由去绑定结果。 可先通过acltdtQueryQueueRoutes获取路由关系数组。

返回值说明

返回0表示成功，返回其它值表示失败。

只有当所有队列关系解绑定成功，本接口才会返回成功；任何一条解绑定失败，本接口返回失败，如果您需要知道具体哪个路由关系解绑定失败，您可以先调用acltdtGetQueueRoute接口从路由关系数组中获取每一个路由关系，再调用acltdtGetQueueRouteParam接口查询绑定关系状态。

11.25.10 acltdtQueryQueueRoutes

函数功能

根据指定条件查询数据队列路由关系。

函数原型

aclError acltdtQueryQueueRoutes(const acltdtQueueRouteQueryInfo *queryInfo, acltdtQueueRouteList *qRouteList)

参数说明

参数名	输入/输出	说明
queryInfo	输入	查询条件的指针。 需提前调用acltdtCreateQueueRouteQueryInfo接口创建acltdtQueueRouteQueryInfo类型的数据。
qRouteList	输入&输出	路由关系数组的指针。 需提前调用acltdtCreateQueueRouteList接口创建acltdtQueueRouteList类型的数据。

返回值说明

返回0表示成功，返回其它值表示失败。

11.25.11 acltdtGrantQueue

函数功能

进程间需要共享队列信息时，可以调用本接口给其它进程授予队列相关的权限，例如Enqueue（指向队列中添加数据）权限、Dequeue（指从队列中获取数据）权限等。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

约束说明

进程间传递队列相关信息时，安全性由用户保证。

函数原型

aclError acltdtGrantQueue(uint32_t qid, int32_t pid, uint32_t permission, int32_t timeout)

参数说明

参数名	输入/输出	说明
qid	输入	队列ID。
pid	输入	被授权进程的ID。
permission	输入	权限标识（队列生产者/消费者）。 用户选择如下多个宏进行逻辑或（例如：ACL_TDT_QUEUE_PERMISSION_DEQUEUE | ACL_TDT_QUEUE_PERMISSION_ENQUEUE），作为permission参数值。每个宏表示某一权限，详细说明如下： ● ACL_TDT_QUEUE_PERMISSION_DEQUEUE：表示Dequeue权限。 ● ACL_TDT_QUEUE_PERMISSION_ENQUEUE：表示Enqueue权限。

参数名	输入/输出	说明
timeout	输入	等待超时时间，取值范围如下： ● -1：阻塞方式，一直等待直到数据成功加入队列。 ● 0：非阻塞方式，立即返回。 ● >0：配置具体的超时时间，单位为毫秒，等到到达超时时间后返回报错。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的超时时间为1ms，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。

返回值说明

返回0表示成功，返回其它值表示失败。

11.25.12 acltdtAttachQueue

函数功能

进程间需要共享队列信息时，在被授权的进程中调用本接口确认当前进程对队列有相应权限。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

aclError acltdtAttachQueue(uint32_t qid, int32_t timeout, uint32_t *permission)

参数说明

参数名	输入/输出	说明
qid	输入	队列ID。

参数名	输入/输出	说明
timeout	输入	等待超时时间，取值范围如下： ● -1：阻塞方式，一直等待直到数据成功加入队列。 ● 0：非阻塞方式，立即返回。 ● >0：配置具体的超时时间，单位为毫秒，等到到达超时时间后返回报错。超时时间受操作系统影响，一般偏差在操作系统的一个时间片内，例如，操作系统的一个时间片为4ms，用户设置的超时时间为1ms，则实际的超时时间在1ms到5ms范围内。在CPU负载高场景下，超时时间仍可能存在波动。
permission	输出	权限标识。 ● 根据输出参数值的倒数第二个bit位判断是否有从队列中获取数据的权限，0就是没有，1就是有。 ● 根据输出参数值的倒数第三个bit位判断是否有向队列中添加数据的权限，0就是没有，1就是有。

返回值说明

返回0表示成功，返回其它值表示失败。

11.26 共享 Buffer 管理

本特性提供了跨进程共享Buffer管理能力，配合共享队列一起使用。共享Buffer的共享范围即是共享队列的授权确定，共享范围包括“一主多从”的这些进程。

11.26.1 acltdtAllocBuf

函数功能

申请共享Buffer内存。

约束说明

- 使用acltdtAllocBuf接口申请内存后，数据区的长度为size参数的大小，在用户还未填入有效数据前，该内存的有效数据长度初始值为0，可在用户向内存中填入有效数据后，再通过**acltdtSetBufDataLen**接口设置有效数据长度。
- 使用acltdtAllocBuf接口申请的内存，需要通过**acltdtFreeBuf**接口释放内存。
- 仅支持在**Control CPU开放形态**或**RC模式**下使用该接口。

函数原型

aclError acltdtAllocBuf(size_t size, uint32_t type, acltdtBuf *buf)

参数说明

参数名	输入/输出	说明
size	输入	用于指定申请数据区大小，单位Byte，不能超过4G。
type	输入	共享Buffer内存类型，支持设置如下枚举值。 typedef enum { ACL_TDT_NORMAL_MEM = 0, ACL_TDT_DVPP_MEM } acltdtAllocBufType;
buf	输出	申请成功，输出共享Buffer指针。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**7.15 共享Buffer管理**。

11.26.2 acltdtFreeBuf

函数功能

释放通过**acltdtAllocBuf**接口申请的mbuf。

约束说明

仅支持在**Control CPU开放形态**或**RC模式**下使用该接口。

函数原型

aclError acltdtFreeBuf(acltdtBuf buf)

参数说明

参数名	输入/输出	说明
buf	输入	指定要释放的mbuf。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**7.15 共享Buffer管理**。

11.26.3 acltdtGetBufData

函数功能

获取共享Buffer的数据区指针和数据区长度，用户可以使用此指针填入数据。

约束说明

- 接口调用顺序：调用**acltdtAllocBuf**或**acltdtCopyBufRef**接口申请到共享Buffer后，因此需由用户调用**acltdtGetBufData**接口获取共享Buffer的内存指针及长度后，再自行向内存中填充有效数据，然后再调用**acltdtSetBufDataLen**接口设置共享Buffer中有效数据的长度，且长度必须小于**acltdtGetBufData**获取到的size大小。
- 仅支持在**Control CPU开放形态**或**RC模式**下使用该接口。

函数原型

aclError acltdtGetBufData(const acltdtBuf buf, void **dataPtr, size_t *size)

参数说明

参数名	输入/输出	说明
buf	输入	共享Buffer指针，须通过**acltdtAllocBuf**接口申请获得。
dataPtr	输出	数据区指针（device侧地址）。
size	输出	数据区的长度，单位为Byte。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**7.15 共享Buffer管理**。

11.26.4 acltdtSetBufUserData

函数功能

设置共享Buffer的私有数据区数据，从用户内存拷贝到共享内存的私有数据区的指定偏移位置，用于设置控制信息做为上下文传递。

约束说明

当前默认私有数据区大小是96Byte，offset+size必须小于或等于96Byte，否则返回报错。

仅支持在**Control CPU开放形态**或**RC模式**下使用该接口。

函数原型

aclError acltdtSetBufUserData (acltdtBuf buf, const void *dataPtr, size_t size, size_t offset)

参数说明

参数名	输入/输出	说明
buf	输入	共享Buffer指针，须通过**acltdtAllocBuf**或**acltdtCopyBufRef**接口申请获得。
dataPtr	输入	存放用户数据的内存地址指针。
size	输入	用户数据的长度，单位为Byte。 数据长度小于或等于96Byte。
offset	输入	地址偏移，单位为Byte。 偏移量小于或等于96Byte。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见**7.15 共享Buffer管理**。

11.26.5 acltdtGetBufUserData

函数功能

获取共享Buffer的私有数据区数据，偏移offset后并拷贝至用户申请的内存区域。

约束说明

当前默认私有数据区大小是96Byte，offset+size必须小于或等于96Byte，否则返回报错。

仅支持在**Control CPU开放形态**或**RC模式**下使用该接口。

函数原型

aclError acltdtGetBufUserData (const acltdtBuf buf, void *dataPtr, size_t size, size_t offset)

参数说明

参数名	输入/输出	说明
buf	输入	共享Buffer指针，须通过**acltdtAllocBuf**或**acltdtCopyBufRef**接口申请获得。
dataPtr	输入	存放用户数据的内存地址指针。
size	输入	用户数据的长度，单位为Byte。 数据长度小于或等于96Byte。
offset	输入	地址偏移，单位为Byte。 偏移量小于或等于96Byte。

返回值说明

返回0表示成功，返回其它值表示失败。

11.26.6 acltdtSetBufDataLen

设置共享Buffer中有效数据的长度。

约束说明

- 接口调用顺序：调用**acltdtAllocBuf**或**acltdtCopyBufRef**接口申请到共享Buffer后，因此需由用户调用**acltdtGetBufData**接口获取共享Buffer的内存指针及长度后，再自行向内存中填充有效数据。
- 仅支持在**Control CPU开放形态**或**RC模式**下使用该接口。

函数原型

aclError acltdtSetBufDataLen(acltdtBuf buf, size_t len)

参数说明

参数名	输入/输出	说明
buf	输入	共享Buffer指针，须通过acltdtAllocBuf或acltdtCopyBufRef接口申请获得。
len	输入	有效数据的长度，单位为Byte。

返回值说明

返回0表示成功，返回其它值表示失败。

参考资源

接口调用流程及示例，参见7.15 共享Buffer管理。

11.26.7 acltdtGetBufDataLen

获取共享Buffer中有效数据的长度。

约束说明

通过acltdtSetBufDataLen接口设置共享Buffer中有效数据的长度后，可调用本接口获取有效数据的长度，否则，通过本接口获取到的长度为0。

仅支持在Control CPU开放形态或RC模式下使用该接口。

函数原型

aclError acltdtGetBufDataLen(acltdtBuf buf, size_t *len)

参数说明

参数名	输入/输出	说明
buf	输入	共享Buffer指针，须通过acltdtAllocBuf或acltdtCopyBufRef接口申请获得。
len	输出	有效数据的长度，单位为Byte。

返回值说明

返回0表示成功，返回其它值表示失败。

11.26.8 acltdtCopyBufRef

对共享Buffer数据区的引用拷贝，创建并返回一个新的Buffer管理结构指向相同的数据区。

约束说明

仅支持在Control CPU开放形态或RC模式下使用该接口。

函数原型

aclError acltdtCopyBufRef(const acltdtBuf buf, acltdtBuf *newBuf)

参数说明

参数名	输入/输出	说明
buf	输入	共享Buffer指针，须通过acltdtAllocBuf或acltdtCopyBufRef接口申请获得。
newBuf	输出	返回一个新的共享Buffer指针，指向相同的数据区。

返回值说明

返回0表示成功，返回其它值表示失败。

11.26.9 acltdtAppendBufChain

将某个共享Buffer挂接到headBuf指向的共享Buffer的尾部，headBuf指向的共享Buffer为共享Buffer链头部的第一个共享Buffer，可以是一个单独的共享Buffer（这相当于共享Buffer链只有一个节点）。

约束说明

仅支持在Control CPU开放形态或RC模式下使用该接口。

共享Buffer链最大支持128个共享Buffer。

函数原型

aclError acltdtAppendBufChain(acltdtBuf headBuf, acltdtBuf buf)

参数说明

参数名	输入/输出	说明
headBuf	输入	共享Buffer链头部的第一个共享Buffer的指针，须通过acltdtAllocBuf或acltdtCopyBufRef接口申请获得。
buf	输入	待Append的buf指针，须通过acltdtAllocBuf或acltdtCopyBufRef接口申请获得。

返回值说明

返回0表示成功，返回其它值表示失败。

11.26.10 acltdtGetBufChainNum

获取共享Buffer链中的共享Buffer数量。

约束说明

仅支持在Control CPU开放形态或RC模式下使用该接口。

函数原型

aclError acltdtGetBufChainNum(acltdtBuf headBuf, uint32_t* num)

参数说明

参数名	输入/输出	说明
headBuf	输入	共享Buffer链头部的第一个共享Buffer的指针，须通过acltdtAllocBuf或acltdtCopyBufRef接口申请获得。
num	输出	共享Buffer链中的共享Buffer数量。

返回值说明

返回0表示成功，返回其它值表示失败。

11.26.11 acltdtGetBufFromChain

获取Mbuf链中第index个Mbuf。

约束说明

仅支持在Control CPU开放形态或RC模式下使用该接口。

函数原型

aclError acltdtGetBufFromChain(acltdtBuf headBuf, uint32_t index, acltdtBuf *buf)

参数说明

参数名	输入/输出	说明
headBuf	输入	共享Buffer链头部的第一个共享Buffer的指针，须通过acltdtAllocBuf或acltdtCopyBufRef接口申请获得。
index	输入	共享Buffer链中的共享Buffer序号（从0开始计数）。
buf	输出	返回第index个共享Buffer指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27 数据类型及其操作接口

11.27.1 aclError

```
typedef int aclError;
```

说明

返回码定义规则：

- 规则1：开发人员的环境异常或者代码逻辑错误，可以通过优化环境或代码逻辑的方式解决问题，此时返回码定义为：1XXXXX。
- 规则2：资源不足（Stream、内存等）、开发人员编程时使用的接口或参数与当前硬件不匹配，可以通过在编程时合理使用资源的方式解决，此时返回码定义为：2XXXXX。
- 规则3：业务功能异常，比如队列满、队列空等，此时返回码定义为3XXXXX。
- 规则4：软硬件内部异常，包括软件内部错误、Device执行失败等，用户无法解决问题，需要将问题反馈给技术支持，此时返回码定义为：5XXXXX。您可以获取日志后单击Link联系技术支持。
- 规则5：无法识别的错误，当前都映射为500000。您可以获取日志后单击Link联系技术支持。

表 11-77 返回码列表

返回码	含义	可能原因及解决方法
static const int ACL_SUCCESS = 0;	执行成功	-
static const int ACL_ERROR_NONE = 0; **须知** 此返回码后续版本会废弃，请使用ACL_SUCCESS返回码。	执行成功。	-
static const int ACL_ERROR_INVALID_PARAM = 100000;	参数校验失败。	请检查接口的入参值是否正确。
static const int ACL_ERROR_UNINITIALIZE = 100001;	未初始化。	• 请检查是否已调用**aclInit**接口进行初始化，请确保已调用**aclInit**接口，且在其它AscendCL接口之前调用。 • 请检查是否已调用对应功能的初始化接口，例如初始化Dump的**aclmdlInitDump**接口、初始化Profiling的**aclprofInit**接口。
static const int ACL_ERROR_REPEAT_INITIALIZE = 100002;	重复初始化或重复加载。	请检查是否调用对应的接口重复初始化或重复加载。
static const int ACL_ERROR_INVALID_FILE = 100003;	无效的文件。	请检查文件是否存在、文件是否能被访问等。
static const int ACL_ERROR_WRITE_FILE = 100004;	写文件失败。	请检查文件路径是否存在、文件是否有写权限等。
static const int ACL_ERROR_INVALID_FILE_SIZE = 100005;	无效的文件大小。	请检查文件大小是否符合接口要求。
static const int ACL_ERROR_PARSE_FILE = 100006;	解析文件失败。	请检查文件内容是否合法。
static const int ACL_ERROR_FILE_MISSING_ATTR = 100007;	文件缺失参数。	请检查文件内容是否完整。

返回码	含义	可能原因及解决方法
static const int ACL_ERROR_FILE_ATTR_INVALID = 100008;	文件参数无效。	请检查文件中参数值是否正确。
static const int ACL_ERROR_INVALID_DUMP_CONFIG = 100009;	无效的Dump配置。	请检查Dump配置是否正确，详细配置请参见《**CANN 开发工具指南（开放态）**》中的“精度比对工具”章节。
static const int ACL_ERROR_INVALID_PROFILING_CONFIG = 100010;	无效的Profiling配置。	请检查Profiling配置是否正确。
static const int ACL_ERROR_INVALID_MODEL_ID = 100011;	无效的模型ID。	请检查模型ID是否正确、模型是否正确加载。
static const int ACL_ERROR_DESERIALIZE_MODEL = 100012;	反序列化模型失败。	模型可能与当前版本不匹配，请重新构建模型。
static const int ACL_ERROR_PARSE_MODEL = 100013;	解析模型失败。	模型可能与当前版本不匹配，请重新构建模型。
static const int ACL_ERROR_READ_MODEL_FAILURE = 100014;	读取模型失败。	请检查模型文件是否存在、模型文件是否能被访问等。
static const int ACL_ERROR_MODEL_SIZE_INVALID = 100015;	无效的模型大小。	模型文件无效，请重新构建模型。
static const int ACL_ERROR_MODEL_MISSING_ATTR = 100016;	模型缺少参数。	模型可能与当前版本不匹配，请重新构建模型。
static const int ACL_ERROR_MODEL_INPUT_NOT_MATCH = 100017;	模型的输入不匹配。	请检查模型的输入是否正确。
static const int ACL_ERROR_MODEL_OUTPUT_NOT_MATCH = 100018;	模型的输出不匹配。	请检查模型的输出是否正确。
static const int ACL_ERROR_MODEL_NOT_DYNAMIC = 100019;	非动态模型。	请检查当前模型是否支持动态场景，如不支持，请重新构建模型。

返回码	含义	可能原因及解决方法
static const int ACL_ERROR_OP_TYPE_NOT_MATCH = 100020;	单算子类型不匹配。	请检查算子类型是否正确。
static const int ACL_ERROR_OP_INPUT_NOT_MATCH = 100021;	单算子的输入不匹配。	请检查算子的输入是否正确。
static const int ACL_ERROR_OP_OUTPUT_NOT_MATCH = 100022;	单算子的输出不匹配。	请检查算子的输出是否正确。
static const int ACL_ERROR_OP_ATTR_NOT_MATCH = 100023;	单算子的属性不匹配。	请检查算子的属性是否正确。
static const int ACL_ERROR_OP_NOT_FOUND = 100024;	单算子未找到。	请检查算子类型是否支持。
static const int ACL_ERROR_OP_LOAD_FAILED = 100025;	单算子加载失败。	模型可能与当前版本不匹配，请重新构建单算子模型。
static const int ACL_ERROR_UNSUPPORTED_DATA_TYPE = 100026;	不支持的数据类型。	请检查数据类型是否存在或当前是否支持。
static const int ACL_ERROR_FORMAT_NOT_MATCH = 100027;	Format不匹配。	请检查Format是否正确。
static const int ACL_ERROR_BIN_SELECTOR_NOT_REGISTERED = 100028;	使用二进制选择方式编译算子接口时，算子未注册选择器。	请检查是否调用**aclopRegisterSelectKernelFunc**接口注册算子选择器。
static const int ACL_ERROR_KERNEL_NOT_FOUND = 100029;	编译算子时，算子Kernel未注册。	请检查是否调用**aclopCreateKernel**接口注册算子Kernel。
static const int ACL_ERROR_BIN_SELECTOR_ALREADY_REGISTERED = 100030;	使用二进制选择方式编译算子接口时，算子重复注册。	请检查是否重复调用**aclopRegisterSelectKernelFunc**接口注册算子选择器。
static const int ACL_ERROR_KERNEL_ALREADY_REGISTERED = 100031;	编译算子时，算子Kernel重复注册。	请检查是否重复调用**aclopCreateKernel**接口注册算子Kernel。

返回码	含义	可能原因及解决方法
static const int ACL_ERROR_INVALID_QUEUE_ID = 100032;	无效的队列ID。	请检查队列ID是否正确。
static const int ACL_ERROR_REPEAT_SUBSCRIBE = 100033;	重复订阅。	请检查针对同一个Stream，是否重复调用**aclrtSubscribeReport**接口。
static const int ACL_ERROR_STREAM_NOT_SUBSCRIBE = 100034; **须知** 此返回码后续版本会废弃，请使用**ACL_ERROR_RT_STREAM_NO_CB_REG**返回码。	Stream未订阅。	请检查是否已调用**aclrtSubscribeReport**接口。
static const int ACL_ERROR_THREAD_NOT_SUBSCRIBE = 100035; **须知** 此返回码后续版本会废弃，请使用**ACL_ERROR_RT_THREAD_SUBSCRIBE**返回码。	线程未订阅。	请检查是否已调用**aclrtSubscribeReport**接口。
static const int ACL_ERROR_WAIT_CALLBACK_TIMEOUT = 100036; **须知** 此返回码后续版本会废弃，请使用**ACL_ERROR_RT_REPORT_TIMEOUT**返回码。	等待callback超时。	请检查是否已调用**aclrtLaunchCallback**接口下发callback任务； 请检查**aclrtProcessReport**接口中超时时间是否合理； 请检查callback任务是否已经处理完成，如果已处理完成，但还调用**aclrtProcessReport**接口，则需优化代码逻辑。
static const int ACL_ERROR_REPEAT_FINALIZE = 100037;	重复去初始化。	请检查是否重复调用**aclFinalize**接口进行去初始化。
static const int ACL_ERROR_NOT_STATIC_AIPP = 100038; **须知** 此返回码后续版本会废弃，请使用**ACL_ERROR_GE_AIPP_NOT_EXIST**返回码。	静态AIPP配置信息不存在。	调用**aclmdlGetFirstAippInfo**接口时，请传入正确的index值。

返回码	含义	可能原因及解决方法
static const int ACL_ERROR_COMPILING_STUB_MODE = 100039;	运行应用前配置的动态库路径是编译桩的路径，不是正确的动态库路径。	请检查动态库路径的配置，确保使用运行模式的动态库。
static const int ACL_ERROR_GROUP_NOT_SET = 100040; **须知** 此返回码后续版本会废弃，请使用**ACL_ERROR_RT_GROUP_NOT_SET**返回码。	未设置Group。	请检查是否已调用**aclrtSetGroup**接口。
static const int ACL_ERROR_GROUP_NOT_CREATE = 100041; **须知** 此返回码后续版本会废弃，请使用**ACL_ERROR_RT_GROUP_NOT_CREATE**返回码。	未创建对应的Group。	请检查调用接口时设置的Group ID是否在支持的范围内，Group ID的取值范围：[0, (Group数量-1)]，用户可调用**aclrtGetGroupCount**接口获取Group数量。
static const int ACL_ERROR_PROF_ALREADY_RUN = 100042;	已存在采集Profiling数据的任务。	• 请检查代码逻辑，“通过调用AscendCL API方式采集Profiling数据”的配置不能与其它方式的Profiling配置并存，只能保留一种，各种方式的Profiling采集配置请参见《**CANN 开发工具指南（开放态）**》中的“性能分析工具”章节。 • 请检查是否对同一个Device重复下发了多次Profiling配置，请参考**11.22.1 Profiling AscendCL API（通过Profiling AscendCL API采集并落盘性能数据）**中的接口调用要求、接口调用顺序调整代码逻辑。
static const int ACL_ERROR_PROF_NOT_RUN = 100043;	未使用**aclprofInit**接口先进行Profiling初始化。	请检查接口调用顺序，参考**11.22.1 Profiling AscendCL API（通过Profiling AscendCL API采集并落盘性能数据）**中的说明。

返回码	含义	可能原因及解决方法
static const int ACL_ERROR_DUMP_ALREADY_RUN = 100044;	已存在获取Dump数据的任务。	请检查在调用**aclmdlInitDump**接口、**aclmdlSetDump**接口、**aclmdlFinalizeDump**接口配置Dump信息前，是否已调用**aclInit**接口配置Dump信息，如是，请调整代码逻辑，保留一种方式配置Dump信息即可。
static const int ACL_ERROR_DUMP_NOT_RUN = 100045;	未使用**aclmdlInitDump**接口先进行Dump初始化。	请检查获取Dump数据的接口调用顺序，参考**aclmdlInitDump**接口处的说明。
static const int ACL_ERROR_PROF_REPEAT_SUBSCRIBE = 148046;	重复订阅同一个模型。	请检查接口调用顺序，参考**11.22.1 Profiling AscendCL API（通过Profiling AscendCL API采集并落盘性能数据）**中的说明。
static const int ACL_ERROR_PROF_API_CONFLICT = 148047;	采集性能数据的接口调用冲突。	两种方式的Profiling性能数据采集接口不能交叉调用，**aclprofInit**接口和**aclprofFinalize**接口之间不能调用**aclprofModelSubscribe**接口、aclprofGet*接口、**aclprofModelUnSubscribe**接口，**aclprofModelSubscribe**接口和**aclprofModelUnSubscribe**接口之间不能调用**aclprofInit**接口、**aclprofStart**接口、**aclprofStop**接口、**aclprofFinalize**。
static const int ACL_ERROR_INVALID_MAX_OPQUEUE_NUM_CONFIG = 148048;	无效的算子缓存信息老化配置。	请检查算子缓存信息老化配置，参考**aclInit**处的配置说明及示例。
static const int ACL_ERROR_INVALID_OPP_PATH = 148049;	没有设置ASCEND_OPP_PATH环境变量，或该环境变量的值设置错误。	请检查是否设置ASCEND_OPP_PATH环境变量，且该环境变量的值是否为opp软件包的安装路径。

返回码	含义	可能原因及解决方法
static const int ACL_ERROR_OP_UNSUPPORTED_DYNAMIC = 148050;	算子不支持动态Shape。	• 请检查单算子模型文件中该算子的Shape是否为动态，如果是动态的，需要修改为固定Shape。 • 请检查编译算子时，aclTensorDesc的Shape是否为动态，如果是动态的，需要按照固定Shape重新创建aclTensorDesc。
static const int ACL_ERROR_RELATIVE_RESOURCE_NOT_CLEARED = 148051;	相关的资源尚未释放。	在销毁通道描述信息时，如果相关的通道尚未销毁则返回此错误码。请检查与此通道描述信息相关联的通道是否被销毁。
static const int ACL_ERROR_UNSUPPORTED_JPEG = 148052;	JPEGD功能不支持的输入图片编码格式（例如算术编码、渐进式编码等）。	实现JPEGD图片解码功能时，仅支持Huffman编码，压缩前的原图像色彩空间为YUV，像素的各分量比例为4:4:4或4:2:2或4:2:0或4:0:0或4:4:0，不支持算术编码、不支持渐进JPEG格式、不支持JPEG2000格式。
static const int ACL_ERROR_BAD_ALLOC = 200000;	申请内存失败。	请检查硬件环境上的内存剩余情况。
static const int ACL_ERROR_API_NOT_SUPPORT = 200001;	接口不支持。	请检查调用的接口当前是否支持。
static const int ACL_ERROR_INVALID_DEVICE = 200002; **须知** 此返回码后续版本会废弃，请使用**ACL_ERROR_RT_INVALID_DEVICEID**返回码。	无效的Device。	请检查Device是否存在。
static const int ACL_ERROR_MEMORY_ADDRESS_UNALIGNED = 200003;	内存地址未对齐。	请检查内存地址是否符合接口要求。
static const int ACL_ERROR_RESOURCE_NOT_MATCH = 200004;	资源不匹配。	请检查调用接口时，是否传入正确的Stream、Context等资源。

返回码	含义	可能原因及解决方法
static const int ACL_ERROR_INVALID_RESOURCE_HANDLE = 200005;	无效的资源句柄。	请检查调用接口时，传入的Stream、Context等资源是否已被销毁或占用。
static const int ACL_ERROR_FEATURE_UNSUPPORTED = 200006;	特性不支持。	您可以获取日志后单击**Link**联系技术支持。
static ACL_ERROR_PROF_MODULES_UNSUPPORTED = 200007;	下发了不支持的Profiling配置。	请参见**11.27.87.1 aclprofCreateConfig**中的说明检查Profiling的配置是否正确。
static const int ACL_ERROR_STORAGE_OVER_LIMIT = 300000;	超出存储上限。	请检查硬件环境上的存储剩余情况。
static const int ACL_ERROR_INTERNAL_ERROR = 500000;	未知内部错误。	您可以获取日志后单击**Link**联系技术支持。
static const int ACL_ERROR_FAILURE = 500001;	系统内部AscendCL的错误。	您可以获取日志后单击**Link**联系技术支持。
static const int ACL_ERROR_GE_FAILURE = 500002;	系统内部GE的错误。	您可以获取日志后单击**Link**联系技术支持。
static const int ACL_ERROR_RT_FAILURE = 500003;	系统内部RUNTIME的错误。	您可以获取日志后单击**Link**联系技术支持。
static const int ACL_ERROR_DRV_FAILURE = 500004;	系统内部DRV（Driver）的错误。	您可以获取日志后单击**Link**联系技术支持。
static const int ACL_ERROR_PROFILING_FAILURE = 500005;	Profiling相关错误。	您可以获取日志后单击**Link**联系技术支持。

表 11-78 内部 RUNTIME 的返回码列表

返回码	含义	可能原因及解决方法
static const int32_t ACL_RT_SUCCESS = 0;	执行成功。	-
static const int32_t ACL_ERROR_RT_PARAM_INVALID = 107000;	参数校验失败。	请检查接口入参是否正确。

返回码	含义	可能原因及解决方法
static const int32_t ACL_ERROR_RT_INVALID_DEVICEID = 107001;	无效的Device ID。	请检查Device ID是否合法。
static const int32_t ACL_ERROR_RT_CONTEXT_NULL = 107002;	context为空。	请检查是否调用**aclrtSetCurrentContext**或**aclrtSetDevice**。
static const int32_t ACL_ERROR_RT_STREAM_CONTEXT = 107003;	stream不在当前context内。	请检查stream所在的context与当前context是否一致。
static const int32_t ACL_ERROR_RT_MODEL_CONTEXT = 107004;	model不在当前context内。	请检查加载的模型与当前context是否一致。
static const int32_t ACL_ERROR_RT_STREAM_MODEL = 107005;	stream不在当前model内。	请检查stream是否绑定过该模型。
static const int32_t ACL_ERROR_RT_EVENT_TIMESTAMP_INVALID = 107006;	event时间戳无效。	请检查event是否创建。
static const int32_t ACL_ERROR_RT_EVENT_TIMESTAMP_REVERSAL = 107007;	event时间戳反转。	请检查event是否创建。
static const int32_t ACL_ERROR_RT_ADDR_UNALIGNED = 107008;	内存地址未对齐。	请检查所申请的内存地址是否对齐，详细内存申请接口的约束请参见**11.11 内存管理**。
static const int32_t ACL_ERROR_RT_FILE_OPEN = 107009;	打开文件失败。	请检查文件是否存在。
static const int32_t ACL_ERROR_RT_FILE_WRITE = 107010;	写文件失败。	请检查文件是否存在或者是否具备写权限。
static const int32_t ACL_ERROR_RT_STREAM_SUBSCRIBE = 107011;	stream未订阅或重复订阅。	请检查当前stream是否订阅或重复订阅。
static const int32_t ACL_ERROR_RT_THREAD_SUBSCRIBE = 107012;	线程未订阅或重复订阅。	请检查当前线程是否订阅或重复订阅。
static const int32_t ACL_ERROR_RT_GROUP_NOT_SET = 107013;	未设置Group。	请检查是否已调用**aclrtSetGroup**接口。

返回码	含义	可能原因及解决方法
static const int32_t ACL_ERROR_RT_GROUP_NOT_CREATE = 107014;	未创建对应的Group。	请检查调用接口时设置的Group ID是否在支持的范围内，Group ID的取值范围：[0, (Group数量-1)]，用户可调用**aclrtGetGroupCount**接口获取Group数量。
static const int32_t ACL_ERROR_RT_STREAM_NO_CB_REG = 107015;	该callback对应的stream未注册到线程。	请检查stream是否已经注册到线程，检查是否调用**aclrtSubscribeReport**接口。
static const int32_t ACL_ERROR_RT_INVALID_MEMORY_TYPE = 107016;	无效的内存类型。	请检查内存类型是否合法。
static const int32_t ACL_ERROR_RT_INVALID_HANDLE = 107017;	无效的资源句柄。	检查对应输入和使用的参数是否正确.
static const int32_t ACL_ERROR_RT_INVALID_MALLOC_TYPE = 107018;	申请使用的内存类型不正确。	检查对应输入和使用的内存类型是否正确。
static const int32_t ACL_ERROR_RT_WAIT_TIMEOUT = 107019;	等待超时。	请尝试重新执行下发任务的接口。
static const int32_t ACL_ERROR_RT_TASK_TIMEOUT = 107020;	执行任务超时。	请排查业务编排是否合理或者设置合理的超时时间。
static const int32_t ACL_ERROR_RT_SYSPARAMOPT_NOT_SET = 107021;	获取当前Context中的系统参数值失败。	未设置当前Context中的系统参数值，请参见**11.6.5 aclrtCtxSetSysParamOpt**。
static const int32_t ACL_ERROR_RT_FEATURE_NOT_SUPPORT = 207000;	特性不支持。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_MEMORY_ALLOCATION = 207001;	内存申请失败。	请检查硬件环境上的存储剩余情况。
static const int32_t ACL_ERROR_RT_MEMORY_FREE = 207002;	内存释放失败。	您可以获取日志后单击**Link**联系技术支持。

返回码	含义	可能原因及解决方法
static const int32_t ACL_ERROR_RT_AICORE_OVER_FLOW = 207003;	aicore算子运算溢出。	请检查对应的aicore算子运算是否有溢出，可以根据dump数据找到对应溢出的算子，再定位具体的算子问题。
static const int32_t ACL_ERROR_RT_NO_DEVICE = 207004;	Device不可用。	请检查Device是否正常运行。
static const int32_t ACL_ERROR_RT_RESOURCE_ALLOC_FAIL = 207005;	内存申请失败。	请检查硬件环境上的存储剩余情况。
static const int32_t ACL_ERROR_RT_NO_PERMISSION = 207006;	没有操作权限。	请检查运行应用的用户权限是否正确。
static const int32_t ACL_ERROR_RT_NO_EVENT_RESOURCE = 207007;	Event资源不足。	请参考**11.9.1 aclrtCreateEvent**接口处的说明检查Event数量是否符合要求。
static const int32_t ACL_ERROR_RT_NO_STREAM_RESOURCE = 207008;	Stream资源不足。	请参考**11.8.1 aclrtCreateStream**接口处的说明检查Stream数量是否符合要求。
static const int32_t ACL_ERROR_RT_NO_NOTIFY_RESOURCE = 207009;	系统内部Notify资源不足。	媒体数据处理的并发任务太多或模型推理时消耗资源太多，建议尝试减少并发任务或卸载部分模型。
static const int32_t ACL_ERROR_RT_NO_MODEL_RESOURCE = 207010;	模型资源不足。	建议卸载部分模型。
static const int32_t ACL_ERROR_RT_NO_CDQ_RESOURCE = 207011;	Runtime内部资源不足。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_OVER_LIMIT = 207012;	队列数目超出上限。	请销毁不需要的队列之后再创建新的队列。
static const int32_t ACL_ERROR_RT_QUEUE_EMPTY = 207013;	队列为空。	不能从空队列中获取数据，请先向队列中添加数据，再获取。
static const int32_t ACL_ERROR_RT_QUEUE_FULL = 207014;	队列已满。	不能向已满的队列中添加数据，请先从队列中获取数据，再添加。

返回码	含义	可能原因及解决方法
static const int32_t ACL_ERROR_RT_REPEATED_INIT = 207015;	队列重复初始化。	建议初始化一次队列即可，不要重复初始化。
static const int32_t ACL_ERROR_RT_DEVIDE_OOM = 207018;	Device侧内存耗尽。	排查Device上的内存使用情况，并根据Device上的内存规格合理规划内存的使用。
static const int32_t ACL_ERROR_RT_INTERNAL_ERROR = 507000;	runtime模块内部错误。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_TS_ERROR = 507001;	Device上的task scheduler模块内部错误。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_STREAM_TASK_FULL = 507002;	stream上的task数量满。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_STREAM_TASK_EMPTY = 507003;	stream上的task数量为空。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_STREAM_NOT_COMPLETE = 507004;	stream上的task未全部执行完成。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_END_OF_SEQUENCE = 507005;	AI CPU上的task执行完成。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_EVENT_NOT_COMPLETE = 507006;	event未完成。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_CONTEXT_RELEASE_ERROR = 507007;	context释放失败。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_SOC_VERSION = 507008;	获取soc version失败。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_TASK_TYPE_NOT_SUPPORT = 507009;	不支持的task类型。	您可以获取日志后单击**Link**联系技术支持。

返回码	含义	可能原因及解决方法
static const int32_t ACL_ERROR_RT_LOST_HEARTBEAT = 507010;	task scheduler丢失心跳。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_MODEL_EXECUTE = 507011;	模型执行失败。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_REPORT_TIMEOUT = 507012;	获取task scheduler的消息失败。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_SYS_DMA = 507013;	system dma（Direct Memory Access）硬件执行错误。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_AICORE_TIMEOUT = 507014;	aicore执行超时。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_AICORE_EXCEPTION = 507015;	aicore执行异常。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_AICORE_TRAP_EXCEPTION = 507016;	aicore trap执行异常。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_AICPU_TIMEOUT = 507017;	AI CPU执行超时。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_AICPU_EXCEPTION = 507018;	AI CPU执行异常。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_AICPU_DATADUMP_RSP_ERR = 507019;	AI CPU执行数据dump后未给task scheduler返回响应。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_AICPU_MODEL_RSP_ERR = 507020;	AI CPU执行模型后未给task scheduler返回响应。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_PROFILING_ERROR = 507021;	profiling功能执行异常。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_IPC_ERROR = 507022;	进程间通信异常。	您可以获取日志后单击**Link**联系技术支持。

返回码	含义	可能原因及解决方法
static const int32_t ACL_ERROR_RT_MODEL_ABORT_NORMAL = 507023;	模型退出。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_KERNEL_UNREGISTERING = 507024;	算子正在去注册。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_RINGBUFFER_NOT_INIT = 507025;	ringbuffer（环形缓冲区）功能未初始化。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_RINGBUFFER_NO_DATA = 507026;	ringbuffer（环形缓冲区）没有数据。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_KERNEL_LOOKUP = 507027;	RUNTIME内部的kernel未注册。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_KERNEL_DUPLICATE = 507028;	重复注册RUNTIME内部的kernel。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_DEBUG_REGISTER_FAIL = 507029;	debug功能注册失败。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_DEBUG_UNREGISTER_FAIL = 507030;	debug功能去注册失败。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_LABEL_CONTEXT = 507031;	标签不在当前context内。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_PROGRAM_USE_OUT = 507032;	注册的program数量超过限制。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_DEV_SETUP_ERROR = 507033;	Device启动失败。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_VECTOR_CORE_TIMEOUT = 507034;	vector core执行超时。	您可以获取日志后单击**Link**联系技术支持。

返回码	含义	可能原因及解决方法
static const int32_t ACL_ERROR_RT_VECTOR_CORE_EXCEPTION = 507035;	vector core执行异常。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_VECTOR_CORE_TRAP_EXCEPTION = 507036;	vector core trap执行异常。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_CDQ_BATCH_ABNORMAL = 507037;	Runtime内部资源申请异常。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_DIE_MODE_CHANGE_ERROR = 507038;	die模式修改异常，不能修改die模式。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_DIE_SET_ERROR = 507039;	单die模式不能指定die。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_INVALID_DIEID = 507040;	指定die id错误。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_DIE_MODE_NOT_SET = 507041;	die模式没有设置。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_AICORE_TRAP_READ_OVERFLOW = 507042;	aicore trap读越界异常。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_AICORE_TRAP_WRITE_OVERFLOW = 507043;	aicore trap写越界异常。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_VECTOR_CORE_TRAP_READ_OVERFLOW = 507044;	vector core trap读越界异常。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_VECTOR_CORE_TRAP_WRITE_OVERFLOW = 507045;	vector core trap写越界异常。	您可以获取日志后单击**Link**联系技术支持。

返回码	含义	可能原因及解决方法
static const int32_t ACL_ERROR_RT_STREAM_SYNC_TIMEOUT = 507046;	在指定的超时等待事件中，指定的stream中所有任务还没有执行完成。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_EVENT_SYNC_TIMEOUT = 507047;	在指定的Event同步等待中，超过指定时间，该Event还有没有执行完。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_FFTS_PLUS_TIMEOUT = 507048;	内部任务执行超时。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_FFTS_PLUS_EXCEPTION = 507049;	内部任务执行异常。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_FFTS_PLUS_TRAP_EXCEPTION = 507050;	内部任务trap异常。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_SEND_MSG = 507051;	数据入队过程中消息发送失败 。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_COPY_DATA = 507052;	数据入队过程中内存拷贝失败。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_DRV_INTERNAL_ERROR = 507899;	Driver模块内部错误。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_AICPU_INTERNAL_ERROR = 507900;	AI CPU模块内部错误。	您可以获取日志后单击**Link**联系技术支持。
static const int32_t ACL_ERROR_RT_SOCKET_CLOSE = 507901;	内部HDC（Host Device Communication）会话链接断开。	您可以获取日志后单击**Link**联系技术支持。

表 11-79 内部 GE 的返回码列表

返回码	含义	可能原因及解决方法
uint32_t ACL_ERROR_GE_PARAM_INVALID = 145000;	参数校验失败。	请检查接口的入参值是否正确。

返回码	含义	可能原因及解决方法
uint32_t ACL_ERROR_GE_EXEC_NOT_INIT = 145001;	未初始化。	• 请检查是否已调用**aclInit**接口进行初始化，请确保已调用**aclInit**接口，且在其它AscendCL接口之前调用。 • 请检查是否已调用对应功能的初始化接口，例如初始化Dump的**aclmdlInitDump**接口、初始化Profiling的**aclprofInit**接口。
uint32_t ACL_ERROR_GE_EXEC_MODEL_PATH_INVALID = 145002;	无效的模型路径。	请检查模型路径是否正确。
uint32_t ACL_ERROR_GE_EXEC_MODEL_ID_INVALID = 145003;	无效的模型ID。	请检查模型ID是否正确、模型是否正确加载。
uint32_t ACL_ERROR_GE_EXEC_MODEL_DATA_SIZE_INVALID = 145006;	无效的模型大小。	无效的模型大小。
uint32_t ACL_ERROR_GE_EXEC_MODEL_ADDR_INVALID = 145007;	无效的模型内存地址。	请检查模型地址是否有效。
uint32_t ACL_ERROR_GE_EXEC_MODEL_QUEUE_ID_INVALID = 145008;	无效的队列ID。	无效的队列ID。
uint32_t ACL_ERROR_GE_EXEC_LOAD_MODEL_REPEATED = 145009;	重复初始化或重复加载。	请检查是否调用对应的接口重复初始化或重复加载。
uint32_t ACL_ERROR_GE_DYNAMIC_INPUT_ADDR_INVALID = 145011;	无效的动态分档输入地址。	请检查动态分档输入地址。
uint32_t ACL_ERROR_GE_DYNAMIC_INPUT_LENGTH_INVALID = 145012;	无效的动态分档输入长度。	请检查动态分档输入长度。

返回码	含义	可能原因及解决方法
uint32_t ACL_ERROR_GE_DYNAMIC_BATCH_SIZE_INVALID = 145013;	无效的动态分档Batch大小。	请检查动态分档Batch大小。
uint32_t ACL_ERROR_GE_AIPP_BATCH_EMPTY = 145014;	无效的AIPP batch size。	请检查AIPP batch size是否正确。
uint32_t ACL_ERROR_GE_AIPP_NOT_EXIST = 145015;	AIPP配置不存在。	请检查AIPP是否配置。
uint32_t ACL_ERROR_GE_AIPP_MODE_INVALID = 145016;	无效的AIPP模式。	请检查模型转换时配置的AIPP模式是否正确。
uint32_t ACL_ERROR_GE_OP_TASK_TYPE_INVALID = 145017;	无效的任务类型。	请检查算子类型是否正确。
uint32_t ACL_ERROR_GE_OP_KERNEL_TYPE_INVALID = 145018;	无效的算子类型。	请检查算子类型是否正确。
uint32_t ACL_ERROR_GE_PLGMGR_PATH_INVALID = 145019;	无效的so文件，包括so文件的路径层级太深、so文件被误删除等情况。	请检查运行应用前配置的环境变量LD_LIBRARY_PATH是否正确，详细描述请参见编译运行处的操作指导。
uint32_t ACL_ERROR_GE_FORMAT_INVALID = 145020;	无效的format。	请检查Tensor数据的format是否有效。
uint32_t ACL_ERROR_GE_SHAPE_INVALID = 145021;	无效的shape。	请检查Tensor数据的shape是否有效。
uint32_t ACL_ERROR_GE_DATATYPE_INVALID = 145022;	无效的数据类型。	请检查Tensor数据的数据类型是否有效。
uint32_t ACL_ERROR_GE_MEMORY_ALLOCATION = 245000;	申请内存失败。	请检查硬件环境上的内存剩余情况。
uint32_t ACL_ERROR_GE_MEMORY_OPERATE_FAILED = 245001;	内存初始化、内存复制操作失败。	请检查内存地址是否正确、硬件环境上的内存是否足够等。

返回码	含义	可能原因及解决方法
uint32_t ACL_ERROR_GE_DEVICE_MEMORY_ALLOCATION_FAILED = 245002;	申请Device内存失败。	Device内存已用完，无法继续申请，请释放部分Device内存，再重新尝试。
uint32_t ACL_ERROR_GE_INTERNAL_ERROR = 545000;	未知内部错误。	您可以获取日志后单击**Link**联系技术支持。
uint32_t ACL_ERROR_GE_LOAD_MODEL = 545001;	系统内部加载模型失败。	您可以获取日志后单击**Link**联系技术支持。
uint32_t ACL_ERROR_GE_EXEC_LOAD_MODEL_PARTITION_FAILED = 545002;	系统内部加载模型失败。	您可以获取日志后单击**Link**联系技术支持。
uint32_t ACL_ERROR_GE_EXEC_LOAD_WEIGHT_PARTITION_FAILED = 545003;	系统内部加载模型权值失败。	您可以获取日志后单击**Link**联系技术支持。
uint32_t ACL_ERROR_GE_EXEC_LOAD_TASK_PARTITION_FAILED = 545004;	系统内部加载模型任务失败。	您可以获取日志后单击**Link**联系技术支持。
uint32_t ACL_ERROR_GE_EXEC_LOAD_KERNEL_PARTITION_FAILED = 545005;	系统内部加载模型算子失败。	您可以获取日志后单击**Link**联系技术支持。
uint32_t ACL_ERROR_GE_EXEC_RELEASE_MODEL_DATA = 545006;	系统内释放模型空间失败。	您可以获取日志后单击**Link**联系技术支持。
uint32_t ACL_ERROR_GE_COMMAND_HANDLE = 545007;	系统内命令操作失败。	您可以获取日志后单击**Link**联系技术支持。
uint32_t ACL_ERROR_GE_GET_TENSOR_INFO = 545008;	系统内获取张量数据失败。	您可以获取日志后单击**Link**联系技术支持。
uint32_t ACL_ERROR_GE_UNLOAD_MODEL = 545009;	系统内卸载模型空间失败。	您可以获取日志后单击**Link**联系技术支持。

11.27.2 aclDataType

```
typedef enum {
    ACL_DT_UNDEFINED = -1, //未知数据类型，默认值
    ACL_FLOAT = 0,
    ACL_FLOAT16 = 1,
    ACL_INT8 = 2,
    ACL_INT32 = 3,
    ACL_UINT8 = 4,
    ACL_INT16 = 6,
    ACL_UINT16 = 7,
    ACL_UINT32 = 8,
    ACL_INT64 = 9,
    ACL_UINT64 = 10,
    ACL_DOUBLE = 11,
    ACL_BOOL = 12,
    ACL_STRING = 13,
    ACL_COMPLEX64 = 16,
    ACL_COMPLEX128 = 17,
    ACL_BF16 = 27,
    ACL_UINT1 = 30,
    ACL_COMPLEX32 = 33,
} aclDataType;
```

11.27.3 aclFloat16

该数据类型仅用于数据传递，不能用于加减乘除等运算。

```
typedef uint16_t aclFloat16;
```

11.27.4 aclFormat

```
typedef enum {
    ACL_FORMAT_UNDEFINED = -1,
    ACL_FORMAT_NCHW = 0,
    ACL_FORMAT_NHWC = 1,
    ACL_FORMAT_ND = 2,
    ACL_FORMAT_NC1HWC0 = 3,
    ACL_FORMAT_FRACTAL_Z = 4,
    ACL_FORMAT_NC1HWC0_C04 = 12,
    ACL_FORMAT_HWCN = 16,
    ACL_FORMAT_NDHWC = 27,
    ACL_FORMAT_FRACTAL_NZ = 29,
    ACL_FORMAT_NCDHW = 30,
    ACL_FORMAT_NDC1HWC0 = 32,
    ACL_FRACTAL_Z_3D = 33
} aclFormat;
```

- UNDEFINED：未知格式，默认值。
- NCHW：NCHW格式。
- NHWC：NHWC格式。
- ND：表示支持任意格式，仅有Square、Tanh等这些单输入对自身处理的算子外，其它需要慎用。
- NC1HWC0：5维数据格式。其中，C0与微架构强相关，该值等于cube单元的size，例如16；C1是将C维度按照C0切分：C1=C/C0， 若结果不整除，最后一份数据需要padding到C0。
- FRACTAL_Z：卷积的权重的格式。
- NC1HWC0_C04：5维数据格式。其中，C0固定为4，C1是将C维度按照C0切分：C1=C/C0， 若结果不整除，最后一份数据需要padding到C0。当前版本不支持。
- HWCN：HWCN格式。

- NDHWC：NDHWC格式。对于3维图像就需要使用带D（Depth）维度的格式。
- FRACTAL_NZ：内部格式，用户目前无需使用。
- NCDHW：NCDHW格式。对于3维图像就需要使用带D（Depth）维度的格式。
- NDC1HWC0：6维数据格式。相比于NC1HWC0，仅多了D（Depth）维度。
- FRACTAL_Z_3D：3D卷积权重格式，例如Conv3D/MaxPool3D/AvgPool3D这些算子都需要这种格式来表达。

说明

各维度的含义如下：N（Batch）表示批量大小、H（Height）表示特征图高度、W（Width）表示特征图宽度、C（Channels）表示特征图通道、D（Depth）表示特征图深度。

11.27.5 aclrtMemMallocPolicy

```
typedef enum aclrtMemMallocPolicy {
    ACL_MEM_MALLOC_HUGE_FIRST,
    ACL_MEM_MALLOC_HUGE_ONLY,
    ACL_MEM_MALLOC_NORMAL_ONLY,
    ACL_MEM_MALLOC_HUGE_FIRST_P2P,
    ACL_MEM_MALLOC_HUGE_ONLY_P2P,
    ACL_MEM_MALLOC_NORMAL_ONLY_P2P,
    ACL_MEM_TYPE_LOW_BAND_WIDTH = 0x0100,
    ACL_MEM_TYPE_HIGH_BAND_WIDTH = 0x1000
} aclrtMemMallocPolicy;
```

此处支持单个枚举项，也支持多个枚举项位或：

- **配置单个枚举项**：

  若配置ACL_MEM_MALLOC_HUGE_FIRST、或ACL_MEM_MALLOC_HUGE_ONLY、或ACL_MEM_MALLOC_NORMAL_ONLY、或ACL_MEM_MALLOC_HUGE_FIRST_P2P、或ACL_MEM_MALLOC_HUGE_ONLY_P2P、或ACL_MEM_MALLOC_NORMAL_ONLY_P2P，则系统内部会根据硬件支持情况选择从高带宽或低带宽物理内存申请内存；

  若配置ACL_MEM_TYPE_LOW_BAND_WIDTH或ACL_MEM_TYPE_HIGH_BAND_WIDTH，则系统内部会默认采取ACL_MEM_MALLOC_HUGE_FIRST，优先申请大页。

- **配置多个枚举项位或**：支持这三项（ACL_MEM_MALLOC_HUGE_FIRST、ACL_MEM_MALLOC_HUGE_ONLY、ACL_MEM_MALLOC_NORMAL_ONLY）与这两项（ACL_MEM_TYPE_LOW_BAND_WIDTH、ACL_MEM_TYPE_HIGH_BAND_WIDTH）组合，**例如**：ACL_MEM_MALLOC_HUGE_FIRST | ACL_MEM_TYPE_HIGH_BAND_WIDTH

表 11-80 枚举项说明

枚举项	说明
ACL_MEM_MALLOC_HUGE_FIRST	当申请的内存小于等于1M时，即使使用该内存分配规则，也是申请普通页的内存。当申请的内存大于1M时，优先申请大页内存，如果大页内存不够，则使用普通页的内存。
ACL_MEM_MALLOC_HUGE_ONLY	仅申请大页，如果大页内存不够，则返回错误。

枚举项	说明
ACL_MEM_MALLOC_NORMAL_ONLY	仅申请普通页，如果普通页内存不够，则返回错误。
ACL_MEM_MALLOC_HUGE_FIRST_P2P	仅Device之间内存复制场景下申请内存时使用该选项，表示优先申请大页内存，如果大页内存不够，则使用普通页的内存。 Atlas 200/300/500 推理产品，不支持该选项。
ACL_MEM_MALLOC_HUGE_ONLY_P2P	仅Device之间内存复制场景下申请内存时使用该选项，仅申请大页内存，如果大页内存不够，则返回错误。 Atlas 200/300/500 推理产品，不支持该选项。
ACL_MEM_MALLOC_NORMAL_ONLY_P2P	仅Device之间内存复制场景下申请内存时使用该选项，仅申请普通页的内存。 Atlas 200/300/500 推理产品，不支持该选项。
ACL_MEM_TYPE_LOW_BAND_WIDTH	从带宽低的物理内存上申请内存。 设置该选项无效，系统默认会根据硬件支持的内存类型选择。
ACL_MEM_TYPE_HIGH_BAND_WIDTH	从带宽高的物理内存上申请内存。 设置该选项无效，系统默认会根据硬件支持的内存类型选择。

11.27.6 acldvppPixelFormat

```
// Supported Pixel Format
enum acldvppPixelFormat {
    PIXEL_FORMAT_YUV_400 = 0,            // 0, YUV400 8bit
    PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // 1, YUV420SP NV12 8bit
    PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // 2, YUV420SP NV21 8bit
    PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3, // 3, YUV422SP 8bit
    PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4, // 4, YVU422SP 8bit
    PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5, // 5, YUV444SP 8bit
    PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6, // 6, YVU444SP 8bit
    PIXEL_FORMAT_YUYV_PACKED_422 = 7,    // 7, YUV422P YUYV 8bit
    PIXEL_FORMAT_UYVY_PACKED_422 = 8,    // 8, YUV422P UYVY 8bit
    PIXEL_FORMAT_YVYU_PACKED_422 = 9,    // 9, YUV422P YVYU 8bit
    PIXEL_FORMAT_VYUY_PACKED_422 = 10,   // 10, YUV422P VYUY 8bit
    PIXEL_FORMAT_YUV_PACKED_444 = 11,    // 11, YUV444P 8bit
    PIXEL_FORMAT_RGB_888 = 12,           // 12, RGB888
    PIXEL_FORMAT_BGR_888 = 13,           // 13, BGR888
    PIXEL_FORMAT_ARGB_8888 = 14,         // 14, ARGB8888
    PIXEL_FORMAT_ABGR_8888 = 15,         // 15, ABGR8888
    PIXEL_FORMAT_RGBA_8888 = 16,         // 16, RGBA8888
    PIXEL_FORMAT_BGRA_8888 = 17,         // 17, BGRA8888
    PIXEL_FORMAT_YUV_SEMI_PLANNER_420_10BIT = 18, // 18, YUV420SP 10bit
    PIXEL_FORMAT_YVU_SEMI_PLANNER_420_10BIT = 19, // 19, YVU420S 10bit
    PIXEL_FORMAT_YVU_PLANAR_420 = 20,        // 20, YVU420P 8bit
    PIXEL_FORMAT_YVU_PLANAR_422,             // YVU422P 8bit
    PIXEL_FORMAT_YVU_PLANAR_444,             // YVU444P 8bit
    PIXEL_FORMAT_RGB_444 = 23,               // RGB444  R:4bit G:4bit B:4bit，当前不支持该格式
    PIXEL_FORMAT_BGR_444,                    // BGR444  R:4bit G:4bit B:4bit，当前不支持该格式
    PIXEL_FORMAT_ARGB_4444,                  // ARGB4444 A:4bit R:4bit G:4bit B:4bit，当前不支持该格式
    PIXEL_FORMAT_ABGR_4444,                  // ABGR4444 A:4bit B:4bit G:4bit R:4bit，当前不支持该格式
    PIXEL_FORMAT_RGBA_4444,                  // RGBA4444 R:4bit G:4bit B:4bit A:4bit，当前不支持该格式
    PIXEL_FORMAT_BGRA_4444,                  // BGRA4444 B:4bit G:4bit R:4bit A:4bit，当前不支持该格式
    PIXEL_FORMAT_RGB_555,                    // RGB555 R:5bit G:5bit B:5bit，当前不支持该格式
    PIXEL_FORMAT_BGR_555,                    // BGR555 B:5bit G:5bit R:5bit，当前不支持该格式
```

```
    PIXEL_FORMAT_RGB_565,                // RGB565 R:5bit G:6bit B:5bit，当前不支持该格式
    PIXEL_FORMAT_BGR_565,                // BGR565 B:5bit G:6bit R:5bit，当前不支持该格式
    PIXEL_FORMAT_ARGB_1555,              // ARGB1555 A:1bit R:5bit G:6bit B:5bit，当前不支持该格式
    PIXEL_FORMAT_ABGR_1555,              // ABGR1555 A:1bit B:5bit G:6bit R:5bit，当前不支持该格式
    PIXEL_FORMAT_RGBA_1555,              // RGBA1555 A:1bit B:5bit G:6bit R:5bit，当前不支持该格式
    PIXEL_FORMAT_BGRA_1555,              // BGRA1555 A:1bit B:5bit G:6bit R:5bit，当前不支持该格式
    PIXEL_FORMAT_ARGB_8565,              // ARGB8565 A:8bit R:5bit G:6bit B:5bit，当前不支持该格式
    PIXEL_FORMAT_ABGR_8565,              // ABGR8565 A:8bit B:5bit G:6bit R:5bit，当前不支持该格式
    PIXEL_FORMAT_RGBA_8565,              // RGBA8565 A:8bit R:5bit G:6bit B:5bit，当前不支持该格式
    PIXEL_FORMAT_BGRA_8565,              // BGRA8565 A:8bit B:5bit G:6bit R:5bit，当前不支持该格式
    PIXEL_FORMAT_RGB_BAYER_8BPP = 50,      // RGB Bayer 8bit，Bayer图像，当前不支持该格式
    PIXEL_FORMAT_RGB_BAYER_10BPP,          // RGB Bayer 10bit，Bayer图像，当前不支持该格式
    PIXEL_FORMAT_RGB_BAYER_12BPP,          // RGB Bayer 12bit，Bayer图像，当前不支持该格式
    PIXEL_FORMAT_RGB_BAYER_14BPP,          // RGB Bayer 14bit，Bayer图像，当前不支持该格式
    PIXEL_FORMAT_RGB_BAYER_16BPP,          // RGB Bayer 16bit，Bayer图像，当前不支持该格式
    PIXEL_FORMAT_BGR_888_PLANAR = 70,      // RGB888 Planar，当前不支持该格式
    PIXEL_FORMAT_HSV_888_PACKAGE,          // HSV Package，HSV图像package格式，当前不支持该格式
    PIXEL_FORMAT_HSV_888_PLANAR,           // HSV Planar，HSV图像Planar格式，当前不支持该格式
    PIXEL_FORMAT_LAB_888_PACKAGE,          // LAB Package，LAB图像package格式，当前不支持该格式
    PIXEL_FORMAT_LAB_888_PLANAR,           // LAB Planar，LAB图像Planar格式，当前不支持该格式
    PIXEL_FORMAT_S8C1,                   // Signed 8bit for 1pixel 1Channel，每个像素用1个8bit有符号数据表
示的单通道图像，当前不支持该格式
    PIXEL_FORMAT_S8C2_PACKAGE,             // Signed 8bit for 1pixel 2Channel Package，每个像素用2个8bit
有符号数表示的双通道图像Package格式，当前不支持该格式
    PIXEL_FORMAT_S8C2_PLANAR,              // Signed 8bit for 1pixel 2Channel Planar，每个像素用2个8bit有
符号数据表的双通道图像Planar格式，当前不支持该格式
    PIXEL_FORMAT_S16C1,                  // Signed 16bit 1pixel 1Channel，每个像素用1个16bit有符号数据表示
的单通道图像，当前不支持该格式
    PIXEL_FORMAT_U8C1,                   // Unsigned 8bit 1pixel 1Channel，每个像素用1个8bit无符号数据表示
的单通道图像，当前不支持该格式
    PIXEL_FORMAT_U16C1,                  // Unsigned 16bit 1pixel 1Channel，每个像素用1个16bit无符号数据
表示的单通道图像，当前不支持该格式
    PIXEL_FORMAT_S32C1,                  // Signed 32bit 1pixel 1Channel，每个像素用1个32bit有符号数据表示
的单通道图像，当前不支持该格式
    PIXEL_FORMAT_U32C1,                  // Unsigned 32bit 1pixel 1Channel，每个像素用1个32bit无符号数据
表示的单通道图像，当前不支持该格式
    PIXEL_FORMAT_U64C1,                  // Unsigned 64bit 1pixel 1Channel，每个像素用1个64bit无符号数据
表示的单通道图像，当前不支持该格式
    PIXEL_FORMAT_S64C1,                  // Signed 64bit 1pixel 1Channel，每个像素用1个64bit有符号数据表示
的单通道图像，当前不支持该格式
    PIXEL_FORMAT_YUV_SEMIPLANAR_440 = 1000, // YUV440SP 8bit
    PIXEL_FORMAT_YVU_SEMIPLANAR_440,       // YVU440SP 8bit
    PIXEL_FORMAT_FLOAT32,                // Float 32bit for 1pixel，每个像素用1个float32数据表示，当前不支
持该格式
    PIXEL_FORMAT_BUTT,
    PIXEL_FORMAT_UNKNOWN = 10000
};
```

11.27.7 acldvppStreamFormat

```
enum acldvppStreamFormat {
    H265_MAIN_LEVEL = 0,
    H264_BASELINE_LEVEL,
    H264_MAIN_LEVEL,
    H264_HIGH_LEVEL
};
```

11.27.8 acldvppJpegFormat

```
enum acldvppJpegFormat {
    ACL_JPEG_CSS_444 = 0,
    ACL_JPEG_CSS_422,
    ACL_JPEG_CSS_420,
    ACL_JPEG_CSS_GRAY,
    ACL_JPEG_CSS_440,
    ACL_JPEG_CSS_411,
```

```
    ACL_JPEG_CSS_UNKNOWN = 1000
};
```

11.27.9 acldvppCscMatrix

Atlas 200/300/500 推理产品，不支持该枚举值。

```
enum acldvppCscMatrix {
    ACL_DVPP_CSC_MATRIX_BT601_WIDE = 0,  //基于BT601 wide标准的色域转换矩阵，默认值
    ACL_DVPP_CSC_MATRIX_BT601_NARROW,    //基于BT601 narrow标准的色域转换矩阵
    ACL_DVPP_CSC_MATRIX_BT709_WIDE,      //基于BT709 wide标准的色域转换矩阵
    ACL_DVPP_CSC_MATRIX_BT709_NARROW,    //基于BT709 narrow标准的色域转换矩阵
    ACL_DVPP_CSC_MATRIX_BT2020_WIDE,     //基于BT2020 wide标准的色域转换矩阵
    ACL_DVPP_CSC_MATRIX_BT2020_NARROW    //基于BT2020 narrow标准的色域转换矩阵
};
```

基于以上各标准的色域转换矩阵，其各分量的转换公式如下：

```
# YUV转RGB：
# | R |   | matrix_r0c0 matrix_r0c1 matrix_r0c2 |   | Y - input_bias_0 |
# | G | = | matrix_r1c0 matrix_r1c1 matrix_r1c2 | * | U - input_bias_1 |
# | B |   | matrix_r2c0 matrix_r2c1 matrix_r2c2 |   | V - input_bias_2 |
# BGR转YUV：
# | Y |   | output_bias_0 |   | matrix_r0c0 matrix_r0c1 matrix_r0c2 |   | R |
# | U | = | output_bias_1 | + | matrix_r1c0 matrix_r1c1 matrix_r1c2 | * | G |
# | V |   | output_bias_2 |   | matrix_r2c0 matrix_r2c1 matrix_r2c2 |   | B |
```

基于不同标准的色域转换矩阵，各参数值如下表所示。

成员名称	描述
ACL_DVPP_CSC_MATRIX_BT601_WIDE	基于BT601 wide标准的色域转换矩阵。默认为ACL_DVPP_CSC_MATRIX_BT601_WIDE 。 基于该标准的色域转换矩阵，各参数值如下： # YUV转RGB： # | R | | 1.000 0.000 1.402 | | Y - 0 | # | G | = | 1.000 -0.344 -0.714 | * | U - 128 | # | B | | 1.000 1.772 0.000 | | V - 128 | # RGB转YUV： # | Y | |-0.5 | | 0.299 0.587 0.114 | | R | # | U | = |127.5 | + | -0.168 -0.331 0.500 | * | G | # | V | |127.5 | | 0.500 -0.419 -0.081 | | B |
ACL_DVPP_CSC_MATRIX_BT601_NARROW	基于BT601 narrow标准的色域转换矩阵。 基于该标准的色域转换矩阵，各参数值如下： # YUV转RGB： # | R | | 1.16438 0.00000 1.59602 | | Y - 16 | # | G | = | 1.16438 -0.39176 -0.81297 | * | U - 128 | # | B | | 1.16438 2.01723 0.00000 | | V - 128 | # RGB转YUV： # | Y | |16 | | 0.25679 0.51564 0.10014 | | R | # | U | = |128| + | -0.14491 -0.29099 0.43922 | * | G | # | V | |128| | 0.42941 -0.36779 -0.07143 | | B |

成员名称	描述
ACL_DVPP_CSC_MATRIX_BT709_WIDE	基于BT709 wide标准的色域转换矩阵。 基于该标准的色域转换矩阵，各参数值如下： # YUV转RGB： # | R | | 1.00000 0.00000 1.57480 | | Y - 0 | # | G | = | 1.00000 -0.18732 -0.46812 | * | U - 128 | # | B | | 1.00000 1.85560 0.00000 | | V - 128 | # RGB转YUV： # | Y | |0 | | 0.21260 0.71520 0.07220 | | R | # | U | = |128| + | -0.11457 -0.38543 0.50000 | * | G | # | V | |128| | 0.50000 -0.45415 -0.04585 | | B |
ACL_DVPP_CSC_MATRIX_BT709_NARROW	基于BT709 narrow标准的色域转换矩阵。 基于该标准的色域转换矩阵，各参数值如下： # YUV转RGB： # | R | | 1.16438 0.00000 1.79274 | | Y - 16 | # | G | = | 1.16438 -0.21325 -0.53291 | * | U - 128 | # | B | | 1.16438 2.11240 0.00000 | | V - 128 | # RGB转YUV： # | Y | |16 | | 0.18259 0.62825 0.06342 | | R | # | U | = |128| + | -0.09840 -0.33857 0.43922 | * | G | # | V | |128| | 0.42941 -0.39894 -0.04027 | | B |
ACL_DVPP_CSC_MATRIX_BT2020_WIDE	基于BT2020 wide标准的色域转换矩阵。 基于该标准的色域转换矩阵，各参数值如下： # YUV转RGB： # | R | | 1.00000 0.00000 1.47460 | | Y - 0 | # | G | = | 1.00000 -0.16455 -0.57135 | * | U - 128 | # | B | | 1.00000 1.88140 0.00000 | | V - 128 | # RGB转YUV： # | Y | |0 | | 0.26270 0.67800 0.05930 | | R | # | U | = |128| + | -0.13963 -0.36037 0.50000 | * | G | # | V | |128| | 0.50000 -0.45979 -0.04021 | | B |
ACL_DVPP_CSC_MATRIX_BT2020_NARROW	基于BT2020 narrow标准的色域转换矩阵。 基于该标准的色域转换矩阵，各参数值如下： # YUV转RGB： # | R | | 1.16438 0.00000 1.67868 | | Y - 16 | # | G | = | 1.16438 -0.18733 -0.65042 | * | U - 128 | # | B | | 1.16438 2.14177 0.00000 | | V - 128 | # RGB转YUV： # | Y | |16 | | 0.22564 0.59558 0.05209 | | R | # | U | = |128| + | -0.11992 -0.31656 0.43922 | * | G | # | V | |128| | 0.42941 -0.40389 -0.03533 | | B |

11.27.10 aclrtContext

```
typedef void *aclrtContext;
```

11.27.11 aclrtStream

```
typedef void *aclrtStream;
```

11.27.12 aclrtEvent

```
typedef void *aclrtEvent;
```

11.27.13 aclrtEventStatus

```
typedef enum aclrtEventStatus {
    ACL_EVENT_STATUS_COMPLETE  = 0, //完成
    ACL_EVENT_STATUS_NOT_READY = 1, //未完成
    ACL_EVENT_STATUS_RESERVED  = 2, //预留
} aclrtEventStatus;
```

11.27.14 aclrtRunMode

```
typedef enum aclrtRunMode {
    ACL_DEVICE,   //昇腾AI软件栈运行在Device的Control CPU或板端环境上
    ACL_HOST,     //昇腾AI软件栈运行在Host侧
} aclrtRunMode;
```

11.27.15 aclTransType

```
typedef enum aclTransType
{
    ACL_TRANS_N,    //不转置，默认为不转置
    ACL_TRANS_T,    //转置
    ACL_TRANS_NZ,   //内部格式，能够提供更好的接口性能，建议使用场景为：某个输入矩阵作为底库，执行一次转NZ格式的操作后，多次使用NZ格式的算子，提升性能。
    ACL_TRANS_NZ_T  //内部格式转置，当前不支持
}aclTransType;
```

11.27.16 aclComputeType

```
typedef enum aclComputeType
{
    ACL_COMPUTE_HIGH_PRECISION,
    ACL_COMPUTE_LOW_PRECISION    //当前不支持
}aclComputeType;
```

11.27.17 aclfvSearchType

Atlas 200/300/500 推理产品，不支持该枚举值。

```
enum aclfvSearchType {
    SEARCH_1_N,  // 1:N operation type
    SEARCH_N_M   // N:M operation type
};
```

11.27.18 aclAippInputFormat

静态AIPP和动态AIPP支持的图片格式不同，详情如下：

```
typedef enum
{
    ACL_YUV420SP_U8 = 1,  //YUV420SP_U8，静态AIPP和动态AIPP都支持
    ACL_XRGB8888_U8=2,    //XRGB8888_U8，静态AIPP和动态AIPP都支持
    ACL_RGB888_U8=3,      //RGB888_U8，静态AIPP和动态AIPP都支持
    ACL_YUV400_U8=4,      //YUV400_U8，静态AIPP和动态AIPP都支持
    ACL_NC1HWC0DI_FP16 = 5;//暂不支持
    ACL_NC1HWC0DI_S8 = 6;//暂不支持
    ACL_ARGB8888_U8 = 7;   //暂不支持
    ACL_YUYV_U8 = 8;     //暂不支持
    ACL_YUV422SP_U8 = 9;  //暂不支持
    ACL_AYUV444_U8 = 10;  // 暂不支持
    ACL_RAW10 = 11;  //暂不支持
    ACL_RAW12 = 12;  //暂不支持
    ACL_RAW16 = 13;  //暂不支持
    ACL_RAW24 = 14;  //暂不支持
```

```
    ACL_AIPP_RESERVED = 0xffff,
} aclAippInputFormat;
```

11.27.19 aclAippInfo

```
typedef struct aclAippInfo {
    aclAippInputFormat inputFormat;
    int32_t srcImageSizeW;
    int32_t srcImageSizeH;
    int8_t cropSwitch;
    int32_t loadStartPosW;
    int32_t loadStartPosH;
    int32_t cropSizeW;
    int32_t cropSizeH;
    int8_t resizeSwitch;
    int32_t resizeOutputW;
    int32_t resizeOutputH;
    int8_t paddingSwitch;
    int32_t leftPaddingSize;
    int32_t rightPaddingSize;
    int32_t topPaddingSize;
    int32_t bottomPaddingSize;
    int8_t cscSwitch;
    int8_t rbuvSwapSwitch;
    int8_t axSwapSwitch;
    int8_t singleLineMode;
    int32_t matrixR0C0;
    int32_t matrixR0C1;
    int32_t matrixR0C2;
    int32_t matrixR1C0;
    int32_t matrixR1C1;
    int32_t matrixR1C2;
    int32_t matrixR2C0;
    int32_t matrixR2C1;
    int32_t matrixR2C2;
    int32_t outputBias0;
    int32_t outputBias1;
    int32_t outputBias2;
    int32_t inputBias0;
    int32_t inputBias1;
    int32_t inputBias2;
    int32_t meanChn0;
    int32_t meanChn1;
    int32_t meanChn2;
    int32_t meanChn3;
    float minChn0;
    float minChn1;
    float minChn2;
    float minChn3;
    float varReciChn0;
    float varReciChn1;
    float varReciChn2;
    float varReciChn3;
    aclFormat srcFormat;      //模型转换前，原始模型的输入format
    aclDataType srcDatatype;  //模型转换前，原始模型的输入datatype
    size_t srcDimNum;         //模型转换前，原始模型输入dims
    size_t shapeCount;        //动态Shape（动态Batch或动态分辨率）场景下的档位数
    aclAippDims outDims[ACL_MAX_SHAPE_COUNT]; //在多shape模型下，各个分档对应的aipp输出dims信
息，需要在模型编译阶段按照shape推导出来，保存在模型中，输出的Format，datatype、输出dims等不随
shape变化而变化
    aclAippExtendInfo *aippExtend; //预留参数，当前版本用户必须传入空指针
} aclAippInfo;
```

11.27.20 aclAippDims

```
typedef struct aclAippDims {
    aclmdlIODims srcDims;     //模型转换前的dims，如[1,3,150,150]
```

```
    size_t srcSize;            //模型转换前输入数据的大小
    aclmdlIODims aippOutdims; //内部数据格式的dims信息
    size_t aippOutSize;        //经过AIPP处理后的输出数据的大小
} aclAippDims;
```

11.27.21 aclmdlIODims

```
const int ACL_MAX_DIM_CNT = 128;
const int ACL_MAX_TENSOR_NAME_LEN = 128;
typedef struct aclmdlIODims {
    char name[ACL_MAX_TENSOR_NAME_LEN]; /**< tensor name */
    size_t dimCount;  /**Shape中的维度个数，如果为标量，则维度个数为0*/
    int64_t dims[ACL_MAX_DIM_CNT]; /**< 维度信息 */
} aclmdlIODims;
```

11.27.22 aclrtGroupAttr

Atlas 200/300/500 推理产品，不支持该枚举值。

Atlas 推理系列产品，不支持该枚举值。

```
typedef enum aclrtGroupAttr
{
    ACL_GROUP_AICORE_INT,     //指定Group对应的aicore个数，属性值的数据类型为整型。
    ACL_GROUP_AIV_INT,        //指定Group对应的vector core个数，属性值的数据类型为整型。当前不支持。
    ACL_GROUP_AIC_INT,        //指定Group对应的aicpu线程数，属性值的数据类型为整型。当前不支持。
    ACL_GROUP_SDMANUM_INT,   //内存异步拷贝的通道数，属性值的数据类型为整型。当前不支持。
    ACL_GROUP_ASQNUM_INT,    //指定Group下可以被同时调度执行的stream个数，小于或等于32，当前系统级最大一共可以同时调度32个stream。属性值的数据类型为整型。当前不支持。
    ACL_GROUP_GROUPID_INT    //指定Group的ID。属性值的数据类型为整型。
} aclrtGroupAttr;
```

11.27.23 aclprofAicoreMetrics

```
typedef enum {
    ACL_AICORE_ARITHMETIC_UTILIZATION = 0,  //各种计算类指标占比统计，包括采集项mac_fp16_ratio、mac_int8_ratio、vec_fp32_ratio、vec_fp16_ratio、vec_int32_ratio、vec_misc_ratio
    ACL_AICORE_PIPE_UTILIZATION = 1,        //计算单元和搬运单元耗时占比，包括采集项vec_ratio、mac_ratio、scalar_ratio、mte1_ratio、mte2_ratio、mte3_ratio、icache_miss_rate（Atlas A2训练系列产品）
    ACL_AICORE_MEMORY_BANDWIDTH = 2,        //外部内存读写类指令占比，包括采集项ub_read_bw、ub_write_bw、l1_read_bw、l1_write_bw、l2_read_bw（Atlas 200/300/500 推理产品）、l2_write_bw（Atlas 200/300/500 推理产品）、main_mem_read_bw、main_mem_write_bw
    ACL_AICORE_L0B_AND_WIDTH = 3,           //内部内存读写类指令占比，包括采集项scalar_ld_ratio、scalar_st_ratio、l0a_read_bw、l0a_write_bw、l0b_read_bw、l0b_write_bw、l0c_read_bw、l0c_write_bw、l0c_read_bw_cube、l0c_write_bw_cube
    ACL_AICORE_RESOURCE_CONFLICT_RATIO = 4,    //流水线队列类指令占比，包括采集项vec_bankgroup_cflt_ratio、vec_bank_cflt_ratio、vec_resc_cflt_ratio、mte1_iq_full_ratio、mte2_iq_full_ratio、mte3_iq_full_ratio、cube_iq_full_ratio、vec_iq_full_ratio、iq_full_ratio
    ACL_AICORE_MEMORY_UB = 5,               //内部内存读写指令占比，包括采集项ub_read_bw_mte（Atlas 200/300/500 推理产品）、ub_write_bw_mte（Atlas 200/300/500 推理产品）、ub_read_bw_vector、ub_write_bw_vector、ub_read_bw_scalar、ub_write_bw_scalar
    ACL_AICORE_NONE = 0xFF
}aclprofAicoreMetrics;
```

11.27.24 acldvppBorderType

Atlas 200/300/500 推理产品，当前版本不支持该枚举值。

```
enum acldvppBorderType {
    BORDER_CONSTANT = 0, //添加有颜色的常数值边界
    BORDER_REPLICATE,    //重复最后一个元素。举例: aaaaaa|abcdefgh|hhhhhhh
    BORDER_REFLECT,      //边界元素的镜像。举例：cba|abcde-fgh|hgf
    BORDER_REFLECT_101   //边界元素的镜像。举例：cba|abcde-fgh|gf
};
```

11.27.25 aclmdlInputAippType

```
typedef enum {
    ACL_DATA_WITHOUT_AIPP = 0,  //该输入无AIPP信息
    ACL_DATA_WITH_STATIC_AIPP,  //该输入带静态AIPP信息
    ACL_DATA_WITH_DYNAMIC_AIPP, //该输入有关联的动态AIPP输入
    ACL_DYNAMIC_AIPP_NODE       //该输入本身就是动态AIPP输入，用于配置动态AIPP参数
} aclmdlInputAippType;
```

11.27.26 acltdtTensorType

Atlas 200/300/500 推理产品，当前版本不支持该枚举值。

Atlas 推理系列产品，当前版本不支持该枚举值。

```
enum acltdtTensorType {
    ACL_TENSOR_DATA_UNDEFINED = -1,
    ACL_TENSOR_DATA_TENSOR,
    ACL_TENSOR_DATA_END_OF_SEQUENCE,
    ACL_TENSOR_DATA_ABNORMAL
};
```

11.27.27 aclCompileOpt

```
typedef enum {
    ACL_PRECISION_MODE,           // 算子精度模式
    ACL_AICORE_NUM,               // 模型编译时使用的AI Core数量
    ACL_AUTO_TUNE_MODE,           // 算子的自动调优模式
    ACL_OP_SELECT_IMPL_MODE,      // 选择算子是高精度实现还是高性能实现
    ACL_OPTYPELIST_FOR_IMPLMODE,  // 列举算子类型的列表，该列表中的算子使用
ACL_OP_SELECT_IMPL_MODE指定的模式
    ACL_OP_DEBUG_LEVEL,           // TBE算子编译debug功能开关
    ACL_DEBUG_DIR,                // 保存模型转换、网络迁移过程中算子编译生成的调试相关过程文件的路径，包
括算子.o/.json/.cce等文件。
    ACL_OP_COMPILER_CACHE_MODE,   // 算子编译磁盘缓存模式
    ACL_OP_COMPILER_CACHE_DIR,    // 算子编译磁盘缓存的目录
    ACL_OP_PERFORMANCE_MODE,      // 通过该选项设置是否按照算子执行高性能的方式编译算子
    ACL_OP_JIT_COMPILE,           // 选择是在线编译算子，还是使用已编译的算子二进制文件
    ACL_OP_DETERMINISTIC,         // 是否开启确定性计算
    ACL_CUSTOMIZE_DTYPES,         // 模型编译时自定义某个或某些算子的计算精度
    ACL_OP_PRECISION_MODE,        // 指定算子内部处理时的精度模式，支持指定一个算子或多个算子。
    ACL_ALLOW_HF32                // hf32是昇腾推出的专门用于算子内部计算的精度类型，当前版本不支持
} aclCompileOpt;
```

表 11-81 编译选项配置

编译选项	取值说明
ACL_PRECISION_MODE	用于配置算子精度模式。如果不配置该编译选项，默认采用allow_fp32_to_fp16。 ● **force_fp32/cube_fp16in_fp32out：** 配置为force_fp32或cube_fp16in_fp32out，效果等同，系统内部都会根据矩阵类算子或矢量类算子，来选择不同的处理方式。cube_fp16in_fp32out为新版本中新增的，对于矩阵计算类算子，该选项语义更清晰。 – 对于矩阵计算类算子，系统内部会按算子实现的支持情况处理： 1. 优先选择输入数据类型为float16且输出数据类型为float32； 2. 如果1中的场景不支持，则选择输入数据类型为float32且输出数据类型为float32； 3. 如果2中的场景不支持，则选择输入数据类型为float16且输出数据类型为float16； 4. 如果3中的场景不支持，则报错。 – 对于矢量计算类算子，表示网络模型中算子支持float16和float32时，强制选择float32，若原图精度为float16，也会强制转为float32。 如果网络模型中存在部分算子，并且该算子实现不支持float32，比如某算子仅支持float16类型，则该参数不生效，仍然使用支持的float16；如果该算子不支持float32，且又配置了黑名单（precision_reduce = false），则会使用float32的AI CPU算子；如果AI CPU算子也不支持，则执行报错。 ● **force_fp16：** 表示网络模型中算子支持float16和float32时，强制选择float16。 ● **allow_fp32_to_fp16：** – 对于矩阵类算子，使用float16。 – 对于矢量类算子，优先保持原图精度，如果网络模型中算子支持float32，则保留原始精度float32，如果网络模型中算子不支持float32，则直接降低精度到float16。 ● **must_keep_origin_dtype：** 保持原图精度。**仅Atlas A2训练系列产品支持bfloat16类型。** – 如果原图中某算子精度为float16，AI Core中该算子的实现不支持float16、仅支持float32和bfloat16，则系统内部会自动采用高精度float32。 – 如果原图中某算子精度为float16，AI Core中该算子的实现不支持float16、仅支持bfloat16，则会使用float16的AI CPU算子；如果AI CPU算子也不支持，则执行报错。

编译选项	取值说明
	– 如果原图中某算子精度为float32，AI Core中该算子的实现不支持float32类型、仅支持float16类型，则会使用float32的AI CPU算子；如果AI CPU算子也不支持，则执行报错。 ● **allow_mix_precision/allow_mix_precision_fp16**：配置为allow_mix_precision或allow_mix_precision_fp16，效果等同，均表示使用混合精度float16和float32数据类型来处理神经网络的过程。allow_mix_precision_fp16为新版本中新增的，语义更清晰，便于理解。 针对网络模型中float32数据类型的算子，按照内置的优化策略，自动将部分float32的算子降低精度到float16，从而在精度损失很小的情况下提升系统性能并减少内存使用。 若配置了该种模式，则可以在OPP软件包安装路径${INSTALL_DIR}/opp/built-in/op_impl/ai_core/tbe/config/*<soc_version>*/aic-*<soc_version>*-ops-info.json内置优化策略文件中查看“precision_reduce”参数的取值： – 若取值为true（白名单），则表示允许将当前float32类型的算子，降低精度到float16。 – 若取值为false（黑名单），则不允许将当前float32类型的算子降低精度到float16，相应算子仍旧使用float32精度。 – 若网络模型中算子没有配置该参数（灰名单），当前算子的混合精度处理机制和前一个算子保持一致，即如果前一个算子支持降精度处理，当前算子也支持降精度；如果前一个算子不允许降精度，当前算子也不支持降精度。
ACL_AICORE_NUM	用于配置模型编译时使用的AI Core数量。 当前版本设置无效。
ACL_AUTO_TUNE_MODE	**该参数后续废弃，请勿配置，否则后续版本可能存在兼容性问题。**若涉及调优，请参见《**AOE工具使用指南**》。 用于配置算子的自动调优模式。 ● GA（Genetic Algorithm）：遗传算法，用于设置Cube算子的调优性能。 ● RL（Reinforcement Learning）：强化学习，用于设置Vector算子的调优性能。

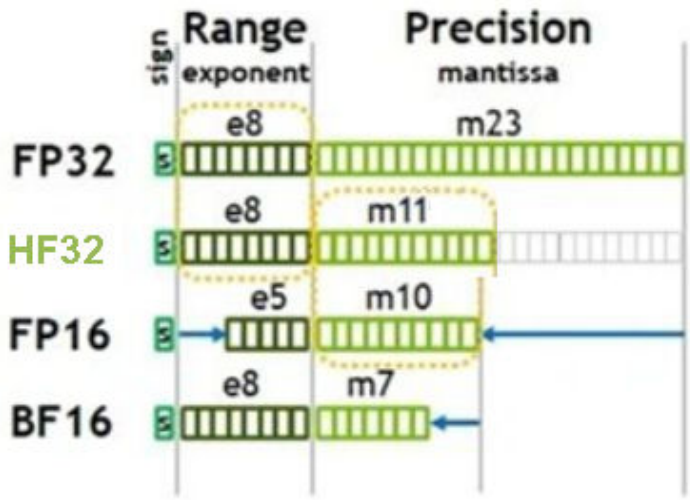
编译选项	取值说明
ACL_OP_SELECT_IMPL_MODE	用于选择算子是高精度实现还是高性能实现。如果不配置该编译选项，默认采用high_precision。 ● **high_precision**：表示算子采用高精度实现模式。 该参数采用系统内置的配置文件设置算子实现模式，内置配置文件路径为${INSTALL_DIR}/opp/built-in/op_impl/ai_core/tbe/impl_mode/high_precision.ini。 为保持兼容，该参数仅对high_precision.ini文件中算子列表生效，通过该列表可以控制算子生效的范围并保证之前版本的网络模型不受影响。 ● **high_performance**：表示算子采用高性能实现模式。 该参数采用系统内置的配置文件设置算子实现模式，内置配置文件路径为${INSTALL_DIR}/opp/built-in/op_impl/ai_core/tbe/impl_mode/high_performance.ini。 为保持兼容，该参数仅对high_performance.ini文件中算子列表生效，通过该列表可以控制算子生效的范围并保证之前版本的网络模型不受影响。 ● **high_precision_for_all**：表示算子采用高精度实现模式。 该参数采用系统内置的配置文件设置算子实现模式，内置配置文件路径为${INSTALL_DIR}/opp/built-in/op_impl/ai_core/tbe/impl_mode/high_precision_for_all.ini，该文件中列表后续可能会跟随版本更新。 **该实现模式不保证兼容**，如果后续新的软件包中有算子新增了实现模式（即配置文件中新增了某个算子的实现模式），之前版本使用high_precision_for_all的网络模型，在新版本上性能可能会下降。 ● **high_performance_for_all**：表示算子采用高性能实现模式。 该参数采用系统内置的配置文件设置算子实现模式，内置配置文件路径为${INSTALL_DIR}/opp/built-in/op_impl/ai_core/tbe/impl_mode/high_performance_for_all.ini，该文件中列表后续可能会跟随版本更新。 **该实现模式不保证兼容**，如果后续新的软件包中有算子新增了实现模式（即配置文件中新增了某个算子的实现模式），之前版本使用high_performance_for_all的网络模型，在新版本上精度可能会下降。
ACL_OPTYPELIST_FOR_IMPLMODE	通过ACL_OPTYPELIST_FOR_IMPLMODE选项设置算子类型的列表（多个算子使用英文逗号进行分隔），与ACL_OP_SELECT_IMPL_MODE选项配合使用，设置列表中的算子通过高精度实现或高性能实现。

编译选项	取值说明
ACL_OP_DEBUG_LEVEL	用于配置TBE算子编译debug功能开关。 ● 0：不开启算子debug功能，在当前执行路径**不生成**算子编译目录kernel_meta。**默认为0。** ● 1：开启算子debug功能，在当前执行路径生成kernel_meta文件夹，并在该文件夹下**生成**.o（算子二进制文件）、.json文件（算子描述文件）以及TBE指令映射文件（算子cce文件*.cce和python-cce映射文件*_loc.json），用于后续分析AICore Error问题。 ● 2：开启算子debug功能，在当前执行路径生成kernel_meta文件夹，并在该文件夹下**生成**.o（算子二进制文件）、.json文件（算子描述文件）以及TBE指令映射文件（算子cce文件*.cce和python-cce映射文件*_loc.json），用于后续分析AICore Error问题，同时设置为2，还会关闭编译优化开关、开启ccec调试功能（ccec编译器选项设置为-O0-g）。 ● 3：不开启算子debug功能，在当前执行路径生成kernel_meta文件夹，并在该文件夹中**生成**.o（算子二进制文件）和.json文件（算子描述文件），分析算子问题时可参考。 ● 4：不开启算子debug功能，在当前执行路径生成kernel_meta文件夹，并在该文件夹下**生成**.o（算子二进制文件）和.json文件（算子描述文件）以及TBE指令映射文件（算子cce文件*.cce）和UB融合计算描述文件（{$kernel_name}_compute.json），可在分析算子问题时进行问题复现、精度比对时使用。 **说明** 配置为2（即开启ccec编译选项）时，会导致算子Kernel（*.o文件）大小增大。动态Shape场景下，由于算子编译时会遍历可能的Shape场景，因此可能会导致算子Kernel文件过大而无法进行编译，此种场景下，建议不要配置ccec编译选项。 由于算子Kernel文件过大而无法编译的报错日志示例如下： message:link error ld.lld: **error: InputSection too large for range extension thunk** ./kernel_meta_xxxxx.o
ACL_DEBUG_DIR	用于配置保存模型转换、网络迁移过程中算子编译生成的调试相关过程文件的路径（默认路径：执行应用的当前路径/kernel_meta），包括算子.o/.json/.cce等文件。具体生成哪些文件以ACL_OP_DEBUG_LEVEL选项设置的取值为准。 路径支持大小写字母（a-z，A-Z）、数字（0-9）、下划线（_）、中划线（-）、句点（.）、中文字符。 **说明** 关于配置算子编译文件的缓存目录，除此处设置枚举值的方式，还可以配置环境变量ASCEND_WORK_PATH，几种方式的优先级为：本节设置枚举值的方式 > 设置环境变量 > 默认路径。 环境变量的详细配置说明请参见《**环境变量参考**》。

编译选项	取值说明
ACL_OP_COMPILER_CACHE_MODE	用于配置算子编译磁盘缓存模式。该编译选项需要与ACL_OP_COMPILER_CACHE_DIR配合使用。 ● enable：表示启用算子编译缓存。启用后可以避免针对相同编译参数及算子参数的算子重复编译，从而提升编译速度。 ● force：启用算子编译缓存功能，区别于enable模式，force模式下会强制刷新缓存，即先删除已有缓存，再重新编译并加入缓存。比如当用户的python变更、依赖库变更、算子调优后知识库变更等，需要先指定为force用于先清理已有的缓存，后续再修改为enable模式，以避免每次编译时都强制刷新缓存。 ● disable：表示禁用算子编译缓存。 该场景下，可以通过环境变量ASCEND_MAX_OP_CACHE_SIZE来限制某个芯片下缓存文件夹的磁盘空间的大小，当编译缓存空间大小达到ASCEND_MAX_OP_CACHE_SIZE设置的取值，且需要删除旧的kernel文件时，可以通过环境变量ASCEND_REMAIN_CACHE_SIZE_RATIO设置需要保留缓存的空间大小比例。配置示例如下：
ACL_OP_COMPILER_CACHE_DIR	用于配置算子编译文件的缓存目录（默认路径：$HOME/atc_data）。该编译选项需要与ACL_OP_COMPILER_CACHE_MODE配合使用。 路径支持大小写字母（a-z，A-Z）、数字（0-9）、下划线（_）、中划线（-）、句点（.）、中文字符。 如果设置了ACL_OP_DEBUG_LEVEL编译选项，则只有编译选项值为0才会启用编译缓存功能，其它取值禁用编译缓存功能。 **说明** 关于配置算子编译文件的缓存目录，除此处设置枚举值的方式，还可以配置环境变量ASCEND_CACHE_PATH，几种方式的优先级为：本节设置枚举值的方式 > 设置环境变量 > 默认路径。 环境变量的详细配置说明请参见《**环境变量参考**》。
ACL_OP_PERFORMANCE_MODE	**该参数已废弃，请勿配置，否则后续版本可能存在兼容性问题。** 通过该选项设置是否按照算子执行高性能的方式编译算子，默认采用normal方式。 取值范围： ● normal：算子编译时按照编译性能最高的方式进行编译。 ● high：算子编译时应该按照算子执行性能最好的方式去泛化编译。

编译选项	取值说明
ACL_OP_JIT_COMPILE	选择是在线编译算子，还是使用已编译的算子二进制文件。 ● enable：在线编译算子，系统根据得到的算子信息进行优化，从而编译出运行性能更优的算子。固定Shape网络场景下，建议设置为enable。 ● disable：优先查找系统中的已编译好的算子二进制文件，如果能查找到，则不再编译算子，编译性能更优；如果查找不到，则再编译算子。动态Shape网络场景下，建议设置为disable。若将本参数设置为disable，则需要安装算子二进制文件包，请参见《**CANN 软件安装指南**》中的“常用操作 > 安装、升级和卸载二进制算子包”章节。 Atlas 200/300/500 推理产品，该选项默认值为enable。 Atlas 推理系列产品，该选项默认值为enable。
ACL_OP_DETERMINISTIC	是否开启确定性计算。 ● 0：默认值，不开启确定性计算。 ● 1：开启确定性计算。 当开启确定性计算功能时，算子在相同的硬件和输入下，多次执行将产生相同的输出。但启用确定性计算往往导致算子执行变慢。 默认情况下，不开启确定性计算，算子在相同的硬件和输入下，多次执行的结果可能不同。这个差异的来源，一般是因为在算子实现中，存在异步的多线程执行，会导致浮点数累加的顺序变化。 通常建议不开启确定性计算，因为确定性计算往往会导致算子执行变慢，进而影响性能。当发现模型多次执行结果不同，或者是进行精度调优时，可开启确定性计算，辅助模型调试、调优。

编译选项	取值说明
ACL_CUSTOMIZE_DTYPES	*.cfg配置文件路径，包含文件名，配置文件中列举需要指定计算精度的算子名称或算子类型，每个算子单独一行。通过该配置，在模型编译时，可自定义某个或某些算子的计算精度。 **配置约束：** ● 路径和文件名支持大小写字母（a-z，A-Z）、数字（0-9）、下划线（_）、中划线（-）、句点（.）、英文冒号(:)、中文字符。 ● 配置文件中若为算子名称，以**Opname::InputDtype:dtype1,...,OutputDtype:dtype1,...**格式进行配置，每个Opname单独一行，dtype1，dtype2..需要与可设置计算精度的算子输入，算子输出的个数一一对应。 ● 配置文件中若为算子类型，以**OpType::TypeName:InputDtype:dtype1,...,OutputDtype:dtype1,...**格式进行配置，每个OpType单独一行，dtype1，dtype2..需要与可设置计算精度的算子输入，算子输出的个数一一对应，且算子OpType必须为基于Ascend IR定义的算子的OpType，OpType可查阅《**算子清单（开放态）**》。 ● 对于同一个算子，如果同时配置了**Opname**和**OpType**的配置项，编译时以**Opname**的配置项为准。 ● 使用该参数指定某个算子的计算精度时，如果模型转换过程中该算子被融合掉，则该算子指定的计算精度不生效。
ACL_OP_PRECISION_MODE	设置算子精度模式的配置文件（.ini格式）路径以及文件名，路径和文件名：支持大小写字母（a-z，A-Z）、数字（0-9）、下划线（_）、中划线（-）、句点（.）、中文字符. ● 配置文件中支持设置如下精度模式： – high_precision：表示高精度。 – high_performance：表示高性能。 – support_out_of_bound_index：表示对gather、scatter和segment类算子的indices输入进行越界校验， 校验会降低算子的执行性能。 ● 构造算子精度模式配置文件*op_precision.ini*，并在该文件中按照算子类型、节点名称设置精度模式，每一行设置一个算子类型或节点名称的精度模式，按节点名称设置精度模式的优先级高于按算子类型。 配置样例如下： [ByOpType] *optype1*=high_precision *optype2*=high_performance *optype3*=support_of_bound_index [ByNodeName] *nodename1*=high_precision *nodename2*=high_performance *nodename3*=support_of_bound_index

编译选项	取值说明
ACL_ALLOW_HF32	**当前版本不支持该参数。** 算子内部计算时是否允许HF32类型替换FP32类型，true允许，false不允许，当前版本该配置仅针对Conv类算子与Matmul类算子生效。默认针对Conv类算子，启用FP32转换为HF32；默认针对Matmul类算子，关闭FP32转换为HF32。 HF32是昇腾推出的专门用于算子内部计算的单精度浮点类型，与其他常用数据类型的比较如下图所示。HF32与FP32支持相同的数值范围，但尾数位精度（11位）却接近FP16（10位）。通过降低精度让HF32单精度数据类型代替原有的FP32单精度数据类型，可大大降低数据所占空间大小，实现性能的提升。  Atlas 200/300/500 推理产品，不支持该配置。 Atlas 推理系列产品，不支持该配置。 **说明** • 配置为true时，哪些算子内部计算时**允许**HF32类型替换FP32类型，可在“CANN软件安装路径/opp/built-in/op_impl/ai_core/tbe/impl_mode/allow_hf32_matmul_**t**_conv_**t**.ini”文件中查看； • 配置为false时，哪些算子内部计算时**不允许**HF32类型替换FP32类型，可在“CANN软件安装路径/opp/built-in/op_impl/ai_core/tbe/impl_mode/allow_hf32_matmul_**f**_conv_**f**.ini”文件中查看。

11.27.28 acldvppChannelDescParamType

```
enum acldvppChannelDescParamType {
    ACL_DVPP_CSC_MATRIX_UINT32 = 0,     // 色域转换矩阵属性，该属性类型对应的值为uint32_t类型
    ACL_DVPP_MODE_UINT32,               // 通道描述信息中的通道模式，该属性类型对应的值为uint32_t类型
    ACL_DVPP_CHANNEL_ID_UINT64,         // 通道ID，该属性类型对应的值为uint64_t类型
    ACL_DVPP_CHANNEL_HEIGHT_UINT32,     // 通道的高度，该属性类型对应的值为uint32_t类型
    ACL_DVPP_CHANNEL_WIDTH_UINT32,      // 通道的宽度，该属性类型对应的值为uint32_t类型
    ACL_DVPP_JPEGD_PRECISION_MODE_ENUM  // JPEGD解码输出图片的宽、高对齐模式，该属性类型对应的
```

```
值为int32_t类型
}
```

表 11-82 属性配置

属性类型	取值说明
ACL_DVPP_CSC_MATRIX_UINT32	色域转换矩阵属性，属性值请参见**acldvppCscMatrix**。 Atlas 200/300/500 推理产品，不支持该选项。
ACL_DVPP_MODE_UINT32	通道描述信息中的通道模式，用于明确图片数据处理通道用于实现哪种功能，目前支持**VPC**、**JPEGD**、**JPEGE**、**PNGD**功能。属性值请参见**acldvppChannelMode**。 Atlas 200/300/500 推理产品，不支持该选项。
ACL_DVPP_CHANNEL_ID_UINT64	通道ID。
ACL_DVPP_CHANNEL_HEIGHT_UINT32	通道的高度，目前仅支持在**JPEGE**功能中使用该属性，其它功能中设置该属性无效。 Atlas 推理系列产品上，默认值和最大值为8192。 Atlas 200/300/500 推理产品，不支持该选项。
ACL_DVPP_CHANNEL_WIDTH_UINT32	通道的宽度，目前仅支持在**JPEGE**功能中使用该属性，其它功能中设置该属性无效。 Atlas 推理系列产品上，默认值和最大值为8192。 Atlas 200/300/500 推理产品，不支持该选项。
ACL_DVPP_JPEGD_PRECISION_MODE_ENUM	JPEGD解码输出图片的宽、高对齐模式，模式选项参见**11.27.46 acldvppJpegdPrecisionMode**。 目前仅支持在**JPEGD**功能中使用该属性，其它功能中设置该属性无效。 Atlas 200/300/500 推理产品，不支持该选项。

11.27.29 aclvdecChannelDescParamType

```
enum aclvdecChannelDescParamType {
    ACL_VDEC_CSC_MATRIX_UINT32 = 0,      //色域转换矩阵属性，该属性类型对应的值为uint32_t类型
    ACL_VDEC_OUT_MODE_UINT32,            //是否实时出帧，该属性类型对应的值为uint32_t类型
    ACL_VDEC_THREAD_ID_UINT64,           //回调线程ID，该属性类型对应的值为uint64_t类型
    ACL_VDEC_CALLBACK_PTR,               //回调函数，该属性类型对应的值是内存指针
    ACL_VDEC_CHANNEL_ID_UINT32,          //通道号，该属性类型对应的值为uint32_t类型
    ACL_VDEC_ENCODE_TYPE_UINT32,         //视频编码协议，该属性类型对应的值为uint32_t类型
    ACL_VDEC_OUT_PIC_FORMAT_UINT32,      //输出图片格式，该属性类型对应的值为uint32_t类型
    ACL_VDEC_OUT_PIC_WIDTH_UINT32,       //解码码流最大宽度，该属性类型对应的值为uint32_t类型
    ACL_VDEC_OUT_PIC_HEIGHT_UINT32,      //解码码流最大高度，该属性类型对应的值为uint32_t类型
    ACL_VDEC_REF_FRAME_NUM_UINT32,       //参考帧数量，该属性类型对应的值为uint32_t类型
    ACL_VDEC_BIT_DEPTH_UINT32            //视频位宽，该属性类型对应的值为uint32_t类型
}
```

表 11-83 属性配置

属性类型	取值说明
ACL_VDEC_CSC_MATRIX_UINT32	色域转换矩阵属性，属性值请参见**acldvppCscMatrix**。 Atlas 200/300/500 推理产品，不支持该选项。
ACL_VDEC_OUT_MODE_UINT32	设置是否实时出帧（即发送一帧解码一帧，无需依赖后续帧的传入）。 取值范围如下： ● 0：默认出帧模式，由于解码过程中存在缓存帧，无法实时输出，因此VDEC需要在收到码流中的多帧数据后，才开始输出解码结果。 ● 1：快速出帧模式，VDEC获取码流中的一帧数据后，就开始实时输出解码结果。只支持简单参考关系的H264/H265标准码流（无长期参考帧，无B帧）。
ACL_VDEC_THREAD_ID_UINT64	回调线程ID。 **说明** 同一个进程内，在不同的Device上注册VDEC解码回调函数的线程时，不能指定同一个线程ID。 同一个Device上，多路VDEC解码可以指定同一个线程ID，但相比一个线程处理一路VDEC解码任务来说，一个线程内串行处理多路VDEC解码任务时，VDEC解码性能可能下降。
ACL_VDEC_CALLBACK_PTR	解码回调函数。 请参见**11.17.9.5 aclvdecCallback**。
ACL_VDEC_CHANNEL_ID_UINT32	通道ID。 Atlas 200/300/500 推理产品，该参数值的取值范围[0, 31]。 Atlas 推理系列产品，该参数值的取值范围[0, 255]。
ACL_VDEC_ENCODE_TYPE_UINT32	视频编码协议H265-main level（0）、H264-baseline level（1）、H264-main level(2)、H264-high level（3）。 请参见**acldvppStreamFormat**。
ACL_VDEC_OUT_PIC_FORMAT_UINT32	YUV图像存储格式，支持如下格式： ● YUV420SP NV12 ● YUV420SP NV21 ● RGB888，Atlas 200/300/500 推理产品不支持该格式 ● BGR888，Atlas 200/300/500 推理产品不支持该格式 如果不设置输出格式，默认使用YUV420SP NV12。
ACL_VDEC_OUT_PIC_WIDTH_UINT32	解码码流最大宽度。 Atlas 200/300/500 推理产品，如果不设置解码码流最大宽度，内部默认使用1920。 Atlas 推理系列产品，如果不设置解码码流最大宽度，内部默认使用4096。

属性类型	取值说明
ACL_VDEC_OUT_PIC_HEIGHT_UINT32	解码码流最大高度。 Atlas 200/300/500 推理产品，如果不设置解码码流最大高度，内部默认使用1120。 Atlas 推理系列产品，如果不设置解码码流最大高度，内部默认使用4096。
ACL_VDEC_REF_FRAME_NUM_UINT32	参考帧数量，取值范围[0, 16]。 Atlas 200/300/500 推理产品，当前版本不支持用户设置该参数。 Atlas 推理系列产品，如果不设置参考帧数量，内部默认使用8。
ACL_VDEC_BIT_DEPTH_UINT32	设置视频位宽，默认值为10-bit。 取值范围如下： ● 0：8-bit ● 1: 10-bit(默认值 ） Atlas 200/300/500 推理产品，当前版本不支持用户设置该参数。

11.27.30 aclvencChannelDescParamType

```
enum aclvencChannelDescParamType {
    ACL_VENC_THREAD_ID_UINT64 = 0,      //回调线程ID，该属性类型对应的值为uint64_t类型
    ACL_VENC_CALLBACK_PTR,              //回调函数，该属性类型对应的值是内存指针
    ACL_VENC_PIXEL_FORMAT_UINT32,       //输入图像格式，该属性类型对应的值为uint32_t类型
    ACL_VENC_ENCODE_TYPE_UINT32,        //视频编码协议，该属性类型对应的值为uint32_t类型
    ACL_VENC_PIC_WIDTH_UINT32,          //输入图片宽度，该属性类型对应的值为uint32_t类型
    ACL_VENC_PIC_HEIGHT_UINT32,         //输入图片高度，该属性类型对应的值为uint32_t类型
    ACL_VENC_KEY_FRAME_INTERVAL_UINT32, //关键帧间隔，该属性类型对应的值为uint32_t类型
    ACL_VENC_BUF_ADDR_PTR,              //编码输出缓存地址
    ACL_VENC_BUF_SIZE_UINT32,           //编码输出缓存大小，该属性类型对应的值为uint32_t类型
    ACL_VENC_RC_MODE_UINT32,            //码率控制模式，该属性类型对应的值为uint32_t类型
    ACL_VENC_SRC_RATE_UINT32,           //输入码流帧率，该属性类型对应的值为uint32_t类型
    ACL_VENC_MAX_BITRATE_UINT32,        //输出码率，该属性类型对应的值为uint32_t类型
    ACL_VENC_MAX_IP_PROP_UINT32         //一个GOP内单个I帧bit数和单个P帧bit数的比例，该属性类型对应
的值为uint32_t类型
}
```

表 11-84 属性配置

属性类型	取值说明
ACL_VENC_THREAD_ID_UINT64	回调线程ID。 **说明** 同一个进程内，在不同的Device上注册VENC编码回调函数的线程时，不能指定同一个线程ID。
ACL_VENC_CALLBACK_PTR	编码回调函数。

属性类型	取值说明
ACL_VENC_PIXEL_FORMAT_UINT32	输入图像格式，支持如下格式： ● PIXEL_FORMAT_YUV_SEMIPLANAR_420 ● PIXEL_FORMAT_YVU_SEMIPLANAR_420
ACL_VENC_ENCODE_TYPE_UINT32	视频编码协议。 参见**11.27.7 acldvppStreamFormat**。
ACL_VENC_PIC_WIDTH_UINT32	图片宽度。
ACL_VENC_PIC_HEIGHT_UINT32	图片高度。
ACL_VENC_KEY_FRAME_INTERVAL_UINT32	关键帧间隔，取值范围[1,65536]。
ACL_VENC_BUF_ADDR_PTR	编码输出缓存指针。 **说明** 在Atlas 200/300/500 推理产品上，当前版本不支持用户设置该参数，由VENC内部管理该缓存地址。 当前版本获取该参数的值为无效值，不支持获取该参数值。
ACL_VENC_BUF_SIZE_UINT32	编码输出缓存大小，单位为Byte。 **说明** 在Atlas 200/300/500 推理产品上，当前版本不支持用户设置该参数，内部默认使用3686400 Byte。 在Atlas 推理系列产品上，如果不设置该参数，参数值默认为8M；如果设置该参数，参数值最小为5M。 当前版本获取该参数的值为无效值，不支持获取该参数值。
ACL_VENC_RC_MODE_UINT32	指定码率控制模式。 ● 0表示使用默认值 ● 1表示变码率VBR模式 ● 2表示定码率CBR模式 **说明** 如果不设置该参数，则采用默认值0。 Atlas 200/300/500 推理产品，默认值0表示CBR模式。 Atlas 推理系列产品，默认值0表示VBR模式。
ACL_VENC_SRC_RATE_UINT32	输入码流帧率，单位fps。 Atlas 200/300/500 推理产品，取值范围0或者[1,120]。 Atlas 推理系列产品，取值范围0或者[1,240]。 如果不设置该参数，默认为30；如果设置为0，表示使用默认值，即30。如果该值和实际输入码流帧率相差太大，会影响输出码率。

属性类型	取值说明
ACL_VENC_MAX_BITRATE_UINT32	输出码率，单位kbps。 Atlas 200/300/500 推理产品，取值范围0或[10,30000]。如果不设置该参数，默认为300；如果设置为0，表示使用默认值，即300。 Atlas 推理系列产品，取值范围[2,614400]，如果不设置该参数，默认为2000；如果设置为0，表示使用默认值，即2000。
ACL_VENC_MAX_IP_PROP_UINT32	一个GOP内单个I帧bit数和单个P帧bit数的比例，取值范围0或[1,100] 。如果不设置该参数，VBR模式下此值默认为80，CBR模式下此值默认为70；如果设置为0，表示使用默认值，即VBR模式下此值默认为80，CBR模式下此值默认为70。

11.27.31 aclmdlConfigAttr

```
typedef enum {
    ACL_MDL_PRIORITY_INT32 = 0,
    ACL_MDL_LOAD_TYPE_SIZET,
    ACL_MDL_PATH_PTR,
    ACL_MDL_MEM_ADDR_PTR,
    ACL_MDL_MEM_SIZET,
    ACL_MDL_WEIGHT_ADDR_PTR,
    ACL_MDL_WEIGHT_SIZET,
    ACL_MDL_WORKSPACE_ADDR_PTR,
    ACL_MDL_WORKSPACE_SIZET,
    ACL_MDL_INPUTQ_NUM_SIZET,
    ACL_MDL_INPUTQ_ADDR_PTR,
    ACL_MDL_OUTPUTQ_NUM_SIZET,
    ACL_MDL_OUTPUTQ_ADDR_PTR,
    ACL_MDL_WORKSPACE_MEM_OPTIMIZE
} aclmdlConfigAttr;
```

表 11-85 模型加载选项配置

选项	取值说明
ACL_MDL_PRIORITY_INT32	模型执行的优先级，数字越小优先级越高，取值[0,7]，可选项。默认值为0。

ACL_MDL_LOAD_TYPE_SIZET	模型加载方式，必选项。
	ACL_MDL_LOAD_TYPE_SIZET（表示模型加载方式）的取值使用如下宏：
	● ACL_MDL_LOAD_FROM_FILE
	`#define ACL_MDL_LOAD_FROM_FILE 1`
	● ACL_MDL_LOAD_FROM_FILE_WITH_MEM
	`#define ACL_MDL_LOAD_FROM_FILE_WITH_MEM 2`
	● ACL_MDL_LOAD_FROM_MEM
	`#define ACL_MDL_LOAD_FROM_MEM 3`
	● ACL_MDL_LOAD_FROM_MEM_WITH_MEM
	`#define ACL_MDL_LOAD_FROM_MEM_WITH_MEM 4`
	● ACL_MDL_LOAD_FROM_FILE_WITH_Q
	`#define ACL_MDL_LOAD_FROM_FILE_WITH_Q 5`

选项	取值说明
ACL_MDL_PATH_PTR	离线模型文件路径的指针，如果选择从文件加载模型，则该选项必选。
ACL_MDL_MEM_ADDR_PTR	模型在内存中的起始地址，如果选择从内存加载模型，则该选项必选。
ACL_MDL_MEM_SIZET	模型在内存中的大小，如果选择从内存加载模型，则该选项必选，与ACL_MDL_MEM_ADDR_PTR选项配合使用。
ACL_MDL_WEIGHT_ADDR_PTR	Device上模型权值内存（存放权值数据）的指针，如果需要由用户管理权值内存，则该选项必选。若不配置该选项，则表示由系统管理内存。
ACL_MDL_WEIGHT_SIZET	权值内存大小，单位为Byte，如果需要由用户管理权值内存，则该选项必选，与ACL_MDL_WEIGHT_ADDR_PTR选项配合使用。
ACL_MDL_WORKSPACE_ADDR_PTR	Device上模型所需工作内存（存放模型执行过程中的临时数据）的指针，如果需要由用户管理工作内存，则该选项必选。若不配置该选项，则表示由系统管理内存。
ACL_MDL_WORKSPACE_SIZET	模型所需工作内存的大小，单位为Byte，如果需要由用户管理工作内存，则该选项必选，与ACL_MDL_WORKSPACE_ADDR_PTR选项配合使用。
ACL_MDL_INPUTQ_NUM_SIZET	模型输入队列大小 ,带队列加载模型时，该选项必选，与ACL_MDL_INPUTQ_ADDR_PTR选项配合使用。
ACL_MDL_INPUTQ_ADDR_PTR	模型输入队列ID的指针，带队列加载模型时，该选项必选，一个模型输入对应一个队列ID。
ACL_MDL_OUTPUTQ_NUM_SIZET	模型输出队列大小，带队列加载模型时，该选项必选，与ACL_MDL_OUTPUTQ_ADDR_PTR选项配置使用。
ACL_MDL_OUTPUTQ_ADDR_PTR	模型输出队列ID的指针，带队列加载模型时，该选项必选，一个模型输出对应一个队列ID。
ACL_MDL_WORKSPACE_MEM_OPTIMIZE	是否开启模型工作内存优化功能，1开启，0不开启。 若关注内存规划或内存资源有限时，建议在模型加载前，开启工作内存优化功能，此时工作内存中不包含存放模型输入、输出数据的内存，工作内存大小会减小，达到节省内存的目的。 在模型执行前，还需要由用户申请存放模型输入、输出数据的内存，因此即使在模型加载时开启工作内存优化功能，也不会影响后续的模型执行。

说明

- 关于如何获取om文件，请参见《**ATC工具使用指南（开放态）**》。
- 对om模型文件大小有限制的场景下，如果使用ATC工具生成om文件时，将--external_weight参数设置为1（1表示将原始网络中的Const/Constant节点的权重保存在单独的文件中，且该文件保存在与om文件同级的weight目录下），那么在使用本接口加载om文件时，需将weight目录与om文件放在同级目录下，这时AscendCL会自行到weight目录下查找权重文件，否则可能会导致单独的权重文件加载不成功。

11.27.32 aclMemType

```
typedef enum {
    ACL_MEMTYPE_DEVICE = 0,  //Device内存
    ACL_MEMTYPE_HOST = 1,    //Host内存
    ACL_MEMTYPE_HOST_COMPILE_INDEPENDENT = 2   //Host内存
}
```

ACL_MEMTYPE_HOST和ACL_MEMTYPE_HOST_COMPILE_INDEPENDENT都标识Host内存，但在使用上有区别：

- ACL_MEMTYPE_HOST：若通过**aclSetCompileopt**接口将ACL_OP_JIT_COMPILE设置为disable，设置该选项时，算子输入或输出的值的变化，不会触发算子重新编译；若通过**aclSetCompileopt**接口将ACL_OP_JIT_COMPILE设置为enable，算子输入或输出的值的变化，会触发算子重新编译。
- ACL_MEMTYPE_HOST_COMPILE_INDEPENDENT：设置该选项时，算子输入或输出的值的变化，都不会触发算子重新编译。若算子编译时依赖其输入或输出的值，此时如果设置为 ACL_MEMTYPE_HOST_COMPILE_INDEPENT，则可能会导致编译失败。

11.27.33 aclOpCompileFlag

该参数已废弃，请勿配置，否则后续版本可能存在兼容性问题。

```
typedef enum aclCompileFlag {
    ACL_OP_COMPILE_DEFAULT,   //精确编译，不设置编译flag时，默认是精确编译
    ACL_OP_COMPILE_FUZZ
}
```

11.27.34 aclrtEventWaitStatus

```
aclrtEventWaitStatus{
    ACL_EVENT_WAIT_STATUS_COMPLETE = 0,        //完成
    ACL_EVENT_WAIT_STATUS_NOT_READY = 1,       //未完成
    ACL_EVENT_WAIT_STATUS_RESERVED = 0xffff,  //预留
}
```

11.27.35 aclprofStepTag

```
typedef enum{
    ACL_STEP_START = 0，  //step  start
    ACL_STEP_END = 1   //step  end
} aclprofStepTag
```

说明

同一个**aclprofStepInfo**对象、同一个tag只能设置一次，否则Profiling解析会出错。

11.27.36 acltdtBuf

```
typedef void *acltdtBuf;
```

11.27.37 acltdtQueueAttrType

```
typedef enum {
    ACL_TDT_QUEUE_NAME_PTR = 0,  //队列名
    ACL_TDT_QUEUE_DEPTH_UINT32   //队列深度
} acltdtQueueAttrType;
```

11.27.38 acltdtQueueRouteParamType

```
typedef enum {
    ACL_TDT_QUEUE_ROUTE_SRC_UINT32 = 0,  //源队列ID
    ACL_TDT_QUEUE_ROUTE_DST_UINT32,      //目标队列ID
    ACL_TDT_QUEUE_ROUTE_STATUS_INT32     //路由绑定状态，0表示未绑定，1表示绑定，2表示绑定异常
} acltdtQueueRouteParamType;
```

11.27.39 acltdtQueueRouteQueryMode

```
typedef enum {
    ACL_TDT_QUEUE_ROUTE_QUERY_SRC = 0,         //指定为只根据源队列ID匹配查询
    ACL_TDT_QUEUE_ROUTE_QUERY_DST = 1,         //指定为只根据目标队列ID匹配查询
    ACL_TDT_QUEUE_ROUTE_QUERY_SRC_AND_DST = 2  //指定为同时根据源、目标队列ID匹配查询
} acltdtQueueRouteQueryMode;
```

11.27.40 acltdtQueueRouteQueryInfoParamType

```
typedef enum {
    ACL_TDT_QUEUE_ROUTE_QUERY_MODE_ENUM = 0,  //查询匹配模式，选择该参数类型后，参数值来源于acltdtQueueRouteQueryMode枚举值
    ACL_TDT_QUEUE_ROUTE_QUERY_SRC_ID_UINT32,  //指定要查询的源队列ID
    ACL_TDT_QUEUE_ROUTE_QUERY_DST_ID_UINT32   //指定要查询的目标队列ID
} acltdtQueueRouteQueryInfoParamType;
```

11.27.41 acldvppChannelMode

```
enum acldvppChannelMode {
    DVPP_CHNMODE_VPC = 1,     //指定图片数据处理通道用于实现VPC功能
    DVPP_CHNMODE_JPEGD = 2,   //指定图片数据处理通道用于实现JPEGD功能
    DVPP_CHNMODE_JPEGE = 4,   //指定图片数据处理通道用于实现JPEGE功能
    DVPP_CHNMODE_PNGD = 8     //指定图片数据处理通道用于实现PNGD功能
};
```

11.27.42 aclopEngineType

```
typedef enum aclEngineType {
   ACL_ENGINE_SYS,      // 不关心具体执行引擎时填写，当前仅支持设置为该选项
   ACL_ENGINE_AICORE,   // 将算子编译成AI Core算子，当前不支持
   ACL_ENGINE_VECTOR    //  将算子编译成Vector Core算子，当前不支持
} aclopEngineType;
```

11.27.43 aclopCompileType

```
typedef enum aclCompileType {
    ACL_COMPILE_SYS,  // 当前仅支持设置为该选项
    ACL_COMPILE_UNREGISTERED   // 当前不支持
} aclopCompileType;
```

11.27.44 aclrtEventRecordedStatus

```
typedef enum aclrtEventRecordedStatus {
    ACL_EVENT_RECORDED_STATUS_NOT_READY = 0,  //Event未被记录到Stream中，或记录到Stream中的Event未被执行或执行失败
    ACL_EVENT_RECORDED_STATUS_COMPLETE = 1,   //记录到Stream中的Event执行成功
} aclrtEventRecordedStatus;
```

11.27.45 aclrtFloatOverflowMode

```
typedef enum aclrtFloatOverflowMode {
    ACL_RT_OVERFLOW_MODE_SATURATION = 0, // 饱和模式，默认值
    ACL_RT_OVERFLOW_MODE_INFNAN,         // Inf/NaN模式（符合IEEE 754标准）
    ACL_RT_OVERFLOW_MODE_UNDEF,
} aclrtFloatOverflowMode;
```

11.27.46 acldvppJpegdPrecisionMode

```
enum acldvppJpegdPrecisionMode {
    ACL_YUVOUT_ALIGN_DOWN = 0,
    ACL_YUVOUT_ALIGN_UP = 1,
    ACL_YUVOUT_ALIGN_DOWN_COMPAT = 2
};
```

对齐模式取值的含义如下：

- ACL_YUVOUT_ALIGN_DOWN：
  - 解码后的输出图片格式为YUV420SP时，若源图的宽、高为奇数时，将输出图片的宽、高向下2对齐
  - 解码后的输出图片格式为YUV422SP时，若源图的宽为奇数时，将输出图片的宽向下2对齐
  - 解码后的输出图片格式为YUV440SP时，若源图的高为奇数时，将输出图片高向下2对齐
- ACL_YUVOUT_ALIGN_UP：
  - 解码后的输出图片格式为YUV420SP时，若源图的宽、高为奇数时，将输出图片的宽、高向上2对齐
  - 解码后的输出图片格式为YUV422SP时，若源图的宽为奇数时，将输出图片的宽向上2对齐
  - 解码后的输出图片格式为YUV440SP时，若源图的高为奇数时，将输出图片高向上2对齐
- ACL_YUVOUT_ALIGN_DOWN_COMPAT：默认值

  兼容旧版本，解码后的输出图片格式为YUV420SP/YUV422SP/YUV440SP时，若源图的宽、高为奇数时，将输出图片的宽、高向下2对齐。

11.27.47 aclmdlExecConfigAttr

```
typedef enum {
    ACL_MDL_STREAM_SYNC_TIMEOUT = 0,
    ACL_MDL_EVENT_SYNC_TIMEOUT,
    ACL_MDL_WORK_ADDR_PTR,
    ACL_MDL_WORK_SIZET,
    ACL_MDL_MPAIMID_SIZET,      /** param reserved */
    ACL_MDL_AICQOS_SIZET,       /** param reserved */
    ACL_MDL_AICOST_SIZET,       /** param reserved */
    ACL_MDL_MEC_TIMETHR_SIZET   /** param reserved */
} aclmdlExecConfigAttr;
```

表 11-86 枚举项说明

枚举项	说明
ACL_MDL_STREAM_SYNC_TIMEOUT	在执行模型推理时控制Stream任务的超时时间。该属性值为INT32类型。
ACL_MDL_EVENT_SYNC_TIMEOUT	在执行模型推理时控制Event任务的超时时间。该属性值为INT32类型。
ACL_MDL_WORK_ADDR_PTR	模型所需工作内存（Device上存放模型执行过程中的临时数据）的指针，由用户管理工作内存。一般用于模型一次加载、多并发执行的场景。 如果同时配置ACL_MDL_WORK_ADDR_PTR以及**aclrtStreamConfigAttr**中的ACL_RT_STREAM_WORK_ADDR_PTR（表示Stream上模型的工作内存），则以ACL_MDL_WORK_ADDR_PTR优先。 当前版本不支持该配置。
ACL_MDL_WORK_SIZET	模型所需工作内存的大小，单位为Byte。一般用于模型一次加载、多并发执行的场景。 当前版本不支持该配置。
ACL_MDL_MPAIMID_SIZET	预留配置。
ACL_MDL_AICQOS_SIZET	预留配置。
ACL_MDL_AICOST_SIZET	预留配置。
ACL_MDL_MEC_TIMETHR_SIZET	预留配置。

11.27.48 aclrtStreamConfigAttr

```
typedef enum {
    ACL_RT_STREAM_WORK_ADDR_PTR = 0,
    ACL_RT_STREAM_WORK_SIZE,
    ACL_RT_STREAM_FLAG,
    ACL_RT_STREAM_PRIORITY,
} aclrtStreamConfigAttr;
```

表 11-87 枚举项说明

枚举项	说明
ACL_RT_STREAM_WORK_ADDR_PTR	某一个Stream上的模型所需工作内存（Device上存放模型执行过程中的临时数据）的指针，由用户管理工作内存。该配置主要用于多模型在同一个Stream上串行执行时想共享工作内存的场景，此时需按多个模型中最大的工作内存来申请内存，可提前使用**aclmdlQuerySize**查询各模型所需的工作内存大小。 如果同时配置ACL_RT_STREAM_WORK_ADDR_PTR以及**aclmdlExecConfigAttr**中的ACL_MDL_WORK_ADDR_PTR（表示某个模型的工作内存），则以aclmdlExecConfigAttr中的ACL_MDL_WORK_ADDR_PTR优先。
ACL_RT_STREAM_WORK_SIZE	模型所需工作内存的大小，单位为Byte。
ACL_RT_STREAM_FLAG	预留配置，默认值为0。
ACL_RT_STREAM_PRIORITY	Stream的优先级，数字越小优先级越高，取值[0,7]。默认值为0。

11.27.49 aclSysParamOpt

```
typedef enum {
    ACL_OPT_DETERMINISTIC = 0,
} aclSysParamOpt;
```

表 11-88 枚举项说明

枚举项	说明
ACL_OPT_DETERMINISTIC	是否开启确定性计算。 ● 0：默认值，不开启确定性计算。 ● 1：开启确定性计算。 当开启确定性计算功能时，算子在相同的硬件和输入下，多次执行将产生相同的输出。但启用确定性计算往往导致算子执行变慢。 默认情况下，不开启确定性计算，算子在相同的硬件和输入下，多次执行的结果可能不同。这个差异的来源，一般是因为在算子实现中，存在异步的多线程执行，会导致浮点数累加的顺序变化。 通常建议不开启确定性计算，因为确定性计算往往会导致算子执行变慢，进而影响性能。当发现模型多次执行结果不同，或者是进行精度调优时，可开启确定性计算，辅助模型调试、调优。

11.27.50 aclrtUtilizationInfo

```
typedef struct aclrtUtilizationInfo {
    int32_t cubeUtilization;   // Cube利用率
    int32_t vectorUtilization; // Vector利用率
    int32_t aicpuUtilization;  // AI CPU利用率
    int32_t memoryUtilization; // Device内存利用率
    aclrtUtilizationExtendInfo *utilizationExtend; // 预留参数，当前设置为null
} aclrtUtilizationInfo;
```

11.27.51 aclrtStreamStatus

```
typedef enum aclrtStreamStatus {
    ACL_STREAM_STATUS_COMPLETE  = 0,      // Stream上的所有任务已完成
    ACL_STREAM_STATUS_NOT_READY = 1,      // Stream上至少有一个任务未完成
    ACL_STREAM_STATUS_RESERVED  = 0xFFFF, // 预留
} aclrtStreamStatus;
```

11.27.52 aclrtMemHandleType

```
typedef enum aclrtMemHandleType {
    ACL_MEM_HANDLE_TYPE_NONE = 0,
} aclrtMemHandleType;
```

11.27.53 aclrtMemAllocationType

```
typedef enum aclrtMemAllocationType {
    ACL_MEM_ALLOCATION_TYPE_PINNED = 0,
} aclrtMemAllocationType;
```

11.27.54 aclrtMemAttr

```
typedef enum aclrtMemAttr {
    ACL_DDR_MEM,          // DDR内存，DDR上所有大页内存+普通内存
    ACL_HBM_MEM,          // HBM内存，HBM上所有大页内存+普通内存
    ACL_DDR_MEM_HUGE,     // DDR大页内存
    ACL_DDR_MEM_NORMAL,   // DDR普通内存
    ACL_HBM_MEM_HUGE,     // HBM大页内存
    ACL_HBM_MEM_NORMAL,   // HBM普通内存
    ACL_DDR_MEM_P2P_HUGE,     // DDR中用于Device间数据复制的大页内存
    ACL_DDR_MEM_P2P_NORMAL,   // DDR中用于Device间数据复制的普通内存
    ACL_HBM_MEM_P2P_HUGE,     // HBM中用于Device间数据复制的大页内存
    ACL_HBM_MEM_P2P_NORMAL,   // HBM中用于Device间数据复制的普通内存
} aclrtMemAttr;
```

11.27.55 aclrtMemLocationType

```
typedef enum aclrtMemLocationType {
    ACL_MEM_LOCATION_TYPE_HOST = 0, // 预留，当前不支持
    ACL_MEM_LOCATION_TYPE_DEVICE,
} aclrtMemLocationType;
```

11.27.56 aclrtMemLocation

```
typedef struct aclrtMemLocation {
    uint32_t id;
    aclrtMemLocationType type;    // 内存所在位置，当前仅支持ACL_MEM_LOCATION_TYPE_DEVICE（表示
Device上的内存）
} aclrtMemLocation;
```

11.27.57 aclrtPhysicalMemProp

```
typedef struct aclrtPhysicalMemProp {
    aclrtMemHandleType handleType;
```

```
    aclrtMemAllocationType allocationType;
    aclrtMemAttr memAttr;
    aclrtMemLocation location;
    uint64_t reserve;
} aclrtPhysicalMemProp;
```

成员名称	描述
handleType	handle类型，当前仅支持ACL_MEM_HANDLE_TYPE_NONE 。
allocationType	内存分配类型，当前仅支持ACL_MEM_ALLOCATION_TYPE_PINNED。
memAttr	内存属性。
location	内存所在位置。
reserve	预留。

11.27.58 aclrtDrvMemHandle

```
typedef void* aclrtDrvMemHandle;
```

11.27.59 aclrtDeviceStatus

```
typedef enum aclrtDeviceStatus {
    ACL_RT_DEVICE_STATUS_NORMAL = 0,      // Device正常可用
    ACL_RT_DEVICE_STATUS_ABNORMAL,        // Device异常
    ACL_RT_DEVICE_STATUS_END = 0xFFFF,    // 预留值
} aclrtDeviceStatus;
```

11.27.60 aclmdlAIPP

11.27.60.1 aclmdlCreateAIPP

函数功能

动态AIPP场景下，根据模型支持的batch size创建aclmdlAIPP类型的数据，用于存放动态AIPP的参数。

如需销毁aclmdlAIPP类型的数据，请参见**aclmdlDestroyAIPP**。

函数原型

aclmdlAIPP *aclmdlCreateAIPP(uint64_t batchSize)

参数说明

参数名	输入/输出	说明
batchSize	输入	原始模型的batch size。

返回值说明

返回aclmdlAIPP类型的指针。

11.27.60.2 aclmdlSetAIPPCscParams

函数功能

动态AIPP场景下，设置CSC色域转换相关的参数，若色域转换开关关闭，则调用该接口设置以下参数无效。

```
YUV转BGR：
| B |   | cscMatrixR0C0 cscMatrixR0C1 cscMatrixR0C2 | | Y - cscInputBiasR0 |
| G | = | cscMatrixR1C0 cscMatrixR1C1 cscMatrixR1C2 | | U - cscInputBiasR1 | >> 8
| R |   | cscMatrixR2C0 cscMatrixR2C1 cscMatrixR2C2 | | V - cscInputBiasR2 |
BGR转YUV：
| Y |   | cscMatrixR0C0 cscMatrixR0C1 cscMatrixR0C2 | | B |        | cscOutputBiasR0 |
| U | = | cscMatrixR1C0 cscMatrixR1C1 cscMatrixR1C2 | | G | >> 8 + | cscOutputBiasR1 |
| V |   | cscMatrixR2C0 cscMatrixR2C1 cscMatrixR2C2 | | R |        | cscOutputBiasR2 |
```

色域转换参数值与转换前图片的格式、转换后图片的格式强相关，您可以参考《**ATC工具使用指南（开放态）**》中的“**色域转换配置说明**”，根据转换前图片格式、转换后图片格式来配置色域转换参数。如果手册中列举的图片格式不满足要求，您需自行根据实际需求配置色域转换参数。

约束说明

如果通过**aclmdlSetAIPPInputFormat**接口设置的原始图像格式为YUV400，则不支持通过本接口设置色域转换参数。

函数原型

aclError aclmdlSetAIPPCscParams(aclmdlAIPP *aippParmsSet, int8_t cscSwitch,

int16_t cscMatrixR0C0, int16_t cscMatrixR0C1, int16_t cscMatrixR0C2,

int16_t cscMatrixR1C0, int16_t cscMatrixR1C1,int16_t cscMatrixR1C2,

int16_t cscMatrixR2C0, int16_t cscMatrixR2C1, int16_t cscMatrixR2C2,

uint8_t cscOutputBiasR0, uint8_t cscOutputBiasR1, uint8_t cscOutputBiasR2,

uint8_t cscInputBiasR0, uint8_t cscInputBiasR1, uint8_t cscInputBiasR2)

参数说明

参数名	输入/输出	说明
aippParmsSet	输出	动态AIPP参数对象的指针。 需提前调用**aclmdlCreateAIPP**接口创建aclmdlAIPP类型的数据。

参数名	输入/输出	说明
csc_switch	输入	色域转换开关，取值范围： ● 0：关闭色域转换开关，设置为0时，则设置以下参数无效 ● 1：打开色域转换开关
cscMatrixR0C0	输入	色域转换矩阵参数。 取值范围：[-32677 ,32676]
cscMatrixR0C1	输入	色域转换矩阵参数。 取值范围：[-32677 ,32676]
cscMatrixR0C2	输入	色域转换矩阵参数。 取值范围：[-32677 ,32676]
cscMatrixR1C0	输入	色域转换矩阵参数。 取值范围：[-32677 ,32676]
cscMatrixR1C1	输入	色域转换矩阵参数。 取值范围：[-32677 ,32676]
cscMatrixR1C2	输入	色域转换矩阵参数。 取值范围：[-32677 ,32676]
cscMatrixR2C0	输入	色域转换矩阵参数。 取值范围：[-32677 ,32676]
cscMatrixR2C1	输入	色域转换矩阵参数。 取值范围：[-32677 ,32676]
cscMatrixR2C2	输入	色域转换矩阵参数。 取值范围：[-32677 ,32676]
cscOutputBiasR0	输入	RGB转YUV时的输出偏移。 取值范围：[0, 255]
cscOutputBiasR1	输入	RGB转YUV时的输出偏移。 取值范围：[0, 255]
cscOutputBiasR2	输入	RGB转YUV时的输出偏移。 取值范围：[0, 255]
cscInputBiasR0	输入	YUV转RGB时的输入偏移。 取值范围：[0, 255]

参数名	输入/输出	说明
cscInputBiasR1	输入	YUV转RGB时的输入偏移。 取值范围：[0, 255]
cscInputBiasR2	输入	YUV转RGB时的输入偏移。 取值范围：[0, 255]

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.60.3 aclmdlSetAIPPInputFormat

函数功能

动态AIPP场景下，必须设置原始输入图像的格式。

函数原型

aclError aclmdlSetAIPPInputFormat(aclmdlAIPP *aippParmsSet, aclAippInputFormat inputFormat)

参数说明

参数名	输入/输出	说明
aippParmsSet	输出	动态AIPP参数对象的指针。 需提前调用**aclmdlCreateAIPP**接口创建aclmdlAIPP类型的数据。
inputFormat	输入	表示原始输入图像的格式。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.60.4 aclmdlSetAIPPRbuvSwapSwitch

函数功能

动态AIPP场景下，设置是否交换R通道与B通道、或者是否交换U通道与V通道。

函数原型

aclError aclmdlSetAIPPRbuvSwapSwitch(aclmdlAIPP *aippParmsSet, int8_t rbuvSwapSwitch)

参数说明

参数名	输入/输出	说明
aippParmsSet	输出	动态AIPP参数对象的指针。 需提前调用aclmdlCreateAIPP接口创建aclmdlAIPP类型的数据。
rbuvSwapSwitch	输入	表示是否交换R通道与B通道、或者是否交换U通道与V通道的开关，取值范围： ● 0：不交换 ● 1：交换

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.60.5 aclmdlSetAIPPAxSwapSwitch

函数功能

动态AIPP场景下，设置RGBA->ARGB或者YUVA->AYUV的交换开关。

函数原型

aclError aclmdlSetAIPPAxSwapSwitch(aclmdlAIPP *aippParmsSet, int8_t axSwapSwitch)

参数说明

参数名	输入/输出	说明
aippParmsSet	输出	动态AIPP参数对象的指针。 需提前调用aclmdlCreateAIPP接口创建aclmdlAIPP类型的数据。
axSwapSwitch	输入	表示RGBA->ARGB或者YUVA->AYUV的交换开关，取值范围： ● 0：不交换 ● 1：交换

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.60.6 aclmdlSetAIPPSrcImageSize

函数功能

动态AIPP场景下，必须设置原始图片的宽和高。

函数原型

aclError aclmdlSetAIPPSrcImageSize(aclmdlAIPP *aippParmsSet, int32_t srcImageSizeW, int32_t srcImageSizeH)

参数说明

参数名	输入/输出	说明
aippParmsSet	输出	动态AIPP参数对象的指针。 需提前调用aclmdlCreateAIPP接口创建aclmdlAIPP类型的数据。
srcImageSizeW	输入	原始图片的宽，对于YUV420SP_U8或YUV422SP_U8或YUYV_U8类型的图像，要求srcImageSizeW取值是偶数。 取值范围： [2,4096]
srcImageSizeH	输入	原始图片的高，对于YUV420SP_U8类型的图像，要求srcImageSizeH取值是偶数。 取值范围： [1,4096]

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.60.7 aclmdlSetAIPPScfParams

函数功能

动态AIPP场景下，设置缩放相关的参数。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

aclError aclmdlSetAIPPScfParams(aclmdlAIPP *aippParmsSet, int8_t scfSwitch,

int32_t scfInputSizeW, int32_t scfInputSizeH,

int32_t scfOutputSizeW, int32_t scfOutputSizeH,

uint64_t batchIndex)

约束说明

缩放比例scfOutputSizeW/scfInputSizeW∈[1/16,16]、scfOutputSizeH/scfInputSizeH∈[1/16,16]。

参数说明

参数名	输入/输出	说明
aippParmsSet	输出	动态AIPP参数对象的指针。 需提前调用**aclmdlCreateAIPP**接口创建aclmdlAIPP类型的数据。
scfSwitch	输入	是否对图片执行缩放操作，取值范围： ● 0：不执行缩放操作，设置为0时，则设置scfInputSizeW、scfInputSizeH、scfOutputSizeW、scfOutputSizeH参数无效 ● 1：执行缩放操作
scfInputSizeW	输入	缩放前图片的宽。 取值范围：[16,4096] 若开启了抠图功能，则缩放前图片的宽与**抠图区域的宽度**保持一致；若未开启抠图功能，则缩放前图片的宽与**原始图片的宽**保持一致。
scfInputSizeH	输入	缩放前图片的高。 取值范围：[16,4096] 若开启了抠图功能，则缩放前图片的高与**抠图区域的高度**保持一致；若未开启抠图功能，则缩放前图片的高与**原始图片的高**保持一致。
scfOutputSizeW	输入	缩放后图片的宽。 取值范围：[16,1920]
scfOutputSizeH	输入	缩放后图片的高。 取值范围：[16,4096]
batchIndex	输入	指定对第几个Batch上的图片执行缩放操作。 取值范围：[0,batchSize) batchSize是在调用**aclmdlCreateAIPP**接口创建aclmdlAIPP类型的数据时设置。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.60.8 aclmdlSetAIPPCropParams

函数功能

动态AIPP场景下，设置抠图相关的参数。

函数原型

aclError aclmdlSetAIPPCropParams(aclmdlAIPP *aippParmsSet, int8_t cropSwitch,

int32_t cropStartPosW, int32_t cropStartPosH,

int32_t cropSizeW, int32_t cropSizeH,

uint64_t batchIndex)

约束说明

若开启抠图功能，则通过**aclmdlSetAIPPSrcImageSize**接口设置的参数与通过aclmdlSetAIPPCropParams接口设置的参数之间必须满足以下公式：

- cropSizeW+cropStartPosW ≤ srcImageSizeW
- cropSizeH+cropStartPosH ≤ srcImageSizeH

参数说明

参数名	输入/输出	说明
aippParmsSet	输出	动态AIPP参数对象的指针。 需提前调用**aclmdlCreateAIPP**接口创建aclmdlAIPP类型的数据。
cropSwitch	输入	是否对图片执行抠图操作，取值范围： ● 0：不执行抠图操作，设置为0时，则设置cropStartPosW、cropStartPosH、cropSizeW、cropSizeH参数无效 ● 1：执行抠图操作
cropStartPosW	输入	抠图时，坐标点起始位置在图中横向的坐标。 对于YUV420SP_U8格式的图像，参数取值要求是偶数。 取值范围：[0,4095]

参数名	输入/输出	说明
cropStartPosH	输入	抠图时，坐标点起始位置在图中纵向的坐标。 对于YUV420SP_U8格式的图像，参数取值要求是偶数。 取值范围：[0,4095]
cropSizeW	输入	抠图区域的宽度。 取值范围：[1,4096]
cropSizeH	输入	抠图区域的高度。 取值范围：[1,4096]
batchIndex	输入	指定对第几个Batch上的图片执行抠图操作。 取值范围：[0,batchSize) batchSize是在调用**aclmdlCreateAIPP**接口创建aclmdlAIPP类型的数据时设置。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.60.9 aclmdlSetAIPPPaddingParams

函数功能

动态AIPP场景下，设置补边相关的参数。

函数原型

aclError aclmdlSetAIPPPaddingParams(aclmdlAIPP *aippParmsSet, int8_t paddingSwitch,

int32_t paddingSizeTop, int32_t paddingSizeBottom,

int32_t paddingSizeLeft, int32_t paddingSizeRight,

uint64_t batchIndex)

约束说明

补边之后，图片的宽必须小于等于1080。

参数说明

参数名	输入/输出	说明
aippParmsSet	输出	动态AIPP参数对象的指针。 需提前调用aclmdlCreateAIPP接口创建aclmdlAIPP类型的数据。
paddingSwitch	输入	是否对图片执行补边操作，取值范围： ● 0：不执行补边操作，设置为0时，则设置paddingSizeTop、paddingSizeBottom、paddingSizeLeft、paddingSizeRight参数无效 ● 1：执行补边操作
paddingSizeTop	输入	在图片上方填充的值。 取值范围：[0, 32]
paddingSizeBottom	输入	在图片下方填充的值。 取值范围：[0, 32]
paddingSizeLeft	输入	在图片左方填充的值。 取值范围：[0, 32]
paddingSizeRight	输入	在图片右方填充的值。 取值范围：[0, 32]
batchIndex	输入	指定对第几个Batch上的图片执行补边操作。 取值范围：[0,batchSize) batchSize是在调用aclmdlCreateAIPP接口创建aclmdlAIPP类型的数据时设置。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.60.10 aclmdlSetAIPPDtcPixelMean

函数功能

动态AIPP场景下，设置通道的均值。

函数原型

```
aclError aclmdlSetAIPPDtcPixelMean(aclmdlAIPP *aippParmsSet,
int16_t dtcPixelMeanChn0,
int16_t dtcPixelMeanChn1,
int16_t dtcPixelMeanChn2,
```

int16_t dtcPixelMeanChn3,

uint64_t batchIndex)

参数说明

参数名	输入/输出	说明
aippParmsSet	输出	动态AIPP参数对象的指针。 需提前调用**aclmdlCreateAIPP**接口创建aclmdlAIPP类型的数据。
dtcPixelMeanChn0	输入	通道0的均值。 取值范围：[0, 255]
dtcPixelMeanChn1	输入	通道1的均值。 取值范围：[0, 255]
dtcPixelMeanChn2	输入	通道2的均值。 取值范围：[0, 255]
dtcPixelMeanChn3	输入	通道3的均值。 如果只有3个通道，将该参数设置为0。 取值范围：[0, 255]
batchIndex	输入	指定对第几个Batch上的图片设置通道均值。 取值范围：[0,batchSize) batchSize是在调用**aclmdlCreateAIPP**接口创建aclmdlAIPP类型的数据时设置。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.60.11 aclmdlSetAIPPDtcPixelMin

函数功能

动态AIPP场景下，设置通道的最小值。

函数原型

aclError aclmdlSetAIPPDtcPixelMin(aclmdlAIPP *aippParmsSet,

float dtcPixelMinChn0,

float dtcPixelMinChn1,

float dtcPixelMinChn2,

float dtcPixelMinChn3,

uint64_t batchIndex)

参数说明

参数名	输入/输出	说明
aippParmsSet	输出	动态AIPP参数对象的指针。 需提前调用**aclmdlCreateAIPP**接口创建aclmdlAIPP类型的数据。
dtcPixelMinChn0	输入	通道0的最小值。 取值范围：[0, 255]
dtcPixelMinChn1	输入	通道1的最小值。 取值范围：[0, 255]
dtcPixelMinChn2	输入	通道2的最小值。 取值范围：[0, 255]
dtcPixelMinChn3	输入	通道3的最小值。如果只有3个通道，将该参数设置为0。 取值范围：[0, 255]
batchIndex	输入	指定对第几个Batch上的图片设置通道最小值。 取值范围：[0,batchSize) batchSize是在调用**aclmdlCreateAIPP**接口创建aclmdlAIPP类型的数据时设置。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.60.12 aclmdlSetAIPPPixelVarReci

函数功能

动态AIPP场景下，设置通道的方差。

函数原型

aclError aclmdlSetAIPPPixelVarReci(aclmdlAIPP *aippParmsSet,

float dtcPixelVarReciChn0,

float dtcPixelVarReciChn1,

float dtcPixelVarReciChn2,

float dtcPixelVarReciChn3,

uint64_t batchIndex)

参数说明

参数名	输入/输出	说明
aippParmsSet	输出	动态AIPP参数对象的指针。 需提前调用**aclmdlCreateAIPP**接口创建aclmdlAIPP类型的数据。
dtcPixelVarReciChn0	输入	通道0的方差的倒数，默认值为1.0。 取值范围：[-65504, 65504]
dtcPixelVarReciChn1	输入	通道1的方差的倒数，默认值为1.0。 取值范围：[-65504, 65504]
dtcPixelVarReciChn2	输入	通道2的方差的倒数，默认值为1.0。 取值范围：[-65504, 65504]
dtcPixelVarReciChn3	输入	通道3的方差的倒数，默认值为1.0。如果只有3个通道，则将该参数设置为1.0。 取值范围：[-65504, 65504]
batchIndex	输入	指定对第几个Batch上的图片设置通道的方差。 取值范围：[0,batchSize) batchSize是在调用**aclmdlCreateAIPP**接口创建aclmdlAIPP类型的数据时设置。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.60.13 aclmdlDestroyAIPP

函数功能

销毁通过**aclmdlCreateAIPP**接口创建的aclmdlAIPP类型的数据。

函数原型

aclError aclmdlDestroyAIPP(const aclmdlAIPP *aippParmsSet)

参数说明

参数名	输入/输出	说明
aippParmsSet	输入	待销毁的aclmdlAIPP类型的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.61 aclopHandle

11.27.61.1 aclopCreateHandle

函数功能

创建一个执行算子的handle。

如需销毁handle，请参见**aclopDestroyHandle**。

约束说明

对于算子有constant输入的场景，如果未调用**aclSetTensorConst**接口设置constant输入，则需调用**aclSetTensorPlaceMent**设置TensorDesc的placement属性，将memType设置为Host内存。

函数原型

```
aclError aclopCreateHandle(const char *opType,
int numInputs,
const aclTensorDesc *const inputDesc[],
int numOutputs,
const aclTensorDesc *const outputDesc[],
const aclopAttr *opAttr,
aclopHandle **handle);
```

参数说明

参数名	输入/输出	说明
opType	输入	算子类型名称的指针。
numInputs	输入	算子输入tensor的数量。
inputDesc	输入	算子输入tensor描述的指针数组。
numOutputs	输入	算子输出tensor的数量。
outputDesc	输入	算子输出tensor描述的指针数组。
opAttr	输入	算子属性的指针。
handle	输出	“aclopHandle数据指针”的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.61.2 aclopDestroyHandle

函数功能

销毁通过**aclopCreateHandle**接口创建的执行算子的handle。

函数原型

void aclopDestroyHandle(aclopHandle *handle)

参数说明

参数名	输入/输出	说明
handle	输入	待销毁的aclopHandle类型的指针。

返回值说明

无

11.27.62 aclDataBuffer

11.27.62.1 aclCreateDataBuffer

函数功能

创建aclDataBuffer类型的数据，该数据类型用于描述内存地址、大小等内存信息。

如需销毁aclDataBuffer类型的数据，请参见**aclDestroyDataBuffer**。

函数原型

aclDataBuffer *aclCreateDataBuffer(void *data, size_t size)

参数说明

参数名	输入/输出	说明
data	输入	存放数据内存地址的指针。data参数支持传入nullptr，表示创建一个空的数据类型，此时size参数值无效。 该内存需由用户自行管理，调用**aclrtMalloc**接口/**aclrtFree**接口申请/释放内存，或调用**aclrtMallocHost**接口/**aclrtFreeHost**接口申请/释放内存。

参数名	输入/输出	说明
size	输入	内存大小，单位Byte。 如果用户需要使用空tensor，则在申请内存时，内存大小最小为1Byte，以保障后续业务正常运行。

返回值说明

返回aclDataBuffer类型的指针。

11.27.62.2 aclDestroyDataBuffer

函数功能

销毁通过**aclCreateDataBuffer**接口创建的aclDataBuffer类型的数据。

此处仅销毁aclDataBuffer类型的数据，调用**aclCreateDataBuffer**接口创建aclDataBuffer类型数据时传入的data的内存需由用户自行释放。

函数原型

aclError aclDestroyDataBuffer(const aclDataBuffer *dataBuffer)

参数说明

参数名	输入/输出	说明
dataBuffer	输入	待销毁的aclDataBuffer类型的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.62.3 aclGetDataBufferAddr

函数功能

获取aclDataBuffer类型中的数据的内存地址。

函数原型

void *aclGetDataBufferAddr(const aclDataBuffer *dataBuffer)

参数说明

参数名	输入/输出	说明
dataBuffer	输入	aclDataBuffer类型的指针。 需提前调用**aclCreateDataBuffer**接口创建aclDataBuffer类型的数据。

返回值说明

返回aclDataBuffer类型中的数据的内存地址。

11.27.62.4 aclGetDataBufferSize

须知

此接口后续版本会废弃，请使用**aclGetDataBufferSizeV2**接口。

函数功能

获取aclDataBuffer类型中数据的内存大小，单位Byte。

函数原型

uint32 aclGetDataBufferSize(const aclDataBuffer *dataBuffer)

参数说明

参数名	输入/输出	说明
dataBuffer	输入	aclDataBuffer类型的指针。 需提前调用**aclCreateDataBuffer**接口创建aclDataBuffer类型的数据。

返回值说明

aclDataBuffer类型中数据的内存大小。

11.27.62.5 aclGetDataBufferSizeV2

函数功能

获取aclDataBuffer类型中数据的内存大小，单位Byte。

函数原型

size_t aclGetDataBufferSizeV2(const aclDataBuffer *dataBuffer)

参数说明

参数名	输入/输出	说明
dataBuffer	输入	aclDataBuffer类型的指针。 需提前调用**aclCreateDataBuffer**接口创建aclDataBuffer类型的数据。

返回值说明

aclDataBuffer类型中数据的内存大小。

11.27.62.6 aclUpdateDataBuffer

函数功能

更新aclDataBuffer中数据的内存及大小。

更新aclDataBuffer后，之前aclDataBuffer中存放数据的内存如果不使用，需及时释放，否则可能会导致内存泄漏。

函数原型

aclError aclUpdateDataBuffer(aclDataBuffer *dataBuffer, void *data, size_t size)

参数说明

参数名	输入/输出	说明
dataBuffer	输入	aclDataBuffer类型的指针。 需提前调用**aclCreateDataBuffer**接口创建aclDataBuffer类型的数据。 该内存需由用户自行管理，调用**aclrtMalloc**接口/**aclrtFree**接口申请/释放内存，或调用**aclrtMallocHost**接口/**aclrtFreeHost**接口申请/释放内存。
data	输入	存放数据内存地址的指针。
size	输入	内存大小，单位Byte。 如果用户需要使用空tensor，则在申请内存时，内存大小最小为1Byte，以保障后续业务正常运行。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.63 aclmdlDataset

11.27.63.1 aclmdlCreateDataset

函数功能

创建aclmdlDataset类型的数据，该数据类型用于描述模型推理时的输入数据、输出数据，模型可能存在多个输入、多个输出，每个输入/输出的内存地址、内存大小用**aclDataBuffer**类型的数据来描述。

如需销毁aclmdlDataset类型的数据，请参见**aclmdlDestroyDataset**。

函数原型

aclmdlDataset *aclmdlCreateDataset()

参数说明

无

返回值说明

返回aclmdlDataset类型的指针。

11.27.63.2 aclmdlDestroyDataset

函数功能

销毁通过**aclmdlCreateDataset**接口创建的aclmdlDataset类型的数据。

函数原型

aclError aclmdlDestroyDataset(const aclmdlDataset *dataset)

参数说明

参数名	输入/输出	说明
dataset	输入	待销毁的aclmdlDataset类型的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.63.3 aclmdlAddDatasetBuffer

函数功能

向aclmdlDataset中增加aclDataBuffer。

函数原型

aclError aclmdlAddDatasetBuffer(aclmdlDataset *dataset, aclDataBuffer *dataBuffer)

参数说明

参数名	输入/输出	说明
dataset	输出	待增加aclDataBuffer的aclmdlDataset地址指针。 需提前调用**aclmdlCreateDataset**接口创建aclmdlDataset类型的数据。
dataBuffer	输入	待增加的aclDataBuffer地址指针。 需提前调用**aclCreateDataBuffer**接口创建aclDataBuffer类型的数据。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.63.4 aclmdlGetDatasetNumBuffers

函数功能

获取aclmdlDataset中aclDataBuffer的个数。

函数原型

size_t aclmdlGetDatasetNumBuffers(const aclmdlDataset *dataset)

参数说明

参数名	输入/输出	说明
dataset	输入	aclmdlDataset类型的指针。 需提前调用**aclmdlCreateDataset**接口创建aclmdlDataset类型的数据。

返回值说明

aclDataBuffer的个数。

11.27.63.5 aclmdlGetDatasetBuffer

函数功能

获取aclmdlDataset中的第n个aclDataBuffer。

函数原型

aclDataBuffer* aclmdlGetDatasetBuffer(const aclmdlDataset *dataset, size_t index)

参数说明

参数名	输入/输出	说明
dataset	输入	aclmdlDataset类型的指针。 需提前调用**aclmdlCreateDataset**接口创建aclmdlDataset类型的数据。
index	输入	表明获取的是第几个aclDataBuffer。

返回值说明

- 获取成功，返回aclDataBuffer的地址。
- 获取失败返回空地址。

11.27.63.6 aclmdlSetDatasetTensorDesc

函数功能

如果模型输入或输出的Shape是动态的，在模型执行之前调用本接口设置模型输入或输出的tensor描述信息。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

对同一个模型，**aclmdlSetDynamicBatchSize**接口、**aclmdlSetDynamicHWSize**接口、**aclmdlSetInputDynamicDims**接口、aclmdlSetDatasetTensorDesc接口，只能调用其中一个接口。

函数原型

aclError aclmdlSetDatasetTensorDesc(aclmdlDataset *dataset, aclTensorDesc *tensorDesc, size_t index)

参数说明

参数名	输入/输出	说明
dataset	输出	待增加aclTensorDesc的aclmdlDataset地址指针，表示模型执行的输入或输出数据结构。 需提前调用**aclmdlCreateDataset**接口创建aclmdlDataset类型的数据，再调用**aclmdlAddDatasetBuffer**接口向aclmdlDataset中增加aclDataBuffer。
tensorDesc	输入	待增加的aclTensorDesc地址指针，表示模型执行时对应的输入或输出的tensor描述。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型的数据，当前只有设置模型输入、输出tensor描述信息中的维度信息有效（对应**aclCreateTensorDesc**接口中的代表维度个数的numDims参数、代表维度大小的dims参数），设置数据类型、Format无效。 此处设置的维度个数、维度大小必须在模型构建时设置的输入Shape范围内。
index	输入	表示第几个输入或输出的序号。 模型存在多个输入、输出时，为避免序号出错，可以先调用**aclmdlGetInputNameByIndex**、**aclmdlGetOutputNameByIndex**接口获取输入、输出的名称，根据输入、输出名称所对应的index来设置。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.63.7 aclmdlGetDatasetTensorDesc

函数功能

如果模型输入或输出的Shape是动态的，模型在模型执行之后可以调用本接口从aclmdlDataset类型的数据（该类型用于描述模型推理时的输入数据、输出数据）中获取指定的输入或者输出的Tensor描述信息。

典型场景举例：如果模型输入Shape是动态的，在模型执行之前调**aclmdlSetDatasetTensorDesc**设置该输入的tensor描述信息，在模型执行之后，调用aclmdlGetDatasetTensorDesc接口获取模型动态输出的Tensor描述信息。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

aclTensorDesc *aclmdlGetDatasetTensorDesc(const aclmdlDataset *dataset, size_t index)

参数说明

参数名	输入/输出	说明
dataset	输入	模型执行的输入或输出数据指针。
index	输入	表示第几个输入或输出的序号。 模型存在多个输入、输出时，为避免序号出错，可以先调用**aclmdlGetInputNameByIndex**、**aclmdlGetOutputNameByIndex**接口获取输入、输出的名称，根据输入、输出名称所对应的index来设置。

返回值说明

返回指定输入或输出的tensor描述信息。

11.27.64 aclmdlDesc

11.27.64.1 aclmdlCreateDesc

函数功能

创建aclmdlDesc类型的数据，表示模型描述信息。

如需销毁aclmdlDesc类型的数据，请参见**aclmdlDestroyDesc**。

函数原型

aclmdlDesc* aclmdlCreateDesc()

参数说明

无

返回值说明

返回aclmdlDesc类型的指针。

11.27.64.2 aclmdlDestroyDesc

函数功能

销毁通过**aclmdlCreateDesc**接口创建的aclmdlDesc类型的数据。

函数原型

aclError aclmdlDestroyDesc(aclmdlDesc *modelDesc)

参数说明

参数名	输入/输出	说明
modelDesc	输入	待销毁的aclmdlDesc类型的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.64.3 aclmdlGetDesc

函数功能

根据模型ID获取该模型的模型描述信息。

函数原型

aclError aclmdlGetDesc(aclmdlDesc *modelDesc, uint32_t modelId)

参数说明

参数名	输入/输出	说明
modelDesc	输出	aclmdlDesc类型的指针。 需提前调用**aclmdlCreateDesc**接口创建aclmdlDesc类型的数据。
modelId	输入	模型ID。 调用**aclmdlLoadFromFile**接口/**aclmdlLoadFromMem**接口/**aclmdlLoadFromFileWithMem**接口/**aclmdlLoadFromMemWithMem**接口加载模型成功后，会返回模型ID。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.64.4 aclmdlGetNumInputs

函数功能

根据模型描述信息获取模型的输入个数。

函数原型

size_t aclmdlGetNumInputs(aclmdlDesc *modelDesc)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型的指针。 需提前调用**aclmdlCreateDesc**接口创建aclmdlDesc类型的数据。

返回值说明

模型的输入个数。

11.27.64.5 aclmdlGetNumOutputs

函数功能

根据模型描述信息获取模型的输出个数。

函数原型

size_t aclmdlGetNumOutputs(aclmdlDesc *modelDesc)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型的指针。 需提前调用**aclmdlCreateDesc**接口创建aclmdlDesc类型的数据。

返回值说明

模型的输出个数。

11.27.64.6 aclmdlGetInputSizeByIndex

函数功能

根据模型描述信息获取指定输入的大小，单位为Byte。

约束说明

如果模型输入的Shape是动态的、且维度的取值为-1（表示此维度可以使用>=1的任意取值），则通过本接口获取的大小为0，用户需根据实际数据占用的内存大小来申请内存。

函数原型

size_t aclmdlGetInputSizeByIndex(aclmdlDesc *modelDesc, size_t index)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型的指针。 需提前调用**aclmdlCreateDesc**接口创建aclmdlDesc类型的数据。
index	输入	指定获取第几个输入的大小，index值从0开始。

返回值说明

针对动态Batch、动态分辨率（宽高）的场景，返回最大档位对应的输入的大小；静态场景下，返回指定输入的大小。单位是Byte。

11.27.64.7 aclmdlGetOutputSizeByIndex

函数功能

根据模型描述信息获取指定输出的大小，单位为Byte。

约束说明

如果通过本接口获取的大小为0，有可能是由于输出Shape的范围不确定，当前支持以下两种处理方式：

- 方式一：**系统内部自行申请对应index的输出内存**，节省内存，但内存数据使用结束后，需由用户释放内存，同时，系统内部申请内存时涉及内存拷贝，可能涉及性能损耗。

  调用aclCreateDataBuffer接口创建存放对应index输出数据的aclDataBuffer类型时，可在data参数处传入nullptr，表示创建一个空的aclDataBuffer类型，然后在模型执行过程中，系统内部自行计算并申请该index输出的内存。
- 方式二：**用户预估输出内存大小，并申请内存**，由用户自行管理内存，但内存大小可能不够或超出，不够时系统会校验报错，超出时会浪费内存。

用户需先根据实际情况预估一块较大的输出内存，在模型执行过程中，系统会校验用户指定的输出内存大小是否符合要求，如果不符合要求，系统会返回报错，并在报错信息中提示具体需要多大的输出内存。您可以通过以下两种方式查看报错：

- 获取应用类日志，查看ERROR级别的报错，如何获取日志请参见《日志参考（开放态）》。
- 在应用程序中调用aclGetRecentErrMsg接口获取报错。

函数原型

size_t aclmdlGetOutputSizeByIndex(aclmdlDesc *modelDesc, size_t index)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型的指针。 需提前调用aclmdlCreateDesc接口创建aclmdlDesc类型的数据。
index	输入	指定获取第几个输出的大小，index值从0开始。

返回值说明

针对动态Batch、动态分辨率（宽高）的场景，返回最大档位对应的输出的大小；静态场景下，返回指定输出的大小。单位是Byte。

11.27.64.8 aclmdlGetInputDims

函数功能

根据模型描述信息获取模型的输入tensor的维度信息。

如果模型中含有静态AIPP配置信息，您可以根据实际需要选择aclmdlGetInputDims接口或aclmdlGetInputDimsV2接口查询维度信息，两者的区别在于：

- 通过aclmdlGetInputDims接口获取的维度信息，各维度的值与输入图像的各维度的值保持一致，详细规则如表11-89所示。
- 通过aclmdlGetInputDimsV2接口获取的维度信息，H维度的值 = 输入图像的H维度值*系数，不同图片格式下C维度的值不同，详细规则如aclmdlGetInputDimsV2接口处的表11-90所示，且各维度值相乘的结果值与通过aclmdlGetInputSizeByIndex接口获取的值保持一致。

函数原型

aclError aclmdlGetInputDims(const aclmdlDesc *modelDesc, size_t index, aclmdlIODims *dims)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型的指针。 需提前调用**aclmdlCreateDesc**接口创建aclmdlDesc类型的数据。
index	输入	指定获取第几个输入的Dims，index值从0开始。
dims	输出	输入维度信息的指针。 针对动态Batch、动态分辨率（宽高）的场景，输入tensor的dims中batch size或宽高为-1，表示其动态可变。例如，输入tensor的format为NCHW，在动态Batch场景下，动态可变的输入tensor的dims为[**-1**,*3,224,224*]；在动态分辨率场景下，动态可变的输入tensor的dims为[*1,3*,**-1**,**-1**]。举例中的斜体部分以实际情况为准。 若tensor的name长度大于127，则在输出的dims.name时，AscendCL会将tensor的name转换为“acl_modelId_*${id}*_input_*${index}*_*${随机字符串}*”格式（如果转换后的tensor的name与模型中已有的tensor的name冲突，则会在转换后的name尾部增加“_*${随机字符串}*”，否则不会增加随机字符串），并在转换后的name与原name之间建立映射关系，用户可调用**aclmdlGetTensorRealName**接口，传入转换后的name，获取原name（若向接口传入原name，则获取的还是原name）；若tensor的name长度小于或等于127，则在输出的dims.name时，按tensor的name输出。

表 11-89 静态 AIPP 场景下的维度定义规则

图像格式	Format参考格式	维度定义规则
YUV420SP_U8	NHWC	n,h,w,c
XRGB8888_U8	NHWC	n,h,w,c
RGB888_U8	NHWC	n,h,w,c
YUV400_U8	NHWC	n,h,w,c
ARGB8888_U8	NHWC	n,h,w,c
YUYV_U8	NHWC	n,h,w,c
YUV422SP_U8	NHWC	n,h,w,c
AYUV444_U8	NHWC	n,h,w,c

返回值说明

返回0表示成功，返回非0表示失败。

11.27.64.9 aclmdlGetInputDimsV2

函数功能

根据模型描述信息获取模型的输入tensor的维度信息。

如果模型中含有静态AIPP配置信息，您可以根据实际需要选择**aclmdlGetInputDims**接口或**aclmdlGetInputDimsV2**接口查询维度信息，两者的区别在于：

- 通过**aclmdlGetInputDims**接口获取的维度信息，各维度的值与输入图像的各维度的值保持一致，详细规则如**表11-89**所示。
- 通过**aclmdlGetInputDimsV2**接口获取的维度信息，H维度的值 = 输入图像的H维度值*系数，不同图片格式下C维度的值不同，详细规则如**aclmdlGetInputDimsV2**接口处的**表11-90**所示，且各维度值相乘的结果值与通过**aclmdlGetInputSizeByIndex**接口获取的值保持一致。

函数原型

aclError aclmdlGetInputDimsV2(const aclmdlDesc *modelDesc, size_t index, aclmdlIODims *dims)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型的指针。 需提前调用**aclmdlCreateDesc**接口创建aclmdlDesc类型的数据。
index	输入	指定获取第几个输入的Dims，index值从0开始。

参数名	输入/输出	说明
dims	输出	输入维度信息的指针。 针对动态Batch、动态分辨率（宽高）的场景，输入tensor的dims中batch size或宽高为-1，表示其动态可变。例如，输入tensor的format为NCHW，在动态Batch场景下，动态可变的输入tensor的dims为[**-1**,*3*,*224*,*224*]；在动态分辨率场景下，动态可变的输入tensor的dims为[*1*,*3*,**-1**,**-1**]。举例中的斜体部分以实际情况为准。 若tensor的name长度大于127，则在输出的dims.name时，AscendCL会将tensor的name转换为“acl_modelId_*${id}*_input_*${index}*_*${随机字符串}*”格式（如果转换后的tensor的name与模型中已有的tensor的name冲突，则会在转换后的name尾部增加“_*${随机字符串}*”，否则不会增加随机字符串），并在转换后的name与原name之间建立映射关系，用户可调用**aclmdlGetTensorRealName**接口，传入转换后的name，获取原name（若向接口传入原name，则获取的还是原name）；若tensor的name长度小于或等于127，则在输出的dims.name时，按tensor的name输出。 针对静态AIPP场景，本接口针对不同格式的图像，对应NHWC的Format格式，当前接口中明确各个维度的定义规则，如**表11-90**所示。

表 11-90 静态 AIPP 场景下的维度定义规则

图像格式	Format参考格式	维度定义规则
YUV420SP_U8	NHWC	n,h*1.5,w,1
XRGB8888_U8	NHWC	n,h,w,4
RGB888_U8	NHWC	n,h,w,3
YUV400_U8	NHWC	n,h,w,1
ARGB8888_U8	NHWC	n,h,w,4
YUYV_U8	NHWC	n,h,w,2
YUV422SP_U8	NHWC	n,h*2,w,1
AYUV444_U8	NHWC	n,h,w,4

返回值说明

返回0表示成功，返回非0表示失败。

11.27.64.10 aclmdlGetOutputDims

函数功能

根据模型描述信息获取指定的模型输出tensor的维度信息。

固定Shape场景下，通过该接口获取指定的模型输出tensor的维度信息。

动态Shape（动态Batch或动态分辨率）场景下，通过该接口获取最大档的维度信息。

函数原型

aclError aclmdlGetOutputDims(const aclmdlDesc *modelDesc, size_t index, aclmdlIODims *dims)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型的指针。 需提前调用**aclmdlCreateDesc**接口创建aclmdlDesc类型的数据。
index	输入	指定获取第几个输出的Dims，index值从0开始。
dims	输出	输出维度信息的指针。 若tensor的name长度大于127，则在输出的dims.name时，系统会将tensor的name转换为“acl_modelId_*${id}*_output_*${index}* _*${随机字符串}*”格式（如果转换后的tensor的name与模型中已有的tensor的name冲突，则会在转换后的name尾部增加“_*${随机字符串}*”，否则不会增加随机字符串），并在转换后的name与原name之间建立映射关系，用户可调用**aclmdlGetTensorRealName**接口，传入转换后的name，获取原name（若向接口传入原name，则获取的还是原name）；若tensor的name长度小于或等于127，则在输出的dims.name时，按tensor的name输出。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.64.11 aclmdlGetInputNameByIndex

函数功能

根据模型描述信息获取模型中指定输入的输入名称。

函数原型

const char *aclmdlGetInputNameByIndex(const aclmdlDesc *modelDesc, size_t index)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型的指针。 需提前调用**aclmdlCreateDesc**接口创建aclmdlDesc类型的数据。
index	输入	指定获取第几个输入的名称，index值从0开始。

返回值说明

返回指定输入的输入名称。

11.27.64.12 aclmdlGetOutputNameByIndex

函数功能

根据模型描述信息获取模型中指定输出的输出算子名称、算子输出边的下标、top名称或输出名称。

函数原型

const char *aclmdlGetOutputNameByIndex(const aclmdlDesc *modelDesc, size_t index)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型的指针。 需提前调用**aclmdlCreateDesc**接口创建aclmdlDesc类型的数据。
index	输入	指定获取第几个输出的输出算子名称、算子输出边下标、top名称或输出名称，index值从0开始。

返回值说明

返回指定输出的输出算子名称、算子输出边的下标、top名称或输出名称。不同原始网络、不同构建模型的方式，调用本接口获取的返回值格式不同。

- Caffe网络

  返回值格式如下，各项之间以冒号分割，如果模型中包含top名称就返回，不包含就不返回：

  输出算子名称:*算子输出边下标*:*top名称*

- TensorFlow网络
  - 使用ATC工具构建om模型的场景下，返回值格式如下，各项之间以冒号分割：

    输出算子名称:*算子输出边下标*

  - 使用Ascend Graph接口构建om模型的场景下，返回值格式如下，各项之间以下划线分割：

    output_*网络输出下标*_*输出算子名称*_*算子输出边下标*

- ONNX网络

  返回值格式如下，各项之间以冒号分割：

  输出算子名称:*算子输出边下标*:*输出名称*

  如果构建om模型时，不指定网络模型中输出算子的名称或网络模型中输出的名称（系统会自动从原始模型中获取输出的名称），或者仅指定网络模型中输出的名称，则返回值中会包含输出名称。

  使用Ascend Graph接口构建om模型时，如果指定了网络模型中输出算子的名称，则返回值格式如下，各项之间以下划线分割：

  output_*网络输出下标*_*输出算子名称*_*算子输出边下标*

11.27.64.13 aclmdlGetInputFormat

函数功能

根据模型描述信息获取模型中指定输入的Format。

函数原型

aclFormat aclmdlGetInputFormat(const aclmdlDesc *modelDesc, size_t index)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型的指针。 需提前调用**aclmdlCreateDesc**接口创建aclmdlDesc类型的数据。
index	输入	指定获取第几个输入的Format，index值从0开始。

返回值说明

返回指定输入的Format。

11.27.64.14 aclmdlGetOutputFormat

函数功能

根据模型描述信息获取模型中指定输出的Format。

函数原型

aclFormat aclmdlGetOutputFormat(const aclmdlDesc *modelDesc, size_t index)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型的指针。 需提前调用**aclmdlCreateDesc**接口创建aclmdlDesc类型的数据。
index	输入	指定获取第几个输出的Format，index值从0开始。

返回值说明

返回指定输出的Format。

11.27.64.15 aclmdlGetInputDataType

函数功能

根据模型描述信息获取模型中指定输入的数据类型。

函数原型

aclDataType aclmdlGetInputDataType(const aclmdlDesc *modelDesc, size_t index)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型的指针。 需提前调用**aclmdlCreateDesc**接口创建aclmdlDesc类型的数据。
index	输入	指定获取第几个输入的数据类型，index值从0开始。

返回值说明

返回指定输入的数据类型。

11.27.64.16 aclmdlGetOutputDataType

函数功能

根据模型描述信息获取模型中指定输出的数据类型。

函数原型

aclDataType aclmdlGetOutputDataType(const aclmdlDesc *modelDesc, size_t index)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型的指针。 需提前调用**aclmdlCreateDesc**接口创建aclmdlDesc类型的数据。
index	输入	指定获取第几个输出的数据类型，index值从0开始。

返回值说明

返回指定输出的数据类型。

11.27.64.17 aclmdlGetInputIndexByName

函数功能

根据模型中的输入名称获取对应输入的索引编号。

函数原型

aclError aclmdlGetInputIndexByName(const aclmdlDesc *modelDesc, const char *name, size_t *index)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型的指针。 需提前调用**aclmdlCreateDesc**接口创建aclmdlDesc类型的数据。

参数名	输入/输出	说明
name	输入	输入名称的指针。
index	输出	输入的索引编号的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.64.18 aclmdlGetOutputIndexByName

函数功能

根据模型中的输出名称获取对应输出的索引编号。

函数原型

aclError aclmdlGetOutputIndexByName(const aclmdlDesc *modelDesc, const char *name, size_t *index)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型的指针。 需提前调用**aclmdlCreateDesc**接口创建aclmdlDesc类型的数据。
name	输入	输出名称的指针。
index	输出	输出的索引编号的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.64.19 aclmdlGetDynamicBatch

函数功能

根据模型描述信息获取模型支持的动态Batch信息。

函数原型

aclError aclmdlGetDynamicBatch(const aclmdlDesc *modelDesc, aclmdlBatch *batch)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型的指针。 需提前调用**aclmdlCreateDesc**接口创建aclmdlDesc类型的数据。

batch	输出	const int ACL_MAX_BATCH_NUM = 128;
		typedef struct aclmdlBatch {
		size_t batchCount; /**模型中支持的batch分档数 */
		uint64_t batch[ACL_MAX_BATCH_NUM]; /**模型中支持的具体分档 */
		} aclmdlBatch;
		batchCount等于0时，表示不支持设置档位信息，以模型中的档位为准。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.64.20 aclmdlGetDynamicHW

函数功能

根据模型描述信息获取模型支持的动态宽高信息。

函数原型

aclError aclmdlGetDynamicHW(const aclmdlDesc *modelDesc, size_t index, aclmdlHW *hw)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型的指针。 需提前调用**aclmdlCreateDesc**接口创建aclmdlDesc类型的数据。
index	输入	预留参数，当前未使用，固定设置为-1。

hw	输出	const int ACL_MAX_HW_NUM = 128;
		typedef struct aclmdlHW {
		size_t hwCount; /**模型中支持的宽高分档数 */
		uint64_t hw[ACL_MAX_HW_NUM][2]; /**模型中支持的具体分档，每组分档中，数组下标为0代表的是高，下标为1代表的是宽 */
		} aclmdlHW;
		hwCount等于0时，表示不支持设置档位信息，以模型中的档位为准。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.64.21 aclmdlGetCurOutputDims

函数功能

根据模型描述信息获取指定的模型输出tensor的实际维度信息。

约束说明

当前仅支持通过本接口获取以下场景中的模型输出tensor的维度信息：

- 通过模型转换设置多档Batch size或分辨率或维度值，实现**动态Batch**或**动态分辨率**或**动态维度（ND格式）**时，如果用户已调用**aclmdlSetDynamicBatchSize**设置Batch、或调用**aclmdlSetDynamicHWSize**接口设置输入图片的宽高、或调用**aclmdlSetInputDynamicDims**接口设置某动态维度的值，则可通过该接口获取指定模型输出tensor的实际维度信息；如果用户未调用**aclmdlSetDynamicBatchSize**接口、或**aclmdlSetDynamicHWSize**接口、或**aclmdlSetInputDynamicDims**接口，则通过该接口可获取最大档的维度信息。
- 固定Shape场景下，通过该接口获取指定的模型输出tensor的维度信息。

函数原型

aclError aclmdlGetCurOutputDims(const aclmdlDesc *modelDesc, size_t index, aclmdlIODims *dims)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型的指针。 需提前调用**aclmdlCreateDesc**接口创建aclmdlDesc类型的数据。
index	输入	指定获取第几个输出的Dims，index值从0开始。
dims	输出	输出实际维度信息的指针。 若tensor的name长度大于127，则在输出的dims.name时，系统会将tensor的name转换为“acl_modelId_*${id}*_output_*${index}* _*${随机字符串}*”格式（如果转换后的tensor的name与模型中已有的tensor的name冲突，则会在转换后的name尾部增加“_*${随机字符串}*”，否则不会增加随机字符串），并在转换后的name与原name之间建立映射关系，用户可调用**aclmdlGetTensorRealName**接口，传入转换后的name，获取原name（若向接口传入原name，则获取的还是原name）；若tensor的name长度小于或等于127，则在输出的dims.name时，按tensor的name输出。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.64.22 aclmdlGetInputDynamicGearCount

函数功能

根据模型描述信息获取模型的输入所支持的动态维度档位数。

约束说明

如果模型构建时没有设置动态维度的分档，那么通过该接口获取的动态维度档位数为0。模型构建的详细说明请参见4.2 模型构建。

函数原型

aclError aclmdlGetInputDynamicGearCount(const aclmdlDesc *modelDesc, size_t index, size_t *gearCount)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型的指针。 需提前调用aclmdlCreateDesc接口创建aclmdlDesc类型的数据。
index	输入	预留参数，当前未使用，固定设置为-1。
gearCount	输出	动态维度档位数的指针。 例如，模型的输入tensor是4维，在模型转换时通过--dynamic_dims参数设置的分档为“1,3,224,224;2,3,224,224;1,3,256,256”,那么通过该接口获取的动态维度档位数为3。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.64.23 aclmdlGetInputDynamicDims

函数功能

根据模型描述信息获取模型的输入所支持的动态维度信息。

约束说明

只有在模型构建时设置了动态维度的分档信息后，才可以调用该接口获取动态维度信息。模型构建的详细说明请参见4.2 模型构建。

例如，模型有三个输入，分别为data(1, 1, 40, -1)，label(1, -1)，mask(-1, -1)， 其中-1表示动态可变。在模型转换时，dynamic_dims参数的配置示例为：--dynamic_dims="20,20,1,1; 40,40,2,2; 80,60,4,4"，则通过本接口获取的动态维度信息为（**aclmdlIODims**结构体内的name暂不使用）：

- 第0档：
  - **aclmdlIODims**结构体内dimCount：8，表示所有输入tensor的维度数量之和
  - **aclmdlIODims**结构体内的dims：“1,1,40,20,1,20,1,1”，表示data(1,1,40,20)+label(1,20)+mask(1,1)
- 第1档：
  - **aclmdlIODims**结构体内dimCount：8，表示所有输入tensor的维度数量之和
  - **aclmdlIODims**结构体内的dims：“1,1,40,40,1,40,2,2”，表示data(1,1,40,40)+label(1,40)+mask(2,2)
- 第2档：
  - **aclmdlIODims**结构体内dimCount：8，表示所有输入tensor的维度数量之和
  - **aclmdlIODims**结构体内的dims：“1,1,40,80,1,60,4,4”，表示data(1,1,40,80)+label(1,60)+mask(4,4)

函数原型

aclError aclmdlGetInputDynamicDims(const aclmdlDesc *modelDesc, size_t index, aclmdlIODims *dims, size_t gearCount)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型的指针。 需提前调用**aclmdlCreateDesc**接口创建aclmdlDesc类型的数据。
index	输入	预留参数，当前未使用，固定设置为-1。
gearCount	输入	模型支持的动态维度档位数，需要先通过**aclmdlGetInputDynamicGearCount**接口获取。
dims	输出	输入的动态维度信息的指针。 dims参数是一个数组，数组中的每个元素指向**aclmdlIODims**结构，**aclmdlIODims**结构体中的dims参数也是也是一个数组，该数组中的每个元素对应每一档中的具体值。 **例如：** aclmdlIODims dims[gearCount]; aclmdlGetInputDynamicDims(model.modelDesc, -1, dims, gearCount);

返回值说明

返回0表示成功，返回非0表示失败。

11.27.64.24 aclmdlGetTensorRealName

函数功能

根据指定名称获取tensor的真实名称。

aclmdlGetTensorRealName接口需要与**aclmdlGetInputDims**/**aclmdlGetInputDimsV2**/**aclmdlGetOutputDims**/**aclmdlGetCurOutputDims**接口配合使用。

函数原型

const char *aclmdlGetTensorRealName(const aclmdlDesc *modelDesc, const char *name)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型的指针。 需提前调用**aclmdlCreateDesc**接口创建aclmdlDesc类型的数据。
name	输入	名称的指针，用于根据该名称获取tensor的真实名称。

返回值说明

返回指向tensor真实名称的指针，该指针的生命周期与modelDesc相同，若modelDesc资源被销毁，则该指针指向的内容也会自动被销毁。

若modelDesc或name为空，则返回nullptr。

11.27.64.25 aclmdlGetOpAttr

函数功能

获取整网中某个模型中某个算子的属性的值。

函数原型

const char *aclmdlGetOpAttr(aclmdlDesc *modelDesc, const char *opName, const char *attr)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型的指针。 需提前调用**aclmdlCreateDesc**接口创建aclmdlDesc类型的数据，再调用**aclmdlGetDesc**接口根据模型ID获取到对应的aclmdlDesc类型的数据。
opName	输入	算子名称的指针。
attr	输入	算子属性的指针。 当前仅支持_datadump_original_op_names属性，用于记录某个算子是由哪些算子融合得到的。通过本接口获取到的_datadump_original_op_names属性值格式为[opName1_len]opName1…..[opNameN_len]opNameN，opNameN_len表示算子名称字符串的长度。 _datadump_original_op_names属性值示例如下，表示某个融合算子由scale2c_branch2c、bn2c_branch2c、res2c_branch2c、res2c、res2c_relu这五个算子融合而成的，算子名称字符串的长度分别为16、13、14、5、10： [16]scale2c_branch2c[13]bn2c_branch2c[14]res2c_branch2c[5]res2c[10]res2c_relu

返回值说明

返回属性值的字符串指针，该指针的生命周期与modelDesc相同，若modelDesc资源被销毁，则该指针指向的内容也会自动被销毁。若opName或者attr属性不存在、或者attr属性值为空，均返回指向空字符串的指针。

若调用该接口失败，则返回nullptr。

11.27.64.26 aclmdlGetDescFromFile

函数功能

根据模型文件获取该模型的模型描述信息。

Atlas 200/300/500 推理产品不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

aclError aclmdlGetDescFromFile (aclmdlDesc *modelDesc, const char *modelPath)

约束说明

通过本接口获取到的模型描述信息，无法应用于**aclmdlGetOpAttr**接口。

参数说明

参数名	输入/输出	说明
modelDesc	输出	aclmdlDesc类型的指针。 需提前调用**aclmdlCreateDesc**接口创建aclmdlDesc类型的数据。
modelPath	输入	离线模型文件路径的指针，路径中包含文件名。运行程序（APP）的用户需要对该存储路径有访问权限。 此处的离线模型文件是适配昇腾AI处理器的离线模型，即*.om文件。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.64.27 aclmdlGetDescFromMem

函数功能

从内存获取该模型的模型描述信息。

Atlas 200/300/500 推理产品不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

aclError aclmdlGetDescFromMem(aclmdlDesc *modelDesc, const void *model, size_t modelSize)

约束说明

通过本接口获取到的模型描述信息，无法应用于**aclmdlGetOpAttr**接口。

参数说明

参数名	输入/输出	说明
modelDesc	输出	aclmdlDesc类型的指针。 需提前调用**aclmdlCreateDesc**接口创建aclmdlDesc类型的数据。

参数名	输入/输出	说明
model	输入	存放模型数据的内存地址指针。 应用运行在Host时，此处需申请Host上的内存；应用运行在Device时，此处需申请Device上的内存。内存申请接口请参见**11.11 内存管理**。
modelSize	输入	内存中的模型数据长度，单位Byte。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.65 aclTensorDesc

11.27.65.1 aclCreateTensorDesc

函数功能

创建aclTensorDesc类型的数据，该数据类型用于描述tensor的数据类型、shape、format等信息。

如需销毁aclTensorDesc类型的数据，请参见**aclDestroyTensorDesc**。

函数原型

aclTensorDesc *aclCreateTensorDesc(aclDataType dataType,

int numDims,

const int64_t *dims,

aclFormat format)

参数说明

参数名	输入/输出	说明
dataType	输入	tensor描述的数据类型。
numDims	输入	tensor描述shape的维度个数。 numDims存在以下约束： ● numDims必须大于或等于0； ● numDims>0时，numDims的值必须与dims数组的长度保持一致； ● numDims=0时，系统不使用dims数组值，dims参数值无效。

参数名	输入/输出	说明
dims	输入	tensor描述维度大小的指针。 dims是一个数组，数组中的每个元素表示tensor中每个维度的大小，如果数组中某个元素的值为0，则为空tensor。 如果用户需要使用空tensor，则在申请内存时，内存大小最小为1Byte，以保障后续业务正常运行。
format	输入	tensor描述的format。

返回值说明

返回aclTensorDesc类型的指针。

11.27.65.2 aclDestroyTensorDesc

函数功能

销毁aclTensorDesc类型的数据。

函数原型

void aclDestroyTensorDesc(const aclTensorDesc *desc)

参数说明

参数名	输入/输出	说明
desc	输入	待销毁的aclTensorDesc类型的指针。

返回值说明

无

11.27.65.3 aclGetTensorDescType

函数功能

获取tensor描述中的数据类型。

函数原型

aclDataType aclGetTensorDescType(const aclTensorDesc *desc)

参数说明

参数名	输入/输出	说明
desc	输入	aclTensorDesc类型的指针。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。

返回值说明

返回指定tensor描述的数据类型。

11.27.65.4 aclGetTensorDescFormat

函数功能

获取tensor描述中的format。

函数原型

aclFormat aclGetTensorDescFormat(const aclTensorDesc *desc)

参数说明

参数名	输入/输出	说明
desc	输入	aclTensorDesc类型的指针。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。

返回值说明

返回指定tensor描述的format。

11.27.65.5 aclGetTensorDescSize

函数功能

获取tensor数据占用的空间大小。

函数原型

size_t aclGetTensorDescSize(const aclTensorDesc *desc)

参数说明

参数名	输入/输出	说明
desc	输入	aclTensorDesc类型的指针。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。

返回值说明

返回指定tensor描述占用的空间大小。

11.27.65.6 aclGetTensorDescElementCount

函数功能

获取tensor描述中的元素个数。

函数原型

size_t aclGetTensorDescElementCount(const aclTensorDesc *desc)

参数说明

参数名	输入/输出	说明
desc	输入	aclTensorDesc类型的指针。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。

返回值说明

返回指定tensor描述中的元素个数。

11.27.65.7 aclGetTensorDescNumDims

函数功能

获取tensor描述中shape的维度个数。

函数原型

size_t aclGetTensorDescNumDims(const aclTensorDesc *desc)

参数说明

参数名	输入/输出	说明
desc	输入	aclTensorDesc类型的指针。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。

返回值说明

返回tensor描述中shape的维度个数。若返回ACL_UNKNOWN_RANK（#define ACL_UNKNOWN_RANK 0xFFFFFFFFFFFFFFFE）表示动态Shape场景下维度个数未知。

11.27.65.8 aclGetTensorDescDim

须知

此接口后续版本会废弃，请使用**aclGetTensorDescDimV2**接口。

函数功能

获取tensor描述中指定维度的大小。

函数原型

int64_t aclGetTensorDescDim(const aclTensorDesc *desc, size_t index)

参数说明

参数名	输入/输出	说明
desc	输入	aclTensorDesc类型的指针。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。
index	输入	指定获取第几个维度的大小，index值从0开始。 用户调用**aclGetTensorDescNumDims**接口获取shape维度个数，这个Index的取值范围：[0, (shape维度个数-1)]。

返回值说明

返回指定tensor描述中指定维度的大小。

11.27.65.9 aclGetTensorDescDimV2

函数功能

获取tensor描述中指定维度的大小。

约束说明

当aclGetTensorDescNumDims接口的返回值为ACL_UNKNOWN_RANK时，表示动态Shape场景下维度个数未知，则不能调用aclGetTensorDescDimV2接口获取指定维度的大小。

函数原型

aclError aclGetTensorDescDimV2(const aclTensorDesc *desc, size_t index, int64_t *dimSize)

参数说明

参数名	输入/输出	说明
desc	输入	aclTensorDesc类型的指针。 需提前调用aclCreateTensorDesc接口创建aclTensorDesc类型。
index	输入	指定获取第几个维度的大小，index值从0开始。 用户调用aclGetTensorDescNumDims接口获取shape维度个数，这个Index的取值范围：[0, (shape维度个数-1)]。
dimSize	输出	tensor描述中index指定维度大小的指针。 当返回的dimSize参数值为-1时，表示维度的大小是动态的。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.65.10 aclGetTensorDescDimRange

函数功能

获取tensor描述中指定维度的范围，[1,-1]表示全shape范围。

约束说明

当aclGetTensorDescNumDims接口的返回值为ACL_UNKNOWN_RANK时，表示动态Shape场景下维度个数未知，则不能调用aclGetTensorDescDimRange接口获取指定维度的范围。

函数原型

aclError aclGetTensorDescDimRange(const aclTensorDesc *desc, size_t index, size_t dimRangeNum, int64_t *dimRange)

参数说明

参数名	输入/输出	说明
desc	输入	aclTensorDesc类型的指针。 需提前调用aclCreateTensorDesc接口创建aclTensorDesc类型。
index	输入	指定获取第几个维度的大小，index值从0开始。 用户调用aclGetTensorDescNumDims接口获取shape维度个数，这个Index的取值范围：[0, (shape维度个数-1)]。
dimRangeNum	输入	dimRange的长度，该值必须大于等于2。
dimRange	输出	tensor描述中index指定维度的Shape范围。 dimRange是一个数组，数组的第一个元素值表示Shape范围的最小值，第二个元素值表示Shape范围的最大值。该数组中仅前2个元素值有效。 当dimRange数组的值为[1,-1]时，表示全Shape范围。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.65.11 aclSetTensorDescName

函数功能

设置TensorDesc的名称。

函数原型

void aclSetTensorDescName(aclTensorDesc *desc, const char *name)

参数说明

参数名	输入/输出	说明
desc	输出	aclTensorDesc类型的指针。 需提前调用aclCreateTensorDesc接口创建aclTensorDesc类型。

参数名	输入/输出	说明
name	输入	TensorDesc名称的指针。

返回值说明

无

11.27.65.12 aclGetTensorDescName

函数功能

获取TensorDesc的名称。

函数原型

const char *aclGetTensorDescName(aclTensorDesc *desc)

参数说明

参数名	输入/输出	说明
desc	输入	aclTensorDesc类型的指针。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。

返回值说明

返回TensorDesc的名称。

11.27.65.13 aclTransTensorDescFormat

函数功能

按指定dstFormat转换源aclTensorDesc中的format，生成新的目标aclTensorDesc，源aclTensorDesc中的format保持不变。同步接口。不支持该接口，预留接口。

函数原型

aclError aclTransTensorDescFormat(const aclTensorDesc *srcDesc, aclFormat dstFormat, aclTensorDesc **dstDesc)

参数说明

参数名	输入/输出	说明
srcDesc	输入	源aclTensorDesc数据的指针。 需提前调用aclCreateTensorDesc接口创建aclTensorDesc类型。
dstFormat	输入	需要设置的目标format。
dstDesc	输出	“目标aclTensorDesc数据指针”的指针。 需提前调用aclCreateTensorDesc接口创建aclTensorDesc类型。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.65.14 aclSetTensorStorageShape

须知

此接口后续版本会废弃，请使用aclSetTensorShape接口。

函数功能

调用aclCreateTensorDesc接口创建tensor描述信息后，可通过aclSetTensorStorageShape接口设置tensor的实际dims信息，当前主要用于pytorch场景下。

函数原型

aclError aclSetTensorStorageShape(aclTensorDesc *desc, int numDims, const int64_t *dims)

参数说明

参数名	输入/输出	说明
desc	输出	aclTensorDesc类型的指针。 需提前调用aclCreateTensorDesc接口创建aclTensorDesc类型。
numDims	输入	需要设置的dims维度数。
dims	输入	dims的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.65.15 aclSetTensorStorageFormat

须知

此接口后续版本会废弃，请使用**aclSetTensorFormat**接口。

函数功能

调用**aclCreateTensorDesc**接口创建tensor描述信息后，可通过aclSetTensorStorageFormat设置tensor的实际Format信息，当前主要用于pytorch场景下。

函数原型

aclError aclSetTensorStorageFormat(aclTensorDesc *desc, aclFormat format)

参数说明

参数名	输入/输出	说明
desc	输出	aclTensorDesc类型的指针。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。
format	输入	需要设置的format。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.65.16 aclSetTensorOriginShape

函数功能

调用**aclCreateTensorDesc**接口创建tensor描述信息后，可通过aclSetTensorOriginShape接口设置tensor的原始dims信息，当前主要用于pytorch场景下。

说明

例如，某算子的原始Format为NHWC（原始Shape为4维）或者NDHWC（原始Shape为5维），为了提高数据访问效率，在进行模型转换或者在线构建网络时会自动转换Format，将原始数据Format转化为昇腾AI处理器内部计算数据时用的Format（NC1HWC0或者NDC1HWC0）,同时进行推导Shape用于内部计算。

函数原型

aclError aclSetTensorOriginShape(aclTensorDesc *desc, int numDims, const int64_t *dims)

参数说明

参数名	输入/输出	说明
desc	输出	aclTensorDesc类型的指针。 需提前调用aclCreateTensorDesc接口创建aclTensorDesc类型。
numDims	输入	需要设置的dims维度数。
dims	输入	dims的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.65.17 aclSetTensorOriginFormat

函数功能

调用aclCreateTensorDesc接口创建tensor描述信息后，可通过aclSetTensorOriginFormat设置tensor的原始Format信息，当前主要用于pytorch场景下。

说明

例如，某算子的原始Format为NHWC（原始Shape为4维）或者NDHWC（原始Shape为5维），为了提高数据访问效率，在进行模型转换或者在线构建网络时会自动转换Format，将原始数据Format转化为昇腾AI处理器内部计算数据时用的Format（NC1HWC0或者NDC1HWC0）,同时进行推导Shape用于内部计算。

函数原型

aclError aclSetTensorOriginFormat(aclTensorDesc *desc, aclFormat format)

参数说明

参数名	输入/输出	说明
desc	输出	aclTensorDesc类型的指针。 需提前调用aclCreateTensorDesc接口创建aclTensorDesc类型。
format	输入	需要设置的format。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.65.18 aclSetTensorShape

函数功能

调用aclCreateTensorDesc接口创建tensor描述信息后，可通过aclSetTensorShape接口设置昇腾AI处理器内部处理tensor时的dims信息，当前主要用于pytorch场景下。

说明

例如，某算子的原始Format为NHWC（原始Shape为4维）或者NDHWC（原始Shape为5维），为了提高数据访问效率，在进行模型转换或者在线构建网络时会自动转换Format，将原始数据Format转化为昇腾AI处理器内部计算数据时用的Format（NC1HWC0或者NDC1HWC0）,同时进行推导Shape用于内部计算。

函数原型

aclError aclSetTensorShape(aclTensorDesc *desc, int numDims, const int64_t *dims)

参数说明

参数名	输入/输出	说明
desc	输出	aclTensorDesc类型的指针。 需提前调用aclCreateTensorDesc接口创建aclTensorDesc类型。
numDims	输入	需要设置的dims维度数。
dims	输入	dims的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.65.19 aclSetTensorFormat

函数功能

调用aclCreateTensorDesc接口创建tensor描述信息后，可通过aclSetTensorFormat设置昇腾AI处理器内部处理tensor时的Format信息，当前主要用于pytorch场景下。

说明

例如，某算子的原始Format为NHWC（原始Shape为4维）或者NDHWC（原始Shape为5维），为了提高数据访问效率，在进行模型转换或者在线构建网络时会自动转换Format，将原始数据Format转化为昇腾AI处理器内部计算数据时用的Format（NC1HWC0或者NDC1HWC0）,同时进行推导Shape用于内部计算。

函数原型

aclError aclSetTensorFormat(aclTensorDesc *desc, aclFormat format)

参数说明

参数名	输入/输出	说明
desc	输出	aclTensorDesc类型的指针。 需提前调用aclCreateTensorDesc接口创建aclTensorDesc类型。
format	输入	需要设置的format。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.65.20 aclSetTensorShapeRange

函数功能

调用aclCreateTensorDesc接口创建tensor描述信息后，可通过本接口设置tensor的各个维度的取值范围。

使用场景：动态Shape的算子，其输入Shape中变化维度用-1表示，但每个变化维度的范围是不一样的，需要显式设置，如Shape为[16,-1,20,-1]，对应的shape range可以是[[16,16],[1,128],[20,20],[1,10]]，表示第一维的Shape范围固定为16，第二维的Shape范围为1到128，第三维的Shape范围固定为20，第四维的Shape范围为1到10。

函数原型

aclError aclSetTensorShapeRange(aclTensorDesc* desc, size_t dimsCount, int64_t dimsRange[][ACL_TENSOR_SHAPE_RANGE_NUM])

参数说明

参数名	输入/输出	说明
desc	输出	aclTensorDesc类型的指针。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。
dimsCount	输入	tensor中dims的维度个数。
dimsRange	输入	dimsRange为每个维度的取值范围，用二维数组表示范围。 如果Shape中的维度值不是-1时，表示固定维度，该维度对应的shape range中最大值和最小值相同；否则，表示动态维度，数组的两个维度分别表示对应tensor dims中的最小值和最大值。 `#define ACL_TENSOR_SHAPE_RANGE_NUM 2`

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.65.21 aclSetTensorDynamicInput

函数功能

调用**aclCreateTensorDesc**接口创建tensor描述信息后，可通过aclSetTensorDynamicInput接口设置多个动态输入的分组标识，例如某个算子包括动态输入为x、必选输入y，其中动态输入有多个，调用aclSetTensorDynamicInput将**dynamicInputName**设置为x，AscendCL内部根据**dynamicInputName**索引到对应的多个动态输入，达到分组的目的。

函数原型

aclError aclSetTensorDynamicInput(aclTensorDesc *desc, const char *dynamicInputName)

参数说明

参数名	输入/输出	说明
desc	输出	aclTensorDesc类型的指针。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。
dynamicInputName	输入	desc中动态输入分组标识的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.65.22 aclGetTensorDescByIndex

函数功能

获取算子的指定Index的tensor描述信息。

函数原型

aclTensorDesc *aclGetTensorDescByIndex(aclTensorDesc *desc, size_t index)

参数说明

参数名	输入/输出	说明
desc	输入	aclTensorDesc类型的指针。 调用aclmdlCreateAndGetOpDesc接口获取算子所有输入/输出的tensor描述，作为本接口的输入。
index	输入	指定获取第几个输入/输出的tensor描述，index值从0开始。 用户调用aclmdlCreateAndGetOpDesc接口获取的算子输入/输出的数量后，这个Index的取值范围：[0, (算子输入/输出的数量-1)]。

返回值说明

返回指定输入/输出的aclTensorDesc类型数据的指针。

11.27.65.23 aclGetTensorDescAddress

函数功能

获取指定算子输入/输出的tensor数据的内存地址。

函数原型

void *aclGetTensorDescAddress(const aclTensorDesc *desc)

参数说明

参数名	输入/输出	说明
desc	输入	aclTensorDesc类型的指针。 调用**aclGetTensorDescByIndex**接口获取算子的指定输入/输出的tensor描述，作为本接口的输入。

返回值说明

返回指定算子输入/输出的tensor数据的内存地址。

11.27.65.24 aclSetTensorConst

函数功能

为算子设置constant输入，指定存放输入tensor数据的内存地址和地址长度。

函数原型

aclError aclSetTensorConst(aclTensorDesc *desc, void *dataBuffer, size_t length)

参数说明

参数名	输入/输出	说明
desc	输出	aclTensorDesc类型的指针。 需提前调用**aclCreateTensorDesc**接口创建aclTensorDesc类型。
dataBuffer	输入	输入tensor数据内存地址指针。 此处算子输入tensor数据的内存必须根据应用运行模式来确定，应用运行在Host时，此处需申请Host上的内存；应用运行在Device时，此处需申请Device上的内存。内存申请接口请参见**11.11 内存管理**。
length	输入	内存地址的长度，单位是Byte。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.65.25 aclSetTensorPlaceMent

函数功能

设置TensorDesc的placement属性，用于标识存放Tensor数据的内存类型。

函数原型

aclError aclSetTensorPlaceMent(aclTensorDesc *desc, aclMemType memType)

参数说明

参数名	输入/输出	说明
desc	输出	aclTensorDesc类型的指针。 需提前调用aclCreateTensorDesc接口创建aclTensorDesc类型。
memType	输入	指定placement属性值。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.65.26 aclSetTensorValueRange

函数功能

调用aclCreateTensorDesc接口创建tensor描述信息后，可通过本接口设置tensor的数据值的范围。

使用场景：部分算子input的值就是该算子的输出Shape，在动态Shape场景下，这个Shape的值就有一个取值范围，在执行算子时，需要在input上设置Shape值的范围（例如：[[16,16],[1,128],[20,20],[1,10]]），AscendCL才能正常编译、执行算子。

函数原型

aclError aclSetTensorValueRange(aclTensorDesc* desc, size_t valueCount, int64_t valueRange[][ACL_TENSOR_VALUE_RANGE_NUM])

参数说明

参数名	输入/输出	说明
desc	输出	aclTensorDesc类型的指针。 需提前调用aclCreateTensorDesc接口创建aclTensorDesc类型。

参数名	输入/输出	说明
valueCount	输入	需设置范围的数据值的个数。
valueRange	输入	valueRange为每个数据值的范围，用二维数组表示范围。 #define ACL_TENSOR_VALUE_RANGE_NUM 2

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.66 aclopAttr

11.27.66.1 aclopCreateAttr

函数功能

创建aclopAttr类型的数据，该数据类型用于保存算子的属性。

如需销毁aclopAttr类型的数据，请参见**aclopDestroyAttr**。

函数原型

aclopAttr *aclopCreateAttr()

参数说明

无

返回值说明

返回aclopAttr类型的指针。

11.27.66.2 aclopDestroyAttr

函数功能

销毁通过**aclopCreateAttr**接口创建的aclopAttr类型的数据。

函数原型

void aclopDestroyAttr(const aclopAttr *attr)

参数说明

参数名	输入/输出	说明
attr	输入	待销毁的aclopAttr类型指针。

返回值说明

无

11.27.66.3 aclopSetAttrBool

函数功能

设置bool类型标量的属性值。

函数原型

aclError aclopSetAttrBool(aclopAttr *attr, const char *attrName, uint8_t attrValue)

参数说明

参数名	输入/输出	说明
attr	输出	aclopAttr类型的指针。 需提前调用aclopCreateAttr接口创建aclopAttr类型。
attrName	输入	属性名的指针。
attrValue	输入	属性值。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.66.4 aclopSetAttrInt

函数功能

设置int64_t类型标量的属性值。

函数原型

aclError aclopSetAttrInt(aclopAttr *attr, const char *attrName, int64_t attrValue)

参数说明

参数名	输入/输出	说明
attr	输出	aclopAttr类型的指针。 需提前调用aclopCreateAttr接口创建aclopAttr类型。
attrName	输入	属性名的指针。
attrValue	输入	属性值。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.66.5 aclopSetAttrFloat

函数功能

设置float类型标量的属性值。

函数原型

aclError aclopSetAttrFloat(aclopAttr *attr, const char *attrName, float attrValue)

参数说明

参数名	输入/输出	说明
attr	输出	aclopAttr类型的指针。 需提前调用aclopCreateAttr接口创建aclopAttr类型。
attrName	输入	属性名的指针。
attrValue	输入	属性值。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.66.6 aclopSetAttrString

函数功能

设置字符串类型标量的属性值。

函数原型

aclError aclopSetAttrString(aclopAttr *attr, const char *attrName, const char *attrValue)

参数说明

参数名	输入/输出	说明
attr	输出	aclopAttr类型的指针。 需提前调用aclopCreateAttr接口创建aclopAttr类型。
attrName	输入	属性名的指针。
attrValue	输入	属性值的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.66.7 aclopSetAttrListBool

函数功能

设置bool类型列表的属性值。

函数原型

aclError aclopSetAttrListBool(aclopAttr *attr, const char *attrName, int numValues, const uint8_t *values)

参数说明

参数名	输入/输出	说明
attr	输出	aclopAttr类型的指针。 需提前调用aclopCreateAttr接口创建aclopAttr类型。
attrName	输入	属性名的指针。
numValues	输入	属性值数目。
values	输入	属性值的数组。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.66.8 aclopSetAttrListInt

函数功能

设置int64_t类型列表的属性值。

函数原型

aclError aclopSetAttrListInt(aclopAttr *attr, const char *attrName, int numValues, const int64_t *values)

参数说明

参数名	输入/输出	说明
attr	输出	aclopAttr类型的指针。 需提前调用aclopCreateAttr接口创建aclopAttr类型。
attrName	输入	属性名的指针。
numValues	输入	属性值数目。
values	输入	属性值的数组。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.66.9 aclopSetAttrListFloat

函数功能

设置float类型列表的属性值。

函数原型

aclError aclopSetAttrListFloat(aclopAttr *attr, const char *attrName, int numValues, const float *values)

参数说明

参数名	输入/输出	说明
attr	输出	aclopAttr类型的指针。 需提前调用aclopCreateAttr接口创建aclopAttr类型。
attrName	输入	属性名的指针。
numValues	输入	属性值数目。

参数名	输入/输出	说明
values	输入	属性值的数组。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.66.10 aclopSetAttrListString

函数功能

设置字符串类型列表的属性值。

函数原型

aclError aclopSetAttrListString(aclopAttr *attr, const char *attrName, int numValues, const char **values)

参数说明

参数名	输入/输出	说明
attr	输出	aclopAttr类型的指针。 需提前调用**aclopCreateAttr**接口创建aclopAttr类型。
attrName	输入	属性名的指针。
numValues	输入	属性值数目。
values	输入	“属性值的数组”的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.66.11 aclopSetAttrListListInt

函数功能

设置int64_t类型列表的属性值。

函数原型

aclError aclopSetAttrListListInt(aclopAttr *attr,

const char *attrName,

int numLists,

const int *numValues,

const int64_t *const values[]);

参数说明

参数名	输入/输出	说明
attr	输出	aclopAttr类型的指针。 需提前调用**aclopCreateAttr**接口创建aclopAttr类型。
attrName	输入	属性名的指针。
numLists	输入	列表数目。
numValues	输入	列表中属性值数目。 numValues是一个数组，数组中的每个元素表示每个列表中的属性值数目。
values	输入	列表中属性值的指针数组。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.66.12 aclopSetAttrDataType

函数功能

设置指定属性的数据类型。

函数原型

aclError aclopSetAttrDataType(aclopAttr *attr, const char *attrName, aclDataType attrValue)

参数说明

参数名	输入/输出	说明
attr	输出	aclopAttr类型的指针。 需提前调用**aclopCreateAttr**接口创建aclopAttr类型。
attrName	输入	属性名的指针。
attrValue	输入	属性值。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.66.13 aclopSetAttrListDataType

函数功能

设置指定属性列表的数据类型。

函数原型

aclError aclopSetAttrListDataType(aclopAttr *attr, const char *attrName, int numValues, const aclDataType values[])

参数说明

参数名	输入/输出	说明
attr	输出	aclopAttr类型的指针。 需提前调用**aclopCreateAttr**接口创建aclopAttr类型。
attrName	输入	属性名的指针。
numValues	输入	属性值数目。
values	输入	属性值的数组。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.67 acldvppChannelDesc

11.27.67.1 acldvppCreateChannelDesc

函数功能

创建acldvppChannelDesc类型的数据，表示创建图片数据处理通道时的通道描述信息。

函数原型

acldvppChannelDesc *acldvppCreateChannelDesc()

参数说明

无

返回值说明

- 返回acldvppChannelDesc类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.67.2 acldvppDestroyChannelDesc

函数功能

销毁通过**acldvppCreateChannelDesc**接口创建的acldvppChannelDesc类型的数据。

函数原型

aclError acldvppDestroyChannelDesc(acldvppChannelDesc *channelDesc)

参数说明

参数名	输入/输出	说明
channelDesc	输入	待销毁的acldvppChannelDesc类型指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.67.3 acldvppGetChannelDescChannelId

函数功能

根据通道描述信息获取通道ID。

约束说明

接口调用顺序：acldvppCreateChannelDesc --> acldvppCreateChannel --> acldvppGetChannelDescChannelId

函数原型

uint64_t acldvppGetChannelDescChannelId(const acldvppChannelDesc *channelDesc)

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针。 需提前调用acldvppCreateChannelDesc接口创建acldvppChannelDesc类型的数据。

返回值说明

返回通道ID。

11.27.67.4 acldvppSetChannelDescMode

函数功能

指定通道描述信息中的通道模式，明确图片数据处理通道用于实现哪种功能，目前支持VPC、JPEGD、JPEGE、PNGD功能。

Atlas 200/300/500 推理产品，不支持该接口。

约束说明

接口调用顺序：acldvppCreateChannelDesc --> acldvppSetChannelDescMode --> acldvppCreateChannel

函数原型

aclError acldvppSetChannelDescMode(acldvppChannelDesc *channelDesc, uint32_t mode)

参数说明

参数名	输入/输出	说明
channelDesc	输出	通道描述信息的指针。 需提前调用acldvppCreateChannelDesc接口创建acldvppChannelDesc类型的数据。
mode	输入	指定通道模式，各模式的取值请参见acldvppChannelMode枚举值。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.67.5 acldvppSetChannelDescParam

函数功能

设置图片数据处理通道的属性参数接口。

函数原型

aclError acldvppSetChannelDescParam(acldvppChannelDesc *channelDesc, acldvppChannelDescParamType paramType, size_t length, const void *param)

参数说明

参数名	输入/输出	说明
channelDesc	输出	通道描述信息的指针。 需提前调用acldvppCreateChannelDesc接口创建acldvppChannelDesc类型的数据。
paramType	输入	属性参数的类型。
length	输入	属性参数值的字节数。 ● 属性参数的类型为*_UINT64时，此处配置为8。 ● 属性参数的类型为*_UINT32时，此处配置为4。 ● 属性参数的类型为*_ENUM时，此处配置为4。
param	输入	属性参数值的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.67.6 acldvppGetChannelDescParam

函数功能

获取图片数据处理通道的属性参数接口。

函数原型

aclError acldvppGetChannelDescParam(const acldvppChannelDesc *channelDesc, acldvppChannelDescParamType paramType, size_t length, size_t *paramRetSize, void *param)

参数说明

参数名	输入/输出	说明
channelDesc	输入	通道描述信息的指针。 需提前调用**acldvppCreateChannelDesc**接口创建acldvppChannelDesc类型的数据，调用**acldvppSetChannelDescParam**接口设置通道属性值。
paramType	输入	属性参数的类型。
length	输入	属性参数值的字节数。 ● 属性参数的类型为*_UINT64时，此处配置为8。 ● 属性参数的类型为*_UINT32时，此处配置为4。 ● 属性参数的类型为*_ENUM时，此处配置为4。
paramRetSize	输出	实际返回的属性参数值字节数的指针。
param	输出	属性参数值的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.68 acldvppPicDesc

11.27.68.1 acldvppCreatePicDesc

函数功能

创建acldvppPicDesc类型的数据，表示图片描述信息。

如需销毁acldvppPicDesc类型的数据，请参见**acldvppDestroyPicDesc**。

函数原型

acldvppPicDesc *acldvppCreatePicDesc()

参数说明

无

返回值说明

- 返回acldvppPicDesc类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.68.2 acldvppSetPicDesc 系列接口

函数功能

设置图片描述参数。

函数原型

```
aclError acldvppSetPicDescData(acldvppPicDesc *picDesc, void *dataDev);
aclError acldvppSetPicDescSize(acldvppPicDesc *picDesc, uint32_t size);
aclError acldvppSetPicDescFormat(acldvppPicDesc *picDesc,
acldvppPixelFormat format);
aclError acldvppSetPicDescWidth(acldvppPicDesc *picDesc, uint32_t width);
aclError acldvppSetPicDescHeight(acldvppPicDesc *picDesc, uint32_t height);
aclError acldvppSetPicDescWidthStride(acldvppPicDesc *picDesc, uint32_t
widthStride);
aclError acldvppSetPicDescHeightStride(acldvppPicDesc *picDesc, uint32_t
heightStride);
aclError acldvppSetPicDescRetCode(acldvppPicDesc *picDesc,uint32_t retCode)
```

参数说明

参数名	输入/输出	说明
picDesc	输出	描述图片信息的指针。 需提前调用**acldvppCreatePicDesc**接口创建acldvppPicDesc类型的数据。
dataDev	输入	Device上图片数据内存的指针，必须是使用**acldvppMalloc**接口申请的内存。
size	输入	内存大小，单位Byte。
format	输入	图片格式。
width	输入	原始图片宽。
height	输入	原始图片高。
widthStride	输入	对齐后的宽。
heightStride	输入	对齐后的高。

参数名	输入/输出	说明
retCode	输入	返回码（0成功，非0失败）。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.68.3 acldvppGetPicDesc 系列接口

函数功能

获取图片描述信息。

函数原型

```
void *acldvppGetPicDescData(const acldvppPicDesc *picDesc);
uint32_t acldvppGetPicDescSize(const acldvppPicDesc *picDesc);
acldvppPixelFormat acldvppGetPicDescFormat(const acldvppPicDesc *picDesc);
uint32_t acldvppGetPicDescWidth(const acldvppPicDesc *picDesc);
uint32_t acldvppGetPicDescHeight(const acldvppPicDesc *picDesc);
uint32_t acldvppGetPicDescWidthStride(const acldvppPicDesc *picDesc);
uint32_t acldvppGetPicDescHeightStride(const acldvppPicDesc *picDesc);
uint32_t acldvppGetPicDescRetCode(const acldvppPicDesc *picDesc);
```

参数说明

参数名	输入/输出	说明
picDesc	输入	描述图片信息的指针。 需提前调用**acldvppCreatePicDesc**接口创建acldvppPicDesc类型的数据，调用**acldvppSetPicDesc系列接口**设置图片描述参数。

返回值说明

参数名	说明
data	图片数据的内存地址。
size	图片数据的内存的大小。

参数名	说明
format	图片格式。
width	原始图片宽。
height	原始图片高。
widthStride	对齐后的宽。
heightStride	对齐后的高。
retCode	返回码（0成功，非0失败）。 VDEC解码时，用户需调用acldvppGetPicDescRetCode接口获取retCode返回码判断是否解码成功，如果解码失败，需要根据日志中的返回码判断具体的问题，返回码请参见**返回码说明**。retCode中部分返回码是以十六进制形式定义的，如用户在编码时以十进制打印返回码的值（例如，65540），则需提前转换为十六进制（例如，0x10004），然后在**返回码说明**中查询。

11.27.68.4 acldvppDestroyPicDesc

函数功能

销毁通过**acldvppCreatePicDesc**接口创建的acldvppPicDesc类型的数据。

函数原型

aclError acldvppDestroyPicDesc(acldvppPicDesc *picDesc)

参数说明

参数名	输入/输出	说明
picDesc	输入	待销毁的acldvppPicDesc类型指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.69 acldvppRoiConfig

11.27.69.1 acldvppCreateRoiConfig

函数功能

创建acldvppRoiConfig类型的数据，用于描述某个区域位置的数据。

如需销毁acldvppRoiConfig类型的数据，请参见**acldvppDestroyRoiConfig**。

函数原型

```
acldvppRoiConfig *acldvppCreateRoiConfig(
uint32_t left,
uint32_t right,
uint32_t top,
uint32_t bottom)
```

参数说明

参数名	输入/输出	说明
left	输入	左偏移。 ● 在Atlas 200/300/500 推理产品上，该参数值必须为偶数。如果用做贴图区域，左偏移需要16对齐 ● 在Atlas 推理系列产品上，该参数值无偶数要求。如果用做贴图区域，左偏移需要16对齐
right	输入	右偏移。 ● 在Atlas 200/300/500 推理产品上，该参数值必须为奇数。 ● 在Atlas 推理系列产品上，该参数值无奇数要求。
top	输入	上偏移。 ● 在Atlas 200/300/500 推理产品上，该参数值必须为偶数。 ● 在Atlas 推理系列产品上，该参数值无偶数要求。
bottom	输入	下偏移。 ● 在Atlas 200/300/500 推理产品上，该参数值必须为奇数。 ● 在Atlas 推理系列产品上，该参数值无奇数要求。

返回值说明

- 返回acldvppRoiConfig类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.69.2 acldvppSetRoiConfig 系列接口

函数功能

设置某个区域的位置信息。

函数原型

aclError acldvppSetRoiConfig(acldvppRoiConfig *config, uint32_t left, uint32_t right, uint32_t top, uint32_t bottom);

aclError acldvppSetRoiConfigLeft (acldvppRoiConfig *config, uint32_t left);

aclError acldvppSetRoiConfigRight (acldvppRoiConfig *config, uint32_t right);

aclError acldvppSetRoiConfigTop (acldvppRoiConfig *config, uint32_t top);

aclError acldvppSetRoiConfigBottom(acldvppRoiConfig *config, uint32_t bottom)

参数说明

参数名	输入/输出	说明
config	输出	区域位置数据的指针。 需提前调用**acldvppCreateRoiConfig**接口创建acldvppRoiConfig类型的数据。
left	输入	左偏移。 ● 在Atlas 200/300/500 推理产品上，该参数值必须为偶数。如果用做贴图区域，左偏移需要16对齐 ● 在Atlas 推理系列产品上，该参数值无偶数要求。如果用做贴图区域，左偏移需要16对齐
right	输入	右偏移。 ● 在Atlas 200/300/500 推理产品上，该参数值必须为奇数。 ● 在Atlas 推理系列产品上，该参数值无奇数要求。

参数名	输入/输出	说明
top	输入	上偏移。 ● 在Atlas 200/300/500 推理产品上，该参数值必须为偶数。 ● 在Atlas 推理系列产品上，该参数值无偶数要求。
bottom	输入	下偏移。 ● 在Atlas 200/300/500 推理产品上，该参数值必须为奇数。 ● 在Atlas 推理系列产品上，该参数值无奇数要求。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.69.3 acldvppDestroyRoiConfig

函数功能

销毁通过**acldvppCreateRoiConfig**接口创建的acldvppRoiConfig类型的数据。

函数原型

aclError acldvppDestroyRoiConfig(acldvppRoiConfig *roiConfig)

参数说明

参数名	输入/输出	说明
roiConfig	输入	待销毁的acldvppRoiConfig类型的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.70 acldvppResizeConfig

11.27.70.1 acldvppCreateResizeConfig

函数功能

创建acldvppResizeConfig类型的数据，表示图片缩放配置数据。

如需销毁acldvppResizeConfig类型的数据，请参见**acldvppDestroyResizeConfig**。

Atlas 200/300/500 推理产品，默认缩放算法为“自研的高阶滤波算法”。

Atlas 推理系列产品，默认缩放算法为“业界通用的Bilinear算法”。

函数原型

acldvppResizeConfig *acldvppCreateResizeConfig()

参数说明

无

返回值说明

- 返回acldvppResizeConfig类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.70.2 acldvppSetResizeConfigInterpolation

函数功能

设置图片缩放算法。

函数原型

aclError acldvppSetResizeConfigInterpolation(acldvppResizeConfig *resizeConfig, uint32_t interpolation)

参数说明

参数名	输入/输出	说明
resizeConfig	输出	待设置的缩放配置数据的指针。 需提前调用**acldvppCreateResizeConfig**接口创建acldvppResizeConfig类型的数据。

参数名	输入/输出	说明
interpolation	输入	指定缩放算法。此处配置的缩放算法建议与训练模型时的缩放算法保持一致。 Atlas 200/300/500 推理产品上，支持如下缩放算法： ● 0：默认值，表示自研的高阶滤波算法。 ● 1：业界通用的Bilinear算法（与OpenCV算法的计算过程类似） ● 2：业界通用的Nearest neighbor 算法（与OpenCV算法的计算过程类似） ● 3：业界通用的Bilinear算法（与Tensorflow算法的计算过程类似） ● 4：业界通用的Nearest neighbor算法（与Tensorflow算法的计算过程类似） Atlas 推理系列产品,上，支持如下缩放算法： ● 0：设置为0时，系统内部也会自动采用1。 ● 1：业界通用的Bilinear算法（与OpenCV算法的计算过程类似）。 设置该算法，且当输入和输出图片格式都为RGB时，在[1/32, 512]的缩放范围内，与OpenCV的单个像素值最大差异为正负1。 ● 2：业界通用的Nearest neighbor 算法（与OpenCV算法的计算过程类似）

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.70.3 acldvppGetResizeConfigInterpolation

函数功能

获取图片缩放算法。

函数原型

uint32 acldvppGetResizeConfigInterpolation(const acldvppResizeConfig * resizeConfig)

参数说明

参数名	输入/输出	说明
resizeConfig	输入	待获取的缩放配置数据的指针。 需提前调用**acldvppCreateResizeConfig**接口创建acldvppResizeConfig类型的数据，调用**acldvppSetResizeConfigInterpolation**接口设置缩放算法。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.70.4 acldvppDestroyResizeConfig

函数功能

销毁通过**acldvppCreateResizeConfig**接口创建的acldvppResizeConfig类型的数据。

函数原型

aclError acldvppDestroyResizeConfig(acldvppResizeConfig *resizeConfig)

参数说明

参数名	输入/输出	说明
resizeConfig	输入	待销毁的acldvppResizeConfig类型的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.71 acldvppJpegeConfig

11.27.71.1 acldvppCreateJpegeConfig

函数功能

创建acldvppJpegeConfig类型的数据，表示图片编码配置数据。

如需销毁acldvppJpegeConfig类型的数据，请参见**acldvppDestroyJpegeConfig**。

函数原型

acldvppJpegeConfig *acldvppCreateJpegeConfig()

参数说明

无

返回值说明

- 返回acldvppJpegeConfig类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.71.2 acldvppSetJpegeConfigLevel

函数功能

设置编码配置数据。

函数原型

aclError acldvppSetJpegeConfigLevel(acldvppJpegeConfig *jpegeConfig, uint32_t level)

参数说明

参数名	输入/输出	说明
jpegeConfig	输出	编码配置数据的指针。 需提前调用acldvppCreateJpegeConfig接口创建acldvppJpegeConfig类型的数据。
level	输入	编码质量范围。 Atlas 200/300/500 推理产品，该参数取值范围：[0, 100]，其中level 0编码质量与level 100差不多，而在[1, 100]内数值越小输出图片质量越差。 Atlas 推理系列产品，该参数取值范围：[1, 100]，数值越小输出图片质量越差。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.71.3 acldvppGetJpegeConfigLevel

函数功能

获取编码配置数据。

函数原型

uint32_t acldvppGetJpegeConfigLevel(const acldvppJpegeConfig *jpegeConfig)

参数说明

参数名	输入/输出	说明
jpegeConfig	输入	编码配置数据的指针。 需提前调用acldvppCreateJpegeConfig接口创建acldvppJpegeConfig类型的数据，再调用acldvppSetJpegeConfigLevel接口设置编码配置数据。

返回值说明

返回编码质量范围。

Atlas 200/300/500 推理产品，该参数取值范围：[0, 100]，其中level 0编码质量与level 100差不多，而在[1, 100]内数值越小输出图片质量越差。

Atlas 推理系列产品，该参数取值范围：[1, 100]，数值越小输出图片质量越差。

11.27.71.4 acldvppDestroyJpegeConfig

函数功能

销毁通过acldvppCreateJpegeConfig接口创建的acldvppJpegeConfig类型的数据。

函数原型

aclError acldvppDestroyJpegeConfig(acldvppJpegeConfig *jpegeConfig)

参数说明

参数名	输入/输出	说明
jpegeConfig	输入	待销毁的acldvppJpegeConfig类型的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.72 aclvdecChannelDesc

11.27.72.1 aclvdecCreateChannelDesc

函数功能

创建aclvdecChannelDesc类型的数据，表示创建视频解码处理通道时的通道描述信息。

如需销毁aclvdecChannelDesc类型的数据，请参见**aclvdecDestroyChannelDesc**。

函数原型

aclvdecChannelDesc *aclvdecCreateChannelDesc()

参数说明

无

返回值说明

- 返回aclvdecChannelDesc类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.72.2 aclvdecSetChannelDesc 系列接口

函数功能

设置视频解码处理通道描述信息的属性值。

函数原型

aclError aclvdecSetChannelDescChannelId(aclvdecChannelDesc *channelDesc, uint32_t channelId);

aclError aclvdecSetChannelDescThreadId(aclvdecChannelDesc *channelDesc, uint64_t threadId);

aclError aclvdecSetChannelDescCallback(aclvdecChannelDesc *channelDesc, aclvdecCallback callback);

aclError aclvdecSetChannelDescEnType(aclvdecChannelDesc *channelDesc, acldvppStreamFormat enType);

aclError aclvdecSetChannelDescOutPicFormat(aclvdecChannelDesc *channelDesc, acldvppPixelFormat outPicFormat);

aclError aclvdecSetChannelDescOutPicWidth(aclvdecChannelDesc *channelDesc, uint32_t outPicWidth);

aclError aclvdecSetChannelDescOutPicHeight(aclvdecChannelDesc *channelDesc, uint32_t outPicHeight);

aclError aclvdecSetChannelDescRefFrameNum(aclvdecChannelDesc *channelDesc, uint32_t refFrameNum);

aclError aclvdecSetChannelDescOutMode(aclvdecChannelDesc *channelDesc, uint32_t outMode)

aclError aclvdecSetChannelDescBitDepth(aclvdecChannelDesc *channelDesc, uint32_t bitDepth)

参数说明

参数名	输入/输出	说明
channelDesc	输出	视频解码处理通道描述信息的指针。 需提前调用aclvdecCreateChannelDesc接口创建aclvdecChannelDesc类型的数据。
channelId	输入	通道ID。 Atlas 200/300/500 推理产品，该参数值的取值范围[0, 31]。 Atlas 推理系列产品，该参数值的取值范围[0, 255]。
threadId	输入	回调线程ID。 **说明** 同一个进程内，在不同的Device上注册VDEC解码回调函数的线程时，不能指定同一个线程ID。 同一个Device上，多路VDEC解码可以指定同一个线程ID，但相比一个线程处理一路VDEC解码任务来说，一个线程内串行处理多路VDEC解码任务时，VDEC解码性能可能下降。
callback	输入	解码回调函数。 请参见11.17.9.5 aclvdecCallback。
enType	输入	视频编码协议H265-main level（0）、H264-baseline level（1）、H264-main level(2)、H264-high level（3）。
outPicFormat	输入	YUV图像存储格式，支持如下格式： ● YUV420SP NV12 ● YUV420SP NV21 ● RGB888，Atlas 200/300/500 推理产品不支持该格式 ● BGR888，Atlas 200/300/500 推理产品不支持该格式 如果不设置输出格式，默认使用YUV420SP NV12。
outPicWidth	输入	解码码流最大宽度。 Atlas 200/300/500 推理产品，如果不设置解码码流最大宽度，内部默认使用1920。 Atlas 推理系列产品，如果不设置解码码流最大宽度，内部默认使用4096。

参数名	输入/输出	说明
outPicHeight	输入	解码码流最大高度。 Atlas 200/300/500 推理产品，如果不设置解码码流最大高度，内部默认使用1120。 Atlas 推理系列产品，如果不设置解码码流最大高度，内部默认使用4096。
refFrameNum	输入	参考帧数量，取值范围[0, 16]。 Atlas 200/300/500 推理产品，当前版本不支持用户设置该参数。 Atlas 推理系列产品，如果不设置参考帧数量，内部默认使用8。
outMode	输入	设置是否实时出帧（即发送一帧解码一帧，无需依赖后续帧的传入）。 取值范围如下： ● 0：默认出帧模式，由于解码过程中存在缓存帧，无法实时输出，因此VDEC需要在收到码流中的多帧数据后，才开始输出解码结果。 ● 1：快速出帧模式，VDEC获取码流中的一帧数据后，就开始实时输出解码结果。只支持简单参考关系的H264/H265标准码流（无长期参考帧，无B帧）。
bitDepth	输入	设置视频位宽，默认值为10-bit。 取值范围如下： ● 0：8-bit ● 1: 10-bit(默认值） Atlas 200/300/500 推理产品，当前版本不支持用户设置该参数。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.72.3 aclvdecGetChannelDesc 系列接口

函数功能

获取视频解码处理通道的描述信息。

函数原型

uint32_t aclvdecGetChannelDescChannelId(const aclvdecChannelDesc *channelDesc);

uint32_t aclvdecGetChannelDescThreadId(const aclvdecChannelDesc *channelDesc);

aclvdecCallback aclvdecGetChannelDescCallback(const aclvdecChannelDesc *channelDesc);

acldvppStreamFormat aclvdecGetChannelDescEnType(const aclvdecChannelDesc *channelDesc);

acldvppPixelFormat aclvdecGetChannelDescOutPicFormat(const aclvdecChannelDesc *channelDesc);

uint32_t aclvdecGetChannelDescOutPicWidth(const aclvdecChannelDesc *channelDesc);

uint32_t aclvdecGetChannelDescOutPicHeight(const aclvdecChannelDesc *channelDesc);

uint32_t aclvdecGetChannelDescRefFrameNum(const aclvdecChannelDesc *channelDesc);

uint32_t aclvdecGetChannelDescOutMode(const aclvdecChannelDesc *channelDesc)

uint32_t aclvdecGetChannelDescBitDepth(const aclvdecChannelDesc *channelDesc)

参数说明

参数名	输入/输出	说明
channelDesc	输入	视频解码数据处理通道描述信息的指针。 需提前调用aclvdecCreateChannelDesc接口创建aclvdecChannelDesc类型的数据，调用aclvdecSetChannelDesc系列接口设置视频解码处理通道描述信息的属性值。

返回值说明

参数名	说明
channelId	解码通道号。 Atlas 200/300/500 推理产品，该参数值的取值范围[0, 31]。 Atlas 推理系列产品，该参数值的取值范围[0, 255]。
threadId	回调线程ID。

参数名	说明
callback	回调函数。 请参见11.17.9.5 aclvdecCallback。
enType	视频编码协议H265-main level（0）、H264-baseline level（1）、H264-main level(2)、H264-high level（3）。
outPicFormat	YUV图像存储格式。
outPicWidth	图片宽度。
outPicHeight	图片高度。
refFrameNum	参考帧数量，取值范围[0, 16]。 Atlas 200/300/500 推理产品，当前版本获取该参数的值为无效值，不支持获取该参数值。
outMode	是否实时出帧（即发送一帧解码一帧，无需依赖后续帧的传入），取值范围如下： ● 0：默认出帧模式，由于解码过程中存在缓存帧，无法实时输出，因此VDEC需要在收到码流中的多帧数据后，才开始输出解码结果。 ● 1：快速出帧模式，VDEC获取码流中的一帧数据后，就开始实时输出解码结果。只支持简单参考关系的H264/H265标准码流（无长期参考帧，无B帧）。
bitDepth	视频位宽。 取值范围如下： ● 0：8-bit ● 1: 10-bit Atlas 200/300/500 推理产品，当前版本获取该参数的值为无效值，不支持获取该参数值。

11.27.72.4 aclvdecSetChannelDescParam

函数功能

设置视频编码处理通道描述信息的属性值。

函数原型

aclError aclvdecSetChannelDescParam(aclvdecChannelDesc *channelDesc, aclvdecChannelDescParamType paramType, size_t length, const void *param)

参数说明

参数名	输入/输出	说明
channelDesc	输出	视频解码处理通道描述信息的指针。 需提前调用aclvdecCreateChannelDesc接口创建aclvdecChannelDesc类型的数据。
paramType	输入	属性参数的类型。
length	输入	属性参数值的字节数。 ● 属性参数的类型为*_UINT64时，此处配置为8。 ● 属性参数的类型为*_UINT32时，此处配置为4。 ● 属性参数的类型为*_PTR时，若操作系统是32位，则此处配置为4；若操作系统是64位，则配置为8。
param	输入	属性参数值的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.72.5 aclvdecGetChannelDescParam

函数功能

获取视频解码处理通道的描述信息。

函数原型

aclError aclvdecGetChannelDescParam(const aclvdecChannelDesc *channelDesc, aclvdecChannelDescParamType paramType, size_t length, size_t *paramRetSize, void *param)

参数说明

参数名	输入/输出	说明
channelDesc	输入	视频解码数据处理通道描述信息的指针。 需提前调用**aclvdecCreateChannelDesc**接口创建aclvdecChannelDesc类型的数据，调用**aclvdecSetChannelDescParam**接口设置视频解码处理通道描述信息的属性值。
paramType	输入	属性参数的类型。
length	输入	属性参数值的字节数。 ● 属性参数的类型为*_UINT64时，此处配置为8。 ● 属性参数的类型为*_UINT32时，此处配置为4。 ● 属性参数的类型为*_PTR时，若操作系统是32位，则此处配置为4；若操作系统是64位，则配置为8。
paramRetSize	输出	实际返回的属性参数值字节数的指针。
param	输出	属性参数值的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.72.6 aclvdecDestroyChannelDesc

函数功能

销毁通过**aclvdecCreateChannelDesc**接口创建的aclvdecChannelDesc类型的数据。

函数原型

aclError aclvdecDestroyChannelDesc(aclvdecChannelDesc *channelDesc)

参数说明

参数名	输入/输出	说明
channelDesc	输入	待销毁的aclvdecChannelDesc类型的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.73 acldvppStreamDesc

11.27.73.1 acldvppCreateStreamDesc

函数功能

创建acldvppStreamDesc类型的数据，表示视频码流描述信息。

如需销毁acldvppStreamDesc类型的数据，请参见**acldvppDestroyStreamDesc**。

函数原型

acldvppStreamDesc *acldvppCreateStreamDesc()

参数说明

无

返回值说明

- 返回acldvppStreamDesc类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.73.2 acldvppSetStreamDesc 系列接口

函数功能

设置视频码流信息的属性值。

函数原型

aclError acldvppSetStreamDescData(acldvppStreamDesc *streamDesc, void *dataDev);

aclError acldvppSetStreamDescSize(acldvppStreamDesc *streamDesc, uint32_t size);

aclError acldvppSetStreamDescFormat(acldvppStreamDesc *streamDesc, acldvppStreamFormat format);

aclError acldvppSetStreamDescTimestamp(acldvppStreamDesc *streamDesc, uint64_t timestamp);

aclError acldvppSetStreamDescRetCode(acldvppStreamDesc *streamDesc, uint32_t retCode);

aclError acldvppSetStreamDescEos(acldvppStreamDesc *streamDesc, uint8_t eos)

参数说明

参数名	输入/输出	说明
streamDesc	输出	要设置的视频码流信息的指针。 需提前调用acldvppCreateStreamDesc接口创建acldvppStreamDesc类型的数据。
dataDev	输入	Device上码流数据内存的指针。
size	输入	内存大小，单位Byte。
format	输入	码流格式（h264/h265）。
timestamp	输入	时间戳。
retCode	输入	返回码（0成功，非0失败）。
eos	输入	是否为最后一帧数据（0否，1是）。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.73.3 acldvppGetStreamDesc 系列接口

函数功能

获取视频码流信息的属性。

函数原型

```
void *acldvppGetStreamDescData(const acldvppStreamDesc *streamDesc);
uint32_t acldvppGetStreamDescSize(const acldvppStreamDesc *streamDesc);
acldvppStreamFormat acldvppGetStreamDescFormat(const acldvppStreamDesc *streamDesc);
uint64_t acldvppGetStreamDescTimestamp(const acldvppStreamDesc *streamDesc);
uint32_t acldvppGetStreamDescRetCode(const acldvppStreamDesc *streamDesc);
uint8_t acldvppGetStreamDescEos(const acldvppStreamDesc *streamDesc)
```

参数说明

参数名	输入/输出	说明
streamDesc	输入	要设置的视频码流信息的指针。 需提前调用**acldvppCreateStreamDesc**接口创建acldvppStreamDesc类型的数据，调用**acldvppSetStreamDesc系列接口**设置视频码流信息的属性值。

返回值说明

参数名	说明
data	Device上码流数据内存。
size	内存大小，单位Byte。
format	码流格式（h264/h265）。
timestamp	时间戳。
retCode	返回码（0成功，非0失败）。
eos	是否为最后一帧数据。

11.27.73.4 acldvppDestroyStreamDesc

函数功能

销毁通过**acldvppCreateStreamDesc**接口创建的acldvppStreamDesc类型的数据。

函数原型

aclError acldvppDestroyStreamDesc(acldvppStreamDesc *streamDesc)

参数说明

参数名	输入/输出	说明
streamDesc	输入	待销毁的acldvppStreamDesc类型的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.74 aclvdecFrameConfig

11.27.74.1 aclvdecCreateFrameConfig

函数功能

创建aclvdecFrameConfig类型的数据，表示创建VDEC解码时的单帧解码配置参数。

如需销毁aclvdecFrameConfig类型的数据，请参见**aclvdecDestroyFrameConfig**。

函数原型

aclvdecFrameConfig *aclvdecCreateFrameConfig()

参数说明

无

返回值说明

- 返回aclvdecFrameConfig类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.74.2 aclvdecDestroyFrameConfig

函数功能

销毁通过**aclvdecCreateFrameConfig**接口创建的aclvdecFrameConfig类型的数据。

函数原型

aclError aclvdecDestroyFrameConfig(aclvdecFrameConfig *vdecFrameConfig)

参数说明

参数名	输入/输出	说明
vdecFrameConfig	输入	待销毁的aclvdecFrameConfig类型的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.75 acldvppBatchPicDesc

11.27.75.1 acldvppCreateBatchPicDesc

函数功能

创建acldvppBatchPicDesc类型的数据，表示批量图片描述信息。

如需销毁acldvppBatchPicDesc类型的数据，请参见**acldvppDestroyBatchPicDesc**。

函数原型

acldvppBatchPicDesc *acldvppCreateBatchPicDesc(uint32_t batchSize)

参数说明

参数名	输入/输出	说明
batchSize	输入	图片数量。

返回值说明

- 返回acldvppBatchPicDesc类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.75.2 acldvppGetPicDesc

函数功能

获取指定图片的图片描述信息。

函数原型

acldvppPicDesc *acldvppGetPicDesc(acldvppBatchPicDesc *batchPicDesc, uint32_t index)

参数说明

参数名	输入/输出	说明
batchPicDesc	输入	批量图片描述信息的指针。 需提前调用**acldvppCreateBatchPicDesc**接口创建acldvppBatchPicDesc类型的数据。
index	输入	指定获取第几个图片的描述信息，index值从0开始。

返回值说明

返回指定图片的图片描述信息。

11.27.75.3 acldvppDestroyBatchPicDesc

函数功能

销毁通过**acldvppCreateBatchPicDesc**接口创建的acldvppBatchPicDesc类型的数据。

函数原型

aclError acldvppDestroyBatchPicDesc (acldvppBatchPicDesc *batchPicDesc)

参数说明

参数名	输入/输出	说明
batchPicDesc	输入	待销毁的acldvppBatchPicDesc类型的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.76 aclvencChannelDesc

11.27.76.1 aclvencCreateChannelDesc

函数功能

创建aclvencChannelDesc类型的数据，表示创建视频编码处理通道时的通道描述信息。

如需销毁aclvencChannelDesc类型的数据，请参见**aclvencDestroyChannelDesc**。

函数原型

aclvencChannelDesc *aclvencCreateChannelDesc()

参数说明

无

返回值说明

- 返回aclvencChannelDesc类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.76.2 aclvencSetChannelDesc 系列接口

函数功能

设置视频编码处理通道描述信息的属性值。

函数原型

```
aclError aclvencSetChannelDescThreadId(aclvencChannelDesc *channelDesc, uint64_t threadId);
aclError aclvencSetChannelDescCallback(aclvencChannelDesc *channelDesc, aclvencCallback callback);
aclError aclvencSetChannelDescEnType(aclvencChannelDesc *channelDesc, acldvppStreamFormat enType);
aclError aclvencSetChannelDescPicFormat(aclvencChannelDesc *channelDesc, acldvppPixelFormat picFormat);
aclError aclvencSetChannelDescPicWidth(aclvencChannelDesc *channelDesc, uint32_t picWidth);
aclError aclvencSetChannelDescPicHeight(aclvencChannelDesc *channelDesc, uint32_t picHeight);
aclError aclvencSetChannelDescKeyFrameInterval(aclvencChannelDesc *channelDesc, uint32_t keyFrameInterval);
aclError aclvencSetChannelDescBufAddr(aclvencChannelDesc *channelDesc, void *bufAddr);
aclError aclvencSetChannelDescBufSize(aclvencChannelDesc *channelDesc, uint32_t bufSize);
aclError aclvencSetChannelDescRcMode(aclvencChannelDesc *channelDesc, uint32_t rcMode);
aclError aclvencSetChannelDescSrcRate(aclvencChannelDesc *channelDesc, uint32_t srcRate);
aclError aclvencSetChannelDescMaxBitRate(aclvencChannelDesc *channelDesc, uint32_t maxBitRate)
```

参数说明

参数名	输入/输出	说明
channelDesc	输出	视频编码处理通道描述信息的指针。 需提前调用aclvencCreateChannelDesc接口创建aclvencChannelDesc类型的数据。
threadId	输入	回调线程ID。 **说明** 同一个进程内，在不同的Device上注册VENC编码回调函数的线程时，不能指定同一个线程ID。

参数名	输入/输出	说明
callback	输入	编码回调函数。
enType	输入	视频编码协议。
picFormat	输入	输入图像格式，支持如下格式： ● PIXEL_FORMAT_YUV_SEMIPLANAR_420 ● PIXEL_FORMAT_YVU_SEMIPLANAR_420
picWidth	输入	图片宽度。
picHeight	输入	图片高度。
keyFrameInterval	输入	关键帧间隔，取值范围[1,65536]。
bufAddr	输入	编码输出缓存指针。 **说明** 在Atlas 200/300/500 推理产品上，当前版本不支持用户设置该参数，由VENC内部管理该缓存地址。
bufSize	输入	编码输出缓存大小，单位为Byte。 **说明** 在Atlas 200/300/500 推理产品上，当前版本不支持用户设置该参数，内部默认使用3686400 Byte。 在Atlas 推理系列产品上，如果不设置该参数，参数值默认为8M；如果设置该参数，参数值最小为5M。
rcMode	输入	指定码率控制模式。 ● 0表示使用默认值 ● 1表示变码率VBR模式 ● 2表示定码率CBR模式 **说明** 如果不设置该参数，则采用默认值0。 Atlas 200/300/500 推理产品，默认值0表示CBR模式。 Atlas 推理系列产品，默认值0表示VBR模式。
srcRate	输入	输入码流帧率，单位fps。 Atlas 200/300/500 推理产品，取值范围0或者[1,120]。 Atlas 推理系列产品，取值范围0或者[1,240]。 如果不设置该参数，默认为30；如果设置为0，表示使用默认值，即30。如果该值和实际输入码流帧率相差太大，会影响输出码率。
maxBitRate	输入	输出码率，单位kbps。 Atlas 200/300/500 推理产品，取值范围0或[10,30000]。如果不设置该参数，默认为300；如果设置为0，表示使用默认值，即300。 Atlas 推理系列产品，取值范围[2,614400]，如果不设置该参数，默认为2000；如果设置为0，表示使用默认值，即2000。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.76.3 aclvencGetChannelDesc 系列接口

函数功能

获取视频编码处理通道的描述信息。

函数原型

```
uint32_t aclvencGetChannelDescChannelId(const aclvencChannelDesc *channelDesc);

uint64_t aclvencGetChannelDescThreadId(const aclvencChannelDesc *channelDesc);

aclvencCallback aclvencGetChannelDescCallback(const aclvencChannelDesc *channelDesc);

acldvppStreamFormat aclvencGetChannelDescEnType(const aclvencChannelDesc *channelDesc);

acldvppPixelFormat aclvencGetChannelDescPicFormat(const aclvencChannelDesc *channelDesc);

uint32_t aclvencGetChannelDescPicWidth(const aclvencChannelDesc *channelDesc);

uint32_t aclvencGetChannelDescPicHeight(const aclvencChannelDesc *channelDesc);

uint32_t aclvencGetChannelDescKeyFrameInterval(const aclvencChannelDesc *channelDesc);

void *aclvencGetChannelDescBufAddr(const aclvencChannelDesc *channelDesc);

uint32_t aclvencGetChannelDescBufSize(const aclvencChannelDesc *channelDesc);

uint32_t aclvencGetChannelDescRcMode(const aclvencChannelDesc *channelDesc);

uint32_t aclvencGetChannelDescSrcRate(const aclvencChannelDesc *channelDesc);

uint32_t aclvencGetChannelDescMaxBitRate(const aclvencChannelDesc *channelDesc)
```

参数说明

参数名	输入/输出	说明
channelDesc	输入	视频编码处理通道描述信息的指针。 需提前调用**aclvencCreateChannelDesc**接口创建aclvencChannelDesc类型的数据，调用**aclvencSetChannelDesc系列接口**设置视频编码处理通道描述信息的属性值。

返回值说明

参数名	说明
channelId	通道号，默认为0。
threadId	回调线程ID。
callback	编码回调函数。
enType	视频编码协议。
picFormat	输入图像格式。
picWidth	图片宽度。
picHeight	图片高度。
keyFrameInterval	关键帧间隔，不能为0。
bufAddr	编码输出缓存地址。 在Atlas 200/300/500 推理产品上，当前版本获取该参数的值为无效值，不支持获取该参数值。
bufSize	编码输出缓存大小，单位为Byte。 在Atlas 200/300/500 推理产品上，当前版本获取该参数的值为无效值，不支持获取该参数值。
rcMode	指定码率控制模式。 ● 1表示变码率VBR模式 ● 2表示定码率CBR模式
srcRate	输入码流帧率，单位fps。
maxBitRate	输出码率，单位kbps。

11.27.76.4 aclvencSetChannelDescParam

函数功能

设置视频编码处理通道的属性参数接口。

函数原型

aclError aclvencSetChannelDescParam(aclvencChannelDesc *channelDesc, aclvencChannelDescParamType paramType, size_t length, const void *param)

参数说明

参数名	输入/输出	说明
channelDesc	输入	视频编码处理通道描述信息的指针。 需提前调用aclvencCreateChannelDesc接口创建aclvencChannelDesc类型的数据。
paramType	输入	属性参数的类型。
length	输入	属性参数值的字节数。 ● 属性参数的类型为*_UINT64时，此处配置为8。 ● 属性参数的类型为*_UINT32时，此处配置为4。 ● 属性参数的类型为*_PTR时，若操作系统是32位，则此处配置为4；若操作系统是64位，则配置为8。
param	输入	属性参数值的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.76.5 aclvencGetChannelDescParam

函数功能

获取视频编码处理通道的描述信息。

函数原型

aclError aclvencGetChannelDescParam(const aclvencChannelDesc *channelDesc, aclvencChannelDescParamType paramType, size_t length, size_t *paramRetSize, void *param)

参数说明

参数名	输入/输出	说明
channelDesc	输入	视频编码处理通道描述信息的指针。 需提前调用**aclvencCreateChannelDesc**接口创建aclvencChannelDesc类型的数据，调用**aclvencSetChannelDescParam**接口设置视频编码处理通道描述信息的属性值。
paramType	输入	属性参数的类型。
length	输入	属性参数值的字节数。 ● 属性参数的类型为*_UINT64时，此处配置为8； ● 属性参数的类型为*_UINT32时，此处配置为4； ● 属性参数的类型为*_PTR时，若操作系统是32位，则此处配置为4；若操作系统是64位，则配置为8。
paramRetSize	输出	实际返回的属性参数值字节数的指针。
param	输出	属性参数值的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.76.6 aclvencDestroyChannelDesc

函数功能

销毁通过**aclvencCreateChannelDesc**接口创建的aclvencChannelDesc类型的数据。

函数原型

aclError aclvencDestroyChannelDesc(aclvencChannelDesc *channelDesc)

参数说明

参数名	输入/输出	说明
channelDesc	输入	待销毁的aclvencChannelDesc类型的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.77 aclvencFrameConfig

11.27.77.1 aclvencCreateFrameConfig

函数功能

创建aclvencFrameConfig类型的数据，表示VENC编码时的单帧编码配置参数。

如需销毁aclvencFrameConfig类型的数据，请参见**aclvencDestroyFrameConfig**。

函数原型

aclvencFrameConfig *aclvencCreateFrameConfig()

参数说明

无

返回值说明

- 返回aclvencFrameConfig类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.77.2 aclvencSetFrameConfig 系列接口

函数功能

设置单帧编码配置参数。

函数原型

aclError aclvencSetFrameConfigForceIFrame(aclvencFrameConfig *config, uint8_t forceIFrame);

aclError aclvencSetFrameConfigEos(aclvencFrameConfig *config, uint8_t eos)

参数说明

参数名	输入/输出	说明
config	输出	单帧编码配置数据的指针。 需提前调用**aclvencCreateFrameConfig**接口创建aclvencFrameConfig类型的数据。

参数名	输入/输出	说明
forceIFrame	输入	是否强制重新开始I帧间隔： ● 0：不强制 ● 1：强制重新开始I帧
eos	输入	是否为结束帧： ● 0：不是 ● 1：是结束帧

返回值说明

返回0表示成功，返回非0表示失败。

11.27.77.3 aclvencGetFrameConfig 系列接口

函数功能

获取单帧编码配置参数。

函数原型

uint8_t aclvencGetFrameConfigForceIFrame(const aclvencFrameConfig *config);

uint8_t aclvencGetFrameConfigEos(const aclvencFrameConfig *config)

参数说明

参数名	输入/输出	说明
config	输入	单帧编码配置数据的指针。 需提前调用aclvencCreateFrameConfig接口创建aclvencFrameConfig类型的数据，调用aclvencSetFrameConfig系列接口设置单帧编码配置参数。

返回值说明

参数名	说明
forceIFrame	是否强制重新开始I帧间隔： ● 0：不强制 ● 1：强制重新开始I帧

参数名	说明
eos	是否为结束帧： ● 0：不是 ● 1：是结束帧

11.27.77.4 aclvencDestroyFrameConfig

函数功能

销毁通过**aclvencCreateFrameConfig**接口创建的aclvencFrameConfig类型的数据。

函数原型

aclError aclvencDestroyFrameConfig(aclvencFrameConfig *config)

参数说明

参数名	输入/输出	说明
config	输入	待销毁的aclvencFrameConfig类型的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.78 acldvppLutMap

11.27.78.1 acldvppCreateLutMap

函数功能

创建acldvppLutMap类型的数据，表示色彩重映射表。在acldvppLutMap类型中，色彩重映射表的维度数量默认为3。

如需销毁acldvppLutMap类型的数据，请参见**acldvppDestroyLutMap**。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

acldvppLutMap *acldvppCreateLutMap()

参数说明

无

返回值说明

- 返回acldvppLutMap类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.78.2 acldvppDestroyLutMap

函数功能

销毁通过**acldvppCreateLutMap**接口创建的acldvppLutMap类型的数据。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

aclError acldvppDestroyLutMap(acldvppLutMap *lutMap)

参数说明

参数名	输入/输出	说明
lutMap	输入	待销毁的acldvppLutMap类型的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.78.3 acldvppGetLutMapDims

函数功能

获取色彩重映射表的维度数量。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

uint32_t acldvppGetLutMapDims(const acldvppLutMap *lutMap)

参数说明

参数名	输入/输出	说明
lutMap	输入	色彩重映射表的指针。 需提前调用acldvppCreateLutMap接口创建acldvppLutMap类型的数据。

返回值说明

返回维度数量。

11.27.78.4 acldvppGetLutMapData

函数功能

根据指定维度获取色彩重映射表数据的内存指针及数组长度。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

```
aclError acldvppGetLutMapData(const acldvppLutMap *lutMap,
uint32_t dim,
uint8_t **data,
uint32_t *len)
```

参数说明

参数名	输入/输出	说明
lutMap	输入	色彩重映射表的指针。 需提前调用acldvppCreateLutMap接口创建acldvppLutMap类型的数据。
dim	输入	指定色彩重映射表的维度。 调用acldvppGetLutMapDims接口获取色彩重映射表的维度数量，dim的取值范围：[0, (维度数量-1)]。
data	输出	“获取指定维度的色彩重映射表数据的内存指针”的指针。
len	输出	data数组长度的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.79 acldvppBorderConfig

11.27.79.1 acldvppCreateBorderConfig

函数功能

创建acldvppBorderConfig类型的数据，表示边界填充配置。

如需销毁acldvppBorderConfig类型的数据，请参见**acldvppDestroyBorderConfig**。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

acldvppBorderConfig *acldvppCreateBorderConfig()

参数说明

无

返回值说明

- 返回acldvppBorderConfig类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.79.2 acldvppSetBorderConfig 系列接口

函数功能

设置边界填充配置参数。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

```
aclError acldvppSetBorderConfigValue(acldvppBorderConfig *borderConfig,
uint32_t index, double value);

aclError acldvppSetBorderConfigBorderType(acldvppBorderConfig
*borderConfig, acldvppBorderType borderType);

aclError acldvppSetBorderConfigTop(acldvppBorderConfig *borderConfig,
uint32_t top);

aclError acldvppSetBorderConfigBottom(acldvppBorderConfig *borderConfig,
uint32_t bottom);

aclError acldvppSetBorderConfigLeft(acldvppBorderConfig *borderConfig,
uint32_t left);
```

aclError acldvppSetBorderConfigRight(acldvppBorderConfig *borderConfig, uint32_t right)

参数说明

参数名	输入/输出	说明
borderConfig	输出	边界填充配置数据的指针。 需提前调用acldvppCreateBorderConfig接口创建acldvppBorderConfig类型的数据。
index	输入	表示颜色分量数组的下标。 0表示R或Y分量，1表示G或U分量，2表示B或V分量，3是预留值。
value	输入	指定填充的各颜色分量的像素值。 仅当borderType设置为BORDER_CONSTANT时，该参数有效。 当前支持用户手动按顺序存放R、G、B分量或Y、U、V分量的值。若输入输出都是YUV格式，填充YUV分量的值；反之，填充的是RGB分量的值。
borderType	输入	填充枚举类型。
top	输入	图像上方填充的像素数。
bottom	输入	图像下方填充的像素数。
left	输入	图像左方填充的像素数。
right	输入	图像右方填充的像素数。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.79.3 acldvppGetBorderConfig 系列接口

函数功能

获取边界填充配置参数。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

double acldvppGetBorderConfigValue(const acldvppBorderConfig *borderConfig, uint32_t index);

```
acldvppBorderType acldvppGetBorderConfigBorderType(const acldvppBorderConfig *borderConfig);
uint32_t acldvppGetBorderConfigTop(const acldvppBorderConfig *borderConfig);
uint32_t acldvppGetBorderConfigBottom(const acldvppBorderConfig *borderConfig);
uint32_t acldvppGetBorderConfigLeft(const acldvppBorderConfig *borderConfig);
uint32_t acldvppGetBorderConfigRight(const acldvppBorderConfig *borderConfig)
```

参数说明

参数名	输入/输出	说明
borderConfig	输入	边界填充配置数据的指针。 需先调用acldvppCreateBorderConfig接口创建acldvppBorderConfig类型的数据，调用acldvppSetBorderConfig系列接口设置边界填充数据。
index	输入	表示颜色分量数组的下标。 0表示R或Y分量，1表示G或U分量，2表示B或V分量，3是预留值。

返回值说明

参数名	说明
value	表示对应颜色分量的值。
borderType	填充枚举类型。
top	图像上方填充的像素数。
bottom	图像下方填充的像素数。
left	图像左方填充的像素数。
right	图像右方填充的像素数。

11.27.79.4 acldvppDestroyBorderConfig

函数功能

销毁通过acldvppCreateBorderConfig接口创建的acldvppBorderConfig类型的数据。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

aclError acldvppDestroyBorderConfig(acldvppBorderConfig *borderConfig)

参数说明

参数名	输入/输出	说明
borderConfig	输入	待销毁的acldvppBorderConfig类型的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.80 acldvppHist

11.27.80.1 acldvppCreateHist

函数功能

创建acldvppHist类型的数据，用于描述颜色分布直方图信息。在acldvppHist类型中，直方图的维度数量默认为3。

如需销毁acldvppHist类型的数据，请参见**acldvppDestroyHist**。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

acldvppHist* acldvppCreateHist()

参数说明

无

返回值说明

- 返回acldvppHist类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.80.2 acldvppDestroyHist

函数功能

销毁通过**acldvppCreateHist**接口创建的acldvppHist类型的数据。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

aclError acldvppDestroyHist(acldvppHist *hist)

参数说明

参数名	输入/输出	说明
hist	输入	待销毁的acldvppHist类型的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.80.3 acldvppGetHistDims

函数功能

获取直方图数据的维度数量。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

uint32_t acldvppGetHistDims(acldvppHist *hist)

参数说明

参数名	输入/输出	说明
hist	输入	直方图描述信息的指针。 需提前调用acldvppCreateHist接口创建直方图描述信息。

返回值说明

返回维度数量。

11.27.80.4 acldvppGetHistData

函数功能

按指定维度获取直方图数据。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

aclError acldvppGetHistData(acldvppHist *hist, uint32_t dim, uint32_t **data, uint16_t *len)

参数说明

参数名	输入/输出	说明
hist	输入	直方图描述信息的指针。 需提前调用acldvppVpcCalcHistAsync接口统计各维度直方图数据。
dim	输入	直方图维度。 调用acldvppGetHistDims接口获取直方图的维度数量，dim的取值范围：[0, (维度数量-1)]。
data	输出	“对应维度的直方图数据的首地址指针”的指针。
len	输出	直方图数据的长度的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.80.5 acldvppGetHistRetCode

函数功能

获取返回码。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

uint32_t acldvppGetHistRetCode(acldvppHist* hist)

参数说明

参数名	输入/输出	说明
hist	输入	直方图描述信息的指针。 需提前调用acldvppVpcCalcHistAsync接口统计各维度直方图数据。

返回值说明

获取直方图统计是否成功的返回码，0表示成功，非0表示失败。

11.27.80.6 acldvppClearHist

函数功能

将直方图统计数据清零。

使用场景：在统计完一张图像的颜色分布直方图信息后，如果想重复使用acldvppHist类型的变量，可以调用本接口先将acldvppHist类型中的直方图统计数据清零，再统计下一张图像的颜色分布直方图信息。

Atlas 200/300/500 推理产品，不支持该接口。

函数原型

aclError acldvppClearHist(acldvppHist *hist)

参数说明

参数名	输入/输出	说明
hist	输入	直方图描述信息的指针。 需提前调用**acldvppCreateHist**接口创建acldvppHist类型的数据。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.81 aclfvRepoRange

11.27.81.1 aclfvCreateRepoRange

函数功能

创建aclfvRepoRange类型的数据，表示创建特征库范围的参数。

如需销毁aclfvRepoRange类型的数据，请参见**aclfvDestroyRepoRange**。

Atlas 200/300/500 推理产品不支持该接口。

函数原型

aclfvRepoRange *aclfvCreateRepoRange(uint32_t id0Min, uint32_t id0Max, uint32_t id1Min, uint32_t id1Max)

参数说明

参数名	输入/输出	说明
id0Min	输入	id0起始值，取值范围：[0,1023]，需小于等于id0Max。
id0Max	输入	id0最大值，取值范围：[0,1023]。
id1Min	输入	id1起始值，取值范围：[0,1023]，需小于等于id1Max
id1Max	输入	id1最大值，取值范围：[0,1023]。

返回值说明

- 返回aclfvRepoRange类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.81.2 aclfvDestroyRepoRange

函数功能

销毁通过**aclfvCreateRepoRange**接口创建的aclfvRepoRange类型的数据。

Atlas 200/300/500 推理产品不支持该接口。

函数原型

aclError aclfvDestroyRepoRange(aclfvRepoRange *repoRange)

参数说明

参数名	输入/输出	说明
repoRange	输入	待销毁的aclfvRepoRange类型的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.82 aclfvFeatureInfo

11.27.82.1 aclfvCreateFeatureInfo

函数功能

创建aclfvFeatureInfo类型的数据，表示创建特征描述信息。

如需销毁aclfvFeatureInfo类型的数据，请参见**aclfvDestroyFeatureInfo**。

Atlas 200/300/500 推理产品不支持该接口。

函数原型

aclfvFeatureInfo *aclfvCreateFeatureInfo(uint32_t id0, uint32_t id1, uint32_t offset, uint32_t featureLen, uint32_t featureCount, uint8_t *featureData, uint32_t featureDataLen)

参数说明

参数名	输入/输出	说明
id0	输入	一级库id，取值范围0-1023，N:M场景默认为0。 通过id0、id1共同确定一个库。
id1	输入	二级库id，取值范围0-1023，N:M场景默认为0。 通过id0、id1共同确定一个库。
offset	输入	添加的首个特征在库中的偏移，偏移值需要与库中已添加特征个数一致。N:M场景默认为0。
featureLen	输入	单个特征长度，当前固定为36Byte，系统内部会校验。
featureCount	输入	特征数量，1:N最大100万，N:M最大1000万。
featureData	输入	特征值指针，根据特征长度连续存储，每条特征前4Byte值为0。
featureDataLen	输入	featureData指针申请的内存长度，用于校验。

返回值说明

- 返回aclfvFeatureInfo类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.82.2 aclfvDestroyFeatureInfo

函数功能

销毁通过**aclfvCreateFeatureInfo**接口创建的aclfvFeatureInfo类型的数据。

Atlas 200/300/500 推理产品不支持该接口。

函数原型

aclError aclfvDestroyFeatureInfo(aclfvFeatureInfo *featureInfo)

参数说明

参数名	输入/输出	说明
featureInfo	输入	待销毁的aclfvFeatureInfo类型的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.83 aclfvQueryTable

11.27.83.1 aclfvCreateQueryTable

函数功能

创建aclfvQueryTable类型的数据，表示创建检索输入表信息。

如需销毁aclfvQueryTable类型的数据，请参见**aclfvDestroyQueryTable**。

Atlas 200/300/500 推理产品不支持该接口。

函数原型

aclfvQueryTable *aclfvCreateQueryTable(uint32_t queryCnt, uint32_t tableLen, uint8_t *tableData, uint32_t tableDataLen)

参数说明

参数名	输入/输出	说明
queryCnt	输入	检索请求数量，1:N场景为1，N:M场景最大1024。

参数名	输入/输出	说明
tableLen	输入	单个表长度，长度固定为32KB，系统内部会校验。
tableData	输入	特征值列表的指针，根据特征长度连续存储。
tableDataLen	输入	tableData指针申请的内存长度，用于校验。

返回值说明

- 返回aclfvQueryTable类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.83.2 aclfvDestroyQueryTable

函数功能

销毁通过**aclfvCreateQueryTable**接口创建的aclfvQueryTable类型的数据。

Atlas 200/300/500 推理产品不支持该接口。

函数原型

aclError aclfvDestroyQueryTable(aclfvQueryTable *queryTable)

参数说明

参数名	输入/输出	说明
queryTable	输入	待销毁的aclfvQueryTable类型的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.84 aclfvSearchInput

11.27.84.1 aclfvCreateSearchInput

函数功能

创建aclfvSearchInput类型的数据，表示创建检索任务输入信息。

如需销毁aclfvSearchInput类型的数据，请参见**aclfvDestroySearchInput**。

Atlas 200/300/500 推理产品不支持该接口。

函数原型

aclfvSearchInput *aclfvCreateSearchInput(aclfvQueryTable *queryTable, aclfvRepoRange *repoRange, uint32_t topk)

参数说明

参数名	输入/输出	说明
queryTable	输入	检索输入表信息的指针。 调用aclfvCreateQueryTable接口创建**aclfvQueryTable**类型的数据。
repoRange	输入	检索特征库范围的指针。 调用aclfvCreateRepoRange接口创建**aclfvRepoRange**类型的数据。
topk	输入	单个检索请求需要返回的结果数量。

返回值说明

- 返回aclfvSearchInput类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.84.2 aclfvDestroySearchInput

函数功能

销毁通过**aclfvCreateSearchInput**接口创建的aclfvSearchInput类型的数据。

Atlas 200/300/500 推理产品不支持该接口。

函数原型

aclError aclfvDestroySearchInput(aclfvSearchInput *searchInput)

参数说明

参数名	输入/输出	说明
searchInput	输入	待销毁的aclfvSearchInput类型的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.85 aclfvSearchResult

11.27.85.1 aclfvCreateSearchResult

函数功能

创建aclfvSearchResult类型的数据，表示创建检索结果信息。

如需销毁aclfvSearchResult类型的数据，请参见**aclfvDestroySearchResult**。

Atlas 200/300/500 推理产品不支持该接口。

函数原型

aclfvSearchResult *aclfvCreateSearchResult(uint32_t queryCnt, uint32_t *resultNum, uint32_t resultNumDataLen, uint32_t *id0, uint32_t *id1, uint32_t *resultOffset, float *resultDistance, uint32_t dataLen)

参数说明

参数名	输入/输出	说明
queryCnt	输入	检索请求个数，1:N场景为1。
resultNum	输入	结果个数的指针。 每个检索请求的结果个数。
resultNumDataLen	输入	resultNum的内存总长度。
id0	输入	一级库id的指针，总个数为topK*queryCnt。
id1	输入	二级库id的指针，总个数为topK*queryCnt。
resultOffset	输入	偏移指针。 表示每个检索请求的检索结果对应底库的偏移，总个数为topK*queryCnt。
resultDistance	输入	每个检索结果与检索请求间的距离的指针。 总个数为topK*queryCnt。
dataLen	输入	申请的内存大小，计算公式为：topK*queryCnt*4Byte。

返回值说明

- 返回aclfvSearchResult类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.85.2 aclfvDestroySearchResult

函数功能

销毁通过**aclfvCreateSearchResult**接口创建的aclfvSearchResult类型的数据。

Atlas 200/300/500 推理产品不支持该接口。

函数原型

aclError aclfvDestroySearchResult(aclfvSearchResult *searchResult)

参数说明

参数名	输入/输出	说明
searchResult	输入	待销毁的aclfvSearchResult类型的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.86 aclfvInitPara

11.27.86.1 aclfvCreateInitPara

函数功能

创建aclfvInitPara类型的数据，表示创建特征向量检索的初始化参数。

如需销毁aclfvInitPara类型的数据，请参见**aclfvDestroyInitPara**。

Atlas 200/300/500 推理产品不支持该接口。

函数原型

aclfvInitPara *aclfvCreateInitPara(uint64_t fsNum)

参数说明

参数名	输入/输出	说明
fsNum	输入	底库特征个数，用于系统内部给特征检索模块申请内存资源。 取值范围：[1,500000000]

返回值说明

- 返回aclfvInitPara类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.86.2 aclfvDestroyInitPara

函数功能

销毁通过**aclfvCreateInitPara**接口创建的aclfvInitPara类型的数据。

Atlas 200/300/500 推理产品不支持该接口。

函数原型

aclError aclfvDestroyInitPara(aclfvInitPara *initPara)

参数说明

参数名	输入/输出	说明
initPara	输入	待销毁的aclfvInitPara类型的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.86.3 aclfvSet1NTopNum

函数功能

设置1:N场景下返回的topK结果数量的最大值，如果不设置，默认topK结果数量的最大值为4800。

Atlas 200/300/500 推理产品不支持该接口。

函数原型

aclError aclfvSet1NTopNum(aclfvInitPara *initPara, uint32_t maxTopNumFor1N)

参数说明

参数名	输入/输出	说明
initPara	输入&输出	特征向量检索初始化参数的指针。 需提前调用aclfvCreateInitPara接口创建aclfvInitPara类型的数据。
maxTopNumFor1N	输入	1:N场景下返回的topK结果的最大值，取值范围：[2,4800]。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.86.4 aclfvSetNMTopNum

函数功能

设置N:M场景下返回的topK结果数量的最大值，如果不设置，默认topK结果数量的最大值为500。

Atlas 200/300/500 推理产品不支持该接口。

函数原型

aclError aclfvSetNMTopNum(aclfvInitPara *initPara, uint32_t maxTopNumForNM)

参数说明

参数名	输入/输出	说明
initPara	输入&输出	特征向量检索初始化参数的指针。 需提前调用aclfvCreateInitPara接口创建aclfvInitPara类型的数据。
maxTopNumForNM	输入	N:M场景下返回的topK结果的最大值，取值范围：[500,4800]。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.87 aclprofConfig

11.27.87.1 aclprofCreateConfig

函数功能

创建aclprofConfig类型的数据，表示创建Profiling配置数据。

aclProfConfig类型数据可以只创建一次、多处使用，用户需要保证数据的一致性和准确性。

如需销毁aclprofConfig类型的数据，请参见**aclprofDestroyConfig**。

约束说明

- 使用aclprofDestroyConfig接口销毁aclprofConfig类型的数据，如不销毁会导致内存未被释放。
- 与**aclprofDestroyConfig**接口配对使用，先调用aclprofCreateConfig接口再调用aclprofDestroyConfig接口。

函数原型

aclprofConfig *aclprofCreateConfig(uint32_t *deviceIdList, uint32_t deviceNums, aclprofAicoreMetrics aicoreMetrics, aclprofAicoreEvents *aicoreEvents, uint64_t dataTypeConfig)

参数说明

参数名	输入/输出	说明
deviceIdList	输入	Device ID列表。
deviceNums	输入	Device的个数。需由用户保证deviceIdList中的Device个数与deviceNums参数值一致，否则可能会导致后续业务异常。
aicoreMetrics	输入	表示AI Core性能指标采集项。
aicoreEvents	输入	表示AI Core事件，目前配置为NULL。

参数名	输入/输出	说明
dataTypeConfig	输入	用户选择如下多个宏进行逻辑或（例如：ACL_PROF_ACL_API | ACL_PROF_AICORE_METRICS），作为dataTypeConfig参数值。每个宏表示某一类性能数据，详细说明如下： ● ACL_PROF_ACL_API：表示采集AscendCL接口的性能数据，包括Device间的同步异步内存复制时延等。 ● ACL_PROF_TASK_TIME：采集算子下发耗时、算子执行耗时数据以及算子基本信息数据，提供更全面的性能分析数据。 ● ACL_PROF_TASK_TIME_L0：采集算子下发耗时、算子执行耗时数据。与ACL_PROF_TASK_TIME相比，由于不采集算子基本信息数据，采集时性能开销较小，可更精准统计相关耗时数据。 ● ACL_PROF_AICORE_METRICS：表示采集AI Core性能指标数据，逻辑或时必须包括该宏，aicoreMetrics入参处配置的性能指标采集项才有效。 ● ACL_PROF_TASK_MEMORY：控制CANN算子的内存占用情况采集开关。仅采集GE组件算子。 ● ACL_PROF_AICPU：表示采集AI CPU任务的开始、结束数据。 ● ACL_PROF_L2CACHE：表示采集L2 Cache数据。 ● ACL_PROF_HCCL_TRACE：控制HCCL数据采集开关。 ● ACL_PROF_MSPROFTX：获取用户和上层框架程序输出的性能数据。 ● ACL_PROF_RUNTIME_API：控制runtime api性能数据采集开关。

返回值说明

- 返回aclprofConfig类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.87.2 aclprofDestroyConfig

函数功能

销毁通过**aclprofCreateConfig**接口创建的aclprofConfig类型的数据。

约束说明

- 与**aclprofCreateConfig**接口配对使用，先调用aclprofCreateConfig接口再调用aclprofDestroyConfig接口。
- 同一aclprofConfig指针重复调用aclprofDestroyConfig接口，会出现重复释放内存的报错。

函数原型

aclError aclprofDestroyConfig(const aclprofConfig *profilerConfig)

参数说明

参数名	输入/输出	说明
profilerConfig	输入	待销毁的aclprofConfig类型的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.88 aclprofSubscribeConfig

11.27.88.1 aclprofCreateSubscribeConfig

函数功能

创建aclprofSubscribeConfig类型的数据，表示创建订阅配置信息。

如需销毁aclprofSubscribeConfig类型的数据，请参见**aclprofDestroySubscribeConfig**。

约束说明

- 使用aclprofDestroySubscribeConfig接口销毁aclprofSubscribeConfig类型的数据，如不销毁会导致内存未被释放。
- 与**aclprofDestroySubscribeConfig**接口配对使用，先调用aclprofCreateSubscribeConfig接口再调用aclprofDestroySubscribeConfig接口。

函数原型

aclprofSubscribeConfig *aclprofCreateSubscribeConfig(int8_t timeInfoSwitch, aclprofAicoreMetrics aicoreMetrics, void *fd)

参数说明

参数名	输入/输出	说明
timeInfoSwitch	输入	是否采集网络模型中算子的性能数据： ● 1：采集 ● 0：不采集
aicoreMetrics	输入	表示AI Core性能指标采集项。 **说明** 订阅接口目前仅提供算子耗时统计的功能，暂时不支持AicoreMetrics采集功能。
fd	输入	用户创建的管道写指针。 用户在调用aclprofModelUnSubscribe接口后，系统内部会在数据发送结束后，关闭该模型的管道写指针。

返回值说明

- 返回aclprofSubscribeConfig类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.88.2 aclprofDestroySubscribeConfig

函数功能

销毁通过**aclprofCreateSubscribeConfig**接口创建的aclprofSubscribeConfig类型的数据。

约束说明

- 与**aclprofCreateSubscribeConfig**接口配对使用，先调用aclprofCreateSubscribeConfig接口再调用aclprofDestroySubscribeConfig接口。
- 同一aclprofSubscribeConfig指针重复调用aclprofDestroySubscribeConfig接口，会出现重复释放内存的报错。

函数原型

aclError aclprofDestroySubscribeConfig(const aclprofSubscribeConfig *profSubscribeConfig)

参数说明

参数名	输入/输出	说明
profSubscribeConfig	输入	待销毁的aclprofSubscribeConfig类型的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.89 aclprofStepInfo

11.27.89.1 aclprofCreateStepInfo

函数功能

创建aclprofStepInfo对象，用于描述迭代信息。

约束说明

- 使用aclprofDestroyStepInfo接口销毁aclprofStepInfo类型的数据，如不销毁会导致内存未被释放。
- 与**aclprofDestroyStepInfo**接口配对使用，先调用aclprofCreateStepInfo接口再调用aclprofDestroyStepInfo接口。

函数原型

aclprofStepInfo* aclprofCreateStepInfo(void)

返回值

- 返回aclprofStepInfo类型的指针，表示成功。
- 返回nullptr，表示失败。

说明

同一个aclprofStepInfo对象、同一个tag只能设置一次，否则Profiling解析会出错。

11.27.89.2 aclprofDestroyStepInfo

函数功能

销毁通过**aclprofCreateStepInfo**接口创建的aclprofStepInfo类型的数据。

约束说明

- 与**aclprofCreateStepInfo**接口配对使用，先调用aclprofCreateStepInfo接口再调用aclprofDestroyStepInfo接口。
- 同一aclprofStepInfo数据重复调用aclprofDestroyStepInfo接口，会出现重复释放内存的报错。

函数原型

void aclprofDestroyStepInfo (aclprofStepInfo * stepinfo)

参数说明

参数名	输入/输出	说明
stepinfo	输入	指定迭代信息。需提前调用**aclprofCreateStepInfo**接口创建**aclprofStepInfo**类型的数据。

11.27.90 acltdtDataItem

11.27.90.1 acltdtCreateDataItem

函数功能

创建acltdtDataItem类型的数据，代表一个业务上的Tensor。

如需销毁acltdtDataItem类型的数据，请参见**acltdtDestroyDataItem**。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

```
acltdtDataItem *acltdtCreateDataItem(acltdtTensorType tdtType,
const int64_t *dims,
size_t dimNum,
aclDataType dataType,
void *data,
size_t size)
```

参数说明

参数名	输入/输出	说明
tdtType	输入	一个枚举，这个tensor是正常数据，还是一个end数据标识，还是一个异常数据标识。
dims	输入	tensor的Shape。
dimNum	输入	tensor的Shape中的维度个数。
dataType	输入	正常数据里的数据类型。
data	输入	数据地址指针。
size	输入	数据长度。

返回值说明

- 返回acltdtDataItem类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.90.2 acltdtDestroyDataItem

函数功能

销毁通过**acltdtCreateDataItem**接口创建的acltdtDataItem类型的数据。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

aclError acltdtDestroyDataItem(acltdtDataItem *dataItem)

参数说明

参数名	输入/输出	说明
dataItem	输入	待销毁的acltdtDataItem类型的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.90.3 acltdtGetTensorTypeFromItem

函数功能

收到dataItem数据之后，从数据描述里首先check这个数据是个正常数据，还是一个end数据，还是一个异常数据。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

acltdtTensorType acltdtGetTensorTypeFromItem(const acltdtDataItem *dataItem)

参数说明

参数名	输入/输出	说明
dataItem	输入	acltdtDataItem类型的指针。 需提前调用**acltdtCreateDataItem**接口创建acltdtDataItem类型的数据。

返回值说明

请参见**acltdtTensorType**。

11.27.90.4 acltdtGetDataTypeFromItem

函数功能

获取正常数据的数据类型。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

aclDataType acltdtGetDataTypeFromItem(const acltdtDataItem *dataItem)

参数说明

参数名	输入/输出	说明
dataItem	输入	acltdtDataItem类型的指针。 需提前调用**acltdtCreateDataItem**接口创建acltdtDataItem类型的数据。

返回值说明

请参见**acltdtTensorType**。

11.27.90.5 acltdtGetDataAddrFromItem

函数功能

得到正常数据的数据地址。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

void *acltdtGetDataAddrFromItem(const acltdtDataItem *dataItem)

参数说明

参数名	输入/输出	说明
dataItem	输入	acltdtDataItem类型的指针。 需提前调用**acltdtCreateDataItem**接口创建acltdtDataItem类型的数据。

返回值说明

返回正常数据的数据地址。

11.27.90.6 acltdtGetDataSizeFromItem

函数功能

得到正常数据的数据长度。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

size_t acltdtGetDataSizeFromItem(const acltdtDataItem *dataItem)

参数说明

参数名	输入/输出	说明
dataItem	输入	acltdtDataItem类型的指针。 需提前调用**acltdtCreateDataItem**接口创建acltdtDataItem类型的数据。

返回值说明

请参见**acltdtTensorType**。

11.27.90.7 acltdtGetDimNumFromItem

函数功能

得到正常tensor数据的数据Shape中的维度个数。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

size_t acltdtGetDimNumFromItem(const acltdtDataItem *dataItem)

参数说明

参数名	输入/输出	说明
dataItem	输入	acltdtDataItem类型的指针。 需提前调用**acltdtCreateDataItem**接口创建acltdtDataItem类型的数据。

返回值说明

请参见**acltdtTensorType**。

11.27.90.8 acltdtGetDimsFromItem

函数功能

得到正常tensor数据的数据Shape。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

aclError acltdtGetDimsFromItem(const acltdtDataItem *dataItem, int64_t *dims, size_t dimNum)

参数说明

参数名	输入/输出	说明
dataItem	输入	acltdtDataItem类型的指针。 需提前调用**acltdtCreateDataItem**接口创建acltdtDataItem类型的数据。
dims	输入&输出	维度信息数组。
dimNum	输入	维度个数。

返回值说明

请参见**acltdtTensorType**。

11.27.91 acltdtDataset

11.27.91.1 acltdtCreateDataset

函数功能

创建acltdtDataset类型的数据，对等一个Vector<tensor>。

如需销毁acltdtDataset类型的数据，请参见**acltdtDestroyDataset**。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

acltdtDataset *acltdtCreateDataset()

参数说明

无

返回值说明

- 返回acltdtDataset类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.91.2 acltdtDestroyDataset

函数功能

销毁通过**acltdtCreateDataset**接口创建的acltdtDataset类型的数据。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

aclError acltdtDestroyDataset(acltdtDataset *dataset)

参数说明

参数名	输入/输出	说明
dataset	输入	待销毁的acltdtDataset类型的指针。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.91.3 acltdtGetDataItem

函数功能

获取acltdtDataset中的第n个acltdtDataItem。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

acltdtDataItem *acltdtGetDataItem(const acltdtDataset *dataset, size_t index)

参数说明

参数名	输入/输出	说明
dataset	输入	acltdtDataset类型的指针。 需提前调用**acltdtCreateDataset**接口创建acltdtDataset类型的数据，再调用**acltdtAddDataItem**接口添加acltdtDataItem。
index	输入	表明获取的是第几个acltdtDataItem。

返回值说明

- 获取成功，返回acltdtDataItem的地址。
- 获取失败返回空地址。

11.27.91.4 acltdtAddDataItem

函数功能

向acltdtDataset中增加acltdtDataItem。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

aclError acltdtAddDataItem(acltdtDataset *dataset, acltdtDataItem *dataItem)

参数说明

参数名	输入/输出	说明
dataset	输出	待增加acltdtDataItem的acltdtDataset地址指针。 需提前调用**acltdtCreateDataset**接口创建acltdtDataset类型的数据。
dataItem	输入	待增加的acltdtDataItem地址指针。 需提前调用**acltdtCreateDataItem**接口创建acltdtDataItem类型的数据。

返回值说明

返回0表示成功，返回非0表示失败。

11.27.91.5 acltdtGetDatasetSize

函数功能

获取acltdtDataset中acltdtDataItem的个数。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

size_t acltdtGetDatasetSize(const acltdtDataset *dataset)

参数说明

参数名	输入/输出	说明
dataset	输入	acltdtDataset类型的指针。 需提前调用**acltdtCreateDataset**接口创建acltdtDataset类型的数据。

返回值说明

acltdtDataItem的个数。

11.27.91.6 acltdtGetDatasetName

函数功能

获取标识acltdtDataset（对等一个Vector<tensor>）的描述符。

Atlas 200/300/500 推理产品，不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

const char *acltdtGetDatasetName (const acltdtDataset *dataset)

参数说明

参数名	输入/输出	说明
dataset	输入	acltdtDataset类型的指针。 需提前调用**acltdtCreateDataset**接口创建acltdtDataset类型的数据。

返回值说明

标识acltdtDataset（对等一个Vector<tensor>）的描述符的指针。

11.27.92 aclmdlConfigHandle

11.27.92.1 aclmdlCreateConfigHandle

函数功能

创建aclmdlConfigHandle类型的数据，表示一个模型加载的配置对象。

如需销毁aclmdlConfigHandle类型的数据，请参见**aclmdlDestroyConfigHandle**。

函数原型

aclmdlConfigHandle *aclmdlCreateConfigHandle()

参数说明

无

返回值说明

- 返回aclmdlConfigHandle类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.92.2 aclmdlDestroyConfigHandle

函数功能

销毁通过**aclmdlCreateConfigHandle**接口创建的aclmdlConfigHandle类型的数据。

函数原型

aclError aclmdlDestroyConfigHandle(aclmdlConfigHandle *handle)

参数说明

参数名	输入/输出	说明
handle	输入	待销毁的aclmdlConfigHandle类型的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.93 acltdtQueueAttr

11.27.93.1 acltdtCreateQueueAttr

函数功能

创建acltdtQueueAttr类型的数据，表示队列属性配置信息。

函数原型

acltdtQueueAttr *acltdtCreateQueueAttr()

参数说明

无

返回值说明

- 返回acltdtQueueAttr类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.93.2 acltdtDestroyQueueAttr

函数功能

销毁通过acltdtCreateQueueAttr接口创建的acltdtQueueAttr类型的数据。

函数原型

aclError acltdtDestroyQueueAttr(const acltdtQueueAttr *attr)

参数说明

参数名	输入/输出	说明
attr	输入	待销毁的acltdtQueueAttr类型的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.93.3 acltdtSetQueueAttr

函数功能

设置队列属性值。

函数原型

aclError acltdtSetQueueAttr(acltdtQueueAttr *attr, acltdtQueueAttrType type, size_t len, const void *param)

参数说明

参数名	输入/输出	说明
attr	输入&输出	队列属性配置信息的指针。 需提前调用**acltdtCreateQueueAttr**接口创建acltdtQueueAttr类型的数据。
type	输入	属性类型。
len	输入	属性值的字节数。 ● 属性参数的类型为*_UINT64时，此处配置为8； ● 属性参数的类型为*_UINT32时，此处配置为4； ● 属性参数的类型为*_PTR时，若操作系统是32位，则此处配置为4；若操作系统是64位，则配置为8。
param	输入	指向属性值的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.93.4 acltdtGetQueueAttr

函数功能

获取队列属性值。

函数原型

aclError acltdtGetQueueAttr(const acltdtQueueAttr *attr, acltdtQueueAttrType type, size_t len, size_t *paramRetSize, void *param)

参数说明

参数名	输入/输出	说明
attr	输入	队列属性配置信息的指针。 需提前调用acltdtCreateQueueAttr接口创建acltdtQueueAttr类型的数据。
type	输入	属性类型。
len	输入	属性值的字节数。 ● 属性参数的类型为*_UINT64时，此处配置为8； ● 属性参数的类型为*_UINT32时，此处配置为4； ● 属性参数的类型为*_PTR时，若操作系统是32位，则此处配置为4；若操作系统是64位，则配置为8。
paramRetSize	输出	实际返回的属性值字节数的指针。
param	输出	指向属性值的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.94 acltdtQueueRoute

11.27.94.1 acltdtCreateQueueRoute

函数功能

创建acltdtQueueRoute类型的数据，表示创建队列路由配置。

函数原型

acltdtQueueRoute* acltdtCreateQueueRoute(uint32_t srcId, uint32_t dstId)

参数说明

参数名	输入/输出	说明
srcId	输入	源队列ID。
dstID	输入	目的队列ID。

返回值说明

- 返回acltdtQueueRoute类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.94.2 acltdtDestroyQueueRoute

函数功能

销毁通过**acltdtCreateQueueRoute**接口创建的acltdtQueueRoute类型的数据。

函数原型

aclError acltdtDestroyQueueRoute(const acltdtQueueRoute *route)

参数说明

参数名	输入/输出	说明
route	输入	待销毁的acltdtQueueRoute类型的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.94.3 acltdtGetQueueRouteParam

函数功能

获取队列路由配置的相关信息，例如源队列ID、目标队列ID等。

函数原型

aclError acltdtGetQueueRouteParam(const acltdtQueueRoute *route, acltdtQueueRouteParamType type, size_t len, size_t *paramRetSize, void *param)

参数说明

参数名	输入/输出	说明
route	输入	队列路由配置信息的指针。 需提前调用acltdtCreateQueueRoute接口创建acltdtQueueRoute类型的数据。
type	输入	路由参数类型。
len	输入	参数值的字节数。 ● 参数的类型为*_UINT64时，此处配置为8； ● 参数的类型为*_UINT32时，此处配置为4； ● 参数的类型为*_PTR时，若操作系统是32位，则此处配置为4；若操作系统是64位，则配置为8。
paramRetSize	输出	实际返回的参数值字节数的指针。
param	输出	指向参数值的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.95 acltdtQueueRouteList

11.27.95.1 acltdtCreateQueueRouteList

函数功能

创建acltdtQueueRouteList类型的数据，表示队列路由配置数组。

函数原型

acltdtQueueRouteList* acltdtCreateQueueRouteList()

参数说明

无

返回值说明

- 返回acltdtQueueRouteList类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.95.2 acltdtDestroyQueueRouteList

函数功能

销毁通过**acltdtCreateQueueRouteList**接口创建的acltdtQueueRouteList类型的数据。

函数原型

aclError acltdtDestroyQueueRouteList(const acltdtQueueRouteList *routeList)

参数说明

参数名	输入/输出	说明
routeList	输入	待销毁的acltdtQueueRouteList类型的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.95.3 acltdtAddQueueRoute

函数功能

向队列路由配置数组中添加队列路由配置信息。

函数原型

aclError acltdtAddQueueRoute(acltdtQueueRouteList *routeList, const acltdtQueueRoute *route)

参数说明

参数名	输入/输出	说明
routeList	输入&输出	队列路由配置数组。 需提前调用**acltdtCreateQueueRouteList**接口创建acltdtQueueRouteList类型的数据。
route	输入	需添加的队列路由配置信息的指针。 需提前调用**acltdtCreateQueueRoute**接口创建acltdtQueueRoute类型的数据。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.95.4 acltdtGetQueueRoute

函数功能

从队列路由配置数组中获取指定的队列路由配置信息。

函数原型

aclError acltdtGetQueueRoute(const acltdtQueueRouteList *routeList, size_t index, acltdtQueueRoute *route)

参数说明

参数名	输入/输出	说明
routeList	输入	队列路由配置数组。 需提前调用acltdtCreateQueueRouteList接口创建acltdtQueueRouteList类型的数据。
index	输入	指定获取哪一个队列路由配置信息，index编号从0开始。
route	输入&输出	需添加的队列路由配置信息的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.95.5 acltdtGetQueueRouteNum

函数功能

从队列路由配置数组中获取路由数量。

函数原型

size_t acltdtGetQueueRouteNum(const acltdtQueueRouteList *routeList)

参数说明

参数名	输入/输出	说明
routeList	输入	队列路由配置数组。 需提前调用acltdtCreateQueueRouteList接口创建acltdtQueueRouteList类型的数据。

返回值说明

返回路由数量。

11.27.96 acltdtQueueRouteQueryInfo

11.27.96.1 acltdtCreateQueueRouteQueryInfo

函数功能

创建acltdtQueueRouteQueryInfo类型的数据，表示队列路由关系查询条件。

函数原型

acltdtQueueRouteQueryInfo* acltdtCreateQueueRouteQueryInfo()

参数说明

无

返回值说明

- 返回acltdtQueueRouteQueryInfo类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.96.2 acltdtDestroyQueueRouteQueryInfo

函数功能

销毁通过acltdtCreateQueueRouteQueryInfo接口创建的acltdtQueueRouteQueryInfo类型的数据。

函数原型

aclError acltdtDestroyQueueRouteQueryInfo(const acltdtQueueRouteQueryInfo *info)

参数说明

参数名	输入/输出	说明
info	输入	待销毁的acltdtQueueRouteQueryInfo类型的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.96.3 acltdtSetQueueRouteQueryInfo

函数功能

设置路由关系查询条件。

函数原型

aclError acltdtSetQueueRouteQueryInfo(acltdtQueueRouteQueryInfo *param, acltdtQueueRouteQueryInfoParamType type, size_t len, const void *value)

参数说明

参数名	输入/输出	说明
param	输入&输出	队列路由关系查询条件的指针。 **acltdtCreateQueueRouteQueryInfo**
type	输入	参数类型。
len	输入	参数值的字节数。 ● 属性参数的类型为*_UINT64时，此处配置为8； ● 属性参数的类型为*_UINT32时，此处配置为4； ● 属性参数的类型为*_PTR时，若操作系统是32位，则此处配置为4；若操作系统是64位，则配置为8。
value	输入	参数值的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.97 aclGraphDumpOption

11.27.97.1 aclCreateGraphDumpOpt

函数功能

创建aclGraphDumpOption类型的数据，表示单算子图Dump配置信息，用于设置图Dump时的选项（当前还没有选项，预留该数据类型，便于后续扩展）。

函数原型

aclGraphDumpOption *aclCreateGraphDumpOpt()

参数说明

无

返回值说明

- 返回aclGraphDumpOption类型的指针，表示成功。
- 返回nullptr，表示失败。

11.27.97.2 aclDestroyGraphDumpOpt

函数功能

销毁通过**aclCreateGraphDumpOpt**接口创建的aclGraphDumpOption类型的数据。

函数原型

aclError aclDestroyGraphDumpOpt(const aclGraphDumpOption *graphDumpOpt)

参数说明

参数名	输入/输出	说明
graphDumpOpt	输入	待销毁的aclGraphDumpOption类型的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.98 aclmdlExecConfigHandle

11.27.98.1 aclmdlCreateExecConfigHandle

函数功能

创建aclmdlExecConfigHandle类型的数据，表示一个模型执行的配置对象。

如需销毁aclmdlExecConfigHandle类型的数据，请参见**aclmdlDestroyExecConfigHandle**。

函数原型

aclmdlExecConfigHandle *aclmdlCreateExecConfigHandle()

参数说明

无

返回值说明

- 返回aclmdlExecConfigHandle类型的指针，表示成功。
- 返回NULL，表示失败。

11.27.98.2 aclmdlDestroyExecConfigHandle

函数功能

销毁通过**aclmdlCreateExecConfigHandle**接口创建的aclmdlExecConfigHandle类型的数据。

函数原型

aclError aclmdlDestroyExecConfigHandle(const aclmdlExecConfigHandle *handle)

参数说明

参数名	输入/输出	说明
handle	输入	待销毁的aclmdlExecConfigHandle类型的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.99 aclrtStreamConfigHandle

11.27.99.1 aclrtCreateStreamConfigHandle

函数功能

创建aclrtStreamConfigHandle类型的数据，表示一个Stream的配置对象。

如需销毁aclrtStreamConfigHandle类型的数据，请参见**aclrtDestroyStreamConfigHandle**。

Atlas 200/300/500 推理产品不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

aclrtStreamConfigHandle *aclrtCreateStreamConfigHandle(void)

参数说明

无

返回值说明

- 返回aclrtStreamConfigHandle类型的指针，表示成功。
- 返回NULL，表示失败。

11.27.99.2 aclrtDestroyStreamConfigHandle

函数功能

销毁通过**aclrtCreateStreamConfigHandle**接口创建的aclrtStreamConfigHandle类型的数据。

Atlas 200/300/500 推理产品不支持该接口。

Atlas 推理系列产品，不支持该接口。

函数原型

aclError aclrtDestroyStreamConfigHandle(aclrtStreamConfigHandle *handle)

参数说明

参数名	输入/输出	说明
handle	输入	待销毁的aclrtStreamConfigHandle类型的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.100 aclrtAllocatorDesc

11.27.100.1 aclrtAllocatorCreateDesc

函数功能

创建aclrtAllocatorDesc类型的数据，表示Allocator描述信息，主要用于注册回调函数。

函数原型

aclrtAllocatorDesc aclrtAllocatorCreateDesc()

返回值说明

- 返回aclrtAllocatorDesc类型的指针，表示成功。
- 返回NULL，表示失败。

11.27.100.2 aclrtAllocatorDestroyDesc

函数功能

销毁通过aclrtAllocatorCreateDesc接口创建的aclrtAllocatorDesc类型的数据。

原型

aclError aclrtAllocatorDestroyDesc(aclrtAllocatorDesc allocatorDesc)

参数说明

参数名	输入/输出	说明
allocatorDesc	输入	待销毁的aclrtAllocatorDesc类型的数据。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.100.3 aclrtAllocatorSetObjToDesc

函数功能

使用用户提供的Allocator场景下，向Allocator描述信息中设置Allocator对象。

函数原型

aclError aclrtAllocatorSetObjToDesc(aclrtAllocatorDesc allocatorDesc, aclrtAllocator allocator)

参数说明

参数名	输入/输出	说明
allocatorDesc	输入	Allocator描述符指针。 需提前调用**aclrtAllocatorCreateDesc**接口设置Allocator描述信息。
allocator	输入	用户提供的Allocator对象指针。

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.100.4 aclrtAllocatorSetAllocFuncToDesc

函数功能

使用用户提供的Allocator场景下，设置"申请内存block"的回调函数。

函数原型

aclError aclrtAllocatorSetAllocFuncToDesc(aclrtAllocatorDesc allocatorDesc, aclrtAllocatorAllocFunc func)

参数说明

参数名	输入/输出	说明
allocatorDesc	输入	Allocator描述符指针。 需提前调用**aclrtAllocatorCreateDesc**接口设置Allocator描述信息。

func	输入	申请内存block的回调函数。
		回调函数定义如下：
		`typedef void *(*aclrtAllocatorAllocFunc)(aclrtAllocator allocator, size_t size);`
		aclrtAllocator的定义如下：
		`typedef void *aclrtAllocator;`

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.100.5 aclrtAllocatorSetAllocAdviseFuncToDesc

函数功能

使用用户提供的Allocator场景下，设置"根据建议地址申请内存block"的回调函数，一般用于内存复用场景。

函数原型

aclError aclrtAllocatorSetAllocAdviseFuncToDesc(aclrtAllocatorDesc allocatorDesc, aclrtAllocatorAllocAdviseFunc func)

参数说明

参数名	输入/输出	说明
allocatorDesc	输入	Allocator描述符指针。 需提前调用aclrtAllocatorCreateDesc接口设置Allocator描述信息。
func	输入	根据建议地址申请内存block的回调函数。 回调函数定义如下： typedef void *(*aclrtAllocatorAllocAdviseFunc)(aclrtAllocator allocator, size_t size, aclrtAllocatorAddr addr); aclrtAllocator、aclrtAllocatorAddr的定义如下： typedef void *aclrtAllocator; typedef void *aclrtAllocatorAddr;

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.100.6 aclrtAllocatorSetFreeFuncToDesc

函数功能

使用用户提供的Allocator场景下，设置"释放内存block"的回调函数。

函数原型

aclError aclrtAllocatorSetFreeFuncToDesc(aclrtAllocatorDesc allocatorDesc, aclrtAllocatorFreeFunc func)

参数说明

参数名	输入/输出	说明
allocatorDesc	输入	Allocator描述符指针。 需提前调用**aclrtAllocatorCreateDesc**接口设置Allocator描述信息。
func	输入	释放内存block的回调函数。 回调函数定义如下： typedef void (*aclrtAllocatorFreeFunc)(aclrtAllocator allocator, aclrtAllocatorBlock block); aclrtAllocator、aclrtAllocatorBlock的定义如下： typedef void *aclrtAllocator; typedef void *aclrtAllocatorBlock;

返回值说明

返回0表示成功，返回其它值表示失败。

11.27.100.7 aclrtAllocatorSetGetAddrFromBlockFuncToDesc

函数功能

使用用户提供的Allocator场景下，设置"根据申请来的block获取device内存地址"的回调函数。

函数原型

aclError aclrtAllocatorSetGetAddrFromBlockFuncToDesc(aclrtAllocatorDesc allocatorDesc, aclrtAllocatorGetAddrFromBlockFunc func)

参数说明

参数名	输入/输出	说明
allocatorDesc	输入	Allocator描述符指针。 需提前调用**aclrtAllocatorCreateDesc**接口设置Allocator描述信息。

参数名	输入/输出	说明

func	输入	根据申请来的block获取device内存地址的回调函数。
		回调函数定义如下：
		`typedef void *(*aclrtAllocatorGetAddrFromBlockFunc)(aclrtAllocatorBlock block);`
		aclrtAllocatorBlock的定义如下：
		`typedef void *aclrtAllocatorBlock;`

返回值说明

返回0表示成功，返回其它值表示失败。

12 FAQ

资源异常问题
推理问题
DVPP处理数据问题
单算子调用问题
编译运行问题

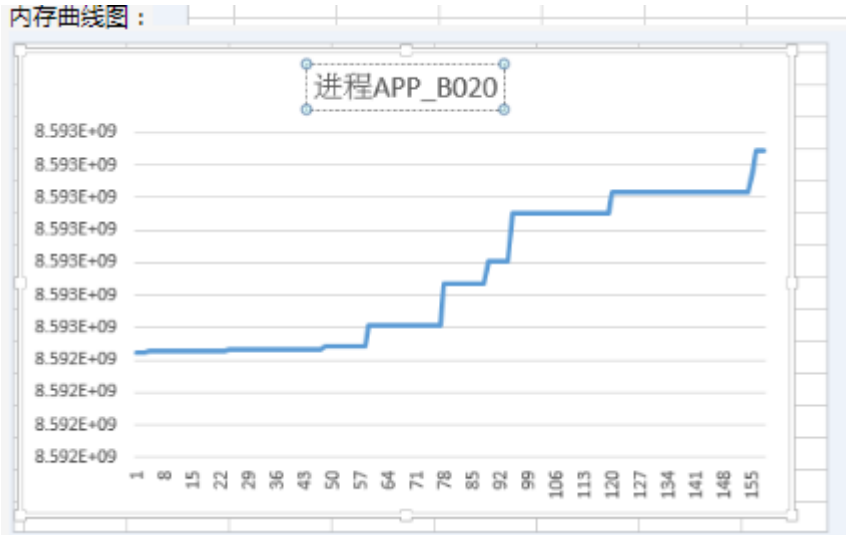
12.1 资源异常问题

12.1.1 内存未释放导致内存泄露

现象描述

测试用例长稳运行时，出现内存泄漏的现象，内存占用持续上升。如图12-1所示。

图 12-1 内存占用持续上升



可能原因

分析上述日志信息，可能存在以下故障原因：

系统存在只申请内存不释放内存的问题，正常情况下，内存申请与释放必须成对出现。

处理步骤

针对分析的故障可能原因，可以参考下面步骤处理：

排查所有的内存申请和释放的地方，保证申请与释放一一对应。例如aclrtMalloc与aclrtFree，aclrtMallocHost与aclrtFreeHost、aclrtCreateStream与aclrtDestroyStream等。

12.1.2 Event 数量超过上限导致 aclrtRecordEvent 接口返回失败

现象描述

调用aclrtRecordEvent接口在Stream中记录一个Event时，日志中的报错如下，红框中是关键日志信息，提示Event ID申请失败：

```
[ERROR] RUNTIME(18401,main):2020-05-29-17:53:11.569.381 18408 StreamSynchronize:stream Synchronize failed
[ERROR] RUNTIME(18401,main):2020-05-29-17:53:11.569.404 18408 StreamSynchronize:stream synchronize failed, error = 14
[ERROR] ASCENDCL(18401,main):2020-05-29-17:53:11.569.427 18408 acldvppDestroyChannel:acl/single_op/dvpp/channel.cpp:193: "synchronize stream failed, result =
[INFO] RUNTIME(18401,main):2020-05-29-17:53:11.569.453 18407 ReceivingRun:report[0].task_id=32772
[INFO] RUNTIME(18401,main):2020-05-29-17:53:11.569.473 18407 TryDelRecordedTask:del public task from stream, stream_id=835, tailTaskId=32772, delTaskId=32772,
[INFO] RUNTIME(18401,main):2020-05-29-17:53:11.569.491 18407 TaskFinished:device_id=0, stream_id=835, sq_id=835, task_id=32772, task_type=7,task_finish_num=4
[ERROR] RUNTIME(18401,main):2020-05-29-17:53:11.569.517 18408 StreamDestroy:stream is not in current ctx
[ERROR] RUNTIME(18401,main):2020-05-29-17:53:11.569.536 18408 StreamDestroy:destroy stream failed, error = 6
[ERROR] ASCENDCL(18401,main):2020-05-29-17:53:11.569.553 18408 DestroyNotifyAndStream:acl/single_op/dvpp/channel.cpp:41: "fail to destroy stream, ret = 6"
[INFO] ASCENDCL(18401,main):2020-05-29-17:53:11.569.586 18408 acldvppDestroyChannelDesc:acl/types/dvpp.cpp:401: "destroy DvppChannelDesc info: channelIndex =
mdListLen = 2097152."
[ERROR] DRV(18401,main):2020-05-29-17:53:11.569.787 [devdrv] [drvEventIdAlloc 647] error.
[ERROR] RUNTIME(18401,main):2020-05-29-17:53:11.569.804 18409 EventIdAlloc:drvEventIdAlloc:errorcode = 7
[INFO] RUNTIME(18401,main):2020-05-29-17:53:11.569.819 18409 EventIdAlloc:id = -1
[ERROR] RUNTIME(18401,main):2020-05-29-17:53:11.569.834 18409 Record:event id alloc error, error = 14
[WARNING] RUNTIME(18401,main):2020-05-29-17:53:11.569.849 18409 Record:fail to init record task
[ERROR] RUNTIME(18401,main):2020-05-29-17:53:11.569.865 18409 Synchronize:fail to record
[ERROR] RUNTIME(18401,main):2020-05-29-17:53:11.569.880 18409 StreamSynchronize:stream Synchronize failed
[ERROR] RUNTIME(18401,main):2020-05-29-17:53:11.569.895 18409 StreamSynchronize:stream synchronize failed, error = 14
[ERROR] ASCENDCL(18401,main):2020-05-29-17:53:11.569.911 18409 acldvppDestroyChannel:acl/single_op/dvpp/channel.cpp:193: "synchronize stream failed, result =
[ERROR] RUNTIME(18401,main):2020-05-29-17:53:11.569.927 18409 StreamDestroy:stream is not in current ctx
[ERROR] RUNTIME(18401,main):2020-05-29-17:53:11.569.942 18409 StreamDestroy:destroy stream failed, error = 6
[ERROR] ASCENDCL(18401,main):2020-05-29-17:53:11.569.958 18409 DestroyNotifyAndStream:acl/single_op/dvpp/channel.cpp:41: "fail to destroy stream, ret = 6"
[INFO] ASCENDCL(18401,main):2020-05-29-17:53:11.569.982 18409 acldvppDestroyChannelDesc:acl/types/dvpp.cpp:401: "destroy DvppChannelDesc info: channelIndex =
mdListLen = 2097152."
```

可能原因

分析上述日志信息，可能存在以下故障原因：Event ID的数量超过上限。

处理步骤

多Stream之间同步等待的场景下，Event ID的资源时可以复用的，复用Event ID的流程是：在调用aclrtRecordEvent接口+aclrtStreamWaitEvent接口后，若指定的Event已完成，则需要及时调用aclrtResetEvent接口释放Event资源。

需要用户按照复用Event ID的流程优化代码逻辑。

12.1.3 进程异常退出后重新执行任务失败

现象描述

进程异常退出时，包括强行终止任务（如ctrl + c或者kill命令终止进程）的场景，然后重新启动任务失败。

可能原因

进程异常退出时，只能依赖系统检测到程序退出后才进行资源释放，释放资源最长需要一分钟的执行时间。如果在未执行完资源释放前执行新的任务，可能导致新执行的任务失败。

处理步骤

进程异常退出后需要等待一分钟，才能保证下一次重新执行任务成功。

12.1.4 进程异常，下一次执行任务报错“unbind model stream failed”

现象描述

用户捕获异常退出信号，并在信号处理函数中释放已申请资源，下一次执行时会报执行失败。此时查看日志，会发现unbind model stream failed报错。

可能原因

进程异常时，Host侧内核态驱动会自动检测并发起对应进程Device侧资源释放的流程，不需要用户捕获进程异常的信号并主动完成清理。若用户主动释放，会影响到系统的资源释放流程。

处理步骤

用户无需关注进程异常退出信号，不要对异常退出信号做处理。

12.1.5 进程异常时资源清理的处理建议

现象描述

用户捕获异常退出信号，并在信号处理函数中释放已申请资源，下一次执行时会报执行失败。此时查看日志，会发现如图12-2所示报错。

图 12-2 unbind model stream failed

```
device-0_20200904215108557.log:482494:[ERROR] TSCH(-1,null):2020-09-04-21:51:13.452.762 63696 (cpuid:0) task_scheduler_engine.c:707 proc_model_stream_unbind: unbind model str
eam failed, stream is running. stream->model_id=512, model_id=512, task.sq_id=514, task.task_id=3
```

可能原因

进程异常时，host侧内核态驱动会自动检测并发起对应进程device侧资源释放的流程，不需要用户捕获进程异常的信号并主动完成清理。若用户主动释放，会影响到系统的资源释放流程。

处理步骤

用户无需关注进程异常退出信号。

12.1.6 用户进程异常退出后重启进程失败

现象描述

用户进程卡住或者用户强制退出进程后，再次重启，重启后发现进程无法正常启动。类似的日志信息如下：

AscendCL日志信息：aclrtProcessReport failed

```
aclrtProcessReport failed, ret = 107012
aclrtProcessReport failed, ret = 107012
```

Runtime日志信息：halResourceIdAlloc xxx failed

```
[ERROR] DRV(2086,rtstest_host):2021-06-09-02:14:46.034.368 [ascend][curpid: 2086, 2086][drv][tsdrv]
[halResourceIdAlloc 477]id is exhausted, type(0 stream), range[0, 1024), dev_id(0), tsid(0).
[ERROR] RUNTIME(2086,rtstest_host):2021-06-09-02:14:46.034.380 [npu_driver.cc:285]2086 StreamIdAlloc:
[driver interface] halResourceIdAlloc streamid failed: device_id=0, tsId=0, drvRetCode=48!
[ERROR] RUNTIME(2086,rtstest_host):2021-06-09-02:14:46.034.401 [stream.cc:448]2086 Setup:Failed to
alloc stream id, retCode=0x702001a.
[ERROR] RUNTIME(2086,rtstest_host):2021-06-09-02:14:46.034.416 [context.cc:1251]2086
StreamCreate:Setup stream failed, retCode=0x702001a.
[ERROR] RUNTIME(2086,rtstest_host):2021-06-09-02:14:46.034.440 [logger.cc:211]2086
StreamCreate:Create stream failed, priority=7 ,flags=0.
[ERROR] RUNTIME(2086,rtstest_host):2021-06-09-02:14:46.034.458 [api_c.cc:461]2086
rtStreamCreateWithFlags:ErrCode=207008, desc=[driver error:no stream resource], InnerCode=0x702001a
[ERROR] RUNTIME(2086,rtstest_host):2021-06-09-02:14:46.034.469 [error_message_manage.cc:26]2086
ReportFuncErrorReason:rtStreamCreateWithFlags execute failed, reason=[driver error:no stream resource]
```

可能原因

通过日志分析无法正常重启的原因可能是public taskid、stream id、eventid等资源申请不到引起的：

- 资源已经被其他进程占用完。
- 上一个进程退出时还未完全释放完资源。

处理步骤

针对上述可能原因，可以按以下方式处理：

- 等待一分钟后再重新启动进程，保证上一个进程资源释放完成。
- 停止其他进程或者等其他进程执行完成后再启动进程。
- 如果通过上述方式处理后仍然申请失败，建议检查是否超过了可用的资源上限，如果未超上限，则需要重启环境强行释放资源、恢复环境。

12.1.7 AI 应用进程未退出，导致休眠唤醒失败

问题现象

休眠失败。

查看应用类日志，系统内部的任务分发模块hwts正处于busy状态，检查发现不满足休眠条件，日志片段示例如下：

```
[ERROR] TSCH(-1,null):2023-01-01-02:53:45.850.781 1 (dieid:0,cpuid:0) device_management_plat.c:563
suspend_ack: suspend pre check fail, hwts is busy
[EVENT] TSCH(-1,null):2023-01-01-02:53:45.850.803 2 (dieid:0,cpuid:0) device_management.c:411
process_low_power_cmd: ts suspend ack ret=1.
```

原因分析

根据休眠唤醒的流程，休眠前AI应用进程必须先退出，相关硬件资源处于idle态，才允许休眠。不满足休眠条件，会有相关报错，本案例中因为AI应用进程未退出，在休眠唤醒时检测到hwts处于busy状态，因此休眠失败。

解决办法

用户需要确保AI应用进程已经运行结束或者优雅退出，推荐使用**kill -2** ***PID***退出相关进程，***PID***需按照替换为实际进程ID。

12.1.8 算子插件未注册报错

现象描述

查看日志， 存在报错某个算子类型不支持：

```
Check op[%s]'s type[%s] failed, it is not supported.
```

或者

进行模型转换的时候，某个算子类型转换不符合预期，被转换成了frameworkop类型。

可能原因

根据日志分析，可能存在以下可能原因：

- 算子插件so未加载成功。
- 算子未注册映射关系，或者未编译到算子的插件so中。

解决措施

针对分析可能的故障原因，可以参考下面步骤处理：

步骤1 确认算子插件so是否加载成功。

1. 算子插件so加载成功打印类似信息：

```
plugin load /usr/local/Ascend/opp/built-in/framework/onnx/libops_all_onnx_plugin.so success.
```

2. 加载失败的告警关键信息：

```
dlopen failed, plugin name:%s. Message(%s).
```

步骤2 如果算子插件so加载成功，则需要继续确认算子注册的映射关系是否编译进加载的插件so中了。

使用nm命令查看so符号表， 如果没有注册， 则需要注册该算子插件， 可以参考《**TBE&AI CPU算子开发指南（开放态）**》的“算子适配”章节内容实现。

说明

nm -D命令可查看so文件符号表。

步骤3 如果算子插件so未加载成功，参考失败告警中Message提示内容处理。

----结束

12.1.9 算子原型未注册报错

现象描述

查看日志， 存在报错某个算子没有原型定义：

```
op[%s] type[%s] have no ir factory.
```

或者

```
IR for op[%s] optype[%s] is not registered.
```

说明

op[%s] type[%s]中的%s分别表示具体的算子名称和算子类型。

可能原因

根据日志分析，可能存在以下可能原因：

- 算子原型so未加载成功。
- 算子未定义注册该类型算子， 并编译到算子的原型so中。

解决措施

针对分析可能的故障原因，可以参考下面步骤处理：

步骤1 确认算子原型so是否加载成功。

1. 算子原型so加载成功打印类似信息：

OpsProtoManager plugin load /usr/local/Ascend/opp/built-in/op_proto/libopsproto.so success.

2. 加载失败的告警关键信息：

OpsProtoManager dlopen failed, plugin name:%s. Message(%s).

步骤2 如果算子原型so加载成功， 需要确认算子原型定义是否编译进加载的so中了。

使用nm查看so符号表， 如果没有注册， 则需要注册该算子原型， 可以参考《**TBE&AI CPU算子开发指南（开放态）**》的“算子原型定义”章节内容实现。

说明

nm -D命令可查看so文件符号表。

步骤3 如果算子原型so未加载成功，参考失败告警中Message提示内容处理。

----结束

12.1.10 动态 shape 模型用户输入和模型推导结果不匹配

现象描述

场景1：模型执行报错，日志信息中包含以下关键信息：“[Check][Size] %s(%s) index[%d] mem size out of range! Expected size: %ld, but given input size: %ld.”

```
(INFO) GE(2284607, ir_build):2021-11-08-19:31:33.920.888 [task_context.cc:257]2284607 AllocateOutput:
To allocate output for node: add2. index = 0, tensor desc = [TensorDescl DataType = 6 ， Format = 2 ·
Shape = [50, ]
[ERROR] GE(2284607, ir_build):2021-11-08-19:31:33.920.900 [task_context.cc:257]2284607 AllocateOutput:
ErrorNo: 50331649() [LOAD][LOAD][Check][Size] add2(Add) index[0] mem size out of range! Expected
size: 128, but given input size: 2.
```

场景2：单算子场景执行报错，日志信息中包含以下关键信息：“[Check][Param:outputs]Output size mismatch. index = %zu, model expect %zu, but given %zu(after align)”

```
(INFO) GE(130191,python):2021-09-27-17:23:17.922.047 [single op.cc:161]131511 ValidateArgs:lnput [1],
aligned_size:96, inputs.length:40, input_sizes_:96
(INFO) GE(130191,python):2021-09-27-17:23:17.922.054 [single op.cc:186]131511 ValidateArgs:Output [0],
aligned_size:3145760, outputs.length:3145728, input_sizes_:18446744073709551615
[ERROR] GE(130191,python):2021-09-27-17:23:17,922.095 [single op.cc:161]131511 ValidateArgs: ErrorNo:
145000(Parameter invalid.) [FINALI][FlNALl][CheCkl][Param.Outputs]Output Size mismatch. index = 0,
model expect 18446744073709551615, but given ....
[ERROR] ASCENDCL(130191,python):2021-09-27-17:23:17,922.213 [op_executor.cpp:68]131511
DoExecteAsync: [FINALI][FlNALl][Exec][Op]Execte op failed. ge result = 145000
```

可能原因

场景1：

此处为模型执行时，校验用户分配的output_buffer大小和模型经过一层层infershape后得到的模型输出的output_buffer大小不匹配。

场景2：

此处为单算子场景执行时，校验用户分配的output_buffer大小和单算子编译时infershape推出的output_size大小不匹配。

解决措施

针对分析的故障可能原因，可以参考下面步骤处理：

针对场景1：

若发生该错误，用户需要检查分配的output_buffer大小是否正确，应该与模型推导的大小保持一致。建议用户分配的output_buffer按照上报的ERROR中提示的大小进行分配。

针对场景2：

若发生该错误，用户需检查分配的output_buffer大小和算子infershape提示的size大小是否一致，建议按照报错提示的大小进行分配。

12.1.11 动态 shape 模型输入大小校验失败

现象描述

动态shape模型输入大小校验失败，日志信息中包含以下关键信息：tensor size mismatches. expected: ..., but given ...

```
plog-187626_20211011143937175.log:17480:[ERROR] GE(187626,python3):2021-10-11-14:39:37:270.942
[execution_engine.cc:450]191154 ValidInputTensor:ErrorNo: 1343225860(Internal errors) [EXEC][EXEC]
[Check][Size] for [PartitionedCall_14(PartitionedCall)] Input[40]:tensor size mismatches.expected:
768032, but given 32.
```

可能原因

动态shape场景每个node都会在执行时对shape进行推导，该故障现象可能为算子的shape校验不合法。

解决措施

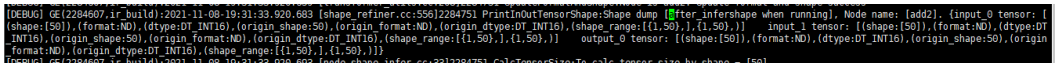
针对分析的故障可能原因，应根据图上的连接关系，对shape的来源进行排查，找出不合理的shape推导：

可搜索执行plog日志查看关键词：

"before_infershape when running"：显示算子shape推导前的输入、输出shape等信息。

“after_infershape when running ”：显示算子shape推导后的输入、输出shape等信息。

从报错节点的输入、输出shape开始进行排查， 检查当前节点shape推导结果是否正确（即判定根据输入shape推出的输出shape是否符合预期）， 如果是输入shape存在问题，则按照相同方法继续排查输入节点的shape推导。



12.1.12 AI Core 算子执行报错

现象描述

Runtime执行报错，在plog日志中Runtime打印了类似fault kernel_name和func_name的关键信息。

plog日志在{install_path}/ascend/log/debug/plog路径下，日志格式为plog-pid_yyymmddhhmmss.log。

```
[ERROR] RUNTIME(4150867,msame):2022-09-22-09:27:46.403.262 [engine.cc:1103]4150867 ReportExceptProc:[EXEC][DEFAULT]Task exception! device_id=0, stream_id=20, task_id=1, type=13, retCode=0x91, [the model stream execute failed].
[ERROR] RUNTIME(4150867,msame):2022-09-22-09:27:46.404.423 [device_error_proc.cc:495]4150867 PrintCoreErrorInfo:[EXEC][DEFAULT]The error from device(0), serial number is 193, there is an aicore error, core id is 8, error code = 0x800000, dump info: pc start: 0x800120080047000, current: 0x1200800471cc, vec error info: 0x7cafc4e, mte error info: 0x3000052, ifu error info: 0xc33f87bd7a80, ccu error info: 0xffd2bbd5005fe9d7, cube error info: 0x84, biu error info: 0, aic error mask: 0x65000200d000288, para base: 0x120080016300, errorStr: The DDR address of the MTE instruction is out of range.
[ERROR] RUNTIME(4150867,msame):2022-09-22-09:27:46.404.443 [device_error_proc.cc:526]4150867 PrintCoreErrorInfo:[EXEC][DEFAULT]report error module_type=5, module_name=EZ9999
[ERROR] RUNTIME(4150867,msame):2022-09-22-09:27:46.404.449 [device_error_proc.cc:526]4150867 PrintCoreErrorInfo:[EXEC][DEFAULT]The extend info from device(0), serial number is 193, there is aicore error, core id is 8, aicore int: 0x10, aicore error2: 0, axi clamp ctrl: 0, axi clamp state: 0x1717, biu status0: 0x101d14000000000, biu status1: 0x80000201020000, clk gate mask: 0, dbg address: 0, ecc en: 0, mte ccu ecc 1bit error: 0x2e80000000000000, vector cube ecc 1bit error: 0, run stall: 0x1, dbg data0: 0, dbg data1: 0, dbg data2: 0, dbg data3: 0, dfx data: 0x8b
[ERROR] RUNTIME(4150867,msame):2022-09-22-09:27:46.404.607 [task.cc:1021]4150867 PrintErrorInfo:[EXEC][DEFAULT]Aicore kernel execute failed, device_id=0, stream_id=23, report_stream_id=20, task_id=24, flip_num=0, fault kernel_name=16805736118314619649-1_0_1_Add_35, func_name=te_add_729e2a87c649f49de98ac1a6fd491b3262ee7db9c1c2d6f4add7d7439aa3d22e_1__kernel0, program id=22, hash=3338199064661472585.
[ERROR] RUNTIME(4150867,msame):2022-09-22-09:27:46.404.618 [task.cc:3275]4150867 ReportErrorInfo:[EXEC][DEFAULT]model execute error, retCode=0x91, [the model stream execute failed].
[ERROR] RUNTIME(4150867,msame):2022-09-22-09:27:46.404.624 [task.cc:3247]4150867 PrintErrorInfo:[EXEC][DEFAULT]model execute task failed, device_id=0, model stream_id=20, model task_id=1, flip_num=0, model_id=3, first_task_id=65535
```

```
[ERROR] RUNTIME(4150867,msame):2022-09-22-09:27:46.404.714 [stream.cc:929]4150867 GetError:[EXEC]
[DEFAULT]Stream Synchronize failed, stream_id=20, retCode=0x91, [the model stream execute failed].
[ERROR] RUNTIME(4150867,msame):2022-09-22-09:27:46.404.742 [model.cc:581]4150867
SynchronizeExecute:[EXEC][DEFAULT]Fail to synchronize forbbiden stream_id=20, retCode=0x7150050!
[ERROR] RUNTIME(4150867,msame):2022-09-22-09:27:46.404.748 [model.cc:605]4150867
GetStreamToSyncExecute:[EXEC][DEFAULT]report error module_type=0, module_name=EE9999
[ERROR] RUNTIME(4150867,msame):2022-09-22-09:27:46.404.753 [model.cc:605]4150867
GetStreamToSyncExecute:[EXEC][DEFAULT]Model synchronize execute failed, model_id=3!
[ERROR] RUNTIME(4150867,msame):2022-09-22-09:27:46.404.774 [logger.cc:856]4150867 ModelExecute:
[EXEC][DEFAULT]Execute model failed.
[ERROR] RUNTIME(4150867,msame):2022-09-22-09:27:46.404.787 [api_c.cc:2063]4150867 rtModelExecute:
[EXEC][DEFAULT]ErrCode=507011, desc=[the model stream execute failed], InnerCode=0x7150050
[ERROR] RUNTIME(4150867,msame):2022-09-22-09:27:46.404.793 [error_message_manage.cc:49]4150867
FuncErrorReason:[EXEC][DEFAULT]report error module_type=3, module_name=EE8888
[ERROR] RUNTIME(4150867,msame):2022-09-22-09:27:46.404.801 [error_message_manage.cc:49]4150867
FuncErrorReason:[EXEC][DEFAULT]rtModelExecute execute failed, reason=[the model stream execute failed]
```

可能原因

从日志报错可知，AI Core 算子执行失败，可能算子本身代码问题：数据输入不匹配、访问越界、计算溢出等异常。

查阅plog日志，根据fault kernel_name和func_name可获取报错算子名称和报错函数名称。

plog日志在/root/ascend/log/debug/plog路径下，日志格式为plog-pid_yyymmddhhmmss.log。

```
[ERROR] RUNTIME(4150867,msame):2022-09-22-09:27:46.404.607 [task.cc:1021]4150867 PrintErrorInfo:
[EXEC][DEFAULT]Aicore kernel execute failed, device_id=0, stream_id=23, report_stream_id=20, task_id=24,
flip_num=0, fault kernel_name=16805736118314619649-1_0_1_Add_35,
func_name=te_add_729e2a87c649f49de98ac1a6fd491b3262ee7db9c1c2d6f4add7d7439aa3d22e_1__kernel0,
 program id=22, hash=3338199064661472585.
```

处理步骤

该类型错误，需要联系技术支持定位排查。 您可以获取日志后单击**Link**联系技术支持。

可能导致的故障

模型下沉场景下，该问题可能导致AscendCL报错Execute model failed，并打印在plog日志中。

```
[ERROR] ASCENDCL(4150867,msame):2022-09-22-09:27:46.404.834 [model.cpp:699]4150867
ModelExecute: [EXEC][DEFAULT][Exec][Model]Execute model failed, ge result[507011], modelId[1]
[ERROR] ASCENDCL(4150867,msame):2022-09-22-09:27:46.404.857 [model.cpp:1547]4150867
aclmdlExecute: [EXEC][DEFAULT][Exec][Model]modelId[1] execute failed, result[507011]
```

非模型下沉场景下，该问题可能导致算子执行失败AscendCL报错get op desc faild，Runtime报错Aicore kernel execute failed，并打印在plog日志中。

```
[ERROR] RUNTIME(2856615,xaclfk):2022-09-15-11:36:47.817.465 [task.cc:1058]2856939 PreCheckTaskErr:
[EXEC][DEFAULT]Kernel task happen error, retCode=0x26, [aicore exception].
[ERROR] RUNTIME(2856615,xaclfk):2022-09-15-11:36:47.817.538 [task.cc:1029]2856939 PrintErrorInfo:
[EXEC][DEFAULT]Aicore kernel execute failed, device_id=0, stream_id=0, report_stream_id=0, task_id=615,
flip_num=0, fault kernel_name=12646006_1663210912148832_-1_0_while/transformer_0/decoder/
layer_0/rnn/rnn/while/Select,
func_name=te_select_7b314df6791292127cb82df985d04ddaf6d069cb31aaccec00e0b8ee2e997f20_1__kernel
0, program id=131, hash=14736095126365135477.
[ERROR] GE(2856615,xaclfk):2022-09-15-11:36:47.818.283 [graph_execute.cc:557]2856939 GetOpDescInfo:
ErrorNo: 4294967295(failed) [EXEC][DEFAULT][Get][OpDescInfo] failed, device_id:0, stream_id:0,
task_id:615.
[ERROR] GE(2856615,xaclfk):2022-09-15-11:36:47.818.308 [ge_executor.cc:1332]2856939 GetOpDescInfo:
```

```
ErrorNo: 4294967295(failed) [EXEC][DEFAULT][Get][OpDescInfo] failed, device_id:0, stream_id:0,
task_id:615.
[ERROR] ASCENDCL(2856615,xaclfk):2022-09-15-11:36:47.818.315 [model.cpp:2216]2856939
aclmdlCreateAndGetOpDesc: [EXEC][DEFAULT][Get][OpDescInfo]get op desc faild, ge result[-1],
deviceId[0], streamId[0], taskId[615]
```

12.1.13 AI CPU 算子执行报错

现象描述

Runtime执行报错，在plog日志中Runtime打印了PrintAicpuErrorInfo的错误信息。

plog日志在{install_path}/ascend/log/debug/plog路径下，日志格式为plog-pid_yyymmddhhmmss.log。

```
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.791.865 [engine.cc:1103]16282
ReportExceptProc:Task exception! device_id=0, stream_id=7, task_id=2, type=1, retCode=0x2a, [aicpu
exception].
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.793.489 [device_error_proc.cc:669]16282
ProcessAicpuErrorInfo:report error module_type=0, module_name=E39999
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.793.498 [device_error_proc.cc:669]16282
ProcessAicpuErrorInfo:An exception occurred during AICPU execution, stream_id:7, task_id:2, errcode:5,
msg:aicpu execute failed.
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.793.932 [task.cc:1050]16282
PreCheckTaskErr:report error module_type=5, module_name=EZ9999
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.793.941 [task.cc:1050]16282
PreCheckTaskErr:Kernel task happen error, retCode=0x2a, [aicpu exception].
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.793.981 [task.cc:759]16282
PrintAicpuErrorInfo:report error module_type=0, module_name=E39999
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.793.990 [task.cc:759]16282
PrintAicpuErrorInfo:Aicpu kernel execute failed, device_id=0, stream_id=7, task_id=2.
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.794.116 [task.cc:777]16282
PrintAicpuErrorInfo:Aicpu kernel execute failed, device_id=0, stream_id=7, task_id=2, flip_num=0, fault
so_name=, fault kernel_name=, fault op_name=Unique, extend_info=(info_type:4, info_len:6,
msg_info:Unique).
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.794.384 [stream.cc:929]16243 GetError:[EXEC]
[DEFAULT]Stream Synchronize failed, stream_id=7, retCode=0x2a, [aicpu exception].
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.794.407 [stream.cc:932]16243 GetError:[EXEC]
[DEFAULT]report error module_type=0, module_name=E39999
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.794.419 [stream.cc:932]16243 GetError:[EXEC]
[DEFAULT]Aicpu kernel execute failed, device_id=0, stream_id=7, task_id=2, flip_num=0, fault so_name=,
fault kernel_name=, fault op_name=Unique, extend_info=(info_type:4, info_len:6, msg_info:Unique)
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.794.482 [logger.cc:305]16243 StreamSynchronize:
[EXEC][DEFAULT]Stream synchronize failed, stream = 0x5643fe3e28d0
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.794.510 [api_c.cc:661]16243 rtStreamSynchronize:
[EXEC][DEFAULT]ErrCode=507018, desc=[aicpu exception], InnerCode=0x715002a
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.794.519 [error_message_manage.cc:49]16243
FuncErrorReason:[EXEC][DEFAULT]report error module_type=3, module_name=EE8888
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.794.532 [error_message_manage.cc:49]16243
FuncErrorReason:[EXEC][DEFAULT]rtStreamSynchronize execute failed, reason=[aicpu exception]
```

可能原因

从日志报错可知，AI CPU算子执行失败，可能算子本身代码问题：数据输入不匹配、访问越界、AI CPU线程挂死等问题。

比如通过查阅AI CPU的device日志，是数据输入不匹配导致。

device日志在/root/ascend/log/debug/device-x/路径下，日志格式为device-pid_yyymmddhhmmss.log。

```
[ERROR] CCECPU(2309,aicpu_scheduler):2022-09-22-11:27:00.733.218 [aicpu_tf_kernel.cc:348][tid:2317]
[TFAdapter] AICPUKernelAndDevice::Run failure, kernel_id=0, op_name=Unique, op_type=UniqueExt,
error=Invalid argument: unique expects a 1D vector.
[ERROR] CCECPU(2309,aicpu_scheduler):2022-09-22-11:27:00.733.242 [tf_adpt_session_mgr.cc:74][tid:2317]
```

```
[TFAdapter] [sessionID:0] Failed to Run kernel, kernel_id=0.
[ERROR] CCECPU(2309,aicpu_scheduler):2022-09-22-11:27:00.733.261 [tf_adpt_session_mgr.cc:434]
[tid:2317][TFAdapter] [sessionID:0] Run kernel on session failed.
[ERROR] CCECPU(2309,aicpu_scheduler):2022-09-22-11:27:00.733.277 [tf_adpt_api.cc:85][tid:2317]
[TFAdapter] [sessionID:0] Invoke TFOperateAPI failed.
[ERROR] CCECPU(2309,aicpu_scheduler):2022-09-22-11:27:00.733.296 [ae_kernel_lib_fwk.cc:229]
[TransformKernelErrorCode][tid:2317][AICPU_PROCESSER] Call tf api return failed:5, input param to tf
api:0x124040017004
[ERROR] CCECPU(2309,aicpu_scheduler):2022-09-22-11:27:00.733.366 [aicpusd_event_process.cpp:1325]
[ExecuteTsKernelTask][tid:2317] Aicpu engine process failed, result[5].
```

处理步骤

该类型错误，需要联系技术支持定位排查。 您可以获取日志后单击**Link**联系技术支持。

12.1.14 调用 Device 失败

现象描述

调用Device失败（容器内），日志中报以下类似信息：

日志信息（1）：日志中有打印ERROR信息“drvDeviceOpen”失败字样，如下所示。

```
[ascend][ERROR][2021-06-26-17:53:15:689064][curpid: 32707, 32707][devdrv][drvDeviceOpen 79]get phys
failed, devId(0), phy_devId(0)
```

日志信息（2）：查看内核日志(dmesg)中有打印“device is in used”字样。如下所示。

```
[52783.010878] [ascend] [ERROR] [devdrv] [devdrv_manager_container_table_devlist_add_ns 832]
<drv_hlt_dsmi_te:5615> device is in used
[52783.010881] [ascend] [ERROR] [tsdrv] [devdrv_open 194] <drv_hlt_dsmi_te:5615> add to list failed.
dev_id(0)
[52783.013546] [ascend] [devdrv] [devdrv_manager_container_get_devnum 1306] <drv_hlt_dsmi_te:5615>
weird device number, dev_num = 0
[52783.013547] [ascend] [ERROR] [devdrv] [devdrv_manager_container_get_devlist_ns 1210]
<drv_hlt_dsmi_te:5615> some devices are used by other docker, mnt_ns = 0xffff8d33a5963720
[52783.013548] [ascend] [ERROR] [devdrv] [devdrv_manager_container_get_davinci_devlist 1248]
<drv_hlt_dsmi_te:5615> get davinci devlist failed, ret(-22).
```

可能原因

针对调用Device失败，可能原因如下：

- 日志信息（1）：Docker容器启动时未映射设备。
- 日志信息（2）：Device被其他容器占用。

处理步骤

- 针对日志信息（1）可能原因，可以参考以下方法处理：

  在容器内使用 **ls /dev** 指令查看是否映射Device。

  ```
  [root@Euler /]# ls /dev
  console  core  fd  full  mqueue  null  ptmx  pts  random  shm  stderr  stdin  stdout  tty  urandom  zero
  [root@Euler /]#
  ```

  如果查看到Docker容器内未映射设备，请参考《**CANN 软件安装指南**》资料启动容器，起Docker的命令中必须加挂载设备的参数，例如：

  --device=/dev/davinciX --device=/dev/davinci_manager --device=/dev/devmm_svm --device=/dev/hisi_hdc

- 针对日志信息（2）可能原因，需要等使用该Device的容器被释放后再使用。
  可以在容器内执行**ls /dev**命令查看当前容器内分配到的DavinciX，使用同样的方法查看其他容器分配到的DavinciX，然后可以通过手工停止Docker容器（例如：docker stop 容器id）的方式提前释放Device。

12.1.15 内存申请失败，出现 OOM 情况

现象描述

内存申请失败，Host侧日志提示EL9999返回码，有如下打印信息：

```
[ERROR] DRV(2936187,python3):2022-04-21-14:19:39.429.481 [ascend][curpid: 2936187, 2969960][drv]
[devmm][devmm_alloc_managed 182]<errno:12, 6> Heap_alloc_managed out of memory. (temp_ptr=0x1;
bytesize=8592031776)
[ERROR] RUNTIME(2936187,python3):2022-04-21-14:19:39.429.491 [npu_driver.cc:780]2969960
DevMemAllocHugePageManaged:report error module_type=1, module_name=EL9999
[ERROR] RUNTIME(2936187,python3):2022-04-21-14:19:39.429.495 [npu_driver.cc:780]2969960
DevMemAllocHugePageManaged:[driver interface] halMemAlloc failed: device_id=1, size=8592031776,
type=0, env_type=3, drvRetCode=6!
```

可能原因

根据日志信息分析，判断为内存申请失败。可能原因：

1. 网络并行运行，导致内存不足。
2. 网络运行需要内存过大，导致内存申请失败。

处理步骤

步骤1 查看运行网络时是否存在并行情况。

步骤2 查询网络运行需要内存大小或者减少batchsize，查看网络是否可以正常运行。

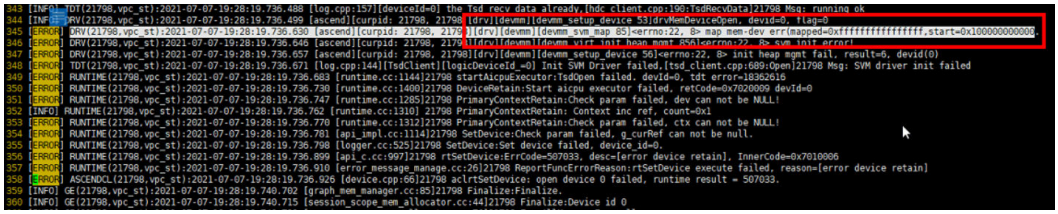
----结束

12.1.16 虚拟地址抢占导致 mmap 失败

现象描述

mmap申请失败，Host侧日志显示drvMemDeviceOpen失败，出现如图12-3日志信息。

图 12-3 drvMemDeviceOpen 失败



可能原因

根据日志信息，判断为mmap地址失败。可能原因：

（1）用户程序编译选项中启动了地址消毒（-lasan）：asan地址消毒会预留所有的虚拟地址空间，而svm在mmap时也需要预留8T的地址空间，二者冲突，所以mmap时会失败。

（2）用户预留地址与SVM模块预留地址相同。

处理步骤

步骤1 关闭编译选项即可。

步骤2 查看用户预留地址是否与SVM模块预留地址重叠，需要修改用户预留地址空间。

----结束

12.2 推理问题

12.2.1 使用 dump 功能未获取 dump 结果

现象描述

日志显示正确执行了Dump功能，但在Dump结果路径下没有Dump的结果。日志信息包含了以下关键字：

```
[INFO] ASCENDCL ****** "HandleDumpConfig end in HandleDumpConfig."
[INFO] ASCENDCL ****** "set HandleDumpConfig success in aclInit"
```

可能原因

分析上述日志信息，可能存在以下故障原因：Dump配置的模型名与实际的模型名不匹配。

处理步骤

针对分析的故障可能原因，可以参考下面步骤处理：

检查Dump配置文件acl.json，确保Dump配置文件合法，例如model_name是否配置正确。示例如下：

```
{
"dump":{
"dump_list":[
{
"model_name":"ResNet-50",
"layer":[
"convlconvl_relu"
]
},
{
```

```
"model_name":"mxnet-model"
}
],
"dump_mode":"output",
"dump_path":"/home/test/output/dump"
}
}
```

通过ATC命令生成模型的json文件，在json文件中查找“"name"”字段对应值，查找模型名称和算子名称，模型名称在"graph"字段外、算子名称在"graph"字段内。

12.2.2 动态 shape 推理申请内存失败

现象描述

模型推理过程中，申请了大小为0的内存，日志报错信息中包含以下关键信息：

```
[INFO] ASCENDCL ****** start to execute aclrtMalloc, size = 0
[ERROR] ASCENDCL ****** malloc size must be greater than zero
```

可能原因

模型为动态shape模型，模型的输出shape中含有-1，所以直接调用aclmdlGetOutputSizeByIndex接口取到的size为0。

然后申请了大小为0的内存，导致失败。

处理步骤

请参见**7.7 模型动态Shape输入推理**章节内容处理。

在aclmdlGetOutputSizeByIndex取到size为0时，用户需要预估一块较大的内存。

12.2.3 异步拷贝调用查询接口报错

现象描述

通过event实现H2D或D2H异步拷贝任务的同步等待时，在调用aclrtQueryEventStatus确认任务完成后，先调用aclrtFreeHost释放Host内存再调用aclrtDestroyEvent接口，可能会有如下报错信息打印：

```
[EVENT] DRV(78295,python):2023-01-10-11:21:48.757.930 [ascend][curpid: 78295, 78295][drv][common][share_log_read 544][ascend] [ERROR] [devmm] <python:3960,3960> Set free error. (ref_lock=1; ref_free=0; ref_count=3)
[ascend] [ERROR] [devmm] <python:3960,3960> Oper address failed. (va=0x120043200000; ref_flag=0x108; ref_lock=0; ref_free:0; ref_count=3; convert=1; async=0))
[ascend] [ERROR] [devmm] <python:3960,3960> Vaddress can not oper. (cmd=0x42204d04; cmd_id=0x4; ret=-22)
[ERROR] DRV(78295,python):2023-01-10-11:21:48.757.946 [ascend][curpid: 78295, 78295][drv][devmm][devmm_ioctl_free_pages 138]<errno:26, 17> Ioctl device error. (ret=17)
[ERROR] DRV(78295,python):2023-01-10-11:21:48.757.951 [ascend][curpid: 78295, 78295][drv][devmm][devmm_virt_heap_free_pages 294]<errno:26, 17> Devmm_ioctl_free failed. (ptr=0x120043200000; heap_type=4025417729)
[ERROR] DRV(78295,python):2023-01-10-11:21:48.757.956 [ascend][curpid: 78295, 78295][drv][devmm][devmm_free_phymem_heap_oper 323]<errno:26, 17> Heap ops failed. (ret=17; va=0x120043200000)
[ERROR] DRV(78295,python):2023-01-10-11:21:48.757.962 [ascend][curpid: 78295, 78295][drv][devmm][devmm_free_nocache_mem_process 992]<errno:26, 17> Free error. (va=0x120043200000; size=123731968; total=123731968; ret=17)
[ERROR] DRV(78295,python):2023-01-10-11:21:48.757.970 [ascend][curpid: 78295, 78295][drv][devmm][devmm_free_to_normal_heap 704]<errno:26, 17> Virt_heap_free_mem failed. (ret=17; va=0x120043200000)
[ERROR] RUNTIME(78295,python):2023-01-10-11:21:48.758.013 [npu_driver.cc:1531]78295 HostMemFree:[FINAL][FINAL]report error module_type=1, module_name=EL9999
[ERROR] RUNTIME(78295,python):2023-01-10-11:21:48.758.021 [npu_driver.cc:1531]78295 HostMemFree:[FINAL][FINAL][drv api] halMemFree failed: drvRetCode=17!
[ERROR] RUNTIME(78295,python):2023-01-10-11:21:48.758.073 [logger.cc:388]78295 HostFree:[FINAL][FINAL]free host memory failed.
[ERROR] RUNTIME(78295,python):2023-01-10-11:21:48.758.097 [api_c.cc:1077]78295 rtFreeHost:[FINAL][FINAL]ErrCode=507899, desc=[driver error:internal error], InnerCode=0x7020022
[ERROR] RUNTIME(78295,python):2023-01-10-11:21:48.758.102 [error_message_manage.cc:49]78295 FuncErrorReason:[FINAL][FINAL]report error module_type=3, module_name=EE8888
[ERROR] RUNTIME(78295,python):2023-01-10-11:21:48.758.110 [error_message_manage.cc:49]78295 FuncErrorReason:[FINAL][FINAL]rtFreeHost execute failed, reason=[driver error:internal error]
[ERROR] ASCENDCL(78295,python):2023-01-10-11:21:48.758.133 [memory.cpp:242]78295 aclrtFreeHost: [FINAL][FINAL]free host memory failed, runtime result = 507899
```

可能原因

报错是因为使用了异步拷贝任务之后下发了一个event record任务，期望使用aclrtQueryEventStatus查询到event record任务是否完成，从而判断异步拷贝任务是否完成，而后释放内存调用aclrtFreeHost。

实际上aclrtQueryEventStatus查询到的是Device执行完任务，并未透传到Host侧，所以此时释放内存，未先销毁Event会有时序问题导致报错。

处理步骤

处理该问题可以参考以下方案：

方案一：使用aclrtSynchronizeStream接口判断任务是否执行完成。

方案二：使用aclrtQueryEventStatus接口时，先调用aclrtDestroyEvent接口，再调用aclrtFreeHost接口，保证无时序问题。

12.2.4 注册算子数超过最大规格

现象描述

推理过程中，用户load model出现报错。在plog日志中Runtime打印了类似ProgramRegister:Program register failed, program out of xxx和Register binary failed的关键信息。

plog日志在{install_path}/ascend/log/debug/plog路径下，日志格式为plog-pid_yyymmddhhmmss.log。

```
[ERROR] RUNTIME(3093,rtstest_host):2021-06-09-02:30:34.400.124 [runtime.cc:967]3093
ProgramRegister:Program register failed, program out of 40000000
[ERROR] RUNTIME(3093,rtstest_host):2021-06-09-02:30:34.400.155 [logger.cc:23]3093
DevBinaryRegister:Register binary failed.
[ERROR] RUNTIME(3093,rtstest_host):2021-06-09-02:30:34.400.182 [api_c.cc:127]3093
rtDevBinaryRegister:ErrCode=507032, desc=[program register num out of use], InnerCode=0x7090007
[ERROR] RUNTIME(3093,rtstest_host):2021-06-09-02:30:34.400.185 [error_message_manage.cc:26]3093
ReportFuncErrorReason:rtDevBinaryRegister execute failed, reason=[program register num out of use]
```

可能原因

通过日志分析报错的原因可能是一个进程内算子等资源注册超过最大规格：

- 模型太大，一个进程内的注册总算子数超过最大规格：Online模式规格为4000万，Offline模式规格为200万。

处理步骤

针对上述可能原因，可以按以下方式处理：

- 分析model，简化模型或者降低动态batch档次。
- 算子数是进程资源，model太大的情况下建议一个进程open一个device。
- 避免同一算子在不同模型中反复注册。
- 注册算子数不超过最大规格。

12.3 DVPP 处理数据问题

12.3.1 DVPP 驱动引擎异常返回码

现象描述

视频解码失败，Device侧日志提示-512返回码：[dvpp_ioct1_vpc 845] call proc failed:-512, engine_id:1，如图12-4所示。

图 12-4 DVPP 引擎返回-512 异常码

```
[ERROR] KERNEL(1200,sklogd):2019-11-29-21:42:07.421.714 [8078.067669]  [dvpp] [dvpp_print_vpc_job 156] outctl out2_select:0
[ERROR] KERNEL(1200,sklogd):2019-11-29-21:42:07.421.740 [8078.067672]  [dvpp] [dvpp_print_vpc_job 184] OUT
[ERROR] KERNEL(1200,sklogd):2019-11-29-21:42:07.421.764 [8078.067673]  [dvpp] [dvpp_print_vpc_job 185] out use_flag:1
[ERROR] KERNEL(1200,sklogd):2019-11-29-21:42:07.421.789 [8078.067675]  [dvpp] [dvpp_print_vpc_job 186] wr0 addr_frame_start:ffff35a00000
[ERROR] KERNEL(1200,sklogd):2019-11-29-21:42:07.421.823 [8078.067677]  [dvpp] [dvpp_print_vpc_job 187] wr1 addr_frame_start:ffff35aca800
[ERROR] KERNEL(1200,sklogd):2019-11-29-21:42:07.421.853 [8078.067679]  [dvpp] [dvpp_print_vpc_job 188] wr2 addr_frame_start:0
[ERROR] KERNEL(1200,sklogd):2019-11-29-21:42:07.421.880 [8078.067680]  [dvpp] [dvpp_print_vpc_job 189] wr3 addr_frame_start:0
[ERROR] KERNEL(1200,sklogd):2019-11-29-21:42:07.421.905 [8078.068693]  [dvpp] [dvpp_ioctl_vpc 845] call proc failed:-512, engine_id:1
```

可能原因

根据日志信息，判断为VPC引擎返回的异常码，再根据-512返回码可以判断为等待视频解码异常中断。可能原因：用户态进程退出，导致等待解码时被异常中断。

处理步骤

查看Host侧的日志文件，检查用户态进程异常退出原因。

其他异常码参考

除样例VPC引擎返回的-512异常码外，DVPP驱动引擎还有其他引擎及各自的返回码，如表12-1所示。

您可以参考上述样例的方法，根据自身业务类型及Device侧日志文件中记录的异常返回码信息，先判断返回异常码的引擎类型，再根据返回码数值获取对应的返回码说明，初步判断故障原因。

表 12-1 DVPP 驱动引擎异常返回码

引擎类型	驱动IOCTL返回码	返回码说明	常见错误日志参考
JPEGE	-22	参数错误或者不支持	常见错误打印： 1.返回值-35：日志“JPEGE time wait, engine_id:xx”编码超时(执行器输入编码参数不对等导致) 2.返回值-512：日志“Signal interrupted, engine_id:xx”编码错误(编码被异常中断)
	-11	引擎处理异常，需要重试	
	-14	拷贝用户态数据时地址异常或者释放引擎失败	
	-16	申请引擎失败	
	-35	编码超时	
	-5	编码失败且复位成功	

引擎类型	驱动IOCTL返回码	返回码说明	常见错误日志参考
	5	编码失败且复位失败	
	-512	等待编码时被异常中断	
PNGD	-22	参数错误或者不支持	常见错误打印： 1.返回值5/-5：日志1“Time out: wait failed xx”解码超时(图片格式不正确等码流异常等导致) 1.返回值5/-5：日志2“Signal interrupted:xx, engine_id:xx”解码时被异常中断
	-14	拷贝用户态数据时地址异常或者释放引擎失败	
	-16	申请引擎失败	
	-5	解码失败且复位成功	
	5	解码失败且复位失败	
JPEGD	-22	参数错误或者不支持或者解码成功但是重置寄存器失败	常见错误打印： 1.返回值-1/-23：日志“Decode abnormal, abnormal status:xx decode error! engine id:xx”硬件上报异常中断(图片格式不正确等码流异常导致) 2.返回值-2/-24：日志“Decode abnormal, abnormal status:xx decode over time!, engine id:xx”解码超时(总线繁忙，系统缺页等导致)
	-14	拷贝用户态数据时地址异常或者释放引擎失败	
	-16	申请引擎失败	
	-1	解码失败且复位成功	
	-2/-3	解码超时且复位成功	
	-23	解码失败并且复位失败	
	-24/-25	解码超时并且复位失败	

引擎类型	驱动IOCTL返回码	返回码说明	常见错误日志参考
VPC	-22	参数错误或者不支持	常见错误打印： 1. 返回值-35：日志“dvpp_vpc_engine_proc pipe wait time out: engine id:xx”解码超时(总线繁忙、异常缺页等导致)。 2. 返回值-32：日志“Hardware process failed, engine id:xx”解码错误(码流异常等导致)。
	-11	引擎处理异常，需要重试	
	-14	拷贝用户态数据时地址异常或者释放引擎失败	
	-16	申请引擎失败	
	-35	解码超时	
	-512	等待解码时被异常中断	
	-32	硬件解码失败驱动捕捉到异常	
VPC_cmdlist	-22	参数错误或者不支持	常见错误打印： 1. 返回值-35：日志“dvpp_vpc_engine_proc pipe wait time out: engine id:xx”解码超时(总线繁忙、异常缺页等导致)。 2. 返回值-32：日志“Hardware process failed, engine id:xx”解码错误(码流异常等导致)。
	-11	引擎处理异常，需要重试	
	-14	拷贝用户态数据时地址异常或者释放引擎失败	
	-16	申请引擎失败	
	-35	解码超时	
	-512	等待解码时被异常中断	
	-32	硬件解码失败	
VENC	-22	参数错误或者不支持	常见错误打印： 1. 返回值-35：日志“wait timeout, Ret value is xx”未等到完成中断，编码失败(执行器配置参数错误或者总线繁忙等导致)。 2. 返回值-61：日志“venc err interrupt state = xx, wait_time_remain = xx”编码失败(执行器配置参数错误)。
	-11	引擎处理异常，需要重试	
	-14	拷贝用户态数据时地址异常或者释放引擎失败	
	-16	申请引擎失败	
	-35	未等到完成中断，编码失败	

引擎类型	驱动IOCTL返回码	返回码说明	常见错误日志参考
	-61	上报编码超时中断	
VDEC	-22	参数错误或者不支持	常见错误打印： 1. 返回值-35：日志“VDM wait time out,wait_time_remain = xx ,engine_id:xx”解码超时(码流异常、总线繁忙等导致)。 2. 返回值=-61：日志“wait_time_remain = xx,have vdm_error,p_vdm_backup_state->VdmState=xx ,engine_id:xx”上报解码错误中断(码流异常等导致)。
	-11	引擎处理异常，需要重试	
	-14	拷贝用户态数据时地址异常或者释放引擎失败	
	-16	申请引擎失败	
	-35	解码超时	
	-61	上报解码错误中断	
	-4	等待解码被异常中断	

12.3.2 VPC 图片处理

12.3.2.1 调用错误的内存申请接口，导致内存地址校验出错

现象描述

不同版本的报错日志可能存在差别：

- 日志示例1
```
device 0, vpc address is illegal, please make sure it has been allocated with hi_mpi_dvpp_malloc or acldvppMalloc.
```
- 日志示例2
```
dvpp_check_mem_usable [Line]:85 mem:0x****f000****4020 is not usable, please check:1. mem not alloced or has been freed;2. make sure mem actual size should be:8017920
```

可能原因

根据日志提示，是由于没有使用指定的接口申请内存。

处理步骤

检查代码，是否使用媒体数据处理V1版本中的**acldvppMalloc**接口/媒体数据处理V2版本中的**hi_mpi_dvpp_malloc**接口申请存放VPC输入或输出数据的内存。

12.3.2.2 使用正确的内存申请接口，但内存大小传值错误

现象描述

不同版本的报错日志可能存在差别：

- 日志示例1
  buffer size(3110400) is smaller than need buffer size(4147200) when format is 3.
- 日志示例2
  device 0, vpc end address is illegal, check allocated buffer size: configured buffer size: 3110400, current pic: format 3 width_stride 1920 height_stride 1080.
- 日志示例3
  dvpp_check_mem_usable [Line]:85 mem:0x****f000****4020 is not usable, please check:1. mem not alloced or has been freed;2. make sure mem actual size should be:8017920

可能原因

1. 代码中申请的内存大小小于该格式所需的输入或输出内存大小;

2. VPC任务接口传入的buffer size正常，与输入格式匹配，但是超出了实际申请的内存长度，所以校验出来结束地址非法。

处理步骤

1. 检查代码，根据**图片格式、宽高对齐、内存约束**中的说明，检查对应格式的内存大小要求;

2. 在代码中增加打印内存长度的日志，检查VPC任务接口传入的buffer size是否与实际申请的内存长度一致。

12.3.2.3 读/写内存地址无效，导致异常中断

现象描述

Device侧内核态日志报错VPC异常（不同版本的报错日志可能存在差别）：

- 日志1：
  vpc get err int: vpc_cvdr_axi_rd_resp_err
- 日志2：
  vpc get err int: vpc_cvdr_axi_wr_resp_err

可能原因

- **cvdr_axi_rd_resp_err**表示读地址越界，可能申请的输入内存太小或内存地址无效，昇腾AI处理器执行读操作时访问到了无效地址。
- **cvdr_axi_wr_resp_err**表示写地址越界，可能申请的输出内存太小或内存地址无效，昇腾AI处理器执行写操作时访问到了无效地址。

处理步骤

1. 在申请DVPP内存的接口处、以及在VPC异常任务接口处增加日志打印，检查申请的输入\输出内存大小与实际使用的输入\输出内存大小是否一致；

2. 在释放DVPP内存的接口处增加打印日志，检查VPC任务完成之前是否存在内存被提前释放的情况。

12.3.2.4 VPC 调用失败

现象描述

VPC模块调用失败（不同版本的报错日志可能存在差别），查看日志有类似如下报错信息：

日志信息（1）：

```
RoiNum(0), inputArea rightOffset is 1918, it should be odd!
```

日志信息（2）：

```
Output bufferSize(65536) should not be smaller than widthStride(256) * heightStride(256) * 3 / 2 = 98304
```

日志信息（3）：

```
Input widthStride(300) is not right, it should be 16 algined!
Input widthStride(16) is not right, it should not be smaller than 32!
```

日志信息（4）：

```
bareDataAddr(0xaaaadeccdcd0), bareDataBufferSize(3133440) should be allocated by acldvppMalloc!
```

日志信息（5）：

```
Both RoiNum(1) outputAddr(0xaaaadeccdcd0) and first roi outputAddr(0xffff00002000) should be
allocated by acldvppMalloc!
```

日志信息（6）：

```
RoiNum(0): inputConfigure cropArea, leftOffset(26) should be smaller than rightOffset(25)!
RoiNum(0): inputConfigure cropArea, upOffset(80) should be smaller than downOffset(79)!
RoiNum(0): inputConfigure cropArea, cropWidth(1931) should not be bigger than width(1920)!
RoiNum(0): inputConfigure cropArea, cropHeight(1270) should not be bigger than height(1088)!
RoiNum(0): inputConfigure cropArea, cropWidth(9) should be between [10, 8192]!
RoiNum(0): inputConfigure cropArea, cropHeight(4) should be between [6, 8192]!
RoiNum(0): inputConfigure cropArea, rightOffset(1921) should be smaller than width(1920)!
RoiNum(0): inputConfigure cropArea, downOffset(1089) should be smaller than height(1088)!
```

日志信息（7）：

```
RoiNum(0): scale must be in [1/32, 16], cropWidth(1920), pasteWidth(10)!
RoiNum(0): scale must be in [1/32, 16], cropHeight(6), pasteHeight(100)!
```

可能原因

针对上面日志信息分析，可能存在以下对应原因：

- 日志信息（1）：VPC抠图区域右偏移坐标需是奇数，日志信息里1918是偶数，不符合要求。
- 日志信息（2）：当输出内存大小应该大于等于宽stride*高stride*3/2，日志信息显示不满足这个条件。
- 日志信息（3）：输入图片的宽stride（即每行图像占用的内存大小）必须是16倍数、且最小值32。日志信息里宽stride是300，不满足16倍数的要求，需要将图像做对齐后，并将宽stride设置成对齐后的值。
- 日志信息（4）：VPC的输入内存需要使用acldvppMalloc接口申请。
- 日志信息（5）：VPC的输出内存需要使用acldvppMalloc接口申请。
- 日志信息（6）：VPC的抠图区域不符合约束要求，因此报错。

- 日志信息（7）：VPC的缩放范围为[1/32, 16]。日志信息提示了缩放范围，并且显示了抠图宽为1920，输出区域的宽为10，通过计算可以得到：10/1920 < 1/32，因此报错。

定位思路

1. 根据日志描述的错误信息，找到VPC对应的配置参数，根据提示进行修改。
2. 根据日志描述的错误信息，参考**5 媒体数据处理（含图像/视频等）**中VPC功能参数的约束修改。

处理步骤

根据提示的错误信息进行修改：

步骤1 如果为日志信息（1），说明输入图片抠图区域的右偏移错误，应该设置为奇数。

步骤2 如果为日志信息（2），说明输入内存的大小不正确，应该检查申请输入内存的代码，申请内存大小应该为1920*1088*3/2，并且bareDataBufferSize这个值也要填写为1920*1088*3/2。

步骤3 如果为日志信息（3），说明输入图片的stride值不符合要求，需设置为16的倍数。

步骤4 如果为日志信息（4）和日志信息（5），代码中申请内存时，需要使用acldvppMalloc接口申请。

步骤5 如果为日志信息（6），需要修改抠图宽度。

步骤6 如果为日志信息（7），需要修改缩放范围。

----结束

12.3.2.5 VPC 参数校验失败

现象描述

调用VPC功能接口返回0xA0078003，即HI_ERR_VPC_ILLEGAL_PARAM，参数超出合法范围。查看日志有类似如下报错信息，不同版本的报错日志可能存在差别：

- VPC缩放宽大于输出宽日志信息（1）：
  ```
  resize width (224) is greater than output width (120)
  ```
  或
  ```
  width should be in [10, 32768], width is 654321
  ```
- VPC抠图宽大于输入宽日志信息（2）：
  ```
  crop width 120 cannot be greater than input width 100
  ```
- 输入的缩放算法不正确日志信息（3）：
  ```
  resize interpolation [8] is not supported
  ```
  或
  ```
  flip mode[5] not supported, support mode [0, 1, 2]
  ```
- 图片内存大小不够日志信息（4）：
  ```
  buffer size(50176) is smaller than need buffer size(95264) when format is 1
  ```
  或
  ```
  buffer size(50176) is smaller than need buffer size(95264) when format is 1
  ```
- 图片地址校验失败日志信息（5）：
  ```
  on device 0, num 0 input addr (start 0x100020003000 end 0x100020004000) is illegal
  ```

或

```
buffer address is null
```

- VPC抠图的输出宽大于输入宽或者输出高大于输入高日志信息（6）：

```
crop width[300] or height[300] is greater than input width[224] or height[224]
```

可能原因

针对上面日志信息分析，可能存在以下对应原因：

- 日志信息（1）：VPC缩放宽大于输出宽，
- 日志信息（2）：VPC抠图宽大于输入宽
- 日志信息（3）：输入的缩放算法不正确
- 日志信息（4）：图片内存大小不够
- 日志信息（5）：图片地址校验失败
- 日志信息（6）：VPC抠图的输出宽大于输入宽或者输出高大于输入高

定位思路

1. 根据日志描述的错误信息，找到VPC对应的配置参数，根据提示进行修改。
2. 根据日志描述的错误信息，参考**5 媒体数据处理（含图像/视频等）**中VPC功能参数的约束修改。

处理步骤

根据提示的错误信息进行修改：

步骤1 如果为日志信息（1），修改resize宽，使其小于等于输出宽。

步骤2 如果为日志信息（2）和（6），修改抠图宽，使其小于输入宽。

步骤3 如果为日志信息（3），修改缩放算法为当前版本支持的类型。

步骤4 如果为日志信息（4），需要按照格式申请足够的内存，并正确配置buffer_size

步骤5 如果为日志信息（5），使用hi_mpi_dvpp_malloc或acldvppMalloc 申请图片地址

----结束

12.3.2.6 VPC 创建通道失败

现象描述

调用hi_mpi_vpc_create_chn创建通道返回错误码HI_ERR_VPC_EXIST，查看日志有如下类似错误信息，不同版本的报错日志可能存在差别：

```
device 0, chn 0 has already been created!
```

或

```
dev 0 chnl 0 is busy
```

可能原因

对于VPC模块，调用hi_mpi_vpc_create_chn指定通道号，创建通道，任何情况下通道号不能重复。原因是该通道已存在了。

定位思路

检查代码，查看通道号的使用。

处理步骤

1、规划通道号，hi_mpi_vpc_create_chn创建时传入未被使用的通道号。

2、或者使用hi_mpi_vpc_sys_create_chn，调用完成后系统会分配一个未被使用过的通道号。

12.3.2.7 VPC 获取结果失败

现象描述

调用hi_mpi_vpc_get_process_result，返回HI_ERR_VPC_ILLEGAL_PARAM，查看日志信息，不同版本的报错日志可能存在差别：

```
this channel doesn't have taskID 8845!, Channel id 0
```

或

```
taskId:8845 does not exist
```

可能原因

调用hi_mpi_vpc_get_process_result传入一个非法的task id。

处理步骤

调用hi_mpi_vpc_get_process_result传入的task id，必须是功能接口中传出的。

将功能接口中传出的task id传入hi_mpi_vpc_get_process_result。

12.3.3 JPEGD 图片解码/VDEC 视频解码

12.3.3.1 调用错误的内存申请接口，导致内存地址校验出错

现象描述

日志报错如下：

```
address check failed ret 0x3, please check: 1. use hi_mpi_dvpp_malloc or aclDvppMalloc to alloc dvpp memory; 2. current buffer size 3110400 may be larger than actually allocated.
```

可能原因

根据日志提示，可能由于：

1. 没有使用指定的DVPP内存申请接口;

2. 接口传入的buffer size小于实际申请的内存大小。

处理步骤

检查代码：

1. 是否使用媒体数据处理V1版本中的**acldvppMalloc**接口/媒体数据处理V2版本中的**hi_mpi_dvpp_malloc**接口申请存放JPEGD图片解码/VDEC视频解码输入或输出数据的内存;

2. 对于DVPP内存申请接口，增加日志打印内存大小及地址，检查接口**hi_mpi_vdec_get_frame**/**aclvdecSendFrame**/**acldvppJpegDecodeAsync**送入的buffer size是否超出了实际申请的内存区域。

12.3.3.2 内存被提前释放，导致解码数据花屏

现象描述

原始H264/H265每一帧视频流正常，解码过程无异常（无异常日志），仅仅输出图片有异常。

可能原因

解码过程无异常，说明送入的码流非异常码流，仅仅输出被破坏，可能由于：

1. 输出内存被别人复用，被踩或者被提前释放;

2. 解码需要的输出内存比实际申请的内存大。

处理步骤

1. 对于DVPP内存申请接口，增加日志打印内存大小及地址，检查VDEC输出内存，检查申请的内存大小是否与实际使用的一致，比如典型的错误场景，VDEC解码输出格式预期是RGB，实际仍按照YUV420sp申请内存。
2. 在DVPP内存释放接口处、以及**hi_mpi_vdec_get_frame**/**aclvdecCallback**/**acldvppJpegDecodeAsync**接口处，增加内存大小及地址的打印日志，确认内存释放时序，是否存在内存地址解码完成前被提前释放的情况。

12.3.3.3 JPEGD 图像解码进程超时退出

现象描述

用户进程退出。

查看应用类日志，发现类似ERROR日志“task timeout, userData= ..., timeout=30, duration=…”和WARNING日志“frames statistic: ACL receive(*n*), send(*n*-1)”，*n*表示处理任务数量，需根据实际情况确定。

日志片段举例如下：

```
[ERROR] AICPU(pid,pName):DateTimeMS [dvpp_timeout_manager.cc:33][OnPulse][tid:2581]
[DVPP_KERNELS] WaitId[10] task timeout, userData=0xe7ffe0001280, timeout=30, duration=30.930062.
[INFO] AICPU(pid,pName):DateTimeMS [dvpp_kernel_base.cc:222][SendTaskCompleteToTs][tid:2581]
[DVPP_KERNELS] Send task complete to ts success, taskId=2, streamId=44, errorCode=1.
[WARNING] DVPP(pid,pName):DateTimeMS [JpegdAsyncManager.cpp:405][API] [PrintFrameCount:405]
[T208] DFX[JPEGD]: frames statistic: ACL receive(16), send(15)
```

可能原因

多路并发解码JPEG图片场景下，如果每一路解码的JPEG图片中，都包含带旋转信息的大分辨率图片，例如3840*2160分辨率及以上的图片，则可能导致图片解码时间过长，从而导致用户进程超时退出。

处理步骤

步骤1 确定大分辨率的图片是否包含旋转信息。

使用JPEG码流分析工具（例如JPEGsnoop）解析大分辨率的图片，查看其是否包含旋转信息，若Orientation信息为1，则表示不旋转；否则，都带有一定角度的旋转，例如下图解析出来的Orientation信息为8，表示顺时针旋转270°。

```
EXIF IFD0 @ Absolute 0x00000026
  Dir Length = 0x0005
  [Orientation                          ] = 8 = Row 0: left, Col 0: bottom
  [Software                             ] = "ACD Systems ........"
  [DateTime                             ] = "2018:04:10 10:43:17"
  [YCbCrPositioning                     ] = Centered
  [ExifOffset                           ] = @ 0x0074
  Offset to Next IFD = 0x00000000
```

步骤2 如果确定这些图片是带旋转的大分辨率图片，则建议用户先调用第三方库（例如OpenCV）进行解码。

----结束

12.3.3.4 JPEGD 图片解码失败

现象描述

JPEGD模块解码失败，查看日志有类似如下报错信息：

日志信息（1）：

```
just support jpeg with YUV 444 422 420 400
do not support progressive mode
do not support arithmetic code, support huffman code only
```

日志信息（2）：

```
EOI segment of the stream is invalid
```

可能原因

分析上面日志信息，可能存在以下可能原因：

1. 数据格式不支持

   JPEGD只支持huffman编码(colorspace: yuv, subsample: 444/422/420/400)，不支持算术编码，不支持渐进编码，不支持jpeg2000格式。

2. 图像数据不完整

处理步骤

针对上述可能原因，请按以下方式处理：

步骤1 针对目前不支持的超规格图像格式，建议用户自行使用第三方软件解码。

步骤2 针对图像数据不完整，根据报错提示，通过第三方软件查看原图像二进制进行确认。

例如“EOI segment of the stream is invalid”或“EOI segment of the stream is invalid, it should be FFD9. Try software decoding.”报错，表示图像缺失最后的EOI

结束符，对应图像二进制类似下图所示。正常JPEG图片最后应该由标记码FF D9结束，该数据最后缺失FF D9标记码。

如果确认原图数据不完整，报错属于正常现象，需更换数据。

```
00005e90h: AB 79 F4 E3 4B CE 38 9C B8 B7 D2 B2 1C 59 4E A5 ; 坦蓋K?涅分?YN?
00005ea0h: 5F 73 B6 36 D6 FF 00 E5 7F FA BF F7 71 A2 47 FC ; _s??.?  鱗  ?
00005eb0h: 91 FC D3 FC 30 4C 6A 84 EB 8E AF 5B 8B 2A 56 BE ; 廖狱0Lj勲暢[?V?
00005ec0h: 49 FA 60 D5 6D F7 67 3F 11 F9 4A D6 E2 CA 35 2A ; I鹳謹鰡?.鴳肘?*
00005ed0h: C0 13 FA B6 FD B0 0A 9D C7 FD BF F8 60 CD 43 E4 ; ?    .瀋  鴃虾?
00005ee0h: 83 F7 47 FF 00 B6 DE 31 80 2C BE E0 60 01 A7 93 ; 淦G  .掇1€,距`.
00005ef0h: FB 23 DF ED 8C FE 21 DF F4 7F D4 1F D5 8D 0D 7F ; ?唔岨!嗥?雪.
00005f00h: 33 F9 9F E3 8C F1 8C 7F 00 00                   ; 3鹳銓駛..
```

步骤3 如果原图像数据完整，可能数据在传输过程中存在损坏，需要在调用图片编码之前，通过fwrite函数将传输给JPEGD的码流保存下来，与原图进行二进制比较。如果不一致，传输过程出现数据缺失，需自行进一步定位传输过程数据缺失问题。

----结束

12.3.3.5 VDEC 视频解码丢帧/丢包

现象描述

视频解码丢帧，出现重影或不连续现象。查看Device侧日志，发现日志中存在以下几个报错的内容信息中的一个或多个，不同版本日志信息可能有所不同。

- H264码流缺少IDR帧日志报错信息（1）

```
[HiDvpp][A618] [Vfmw]:ppssps_check [Line]:6803 pps with this pic_parameter_set_id = %d haven't decode
[HiDvpp][A618] [Vfmw]:process_slice_header_first_part [Line]:7401 PPS or SPS of this slice not valid
[HiDvpp][A618] [Vfmw]:h264_dec_slice [Line]:7915 sliceheader dec err
```

或

```
[ERROR] DVPP:2020-12-31-23:51:51.339.518 [VDEC][PPSSPSCheckTmpId:7065][T3] PPSSPSCheckTmpId: pps with this pic_parameter_set_id = 0 haven't decode
[ERROR] DVPP:2020-12-31-23:51:51.339.616 [VDEC][ProcessSliceHeaderFirstPart:7627][T3]  PPS or SPS of this slice not valid
[ERROR] DVPP:2020-12-31-23:51:51.339.678 [VDEC][InquiresSlceProperty:10582][T3]  sliceheader dec err
```

- H264码流缺少I帧日志报错信息（2）

```
[HiDvpp][A618] [Vfmw]:h264_dec_slice [Line]:7983 init pic err, find next recover point or next valid sps, pps, or exit
[HiDvpp][A618] [Vfmw]:h264_dec_slice [Line]:3716 dec list error, ret=-1
[HiDvpp][A618] [Vfmw]:receive_packet [Line]:10676 nal_release_err
```

或

```
[ERROR] DVPP:2020-12-31-20:51:51.318.218 [VDEC][InitPic:6039][T3]  line 6039: frame gap(=48) > dpb size(=2)
[ERROR] DVPP:2020-12-31-20:51:51.318.266 [VDEC][H264_DecSlice:8238][T3]  init pic err, find next recover point or next valid sps, pps, or exit
[ERROR] DVPP:2020-12-31-20:51:51.318.336 [VDEC][H264_DecOneNal:10077][T3]  DecList error, ret=-1
[ERROR] DVPP:2020-12-31-20:51:51.318.392 [VDEC][ReceivePacket:10400][T3]  nal_release-err
```

- H264码流缺少P帧日志报错信息（3）

```
[HiDvpp][A618] [Vfmw]:init_list_x [Line]:4829 for P slice size of list equal 0.ctx->dpb.ref_frames_in_buffer:0.
[HiDvpp][A618] [Vfmw]:dec_list [Line]:5068 init list error.
[HiDvpp][A618] [Vfmw]:h264_dec_list [Line]:4829 dec_list error, ret=-1
[HiDvpp][A618] [Vfmw]:h264_dec_one_nal [Line]:10298 slice_check failed, clear current slice.
```

或

```
[ERROR] DVPP:2020-12-31-20:30:22.188.008 [VDEC][InitListX:4513][T3]  for P slice size of list equal 0.
[ERROR] DVPP:2020-12-31-20:30:22.188.056 [VDEC][DecList:4832][T3] line: 4832 InitListX failed
[ERROR] DVPP:2020-12-31-20:30:22.188.128 [VDEC][H264_DecSlice:8260][T3] DecList error, ret=-1
[ERROR] DVPP:2020-12-31-20:30:22.188.199 [VDEC][H264_DecOneNal:10077][T3] Decoder Slice
failed
```

- H264码流缺少B帧日志报错信息（4）

```
[HiDvpp][A618] [Vfmw]:init_list_x [Line]:4865 for B slice size of two list all equal 0.
[HiDvpp][A618] [Vfmw]:dec_list [Line]:5068 init list error.
[HiDvpp][A618] [Vfmw]:h264_dec_list [Line]:4829 dec_list error, ret=-1
```

或

```
[ERROR] DVPP:2020-12-31-10:20:28.528.090 [VDEC][InitListX:4653][T3] for B slice size of two list all
equal 0.
[ERROR] DVPP:2020-12-31-10:20:28.528.168 [VDEC][DecList:4830][T3] line: 4832 InitListX failed
[ERROR] DVPP:2020-12-31-10:20:28.528.266 [VDEC][H264_DecSlice:8257][T3] DecList error, ret=-1
```

- H265码流缺少IDR帧日志报错信息（5）

```
[HiDvpp][A618] [Vfmw]:hevc_vps_sps_pps_check [Line]:7300 pps with this pic_parameter_set_id = 0
haven't be decoded
[HiDvpp][A618] [Vfmw]:hevc_dec_slice_segment_header [Line]:3857 hevc_vps_sps_pps_check !=
HEVC_DEC_NORMAL
[HiDvpp][A618] [Vfmw]:hevc_inquire_slice_property [Line]:9004 hevc_dec_slice_segment_header dec
err
[HiDvpp][A618] [Vfmw]:hevc_dec_decode_packet[Line]:9004 hevc_inquire_slice_property error.
```

或

```
[ERROR] DVPP:2020-12-31-10:30:22.130.500 [VDEC][HEVC_VpsSpsPpsCheck:8084][T10] pps with this
pic_parameter_set_id = 0 haven't be decoded
[ERROR] DVPP:2020-12-31-10:30:22.130.598 [VDEC][HEVC_DecSliceSegmentHeader:2793][T10]
HEVC_VpsSpsPpsCheck != HEVC_DEC_NORMAL
[ERROR] DVPP:2020-12-31-10:30:22.130.686 [VDEC][HEVC_InquireSliceProperty:10169][T10]
HEVC_DecSliceSegmentHeader dec err
[ERROR] DVPP:2020-12-31-10:30:22.130.789 [VDEC][HEVCDEC_DecodePacket:753][T10]
HEVC_InquireSliceProperty error.
```

- H265码流缺少I帧或者P帧日志报错现象（6）

```
[HiDvpp][A618] [Vfmw]:hevc_ref_pic_process [Line]:3474 ref frame(poc 15) lost.
[HiDvpp][A618] [Vfmw]:hevc_create_lost_picture [Line]:5839 DPB no suited fs for lost pic.
[HiDvpp][A618] [Vfmw]:hevc_create_lost_picture [Line]:5847 take poc(17) to create lost poc(15).
```

或

```
[ERROR] DVPP:2020-12-31-11:22:28.800.158 [VDEC][HEVC_RefPicProcess:2480][T10] Ref frame(poc
15) lost.
[ERROR] DVPP:2020-12-31-11:22:28.800.236 [VDEC][HEVC_CreateLostPicture:6392][T10] DPB no
suited fs for lost pic.
[ERROR] DVPP:2020-12-31-11:22:28.800.352 [VDEC][HEVC_RefPicProcess:2480][T10] Ref frame(poc
18) lost.
[ERROR] DVPP:2020-12-31-11:22:28.800.426 [VDEC][HEVC_CreateLostPicture:6392] [T10] DPB no
suited fs for lost pic.
[ERROR] DVPP:2020-12-31-11:22:28.800.522 [VDEC][HEVC_RefPicProcess:2480] [T10] Ref frame(poc
18) lost.
```

- H265码流缺少I帧或者B帧日志报错信息（7）

```
[ERROR] DVPP:2020-12-31-11:56:35.038.109 [VDEC][HEVC_RefPicProcess:2480] [T56] Ref frame(poc
15) lost.
[ERROR] DVPP:2020-12-31-11:56:35.038.283 [VDEC][HEVC_CreateLostPicture:6392] [T56] Take
poc(17) to create lost poc(15).
[ERROR] DVPP:2020-12-31-11:56:35.038.502 [VDEC][FSP_SetRef:934] [T56] check condition:
pstLogicFs->IsDummyFs == 0 fail
[ERROR] DVPP:2020-12-31-11:56:35.038.801 [VDEC][FSP_SetRef:934] [T56] check condition:
pstLogicFs->IsDummyFs == 0 fail
[ERROR] DVPP:2020-12-31-11:56:35.039.128 [VDEC][FSP_SetRef:934] [T56] check condition:
pstLogicFs->IsDummyFs == 0 fail
```

可能原因

分析上述日志报错信息现象，分别可能存在以下可能原因：

- 日志报错信息（1）可能原因：H264码流缺少IDR帧
- 日志报错信息（2）可能原因：H264码流缺少I帧
- 日志报错信息（3）可能原因：H264码流缺少P帧
- 日志报错信息（4）可能原因：H264码流缺少B帧
- 日志报错信息（5）可能原因：H265码流缺少IDR帧
- 日志报错信息（6）可能原因：H265码流缺少I帧或者P帧
- 日志报错信息（7）可能原因：H265码流缺少I帧或者B帧

处理步骤

针对可能原因分析，参考以下步骤处理：

步骤1 检查输入的源码流是否有问题。

使用第三方工具（如：eseye u）对输入码流进行检查，查看码流是否异常。

步骤2 若查看的源码流结果为正常，则可能码流在传输给设备侧VDEC的过程中遭到破坏，需要在调用发送码流接口之前，通过fwrite函数将输送给VDEC的码流保存下来。

- 使用第三方工具对保存的码流进行检查，如果码流异常，用户需自行排查将码流送进去之前是否有送流问题。
- 通过对应版本的sample，解码这段保留下来的码流，验证码流是否正常或VDEC是否支持该格式。

  如果sample 解码正常，那就是开发代码有问题，可以参考VDEC示例代码，找到对应的视频解码的代码参考优化。

----结束

12.3.3.6 VDEC 视频解码花屏

现象描述

输入码流给VDEC进行解码，得到的解码数据不正确，产生花屏现象，如图12-5所示。并且日志中存在类似“decode error”、“input stream error, can't decode, report to user”、“num_ref_idx_l0_active(30) out of range(0,16)”、“dvpp_vdec_vdm_process failed”、“Chan 0 ErrRatio = 44”信息。

图 12-5 视频花屏

可能原因

输入的码流中某些帧数据不完整、存在坏帧，导致硬件解码产生花屏。

处理步骤

针对可能原因分析，参考以下步骤处理：

步骤1 检查输入的源码流是否有问题。

使用第三方工具（如：eseye u等）对输入码流进行解码播放，查看是否存在花屏，若不花屏则进行**步骤2**；若花屏则替换输入码流。

步骤2 若查看的源码流结果为正常，则可能码流在传输给设备侧VDEC的过程中遭到破坏，需要在调用发送码流接口之前，通过fwrite函数将输送给VDEC的码流保存下来。

- 使用第三方工具对保存的码流进行检查，如果码流异常，用户需自行排查将码流送进去之前是否有送流问题。
- 通过对应版本的sample，解码这段保留下来的码流，验证码流是否正常或VDEC是否支持该格式。

  如果sample 解码正常，那就是开发代码有问题，可以参考**5 媒体数据处理（含图像/视频等）**中的VDEC示例代码，找到对应的视频解码的代码参考优化。

----结束

12.3.3.7 VDEC 视频解码性能问题

现象描述

VDEC解码性能下降，小于VDEC对外发布的性能规格，导致发生卡顿等现象。

可能原因

分析VDEC性能下降导致的卡顿故障，可能原因有：

- 视频解码回调函数中耗时过长，影响解码性能。
- 输入码流中I帧比例过大，解码I帧耗时比解码P帧耗时长，影响解码性能。
- 输入码流中存在异常帧，影响解码性能。

处理步骤

针对上述可能的故障原因，可以参考以下方式进行处理：

步骤1 在回调函数中打点测试耗时，查看耗时是否过长(回调函数允许的最大耗时和帧率相关，计算公式为：最大耗时=1/帧率，例如帧率=30fps，则最大耗时=1/（30fps）=0.033s)。

步骤2 使用第三方工具打开输入码流，查看I帧比例是否过大。一般GOP值为30(即I帧间隔为30)，如果I帧比例过大，则需要替换成正常码流进行性能测试。

步骤3 使用第三方工具打开输入码流，查看是否存在异常帧（如用第三方工具打开显示花屏或解码报错），如果存在异常帧，会造成不满足规格的现象。

----结束

12.3.3.8 VDEC 视频解码失败

现象描述

视频解码失败，日志中打印如下所示的异常信息，不同版本日志信息可能有所不同：

- 日志信息：Invalid width or height

```
[HiDvpp][A618] [Vfmw] vdec_drv_check_video_wh [Line]:127 device 0 chn 0 type(255) pic
size(32x32) is out of range[128,128]x[4096,4096].
```

或

```
[ERROR] DVPP:2020-12-31-20:50:51.318.218 [VDEC][IsValidResolution:639][T56] Invalid width or
height, valid range (w:128~4096) (h:128~4096), current width = 32, height = 32, realWidth = 18,
realHeight = 18
[ERROR] DVPP:2020-12-31-20:50:51.318.365 [VDEC][upgrade_picture_info_in_detail:679][T56] check
condition: ret == OMX_ErrorNone fail
[ERROR] DVPP:2020-12-31-20:50:51.318.429 [VDEC][EventHandLer:386][T56] Dynamic Resouces
Unavailable now
[ERROR] DVPP:2020-12-31-20:50:51.368.106 [VDEC][hanlde_release_instance:1352][T56] wait
Componment Exit Message Thread
```

- 日志信息：bit_depth_luma(*) not equal *

```
[HiDvpp][A618] [Vfmw] process_sps [Line]:8994 bit_depth_luma(%d) not equal 8.
[HiDvpp][A618] [Vfmw] hevc_process_sps [Line]:1462 chn 0, bit_depth_luma(9) is not supported,
hevc only support 8 or 10 bit depth.
```

或

```
[ERROR] DVPP(13757,dvpp_performance):2020-12-31-21:18:26.998.657 [VDEC][ProcessSPS:9055]
[T26] bit_depth_luma(10) not epual 8.
[ERROR] DVPP(13757,dvpp_performance):2020-12-31-21:18:26.998.763 [VDEC][ProcessSPS:9070]
[T26] bit_depth_chroma(10) not epual 8.
```

可能原因

针对上述日志分析，可能存在输入码流规格不符合软硬件约束。

处理步骤

根据5 媒体数据处理（含图像/视频等）中的VDEC功能及约束，使用第三方工具（如：eseye u等）对输入码流进行检查，查看码流类型、码流宽高等信息是否符合要求。如果码流不满足要求，请替换符合要求的码流。

12.3.3.9 VDEC 视频帧解码失败不触发回调函数

现象描述

用户输入码流给VDEC解码，某些帧或所有帧都没有触发回调函数，用户收不到解码结果。

可能原因

码流中某些帧是坏帧，导致语法解析不出这些帧的含义，或者解码这些帧失败，从而不调用回调函数。

处理步骤

针对分析的可能原因，请参考以下步骤进行处理：

步骤1 查看日志中是否有12.3.3.5 VDEC视频解码丢帧/丢包中的日志报错信息，若有，则是因为异常帧解码失败导致没有回调。

步骤2 若没有12.3.3.5 VDEC视频解码丢帧/丢包中的日志报错信息，则调整DVPP模块或全局日志级别为info，查看下述三条日志打印的总次数是否和输入的帧数相等。

- （1）"The queue is empty, so call the non-intelligent pointer callback interface."
- （2）"The queue is not empty, so call the smart pointer callback interface."
- （3）"The queue is not empty, but hiai_data_sp is nullptr."

生成上述3种日志信息的场景如下：

- 未使用hiai_data_sp，成功解码返回，打印（1）日志，然后调用用户注册的回调函数。
- 每一帧对应设置一个hiai_data_sp，成功解码返回，打印（2）日志，然后调用用户注册的回调函数。
- N帧对应设置一个hiai_data_sp，第1帧成功解码返回，打印（2）日志，然后调用用户注册的回调函数；其他N-1帧，成功解码后打印（3）日志信息，并调用用户注册的回调函数。

不论上述场景中哪一种，均会调用用户注册的回调函数，即只要调用一次用户回调函数就说明解码返回一帧。所以上述三条日志出现的次数总和与用户输入总帧数相等，则说明无解码丢帧，用户需排查自身接收解码结果的统计是否有误。

----结束

12.3.3.10 VDEC 视频解码异常导致进程卡死，无法退出

现象描述

用户进程卡死，无法退出。

查看应用类日志，一直重复提示信息“**fault kernel_name=DvppSendVdecFrame**”、“**Kernel task happen error, retCode=0x28, [aicpu timeout]**”，表示AI CPU异常，无法处理VDEC解码任务，导致任务超时。

日志片段举例如下：

```
[ERROR] RUNTIME(pid,pName):DateTimeMS [task.cc:878]1827 PreCheckTaskErr:[DVPP][DEFAULT]Kernel task happen error, retCode=0x28, [aicpu timeout].
[ERROR] RUNTIME(pid,pName):DateTimeMS [task.cc:676]1827 PrintAicpuErrorInfo:[DVPP][DEFAULT]Aicpu kernel execute failed, device_id=0, stream_id=177, task_id=4, fault so_name=libdvpp_kernels.so, fault kernel_name=DvppSendVdecFrame, fault op_name=, extend_info=.
[ERROR] RUNTIME(pid,pName):DateTimeMS [task.cc:878]1831 PreCheckTaskErr:[DVPP][DEFAULT]Kernel task happen error, retCode=0x28, [aicpu timeout].
[ERROR] RUNTIME(pid,pName):DateTimeMS [task.cc:676]1831 PrintAicpuErrorInfo:[DVPP][DEFAULT]Aicpu kernel execute failed, device_id=0, stream_id=170, task_id=8, fault so_name=libdvpp_kernels.so, fault kernel_name=DvppSendVdecFrame, fault op_name=, extend_info=.
[ERROR] RUNTIME(pid,pName):DateTimeMS [engine.cc:960]1766 ReportExceptProc:[DVPP][DEFAULT]Task exception! device_id=0, stream_id=107, task_id=8, type=1, retCode=0x28.
[ERROR] RUNTIME(pid,pName):DateTimeMS [engine.cc:960]1773 ReportExceptProc:[DVPP][DEFAULT]Task exception! device_id=0, stream_id=130, task_id=4, type=1, retCode=0x28.
```

可能原因

Device内存不足，AI CPU无法处理VDEC解码任务，导致任务超时。

处理步骤

步骤1 在使用媒体数据处理V1版本的VDEC视频解码功能前，可参考**11.17.9.1 功能及约束说明**中“每路VDEC解码的内存消耗计算公式”，预估需使用的Device内存，并合理规划Device上的内存。

步骤2 优化应用程序的代码逻辑，增加异常处理机制，获取VDEC解码异常信息，强制退出进程。

在调用aclinit接口之后，定义异常回调函数，并调用aclrtSetExceptionInfoCallback接口设置异常回调函数，用于获取任务异常信息，以便在异常分支中根据任务异常信息来判断是否退出应用进程。

基本接口调用逻辑如下：

1. 定义并实现异常回调函数fn(aclrtExceptionInfoCallback类型)，回调函数原型为：
   typedef void (*aclrtExceptionInfoCallback)(aclrtExceptionInfo *exceptionInfo)

   在异常回调函数fn内调用**aclrtGetDeviceIdFromExceptionInfo**、**aclrtGetStreamIdFromExceptionInfo**、**aclrtGetTaskIdFromExceptionInfo**接口分别获取Device ID、Stream ID、Task ID。

   根据Stream ID、Task ID判断Device是否异常，若异常，则强制退出进程。

   异常回调函数实现示例如下：

```
void dvpp_callback(aclrtExceptionInfo * exception_info)
{
    uint32_t taskId = aclrtGetTaskIdFromExceptionInfo(exception_info);
    uint32_t streamId = aclrtGetStreamIdFromExceptionInfo(exception_info);
    uint32_t deviceId = aclrtGetDeviceIdFromExceptionInfo(exception_info);

    if(taskId == 0xffffffff) || (streamId == 0xffffffff) {
        //Device异常，强制退出进程
    } else {
        //任务异常，如果频繁出现（例如，统计1秒内触发异常回调函数的次数），进程退出
    }
    return;
}
```

2. 调用**aclrtSetExceptionInfoCallback**接口设置异常回调函数。
3. 执行VDEC解码，接口调用流程请参见**5.4.5 VDEC视频解码**。

----结束

12.3.3.11 retCode 返回值设置错误，导致 VDEC 视频解码异常

现象描述

调用aclvdecSendFrame接口发送一帧码流后，继续复用output输出图片描述信息，进行后续帧码流的解码操作，结果反复出现解码不成功、解码异常的情况。

日志片段举例如下：

```
Channel[0]: success to aclvdecSendFrame, loop=1, count=7
get frame success, totalCount=7
packet.size is 26084.
Channel[0]: begin to send frame, loop=1, count=8
acldvppGetPicDescRetCode, retCode: 2.
Vdec ERROR!!!!!!!!!!!!!!!!
errCount is 3. total count is 3.
!!!!!!!!!!!!!!!!!acldvppGetPicDescRetCode, retCode: 2.right_count:0,fail_count:3,total_count:3
Channel[0]: success to aclvdecSendFrame, loop=1, count=8
get frame success, totalCount=8
packet.size is 27927.
Channel[0]: begin to send frame, loop=1, count=9
acldvppGetPicDescRetCode, retCode: 2.
Vdec ERROR!!!!!!!!!!!!!!!!
errCount is 4. total count is 4.
!!!!!!!!!!!!!!!!!acldvppGetPicDescRetCode, retCode: 2.right_count:0,fail_count:4,total_count:4
```

可能原因

根据日志中的提示，通过acldvppGetPicDescRetCode接口获取到的retCode为2，retCode为非0值时，表示解码异常。

再查看代码逻辑时，发现由于前一帧码流解码失败，retCode被置为2，在复用output输出图片描述信息时，retCode也继承了前一帧解码失败的状态值2，导致AscendCL在解码后续帧时，获取到retCode值为2，就一直判断解码是失败。

处理步骤

如果存在复用output输出图片描述信息的场景，需先调用**acldvppSetPicDescRetCode**设置为0，防止前一帧解码异常的状态影响后续解码。

12.3.3.12 VDEC 视频解码无报错，但无解码结果数据、CPU 占用率高

现象描述

查看应用类日志，无ERROR报错、无解码结果数据输出，另外，在运行应用程序的Linux服务器上执行top命令，该应用进程的cpu占用率持续升高。

可能原因

1. 无ERROR、无解码结果数据输出，判断可能是因为解码发帧接口**aclvdecSendFrame**调用正常，但未触发回调函数，无法获取解码结果数据。

2. 检查触发回调函数的代码逻辑。

   按照VDEC视频解码的接口调用逻辑：由用户提前创建一个单独的线程，并自定义线程函数，在线程函数内调用**aclrtProcessReport**接口，通过该接口配置超时时间，等待指定的超时时间后，触发回调函数，获取解码结果数据。

```
void *ThreadFunc(aclrtContext sharedContext)
{
    if (sharedContext == nullptr) {
        ERROR_LOG("sharedContext can not be nullptr");
        return ((void*)(-1));
    }
    INFO_LOG("use shared context for this thread");
    aclError ret = aclrtSetCurrentContext(sharedContext);
    if (ret != ACL_SUCCESS) {
        ERROR_LOG("aclrtSetCurrentContext failed, errorCode = %d", static_cast<int32_t>(ret));
        return ((void*)(-1));
    }

    INFO_LOG("thread start ");
    while (runFlag) {
        // Notice: timeout 1000ms
        (void)aclrtProcessReport(1000);
    }
    return (void*)0;
}
```

3. 如果触发回调函数的接口调用逻辑正确，则在**aclrtProcessReport**接口处增加日志打印，判断应用运行过程中线程是否成功调用了**aclrtProcessReport**接口，只有成功调用**aclrtProcessReport**接口，才会触发回调函数。

   示例代码如下：

```
while (runFlag) {
        // Notice: timeout 1000ms
        aclError ret = aclrtProcessReport(1000);
       printf("aclrtProcessReport failed, ret=%d.\n", ret);
}
```

4. 修改代码后，重新编译运行应用。

   在终端屏幕重复出现以下打印信息，表示调用**aclrtProcessReport**接口失败：

```
aclrtProcessReport failed, ret = 107012
```

   查阅该接口的返回值说明，107012表示线程未订阅或重复订阅。

5. 检查代码逻辑，检查是否调用**aclvdecSetChannelDescThreadId**接口绑定用户新建的线程，按照VDEC视频解码的接口调用逻辑，只有调用该接口绑定用户线程，才可以触发调用**aclrtProcessReport**接口，进而触发回调函数。

6. 修改代码后，重新编译运行应用，VDEC视频解码正常，正常输出解码结果数据，同时cpu占用率下降。

解决办法

参见VDEC的**5.4.5 VDEC视频解码**或者参考**Link**上的VDEC功能样例开发VDEC功能。

其中，需关注以下注意点：

- 创建新线程，并自定义线程函数，在线程函数内调用**aclrtProcessReport**接口，等待指定时间后，触发回调函数中的回调函数。
- 需调用**aclvdecSetChannelDescThreadId**接口绑定用户创建的新线程。
- 释放资源时，依次销毁通道、销毁通道描述信息后，才可以销毁中用户创建的新线程。

12.3.3.13 复用输出图片描述类型，VDEC 视频解码报错，提示有不支持的图片格式

问题现象

循环调用aclvdecSendFrame接口解码视频中的每一帧码流时，在遇到异常帧之后，解码下一帧就会报错，退出应用进程。

分别查看Host侧日志、Device侧日志，发现Device日志中提示**the out format 0 is not supported**，日志片段如下：

- Device侧日志：

```
[ERROR] KERNEL(2234,sklogd):2023-06-13-19:21:22.987.969 [klogd.c:246][652145.056916] [HiDvpp] [A618] [Vdec]:vdec_check_resize_param [Line]:6768 pid 23973 usr chn 0 device 0 chn 0 the out format 0 is not supported.
```

- Host侧日志：

```
[ERROR] RUNTIME(17174,AIMCDemo):2023-06-13-19:21:23.664.211 [api_c.cc:721]17184 rtStreamSynchronize:[DVPP][DEFAULT]ErrCode=507018, desc=[aicpu exception], InnerCode=0x715002a
[ERROR] RUNTIME(17174,AIMCDemo):2023-06-13-19:21:23.664.215 [error_message_manage.cc:49]17184 FuncErrorReason:[DVPP][DEFAULT]report error module_type=3, module_name=EE8888
[ERROR] RUNTIME(17174,AIMCDemo):2023-06-13-19:21:23.664.221 [error_message_manage.cc:49]17184 FuncErrorReason:[DVPP][DEFAULT]rtStreamSynchronize execute failed, reason=[aicpu exception]
[INFO] GE(17174,AIMCDemo):2023-06-13-19:21:23.664.227 [error_manager.cc:252]17184 ReportInterErrMessage:report error_message, error_code:EE8888, work_stream_id:1717417184
[ERROR] ASCENDCL(17174,AIMCDemo):2023-06-13-19:21:23.664.234 [video_processor_v200.cpp:1089]17184 aclvdecSendFrame: [DVPP][DEFAULT][Sync][Stream]vdec fail to synchronize sendFrameStream, runtime errorCode = 507018, channelId = 0.
```

原因分析

检查应用代码，发现循环解码视频中的每一帧码流时，复用aclvdecSendFrame接口的输出图片描述类型acldvppPicDesc，但在下一次解码前没有重新设置输出图片format、width、height、widthStride、heightStride，这时，如果前一帧解码失败，acldvppPicDesc的参数format、width、height、widthStride、heightStride变成默认值0，width、height、widthStride、heightStride为0时，vdec会以实际图片宽高解码输出，但format为0，表示YUV400格式，vdec不支持解码输出该格式，会导致下一帧参数不合法解码失败。

解决方法

优化应用代码逻辑，复用输出图片描述类型acldvppPicDesc时，在下一次解码前需重新设置输出图片format、width、height、widthStride、heightStride。

正例代码片段：

```
aclError ret;
int restLen = 10;
uint32_t inBufferSize = 0;
void *g_picOutBufferDev;
void *inBufferDev = nullptr;
acldvppPicDesc *picOutputDesc;
size_t dataSize = (INPUT_WIDTH * INPUT_HEIGHT * 3) / 2;

// 申请一个picOutputDesc,每帧复用
picOutputDesc = acldvppCreatePicDesc();
// read file to device memory
 ReadFileToDeviceMem(filePath.c_str(), inBufferDev, inBufferSize);
 while (restLen > 0) {
        // 等待前一个使用picOutputDesc解码帧结束，重新复用picOutputDesc,并针对这一帧重新设置Format、width、height、widthStride、heightStride参数值
        ret = acldvppSetPicDescFormat(picOutputDesc, static_cast<acldvppPixelFormat>(1)); // 1：YUV420 semi-planner（nv12）
```

```
        ret = acldvppSetPicDescWidth(picOutputDesc, 1920);
        ret = acldvppSetPicDescHeight(picOutputDesc, 1080);
        ret = acldvppSetPicDescWidthStride(picOutputDesc, 1920);
        ret = acldvppSetPicDescHeightStride(picOutputDesc, 1080);
        ret = acldvppMalloc(&g_picOutBufferDev, dataSize);
        ret = acldvppSetPicDescData(picOutputDesc, g_picOutBufferDev);
        ret = acldvppSetPicDescSize(picOutputDesc, dataSize);
        ret = aclvdecSendFrame(vdecChannelDesc, streamInputDesc, picOutputDesc, nullptr, nullptr);
        restLen = restLen - 1;
}
```

反例代码片段：

```
aclError ret;
int restLen = 10;
uint32_t inBufferSize = 0;
void *g_picOutBufferDev;
void *inBufferDev = nullptr;
acldvppPicDesc *picOutputDesc;
size_t dataSize = (INPUT_WIDTH * INPUT_HEIGHT * 3) / 2;

// 申请一个picOutputDesc，每帧复用，且对Format、width、height、widthStride、heightStride参数值只设置了一次
picOutputDesc = acldvppCreatePicDesc();
ret = acldvppSetPicDescFormat(picOutputDesc, static_cast<acldvppPixelFormat>(1)); // 1：YUV420 semi-planner（nv12）
ret = acldvppSetPicDescWidth(picOutputDesc, 1920);
ret = acldvppSetPicDescHeight(picOutputDesc, 1080);
ret = acldvppSetPicDescWidthStride(picOutputDesc, 1920);
ret = acldvppSetPicDescHeightStride(picOutputDesc, 1080);
// read file to device memory
ReadFileToDeviceMem(filePath.c_str(), inBufferDev, inBufferSize);
while (restLen > 0) {
        // 等待前一个使用picOutputDesc解码帧结束，重新复用picOutputDesc,但没有重新设置Format、width、height、widthStride、heightStride参数值
    // 如果前一帧解码失败，picOutputDesc_的参数Format、width、height、widthStride、heightStride变成默认值0，
    // width、height、widthStride、heightStride为0时，vdec会以实际图片宽高解码输出，但Format为0，表示YUV400格式，vdec不支持解码输出该格式，会导致本帧参数不合法解码失败
        ret = acldvppMalloc(&g_picOutBufferDev, dataSize);
        ret = acldvppSetPicDescData(picOutputDesc, g_picOutBufferDev);
        ret = acldvppSetPicDescSize(picOutputDesc, dataSize);
        ret = aclvdecSendFrame(vdecChannelDesc, streamInputDesc, picOutputDesc, nullptr, nullptr);
        restLen = restLen - 1;
    }
```

12.3.3.14 异常码流导致 VDEC 视频解码失败

问题现象

调用hi_mpi_vdec_get_frame接口成功，返回的解码结果为失败(frame_info->v_frame.frame_flag = 1)，表示可能由于输入码流异常、设置的码流格式与实际码流不一致等问题导致解码失败。

常见语法解析日志报错示例如下：

- 日志示例1
  ```
  pid 0 usr chn 0 device 0 chn 0, input stream error, can't decode, report to user
  ```
- 日志示例2
  ```
  ppssps_check_tmp_id: pps is null with this pic_parameter_set_id = 1 haven't decode
  ```
- 日志示例3
  ```
  PPS or SPS of this slice not valid
  ```
- 日志示例4
  ```
  sliceheader dec err
  ```

- 日志示例5
  ```
  H264DEC inquire_slice_property error
  ```
- 日志示例6
  ```
  hevc_inquire_slice_property error
  ```
- 日志示例7
  ```
  ref frame(poc 15) lost
  ```
- 日志示例8
  ```
  SH hevc_dec_short_term_ref_pic_set error
  ```
- 日志示例9
  ```
  p_temp_r_pset->num_negative_pics(66) out of range(0,15)
  ```

原因分析

frame_flag = 1解码失败，一般是由于送给VDEC视频解码模板的输入码流存在异常，或者网络不稳定导致播包丢帧导致码流异常，所以解码会报错失败，针对这种情况，可以通过保存输入码流来进行判断输入码流是否异常。

解决方法

1. 在调用hi_mpi_vdec_send_stream接口成功之后，将输入码流进行写文件保存，可参考如下代码。
   - 适用于Ascend RC形态：
     ```
     ret = hi_mpi_vdec_send_stream(chn, stream, vdec_pic_info, milli_sec);
     if (ret == HI_SUCCESS) {
         FILE *fd = NULL;
         fd = fopen("input_stream", "a+");
         fwrite(stream->addr, stream->len, 1, fd);
         fclose(fd);
     }
     ```
   - 适用于Ascend EP形态：

     需要先将码流数据从Host传输至Device，再发送解码码流。
     ```
      aclrtMemcpy(stream->addr, stream->len, buf, size, ACL_MEMCPY_HOST_TO_DEVICE);
     ret = hi_mpi_vdec_send_stream(chn, stream, vdec_pic_info, milli_sec);
     if (ret == HI_SUCCESS) {
         FILE *fd = NULL;
         fd = fopen("input_stream", "a+");
         fwrite(buf, size, 1, fd);
         fclose(fd);
     }
     ```
2. 使用第三方工具查看保存下来的input_stream码流，检查是否存在异常，例如花屏、报错提示等。

12.3.3.15 非 annex-B 格式的码流导致 VDEC 视频解码失败

问题现象

解码失败。

日志示例如下：

```
pid 0 usr chn 0 device 0 video format unsupport at event chn 0
```

或

```
pid 0 usr chn 0 device 0 chn 0, input stream error, can't decode, report to user
```

原因分析

VDEC视频解码模块目前只支持annex-B格式的裸码流，不支持其它格式（例如AVCC格式）的裸码流。

解决方法

annex-B格式裸码流，码流样式如下，固定以0x00000001开始（以H264为例），建议用户使用第三方工具打开码流，以16进制方式查看码流并排查格式。

```
<pre><code>0x0000 | 00 00 00 01 67 64 00 0A AC 72 84 44 26 84 00 00
```

12.3.4 JPEGE 图片编码/VENC 视频编码

12.3.4.1 调用错误的内存申请接口，导致内存地址校验出错

现象描述

日志报错如下：

- 日志1
  device:0 chan 0, venc or jpege input buffer is invalid, make sure it has been allocated with hi_mpi_dvpp_malloc or aclDvppMalloc.
- 日志2
  device:0 chan 0, venc or jpege output buffer is invalid, make sure it has been allocated with hi_mpi_dvpp_malloc or aclDvppMalloc.
- 日志3
  device:0 chan 0, jpege input buffer is invalid, make sure it has been allocated with hi_mpi_dvpp_malloc or aclDvppMalloc.
- 日志4
  device:0 chan 0, jpege output buffer is invalid, make sure it has been allocated with hi_mpi_dvpp_malloc or aclDvppMalloc.

可能原因

根据日志提示，是由于没有使用指定的接口申请内存。

处理步骤

使用媒体数据处理V1版本中的**acldvppMalloc**接口/媒体数据处理V2版本中的**hi_mpi_dvpp_malloc**接口申请DVPP内存，存放JPEGE图片编码/VENC视频编码输入或输出数据。

12.3.4.2 内存被提前释放，导致编码数据花屏

现象描述

编码出来的数据花屏，其他无异常日志信息。

可能原因

VENC输入内存是YUV图片数据，输入内存被踩或者被提前释放。

处理步骤

在DVPP内存释放接口处、以及**hi_mpi_venc_get_stream**/**aclvencCallback**/**acldvppJpegEncodeAsync**接口处，增加内存大小及地址的打印日志，确认内存释放时序，是否存在输入内存在编码完成前被提前释放的情况。

12.3.4.3 使用正确的内存申请接口，但内存大小传值错误

现象描述

日志报错如下：

```
device:0 chan 0, output terminal address is invalid, maybe outBufSize:3110400 is invalid.
```

可能原因

传入的buffer size太大，超出了实际申请的buffer范围，导致内部结束地址校验出错。

处理步骤

如果内存申请接口使用正常，业务流程中dvpp内存申请接口增加地址及长度日志，检查接口**hi_mpi_venc_send_frame**/**hi_mpi_venc_send_jpege_frame**/**aclvencSendFrame**/**acldvppJpegEncodeAsync**传入buffer长度是否一致。

12.3.4.4 VENC 创建通道失败

现象描述

调用VENC创建通道的接口hi_mpi_venc_create_chn返回值非0，通道创建失败。

可能原因

导致创建通道失败可能原因有以下：

- 用户传入的通道ID超出了规定的合法范围，VENC规定通道ID在[0,255]总共256个通道。
- 用户传入的通道属性参数不在规定的合理范围内或设置了暂不支持的参数。
- 用户试图创建已经存在的通道，比如开始创建了通道号为0的通道，在这个通道还没销毁的情况下，又去创建通道号为0的通道。

处理步骤

针对分析的可能原因，请参考以下方法处理：

- 首先确定hi_mpi_venc_create_chn接口失败时返回的错误码是多少。
- 如果是0xa0088002，则说明用户传入的通道ID超出了规定的合法范围，需要用户修改通道ID在[0,255]内。
- 如果是0xa0088003或0xa0088008，则说明用户传入的通道属性参数不在规定的合理范围内或设置了暂不支持的参数，

  具体是哪个参数传入有问题可以进一步查看内核日志打印，如下图所示，是传入的分辨率不正确。

  ```
  [Venc]:venc_drv_check_resolution [Line]:342 max picture width (0) err! should in [128,4096]!
  ```

常见原因：1.入参的结构体没有进行memset初始化，导致有些参数如果没有主动设置就会是一些随机值；2.头文件不匹配，导致枚举类型传入和预期不符合; 3.参数支持范围不了解，各个参数的支持范围可以详细查看DVPP对外接口文档。

- 如果是0xa0088004，则说明用户试图创建已经存在的通道，比如开始创建了通道号为0的通道，在这个通道还没销毁的情况下，又去创建通道号为0的通道

  ```
  [Venc]:venc_create_chn [Line]:2449 device:0 chnl:0 had been created!
  ```

  这种情况建议用户排查下通道号管理是否存在问题。

12.3.4.5 VENC 送入待编码图像帧失败

现象描述

调用VENC发送帧的接口hi_mpi_venc_send_frame返回值非0，发送失败。

可能原因

导致发送帧失败可能原因有以下：

- 用户传入的图像帧参数不在规定的合理范围内或设置了暂不支持的参数。
- 用户送帧的频率太快，大于了性能的规格。

处理步骤

针对分析的可能原因，请参考以下方法处理：

- 如果是0xa0088003或0xa0088008，则说明用户传入的图像帧参数不在规定的合理范围内或设置了暂不支持的参数，

  具体是哪个参数传入有问题可以进一步查看内核日志打印，如下图所示，是传入的YUV格式不正确。

  ```
  [Venc]:hevc_check_pixel_format [Line]:1110 H.265 don't support format 5,should be NV12(1) or NV21(2)
  ```

  常见原因：1.入参的结构体没有进行memset初始化，导致有些参数如果没有主动设置就会是一些随机值；2.头文件不匹配，导致枚举类型传入和预期不符合; 3.参数支持范围不了解，各个参数的支持范围可以详细查看DVPP对外接口文档。

- 如果是0xa008800d，则说明VENC的输入空闲队列已满，此时无法再继续往内部送入数据帧，这种问题一般是由于用户送帧的频率太快，大于了芯片处理的速度，导致输入队列堆积，VENC的输入空闲队列长度为6帧，只要堆积到了6帧再继续往里送就会出现这个报错。这种情况建议用户控制调用hi_mpi_venc_send_frame的时间间隔，比如编码帧率30fps，调用的间隔可以控制在33ms一帧。

12.3.4.6 buf_size 参数设置不合理导致视频编码异常

视频编码场景下，需通过**hi_venc_attr**结构体中buf_size参数值来设置编码缓冲区的内存大小，buf_size参数值设置地不合理，可能会导致视频编码耗时长或编码失败。

现象及可能原因

视频编码耗时长或编码失败的场景下，使用dvpp的proc信息排查问题，proc信息中关键信息含义如下：EncStart表示启动编码的帧数，EndSuccessed表示成功编码的帧数，Lost和Disc（Disacrd）表示编码失败的帧数，Recode表示重编的次数。

1. 运行编码进程后，登录Host，在有读、写、执行权限的目录下执行**msnpureport -a**命令，导出Device的日志信息。
2. 按导出时间，进入对应时间戳的目录下，打开“module_info\dev-os-0\dvpp\dvpp_proc.log”日志：
   - 出现类似下面红框的现象：Lost和Disc的数量为0或很低，但是Recode的数量比较大，表示大部分帧能够编码成功，但是重编次数太多。

     当实际的编码结果大小大于编码缓冲区中的可用内存大小时，编码模块会自动调整参数重编，减小编码结果数据大小。因此buf_size设置的太小，缓冲帧数少，导致出现重编的概率高，进而导致编码时延增加，帧率变低，性能下降。

```
[H265E] Version: [HiDVPP]

-----MODULE PARAM----------------------------------------------------------------
      OnePack       H265eVBSource        PowerSaveEn        MiniBufMode       bQpHstgrmEn
            1                   2                  0                  1                 1
-----CHN ATTR-------------------------------------------------------------------
     ID  MaxWidth MaxHeight     Width    Height   Profile   C2GEn   BufSize   ByFrame   GopMode   MaxStrCnt
      1      1920      1200      1920      1200        mp       N   1152000         Y   NormalP         200
      2      3840      2160      3840      2160        mp       N   4147200         Y   NormalP         200
      3      3840      2160      3840      2160        mp       N   4147200         Y   NormalP         200
      4      1920      1200      1920      1200        mp       N   1152000         Y   NormalP         200
      5      1920      1200      1920      1200        mp       N   1152000         Y   NormalP         200
      6      1920      1200      1920      1200        mp       N   1152000         Y   NormalP         200
      7      1920      1200      1920      1200        mp       N   1152000         Y   NormalP         200
      8      1920      1200      1920      1200        mp       N   1152000         Y   NormalP         200
      9      1920      1200      1920      1200        mp       N   1152000         Y   NormalP         200
     10      1920      1200      1920      1200        mp       N   1152000         Y   NormalP         200
     11      1920      1200      1920      1200        mp       N   1152000         Y   NormalP         200

-----PICTURE INFO---------------------------------------------------------------
     ID     EncdStart   EncdSucceed        Lost        Disc       Pskip      Recode      RlsStr     UnrdStr
      1          6597          6594           0           0           0           3        6594           0
      2          6594          6593           0           0           0           0        6593           0
      3          6816          6593           0           0           0         222        6593           0
      4          6597          6594           0           0           0           3        6594           0
      5          6596          6594           0           0           0           2        6594           0
```

   - 出现类似下面红框的现象：Lost和Disc的数量比较大，同时Recode的数量也比较大，表示有比较多的帧编码失败了。

     当实际的编码结果大小大于编码缓冲区中的可用内存大小时，编码模块会自动调整参数重编，减小编码结果数据大小。如果重编次数全部用完，但是编码结果大小依然大于编码缓冲区中的可用内存大小，此时编码模块会将该帧丢弃。因此buf_size设置的太小，缓冲帧数少，导致出现重编的概率高，丢帧概率高。

```
[H265E] Version: [HiDVPP]

-----MODULE PARAM----------------------------------------------------------------
      OnePack       H265eVBSource        PowerSaveEn        MiniBufMode       bQpHstgrmEn
            1                   2                  0                  1                 1
-----CHN ATTR-------------------------------------------------------------------
     ID  MaxWidth MaxHeight     Width    Height   Profile   C2GEn   BufSize   ByFrame   GopMode   MaxStrCnt
      1      1920      1200      1920      1200        mp       N   1152000         Y   NormalP         200
      2      3840      2160      3840      2160        mp       N   4147200         Y   NormalP         200
      3      3840      2160      3840      2160        mp       N   4147200         Y   NormalP         200
      4      1920      1200      1920      1200        mp       N   1152000         Y   NormalP         200
      5      1920      1200      1920      1200        mp       N   1152000         Y   NormalP         200
      6      1920      1200      1920      1200        mp       N   1152000         Y   NormalP         200
      7      1920      1200      1920      1200        mp       N   1152000         Y   NormalP         200
      8      1920      1200      1920      1200        mp       N   1152000         Y   NormalP         200
      9      1920      1200      1920      1200        mp       N   1152000         Y   NormalP         200
     10      1920      1200      1920      1200        mp       N   1152000         Y   NormalP         200
     11      1920      1200      1920      1200        mp       N   1152000         Y   NormalP         200

-----PICTURE INFO---------------------------------------------------------------
     ID     EncdStart   EncdSucceed        Lost        Disc       Pskip      Recode      RlsStr     UnrdStr
      1       4325378       4325372           0           0           0           6     4325372           0
      2       4325372       4325372           0           0           0           0     4325372           0
      3       4325375       4325372           0           0           0           3     4325372           0
      4       4325375       4325372           0           0           0           3     4325372           0
      5       4325373       4325372           0           0           0           1     4325372           0
      6      12976116             0     4325371     4325371           0     8650744     4325371           0
      7       4325377       4325372           0           0           0           5     4325372           0
      8       4325372       4325372           0           0           0           0     4325372           0
      9       4325372       4325372           0           0           0           0     4325372           0
     10       4325372       4325372           0           0           0           0     4325372           0
     11       4325372       4325372           0           0           0           0     4325372           0
```

处理步骤

创建编码通道时，合理设置buf_size参数，具体参见**hi_venc_attr**结构体中buf_size成员的说明。

12.3.4.7 VENC 编码无输出

现象描述

VENC通道创建，发送帧都成功，但调用VENC获取码流的接口hi_mpi_venc_get_stream返回0xa008800e，获取不到编码后码流数据。

可能原因

导致发送帧失败可能原因有以下：

- 用户传入的图像帧内存不是使用dvpp内存申请接口hi_mpi_dvpp_malloc申请的。
- 用户传入的图像帧内存和分辨率不匹配。
- OS内存管理出现问题。

处理步骤

针对分析的可能原因，请参考以下方法处理：

- 查看日志是否有出现看门狗相关的打印，如下图所示，如果出现看门狗，基本可以确认是内存使用存在问题
```
[Chnl]:chnl_watch_dog_timer_isr [Line]:1141 find VEDU_0  down,now reset it
[Chnl]:chnl_watch_dog_blackbox [Line]:1097 vpu_id is 0, venc watchdog fail enter blackbx
```
- 排查用户代码中输入图像帧的内存申请方式，如果不是使用dvpp内存申请接口hi_mpi_dvpp_malloc申请的，VENC芯片将无法正常访问该内存，导致编码无输出，需要修改为使用hi_mpi_dvpp_malloc申请输入内存。
- 排查输入图像帧的内存和分辨率是否不匹配，比如NV12或NV21格式的YUV，一帧图像大小为宽x高x1.5，如果实际送给VENC的内存大小比设置的分辨率宽x高x1.5小的话，会产生访问越界等不可预期的行为，也会导致编码无输出。需要保证申请的内存大小和实际设置的分辨率参数匹配。
- 如果以上排查都不存在问题，可能是OS的内存管理出现问题。您可以获取日志后单击**Link**联系技术支持。

12.4 单算子调用问题

12.4.1 执行单算子产生 coredump 的定位处理

现象描述

单算子执行结束，出现重复释放内存，导致coredump，屏幕显示关键日志信息：

```
double free or corruption ( ! prev )
```

可能原因

分析屏显日志信息，可能存在以下故障原因：代码中出现重复释放内存的操作。

处理步骤

通过gdb挂载可执行文件，通过查看栈信息做排查：

- 重复释放内存代码是否是用户自身代码bug，如果是则需修复代码bug。
- 提供栈信息，联系技术支持。您可以获取日志后单击**Link**联系技术支持。

具体步骤如下：

步骤1 gdb挂载可执行文件。

步骤2 执行gdb调试。

步骤3 查看调用栈。

如果该问题非用户代码问题，需要联系技术支持定位排查。 您可以获取日志后单击**Link**联系技术支持。

----结束

12.4.2 单算子匹配失败

现象描述

单算子执行过程中，出现匹配失败，日志显示如下类似信息。

离线加载执行场景：

```
EH9999 [Match][OpModel]failed to match model, opName = xxx Has not been compiled or loaded, Please make sure the op executed and the op compiled is matched, you can check the op type, op inputs number, outputs number, input format, origin format, datatype, memtype, attr, dim range, and so on.
```

在线加载执行场景：

```
EH9999 [Match][OpModel]MatchOpModel fail from static map or dynamic map. Please make sure the op executed and the op compiled is matched, you can check the op type, op inputs number, outputs number, input format, origin format, datatype, memtype, attr, dim range, and so on.
```

可能原因

- 离线场景：算子om文件未出现在aclSetModelDir中指定的路径下。
- 在线场景：一些特殊的匹配规则未适配，导致算子匹配失败。

处理步骤

- 离线场景：重新编译缺少的算子，并复制到aclSetModelDir中指定的路径下，余下步骤按离线单算子推理步骤进行。
- 在线场景：根据报错信息检查前后两次编译的算子的opType、dataType、format等信息是否一致。

12.4.3 调用 NN 类算子接口超时导致失败

现象描述

调用NN类算子接口时，如果遇到下面两种场景，均属于算子执行超时的表现。

- **场景1**：

单算子执行过程中，Runtime执行报错E39999，在plog日志中Runtime打印了PrintAicpuErrorInfo的错误信息。

plog日志在{install_path}/ascend/log/debug/plog路径下，日志格式为plog-pid_yyymmddhhmmss.log。

– Host侧日志报错样例如下：

```
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.791.865 [engine.cc:1103]16282
ReportExceptProc:Task exception! device_id=0, stream_id=7, task_id=2, type=1, retCode=0x2a,
[aicpu exception].
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.793.489
[device_error_proc.cc:669]16282 ProcessAicpuErrorInfo:report error module_type=0,
module_name=E39999
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.793.498
[device_error_proc.cc:669]16282 ProcessAicpuErrorInfo:An exception occurred during AICPU
execution, stream_id:7, task_id:2, errcode:5, msg:aicpu execute failed.
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.793.932 [task.cc:1050]16282
PreCheckTaskErr:report error module_type=5, module_name=EZ9999
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.793.941 [task.cc:1050]16282
PreCheckTaskErr:Kernel task happen error, retCode=0x2a, [aicpu exception].
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.793.981 [task.cc:759]16282
PrintAicpuErrorInfo:report error module_type=0, module_name=E39999
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.793.990 [task.cc:759]16282
PrintAicpuErrorInfo:Aicpu kernel execute failed, device_id=0, stream_id=7, task_id=2.
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.794.116 [task.cc:777]16282
PrintAicpuErrorInfo:Aicpu kernel execute failed, device_id=0, stream_id=7, task_id=2, flip_num=0,
fault so_name=, fault kernel_name=, fault op_name=Unique, extend_info=(info_type:4,
info_len:6, msg_info:Unique).
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.794.384 [stream.cc:929]16243 GetError:
[EXEC][DEFAULT]Stream Synchronize failed, stream_id=7, retCode=0x2a, [aicpu exception].
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.794.407 [stream.cc:932]16243 GetError:
[EXEC][DEFAULT]report error module_type=0, module_name=E39999
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.794.419 [stream.cc:932]16243 GetError:
[EXEC][DEFAULT]Aicpu kernel execute failed, device_id=0, stream_id=7, task_id=2, flip_num=0,
fault so_name=, fault kernel_name=, fault op_name=Unique, extend_info=(info_type:4,
info_len:6, msg_info:Unique)
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.794.482 [logger.cc:305]16243
StreamSynchronize:[EXEC][DEFAULT]Stream synchronize failed, stream = 0x5643fe3e28d0
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.794.510 [api_c.cc:661]16243
rtStreamSynchronize:[EXEC][DEFAULT]ErrCode=507018, desc=[aicpu exception],
InnerCode=0x715002a
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.794.519
[error_message_manage.cc:49]16243 FuncErrorReason:[EXEC][DEFAULT]report error
module_type=3, module_name=EE8888
[ERROR] RUNTIME(16243,msame):2022-09-22-11:27:01.794.532
[error_message_manage.cc:49]16243 FuncErrorReason:[EXEC][DEFAULT]rtStreamSynchronize
execute failed, reason=[aicpu exception]
```

– Device侧日志报错样例如下：

```
[ERROR] CCECPU(21711,aicpu_scheduler):2024-05-31-20:14:39.806.495
[aicpusd_monitor.cpp:437][HandleTaskTimeout][tid:21724] Send timeout to tsdaemon,
tsdaemon will kill aicpu-sd process, thread index[2], op name[RealDiv], serialNo=279,
stream_id=7, task_id=6812, nowTick:1846897065957, startTick:1845482875742,
timeOut:1400000000, tickFreq:50000000.
```

- **场景2：**

  单算子执行过程中，日志报错提示错误码ErrCode=507017, desc=[aicpu timeout]信息。

可能原因

- 算子的输入/输出Shape太大导致算子执行缓慢。
- 硬件性能较差，不足以支撑算子大量的复杂计算。

处理步骤

步骤1 该类型的错误，可尝试使用**11.15.9 aclrtSetOpExecuteTimeOut**接口，适当调大算子执行的超时时间。

接口原型定义如下:

```
aclError aclrtSetOpExecuteTimeOut(uint32_t timeout)      // timeout单位为秒
```

步骤2 若**步骤1**仍未能解决问题，也可以直接联系技术支持定位排查。您可以获取日志后单击**Link**联系技术支持。

----结束

12.5 编译运行问题

12.5.1 编译运行应用样例报错，提示找不到头文件或库文件

现象描述

获取**10 应用样例参考**章节中应用源码:

- 编译源码时，提示找不到头文件acl.h，报错片段举例如下:
```
 fatal error: acl/acl.h: No such file or directory
 #include "acl/acl.h"
          ^~~~~~~~~~~
compilation terminated.
CMakeFiles/main.dir/build.make:62: recipe for target 'CMakeFiles/main.dir/main.cpp.o' failed
make[2]: *** [CMakeFiles/main.dir/main.cpp.o] Error 1
CMakeFiles/Makefile2:67: recipe for target 'CMakeFiles/main.dir/all' failed
make[1]: *** [CMakeFiles/main.dir/all] Error 2
Makefile:129: recipe for target 'all' failed
```
- 编译源码时，提示找不到库文件libascendcl.so（报错中的-lascendcl，-l表示查找库文件，ascendcl前后分别加上lib和.so组成库文件的名称libascendcl.so），报错片段举例如下:
```
/usr/bin/ld: cannot find -lascendcl
collect2: error: ld returned 1 exit status
CMakeFiles/main.dir/build.make:94: recipe for target '/home/HwHiAiUser/sample/
resnet50_firstapp/out/main' failed
make[2]: *** [/home/HwHiAiUser/sample/resnet50_firstapp/out/main] Error 1
CMakeFiles/Makefile2:67: recipe for target 'CMakeFiles/main.dir/all' failed
make[1]: *** [CMakeFiles/main.dir/all] Error 2
Makefile:129: recipe for target 'all' failed
```

可能原因

编译应用源码依赖定义AscendCL接口的头文件和库文件，**10 应用样例参考**章节中的样例是通过分别配置{DDK_PATH}、{NPU_HOST_LIB}环境变量来查找头文件、库文件，当前报错提示找不到头文件、库文件，则可能是{DDK_PATH}、{NPU_HOST_LIB}环境变量配置地不正确。

处理步骤

步骤1 登录编译源码的环境，分别使用如下命令查看{DDK_PATH}、{NPU_HOST_LIB}环境变量的值。

- 查看{DDK_PATH}环境变量的值：
```
echo $DDK_PATH
```
回显信息示例如下（若无回显信息，则环境变量未配置，跳转到**步骤4**配置该环境变量）：
```
/home/HwHiAiUser/Ascend/ascend-toolkit/latest
```
- 查看{NPU_HOST_LIB}环境变量的值：
```
echo $NPU_HOST_LIB
```
回显信息示例如下（若无回显信息，则环境变量未配置，跳转到**步骤4**配置该环境变量）：
```
/home/HwHiAiUser/Ascend/ascend-toolkit/latest/runtime/lib64/stub
```

步骤2 根据**步骤1**中获取到的{DDK_PATH}环境变量值，检查对应路径下是否存在头文件。

样例中的编译脚本会根据“{DDK_PATH}环境变量值/runtime/include/acl”目录查找编译依赖的头文件，因此可先检查“{DDK_PATH}环境变量值/runtime/include/acl”路径是否存在，若存在，则检查该路径下的acl.h头文件是否存在；若路径或头文件有一个不存在，则需要重新配置{DDK_PATH}环境变量，跳转到**步骤4**。

- 检查路径是否存在，命令示例如下：
```
cd /home/HwHiAiUser/Ascend/ascend-toolkit/latest/runtime/include/acl
```
- 检查acl.h是否存在，命令示例如下：
```
ll acl.h
```

步骤3 根据**步骤1**中获取到的{NPU_HOST_LIB}环境变量值，检查对应路径下是否存在库文件。

样例中的编译脚本会根据{NPU_HOST_LIB}环境变量指向的路径查找编译依赖的库文件，因此可先检查{NPU_HOST_LIB}环境变量指向的路径是否存在，若存在，则检查该路径下的libascendcl.so库文件是否存在；若路径或库文件有一个不存在，则需要重新配置{NPU_HOST_LIB}环境变量，跳转到**步骤4**。

- 检查路径是否存在，命令示例如下：
```
cd /home/HwHiAiUser/Ascend/ascend-toolkit/latest/runtime/lib64/stub
```
- 检查libascendcl.so是否存在，命令示例如下：
```
ll libascendcl.so
```

步骤4 配置环境变量。

编译脚本会根据“{DDK_PATH}环境变量值/runtime/include/acl”目录查找编译依赖的头文件，根据{NPU_HOST_LIB}环境变量指向的目录查找编译依赖的库文件。“$HOME/Ascend”请替换“Ascend-cann-toolkit”包的实际安装路径。

- 当开发环境与运行环境的操作系统架构相同时，配置示例如下所示：
```
export DDK_PATH=$HOME/Ascend/ascend-toolkit/latest
export NPU_HOST_LIB=$DDK_PATH/runtime/lib64/stub
```
- 当开发环境与运行环境的操作系统架构不同时，配置示例如下所示：

  例如，当开发环境为X86架构、运行环境为AArch64架构时，则涉及交叉编译，需在开发环境上安装AArch64架构的软件包，将{DDK_PATH}环境变量的路径指向AArch64架构的软件包安装目录（如下所示），便于使用与运行环境架构相同的软件包中的头文件和库文件来编译代码。
```
export DDK_PATH=$HOME/Ascend/ascend-toolkit/latest/arm64-linux
export NPU_HOST_LIB=$DDK_PATH/runtime/lib64/stub
```

说明

- 您可以登录对应的环境，执行**uname -a**命令查询其操作系统的架构。
- 如果不清楚“Ascend-cann-toolkit”包的安装路径，也可以使用**find -name "*filename*"**命令，查找acl.h、libascendcl.so文件所在的路径，再配置环境变量。

  当环境上安装多个软件版本时，请根据实际情况选择版本，其中，latest目录默认指向最后安装的软件版本。

  当同一个版本下有多个acl.h、libascendcl.so文件时，是由于兼容旧版本的原因，其中部分文件是软链接，方便用户在旧版本下编译应用源码。

----结束

12.5.2 执行应用程序的权限不足导致 AscendCL 初始化报错

问题现象

用户进程报错并退出。

查看应用类日志，提示获取Device信息失败，最终导致AscendCL初始化失败，日志片段示例如下：

```
[ERROR] RUNTIME(89696,main):2023-03-07-17:13:27.994.635 [runtime.cc:1065]89696 CheckHaveDevice:
[INIT][DEFAULT]Call halGetDeviceInfo failed: drvRet=4, module type=0, info type=1.
[ERROR] ASCENDCL(89696,main):2023-03-07-17:13:27.994.723 [acl.cpp:164]89696 aclInit: [INIT][DEFAULT]
[Init][Version]init soc version failed, ret = 507008
[ERROR] RUNTIME(89696,main):2023-03-07-17:13:27.994.774 [api_impl.cc:3490]89696
GetDevErrMsg:report error module_type=3, module_name=EE8888
[ERROR] RUNTIME(89696,main):2023-03-07-17:13:27.994.798 [api_impl.cc:3490]89696 GetDevErrMsg:ctx is
NULL!
[ERROR] RUNTIME(89696,main):2023-03-07-17:13:27.994.827 [api_impl.cc:3546]89696 GetDevMsg:Failed
to GetDeviceErrMsg, retCode=0x7070001.
[ERROR] RUNTIME(89696,main):2023-03-07-17:13:27.994.849 [logger.cc:1348]89696
GetDevMsg:GetDeviceMsg failed, getMsgType=0.
[ERROR] RUNTIME(89696,main):2023-03-07-17:13:27.994.888 [api_c.cc:3595]89696
rtGetDevMsg:ErrCode=107002, desc=[context pointer null], InnerCode=0x7070001
[ERROR] RUNTIME(89696,main):2023-03-07-17:13:27.994.910 [error_message_manage.cc:49]89696
FuncErrorReason:report error module_type=3, module_name=EE8888
[ERROR] RUNTIME(89696,main):2023-03-07-17:13:27.994.932 [error_message_manage.cc:49]89696
FuncErrorReason:rtGetDevMsg execute failed, reason=[context pointer null]
EL0003: The argument is invalid.
        Solution: Try again with a valid argument.
        TraceBack (most recent call last):
        [Init][Version]init soc version failed, ret = 507008[FUNC:ReportInnerError][FILE:log_inner.cpp]
[LINE:145]
        ctx is NULL![FUNC:GetDevErrMsg][FILE:api_impl.cc][LINE:3490]
        rtGetDevMsg execute failed, reason=[context pointer null][FUNC:FuncErrorReason]
[FILE:error_message_manage.cc][LINE:49]

[ERROR] acl init failed
[ERROR] Sample init resource failed
```

原因分析

可能存在以下原因：

- Device状态异常，未正常启动。
- 执行应用程序的用户权限不足，无法查询Device信息。

解决办法

1. 首先，确认Device是否正常启动。
   a. 以root用户登录安装Driver包的环境，执行以下命令查询其安装路径。
   ```
   cat /etc/ascend_install.info
   ```
   在该文件中，Driver_Install_Path_Param表示Driver包的安装路径。
   b. 进入Driver安装路径，使用upgrade-tool工具查看下Device侧运行文件系统版本，如果能正常查询，则说明Device侧已经正常启动。
   ```
   ./upgrade-tool --device_index -1 --system_version
   ```
   正常查询返回信息类似如下：
   ```
   [root@localhost tools]# /usr/local/Ascend/driver/tools/upgrade-tool --device_index -1 --system_version
   {
   Get system version [redacted] ) succeed, deviceId(0)
           {"device_id":0, "version" [redacted]
   Get system version [redacted] ) succeed, deviceId(1)
           {"device_id":1, "version": [redacted]
   }
   ```
2. 其次，检查运行应用程序的用户权限是否正确。
   要求运行应用程序的用户，需与Driver运行用户在一个属组内。在“cat /etc/passwd”文件中，可查看用户属组，Driver的默认运行用户为HwHiAiUser。
   修改用户属组的命令示例如下：
   ```
   usermod -g 组名 用户名
   ```
3. 如果以上方法解决不了问题，则需要参考如下步骤将获取日志，反馈给技术支持。您可以获取日志后单击**Link**联系技术支持。
   a. 登录到运行应用程序的环境，执行如下命令将日志级别设置为Debug。
   ```
   export ASCEND_GLOBAL_LOG_LEVEL=0
   ```
   b. 重新运行应用程序。
   c. 从日志存放路径下获取应用类日志。
   存放日志的默认路径为“$HOME/ascend/log”。
   d. 使用msnpureport工具，获取指定Device上的Debug日志。
   命令示例如下，其中deviceID需要设置为指定Device的ID：
   ```
   msnpureport -g debug -d deviceID
   ```

12.5.3 AscendCL 接口执行无输出无报错

现象描述

AscendCL执行过程中，无接口报错，但是没得到预期结果。

可能原因

- AscendCL执行过程中链接到了stub中的so。
- 异步场景下，在拷贝输出数据时没有做流同步。

处理步骤

针对第一种情况：使用ldd命令查看执行文件链接的so是否正确，保证链接了有效的so。

针对第二种情况：请参考《**CANN AscendCL应用软件开发指南（C&C++）（开放态）**》手册中“AscendCL API参考”章节的“aclmdlExecuteAsync”接口介绍，排查代码。

12.5.4 APP 使用 dvpp 接口编译失败

现象描述

编译提示DVPP的相关接口未定义，编译报错，日志关键字包括：undefined reference to ***

可能原因

分析上述日志信息，可能存在以下故障原因：

DVPP与AscendCL已经分别打包到libacl_dvpp.so与libascendcl.so，测试用例使用了DVPP的相关接口，但没有链接libacl_dvpp.so。

处理步骤

针对分析的可能原因，可以参考下面步骤处理：

排查测试用例是否使用了预处理的接口，但未链接libacl_dvpp.so。如果未链接，需要在编译文件中链接libacl_dvpp.so。

需要排查CmakeLists中的target_link_libraries()选项是否连接了acl_dvpp这个target。

示例：

```
add_executable(main
utils.cpp
main.cpp)
target_link_libraries(main
ascendcl acl_dvpp stdc++)
```

13 附录

使用约束
表达约定
Atlas 200/300/500 推理产品->Atlas 推理系列产品的媒体数据处理接口迁移指引

13.1 使用约束

表 13-1 总体约束列表

分类	约束项
关于低功耗	进入系统休眠前，需要确保将正在运行的AI推理业务、媒体数据处理业务等进程退出。等待唤醒成功后，再继续执行业务。

分类	约束项

关于进程	● 不支持使用fork函数以及封装了fork的函数（如system、posix_spawnp等）创建多个子进程，且在进程中调用AscendCL接口的场景，否则进程运行时会报错或者卡死 。
	● 对于Atlas 200/300/500 推理产品，物理机场景下，一个Device上最多只能支持64个用户进程，Host最多只能支持Device个数*64个进程；虚拟机场景下，一个Device上最多只能支持32个用户进程，Host最多只能支持Device个数*32个进程。
	● 对于Atlas 推理系列产品，物理机场景下，一个Device上最多只能支持64个用户进程，Host最多只能支持Device个数*64个进程；虚拟机场景下，一个Device上最多只能支持32个用户进程，Host最多只能支持Device个数*32个进程。
	● 开放形态下，不建议将进程或线程的优先级设置在[1,99]范围内，否则，可能由于进程或线程会一直占用CPU资源，导致其它系统服务无法得到响应，进而造成Device异常。
	如果必须将进程或线程的优先级设置在[1,99]范围内，则建议该进程或线程必须实时响应、且可以快速完成业务处理，避免长期占用CPU。
	查询进程和线程优先级的命令如下：
	– 查询进程的优先级：（假设进程pid =1407）
	`cat /proc/1407/sched`
	查询结果中，prio字段值表示进程优先级。
	– 查询线程优先级：
	先查询对应线程的ID（查询结果中的SPID字段值）
	`ps -T -p 1047`
	对应线程（假设线程ID为1048）的优先级查询：
	`cat /proc/1407/task/1048/sched`
	查询结果中，prio字段值表示线程优先级。
	说明
	Atlas 200/300/500 推理产品支持开放形态。
	Atlas 推理系列产品支持开放形态。

关于创建类和销毁类接口	● 对于创建类接口（例如：aclrtCreateStream、aclrtCreateEvent、aclCreateDataBuffer等），用户调用该类接口创建对应的资源后，资源使用完成后，建议及时调用对应的销毁类接口（例如：aclrtDestroyStream、aclrtDestroyEvent、aclDestroyDataBuffer等），否则，程序可能会异常。 ● 对于销毁类接口（例如：aclrtDestroyStream、aclrtDestroyEvent、aclrtFree、aclDestroyDataBuffer等），用户调用该类接口后，不能继续使用已释放或销毁的资源，建议用户调用销毁类接口后，将相关资源设置为无效值（例如，置为NULL）。

分类	约束项
关于内存	• 不支持在aclrtMemcpyAsync、aclrtMemsetAsync接口等异步操作内存过程中使用fork以及封装了fork的函数，如system、posix_spawnp等，否则会导致进程运行时会报错，甚至卡死等不可预期的错误。 • 使用AscendCL提供的内存申请接口（例如aclrtMalloc、acldvppMalloc等）申请内存后，为确保内存中不会有脏数据，建议在使用内存前先调用**aclrtMemset**或**aclrtMemsetAsync**接口先清空内存，例如aclrtMemset(devBufferPtr, devBufferSize, 0, devBufferSize)。 • **开放形态**下且配置cgroup时，如果应用程序中涉及aclrtMalloc、acldvppMalloc、hi_mpi_dvpp_malloc等内存申请接口，应用程序在Device上运行时，当前默认在内存不足时，应用程序可能会挂起，等待内存资源，用户可以根据实际需求选择启用操作系统提供的一些配置（例如，enable_oom_killer），这样在内存不足时，应用程序会自动退出，不会一直等待。 若启用enable_oom_killer，您需登录Device，在“/proc/sys/vm”目录下，以root用户启用enable_oom_killer，命令示例如下，1表示启用，0表示禁用： `echo 1 > enable_oom_killer` 基于Memory Cgroup机制实现业务内存资源控制的详细配置请参见《**CANN 软件安装指南 (开放态)**》中的“使用Memory Cgroup机制实现业务内存资源控制”。 **说明** Atlas 200/300/500 推理产品支持开放形态。 Atlas 推理系列产品支持开放形态。
旧版本昇腾AI处理器->新版本昇腾AI处理器的应用迁移	需在迁移后的昇腾AI处理版本上重新转换模型、编译应用程序，否则可能存在应用执行异常的情况。

13.2 表达约定

接口命名规则

接口命名同时满足如下规则：

1. 规则1：acl+*接口类别缩写+操作动词+对象*
2. 规则2：操作动词和对象均采用首字母大写

媒体数据处理V2版本下的接口命名规则例外，这一类接口命名以“hi_mpi”开头。

接口类别

接口类别	缩写	描述
runtime	rt	表示运行管理类的接口。
DVPP	dvpp或vdec或venc	表示媒体数据处理类的接口
AIPP	aipp	表示aipp（AI Preprocessing）类的接口
CBLAS	blas	表示blas类接口
model	mdl	表示模型推理类的接口
graph	grph	表示graph类的接口
driver	drv	表示驱动类的接口
OP	op	表示算子执行类的接口
fv	fv	表示特征向量检索接口
Profiling	prof	表示Profiling配置类接口
tdt	tdt	表示Tensor数据传输接口

注：

1. 缩写原则上不超过4个字母。

2. 在接口命名中，如果类别与操作对象重叠时，操作动词后的对象将省略。

如：aclmdlLoadFromFileWithMem，表示model类接口，这个接口表示含义是load model from file，因此在接口命名中Load后面 mdl将被省略。

变量命名

本文代码示例中涉及的变量，其中，命名带下划线的变量（例如：deviceId_）表示类的私有变量。

13.3 Atlas 200/300/500 推理产品->Atlas 推理系列产品的媒体数据处理接口迁移指引

13.3.1 功能支持度对比列表

本手册中媒体数据处理V1版本与媒体数据处理V2版本的接口都是描述处理媒体数据的接口，用于实现抠图、图片缩放、格式转换等功能，但两套接口不能混用。V2版本的功能比V1版本更多，例如：

- JPEGE：V2版本接口支持高级的参数配置，如huffman表配置。

- VENC：V2版本接口支持更加细化的码控参数配置和效果调优，如I/P帧QP、宏块码控等。
- VDEC：V2版本接口支持更细化的内存控制，如设置输入码流缓存。

建议使用V2版本中的接口，保证后续版本接口功能以及业务的连续演进。

模块	功能或约束	Atlas 200/300/500推理产品媒体数据处理V1接口（接口名以acldvpp开头）	Atlas 推理系列产品媒体数据处理V1接口（接口名以acldvpp开头）	Atlas 推理系列产品媒体数据处理V2接口（接口名以hi_mpi开头）
JPEGE	图片编码成jpeg	支持	支持	支持
	huffman表可配置	不支持	不支持	支持
PNGD	PNG解码	支持	支持	支持
JPEGD	jpeg图片解码成YUV	支持	支持	支持
VENC	h264编码	支持	支持	支持
	h265编码	支持	支持	支持
	码率可调	支持	支持	支持
	帧率可调	支持	支持	支持
	定码率（CBR）	支持	支持	支持
	变码率（VBR）	支持	支持	支持
	强制I帧	支持	支持	支持
	长时间稳定的可变比特率（CVBR）	不支持	不支持	支持
	自适应可变比特率（AVBR）	不支持	不支持	支持
	基于主观图像质量的可变比特率（QVBR）	不支持	不支持	支持
	I帧间隔（GOP）	支持	支持	支持
	场景模式配置	不支持	不支持	支持
VDEC	H264解码	支持	支持	支持
	H265解码	支持	支持	支持
	隔行扫描	支持	支持	支持
	实时出帧	支持	支持	支持
	抽帧	支持	支持	支持

模块	功能或约束	Atlas 200/300/500 推理产品媒体数据处理V1接口（接口名以acldvpp开头）	Atlas 推理系列产品媒体数据处理V1接口（接口名以acldvpp开头）	Atlas 推理系列产品媒体数据处理V2接口（接口名以hi_mpi开头）
	解码缩放	支持	支持	支持
	输出RGB888	不支持	支持	支持
VPC	抠图	支持	支持	支持
	缩放	支持	支持	支持
	填充功能	不支持	支持	支持
	抠图缩放	支持	支持	支持
	抠图缩放贴图	支持	支持	支持
	抠图缩放填充	不支持	支持	支持
	一图多框贴图	支持	支持	支持
	一图多框填充	不支持	支持	支持
	多图多框贴图	支持	支持	支持
	多图多框填充	不支持	支持	支持
	色域转换（CSC）	支持	支持	支持
	色域转换（CSC）系数配置	不支持	支持	支持
	金字塔	不支持	支持	支持
	直方图均衡（LUT）	不支持	支持	支持
	直方图统计	不支持	支持	支持

13.3.2 Atlas 200/300/500 推理产品媒体数据处理 V1->Atlas 推理系列产品媒体数据处理 V1 迁移指引

13.3.2.1 使用场景说明

媒体数据处理V1版本接口（接口名以acldvpp开头）从Atlas 200/300/500 推理产品迁移到Atlas 推理系列产品上，由于是同一套接口在不同昇腾处理器上执行，接口的调用逻辑、调用示例基本一致，头文件、库文件不涉及修改，但迁移时存在一些兼容性的功能，本节仅介绍**涉及兼容性的功能列表**，更多功能及接口说明请参见如下章节。

- 接口调用流程及示例在“图像/视频数据处理>**媒体数据处理V1**”章节。

- 接口说明在“AscendCL API参考>**媒体数据处理V1**”章节。

13.3.2.2 兼容性说明

功能或约束	涉及的AscendCL接口	Atlas 200/300/500 推理产品的实现	Atlas 推理系列产品的实现	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
设置VENC输出buffer。	aclvencSetChannelDescBufAddr接口 aclvencSetChannelDescParam接口	用户不需要设置输出buffer，回调时进行一次内存拷贝，拷贝至用户输出buffer。	需要用户设置输出buffer，编码输出结果**无需多一次拷贝操作**。	需要调用aclvencSetChannelDescBufAddr接口或aclvencSetChannelDescParam接口显式设置输出buffer。

13.3.2.3 优化建议

功能或约束	涉及的AscendCL接口	Atlas 200/300/500 推理产品的实现	Atlas 推理系列产品的实现	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
VPC功能中，设置缩放算法。	acldvppSetResizeConfigInterpolation接口	支持5种算法： ● 0：默认值。表示自研的高阶滤波算法。 ● 1：业界通用的Bilinear算法（与OpenCV算法的计算过程类似） ● 2：业界通用的Nearest neighbor算法（与OpenCV算法的计算过程类似） ● 3：业界通用的Bilinear算法（与Tensorflow算法的计算精过程类似）。 ● 4：业界通用的Nearest neighbor算法（与Tensorflow算法的计算精过程类似）。	支持2种算法： ● 0：默认值。设置为0时，系统内部也会自动采用1。 ● 1：业界通用的Bilinear算法（与OpenCV算法的计算过程类似） ● 2：业界通用的Nearest neighbor 算法（与OpenCV算法的计算过程类似）	如果用户没有显式调用该接口，迁移到Atlas 推理系列产品后，无需修改代码，直接使用Atlas 推理系列产品上默认的缩放算法，**精度更高**。 如果用户显式调用该接口将缩放算法设置为1或2，则无需修改代码。 如果用户显式调用该接口将缩放算法设置为3或4，则用户修改代码，设置Atlas 推理系列产品所支持的缩放算法。修改代码后，需要重新编译。

功能或约束	涉及的AscendCL接口	Atlas 200/300/500 推理产品的实现	Atlas 推理系列产品的实现	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
VPC功能中，YUV400格式图像处理	acldvppSetPicDescSize接口	Atlas 200/300/500 推理产品不支持YUV400格式图像处理，使用YUV420格式进行处理(取Y分量数据)，因此需要将输入图片的格式设置为YUV420，内存大小设置为不小于widthStride*heightStride*3/2的值，VPC会根据YUV420图片格式校验内存大小。	Atlas 推理系列产品支持YUV400格式图像处理，因此直接将输入图片的格式设置为YUV400，内存大小设置为不小于widthStride*heightStride的值，VPC会根据YUV400图片格式校验内存大小。	迁移到Atlas 推理系列产品后，用户可以继续使用“从YUV420格式中取Y分量”来实现YUV400的方式，这样，就不涉及修改代码，也无需重新编译。 另外，由于在Atlas 推理系列产品上扩展支持了YUV400格式，所以如果用户选择直接使用YUV400格式，就需要修改代码，将输出格式设置为YUV400，将内存大小设置为不小于“widthStride*heightStride”的值，**节省内存**。修改代码后，用户需要重新编译。

功能或约束	涉及的AscendCL接口	Atlas 200/300/500 推理产品的实现	Atlas 推理系列产品的实现	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
VDEC功能中，设置解码码流最大宽度和高度。	aclvdecSetChannelDescOutPicWidth接口 aclvdecSetChannelDescOutPicHeight接口	● 解码图像帧存大小：创建通道时不会使用解码码流最大宽度和高度申请内部使用的帧存，会在解码过程中根据码流信息获取码流的真实宽高，申请帧存。 ● 输入码流缓存大小：1080P分辨率及以下的输入码流，输入码流缓存大小默认为6M；1080P分辨率以上的输入码流，输入码流缓存大小默认为9M。	● 解码图像帧存大小：创建通道时不会使用解码码流最大宽度和高度申请内部使用的帧存，会在解码过程中根据码流信息获取码流的真实宽高，申请帧存。 ● 输入码流缓存大小：解码码流最大宽度*解码码流最大高度*2。	迁移到Atlas 推理系列产品后，用户无需修改代码，系统内部会自行根据解码码流的信息申请帧存。 另外，在Atlas 推理系列产品上，系统内部申请的帧存大小需在一个最大值的范围内，该最大值与用户设置的解码码流最大宽度和高度、参考帧数量、码流位宽参数有关，帧存最大值 ≈ (*最大宽度* * *最大高度* * 3/2) * *码流位宽* / 8 * (*参考帧数量* + 3) 建议用户迁移时合理规划VDEC内存，详细描述请参见**性能指标说明（Atlas 推理系列产品）**。

功能或约束	涉及的AscendCL接口	Atlas 200/300/500 推理产品的实现	Atlas 推理系列产品的实现	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
VDEC功能中，设置参考帧数量。	aclvdecSetChannelDescRefFrameNum接口	不支持设置参考帧。	支持设置参考帧，如果不设置，默认参考帧数量是8。 在创建通道时，会根据默认值申请帧存，但在解码过程中，会根据码流信息获取码流的真实参考帧信息，进行帧存自适应。	
VENC功能中，指定输出码率。	aclvencSetChannelDescMaxBitRate接口 aclvencSetChannelDescParam接口	输出码率默认值为300。	输出码率默认值为2000。	如果用户没有显式调用接口设置输出码率，迁移到Atlas 推理系列产品后，也无需修改代码，直接使用Atlas 推理系列产品上默认的输出码率值2000，**画质更优**。 如果用户需要调整画质，则需根据实际情况设置输出码率。修改代码后，需要重新编译。 要注意，输出码率不一样，编码输出码流大小会不一样。

功能或约束	涉及的AscendCL接口	Atlas 200/300/500 推理产品的实现	Atlas 推理系列产品的实现	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
设置通道模式。 （提醒：明确图片数据处理通道用于实现哪种功能（目前支持VPC、JPEGD、JPEGE、PNGD功能），若不调用该接口，则系统默认先创建VPC模式的通道，可能会占用VPC通道数，推荐用户根据实际功能指定通道模式。）	acldvppSetChannelDescMode接口	不支持设置通道模式。	支持设置通道模式，如果不设置通道模式，为了减少通道和内存资源的浪费，会默认先创建VPC模式的通道（会占用VPC通道数）。	迁移到Atlas 推理系列产品后，用户可以不调用该接口设置通道模式，就不涉及修改代码，也无需重新编译。此时，系统内部为了减少通道和内存资源的浪费，默认先创建VPC模式的通道（会占用VPC通道数），在用户调用JPEGD或JPEGE或PNGD功能的接口时，会触发系统内部自动创建JPEGD或JPEGE或PNGD模式的通道。 建议用户根据实际业务，调用该接口设置对应的通道模式，减少VPC通道的占用和内存资源的浪费。

13.3.3 Atlas 200/300/500 推理产品媒体数据处理 V1->Atlas 推理系列产品媒体数据处理 V2 迁移指引

13.3.3.1 使用场景说明

从Atlas 200/300/500 推理产品的AscendCL媒体数据处理V1版本接口（接口名以acldvpp或aclvdec或aclvenc开头）迁移到Atlas 推理系列产品的AscendCL媒体数据处理V2版本接口（接口名以hi_mpi开头）的场景下，由于接口的调用逻辑、调用示例，差别较大，为方便用户迁移，本节仅根据**常用的关键功能点**列举了V1、V2版本之间接口的对应关系，供迁移的开发人员参考，更多功能及接口说明请参见下表中的相关章节。

表 13-2 应用开发指南中的相关章节说明

媒体数据处理V1版本接口相关章节	媒体数据处理V2版本接口相关章节
接口调用流程及示例在“图像/视频数据处理>**媒体数据处理V1**”章节。	接口调用流程在“图像/视频数据处理>**媒体数据处理V2**”章节。
接口说明在“AscendCL API参考>**媒体数据处理V1**”章节。	接口说明在“AscendCL API参考>**媒体数据处理V2**”章节

13.3.3.2 头文件迁移

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
CANN软件安装目录/include/acl/ops/acl_dvpp.h	CANN软件安装目录/include/acl/dvpp下的头文件	用户需要修改代码，include“CANN软件安装目录/include/acl/dvpp”目录下的hi_dvpp.h文件。

13.3.3.3 库文件迁移

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
CANN软件安装目录/lib64/libacl_dvpp.so	CANN软件安装目录/lib64/libacl_dvpp_mpi.so	用户需要修改编译应用程序的脚本，修改依赖的库文件，改成libacl_dvpp_mpi.so。

13.3.3.4 VPC 功能迁移

VPC功能，命名以acldvpp开头的接口与命名以hi_mpi开头的接口，最大的区别在于：用acldvpp接口实现VPC功能时，设置输入、输出图片信息，都是通过set接口，图片处理结束后，再通过get接口获取结果数据所在的内存地址；用hi_mpi命名的接口实现VPC功能时，设置输入、输出图片信息都是通过给结构体中的成员变量赋值，图片处理结束后，从对应的内存地址中获取结果数据。

13.3.3.4.1 媒体数据处理模块初始化

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
无需进行媒体数据处理模块初始化	hi_mpi_sys_init	用户需要修改代码，增加调用媒体数据处理模块初始化的接口hi_mpi_sys_init。

13.3.3.4.2 创建通道

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi 接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
创建通道描述信息的结构体： acldvppCreateChannelDesc	无对应接口，直接使用hi_vpc_chn_attr结构体，且该结构体参数内部保留，用户配置任意值（除空指针）即可。	用户需要修改代码，直接声明hi_vpc_chn_attr结构体的变量。
创建通道： acldvppCreateChannel	创建通道： hi_mpi_vpc_create_chn	用户需要修改代码，改用hi_mpi_vpc_create_chn接口创建通道。

13.3.3.4.3 内存申请

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi 接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
申请内存： acldvppMalloc	申请内存： hi_mpi_dvpp_malloc	用户需要修改代码，改用hi_mpi_dvpp_malloc接口申请媒体数据处理功能需要的内存。
释放内存 acldvppFree	释放内存： hi_mpi_dvpp_free	用户需要修改代码，改用hi_mpi_dvpp_free接口释放内存。

13.3.3.4.4 VPC 抠图缩放贴图功能（一图一框）

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
配置输入图片信息： acldvppCreatePicDesc acldvppSetPicDescData acldvppSetPicDescSize acldvppSetPicDescWidth acldvppSetPicDescHeight acldvppSetPicDescWidthStride acldvppSetPicDescHeightStride acldvppSetPicDescFormat acldvppDestroyPicDesc	无对应接口，配置输入图片参数时，直接对hi_vpc_pic_info结构体的成员赋值： picture_address; picture_buffer_size; picture_width; picture_height; picture_width_stride; picture_height_stride; picture_format;	用户需要修改代码，对结构体hi_vpc_pic_info成员赋值来配置输入图片信息的参数。
配置输出图片信息： acldvppCreatePicDesc acldvppSetPicDescData acldvppSetPicDescSize acldvppSetPicDescWidth acldvppSetPicDescHeight acldvppSetPicDescWidthStride acldvppSetPicDescHeightStride acldvppSetPicDescFormat acldvppDestroyPicDesc	无对应接口，配置输出图片参数时，直接对hi_vpc_crop_resize_paste_region结构体内的dest_pic_info成员赋值，dest_pic_info成员是hi_vpc_pic_info结构体，该结构体内的成员如下： picture_address; picture_buffer_size; picture_width; picture_height; picture_width_stride; picture_height_stride; picture_format;	用户需要修改代码，改为对hi_vpc_crop_resize_paste_region结构体内的dest_pic_info成员赋值。

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
配置从输入图片中的抠图位置信息： acldvppCreateRoiConfig acldvppSetRoiConfig acldvppSetRoiConfigLeft acldvppSetRoiConfigRight acldvppSetRoiConfigTop acldvppSetRoiConfigBottom acldvppDestroyRoiConfig	无对应接口，配置抠图位置信息时，直接对hi_vpc_crop_resize_paste_region结构体内的crop_region成员赋值，crop_region成员是hi_vpc_crop_region结构体，该结构体内的成员如下： top_offset; left_offset; crop_width; crop_height;	用户需要修改代码，改为对hi_vpc_crop_resize_paste_region结构体内的crop_region成员赋值。
配置缩放算法信息： acldvppCreateResizeConfig acldvppSetResizeConfigInterpolation acldvppDestroyResizeConfig	无对应接口，配置缩放信息时，直接对hi_vpc_crop_resize_paste_region结构体内的resize_info成员赋值，resize_info成员是hi_vpc_resize_info结构体，该结构体内的成员如下： resize_width; resize_height; interpolation;	用户需要修改代码，改为对hi_vpc_crop_resize_paste_region结构体内的resize_info成员赋值。

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
配置输出图片中的贴图位置信息： acldvppCreateRoiConfig acldvppSetRoiConfig acldvppSetRoiConfigLeft acldvppSetRoiConfigRight acldvppSetRoiConfigTop acldvppSetRoiConfigBottom acldvppDestroyRoiConfig	无对应接口，配置贴图位置信息时，直接对hi_vpc_crop_resize_paste_region结构体内的dest_top_offset成员、dest_left_offset成员赋值。	用户需要修改代码，改为对hi_vpc_crop_resize_paste_region结构体内的dest_top_offset成员、dest_left_offset成员赋值。
抠图缩放贴图功能：acldvppVpcCropResizePasteAsync	抠图缩放贴图功能： hi_mpi_vpc_crop_resize_paste	用户需要修改代码，改成调用hi_mpi_vpc_crop_resize_paste接口。 注意：hi_mpi_vpc_crop_resize_paste接口支持一图一框和一图多框，如果想使用一图多框功能，则需要配置每个框的抠图位置信息、缩放算法信息、贴图位置信息。

13.3.3.4.5 VPC 批量抠图缩放贴图功能

Atlas 200/300/500 推理产品 acldvpp 接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
配置输入batch的图片信息： acldvppCreateBatchPicDesc acldvppGetPicDesc acldvppSetPicDescData acldvppSetPicDescSize acldvppSetPicDescWidth acldvppSetPicDescHeight acldvppSetPicDescWidthStride acldvppSetPicDescHeightStride acldvppSetPicDescFormat acldvppDestroyBatchPicDesc	无对应接口，配置输入图片参数时，直接对hi_vpc_pic_info结构体数据的成员赋值，其中数组的每一个元素代表每一个输入图片，pic_num代表batch大小： picture_address; picture_buffer_size; picture_width; picture_height; picture_width_stride; picture_height_stride; picture_format;	用户需要修改代码，对结构体hi_vpc_pic_info成员赋值来配置输入图片信息的参数。
配置每个输入图片的抠图数量： roiNums size	数据结构对应如下，数组长度跟输入pic_num长度一致： count[]	用户需要修改代码，改为对hi_vpc_crop_resize_paste_region结构体内的dest_pic_info成员赋值。

Atlas 200/300/500 推理产品 acldvpp 接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
配置输出batch的图片信息： acldvppCreateBatchPicDesc acldvppGetPicDesc acldvppSetPicDescData acldvppSetPicDescSize acldvppSetPicDescWidth acldvppSetPicDescHeight acldvppSetPicDescWidthStride acldvppSetPicDescHeightStride acldvppSetPicDescFormat acldvppDestroyBatchPicDesc	无对应接口，配置输出图片参数时，直接对hi_vpc_crop_resize_paste_region结构体内的dest_pic_info成员赋值，dest_pic_info成员是hi_vpc_pic_info结构体，该结构体内的成员如下： picture_address; picture_buffer_size; picture_width; picture_height; picture_width_stride; picture_height_stride; picture_format;	用户需要修改代码，改为对hi_vpc_crop_resize_paste_region结构体内的dest_pic_info成员赋值。
配置从输入图片中的抠图位置信息： acldvppCreateRoiConfig acldvppSetRoiConfig acldvppSetRoiConfigLeft acldvppSetRoiConfigRight acldvppSetRoiConfigTop acldvppSetRoiConfigBottom acldvppDestroyRoiConfig	无对应接口，配置抠图位置信息时，直接对hi_vpc_crop_resize_paste_region结构体内的crop_region成员赋值，crop_region成员是hi_vpc_crop_region结构体，该结构体内的成员如下： top_offset; left_offset; crop_width; crop_height;	用户需要修改代码，改为对hi_vpc_crop_resize_paste_region结构体内的crop_region成员赋值。

Atlas 200/300/500 推理产品 acldvpp 接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
配置缩放算法信息： acldvppCreateResizeConfig acldvppSetResizeConfigInterpolation acldvppDestroyResizeConfig	无对应接口，配置缩放信息时，直接对hi_vpc_crop_resize_paste_region结构体内的resize_info成员赋值，resize_info成员是hi_vpc_resize_info结构体，该结构体内的成员如下： resize_width; resize_height; interpolation;	用户需要修改代码，改为对hi_vpc_crop_resize_paste_region结构体内的resize_info成员赋值。
配置输出图片中的贴图位置信息： acldvppCreateRoiConfig acldvppSetRoiConfig acldvppSetRoiConfigLeft acldvppSetRoiConfigRight acldvppSetRoiConfigTop acldvppSetRoiConfigBottom acldvppDestroyRoiConfig	无对应接口，配置贴图位置信息时，直接对hi_vpc_crop_resize_paste_region结构体内的dest_top_offset成员、dest_left_offset成员赋值。	用户需要修改代码，改为对hi_vpc_crop_resize_paste_region结构体内的dest_top_offset成员、dest_left_offset成员赋值。
批量抠图缩放贴图功能： acldvppVpcBatchCropResizePasteAsync	批量抠图缩放贴图功能： hi_mpi_vpc_batch_crop_resize_paste	用户需要修改代码，改成调用hi_mpi_vpc_batch_crop_resize_paste接口。 需注意：原始图片数量pic_num，与source_pic数组长度、count数组长度保持一致。

13.3.3.4.6 VPC 缩放功能

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
配置输入图片信息： acldvppCreatePicDesc acldvppSetPicDescData acldvppSetPicDescSize acldvppSetPicDescWidth acldvppSetPicDescHeight acldvppSetPicDescWidthStride acldvppSetPicDescHeightStride acldvppSetPicDescFormat acldvppDestroyPicDesc	无对应接口，配置输入图片参数时，直接对source_pic赋值，source_pic是hi_vpc_pic_info结构体，该结构体内的成员如下： picture_address; picture_buffer_size; picture_width; picture_height; picture_width_stride; picture_height_stride; picture_format;	用户需要修改代码，改为对source_pic赋值。
配置输出图片信息： acldvppCreatePicDesc acldvppSetPicDescData acldvppSetPicDescSize acldvppSetPicDescWidth acldvppSetPicDescHeight acldvppSetPicDescWidthStride acldvppSetPicDescHeightStride acldvppSetPicDescFormat acldvppDestroyPicDesc	无对应接口，配置输出图片参数时，直接对dest_pic赋值，dest_pic是hi_vpc_pic_info结构体，该结构体内的成员如下： picture_address; picture_buffer_size; picture_width; picture_height; picture_width_stride; picture_height_stride; picture_format;	用户需要修改代码，改为对dest_pic赋值。

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
配置缩放算法信息： acldvppCreateResizeConfig acldvppSetResizeConfigInterpolation acldvppDestroyResizeConfig	无对应接口，直接对interpolation进行赋值	用户需要修改代码，改为对interpolation赋值。
缩放功能： acldvppVpcResizeAsync	缩放功能： hi_mpi_vpc_resize	用户需要修改代码，改成调用hi_mpi_vpc_resize接口。

13.3.3.4.7 获取处理结果

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
等待任务处理完成后，从接口acldvppVpcCropResizePasteAsync的输出参数outputDesc中调用对应的get接口获取输出图片数据。 ● 等待任务完成接口如下：aclrtSynchronizeStream ● 对应的get接口如下：acldvppGetPicDescData acldvppGetPicDescWidth acldvppGetPicDescHeight acldvppGetPicDescWidthStride acldvppGetPicDescHeightStride acldvppGetPicDescFormat	等待任务处理完成后，获取输出图片数据。 ● 等待任务完成的接口如下：hi_mpi_vpc_get_process_result ● 针对不同功能，获取输出图片数据的参数如下： 使用hi_mpi_vpc_crop_resize_paste接口实现抠图缩放贴图功能时，从hi_vpc_crop_resize_paste_region.dest_pic_info.picture_address参数获取输出图片数据。 使用hi_mpi_vpc_batch_crop_resize_paste接口实现批量抠图缩放贴图功能时，从hi_vpc_crop_resize_paste_region.dest_pic_info.picture_address参数获取输出图片数据。 使用hi_mpi_vpc_resize接口实现缩放功能时，从dest_pic.picture_address参数获取输出图片数据。	用户需要修改代码，将aclrtSynchronizeStream接口修改成hi_mpi_vpc_get_process_result接口，并改为直接从对应的成员参数中获取输出图片数据。

13.3.3.4.8 销毁通道

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
销毁通道： acldvppDestroyChannel	销毁通道： hi_mpi_vpc_destroy_chn	用户需要修改代码，改为调用hi_mpi_vpc_destroy_chn接口。

13.3.3.4.9 媒体数据处理模块去初始化

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
无需进行媒体数据处理模块去初始化	hi_mpi_sys_exit	用户需要修改代码，增加调用媒体数据处理模块去初始化的接口hi_mpi_sys_exit。

13.3.3.5 VDEC 功能迁移

VDEC功能，命名以acldvpp或aclvdec开头的接口与命名以hi_mpi开头的接口，最大的区别在于：用acldvpp接口实现VDEC功能时，需设置回调函数，每解码一帧数据会自动调用回调函数，在回调函数内调用对应acldvpp的Get接口获取解码数据的内存地址、内存大小等；用hi_mpi命名的接口实现VDEC功能时，需要用户另起线程调用hi_mpi_vdec_get_frame接口来获取解码结果。

13.3.3.5.1 媒体数据处理模块初始化

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
无需进行媒体数据处理模块初始化	hi_mpi_sys_init	用户需要修改代码，增加调用媒体数据处理模块初始化的接口hi_mpi_sys_init。

13.3.3.5.2 创建通道

Atlas 200/300/500 推理产品 aclvdec接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
创建通道描述信息的结构体： aclvdecCreateChannelDesc	无对应接口，直接使用hi_vdec_chn_attr结构体：	用户需要修改代码，直接声明hi_vdec_chn_attr结构体的变量。

Atlas 200/300/500 推理产品 aclvdec接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
设置通道描述的参数： aclvdecSetChannelDescEnType aclvdecSetChannelDescOutPicWidth aclvdecSetChannelDescOutPicHeight aclvdecSetChannelDescRefFrameNum aclvdecSetChannelDescChannelId（仅acl有） aclvdecSetChannelDescThreadId（仅acl有） aclvdecSetChannelDescCallback（仅acl有） aclvdecSetChannelDescOutPicFormat（仅acl有） aclvdecSetChannelDescOutMode（仅acl有）	无对应接口，直接对hi_vdec_chn_attr结构体的成员赋值： hi_vdec_chn_attr.type hi_vdec_chn_attr.pic_width hi_vdec_chn_attr.pic_height hi_vdec_chn_attr.video_attr.ref_frame_num hi_vdec_chn_attr.mode（仅hi_mpi有） hi_vdec_chn_attr.stream_buf_size（仅hi_mpi有） hi_vdec_chn_attr.frame_buf_cnt（仅hi_mpi有） hi_vdec_chn_attr.frame_buf_size（仅hi_mpi有） hi_vdec_chn_attr.video_attr.temporal_mvp_en（仅hi_mpi有） hi_vdec_chn_attr.video_attr.tmv_buf_size（仅hi_mpi有）	用户需要修改代码，对结构体hi_vdec_chn_attr成员赋值来配置通道描述的参数。 需注意，对于仅acl有的这几个参数设置，在hi_mpi上也有对应的接口： ● 通道ID，不在hi_vdec_chn_attr结构体中，而是在用hi_mpi_vdec_create_chn接口创建通道时传入。 ● 线程ID、Callback（这两个跟回调函数相关），不在hi_vdec_chn_attr结构体中，因为使用hi mpi接口时不需要设置回调函数，需要自行另起线程，调用hi_mpi_vdec_get_frame接口来获取解码结果。 ● 输出图片格式outPicFormat参数不在hi_vdec_chn_attr结构体内，是在调用hi_mpi_vdec_send_stream接口发送码流数据时通过hi_vdec_pic_info.pixel_format参数指定。 ● 设置是否实时出帧outMode参数（即发送一帧解码一帧，无需依赖后续帧的传入）不在hi_vdec_chn_attr结构体内，是在调用hi_mpi_vdec_set_chn_param接口设置通道属性时通过hi_vdec_chn_param.hi_vdec_video_param.hi_video_out_order参

Atlas 200/300/500 推理产品 aclvdec接口	Atlas 推理系列产品 hi_mpi 接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
		数指定的，其中，解码序表示快速出帧。
无对应接口	设置通道属性： hi_mpi_vdec_set_chn_param	设置解码出帧模式时，需要调用hi_mpi_vdec_set_chn_param接口进行设置。
创建通道 aclvdecCreateChannel	创建通道 hi_mpi_vdec_create_chn	用户需要修改代码，改用hi_mpi_vdec_create_chn接口创建通道。

13.3.3.5.3 内存申请

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi 接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
申请内存： acldvppMalloc	申请内存： hi_mpi_dvpp_malloc	用户需要修改代码，改用hi_mpi_dvpp_malloc接口申请媒体数据处理功能需要的内存。
释放内存 acldvppFree	释放内存： hi_mpi_dvpp_free	用户需要修改代码，改用hi_mpi_dvpp_free接口释放内存。

13.3.3.5.4 发送码流

Atlas 200/300/500 推理产品 acldvpp/aclvdec接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
配置输入码流信息： acldvppSetStreamDescData acldvppSetStreamDescSize acldvppSetStreamDescEos	无对应接口，配置输入码流参数时，直接对hi_vdec_stream结构体的成员赋值： hi_vdec_stream.addr hi_vdec_stream.len hi_vdec_stream.end_of_stream	用户需要修改代码，对结构体hi_vdec_chn_attr成员赋值来配置输入码流信息的参数。 hi_mpi接口需要在所有码流数据发送结束后主动发送eos帧，将hi_vdec_stream.end_of_stream设置为HI_TRUE，其他码流发送时将hi_vdec_stream.end_of_stream设置为HI_FALSE。
配置输出图片信息： acldvppSetPicDescData acldvppSetPicDescSize acldvppSetPicDescFormat acldvppSetPicDescWidth acldvppSetPicDescHeight acldvppSetPicDescWidthStride acldvppSetPicDescHeightStride	无对应接口，配置输出图片参数时，直接对hi_vdec_pic_info结构体的成员赋值： hi_vdec_pic_info.vir_addr hi_vdec_pic_info.buffer_size hi_vdec_pic_info.pixel_format hi_vdec_pic_info.width hi_vdec_pic_info.height hi_vdec_pic_info.width_stride hi_vdec_pic_info.height_stride hi_vdec_stream.need_display（仅hi_mpi有）	用户需要修改代码，对结构体hi_vdec_pic_info成员赋值来配置输出图片信息的参数。 需注意，acl接口中，如果不想获取某一帧的解码结果，通过调用aclvdecSendSkippedFrame接口实现，但在hi_mpi接口中，是通过设置hi_vdec_stream.need_display来完成。
无对应接口，无需通知解码器开始接收码流数据	通知解码器开始接收码流数据（一个通道只需要调用一次该接口）： hi_mpi_vdec_start_recv_stream	用户需要修改代码，增加调用hi_mpi_vdec_start_recv_stream接口。

Atlas 200/300/500 推理产品 acldvpp/aclvdec接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
aclvdecSendSkippedFrame	hi_vdec_stream.need_display（仅hi_mpi有）	acl接口中，如果不想获取某一帧的解码结果，通过调用aclvdecSendSkippedFrame接口实现，但在hi_mpi接口中，是通过设置hi_vdec_stream.need_display来完成。
每一帧都需调用以下接口发送解码码流： aclvdecSendFrame	每一帧都需调用以下接口发送解码码流： hi_mpi_vdec_send_stream	用户需要修改代码，改为调用hi_mpi_vdec_send_stream接口发送码流，同时需要传入通道号。

13.3.3.5.5 接收码流

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
设置通道描述信息时已注册回调函数，每解码一帧数据会自动调用回调函数，在回调函数内调用对应的Get接口获取解码数据的内存地址、内存大小等。 acldvppGetPicDescData acldvppGetPicDescWidth acldvppGetPicDescHeight acldvppGetPicDescWidthStride acldvppGetPicDescHeightStride acldvppGetPicDescFormat acldvppGetPicDescRetCode（仅acl有）	调用hi_mpi_vdec_get_frame接口获取一帧解码结果，再从hi_video_frame_info结构体的以下成员中获取解码数据： hi_video_frame_info.v_frame.virt_addr[0] hi_video_frame_info.v_frame.width hi_video_frame_info.v_frame.height hi_video_frame_info.v_frame.width_stride[0] hi_video_frame_info.v_frame.height_stride[0] hi_video_frame_info.v_frame.pixel_format hi_video_frame_info.v_frame.frame_flag（与retCode对应）	用户需要修改代码，改为另起线程，调用hi_mpi_vdec_get_frame接口来主动获取解码结果。

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
无对应接口，无需通知解码器停止接收码流数据	通知解码器停止接收码流数据（一个通道只需要调用一次该接口）： hi_mpi_vdec_stop_recv_stream	用户需要修改代码，增加调用hi_mpi_vdec_stop_recv_stream接口。

13.3.3.5.6 销毁通道

Atlas 200/300/500 推理产品 aclvdec接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
销毁通道： aclvdecDestroyChannel	销毁通道： hi_mpi_vdec_destroy_chn	用户需要修改代码，改为调用hi_mpi_vdec_destroy_chn接口。 需注意，在调用hi_mpi接口销毁通道前，可以先调用hi_mpi_vdec_query_status接口获取解码器状态，判断是否所有码流数据都已完成解码，然后进行通道的销毁。

13.3.3.5.7 媒体数据处理模块去初始化

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
无需进行媒体数据处理模块去初始化	hi_mpi_sys_exit	用户需要修改代码，增加调用媒体数据处理模块去初始化的接口hi_mpi_sys_exit。

13.3.3.6 JPEGD 功能迁移

JPEGD功能，命名以acldvpp开头的接口与命名以hi_mpi开头的接口，最大的区别在于：用acldvpp接口实现JPEGD功能时，无需通知解码器开始或结束接收码流数据；用hi_mpi命名的接口实现JPEGD功能时，需通知解码器开始或结束接收码流数据，另外，用户可以另起线程调用hi_mpi_vdec_get_frame接口来获取解码结果。

13.3.3.6.1 媒体数据处理模块初始化

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
无需进行媒体数据处理模块初始化	hi_mpi_sys_init	用户需要修改代码，增加调用媒体数据处理模块初始化的接口hi_mpi_sys_init。

13.3.3.6.2 创建通道

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
创建通道描述信息的结构体： acldvppCreateChannelDesc	无对应接口，直接使用hi_vdec_chn_attr结构体。	用户需要修改代码，直接声明hi_vdec_chn_attr结构体的变量。
无对应接口	直接对hi_vdec_chn_attr结构体的成员赋值： hi_vdec_chn_attr.type（仅hi_mpi有） hi_vdec_chn_attr.pic_width（仅hi_mpi有） hi_vdec_chn_attr.pic_height（仅hi_mpi有） hi_vdec_chn_attr.mode（仅hi_mpi有） hi_vdec_chn_attr.stream_buf_size（仅hi_mpi有） hi_vdec_chn_attr.frame_buf_cnt（仅hi_mpi有） hi_vdec_chn_attr.frame_buf_size（仅hi_mpi有）	用户需要修改代码，对结构体hi_vdec_chn_attr成员赋值来配置通道描述的参数。 需注意，对于仅acl有的这几个参数设置，在hi_mpi上也有对应的接口：
创建通道 acldvppCreateChannel	创建通道 hi_mpi_vdec_create_chn	用户需要修改代码，改用hi_mpi_vdec_create_chn接口创建通道。

13.3.3.6.3 预估输出内存大小

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
acldvppJpegPredictDecSize	hi_mpi_dvpp_get_image_info 源格式解码场景下，可调用hi_mpi_dvpp_get_image_info接口获取解码输出内存的大小； 输出图片格式为YUV420SP的场景下，需使用计算公式计算输出内存大小：hi_img_info.width_stride * hi_img_info.height_stride * 3 /2	用户需要修改代码，改用hi_mpi_dvpp_get_image_info接口或使用计算公式获取图片解码后的输出内存大小。 注意：width_stride 、height_stride的取值要求请参见**图片格式、宽高对齐、内存约束**。

13.3.3.6.4 内存申请

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
申请内存： acldvppMalloc	申请内存： hi_mpi_dvpp_malloc	用户需要修改代码，改用hi_mpi_dvpp_malloc接口申请媒体数据处理功能需要的内存。
释放内存 acldvppFree	释放内存： hi_mpi_dvpp_free	用户需要修改代码，改用hi_mpi_dvpp_free接口释放内存。

13.3.3.6.5 发送图片

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
配置输入码流信息： 无对应接口，在调用acldvppJpegDecodeAsync时直接传入输入图片信息	配置输入码流参数时，直接对hi_vdec_stream结构体的成员赋值： hi_vdec_stream.addr hi_vdec_stream.len hi_vdec_stream.end_of_stream hi_vdec_stream.end_of_frame hi_vdec_stream.display	用户需要修改代码，对结构体hi_vdec_chn_attr成员赋值来配置输入码流信息的参数。
配置输出图片信息： acldvppCreatePicDesc acldvppSetPicDescData acldvppSetPicDescSize acldvppSetPicDescFormat acldvppSetPicDescWidth acldvppSetPicDescHeight acldvppSetPicDescWidthStride acldvppSetPicDescHeightStride	无对应接口，配置输出图片参数时，直接对hi_vdec_pic_info结构体的成员赋值： hi_vdec_pic_info.vir_addr hi_vdec_pic_info.buffer_size hi_vdec_pic_info.pixel_format hi_vdec_pic_info.width hi_vdec_pic_info.height hi_vdec_pic_info.width_stride hi_vdec_pic_info.height_stride	用户需要修改代码，对结构体hi_vdec_pic_info成员赋值来配置输出图片信息的参数。
无对应接口，无需通知解码器开始接收码流数据	通知解码器开始接收码流数据（一个通道只需要调用一次该接口）： hi_mpi_vdec_start_recv_stream	用户需要修改代码，增加调用hi_mpi_vdec_start_recv_stream接口。
每一帧都需调用以下接口发送图片： acldvppJpegDecodeAsync	每一帧都需调用以下接口发送解码码流： hi_mpi_vdec_send_stream	用户需要修改代码，改为调用hi_mpi_vdec_send_stream接口发送码流，同时需要传入通道号。

13.3.3.6.6 接收图片

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
需要调用aclrtSynchronizeStream流同步接口，等待图片解码结束，然后获取解码结果： acldvppGetPicDescData acldvppGetPicDescWidth acldvppGetPicDescHeight acldvppGetPicDescWidthStride acldvppGetPicDescHeightStride acldvppGetPicDescFormat acldvppGetPicDescRetCode（仅acl有）	调用hi_mpi_vdec_get_frame接口获取一帧解码结果，再从hi_video_frame_info结构体的以下成员中获取解码结果： hi_video_frame_info.v_frame.virt_addr[0] hi_video_frame_info.v_frame.width hi_video_frame_info.v_frame.height hi_video_frame_info.v_frame.width_stride hi_video_frame_info.v_frame.height_stride hi_video_frame_info.v_frame.pixel_format hi_video_frame_info.v_frame.frame_flag（仅hi_mpi有）	用户需要修改代码，改为另起线程，调用hi_mpi_vdec_get_frame接口来主动获取解码结果。
无对应接口，无需通知解码器停止接收码流数据	通知解码器停止接收码流数据（一个通道只需要调用一次该接口）： hi_mpi_vdec_stop_recv_stream	用户需要修改代码，增加调用hi_mpi_vdec_stop_recv_stream接口。

13.3.3.6.7 销毁通道

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
销毁通道： acldvppDestroyChannel	销毁通道： hi_mpi_vdec_destroy_chn	用户需要修改代码，改为调用hi_mpi_vdec_destroy_chn接口。 需注意，在调用hi_mpi接口销毁通道前，可以先调用hi_mpi_vdec_query_status接口获取解码器状态，判断是否所有图片数据都已完成解码，然后进行通道的销毁。

13.3.3.6.8 媒体数据处理模块去初始化

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
无需进行媒体数据处理模块去初始化	hi_mpi_sys_exit	用户需要修改代码，增加调用媒体数据处理模块去初始化的接口hi_mpi_sys_exit。

13.3.3.7 JPEGE 功能迁移

JPEGE功能，命名以acldvpp开头的接口与命名以hi_mpi开头的接口，最大的区别在于：用acldvpp接口实现JPEGE功能时，每编码一帧图片，需要传入输出内存地址，并且通过流同步等待一帧编码完成，然后在传入的输出内存中获取编码结果；用hi_mpi命名的接口实现JPEGE功能时，通过用户另起线程调用hi_mpi_venc_get_stream接口来异步获取编码结果。

13.3.3.7.1 媒体数据处理模块初始化

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
无需进行媒体数据处理模块初始化	hi_mpi_sys_init	用户需要修改代码，增加调用媒体数据处理模块初始化的接口hi_mpi_sys_init。

13.3.3.7.2 创建通道

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
创建通道描述信息的结构体： acldvppCreateChannelDesc	无对应接口，直接使用hi_venc_chn_attr结构体。	用户需要修改代码，直接声明hi_venc_chn_attr结构体的变量。
无相关接口	直接对hi_venc_chn_attr结构体的成员赋值： hi_venc_chn_attr.venc_attr.type hi_venc_chn_attr.venc_attr.pic_width hi_venc_chn_attr.venc_attr.pic_height	用户需要修改代码，对结构体hi_venc_chn_attr成员赋值来配置通道描述的参数。

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
创建通道 acldvppCreateChannel	创建通道 hi_mpi_venc_create_chn	用户需要修改代码，改用hi_mpi_venc_create_chn接口创建通道。

13.3.3.7.3 内存申请

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
调用acldvppJpegPredictEncSize来预估JPEG图片编码后所需的输出内存的大小，再调用acldvppMalloc接口申请输出内存。	输出内存可以由系统管理，这时用户无需申请输出内存； 输出内存也可以由用户管理，这时用户可调用hi_mpi_venc_get_jpege_predicted_size接口预估JPEG图片编码后所需的输出内存的大小，再调用hi_mpi_dvpp_malloc接口申请内存。	用户需要修改代码，改用hi_mpi_dvpp_malloc接口申请媒体数据处理功能需要的内存。
释放内存 acldvppFree	释放内存： hi_mpi_dvpp_free	用户需要修改代码，改用hi_mpi_dvpp_free接口释放内存。

13.3.3.7.4 发送编码图片

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
配置输入图片信息： acldvppCreatePicDesc acldvppSetPicDescData acldvppSetPicDescSize acldvppSetPicDescFormat acldvppSetPicDescWidth acldvppSetPicDescHeight acldvppSetPicDescWidthStride acldvppSetPicDescHeightStride	无对应接口，配置输入图片时，直接对hi_video_frame_info结构体的成员赋值： hi_video_frame_info.v_frame.virt_addr hi_video_frame_info.v_frame.pixel_format hi_video_frame_info.v_frame.width hi_video_frame_info.v_frame.height hi_video_frame_info.v_frame.width_stride hi_video_frame_info.v_frame.height_stride	用户需要修改代码，对结构体hi_video_frame_info成员赋值来配置输入图片的参数。 hi_mpi接口无需设置输入数据的大小，系统内部会根据图片分辨率、格式等信息自行计算。
配置图片编码质量 acldvppCreateJpegeConfig acldvppSetJpegeConfigLevel	配置图片编码质量，直接修改hi_venc_jpeg_param.qfactor的值 hi_mpi_venc_set_jpeg_param	用户需要修改代码，改为调用hi_mpi_venc_set_jpeg_param来配置JPEGE的图片编码质量。
无对应接口，无需通知编码器开始接收编码图片	通知编码器开始接收输入数据（一个通道只需要调用一次该接口）： hi_mpi_venc_start_chn	用户需要修改代码，增加调用hi_mpi_venc_start_chn接口。
每一帧都需调用以下接口发送编码图片，同时传入用户申请好的输出内存地址： acldvppJpegEncodeAsync	由系统管理输出内存时，需调用hi_mpi_venc_send_frame接口，每一帧都需调用该接口发送编码图片； 由用户管理输出内存时，需调用hi_mpi_venc_send_jpege_frame接口，每一帧都需调用该接口发送编码图片。	用户需要修改代码，改为调用hi_mpi_venc_send_frame接口或hi_mpi_venc_send_jpege_frame接口发送编码图片，同时需要传入通道号。

13.3.3.7.5 获取编码图片

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
需要调用aclrtSynchronizeStream流同步接口，等待图片编码结束，然后获取编码结果： 编码后图片存放地址为调用acldvppJpegEncodeAsync时传入的输出内存地址，图片大小从acldvppJpegEncodeAsync输出参数size获取	调用hi_mpi_venc_get_stream接口获取一帧编码结果，再从hi_venc_stream结构体的以下成员中获取编码数据： hi_venc_stream.pack[i].addr hi_venc_stream.pack[i].len hi_venc_stream.pack[i].data_type（从该成员结构体中获取输出图片格式） hi_venc_stream.pack[i].pts	用户需要修改代码，改为另起线程，调用hi_mpi_venc_get_stream接口来主动获取编码结果。 需注意，用户需要通过调用hi_mpi_venc_query_status得到每次编码出的包数n，然后再调用hi_mpi_venc_get_stream前，提前给参数hi_venc_stream.pack分配不小于n * sizeof（hi_venc_pack）大小的内存。 获取编码结果成功后，需调用hi_mpi_venc_release_stream接口释放码流缓存。
无对应接口，无需通知编码器停止接收编码数据帧	通知编码器停止接收输入数据（一个通道只需要调用一次该接口）： hi_mpi_venc_stop_chn	用户需要修改代码，增加调用hi_mpi_venc_stop_chn接口。

13.3.3.7.6 销毁通道

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
销毁通道： acldvppDestroyChannel	销毁通道： hi_mpi_venc_destroy_chn	用户需要修改代码，改为调用hi_mpi_venc_destroy_chn接口。 需注意，在调用hi_mpi接口销毁通道前，可以先调用hi_mpi_venc_query_status接口获取编码器状态，判断是否所有输入数据都已完成编码，然后进行通道的销毁。

13.3.3.7.7 媒体数据处理模块去初始化

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
无需进行媒体数据处理模块去初始化	hi_mpi_sys_exit	用户需要修改代码，增加调用媒体数据处理模块去初始化的接口hi_mpi_sys_exit。

13.3.3.8 VENC 功能迁移

VENC功能，命名以acldvpp或aclvenc开头的接口与命名以hi_mpi开头的接口，最大的区别在于：用acldvpp接口实现VENC功能时，需设置回调函数，每编码一帧数据会自动调用回调函数，在回调函数内调用对应的acl的Get接口获取编码数据的内存地址、内存大小等；用hi_mpi命名的接口实现VENC功能时，需要用户另起线程调用hi_mpi_venc_get_stream接口来获取编码结果。

13.3.3.8.1 媒体数据处理模块初始化

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
无需进行媒体数据处理模块初始化	hi_mpi_sys_init	用户需要修改代码，增加调用媒体数据处理模块初始化的接口hi_mpi_sys_init。

13.3.3.8.2 创建通道

Atlas 200/300/500 推理产品 aclvenc接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
创建通道描述信息的结构体： aclvencCreateChannelDesc	无对应接口，直接使用hi_venc_chn_attr结构体。	用户需要修改代码，直接声明hi_venc_chn_attr结构体的变量。

Atlas 200/300/500 推理产品 aclvenc接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
设置通道描述的参数： aclvencSetChannelDescEnType aclvencSetChannelDescPicWidth aclvencSetChannelDescPicHeight aclvencSetChannelDescKeyFrameInterval aclvencSetChannelDescThreadId（仅acl有） aclvencSetChannelDescCallback（仅acl有） aclvencSetChannelDescRcMode aclvencSetChannelDescSrcRate aclvencSetChannelDescMaxBitRate aclvencSetChannelDescPicFormat（仅acl有）	无对应接口，直接对hi_venc_chn_attr结构体的成员赋值： hi_venc_chn_attr.venc_attr.type hi_venc_chn_attr.venc_attr.pic_width hi_venc_chn_attr.venc_attr.pic_height hi_venc_chn_attr.rc_attr.h26x_xbr.gop hi_venc_chn_attr.rc_attr.rc_mode hi_venc_chn_attr.rc_attr.h26x_xbr.src_frame_rate hi_venc_chn_attr.rc_attr.h26x_xbr.bit_rate	用户需要修改代码，对结构体hi_venc_chn_attr成员赋值来配置通道描述的参数。 需注意，对于仅acl有的这几个参数设置，在hi_mpi上也有对应的接口： ● 线程ID、Callback（这两个跟回调函数相关），不在hi_venc_chn_attr结构体中，因为使用hi mpi接口时不需要设置回调函数，需要自行另起线程，调用hi_mpi_venc_get_stream接口来获取编码结果。 ● 输入图片格式不在hi_venc_chn_attr结构体中，是在调用hi_mpi_venc_send_frame接口发送编码数据帧时通过hi_video_frame_info.v_frame.pixel_format参数指定。
创建通道： aclvencCreateChannel	创建通道： hi_mpi_venc_create_chn	用户需要修改代码，改用hi_mpi_venc_create_chn接口创建通道。

13.3.3.8.3 内存申请

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
申请内存： acldvppMalloc	申请内存： hi_mpi_dvpp_malloc	用户需要修改代码，改用hi_mpi_dvpp_malloc接口申请媒体数据处理功能需要的内存。

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
释放内存 acldvppFree	释放内存： hi_mpi_dvpp_free	用户需要修改代码，改用hi_mpi_dvpp_free接口释放内存。

13.3.3.8.4 发送编码视频帧

Atlas 200/300/500 推理产品 acldvpp/aclvenc接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
配置输入视频帧信息： acldvppSetPicDescData acldvppSetPicDescSize acldvppSetPicDescFormat acldvppSetPicDescWidth acldvppSetPicDescHeight acldvppSetPicDescWidthStride acldvppSetPicDescHeightStride	无对应接口，配置输入视频帧时，直接对hi_video_frame_info结构体的成员赋值： hi_video_frame_info.v_frame.virt_addr hi_video_frame_info.v_frame.pixel_format hi_video_frame_info.v_frame.width hi_video_frame_info.v_frame.height hi_video_frame_info.v_frame.width_stride hi_video_frame_info.v_frame.height_stride	用户需要修改代码，对结构体hi_video_frame_info成员赋值来配置输入视频帧的参数。 hi_mpi接口无需设置输入数据的大小，系统内部会根据图片分辨率、格式等信息自行计算。
无对应接口，无需通知编码器开始接收编码数据帧	通知编码器开始接收编码数据帧（一个通道只需要调用一次该接口）： hi_mpi_venc_start_chn	用户需要修改代码，增加调用hi_mpi_venc_start_chn接口。
每一帧都需调用以下接口发送编码视频帧： aclvencSendFrame	每一帧都需调用以下接口发送编码视频帧： hi_mpi_venc_send_frame	用户需要修改代码，改为调用hi_mpi_venc_send_frame接口发送编码视频帧，同时需要传入通道号。

13.3.3.8.5 获取码流

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
设置通道描述信息时已注册回调函数，每编码一帧数据会自动调用回调函数，在回调函数内调用对应的Get接口获取编码数据的内存地址、内存大小等。 acldvppGetStreamDescData acldvppGetStreamDescSize acldvppGetStreamDescFormat acldvppGetStreamDescTimestamp	调用hi_mpi_venc_get_stream接口获取一帧编码结果，再从hi_venc_stream结构体的以下成员中获取编码数据： hi_venc_stream.pack[i].addr hi_venc_stream.pack[i].len hi_venc_stream.pack[i].data_type（从该成员结构体中获取输出图片格式） hi_venc_stream.pack[i].pts	用户需要修改代码，改为另起线程，调用hi_mpi_venc_get_stream接口来主动获取编码结果。 需注意，用户需要通过调用hi_mpi_venc_query_status得到每次编码出的包数n，然后再调用hi_mpi_venc_get_stream前，提前给参数hi_venc_stream.pack分配不小于n * sizeof（hi_venc_pack）大小的内存。 获取编码结果成功后，需调用hi_mpi_venc_release_stream接口释放码流缓存。
无对应接口，无需通知编码器停止接收编码数据帧	通知编码器停止接收编码数据帧（一个通道只需要调用一次该接口）： hi_mpi_venc_stop_chn	用户需要修改代码，增加调用hi_mpi_venc_stop_chn接口。

13.3.3.8.6 销毁通道

Atlas 200/300/500 推理产品 aclvenc接口	Atlas 推理系列产品 hi_mpi 接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
销毁通道： aclvencDestroyChannel	销毁通道： hi_mpi_venc_destroy_chn	用户需要修改代码，改为调用hi_mpi_venc_destroy_chn接口。 需注意，在调用hi_mpi接口销毁通道前，可以先调用hi_mpi_venc_query_status接口获取编码器状态，判断是否所有输入数据都已完成编码，然后进行通道的销毁。

13.3.3.8.7 媒体数据处理模块去初始化

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi 接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
无需进行媒体数据处理模块去初始化	hi_mpi_sys_exit	用户需要修改代码，增加调用媒体数据处理模块去初始化的接口hi_mpi_sys_exit。

13.3.3.9 PNGD 功能迁移

13.3.3.9.1 媒体数据处理模块初始化

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi 接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
无需进行媒体数据处理模块初始化	hi_mpi_sys_init	用户需要修改代码，增加调用媒体数据处理模块初始化的接口hi_mpi_sys_init。

13.3.3.9.2 创建通道

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
创建通道描述信息的结构体： acldvppCreateChannelDesc	无对应接口，直接使用hi_pngd_chn_attr结构体。	用户需要修改代码，直接声明hi_pngd_chn_attr结构体的变量。
创建通道： acldvppCreateChannel	创建通道： hi_mpi_pngd_create_chn	用户需要修改代码，改用hi_mpi_pngd_create_chn接口创建通道。

13.3.3.9.3 预估输出内存

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
预估输出内存： acldvppPngPredictDecSize	源格式解码场景下，可调用hi_mpi_png_get_image_info接口获取解码输出内存的大小； 其它场景下，需使用计算公式计算输出内存大小：hi_img_info.width_stride * hi_img_info.height_stride	用户需要修改代码，改用hi_mpi_png_get_image_info接口或使用计算公式获取图片解码后的输出内存大小。 注意：width_stride 、height_stride的取值要求请参见**图片格式、宽高对齐、内存约束**。

13.3.3.9.4 内存申请

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
申请内存： acldvppMalloc	申请内存： hi_mpi_dvpp_malloc	用户需要修改代码，改用hi_mpi_dvpp_malloc接口申请媒体数据处理功能需要的内存。
释放内存 acldvppFree	释放内存： hi_mpi_dvpp_free	用户需要修改代码，改用hi_mpi_dvpp_free接口释放内存。

13.3.3.9.5 发送图片

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
配置输入码流信息： 无对应接口，在调用acldvppPngDecodeAsync时直接传入输入图片信息	配置输入码流参数时，直接对hi_img_stream结构体的成员赋值： hi_img_stream.type hi_img_stream.addr hi_img_stream.len hi_img_stream.pts	用户需要修改代码，对结构体hi_vdec_chn_attr成员赋值来配置输入码流信息的参数。
配置输出图片信息： acldvppCreatePicDesc acldvppSetPicDescData acldvppSetPicDescSize acldvppSetPicDescFormat acldvppSetPicDescWidth acldvppSetPicDescHeight acldvppSetPicDescWidthStride acldvppSetPicDescHeightStride	无对应接口，配置输出图片参数时，直接对hi_pic_info结构体的成员赋值： hi_pic_info.picture_address hi_pic_info.picture_buffer_size hi_pic_info.picture_format hi_pic_info.picture_width hi_pic_info.picture_height hi_pic_info.picture_width_stride hi_pic_info.picture_height_stride	用户需要修改代码，对结构体hi_pic_info成员赋值来配置输出图片信息的参数。
每一帧都需调用以下接口发送图片： acldvppPngDecodeAsync	每一帧都需调用以下接口发送解码码流： hi_mpi_pngd_send_stream	用户需要修改代码，改为调用hi_mpi_pngd_send_stream接口发送码流，同时需要传入通道号。

13.3.3.9.6 接收图片

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
需要调用aclrtSynchronizeStream流同步接口，等待图片解码结束，然后获取解码结果： acldvppGetPicDescData acldvppGetPicDescWidth acldvppGetPicDescHeight acldvppGetPicDescWidthStride acldvppGetPicDescHeightStride acldvppGetPicDescFormat acldvppGetPicDescRetCode（仅acl有，表示解码状态，即解码是否成功）	调用hi_mpi_pngd_get_image_data接口获取一帧解码结果，再从hi_pic_info结构体的以下成员中获取解码结果： hi_pic_info.picture_address hi_pic_info.picture_width hi_pic_info.picture_height hi_pic_info.picture_width_stride hi_pic_info.picture_height_stride hi_pic_info.picture_format hi_pic_info.picture_buffer_size（仅himpi有）	用户需要修改代码，改为另起线程，调用hi_mpi_pngd_get_image_data接口来主动获取解码结果。 注意，hi_mpi接口中，如果能获取到解码数据，则表示解码成功；如果获取不到解码数据，则解码失败。

13.3.3.9.7 销毁通道

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
销毁通道： acldvppDestroyChannel	销毁通道： hi_mpi_pngd_destroy_chn	用户需要修改代码，改为调用hi_mpi_pngd_destroy_chn接口。 需注意，在调用hi_mpi接口销毁通道前，需要确保所有图片数据都已完成解码，然后进行通道的销毁。

13.3.3.9.8 媒体数据处理模块去初始化

Atlas 200/300/500 推理产品 acldvpp接口	Atlas 推理系列产品 hi_mpi接口	Atlas 200/300/500 推理产品->Atlas 推理系列产品迁移时，对用户的影响
无需进行媒体数据处理模块去初始化	hi_mpi_sys_exit	用户需要修改代码，增加调用媒体数据处理模块去初始化的接口hi_mpi_sys_exit。