Atlas 200I DK A2 开发者套件
23.0.RC3

AscendCL 应用开发指南（Python）

文档版本 01

发布日期 2023-11-16

华为技术有限公司

版权所有 © 华为技术有限公司 2023。 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

漏洞声明

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该政策可参考华为公司官方网站的网址：https://www.huawei.com/cn/psirt/vul-response-process。
如企业客户须获取漏洞信息，请访问：https://securitybulletin.huawei.com/enterprise/cn/security-advisory。

目 录

1 学习向导.....1
2 快速入门.....3
3 概述.....8
3.1 基于 AscendCL 的 pyACL.....8
3.2 基本概念.....9
3.3 pyACL 开发流程.....15
3.4 pyACL 接口调用流程.....16
3.5 注意事项.....17
3.6 应用开发环境准备.....17
4 开发基础推理应用.....19
4.1 开发流程.....19
4.2 模型构建.....20
4.3 pyACL 初始化与去初始化.....20
4.4 运行管理资源申请与释放.....21
4.5 数据传输.....24
4.5.1 接口调用流程.....24
4.5.2 Device 内的数据传输.....25
4.5.3 从 Device 到 Host 的数据传输.....26
4.6 模型推理基本场景.....27
4.6.1 模型加载.....27
4.6.2 模型执行.....30
4.6.2.1 接口调用流程.....30
4.6.2.2 示例代码.....35
4.6.3 模型卸载.....36
5 图像/视频数据处理.....38
5.1 媒体数据处理基础知识.....38
5.2 V1 与 V2 版本的差别.....41
5.3 媒体数据处理 V1.....41
5.3.1 功能支持度说明.....41
5.3.2 VPC 图像处理典型功能.....41
5.3.3 JPEGD 图片解码.....54
5.3.4 JPEGE 图片编码.....57

5.3.5 PNGD 图片解码.....60
5.3.6 VDEC 视频解码.....64
5.3.7 VENC 视频编码.....74
5.4 媒体数据处理 V2.....80
5.4.1 功能支持度说明.....80
5.4.2 VPC 图片处理典型功能.....80
5.4.3 JPEGD 图片解码.....85
5.4.4 JPEGE 图片编码.....89
5.4.5 PNGD 图片解码.....93
5.4.6 VDEC 视频解码.....96
5.4.7 VENC 视频编码.....100
6 单算子调用.....109
6.1 简介.....109
6.2 开发流程.....111
6.3 接口调用流程.....112
6.4 调用 CBLAS 接口.....114
6.5 固定 Shape 算子.....116
6.6 动态 Shape 算子（不注册算子选择器）.....116
6.7 动态 Shape 算子（注册算子选择器）.....117
7 扩展更多特性.....120
7.1 内存二次分配管理.....120
7.2 多 Device 场景.....123
7.3 多模型串联推理.....124
7.4 多 Batch 模型推理.....125
7.5 模型异步推理.....126
7.5.1 接口调用流程.....126
7.5.2 示例代码.....128
7.6 模型动态推理.....130
7.6.1 动态 Batch/动态分辨率/动态维度（设置多档维度值）.....130
7.7 模型动态 AIPP 推理.....134
7.7.1 接口调用流程.....134
7.7.2 动态 AIPP（单个动态 AIPP 输入）.....136
7.7.3 动态 AIPP（多个动态 AIPP 输入）.....137
7.8 Stream 管理.....138
7.8.1 原理介绍.....138
7.8.2 多 Stream 接口调用流程.....138
7.8.3 单线程单 Stream.....140
7.8.4 单线程多 Stream.....140
7.8.5 多线程多 Stream.....140
7.9 同步等待.....141
7.9.1 基本原理.....141
7.9.2 关于 Event 的同步等待.....141

7.9.3 关于 Stream 内任务的同步等待 142
7.9.4 关于 Stream 间任务的同步等待 142
7.9.5 关于 Device 的同步等待 142
7.10 AI Core 异常信息获取 143
7.11 Profiling 性能数据采集 144
7.12 溢出算子数据采集及分析 149
8 应用调试 152
9 应用样例参考 153
9.1 样例使用指导 153
9.1.1 环境准备 153
9.1.1.1 Python 环境 153
9.1.1.2 准备环境 154
9.1.1.3 环境变量配置 155
9.1.2 样例介绍 155
9.1.2.1 实现矩阵-矩阵相加运算 155
9.1.2.1.1 样例介绍 155
9.1.2.2 基于 Caffe ResNet-50 网络实现图片分类（视频解码+同步推理） 157
9.1.2.2.1 样例介绍 157
9.1.2.3 基于 Caffe ResNet-50 网络实现图片分类（同步推理） 162
9.1.2.3.1 样例介绍 162
9.1.2.4 基于 Caffe ResNet-50 网络实现图片分类（异步推理） 164
9.1.2.4.1 样例介绍 164
9.1.2.5 媒体数据处理（视频编码） 168
9.1.2.5.1 样例介绍 168
10 pyACL API 参考 169
10.1 废弃接口/返回码列表 169
10.2 系统配置 170
10.2.1 函数：init 170
10.2.2 函数：finalize 172
10.2.3 函数：get_version 173
10.2.4 函数：get_soc_name 174
10.2.5 函数：get_recent_err_msg 174
10.2.6 函数：set_device_sat_mode 175
10.2.7 函数：get_device_sat_mode 175
10.3 公共接口 175
10.3.1 函数：float16_to_float 176
10.3.2 函数：float_to_float16 176
10.3.3 函数：create_data_buffer 176
10.3.4 函数：destroy_data_buffer 177
10.3.5 函数：get_data_buffer_addr 177
10.3.6 函数：get_data_buffer_size 177

10.3.7 函数：get_data_buffer_size_v2 178
10.3.8 函数：update_data_buffer 178
10.3.9 函数：data_type_size 179
10.3.10 函数：create_tensor_desc 179
10.3.11 函数：destroy_tensor_desc 180
10.3.12 函数：get_tensor_desc_type 180
10.3.13 函数：get_tensor_desc_format 181
10.3.14 函数：get_tensor_desc_size 181
10.3.15 函数：get_tensor_desc_element_count 181
10.3.16 函数：get_tensor_desc_num_dims 182
10.3.17 函数：get_tensor_desc_dim 182
10.3.18 函数：get_tensor_desc_dim_v2 183
10.3.19 函数：get_tensor_desc_dim_range 183
10.3.20 函数：set_tensor_desc_name 184
10.3.21 函数：get_tensor_desc_name 184
10.3.22 函数：set_tensor_storage_shape 185
10.3.23 函数：set_tensor_storage_format 185
10.3.24 函数：set_tensor_origin_shape 186
10.3.25 函数：set_tensor_origin_format 186
10.3.26 函数：set_tensor_shape 187
10.3.27 函数：set_tensor_format 188
10.3.28 函数：set_tensor_const 188
10.3.29 函数：set_tensor_shape_range 189
10.3.30 函数：set_tensor_dynamic_input 189
10.3.31 函数：get_tensor_desc_by_index 190
10.3.32 函数：get_tensor_desc_address 190
10.3.33 函数：set_tensor_place_ment 191
10.3.34 函数：set_tensor_value_range 191
10.3.35 函数：trans_tensor_desc_format 192
10.3.36 函数：app_log 192
10.4 Util 模块 193
10.4.1 函数：numpy_to_ptr 193
10.4.2 函数：numpy_contiguous_to_ptr 194
10.4.3 函数：ptr_to_numpy 195
10.4.4 函数：bytes_to_ptr 196
10.4.5 函数：ptr_to_bytes 196
10.4.6 thread 197
10.4.6.1 函数：start_thread 197
10.4.6.2 函数：stop_thread 197
10.5 Runtime（rt） 198
10.5.1 Device 管理 198
10.5.1.1 函数：set_device 198

10.5.1.2 函数：reset_device 199
10.5.1.3 函数：get_device 199
10.5.1.4 函数：get_run_mode 200
10.5.1.5 函数：set_ts_device 200
10.5.1.6 函数：get_device_count 201
10.5.2 Context 管理 201
10.5.2.1 函数：create_context 201
10.5.2.2 函数：destroy_context 202
10.5.2.3 函数：set_context 203
10.5.2.4 函数：get_context 203
10.5.3 Stream 管理 204
10.5.3.1 函数：create_stream 204
10.5.3.2 函数：create_stream_with_config 205
10.5.3.3 函数：destroy_stream 205
10.5.3.4 函数：destroy_stream_force 206
10.5.3.5 函数：set_stream_overflow_switch 206
10.5.3.6 函数：get_stream_overflow_switch 207
10.5.3.7 函数：set_stream_failure_mode 207
10.5.4 同步等待 208
10.5.4.1 函数：create_event 208
10.5.4.2 函数：create_event_with_flag 209
10.5.4.3 函数：destroy_event 209
10.5.4.4 函数：record_event 210
10.5.4.5 函数：reset_event 211
10.5.4.6 函数：query_event 211
10.5.4.7 函数：query_event_status 212
10.5.4.8 函数：query_event_wait_status 212
10.5.4.9 函数：synchronize_event 213
10.5.4.10 函数：event_elapsed_time 213
10.5.4.11 函数：stream_wait_event 214
10.5.4.12 函数：synchronize_device 214
10.5.4.13 函数：synchronize_stream 215
10.5.4.14 函数：synchronize_stream_with_timeout 215
10.5.4.15 函数：subscribe_report 216
10.5.4.16 函数：launch_callback 217
10.5.4.17 函数：process_report 217
10.5.4.18 函数：unsubscribe_report 218
10.5.4.19 函数：set_exception_info_callback 218
10.5.4.20 函数：get_task_id_from_exception_info 219
10.5.4.21 函数：get_stream_id_from_exception_info 220
10.5.4.22 函数：get_thread_id_from_exception_info 220
10.5.4.23 函数：get_device_id_from_exception_info 221

10.5.4.24 函数：get_error_code_from_exception_info 221
10.5.4.25 函数：set_op_wait_timeout 222
10.5.5 内存管理 222
10.5.5.1 总体说明 222
10.5.5.2 函数：malloc 223
10.5.5.3 函数：malloc_cached 224
10.5.5.4 函数：mem_flush 225
10.5.5.5 函数：mem_invalidate 226
10.5.5.6 函数：get_mem_info 226
10.5.5.7 函数：free 227
10.5.5.8 函数：malloc_host 228
10.5.5.9 函数：free_host 228
10.5.5.10 函数：memset 229
10.5.5.11 函数：memset_async 229
10.5.5.12 函数：memcpy 230
10.5.5.13 函数：memcpy_async 231
10.5.5.14 函数：memcpy2d 232
10.5.5.15 函数：memcpy2d_async 232
10.6 模型加载与执行（mdl） 233
10.6.1 模型加载与执行 233
10.6.1.1 函数：load_from_file 233
10.6.1.2 函数：load_from_mem 234
10.6.1.3 函数：load_from_file_with_mem 235
10.6.1.4 函数：load_from_mem_with_mem 236
10.6.1.5 函数：execute 237
10.6.1.6 函数：execute_async 238
10.6.1.7 函数：unload 239
10.6.1.8 函数：query_size 240
10.6.1.9 函数：query_size_from_mem 240
10.6.1.10 函数：set_dynamic_batch_size 241
10.6.1.11 函数：set_dynamic_hw_size 242
10.6.1.12 函数：create_and_get_op_desc 242
10.6.1.13 函数：init_dump 244
10.6.1.14 函数：set_dump 245
10.6.1.15 函数：finalize_dump 247
10.6.1.16 函数：get_aipp_type 247
10.6.1.17 函数：set_aipp_by_input_index 248
10.6.1.18 函数：set_input_dynamic_dims 250
10.6.1.19 函数：create_config_handle 251
10.6.1.20 函数：destroy_config_handle 252
10.6.1.21 函数：set_config_opt 252
10.6.1.22 函数：load_with_config 253

10.6.1.23 函数：get_tensor_real_name.....254
10.6.2 aclmdlAIPP.....255
10.6.2.1 函数：create_aipp.....255
10.6.2.2 函数：destroy_aipp.....255
10.6.2.3 设置动态 AIPP 参数.....255
10.6.2.3.1 函数：set_aipp_input_format.....255
10.6.2.3.2 函数：set_aipp_csc_params.....256
10.6.2.3.3 函数：set_aipp_rbuv_swap_switch.....259
10.6.2.3.4 函数：set_aipp_ax_swap_switch.....259
10.6.2.3.5 函数：set_aipp_src_image_size.....260
10.6.2.3.6 函数：set_aipp_crop_params.....261
10.6.2.3.7 函数：set_aipp_padding_params.....262
10.6.2.3.8 函数：set_aipp_dtc_pixel_mean.....263
10.6.2.3.9 函数：set_aipp_dtc_pixel_min.....263
10.6.2.3.10 函数：set_aipp_pixel_var_reci.....264
10.6.2.3.11 函数：set_input_aipp.....265
10.6.3 aclmdlDataset.....267
10.6.3.1 总体说明.....267
10.6.3.2 函数：create_dataset.....268
10.6.3.3 函数：destroy_dataset.....268
10.6.3.4 函数：add_dataset_buffer.....268
10.6.3.5 函数：get_dataset_num_buffers.....269
10.6.3.6 函数：get_dataset_buffer.....269
10.6.3.7 函数：set_dataset_tensor_desc.....270
10.6.3.8 函数：get_dataset_tensor_desc.....270
10.6.4 aclmdlDesc.....271
10.6.4.1 函数：create_desc.....271
10.6.4.2 函数：destroy_desc.....271
10.6.4.3 函数：get_desc.....272
10.6.4.4 函数：get_num_inputs.....272
10.6.4.5 函数：get_num_outputs.....273
10.6.4.6 函数：get_input_size_by_index.....273
10.6.4.7 函数：get_output_size_by_index.....273
10.6.4.8 函数：get_input_dims.....274
10.6.4.9 函数：get_input_dims_v2.....276
10.6.4.10 函数：get_output_dims.....278
10.6.4.11 函数：get_input_name_by_index.....279
10.6.4.12 函数：get_output_name_by_index.....279
10.6.4.13 函数：get_input_format.....280
10.6.4.14 函数：get_output_format.....281
10.6.4.15 函数：get_input_data_type.....281
10.6.4.16 函数：get_output_data_type.....282

10.6.4.17 函数：get_input_index_by_name....283
10.6.4.18 函数：get_output_index_by_name....284
10.6.4.19 函数：get_dynamic_batch....284
10.6.4.20 函数：get_dynamic_hw....285
10.6.4.21 函数：get_cur_output_dims....285
10.6.4.22 函数：get_first_aipp_info....286
10.6.4.23 函数：get_input_dynamic_gear_count....287
10.6.4.24 函数：get_input_dynamic_dims....287
10.6.4.25 函数：get_op_attr....289
10.7 媒体数据处理 V1....289
10.7.1 总体说明....289
10.7.2 基本概念....291
10.7.3 内存申请与释放....292
10.7.3.1 函数：dvpp_malloc....292
10.7.3.2 函数：dvpp_free....293
10.7.4 通道创建与释放....294
10.7.4.1 VPC/JPEGD/JPEGE/PNGD 图片处理通道....294
10.7.4.1.1 函数：dvpp_create_channel....294
10.7.4.1.2 函数：dvpp_destroy_channel....294
10.7.4.2 VDEC 视频解码通道....295
10.7.4.2.1 函数：vdec_create_channel....295
10.7.4.2.2 函数：vdec_destroy_channel....296
10.7.4.3 VENC 视频编码通道....297
10.7.4.3.1 函数：venc_create_channel....297
10.7.4.3.2 函数：venc_destroy_channel....298
10.7.5 common....298
10.7.5.1 函数：dvpp_create_channel_desc....298
10.7.5.2 函数：dvpp_destroy_channel_desc....299
10.7.5.3 函数：dvpp_get_channel_desc_channel_id....299
10.7.5.4 函数：dvpp_set_channel_desc_mode....300
10.7.5.5 函数：dvpp_set_channel_desc_param....300
10.7.5.6 函数：dvpp_get_channel_desc_param....301
10.7.5.7 函数：vdec_set_channel_desc_param....302
10.7.5.8 函数：vdec_get_channel_desc_param....302
10.7.5.9 函数：dvpp_create_pic_desc....303
10.7.5.10 函数：dvpp_destroy_pic_desc....303
10.7.5.11 函数：dvpp_set_pic_desc_data....304
10.7.5.12 函数：dvpp_set_pic_desc_size....304
10.7.5.13 函数：dvpp_set_pic_desc_format....305
10.7.5.14 函数：dvpp_set_pic_desc_width....305
10.7.5.15 函数：dvpp_set_pic_desc_height....305
10.7.5.16 函数：dvpp_set_pic_desc_width_stride....306

10.7.5.17 函数：dvpp_set_pic_desc_height_stride 306
10.7.5.18 函数：dvpp_set_pic_desc_ret_code 307
10.7.5.19 函数：dvpp_get_pic_desc_data 307
10.7.5.20 函数：dvpp_get_pic_desc_size 308
10.7.5.21 函数：dvpp_get_pic_desc_format 308
10.7.5.22 函数：dvpp_get_pic_desc_width 308
10.7.5.23 函数：dvpp_get_pic_desc_height 309
10.7.5.24 函数：dvpp_get_pic_desc_width_stride 309
10.7.5.25 函数：dvpp_get_pic_desc_height_stride 309
10.7.5.26 函数：dvpp_get_pic_desc_ret_code 310
10.7.5.27 函数：dvpp_create_stream_desc 310
10.7.5.28 函数：dvpp_destroy_stream_desc 310
10.7.5.29 函数：dvpp_set_stream_desc_data 311
10.7.5.30 函数：dvpp_set_stream_desc_size 311
10.7.5.31 函数：dvpp_set_stream_desc_format 312
10.7.5.32 函数：dvpp_set_stream_desc_timestamp 312
10.7.5.33 函数：dvpp_set_stream_desc_ret_code 313
10.7.5.34 函数：dvpp_set_stream_desc_eos 313
10.7.5.35 函数：dvpp_get_stream_desc_data 314
10.7.5.36 函数：dvpp_get_stream_desc_size 314
10.7.5.37 函数：dvpp_get_stream_desc_format 314
10.7.5.38 函数：dvpp_get_stream_desc_timestamp 315
10.7.5.39 函数：dvpp_get_stream_desc_ret_code 315
10.7.5.40 函数：dvpp_get_stream_desc_eos 316
10.7.5.41 函数：dvpp_create_batch_pic_desc 316
10.7.5.42 函数：dvpp_destroy_batch_pic_desc 316
10.7.5.43 函数：dvpp_get_pic_desc 317
10.7.5.44 函数：dvpp_create_border_config 317
10.7.5.45 函数：dvpp_destroy_border_config 318
10.7.5.46 函数：dvpp_set_border_config_value 318
10.7.5.47 函数：dvpp_set_border_config_border_type 319
10.7.5.48 函数：dvpp_set_border_config_top 319
10.7.5.49 函数：dvpp_set_border_config_bottom 320
10.7.5.50 函数：dvpp_set_border_config_left 320
10.7.5.51 函数：dvpp_set_border_config_right 321
10.7.5.52 函数：dvpp_get_border_config_value 321
10.7.5.53 函数：dvpp_get_border_config_border_type 321
10.7.5.54 函数：dvpp_get_border_config_top 322
10.7.5.55 函数：dvpp_get_border_config_bottom 322
10.7.5.56 函数：dvpp_get_border_config_left 322
10.7.5.57 函数：dvpp_get_border_config_right 323
10.7.6 VPC 功能 323

10.7.6.1 功能说明 323
10.7.6.2 约束说明 325
10.7.6.3 性能指标说明 329
10.7.6.4 函数：dvpp_vpc_resize_async 331
10.7.6.5 函数：dvpp_vpc_crop_async 332
10.7.6.6 函数：dvpp_vpc_crop_resize_async 333
10.7.6.7 函数：dvpp_vpc_batch_crop_async 334
10.7.6.8 函数：dvpp_vpc_batch_crop_resize_async 336
10.7.6.9 函数：dvpp_vpc_crop_and_paste_async 338
10.7.6.10 函数：dvpp_vpc_crop_resize_paste_async 339
10.7.6.11 函数：dvpp_vpc_batch_crop_and_paste_async 341
10.7.6.12 函数：dvpp_vpc_batch_crop_resize_paste_async 343
10.7.6.13 函数：dvpp_vpc_batch_crop_resize_make_border_async 345
10.7.6.14 函数：dvpp_vpc_convert_color_async 347
10.7.6.15 函数：dvpp_vpc_pyr_down_async 348
10.7.6.16 函数：dvpp_create_roi_config 349
10.7.6.17 函数：dvpp_destroy_roi_config 350
10.7.6.18 函数：dvpp_set_roi_config 350
10.7.6.19 函数：dvpp_set_roi_config_left 351
10.7.6.20 函数：dvpp_set_roi_config_right 351
10.7.6.21 函数：dvpp_set_roi_config_top 351
10.7.6.22 函数：dvpp_set_roi_config_bottom 352
10.7.6.23 函数：dvpp_create_resize_config 352
10.7.6.24 函数：dvpp_destroy_resize_config 352
10.7.6.25 函数：dvpp_set_resize_config_interpolation 353
10.7.6.26 函数：dvpp_get_resize_config_interpolation 354
10.7.7 JPEGD 功能 354
10.7.7.1 功能及约束说明 354
10.7.7.2 性能指标说明 357
10.7.7.3 函数：dvpp_jpeg_decode_async 358
10.7.7.4 函数：dvpp_jpeg_get_image_info 359
10.7.7.5 函数：dvpp_jpeg_get_image_info_v2 359
10.7.7.6 函数：dvpp_jpeg_predict_dec_size 360
10.7.8 JPEGE 功能 360
10.7.8.1 功能及约束说明 360
10.7.8.2 性能指标说明 362
10.7.8.3 函数：dvpp_jpeg_encode_async 362
10.7.8.4 函数：dvpp_jpeg_predict_enc_size 363
10.7.8.5 函数：dvpp_create_jpege_config 364
10.7.8.6 函数：dvpp_destroy_jpege_config 364
10.7.8.7 函数：dvpp_set_jpege_config_level 365
10.7.8.8 函数：dvpp_get_jpege_config_level 365

10.7.9 VDEC 功能 365
10.7.9.1 功能及约束说明 366
10.7.9.2 性能指标说明 367
10.7.9.3 函数：vdec_send_frame 368
10.7.9.4 函数：vdec_send_skipped_frame 370
10.7.9.5 函数：vdec_create_frame_config 371
10.7.9.6 函数：vdec_destroy_frame_config 372
10.7.9.7 函数：vdec_create_channel_desc 372
10.7.9.8 函数：vdec_destroy_channel_desc 373
10.7.9.9 函数：vdec_set_channel_desc_channel_id 373
10.7.9.10 函数：vdec_set_channel_desc_thread_id 373
10.7.9.11 函数：vdec_set_channel_desc_callback 374
10.7.9.12 函数：vdec_set_channel_desc_entype 374
10.7.9.13 函数：vdec_set_channel_desc_out_pic_format 375
10.7.9.14 函数：vdec_set_channel_desc_out_pic_width 375
10.7.9.15 函数：vdec_set_channel_desc_out_pic_height 376
10.7.9.16 函数：vdec_set_channel_desc_ref_frame_num 376
10.7.9.17 函数：vdec_set_channel_desc_out_mode 377
10.7.9.18 函数：vdec_set_channel_desc_bit_depth 377
10.7.9.19 函数：vdec_get_channel_desc_channel_id 378
10.7.9.20 函数：vdec_get_channel_desc_thread_id 378
10.7.9.21 函数：vdec_get_channel_desc_callback 379
10.7.9.22 函数：vdec_get_channel_desc_entype 379
10.7.9.23 函数：vdec_get_channel_desc_out_pic_format 379
10.7.9.24 函数：vdec_get_channel_desc_out_pic_width 380
10.7.9.25 函数：vdec_get_channel_desc_out_pic_height 380
10.7.9.26 函数：vdec_get_channel_desc_ref_frame_num 380
10.7.9.27 函数：vdec_get_channel_desc_out_mode 381
10.7.9.28 函数：vdec_get_channel_desc_bit_depth 381
10.7.10 VENC 功能 382
10.7.10.1 功能及约束说明 382
10.7.10.2 性能指标说明 383
10.7.10.3 函数：venc_send_frame 383
10.7.10.4 函数：venc_create_channel_desc 384
10.7.10.5 函数：venc_destroy_channel_desc 385
10.7.10.6 函数：venc_set_channel_desc_thread_id 385
10.7.10.7 函数：venc_set_channel_desc_callback 385
10.7.10.8 函数：venc_set_channel_desc_entype 386
10.7.10.9 函数：venc_set_channel_desc_pic_format 386
10.7.10.10 函数：venc_set_channel_desc_pic_width 387
10.7.10.11 函数：venc_set_channel_desc_pic_height 387
10.7.10.12 函数：venc_set_channel_desc_key_frame_interval 388

10.7.10.13 函数：venc_set_channel_desc_buf_addr....388
10.7.10.14 函数：venc_set_channel_desc_buf_size....388
10.7.10.15 函数：venc_set_channel_desc_rc_mode....389
10.7.10.16 函数：venc_set_channel_desc_src_rate....389
10.7.10.17 函数：venc_set_channel_desc_max_bit_rate....390
10.7.10.18 函数：venc_get_channel_desc_channel_id....390
10.7.10.19 函数：venc_get_channel_desc_thread_id....391
10.7.10.20 函数：venc_get_channel_desc_callback....391
10.7.10.21 函数：venc_get_channel_desc_entype....391
10.7.10.22 函数：venc_get_channel_desc_pic_format....392
10.7.10.23 函数：venc_get_channel_desc_pic_width....392
10.7.10.24 函数：venc_get_channel_desc_pic_height....392
10.7.10.25 函数：venc_get_channel_desc_key_frame_interval....393
10.7.10.26 函数：venc_get_channel_desc_buf_addr....393
10.7.10.27 函数：venc_get_channel_desc_buf_size....393
10.7.10.28 函数：venc_get_channel_desc_rc_mode....394
10.7.10.29 函数：venc_get_channel_desc_src_rate....394
10.7.10.30 函数：venc_get_channel_desc_max_bit_rate....394
10.7.10.31 函数：venc_set_channel_desc_param....395
10.7.10.32 函数：venc_get_channel_desc_param....395
10.7.10.33 函数：venc_create_frame_config....396
10.7.10.34 函数：venc_destroy_frame_config....396
10.7.10.35 函数：venc_set_frame_config_force_i_frame....397
10.7.10.36 函数：venc_set_frame_config_eos....397
10.7.10.37 函数：venc_get_frame_config_force_i_frame....397
10.7.10.38 函数：venc_get_frame_config_eos....398
10.7.11 PNGD 功能....398
10.7.11.1 功能及约束说明....398
10.7.11.2 性能指标说明....400
10.7.11.3 函数：dvpp_png_decode_async....400
10.7.11.4 函数：dvpp_png_get_image_info....402
10.7.11.5 函数：dvpp_png_predict_dec_size....402
10.8 媒体数据处理 V2....403
10.8.1 总体说明....403
10.8.2 基本概念....404
10.8.3 公共接口....405
10.8.3.1 acl.himpi.sys_init....405
10.8.3.2 acl.himpi.sys_exit....405
10.8.3.3 acl.himpi.dvpp_malloc....406
10.8.3.4 acl.himpi.dvpp_free....407
10.8.3.5 acl.himpi.dvpp_get_image_info....407
10.8.3.6 acl.himpi.dvpp_get_version....407

10.8.3.7 acl.himpi.sys_create_epoll....408
10.8.3.8 acl.himpi.sys_ctl_epoll....408
10.8.3.9 acl.himpi.sys_wait_epoll....409
10.8.3.10 acl.himpi.sys_close_epoll....410
10.8.3.11 acl.himpi.sys_get_image_align_info....411
10.8.3.12 acl.himpi.sys_set_chn_csc_matrix....411
10.8.3.13 acl.himpi.sys_get_chn_csc_matrix....412
10.8.4 VPC 图像处理接口....413
10.8.4.1 VPC 功能....413
10.8.4.2 约束说明....416
10.8.4.3 性能指标说明....422
10.8.4.4 acl.himpi.vpc_create_chn....423
10.8.4.5 acl.himpi.vpc_destroy_chn....424
10.8.4.6 acl.himpi.vpc_resize....424
10.8.4.7 acl.himpi.vpc_crop....426
10.8.4.8 acl.himpi.vpc_crop_resize....427
10.8.4.9 acl.himpi.vpc_crop_resize_paste....428
10.8.4.10 acl.himpi.vpc_convert_color....429
10.8.4.11 acl.himpi.vpc_convert_color_v2....430
10.8.4.12 acl.himpi.vpc_convert_color_to_yuv420....431
10.8.4.13 acl.himpi.vpc_copy_make_border....433
10.8.4.14 acl.himpi.vpc_pyrdown....434
10.8.4.15 acl.himpi.vpc_calc_hist....436
10.8.4.16 acl.himpi.vpc_calc_hist_v2....437
10.8.4.17 acl.himpi.vpc_equalize_hist....439
10.8.4.18 acl.himpi.vpc_get_process_result....439
10.8.4.19 acl.himpi.vpc_sys_create_chn....440
10.8.4.20 acl.himpi.vpc_crop_resize_make_border....441
10.8.4.21 acl.himpi.vpc_batch_crop_resize_paste....442
10.8.4.22 acl.himpi.vpc_batch_crop_resize_make_border....444
10.8.4.23 acl.himpi.vpc_blur....446
10.8.4.24 acl.himpi.vpc_crop_resize_resize_paste....447
10.8.4.25 acl.himpi.vpc_dilate....449
10.8.4.26 acl.himpi.vpc_draw_cover....450
10.8.4.27 acl.himpi.vpc_draw_line....450
10.8.4.28 acl.himpi.vpc_draw_mosaic....451
10.8.4.29 acl.himpi.vpc_draw_osd....452
10.8.4.30 acl.himpi.vpc_erode....453
10.8.4.31 acl.himpi.vpc_filter2d....454
10.8.4.32 acl.himpi.vpc_flip....455
10.8.4.33 acl.himpi.vpc_gaussian_blur....457
10.8.4.34 acl.himpi.vpc_get_affine_lut....458

10.8.4.35 acl.himpi.vpc_get_affine_transform 458
10.8.4.36 acl.himpi.vpc_get_lut_mem_size 459
10.8.4.37 acl.himpi.vpc_get_perspective_lut 459
10.8.4.38 acl.himpi.vpc_get_remap_lut 460
10.8.4.39 acl.himpi.vpc_get_rotation_matrix 461
10.8.4.40 acl.himpi.vpc_lut_remap 461
10.8.4.41 acl.himpi.vpc_median_blur 463
10.8.4.42 acl.himpi.vpc_rotate 464
10.8.4.43 acl.himpi.vpc_set_chn_workspace 465
10.8.4.44 acl.himpi.vpc_warp_affine 466
10.8.5 VDEC 视频解码/JPEGD 图像解码接口 468
10.8.5.1 JPEGD 功能及约束说明 468
10.8.5.2 JPEGD 性能指标数据 471
10.8.5.3 VDEC 功能及约束说明 472
10.8.5.4 VDEC 性能指标数据 473
10.8.5.5 acl.himpi.vdec_create_chn 475
10.8.5.6 acl.himpi.vdec_destroy_chn 476
10.8.5.7 acl.himpi.vdec_get_chn_attr 477
10.8.5.8 acl.himpi.vdec_set_chn_attr 477
10.8.5.9 acl.himpi.vdec_start_recv_stream 478
10.8.5.10 acl.himpi.vdec_stop_recv_stream 478
10.8.5.11 acl.himpi.vdec_query_status 479
10.8.5.12 acl.himpi.vdec_reset_chn 480
10.8.5.13 acl.himpi.vdec_set_chn_param 480
10.8.5.14 acl.himpi.vdec_get_chn_param 481
10.8.5.15 acl.himpi.vdec_set_protocol_param 481
10.8.5.16 acl.himpi.vdec_get_protocol_param 482
10.8.5.17 acl.himpi.vdec_send_stream 483
10.8.5.18 acl.himpi.vdec_get_frame 484
10.8.5.19 acl.himpi.vdec_release_frame 486
10.8.5.20 acl.himpi.vdec_get_fd 486
10.8.5.21 acl.himpi.vdec_close_fd 487
10.8.5.22 acl.himpi.vdec_get_pic_buf_size 487
10.8.5.23 acl.himpi.vdec_get_tmv_buf_size 488
10.8.5.24 acl.himpi.vdec_set_jpegd_precision_mode 488
10.8.5.25 acl.himpi.vdec_get_jpegd_output_info 489
10.8.6 VENC 视频/JPEGE 图像编码接口 489
10.8.6.1 JPEGE 功能及约束说明 489
10.8.6.2 JPEGE 性能指标说明 491
10.8.6.3 VENC 功能及约束说明 492
10.8.6.4 VENC 性能指标说明 493
10.8.6.5 acl.himpi.venc_create_chn 493

10.8.6.6 acl.himpi.venc_destroy_chn.....495
10.8.6.7 acl.himpi.venc_start_chn.....495
10.8.6.8 acl.himpi.venc_stop_chn.....496
10.8.6.9 acl.himpi.venc_query_status.....497
10.8.6.10 acl.himpi.venc_get_stream.....498
10.8.6.11 acl.himpi.venc_release_stream.....501
10.8.6.12 acl.himpi.venc_send_frame.....501
10.8.6.13 acl.himpi.venc_set_mod_param.....502
10.8.6.14 acl.himpi.venc_get_mod_param.....503
10.8.6.15 acl.himpi.venc_request_idr.....503
10.8.6.16 acl.himpi.venc_get_fd.....504
10.8.6.17 acl.himpi.venc_close_fd.....504
10.8.6.18 acl.himpi.venc_set_jpeg_param.....505
10.8.6.19 acl.himpi.venc_get_jpeg_param.....506
10.8.6.20 acl.himpi.venc_set_chn_param.....506
10.8.6.21 acl.himpi.venc_set_scene_mode.....507
10.8.6.22 acl.himpi.venc_get_scene_mode.....507
10.8.6.23 acl.himpi.venc_set_rc_param.....508
10.8.6.24 acl.himpi.venc_get_rc_param.....508
10.8.6.25 acl.himpi.venc_set_jpeg_huffman_param.....509
10.8.6.26 acl.himpi.venc_get_jpeg_huffman_param.....510
10.8.6.27 acl.himpi.venc_compact_jpeg_tables.....510
10.8.6.28 acl.himpi.venc_send_jpege_frame.....511
10.8.6.29 acl.himpi.venc_get_jpege_predicted_size.....512
10.8.6.30 acl.himpi.venc_get_cu_pred.....512
10.8.6.31 acl.himpi.venc_set_cu_pred.....513
10.8.6.32 acl.himpi.venc_get_h264_vui.....514
10.8.6.33 acl.himpi.venc_set_h264_vui.....515
10.8.6.34 acl.himpi.venc_get_h265_vui.....515
10.8.6.35 acl.himpi.venc_set_h265_vui.....516
10.8.6.36 acl.himpi.venc_get_intra_refresh.....517
10.8.6.37 acl.himpi.venc_set_intra_refresh.....517
10.8.6.38 acl.himpi.venc_get_ref_param.....518
10.8.6.39 acl.himpi.venc_set_ref_param.....519
10.8.6.40 acl.himpi.venc_get_roi_attr.....521
10.8.6.41 acl.himpi.venc_set_roi_attr.....522
10.8.7 PNGD 图像处理接口.....523
10.8.7.1 功能及约束说明.....523
10.8.7.2 性能指标说明.....525
10.8.7.3 acl.himpi.pngd_create_chn.....526
10.8.7.4 acl.himpi.pngd_destroy_chn.....526
10.8.7.5 acl.himpi.pngd_send_stream.....527

10.8.7.6 acl.himpi.pngd_get_image_data....528
10.8.7.7 acl.himpi.png_get_image_info....529
10.8.8 数据类型....529
10.8.8.1 总体说明....529
10.8.8.2 公共....530
10.8.8.2.1 基本数据类型说明....530
10.8.8.2.2 hi_pixel_format....530
10.8.8.2.3 hi_payload_type....534
10.8.8.2.4 hi_video_field....536
10.8.8.2.5 hi_dynamic_range....536
10.8.8.2.6 hi_color_gamut....537
10.8.8.2.7 hi_video_supplement....538
10.8.8.2.8 hi_video_frame....539
10.8.8.2.9 hi_video_frame_info....541
10.8.8.2.10 hi_vb_src....542
10.8.8.2.11 hi_mod_id....542
10.8.8.2.12 hi_data_bit_width....543
10.8.8.2.13 hi_video_format....543
10.8.8.2.14 hi_img_info....544
10.8.8.2.15 hi_pic_buf_attr....545
10.8.8.2.16 hi_jpeg_raw_format....545
10.8.8.2.17 hi_dvpp_epoll_ctl_op....545
10.8.8.2.18 hi_dvpp_epoll_event_type....546
10.8.8.2.19 hi_dvpp_epoll_event....546
10.8.8.2.20 hi_csc_matrix....547
10.8.8.2.21 hi_coefficient....549
10.8.8.2.22 hi_csc_coefficient....549
10.8.8.2.23 hi_video_size....550
10.8.8.2.24 hi_rect....550
10.8.8.2.25 hi_op_mode....551
10.8.8.2.26 hi_point....551
10.8.8.2.27 hi_size....551
10.8.8.2.28 hi_compress_mode....552
10.8.8.2.29 hi_module_type....552
10.8.8.2.30 hi_img_base_info....553
10.8.8.2.31 hi_img_align_info....553
10.8.8.3 视频/图像解码....554
10.8.8.3.1 hi_vdec_send_mode....554
10.8.8.3.2 hi_vdec_frame_type....554
10.8.8.3.3 hi_quick_mark_mode....555
10.8.8.3.4 hi_vdec_pic_info....556
10.8.8.3.5 hi_vdec_video_attr....558

10.8.8.3.6 hi_vdec_chn_attr.....559
10.8.8.3.7 hi_h264_protocol_param.....560
10.8.8.3.8 hi_h265_protocol_param.....561
10.8.8.3.9 hi_vdec_protocol_param.....562
10.8.8.3.10 hi_video_dec_mode.....562
10.8.8.3.11 hi_video_out_order.....563
10.8.8.3.12 hi_vdec_video_param.....563
10.8.8.3.13 hi_vdec_param_picture.....564
10.8.8.3.14 hi_vdec_chn_param.....565
10.8.8.3.15 hi_vdec_dec_err.....565
10.8.8.3.16 hi_vdec_chn_status.....566
10.8.8.3.17 hi_vdec_stream.....567
10.8.8.3.18 hi_vdec_supplement_info.....568
10.8.8.3.19 hi_vdec_video_supplement_info.....568
10.8.8.3.20 hi_jpegd_precision_mode.....569
10.8.8.4 视频/图像编码.....569
10.8.8.4.1 hi_venc_stream.....569
10.8.8.4.2 hi_venc_pack.....570
10.8.8.4.3 hi_venc_chn_attr.....571
10.8.8.4.4 hi_venc_attr.....572
10.8.8.4.5 hi_venc_rc_attr.....576
10.8.8.4.6 hi_venc_rc_mode.....578
10.8.8.4.7 hi_venc_h264_cbr.....580
10.8.8.4.8 hi_venc_h264_vbr.....581
10.8.8.4.9 hi_venc_h264_avbr.....583
10.8.8.4.10 hi_venc_h264_qvbr.....584
10.8.8.4.11 hi_venc_h264_cvbr.....585
10.8.8.4.12 hi_venc_h264_fixqp.....587
10.8.8.4.13 hi_venc_h264_qpmap.....588
10.8.8.4.14 hi_venc_mjpeg_cbr.....589
10.8.8.4.15 hi_venc_mjpeg_vbr.....589
10.8.8.4.16 hi_venc_mjpeg_fixqp.....589
10.8.8.4.17 hi_venc_h265_cbr.....589
10.8.8.4.18 hi_venc_h265_vbr.....591
10.8.8.4.19 hi_venc_h265_avbr.....593
10.8.8.4.20 hi_venc_h265_qvbr.....594
10.8.8.4.21 hi_venc_h265_cvbr.....596
10.8.8.4.22 hi_venc_h265_fixqp.....597
10.8.8.4.23 hi_venc_h265_qpmap.....598
10.8.8.4.24 hi_venc_gop_attr.....599
10.8.8.4.25 hi_venc_gop_mode.....599
10.8.8.4.26 hi_venc_gop_normal_p.....600

10.8.8.4.27 hi_venc_gop_dual_p.....600
10.8.8.4.28 hi_venc_gop_smart_p.....601
10.8.8.4.29 hi_venc_gop_adv_smart_p.....601
10.8.8.4.30 hi_venc_gop_bipred_b.....602
10.8.8.4.31 hi_venc_start_param.....602
10.8.8.4.32 hi_venc_chn_status.....602
10.8.8.4.33 hi_venc_h264_stream_info.....603
10.8.8.4.34 hi_venc_ref_type.....604
10.8.8.4.35 hi_venc_jpeg_stream_info.....604
10.8.8.4.36 hi_venc_h265_stream_info.....605
10.8.8.4.37 hi_venc_prores_stream_info.....606
10.8.8.4.38 hi_venc_mod_param.....606
10.8.8.4.39 hi_venc_mod_type.....607
10.8.8.4.40 hi_venc_venc_mod_param.....608
10.8.8.4.41 hi_venc_h264_mod_param.....608
10.8.8.4.42 hi_venc_h265_mod_param.....609
10.8.8.4.43 hi_venc_jpeg_mod_param.....610
10.8.8.4.44 hi_venc_jpeg_param.....611
10.8.8.4.45 hi_venc_h264_adv_stream_info.....612
10.8.8.4.46 hi_venc_h265_adv_stream_info.....613
10.8.8.4.47 hi_venc_data_type.....614
10.8.8.4.48 hi_venc_h264_nalu_type.....615
10.8.8.4.49 hi_venc_jpege_pack_type.....615
10.8.8.4.50 hi_venc_h265_nalu_type.....615
10.8.8.4.51 hi_venc_prores_pack_type.....616
10.8.8.4.52 hi_venc_chn_param.....616
10.8.8.4.53 hi_venc_stream_info.....617
10.8.8.4.54 hi_venc_pack_info.....618
10.8.8.4.55 hi_frame_rate_ctrl.....618
10.8.8.4.56 hi_venc_rc_qpmap_mode.....619
10.8.8.4.57 hi_venc_sse_info.....619
10.8.8.4.58 hi_venc_h264_attr.....620
10.8.8.4.59 hi_venc_h265_attr.....620
10.8.8.4.60 hi_venc_pic_recv_mode.....621
10.8.8.4.61 hi_venc_jpeg_attr.....621
10.8.8.4.62 hi_venc_prores_frame_rate.....622
10.8.8.4.63 hi_venc_prores_aspect_ratio.....622
10.8.8.4.64 hi_venc_prores_attr.....622
10.8.8.4.65 hi_crop_info.....622
10.8.8.4.66 hi_venc_scene_mode.....623
10.8.8.4.67 hi_venc_rc_param.....623
10.8.8.4.68 hi_venc_h264_cbr_param.....627

10.8.8.4.69 hi_venc_h264_vbr_param....628
10.8.8.4.70 hi_venc_h264_avbr_param....629
10.8.8.4.71 hi_venc_h264_qvbr_param....631
10.8.8.4.72 hi_venc_h264_cvbr_param....633
10.8.8.4.73 hi_venc_h265_vbr_param....635
10.8.8.4.74 hi_venc_h265_avbr_param....636
10.8.8.4.75 hi_venc_h265_qvbr_param....638
10.8.8.4.76 hi_venc_h265_cvbr_param....640
10.8.8.4.77 hi_venc_h265_cbr_param....642
10.8.8.4.78 hi_venc_scene_chg_detect....643
10.8.8.4.79 hi_venc_huffman_dc_table....644
10.8.8.4.80 hi_venc_huffman_ac_table....644
10.8.8.4.81 hi_venc_jpeg_huffman_param....645
10.8.8.4.82 hi_venc_mpf_cfg....645
10.8.8.4.83 hi_venc_mjpeg_cbr_param....646
10.8.8.4.84 hi_venc_mjpeg_vbr_param....646
10.8.8.4.85 hi_venc_roi_attr....646
10.8.8.4.86 hi_venc_ref_param....647
10.8.8.4.87 hi_venc_h264_vui....647
10.8.8.4.88 hi_venc_h265_vui....648
10.8.8.4.89 hi_venc_vui_aspect_ratio....648
10.8.8.4.90 hi_venc_vui_h264_time_info....649
10.8.8.4.91 hi_venc_vui_h265_time_info....650
10.8.8.4.92 hi_venc_vui_video_signal....650
10.8.8.4.93 hi_venc_vui_bitstream_restric....651
10.8.8.4.94 hi_venc_cu_prediction....651
10.8.8.4.95 hi_venc_intra_refresh....653
10.8.8.4.96 hi_venc_intra_refresh_mode....655
10.8.8.5 图像处理....655
10.8.8.5.1 hi_vpc_pic_info....655
10.8.8.5.2 hi_vpc_crop_region....656
10.8.8.5.3 hi_vpc_resize_info....656
10.8.8.5.4 hi_vpc_crop_region_info....657
10.8.8.5.5 hi_vpc_crop_resize_region....657
10.8.8.5.6 hi_vpc_crop_resize_paste_region....657
10.8.8.5.7 hi_vpc_bord_type....658
10.8.8.5.8 hi_vpc_make_border_info....659
10.8.8.5.9 hi_vpc_histogram_config....659
10.8.8.5.10 hi_vpc_lut_remap....660
10.8.8.5.11 hi_csc_conf....660
10.8.8.5.12 hi_vpc_scalar....661
10.8.8.5.13 hi_vpc_chn_attr....661

10.8.8.5.14 hi_vpc_crop_resize_border_region 662
10.8.8.5.15 hi_vpc_crop_resize_resize_paste_region 663
10.8.8.5.16 hi_vpc_workspace_param 664
10.8.8.5.17 hi_workspace_func 665
10.8.8.5.18 hi_remap_lut 665
10.8.8.5.19 hi_point_pair_info 666
10.8.8.5.20 hi_float_point 666
10.8.8.5.21 hi_warp_transform_param 667
10.8.8.5.22 hi_transform_config 667
10.8.8.5.23 hi_morph_shapes 668
10.8.8.5.24 hi_rotation 668
10.8.8.5.25 hi_blk_size 669
10.8.8.5.26 hi_median_blur_param 669
10.8.8.5.27 hi_median_blur_config 670
10.8.8.5.28 hi_blur_param 670
10.8.8.5.29 hi_blur_config 671
10.8.8.5.30 hi_gaussian_blur_param 671
10.8.8.5.31 hi_gaussian_blur_config 672
10.8.8.5.32 hi_filter_2d_param 672
10.8.8.5.33 hi_filter_2d_config 673
10.8.8.5.34 hi_mosaic_param 673
10.8.8.5.35 hi_rotate_param 674
10.8.8.5.36 hi_mosaic 674
10.8.8.5.37 hi_cover_type 675
10.8.8.5.38 hi_cover_param 675
10.8.8.5.39 hi_cover 676
10.8.8.5.40 hi_quad_cover 676
10.8.8.5.41 hi_line_param 677
10.8.8.5.42 hi_line 677
10.8.8.5.43 hi_osd_param 678
10.8.8.5.44 hi_osd 678
10.8.8.5.45 hi_osd_inverted_color 679
10.8.8.5.46 hi_transform_matrix 680
10.8.8.5.47 hi_matrix_type 680
10.8.8.5.48 hi_flip_param 681
10.8.8.5.49 hi_flip_mode 681
10.8.8.5.50 hi_map_param 682
10.8.8.6 PNGD 图像解码 682
10.8.8.6.1 hi_pngd_chn_attr 683
10.8.8.6.2 hi_img_stream 683
10.8.8.6.3 hi_pic_info 683
10.8.8.6.4 hi_png_color_format 684

10.8.9 返回码列表 684
10.8.9.1 公共返回码 684
10.8.9.2 VPC 图像处理返回码 685
10.8.9.3 VDEC 视频/JPEGD 图像解码返回码 686
10.8.9.4 VENC 视频/JPEGE 图像编码返回码 688
10.8.9.5 PNGD 图像解码返回码 690
10.9 算子加载与执行（op） 691
10.9.1 函数：set_model_dir 691
10.9.2 函数：load 692
10.9.3 函数：set_op_execute_time_out 692
10.9.4 函数：execute 693
10.9.5 函数：execute_v2 694
10.9.6 函数：execute_with_handle 696
10.9.7 函数：cast 696
10.9.8 函数：create_handle_for_cast 697
10.9.9 函数：create_attr 698
10.9.10 函数：destroy_attr 698
10.9.11 函数：set_attr_bool 698
10.9.12 函数：set_attr_int 699
10.9.13 函数：set_attr_float 699
10.9.14 函数：set_attr_string 700
10.9.15 函数：set_attr_list_bool 700
10.9.16 函数：set_attr_list_int 701
10.9.17 函数：set_attr_list_float 702
10.9.18 函数：set_attr_list_string 702
10.9.19 函数：set_attr_list_list_int 703
10.9.20 函数：create_handle 703
10.9.21 函数：destroy_handle 704
10.9.22 函数：infer_shape 704
10.9.23 函数：set_attr_data_type 706
10.9.24 函数：set_attr_list_data_type 707
10.10 CBLAS 接口（blas） 707
10.10.1 关于 A、B、C 矩阵的 Shape 及内存大小计算公式 707
10.10.2 函数：gemv_ex 708
10.10.3 函数：create_handle_for_gemv_ex 710
10.10.4 函数：hgemv 711
10.10.5 函数：create_handle_for_hgemv 712
10.10.6 函数：s8gemv 713
10.10.7 函数：create_handle_for_s8gemv 714
10.10.8 函数：gemm_ex 715
10.10.9 函数：create_handle_for_gemm_ex 717
10.10.10 函数：hgemm 718

10.10.11 函数：create_handle_for_hgemm 719
10.10.12 函数：s8gemm 720
10.10.13 函数：create_handle_for_s8gemm 722
10.11 算子编译 722
10.11.1 函数：register_compile_func 722
10.11.2 函数：unregister_compile_func 723
10.11.3 函数：create_kernel 724
10.11.4 函数：set_kernel_args 725
10.11.5 函数：set_kernel_workspace_sizes 725
10.11.6 函数：update_params 726
10.12 特征向量检索 726
10.12.1 总体说明 726
10.12.2 函数：init 727
10.12.3 函数：release 728
10.12.4 函数：repo_add 728
10.12.5 函数：repo_del 729
10.12.6 函数：delete 730
10.12.7 函数：modify 730
10.12.8 函数：search 731
10.12.9 函数：create_init_para 732
10.12.10 函数：destory_init_para 732
10.12.11 函数：set_1n_top_num 733
10.12.12 函数：set_nm_top_num 733
10.12.13 函数：create_feature_info 734
10.12.14 函数：destroy_feature_info 735
10.12.15 函数：create_repo_range 735
10.12.16 函数：destroy_repo_range 736
10.12.17 函数：create_query_table 736
10.12.18 函数：destroy_query_table 737
10.12.19 函数：create_search_input 738
10.12.20 函数：destroy_search_input 738
10.12.21 函数：create_search_result 739
10.12.22 函数：destroy_search_result 740
10.13 Profiling 数据采集 740
10.13.1 Profiling pyACL API（通过 Profiling pyACL API 采集并落盘性能数据） 740
10.13.1.1 功能介绍 740
10.13.1.2 函数：create_config 742
10.13.1.3 函数：set_config 742
10.13.1.4 函数：destroy_config 743
10.13.1.5 函数：init 744
10.13.1.6 函数：start 744
10.13.1.7 函数：stop 744

10.13.1.8 函数：finalize ... 745
10.13.2 Profiling pyACL API for Extension（Profiling pyACL API 扩展接口） ... 745
10.13.2.1 功能介绍 ... 745
10.13.2.2 函数：create_stamp ... 746
10.13.2.3 函数：set_stamp_trace_message ... 746
10.13.2.4 函数：mark ... 747
10.13.2.5 函数：push ... 748
10.13.2.6 函数：pop ... 748
10.13.2.7 函数：range_start ... 749
10.13.2.8 函数：range_stop ... 749
10.13.2.9 函数：destroy_stamp ... 750
10.13.3 Profiling pyACL API for Subscription（订阅算子信息的 Profiling pyACL API） ... 750
10.13.3.1 功能介绍 ... 751
10.13.3.2 函数：create_subscribe_config ... 751
10.13.3.3 函数：destroy_subscribe_config ... 752
10.13.3.4 函数：model_subscribe ... 753
10.13.3.5 函数：model_un_subscribe ... 753
10.13.3.6 函数：get_op_desc_size ... 754
10.13.3.7 函数：get_op_num ... 754
10.13.3.8 函数：get_op_type_len ... 755
10.13.3.9 函数：get_op_name_len ... 755
10.13.3.10 函数：get_op_type ... 756
10.13.3.11 函数：get_op_type_v2 ... 757
10.13.3.12 函数：get_op_name ... 757
10.13.3.13 函数：get_op_name_v2 ... 758
10.13.3.14 函数：get_op_start ... 759
10.13.3.15 函数：get_op_end ... 759
10.13.3.16 函数：get_op_duration ... 760
10.13.3.17 函数：get_model_id ... 760
10.13.4 Profiling pyACL API（PyTorch 场景标记迭代时间） ... 761
10.13.4.1 函数：get_step_timestamp ... 761
10.13.4.2 函数：create_step_info ... 762
10.13.4.3 函数：destroy_step_info ... 762
10.14 数据类型及其操作接口 ... 762
10.14.1 aclError ... 762
10.14.2 aclDataType ... 780
10.14.3 aclFloat16 ... 780
10.14.4 aclFormat ... 781
10.14.5 acldvppPixelFormat ... 782
10.14.6 acldvppStreamFormat ... 784
10.14.7 acldvppJpegFormat ... 784
10.14.8 acldvppCscMatrix ... 785

10.14.9 aclrtEventStatus 787
10.14.10 aclrtRunMode 787
10.14.11 aclTransType 787
10.14.12 aclComputeType 787
10.14.13 aclfvSearchType 787
10.14.14 aclAippInputFormat 788
10.14.15 aclAippInfo 788
10.14.16 aclAippDims 791
10.14.17 aclmdlIODims 791
10.14.18 aclrtGroupAttr 791
10.14.19 aclprofAicoreMetrics 792
10.14.20 acldvppBorderType 793
10.14.21 acldvppJpegdPrecisionMode 793
10.14.22 aclmdlInputAippType 794
10.14.23 acltdtTensorType 794
10.14.24 acldvppChannelDescParamType 795
10.14.25 aclvdecChannelDescParamType 795
10.14.26 aclvencChannelDescParamType 797
10.14.27 aclmdlConfigAttr 798
10.14.28 aclMemType 800
10.14.29 aclOpCompileFlag 800
10.14.30 aclrtEventWaitStatus 801
10.14.31 aclrtEventRecordedStatus 801
10.14.32 aclprofStepTag 801
10.14.33 acltdtQueueAttrType 801
10.14.34 acltdtQueueRouteParamType 802
10.14.35 acltdtQueueRouteQueryMode 802
10.14.36 acltdtQueueRouteQueryInfoParamType 802
10.14.37 acldvppChannelMode 802
10.14.38 aclrtFloatOverflowMode 803

11 附录 804

11.1 pyACL 表达约定 804
11.2 使用约束 805

1 学习向导

概述

本文用于指导开发人员基于现有模型、使用pyACL提供的Python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。

通过本文档您可以达成：

- 了解pyACL的功能架构、基本概念以及接口的典型调用流程。
- 使用pyACL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。

掌握以下经验和技能可以更好地理解本文档：

- 具备Python语言程序开发能力。
- 对机器学习、深度学习有一定的了解。

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，建议先从了解**3 概述**，再通过**开发基础推理应用**、**单算子调用**等章节的接口调用流程+示例代码来深入学习。

- pyACL在整体架构的什么位置?
- pyACL中的Device、Stream、Context是用来做什么的?
- 使用pyACL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pyACL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pyACL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

在本章节中，您可以通过各功能点介绍以及关键接口代码示例了解使用pyACL接口（Python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。

开发场景

开发图片分类应用，对2张分辨率为1024*683的*.jpg图片分类。若想直接运行该样例，请参见**基于Caffe ResNet-50网络实现图片分类（同步推理）**。

该场景涉及以下功能点，关键代码请参见**关键代码开发**：

功能点	功能简介	关键接口调用流程
pyACL初始化与去初始化	使用pyACL接口开发应用时，必须先初始化，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。 有初始化就有去初始化，在确定完成了pyACL的所有调用之后，或者进程退出之前，需调用接口实现pyACL去初始化。 接口调用流程及示例请参见**4.3 pyACL初始化与去初始化**。	• 调用**acl.init**接口初始化pyACL配置。 • 调用**acl.finalize**接口实现pyACL去初始化。

功能点	功能简介	关键接口调用流程
运行管理资源申请与释放	您需要按顺序依次申请如下资源：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，需要释放运行管理资源，包括Stream、Context、Device。释放资源时，需要对应顺序释放：Stream，Context，Device。 Stream、Context、Device资源的介绍请参见**表3-1**。 功能接口调用流程及示例请参见**4.4 运行管理资源申请与释放**。	• 调用**acl.rt.set_device**接口指定用于运算的Device。 • 调用**acl.rt.get_run_mode**接口获取昇腾AI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**acl.rt.reset_device**接口复位当前运算的Device，回收Device上的资源。
		• 调用**acl.rt.create_context**接口创建Context。 • 调用**acl.rt.destroy_context**接口销毁Context。
		• 调用**acl.rt.create_stream**接口创建Stream。 • 调用**acl.rt.destroy_stream**接口销毁Stream。
内存申请与释放	调用pyACL提供的接口申请或释放内存。	• 调用**acl.rt.malloc**接口申请Device上的内存。 • 调用**acl.rt.free**接口释放Device上的内存。
数据传输	通过内存复制的方式实现数据传输。 功能接口调用流程及示例请参见**4.5 数据传输**。	如果在板端环境上运行应用，则无需进行数据传输。
模型推理	本文介绍的场景是图片分类，因此需要选取开源的分类网络，此处选择的是Caffe ResNet-50网络，将开源的ResNet-50网络转换为适配昇腾AI处理器的离线模型（*.om文件），使用该离线模型推理图片所属的类别。 ResNet-50网络对输入图片宽高的要求为224*224，且要求输入图片格式为RGB。但当前输入图片是*.jpg格式，因此下文的样例中使用Python库对图片进行预处理。 接口调用流程及示例请参见**4.6 模型推理基本场景**。	• 使用ATC（Ascend Tensor Compiler）工具将ResNet-50网络转换为*.om文件（适配昇腾AI处理器的离线模型）。请参见《**ATC工具使用指南**》。 • 调用**acl.mdl.load_from_file**接口从*.om文件加载模型。 • 调用**acl.mdl.execute**接口执行模型推理，同步接口。 • 调用**acl.mdl.unload**接口卸载模型。

功能点	功能简介	关键接口调用流程
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。	• 调用**acl.mdl.get_dataset_num_buffers**接口从输出aclmdlDataset获取aclDataBuffer的个数num。 • 循环遍历输出aclmdlDataset获取aclDataBuffer数据： – 调用**acl.mdl.get_dataset_buffer**接口获取aclDataBuffer。 – 调用**acl.get_data_buffer_addr**接口获取aclDataBuffer中数据。 – 调用**acl.get_data_buffer_size_v2**获取aclDataBuffer中数据大小。 – 调用**acl.rt.malloc_host**和**acl.rt.memcpy**接口做数据搬移，从Device->Host。 – 调用**acl.util.ptr_to_numpy**接口将指针转换numpy对象：使用struct包处理numpy对象，完成top5的置信。 `top_k = np.array(struct.unpack("1000f", bytearray(result))).flatten().argsort()[-1:-6:-1]`

关键代码开发

即使您不理解所有细节也没关系，此处仅仅是关键步骤的代码示例，不可以直接拷贝运行，仅供参考，旨在帮助您快速了解整体的代码逻辑。若想获取该样例的完整代码，请参见**9 应用样例参考**。

步骤1 导入acl模块。

```
import acl
```

步骤2 pyACL初始化。

```
ret = acl.init()
```

步骤3 运行管理资源申请。

```
self.device_id = 0

# 指定运算的Device。
ret = acl.rt.set_device(self.device_id)

# 显式创建一个Context，用于管理Stream对象。
self.context, ret = acl.rt.create_context(self.device_id)
```

步骤4 加载模型，并获取模型描述信息。

在模型加载前，需要将第三方网络（例如，Caffe ResNet-50网络）转换为适配昇腾AI处理器的离线模型（*.om文件），请参见《**ATC工具使用指南**》。

```
# 初始化变量。
self.model_path = './model/resnet50.om'

# 加载离线模型文件，返回标识模型的ID。
self.model_id, ret = acl.mdl.load_from_file(self.model_path)

# 根据加载成功的模型的ID，获取该模型的描述信息。
self.model_desc = acl.mdl.create_desc()
ret = acl.mdl.get_desc(self.model_desc, self.model_id)
```

步骤5 准备模型推理的输入、输出数据结构。

```
# 初始化变量。
ACL_MEM_MALLOC_HUGE_FIRST = 0

# 1.准备模型推理的输入数据集。
# 创建aclmdlDataset类型的数据，描述模型推理的输入。
self.load_input_dataset = acl.mdl.create_dataset()
# 获取模型输入的数量。
input_size = acl.mdl.get_num_inputs(self.model_desc)
self.input_data = []
# 循环为每个输入申请内存，并将每个输入添加到aclmdlDataset类型的数据中。
for i in range(input_size):
    buffer_size = acl.mdl.get_input_size_by_index(self.model_desc, i)
    # 申请输入内存。
    buffer, ret = acl.rt.malloc(buffer_size, ACL_MEM_MALLOC_HUGE_FIRST)
    data = acl.create_data_buffer(buffer, buffer_size)
    _, ret = acl.mdl.add_dataset_buffer(self.load_input_dataset, data)
    self.input_data.append({"buffer": buffer, "size": buffer_size})

# 2.准备模型推理的输出数据集。
# 创建aclmdlDataset类型的数据，描述模型推理的输出。
self.load_output_dataset = acl.mdl.create_dataset()
# 获取模型输出的数量。
output_size = acl.mdl.get_num_outputs(self.model_desc)
self.output_data = []
# 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 i in range(output_size):
    buffer_size = acl.mdl.get_output_size_by_index(self.model_desc, i)
    # 申请输出内存。
    buffer, ret = acl.rt.malloc(buffer_size, ACL_MEM_MALLOC_HUGE_FIRST)
    data = acl.create_data_buffer(buffer, buffer_size)
    _, ret = acl.mdl.add_dataset_buffer(self.load_output_dataset, data)
    self.output_data.append({"buffer": buffer, "size": buffer_size})
```

步骤6 准备模型推理的输入数据，并进行推理。

```
ACL_MEMCPY_HOST_TO_DEVICE = 1
ACL_MEMCPY_DEVICE_TO_HOST = 2
NPY_BYTE = 1
images_list = ["./data/dog1_1024_683.jpg", "./data/dog2_1024_683.jpg"]

for image in images_list:
    # 1.自定义函数transfer_pic，使用Python库读取图片文件，并对图片进行缩放、剪裁等操作，transfer_pic函数的实现请参考样例中源代码。
    img = transfer_pic(image)

    # 2.准备模型推理的输入数据，运行模式为ACL_HOST，当前实例代码中模型只有一个输入。

    bytes_data = img.tobytes()
    np_ptr = acl.util.bytes_to_ptr(bytes_data)
    # 将图片数据从Host传输到Device。
    ret = acl.rt.memcpy(self.input_data[0]["buffer"], self.input_data[0]["size"], np_ptr,
                        self.input_data[0]["size"], ACL_MEMCPY_HOST_TO_DEVICE)

    # 3.执行模型推理。
    # self.model_id表示模型ID，在模型加载成功后，会返回标识模型的ID。
    ret = acl.mdl.execute(self.model_id, self.load_input_dataset, self.load_output_dataset)
```

```
    # 4.处理模型推理的输出数据，输出top5置信度的类别编号。
    inference_result = []
    for i, item in enumerate(self.output_data):
        buffer_host, ret = acl.rt.malloc_host(self.output_data[i]["size"])
        # 将推理输出数据从Device传输到Host。
        ret = acl.rt.memcpy(buffer_host, self.output_data[i]["size"], self.output_data[i]["buffer"],
                            self.output_data[i]["size"], ACL_MEMCPY_DEVICE_TO_HOST)

        bytes_out = acl.util.ptr_to_bytes(buffer_host, self.output_data[i]["size"])
        data = np.frombuffer(bytes_out, dtype=np.byte)
        inference_result.append(data)
    tuple_st = struct.unpack("1000f", bytearray(inference_result[0]))
    vals = np.array(tuple_st).flatten()
    top_k = vals.argsort()[-1:-6:-1]
    print("======== top5 inference results: =============")
    for j in top_k:
        print("[%d]: %f" % (j, vals[j]))

# 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。
while self.input_data:
    item = self.input_data.pop()
    ret = acl.rt.free(item["buffer"])
input_number = acl.mdl.get_dataset_num_buffers(self.load_input_dataset)
for i in range(input_number):
    data_buf = acl.mdl.get_dataset_buffer(self.load_input_dataset, i)
    if data_buf:
        ret = acl.destroy_data_buffer(data_buf)
ret = acl.mdl.destroy_dataset(self.load_input_dataset)

# 释放输出资源，包括数据结构和内存。
while self.output_data:
    item = self.output_data.pop()
    ret = acl.rt.free(item["buffer"])
output_number = acl.mdl.get_dataset_num_buffers(self.load_output_dataset)
for i in range(output_number):
    data_buf = acl.mdl.get_dataset_buffer(self.load_output_dataset, i)
    if data_buf:
        ret = acl.destroy_data_buffer(data_buf)
ret = acl.mdl.destroy_dataset(self.load_output_dataset)
```

步骤7 卸载模型，并释放模型描述信息。

```
# 卸载模型。
ret = acl.mdl.unload(self.model_id)

# 释放模型描述信息。
if self.model_desc:
    ret = acl.mdl.destroy_desc(self.model_desc)
    self.model_desc = None
```

步骤8 运行管理资源释放。

```
# 释放Context。
if self.context:
    ret = acl.rt.destroy_context(self.context)
    self.context = None

# 释放Device。
ret = acl.rt.reset_device(self.device_id)
```

步骤9 pyACL去初始化。

```
ret = acl.finalize()
```

----结束

3 概述

基于AscendCL的pyACL

基本概念

pyACL开发流程

pyACL接口调用流程

注意事项

应用开发环境准备

3.1 基于 AscendCL 的 pyACL

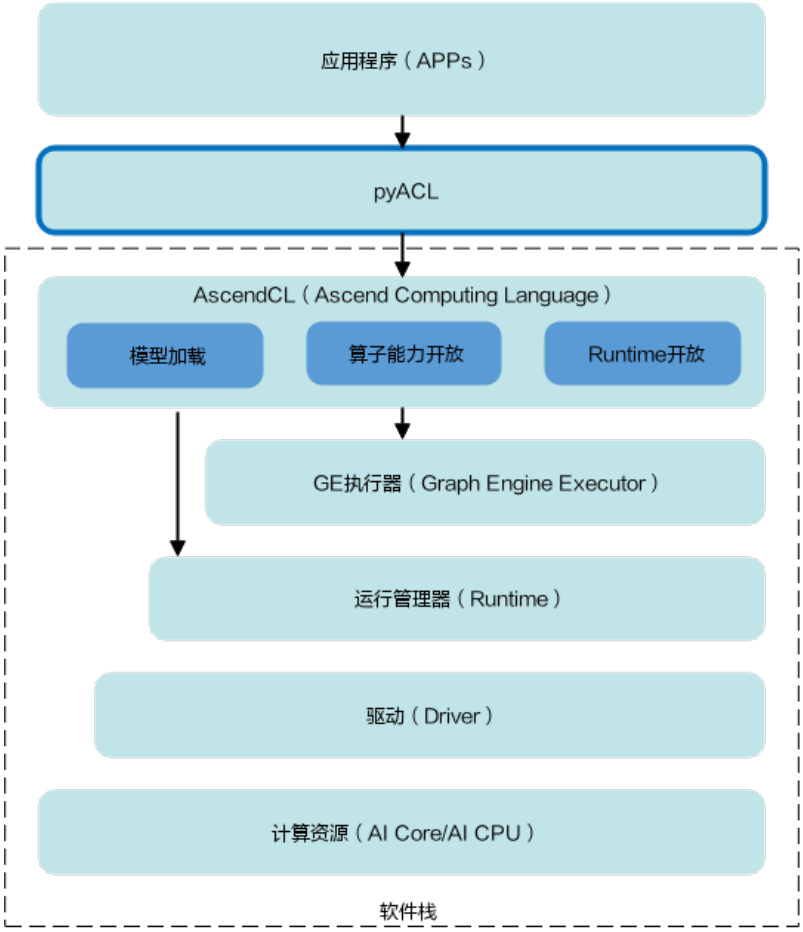
AscendCL（Ascend Computing Language）是一套用于在昇腾平台上开发深度神经网络推理应用的C语言API库，提供Device管理、Context管理、Stream管理、内存管理、模型加载与执行、算子加载与执行、媒体数据处理等C语言API库供用户开发深度神经网络应用，用于实现目标识别、图像分类等功能。

在运行应用时，AscendCL调用GE执行器提供的接口实现模型和算子的加载与执行、调用运行管理器的接口实现Device管理、Context管理、Stream管理、内存管理等。

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。

pyACL（Python Ascend Computing Language）就是在AscendCL的基础上使用CPython封装得到的Python API库，使用户可以通过Python进行昇腾AI处理器的运行管理、资源管理等。

图 3-1 逻辑架构图



3.2 基本概念

表 3-1 概念介绍

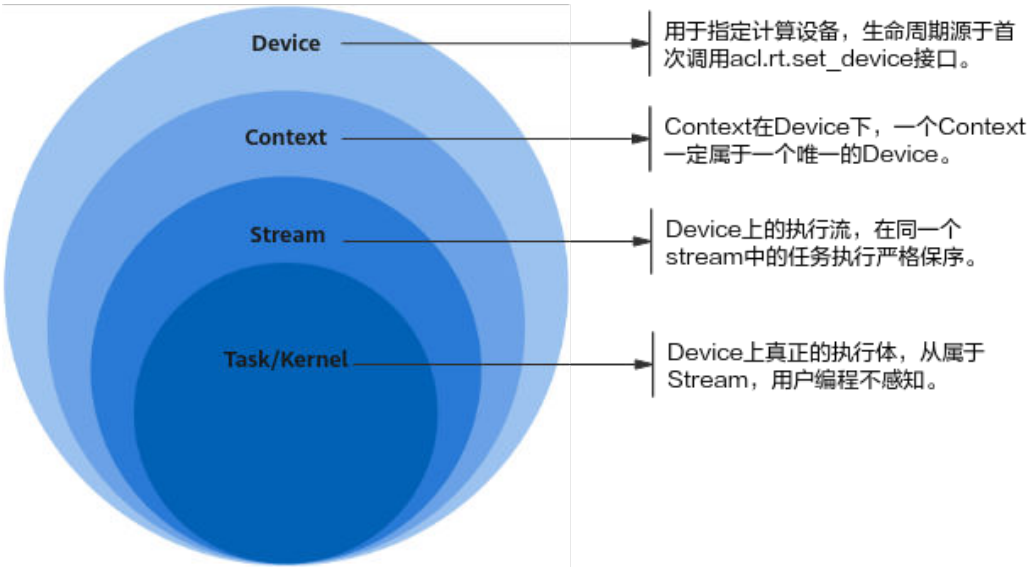
概念	描述
同步/异步	本文中提及的同步、异步是站在调用者和执行者的角度，在当前场景下，若在Host调用接口后不等待Device执行完成再返回，则表示Host的调度是异步的；若在Host调用接口后需等待Device执行完成再返回，则表示Host的调度是同步的。
进程/线程	本文中提及的进程、线程，若无特别注明，则表示Host上的进程、线程。
Host	Host指与Device相连接的X86服务器、ARM服务器，会利用Device提供的NN（Neural-Network）计算能力，完成业务。
Device	Device指安装了昇腾AI处理器的硬件设备，利用PCIe接口与Host侧连接，为Host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。

概念	描述
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**acl.rt.set_device**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**acl.rt.destroy_context**接口来释放。 ● 显式创建的Context：**推荐**，在进程或线程中调用**acl.rt.create_context**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**acl.rt.set_device**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**acl.rt.destroy_stream**接口来释放。 ● 显式创建的Stream：**推荐**，在进程或线程中调用**acl.rt.create_stream**接口显式创建一个Stream。
Event	支持调用pyACL接口同步Stream之间的任务，包括同步Host与Device之间的任务、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。

概念	描述
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其中一种方式来处理图片，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。
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。
RC模式	以昇腾 AI 处理器的PCIe的工作模式进行区分，如果PCIe工作在主模式，可以扩展外设，则称为RC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**acl.rt.set_device**接口。
  - 每次调用**acl.rt.set_device**接口，系统会进行引用计数加1；调用**acl.rt.reset_device**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**acl.rt.set_device**接口，终结于调用**acl.rt.reset_device**接口使引用计数为零时。隐式Context只会被创建一次，调用**acl.rt.set_device**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**acl.rt.create_context**接口，终结于调用**acl.rt.destroy_context**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**acl.rt.create_stream**），当前线程在同一时刻内只能使用其中一个Context，建议通过**acl.rt.set_context**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**acl.rt.set_context**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**acl.rt.create_stream**，终结于调用**acl.rt.destroy_stream**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**acl.rt.set_context**进行Device的快速切换。示例代码如下，仅供参考，不可以直接拷贝运行：

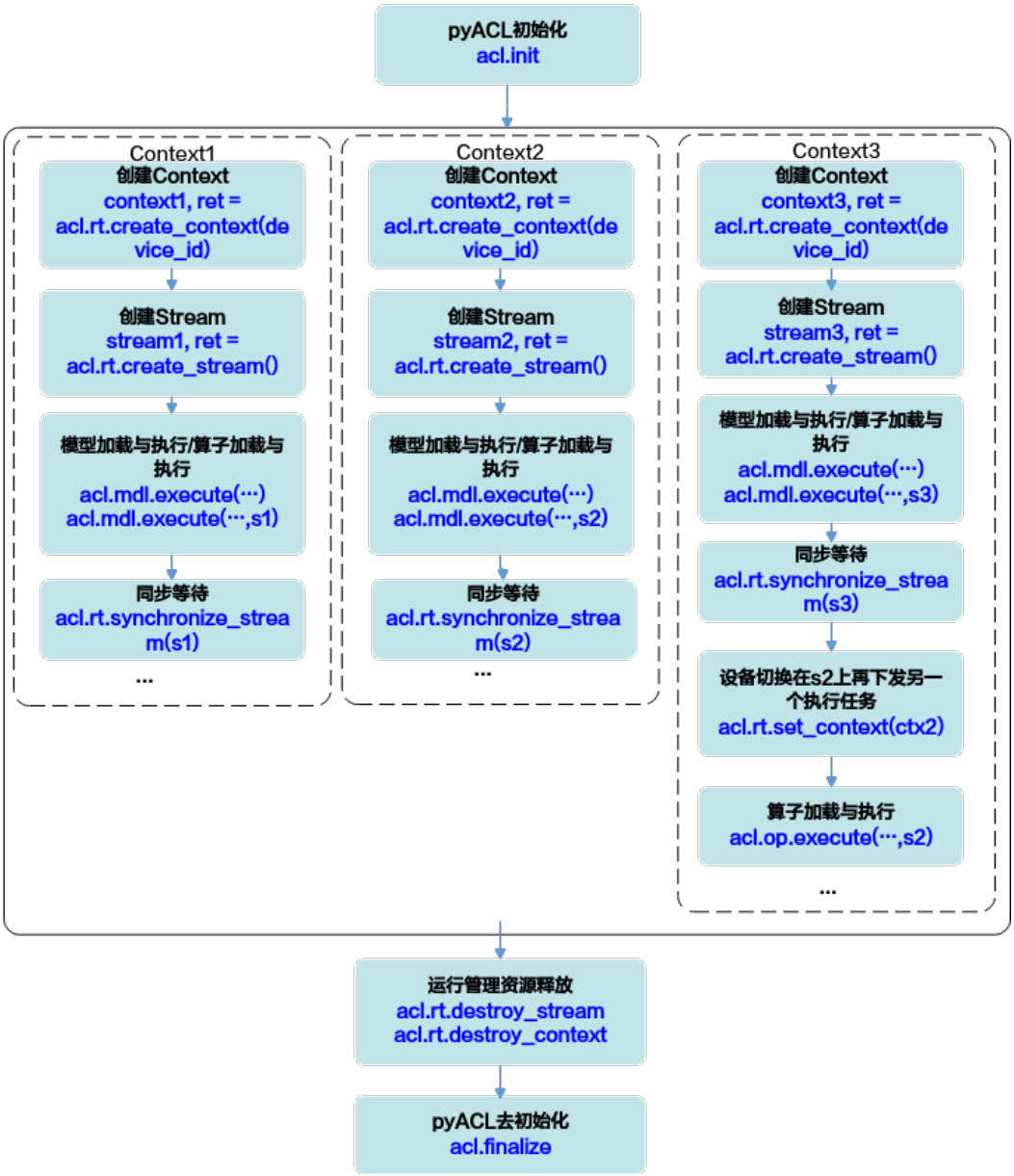
```
...
ctx1, ret = acl.rt.create_context(0)
stream, ret = acl.rt.create_stream()
ret = acl.op.execute_v2(op_type, input_desc, inputs, output_desc, outputs, attr, stream)
ctx2, ret = acl.rt.create_context(1)

# 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。
stream2, ret = acl.rt.create_stream()
ret = acl.op.execute_v2(op_type2, input_desc, inputs, output_desc, outputs, attr, stream2)
ret = acl.rt.set_context(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。
ret = acl.op.execute_v2(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，也可以每个线程创建一个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调度Device侧，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**acl.rt.set_context**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0。
- 默认Context不允许用户执行**acl.rt.get_context**或**acl.rt.set_context**操作，也不允许执行**acl.rt.destroy_context**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，不可以直接拷贝运行：

```
# …
ret = acl.init(config_path)
```

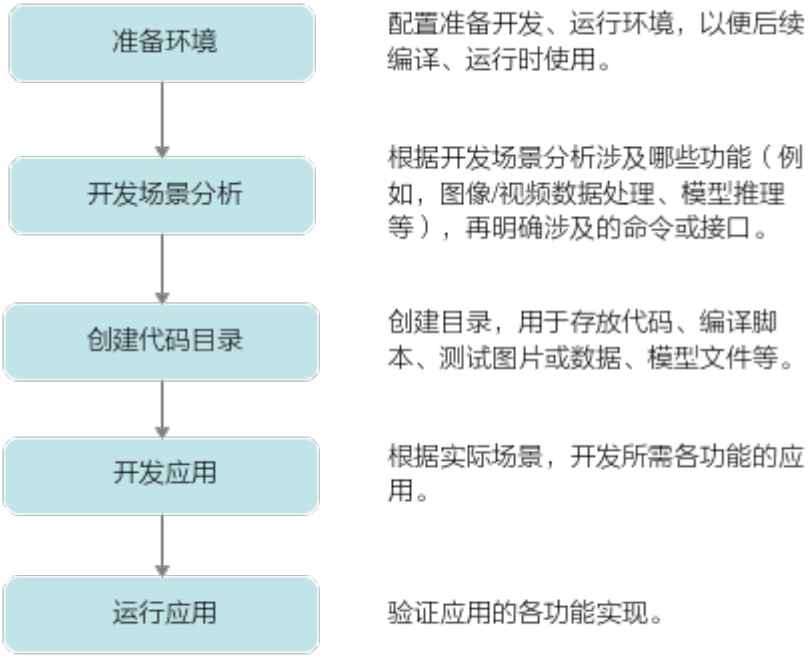
```
ret = acl.rt.set_device(device_id)

# 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。
# …
ret = acl.op.execute(op1, input_desc, inputs, output_desc, outputs, attr, 0)  # 最后一个0表示在default stream上执行算子op1。
ret = acl.op.execute(op2, input_desc, inputs, output_desc, outputs, attr, 0)  # 最后一个0表示在default stream上执行算子op2。
ret = acl.rt.synchronize_stream(0)

# 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。
# …
ret = acl.rt.reset_device(device_id)  # 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。

3.3 pyACL 开发流程

图 3-4 开发流程



3.4 pyACL 接口调用流程

调用pyACL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pyACL接口开发AI应用的整体接口调用流程。

图 3-5 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，则需要数据预处理，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. pyACL初始化。

   调用**acl.init**接口实现初始化pyACL。

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**运行管理资源申请流程**。

3. 模型推理/单算子调用/媒体数据处理。

- 模型推理。
  i. 生成模型om文件：模型推理场景下，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. 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**。
  iii. （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 图像/视频数据处理**。
  iv. 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**。
  v. （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。
  vi. 模型卸载：调用**acl.mdl.unload**接口卸载模型。
- 算子调用。
  接口调用流程，请参见**6.3 接口调用流程**。
  - 生成算子om文件，需使用ATC工具将算子定义文件（*.json）编译成适配昇腾AI处理器的离线模型（*.om文件），请参见《**ATC工具使用指南**》。
  - 加载算子om文件，运行算子时使用。
  - 执行算子，输出算子的运行结果。

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**。
   具体流程，请参见**运行管理资源释放流程**。
5. pyACL去初始化。
   调用**acl.finalize**接口实现pyACL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**10.5.5 内存管理**。数据类型的创建与销毁的接口请参见**10.14 数据类型及其操作接口**。

3.5 注意事项

- 关于日志的处理机制和日志级别设置等功能，请参见《**日志参考**》。
- pyACL部分功能可能会涉及到ATC工具来进行模型转换，请参见《**ATC工具使用指南**》。
- 生成pyACL中的“acl.so”依赖的Python版本范围为3.7.5~3.9.2。

3.6 应用开发环境准备

部署开发环境和运行环境，请参见对应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

  - pyACL库文件路径：CANN软件安装后文件存储路径/lib64

    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。

    安装CANN软件后，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。
  - 本文中的操作步骤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。

- （可选）通过环境变量“ASCEND_RT_VISIBLE_DEVICES”设置Device ID，指定应用进程可用的Device。支持一次指定一个或多个Device ID。通过设置该环境变量，可以实现不修改应用程序、但调整Device的功能。

  示例场景：例如板端环境上的可用Device数量为8，Device ID分别为：0、1、2、3、4、5、6、7。

  – 指定一个Device ID，示例表示应用进程可使用Device ID为1的Device

    ```
    # acl.rt.get_device_count接口获取到的可用Device数量为1，acl.rt.set_device(0)时，索引0对应的Device ID是1
    export ASCEND_RT_VISIBLE_DEVICES = 1
    ```

  – 指定多个Device ID，以下两个均示例表示应用进程可使用Device ID为2,3,4的Device， Device ID的顺序可以任意设置，但顺序会影响Device ID的索引值。

    ```
    # acl.rt.get_device_count接口获取到的可用Device数量为3，acl.rt.set_device(0)时，索引0对应的Device ID是2
    export ASCEND_RT_VISIBLE_DEVICES = 2,3,4
    # acl.rt.get_device_count接口获取到的可用Device数量为3，acl.rt.set_device(0)时，索引0对应的Device ID是4
    export ASCEND_RT_VISIBLE_DEVICES = 4,3,2
    ```

  – 指定多个Device ID，但出现无效值时，则仅无效值之前的Device可用。示例中仅2和3可用。

    ```
    # acl.rt.get_device_count接口获取到的可用Device数量为2，acl.rt.set_device(0)时，索引0对应的Device ID是2
    export ASCEND_RT_VISIBLE_DEVICES = 2,3,-1,5
    ```

  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：

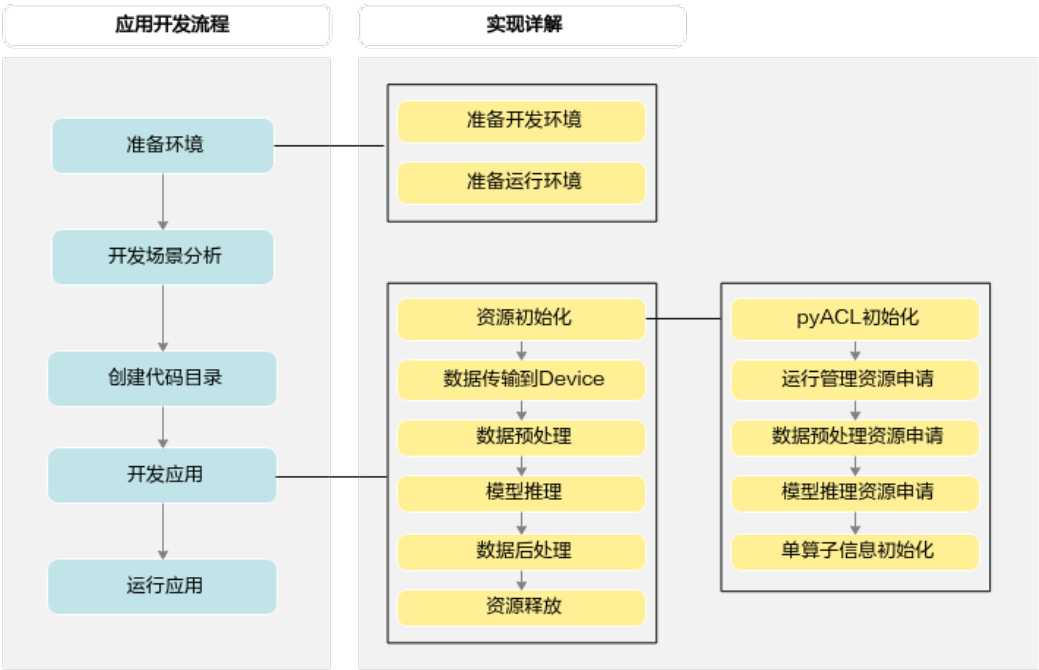
  a. 以安装用户在任意目录下执行**vi ~/.bashrc**，在该文件最后添加上述内容。
  b. 执行**:wq!**命令保存文件并退出。
  c. 执行**source ~/.bashrc**使环境变量生效。

4 开发基础推理应用

开发流程
模型构建
pyACL初始化与去初始化
运行管理资源申请与释放
数据传输
模型推理基本场景

4.1 开发流程

图 4-1 开发流程



版权所有 © 华为技术有限公司

1. 准备环境，包括开发环境和运行环境。
2. 开发场景分析。

   根据开发场景分析涉及哪些功能（例如，数据传输、模型推理等）的开发，确定功能后，再明确涉及的命令或接口。
3. 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，可参考：

```
├App名称
├── caffe_model      # 该目录下存放模型转换相关的配置文件、模型文件。
│   ├── xxx.cfg
│   ├── xxx.prototxt
├── data
│   ├── xxx.jpg     # 测试数据。
│
├── model
│   ├── xxx.om      # 转换后的模型文件。
│
├── xxx.py          # python脚本。
├── xxx.py
```

4. 开发应用。
   a. pyACL初始化，请参见**4.3 pyACL初始化与去初始化**。

      使用pyACL接口开发应用时，必须先调用**acl.init**接口进行pyACL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 模型推理基本场景**。

      模型推理结束后，需及时释放相关资源。

      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。
   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pyACL去初始化，请参见**4.3 pyACL初始化与去初始化**。
5. 运行应用，包括模型转换、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Caffe、TensorFlow等），不能直接在昇腾AI处理器上运行推理，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），模型转换的方法请参见《**转换模型**》。

4.3 pyACL 初始化与去初始化

关于pyACL初始化与去初始化的接口调用流程，请参见**3.4 pyACL接口调用流程**。

基本原理

您必须调用**acl.init**接口初始化pyACL，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，acl.init接口中可不传入参数，或者可将配置文件配置为空json串（即配置文件中只有{}）。在acl.init接口中不传入参数，示例如下：

```
ret = acl.init()
```

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

示例代码

您可以从**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，不可以直接拷贝运行，仅供参考。

```
import acl
# ......
# 初始化基本配置。
# 此处的“..”表示相对路径，相对可执行文件所在的目录。
acl_config_path = "../src/acl.json"
ret = acl.init(acl_config_path)
# ......

# 去初始化。
ret = acl.finalize()
# ......
```

4.4 运行管理资源申请与释放

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**3.4 pyACL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。

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。

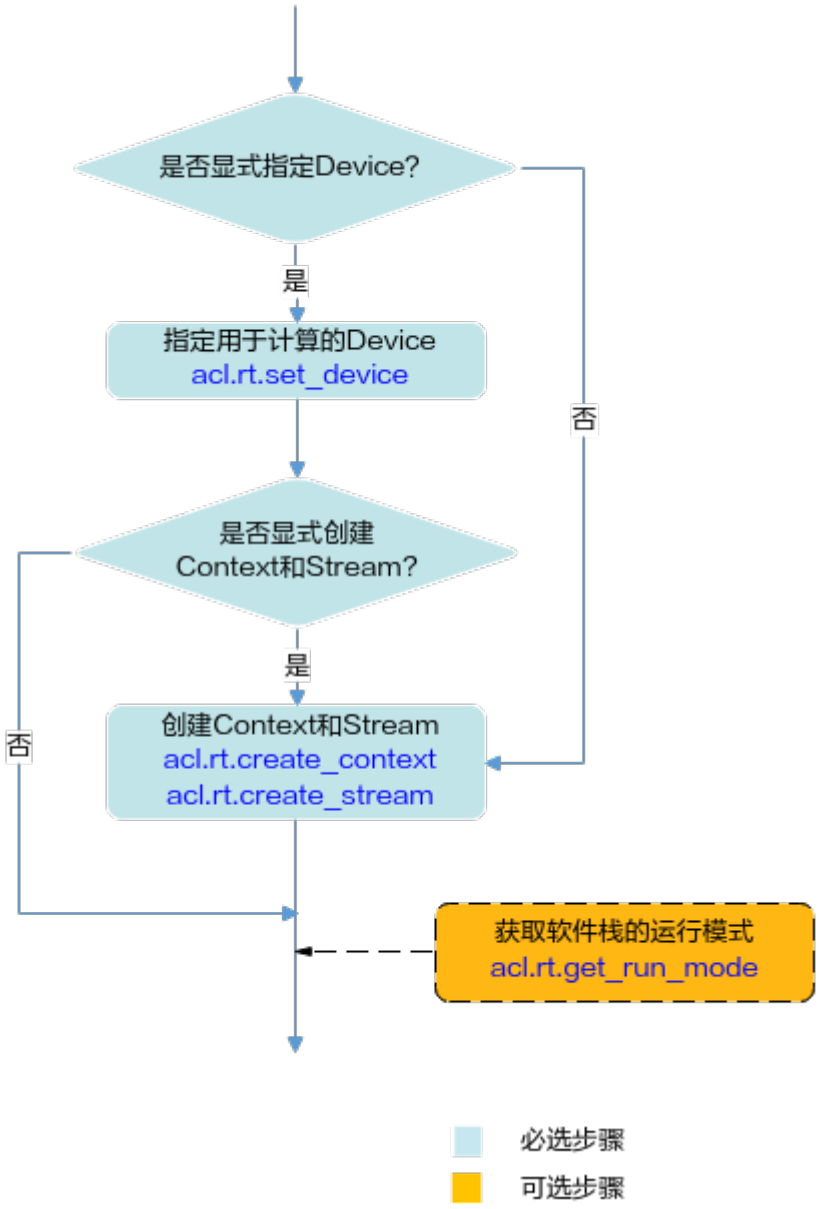
您需要按照Device、Context、Stream的顺序依次申请。其中，创建Context、Stream的方式分为隐式创建和显式创建，其适用场景有所不同：

- 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**7.8 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**acl.rt.set_device**接口显式指定用于运算的Device。
     - 调用**acl.rt.create_context**接口显式创建Context，调用**acl.rt.create_stream**接口显式创建Stream。
     - 如果不显式创建Context和Stream，您可以使用**acl.rt.set_device**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**acl.rt.destroy_context**接口来释放。

- 一个Device对应一个默认Stream，默认Stream不能通过**acl.rt.destroy_stream**接口来释放。默认Stream作为接口入参时，直接传0。
- 默认Context、默认Stream，是在调用**acl.rt.reset_device**接口后自动释放。

- 不显式指定用于运算的Device。

  调用**acl.rt.create_context**接口显式创建Context，调用**acl.rt.create_stream**接口显式创建Stream。系统在显式创建Context时，系统内部会调用**acl.rt.set_device**接口指定运行的Device，Device ID通过**acl.rt.create_context**接口传入。

2. （可选）调用**acl.rt.get_run_mode**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。

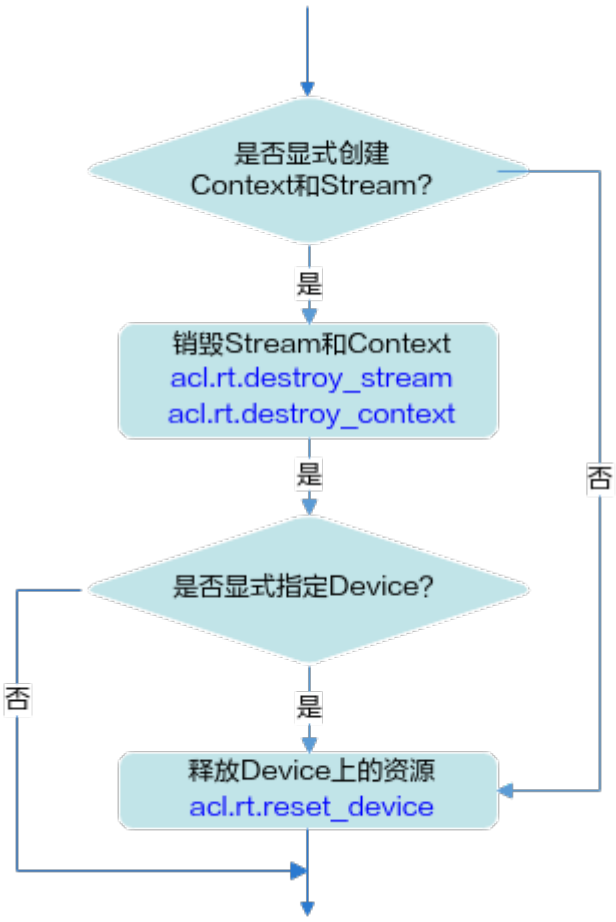
   如果查询结果为ACL_HOST，则数据传输时涉及申请Host上的内存。

   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**acl.rt.destroy_stream**接口释放Stream，再调用**acl.rt.destroy_context**接口释放Context。若显式调用**acl.rt.set_device**接口指定运算的Device时，还需调用**acl.rt.reset_device**接口释放Device上的资源。
- 不显式创建Context和Stream时，仅需调用**acl.rt.reset_device**接口释放Device上的资源。

示例代码

您可以从**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，不可以直接拷贝运行，仅供参考。

```
import acl
#......

# ======运行管理资源申请======
# 1.指定运算的Device。
ret = acl.rt.set_device(device_id)

# 2.显式创建一个Context，用于管理Stream对象。
context, ret = acl.rt.create_context(device_id)

# 3.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。
stream, ret = acl.rt.create_stream()
# ======运行管理资源申请======

#......

# ======运行管理资源释放======
ret = acl.rt.destroy_stream(stream)
ret = acl.rt.destroy_context(context)
ret = acl.rt.reset_device(device_id)
# ======运行管理资源释放======

#......
```

4.5 数据传输

4.5.1 接口调用流程

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。
   - Device上的内存，使用pyACL提供的**acl.rt.malloc**或**acl.rt.malloc_host**接口申请内存。如果涉及数据预处理（例如，图片解码、缩放等）时，需使用**acl.media.dvpp_malloc**或**acl.himpi.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**acl.rt.memcpy**接口。
- 异步内存复制：调用**acl.rt.memcpy_async**接口，再调用**acl.rt.synchronize_stream**接口实现Stream内任务的同步等待。
- 调用同步或异步内存复制接口时，支持以下类型的复制（可点击链接查看对应示例代码）：
  - **Device到Host的内存复制**
  - **一个Deviece内的内存复制**

说明

Ascend RC场景下，不涉及Host上的内存申请、Host内的数据传输、Host与Device之间的数据传输。

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**acl.rt.malloc_host**接口申请Host内存、使用**acl.rt.malloc**接口申请Device内存时：

  需先调用**acl.rt.get_run_mode**接口获取软件栈的运行模式。

  - 当查询结果为**ACL_HOST = 1**，则数据传输时涉及申请Host上的内存。
  - 当查询结果为**ACL_DEVICE = 0** ，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**acl.rt.malloc_host**接口（该接口支持申请Host或Device内存），pyACL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**acl.rt.get_run_mode**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。

4.5.2 Device 内的数据传输

同步内存复制

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，不可以直接拷贝运行，仅供参考，示例代码如下：

```
import acl
# ......

# 1.申请内存。
size = 1 * 1024 * 1024
dev_ptr_a, ret = acl.rt.malloc(size, ACL_MEM_MALLOC_NORMAL_ONLY)
dev_ptr_b, ret = acl.rt.malloc(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read_file由用户实现。
read_file(fileName, dev_ptr_a, size)

# 3.同步内存复制。
# 同步内存复制，dev_ptr_a表示Device上源内存地址指针，dev_ptr_b表示Device上目的内存地址指针，size表示内存大小。
# ACL_MEMCPY_DEVICE_TO_DEVICE = 3。
ret = acl.rt.memcpy(dev_ptr_b, size, dev_ptr_a, size, ACL_MEMCPY_DEVICE_TO_DEVICE)

# 4.使用完内存中的数据后，需及时释放资源。
ret = acl.rt.free(dev_ptr_a)
ret = acl.rt.free(dev_ptr_b)

# ......
```

异步内存复制

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，不可以直接拷贝运行，仅供参考，示例代码如下：

```
import acl
# ......

# 1.申请内存。
size = 1 * 1024 * 1024
dev_ptr_a, ret = acl.rt.malloc(size, ACL_MEM_MALLOC_NORMAL_ONLY)
dev_ptr_b, ret = acl.rt.malloc(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read_file由用户实现。
read_file(fileName, dev_ptr_a, size)

# 3.异步内存复制。
# 异步内存复制，dev_ptr_a表示Device上源内存地址指针，dev_ptr_b表示Device上目的内存地址指针，size表示内存大小。
# ACL_MEMCPY_DEVICE_TO_DEVICE = 3。
stream = acl.rt_create_stream()
ret = acl.rt.memcpy_async(dev_ptr_b, size, dev_ptr_a, size, ACL_MEMCPY_DEVICE_TO_DEVICE, stream)
ret = acl.rt.synchronize_stream(stream)

# 4.使用完内存中的数据后，需及时释放资源。
ret = acl.rt.destroy_stream(stream)
ret = acl.rt.free(dev_ptr_a)
ret = acl.rt.free(dev_ptr_b)

# ......
```

4.5.3 从 Device 到 Host 的数据传输

同步内存复制

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，不可以直接拷贝运行，仅供参考，示例代码如下：

```
import acl
# ......

# 1.申请内存。
size = 1 * 1024 * 1024
host_ptr_b, ret = acl.rt.malloc_host(size)
dev_ptr_a, ret = acl.rt.malloc(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read_file由用户实现。
read_file(fileName, dev_ptr_a, size)

# 3.同步内存复制。
# 同步内存复制，dev_ptr_a表示Device上源内存地址指针，host_ptr_b表示Host上目的内存地址指针，size表示内存大小。
# ACL_MEMCPY_DEVICE_TO_HOST = 2。
ret = acl.rt.memcpy(host_ptr_b, size, dev_ptr_a, size, ACL_MEMCPY_DEVICE_TO_HOST)

# 4.使用完内存中的数据后，需及时释放资源。
ret  = acl.rt.free_host(host_ptr_b)
ret = acl.rt.free(dev_ptr_a)

# ......
```

异步内存复制

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，不可以直接拷贝运行，仅供参考，示例代码如下：

```
import acl
# ......

# 1.申请内存。
size = 1 * 1024 * 1024
# acl.rt.malloc 申请的Device侧内存系统保证已经符合64对齐。
dev_ptr_a, ret = acl.rt.malloc(size, ACL_MEM_MALLOC_NORMAL_ONLY)
# 异步内存复制要求，内存首地址64字节对齐，使用acl.rt.malloc_host 需多申请64字节。
host_ptr_b, ret = acl.rt.malloc_host(size + 64)
# host申请的内存需要用户自己64对齐处理。
host_align = host_ptr_b+ 64 - host_ptr_b % 64

# 2.申请内存后，可向内存中读入数据，该自定义函数read_file由用户实现。
read_file(fileName, dev_ptr_a, size)

# 3.异步内存复制。
# 异步内存复制，dev_ptr_a表示Device上源内存地址指针，host_align 表示Host上目的内存地址指针，size表示内存大小。
# ACL_MEMCPY_DEVICE_TO_HOST = 2。
stream = acl.rt_create_stream()
ret = acl.rt.memcpy_async(host_align, size , dev_ptr_a, size, ACL_MEMCPY_DEVICE_TO_HOST, stream)
ret = acl.rt.synchronize_stream(stream)

# 4.使用完内存中的数据后，需及时释放资源。
ret = acl.rt.destroy_stream(stream)
ret = acl.rt.free_host(host_ptr_b)
ret = acl.rt.free(dev_ptr_a)

# ......
```

4.6 模型推理基本场景

4.6.1 模型加载

按照4.2 模型构建中的说明构建出模型后，再加载该模型，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3.4 pyACL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 单算子调用。

pyACL提供两套模型加载的接口：

- 如图4-4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。
- 如图4-5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。

图 4-4 模型加载流程（通过不同接口区分加载方式）

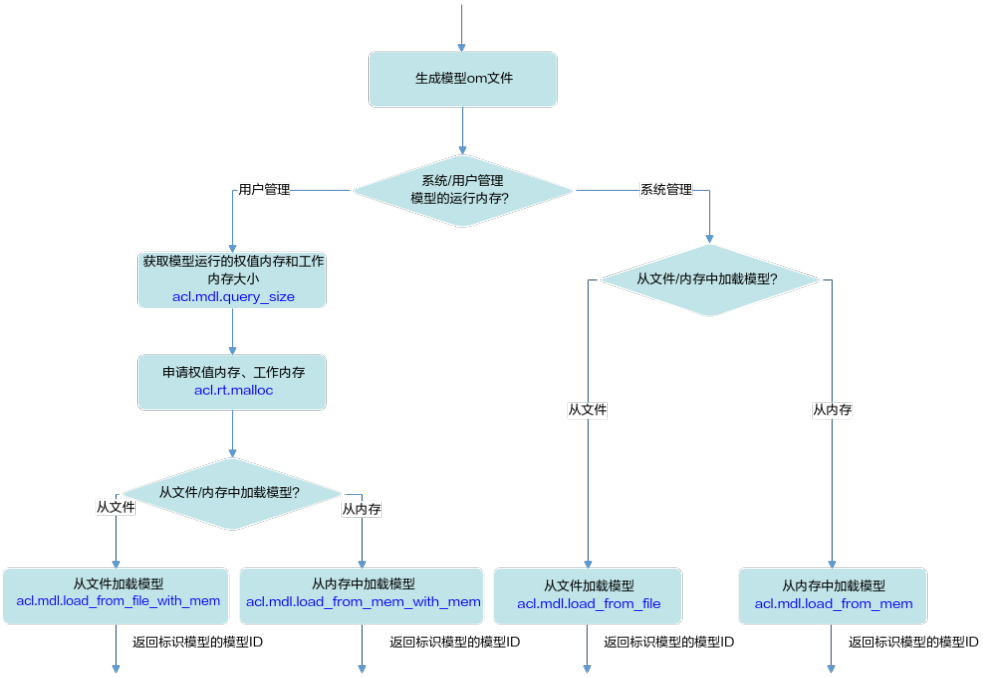
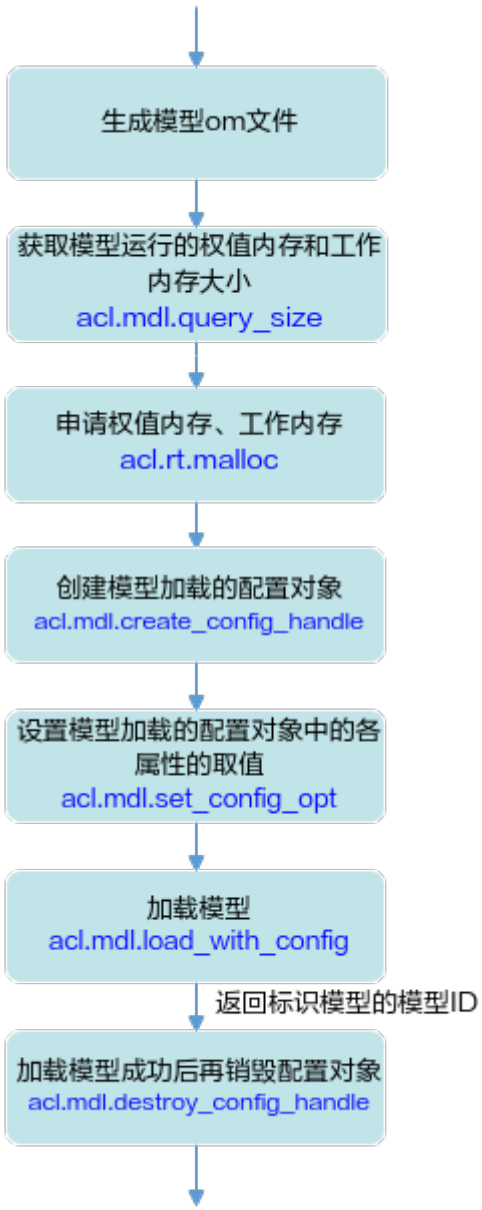


图 4-5 模型加载流程（通过接口中的配置参数区分加载方式）



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**acl.mdl.query_size**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**acl.mdl.query_size**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，**acl.mdl.load_from_file**、**acl.mdl.load_from_mem**）。
- 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：
  - 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。

  - **acl.mdl.load_from_file**：从文件加载离线模型数据，由系统内部管理内存。
  - **acl.mdl.load_from_mem**：从内存加载离线模型数据，由系统内部管理内存。
  - **acl.mdl.load_from_file_with_mem**：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）。
  - **acl.mdl.load_from_mem_with_mem**：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）。
- 使用**acl.mdl.set_config_opt**接口、**acl.mdl.load_with_config**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。

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**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，不可以直接拷贝运行，仅供参考。

```
# 初始化变量。
model_path = "./model/resnet50.om"
# ......

# 加载离线模型文件（适配昇腾AI处理器的离线模型），由系统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。
model_id, ret = acl.mdl.load_from_file(model_path)

# ......
```

4.6.2 模型执行

4.6.2.1 接口调用流程

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**3.4 pyACL接口调用流程**了解整体流程，再查看本节中的流程说明。

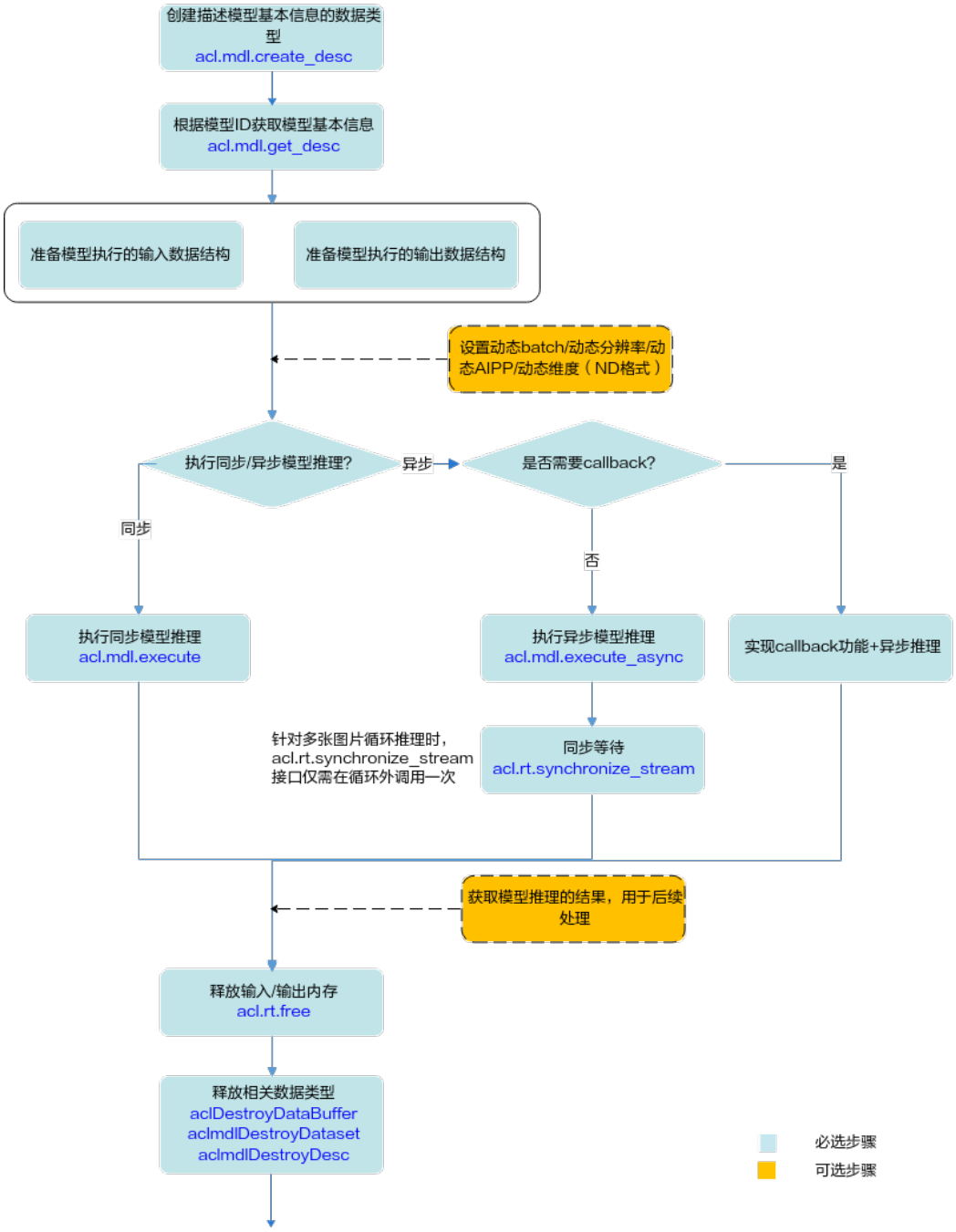
本节描述的是整网模型执行的接口调用流程，对于算子模型加载与执行的详细说明请参见**6 单算子调用**。

- **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

acl.destroy_data_buffer接口销毁相应的aclDataBuffer类型，并调用**acl.rt.free**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**acl.mdl.create_desc**接口创建描述模型基本信息的数据类型。
2. 调用**acl.mdl.get_desc**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**7.6 模型动态推理**、**7.7 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数后，才能将输入数据发送给模型进行推理。不满足Batch数时，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：

   – 同步推理时调用**acl.mdl.execute**接口。
   – 异步推理时调用**acl.mdl.execute_async**接口。

     对于异步接口，还需调用**acl.rt.synchronize_stream**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**7.5 模型异步推理**。

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**acl.rt.free**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，需及时依次调用**acl.destroy_data_buffer**接口和**acl.mdl.destroy_dataset**接口释放描述模型输入的数据。如果存在多个输入、输出，需调用多次**acl.destroy_data_buffer**接口。

准备模型执行的输入/输出数据结构

pyACL提供了以下数据类型来描述模型、模型输入、模型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**acl.mdl.get_desc**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。

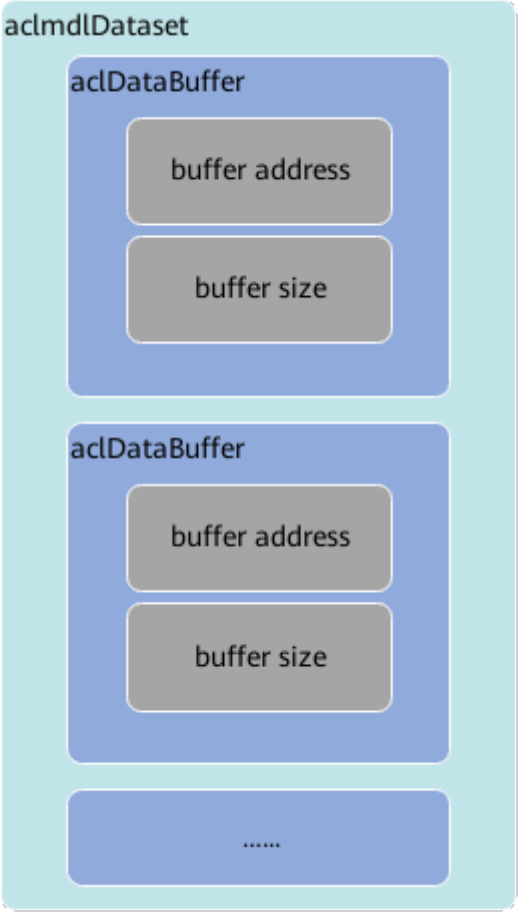
- 使用**aclmdlDataset**类型的数据描述模型的输入/输出数据，模型可能存在多个输入、多个输出。

  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。

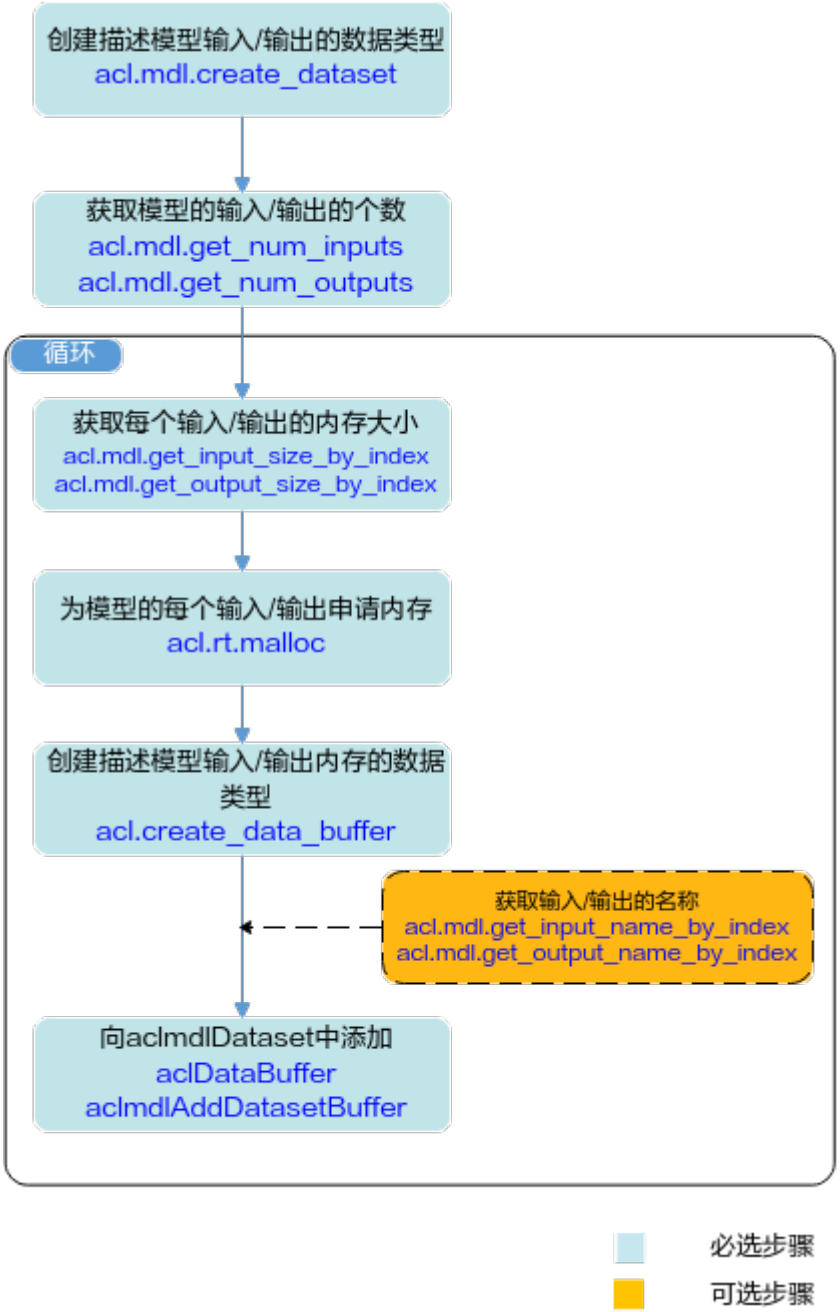
  调用aclDataBuffer类型下的操作接口获取内存地址、内存大小等。

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**acl.mdl.get_num_inputs**、**acl.mdl.get_num_outputs**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**acl.mdl.get_input_size_by_index**、**acl.mdl.get_output_size_by_index**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**acl.mdl.get_input_size_by_index**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**acl.mdl.get_input_name_by_index**、**acl.mdl.get_output_name_by_index**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。

4.6.2.2 示例代码

您可以从**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，不可以直接拷贝运行，仅供参考。

示例代码（执行模型）

```
ACL_MEMCPY_HOST_TO_DEVICE = 1
ACL_MEMCPY_DEVICE_TO_HOST = 2
NPY_BYTE = 1
images_list = ["./data/dog1_1024_683.jpg", "./data/dog2_1024_683.jpg"]

for image in images_list:
    # 1.自定义函数transfer_pic，使用Python库读取图片文件，并对图片进行缩放、剪裁等操作。
    # transfer_pic函数的实现请参考样例中源代码。
    img = transfer_pic(image)

    # 2.准备模型推理的输入数据，运行模式默认为运行模式为ACL_HOST，当前实例代码中模型只有一个输入。

    bytes_data = img.tobytes()
    np_ptr = acl.util.bytes_to_ptr(bytes_data)
    # 将图片数据从Host传输到Device。
    ret = acl.rt.memcpy(self.input_data[0]["buffer"], self.input_data[0]["size"], np_ptr,
                        self.input_data[0]["size"], ACL_MEMCPY_HOST_TO_DEVICE)

    # 3.执行模型推理。
    # self.model_id表示模型ID，在模型加载成功后，会返回标识模型的ID。
    ret = acl.mdl.execute(self.model_id, self.load_input_dataset, self.load_output_dataset)

# ......
```

示例代码（准备模型的输入和输出数据结构）

```
# 初始化变量。
ACL_MEM_MALLOC_HUGE_FIRST = 0

# 1.根据模型的ID，获取该模型的描述信息。
# self.model_desc为aclmdlDesc类型。
self.model_desc = acl.mdl.create_desc()
ret = acl.mdl.get_desc(self.model_desc, self.model_id)

# 2.准备模型推理的输入数据集。
# 创建aclmdlDataset类型的数据，描述模型推理的输入。
self.load_input_dataset = acl.mdl.create_dataset()
# 获取模型输入的数量。
input_size = acl.mdl.get_num_inputs(self.model_desc)
self.input_data = []
# 循环为每个输入申请内存，并将每个输入添加到aclmdlDataset类型的数据中。
for i in range(input_size):
    buffer_size = acl.mdl.get_input_size_by_index(self.model_desc, i)
    # 申请输入内存。
    buffer, ret = acl.rt.malloc(buffer_size, ACL_MEM_MALLOC_HUGE_FIRST)
    data = acl.create_data_buffer(buffer, buffer_size)
    _, ret = acl.mdl.add_dataset_buffer(self.load_input_dataset, data)
    self.input_data.append({"buffer": buffer, "size": buffer_size})

# 3.准备模型推理的输出数据集。
# 创建aclmdlDataset类型的数据，描述模型推理的输出。
self.load_output_dataset = acl.mdl.create_dataset()
```

```
# 获取模型输出的数量。
output_size = acl.mdl.get_num_outputs(self.model_desc)
self.output_data = []
# 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 i in range(output_size):
    buffer_size = acl.mdl.get_input_size_by_index(self.model_desc, i)
    # 申请输出内存。
    buffer, ret = acl.rt.malloc(buffer_size, ACL_MEM_MALLOC_HUGE_FIRST)
    data = acl.create_data_buffer(buffer, buffer_size)
    _, ret = acl.mdl.add_dataset_buffer(self.load_output_dataset, data)
    self.output_data.append({"buffer": buffer, "size": buffer_size})

# ......
```

示例代码（处理推理结果：直接处理内存中的数据）

以图片分类网络为例，模型执行结束后，需处理每一张图片的模型推理结果，直接输出top5置信度的类别编号。

```
# 处理模型推理的输出数据，输出top5置信度的类别编号。
inference_result = []
for i, item in enumerate(self.output_data):
    buffer_host, ret = acl.rt.malloc_host(self.output_data[i]["size"])
    # 将推理输出数据从Device传输到Host。
    ret = acl.rt.memcpy(buffer_host, self.output_data[i]["size"], self.output_data[i]["buffer"],
                        self.output_data[i]["size"], ACL_MEMCPY_DEVICE_TO_HOST)

    bytes_out = acl.util.ptr_to_bytes(buffer_host, self.output_data[i]["size"])
    data = np.frombuffer(bytes_out, dtype=np.byte)

    inference_result.append(data)
    tuple_st = struct.unpack("1000f", bytearray(inference_result[0]))
    vals = np.array(tuple_st).flatten()
    top_k = vals.argsort()[-1:-6:-1]
    print("======== top5 inference results: =============")
    for j in top_k:
        print("[%d]: %f" % (j, vals[j]))

# ......
```

示例代码（释放模型的输入、输出资源）

```
# 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。
while self.input_data:
    item = self.input_data.pop()
    ret = acl.rt.free(item["buffer"])
input_number = acl.mdl.get_dataset_num_buffers(self.load_input_dataset)
for i in range(input_number):
    data_buf = acl.mdl.get_dataset_buffer(self.load_input_dataset, i)
    if data_buf:
        ret = acl.destroy_data_buffer(data_buf)
ret = acl.mdl.destroy_dataset(self.load_input_dataset)

# 释放输出资源，包括数据结构和内存。
while self.output_data:
    item = self.output_data.pop()
    ret = acl.rt.free(item["buffer"])
output_number = acl.mdl.get_dataset_num_buffers(self.load_output_dataset)
for i in range(output_number):
    data_buf = acl.mdl.get_dataset_buffer(self.load_output_dataset, i)
    if data_buf:
        ret = acl.destroy_data_buffer(data_buf)
ret = acl.mdl.destroy_dataset(self.load_output_dataset)
```

4.6.3 模型卸载

关于模型卸载的接口调用流程，请参见**3.4 pyACL接口调用流程**。

基本原理

在模型推理结束后，还需要通过**acl.mdl.unload**接口卸载模型，并销毁aclmdlDesc类型的模型描述信息。

示例代码

```
# 卸载模型。
ret = acl.mdl.unload(self.model_id)

# 释放模型描述信息。
if self.model_desc:
    ret = acl.mdl.destroy_desc(self.model_desc)
    self.model_desc = None
```

5 图像/视频数据处理

媒体数据处理基础知识
V1与V2版本的差别
媒体数据处理V1
媒体数据处理V2

5.1 媒体数据处理基础知识

本章主要介绍图像/视频数据处理的具体功能、接口调用流程以及示例代码。

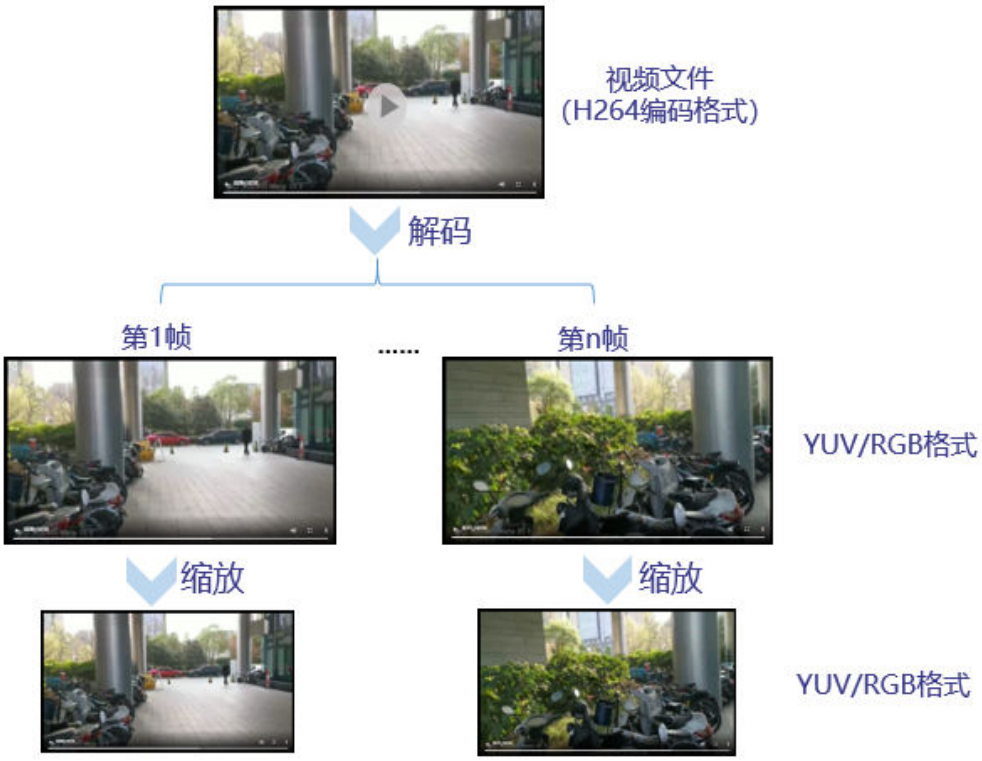
DVPP 图像/视频数据处理的典型使用场景

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。

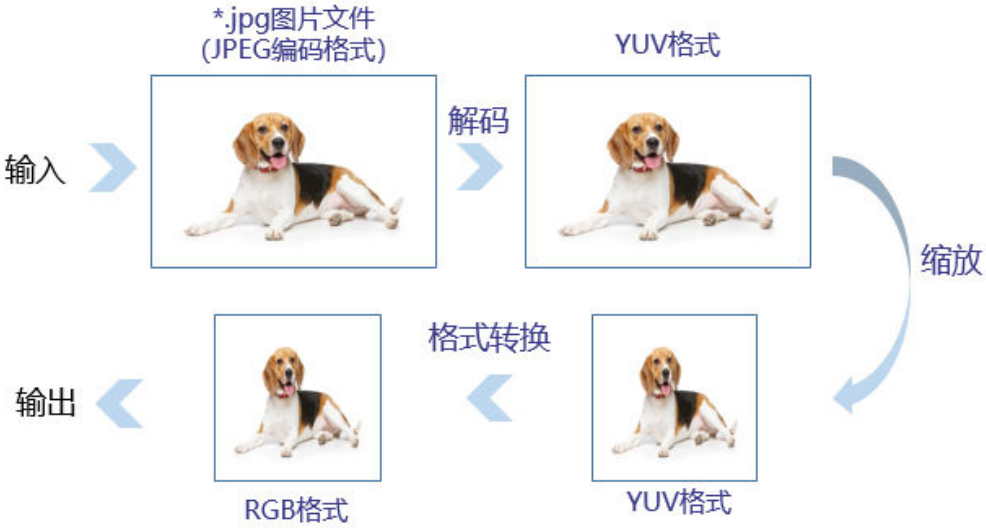
图 5-1 视频解码、缩放使用场景图



- 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 5-2 图片解码、缩放、格式转换使用场景图



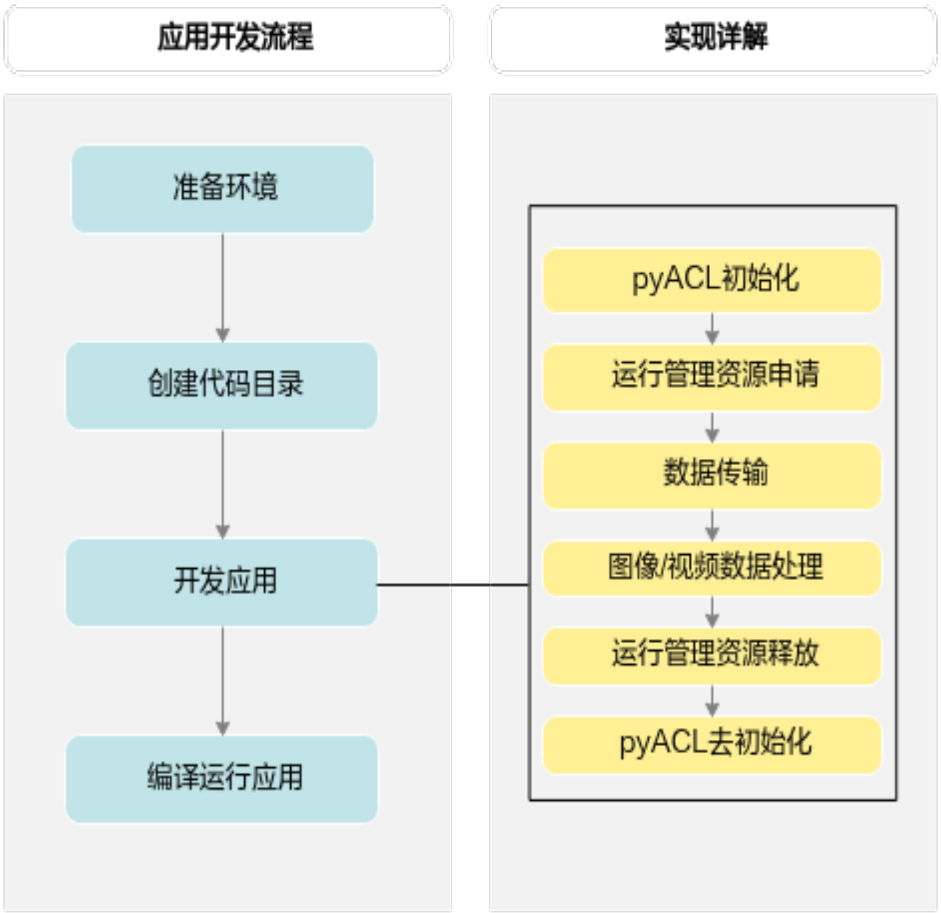
- 抠图、缩放、格式转换。

  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 5-3 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 5-4 开发流程



1. 准备环境。

请参见3.6 应用开发环境准备。

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、脚本、测试图片数据、模型文件等。

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。

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

5. 运行应用，请参见8 应用调试。

5.2 V1 与 V2 版本的差别

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，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。

昇腾AI处理器对V2版本各功能的支持度，请参见5.4.1 功能支持度说明。

5.3 媒体数据处理 V1

5.3.1 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下表所示。

昇腾AI处理器	VPC	JPEGD	JPEGE	PNGD	VDEC	VENC
Atlas 200/500 A2 推理产品	√	√	√	√	√	√

5.3.2 VPC 图像处理典型功能

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。关于VPC功能的详细介绍请参见10.7.6.1 功能说明，关于VPC功能对输入、输出的约束要求，请参见10.7.6.2 约束说明。

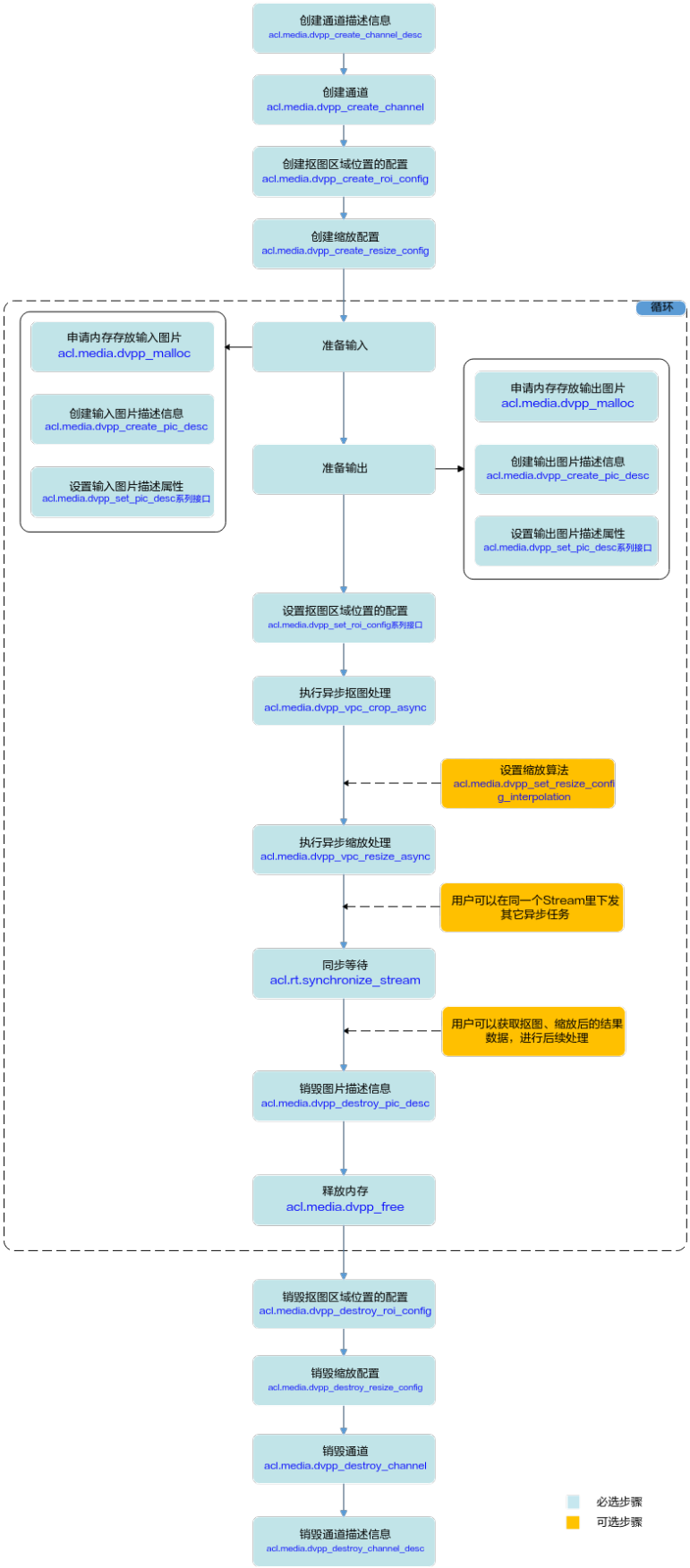
本节以抠图、缩放为例说明VPC图像处理时的接口调用流程，同时配合以下典型功能的示例代码辅助理解该接口调用流程：

- 图片缩放示例代码
- 格式转换示例代码
- 抠图（一图一框）示例代码
- 抠图缩放（一图一框）示例代码
- 抠图贴图（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**3.4 pyACL接口调用流程**了解整体流程，再查看本节中的流程说明。

图 5-5 抠图缩放流程


创建通道描述信息
acl.media.dvpp_create_channel_desc
创建通道
acl.media.dvpp_create_channel
创建抠图区域位置的配置
acl.media.dvpp_create_roi_config
创建缩放配置
acl.media.dvpp_create_resize_config
循环
申请内存存放输入图片
acl.media.dvpp_malloc
创建输入图片描述信息
acl.media.dvpp_create_pic_desc
设置输入图片描述属性
acl.media.dvpp_set_pic_desc系列接口
准备输入
准备输出
申请内存存放输出图片
acl.media.dvpp_malloc
创建输出图片描述信息
acl.media.dvpp_create_pic_desc
设置输出图片描述属性
acl.media.dvpp_set_pic_desc系列接口
设置抠图区域位置的配置
acl.media.dvpp_set_roi_config系列接口
执行异步抠图处理
acl.media.dvpp_vpc_crop_async
设置缩放算法
acl.media.dvpp_set_resize_config_interpolation
执行异步缩放处理
acl.media.dvpp_vpc_resize_async
用户可以在同一个Stream里下发其它异步任务
同步等待
acl.rt.synchronize_stream
用户可以获取抠图、缩放后的结果数据，进行后续处理
销毁图片描述信息
acl.media.dvpp_destroy_pic_desc
释放内存
acl.media.dvpp_free
销毁抠图区域位置的配置
acl.media.dvpp_destroy_roi_config
销毁缩放配置
acl.media.dvpp_destroy_resize_config
销毁通道
acl.media.dvpp_destroy_channel
销毁通道描述信息
acl.media.dvpp_destroy_channel_desc
必选步骤
可选步骤

关键接口的说明如下（以抠图、缩放处理为例）：

1. 调用**acl.media.dvpp_create_channel**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**acl.media.dvpp_create_channel_desc**接口创建通道描述信息。
2. 调用**acl.media.dvpp_create_roi_config**接口、**acl.media.dvpp_create_resize_config**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**acl.media.dvpp_malloc**申请内存。
4. 执行抠图、缩放。
   - 关于抠图：
     - 调用**acl.media.dvpp_vpc_crop_async**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“crop_area”不一致时会对图片再做一次缩放操作。
     - 当前系统还提供了**acl.media.dvpp_vpc_crop_and_paste_async**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“crop_area”的宽高与贴图区域“paste_area”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**acl.media.dvpp_vpc_resize_async**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**acl.rt.synchronize_stream**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**acl.media.dvpp_free**接口释放输入、输出内存。
6. 调用**acl.media.dvpp_destroy_roi_config**接口、**acl.media.dvpp_destroy_resize_config**接口分别销毁抠图区域位置的配置、缩放配置。
7. 调用**acl.media.dvpp_destroy_channel**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**acl.media.dvpp_destroy_channel_desc**接口销毁通道描述信息。

图片缩放示例代码

- 调用**acl.media.dvpp_create_channel**接口创建图片数据处理的通道、调用**acl.media.dvpp_destroy_channel**接口销毁图片数据处理的通道。
- 调用**acl.media.dvpp_vpc_resize_async**异步接口，将输入图片缩放到输出图片大小。对于异步接口，还需调用**acl.rt.synchronize_stream**接口阻塞程序运行，直到指定Stream中的所有任务都完成。

- 调用**acl.media.dvpp_create_resize_config**接口创建图片缩放配置数据。
  Atlas 200/500 A2推理产品，默认缩放算法为“业界通用的Bilinear算法”。
  如果不想使用默认缩放算法，也可以调用**acl.media.dvpp_set_resize_config_interpolation**接口指定缩放算法。

您可以从**获取样例**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，不可以直接拷贝运行，仅供参考。

```
import acl
# ......

# 1.ACL初始化。
ret = acl.init()

# 2.运行管理资源申请,包括Device、Context、Stream。
self.device_id = 0
ret = acl.rt.set_device(self.device_id)
self.context, ret = acl.rt.create_context(self.device_id)
self.stream, ret = acl.rt.create_stream()

# 3.创建图片缩放配置数据，不支持指定缩放算法，默认缩放算法为“最近邻插值”。
# self.resize_config是acldvppResizeConfig类型。
self.resize_config = acl.media.dvpp_create_resize_config()

# 4.创建图片数据处理通道时的通道描述信息，self.dvpp_channel_desc是acldvppChannelDesc类型。
self.dvpp_channel_desc = acl.media.dvpp_create_channel_desc()

# 5.创建图片数据处理的通道。
ret = acl.media.dvpp_create_channel(self.dvpp_channel_desc)

# 6.申请缩放输入内存，内存大小根据计算公式得出; w和h是输入图片的实际尺寸。
width_stride = ((w + 15) // 16) * 16
height_stride = ((h + 1) // 2) * 2
buffer_size = (width_stride * height_stride * 3) // 2
dev_in, ret = acl.media.dvpp_malloc(buffer_size )
self.dev_buffer['input_0'] = dev
# 7.申请缩放输出内存，内存大小根据计算公式得出; w和h是输入图片的缩放尺寸。
width_stride = ((w + 15) // 16) * 16
height_stride = ((h + 1) // 2) * 2
buffer_size = (width_stride * height_stride * 3) // 2
dev_out, ret = acl.media.dvpp_malloc(buffer_size )
self.dev_buffer['output_0'] = dev

# 8.创建缩放输入图片的描述信息，并设置各属性值。
# self.input_desc是acldvppPicDesc类型。
self.input_desc = acl.media.dvpp_create_pic_desc()
ret = acl.media.dvpp_set_pic_desc_data(self.input_desc, buffer)
ret = acl.media.dvpp_set_pic_desc_format(self.input_desc, YUV420)
ret = acl.media.dvpp_set_pic_desc_width(self.input_desc, width)
ret = acl.media.dvpp_set_pic_desc_height(self.input_desc, height)
ret = acl.media.dvpp_set_pic_desc_width_stride(self.input_desc, wstride)
ret = acl.media.dvpp_set_pic_desc_height_stride(self.input_desc, hstride)
ret = acl.media.dvpp_set_pic_desc_size(self.input_desc, size)

# 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。
# self.output_desc是acldvppPicDesc类型。
self.output_desc= acl.media.dvpp_create_pic_desc()
ret = acl.media.dvpp_set_pic_desc_data(self.output_desc, buffer)
ret = acl.media.dvpp_set_pic_desc_format(self.output_desc, YUV420)
ret = acl.media.dvpp_set_pic_desc_width(self.output_desc, width)
ret = acl.media.dvpp_set_pic_desc_height(self.output_desc, height)
ret = acl.media.dvpp_set_pic_desc_width_stride(self.output_desc, wstride)
ret = acl.media.dvpp_set_pic_desc_height_stride(self.output_desc, hstride)
ret = acl.media.dvpp_set_pic_desc_size(self.output_desc, size)
```

```
# 10.执行异步缩放，再调用acl.rt.synchronize_stream接口阻塞Host运行，直到指定Stream中的所有任务都完成。
ret = acl.media.dvpp_vpc_resize_async(self.dvpp_channel_desc, self.input_desc,
                                      self.output_desc, self.resize_config,
                                      self.stream)
ret = acl.rt.synchronize_stream(self.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。
ret = acl.media.dvpp_destroy_pic_desc(self.output_desc)
ret = acl.media.dvpp_destroy_pic_desc(self.input_desc)
ret = acl.media.dvpp_free(self.dev_buffer['input_0'] )
ret = acl.media.dvpp_free(self.dev_buffer['output_0'] )

# 12.释放运行管理资源。
ret = acl.rt.destroy_stream(self.stream)
ret = acl.rt.destroy_context(self.context)
ret = acl.rt.reset_device(self.device_id)
ret = acl.finalize()
# ......
```

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acl.media.dvpp_vpc_crop_async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acl.media.dvpp_vpc_convert_color_async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。

```
import acl
# ......

ACL_MEMCPY_HOST_TO_DEVICE = 1
ACL_MEMCPY_DEVICE_TO_HOST = 2
PIXEL_FORMAT_YUV_400 = 0
PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1

# 1.ACL初始化。
ret = acl.init()

# 2.运行管理资源申请,包括Device、Context、Stream。
device_id = 0
ret = acl.rt.set_device(device_id)
context, ret = acl.rt.create_context(device_id)
stream, ret = acl.rt.create_stream()

# 3.创建图片数据处理通道时的通道描述信息，dvpp_channel_desc是acldvppChannelDesc类型。
dvpp_channel_desc = acl.media.dvpp_create_channel_desc()

# 4.创建图片数据处理的通道。
ret = acl.media.dvpp_create_channel(dvpp_channel_desc)

# 5.申请输入内存。
input_width_stride = ((input_width + 15) // 16) * 16
input_height_stride = ((input_height + 1) // 2) * 2
# input_width_stride、input_height_stride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。
in_buffer_size = (input_width_stride * input_height_stride * 3) // 2
in_dev_buffer, ret = acl.media.dvpp_malloc(in_buffer_size)
np_yuv = np.fromfile(path, dtype=np.byte)
buffer_size = np_yuv.itemsize * np_yuv.size
bytes_data = np_yuv.tobytes()
```

```
np_yuv_ptr = acl.util.bytes_to_ptr(bytes_data)
ret = acl.rt.memcpy(in_dev_buffer, buffer_size, np_yuv_ptr,
                    buffer_size, ACL_MEMCPY_HOST_TO_DEVICE)

# 6.申请色域转换输出内存。
output_width_stride = ((output_width + 15) // 16) * 16
output_height_stride = ((output_height + 1) // 2) * 2
# output_width_stride、output_height_stride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。
out_buffer_size = (output_width_stride * output_height_stride * 3) // 2
out_dev_buffer, ret = acl.media.dvpp_malloc(out_buffer_size)

# 7.创建色域转换输入图片的描述信息，并设置各属性值。
input_desc = acl.media.dvpp_create_pic_desc()
ret = acl.media.dvpp_set_pic_desc_data(input_desc, in_dev_buffer)
ret = acl.media.dvpp_set_pic_desc_format(input_desc, PIXEL_FORMAT_YUV_SEMIPLANAR_420)
ret = acl.media.dvpp_set_pic_desc_width(input_desc, input_width)
ret = acl.media.dvpp_set_pic_desc_height(input_desc, input_height)
ret = acl.media.dvpp_set_pic_desc_width_stride(input_desc, input_width_stride)
ret = acl.media.dvpp_set_pic_desc_height_stride(input_desc, input_height_stride)
ret = acl.media.dvpp_set_pic_desc_size(input_desc, in_buffer_size)

# 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。
output_desc = acl.media.dvpp_create_pic_desc()
ret = acl.media.dvpp_set_pic_desc_data(output_desc, out_dev_buffer)
ret = acl.media.dvpp_set_pic_desc_format(output_desc, PIXEL_FORMAT_YUV_400)
ret = acl.media.dvpp_set_pic_desc_width(output_desc, output_width)
ret = acl.media.dvpp_set_pic_desc_height(output_desc, output_height)
ret = acl.media.dvpp_set_pic_desc_width_stride(output_desc, output_width_stride)
ret = acl.media.dvpp_set_pic_desc_height_stride(output_desc, output_height_stride)
ret = acl.media.dvpp_set_pic_desc_size(output_desc, out_buffer_size)

# 9.执行异步色域转换，再调用acl.rt.synchronize_stream接口阻塞程序运行，直到指定Stream中的所有任务都完
成。
ret = acl.media.dvpp_vpc_convert_color_async(dvpp_channel_desc, input_desc,
                                             output_desc, stream)
ret = acl.rt.synchronize_stream(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。
ret = acl.media.dvpp_destroy_pic_desc(input_desc)
ret = acl.media.dvpp_destroy_pic_desc(output_desc)

np_output = np.zeros(out_buffer_size, dtype=np.byte)
bytes_data = np_output.tobytes()
np_output_ptr = acl.util.bytes_to_ptr(bytes_data)
# 将Device的处理结果数据传输到Host。
ret = acl.rt.memcpy(np_output_ptr, out_buffer_size, out_dev_buffer,
                    out_buffer_size, ACL_MEMCPY_DEVICE_TO_HOST)
ret = acl.media.dvpp_free(in_dev_buffer)
ret = acl.media.dvpp_free(out_dev_buffer)

# 11.释放运行管理资源。
ret = acl.rt.destroy_stream(self.stream)
ret = acl.rt.destroy_context(self.context)
ret = acl.rt.reset_device(self.device_id)
ret = acl.finalize()
# ......
```

抠图（一图一框）示例代码

调用**acl.media.dvpp_vpc_crop_async**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。

您可以从**获取样例**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，不可以直接拷贝运行，仅供参考。

```
import acl
# ......

# 1.pyACL初始化。
ret = acl.init()

# 2.运行管理资源申请,包括Device、Context、Stream。
self.device_id = 0
ret = acl.rt.set_device(self.device_id)
self.context, ret = acl.rt.create_context(self.device_id)
self.stream, ret = acl.rt.create_stream()

# 3.指定抠图区域的位置、指定贴图区域的位置。
# 按左上角为原点做偏移。w，h为图片原始宽高。
self.crop_area = acl.media.dvpp_create_roi_config(w // 2, w - 1, h // 2, h - 1)

# 4.创建图片数据处理通道时的通道描述信息，dvpp_channel_desc 是acldvppChannelDesc类型。
self.dvpp_channel_desc = acl.media.dvpp_create_channel_desc()

# 5.创建图片数据处理的通道。
ret = acl.media.dvpp_create_channel(self.dvpp_channel_desc)

# 6.创建输入输出图片的描述信息，并设置各属性值。
self.input_desc = acl.media.dvpp_create_pic_desc()
assert self.input_desc is not None
# 6.1 自定义方法 set_picture_desc 设置输入图片描述。
# 根据计算公式计算内存大小存储图片数据，并设置到图片描述，同时设置其他属性。
self.set_picture_desc(self.input_desc, w, h, "input", 0)

self.output_desc = acl.media.dvpp_create_pic_desc()
assert self.output_desc is not None
# 6.2 自定义方法 set_picture_desc 设置输出图片描述。
out_buffer_size = self.set_picture_desc(self.output_desc, w // 2, h // 2, "output", 0)

# 7.执行异步抠图，再调用acl.rt.synchronize_stream接口阻塞Host运行，直到指定Stream中的所有任务都完成。
ret = acl.media.dvpp_vpc_crop_async(self.dvpp_channel_desc, self.input_desc,
                                    self.output_desc, self.crop_area, self.stream)
ret = acl.rt.synchronize_stream(self.stream)

# 8.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。
ret = acl.media.dvpp_destroy_pic_desc(self.input_desc)
ret = acl.media.dvpp_destroy_pic_desc(self.output_desc)
# dev_buffer是字典对象，存储device侧申请内存。
for key in self.dev_buffer.keys():
    if self.dev_buffer[key]:
        ret = acl.media.dvpp_free(self.dev_buffer[key])
if self.dvpp_channel_desc:
    ret = acl.media.dvpp_destroy_channel(self.dvpp_channel_desc)
    assert ret == 0
    ret = acl.media.dvpp_destroy_channel_desc(self.dvpp_channel_desc)
    assert ret == 0

# 9.释放运行管理资源。
ret = acl.rt.destroy_stream(self.stream)
ret = acl.rt.destroy_context(self.context)
ret = acl.rt.reset_device(self.device_id)

# 10.pyACL去初始化。
ret = acl.finalize()
# ......
```

抠图缩放（一图一框）示例代码

- 调用**acl.media.dvpp_vpc_crop_resize_async**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。对于异步接口，还需调用**acl.rt.synchronize_stream**接口阻塞程序运行，直到指定Stream中的所有任务都完成。

调用**acl.media.dvpp_vpc_crop_resize_async**接口实现抠图缩放时，支持指定缩放算法。

- 输出图片区域与抠图区域“crop_area”不一致时会对图片再做一次缩放操作。

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，不可以直接拷贝运行，仅供参考。

```
import acl
# ......

# 1.pyACL初始化。
ret = acl.init()

# 2.运行管理资源申请,包括Device、Context、Stream。
self.device_id = 0
ret = acl.rt.set_device(self.device_id)
self.context, ret = acl.rt.create_context(self.device_id)
self.stream, ret = acl.rt.create_stream()

# 3.指定抠图区域的位置、指定贴图区域的位置。
# 按左上角为原点做偏移。w，h为图片原始宽高。
self.crop_area = acl.media.dvpp_create_roi_config(w // 2, w - 1, h // 2, h - 1)

# 4.创建图片数据处理通道时的通道描述信息，dvpp_channel_desc 是acldvppChannelDesc类型。
self.dvpp_channel_desc = acl.media.dvpp_create_channel_desc()

# 5.创建图片数据处理的通道。
ret = acl.media.dvpp_create_channel(self.dvpp_channel_desc)

# 6.创建输入输出图片的描述信息，并设置各属性值。
self.input_desc = acl.media.dvpp_create_pic_desc()
assert self.input_desc is not None
# 6.1 自定义方法 set_picture_desc 设置输入图片描述。
# 根据计算公式计算内存大小存储图片数据，并设置到图片描述，同时设置其他属性。
self.set_picture_desc(self.input_desc, w, h, "input", 0)

self.output_desc = acl.media.dvpp_create_pic_desc()
assert self.output_desc is not None
# 6.2 自定义方法 set_picture_desc 设置输出图片描述。
resize_width = w // 4
resize_height = h // 4
out_buffer_size = self.set_picture_desc(self.output_desc, resize_width, resize_height, "output", 0)

# 7.执行异步抠图，再调用acl.rt.synchronize_stream接口阻塞Host运行，直到指定Stream中的所有任务都完成。
self.resize_config = acl.media.dvpp_create_resize_config()
ret = acl.media.dvpp_vpc_crop_resize_async(self.dvpp_channel_desc, self.input_desc,
                                           self.output_desc, self.crop_area, self.resize_config,
                                           self.stream)
ret = acl.rt.synchronize_stream(self.stream)

# 8.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。
ret = acl.media.dvpp_destroy_pic_desc(self.input_desc)
ret = acl.media.dvpp_destroy_pic_desc(self.output_desc)
# dev_buffer是字典对象，存储device侧申请内存。
for key in self.dev_buffer.keys():
    if self.dev_buffer[key]:
        ret = acl.media.dvpp_free(self.dev_buffer[key])
if self.dvpp_channel_desc:
    ret = acl.media.dvpp_destroy_channel(self.dvpp_channel_desc)
    assert ret == 0
    ret = acl.media.dvpp_destroy_channel_desc(self.dvpp_channel_desc)
    assert ret == 0

# 9.释放运行管理资源。
ret = acl.rt.destroy_stream(self.stream)
ret = acl.rt.destroy_context(self.context)
ret = acl.rt.reset_device(self.device_id)
```

```
# 10.pyACL去初始化。
ret = acl.finalize()
# ......
```

抠图贴图（一图一框）示例代码

- 调用acl.media.dvpp_vpc_crop_and_paste_async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。对于异步接口，还需调用acl.rt.synchronize_stream接口阻塞程序运行，直到指定Stream中的所有任务都完成。
- 抠图区域“crop_area”的宽高与贴图区域“paste_area”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。

您可以从获取样例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，不可以直接拷贝运行，仅供参考。

```
import acl
# ......

# 1.pyACL初始化。
ret = acl.init()

# 2.运行管理资源申请，包括Device、Context、Stream。
self.device_id = 0
ret = acl.rt.set_device(self.device_id)
self.context, ret = acl.rt.create_context(self.device_id)
self.stream, ret = acl.rt.create_stream()

# 3.指定抠图区域的位置、指定贴图区域的位置。
# 按左上角为原点做偏移。w，h为图片原始宽高。
self.crop_area = acl.media.dvpp_create_roi_config(w // 2, w - 1, h // 2, h - 1)
self.paste_area = acl.media.dvpp_create_roi_config(w // 2, w - 1, h // 2, h - 1)

# 4.创建图片数据处理通道时的通道描述信息，dvpp_channel_desc 是acldvppChannelDesc类型。
self.dvpp_channel_desc = acl.media.dvpp_create_channel_desc()

# 5.创建图片数据处理的通道。
ret = acl.media.dvpp_create_channel(self.dvpp_channel_desc)

# 6.创建输入输出图片的描述信息，并设置各属性值。
self.input_desc = acl.media.dvpp_create_pic_desc()
assert self.input_desc is not None
# 6.1 自定义方法 set_picture_desc 设置输入图片描述。
# 根据计算公式计算内存大小存储图片数据，并设置到图片描述，同时设置其他属性。
self.set_picture_desc(self.input_desc, w, h, "input", 0)

self.output_desc = acl.media.dvpp_create_pic_desc()
assert self.output_desc is not None
# 6.2 自定义方法 set_picture_desc 设置输出图片描述。
out_buffer_size = self.set_picture_desc(self.output_desc, w, h, "output", 0)

# 7.执行异步抠图粘贴，再调用acl.rt.synchronize_stream接口阻塞Host运行，直到指定Stream中的所有任务都完成。
ret = acl.media.dvpp_vpc_crop_and_paste_async(self.dvpp_channel_desc,
                                              self.input_desc, self.output_desc,
                                              self.crop_area, self.paste_area,
                                              self.stream)
ret = acl.rt.synchronize_stream(self.stream)

# 8.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。
ret = acl.media.dvpp_destroy_pic_desc(self.input_desc)
```

```
ret = acl.media.dvpp_destroy_pic_desc(self.output_desc)
# dev_buffer是字典对象，存储device侧申请内存。
for key in self.dev_buffer.keys():
    if self.dev_buffer[key]:
        ret = acl.media.dvpp_free(self.dev_buffer[key])
if self.dvpp_channel_desc:
    ret = acl.media.dvpp_destroy_channel(self.dvpp_channel_desc)
    assert ret == 0
    ret = acl.media.dvpp_destroy_channel_desc(self.dvpp_channel_desc)
    assert ret == 0

# 9.释放运行管理资源。
ret = acl.rt.destroy_stream(self.stream)
ret = acl.rt.destroy_context(self.context)
ret = acl.rt.reset_device(self.device_id)

# 10.pyACL去初始化。
ret = acl.finalize()
# ......
```

抠图贴图缩放（一图一框）示例代码

- 调用acl.media.dvpp_vpc_crop_resize_paste_async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。对于异步接口，还需调用acl.rt.synchronize_stream接口阻塞程序运行，直到指定Stream中的所有任务都完成。

  调用acl.media.dvpp_vpc_crop_resize_paste_async实现抠图缩放贴图时，支持指定缩放算法。
- 抠图区域“crop_area”的宽高与贴图区域“paste_area”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，不可以直接拷贝运行，仅供参考。

```
import acl
# ......

# 1.pyACL初始化。
ret = acl.init()

# 2.运行管理资源申请，包括Device、Context、Stream。
self.device_id = 0
ret = acl.rt.set_device(self.device_id)
self.context, ret = acl.rt.create_context(self.device_id)
self.stream, ret = acl.rt.create_stream()

# 3.创建图片数据处理通道时的通道描述信息，dvpp_channel_desc 是acldvppChannelDesc类型。
self.dvpp_channel_desc = acl.media.dvpp_create_channel_desc()

# 4.创建图片数据处理的通道。
ret = acl.media.dvpp_create_channel(self.dvpp_channel_desc)

# 5.创建输出图片的描述信息，并设置各属性值。
self.output_desc = acl.media.dvpp_create_pic_desc()
assert self.output_desc is not None
out_buffer_size = self.set_picture_desc(self.output_desc, w // 2, h // 2, "output", 0)

# 6.创建输入图片的描述信息，并设置各属性值。
self.input_desc = acl.media.dvpp_create_pic_desc()
assert self.input_desc is not None
self.set_picture_desc(self.input_desc, w, h, "input", 0)
```

```
# 7.加载图片数据，并拷贝到Device侧。
np_yuv = np.fromfile(path, dtype=np.byte)
in_buffer_size = np_yuv.itemsize * np_yuv.size
bytes_data = np_yuv.tobytes()
np_yuv_ptr = acl.util.bytes_to_ptr(bytes_data)
ret = acl.rt.memcpy(self.dev_buffer["input_0"], in_buffer_size, np_yuv_ptr,
                    in_buffer_size, ACL_MEMCPY_HOST_TO_DEVICE)
assert ret == 0

# 8.指定抠图、贴图区域的位置。
self.crop_area = acl.media.dvpp_create_roi_config(w // 2, w - 1, h // 2, h - 1)
self.paste_area = acl.media.dvpp_create_roi_config(w // 4, w // 2 - 1, h // 4, h // 2 - 1)

# 9.执行异步抠图贴图缩放。
self.resize_config = acl.media.dvpp_create_resize_config()
ret = acl.media.dvpp_vpc_crop_resize_paste_async(self.dvpp_channel_desc,
                                                 self.input_desc, self.output_desc,
                                                 self.crop_area, self.paste_area,
                                                 self.resize_config, self.stream)
assert ret == 0
ret = acl.rt.synchronize_stream(self.stream)
assert ret == 0

# 10.将输出数据拷贝到Host侧。
np_output = np.zeros(out_buffer_size, dtype=np.byte)
bytes_data = np_output.tobytes()
np_output_ptr = acl.util.bytes_to_ptr(bytes_data)
ret = acl.rt.memcpy(np_output_ptr, out_buffer_size, self.dev_buffer["output_0"],
                    out_buffer_size, ACL_MEMCPY_DEVICE_TO_HOST)
assert ret == 0

# 11.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。
ret = acl.media.dvpp_destroy_pic_desc(self.input_desc)
ret = acl.media.dvpp_destroy_pic_desc(self.output_desc)
# dev_buffer是字典对象，存储device侧申请内存。
for key in self.dev_buffer.keys():
    if self.dev_buffer[key]:
        ret = acl.media.dvpp_free(self.dev_buffer[key])
if self.dvpp_channel_desc:
    ret = acl.media.dvpp_destroy_channel(self.dvpp_channel_desc)
    assert ret == 0
    ret = acl.media.dvpp_destroy_channel_desc(self.dvpp_channel_desc)
    assert ret == 0

# 12.释放运行管理资源。
ret = acl.rt.destroy_stream(self.stream)
ret = acl.rt.destroy_context(self.context)
ret = acl.rt.reset_device(self.device_id)

# 13.pyACL去初始化。
ret = acl.finalize()
# ......
```

抠图贴图（一图多框）示例代码

- 调用**dvpp_vpc_batch_crop_and_paste_async**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。对于异步接口，还需调用**acl.rt.synchronize_stream**接口阻塞程序运行，直到指定Stream中的所有任务都完成。
- 抠图区域“crop_area”的宽高与贴图区域“paste_area”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，不可以直接拷贝运行，仅供参考。

```
import acl
# ......

# 1.pyACL初始化。
ret = acl.init()

# 2.运行管理资源申请,包括Device、Context、Stream。
self.device_id = 0
ret = acl.rt.set_device(self.device_id)
self.context, ret = acl.rt.create_context(self.device_id)
self.stream, ret = acl.rt.create_stream()

# 3.创建图片批处理输入输出描述。
self.out_batch_pic_desc = acl.media.dvpp_create_batch_pic_desc(self.out_batch_size)
self.in_batch_pic_desc = acl.media.dvpp_create_batch_pic_desc(self.in_batch_size)
# 3.1 指定批量抠图区域的位置、指定批量贴图区域的位置。
# 3.2 设置输入输出图片描述: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。
for i in range(self.in_batch_size):
    input_desc = acl.media.dvpp_get_pic_desc(self.in_batch_pic_desc, i)
    assert input_desc is not None
    # 申请内存并设置输出图片描述。
    self.set_picture_desc(input_desc, w, h, "input", i)
    # copy from host to device.
    key = "input" + '_' + str(i)
    ret = acl.rt.memcpy(self.dev_buffer[key], in_buffer_size, np_yuv_ptr,
                        in_buffer_size, 1)
    assert ret == 0
    # 创建roiNums,每张图对应需要抠图和贴图的数量。
    roiList.append(self.out_batch_size // self.in_batch_size)
for i in range(self.out_batch_size):
    output_desc = acl.media.dvpp_get_pic_desc(self.out_batch_pic_desc, i)
    assert output_desc is not None
    # 自定义方法申请内存并设置输出图片描述。
    self.set_picture_desc(output_desc, w, h, "output", i)
    if i == 0:
        crop_area = acl.media.dvpp_create_roi_config(w // 2, w - 1, h // 2, h - 1)
    else:
        crop_area = acl.media.dvpp_create_roi_config(0, w - 1, h // 2, h - 1)
    paste_area = acl.media.dvpp_create_roi_config(w // 2, w - 1, h // 2, h - 1)
    self.cropList.append(crop_area)
    self.pasteList.append(paste_area)
# 3.3 roiList,每张图对应需要抠图和贴图的数量。输出图片数相对输入多出来的，加到最后一张输入图片的输出。
total_num = 0
for i in range(self.in_batch_size):
    total_num += roiList[i]
if self.out_batch_size % self.in_batch_size != 0:
    roiList[-1] = self.out_batch_size - total_num + roiList[-1]

# 4.创建图片数据处理通道时的通道描述信息，dvppChannelDesc_是acldvppChannelDesc类型。
self.dvpp_channel_desc = acl.media.dvpp_create_channel_desc()

# 5.创建图片数据处理的通道。
ret = acl.media.dvpp_create_channel(self.dvpp_channel_desc)

# 6.执行异步缩放，再调用acl.rt.synchronize_stream接口阻塞程序运行，直到指定Stream中的所有任务都完成。
_, ret = acl.media.dvpp_vpc_batch_crop_and_paste_async(self.dvpp_channel_desc, self.in_batch_pic_desc,
                                                       roiList, self.out_batch_pic_desc, self.cropList,
                                                       self.pasteList, self.stream)
ret = acl.rt.synchronize_stream(self.stream)

# 7.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。
# 7.1 释放图片描述和批量图片描述。
self._free_pic_desc()
for i in range(len(self.cropList)):
    ret = acl.media.dvpp_destroy_roi_config(self.cropList[i])
    assert ret == 0
for i in range(len(self.pasteList)):
```

```
    ret = acl.media.dvpp_destroy_roi_config(self.pasteList[i])
    assert ret == 0
for key in self.dev_buffer.keys():
    if self.dev_buffer[key]:
        ret = acl.media.dvpp_free(self.dev_buffer[key])
# 7.2 销毁通道和通道描述。
if self.dvpp_channel_desc:
    ret = acl.media.dvpp_destroy_channel(self.dvpp_channel_desc)
    assert ret == 0
    ret = acl.media.dvpp_destroy_channel_desc(self.dvpp_channel_desc)
    assert ret == 0

# 8.释放运行管理资源。
ret = acl.rt.destroy_stream(self.stream)
ret = acl.rt.destroy_context(self.context)
ret = acl.rt.reset_device(self.device_id)

9.pyACL去初始化。
ret = acl.finalize()
# ......
```

5.3.3 JPEGD 图片解码

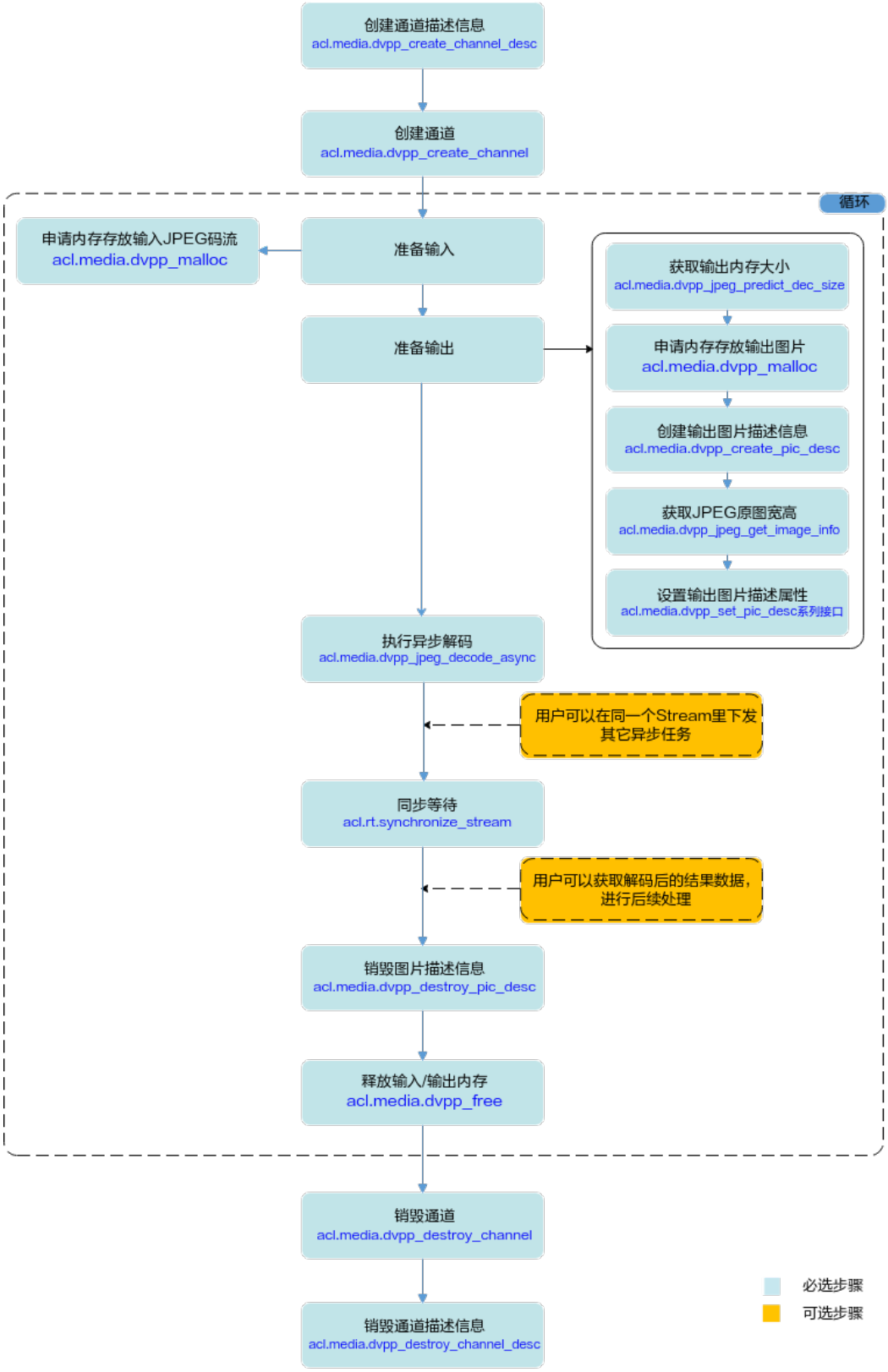
JPEGD（JPEG Decoder）负责完成图像解码功能，将.jpg、.jpeg、.JPG、.JPEG图片解码成YUV格式图片。关于JPEGD功能的详细介绍及约束请参见**10.7.7.1 功能及约束说明**。

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。

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**3.4 pyACL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 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：

版权所有 © 华为技术有限公司

1. 调用**acl.media.dvpp_create_channel**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**acl.media.dvpp_create_channel_desc**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**acl.media.dvpp_malloc**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**acl.media.dvpp_jpeg_predict_dec_size**接口预估JPEG图片解码后所需的输出内存的大小。

3. 调用**acl.media.dvpp_jpeg_decode_async**异步接口进行解码。

   对于异步接口，还需调用**acl.rt.synchronize_stream**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**acl.media.dvpp_free**接口释放输入、输出内存。

5. 调用**acl.media.dvpp_destroy_channel**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**acl.media.dvpp_destroy_channel_desc**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，不可以直接拷贝运行，仅供参考。

```
import acl
# ......

# 1.pyACL初始化。
ret = acl.init()

# 2.运行管理资源申请,包括Device、Context、Stream。
self.device_id = 0
ret = acl.rt.set_device(self.device_id)
context, ret = acl.rt.create_context(self.device_id)
stream, ret = acl.rt.create_stream()

# 3.创建图片数据处理通道时的通道描述信息，self.dvpp_channel_desc是acldvppChannelDesc类型。
self.dvpp_channel_desc = acl.media.dvpp_create_channel_desc()

# 4.创建图片数据处理的通道。
ret = acl.media.dvpp_create_channel(self.dvpp_channel_desc)

# 5.申请内存。
# 加载 YUV格式的图片数据。
# 申请Device内存inputDevBuff。
# 将通过acl.rt.memcpy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。
np_jpg = np.fromfile(self.path, dtype=np.byte)
self.in_buffer_size = np_jpg.itemsize * np_jpg.size
bytes_data = np_jpg.tobytes()
np_jpg_ptr = acl.util.bytes_to_ptr(bytes_data)
self.in_buffer_dev, ret = acl.media.dvpp_malloc(self.in_buffer_size)
ret = acl.rt.memcpy(self.in_buffer_dev, self.in_buffer_size, np_jpg_ptr,
                    self.in_buffer_size, 1)

# 6 输出内存，按16对齐方式计算并申请Device内存self.out_buffer_dev,存放编码后的输出数据。
height_stride = ((h + 1) // 2) * 2
width_stride = ((w + 15) // 16) * 16
# malloc for output
self.out_buffer_size = (width_stride * height_stride * 3) // 2
self.out_buffer_dev, ret = acl.media.dvpp_malloc(self.out_buffer_size)

# 7. 创建解码输出图片的描述信息，设置各属性值。
# 解码设置输出内存和size到图片描述，用于数据占位。
```

```
self.pic_desc = acl.media.dvpp_create_pic_desc()
ret = acl.media.dvpp_set_pic_desc_data(self.pic_desc, self.out_buffer_dev)
ret = acl.media.dvpp_set_pic_desc_size(self.pic_desc, self.out_buffer_size)
ret = acl.media.dvpp_set_pic_desc_format(
        self.pic_desc, PIXEL_FORMAT_YUV_SEMIPLANAR_420)
ret = acl.media.dvpp_set_pic_desc_width(self.pic_desc, w)
ret = acl.media.dvpp_set_pic_desc_height(self.pic_desc, h)
ret = acl.media.dvpp_set_pic_desc_width_stride(self.pic_desc, width_stride)
ret = acl.media.dvpp_set_pic_desc_height_stride(self.pic_desc, height_stride)

# 8. 执行异步解码，再调用acl.rt.synchronize_stream接口阻塞Host运行，直到指定Stream中的所有任务都完成。
ret = acl.media.dvpp_jpeg_decode_async(self.dvpp_channel_desc,
                                       self.in_buffer_dev,
                                       self.in_buffer_size,
                                       self.pic_desc, self.stream)
ret = acl.rt.synchronize_stream(self.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。
ret = acl.media.dvpp_destroy_pic_desc(self.pic_desc)
ret = acl.media.dvpp_free(self.in_buffer_dev)
ret = acl.media.dvpp_free(self.out_buffer_dev)
ret = acl.media.dvpp_destroy_channel(self.dvpp_channel_desc)
ret = acl.media.dvpp_destroy_channel_desc(self.dvpp_channel_desc)

# 10. 释放运行管理资源。
ret = acl.rt.destroy_stream(self.stream)
ret = acl.rt.destroy_context(self.context)
ret = acl.rt.reset_device(self.device_id)
ret = acl.finalize()

# 11. pyACL去初始化。

# ......
```

5.3.4 JPEGE 图片编码

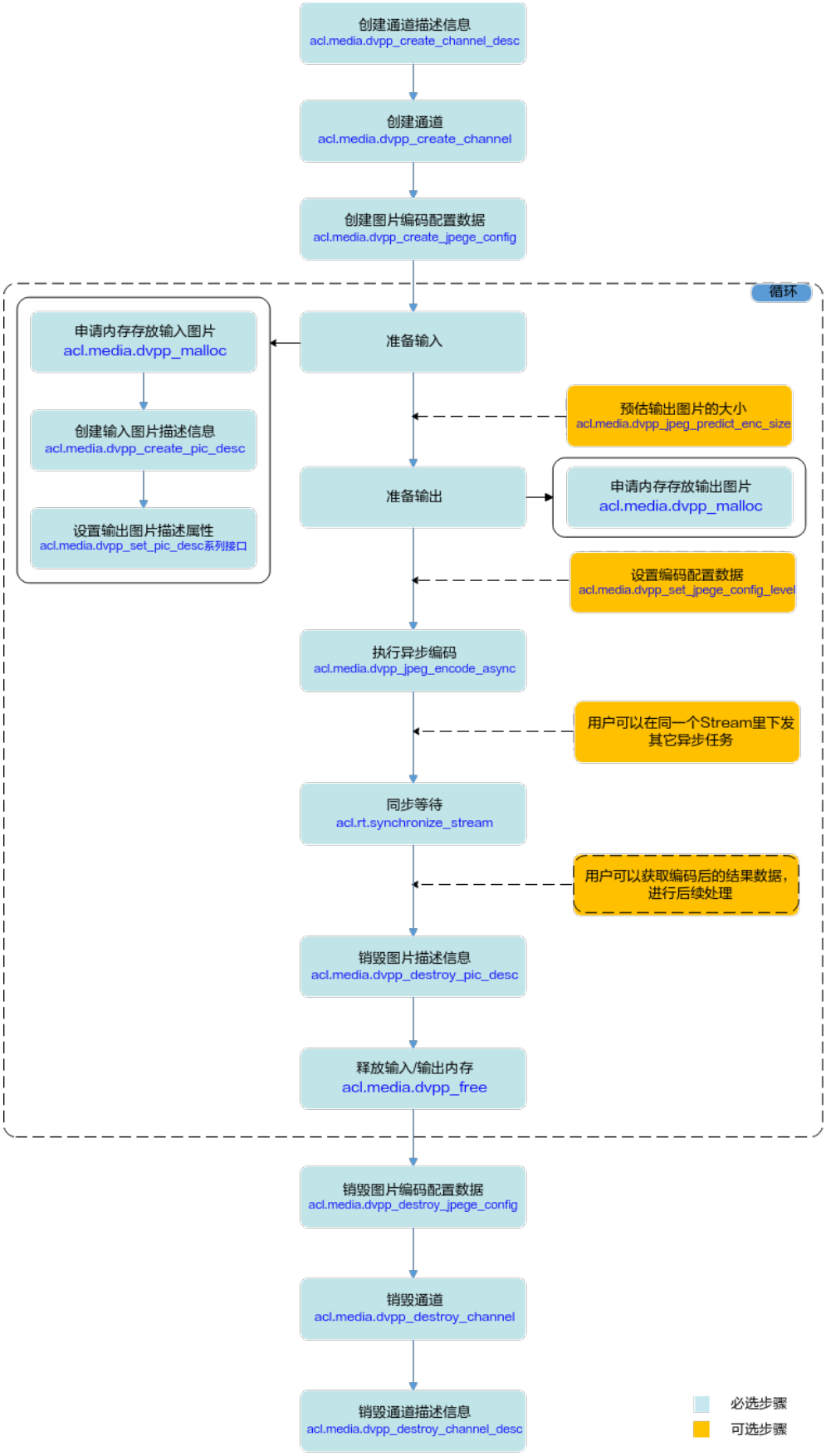
JPEGE（JPEG Encoder）负责完成图像编码功能，将YUV格式图片编码成.jpg图片。关于JPEGE功能的详细介绍及约束请参见**10.7.8.1 功能及约束说明**。

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。

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**3.4 pyACL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 图片编码



版权所有 © 华为技术有限公司

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**acl.media.dvpp_create_channel_desc**接口创建通道描述信息。
2. 调用**acl.media.dvpp_create_channel**接口创建图片数据处理的通道。
3. 调用**acl.media.dvpp_create_jpege_config**接口创建图片编码配置数据。
4. 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**acl.media.dvpp_malloc**申请内存。

   在申请输出内存前，可调用**acl.media.dvpp_jpeg_predict_enc_size**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。
5. 调用**acl.media.dvpp_jpeg_encode_async**异步接口进行编码。

   对于异步接口，还需调用**acl.rt.synchronize_stream**接口阻塞程序运行，直到指定Stream中的所有任务都完成。
6. 调用**acl.media.dvpp_destroy_jpege_config**接口销毁图片编码配置数据。
7. 在编码结束后，需及时调用**acl.media.dvpp_free**接口释放输入、输出内存。
8. 调用**acl.media.dvpp_destroy_channel**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**acl.media.dvpp_destroy_channel_desc**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，不可以直接拷贝运行，仅供参考。

```
import acl
# ......

# 1.pyACL初始化。
ret = acl.init()

# 2.运行管理资源申请,包括Device、Context、Stream。
self.device_id = 0
ret = acl.rt.set_device(self.device_id)
self.context, ret = acl.rt.create_context(self.device_id)
self.stream, ret = acl.rt.create_stream()

# 3.创建图片数据处理通道时的通道描述信息，self.dvpp_channel_desc是acldvppChannelDesc类型。
self.dvpp_channel_desc = acl.media.dvpp_create_channel_desc()

# 4.创建图片数据处理的通道。
ret = acl.media.dvpp_create_channel(self.dvpp_channel_desc)

# 5.申请内存。
# 5.1 输入内存。
#  加载 YUV格式的图片数据,将通过acl.rt.memcpy接口将Host的图片数据传输到Device。
np_jpg = np.fromfile(self.path, dtype=np.byte)
self.in_buffer_size = np_jpg.itemsize * np_jpg.size
bytes_data = np_jpg.tobytes()
np_jpg_ptr = acl.util.bytes_to_ptr(bytes_data)
self.in_buffer_dev, ret = acl.media.dvpp_malloc(self.in_buffer_size)
ret = acl.rt.memcpy(self.in_buffer_dev, self.in_buffer_size, np_jpg_ptr,
                    self.in_buffer_size, 1)

# 5.2 输出内存，按16对齐方式计算并申请Device内存self.out_buffer_dev,存放编码后的输出数据。
height_stride = ((h + 1) // 2) * 2
width_stride = ((w + 15) // 16) * 16
# malloc for output
self.out_buffer_size = (width_stride * height_stride * 3) // 2
self.out_buffer_dev, ret = acl.media.dvpp_malloc(self.out_buffer_size

# 6. 创建编码输入图片的描述信息，并设置各属性值。
```

```
# jpeg 编码设置输入yuv图片数据及size到图片描述。
self.pic_desc = acl.media.dvpp_create_pic_desc()
ret = acl.media.dvpp_set_pic_desc_data(self.pic_desc, self.in_buffer_dev)
ret = acl.media.dvpp_set_pic_desc_size(self.pic_desc, self.in_buffer_size)
ret = acl.media.dvpp_set_pic_desc_format(self.pic_desc, PIXEL_FORMAT_YUV_SEMIPLANAR_420)
ret = acl.media.dvpp_set_pic_desc_width(self.pic_desc, w)
ret = acl.media.dvpp_set_pic_desc_height(self.pic_desc, h)
ret = acl.media.dvpp_set_pic_desc_width_stride(self.pic_desc, width_stride)
ret = acl.media.dvpp_set_pic_desc_height_stride(self.pic_desc, height_stride)

# 7. 创建图片编码配置数据，设置编码质量, 预测输出的图片尺寸并分配内存。
# 编码质量范围[0, 100]，其中level 0编码质量与level 100相近，而在[1, 100]内数值越小输出图片质量越差。
self.jpege_config = acl.media.dvpp_create_jpege_config()
ret = acl.media.dvpp_set_jpege_config_level(self.jpege_config, 100)
self.out_buffer_size, ret = acl.media.dvpp_jpeg_predict_enc_size(
        self.pic_desc, self.jpege_config)
self.out_buffer_dev, ret = acl.media.dvpp_malloc(self.out_buffer_size)

# 8. 执行异步编码，再调用acl.rt.synchronize_stream接口阻塞Host运行，直到指定Stream中的所有任务都完成。
acl.rt.set_context(self.context)
np_out_size = np.array([self.out_buffer_size], dtype=np.int32)
bytes_data = np_out_size.tobytes()
np_out_size_ptr = acl.util.bytes_to_ptr(bytes_data)
ret = acl.media.dvpp_jpeg_encode_async(self.dvpp_channel_desc, self.pic_desc,
                                       self.out_buffer_dev, np_out_size_ptr,
                                       self.jpege_config, self.stream)
ret = acl.rt.synchronize_stream(self.stream)

# 9.申请Host内存hostPtr。
# 9.1 将编码后的输出图片回传到Host，再将Host内存中的数据写入文件,写完文件后，需及时调用acl.rt.free_host接口释放Host内存。
# 9.2 创建numpy对象并转成host侧指针存储输出内容。
np_output = np.zeros(size, dtype=np.byte)
bytes_data = np_output.tobytes()
np_output_ptr = acl.util.bytes_to_ptr(bytes_data)
size = int(np_out_size[0])
ret = acl.rt.memcpy(np_output_ptr, size, self.out_buffer_dev, size, 2)

# 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。
ret = acl.media.dvpp_destroy_jpege_config(self.jpege_config)
ret = acl.media.dvpp_destroy_pic_desc(self.pic_desc)
ret = acl.media.dvpp_free(self.in_buffer_dev)
ret = acl.media.dvpp_free(self.out_buffer_dev)
ret = acl.media.dvpp_destroy_channel(self.dvpp_channel_desc)
ret = acl.media.dvpp_destroy_channel_desc(self.dvpp_channel_desc)

# 11. 释放运行管理资源。
ret = acl.rt.destroy_stream(self.stream)
ret = acl.rt.destroy_context(self.context)
ret = acl.rt.reset_device(self.device_id);
ret = acl.finalize()

# 12.pyACL去初始化。

# ....
```

5.3.5 PNGD 图片解码

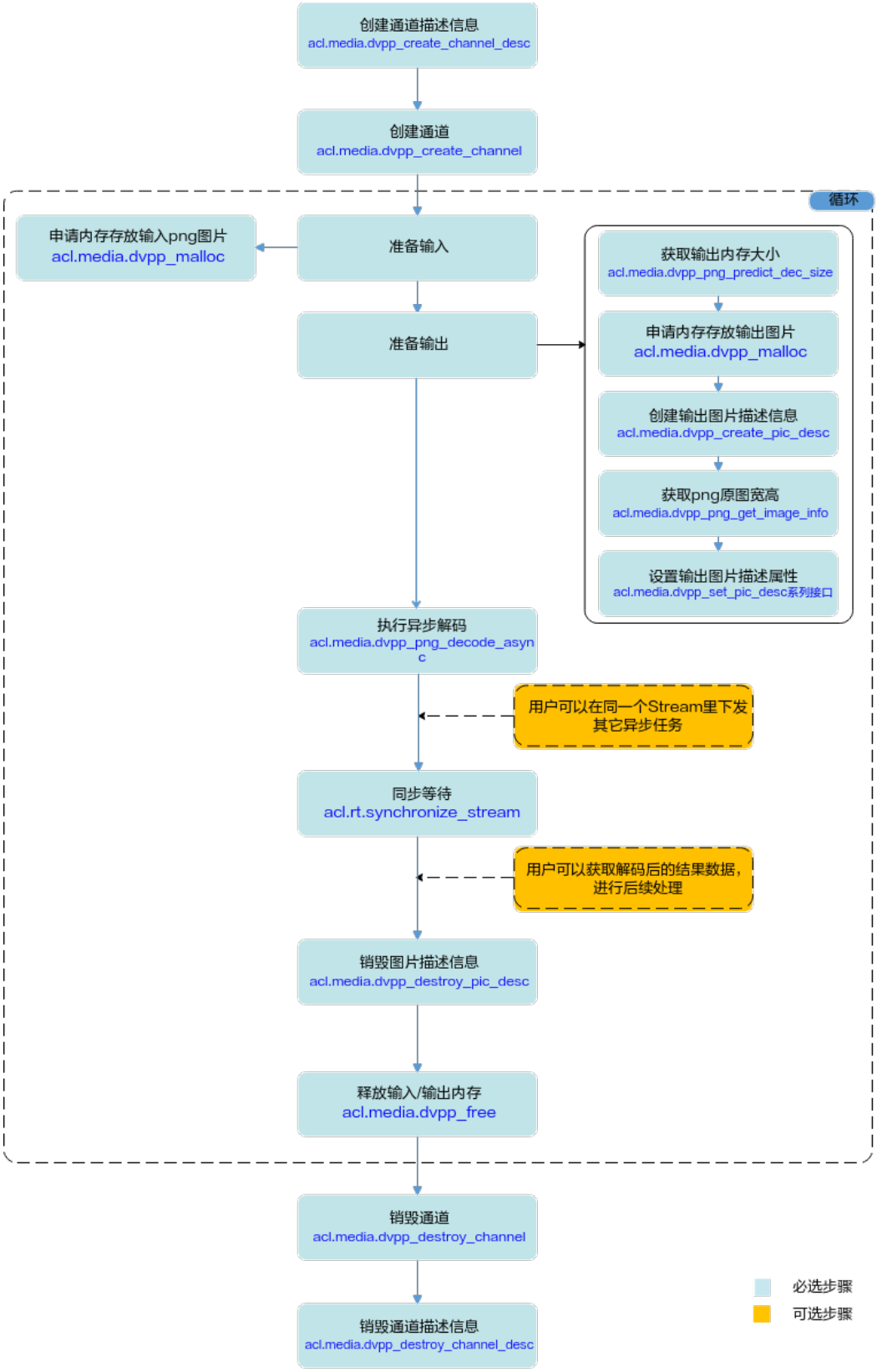
PNGD（PNG decoder）负责PNG格式图片的解码。关于PNGD功能的详细介绍及约束请参见**10.7.11.1 功能及约束说明**。

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。

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**3.4 pyACL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 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**acl.media.dvpp_create_channel_desc**接口创建通道描述信息。
2. 调用**acl.media.dvpp_create_channel**接口创建图片数据处理的通道。
3. 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**acl.media.dvpp_malloc**申请内存。

   在申请输出内存前，可调用**acl.media.dvpp_png_predict_dec_size**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。
4. 调用**acl.media.dvpp_png_decode_async**异步接口进行解码。

   对于异步接口，还需调用**acl.rt.synchronize_stream**接口阻塞程序运行，直到指定Stream中的所有任务都完成。
5. 在解码结束后，需及时调用**acl.media.dvpp_free**接口释放输入、输出内存。
6. 调用**acl.media.dvpp_destroy_channel**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**acl.media.dvpp_destroy_channel_desc**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，同时通过“ERROR_LOG”记录报错日志、通过“INFO_LOG”记录各动作的提示日志，示例代码中不一一列举。以下是关键步骤的代码示例，不可以直接拷贝运行，仅供参考。

```
import acl
# ......

# 1.pyACL初始化。
ACL_CONFIG_PATH = "../src/acl.json"
ret = acl.init(ACL_CONFIG_PATH)

# 2.运行管理资源申请,包括Device、Context、Stream。
device_id = 0
ret = acl.rt.set_device(device_id)
context, ret = acl.rt.create_context(device_id)
stream, ret = acl.rt.create_stream()

# 3.创建图片数据处理通道时的通道描述信息，dvpp_channel_desc是acldvppChannelDesc类型对象。
dvpp_channel_desc = acl.media.dvpp_create_channel_desc()

# 4.创建图片数据处理的通道。
ret = acl.media.dvpp_create_channel(dvpp_channel_desc)

# 5.申请输入内存。
pic_name = "../data/test_png_dec.png"
np_png = np.fromfile(pic_name, dtype=np.byte)
in_buffer_size = np_png.itemsize * np_png.size
bytes_data = np_png.tobytes()
np_png_ptr = acl.util.bytes_to_ptr(bytes_data)
in_dev_buffer, ret = acl.media.dvpp_malloc(in_buffer_size)
ACL_MEMCPY_HOST_TO_DEVICE = 1
ret = acl.rt.memcpy(in_dev_buffer, in_buffer_size, np_png_ptr, in_buffer_size,
    ACL_MEMCPY_HOST_TO_DEVICE)

# 6.申请解码输出内存。
PIXEL_FORMAT_RGB_888 = 12
out_buffer_size, ret = acl.media.dvpp_png_predict_dec_size(np_png_ptr, in_buffer_size,
    PIXEL_FORMAT_RGB_888)
out_dev_buffer, ret = acl.media.dvpp_malloc(out_buffer_size)

# 7.创建解码输出图片的描述信息，设置各属性值。
output_desc = acl.media.dvpp_create_pic_desc()
width, height, components, ret = acl.media.dvpp_png_get_image_info(np_png_ptr, in_buffer_size)
ret = acl.media.dvpp_set_pic_desc_data(output_desc, out_dev_buffer)
ret = acl.media.dvpp_set_pic_desc_size(output_desc, out_buffer_size)
```

```
ret = acl.media.dvpp_set_pic_desc_format(output_desc, PIXEL_FORMAT_RGB_888)

# 8.执行异步解码，再调用acl.rt.synchronize_stream接口阻塞程序运行，直到指定Stream中的所有任务都完成。
ret = acl.media.dvpp_png_decode_async(dvpp_channel_desc, in_dev_buffer, in_buffer_size,
    output_desc, stream)
ret = acl.rt.synchronize_stream(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。
ret = acl.media.dvpp_destroy_pic_desc(output_desc)
ret = acl.media.dvpp_free(in_dev_buffer)
ret = acl.media.dvpp_free(out_dev_buffer)
ret = acl.media.dvpp_destroy_channel(dvpp_channel_desc)
ret = acl.media.dvpp_destroy_channel_desc(dvpp_channel_desc)

# 10.释放运行管理资源。
ret = acl.rt.destroy_stream(stream)
ret = acl.rt.destroy_context(context)
ret = acl.rt.reset_device(device_id)
ret = acl.finalize()

# 11.pyACL去初始化。

# ....
```

5.3.6 VDEC 视频解码

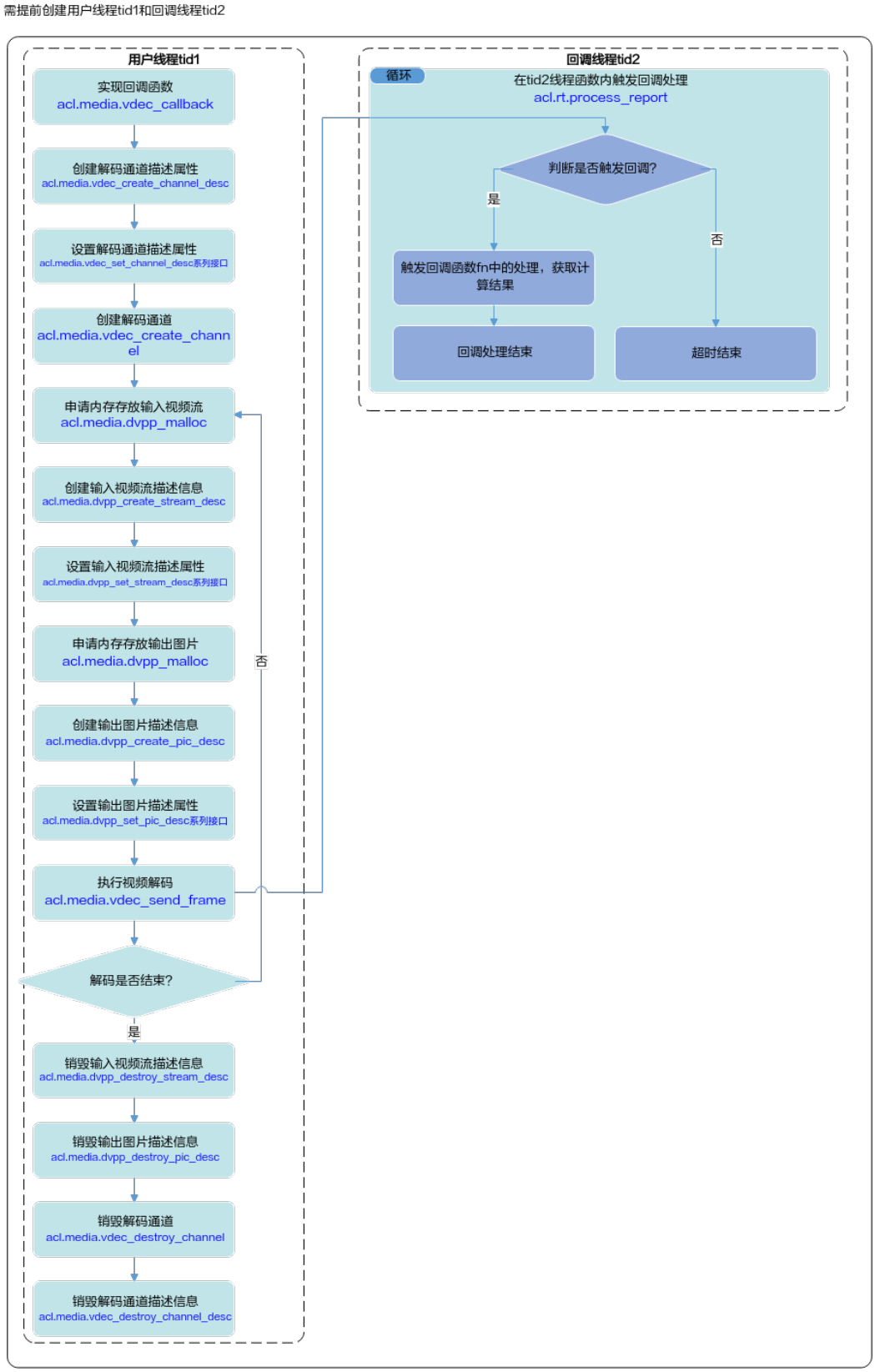
VDEC（Video Decoder）负责将H264/H265格式的视频码流解码为YUV/RGB格式的图片。关于VDEC功能的详细介绍及约束请参见**10.7.9.1 功能及约束说明**。

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。

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**3.4 pyACL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 视频解码流程



实现视频的解码，关键接口的说明如下：

1. **调用acl.media.vdec_create_channel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acl.media.vdec_create_channel_desc接口创建通道描述信息。
     ii. 调用acl.media.vdec_set_channel_desc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10.7.9.11 函数：vdec_set_channel_desc_callback。

       在回调函数内，用户需调用acl.media.dvpp_get_pic_desc_ret_code接口获取“ret_code”返回码判断是否解码成功，“ret_code”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acl.rt.process_report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acl.media.vdec_set_channel_desc_out_pic_format接口设置输出格式，则默认使用“YUV420SP NV12”。
   - acl.media.vdec_create_channel接口内部封装了如下接口，无需用户单独调用：
     i. acl.rt.create_stream接口：显式创建Stream，VDEC内部使用。
     ii. acl.rt.subscribe_report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acl.media.vdec_set_channel_desc系列接口时指定的。
2. **调用acl.media.vdec_send_frame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acl.media.dvpp_create_stream_desc接口创建输入视频码流描述信息，并调用acl.media.dvpp_set_stream_desc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acl.media.dvpp_create_pic_desc接口创建输出图片描述信息，并调用acl.media.dvpp_set_pic_desc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：**
     - acl.media.vdec_send_frame接口内部封装了acl.rt.launch_callback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acl.rt.launch_callback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“ret_code”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**acl.media.vdec_send_frame**接口的“user_data”参数处定义，然后解码的帧序号可以通过“user_data”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**acl.media.vdec_send_skipped_frame**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“dvpp_pic_desc”为0。

3. **调用acl.media.vdec_destroy_channel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。
   - **acl.media.vdec_destroy_channel**接口内部封装了如下接口，无需用户单独调用。
     - **acl.rt.unsubscribe_report**接口：取消线程注册（Stream上的回调函数不再由指定线程处理）。
     - **acl.rt.destroy_stream**接口：销毁Stream。
   - 销毁通道后，需调用**acl.media.vdec_destroy_channel_desc**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**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，不可以直接拷贝运行，仅供参考。

```
import acl
# ......

# 1.pyACL初始化。
ret = acl.init()

# 2.运行管理资源申请,包括Device、Context、Stream。
ret = acl.rt.set_device(self.device_id)
self.context, ret = acl.rt.create_context(self.device_id)
self.stream, ret = acl.rt.create_stream()

# 3.创建回调函数。
def _callback(self, input_stream_desc, output_pic_desc, user_data):
    # 输入码流描述。
    if input_stream_desc:
        ret = acl.media.dvpp_destroy_stream_desc(input_stream_desc)
        if ret != 0:
            print("acl.media.dvpp_destroy_stream_desc failed")
    # 输出图片描述。
    if output_pic_desc:
        vdec_out_buffer = acl.media.dvpp_get_pic_desc_data(output_pic_desc)
        ret_code = acl.media.dvpp_get_pic_desc_ret_code(output_pic_desc)
        data_size = acl.media.dvpp_get_pic_desc_size(output_pic_desc)
        self.images_buffer.append(dict({"buffer": vdec_out_buffer,
                                        "size": data_size}))
        ret = acl.media.dvpp_destroy_pic_desc(output_pic_desc)
        if ret != 0:
            print("acl.media.dvpp_destroy_pic_desc failed")
    self.output_count += 1
    print("[Vdec] [_callback] _callback exit success")
    # 在此样例中，释放vdec_out_buffer在vdec流程之后处理。

# 4.创建回调处理线程，并提交注册由此线程执行回调函数。
timeout = 100
```

```
cb_thread_id, ret = acl.util.start_thread(self._thread_func, [timeout])
acl.rt.subscribe_report(cb_thread_id, self.stream)

# 5.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。
def init_resource(self, cb_thread_id):
    print("[Vdec] class Vdec init resource stage:")
    self.vdec_channel_desc = acl.media.vdec_create_channel_desc()
    acl.media.vdec_set_channel_desc_channel_id(self.vdec_channel_desc, self._channel_id)
    acl.media.vdec_set_channel_desc_thread_id(self.vdec_channel_desc, cb_thread_id)
    acl.media.vdec_set_channel_desc_callback(self.vdec_channel_desc, self._callback)
    acl.media.vdec_set_channel_desc_entype(self.vdec_channel_desc, self._en_type)
    acl.media.vdec_set_channel_desc_out_pic_format(self.vdec_channel_desc, self._format)
    acl.media.vdec_create_channel(self.vdec_channel_desc)
    print("[Vdec] class Vdec init resource stage success")

# 6.创建视频码流处理的通道。
ret = acl.media.vdec_create_channel(self.vdec_channel_desc)
# ......
# 7.申请Device内存dataDev，存放视频解码的输入视频数据。
# 将通过acl.rt.memcpy接口将Host的图片数据传输到Device，数据传输完成后，需及时调用acl.rt.free_host接口释放Host内存。
# 输出图片尺寸对齐方式设置。
output_pic_size = (self.input_width * self.input_height * 3) // 2
img = np.fromfile(img_path, dtype=self.dtype)
input_stream_size = img.size
bytes_data = img.tobytes()
img_ptr = acl.util.bytes_to_ptr(bytes_data)
self.input_stream_mem, ret = acl.media.dvpp_malloc(input_stream_size)
ret = acl.rt.memcpy(self.input_stream_mem,
                    input_stream_size,
                    img_ptr,
                    input_stream_size,
                    ACL_MEMCPY_HOST_TO_DEVICE)

# 8.循环10次执行视频解码，输出10张YUV420SP NV12格式的图片。
def forward(self, output_pic_size, input_stream_size):
    self.frame_config = acl.media.vdec_create_frame_config()
    for i in range(self.rest_len):
        print("[Vdec] forward index:{}".format(i))
        # 8.1 创建输入视频码流描述信息，设置码流信息的属性。
        self._set_input(input_stream_size)
        # 8.2 创建输出图片描述信息，设置图片描述信息的属性。
        self._set_pic_output(output_pic_size)
        # 8.3 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 = acl.media.vdec_send_frame(self.vdec_channel_desc,
                                        self.dvpp_stream_desc,
                                        self.dvpp_pic_desc,
                                        self.frame_config,
                                        None)
        check_ret("acl.media.vdec_send_frame", ret)
        print('[Vdec] vdec_send_frame stage success')

# 9.释放图片处理通道、图片描述信息。
ret = acl.media.vdec_destroy_channel(self.vdec_channel_desc)
acl.media.vdec_destroy_channel_desc(self.vdec_channel_desc)

# 10. 释放运行管理资源。
ret = acl.rt.destroy_dtream(self.stream)
ret = acl.rt.destroy_context(self.context)
ret = acl.rt.reset_device(self.device_id)
ret = acl.finalize()

# 11.pyACL去初始化。

# ....
```

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。	日志的详细介绍，请参见《日志参考》。
AICPU_DVPP_KERNEL_STATE_DVPP_ERROR = 2	AscendCL内部调用其它模块的接口失败。	日志的详细介绍，请参见《日志参考》。
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。	日志的详细介绍，请参见《日志参考》。
AICPU_DVPP_KERNEL_STATE_QUEUE_FULL = 6	系统内部队列满。	日志的详细介绍，请参见《日志参考》。
AICPU_DVPP_KERNEL_STATE_QUEUE_EMPTY = 7	系统内部队列空。	日志的详细介绍，请参见《日志参考》。
AICPU_DVPP_KERNEL_STATE_QUEUE_NOT_EXIST = 8	系统内部队列不存在。	日志的详细介绍，请参见《日志参考》。
AICPU_DVPP_KERNEL_STATE_GET_CONTEX_FAILED = 9	获取系统内部上下文失败。	日志的详细介绍，请参见《日志参考》。
AICPU_DVPP_KERNEL_STATE_SUBMIT_EVENT_FAILED = 10	提交系统内部事件失败。	日志的详细介绍，请参见《日志参考》。
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。	日志的详细介绍，请参见《日志参考》。
AICPU_DVPP_KERNEL_STATE_VPC_OPERATE_FAILED = 13	系统内部接口处理失败。	日志的详细介绍，请参见《日志参考》。

版权所有 © 华为技术有限公司

返回码	含义	可能原因及解决方法
AICPU_DVPP_KERNEL_STATE_CHANNEL_ABNORMAL = 14	当前通道异常。	日志的详细介绍，请参见《日志参考》。
ERR_INVALID_STATE = 0x10001	VDEC解码器状态异常错误。	日志的详细介绍，请参见《日志参考》。
ERR_HARDWARE = 0x10002	硬件错误，包含解码器启动、执行、停止等异常。	日志的详细介绍，请参见《日志参考》。
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**。
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。

返回码	含义	可能原因及解决方法
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。	日志的详细介绍，请参见《日志参考》。
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）。	日志的详细介绍，请参见《日志参考》。
ERR_GET_CHANNEL_HANDLE_FAIL = 0x10011	系统内部获取某路视频流的解码句柄失败。	日志的详细介绍，请参见《日志参考》。
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。
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。	请联系工程师。
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。	-

返回码	含义	可能原因及解决方法
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。
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。 暂未使用，预留。	-
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。 日志的详细介绍，请参见《日志参考》。
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。 日志的详细介绍，请参见《日志参考》。
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。 日志的详细介绍，请参见《日志参考》。
0xA0058008	不支持的参数或者功能。	请根据日志检查接口的入参。 日志的详细介绍，请参见《日志参考》。
0xA0058009	该操作不允许，如试图修改静态配置参数。	日志的详细介绍，请参见《日志参考》。
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。**

版权所有 © 华为技术有限公司

返回码	含义	可能原因及解决方法
0xA005800D	分配缓存失败，如申请的数据缓冲区太大。暂未使用，预留。	-
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。	日志的详细介绍，请参见《日志参考》。
0xA0058012	系统忙。	日志的详细介绍，请参见《日志参考》。
0xA0058013	缓存小于实际需要的大小。 暂未使用，预留。	-
0xA0058014	硬件或软件处理超时。 暂未使用，预留。	-
0xA0058015	内部系统错误。	日志的详细介绍，请参见《日志参考》。
0xA005803F	最大的返回码，该模块的错误码必须小于该值。	-

5.3.7 VENC 视频编码

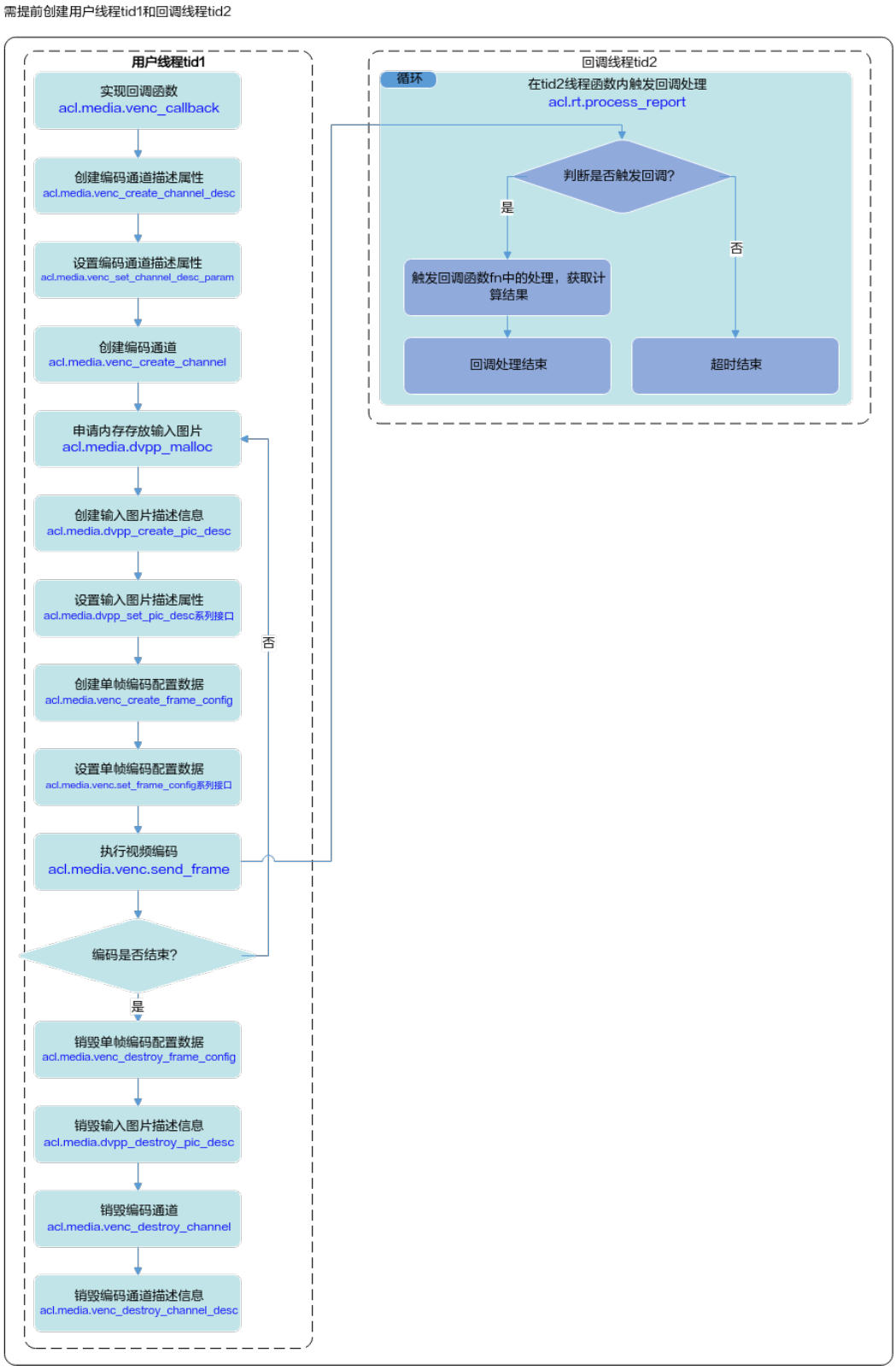
VENC（Video Encoder）将YUV420SP格式的图片编码成H264/H265格式的视频码流。关于VENC功能的详细介绍及约束请参见**10.7.10.1 功能及约束说明**。

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。

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**3.4 pyACL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 视频编码流程



实现视频的编码，关键接口的说明如下：

1. **调用acl.media.venc_create_channel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**acl.media.venc_create_channel_desc**接口创建通道描述信息。
  ii. 调用**acl.media.venc_set_channel_desc_param**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，回调函数的原型前参见**10.7.10.7 函数：venc_set_channel_desc_callback**。

        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**acl.rt.process_report**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**acl.media.venc_set_channel_desc_param**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acl.media.venc.set_channel_desc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。
- **acl.media.venc_create_channel**接口内部封装了如下接口，无需用户单独调用：
  i. **acl.rt.create_stream**接口：显式创建Stream，VENC内部使用。
  ii. **acl.rt.subscribe_report**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**acl.media.venc_set_channel_desc_param**接口时指定的。

2. **调用acl.media.venc_send_frame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**acl.media.dvpp_create_pic_desc**接口创建输入图片描述信息，并调用acl.media.dvpp_set_pic_desc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**acl.media.venc_create_frame_config**接口创建单帧编码配置数据，并调用acl.media.venc_set_frame_config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**，**acl.media.venc_send_frame**接口内部封装了**acl.rt.launch_callback**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**acl.rt.launch_callback**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acl.media.venc_destroy_channel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。
   - **acl.media.venc_destroy_channel**接口内部封装了如下接口，无需用户单独调用。
     - **acl.rt.unsubscribe_report**接口：取消线程注册（Stream上的回调函数不再由指定线程处理）。

- **acl.rt.destroy_stream**接口：销毁Stream。
  - 销毁通道后，需调用**acl.media.venc_destroy_channel_desc**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

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，不可以直接拷贝运行，仅供参考。

```
import acl
# ......

# 1.资源初始化：创建 AclVenc 对象进行初始化。
# 1.1 pyACL初始化。
ret = acl.init()
# 1.2 运行管理资源申请,包括Device、Context。
device_id = 0
ret = acl.rt.set_device(device_id)
self.context, ret = acl.rt.create_context(device_id)
# 调用acl.rt.get_run_mode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。
runMode, ret = acl.rt.get_run_mode()

# 2.创建执行回调函数的线程及线程函数。
def cb_thread_func(self, args_list):
    ctx = args_list[0]
    timeout = args_list[1]
    print("[info] thread args_list = ", self.ctx, timeout, self.g_callbackRunFlag, "\n")
    ret = acl.rt.set_context(ctx)
    assert ret == 0

    while self.g_callbackRunFlag is True:
        print("[info] thread g_callbackRunFlag = ", self.g_callbackRunFlag, "\n")
        ret = acl.rt.process_report(timeout)
        print("[info] acl.rt.process_report =", ret)

timeout = 1000
g_callbackRunFlag = True
self.cb_thread_id, ret = acl.util.start_thread(self.cb_thread_func, [self.ctx, timeout])

# 3.创建回调函数。
# 3.1 获取流数据输出。
def get_stream_data(self, stream_desc):
    stream_data = acl.media.dvpp_get_stream_desc_data(stream_desc)
    assert stream_data is not None
    stream_data_size = acl.media.dvpp_get_stream_desc_size(stream_desc)
    print("[info] stream_data size", stream_data_size)
    # stream memcpy d2h
    np_data = np.zeros(stream_data_size, dtype=np.byte)

    bytes_data = np_data.tobytes()
    np_data_ptr = acl.util.bytes_to_ptr(bytes_data)
    ret = acl.rt.memcpy(np_data_ptr, stream_data_size,
                        stream_data, stream_data_size,
                        memcpy_kind.get("ACL_MEMCPY_DEVICE_TO_HOST"))
    assert ret == 0
    return np_data
# 3.2 回调函数将视频流保存文件。
def callback_func(self, input_pic_desc, output_stream_desc, user_data):
    # 获取视频编码结果数据, 转换成numpy对象。
    output_numpy = self.get_stream_data(output_stream_desc)
    with open('./data/output.h265', 'ab') as f:
        f.write(output_numpy)
    print("[INFO] [callback_func] stream size =",
acl.media.dvpp_get_stream_desc_size(output_stream_desc))
```

```
# 4.创建视频编码处理通道时的通道描述信息。
self.venc_channel_desc = acl.media.venc_create_channel_desc()

# 5.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。
venc_format = 1
venc_entype = 0
input_width = 1280
input_height = 720
ret = acl.media.venc_set_channel_desc_thread_id(self.venc_channel_desc, self.cb_thread_id)
ret = acl.media.venc_set_channel_desc_callback(self.venc_channel_desc, self.callback_func)
ret = acl.media.venc_set_channel_desc_entype(self.venc_channel_desc, venc_entype)
ret = acl.media.venc_set_channel_desc_pic_format(self.venc_channel_desc, venc_format)
ret = acl.media.venc_set_channel_desc_key_frame_interval(self.venc_channel_desc, 16)
ret = acl.media.venc_set_channel_desc_pic_height(self.venc_channel_desc, input_height)
ret = acl.media.venc_set_channel_desc_pic_width(self.venc_channel_desc, input_width )

# 6.创建视频码流处理的通道、单帧编码配置数据。
ret = acl.media.venc_create_channel(self.venc_channel_desc)
# vencFrameConfig_是aclvencFrameConfig类型。
frame_config = acl.media.venc_create_frame_config()

# 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。
output_pic_size = (input_width * input_height * 3) // 2
print("[INFO] output_pic_size:", output_pic_size, " load vdec file:", venc_file_path)
file_context = np.fromfile(venc_file_path, dtype=np.byte)
file_size = file_context.size
bytes_data = file_context.tobytes()
file_mem = acl.util.bytes_to_ptr(bytes_data)
input_size = file_size
# 如果调用acl.rt.get_run_mode接口获取软件栈的运行模式为ACL_HOST，则需要通过acl.rt.memcpy接口将
Host的图片数据传输到Device，数据传输完成后，需及时调用acl.rt.free_host接口释放Host内存。
# 如果调用acl.rt.get_run_mode接口获取软件栈的运行模式为ACL_DEVICE，则直接申请Device内存，存放输入
图片数据。
# 此处是运行模式为 ACL_HOST。
input_mem, ret = acl.media.dvpp_malloc(input_size)
assert ret == 0
ret = acl.rt.memcpy(input_mem, input_size, file_mem,
                    file_size, memcpy_kind.get("ACL_MEMCPY_HOST_TO_DEVICE"))
assert ret == 0

# 8.执行视频编码。
def venc_set_frame_config(self, frame_confg, eos, iframe):
    ret = acl.media.venc_set_frame_config_eos(frame_confg, eos)
    assert ret == 0
    ret = acl.media.venc_set_frame_config_force_i_frame(frame_confg, iframe)
    assert ret == 0

def venc_process(self, venc_channel_desc, input_mem, input_size, frame_config):
   # 设置为非结束帧。
    self.venc_set_frame_config(frame_config, 0, 0)
    print("[INFO] set frame config")
    self.test_get_frame_config(frame_config)

    # set picture description
    dvpp_pic_desc = acl.media.dvpp_create_pic_desc()
    assert dvpp_pic_desc is not None
    ret = acl.media.dvpp_set_pic_desc_data(dvpp_pic_desc, input_mem)
    assert ret == 0
    ret = acl.media.dvpp_set_pic_desc_size(dvpp_pic_desc, input_size)
    assert ret == 0
    print("[INFO] set pic desc")

    # 发送1张图片到编码器进行编码。
    venc_cnt = 16
    while venc_cnt:
        ret = acl.media.venc_send_frame(venc_channel_desc, dvpp_pic_desc, 0, frame_config, 0)
        assert ret == 0
```

```
        print("[INFO] venc send frame")
        venc_cnt -= 1
    # 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。
    self.venc_set_frame_config(frame_config, 1, 0)
    ret = acl.media.venc_send_frame(venc_channel_desc, 0, 0, frame_config, 0)
    assert ret == 0

# 9.释放资源。
ret = acl.media.dvpp_free(input_mem)
ret = acl.media.venc_destroy_frame_config(frame_config)
assert ret == 0
print("[INFO] free resources")
g_callbackRunFlag = False
ret = acl.util.stop_thread(self.cb_thread_id)
print("[INFO] thread join")

# 10. 释放运行管理资源。
ret = acl.rt.destroy_stream(self.stream)
ret = acl.rt.destroy_context(self.context)
ret = acl.rt.reset_device(self.device_id);
ret = acl.finalize()

# 11.pyACL去初始化。

# ....
```

如果调用**acl.media.venc_set_channel_desc_param**接口设置通道描述信息的属性，调用**acl.media.venc_get_channel_desc_param**接口获取通道描述信息中的属性值，示例代码如下：

```
ACL_VENC_THREAD_ID_UINT64 = 0          #回调线程ID。
ACL_VENC_CALLBACK_PTR = 1              #回调函数。
ACL_VENC_PIXEL_FORMAT_UINT32 = 2       #输入图像格式。
ACL_VENC_ENCODE_TYPE_UINT32 = 3        #视频编码协议。
ACL_VENC_PIC_WIDTH_UINT32 = 4          #输入图片宽度。
ACL_VENC_PIC_HEIGHT_UINT32 = 5         #输入图片高度。
ACL_VENC_KEY_FRAME_INTERVAL_UINT32 = 6 #关键帧间隔。
ACL_VENC_BUF_ADDR_PTR = 7              #编码输出缓存地址。
ACL_VENC_BUF_SIZE_UINT32 = 8           #编码输出缓存大小。
ACL_VENC_RC_MODE_UINT32 = 9            #码率控制模式。
ACL_VENC_SRC_RATE_UINT32 = 10          #输入码流帧率。
ACL_VENC_MAX_BITRATE_UINT32 = 11       #输出码率。
ACL_VENC_MAX_IP_PROP_UINT32 = 12       #一个GOP内单个I帧bit数和单个P帧bit数的比例。

# 设置回调函数。
ret = acl.media.venc_set_channel_desc_param(self.venc_channel_desc, ACL_VENC_CALLBACK_PTR,
self.callback_func)
# 获取回调函数。
get_cb_func, ret = acl.media.venc_get_channel_desc_param(self.venc_channel_desc,
ACL_VENC_CALLBACK_PTR)

PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1
# 设置输入图片格式。
ret = acl.media.venc_set_channel_desc_param(self.venc_channel_desc,
ACL_VENC_PIXEL_FORMAT_UINT32, PIXEL_FORMAT_YUV_SEMIPLANAR_420)
# 获取输入图片格式。
get_pic_format, ret = acl.media.venc_get_channel_desc_param(self.venc_channel_desc,
ACL_VENC_PIXEL_FORMAT_UINT32)

width = 128
# 设置图片宽度。
ret = acl.media.venc_set_channel_desc_param(self.venc_channel_desc, ACL_VENC_PIC_WIDTH_UINT32,
width)
# 获取图片宽度。
get_width, ret = acl.media.venc_get_channel_desc_param(self.venc_channel_desc,
ACL_VENC_PIC_WIDTH_UINT32)

height = 128
# 设置图片高度。
```

```
ret = acl.media.venc_set_channel_desc_param(self.venc_channel_desc, ACL_VENC_PIC_HEIGHT_UINT32,
height)
# 获取图片高度。
get_height, ret = acl.media.venc_get_channel_desc_param(self.venc_channel_desc,
ACL_VENC_PIC_HEIGHT_UINT32)
```

5.4 媒体数据处理 V2

5.4.1 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 200/500 A2 推理产品	√	√	√	√	√	√

5.4.2 VPC 图片处理典型功能

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。关于VPC功能的详细介绍请参见**10.8.4.1 VPC功能**，关于VPC功能对输入、输出的约束要求，请参见**10.8.4.2 约束说明**。

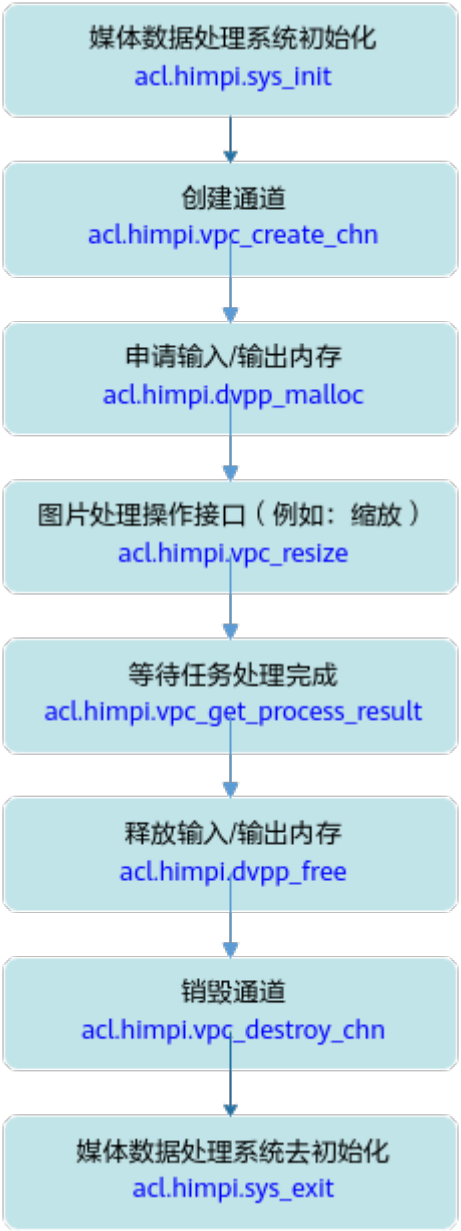
本节以抠图、缩放为例说明VPC图像处理时的接口调用流程，同时配合以下典型功能的示例代码辅助理解该接口调用流程：

- 图片缩放示例代码
- 抠图示例代码

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**3.4 pyACL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 接口调用流程（以缩放为例）



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**acl.himpi.sys_init**接口进行媒体数据处理系统初始化。
2. 调用**acl.himpi.vpc_create_chn**接口创建通道。
3. 调用**acl.himpi.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**10.7.6 VPC功能**下的接口说明。

   调用**acl.himpi.vpc_resize**接口，按指定区域从输入图片中抠图，抠出的图作为输出图片。**acl.himpi.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**acl.himpi.vpc_get_process_result**接口等待任务完成。

   – 可以跟**acl.himpi.vpc_resize**接口在同一个线程中调用**acl.himpi.vpc_get_process_result**接口，也可以新起一个线程调用

acl.himpi.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。

5. 调用acl.himpi.dvpp_free接口释放输入、输出内存。
6. 调用acl.himpi.vpc_destroy_chn接口销毁通道。
7. 调用acl.himpi.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

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）。
run_mode, ret = acl.rt.get_run_mode()

# 2.pyACL 初始化。
ret = acl.init()

# 3.运行管理资源申请(依次申请Device、Context)。
ret = acl.rt.set_device(0)
context, ret = acl.rt.create_context(0)

# 4.初始化媒体数据处理系统。
ret = acl.himpi.sys_init()

# 5.创建通道。
channel_id = 0
chn_attr = {'attr': 0, 'pic_width': 0, 'pic_height': 0}
ret = acl.himpi.vpc_create_chn(channel_id, chn_attr)

# 6.执行缩放。
# 6.1 构造存放输入图片信息的字典。
input_pic = {'picture_width': 1920,
             'picture_height': 1080,
             'picture_width_stride': 1920,
             'picture_height_stride': 1080,
             'picture_format': HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420}
input_pic["picture_buffer_size"] = input_pic["picture_width_stride"] * input_pic["picture_height_stride"] * 3 //
2
picture_address, ret = acl.himpi.dvpp_malloc(0, input_pic["picture_buffer_size"])
input_pic["picture_address"] = picture_address

# 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acl.rt.memcpy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。
# 直接将输入图片数据读入Device内存。
if run_mode == ACL_HOST:
    # 将输入图片读入内存中。
    vpc_file = np.fromfile(vpc_file_path, dtype=np.byte)
    vpc_file_size = vpc_file.itemsize * vpc_file.size

    bytes_data = vpc_file.tobytes()
    vpc_file_ptr = acl.util.bytes_to_ptr(bytes_data)
    # 数据传输。
    ret = acl.rt.memcpy(input_pic["picture_address"], input_pic["picture_buffer_size"],
                        vpc_file_ptr, vpc_file_size, ACL_MEMCPY_HOST_TO_DEVICE)
else:
    # 将输入图片读入内存中。
    vpc_file = np.fromfile(vpc_file_path, dtype=np.byte)
    vpc_file_size = vpc_file.itemsize * vpc_file.size

    bytes_data = vpc_file.tobytes()
    vpc_file_ptr = acl.util.bytes_to_ptr(bytes_data)
    # 数据传输。
```

```
    ret = acl.rt.memcpy(input_pic["picture_address"], input_pic["picture_buffer_size"],
                        vpc_file_ptr, vpc_file_size, ACL_MEMCPY_DEVICE_TO_DEVICE)


# 6.3 构造存放输出图片信息的字典。
output_pic = {'picture_width': 960,
              'picture_height': 540,
              'picture_width_stride': 960,
              'picture_height_stride': 540,
              'picture_format': HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420}
output_pic["picture_buffer_size"] = output_pic["picture_width_stride"] * output_pic["picture_height_stride"]
* 3 // 2
picture_address, ret = acl.himpi.dvpp_malloc(0, output_pic["picture_buffer_size"])
output_pic["picture_address"] = picture_address
# 初始化内存。
ret = acl.rt.memset(output_pic["picture_address"], output_pic["picture_buffer_size"], 0,
output_pic["picture_buffer_size"])

# 6.4 调用缩放接口。
testid, ret = acl.himpi.vpc_resize(channel_id, input_pic, output_pic, 0, 0, 0, -1)

# 6.5 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。
ret = acl.himpi.vpc_get_process_result(channel_id, task_id, -1)

# 6.6 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
acl.rt.memcpy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。
# 6.6 VPC的输出图片数据可以直接作为模型推理的输入。
if run_mode == ACL_HOST:
        np_output = np.zeros(output_pic.get('picture_buffer_size'), dtype=np.byte)

        bytes_data = np_output.tobytes()
        np_output_ptr = acl.util.bytes_to_ptr(bytes_data)
        ret = acl.rt.memcpy(np_output_ptr, output_pic.get('picture_buffer_size'),
output_pic.get("picture_address"),
                            output_pic.get('picture_buffer_size'), ACL_MEMCPY_DEVICE_TO_HOST)
        # ......
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。
    # ......
}

# 6.7 释放输入、输出内存。
ret = acl.himpi.dvpp_free(input_pic['picture_address'])
ret = acl.himpi.dvpp_free(output_pic['picture_address'])

# 7.销毁通道。
ret = acl.himpi.vpc_destroy_chn(channel_id)

# 8. 媒体数据处理系统去初始化。
ret = acl.himpi.sys_exit()

# 9. 释放运行管理资源(依次释放Context、Device)。
ret = acl.rt.destroy_context(context)
ret = acl.rt.reset_device(0)

# 10.pyACL去初始化。
ret = acl.finalize()
```

抠图示例代码

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）。
run_mode, ret = acl.rt.get_run_mode()

# 2.pyACL 初始化。
ret = acl.init()
```

```
# 3.运行管理资源申请(依次申请Device、Context)。
ret = acl.rt.set_device(0)
context, ret = acl.rt.create_context(0)

# 4.初始化媒体数据处理系统。
ret = acl.himpi.sys_init()

# 5.创建通道。
channel_id = 0
chn_attr = {'attr': 0, 'pic_width': 0, 'pic_height': 0}
ret = acl.himpi.vpc_create_chn(channel_id, chn_attr)

# 6.执行抠图。
# 6.1 构造存放输入图片信息的字典。
input_pic = {'picture_width': 1920,
             'picture_height': 1080,
             'picture_width_stride': 1920,
             'picture_height_stride': 1080,
             'picture_format': HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420}
input_pic["picture_buffer_size"] = input_pic["picture_width_stride"] * input_pic["picture_height_stride"] * 3 //
2
picture_address, ret = acl.himpi.dvpp_malloc(0, input_pic["picture_buffer_size"])
input_pic["picture_address"] = picture_address

# 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acl.rt.memcpy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。
# 直接将输入图片数据读入Device内存。
if run_mode == ACL_HOST:
    # 将输入图片读入内存中。
    vpc_file = np.fromfile(vpc_file_path, dtype=np.byte)
    vpc_file_size = vpc_file.itemsize * vpc_file.size

    bytes_data = vpc_file.tobytes()
    vpc_file_ptr = acl.util.bytes_to_ptr(bytes_data)
    # 数据传输。
    ret = acl.rt.memcpy(input_pic["picture_address"], input_pic["picture_buffer_size"],
                        vpc_file_ptr, vpc_file_size, ACL_MEMCPY_HOST_TO_DEVICE)
else:
    # 将输入图片读入内存中。
    vpc_file = np.fromfile(vpc_file_path, dtype=np.byte)
    vpc_file_size = vpc_file.itemsize * vpc_file.size

    bytes_data = vpc_file.tobytes()
    vpc_file_ptr = acl.util.bytes_to_ptr(bytes_data)
    # 数据传输。
    ret = acl.rt.memcpy(input_addr, input_size, vpc_file_ptr, vpc_file_size, ACL_MEMCPY_DEVICE_TO_DEVICE)

# 6.3 构造存放输出图片信息的字典。
multithreading_count = 1
crop_region_infos = []
for i in range(multithreading_count):
    output_pic = {'picture_width': 960,
                  'picture_height': 540,
                  'picture_width_stride': 960,
                  'picture_height_stride': 540,
                  'picture_format': HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420}
    output_pic["picture_buffer_size"] = output_pic["picture_width_stride"] *
output_pic["picture_height_stride"] * 3 // 2
    picture_address, ret = acl.himpi.dvpp_malloc(0, output_pic["picture_buffer_size"])
    output_pic["picture_address"] = picture_address

    ret = acl.rt.memset(output_pic["picture_address"], output_pic["picture_buffer_size"], 0,
output_pic["picture_buffer_size"])
    crop_region = {'top_offset': 0, 'left_offset': 0, 'crop_width': 960, 'crop_height': 540}
    crop_region_info = {'dest_pic_info': output_pic, 'crop_region': crop_region}
    crop_region_infos.append(crop_region_info)
```

```
# 6.4 调用抠图接口。
task_id, ret = acl.himpi.vpc_crop(channel_id, input_pic, crop_region_infos, 1, -1)

# 6.5 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。
ret = acl.himpi.vpc_get_process_result(channel_id, task_id, -1)

# 6.6 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
acl.rt.memcpy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。
# 6.6 VPC的输出图片数据可以直接作为模型推理的输入。
if run_mode == ACL_HOST:
    np_output = np.zeros(output_pic.get('picture_buffer_size'), dtype=np.byte)

    bytes_data = np_output.tobytes()
    np_output_ptr = acl.util.bytes_to_ptr(bytes_data)
    ret = acl.rt.memcpy(np_output_ptr, output_pic.get('picture_buffer_size'), output_pic.get("picture_address"),
                        output_pic.get('picture_buffer_size'), ACL_MEMCPY_DEVICE_TO_HOST)
    # ......
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。
    # ......
}

# 6.7 释放输入、输出内存。
ret = acl.himpi.dvpp_free(input_pic['picture_address'])
ret = acl.himpi.dvpp_free(output_pic['picture_address']))

# 7.销毁通道。
ret = acl.himpi.vpc_destroy_chn(channel_id)

# 8. 媒体数据处理系统去初始化。
ret = acl.himpi.sys_exit()

# 9. 释放运行管理资源(依次释放Context、Device)。
ret = acl.rt.destroy_context(context)
ret = acl.rt.reset_device(0)

# 10.pyACL去初始化。
ret = acl.finalize()
```

5.4.3 JPEGD 图片解码

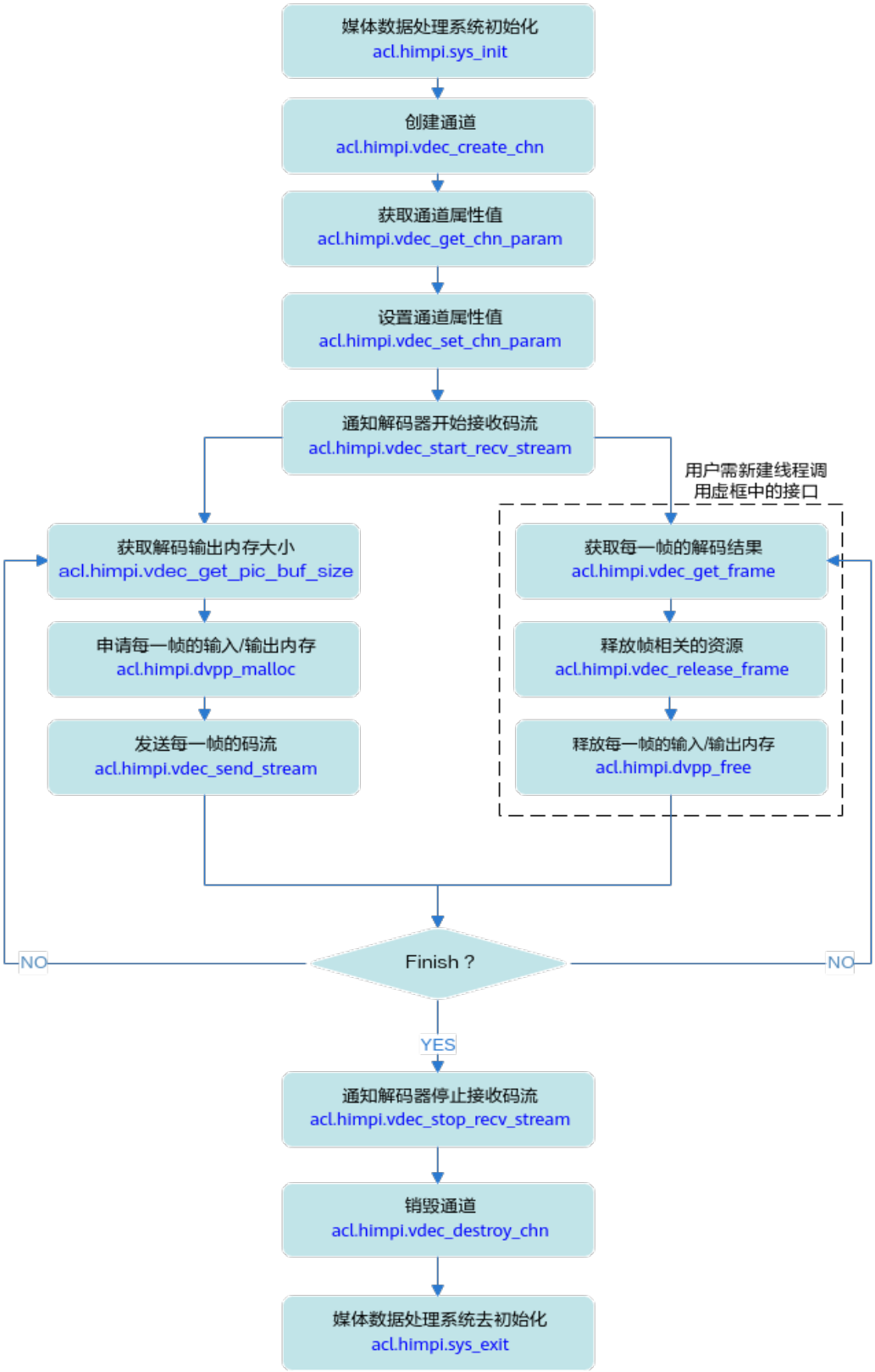
JPEGD（JPEG Decoder）负责完成图像解码功能，将.jpg、.jpeg、.JPG、.JPEG图片解码成YUV格式图片。关于JPEGD功能的详细介绍及约束请参见**10.8.5.1 JPEGD功能及约束说明**。

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。

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**3.4 pyACL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 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**acl.himpi.sys_init**接口进行媒体数据处理系统初始化。
2. 调用**acl.himpi.vdec_create_chn**接口创建通道。
3. 调用**acl.himpi.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**acl.himpi.vdec_start_recv_stream**接口通知解码器启动接收码流，再调用**acl.himpi.vdec_send_stream**接口发送解码码流，**acl.himpi.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**acl.himpi.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**acl.himpi.vdec_release_frame**接口释放帧相关的资源。

   解码结束后，需调用**acl.himpi.vdec_stop_recv_stream**接口通知解码器停止接收码流。
5. 调用**acl.himpi.dvpp_free**接口释放输入、输出内存。
6. 调用**acl.himpi.vdec_destroy_chn**接口销毁通道。
7. 调用**acl.himpi.sys_exit**接口进行媒体数据处理系统去初始化。

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）。
run_mode, ret = acl.rt.get_run_mode()

# 2.pyACL 初始化。
ret = acl.init()

# 3.运行管理资源申请(依次申请Device、Context)。
ret = acl.rt.set_device(0)
context, ret = acl.rt.create_context(0)

# 4.初始化媒体数据处理系统。
ret = acl.himpi.sys_init()

# 5.创建通道。
channel_id = 0
attr = {'type': HI_PT_JPEG, 'mode': HI_VDEC_SEND_MODE_FRAME,
        'pic_width': 1920, 'pic_height': 1080,
        'stream_buf_size': 1920 * 1080, 'frame_buf_size': 0,
        'frame_buf_cnt': 9}

ret = acl.himpi.vdec_create_chn(channel_id, attr)
# 6.设置通道属性。
jpegd_param_dict, ret = acl.himpi.vdec_get_chn_param(channel_id)
jpegd_param_dict["pic_param"]["pixel_format"] = HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420
jpegd_param_dict["pic_param"]["alpha"] = 255
jpegd_param_dict["display_frame_num"] = 0
ret = acl.himpi.vdec_set_chn_param(i, jpegd_param_dict)

# 7.解码器启动接收码流。
ret = acl.himpi.vdec_start_recv_stream(channel_id)

# 8.发送码流。
# 8.1 申请输入内存。
# input_size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。
input_size = 1024;
input_addr, ret = acl.himpi.dvpp_malloc(0, input_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acl.rt.memcpy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。
#直接将输入图片数据读入Device内存。
if run_mode == ACL_HOST:
    # 申请Host内存。
```

```
    input_buffer, ret= acl.rt.malloc_host(input_size)
    # 将输入图片读入内存中。
    vdec_file = np.fromfile(vdec_file_path, dtype=np.byte)
    vdec_file_size = vdec_file.itemsize * vdec_file.size

    bytes_data = vdec_file.tobytes()
    vdec_file_ptr = acl.util.bytes_to_ptr(bytes_data)
    # 数据传输。
    ret = acl.rt.memcpy(input_addr, input_size, vdec_file_ptr, vdec_file_size, ACL_MEMCPY_HOST_TO_DEVICE)
else:
    # 将输入图片读入内存中。
    vdec_file = np.fromfile(vdec_file_path, dtype=np.byte)
    vdec_file_size = vdec_file.itemsize * vdec_file.size

    bytes_data = vdec_file.tobytes()
    vdec_file_ptr = acl.util.bytes_to_ptr(bytes_data)
    # 数据传输。
    ret = acl.rt.memcpy(input_addr, input_size, vdec_file_ptr, vdec_file_size,
ACL_MEMCPY_DEVICE_TO_DEVICE)

# 8.2 构造存放输入图片信息的字典。
stream = {'end_of_frame': HI_TRUE, 'end_of_stream': HI_FALSE,
          'need_display': HI_TRUE, 'pts': 0,
          'len': input_size, 'addr': vdec_file_ptr}
img_info, ret = acl.himpi.dvpp_get_image_info(HI_PT_JPEG, stream)
if run_mode == ACL_HOST:
    # 如果不使用Host上的数据，需及时释放。
    ret = acl.rt.free_host(input_buffer)

stream['addr'] =  input_addr

# 8.3 构造存放输出图片信息的字典，并申请输出内存。
out_pic_info = {"width": img_info['width'],
                "height": img_info['height'],
                "width_stride": img_info['width_stride'],
                "height_stride": img_info['height_stride'],
                "pixel_format": HI_PIXEL_FORMAT_UNKNOWN,
                "buffer_size": img_info['img_buf_size']}
out_buffer, ret = acl.himpi.dvpp_malloc(0, out_pic_info['buffer_size'])
out_pic_info['vir_addr '] = out_buffer

# 8.4 发送需解码的输入图片。
ret = acl.himpi.vdec_send_stream(channel_id, stream, out_pic_info, 0)

# 9.接收解码结果。
# 9.1 获取解码结果。
frame_info, supplement, stream, ret = acl.himpi.vdec_get_frame(channel_id, 0)
if ret == 0:
    dec_result = frame_info['v_frame']['frame_flag']
    if dec_result == 0: #  0: Decode success
        print("Chn %u GetFrame Success, Decode Success \n"%channel_id)
    else: # Decode fail
        print("Chn %u GetFrame Success, Decode Fail \n"%channel_id)

# 9.2 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。
# 9.2 获取JPEGD的输出图片数据。

if run_mode == ACL_HOST:
    # 申请Host内存。
    output_buffer, ret= acl.rt.malloc_host(out_pic_info['buffer_size'])
    # 数据传输。
    ret = acl.rt.memcpy(output_buffer, out_pic_info['buffer_size'], frame_info['v_frame']['virt_addr'][0],
out_pic_info['buffer_size'], ACL_MEMCPY_DEVICE_TO_HOST)

    # ......
    # 数据使用完成后，及时释放不使用的内存。
    ret = acl.rt.free_host(output_buffer)
else:
```

```
    # 可以直接使用JPEGD的输出图片数据，在outputPic.picture_address指向的内存中。
    # ......

# 9.3 释放输入、输出内存。
ret = acl.himpi.dvpp_free(frame_info['v_frame']['virt_addr'][0])
ret = acl.himpi.dvpp_free(stream['addr'])

# 9.4 释放资源。
ret = acl.himpi.vdec_release_frame(channel_id, frame_info)

# 10.解码器停止接收码流。
ret = acl.himpi.vdec_stop_recv_stream(channel_id)

# 7.销毁通道。
ret = acl.himpi.vdec_destroy_chn(channel_id)

# 8. 媒体数据处理系统去初始化。
ret = acl.himpi.sys_exit()

# 9. 释放运行管理资源(依次释放Context、Device)。
ret = acl.rt.destroy_context(context)
ret = acl.rt.reset_device(0)

# 10.pyACL去初始化。
ret = acl.finalize()
```

5.4.4 JPEGE 图片编码

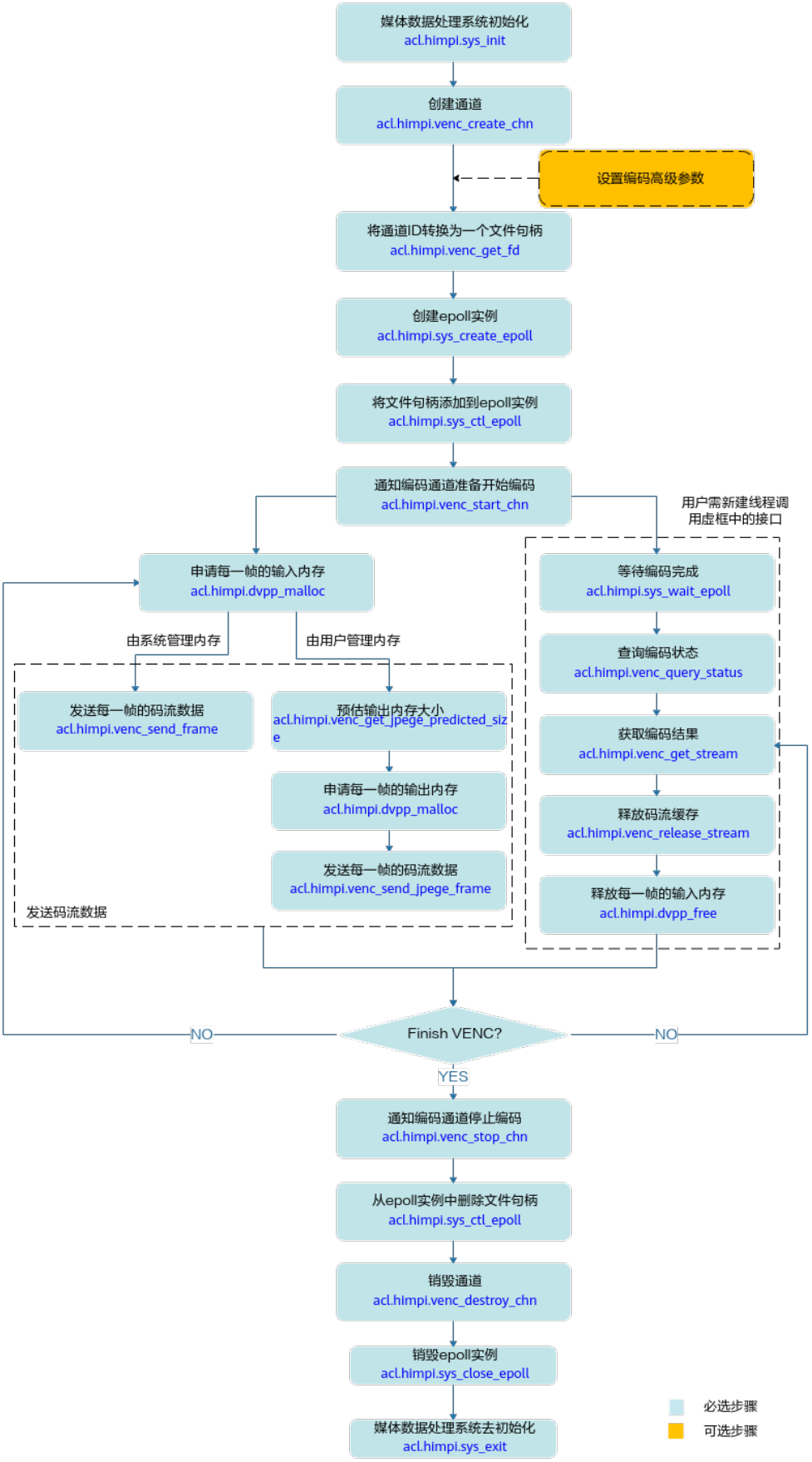
JPEGE（JPEG Encoder）负责完成图像编码功能，将YUV格式图片编码成.jpg图片。关于JPEGE功能的详细介绍请参见**10.8.6.1 JPEGE功能及约束说明**。

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。

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**3.4 pyACL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 接口调用流程


媒体数据处理系统初始化
acl.himpi.sys_init
创建通道
acl.himpi.venc_create_chn
设置编码高级参数
将通道ID转换为一个文件句柄
acl.himpi.venc_get_fd
创建epoll实例
acl.himpi.sys_create_epoll
将文件句柄添加到epoll实例
acl.himpi.sys_ctl_epoll
通知编码通道准备开始编码
acl.himpi.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
acl.himpi.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
acl.himpi.venc_send_frame
预估输出内存大小
acl.himpi.venc_get_jpege_predicted_size
申请每一帧的输出内存
acl.himpi.dvpp_malloc
发送每一帧的码流数据
acl.himpi.venc_send_jpege_frame
发送码流数据
等待编码完成
acl.himpi.sys_wait_epoll
查询编码状态
acl.himpi.venc_query_status
获取编码结果
acl.himpi.venc_get_stream
释放码流缓存
acl.himpi.venc_release_stream
释放每一帧的输入内存
acl.himpi.dvpp_free
NO
Finish VENC?
NO
YES
通知编码通道停止编码
acl.himpi.venc_stop_chn
从epoll实例中删除文件句柄
acl.himpi.sys_ctl_epoll
销毁通道
acl.himpi.venc_destroy_chn
销毁epoll实例
acl.himpi.sys_close_epoll
媒体数据处理系统去初始化
acl.himpi.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. 调用**acl.himpi.sys_init**接口进行媒体数据处理系统初始化。
2. 调用**acl.himpi.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**acl.himpi.venc_set_jpeg_param**~**acl.himpi.venc_compact_jpeg_tables**章节中的接口说明。
3. 调用**acl.himpi.venc_get_fd**将通道ID转换为一个文件句柄。
4. 调用**acl.himpi.sys_create_epoll**函数创建DVPP epoll实例。
5. 调用**acl.himpi.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。
6. 调用**acl.himpi.venc_start_chn**函数通知通道准备开始编码。
7. 调用**acl.himpi.dvpp_malloc**接口申请存放Device上输入数据的内存。
8. 启动一个用户态线程，调用**acl.himpi.sys_wait_epoll**函数等待编码完成。
9. 之后用户就可以调用**acl.himpi.venc_send_frame**函数发送待编码的码流。
10. 一旦编码完成，**acl.himpi.sys_wait_epoll**函数或select函数或poll函数就会返回，用户就可以调用**acl.himpi.venc_query_status**接口查询编码状态，再调用**acl.himpi.venc_get_stream**函数获取编码结果。
11. 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**acl.himpi.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。
12. 调用**acl.himpi.dvpp_free**接口释放输入内存。
13. 当用户不需发送图像到目的通道继续编码时，需要调用**acl.himpi.venc_stop_chn**函数通知该通道不再接收新的输入图片。
14. 调用**acl.himpi.sys_ctl_epoll**函数从epoll实例中删除编码通道的文件句柄。
15. 当用户完成所有编码之后，需要调用**acl.himpi.venc_destroy_chn**释放编码通道以及内部内存资源。
16. 调用**acl.himpi.sys_close_epoll**函数销毁DVPP epoll实例。
17. 调用**acl.himpi.sys_exit**接口进行媒体数据处理系统去初始化。

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）。
run_mode, ret = acl.rt.get_run_mode()

# 2.pyACL 初始化。
ret = acl.init()

# 3.运行管理资源申请(依次申请Device、Context)。
ret = acl.rt.set_device(0)
context, ret = acl.rt.create_context(0)

# 4.初始化媒体数据处理系统。
ret = acl.himpi.sys_init()

# 5.创建通道。
```

```
channel_id = 0
venc_attr = {'type': HI_PT_JPEG, 'profile': 0,
             'max_pic_width': 128, 'pic_width': 128,
             'max_pic_height': 128, 'pic_height': 128,
             'buf_size': 1024 * 1024 * 2, 'is_by_frame': 1}
jpeg_attr = {'recv_mode':HI_VENC_PIC_RECV_SINGLE,
            'mpf_cfg':{'large_thumbnail_num ': 0}}
attr = {'venc_attr':venc_attr, 'jpeg_attr':jpeg_attr}
ret = acl.himpi.venc_create_chn(channel_id, attr)

# 6.设置JPEGE参数。
param, ret = acl.himpi.venc_get_jpeg_param(channel_id)
param['qfactor'] = 100
ret = acl.himpi.venc_set_jpeg_param(channel_id, param)

# 7.通知编码器开始接收输入数据。
recv_param = {'recv_pic_num': -1}
ret = acl.himpi.venc_start_chn(channel_id, recv_param)

# 8.发送输入数据。
# 8.1 申请输入内存。
input_size = 128 * 128 * 3 // 2
input_addr, ret = acl.himpi.dvpp_malloc(0, input_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acl.rt.memcpy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。
# 直接将输入图片数据读入Device内存。
if run_mode == ACL_HOST:
    # 将输入图片读入内存中。
    jpege_file = np.fromfile(jpege_filee_path, dtype=np.byte)
    jpege_file_size = jpege_file.itemsize * jpege_file.size

    bytes_data = jpege_file.tobytes()
    jpege_file_ptr = acl.util.bytes_to_ptr(bytes_data)
    # 数据传输。
    ret = acl.rt.memcpy(input_addr, input_size, jpege_file_ptr, jpege_file_size,
ACL_MEMCPY_HOST_TO_DEVICE)
else:
    # 将输入图片读入内存中。
    jpege_file = np.fromfile(jpege_file_path, dtype=np.byte)
    jpege_file = jpege_file.itemsize * jpege_file.size

    bytes_data = jpege_file.tobytes()
    jpege_file_ptr = acl.util.bytes_to_ptr(bytes_data)
    # 数据传输。
    ret = acl.rt.memcpy(input_addr, input_size, jpege_file_ptr, jpege_file_size,
ACL_MEMCPY_DEVICE_TO_DEVICE)

# 8.2 发送输入数据，开始编码。
v_frame = {'width': 128,
           'height': 128,
           'field': HI_VIDEO_FIELD_FRAME,
           'pixel_format': HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420,
           'video_format': HI_VIDEO_FORMAT_LINEAR,
           'compress_mode': HI_COMPRESS_MODE_NONE,
           'dynamic_range': HI_DYNAMIC_RANGE_SDR8,
           'color_gamut': HI_COLOR_GAMUT_BT709,
           'header_stride': [0, 0, 0],
           'width_stride': [128, 0, 0],
           'height_stride': [0, 0, 0],
           'header_phys_addr': [0, 0, 0],
           'phys_addr': [0, 0, 0],
           'header_virt_addr': [0, 0, 0],
           'virt_addr': [input_addr, 0, 0],
           'time_ref': 0,'pts': cur_time}
frame = {'v_frame':v_frame, 'pool_id':0, 'mod_id':HI_ID_VGS}
ret = acl.himpi.venc_send_frame(channel_id, frame, 0)
```

```
# 9.获取编码结果。
# 9.1 通过EPOLL处理编码完成事件。
fd = acl.himpi.venc_get_fd(channel_id)
epoll_fd, ret = acl.himpi.sys_create_epoll(10)

event['data'] = fd
event['events'] = HI_DVPP_EPOLL_IN
ret = acl.himpi.sys_ctl_epoll(epoll_fd, HI_DVPP_EPOLL_CTL_ADD, fd, event)

# 编码完成前，会超时阻塞在这里，一旦完成，才会往下执行。
events, eventCount, ret = acl.himpi.sys_wait_epoll(epoll_fd, 3, 1000);

# 9.2 获取编码结果。
status, ret = acl.himpi.venc_query_status(channel_id)
stream = {'pack_cnt': status['cur_packs']}
stream, ret = acl.himpi.venc_get_stream(self.channel_id, stream, 1000)
# 9.3 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
acl.rt.memcpy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。
# 9.3 获取编码输出码流数据。
if run_mode == ACL_HOST:
    # 申请Host内存。
    output_buffer, ret= acl.rt.malloc_host(output_ize)
    # 数据传输。
    ret = acl.rt.memcpy(output_buffer, output_ize, stream['pack'][0]['addr'], output_ize,
ACL_MEMCPY_DEVICE_TO_HOST)
    # ......
    # 数据使用完成后，及时释放不使用的内存。
    ret = acl.rt.free_host(output_buffer)
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。
    # ......

# 10.释放输入内存和输出码流。
ret = acl.himpi.dvpp_free(input_addr)
ret = acl.himpi.venc_release_stream(channel_id, stream)

# 11.通知编码器停止接收输入数据。
ret = acl.himpi.venc_stop_chn(channel_id)
ret = acl.himpi.sys_ctl_epoll(epoll_fd, HI_DVPP_EPOLL_CTL_DEL, fd, event)
ret = acl.himpi.sys_close_epoll(epoll_fd)

# 12.销毁通道。
ret = acl.himpi.venc_destroy_chn(channel_id)

# 13. 媒体数据处理系统去初始化。
ret = acl.himpi.sys_exit()

# 14. 释放运行管理资源(依次释放Context、Device)。
ret = acl.rt.destroy_context(context)
ret = acl.rt.reset_device(0)

# 15.pyACL去初始化。
ret = acl.finalize()
```

5.4.5 PNGD 图片解码

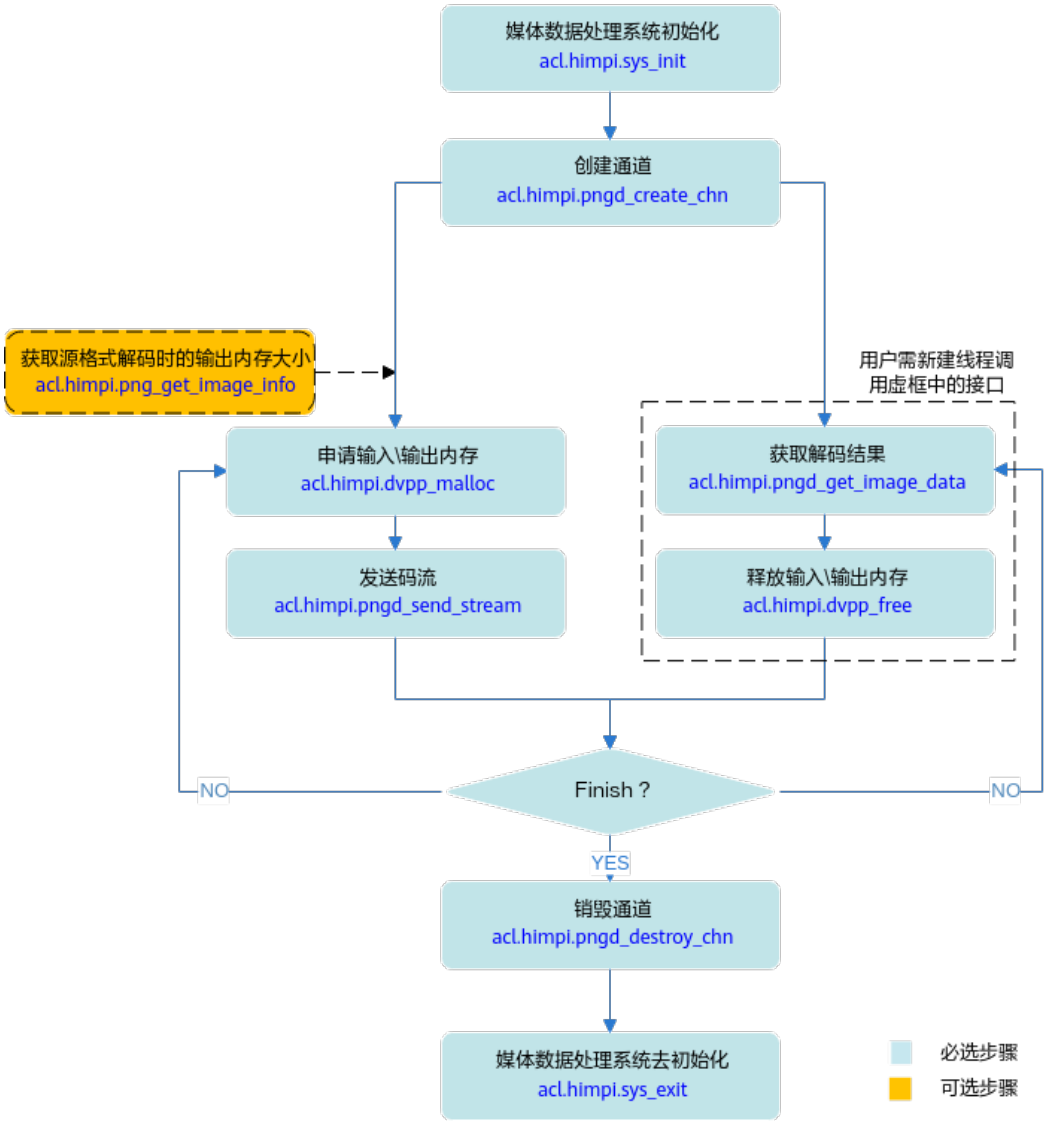
PNGD（PNG decoder）负责PNG格式图片的解码。关于PNGD功能的详细介绍请参见**10.8.7 PNGD图像处理接口**。

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。

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**3.4 pyACL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



当前系统支持解码PNG图片，关键接口的说明如下：

1. 调用**acl.himpi.sys_init**接口进行媒体数据处理系统初始化。
2. 调用**acl.himpi.pngd_create_chn**接口创建通道。
3. 调用**acl.himpi.dvpp_malloc**接口申请Device上的内存，存放输入或输出数据。
4. 调用**acl.himpi.pngd_send_stream**接口发送解码码流，**acl.himpi.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**acl.himpi.pngd_get_image_data**接口获取解码结果数据。
5. 调用**acl.himpi.dvpp_free**接口释放输入、输出内存。
6. 调用**acl.himpi.pngd_destroy_chn**接口销毁通道。
7. 调用**acl.himpi.sys_exit**接口进行媒体数据处理系统去初始化。

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）。
run_mode, ret = acl.rt.get_run_mode()

# 2.pyACL 初始化。
ret = acl.init()

# 3.运行管理资源申请(依次申请Device、Context)。
ret = acl.rt.set_device(0)
context, ret = acl.rt.create_context(0)

# 4.初始化媒体数据处理系统。
ret = acl.himpi.sys_init()

# 5.创建通道。
attr = {"stream_que_cnt": STREAM_QUE_CNT}
channel_id = 0
ret = acl.himpi.pngd_create_chn(channel_id, attr)

# 6.发送码流。
# 6.1 申请输入内存。
input_addr, ret = acl.himpi.dvpp_malloc(0, input_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acl.rt.memcpy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。
# 直接将输入图片数据读入Device内存。
if run_mode == ACL_HOST:
    # 将输入图片读入内存中。
    jpege_file = np.fromfile(jpege_filee_path, dtype=np.byte)
    jpege_file_size = jpege_file.itemsize * jpege_file.size

    bytes_data = jpege_file.tobytes()
    jpege_file_ptr = acl.util.bytes_to_ptr(bytes_data)
    # 数据传输。
    ret = acl.rt.memcpy(input_addr, input_size, jpege_file_ptr, jpege_file_size,
ACL_MEMCPY_HOST_TO_DEVICE)
else:
    # 将输入图片读入内存中。
    jpege_file = np.fromfile(jpege_file_path, dtype=np.byte)
    jpege_file = jpege_file.itemsize * jpege_file.size

    bytes_data = jpege_file.tobytes()
    jpege_file_ptr = acl.util.bytes_to_ptr(bytes_data)
    # 数据传输。
    ret = acl.rt.memcpy(input_addr, input_size, jpege_file_ptr, jpege_file_size,
ACL_MEMCPY_DEVICE_TO_DEVICE)

# 6.2 构造存放输入图片信息的字典。
stream = {"type": HI_PT_PNG, "addr": input_addr, "len": input_size, "pts": 0}
img_info, ret = acl.himpi.png_get_image_info(stream)

# 6.3 构造存放输出图片信息的字典，并申请输出内存。
output_addr, ret = acl.himpi.dvpp_malloc(0, img_info['img_buf_size']);
out_pic_info =  {"picture_address": output_addr,
                 "picture_buffer_size": img_info['img_buf_size'],
                 "picture_width": img_info['width'],
                 "picture_height": img_info['height'],
                 "picture_width_stride": img_info['width_stride'],
                 "picture_height_stride": img_info['height_stride'],
                 "picture_format": HI_PIXEL_FORMAT_UNKNOWN}
ret = acl.himpi.pngd_send_stream(channel_id, stream, out_pic_info, 0)

# 7.接收解码结果。
# 7.1 获取解码结果。
pic_info, get_stream, ret = acl.himpi.pngd_get_image_data(channel_id, -1)
if ret == 0: #  Decode success
    print("Chn %u GetFrame Success, Decode Success \n"%channel_id)
elif ret == HI_ERR_PNGD_BUF_EMPTY: # Decoding
    print("Chn %u Decoding, try again \n"%channel_id)
```

```
else: # Decode Fail
    print("Chn %u GetFrame Success, Decode Fail \n"%channel_id)

# 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
acl.rt.memcpy接口将Device的输出图片数据传输到Host。
# 7.2 获取PNGD的输出图片数据。
if run_mode == ACL_HOST:
    # 申请Host内存。
    output_buffer, ret= acl.rt.malloc_host(output_ize)
    # 数据传输。
    ret = acl.rt.memcpy(output_buffer, output_ize, stream['pack'][0]['addr'], output_ize,
ACL_MEMCPY_DEVICE_TO_HOST)
    # ......
    # 数据使用完成后，及时释放不使用的内存。
    ret = acl.rt.free_host(output_buffer)
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 释放输入内存和输出码流。
ret = acl.himpi.dvpp_free(input_addr)
ret = acl.himpi.dvpp_free(output_addr)

# 8.销毁通道。
ret = acl.himpi.pngd_destroy_chn(channel_id)

# 9. 媒体数据处理系统去初始化。
ret = acl.himpi.sys_exit()

# 10. 释放运行管理资源(依次释放Context、Device)。
ret = acl.rt.destroy_context(context)
ret = acl.rt.reset_device(0)

# 11.pyACL去初始化。
ret = acl.finalize()
```

5.4.6 VDEC 视频解码

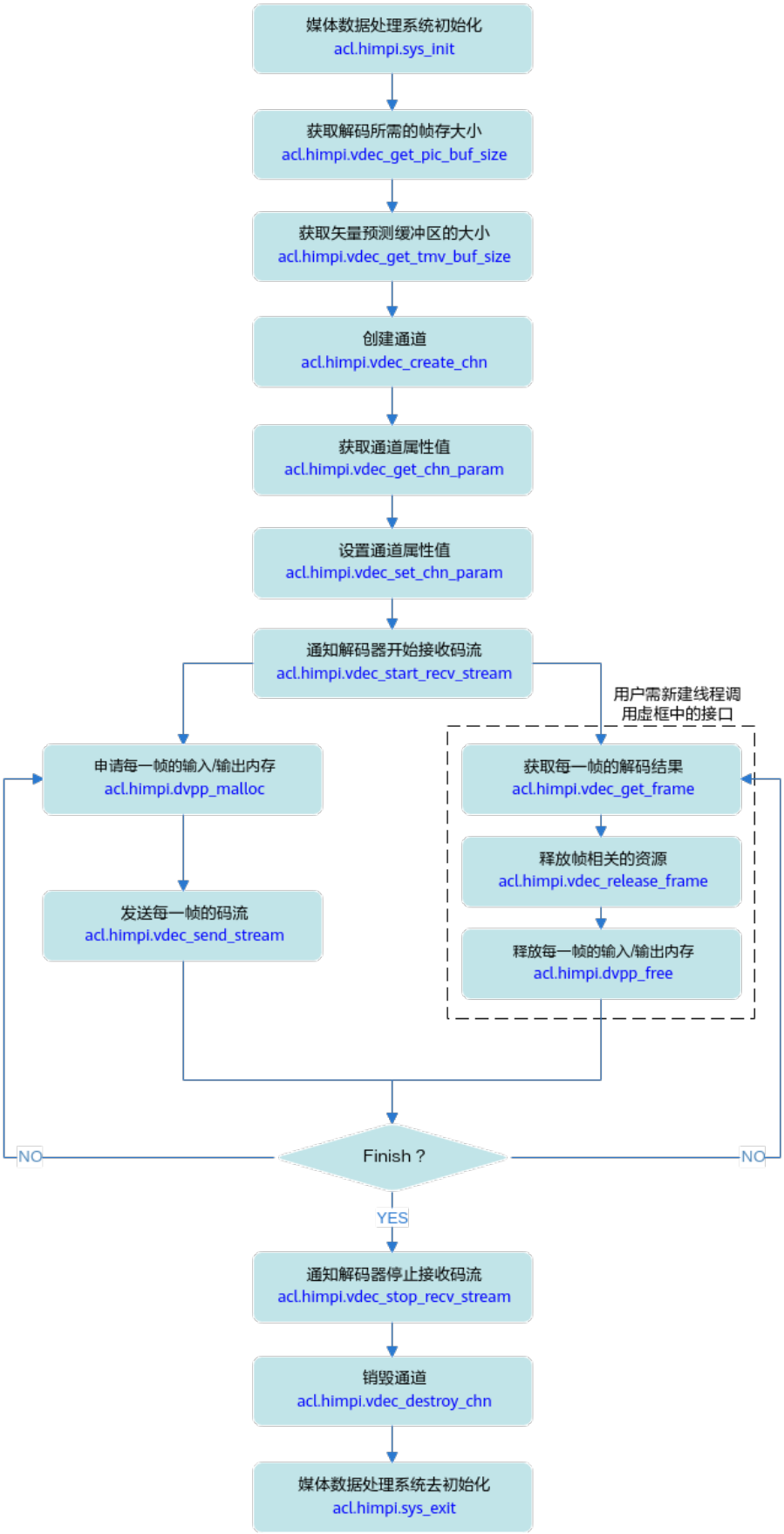
VDEC（Video Decoder）负责将H264/H265格式的视频码流解码为YUV/RGB格式的图片。关于VDEC功能的详细介绍及约束请参见**10.8.5.3 VDEC功能及约束说明**。

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。

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**3.4 pyACL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**acl.himpi.sys_init**接口进行媒体数据处理系统初始化。
2. 调用**acl.himpi.vdec_get_pic_buf_size**接口获取解码所需的帧存大小、调用**acl.himpi.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**acl.himpi.vdec_create_chn**接口创建通道。
4. 调用**acl.himpi.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**acl.himpi.vdec_start_recv_stream**接口通知解码器启动接收码流，再调用**acl.himpi.vdec_send_stream**接口发送解码码流，**acl.himpi.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**acl.himpi.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**acl.himpi.vdec_release_frame**接口释放帧相关的资源。

   解码结束后，需调用**acl.himpi.vdec_stop_recv_stream**接口通知解码器停止接收码流。
6. 调用**acl.himpi.dvpp_free**接口释放输入、输出内存。
7. 调用**acl.himpi.vdec_destroy_chn**接口销毁通道。
8. 调用**acl.himpi.sys_exit**接口进行媒体数据处理系统去初始化。

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）。
run_mode, ret = acl.rt.get_run_mode()

# 2.pyACL 初始化。
ret = acl.init()

# 3.运行管理资源申请(依次申请Device、Context)。
ret = acl.rt.set_device(0)
context, ret = acl.rt.create_context(0)

# 4.初始化媒体数据处理系统。
ret = acl.himpi.sys_init()

# 5.创建通道。
channel_id = 0
buf_attr = [1920, 1080, 0, 8, HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420, HI_COMPRESS_MODE_NONE]
tmv_buf_size = acl.himpi.vdec_get_tmv_buf_size(HI_PT_H264, 1920, 1080)
attr = {'type': HI_PT_H264, 'mode': HI_VDEC_SEND_MODE_FRAME,
        'pic_width': 1920, 'pic_height': 1080,
        'stream_buf_size': 1920 * 1080 * 3 // 2, 'frame_buf_size': 0,
        'frame_buf_cnt': 16, 'video_attr': {'ref_frame_num': 12, 'temporal_mvp_en': HI_TRUE,
                                            'tmv_buf_size': tmv_buf_size}}
ret = acl.himpi.vdec_create_chn(channel_id, attr)

# 6.设置通道属性。
video_param_dict, ret = acl.himpi.vdec_get_chn_param(channel_id)
video_param_dict["video_param"]["dec_mode "] = HI_VIDEO_DEC_MODE_IPB
video_param_dict["video_param"]["compress_mode "] = HI_COMPRESS_MODE_HFBC
video_param_dict["video_param"]["video_format "] = HI_VIDEO_FORMAT_TILE_64x16
video_param_dict["video_param"]["out_order"] = HI_VIDEO_OUT_ORDER_DISPLAY
video_param_dict["display_frame_num"] = 3
ret = acl.himpi.vdec_set_chn_param(i, video_param_dict)

# 7.解码器启动接收码流。
ret = acl.himpi.vdec_start_recv_stream(channel_id)
```

```
# 8.发送码流。
# 8.1 申请输入内存。
# input_size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。
input_size = 1024;
input_addr, ret = acl.himpi.dvpp_malloc(0, input_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acl.rt.memcpy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。
#直接将输入图片数据读入Device内存。
if run_mode == ACL_HOST:
    # 申请Host内存。
    input_buffer, ret= acl.rt.malloc_host(input_size)
    # 将输入图片读入内存中。
    vdec_file = np.fromfile(vdec_file_path, dtype=np.byte)
    vdec_file_size = vdec_file.itemsize * vdec_file.size

    bytes_data = vdec_file.tobytes()
    vdec_file_ptr = acl.util.bytes_to_ptr(bytes_data)
    # 数据传输。
    ret = acl.rt.memcpy(input_addr, input_size, vdec_file_ptr, vdec_file_size, ACL_MEMCPY_HOST_TO_DEVICE)
else:
    # 将输入图片读入内存中。
    vdec_file = np.fromfile(vdec_file_path, dtype=np.byte)
    vdec_file_size = vdec_file.itemsize * vdec_file.size

    bytes_data = vdec_file.tobytes()
    vdec_file_ptr = acl.util.bytes_to_ptr(bytes_data)
    # 数据传输。
    ret = acl.rt.memcpy(input_addr, input_size, vdec_file_ptr, vdec_file_size,
ACL_MEMCPY_DEVICE_TO_DEVICE)

# 8.2 申请输出内存。
output_size = 1920 * 1080 * 3 // 2
output_addr, ret = acl.himpi.dvpp_malloc(0, output_size);

# 8.3 构造存放一帧输入码流信息的字典。
stream = {'end_of_frame': HI_TRUE, 'end_of_stream': HI_FALSE,
          'need_display': HI_TRUE, 'pts': 0,
          'len': input_size, 'addr': input_addr}
# 8.4 构造存放一帧输出结果信息的字典。
out_pic_info = {"width": 1920,
                "height": 1080,
                "width_stride": 1920,
                "height_stride": 1080,
                "pixel_format": HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420,
                'vir_addr': output_addr,
                "buffer_size": output_size}

# 8.4 发送一帧码流。
ret = acl.himpi.vdec_send_stream(channel_id, stream, out_pic_info, 0)

# 9.接收解码结果。
# 9.1 获取解码结果。
frame_info, supplement, stream, ret = acl.himpi.vdec_get_frame(channel_id, 0)
if ret == 0:
    dec_result = frame_info['v_frame']['frame_flag']
    if dec_result == 0: #  0: Decode success
        print("Chn %u GetFrame Success, Decode Success \n"%channel_id)
    elif dec_result == 1: # 1:Decode fail
        print("Chn %u GetFrame Success, Decode Fail \n"%channel_id)
    elif dec_result == 2: # 2:This result is returned for the second field of
        print("Chn %u GetFrame Success, No Picture \n"%channel_id)
    elif dec_result == 3: # 3: Reference frame number set error
        print("Chn %u GetFrame Success, RefFrame Num Error \n"%channel_id)
    elif dec_result == 4: # 4: Reference frame size set error
        print("Chn %u GetFrame Success, RefFrame Size Error \n"% channel_id)
# 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acl.rt.memcpy接口将Device的输出图片数据传输到Host。
```

```
# 9.2 获取解码结果数据。
if run_mode == ACL_HOST:
    # 申请Host内存。
    output_buffer, ret= acl.rt.malloc_host(out_pic_info['buffer_size'])
    # 数据传输。
    ret = acl.rt.memcpy(output_buffer, out_pic_info['buffer_size'], frame_info['v_frame']['virt_addr'][0],
out_pic_info['buffer_size'], ACL_MEMCPY_DEVICE_TO_HOST)
    # ......
    # 数据使用完成后，及时释放不使用的内存。
    ret = acl.rt.free_host(output_buffer)
else:
    # 可以直接使用JPEGD的输出图片数据，在outputPic.picture_address指向的内存中。
    # ......

# 9.3 释放输入、输出内存。
ret = acl.himpi.dvpp_free(frame_info['v_frame']['virt_addr'][0])
ret = acl.himpi.dvpp_free(stream['addr'])

# 9.4 释放资源。
ret = acl.himpi.vdec_release_frame(channel_id, frame_info)

# 10.解码器停止接收码流。
ret = acl.himpi.vdec_stop_recv_stream(channel_id)

# 7.销毁通道。
ret = acl.himpi.vdec_destroy_chn(channel_id)

# 8. 媒体数据处理系统去初始化。
ret = acl.himpi.sys_exit()

# 9. 释放运行管理资源(依次释放Context、Device)。
ret = acl.rt.destroy_context(context)
ret = acl.rt.reset_device(0)

# 10.pyACL去初始化。
ret = acl.finalize()
```

5.4.7 VENC 视频编码

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功能的详细介绍请参见**10.8.6.3 VENC功能及约束说明**。

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。

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**3.4 pyACL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 接口调用流程

媒体数据处理系统初始化
acl.himpi.sys_init
创建通道
acl.himpi.venc_create_chn
设置编码高级参数
将通道ID转换为一个文件句柄
acl.himpi.venc_get_fd
创建epoll实例
acl.himpi.sys_create_epoll
将文件句柄添加到epoll实例
acl.himpi.sys_ctl_epoll
通知编码通道准备开始编码
acl.himpi.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
acl.himpi.dvpp_malloc
发送每一帧的码流数据
acl.himpi.venc_send_frame
等待编码完成
acl.himpi.sys_wait_epoll
查询编码状态
acl.himpi.venc_query_status
获取编码结果
acl.himpi.venc_get_stream
释放码流缓存
acl.himpi.venc_release_stream
释放每一帧的输入内存
acl.himpi.dvpp_free
Finish VENC?
NO
NO
YES
通知编码通道停止编码
acl.himpi.venc_stop_chn
从epoll实例中删除文件句柄
acl.himpi.sys_ctl_epoll
销毁通道
acl.himpi.venc_destroy_chn
销毁epoll实例
acl.himpi.sys_close_epoll
媒体数据处理系统去初始化
acl.himpi.sys_exit
必选步骤
可选步骤

当前系统支持H264/H256格式的视频码流，关键接口的说明如下：

1. 调用**acl.himpi.sys_init**接口进行媒体数据处理系统初始化。
2. 调用**acl.himpi.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**acl.himpi.venc_set_jpeg_param**~**acl.himpi.venc_compact_jpeg_tables**章节中的接口说明。
3. 调用**acl.himpi.venc_get_fd**将通道ID转换为一个文件句柄。
4. 调用**acl.himpi.sys_create_epoll**函数创建DVPP epoll实例。
5. 调用**acl.himpi.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。
6. 调用**acl.himpi.venc_start_chn**函数通知通道准备开始编码。
7. 调用**acl.himpi.dvpp_malloc**接口申请存放Device上输入数据的内存。
8. 启动一个用户态线程，调用**acl.himpi.sys_wait_epoll**函数等待编码完成。
9. 之后用户就可以调用**acl.himpi.venc_send_frame**函数发送待编码的码流。
10. 一旦编码完成，**acl.himpi.sys_wait_epoll**函数或select函数或poll函数就会返回，用户就可以调用**acl.himpi.venc_query_status**接口查询编码状态，再调用**acl.himpi.venc_get_stream**函数获取编码结果。
11. 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**acl.himpi.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。
12. 调用**acl.himpi.dvpp_free**接口释放输入内存。
13. 当用户不需发送图像到目的通道继续编码时，需要调用**acl.himpi.venc_stop_chn**函数通知该通道不再接收新的输入图片。
14. 调用**acl.himpi.sys_ctl_epoll**函数从epoll实例中删除编码通道的文件句柄。
15. 当用户完成所有编码之后，需要调用**acl.himpi.venc_destroy_chn**释放编码通道以及内部内存资源。
16. 调用**acl.himpi.sys_close_epoll**函数销毁DVPP epoll实例。
17. 调用**acl.himpi.sys_exit**接口进行媒体数据处理系统去初始化。

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**acl.himpi.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，通过通过**hi_venc_chn_attr**["venc_attr"]字典内的“profile”属性来确定。
- 帧率，通过**hi_venc_chn_attr**["rc_attr"]字典内的“src_frame_rate”输入帧率参数、“dst_frame_rate”输出帧率属性来确定。
- GOP，通过**hi_venc_chn_attr**["rc_attr"]字典内的“gop”属性来确定。
- 码率，通过**hi_venc_chn_attr**["rc_attr"]字典内的“bit_rate”、“max_bit_rate”或“target_bit_rate”属性来确定。

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	50或60	● 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	50或60	● 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。
1080P（蓝光） 1920*1080	25或30	50或60	● 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。

画质/分辨率	帧率	GOP	码率（mbps）
720P（高清） 1280*720	25或30	50或60	● 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	50或60	● 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	50或60	● 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。
270P（流畅） 480*270	25或30	50或60	● 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	50或60	● 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	acl.himpi.venc_create_chn	hi_venc_chn_attr["rc_attr"]字典内的“rc_mode”属性	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；
码率控制模型统计时间	acl.himpi.venc_create_chn	hi_venc_chn_attr["rc_attr"]字典内各模式属性值结构体内的“stats_time”属性	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。

配置项	接口	参数名	说明
宏块级码率控制参数	acl.himpi.venc_set_rc_param	hi_venc_rc_param字典内的“threshold_i”、“threshold_p”、“threshold_b”、“direction”、“row_qp_delta”属性。	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	acl.himpi.venc_create_chn	hi_venc_rc_param字典内的“first_frame_start_qp”属性	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”属性，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”配置为“35”，同时将“max_reencode_times”配置为“0”，会获得较好的编码质量。
编码场景模式	acl.himpi.venc_set_scene_mode	hi_venc_scene_mode类	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

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）。
run_mode, ret = acl.rt.get_run_mode()
```

```
# 2.pyACL 初始化。
ret = acl.init()

# 3.运行管理资源申请(依次申请Device、Context)。
ret = acl.rt.set_device(0)
context, ret = acl.rt.create_context(0)

# 4.初始化媒体数据处理系统。
ret = acl.himpi.sys_init()

# 5.设置VENC模块参数。
param = {'mod_type':HI_VENC_MOD_H265}
param, ret = acl.himpi.venc_get_mod_param(param)
param['jpeg_mod_param']['one_stream_buf'] = 1
ret = acl.himpi.venc_set_mod_param(param)

# 6.创建通道。
channel_id = 0
venc_attr = {'type': HI_VENC_MOD_H265, 'profile': 0,
             'max_pic_width': 128, 'pic_width': 128,
             'max_pic_height': 128, 'pic_height': 128,
             'buf_size': 1024 * 1024 * 2, 'is_by_frame': 1}
rc_attr = {'rc_mode':HI_VENC_RC_MODE_H265_VBR,
           'h265_vbr':{'gop': 30, 'stats_time': 1,
                       'src_frame_rate': 30, 'dst_frame_rate': 30,
                       'max_bit_rate': 4000}}
gop_attr = {'gop_mode':0, 'normal_p':{'ip_qp_delta':3}}
attr = {'venc_attr':venc_attr, 'rc_attr':rc_attr, 'gop_attr':gop_attr}
ret = acl.himpi.venc_create_chn(channel_id, attr)

# 7.通知编码器开始接收输入数据。
recv_param = {'recv_pic_num':-1}
ret = acl.himpi.venc_start_chn(channel_id, recv_param)

# 8.发送输入数据。
# 8.1 申请输入内存。
input_size = 128 * 128 * 3 // 2
input_addr, ret = acl.himpi.dvpp_malloc(0, input_size);

# 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acl.rt.memcpy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。
# 直接将输入数据读入Device内存。
if run_mode == ACL_HOST:
    # 将输入图片读入内存中。
    jpege_file = np.fromfile(jpege_filee_path, dtype=np.byte)
    jpege_file_size = jpege_file.itemsize * jpege_file.size

    bytes_data = jpege_file.tobytes()
    jpege_file_ptr = acl.util.bytes_to_ptr(bytes_data)
    # 数据传输。
    ret = acl.rt.memcpy(input_addr, input_size, jpege_file_ptr, jpege_file_size,
ACL_MEMCPY_HOST_TO_DEVICE)
else:
    # 将输入图片读入内存中。
    jpege_file = np.fromfile(jpege_file_path, dtype=np.byte)
    jpege_file = jpege_file.itemsize * jpege_file.size

    bytes_data = jpege_file.tobytes()
    jpege_file_ptr = acl.util.bytes_to_ptr(bytes_data)
    # 数据传输。
    ret = acl.rt.memcpy(input_addr, input_size, jpege_file_ptr, jpege_file_size,
ACL_MEMCPY_DEVICE_TO_DEVICE)

# 8.2 发送输入数据，开始编码。
v_frame = {'width': 128,
           'height': 128,
           'field': HI_VIDEO_FIELD_FRAME,
           'pixel_format': HI_PIXEL_FORMAT_YUV_SEMIPLANAR_420,
```

```
            'video_format': HI_VIDEO_FORMAT_LINEAR,
            'compress_mode': HI_COMPRESS_MODE_NONE,
            'dynamic_range': HI_DYNAMIC_RANGE_SDR8,
            'color_gamut': HI_COLOR_GAMUT_BT709,
            'header_stride': [0, 0, 0],
            'width_stride': [128, 0, 0],
            'height_stride': [0, 0, 0],
            'header_phys_addr': [0, 0, 0],
            'phys_addr': [0, 0, 0],
            'header_virt_addr': [0, 0, 0],
            'virt_addr': [input_addr, 0, 0],
            'time_ref': 0,'pts': cur_time}
frame = {'v_frame':v_frame, 'pool_id':0, 'mod_id':HI_ID_VENC}
ret = acl.himpi.venc_send_frame(channel_id, frame, 0)

# 9.获取编码结果。
# 9.1 通过EPOLL处理编码完成事件。
fd = acl.himpi.venc_get_fd(channel_id)
epoll_fd, ret = acl.himpi.sys_create_epoll(10)

event['data'] = fd
event['events'] = HI_DVPP_EPOLL_IN
ret = acl.himpi.sys_ctl_epoll(epoll_fd, HI_DVPP_EPOLL_CTL_ADD, fd, event)

# 编码完成前，会超时阻塞在这里，一旦完成，才会往下执行。
events, eventCount, ret = acl.himpi.sys_wait_epoll(epoll_fd, 3, 1000);

# 9.2 获取编码结果。
status, ret = acl.himpi.venc_query_status(channel_id)
stream = {'pack_cnt': status['cur_packs']}
stream, ret = acl.himpi.venc_get_stream(self.channel_id, stream, 1000)
# 9.3 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
acl.rt.memcpy接口将Device的输出码流传输到Host。
# 9.3 获取编码输出码流数据。
if run_mode == ACL_HOST:
    # 申请Host内存。
    output_buffer, ret= acl.rt.malloc_host(output_ize)
    # 数据传输。
    ret = acl.rt.memcpy(output_buffer, output_ize, stream['pack'][0]['addr'], output_ize,
ACL_MEMCPY_DEVICE_TO_HOST)
    # ......
    # 数据使用完成后，及时释放不使用的内存。
    ret = acl.rt.free_host(output_buffer)
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。
    # ......

# 10.释放输入内存和输出码流。
ret = acl.himpi.dvpp_free(input_addr)
ret = acl.himpi.venc_release_stream(channel_id, stream)

# 11.通知编码器停止接收输入数据。
ret = acl.himpi.venc_stop_chn(channel_id)
ret = acl.himpi.sys_ctl_epoll(epoll_fd, HI_DVPP_EPOLL_CTL_DEL, fd, event)
ret = acl.himpi.sys_close_epoll(epoll_fd)

# 12.销毁通道。
ret = acl.himpi.venc_destroy_chn(channel_id)

# 13. 媒体数据处理系统去初始化。
ret = acl.himpi.sys_exit()

# 14. 释放运行管理资源(依次释放Context、Device)。
ret = acl.rt.destroy_context(context)
ret = acl.rt.reset_device(0)

# 15.pyACL去初始化。
ret = acl.finalize()
```

6 单算子调用

简介
开发流程
接口调用流程
调用CBLAS接口
固定Shape算子
动态Shape算子（不注册算子选择器）
动态Shape算子（注册算子选择器）

6.1 简介

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pyACL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。

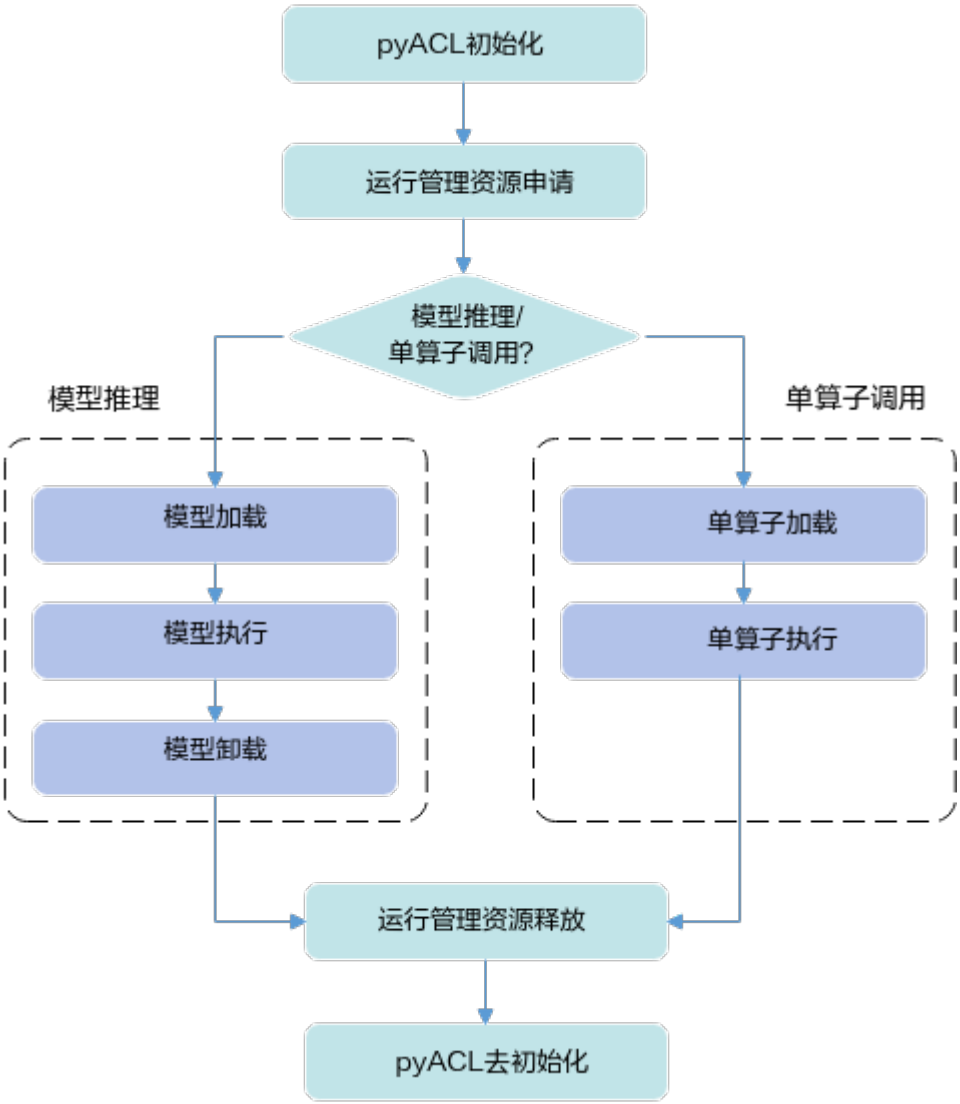
另外，自定义的算子，也可以通过单算子调用的方式来验证算子的功能。

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，找出基本的共同点、不同点。

- 共同点：
  - 需要pyACL初始化和去初始化、运行管理资源申请和释放。
  - 涉及加载、执行的步骤，但是要注意，两者的加载、执行是调用不同的pyACL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。

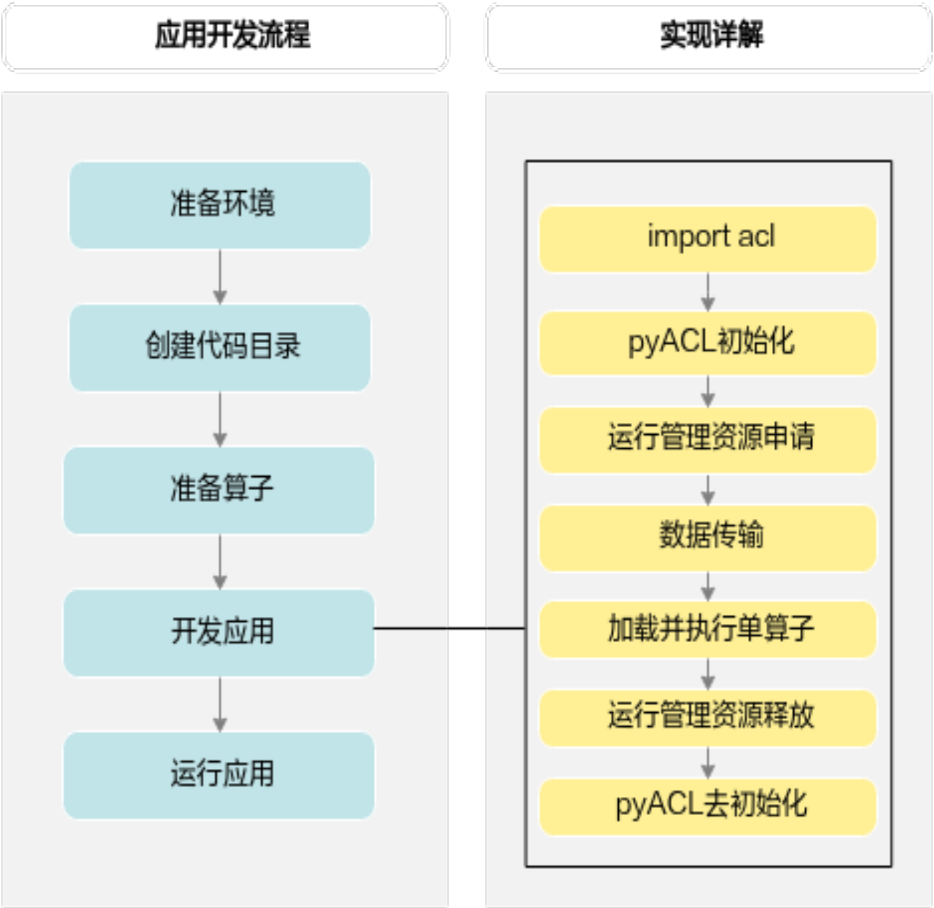
图 6-1 单算子调用与模型推理的流程对比



版权所有 © 华为技术有限公司

6.2 开发流程

图 6-2 开发流程



1. 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、脚本、测试图片数据、模型文件等。

2. 准备算子。

   准备算子时，有以下方式（参见**6.3 接口调用流程**中的说明）：

   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pyACL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**》。

3. 开发应用。

   单算子调用的流程请参见**6.3 接口调用流程**及相关的示例代码。

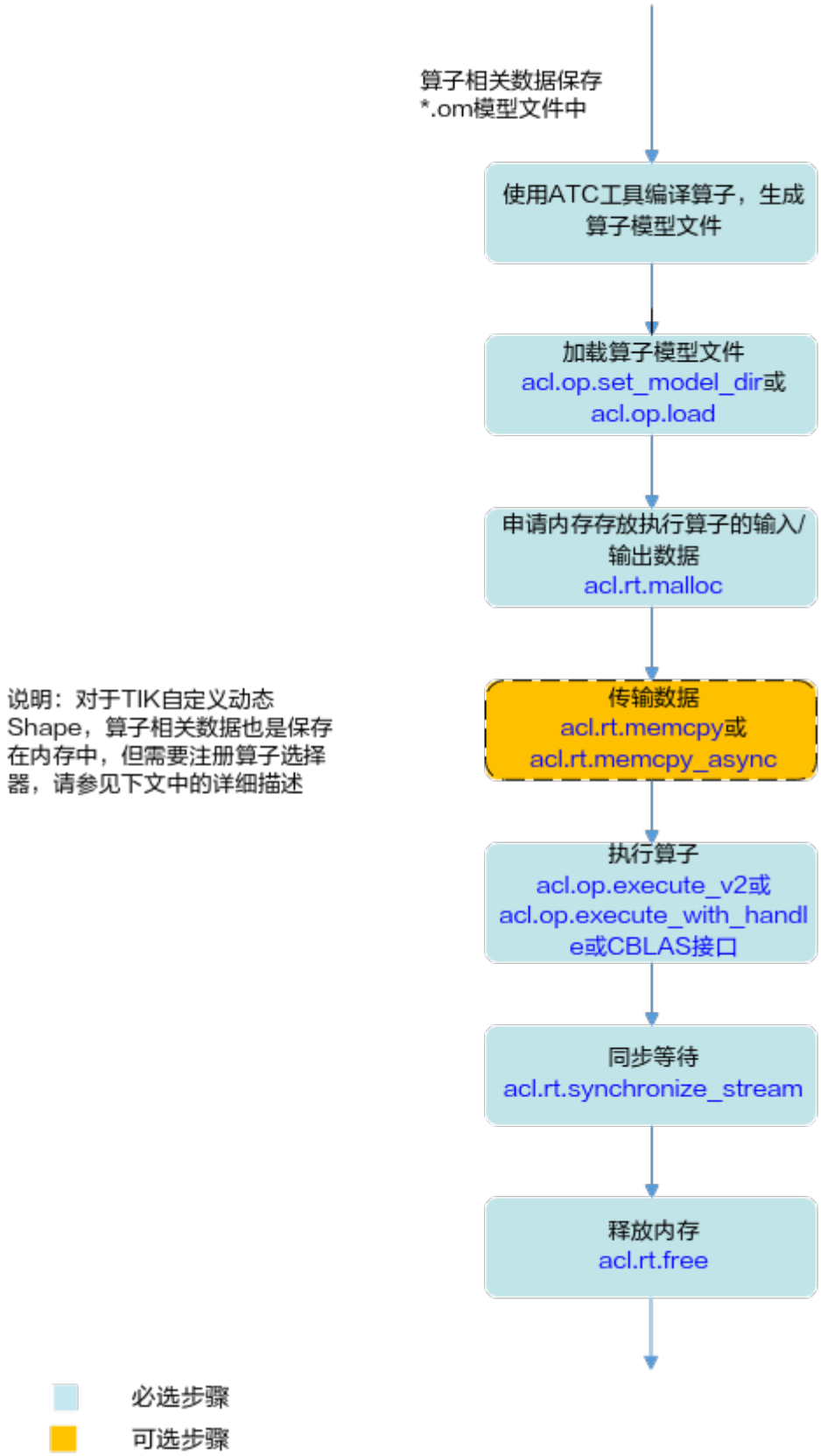
4. 运行应用，请参见**8 应用调试**。

6.3 接口调用流程

开发应用时，如果涉及执行单个算子，则应用程序中必须包含执行单个算子的代码逻辑。关于执行单个算子的接口调用流程，请先参见**3.4 pyACL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**》。

图 6-3 算子调用流程


算子相关数据保存
*.om模型文件中
使用ATC工具编译算子，生成
算子模型文件
加载算子模型文件
acl.op.set_model_dir或
acl.op.load
申请内存存放执行算子的输入/
输出数据
acl.rt.malloc
说明：对于TIK自定义动态
Shape，算子相关数据也是保存
在内存中，但需要注册算子选择
器，请参见下文中的详细描述
传输数据
acl.rt.memcpy或
acl.rt.memcpy_async
执行算子
acl.op.execute_v2或
acl.op.execute_with_handl
e或CBLAS接口
同步等待
acl.rt.synchronize_stream
释放内存
acl.rt.free
必选步骤
可选步骤

关键接口的说明如下：

1. 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**acl.op.set_model_dir**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**acl.op.load**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。
2. 调用**acl.rt.malloc**接口申请Device上的内存，存放执行算子的输入、输出数据。
3. 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**acl.op.infer_shape**接口、**acl.get_tensor_desc_num_dims**接口、**acl.get_tensor_desc_dim_v2**接口、**acl.get_tensor_desc_dim_range**等接口，推导或预估算子的输出Shape，作为算子执行接口**acl.op.execute_v2**的输入。
4. 执行算子。

   - 对于被封装成pyACL接口的算子（参见**6.4 调用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**acl.blas.gemm_ex**接口（封装GEMM算子）、**acl.op.cast**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**acl.blas.create_handle_for_gemm_ex**接口、**acl.op.create_handle_for_cast**接口等创建handle后，还需要调用**acl.op.execute_with_handle**接口执行算子。
   - 对于未被封装成pyACL接口的算子，目前执行以下两种执行方式：
     - 不以handle方式执行算子，调用**acl.op.execute_v2**接口执行算子。
     - 以handle方式执行算子，调用**acl.op.create_handle**接口创建handle，再调用**acl.op.execute_with_handle**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Handle使用结束后，需调用**acl.op.destroy_handle**接口释放。
5. 调用**acl.rt.synchronize_stream**接口阻塞应用运行，直到指定Stream中的所有任务都完成。
6. 调用**acl.rt.free**接口释放内存。

6.4 调用 CBLAS 接口

基本原理

目前，pyACL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pyACL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**acl.blas.gemm_ex**接口（封装GEMM算子）、**acl.op.cast**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**acl.blas.create_handle_for_gemm_ex**接口、**acl.op.create_handle_for_cast**接口等创建handle后，还需要调用**acl.op.execute_with_handle**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Handle使用结束后，需调用**acl.op.destroy_handle**接口释放。

示例代码

本章以**acl.blas.gemm_ex**接口为例，该示例中矩阵乘的计算公式为：C = αAB + βC。

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，不可以直接拷贝运行，仅供参考。

```
import acl
# ......

ACL_MEM_MALLOC_NORMAL_ONLY = 2
ACL_TRANS_N = 0
ACL_COMPUTE_HIGH_PRECISION = 0
ACL_MEMCPY_HOST_TO_DEVICE = 1
ACL_MEMCPY_DEVICE_TO_HOST = 2

# 1.pyACL初始化。
ret = acl.init("test_data/config/acl.json")

# 2.设置单算子模型文件所在的目录。
ret = acl.op.set_model_dir("op_models")

# 3.指定用于运算的设备。
device_id = 0
ret = acl.rt.set_device(device_id)

# 4.申请内存。
# 4.1 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，size_a表示矩阵A数据的大小，size_b表示矩阵B数据的大小，size_c表示矩阵C数据的大小。
in_dtype, out_dtype = 1, 1
size_a = m * k * acl.data_type_size(acl_dtype)
size_b = m * k * acl.data_type_size(acl_dtype)
size_c = m * k * acl.data_type_size(acl_dtype)
dev_matrix_a, ret = acl.rt.malloc(size_a, ACL_MEM_MALLOC_NORMAL_ONLY)
dev_matrix_b, ret = acl.rt.malloc(size_b, ACL_MEM_MALLOC_NORMAL_ONLY)
dev_matrix_c, ret = acl.rt.malloc(size_c, ACL_MEM_MALLOC_NORMAL_ONLY)
# 4.2 申请Host上的内存。
# 对于该矩阵乘示例，m表示矩阵A的行数与矩阵C的行数，n表示矩阵B的列数与矩阵C的列数。
# k表示矩阵A的列数与矩阵B的行数。
host_matrix_a, ret = acl.rt.malloc_host(size_a)
host_matrix_b, ret = acl.rt.malloc_host(size_b)
host_matrix_c, ret = acl.rt.malloc_host(size_c)

# 5.准备输入数据。
# 从文件读入到host_matrix_a和host_matrix_b中。
# 对于该矩阵乘示例，将矩阵A和矩阵B的数据从Host复制到Device。
ret = acl.rt.memcpy(dev_matrix_a, size_a, host_matrix_a, size_a, ACL_MEMCPY_HOST_TO_DEVICE)
ret = acl.rt.memcpy(dev_matrix_b, size_b, host_matrix_b, size_b, ACL_MEMCPY_HOST_TO_DEVICE)
```

```
# 6.执行单算子。
stream, ret = acl.rt.create_stream()
# 对于该示例，调用acl.blas.gemm_ex接口（异步接口）实现矩阵-矩阵的乘法。
ret = acl.blas.gemm_ex(ACL_TRANS_N, ACL_TRANS_N, ACL_TRANS_N, m, n, k, dev_alpha,
                       dev_matrix_a, k, input_type, dev_matrix_b, n, input_type, dev_beta,
                       dev_matrix_c, n, output_type, ACL_COMPUTE_HIGH_PRECISION, stream)
# 调用acl.rt.synchronize_stream接口阻塞Host运行，直到指定Stream中的所有任务都完成。
ret = acl.rt.synchronize_stream(stream)

# 7.将算子的输出数据从Device复制到Host。
ret = acl.rt.memcpy(host_matrix_c, size_c, dev_matrix_c, size_c, ACL_MEMCPY_DEVICE_TO_HOST)

# 8.释放运行管理资源。
ret = acl.rt.destroy_stream(stream)
ret = acl.rt.reset_device(device_id)
ret = acl.finalize()
# ......
```

6.5 固定 Shape 算子

对于昇腾AI处理器不支持的算子，用户需参考《**TBE&AI CPU算子开发指南**》先完成自定义算子的开发，再参考如下内容执行单算子。

6.6 动态 Shape 算子（不注册算子选择器）

基本原理

对于支持动态Shape的算子：

- 如果算子输出Shape明确，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**6.3 接口调用流程**，执行固定Shape算子的示例代码请参见**6.5 固定Shape算子**。
- 如果无法明确算子的输出Shape，在调用**acl.op.execute_v2**接口前，需用户调用**acl.op.infer_shape**接口、**acl.get_tensor_desc_num_dims**接口、**acl.get_tensor_desc_dim_v2**接口、**10.3.19 函数：get_tensor_desc_dim_range**等接口，推导或预估算子的输出Shape，作为算子执行接口**acl.op.execute_v2**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，不可以直接拷贝运行，仅供参考。

```
import acl
# ......

# 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。
ret = acl.op.infer_shape(op_type, self.in_desc_list, self.in_list,
                         num_outputs, self.out_desc_list, self.attr)

tensor_dims = []
# 循环算子的每一个输出，推导或预估Shape值：
for i in range(len(infer_desc_list)):
    dim_nums = acl.get_tensor_desc_num_dims(infer_desc_list[i])
    dim_size = []
    for j in range(dim_nums):
```

```
        dim, ret = acl.get_tensor_desc_dim_v2(infer_desc_list[i], j)
        # 表示动态Shape场景下维度值是动态的。
        if dim == -1:
            # 获取Shape范围，使用该范围的Shape最大值来构造输出tensor_desc，作为acl.op.execute_v2的输入。
            dim_range, ret = acl.get_tensor_desc_dim_range(infer_desc_list[i], j, 2)
            dim = dim_range[1]
        dim_size.append(dim)
    tensor_dims.append(dim_size)

# 以上给出了执行算子时输出的shape，根据tensor_dims中的dims构造输出tensor_desc（即self.out_desc_list参数值），用于调用acl.op.execute_v2。
ret = acl.op.execute_v2(op_type, self.in_desc_list, self.in_list, self.out_desc_list,
                        self.out_list, self.attr, self.stream)

out_tensor_dims = []
# 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，设在算子执行结束后，需增加下面的调用，获取准确的shape。
for i in range(len(self.out_desc_list)):
    dim_nums = acl.get_tensor_desc_num_dims(self.out_desc_list[i])
    dim_size = []
    for j in range(dim_nums):
        dim, ret = acl.get_tensor_desc_dim_v2(self.out_desc_list[i], j)
        dim_size.append(dim)
    out_tensor_dims.append(dim_size)

# ......
```

6.7 动态 Shape 算子（注册算子选择器）

基本原理

动态Shape场景下，算子加载与执行的流程如下：

1. 资源初始化，包括pyACL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acl.init接口实现pyACL初始化。
   - 调用pyACL接口注册要编译的自定义算子：
     - 调用acl.op.register_compile_func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：

```
op_selector(in_num, in_desc, out_num, out_desc, op_attr, op_kernel_desc)
    """
    算子选择器：函数和参数名称可自定义，参数个数和类型必须匹配。
    :param in_num: 输入tensor描述数。
    :param in_desc: 输入tensor描述list。
    :param out_num: 输出tensor描述数。
    :param out_desc: 输出tensor描述list。
    :param op_attr: 算子属性的地址对象，用于设置算子属性信息。
    :param op_kernel_desc:算子内核描述地址对象，用于动态Shape场景下，设置算子Workspace参数。
    :return:
    """
```

       - 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，并将调用acl.op.set_kernel_args接口，设置算子Tiling参数、执行并发数等。

    - 调用**acl.op.create_kernel**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**acl.rt.set_device**接口指定运算的Device。
  - 调用**acl.rt.create_context**接口显式创建一个Context，调用**acl.rt.create_stream**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**acl.rt.set_device**接口时隐式创建的，默认Stream作为接口入参时，直接传0。
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. 将算子输入数据从Host复制到Device上。
  - 调用**acl.rt.memcpy**接口实现同步内存复制，内存使用结束后需及时释放。
  - 调用**acl.rt.memcpy_async**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**acl.op.update_params**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**acl.op.execute_v2**接口加载并执行算子。
6. 将算子运算的输出数据从Device上复制到Host上（提前申请Host上的内存）。
  - 调用**acl.rt.memcpy**接口实现同步内存复制，内存使用结束后需及时释放。
  - 调用**acl.rt.memcpy_async**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**acl.rt.destroy_stream**接口释放Stream。

    不涉及显式创建Stream，使用默认Stream时，无需调用**acl.rt.destroy_stream**接口释放Stream。
  - 调用**acl.rt.destroy_context**接口释放Context。

    不涉及显式创建Context，使用默认Context时，无需调用**acl.rt.destroy_context**接口释放Context。
  - 调用**acl.rt.reset_device**接口释放Device。
8. 调用**acl.finalize**接口实现pyACL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。

```
import acl
import numpy as np
# ......

ACL_ENGINE_AICORE = 1
ACL_FLOAT16 = 1
ACL_FORMAT_ND = 2
ACL_MEM_MALLOC_NORMAL_ONLY = 2
ACL_MEMCPY_HOST_TO_DEVICE = 1
ACL_MEMCPY_DEVICE_TO_HOST= 2
```

```
device_id = 0

# 1.资源初始化。
ret = acl.init()
ret = acl.rt.set_device(device_id)
stream, ret = acl.rt.create_stream()
ret = acl.op.register_compile_func("add", op_select)
# 将算子Kernel的*.o文件需要用户提前编译好，并调用numpy加载.o文件并转成地址对象，op_data_size_0表示
第一个.o文件占用的内存大小。
# 如果有多个算子Kernel的*.o文件，需要多次调用该接口。
ret = acl.op.create_kernel("add", "cce_add_11_33_float16_11_33_float16__kernel0",
                           "cce_add_11_33_float16_11_33_float16__kernel0",
                           np_op_0_ptr, op_data_size_0, ACL_ENGINE_AICORE, 0)

# 2.构造add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。
# 输入参数。
a = np.random.rand(2, 1).astype(np.float16)
b = np.random.rand(2, 1).astype(np.float16)
bytes_data = a.tobytes()
a_ptr = acl.util.bytes_to_ptr(bytes_data)
bytes_data = b.tobytes()
b_ptr = acl.util.bytes_to_ptr(bytes_data)
input_desc_list = [acl.create_tensor_desc(ACL_FLOAT16, [2, 1], ACL_FORMAT_ND),
                   acl.create_tensor_desc(ACL_FLOAT16, [2, 1], ACL_FORMAT_ND)]
output_desc_list = [acl.create_tensor_desc(ACL_FLOAT16, [2, 1], ACL_FORMAT_ND)]
# 申请device侧内存。
size_a = acl.get_tensor_desc_size(input_desc_list[0])
size_b = acl.get_tensor_desc_size(input_desc_list[1])
size_c = acl.get_tensor_desc_size(output_desc_list[0])
dev_a, ret = acl.rt.malloc(size_a, ACL_MEM_MALLOC_NORMAL_ONLY)
dev_b, ret = acl.rt.malloc(size_b, ACL_MEM_MALLOC_NORMAL_ONLY)
dev_c, ret = acl.rt.malloc(size_c, ACL_MEM_MALLOC_NORMAL_ONLY)

# 3.将算子输入数据从Host复制到Device上。
ret = acl.rt.memcpy(dev_a, size_a, a_ptr, size_a, ACL_MEMCPY_HOST_TO_DEVICE)
ret = acl.rt.memcpy(dev_b, size_b, b_ptr, size_b, ACL_MEMCPY_HOST_TO_DEVICE)

# 4.调用acl.op.update_params接口编译算子。
op_attr = acl.op.create_attr()
ret = acl.op.update_params("add", input_desc_list, output_desc_list, op_attr )

# 5.调用acl.op.execute接口加载并执行算子。
in_data_list = [acl.create_data_buffer(dev_a, size_a),
                acl.create_data_buffer(dev_b, size_b)]
out_data_list = [acl.create_data_buffer(dev_c, size_c)]
ret = acl.op.execute_v2("add", input_desc_list, in_data_list, output_desc_list,
                        out_data_list, op_attr, stream)

# 6.将算子运算的输出数据从Device上复制到Host上（提前申请Host上的内存）。
host_ptr, ret = acl.rt.malloc_host(size_c)
ret = acl.rt.memcpy(host_ptr, size_c, dev_c, size_c, ACL_MEMCPY_DEVICE_TO_HOST)
bytes_out = acl.util.ptr_to_bytes(host_ptr, size_c)
out_np = np.frombuffer(bytes_out, dtype=np.byte).reshape((size_c,))

# 7.按顺序释放资源。
# 7.1 释放算子的输入输出Tensor描述。
# 7.2 释放Host上的内存。
# 7.3 释放Device上的内存。
# 7.4 释放设备管理资源。
ret = acl.rt.destroy_stream(stream)
ret = acl.rt.reset_device(device_id)
ret = acl.finalize()
# ......
```

7 扩展更多特性

内存二次分配管理
多Device场景
多模型串联推理
多Batch模型推理
模型异步推理
模型动态推理
模型动态AIPP推理
Stream管理
同步等待
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

7.1 内存二次分配管理

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**10.5.5.1 总体说明**。

接口	用途	输入内存/输出内存
acl.rt.memcpy_async	实现Host内、Host与Device之间、Device内的异步内存复制。	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。
acl.rt.malloc	在Device上申请size大小的线性内存，通过**dev_ptr**返回已分配内存的指针地址。	• 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。
acl.rt.malloc_host	应用在Host上运行时，调用该接口申请的是Host内存，由系统保证内存首地址64字节对齐。应用在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。同步接口。	• 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。

接口	用途	输入内存/输出内存
acl.rt.malloc_cached	申请Device上的内存，同步接口，该接口在任何场景下申请的内存都是支持cache缓存。在Device上申请size大小的线性内存，通过**dev_ptr**返回已分配内存的指针地址。	其它约束与acl.rt.malloc接口相同。

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5.3 媒体数据处理V1。

图 7-1 VDEC 场景

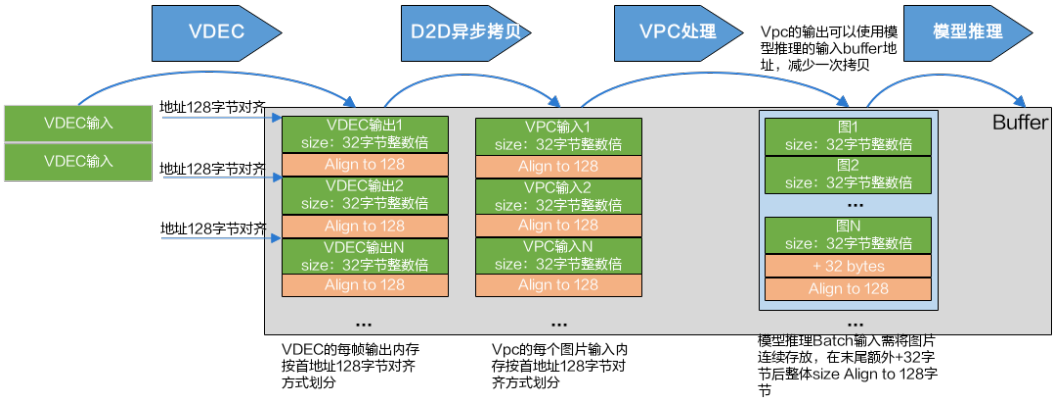
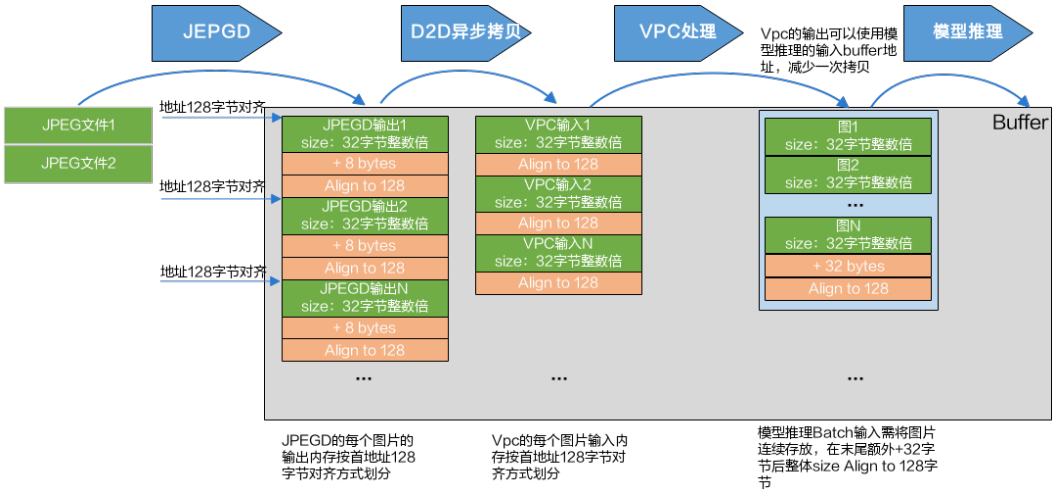


图 7-2 JPEGD 场景



7.2 多 Device 场景

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**3.4 pyACL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**acl.rt.set_context**接口）来切换Device，比使用**acl.rt.set_device**接口效率高。
- 调用**acl.rt.synchronize_device**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 开发基础推理应用**。
- 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**4.6 模型推理基本场景**。

多模型推理与单模型推理的不同点如下：

- 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**acl.mdl.execute**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**4.6 模型推理基本场景**。

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参数需设置具体的Batch数，详细说明请参见《**ATC工具使用指南**》。
- 在推理前，需要编写一段代码逻辑：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，不可以直接拷贝运行，仅供参考，此处以**8Batch**为例：

```
batch_size = 8
device_num = 1
device_id = 0

# 获取模型第一个输入的大小
model_input_size = acl.mdl.get_input_size_by_index(model_desc, 0)
# 获取每个Batch输入数据的大小
single_buff_size = model_input_size / batch_size

# 定义该变量，用于累加Batch数是否达到8Batch。
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。
pos = 0
infer_file_vec = []

for i in len(files_list):
    # 每8个文件，申请一次Device上的内存，存放8Batch的输入数据 。
    if cnt % batch_size == 0:
        pos = 0
        infer_file_vec.clear()
        # 申请Device上的内存
        p_batch_dst, ret = acl.rt.malloc(model_input_size, ACL_MEM_MALLOC_NORMAL_ONLY)

    # TODO: 从某个目录下读入文件，计算文件大小file_size。

    # 根据文件大小，申请Host上的内存，存放文件数据。
    p_img_buf, ret = acl.rt.malloc_host(file_size)

    # 将Host上的文件数据复制到Device的内存中。
    ret = acl.rt.memcpy(p_batch_dst + pos, file_size, p_img_buf, file_size, ACL_MEMCPY_HOST_TO_DEVICE)
    pos += file_size
    # 释放Host上的内存。
    acl.rt.free_host(p_img_buf)
    # 将第i个文件存入vector中，同时cnt++
    infer_file_vec.append(files_list[i])
    cnt++
    # 每8Batch的输入数据送给模型推理进行推理。
    if cnt % batch_size == 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acl.mdl.execute接口执行模型推理。
        # TODO: 推理结束后，调用acl.rt.free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 % batch_size != 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acl.mdl.execute接口执行模型推理。
```

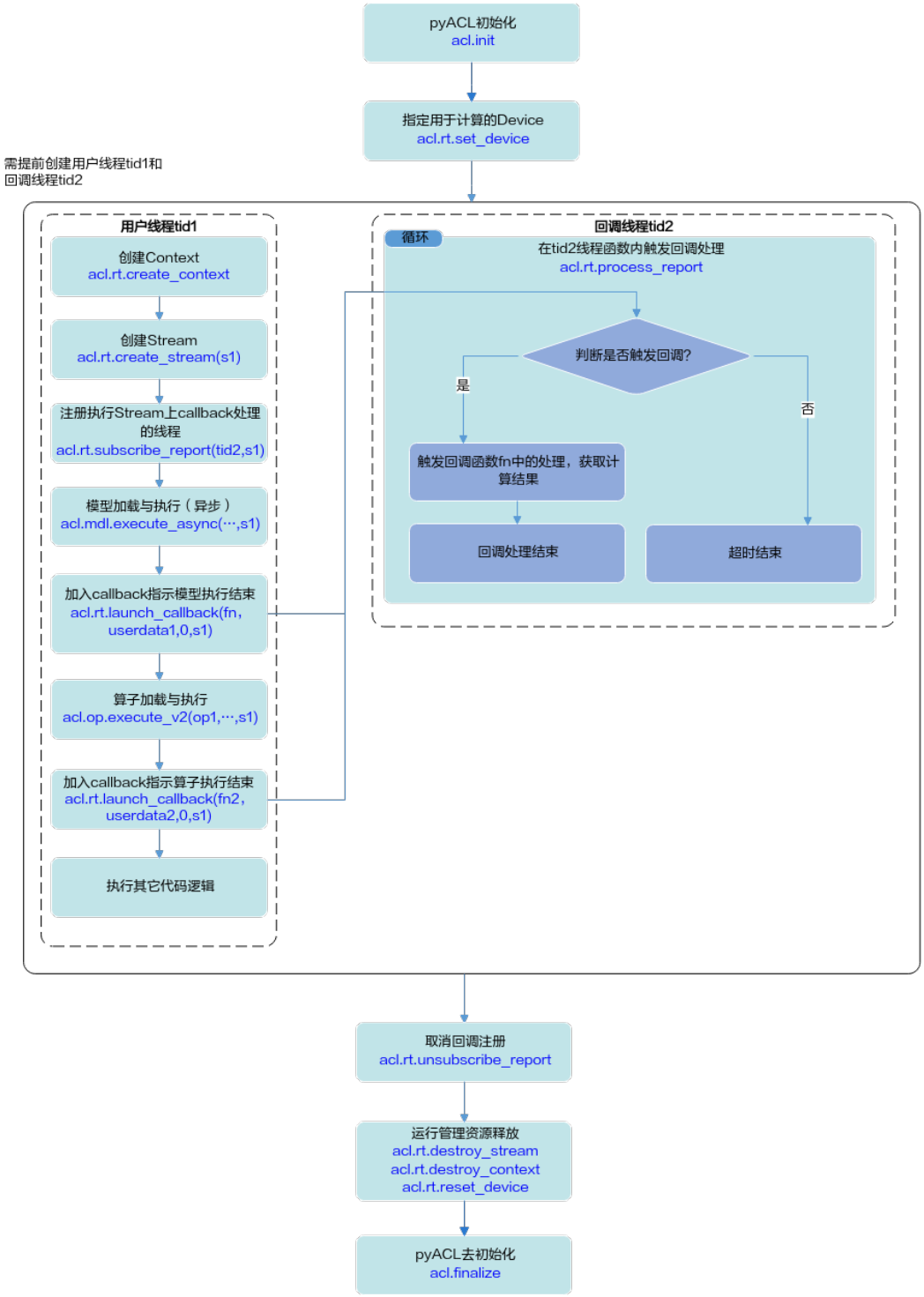
```
    # TODO: 推理结束后，调用acl.rt.free接口释放Device上的内存。
# ......
```

7.5 模型异步推理

7.5.1 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**acl.rt.process_report**接口，等待指定时间后，触发**1**中的回调函数。
3. 调用**acl.rt.subscribe_report**接口：指定处理Stream上回调函数的线程，线程与**2**中保持一致。

4. 调用**acl.rt.launch_callback**接口：在Stream的任务队列中增加一个需要执行的回调函数，回调函数与**1**中保持一致。
5. 调用**acl.rt.unsubscribe_report**接口：取消线程注册（Stream上的回调函数不再由指定线程处理）。
6. 异步推理时调用**acl.mdl.execute_async**接口。对于异步接口，还需调用**acl.rt.synchronize_stream**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**acl.rt.synchronize_stream**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。

7.5.2 示例代码

您可以从**样例介绍**中查看完整样例代码。

在acl_resnet50_async样例中：

- 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10次。
  - device id默认使用编号为0的设备。
  - 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10个。
- 运行可执行文件，带参数时：
  - 第一个参数表示使用设备ID。
  - 第二个参数表示执行模型异步推理的次数。
  - 第三个参数表示下发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。
  - 第四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1：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2：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。
  - 第五个参数表示要加载执行的模型的路径。
  - 第六个参数表示输入的图片路径。支持图片格式如下：
    IMG_EXT = ['.jpg', '.JPG', '.png', '.PNG', '.bmp', '.BMP', '.jpeg', '.JPEG']

```
import acl
import argparse
# ......

images_list = [os.path.join(args.images_path, img) for img in os.listdir(args.images_path) \
               if os.path.splitext(img)[1] in IMG_EXT]
data_list = []
for image in images_list:
    # 自定义函数transfer_pic，完成以下功能：
    # 加载图片到内存，并resize成 224x224尺寸。
```

```
    transfer_pic(image)
    dst_im = np.fromfile(image.replace(".jpg", ".bin"), dtype=np.byte)
    data_list.append(dst_im)

# 1.资源初始化： 样例中通过Net类实现资源初始化操作。
# 1.1 pyACL初始化。
ret = acl.init()
# 1.2 运行管理资源申请。
ret = acl.rt.set_device(self.device_id)
self.context, ret = acl.rt.create_context(self.device_id)
self.stream, ret = acl.rt.create_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。
self.run_mode, ret = acl.rt.get_run_mode()
# 1.3 申请模型推理资源。
# 1.4 加载模型。
# 加载离线模型文件，模型加载成功，返回标识模型的ID。
self.model_id, ret = acl.mdl.load_from_file(self.model_path)
# 1.5 根据模型的ID，获取该模型的描述信息。
self.model_desc = acl.mdl.create_desc()
ret = acl.mdl.get_desc(self.model_desc, self.model_id)

# 2.模型推理。
# 2.1 根据输入参数内存池中内存块的个数申请内存，拷贝图片数据到device侧。
def _data_interaction(self, images_dataset_list):
    for idx in range(self.memory_pool):
        img_idx = idx % len(images_dataset_list)
        img_input = self._load_input_data(images_dataset_list[img_idx])
        infer_ouput = self._load_output_data()
        self.dataset_list.append([img_input, infer_ouput])
# 2.2 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acl.rt.process_report接口，等待指定时间后，触发回调函数处理。
tid, ret = acl.util.start_thread(self._process_callback, [self.context, 50])
# 2.3 指定处理Stream上回调函数的线程。
ret = acl.rt.subscribe_report(tid, self.stream)
# 2.4 创建回调函数，用户处理模型推理的结果，由用户自行定义。
def callback_func(self, delete_list):
    for temp in delete_list:
        _, infer_output = temp
        # device to host
        num = acl.mdl.get_dataset_num_buffers(infer_output)
        for i in range(num):
            temp_output_buf = acl.mdl.get_dataset_buffer(infer_output, i)
            infer_output_ptr = acl.get_data_buffer_addr(temp_output_buf)
            infer_output_size = acl.get_data_buffer_size(temp_output_buf)
            output_host, ret = acl.rt.malloc_host(infer_output_size)
            ret = acl.rt.memcpy(output_host,
                                infer_output_size,
                                infer_output_ptr,
                                infer_output_size,
                                ACL_MEMCPY_DEVICE_TO_HOST)
            output_host_dict = ["buffer": output_host, "size": infer_output_size}]
            result = self.get_result(output_host_dict)
            st = struct.unpack("1000f", bytearray(result[0]))
            vals = np.array(st).flatten()
            top_k = vals.argsort()[-1:-6:-1]
            print("\n======== top5 inference results: =============")
            for n in top_k:
                print("[%d]: %f" % (n, vals[n]))
            ret = acl.rt.free_host(output_host)
# 2.5 自定义函数forward，执行模型推理。
def forward(self):
    self.excute_dataset = []
    for idx in range(self.excute_times):
        img_data, infer_output = self.dataset_list.pop(0)
        ret = acl.mdl.execute_async(self.model_id,
                                    img_data,
                                    infer_output,
                                    self.stream)
```

```
        if self.is_callback:
            self.excute_dataset.append([img_data, infer_output])
            self._get_callback(idx)
# 2.6 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。
ret = acl.rt.synchronize_stream(self.stream)
# 2.7 取消线程注册，Stream上的回调函数不再由指定线程处理。
ret = acl.rt.unsubscribe_report(tid, self.stream)
self.is_exist = True
ret = acl.util.stop_thread(tid)

# 3.释放运行管理资源。
ret = acl.rt.destroy_stream(self.stream)
ret = acl.rt.destroy_context(self.context)
ret = acl.rt.reset_device(self.model_id)

# 4.pyACL去初始化。
ret = acl.finalize()
# ......
```

7.6 模型动态推理

7.6.1 动态 Batch/动态分辨率/动态维度（设置多档维度值）

接口调用流程

动态Shape输入场景下模型推理与**4 开发基础推理应用**的流程类似，都涉及pyACL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 开发基础推理应用**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pyACL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pyACL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pyACL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**》。

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**acl.mdl.get_input_index_by_name**接口根据输入名称（固定为“ascend_mbatch_shape_data”）获取模型中标识该输入的index。

ii. 调用**acl.mdl.get_input_size_by_index**根据index获取输入内存大小。

iii. 调用**acl.rt.malloc**接口根据**2.ii**中的大小申请内存。

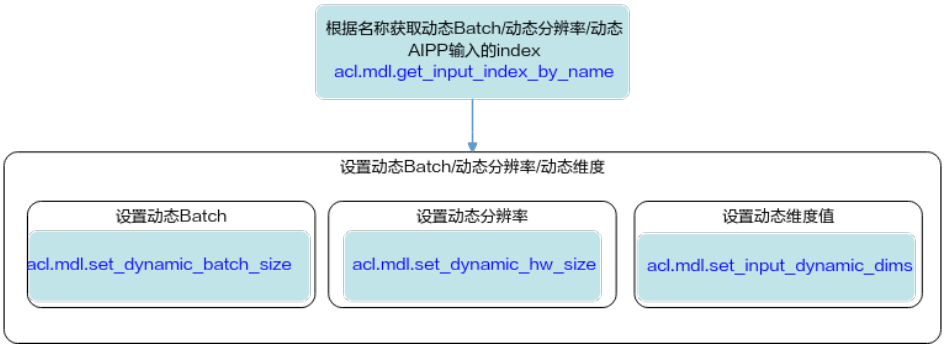
申请动态Batch/动态分辨率/动态AIPP/动态维度输入对应的内存后，无需用户设置该内存中的数据（否则可能会导致业务异常），用户调用**2.ii**中的接口后，系统会自动向该内存中填入数据。

iv. 调用**acl.create_data_buffer**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**acl.mdl.create_dataset**接口创建aclmdlDataset类型的数据，并调用**acl.mdl.add_dataset_buffer**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**acl.mdl.get_input_index_by_name**接口根据输入名称（固定为“ascend_mbatch_shape_data”）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**acl.mdl.set_dynamic_batch_size**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**acl.mdl.get_dynamic_batch**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**acl.mdl.set_dynamic_hw_size**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**acl.mdl.get_dynamic_hw**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**acl.mdl.set_input_dynamic_dims**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**acl.mdl.get_input_dynamic_dims**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。

须知

- 对同一个模型，不能同时调用**acl.mdl.set_dynamic_batch_size**接口设置动态Batch、调用**acl.mdl.set_dynamic_hw_size**接口设置动态分辨率、调用**acl.mdl.set_input_dynamic_dims**接口设置动态维度的维度值，只能调用其中一种。
- 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**acl.mdl.get_output_size_by_index**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**acl.mdl.create_aipp**接口设置“batchSize”时，“batchSize”要设置为最大Batch数。
  - 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**acl.mdl.set_aipp_src_image_size**接口设置的宽、高与通过**acl.mdl.set_dynamic_hw_size**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。
- 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。
- 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。

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，不可以直接拷贝运行，仅供参考。

```
# 1.模型加载，加载成功后，再设置动态Batch。
# ......

# 2.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。
# ......

# 3.自定义函数，设置动态Batch。
def model_set_dynamicinfo():
    # 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“ascend_mbatch_shape_data”。
    index, ret = acl.mdl.get_input_index_by_name(model_desc, "ascend_mbatch_shape_data")
    # 3.2 设置Batch，model_id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 = 8
    ret = acl.mdl.set_dynamic_batch_size(model_id_, input, index, batch_size)
    # ......

# 4.自定义函数，执行模型。
def model_execute(index):
```

```
    # 4.1 调用自定义函数，设置动态Batch。
    ret = model_set_dynamicinfo()
    # 4.2 执行模型，model_id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 = acl.mdl.execute(model_id, input, output)
    # ......
}

# 5.处理模型推理结果。
# ......
```

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.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。
# ......

# 3.自定义函数，设置动态分辨率。
def model_set_dynamicInfo():
    # 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
“ascend_mbatch_shape_data”。
    index, ret = acl.mdl.get_input_index_by_name(modelDesc, "ascend_mbatch_shape_data")
    # 3.2 设置输入图片分辨率，model_id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。
    height = 224
    width = 224
    ret = acl.mdl.set_dynamic_hw_size(model_id, input, index, height, width)
    # ......

# 4.自定义函数，执行模型。
def model_execute(index):
    # 4.1 调用自定义函数，设置动态分辨率。
    ret = model_set_dynamicInfo()
    # 4.2 执行模型，model_id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 = acl.mdl.execute(model_id, input, output)
    # ......

# 5.处理模型推理结果。
# ......
```

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，示例代码中不一一列举。以下是关键步骤的代码示例，不可以直接拷贝运行，仅供参考。

```
import acl
# ......

# 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model_desc，准备模型的输入数据input_dataset和模型的输出数据output_dataset。
# ......

# 3.自定义函数，设置动态维度。
def model_set_dynamic_info():
    # 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“ascend_mbatch_shape_data”。
    index, ret = acl.mdl.get_input_index_by_name(model_desc, "ascend_mbatch_shape_data")
    # 3.2 设置具体档位信息，包括维度数dimCount和各个维度的数值，model_id表示加载成功的模型的ID，
input_dataset表示aclmdlDataset类型的数据，index表示标识动态维度输入的输入index。
    current_dims = {'name': '', 'dimCount': 4, 'dims': [8, 3, 224, 224]}
    ret = acl.mdl.set_input_dynamic_dims(mode_id, input_dataset, index, current_dims)
    # ......
```

```
# 4.自定义函数，执行模型。
def ModelExecute(int index):
    # 4.1 调用自定义函数，设置动态维度。
    ret = model_set_dynamic_info()
    # 4.2 执行模型，model_id表示加载成功的模型的ID，input_dataset和output_dataset分别表示模型的输入和输出。
    ret = acl.mdl.execute(model_id, input_dataset, output_dataset)
    # ......

# 5.处理模型推理结果。
# ......
```

7.7 模型动态 AIPP 推理

7.7.1 接口调用流程

动态AIPP场景下模型推理与**4 开发基础推理应用**的流程类似，都涉及pyACL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 开发基础推理应用**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。

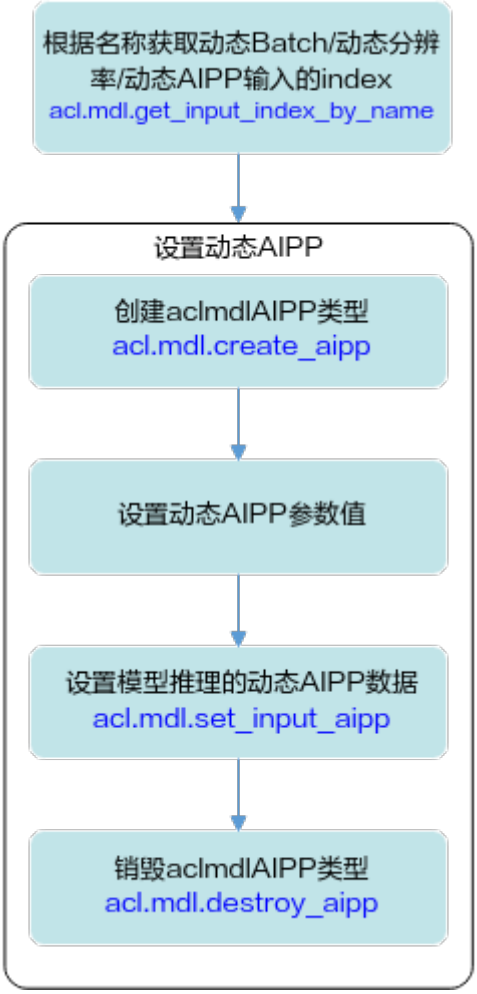
  例如，a输入的AIPP配置是动态的，在om模型中，会有与a对应的b输入来描述a的AIPP配置信息。在模型执行时，准备a输入的数据结构请参见**准备模型执行的输入/输出数据结构**，准备b输入的数据结构、设置b输入的数据请参见以下内容。

- **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**acl.mdl.get_input_index_by_name**接口根据输入名称（固定为“ascend_dynamic_aipp_data”）获取模型中标识该输入的index。
    ii. 调用**acl.mdl.get_input_size_by_index**根据index获取输入内存大小。
    iii. 调用**acl.rt.malloc**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**acl.create_data_buffer**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**acl.mdl.create_dataset**接口创建aclmdlDataset类型的数据，并调用**acl.mdl.add_dataset_buffer**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**acl.mdl.get_input_index_by_name**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**acl.mdl.create_aipp**接口创建aclmdlAIPP类型。
2) 根据实际需求，调用**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，**acl.mdl.set_aipp_src_image_size**接口（设置原始图片的宽和高）必须调用。
4) 调用**acl.mdl.set_input_aipp**接口设置模型推理时的动态AIPP数据。
5) 及时调用**acl.mdl.destroy_aipp**接口销毁aclmdlAIPP类型。

说明

pyACL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 图像/视频数据处理**。

7.7.2 动态 AIPP（单个动态 AIPP 输入）

基本原理

若模型推理时包含**动态AIPP**特性，在模型推理时，需调用pyACL提供的接口设置模型推理时需使用的AIPP配置，模型支持的AIPP模式已提前在构建模型时配置（使用ATC工具的“insert_op_conf”参数）。

ATC工具的参数说明请参见《**ATC工具使用指南**》。

须知

- 动态AIPP和动态Batch同时使用时：
  - 调用**acl.mdl.create_aipp**接口设置“batchSize”时，“batchSize”要设置为最大Batch数。
  - 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**acl.mdl.set_aipp_src_image_size**接口设置的宽、高与通过**acl.mdl.set_dynamic_hw_size**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。
- 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。

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，不可以直接拷贝运行，仅供参考。

```
import acl
# ......

ACL_YUV420SP_U8 = 1

# 1.模型加载，加载成功后，再设置动态AIPP参数值。
# ......

# 2.创建aclmdlDataset类型的数据，用于描述模型的输入数据input_dataset和模型的输出数据output_dataset。
# ......

# 3.自定义函数，设置动态AIPP参数值。
def model_set_dynamic_aipp():
    # 3.1 获取标识动态AIPP输入的index。
    index, ret = acl.mdl.get_input_index_by_name(model_desc, "ascend_dynamic_aipp_data")

    # 3.2 设置动态AIPP参数值。
    batch_number = 1
    aipp_dynamic_set = acl.mdl.create_aipp(batch_number)
```

```
    ret = acl.mdl.set_aipp_src_image_size(aipp_dynamic_set, 256, 224)
    ret = acl.mdl.set_aipp_input_format(aipp_dynamic_set, ACL_YUV420SP_U8)
    ret = acl.mdl.set_aipp_csc_params(aipp_dynamic_set, 1, 256, 443, 0, 256, -86, -178, 256,
                                      0, 350, 0, 0, 0, 0, 128, 128)
    ret = acl.mdl.set_aipp_rbuv_swap_switch(aipp_dynamic_set, 0)
    ret = acl.mdl.set_aipp_dtc_pixel_mean(aipp_dynamic_set, 0, 0, 0, 0, 0)
    ret = acl.mdl.set_aipp_dtc_pixel_min(aipp_dynamic_set, 0, 0, 0, 0, 0)
    ret = acl.mdl.set_aipp_pixel_var_reci(aipp_dynamic_set, 1, 1, 1, 1, 0)
    ret = acl.mdl.set_aipp_crop_params(aipp_dynamic_set, 1, 0, 0, 224, 224, 0)
    ret = acl.mdl.set_input_aipp(model_id, input_dataset, index, aipp_dynamic_set)
    ret = acl.mdl.destroy_aipp(aipp_dynamic_set)
    # ......

# 4.自定义函数，执行模型。
def model_execute(index)
    # 4.1 调用自定义函数，设置动态AIPP参数值。
    ret = model_set_dynamic_aipp()
    # 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 = acl.mdl.execute(model_id, input_dataset, output_dataset)
    # ......

# 5.处理模型推理结果。
# ......
```

7.7.3 动态 AIPP（多个动态 AIPP 输入）

基本原理

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**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**acl.mdl.get_aipp_type**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**acl.mdl.set_aipp_by_input_index**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**acl.mdl.get_input_name_by_index**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，示例代码中不一一列举。以下是关键步骤的代码示例，不可以直接拷贝运行，仅供参考。

```
import acl
# ......

ACL_DATA_WITH_DYNAMIC_AIPP = 2
ACL_YUV420SP_U8 = 1

# 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model_desc，准备模型的输入数据input_dataset和模型的输出数据output_dataset。
# ......

# 3.自定义函数，设置动态AIPP参数值。
def model_set_dynamic_aipp():
    # 3.1 获取标识动态AIPP输入的index。
    need_dynamic_aipp = []
    input_num = acl.mdl.get_num_inputs(model_desc)
    for index in range(input_num):
```

```
        aipp_type, dynamic_attached_index, ret = acl.mdl.get_aipp_type(model_id, index)
        if aipp_type == ACL_DATA_WITH_DYNAMIC_AIPP:
            need_dynamic_aipp.append(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 len(need_dynamic_aipp) != 2:
        return 1
    # 创建第一个动态aipp配置参数。
    batch_number_first = 1
    aipp_dynamic_set_first = acl.mdl.create_aipp(batch_number_first)
    ret = acl.mdl.set_aipp_src_image_size(aipp_dynamic_set_first, 256, 224)
    ret = acl.mdl.set_aipp_input_format(aipp_dynamic_set_first, ACL_YUV420SP_U8)
    ret = acl.mdl.set_aipp_csc_params(aipp_dynamic_set_first, 1, 256, 443, 0, 256, -86,
                                      -178, 256, 0, 350, 0, 0, 0, 0, 128, 128)
    ret = acl.mdl.set_aipp_rbuv_swap_switch(aipp_dynamic_set_first, 0)
    ret = acl.mdl.set_aipp_dtc_pixel_mean(aipp_dynamic_set_first, 0, 0, 0, 0, 0)
    ret = acl.mdl.set_aipp_dtc_pixel_min(aipp_dynamic_set_first, 0, 0, 0, 0, 0)
    ret = acl.mdl.set_aipp_pixel_var_reci(aipp_dynamic_set_first, 1.0, 1.0, 1.0, 1.0, 0)
    ret = acl.mdl.set_aipp_crop_params(aipp_dynamic_set_first, 1, 2, 2, 224, 224, 0)
    # 设置模型推理时的动态aipp参数值。
    ret = acl.mdl.set_aipp_by_input_index(model_id, input_dataset, need_dynamic_aipp[0],
                                          aipp_dynamic_set_first)
    ret = acl.mdl.destroy_aipp(aipp_dynamic_set_first)
    # 创建第二个动态aipp配置参数。
    batch_number_second = 2
    aipp_dynamic_set_second = acl.mdl.create_aipp(batch_number_second)
    ret = acl.mdl.set_aipp_src_image_size(aipp_dynamic_set_second, 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 = acl.mdl.set_aipp_by_input_index(model_id, input_dataset, need_dynamic_aipp[1],
                                          aipp_dynamic_set_second)
    ret = acl.mdl.destroy_aipp(aipp_dynamic_set_second)
    return ret

# 4.自定义函数，执行模型。
def ModelExecute(int index):
    # 4.1 调用自定义函数，设置动态AIPP参数值。
    ret = model_set_dynamic_aipp()
    # 4.2 执行模型，modelId_表示加载成功的模型的ID，input_dataset和output_dataset分别表示模型的输入和
输出。
    ret = acl.mdl.execute(model_id, input_dataset, output_dataset)
    # ......

# 5.处理模型推理结果。
# ......
```

7.8 Stream 管理

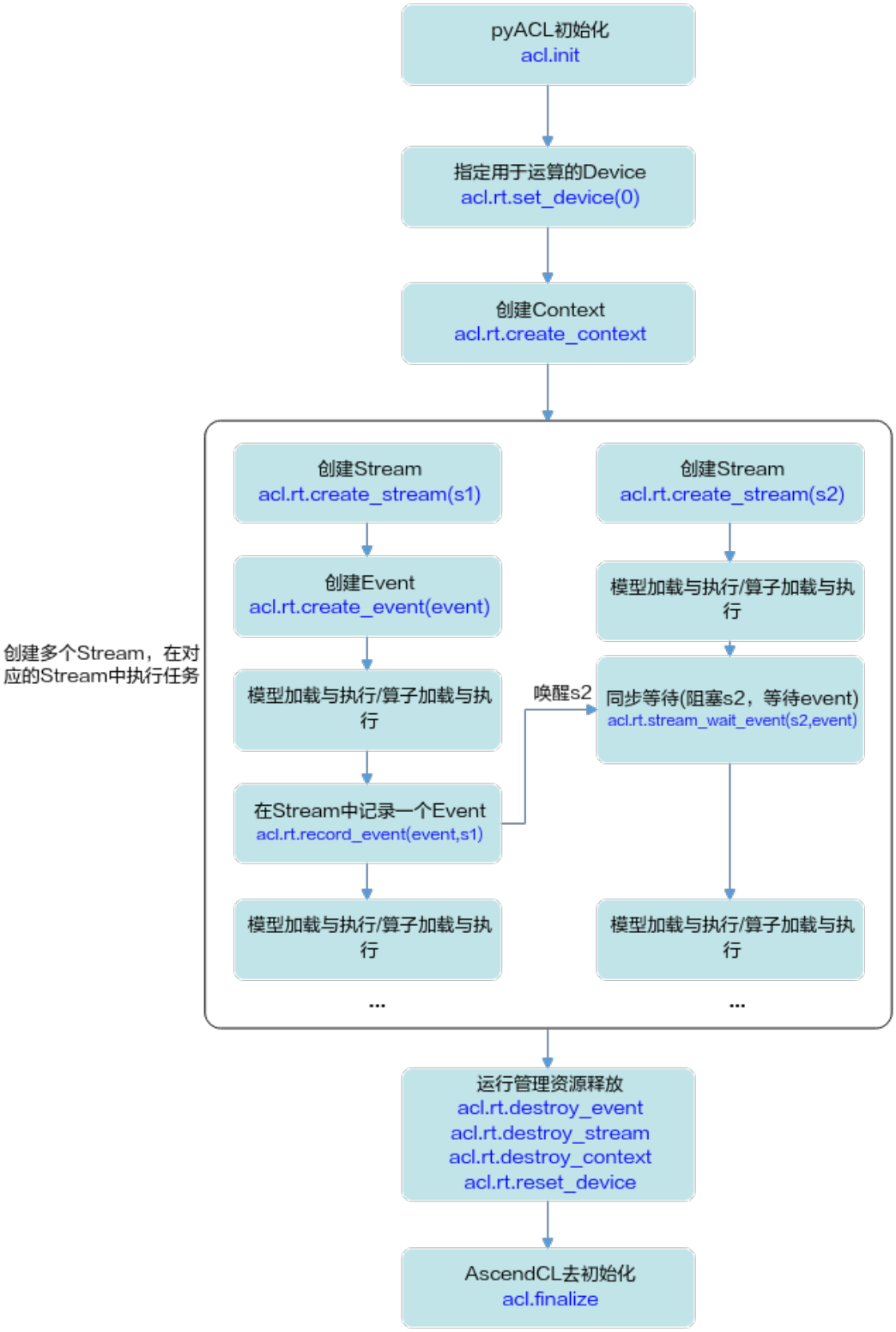
7.8.1 原理介绍

在pyACL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0。

7.8.2 多 Stream 接口调用流程

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**3.4 pyACL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**acl.rt.stream_wait_event**接口阻塞指定Stream的运行，直到指定的Event完成。需在调用**acl.rt.stream_wait_event**接口前，先调用**acl.rt.record_event**接口。调用示例请参见**7.9.4 关于Stream间任务的同步等待**。

模型加载与执行的流程请参见**4 开发基础推理应用**。

算子加载与执行的流程请参见**6 单算子调用**。

7.8.3 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，不可以直接拷贝运行，仅供参考。

```
import acl
# ......
ACL_MEMCPY_HOST_TO_DEVICE = 1

# 显式创建一个Stream。
stream, ret = acl.rt.create_stream()

# 调用触发任务的接口，传入stream参数。
ret = acl.rt.memcpy_async(dev_ptr, dev_size, host_ptr, host_size,
                          ACL_MEMCPY_HOST_TO_DEVICE, stream)
# 调用acl.rt.synchronize_stream接口，阻塞应用程序运行，直到指定Stream中的所有任务都完成。
ret = acl.rt.synchronize_stream(stream)

# Stream使用结束后，显式销毁Stream。
ret = acl.rt.destroy_stream(stream)
# ......
```

7.8.4 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，不可以直接拷贝运行，仅供参考。

```
import acl
# ......

device_id = 0
model_id_1 = 0
model_id_2 = 1

# 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acl.rt.create_context接口设置当前线程的Context。
context, ret = acl.rt.create_context(device_id)

stream1, ret = acl.rt.create_stream()
# 调用触发任务的接口，例如异步模型推理，任务下发在stream1。
ret = acl.mdl.execute_async(model_id_1, dataset_in_1, dataset_out_1, stream1)

stream2, ret = acl.rt.create_stream()
# 调用触发任务的接口，例如异步模型推理， 任务下发在stream2。
ret = acl.mdl.execute_async(model_id_2, dataset_in_2, dataset_out_2, stream2)

# 流同步。
ret = acl.rt.synchronize_stream(stream1)
ret = acl.rt.synchronize_stream(stream2)

# 释放资源。
ret = acl.rt.destroy_stream(stream2)
ret = acl.rt.destroy_stream(stream1)
ret = acl.rt.destroy_context(context)

# ....
```

7.8.5 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，不可以直接拷贝运行，仅供参考。

```
import acl
# ......
def run_thread(stream):
    device_id = 0

    # 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acl.rt.set_context接口设置当前线程的Context。
    context, ret = acl.rt.create_context(device_id)
    stream, ret = acl.rt.create_stream()

    # 调用触发任务的接口。
    # ....

    # 释放资源。
    ret = acl.rt.destroy_stream(stream)
    ret = acl.rt.destroy_context(context)


# 创建2个线程，每个线程对应一个Stream。
thread_id1, ret = acl.util.start_thread(run_thread, stream1)
thread_id2, ret = acl.util.start_thread(run_thread, stream2)
# 显式调用join函数确保结束线程。
ret = acl.util.stop_thread(thread_id1)
ret = acl.util.stop_thread(thread_id2)
```

7.9 同步等待

7.9.1 基本原理

同步机制

pyACL提供以下几种同步机制：

- Event的同步等待：调用**acl.rt.synchronize_event**接口，阻塞应用程序运行，等待Event完成。
- Stream内任务的同步等待：调用**acl.rt.synchronize_stream**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：调用**acl.rt.stream_wait_event**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**7.8.2 多Stream接口调用流程**。
- Device的同步等待：调用**acl.rt.synchronize_device**接口，阻塞应用程序运行，直到正在运算中的Device完成运算。

7.9.2 关于 Event 的同步等待

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，不可以直接拷贝运行，仅供参考。

```
import acl
# ......
# 创建一个Event。
event, ret = acl.rt.create_event()

# 创建一个Stream。
stream, ret = acl.rt.create_stream()

# stream末尾添加了一个event。
ret = acl.rt.record_event(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。
ret = acl.rt.synchronize_event(event)

# 显式销毁资源。
ret = acl.rt.destroy_stream(stream)
ret = acl.rt.destroy_event(event)
# ......
```

7.9.3 关于 Stream 内任务的同步等待

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，不可以直接拷贝运行，仅供参考。

```
import acl
# ......

# 显式创建一个Stream。
stream, ret = acl.rt.create_stream()

# 调用触发任务的接口，传入stream参数。
ret = acl.rt.memcpy_async(dev_ptr, dev_size, host_ptr, host_size,
                          ACL_MEMCPY_HOST_TO_DEVICE, stream)
# 调用acl.rt.synchronize_stream接口，阻塞应用程序运行，直到指定Stream中的所有任务都完成。
ret = acl.rt.synchronize_stream(stream)

# Stream使用结束后，显式销毁Stream。
ret = acl.rt.destroy_stream(stream)
# ......
```

7.9.4 关于 Stream 间任务的同步等待

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，不可以直接拷贝运行，仅供参考。

```
import acl
# ......
# 创建一个Event。
event, ret = acl.rt.create_event()

# 创建两个Stream。
s1, ret = acl.rt.create_stream()
s2, ret = acl.rt.create_stream()

# 在s1末尾添加了一个event。
ret = acl.rt.record_event(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。
ret = acl.rt.stream_wait_event(s2, event)

# 显式销毁资源。
ret = acl.rt.destroy_stream(s2)
ret = acl.rt.destroy_stream(s1)
ret = acl.rt.destroy_event(event)
# ......
```

7.9.5 关于 Device 的同步等待

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，示例代码中不一一列举。以下是关键步骤的代码示例，不可以直接拷贝运行，仅供参考。

```
import acl
# ......

device_id = 0
```

```
# 指定Device。
ret = acl.rt.set_device(device_id)

# 创建context。
context, ret = acl.rt.create_context(device_id)

# 创建stream。
stream, ret = acl.rt.create_stream()

# 阻塞应用程序运行，直到正在运算中的Device完成运算。
ret = acl.rt.synchronize_device()

# 资源销毁。
ret = acl.rt.synchronize_stream(stream)
ret = acl.rt.destroy_context(context)
ret = acl.rt.reset_device(device_id)
```

7.10 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，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**acl.rt.set_exception_info_callback** 。

   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**acl.rt.get_device_id_from_exception_info**、**acl.rt.get_stream_id_from_exception_info**、**acl.rt.get_task_id_from_exception_info**接口分别获取Device ID、Stream ID、Task ID。

   b. 在异常回调函数fn内调用**acl.mdl.create_and_get_op_desc**接口获取算子的描述信息。

   c. 在异常回调函数fn内调用**acl.get_tensor_desc_by_index**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**acl.get_tensor_desc_address**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**acl.get_tensor_desc_type**接口获取tensor描述中的数据类型、调用**acl.get_tensor_desc_format**接口获取tensor描述中的Format、调用**acl.get_tensor_desc_num_dims**接口获取tensor描述中的Shape维度个数、调用**acl.get_tensor_desc_dim_v2**接口获取Shape中指定维度的大小。

2. 调用**acl.rt.set_exception_info_callback**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，示例代码中不一一列举。以下是关键步骤的代码示例，不可以直接拷贝运行，仅供参考。

示例中，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**，模型加载的接口调用流程请参见**接口调用流程**，模型推理的接口调用流程、准备模型推理的输入/输出数据的接口调用流程请参见**准备模型执行的输入/输出数据结构**。

```
import acl
import numpy as np
# ......

# 1.申请运行管理资源，包括设置用于计算的Device、创建Context、创建Stream。
# ......

# 2.模型加载，加载成功后，返回标识模型的model_id。
# ......

# 3.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。
# ......

# 4.实现异常回调函数。
def exception_callback(info):
    stream_id = acl.rt.get_stream_id_from_exception_info(info)
    thread_id = acl.rt.get_thread_id_from_exception_info(info)
    device_id = acl.rt.get_device_id_from_exception_info(info)
    task_id = acl.rt.get_task_id_from_exception_info(info)
    # 用户可以将获取的算子信息写入到文件，或者另起线程侦听异常回调，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打屏。
    op_name, input_desc, num_inputs, output_desc, num_outputs, ret = \
        acl.mdl.create_and_get_op_desc(device_id, stream_id, task_id, 256)

    # 可以调用acl 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 i in range(num_inputs):
        desc = acl.get_tensor_desc_by_index(input_desc, i)
        address = acl.get_tensor_desc_address(desc)
        num_dims = acl.get_tensor_desc_num_dims(desc)
        dim_0, ret = acl.get_tensor_desc_dim_v2(desc, 0)

    for i in range(num_outputs):
        desc = acl.get_tensor_desc_by_index(output_desc, i)
        address = acl.get_tensor_desc_address(desc)
        num_dims = acl.get_tensor_desc_num_dims(desc)
        dim_0, ret = acl.get_tensor_desc_dim_v2(desc, 0)

    acl.destroy_tensor_desc(input_desc)
    acl.destroy_tensor_desc(output_desc)

# 5.设置异常回调。
ret = acl.rt.set_exception_info_callback(exception_callback)

# 6.执行模型。
ret = acl.mdl.execute(model_id, input, output)

# 7.处理模型推理结果。
# ......

# 8.释放描述模型输入/输出信息、内存等资源，卸载模型。
# ......

# 9.释放运行管理资源。
# ......
```

7.11 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 pyACL API（通过Profiling pyACL 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《性能分析工具使用指南》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：

- **acl.prof.init**接口、**acl.prof.start**接口、**acl.prof.stop**接口、**acl.prof.finalize**接口配合使用，实现该方式的性能数据采集。该方式可获取pyACL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**acl.init**接口，在pyACL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pyACL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**acl.init**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**acl.init**接口处的说明，不在本章节描述。

Profiling pyACL API for Extension（Profiling pyACL 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（**acl.prof.start**接口、**acl.prof.stop**接口之间）调用Profiling pyACL 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《性能分析工具使用指南》下的“Profiling数据解析”和“Profiling数据导出”。

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：在**acl.prof.start**和**acl.prof.stop**接口之间调用**acl.prof.create_stamp**、**acl.prof.push**、**acl.prof.pop**、**acl.prof.range_start**、**acl.prof.range_stop**、**acl.prof.destroy_stamp**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。

Profiling pyACL API for Subscription（订阅算子信息的Profiling pyACL 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pyACL的接口获取性能数据。

接口调用方式：**acl.prof.model_subscribe**接口、acl.prof.get*接口、**acl.prof.model_un_subscribe**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 pyACL 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，不可以直接拷贝运行，仅供参考。

示例中，运行管理资源申请与释放请参见运行管理资源申请流程与运行管理资源释放流程，模型加载的接口调用流程请参见接口调用流程，模型推理的接口调用流程、准备模型推理的输入/输出数据的接口调用流程请参见准备模型执行的输入/输出数据结构。

```
import acl
import numpy as np
```

```
# ......

# 1.申请运行管理资源，包括设置用于计算的Device、创建Context、创建Stream。
# ......

# 2.模型加载，加载成功后，返回标识模型的model_id。
# ......

# 3.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。
# ......

# 4.profiling初始化。
# 设置数据落盘路径。
PROF_INIT_PATH='...'
ret = acl.prof.init(PROF_INIT_PATH)

# 5.进行profiling配置。
device_list = [0]
ACL_PROF_ACL_API = 0x0001
ACL_PROF_TASK_TIME = 0x0002
ACL_PROF_AICORE_METRICS = 0x0004
ACL_PROF_AICPU_TRACE = 0x0008
ACL_PROF_SYS_HARDWARE_MEM_FREQ = 3

# 创建配置类型指针地址。
prof_config = acl.prof.create_config(device_list, 0, 0, ACL_PROF_ACL_API | ACL_PROF_TASK_TIME |
ACL_PROF_AICPU | ACL_PROF_AICORE_METRICS | ACL_PROF_HCCL_TRACE)
mem_freq = "15"
ret = acl.prof.set_config(ACL_PROF_SYS_HARDWARE_MEM_FREQ, mem_freq)
ret = acl.prof.start(prof_config)

# 6.执行模型。
ret = acl.mdl.execute(model_id, input, output)

# 7.处理模型推理结果。
# ......

# 8.释放描述模型输入/输出信息、内存等资源，卸载模型。
# ......

# 9.关闭profiling配置, 释放配置资源, 释放profiling组件资源。
ret = acl.prof.stop(prof_config)
ret = acl.prof.destroy_config(prof_config)
ret = acl.prof.finalize()

# 10.释放运行管理资源
# ......
```

Profiling pyACL 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，不可以直接拷贝运行，仅供参考。

示例中，运行管理资源申请与释放请参见**运行管理资源申请流程**与**运行管理资源释放流程**，模型加载的接口调用流程请参见**接口调用流程**，模型推理的接口调用流程、准备模型推理的输入/输出数据的接口调用流程请参见**4.6.2.1 接口调用流程**。

```
for i in range(200000):
    stamp = acl.prof.create_stamp()
     if stamp == 0:
        print("create stamp failed")
        return FAILED

    msg = "test msprof tx"
    msg_len = len(msg)
    ret = acl.prof.set_stamp_trace_message(stamp, msg, msg_len)
    ret = acl.prof.mark(stamp)
```

```
    ret = acl.prof.destroy_stamp(stamp)
```

或

```
for i in range(200000):
    stamp = acl.prof.create_stamp()
     if stamp == 0:
        print("create stamp failed")
        return FAILED

    msg = "test msprof tx"
    msg_len = len(msg)
    ret = acl.prof.set_stamp_trace_message(stamp, msg, msg_len)

# acl.prof.push 和 acl.prof.pop 接口配对使用，完成单线程采集
    ret = acl.prof.push(stamp)
    ret = acl.prof.pop()

    ret = acl.prof.destroy_stamp(stamp)
```

或

```
for i in range(200000):
    stamp = acl.prof.create_stamp()
     if stamp == 0:
        print("create stamp failed")
        return FAILED

    msg = "test msprof tx"
    msg_len = len(msg)
    ret = acl.prof.set_stamp_trace_message(stamp, msg, msg_len)

# acl.prof.range_start 和 acl.prof.range_stop 接口配对使用，可以完成多线程采集
    range_id = 0
    range_id, ret = acl.prof.range_start(stamp)
    ret = acl.prof.range_stop(range_id)

    ret = acl.prof.destroy_stamp(stamp)
```

Profiling pyACL API for Subscription 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，不可以直接拷贝运行，仅供参考。

示例中，运行管理资源申请与释放请参见**运行管理资源申请流程**与**运行管理资源释放流程**，模型加载的接口调用流程请参见**接口调用流程**，模型推理的接口调用流程、准备模型推理的输入/输出数据的接口调用流程请参见**4.6.2.1 接口调用流程**。

```
import acl
import numpy as np
# ......

# 1.申请运行管理资源，包括设置用于计算的Device、创建Context、创建Stream。
# ......

# 2.模型加载，加载成功后，返回标识模型的model_id。
# ......

# 3.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。
# ......

# 4.创建管道，用于读取以及写入模型订阅的数据。
r, w = os.pipe()

# 5.创建模型订阅的配置并且进行模型订阅。
ACL_AICORE_NONE = 0xFF
subscribe_config = acl.prof.create_subscribe_config(1, ACL_AICORE_NONE, w)
```

```
# 模型订阅需要传入模型的model_id。
ret = acl.prof.model_subscribe(model_id, subscribe_config)

# 6.实现管道读取订阅数据的函数。
# 6.1 自定义函数，实现从用户内存中读取订阅数据的函数。
def get_model_info(data, data_len):
    # 获取算子信息个数。
    op_number, ret = acl.prof.get_op_num(data, data_len)
    # 遍历用户内存的算子信息。
    for i in range(op_number):
        # 获取算子的模型id。
        model_id = acl.prof.get_model_id(data, data_len, i)
        # 获取算子的类型名称。
        op_type, ret = acl.prof.get_op_type(data, data_len, i, 65)
        # 获取算子的名称。
        op_name, ret = acl.prof.get_op_name(data, data_len, i, 275)
        # 获取算子的执行开始时间。
        op_start = acl.prof.get_op_start(data, data_len, i)
        # 获取算子的执行结束时间。
        op_end = acl.prof.get_op_end(data, data_len, i)
        # 获取算子执行的耗时时间。
        op_duration = acl.prof.get_op_duration(data, data_len, i)

# 6.2 自定义函数，实现从管道中读取数据到用户内存的函数。
def prof_data_read(args):
    fd, ctx = args
    ret = acl.rt.set_context(ctx)
    # 获取单位算子信息的大小（Byte）。
    buffer_size, ret = acl.prof.get_op_desc_size()
    # 设置每次从管道中读取的算子信息个数。
    N = 10
    # 计算存储算子信息的内存的大小。
    data_len = buffer_size * N
    # 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可能小于buffer_size * N，如果管道中没有数
据，默认会阻塞直到读取到数据为止。
    while True:
        data = os.read(fd, data_len)
        if len(data) == 0:
            break
        np_data = np.array(data)

        bytes_data = np_data.tobytes()
        np_data_ptr = acl.util.bytes_to_ptr(bytes_data)
        size = np_data.itemsize * np_data.size
        # 调用6.1实现的函数解析内存中的数据。
        get_model_info(np_data_ptr, size)

# 7.启动线程读取管道数据并解析。
thr_id, ret = acl.util.start_thread(prof_data_read, [r, context])

# 8.执行模型。
ret = acl.mdl.execute(model_id, input, output)

# 9.处理模型推理结果。
# ......

# 10.释放描述模型输入/输出信息、内存等资源，卸载模型。
# ......

# 11.取消订阅，释放订阅相关资源。
ret = acl.prof.model_un_subscribe(model_id)
ret = acl.util.stop_thread(thr_id)
os.close(r)
ret = acl.prof.destroy_subscribe_config(subscribe_config)

# 12.释放运行管理资源。
# ......
```

7.12 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》。

采集溢出算子信息

在调用调用acl.init接口初始化pyACL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

8 应用调试

运行应用

编译运行应用的步骤，请参考**基于Caffe ResNet-50网络实现图片分类（同步推理）**。

相关注意点如下：

1. 模型转换，详细说明请参见《**ATC工具使用指南**》。
2. 运行时，需将pyACL初始化配置文件（acl.json）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Host的同一个目录下。

   如果在**pyACL初始化**阶段，在**acl.init**接口中不传入参数，则无需将pyACL初始化配置文件（acl.json）所在的目录上传到Host。
3. 运行代码时，直接运行对应的python脚本即可。如：

```
python3 main.py
```

问题定位

运行应用时如果出错，您可以参见《**日志参考**》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**11.2 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，可调用pyACL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**7.10 AI Core异常信息获取**，查看原理及调用示例。

9 应用样例参考

样例使用指导

9.1 样例使用指导

9.1.1 环境准备

9.1.1.1 Python 环境

前提条件

请参见**表9-1**完成环境准备。

表 9-1 安装软件

名称	版本	说明
Python	Python3.7.5	安装方式与Python依赖版本的限制请参见《**CANN 软件安装指南**》。
	Python3.8.0 ~ Python3.8.10	
	Python3.9.2	
Python依赖	-	
Numpy	≥ 1.18.2	Python的一种开源的数值计算扩展。 使用**pip3 install numpy**安装Numpy。
Pillow	7.2.0	Python的图像处理库（Python Imaging Library）。 使用**pip3 install Pillow**命令安装Pillow。

说明

本节所列的python、pip命令，实际命名与用户机器中软链接设置的命名一致，以对应python3.7.5版本为示例，请用户自行替换。

CentOS7 aarch64 环境下安装 Pillow 的说明

CentOS7 aarch64环境中通过**pip3.7 install Pillow**安装Pillow之后，在执行**from PIL import Image**的时候会报错：

```
ImportError: /usr/local/python3.7.5/lib/python3.7/site-packages/PIL/_imaging.cpython-37m-aarch64-linux-gnu.so: ELF load command alignment not page-aligned
```

这种情况下，需要用源码安装Pillow。若已经通过pip安装，需要运行**pip3.7 uninstall Pillow**进行卸载。

步骤1 在链接中获取Pillow的源码压缩包，以Pillow-7.2.0.tar.gz为例。

步骤2 检查下列Pillow的依赖是否安装，如果没有安装则进行安装。

```
sudo yum install python-devel
sudo yum install zlib-devel
sudo yum install libjpeg-turbo-devel
```

步骤3 解压Pillow源码压缩包并进行编译安装。

```
tar -xvf Pillow-7.2.0.tar.gz
cd Pillow-7.2.0/
python3.7 setup.py build
python3.7 setup.py install
```

----结束

9.1.1.2 准备环境

您需要部署开发环境和运行环境，请参见《**CANN 软件安装指南**》。

- 本文以如下安装路径示例来说明操作步骤，实际操作前，**请务必**获取这些组件的实际安装路径，以便后续操作时使用：
  - 以**HwHiAiUser**用户安装Driver组件，Driver组件的默认安装路径为“${HOME}/Ascend”，在该路径下安装成功后，包括“driver”目录。
  - 以**HwHiAiUser**用户安装cann-toolkit软件包，cann-toolkit软件包的默认安装路径为“${HOME}/Ascend”，在该路径下安装成功后，包括“ascend-toolkit”目录。
  - 以**HwHiAiUser**用户安装cann-nnrt软件包，cann-nnrt软件包的默认安装路径为“${HOME}/Ascend”，在该路径下安装成功后，包括“nnrt”目录。
- pyACL在cann-toolkit软件包、cann-nnrt软件包中均有集成，用户可根据使用场景自行选择其中之一进行安装。cann-toolkit软件包为开发套件包，适用于开发环境。cann-nnrt软件包为离线推理引擎包
- 本文以**HwHiAiUser**用户作为开发环境、运行环境的运行用户为例来说明操作步骤，实际操作前，**请务必**获取开发环境、运行环境的运行用户，以便后续操作时使用。
- 用户使用**export**命令在当前终端窗口下声明环境变量，关闭Shell终端或切换用户时环境变量失效。

9.1.1.3 环境变量配置

在安装完CANN软件包之后，请务必自行配置pyACL相关的环境变量，否则，将无法正常“import acl”。

- 若环境中安装了cann-toolkit软件包：
```
# 以root用户安装toolkit包。
. /usr/local/Ascend/ascend-toolkit/set_env.sh
# 以非root用户安装toolkit包。
. ${HOME}/Ascend/ascend-toolkit/set_env.sh
```
- 若环境中安装了cann-nnrt软件包：
```
# 以root用户安装nnrt包。
. /usr/local/Ascend/nnrt/set_env.sh
# 以非root用户安装nnrt包。
. ${HOME}/Ascend/nnrt/set_env.sh
```

9.1.2 样例介绍

9.1.2.1 实现矩阵-矩阵相加运算

9.1.2.1.1 样例介绍

获取样例

单击**Gitee**，进入Ascend samples开源仓，详细参见README中的“版本说明”下载配套版本的sample包，从“python/level1_single_api/1_acl/4_blas/acl_operator_add”目录下获取acl_operator_add样例。

功能描述

此样例实现了对自定义算子的功能验证，通过将自定义算子转换为单算子离线模型文件，然后通过ACL加载单算子模型文件进行运行。

该实现矩阵-矩阵相加的运算示例为：C = A + B，其中A、B、C都是8*16的矩阵，类型为int32，矩阵加的结果是一个8 * 16的矩阵。

主要接口

主要接口如**表9-2**所示。

表 9-2 主要接口介绍

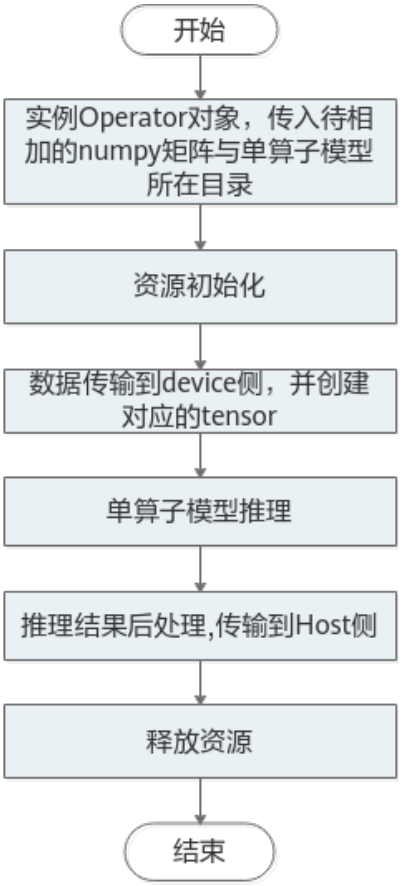
功能	对应ACL模块	ACL 接口函数	功能说明
资源初始化	初始化	acl.init	初始化ACL配置。
	Device管理	acl.rt.set_device	指定用于运算的Device。
	Context管理	acl.rt.create_context	创建Context。
	Stream管理	acl.rt.create_stream	创建Stream。

功能	对应ACL模块	ACL 接口函数	功能说明
	算子加载与执行	acl.op.set_model_dir	加载模型文件的目录。
数据后处理	算子加载与执行	acl.op.create_attr	创建aclopAttr类型的数据。
	--	acl.create_tensor_desc	创建aclTensorDesc类型的数。
	--	acl.get_tensor_desc_size	获取tensor描述占用的空间大小。
	--	acl.create_data_buffer	创建aclDataBuffer类型的数据。
数据交互	内存管理	acl.rt.memcpy	数据传输，Host->Device或Device->Host。
	内存管理	acl.rt.malloc	申请Device上的内存。
	内存管理	acl.rt.malloc_host	申请Host上的内存。
单算子推理	算子加载与执行	acl.op.execute	异步加载并执行指定的算子。
公共模块	--	acl.util.ptr_to_numpy	指针转numpy类型数据。
	--	acl.util.numpy_to_ptr	numpy类型数据转指针。
资源释放	内存管理	acl.rt.free	释放Device上的内存。
	内存管理	acl.rt.free_host	释放Host上的内存。
	Stream管理	acl.rt.destroy_stream	销毁Stream。
	Context管理	acl.rt.destroy_context	销毁Context。
	Device管理	acl.rt.reset_device	复位当前运算的Device，回收Device上的资源。
	去初始化	acl.finalize	实现ACL去初始化。

单算子矩阵相加流程图流程图

单算子矩阵相加流程图流程图如图9-1所示。

图 9-1 单算子矩阵相加流程图



目录结构

如下为模型文件转换后的示例目录结构，“op_models”文件夹是转换后生成的。

```
acl_operator_add
├──src
│  ├── acl_execute_add.py //运行文件。
│  └── constant.py //常量定义。
└── test_data
   ├── config
   │  ├── acl.json //系统初始化的配置文件。
   │  └── add_op.json //矩阵相加算子的描述信息。
   └── op_models
      └── 0_Add_3_2_8_16_3_2_8_16_3_2_8_16.om //矩阵相加算子的模型文件。
```

9.1.2.2 基于 Caffe ResNet-50 网络实现图片分类（视频解码+同步推理）

9.1.2.2.1 样例介绍

获取样例

单击**Gitee**，进入Ascend samples开源仓，详细参见README中的“版本说明”下载配套版本的sample包，从“python/level2_simple_inference/1_classification/vdec_resnet50_classification”目录下获取vdec_resnet50_classification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）。该样例中加载该om文件，将1个h265格式（*.h265）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，并对缩放的图片进行推理，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。

主要接口

主要接口如表9-3所示。

表 9-3 主要接口介绍

功能	对应ACL模块	ACL 接口函数	功能说明
资源初始化	初始化	acl.init	初始化ACL配置。
	Device管理	acl.rt.set_device	指定用于运算的Device。
	Context管理	acl.rt.create_context	创建Context。
	Stream管理	acl.rt.create_stream	创建Stream。
模型初始化	模型加载与执行	acl.mdl.load_from_file	从*.om文件加载模型到device侧。
	数据类型及操作接口	acl.mdl.create_desc	创建模型描述数据类型。
	数据类型及操作接口	acl.mdl.get_desc	获取模型描述数据类型。
数据预处理	媒体数据模块	acl.media.vdec_send_frame	视频解码接口。
	数据类型及操作接口	acl.media.vdec_set_channel_desc系列接口	设置视频处理通道描述信息。
	数据类型及操作接口	acl.media.dvpp_vpc_resize_async	将输入图片缩放到输出图片大小。

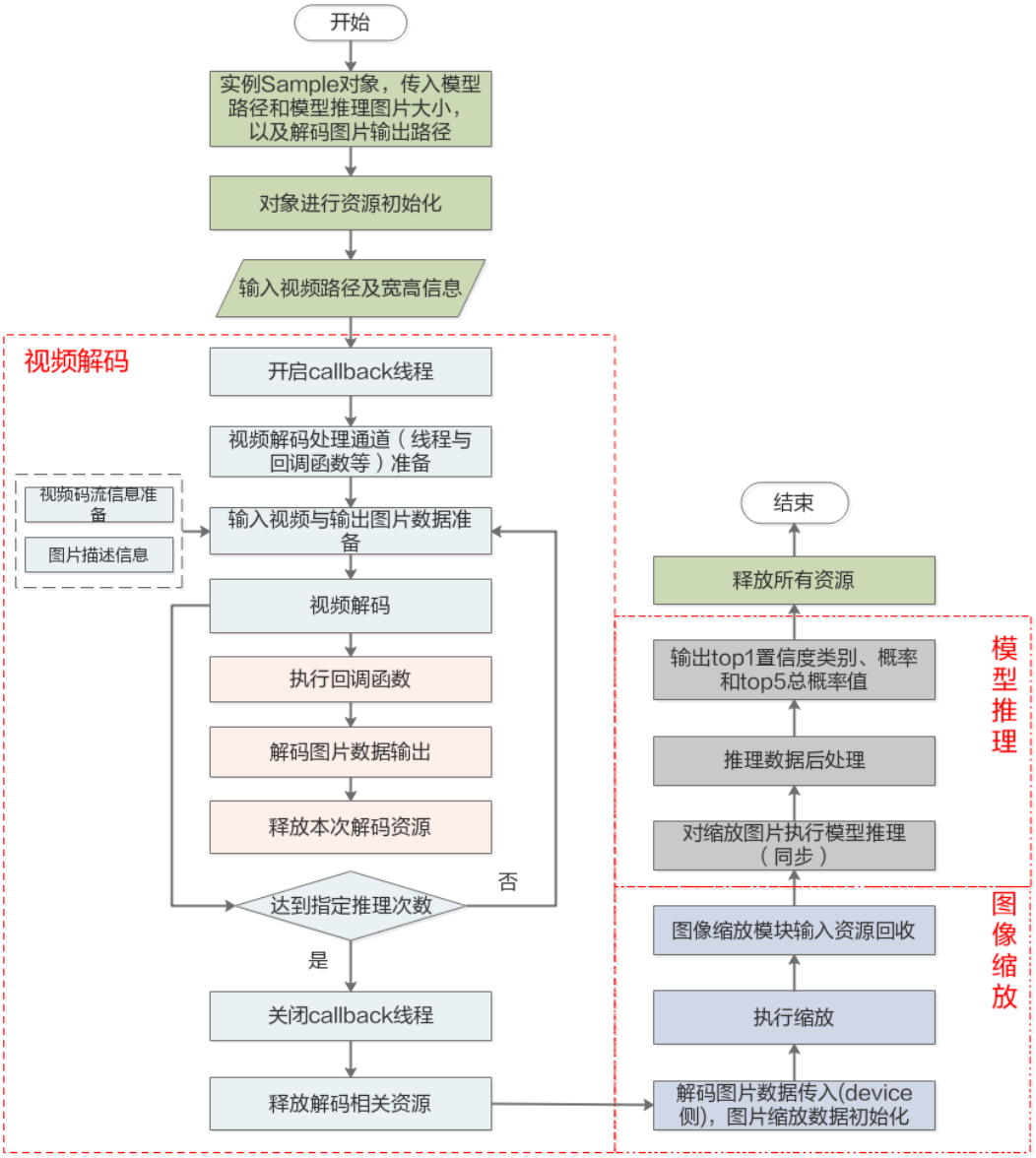
功能	对应ACL模块	ACL 接口函数	功能说明
	数据类型及操作接口	acl.media.dvpp_set_pic_desc系列接口	设置图片描述相关参数。
模型推理	模型加载与执行	acl.mdl.execute	执行模型同步推理。
数据后处理	数据类型及操作接口	acl.op.create_attr	创建aclopAttr类型的数据。
	数据类型及操作接口	acl.create_tensor_desc	创建aclTensorDesc类型的数。
	数据类型及操作接口	acl.get_tensor_desc_size	获取tensor描述占用的空间大小。
	数据类型及操作接口	acl.create_data_buffer	创建aclDataBuffer类型的数据。
数据交互	内存管理	acl.rt.memcpy	数据传输，Host->Device或Device->Host。
	内存管理	acl.media.dvpp_malloc	分配内存给Device侧媒体数据处理时使用。
	内存管理	acl.rt.maclloc	申请Device上的内存。
	内存管理	acl.rt.malloc_host	申请Host上的内存。
单算子推理	算子加载与执行	acl.op.execute	异步加载并执行指定的算子。
公共模块	--	acl.util.ptr_to_numpy	指针转numpy类型数据。
	--	acl.util.numpy_to_ptr	numpy类型数据转指针。

功能	对应ACL模块	ACL 接口函数	功能说明
资源释放	内存管理	acl.rt.free	释放Device上的内存。
	内存管理	acl.media.dvpp_free	通过acl.media.dvpp_malloc接口申请的内存。
	内存管理	acl.rt.free_host	释放Host上的内存。
	模型加载与执行	acl.mdl.unload	卸载模型。
	Stream管理	acl.rt.destroy_stream	销毁Stream。
	Context管理	acl.rt.destroy_context	销毁Context。
	Device管理	acl.rt.reset_device	复位当前运算的Device，回收Device上的资源。
	初始化	acl.finalize	实现ACL去初始化。

视频解码及模型推理流程图

视频解码及模型推理流程图如图9-2所示。

图 9-2 视频解码及模型推理流程图



目录结构

目录结构如下所示。

```
vdec_resnet50_classification
├──src
│  ├── acl_dvpp.py //图片缩放实现文件。
│  ├── acl_model.py //模型推理实现文件。
│  ├── acl_sample.py //运行文件。
│  ├── acl_util.py //工具类函数实现文件。
│  ├── acl_vdec.py //视频解码实现文件。
│  └── constant.py //常量定义。
├── data
│  └── vdec_h265_1frame_rabbit_1280x720.h265 //用户待处理的视频文件，由用户自行获取。
├── caffe_model
│  ├── aipp.cfg
│  ├── resnet50.caffemodel //resnet50模型。
│  └── resnet50.prototxt // resnet50模型的网络文件。
└── model
   └── resnet50_aipp.om //推理模型。
```

9.1.2.3 基于 Caffe ResNet-50 网络实现图片分类（同步推理）

9.1.2.3.1 样例介绍

获取样例

单击**Gitee**，进入Ascend samples开源仓，详细参见README中的“版本说明”下载配套版本的sample包，从“python/level2_simple_inference/1_classification/resnet50_imagenet_classification”目录下获取resnet50_imagenet_classification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（*.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。

主要接口

主要接口如**表9-4**所示。

表 9-4 主要接口介绍

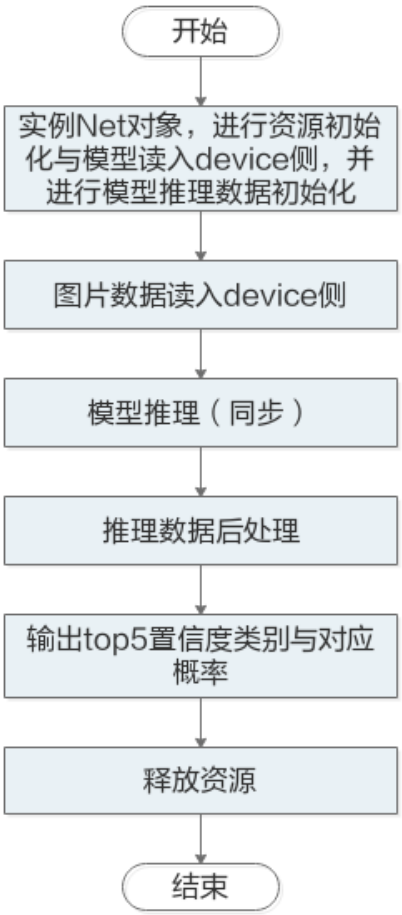
功能	对应ACL模块	ACL 接口函数	功能说明
资源初始化	初始化	acl.init	初始化ACL配置。
	Device管理	acl.rt.set_device	指定用于运算的Device。
	Context管理	acl.rt.create_context	创建Context。
模型初始化	模型加载与执行	acl.mdl.load_from_file	从*.om文件加载模型到device侧。
	数据类型及操作接口	acl.mdl.create_desc	创建模型描述数据类型。
	数据类型及操作接口	acl.mdl.get_desc	获取模型描述数据类型。
模型推理	模型加载与执行	acl.mdl.execute	执行模型同步推理。
数据交互	内存管理	acl.rt.memcpy	数据传输，Host->Device或Device->Host。
	内存管理	acl.rt.malloc	申请Device上的内存。

功能	对应ACL模块	ACL 接口函数	功能说明
	内存管理	acl.rt.malloc_host	申请Host上的内存。
公共模块	--	acl.util.ptr_to_numpy	指针转numpy类型数据。
	--	acl.util.numpy_to_ptr	numpy类型数据转指针。
数据后处理	数据类型及操作接口	acl.mdl.get_dataset_buffer	获取数据集中信息。
	数据类型及操作接口	acl.mdl.get_dataset_num_buffers	获取数据集中信息。
资源释放	内存管理	acl.rt.free	释放Device上的内存。
	内存管理	acl.rt.free_host	释放Host上的内存。
	模型加载与执行	acl.mdl.unload	卸载模型。
	Context管理	acl.rt.destroy_context	销毁Context。
	Device管理	acl.rt.reset_device	复位当前运算的Device，回收Device上的资源。
	初始化	acl.finalize	实现ACL去初始化。

模型同步推理流程图

模型同步推理流程图如图9-3所示。

图 9-3 模型同步推理流程图



目录结构

目录结构如下所示。

```
resnet50_imagenet_classification
├──src
│  ├── acl_net.py //运行文件。
│  └── constant.py //常量定义。
├── data
│  ├── dog1_1024_683.jpg //测试图片数据。
│  └── dog2_1024_683.jpg //测试图片数据。
├── caffe_model
│  ├── resnet50.caffemodel //resnet50模型。
│  └── resnet50.prototxt // resnet50模型的网络文件。
└── model
   └── resnet50.om //转换后的模型文件。
```

9.1.2.4 基于 Caffe ResNet-50 网络实现图片分类（异步推理）

9.1.2.4.1 样例介绍

获取样例

单击**Gitee**，进入Ascend samples开源仓，详细参见README中的“版本说明”下载配套版本的sample包，从“python/level2_simple_inference/1_classification/

resnet50_async_imagenet_classification”目录下获取resnet50_async_imagenet_classification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）。该样例中加载该om文件，对2张jpg（*.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，可通过“--execute_times”参数设置，默认为4次），分别得到n次推理结果后，再对推理结果进行处理，输出top5置信度的类别标识。

主要接口

主要接口如表9-5所示。

表 9-5 主要接口介绍

功能	对应ACL模块	ACL 接口函数	功能说明
资源初始化	初始化	acl.init	初始化ACL配置。
	Device管理	acl.rt.set_device	指定用于运算的Device。
	Context管理	acl.rt.create_context	创建Context。
模型初始化	模型加载与执行	acl.mdl.load_from_file	从*.om文件加载模型到device侧。
	数据类型及操作接口	acl.mdl.create_desc	创建模型描述数据类型。
	数据类型及操作接口	acl.mdl.get_desc	获取模型描述数据类型。
模型推理	模型加载与执行	acl.mdl.execute_async	执行模型异步推理。
同步等待	同步等待	acl.rt.subscribe_report	指定处理Stream上回调函数的线程。
	同步等待	acl.rt.unsubscribe_report	取消线程注册，Stream上的回调函数不再由指定线程处理。

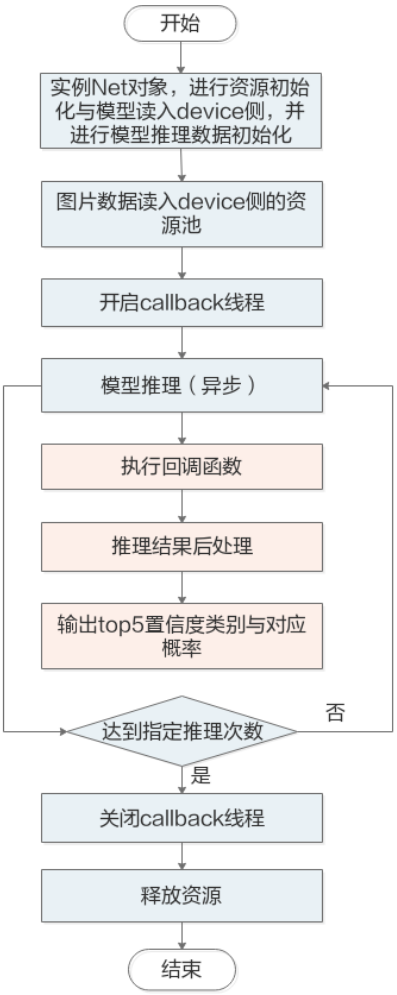
功能	对应ACL模块	ACL 接口函数	功能说明
	同步等待	acl.rt.launch_callback	在Stream的任务队列中增加一个需要在Host/Device上执行的回调函数。
	同步等待	acl.rt.process_report	等待指定时间后，触发回调处理。
数据交互	内存管理	acl.rt.memcpy	数据传输，Host->Device或Device->Host。
	内存管理	acl.rt.malloc	申请Device上的内存。
	内存管理	acl.rt.malloc_host	申请Host上的内存。
公共模块	--	acl.util.ptr_to_numpy	指针转numpy类型数据。
	--	acl.util.numpy_to_ptr	numpy类型数据转指针。
	--	acl.util.start_thread	启动一个回调函数线程。
	--	acl.util.stop_thread	回收一个回调函数线程。
数据后处理	数据类型及操作接	acl.mdl.get_dataset_buffer	获取数据集中信息。
	数据类型及操作接口	acl.mdl.get_dataset_num_buffers	获取数据集中信息。
资源释放	内存管理	acl.rt.free	释放Device上的内存。
	内存管理	acl.rt.free_host	释放Host上的内存。
	模型加载与执行	acl.mdl.unload	卸载模型。
	Context管理	acl.rt.destroy_context	销毁Context。

功能	对应ACL模块	ACL 接口函数	功能说明
	Device管理	acl.rt.reset_device	复位当前运算的Device，回收Device上的资源。
	初始化	acl.finalize	实现ACL去初始化。

模型异步推理流程图

模型异步推理流程图如图9-4所示。

图 9-4 模型异步推理流程图



目录结构

目录结构如下所示。

```
resnet50_async_imagenet_classification
├──src
```

```
│   ├── acl_net.py //运行文件。
│   └── constant.py //常量定义。
├── data
│   ├── dog1_1024_683.jpg //测试图片数据。
│   └── dog2_1024_683.jpg //测试图片数据。
├── caffe_model
│   ├── resnet50.caffemodel //resnet50模型。
│   └── resnet50.prototxt // resnet50模型的网络文件。
└── model
    └── resnet50.om //转换后的模型文件。
```

9.1.2.5 媒体数据处理（视频编码）

9.1.2.5.1 样例介绍

获取样例

单击**Gitee**，进入Ascend samples开源仓，详细参见README中的“版本说明”下载配套版本的sample包，从“python/level2_simple_inference/0_data_process/venc_image”目录下获取venc_image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，由代码中的“venc_cnt”参数来控制，默认为16次）。

目录结构

如下为模型文件转换后的示例目录结构，model文件夹是转换后生成的。

```
venc_image
├──src
│   └── acl_venc.py //视频编码实现文件
└── data
    └── dvpp_venc_128x128_nv12.yuv //用户待处理的图片文件
```

10 pyACL API 参考

废弃接口/返回码列表
系统配置
公共接口
Util模块
Runtime（rt）
模型加载与执行（mdl）
媒体数据处理V1
媒体数据处理V2
算子加载与执行（op）
CBLAS接口（blas）
算子编译
特征向量检索
Profiling数据采集
数据类型及其操作接口

10.1 废弃接口/返回码列表

接口

- **acl.op.execute**接口
  此接口后续版本会废弃，请使用**acl.op.execute_v2**接口。
- **acl.get_tensor_desc_dim**接口
  此接口后续版本会废弃，请使用**acl.get_tensor_desc_dim_v2**接口。
- **acl.get_data_buffer_size**接口
  此接口后续版本会废弃，请使用**acl.get_data_buffer_size_v2**接口。

- **acl.set_tensor_storage_shape**接口
  此接口后续版本会废弃，请使用**acl.set_tensor_shape**接口。
- **acl.set_tensor_storage_format**接口
  此接口后续版本会废弃，请使用**acl.set_tensor_format**接口。
- **acl.rt.query_event**接口
  此接口后续版本会废弃，请使用**acl.rt.query_event_status**接口。
- **acl.util.numpy_to_ptr**接口
  此接口后续版本会废弃，请使用**acl.util.bytes_to_ptr**接口。
- **acl.util.numpy_contiguous_to_ptr**接口
  此接口后续版本会废弃，请使用**acl.util.bytes_to_ptr**接口。
- **acl.util.ptr_to_numpy**接口
  此接口后续版本会废弃，请使用**acl.util.ptr_to_bytes**接口。

返回码

- **ACL_ERROR_NONE**
  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。

10.2 系统配置

10.2.1 函数：init

C函数原型	aclError aclInit(const char *configPath)

Python函数	**ret = acl.init(config_path)**
函数功能	pyACL初始化函数，同步接口。
输入说明	**config_path**：str，配置文件所在的路径，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，可直接调用acl.init接口不传入参数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3种配置：模型Dump配置、单算子Dump配置、溢出算子Dump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《**精度比对工具使用指南**》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）”。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。 ● 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）**。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	● 一个进程内只能调用一次acl.init接口。 ● 使用pyACL接口开发应用时，必须先调用acl.init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。
相关接口	pyACL还提供了其它使能Dump或Profiling的接口，如下，与acl.init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**acl.mdl.init_dump**、**acl.mdl.set_dump**、**acl.mdl.finalize_dump**。 ● 获取Profiling数据，参见**Profiling配置**。
参考资源	接口调用示例，参见**4.3 pyACL初始化与去初始化**。

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7.12 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acl.op.set_model_dir接口加载指定目录下的单算子模型或调用acl.op.load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。
- pyACL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"
}
```

10.2.2 函数：finalize

C函数原型	**aclError aclFinalize()**
Python函数	**ret = acl.finalize()**
函数功能	pyACL去初始化函数，用于释放进程内的pyACL相关资源，同步接口。

输入说明	无
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	应用的进程退出前，应显式调用该接口实现pyACL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acl.finalize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。
注意事项	无
参考资源	接口调用示例，参见4.3 pyACL初始化与去初始化。

10.2.3 函数：get_version

C函数原型	**aclError aclrtGetVersion(int32_t *majorVersion, int32_t *minorVersion, int32_t *patchVersion)**
Python函数	**major_version, minor_version, patch_version, ret = acl.get_version()**
函数功能	查询接口版本号，pyACL接口版本号命名可以采用：A.B.C模式，其中C表示bug修复，B表示新增接口，A表示有不兼容修改。同步接口。
输入说明	无。
返回值说明	**major_version**：int，从1开始，如果出现接口的不兼容变更时，加1。 **minor_version**：int，从0开始，按照迭代周期，有新增接口时加1。 **patch_version**：int，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”。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.2.4 函数：get_soc_name

C函数原型	**const char *aclrtGetSocName()**
Python函数	**name =acl.get_soc_name()**
函数功能	查询当前运行环境的芯片版本，同步接口。
输入说明	无
返回值说明	**name**：str，返回芯片版本字符串。如果通过该接口获取芯片版本失败，则返回None。
约束说明	无
注意事项	无

10.2.5 函数：get_recent_err_msg

C函数原型	**const char *aclGetRecentErrMsg()**
Python函数	**msg =acl.get_recent_err_msg()**
函数功能	获取并清空与本接口在同一个线程中的其它pyACL接口调用失败时的错误描述信息。同步接口。
输入说明	无
返回值说明	**msg**：str，返回错误描述字符串。 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None。
约束说明	● 建议在每次调用pyACL接口失败时都调用acl.get_recent_err_msg接口，以便获取调用pyACL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acl.get_recent_err_msg接口后，只有最后一次调用acl.get_recent_err_msg()接口返回的错误描述字符串的指针有效，之前acl.get_recent_err_msg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pyACL接口失败时调用acl.get_recent_err_msg接口，则可能导致错误信息堆积、丢失。
注意事项	无

10.2.6 函数：set_device_sat_mode

C函数原型	**aclError aclrtSetDeviceSatMode(aclrtFloatOverflowMode mode)**
Python函数	**ret = acl.rt.set_device_sat_mode(mode)**
函数功能	设置当前Context对应的Device的浮点计算结果输出模式，同步接口。 Atlas 200/500 A2推理产品，当前版本不支持该接口。
输入说明	**mode**：int 设置浮点计算结果输出模式。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
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。
注意事项	无

10.2.7 函数：get_device_sat_mode

C函数原型	**aclError aclrtGetDeviceSatMode(aclrtFloatOverflowMode *mode)**
Python函数	**mode, ret = acl.rt.get_device_sat_mode()**
函数功能	查询当前Context对应的Device的浮点计算结果输出模式，同步接口。 Atlas 200/500 A2推理产品，当前版本不支持该接口。
输入说明	无
返回值说明	**mode**：int 获取浮点计算结果输出模式。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.3 公共接口

10.3.1 函数：float16_to_float

C函数原型	**float aclFloat16ToFloat(aclFloat16 value)**
Python函数	**output = acl.float16_to_float(value)**
函数功能	将**aclFloat16**类型的数据转换为float（指float32）类型的数据。
输入说明	**value**：int，待转化的数据，**aclFloat16**类型的数据。
返回值说明	**output**：float，浮点数类型。
约束说明	无
注意事项	无

10.3.2 函数：float_to_float16

C函数原型	**aclFloat16 aclFloatToFloat16(float value)**
Python函数	**output = acl.float_to_float16(value)**
函数功能	将float（指float32）类型的数据转换为**aclFloat16**类型的数据。
输入说明	**value**：float，待转化的数据。
返回值说明	**output**：int，**aclFloat16**类型的数据。
约束说明	无
注意事项	无

10.3.3 函数：create_data_buffer

C函数原型	**aclDataBuffer *aclCreateDataBuffer(void *data, size_t size)**
Python函数	**output = acl.create_data_buffer(data, size)**
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。同步接口。
输入说明	**data**：int，表述存储aclDataBuffer数据的指针地址。data参数支持传入0，表示创建一个空的数据类型，此时size参数值无效。该地址对象对应的数据内存需由用户自行管理，调用**acl.rt.malloc**接口申请内存，调用**acl.rt.free**接口释放内存。 **size**：int，内存大小。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	**output**：int， 输出aclDataBuffer的指针地址。
约束说明	无

注意事项	无

10.3.4 函数：destroy_data_buffer

C函数原型	**aclError aclDestroyDataBuffer(const aclDataBuffer *dataBuffer)**
Python函数	**ret = acl.destroy_data_buffer(data_buffer)**
函数功能	销毁aclDataBuffer类型的数据。 此处仅销毁aclDataBuffer类型的数据，调用**acl.create_data_buffer**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。
输入说明	**data_buffer**：int，表述aclDataBuffer的指针地址。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无

10.3.5 函数：get_data_buffer_addr

C函数原型	**void *aclGetDataBufferAddr(const aclDataBuffer *dataBuffer)**
Python函数	**output = acl.get_data_buffer_addr(data_buffer)**
函数功能	获取aclDataBuffer类型中的数据的内存地址。
输入说明	**data_buffer**：int，表述data_buffer的指针地址。
返回值说明	**output**：int，aclDataBuffer类型中的数据的指针地址。
约束说明	无
注意事项	无

10.3.6 函数：get_data_buffer_size

须知

此接口后续版本会废弃，请使用**acl.get_data_buffer_size_v2**接口。

C函数原型	**size_t aclGetDataBufferSize(const aclDataBuffer *dataBuffer)**
Python函数	**size = acl.get_data_buffer_size(data_buffer)**
函数功能	获取aclDataBuffer类型中数据的内存大小，单位Byte。
输入说明	**data_buffer**: int，data_buffer的指针地址。
返回值说明	**size**：int，aclDataBuffer类型中数据的内存大小。
约束说明	无
注意事项	无

10.3.7 函数：get_data_buffer_size_v2

C函数原型	**size_t aclGetDataBufferSizeV2(const aclDataBuffer *dataBuffer)**
Python函数	**size = acl.get_data_buffer_size_v2(data_buffer)**
函数功能	获取aclDataBuffer类型中数据的内存大小，单位Byte。
输入说明	**data_buffer**：int，data_buffer的指针地址。
返回值说明	**size**：int，aclDataBuffer类型中数据的内存大小。
约束说明	无
注意事项	无

10.3.8 函数：update_data_buffer

C函数原型	**aclError aclUpdateDataBuffer(aclDataBuffer *dataBuffer, void *data, size_t size)**
Python函数	**ret = acl.update_data_buffer(data_buffer, data, size)**
函数功能	更新aclDataBuffer中数据的内存及大小。同步接口。 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。

输入说明	**data_buffer**：int，aclDataBuffer类型的指针地址。 需提前调用**acl.create_data_buffer**接口创建aclDataBuffer类型的数据。 该内存需由用户自行管理，调用**acl.rt.malloc**接口/**acl.rt.free**接口申请/释放内存，或调用**acl.rt.malloc_host**接口/**acl.rt.free_host**接口申请/释放内存。 **data**：int，存放数据内存地址的指针地址。 **size**：int，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。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无

10.3.9 函数：data_type_size

C函数原型	**size_t aclDataTypeSize(aclDataType dataType)**
Python函数	**size = acl.data_type_size(data_type)**
函数功能	获取aclDataType数据的大小，单位Byte，同步接口。
输入说明	**data_type**：int，指定要获取大小的aclDataType数据。
返回值说明	**size**：int，aclDataType数据的大小，单位Byte。
约束说明	无
注意事项	无

10.3.10 函数：create_tensor_desc

C函数原型	**aclTensorDesc *aclCreateTensorDesc(aclDataType dataType, int numDims, const int64_t *dims, aclFormat format)**
Python函数	**tensor_desc = acl.create_tensor_desc(data_type, dims, format)**
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。同步接口。 如需销毁aclTensorDesc类型的数据，请参见**acl.destroy_tensor_desc**。

输入说明	**data_type**：int，表述tensor描述的数据类型。 **dims**：int list，tensor描述指定维度的大小。dims是一个int的list，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**：int，tensor描述的format。
返回值说明	**tensor_desc**：int，创建tensor描述的地址对象。
约束说明	无
注意事项	无

10.3.11 函数：destroy_tensor_desc

C函数原型	**void aclDestroyTensorDesc(const aclTensorDesc *desc)**
Python函数	**acl.destroy_tensor_desc(desc)**
函数功能	销毁aclTensorDesc类型的数据，同步接口。
输入说明	**desc**：int，待销毁的aclTensorDesc类型的指针地址。
返回值说明	无
约束说明	无
注意事项	无

10.3.12 函数：get_tensor_desc_type

C函数原型	**aclDataType aclGetTensorDescType(const aclTensorDesc *desc)**
Python函数	**type = acl.get_tensor_desc_type(desc)**
函数功能	获取tensor描述中的数据类型，同步接口。
输入说明	**desc**：int，aclTensorDesc类型的指针地址。 需提前调用**acl.create_tensor_desc**接口创建aclTensorDesc类型。
返回值说明	**Type**：int，返回指定tensor描述的数据类型。
约束说明	无
注意事项	无

10.3.13 函数：get_tensor_desc_format

C函数原型	**aclFormat aclGetTensorDescFormat(const aclTensorDesc *desc)**
Python函数	**format = acl.get_tensor_desc_format(desc)**
函数功能	获取tensor描述中的format，同步接口。
输入说明	**desc**：int，aclTensorDesc类型的指针地址。 需提前调用**acl.create_tensor_desc**接口创建aclTensorDesc类型。
返回值说明	**format**：int， 返回指定tensor描述的格式。
约束说明	无
注意事项	无

10.3.14 函数：get_tensor_desc_size

C函数原型	**size_t aclGetTensorDescSize(const aclTensorDesc *desc)**
Python函数	**size = acl.get_tensor_desc_size(desc)**
函数功能	获取tensor描述占用的空间大小，同步接口。
输入说明	**desc**：int，aclTensorDesc类型的指针地址。 需提前调用**acl.create_tensor_desc**接口创建aclTensorDesc类型。
返回值说明	**size**：int，tensor描述占用的空间大小。
约束说明	无
注意事项	无

10.3.15 函数：get_tensor_desc_element_count

C函数原型	**size_t aclGetTensorDescElementCount(const aclTensorDesc *desc)**
Python函数	**size = acl.get_tensor_desc_element_count(desc)**
函数功能	获取tensor描述中的元素个数，同步接口。
输入说明	**desc**：int，aclTensorDesc类型的指针地址。 需提前调用**acl.create_tensor_desc**接口创建aclTensorDesc类型。
返回值说明	**size**：int，获取tensor描述中的元素个数。
约束说明	无

注意事项	无

10.3.16 函数：get_tensor_desc_num_dims

C函数原型	**size_t aclGetTensorDescNumDims(const aclTensorDesc *desc)**
Python函数	**size = acl.get_tensor_desc_num_dims(desc)**
函数功能	获取tensor描述中shape的维度个数，同步接口。
输入说明	**desc**：int，aclTensorDesc类型的指针地址。 需提前调用**acl.create_tensor_desc**接口创建aclTensorDesc类型。
返回值说明	**size**：int，获取tensor描述中shape的维度个数。 若返回**ACL_UNKNOWN_RANK（0xFFFFFFFFFFFFFFFE）**表示动态Shape场景下维度个数未知。
约束说明	无
注意事项	无

10.3.17 函数：get_tensor_desc_dim

须知

此接口后续版本会废弃，请使用**acl.get_tensor_desc_dim_v2**接口。

C函数原型	**int64_t aclGetTensorDescDim(const aclTensorDesc *desc, size_t index)**
Python函数	**num = acl.get_tensor_desc_dim(desc, index)**
函数功能	获取tensor描述中指定维度的大小，同步接口。
输入说明	**desc**：int，aclTensorDesc类型的指针地址。 需提前调用**acl.create_tensor_desc**接口创建aclTensorDesc类型。 **index**：int，指定获取第几个维度的大小，index值从0开始。 用户调用acl.get_tensor_desc_dim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	**num**：int，获取tensor描述中指定维度的大小。
约束说明	无
注意事项	无

10.3.18 函数：get_tensor_desc_dim_v2

C函数原型	**aclError aclGetTensorDescDimV2(const aclTensorDesc *desc, size_t index, int64_t *dimSize)**
Python函数	**dim_size, ret = acl.get_tensor_desc_dim_v2(desc, index)**
函数功能	获取tensor描述中指定维度的大小。
输入说明	**desc**：int，aclTensorDesc类型的指针地址。 需提前调用**acl.create_tensor_desc**接口创建aclTensorDesc类型。 **index**：int，指定获取第几个维度的大小，index值从0开始。 用户调用**acl.get_tensor_desc_num_dims**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	**dim_size**：int，tensor描述中index指定维度的大小。当返回的dim_size参数值为“-1”时，表示维度的大小是动态的。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	当acl.get_tensor_desc_num_dims接口的返回值为**ACL_UNKNOWN_RANK（0xFFFFFFFFFFFFFFFE）**时，表示动态Shape场景下维度个数未知，则不能调用acl.get_tensor_desc_dim_v2接口获取指定维度的范围。
注意事项	无

10.3.19 函数：get_tensor_desc_dim_range

C函数原型	**aclError aclGetTensorDescDimRange(const aclTensorDesc *desc, size_t index, size_t dimRangeNum, int64_t *dimRange)**
Python函数	**dim_range, ret = acl.get_tensor_desc_dim_range(desc, index, dim_range_num)**
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。
输入说明	**desc**：int，aclTensorDesc类型的指针地址。 需提前调用**acl.create_tensor_desc**接口创建aclTensorDesc类型。 **index**：int，指定获取第几个维度的大小，index值从0开始。 用户调用acl.get_tensor_desc_num_dims接口获取shape维度个数，这个Index的取值范围：[0, (*shape维度个数*-1)]。 **dim_range_num**：int，dim_range的长度，该值必须大于或等于2。

返回值说明	**dim_range**：list，tensor描述中index指定维度的Shape范围。 ● 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dim_range数组的值为[1,-1]时，表示全Shape范围。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	当acl.get_tensor_desc_num_dims接口的返回值为**ACL_UNKNOWN_RANK（0xFFFFFFFFFFFFFFFE）**时，表示动态Shape场景下维度个数未知，则不能调用acl.get_tensor_desc_dim_range接口获取指定维度的范围。
注意事项	无

10.3.20 函数：set_tensor_desc_name

C函数原型	**void aclSetTensorDescName(aclTensorDesc *desc, const char *name)**
Python函数	**acl.set_tensor_desc_name(desc, name)**
函数功能	设置TensorDesc的名称，同步接口。
输入说明	**desc**：int，aclTensorDesc类型的指针地址。 需提前调用acl.create_tensor_desc接口创建aclTensorDesc类型。 **name**：str，tensor描述的名称。
返回值说明	无
约束说明	无
注意事项	无

10.3.21 函数：get_tensor_desc_name

C函数原型	**const char *aclGetTensorDescName(aclTensorDesc *desc)**
Python函数	**name = acl.get_tensor_desc_name(desc)**
函数功能	获取tensor描述的名称，同步接口。
输入说明	**desc**：int，aclTensorDesc类型的指针地址。 需提前调用acl.create_tensor_desc接口创建aclTensorDesc类型。
返回值说明	**name**：str，tensor描述的名称。
约束说明	无

注意事项	无

10.3.22 函数：set_tensor_storage_shape

须知

此接口后续版本会废弃，请使用acl.set_tensor_shape接口。

C函数原型	**aclError aclSetTensorStorageShape(aclTensorDesc *desc, int numDims, const int64_t *dims)**
Python函数	**ret = acl.set_tensor_storage_shape(desc, dims)**
函数功能	调用acl.create_tensor_desc接口创建tensor描述信息后，可通过acl.set_tensor_storage_shape设置tensor的实际dims信息，当前主要用于pytorch场景下，同步接口。
输入说明	**desc**：int，aclTensorDesc类型的指针地址。 需提前调用acl.create_tensor_desc接口创建aclTensorDesc类型。 **dims**：list，要设置的dims维度数。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.3.23 函数：set_tensor_storage_format

须知

此接口后续版本会废弃，请使用acl.set_tensor_format接口。

C函数原型	**aclError aclSetTensorStorageFormat(aclTensorDesc *desc, aclFormat format)**
Python函数	**ret = acl.set_tensor_storage_format(desc, format)**
函数功能	调用acl.create_tensor_desc接口创建tensor描述信息后，可通过acl.set_tensor_storage_format设置tensor的实际format信息，当前主要用于pytorch场景下，同步接口。

输入说明	**desc**：int，aclTensorDesc类型的指针。 需提前调用**acl.create_tensor_desc**接口创建aclTensorDesc类型。 **format**：int， 指定tensor描述的格式。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无

10.3.24 函数：set_tensor_origin_shape

C函数原型	**aclError aclSetTensorOriginShape(aclTensorDesc *desc, int numDims, const int64_t *dims)**
Python函数	**ret = acl.set_tensor_origin_shape(desc, dims)**
函数功能	调用**acl.create_tensor_desc**接口创建tensor描述信息后，可通过acl.set_tensor_origin_shape设置tensor的原始dims信息，当前主要用于pytorch场景下，同步接口。 **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。
输入说明	**desc**：int，tensor描述数据的指针。 **dims**：list，要设置的dims。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无

10.3.25 函数：set_tensor_origin_format

C函数原型	**aclError aclSetTensorOriginFormat(aclTensorDesc *desc, aclFormat format)**
Python函数	**ret = acl.set_tensor_origin_format(desc, format)**

函数功能	调用**acl.create_tensor_desc**接口创建tensor描述信息后，可通过acl.set_tensor_origin_format设置tensor的原始format信息，当前主要用于pytorch场景下，同步接口。 **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。
输入说明	**desc**：int，tensor描述地址对象。 **format**：int， 指定tensor描述的格式。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无

10.3.26 函数：set_tensor_shape

C函数原型	**aclError aclSetTensorShape(aclTensorDesc *desc, int numDims, const int64_t *dims)**
Python函数	**ret = acl.set_tensor_shape(desc, dims)**
函数功能	调用**acl.create_tensor_desc**接口创建tensor描述信息后，可通过acl.set_tensor_shape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，同步接口。 **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。
输入说明	**desc**：int，tensor描述数据的指针。 **dims**：list，要设置的dims。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无

10.3.27 函数：set_tensor_format

C函数原型	**aclError aclSetTensorFormat(aclTensorDesc *desc, aclFormat format)**
Python函数	**ret = acl.set_tensor_format(desc, format)**
函数功能	调用acl.create_tensor_desc接口创建tensor描述信息后，可通过acl.set_tensor_format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，同步接口。 **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。
输入说明	**desc**：int，tensor描述地址对象。 **format**：int， 指定tensor描述的格式。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.3.28 函数：set_tensor_const

C函数原型	**aclError aclSetTensorConst(aclTensorDesc *desc, void *dataBuffer, size_t length)**
Python函数	**ret = acl.set_tensor_const(desc, dataBuffer, length)**
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。同步接口。
输入说明	**desc**：aclTensorDesc类型的指针。需提前调用acl.create_tensor_desc接口创建aclTensorDesc类型。 **dataBuffer**：存放constant输入tensor数据的内存地址，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。内存申请接口请参见10.5.5 内存管理。 ● 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。 **length**：内存地址的长度，单位是Byte。

返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.3.29 函数：set_tensor_shape_range

C函数原型	**aclError aclSetTensorShapeRange(aclTensorDesc* desc, size_t dimsCount, int64_t dimsRange[][ACL_TENSOR_SHAPE_RANGE_NUM])**
Python函数	**ret = acl.set_tensor_shape_range(desc, dims_range)**
函数功能	调用acl.create_tensor_desc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。同步接口。
输入说明	**desc**：int，tensor描述地址对象。 **dims_range**：list list，dims_range为每个维度的取值范围，用二维list表示范围。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	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。

10.3.30 函数：set_tensor_dynamic_input

C函数原型	**aclError aclSetTensorDynamicInput(aclTensorDesc *desc, const char *dynamicInputName)**
Python函数	**ret = acl.set_tensor_dynamic_input(desc, dynamic_input_name)**
函数功能	调用acl.create_tensor_desc接口创建tensor描述信息后，可通过acl.set_tensor_dynamic_input接口设置算子动态输入的动态输入属性的名称。

输入说明	**desc**：int，tensor描述地址对象。 **dynamic_input_name**：int，设置desc中的动态输入属性的名称。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.3.31 函数：get_tensor_desc_by_index

C函数原型	**aclTensorDesc *aclGetTensorDescByIndex(aclTensorDesc *desc, size_t index)**
Python函数	**tensor_desc = acl.get_tensor_desc_by_index(desc, index)**
函数功能	获取算子的指定Index的tensor描述信息。
输入说明	**desc**：int，tensor描述地址对象。调用acl.mdl.create_and_get_op_desc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**：int，指定获取第几个维度的大小，index值从0开始。 用户调用acl.mdl.create_and_get_op_desc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	**tensor_desc** ：返回指定输入/输出的tensor描述地址对象。
约束说明	无
注意事项	无

10.3.32 函数：get_tensor_desc_address

C函数原型	**void *aclGetTensorDescAddress(const aclTensorDesc *desc)**
Python函数	**addr = acl.get_tensor_desc_address(desc)**
函数功能	获取指定算子输入/输出的tensor数据的内存地址。
输入说明	**desc**：int，tensor描述地址对象。调用acl.get_tensor_desc_by_index接口获取算子所有输入/输出的tensor描述，作为本接口的输入。
返回值说明	**addr**：返回指定算子输入/输出的tensor数据的内存地址。

约束说明	无
注意事项	无

10.3.33 函数：set_tensor_place_ment

C函数原型	**aclError aclSetTensorPlaceMent(aclTensorDesc *desc, aclMemType memType)**
Python函数	**ret = acl.set_tensor_place_ment(desc, mem_type)**
函数功能	设置tensor描述的placement属性，用于标识存放tensor数据的内存为Host内存或Device内存。同步接口。
输入说明	**desc**：int，tensor描述地址对象。 需提前调用**acl.create_tensor_desc**接口创建算子输入/输出的tensor描述，作为本接口的输入。 **mem_type**：int，指定placement属性值，其取值参考**aclMemType**。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无

10.3.34 函数：set_tensor_value_range

C函数原型	**aclError aclSetTensorValueRange(aclTensorDesc* desc, size_t valueCount, int64_t valueRange[] [ACL_TENSOR_VALUE_RANGE_NUM])**
Python函数	**ret = acl.set_tensor_value_range(desc, value_range)**
函数功能	调用**acl.create_tensor_desc**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。同步接口。 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]]），pyACL才能正常编译、执行算子。
输入说明	**desc** : int aclTensorDesc类型的指针。需提前调用**10.3.10 函数：create_tensor_desc**接口创建aclTensorDesc类型。 **value_range**: list value_range为每个数据值的范围，用二维数组表示范围。 ACL_TENSOR_VALUE_RANGE_NUM = 2。

返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.3.35 函数：trans_tensor_desc_format

C函数原型	**aclError aclTransTensorDescFormat(const aclTensorDesc *srcDesc, aclFormat dstFormat, aclTensorDesc **dstDesc)**
Python函数	**dstDesc, ret = acl.trans_tensor_desc_format(srcDesc, dstFormat)**
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。当前版本不支持该接口，预留接口。
输入说明	**srcDesc**：int，输入源aclTensorDesc数据的指针地址。需提前调用10.3.10 函数：create_tensor_desc接口创建aclTensorDesc类型。 **dstFormat**：int 需要设置的目标format
输出说明	**dstDesc**：int，输出“目标aclTensorDesc数据指针”的指针地址。需提前调用aclCreateTensorDesc接口创建aclTensorDesc类型。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.3.36 函数：app_log

C函数原型	**ACL_APP_LOG(aclLogLevel logLevel, const char *message)**
Python函数	**acl.app_log(log_level, message)**
函数功能	将日志记录到系统相应的日志文件中。

输入说明	**log_level**：int，设定日志级别。 ● 0：DEBUG ● 1：INFO ● 2：WARNING ● 3：ERROR **message**：str，记录的日志信息。
返回值说明	无
约束说明	无
注意事项	无

10.4 Util 模块

10.4.1 函数：numpy_to_ptr

C函数原型	无
Python函数	**ptr = acl.util.numpy_to_ptr(data)**
函数功能	将numpy数组转换成为void*数据，可以将转换好的数据传递给C函数直接使用。
输入说明	**data**：numpy类型数据。
返回值说明	**ptr**：int，可以被C语言访问的数据。
约束说明	访问的是int类型的数据，需要将void*转换为int*。

注意事项	• 该函数输入的生命周期需要大于输出的生命周期，否则可能会产生未定义行为。 • 若传入的numpy数组在内存上是非行连续的，在接口调用时会打印如下提示信息： Warning：The input ndarray is discontiguous. Please use acl.util.numpy_contiguous_to_ptr instead. 请使用**acl.util.numpy_contiguous_to_ptr**接口获取numpy数组的地址对象，否则在使用**acl.rt.memcpy**拷贝数据时，由于**acl.rt.memcpy**拷贝的是连续内存地址，会导致拷贝的数据和目标拷贝数据不一致的问题。 • 可以通过numpy数组的flags属性来查看numpy数组在内存上是否连续： print(data.flags) 得到结果如下： **C_CONTIGUOUS : True** **F_CONTIGUOUS : False** OWNDATA : True WRITEABLE : True ALIGNED : True WRITEBACKIFCOPY : False UPDATEIFCOPY : False 其中“C_CONTIGUOUS：True”，则说明numpy数组在内存上行连续。“F_CONTIGUOUS : False”，则说明numpy数组在内存上列不连续。 • 该接口即将废弃，建议使用**acl.util.bytes_to_ptr**接口。 • 修改示例如下： np_ptr = acl.util.numpy_to_ptr(np_arr_in) 修改后使用： bytes_in = np_arr_in.tobytes() bytes_ptr = acl.util.bytes_to_ptr(bytes_in) • 若要继续使用该接口，需要运行环境为python≥3.8且numpy≥1.22.0。

10.4.2 函数：numpy_contiguous_to_ptr

C函数原型	无
Python函数	**ptr, data_out = acl.util.numpy_contiguous_to_ptr(data_in)**
函数功能	将numpy数组转换成为void*数据，可以将转换好的数据传递给C函数直接使用。 若传入的numpy数组在内存上是非行连续的，先将其内存转行连续，然后将numpy数组转换成为void*数据。
输入说明	**data_in**：numpy类型数据。
返回值说明	**ptr**：int，可以被C语言访问的数据。 **data_out**：numpy类型数据，若传入的numpy数组“data_in”内存非行连续，则返回值“data_out”即为其内存转行连续后的numpy数组。

约束说明	访问的是int类型的数据，需要将void*转换为int*。

注意事项	● 该函数输入的生命周期需要大于输出的生命周期，否则可能会产生未定义行为。
	● 该接口即将废弃，建议使用**acl.util.bytes_to_ptr**接口。
	● 修改示例如下：
	`np_ptr, data_out = acl.util.numpy_contiguous_to_ptr(np_arr_in)`
	修改后使用：
	`bytes_in = np_arr_in.tobytes()`
	`bytes_ptr = acl.util.bytes_to_ptr(bytes_in)`
	● 若要继续使用该接口，需要运行环境为python≥3.8且numpy≥1.22.0。

10.4.3 函数：ptr_to_numpy

C函数原型	无
Python函数	**output = acl.util.ptr_to_numpy(ptr, shape, type)**
函数功能	将void*数据转换为numpy数组，可以使python代码直接访问。
输入说明	**ptr**：int，C中的指针，是能够访问的数据的首地址。 **shape**：tuple，需要构造的numpy的shape。 **type**：int，表示ptr中数据的数据类型。 下面举例一些常用的类型（未列出类型详见**Numpy手册**，NumPy C-API中的数据类型API，以下列出类型以手册中数据为准）： ● 0：NPY_BOOL ● 1：NPY_BYTE，NPY_INT8 ● 2：NPY_UINT8 ● 3：NPY_SHORT，NPY_INT16 ● 4：NPY_USHORT，NPY_UINT16 ● 5：NPY_INT，NPY_INT32 ● 6：NPY_UINT，NPY_UINT32 ● 7：NPY_INT64 ● 8：NPY_UINT64 ● 9：NPY_LONGLONG ● 10：NPY_ULONGLONG ● 11：NPY_FLOAT32 ● 12：NPY_DOUBLE ● 23：NPY_HALF，NPY_FLOAT16
返回值说明	**output**：numpy类型。

约束说明	无

注意事项	● 该接口即将废弃，建议使用**acl.util.ptr_to_bytes**接口。
	● 修改示例如下：
	`np_arr_out = acl.util.ptr_to_numpy(host_ptr, np_arr_in.shape, NPY_INT32)`
	修改后使用：
	`bytes_out = acl.util.ptr_to_bytes(ptr, size)`
	`np_arr_out = np.frombuffer(bytes_out, dtype=np.int32).reshape(np_arr_in.shape)`
	● 若要继续使用该接口，需要运行环境为python≥3.8且numpy≥1.22.0。

10.4.4 函数：bytes_to_ptr

C函数原型	无
Python函数	**ptr = acl.util.bytes_to_ptr(data)**
函数功能	将bytes对象转换成为void*数据，可以将转换好的数据传递给C函数直接使用。
输入说明	**data**：bytes类型数据。
返回值说明	**ptr**：int，可以被C语言访问的数据。
约束说明	该函数输入的生命周期需要大于输出的生命周期，否则可能会产生未定义行为。

注意事项	推荐使用**acl.util.bytes_to_ptr**替代**acl.util.ptr_to_bytes**。
	修改示例：
	`bytes_in = np_arr_in.tobytes()`
	`bytes_ptr = acl.util.bytes_to_ptr(bytes_in)`

10.4.5 函数：ptr_to_bytes

C函数原型	无
Python函数	**bytes_out = acl.util.ptr_to_bytes(ptr, size)**
函数功能	将void*数据转换为bytes对象，可以使python代码直接访问。
输入说明	**ptr**：int，C的指针地址，能够访问的数据的首地址。 **size**：int，数据的大小，单位Byte。
返回值说明	**bytes_out** ：bytes对象。
约束说明	无

注意事项	由于**acl.util.ptr_to_bytes**会导致数据性能下降，建议使用**10.4.4 函数：bytes_to_ptr**。 bytes_out = acl.util.ptr_to_bytes(bytes_ptr, size)

10.4.6 thread

10.4.6.1 函数：start_thread

C函数原型	无
Python函数	**thr_id, ret = acl.util.start_thread(thr_func, args_list)**
函数功能	创建线程，在线程中执行传入的Python函数。
输入说明	**thr_func**：Python函数对象，表示用户自定义的Python函数。 例如： def thr_func(args_list): pass **args_list**：list，表示用户自定义的python函数（thr_func）需要的传入参数。
返回值说明	**thr_id**：int，线程ID号。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	新建线程，无法自动获取Context，需要调用**acl.rt.set_context**接口显式指定Context。

10.4.6.2 函数：stop_thread

C函数原型	无
Python函数	**ret = acl.util.stop_thread(thr_id)**
函数功能	等待由**acl.util.start_thread**接口创建的线程退出。
输入说明	**thr_id**：int，线程ID号。需指定由**acl.util.start_thread**接口获得的线程ID号。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。

约束说明	无
注意事项	无

10.5 Runtime（rt）

10.5.1 Device 管理

10.5.1.1 函数：set_device

C函数原型	**aclError aclrtSetDevice(int32_t deviceId)**
Python函数	**ret = acl.rt.set_device(device_id)**
函数功能	指定当前进程或线程中用于运算的Device，同时隐式创建默认Context，同步接口。 对于Atlas 200/500 A2推理产品，该默认Context中包含1个默认Stream。
输入说明	**device_id**：int，device设备号。 用户调用**acl.rt.get_device_count**接口获取可用的Device数量后，取值范围：[0, (*可用的Device数量*-1)]。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	如果多次调用**acl.rt.set_device**接口而不调用**acl.rt.reset_device**接口释放本进程使用的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**acl.rt.set_device**接口和**acl.rt.reset_device**接口配对使用，在不使用Device上资源时，通过调用**acl.rt.reset_device**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**acl.rt.set_device**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**acl.rt.set_device**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。 ● 在某一进程中指定Device，该进程内的多个线程可共用此Device显式创建Context（**acl.rt.create_context**接口）。 ● 多Device场景下，可在进程中通过**acl.rt.set_device**接口切换到其它Device。但利用Context（调用**acl.rt.set_context**接口）来切换Device，比使用**acl.rt.set_device**接口效率高。
注意事项	无

参考资源	接口调用流程与示例，请参见4.4 运行管理资源申请与释放、7.9 同步等待。

10.5.1.2 函数：reset_device

C函数原型	**aclError aclrtResetDevice(int32_t deviceId)**
Python函数	**ret = acl.rt.reset_device(device_id)**
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，同步接口。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。
输入说明	**device_id**：int，device设备号。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
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acl.rt.destroy_event接口释放Event/调用acl.rt.destroy_stream接口释放显式创建的Stream-->调用acl.rt.destroy_context释放显式创建的Context-->调用acl.rt.reset_device接口。
注意事项	无
参考资源	接口调用流程与示例，请参见4.4 运行管理资源申请与释放、7.9 同步等待。

10.5.1.3 函数：get_device

C函数原型	**aclError aclrtGetDevice(int32_t *deviceId)**
Python函数	**device_id, ret = acl.rt.get_device()**
函数功能	获取当前正在使用的Device的ID，同步接口。
输入说明	无
返回值说明	**device_id**：int，正在使用的Device设备号。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

约束说明	如果没有调用**acl.rt.set_device**接口显式指定Device或没有调用**acl.rt.create_context**接口隐式指定Device，则调用**acl.rt.get_device**接口时，返回错误。
注意事项	无

10.5.1.4 函数：get_run_mode

C函数原型	**aclError aclrtGetRunMode(aclrtRunMode *runMode)**
Python函数	**run_mode, ret = acl.rt.get_run_mode()**
函数功能	获取当前昇腾AI软件栈的运行模式，同步接口。
输入说明	无
返回值说明	**run_mode**：int，表示运行模式。 ● 0：ACL_DEVICE，昇腾AI软件栈运行在Device的Control CPU或板端环境上。 ● 1：ACL_HOST，昇腾AI软件栈运行在Host CPU上。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无
参考资源	接口调用流程与示例，请参见**4.4 运行管理资源申请与释放**。

10.5.1.5 函数：set_ts_device

C函数原型	**aclError aclrtSetTsDevice(aclrtTsId tsId)**
Python函数	**ret = acl.rt.set_ts_device(ts_id)**
函数功能	设置本次计算需要使用的Task Schedule，仅昇腾AI处理器支持vector core计算单元才生效，同步接口。
输入说明	**ts_id**：int，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，则设置该参数无效，默认使用AICORE Task Schedule。 ● 0：ACL_TS_ID_AICORE，使用AICORE Task Schedule。 ● 1：ACL_TS_ID_AIVECTOR，使用AIVECTOR Task Schedule。 ● 2：ACL_TS_ID_RESERVED。

返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.5.1.6 函数：get_device_count

C函数原型	**aclError aclrtGetDeviceCount(uint32_t *count)**
Python函数	**count, ret = acl.rt.get_device_count()**
函数功能	获取可用Device的数量，同步接口。
输入说明	无
输出说明	无
返回值说明	**count**：int，获取Device数量。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.5.2 Context 管理

10.5.2.1 函数：create_context

C函数原型	**aclError aclrtCreateContext(aclrtContext *context, int32_t deviceId)**
Python函数	**context, ret = acl.rt.create_context(device_id)**
函数功能	在当前进程或线程中显式创建一个Context。同步接口。 对于Atlas 200/500 A2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。
输入说明	**device_id**：int，指定需要创建Context的Device的ID设备号。

返回值说明	**context**：int，表示创建的Context在C中的指针。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	支持以下使用场景： ● 若不调用acl.rt.create_context接口显式创建Context，那系统会使用默认Context，该默认Context是在调用acl.rt.set_device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。 – （推荐）显式创建Context：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。 ● 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acl.rt.create_stream），当前线程在同一时刻内只能使用其中一个Context，建议通过acl.rt.set_context接口明确指定当前线程的Context，增加程序的可维护性。 **说明**
注意事项	无
参考资源	接口调用流程与示例，请参见4.4 运行管理资源申请与释放、7.9 同步等待。

10.5.2.2 函数：destroy_context

C函数原型	**aclError aclrtDestroyContext(aclrtContext context)**
Python函数	**ret = acl.rt.destroy_context(context)**
函数功能	销毁一个Context，释放Context的资源，同步接口。只能销毁通过acl.rt.create_context接口创建的Context。
输入说明	**context**：int，指定需要销毁的Context对象指针，通常是通过create，或者get获取到的Context对象。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无
参考资源	接口调用流程与示例，请参见4.4 运行管理资源申请与释放、7.9 同步等待。

10.5.2.3 函数：set_context

C函数原型	**aclError aclrtSetCurrentContext(aclrtContext context)**
Python函数	**ret = acl.rt.set_context(context)**
函数功能	设置线程的Context，同步接口。
输入说明	**context**：int，线程当前的Context对象指针。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	● 支持以下场景： – 如果在某线程（例如：thread1）中调用**acl.rt.create_context**接口显式创建一个Context（例如：ctx1），则可以不调用acl.rt.set_context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**acl.rt.create_context**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**acl.rt.destroy_context**接口来释放默认Context。 – 如果多次调用acl.rt.set_context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**acl.rt.create_context**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。
注意事项	该接口的入参有效性须由用户自己保证。
参考资源	接口调用流程，参见**7.2 多Device场景**。

10.5.2.4 函数：get_context

C函数原型	**aclError aclrtGetCurrentContext(aclrtContext *context)**
Python函数	**context, ret = acl.rt.get_context()**
函数功能	获取线程的Context，同步接口。 如果用户多次调用**acl.rt.set_context**接口设置当前线程的Context，则获取的是最后一次设置的Context。
输入说明	无

返回值说明	**context**：int，表示创建的Context对象指针。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.5.3 Stream 管理

10.5.3.1 函数：create_stream

C函数原型	**aclError aclrtCreateStream(aclrtStream *stream)**
Python函数	**stream, ret = acl.rt.create_stream()**
函数功能	在当前进程或线程中创建一个Stream，同步接口。
输入说明	无
返回值说明	**stream**：int，表示创建的Stream对象指针。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	每个Context对应一个默认Stream，该默认Stream是调用acl.rt.set_device接口或acl.rt.create_context接口隐式创建的。推荐调用acl.rt.create_stream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 200/500 A2推理产品，硬件资源最多支持512个Stream，如果已存在多个默认Stream，只能显式创建N个Stream（N=512-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510个Stream。
参考资源	接口调用流程与示例，请参见4.4 运行管理资源申请与释放、7.9 同步等待。

10.5.3.2 函数：create_stream_with_config

C函数原型	**aclError aclrtCreateStreamWithConfig(aclrtStream *stream, uint32_t priority, uint32_t flag)**
Python函数	**stream, ret = acl.rt.create_stream_with_config(priority, flag)**
函数功能	在当前进程或线程中创建一个Stream，同步接口。 使用本接口与**acl.rt.create_stream**接口相比是等价的。
输入说明	**priority**：int，优先级。当前固定设置为0，预留参数，暂不使用。 **flag**：int，Stream的flag。 取值范围： ● **ACL_STREAM_FAST_LAUNCH = 1**，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**10.5.3.1 函数：create_stream**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 = 2**，使用该模式创建出来的Stream，在调用**10.5.4.13 函数：synchronize_stream**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**10.5.3.1 函数：create_stream**接口创建出来的Stream，在调用**10.5.4.13 函数：synchronize_stream**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**acl.rt.create_stream**接口。
返回值说明	**stream**：int，表示创建的Stream对象指针地址。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。

10.5.3.3 函数：destroy_stream

C函数原型	**aclError aclrtDestroyStream(aclrtStream stream)**
Python函数	**ret = acl.rt.destroy_stream(stream)**
函数功能	销毁指定Stream，销毁通过**acl.rt.create_stream**或**acl.rt.create_stream_with_config**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。同步接口。

输入说明	**stream**：int，待销毁的Stream指针。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	● 在调用acl.rt.destroy_stream接口销毁指定Stream前，需要先调用acl.rt.synchronize_stream接口确保Stream中的任务都已完成。 ● 调用acl.rt.destroy_stream接口销毁指定Stream时，需确保该Stream在当前Context下。 ● 在调用acl.rt.destroy_stream接口销毁指定Stream时，需确保其它接口没有正在使用该Stream。
参考资源	接口调用流程与示例，请参见4.4 运行管理资源申请与释放、7.9 同步等待。

10.5.3.4 函数：destroy_stream_force

C函数原型	**aclError aclrtDestroyStreamForce(aclrtStream stream)**
Python函数	**ret = acl.rt.destroy_stream_force(stream)**
函数功能	销毁指定Stream，销毁通过acl.rt.create_stream或acl.rt.create_stream_with_config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。同步接口。
输入说明	**stream**：int，待销毁的Stream地址。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
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。

10.5.3.5 函数：set_stream_overflow_switch

C函数原型	**aclError aclrtSetStreamOverflowSwitch(aclrtStream stream, uint32_t flag)**
Python函数	**ret = acl.rt.set_stream_overflow_switch(stream, flag)**
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，同步接口。 Atlas 200/500 A2推理产品，不支持该接口。

输入说明	**stream**：int，操作Stream，若传入0，则操作默认Stream。 **flag**：int，溢出检测开关，取值范围如下： ● 0：关闭 ● 1：打开
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	● 在调用本接口前，可调用acl.rt.set_device_sat_mode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。

10.5.3.6 函数：get_stream_overflow_switch

C函数原型	**aclError aclrtGetStreamOverflowSwitch(aclrtStream stream, uint32_t flag)**
Python函数	**flag, ret = acl.rt.get_stream_overflow_switch(stream)**
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 200/500 A2推理产品，当前版本不支持该接口。
输入说明	**stream**：int，待操作Stream，若传入0，则操作默认Stream。
返回值说明	**flag**：int，溢出检测开关，取值范围如下： ● 0：关闭 ● 1：打开 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.5.3.7 函数：set_stream_failure_mode

C函数原型	**aclError aclrtSetStreamFailureMode(aclrtStream stream, uint64_t mode)**
Python函数	**ret = acl.rt.set_stream_failure_mode(stream, mode)**
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。同步接口。

输入说明	**stream**：int，待操作Stream。 **mode**：int，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 = 0：默认值，某个任务失败后，继续执行下一个任务。 ● ACL_STOP_ON_FAILURE = 1：某个任务失败后，停止执行后续的任务。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	“stream”参数不支持指定默认Stream（即向该参数传入0）的任务调度模式。

10.5.4 同步等待

10.5.4.1 函数：create_event

C函数原型	**aclError aclrtCreateEvent(aclrtEvent *event)**
Python函数	**event, ret = acl.rt.create_event()**
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、Event内的同步等待等场景。同步接口。
输入说明	无
返回值说明	**event**：int，创建的Event对象。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无
相关接口	您也可以使用**acl.rt.create_event_with_flag**接口创建Event，由于该接口当前仅支持将“flag”设置为“0x00000008”，表示创建的Event数量不受限制、且创建出来的Event仅用于统计两个Event之间的耗时。
参考资源	● 接口调用流程，参见**7.8.2 多Stream接口调用流程**。 ● 接口调用示例，参见**7.9.2 关于Event的同步等待**、**7.9.4 关于Stream间任务的同步等待**。

10.5.4.2 函数：create_event_with_flag

C函数原型	**aclError aclrtCreateEventWithFlag(aclrtEvent *event, uint32_t flag)**
Python函数	**event, ret = acl.rt.create_event_with_flag(flag)**
函数功能	创建一个Event，同步接口。
输入说明	**flag**：int，Event指针的flag。 ● ACL_EVENT_TIME_LINE = 0x00000008，表示创建的Event数量不受限制且创建出来的Event可用于统计两个Event之间的耗时。 ● ACL_EVENT_SYNC = 0x00000001，表示创建的Event数量受限且创建出来的可用于多Stream间的同步等待。 flag为bitmap，支持设置为单个flag或者对多个flag进行或操作。使用本接口创建Event时，若将flag参数值包含“ACL_EVENT_SYNC”，则创建出来的Event数量受限，具体如下： Atlas 200/500 A2推理产品，单个Device上最多支持1023个Event。
返回值说明	**event**：int，创建的Event对象指针。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	通过该接口创建Event时，由于当前仅支持将“flag”设置为“0x00000008”，表示创建的Event数量不受限制、且创建出来的Event仅用于统计两个Event之间的耗时。
注意事项	无
相关接口	您也可以使用**acl.rt.create_event**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。
参考资源	● 接口调用流程，参见**7.8.2 多Stream接口调用流程**。 ● 接口调用示例，参见**7.9.2 关于Event的同步等待**、**7.9.4 关于Stream间任务的同步等待**。

10.5.4.3 函数：destroy_event

C函数原型	**aclError aclrtDestroyEvent(aclrtEvent event)**
Python函数	**ret = acl.rt.destroy_event(event)**
函数功能	销毁一个Event，只能销毁通过**acl.rt.create_event**/**acl.rt.create_event_with_flag**接口创建的Event，同步接口。在销毁时，用户需确保等待**acl.rt.synchronize_event**接口或**acl.rt.stream_wait_event**接口涉及的任务都结束后，再销毁。

输入说明	**event**：int，待销毁的Event指针。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	在调用acl.rt.destroy_event接口销毁指定Event时，需确保其它接口没有正在使用该Event。
注意事项	无
参考资源	● 接口调用流程，参见7.8.2 多Stream接口调用流程。 ● 接口调用示例，参见7.9.2 关于Event的同步等待、7.9.4 关于Stream间任务的同步等待。

10.5.4.4 函数：record_event

C函数原型	**aclError aclrtRecordEvent(aclrtEvent event, aclrtStream stream)**
Python函数	**ret = acl.rt.record_event(event, stream)**
函数功能	在Stream中记录一个Event，同步接口。
输入说明	**event**：int，待记录的Event指针。 **stream**：int，将该Event记录在指定的Stream中，如果使用默认Stream，此处设置为0。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	● acl.rt.record_event接口与acl.rt.stream_wait_event接口配合使用时，主要用于多Stream之间同步的场景，在调用acl.rt.record_event接口时，系统内部会申请Event资源，在调用acl.rt.stream_wait_event接口之后，请及时调用acl.rt.reset_event接口释放Event资源。 **接口调用顺序**：acl.rt.create_event-->acl.rt.record_event-->acl.rt.stream_wait_event-->acl.rt.reset_event ● 由于在调用acl.rt.record_event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 200/500 A2推理产品，单个Device上最多支持65535个Event。
注意事项	无
参考资源	● 接口调用流程，参见7.8.2 多Stream接口调用流程。 ● 接口调用示例，参见7.9.2 关于Event的同步等待、7.9.4 关于Stream间任务的同步等待。

10.5.4.5 函数：reset_event

C函数原型	**aclError aclrtResetEvent(aclrtEvent event, aclrtStream stream)**
Python函数	**ret = acl.rt.reset_event(event, stream)**
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。
输入说明	**event**：int，待复位的Event对象指针。 **stream**：int，指定Event所在的Stream对象指针。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	在复位Event时，涉及重置Event的状态，Event的状态是在调用acl.rt.record_event接口时记录，因此在调用acl.rt.reset_event接口前，需要先调用acl.rt.record_event接口。 **接口调用顺序：acl.rt.create_event-->acl.rt.record_event-->**acl.rt.reset_event**-->acl.rt.synchronize_stream**
注意事项	无

10.5.4.6 函数：query_event

C函数原型	**aclError aclrtQueryEvent(aclrtEvent event, aclrtEventStatus *status)**
Python函数	**status, ret = acl.rt.query_event(event)**
函数功能	查询与本接口在同一线程中的10.5.4.4 函数：record_event接口所记录的Event是否执行完成。同步接口。
输入说明	**event**：int，指定待查询的Event对象指针。
返回值说明	**status**：int，表示的是Event的状态。 ● 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。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	此接口后续版本会废弃，请使用10.5.4.7 函数：query_event_status接口。

10.5.4.7 函数：query_event_status

C函数原型	**aclError aclrtQueryEventStatus(aclrtEvent event, aclrtEventRecordedStatus *status)**
Python函数	**status, ret = acl.rt.query_event_status(event)**
函数功能	本接口是**acl.rt.query_event**接口的扩展接口，可查询与本接口在同一线程或不同线程中的**10.5.4.4 函数：record_event**接口所记录的Event是否执行完成。同步接口
输入说明	**event**：int，指定待查询的Event对象指针地址。
返回值说明	**status**：int，表示的是Event的状态。 ● 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。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无

10.5.4.8 函数：query_event_wait_status

C函数原型	**aclError aclrtQueryEventWaitStatus(aclrtEvent event, aclrtEventWaitStatus *status)**
Python函数	**status, ret=acl.rt.query_event_wait_status(event)**
函数功能	调用**acl.stream_wait_event**接口后，查询Stream是否接收到指定Event。如果多个Stream等待同一个Event，则多个Stream都接收到Event，通过该接口查询的Event状态才是已完成，同步接口。
输入说明	**event**：指定待查询的Event。
返回值说明	**status**：Event的状态，返回类型**aclrtEventStatus**。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无

相关接口	无

10.5.4.9 函数：synchronize_event

C函数原型	**aclError aclrtSynchronizeEvent(aclrtEvent event)**
Python函数	**ret = acl.rt.synchronize_event(event)**
函数功能	阻塞应用程序运行，等待Event完成，同步接口。
输入说明	**event**：int，指定需等待的Event对象指针。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无
参考资源	接口调用示例，参见7.9.2 关于Event的同步等待。

10.5.4.10 函数：event_elapsed_time

C函数原型	**aclError aclrtEventElapsedTime(float *ms, aclrtEvent startEvent, aclrtEvent endEvent)**
Python函数	**ms, ret = acl.rt.event_elapsed_time(start, end)**
函数功能	统计两个Event之间的耗时，同步接口。
输入说明	**start**：int，指定起始Event对象指针。 **end**：int，指定结尾Event对象指针。
返回值说明	**ms**：float，表示的是两个Event之间的耗时，单位是毫秒。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	接口调用顺序：调用acl.rt.create_event/acl.rt.create_event_with_flag接口创建event-->调用acl.rt.record_event接口在同一个stream中记录起始event、结尾event-->调用acl.rt.synchronize_stream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acl.rt.event_elapsed_time接口统计两个Event之间的耗时。
注意事项	无

10.5.4.11 函数：stream_wait_event

C函数原型	**aclError aclrtStreamWaitEvent(aclrtStream stream, aclrtEvent event)**
Python函数	**ret = acl.rt.stream_wait_event(stream, event)**
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。
输入说明	**stream**：int，指定需要等待Event完成的Stream。 如果使用默认Stream，此处设置为0。 **event**：int，需等待的Event。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	● acl.rt.record_event接口与acl.rt.stream_wait_event接口配合使用时，主要用于多Stream之间同步的场景，在调用acl.rt.record_event接口时，系统内部会申请Event资源，在调用acl.rt.stream_wait_event接口之后，请及时调用acl.rt.reset_event接口释放Event资源。 **接口调用顺序**：acl.rt.create_event-->acl.rt.record_event-->acl.rt.stream_wait_event-->acl.rt.reset_event ● 一个进程内，调用acl.init接口初始化pyACL后，调用acl.rt.set_op_wait_timeout接口设置超时时间，本进程内后续调用acl.rt.stream_wait_event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pyACL会返回报错。 由于acl.rt.stream_wait_event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acl.rt.synchronize_stream接口后，会返回报错。
注意事项	无
参考资源	● 接口调用流程，参见7.8.2 多Stream接口调用流程。 ● 接口调用示例，参见7.9.2 关于Event的同步等待、7.9.4 关于Stream间任务的同步等待。

10.5.4.12 函数：synchronize_device

C函数原型	**aclError ret = aclrtSynchronizeDevice();**
Python函数	**ret = acl.rt.synchronize_device()**
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，同步接口。

输入说明	无
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	一个进程内多线程场景下，不支持多线程并发调用acl.rt.synchronize_device接口。
注意事项	无
参考资源	● 接口调用流程，参见**7.8.2 多Stream接口调用流程**。 ● 接口调用示例，参见**7.9.2 关于Event的同步等待**。

10.5.4.13 函数：synchronize_stream

C函数原型	**aclError aclrtSynchronizeStream(aclrtStream stream)**
Python函数	**ret = acl.rt.synchronize_stream(stream)**
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，同步接口。
输入说明	**stream**：int，指定需要完成所有任务的Stream对象。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无
参考资源	接口调用示例，参见**7.9.4 关于Stream间任务的同步等待**。

10.5.4.14 函数：synchronize_stream_with_timeout

C函数原型	**aclError aclrtSynchronizeStreamWithTimeout(aclrtStream stream, int32_t timeout)**
Python函数	**ret = acl.rt.synchronize_stream_with_timeout(stream, timeout)**
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**10.5.4.13 函数：synchronize_stream**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。同步接口。

输入说明	**stream**：int，指定需要完成所有任务的Stream对象。 **timeout**：int，接口的超时时间。取值说明如下： ● -1：表示永久等待，和接口**10.5.4.13 函数：synchronize_stream**功能一样。 ● >0：配置具体的超时时间，单位是毫秒。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无
参考资源	无

10.5.4.15 函数：subscribe_report

C函数原型	**aclError ret = aclrtSubscribeReport(thread_id, stream);**
Python函数	**ret = acl.rt.subscribe_report(thread_id, stream)**
函数功能	指定处理stream上回调函数的线程。同步接口。
输入说明	**thread_id**：int，指定线程id。 **stream**：int，指定需要处理的Stream对象指针。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	● 支持多次调用acl.rt.subscribe_report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acl.rt.subscribe_report接口给同一个Stream注册多个处理回调函数的线程。 ● 单进程内调用acl.rt.subscribe_report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acl.rt.subscribe_report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。
注意事项	无

参考资源	● 接口调用流程，参见7.5.1 接口调用流程。 ● 接口调用示例，参见7.5.2 示例代码。

10.5.4.16 函数：launch_callback

C函数原型	**aclError ret = aclrtLaunchCallback(aclrtCallback fn, void *userData, aclrtCallbackBlockType blockType, aclrtStream stream)**
Python函数	**ret = acl.rt.launch_callback(fn, user_data_list, block_type, stream)**
函数功能	在Stream的任务队列中增加一个需要执行的回调函数。异步接口。
输入说明	**fn**：function，表示Python侧的回调函数。 **user_data_list**：list，表示需要传递给回调函数的参数（目前传入的是list类型的数据）。 **block_type**：int，指定线程是否阻塞。 ● 0：表示非阻塞。 ● 1：表示阻塞。 **stream**：int，表示指定的Stream。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无
参考资源	● 接口调用流程，参见7.5.1 接口调用流程。 ● 接口调用示例，参见7.5.2 示例代码。

10.5.4.17 函数：process_report

C函数原型	**aclError ret = aclrtProcessReport(timeout);**
Python函数	**ret = acl.rt.process_report(timeout)**
函数功能	等待指定时间后，触发回调处理，由acl.rt.subscribe_report接口指定的线程处理回调。同步接口。 用户需新建一个线程，在线程函数内调用acl.rt.process_report接口。

输入说明	**timeout**：int，指定超时时间，单位为ms。取值范围为： ● -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	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	需要在callback函数中调用该函数。
注意事项	无
资源参考	● 接口调用流程，参见7.5.1 接口调用流程。 ● 接口调用示例，参见7.5.2 示例代码。

10.5.4.18 函数：unsubscribe_report

C函数原型	**aclError ret = aclrtUnSubscribeReport(thread_id, stream);**
Python函数	**ret = acl.rt.unsubscribe_report(thread_id, stream)**
函数功能	取消线程注册，Stream上的回调函数不再由指定线程处理。同步接口。
输入说明	**thread_id**：int，指定线程id。 **stream**：int，指定需要处理的Stream对象指针。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无
参考资源	● 接口调用流程，参见7.5.1 接口调用流程。 ● 接口调用示例，参见7.5.2 示例代码。

10.5.4.19 函数：set_exception_info_callback

C函数原型	**aclError ret = aclrtSetExceptionInfoCallback(aclrtExceptionInfoCallback callback)**
Python函数	**ret = acl.rt.set_exception_info_callback(fn)**

函数功能	设置异常回调函数，同步接口。

输入说明	**fn**：function，表示Python侧的回调函数，格式如下：
	`def exception_callback(exception_info)`
	`"""`
	`:exception_info: 表示异常信息aclrtExceptionInfo结构体的指针对象`
	`:return:`
	`"""`

返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
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**acl.rt.get_task_id_from_exception_info**、**acl.rt.get_stream_id_from_exception_info**、**acl.rt.get_thread_id_from_exception_info**、**acl.rt.get_device_id_from_exception_info**、**acl.rt.get_error_code_from_exception_info**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**acl.op.execute_v2**接口前，调用acl.rt.set_exception_info_callback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acl.rt.set_exception_info_callback接口，将入参设置为空指针。
注意事项	无
参考资源	接口调用示例，参见**7.10 AI Core异常信息获取**。

10.5.4.20 函数：get_task_id_from_exception_info

C函数原型	**uint32_t streamId = aclrtGetTaskIdFromExceptionInfo(const aclrtExceptionInfo *info)**
Python函数	**task_id = acl.rt.get_task_id_from_exception_info(info)**
函数功能	获取异常信息中的任务ID，同步接口。
输入说明	**info**：int，表示异常信息aclrtExceptionInfo结构体的指针对象。 在执行任务之前调用**acl.rt.set_exception_info_callback**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	**task_id**：int，返回异常信息中的任务ID，返回值为0xFFFFFFFF（以十六进制为例）时表示异常信息为空。
约束说明	无
注意事项	无
参考资源	接口调用示例，参见**7.10 AI Core异常信息获取**。

10.5.4.21 函数：get_stream_id_from_exception_info

C函数原型	**uint32_t streamId = aclrtGetStreamIdFromExceptionInfo(const aclrtExceptionInfo *info)**
Python函数	**stream_id = acl.rt.get_stream_id_from_exception_info(info)**
函数功能	获取异常信息中的Stream ID，同步接口。
输入说明	**info**：int，表示异常信息aclrtExceptionInfo结构体的指针对象。 在执行任务之前调用**acl.rt.set_exception_info_callback**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	**stream_id**：int，返回异常信息中的Stream ID，返回值为“0xFFFFFFFF”（以十六进制为例）时表示异常信息为空。
约束说明	无
注意事项	无
参考资源	接口调用示例，参见**7.10 AI Core异常信息获取**。

10.5.4.22 函数：get_thread_id_from_exception_info

C函数原型	**uint32_t streamId = aclrtGetThreadIdFromExceptionInfo(const aclrtExceptionInfo *info)**
Python函数	**thread_id = acl.rt.get_thread_id_from_exception_info(info)**
函数功能	获取异常信息中的线程ID，同步接口。
输入说明	**info**：int，表示异常信息aclrtExceptionInfo结构体的对象。 在执行任务之前调用**acl.rt.set_exception_info_callback**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	**thread_id**：int，返回异常信息中的线程ID，返回值为“0xFFFFFFFF”（以十六进制为例）时表示异常信息为空。
约束说明	无

注意事项	无

10.5.4.23 函数：get_device_id_from_exception_info

C函数原型	**uint32_t aclrtGetDeviceIdFromExceptionInfo(const aclrtExceptionInfo *info)**
Python函数	**device_id = acl.rt.get_device_id_from_exception_info(info)**
函数功能	获取异常信息中的Device ID，同步接口。
输入说明	**info**：int，表示异常信息aclrtExceptionInfo结构体的对象。 在执行任务之前调用**acl.rt.set_exception_info_callback**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	**device_id**：int，返回异常信息中的Device ID，返回值为“0xFFFFFFFF”（以十六进制为例）时表示异常信息为空。
约束说明	无
注意事项	无
参考资源	接口调用示例，参见**7.10 AI Core异常信息获取**。

10.5.4.24 函数：get_error_code_from_exception_info

C函数原型	**uint32_t aclrtGetErrorCodeFromExceptionInfo(const aclrtExceptionInfo *info)**
Python函数	**ret = acl.rt.get_error_code_from_exception_info(info)**
函数功能	获取异常信息中的错误码，同步接口。该接口与**acl.rt.set_exception_info_callback**接口配合使用。
输入说明	**info**：int，异常信息的指针地址。 在执行任务之前调用**acl.rt.set_exception_info_callback**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	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败,返回值为0xFFFFFFFF（以十六进制为例）时表示Device异常。
约束说明	无
注意事项	无

10.5.4.25 函数：set_op_wait_timeout

C函数原型	**aclError aclrtSetOpWaitTimeout(uint32_t timeout)**
Python函数	**ret = acl.rt.set_op_wait_timeout(timeout)**
函数功能	本接口用于设置等待Event完成的超时时间。同步接口。
输入说明	**timeout**：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	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
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acl.init接口初始化pyACL后，调用本接口设置超时时间，本进程内后续调用acl.rt.stream_wait_event接口下发的任务支持在所设置的超时时间内等待。 由于acl.rt.stream_wait_event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acl.rt.synchronize_stream接口后，会返回报错。
注意事项	无

10.5.5 内存管理

10.5.5.1 总体说明

- 若涉及媒体数据处理功能，关于内存使用，有以下注意事项：
  a. 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acl.media.dvpp_malloc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acl.himpi.dvpp_malloc接口申请内存。
  b. 调用上述接口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acl.rt.malloc接口、或acl.rt.malloc_host接口、或acl.rt.malloc_cached接口申请内存。

10.5.5.2 函数：malloc

C函数原型	**aclError aclrtMalloc(void **devPtr, size_t size, aclrtMemMallocPolicy policy)**
Python函数	**dev_ptr, ret = acl.rt.malloc(size, policy)**
函数功能	申请Device上的内存，同步接口。 在Device上申请size大小的线性内存，通过**dev_ptr**返回已分配内存的指针地址。 通过该接口申请的Device内存都支持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**acl.media.dvpp_malloc**申请内存。
输入说明	**size**：int，申请内存的大小，单位Byte，size不能为0。 **policy**：int，内存分配规则。 ● 0：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。 ● 1：ACL_MEM_MALLOC_HUGE_ONLY，仅申请大页，如果大页内存不够，则返回错误。 ● 2：ACL_MEM_MALLOC_NORMAL_ONLY，仅申请普通页。 ● 3：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。预留选项。 ● 4：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。预留选项。 ● 5：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。预留选项。
返回值说明	**dev_ptr**：int，指向Device上已分配内存的指针地址。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。

约束说明	• 使用**acl.rt.malloc**接口申请的内存，需要通过**acl.rt.free**接口释放内存。 • 频繁调用**acl.rt.malloc**接口申请内存、调用**acl.rt.free**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**acl.rt.malloc**接口申请内存时，会对用户输入的size按向上对齐成32字节整数倍后，再多加32字节。 • 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。
注意事项	无
参考资源	接口调用示例，参见**4.5 数据传输**。

10.5.5.3 函数：malloc_cached

C函数原型	**aclError aclrtMallocCached(void **devPtr, size_t size, aclrtMemMallocPolicy policy)**
Python函数	**dev_ptr, ret = acl.rt.malloc_cached(size, policy)**
函数功能	申请Device上的内存，同步接口，该接口在任何场景下申请的内存都是支持cache缓存。在Device上申请size大小的线性内存，通过**dev_ptr**返回已分配内存的指针地址。 使用**acl.rt.malloc_cached**接口申请的内存与使用**acl.rt.malloc**接口申请的内存是等价的，都支持cache缓存。 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**acl.media.dvpp_malloc**申请内存。

输入说明	**size**：int，申请内存的大小，单位Byte。size不能为0。 **policy**：int，内存分配规则。 ● 0：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。 ● 1：ACL_MEM_MALLOC_HUGE_ONLY，仅申请大页，如果大页内存不够，则返回错误。 ● 2：ACL_MEM_MALLOC_NORMAL_ONLY，仅申请普通页。 ● 3：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。预留选项。 ● 4：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。预留选项。 ● 5：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。预留选项。
返回值说明	**dev_ptr**：int，指向Device上已分配内存的指针地址。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	其它约束与**acl.rt.malloc**接口相同。
注意事项	无

10.5.5.4 函数：mem_flush

C函数原型	**aclError aclrtMemFlush(void *devPtr, size_t size)**
Python函数	**ret = acl.rt.mem_flush(dev_ptr, size)**
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。
输入说明	**dev_ptr**：int，要flush的ddr内存起始地址。 **size**：int，要flush的ddr内存大小，单位Byte。size不能为0。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无

注意事项	无

10.5.5.5 函数：mem_invalidate

C函数原型	**aclError aclrtMemInvalidate(void *devPtr, size_t size)**
Python函数	**ret = acl.rt.mem_invalidate(dev_ptr, size)**
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。
输入说明	**dev_ptr**：int，需要指定ddr内存对应的cache无效。 **size**：int，ddr内存大小，单位Byte。size不能为0。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.5.5.6 函数：get_mem_info

C函数原型	**aclError aclrtGetMemInfo(aclrtMemAttr attr, size_t *free, size_t *total)**
Python函数	**free, total, ret = acl.rt.get_mem_info(attr)**
函数功能	获取指定属性的pyACL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。同步接口。 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。

输入说明	**attr**：int，需要查询的内存的属性值。 ● 0：ACL_DDR_MEM，DDR内存，DDR上所有大页内存+普通内存。 ● 1：ACL_HBM_MEM，HBM内存，HBM上所有大页内存+普通内存。 ● 2：ACL_DDR_MEM_HUGE，DDR大页内存。 ● 3：ACL_DDR_MEM_NORMAL，DDR普通内存。 ● 4：ACL_HBM_MEM_HUGE，HBM大页内存。 ● 5：ACL_HBM_MEM_NORMAL，HBM普通内存。 ● 6：ACL_DDR_MEM_P2P_HUGE，DDR中用于Device间数据复制的大页内存。预留选项。 ● 7：ACL_DDR_MEM_P2P_NORMAL，DDR中用于Device间数据复制的普通内存。预留选项。 ● 8：ACL_HBM_MEM_P2P_HUGE，HBM中用于Device间数据复制的大页内存。预留选项。 ● 9：ACL_HBM_MEM_P2P_NORMAL，HBM中用于Device间数据复制的普通内存。预留选项。
返回值说明	● **free**：int，对应属性的内存的空闲大小，单位Byte。 ● **total**：int，对应属性的内存的总大小，单位Byte。 ● **ret**：int，错误码。 – 返回0表示成功。 – 返回其它值表示失败。
约束说明	调用本接口前是必须先指定用于计算的Device，因此本接口中不体现Device ID。
注意事项	无

10.5.5.7 函数：free

C函数原型	**aclError aclrtFree(void *devPtr)**
Python函数	**ret= acl.rt.free(dev_ptr)**
函数功能	释放Device上的内存，同步接口。 acl.rt.free接口只能释放通过acl.rt.malloc接口申请的内存。
输入说明	**dev_ptr**：int，待释放内存的指针对象。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无

注意事项	无
参考资源	接口调用示例，参见**4.5 数据传输**。

10.5.5.8 函数：malloc_host

C函数原型	**aclError aclrtMallocHost(void **hostPtr, size_t size)**
Python函数	**host_ptr, ret= acl.rt.malloc_host(size)**
函数功能	应用在Host上运行时，调用该接口申请的是Host内存，由系统保证内存首地址64字节对齐。应用在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。同步接口。
输入说明	**size**：int，申请内存的大小，单位Byte。size不能为0。
返回值说明	**host_ptr**：int，指向Host上已分配内存的指针的指针对象。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	● 使用acl.rt.malloc_host接口申请的内存，需要通过**acl.rt.free_host**接口释放内存。 ● 频繁调用**acl.rt.malloc_host**接口申请内存、调用**acl.rt.free_host**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。
注意事项	无
参考资源	接口调用示例，参见**4.5 数据传输**。

10.5.5.9 函数：free_host

C函数原型	**aclError aclrtFreeHost(void *hostPtr)**
Python函数	**ret= acl.rt.free_host(host_ptr)**

函数功能	释放Host上的内存，同步接口。 acl.rt.free_host接口只能释放通过acl.rt.malloc_host接口申请的内存。
输入说明	**host_ptr**：int，待释放内存的对象的指针。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无
参考资源	接口调用示例，参见4.5 数据传输。

10.5.5.10 函数：memset

C函数原型	**aclError aclrtMemset(void *devPtr, size_t maxCount, int32_t value, size_t count)**
Python函数	**ret = acl.rt.memset(dev_ptr, max_count, value, count)**
函数功能	初始化内存，将内存中的内容设置为指定的值，同步接口。 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。
输入说明	**dev_ptr**：int，内存的起始地址。 **max_count**：int，内存的最大长度，单位Byte。 **value**：int，设置的值。 **count**：int，需要设置为指定值的内存长度，单位Byte。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.5.5.11 函数：memset_async

C函数原型	**aclError aclrtMemsetAsync(void *devPtr, size_t maxCount, int32_t value, size_t count, aclrtStream stream)**
Python函数	**ret = acl.rt.memset_async(dev_ptr, max_count, value, count, stream)**

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。
输入说明	**dev_ptr**：int，内存的起始地址。 **max_count**：int，内存的最大长度，单位Byte。 **value**：int，设置的值。 **count**：int，需要设置为指定值的内存长度，单位Byte。 **stream**：int，Stream id号。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
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，一定要调用acl.rt.synchronize_stream接口确保内存初始化的任务已执行完成。
注意事项	无

10.5.5.12 函数：memcpy

C函数原型	**aclError aclrtMemcpy(void *dst, size_t destMax, const void *src, size_t count, aclrtMemcpyKind kind)**
Python函数	**ret = acl.rt.memcpy(dst, dest_max, src, count, kind)**
函数功能	实现Host内、Host与Device之间、Device内的同步内存复制。
输入说明	**dst**：int，目的内存地址指针对象。 **dest_max**：int，目的内存地址的最大内存长度，单位Byte。 **src**：int，源内存地址指针对象。 **count**：整型，内存复制的长度，单位Byte。 **kind**：整型，内存复制的类型，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。 ● 0：ACL_MEMCPY_HOST_TO_HOST，Host内的内存复制 ● 1：ACL_MEMCPY_HOST_TO_DEVICE，Host到Device的内存复制 ● 2：ACL_MEMCPY_DEVICE_TO_HOST，Device到Host的内存复制 ● 3：ACL_MEMCPY_DEVICE_TO_DEVICE，Device内的内存复制
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无

注意事项	无
参考资源	接口调用示例，参见4.5 数据传输。

10.5.5.13 函数：memcpy_async

C函数原型	**aclError aclrtMemcpyAsync(void *dst, size_t destMax, const void *src, size_t count, aclrtMemcpyKind kind, aclrtStream stream)**
Python函数	**ret = acl.rt.memcpy_async(dst, dest_max, src, count, kind, stream)**
函数功能	实现Host内、Host与Device之间、Device内的异步内存复制。
输入说明	**dst**：int，目的内存地址指针对象。 **dest_max**：int，目的内存地址的最大内存长度，单位Byte。 **src**：int，源内存地址指针对象。 **count**：int，内存复制的长度，单位Byte。 **kind**：int，内存复制的类型。 ● 0：ACL_MEMCPY_HOST_TO_HOST，Host内的内存复制 ● 1：ACL_MEMCPY_HOST_TO_DEVICE， Host到Device的内存复制 ● 2：ACL_MEMCPY_DEVICE_TO_HOST，Device到Host的内存复制 ● 3：ACL_MEMCPY_DEVICE_TO_DEVICE，Device内的内存复制 **stream**：int，stream ID。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
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，一定要调用acl.rt.synchronize_stream接口确保内存复制的任务已执行完成。 ● 本接口不支持异步Host内的内存复制功能，因此调用本接口选择ACL_MEMCPY_HOST_TO_HOST类型进行内存复制时，任务下发成功，但系统内部处理该任务时会返回失败。
注意事项	无
参考资源	接口调用示例，参见4.5 数据传输。

10.5.5.14 函数：memcpy2d

C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)**
Python函数	**ret = acl.rt.memcpy2d(dst, dpitch, src, spitch, width, height, kind)**
函数功能	实现同步内存复制，主要用于矩阵数据的复制。同步接口。

输入说明	**dst**：目的内存地址指针。
	dpitch：目的内存中相邻两列向量的地址距离。
	src：源内存地址指针。
	spitch：源内存中相邻两列向量的地址距离。
	width：待复制的矩阵宽度。
	height：待复制的矩阵高度。
	kind：内存复制的类型。
	ACL_MEMCPY_HOST_TO_HOST = 0 // Host内的内存复制
	ACL_MEMCPY_HOST_TO_DEVICE = 1 // Host到Device的内存复制
	ACL_MEMCPY_DEVICE_TO_HOST = 2 // Device到Host的内存复制
	ACL_MEMCPY_DEVICE_TO_DEVICE = 3 // Device内或Device间的内存复制

返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
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 200/500 A2推理产品，Ascend RC形态下，不支持调用本接口。

10.5.5.15 函数：memcpy2d_async

C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)**
Python函数	**ret = acl.rt.memcpy2d_async(dst, dpitch, src, spitch, width, height, kind, stream)**
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。

输入说明	**dst**：目的内存地址指针。
	dpitch：目的内存中相邻两列向量的地址距离。
	src：源内存地址指针。
	spitch：源内存中相邻两列向量的地址距离。
	width：待复制的矩阵宽度。
	height：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：内存复制的类型。
	ACL_MEMCPY_HOST_TO_HOST = 0 // Host内的内存复制
	ACL_MEMCPY_HOST_TO_DEVICE = 1 // Host到Device的内存复制
	ACL_MEMCPY_DEVICE_TO_HOST = 2 // Device到Host的内存复制
	ACL_MEMCPY_DEVICE_TO_DEVICE = 3 // Device内或Device间的内存复制
	stream：指定stream。

返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
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，一定要调用**acl.rt.synchronize_stream**。 ● 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 200/500 A2推理产品，Ascend RC形态下，不支持调用本接口。

10.6 模型加载与执行（mdl）

10.6.1 模型加载与执行

10.6.1.1 函数：load_from_file

C函数原型	**aclError aclmdlLoadFromFile(const char *modelPath, uint32_t *modelId)**
Python函数	**model_id, ret = acl.mdl.load_from_file(model_path)**
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，同步接口。 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。

输入说明	**model_path**：str，离线模型文件的存储路径，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4.2 模型构建。 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pyACL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。
返回值说明	**model_id**：int，系统完成模型加载后生成的模型ID对应的指针对象。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
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acl.rt.create_context）。
注意事项	无
相关接口	当前pyACL还提供了acl.mdl.set_config_opt接口、acl.mdl.load_with_config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。

10.6.1.2 函数：load_from_mem

C函数原型	**aclError aclmdlLoadFromMem(const void* model, size_t modelSize, uint32_t* modelId)**
Python函数	**model_id, ret = acl.mdl.load_from_mem(model, model_size)**
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，同步接口。 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。
输入说明	**model**：int，模型数据的内存地址对应的指针对象。 ● 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10.5.5 内存管理。 **model_size**：int，内存中的模型数据长度，单位Byte。

返回值说明	**model_id**：int，系统完成模型加载后生成的模型ID。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
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**acl.rt.create_context**）。
注意事项	无
相关接口	当前pyACL还提供了**acl.mdl.set_config_opt**接口、**acl.mdl.load_with_config**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**。

10.6.1.3 函数：load_from_file_with_mem

C函数原型	**aclError aclmdlLoadFromFileWithMem(const char *modelPath, uint32_t *modelId, void *workPtr, size_t workSize, void *weightPtr, size_t weightSize)**
Python函数	**model_id, ret = acl.mdl.load_from_file_with_mem(model_path, work_ptr, work_size, weight_ptr, weight_size)**
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，同步接口。 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。
输入说明	**model_path**：str，离线模型文件的存储路径，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**4.2 模型构建**。 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pyACL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。 **work_ptr**：int，Device上模型所需工作内存（存放模型输入/输出等数据）的地址，由用户自行管理。 **work_size**：int，模型所需工作内存的大小，单位Byte。 **weight_ptr**：int，Device上模型权值内存（存放权值数据）的指针，由用户自行管理。 **weight_size**：int，模型所需权值内存的大小，单位Byte。

返回值说明	**model_id**：系统完成模型加载后生成的模型ID。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
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acl.rt.create_context）。
注意事项	无
相关接口	当前pyACL还提供了acl.mdl.set_config_opt接口、acl.mdl.load_with_config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。

10.6.1.4 函数：load_from_mem_with_mem

C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)**
Python函数	**model_id，ret = acl.mdl.load_from_mem_with_mem(model, model_size, work_ptr, work_size, weight_ptr, weight_size)**
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，同步接口。 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。
输入说明	**model**：int，模型数据的内存地址。 ● 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。 ● 应用的运行模式可使用acl.rt.get_run_mode接口获取。 **model_size**：int，模型数据长度，单位Byte。 **work_ptr**：int，Device上模型所需工作内存（存放模型输入/输出等数据）的地址，由用户自行管理。 **work_size**：int，模型所需工作内存的大小，单位Byte。 **weight_ptr**：int，Device上模型权值内存（存放权值数据）的指针，由用户自行管理。 **weight_size**：int，模型所需权值内存的大小，单位Byte。
返回值说明	**model_id**：int，系统完成模型加载后生成的模型ID。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

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**acl.rt.create_context**）。
注意事项	无
相关接口	当前pyACL还提供了**acl.mdl.set_config_opt**接口、**acl.mdl.load_with_config**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**。

10.6.1.5 函数：execute

C函数原型	**aclError aclmdlExecute(uint32_t modelId, const aclmdlDataset *input, aclmdlDataset *output)**
Python函数	**ret = acl.mdl.execute(model_id, input, output)**
函数功能	执行模型推理，直到返回推理结果，同步接口。
输入说明	**model_id**：int，指定需要执行推理的模型的ID。 **input**：int，模型推理的输入数据对应的指针对象。 **output**：int，模型推理的输出数据对应的指针对象。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。

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) -> acl.rt.memcpy刷新输入输出内存 -> acl.mdl.execute执行推理 -> unlock(handle1)
	
	// 线程B的接口调用顺序：
	lock(handle1) -> acl.rt.memcpy刷新输入输出内存 -> acl.mdl.execute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**acl.rt.create_context**）。
	• 存放模型输入/输出数据的内存，可以使用以下接口申请：**acl.rt.malloc**、或**acl.rt.malloc_host**、或**acl.rt.malloc_cached**接口、或**acl.media.dvpp_malloc**接口、或**acl.himpi.dvpp_malloc**接口。
	其中，**acl.media.dvpp_malloc**接口和**acl.himpi.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**。

注意事项	input和output指针对象需要预先创建。
参考资源	• 接口调用流程，参见**4.6.2.1 接口调用流程**。 • 接口调用示例，参见**4.6.2.2 示例代码**。

10.6.1.6 函数：execute_async

C函数原型	**aclError aclmdlExecuteAsync(uint32_t modelId, const aclmdlDataset *input, aclmdlDataset *output, aclrtStream stream)**
Python函数	**ret = acl.mdl.execute_async(modelId, input, output_, stream)**
函数功能	执行模型推理，异步接口。
输入说明	**model_id**：int，指定需要执行推理的模型的ID。 **input**：int，模型推理的输入数据的指针对象。 **output**：int，模型推理的输出数据的指针对象。 **stream**：int，指定Stream的指针对象。
返回值说明	**ret**：int，错误码。 • 返回0表示成功。 • 返回**其它值**表示失败。

约束说明	• 对同一个modelId的模型，不能调用acl.mdl.execute_async接口执行多Stream并发场景下的模型推理。
	• 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) -> acl.rt.memcpy_async(stream1)刷新输入输出内存 ->
	acl.mdl.execute_async(modelId1,stream1)执行推理 -> unlock(handle1)
	
	// 线程B的接口调用顺序：
	lock(handle1) -> acl.rt.memcpy_async(stream1)刷新输入输出内存 ->
	acl.mdl.execute_async(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**acl.rt.create_context**）。
	• 存放模型输入/输出数据的内存，可以使用以下接口申请：**acl.rt.malloc**、或**acl.rt.malloc_host**、或**acl.rt.malloc_cached**接口、或**acl.media.dvpp_malloc**接口、或**acl.himpi.dvpp_malloc**接口。
	其中，**acl.media.dvpp_malloc**接口和**acl.himpi.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**。

注意事项	stream、input和output指针对象需要预先创建。
参考资源	• 接口调用流程，参见**7.5.1 接口调用流程**。 • 接口调用示例，参见**7.5.2 示例代码**。

10.6.1.7 函数：unload

C函数原型	**aclError aclmdlUnload(uint32_t modelId)**
Python函数	**ret = acl.mdl.unload(model_id)**
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，同步接口。
输入说明	**model_id**：int，指定需卸载的模型的ID。
返回值说明	**ret**：int，错误码。 • 返回0表示成功。 • 返回**其它值**表示失败。
约束说明	• 在调用acl.mdl.unload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**acl.rt.create_context**）。

注意事项	无
参考资源	接口调用流程，参见**3.4 pyACL接口调用流程**。

10.6.1.8 函数：query_size

C函数原型	**aclError aclmdlQuerySize(const char *fileName, size_t *workSize, size_t *weightSize)**
Python函数	**work_size, weight_size, ret = acl.mdl.query_size(file_name)**
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，同步接口。
输入说明	**file_name**：str，需要获取内存信息的模型路径，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。
返回值说明	**work_size**：int，模型执行时所需的工作内存的大小，单位Byte。 **weight_size**：int，模型执行时所需权值内存的大小，单位Byte。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
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**acl.mdl.query_size**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**acl.mdl.query_size**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，**acl.mdl.load_from_file**、**acl.mdl.load_from_mem**）。
参考资源	接口调用流程，参见**4.6.1 模型加载**。

10.6.1.9 函数：query_size_from_mem

C函数原型	**aclError aclmdlQuerySizeFromMem(const void *model, size_t modelSize, size_t *workSize, size_t *weightSize)**
Python函数	**work_size, weight_size, ret = acl.mdl.query_size_from_mem(model, model_size)**
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，同步接口。
输入说明	**model**：int，需要获取内存信息的模型数据对应的指针对象。 **model_size**：int，模型数据长度，单位Byte。

返回值说明	**work_size**：int，模型执行时所需的工作内存的大小，单位Byte。 **weight_size**：int，模型执行时所需权值内存的大小，单位Byte。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	执行和权重内存为Device内存，而且需要用户申请和释放。
注意事项	无

10.6.1.10 函数：set_dynamic_batch_size

C函数原型	**aclError aclmdlSetDynamicBatchSize(uint32_t modelId, aclmdlDataset *dataset, size_t index, uint64_t batchSize)**
Python函数	**ret = acl.mdl.set_dynamic_batch_size(model_id, dataset, index, batch_size)**
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）。
输入说明	**model_id**：int，模型ID。 ● 调用acl.mdl.load_from_file接口/acl.mdl.load_from_mem接口/acl.mdl.load_from_file_with_mem接口/acl.mdl.load_from_mem_with_mem接口加载模型成功后，会返回模型ID。 **dataset**：int，表示模型的输入数据对应的指针对象。 ● 使用acl.model_dataset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**：int，标识动态Batch输入的输入index，需调用acl.mdl.get_input_index_by_name接口获取，动态Batch和动态分辨率输入的名称固定为"ascend_mbatch_shape_data"，动态AIPP输入的名称固定为"ascend_dynamic_aipp_data"。 **batch_size**：int，指定模型推理时的批量大小Batch。 可以调用**acl.mdl.get_dynamic_batch**接口获取指定模型支持的Batch档位数以及每一档中的Batch数。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	调用acl.mdl.get_input_index_by_name接口获取，动态Batch和动态分辨率输入的名称固定为"ascend_mbatch_shape_data"，动态AIPP输入的名称固定为"ascend_dynamic_aipp_data"。
参考资源	接口调用流程与示例，参见**7.6.1 动态Batch/动态分辨率/动态维度（设置多档维度值）**。

10.6.1.11 函数：set_dynamic_hw_size

C函数原型	**aclError aclmdlSetDynamicHWSize(uint32_t modelId, aclmdlDataset *dataset, size_t index, uint64_t height, uint64_t width)**
Python函数	**ret = acl.mdl.set_dynamic_hw_size(model_id, dataset, index, height, width)**
函数功能	设置模型推理时输入图片的高和宽。
输入说明	**model_id**：int，模型ID。调用acl.mdl.load_from_file接口/acl.mdl.load_from_mem接口/acl.mdl.load_from_file_with_mem接口/acl.mdl.load_from_mem_with_mem接口加载模型成功后，会返回模型ID。 **dataset**：表示模型的输入数据。使用acl.model_dataset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**：int，标识动态Batch输入的输入index，需调用acl.mdl.get_input_index_by_name接口获取，动态Batch和动态分辨率输入的名称固定为“ascend_mbatch_shape_data”，动态AIPP输入的名称固定为“ascend_dynamic_aipp_data”。 **height**：int，需设置的H值。 **width**：int，需设置的W值。 可以调用acl.mdl.get_dynamic_hw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	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	调用acl.mdl.get_input_index_by_name接口获取，动态Batch和动态分辨率输入的名称固定为"ascend_mbatch_shape_data"，动态AIPP输入的名称固定为"ascend_dynamic_aipp_data"。
参考资源	接口调用流程与示例，参见7.6.1 动态Batch/动态分辨率/动态维度（设置多档维度值）。

10.6.1.12 函数：create_and_get_op_desc

C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)**

Python函数	**op_name, input_desc, num_inputs, output_desc, num_outputs, ret = acl.mdl.create_and_get_op_desc(device_id, stream_id, task_id, op_name_len)**
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。同步接口。
输入说明	**device_id**：int，指定Device ID。 调用**acl.rt.get_device_id_from_exception_info**接口获取异常信息中的Device ID，作为本接口的输入。 **stream_id**：int，指定Stream ID。 调用**acl.rt.get_stream_id_from_exception_info**接口获取异常信息中的Stream ID，作为本接口的输入。 **task_id**：int，指定Task ID。 调用**acl.rt.get_task_id_from_exception_info**接口获取异常信息中的Task ID，作为本接口的输入。 **op_name_len**：int，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。
返回值说明	**op_name**：string，算子名称字符串指针。 **input_desc**：int，算子所有输入的tensor描述，指向一块连续内存的首地址。 **num_inputs**：int，输入个数。 **output_desc**：int，算子所有输出的tensor描述，指向一块连续内存的首地址。 **num_outputs**：int，输出个数。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

约束说明	**使用场景举例**：执行整网模型推理时，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**acl.rt.set_exception_info_callback** 。 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**acl.rt.get_device_id_from_exception_info**、**acl.rt.get_stream_id_from_exception_info**、**acl.rt.get_task_id_from_exception_info**接口分别获取Device ID、Stream ID、Task ID。 b. 在异常回调函数fn内调用**acl.mdl.create_and_get_op_desc**接口获取算子的描述信息。 c. 在异常回调函数fn内调用**acl.get_tensor_desc_by_index**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**acl.get_tensor_desc_address**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**acl.get_tensor_desc_type**接口获取tensor描述中的数据类型、调用**acl.get_tensor_desc_format**接口获取tensor描述中的Format、调用**acl.get_tensor_desc_num_dims**接口获取tensor描述中的Shape维度个数、调用**acl.get_tensor_desc_dim_v2**接口获取Shape中指定维度的大小。 2. 调用**acl.rt.set_exception_info_callback**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。
注意事项	无
参考资源	接口调用示例，参见**7.10 AI Core异常信息获取**。

10.6.1.13 函数：init_dump

C函数原型	**aclError aclmdlInitDump()**
Python函数	**ret = acl.mdl.init_dump()**
函数功能	Dump初始化。同步接口。
输入说明	无
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。

约束说明	● acl.mdl.init_dump接口需要与acl.mdl.set_dump接口、acl.mdl.finalize_dump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，需要设置不同的的Dump配置信息，接口调用顺序：acl.init接口-->acl.mdl.init_dump接口-->acl.mdl.set_dump接口-->模型加载-->模型执行-->acl.mdl.finalize_dump接口-->模型卸载-->acl.mdl.init_dump接口-->acl.mdl.set_dump接口-->模型加载-->模型执行-->acl.mdl.finalize_dump接口-->模型卸载-->执行其它任务-->acl.finalize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：acl.init接口-->acl.mdl.init_dump接口-->acl.mdl.set_dump接口-->模型加载-->模型执行-->acl.mdl.finalize_dump接口-->模型卸载-->模型加载-->模型执行-->执行其它任务-->acl.finalize接口 ● 如果已经通过acl.init接口配置了dump信息，则调用acl.mdl.init_dump接口时会返回失败。 ● 必须在调用acl.init接口之后、模型加载接口之前调用acl.mdl.init_dump接口。
注意事项	无
相关接口	pyACL还提供了**acl.init**接口，在pyACL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**acl.init**接口，如果要修改Dump配置信息，需修改*.json文件中的配置。

10.6.1.14 函数：set_dump

C函数原型	**aclError aclmdlSetDump(const char *dumpCfgPath)**
Python函数	**ret = acl.mdl.set_dump(dump_cfg_path)**
函数功能	设置dump参数。同步接口。
输入说明	**dump_cfg_path**：string，配置文件所在的路径，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《**精度比对工具使用指南**》中的“比对数据准备>准备离线模型dump数据文件”。如果不涉及配置信息，需向acl.mdl.set_dump接口中不加参数。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。

约束说明	● acl.mdl.init_dump接口需要与acl.mdl.set_dump接口、acl.mdl.finalize_dump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，需要设置不同的的Dump配置信息，接口调用顺序：acl.init接口-->acl.mdl.init_dump接口-->acl.mdl.set_dump接口-->模型加载-->模型执行-->acl.mdl.finalize_dump接口-->模型卸载-->acl.mdl.init_dump接口-->acl.mdl.set_dump接口-->模型加载-->模型执行-->acl.mdl.finalize_dump接口-->模型卸载-->执行其它任务-->acl.finalize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：acl.init接口-->acl.mdl.init_dump接口-->acl.mdl.set_dump接口-->模型加载-->模型执行-->acl.mdl.finalize_dump接口-->模型卸载-->模型加载-->模型执行-->执行其它任务-->acl.finalize接口 ● 如果已经通过acl.init接口配置了dump信息，则调用acl.mdl.init_dump接口时会返回失败。 ● 必须在调用acl.init接口之后、模型加载接口之前调用acl.mdl.init_dump接口。
注意事项	无
相关接口	pyACL还提供了**acl.init**接口，在pyACL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**acl.init**接口，如果要修改Dump配置信息，需修改*.json文件中的配置。

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":"/home/HwHiAiUser/ide_daemon/dump",
            "dump_mode":"output"
   }

}
```

10.6.1.15 函数：finalize_dump

C函数原型	**aclError aclmdlFinalizeDump()**
Python函数	**ret = acl.mdl.finalize_dump()**
函数功能	Dump去初始化。同步接口。
输入说明	无
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	● acl.mdl.init_dump接口需要与acl.mdl.set_dump接口、acl.mdl.finalize_dump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，需要设置不同的的Dump配置信息，接口调用顺序：acl.init接口-->acl.mdl.init_dump接口-->acl.mdl.set_dump接口-->模型加载-->模型执行-->acl.mdl.finalize_dump接口-->模型卸载-->acl.mdl.init_dump接口-->acl.mdl.set_dump接口-->模型加载-->模型执行-->acl.mdl.finalize_dump接口-->模型卸载-->执行其它任务-->acl.finalize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：acl.init接口-->acl.mdl.init_dump接口-->acl.mdl.set_dump接口-->模型加载-->模型执行-->acl.mdl.finalize_dump接口-->模型卸载-->模型加载-->模型执行-->执行其它任务-->acl.finalize接口 ● 如果已经通过acl.init接口配置了dump信息，则调用acl.mdl.init_dump接口时会返回失败。 ● 必须在调用acl.init接口之后、模型加载接口之前调用acl.mdl.init_dump接口。
注意事项	无
相关接口	pyACL还提供了acl.init接口，在pyACL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acl.init接口，如果要修改Dump配置信息，需修改*.json文件中的配置。

10.6.1.16 函数：get_aipp_type

C函数原型	**aclError aclmdlGetAippType(uint32_t modelId, size_t index, aclmdlInputAippType *type, size_t *dynamicAttachedDataIndex)**

Python函数	**type, dynamic_attached_data_index, ret = acl.mdl.get_aipp_type(model_id, index)**
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。
输入说明	**model_id**：int，指定模型的ID。 调用acl.mdl.load_from_file接口/acl.mdl.load_from_mem接口/acl.mdl.load_from_file_with_mem接口/acl.mdl.load_from_mem_with_mem接口加载模型成功后，会返回模型ID。 **index**：int，模型中输入的index。
返回值说明	**type**：int，表示指定模型输入的AIPP类型。 **dynamic_attached_data_index**：int， ● 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。 – 0：ACL_DATA_WITHOUT_AIPP #该输入无AIPP信息 – 1：ACL_DATA_WITH_STATIC_AIPP #该输入带静态AIPP信息 – 2：ACL_DATA_WITH_DYNAMIC_AIPP #该输入有关联的动态AIPP输入 – 3：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 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无
参考资源	接口调用示例，参见**7.7.3 动态AIPP（多个动态AIPP输入）**。

10.6.1.17 函数：set_aipp_by_input_index

C函数原型	**aclError aclmdlSetAIPPByInputIndex(uint32_t modelId, aclmdlDataset *dataset, size_t index, const aclmdlAIPP *aippParmsSet)**
Python函数	**ret = acl.mdl.set_aipp_by_input_index(model_id, dataset_in, index, aipp)**
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，同步接口。 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->减均值/归一化->padding

输入说明	**model_id**：int，模型ID。 调用acl.mdl.load_from_file接口/acl.mdl.load_from_mem接口/acl.mdl.load_from_file_with_mem接口/acl.mdl.load_from_mem_with_mem接口加载模型成功后，会返回模型ID。 **dataset_in**：int，表示模型推理的输入数据的指针地址。 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**：int，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**acl.mdl.get_aipp_type**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**acl.mdl.get_aipp_type**接口的index参数值保持一致；若无，则无需设置动态AIPP参数。 **aipp**：int，动态AIPP参数对象的指针地址。 提前调用**acl.mdl.create_aipp**接口创建aclmdlAIPP类型的数据。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
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10-1**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**acl.mdl.set_aipp_srcImage_size**、**acl.mdl.set_aipp_scf_params**、**acl.mdl.set_aipp_crop_params**、**acl.mdl.set_aipp_padding_params**接口的入参： 见**表10-1**。
注意事项	无
参考资源	接口调用示例，参见**7.7.3 动态AIPP（多个动态AIPP输入）**。

表 10-1 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

抠图	缩放	补边（padding）	动态AIPP输出图片的宽、高
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
是	是	是	aippOutputW=scfOutputSizeW + paddingSizeLeft + paddingSizeRight，aippOutputH=scfOutputSizeH + paddingSizeTop + paddingSizeBottom

10.6.1.18 函数：set_input_dynamic_dims

C函数原型	**aclError aclmdlSetInputDynamicDims(uint32_t modelId, aclmdlDataset *dataset, size_t index, const aclmdlIODims *dims)**
Python函数	**ret = acl.mdl.set_input_dynamic_dims(model_id, dataset_in, index, dims)**
函数功能	如果模型输入的Shape是动态的，可以调用该接口设置模型推理时具体维度的值。

输入说明	**model_id**：int，模型ID。 调用acl.mdl.load_from_file接口/acl.mdl.load_from_mem接口/acl.mdl.load_from_file_with_mem接口/acl.mdl.load_from_mem_with_mem接口加载模型成功后，会返回模型ID。 **dataset_in**：int，表示模型的输入数据。 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**：int，标识动态维度的输入index。 需调用acl.mdl.get_input_index_by_name接口获取，输入名称固定为"ascend_mbatch_shape_data"。 **dims**：dict，指定具体某一档的所有维度信息。 可以调用**acl.mdl.get_input_dynamic_dims**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_shap="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输入示例如下（name暂不使用）： dims = {'dimCount': 8, 'name': '', 'dims': [1,1,40,20,1,20,1,1]}
返回值说明	**ret**：int，错误码 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无
参考资源	接口调用流程与示例，参见7.6.1 动态Batch/动态分辨率/动态维度（设置多档维度值）。

10.6.1.19 函数：create_config_handle

C函数原型	**aclmdlConfigHandle *aclmdlCreateConfigHandle()**
Python函数	**config_handle = acl.mdl.create_config_handle()**
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。同步接口。 如需销毁aclmdlConfigHandle类型的数据，请参见**acl.mdl.destroy_config_handle**。
输入说明	无

返回值说明	**config_handle**：int，aclmdlConfigHandle类型的指针对象。 ● 返回非0表示成功。 ● 返回0表示失败。
约束说明	无
注意事项	无

10.6.1.20 函数：destroy_config_handle

C函数原型	**aclError aclmdlDestroyConfigHandle(aclmdlConfigHandle *handle)**
Python函数	**ret = acl.mdl.destroy_config_handle(config_handle)**
函数功能	销毁通过**acl.mdl.create_config_handle**接口创建的aclmdlConfigHandle类型的数据。同步接口。
输入说明	**config_handle**：int，待销毁的aclmdlConfigHandle类型的指针对象。
返回值说明	**ret**：int，错误码 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无

10.6.1.21 函数：set_config_opt

C函数原型	**aclError aclmdlSetConfigOpt(aclmdlConfigHandle *handle, aclmdlConfigAttr attr, const void *attrValue, size_t valueSize)**
Python函数	**ret = acl.mdl.set_config_opt(config_handle, attr, attrValue)**
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。
输入说明	**config_handle**：int，指定模型加载的配置对象。需提前调用**acl.mdl.create_config_handle**接口创建该对象。 **attr**：int，指定需设置的属性，取值参考**aclmdlConfigAttr**。 **attrValue**：int，指定设置的属性值。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。

约束说明	按如下接口调用顺序可实现模型加载的功能： 1. 调用acl.mdl.create_config_handle接口创建模型加载的配置对象。 2. 多次调用acl.mdl.set_config_opt接口设置配置对象中每个属性的值。 3. 调用acl.mdl.load_with_config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acl.mdl.destroy_config_handle接口销毁。
注意事项	无
相关接口	使用acl.mdl.set_config_opt接口、acl.mdl.load_with_config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pyACL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。 ● acl.mdl.load_from_file接口 ● acl.mdl.load_from_mem接口 ● acl.mdl.load_from_file_with_mem接口 ● acl.mdl.load_from_mem_with_mem接口

10.6.1.22 函数：load_with_config

C函数原型	**aclError aclmdlLoadWithConfig(const aclmdlConfigHandle *handle, uint32_t *modelId)**
Python函数	**model_id, ret = acl.mdl.load_with_config(config_handle)**
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。
输入说明	**config_handle**：int，指定模型加载的配置对象。需提前调用acl.mdl.create_config_handle接口创建该对象，与acl.mdl.set_config_opt中的config_handle保持一致。
返回值说明	**model_id**：int，系统完成模型加载后生成的模型ID。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

约束说明	按如下接口调用顺序可实现模型加载的功能： 1. 调用**acl.mdl.create_config_handle**接口创建模型加载的配置对象。 2. 多次调用**acl.mdl.set_config_opt**接口设置配置对象中每个属性的值。 3. 调用**acl.mdl.load_with_config**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**acl.mdl.destroy_config_handle**接口销毁。
注意事项	无
相关接口	使用**acl.mdl.set_config_opt**接口、**acl.mdl.load_with_config**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pyACL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。 ● **acl.mdl.load_from_file**接口 ● **acl.mdl.load_from_mem**接口 ● **acl.mdl.load_from_file_with_mem**接口 ● **acl.mdl.load_from_mem_with_mem**接口

10.6.1.23 函数：get_tensor_real_name

C函数原型	**const char *aclmdlGetTensorRealName(const aclmdlDesc *modelDesc, const char *name)**
Python函数	**real_name = acl.mdl.get_tensor_real_name(model_desc, name)**
函数功能	根据指定名称获取tensor的真实名称。同步接口。 acl.mdl.get_tensor_real_name接口需要与**acl.mdl.get_input_dims**/**acl.mdl.get_input_dims_v2**/**acl.mdl.get_output_dims**/**acl.mdl.get_cur_output_dims**接口配合使用。
输入说明	**model_desc**：int，aclmdlDesc类型的指针对象。 需提前调用**acl.mdl.create_desc**接口创建aclmdlDesc类型的数据。 **name**：str，指定名称，用于根据该名称获取tensor的真实名称。
返回值说明	**real_name**：str，返回指向tensor真实名称的指针，该指针的生命周期与model_desc相同，若model_desc资源被销毁，则该指针指向的内容也会自动被销毁。 若model_desc或name为空，则返回0。
约束说明	无
注意事项	无

10.6.2 aclmdlAIPP

10.6.2.1 函数：create_aipp

C函数原型	**aclmdlAIPP *aclmdlCreateAIPP(uint64_t batchSize)**
Python函数	**aipp_ptr = acl.mdl.create_aipp(batch_size)**
函数功能	动态AIPP场景下，根据模型支持的Batch数创建的acl.mdl.create_aipp类型的数据，用于存放动态AIPP的参数。同步接口。
输入说明	**batch_size**：int，模型的Batch数。
返回值说明	**aipp_ptr**：int，aclmdlAIPP类型数据对象地址。
约束说明	无
注意事项	无

10.6.2.2 函数：destroy_aipp

C函数原型	**aclError aclmdlDestroyAIPP(const aclmdlAIPP *aippParmsSet)**
Python函数	**ret = acl.mdl.destroy_aipp(aipp_parms_set)**
函数功能	销毁aclmdlAIPP类型的数据。同步接口。
输入说明	**aipp_parms_set**：int，待销毁的aclmdlAIPP数据类型对象的指针。
返回值说明	**ret**：int，错误码 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	销毁类接口：调用该接口后，不能继续使用已释放或销毁的资源，建议调用该接口后，将相关资源设置为无效值（例如，置为0或None）。
注意事项	无

10.6.2.3 设置动态 AIPP 参数

10.6.2.3.1 函数：set_aipp_input_format

C函数原型	**aclError aclmdlSetAIPPInputFormat(aclmdlAIPP *aippParmsSet, AippInputFormat inputFormat)**
Python函数	**ret = acl.mdl.set_aipp_input_format(aipp_parms_set, input_format)**

函数功能	动态AIPP场景下，必须设置原始输入图像的格式，同步接口。
输入说明	**aipp_parms_set**：int，动态AIPP参数对象的指针。 提前调用acl.mdl.create_aipp接口创建aclmdlAIPP类型的数据。 **input_format**：int，表示原始输入图像的格式。 取值范围如下： ● 1：ACL_YUV420SP_U8 ● 2：ACL_XRGB8888_U8 ● 3：ACL_RGB888_U8 ● 4：ACL_YUV400_U8
返回值说明	**ret**：int，错误码 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.6.2.3.2 函数：set_aipp_csc_params

C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)**
Python函数	**ret = acl.mdl.set_aipp_csc_params(aipp_parms_set, csc_switch, csc_matrix_R0C0, csc_matrix_R0C1, csc_matrix_R0C2,csc_matrix_R1C0, csc_matrix_R1C1, csc_matrix_R1C2, csc_matrix_R2C0, csc_matrix_R2C1, csc_matrix_R2C2, csc_output_bias_R0, csc_output_bias_R1, csc_output_bias_R2, csc_input_bias_R0, csc_input_bias_R1, csc_input_bias_R2)**

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，则调用该接口设置的参数无效。同步接口。 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，您可以参见《**ATC工具使用指南**》下的色域转换配置说明，按照转换前图片格式、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。

输入说明	**aipp_parms_set**：int，动态AIPP参数对象的指针。 提前调用acl.mdl.create_aipp接口创建aclmdlAIPP类型的数据。 **csc_switch**：int，色域转换开关，取值范围： ● 0：关闭色域转换开关，默认为0。 ● 1：打开色域转换开关。 **csc_matrix_R0C0**：int，色域转换矩阵参数。 ● 取值范围：[-32677 ,32676] **csc_matrix_R0C1**：int，色域转换矩阵参数。 ● 取值范围：[-32677 ,32676] **csc_matrix_R0C2**：int，色域转换矩阵参数。 ● 取值范围：[-32677 ,32676] **csc_matrix_R1C0**：int，色域转换矩阵参数。 ● 取值范围：[-32677 ,32676] **csc_matrix_R1C1**：int，色域转换矩阵参数。 ● 取值范围：[-32677 ,32676] **csc_matrix_R1C2**：int，色域转换矩阵参数。 ● 取值范围：[-32677 ,32676] **csc_matrix_R2C0**：int，色域转换矩阵参数。 ● 取值范围：[-32677 ,32676] **csc_matrix_R2C1**：int，色域转换矩阵参数。 ● 取值范围：[-32677 ,32676] **csc_matrix_R2C2**：int，色域转换矩阵参数。 ● 取值范围：[-32677 ,32676] **csc_output_bias_R0**：int，RGB转YUV时的输出偏移，默认值为0。 ● 取值范围：[0, 255] **csc_output_bias_R1**：int，RGB转YUV时的输出偏移，默认值为0。 ● 取值范围：[0, 255] **csc_output_bias_R2**：int，RGB转YUV时的输出偏移，默认值为0。 ● 取值范围：[0, 255] **csc_input_bias_R0**：int，YUV转RGB时的输入偏移，默认值为0。 ● 取值范围：[0, 255] **csc_input_bias_R1**：int，YUV转RGB时的输入偏移，默认值为0。 ● 取值范围：[0, 255] **csc_input_bias_R2**：int，YUV转RGB时的输入偏移，默认值为0。 ● 取值范围：[0, 255]
返回值说明	**ret**：int，错误码 ● 返回0表示成功。 ● 返回**其它值**表示失败。

约束说明	如果通过acl.mdl.set_aipp_input_format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。
注意事项	无

10.6.2.3.3 函数：set_aipp_rbuv_swap_switch

C函数原型	**aclError aclmdlSetAIPPRbuvSwapSwitch(aclmdlAIPP *aippParmsSet, int8_t rbuvSwapSwitch)**
Python函数	**ret = acl.mdl.set_aipp_rbuv_swap_switch(aipp_parms_set, rbuv_swap_switch)**
函数功能	动态AIPP场景下，CSC色域转换前，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，同步接口。
输入说明	**aipp_parms_set**：int，动态AIPP参数对象的指针。 ● 提前调用acl.mdl.create_aipp接口创建aclmdlAIPP类型的数据。 **rbuv_swap_switch**：int，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，默认为0。 ● 1：交换。
返回值说明	**ret**：int，错误码 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.6.2.3.4 函数：set_aipp_ax_swap_switch

C函数原型	**aclError aclmdlSetAIPPAxSwapSwitch(aclmdlAIPP *aippParmsSet, int8_t axSwapSwitch)**
Python函数	**ret = acl.mdl.set_aipp_ax_swap_switch(aipp_parms_set, ax_swap_switch)**
函数功能	动态AIPP场景下，设置RGBA->ARGB或者YUVA->AYUV的交换开关，同步接口。

输入说明	**aipp_parms_set**：int，动态AIPP参数对象的指针。 ● 提前调用acl.mdl.create_aipp接口创建aclmdlAIPP类型的数据。 **ax_swap_switch**：int，表示GBA->ARGB或者YUVA->AYUV的交换开关。取值范围： ● 0：不交换，默认为0。 ● 1：交换。
返回值说明	**ret**：int，错误码 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.6.2.3.5 函数：set_aipp_src_image_size

C函数原型	**aclError aclmdlSetAIPPSrcImageSize(aclmdlAIPP *aippParmsSet, int32_t srcImageSizeW, int32_t srcImageSizeH)**
Python函数	**ret = acl.mdl.set_aipp_src_image_size(aipp_parms_set, src_image_size_w, src_image_size_h)**
函数功能	动态AIPP场景下，必须设置原始图片的宽和高，同步接口。
输入说明	**aipp_parms_set**：int，动态AIPP参数对象。 ● 提前调用acl.mdl.create_aipp接口创建aclmdlAIPP类型的数据。 **src_image_size_w**：int，原始图片的宽，对于YUV420SP_U8类型的图像，要求取值是偶数。 ● 取值范围： [2,4096] **src_image_size_h**：int，原始图片的高，对于YUV420SP_U8类型的图像，要求取值是偶数。 ● 取值范围： [1,4096]
返回值说明	**ret**：int，错误码 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.6.2.3.6 函数：set_aipp_crop_params

C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)**
Python函数	**ret = acl.mdl.set_aipp_crop_params(aipp_parms_set, crop_switch, crop_start_pos_w, crop_start_pos_h, crop_size_w, crop_size_h, batch_index)**
函数功能	动态AIPP场景下，设置抠图相关的参数，同步接口。
输入说明	**aipp_parms_set**：int，动态AIPP参数对象的指针。 ● 提前调用acl.mdl.create_aipp接口创建aclmdlAIPP类型的数据。 **crop_switch**：int，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。 **crop_start_pos_w**：int，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。对于YUV422SP_U8/YUYV_U8格式的图像，参数取值要求是偶数。 ● 取值范围：[0,4096] **crop_start_pos_h**：int，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：[0,4096] **crop_size_w**：int，抠图区域的宽度。对于YUV420SP_U8格式的图像，参数取值要求是偶数。 ● 取值范围：[1,4096] **crop_size_h**：int，抠图区域的高度。对于YUV420SP_U8格式的图像，参数取值要求是偶数。 ● 取值范围：[1,4096] **batch_index**：int，指定对第几个Batch上的图片执行抠图操作，默认为0。 ● 取值范围：[0,batch_size)，batch_size是在调用acl.mdl.create_aipp接口创建aclmdlAIPP类型的数据时设置。
返回值说明	**ret**：int，错误码 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	若开启抠图功能，则通过acl.mdl.set_aipp_src_image_size接口设置的参数与通过acl.mdl.set_aipp_crop_params接口设置的参数之间必须满足以下公式： ● srcImageSizeW≥cropSizeW+cropStartPosW ● srcImageSizeH≥cropSizeH+cropStartPosH

注意事项	无

10.6.2.3.7 函数：set_aipp_padding_params

C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)**
Python函数	**ret = acl.mdl.set_aipp_padding_params(aipp_parms_set, padding_switch, padding_size_top, padding_size_bottom, padding_size_left, padding_size_right, batch_index)**
函数功能	动态AIPP场景下，设置补边相关的参数，同步接口。
输入说明	**aipp_parms_set**：int，动态AIPP参数对象的指针。 ● 提前调用acl.mdl.create_aipp接口创建aclmdlAIPP类型的数据。 **padding_switch**：int，是否对图片执行补边操作。取值范围： ● 0：不执行补边操作，默认为0。 ● 1：执行补边操作。 **padding_size_top**：int，在图片上方填充的值。 ● 取值范围：[0, 32] **padding_size_bottom**：int，在图片下方填充的值。 ● 取值范围：[0, 32] **padding_size_left**：int，在图片左方填充的值。 ● 取值范围：[0, 32] **padding_size_right**：int，在图片右方填充的值。 ● 取值范围：[0, 32] **batch_index**：int，指定对第几个Batch上的图片执行补边操作，默认为0。 ● 取值范围：[0,batch_size)，batch_size是在调用acl.mdl.create_aipp接口创建aclmdlAIPP类型的数据时设置。
返回值说明	**ret**：int，错误码 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	补边之后，图片的宽必须小于等于1080。
注意事项	无

10.6.2.3.8 函数：set_aipp_dtc_pixel_mean

C函数原型	**aclError aclmdlSetAIPPDtcPixelMean(aclmdlAIPP *aippParmsSet, int16_t dtcPixelMeanChn0, int16_t dtcPixelMeanChn1, int16_t dtcPixelMeanChn2, int16_t dtcPixelMeanChn3, uint64_t batchIndex)**
Python函数	**ret = acl.mdl.set_aipp_dtc_pixel_mean(aipp_parms_set, dtc_pixel_mean_chn0, dtc_pixel_mean_chn1, dtc_pixel_mean_chn2, dtc_pixel_mean_chn3, batch_index)**
函数功能	动态AIPP场景下，设置通道的均值，同步接口。
输入说明	**aipp_parms_set**：int，动态AIPP参数对象的指针。 ● 提前调用acl.mdl.create_aipp接口创建aclmdlAIPP类型的数据。 **dtc_pixel_mean_chn0**：int，通道0的均值。 ● 取值范围：[0, 255] **dtc_pixel_mean_chn1**：int，通道1的均值。 ● 取值范围：[0, 255] **dtc_pixel_mean_chn2**：int，通道2的均值。 ● 取值范围：[0, 255] **dtc_pixel_mean_chn3**：int，通道3的均值。如果只有3个通道，则该参数传默认值0。 ● 取值范围：[0, 255] **batch_index**：int，指定对第几个Batch上的图片设置通道均值，默认为0。 ● 取值范围：[0,batch_size)，batch_size是在调用acl.mdl.create_aipp接口创建aclmdlAIPP类型的数据时设置。
返回值说明	**ret**：int，错误码 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.6.2.3.9 函数：set_aipp_dtc_pixel_min

C函数原型	**aclError aclmdlSetAIPPDtcPixelMin(aclmdlAIPP *aippParmsSet, float dtcPixelMinChn0, float dtcPixelMinChn1, float dtcPixelMinChn2, float dtcPixelMinChn3, uint64_t batchIndex)**
Python函数	**ret = acl.mdl.set_aipp_dtc_pixel_min(aipp_parms_set, dtc_pixel_mean_chn0, dtc_pixel_mean_chn1, dtc_pixel_mean_chn2, dtc_pixel_mean_chn3, batch_index)**
函数功能	动态AIPP场景下，设置通道的最小值，同步接口。

输入说明	**aipp_parms_set**：int，动态AIPP参数对象的指针。 ● 提前调用acl.mdl.create_aipp接口创建aclmdlAIPP类型的数据。 **dtc_pixel_mean_chn0**：int，通道0的均值。 ● 取值范围：[0, 255] **dtc_pixel_mean_chn1**：int，通道1的均值。 ● 取值范围：[0, 255] **dtc_pixel_mean_chn2**：int，通道2的均值。 ● 取值范围：[0, 255] **dtc_pixel_mean_chn3**：int，通道3的均值。如果只有3个通道，则该参数传默认值0。 ● 取值范围：[0, 255] **batch_index**：int，指定对第几个Batch上的图片设置通道均值，默认为0。 ● 取值范围：[0,batch_size)，batch_size是在调用acl.mdl.create_aipp接口创建aclmdlAIPP类型的数据时设置。
返回值说明	**ret**：int，错误码 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.6.2.3.10 函数：set_aipp_pixel_var_reci

C函数原型	**aclError aclmdlSetAIPPPixelVarReci(aclmdlAIPP *aippParmsSet, float dtcPixelVarReciChn0, float dtcPixelVarReciChn1, float dtcPixelVarReciChn2, float dtcPixelVarReciChn3, uint64_t batchIndex)**
Python函数	**ret = acl.mdl.set_aipp_pixel_var_reci(aipp_parms_set, dtc_pixel_var_reci_chn0, dtc_pixel_var_reci_chn1, dtc_pixel_var_reci_chn2, dtc_pixel_var_reci_chn3, batch_index)**
函数功能	动态AIPP场景下，设置通道的方差，同步接口。

输入说明	**aipp_parms_set**：int，动态AIPP参数对象。 ● 提前调用acl.mdl.create_aipp接口创建aclmdlAIPP类型的数据。 **dtc_pixel_var_reci_chn0**：int，通道0的方差，默认值为1.0。 ● 取值范围：[-65504, 65504] **dtc_pixel_var_reci_chn1**：int，通道1的方差，默认值为1.0。 ● 取值范围：[-65504, 65504] **dtc_pixel_var_reci_chn2**：int，通道2的方差，默认值为1.0。 ● 取值范围：[-65504, 65504] **dtc_pixel_var_reci_chn3**：int，通道3的方差。如果只有3个通道，则该参数传默认值1.0。 ● 取值范围：[-65504, 65504] **batch_index**：int，指定对第几个Batch上的图片设置通道的方差。 ● 取值范围：[0,batch_size)，batch_size是在调用acl.mdl.create_aipp接口创建aclmdlAIPP类型的数据时设置。
返回值说明	**ret**：int，错误码 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.6.2.3.11 函数：set_input_aipp

C函数原型	**aclError aclmdlSetInputAIPP(uint32_t modelId, aclmdlDataset *dataset, size_t index, const aclmdlAIPP *aippParmsSet)**
Python函数	**ret = acl.mdl.set_input_aipp(model_id, dataset, index, aipp_parms_set)**
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，同步接口。 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。

输入说明	**model_id**：int，模型的ID。 ● 调用acl.mdl.load_from_file接口/acl.mdl.load_from_mem接口/acl.mdl.load_from_file_with_mem接口/acl.mdl.load_from_mem_with_mem接口加载模型成功后，会返回模型ID。 **dataset**：int，表示模型的输入数据。 ● 使用acl.model_dataset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**：int，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acl.mdl.get_aipp_type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acl.mdl.get_input_index_by_name接口获取，输入名称固定为"ascend_dynamic_aipp_data"。 **aipp_parms_set**：int，动态AIPP参数对象的指针。 ● 提前调用acl.mdl.create_aipp接口创建aclmdlAIPP类型的数据。
返回值说明	**ret**：int，错误码 ● 返回0表示成功。 ● 返回其它值表示失败。
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10-2**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**acl.mdl.set_aipp_src_image_size**、**acl.mdl.set_aipp_crop_params**、**acl.mdl.set_aipp_padding_params**接口的入参。
注意事项	无
参考资源	接口调用流程与示例，参见**7.7.2 动态AIPP（单个动态AIPP输入）**。

表 10-2 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

抠图	缩放	补边（padding）	动态AIPP输出图片的宽、高
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
是	是	是	aippOutputW=scfOutputSizeW + paddingSizeLeft + paddingSizeRight，aippOutputH=scfOutputSizeH + paddingSizeTop + paddingSizeBottom

10.6.3 aclmdlDataset

10.6.3.1 总体说明

您可以通过以下接口获取aclmdlDataset类型的数据、并对aclmdlDataset的数据执行一些操作。该数据类型下的操作接口都是同步接口。

aclmdlDataset主要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。

10.6.3.2 函数：create_dataset

C函数原型	**aclmdlDataset *aclmdlCreateDataset()**
Python函数	**mdl_dataset = acl.mdl.create_dataset()**
函数功能	创建aclmdlDataset类型的数据。
输入说明	无
返回值说明	**mdl_dataset**：int，aclmdlDataset地址。
约束说明	无
注意事项	无

10.6.3.3 函数：destroy_dataset

C函数原型	**aclError aclmdlDestroyDataset(const aclmdlDataset *dataset)**
Python函数	**ret = acl.mdl.destroy_dataset(dataset)**
函数功能	销毁aclmdlDataset类型的数据。
输入说明	**dataset**：int，待销毁的aclmdlDataset的指针。
返回值说明	**ret**：int，错误码 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	销毁类接口：调用该接口后，不能继续使用已释放或销毁的资源，建议调用该接口后，将相关资源设置为无效值（例如，置为0或None）。
注意事项	无

10.6.3.4 函数：add_dataset_buffer

C函数原型	**aclError aclmdlAddDatasetBuffer(aclmdlDataset *dataset, aclDataBuffer *dateBuffer)**
Python函数	**dataset, ret = acl.mdl.add_dataset_buffer(dataset, date_buffer)**
函数功能	向aclmdlDataset中增加aclDataBuffer。
输入说明	**dataset**：int，待增加aclDataBuffer的aclmdlDataset地址。 **date_buffer**：int，待增加的aclDataBuffer地址。

返回值说明	**dataset**：int，待增加aclDataBuffer的aclmdlDataset地址。 **ret**：int，错误码 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	dataset需要带入和返回。

10.6.3.5 函数：get_dataset_num_buffers

C函数原型	**size_t aclmdlGetDatasetNumBuffers(const aclmdlDataset *dataset)**
Python函数	**index = acl.mdl.get_dataset_num_buffers(dataset)**
函数功能	获取aclmdlDataset中aclDataBuffer的个数。
输入说明	**dataset**：int，aclmdlDataset数据对象的指针。
返回值说明	**index**：int，aclDataBuffer数据类型的个数。
约束说明	无
注意事项	无

10.6.3.6 函数：get_dataset_buffer

C函数原型	**aclDataBuffer* aclmdlGetDatasetBuffer(const aclmdlDataset *dataset, size_t index)**
Python函数	**data_buffer = acl.mdl.get_dataset_buffer(dataset, index)**
函数功能	获取aclmdlDataset中的第n个aclDataBuffer。
输入说明	**dataset**：int，aclmdlDataset数据对象的指针。 **index**：int，表明获取的是第几个aclDataBuffer。
返回值说明	**data_buffer**：int，aclDataBuffer数据对象指针。 ● 获取成功，返回aclDataBuffer的地址。 ● 获取失败返回空地址。
约束说明	无
注意事项	无

10.6.3.7 函数：set_dataset_tensor_desc

C函数原型	**aclError aclmdlSetDatasetTensorDesc(aclmdlDataset *dataset, aclTensorDesc *tensorDesc, size_t index)**
Python函数	**dataset, ret = acl.mdl.set_dataset_tensor_desc(dataset, tensorDesc, index)**
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，同步接口。 **该接口预留，当前版本暂不支持。**
输入说明	**dataset**：待增加aclTensorDesc的aclmdlDataset地址，表示模型执行的输入或输出数据结构。 需提前调用**acl.mdl.create_dataset**接口创建aclmdlDataset类型的数据，再调用**acl.mdl.add_dataset_buffer**接口向aclmdlDataset中增加aclDataBuffer。 **tensorDesc**：待增加的aclTensorDesc地址，表示模型执行时对应的输入或输出的tensor描述。 需提前调用**acl.create_tensor_desc**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**acl.create_tensor_desc**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，模型构建的详细说明请参见模型构建。 **index**：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**acl.mdl.get_input_name_by_index**、**acl.mdl.get_output_name_by_index**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	**dataset**：待增加aclTensorDesc的aclmdlDataset地址，表示模型执行的输入或输出数据结构。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	对同一个模型，**acl.mdl.set_dynamic_batch_size**接口、**acl.mdl.set_dynamic_hw_size**接口、**acl.mdl.set_input_dynamic_dims**接口、**acl.mdl.set_dataset_tensor_desc**接口，只能调用其中一个接口。

10.6.3.8 函数：get_dataset_tensor_desc

C函数原型	**aclTensorDesc *aclmdlGetDatasetTensorDesc(const aclmdlDataset *dataset, size_t index)**
Python函数	**tensorDesc =acl.mdl.get_dataset_tensor_desc(dataset, index)**

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。 **该接口预留，当前版本暂不支持。** **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**acl.mdl.set_dataset_tensor_desc**设置该输入的tensor描述信息，在模型执行之后，调用acl.mdl.get_dataset_tensor_desc接口获取模型动态输出的Tensor描述信息。
输入说明	**dataset**：表示模型执行的输入或输出数据结构。 **index**：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**acl.mdl.get_input_name_by_index**、**acl.mdl.get_output_name_by_index**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	**tensorDesc**：返回指定输入或输出的tensor描述信息。

10.6.4 aclmdlDesc

10.6.4.1 函数：create_desc

C函数原型	**aclmdlDesc* aclmdlCreateDesc()**
Python函数	**desc = acl.mdl.create_desc()**
函数功能	创建aclmdlDesc类型的数据。同步接口。
输入说明	无
返回值说明	**desc**：int，aclmdlDesc类型对象的地址。
约束说明	无
注意事项	无

10.6.4.2 函数：destroy_desc

C函数原型	**aclError aclmdlDestroyDesc(aclmdlDesc *modelDesc)**
Python函数	**ret = acl.mdl.destroy_desc(model_desc)**
函数功能	销毁aclmdlDesc类型的数据。同步接口。
输入说明	**model_desc**：int，待销毁的aclmdlDesc的指针。

返回值说明	**ret**：int，错误码 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	销毁类接口：调用该接口后，不能继续使用已释放或销毁的资源，建议调用该接口后，将相关资源设置为无效值（例如，置为0或None）。
注意事项	无

10.6.4.3 函数：get_desc

C函数原型	**aclError aclmdlGetDesc(aclmdlDesc *modelDesc, uint32_t modelId)**
Python函数	**ret = acl.mdl.get_desc(model_desc, model_id)**
函数功能	根据模型ID获取该模型的aclmdlDesc类型数据。同步接口。
输入说明	**model_desc**：int，aclmdlDesc类型数据的指针。 **model_id**：int，模型ID。
返回值说明	**ret**：int，错误码 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.6.4.4 函数：get_num_inputs

C函数原型	**size_t aclmdlGetNumInputs(aclmdlDesc *modelDesc)**
Python函数	**size = acl.mdl.get_num_inputs(model_desc)**
函数功能	根据aclmdlDesc类型的数据，获取模型的输入个数。同步接口。
输入说明	**model_desc**：int，aclmdlDesc类型数据的指针。
返回值说明	**size**：int，模型的输入个数。
约束说明	无
注意事项	无

10.6.4.5 函数：get_num_outputs

C函数原型	**size_t aclmdlGetNumOutputs(aclmdlDesc *modelDesc)**
Python函数	**size = acl.mdl.get_num_outputs(model_desc)**
函数功能	根据aclmdlDesc类型的数据，获取模型的输出个数。同步接口。
输入说明	**model_desc**：int，aclmdlDesc类型数据的指针。
返回值说明	**size**：int，模型的输出个数。
约束说明	无
注意事项	无

10.6.4.6 函数：get_input_size_by_index

C函数原型	**size_t aclmdlGetInputSizeByIndex(aclmdlDesc *modelDesc, size_t index)**
Python函数	**size = acl.mdl.get_input_size_by_index(model_desc, index)**
函数功能	根据aclmdlDesc类型的数据，获取指定输入的大小，单位为Byte。同步接口。
输入说明	**model_desc**：int，aclmdlDesc类型数据的指针。 需提前调用**acl.mdl.create_desc**接口创建aclmdlDesc类型的数据。 **index**：int，指定获取第几个输入的大小，index值从0开始。
返回值说明	**size**：int，返回指定输入的大小，单位是Byte。 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。
约束说明	无
注意事项	无

10.6.4.7 函数：get_output_size_by_index

C函数原型	**size_t aclmdlGetOutputSizeByIndex(aclmdlDesc *modelDesc, size_t index)**
Python函数	**size = acl.mdl.get_output_size_by_index(model_desc, index)**
函数功能	**根据**aclmdlDesc类型的数据，获取指定输出的大小，单位为Byte。同步接口。

输入说明	**model_desc**：int，aclmdlDesc类型数据的指针。需提前调用**acl.mdl.create_desc**接口创建aclmdlDesc类型的数据。 **index**：int，指定获取第几个输出的大小，index值从0开始。
返回值说明	**size**：int，返回指定输出的大小，单位是Byte。 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。
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create_data_buffer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**》。 – 在应用程序中调用**10.2.5 函数：get_recent_err_msg**接口获取报错。
注意事项	无

10.6.4.8 函数：get_input_dims

C函数原型	**aclError aclmdlGetInputDims(const aclmdlDesc *modelDesc, size_t index, aclmdlIODims *dims)**
Python函数	**dims, ret = acl.mdl.get_input_dims(model_desc, index)**

函数功能	根据模型描述获取模型的输入tensor的维度信息。同步接口。 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acl.mdl.get_input_dims接口或acl.mdl.get.input_dims_v2接口查询维度信息，两者的区别在于： ● 通过acl.mdl.get_input_dims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10-3**所示。 ● 通过acl.mdl.get_input_dims_v2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acl.mdl.get_input_dims_v2接口处的**表10-4**所示，且各维度值相乘的结果值与通过acl.mdl.get_input_size_by_index接口获取的值保持一致。
输入说明	**model_desc**：int，aclmdlDesc类型数据的指针。 **index**：int，指定获取第几个输入的dims，index值从0开始。

返回值说明	**dims**：dict，针对多batch或多种宽高的场景，按照该字典的数组返回其支持的最大档的组合
	● dims = {
	"name": xxx, #tensor name
	"dimCount":xxx，#shape中的维度个数
	"dims": [xx, xx, xx] # 维度信息
	}
	针对动态Batch、动态分辨率（宽高）的场景，输入tensor的dims中Batch数或宽高为-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，pyACL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**acl.mdl.get_tensor_real_name**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。
	ret：int，错误码。
	● 返回0表示成功。
	● 返回**其它值**表示失败。

约束说明	无
注意事项	无

表 10-3 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

10.6.4.9 函数：get_input_dims_v2

C函数原型	**aclError aclmdlGetInputDimsV2(const aclmdlDesc *modelDesc, size_t index, aclmdlIODims *dims)**
Python函数	**dims, ret = acl.mdl.get_input_dims_v2(model_desc, index)**
函数功能	根据模型描述获取模型的输入tensor的维度信息。同步接口。 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acl.mdl.get_input_dims接口或acl.mdl.get.input_dims_v2接口查询维度信息，两者的区别在于： ● 通过acl.mdl.get_input_dims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10-3**所示。 ● 通过acl.mdl.get_input_dims_v2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acl.mdl.get_input_dims_v2接口处的**表10-4**所示，且各维度值相乘的结果值与通过**acl.mdl.get_input_size_by_index**接口获取的值保持一致。
输入说明	**model_desc**：int，aclmdlDesc类型数据的指针。 **index**：int，指定获取第几个输入的大小，index值从0开始。

返回值说明	**dims**：dict，针对多batch或多种宽高的场景，按照该字典的数组返回其支持的最大档的组合。
	● dims = {
	"name": xxx, #tensor name
	"dimCount":xxx，#shape中的维度个数
	"dims": [xx, xx, xx] # 维度信息
	}
	针对动态Batch、动态分辨率（宽高）的场景，输入tensor的dims中Batch数或宽高为-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dims.name时，pyACL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**acl.mdl.get_tensor_real_name**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10-4**所示。
	ret：int，错误码。
	● 返回0表示成功。
	● 返回**其它值**表示失败。

约束说明	无
注意事项	无

表 10-4 动态 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"

图像格式	Format参考格式	维度定义规则
AYUV444_U8	NHWC	"n,h,w,4"

10.6.4.10 函数：get_output_dims

C函数原型	**aclError aclmdlGetOutputDims(const aclmdlDesc *modelDesc, size_t index, aclmdlIODims *dims)**
Python函数	**dims, ret = acl.mdl.get_output_dims(model_desc, index)**
函数功能	根据模型描述获取指定的模型输出tensor的维度信息。同步接口。 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。
输入说明	**model_desc**：int，aclmdlDesc类型数据的指针。 **index**：int，指定获取第几个输入的大小，index值从0开始。

返回值说明	**dims**：dict，针对多batch或多种宽高的场景，按照该字典的数组返回其支持的最大档的组合。
	● dims = {
	"name": xxx, #tensor name
	"dimCount":xxx，#shape中的维度个数
	"dims": [xx, xx, xx] # 维度信息
	}
	若tensor的name长度大于127，则在输出dims.name时，pyACL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**acl.mdl.get_tensor_real_name**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。
	ret：int，错误码。
	● 返回0表示成功。
	● 返回**其它值**表示失败。

约束说明	无
注意事项	无

10.6.4.11 函数：get_input_name_by_index

C函数原型	**const char *aclmdlGetInputNameByIndex(const aclmdlDesc *modelDesc, size_t index)**
Python函数	**input_name = acl.mdl.get_input_name_by_index(model_desc, index)**
函数功能	获取模型中指定输入的输入名称。同步接口。
输入说明	**model_desc**：int，aclmdlDesc类型数据的指针。 **index**：int，指定获取第几个输入的大小，index值从0开始。
返回值说明	**input_name**：str，返回指定输入的输入名称。
约束说明	无
注意事项	无

10.6.4.12 函数：get_output_name_by_index

C函数原型	**const char *aclmdlGetOutputNameByIndex(const aclmdlDesc *modelDesc, size_t index)**
Python函数	**output_name = acl.mdl.get_output_name_by_index(model_desc, index)**
函数功能	获取模型中指定输出的输出名称。同步接口。
输入说明	**model_desc**：int，aclmdlDesc类型数据的指针。 **index**：int，指定获取第几个输入的大小，index值从0开始。

返回值说明	**output_name**：str，返回指定输入的输入名称。 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*
约束说明	无
注意事项	无

10.6.4.13 函数：get_input_format

C函数原型	**aclFormat aclmdlGetInputFormat(const aclmdlDesc *modelDesc, size_t index)**
Python函数	**format_info = acl.mdl.get_input_format(model_desc, index)**
函数功能	获取模型中指定输入的Format。同步接口。
输入说明	**model_desc**：int，aclmdlDesc类型数据的指针。 **index**：int，指定获取第几个输入的大小，index值从0开始。

返回值说明	**format_info**：int，返回指定输入的Format。 ● -1：ACL_FORMAT_UNDEFINED，未知格式，默认值。 ● 0：ACL_FORMAT_NCHW，NCHW格式。 ● 1：ACL_FORMAT_NHWC，NHWC格式。 ● 2：ACL_FORMAT_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，其他需要慎用。 ● 3：ACL_FORMAT_NC1HWC0，华为自研的5维数据格式。 ● 4：ACL_FORMAT_FRACTAL_Z，卷积的权重格式。
约束说明	无
注意事项	无

10.6.4.14 函数：get_output_format

C函数原型	**aclFormat aclmdlGetOutputFormat(const aclmdlDesc *modelDesc, size_t index)**
Python函数	**format_info = acl.mdl.get_output_format(model_desc, index)**
函数功能	获取模型中指定输出的Format。同步接口。
输入说明	**model_desc**：int，aclmdlDesc类型数据的指针。 **index**：int，指定获取第几个输入的大小，index值从0开始。
返回值说明	**format_info**：int，返回指定输入的Format。 ● -1：ACL_FORMAT_UNDEFINED，未知格式，默认值。 ● 0：ACL_FORMAT_NCHW，NCHW格式。 ● 1：ACL_FORMAT_NHWC，NHWC格式。 ● 2：ACL_FORMAT_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他需要慎用。 ● 3：ACL_FORMAT_NC1HWC0，华为自研的5维数据格式。 ● 4：ACL_FORMAT_FRACTAL_Z，卷积的权重格式。
约束说明	无
注意事项	无

10.6.4.15 函数：get_input_data_type

C函数原型	**aclDataType aclmdlGetInputDataType(const aclmdlDesc *modelDesc, size_t index)**
Python函数	**data_type = acl.mdl.get_input_data_type(model_desc, index)**

函数功能	获取模型中指定输入的数据类型。同步接口。
输入说明	**model_desc**：int，aclmdlDesc类型数据的指针。 需提前调用**acl.mdl.create_desc**接口创建aclmdlDesc类型的数据。 **index**：int，指定获取第几个输入的大小，index值从0开始。
返回值说明	**data_type**：int，返回指定输入的数据类型。 ● -1：ACL_DT_UNDEFINED，未知数据类型，默认值。 ● 0：ACL_FLOAT ● 1：ACL_FLOAT16 ● 2：ACL_INT8 ● 3：ACL_INT32 ● 4：ACL_UINT8 ● 6：ACL_INT16 ● 7：ACL_UINT16 ● 8：ACL_UINT32 ● 9：ACL_INT64 ● 10：ACL_UINT64 ● 11：ACL_DOUBLE ● 12：ACL_BOOL
约束说明	无
注意事项	无

10.6.4.16 函数：get_output_data_type

C函数原型	**aclDataType aclmdlGetOutputDataType(const aclmdlDesc *modelDesc, size_t index)**
Python函数	**data_type = acl.mdl.get_output_data_type(model_desc, index)**
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。同步接口。
输入说明	**model_desc**：int，aclmdlDesc类型数据的指针。 **index**：int，指定获取第几个输入的大小，index值从0开始。

返回值说明	ata_type：int，返回指定输入的数据类型。 ● -1：ACL_DT_UNDEFINED，未知数据类型，默认值。 ● 0：ACL_FLOAT ● 1：ACL_FLOAT16 ● 2：ACL_INT8 ● 3：ACL_INT32 ● 4：ACL_UINT8 ● 6：ACL_INT16 ● 7：ACL_UINT16 ● 8：ACL_UINT32 ● 9：ACL_INT64 ● 10：ACL_UINT64 ● 11：ACL_DOUBLE ● 12：ACL_BOOL
约束说明	无
注意事项	无

10.6.4.17 函数：get_input_index_by_name

C函数原型	**aclError aclmdlGetInputIndexByName(const aclmdlDesc *modelDesc, const char *name, size_t *index)**
Python函数	**index, ret = acl.mdl.get_input_index_by_name(model_desc, name)**
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。同步接口。
输入说明	**model_desc**：int，aclmdlDesc类型数据的指针。 **name**：str，指定输入名称。
返回值说明	**index**：int，获取对应输入的索引编号。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	动态Batch和动态分辨率输入的名称固定为"ascend_mbatch_shape_data"，动态AIPP输入的名称固定为"ascend_dynamic_aipp_data"。

10.6.4.18 函数：get_output_index_by_name

C函数原型	**aclError aclmdlGetOutputIndexByName(const aclmdlDesc *modelDesc, const char *name, size_t *index)**
Python函数	**index, ret = acl.mdl.get_output_index_by_name(model_desc, name)**
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。同步接口。
输入说明	**model_desc**：int，aclmdlDesc类型数据的指针。 **name**：str，指定输入名称。
返回值说明	**index**：int，获取对应输入的索引编号。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.6.4.19 函数：get_dynamic_batch

C函数原型	**aclError aclmdlGetDynamicBatch(const aclmdlDesc *modelDesc, aclmdlBatch *batch)**
Python函数	**batch, ret = acl.mdl.get_dynamic_batch(model_desc)**
函数功能	获取模型支持的动态Batch信息。同步接口。
输入说明	**model_desc**：int，aclmdlDesc类型数据的指针。

返回值说明	**batch**：dict，获取的模型支持的动态Batch信息
	● batch = {
	"batchCount": int，#模型中支持的batch分档数
	"batch":[xx, xx,] # 模型中支持的具体分档
	}
	模型中支持的最大batch数为128
	● 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。
	ret：int，错误码。
	● 返回0表示成功。
	● 返回其它值表示失败。

约束说明	无
注意事项	无

10.6.4.20 函数：get_dynamic_hw

C函数原型	**aclError aclmdlGetDynamicHW(const aclmdlDesc *modelDesc, size_t index, aclmdlHW *hw)**
Python函数	**hw_info, ret = acl.mdl.get_dynamic_hw(model_desc, index)**
函数功能	获取模型支持的动态宽高信息。同步接口。
输入说明	**model_desc**：int，aclmdlDesc类型数据的指针。 **index**：预留参数，当前未使用，固定设置为-1。

返回值说明	**hw_info**：dict，获取的模型支持的宽高信息
	● hw_info= {
	"hwCount": int，#模型中支持的宽高分档数
	"hw":[xx, xx,] # 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
	}
	最大的HW数量为128
	● 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。
	ret：int，错误码。
	● 返回0表示成功。
	● 返回其它值表示失败。

约束说明	无
注意事项	无

10.6.4.21 函数：get_cur_output_dims

C函数原型	**aclError aclmdlGetCurOutputDims(const aclmdlDesc *modelDesc, size_t index, aclIODims *dims)**
Python函数	**dims, ret = acl.mdl.get_cur_output_dims(model_desc, index)**
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。同步接口。
输入说明	**model_desc**：int，aclmdlDesc类型数据的指针。 需提前调用**acl.mdl.create_desc**接口创建aclmdlDesc类型的数据。 **index**：int，指定获取第几个输出的Dims，index值从0开始。

返回值说明	**dims**：dict，输出实际维度信息。 若tensor的name长度大于127，则在输出dims.name时，pyACL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**acl.mdl.get_tensor_real_name**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。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	动态Shape（动态Batch或动态分辨率或动态维度（ND格式））场景下，如果用户已调用**acl.mdl.set_dynamic_batch_size**设置Batch、或调用**acl.mdl.set_dynamic_hw_size**接口设置输入图片的宽高、或调用**acl.mdl.set_input_dynamic_dims**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**acl.mdl.set_dynamic_batch_size**接口、或**acl.mdl.set_dynamic_hw_size**接口、或**acl.mdl.set_input_dynamic_dims**接口，则通过该接口可获取最大档的维度信息。 非动态Shape场景下，通过该接口获取指定的模型输出tensor的维度信息。
注意事项	无

10.6.4.22 函数：get_first_aipp_info

C函数原型	**aclError aclmdlGetFirstAippInfo(uint32_t modelId, size_t index, aclAippInfo *aippInfo)**
Python函数	**aipp_info, ret = acl.mdl.get_first_aipp_info(model_id, index)**
函数功能	获取模型静态AIPP的配置信息。同步接口。 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。
输入说明	**model_id**：int，模型ID。 调用acl.mdl.load_from_file接口/acl.mdl.load_from_mem接口/acl.mdl.load_from_file_with_mem接口/acl.mdl.load_from_mem_with_mem接口加载模型成功后，会返回模型ID。 **index**：int，指定模型的第几个输入。

返回值说明	**aipp_info**：dict，获取指定输入上静态AIPP的配置信息。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
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。 aclAippInfo结构体以字典返回，字典中的key对应结构体的成员名。
注意事项	无

10.6.4.23 函数：get_input_dynamic_gear_count

C函数原型	**aclError aclmdlGetInputDynamicGearCount(const aclmdlDesc *modelDesc, size_t index, size_t *gearCount)**
Python函数	**gear_count, ret = acl.mdl.get_input_dynamic_gear_count(model_desc, index)**
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。同步接口。
输入说明	**model_desc**：int，aclmdlDesc类型数据的指针。 **index**：int，预留参数，当前未使用，固定设置为-1。
返回值说明	**gear_count**：int， 动态维度档位数。 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。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
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。
注意事项	无

10.6.4.24 函数：get_input_dynamic_dims

C函数原型	**aclError aclmdlGetInputDynamicDims(const aclmdlDesc *modelDesc, size_t index, aclmdlIODims *dims, size_t gearCount)**

Python函数	**dims_out, ret = acl.mdl.get_input_dynamic_dims(model_desc, index, gear_count)**
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。同步接口。
输入说明	**model_desc**：int，aclmdlDesc类型数据的指针。 **index**：int，预留参数，当前未使用，固定设置为-1。 **gear_count**：int，模型支持的动态维度档位数，需要先通过**acl.mdl.get_input_dynamic_gear_count**接口获取。
返回值说明	**dims_out**：list，获取输入的动态维度信息。 dims_out参数是一个列表，模型有几个输入，则该列表就有几个元素。列表中的每个元素都是一个**10.14.17 aclmdlIODims**的字典，aclmdlIODims字典中的dims是一个列表，该列表中的每个元素对应每一档中的具体值。 例如： dims_out, ret = acl.mdl.get_input_dynamic_dims(model_desc, -1, gear_count) **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	只有在模型转换时通过--dynamic_dims参数设置了分档信息后，才可以调用该接口获取动态维度信息。 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)
注意事项	无

10.6.4.25 函数：get_op_attr

C函数原型	**const char *aclmdlGetOpAttr(aclmdlDesc *modelDesc, const char *opName, const char *attr)**
Python函数	**attrValue = acl.mdl.get_op_attr(modelDesc, opName, attr)**
函数功能	获取整网中某个模型中某个算子的属性的值。同步接口。
输入说明	**modelDesc**：aclmdlDesc类型的指针。 需提前调用 **acl.mdl.create_desc**接口创建aclmdlDesc类型的数据，再调用 **acl.mdl.get_desc**接口根据模型ID获取到对应的aclmdlDesc类型的数据。 **opName**：算子名称。 **attr**：算子属性。 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	**attrValue**：返回属性值的字符串，若opName或者attr属性不存在、或者attr属性值为空，均返回空字符串。 若调用该接口失败，则返回None。
约束说明	无
注意事项	无

10.7 媒体数据处理 V1

10.7.1 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版本的接口功能范围相同，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。
  - 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。

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10-5所示。

表 10-5 功能说明

功能	说明
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10.7.6.1 功能说明。
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10.7.7.1 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10.7.8.1 功能及约束说明。
VDEC（Video Decoder）	负责视频解码，详细描述请参见10.7.9.1 功能及约束说明。
VENC（Video Encoder）	负责视频编码，详细描述请参见10.7.10.1 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10.7.11.1 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10-6所示。

表 10-6 功能支持度说明

昇腾AI处理器	VPC	JPEGD	JPEGE	PNGD	VDEC	VENC
Atlas 200/500 A2 推理产品	√	√	√	√	√	√

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**acl.rt.synchronize_stream**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**acl.rt.synchronize_stream**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，**acl.rt.synchronize_stream**）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**acl.media.dvpp_malloc**申请内存、调用**acl.media.dvpp_free**接口释放内存。
  b. 调用**1**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**acl.rt.malloc**接口、或**acl.rt.malloc_host**接口、或**acl.rt.malloc_cached**接口申请内存。
- 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**10.7.4 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 200/500 A2推理产品上，各功能的通道数有一定的限制，VPC的通道数最多128，JPEGD的通道数最多128，VDEC的通道数最多128，JPEGE的通道数最多128，VENC的通道数最多128，PNGD的通道数最多128。创建通道的接口请参见**10.7.4 通道创建与释放**。

10.7.2 基本概念

表 10-7 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**10.7.7.1 功能及约束说明**。 ● JPEGE功能，请参见**10.7.8.1 功能及约束说明**。 ● PNGD功能，请参见**10.7.11.1 功能及约束说明**。 ● VDEC功能，请参见**10.7.9.1 功能及约束说明**。 ● VENC功能，请参见**10.7.10.1 功能及约束说明**。

概念	描述
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**10.7.7.1 功能及约束说明**。 ● JPEGE功能，请参见**10.7.8.1 功能及约束说明**。 ● PNGD功能，请参见**10.7.11.1 功能及约束说明**。 ● VDEC功能，请参见**10.7.9.1 功能及约束说明**。 ● VENC功能，请参见**10.7.10.1 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10-1**。
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：系数为1。 ● YUV422packed：系数为2。 ● YUV444packed、RGB888：系数为3。 ● ARGB8888：系数为4。
8K	8K表示宽或高在4096~8192（不包括4096）范围内的输入图片。

10.7.3 内存申请与释放

10.7.3.1 函数：dvpp_malloc

C函数原型	**aclError acldvppMalloc(void **devPtr, size_t size)**
Python函数	**dev_ptr, ret = acl.media.dvpp_malloc(size)**

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对齐），同步接口。
输入说明	**size**：int，申请内存的大小，单位Byte。
返回值说明	**dev_ptr**：int，“Device上已分配内存的指针”的指针。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
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**acl.media.dvpp_free**接口释放内存。 ● 频繁调用acl.media.dvpp_malloc接口申请内存、调用**acl.media.dvpp_free**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acl.media.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。 ● 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。
注意事项	无
参考资源	● 接口调用流程，参见**5.3 媒体数据处理V1**。 ● 接口调用示例，参见**5.3 媒体数据处理V1**。

10.7.3.2 函数：dvpp_free

C函数原型	**aclError acldvppFree(void *devPtr)**
Python函数	**ret = acl.media.dvpp_free(dev_ptr)**
函数功能	释放通过**acl.media.dvpp_malloc**接口申请的内存，同步接口。
输入说明	**dev_ptr**：int，待释放内存的指针。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。

约束说明	无
注意事项	无

10.7.4 通道创建与释放

10.7.4.1 VPC/JPEGD/JPEGE/PNGD 图片处理通道

10.7.4.1.1 函数：dvpp_create_channel

C函数原型	**aclError acldvppCreateChannel(acldvppChannelDesc *channelDesc)**
Python函数	**ret = acl.media.dvpp_create_channel(dvpp_channel_desc)**
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。
输入说明	**dvpp_channel_desc**：int，指定通道描述信息。调用acl.media.dvpp_create_channel_desc接口创建该输入数据。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
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。 ● acl.media.dvpp_create_channel接口内部调用1次acl.rt.create_stream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，acl.rt.create_stream接口的约束需要遵循，例如Stream的数量等。
注意事项	无

10.7.4.1.2 函数：dvpp_destroy_channel

C函数原型	**aclError acldvppDestroyChannel(acldvppChannelDesc *channelDesc)**
Python函数	**ret = acl.media.dvpp_destroy_channel(dvpp_channel_desc)**
函数功能	销毁图片数据处理的通道，只能销毁通过acl.media.dvpp_create_channel接口创建的通道，同步接口。

输入说明	**dvpp_channel_desc**：int，指定通道描述信息。与调用**acl.media.dvpp_create_channel**接口创建通道时指定的dvpp_channel_desc保持一致。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	如果在调用**acl.media.dvpp_destroy_channel**接口销毁通道前，已调用**acl.media.dvpp_destroy_channel_desc**接口销毁了通道描述信息，那么在调用**acl.media.dvpp_destroy_channel**接口销毁通道时会报错。
注意事项	无

10.7.4.2 VDEC 视频解码通道

10.7.4.2.1 函数：vdec_create_channel

C函数原型	**aclError aclvdecCreateChannel(aclvdecChannelDesc *channelDesc)**
Python函数	**ret = acl.media.vdec_create_channel(vdec_channel_desc)**
函数功能	创建视频数据处理的通道，同一个通道可以重复使用，销毁后不再可用，同步接口。
输入说明	**vdec_channel_desc**：int，vdec通道描述地址对象, 指定通道描述信息。 调用**acl.media.vdec_create_channel_desc**接口创建vdec_channel_desc类型的数据，调用acl.media.vdec_set_channel_desc系列接口设置通道描述信息的属性。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。

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。 ● **acl.media.vdec_create_channel**接口内部封装了**acl.rt.create_stream**接口和**acl.rt.subscribe_report**： – 调用2次**acl.rt.create_stream**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**acl.media.vdec_destroy_channel**接口销毁通道后才能自动释放。 – 调用**acl.rt.subscribe_report**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acl.media.vdec_set_channel_desc系列接口时指定的。 用户在实现VDEC功能时，无需再单独调用**acl.rt.create_stream**接口、**acl.rt.subscribe_report**接口，但**acl.rt.create_stream**接口、**acl.rt.subscribe_report**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。
注意事项	无
参考资源	● 接口调用流程，参见**5.3 媒体数据处理V1**。 ● 接口调用示例，参见**5.3 媒体数据处理V1**。

10.7.4.2.2 函数：vdec_destroy_channel

C函数原型	**aclError aclvdecDestroyChannel(aclvdecChannelDesc *channelDesc)**
Python函数	**ret = acl.media.vdec_destroy_channel(vdec_channel_desc)**
函数功能	销毁视频数据处理的通道，只能销毁通过**acl.media.vdec_create_channel**接口创建的通道，同步接口。
输入说明	**vdec_channel_desc**：int，vdec通道描述地址对象，指定通道描述信息。与调用**acl.media.vdec_create_channel**接口创建通道时指定的vdec_channel_desc保持一致。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。

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。 • **acl.media.vdec_destroy_channel**接口内部封装了**acl.rt.unsubscribe_report**接口取消线程注册（Stream上的回调函数不再由指定线程处理）、**acl.rt.destroy_stream**接口销毁Stream。用户在实现VDEC功能时，无需再单独调用**acl.rt.unsubscribe_report**接口、**acl.rt.destroy_stream**接口。 • 如果在调用**acl.media.vdec_destroy_channel**接口销毁通道前，已调用**acl.media.vdec_destroy_channel_desc**接口销毁了通道描述信息，那么在调用**acl.media.vdec_destroy_channel**接口销毁通道时会报错。
注意事项	无

10.7.4.3 VENC 视频编码通道

10.7.4.3.1 函数：venc_create_channel

C函数原型	**aclError aclvencCreateChannel(aclvencChannelDesc *channelDesc)**
Python函数	**ret = acl.media.venc_create_channel(venc_channel_desc)**
函数功能	创建视频数据编码处理的通道，同一个通道可以重复使用，销毁后不再可用，同步接口。
输入说明	**venc_channel_desc**：int，指定通道描述信息。 调用**acl.media.venc_create_channel_desc**接口创建aclvencChannelDesc类型的数据，调用acl.media.venc_set_channel_desc系列接口设置通道描述信息的属性。
返回值说明	**ret**：int，错误码。 • 返回0表示成功。 • 返回**其它值**表示失败。
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。 • acl.media.venc_create_channel接口内部调用2次**acl.rt.create_stream**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**acl.media.venc_destroy_channel**接口销毁通道后才能自动释放。 用户在实现VENC功能时，**acl.rt.create_stream**接口的约束需要遵循，例如Stream的数量等。
注意事项	无

参考资源	● 接口调用流程，参见**5.3 媒体数据处理V1**。 ● 接口调用示例，参见**5.3 媒体数据处理V1**。

10.7.4.3.2 函数：venc_destroy_channel

C函数原型	**aclError aclvencDestroyChannel(aclvencChannelDesc *channelDesc)**
Python函数	**ret = acl.media.venc_destroy_channel(venc_channel_desc)**
函数功能	销毁视频数据编码处理的通道，只能销毁通过**acl.media.venc_create_channel**接口创建的通道，同步接口。
输入说明	**venc_channel_desc**：int，指定通道描述信息，与调用**acl.media.venc_create_channel**接口创建通道时指定的channelDesc保持一致。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
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**acl.media.venc_destroy_channel**接口销毁通道前，已调用**acl.media.venc_destroy_channel_desc**接口销毁了通道描述信息，那么在调用**acl.media.venc_destroy_channel**接口销毁通道时会报错。
注意事项	无

10.7.5 common

10.7.5.1 函数：dvpp_create_channel_desc

C函数原型	**acldvppChannelDesc *acldvppCreateChannelDesc()**
Python函数	**dvpp_channel_desc = acl.media.dvpp_create_channel_desc()**
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。该接口为同步接口。
输入说明	无
返回值说明	**dvpp_channel_desc**：int ● 若是非0值，表示成功。 ● 若是0，表示失败。

约束说明	无
注意事项	无

10.7.5.2 函数：dvpp_destroy_channel_desc

C函数原型	**aclError acldvppDestroyChannelDesc(acldvppChannelDesc *channelDesc)**
Python函数	**ret = acl.media.dvpp_destroy_channel_desc(dvpp_channel_desc)**
函数功能	销毁通过**acl.media.dvpp_create_channel_desc**接口创建的acldvppChannelDesc类型的数据。同步接口。
输入说明	**dvpp_channel_desc**：int，指定需要销毁的通道描述信息。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
注意事项	无

10.7.5.3 函数：dvpp_get_channel_desc_channel_id

C函数原型	**uint64_t acldvppGetChannelDescChannelId(const acldvppChannelDesc *channelDesc)**
Python函数	**channel_id = acl.media. dvpp_get_channel_desc_channel_id (dvpp_channel_desc)**
函数功能	根据通道描述信息获取通道ID。同步接口。
输入说明	**dvpp_channel_desc**：int，指定通道描述信息。 调用**acl.media.dvpp_create_channel_desc**接口创建的dvpp_channel_desc数据。
返回值说明	**channel_id**：int，返回通道ID。
约束说明	**接口调用顺序：acl.media.dvpp_create_channel_desc** --> **acl.media.dvpp_create_channel**--> **acl.media.dvpp_get_channel_desc_channel_id**。
注意事项	无

10.7.5.4 函数：dvpp_set_channel_desc_mode

C函数原型	**aclError acldvppSetChannelDescMode(acldvppChannelDesc *channelDesc, uint32_t mode)**
Python函数	**ret = acl.media.dvpp_set_channel_desc_mode(dvpp_channel_desc, mode)**
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。同步接口。
输入说明	**dvpp_channel_desc**：int，指定通道描述信息。 需提前调用**acl.media.dvpp_create_channel_desc**接口创建的dvpp_channel_desc数据。 **mode**：int，指定通道模式。 ● 1：DVPP_CHNMODE_VPC，指定图片数据处理通道用于实现VPC功能。 ● 2：DVPP_CHNMODE_JPEGD，指定图片数据处理通道用于实现JPEGD功能。 ● 4：DVPP_CHNMODE_JPEGE，指定图片数据处理通道用于实现JPEGE功能。 ● 8：DVPP_CHNMODE_PNGD，指定图片数据处理通道用于实现PNGD功能。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	**接口调用顺序：acl.media.dvpp_create_channel_desc**--> **acl.media.dvpp_set_channel_desc_mode** --> **acl.media.dvpp_create_channel**。
注意事项	无

10.7.5.5 函数：dvpp_set_channel_desc_param

C函数原型	**aclError acldvppSetChannelDescParam(acldvppChannelDesc *channelDesc, acldvppChannelDescParamType paramType, size_t length, const void *param)**
Python函数	**channelDesc, ret=acl.media.dvpp_set_channel_desc_param(channelDesc, paramType, param)**
函数功能	设置图片数据处理通道的属性参数接口，同步接口。

输入说明	**channelDesc**：指定通道描述信息。 需提前调用**acl.media.dvpp_create_channel_desc**接口创建acldvppChannelDesc类型的数据。 **paramType**：属性参数的类型。 **param**：指向属性参数值。
返回值说明	**channelDesc**：指定通道描述信息。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无

10.7.5.6 函数：dvpp_get_channel_desc_param

C函数原型	**aclError acldvppGetChannelDescParam(const acldvppChannelDesc *channelDesc, acldvppChannelDescParamType paramType, size_t length, size_t *paramRetSize, void *param)**
Python函数	**paramRetSize, param, ret=acl.media.dvpp_get_channel_desc_param(channelDesc, paramType)**
函数功能	获取图片数据处理通道的属性参数接口，同步接口。
输入说明	**channelDesc**：指定通道描述信息。 需提前调用**acl.media.dvpp_create_channel_desc**接口创建acldvppChannelDesc类型的数据，调用**acl.media.dvpp_set_channel_desc_param**接口设置通道属性值。 **paramType**：属性参数的类型。
返回值说明	**paramRetSize**：实际返回的属性参数值的字节数。 **param**：指向属性参数值的指针。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无

10.7.5.7 函数：vdec_set_channel_desc_param

C函数原型	**aclError aclvdecSetChannelDescParam(aclvdecChannelDesc *channelDesc, aclvdecChannelDescParamType paramType, size_t length, const void *param)**
Python函数	**channelDesc, ret = acl.media.vdec_set_channel_desc_param(channelDesc, paramType, param)**
函数功能	设置视频编码处理通道描述信息的属性。同步接口。
输入说明	**channelDesc**：表示视频解码处理通道的描述信息。 需提前调用acl.media.dvpp_create_channel_desc接口创建aclvdecChannelDesc类型的数据。 **paramType**：属性参数的类型。 **param**：指向属性参数值的指针。
返回值说明	**channelDesc**：表示视频解码处理通道的描述信息。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.7.5.8 函数：vdec_get_channel_desc_param

C函数原型	**aclError aclvdecGetChannelDescParam(const aclvdecChannelDesc *channelDesc, aclvdecChannelDescParamType paramType, size_t length, size_t *paramRetSize, void *param)**
Python函数	**paramRetSize, param, ret = acl.media.vdec_get_channel_desc_param(channelDesc, paramType)**
函数功能	获取视频解码处理通道的描述信息，同步接口。
输入说明	**channelDesc**：表示视频解码数据处理通道的描述信息。 需提前调用acl.media.dvpp_create_channel_desc接口创建aclvdecChannelDesc类型的数据，调用acl.media.vdec_set_channel_desc_param接口设置视频编码处理通道描述信息的属性值。 **paramType**：属性参数的类型。

返回值说明	**paramRetSize**：实际返回的属性参数值的字节数。 **param**：指向属性参数值。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.7.5.9 函数：dvpp_create_pic_desc

C函数原型	**acldvppPicDesc *acldvppCreatePicDesc()**
Python函数	**dvpp_pic_desc = acl.media.dvpp_create_pic_desc()**
函数功能	创建acldvppPicDesc类型的数据，表示图片描述信息。同步接口。 如需销毁acldvppPicDesc类型的数据，请参见**acl.media.dvpp_destroy_pic_desc**。
输入说明	无
返回值说明	**dvpp_pic_desc**：int ● 非0，表示成功。 ● 0，表示失败。
约束说明	无
注意事项	无

10.7.5.10 函数：dvpp_destroy_pic_desc

C函数原型	**aclError acldvppDestroyPicDesc(acldvppPicDesc *picDesc)**
Python函数	**ret = acl.media.dvpp_destroy_pic_desc(dvpp_pic_desc)**
函数功能	销毁通过**acl.media.dvpp_create_pic_desc**接口创建的acldvppPicDesc类型的数据。同步接口。
输入说明	**dvpp_pic_desc**：int，指定待销毁的图片描述信息。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无

注意事项	无

10.7.5.11 函数：dvpp_set_pic_desc_data

C函数原型	**aclError acldvppSetPicDescData(acldvppPicDesc *picDesc, void *dataDev)**
Python函数	**ret = acl.media.dvpp_set_pic_desc_data(dvpp_pic_desc, data_dev)**
函数功能	设置图片描述参数：图片数据内存。同步接口。
输入说明	**dvpp_pic_desc**：int，图片描述信息。 **data_dev**：int，Device上图片数据内存，必须是使用**acl.media.dvpp_malloc**接口申请的内存。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无

10.7.5.12 函数：dvpp_set_pic_desc_size

C函数原型	**aclError acldvppSetPicDescSize(acldvppPicDesc *picDesc, uint32_t size)**
Python函数	**ret = acl.media.dvpp_set_pic_desc_size(dvpp_pic_desc, size)**
函数功能	设置图片描述参数：图片数据内存大小。同步接口。
输入说明	**dvpp_pic_desc**：int，图片描述信息。 **size**：int，内存大小，单位Byte。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无

10.7.5.13 函数：dvpp_set_pic_desc_format

C函数原型	**aclError acldvppSetPicDescFormat(acldvppPicDesc *picDesc, acldvppPixelFormat format)**
Python函数	**ret = acl.media.dvpp_set_pic_desc_format(dvpp_pic_desc, format)**
函数功能	设置图片描述参数：图片格式。同步接口。
输入说明	**dvpp_pic_desc**：int，图片描述信息。 **format**：int，图片格式，详细参数请参见10.14.5 acldvppPixelFormat。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.7.5.14 函数：dvpp_set_pic_desc_width

C函数原型	**aclError acldvppSetPicDescWidth(acldvppPicDesc *picDesc, uint32_t width)**
Python函数	**ret = acl.media.dvpp_set_pic_desc_width(dvpp_pic_desc, width)**
函数功能	设置图片描述参数：原始图片宽。同步接口。
输入说明	**dvpp_pic_desc**：int，图片描述信息。 **width**：int，原始图片宽。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.7.5.15 函数：dvpp_set_pic_desc_height

C函数原型	**aclError acldvppSetPicDescHeight(acldvppPicDesc *picDesc, uint32_t height)**
Python函数	**ret = acl.media.dvpp_set_pic_desc_height(dvpp_pic_desc, height)**

函数功能	设置图片描述参数：原始图片高。同步接口。
输入说明	**dvpp_pic_desc**：int，图片描述信息。 **height**：int，原始图片高。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.7.5.16 函数：dvpp_set_pic_desc_width_stride

C函数原型	**aclError acldvppSetPicDescWidthStride(acldvppPicDesc *picDesc, uint32_t widthStride)**
Python函数	**ret = acl.media.dvpp_set_pic_desc_width_stride(dvpp_pic_desc, width_stride)**
函数功能	设置图片描述参数：对齐后的宽。同步接口。
输入说明	**dvpp_pic_desc**：int，图片描述信息。 **width_stride**：int，对齐后的宽。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.7.5.17 函数：dvpp_set_pic_desc_height_stride

C函数原型	**aclError acldvppSetPicDescHeightStride(acldvppPicDesc *picDesc, uint32_t heightStride)**
Python函数	**ret = acl.media.dvpp_set_pic_desc_height_stride(dvpp_pic_desc, height_stride)**
函数功能	设置图片描述参数：对齐后的高。同步接口。
输入说明	**dvpp_pic_desc**：int，图片描述信息。 **height_stride**：int，对齐后的高。

返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.7.5.18 函数：dvpp_set_pic_desc_ret_code

C函数原型	**aclError acldvppSetPicDescRetCode(acldvppPicDesc *picDesc, uint32_t retCode)**
Python函数	**ret = acl.media.dvpp_set_pic_desc_ret_code(dvpp_pic_desc, ret_code)**
函数功能	设置图片描述参数：返回码。同步接口。
输入说明	**dvpp_pic_desc**：int，图片描述信息。 **ret_code**：int 。返回码： ● 0成功。 ● 非0失败。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.7.5.19 函数：dvpp_get_pic_desc_data

C函数原型	**void *acldvppGetPicDescData(const acldvppPicDesc *picDesc)**
Python函数	**data_dev = acl.media.dvpp_get_pic_desc_data(dvpp_pic_desc)**
函数功能	获取图片描述参数：图片数据内存。
输入说明	**dvpp_pic_desc**：int，图片描述信息。
返回值说明	**data_dev**：int，图片数据内存。
约束说明	无
注意事项	无

10.7.5.20 函数：dvpp_get_pic_desc_size

C函数原型	**uint32_t acldvppGetPicDescSize(const acldvppPicDesc *picDesc)**
Python函数	**size = acl.media.dvpp_get_pic_desc_size(dvpp_pic_desc)**
函数功能	获取图片描述参数：图片数据内存大小。
输入说明	**dvpp_pic_desc**：int，图片描述信息。
返回值说明	**size**：int，图片数据内存大小。
约束说明	无
注意事项	无

10.7.5.21 函数：dvpp_get_pic_desc_format

C函数原型	**acldvppPixelFormat acldvppGetPicDescFormat(const acldvppPicDesc *picDesc)**
Python函数	**format = acl.media.dvpp_get_pic_desc_format(dvpp_pic_desc)**
函数功能	获取图片描述参数：图片格式。
输入说明	**dvpp_pic_desc**：int，图片描述信息。
返回值说明	**format**：int，图片格式。具体数值请参考**acl.media.dvpp_set_pic_desc_format**接口。
约束说明	无
注意事项	无

10.7.5.22 函数：dvpp_get_pic_desc_width

C函数原型	**uint32_t acldvppGetPicDescWidth(const acldvppPicDesc *picDesc)**
Python函数	**width = acl.media.dvpp_get_pic_desc_width(dvpp_pic_desc)**
函数功能	获取图片描述参数：原始图片宽。
输入说明	**dvpp_pic_desc**：int，图片描述信息。
返回值说明	**width**：int，原始图片宽。
约束说明	无
注意事项	无

10.7.5.23 函数：dvpp_get_pic_desc_height

C函数原型	**uint32_t acldvppGetPicDescHeight(const acldvppPicDesc *picDesc)**
Python函数	**height = acl.media.dvpp_get_pic_desc_height(dvpp_pic_desc)**
函数功能	获取图片描述参数：原始图片高。
输入说明	**dvpp_pic_desc**：int，图片描述信息。
返回值说明	**height**：int，原始图片高。
约束说明	无
注意事项	无

10.7.5.24 函数：dvpp_get_pic_desc_width_stride

C函数原型	**uint32_t acldvppGetPicDescWidthStride(const acldvppPicDesc *picDesc)**
Python函数	**width_stride = acl.media.dvpp_get_pic_desc_width_stride(dvpp_pic_desc)**
函数功能	获取图片描述参数：对齐后的宽。
输入说明	**dvpp_pic_desc**：int，图片描述信息。
返回值说明	**width_stride**：int，对齐后的宽。
约束说明	无
注意事项	无

10.7.5.25 函数：dvpp_get_pic_desc_height_stride

C函数原型	**uint32_t acldvppGetPicDescHeightStride(const acldvppPicDesc *picDesc)**
Python函数	**height_stride = acl.media.dvpp_get_pic_desc_height_stride(dvpp_pic_desc)**
函数功能	获取图片描述参数：对齐后的高。
输入说明	**dvpp_pic_desc**：int，图片描述信息。
返回值说明	**height_stride**：int，对齐后的高。
约束说明	无
注意事项	无

10.7.5.26 函数：dvpp_get_pic_desc_ret_code

C函数原型	**uint32_t acldvppGetPicDescRetCode(const acldvppPicDesc *picDesc)**
Python函数	**ret_code = acl.media.dvpp_get_pic_desc_ret_code(dvpp_pic_desc)**
函数功能	获取图片描述参数：返回码。
输入说明	**dvpp_pic_desc**：int，图片描述信息。
返回值说明	**ret_code**：int，返回码。 ● 0成功。 ● 非0失败。
约束说明	无
注意事项	无

10.7.5.27 函数：dvpp_create_stream_desc

C函数原型	**acldvppStreamDesc *acldvppCreateStreamDesc()**
Python函数	**dvpp_stream_desc = acl.media.dvpp_create_stream_desc()**
函数功能	创建视频码流描述信息。该接口为同步接口。
输入说明	无
返回值说明	**dvpp_stream_desc**：int ● 非0，表示成功。 ● 0，表示失败。
约束说明	无
注意事项	无

10.7.5.28 函数：dvpp_destroy_stream_desc

C函数原型	**aclError acldvppDestroyStreamDesc(acldvppStreamDesc *streamDesc)**
Python函数	**ret = acl.media.dvpp_destroy_stream_desc(dvpp_stream_desc)**
函数功能	销毁视频码流描述信息，只能销毁通过**acl.media.dvpp_create_stream_desc**接口创建的视频码流描述信息。该接口为同步接口。

输入说明	**dvpp_stream_desc**：int，指定待销毁的视频码流描述信息。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	销毁类接口：调用该接口后，不能继续使用已释放或销毁的资源，建议调用该接口后，将相关资源设置为无效值（例如，置为0或None）
注意事项	无

10.7.5.29 函数：dvpp_set_stream_desc_data

C函数原型	**aclError acldvppSetStreamDescData(acldvppStreamDesc *streamDesc, void *dataDev)**
Python函数	**ret = acl.media.dvpp_set_stream_desc_data(dvpp_stream_desc, data_dev)**
函数功能	设置视频码流描述参数：Device上视频码流数据内存。
输入说明	**dvpp_stream_desc**：int，视频码流描述信息。 **data_dev**：int，Device上视频码流数据内存。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.7.5.30 函数：dvpp_set_stream_desc_size

C函数原型	**aclError acldvppSetStreamDescSize(acldvppStreamDesc *streamDesc, uint32_t size)**
Python函数	**ret = acl.media.dvpp_set_stream_desc_size(dvpp_stream_desc, size)**
函数功能	设置视频码流描述参数：视频码流数据内存大小。
输入说明	**dvpp_stream_desc**：int，视频码流描述信息。 **size**：int，内存大小，单位Byte。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

约束说明	无
注意事项	无

10.7.5.31 函数：dvpp_set_stream_desc_format

C函数原型	**aclError acldvppSetStreamDescFormat(acldvppStreamDesc *streamDesc, acldvppStreamFormat format)**
Python函数	**ret = acl.media.dvpp_set_stream_desc_format(dvpp_stream_desc, format)**
函数功能	设置视频码流描述参数：视频码流格式。
输入说明	**dvpp_stream_desc**：int，视频码流描述信息。 **format**：int，视频码流格式（h264/h265）。 ● 0：H265_MAIN_LEVEL ● 1：H264_BASELINE_LEVEL ● 2：H264_MAIN_LEVEL ● 3：H264_HIGH_LEVEL
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.7.5.32 函数：dvpp_set_stream_desc_timestamp

C函数原型	**aclError acldvppSetStreamDescTimestamp(acldvppStreamDesc *streamDesc, uint64_t timestamp)**
Python函数	**ret = acl.media.dvpp_set_stream_desc_timestamp(dvpp_stream_desc, timestamp)**
函数功能	设置视频码流描述参数：时间戳。
输入说明	**dvpp_stream_desc**：int，视频码流描述信息。 **timestamp**：int，时间戳。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

约束说明	无
注意事项	无

10.7.5.33 函数：dvpp_set_stream_desc_ret_code

C函数原型	**aclError acldvppSetStreamDescRetCode(acldvppStreamDesc *streamDesc, uint32_t retCode)**
Python函数	**ret = acl.media.dvpp_set_stream_desc_ret_code(dvpp_stream_desc, ret_code)**
函数功能	设置视频码流描述参数：返回码。
输入说明	**dvpp_stream_desc**：int，视频码流描述信息。 **ret_code**：int，返回码 ● 返回0成功。 ● 返回非0失败 。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.7.5.34 函数：dvpp_set_stream_desc_eos

C函数原型	**aclError acldvppSetStreamDescEos(acldvppStreamDesc *streamDesc, uint8_t eos)**
Python函数	**ret = acl.media.dvpp_set_stream_desc_eos(dvpp_stream_desc, eos)**
函数功能	设置视频码流描述参数：是否为最后一帧数据。
输入说明	**dvpp_stream_desc**：int，视频码流描述信息。 **eos**：int，是否为最后一帧数据（0否，1是）。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.7.5.35 函数：dvpp_get_stream_desc_data

C函数原型	**void *acldvppGetStreamDescData(const acldvppStreamDesc *streamDesc)**
Python函数	**data_dev = acl.media.dvpp_get_stream_desc_data(dvpp_stream_desc)**
函数功能	获取视频码流描述参数：Device上视频码流数据内存。
输入说明	**dvpp_stream_desc**：int，视频码流描述信息。
返回值说明	**data_dev**：int，Device上视频码流数据内存。
约束说明	无
注意事项	无

10.7.5.36 函数：dvpp_get_stream_desc_size

C函数原型	**uint32_t acldvppGetStreamDescSize(const acldvppStreamDesc *streamDesc)**
Python函数	**size = acl.media.dvpp_get_stream_desc_size(dvpp_stream_desc)**
函数功能	获取视频码流描述参数：视频码流数据内存大小。
输入说明	**dvpp_stream_desc**：int，视频码流描述信息。
返回值说明	**size**：int，内存大小，单位Byte。
约束说明	无
注意事项	无

10.7.5.37 函数：dvpp_get_stream_desc_format

C函数原型	**acldvppStreamFormat acldvppGetStreamDescFormat(const acldvppStreamDesc *streamDesc)**
Python函数	**format = acl.media.dvpp_get_stream_desc_format(dvpp_stream_desc)**
函数功能	获取视频码流描述参数：视频码流格式。
输入说明	**dvpp_stream_desc**：int，视频码流描述信息。

返回值说明	**format**：int，视频码流格式（h264/h265）。 ● 0：H265_MAIN_LEVEL ● 1：H264_BASELINE_LEVEL ● 2：H264_MAIN_LEVEL ● 3：H264_HIGH_LEVEL
约束说明	无
注意事项	无

10.7.5.38 函数：dvpp_get_stream_desc_timestamp

C函数原型	**uint64_t acldvppGetStreamDescTimestamp(const acldvppStreamDesc *streamDesc)**
Python函数	**timestamp = acl.media.dvpp_get_stream_desc_timestamp(dvpp_stream_desc)**
函数功能	获取视频码流描述参数：时间戳。
输入说明	**dvpp_stream_desc**：int，视频码流描述信息。
返回值说明	**timestamp**：int，时间戳。
约束说明	无
注意事项	无

10.7.5.39 函数：dvpp_get_stream_desc_ret_code

C函数原型	**uint32_t acldvppGetStreamDescRetCode(const acldvppStreamDesc *streamDesc)**
Python函数	**ret_code = acl.media.dvpp_get_stream_desc_ret_code(dvpp_stream_desc)**
函数功能	获取视频码流描述参数：返回码。
输入说明	**dvpp_stream_desc**：int，视频码流描述信息。
返回值说明	**ret_code**：int，返回码。 ● 返回0成功。 ● 返回非0失败。
约束说明	无
注意事项	无

10.7.5.40 函数：dvpp_get_stream_desc_eos

C函数原型	**uint8_t acldvppGetStreamDescEos(const acldvppStreamDesc *streamDesc)**
Python函数	**eos = acl.media.dvpp_get_stream_desc_eos(dvpp_stream_desc)**
函数功能	获取视频码流描述参数：是否为最后一帧数据。
输入说明	**dvpp_stream_desc**：int，视频码流描述信息。
返回值说明	**eos**：int，是否为最后一帧数据（0否，1是）。
约束说明	无
注意事项	无

10.7.5.41 函数：dvpp_create_batch_pic_desc

C函数原型	**acldvppBatchPicDesc *acldvppCreateBatchPicDesc(uint32_t batchSize)**
Python函数	**batch_pic_desc = acl.media.dvpp_create_batch_pic_desc(batch_size)**
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**acl.media.dvpp_destroy_batch_pic_desc**。
输入说明	**batch_size**：int，图片数量。
返回值说明	**batch_pic_desc**：int，返回创建描述信息的地址。 ● 若是非0值，表示成功。 ● 若是0，表示失败。
约束说明	无
注意事项	无

10.7.5.42 函数：dvpp_destroy_batch_pic_desc

C函数原型	**aclError acldvppDestroyBatchPicDesc (acldvppBatchPicDesc *batchPicDesc)**
Python函数	**ret = acl.media.dvpp_destroy_batch_pic_desc(batch_pic_desc)**
函数功能	销毁通过**acl.media.dvpp_create_batch_pic_desc**接口创建的acldvppBatchPicDesc类型的数据。同步接口。
输入说明	**batch_pic_desc**：int，指定需要销毁的acldvppBatchPicDesc。

返回值说明	**ret**：int，返回码。 ● 返回0成功。 ● 返回非0失败。
约束说明	销毁类接口：调用该接口后，不能继续使用已释放或销毁的资源，建议调用该接口后，将相关资源设置为无效值（例如，置为0或None）。
注意事项	无

10.7.5.43 函数：dvpp_get_pic_desc

C函数原型	**acldvppPicDesc *acldvppGetPicDesc(acldvppBatchPicDesc *batchPicDesc, uint32_t index)**
Python函数	**pic_desc = acl.media.dvpp_get_pic_desc(batch_pic_desc, index)**
函数功能	获取指定图片的图片描述信息。同步接口。
输入说明	**batch_pic_desc**：int，指定批量图片描述信息。 **index**：int，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	**pic_desc**：返回指定图片的图片描述信息。
约束说明	无
注意事项	无

10.7.5.44 函数：dvpp_create_border_config

C函数原型	**acldvppBorderConfig *acldvppCreateBorderConfig()**
Python函数	**pic_desc, ret = acl.media.dvpp_create_border_config()**
函数功能	创建acldvppBorderConfig类型的数据，表示边界填充配置。同步接口。 如需销毁acldvppBorderConfig类型的数据，请参见**acl.media.dvpp_destroy_border_config**。
输入说明	无
返回值说明	**pic_desc**：返回指定图片的图片描述信息。 **ret**：int，返回码。 ● 返回0成功。 ● 返回非0失败。
约束说明	无

注意事项	无

10.7.5.45 函数：dvpp_destroy_border_config

C函数原型	**aclError acldvppDestroyBorderConfig(acldvppBorderConfig *borderConfig)**
Python函数	**ret = acl.media.dvpp_destroy_border_config(border_config)**
函数功能	销毁通过**dvpp_create_border_config**接口创建的acldvppBorderConfig类型的数据。同步接口。
输入说明	**border_config**：待销毁的acldvppBorderConfig类型的指针。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无

10.7.5.46 函数：dvpp_set_border_config_value

C函数原型	**aclError acldvppSetBorderConfigValue(acldvppBorderConfig *borderConfig, uint32_t index, double value);**
Python函数	**ret = acl.media.dvpp_set_border_config_value(border_config, index, value)**
函数功能	设置边界填充配置参数。同步接口。
输入说明	**border_config**：表示边界填充配置数据。需提前调用**acl.media.dvpp_create_border_config**接口创建acldvppBorderConfig类型的数据. **index**：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**：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。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无

注意事项	无

10.7.5.47 函数：dvpp_set_border_config_border_type

C函数原型	**aclError acldvppSetBorderConfigBorderType(acldvppBorderConfig *borderConfig, acldvppBorderType borderType);**
Python函数	**ret = acl.media.dvpp_set_border_config_border_type(border_config, border_type)**
函数功能	设置边界填充配置参数。同步接口。
输入说明	**border_config**输出 表示边界填充配置数据。需提前调用acl.media.dvpp_create_border_config接口创建acldvppBorderConfig类型的数据 **border_type**：填充枚举类型。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.7.5.48 函数：dvpp_set_border_config_top

C函数原型	**aclError acldvppSetBorderConfigTop(acldvppBorderConfig *borderConfig, uint32_t top);**
Python函数	**ret = acl.media.dvpp_set_border_config_top(border_config, top)**
函数功能	设置边界填充配置参数。同步接口。
输入说明	**border_config**：表示边界填充配置数据。需提前调用acl.media.dvpp_create_border_config接口创建acldvppBorderConfig类型的数据 **top**：图像上方填充的像素数。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.7.5.49 函数：dvpp_set_border_config_bottom

C函数原型	**aclError acldvppSetBorderConfigBottom(acldvppBorderConfig *borderConfig, uint32_t bottom);**
Python函数	**ret = acl.media.dvpp_set_border_config_bottom(border_config, bottom)**
函数功能	设置边界填充配置参数。同步接口。
输入说明	**border_config**：表示边界填充配置数据。需提前调用acl.media.dvpp_create_border_config接口创建acldvppBorderConfig类型的数据 **bottom**：图像下方填充的像素数。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.7.5.50 函数：dvpp_set_border_config_left

C函数原型	**aclError acldvppSetBorderConfigLeft(acldvppBorderConfig *borderConfig, uint32_t left);**
Python函数	**ret = acl.media.dvpp_set_border_config_left(border_config, left)**
函数功能	设置边界填充配置参数。同步接口。
输入说明	**border_config**输出 表示边界填充配置数据。需提前调用acl.media.dvpp_create_border_config接口创建acldvppBorderConfig类型的数据 **left**：图像左方填充的像素数。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.7.5.51 函数：dvpp_set_border_config_right

C函数原型	**aclError acldvppSetBorderConfigRight(acldvppBorderConfig *borderConfig, uint32_t right)**
Python函数	**ret = acl.media.dvpp_set_border_config_right(border_config, right)**
函数功能	设置边界填充配置参数。同步接口。
输入说明	**border_config**：表示边界填充配置数据。需提前调用acl.media.dvpp_create_border_config接口创建acldvppBorderConfig类型的数据 **right**：图像右方填充的像素数。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.7.5.52 函数：dvpp_get_border_config_value

C函数原型	**double acldvppGetBorderConfigValue(const acldvppBorderConfig *borderConfig, uint32_t index);**
Python函数	**value = acl.media.dvpp_get_border_config_value(border_config, index)**
函数功能	获取边界填充配置参数。同步接口。
输入说明	**border_config**：表示边界填充配置数据。 **index**：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	**value**：表示对应颜色分量的值。
约束说明	无
注意事项	无

10.7.5.53 函数：dvpp_get_border_config_border_type

C函数原型	**acldvppBorderType acldvppGetBorderConfigBorderType(const acldvppBorderConfig *borderConfig);**
Python函数	**border_type = acl.media.dvpp_get_border_config_border_type(border_config)**

函数功能	获取边界填充配置参数。同步接口。
输入说明	**border_config**：表示边界填充配置数据。
返回值说明	**border_type** ：填充枚举类型。
约束说明	无
注意事项	无

10.7.5.54 函数：dvpp_get_border_config_top

C函数原型	**uint32_t acldvppGetBorderConfigTop(const acldvppBorderConfig *borderConfig);**
Python函数	**top = acl.media.dvpp_get_border_config_top(border_config)**
函数功能	获取边界填充配置参数。同步接口。
输入说明	**border_config**：表示边界填充配置数据。
返回值说明	**top**：图像上方填充的像素数。
约束说明	无
注意事项	无

10.7.5.55 函数：dvpp_get_border_config_bottom

C函数原型	**uint32_t acldvppGetBorderConfigBottom(const acldvppBorderConfig *borderConfig);**
Python函数	**bottom = acl.media.dvpp_get_border_config_bottom(border_config)**
函数功能	获取边界填充配置参数。同步接口。
输入说明	**border_config**：表示边界填充配置数据。
返回值说明	**bottom**：图像下方填充的像素数。
约束说明	无
注意事项	无

10.7.5.56 函数：dvpp_get_border_config_left

C函数原型	**uint32_t acldvppGetBorderConfigLeft(const acldvppBorderConfig *borderConfig);**

Python函数	**left= acl.media.dvpp_get_border_config_left(border_config)**
函数功能	获取边界填充配置参数。同步接口。
输入说明	**border_config**：表示边界填充配置数据。
返回值说明	**left**：图像左方填充的像素数。
约束说明	无
注意事项	无

10.7.5.57 函数：dvpp_get_border_config_right

C函数原型	**uint32_t acldvppGetBorderConfigRight(const acldvppBorderConfig *borderConfig)**
Python函数	**right = acl.media.dvpp_get_border_config_right(border_config)**
函数功能	获取边界填充配置参数。同步接口。
输入说明	**border_config**：表示边界填充配置数据。需先调用**acl.media.dvpp_create_border_config**接口获取边界填充数据。
返回值说明	**right**：图像右方填充的像素数。
约束说明	无
注意事项	无

10.7.6 VPC 功能

10.7.6.1 功能说明

功能说明

VPC（vision preprocessing core）功能包括：

- **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。
- **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。

  Atlas 200/500 A2推理产品，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 200/500 A2 推理产品)

图 10-1 VPC 功能示意图（抠图+缩放+叠加）

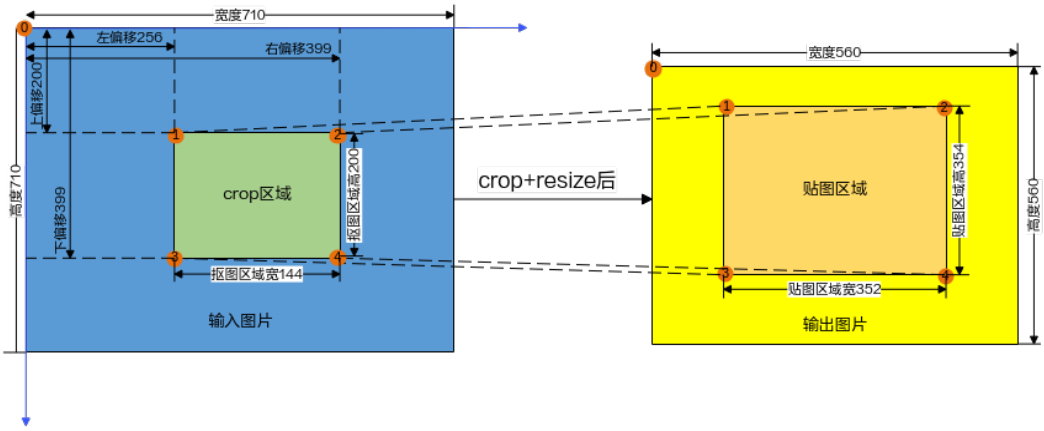


图 10-2 VPC 功能示意图（拼接）

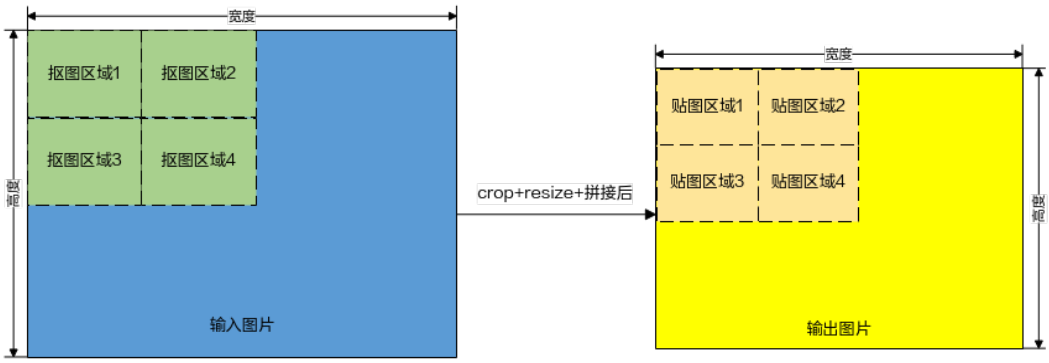
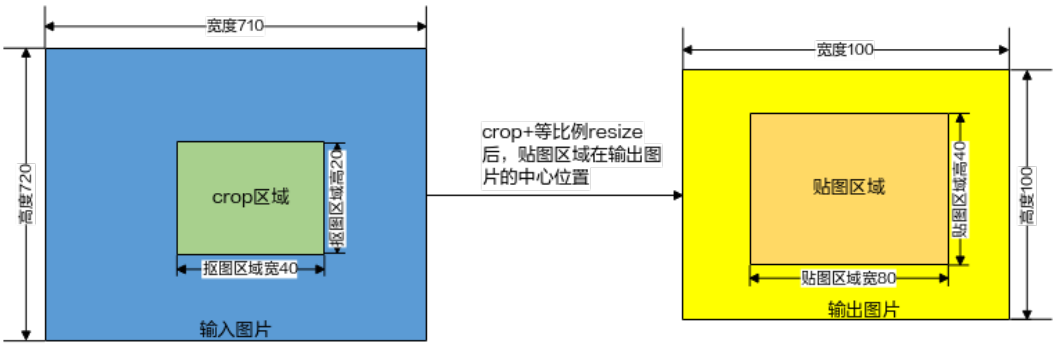


图 10-3 等比例缩放（贴图区域在输出图片的中心位置），即缩放前后图片的宽高比例相同



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

10.7.6.2 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，支持如下分辨率的输入图片。

  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**10.7.6 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

昇腾AI处理器	分辨率范围
Atlas 200/500 A2推理产品	10*6~4096*8192

图片格式、宽高对齐、内存约束

VPC在处理图片时，需调用**acl.media.dvpp_malloc**接口申请Device上的输入、输出内存，调用**acl.media.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、宽高对齐约束如下，参见**表10-8**、**表10-9**，其中，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**10.7.2 基本概念**。
- 宽stride最小10、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 10-8 输入图片格式、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
YUV400 8bit	无对齐要求	宽stride无对齐要求，与宽相同即可；高stride无对齐要求，与高相同即可。 内存大小（单位Byte）= 宽stride * 高stride

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
YUV420SP NV12 8bit	宽2对齐 高2对齐	宽stride无对齐要求，与宽相同即可；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无对齐要求，与宽相同即可；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无对齐要求，与宽相同即可；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		
YUV422Packed YVYU 8bit		
YUV422Packed VYUY 8bit		
YUV444Packed 8bit	无对齐要求	宽stride为宽乘以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		
YUV440SP 8bit	宽无对齐要求 高2对齐	宽stride无对齐要求，与宽相同即可；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 10-9 输出图片格式、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
YUV400 8bit	无对齐要求	宽stride无对齐要求，与宽相同即可；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无对齐要求，与宽相同即可；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无对齐要求，与宽相同即可；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

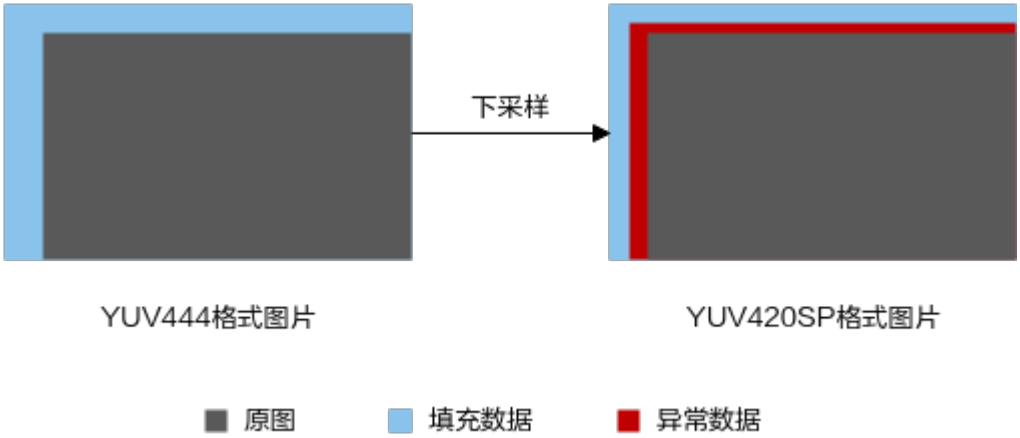
- 抠图区域不超出输入图片区域。
- 贴图区域不超出输出图片区域，最大贴图个数256个。
- 抠图、贴图区域的奇数、偶数限制为：

  抠图起始坐标无奇数、偶数限制；当输出图片格式为YUV420时，贴图区奇数、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；当输出图片格式为YUV422时，贴图区奇数、偶数限制为：左偏移为偶数、右偏移为奇数。

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 10-4 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。

10.7.6.3 性能指标说明

性能指标说明

单个Device场景下的性能指标参考如下（1路对应一个通道，一个通道对应一个线程，或者n路对应一个通道，一个通道对应n个线程）：

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<2）	n*800fps
• 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≥2）	1600fps

场景举例	总帧率
• 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<2）	n*200fps
• 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≥2）	400fps
• 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<2）	n*100fps
• 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≥2）	200fps

说明

调用VPC批处理接口（接口命名中包含batch，例如**acl.media.dvpp_vpc_batch_crop_async**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。

10.7.6.4 函数：dvpp_vpc_resize_async

C函数原型	**aclError acldvppVpcResizeAsync(acldvppChannelDesc *channelDesc, acldvppPicDesc *inputDesc, acldvppPicDesc *outputDesc, acldvppResizeConfig *resizeConfig, aclrtStream stream)**
Python函数	**ret = acl.media.dvpp_vpc_resize_async(dvpp_channel_desc, input_dvpp_pic_desc, output_dvpp_pic_desc, dvpp_resize_config, stream)**
函数功能	将输入图片缩放到输出图片大小，异步接口。
输入说明	**dvpp_channel_desc**：int，指定通道描述信息的指针地址，与调用**acl.media.dvpp_create_channel**接口创建通道时指定的dvpp_channel_desc保持一致。 **input_dvpp_pic_desc**：int，输入图片信息的指针地址。 ● 调用**acl.media.dvpp_create_pic_desc**接口创建图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc系列接口设置图片描述参数（例如，图片格式、宽、高等）。 **output_dvpp_pic_desc**：int，输出图片信息的指针地址。 output_dvpp_pic_desc参数作为输入时，需要用户调用如下接口： ● 调用**acl.media.dvpp_create_pic_desc**接口创建图片描述信息。 ● 调用acl.dvpp.set_pic_desc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）。 **dvpp_resize_config**：int，指定图片缩放配置数据的指针地址。 调用**acl.media.dvpp_create_resize_config**接口创建图片缩放配置数据。 **stream**：int，指定Stream对象。
返回值说明	**ret**：int，错误码。 ● 返回0表示任务下发成功。 ● 返回非0表示任务下发失败。

约束说明

昇腾AI处理器	约束
Atlas 200/500 A2推理产品	● 关于输入、输出图片分辨率： – 输入图片分辨率：10*6~8192*8192 – 输出图片分辨率：10*6~4096*8192 ● 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**。

10.7.6.5 函数：dvpp_vpc_crop_async

C函数原型	**aclError acldvppVpcCropAsync(acldvppChannelDesc *channelDesc, acldvppPicDesc *inputDesc, acldvppPicDesc *outputDesc, acldvppRoiConfig *cropArea, aclrtStream stream)**
Python函数	**ret = acl.media.dvpp_vpc_crop_async(dvpp_channel_desc, input_dvpp_pic_desc, output_dvpp_pic_desc, crop_area, stream)**
函数功能	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，异步接口。 输出图片区域与crop_area不一致时会对图片再做一次缩放操作。 Atlas 200/500 A2推理产品，默认缩放算法为“业界通用的Bilinear算法”。
输入说明	**dvpp_channel_desc**：int，指定通道描述信息的指针地址，与调用**acl.media.dvpp_create_channel**接口创建通道时指定的dvpp_channel_desc保持一致。 **input_dvpp_pic_desc**：int，输入图片信息的指针地址。 ● 调用**acl.media.dvpp_create_pic_desc**接口创建图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc系列接口设置图片描述参数（例如，图片格式、宽、高等）。 **output_dvpp_pic_desc**：int，输出图片信息的指针地址。 outputDesc参数作为输入时，需要用户调用如下接口： ● 调用**acl.media.dvpp_create_pic_desc**接口创建图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc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）。 **crop_area**：int，抠图区域位置的指针地址。调用**acl.media.dvpp_create_roi_config**接口创建区域位置数据。 **stream**：int，指定Stream对象。
返回值说明	**ret**：int，错误码。 ● 返回0，表示任务下发成功。 ● 返回非0，表示任务下发失败。

约束说明

昇腾AI处理器	约束
Atlas 200/500 A2推理产品	● 关于输入、输出图片分辨率： – 输入图片分辨率：10*6~16384*16384 当输出图片宽度大于4096时，输入图片最小分辨率为128*16。 – 输出图片分辨率：10*6~16384*16384 ● 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**。 ● 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。

10.7.6.6 函数：dvpp_vpc_crop_resize_async

C函数原型	**aclError acldvppVpcCropResizeAsync(acldvppChannelDesc *channelDesc, acldvppPicDesc *inputDesc, acldvppPicDesc *outputDesc, acldvppRoiConfig *cropArea, acldvppResizeConfig *resizeConfig, aclrtStream stream)**
Python函数	**ret = acl.media.dvpp_vpc_crop_resize_async(dvpp_channel_desc, input_dvpp_pic_desc, output_dvpp_pic_desc, crop_area, resize_config, stream)**
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crop_area不一致时会对图片再做一次缩放操作，按照resize_config处配置的缩放算法。

输入说明	**dvpp_channel_desc**：int，指定通道描述信息的指针地址，与调用**acl.media.dvpp_create_channel**接口创建通道时指定的dvpp_channel_desc保持一致。 **input_dvpp_pic_desc**：int，输入图片信息的指针地址。 ● 调用**acl.media.dvpp_create_pic_desc**接口创建图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc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）。 **output_dvpp_pic_desc**：int，输出图片信息的指针地址。 ● 调用**acl.media.dvpp_create_pic_desc**接口创建图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc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）。 **crop_area**：int，抠图区域位置的指针地址。调用**acl.media.dvpp_create_roi_config**接口创建区域位置数据。 **resize_config**：int，指定图片缩放配置数据的指针地址。需提前调用**acl.media.dvpp_create_resize_config**接口创建图片缩放配置数据。 **stream**：int，指定Stream对象。
返回值说明	**ret**：int，错误码。 ● 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**。

约束说明

昇腾AI处理器	约束
Atlas 200/500 A2推理产品	● 关于输入、输出图片分辨率： – 输入图片分辨率：10*6~8192*8192 – 输出图片分辨率：10*6~4096*8192 ● 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**。

10.7.6.7 函数：dvpp_vpc_batch_crop_async

C函数原型	**aclError acldvppVpcBatchCropAsync(acldvppChannelDesc *channelDesc, acldvppBatchPicDesc *srcBatchPicDescs, uint32_t *roiNums, uint32_t size, acldvppBatchPicDesc *dstBatchPicDescs, acldvppRoiConfig *cropAreas[], aclrtStream stream)**

Python函数	**dst_batch_pic_descs, ret = acl.media.dvpp_vpc_batch_crop_async(dvpp_channel_desc, src_batch_pic_descs, roi_nums_list, dst_batch_pic_descs, crop_area_list，stream)**
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 200/500 A2推理产品，默认缩放算法为“业界通用的Bilinear算法”。
输入说明	**dvpp_channel_desc**：int，指定通道描述信息的指针地址，与调用**acl.media.dvpp_create_channel**接口创建通道时指定的dvpp_channel_desc保持一致。 **src_batch_pic_descs**：int，指定批量输入图片描述信息的指针地址。 ● 调用**acl.media.dvpp_create_batch_pic_desc**接口创建批量图片描述信息。 ● 调用**acl.media.dvpp_get_pic_desc**接口获取指定图片的图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc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）。 **roi_nums_list**：list，表示抠图的数量的列表，当前最大抠图数量为256。 ● roi_nums_list总和小于等于256，与dst_batch_pic_descs中的batch_size值保持一致。 ● roi_nums_list[0] + ... + roi_nums_list[size - 1] <= 256 **dst_batch_pic_descs**：int，指定批量输出图片描述信息的指针地址。 ● 用**acl.media.dvpp_create_batch_pic_desc**接口创建批量图片描述信息。 ● 调用**acl.media.dvpp_get_pic_desc**接口获取指定图片的图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc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）。 **crop_area_list**：list，抠图区域位置数组。调用**acl.media.dvpp_create_roi_config**接口创建区域位置数据。 **stream**：int，指定Stream。
返回值说明	**dst_batch_pic_descs**：int，批量输出图片描述信息的指针地址。 **ret**：int，错误码。 ● 返回0，表示任务下发成功。 ● 返回非0，表示任务下发失败。

约束说明

昇腾AI处理器	约束
Atlas 200/500 A2推理产品	● 关于输入、输出图片分辨率： – 输入图片分辨率：10*6~8192*8192 – 输出图片分辨率：10*6~4096*8192 ● 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。

10.7.6.8 函数：dvpp_vpc_batch_crop_resize_async

C函数原型	**aclError acldvppVpcBatchCropResizeAsync(acldvppChannelDesc *channelDesc, acldvppBatchPicDesc *srcBatchPicDescs, uint32_t *roiNums, uint32_t size, acldvppBatchPicDesc *dstBatchPicDescs, acldvppRoiConfig *cropAreas[], acldvppResizeConfig resizeConfig, aclrtStream stream)**
Python函数	**ret = acl.media.dvpp_vpc_batch_crop_resize_async(dvpp_channel_desc, src_batch_pic_descs, roi_nums_list，dst_batch_pic_descs, crop_area_list，resize_config, stream)**
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resize_config处配置的缩放算法。异步接口。

输入说明	**dvpp_channel_desc**：int，指定通道描述信息，与调用**acl.media.dvpp_create_channel**接口创建通道时指定的dvpp_channel_desc保持一致。 **src_batch_pic_descs**：int，指定批量输入图片描述信息。 ● 调用**acl.media.dvpp_create_batch_pic_desc**接口创建批量图片描述信息。 ● 调用**acl.media.dvpp_get_pic_desc**接口获取指定图片的图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc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）。 **roi_nums_list**：list，表示抠图的数量的list，当前最大抠图数量为256。 ● 列表总和小于等于256，与dst_batch_pic_descs中的batch_size值保持一致 ● roi_nums_list[0] + ... + roi_nums_list[size - 1] <= 256 **dst_batch_pic_descs**：int，指定批量输出图片描述信息。 ● 调用**acl.media.dvpp_create_batch_pic_desc**接口创建批量图片描述信息。 ● 调用**acl.media.dvpp_get_pic_desc**接口获取指定图片的图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc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）。 **crop_area_list**：list，抠图区域位置数组。 调用**acl.media.dvpp_create_roi_config**接口创建区域位置数据。 **resize_config**：int，指定图片缩放配置数据。 需提前调用**acl.media.dvpp_create_resize_config**接口创建图片缩放配置数据。 **stream**：int，指定Stream。
返回值说明	**dst_batch_pic_descs**：int，批量输出图片描述信息。 **ret**：int，错误码。 ● 返回0，表示任务下发成功。 ● 返回非0，表示任务下发失败。

约束说明

昇腾AI处理器	约束
Atlas 200/500 A2推理产品	● 关于输入、输出图片分辨率： – 输入图片分辨率：10*6~8192*8192 – 输出图片分辨率：10*6~4096*8192 ● 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**。

10.7.6.9 函数：dvpp_vpc_crop_and_paste_async

C函数原型	**aclError acldvppVpcCropAndPasteAsync(acldvppChannelDesc *channelDesc, acldvppPicDesc *inputDesc, acldvppPicDesc *outputDesc, acldvppRoiConfig *cropArea, acldvppRoiConfig *pasteArea, aclrtStream stream)**
Python函数	**ret = acl.media.dvpp_vpc_crop_and_paste_async(dvpp_channel_desc, input_dvpp_pic_desc, output_dvpp_pic_desc, crop_area, paste_area, stream)**
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paste_area的宽高与crop_area的宽高不一致时会对图片做一次缩放操作。 Atlas 200/500 A2推理产品，默认缩放算法为“业界通用的Bilinear算法”。
输入说明	**dvpp_channel_desc**：int，指定通道描述信息，与调用**acl.media.dvpp_create_channel**接口创建通道时指定的dvpp_channel_desc保持一致。 **input_dvpp_pic_desc**：int，输入图片信息。 ● 调用**acl.media.dvpp_create_pic_desc**接口创建图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc系列接口设置图片描述参数（例如，图片格式、宽、高等）。 **output_dvpp_pic_desc**：int，输出图片信息。 ● 调用**acl.media.dvpp_create_pic_desc**接口创建图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc系列接口设置图片描述参数（例如，图片格式、宽、高等）。 **crop_area**：int，抠图区域位置。 调用**acl.media.dvpp_create_roi_config**接口创建区域位置数据。 **paste_area**：int，贴图区域位置。 ● 调用**acl.media.dvpp_create_roi_config**接口创建区域位置数据。 ● 贴图区域左偏移需要16对齐。 **stream**：int，指定Stream对象的。
返回值说明	**ret**：int，错误码。 ● 返回0，表示任务下发成功。 ● 返回非0，表示任务下发失败。

约束说明

昇腾AI处理器	约束
Atlas 200/500 A2推理产品	• 关于图片分辨率、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**。 • 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，贴图的顶部偏移、左侧偏移建议为偶数。

10.7.6.10 函数：dvpp_vpc_crop_resize_paste_async

C函数原型	**aclError acldvppVpcCropResizePasteAsync(acldvppChannelDesc *channelDesc, acldvppPicDesc *inputDesc, acldvppPicDesc *outputDesc, acldvppRoiConfig *cropArea, acldvppRoiConfig *pasteArea, acldvppResizeConfig *resizeConfig, aclrtStream stream)**
Python函数	**ret = acl.media.dvpp_vpc_crop_resize_paste_async(dvpp_channel_desc, input_dvpp_pic_desc, output_dvpp_pic_desc, crop_area, paste_area, resize_config, stream)**
函数功能	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。异步接口。 crop_area的宽高与paste_area宽高不一致时会对图片再做一次缩放操作，按照resize_config处配置的缩放算法。

输入说明	**dvpp_channel_desc**：int，指定通道描述信息，与调用**acl.media.dvpp_create_channel**接口创建通道时指定的dvpp_channel_desc保持一致。 **input_dvpp_pic_desc**：int，输入图片信息。 ● 调用**acl.media.dvpp_create_pic_desc**接口创建图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc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）。 **output_dvpp_pic_desc**：int，输出图片信息。 ● 调用**acl.media.dvpp_create_pic_desc**接口创建图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc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）。 **crop_area**：int，抠图区域位置。 调用**acl.media.dvpp_create_roi_config**接口创建区域位置数据。 **paste_area**：int，贴图区域位置。 ● 调用**acl.media.dvpp_create_roi_config**接口创建区域位置数据。 ● 贴图区域左偏移需要16对齐。 **resize_config**：int，指定图片缩放配置数据。 需提前调用**acl.media.dvpp_create_resize_config**接口创建图片缩放配置数据。 **stream**：int，指定Stream对象的。
返回值说明	**ret**：int，错误码。 ● 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**。

约束说明

昇腾AI处理器	约束
Atlas 200/500 A2推理产品	● 关于图片分辨率、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**。 ● 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，贴图的顶部偏移、左侧偏移建议为偶数。

10.7.6.11 函数：dvpp_vpc_batch_crop_and_paste_async

C函数原型	**aclError acldvppVpcBatchCropAndPasteAsync(acldvppChannelDesc *channelDesc, acldvppBatchPicDesc *srcBatchPicDesc, uint32_t *roiNums, uint32_t size, acldvppBatchPicDesc *dstBatchPicDesc, acldvppRoiConfig *cropAreas[], acldvppRoiConfig *pasteAreas[], aclrtStream stream)**
Python函数	**dst_batch_pic_descs, ret = acl.media.dvpp_vpc_batch_crop_and_paste_async(dvpp_channel_desc, src_batch_pic_descs, roi_nums_list，dst_batch_pic_descs, crop_area_list，paste_area_list，stream)**
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paste_area的宽高与crop_area的宽高不一致时会对图片做一次缩放操作。 Atlas 200/500 A2推理产品，默认缩放算法为“业界通用的Bilinear算法”。

输入说明	**dvpp_channel_desc**：int，指定通道描述信息，与调用**acl.media.dvpp_create_channel**接口创建通道时指定的dvpp_channel_desc保持一致。 **src_batch_pic_descs**：int，指定批量输入图片描述信息。 ● 调用**acl.media.dvpp_create_batch_pic_desc**接口创建批量图片描述信息。 ● 调用**acl.media.dvpp_get_pic_desc**接口获取指定图片的图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc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）。 **roi_nums_list**：list，表示抠图的数量的list，当前最大抠图数量为256。 ● roi_nums_list总和小于等于256，与dst_batch_pic_descs中的batch_size值保持一致。 ● roi_nums_list[0] + ... + roi_nums_list[size - 1] <= 256 **dst_batch_pic_descs**：int，指定批量输出图片描述信息。 ● 调用**acl.media.dvpp_create_batch_pic_desc**接口创建批量图片描述信息。 ● 调用**acl.media.dvpp_get_pic_desc**接口获取指定图片的图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc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）。 **crop_area_list**：list，抠图区域位置数组。 ● 调用**acl.media.dvpp_create_roi_config**接口创建区域位置数据。 ● crop_area、paste_area中元素个数与dst_batch_pic_descs中的batch_size值相等。 **paste_area_list**：list，贴图区域位置。 ● 调用**acl.media.dvpp_create_roi_config**接口创建区域位置数据。 ● 贴图区域左偏移需要16对齐。 ● crop_area、paste_area中元素个数与dst_batch_pic_descs中的batch_size值相等。 **stream**：int，指定Stream。
返回值说明	**dst_batch_pic_descs**：int，批量输出图片描述信息。 **ret**：int，错误码。 ● 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**。

约束说明

昇腾AI处理器	约束
Atlas 200/500 A2推理产品	• 关于图片分辨率、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**。 • 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，贴图的顶部偏移、左侧偏移建议为偶数。

10.7.6.12 函数：dvpp_vpc_batch_crop_resize_paste_async

C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)**
Python函数	**ret = acl.media.dvpp_vpc_batch_crop_resize_paste_async(dvpp_channel_desc, src_batch_pic_descs, roi_nums_list，dst_batch_pic_descs, crop_area_list，paste_area_list，resize_config, stream)**
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paste_area的宽高与crop_area的宽高不一致时会对图片做一次缩放操作，按照resize_config处配置的缩放算法。

输入说明	**dvpp_channel_desc**：int，指定通道描述信息，与调用**acl.media.dvpp_create_channel**接口创建通道时指定的dvpp_channel_desc保持一致。 **src_batch_pic_descs**：int，指定批量输入图片描述信息。 ● 调用**acl.media.dvpp_create_batch_pic_desc**接口创建批量图片描述信息。 ● 调用**acl.media.dvpp_get_pic_desc**接口获取指定图片的图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc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）。 **roi_nums_list**：list，指定每个输入图片的抠图数量，当前最大抠图数量为256。 ● 列表总和小于等于256，与dst_batch_pic_descs中的batch_size值保持一致。 ● roi_nums_list[0] + ... + roi_nums_list[size - 1] <= 256 **dst_batch_pic_descs**：int，指定批量输出图片描述信息。 ● 调用**acl.media.dvpp_create_batch_pic_desc**接口创建批量图片描述信息。 ● 调用**acl.media.dvpp_get_pic_desc**接口获取指定图片的图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc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）。 **crop_area_list**：list，抠图区域位置数组。 ● 调用**acl.media.dvpp_create_roi_config**接口创建区域位置数据。 ● crop_area_list、paste_area_list中元素个数与dst_batch_pic_descs中的batch_size值相等。 **paste_area_list**：list，贴图区域位置。 ● 调用**acl.media.dvpp_create_roi_config**接口创建区域位置数据。 ● 贴图区域左偏移需要16对齐。 ● crop_area_list、paste_area_list中元素个数与dst_batch_pic_descs中的batch_size值相等。 **resize_config**：int，指定图片缩放配置数据。 需提前调用**acl.media.dvpp_create_resize_config**接口创建图片缩放配置数据。 **stream**：int，指定Stream。
返回值说明	**ret**：int，错误码。 ● 返回0表示任务下发成功。 ● 返回非0表示任务下发失败。

约束说明

昇腾AI处理器	约束
Atlas 200/500 A2推理产品	• 关于图片分辨率、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**。 • 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，贴图的顶部偏移、左侧偏移建议为偶数。

10.7.6.13 函数：dvpp_vpc_batch_crop_resize_make_border_async

C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)**
Python函数	**ret = acl.media.dvpp_vpc_batch_crop_resize_make_border_async(dvpp_channel_desc, src_batch_pic_descs, roi_nums_list, dst_batch_pic_descs, crop_area_list, border_cfgs_list, resize_config, stream)**
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，对每个子图进行进行缩放，扩边填充后输出至目标图像内存地址。异步接口。

输入说明	**dvpp_channel_desc**：int，指定通道描述信息，与调用**acl.media.dvpp_create_channel**接口创建通道时指定的dvpp_channel_desc保持一致。 **src_batch_pic_descs**：int，指定批量输入图片描述信息。 ● 调用**acl.media.dvpp_create_batch_pic_desc**接口创建批量图片描述信息。 ● 调用**acl.media.dvpp_get_pic_desc**接口获取指定图片的图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc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）。 **roi_nums_list**：list，指定每个输入图片的抠图数量，当前最大抠图数量为256。 ● 列表总和小于等于256，与dst_batch_pic_descs中的batch_size值保持一致。 ● roi_nums_list[0] + ... + roi_nums_list[size - 1] <= 256 **dst_batch_pic_descs**：int，指定批量输出图片描述信息。 ● 调用**acl.media.dvpp_create_batch_pic_desc**接口创建批量图片描述信息。 ● 调用**acl.media.dvpp_get_pic_desc**接口获取指定图片的图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc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）。 **crop_area_list**：list，抠图区域位置数组。 ● 调用**acl.media.dvpp_create_roi_config**接口创建区域位置数据。 ● crop_area_list、paste_area_list中元素个数与dst_batch_pic_descs中的batch_size值相等。 **border_cfgs_list**：list，贴图区域位置。 ● 扩边填充配置数组。 ● border_cfgs数组中的元素与crop_areas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。 **resize_config**：int，指定图片缩放配置数据。 需提前调用**acl.media.dvpp_create_resize_config**接口创建图片缩放配置数据。本接口不支持“华为自研的高阶滤波算法”。 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**：int，指定Stream。
返回值说明	**ret**：int，错误码。 ● 返回0，表示任务下发成功。 ● 返回非0，表示任务下发失败。

约束说明	• VPC功能中，关于图片分辨率、图片格式、宽高对齐、内存等约束，请参见**10.7.6.2 约束说明**。 • 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。

约束说明

昇腾AI处理器	约束
所有昇腾AI处理器	• 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。
Atlas 200/500 A2推理产品	关于图片分辨率、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**。

10.7.6.14 函数：dvpp_vpc_convert_color_async

C函数原型	**aclError acldvppVpcConvertColorAsync(acldvppChannelDesc *channelDesc, acldvppPicDesc *inputDesc, acldvppPicDesc *outputDesc, aclrtStream stream)**
Python函数	**ret = acl.media.dvpp_vpc_convert_color_async(dvpp_channel_desc, input_dvpp_pic_desc, output_dvpp_pic_desc, stream)**
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。

输入说明	**dvpp_channel_desc**：int，指定通道描述信息，与调用**acl.media.dvpp_create_channel**接口创建通道时指定的dvpp_channel_desc保持一致。 **input_dvpp_pic_desc**：int，输入图片信息。 ● 调用**acl.media.dvpp_create_pic_desc**接口创建图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc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）。 **output_dvpp_pic_desc**：int，输出图片信息。 作为输入时，需要用户调用如下接口： ● 调用**acl.media.dvpp_create_pic_desc**接口创建图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc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**：int，指定Stream对象的。
返回值说明	**ret**：int，错误码。 ● 返回0表示任务下发成功。 ● 返回非0表示任务下发失败。
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**。

10.7.6.15 函数：dvpp_vpc_pyr_down_async

C函数原型	**aclError acldvppVpcPyrDownAsync(acldvppChannelDesc *channelDesc, acldvppPicDesc *inputDesc, acldvppPicDesc *outputDesc, void *reserve, aclrtStream stream)**
Python函数	**ret = acl.media.dvpp_vpc_pyr_down_async(dvpp_channel_desc, input_dvpp_pic_desc, output_dvpp_pic_desc, reserve, stream)**
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 200/500 A2推理产品，不支持该接口。

输入说明	**dvpp_channel_desc**：int，指定通道描述信息，与调用**acl.media.dvpp_create_channel**接口创建通道时指定的dvpp_channel_desc保持一致。 **input_dvpp_pic_desc**：int，输入图片信息。 ● 调用**acl.media.dvpp_create_pic_desc**接口创建图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc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）。 **output_dvpp_pic_desc**：int，输出图片信息。 作为输入时，需要用户调用如下接口： ● 调用**acl.media.dvpp_create_pic_desc**接口创建图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc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**：int，预留金字塔缩放配置参数。 **stream**：int，指定Stream。
返回值说明	**ret**：int，错误码。 ● 返回0，表示任务下发成功。 ● 返回非0，表示任务下发失败。
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**。

10.7.6.16 函数：dvpp_create_roi_config

C函数原型	**acldvppRoiConfig *acldvppCreateRoiConfig(uint32_t left, uint32_t right, uint32_t top, uint32_t bottom)**
Python函数	**dvpp_roi_config = acl.media.dvpp_create_roi_config(left, right, top, bottom)**
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。同步接口。 如需销毁acldvppRoiConfig类型的数据，请参见**acl.media.dvpp.destroy_roi_config**。

输入说明	**left**：int，左偏移。 **right**：int，右偏移。 **top**：int，上偏移。 **bottom**：int，下偏移。
返回值说明	**dvpp_roi_config** ：int，dvpp_roi_config的指针地址。 ● 返回非0值，表示成功。 ● 返回0，表示失败。

10.7.6.17 函数：dvpp_destroy_roi_config

C函数原型	**aclError acldvppDestroyRoiConfig(acldvppRoiConfig *roiConfig)**
Python函数	**ret = acl.media.dvpp_destroy_roi_config(roi_config)**
函数功能	销毁通过**acl.media.dvpp_create_roi_config**接口创建的数据。同步接口。
输入说明	**roi_config**：int，指定待销毁的数据。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。

10.7.6.18 函数：dvpp_set_roi_config

C函数原型	**aclError acldvppSetRoiConfig(acldvppRoiConfig *config, uint32_t left, uint32_t right, uint32_t top, uint32_t bottom)**
Python函数	**ret = acl.media.dvpp_set_roi_config(roi_config, left, right, top, bottom)**
函数功能	设置某个区域的位置信息。同步接口。
输入说明	**roi_config**：int，指定区域位置数据。调用**acl.media.dvpp_create_roi_config**接口创建的数据。 **left**：int，左偏移。 **right**：int，右偏移。 **top**：int，上偏移。 **bottom**：int，下偏移。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。

10.7.6.19 函数：dvpp_set_roi_config_left

C函数原型	**aclError acldvppSetRoiConfigLeft(acldvppRoiConfig *config, uint32_t left)**
Python函数	**ret = acl.media.dvpp_set_roi_config_left(roi_config, left)**
函数功能	设置某个区域的位置信息：左偏移。
输入说明	**roi_config**：int，描述某个区域位置的数据。调用acl.media.dvpp_create_roi_config接口创建的数据。 **left**：int，左偏移。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.6.20 函数：dvpp_set_roi_config_right

C函数原型	**aclError acldvppSetRoiConfigRight (acldvppRoiConfig *config, uint32_t right)**
Python函数	**ret = acl.media.dvpp_set_roi_config_right (roi_config, right)**
函数功能	设置某个区域的位置信息：右偏移。
输入说明	**roi_config**：int，描述某个区域位置的数据。调用acl.media.dvpp_create_roi_config接口创建的数据。 **right**：int，右偏移。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.6.21 函数：dvpp_set_roi_config_top

C函数原型	**aclError acldvppSetRoiConfigTop(acldvppRoiConfig *config, uint32_t top)**
Python函数	**ret = acl.media.dvpp_set_roi_config_top(roi_config, top)**
函数功能	设置某个区域的位置信息：上偏移。
输入说明	**roi_config**：int，描述某个区域位置的数据。调用acl.media.dvpp_create_roi_config接口创建的数据。 **top**：int，上偏移。

返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.6.22 函数：dvpp_set_roi_config_bottom

C函数原型	**aclError acldvppSetRoiConfigBottom(acldvppRoiConfig *config, uint32_t bottom)**
Python函数	**ret = acl.media.dvpp_set_roi_config_bottom(roi_config, bottom)**
函数功能	设置某个区域的位置信息：下偏移。
输入说明	**roi_config**：int，描述某个区域位置的数据。调用acl.media.dvpp_create_roi_config接口创建的数据。 **bottom**：int，下偏移。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.6.23 函数：dvpp_create_resize_config

C函数原型	**acldvppResizeConfig *acldvppCreateResizeConfig()**
Python函数	**dvpp_resize_config = acl.media.dvpp_create_resize_config()**
函数功能	创建acldvppResizeConfig类型的数据，表示图片缩放配置数据。同步接口。 如需销毁acldvppResizeConfig类型的数据，请参见acl.media.dvpp_destroy_resize_config。 Atlas 200/500 A2推理产品，默认缩放算法为“业界通用的Bilinear算法”。
返回值说明	**dvpp_resize_config**：int。 ● 返回非0值，表示成功。 ● 返回0，表示失败。

10.7.6.24 函数：dvpp_destroy_resize_config

C函数原型	**aclError acldvppDestroyResizeConfig(acldvppResizeConfig *resizeConfig)**

Python函数	**ret = acl.media.dvpp_destroy_resize_config(dvpp_resize_config)**
函数功能	销毁通过**acl.media.dvpp_create_resize_config**接口创建的缩放配置数据。同步接口。
输入说明	**dvpp_resize_config**：int，指定待销毁的数据。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。

10.7.6.25 函数：dvpp_set_resize_config_interpolation

C函数原型	**aclError acldvppSetResizeConfigInterpolation(acldvppResizeConfig *resizeConfig, uint32_t interpolation)**
Python函数	**ret = acl.media.dvpp_set_resize_config_interpolation(dvpp_resize_config, interpolation)**
函数功能	设置图片缩放算法。同步接口。
输入说明	**dvpp_resize_config**：int，指定待设置的缩放配置数据。需提前调用**acl.media.dvpp_create_resize_config**接口创建acldvppResizeConfig类型的数据。 **interpolation**：int，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 200/500 A2推理产品，支持如下缩放算法： ● 0：默认值，表示华为自研的高阶滤波算法。 ● 1：业界通用的Bilinear算法（与OpenCV-3.4.2版本算法的计算结果相同）。 ● 2：业界通用的Nearest neighbor 算法（与OpenCV-3.4.2版本算法的计算结果相同）。 ● 3：业界通用的Bilinear算法（与1相同）。 ● 4：业界通用的Nearest neighbor算法（与2相同）。 ● 5：华为自研的高阶滤波算法。 当前该缩放算法仅支持输入图片分辨率范围为10*6~8192*8192、输出图片分辨率的范围为10*6~4096*8192；输入图片格式仅支持YUV、但不包括Planner格式、也不包括YUV400、YUV440，输出图片格式仅支持YUV420SP NV12 8bit、YUV420SP NV21 8bit；宽或高的缩放范围：[1/32, 16]。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。

10.7.6.26 函数：dvpp_get_resize_config_interpolation

C函数原型	**uint32 acldvppGetResizeConfigInterpolation(const acldvppResizeConfig * resizeConfig)**
Python函数	**interpolation = acl.media.dvpp_get_resize_config_interpolation(dvpp_resize_config)**
函数功能	获取图片缩放算法。同步接口。
输入说明	**dvpp_resize_config**：int，指定待获取的缩放配置数据。 需提前调用**acl.media.dvpp_create_resize_config**接口创建acldvppResizeConfig类型的数据，调用**acl.media.dvpp_set_resize_config_interpolation**接口设置缩放算法。
返回值说明	**interpolation**：int，缩放算法，具体请参见**acl.media.dvpp_create_resize_config**。

10.7.7 JPEGD 功能

10.7.7.1 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

  JPEGD 422格式旋转那图片，若原图高为奇数，并且旋转方式为宽高对换，则旋转后宽存在绿边，建议用户将宽向下对齐到偶数使用（即去除绿边），比如原图为200*101，若旋转后为101*200，则建议用户实际使用区域为100*200。

  JPEGD 440格式旋转那图片，若原图宽为奇数，并且旋转方式为宽高对换，则旋转后高存在绿边，建议用户将高向下对齐到偶数使用（即去除绿边），比如原图为201*101，若旋转后为100*201，则建议用户实际使用区域为100*200。

- **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**acl.media.dvpp_jpeg_get_image_info_v2**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**10.7.6.2 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

昇腾AI处理器	分辨率范围
Atlas 200/500 A2 推理产品	最大分辨率：16384*16384，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**acl.media.dvpp_malloc**接口申请Device上的输入、输出内存，调用**acl.media.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。

- 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**acl.media.dvpp_jpeg_predict_dec_size**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**10.7.2 基本概念**。

表 10-10 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
	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		
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 * 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

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
	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

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acl.media.dvpp_set_pic_desc_width和acl.media.dvpp_set_pic_desc_height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。

10.7.7.2 性能指标说明

性能指标说明

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。

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**acl.rt.synchronize_stream**接口。

1080p指分辨率为1920*1080的图片；4K指分辨率为3840*2160的图片。单个Device的基本场景性能指标参考如下（1路对应一个通道，一个通道对应一个线程）：

场景举例	总帧率
1080p*n路（1≤n≤2）	n*256fps

场景举例	总帧率
1080p*n路（n>2）	512fps
4k*n路（1≤n≤2）	n*64fps
4k*n路（n>2）	128fps

10.7.7.3 函数：dvpp_jpeg_decode_async

C函数原型	**aclError acldvppJpegDecodeAsync(acldvppChannelDesc *channelDesc, const void *data, uint32_t size, acldvppPicDesc *outputDesc, aclrtStream stream)**
Python函数	**ret = acl.media.dvpp_jpeg_decode_async(dvpp_channel_desc, data, size, output_dvpp_pic_desc, stream)**
函数功能	解码.jpg、.jpeg、.JPG、.JPEG图片，异步接口。
输入说明	**dvpp_channel_desc**：int，指定通道描述信息，与调用**acl.media.dvpp_create_channel**接口创建通道时指定的dvpp_channel_desc保持一致。 **data**：int，输入图片的内存地址。 **size**：int，输入图片的实际数据大小，单位Byte。 **output_dvpp_pic_desc**：int，输出图片信息。 ● 调用**acl.media.dvpp_create_pic_desc**接口创建图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc系列接口设置输出图片的内存地址、内存大小、格式、width_stride、height_stride。 – 输出图片的内存大小可提前调用**acl.media.dvpp_jpeg_predict_dec_size**接口预估。 – JPEG原图的宽、高可通过**acl.media.dvpp_jpeg_get_image_info**/**acl.media.dvpp_jpeg_get_image_info_v2**接口获取。 – 输出图片格式、输出图片的宽高对齐要求，请参见**10.7.7.1 功能及约束说明**。 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acl.media.dvpp_get_pic_desc系列接口获取输出图片的宽/高。 **说明** 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acl.media.dvpp_get_pic_desc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**：int，指定Stream对象的。

返回值说明	**ret**：int，错误码。 ● 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5.3.3 JPEGD图片解码。

10.7.7.4 函数：dvpp_jpeg_get_image_info

C函数原型	**aclError acldvppJpegGetImageInfo(const void *data, uint32_t size, uint32_t *width, uint32_t *height, int32_t *components)**
Python函数	**width, height, components, ret = acl.media.dvpp_jpeg_get_image_info(data, size)**
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10.14.1 aclError。
输入说明	**data**：int，存放JPEG图片数据的内存的地址。 **size**：int，内存大小，单位为Byte。
返回值说明	**width**：int，图片的宽。 **height**：int，图片的高。 **components**：int，颜色通道个数。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.7.5 函数：dvpp_jpeg_get_image_info_v2

C函数原型	**aclError acldvppJpegGetImageInfoV2(const void *data, uint32_t size, uint32_t *width, uint32_t *height, int32_t *components, acldvppJpegFormat *format)**
Python函数	**width, height, components, format, ret = acl.media.dvpp_jpeg_get_image_info_v2(data, size)**
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10.14.1 aclError。
输入说明	**data**：int，存放JPEG图片数据的内存的地址。 **size**：int，内存大小，单位为Byte。

返回值说明	**width**：int，图片的宽。 **height**：int，图片的高。 **components**：int，颜色通道个数。 **format**：int，JPEG原图编码格式。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.7.6 函数：dvpp_jpeg_predict_dec_size

C函数原型	**aclError acldvppJpegPredictDecSize(const void *data, uint32_t dataSize, acldvppPixelFormat outputPixelFormat, uint32_t *decSize)**
Python函数	**dec_size, ret = acl.media.dvpp_jpeg_predict_dec_size(data, data_size, format)**
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。
输入说明	**data**：int，Host上存放JPEG图片数据的内存的地址。 **data_size**：int，内存大小，单位为Byte。 **format**：int，解码后的输出图片的格式。详情见acl.media.dvpp_set_pic_desc_format介绍。
返回值说明	**dec_size**：int，预估JPEG图片解码后所需的输出内存的大小，单位为Byte。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.8 JPEGE 功能

10.7.8.1 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：

昇腾AI处理器	分辨率范围
Atlas 200/500 A2推理产品	最大分辨率：16384*16384，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**acl.media.dvpp_malloc**接口申请Device上的输入、输出内存，调用**acl.media.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。

- 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**acl.media.dvpp_jpeg_predict_enc_size**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**10.7.2 基本概念**。

表 10-11 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 YUV420SP NV21 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。

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
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			

10.7.8.2 性能指标说明

性能指标说明

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acl.rt.synchronize_stream接口。

1080p指分辨率为1920*1080的图片；4K指分辨率为3840*2160的图片。单个Device的基本场景性能指标参考如下（1路对应一个通道，一个通道对应一个线程）：

场景举例（输入图片格式：YUV420 8bit）	总帧率
1080p*n路	256fps
4k*n路	64fps

10.7.8.3 函数：dvpp_jpeg_encode_async

C函数原型	**aclError acldvppJpegEncodeAsync(acldvppChannelDesc *channelDesc, acldvppPicDesc *inputDesc, const void *data, uint32_t *size, acldvppJpegeConfig *config, aclrtStream stream)**
Python函数	**ret = acl.media.dvpp_jpeg_encode_async(dvpp_channel_desc, input_dvpp_pic_desc, data, size_ptr, dvpp_jpege_config, stream)**
函数功能	将YUV格式图片编码成.jpg图片，异步接口。

输入说明	**dvpp_channel_desc**：int，指定通道描述信息，与调用**acl.media.dvpp_create_channel**接口创建通道时指定的dvpp_channel_desc保持一致。 **input_dvpp_pic_desc**：int，输入图片描述信息。 ● 调用**acl.media.dvpp_create_pic_desc**接口创建图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc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**10.7.8.1 功能及约束说明**。 **data**：int，输出内存地址，存放编码后的数据。 **size_ptr**：int，指向输出内存大小，内存大小单位Byte。 ● 可提前调用**acl.media.dvpp_jpeg_predict_enc_size**接口预估输出内存大小out_buffer_size。 ● 调用np_out_size = np.array([out_buffer_size], dtype=np.int32)创建numpy array类型数据。再调用bytes_data = np_out_size.tobytes()-> \->size_ptr = **acl.util.bytes_to_ptr**(bytes_data)，将数据转换为对象作为入参。注意：函数调用成功后，size_ptr指向的数据（即np_out_size）会发生改变。 **dvpp_jpege_config**：int，表示图片编码配置数据。 ● 调用**acl.media.dvpp_create_jpege_config**接口创建图片编码配置数据，调用**acl.media.dvpp_set_jpege_config_level**接口设置编码配置数据。 **stream**：int，指定Stream对象。
返回值说明	**ret**：int，错误码。 ● 返回0，表示任务下发成功。 ● 返回非0，表示任务下发失败。

10.7.8.4 函数：dvpp_jpeg_predict_enc_size

C函数原型	**aclError acldvppJpegPredictEncSize(const acldvppPicDesc *inputDesc, const acldvppJpegeConfig *config, uint32_t *size)**
Python函数	**size, ret = acl.media.dvpp_jpeg_predict_enc_size(input_dvpp_pic_desc, config)**
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。

输入说明	**input_dvpp_pic_desc**: int，输入图片描述信息。 ● 调用**acl.media.dvpp_create_pic_desc**接口创建图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc系列接口设置图片描述参数（例如，图片格式、宽、高等）。 **config**：int，图片编码配置数据。 调用**acl.media.dvpp_create_jpege_config**接口创建图片编码配置数据。
返回值说明	**size**：int，预估JPEG图片编码后所需的输出内存的大小，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**acl.media.dvpp_jpeg_encode_async**接口的输出参数size获取。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。

10.7.8.5 函数：dvpp_create_jpege_config

C函数原型	**acldvppJpegeConfig *acldvppCreateJpegeConfig()**
Python函数	**dvpp_jpege_config = acl.media.dvpp_create_jpege_config()**
函数功能	创建acldvppJpegeConfig类型的数据，表示图片编码配置数据。同步接口。 如需销毁acldvppJpegeConfig类型的数据，请参见**acl.media.dvpp_destroy_jpege_config**。
返回值说明	**dvpp_jpege_config**：int，dvpp_jpege_config的指针地址。 ● 返回非0值，表示成功。 ● 返回0，表示失败。

10.7.8.6 函数：dvpp_destroy_jpege_config

C函数原型	**aclError acldvppDestroyJpegeConfig(acldvppJpegeConfig *jpegeConfig)**
Python函数	**ret = acl.media.dvpp_destroy_jpege_config(dvpp_jpege_config)**
函数功能	销毁通过**acl.media.dvpp_create_jpege_config**接口创建的编码配置数据。同步接口。
输入说明	**dvpp_jpege_config**：int，指定待销毁的数据。

返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.8.7 函数：dvpp_set_jpege_config_level

C函数原型	**aclError acldvppSetJpegeConfigLevel(acldvppJpegeConfig *jpegeConfig, uint32_t level)**
Python函数	**ret = acl.media.dvpp_set_jpege_config_level(dvpp_jpege_config, level)**
函数功能	设置编码配置数据。同步接口。
输入说明	**dvpp_jpege_config**：int，指定编码配置数据。调用acl.media.dvpp_create_jpege_config接口创建的编码配置数据。 **level**：int，编码质量范围。 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	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.8.8 函数：dvpp_get_jpege_config_level

C函数原型	**uint32_t acldvppGetJpegeConfigLevel(const acldvppJpegeConfig *jpegeConfig)**
Python函数	**level = acl.media.dvpp_get_jpege_config_level(dvpp_jpege_config)**
函数功能	获取编码配置数据。同步接口。
输入说明	**dvpp_jpege_config**：int，指定编码配置数据。调用acl.media.dvpp_create_jpege_config接口创建acldvppJpegeConfig类型的数据，调用acl.media.dvpp_set_jpege_config_level接口设置编码配置数据。
返回值说明	**level**：int，返回编码质量范围。 该参数取值范围：[1, 100]，数值越小输出图片质量越差。

10.7.9 VDEC 功能

10.7.9.1 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：

昇腾AI处理器	分辨率范围
Atlas 200/500 A2 推理产品	对于H264输入码流，最大分辨率8192*8192，最小分辨率128*128。 对于H265输入码流，最大分辨率5400*8192，最小分辨率128*128。

- 输出图片分辨率：

昇腾AI处理器	分辨率范围
Atlas 200/500 A2 推理产品	输出图片最大分辨率8192*8192；当输入码流宽度小于或等于4096时，输出图片最小分辨率为10*6，当输入码流宽度大于4096时，输出图片最小分辨率为128*128。

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**acl.media.dvpp_malloc**/**acl.media.dvpp_free**接口申请/释放内存或调用**acl.rt.malloc**/**acl.rt.free**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**acl.media.dvpp_malloc**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**acl.rt.malloc**接口申请内存。

- 输出内存

  需调用**acl.media.dvpp_malloc**接口申请Device上的输出内存，调用**acl.media.dvpp_free**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**10.7.2 基本概念**。

表 10-12 码流/图片格式、宽高对齐、内存大小约束

输入码流格式	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
• 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。	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		
	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。

10.7.9.2 性能指标说明

性能指标说明

1080p指分辨率为1920*1080的图片。单个Device的基本场景性能指标参考如下（1路对应一个通道，一个通道对应一个线程）：

场景举例	总帧率
720p*n路 (1≤n≤4)	n*600fps
720p*n路 (n>4)	2400fps
1080p*n路 (1≤n≤4)	n*300fps
1080p*n路 (n>4)	1200fps
4k*n路 (1≤n≤4)	n*75fps

场景举例	总帧率
4k*n路(n>4)	300fps

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**acl.media.vdec_set_channel_desc_ref_frame_num**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）
16MB + *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	62.5MB（参考帧数量8个） 41.5MB（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	68MB（参考帧数量8个） 44MB（参考帧数量2个）

10.7.9.3 函数：vdec_send_frame

C函数原型	**aclError aclvdecSendFrame(aclvdecChannelDesc *channelDesc, acldvppStreamDesc *input, acldvppPicDesc *output, aclvdecFrameConfig *config, void* userData)**

Python函数	**ret = acl.media.vdec_send_frame(vdec_channel_desc, dvpp_stream_desc, dvpp_pic_desc, vdec_frame_config, user_data)**
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。
输入说明	**vdec_channel_desc**：int，指定通道描述信息的指针地址。 与调用**acl.media.dvpp_create_channel**接口创建通道时指定的dvpp_channel_desc保持一致。 调用acl.media.vdec_set_channel_desc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。 **dvpp_stream_desc**：int，输入码流描述信息的指针地址，输入内存用户需提前申请。 ● 调用**acl.media.dvpp_create_stream_desc**接口创建码流描述信息。调用acl.media.dvpp_set_stream_desc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**acl.media.dvpp_set_stream_desc_format**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**10.7.9.1 功能及约束说明**。 **dvpp_pic_desc**：int，输出图片描述信息的指针地址，输出内存用户需提前申请。 dvpp_pic_desc参数作为输入时，需要用户调用如下接口： ● 调用**acl.media.dvpp_create_pic_desc**接口创建图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc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、width_stride、height_stride这4个参数，且当width<16时，width_stride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**10.7.9.1 功能及约束说明**。 dvpp_pic_desc参数作为输出时，用户需在回调函数中调用acl.media.dvpp_get_pic_desc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。 **vdec_frame_config**：int，解码配置的指针地址，预留，当前可填0。 **user_data**：python对象，用户自定义数据的指针地址。 如果用户需要获取解码的帧序号，则可以在user_data参数处定义，然后解码的帧序号可以通过user_data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	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	● 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的空码流包，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**acl.media.vdec_send_frame**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。 ● **acl.media.vdec_send_frame**接口内部封装了**acl.rt.launch_callback**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**acl.rt.launch_callback**接口。
参考资源	接口调用流程及示例，参见**5.3.6 VDEC视频解码**。

10.7.9.4 函数：vdec_send_skipped_frame

C函数原型	**aclError aclvdecSendSkippedFrame(aclvdecChannelDesc *channelDesc, acldvppStreamDesc *input, aclvdecFrameConfig *config, void *userData)**
Python函数	**ret = acl.media.vdec_send_skipped_frame(vdec_channel_desc, dvpp_stream_desc, vdec_frame_config, user_data)**
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dvpp_pic_desc为0。异步接口。

输入说明	**vdec_channel_desc**：int，指定通道描述信息， 与调用**acl.media.dvpp_create_channel**接口创建通道时指定的dvpp_channel_desc保持一致。 调用acl.media.vdec_set_channel_desc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。 **dvpp_stream_desc**：int，输入码流描述信息，输入内存用户需提前申请。 ● 调用**acl.media.dvpp_create_stream_desc**接口创建码流描述信息，调用acl.media.dvpp_set_stream_desc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**acl.media.dvpp_set_stream_desc_format**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**10.7.9.1 功能及约束说明**。 **vdec_frame_config**：int，解码配置，预留，当前可填0。 **user_data**：python对象，用户自定义数据。 如果用户需要获取解码的帧序号，则可以在“user_data”参数处定义，然后解码的帧序号可以通过“user_data”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	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	● 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**acl.media.vdec_send_frame**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。 ● **acl.media.vdec_send_skipped_frame**接口内部封装了**acl.rt.launch_callback**接口，用于在Stream的任务队列中增加一个需要在Host上执行的回调函数。用户在实现VDEC功能时，无需再单独调用**acl.rt.launch_callback**接口。

10.7.9.5 函数：vdec_create_frame_config

C函数原型	**aclvdecFrameConfig *aclvdecCreateFrameConfig()**
Python函数	**vdec_frame_config = acl.media.vdec_create_frame_config()**

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。同步接口。 如需销毁aclvdecFrameConfig类型的数据，请参见**acl.media.vdec_destroy_frame_config**。
返回值说明	**vdec_frame_config**：int，vdec_frame_config的指针地址。 ● 返回非0值，表示成功。 ● 返回0，表示失败。

10.7.9.6 函数：vdec_destroy_frame_config

C函数原型	**aclError aclvdecDestroyFrameConfig(aclvdecFrameConfig *vdecFrameConfig)**
Python函数	**ret = acl.media.vdec_destroy_frame_config(vdec_frame_config)**
函数功能	销毁通过**acl.media.vdec_create_frame_config**接口创建的配置数据。同步接口。
输入说明	**vdec_frame_config**：int，指定待销毁的数据。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。

10.7.9.7 函数：vdec_create_channel_desc

C函数原型	**aclvdecChannelDesc *aclvdecCreateChannelDesc()**
Python函数	**vdec_channel_desc = acl.media.vdec_create_channel_desc()**
函数功能	创建vdec_channel_desc类型的数据，表示创建视频数据解码处理通道时的通道描述信息。该接口为同步接口。 如需销毁aclvdecChannelDesc类型的数据，请参见**acl.media.vdec_destroy_channel_desc**。
返回值说明	**vdec_channel_desc**：int，vdec通道描述地址对象。 ● 返回vdec_channel_desc类型数据，表示成功。 ● 返回0，表示失败。

10.7.9.8 函数：vdec_destroy_channel_desc

C函数原型	**aclError aclvdecDestroyChannelDesc(aclvdecChannelDesc *channelDesc)**
Python函数	**ret = acl.media.vdec_destroy_channel_desc(vdec_channel_desc)**
函数功能	销毁通过**acl.media.vdec_create_channel_desc**接口创建的aclvdecChannelDesc类型的数据。同步接口。
输入说明	**vdec_channel_desc**：int，vdec通道描述地址对象, 指定需要销毁的数据。
返回值说明	**ret**：int ，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。

10.7.9.9 函数：vdec_set_channel_desc_channel_id

C函数原型	**aclError aclvdecSetChannelDescChannelId(aclvdecChannelDesc *channelDesc, uint32_t channelId)**
Python函数	**ret = acl.media.vdec_set_channel_desc_channel_id(vdec_channel_desc, channel_id)**
函数功能	设置视频解码处理通道描述信息的属性：解码通道号。
输入说明	**vdec_channel_desc**：int，vdec通道描述地址对象, 表示视频数据解码处理通道的描述信息。 **channel_id**：int，通道ID。 该参数值的取值范围[0, 255]。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。

10.7.9.10 函数：vdec_set_channel_desc_thread_id

C函数原型	**aclError aclvdecSetChannelDescThreadId(aclvdecChannelDesc *channelDesc, uint64_t threadId)**
Python函数	**ret = acl.media.vdec_set_channel_desc_thread_id(vdec_channel_desc, thread_id)**
函数功能	设置视频解码处理通道描述信息的属性：回调线程ID。

输入说明	**vdec_channel_desc**：int，vdec通道描述地址对象, 表示视频数据解码处理通道的描述信息。 **thread_id**：int，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。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.9.11 函数：vdec_set_channel_desc_callback

C函数原型	**aclError aclvdecSetChannelDescCallback(aclvdecChannelDesc *channelDesc, aclvdecCallback callback)**
Python函数	**ret = acl.media.vdec_set_channel_desc_callback(vdec_channel_desc, callback)**
函数功能	设置视频解码处理通道描述信息的属性：回调函数。

输入说明	**vdec_channel_desc**：int，vdec通道描述地址对象, 表示视频数据解码处理通道的描述信息。
	callback：python函数对象，回调函数。
	Python侧格式如下：
	`def vdec_call_back_func(dvpp_stream_desc, dvpp_pic_desc, user_data):`
	`pass`

返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.9.12 函数：vdec_set_channel_desc_entype

C函数原型	**aclError aclvdecSetChannelDescEnType(aclvdecChannelDesc *channelDesc, acldvppStreamFormat enType)**
Python函数	**ret = acl.media.vdec_set_channel_desc_entype(vdec_channel_desc, entype)**
函数功能	设置视频解码处理通道描述信息的属性：视频编码协议。

输入说明	**vdec_channel_desc**：int，vdec通道描述地址对象, 表示视频数据解码处理通道的描述信息。 **entype**：int，视频编码协议。 ● 0：H265_MAIN_LEVEL ● 1：H264_BASELINE_LEVEL ● 2：H264_MAIN_LEVEL ● 3：H264_HIGH_LEVEL
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.9.13 函数：vdec_set_channel_desc_out_pic_format

C函数原型	**aclError aclvdecSetChannelDescOutPicFormat(aclvdecChannelDesc *channelDesc, acldvppPixelFormat outPicFormat)**
Python函数	**ret = acl.media.vdec_set_channel_desc_out_pic_format(vdec_channel_desc, out_pic_format)**
函数功能	设置视频解码处理通道描述信息的属性：YUV图像存储格式。
输入说明	**vdec_channel_desc**：int，vdec通道描述地址对象, 表示视频数据解码处理通道的描述信息。 **out_pic_format**：int，YUV图像存储格式。 支持如下格式： ● YUV420SP NV12 ● YUV420SP NV21 ● RGB888 ● BGR888 如果不设置输出格式，默认使用YUV420SP NV12。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.9.14 函数：vdec_set_channel_desc_out_pic_width

C函数原型	**aclError aclvdecSetChannelDescOutPicWidth(aclvdecChannelDesc *channelDesc, uint32_t outPicWidth)**

Python函数	**ret = acl.media.vdec_set_channel_desc_out_pic_width(vdec_channel_desc, out_pic_width)**
函数功能	设置视频解码处理通道描述信息的属性：图片宽度。
输入说明	**vdec_channel_desc**：int，vdec通道描述地址对象, 表示视频数据解码处理通道的描述信息。 **out_pic_width**：int，解码码流最大宽度。 如果不设置解码码流最大宽度，内部默认使用4096。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.9.15 函数：vdec_set_channel_desc_out_pic_height

C函数原型	**aclError aclvdecSetChannelDescOutPicHeight(aclvdecChannelDesc *channelDesc, uint32_t outPicHeight)**
Python函数	**ret = acl.media.vdec_set_channel_desc_out_pic_height(vdec_channel_desc, out_pic_height)**
函数功能	设置视频解码处理通道描述信息的属性：图片高度。
输入说明	**vdec_channel_desc**：int，vdec通道描述地址对象, 表示视频数据解码处理通道的描述信息。 **out_pic_height**：int，解码码流最大高度。 如果不设置解码码流最大高度，内部默认使用4096。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.9.16 函数：vdec_set_channel_desc_ref_frame_num

C函数原型	**aclError aclvdecSetChannelDescRefFrameNum(aclvdecChannelDesc *channelDesc, uint32_t refFrameNum)**
Python函数	**ret = acl.media.vdec_set_channel_desc_ref_frame_num(vdec_channel_desc, ref_frame_num)**
函数功能	设置视频解码处理通道描述信息的属性：参考帧数量。

输入说明	**vdec_channel_desc**：int，vdec通道描述地址对象, 表示视频数据解码处理通道的描述信息。 **ref_frame_num**：int，参考帧数量，取值范围[0, 16]。 如果不设置参考帧数量，内部默认使用8。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.9.17 函数：vdec_set_channel_desc_out_mode

C函数原型	**aclError aclvdecSetChannelDescOutMode(aclvdecChannelDesc *channelDesc, uint32_t outMode)**
Python函数	**ret = acl.media.vdec_set_channel_desc_out_mode(vdec_channel_desc, out_mode)**
函数功能	设置视频解码处理通道描述信息的属性：是否实时出帧。
输入说明	**vdec_channel_desc**：int，vdec通道描述地址对象, 表示视频数据解码处理通道的描述信息。 **out_mode**：int，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）。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.9.18 函数：vdec_set_channel_desc_bit_depth

C函数原型	**aclError aclvdecSetChannelDescBitDepth(aclvdecChannelDesc *channelDesc, uint32_t bitDepth)**
Python函数	**ret = acl.media.vdec_set_channel_desc_bit_depth(vdec_channel_desc, desc_bit_depth)**
函数功能	设置视频解码处理通道描述信息的属性。

输入说明	**vdec_channel_desc**：int，表示视频解码处理通道的描述信息。 **desc_bit_depth**：int，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）
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.9.19 函数：vdec_get_channel_desc_channel_id

C函数原型	**uint32_t aclvdecGetChannelDescChannelId(const aclvdecChannelDesc *channelDesc)**
Python函数	**channel_id = acl.media.vdec_get_channel_desc_channel_id(vdec_channel_desc)**
函数功能	获取视频解码处理通道描述信息的属性：解码通道号。
输入说明	**vdec_channel_desc**：int，vdec通道描述地址对象, 表示视频数据解码处理通道的描述信息。
返回值说明	**channel_id**：int，解码通道号。 该参数值的取值范围[0, 255]。

10.7.9.20 函数：vdec_get_channel_desc_thread_id

C函数原型	**uint32_t aclvdecGetChannelDescThreadId(const aclvdecChannelDesc *channelDesc);**
Python函数	**thread_id = acl.media.vdec_get_channel_desc_thread_id(vdec_channel_desc)**
函数功能	获取视频解码处理通道描述信息的属性：回调线程ID。
输入说明	**vdec_channel_desc**：int，vdec通道描述地址对象, 表示视频数据解码处理通道的描述信息。
返回值说明	**thread_id**：int，回调线程ID。

10.7.9.21 函数：vdec_get_channel_desc_callback

C函数原型	**aclvdecCallback aclvdecGetChannelDescCallback(const aclvdecChannelDesc *channelDesc)**
Python函数	**callback = acl.media.vdec_get_channel_desc_callback(vdec_channel_desc)**
函数功能	获取视频解码处理通道描述信息的属性：回调函数。
输入说明	**vdec_channel_desc**：int，vdec通道描述地址对象, 表示视频数据解码处理通道的描述信息。
返回值说明	**callback**：python函数对象，回调函数。 Python侧格式如下： def vdec_call_back_func(dvpp_stream_desc, dvpp_pic_desc, user_data): pass

10.7.9.22 函数：vdec_get_channel_desc_entype

C函数原型	**acldvppStreamFormat aclvdecGetChannelDescEnType(const aclvdecChannelDesc *channelDesc);**
Python函数	**entype = acl.media.vdec_get_channel_desc_entype(vdec_channel_desc)**
函数功能	获取视频解码处理通道描述信息的属性：视频编码协议。
输入说明	**vdec_channel_desc**：int，vdec通道描述地址对象, 表示视频数据解码处理通道的描述信息。
返回值说明	**entype**：int，视频编码协议。 ● H265_MAIN_LEVEL = 0 ● H264_BASELINE_LEVEL = 1 ● H264_MAIN_LEVEL = 2 ● H264_HIGH_LEVEL = 3

10.7.9.23 函数：vdec_get_channel_desc_out_pic_format

C函数原型	**acldvppPixelFormat aclvdecGetChannelDescOutPicFormat(const aclvdecChannelDesc *channelDesc);**
Python函数	**out_pic_format = acl.media.vdec_get_channel_desc_out_pic_format(vdec_channel_desc)**
函数功能	获取视频解码处理通道描述信息的属性：YUV图像存储格式。

输入说明	**vdec_channel_desc**：int，vdec通道描述地址对象, 表示视频数据解码处理通道的描述信息。
返回值说明	**out_pic_format**：int，YUV图像存储格式。具体数值请参考**acl.media.dvpp_set_pic_desc_format**接口。

10.7.9.24 函数：vdec_get_channel_desc_out_pic_width

C函数原型	**uint32_t aclvdecGetChannelDescOutPicWidth(const aclvdecChannelDesc *channelDesc);**
Python函数	**out_pic_width = acl.media.vdec_get_channel_desc_out_pic_width(vdec_channel_desc)**
函数功能	获取视频解码处理通道描述信息的属性：图片宽度。
输入说明	**vdec_channel_desc**：int，vdec通道描述地址对象, 表示视频数据解码处理通道的描述信息。
返回值说明	**out_pic_width**：int，图片宽度。

10.7.9.25 函数：vdec_get_channel_desc_out_pic_height

C函数原型	**uint32_t aclvdecGetChannelDescOutPicHeight(const aclvdecChannelDesc *channelDesc);**
Python函数	**out_pic_height = acl.media.vdec_get_channel_desc_out_pic_height(vdec_channel_desc)**
函数功能	获取视频解码处理通道描述信息的属性：图片高度。
输入说明	**vdec_channel_desc**：int，vdec通道描述地址对象, 表示视频数据解码处理通道的描述信息。
返回值说明	**out_pic_height**：int，图片高度。

10.7.9.26 函数：vdec_get_channel_desc_ref_frame_num

C函数原型	**uint32_t aclvdecGetChannelDescRefFrameNum(const aclvdecChannelDesc *channelDesc)**
Python函数	**ref_frame_num = acl.media.vdec_get_channel_desc_ref_frame_num(vdec_channel_desc)**
函数功能	获取视频解码处理通道描述信息的属性：参考帧数量。

输入说明	**vdec_channel_desc**：int，vdec通道描述地址对象, 表示视频数据解码处理通道的描述信息。
返回值说明	**ref_frame_num**：int，参考帧数量，取值范围[0, 16]。

10.7.9.27 函数：vdec_get_channel_desc_out_mode

C函数原型	**uint32_t aclvdecGetChannelDescOutMode(const aclvdecChannelDesc *channelDesc)**
Python函数	**out_mode = acl.media.vdec_get_channel_desc_out_mode(vdec_channel_desc)**
函数功能	获取视频解码处理通道的描述信息：是否实时出帧。
输入说明	**vdec_channel_desc**：int，vdec通道描述地址对象, 表示视频数据解码处理通道的描述信息。
返回值说明	**out_mode**：int，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）。

10.7.9.28 函数：vdec_get_channel_desc_bit_depth

C函数原型	**uint32_t aclvdecGetChannelDescBitDepth(const aclvdecChannelDesc *channelDesc)**
Python函数	**desc_bit_depth = acl.media.vdec_get_channel_desc_bit_depth(vdec_channel_desc)**
函数功能	获取视频解码处理通道的描述信息。
输入说明	**vdec_channel_desc**：int，表示视频解码数据处理通道的描述信息。
返回值说明	**desc_bit_depth**：int，视频位宽。 取值范围如下： ● 0：8-bit ● 1: 10-bit

10.7.10 VENC 功能

10.7.10.1 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。

不支持单进程多线程场景。

分辨率约束

- 输入图片分辨率：

昇腾AI处理器	分辨率范围
Atlas 200/500 A2推理产品	最大分辨率8192*8192，最小分辨率114*114。

- 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**acl.media.dvpp_malloc**接口申请Device上的输入内存，调用**acl.media.dvpp_free**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**10.7.2 基本概念**。

表 10-13 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stride * 3/2	• H264 BP/MP/HP • H265 MP（仅支持Slice码流）
YUV420SP NV21 8bit			

10.7.10.2 性能指标说明

性能指标说明

单个Device的基本场景性能指标参考如下（1路对应一个通道，一个通道对应一个线程）：

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。

场景举例	总帧率
720p*n路(1≤n≤3)	n*400fps
720p*n路(n>3)	1200fps
1080p*n路(1≤n≤3)	n*200fps
1080p*n路(n>3)	600fps
4k*n路(1≤n≤3)	n*50fps
4k*n路(n>3)	150fps

注：其它分辨率可以等量估算。

10.7.10.3 函数：venc_send_frame

C函数原型	**aclError aclvencSendFrame(aclvencChannelDesc *channelDesc, acldvppPicDesc *input, void *reserve, aclvencFrameConfig *config, void *userdata)**
Python函数	**ret = acl.media.venc_send_frame(venc_channel_desc, input, reserve, config, userdata)**
函数功能	将待编码的图片传到编码器进行编码。异步接口。

输入说明	**venc_channel_desc**：int，指定通道描述。 与调用**acl.media.venc_create_channel**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**：int，输入图片描述信息，输入内存用户需提前申请。 ● 调用**acl.media.dvpp_create_pic_desc**接口创建图片描述信息，调用acl.media.dvpp_set_pic_desc系列接口设置图片描述参数（例如，图片格式、宽、高等）。 输入图片格式以通道描述信息中设置的格式为准，调用**acl.media.dvpp_set_pic_desc_format**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**10.7.10.1 功能及约束说明**。 **reserve**：int，预留参数，暂不使用，设值为0。 **config**：int，单帧配置数据。 **userdata**：Python对象，用户自定义数据。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	● 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。

10.7.10.4 函数：venc_create_channel_desc

C函数原型	**aclvencChannelDesc *aclvencCreateChannelDesc()**
Python函数	**venc_channel_desc = acl.media.venc_create_channel_desc()**
函数功能	创建aclvencChannelDesc类型的数据，表示创建视频数据编码处理通道时的通道描述信息。同步接口。 如需销毁aclvencChannelDesc类型的数据，请参见**acl.media.venc_destroy_channel_desc**。
返回值说明	**venc_channel_desc**：int，venc_channel_desc的指针地址。 ● 返回非0值，表示成功。 ● 返回0，表示失败。

10.7.10.5 函数：venc_destroy_channel_desc

C函数原型	**aclError aclvencDestroyChannelDesc(aclvencChannelDesc *channelDesc)**
Python函数	**ret = acl.media.venc_destroy_channel_desc(venc_channel_desc)**
函数功能	销毁通过**acl.media.venc_create_channel_desc**接口创建的aclvencChannelDesc类型的数据。同步接口。
输入说明	**venc_channel_desc**：int，指定需要销毁的venc_channel_desc。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。

10.7.10.6 函数：venc_set_channel_desc_thread_id

C函数原型	**aclError aclvencSetChannelDescThreadId(aclvencChannelDesc *channelDesc, uint64_t threadId)**
Python函数	**ret = acl.media.venc_set_channel_desc_thread_id(venc_channel_desc, thread_id)**
函数功能	设置视频编码处理通道描述信息的属性：回调线程ID。
输入说明	**venc_channel_desc**：int，表示视频数据编码处理通道的描述信息。 **thread_id**：int，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。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。

10.7.10.7 函数：venc_set_channel_desc_callback

C函数原型	**aclError aclvencSetChannelDescCallback(aclvencChannelDesc *channelDesc, aclvencCallback callback)**
Python函数	**ret = acl.media.venc_set_channel_desc_callback(venc_channel_desc, callback)**
函数功能	设置视频编码处理通道描述信息的属性：回调函数。

输入说明	**venc_channel_desc**：int，表示视频数据编码处理通道的描述信息。 **callback**：python函数对象，编码回调函数。 Python侧格式如下： def venc_call_back_func(dvpp_pic_desc, dvpp_stream_desc, user_data): pass
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.10.8 函数：venc_set_channel_desc_entype

C函数原型	**aclError aclvencSetChannelDescEnType(aclvencChannelDesc *channelDesc, acldvppStreamFormat enType);**
Python函数	**ret = acl.media.venc_set_channel_desc_entype(venc_channel_desc, venc_entype)**
函数功能	设置视频编码处理通道描述信息的属性：视频编码协议。
输入说明	**venc_channel_desc**：int，表示视频数据编码处理通道的描述信息。 **venc_entype**：int，视频编码协议。 ● H265-main level（0） ● H264-baseline level（1） ● H264-main level（2） ● H264-high level（3）
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.10.9 函数：venc_set_channel_desc_pic_format

C函数原型	**aclError aclvencSetChannelDescPicFormat(aclvencChannelDesc *channelDesc, acldvppPixelFormat picFormat);**
Python函数	**ret = acl.media.venc_set_channel_desc_pic_format(venc_channel_desc, venc_format)**
函数功能	设置视频编码处理通道描述信息的属性：输入图像格式。

输入说明	**venc_channel_desc**：int，表示视频数据编码处理通道的描述信息。 **venc_format**：int，输入图像格式，支持如下格式： ● PIXEL_FORMAT_YUV_SEMIPLANAR_420 ● PIXEL_FORMAT_YVU_SEMIPLANAR_420
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.10.10 函数：venc_set_channel_desc_pic_width

C函数原型	**aclError aclvencSetChannelDescPicWidth(aclvencChannelDesc *channelDesc, uint32_t picWidth);**
Python函数	**ret = acl.media.venc_set_channel_desc_pic_width(venc_channel_desc, in_pic_width)**
函数功能	设置视频编码处理通道描述信息的属性：图像宽度。
输入说明	**venc_channel_desc**：int，表示视频数据编码处理通道的描述信息。 **in_pic_width**：int，图片宽度。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.10.11 函数：venc_set_channel_desc_pic_height

C函数原型	**aclError aclvencSetChannelDescPicHeight(aclvencChannelDesc *channelDesc, uint32_t picHeight);**
Python函数	**ret = acl.media.venc_set_channel_desc_pic_height(venc_channel_desc, in_pic_height)**
函数功能	设置视频编码处理通道描述信息的属性：图像高度。
输入说明	**venc_channel_desc**：int，表示视频数据编码处理通道的描述信息。 **in_pic_height**：int，图片高度。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.10.12 函数：venc_set_channel_desc_key_frame_interval

C函数原型	**aclError aclvencSetChannelDescKeyFrameInterval(aclvencChannelDesc *channelDesc, uint32_t keyFrameInterval);**
Python函数	**ret = acl.media.venc_set_channel_desc_key_frame_interval(venc_channel_desc, key_frame_interval)**
函数功能	设置视频编码处理通道描述信息的属性：关键帧间隔。
输入说明	**venc_channel_desc**：int，表示视频数据编码处理通道的描述信息。 **key_frame_interval**：int，关键帧间隔，取值范围[1, 65536]。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.10.13 函数：venc_set_channel_desc_buf_addr

C函数原型	**aclError aclvencSetChannelDescBufAddr(aclvencChannelDesc *channelDesc, void *bufAddr)**
Python函数	**ret = acl.media.venc_set_channel_desc_buf_addr(venc_channel_desc, buf_addr)**
函数功能	设置视频编码处理通道描述信息的属性: 编码输出缓存地址。
输入说明	**venc_channel_desc**：int，表示视频数据编码处理通道的描述信息。 **buf_addr**：int，编码输出缓存地址。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.10.14 函数：venc_set_channel_desc_buf_size

C函数原型	**aclError aclvencSetChannelDescBufSize(aclvencChannelDesc *channelDesc, uint32_t bufSize)**
Python函数	**ret = acl.media.venc_set_channel_desc_buf_size(venc_channel_desc, buf_size)**
函数功能	设置视频编码处理通道描述信息的属性: 编码输出缓存大小。

输入说明	**venc_channel_desc**：int，表示视频数据编码处理通道的描述信息。 **buf_size**：int，编码输出缓存大小，单位为Byte。 **说明** 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。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.10.15 函数：venc_set_channel_desc_rc_mode

C函数原型	**aclError aclvencSetChannelDescRcMode(aclvencChannelDesc *channelDesc, uint32_t rcMode)**
Python函数	**ret = acl.media.venc_set_channel_desc_rc_mode(venc_channel_desc, rc_mode)**
函数功能	设置视频编码处理通道描述信息的属性。
输入说明	**venc_channel_desc**：int，表示视频编码处理通道的描述信息。 **rc_mode**：int，指定码率控制模式，如果不设置该参数，默认为CBR模式。0表示使用默认值，即CBR模式。 ● 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。 默认值0表示VBR模式。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.7.10.16 函数：venc_set_channel_desc_src_rate

C函数原型	**aclError aclvencSetChannelDescSrcRate(aclvencChannelDesc *channelDesc, uint32_t srcRate)**
Python函数	**ret = acl.media.venc_set_channel_desc_src_rate(venc_channel_desc, src_rate)**
函数功能	设置视频编码处理通道描述信息的属性。

输入说明	**venc_channel_desc**：int，表示视频编码处理通道的描述信息。 **src_rate**：int，输入码流帧率，单位fps。 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。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.7.10.17 函数：venc_set_channel_desc_max_bit_rate

C函数原型	**aclError aclvencSetChannelDescMaxBitRate(aclvencChannelDesc *channelDesc, uint32_t maxBitRate)**
Python函数	**ret = acl.media.venc_set_channel_desc_max_bit_rate(venc_channel_desc, max_bit_rate)**
函数功能	设置视频编码处理通道描述信息的属性。
输入说明	**venc_channel_desc**：int，表示视频编码处理通道的描述信息。 **max_bit_rate**：int，输出码率，单位kbps。 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	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.7.10.18 函数：venc_get_channel_desc_channel_id

C函数原型	**uint32_t aclvencGetChannelDescChannelId(const aclvencChannelDesc *channelDesc);**
Python函数	**channel_id = acl.media.venc_get_channel_desc_channel_id(venc_channel_desc)**

函数功能	获取视频编码处理通道的描述信息：通道号。
输入说明	**venc_channel_desc**：int，表示视频数据编码处理通道的描述信息。
返回值说明	**channel_id**：int，通道号，默认为0。

10.7.10.19 函数：venc_get_channel_desc_thread_id

C函数原型	**uint64_t aclvencGetChannelDescThreadId(const aclvencChannelDesc *channelDesc);**
Python函数	**thread_id = acl.media.venc_get_channel_desc_thread_id(venc_channel_desc)**
函数功能	获取视频编码处理通道的描述信息：回调线程ID。
输入说明	**venc_channel_desc**：int，表示视频数据编码处理通道的描述信息。
返回值说明	**thread_id**：int，回调线程ID。

10.7.10.20 函数：venc_get_channel_desc_callback

C函数原型	**aclvencCallback aclvencGetChannelDescCallback(const aclvencChannelDesc *channelDesc);**
Python函数	**callback = acl.media.venc_get_channel_desc_callback(venc_channel_desc)**
函数功能	获取视频编码处理通道的描述信息：编码回调函数。
输入说明	**venc_channel_desc**：int，表示视频数据编码处理通道的描述信息。

返回值说明	**callback**：python函数对象，编码回调函数。
	Python侧格式如下：
	`def venc_call_back_func(dvpp_pic_desc, dvpp_stream_desc, user_data):`
	`pass`

10.7.10.21 函数：venc_get_channel_desc_entype

C函数原型	**acldvppStreamFormat aclvencGetChannelDescEnType(const aclvencChannelDesc *channelDesc);**
Python函数	**entype = acl.media.venc_get_channel_desc_entype(venc_channel_desc)**
函数功能	获取视频编码处理通道的描述信息：视频编码协议。
输入说明	**venc_channel_desc**：int，表示视频数据编码处理通道的描述信息。

返回值说明	**entype**：int，视频编码协议。 ● H265-main level（0） ● H264-baseline level（1） ● H264-main level（2） ● H264-high level（3）

10.7.10.22 函数：venc_get_channel_desc_pic_format

C函数原型	**acldvppPixelFormat aclvencGetChannelDescPicFormat(const aclvencChannelDesc *channelDesc);**
Python函数	**in_pic_format = acl.media.venc_get_channel_desc_pic_format(venc_channel_desc)**
函数功能	获取视频编码处理通道的描述信息：输入图像格式。
输入说明	**venc_channel_desc**：int，表示视频数据编码处理通道的描述信息。
返回值说明	**in_pic_format**：int，输入图像格式。

10.7.10.23 函数：venc_get_channel_desc_pic_width

C函数原型	**uint32_t aclvencGetChannelDescPicWidth(const aclvencChannelDesc *channelDesc);**
Python函数	**in_pic_width = acl.media.venc_get_channel_desc_pic_width(venc_channel_desc)**
函数功能	获取视频编码处理通道的描述信息：图片宽度。
输入说明	**venc_channel_desc**：int，表示视频数据编码处理通道的描述信息。
返回值说明	**in_pic_width**：int，图片宽度。

10.7.10.24 函数：venc_get_channel_desc_pic_height

C函数原型	**uint32_t aclvencGetChannelDescPicHeight(const aclvencChannelDesc *channelDesc);**
Python函数	**in_pic_height = acl.media.venc_get_channel_desc_pic_height(venc_channel_desc)**
函数功能	获取视频编码处理通道的描述信息：图片高度。
输入说明	**venc_channel_desc**：int，表示视频数据编码处理通道的描述信息。

返回值说明	**in_pic_height**：int，图片高度。

10.7.10.25 函数：venc_get_channel_desc_key_frame_interval

C函数原型	**uint32_t aclvencGetChannelDescKeyFrameInterval(const aclvencChannelDesc *channelDesc)**
Python函数	**key_frame_interval = acl.media.venc_get_channel_desc_key_frame_interval(venc_channel_desc)**
函数功能	获取视频编码处理通道的描述信息：关键帧间隔。
输入说明	**venc_channel_desc**：int，表示视频数据编码处理通道的描述信息。
返回值说明	**key_frame_interval**：int，关键帧间隔，不能为0。

10.7.10.26 函数：venc_get_channel_desc_buf_addr

C函数原型	**void *aclvencGetChannelDescBufAddr(const aclvencChannelDesc *channelDesc)**
Python函数	**buf_addr = acl.media.venc_get_channel_desc_buf_addr(venc_channel_desc)**
函数功能	获取视频编码处理通道描述信息的属性: 编码输出缓存地址。
输入说明	**venc_channel_desc**：int，表示视频数据编码处理通道的描述信息。
返回值说明	**buf_addr**：int，编码输出缓存地址。

10.7.10.27 函数：venc_get_channel_desc_buf_size

C函数原型	**uint32_t aclvencGetChannelDescBufSize(const aclvencChannelDesc *channelDesc)**
Python函数	**buf_size = acl.media.venc_get_channel_desc_buf_size(venc_channel_desc)**
函数功能	获取视频编码处理通道描述信息的属性：编码输出缓存大小。
输入说明	**venc_channel_desc**：int，表示视频数据编码处理通道的描述信息。
返回值说明	**buf_size**：int，编码输出缓存大小，单位为Byte。

10.7.10.28 函数：venc_get_channel_desc_rc_mode

C函数原型	**uint32_t aclvencGetChannelDescRcMode(const aclvencChannelDesc *channelDesc)**
Python函数	**rc_mode = acl.media.venc_get_channel_desc_rc_mode(venc_channel_desc)**
函数功能	获取视频编码处理通道的描述信息。
输入说明	**venc_channel_desc**：int，表示视频编码处理通道的描述信息。
返回值说明	**rc_mode**：int，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。

10.7.10.29 函数：venc_get_channel_desc_src_rate

C函数原型	**uint32_t aclvencGetChannelDescSrcRate(const aclvencChannelDesc *channelDesc)**
Python函数	**src_rate = acl.media.venc_get_channel_desc_src_rate(venc_channel_desc)**
函数功能	获取视频编码处理通道的描述信息。
输入说明	**venc_channel_desc**：int，表示视频编码处理通道的描述信息。
返回值说明	**src_rate** ：int，输入码流帧率，单位fps。

10.7.10.30 函数：venc_get_channel_desc_max_bit_rate

C函数原型	**uint32_t aclvencGetChannelDescMaxBitRate(const aclvencChannelDesc *channelDesc)**
Python函数	**max_bit_rate = acl.media.venc_get_channel_desc_max_bit_rate(venc_channel_desc)**
函数功能	获取视频编码处理通道的描述信息。
输入说明	**venc_channel_desc**：int，表示视频编码处理通道的描述信息。
返回值说明	**max_bit_rate**：int，输出码率，单位kbps。

10.7.10.31 函数：venc_set_channel_desc_param

C函数原型	**aclError aclvencSetChannelDescParam(aclvencChannelDesc *channelDesc, aclvencChannelDescParamType paramType, size_t length, const void *param)**
Python函数	**ret = acl.media.venc_set_channel_desc_param(venc_channel_desc, param_type, param)**
函数功能	设置视频编码处理通道的属性参数接口。同步接口。
输入说明	**venc_channel_desc**：int，表示视频编码处理通道的描述信息。 **param_type**：int，属性参数的类型，参考aclvencChannelDescParamType。 **param**：属性参数值。 ● 属性参数的类型为uint64_t或uint32_t时，此处为对应的参数值。 ● 属性参数的类型为回调函数时，此处为python函数对象，编码回调函数。 python侧格式如下： def venc_call_back_func(dvpp_pic_desc, dvpp_stream_desc, user_data): pass ● 属性参数的类型为指针时，此处为指针指向的地址。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.10.32 函数：venc_get_channel_desc_param

C函数原型	**aclError aclvencGetChannelDescParam(const aclvencChannelDesc *channelDesc, aclvencChannelDescParamType paramType, size_t length, size_t *paramRetSize, void *param)**
Python函数	**attr, ret = acl.media.venc_get_channel_desc_param(venc_channel_desc, param_type)**
函数功能	获取视频编码处理通道的描述信息。同步接口。
输入说明	**venc_channel_desc**：int，表示视频编码处理通道的描述信息。 **param_type**：int，属性参数的类型，参考aclvencChannelDescParamType。

返回值说明	**attr**：属性值
	● 属性参数的类型为uint64_t或uint32_t时，此处为对应的属性值。
	● 属性参数的类型为回调函数时，此处为python函数对象，编码回调函数。
	python侧格数如下：
	`def venc_call_back_func(dvpp_pic_desc, dvpp_stream_desc, user_data):`
	`pass`
	● 属性参数的类型为指针时，此处为指针指向的地址。
	ret：int，错误码。
	● 返回0表示成功。
	● 返回其它值表示失败。

10.7.10.33 函数：venc_create_frame_config

C函数原型	**aclvencFrameConfig *aclvencCreateFrameConfig()**
Python函数	**frame_config = acl.media.venc_create_frame_config()**
函数功能	创建aclvencFrameConfig类型的数据，创建VENC编码时的单帧编码配置参数。同步接口。
返回值说明	**frame_config**：int。 ● 返回非0值，表示成功。 ● 返回0，表示失败。

10.7.10.34 函数：venc_destroy_frame_config

C函数原型	**aclError aclvencDestroyFrameConfig(aclvencFrameConfig *config)**
Python函数	**ret = acl.media.venc_destroy_frame_config(frame_config)**
函数功能	销毁通过acl.media.venc_create_frame_config接口创建的aclvencFrameConfig类型的数据。同步接口。
输入说明	**frame_config**：int，指定需要销毁的config。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.10.35 函数：venc_set_frame_config_force_i_frame

C函数原型	**aclError aclvencSetFrameConfigForceIFrame(aclvencFrameConfig *config, uint8_t forceIFrame);**
Python函数	**ret = acl.media.venc_set_frame_config_force_i_frame(config, force_i_frame)**
函数功能	设置单帧编码配置参数。同步接口。
输入说明	**config**：int，表示单帧编码配置数据。 **force_i_frame**：int，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。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.10.36 函数：venc_set_frame_config_eos

C函数原型	**aclError aclvencSetFrameConfigEos(aclvencFrameConfig *config, uint8_t eos)**
Python函数	**ret = acl.media.venc_set_frame_config_eos(config, eos)**
函数功能	设置单帧编码配置参数。同步接口。
输入说明	**config**：int，表示单帧编码配置数据。 **eos**：uint8_t，是否为结束帧。 ● 0：不是结束帧。 ● 1：是结束帧。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.10.37 函数：venc_get_frame_config_force_i_frame

C函数原型	**uint8_t aclvencGetFrameConfigForceIFrame(const aclvencFrameConfig *config);**
Python函数	**force_i_frame = acl.media.venc_get_frame_config_force_i_frame(config)**
函数功能	获取单帧编码配置参数。同步接口。

输入说明	**config**：int，表示单帧编码配置数据。
返回值说明	**force_i_frame**：int，是否强制重新开始I帧间隔。 ● 0：不强制重新开始。 ● 1：强制重新开始I帧。

10.7.10.38 函数：venc_get_frame_config_eos

C函数原型	**uint8_t aclvencGetFrameConfigEos(const aclvencFrameConfig *config)**
Python函数	**eos = acl.media.venc_get_frame_config_eos(config)**
函数功能	获取单帧编码配置参数。
输入说明	**config**：int，表示单帧编码配置数据。
返回值说明	**eos**：uint8_t，是否为结束帧。 ● 0：不是结束帧。 ● 1：是结束帧。

10.7.11 PNGD 功能

10.7.11.1 功能及约束说明

功能说明

PNGD（PNG Decoder）功能：实现PNG格式图片的硬件解码。

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**acl.media.dvpp_malloc**接口申请Device上的输入、输出内存，调用**acl.media.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。

- 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**acl.media.dvpp_png_predict_dec_size**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**10.7.2 基本概念**。

表 10-14 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**acl.media.dvpp_png_predict_dec_size**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**acl.media.dvpp_png_predict_dec_size**接口预估。
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**acl.media.dvpp_png_predict_dec_size**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**acl.media.dvpp_png_predict_dec_size**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
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acl.media.dvpp_png_predict_dec_size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acl.media.dvpp_png_predict_dec_size接口预估。

10.7.11.2 性能指标说明

性能指标说明

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acl.rt.synchronize_stream接口。

1080p指分辨率为1920*1080的图片；4K指分辨率为3840*2160的图片。单个Device的基本场景性能指标参考如下（1路对应一个通道，一个通道对应一个线程）：

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

10.7.11.3 函数：dvpp_png_decode_async

C函数原型	**aclError acldvppPngDecodeAsync(acldvppChannelDesc *channelDesc, const void *data, uint32_t size, acldvppPicDesc *outputDesc, aclrtStream stream)**
Python函数	**ret = acl.media.dvpp_png_decode_async(dvpp_channel_desc, data, size, output_desc, stream)**

函数功能	解码png图片，异步接口。
输入说明	**dvpp_channel_desc**：int，指定通道描述信息。与调用**acl.media.dvpp_create_channel**接口创建通道时指定的dvpp_channel_desc保持一致。 **data**：int，输入图片的内存地址。 **size**：int，输入图片的实际数据大小，单位Byte。 **output_desc**：int，输出图片信息。 参数作为输入时，需要用户调用如下接口： ● 调用**acl.media.dvpp_create_pic_desc**接口创建图片描述信息。 ● 调用acl.media.dvpp_set_pic_desc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**acl.media.dvpp_png_predict_dec_size**接口获取。 – png原图信息（例如宽、高）可通过**acl.media.dvpp_png_get_image_info**接口获取。 – 输出图片格式、内存等要求，请参见**10.7.11.1 功能及约束说明**。 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。 **stream**：int，指定Stream。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败

表 10-15 支持格式

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
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

10.7.11.4 函数：dvpp_png_get_image_info

C函数原型	**aclError acldvppPngGetImageInfo(const void *data, uint32_t dataSize, uint32_t *width, uint32_t *height, int32_t *components)**
Python函数	**width, height, components, ret = acl.media.dvpp_png_get_image_info(data, data_size)**
函数功能	从存放png图片数据的内存中读取png图片的宽、高。
输入说明	**data**：int，存放png图片数据的内存的地址。 **data_size**：int，内存大小，单位为Byte。
返回值说明	**width**：int，图片的宽。 **height**：int，图片的高。 **components**：int，颜色通道个数。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.7.11.5 函数：dvpp_png_predict_dec_size

C函数原型	**aclError acldvppPngPredictDecSize(const void *data, uint32_t dataSize, acldvppPixelFormat outputPixelFormat, uint32_t *decSize)**
Python函数	**dec_size ,ret = acl.media.dvpp_png_predict_dec_size(data, data_size, output_pixel_format)**
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。
输入说明	**data**：int，存放png图片数据的内存的地址。 **data_size**：int，内存大小，单位为Byte。 **output_pixel_format**：int，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
返回值说明	**dec_size**：int，png图片解码后所需的输出内存的大小，单位为Byte。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

10.8 媒体数据处理 V2

10.8.1 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，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。

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 200/500 A2 推理产品	√	√	√	√	√	√

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**10.8.3.3 acl.himpi.dvpp_malloc**申请内存、调用**10.8.3.4 acl.himpi.dvpp_free**接口释放内存。
  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**acl.rt.malloc**接口、或**acl.rt.malloc_host**接口、或**acl.rt.malloc_cached**接口申请内存。
- 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**10.8.4.1 VPC功能**、**10.8.5 VDEC视频解码/JPEGD图像解码接口**、**10.8.6 VENC**

视频/JPEGE图像编码接口、10.8.7 PNGD图像处理接口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。

10.8.2 基本概念

表 10-16 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10.8.5.1 JPEGD功能及约束说明。 ● JPEGE功能，请参见10.8.6.1 JPEGE功能及约束说明。 ● PNGD功能，请参见10.8.7.1 功能及约束说明。 ● VDEC功能，请参见10.8.5.3 VDEC功能及约束说明。 ● VENC功能，请参见10.8.6.3 VENC功能及约束说明。
高stride（heightStride）	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10.8.5.1 JPEGD功能及约束说明。 ● JPEGE功能，请参见10.8.6.1 JPEGE功能及约束说明。 ● PNGD功能，请参见10.8.7.1 功能及约束说明。 ● VDEC功能，请参见10.8.5.3 VDEC功能及约束说明。 ● VENC功能，请参见10.8.6.3 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。

概念	描述
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10-5。

10.8.3 公共接口

10.8.3.1 acl.himpi.sys_init

c函数原型	**hi_s32 hi_mpi_sys_init()**
Python函数	**ret = acl.himpi.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。
返回值说明	**ret**：int，错误码。 返回0表示成功。 返回非0 表示失败，参见10.8.9.1 公共返回码。
约束说明	● 需在申请pyACL运行管理资源后调用本接口。pyACL运行管理资源申请流程请参见运行管理资源申请流程。 ● 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acl.himpi.sys_init接口进行初始化。
参考资源	接口调用流程示例，参见5.4 媒体数据处理V2。

10.8.3.2 acl.himpi.sys_exit

c函数原型	**hi_s32 hi_mpi_sys_exit()**
Python函数	**ret = acl.himpi.sys_exit()**
函数功能	用户需要退出编解码、图像处理操作之后，需先调用本接口对媒体数据处理系统底层进行去初始化。
返回值说明	**ret**：int，错误码。 返回0表示成功。 返回非0表示失败，具体请参见10.8.9.1 公共返回码。

约束说明	● 可以反复去初始化，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**acl.himpi.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pyACL运行管理资源前调用本接口。pyACL运行管理资源申请流程请参见**运行管理资源申请流程**。
参考资源	接口调用流程示例，参见**5.4 媒体数据处理V2**。

10.8.3.3 acl.himpi.dvpp_malloc

c函数原型	**hi_s32 hi_mpi_dvpp_malloc(hi_u32 dev_id, hi_void **dev_ptr, hi_u64 size)**
Python函数	**dev_ptr, ret = acl.himpi.dvpp_malloc(dev_id, size)**
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）。
输入说明	● **dev_id**：Device ID，预留参数，该参数配置无效。 ● **size**：申请内存的大小，单位Byte。
返回值说明	**ret**：int，错误码。 返回0表示成功。 返回非0表示失败，参见**10.8.9.1 公共返回码**。 **devPtr**：int，指向“Device上已分配内存的指针”的指针地址。
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**acl.himpi.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。 ● 调用**acl.himpi.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。
参考资源	接口调用流程示例，参见**5.4 媒体数据处理V2**。

10.8.3.4 acl.himpi.dvpp_free

c函数原型	**hi_s32 hi_mpi_dvpp_free(hi_void *dev_ptr)**
Python函数	**ret = acl.himpi.dvpp_free(dev_ptr)**
函数功能	释放Device上的内存。 **acl.himpi.dvpp_free**接口只能释放通过**10.8.3.3 acl.himpi.dvpp_malloc**接口申请的内存。
输入说明	**dev_ptr**：int，待释放内存的指针地址。
返回值说明	**ret**：int 错误码。 ● 返回0表示成功。 ● 返回非0表示失败。参见**10.8.9.1 公共返回码**
参考资源	接口调用流程示例，参见**5.4 媒体数据处理V2**。

10.8.3.5 acl.himpi.dvpp_get_image_info

c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);**
Python函数	**img_info, ret = acl.himpi.dvpp_get_image_info(img_type, stream)**
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**10.8.9.1 公共返回码**。
输入说明	● **img_type**：要解析的码流类型。当前仅支持码流类型为HI_PT_JPEG。 ● **stream**：输入码流信息。
返回值说明	**img_info**：图片信息字典。 **ret**：int，错误码。 ● 返回0，表示成功。 ● 返回非0，表示失败。参见**10.8.9.1 公共返回码**。

10.8.3.6 acl.himpi.dvpp_get_version

c函数原型	**hi_s32 hi_mpi_dvpp_get_version(hi_s32 *major_version, hi_s32 *minor_version, hi_s32 *patch_version)**

Python函数	**major_version, minor_version, patch_version, ret = acl.himpi.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。
返回值说明	• **major_version**：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**：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**：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。 **ret**：int，错误码。 • 返回0表示成功。 • 返回非0表示失败。参见**10.8.9.1 公共返回码**。

10.8.3.7 acl.himpi.sys_create_epoll

c函数原型	**hi_s32 hi_mpi_sys_create_epoll(hi_s32 size, hi_s32 *epoll_fd);**
Python函数	**epoll_fd, ret = acl.himpi.sys_create_epoll(size)**
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。
输入说明	**size**：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。
返回值说明	**epoll_fd**：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。 **ret**：int，错误码。 • 返回0表示成功。 • 返回非0表示失败，参见**10.8.9.1 公共返回码**。
约束说明	• **acl.himpi.sys_create_epoll**、**acl.himpi.sys_ctl_epoll**、**acl.himpi.sys_wait_epoll**、**acl.himpi.sys_close_epoll**这几个接口要配合使用。使用epoll编程模型操作DVPP通道的输入/输出数据。 • VPC功能不支持使用该接口。

10.8.3.8 acl.himpi.sys_ctl_epoll

c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);**

Python函数	**ret = acl.himpi.sys_ctl_epoll(epoll_fd, operation, fd, event)**
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）。
输入说明	• **epoll_fd**：Epoll实例句柄，句柄需提前通过acl.himpi.sys_create_epoll接口创建。 • **operation**：操作类型（增、删、改），参见acl.hi.dvpp_epoll_ctl_op。 • **fd**：媒体数据处理通道的文件句柄。VENC场景下，您可以调用acl.himpi.venc_get_fd接口提前获取该文件句柄。VDEC场景下，您可以调用acl.himpi.vdec_get_fd接口提前获取该文件句柄。 • **event**：Epoll事件信息的指针地址。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，该字段无效。
返回值说明	**ret**：int，错误码。 • 返回0，表示成功。 • 返回非0，表示失败，参见10.8.9.1 公共返回码。
约束说明	• acl.himpi.sys_create_epoll、acl.himpi.sys_ctl_epoll、acl.himpi.sys_wait_epoll、acl.himpi.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。
参考资源	接口调用流程及示例，参见5.4.4 JPEGE图片编码、5.4.7 VENC视频编码。

10.8.3.9 acl.himpi.sys_wait_epoll

c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);**
Python函数	**event, event_num, ret = acl.himpi.sys_wait_epoll(epoll_fd, max_events, timeout)**
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。

输入说明	● **epoll_fd**：媒体数据处理Epoll实例句柄的指针。 ● **max_events**：最大可返回的事件数，不应超过events数组的元素个数。 ● **timeout**：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。
返回值说明	**event**：hi_dvpp_epoll_event数组，保存产生的事件信息。 **event_num**：Epoll实际产生的事件数。 ret：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见10.8.9.1 公共返回码。
约束说明	● acl.himpi.sys_create_epoll、acl.himpi.sys_ctl_epoll、acl.himpi.sys_wait_epoll、acl.himpi.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。
参考资源	接口调用流程及示例，参见5.4.4 JPEGE图片编码、5.4.7 VENC视频编码。

10.8.3.10 acl.himpi.sys_close_epoll

c函数原型	**hi_s32 hi_mpi_sys_close_epoll(hi_s32 epoll_fd)**
Python函数	**ret = acl.himpi.sys_close_epoll(epoll_fd)**
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。
输入说明	**epoll_fd**：待销毁媒体数据处理Epoll实例句柄，句柄通过acl.himpi.sys_create_epoll接口创建。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见10.8.9.1 公共返回码。
约束说明	● acl.himpi.sys_create_epoll、acl.himpi.sys_ctl_epoll、acl.himpi.sys_wait_epoll、acl.himpi.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。
注意事项	无

参考资源	接口调用流程及示例，参见**5.4.4 JPEGE图片编码**、**5.4.7 VENC视频编码**。

10.8.3.11 acl.himpi.sys_get_image_align_info

c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)**
Python函数	**img_align_info, ret = acl.himpi.sys_get_image_align_info(mod_type, mod_num, img_base_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。
输入说明	• **mod_type**：模块类型数组。模块类型是整数型常量. 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**：模块类型数组长度。 • **img_base_info**：图片基本信息字典。
返回值说明	**img_align_info**：图片对齐信息、内存大小的字典。 **ret**：int，错误码。 • 返回0表示成功。 • 返回非0表示失败，参见**10.8.9.1 公共返回码**。
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**10.8.4 VPC图像处理接口**、**10.8.5 VDEC视频解码/JPEGD图像解码接口**、**10.8.6 VENC视频/JPEGE图像编码接口**、**10.8.7 PNGD图像处理接口**下的约束说明。当前不支持设置HI_PIXEL_FORMAT_UNKNOWN格式。

10.8.3.12 acl.himpi.sys_set_chn_csc_matrix

c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);**
Python函数	**ret = acl.himpi.sys_set_chn_csc_matrix(mode, chn, csc_matrix, csc_coefficient)**

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。
输入说明	• **mode**：模块ID，当前只支持HI_ID_VDEC和HI_ID_VPC。 • **chn**：通道号。当mode设置为HI_ID_VDEC时，通道号范围请参见**10.8.5.5 acl.himpi.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**10.8.4.4 acl.himpi.vpc_create_chn**处chn参数的取值范围。 • **csc_matrix**：色域转换矩阵的数据标准。 • **csc_coefficient**：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	**ret**：int，错误码。 • 返回0，表示成功。 • 返回非0，表示失败，参见**10.8.9.1 公共返回码**。
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. **acl.himpi.sys_set_chn_csc_matrix**（第一次修改色域转换参数） 2. **acl.himpi.vpc_crop**（抠图） 3. **acl.himpi.vpc_get_process_result**（等待抠图任务结束） 4. **acl.himpi.sys_set_chn_csc_matrix**（第二次修改色域转换参数） 5. **acl.himpi.vpc_resize**（缩放） 6. **acl.himpi.vpc_get_process_result**（等待缩放任务结束）

10.8.3.13 acl.himpi.sys_get_chn_csc_matrix

c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);**

Python函数	**csc_matrix, csc_coefficient, ret = acl.himpi.sys_get_chn_csc_matrix(mode, chn)**
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。
输入说明	• **mode**：模块ID，当前只支持HI_ID_VDEC和HI_ID_VPC。 • **chn**：通道号。 – 当mode设置为HI_ID_VDEC时，通道号范围请参见**acl.himpi.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**acl.himpi.vpc_create_chn**处chn参数的取值范围。
返回值说明	• **ret**：int，错误码。 – 返回0，表示成功。 – 返回非0，表示失败，参见**10.8.9.1 公共返回码**。 • **csc_matrix**：色域转换矩阵的数据标准。 • **csc_coefficient**：色域转换矩阵参数值。
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**acl.himpi.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**。

10.8.4 VPC 图像处理接口

10.8.4.1 VPC 功能

功能说明

VPC（Vision Preprocessing Core）功能如下所示，

- **抠图**，从输入图片中抠出需要用的图片区域。
- **缩放**，对图片进行缩放，支持单图裁剪缩放、一图多框裁剪缩放、原图缩放等多种缩放方式。
- **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。
- **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**，对图像进行格式转换。
- **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。
- **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，功能示意图请参见**图10-8**。

  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。
- 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。
- 旋转：对输入图片做固定角度的旋转，支持90度/180度/270度。
- 马赛克：对输入图片做马赛克处理。
- 覆盖: 对输入图片做局部覆盖操作。
- 画线：对输入图片做画线处理。
- 添加水印：对输入图片做添加水印的处理。

功能示意图

图 10-5 VPC 功能示意图（抠图+缩放+叠加）

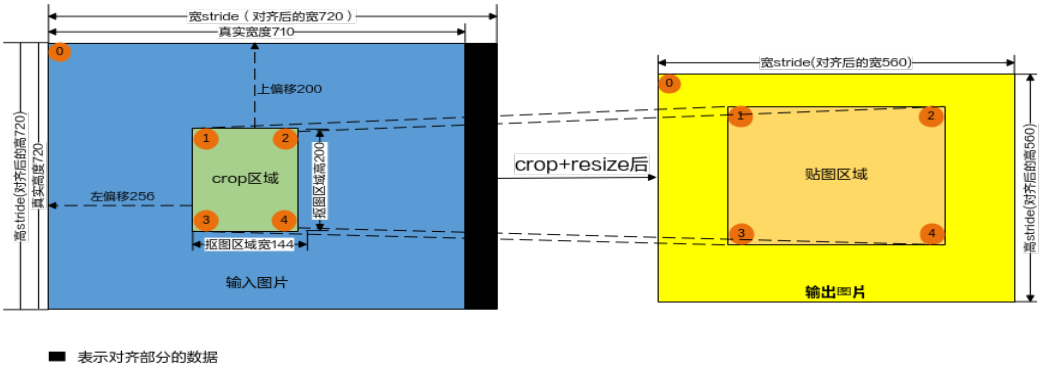


图 10-6 VPC 功能示意图（拼接）

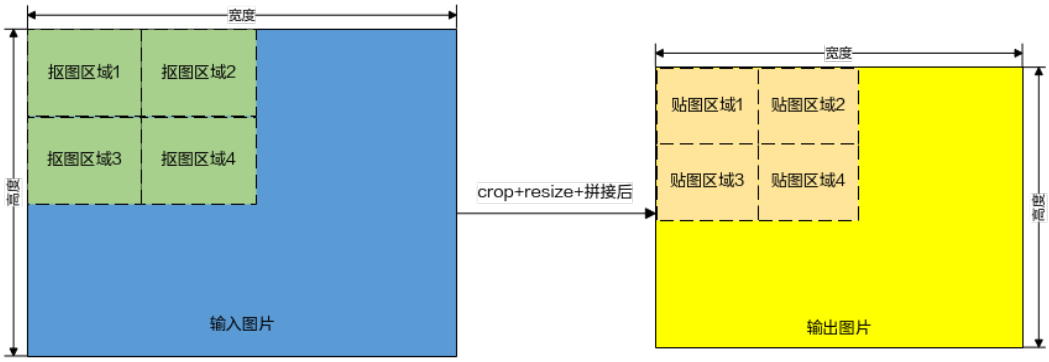


图 10-7 等比例缩放（贴图区域在输出图片的中心位置），即缩放前后图片的宽高比例相同

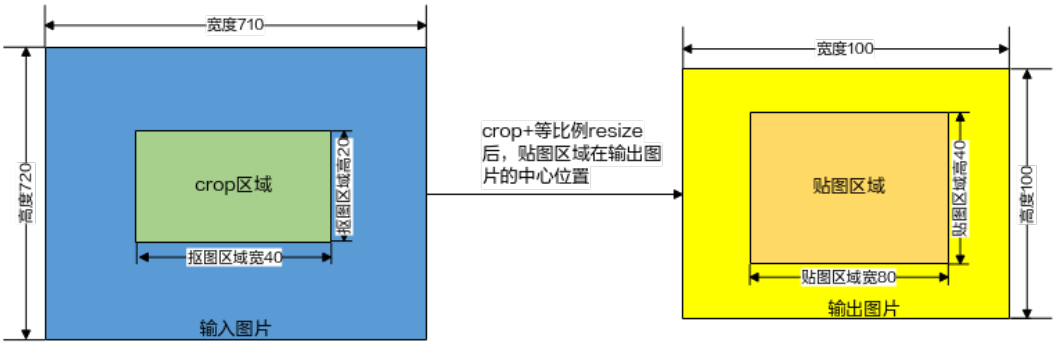
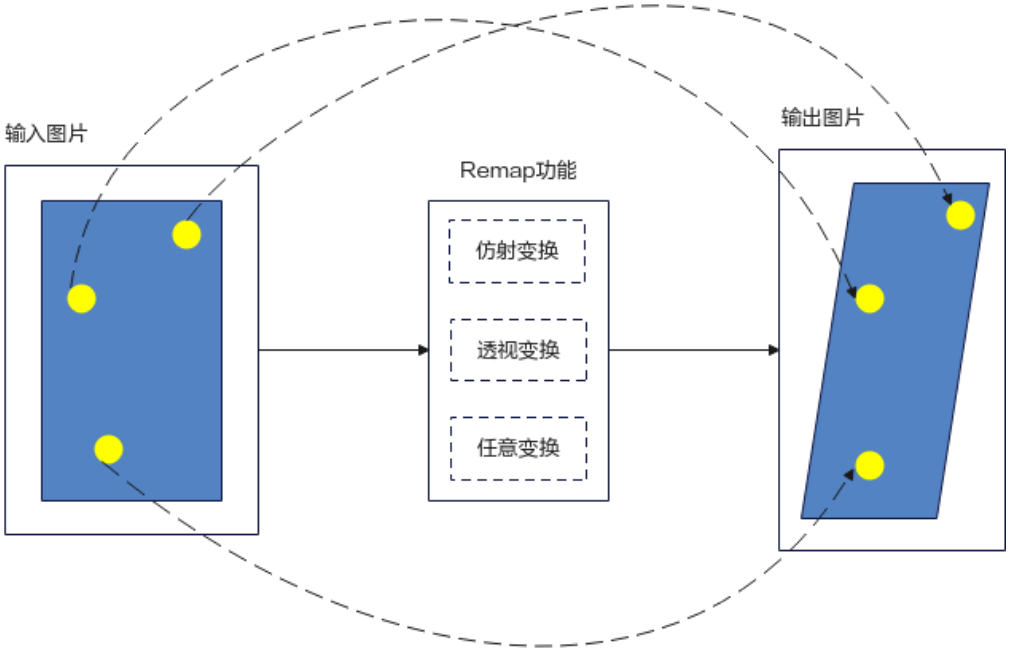


图 10-8 Remap 功能



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

10.8.4.2 约束说明

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**10.8.4 VPC图像处理接口**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

昇腾AI处理器	分辨率范围
Atlas 200/500 A2推理产品	10*6~4096*8192

图片格式、宽高对齐、内存约束

VPC在处理图片时，需调用**acl.himpi.dvpp_malloc**接口申请Device上的输入、输出内存，调用**acl.himpi.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、宽高对齐约束如下，参见表10-17、表10-18。

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10.8.2 基本概念。
- 宽stride最小10、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10.8.4 VPC图像处理接口下各接口的说明。

表 10-17 输入图片格式、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 200/500 A2 推理产品	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无对齐要求，与宽相同即可； 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无对齐要求，与宽相同即可； 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无对齐要求，与宽相同即可； 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无对齐要求，与宽相同即可； 高stride无对齐要求，与高相同即可。	宽stride * 高stride * 3

-	图片格式	图片宽、高对齐要求	图片宽stride、高stride对齐要求 Atlas 200/500 A2推理产品	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*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*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*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无对齐要求，与宽相同即可； 高stride为高2对齐后的值。	宽stride * 高stride * 2
该部分格式仅以下版本支持： Atlas 200/500 A2推理产品	YVU420Planar YUV420Planar	宽2对齐，高2对齐	宽stride无对齐要求，与宽相同即可；高stride无对齐要求，与高相同即可。	宽stride * 高stride * 3/2
	YVU422Planar YUV422Planar	无对齐要求	宽stride无对齐要求，与宽相同即可；高stride无对齐要求，与高相同即可。	宽stride * 高stride * 2
	YVU444Planar YUV444Planar	无对齐要求	宽stride无对齐要求，与宽相同即可；高stride无对齐要求，与高相同即可。	宽stride * 高stride * 3

-	图片格式	图片宽、高对齐要求	图片宽stride、高stride对齐要求 Atlas 200/500 A2推理产品	内存大小要求（单位Byte）
	YVU444Packed 8bit	无对齐要求	宽stride无对齐要求，宽stride为宽*3的值；高stride无对齐要求，与高相同即可。	宽stride * 高stride
	YUV440Planar YVU440Planar	宽无对齐要求，但宽≤4096 高2对齐	宽stride无对齐要求，与宽相同即可；高stride无对齐要求，与高相同即可。	宽stride * 高stride * 2
	RGB888Planar BGR888Planar	无对齐要求	宽stride无对齐要求，与宽相同即可；高stride无对齐要求，与高相同即可。	宽stride * 高stride * 3
RGB888 FP32 BGR888 FP32	无对齐要求	宽stride无对齐要求，宽stride为宽*3*4（4表示FP32占4个字节）的值；高stride无对齐要求，与高相同即可。	宽stride * 高stride	

表 10-18 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 200/500 A2推理产品	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无对齐要求，与宽相同即可； 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 200/500 A2推理产品	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无对齐要求，与宽相同即可； 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无对齐要求，与宽相同即可； 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*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*4的值； 高stride无对齐要求，与高相同即可。	宽stride * 高stride
该部分格式仅以下版本支持： Atlas 200/500 A2推理产品	YUV422Packed YUYV 8bit YUV422Packed UYVY 8bit YUV422Packed YVYU 8bit YUV422Packed VYUY 8bit	宽无对齐要求 高2对齐	宽stride为宽*2的值； 高stride无对齐要求，与高相同即可。	宽stride * 高stride
该部分格式仅以下版本支持： Atlas 200/500 A2推理产品	YVU444Packed	无对齐要求	宽stride为宽*3的值； 高stride无对齐要求，与高相同即可。	宽stride * 高stride
	RGB888Planar BGR888Planar	无对齐要求	宽stride无对齐要求，与宽相同即可；高stride无对齐要求，与高相同即可。	宽stride * 高stride * 3

-	图片格式	图片宽、高对齐要求	图片宽stride、高stride对齐要求 Atlas 200/500 A2推理产品	内存大小要求（单位Byte）
RGB888 FP32 BGR888 FP32	无对齐要求	宽stride无对齐要求，宽stride为宽*3*4（4表示FP32占4个字节）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。
- 贴图区域不超出输出图片区域，最大贴图个数256个。
- 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。

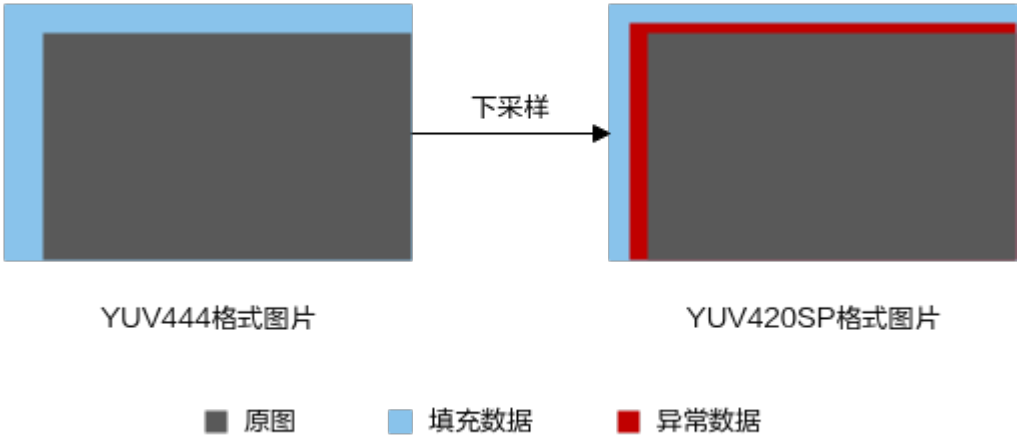
  输出图片格式为YUV422SP，贴图区域奇数、偶数限制为：左偏移为偶数、右偏移为奇数。

  其它格式，贴图区域没有偏移奇偶数的限制。

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 10-9 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。

10.8.4.3 性能指标说明

性能指标说明

单个Device场景下的性能指标参考如下（1路对应一个通道，一个通道对应一个线程，或者n路对应一个通道，一个通道对应n个线程）：

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<2）	n*800fps
• 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≥2）	1600fps
• 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<2）	n*200fps

场景举例	总帧率
• 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≥2）	400fps
• 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<2）	n*100fps
• 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≥2）	200fps

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acl.himpi.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。

10.8.4.4 acl.himpi.vpc_create_chn

c函数原型	**hi_s32 hi_mpi_vpc_create_chn(hi_vpc_chn chn, const hi_vpc_chn_attr *attr);**
Python函数	**ret = acl.himpi.vpc_create_chn(chn, attr)**
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。
输入说明	• **chn**：int，图片处理通道号。 该参数的取值范围：[0, 128)，通道总数最多128。 • **attr**：图片处理通道属性指针。
返回值说明	**ret**：int，错误码。 • 返回0，表示成功。 • 返回非0，表示失败，参见**10.8.9.2 VPC图像处理返回码**。
约束说明	• 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。

10.8.4.5 acl.himpi.vpc_destroy_chn

c函数原型	**hi_s32 hi_mpi_vpc_destroy_chn(hi_vpc_chn chn);**
Python函数	**ret = acl.himpi.vpc_destroy_chn(chn)**
函数功能	销毁创建的图像处理通道。
输入说明	**chn**：图片处理通道号。 该参数的取值范围：[0, 128)，通道总数最多128。
返回值说明	**ret**：int，错误码。 ● 返回0，表示成功。 ● 返回非0，表示失败，参见**10.8.9.2 VPC图像处理返回码**。
约束说明	通道号不能超出最大的通道号范围。

10.8.4.6 acl.himpi.vpc_resize

c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);**
Python函数	**task_id, ret = acl.himpi.vpc_resize(chn, source_pic, dest_pic, fx, fy, interpolation, milli_sec)**
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。

输入说明	• **chn**：int，图片处理通道号。 该参数的取值范围：[0, 128)，通道总数最多128。 • **source_pic**：原始图片信息。 • **dest_pic**：目标图片信息。 • **fx**：宽的缩放比例。 • **fy**：高的缩放比例。 • **interpolation**：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。 支持如下缩放算法： – 0：业界通用的Bilinear算法（与OpenCV-3.4.2版本算法的计算结果相同）。 – 1：业界通用的Nearest neighbor 算法（与OpenCV-3.4.2版本算法的计算结果相同）。 • **milli_sec**：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	• **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。 • **task_id**：此次任务分配的ID，用来区分任务。

约束说明

昇腾AI处理器	约束
Atlas 200/500 A2推理产品	● 使用本接口对输入、输出图片分辨率的要求如下： – 输入图片分辨率：10*6~16384*16384 当输出图片宽度大于4096时，输入图片最小分辨率为128*16。 当输入图片宽度大于8192，输入图片高度最小为16。 – 输出图片分辨率：10*6~16384*16384 ● 关于图片格式、宽高对齐、内存等约束，请参见**图片格式、宽高对齐、内存约束**。 ● 设置fx/fy之后，缩放后的图片的分辨率必须在[10*6~16384*16384]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。

10.8.4.7 acl.himpi.vpc_crop

c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);**
Python函数	**task_id, ret = acl.himpi.vpc_crop(chn, source_pic, crop_info, count, milli_sec)**
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。
输入说明	● **chn**：int，图片处理通道号。 该参数的取值范围：[0, 128)，通道总数最多128。 ● **source_pic**: 原始图片信息。 ● **crop_info**：抠图图片信息数组，该数组长度与count参数值保持一致。 ● **count**: 抠图区域的数量，取值范围[1,256]。 ● **milli_sec**：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	● **task_id**：此次任务分配的ID，用来区分任务。 ● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。
参考资源	接口调用流程，参见**典型功能接口调用流程（以缩放为例）**。

约束说明

昇腾AI处理器	约束
Atlas 200/500 A2推理产品	● 关于图片分辨率、图片格式、宽高对齐、内存等约束，请参见**10.8.4.2 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。

10.8.4.8 acl.himpi.vpc_crop_resize

c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);**
Python函数	**task_id, ret = acl.himpi.vpc_crop_resize(chn, source_pic, crop_resize_info, count, milli_sec)**
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。
输入说明	● **chn**：int，图片处理通道号。 该参数的取值范围：[0, 128)，通道总数最多128。 ● **source_pic**：原始图片信息。 ● **crop_resize_info**： 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**：抠图并缩放的图片数量，取值范围[1,256]。 ● **milli_sec**：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	● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码** ● **task_id**：任务分配的ID，用来区分任务。
注意事项	无

约束说明

昇腾AI处理器	约束
Atlas 200/500 A2推理产品	● 关于图片分辨率、图片格式、宽高对齐、内存等约束，请参见**10.8.4.2 约束说明**。 ● 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。

10.8.4.9 acl.himpi.vpc_crop_resize_paste

c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);**
Python函数	**task_id, ret = acl.himpi.vpc_crop_resize_paste(chn, source_pic, crop_resize_paste_info, count, milli_sec)**
函数功能	从输入图片中抠图，对抠出的子图进行缩放后，放在用户输出图片的指定区域。异步接口。
输入说明	● **chn**：int，图片处理通道号。 该参数的取值范围：[0, 128)，通道总数最多128。 ● **source_pic**：原始图片信息。 ● **crop_resize_paste_info**：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**：抠图缩放贴图的图片数量，取值范围[1,256]。 ● **milli_sec**：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	● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。 ● **task_id**：任务分配的ID，用来区分任务。
注意事项	无
参考资源	无

约束说明

昇腾AI处理器	约束
Atlas 200/500 A2推理产品	● 关于图片分辨率、图片格式、宽高对齐、内存等约束，请参见**10.8.4.2 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，贴图的顶部偏移、左侧偏移需为偶数。 ● 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。

10.8.4.10 acl.himpi.vpc_convert_color

c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);**
Python函数	**task_id, ret = acl.himpi.vpc_convert_color(chn, source_pic, dest_pic, milli_sec)**
函数功能	转换图片的格式。异步接口。

输入说明	● **chn**：int，图片处理通道号。 该参数的取值范围：[0, 128)，通道总数最多128。 ● **source_pic**：原始图片信息。输入分辨率为10*6~4096*4096（包括4096）。 ● **dest_pic**：目标图片信息。 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**。 ● **milli_sec**：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	● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。 ● **task_id**：此次任务分配的ID，用来区分任务。
约束说明	输入、输出图片的格式请参见**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。

10.8.4.11 acl.himpi.vpc_convert_color_v2

c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)**
Python函数	**task_id, ret = acl.himpi.vpc_convert_color_v2(chn, source_pic, dest_pic, conf, milli_sec)**
函数功能	转换图片的格式，在**10.8.4.10 acl.himpi.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。

输入说明	• **chn**：int，图片处理通道号。 该参数的取值范围：[0, 128)，通道总数最多128。 • **source_pic**：原始图片信息。输入分辨率为10*6~4096*4096（包括4096）。 • **dest_pic**：目标图片信息。 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**。 • **conf**：输出图片透明度值。 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, • **milli_sec**：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	• **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。 • **task_id**：任务分配的ID，用来区分任务。
约束说明	输入、输出图片的格式请参见**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。

10.8.4.12 acl.himpi.vpc_convert_color_to_yuv420

c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);**

Python函数	**task_id, ret = acl.himpi.vpc_convert_color_to_yuv420(chn, source_pic, dest_pic, milli_sec)**
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。

输入说明	• **chn**：int，图片处理通道号。
	该参数的取值范围：[0, 128)，通道总数最多128。
	• **source_pic**：原始图片信息。
	– 输入分辨率为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_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
	• **dest_pic**：目标图片信息。
	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。
	– 配置图片格式时，当前仅支持**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
	– 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**。
	• **milli_sec**：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	● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。 ● **task_id**：此次任务分配的ID，用来区分任务。
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**10.8.4.10 acl.himpi.vpc_convert_color**接口输出的二进制不一致。

10.8.4.13 acl.himpi.vpc_copy_make_border

c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);**
Python函数	**task_id, ret = acl.himpi.vpc_copy_make_border(chn, source_pic, dest_pic, make_border_info, milli_sec)**
函数功能	对输入图像进行边界填充。异步接口。
输入说明	● **chn**：int，图片处理通道号。 该参数的取值范围：[0, 128)，通道总数最多128。 ● **source_pic**：原始图片信息。 ● **dest_pic**：目标图片信息。 ● **make_border_info**：边界填充信息。 ● **milli_sec**：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	● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。 ● **task_id**：任务分配的ID，用来区分任务。

约束说明

昇腾AI处理器	约束
Atlas 200/500 A2推理产品	● 输入、输出图片分辨率为10*6~4096*8192。 ● 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，请参见请参见图片格式、宽高对齐、内存约束。 ● 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。

10.8.4.14 acl.himpi.vpc_pyrdown

c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);**
Python函数	**task_id, ret = acl.himpi.vpc_pyrdown(chn, source_pic, dest_pic, filter_level, gaussian_filter, divisor, make_border_info, milli_sec)**
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 200/500 A2推理产品，不支持该接口。

输入说明	• **chn**：int，图片处理通道号。 该参数的取值范围：[0, 128)，通道总数最多128。 • **source_pic**：输入图片信息。输入图片的分辨率最大支持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**：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**：参数有效范围是1-4，指定金字塔图像层数以及图像数量。 • **gaussian_filter**：高斯滤波参数，有默认值，如下： {{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**：滤波器除数，必须是2的幂次方。 • **make_border_info**：边界填充信息。填充固定填充2个像素，支持的填充类型为：HI_BORDER_CONSTANT、HI_BOARD_REPLICATE和HI_BOARD_REFLECT。 • **milli_sec**：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	• **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。 • **task_id**：任务分配的ID，用来区分任务。

10.8.4.15 acl.himpi.vpc_calc_hist

须知

此接口后续版本会废弃，请使用10.8.4.16 acl.himpi.vpc_calc_hist_v2接口。

c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);**
Python函数	**hist_config, task_id, ret = acl.himpi.vpc_calc_hist(chn, source_pic, milli_sec)**
函数功能	统计图像每个通道的像素值分布。异步接口。

输入说明	• **chn**：int，图片处理通道号。
	该参数的取值范围：[0, 128)，通道总数最多128。
	• **source_pic**：输入图片信息。
	– 输入图片分辨率：10*6~4096*8192（包括4096*8192）。
	– 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
	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。
	• **milli_sec**：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	● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码** ● **hist_config**：统计结果。 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**：任务分配的ID，用来区分任务。

10.8.4.16 acl.himpi.vpc_calc_hist_v2

c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);**
Python函数	**task_id, ret = acl.himpi.vpc_calc_hist_v2(chn, source_pic, milli_sec, hist_config)**
函数功能	统计图像每个通道的像素值分布。异步接口。

输入说明	● **chn**：int，图片处理通道号。
	该参数的取值范围：[0, 128)，通道总数最多128。
	● **source_pic**：输入图片信息。
	– 输入分辨率为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_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
	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。
	● **milli_sec**：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。
	● **hist_config**：统计结果指针地址。
	hist_config中的地址指向包含统计结果的结构体，用来存放每张图像3个分量0~255的分布情况。
	调用np_out_size = np.array([out_buffer_size], dtype=np.int32)创建numpy array类型数据。
	再调用bytes_data = np_out_size.tobytes()-> \->size_ptr = **acl.util.bytes_to_ptr**(bytes_data)，将数据转换为对象作为入参。函数调用成功后，size_ptr指向的数据（即np_out_size）会发生改变。

返回值说明	● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。 ● **task_id**：任务分配的ID，用来区分任务。

10.8.4.17 acl.himpi.vpc_equalize_hist

c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);**
Python函数	**task_id, ret = acl.himpi.vpc_equalize_hist(chn, source_pic, dest_pic, lut_remap, milli_sec)**
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，主要用户图像增强的场景。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。

输入说明	● **chn**：int，图片处理通道号。
	该参数的取值范围：[0, 128)，通道总数最多128。
	● **source_pic**：输入图片信息。
	– 输入分辨率为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
	● **dest_pic**：目标图片信息。输出图片格式与输入图片格式保持一致。
	● **lut_remap**：图像lut_remap的配置。lut_remap是一个768 Byte的数组，配置给硬件进行图片像素重映射。
	● **milli_sec**：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	● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见10.8.9.2 VPC图像处理返回码。 ● **task_id**：任务分配的ID，用来区分任务。

10.8.4.18 acl.himpi.vpc_get_process_result

c函数原型	**hi_s32 hi_mpi_vpc_get_process_result(hi_vpc_chn chn, hi_u32 task_id, hi_s32 milli_sec);**

Python函数	**ret = acl.himpi.vpc_get_process_result(chn, task_id, milli_sec)**
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：**10.8.4.7 acl.himpi.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。
输入说明	● **chn**：int，图片处理通道号。 该参数的取值范围：[0, 128)，通道总数最多128。 ● **milli_sec**：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**：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**acl.himpi.vpc_resize**~**acl.himpi.vpc_batch_crop_resize_make_border**。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。

10.8.4.19 acl.himpi.vpc_sys_create_chn

c函数原型	**hi_s32 hi_mpi_vpc_sys_create_chn(hi_vpc_chn *chnl, const hi_vpc_chn_attr *attr);**
Python函数	**chnl, ret = acl.himpi.vpc_sys_create_chn(attr)**
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。
输入说明	**attr**：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	● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。 ● **chnl**：图片处理通道号，调用该接口后，由系统返回可用通道号给用户。 该参数的取值范围：[0, 128)，通道总数最多128。

10.8.4.20 acl.himpi.vpc_crop_resize_make_border

c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);**
Python函数	**task_id, ret = acl.himpi.vpc_crop_resize_make_border(chn, source_pic, crop_resize_make_border_info, count, milli_sec)**
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。
输入说明	● **chn**：int，图片处理通道号。 该参数的取值范围：[0, 128)，通道总数最多128。 ● **source_pic**：原始图片信息。 ● **crop_resize_make_border_info**： 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**：抠图缩放填充的图片数量，取值范围[1,256]。 ● **milli_sec**：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	● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。 ● **task_id**：任务分配的ID，用来区分任务。
注意事项	无
参考资源	无

约束说明

昇腾AI处理器	约束
Atlas 200/500 A2推理产品	● 关于图片分辨率、图片格式、宽高对齐、内存等约束，请参见10.8.4.2 约束说明。 ● 设置fx/fy之后，缩放后的图片的分辨率必须在[10*6~16384*16384]范围内，否则返回错误。 ● 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。

10.8.4.21 acl.himpi.vpc_batch_crop_resize_paste

c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);**
Python函数	**task_id, ret =acl.himpi.vpc_batch_crop_resize_paste(chn, source_pic, pic_num, crop_resize_paste_info, count, milli_sec)**
函数功能	acl.himpi.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。

输入说明	● **chn**：int，图片处理通道号。 该参数的取值范围：[0, 128)，通道总数最多128。 ● **source_pic**：原始图片信息的指针数组。 ● **pic_num**：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**：抠图缩放并贴图信息字典链表，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**：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]。 ● **milli_sec**：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	● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。 ● **task_id**：任务分配的ID，用来区分任务。
注意事项	无
参考资源	无

约束说明

昇腾AI处理器	约束
Atlas 200/500 A2推理产品	● 关于图片分辨率、各图片格式的宽高对齐、内存大小约束，请参见10.8.4.2 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，贴图的顶部偏移、左侧偏移建议为偶数。 ● 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。

10.8.4.22 acl.himpi.vpc_batch_crop_resize_make_border

c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);**
Python函数	**task_id, ret = acl.himpi.vpc_batch_crop_resize_make_border(chn, source_pic, pic_num, crop_resize_make_border_info, count, milli_sec)**
函数功能	10.8.4.20 acl.himpi.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。

输入说明	● **chn**：int，图片处理通道号。 该参数的取值范围：[0, 128)，通道总数最多128。 ● **source_pic**：原始图片信息的指针数组。 ● **pic_num**：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**：抠图缩放填充的信息字典，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，可以填充到4096*4096。 ● **count**：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]。 ● **milli_sec**：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	● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。 ● **task_id**：任务分配的ID，用来区分任务。
注意事项	无
参考资源	无

约束说明

昇腾AI处理器	约束
Atlas 200/500 A2推理产品	● 关于图片分辨率、各图片格式的宽高对齐、内存大小约束，请参见10.8.4.2 约束说明。 ● 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。

10.8.4.23 acl.himpi.vpc_blur

c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)**
Python函数	**task_id, ret = acl.himpi.vpc_blur(chn, blur_param, milli_sec)**
函数功能	对输入图片做均值滤波处理。

输入说明	● **chn**：int，图片处理通道号。
	该参数的取值范围：[0, 128)，通道总数最多128。
	● **blur_param**：dict，滤波信息字典。
	在**10.8.8.5.28 hi_blur_param**字典内配置输入、输出图片信息，以及均值滤波参数。
	– 输入、输出图片分辨率的取值范围为：10*6~4096*8192，且输入、输出图片的分辨率需保持一致。
	– 支持如下输入图片格式（输出图片格式与输入保持一致）：
	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_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	HI_PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 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_YVU_PACKED_444 = 58, // YVU444 Package 8bit
	HI_PIXEL_FORMAT_RGB_888_PLANAR = 69, // RGB888 Planar
	HI_PIXEL_FORMAT_BGR_888_PLANAR = 70, // BGR888 Planar
	● **milli_sec**：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	● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。 ● **task_id**：任务分配的ID，用来区分任务。
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，在配置边界填充类型时，建议设置为“边界复制模式”，避免输出图片边缘异常数据。 实现原理和opencv一致，浮点转定点运算部分有差异，精度不能完全对标。

10.8.4.24 acl.himpi.vpc_crop_resize_resize_paste

c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)**

Python函数	**task_id, ret = acl.himpi.vpc_crop_resize_resize_paste(chn, source_pic,crop_resize_resize_paste_info,count,milli_sec)**
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 200/500 A2推理产品，不支持该接口。
输入说明	• **chn**：int，图片处理通道号。 该参数的取值范围：[0, 128)，通道总数最多128。 • **source_pic**：dict，原始图片信息字典，请参见**10.8.8.5.1 hi_vpc_pic_info** 。 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。 • **crop_resize_resize_paste_info**：list，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。元素类型参见**10.8.8.5.15 hi_vpc_crop_resize_resize_paste_region**。 • **count**：int，抠图缩放贴图的图片数量，取值范围[1,256]。 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。 • **milli_sec**：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	• **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。 • **task_id**：任务分配的ID，用来区分任务。
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**字典中指定的地址。 • 两次缩放中，只能先缩小再放大，每次缩放后的图片的分辨率必须在[10*6, 4096*4096]范围内、宽高缩放比例在[1/32, 32]范围内，两次缩放之后的图片大小必须与抠图的大小保持一致。 • VPC功能的约束请参见**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，贴图的顶部偏移、左侧偏移建议为偶数。

10.8.4.25 acl.himpi.vpc_dilate

c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)**
Python函数	**task_id, ret = acl.himpi.vpc_dilate(chn, dilate_param, milli_sec)**
函数功能	对输入图片做膨胀处理。

输入说明	• **chn**：int，图片处理通道号。
	该参数的取值范围：[0, 128)，通道总数最多128。
	• **dilate_param**：dict，膨胀信息字典。
	在10.8.8.5.28 hi_blur_param字典内配置输入、输出图片信息，以及膨胀参数。
	– 输入、输出图片分辨率的取值范围为：10*6~4096*8192，且输入、输出图片的分辨率需保持一致。
	– 支持如下输入图片格式（输出图片格式与输入保持一致）：
	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_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	HI_PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 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_YVU_PACKED_444 = 58, // YVU444 Package 8bit
	HI_PIXEL_FORMAT_RGB_888_PLANAR = 69, // RGB888 Planar
	HI_PIXEL_FORMAT_BGR_888_PLANAR = 70, // BGR888 Planar
	• **milli_sec**：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	• **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见10.8.9.2 VPC图像处理返回码。 • **task_id**：任务分配的ID，用来区分任务。
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，在配置边界填充类型时，建议设置为“边界复制模式”，避免输出图片边缘异常数据。

10.8.4.26 acl.himpi.vpc_draw_cover

c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)**
Python函数	**task_id, ret = acl.himpi.vpc_draw_cover(chn, cover_param, milli_sec)**
函数功能	对输入图片的部分区域做覆盖处理。

输入说明	• **chn**：int，图片处理通道号。
	该参数的取值范围：[0, 128)，通道总数最多128。
	• **cover_param**：dict，覆盖信息字典。
	在10.8.8.5.38 hi_cover_param字典内配置输入、输出图片信息，以及覆盖参数。
	– 输入、输出图片分辨率的取值范围为：10*6~4096*8192，且输入、输出图片的分辨率需保持一致。
	– 支持如下输入图片格式（输出图片格式与输入保持一致）：
	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_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	HI_PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 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_YVU_PACKED_444 = 58, // YVU444 Package 8bit
	HI_PIXEL_FORMAT_RGB_888_PLANAR = 69, // RGB888 Planar
	HI_PIXEL_FORMAT_BGR_888_PLANAR = 70, // BGR888 Planar
	• **milli_sec**：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	• **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见10.8.9.2 VPC图像处理返回码。 • **task_id**：任务分配的ID，用来区分任务。

10.8.4.27 acl.himpi.vpc_draw_line

c函数原型	**hi_s32 hi_mpi_vpc_draw_line(hi_vpc_chn chn, const hi_line_param* cover_param, hi_u32 *task_id, hi_s32 milli_sec)**
Python函数	**task_id, ret = acl.himpi.vpc_draw_line(chn, cover_param, milli_sec)**

函数功能	对输入图片做画线处理。

输入说明	● **chn**：int，图片处理通道号。
	该参数的取值范围：[0, 128)，通道总数最多128。
	● **line_param**：画线信息字典。
	在**10.8.8.5.41 hi_line_param**字典内配置输入、输出图片信息，以及画线参数。
	– 输入、输出图片分辨率的取值范围为：10*6~4096*8192，且输入、输出图片的分辨率需保持一致。
	– 支持如下输入图片格式（输出图片格式与输入保持一致）：
	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_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	HI_PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 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_YVU_PACKED_444 = 58, // YVU444 Package 8bit
	HI_PIXEL_FORMAT_RGB_888_PLANAR = 69, // RGB888 Planar
	HI_PIXEL_FORMAT_BGR_888_PLANAR = 70, // BGR888 Planar
	● **milli_sec**：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	● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。 ● **task_id**：任务分配的ID，用来区分任务。

10.8.4.28 acl.himpi.vpc_draw_mosaic

c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)**
Python函数	**task_id, ret = acl.himpi.vpc_draw_mosaic(chn, mosaic_param, milli_sec)**
函数功能	对输入图片的部分区域做马赛克处理。

输入说明	● **chn**：int，图片处理通道号。
	该参数的取值范围：[0, 128)，通道总数最多128。
	● **mosaic_param**：dict，马赛克信息。
	在**10.8.8.5.34 hi_mosaic_param**字典内配置输入、输出图片信息，以及马赛克参数。
	– 输入、输出图片分辨率的取值范围为：10*6~4096*8192，且输入、输出图片的分辨率需保持一致。
	– 支持如下输入图片格式（输出图片格式与输入保持一致）：
	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_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	HI_PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 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_YVU_PACKED_444 = 58, // YVU444 Package 8bit
	HI_PIXEL_FORMAT_RGB_888_PLANAR = 69, // RGB888 Planar
	HI_PIXEL_FORMAT_BGR_888_PLANAR = 70, // BGR888 Planar
	● **milli_sec**：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	● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。 ● **task_id**：任务分配的ID，用来区分任务。

10.8.4.29 acl.himpi.vpc_draw_osd

c函数原型	**hi_s32 hi_mpi_vpc_draw_osd(hi_vpc_chn chn, const hi_osd_param* osd_param, hi_u32 *task_id, hi_s32 milli_sec)**
Python函数	**task_id, ret = acl.himpi.vpc_draw_osd(chn, osd_param, task_id, milli_sec)**
函数功能	对输入图片做叠加处理，可用于添加水印的场景。

输入说明	● **chn**：int，图片处理通道号。
	该参数的取值范围：[0, 128)，通道总数最多128。
	● **osd_param**：dict，叠加信息字典。
	在**10.8.8.5.43 hi_osd_param**字典内配置输入、输出图片信息，以及叠加参数。
	– 输入、输出图片分辨率的取值范围为：10*6~4096*8192，且输入、输出图片的分辨率需保持一致。
	– 支持如下输入图片格式（输出图片格式与输入保持一致）：
	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_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	HI_PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 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_YVU_PACKED_444 = 58, // YVU444 Package 8bit
	HI_PIXEL_FORMAT_RGB_888_PLANAR = 69, // RGB888 Planar
	HI_PIXEL_FORMAT_BGR_888_PLANAR = 70, // BGR888 Planar
	● **milli_sec**：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	● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。 ● **task_id**：任务分配的ID，用来区分任务。

10.8.4.30 acl.himpi.vpc_erode

c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)**
Python函数	**task_id, ret = acl.himpi.vpc_erode(chn, erode_param, milli_sec)**
函数功能	对输入图片做腐蚀处理。

输入说明	● **chn**：int，图片处理通道号。
	该参数的取值范围：[0, 128)，通道总数最多128。
	● **erode_param**：dict，腐蚀信息字典。
	在**10.8.8.5.28 hi_blur_param**字典内配置输入、输出图片信息，以及腐蚀参数。
	– 输入、输出图片分辨率的取值范围为：10*6~4096*8192，且输入、输出图片的分辨率需保持一致。
	– 支持如下输入图片格式（输出图片格式与输入保持一致）：
	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_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	HI_PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 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_YVU_PACKED_444 = 58, // YVU444 Package 8bit
	HI_PIXEL_FORMAT_RGB_888_PLANAR = 69, // RGB888 Planar
	HI_PIXEL_FORMAT_BGR_888_PLANAR = 70, // BGR888 Planar
	● **milli_sec**：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	● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。 ● **task_id**：任务分配的ID，用来区分任务。
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，在配置边界填充类型时，建议设置为“边界复制模式”，避免输出图片边缘异常数据。

10.8.4.31 acl.himpi.vpc_filter2d

c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)**
Python函数	**task_id, ret = acl.himpi.vpc_filter2d(chn, filter_2d_param, milli_sec)**
函数功能	对输入图片做2D卷积滤波处理。

输入说明	● **chn**：int，图片处理通道号。
	该参数的取值范围：[0, 128)，通道总数最多128。
	● **filter_2d_param**：dict，2D卷积滤波信息字典。
	在**10.8.8.5.32 hi_filter_2d_param**字典内配置输入、输出图片信息，以及卷积滤波参数。
	– 输入、输出图片分辨率的取值范围为：10*6~4096*8192，且输入、输出图片的分辨率需保持一致。
	– 支持如下输入图片格式（输出图片格式与输入保持一致）：
	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_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	HI_PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 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_YVU_PACKED_444 = 58, // YVU444 Package 8bit
	HI_PIXEL_FORMAT_RGB_888_PLANAR = 69, // RGB888 Planar
	HI_PIXEL_FORMAT_BGR_888_PLANAR = 70, // BGR888 Planar
	● **milli_sec**：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	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。 ● **task_id**：任务分配的ID，用来区分任务。
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，在配置边界填充类型时，建议设置为“边界复制模式”，避免输出图片边缘异常数据。 实现原理和OpenCV一致，浮点转定点运算部分有差异，精度不能完全对标。 当前版本仅支持处理float32类型的整数，如0.0, 1.0。

10.8.4.32 acl.himpi.vpc_flip

c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)**
Python函数	**task_id, ret = acl.himpi.vpc_flip(chn, flip_param, milli_sec)**

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。

输入说明	• **chn**：int，图片处理通道号。
	该参数的取值范围：[0, 128)，通道总数最多128。
	• **flip_param**：dict，翻转信息字典。
	在**10.8.8.5.48 hi_flip_param**字典内配置输入、输出图片信息以及翻转模式。
	– 输入、输出图片分辨率的取值范围为：10*6~4096*8192，且输入、输出图片的分辨率需保持一致。
	– 支持如下输入图片格式（输出格式需与输入保持一致）：
	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_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, // YUV444 Package 8bit
	HI_PIXEL_FORMAT_RGB_888 = 12, // RGB888
	HI_PIXEL_FORMAT_BGR_888 = 13, // BGR888
	HI_PIXEL_FORMAT_ARGB_8888 = 14,
	// ARGB_8888
	HI_PIXEL_FORMAT_ABGR_8888 = 15,
	// ABGR_8888
	HI_PIXEL_FORMAT_RGBA_8888 = 16,
	// RGBA_8888
	HI_PIXEL_FORMAT_BGRA_8888 = 17,
	// BGRA_8888
	HI_PIXEL_FORMAT_YVU_PLANAR_444 = 22,
	// YVU444 Planar 8bit
	HI_PIXEL_FORMAT_YVU_PACKED_444 = 58, // YVU444 Package 8bit
	HI_PIXEL_FORMAT_RGB_888_PLANAR = 69, // RGB888 Planar
	HI_PIXEL_FORMAT_BGR_888_PLANAR = 70, // BGR888 Planar
	• **milli_sec**：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	• **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。 • **task_id**：任务分配的ID，用来区分任务。

10.8.4.33 acl.himpi.vpc_gaussian_blur

c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)**
Python函数	**task_id, ret = acl.himpi.vpc_gaussian_blur(chn, gaussian_blur_param, milli_sec)**
函数功能	对输入图片做高斯滤波处理。

输入说明	• **chn**：int，图片处理通道号。
	该参数的取值范围：[0, 128)，通道总数最多128。
	• **gaussian_blur_param**：dict，高斯滤波信息字典。
	在**10.8.8.5.30 hi_gaussian_blur_param**字典内配置输入、输出图片信息，以及高斯滤波参数。
	– 输入、输出图片分辨率的取值范围为：10*6~4096*8192，且输入、输出图片的分辨率需保持一致。
	– 支持如下输入图片格式（输出图片格式与输入保持一致）：
	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_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	HI_PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 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_YVU_PACKED_444 = 58, // YVU444 Package 8bit
	HI_PIXEL_FORMAT_RGB_888_PLANAR = 69, // RGB888 Planar
	HI_PIXEL_FORMAT_BGR_888_PLANAR = 70, // BGR888 Planar
	• **milli_sec**：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	• **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。 • **task_id**：任务分配的ID，用来区分任务。
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，在配置边界填充类型时，建议设置为“边界复制模式”，避免输出图片边缘异常数据。 实现原理和opencv一致，浮点转定点运算部分有差异，精度不能完全对标。

10.8.4.34 acl.himpi.vpc_get_affine_lut

c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)**
Python函数	**remap_lut, ret = acl.himpi.vpc_get_affine_lut(point_pair_info, interpolation, remap_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。
输入说明	• **point_pair_info**：dict，包含三对输入、输出图片中的像素点坐标信息的字典，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**：int，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）。 • **remap_lut**：dict，仿射变换像素位置重映射表信息字典。字典中的lut属性需由用户提前申请内存，内存大小“lut_size”属性可调用**acl.himpi.vpc_get_lut_mem_size**接口获取。
返回值说明	• **remap_lut**：dict，仿射变换像素位置重映射表信息字典。在成功调用本接口后，用户可通过lut属性中的地址获取仿射变换LUT表信息。 • **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。

10.8.4.35 acl.himpi.vpc_get_affine_transform

c函数原型	**hi_s32 hi_mpi_vpc_get_affine_transform(hi_point_pair_info *point_pair_info, hi_transform_matrix *matrix)**
Python函数	**matrix, ret = acl.himpi.vpc_get_affine_transform(point_pair_info)**
函数功能	获取仿射变换需要的变换矩阵。
输入说明	**point_pair_info**：dict，包含三对输入、输出图片中的像素点坐标信息的字典，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。

返回值说明	**matrix**：dict，仿射变换需要的变换矩阵信息字典，其中“matrix_type”参数值请参考实际输入矩阵进行配置，具体参见10.8.8.5.46 hi_transform_matrix。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见10.8.9.2 VPC图像处理返回码。

10.8.4.36 acl.himpi.vpc_get_lut_mem_size

c函数原型	**hi_s32 hi_mpi_vpc_get_lut_mem_size(hi_u32 width, hi_u32 height, hi_u32 *lut_size)**
Python函数	**lut_size, ret = acl.himpi.vpc_get_lut_mem_size(width, height)**
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。
输入说明	● **width**：int，输出图像宽度。 ● **height**：int，输出图像高度。
返回值说明	● **lut_size**：int，像素位置重映射表的内存大小，单位Byte。 ● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见10.8.9.2 VPC图像处理返回码。
约束说明	调用本接口时，width * height的取值范围为：10*6~4096*8192。

10.8.4.37 acl.himpi.vpc_get_perspective_lut

c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)**
Python函数	**remap_lut, ret = acl.himpi.vpc_get_perspective_lut(point_pair_info, interpolation, remap_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。

输入说明	• **point_pair_info**：dict，包含四对输入、输出图片中的像素点坐标信息字典，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**：int，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）。 • **remap_lut**：dict，透视变换像素位置重映射表字典。字典中的lut属性需由用户提前申请内存，内存大小“lut_size”属性可调用acl.himpi.vpc_get_lut_mem_size接口获取。
返回值说明	• **remap_lut**：dict，透视变换像素位置重映射表字典。在成功调用本接口后，用户可通过lut属性中的地址获取仿射变换LUT表信息。 • **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见10.8.9.2 VPC图像处理返回码。

10.8.4.38 acl.himpi.vpc_get_remap_lut

c函数原型	**hi_s32 hi_mpi_vpc_get_remap_lut(hi_map_param *map_param, hi_remap_lut *remap_lut)**
Python函数	**remap_lut, ret =acl.himpi.vpc_get_remap_lut(map_param, remap_lut)**
函数功能	根据用户输入的Remap表（例如仿射、透视、鱼眼等），获取系统使用的像素位置重映射表。同步接口。
输入说明	• **map_param**：dict，用户原始map表信息字典。请参见10.8.8.5.50 hi_map_param。 • **remap_lut**：dict，透视变换像素位置重映射表字典。字典中的lut属性需由用户提前申请内存，内存大小“lut_size”属性可调用acl.himpi.vpc_get_lut_mem_size接口获取。
返回值说明	• **remap_lut**：dict，透视变换像素位置重映射表字典。在成功调用本接口后，用户可通过lut属性中的地址获取仿射变换LUT表信息。 • **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见10.8.9.2 VPC图像处理返回码。

10.8.4.39 acl.himpi.vpc_get_rotation_matrix

c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)**
Python函数	**matrix, ret = acl.himpi.vpc_get_rotation_matrix(center_point, angle, scale)**
函数功能	获取仿射变换需要的旋转矩阵。
输入说明	• **center_point**：dict，输入图像的旋转中心点字典。 • **angle**：float，旋转角度，正值表示逆时针旋转。 • **scale**：float，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。
返回值说明	• **matrix**：dict，仿射变换需要的变换矩阵信息字典，其中“matrix_type”参数值请参考实际输入矩阵进行配置，具体参见**10.8.8.5.46 hi_transform_matrix**。 • **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。

10.8.4.40 acl.himpi.vpc_lut_remap

c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)**
Python函数	**task_id, ret = acl.himpi.vpc_lut_remap(chn, transform_param, remap_lut, milli_sec)**
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。

输入说明	● **chn**：int，图片处理通道号。
	该参数的取值范围：[0, 128)，通道总数最多128。
	● **transform_param**：dict，Remap变换参数字典。
	在**10.8.8.5.21 hi_warp_transform_param**字典内配置输入、输出图片信息时：
	– 输入、输出图片分辨率的取值范围为：10*6~4096*8192。
	– 支持如下输入图片格式：
	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_RGB_888 = 12, // RGB888
	HI_PIXEL_FORMAT_BGR_888 = 13, // BGR888
	HI_PIXEL_FORMAT_YUV_PACKED_444 = 11, // YUV444P 8bit
	HI_PIXEL_FORMAT_YVU_PACKED_444 = 58, // YVU444P 8bit
	– 支持如下输出图片格式：
	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_RGB_888 = 12, // RGB888
	HI_PIXEL_FORMAT_BGR_888 = 13, // BGR888
	HI_PIXEL_FORMAT_YUV_PACKED_444 = 11, // YUV444P 8bit
	HI_PIXEL_FORMAT_YVU_PACKED_444 = 58, // YVU444P 8bit
	HI_PIXEL_FORMAT_YUYV_PACKED_422 = 7, // YUV422P YUYV 8bit
	HI_PIXEL_FORMAT_UYVY_PACKED_422 = 8, // YUV422P UYVY 8bit
	HI_PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422P YVYU 8bit
	HI_PIXEL_FORMAT_VYUY_PACKED_422 = 10, // YUV422P VYUY 8bit
	● **remap_lut**：像素位置重映射表信息，具体请参见**10.8.8.5.18 hi_remap_lut**。
	需提前调用以下get接口获取像素位置重映射表信息：**acl.himpi.vpc_get_affine_lut**、或**acl.himpi.vpc_get_perspective_lut**、或**acl.himpi.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**：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	● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。 ● **task_id**：任务分配的ID，用来区分任务。

约束说明	• 在任务执行过程中，需要临时缓存，因此每个通道第一次执行本接口前需先调用**acl.himpi.vpc_set_chn_workspace**接口申请内部临时缓存。 由于硬件约束，除YUV420SP NV12、YUV420SP NV21图片格式外，其它格式在输入与输出保持一致的情况下，无需调用**acl.himpi.vpc_set_chn_workspace**接口申请内部临时缓存，达到节省内存的目的。 • VPC功能中，关于图片分辨率、图片格式、宽高对齐、内存等约束，请参见**约束说明**。 • 单张1080P（1920*1080）的图像处理超时时间建议设置10ms，其他分辨率，按照图像尺寸进行近似的等价折算，例如，4096*4096分辨率的图像建议超时时间设置为40ms。 • 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，在配置边界填充类型时，建议设置为“边界复制模式”，避免输出图片边缘异常数据。

10.8.4.41 acl.himpi.vpc_median_blur

c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)**
Python函数	**task_id, ret = acl.himpi.vpc_median_blur(chn, median_blur_param, milli_sec)**
函数功能	对输入图片做中值滤波处理。

输入说明	● **chn**：int，图片处理通道号。
	该参数的取值范围：[0, 128)，通道总数最多128。
	● **median_blur_param**：dict，中值滤波信息字典。
	在**10.8.8.5.26 hi_median_blur_param**字典内配置输入、输出图片信息，以及中值滤波参数。其中，中值滤波参数在**10.8.8.5.26 hi_median_blur_param**字典内配置。
	– 输入、输出图片分辨率的取值范围为：10*6~4096*8192，且输入、输出图片的分辨率需保持一致。
	– 支持如下输入图片格式（输出图片格式与输入保持一致）：
	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_YUYV_PACKED_422 = 7, // YUV422Packed YUYV 8bit
	HI_PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422Packed YVYU 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_YVU_PACKED_444 = 58, // YVU444 Package 8bit
	HI_PIXEL_FORMAT_RGB_888_PLANAR = 69, // RGB888 Planar
	HI_PIXEL_FORMAT_BGR_888_PLANAR = 70, // BGR888 Planar
	● **milli_sec**：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	● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。 ● **task_id**：任务分配的ID，用来区分任务。

10.8.4.42 acl.himpi.vpc_rotate

c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)**
Python函数	**task_id, ret = acl.himpi.vpc_rotate(chn, rotate_param, milli_sec)**
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。

输入说明	● **chn**：int，图片处理通道号。
	该参数的取值范围：[0, 128)，通道总数最多128。
	● **rotate_param**：dict，旋转信息字典。
	在**10.8.8.5.35 hi_rotate_param**字典内配置输入、输出图片信息，以及旋转参数。
	– 输入、输出图片分辨率的取值范围为：10*6~4096*8192。
	– 支持如下输入图片格式（输出图片格式与输入保持一致）：
	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_PACKED_444 = 11, // YUV444 Package 8bit
	HI_PIXEL_FORMAT_RGB_888 = 12, // RGB888
	HI_PIXEL_FORMAT_BGR_888 = 13, // BGR888
	HI_PIXEL_FORMAT_YVU_PACKED_444 = 58, // YVU444 Package 8bit
	HI_PIXEL_FORMAT_RGB_888_PLANAR = 69, // RGB888 Planar
	HI_PIXEL_FORMAT_BGR_888_PLANAR = 70, // BGR888 Planar
	● **milli_sec**：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	● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。 ● **task_id**：任务分配的ID，用来区分任务。

10.8.4.43 acl.himpi.vpc_set_chn_workspace

c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)**
Python函数	**ret = acl.himpi.vpc_set_chn_workspace(chn, workspace_param, param_cnt)**
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。

输入说明	● **chn**：int，图片处理通道号。 该参数的取值范围：[0, 128)，通道总数最多128。 ● **workspace_param**：list，元素类型为**10.8.8.5.16 hi_vpc_workspace_param**字典的数组。 此处需根据所使用的Remap功能或仿射变换功能来设置**10.8.8.5.16 hi_vpc_workspace_param**["func"]参数值。 若在一个应用程序中，涉及Remap、仿射变换两种功能，则可以在**10.8.8.5.16 hi_vpc_workspace_param**数组中包含2个元素，一个元素的hi_vpc_workspace_param["func"]设置为REMAP，一个元素的hi_vpc_workspace_param["func"]设置为AFFINE。 ● **param_cnt**：int，数组长度，需与workspace_param数组长度保持一致。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。
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。

10.8.4.44 acl.himpi.vpc_warp_affine

c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)**
Python函数	**task_id, ret = acl.himpi.vpc_warp_affine(chn, matrix, transform_param, milli_sec)**
函数功能	对输入图片做仿射变换。

输入说明	• **chn**：int，图片处理通道号。
	该参数的取值范围：[0, 128)，通道总数最多128。
	• **matrix**：dict，仿射变换需要的变换矩阵信息字典，具体参见**10.8.8.5.46 hi_transform_matrix**。使用**acl.himpi.vpc_get_affine_transform**或者**acl.himpi.vpc_get_rotation_matrix**接口获取的仿射变换矩阵信息字典。
	• **transform_param**：dict，仿射变换参数，在**10.8.8.5.21 hi_warp_transform_param**字典内配置输入、输出图片信息。
	– 输入、输出图片分辨率的取值范围为：10*6~4096*8192。
	– 支持如下输入图片格式：
	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_RGB_888 = 12, // RGB888
	HI_PIXEL_FORMAT_BGR_888 = 13, // BGR888
	HI_PIXEL_FORMAT_YUV_PACKED_444 = 11, // YUV444P 8bit
	HI_PIXEL_FORMAT_YVU_PACKED_444 = 58, // YVU444P 8bit
	– 支持如下输出图片格式：
	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_RGB_888 = 12, // RGB888
	HI_PIXEL_FORMAT_BGR_888 = 13, // BGR888
	HI_PIXEL_FORMAT_YUV_PACKED_444 = 11, // YUV444P 8bit
	HI_PIXEL_FORMAT_YVU_PACKED_444 = 58, // YVU444P 8bit
	HI_PIXEL_FORMAT_YUYV_PACKED_422 = 7, // YUV422P YUYV 8bit
	HI_PIXEL_FORMAT_UYVY_PACKED_422 = 8, // YUV422P UYVY 8bit
	HI_PIXEL_FORMAT_YVYU_PACKED_422 = 9, // YUV422P YVYU 8bit
	HI_PIXEL_FORMAT_VYUY_PACKED_422 = 10, // YUV422P VYUY 8bit
	• **milli_sec**：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	• **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.2 VPC图像处理返回码**。 • **task_id**：任务分配的ID，用来区分任务。
约束说明	• 在任务执行过程中，需要临时缓存，因此每个通道第一次执行本接口前需先调用**acl.himpi.vpc_set_chn_workspace**接口申请内部临时缓存。 • 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，在配置边界填充类型时，建议设置为“边界复制模式”，避免输出图片边缘异常数据。

10.8.5 VDEC 视频解码/JPEGD 图像解码接口

10.8.5.1 JPEGD 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

  JPEGD 422格式旋转那图片，若原图高为奇数，并且旋转方式为宽高对换，则旋转后宽存在绿边，建议用户将宽向下对齐到偶数使用（即去除绿边），比如原图为200*101，若旋转后为101*200，则建议用户实际使用区域为100*200。

  JPEGD 440格式旋转那图片，若原图宽为奇数，并且旋转方式为宽高对换，则旋转后高存在绿边，建议用户将高向下对齐到偶数使用（即去除绿边），比如原图为201*101，若旋转后为100*201，则建议用户实际使用区域为100*200。

- **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**acl.himpi.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**acl.himpi.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**acl.himpi.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**10.8.4.2 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

昇腾AI处理器	分辨率范围
Atlas 200/500 A2 推理产品	最大分辨率：16384*16384，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**acl.himpi.dvpp_malloc**接口申请Device上的输入、输出内存，调用**acl.himpi.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。

- 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下表中的计算公式。

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**10.8.2 基本概念**。

表 10-19 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		

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(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		
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	宽没有对齐要求 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。

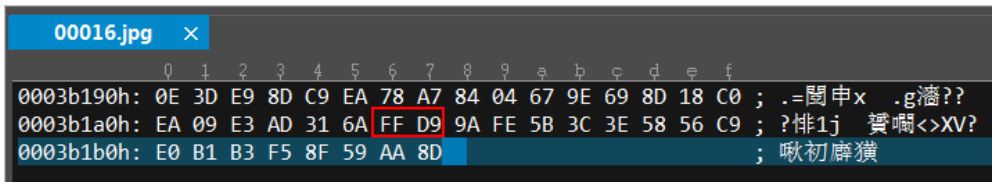
精度相关约束

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VPC在处理图片前，应首先对JPEGD对齐后的输出图片进行原分辨率裁剪，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。



10.8.5.2 JPEGD 性能指标数据

性能指标说明

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。

1080p指分辨率为1920*1080的图片；4K指分辨率为3840*2160的图片。单个Device的基本场景性能指标参考如下（1路对应一个通道，一个通道对应一个线程）：

场景举例	总帧率
1080p*n路（1≤n≤2）	n*256fps
1080p*n路（n>2）	512fps
4k*n路（1≤n≤2）	n*64fps
4k*n路（n>2）	128fps

10.8.5.3 VDEC 功能及约束说明

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：

  对于H264输入码流，最大分辨率8192*8192，最小分辨率128*128。

  对于H265输入码流，最大分辨率5400*8192，最小分辨率128*128。

- 输出图片分辨率：

  输出图片最大分辨率8192*8192；当输入码流宽度小于或等于4096时，输出图片最小分辨率为10*6，当输入码流宽度大于4096时，输出图片最小分辨率为128*128。

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**acl.himpi.dvpp_malloc**接口/**acl.himpi.dvpp_free**接口申请/释放内存或调用**acl.rt.malloc**/**acl.rt.free**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**acl.himpi.dvpp_malloc**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**acl.rt.malloc**接口申请内存。

- 输出内存。

  需调用**acl.himpi.dvpp_malloc**接口申请Device上的输出内存，调用**acl.himpi.dvpp_free**接口释放输出内存，内存大小请参见下表中的计算公式。

- 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。

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**10.8.2 基本概念**。

表 10-20 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
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	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
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	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

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。

10.8.5.4 VDEC 性能指标数据

性能指标说明

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。

单个Device的基本场景性能指标参考如下（1路对应一个通道，一个通道对应一个线程）：

场景举例	总帧率
720p*n路(1≤n≤4)	n*600fps
720p*n路(n>4)	2400fps
1080p*n路(1≤n≤4)	n*300fps
1080p*n路(n>4)	1200fps
4k*n路(1≤n≤4)	n*75fps
4k*n路(n>4)	300fps

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**acl.himpi.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**acl.himpi.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**acl.himpi.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**acl.himpi.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）

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）

10.8.5.5 acl.himpi.vdec_create_chn

c函数原型	**hi_s32 hi_mpi_vdec_create_chn(hi_vdec_chn chn, const hi_vdec_chn_attr *attr);**
Python函数	**ret = acl.himpi.vdec_create_chn(chn, attr)**
函数功能	根据设置的通道属性创建解码通道。
输入说明	● **chn**(hi_vdec_chn)：解码通道号 。 – 该参数的取值范围：[0, 128)，JPEGD功能和VDEC功能共用通道，且通道总数最多128。 ● **attr**(**hi_vdec_chn_attr**)：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	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.1 公共返回码**。

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**acl.himpi.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**acl.himpi.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。

10.8.5.6 acl.himpi.vdec_destroy_chn

c函数原型	**hi_s32 hi_mpi_vdec_destroy_chn(hi_vdec_chn chn);**
Python函数	**ret = acl.himpi.vdec_destroy_chn(chn)**
函数功能	销毁解码通道，以释放相关资源。
输入说明	**chn**：int，解码通道号。 • 该参数的取值范围：[0, 128)，JPEGD功能和VDEC功能共用通道，且通道总数最多128。
返回值说明	**ret**：int，错误码。 • 返回0表示成功。 • 返回非0表示失败，参见**10.8.9.3 VDEC视频/JPEGD图像解码返回码**.
注意事项	无

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”。

10.8.5.7 acl.himpi.vdec_get_chn_attr

c函数原型	**hi_s32 hi_mpi_vdec_get_chn_attr(hi_vdec_chn chn, hi_vdec_chn_attr *attr);**
Python函数	**attr, ret= acl.himpi.vdec_get_chn_attr(chn)**
函数功能	获取视频解码通道属性。
输入说明	**chn**：解码通道号。 ● 该参数的取值范围：[0, 128)，JPEGD功能和VDEC功能共用通道，且通道总数最多128。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.3 VDEC视频/JPEGD图像解码返回码**。 **attr**：解码通道属性字典。
注意事项	无
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”。
参考资源	无

10.8.5.8 acl.himpi.vdec_set_chn_attr

c函数原型	**hi_s32 hi_mpi_vdec_set_chn_attr(hi_vdec_chn chn, const hi_vdec_chn_attr *attr);**
Python函数	**ret = acl.himpi.vdec_set_chn_attr(chn, attr)**
函数功能	设置解码通道属性。 Atlas 200/500 A2推理产品，当前版本不支持该接口。
输入说明	● **chn**：解码通道号。 – 该参数的取值范围：[0, 128)，JPEGD功能和VDEC功能共用通道，且通道总数最多128。 ● **attr**：解码通道属性指针。

返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.3 VDEC视频/JPEGD图像解码返回码**。
注意事项	无
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。
参考资源	无

10.8.5.9 acl.himpi.vdec_start_recv_stream

c函数原型	**hi_s32 hi_mpi_vdec_start_recv_stream(hi_vdec_chn chn);**
Python函数	**ret = acl.himpi.vdec_start_recv_stream(chn)**
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。
输入说明	**chn**：解码通道号。 ● 该参数的取值范围：[0, 128)，JPEGD功能和VDEC功能共用通道，且通道总数最多128。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.3 VDEC视频/JPEGD图像解码返回码**。
注意事项	无
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**acl.himpi.vdec_send_stream**发送码流。 ● 重复调用启动接收码流接口时，返回成功。

10.8.5.10 acl.himpi.vdec_stop_recv_stream

c函数原型	**hi_s32 hi_mpi_vdec_stop_recv_stream(hi_vdec_chn chn);**
Python函数	**ret = acl.himpi.vdec_stop_recv_stream(chn)**

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。
输入说明	**chn**：解码通道号。 ● 该参数的取值范围：[0, 128)，JPEGD功能和VDEC功能共用通道，且通道总数最多128。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.3 VDEC视频/JPEGD图像解码返回码**.
注意事项	无
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，调用发送码流接口**acl.himpi.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。

10.8.5.11 acl.himpi.vdec_query_status

c函数原型	**hi_s32 hi_mpi_vdec_query_status(hi_vdec_chn chn, hi_vdec_chn_status *status);**
Python函数	**status, ret = acl.himpi.vdec_query_status(chn)**
函数功能	解码过程中通过此接口来查询解码通道状态。
输入说明	**chn**：解码通道号。 ● 该参数的取值范围：[0, 128)，JPEGD功能和VDEC功能共用通道，且通道总数最多128。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.3 VDEC视频/JPEGD图像解码返回码**。 **status**：获取的通道状态。
注意事项	无
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”。
参考资源	无

10.8.5.12 acl.himpi.vdec_reset_chn

c函数原型	**hi_s32 hi_mpi_vdec_reset_chn(hi_vdec_chn chn);**
Python函数	**ret = acl.himpi.vdec_reset_chn(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。
输入说明	**chn**：解码通道号。 ● 该参数的取值范围：[0, 128)，JPEGD功能和VDEC功能共用通道，且通道总数最多128。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.3 VDEC视频/JPEGD图像解码返回码**。
注意事项	无
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”。
参考资源	无

10.8.5.13 acl.himpi.vdec_set_chn_param

c函数原型	**hi_s32 hi_mpi_vdec_set_chn_param(hi_vdec_chn chn, const hi_vdec_chn_param *chn_param);**
Python函数	**ret = acl.himpi.vdec_set_chn_param(chn, 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。
输入说明	● **chn**：解码通道号。 – 该参数的取值范围：[0, 128)，JPEGD功能和VDEC功能共用通道，且通道总数最多128。 ● **chn_param**：解码通道参数指针。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.3 VDEC视频/JPEGD图像解码返回码**。
使用说明	无

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。 ● 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。

10.8.5.14 acl.himpi.vdec_get_chn_param

c函数原型	**hi_s32 hi_mpi_vdec_get_chn_param(hi_vdec_chn chn, hi_vdec_chn_param *chn_param);**
Python函数	**chn_param, ret = acl.himpi.vdec_get_chn_param(chn)**
函数功能	修改通道参数前可先调用此接口获取通道参数。
输入说明	**chn**：解码通道号。 ● 该参数的取值范围：[0, 128)，JPEGD功能和VDEC功能共用通道，且通道总数最多128。
返回值说明	**chn_param**：解码通道参数指针。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.3 VDEC视频/JPEGD图像解码返回码**。
使用说明	无
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”。

10.8.5.15 acl.himpi.vdec_set_protocol_param

c函数原型	**hi_s32 hi_mpi_vdec_set_protocol_param(hi_vdec_chn chn, const hi_vdec_protocol_param* protocol_param);**
Python函数	**ret = acl.himpi.vdec_set_protocol_param(chn, 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。 当前版本不支持该接口。

输入说明	● **chn**：解码通道号。 – 该参数的取值范围：[0, 128)，JPEGD功能和VDEC功能共用通道，且通道总数最多128。 ● **protocol_param**：解码通道参数指针。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.3 VDEC视频/JPEGD图像解码返回码**。
注意事项	无
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。
参考资源	无

10.8.5.16 acl.himpi.vdec_get_protocol_param

c函数原型	**hi_s32 hi_mpi_vdec_get_protocol_param(hi_vdec_chn chn, hi_vdec_protocol_param* protocol_param);**
Python函数	**protocol_param, ret = acl.himpi.vdec_get_protocol_param(chn)**
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。
输入说明	● **chn**：解码通道号。 – 该参数的取值范围：[0, 128)，JPEGD功能和VDEC功能共用通道，且通道总数最多128。
返回值说明	● **protocol_param**：解码通道参数指针。 ● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.3 VDEC视频/JPEGD图像解码返回码**。
注意事项	无
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”。

参考资源	无

10.8.5.17 acl.himpi.vdec_send_stream

c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);**
Python函数	**ret = acl.himpi.vdec_send_stream(chn, stream, vdec_pic_info, milli_sec)**
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。
输入说明	● **chn**：解码通道号。 – 该参数的取值范围：[0, 128)，JPEGD功能和VDEC功能共用通道，且通道总数最多128。 ● **stream**：输入码流信息。 该字典内的addr参数配置的地址为Device上的内存地址。 ● **vdec_pic_info**：输出图片信息字典。 ● **milli_sec**：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	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.3 VDEC视频/JPEGD图像解码返回码**。
注意事项	无

约束说明	• 发送数据前必须保证已经调用**acl.himpi.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**字典中将“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**字典中配置的输出内存中，此时允许将输出内存地址配置为“None”。**acl.himpi.vdec_get_frame**接口仍然能获取到字典**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“None”，则无需释放）。 • 解码时，输入输出内存均需要在调用**acl.himpi.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**10.8.5.3 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**10.8.6.1 JPEGE功能及约束说明**。

10.8.5.18 acl.himpi.vdec_get_frame

c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);**
Python函数	**frame_info, supplement, stream, ret = acl.himpi.vdec_get_frame(chn, milli_sec)**
函数功能	解码后，获取视频解码通道的解码图像及输入Stream。

输入说明	• **chn**：解码通道号。 – 该参数的取值范围：[0, 128)，JPEGD功能和VDEC功能共用通道，且通道总数最多128。 • **milli_sec**：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	• **ret**：int，错误码。 – 返回0表示成功。但有可能是解码失败的数据，解码成功或失败，需要查看**hi_video_frame**字典（**hi_video_frame_info**字典的成员）中的“frame_flag”参数。 – 返回非0表示失败，参见**10.8.9.3 VDEC视频/JPEGD图像解码返回码** • **stream**：获取已完成解码的输入码流信息指针。 • **frame_info**：获取已经完成解码的图像信息。解码后的数据存放在Device内存中。 • **supplement**：获取的解码图像补充信息。预留参数，暂未使用。
注意事项	无
约束说明	• 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_VDEC_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_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，会立刻返回错误码“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图像则会返回错误码 “HI_ERR_VDEC_BUF_EMPTY”。 • 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**acl.himpi.vdec_set_jpegd_precision_mode**接口。 • 调用acl.himpi.vdec_get_frame接口且返回成功后，需调用**acl.himpi.vdec_release_frame**接口释放解码相关接口。

10.8.5.19 acl.himpi.vdec_release_frame

c函数原型	**hi_s32 hi_mpi_vdec_release_frame(hi_vdec_chn chn, const hi_video_frame_info *frame_info);**
Python函数	**ret = acl.himpi.vdec_release_frame(chn, frame_info)**
函数功能	解码之后，释放资源。
输入说明	● **chn**：解码通道号。 – 该参数的取值范围：[0, 128)，JPEGD功能和VDEC功能共用通道，且通道总数最多128。 ● **frame_info**：解码后的图像信息指针，调用**10.8.5.18 acl.himpi.vdec_get_frame**接口获取。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.3 VDEC视频/JPEGD图像解码返回码**
注意事项	无
约束说明	● 此接口需与**acl.himpi.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**acl.himpi.vdec_get_frame**接口从该通道获取的数据，不得对数据信息字典进行任何修改。

10.8.5.20 acl.himpi.vdec_get_fd

c函数原型	**hi_s32 hi_mpi_vdec_get_fd(hi_vdec_chn chn);**
Python函数	**ret = acl.himpi.vdec_get_fd(chn)**
函数功能	获取视频解码通道的设备文件句柄。
输入说明	**chn**：解码通道号。 ● 该参数的取值范围：[0, 128)，JPEGD功能和VDEC功能共用通道，且通道总数最多128。
返回值说明	**ret**：int，错误码。 ● 0或正数值：有效返回值。 ● 非正数值：无效返回值。
注意事项	无
约束说明	在进程退出时会自动关闭文件句柄，回收资源。
参考资源	无

10.8.5.21 acl.himpi.vdec_close_fd

c函数原型	**hi_s32 hi_mpi_vdec_close_fd(hi_vdec_chn chn);**
Python函数	**ret = acl.himpi.vdec_close_fd(chn)**
函数功能	关闭视频解码的设备文件句柄。 当前版本不支持该接口。
输入说明	**chn**：解码通道号。 ● 该参数的取值范围：[0, 128)，JPEGD功能和VDEC功能共用通道，且通道总数最多128。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.3 VDEC视频/JPEGD图像解码返回码**。
注意事项	无
约束说明	无
参考资源	无

10.8.5.22 acl.himpi.vdec_get_pic_buf_size

c函数原型	**hi_u32 hi_vdec_get_pic_buf_size(hi_payload_type type, hi_pic_buf_attr *buf_attr);**
Python函数	**ret = acl.himpi.vdec_get_pic_buf_size(type, buf_attr)**
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。
输入说明	● **type**：解码视频类型，当前支持HI_PT_H264、HI_PT_H265、HI_PT_JPEG。 ● **buf_attr**：图片参数字典。
返回值说明	**ret**：图像存储需要的图像大小。 ● 非0为需要存储的图像大小。 ● 若解码视频类型为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”。
注意事项	无
约束说明	提供给用户来计算输出图片缓冲区大小。

10.8.5.23 acl.himpi.vdec_get_tmv_buf_size

c函数原型	**hi_u32 hi_vdec_get_tmv_buf_size(hi_payload_type type, hi_u32 width, hi_u32 height);**
Python函数	**ret = acl.himpi.vdec_get_tmv_buf_size(type, width, height)**
函数功能	取解码图像需要的矢量预测缓冲区的大小。
输入说明	• **type**：解码视频类型。 • **width**：解码图像宽。 • **height**：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。
注意事项	无
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。

10.8.5.24 acl.himpi.vdec_set_jpegd_precision_mode

c函数原型	**hi_u32 hi_mpi_vdec_set_jpegd_precision_mode(hi_vdec_chn chn, const hi_jpegd_precision_mode mode)**
Python函数	**ret = acl.himpi.vdec_set_jpegd_precision_mode(chn, mode)**
函数功能	设置JPEGD解码输出图片的宽、高对齐模式。
输入说明	• **chn**：解码通道号。 – 该参数的取值范围：[0, 128)，JPEGD功能和VDEC功能共用通道，且通道总数最多128。 • **mode**：设置JPEGD解码输出图片的宽、高对齐模式。
返回值说明	**ret**：int，错误码。 • 0：成功 • 非0：失败，参见**10.8.9.3 VDEC视频/JPEGD图像解码返回码**
注意事项	无
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**acl.himpi.vdec_start_recv_stream**接口之前调用本接口设置对齐模式，在调用**acl.himpi.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。

10.8.5.25 acl.himpi.vdec_get_jpegd_output_info

c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)**
Python函数	**img_info, ret = acl.himpi.vdec_get_jpegd_output_info(stream, output_format)**
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**10.8.9.1 公共返回码**。
输入说明	• **stream**：dict，输入码流信息字典。具体参见**10.8.8.3.17 hi_vdec_stream**。 RC模式，该字典内的addr参数配置的地址为Device上的内存地址。 • **output_format**：int，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 = 0**。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**。
返回值说明	• **img_info**：dict，输出图片信息字典，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pixel_format参数返回。具体参见**10.8.8.2.14 hi_img_info**。 • **ret**：int，错误码。 – 0：成功。 – 非0：失败，参见**10.8.9.3 VDEC视频/JPEGD图像解码返回码**。

10.8.6 VENC 视频/JPEGE 图像编码接口

10.8.6.1 JPEGE 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。

图片格式、宽高对齐、内存约束

实现JPEGE图片编码功能时：

- 输入内存

需调用**acl.himpi.dvpp_malloc**接口申请Device上的输入、输出内存，调用**acl.himpi.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。内存大小请参见下表中的计算公式。

- 输出内存

  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**acl.himpi.venc_send_frame**接口发送原始图像进行图像编码。

    在调用**acl.himpi.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**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**acl.himpi.venc_send_jpege_frame**接口发送原始图像进行图像编码。

    在调用**acl.himpi.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**hi_venc_attr**），然后调用**acl.himpi.venc_get_jpege_predicted_size**接口预估输出内存大小，调用**acl.himpi.dvpp_malloc**/**acl.himpi.dvpp_free**接口申请/释放输出内存。

    该方式下，直接在调用**acl.himpi.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。**

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**10.8.2 基本概念**。

表 10-21 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			

10.8.6.2 JPEGE 性能指标说明

性能指标说明

1080p指分辨率为1920*1080的图片；4K指分辨率为3840*2160的图片。单个Device的基本场景性能指标参考如下（1路对应一个通道，一个通道对应一个线程）：

场景举例（输入图片格式：YUV420 8bit）	总帧率
1080p*n路	256fps
4k*n路	64fps

10.8.6.3 VENC 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。

分辨率约束

- 输入图片分辨率：
  最大分辨率8192*8192，最小分辨率114*114
- 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**acl.himpi.dvpp_malloc**接口申请Device上的输入内存，调用**acl.himpi.dvpp_free**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**hi_pixel_format**，宽stride、高stride等概念请参见**10.8.2 基本概念**。

表 10-22 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stride * 3/2	• H264 BP/MP/HP • H265 MP（仅支持Slice码流）
YUV420SP NV21 8bit			

10.8.6.4 VENC 性能指标说明

性能指标说明

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。

单个Device的基本场景性能指标参考如下（1路对应一个通道，一个通道对应一个线程）：

场景举例	总帧率
720p*n路(1≤n≤3)	n*400fps
720p*n路(n>3)	1200fps
1080p*n路(1≤n≤3)	n*200fps
1080p*n路(n>3)	600fps
4k*n路(1≤n≤3)	n*50fps
4k*n路(n>3)	150fps

注：其它分辨率可以等量估算。

10.8.6.5 acl.himpi.venc_create_chn

c函数原型	**hi_s32 hi_mpi_venc_create_chn(hi_venc_chn chn, const hi_venc_chn_attr *attr);**
Python函数	**ret = acl.himpi.venc_create_chn(chn, attr)**
函数功能	创建编码通道。
输入说明	• **chn**：编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。 • **attr**：编码通道属性字典。
返回值说明	**ret**：int，错误码。 • 返回0表示成功。 • 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
注意事项	无

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10-23**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：3840x2160(4k)、1920x1080(1080P)、1280x720(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。

参考信息

表 10-23 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 x max_pic_height x 3/4。 • 当 mini_buf_mode=1时，Buffer≥32*1024 byte。	Frame/Slice	Baseline Mainprofile Highprofile
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 x 3/4。 • mini_buf_mode=1时，Buffer≥32*1024 byte。	Frame/Slice	Main profile Main 10 profile（暂不支持）

编码协议	编码方式	码流 buffer 深度	获取码流模式	编码 profile
		• 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）

10.8.6.6 acl.himpi.venc_destroy_chn

c函数原型	**hi_s32 hi_mpi_venc_destroy_chn(hi_venc_chn chn);**
Python函数	**ret = acl.himpi.venc_destroy_chn(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。
输入说明	**chn**：编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。
返回值说明	**ret**：int，错误码。 • 返回0表示成功。 • 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
注意事项	无
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。

10.8.6.7 acl.himpi.venc_start_chn

c函数原型	**hi_s32 hi_mpi_venc_start_chn (hi_venc_chn chn, const hi_venc_start_param *recv_param);**
Python函数	**ret = acl.himpi.venc_start_chn(chn, recv_param)**
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。

输入说明	● **chn**：编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。 ● **recv_param**：接收图像参数字典，用于指定需要接收的图像帧数。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
注意事项	无
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。

10.8.6.8 acl.himpi.venc_stop_chn

c函数原型	**hi_s32 hi_mpi_venc_stop_chn(hi_venc_chn chn);**
Python函数	**ret= acl.himpi.venc_stop_chn(chn)**
函数功能	停止编码通道接收输入图片。
输入说明	**chn**：编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
注意事项	无
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。

10.8.6.9 acl.himpi.venc_query_status

c函数原型	**hi_s32 hi_mpi_venc_query_status(hi_venc_chn chn, hi_venc_chn_status *status);**
Python函数	**status, ret = acl.himpi.venc_query_status(chn)**
函数功能	查询编码通道状态。
输入说明	**chn**：编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。
返回值说明	● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**. ● **status**：编码通道的状态。
使用说明	无
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： – 在编码通道状态字典中，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。 – 在编码通道状态字典中，left_stream_bytes表示码流 buffer 中剩余的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。 – 在编码通道状态字典中，left_stream_frames表示码流 buffer 中剩余的帧数目。 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。 – 在编码通道状态字典中，cur_packs表示当前帧的码流包个数。在调用**acl.himpi.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。 – 在编码通道状态字典中，left_recv_pics表示调用**acl.himpi.venc_start_chn**接口后剩余等待接收的帧数目。 – 在编码通道状态字典中，left_enc_pics表示调用**acl.himpi.venc_start_chn**接口后剩余等待编码的帧数目。 – 如果调用**acl.himpi.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。

10.8.6.10 acl.himpi.venc_get_stream

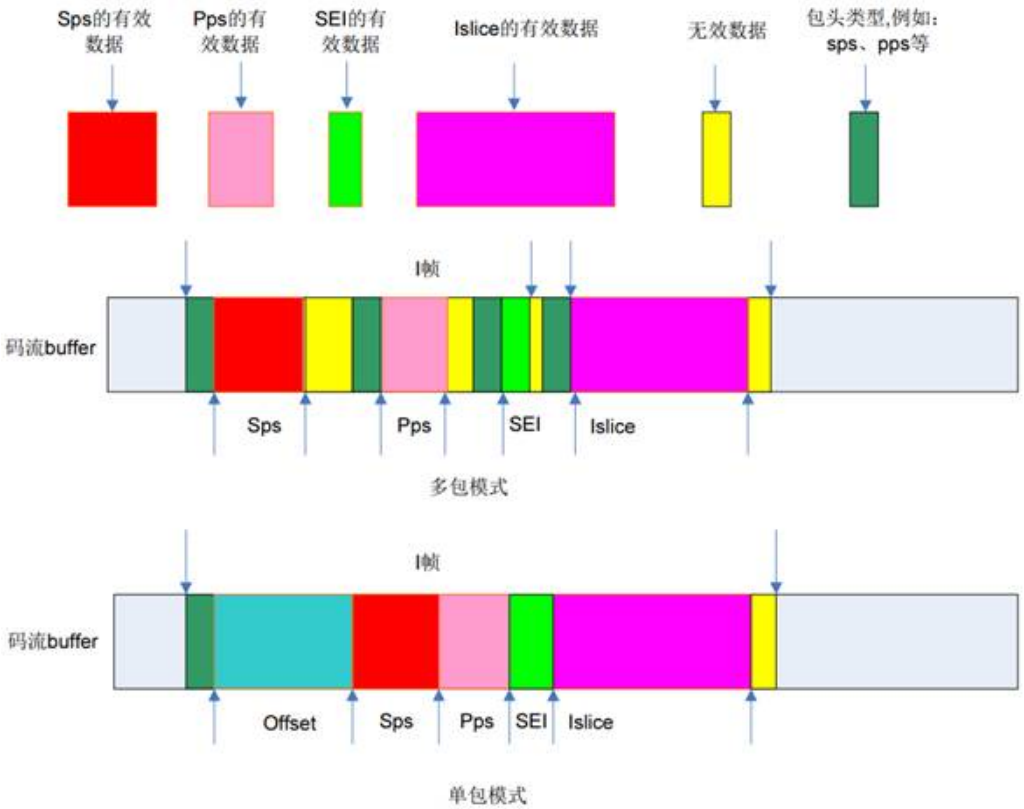
c函数原型	**hi_s32 hi_mpi_venc_get_stream(hi_venc_chn chn, hi_venc_stream *stream, hi_s32 milli_sec);**
Python函数	**stream, ret = acl.himpi.venc_get_stream(chn, stream, milli_sec)**
函数功能	获取编码的码流。
输入说明	● **chn**：编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。 ● **stream**：pack_cnt传入cur_packs个包信息字典，自动分配内存。 ● **milli_sec**：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	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**. **stream**：码流字典。该字典内的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。
注意事项	无

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**acl.himpi.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**acl.himpi.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**acl.himpi.venc_get_stream**获取编码码流； – 调用**acl.himpi.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10-10**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**acl.himpi.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。

其他参考信息

图 10-10 码流包结构图



10.8.6.11 acl.himpi.venc_release_stream

c函数原型	**hi_s32 hi_mpi_venc_release_stream(hi_venc_chn chn, hi_venc_stream *stream);**
Python函数	**ret = acl.himpi.venc_release_stream(chn, stream)**
函数功能	释放码流缓存。
输入说明	• **chn**：编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。 • **stream**：码流字典。如果 stream为空，则返回错误码“HI_ERR_VENC_NULL_PTR”。
返回值说明	**ret**：int，错误码。 • 返回0表示成功。 • 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
注意事项	无
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 **acl.himpi.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”。

10.8.6.12 acl.himpi.venc_send_frame

c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)**
Python函数	**ret = acl.himpi.venc_send_frame(chn, frame, milli_sec)**
函数功能	发送原始图像进行视频或图像编码。

输入说明	● **chn**：编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。 ● **frame**：原始图像信息字典。 ● **milli_sec**：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	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
使用说明	无
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，当数据格式为YUV420SP（NV12，NV21）时，图像高度必须为2对齐。 ● 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 ● 视频编码，对输入、输出的约束请参见**10.8.6.3 VENC功能及约束说明**。 ● 图像编码，对输入、输出的约束请参见**10.8.6.1 JPEGE功能及约束说明**。

10.8.6.13 acl.himpi.venc_set_mod_param

c函数原型	**hi_s32 hi_mpi_venc_set_mod_param(const hi_venc_mod_param *mod_param)**
Python函数	**ret = acl.himpi.venc_set_mod_param(mod_param)**
函数功能	设置编码相关的模块参数。
输入说明	**mod_param**：编码模块参数指针。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
注意事项	无

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”。 ● 可以设置编码相关模块的参数，例如venc、h264e、h265e、jpege等。

10.8.6.14 acl.himpi.venc_get_mod_param

c函数原型	**hi_s32 hi_mpi_venc_get_mod_param(hi_venc_mod_param *mod_param)**
Python函数	**mod_param, ret = acl.himpi.venc_get_mod_param(mod_param)**
函数功能	获取编码相关的模块参数。
输入说明	**mod_param**：编码模块类型。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。 **mod_param**：编码模块参数指针。
注意事项	无
约束说明	可以获取编码相关模块的参数，例如VENC、H264E、H265E、JPEGE等。

10.8.6.15 acl.himpi.venc_request_idr

c函数原型	**hi_s32 hi_mpi_venc_request_idr(hi_venc_chn chn, hi_bool instant);**
Python函数	**ret = acl.himpi.venc_request_idr(chn, instant)**
函数功能	请求IDR帧。
输入说明	● **chn**：编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。 ● **instant**：是否使能立即编码IDR帧。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。

使用说明	无
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B帧模式）下，请求IDR帧会延时生效。
参考资源	无

10.8.6.16 acl.himpi.venc_get_fd

c函数原型	**hi_s32 hi_mpi_venc_get_fd(hi_venc_chn chn);**
Python函数	**ret = acl.himpi.venc_get_fd(chn)**
函数功能	获取编码通道对应的文件句柄。
输入说明	**chn**：编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。
返回值说明	**ret**：int，编码通道对应的文件句柄。 ● 0或正数值：有效返回值。 ● 非正数值：无效返回值。
使用说明	无
约束说明	在进程退出时会自动关闭文件句柄，回收资源。

10.8.6.17 acl.himpi.venc_close_fd

c函数原型	**hi_s32 hi_mpi_venc_close_fd(hi_venc_chn chn);**
Python函数	**ret = acl.himpi.venc_close_fd(chn)**
函数功能	关闭编码通道对应的设备文件句柄。 当前版本不支持该接口。

输入说明	**chn**：编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。
返回值说明	**ret**：int，错误码。 ● 0：关闭成功。 ● -1：关闭失败。
注意事项	无
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**acl.himpi.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。
参考资源	无

10.8.6.18 acl.himpi.venc_set_jpeg_param

c函数原型	**hi_s32 hi_mpi_venc_set_jpeg_param(hi_venc_chn chn, const hi_venc_jpeg_param *jpeg_param)**
Python函数	**ret = acl.himpi.venc_set_jpeg_param(chn, 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。
输入说明	● **chn**：编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。 ● **jpeg_param**：JPEG协议编码通道的高级参数集合。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
注意事项	无
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**acl.himpi.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**字典内的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。
参考资源	无

10.8.6.19 acl.himpi.venc_get_jpeg_param

c函数原型	**hi_s32 hi_mpi_venc_get_jpeg_param(hi_venc_chn chn, hi_venc_jpeg_param *jpeg_param)**
Python函数	**jpeg_param, ret = acl.himpi.venc_get_jpeg_param(chn)**
函数功能	获取JPEG协议编码通道的量化表高级配置参数。
输入说明	**chn**：编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。
返回值说明	● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。 ● **jpeg_param**：JPEG协议编码通道的高级参数集合。
注意事项	无
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。 用户在调用**acl.himpi.venc_set_jpeg_param**前，可以调用该接口获取当前JPEGE的量化表参数。
参考资源	无

10.8.6.20 acl.himpi.venc_set_chn_param

c函数原型	**hi_s32 hi_mpi_venc_set_chn_param(hi_venc_chn chn, const hi_venc_chn_param *chn_param);**
Python函数	**ret = acl.himpi.venc_set_chn_param(chn, chn_param)**
函数功能	设置通道参数。
输入说明	● **chn**：编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。 ● **chn_param**：VENC的通道参数。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
注意事项	无

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。
参考资源	无

10.8.6.21 acl.himpi.venc_set_scene_mode

c函数原型	**hi_s32 hi_mpi_venc_set_scene_mode(hi_venc_chn chn, const hi_venc_scene_mode scene_mode);**
Python函数	**ret = acl.himpi.venc_set_scene_mode(chn, scene_mode)**
函数功能	设置场景模式。
输入说明	• **chn**：编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。 • **scene_mode**：场景模式。目前只可以配置为以下值： 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
返回值说明	**ret**：int，错误码。 • 返回0表示成功。 • 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
注意事项	无
约束说明	• 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。 • 本接口在编码通道创建之后、编码通道销毁之前调用。
参考资源	无

10.8.6.22 acl.himpi.venc_get_scene_mode

c函数原型	**hi_s32 hi_mpi_venc_get_scene_mode(hi_venc_chn chn, hi_venc_scene_mode *scene_mode)**
Python函数	**scene_mode, ret = acl.himpi.venc_get_scene_mode(chn)**
函数功能	获取场景模式。
输入说明	**chn**：编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。

返回值说明	● **scene_mode**：int，当前场景模式。详情请参见**10.8.8.4.66 hi_venc_scene_mode**。 ● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
注意事项	无
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。
参考资源	无

10.8.6.23 acl.himpi.venc_set_rc_param

c函数原型	**hi_s32 hi_mpi_venc_set_rc_param(hi_venc_chn chn, const hi_venc_rc_param *rc_param);**
Python函数	**ret = acl.himpi.venc_set_rc_param(chn，rc_param)**
函数功能	设置编码通道码率控制器的高级参数的值。
输入说明	● **chn**：编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。 ● **rc_param**：编码通道码率控制器的高级参数的指针地址。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
注意事项	无
约束说明	● 本接口在编码通道创建之后、编码通道销毁之前设置。 ● 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**acl.himpi.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。
参考资源	无

10.8.6.24 acl.himpi.venc_get_rc_param

c函数原型	**hi_s32 hi_mpi_venc_get_rc_param(hi_venc_chn chn, hi_venc_rc_param *rc_param);**

Python函数	**rc_param, ret = acl.himpi.venc_get_rc_param(chn, mode)**
函数功能	获取编码通道码率控制器的高级参数的值。
输入说明	• **chn**：编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。 • **mode**: 编码通道码率控制器模式。
返回值说明	**ret**：int，错误码。 • 返回0表示成功。 • 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。 **rc_param**：编码通道码率控制器的高级参数指针。
注意事项	无
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。
参考资源	无

10.8.6.25 acl.himpi.venc_set_jpeg_huffman_param

c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);**
Python函数	**ret = acl.himpi.venc_set_jpeg_huffman_param(chn, jpeg_huffman_param)**
函数功能	设置编码通道的JPEGE huffman编码高级参数。
输入说明	• **chn**：编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。 • **jpeg_huffman_param**：huffman编码参数指针。
返回值说明	**ret**：int，错误码。 • 返回0表示成功。 • 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
注意事项	无
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。
参考资源	无

10.8.6.26 acl.himpi.venc_get_jpeg_huffman_param

c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);**
Python函数	**jpeg_huffman_data, ret = acl.himpi.venc_get_jpeg_huffman_param(chn)**
函数功能	获取编码通道的JPEGE huffman编码高级参数。
输入说明	**chn**：编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。 **jpeg_huffman_data**：huffman高级参数指针。
使用说明	无
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**acl.himpi.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。
参考资源	无

10.8.6.27 acl.himpi.venc_compact_jpeg_tables

c函数原型	**hi_s32 hi_mpi_venc_compact_jpeg_tables(hi_venc_chn chn, hi_u32 table_type, hi_bool enable);**
Python函数	**ret = acl.himpi.venc_compact_jpeg_tables(chn, table_type, enable)**
函数功能	配置是否对高级参数huffman或QT表进行压缩。
输入说明	● **chn**：编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。 ● **table_type**：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**： 是否压缩，0表示不压缩，1表示压缩。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
注意事项	无

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。
参考资源	无

10.8.6.28 acl.himpi.venc_send_jpege_frame

c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)**
Python函数	**ret = acl.himpi.venc_send_jpege_frame(chn, frame, jpege_stream, milli_sec)**
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。
输入说明	• **chn**：编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。 • **frame(hi_video_frame_info)**：原始图像信息的指针。 • **jpege_stream(hi_img_stream)**：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**acl.himpi.venc_get_jpege_predicted_size**获取，再可调用**acl.himpi.dvpp_malloc**接口申请输出内存。 • **milli_sec**：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	**ret**：int，错误码。 • 返回0表示成功。 • 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
注意事项	无

约束说明	● 在调用10.8.6.5 acl.himpi.venc_create_chn接口创建通道时，将参数hi_venc_chn_attr.get("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。 ● 图像编码，对输入、输出的约束请参见10.8.6.1 JPEGE功能及约束说明。

10.8.6.29 acl.himpi.venc_get_jpege_predicted_size

c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)**
Python函数	**size，ret = acl.himpi.venc_get_jpege_predicted_size(frame, jpeg_param)**
函数功能	预估JPEGE编码一帧图片所需的输出缓冲区大小。
输入说明	**frame(hi_video_frame_info)**：int，原始图像信息的指针地址。 **jpeg_param(hi_venc_jpeg_param)**：JPEG协议编码通道的高级参数集合的指针，预留参数。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见10.8.9.4 VENC视频/JPEGE图像编码返回码。 **size**：输出缓冲区大小的指针，单位为Byte。

10.8.6.30 acl.himpi.venc_get_cu_pred

c函数原型	**hi_s32 hi_mpi_venc_get_cu_pred(hi_venc_chn chn, hi_venc_cu_prediction *cu_pred)**
Python函数	**cu_pred, ret = acl.himpi.venc_get_cu_pred(chn)**
函数功能	获取通道CU/MB的模式选择的倾向性设置。
输入说明	**chn**：int，编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。

返回值说明	**cu_pred**：dict，CU模式选择的倾向性字典。具体参见**10.8.8.4.94 hi_venc_cu_prediction**。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
注意事项	无
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。
参考资源	无

10.8.6.31 acl.himpi.venc_set_cu_pred

c函数原型	**hi_s32 hi_mpi_venc_set_cu_pred(hi_venc_chn chn, const hi_venc_cu_prediction *cu_pred)**
Python函数	**ret = acl.himpi.venc_set_cu_pred(chn, 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。
输入说明	● **chn**：int，编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。 ● **cu_pred**：CU模式选择的倾向性参数字典。具体参见**10.8.8.4.94 hi_venc_cu_prediction**。 该字典内需包含以下参数： – 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	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
注意事项	无
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**acl.himpi.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。
参考资源	无

10.8.6.32 acl.himpi.venc_get_h264_vui

c函数原型	**hi_s32 hi_mpi_venc_get_h264_vui(hi_venc_chn chn, hi_venc_h264_vui *h264_vui)**
Python函数	**h264_vui, ret = acl.himpi.venc_get_h264_vui(chn)**
函数功能	获取H.264协议编码通道的Vui配置。
输入说明	**chn**：int，编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。
返回值说明	**h264_vui**：dict，H.264协议编码通道的Vui属性。具体请参见**10.8.8.4.87 hi_venc_h264_vui**。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
注意事项	无
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。

参考资源	无

10.8.6.33 acl.himpi.venc_set_h264_vui

c函数原型	**hi_s32 hi_mpi_venc_set_h264_vui(hi_venc_chn chn, const hi_venc_h264_vui *h264_vui)**
Python函数	**ret = acl.himpi.venc_set_h264_vui(chn, 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，不影响视频解码过程，但会建议解码器做一些矫正处理。
输入说明	● **chn**：int, 编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。 ● **h264_vui**：dict，H.264协议编码通道的Vui参数字典。具体请参见**10.8.8.4.87 hi_venc_h264_vui**。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
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acl.himpi.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。

10.8.6.34 acl.himpi.venc_get_h265_vui

c函数原型	**hi_s32 hi_mpi_venc_get_h265_vui(hi_venc_chn chn, hi_venc_h265_vui *h265_vui)**
Python函数	**h265_vui, ret = acl.himpi.venc_get_h265_vui(chn)**

函数功能	获取H.265协议编码通道的Vui配置。
输入说明	**chn**：int，编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。
返回值说明	● **h265_vui**：dict，H.265协议编码通道的Vui属性字典。参见**10.8.8.4.88 hi_venc_h265_vui**。 ● **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
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。

10.8.6.35 acl.himpi.venc_set_h265_vui

c函数原型	**hi_s32 hi_mpi_venc_set_h265_vui(hi_venc_chn chn, const hi_venc_h265_vui *h265_vui)**
Python函数	**ret = acl.himpi.venc_set_h265_vui(chn, 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，不影响视频解码过程，但会建议解码器做一些矫正处理。
输入说明	● **chn**：int，编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。 ● **h265_vui**：dict，H.265协议编码通道的Vui属性字典。参见**10.8.8.4.88 hi_venc_h265_vui**。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。

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**acl.himpi.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。

10.8.6.36 acl.himpi.venc_get_intra_refresh

c函数原型	**hi_s32 hi_mpi_venc_get_intra_refresh(hi_venc_chn chn, hi_venc_intra_refresh *intra_refresh)**
Python函数	**intra_refresh, ret = acl.himpi.venc_get_intra_refresh(chn)**
函数功能	获取P帧刷新Islice的设置参数。
输入说明	**chn**：int，编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。
返回值说明	• **intra_refresh**：dict，刷Islice的参数字典。参见**10.8.8.4.95 hi_venc_intra_refresh**。 • **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
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。

10.8.6.37 acl.himpi.venc_set_intra_refresh

c函数原型	**hi_s32 hi_mpi_venc_set_intra_refresh(hi_venc_chn chn, const hi_venc_intra_refresh *intra_refresh);**
Python函数	**ret = acl.himpi.venc_set_intra_refresh(chn, intra_refresh)**
函数功能	设置P帧刷新Islice的参数。

输入说明	• **chn**：int，编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。 • **intra_refresh**：dict，刷Islice的参数字典。参见**10.8.8.4.95 hi_venc_intra_refresh**。
返回值说明	**ret**：int，错误码。 • 返回0表示成功。 • 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
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**acl.himpi.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**acl.himpi.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。

10.8.6.38 acl.himpi.venc_get_ref_param

c函数原型	**hi_s32 hi_mpi_venc_get_ref_param(hi_venc_chn chn, const hi_venc_ref_param *ref_param)**
Python函数	**ref_param, ret = acl.himpi.venc_get_ref_param(chn)**
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**。
输入说明	**chn**：int，编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。
返回值说明	• **ref_param**：dict，H.264/H.265编码通道高级跳帧参考参数字典。参见**10.8.8.4.86 hi_venc_ref_param**。 • **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
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。

10.8.6.39 acl.himpi.venc_set_ref_param

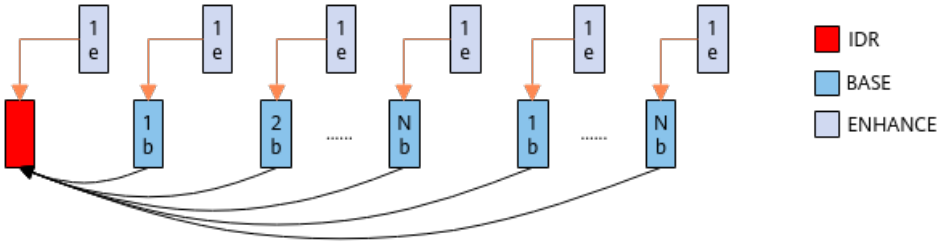
c函数原型	**hi_s32 hi_mpi_venc_set_ref_param(hi_venc_chn chn, const hi_venc_ref_param *ref_param)**
Python函数	**ret = acl.himpi.venc_set_ref_param(chn, ref_param)**
函数功能	设置H.264/H.265编码通道高级跳帧参考参数，用于改变帧间参考关系。关于高级跳帧模式的详细说明请参见**高级跳帧模式说明**。
输入说明	• **chn**：int，编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。 • **ref_param**：dict，H.264/H.265编码通道高级跳帧参考参数字典。参见**10.8.8.4.86 hi_venc_ref_param**。
返回值说明	**ret**：int，错误码。 • 返回0表示成功。 • 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
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 。

高级跳帧模式说明

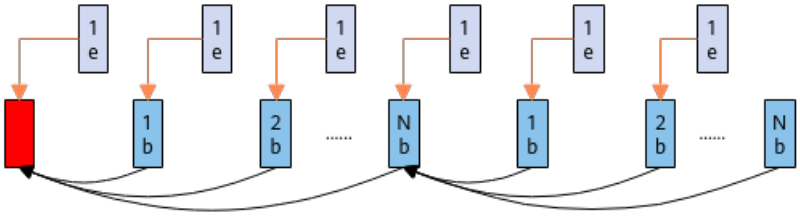
高级跳帧参考模式涉及3 个参数：base、enhance 和pred_en，其含义请参考hi_venc_ref_param。高级跳帧参考模式示意图如下图所示。

- 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。

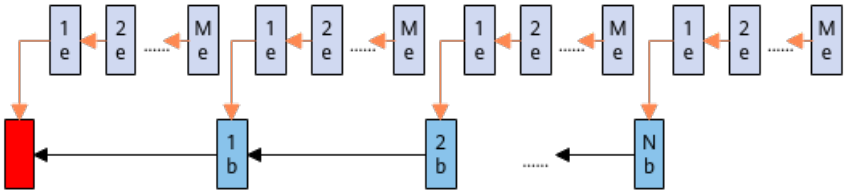
- normal_p高级跳帧参考示意图



(base = N, enhance = 1, pred_en = HI_FALSE)

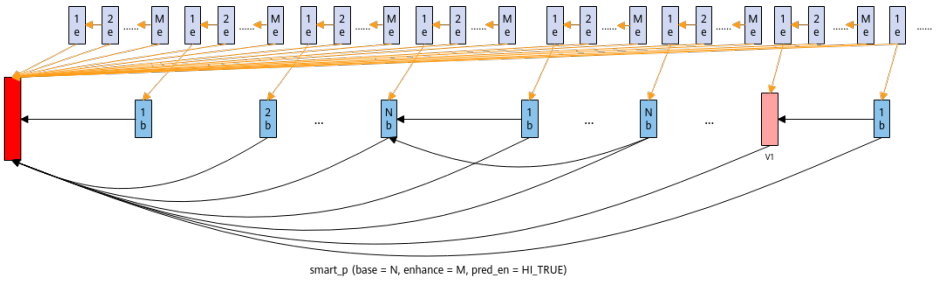


(base = N, enhance = 1, pred_en = HI_TRUE)

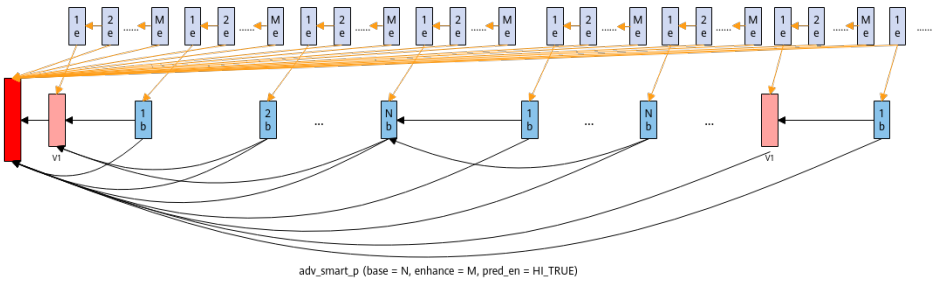


(base = 1, enhance = M, pred_en = HI_TRUE)

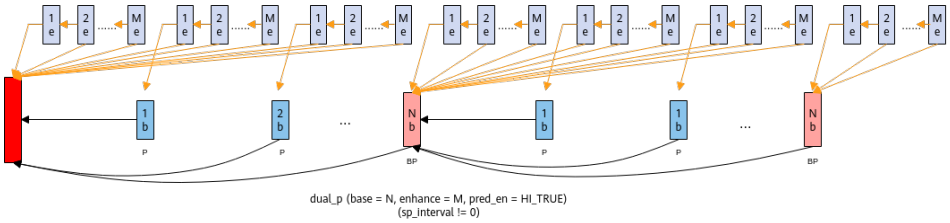
- smart_p高级跳帧参考示意图



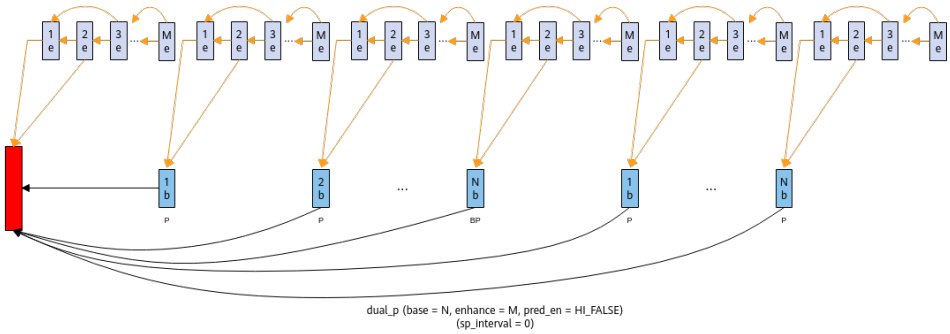
- adv_smart_p高级跳帧参考示意图



- 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



10.8.6.40 acl.himpi.venc_get_roi_attr

c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)**
Python函数	**roi_attr, ret = acl.himpi.venc_get_roi_attr(chn, idx)**
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**。
输入说明	• **chn**：int，编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。 • **idx**：int，H.264/H.265协议编码通道ROI区域索引。
返回值说明	• **roi_attr**：dict，ROI区域参数字典。参见**10.8.8.4.85 hi_venc_roi_attr**。 • **ret**：int，错误码。 – 返回0表示成功。 – 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
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。

10.8.6.41 acl.himpi.venc_set_roi_attr

c函数原型	**hi_s32 hi_mpi_venc_set_roi_attr(hi_venc_chn chn, const hi_venc_roi_attr *roi_attr)**
Python函数	**ret = acl.himpi.venc_set_roi_attr(chn, 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**。
输入说明	**chn**：int，编码通道号。 编码通道号的取值范围：[0, 128)，JPEGE功能和VENC功能共用通道，且通道总数最多128。 **roi_attr**：dict，ROI区域参数字典。参见**10.8.8.4.85 hi_venc_roi_attr**。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.4 VENC视频/JPEGE图像编码返回码**。
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**acl.himpi.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。 ● 设置该接口后，可能会出现感兴趣区域QP值不全为设置值的现象；这是因为在编码过程中，有些编码块的残差会为0，没有数据需要被量化，QP值不生效所致，属于正常现象，不影响编码效果。

ROI 编码功能说明

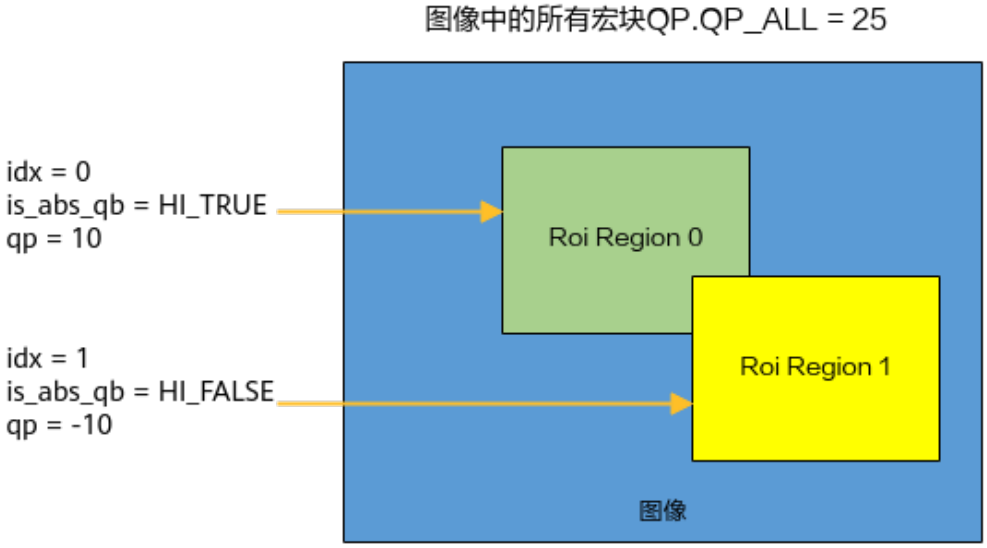
ROI（Region Of Interest）编码：对感兴趣区域编码。

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。



10.8.7 PNGD 图像处理接口

10.8.7.1 功能及约束说明

功能说明

PNGD（PNG Decoder）功能：实现PNG格式图片的硬件解码。

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

输入图片格式	操作	实际输出图片格式
RGBA/AGRAY	将输出图片格式设置为HI_PIXEL_FORMAT_UNKNOWN	RGBA8888

- 先调用**acl.himpi.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**acl.himpi.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。

图片格式、宽高对齐、内存约束

实现图片解码功能时，需调用**acl.himpi.dvpp_malloc**接口申请Device上的输入、输出内存，调用**acl.himpi.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。

- 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下表中的计算公式。

  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。

  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**10.8.2 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。

表 10-24 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

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。

10.8.7.2 性能指标说明

性能指标说明

1080p指分辨率为1920*1080的图片；4K指分辨率为3840*2160的图片。单个Device的基本场景性能指标参考如下（1路对应一个通道，一个通道对应一个线程）：

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

10.8.7.3 acl.himpi.pngd_create_chn

c函数原型	**hi_s32 hi_mpi_pngd_create_chn(hi_pngd_chn chn, const hi_pngd_chn_attr *attr)**
Python函数	**ret = acl.himpi.pngd_create_chn(chn, attr)**
函数功能	根据设置的通道属性创建解码通道
输入说明	**chn：** 解码通道号。 该参数的取值范围为[0, 128)，通道总数最多128。 **attr(hi_pngd_chn_attr)：** 解码通道属性指针。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.5 PNGD图像解码返回码**。
注意事项	无
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，可考虑扩展操作系统内存。 ● 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。

10.8.7.4 acl.himpi.pngd_destroy_chn

c函数原型	**hi_s32 hi_mpi_pngd_destroy_chn(hi_pngd_chn chn)**

Python函数	**ret = acl.himpi.pngd_destroy_chn(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。
输入说明	**chn**：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	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.5 PNGD图像解码返回码**。
注意事项	无
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。

10.8.7.5 acl.himpi.pngd_send_stream

c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)**
Python函数	**ret = acl.himpi.pngd_send_stream(chn, stream, png_pic_info, milli_sec)**
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。
输入说明	**chn** ：解码通道号。 该参数的取值范围[0, 128)，通道总数最多128。 **stream(hi_img_stream)**：输入码流信息。该字典内的addr参数配置的地址为Device上的内存地址。 **png_pic_info(hi_pic_info)**：输出图片信息字典。 **milli_sec**：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	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回非0表示失败，参见**10.8.9.5 PNGD图像解码返回码**。
注意事项	无

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**10.8.7.1 功能及约束说明**。 • 解码时，输入输出内存必须使用**acl.himpi.dvpp_malloc**接口和**acl.himpi.dvpp_free**接口进行申请和释放，输入输出内存均需要在调用**10.8.7.6 acl.himpi.pngd_get_image_data**接口获取结果之后才能进行释放。

10.8.7.6 acl.himpi.pngd_get_image_data

c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)**
Python函数	**png_pic_info, stream, ret = acl.himpi.pngd_get_image_data(chn, milli_sec)**
函数功能	解码后，获取解码通道的解码图像及输入Stream。
输入说明	**chn**：解码通道号。 该参数的取值范围[0, 128)，通道总数最多128。 **milli_sec**：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	• **stream(hi_img_stream)**：获取已完成解码的输入码流信息指针。 • **png_pic_info(hi_pic_info)**：获取已经完成解码的图像信息。解码后的数据存放在Device内存中。 • **ret**：int， – 返回0表示获取数据成功。 – 返回非0表示失败，参见**10.8.9.5 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

注意事项	无
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**10.8.7.1 功能及约束说明**。

10.8.7.7 acl.himpi.png_get_image_info

c函数原型	**hi_s32 hi_mpi_png_get_image_info(const hi_img_stream *png_stream, hi_img_info *img_info)**
Python函数	**img_info, ret = acl.himpi.png_get_image_info(png_stream)**
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。
输入说明	**png_stream(hi_img_stream)**：输入码流信息。
返回值说明	**img_info(hi_img_info)**：图片信息字典。 **ret**：int，错误码。 • 返回0表示成功。 • 返回非0表示失败，参见**10.8.9.5 PNGD图像解码返回码**。
注意事项	无
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。
参考资源	无

10.8.8 数据类型

10.8.8.1 总体说明

本章节描述DVPP各个模块所用到的字典、全局变量等，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。

命名规则：

- 各模块均需要的模块放在公共部分，各模块特性部分尽量放在各模块内，并且以模块名开头。
- 例如HI_COMPRESS_MODE_BUTT，表示保留值。

10.8.8.2 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。

10.8.8.2.1 基本数据类型说明

```
HI_FALSE = 0
HI_TRUE = 1
```

10.8.8.2.2 hi_pixel_format

说明

DVPP所支持的各种图片格式。

定义

```
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，当前不支持该格式
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，当前不支持该格式
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
```

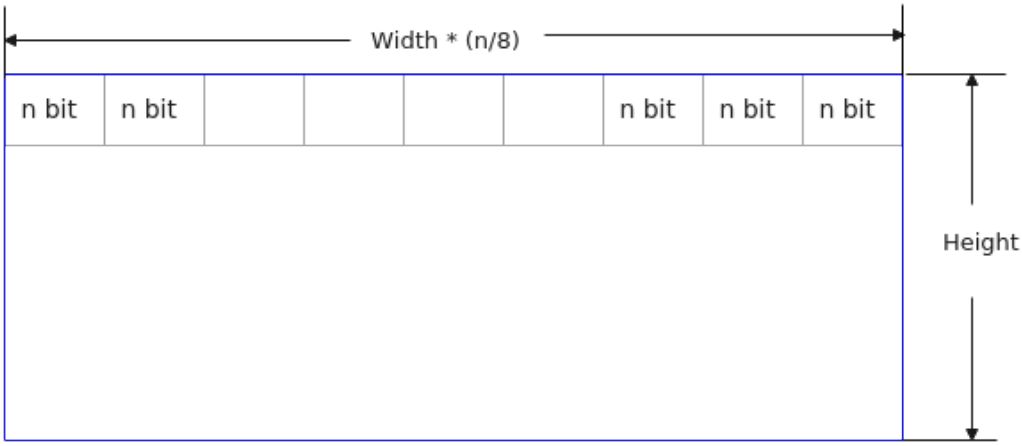
```
    HI_PIXEL_FORMAT_RGBA_8565            = 39// RGBA8565 A:8bit R:5bit G:6bit B:5bit，当前不支持该格式
    HI_PIXEL_FORMAT_BGRA_8565            = 40// BGRA8565 A:8bit B:5bit G:6bit R:5bit，当前不支持该格式
    HI_PIXEL_FORMAT_ARGB_CLUT2            = 41// ARGB Color Lookup Table 2bit
    HI_PIXEL_FORMAT_ARGB_CLUT4            = 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

    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
    HI_PIXEL_FORMAT_BGR_888_PLANAR_BF16    = 1009// BGR888 Planar 每个像素用1个BFloat16数据表示
    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
    HI_PIXEL_FORMAT_BGR_888_BF16           = 1015// BGR888 Package 每个像素用1个BFloat16数据表示

    HI_PIXEL_FORMAT_UNKNOWN = 10000
```

参考信息

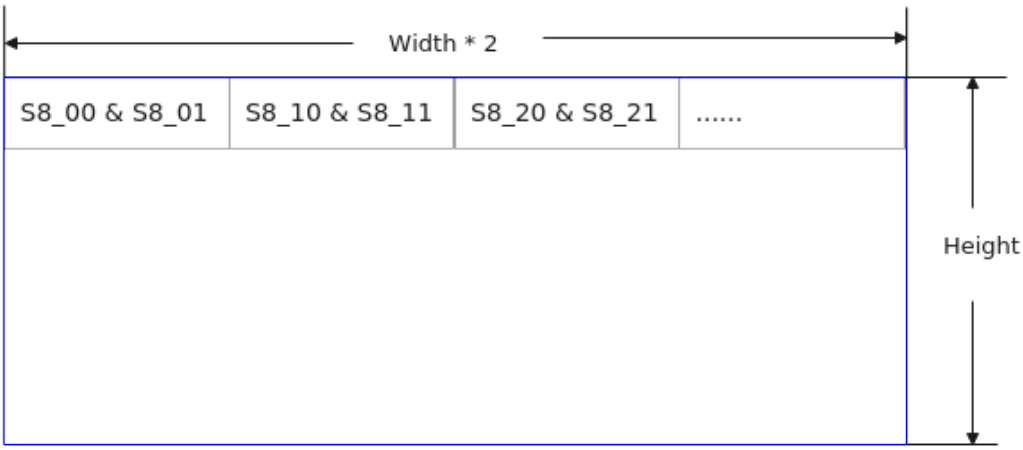
下文以HI_PIXEL_FORMAT_S8C1、HI_PIXEL_FORMAT_S8C2_PACKAGE、HI_PIXEL_FORMAT_S8C2_PLANAR为例说明排布格式，供参考。

图 10-11 HI_PIXEL_FORMAT_S8C1



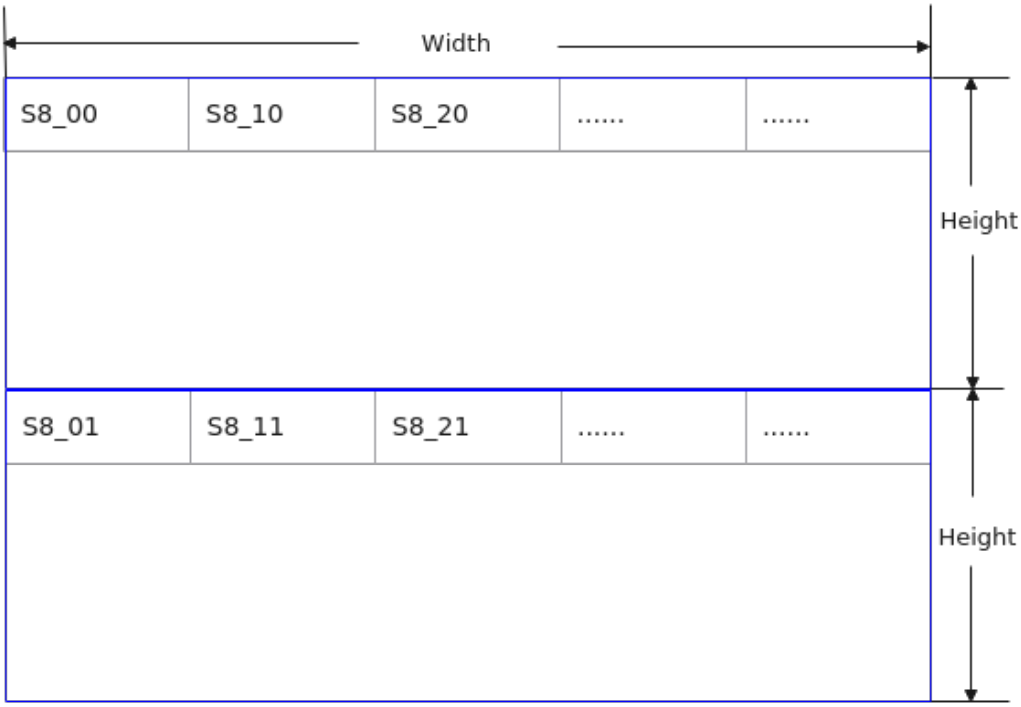
n表示bit位数
Width表示图片宽
Height表示图片高

图 10-12 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 10-13 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

10.8.8.2.3 hi_payload_type

说明

定义视频类型。

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

```
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
# add by hisilicon
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         = 2001
```

10.8.8.2.4 hi_video_field

说明

定义视频图像帧场类型。

定义

```
HI_VIDEO_FIELD_TOP        = 0x1, # even field
HI_VIDEO_FIELD_BOTTOM     = 0x2, # odd field
HI_VIDEO_FIELD_INTERLACED = 0x3, # two interlaced fields
HI_VIDEO_FIELD_FRAME      = 0x4, # frame
HI_VIDEO_FIELD_BUTT       = 0x5
```

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。

10.8.8.2.5 hi_dynamic_range

说明

定义动态范围。

定义

```
HI_DYNAMIC_RANGE_SDR8  = 0
HI_DYNAMIC_RANGE_SDR10 = 1
HI_DYNAMIC_RANGE_HDR10 = 2
HI_DYNAMIC_RANGE_HLG   = 3
HI_DYNAMIC_RANGE_SLF   = 4
HI_DYNAMIC_RANGE_XDR   = 5
HI_DYNAMIC_RANGE_BUTT  = 6
```

成员

成员名称	描述
HI_DYNAMIC_RANGE_SDR8	8bit 数据的标准动态范围。

成员名称	描述
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 $1 \div 12 >= L_c >= 0$ $a = 0.17883277, b = 0.28466892, c = 0.55991073$

10.8.8.2.6 hi_color_gamut

说明

定义色域范围。

定义

```
HI_COLOR_GAMUT_BT601   = 0
HI_COLOR_GAMUT_BT709   = 1
HI_COLOR_GAMUT_BT2020  = 2
```

```
HI_COLOR_GAMUT_USER    = 3
HI_COLOR_GAMUT_BUTT    = 4
```

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。

10.8.8.2.7 hi_video_supplement

说明

定义视频图像帧补充信息。

定义

```
hi_video_supplement = {"misc_info_phys_addr": misc_info_phys_addr, # default allocated buffer
                       "jpeg_dcf_phys_addr": jpeg_dcf_phys_addr,
                       "isp_info_phys_addr": isp_info_phys_addr,
                       "low_delay_phys_addr": low_delay_phys_addr,
                       "bnr_rnt_phys_addr": bnr_rnt_phys_addr,
                       "motion_data_phys_addr": motion_data_phys_addr,
                       "frame_dng_phys_addr": frame_dng_phys_addr,
                       "misc_info_virt_addr": misc_info_virt_addr, # misc info
                       "jpeg_dcf_virt_addr": jpeg_dcf_virt_addr, # jpeg_dcf, used in JPEG DCF
                       "isp_info_virt_addr": isp_info_virt_addr, # isp_frame_info, used in ISP debug, when get raw
and send raw
                       "low_delay_virt_addr": low_delay_virt_addr, # used in low delay
                       "bnr_mot_virt_addr": bnr_mot_virt_addr, # used for 3dnr from bnr mot
                       "motion_data_virt_addr": motion_data_virt_addr, # vpss 3dnr use: gme motion data, filter
motion data, gyro data
                       "frame_dng_virt_addr": frame_dng_virt_addr}
```

成员

成员名称	描述
misc_info_phys_addr	整型，视频扩展信息的物理地址。暂不支持。
jpeg_dcf_phys_addr	整型，Jpeg DCF信息的物理地址。暂不支持。
isp_info_phys_addr	整型，ISP辅助信息的物理地址。暂不支持。
low_delay_phys_addr	整型，低延时信息物理地址。暂不支持。
bnr_rnt_phys_addr	整型，BNR信息物理地址。暂不支持。
motion_data_phys_addr	整型，运动数据物理地址。暂不支持。

成员名称	描述
frame_dng_phys_addr	整型，DNG信息的物理地址。暂不支持。
misc_info_virt_addr	整型，视频扩展信息的虚拟地址。暂不支持。
jpeg_dcf_virt_addr	整型，Jpeg DCF信息的虚拟地址。暂不支持。
isp_info_virt_addr	整型，ISP辅助信息的内核态虚拟地址。暂不支持。
low_delay_virt_addr	整型，低延时信息虚拟地址。暂不支持。
bnr_mot_virt_addr	整型，BNR信息虚拟地址。暂不支持。
motion_data_virt_addr	整型，运动信息虚拟地址。暂不支持。
frame_dng_virt_addr	整型，DNG格式信息的虚拟地址。暂不支持。

10.8.8.2.8 hi_video_frame

说明

定义视频原始图像帧字典。

定义

```
hi_video_frame = {"width": width,
                  "height": height,
                  "field": hi_video_field,
                  "pixel_format": hi_pixel_format,
                  "video_format": hi_video_format,
                  "compress_mode": hi_compress_mode,
                  "dynamic_range": hi_dynamic_range,
                  "color_gamut": hi_color_gamut,
                  "header_stride": [header_stride, header_stride, header_stride],
                  "width_stride": [width_stride, width_stride, width_stride],
                  "height_stride": [height_stride, height_stride, height_stride],
                  "header_phys_addr": [header_phys_addr, header_phys_addr, header_phys_addr],
                  "phys_addr": [phys_addr, phys_addr, phys_addr],
                  "header_virt_addr:[header_virt_addr_list, header_virt_addr_list, header_virt_addr_list],
                  "virt_addr": [virt_addr, virt_addr, virt_addr],
                  "time_ref": time_ref,
                  "pts": pts,
                  "user_data": [user_data],
                  "frame_flag": frame_flag, # frame_flag, can be OR operation.
                  "supplement": hi_video_supplement}
```

成员

成员名称	描述
width	整型，图像宽度。
height	整型，图像高度。
field	整型，帧场模式。预留参数。
pixel_format	整型，视频图像像素格式。

成员名称	描述
video_format	整型，视频图像格式。
compress_mode	整型，视频压缩模式。
dynamic_range	整型，动态范围。预留参数。
color_gamut	整型，色域范围。预留参数。
header_stride	列表，图像压缩头跨距。预留参数。
width_stride	列表，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	列表，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。
header_phys_addr	列表，压缩头物理地址。预留参数。
phys_addr	列表，物理地址。预留参数。
header_virt_addr	列表，压缩头虚拟地址。预留参数。
virt_addr	列表，图像在Device内存中的起始虚拟地址。
time_ref	整型，图像帧序列号。 解码场景下，该参数预留。 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	整型，图像时间戳。
user_data	列表，私有数据。预留参数。

成员名称	描述
frame_flag	● 整形 ● 解码场景下，表示该帧解码是否成功，取值范围： – 0：成功 – 1：失败，如果失败，用户需要及时释放内存 – 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：失败，参考帧个数设置错误。 – 4：失败，VDEC解码帧存大小设置错误，JPEGD解码输出内存大小设置错误。 – 5：失败，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。 ● 在编码场景下，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），仅支持H265协议，取值范围如下： – 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。
supplement	字典，图像的补充信息。预留参数。

注意事项

- 10bit 数据不压缩时在内存中的存储方式是是紧凑排列的。
- 10bit YUV 段压缩数据存储是前 8bit 和后 2bit 分开存储。
- 10bit tile 64x16 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。

10.8.8.2.9 hi_video_frame_info

说明

定义视频图像帧信息字典。

定义

```
hi_video_frame_info = {"v_frame": hi_video_frame, "pool_id" pool_id, "mod_id": hi_mod_id}
```

成员

成员名称	描述
v_frame	字典，视频图像帧属性字典。
pool_id	整型，视频缓存池ID，预留参数。
mod_id	整型，DVPP内的子模块ID，预留参数。

10.8.8.2.10 hi_vb_src

说明

定义VB来源选择。

定义

```
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    = 4
```

10.8.8.2.11 hi_mod_id

说明

定义模块ID类型。

定义

```
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 # Object detection trace
HI_ID_VQA          = 49 # Video quality analysis
HI_ID_LPR          = 50 # Object detection trace
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         = 256
```

10.8.8.2.12 hi_data_bit_width

说明

定义数据比特位宽类型。

目前DVPP模块仅支持HI_DATA_BIT_WIDTH_8和HI_DATA_BIT_WIDTH_10两个数据位宽格式。

定义

```
HI_DATA_BIT_WIDTH_8    = 0
HI_DATA_BIT_WIDTH_10   = 1
HI_DATA_BIT_WIDTH_12   = 2
HI_DATA_BIT_WIDTH_14   = 3
HI_DATA_BIT_WIDTH_16   = 4
HI_DATA_BIT_WIDTH_BUTT = 5
```

10.8.8.2.13 hi_video_format

说明

定义视频格式。

定义

```
HI_VIDEO_FORMAT_LINEAR     = 0 # nature video line
HI_VIDEO_FORMAT_TILE_64x16 = 1 # tile cell: 64pixel x 16line
HI_VIDEO_FORMAT_BUTT       = 2
```

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。

10.8.8.2.14 hi_img_info

说明

定义图片信息的字典。

定义

```
hi_img_info = {"width": width,
               "height": height,
               "width_stride": width_stride,
               "height_stride": height_stride,
               "img_buf_size": img_buf_size,
               "pixel_format": hi_jpeg_raw_format/hi_png_color_format,
               "reserved": [4]}
```

成员

成员名称	描述
width	整型，图片宽。
height	整型，图片高。
width_stride	整型，图片宽stride。
height_stride	整型，图片高stride。
img_buf_size	整型，存放图片的内存大小，单位是byte。
pixel_format	整型，图片格式。
reserved	预留参数，当前版本请设置为全0列表，且不应该显式使用该字段。

10.8.8.2.15 hi_pic_buf_attr

说明

定义图片缓冲区信息的字典。

定义

```
hi_pic_buf_attr = {"width": width,
                   "height": height,
                   "align": align,
                   "bit_width": hi_data_bit_width,
                   "pixel_format": hi_pixel_format,
                   "compress_mode": hi_compress_mode}
```

成员

成员名称	描述
width	整型，图片宽。
height	整型，图片高。
align	整型，对齐参数，预留参数，暂不支持。
bit_width	整型，图片位宽。
pixel_format	整型，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**10.8.5.1 JPEGD功能及约束说明**。
compress_mode	整型，压缩模式。

10.8.8.2.16 hi_jpeg_raw_format

说明

定义JPEG源图片格式。

定义

```
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
```

10.8.8.2.17 hi_dvpp_epoll_ctl_op

说明

定义DVPP epoll操作类型。

定义

```
HI_DVPP_EPOLL_CTL_ADD  = 1
HI_DVPP_EPOLL_CTL_MOD  = 2
HI_DVPP_EPOLL_CTL_DEL  = 3
HI_DVPP_EPOLL_CTL_BUTT = 4
```

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通道的处理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。

10.8.8.2.18 hi_dvpp_epoll_event_type

说明

定义DVPP epoll事件类型。

定义

```
HI_DVPP_EPOLL_IN  = 1
HI_DVPP_EPOLL_OUT = 1 << 1
HI_DVPP_EPOLL_ET  = 1 << 31
```

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。

10.8.8.2.19 hi_dvpp_epoll_event

说明

定义DVPP epoll事件信息。

定义

```
hi_dvpp_epoll_event = {"events": events, "data": data}
```

成员

成员名称	描述
events	整型，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：DVPP_EPOLL_IN | DVPP_EPOLL_OUT。
data	整型，关联到DVPP通道的用户自定义数据。

10.8.8.2.20 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

```
HI_CSC_MATRIX_BT601_WIDE    = 0
HI_CSC_MATRIX_BT601_NARROW  = 1
HI_CSC_MATRIX_BT709_WIDE    = 2
HI_CSC_MATRIX_BT709_NARROW  = 3
HI_CSC_MATRIX_BT2020_WIDE   = 4
HI_CSC_MATRIX_BT2020_NARROW = 5
HI_CSC_MATRIX_USER          = 100
HI_CSC_MATRIX_BUTT          = 101
```

成员

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 |

成员名称	描述
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 |
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。

10.8.8.2.21 hi_coefficient

说明

定义色域转换矩阵参数。

定义

```
hi_coefficient = {"csc_matrix_r0_c0": csc_matrix_r0_c0,
                  "csc_matrix_r0_c1": csc_matrix_r0_c1
                  "csc_matrix_r0_c2": csc_matrix_r0_c2
                  "csc_matrix_r1_c0": csc_matrix_r1_c0
                  "csc_matrix_r1_c1": csc_matrix_r1_c1
                  "csc_matrix_r1_c2": csc_matrix_r1_c2
                  "csc_matrix_r2_c0": csc_matrix_r2_c0
                  "csc_matrix_r2_c1": csc_matrix_r2_c1
                  "csc_matrix_r2_c2": csc_matrix_r2_c2
                  "csc_bias_r0": csc_bias_r0
                  "csc_bias_r1": csc_bias_r1
                  "csc_bias_r2": csc_bias_r2}
```

成员

成员名称	描述
csc_matrix_r0_c0	整型，设置色域转换矩阵参数R0C0。
csc_matrix_r0_c1	整型，设置色域转换矩阵参数R0C1。
csc_matrix_r0_c2	整型，设置色域转换矩阵参数R0C2。
csc_matrix_r1_c0	整型，设置色域转换矩阵参数R1C0。
csc_matrix_r1_c1	整型，设置色域转换矩阵参数R1C1。
csc_matrix_r1_c2	整型，设置色域转换矩阵参数R1C2。
csc_matrix_r2_c0	整型，设置色域转换矩阵参数R2C0。
csc_matrix_r2_c1	整型，设置色域转换矩阵参数R2C1。
csc_matrix_r2_c2	整型，设置色域转换矩阵参数R2C2。
csc_bias_r0	整型，设置偏移R0。
csc_bias_r1	整型，设置偏移R1。
csc_bias_r2	整型，设置偏移R2。

10.8.8.2.22 hi_csc_coefficient

说明

定义色域转换矩阵参数的字典。

定义

```
hi_csc_coefficient = {"yuv_to_rgb_coefficient": hi_coefficient,
                      "rgb_to_yuv_coefficient": hi_coefficient}
```

成员

成员名称	描述
yuv_to_rgb_coefficient	字典，yuv转rgb的色域转换参数。
rgb_to_yuv_coefficient	字典，rgb转yuv的色域转换参数。

10.8.8.2.23 hi_video_size

说明

预留字典，暂不支持。

定义

```
hi_video_size = {"width": width, "height": height}
```

10.8.8.2.24 hi_rect

说明

定义矩形宽高和左上角坐标

定义

```
hi_rect = {"x": x,
           "y": y,
           "width": width,
           "height": height}
```

成员

成员名称	描述
x	整型，左上角x轴
y	整型，左上角y轴
width	整型，矩形宽度
height	整型，矩形高度

10.8.8.2.25 hi_op_mode

说明

定义操作模式枚举。

定义

```
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
```

10.8.8.2.26 hi_point

说明

定义坐标点结构体。

定义

```
hi_point = {"x": x,
            "y": y}
```

成员

成员名称	描述
x	整型，x轴坐标值。
y	整型，y轴坐标值。

10.8.8.2.27 hi_size

说明

定于矩形尺寸的结构体。

定义

```
hi_size = {"width": width,
           "height": height}
```

成员

成员名称	描述
width	整型，宽度
height	整型，高度

10.8.8.2.28 hi_compress_mode

说明

定义视频压缩数据格式枚举。

定义

```
HI_COMPRESS_MODE_NONE  = 0
HI_COMPRESS_MODE_SEG   = 1
HI_COMPRESS_MODE_TILE  = 2
HI_COMPRESS_MODE_HFBC  = 3
HI_COMPRESS_MODE_LINE  = 4
HI_COMPRESS_MODE_FRAME = 5
HI_COMPRESS_MODE_BUTT  = 6
```

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BS9SX1A AI处理器SoC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。

10.8.8.2.29 hi_module_type

说明

定义模块类型。

定义

```
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
```

10.8.8.2.30 hi_img_base_info

说明

定义图片基本信息。

定义

```
hi_img_base_info = {"width": width,
                    "height": height,
                    "pixel_format": hi_pixel_format,
                    "reserved": [2]}
```

成员

成员名称	描述
width	整型，图片宽度。
height	整型，图片高度。
pixel_format	整型，图片格式。
reserved	列表，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。

10.8.8.2.31 hi_img_align_info

说明

定义图片对齐信息。

定义

```
hi_img_align_info = {"width_stride": width_stride,
                     "height_stride": height_stride,
                     "img_buf_size": img_buf_size,
                     "reserved": [2]}
```

成员

成员名称	描述
width_stride	整型，图片宽度对齐后的值。
height_stride	整型，图片高度对齐后的值。
img_buf_size	整型，存放图片数据所需的内存大小，单位Byte。
reserved	列表，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。

10.8.8.3 视频/图像解码

10.8.8.3.1 hi_vdec_send_mode

说明

定义码流发送方式。

定义

```
HI_VDEC_SEND_MODE_STREAM = 0 # send by stream
HI_VDEC_SEND_MODE_FRAME  = 1 # send by frame
HI_VDEC_SEND_MODE_COMPAT = 2 # One frame supports multiple packets sending.
HI_VDEC_SEND_MODE_BUTT   = 3 # reserved
```

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

- 仅支持HI_VDEC_SEND_MODE_FRAME。
- 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。
- 不支持一帧图像分多包进行发送的模式。

10.8.8.3.2 hi_vdec_frame_type

说明

定义码流发送方式。

定义

```
HI_VDEC_FRAME_TYPE_I    = 0
HI_VDEC_FRAME_TYPE_P    = 1
HI_VDEC_FRAME_TYPE_B    = 2
HI_VDEC_FRAME_TYPE_BUTT = 3
```

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。

10.8.8.3.3 hi_quick_mark_mode

说明

定义快速释放参考帧模式。

定义

```
HI_QUICK_MARK_ADAPT = 0
HI_QUICK_MARK_FORCE = 1
HI_QUICK_MARK_NONE  = 2
HI_QUICK_MARK_BUTT  = 3
```

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。
HI_QUICK_MARK_NONE	普通模式。预留参数，暂未支持。
HI_QUICK_MARK_BUTT	保留值。

注意事项

无。

10.8.8.3.4 hi_vdec_pic_info

说明

定义视频原始图像帧字典。

定义

```
hi_vdec_pic_info = {"width": width,
                    "height": height,
                    "width_stride": width_stride,
                    "height_stride": height_stride,
                    "pixel_format": hi_pixel_format,
                    "vir_addr": vir_addr,
                    "buffer_size": buffer_size,
                    "offset_top": offset_top,
                    "offset_bottom": offset_bottom,
                    "offset_left": offset_left,
                    "offset_right": offset_right}
```

成员

成员名称	描述
width	整型，图像宽度，视频解码时使用该参数。 ● 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），当输入码流宽度小于或等于4096时，解码图像经过缩放后的输出宽度范围为[10, 8192]，当输入码流宽度大于4096时，解码图像经过缩放后的输出宽度范围为[128, 8192]。 以下场景，VDEC不对输出图片进行缩放：在hi_vdec_pic_info内，将width参数值和height参数值同时设置为0。
height	整型，图像高度，视频解码时使用该参数。 ● 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），当输入码流宽度小于或等于4096时，解码图像经过缩放后的输出高度范围为[6, 8192]；当输入码流宽度大于4096时，解码图像经过缩放后的输出高度范围为[128, 8192]。
width_stride	整型，图像宽度步长，视频解码时，YUV420 semi-planar/YVU420 semi-planar输出时，该值为宽向上16对齐；RGB888/BGR888输出时，该值为宽向上16对齐，再乘3。 取值范围：[32, 16384]。
height_stride	整型，图像高度步长，视频解码时，YUV420 semi-planar/YVU420 semi-planar输出时，该值为高向上2对齐；RGB888/BGR888输出时，该值等于高。 取值范围：[6, 16384]。

成员名称	描述
pixel_format	整型，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**10.8.5.3 VDEC功能及约束说明**，JPEGD图片解码输出图片格式参见**10.8.5.1 JPEGD功能及约束说明**。
vir_addr	整型，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	整型，存放解码图像的缓冲区大小。
offset_top	整型，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10-14**所示。
offset_bottom	整型，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10-14**所示。
offset_left	整型，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10-14**所示。
offset_right	整型，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10-14**所示。

注意事项

10bit的数据经过VPC处理（图像处理）之后均会变成8bit。

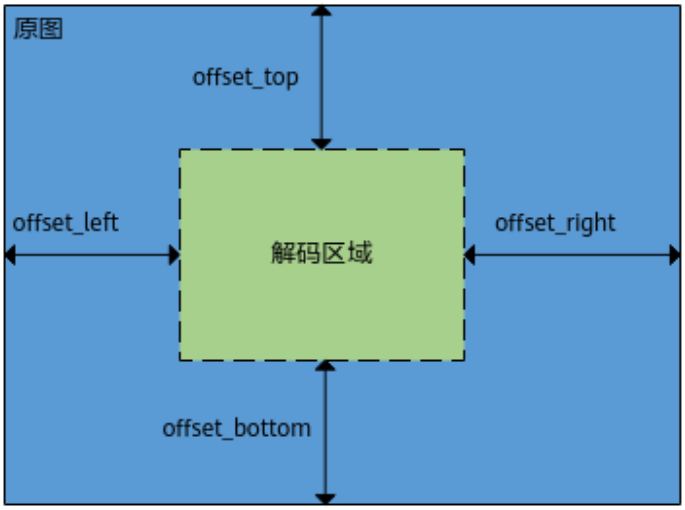
参考信息

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 10-14 按区域解码功能示意图



10.8.8.3.5 hi_vdec_video_attr

说明

定义视频解码视频属性。

定义

```
hi_vdec_video_attr = {"ref_frame_num": ref_frame_num, "temporal_mvp_en": temporal_mvp_en,
"tmv_buf_size": tmv_buf_size}
```

成员

成员名称	描述
ref_frame_num	整型，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 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。

成员名称	描述
temporal_mvp_en	整型，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**acl.himpi.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	整型，视频解码图像 Tmv（Temporal Motion Vector）Buffer 大小，temporal_mvp_en 为 1 时有效。 用户可先调用**acl.himpi.vdec_get_tmv_buf_size**接口获取视频解码图像TMV缓存大小。

10.8.8.3.6 hi_vdec_chn_attr

说明

定义解码通道属性字典。

定义

```
hi_vdec_chn_attr= {"type": hi_payload_type, # RW; video type to be decoded
                   "mode": hi_vdec_send_mode, # RW; send by stream or by frame
                   "pic_width": pic_width, # RW; max pic width
                   "pic_height": pic_height, # RW; max pic height
                   "stream_buf_size": stream_buf_size, # stream buf size(Byte)
                   "frame_buf_size": frame_buf_size, # RW; frame buf size(Byte)
                   "frame_buf_cnt": frame_buf_cnt, # RW; frame buf num
                   "video_attr": hi_vdec_video_attr} # RW; structure with video
```

成员

成员名称	描述
type	整型，解码协议类型值。 静态属性。
mode	整型，码流发送方式。 H.264/H.265/JPEG仅支持按帧发送码流。 静态属性。
pic_width	整型，通道支持的解码图像最大宽（以像素为单位） 静态属性。

成员名称	描述
pic_height	整型，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	整型，码流缓存的大小。 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	整型，解码图像帧存 buffer 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	整型，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	字典，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。

10.8.8.3.7 hi_h264_protocol_param

说明

与 H.264 协议相关的内存分配参数。

定义

```
hi_h264_protocol_param = {"max_slice_num": max_slice_num, # RW, Range: [1, 300]; max slice num support
                          "max_sps_num": max_sps_num, # RW, Range: [1, 32]; max sps num  support
                          "max_pps_num": max_pps_num } # RW, Range: [1, 256]; max pps num  support
```

成员

成员名称	描述
max_slice_num	整型，该通道解码支持的最大 Slice 个数。 取值范围：[1, 300]。 Default：16。
max_sps_num	整型，该通道解码支持的最大 SPS 个数。 取值范围：[1, 32]。 Default：2。
max_pps_num	整型，该通道解码支持的最大 PPS 个数。 取值范围：[1, 256]。 Default：2。

10.8.8.3.8 hi_h265_protocol_param

说明

定义与 H.265 协议相关的内存分配参数。

定义

```
hi_h265_protocol_param = {"max_slice_segment_num": max_slice_segment_num, # RW; max slice segment
num support
                          "max_vps_num": max_vps_num, # RW; max vps num support
                          "max_sps_num": max_sps_num, # RW; max sps num support
                          "max_pps_num":max_pps_num } # RW; max pps num support
```

成员

成员名称	描述
max_slice_segment_num	整型，该通道解码支持的最大 SliceSegment 个数。 取值范围：[1, 600] Default：16。
max_vps_num	整型，该通道解码支持的最大 VPS 个数。 取值范围：[1, 16]。 Default：2。

成员名称	描述
max_sps_num	整型，该通道解码支持的最大 SPS 个数。 取值范围：[1, 16]。 Default：2。
max_pps_num	整型，该通道解码支持的最大 PPS 个数。 取值范围：[1, 64]。 Default：2。

10.8.8.3.9 hi_vdec_protocol_param

说明

定义与协议相关的内存分配参数。

定义

```
hi_vdec_protocol_param = {"type": type,            # RW; video type to be decoded, only h264 and h265 supported
                          "h264_param": hi_h264_protocol_param, # protocol param structure for h264

hi_vdec_protocol_param = {"type": type,            # RW; video type to be decoded, only h264 and h265 supported
                          "h265_param": hi_h265_protocol_param } # protocol param structure for h265
```

成员

成员名称	描述
type	整型，解码通道支持的协议。
h264_param	字典，H.264 协议参数。
h265_param	字典，H.265 协议参数。

10.8.8.3.10 hi_video_dec_mode

说明

定义视频解码模式。

定义

```
HI_VIDEO_DEC_MODE_IPB  = 0
HI_VIDEO_DEC_MODE_IP   = 1
HI_VIDEO_DEC_MODE_I    = 2
HI_VIDEO_DEC_MODE_BUTT = 3
```

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。

10.8.8.3.11 hi_video_out_order

说明

定义视频解码输出顺序。

定义

```
HI_VIDEO_OUT_ORDER_DISPLAY = 0
HI_VIDEO_OUT_ORDER_DEC     = 1
HI_VIDEO_OUT_ORDER_BUTT    = 2
```

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。

10.8.8.3.12 hi_vdec_video_param

说明

定义视频解码高级参数。

定义

```
hi_vdec_video_param = {"composite_dec_en": composite_dec_en,
                       "slice_input_en": slice_input_en,
                       "err_threshold": err_threshold,
                       "dec_mode": hi_video_dec_mode, # frames output order ,0: the same with display order , 1:
```

```
the same with decoder order
                "out_order": hi_video_out_order,
                "compress_mode": hi_compress_mode, # RW; compress mode
                "video_format": hi_video_format, # RW; video format
                "quick_mark_mode": hi_quick_mark_mode }
```

成员

成员名称	描述
composite_dec_en	整型，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，当前不支持。
dec_mode	整型, 解码模式。 Default：HI_VIDEO_DEC_MODE_IPB。
out_order	整型，解码图像输出顺序。 Default：HI_VIDEO_OUT_ORDER_DISPLAY
compress_mode	整型，解码图像压缩模式。 Default：HI_COMPRESS_MODE_HFBC
video_format	整型，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	整型，快速释放参考帧模式。预留参数，暂未支持。

10.8.8.3.13 hi_vdec_param_picture

说明

定义图像解码高级参数。

定义

```
hi_vdec_param_picture = {"pixel_format": hi_pixel_format, # RW; out put pixel format
                "u32Alpha": u32Alpha }
```

成员

成员名称	描述
pixel_format	整型，图片像素格式。保留参数。
u32Alpha	整型，RGB解码透明度。保留参数。

10.8.8.3.14 hi_vdec_chn_param

说明

定义解码通道高级参数。

定义

```
hi_vdec_chn_param = {"type": hi_payload_type , # RW; video type to be decoded
                     "display_frame_num": display_frame_num, # RW, Range: [0, 16]; display frame num
                     "video_param": hi_vdec_video_param} # structure with video ( h265/h264/VP9)

hi_vdec_chn_param = {"type": hi_payload_type , # RW; video type to be decoded
                     "display_frame_num": display_frame_num, # RW, Range: [0, 16]; display frame num
                     "pic_param": hi_vdec_param_picture } # structure with picture (jpeg)
```

成员

成员名称	描述
type	整型，解码协议。
display_frame_num	整型，解码缓存图像的最小帧数。 取值范围：[0, 16]。 Default：2。
video_param	字典，视频(H.264/H.265)解码高级参数。
pic_param	字典，图片(JPEG)解码高级参数。

10.8.8.3.15 hi_vdec_dec_err

说明

定义解码错误信息字典。

定义

```
hi_vdec_dec_err = {"set_pic_size_err": set_pic_size_err, # R; Picture width or height is larger than channel
width or height.
                   "set_protocol_num_err": set_protocol_num_err, # R; Protocol num is not enough. eg: slice, pps,
sps.
                   "set_ref_num_err": set_ref_num_err, # R; Reference num is not enough.
                   "set_pic_buf_size_err": set_pic_buf_size_err, # R; The buffer size of picture is not enough.
                   "format_err": format_err, # R; Format error. eg: do not support filed.
                   "stream_unsupport": stream_unsupport, # R; Unsupported the stream specification.
                   "pack_err": pack_err # R; Stream package error.
                   "stream_size_over": stream_size_over, # R; the stream size is too big and force discard stream
                   "stream_not_release": stream_not_release } # R; the stream not released for too long time
```

成员

成员	描述
set_pic_size_err	整型，图像的宽（或高）比通道的宽（或高）大，保留。

成员	描述
set_protocol_num_err	整型，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	整型，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	整型，图像 buffer 内存大小不够。
format_err	整型，不支持的格式。JPEG的Progressive码流。
stream_unsupport	整型，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	整型，码流有错。
stream_size_over	整型，一帧码流太大了，当整个SCDbuffer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	整型，VFMW （video firmware）内部管理码流错误，出现长时间不释放码流的情况，保留。

10.8.8.3.16 hi_vdec_chn_status

说明

定义通道状态字典。

定义

```
hi_vdec_chn_status = {"type": hi_payload_type, # R; video type to be decoded
                      "left_stream_bytes": left_stream_bytes, # R; left stream bytes waiting for decode
                      "left_stream_frames": left_stream_frames, # R; left frames waiting for decode,only valid for
HI_VDEC_SEND_MODE_FRAME
                      "left_decoded_frames": left_decoded_frames, # R; pics waiting for output.
                      "is_started": is_started, # R; had started recv stream?
                      "recv_stream_frames": recv_stream_frames, # R; how many frames of stream has been
received. valid when send by frame.
                      "dec_stream_frames": dec_stream_frames, # R; how many frames of stream has been
decoded. valid when send by frame.
                      "dec_err": hi_vdec_dec_err, # R; information about decode error.
                      "width": width,
                      "height": height,
                      "latest_frame_pts": latest_frame_pts }
```

成员

成员名称	描述
type	整型，解码协议。
left_stream_bytes	整型，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。

成员名称	描述
left_stream_frames	整型，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	整型，图像 buffer 中剩余的 pic 数目。
is_started	整型，解码器是否已经启动接收码流。
recv_stream_frames	整型，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	整型，码流 buffer 中已解码帧数。仅按帧发送时有效。
dec_err	字典，解码错误信息。
width	整型，解码后输出图片的宽。
height	整型，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。

10.8.8.3.17 hi_vdec_stream

说明

定义视频/图像解码的码流字典。

定义

```
hi_vdec_stream = {"end_of_frame": end_of_frame, # W; is the end of a frame.
                  "end_of_stream": end_of_stream, # W; is the end of all stream.
                  "need_display": need_display, # W; is the current frame displayed. only valid by
HI_VDEC_SEND_MODE_FRAME.
                  "pts": pts, # W; time stamp
                  "private_data": private_data, # W; private data,reserved
                  "len": len, # W; length of stream
                  "addr": addr} # W; stream address
```

成员

成员	描述
end_of_frame	整型，码流包的长度，以Byte为单位。 当前帧是否结束（此值目前暂无效）。
end_of_stream	整型，码流包的时间戳，以us为单位。 是否发完所有码流。
need_display	整型，当前帧是否输出显示。 0：不显示，输出Buffer无解码数据，允许输出Buffer设为None。 1：显示，设置为1，才能输出解码结果。

成员	描述
pts	整型，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	整型，码流包的长度，以Byte为单位。
addr	整型，码流包的地址。

注意事项

- 按帧发送时，解码图像的时间戳等于码流包中的时间戳。
- 按流发送时，解码图像的时间戳等于 0。
- 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。
- 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）。

10.8.8.3.18 hi_vdec_supplement_info

说明

定义输出帧补充信息， 暂不支持。

定义

```
hi_vdec_supplement_info = {"type": hi_payload_type , # RW; Video type to be decoded.
                           "video_supplement_info": hi_vdec_video_supplement_info} # Structure with video (h265/h264)
```

10.8.8.3.19 hi_vdec_video_supplement_info

说明

定义输出视频帧补充信息， 暂不支持。

定义

```
hi_vdec_video_supplement_info = {"frame_type": hi_vdec_frame_type,
                                 "err_rate": err_rate,
                                 "poc": poc}
```

成员

成员名称	描述
frame_type	整型，解码帧类型。预留参数，暂未支持。
err_rate	整型，错误率。预留参数，暂未支持。
poc	整型，poc值。预留参数，暂未支持。

10.8.8.3.20 hi_jpegd_precision_mode

说明

定义JPEGD解码输出图片的宽、高对齐模式。

定义

```
YUVOUT_ALIGN_DOWN = 0
YUVOUT_ALIGN_UP = 1
YUVOUT_ALIGN_DOWN_COMPAT = 2
```

成员名称	描述
YUVOUT_ALIGN_DOWN	整型，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。

10.8.8.4 视频/图像编码

10.8.8.4.1 hi_venc_stream

说明

定义帧码流类型字典。

定义

```
hi_venc_stream = {"pack": hi_venc_pack, # R; stream pack attribute
                  "pack_cnt": pack_cnt, # R; the pack number of one frame stream
                  "seq": seq} # R; the list number of stream

hi_venc_stream = {"pack": hi_venc_pack, # R; stream pack attribute
                  "pack_cnt": pack_cnt, # R; the pack number of one frame stream
```

```
                "seq": seq, # R; the list number of stream
                "h264_info": hi_venc_h264_stream_info, # R; the stream info of h264
                "h264_adv_info": hi_venc_h264_adv_stream_info} # R; the stream info of h264

hi_venc_stream = {"pack": hi_venc_pack, # R; stream pack attribute
                "pack_cnt": pack_cnt, # R; the pack number of one frame stream
                "seq": seq, # R; the list number of stream
                "h264_info": hi_venc_h264_stream_info} # R; the stream info of h264

hi_venc_stream = {"pack": hi_venc_pack, # R; stream pack attribute
                "pack_cnt": pack_cnt, # R; the pack number of one frame stream
                "seq": seq, # R; the list number of stream
                "jpeg_info": hi_venc_jpeg_stream_info}

hi_venc_stream = {"pack": hi_venc_pack, # R; stream pack attribute
                "pack_cnt": pack_cnt, # R; the pack number of one frame stream
                "seq": seq, # R; the list number of stream
                "h265_info": hi_venc_h265_stream_info} # R; the stream info of h265

hi_venc_stream = {"pack": hi_venc_pack, # R; stream pack attribute
                "pack_cnt": pack_cnt, # R; the pack number of one frame stream
                "seq": seq, # R; the list number of stream
                "h265_info": hi_venc_h265_stream_info, # R; the stream info of h265
                "h265_adv_info": hi_venc_h265_adv_stream_info} #R; the stream info of h265

hi_venc_stream = {"pack": hi_venc_pack, # R; stream pack attribute
                "pack_cnt": pack_cnt, # R; the pack number of one frame stream
                "seq": seq, # R; the list number of stream
                "prores_info": hi_venc_prores_stream_info}
```

成员

成员名称	描述
pack	整型，帧码流包结构。
pack_cnt	整型，一帧码流的所有包的个数。
seq	整型，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。
h264_info/h265_info	字典，码流特征信息。
jpeg_info/prores_info	码流特征信息，暂不支持。
h264_adv_info/h265_adv_info	字典，码流高级特征信息。

10.8.8.4.2 hi_venc_pack

说明

定义帧码流包字典。

定义

```
hi_venc_pack = {"phys_addr": phys_addr,
                "addr": addr,
                "len": len,
                "pts": pts,
                "is_frame_end": is_frame_end,
```

```
            "data_type": hi_venc_data_type,
            "offset": offset,
            "data_num": data_num
            "pack_info": [hi_venc_pack_info]}

hi_venc_pack = {"input_addr": input_addr,
            "addr": addr,
            "len": len,
            "pts": pts,
            "is_frame_end": is_frame_end,
            "data_type": hi_venc_data_type,
            "offset": offset,
            "data_num": data_num
            "pack_info": [hi_venc_pack_info]}
```

成员

成员名称	描述
addr	整型，码流包首地址。
phys_addr	整型, 码流包物理地址。
input_addr	整型，用户输入码流数据的内存地址。
len	整型，码流包长度。 当len=0，表示视频/图像编码失败。
data_type	整型，码流类型，支持 H.265/H.264/JPEG协议类型的数据包。
pts	整型，时间戳。单位：us。
is_frame_end	整型，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。
offset	整型，码流包中有效数据与码流包首地址 addr 的偏移。
data_num	整型，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]	列表，当前码流包数据中包含码流包数据信息。

10.8.8.4.3 hi_venc_chn_attr

说明

定义编码通道属性字典。

定义

```
hi_venc_chn_attr = {"venc_attr": hi_venc_attr, "rc_attr": hi_venc_rc_attr, "gop_attr": hi_venc_gop_attr}
```

成员

成员名称	描述
venc_attr	字典，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	字典，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	字典，Gop Mode类型的字典。 对于H264、H265协议，需要配置该参数。

10.8.8.4.4 hi_venc_attr

说明

定义编码器属性字典。

定义

```
hi_venc_attr = {"type": hi_payload_type,
                "max_pic_width": max_pic_width,
                "max_pic_height": max_pic_height,
                "buf_size": buf_size,
                "profile": profile,
                "is_by_frame": is_by_frame,
                "pic_width": pic_width,
                "pic_height": pic_height,
                "h264_attr":hi_venc_h264_attr} # attributes of H264e

hi_venc_attr = {"type": type,
                "max_pic_width": max_pic_width,
                "max_pic_height": max_pic_height,
                "buf_size": buf_size,
                "profile": profile,
                "is_by_frame": is_by_frame,
                "pic_width": pic_width,
                "pic_height": pic_height,
                "h265_attr":hi_venc_h265_attr} # attributes of H265e

hi_venc_attr = {"type": hi_payload_type,
                "max_pic_width": max_pic_width,
                "max_pic_height": max_pic_height,
                "buf_size": buf_size,
                "profile": profile,
                "is_by_frame": is_by_frame,
                "pic_width": pic_width,
                "pic_height": pic_height,
                "jpeg_attr":hi_venc_jpeg_attr} # attributes of jpeg

hi_venc_attr = {"type": hi_payload_type,
                "max_pic_width": max_pic_width,
                "max_pic_height": max_pic_height,
```

```
        "buf_size": buf_size,
        "profile": profile,
        "is_by_frame": is_by_frame,
        "pic_width": pic_width,
        "pic_height": pic_height,
        "prores_attr":hi_venc_prores_attr} # attributes of prores
```

成员

成员名称	描述
type	整型，编码协议类型。
max_pic_width	整型，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10-25**。
max_pic_height	整型，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10-25**。

成员名称	描述
buf_size	整型，码流buffer大小，单位为Byte，静态属性。 ● 调用**acl.himpi.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**acl.himpi.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。 – **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。 – **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	整型，编码的等级，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）。

成员名称	描述
is_by_frame	整型，帧/包模式获取码流，静态属性。 ● HI_TRUE：按帧获取。 ● HI_FALSE：按包获取。
pic_width	编码通道宽度，必须是MIN_ALIGN的整数倍。 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10-25。 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。 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10-25。 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。
h265_attr	编码协议的属性。
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 10-25 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

10.8.8.4.5 hi_venc_rc_attr

说明

定义编码通道码率控制器属性。

定义

```
hi_venc_rc_attr = {"rc_mode": rc_mode, # RW; the type of rc
                   "h264_cbr": hi_venc_h264_cbr}

hi_venc_rc_attr = {"rc_mode": rc_mode, # RW; the type of rc
                   "h264_vbr": hi_venc_h264_vbr}

hi_venc_rc_attr = {"rc_mode": rc_mode, # RW; the type of rc
                   "h264_avbr": hi_venc_h264_avbr}

hi_venc_rc_attr = {"rc_mode": rc_mode, # RW; the type of rc
                   "h264_qvbr": hi_venc_h264_qvbr}

hi_venc_rc_attr = {"rc_mode": rc_mode, # RW; the type of rc
                   "h264_cvbr": hi_venc_h264_cvbr}

hi_venc_rc_attr = {"rc_mode": rc_mode, # RW; the type of rc
                   "h265_cbr": hi_venc_h265_cbr}

hi_venc_rc_attr = {"rc_mode": rc_mode, # RW; the type of rc
                   "h265_vbr": hi_venc_h265_vbr}

hi_venc_rc_attr = {"rc_mode": rc_mode, # RW; the type of rc
                   "h265_avbr": hi_venc_h265_avbr}

hi_venc_rc_attr = {"rc_mode": rc_mode, # RW; the type of rc
                   "h265_qvbr": hi_venc_h265_qvbr}

hi_venc_rc_attr = {"rc_mode": rc_mode, # RW; the type of rc
                   "h265_cvbr": hi_venc_h265_cvbr}
```

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。 对于不同协议，相同 RC 模式的属性变量基本一致。**表10-26**介绍了几种RC模式的公共的属性。
h264_cbr	字典，H.264协议编码通道CBR模式属性。
h264_vbr	字典，H.264协议编码通道VBR模式属性。
h264_avbr	字典，H.264协议编码通道AVBR模式属性。
h264_qvbr	字典，H.264协议编码通道QVBR模式属性。
h264_cvbr	字典，H.264协议编码通道CVBR模式属性。
h264_fixqp	字典，H.264协议编码通道FIXQP模式属性。

成员名称	描述
h264_qpmap	字典，H.264协议编码通道QPMAP模式属性。预留参数，暂不支持。
mjpeg_fixqp	字典，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。
mjpeg_cbr	字典，Mjpeg协议编码通道CBR模式属性。预留参数，暂不支持。
mjpeg_vbr	字典，Mjpeg协议编码通道VBR模式属性。预留参数，暂不支持。
h265_cbr	字典，H.265协议编码通道CBR模式属性。
h265_vbr	字典，H.265协议编码通道VBR模式属性。
h265_avbr	字典，H.265协议编码通道AVBR模式属性。
h265_qvbr	字典，H.265协议编码通道QVBR模式属性。
h265_cvbr	字典，H.265协议编码通道CVBR模式属性。
h265_fixqp	字典，H.265协议编码通道FIXQP模式属性。
h265_qpmap	字典，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 10-26 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

10.8.8.4.6 hi_venc_rc_mode

说明

定义编码通道码率控制器模式。

定义

```
HI_VENC_RC_MODE_H264_CBR    = 1
HI_VENC_RC_MODE_H264_VBR    = 2
HI_VENC_RC_MODE_H264_AVBR   = 3
HI_VENC_RC_MODE_H264_QVBR   = 4
HI_VENC_RC_MODE_H264_CVBR   = 5
HI_VENC_RC_MODE_H264_FIXQP  = 6
HI_VENC_RC_MODE_H264_QPMAP  = 7
HI_VENC_RC_MODE_MJPEG_CBR   = 8
HI_VENC_RC_MODE_MJPEG_VBR   = 9
HI_VENC_RC_MODE_MJPEG_FIXQP = 10
HI_VENC_RC_MODE_H265_CBR    = 11
HI_VENC_RC_MODE_H265_VBR    = 12
HI_VENC_RC_MODE_H265_AVBR   = 13
HI_VENC_RC_MODE_H265_QVBR   = 14
HI_VENC_RC_MODE_H265_CVBR   = 15
HI_VENC_RC_MODE_H265_FIXQP  = 16
HI_VENC_RC_MODE_H265_QPMAP  = 17
HI_VENC_RC_MODE_BUTT        = 18
```

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。
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。
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。

成员名称	描述
HI_VENC_RC_MODE_H265_QPMAP	H.265 QPMAP模式。预留参数，暂不支持。

10.8.8.4.7 hi_venc_h264_cbr

说明

定义H.264编码通道CBR属性字典。

定义

```
hi_venc_h264_cbr = {"gop": gop,
                    "stats_time": stats_time,
                    "src_frame_rate": src_frame_rate,
                    "dst_frame_rate": dst_frame_rate,
                    "bit_rate": bit_rate }
```

成员

成员名称	描述
gop	整型，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	整型，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]。

成员名称	描述
src_frame_rate	整型，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	整型，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	整型，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10-27**所示。注意，**表10-27**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 10-27 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

10.8.8.4.8 hi_venc_h264_vbr

说明

定义H.264编码通道VBR属性字典。

定义

```
hi_venc_h264_vbr = {"gop": gop,
                    "stats_time": stats_time,
                    "src_frame_rate": src_frame_rate,
```

```
            "dst_frame_rate": dst_frame_rate,
            "max_bit_rate": max_bit_rate}
```

成员

成员名称	描述
gop	整型，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	整型，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	整型，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	整型，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	整型，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]。

10.8.8.4.9 hi_venc_h264_avbr

说明

定义 H.264 编码通道AVBR属性字典。

定义

```
hi_venc_h264_avbr = {"gop": gop,
                     "stats_time": stats_time,
                     "src_frame_rate": src_frame_rate,
                     "dst_frame_rate": dst_frame_rate,
                     "max_bit_rate": max_bit_rate}
```

成员

成员名称	描述
gop	整型，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	整型，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	整型，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]。

成员名称	描述
dst_frame_rate	整型，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	整型，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]

10.8.8.4.10 hi_venc_h264_qvbr

说明

定义H.264协议编码通道QVBR模式属性字典。

定义

```
hi_venc_h264_qvbr = {"gop": gop,
                     "stats_time": stats_time,
                     "src_frame_rate": src_frame_rate,
                     "dst_frame_rate": dst_frame_rate,
                     "target_bit_rate": target_bit_rate}
```

成员

成员名称	描述
gop	整型，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	整型，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	整型，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	整型，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	整型，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]

10.8.8.4.11 hi_venc_h264_cvbr

说明

定义H.264协议编码通道CVBR模式属性字典。

定义

```
hi_venc_h264_cvbr = {"gop": gop,
                     "stats_time": stats_time,
                     "src_frame_rate": src_frame_rate,
                     "dst_frame_rate": dst_frame_rate,
                     "max_bit_rate": max_bit_rate,
                     "short_term_stats_time": short_term_stats_time,
                     "long_term_stats_time": long_term_stats_time,
                     "long_term_max_bit_rate": long_term_max_bit_rate,
                     "long_term_min_bit_rate": long_term_min_bit_rate}
```

成员

成员名称	描述
gop	整型，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	整型，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]。
src_frame_rate	整型，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	整型，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	整型，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]

成员名称	描述
short_term_stats_time	整型，码率短期统计时间，以秒为单位。 取值范围：[1, 120]。
long_term_stats_time	整型，码率长期统计时间，默认为分钟。 取值范围：[1, 1440]。
long_term_max_bit_rate	整型，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	整型，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] 。

10.8.8.4.12 hi_venc_h264_fixqp

说明

定义 H.264 编码通道FIXQP属性字典。

定义

```
hi_venc_h264_fixqp = {"gop": gop,
                      "src_frame_rate": src_frame_rate,
                      "dst_frame_rate": dst_frame_rate,
                      "i_qp": i_qp,
                      "p_qp": p_qp,
                      "b_qp": b_qp}
```

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	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时播放画面会不连贯。 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时播放画面会不连贯。 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。

10.8.8.4.13 hi_venc_h264_qpmap

说明

定义 H.264 编码通道QPMAP属性字典。暂不支持。

定义

```
hi_venc_h264_cvbr = {"gop": gop,
                     "stats_time": stats_time,
```

```
                "src_frame_rate": src_frame_rate,
                "dst_frame_rate": dst_frame_rate}
```

10.8.8.4.14 hi_venc_mjpeg_cbr

说明

定义MJPEG编码通道CBR属性字典。暂不支持。

定义

```
hi_venc_mjpeg_cbr = {"stats_time": stats_time, # RW; Range:[1, 60]; the rate statistic time, the unit is
senconds(s)
                "src_frame_rate": src_frame_rate, # RW; Range:[1, 240]; the input frame rate of the venc
channel
                "dst_frame_rate": dst_frame_rate, # RW; Range:[1, 240]; the target frame rate of the venc
channel,
                                                # can not be larger than src_frame_rate
                "bit_rate": bit_rate} # RW; Range:[2, 614400]; average bitrate} hi_venc_mjpeg_cbr;
```

10.8.8.4.15 hi_venc_mjpeg_vbr

说明

定义MJPEG编码通道VBR属性字典。暂不支持。

定义

```
hi_venc_mjpeg_cbr = {"stats_time": stats_time, # RW; Range:[1, 60]; the rate statistic time, the unit is
senconds(s)
                "src_frame_rate": src_frame_rate, # RW; Range:[1, 240]; the input frame rate of the venc
channel
                "dst_frame_rate": dst_frame_rate, # RW; Range:[1, 240]; the target frame rate of the venc
channel,
                                                # can not be larger than src_frame_rate
                "max_bit_rate": max_bit_rate} # RW; Range:[2, 614400];the max bitrate}hi_venc_mjpeg_vbr;
```

10.8.8.4.16 hi_venc_mjpeg_fixqp

说明

定义MJPEG编码通道FIXQP属性字典。暂不支持。

定义

```
hi_venc_mjpeg_fixqp = {"src_frame_rate": src_frame_rate, # RW; Range:[1, 240]; the input frame rate of the
venc channel
                  "dst_frame_rate": dst_frame_rate, # RW; Range:[1, 240]; the target frame rate of the venc
channel,
                                                # can not be larger than src_frame_rate
                "qfactor": qfactor} # RW; Range:[1,99];image quality.
```

10.8.8.4.17 hi_venc_h265_cbr

说明

定义 H.265编码通道CBR属性字典。

定义

```
hi_venc_h265_cbr = {"gop": gop,
                    "stats_time": stats_time,
                    "src_frame_rate": src_frame_rate,
                    "dst_frame_rate": dst_frame_rate,
                    "bit_rate": bit_rate}
```

成员

成员名称	描述
gop	整型，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	整型，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。 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	整型，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	整型，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	整型，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10-28**所示。注意，**表10-28**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 10-28 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

10.8.8.4.18 hi_venc_h265_vbr

说明

定义 H.265编码通道VBR属性字典。

定义

```
hi_venc_h265_vbr = {"gop": gop,
                    "stats_time": stats_time,
                    "src_frame_rate": src_frame_rate,
                    "dst_frame_rate": dst_frame_rate,
                    "max_bit_rate": max_bit_rate}
```

成员

成员名称	描述
gop	整型，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	整型，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	整型，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	整型，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	整型，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。

10.8.8.4.19 hi_venc_h265_avbr

说明

定义H.265编码通道AVBR属性字典。

定义

```
hi_venc_h265_avbr = {"gop": gop,
                     "stats_time": stats_time,
                     "src_frame_rate": src_frame_rate,
                     "dst_frame_rate": dst_frame_rate,
                     "max_bit_rate": max_bit_rate }
```

成员

成员名称	描述
gop	整型，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	整型，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	整型，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	整型，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	整型，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]。

10.8.8.4.20 hi_venc_h265_qvbr

说明

定义H.265协议编码通道QVBR模式属性字典。

定义

```
hi_venc_h265_qvbr = {"gop": gop,
                     "stats_time": stats_time,
                     "src_frame_rate": src_frame_rate,
                     "dst_frame_rate": dst_frame_rate,
                     "target_bit_rate": target_bit_rate }
```

成员

成员名称	描述
gop	整型，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	整型，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	整型，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	整型，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	整型，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]。

10.8.8.4.21 hi_venc_h265_cvbr

说明

定义H.265协议编码通道CVBR模式属性字典。

定义

```
hi_venc_h265_cvbr = {"gop": gop,
                     "stats_time": stats_time,
                     "src_frame_rate": src_frame_rate,
                     "dst_frame_rate": dst_frame_rate,
                     "max_bit_rate": max_bit_rate,
                     "short_term_stats_time": short_term_stats_time,
                     "long_term_stats_time": long_term_stats_time,
                     "long_term_max_bit_rate": long_term_max_bit_rate,
                     "long_term_min_bit_rate": long_term_min_bit_rate}
```

成员

成员名称	描述
gop	整型，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	整型，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]。
src_frame_rate	整型，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]。

成员名称	描述
dst_frame_rate	整型，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	整型，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	整型，码率短期统计时间，以秒为单位。 取值范围：[1, 120]。
long_term_stats_time	整型，码率长期统计时间，默认为分钟。 取值范围：[1, 1440]。
long_term_max_bit_rate	整型，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	整型，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]。

10.8.8.4.22 hi_venc_h265_fixqp

说明

定义H.265编码通道FIXQP属性字典。

定义

```
hi_venc_h265_fixqp = {"gop": gop,
                      "src_frame_rate": src_frame_rate,
                      "dst_frame_rate": dst_frame_rate,
                      "i_qp": i_qp,
                      "p_qp": p_qp,
                      "b_qp": b_qp}
```

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	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。

10.8.8.4.23 hi_venc_h265_qpmap

说明

定义H.265编码通道QPMAP属性字典。暂不支持。

定义

```
hi_venc_h265_qpmap = {"gop": gop,
                      "stats_time": stats_time,
```

```
                "src_frame_rate": src_frame_rate,
                "dst_frame_rate": dst_frame_rate,
                "hi_venc_rc_qpmap_mode": hi_venc_rc_qpmap_mode}
```

10.8.8.4.24 hi_venc_gop_attr

说明

定义编码器GOP属性字典。

定义

```
hi_venc_gop_attr = {"gop_mode": hi_venc_gop_mode,
                    "normal_p": hi_venc_gop_normal_p}

hi_venc_gop_attr = {"gop_mode": hi_venc_gop_mode,
                    "dual_p": hi_venc_gop_dual_p }

hi_venc_gop_attr = {"gop_mode": hi_venc_gop_mode,
                    "smart_p": hi_venc_gop_smart_p }

hi_venc_gop_attr = {"gop_mode": hi_venc_gop_mode,
                    "adv_smart_p": hi_venc_gop_adv_smart_p}

hi_venc_gop_attr = {"gop_mode": hi_venc_gop_mode,
                    "bipred_b": hi_venc_gop_bipred_b}
```

成员

成员名称	描述
gop_mode	整型，编码GOP类型。
normal_p	字典，编码单参考帧P帧GOP属性字典。
dual_p	字典，编码双参考帧P帧GOP属性字典。
smart_p	字典，编码智能P帧GOP属性字典。
adv_smart_p	字典，编码高级智能P帧GOP属性字典，暂不支持。
bipred_b	字典，编码B帧GOP属性字典，暂不支持。

10.8.8.4.25 hi_venc_gop_mode

说明

定义 H.264/H.265 GOP类型。

定义

```
HI_VENC_GOP_MODE_NORMAL_P    = 0 # NORMALP
HI_VENC_GOP_MODE_DUAL_P      = 1 # DUALP
HI_VENC_GOP_MODE_SMART_P     = 2 # SMARTP
HI_VENC_GOP_MODE_ADV_SMART_P = 3 # ADVSMARTP
HI_VENC_GOP_MODE_BIPRED_B    = 4 # BIPREDB
```

```
HI_VENC_GOP_MODE_LOW_DELAY_B = 5 # LOWDELAYB
HI_VENC_GOP_MODE_BUTT        = 6
```

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。
HI_VENC_GOP_MODE_SMART_P	编码智能P帧GOP类型。
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。

10.8.8.4.26 hi_venc_gop_normal_p

说明

定义编码单参考帧P帧GOP属性字典。

定义

```
hi_venc_gop_normal_p = {"ip_qp_delta": ip_qp_delta}
```

成员

成员名称	描述
ip_qp_delta	整型，I帧相对P帧的QP（Quantisation Parameter）差值。 取值范围：[-10, 30]。

10.8.8.4.27 hi_venc_gop_dual_p

说明

定义编码双参考帧 P 帧 GOP 属性字典。

定义

```
hi_venc_gop_dual_p = {"sp_interval": sp_interval,
                      "sp_qp_delta": sp_qp_delta,
                      "ip_qp_delta": ip_qp_delta}
```

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]。

10.8.8.4.28 hi_venc_gop_smart_p

说明

定义编码智能 P 帧 GOP 属性字典。

定义

```
hi_venc_gop_smart_p = {"bg_interval": bg_interval,
                       "bg_qp_delta": bg_qp_delta,
                       "vi_qp_delta": vi_qp_delta}
```

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。
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]。

10.8.8.4.29 hi_venc_gop_adv_smart_p

说明

定义编码高级智能 P 帧 GOP 属性字典，暂不支持。

定义

```
hi_venc_gop_adv_smart_p= {"bg_interval": bg_interval,
                          "bg_qp_delta": bg_qp_delta,
                          "vi_qp_delta": vi_qp_delta}
```

10.8.8.4.30 hi_venc_gop_bipred_b

说明

定义编码B帧GOP属性字典，暂不支持。

定义

```
hi_venc_gop_bipred_b = {"b_frame_num": b_frame_num, # RW; Range:[1, 3]; Number of B frames
                        "b_qp_delta": b_qp_delta, # RW; Range:[-10, 30]; QP variance between P frame and B frame
                        "ip_qp_delta": ip_qp_delta} # RW; Range:[-10, 30]; QP variance between P frame and I
frame
```

10.8.8.4.31 hi_venc_start_param

说明

定义编码通道连续接收并编码的帧数字典。

定义

```
hi_venc_start_param = {"recv_pic_num": recv_pic_num}
```

成员

成员名称	描述
recv_pic_num	整型，编码通道连续接收并编码的帧数。 取值范围：-1或正整数。

10.8.8.4.32 hi_venc_chn_status

说明

定义编码通道的状态字典。

定义

```
hi_venc_chn_status = {"left_pics": left_pics, # R; left picture number
                      "left_stream_bytes": left_stream_bytes, # R; left stream bytes
                      "left_stream_frames": left_stream_frames, # R; left stream frames
                      "cur_packs": cur_packs, # R; pack number of current frame
                      "left_recv_pics": left_recv_pics, # R; Number of frames to be received.
                      "left_enc_pics": left_enc_pics, # R; Number of frames to be encoded.
                      "is_jpeg_snap_end": is_jpeg_snap_end, # R; the end of Snap.
                      "release_pic_pts": release_pic_pts,
                      "stream_info": hi_venc_stream_info}
```

成员

成员名称	描述
left_pics	整型，待编码的图像数，暂不支持。
left_stream_bytes	整型，码流buffer剩余的byte 数，暂不支持。
left_stream_frames	整型，码流buffer剩余的帧数，暂不支持。
cur_packs	整型，当前帧的码流包个数。
left_recv_pics	整型，剩余待接收的帧数，在用户调用接口**10.8.6.7 acl.himpi.venc_start_chn**设置接收帧数后有效，暂不支持。
left_enc_pics	整型，剩余待编码的帧数，在用户调用接口**10.8.6.7 acl.himpi.venc_start_chn**设置接收帧数后有效，暂不支持。
is_jpeg_snap_end	整型，Jpege 抓拍模式下指示抓拍过程是否结束，暂不支持。
release_pic_pts	整型，预留参数，暂不支持。
stream_info	整型，码流信息。

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。

10.8.8.4.33 hi_venc_h264_stream_info

说明

定义H.264协议码流特征信息。

定义

```
hi_venc_h264_stream_info = {"pic_bytes": pic_bytes,
                            "inter16x16_mb_num": inter16x16_mb_num,
                            "inter8x8_mb_num": inter8x8_mb_num,
                            "intra16_mb_num": intra16_mb_num,
                            "intra8_mb_num": intra8_mb_num,
                            "intra4_mb_num": intra4_mb_num,
                            "ref_type": hi_venc_ref_type,
                            "update_attr_cnt": update_attr_cnt,
                            "start_qp": start_qp,
                            "mean_qp": mean_qp,
                            "is_p_skip": is_p_skip }
```

成员

成员名称	描述
pic_bytes	整型，编码当前帧的字节（BYTE）数。
inter16x16_mb_num	整型，编码当前帧中采用 Inter16x16 预测模式的宏块数。暂不支持。
inter8x8_mb_num	整型，编码当前帧中采用 Inter8x8 预测模式的宏块数。暂不支持。
intra16_mb_num	整型，编码当前帧中采用 Intra16 预测模式的宏块数。暂不支持。
intra8_mb_num	整型，编码当前帧中采用 Intra8 预测模式的宏块数。暂不支持。
intra4_mb_num	整型，编码当前帧中采用 Intra4 预测模式的宏块数。暂不支持。
ref_type	整型，高级跳帧参考下的编码帧类型。暂不支持。
update_attr_cnt	整型，通道属性或参数(包含 RC 参数)被设置的次数。暂不支持。
start_qp	整型，编码当前帧的 startqp 值。
mean_qp	整型，编码当前帧的平均 QP 帧。
is_p_skip	整型，标识当前帧是否为 pskip 帧。暂不支持。

10.8.8.4.34 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

```
HI_VENC_BASE_IDR_SLICE                = 0 # the Idr frame at Base layer
HI_VENC_BASE_P_SLICE_REF_TO_IDR       = 1 # the P frame at Base layer referenced by other frames at Base layer and reference to Idr frame
HI_VENC_BASE_P_SLICE_REF_BY_BASE      = 2 # the P frame at Base layer referenced by other frames at Base layer
HI_VENC_BASE_P_SLICE_REF_BY_ENHANCE   = 3 # the P frame at Base layer referenced by other frames at Enhance layer
HI_VENC_ENHANCE_P_SLICE_REF_BY_ENHANCE = 4 #the P frame at Enhance layer referenced by other frames at Enhance layer
HI_VENC_ENHANCE_P_SLICE_NOT_FOR_REF   = 5 # the P frame at Enhance layer not referenced
HI_VENC_P_SLICE_BUTT                  = 6
```

10.8.8.4.35 hi_venc_jpeg_stream_info

说明

定义JPEG协议码流特征信息， 暂不支持。

定义

```
hi_venc_jpeg_stream_info = {"pic_bytes": pic_bytes,
                            "update_attr_cnt": update_attr_cnt,
                            "qfactor": qfactor}
```

成员

成员名称	描述
pic_bytes	整型，一帧JPEG码流大小，以字节（byte）为单位。
update_attr_cnt	整型，通道属性或参数（包含 RC 参数）被设置的次数。
qfactor	整型，编码当前帧的 Qfactor。

10.8.8.4.36 hi_venc_h265_stream_info

说明

定义H.265协议码流特征信息。

定义

```
hi_venc_h265_stream_info = {"pic_bytes": pic_bytes,
                            "inter64x64_cu_num": inter64x64_cu_num,
                            "inter32x32_cu_num": inter32x32_cu_num,
                            "inter16x16_cu_num": inter16x16_cu_num,
                            "inter8x8_cu_num": inter8x8_cu_num,
                            "intra32x32_cu_num": intra32x32_cu_num,
                            "intra16x16_cu_num": intra16x16_cu_num,
                            "intra8x8_cu_num": intra8x8_cu_num,
                            "intra4x4_cu_num": intra4x4_cu_num,
                            "ref_type": hi_venc_ref_type,
                            "update_attr_cnt": update_attr_cnt,
                            "start_qp": start_qp,
                            "mean_qp": mean_qp,
                            "is_p_skip": is_p_skip}
```

成员

成员名称	描述
pic_bytes	整型，编码当前帧的字节（BYTE）数。
inter64x64_cu_num	整型，编码当前帧中采用 Inter64x64 预测模式的 CU 块数。暂不支持。
inter32x32_cu_num	整型，编码当前帧中采用 Inter32x32 预测模式的 CU 块数。暂不支持。
inter16x16_cu_num	整型，编码当前帧中采用 Inter16x16 预测模式的 CU 块数。暂不支持。
inter8x8_cu_num	整型，编码当前帧中采用 Inter8x8 预测模式的 CU 块数。暂不支持。

成员名称	描述
intra32x32_cu_num	整型，编码当前帧中采用 Intra32x32 预测模式的 CU 块数。暂不支持。
intra16x16_cu_num	整型，编码当前帧中采用 Intra16x16 预测模式的 CU 块数。暂不支持。
intra8x8_cu_num	整型，编码当前帧中采用 Intra8x8 预测模式的 CU 块数。暂不支持。
intra4x4_cu_num	整型，编码当前帧中采用 Intra4x4 预测模式的 CU 块数。暂不支持。
ref_type	整型，高级跳帧参考下的编码帧类型。暂不支持。
update_attr_cnt	整型，通道属性或参数（包含 RC 参数）被设置的次数。暂不支持。
start_qp	整型，编码当前帧的 startqp 值。
mean_qp	整型，编码当前帧的平均 QP 值。
is_p_skip	整型，标识当前帧是否为 pskip 帧。暂不支持。

注意事项

ref_type请参见hi_venc_h264_stream_info中关于ref_type变量的说明。

10.8.8.4.37 hi_venc_prores_stream_info

说明

定义PRORES协议码流特征信息， 暂不支持。

定义

```
hi_venc_prores_stream_info= {"pic_bytes": pic_bytes,
                             "update_attr_cnt": update_attr_cnt}
```

成员

成员名称	描述
pic_bytes	整型，编码当前帧的字节（BYTE）数。
update_attr_cnt	整型，通道属性或参数被设置的次数。

10.8.8.4.38 hi_venc_mod_param

说明

编码相关模块参数。

定义

```
hi_venc_mod_param = {"mod_type": hi_venc_mod_type,
                     "venc_mod_param": hi_venc_venc_mod_param}

hi_venc_mod_param = {"mod_type": hi_venc_mod_type,
                     "h264_mod_param": hi_venc_h264_mod_param}

hi_venc_mod_param = {"mod_type": hi_venc_mod_type,
                     "h265_mod_param": hi_venc_h265_mod_param}

hi_venc_mod_param = {"mod_type": hi_venc_mod_type,
                     "jpeg_mod_param": hi_venc_jpeg_mod_param}
```

成员

成员名称	描述
mod_type	整型，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	字典，VENC内部各个模块的工作参数。

10.8.8.4.39 hi_venc_mod_type

说明

编码相关模块参数类型。

定义

```
HI_VENC_MOD_VENC = 1
HI_VENC_MOD_H264 = 2
HI_VENC_MOD_H265 = 3
HI_VENC_MOD_JPEG = 4
HI_VENC_MOD_RC   = 5
HI_VENC_MOD_BUTT = 6
```

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。
HI_VENC_MOD_BUTT	保留值。

10.8.8.4.40 hi_venc_venc_mod_param

说明

Device上的venc内公共模块参数。

定义

```
hi_venc_venc_mod_param = {"buf_cache": buf_cache,
                          "frame_buf_recycle": frame_buf_recycle}
```

成员

成员名称	描述
buf_cache	整型，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	整型，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。

10.8.8.4.41 hi_venc_h264_mod_param

说明

Device上的venc内h264e模块参数。

定义

```
hi_venc_h264_mod_param = {"one_stream_buf": one_stream_buf, # RW; Range:[0, 1]; one stream buffer
                          "mini_buf_mode": mini_buf_mode, # RW; Range:[0, 1]; H264e MiniBufMode
                          "low_power_mode": low_power_mode, # RW; Range:[0, 1]; H264e PowerSaveEn
                          "vb_src": hi_vb_src, # RW; H264e VBSource
                          "qp_hist_en": qp_hist_en, # RW; Range:[0, 1]
                          "max_user_data_len": max_user_data_len} # RW; Range:[0, 65536]; one user data buffer
len
```

成员

成员名称	描述
one_stream_buf	整型，编码码流帧配置模式，默认值为1。 ● 0：多包模式。 ● 1：单包模式。

成员名称	描述
mini_buf_mode	整型，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	整型，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：低功耗使能后会导致图像质量损失。
vb_src	整型，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	整型，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	整型，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。

10.8.8.4.42 hi_venc_h265_mod_param

说明

Device上的venc内h265e模块参数。

定义

```
hi_venc_h265_mod_param = {"one_stream_buf": one_stream_buf, # RW; Range:[0, 1]; one stream buffer
                          "mini_buf_mode": mini_buf_mode, # RW; Range:[0, 1]; H265e MiniBufMode
                          "low_power_mode": low_power_mode, # RW; Range:[0, 2]; H265e PowerSaveEn
                          "vb_src": hi_vb_src, # RW; H265e VBSource
                          "qp_hist_en": qp_hist_en, # RW; Range:[0, 1]
                          "max_user_data_len": max_user_data_len} # RW; Range:[0, 65536]; one user data buffer
len
```

成员

成员名称	描述
one_stream_buf	整型，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	整型，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	整型，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	整型，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	整型，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 。
max_user_data_len	整型，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。

10.8.8.4.43 hi_venc_jpeg_mod_param

说明

Device上的venc内jpege模块中参数。

定义

```
hi_venc_jpeg_mod_param = {"one_stream_buf": one_stream_buf, # RW; Range:[0, 1]; one stream buffer
                          "mini_buf_mode": mini_buf_mode, # RW; Range:[0, 1]; Jpege MiniBufMode
```

```
                    "clear_stream_buf": clear_stream_buf, # RW; Range:[0, 1]; JpegClearStreamBuf
                    "dering_mode": dering_mode} # RW; Range:[0, 1]; Jpege Dering Mode };
```

成员

成员名称	描述
one_stream_buf	整型，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	整型，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	整型，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。

10.8.8.4.44 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字典。

定义

```
hi_venc_jpeg_param = {"qfactor": qfactor, # RW; Range:0xFFFFFFFF or [1,100]; Qfactor value
                    "y_qt": y_qt[64], # RW; Range:[1, 255]; Y quantization table
                    "cb_qt": cb_qt[64], # RW; Range:[1, 255]; Cb quantization table
                    "cr_qt": cr_qt[64], # RW; Range:[1, 255]; Cr quantization table
                    "mcu_per_ecs": mcu_per_ecs, # RW; the max MCU number is (picwidth + 15) >> 4 x
(picheight +15) >> 4 x 2]; MCU number of one ECS
                    "ecs_output_en": ecs_output_en}
```

成员

成员名称	描述
qfactor	整型，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	列表，Y 量化表。 取值范围：[1, 255]。
cb_qt	列表，Cb 量化表。 取值范围：[1, 255]。
cr_qt	列表，Cr 量化表。 取值范围：[1, 255]。
mcu_per_ecs	整型，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 x(picheight+15)>>4 x 2]
ecs_output_en	整型，预留参数，暂不支持。

10.8.8.4.45 hi_venc_h264_adv_stream_info

说明

定义H.264协议码流高级特征信息。

定义

```
hi_venc_h264_adv_stream_info = {"residual_bits": residual_bits, # R; the residual num
                                "head_bits": head_bits, # R; the head bit num
                                "madi_val": madi_val, # R; the madi value
                                "madp_val": madp_val, # R; the madp value
                                "psnr_val": psnr_val, # R; the PSNR value
                                "sse_lcu_cnt": sse_lcu_cnt, # R; the lcu cnt of the sse
                                "sse_sum": sse_sum, R; the sum of the sse
                                "sse_info": [hi_venc_sse_info], # R; the information of the sse
                                "qp_hist": [qp_hist], # R; the Qp histogram value
                                "move_scene16x16_num": move_scene16x16_num, # R; the 16x16 cu num of the
move scene
                                "move_scene_bits": move_scene_bits} # R; the stream bit num of the move scene
```

成员

成员名称	描述
residual_bits	整型，编码当前帧残差（bit）数。暂不支持。
head_bits	整型，编码当前帧头信息的（bit）数。暂不支持。
madi_val	整型，编码当前帧空域纹理复杂度Madi值。暂不支持。
madp_val	整型，编码当前帧时域运动复杂度Madp值。暂不支持。
psnr_val	整型，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小于或等于3840*2160分辨率时获取psnr_val值，否则psnr_val值无效。
sse_lcu_cnt	整型，编码当前帧中LCU个数。
sse_sum	整型，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	列表，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。
qp_hist	列表，编码当前帧Qp直方图。暂不支持。
move_scene16x16_num	整型，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。暂不支持。
move_scene_bits	整型，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。暂不支持。

10.8.8.4.46 hi_venc_h265_adv_stream_info

说明

定义H.265协议码流高级特征信息。

定义

```
hi_venc_h265_adv_stream_info = {"residual_bits": residual_bits, # R; the residual num
                                "head_bits": head_bits, # R; the head bit num
                                "madi_val": madi_val, # R; the madi value
                                "madp_val": madp_val, # R; the madp value
                                "psnr_val": psnr_val, # R; the PSNR value
                                "sse_lcu_cnt": sse_lcu_cnt, # R; the lcu cnt of the sse
                                "sse_sum": sse_sum, R; the sum of the sse
```

```
                    "sse_info": [hi_venc_sse_info], # R; the information of the sse
                    "qp_hist": [qp_hist], # R; the Qp histogram value
                    "move_scene32x32_num": move_scene32x32_num, # R; the 32x32 cu num of the
move scene
                    "move_scene_bits": move_scene_bits} # R; the stream bit num of the move scene
```

成员

成员名称	描述
residual_bits	整型，编码当前帧残差（bit）数 。暂不支持。
head_bits	整型，编码当前帧头信息的（bit）数 。暂不支持。
madi_val	整型，编码当前帧空域纹理复杂度Madi值 。暂不支持。
madp_val	整型，编码当前帧时域运动复杂度Madp值。暂不支持。
psnr_val	整型，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小于或等于3840*2160分辨率时获取psnr_val值，否则psnr_val值无效。
sse_lcu_cnt	整型，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	列表，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。
qp_hist	列表，编码当前帧Qp直方图。暂不支持。
move_scene32x32_num	整型，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。暂不支持。
move_scene_bits	整型，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。暂不支持。

10.8.8.4.47 hi_venc_data_type

说明

定义码流结果类型。

定义

```
hi_venc_data_type = {"h264_type": hi_venc_h264_nalu_type, # R; H264E NALU types
                     "jpeg_type": hi_venc_jpege_pack_type, # R; JPEGE pack types
```

```
                "h265_type": hi_venc_h265_nalu_type, # R; H265E NALU types
                "prores_type": hi_venc_prores_pack_type}
```

成员

成员名称	描述
h264_type	整型，H.264 码流包类型。
jpeg_type	整型, JPEG 码流包类型。
h265_type	整型，H.265 码流包类型。
prores_type	整型，PRORES 码流包类型。

10.8.8.4.48 hi_venc_h264_nalu_type

说明

定义H.264码流NALU类型。

定义

```
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      = 9
```

10.8.8.4.49 hi_venc_jpege_pack_type

说明

定义JPEG码流的PACK类型。

定义

```
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    = 11
```

10.8.8.4.50 hi_venc_h265_nalu_type

说明

定义H.265码流NALU类型。

定义

```
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
```

10.8.8.4.51 hi_venc_prores_pack_type

说明

定义PRORES码流的PACK类型，暂不支持。

定义

```
HI_VENC_PRORES_PACK_PIC = 1 # PIC types
HI_VENC_PRORES_PACK_BUTT = 2
```

10.8.8.4.52 hi_venc_chn_param

说明

定义Venc通道参数字典。

定义

```
hi_venc_chn_param = {"color_to_grey_en": color_to_grey_en,
                     "priority": priority,
                     "max_stream_cnt": max_stream_cnt,
                     "poll_wake_up_frame_cnt": poll_wake_up_frame_cnt,
                     "crop_info": hi_crop_info,
                     "frame_rate": hi_frame_rate_ctrl}
```

成员

成员名称	描述
color_to_grey_en	整型，开启或关闭一个通道的彩转灰功能。
priority	整型，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	整型，最大码流缓存帧数，暂不支持。
poll_wake_up_frame_cnt	整型，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	字典，预留参数，暂不支持。
frame_rate	字典，通道帧率控制参数，暂不支持。

10.8.8.4.53 hi_venc_stream_info

说明

定义码流信息字典。

定义

```
hi_venc_stream_info = {"ref_type": hi_venc_ref_type, # R;Type of encoded frames in advanced frame
skipping reference mode
                "pic_bytes": pic_bytes, # R;the coded picture stream byte number
                "pic_cnt": pic_cnt,# R;When channel attributes 'is_by_frame == 1', it means count of frames.
                            # When channel attributes 'is_by_frame == 0', it means count of packets
                "start_qp": start_qp, # R;the start Qp of encoded frames
                "mean_qp": mean_qp, # R;the mean Qp of encoded frames
                "is_p_skip": is_p_skip,
                "residual_bits": residual_bits, # R;residual
                "head_bits": head_bits, # R;head information
                "madi_val": madi_val, # R;madi
                "madp_val": madp_val, # R;madp
                "sse_sum": sse_sum, # R;Sum of MSE value
                "sse_lcu_cnt": sse_lcu_cnt, # R;Sum of LCU number
                "psnr_val": psnr_val} # R;PSNR
```

成员

成员名称	描述
ref_type	整型，高级跳帧参考下的编码帧类型。暂不支持。
pic_bytes	整型，已编码码流的字节数。
pic_cnt	整型，当通道属性is_by_frame == 1时，表示帧数；当is_by_frame == 0时，表示包数。暂不支持。
start_qp	整型，编码帧的起始QP。
mean_qp	整型，编码帧的平均QP。
is_p_skip	整型，是否P帧是skip帧。暂不支持。
residual_bits	整型，编码当前帧残差bit数。暂不支持。
head_bits	整型，编码当前帧头信息bit数。暂不支持。
madi_val	整型，编码当前帧空域纹理复杂度Madi值。暂不支持。
madp_val	整型，编码当前帧时域运动复杂度Madp值。暂不支持。
sse_sum	整型，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	整型，编码当前帧中LCU个数。

成员名称	描述
psnr_val	整型，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小于或等于3840*2160分辨率时获取psnr_val值，否则psnr_val值无效。

10.8.8.4.54 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字典。

定义

```
hi_venc_pack_info = {"pack_type": hi_venc_data_type, # R; the pack type
                     "pack_offset": pack_offset,
                     "pack_len": pack_len}
```

成员

成员名称	描述
pack_type	整型，当前码流包数据包含其他码流包的类型。
pack_offset	整型，当前码流包数据包含其他码流包数据的偏移。
pack_len	整型，当前码流包数据包含其他码流包数据的大小。

10.8.8.4.55 hi_frame_rate_ctrl

说明

定义通道帧率控制参数，暂不支持。

定义

```
hi_frame_rate_ctrl = {"src_frame_rate": src_frame_rate, # RW; Range:[0, 240]; Input frame rate of a channel
                      "dst_frame_rate": dst_frame_rate} # RW; Range:[0, 240]; Output frame rate of a channel}
```

成员

成员名称	描述
src_frame_rate	整型，进入通道的帧率，以fps为单位。 取值范围：-1 或[1, 240]。
dst_frame_rate	整型，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]。

10.8.8.4.56 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

```
HI_VENC_RC_QPMAP_MODE_MEAN_QP = 0
HI_VENC_RC_QPMAP_MODE_MIN_QP  = 1
HI_VENC_RC_QPMAP_MODE_MAX_QP  = 2
HI_VENC_RC_QPMAP_MODE_BUTT    = 3
```

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。

10.8.8.4.57 hi_venc_sse_info

说明

定义H264/H265协议SSE信息，暂不支持。

定义

```
hi_venc_sse_info = {"enable": enable, # RW; Range:[0,1]; Region SSE enable
                    "sse_val": sse_val} # R; Region SSE value
```

成员

成员名称	描述
enable	整型，区域SSE使能。1表示使能，0表示不使能
sse_val	整型，区域SSE值。

10.8.8.4.58 hi_venc_h264_attr

说明

定义H.264编码属性字典，

定义

```
hi_venc_h264_attr = {"rcn_ref_share_buf_en": rcn_ref_share_buf_en, # RW; Range:[0, 1];Whether to enable the Share Buf of Rcn and Ref
                     "frame_buf_ratio": frame_buf_ratio}
```

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。

10.8.8.4.59 hi_venc_h265_attr

说明

定义H.265编码属性字典。

定义

```
hi_venc_h265_attr = {"rcn_ref_share_buf_en": rcn_ref_share_buf_en, # RW; Range:[0, 1];Whether to enable the Share Buf of Rcn and Ref
                     "frame_buf_ratio": frame_buf_ratio}
```

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。

10.8.8.4.60 hi_venc_pic_recv_mode

说明

定义JPEG编码属性字典，暂不支持。

定义

```
HI_VENC_PIC_RECV_SINGLE = 0
HI_VENC_PIC_RECV_MULTI  = 1
HI_VENC_PIC_RECV_BUTT   = 2
```

10.8.8.4.61 hi_venc_jpeg_attr

说明

定义JPEG编码属性字典，暂不支持。

定义

```
hi_venc_jpeg_attr = {"dcf_en": dcf_en, # RW; Range:[0, 1]; support dcf
                     "mpf_cfg": hi_venc_mpf_cfg, # RW; Range:[0, 1]; config of Mpf
                     "recv_mode": hi_venc_pic_recv_mode} # RW; Config the receive mode;
```

10.8.8.4.62 hi_venc_prores_frame_rate

说明

PRORES帧率全局，暂不支持。

定义

```
HI_VENC_PRORES_FRAME_RATE_UNKNOWN = 0
HI_VENC_PRORES_FRAME_RATE_23_976  = 1
HI_VENC_PRORES_FRAME_RATE_24      = 2
HI_VENC_PRORES_FRAME_RATE_25      = 3
HI_VENC_PRORES_FRAME_RATE_29_97   = 4
HI_VENC_PRORES_FRAME_RATE_30      = 5
HI_VENC_PRORES_FRAME_RATE_50      = 6
HI_VENC_PRORES_FRAME_RATE_59_94   = 7
HI_VENC_PRORES_FRAME_RATE_60      = 8
HI_VENC_PRORES_FRAME_RATE_100     = 9
HI_VENC_PRORES_FRAME_RATE_119_88  = 10
HI_VENC_PRORES_FRAME_RATE_120     = 11
HI_VENC_PRORES_FRAME_RATE_BUTT    = 12
```

10.8.8.4.63 hi_venc_prores_aspect_ratio

说明

PRORES宽高比格式，暂不支持。

定义

```
# the aspect ratio of PRORES
HI_VENC_PRORES_ASPECT_RATIO_UNKNOWN = 0
HI_VENC_PRORES_ASPECT_RATIO_SQUARE  = 1
HI_VENC_PRORES_ASPECT_RATIO_4_3     = 2
HI_VENC_PRORES_ASPECT_RATIO_16_9    = 3
HI_VENC_PRORES_ASPECT_RATIO_BUTT    = 4
```

10.8.8.4.64 hi_venc_prores_attr

说明

定义PRORES属性字典，暂不支持。

定义

```
hi_venc_prores_attr = {"identifier": [identifier],
                       "frame_rate_code": hi_venc_prores_frame_rate,
                       "aspect_ratio": hi_venc_prores_aspect_ratio}
```

10.8.8.4.65 hi_crop_info

说明

预留字典，暂不支持。

定义

```
hi_crop_info= {"enable": enable, # RW; Range:[0, 1]; Crop region enable
               "stRect": hi_rect} # RW; Crop region, note: s32X must be multi of 16
```

10.8.8.4.66 hi_venc_scene_mode

说明

定义编码模式场景。

定义

```
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 = 3
```

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。

10.8.8.4.67 hi_venc_rc_param

说明

定义编码通道码率控制器的高级参数。

定义

```
hi_venc_rc_param = {"threshold_i": threshold_i[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                    "threshold_p": threshold_p[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                    "threshold_b": threshold_b[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                    "direction": direction,
                    "row_qp_delta": row_qp_delta,
                    "first_frame_start_qp": first_frame_start_qp,
                    "scene_chg_detect": scene_chg_detect,
                    "h264_cbr_param": hi_venc_h264_cbr_param }

hi_venc_rc_param = {"threshold_i": threshold_i[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                    "threshold_p": threshold_p[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                    "threshold_b": threshold_b[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                    "direction": direction,
                    "row_qp_delta": row_qp_delta,
                    "first_frame_start_qp": first_frame_start_qp,
                    "scene_chg_detect": scene_chg_detect,
                    "h264_vbr_param":  hi_venc_h264_vbr_param}

hi_venc_rc_param = {"threshold_i": threshold_i[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                    "threshold_p": threshold_p[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                    "threshold_b": threshold_b[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                    "direction": direction,
                    "row_qp_delta": row_qp_delta,
                    "first_frame_start_qp": first_frame_start_qp,
                    "scene_chg_detect": scene_chg_detect,
                    "h264_avbr_param": hi_venc_h264_avbr_param}

hi_venc_rc_param = {"threshold_i": threshold_i[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                    "threshold_p": threshold_p[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                    "threshold_b": threshold_b[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                    "direction": direction,
                    "row_qp_delta": row_qp_delta,
                    "first_frame_start_qp": first_frame_start_qp,
                    "scene_chg_detect": scene_chg_detect,
                    "h264_qvbr_param": hi_venc_h264_qvbr_param}

hi_venc_rc_param = {"threshold_i": threshold_i[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                    "threshold_p": threshold_p[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                    "threshold_b": threshold_b[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                    "direction": direction,
```

```
                "row_qp_delta": row_qp_delta,
                "first_frame_start_qp": first_frame_start_qp,
                "scene_chg_detect": scene_chg_detect,
                "h264_cvbr_param": hi_venc_h264_cvbr_param}

hi_venc_rc_param = {"threshold_i": threshold_i[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                "threshold_p": threshold_p[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                "threshold_b": threshold_b[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                "direction": direction,
                "row_qp_delta": row_qp_delta,
                "first_frame_start_qp": first_frame_start_qp,
                "scene_chg_detect": scene_chg_detect,
                "h265_cbr_param": hi_venc_h265_cbr_param}

hi_venc_rc_param = {"threshold_i": threshold_i[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                "threshold_p": threshold_p[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                "threshold_b": threshold_b[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                "direction": direction,
                "row_qp_delta": row_qp_delta,
                "first_frame_start_qp": first_frame_start_qp,
                "scene_chg_detect": scene_chg_detect,
                "h265_vbr_param": hi_venc_h265_vbr_param}

hi_venc_rc_param = {"threshold_i": threshold_i[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                "threshold_p": threshold_p[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                "threshold_b": threshold_b[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                "direction": direction,
                "row_qp_delta": row_qp_delta,
                "first_frame_start_qp": first_frame_start_qp,
                "scene_chg_detect": scene_chg_detect,
                "h265_avbr_param": hi_venc_h265_avbr_param}

hi_venc_rc_param = {"threshold_i": threshold_i[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                "threshold_p": threshold_p[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                "threshold_b": threshold_b[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                "direction": direction,
                "row_qp_delta": row_qp_delta,
                "first_frame_start_qp": first_frame_start_qp,
                "scene_chg_detect": scene_chg_detect,
                "h265_qvbr_param": hi_venc_h265_qvbr_param}

hi_venc_rc_param = {"threshold_i": threshold_i[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                "threshold_p": threshold_p[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                "threshold_b": threshold_b[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                "direction": direction,
                "row_qp_delta": row_qp_delta,
                "first_frame_start_qp": first_frame_start_qp,
                "scene_chg_detect": scene_chg_detect,
                "h265_cvbr_param": hi_venc_h265_cvbr_param}

hi_venc_rc_param = {"threshold_i": threshold_i[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                "threshold_p": threshold_p[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                "threshold_b": threshold_b[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                "direction": direction,
                "row_qp_delta": row_qp_delta,
                "first_frame_start_qp": first_frame_start_qp,
                "scene_chg_detect": scene_chg_detect,
                "mjpeg_cbr_param": hi_venc_mjpeg_cbr_param}

hi_venc_rc_param = {"threshold_i": threshold_i[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                "threshold_p": threshold_p[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                "threshold_b": threshold_b[HI_VENC_TEXTURE_THRESHOLD_SIZE],
                "direction": direction,
                "row_qp_delta": row_qp_delta,
                "first_frame_start_qp": first_frame_start_qp,
                "scene_chg_detect": scene_chg_detect,
                "mjpeg_vbr_param": hi_venc_mjpeg_vbr_param}
```

成员

成员名称	描述
threshold_i	列表， 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	列表， 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	列表， 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	整型，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	整型，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字典内设置。 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	整型，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	字典，H.264通道CBR（Constant Bit Rate）码率控制模式高级参数。
h264_vbr_param	字典，H.264通道VBR（Variable Bit Rate）码率控制模式高级参数。
h264_avbr_param	字典，H.264通道AVBR（Adaptive Variable Bit Rate）码率控制模式高级参数。
h264_qvbr_param	字典，H.264通道QVBR (Quality Variable Bit Rate)码率控制模式高级参数。
h264_cvbr_param	字典，H.264通道CVBR (Constrained Variable Bit Rate)码率控制模式高级参数。
h265_cbr_param	字典，H.265通道CBR（Constant Bit Rate）码率控制模式高级参数。
h265_vbr_param	字典，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	字典，H.265通道AVBR（Adaptive Variable Bit Rate）码率控制模式高级参数。
h265_qvbr_param	字典，H.265通道QVBR (Quality Variable Bit Rate)码率控制模式高级参数。
h265_cvbr_param	字典，H.265通道CVBR (Constrained Variable Bit Rate)码率控制模式高级参数。
mjpeg_cbr_param	字典，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	字典，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。

10.8.8.4.68 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

```
hi_venc_h264_cbr_param = {"max_i_proportion": max_i_proportion,
                          "min_i_proportion": min_i_proportion,
                          "max_qp": max_qp,
                          "min_qp": min_qp,
                          "max_i_qp": max_i_qp,
                          "min_i_qp": min_i_qp,
                          "max_reencode_times": max_reencode_times,
                          "qpmap_en": qpmap_en}
```

成员

成员名称	描述
min_i_proportion	整型，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	整型，帧最大QP，用于钳位质量。 取值范围：[0, 51]。 默认值：51。
min_qp	整型，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
max_reencode_times	整型，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_i_qp	整型，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	整型，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。
qpmap_en	整型，预留参数，暂不支持。

10.8.8.4.69 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

```
hi_venc_h264_vbr_param = {"chg_pos": chg_pos,
                          "max_i_proportion": max_i_proportion,
                          "min_i_proportion": min_i_proportion,
                          "max_reencode_times": max_reencode_times,
                          "qpmap_en": qpmap_en,
                          "max_qp": max_qp,
                          "min_qp": min_qp,
                          "max_i_qp": max_i_qp,
                          "min_i_qp": min_i_qp}
```

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内的成员变量。

成员名称	描述
chg_pos	整型，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	整型，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	整型，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	整型，预留参数，暂不支持。
max_qp	整型，P、B帧的最大QP。 取值范围：[0, 51]。 默认值：51。
min_qp	整型，P、B帧的最小QP。 取值范围：[0, max_qp]。 默认值：24。
max_i_qp	整型，I帧的最大QP。 取值范围：[0, 51]。 默认值：51。
min_i_qp	整型，I帧的最小QP。 取值范围：[0, max_i_qp]。 默认值：24。

10.8.8.4.70 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

```
hi_venc_h264_avbr_param = {"chg_pos": chg_pos,
                           "max_i_proportion": max_i_proportion,
                           "min_i_proportion": min_i_proportion,
                           "max_reencode_times": max_reencode_times,
                           "qpmap_en": qpmap_en,
                           "min_still_percent": min_still_percent,
                           "max_still_qp": max_still_qp,
```

```
                "min_still_psnr": min_still_psnr,
                "max_qp": max_qp,
                "min_qp": min_qp,
                "max_i_qp": max_i_qp,
                "min_i_qp": min_i_qp,
                "min_qp_delta": min_qp_delta,
                "motion_sensitivity": motion_sensitivity,
                "save_bitrate_en": save_bitrate_en}
```

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内的成员变量。

成员名称	描述
chg_pos	整型，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	整型，预留参数，暂不支持。
max_i_proportion	整型，最大IP帧码率的比值。 取值范围：[1,100]。 默认值：100。
max_reencode_times	整型，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	整型，预留参数，暂不支持。
min_still_percent	整型，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	整型，静止场景I帧QP的最大值。 取值范围：[MinIQp, MaxIQp]。 默认值：35。
min_still_psnr	整型，预留参数，暂不支持。

成员名称	描述
max_qp	整型，P、B帧的最大QP。 取值范围：[0, 51]。 默认值：51。
min_qp	整型，P、B帧的最小QP。 取值范围：[0, max_qp]。 默认值：24。
max_i_qp	整型，I帧的最大QP。 取值范围：[0, 51]。 默认值：51。
min_i_qp	整型，I帧的最小QP。 取值范围：[0, max_i_qp]。 默认值：24。
min_qp_delta	整型，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	整型，运动敏感度。 取值范围：[0, 100]。 默认值：100。
save_bitrate_en	整型，预留参数，暂不支持。

10.8.8.4.71 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

```
hi_venc_h264_qvbr_param = {"max_i_proportion": max_i_proportion,
                           "min_i_proportion": min_i_proportion,
                           "max_reencode_times": max_reencode_times,
                           "qpmap_en": qpmap_en,
                           "max_qp": max_qp,
                           "min_qp": min_qp,
                           "max_i_qp": max_i_qp,
                           "min_i_qp": min_i_qp,
                           "max_bit_percent": max_bit_percent,
                           "min_bit_percent": min_bit_percent,
```

```
                    "max_psnr_fluctuate": max_psnr_fluctuate,
                    "min_psnr_fluctuate": min_psnr_fluctuate}
```

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内的成员变量。

成员名称	描述
min_i_proportion	整型，预留参数，暂不支持。
max_i_proportion	整型，最大IP帧码率的比值。 取值范围：[1,100]。 默认值：100。
max_reencode_times	整型，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	整型，预留参数，暂不支持。
max_qp	整型，P、B帧的最大QP。 取值范围：[MinQp, 51]。 默认值：51。
min_qp	整型，P、B帧的最小QP。 取值范围：[0, 51]。 默认值：16。
max_i_qp	整型，I帧的最大QP。 取值范围：[MinIQp, 51]。 默认值：51。
min_i_qp	整型，I帧的最小QP。 取值范围：[0, 51]。 默认值：16。

成员名称	描述
max_bit_percent	整型，码率百分比上限。 取值范围：[min_bit_percent, 180]。 默认值：110。
min_bit_percent	整型，码率百分比下限。 取值范围：[30, 180]。 默认值：45。
max_psnr_fluctuate	整型，Psnr上限。 取值范围：[min_psnr_fluctuate, 40]。 默认值：40。
min_psnr_fluctuate	整型，Psnr下限。 取值范围：[18, 40]。 默认值：23。

10.8.8.4.72 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

```
hi_venc_h264_cvbr_param = {"max_i_proportion": max_i_proportion,
                           "min_i_proportion": min_i_proportion,
                           "max_reencode_times": max_reencode_times,
                           "qpmap_en": qpmap_en,
                           "max_qp": max_qp,
                           "min_qp": min_qp,
                           "max_i_qp": max_i_qp,
                           "min_i_qp": min_i_qp,
                           "min_qp_delta": min_qp_delta,
                           "max_qp_delta": max_qp_delta,
                           "extra_bit_percent": extra_bit_percent,
                           "long_term_stats_time_unit": long_term_stats_time_unit,
                           "save_bitrate_en": save_bitrate_en}
```

成员

成员名称	描述
min_i_proportion	整型，预留参数，暂不支持。
max_i_proportion	整型，最大IP帧码率的比值。 取值范围：[1,100]。 默认值：100。

成员名称	描述
max_reencode_times	整型，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	整型，预留参数，暂不支持。
max_qp	整型，P、B帧的最大QP。 取值范围：[min_qp, 51]。 默认值：47。
min_qp	整型，P、B帧的最小QP。 取值范围：[0, 51]。 默认值：22。
max_i_qp	整型，I帧的最大QP。 取值范围：[min_i_qp, 51]。 默认值：47。
min_i_qp	整型，I 帧的最小QP。 取值范围：[0, 51]。 默认值：20。
min_qp_delta	整型，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	整型，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	整型，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	整型，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	整型，预留参数，暂不支持。

10.8.8.4.73 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

```
hi_venc_h264_cvbr_param = {"chg_pos": chg_pos,
                           "max_i_proportion": max_i_proportion,
                           "min_i_proportion": min_i_proportion,
                           "max_reencode_times": max_reencode_times,
                           "max_qp": max_qp,
                           "min_qp": min_qp,
                           "max_i_qp": max_i_qp,
                           "min_i_qp": min_i_qp,
                           "qpmap_en": qpmap_en,
                           "qpmap_mode": hi_venc_rc_qpmap_mode}
```

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内的成员变量。

成员名称	描述
chg_pos	整型，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	整型，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	整型，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	整型，P帧的最大QP。 取值范围：[0, 51]。 默认值：51。
min_qp	整型，P帧的最小QP。 取值范围：[0, max_qp]。 默认值：24。
max_i_qp	整型，I帧的最大QP。 取值范围：[0, 51]。 默认值：51。
min_i_qp	整型，I帧的最小QP。 取值范围：[0, max_i_qp]。 默认值：24。
qpmap_en	整型，预留参数，暂不支持。
qpmap_en	整型，预留参数，暂不支持。

10.8.8.4.74 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

```
hi_venc_h264_cvbr_param = {"chg_pos": chg_pos,
                           "max_i_proportion": max_i_proportion,
                           "min_i_proportion": min_i_proportion,
                           "max_reencode_times": max_reencode_times,
                           "min_still_percent": min_still_percent,
```

```
                "max_still_qp": max_still_qp,
                "min_still_psnr": min_still_psnr,
                "max_qp": max_qp,
                "min_qp": min_qp,
                "max_i_qp": max_i_qp,
                "min_i_qp": min_i_qp,
                "min_qp_delta": min_qp_delta,
                "motion_sensitivity": motion_sensitivity,
                "qpmap_en": qpmap_en,
                "qpmap_mode": hi_venc_rc_qpmap_mode}
```

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内的成员变量。

成员名称	描述
chg_pos	整型，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	整型，预留参数，暂不支持。
max_i_proportion	整型，最大IP帧码率的比值。 取值范围：[1,100]。 默认值：100。
max_reencode_times	整型，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	整型，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	整型，静止场景I帧QP的最大值。 取值范围：[MinIQp, MaxIQp]。 默认值：35。
min_still_psnr	整型，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	整型，P帧的最小QP。 取值范围：[0, max_qp]。 默认值：24。
max_i_qp	整型，I帧的最大QP。 取值范围：[0, 51]。 默认值：51。
min_i_qp	整型，I帧的最小QP。 取值范围：[0, max_i_qp]。 默认值：24。
min_qp_delta	整型，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	整型，运动敏感度。 取值范围：[0, 100]。 默认值：100。
qpmap_en	整型，预留参数，暂不支持。
qpmap_en	整型，预留参数，暂不支持。

10.8.8.4.75 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

```
hi_venc_h265_qvbr_param = {"max_i_proportion": max_i_proportion,
                           "min_i_proportion": min_i_proportion,
                           "max_reencode_times": max_reencode_times,
                           "qpmap_en": qpmap_en,
                           "qpmap_mode": hi_venc_rc_qpmap_mode,
                           "max_qp": max_qp,
                           "min_qp": min_qp,
                           "max_i_qp": max_i_qp,
```

```
                "min_i_qp": min_i_qp,
                "max_bit_percent": max_bit_percent,
                "min_bit_percent": min_bit_percent,
                "max_psnr_fluctuate": max_psnr_fluctuate,
                "min_psnr_fluctuate": min_psnr_fluctuate}
```

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内的成员变量。

成员名称	描述
min_i_proportion	整型，预留参数，暂不支持。
max_i_proportion	整型，最大IP 帧码率的比值。 取值范围：[1,100]。 默认值：100。
max_reencode_times	整型，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	整型，预留参数，暂不支持。
qpmap_en	整型，预留参数，暂不支持。
max_qp	整型，P、B帧的最大QP。 取值范围：[MinQp, 51]。 默认值：51。
min_qp	整型，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	整型，I帧的最大QP。 取值范围：[MinIQp, 51]。 默认值：51。
min_i_qp	整型，I帧的最小QP。 取值范围：[0, 51]。 默认值：16。
max_bit_percent	整型，码率百分比上限。 取值范围：[min_bit_percent, 180]。 默认值：110。
min_bit_percent	整型，码率百分比下限。 取值范围：[30, 180]。 默认值：45。
max_psnr_fluctuate	整型，Psnr 上限。 取值范围：[min_psnr_fluctuate, 40]。 默认值：40。
min_psnr_fluctuate	整型，Psnr下限。 取值范围：[18, 40]。 默认值：23。

10.8.8.4.76 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

```
hi_venc_h265_qvbr_param = {"max_i_proportion": max_i_proportion,
                           "min_i_proportion": min_i_proportion,
                           "max_reencode_times": max_reencode_times,
                           "qpmap_en": qpmap_en,
                           "qpmap_mode": hi_venc_rc_qpmap_mode,
                           "max_qp": max_qp,
                           "min_qp": min_qp,
                           "max_i_qp": max_i_qp,
                           "min_i_qp": min_i_qp,
                           "min_qp_delta": min_qp_delta,
                           "max_qp_delta": max_qp_delta,
                           "extra_bit_percent": extra_bit_percentm,
                           "long_term_stats_time_unit": long_term_stats_time_unit}
```

成员

成员名称	描述
min_i_proportion	整型，预留参数，暂不支持。
max_i_proportion	整型，最大IP 帧码率的比值。 取值范围：[1,100]。 默认值：100。
max_reencode_times	整型，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	整型，预留参数，暂不支持。
qpmap_en	整型，预留参数，暂不支持。
max_qp	整型，P、B帧的最大QP。 取值范围：[MinQp, 51]。 默认值：47。
min_qp	整型，P、B帧的最小QP。 取值范围：[0, 51]。 默认值：22。
max_i_qp	整型，I帧的最大QP。 取值范围：[MinIQp, 51]。 默认值：47。
min_i_qp	整型，I帧的最小QP。 取值范围：[0, 51]。 默认值：20。
min_qp_delta	整型，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。

成员名称	描述
max_qp_delta	整型，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。
extra_bit_percent	整型，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	整型，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。

10.8.8.4.77 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

```
hi_venc_h265_qvbr_param = {"max_i_proportion": max_i_proportion,
                           "min_i_proportion": min_i_proportion,
                           "max_qp": max_qp,
                           "min_qp": min_qp,
                           "max_i_qp": max_i_qp,
                           "min_i_qp": min_i_qp,
                           "max_reencode_times": max_reencode_times,
                           "qpmap_en": qpmap_en,
                           "qpmap_mode": hi_venc_rc_qpmap_mode}
```

成员

成员名称	描述
min_i_proportion	整型，预留参数，暂不支持。

成员名称	描述
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	整型，帧最大QP，用于钳位质量。 取值范围：[0, 51]。 默认值：51。
min_qp	整型，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	整型，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	整型，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	整型，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	整型，预留参数，暂不支持。
qpmap_en	整型，预留参数，暂不支持

10.8.8.4.78 hi_venc_scene_chg_detect

说明

定义编码场景检测控制参数。

定义

```
hi_venc_scene_chg_detect = {"detect_scene_chg_en": detect_scene_chg_en,
                            "adapt_insert_idr_frame_en": adapt_insert_idr_frame_en}
```

成员

成员名称	描述
detect_scene_chg_en	整型，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	整型，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

10.8.8.4.79 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

```
hi_venc_huffman_dc_table = {"dc_bits": dc_bits[16], # RW; Range:[0, 255]; len: 16
                            "dc_value": dc_value[12]} # RW; Range:[0, 255];len 12
```

成员

成员名称	描述
dc_bits[]	列表, 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[]	列表，dc_value数据存放的是码字对应的符号值。

10.8.8.4.80 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

```
hi_venc_huffman_ac_table = {"ac_bits": ac_bits[16],
                            "ac_value": ac_value[162]}
```

成员

成员名称	描述
ac_bits	列表，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	列表，ac_value数据存放的是码字对应的符号值。

10.8.8.4.81 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

```
hi_venc_jpeg_huffman_param = {"dc_tables": hi_venc_huffman_dc_table[3], # y, Cb, Cr
                              "ac_tables": hi_venc_huffman_ac_table[3], # y, Cb, Cr
                              "reserved": reserved[2]}
```

成员

成员名称	描述
dc_tables	列表，亮度DC huffman编码表数据。 Cb分量和CR分量需保持一致。
ac_tables	列表，色度AC huffman编码表数据。 Cb分量和CR分量需保持一致。
reserved	列表，预留位，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示对reserved字段进行访问。

10.8.8.4.82 hi_venc_mpf_cfg

说明

预留字典，暂不支持。

定义

```
hi_venc_mpf_cfg = {"large_thumbnail_num": large_thumbnail_num,
                   "large_thumbnail_size": hi_video_size[HI_VENC_MAX_MPF_NUM]}
```

10.8.8.4.83 hi_venc_mjpeg_cbr_param

说明

预留字典，暂不支持。

定义

```
hi_venc_mjpeg_cbr_param = {"max_qfactor": max_qfactor,
                           "min_qfactor": min_qfactor}
```

10.8.8.4.84 hi_venc_mjpeg_vbr_param

说明

预留字典，暂不支持。

定义

```
hi_venc_mjpeg_vbr_param = {"chg_pos": chg_pos,
                           "max_qfactor": max_qfactor,
                           "min_qfactor": min_qfactor}
```

10.8.8.4.85 hi_venc_roi_attr

定义编码感兴趣区域信息。

定义

```
hi_venc_roi_attr = {"idx": idx,
                    "enable": enable,
                    "is_abs_qp": is_abs_qp,
                    "qp": qp,
                    "rect": hi_rect}
```

成员

成员名称	描述
idx	整型，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	整型，是否使能这个ROI区域。 Cb分量和Cr分量需保持一致。
is_abs_qp	整型，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

成员名称	描述
qp	整型，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	字典，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。

10.8.8.4.86 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

```
hi_venc_ref_param = {"base": base,
                     "enhance": enhance,
                     "pred_en": pred_en}
```

成员

成员名称	描述
base	整型，base层的周期。 取值范围：（0，+∞）。
enhance	整型，enhance层的周期。 取值范围：[0, 255]。
pred_en	整型，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。

10.8.8.4.87 hi_venc_h264_vui

说明

定义H.264协议编码通道Vui结构体。

定义

```
hi_venc_h264_vui = {"vui_aspect_ratio": hi_venc_vui_aspect_ratio,
                    "vui_time_info": hi_venc_vui_aspect_ratio,
```

```
                "vui_video_signal": hi_venc_vui_video_signal,
                "vui_bitstream_restric": hi_venc_vui_bitstream_restric}
```

成员

成员名称	描述
vui_aspect_ratio	字典，预留参数，具体含义请参见H.264协议。
vui_time_info	字典，预留参数，具体含义请参见H.264协议。
vui_video_signal	字典，具体含义请参见H.264协议。
vui_bitstream_restric	字典，预留参数，具体含义请参见H.264协议。

10.8.8.4.88 hi_venc_h265_vui

说明

定义H.265协议编码通道Vui结构体。

定义

```
hi_venc_h265_vui = {"vui_aspect_ratio": hi_venc_vui_aspect_ratio,
                "vui_time_info": hi_venc_vui_aspect_ratio,
                "vui_video_signal": hi_venc_vui_video_signal,
                "vui_bitstream_restric": hi_venc_vui_bitstream_restric}
```

成员

成员名称	描述
vui_aspect_ratio	字典，预留参数，具体含义请参见H.265协议。
vui_time_info	字典，预留参数，具体含义请参见H.265协议。
vui_video_signal	字典，具体含义请参见H.265协议。
vui_bitstream_restric	字典，预留参数，具体含义请参见H.265协议。

10.8.8.4.89 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

```
hi_venc_vui_aspect_ratio = {"aspect_ratio_info_present_flag": aspect_ratio_info_present_flag,
                "aspect_ratio_idc": aspect_ratio_idc,
                "overscan_info_present_flag": overscan_info_present_flag,
                "overscan_appropriate_flag": overscan_appropriate_flag,
```

```
                    "sar_width": sar_width,
                    "sar_height": sar_height}
```

成员

成员名称	描述
aspect_ratio_info_present_flag	整型，具体含义请参见H.264/H.265协议，系统默认值为0。
aspect_ratio_idc	整型，具体含义请参见H.264/H.265协议，系统默认值为1。
overscan_info_present_flag	整型，具体含义请参见H.264/H.265协议，系统默认值为0。
overscan_appropriate_flag	整型，具体含义请参见H.264/H.265协议，系统默认值为0。
sar_width	整型，具体含义请参见H.264/H.265协议，系统默认值为1。
sar_height	整型，具体含义请参见H.264/H.265协议，系统默认值为1。

10.8.8.4.90 hi_venc_vui_h264_time_info

说明

定义H.264协议编码通道Vui中TimeInfo信息的结构体。预留结构体，暂不支持用户配置。

定义

```
hi_venc_vui_h264_time_info = {"timing_info_present_flag": timing_info_present_flag，
                              "fixed_frame_rate_flag": fixed_frame_rate_flag,
                              "num_units_in_tick": num_units_in_tick,
                              "time_scale": time_scale}
```

成员

成员名称	描述
timing_info_present_flag	整型，具体含义请参见H.264协议，系统默认值为0。
fixed_frame_rate_flag	整型，具体含义请参见H.264协议，系统默认值为1。
num_units_in_tick	整型，具体含义请参见H.264协议，系统默认值为1。
time_scale	整型，具体含义请参见H.264协议，系统默认值为60。

10.8.8.4.91 hi_venc_vui_h265_time_info

说明

定义H.264协议编码通道Vui中TimeInfo信息的结构体。预留结构体，暂不支持用户配置。

定义

```
hi_venc_vui_h264_time_info = {"timing_info_present_flag": timing_info_present_flag,
                              "fixed_frame_rate_flag": fixed_frame_rate_flag,
                              "num_units_in_tick": num_units_in_tick,
                              "time_scale": time_scale}
```

成员

成员名称	描述
timing_info_present_flag	整型，具体含义请参见H.264协议，系统默认值为0。
fixed_frame_rate_flag	整型，具体含义请参见H.264协议，系统默认值为1。
num_units_in_tick	整型，具体含义请参见H.264协议，系统默认值为1。
time_scale	整型，具体含义请参见H.264协议，系统默认值为60。

10.8.8.4.92 hi_venc_vui_video_signal

说明

定义H.264/H.265协议编码通道Vui中VideoSignal信息的结构体。

定义

```
hi_venc_vui_video_signal = {"video_signal_type_present_flag": video_signal_type_present_flag,
                            "video_format": video_format,
                            "video_full_range_flag": video_full_range_flag,
                            "colour_description_present_flag": colour_description_present_flag,
                            "colour_primaries": colour_primaries,
                            "transfer_characteristics": transfer_characteristics,
                            "matrix_coefficients": matrix_coefficients}
```

成员

成员名称	描述
video_signal_type_present_flag	整型，具体含义请参见H.264/H.265协议，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。

成员名称	描述
video_format	整型，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	整型，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。
colour_description_present_flag	整型，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	整型，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	整型，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	整型，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。

10.8.8.4.93 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

```
hi_venc_vui_bitstream_restric = {"bitstream_restriction_flag": bitstream_restriction_flag}
```

成员

成员名称	描述
bitstream_restriction_flag	整型，具体含义请参见H.265协议，系统默认值为0。

10.8.8.4.94 hi_venc_cu_prediction

说明

定义CU模式选择的倾向性配置结构体。

定义

```
hi_venc_cu_prediction = {"pred_mode": hi_op_mode,
                         "intra32_cost": intra32_cost,
                         "intra16_cost": intra16_cost,
                         "intra8_cost": intra8_cost,
                         "intra4_cost": intra4_cost,
                         "inter64_cost": inter64_cost,
                         "inter32_cost": inter32_cost,
                         "inter16_cost": inter16_cost,
                         "inter8_cost": inter8_cost}
```

成员

成员名称	描述
pred_mode	整型，倾向性选择模式，支持自动模式、手动模式两种。
intra32_cost	整型，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	整型，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	整型，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	整型，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	整型，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。

成员名称	描述
inter32_cost	整型，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。
inter16_cost	整型，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	整型，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。

10.8.8.4.95 hi_venc_intra_refresh

说明

P帧刷Islice控制参数。

定义

```
hi_venc_intra_refresh = {"refresh_enable": refresh_enable,
                         "intra_refresh_mode": hi_venc_intra_refresh_mode,
                         "refresh_num": refresh_num,
                         "req_i_qp": req_i_qp}
```

成员

成员名称	描述
refresh_enable	整型，是否使能刷Islice功能。 ● 0：不使能，默认值； ● 1：使能 。
intra_refresh_mode	整型，I宏块刷新模式，分为按行刷新和按列刷新。默认按行刷新。 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10-29**中的计算公式。
req_i_qp	整型，I帧QP值。 **acl.himpi.venc_set_intra_refresh**接口与**acl.himpi.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 10-29 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； – H.265时，该参数值固定为32。

10.8.8.4.96 hi_venc_intra_refresh_mode

说明

P帧刷Islice模式定义。

定义

```
INTRA_REFRESH_ROW      = 0                # 按行刷新
INTRA_REFRESH_COLUMN  = 1                  # 按列刷新
INTRA_REFRESH_BUTT     = 2
```

10.8.8.5 图像处理

10.8.8.5.1 hi_vpc_pic_info

说明

图像信息字典，用来描述VPC功能输入和输出图片信息。

定义

```
hi_vpc_pic_info = {"picture_address": picture_address,
                   "picture_buffer_size": picture_buffer_size,
                   "picture_width": picture_width,
                   "picture_height": picture_height,
                   "picture_width_stride": picture_width_stride,
                   "picture_height_stride": picture_height_stride,
                   "picture_format": hi_pixel_format}
```

成员

成员名称	描述
picture_address	整型，存放图片数据的Device地址。
picture_buffer_size	整型，存放图片数据的缓冲区大小。
picture_width	整型，图片真实宽。
picture_height	整型，图片真实高。
picture_width_stride	整型，图片宽Stride。
picture_height_stride	整型，图片高Stride。
picture_format	整型，目标图片的格式。

10.8.8.5.2 hi_vpc_crop_region

说明

VPC抠图功能需要的参数。

定义

```
hi_vpc_crop_region = {"top_offset": top_offset,
                      "left_offset": left_offset,
                      "crop_width": crop_width,
                      "crop_height": crop_height}
```

成员

成员名称	描述
top_offset	整型，上偏移。
left_offset	整型，左偏移。
crop_width	整型，抠图区域宽。
crop_height	整型，抠图区域高。

10.8.8.5.3 hi_vpc_resize_info

说明

VPC缩放功能需要的参数。

定义

```
hi_vpc_resize_info = {"resize_width": resize_width,
                      "resize_height": resize_height,
                      "interpolation": interpolation}
```

成员

成员名称	描述
resize_width	整型，缩放之后的宽。
resize_height	整型，缩放之后的高。
interpolation	整型，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）

10.8.8.5.4 hi_vpc_crop_region_info

说明

VPC抠图功能需要的参数。

定义

```
hi_vpc_crop_region_info = {"dest_pic_info": hi_vpc_pic_info,
                           "crop_region": hi_vpc_crop_region}
```

成员

成员名称	描述
dest_pic_info	字典，抠图目标图片信息。
crop_region	字典，抠图区域信息。

10.8.8.5.5 hi_vpc_crop_resize_region

说明

VPC抠图并缩放功能需要的参数。

定义

```
hi_vpc_crop_resize_region = {"dest_pic_info": hi_vpc_pic_info,
                             "crop_region": hi_vpc_crop_region,
                             "resize_info": hi_vpc_resize_info}
```

成员

成员名称	描述
dest_pic_info	字典，目标图片信息。
crop_region	字典，抠图区域信息。
resize_info	字典，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。

10.8.8.5.6 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

```
hi_vpc_crop_resize_paste_region = {"dest_pic_info": hi_vpc_pic_info,
                                   "crop_region": hi_vpc_crop_region,
                                   "resize_info": hi_vpc_resize_info,
                                   "dest_top_offset": dest_top_offset,
                                   "dest_left_offset": dest_left_offset}
```

成员

成员名称	描述
dest_pic_info	字典，目标图片信息。
crop_region	字典，抠图区域信息。
resize_info	字典，图片缩放信息。
dest_top_offset	整型，贴图到另外一张图的顶部偏移。
dest_left_offset	整型，贴图到另外一张图的左侧偏移。 左偏移需保持16像素对齐。

10.8.8.5.7 hi_vpc_bord_type

说明

定义边界填充的类型。

定义

```
HI_BORDER_CONSTANT    = 0
HI_BORDER_REPLICATE   = 1
HI_BORDER_REFLECT     = 2
HI_BORDER_REFLECT_101 = 3
```

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。硬件限制，不支持该选项。 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。硬件限制，不支持该选项。 举例，其中*表示任意图像元素：cb|abc****fgh|gf。

注意事项

以下选项支持HI_BORDER_CONSTANT和HI_BORDER_REPLICATE可以填充到4096*8192：

- Atlas 200/500 A2推理产品

10.8.8.5.8 hi_vpc_make_border_info

说明

VPC图像填充需要用的信息。

定义

```
hi_vpc_crop_resize_paste_region = {"top": top,
                                   "bottom": bottom,
                                   "left": left,
                                   "right": right,
                                   "border_type": hi_vpc_bord_type,
                                   "scalar_value": hi_vpc_scalar}
```

成员

成员名称	描述
top	整型，顶部填充像素数。
bottom	整型，底部填充像素数。
left	整型，左侧填充像素数。
right	整型，右侧填充像素数。
border_type	整型, 扩充边缘像素的类型。
scalar_value	字典，仅在填充类型为HI_BORDER_CONSTANT的时候有效，指定填充的像素值。

10.8.8.5.9 hi_vpc_histogram_config

说明

存放直方图统计结果的字典。

定义

```
hi_vpc_histogram_config = {"histogram_y_or_r": histogram_y_or_r[256], # 256 level statistics of Y or R
component
                           "histogram_u_or_g": histogram_u_or_g[256], # 256 level statistics of U or G component
                           "histogram_v_or_b": histogram_v_or_b[256]} # 256 level statistics of V or B component
```

成员

成员名称	描述
histogram_y_or_r	列表，Y或者R分量，像素分布情况。
histogram_u_or_g	列表，U或者G分量，像素分布情况。
histogram_v_or_b	列表，V或者B分量，像素分布情况。

10.8.8.5.10 hi_vpc_lut_remap

说明

配置图像各分量重映射信息的字典。

定义

```
hi_vpc_lut_remap = {"map_value_y_or_r": map_value_y_or_r[256], # remap value of Y or R component
                    "map_value_u_or_g": map_value_u_or_g[256], # remap value of U or G component
                    "map_value_v_or_b": map_value_v_or_b[256]} # remap value of V or B component
```

成员

成员名称	描述
map_value_y_or_r	列表，Y或者R分量的重映射配置信息。
map_value_u_or_g	列表，U或者G分量的重映射配置信息。
map_value_v_or_b	列表，V或者B分量的重映射配置信息。

10.8.8.5.11 hi_csc_conf

说明

定义色域配置信息。

定义

```
hi_csc_conf = {"alpha": alpha,
               "attr": 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。

10.8.8.5.12 hi_vpc_scalar

说明

定义图像处理像素各分量字典。

定义

```
hi_vpc_scalar = {"val": val[4]}
```

成员

成员名称	描述
val	列表，存放颜色分量的值，取值范围[0,255]。 当前仅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。 若输入输出都是YUV格式，填充YUV分量的值，反之，填充的是RGB分量的值。

10.8.8.5.13 hi_vpc_chn_attr

说明

定义图像处理通道属性字典。

定义

```
hi_vpc_chn_attr = {"attr": attr,
                   "pic_width": pic_width, # RW; reserved
                   "pic_height": pic_height} # RW; reserved
```

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。预留字段，暂不支持。 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**10.8.4.21 acl.himpi.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	整型，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。
pic_height	整型，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。

10.8.8.5.14 hi_vpc_crop_resize_border_region

说明

VPC抠图缩放填充信息字典。

定义

```
hi_vpc_crop_resize_border_region = {"dest_pic_info": hi_vpc_pic_info,
                                    "crop_region": hi_vpc_crop_region,
                                    "resize_info": hi_vpc_resize_info,
                                    "dest_top_offset": dest_top_offset,
                                    "dest_left_offset": dest_left_offset,
                                    "border_type": hi_vpc_bord_type,
                                    "scalar_value": hi_vpc_scalar}
```

成员

成员名称	描述
dest_pic_info	字典，目标图片信息。
crop_region	字典，原图抠图区域信息。
resize_info	字典，抠图缩放信息。
dest_top_offset	整型，贴图到另外一张图的顶部偏移。
dest_left_offset	整型，贴图到另外一张图的左侧偏移。
border_type	整型，扩充边缘像素的类型。
scalar_value	字典，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。

10.8.8.5.15 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

```
hi_vpc_crop_resize_resize_paste_region = {"dest_pic_info": hi_vpc_pic_info,
                                          "crop_region": hi_vpc_crop_region,
                                          "resize_info1": hi_vpc_resize_info,
                                          "resize_info2": hi_vpc_resize_info,
                                          "dest_top_offset": dest_top_offset,
                                          "dest_left_offset": dest_left_offset,
                                          "reserved": [2]}
```

成员

成员名称	描述
dest_pic_info	字典，目标图片信息。
crop_region	字典，抠图区域信息。
resize_info1	字典，第一次图片缩放的信息。
resize_info2	字典，第二次图片缩放的信息。

成员名称	描述
dest_top_offset	整型，贴图到另外一张图的顶部偏移。
dest_left_offset	整型，贴图到另外一张图的左侧偏移。
reserved	列表，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。

10.8.8.5.16 hi_vpc_workspace_param

说明

定义设置系统内部临时缓存相关的参数。

定义

```
hi_vpc_workspace_param = {"func": hi_workspace_func,
                          "max_source_pic_width": max_source_pic_width,
                          "max_source_pic_height": max_source_pic_height,
                          "max_dest_pic_width": max_dest_pic_width,
                          "max_dest_pic_height": max_dest_pic_height,
                          "reserved": reserved}
```

成员

输入、输出图片分辨率的取值范围为：10*6~4096*8192。

参数名	输入/输出	说明
func	输入	整型，指定为哪种功能申请临时缓存。
max_source_pic_width	输入	整型，输入图片的最大宽度。
max_source_pic_height	输入	整型，输入图片的最大高度。
max_dest_pic_width	输入	整型，输出图片的最大宽度。
max_dest_pic_height	输入	整型，输出图片的最大高度。
reserved	输入	整型，预留参数，为保证后续版本兼容性，请务必使用memset配置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。

10.8.8.5.17 hi_workspace_func

说明

定义需申请临时缓存的功能枚举。

定义

```
REMAP = 0,
AFFINE = 1
```

设置为REMAP时，只能调用lut_remap接口

设置为AFFINE时，只能调用warp_affine接口

10.8.8.5.18 hi_remap_lut

说明

定义像素位置重映射表信息。

定义

```
hi_remap_lut = {"src_pic_width": src_pic_width,
                "src_pic_height": src_pic_height,
                "dest_pic_width": dest_pic_width,
                "dest_pic_height": dest_pic_height,
                "lut": lut,
                "lut_size": lut_size,
                "reserved": reserved}
```

成员

输入、输出图片分辨率取值范围为：10*6~4096*8192。

成员名称	描述
src_pic_width	整型，输入图片宽度。
src_pic_height	整型，输入图片高度。
dest_pic_width	整型，输出图片宽度。
dest_pic_height	整型，输出图片高度。
lut_size	整型，像素位置重映射表的内存大小，单位Byte。 可调用**10.8.4.36 acl.himpi.vpc_get_lut_mem_size**接口获取内存大小。
lut	整型，像素位置重映射表的内存地址，该内存需由用户提前申请。

成员名称	描述
reserved	整型，预留参数，为保证后续版本兼容性，请务必使用memset配置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。

10.8.8.5.19 hi_point_pair_info

说明

输入、输出图片中的像素点坐标信息。

定义

```
hi_point_pair_info = {"src_point": hi_point[4],
                      "src_point_count": src_point_count,
                      "dest_point": hi_point[4],
                      "dest_point_count": dest_point_count}
```

成员

成员名称	描述
src_point	字典，输入图片像素点坐标。
src_point_count	整型，输入图片像素点坐标数量。
dest_point	字典，输出图片像素点坐标。
dest_point_count	整型，输出图片像素点坐标数量。

10.8.8.5.20 hi_float_point

说明

坐标信息。

定义

```
hi_float_point = {"x": x,
                  "y": y}
```

成员

成员名称	描述
x	浮点型，像素点x坐标。
y	浮点型，像素点y坐标。

10.8.8.5.21 hi_warp_transform_param

说明

Remap变换参数。

定义

```
hi_warp_transform_param = {"src": hi_vpc_pic_info,
                           "dst": hi_vpc_pic_info,
                           "transform_config": hi_transform_config}
```

成员

成员名称	描述
src	字典，输入图片信息
dst	字典，输出图片信息。
transform_config	字典，仿射/透视/Remap变换配置信息。

10.8.8.5.22 hi_transform_config

说明

定义仿射/透视/Remap变换配置信息。

定义

```
hi_transform_config = {"interpolation": interpolation,
                       "border_type": hi_vpc_bord_type,
                       "scalar_value": hi_vpc_scalar}
```

成员

成员名称	描述
interpolation	整型，支持如下缩放算法： ● 0：业界通用的Bilinear算法 ● 1：业界通用的Nearest neighbor 算法
border_type	整型，边界填充的类型。 支持如下取值 ● 0：寄存器固定值 ● 1：边界复制模式
scalar_value	字典，图像填充的像素值，只支持3通道。 如果输入图片格式为YUV400，则val[0]存放Y分量，val[1]、val[2]、val[3预留，需初始化为0。

10.8.8.5.23 hi_morph_shapes

说明

定义滤波卷积形状。

定义

```
MORPH_RECT    = 0,
MORPH_CROSS   = 1,
MORPH_ELLIPSE = 2,
MORPH_MAX     = 100
```

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。

10.8.8.5.24 hi_rotation

说明

定义旋转角度。

定义

```
HI_ROTATION_90  = 0,
HI_ROTATION_180 = 1,
HI_ROTATION_270 = 2
```

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。

10.8.8.5.25 hi_blk_size

说明

定义单个马赛克框的大小。

定义

```
HI_BLK_SIZE_4   = 0,
HI_BLK_SIZE_8   = 1,
HI_BLK_SIZE_16  = 2,
HI_BLK_SIZE_32  = 3,
HI_BLK_SIZE_64  = 4,
HI_BLK_SIZE_128 = 5
```

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。

10.8.8.5.26 hi_median_blur_param

说明

中值滤波信息。

定义

```
hi_median_blur_param = {"src": hi_vpc_pic_info,
                        "dst": hi_vpc_pic_info,
                        "median_blur_cfg": hi_median_blur_config}
```

成员

成员名称	描述
src	字典，输入图片信息。

成员名称	描述
dst	字典，输出图片信息。
median_blur_cfg	字典，中值滤波配置参数。

10.8.8.5.27 hi_median_blur_config

说明

中值滤波参数。

定义

```
hi_median_blur_config = {"kernel_size": kernel_size}
```

成员

成员名称	描述
kernel_size	整型，中值滤波卷积核尺寸，只支持1，3或者5。

10.8.8.5.28 hi_blur_param

说明

滤波信息。

定义

```
hi_blur_param = {"src" : hi_vpc_pic_info,
                 "dst" : hi_vpc_pic_info,
                 "blur_config" : hi_blur_config}
```

成员

成员名称	描述
src	字典，输入图片信息。
dst	字典，输出图片信息。
blur_cfg	字典，滤波配置参数。

10.8.8.5.29 hi_blur_config

说明

滤波参数。

定义

```
hi_blur_config = {"kernel_size": hi_size,
                  "morph_shapes": hi_morph_shapes,
                  "anchor": hi_point,
                  "iterations": iterations,
                  "border_type": hi_vpc_bord_type,
                  "scalar_value": hi_vpc_scalar}
```

成员

成员名称	描述
kernel_size	字典，卷积核的尺寸，kernel_size的宽和高只能设置为1，3或者5。
morph_shapes	整型，滤波的卷积形状。
anchor	字典，卷积核锚点，需要设置成(-1, -1)，预留字段。
iterations	整型，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	整型，边界填充类型，只支持HI_BORDER_CONSTANT和HI_BORDER_REPLICATE两种填充模式。
scalar_value	字典，边界填充值。

10.8.8.5.30 hi_gaussian_blur_param

说明

高斯滤波信息。

定义

```
hi_gaussian_blur_param = {"src": hi_vpc_pic_info,
                          "dst": hi_vpc_pic_info,
                          "gaussian_blur_cfg": hi_gaussian_blur_config}
```

成员

成员名称	描述
src	字典，输入图片信息。
dst	字典，输出图片信息。

成员名称	描述
gaussian_blur_cfg	字典，高斯滤波配置参数。

10.8.8.5.31 hi_gaussian_blur_config

说明

高斯滤波参数。

定义

```
hi_gaussian_blur_config = {"kernel_size": hi_size,
                           "sigma_x": sigma_x,
                           "sigma_y": sigma_y,
                           "border_type": hi_vpc_bord_type,
                           "scalar_value": hi_vpc_scalar,
                           "reserved": [2]}
```

成员

成员名称	描述
kernel_size	字典，卷积核尺寸，kernel_size的宽和高只能设置为1，3或者5。
sigma_x	整型，高斯滤波x轴sigma值。
sigma_y	整型，高斯滤波y轴sigma值，如果sigma_y=0，则sigma_y=sigma_x。
border_type	整型，边界填充类型，只支持HI_BORDER_CONSTANT和HI_BORDER_REPLICATE两种填充模式。
scalar_value	字典，边界填充值。
reserved[2]	列表，预留参数，代码中必须避免显式对reserved字段进行访问。

10.8.8.5.32 hi_filter_2d_param

说明

2D卷积滤波信息。

定义

```
hi_filter_2d_param = {"src": hi_vpc_pic_info,
                      "dst": hi_vpc_pic_info,
                      "filter_2d_cfg": hi_filter_2d_config}
```

成员

成员名称	描述
src	字典，输入图片信息。
dst	字典，输出图片信息。
filter_2d_cfg	字典，2D卷积滤波配置参数。

10.8.8.5.33 hi_filter_2d_config

说明

2D卷积滤波参数。

定义

```
hi_filter_2d_config = {"filter": [5][5],
                       "kernel_size": hi_size,
                       "anchor": hi_point,
                       "delta": delta,
                       "border_type": hi_vpc_bord_type,
                       "scalar_value": hi_vpc_scalar,
                       "reserved": [2]}
```

成员

成员名称	描述
filter	列表，自定义卷积核。
kernel_size	字典，卷积核尺寸，kernel_size的宽和高只能设置为1，3或者5。
anchor	字典，卷积核锚点，预留字段，需要设置成(-1, -1)。
delta	整型，增量值，会被加到执行卷积之后的每个像素值之上。
border_type	整型，边界填充类型，只支持HI_BORDER_CONSTANT和HI_BORDER_REPLICATE两种填充模式。
scalar_value	字典，边界填充值。
reserved[2]	列表，预留参数，在代码中必须避免显式对reserved字段进行访问。

10.8.8.5.34 hi_mosaic_param

说明

马赛克信息。

定义

```
hi_mosaic_param = {"src": hi_vpc_pic_info,
                   "dst": hi_vpc_pic_info,
                   "count": count,
                   "mosaic": hi_mosaic}
```

成员

成员名称	描述
src	字典，输入图片信息。
dst	字典，输出图片信息。
count	整型，马赛克区域个数，取值范围[1, 100]。
mosaic	列表，马赛克区域配置参数列表。

10.8.8.5.35 hi_rotate_param

说明

固定角度旋转信息。

定义

```
hi_rotate_param = {"src": hi_vpc_pic_info,
                   "dst": hi_vpc_pic_info,
                   "angle": hi_rotation}
```

成员

成员名称	描述
src	字典，输入图片信息。
dst	字典，输出图片信息。
angle	整型，旋转角度。

10.8.8.5.36 hi_mosaic

说明

马赛克参数。

定义

```
hi_mosaic = {"blk_size": hi_blk_size,
             "rect": hi_rect}
```

成员

成员名称	描述
blk_size	整型，单个马赛克的大小，只支持HI_BLK_SIZE_8、HI_BLK_SIZE_16、HI_BLK_SIZE_32。
rect	字典，马赛克区域，其宽、高最小为8，可以部分区域在图片外。

10.8.8.5.37 hi_cover_type

说明

定义覆盖类型。

定义

```
HI_COVER_RECT = 0
HI_COVER_QUAD = 1
```

成员

成员名称	描述
HI_COVER_RECT	矩形覆盖。
HI_COVER_QUAD	任意四边形覆盖。

10.8.8.5.38 hi_cover_param

说明

覆盖信息。

定义

```
hi_cover_param = {"src": hi_vpc_pic_info,
                  "dst": hi_vpc_pic_info,
                  "count": count,
                  "cover": hi_cover}
```

成员

成员名称	描述
src	字典，输入图片信息。
dst	字典，输出图片信息。
count	整型，覆盖区域个数，取值范围[1, 100]。

成员名称	描述
cover	列表，覆盖区域配置参数列表。

10.8.8.5.39 hi_cover

说明

覆盖参数。

定义

```
hi_cover = {"type": hi_cover_type,
            "rect"/"quad": hi_rect/hi_quad_cover,
            "color": color}
```

成员

成员名称	描述
type	整型，覆盖类型。
rect	字典，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	字典，自定义覆盖区域。
color	整型，覆盖区域的颜色值，低24位有效。

10.8.8.5.40 hi_quad_cover

说明

自定义覆盖区域信息。

定义

```
hi_quad_cover = {"is_solid": is_solid,
                 "thick": thick,
                 "point": hi_point[4]}
```

成员

成员名称	描述
is_solid	整型，预留字段，暂不支持。
thick	整型，预留字段，暂不支持。
point[4]	列表，四边形4个点的坐标。

10.8.8.5.41 hi_line_param

说明

画线信息。

定义

```
hi_line_param = {"src": hi_vpc_pic_info,
                 "dst": hi_vpc_pic_info,
                 "count": count,
                 "line": hi_line}
```

成员

成员名称	描述
src	字典，输入图片信息。
dst	字典，输出图片信息。
count	整型，画线条数，取值范围[1, 100]。
line	列表，画线配置参数列表。 当图片格式为YUV422SP、YUV420SP时，要求起始点、结束点坐标值为偶数。

10.8.8.5.42 hi_line

说明

画线参数。

定义

```
hi_line = {"start_point": hi_point,
           "end_point": hi_point,
           "thick": thick,
           "color": color}
```

成员

成员名称	描述
start_point	字典，起始点坐标。
end_point	字典，结束点坐标。 结束点坐标不能起始点坐标相同。
thick	整型，线的粗细值，取值需要2对齐。
color	整型，线的颜色值，低24位有效。

10.8.8.5.43 hi_osd_param

说明

叠加信息。

定义

```
hi_osd_param = {"src": hi_vpc_pic_info,
                "dst": hi_vpc_pic_info,
                "count": count,
                "osd": hi_osd,
                "clut": clut[16]}
```

成员

成员名称	描述
src	字典，输入图片信息。
dst	字典，输出图片信息。
count	整型，叠加区域个数，取值范围[1, 100]。
osd	列表，叠加区域配置参数类别。
clut	列表，clut查找表系数。

10.8.8.5.44 hi_osd

说明

叠加参数。

定义

```
hi_osd = {"rect": hi_rect,
          "pixel_format": hi_pixel_format,
          "picture_address": picture_address,
          "stride": stride,
          "bg_alpha": bg_alpha,
          "fg_alpha": fg_alpha,
          "osd_inverted_color": hi_osd_inverted_color}
```

成员

成员名称	描述
rect	字典，叠加区域坐标。

成员名称	描述

pixel_format	整型，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:6bit 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	整型，被叠加图片的地址。
stride	整型，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	整型，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	整型，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	整型，反色类型。

10.8.8.5.45 hi_osd_inverted_color

说明

定义OSD反色类型。

定义

```
HI_OSD_INVERTED_COLOR_NONE  = 0,
HI_OSD_INVERTED_COLOR_RGB    = 1,
HI_OSD_INVERTED_COLOR_ALPHA  = 2,
HI_OSD_INVERTED_COLOR_ALL    = 3
```

成员

成员名称	描述
HI_OSD_INVERTED_COLOR_NONE	不反色。
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。

10.8.8.5.46 hi_transform_matrix

说明

定义仿射变换矩阵参数。

定义

```
{"matrix_type": hi_matrix_type,
 "matrix": matrix[3][4]}
```

成员

成员名称	描述
matrix_type	整型，矩阵类型。
matrix	列表，仿射变换矩阵。

10.8.8.5.47 hi_matrix_type

说明

定义仿射变换中的矩阵类型。

定义

```
HI_MATRIX_2_PLUS_3 = 0,
HI_MATRIX_3_PLUS_3 = 1,
HI_MATRIX_3_PLUS_4 = 2
```

成员

成员名称	描述
HI_MATRIX_2_PLUS_3	2*3 Matrix。
HI_MATRIX_3_PLUS_3	3*3 Matrix。
HI_MATRIX_3_PLUS_4	3*4 Matrix。

10.8.8.5.48 hi_flip_param

说明

翻转信息。

定义

```
hi_flip_param = {"src": hi_vpc_pic_info,
                 "dst": hi_vpc_pic_info,
                 "flip_mode": hi_flip_mode}
```

成员

成员名称	描述
src	字典，输入图片信息。
dst	字典，输出图片信息。
flip_mode	整型，翻转模式参数。

10.8.8.5.49 hi_flip_mode

说明

定义翻转模式类型。

定义

```
HI_FLIP_HOR  = 0,
HI_FLIP_VER  = 1,
HI_FLIP_BOTH = 2
```

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。

10.8.8.5.50 hi_map_param

说明

用户原始map表信息。

定义

```
hi_map_param = {"map1": map1,
                "map2": map2,
                "map_size": map_size,
                "src_pic_width": src_pic_width,
                "src_pic_height": src_pic_height,
                "dest_pic_width": dest_pic_width,
                "dest_pic_height": dest_pic_height,
                "reserved": reserved}
```

成员

成员名称	描述
map1	整型，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	整型，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	整型，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	整型，map表对应的输入图片的宽度。
src_pic_height	整型，map表对应的输入图片的高度。
dest_pic_width	整型，map表对应的输出图像的宽度。
dest_pic_height	整型，map表对应的输出图像的高度。
reserved	整型，预留参数，在代码中必须避免显式对reserved字段进行访问。

10.8.8.6 PNGD 图像解码

10.8.8.6.1 hi_pngd_chn_attr

说明

定义PNGD解码通道属性字典。

定义

```
hi_pngd_chn_attr = {"stream_que_cnt": stream_que_cnt, "reserved": reserved[4]}
```

成员

成员名称	描述
stream_que_cnt	整型，PNGD图片缓存队列大小。预留参数。
reserved	列表，预留参数，在代码中必须避免显示对reserved字段进行访问。

10.8.8.6.2 hi_img_stream

说明

定义图片码流信息。

定义

```
hi_img_stream = {"type":hi_payload_type, "addr": addr, "len": len, "pts": pts, "reserved": reserved[2]}
```

成员

成员名称	描述
type	整型，图片类型。
addr	整型，图片在内存中的起始虚拟地址。
len	整型，存放图片的内存大小，单位Byte。
pts	整型，图片时戳，预留参数。
reserved	预留参数，在代码中必须避免显示对reserved字段进行访问。

10.8.8.6.3 hi_pic_info

说明

定义图片信息。

定义

```
hi_pic_info = {"picture_address": picture_address,
               "picture_buffer_size": picture_buffer_size,
               "picture_width": picture_width,
               "picture_height": picture_height,
               "picture_width_stride": picture_width_stride,
               "picture_height_stride": picture_height_stride,
               "picture_format": hi_pixel_format}
```

成员

成员名称	描述
picture_address	整型，图片在Device内存中的起始虚拟地址。
picture_buffer_size	整型，存放图片的内存大小，单位Byte。
picture_width	整型，图片宽。
picture_height	整型，图片高。
picture_width_stride	整型，图片宽stride。
picture_height_stride	整型，图片高stride。
picture_format	整型，图片格式。

10.8.8.6.4 hi_png_color_format

说明

定义PNG源图片格式的枚举。

定义

```
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
```

10.8.9 返回码列表

10.8.9.1 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。
0xA0028003	HI_ERR_SYS_ILLEGAL_PARAM	参数设置无效

错误代码	宏定义	描述
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。

10.8.9.2 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
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

错误代码	宏定义	描述
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**acl.himpi.dvpp_malloc**接口申请，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。

10.8.9.3 VDEC 视频/JPEGD 图像解码返回码

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。

错误代码	宏定义	描述
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。
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	-	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。
0xA00E8001	-	无效的Device ID。
0xA00E8002	-	无效的channel ID。
0xA00E8003	-	参数不合法，例如不合法的枚举值。
0xA00E8004	-	资源已存在。
0xA00E8005	-	资源不存在。
0xA00E8006	-	函数参数中有空指针。
0xA00E8007	-	使能系统、Device或通道前未配置对应的参数。
0xA00E8008	-	不支持的参数或者功能。
0xA00E8009	-	该操作不允许，如试图修改静态配置参数。
0xA00E800A	-	无效的pipe ID。

错误代码	宏定义	描述
0xA00E800B	-	无效的stitch group ID。
0xA00E800C	-	分配内存失败，如系统内存不足。
0xA00E800D	-	分配缓存失败，如申请的数据缓冲区太大。
0xA00E800E	-	缓冲区中无数据。
0xA00E800F	-	缓冲区中数据满。
0xA00E8010	-	系统没有初始化或者相关依赖的模块没有加载。
0xA00E8011	-	地址错误。
0xA00E8012	-	系统忙。
0xA00E8013	-	缓存小于实际需要的大小。
0xA00E8014	-	硬件或软件处理超时。
0xA00E8015	HI_ERR_VDEC_SYS_ERROR	内部系统错误。
0xA00E803F	-	最大的返回码，该模块的错误码必须小于该值。

10.8.9.4 VENC 视频/JPEGE 图像编码返回码

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

错误代码	宏定义	描述
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
0xA0088012	HI_ERR_VENC_BUSY	系统忙
0xA0088013	-	缓存小于实际需要的大小。
0xA0088014	-	硬件或软件处理超时。
0xA0088015	-	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。
0xA00B8001	-	无效的Device ID。
0xA00B8002	-	无效的channel ID。
0xA00B8003	-	参数不合法，例如不合法的枚举值。
0xA00B8004	-	资源已存在。
0xA00B8005	-	资源不存在。
0xA00B8006	-	函数参数中有空指针。
0xA00B8007	-	使能系统、Device或通道前未配置对应的参数。
0xA00B8008	-	不支持的参数或者功能。
0xA00B8009	-	该操作不允许，如试图修改静态配置参数。
0xA00B800A	-	无效的pipe ID。

错误代码	宏定义	描述
0xA00B800B	-	无效的stitch group ID。
0xA00B800C	-	分配内存失败，如系统内存不足。
0xA00B800D	-	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
0xA00B800E	-	缓冲区中无数据。
0xA00B800F	-	缓冲区中数据满。
0xA00B8010	-	系统没有初始化或者相关依赖的模块没有加载。
0xA00B8011	-	地址错误。
0xA00B8012	-	系统忙。
0xA00B8013	-	缓存小于实际需要的大小。
0xA00B8014	-	硬件或软件处理超时。
0xA00B8015	HI_ERR_VENC_SYS_ERROR	内部系统错误。
0xA00B803F	-	最大的返回码，该模块的错误码必须小于该值。

10.8.9.5 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

错误代码	宏定义	描述
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
0xA0408012	HI_ERR_PNGD_BUSY	系统忙
0xA0408013	-	缓存小于实际需要的大小。
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。

10.9 算子加载与执行（op）

10.9.1 函数：set_model_dir

C函数原型	**aclError aclopSetModelDir(const char *modelDir)**
Python函数	**ret = acl.op.set_model_dir(model_dir)**
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，同步接口。
输入说明	**model_dir**：str，表示指定模型文件所在目录。 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	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	Atlas 200/500 A2推理产品，在Host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40000000。

注意事项	无
参考资源	接口调用流程、示例，参见**6 单算子调用**。

10.9.2 函数：load

C函数原型	**aclError aclopLoad(const void *model, size_t modelSize)**
Python函数	**ret = acl.op.load(model, model_size)**
函数功能	从内存中加载单算子模型数据，由用户管理内存。同步接口。
输入说明	**model**：int，表示单算子模型数据的地址对象。 **model_size**：int，表示内存中的模型数据长度，单位Byte。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	Atlas 200/500 A2推理产品，在Host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40000000。
注意事项	无
参考资源	接口调用流程，参见**6 单算子调用**。

10.9.3 函数：set_op_execute_time_out

C函数原型	**aclError aclrtSetOpExecuteTimeOut(uint32_t timeout)**
Python函数	**ret = acl.rt.set_op_execute_time_out(timeout)**
函数功能	设置算子执行的超时时间。同步接口。
输入说明	**timeout**：int，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	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	不调用本接口，对于AI CPU算子默认超时时间为30秒，非AI CPU算子默认不超时；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。

10.9.4 函数：execute

须知

此接口后续版本会废弃，请使用acl.op.execute_v2接口。

C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);**
Python函数	**ret = acl.op.execute(op_type, input_desc, inputs, output_desc, outputs, attr, stream)**
函数功能	异步执行指定的算子。
输入说明	**op_type**：str，指定算子类型名称。 **input_desc**：list，表示算子输入tensor的描述, 整形列表，包含多个acl的tensor描述地址对象。 **inputs**：list，表示算子输入tensor，整形列表，包含多个aclDataBuffer数据地址对象。 **output_desc**：list，表示算子输出tensor的描述，整形列表，包含多个acl的tensor描述地址对象。 **outputs**：list，表示算子输出tensor，整形列表，包含多个aclDataBuffer数据地址对象。 **attr**：int， 算子的属性地址对象。 **stream**：int， 该算子需要加载的Stream对象。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
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acl.op.execute时，pyACL根据op_type、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。
注意事项	无

10.9.5 函数：execute_v2

C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);**
Python函数	**ret = acl.op.execute_v2(op_type, input_desc, inputs, output_desc, outputs, attr, stream)**
函数功能	同步或者异步执行指定的算子。
输入说明	**op_type**：str，指定算子类型名称。 **input_desc**：list，表示算子输入tensor的描述, 整形列表，包含多个acl的tensor描述地址对象。 **inputs**：list，表示算子输入tensor，整形列表，包含多个aclDataBuffer数据地址对象。 **output_desc**：list，表示算子输出tensor的描述，整形列表，包含多个acl的tensor描述地址对象。 **outputs**：list，表示算子输出tensor，整形列表，包含多个aclDataBuffer数据地址对象。 **attr**：int，算子的属性地址对象。 **stream**：int，该算子需要加载的Stream对象。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。

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acl.op.execute_v2接口时，pyACL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acl.op.infer_shape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acl.op.execute_v2的输入。该场景下，acl.op.execute_v2接口是异步接口。 ● 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acl.op.execute_v2的输入。该场景下，在调用acl.op.execute_v2接口执行算子后，系统内部会计算出准确的输出Shape，通过acl.op.execute_v2接口的outputDesc参数输出，此时acl.op.execute_v2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acl.op.execute_v2的输入。该场景下，在调用acl.op.execute_v2接口执行算子后，系统内部会计算出准确的输出Shape，通过acl.op.execute_v2接口的outputDesc参数输出，此时acl.op.execute_v2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：**acl.create_tensor_desc**(ACL_DT_UNDEFINED, 0, [], 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[]。 ● 则需按此种方式创建aclDataBuffer类型的数据：acl.create_data_buffer([], 0)，同时aclDataBuffer中的数据不需要释放。 执行有constant输入的算子时，需要先调用**acl.set_tensor_const**接口设置constant输入。 调用acl.op.execute_v2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**acl.set_tensor_const**接口设置constant输入，则需调用**acl.set_tensor_place_ment**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**acl.set_tensor_place_ment**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。

注意事项	无
参考资源	接口调用流程、示例，参见6 单算子调用。

10.9.6 函数：execute_with_handle

C函数原型	**aclError aclopExecWithHandle(aclopHandle *handle, int numInputs, const aclDataBuffer *const inputs[], int numOutputs, aclDataBuffer *const outputs[], aclrtStream stream);**
Python函数	**ret = acl.op.execute_with_handle(handle, inputs,outputs, stream)**
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acl.op.execute_v2。异步接口。
输入说明	**handle**：int，指定执行算子的handle，指针地址。 需提前调用acl.op.create_handle接口创建aclopHandle类型的数据。 **inputs**：list，算子输入tensor， 整形列表，包含多个aclDataBuffer数据地址对象。 需提前调用acl.create_data_buffer接口创建aclDataBuffer类型的数据。 **outputs**：list，算子输出tensor，整形列表，包含多个aclDataBuffer数据地址对象。 需提前调用acl.create_data_buffer接口创建aclDataBuffer类型的数据。 **stream**：int，执行算子所在的Stream， stream地址对象。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
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。 ● 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：acl.create_data_buffer(nullptr, 0)，同时aclDataBuffer中的数据不需要释放，因为是空的数据类型。

10.9.7 函数：cast

C函数原型	**aclError aclopCast(const aclTensorDesc *srcDesc, const aclDataBuffer *srcBuffer, const aclTensorDesc *dstDesc, aclDataBuffer *dstBuffer, uint8_t truncate, aclrtStream stream)**

Python函数	**ret = acl.op.cast(src_desc, src_buffer, dst_desc, dst_buffer, truncate, stream)**
函数功能	转换输入数据的数据类型，异步接口。
输入说明	**src_desc**: int，输入tensor描述指针地址。 **src_buffer**：int，输入tensor，调用**acl.create_data_buffer**创建的aclDataBuffer指针地址。 **dst_desc**：int， 输出tensor的描述指针地址。 **dst_buffer**：int，输出tensor，调用**acl.create_data_buffer**创建的aclDataBuffer指针地址。 **truncate**：int，预留。 **stream**：int，执行算子所在的Stream，stream地址对象。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无
参考资源	接口调用示例，参见**6.3 接口调用流程**。

10.9.8 函数：create_handle_for_cast

C函数原型	**aclError aclopCreateHandleForCast(const aclTensorDesc *srcDesc, const aclTensorDesc *dstDesc, uint8_t truncate, aclopHandle **handle)**
Python函数	**handle, ret = acl.op.create_handle_for_cast(src_desc, dst_desc, truncate)**
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**acl.op.execute_with_handle**接口执行算子。
输入说明	**src_desc**：int，输入tensor描述指针地址。 **dst_desc**：int，输出tensor描述指针地址。 **truncate**：预留。
返回值说明	**handle**：int，输出的handle的指针地址。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败
约束说明	无
注意事项	无

10.9.9 函数：create_attr

C函数原型	**aclopAttr *aclopCreateAttr()**
Python函数	**attr = acl.op.create_attr()**
函数功能	创建acl算子属性类型的数据，同步接口。
输入说明	无
返回值说明	**attr**：int，acl算子属性地址对象。
约束说明	无
注意事项	无

10.9.10 函数：destroy_attr

C函数原型	**void aclopDestroyAttr(const aclopAttr *attr)**
Python函数	**acl.op.destroy_attr(attr)**
函数功能	销毁acl算子属性类型的数据，同步接口。
输入说明	**attr**：int，acl算子属性地址对象, 此对象使用调用acl.op.create_attr创建。
返回值说明	无
约束说明	销毁类接口：调用该接口后，不能继续使用已释放或销毁的资源，建议调用该接口后，将相关资源设置为无效值（例如，置为0或None）。
注意事项	无

10.9.11 函数：set_attr_bool

C函数原型	**aclError aclopSetAttrBool(aclopAttr *attr, const char *attrName, uint8_t attrValue)**
Python函数	**ret = acl.op.set_attr_bool(attr, attr_name, attr_value)**
函数功能	设置bool类型标量的属性值，同步接口。

输入说明	**attr**：int，acl算子属性地址对象, 此对象使用调用acl.op.create_attr创建。 **attr_name**：str，属性名。 **attr_value**：int，属性值。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.9.12 函数：set_attr_int

C函数原型	**aclError aclopSetAttrInt(aclopAttr *attr, const char *attrName, int64_t attrValue);**
Python函数	**ret = acl.op.set_attr_int(attr, attr_name, attr_value)**
函数功能	设置int64_t类型标量的属性值，同步接口。
输入说明	**attr**：int，acl算子属性地址对象, 此对象使用调用acl.op.create_attr创建。 **attr_name**：str，属性名。 **attr_value**：int，属性值。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.9.13 函数：set_attr_float

C函数原型	**aclError aclopSetAttrFloat(aclopAttr *attr, const char *attrName, float attrValue);**
Python函数	**ret = acl.op.set_attr_float(attr, attr_name, attr_value)**
函数功能	设置float类型标量的属性值，同步接口。

输入说明	**attr**：int，acl算子属性地址对象, 此对象使用调用acl.op.create_attr创建。 **attr_name**：str，属性名。 **attr_value**：float，属性值。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.9.14 函数：set_attr_string

C函数原型	**aclError aclopSetAttrString(aclopAttr *attr, const char *attrName, const char* attrValue);**
Python函数	**ret = acl.op.set_attr_string(attr, attr_name, attr_value)**
函数功能	设置字符串类型标量的属性值，同步接口。
输入说明	**attr**：int，acl算子属性地址对象, 此对象使用调用acl.op.create_attr创建。 **attr_name**：str，属性名。 **attr_value**：str，属性值。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.9.15 函数：set_attr_list_bool

C函数原型	**aclError aclopSetAttrListBool(aclopAttr *attr, const char *attrName, int numValues, const uint8_t *attrValue);**
Python函数	**ret = acl.op.set_attr_list_bool(attr, attr_name, attr_value)**
函数功能	设置bool类型列表的属性值，同步接口。

输入说明	**attr**：int，acl算子属性地址对象, 此对象使用调用acl.op.create_attr创建。 **attr_name**：str，属性名。 **attr_value**：同时支持两种数据类型作为输入。 1. list, list中每个元素的数据类型为int，取值范围为0~255。 2. numpy, 一维数组numpy对象，类型为uint8。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	● 建议attr_value使用list数据类型，后续将不支持numpy作为输入。 ● 若attr_value需使用numpy数组，需要运行环境为python>=3.8且numpy>=1.22.0。

10.9.16 函数：set_attr_list_int

C函数原型	**aclError aclopSetAttrListInt(aclopAttr *attr, const char *attrName, int numValues, const int64_t *attrValue);**
Python函数	**ret = acl.op.set_attr_list_int(attr, attr_name, attr_value)**
函数功能	设置int64_t类型列表的属性值，同步接口。
输入说明	**attr**：int，acl算子属性地址对象, 此对象使用调用acl.op.create_attr创建。 **attr_name**：str，属性名。 **attr_value**：同时支持两种数据类型作为输入。 1. list, list中每个元素的数据类型为int。 2. numpy, 一维数组numpy对象，类型为int64。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	● 建议attr_value使用list数据类型，后续将不支持numpy作为输入。 ● 若attr_value需使用numpy数组，需要运行环境为python>=3.8且numpy>=1.22.0。

10.9.17 函数：set_attr_list_float

C函数原型	**aclError aclopSetAttrListFloat(aclopAttr *attr, const char *attrName, int numValues, const float *attrValue);**
Python函数	**ret = acl.op.set_attr_list_float(attr, attr_name, num_values , attr_value)**
函数功能	设置float类型列表的属性值，同步接口。
输入说明	**attr**：int，acl算子属性地址对象, 此对象使用调用acl.op.create_attr创建。 **attr_name**: str，属性名。 **attr_value**：同时支持两种数据类型作为输入。 1. list, list中每个元素的数据类型为float。 2. numpy object , 一维数组numpy对象，类型为float32。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	● 建议attr_value使用list数据类型，后续将不支持numpy作为输入。 ● 若attr_value需使用numpy数组，需要运行环境为python>=3.8且numpy>=1.22.0。

10.9.18 函数：set_attr_list_string

C函数原型	**aclError aclopSetAttrListString(aclopAttr *attr, const char *attrName, int numValues, const char **attrValue);**
Python函数	**ret = acl.op.set_attr_list_string(attr, attr_name , attr_value)**
函数功能	设置字符串类型列表的属性值，同步接口。
输入说明	**attr**：int，acl算子属性地址对象, 此对象使用调用acl.op.create_attr创建。 **attr_name**：str，属性名。 **attr_value**：list，每个元素是字符串的列表。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.9.19 函数：set_attr_list_list_int

C函数原型	**aclError aclopSetAttrListListInt(aclopAttr *attr, const char *attrName, int numLists, const int *numValues, const int64_t *const values[]);**
Python函数	**ret = acl.op.set_attr_list_list_int(attr, attr_name , values)**
函数功能	设置int64_t类型列表的列表属性值，同步接口。
输入说明	**attr**：int，acl算子属性地址对象, 此对象使用调用acl.op.create_attr创建。 **attr_name**：str，属性名。 **values**：list，列表中每个元素是一个list（list的每个元素数据类型为int）或一个一维数组numpy对象（numpy对象的类型为int64）。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	● 建议attr_value使用list数据类型，后续将不支持numpy作为输入。 ● 若attr_value需使用numpy数组，需要运行环境为python>=3.8且numpy>=1.22.0。

10.9.20 函数：create_handle

C函数原型	**aclError aclopCreateHandle(const char *opType, int numInputs,int numInputs, const aclTensorDesc *const inputDesc[], int numOutputs, const aclTensorDesc *const outputDesc[], const aclopAttr *opAttr, aclopHandle **handle);**
Python函数	**handle, ret = acl.op.create_handle(op_type, input_desc, output_desc, op_attr)**
函数功能	创建一个执行算子的handle，同步接口。

输入说明	**op_type**：str，指定算子类型名称字符串类型。 **input_desc**：list，算子输入tensor的描述，整形列表。 对于算子有constant输入的场景，调用**acl.set_tensor_place_ment**设置TensorDesc的placement属性，不支持将mem_type设置为Host内存。 **output_desc**：list，算子输出tensor的描述，整形列表。 对于算子有constant输入的场景，调用**acl.set_tensor_place_ment**设置TensorDesc的placement属性，不支持将mem_type设置为Host内存。 **op_attr**：int，算子属性地址对象，此对象使用调用acl.op.create_attr创建。
返回值说明	**handle**：int，算子handle地址对象。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无

10.9.21 函数：destroy_handle

C函数原型	**void aclopDestroyHandle(aclopHandle *handle)**
Python函数	**acl.op.destroy_handle(handle)**
函数功能	销毁一个执行算子的handle，同步接口。
输入说明	**handle**：int，待销毁的acl的算子Handle的地址对象。
返回值说明	无
约束说明	销毁类接口：调用该接口后，不能继续使用已释放或销毁的资源，建议调用该接口后，将相关资源设置为无效值（例如，置为0或None）。
注意事项	无

10.9.22 函数：infer_shape

C函数原型	**aclError aclopInferShape(const char *opType, int numInputs, aclTensorDesc *inputDesc[], aclDataBuffer *inputs[], int numOutputs, aclTensorDesc *outputDesc[], aclopAttr *attr)**
Python函数	**ret = acl.op.infer_shape(op_type, in_desc_list, in_list, num_outputs, out_desc_list, attr)**

函数功能	根据算子的输入Shape、输入值推导出算子的输出Shape。
输入说明	**op_type**：int，指定算子类型名称。 **in_desc_list**：list，算子输入tensor的描述。需提前调用**create_tensor_desc**接口创建aclTensorDesc类型 **in_list**：list，算子输入tensor。 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**内存管理**。此处需申请Device上的内存。内存申请接口请参见**内存管理**。 **num_outputs**：int，算子输出tensor的数量。 **attr**：int，算子的属性地址对象。
输出说明	**out_desc_list**：list，算子输出tensor的描述。需提前调用**create_tensor_desc**接口创建aclTensorDesc类型。out_desc_list数组中的元素个数必须与num_outputs参数值保持一致。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。

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**get_tensor_desc_dim_range**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或动态输出，在调用acl.op.infer_shape接口推导算子的输出Shape前，需先调用**set_tensor_desc_name**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**》。） – 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**set_tensor_desc_name**接口设置名称的代码示例如下： **acl.set_tensor_desc_name**(input_tensor_desc[0], "x"); **acl.set_tensor_desc_name**(input_tensor_desc[1], "y0"); **acl.set_tensor_desc_name**(input_tensor_desc[2], "y1"); **acl.set_tensor_desc_name**(input_tensor_desc[0], "z");
注意事项	无

10.9.23 函数：set_attr_data_type

C函数原型	**aclError aclopSetAttrDataType(aclopAttr *attr, const char *attrName, aclDataType attrValue)**
Python函数	**attr, ret = acl.op.set_attr_data_type(attr, attr_name, attr_value)**
函数功能	设置指定属性的数据类型。同步接口。
输入说明	**attr_name**：属性名。 **attr_value**：属性值。

返回值说明	**attr**：aclopAttr类型的指针。需提前调用**acl.op.create_attr**接口创建aclTensorDesc类型。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无

10.9.24 函数：set_attr_list_data_type

C函数原型	**aclError aclopSetAttrListDataType(aclopAttr *attr, const char *attrName, int numValues, const aclDataType values[])**
Python函数	**attr, ret = acl.op.set_attr_list_data_type(attr, attr_name, num_values, values)**
函数功能	设置指定属性列表的数据类型。同步接口。
输入说明	**attr_name**：属性名。 **num_values**：属性值数目。 **values**：属性值的数组。
返回值说明	**attr**：aclopAttr类型的指针。需提前调用**acl.op.create_attr**接口创建aclTensorDesc类型。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无

10.10 CBLAS 接口（blas）

10.10.1 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 * acl.data_type_size(data_type_a)
  - B矩阵：shape = (k, n)；内存大小 = k * n * acl.data_type_size(data_type_b)
  - C矩阵：shape = (m, n)；内存大小 = m * n * acl.data_type_size(data_type_c)

- 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 * acl.data_type_size(data_type_a)
    - B矩阵：shape = (⌈n/16⌉, ⌈k/16⌉, 16, 16)；内存大小 = ⌈k/16⌉ * 16 * ⌈n/16⌉ * 16 * acl.data_type_size(data_type_b)
    - C矩阵：shape = (⌈n/16⌉, ⌈m/16⌉, 16, 16)；内存大小 = ⌈m/16⌉ * 16 * ⌈n/16⌉ * 16 * acl.data_type_size(data_type_c)
  - 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 * acl.data_type_size(data_type_a)
    - B矩阵：shape = (⌈k/32⌉, ⌈n/16⌉, 32, 16)；内存大小 = ⌈k/32⌉ * 32 * ⌈n/16⌉ * 16 * acl.data_type_size(data_type_b)
    - C矩阵：shape = (⌈n/16⌉, ⌈m/16⌉, 16, 16)；内存大小 = ⌈m/16⌉ * 16 * ⌈n/16⌉ * 16 * acl.data_type_size(data_type_c)

说明

⌈ ⌉表示向上对齐。

10.10.2 函数：gemv_ex

C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);**

Python函数	**ret = acl.blas.gemv_ex(trans_a, m, n,alpha, a, lda, data_type_a, x, incx, data_type_x, beta, y, incy, data_type_y, type, stream)**
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。
输入说明	**trans_a**：int，矩阵A是否转置的标记。 **m**：int，矩阵A的行数，存储矩阵乘数据时，行优先。 **n**：int，矩阵A的列数。 **alpha**：int， 用于执行乘操作的标量alpha数据指针的地址。 **a**：int，矩阵A的数据指针的地址。 **lda**：int，A矩阵的主位，此时选择转置，按行优先，则lda为A的列数。 **data_type_a**：int，矩阵A的数据类型。 **x**：int，向量x的数据指针的地址。 **incx**：int，x连续元素之间的步长。 **data_type_x**：int，向量x的数据类型。 **beta**：int，用于执行乘操作的标量beta的数据指针地址。 **y**：int，向量y的数据指针的地址。 **incy**：int，y连续元素之间的步长。 **data_type_y**：int，向量y的数据类型。 **type**：int，计算精度。 **stream**：int，执行算子所在的Stream。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
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
注意事项	无
参考资源	接口调用流程，参见6.3 接口调用流程。

10.10.3 函数：create_handle_for_gemv_ex

C函数原型	**aclError aclblasCreateHandleForGemvEx(aclTransType transA,** **int m,** **int n,** **aclDataType dataTypeA,** **aclDataType dataTypeX,** **aclDataType dataTypeY,** **aclComputeType type,** **aclopHandle **handle)**
Python函数	**handle, ret = acl.blas.create_handle_for_gemv_ex(trans_a, m, n, data_type_a, data_type_x, data_type_y, type)**
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**acl.op.execute_with_handle**接口执行算子。
输入说明	**trans_a**：int，矩阵A是否转置的标记。 **m**：int，矩阵A的行数，存储矩阵乘数据时，行优先。 **n**：int，矩阵A的列数。 **data_type_a**：int，矩阵A的数据类型。 **data_type_x**：int，向量x的数据类型。 **data_type_y**：int，向量y的数据类型。 **type**：int，计算精度。
返回值说明	**handle**：int，“执行算子的handle数据的指针”的指针。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
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
注意事项	无

10.10.4 函数：hgemv

C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)**
Python函数	**ret = acl.blas.hgemv(trans_a, m, n, alpha, a, lda, x, incx, beta, y, incy, type, stream)**
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。
输入说明	**trans_a**：int，矩阵A是否转置的标记 **m**：int，矩阵A的行数，存储矩阵乘数据时，行优先 **n**：int，矩阵A的列数 **alpha**：int，用于执行乘操作的标量alpha数据指针的地址 **a**：int，矩阵A的数据指针的地址 **lda**：int，A矩阵的主位，此时选择转置，按行优先，则lda为A的列数 **x**：int，向量x的数据指针的地址 **incx**：int，x连续元素之间的步长 **beta**：int，用于执行乘操作的标量beta的数据指针地址 **y**：int，向量y的数据指针的地址 **incy**：int，y连续元素之间的步长 **type**：int，计算精度 **stream**：int，执行算子所在的Stream
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无
参考资源	接口调用流程，参见6.3 接口调用流程。

10.10.5 函数：create_handle_for_hgemv

C函数原型	**aclError aclblasCreateHandleForHgemv(aclTransType transA,** **int m,** **int n,** **aclComputeType type,** **aclopHandle **handle)**
Python函数	**handle, ret = acl.blas.create_handle_for_hgemv(trans_a, m, n, type)**
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**acl.op.execute_with_handle**接口执行算子。
输入说明	**trans_a**：int，矩阵A是否转置的标记。 **m**：int，矩阵A的行数，存储矩阵乘数据时，行优先。 **n**：int，矩阵A的列数。 **type**：int，计算精度。
返回值说明	**handle**：int，“执行算子的handle数据的指针”的指针。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无

10.10.6 函数：s8gemv

C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)**
Python函数	**ret = acl.blas.s8gemv(trans_a, m, n, alpha, a, lda, x, incx, beta, y, incy, type, stream)**
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。
输入说明	**trans_a**：int，矩阵A是否转置的标记。 **m**：int，矩阵A的行数，存储矩阵乘数据时，行优先。 **n**：int，矩阵A的列数。 **alpha**：int， 用于执行乘操作的标量alpha数据指针的地址。 **a**：int，矩阵A的数据指针的地址。 **lda**：int，A矩阵的主位，此时选择转置，按行优先，则lda为A的列数。 **x**：int，向量x的数据指针的地址。 **incx**：int，x连续元素之间的步长。 **beta**：int，用于执行乘操作的标量beta的数据指针地址。 **y**：int，向量y的数据指针的地址。 **incy**：int，y连续元素之间的步长。 **type**：int，计算精度。 **stream**：int，执行算子所在的Stream。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无
参考资源	接口调用流程，参见6.3 接口调用流程。

10.10.7 函数：create_handle_for_s8gemv

C函数原型	**aclError aclblasCreateHandleForS8gemv(aclTransType transA,** **int m,** **int n,** **aclComputeType type,** **aclopHandle **handle)**
Python函数	**handle, ret = acl.blas.create_handle_for_s8gemv(trans_a, m, n, type)**
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**acl.op.execute_with_handle**接口执行算子。
输入说明	**trans_a**：int，矩阵A是否转置的标记。 **m**：int，矩阵A的行数，存储矩阵乘数据时，行优先。 **n**：int，矩阵A的列数。 **type**：int，计算精度。
返回值说明	**handle**：int，“执行算子的handle数据的指针”的指针。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无

10.10.8 函数：gemm_ex

C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)**
Python函数	**ret = acl.blas.gemm_ex(trans_a, trans_b, trans_c,** **m, n, k,** **alpha, matrix_a, lda, data_type_a,** **matrix_b, ldb, data_type_b,** **beta, matrix_c, ldc, data_type_c,** **type, stream)**
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。

输入说明	**trans_a**：int，矩阵A是否转置的标记。 **trans_b**：int，B矩阵是否转置的标记。 **trans_c**：int，C矩阵的标记，当前仅支持aclTrans_N。 **m**：int，矩阵A的行数与矩阵C的行数。 **n**：int，矩阵B的列数与矩阵C的列数。 **k**：int，矩阵A的列数与矩阵B的行数。 **alpha**：int， 用于执行乘操作的标量alpha数据指针的地址。 **matrix_a**：int，矩阵A的数据指针的地址。 **lda**：int，A矩阵的主维，此时选择转置，按行优先，则lda为A的列数。 **data_type_a**：int，矩阵A的数据类型。 **matrix_b**：int，矩阵B的数据指针的地址。 **ldb**：int，B矩阵的主维，此时选择转置，按行优先，则leading dimension为B的列数。 **data_type_b**：int，矩阵B的数据类型。 **beta**：int，用于执行乘操作的标量beta的数据指针地址。 **matrix_c**：int，矩阵C的数据指针的地址。 **ldc**：int，C矩阵的主维，当前预留。 **data_type_c**：int，矩阵C的数据类型。 **type**：int，计算精度。 **stream**：int，执行算子所在的Stream。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
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 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
注意事项	无
参考资源	接口调用流程、示例，参见6.3 接口调用流程。

10.10.9 函数：create_handle_for_gemm_ex

C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)**
Python函数	**handle, ret = acl.blas.create_handle_for_gemm_ex(trans_a, trans_b, trans_c, m, n, k, data_type_a, data_type_b, data_type_c, type)**
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**acl.op.execute_with_handle**接口执行算子。
输入说明	**trans_a**：int，矩阵A是否转置的标记。 **trans_b**：int，B矩阵是否转置的标记。 **trans_c**：int，C矩阵的标记，当前仅支持aclTrans_N。 **m**：int，矩阵A的行数与矩阵C的行数。 **n**：int，矩阵B的列数与矩阵C的列数。 **k**：int，矩阵A的列数与矩阵B的行数。 **data_type_a**：int，矩阵A的数据类型。 **data_type_b**：int，矩阵B的数据类型。 **data_type_c**：int，矩阵C的数据类型。 **type**：int，计算精度。
返回值说明	**handle**：int，“执行算子的handle数据的指针”的指针。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。

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 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
注意事项	无

10.10.10 函数：hgemm

C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)**
Python函数	**ret = acl.blas.hgemm(trans_a, trans_b, trans_c,** **m, n, k,** **alpha, matrix_a, lda,** **matrix_b, ldb,** **beta, matrix_c, ldc,** **type, stream)**
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。

输入说明	**trans_a**：int，矩阵A是否转置的标记。 **trans_b**：int，B矩阵是否转置的标记。 **trans_c**：int，C矩阵的标记，当前仅支持aclTrans_N。 **m**：int，矩阵A的行数与矩阵C的行数。 **n**：int，矩阵B的列数与矩阵C的列数。 **k**：int，矩阵A的列数与矩阵B的行数。 **alpha**：int， 用于执行乘操作的标量alpha数据指针的地址。 **matrix_a**：int，矩阵A的数据指针的地址。 **lda**：int，A矩阵的主维，此时选择转置，按行优先，则lda为A的列数. **matrix_b**：int，矩阵B的数据指针的地址。 **ldb**：int，B矩阵的主维，此时选择转置，按行优先，则leading dimension为B的列数。 **beta**：int，用于执行乘操作的标量beta的数据指针地址。 **matrix_c**：int，矩阵C的数据指针的地址。 **ldc**：int，C矩阵的主维，当前预留。 **type**：int，计算精度。 **stream**：int，执行算子所在的Stream。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无
参考资源	接口调用流程，参见6.3 接口调用流程。

10.10.11 函数：create_handle_for_hgemm

C函数原型	**aclError aclblasCreateHandleForHgemm(aclTransType transA,** **aclTransType transB,** **aclTransType transC,** **int m,** **int n,** **int k,** **aclComputeType type,** **aclopHandle **handle)**
Python函数	**handle, ret = acl.blas.create_handle_for_hgemm(trans_a, trans_b, trans_c, m, n, k, type)**

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**acl.op.execute_with_handle**接口执行算子。
输入说明	**trans_a**：int，矩阵A是否转置的标记。 **trans_b**：int，B矩阵是否转置的标记。 **trans_c**：int，C矩阵的标记，当前仅支持aclTrans_N。 **m**：int，矩阵A的行数与矩阵C的行数。 **n**：int，矩阵B的列数与矩阵C的列数。 **k**：int，矩阵A的列数与矩阵B的行数。 **type**：int，计算精度。
返回值说明	**handle**：int，“执行算子的handle数据的指针”的指针。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无

10.10.12 函数：s8gemm

C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)**

Python函数	**ret = acl.blas.s8gemm(trans_a, trans_b, trans_c,** **m, n, k,** **alpha, matrix_a, lda,** **matrix_b, ldb,** **beta, matrix_c, ldc,** **type, stream)**
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。
输入说明	**trans_a**：int，矩阵A是否转置的标记。 **trans_b**：int，B矩阵是否转置的标记。 **trans_c**：int，C矩阵的标记，当前仅支持aclTrans_N。 **m**：int，矩阵A的行数与矩阵C的行数。 **n**：int，矩阵B的列数与矩阵C的列数。 **k**：int，矩阵A的列数与矩阵B的行数。 **alpha**：int， 用于执行乘操作的标量alpha数据指针的地址。 **matrix_a**：int，矩阵A的数据指针的地址。 **lda**：int，A矩阵的主维，此时选择转置，按行优先，则lda为A的列数。 **matrix_b**：int，矩阵B的数据指针的地址。 **ldb**：int，B矩阵的主维，此时选择转置，按行优先，则leading dimension为B的列数。 **beta**：int，用于执行乘操作的标量beta的数据指针地址。 **matrix_c**：int，矩阵C的数据指针的地址。 **ldc**：int，C矩阵的主维，当前预留。 **type**：int，计算精度。 **stream**：int，执行算子所在的Stream。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无
参考资源	接口调用流程，参见6.3 接口调用流程。

10.10.13 函数：create_handle_for_s8gemm

C函数原型	**aclError aclblasCreateHandleForS8gemm(aclTransType transA,** **aclTransType transB,** **aclTransType transC,** **int m,** **int n,** **int k,** **aclComputeType type,** **aclopHandle **handle)**
Python函数	**handle, ret = acl.blas.create_handle_for_s8gemm(trans_a, trans_b, trans_c, m, n, k, type)**
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**acl.op.execute_with_handle**接口执行算子。
输入说明	**trans_a**：int，矩阵A是否转置的标记。 **trans_b**：int，B矩阵是否转置的标记。 **trans_c**：int，C矩阵的标记，当前仅支持aclTrans_N。 **m**：int，矩阵A的行数与矩阵C的行数。 **n**：int，矩阵B的列数与矩阵C的列数。 **k**：int，矩阵A的列数与矩阵B的行数。 **type**：int，计算精度。
返回值说明	**handle**：int，“执行算子的handle数据的指针”的指针。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无

10.11 算子编译

10.11.1 函数：register_compile_func

C函数原型	**aclError aclopRegisterCompileFunc(const char *opType, aclopCompileFunc func)**

Python函数	**ret = acl.op.register_compile_func(op_type, func)**
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。 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acl.op.unregister_compile_func接口取消注册，然后再调用acl.op.register_compile_func接口重新注册。

输入说明	**op_type**：str，算子类型。
	func：python函数对象，算子选择器回调函数，函数定义：
	Python侧格式如下：
	`def call_back_func(num_inputs, input_desc, num_outputs, output_desc, op_attr, aclop_kernel_desc):`
	`pass`

返回值说明	ret：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.11.2 函数：unregister_compile_func

C函数原型	**aclError aclopUnregisterCompileFunc(const char *opType)**
Python函数	**ret = acl.op.unregister_compile_func(op_type)**
函数功能	动态Shape场景下，取消注册算子选择器。
输入说明	**op_type**：str，算子类型。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.11.3 函数：create_kernel

C函数原型	**aclError aclopCreateKernel(const char *opType,** **const char *kernelId,** **const char *kernelName,** **void *binData,** **int binSize,** **aclopEngineType enginetype,** **aclDataDeallocator deallocator)**
Python函数	**ret = acl.op.create_kernel(op_type, kernel_id, kernel_name, bin_data, bin_size, enginetype, deallocator)**
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。
输入说明	**op_type**：str，算子类型。 **kernel_id**：str，算子执行时要指定的Kernel ID。 **kernel_name**：str，算子Kernel名称，和算子二进制文件中的kernelName保持一致。 **bin_data**：int，算子Kernel文件的内存地址。 **bin_size**：int，算子Kernel文件的内存大小，单位为byte。 **enginetype**：int，表示算子执行引擎，该参数只有acl.op.update.params接口的**compile_flag**参数值为ACL_COMPILE_SYS时有效。 ● 0：ACL_ENGINE_SYS，不关心具体执行引擎时填写。 ● 1：ACL_ENGINE_AICORE，将算子编译成AI Core算子。 ● 2：ACL_ENGINE_VECTOR，将算子编译成Vector Core算子。 **deallocator**：int，指定是否自动释放bin_data内存。 ● 0：不自动释放，数据由调用者自行编写代码进行释放。 ● 1：自动释放，内部会设置回调函数释放bin_data内存。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

10.11.4 函数：set_kernel_args

C函数原型	**aclError aclopSetKernelArgs(aclopKernelDesc *kernelDesc,** **const char *kernelId,** **uint32_t blockDim,** **const void *args,** **uint32_t argSize)**
Python函数	**ret = acl.op.set_kernel_args(kernel_desc, kernel_id, block_dim, args, arg_size)**
函数功能	动态Shape场景下，设置算子Tiling参数、执行并发数。
输入说明	**kernel_desc**：int，Kernel描述缓存，aclopKernelDesc类型的指针。 **kernel_id**：str，算子执行时要指定的Kernel ID，与调用acl.op.create.kernel时传递的kernel_id一致。 **block_dim**：int，Kernel执行的并发数。 **bin_data**：int，算子Kernel文件的内存地址。 **bin_size**：int，算子Kernel文件的内存大小，单位为byte。 **args**：int，Tiling参数，需要通过numpy数组tobytes方法获取bytes对象，然后使用**acl.utils.bytes_to_ptr()**接口获取Tiling参数指针地址。 **arg_size**：int，Tiling参数内存大小，单位为Byte，为numpy.tobytes()转换得到bytes的对象长度，可用len函数获取。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无

10.11.5 函数：set_kernel_workspace_sizes

C函数原型	**aclError aclopSetKernelWorkspaceSizes(aclopKernelDesc *kernelDesc, int numWorkspaces, size_t *workspaceSizes)**
Python函数	**ret = acl.op.set_kernel_workspace_sizes(kernel_desc, num_workspaces, workspace_sizes)**
函数功能	动态Shape场景下，设置算子Workspace参数。
输入说明	**kernel_desc**：int，Kernel描述缓存，aclopKernelDesc类型的指针。 **num_workspaces**：int，Workspaces个数。 **workspace_sizes**：int，Workspaces大小的数组地址。

返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	为非必须接口，根据算子情况可选。
注意事项	无

10.11.6 函数：update_params

C函数原型	**aclError aclopUpdateParams(const char *opType, int numInputs, const aclTensorDesc *const inputDesc[], int numOutputs, const aclTensorDesc *const outputDesc[], const aclopAttr *attr)**
Python函数	**ret = acl.op.update_params(op_type, input_desc, output_desc, attr)**
函数功能	动态Shape场景下，设置算子Tiling参数、执行并发数。
输入说明	**op_type**：str，算子类型名称。 **input_desc**：list，算子输入tensor的描述。 **output_desc**：list，算子输出tensor的描述。 **attr**：int，算子属性。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无

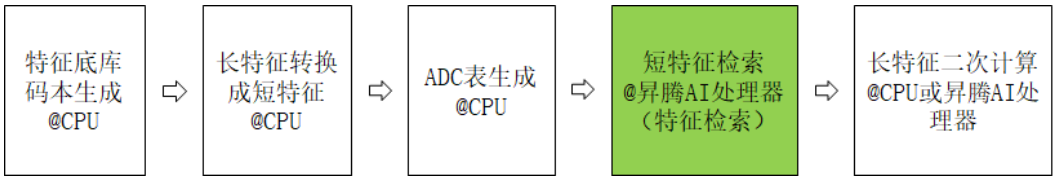
10.12 特征向量检索

10.12.1 总体说明

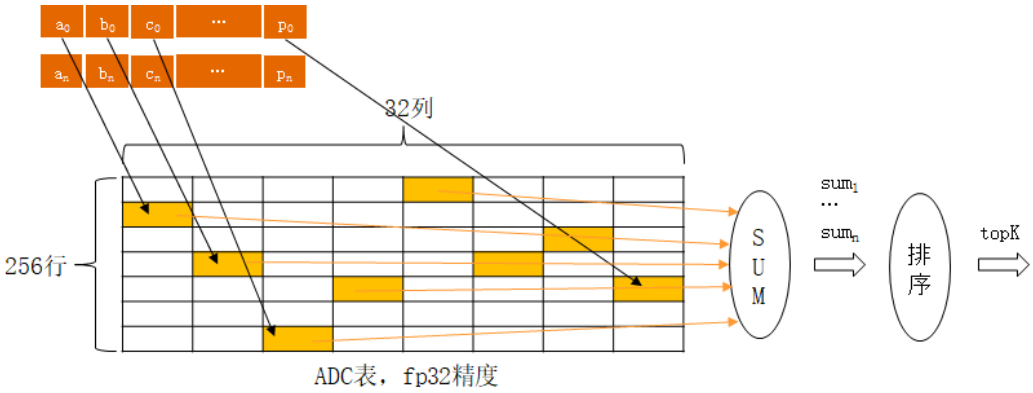
在图像检索场景，通常使用深度学习算法处理目标等感兴趣的图像，提取出高维特征向量，并将一组感兴趣的图像提取得到的特征向量作为底库，待搜索的特征向量与底库中的向量依次进行相似度计算，按相似度排序返回检索结果。为了进一步降低存储和计算的开销，对浮点型长特征向量进行量化处理、转换为短特征再进行检索比对的技术被广泛应用。

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。

10.12.2 函数：init

C函数原型	**aclError aclfvInit(aclfvInitPara *initPara)**
Python函数	**ret = acl.fv.init(init_para)**
函数功能	初始化特征检索模块。同步接口。与 **acl.fv.release**接口配套使用。 Atlas 200/500 A2推理产品不支持该接口。
输入说明	**init_para**：int，指定特征向量检索的初始化参数。 ● 调用**acl.fv.create_init_para**接口创建aclfvInitPara类型的数据。 ● 1:N场景下调用**acl.fv.set_1n_top_num**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**acl.fv.set_nm_top_num**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	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	一个进程内只能调用一次**acl.fv.init**接口。

注意事项	无
参考资源	无

10.12.3 函数：release

C函数原型	**aclError aclfvRelease()**
Python函数	**ret = acl.fv.release()**
函数功能	特征检索模块去初始化，释放内存空间，与**acl.fv.init**接口配套使用。同步接口。 Atlas 200/500 A2推理产品不支持该接口。
输入说明	无
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无
参考资源	无

10.12.4 函数：repo_add

C函数原型	**aclError aclfvRepoAdd(aclfvSearchType type, aclfvFeatureInfo *featureInfo)**
Python函数	**ret = acl.fv.repo_add(type, feature_info)**
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。同步接口。 Atlas 200/500 A2推理产品不支持该接口。
输入说明	● **type**：int，检索类型，请参见**10.14.13 aclfvSearchType**处说明。 ● **feature_info**：int，指针地址，指定特征描述信息。需提前调用**acl.fv.create_feature_info**接口创建aclfvFeatureInfo类型的数据。

返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
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10.5.3.1 函数：create_stream处的说明。
注意事项	无
参考资源	无

10.12.5 函数：repo_del

C函数原型	**aclError aclfvRepoDel(aclfvSearchType type, aclfvRepoRange *repoRange)**
Python函数	**ret = acl.fv.repo_del(type, repo_range)**
函数功能	删除底库。同步接口。 Atlas 200/500 A2推理产品不支持该接口。
输入说明	● **type**：int，检索类型，请参见10.14.13 aclfvSearchType处说明。 ● **repo_range**：int，指针地址，指定特征删除范围。需提前调用acl.fv.create_repo_range接口创建aclfvRepoRange类型的数据。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
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10.5.3.1 函数：create_stream处的说明。
注意事项	无
参考资源	无

10.12.6 函数：delete

C函数原型	**aclError aclfvDel(aclfvFeatureInfo *featureInfo)**
Python函数	**ret = acl.fv.delete(feature_info)**
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。同步接口。 Atlas 200/500 A2推理产品不支持该接口。
输入说明	**feature_info**：int，指针地址，指定特征描述信息。 需提前调用**acl.fv.create_feature_info**接口创建aclfvFeatureInfo类型的数据。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
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**10.5.3.1 函数：create_stream**处的说明。
注意事项	无
参考资源	无

10.12.7 函数：modify

C函数原型	**aclError aclfvModify(aclfvFeatureInfo *featureInfo)**
Python函数	**ret = acl.fv.modify(feature_info)**
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。同步接口。 Atlas 200/500 A2推理产品不支持该接口。
输入说明	**feature_info**：int，指针地址，指定特征描述信息。需提前调用**acl.fv.create_feature_info**接口创建aclfvFeatureInfo类型的数据。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
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**10.5.3.1 函数：create_stream**处的说明。

注意事项	无
参考资源	无

10.12.8 函数：search

C函数原型	**aclError aclfvSearch(aclfvSearchType type, aclfvSearchInput *searchInput, aclfvSearchResult *searchRst)**
Python函数	**ret = acl.fv.search(type, search_input, search_result)**
函数功能	特征1:N或N:M检索。同步接口。 Atlas 200/500 A2推理产品不支持该接口。
输入说明	• **type**：int，指定特征描述信息，请参见**10.14.13 aclfvSearchType**处说明。 • **search_input**：int，指针地址，检索输入信息。需提前调用**acl.fv.create_search_input**接口创建aclfvSearchInput类型的数据。 • **search_result**：int，指针地址，检索输出结果。需提前调用**acl.fv.create_search_result**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**acl.fv.create_search_input**接口的topk参数设置，但与调用**acl.fv.init**接口进行初始化时设置的topK结果数量最大值有关，若调用**acl.fv.create_search_input**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**acl.fv.create_search_input**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**acl.fv.create_search_input**接口设置的topk参数值返回检索结果。
返回值说明	**ret**：int，错误码。 • 返回0表示成功。 • 返回**其它值**表示失败。
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**acl.fv.search**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**10.5.3.1 函数：create_stream**处的说明。
注意事项	无

参考资源	无

10.12.9 函数：create_init_para

C函数原型	**aclfvInitPara *aclfvCreateInitPara(uint64_t fsNum)**
Python函数	**init_para = acl.fv.create_init_para(fs_num)**
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。同步接口。 如需销毁aclfvInitPara类型的数据，请参见**10.12.10 函数：destory_init_para**。 Atlas 200/500 A2推理产品不支持该接口。
输入说明	**fs_num**：int，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	**init_para** :int。 ● 返回aclfvInitPara类型，表示成功。 ● 返回0，表示失败。
约束说明	无
注意事项	无
参考资源	无

10.12.10 函数：destory_init_para

C函数原型	**aclError aclfvDestroyInitPara(aclfvInitPara *initPara)**
Python函数	**ret = acl.fv.destory_init_para(init_para)**
函数功能	销毁通过**acl.fv.create_init_para**接口创建的aclfvInitPara类型的数据。同步接口。 Atlas 200/500 A2推理产品不支持该接口。
输入说明	**init_para**：int，待销毁的aclfvInitPara类型。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无

注意事项	无
参考资源	无

10.12.11 函数：set_1n_top_num

C函数原型	**aclError aclfvSet1NTopNum(aclfvInitPara *initPara, uint32_t maxTopNumFor1N)**
Python函数	**ret = acl.fv.set_1n_top_num(init_para, max_top_num_for_1n)**
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。同步接口。 Atlas 200/500 A2推理产品不支持该接口。
输入说明	● **init_para**：int，指定特征向量检索的初始化参数。需提前调用**acl.fv.create_init_para**接口创建aclfvInitPara类型的数据。 ● **max_top_num_for_1n**：int，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	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无
参考资源	无

10.12.12 函数：set_nm_top_num

C函数原型	**aclError aclfvSetNMTopNum(aclfvInitPara *initPara, uint32_t maxTopNumForNM)**
Python函数	**ret = acl.fv.set_nm_top_num(init_para, max_top_num_for_nm)**
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。同步接口。 Atlas 200/500 A2推理产品不支持该接口。

输入说明	● **init_para**：int，指定特征向量检索的初始化参数。需提前调用acl.fv.create_init_para接口创建aclfvInitPara类型的数据。 ● **max_top_num_for_nm**：int，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	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无
参考资源	无

10.12.13 函数：create_feature_info

C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)**
Python函数	**feature_info = acl.fv.create_feature_info(id0, id1, offset, feature_len, feature_count, feature_data, feature_len)**
函数功能	创建aclfvFeatureInfo类型的数据，表示创建特征描述信息。同步接口。 如需销毁aclfvFeatureInfo类型的数据，请参见10.12.14 函数：destroy_feature_info。 Atlas 200/500 A2推理产品不支持该接口。
输入说明	● **id0**：int，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**：int，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**：int，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。 ● **feature_len**：int，单个特征长度，当前固定为36Byte，系统内部会校验。 ● **feature_count**：int，特征数量，1:N最大100万，N:M最大1000万。 ● **feature_data**：int，特征值，根据特征长度连续存储，每条特征前4Byte值为0。 ● **feature_len**：int，featureData申请的内存长度，用于校验。

返回值说明	**feature_info**：int，指针地址。 ● 返回aclfvFeatureInfo类型，表示成功。 ● 返回0，表示失败。
约束说明	无
注意事项	无
参考资源	无

10.12.14 函数：destroy_feature_info

C函数原型	**aclError aclfvDestroyFeatureInfo(aclfvFeatureInfo *featureInfo)**
Python函数	**ret = acl.fv.destroy_feature_info(feature_info)**
函数功能	销毁通过**acl.fv.create_feature_info**接口创建的aclfvFeatureInfo类型的数据。同步接口。 Atlas 200/500 A2推理产品不支持该接口。
输入说明	**feature_info**：int，指针地址，待销毁的aclfvFeatureInfo类型。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无
参考资源	无

10.12.15 函数：create_repo_range

C函数原型	**aclfvRepoRange *aclfvCreateRepoRange(uint32_t id0Min, uint32_t id0Max, uint32_t id1Min, uint32_t id1Max)**
Python函数	**repo_range = acl.fv.create_repo_range(id0min, id0max, id1min, id1max)**
函数功能	创建aclfvRepoRange类型的数据，表示创建特征库范围的参数。同步接口。 如需销毁aclfvRepoRange类型的数据，请参见**10.12.16 函数：destroy_repo_range**。 Atlas 200/500 A2推理产品不支持该接口。

输入说明	• **id0min**：int，id0起始值，取值范围：[0,1023]，需小于等于id0Max。 • **id0max**：int，id0最大值，取值范围：[0,1023]。 • **id1min**：int，id1起始值，取值范围：[0,1023]，需小于等于id1Max • **id1max**：int，id1最大值，取值范围：[0,1023]。
返回值说明	**repo_range**：int，指针地址。 • 返回aclfvRepoRange类型，表示成功。 • 返回0，表示失败。
约束说明	无
注意事项	无
参考资源	无

10.12.16 函数：destroy_repo_range

C函数原型	**aclError aclfvDestroyRepoRange(aclfvRepoRange *repoRange)**
Python函数	**ret = acl.fv.destroy_repo_range(repo_range)**
函数功能	销毁通过**acl.fv.create_repo_range**接口创建的aclfvRepoRange类型的数据。同步接口。 Atlas 200/500 A2推理产品不支持该接口。
输入说明	**repo_range**：int，指针地址，待销毁的aclfvRepoRange类型。
返回值说明	**ret**：int，错误码。 • 返回0表示成功。 • 返回**其它值**表示失败。
约束说明	无
注意事项	无
参考资源	无

10.12.17 函数：create_query_table

C函数原型	**aclfvQueryTable *aclfvCreateQueryTable(uint32_t queryCnt, uint32_t tableLen, uint8_t *tableData, uint32_t tableDataLen)**
Python函数	**query_table = acl.fv.create_query_table(query_cnt, table_len, table_data, table_data_len)**

函数功能	创建aclfvQueryTable类型的数据，表示创建检索输入表信息。同步接口。 如需销毁aclfvQueryTable类型的数据，请参见**10.12.18 函数：destroy_query_table**。 Atlas 200/500 A2推理产品不支持该接口。
输入说明	• **query_cnt**：int，检索请求数量，1:N场景为1，N:M场景最大1024。 • **table_len**：int，单个表长度，长度固定为32KB，系统内部会校验。 • **table_data**：int，特征值列表，根据特征长度连续存储。 • **table_data_len**：int，tableData申请的内存长度，用于校验。
返回值说明	**query_table**：int，指针地址。 • 返回aclfvQueryTable类型，表示成功。 • 返回0，表示失败。
约束说明	无
注意事项	无
参考资源	无

10.12.18 函数：destroy_query_table

C函数原型	**aclError aclfvDestroyQueryTable(aclfvQueryTable *queryTable)**
Python函数	**ret = acl.fv.destroy_query_table(query_table)**
函数功能	销毁通过**acl.fv.create_query_table**接口创建的aclfvQueryTable类型的数据。同步接口。 Atlas 200/500 A2推理产品不支持该接口。
输入说明	**query_table**：int，指针地址，待销毁的aclfvQueryTable类型。
返回值说明	**ret**：int，错误码。 • 返回0表示成功。 • 返回**其它值**表示失败。
约束说明	无
注意事项	无
参考资源	无

10.12.19 函数：create_search_input

C函数原型	**aclfvSearchInput *aclfvCreateSearchInput(aclfvQueryTable *queryTable, aclfvRepoRange *repoRange, uint32_t topk)**
Python函数	**search_input = acl.fv.create_search_input(query_table, repo_range, topk)**
函数功能	创建aclfvSearchInput类型的数据，表示创建检索任务输入信息。同步接口。 如需销毁aclfvSearchInput类型的数据，请参见**10.12.20 函数：destroy_search_input**。 Atlas 200/500 A2推理产品不支持该接口。
输入说明	• **query_table**：int，指针地址，检索的输入表信息。调用**acl.fv.create_query_table**接口创建aclfvQueryTable类型的数据。 • **repo_range**：int，指针地址，检索的特征库范围。调用**acl.fv.create_repo_range**接口创建aclfvRepoRange类型的数据。 • **topk**：单个检索请求需要返回的结果数量。
返回值说明	**search_input**：int，指针地址。 • 返回aclfvSearchInput类型，表示成功。 • 返回0，表示失败。
约束说明	无
注意事项	无
参考资源	无

10.12.20 函数：destroy_search_input

C函数原型	**aclError aclfvDestroySearchInput(aclfvSearchInput *searchInput)**
Python函数	**ret = acl.fv.destroy_search_input(search_input)**
函数功能	销毁通过**acl.fv.create_search_input**接口创建的aclfvSearchInput类型的数据。同步接口。 Atlas 200/500 A2推理产品不支持该接口。
输入说明	**search_input**：int，指针地址，待销毁的aclfvSearchInput类型。
返回值说明	**ret**：int，错误码。 • 返回0表示成功。 • 返回**其它值**表示失败。

约束说明	无
注意事项	无
参考资源	无

10.12.21 函数：create_search_result

C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)**
Python函数	**search_result = acl.fv.create_search_result(query_cnt, result_num, result_num_data_len, id_0, id_1, result_offset, result_distance, data_len)**
函数功能	创建aclfvSearchResult类型的数据，表示创建检索结果信息。同步接口。 如需销毁aclfvSearchResult类型的数据，请参见**10.12.22 函数：destroy_search_result**。 Atlas 200/500 A2推理产品不支持该接口。
输入说明	• **query_cnt**：int，检索请求个数，1:N场景为1。 • **result_num**：int，每个检索请求的结果个数。 • **result_num_data_len**：int，resultNum的内存总长度。 • **id_0**：int，一级库id，总个数为topK*queryCnt。 • **id_1**：int，二级库id，总个数为topK*queryCnt。 • **result_offset**：int，每个检索请求的检索结果对应底库的偏移，总个数为topK*queryCnt。 • **result_distance**：int，每个检索结果与检索请求间的距离，总个数为topK*queryCnt。 • **data_len**：int，申请的内存大小，计算公式为：topK*queryCnt*4Byte。
返回值说明	**search_result**：int，指针地址。 • 返回aclfvSearchResult类型，表示成功。 • 返回0，表示失败。
约束说明	无
注意事项	无
参考资源	无

10.12.22 函数：destroy_search_result

C函数原型	**aclError aclfvDestroySearchResult(aclfvSearchResult *searchResult)**
Python函数	**ret = acl.fv.destroy_search_result(search_result)**
函数功能	销毁通过**acl.fv.create_search_result**接口创建的aclfvSearchResult类型的数据。同步接口。 Atlas 200/500 A2推理产品不支持该接口。
输入说明	**search_result**：int，指针地址，待销毁的aclfvSearchResult类型。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无
参考资源	无

10.13 Profiling 数据采集

10.13.1 Profiling pyACL API（通过 Profiling pyACL API 采集并落盘性能数据）

10.13.1.1 功能介绍

功能

该章节描述通过Profiling pyACL 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《**性能分析工具使用指南**》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：

- **acl.prof.init**接口、**acl.prof.start**接口、**acl.prof.stop**接口、**acl.prof.finalize**接口配合使用，实现该方式的性能数据采集。该方式可获取pyACL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**acl.init**接口，在pyACL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pyACL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acl.init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acl.init接口处的说明，不在本章节描述。

总体约束

不能与10.13.3 Profiling pyACL API for Subscription（订阅算子信息的Profiling pyACL API）的接口交叉调用：acl.prof.init接口和acl.prof.finalize接口之间不能调用acl.prof.model_subscribe接口、aclprofGet*接口、acl.prof.model_un_subscribe接口。

接口约束说明

- **调用接口要求**：
  - acl.prof.init接口必须在acl.init接口之后、模型加载之前调用。

    如果已经通过acl.init接口配置了Profiling信息，则调用acl.prof.init接口、acl.prof.start接口、acl.prof.stop接口、acl.prof.finalize时，会返回报错。

    如果没有调用acl.prof.init接口，调用acl.prof.start接口、acl.prof.stop接口、acl.prof.finalize时，会返回报错。
  - acl.prof.start接口在模型执行之前调用，若在模型执行过程中调用acl.prof.start接口，Profling采集到的数据为调用acl.prof.start接口之后的数据，可能导致数据不完整。

    调用acl.prof.start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acl.prof.start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，acl.prof.init接口与acl.prof.finalize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。
  - acl.prof.start接口与acl.prof.stop接口需配对使用。
  - Profiling msproftx采集功能相关接口须在acl.prof.start接口与acl.prof.stop接口之间调用。其中配对使用的接口有：acl.prof.create_stamp/acl.prof.destroy_stamp、acl.prof.push/acl.prof.pop、acl.prof.range_start/acl.prof.range_stop。
  - 调用acl.finalize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acl.finalize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：

    acl.init接口-->acl.prof.init接口-->acl.prof.start接口(指定Device 0和Device 1)-->模型1加载-->模型1执行-->acl.prof.stop接口(与acl.prof.start接口的aclprofConfig数据保持一致)-->acl.prof.start接口(指定Device 1和Device 2)-->模型2加载-->模型2执行-->acl.prof.stop接口(与acl.prof.start接口的aclprofConfig数据保持一致)-->acl.prof.finalize接口-->执行其它任务-->模型卸载-->acl.finalize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acl.prof.start接口，指定的Device重复”为例：

acl.init接口-->acl.prof.init接口-->acl.prof.start接口(指定Device 0和Device 1)-->acl.prof.start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-->acl.prof.stop接口-->acl.prof.stop接口-->acl.prof.finalize-->执行其它任务-->模型卸载-->acl.finalize接口

10.13.1.2 函数：create_config

C函数原型	**aclprofConfig *aclprofCreateConfig(uint32_t *deviceIdlist, uint32_t deviceNums, aclprofAicoreMetrics aicoreMetrics, aclprofAicoreEvents *aicoreEvents, uint64_t dataTypeConfig)**
Python函数	**prof_config = acl.prof.create_config(device_list，aicore_metrics, aicore_events, data_type_config)**
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。
输入说明	**device_list**：Device ID列表。 **aicore_metrics**：表示aclprofAicoreMetrics。 **aicore_events**：表示AI Core事件，目前配置为0。 **data_type_config**： 用户选择如下多个值进行逻辑或（例如：ACL_PROF_ACL_API|ACL_PROF_AICORE_METRICS），作为data_type_config参数值。每个值表示某一类性能数据，详细说明如下： ● ACL_PROF_ACL_API：表示采集pyACL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：表示采集AI Core算子的执行时间，Device间的同步异步内存复制时延等。 ● 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_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_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	**prof_config**：int。 ● 返回非0值表示成功，返回值为返回aclprofConfig类型的指针。 ● 返回0表示失败。

10.13.1.3 函数：set_config

C函数原型	**aclError aclprofSetConfig(aclprofConfigType configType, const char *config, uint32_t configLength)**

Python函数	**ret = acl.prof.set_config(config_type, config)**
函数功能	**acl.prof.create_config**接口的扩展接口，用于设置性能数据采集参数。同步接口。 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。
输入说明	**config_type**：int，用户选择如下多个常量进行逻辑或（例如：ACL_PROF_SYS_HARDWARE_MEM_FREQ | ACL_PROF_SYS_IO_FREQ），作为config_type参数值。每个常量表示某一类性能数据，详细说明如下： ● ACL_PROF_STORAGE_LIMIT = 1：指定落盘目录允许存放的最大文件容量，有效取值范围为[200MB, 4294967296MB]。 ● ACL_PROF_SYS_HARDWARE_MEM_FREQ = 3：DDR、LLC的读写带宽数据采集频率以及acc_pmu数据和SOC传输带宽信息采集频率，范围[1,100]。 ● ACL_PROF_LLC_MODE = 4：LLC Profiling采集事件。 – 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 = 5：NIC采集频率，范围[1,100]。容器场景下本参数不生效。 ● ACL_PROF_DVPP_FREQ = 7：DVPP采集频率，范围[1,100]。 **config**：str，配置项的参数值。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	先调用**acl.prof.set_config**接口再调用**acl.prof.start**接口，可根据需求选择调用该接口。

10.13.1.4 函数：destroy_config

C函数原型	**aclError aclprofDestroyConfig(const aclprofConfig *profilerConfig)**
Python函数	**ret = acl.prof.destroy_config(profiler_config)**
函数功能	销毁aclprofConfig类型的数据，只能销毁通过**acl.prof.create_config**接口创建的aclprofConfig类型。
输入说明	**profiler_config**：待销毁的aclprofConfig类型的指针地址。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。

10.13.1.5 函数：init

C函数原型	**aclError aclprofInit(const char *profilerResultPath, size_t length)**
Python函数	**ret = acl.prof.init(profiler_result_path)**
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。同步接口。
输入说明	**profiler_result_path**：str，指定保存性能数据的文件的路径，此路径为绝对路径。
返回值说明	**ret**：int，错误码。 ● 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 Profiling性能数据采集。

10.13.1.6 函数：start

C函数原型	**aclError aclprofStart(const aclprofConfig *profilerConfig)**
Python函数	**ret = acl.prof.start(profiler_config)**
函数功能	下发Profiling请求，使能对应数据的采集。同步接口。 用户可根据需要，在模型执行过程中按需调用acl.prof.start接口，Profling采集到的数据为调用该接口之后的数据。
输入说明	**profiler_config**：int，指定Profiling配置数据。 需提前调用acl.prof.create_config接口创建aclprofConfig类型的数据。
返回值说明	**ret**：int，错误码。 ● 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 Profiling性能数据采集。

10.13.1.7 函数：stop

C函数原型	**aclError aclprofStop(const aclprofConfig *profilerConfig)**
Python函数	**ret = acl.prof.stop(profiler_config)**
函数功能	停止Profiling数据采集。同步接口。

输入说明	**profiler_config**：int，指定停止Profiling数据采集的配置的指针地址。 与**acl.prof.start**接口中的aclprofConfig类型数据保持一致。
返回值说明	**ret**：int，错误码。 ● 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 Profiling性能数据采集**。

10.13.1.8 函数：finalize

C函数原型	**aclError aclprofFinalize()**
Python函数	**ret = acl.prof.finalize()**
函数功能	结束Profiling。同步接口。
输入说明	无。
返回值说明	**ret**：int，错误码。 ● 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 Profiling性能数据采集**。

10.13.2 Profiling pyACL API for Extension（Profiling pyACL API扩展接口）

10.13.2.1 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（**acl.prof.start**接口、**acl.prof.stop**接口之间）调用Profiling pyACL 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《**性能分析工具使用指南**》下的“Profiling数据解析”和“Profiling数据导出”。

接口调用方式：在**acl.prof.start**和**acl.prof.stop**接口之间调用**acl.prof.create_stamp**、**acl.prof.set_stamp_trace_message**、**acl.prof.mark**、**acl.prof.push**、**acl.prof.pop**、**acl.prof.range_start**、**acl.prof.range_stop**、**acl.prof.destroy_stamp**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acl.prof.start接口与acl.prof.stop接口之间调用。其中配对使用的接口有：acl.prof.create_stamp/acl.prof.destroy_stamp、acl.prof.push/acl.prof.pop、acl.prof.range_start/acl.prof.range_stop。
- **接口调用顺序**：acl.prof.start接口(指定Device 0和Device 1)-->acl.prof.create_stamp接口-->acl.prof.set_stamp_trace_message接口-->acl.prof.mark接口-->(acl.prof.push接口-->acl.prof.pop接口)或(acl.prof.range_start接口-->acl.prof.range_stop接口)-->acl.prof.destroy_stamp接口-->acl.prof.stop接口(与acl.prof.start接口的aclprofConfig数据保持一致)。

10.13.2.2 函数：create_stamp

C函数原型	**void *aclprofCreateStamp()**
Python函数	**stamp = acl.prof.create_stamp()**
函数功能	创建msproftx事件标记。后续调用10.13.2.4 函数：mark、10.13.2.3 函数：set_stamp_trace_message、10.13.2.5 函数：push和10.13.2.7 函数：range_start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。同步接口。 与10.13.2.9 函数：destroy_stamp接口配对使用，需提前调用10.13.1.6 函数：start接口。
输入说明	无
返回值说明	**stamp**：int，指针地址。 ● 返回0表示失败。 ● 返回void类型的指针地址表示成功。
约束说明	无
注意事项	无
参考资源	接口调用示例，参见7.11 Profiling性能数据采集。

10.13.2.3 函数：set_stamp_trace_message

C函数原型	**aclError aclprofSetStampTraceMessage(void *stamp, const char *msg, uint32_t msgLen)**
Python函数	**ret = acl.prof.set_stamp_trace_message(stamp, msg, msg_len)**

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。同步接口。 在**10.13.2.2 函数：create_stamp**接口和**10.13.2.9 函数：destroy_stamp**接口之间调用。 Profiling解析并导出结果请参见《**性能分析工具使用指南**》下的“Profiling数据说明>msproftx数据说明”。
输入说明	**stamp**：int, Stamp指针地址，指代msproftx事件标记。指定**10.13.2.2 函数：create_stamp**接口的指针。 **msg**：int, 字符串内容地址。 **msg_len**：int, 字符串长度。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无
参考资源	接口调用示例，参见**7.11 Profiling性能数据采集**。

10.13.2.4 函数：mark

C函数原型	**aclError aclprofMark(void *stamp)**
Python函数	**ret = acl.prof.mark(stamp)**
函数功能	msproftx标记瞬时事件。同步接口。 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。 在**10.13.2.2 函数：create_stamp**接口和**10.13.2.9 函数：destroy_stamp**接口之间调用。
输入说明	**stamp**：int, Stamp指针地址，指代msproftx事件标记。指定**10.13.2.2 函数：create_stamp**接口的指针。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无
参考资源	接口调用示例，参见**7.11 Profiling性能数据采集**。

10.13.2.5 函数：push

C函数原型	**aclError aclprofPush(void *stamp)**
Python函数	**ret = acl.prof.push(stamp)**
函数功能	msproftx用于记录事件发生的时间跨度的开始时间。同步接口。 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。 与**10.13.2.6 函数：pop**接口成对使用，表示时间跨度的开始和结束。 不能跨线程调用，若需要跨线程可使用**10.13.2.7 函数：range_start**/**10.13.2.8 函数：range_stop**接口。 在**10.13.2.2 函数：create_stamp**接口和**10.13.2.9 函数：destroy_stamp**接口之间调用。
输入说明	**stamp**：int, Stamp指针地址，指代msproftx事件标记。指定**acl.prof.create_stamp**接口的指针。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无
参考资源	接口调用示例，参见**7.11 Profiling性能数据采集**。

10.13.2.6 函数：pop

C函数原型	**aclError aclprofPop()**
Python函数	**ret = acl.prof.pop()**
函数功能	msproftx用于记录事件发生的时间跨度的结束时间。同步接口。 调用此接口后，Profiling自动在Stamp指针中记录采集结束的时间戳。 与**10.13.2.5 函数：push**接口成对使用，表示时间跨度的开始和结束。 不能跨线程调用。若需要跨线程可使用**10.13.2.7 函数：range_start**/**10.13.2.8 函数：range_stop**接口。 在**10.13.2.2 函数：create_stamp**接口和**10.13.2.9 函数：destroy_stamp**接口之间调用。
输入说明	无

返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无
参考资源	接口调用示例，参见**7.11 Profiling性能数据采集**。

10.13.2.7 函数：range_start

C函数原型	**aclError aclprofRangeStart(void *stamp, uint32_t *rangeId)**
Python函数	**range_id, ret = acl.prof.range_start(stamp)**
函数功能	msproftx用于记录事件发生的时间跨度的开始时间。同步接口。 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。 与**10.13.2.8 函数：range_stop**接口成对使用，表示时间跨度的开始和结束。 可以跨线程调用。 在**10.13.2.2 函数：create_stamp**接口和**10.13.2.9 函数：destroy_stamp**接口之间调用。
输入说明	**stamp**：int Stamp指针，指代msproftx事件标记。指定**10.13.2.2 函数：create_stamp**接口的指针。
返回值说明	**range_id**：int msproftx事件标记的唯一标识。用于在跨线程时区分。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无
参考资源	接口调用示例，参见**7.11 Profiling性能数据采集**。

10.13.2.8 函数：range_stop

C函数原型	**aclError aclprofRangeStop(uint32_t *rangeId)**
Python函数	**ret = acl.prof.range_stop(range_id)**

函数功能	msproftx用于记录事件发生的时间跨度的结束时间。同步接口。 调用此接口后，Profiling自动在Stamp指针中记录采集结束的时间戳。 与**10.13.2.8 函数：range_stop**接口成对使用，表示时间跨度的开始和结束。 可以跨线程调用。 在**10.13.2.2 函数：create_stamp**接口和**10.13.2.9 函数：destroy_stamp**接口之间调用。
输入说明	**range_id**：int msproftx事件标记的唯一标识。用于在跨线程时区分。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无
参考资源	接口调用示例，参见**7.11 Profiling性能数据采集**。

10.13.2.9 函数：destroy_stamp

C函数原型	**void aclprofDestroyStamp(void *stamp)**
Python函数	**acl.prof.destroy_stamp(stamp)**
函数功能	释放msproftx事件标记。同步接口。 与**10.13.2.2 函数：create_stamp**接口配对使用，在**10.13.1.7 函数：stop**接口前调用。
输入说明	**stamp**：int, Stamp指针地址，指代msproftx事件标记。指定**10.13.2.2 函数：create_stamp**接口的指针。
返回值说明	无
约束说明	无
注意事项	无
参考资源	接口调用示例，参见**7.11 Profiling性能数据采集**。

10.13.3 Profiling pyACL API for Subscription（订阅算子信息的Profiling pyACL API）

10.13.3.1 功能介绍

功能

该章节描述使用订阅算子信息的Profiling pyACL 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pyACL的接口获取性能数据。

接口调用方式：**acl.prof.model_subscribe**接口、acl.prof.get*接口、**acl.prof.model_un_subscribe**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**10.13.1 Profiling pyACL API（通过Profiling pyACL API采集并落盘性能数据）**的接口交叉调用：**acl.prof.model_subscribe**接口和**acl.prof.model_un_subscribe**接口之间不能调用**acl.prof.init**接口、**acl.prof.start**接口、**acl.prof.stop**接口、**acl.prof.finalize**接口。

接口约束说明

- **接口调用要求**：
  - **acl.prof.model_subscribe**接口在模型执行之前调用，若在模型执行过程中调用**acl.prof.model_subscribe**接口，Profling采集到的数据为调用**acl.prof.model_subscribe**接口之后的数据，可能导致数据不完整。
  - **acl.prof.model_subscribe**接口需与**acl.prof.model_un_subscribe**接口配对使用，不能在调用**acl.prof.model_un_subscribe**接口前，多次调用**acl.prof.model_subscribe**接口重复订阅相同的模型。
  - 不能调用**acl.prof.model_subscribe**接口订阅不存在的模型ID。
  - 不能调用**acl.prof.model_un_subscribe**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-->**acl.prof.model_subscribe**接口-->**acl.prof.get_op_desc_size**接口-->**acl.prof.get_op_num**接口-->**acl.prof.get_op_type**/**acl.prof.get_op_name**/**acl.prof.get_op_start**/**acl.prof.get_op_end**/**acl.prof.get_op_duration**/**acl.prof.get_model_id**接口-->**acl.prof.model_un_subscribe**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-->**acl.prof.model_subscribe**接口(指定模型1)-->**acl.prof.model_subscribe**接口(指定模型1)-->**acl.prof.model_un_subscribe**接口

10.13.3.2 函数：create_subscribe_config

C函数原型	**aclprofSubscribeConfig *aclprofCreateSubscribeConfig(int8_t timeInfoSwitch, aclprofAicoreMetrics aicoreMetrics, void *fd)**

Python函数	**subscribe_config = acl.prof.create_subscribe_config(time_info_switch, aicore_metrics, fd)**
函数功能	创建aclprofSubscribeConfig类型的数据，表示创建订阅配置信息。同步接口。
输入说明	**time_info_switch**：int，是否采集网络模型中算子的性能数据： ● 1：采集 ● 0：不采集 **aicore_metrics**：int，表示AI Core性能指标采集项，参考**aclprofAicoreMetrics**。 **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**：int，用户创建管道的写文件描述符。 用户在调用**acl.prof.model_un_subscribe**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文件描述符。
返回值说明	**subscribe_config**：int。 ● 返回非0值表示成功，返回aclprofSubscribeConfig的地址对象。 ● 返回0表示失败。
约束说明	无
注意事项	无

10.13.3.3 函数：destroy_subscribe_config

C函数原型	**aclError aclprofDestroySubscribeConfig(const aclprofSubscribeConfig *profSubscribeConfig)**
Python函数	**ret = acl.prof.destroy_subscribe_config(subscribe_config)**
函数功能	销毁aclprofSubscribeConfig类型的数据，只能销毁通过**acl.prof.create_subscribe_config**接口创建的aclprofSubscribeConfig类型。同步接口。
输入说明	**subscribe_config**：int，待销毁的aclprofSubscribeConfig类型的地址对象。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无

10.13.3.4 函数：model_subscribe

C函数原型	**aclError aclprofModelSubscribe(uint32_t modelId, const aclprofSubscribeConfig *profSubscribeConfig)**
Python函数	**ret = acl.prof.model_subscribe(model_id, subscribe_config)**
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。同步接口。需要与acl.prof.model_un_subscribe接口配对使用。
输入说明	**model_id**：int，待订阅的网络模型的ID。调用acl.mdl.load_from_file接口/acl.mdl.load_from_mem接口/acl.mdl.load_from_file_with_mem接口/acl.mdl.load_from_mem_with_mem接口加载模型成功后，会返回模型ID。 **subscribe_config**：int，待订阅的配置信息。调用acl.prof.create_subscribe_config接口创建配置数据。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无
参考资源	接口调用示例，参见7.11 Profiling性能数据采集。

10.13.3.5 函数：model_un_subscribe

C函数原型	**aclError aclprofModelUnSubscribe(uint32_t modelId)**
Python函数	**ret = acl.prof.model_un_subscribe(model_id)**
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。同步接口。需要与acl.prof.model_subscribe接口配对使用。
输入说明	**model_id**：int，已订阅的网络模型的ID。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无
参考资源	接口调用示例，参见7.11 Profiling性能数据采集。

10.13.3.6 函数：get_op_desc_size

C函数原型	**aclError aclprofGetOpDescSize(size_t *opDescSize)**
Python函数	**op_desc_size, ret = acl.prof.get_op_desc_size()**
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。同步接口。 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。
输入说明	无
返回值说明	**op_desc_size**：int，算子数据结构的大小。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无
参考资源	接口调用示例，参见7.11 Profiling性能数据采集。

10.13.3.7 函数：get_op_num

C函数原型	**aclError aclprofGetOpNum(const void *opInfo, size_t opInfoLen, uint32_t *opNumber)**
Python函数	**op_num, ret = acl.prof.get_op_num(op_info, op_info_len)**
函数功能	获取指定内存中算子的数量。同步接口。 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。
输入说明	**op_info**：int，指定算子信息的内存地址。 调用acl.prof.get_op_desc_size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。 **op_info_len**：int，算子信息的长度。
返回值说明	**op_num**：int，算子的数量。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无

注意事项	无
参考资源	接口调用示例，参见7.11 Profiling性能数据采集。

10.13.3.8 函数：get_op_type_len

C函数原型	**aclError aclprofGetOpTypeLen(const void *opInfo, size_t opInfoLen, uint32_t index, size_t *opTypeLen)**
Python函数	**opTypeLen, ret = acl.prof.get_op_type_len(opInfo, opInfoLen, index)**
函数功能	获取算子类型的字符串长度，用于内存申请，同步接口。 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。
输入说明	**opInfo**：包含算子信息的地址。 **opInfoLen**：算子信息的长度。 **index**：指定获取第几个算子的算子类型名称。 用户调用acl.prof.get_op_num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	**opTypeLen**：opType的长度。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无
参考资源	接口调用示例，参见7.11 Profiling性能数据采集。

10.13.3.9 函数：get_op_name_len

C函数原型	**aclError aclprofGetOpNameLen(const void *opInfo, size_t opInfoLen, uint32_t index, size_t *opNameLen)**
Python函数	**opNameLen, ret = acl.prof.get_op_name_len(opInfo, opInfoLen, index)**
函数功能	获取算子名称的字符串长度，用于内存申请，同步接口。 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。

输入说明	**opInfo**：包含算子信息的地址。 **opInfoLen**：算子信息的长度。 **index**：指定获取第几个算子的算子名称长度。 用户调用acl.prof.get_op_num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	**opNameLen**：opName的实际内存申请长度。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	无
参考资源	接口调用示例，参见7.11 Profiling性能数据采集。

10.13.3.10 函数：get_op_type

C函数原型	**aclError aclprofGetOpType(const void *opInfo, size_t opInfoLen, uint32_t index, char *opType, size_t opTypeLen)**
Python函数	**op_type, ret = acl.prof.get_op_type(op_info, op_info_len, index, op_type_len)**
函数功能	获取指定算子的算子类型名称。同步接口。 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。
输入说明	**op_info**：int，指定算子信息的内存地址。 **op_info_len**：int，算子信息的长度。 **index**：int，指定获取第几个算子的算子类型名称。 用户调用acl.prof.get_op_num接口获取算子数量后，这个index的取值范围：[0, (算子数量-1)]。 **op_type_len**：int，op_type的长度。取值范围：[1, 65]。
返回值说明	**op_type**：str，算子类型名称。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	acl.prof.get_op_type和acl.prof.get_op_type_v2接口功能一致，但acl.prof.get_op_type_v2接口会获取算子类型长度，并分配相应的空间，不需要用户传参来指定算子类型所需空间大小，建议优先使用acl.prof.get_op_type_v2接口。
参考资源	接口调用示例，参见7.11 Profiling性能数据采集。

10.13.3.11 函数：get_op_type_v2

C函数原型	**aclError aclprofGetOpType(const void *opInfo, size_t opInfoLen, uint32_t index, char *opType, size_t opTypeLen)**
Python函数	**op_type, ret = acl.prof.get_op_type_v2(op_info, op_info_len, index)**
函数功能	获取指定算子的算子类型名称。同步接口。 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。
输入说明	**op_info**：int，指定算子信息的内存地址。 **op_info_len**：int，算子信息的长度。 **index**：int，指定获取第几个算子的算子类型名称。 用户调用acl.prof.get_op_num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	**op_type**：str，算子类型名称。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	acl.prof.get_op_type_v2和acl.prof.get_op_type接口功能一致，但acl.prof.get_op_type_v2接口会获取算子类型长度，并分配相应的空间，不需要用户传参来指定算子类型所需空间大小，建议优先使用acl.prof.get_op_type_v2接口。
参考资源	接口调用示例，参见7.11 Profiling性能数据采集。

10.13.3.12 函数：get_op_name

C函数原型	**aclError aclprofGetOpName(const void *opInfo, size_t opInfoLen, uint32_t index, char *opName, size_t opNameLen)**
Python函数	**op_name, ret = acl.prof.get_op_name(op_info, op_info_len, index, op_name_len)**
函数功能	获取指定算子的算子名称。同步接口。 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。

输入说明	**op_info**：int，指定算子信息的内存地址。 **op_info_len**：int，算子信息的长度。 **index**：int，指定获取第几个算子的算子名称。 用户调用acl.prof.get_op_num接口获取算子数量后，这个index的取值范围：[0, (算子数量-1)]。 **op_name_len**：int，op_name的长度。取值范围：[1, 257]。
返回值说明	**op_name**：str，算子名称。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无
注意事项	acl.prof.get_op_name和acl.prof.get_op_name_v2接口功能一致，但acl.prof.get_op_name_v2接口会获取算子名称长度，并分配相应的空间，不需要用户传参来指定算子名称所需空间大小，建议优先使用acl.prof.get_op_name_v2接口。
参考资源	接口调用示例，参见7.11 Profiling性能数据采集。

10.13.3.13 函数：get_op_name_v2

C函数原型	**aclError aclprofGetOpName(const void *opInfo, size_t opInfoLen, uint32_t index, char *opName, size_t opNameLen)**
Python函数	**op_name, ret = acl.prof.get_op_name_v2(op_info, op_info_len, index)**
函数功能	获取指定算子的算子名称。同步接口。 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。
输入说明	**op_info**：int，指定算子信息的内存地址。 **op_info_len**：int，算子信息的长度。 **index**：int，指定获取第几个算子的算子名称。 用户调用acl.prof.get_op_num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	**op_name**：str，算子名称。 **ret**：int，错误码。 ● 返回0表示成功。 ● 返回其它值表示失败。
约束说明	无

注意事项	acl.prof.get_op_name_v2和acl.prof.get_op_name接口功能一致，但acl.prof.get_op_name_v2接口会获取算子名称长度，并分配相应的空间，不需要用户传参来指定算子名称所需空间大小，建议优先使用acl.prof.get_op_name_v2接口。
参考资源	接口调用示例，参见7.11 Profiling性能数据采集。

10.13.3.14 函数：get_op_start

C函数原型	**uint64_t aclprofGetOpStart(const void *opInfo, size_t opInfoLen, uint32_t index)**
Python函数	**op_start = acl.prof.get_op_start(op_info, op_info_len, index)**
函数功能	获取算子执行的开始时间，单位为ns。同步接口。 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。
输入说明	**op_info**：int，指定算子信息的内存地址。 **op_info_len**：int，算子信息的长度。 **index**：int，指定获取第几个算子的算子名称。 用户调用acl.prof.get_op_num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	**op_start**：int，算子执行的开始时间。
约束说明	无
注意事项	无
参考资源	接口调用示例，参见7.11 Profiling性能数据采集。

10.13.3.15 函数：get_op_end

C函数原型	**uint64_t aclprofGetOpEnd(const void *opInfo, size_t opInfoLen, uint32_t index)**
Python函数	**op_end = acl.prof.get_op_end(op_info, op_info_len, index)**
函数功能	获取算子执行的结束时间，单位为ns。同步接口。 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。

输入说明	**op_info**：int，指定算子信息的内存地址。 **op_info_len**：int，算子信息的长度。 **index**：int，指定获取第几个算子的算子名称。 用户调用acl.prof.get_op_num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	**op_end**：int，算子执行的结束时间。
约束说明	无
注意事项	无
参考资源	接口调用示例，参见7.11 Profiling性能数据采集。

10.13.3.16 函数：get_op_duration

C函数原型	**uint64_t aclprofGetOpDuration(const void *opInfo, size_t opInfoLen, uint32_t index)**
Python函数	**op_duration = acl.prof.get_op_duration(op_info, op_info_len, index)**
函数功能	获取算子执行的耗时时间，单位为ns。同步接口。 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。
输入说明	**op_info**：int，指定算子信息的内存地址。 **op_info_len**：int，算子信息的长度。 **index**：int，指定获取第几个算子的算子名称。 用户调用acl.prof.get_op_num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	**op_duration**：int，算子执行的耗时时间。
约束说明	无
注意事项	无
参考资源	接口调用示例，参见7.11 Profiling性能数据采集。

10.13.3.17 函数：get_model_id

C函数原型	**size_t aclprofGetModelId(const void *opInfo, size_t opInfoLen, uint32_t index)**
Python函数	**model_id = acl.prof.get_model_id(op_info, op_info_len, index)**

函数功能	获取指定算子所在模型的ID。同步接口。 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。
输入说明	**op_info**：int，指定算子信息的内存地址。 **op_info_len**：int，算子信息的长度。 **index**：int，指定获取第几个算子的算子名称。 用户调用**acl.prof.get_op_num**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	**model_id**：int，模型的ID。
约束说明	无
注意事项	无
参考资源	接口调用示例，参见**7.11 Profiling性能数据采集**。

10.13.4 Profiling pyACL API（PyTorch 场景标记迭代时间）

10.13.4.1 函数：get_step_timestamp

C函数原型	**aclError aclprofGetStepTimestamp(aclprofStepInfo* stepInfo, aclprofStepTag tag, aclrtStream stream)**
Python函数	**ret =acl.prof.get_step_timestamp(stepinfo, tag, stream)**
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。
输入说明	**stepinfo**：指定迭代信息。需提前调用**acl.prof.create_step_info**接口创建aclprofStepInfo类型的数据。 **tag**：用于标记迭代开始或结束。在迭代开始时传入ACL_STEP_START，迭代结束时需传入ACL_STEP_END。 **stream**：指定Stream。
返回值说明	**ret**：int，错误码。 ● 返回0表示成功。 ● 返回**其它值**表示失败。
约束说明	无
注意事项	无

10.13.4.2 函数：create_step_info

C函数原型	**aclprofStepInfo* aclprofCreateStepInfo()**
Python函数	**info = acl.prof.create_step_info()**
函数功能	创建aclprofStepInfo对象，用于描述迭代信息，同步接口。
输入说明	无
返回值说明	**info**： ● 返回aclprofStepInfo类型，表示成功。 ● 返回None，表示失败。
约束说明	同一个**aclprofStepInfo**对象、同一个tag只能设置一次，否则**Profiling**解析会出错。
注意事项	无

10.13.4.3 函数：destroy_step_info

C函数原型	**void aclprofDestroyStepInfo (aclprofStepInfo * stepinfo)**
Python函数	**acl.prof.destroy_step_info(stepinfo)**
函数功能	销毁通过**acl.prof.create_step_info**接口创建的aclprofStepInfo类型的数据，同步接口。
输入说明	**stepinfo**：指定迭代信息，需提前调用**acl.prof.create_step_info**接口创建aclprofStepInfo类型的数据。
返回值说明	无
约束说明	无
注意事项	无

10.14 数据类型及其操作接口

10.14.1 aclError

表 10-30 返回码列表

返回码	说明	可能原因及解决方法
ACL_SUCCESS = 0	执行成功。	-

返回码	说明	可能原因及解决方法
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。	-
ACL_ERROR_INVALID_PARAM = 100000	参数校验失败。	请检查接口的入参值是否正确。
ACL_ERROR_UNINITIALIZE = 100001	ACL未初始化。	● 请检查是否已调用acl.init接口进行初始化，请确保已调用acl.init接口，且在其它pyACL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acl.mdl.init_dump接口。
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。
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。
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。
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。
ACL_ERROR_PARSE_FILE = 100006	解析文件失败。	请检查文件内容是否合法。
ACL_ERROR_FILE_MISSING_ATTR = 100007	文件缺失参数。	请检查文件内容是否完整。
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。
ACL_ERROR_INVALID_DUMP_CONFIG = 100009	无效的Dump配置。	请检查acl.init接口的配置文件中的Dump配置是否正确，详细配置请参见《精度比对工具使用指南》中的“比对数据准备>准备离线模型dump数据文件”。
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。

版权所有 © 华为技术有限公司

返回码	说明	可能原因及解决方法
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，请重新转换模型。
ACL_ERROR_PARSE_MODEL = 100013	解析模型失败。	模型可能与当前版本不匹配，请重新转换模型。
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。
ACL_ERROR_MODEL_SIZE_INVALID = 100015	无效的模型大小。	模型文件无效，请重新转换模型。
ACL_ERROR_MODEL_MISSING_ATTR = 100016	模型缺少参数。	模型可能与当前版本不匹配，请重新转换模型。
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。
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。
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，请重新转换模型。
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。
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。
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。
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。
ACL_ERROR_OP_NOT_FOUND = 100024	单算子未找到。	请检查算子类型是否支持。
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，请重新转换模型。
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。
ACL_ERROR_FORMAT_NOT_MATCH = 100027	Format不匹配。	请检查Format是否正确。

返回码	说明	可能原因及解决方法
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acl.op.register_compile_func接口注册算子选择器。
ACL_ERROR_KERNEL_NOT_FOUND = 100029	编译算子时，算子Kernel未注册。	请检查是否调用acl.op.create_kernel接口注册算子Kernel。
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acl.op.register_compile_func接口注册算子选择器。
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acl.op.create_kernel接口注册算子Kernel。
ACL_ERROR_INVALID_QUEUE_ID = 100032	无效的队列ID。	请检查队列ID是否正确。
ACL_ERROR_REPEAT_SUBSCRIBE = 100033	重复订阅。	请检查针对同一个Stream，是否重复调用**acl.rt.subscribe_report**接口。
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**acl.rt.subscribe_report**接口。
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**acl.rt.subscribe_report**接口。
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acl.rt.launch_callback接口下发callback任务。 请检查acl.rt.process_report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acl.rt.process_report接口，则需优化代码逻辑。

版权所有 © 华为技术有限公司

返回码	说明	可能原因及解决方法
ACL_ERROR_REPEAT_FINALIZE = 100037	重复去初始化。	请检查是否重复调用acl.finalize接口进行去初始化。
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acl.mdl.get_first_aipp_info接口时，请传入正确的index值。
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。
ACL_ERROR_DUMP_ALREADY_RUN = 100044	已存在获取Dump数据的任务。	请检查在调用**acl.mdl.init_dump**接口、**acl.mdl.set_dump**接口、**acl.mdl.finalize_dump**接口配置Dump信息前，是否已调用**acl.init**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。
ACL_ERROR_DUMP_NOT_RUN = 100045	未使用**acl.mdl.init_dump**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**acl.mdl.init_dump**接口处的说明。
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**acl.init**处的配置说明及示例。
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。
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。

版权所有 © 华为技术有限公司

返回码	说明	可能原因及解决方法
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。
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。
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。
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。
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。
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。
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。
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。
ACL_ERROR_FEATURE_UNSUPPORTED = 200006	特性不支持。	请根据日志报错排查问题，或联系华为工程师。 日志的详细介绍，请参见《**日志参考**》。
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。

返回码	说明	可能原因及解决方法
ACL_ERROR_INTERNAL_ERROR = 500000	未知内部错误。	请根据日志报错排查问题，或联系华为工程师。 日志的详细介绍，请参见《日志参考》。
ACL_ERROR_FAILURE = 500001	系统内部ACL的错误。	请根据日志报错排查问题，或联系华为工程师。 日志的详细介绍，请参见《日志参考》。
ACL_ERROR_GE_FAILURE = 500002	系统内部GE的错误。	请根据日志报错排查问题，或联系华为工程师。 日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_FAILURE = 500003	系统内部RUNTIME的错误。	请根据日志报错排查问题，或联系华为工程师。 日志的详细介绍，请参见《日志参考》。
ACL_ERROR_DRV_FAILURE = 500004	系统内部DRV（Driver）的错误。	请根据日志报错排查问题，或联系华为工程师。 日志的详细介绍，请参见《日志参考》。

表 10-31 内部 RUNTIME 的返回码列表

返回码	含义	可能原因及解决方法
ACL_RT_SUCCESS = 0	执行成功。	-
ACL_ERROR_RT_PARAM_INVALID = 107000	参数校验失败。	请检查接口入参是否正确。
ACL_ERROR_RT_INVALID_DEVICEID = 107001	无效的Device ID。	请检查Device ID是否合法。
ACL_ERROR_RT_CONTEXT_NULL = 107002	context为空。	请检查是否调用**acl.rt.set_context**或**acl.rt.set_device**。
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。
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。
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。

返回码	含义	可能原因及解决方法
ACL_ERROR_RT_EVENT_TIMESTAMP_INVALID = 107006	event时间戳无效。	请检查Event是否创建。
ACL_ERROR_RT_EVENT_TIMESTAMP_REVERSAL = 107007	event时间戳反转。	请检查Event是否创建。
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**10.5.5 内存管理**。
ACL_ERROR_RT_FILE_OPEN = 107009	打开文件失败。	请检查文件是否存在。
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。
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。
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。
ACL_ERROR_RT_GROUP_NOT_SET = 107013	未设置Group。	-
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)]。
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**acl.rt.subscribe_report**接口。
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。
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.
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。
ACL_ERROR_RT_WAIT_TIMEOUT = 107019;	执行任务超时。	请尝试重新执行下发任务的接口。

返回码	含义	可能原因及解决方法
ACL_ERROR_RT_FEATURE_NOT_SUPPORT = 207000	特性不支持。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
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。
ACL_ERROR_RT_MEMORY_FREE = 207002	内存释放失败。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
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。
ACL_ERROR_RT_NO_DEVICE = 207004	Device不可用。	请检查Device是否正常运行。
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。
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。
ACL_ERROR_RT_NO_EVENT_RESOURCE = 207007	Event资源不足。	请参考acl.rt.create_event接口处的说明检查Event数量是否符合要求。
ACL_ERROR_RT_NO_STREAM_RESOURCE = 207008	Stream资源不足。	请参考acl.rt.create_stream接口处的说明检查Stream数量是否符合要求。
ACL_ERROR_RT_NO_NOTIFY_RESOURCE = 207009	系统内部Notify资源不足。	数据预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。
ACL_ERROR_RT_NO_MODEL_RESOURCE = 207010	模型资源不足。	建议卸载部分模型。
ACL_ERROR_RT_NO_CDQ_RESOURCE = 207011;	Runtime内部资源不足。	请根据日志报错排查问题，或联系工程师。日志的详细介绍，请参见《日志参考》。
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。

版权所有 © 华为技术有限公司

返回码	含义	可能原因及解决方法
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。
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。
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。
ACL_ERROR_RT_INTERNEL_ERROR = 507000	Host上的runtime模块内部错误。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_TS_ERROR = 507001	Device上的task scheduler模块内部错误。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_STREAM_TASK_FULL = 507002	stream上的task数量满。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_STREAM_TASK_EMPTY = 507003	stream上的task数量为空。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_STREAM_NOT_COMPLETE = 507004	stream上的task未全部执行完成。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_END_OF_SEQUENCE = 507005	AI CPU上的task执行完成。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_EVENT_NOT_COMPLETE = 507006	event未完成。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。

返回码	含义	可能原因及解决方法
ACL_ERROR_RT_CONTEXT_RELEASE_ERROR = 507007	context释放失败。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_SOC_VERSION = 507008	获取soc version失败。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_TASK_TYPE_NOT_SUPPORT = 507009	不支持的task类型。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_LOST_HEARTBEAT = 507010	task scheduler丢失心跳。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_MODEL_EXECUTE = 507011	模型执行失败。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_REPORT_TIMEOUT = 507012	获取task scheduler的消息失败。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_SYS_DMA = 507013	system dma（Direct Memory Access）硬件执行错误。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_AICORE_TIMEOUT = 507014	aicore执行超时。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_AICORE_EXCEPTION = 507015	aicore执行异常。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_AICORE_TRAP_EXCEPTION = 507016	aicore trap执行异常。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。

版权所有 © 华为技术有限公司

返回码	含义	可能原因及解决方法
ACL_ERROR_RT_AICPU_TIMEOUT = 507017	aicpu执行超时。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_AICPU_EXCEPTION = 507018	aicpu执行异常。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_AICPU_DATADUMP_RSP_ERR = 507019	aicpu执行数据dump后未给task scheduler返回响应。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_AICPU_MODEL_RSP_ERR = 507020	aicpu执行模型后未给task scheduler返回响应。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_IPC_ERROR = 507022	进程间通信异常。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_MODEL_ABORT_NORMAL = 507023	模型退出。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_KERNEL_UNREGISTERING = 507024	算子正在去注册。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_RINGBUFFER_NOT_INIT = 507025	ringbuffer（环形缓冲区）功能未初始化。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_RINGBUFFER_NO_DATA = 507026	ringbuffer（环形缓冲区）没有数据。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_KERNEL_LOOKUP = 507027	RUNTIME内部的kernel未注册。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。

返回码	含义	可能原因及解决方法
ACL_ERROR_RT_KERNEL_DUPLICATE = 507028	重复注册RUNTIME内部的kernel。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_DEBUG_REGISTER_FAIL = 507029	debug功能注册失败。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_DEBUG_UNREGISTER_FAIL = 507030	debug功能去注册失败。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_LABEL_CONTEXT = 507031	标签不在当前Context内。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_PROGRAM_USE_OUT = 507032	注册的program数量超过限制。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_DEV_SETUP_ERROR = 507033	Device启动失败。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_VECTOR_CORE_TIMEOUT = 507034;	vector core执行超时。	请根据日志报错排查问题，或联系工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_VECTOR_CORE_EXCEPTION = 507035;	vector core执行异常。	请根据日志报错排查问题，或联系工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_VECTOR_CORE_TRAP_EXCEPTION = 507036;	vector core trap执行异常。	请根据日志报错排查问题，或联系工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_CDQ_BATCH_ABNORMAL = 507037;	Runtime内部资源申请异常。	请根据日志报错排查问题，或联系工程师。日志的详细介绍，请参见《日志参考》。

版权所有 © 华为技术有限公司

返回码	含义	可能原因及解决方法
ACL_ERROR_RT_DIE_MODE_CHANGE_ERROR = 507038;	die模式修改异常，不能修改die模式。	请根据日志报错排查问题，或联系工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_DIE_SET_ERROR = 507039;	单die模式不能指定die。	请根据日志报错排查问题，或联系工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_INVALID_DIEID = 507040;	指定die id错误。	请根据日志报错排查问题，或联系工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_DIE_MODE_NOT_SET = 507041;	die模式没有设置。	请根据日志报错排查问题，或联系工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_AICORE_TRAP_READ_OVERFLOW = 507042;	aicore trap读越界异常。	请根据日志报错排查问题，或联系工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_AICORE_TRAP_WRITE_OVERFLOW = 507043;	aicore trap写越界异常。	请根据日志报错排查问题，或联系工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_VECTOR_CORE_TRAP_READ_OVERFLOW = 507044;	vector core trap读越界异常。	请根据日志报错排查问题，或联系工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_VECTOR_CORE_TRAP_WRITE_OVERFLOW = 507045;	vector core trap写越界异常。	请根据日志报错排查问题，或联系工程师。日志的详细介绍，请参见《日志参考》。
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。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
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。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。

版权所有 © 华为技术有限公司

返回码	含义	可能原因及解决方法
ACL_ERROR_RT_FFTS_PLUS_TIMEOUT = 507048;	内部任务执行超时。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_FFTS_PLUS_EXCEPTION = 507049;	内部任务执行异常。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_FFTS_PLUS_TRAP_EXCEPTION = 507050;	内部任务trap异常。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
static const int32_t ACL_ERROR_RT_SEND_MSG = 507051;	数据入队过程中消息发送失败 。	请根据日志报错排查问题，或联系工程师。 日志的详细介绍，请参见《日志参考》。
static const int32_t ACL_ERROR_RT_COPY_DATA = 507052;	数据入队过程中内存拷贝失败。	请根据日志报错排查问题，或联系工程师。 日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_DRV_INTERNAL_ERROR = 507899	Driver模块内部错误。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_AICPU_INTERNAL_ERROR = 507900;	AI CPU模块内部错误。	请根据日志报错排查问题，或联系工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_RT_SOCKET_CLOSE = 507901;	内部HDC（Host Device Communication）会话链接断开。	请根据日志报错排查问题，或联系工程师。日志的详细介绍，请参见《日志参考》。

表 10-32 内部 GE 的返回码列表

返回码	含义	可能原因及解决方法
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
ACL_ERROR_GE_EXEC_NOT_INIT = 145001	未初始化。	• 请检查是否已调用**acl.init**接口进行初始化，请确保已调用**acl.init**接口，且在其它pyACL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**acl.mdl.init_dump**接口、初始化Profiling的**acl.prof.init**接口。
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。
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。
ACL_ERROR_GE_EXEC_MODEL_DATA_SIZE_INVALID = 145006	无效的模型大小。	无效的模型大小。
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。
ACL_ERROR_GE_EXEC_MODEL_QUEUE_ID_INVALID = 145008	无效的队列ID。	无效的队列 ID。
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。
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。
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。
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。
ACL_ERROR_GE_AIPP_BATCH_EMPTY = 145014	无效的AIPP Batch数。	请检查AIPP Batch数是否正确。

返回码	含义	可能原因及解决方法
ACL_ERROR_GE_AIPP_NOT_EXIST = 145015	AIPP配置不存在。	请检查AIPP是否配置。
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。
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。
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。
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。
ACL_ERROR_GE_FORMAT_INVALID = 145020	无效的format。	请检查Tensor数据的format是否有效。
ACL_ERROR_GE_SHAPE_INVALID = 145021	无效的shape。	请检查Tensor数据的shape是否有效。
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。
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。
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。
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。
ACL_ERROR_GE_INTERNAL_ERROR = 545000	未知内部错误。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_GE_LOAD_MODEL = 545001	系统内部加载模型失败。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。

返回码	含义	可能原因及解决方法
ACL_ERROR_GE_EXEC_LOAD_MODEL_PARTITION_FAILED = 545002	系统内部加载模型失败。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_GE_EXEC_LOAD_WEIGHT_PARTITION_FAILED = 545003	系统内部加载模型权值失败。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_GE_EXEC_LOAD_TASK_PARTITION_FAILED = 545004	系统内部加载模型任务失败。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_GE_EXEC_LOAD_KERNEL_PARTITION_FAILED = 545005	系统内部加载模型算子失败。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_GE_EXEC_RELEASE_MODEL_DATA = 545006	系统内释放模型空间失败。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_GE_COMMAND_HANDLE = 545007	系统内命令操作失败。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_GE_GET_TENSOR_INFO = 545008	系统内获取张量数据失败。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。
ACL_ERROR_GE_UNLOAD_MODEL = 545009	系统内卸载模型空间失败。	请根据日志报错排查问题，或联系华为工程师。日志的详细介绍，请参见《日志参考》。

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，需要将问题反馈给华为的，此时返回码定义为：5XXXXX。
- 规则5：无法识别的错误，当前都映射为500000。

10.14.2 aclDataType

数据类型	说明
ACL_DT_UNDEFINED = -1	未知数据类型，默认值
ACL_FLOAT16 = 1	float16
ACL_INT8 = 2	int8
ACL_INT32 = 3	int32
ACL_UINT8 = 4	uint8
ACL_INT16 = 6	int16
ACL_UINT16 = 7	uint16
ACL_UINT32 = 8	uint32
ACL_INT64 = 9	int64
ACL_UINT64 = 10	uint64
ACL_DOUBLE = 11	double
ACL_BOOL = 12	bool
ACL_STRING = 13	string
ACL_COMPLEX64 = 16	complex64
ACL_COMPLEX128 = 17	complex128
ACL_BF16 = 27	bf16

10.14.3 aclFloat16

数据格式	说明
aclFloat16	如果需要传递aclFloat16类型数据，即指传递uint16_t类型数据。

说明

该数据类型仅用于数据传递，不能用于加减乘除等运算。

10.14.4 aclFormat

数据格式	说明
ACL_FORMAT_UNDEFINED = -1	未知数据类型，默认值。
ACL_FORMAT_NCHW = 0	NCHW格式。
ACL_FORMAT_NHWC = 1	NHWC格式。
ACL_FORMAT_ND = 2	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。
ACL_FORMAT_NC1HWC0 = 3	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。
ACL_FORMAT_FRACTAL_Z = 4	卷积的权重的格式。
ACL_FORMAT_NC1HWC0_C04 = 12	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。
ACL_FORMAT_HWCN = 16	HWCN格式。
ACL_FORMAT_NDHWC = 27	NDHWC格式。对于3维图像就需要使用带D（Depth）维度的格式。
ACL_FORMAT_FRACTAL_NZ = 29	内部格式，用户目前无需使用。
ACL_FORMAT_NCDHW = 30	NCDHW格式。对于3维图像就需要使用带D（Depth）维度的格式。
ACL_FORMAT_NDC1HWC0 = 32	6维数据格式。相比于NC1HWC0，仅多了D（Depth）维度。
ACL_FRACTAL_Z_3D = 33	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。

10.14.5 acldvppPixelFormat

像素格式
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 NV12 8bit
PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4 # YUV422SP NV21 8bit
PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5 # YUV444SP NV12 8bit
PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6 # YUV444SP NV21 8bit
PIXEL_FORMAT_YUYV_PACKED_422 = 7 # YUV422P YUYV 8bit
PIXEL_FORMAT_UYVY_PACKED_422 = 8 # YUV422P UYVY 8bit
PIXEL_FORMAT_YVYU_PACKED_422 = 9 # YUV422P YVYU 8bit
PIXEL_FORMAT_VYUY_PACKED_422 = 10 # YUV422P VYUY 8bit
PIXEL_FORMAT_YUV_PACKED_444 = 11 # YUV444P 8bit
PIXEL_FORMAT_RGB_888 = 12 # RGB888
PIXEL_FORMAT_BGR_888 = 13 # BGR888
PIXEL_FORMAT_ARGB_8888 = 14 # ARGB8888
PIXEL_FORMAT_ABGR_8888 = 15 # ABGR8888
PIXEL_FORMAT_RGBA_8888 = 16 # RGBA8888
PIXEL_FORMAT_BGRA_8888 = 17 # BGRA8888
PIXEL_FORMAT_YUV_SEMI_PLANNER_420_10BIT = 18 # YUV420SP 10bit
PIXEL_FORMAT_YVU_SEMI_PLANNER_420_10BIT = 19 # YVU420sp 10bit
PIXEL_FORMAT_YVU_PLANAR_420 = 20 # YUV420P 8bit
PIXEL_FORMAT_YVU_PLANAR_422 = 21 # YUV422P 8bit
PIXEL_FORMAT_YVU_PLANAR_444 = 22 # YUV444P 8bit
PIXEL_FORMAT_RGB_444 = 23
PIXEL_FORMAT_BGR_444 = 24
PIXEL_FORMAT_ARGB_4444 = 25
PIXEL_FORMAT_ABGR_4444 = 26
PIXEL_FORMAT_RGBA_4444 = 27
PIXEL_FORMAT_BGRA_4444 = 28

像素格式
PIXEL_FORMAT_RGB_555 = 29
PIXEL_FORMAT_BGR_555 = 30
PIXEL_FORMAT_RGB_565 = 31
PIXEL_FORMAT_BGR_565= 32
PIXEL_FORMAT_ARGB_1555 = 33
PIXEL_FORMAT_ABGR_1555 = 34
PIXEL_FORMAT_RGBA_1555 = 35
PIXEL_FORMAT_BGRA_1555 = 36
PIXEL_FORMAT_ARGB_8565 = 37
PIXEL_FORMAT_ABGR_8565 = 38
PIXEL_FORMAT_RGBA_8565 = 39
PIXEL_FORMAT_BGRA_8565 = 40
PIXEL_FORMAT_RGB_BAYER_8BPP = 50
PIXEL_FORMAT_RGB_BAYER_10BPP = 51
PIXEL_FORMAT_RGB_BAYER_12BPP = 52
PIXEL_FORMAT_RGB_BAYER_14BPP = 53
PIXEL_FORMAT_RGB_BAYER_16BPP = 54
PIXEL_FORMAT_BGR_888_PLANAR = 70
PIXEL_FORMAT_HSV_888_PACKAGE = 71
PIXEL_FORMAT_HSV_888_PLANAR = 72
PIXEL_FORMAT_LAB_888_PACKAGE = 73
PIXEL_FORMAT_LAB_888_PLANAR = 74
PIXEL_FORMAT_S8C1 = 75
PIXEL_FORMAT_S8C2_PACKAGE = 76
PIXEL_FORMAT_S8C2_PLANAR = 77
PIXEL_FORMAT_S16C1 = 78
PIXEL_FORMAT_U8C1 = 79
PIXEL_FORMAT_U16C1 = 80
PIXEL_FORMAT_S32C1 = 81
PIXEL_FORMAT_U32C1 = 82

像素格式
PIXEL_FORMAT_U64C1 = 83
PIXEL_FORMAT_S64C1 = 84
PIXEL_FORMAT_YUV_SEMIPLANAR_440 = 1000
PIXEL_FORMAT_YVU_SEMIPLANAR_440 = 1001
PIXEL_FORMAT_FLOAT32 = 1002
PIXEL_FORMAT_BUTT = 1003
PIXEL_FORMAT_UNKNOWN = 10000

10.14.6 acldvppStreamFormat

视频流格式
H265_MAIN_LEVEL = 0
H264_BASELINE_LEVEL = 1
H264_MAIN_LEVEL = 2
H264_HIGH_LEVEL = 3

10.14.7 acldvppJpegFormat

视频流格式
ACL_JPEG_CSS_444 = 0
ACL_JPEG_CSS_422 = 1
ACL_JPEG_CSS_420 = 2
ACL_JPEG_CSS_GRAY = 3
ACL_JPEG_CSS_440 = 4
ACL_JPEG_CSS_411 = 5
ACL_JPEG_CSS_UNKNOWN = 1000

10.14.8 acldvppCscMatrix

数据格式	说明
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 = 1	基于BT601 narrow标准的色域转换矩阵。
ACL_DVPP_CSC_MATRIX_BT709_WIDE = 2	基于BT709 wide标准的色域转换矩阵。
ACL_DVPP_CSC_MATRIX_BT709_NARROW = 3	基于BT709 narrow标准的色域转换矩阵。
ACL_DVPP_CSC_MATRIX_BT2020_WIDE = 4	基于BT2020 wide标准的色域转换矩阵。
ACL_DVPP_CSC_MATRIX_BT2020_NARROW = 5	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 |

成员名称	描述
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 |
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 |

10.14.9 aclrtEventStatus

数据格式	说明
ACL_EVENT_STATUS_COMPLETE = 0	完成
ACL_EVENT_STATUS_NOT_READY = 1	未完成
ACL_EVENT_STATUS_RESERVED = 2	预留

10.14.10 aclrtRunMode

数据格式	说明
ACL_DEVICE = 0	昇腾AI软件栈运行在Device的Control CPU或板端环境上
ACL_HOST = 1	昇腾AI软件栈运行在Host侧

10.14.11 aclTransType

转置方式	说明
ACL_TRANS_N = 0	不转置，默认为不转置。
ACL_TRANS_T = 1	转置。
ACL_TRANS_NZ = 2	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 = 3	内部格式转置，当前不支持。

10.14.12 aclComputeType

计算精度类型	说明
ACL_COMPUTE_HIGH_PRECISION = 0	高精度
ACL_COMPUTE_LOW_PRECISION = 1	当前不支持

10.14.13 aclfvSearchType

Atlas 200/500 A2推理产品，不支持该枚举值。

特征向量搜索方式	说明
SEARCH_1_N = 0	1:N operation type
SEARCH_N_M = 1	N:M operation type

10.14.14 aclAippInputFormat

Aipp输入图像格式	说明
ACL_YUV420SP_U8 = 1	静态AIPP和动态AIPP都支持。
ACL_XRGB8888_U8 = 2	静态AIPP和动态AIPP都支持
ACL_RGB888_U8 = 3	静态AIPP和动态AIPP都支持。
ACL_YUV400_U8 = 4	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	暂不支持

10.14.15 aclAippInfo

静态Aipp配置信息	含义
inputFormat	AIPP输入图像格式(aclAippInputFormat类型数据)。
srcImageSizeW	原始图片的宽，对于YUV420SP_U8类型的图像，要求取值是偶数。 取值范围： [1,4096]

静态Aipp配置信息	含义
srcImageSizeH	原始图片的高，对于YUV420SP_U8类型的图像，要求取值是偶数。 取值范围： [1,4096]
cropSwitch	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。
loadStartPosW	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]
loadStartPosH	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	抠图区域的宽度。 对于YUV420SP_U8格式的图像，参数取值要求是偶数。 取值范围：[1,4096]
cropSizeH	抠图区域的高度。 对于YUV420SP_U8格式的图像，参数取值要求是偶数。 取值范围：[1,4096]
resizeSwitch	AIPP处理图片时是否支持缩放，保留字段，暂不支持该功能。
resizeOutputW	缩放后图像的宽度。
resizeOutputH	缩放后图像的高度。
paddingSwitch	AIPP处理图片时padding使能开关。
leftPaddingSize	W左填充。
rightPaddingSize	W右填充。
topPaddingSize	H上填充。
bottomPaddingSize	H下填充。
cscSwitch	色域转换开关，静态AIPP配置。
rbuvSwapSwitch	色域转换前，R通道与B通道交换开关/U通道与V通道交换开关。
axSwapSwitch	色域转换前，RGBA->ARGB, YUVA->AYUV交换开关。
singleLineMode	单行处理模式（只处理抠图后的第一行）开关。

静态Aipp配置信息	含义
matrixR0C0	3*3 CSC矩阵元素。
matrixR0C1	3*3 CSC矩阵元素。
matrixR0C2	3*3 CSC矩阵元素。
matrixR1C0	3*3 CSC矩阵元素。
matrixR1C1	3*3 CSC矩阵元素。
matrixR1C2	3*3 CSC矩阵元素。
matrixR2C0	3*3 CSC矩阵元素。
matrixR2C1	3*3 CSC矩阵元素。
matrixR2C2	3*3 CSC矩阵元素。
outputBias0	RGB转YUV时的输出偏移。
outputBias1	RGB转YUV时的输出偏移。
outputBias2	RGB转YUV时的输出偏移。
inputBias0	YUV转RGB时的输入偏移。
inputBias1	YUV转RGB时的输入偏移。
inputBias2	YUV转RGB时的输入偏移。
meanChn0	每个通道的均值。
meanChn1	每个通道的均值。
meanChn2	每个通道的均值。
meanChn3	每个通道的均值。
minChn0	每个通道的最小值。
minChn1	每个通道的最小值。
minChn2	每个通道的最小值。
minChn3	每个通道的最小值。
varReciChn0	每个通道的方差。
varReciChn1	每个通道的方差。
varReciChn2	每个通道的方差。
varReciChn3	每个通道的方差。
srcFormat	模型转换前，原始模型的输入format(aclFormat类型数据)。
srcDatatype	模型转换前，原始模型的输入datatype（aclDataType类型数据）。

静态Aipp配置信息	含义
srcDimNum	模型转换前，原始模型输入dims。
shapeCount	动态Shape（动态Batch或动态分辨率）场景下的档位数。
outDims	输出数据维度(aclAippDims类型)。
aippExtend	预留参数，当前版本用户必须传入空指针。

10.14.16 aclAippDims

aipp维度信息	含义
srcDims	模型输入输出维度信息(aclmdlIODims类型)数据，模型转换前的dims，如[1,3,150,150]
srcSize	模型转换前输入数据的大小
aippOutdims	内部数据格式的dims信息
aippOutSize	经过AIPP处理后的输出数据的大小

10.14.17 aclmdlIODims

模型输入输出维度信息	含义
name	tensor name
dimCount	Shape中的维度个数
dims	维度信息

10.14.18 aclrtGroupAttr

Atlas 200/500 A2推理产品，不支持该枚举值。

数据格式	说明
ACL_GROUP_AICORE_INT= 0	指定Group对应的aicore个数，属性值的数据类型为整型。
ACL_GROUP_AIV_INT = 1	指定Group对应的vector core个数，属性值的数据类型为整型。
ACL_GROUP_AIC_INT = 2	指定Group对应的aicpu线程数，属性值的数据类型为整型。

数据格式	说明
ACL_GROUP_SDMANUM_INT = 3	内存异步拷贝的通道数，属性值的数据类型为整型。
ACL_GROUP_SDMANUM_INT = 4	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。
ACL_GROUP_GROUPID_INT = 5	指定Group的ID。属性值的数据类型为整型。

10.14.19 aclprofAicoreMetrics

AI Core性能指标采集项	含义
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。
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、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_vector、ub_write_bw_vector、ub_read_bw_scalar、ub_write_bw_scalar。
ACL_AICORE_L2_CACHE = 6	读写cache命中次数和缺失后重新分配次数， 包括采集项write_cache_hit、write_cache_miss_allocate、r*_read_cache_hit、r*_read_cache_miss_allocate。

AI Core性能指标采集项	含义
ACL_AICORE_PIPE_EXECUTE_UTILIZATION = 7	计算单元和搬运单元耗时占比，包括采集项vec_exe_ratio、mac_exe_ratio、scalar_exe_ratio、mte1_exe_ratio、mte2_exe_ratio、mte3_exe_ratio、fixpipe_exe_ratio。
ACL_AICORE_NONE = 0xFF	表示不采集数据。

10.14.20 acldvppBorderType

数据格式	说明
BORDER_CONSTANT= 0	添加有颜色的常数值边界
BORDER_REPLICATE = 1	重复最后一个元素。举例: aaaaaa|abcdefgh|hhhhhhh
BORDER_REFLECT= 2	边界元素的镜像。举例：cba|abcde-fgh|hgf
BORDER_REFLECT_101 = 3	边界元素的镜像。举例：cba|abcde-fgh|gf

10.14.21 acldvppJpegdPrecisionMode

数据格式	说明
ACL_YUVOUT_ALIGN_DOWN = 0	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。

数据格式	说明
ACL_YUVOUT_ALIGN_UP = 1	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 = 2	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。

10.14.22 aclmdlInputAippType

模型输入的AIPP类型	含义
ACL_DATA_WITHOUT_AIPP = 0	该输入无AIPP信息。
ACL_DATA_WITH_STATIC_AIPP = 1	该输入带静态AIPP信息。
ACL_DATA_WITH_DYNAMIC_AIP = 2	该输入有关联的动态AIPP输入。
ACL_DYNAMIC_AIPP_NODE = 3	该输入本身就是动态AIPP输入，用于配置动态AIPP参数。

10.14.23 acltdtTensorType

Atlas 200/500 A2推理产品，当前版本不支持该枚举值。

数据格式
ACL_TENSOR_DATA_UNDEFINE= 0
ACL_TENSOR_DATA_TENSOR= 1
ACL_TENSOR_DATA_END_OF_SEQUENCE= 2

数据格式
ACL_TENSOR_DATA_ABNORMAL= 3

10.14.24 acldvppChannelDescParamType

数据格式	说明
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，属性值请参见**acldvppCscMatrix**。
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**。
ACL_DVPP_CHANNEL_ID_UINT64	通道ID。
ACL_DVPP_CHANNEL_HEIGHT_UINT32	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。 默认值为8192，最大值为16384，若输入图片高大于8192，则需要调用**acl.media.dvpp_set_channel_desc_param**接口设置通道高度。
ACL_DVPP_CHANNEL_WIDTH_UINT32	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。 默认值为8192，最大值为16384，若输入图片宽大于8192，则需要调用**acl.media.dvpp_set_channel_desc_param**接口设置通道宽度。
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**10.14.21 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。

10.14.25 aclvdecChannelDescParamType

数据格式	说明
ACL_VDEC_CSC_MATRIX_UINT32 = 0	色域转换矩阵属性，属性值请参见**acldvppCscMatrix**。
ACL_VDEC_OUT_MODE_UINT32 = 1	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 = 2	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 = 3	解码回调函数，格式如下：
	`def vdec_call_back_func(dvpp_stream_desc, dvpp_pic_desc, user_data):`
	`pass`

ACL_VDEC_CHANNEL_ID_UINT32 = 4	通道ID。
ACL_VDEC_ENCODE_TYPE_UINT32 = 5	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 = 6	YUV图像存储格式，支持如下格式： ● YUV420SP NV12 ● YUV420SP NV21 ● RGB888 ● BGR888 如果不设置输出格式，默认使用YUV420SP NV12。
ACL_VDEC_OUT_PIC_WIDTH_UINT32 = 7	解码码流最大宽度。

数据格式	说明
ACL_VDEC_OUT_PIC_HEIGHT_UINT32 = 8	解码码流最大高度。
ACL_VDEC_REF_FRAME_NUM_UINT32 = 9	参考帧数量，取值范围[0, 16]。
ACL_VDEC_BIT_DEPTH_UINT32 = 10	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）

10.14.26 aclvencChannelDescParamType

视频编码处理通道属性类型	说明
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。 **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 = 1	回调函数。
ACL_VENC_PIXEL_FORMAT_UINT32 = 2	输入图像格式，该属性类型对应的值为uint32_t类型。 支持如下格式： ● PIXEL_FORMAT_YUV_SEMIPLANAR_420 ● PIXEL_FORMAT_YVU_SEMIPLANAR_420
ACL_VENC_ENCODE_TYPE_UINT32 = 3	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 = 4	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 = 5	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 = 6	关键帧间隔，取值范围[1,65536]。
ACL_VENC_BUF_ADDR_PTR = 7	编码输出缓存指针。

视频编码处理通道属性类型	说明
ACL_VENC_BUF_SIZE_UINT32 = 8	编码输出缓存大小，单位为Byte。 **说明** 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。
ACL_VENC_RC_MODE_UINT32 = 9	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。 默认值0表示VBR模式。
ACL_VENC_SRC_RATE_UINT32 = 10	输入码流帧率，单位fps。 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。
ACL_VENC_MAX_BITRATE_UINT32 = 11	输出码率，单位kbps。
ACL_VENC_MAX_IP_PROP_UINT32 = 12	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。

10.14.27 aclmdlConfigAttr

模型加载配置类型	含义
ACL_MDL_PRIORITY_INT32 = 0	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 = 1	模型加载方式，具体请参见**表10-33**，必选项。
ACL_MDL_PATH_STR = 2	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 = 3	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 = 4	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。

模型加载配置类型	含义
ACL_MDL_WEIGHT_ADDR_PTR = 5	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 = 6	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 = 7	Device上模型所需工作内存（存放模型输入/输出等数据）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 = 8	模型所需工作内存的大小，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_WORKSPACE_ADDR_PTR选项配合使用。
ACL_MDL_INPUTQ_NUM_SIZET = 9	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 = 10	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 = 11	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选项配合使用。
ACL_MDL_OUTPUTQ_ADDR_PTR = 12	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。

表 10-33 ACL_MDL_LOAD_TYPE_SIZET 对应的属性值

模型加载方式	含义
ACL_MDL_LOAD_FROM_FILE = 1	从文件加载离线模型数据，由系统内部管理内存。
ACL_MDL_LOAD_FROM_FILE_WITH_MEM = 2	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。
ACL_MDL_LOAD_FROM_MEM = 3	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。

模型加载方式	含义
ACL_MDL_LOAD_FROM_MEM_WITH_MEM = 4	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。
ACL_MDL_LOAD_FROM_FILE_WITH_Q = 5	从文件加载离线模型数据，模型的输入、输出数据都存放在队列中。

说明

关于如何获取om文件，请参见**4.2 模型构建**。

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pyACL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。

10.14.28 aclMemType

内存类型	含义
ACL_MEMTYPE_DEVICE = 0	Device内存。
ACL_MEMTYPE_HOST = 1	Host内存。
ACL_MEMTYPE_HOST_COMPILE_INDEPENDENT = 2	Host内存。

说明

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：设置该选项时，模糊编译场景下，算子输入或输出的值的变化，不会触发算子重新编译；精确编译场景下，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，无论是模糊编译，还是精确编译，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。

10.14.29 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。

数据格式	说明
ACL_OP_COMPILE_DEFAULT= 0	精确编译，不设置编译flag时，默认是精确编译

数据格式	说明
ACL_OP_COMPILE_FUZZ = 1	模糊编译

10.14.30 aclrtEventWaitStatus

事件状态	说明
ACL_EVENT_STATUS_COMPLETE = 0	完成
ACL_EVENT_STATUS_NOT_READY = 1	未完成
ACL_EVENT_STATUS_RESERVED = 2	预留

10.14.31 aclrtEventRecordedStatus

事件状态	说明
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

10.14.32 aclprofStepTag

数据格式	说明
ACL_STEP_START = 0	step start
ACL_STEP_END = 1	step end

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。

10.14.33 acltdtQueueAttrType

数据格式	说明
ACL_TDT_QUEUE_NAME_PTR = 0	队列名
ACL_TDT_QUEUE_DEPTH_UINT32 = 1	队列深度

10.14.34 acltdtQueueRouteParamType

数据格式	说明
ACL_TDT_QUEUE_ROUTE_SRC_UINT32 = 0	源队列ID
ACL_TDT_QUEUE_ROUTE_DST_UINT32 = 1	目标队列ID
ACL_TDT_QUEUE_ROUTE_STATUS_INT32 = 2	路由绑定状态，0表示未绑定，1表示绑定，2表示绑定异常

10.14.35 acltdtQueueRouteQueryMode

数据格式	说明
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

10.14.36 acltdtQueueRouteQueryInfoParamType

数据格式	说明
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**值
ACL_TDT_QUEUE_ROUTE_QUERY_SRC_ID_UINT32 = 1	指定要查询的源队列ID
ACL_TDT_QUEUE_ROUTE_QUERY_DST_ID_UINT32 = 2	指定要查询的目标队列ID

10.14.37 acldvppChannelMode

数据格式	说明
DVPP_CHNMODE_VPC = 1	指定图片数据处理通道用于实现VPC功能

数据格式	说明
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

10.14.38 aclrtFloatOverflowMode

数据格式	说明
ACL_RT_OVERFLOW_MODE_SATURATION = 0	饱和模式，默认值。
ACL_RT_OVERFLOW_MODE_INFNAN = 1	Inf/NaN模式（符合IEEE 754标准）。
ACL_RT_OVERFLOW_MODE_UNDEF = 2	预留参数。

11 附录

pyACL表达约定
使用约束

11.1 pyACL 表达约定

接口命名规则

接口命名同时满足如下规则：

1. acl+*接口类别缩写*+*操作动词*+*对象*
2. 操作动词和对象均采用首字母大写。

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	media	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类的接口
model	mdl	表示模型推理类的接口
OP	op	表示算子执行类的接口
fv	fv	表示特征向量检索的接口
Profiling	prof	表示Profiling配置类接口

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：acl.mdl.load_from_file_with_mem，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。

11.2 使用约束

- 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。
- 不支持使用fork函数创建多个进程，且在进程中调用pyACL接口的场景，否则进程运行时会报错或者卡死。
- 不支持在**acl.rt.memcpy_async**、**acl.rt.memset_async**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。
- 对于创建类接口（例如：**acl.rt.create_stream**、**acl.rt.create_event**、**acl.create_data_buffer**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：**acl.rt.destroy_stream**、**acl.rt.destroy_event**、**acl.destroy_data_buffer**等），否则，程序可能会异常。
- 对于销毁类接口（例如：**acl.rt.destroy_stream**、**acl.rt.destroy_event**、**acl.rt.free**、**acl.destroy_data_buffer**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，设置为None）。
- **Ascend RC形态**，一个Device上最多只能支持64个用户进程。
- 使用pyACL提供的内存申请接口（例如**acl.rt.malloc**、**acl.media.dvpp_malloc**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**acl.rt.memset**或**acl.rt.memset_async**接口先清空内存，例如acl.rt.memset(dev_buffer_ptr, dev_buffer_size, 0, dev_buffer_size)。
- Ascend RC形态下，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
  ```