Atlas 200 DK V100R020C00

应用软件开发指南

文档版本	01
发布日期	2021-03-20

华为技术有限公司

商标声明


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默示的声明或保证。

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。

目 录

1 新手指引

1.1 读者对象

本文档适用于基于ACL接口进行应用开发的人员，通过本文档您可以达成：

- 了解ACL的功能架构、基本概念以及接口的典型调用流程。
- 使用ACL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C++/C语言程序开发能力。
- 对机器学习、深度学习有一定的了解。

1.2 文档结构总览

本文档分为以下几章：

- **2 简介**：介绍ACL（Ascend Computing Language）的基础知识，包括功能、基本概念、各概念之间的关系、如何获取样例等。
- **3 接口调用流程介绍**：介绍各场景下的ACL接口调用流程。
- **4 开发流程**：介绍使用ACL提供的接口开发应用的基本步骤。
- **5 准备环境**：介绍准备开发环境和运行环境时需要参考的文档。
- **6 开发首个应用**：以开发图片分类应用为例，按照开发流程，结合示例代码，介绍各步骤的基本原理。
- **7 开发关键功能的详细介绍**：ACL各功能点的详细介绍。
- ACL 接口和样例：
  - **8 ACL API参考**：介绍API的功能、原型、参数等。

- **9 ACL样例使用指导**：介绍如何使用ACL提供的样例。

1.3 表达约定

接口命名规则

接口命名同时满足如下规则：

1. 规则1：acl+*接口类别缩写+操作动词+对象*
2. 规则2：操作动词和对象均采用首字母大写

接口类别

接口类别	缩写	描述
runtime	rt	表示运行管理类的接口。
DVPP	dvpp	表示媒体数据处理类的接口
AIPP	aipp	表示aipp（AI Preprocessing）类的接口
CBLAS	blas	表示blas类接口
model	mdl	表示模型推理类的接口
graph	grph	表示graph类的接口
driver	drv	表示驱动类的接口
OP	op	表示算子执行类的接口
fv	fv	表示特征向量检索接口

注：

1. 缩写原则上不超过4个字母

2. 在接口命名中，如果类别与操作对象重叠时，操作动词后的对象将省略。

如：aclmdlLoadFromFileWithMem，表示model类接口，这个接口表示含义是load model from file，因此在接口命名中Load后面 mdl将被省略。

变量命名

本文代码示例中涉及的变量，其中，命名带下划线的变量（例如：deviceId_）表示类的私有变量。

关于销毁类接口的使用约束

对于销毁类接口（例如：aclrtFree、aclDestroyDataBuffer等），用户调用该类接口后，不能继续使用已释放或销毁的资源，建议用户调用销毁类接口后，将相关资源设置为无效值（例如，置为nullptr）。

2 简介

本文用于指导开发人员基于现有模型、使用ACL（Ascend Computing Language）提供的C++ 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。

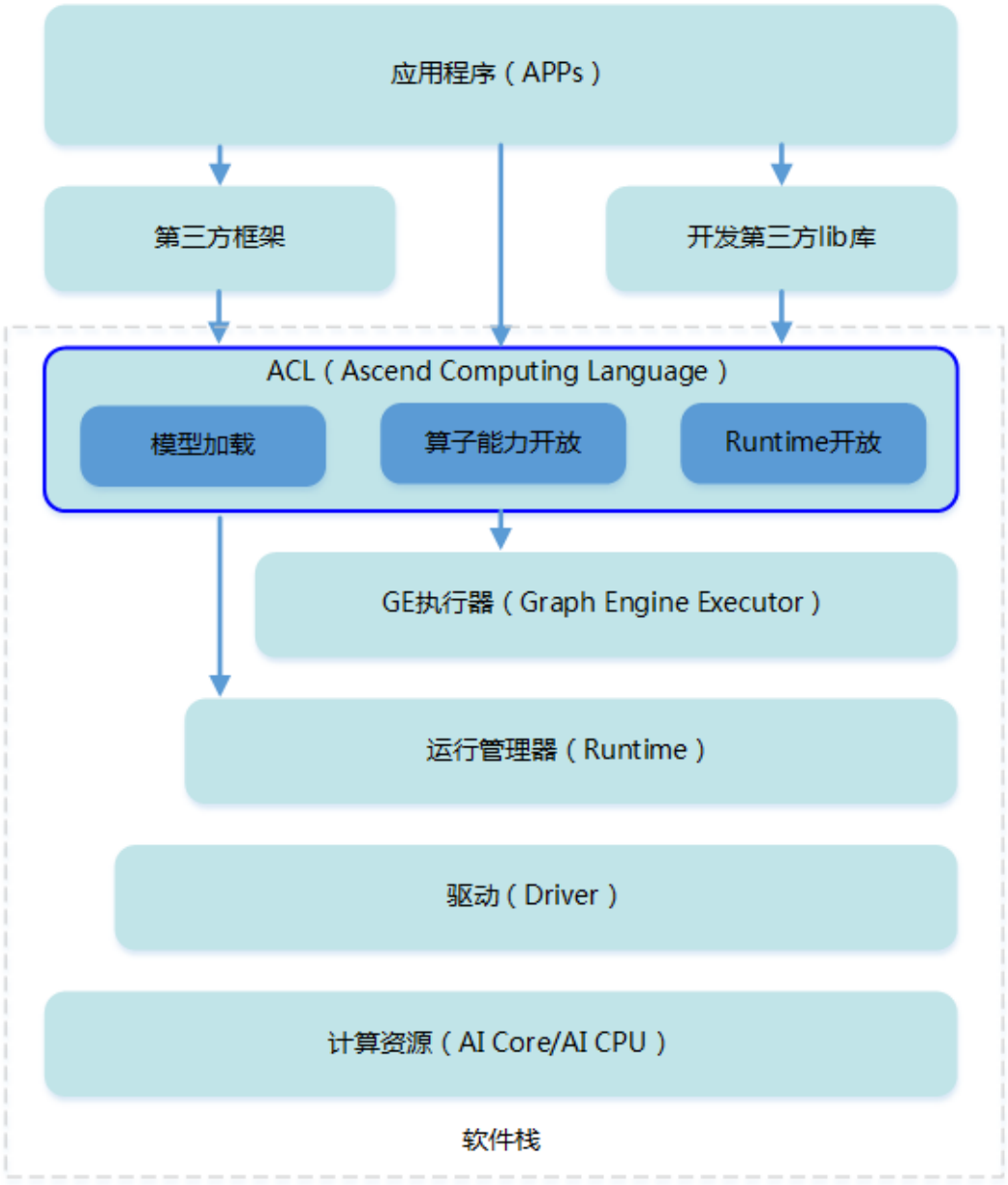
2.1 什么是 ACL

ACL（Ascend Computing Language）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++ 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。用户可以通过第三方框架调用ACL接口，以便使用昇腾AI处理器的计算能力；用户还可以使用ACL封装实现第三方lib库，以便提供昇腾AI处理器的运行管理、资源管理能力。

在运行应用时，ACL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。

图 2-1 逻辑架构图



2.2 基本概念

表 2-1 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。

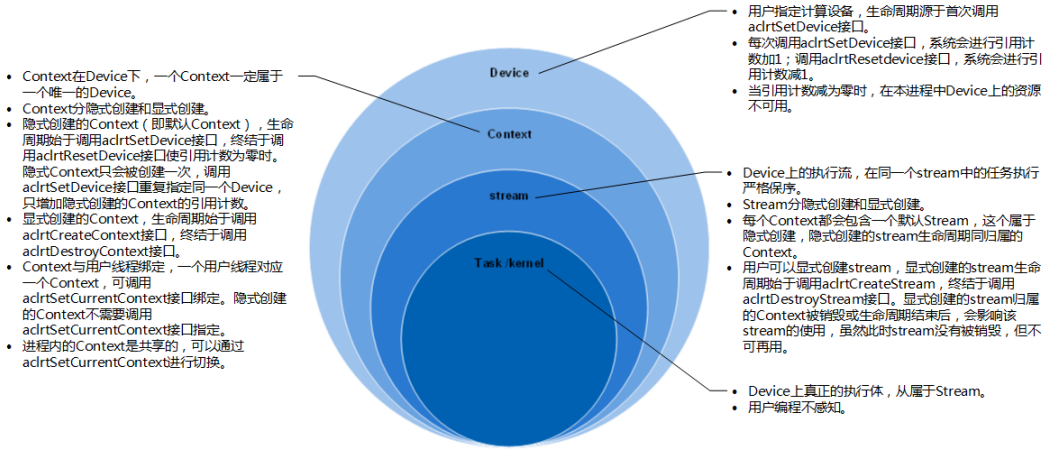
概念	描述
Host	Host指与Device相连接的X86服务器、ARM服务器，会利用Device提供的NN（Neural-Network）计算能力，完成业务。
Device	Device指安装了芯片的硬件设备，利用PCIe接口与Host侧连接，为Host提供NN计算能力。
Context	Context作为一个容器，管理了所有对象（包括Stream、Event、设备内存等）的生命周期。不同Context的Stream、不同Context的Event是完全隔离的，无法建立同步等待关系。 Context分为两种： ● 默认Context：调用**aclrtSetDevice**接口指定用于运算的Device时，系统会自动隐式创建一个默认Context，一个Device对应一个默认Context，默认Context不能通过**aclrtDestroyContext**接口来释放。 ● 显式创建的Context：**推荐，**在进程或线程中调用**aclrtCreateContext**接口显式创建一个Context。
Stream	Stream用于维护一些异步操作的执行顺序，确保按照应用程序中的代码调用顺序在Device上执行。 基于Stream的kernel执行和数据传输能够实现Host运算操作、Host与Device间的数据传输、Device内的运算并行。 Stream分两种： ● 默认Stream：调用**aclrtSetDevice**接口指定用于运算的Device时，系统会自动隐式创建一个默认Stream，一个Device对应一个默认Stream，默认Stream不能通过**aclrtDestroyStream**接口来释放。 ● 显式创建的Stream：**推荐，**在进程或线程中调用**aclrtCreateStream**接口显式创建一个Stream。
Event	支持调用ACL接口同步Stream之间的任务，包括同步Host与Device之间的任务、Device与Device间的任务。 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（AI Preprocessing）用于在AI Core上完成图像预处理，包括色域转换（转换图像格式）、图像归一化（减均值/乘系数）和抠图（指定抠图起始点，抠出神经网络需要大小的图片）等。 AIPP区分为静态AIPP和动态AIPP。您只能选择静态AIPP或动态AIPP方式来处理图片，不能同时配置静态AIPP和动态AIPP两种方式。 ● 静态AIPP：模型转换时设置AIPP模式为静态，同时设置AIPP参数，模型生成后，AIPP参数值被保存在离线模型（*.om）中，每次模型推理过程采用固定的AIPP预处理参数（无法修改）。 如果使用静态AIPP方式，多Batch情况下共用同一份AIPP参数。 ● 动态AIPP：模型转换时仅设置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。设置动态AIPP参数值的接口请参见**8.15.17.2 设置动态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数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数是动态可变的。 ● 动态分辨率H*W: 用户执行推理时,每张图片的分辨率H*W是动态可变的。
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。

2.3 进程、线程、Device、Context、Stream 之间的关系

各基本概念的介绍请参见**2.2 基本概念**。

Device、Context、Stream 之间的关系



线程、Context、Stream 之间的关系

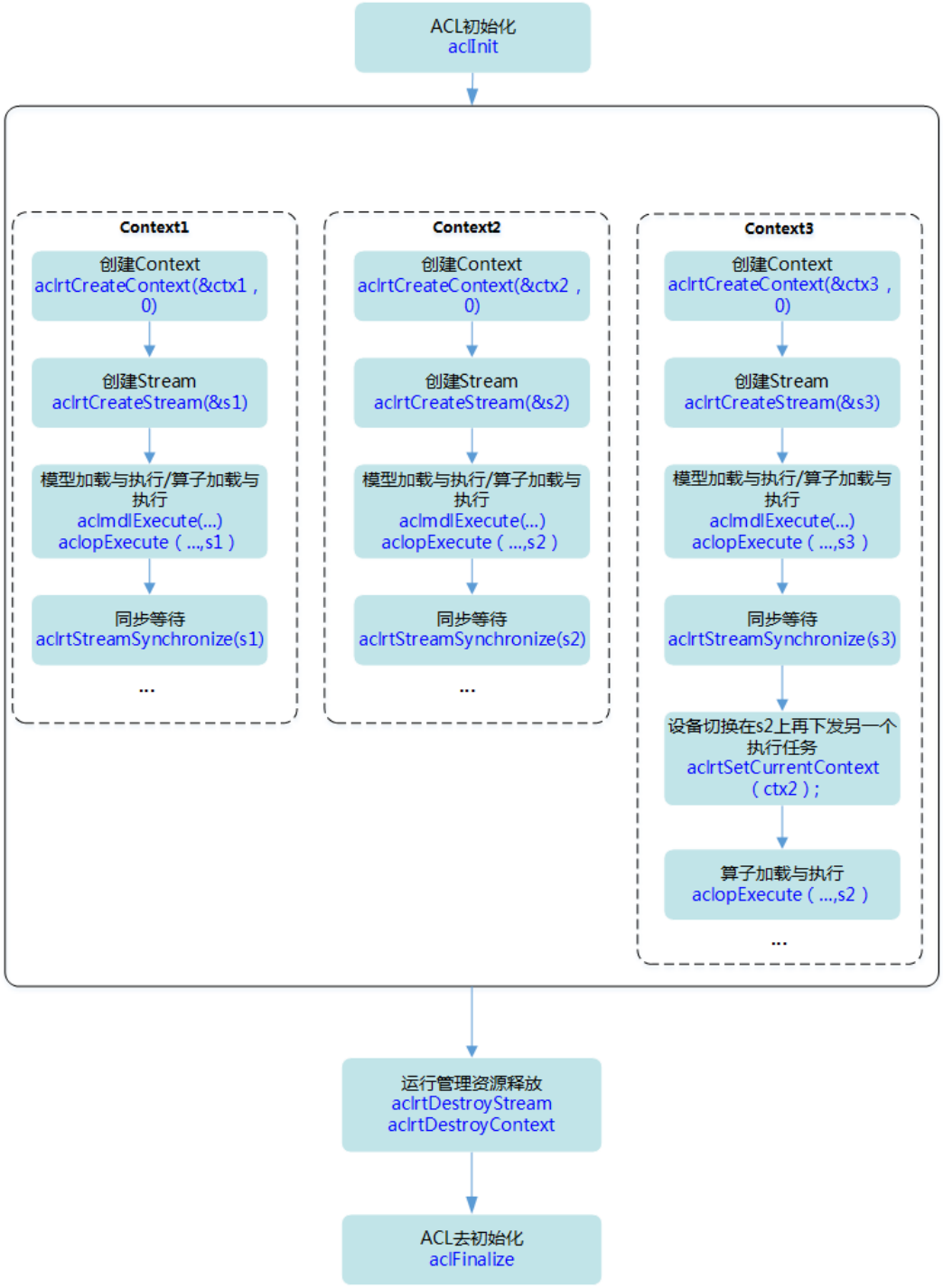
- 一个用户线程一定会绑定一个Context，所有Device的资源使用或调度，都必须基于Context。
- 一个线程中当前会有一个唯一的Context在用，Context中已经关联了本线程要使用的Device。
- 可以通过**aclrtSetCurrentContext**进行Device的快速切换。示例代码如下：

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

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

/*在当前线程中，通过Context切换，使后续计算任务在对应的Device 0上进行*/
aclopExecute(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调度依赖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**aclrtSetCurrentContext**设置当前需要使用的Context。



默认 Context 和默认 Stream 的使用场景

- Device上的操作下发前，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Context不允许用户执行**aclrtGetCurrentContext**或**aclrtSetCurrentContext**操作，也不允许执行**aclrtDestroyContext**操作。
- 默认Context、默认Stream一般适用于简单应用，用户仅仅需要一个Device的计算场景下。多线程应用程序建议全部使用显式创建的Context和Stream。

示例代码如下：

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

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

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

多线程、多 stream 的性能说明

- 线程调度依赖运行的操作系统，Stream上下发了任务后，Stream的调度由Device的调度单元调度，但如果一个进程内的多Stream上的任务在Device存在资源争抢的时候，性能可能会比单Stream低。
- 当前芯片有不同的执行部件，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与多线程单Stream（进程属于多线程，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。

2.4 ACL 内存申请使用说明

用户内存管理有两种管理方式：

1. 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。
2. 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。

接口	用途	输入内存/输出内存
aclrtMemcpyAsync	实现Host内、Host与Device之间、Device内的异步内存复制。	如果是片内Device到Device的内存复制，源地址和目的地址都必须64字节对齐。
aclrtMalloc	申请Device上的内存，同步接口。	• 使用aclrtMalloc接口申请的内存，需要通过aclrtFree接口释放内存。 • 频繁调用aclrtMalloc接口申请内存、调用aclrtFree接口释放内存，会损耗性能，建议用户提前做内存预先分配或二次管理，避免频繁申请/释放内存。 • 调用aclrtMalloc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。

接口	用途	输入内存/输出内存
acldvppMalloc	该接口主要用于分配内存给Device侧媒体数据处理时使用，申请的大页内存满足数据处理的要求（例如，内存首地址128对齐），同步接口。调用该接口申请内存后，必须使用acldvppFree接口释放内存。 媒体数据处理各功能的详细介绍请参见8.12 媒体数据处理。	**实现VPC功能时，输入/输出内存的要求如下**： • 内存地址起始要求16对齐。调用acldvppMalloc接口/acldvppFree接口申请或释放内存。 • 内存大小与图片数据的格式相关，计算公式如下： YUV400SP、YUV420SP：widthStride*heightStride*3/2 YUV422SP：widthStride*heightStride*2 YUV444SP：widthStride*heightStride*3 YUV422packed：widthStride*heightStride YUV444packed、RGB888：widthStride*heightStride XRGB8888：widthStride*heightStride
		实现JPEGD功能时： • 输入内存： – 输入内存的大小就是指实际的输入图片所占用的大小。 – 输入内存首地址要求128对齐。Device的内存，调用acldvppMalloc接口/acldvppFree接口申请或释放内存。 • 输出内存： – 输出内存大小与图片数据的格式相关，计算公式如下： YUV420SP：widthStride*heightStride*3/2 YUV422SP：widthStride*heightStride*2 YUV444SP：widthStride*heightStride*3 – 输出内存首地址要求128字节对齐。Device的内存，调用acldvppMalloc接口/acldvppFree接口申请或释放内存。如果申请大块内存时，内存申请计算应该是（n表示图片数量）：输出内存大小 + (n-1)*AlignTo128(输出内存大小+8)

接口	用途	输入内存/输出内存
		实现JPEGE功能时： ● 关于输入内存： – 输入内存大小与图片数据的格式相关，计算公式如下： YUV422packed：widthStride*heightStride YUV420SP：widthStride*heightStride*3/2 – 输入内存首地址要求128对齐。Device的内存，调用**acldvppMalloc**接口/**acldvppFree**接口申请或释放内存。 ● 关于输出内存： – 输出内存的大小就是指实际的编码后图片所占用的大小。 – 输出内存首地址要求128对齐。Device的内存，调用**acldvppMalloc**接口/**acldvppFree**接口申请或释放内存。
		实现VDEC功能时： ● 关于输入内存： Device的内存，支持调用**aclrtMalloc**接口/**aclrtFree**接口申请/释放内存，也支持调用**acldvppMalloc**/**acldvppFree**接口申请/释放内存。 ● 关于输出内存： – 输出内存大小与图片数据的格式相关，计算公式如下： YUV420SP：widthStride*heightStride*3/2 – 输出的内存首地址要求16对齐。Device的内存，调用**aclrtMalloc**接口/**aclrtFree**接口申请或释放内存。
		实现VENC功能时： ● 关于输入内存 Device的内存，调用**acldvppMalloc**/**acldvppFree**申请/释放内存。 ● 关于输出内存 不需要用户管理输出内存，由系统管理内存。
aclrtMallocHost	申请Host上的内存，同步接口。	● 申请的内存不能在Device使用，需要显式拷贝到Device。 ● 使用aclrtMallocHost接口申请的内存，需要通过**aclrtFreeHost**接口释放内存。

关于媒体数据处理时自行管理内存时的典型场景：

图 2-2 VDEC 场景

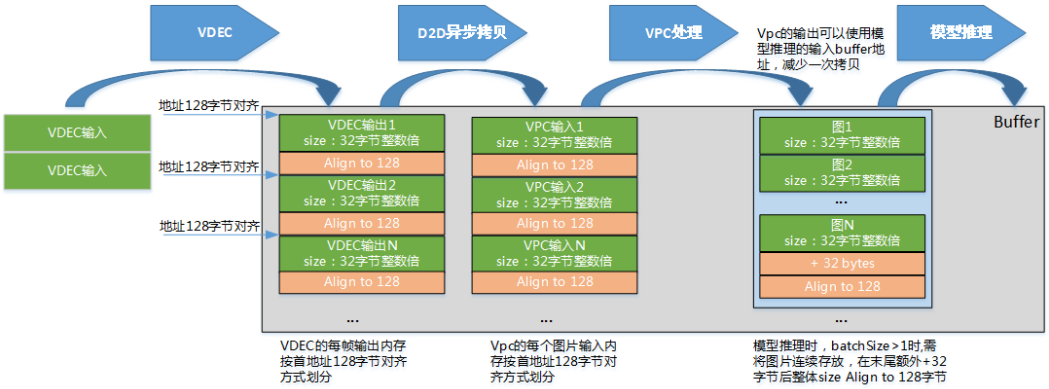
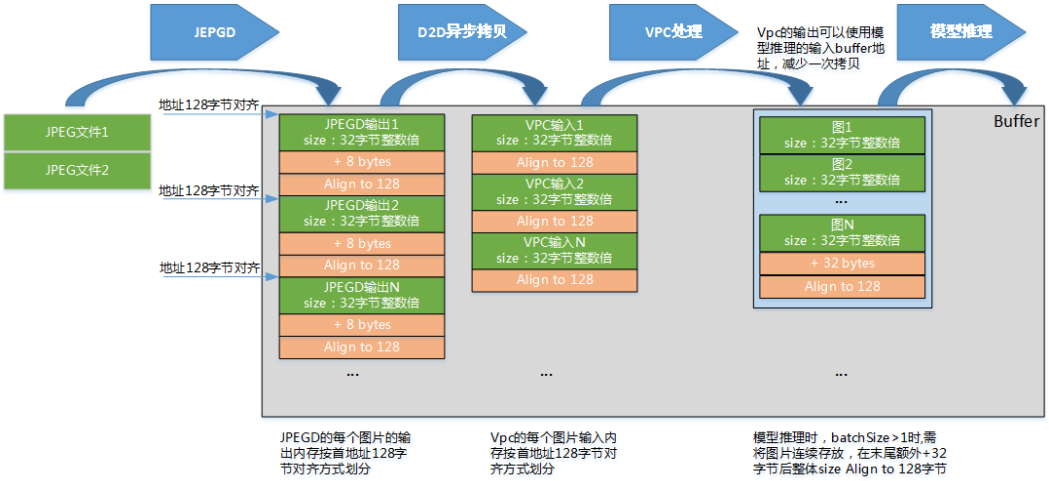


图 2-3 JPEGD 场景



2.5 如何获取 Sample

通过Sample，您可以查看接口调用的示例代码。

当前ACL提供了如下样例，每个样例的关键功能点如表2-2所示，每个样例的获取、基本原理和使用指导请参见《应用软件开发指南》中的“AscendCL样例使用指导”。

- acl_execute_gemm：实现两个矩阵相乘的运算。
- acl_dvpp_resnet50：基于Caffe resnet-50网络（单输入、单Batch）实现图片分类的功能，包含图片数据预处理（解码、缩放）。
- acl_vpc_jpege_resnet50：基于Caffe ResNet-50网络（单输入、单Batch）实现图片分类的功能，包含图片解码、抠图、缩放、编码。
- acl_vdec_resnet50：基于Caffe ResNet-50网络（单输入、单Batch）实现图片分类的功能，包含视频解码。
- acl_resnet50：基于Caffe resnet-50网络（单输入、单Batch）实现图片分类的功能，不包含图片数据预处理（解码、缩放）,同步推理。
- acl_resnet50_async：基于Caffe resnet-50网络（单输入、单Batch）实现图片分类的功能，不包含图片数据预处理（解码、缩放），异步推理。
- dvpp：基于DVPP实现图片的抠图、缩放、解码、编码以及视频的解码、编码功能。

表 2-2 Sample 说明

Sample名称	Device管理	Context管理	Stream管理	同步等待	数据传输	数据预处理	模型推理	数据后处理	单算子调用
acl_execute_gemm	-	-	√	√	√	-	-	-	√
acl_dvpp_resnet50	√	√	√	√	√	√	√	√	√
acl_vpc_jpege_resnet50	√	√	√	√	√	√	√	√	√
acl_vdec_resnet50	√	√	√	√	√	√	√	√	√
acl_resnet50	√	√	√	-	-	-	√	√	-
acl_resnet50_async	√	√	√	√	-	-	√	√	-
dvpp	-	-	-	-	√	√	-	-	-

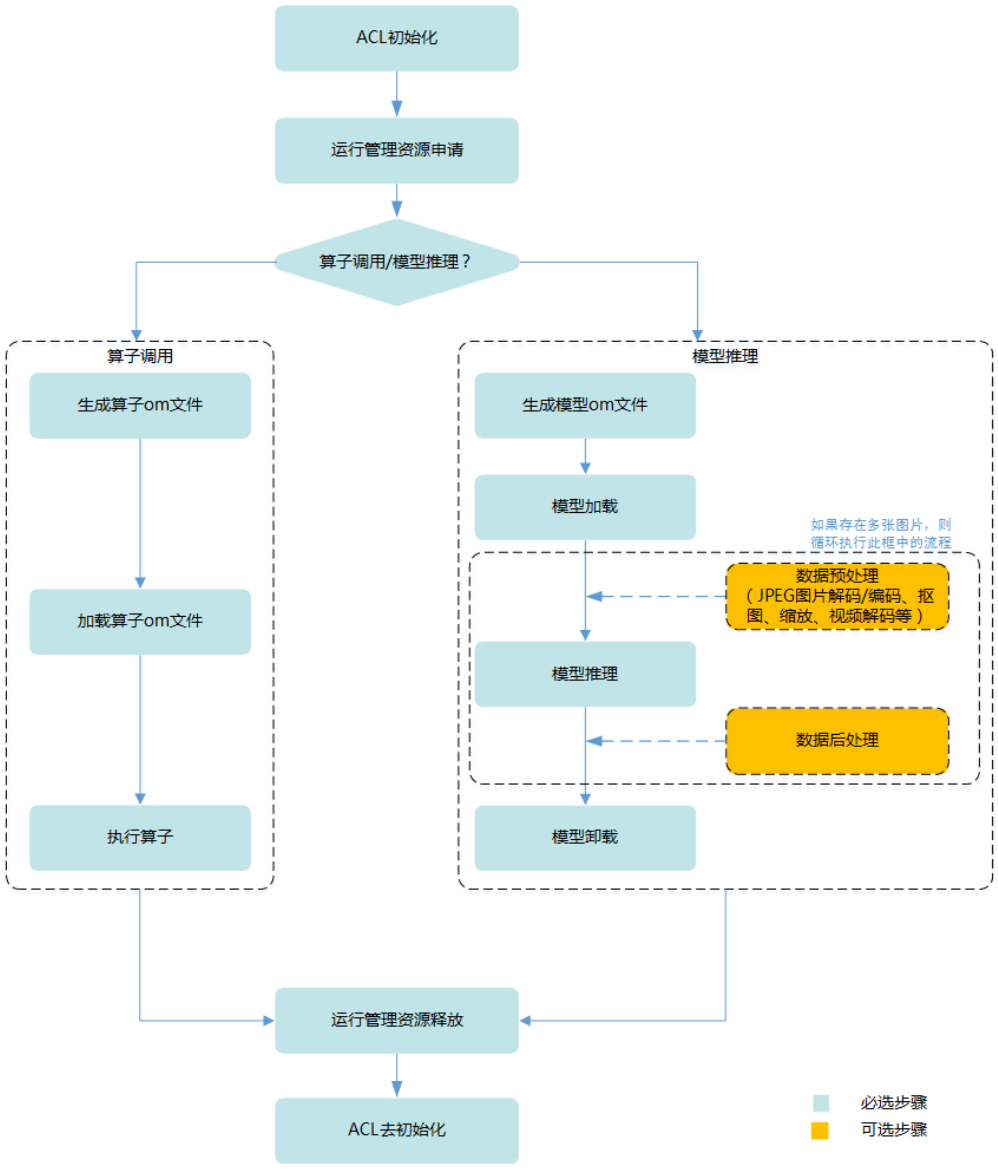
2.6 如何查看日志

运行应用时如果出错，您可以参见《**日志参考**》获取日志文件，查看日志文件中详细报错。

3 接口调用流程介绍

3.1 主要接口调用流程

图 3-1 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. ACL初始化：调用**aclInit**接口实现初始化ACL。
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**3.2 运行管理资源申请**。
3. 算子调用，具体流程，请参见**3.6 算子调用**。

a. 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工具使用指导**》。
b. 加载算子om文件，运行算子时使用。
c. 执行算子，输出算子的运行结果。

4. 模型推理。
   a. 生成模型om文件：需使用ATC工具将第三方网络（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工具使用指导**》。
   b. 模型加载：模型推理前，需要先将对应的模型加载到系统中。具体流程，请参见**3.3 模型加载**。
   c. （可选）数据预处理：可实现JPEG图片解码、视频解码、抠图/图片缩放/格式转换等、JPEG图片编码等功能。具体流程，请参见**3.4 数据预处理**。
   d. 模型推理：使用模型实现图片分类、目标识别等功能，目前ACL提供同步推理接口和异步推理接口，支持动态Batch、动态分辨率、**动态AIPP**等场景。具体流程，请参见**3.5 模型推理**。
   e. （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。
   f. 模型卸载：调用**aclmdlUnload**接口卸载模型。
5. 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**3.8 运行管理资源释放**。
6. ACL去初始化：调用**aclFinalize**接口实现ACL去初始化。

说明

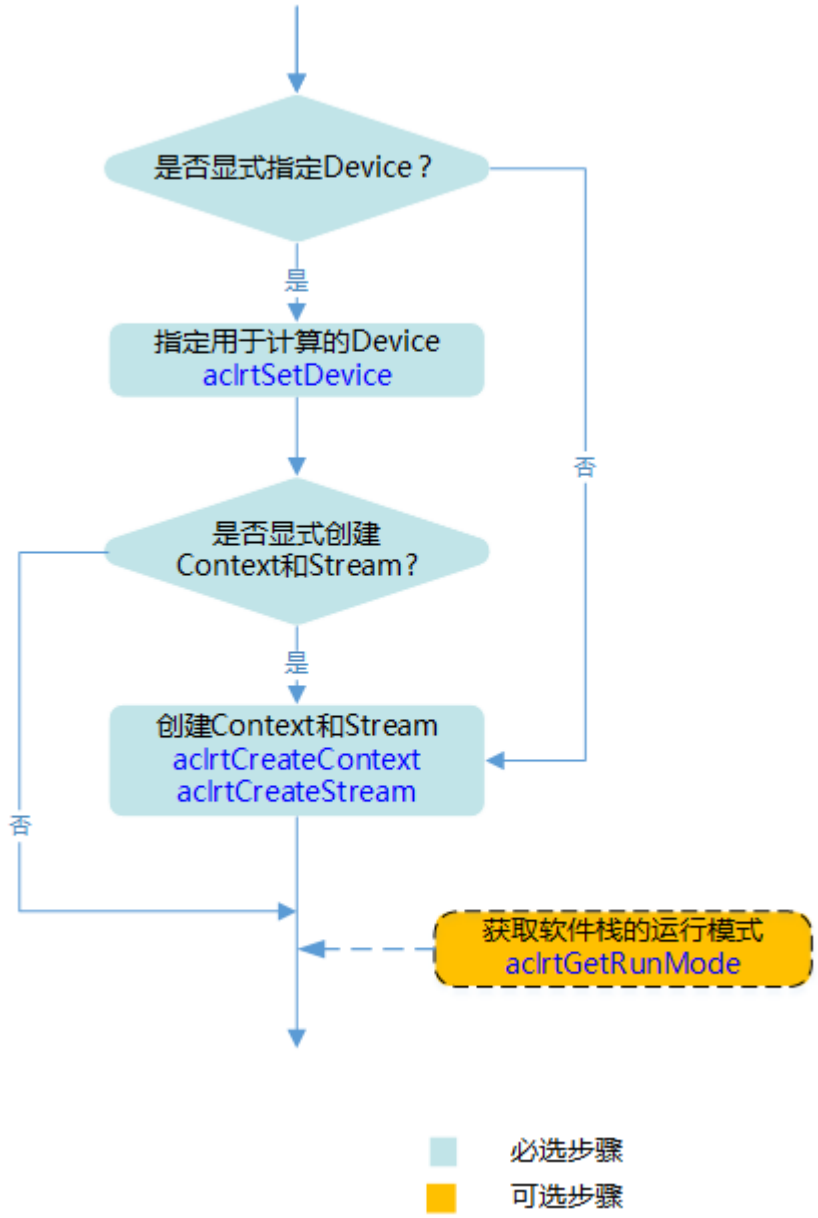
在应用开发过程中，各环节都涉及内存的申请与释放、数据传输（通过内存复制实现）、数据类型的创建与销毁，因此未在图中一一标识，关于内存申请与释放、内存复制的接口请参见**8.7 内存管理**，数据类型的创建与销毁的接口请参见**8.15 数据类型及其操作接口**。

3.2 运行管理资源申请

您需要依次申请运行管理资源，包括：**Device**、**Context**、**Stream**。其中创建Context、Stream的方式分为隐式创建和显式创建。

- 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，执行结果取决于线程调度的顺序。
- 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。

图 3-2 运行管理资源申请流程



1. 申请运行管理资源时，需按顺序依次申请：**Device**、**Context**、**Stream**。
   - 调用**aclrtSetDevice**接口显式指定用于运算的Device。
     - 调用**aclrtCreateContext**接口显式创建Context，调用**aclrtCreateStream**接口显式创建Stream。
     - 不显式创建Context和Stream，系统会使用默认Context、默认Stream，该默认Context、默认Stream是在调用**aclrtSetDevice**接口时隐式创建的。

       默认Stream作为接口入参时，直接传NULL。
   - 不显式指定用于运算的Device。

     调用**aclrtCreateContext**接口显式创建Context，调用**aclrtCreateStream**接口显式创建Stream。系统在显式创建Context时，系统内部会调用

aclrtSetDevice接口指定运行的Device，Device ID通过aclrtCreateContext接口传入。

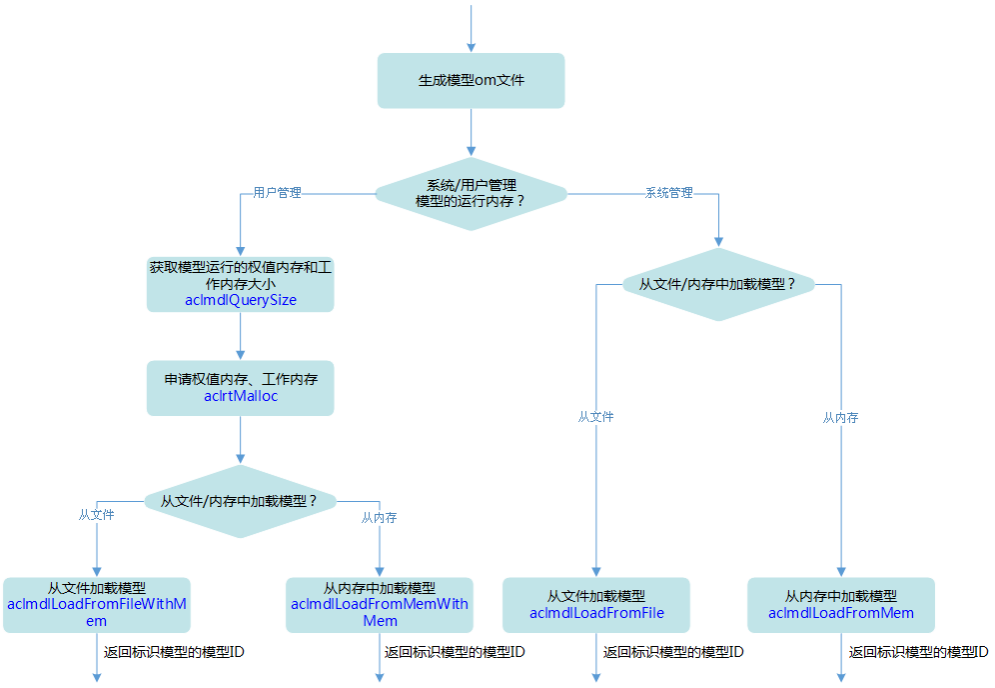
2. （可选）当同一个应用的可执行文件既支持在Host执行，也支持在Device上执行时，在编程时需要调用aclrtGetRunMode接口获取软件栈的运行模式，根据运行模式来判断后续的内存申请接口调用逻辑。
   a. 如果应用的可执行文件在Host上执行，则可能涉及Host与Device之间的数据传输，需要调用aclrtMemcpy接口（同步接口）或aclrtMemcpyAsync接口（异步接口）通过内存复制的方式实现数据传输。
   b. 如果应用的可执行文件在Device上执行，则不涉及Host与Device之间的数据传输。

   说明

   Ascend RC场景下，Host和Device都部署在开发者板上，也不涉及Host与Device之间的数据传输。

3.3 模型加载

图 3-3 模型加载流程



- 在模型加载前，需使用ATC工具将第三方网络（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工具使用指导》。
  - 静态多Batch场景下，在转换模型时，需通过input_shape参数设置Batch数。
  - 动态Batch场景下，在转换模型时，需通过dynamic_batch_size参数设置每个档位的Batch数。
  - 动态分辨率场景下，在转换模型时，需通过dynamic_image_size参数设置每个档位的输入图片分辨率。
  - 动态AIPP场景下，在转换模型时，需通过insert_op_conf参数中的配置文件指定aipp_mode为dynamic。

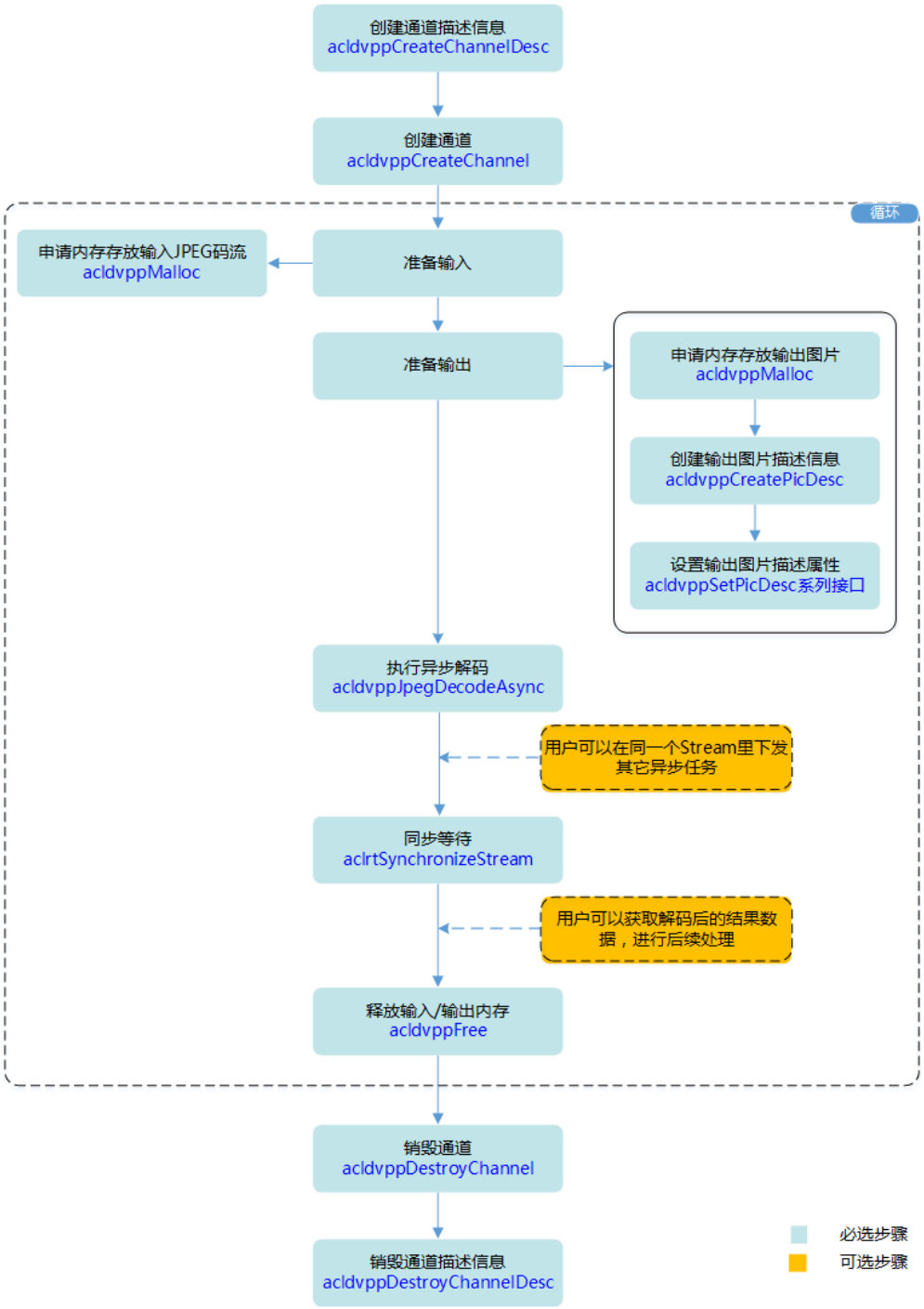
- 支持以下方式加载模型，模型加载成功后，返回标识模型的模型ID：
  - **aclmdlLoadFromFile**：从文件加载离线模型数据，由系统内部管理内存。
  - **aclmdlLoadFromMem**：从内存加载离线模型数据，由系统内部管理内存。
  - **aclmdlLoadFromFileWithMem**：从文件加载离线模型数据，由用户自行管理模型运行的内存（包括工作内存和权值内存，工作内存用于模型的输入输出等数据，权值内存用于存放权值数据）。
  - **aclmdlLoadFromMemWithMem**：从内存加载离线模型数据，由用户自行管理模型运行的内存（包括工作内存和权值内存）。

3.4 数据预处理

数据预处理包括：可实现JPEG图片解码、视频解码、抠图/图片缩放、JPEG图片编码、视频编码等功能。

3.4.1 JPEG 图片解码

图 3-4 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YUV420SP、YUV422SP、YUV444SP编码格式的图片。关键流程说明如下：

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

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

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

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

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

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

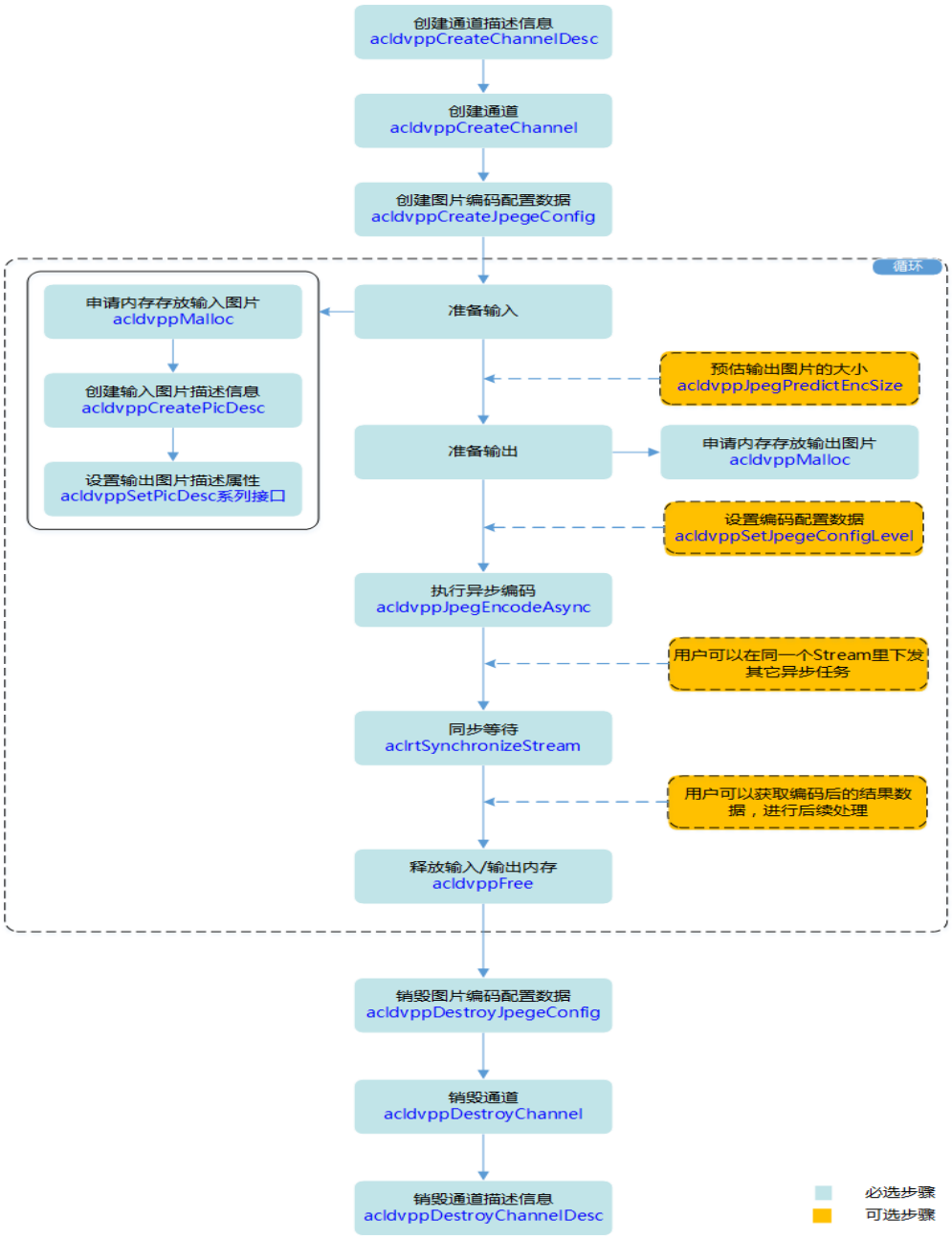
4. 在解码结束后，需及时调用**acldvppFree**接口释放输入、输出内存。

5. 调用**acldvppDestroyChannel**接口销毁图片数据处理的通道。

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

3.4.2 JPEG 图片编码

图 3-5 JPEG 图片编码



当前系统支持将YUV格式图片编码成.jpg图片，关键流程说明如下：

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

   创建图片数据处理的通道前，需先调用**acldvppCreateChannelDesc**接口创建通道描述信息。
2. 调用**acldvppCreateJpegeConfig**接口创建图片编码配置数据。
3. 实现JPEG图片编码功能前，若需要申请Device上的内存存放输入或输出数据，需调用**acldvppMalloc**申请内存。

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

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

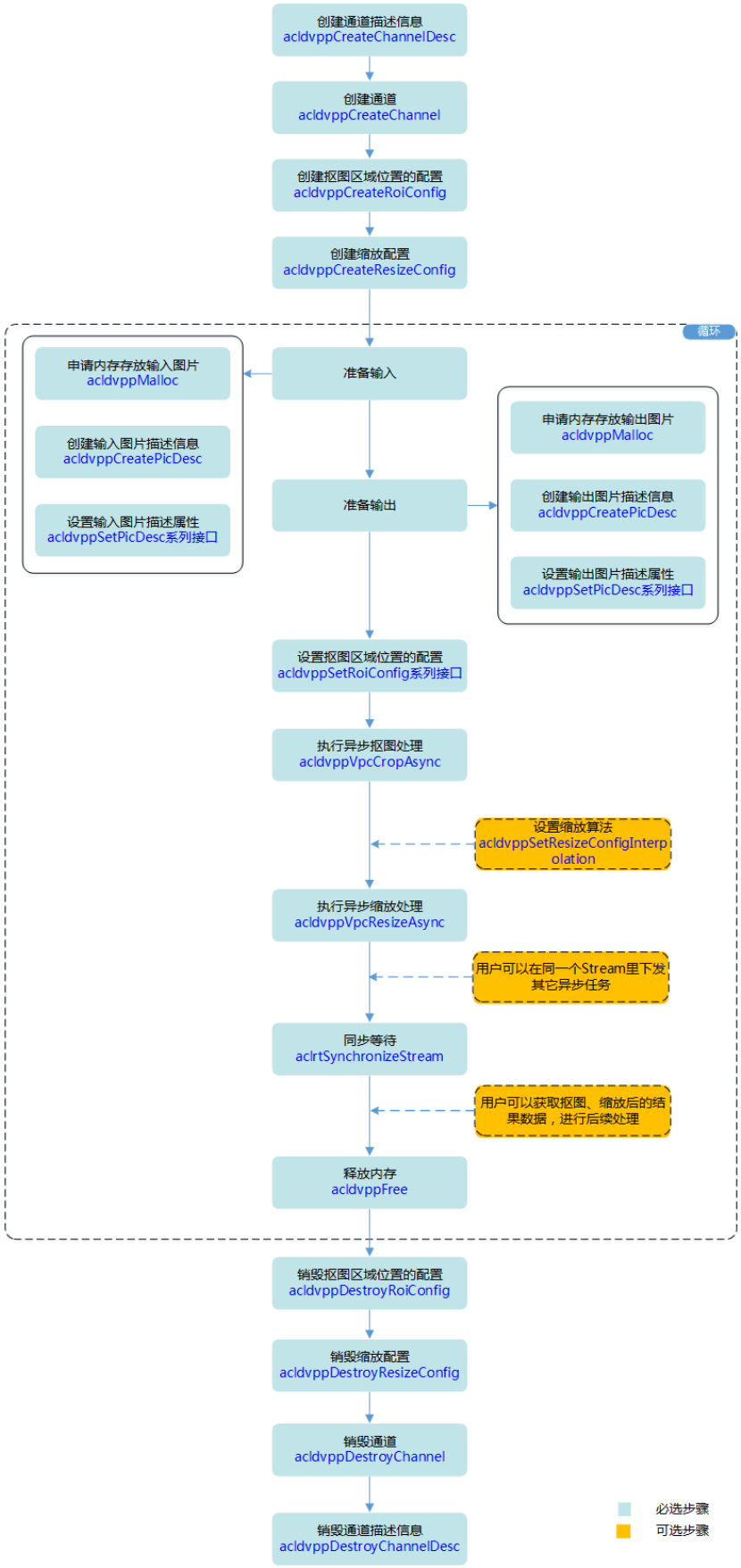
   对于异步接口，还需调用**aclrtSynchronizeStream**接口阻塞Host运行，直到指定Stream中的所有任务都完成。

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

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

3.4.3 抠图/缩放

图 3-6 抠图缩放流程



当前系统支持对输入图片做抠图、缩放，关键流程说明如下：

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

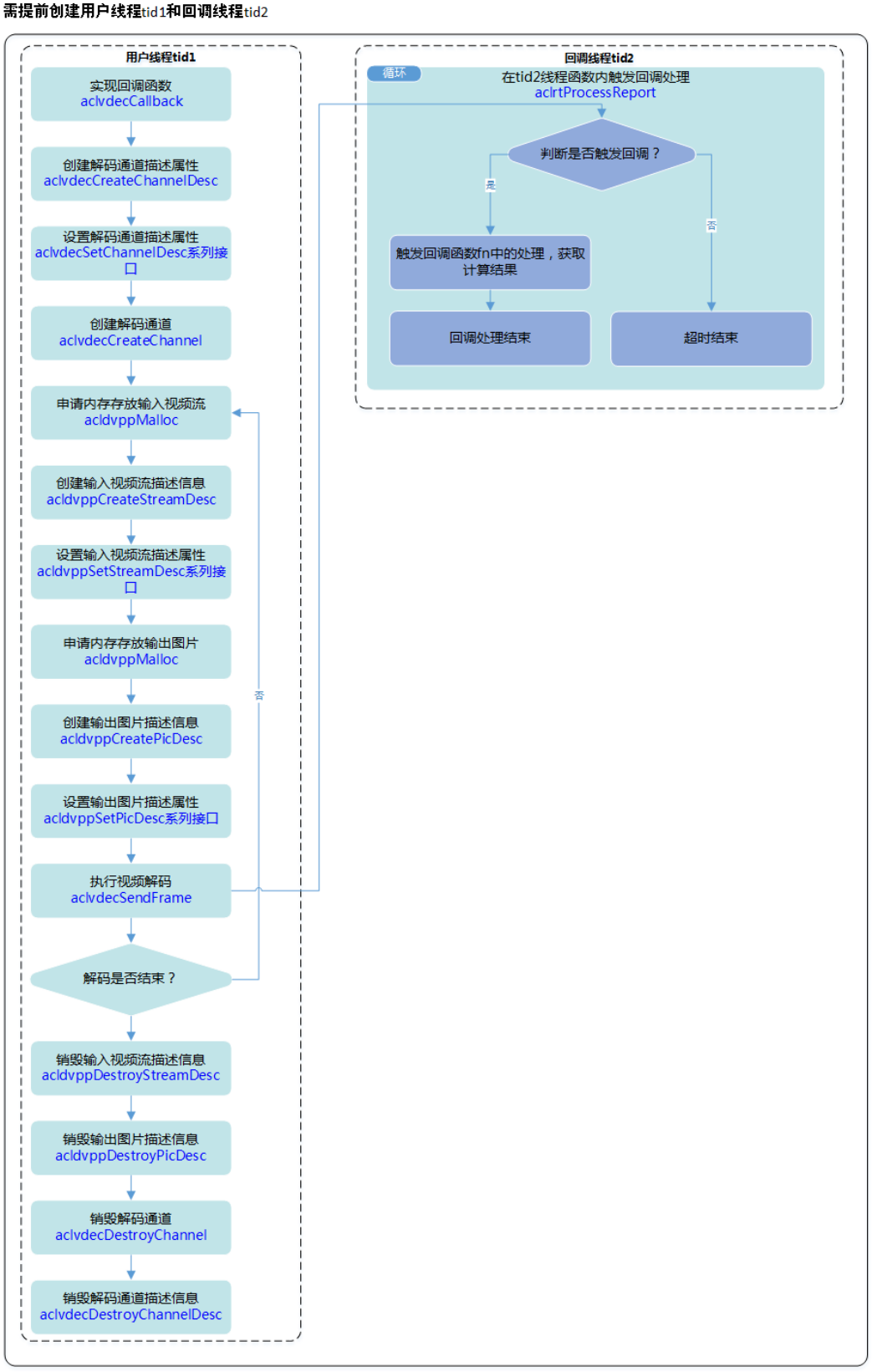
   创建图片数据处理的通道前，需先调用**acldvppCreateChannelDesc**接口创建通道描述信息。
2. 调用**acldvppCreateRoiConfig**接口、**acldvppCreateResizeConfig**接口分别创建抠图区域位置的配置、缩放配置。
3. 实现抠图、缩放功能前，若需要申请Device上的内存存放输入或输出数据，需调用**acldvppMalloc**申请内存。
4. 执行抠图、缩放。
   - 关于抠图：
     - 调用**acldvppVpcCropAsync**异步接口，按指定区域从输入图片中抠图，再将抠的图片存放到输出内存中，作为输出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

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

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

3.4.4 视频解码

图 3-7 视频解码流程



视频解码的关键流程说明如下：

1. **调用aclvdecCreateChannel接口创建视频码流数据处理的通道。**
   - 创建视频码流数据处理通道前，需先执行以下操作：
     i. 调用aclvdecCreateChannelDesc接口创建通道描述信息。
     ii. 调用aclvdecSetChannelDesc系列接口设置通道描述信息的属性，包括解码通道号、线程、回调函数、视频编码协议等，其中：
        1) 回调函数需由用户提前创建，用于在视频解码后，获取解码数据，并及时释放相关资源，回调函数的原型请参见8.12.7.3 aclvdecCallback。

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

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

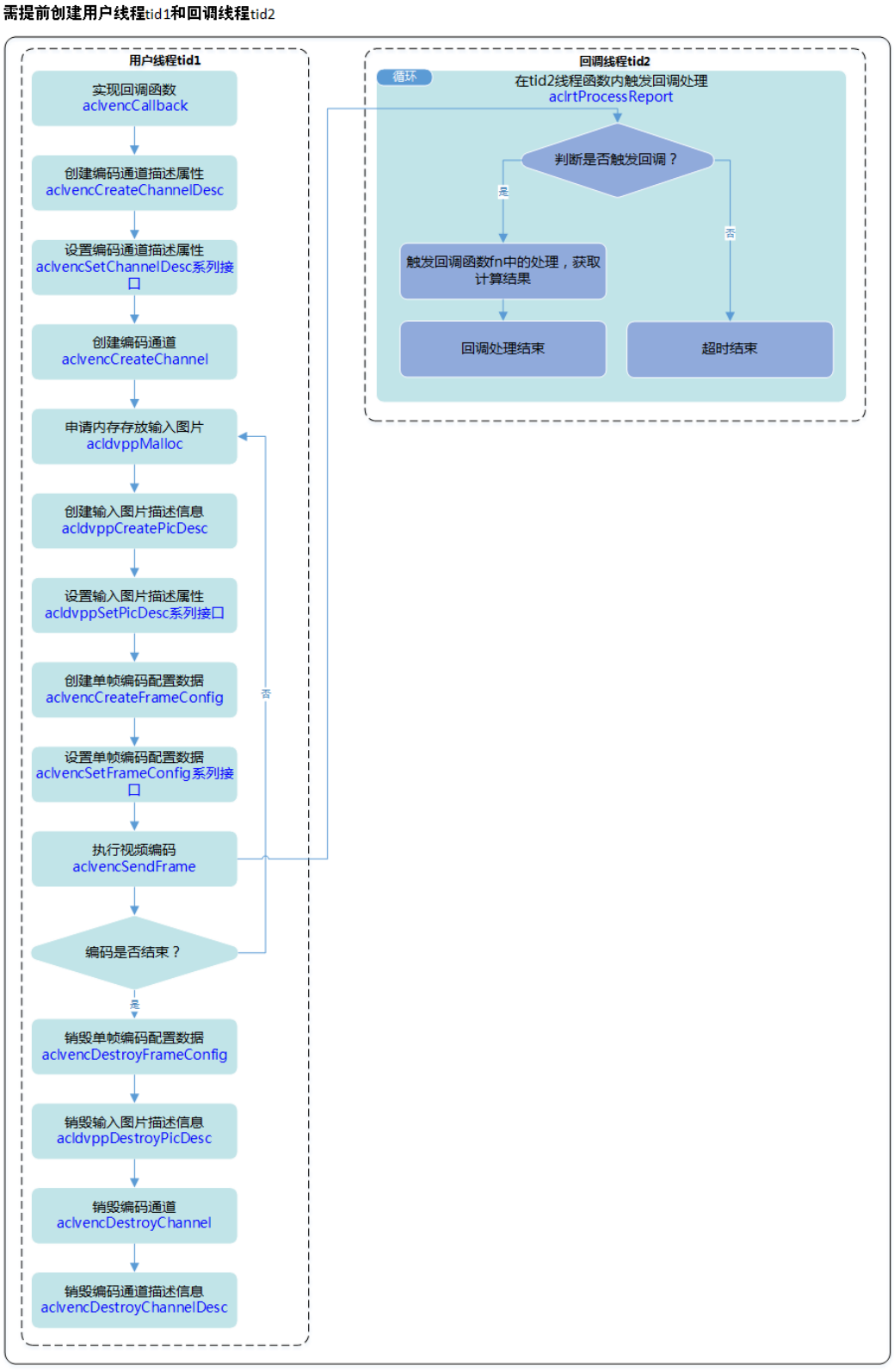
           说明

           如果不调用aclvdecSetChannelDescOutPicFormat接口设置输出格式，则默认使用YUV420SP NV12。
   - aclvdecCreateChannel接口内部封装了如下接口，无需用户单独调用：
     i. aclrtCreateStream接口：显式创建Stream，VDEC内部使用。
     ii. aclrtSubscribeReport接口：指定处理Stream上回调函数的线程，回调函数和线程是由用户调用aclvdecSetChannelDesc系列接口时指定的。
2. **调用aclvdecSendFrame接口将视频码流解码成YUV420SP格式的图片。**
   - 视频解码前，需先执行以下操作：
     - 调用acldvppCreateStreamDesc接口创建输入视频码流描述信息，并调用acldvppSetStreamDesc系列接口设置输入视频的内存地址、内存大小、码流格式等属性。
     - 调用acldvppCreatePicDesc接口创建输出图片描述信息，并调用acldvppSetPicDesc系列接口设置输出图片的内存地址、内存大小、图片格式等属性。
   - aclvdecSendFrame接口内部封装了aclrtLaunchCallback接口，用于在Stream的任务队列中增加一个需要在Host上执行的回调函数。用户无需单独调用aclrtLaunchCallback接口。
3. **调用aclvdecDestroyChannel接口销毁视频处理的通道。**
   - 系统会等待已发送帧解码完成且用户的回调函数处理完成后再销毁通道。
   - aclvdecDestroyChannel接口内部封装了如下接口，无需用户单独调用：
     - aclrtUnSubscribeReport接口：取消线程注册（Stream上的回调函数不再由指定线程处理）。
     - aclrtDestroyStream接口：销毁Stream。

- 销毁通道后，需调用**aclvdecDestroyChannelDesc**接口销毁通道描述信息。

3.4.5 视频编码

图 3-8 视频编码流程



视频编码的关键流程说明如下：

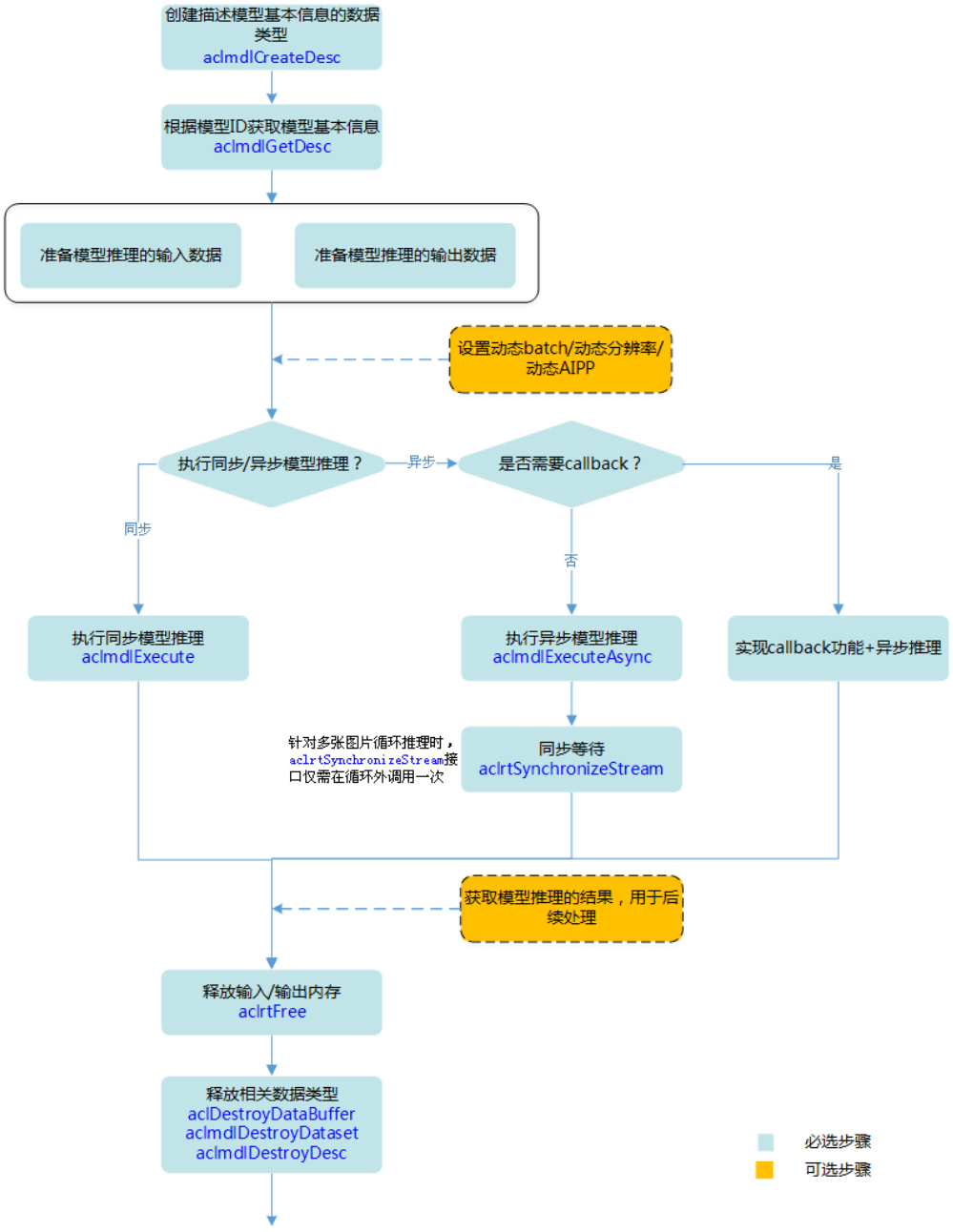
1. **调用aclvencCreateChannel接口创建视频编码处理的通道。**
   - 创建视频编码处理通道前，需先执行以下操作：
     i. 调用**aclvencCreateChannelDesc**接口创建通道描述信息。
     ii. 调用**aclvencSetChannelDesc系列接口**设置通道描述信息的属性，包括线程、回调函数、视频编码协议、输入图片格式等，其中：
        1) 回调函数需由用户提前创建，用于在视频编码后，获取编码数据，并及时释放相关资源，回调函数的原型前参见**8.12.8.3 aclvencCallback**。

           视频编码结束后，建议用户在回调函数内及时释放输入图片内存、以及相应的图片描述信息。视频编码的输出内存由系统管理，不由用户管理，因此无需用户释放。
        2) 线程需由用户提前创建，并自定义线程函数，在线程函数内调用**aclrtProcessReport**接口，等待指定时间后，触发**1.ii.1)**中的回调函数。
   - **aclvencCreateChannel**接口内部封装了如下接口，无需用户单独调用：
     i. **aclrtCreateStream**接口：显式创建Stream，VENC内部使用。
     ii. **aclrtSubscribeReport**接口：指定处理Stream上回调函数的线程，回调函数和线程是由用户调用**aclvencSetChannelDesc系列接口**时指定的。
2. **调用aclvencSendFrame接口将YUV420SP格式的图片编码成H264/H265格式的视频码流。**
   - 视频编码前，需先执行以下操作：
     - 调用**acldvppCreatePicDesc**接口创建输入图片描述信息，并调用**acldvppSetPicDesc系列接口**设置输入图片的内存地址、内存大小、图片格式等属性。
     - 调用**aclvencCreateFrameConfig**接口创建单帧编码配置数据，并调用**aclvencSetFrameConfig系列接口**设置是否强制重新开始I帧间隔、是否结束帧。
   - **aclvencSendFrame**接口内部封装了**aclrtLaunchCallback**接口，用于在Stream的任务队列中增加一个需要在Host上执行的回调函数。用户无需单独调用**aclrtLaunchCallback**接口。
3. **调用aclvencDestroyChannel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。
   - **aclvencDestroyChannel**接口内部封装了如下接口，无需用户单独调用：
     - **aclrtUnSubscribeReport**接口：取消线程注册（Stream上的回调函数不再由指定线程处理）。
     - **aclrtDestroyStream**接口：销毁Stream。
   - 销毁通道后，需调用**aclvencDestroyChannelDesc**接口销毁通道描述信息。

3.5 模型推理

3.5.1 基本的模型推理流程

图 3-9 基本的模型推理流程



模型推理的关键流程说明如下：

1. 调用aclmdlCreateDesc接口创建描述模型基本信息的数据类型。
2. 调用aclmdlGetDesc接口根据**3.3 模型加载**中返回的模型ID获取模型基本信息。
3. 准备模型推理的输入、输出数据，具体流程，请参见**3.5.3 准备模型推理的输入/输出数据**。
4. 设置动态Batch、动态分辨率、动态AIPP的具体流程，请参见**3.5.2 设置动态Batch/动态分辨率/动态AIPP**。

在设置动态Batch、动态分辨率、动态AIPP前，需确保所加载的om模型中已包含动态Batch、动态分辨率、动态AIPP相关的配置，详细说明请参见**3.3 模型加载**。

5. 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**aclmdlExecute**接口
   - 异步推理时调用**aclmdlExecuteAsync**接口

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

     如果同时需要实现Callback功能，请参见**3.7.3 Callback场景**。

6. 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。

7. 释放内存。

   调用**aclrtFree**接口释放Device上的内存。

8. 释放相关数据类型的数据。

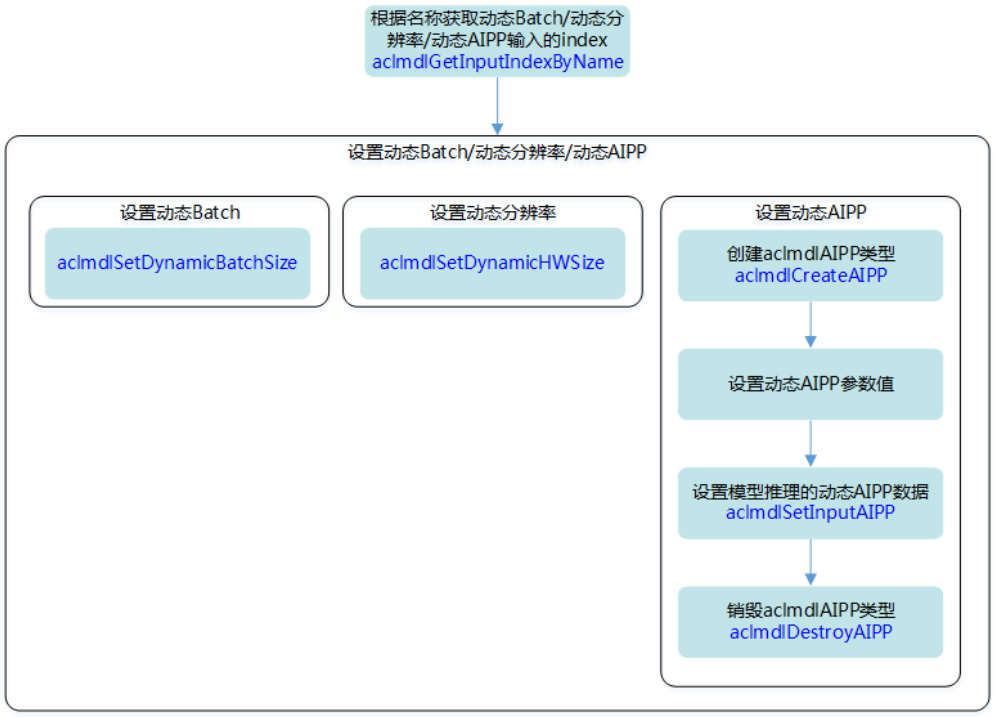
   在模型推理结束后，需及时调用**aclDestroyDataBuffer**接口和**aclmdlDestroyDataset**接口释放描述模型输入的数据，且先调用**aclDestroyDataBuffer**接口，再调用**aclmdlDestroyDataset**接口。如果存在多个输入、输出，需调用多次**aclDestroyDataBuffer**接口。

3.5.2 设置动态 Batch/动态分辨率/动态 AIPP

须知

对同一个模型，不能同时调用**aclmdlSetDynamicBatchSize**接口设置动态Batch、调用**aclmdlSetDynamicHWSize**接口设置动态分辨率。

图 3-10 设置动态 Batch/动态分辨率/动态 AIPP 流程



1. 准备模型推理的动态Batch/动态分辨率/动态AIPP输入的数据，详细流程请参见**3.5.3 准备模型推理的输入/输出数据**。

   申请动态Batch/动态分辨率/动态AIPP输入对应的内存前，需要先调用**aclmdlGetInputIndexByName**接口根据输入名称获取模型中标识动态Batch/动态分辨率/动态AIP输入的index，动态Batch和动态分辨率输入的名称固定为ACL_DYNAMIC_TENSOR_NAME，动态AIPP输入的名称固定为ACL_DYNAMIC_AIPP_NAME。再调用**aclmdlGetInputSizeByIndex**接口根据index获取内存大小。

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

   说明

   - ACL_DYNAMIC_AIPP_NAME是一个宏，宏的定义如下：
     ```
     #define ACL_DYNAMIC_AIPP_NAME "ascend_dynamic_aipp_data"
     ```
   - ACL_DYNAMIC_TENSOR_NAME是一个宏，宏的定义如下：
     ```
     #define ACL_DYNAMIC_TENSOR_NAME "ascend_mbatch_shape_data"
     ```

2. 在成功加载模型之后，执行模型之前，设置动态Batch数/动态分辨率/动态AIPP参数值。

   a. 调用**aclmdlGetInputIndexByName**接口根据输入名称（固定为ACL_DYNAMIC_TENSOR_NAME）获取模型中标识动态Batch输入的index。

   b. 设置动态Batch数/动态分辨率/动态AIPP参数值

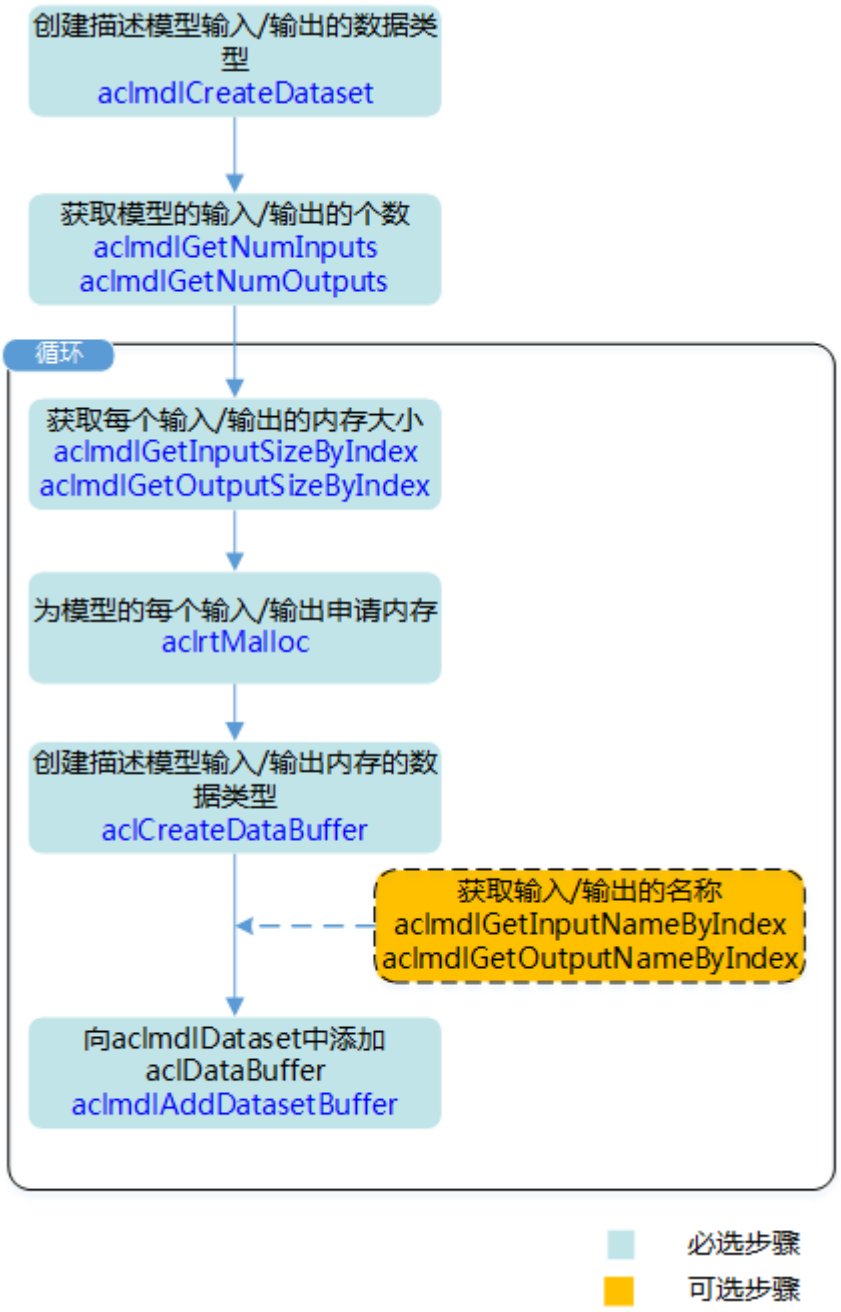
      - 调用**aclmdlSetDynamicBatchSize**接口设置动态Batch数。
      - 调用**aclmdlSetDynamicHWSize**接口设置动态分辨率。

- 调用如下接口设置模型推理的动态AIPP数据。
  1) 调用**aclmdlCreateAIPP**接口创建aclmdlAIPP类型。
  2) 根据实际需求，调用**8.15.17.2 设置动态AIPP参数**中提供的接口设置动态AIPP参数值。

     动态AIPP场景下，**aclmdlSetAIPPSrcImageSize**接口（设置原始图片的宽和高）必须调用。
  3) 调用**aclmdlSetInputAIPP**接口设置模型推理时的动态AIPP数据。
  4) 及时调用**aclmdlDestroyAIPP**接口销毁aclmdlAIPP类型。

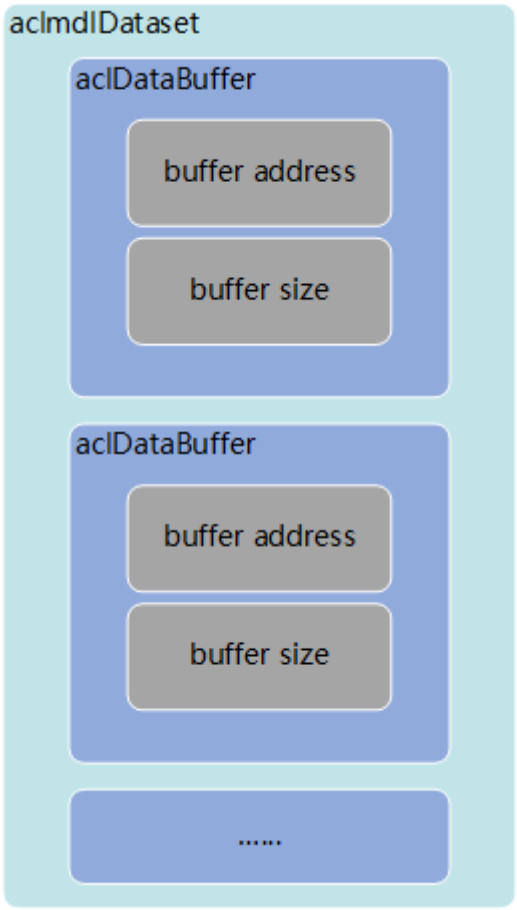
3.5.3 准备模型推理的输入/输出数据

图 3-11 模型推理的输入/输出数据准备流程



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），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，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。

图 3-12 aclmdlDataset 类型与 aclDataBuffer 类型的关系



如果需要将Host上数据传输到Device，则需要调用**aclrtMemcpy**接口（同步接口）或**aclrtMemcpyAsync**接口（异步接口）通过内存复制的方式实现数据传输。

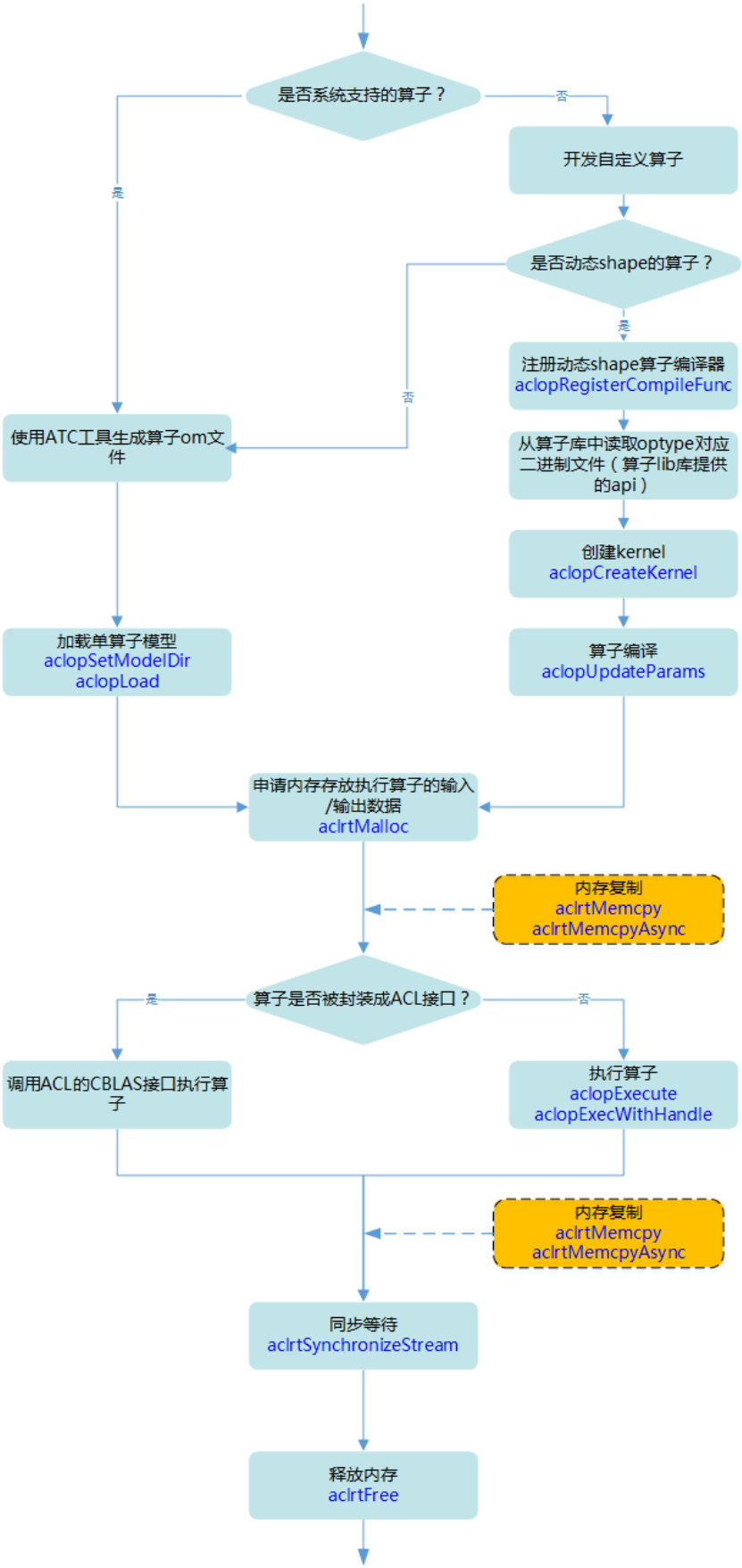
模型存在多个输入、输出时，用户在向aclmdlDataset中添加aclDataBuffer时，为避免顺序出错，可以先获取输入、输出的名称，根据输入、输出名称所对应的index的顺序添加。

说明

对于静态多Batch场景，在创建aclDataBuffer类型的数据时，内存大小需调用**aclmdlGetInputSizeByIndex**接口根据index获取。

3.6 算子调用

图 3-13 算子调用流程



算子调用的基本流程如下（系统支持的算子请参见《 》）：

1. 加载/编译算子。
   - 对于固定Shape的算子，调用ACL接口加载算子：
     - 单算子模型文件，需要用户提前参见《**ATC工具使用指导**》中的使用ATC工具转换模型将单算子定义文件（*.json）编译成适配昇腾AI处理器的离线模型（*.om文件）。
     - 加载单算子模型文件，有两种方式：

       调用**aclopSetModelDir**接口，设置加载模型文件的目录，目录下存放单算子模型文件（*.om文件）。

       调用**aclopLoad**接口，从内存中加载单算子模型数据，由用户管理内存。单算子模型数据是指“单算子编译成*.om文件后，再将om文件读取到内存中”的数据。
   - 对于动态Shape的算子，需提前注册要编译的自定义算子：
     - 调用**aclopRegisterCompileFunc**接口注册算子选择器（即选择Tiling策略的函数），用于在算子执行时，能针对不同Shape，选择相应的Tiling策略。

       算子选择器需由用户提前定义并实现，算子选择器的实现样例请参见《**TBE自定义算子开发指南**》中的：
       - 函数原型：
         ```
         typedef aclError (*aclopCompileFunc)(int numInputs, const aclTensorDesc *const inputDesc[], int numOutputs, const aclTensorDesc *const outputDesc[], const aclopAttr *opAttr, aclopKernelDesc *aclopKernelDesc);
         ```
       - 函数实现：

         用户自行编写代码逻辑实现Tiling策略选择、Tiling参数生成，并将调用aclopSetKernelArgs接口，设置算子Tiling参数、执行并发数等。
     - 调用**aclopCreateKernel**接口将算子注册到系统内部，用于在算子执行时，查找到算子实现代码。
     - 调用**aclopUpdateParams**接口编译指定算子，触发算子选择器的调用逻辑。
2. 调用**aclrtMalloc**接口申请Device上的内存，存放执行算子的输入、输出数据。

   如果需要将Host上数据传输到Device，则需要调用**aclrtMemcpy**接口（同步接口）或**aclrtMemcpyAsync**接口（异步接口）通过内存复制的方式实现数据传输。
3. 执行算子。
   - 如果是系统内置的算子GEMM，该算子已经被封装成ACL接口，用户可直接调用**CBLAS接口**执行该算子。
   - 如果是系统内置的算子，但未被封装成ACL接口，当前支持以下两种方式执行算子：
     - 您需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、调用**aclopExecute**接口加载并执行算子。

       该方式下，每次调用**aclopExecute**接口执行算子，系统内部都会根据算子描述信息匹配内存中的模型。

- 您需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、调用**aclopCreateHandle**接口创建一个Handle、再调用**aclopExecWithHandle**接口加载并执行算子。

  该方式下，在调用**aclopCreateHandle**接口时，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，每次调用**aclopExecWithHandle**接口执行算子时，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。Handle使用结束后，需调用**aclopDestroyHandle**接口释放。

– 如果不是系统内置的算子，则需要用户先进行算子开发（请参见《TBE自定义算子开发指导》），再调用**•如果是系统内置的算子，但未被封装成ACL接口，...**中提供的方式执行算子。

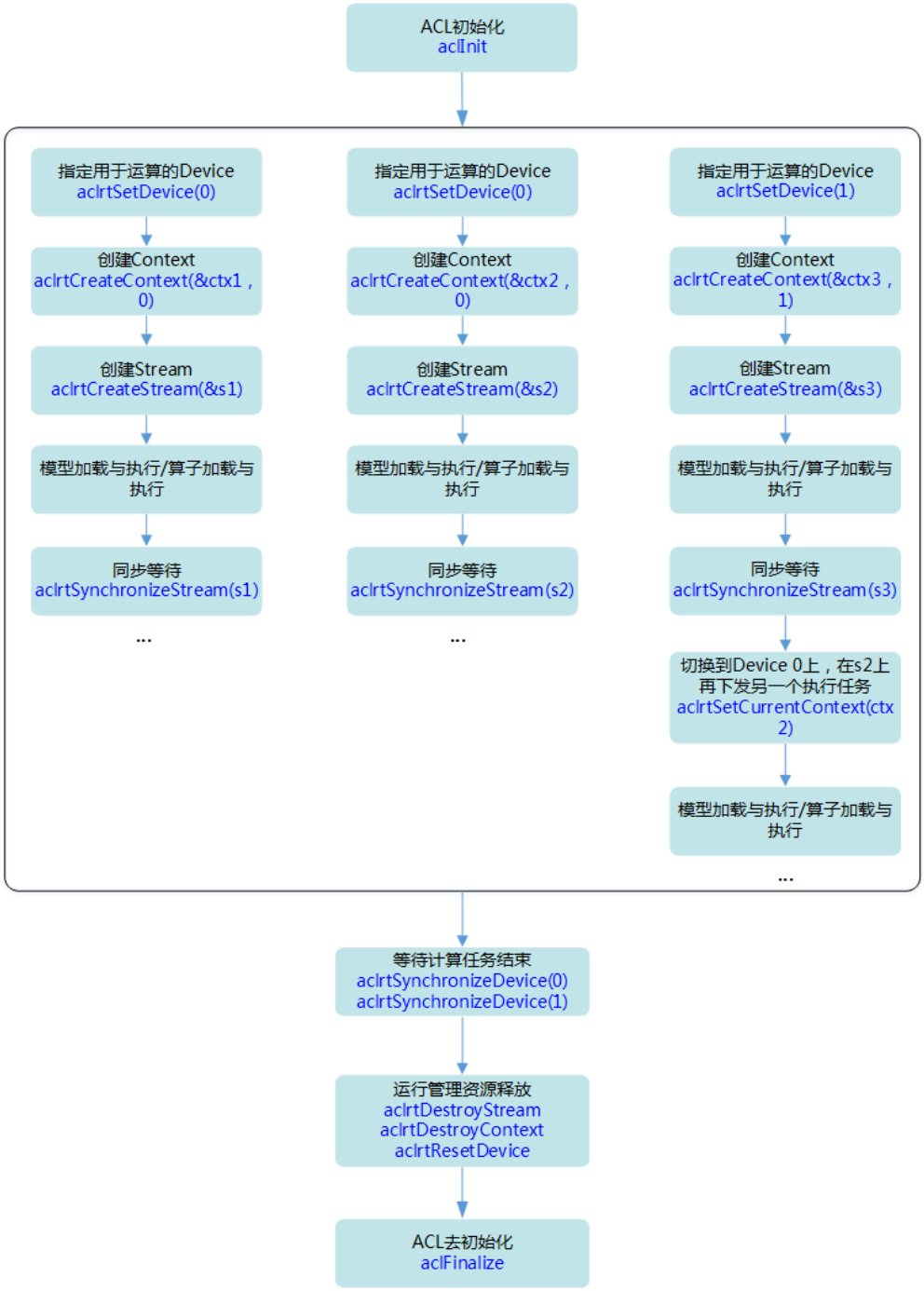
4. 调用**aclrtSynchronizeStream**接口阻塞应用运行，直到指定Stream中的所有任务都完成。
5. 调用**aclrtFree**接口释放内存。

   如果需要将Device上的算子执行结果数据传输到Host，则需要调用**aclrtMemcpy**接口（同步接口）或**aclrtMemcpyAsync**接口（异步接口）通过内存复制的方式实现数据传输，然后再释放内存。

3.7 同步等待

3.7.1 多 Device 场景

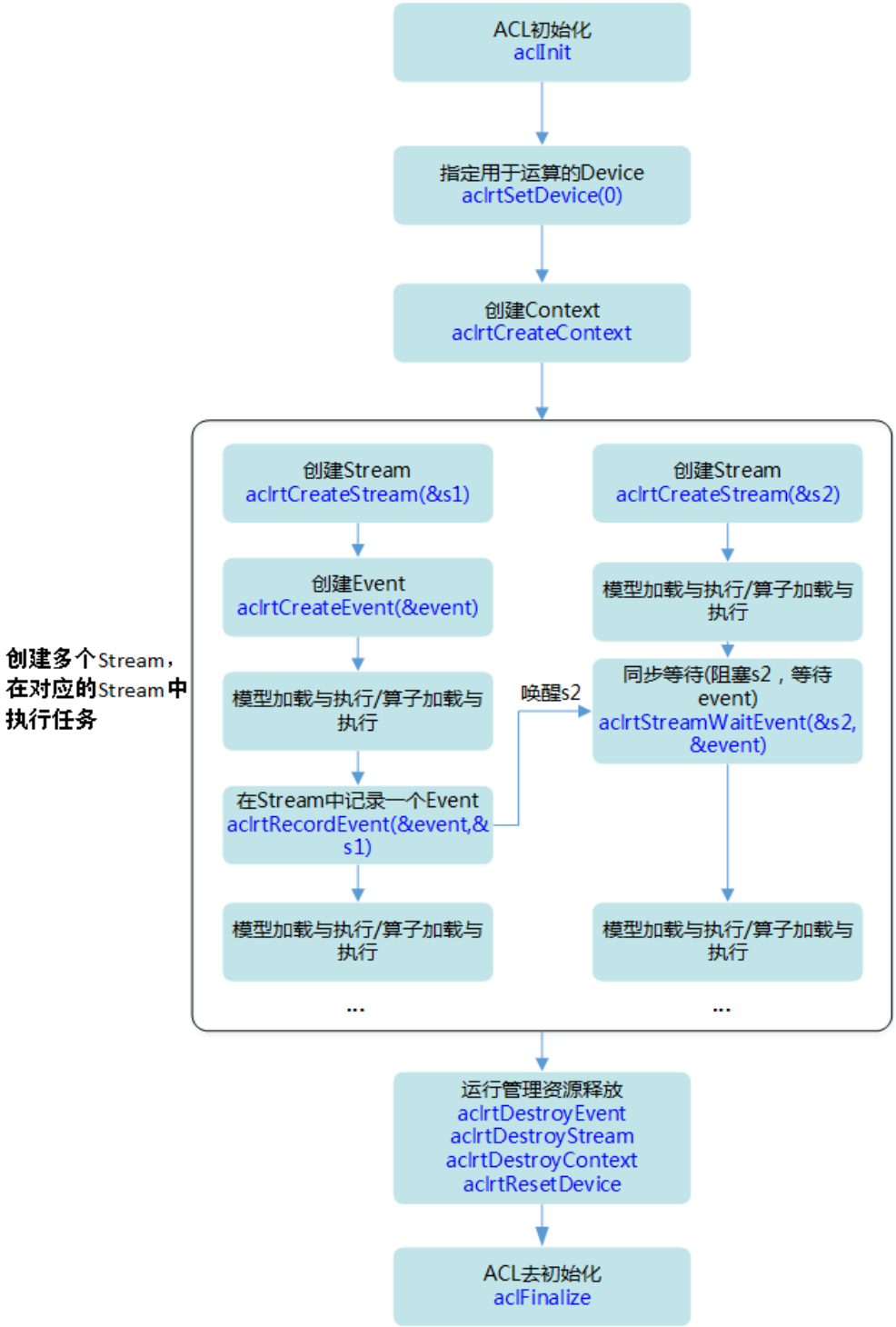
图 3-14 同步等待流程_多 Device 场景



- 在多Device时，利用Context切换（调用**aclrtSetCurrentContext**接口）来切换Device，比使用**aclrtSetDevice**接口效率高。
- 调用**aclrtSynchronizeDevice**接口等待Device上的计算任务结束。
- 模型加载的流程请参见**3.3 模型加载**，模型执行的流程请参见**3.5 模型推理**。
- 算子加载与执行的流程请参见**3.6 算子调用**。

3.7.2 多 Stream 场景

图 3-15 同步等待流程_多 Stream 场景



多Stream之间任务的同步等待可以利用Event实现，调用**aclrtStreamWaitEvent**接口阻塞指定Stream的运行，直到指定的Event完成。

模型加载的流程请参见**3.3 模型加载**，模型执行的流程请参见**3.5 模型推理**。

算子加载与执行的流程请参见**3.6 算子调用**。

3.7.3 Callback 场景

图 3-16 同步等待流程_Callback 场景_模型推理

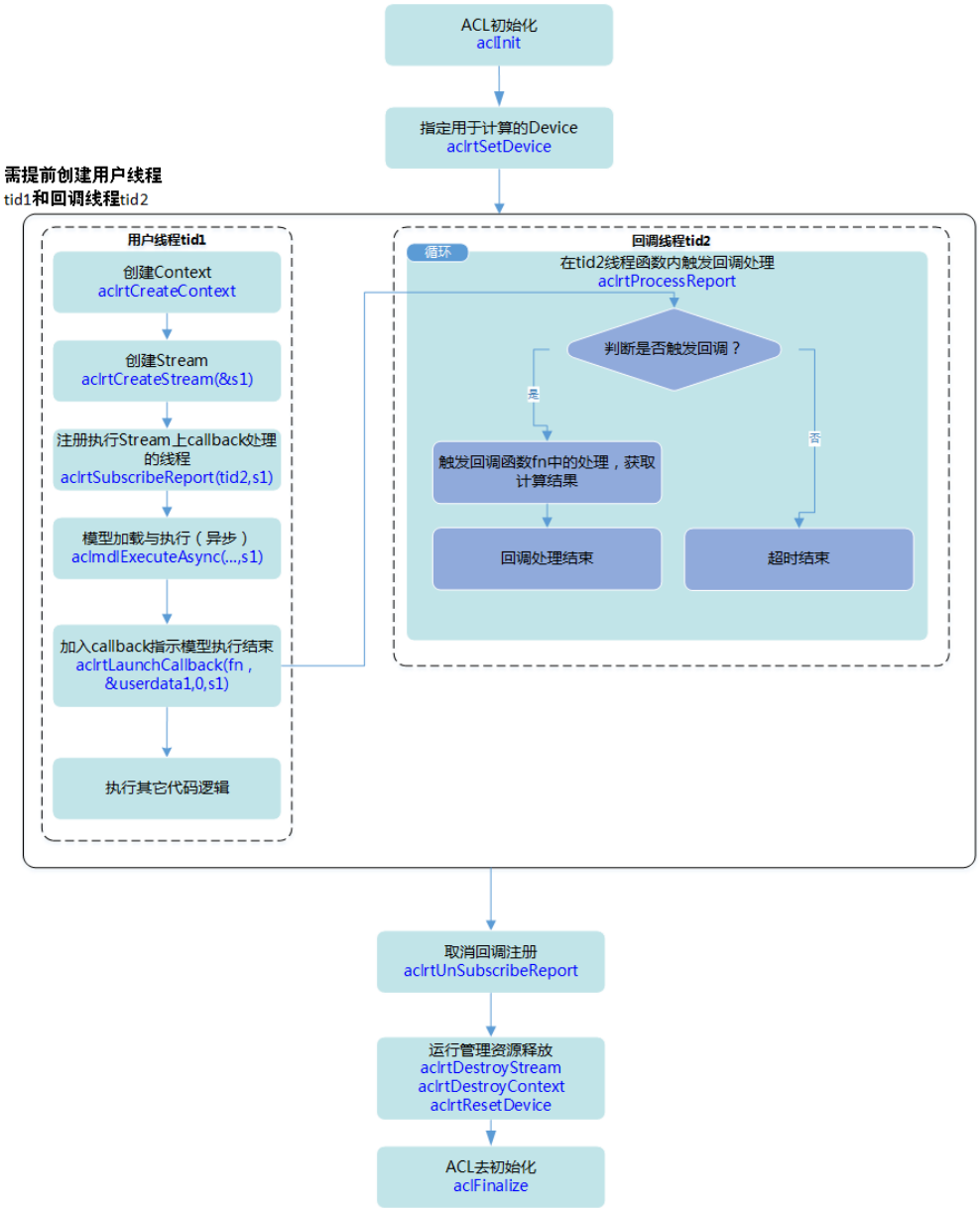
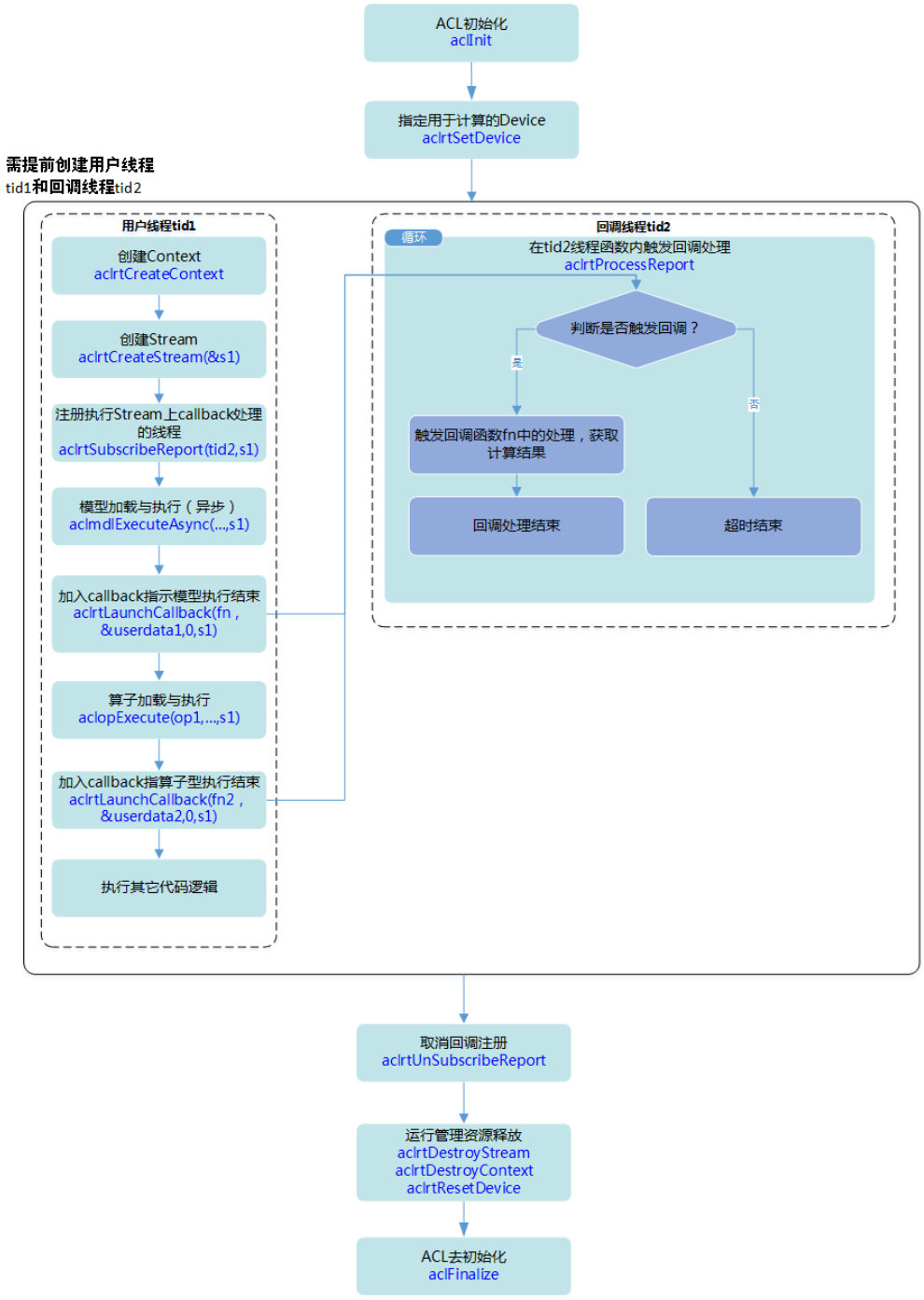


图 3-17 同步等待流程_Callback 场景_模型推理_算子调用



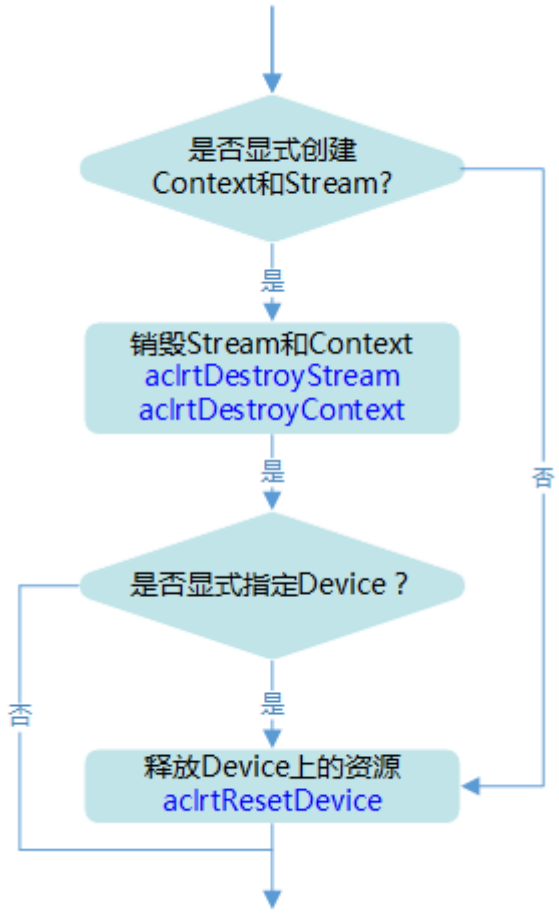
Callback流程中关键流程说明如下：

- 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。
- 线程需由用户提前创建，并自定义线程函数，在线程函数内调用**aclrtProcessReport**接口，等待指定时间后，触发 **回调函数需由用户提前创建。**中的回调函数。
- 调用**aclrtSubscribeReport**接口：指定处理Stream上回调函数的线程，线程与 **线程需由用户提前创建，并自定义线程函数，在线程...**中保持一致。

- 调用**aclrtLaunchCallback**接口：在Stream的任务队列中增加一个需要在Host/Device上执行的回调函数，回调函数与 **回调函数需由用户提前创建。**中保持一致。
- 调用**aclrtUnSubscribeReport**接口：取消线程注册（Stream上的回调函数不再由指定线程处理）。
- 如果是异步推理Callback场景，为确保Stream中所有任务都完成、模型推理的结果数据都经过Callback函数处理，在stream销毁前，需要调用一次**aclrtSynchronizeStream**接口。

3.8 运行管理资源释放

图 3-18 运行管理资源释放流程

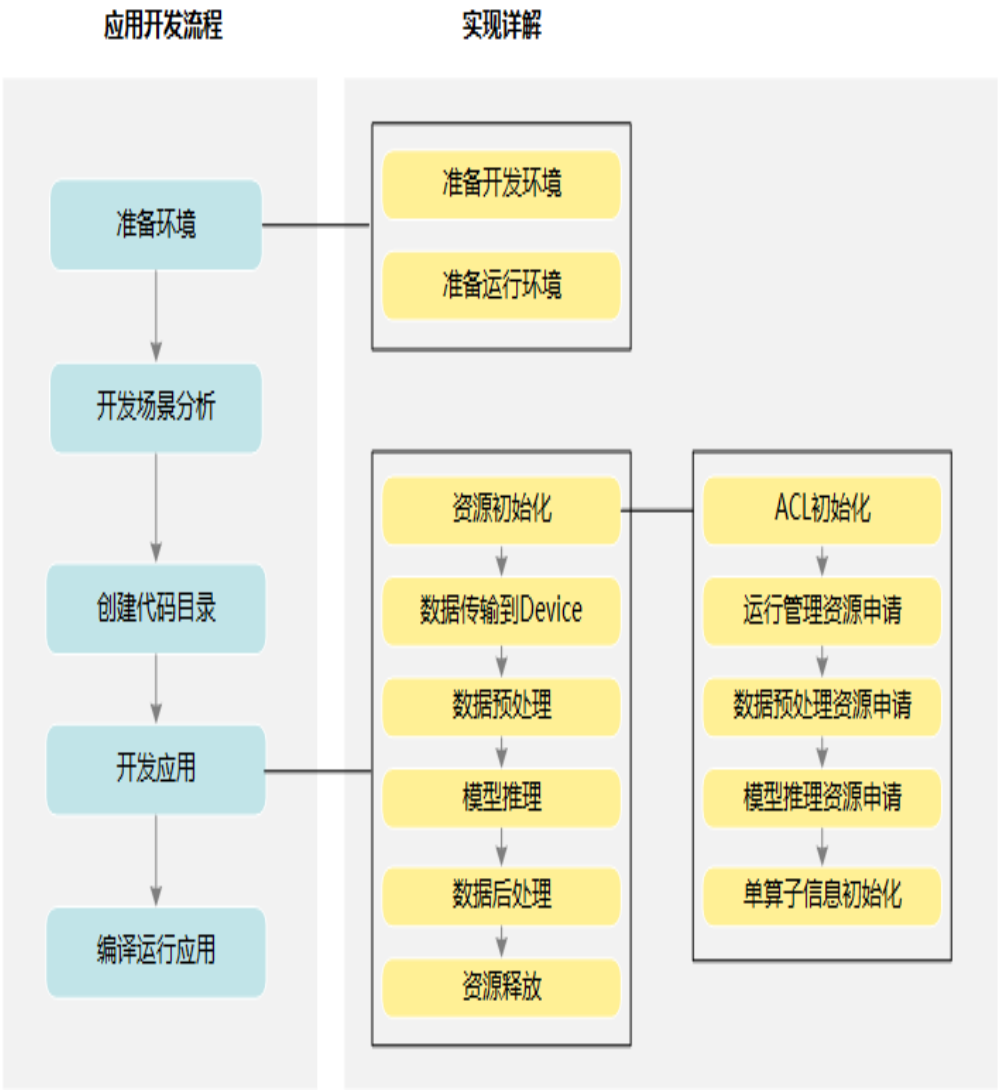


释放运行管理资源时，需按顺序依次释放：**Stream**、**Context**、**Device**。

- 显式创建Context和Stream时，需调用**aclrtDestroyStream**接口释放Stream，再调用**aclrtDestroyContext**接口释放Context。若显式调用**aclrtSetDevice**接口指定运算的Device时，还需调用**aclrtResetDevice**接口释放Device上的资源。
- 不显式创建Context和Stream时，仅需调用**aclrtResetDevice**接口释放Device上的资源。

4 开发流程

图 4-1 开发流程



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，请参见**6.1 开发场景分析**。

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，请参见**6.2 创建代码目录**。

4. 开发应用。

   a. 资源初始化，包括ACL初始化、运行管理资源申请、数据预处理资源申请、模型推理资源申请、单算子信息初始化等，请参见**6.3.1 资源初始化**。

      使用ACL接口开发应用时，必须先调用**aclInit**接口进行ACL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. 将数据从Host传输到Device，请参见**6.3.2 数据传输到Device**。

   c. 数据传输到Device后，若需要解码、缩放等操作，还需要进行数据预处理，输出YUV420 SP格式的图片，作为模型推理的输入。请参见**6.3.3 数据预处理（图片解码+缩放）**。

      数据预处理结束后，需及时释放相关资源。

   d. 同步执行模型推理。请参见**6.3.4 模型推理（单Batch+固定shape+静态AIPP+单模型）**。

      模型推理结束后，需及时释放相关资源。

   e. 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。请参见**6.3.5 数据后处理（调用单算子），回传结果到Host**。

      数据后处理结束后，需及时释放相关资源。

   f. 所有数据处理结束后，需及时释放运行管理资源，再调用**aclFinalize**接口实现ACL去初始化。

5. 编译运行应用，包括模型转换、编译代码、运行应用，请参见**6.4 编译运行应用**。

5 准备环境

5.1 准备环境（Ascend RC）

- 部署开发环境，请参见《Atlas 200 DK 使用指南》中的**安装开发环境**。
  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。
  - 从“Acllib组件的安装目录/acllib/include/acl”目录下获取调用ACL接口所需的头文件。
  - 从“Acllib组件的安装目录/acllib/lib64”目录下获取编译ACL接口所需的库文件。
- 部署运行环境，请参见《Atlas 200 DK 使用指南》中的**安装运行环境**。
  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。

6 开发首个应用

6.1 开发场景分析

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。

场景分析

您可从以下几方面入手分析该应用需包含哪些功能，这些功能点对应哪些ACL接口。

功能列表	实现分析	涉及的命令或关键接口
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。	• 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工具使用指导**》中的使用ATC工具转换模型。 • 模型加载 aclmdlLoadFromFileWithMem • 模型执行 aclmdlExecute • 模型卸载 aclmdlUnload

功能列表	实现分析	涉及的命令或关键接口
数据预处理（指解码/抠图/缩放/编码/色域转换等）	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。 但是输入图片的*.jpg、宽高是1024*683，与resnet-50网络的输入要求不一致，因此需要先将*.jpg图片解码成YUV420SP格式的图片，再缩放成宽高为224*224的图片，最后再通过色域转换将YUV420SP格式的图片转换成RGB格式的图片。	• 解码、缩放 – acldvppCreateChannelAsync – acldvppJpegDecodeAsync – acldvppVpcResizeAsync – acldvppDestroyChannelAsync • 色域转换 使用ATC（Ascend Tensor Compiler）工具转换模型时，通过--insert_op_conf参数配置色域转换，将色域转换的各参数值保存在*.om文件中，在模型执行时，进行色域转换。 请参见《**ATC工具使用指导**》中的AIPP配置。
数据后处理	处理模型推理的结果，通过调用Cast和ArgMaxD两个单算子从推理结果中查找每张图片最大置信度的类别标识。 调用算子Cast将推理结果的数据类型从float32转成float16，再调用ArgMaxD算子从推理结果中查找最大置信度的类别标识。	• 将Cast和ArgMaxD两个单算子的定义文件*.json编译成适配昇腾AI处理器的离线模型（*.om文件），用于验证单算子的运行。 请参见《**ATC工具使用指导**》中的使用ATC工具转换模型。 • 算子执行：aclopExecute
数据传输	涉及将Host的图片数据传输到Device，供Device处理数据时使用，完成数据处理后，Device需将处理结果传回Host。	内存拷贝：aclrtMemcpy

6.2 创建代码目录

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

您可以参考如下目录结构：

```
├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

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

├─── out
|    ├─── op_models       //该目录下存放算子om文件
```

```
|    ├──── App的可执行文件      //表示编译App代码生成的可执行文件

├──── src     //该目录下存放编译脚本、函数的实现文件
|    ├──── xxx.json          //系统初始化的配置文件
|    ├──── CMakeLists.txt          //编译脚本
|    ├──── xxx.cpp
```

6.3 开发应用

6.3.1 资源初始化

6.3.1.1 ACL 初始化

基本原理

您必须调用**aclInit**接口初始化ACL，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导**》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**aclInit**接口中传入空指针，示例如下：

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

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。

示例代码如下，您可以从**acl_dvpp_resnet50**样例的“src/sample_process.cpp”文件中查看完整样例代码。

```
#include "acl/acl.h"
//......
//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
//例如，编译出来的可执行文件存放在out目录下，此处的..就表示out目录的上一级目录
const char *aclConfigPath = "../src/acl.json";
aclError ret = aclInit(aclConfigPath);
//......
```

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acl.json配置文件中配置如下内容：

```
{
   "dump":{
      "dump_list":[
         {
            "model_name":"ResNet-50",
            "layer":[
                 "conv1conv1_relu",
                 "res2a_branch2ares2a_branch2a_relu",
                 "res2a_branch1",
                 "pool1"
            ]
         }
```

```
    ],
    "dump_path":"/MyApp20/dump",
        "dump_mode":"input"
  }
}
```

6.3.1.2 运行管理资源申请（单进程+单线程+单 Stream）

多线程、多Stream的场景请参见**7.1 Stream管理**。

基本原理

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。

- 关于Context和Stream
  - 显式创建Context时，调用**aclrtCreateContext**接口，此时需显式调用**aclrtDestroyContext**接口释放。
  - 显式创建Stream时，调用**aclrtCreateStream**接口，此时需显式调用**aclrtDestroyStream**接口释放。
  - 如果不显式创建Context和Stream，您可以使用**aclrtSetDevice**接口隐式创建的默认Context和默认Stream，但默认Context和默认Stream存在如下限制：
    - 一个Device对应一个默认Context，默认Context不能通过**aclrtDestroyContext**接口来释放。
    - 一个Device对应一个默认Stream，默认Stream不能通过**aclrtDestroyStream**接口来释放。默认Stream作为接口入参时，直接传NULL。
- 关于单进程、单线程、单Stream场景
  - 单进程：一个应用程序对应一个进程。
  - 单线程：不显式创建多个线程时，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，ACL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。

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。

示例代码如下，您可以从**acl_dvpp_resnet50**样例的“src/sample_process.cpp”文件中查看完整样例代码。

```
#include "acl/acl.h"
//.....

//1.指定运算的Device。
ret = aclrtSetDevice(deviceId_);

//2.显式创建一个Context，用于管理Stream对象。
ret = aclrtCreateContext(&context_, deviceId_);

//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。
ret = aclrtCreateStream(&stream_);
```



```
//.....
```

6.3.1.3 数据预处理资源申请

基本原理

在数据预处理（图片解码、缩放）之前，需创建图片数据处理的通道，并指定缩放算法为“最近邻插值”。

示例代码

示例代码如下，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。

您可以从**acl_dvpp_resnet50**样例的“src/sample_process.cpp”、“src/dvpp_process.cpp”文件中查看完整样例代码。

```
#include "acl/acl.h"
//.....

//1.创建图片数据处理通道时的通道描述信息，dvppChannelDesc_为acldvppChannelDesc类型。
dvppChannelDesc_ = acldvppCreateChannelDesc();

//2.异步创建图片数据处理的通道。
aclError ret = acldvppCreateChannel(dvppChannelDesc_);

//3.指定缩放算法，resizeConfig_为acldvppResizeConfig类型。
resizeConfig_ = acldvppCreateResizeConfig(0);
//.....
```

6.3.1.4 模型推理资源申请

基本原理

在模型推理前，需要从离线模型文件（适配昇腾AI处理器的离线模型）中加载模型数据到内存中。

- 关于模型转换

  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，请参见《**ATC工具使用指导**》中的使用ATC工具转换模型。

  若涉及色域转换（转换图像格式），在模型转换时，还需参见《**ATC工具使用指导**》中的AIPP配置进行配置。

- 关于获取模型占用的内存大小

  当由用户管理内存时，为确保内存不浪费，在申请工作内存、权值内存前，需要调用**aclmdlQuerySize**接口查询模型运行时所需工作内存、权值内存的大小。

- 关于模型加载

  加载模型数据分为以下4种方式：

  - **aclmdlLoadFromFile**：从文件加载离线模型数据，由系统内部管理内存。
  - **aclmdlLoadFromMem**：从内存加载离线模型数据，由系统内部管理内存。
  - **aclmdlLoadFromFileWithMem**：从文件加载离线模型数据，由用户自行管理模型运行的内存（包括工作内存和权值内存）。

  - **aclmdlLoadFromMemWithMem**：从内存加载离线模型数据，由用户自行管理模型运行的内存（包括工作内存和权值内存）。

- 关于模型的输入、输出数据

  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。

示例代码

示例代码如下，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。

示例代码如下，您可以从**acl_dvpp_resnet50**样例的“src/sample_process.cpp”、“src/model_process.cpp”文件中查看完整样例代码。

```
#include "acl/acl.h"
//......
//1.初始化变量。
//此处的..表示相对路径，相对可执行文件所在的目录
//例如，编译出来的可执行文件存放在out目录下，此处的..就表示out目录的上一级目录
const char* omModelPath = "../model/resnet50_aipp.om";
//......

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

//3.根据2中的工作内存大小，申请Device上模型执行的工作内存。
ret = aclrtMalloc(&modelMemPtr_, modelMemSize_, ACL_MEM_MALLOC_NORMAL_ONLY);

//4.根据2中的权值内存的大小，申请Device上模型执行的权值内存。
ret = aclrtMalloc(&modelWeightPtr_, modelWeightSize_, ACL_MEM_MALLOC_NORMAL_ONLY);

//5.加载离线模型文件（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(包括权值内存、工作内存)。
//模型加载成功，返回标识模型的ID。
ret = aclmdlLoadFromFileWithMem(modelPath, &modelId_, modelMemPtr_,
      modelMemSize_, modelWeightPtr_, modelWeightSize_);

//6.根据5中加载成功的模型的ID，获取该模型的描述信息。
//modelDesc_为aclmdlDesc类型。
modelDesc_ = aclmdlCreateDesc();
ret = aclmdlGetDesc(modelDesc_, modelId_);

//7.创建aclmdlDataset类型的数据，描述模型推理的输出。
//output_为aclmdlDataset类型
output_ = aclmdlCreateDataset();

//7.1 获取模型的输出个数.
size_t outputSize = aclmdlGetNumOutputs(modelDesc_);

//7.2 循环为每个输出申请内存，并将每个输出添加到aclmdlDataset类型的数据中.
for (size_t i = 0; i < outputSize; ++i) {
   size_t buffer_size = aclmdlGetOutputSizeByIndex(modelDesc_, i);
   void *outputBuffer = nullptr;
   aclError ret = aclrtMalloc(&outputBuffer, buffer_size, ACL_MEM_MALLOC_NORMAL_ONLY);
   aclDataBuffer* outputData = aclCreateDataBuffer(outputBuffer, buffer_size);
   ret = aclmdlAddDatasetBuffer(output_, outputData);
   }
//......
```


6.3.1.5 单算子信息初始化

基本原理

在数据后处理环节，需要调用算子Cast将推理结果的数据类型从float32转成float16，再调用ArgMaxD算子从推理结果中查找最大置信度的类别标识。因此需要提前将Cast和ArgMaxD两个单算子的定义文件*.json编译成适配昇腾AI处理器的离线模型（*.om文件），用于验证单算子的运行，请参见《**ATC工具使用指导**》。

转换后的*.om文件存放的路径应该与**aclopSetModelDir**接口中的路径保持一致。

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。

示例代码如下，您可以从**acl_dvpp_resnet50**样例的“src/sample_process.cpp”、“src/singleOp_process.cpp”文件中查看完整样例代码。

```
#include "acl/acl.h"
//......
//设置单算子模型文件所在的目录
//该目录相对可执行文件所在的目录，例如，编译出来的可执行文件存放在out目录下，此处就表示out/op_models目录
aclError ret = aclopSetModelDir("op_models");
//......
```

6.3.2 数据传输到 Device

基本原理

在Host将图片数据读入内存，再调用**aclrtMemcpy**接口同步将Host内存中的数据复制到Device内存中，供Device处理。若需实现异步数据复制，需使用**aclrtMemcpyAsync**接口。

- 若需要在Device进行数据预处理（jpg图片解码、缩放），则与数据预处理相关的内存必须通过**acldvppMalloc**接口申请。
- 若无需在Device进行数据预处理，则Device内存可以通过**aclrtMalloc**接口申请。

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。

示例代码如下，您可以从**acl_dvpp_resnet50**样例的“src/sample_process.cpp”、“src/dvpp_process.cpp”、“src/utils.cpp”文件中查看完整样例代码。

```
#include <iostream>
#include <fstream>
#include <cstring>
#include "acl/acl.h"
#include "acl/ops/acl_dvpp.h"
//......
//PicDesc是自定义的一个结构体，用于描述图片的路径和名称picName、图片宽width、图片高height。
PicDesc testPic[] = {
        { "../data/dog1_1024_683.jpg", 1024, 683 },
        { "../data/dog2_1024_683.jpg", 1024, 683 },
    };
```



```
//循环处理每张图片
for (size_t index = 0; index < sizeof(testPic) / sizeof(testPic[0]); ++index) {

    //1.自定义函数，调用C++标准库std::ifstream中的函数读取图片文件，并输出图片文件占用的内存大小inputHostBuffSize，返回值为图片文件在Host上的内存地址inputHostBuff。
    char* inputHostBuff = ReadBinFile(testPic.picName, inputHostBuffSize);

    //2.申请图片解码的输入内存
    void *inBufferDev = nullptr;
    uint32_t inBufferSize = inputHostBuffSize;
    aclError ret = acldvppMalloc(&inBufferDev, inBufferSize);

    //3.通过内存拷贝的方式，将Host数据传输到Device。
    ret = aclrtMemcpy(inBufferDev, inBufferSize, inputHostBuff, inputHostBuffSize, ACL_MEMCPY_HOST_TO_DEVICE);
    //TODO：数据预处理（图片解码+缩放）
}
//.....
```

6.3.3 数据预处理（图片解码+缩放）

基本原理

数据传输到Device后，经过数据预处理（解码、缩放）后，输出源图编码格式或YUV420 SP格式的图片，作为模型推理的输入。

- 解码
  - 针对不同的源图编码格式，解码后，输出如下格式的图片：
    - jpeg(YUV444SP) -> YUV444SP V在前U在后 、YUV420 SP V在前U在后、YUV420SP U在前V在后。
    - jpeg(YUV422SP) -> YUV422SP V在前U在后、YUV420SP V在前U在后、YUV420SP U在前V在后。
    - jpeg(YUV420SP) -> YUV420SP V在前U在后、YUV420SP U在前V在后。
    - jpeg(YUV400) -> YUV420SP，UV数据采用0 x 80填充。
  - 解码对输出图片的宽高有对齐要求：宽128对齐、高16对齐。
  - 解码后输出图片内存的计算公式：
    - YUV420SP：对齐后的宽*对齐后的高*3/2
    - YUV422SP：对齐后的宽*对齐后的高*2
    - YUV444SP：对齐后的宽*对齐后的高*3
- 缩放
  - 缩放对图片的宽、高有对齐要求（宽16对齐、高2对齐）。
  - 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。
- 资源释放
  - 数据预处理结束后，除数据预处理的输出内存（要作为模型推理的输入，需等模型推理结束后才能释放），其它相关资源需及时释放，例如数据预处理的输入内存、图片描述信息等。

- 与数据预处理相关的内存必须通过**acldvppMalloc**接口申请，通过**acldvppFree**接口释放。

关于其它数据预处理的功能，请参见**7.4 数据预处理**。

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。

示例代码如下，您可以从**acl_dvpp_resnet50**样例的“src/sample_process.cpp”、“src/dvpp_process.cpp”文件中查看完整样例代码。

```
#include "acl/acl.h"
#include "acl/ops/acl_dvpp.h"
//.....
//初始化变量，请参见样例代码

//循环处理每张图片，testPic是自定义的一个结构体，用于描述图片的名称picName、图片宽width、图片高height。
for (size_t index = 0; index < sizeof(testPic) / sizeof(testPic[0]); ++index) {

   //1. 申请缩放输出内存resizeOutBufferDev_
   //内存大小resizeOutBufferSize_根据计算公式得出（YUV420SP：对齐后的宽*对齐后的高*3/2）
   aclError ret = acldvppMalloc(&resizeOutBufferDev_, resizeOutBufferSize_)

   //2. 申请解码输出内存，并创建解码输出图片的描述信息

   //2.1 申请解码输出内存decodeOutDevBuffer_，内存大小decodeOutBufferSize根据计算公式得出（YUV420SP：对齐后的宽*对齐后的高*3/2）
   ret = acldvppMalloc(&decodeOutDevBuffer_, decodeOutBufferSize)

   //2.2 创建解码输出图片的描述信息，设置各属性值
   decodeOutputDesc_ = acldvppCreatePicDesc();
   acldvppSetPicDescData(decodeOutputDesc_, decodeOutDevBuffer_);
   acldvppSetPicDescFormat(decodeOutputDesc_, PIXEL_FORMAT_YUV_SEMIPLANAR_420);
   acldvppSetPicDescWidth(decodeOutputDesc_, inputWidth_);
   acldvppSetPicDescHeight(decodeOutputDesc_, inputHeight_);
   acldvppSetPicDescWidthStride(decodeOutputDesc_, decodeOutWidthStride);
   acldvppSetPicDescHeightStride(decodeOutputDesc_, decodeOutHeightStride);
   acldvppSetPicDescSize(decodeOutputDesc_, decodeOutBufferSize);


   //3. 执行异步解码，再调用aclrtSynchronizeStream接口阻塞Host运行，直到指定Stream中的所有任务都完成
   ret = acldvppJpegDecodeAsync(dvppChannelDesc_, inDevBuffer_, inDevBufferSize_, decodeOutputDesc_, stream_);

   ret = aclrtSynchronizeStream(stream_);

   //4. 解码结束后，释放资源，包括解码输出图片的描述信息
   acldvppDestroyPicDesc(decodeOutputDesc_);

   //5. 创建缩放输入图片的描述信息，并设置各属性值
   resizeInputDesc_ = acldvppCreatePicDesc();
   acldvppSetPicDescData(resizeInputDesc_, decodeOutDevBuffer_);
   acldvppSetPicDescFormat(resizeInputDesc_, PIXEL_FORMAT_YVU_SEMIPLANAR_420);
   acldvppSetPicDescWidth(resizeInputDesc_, inputWidth_);
   acldvppSetPicDescHeight(resizeInputDesc_, inputHeight_);
   acldvppSetPicDescWidthStride(resizeInputDesc_, jpegOutWidthStride);
   acldvppSetPicDescHeightStride(resizeInputDesc_, jpegOutHeightStride);
   acldvppSetPicDescSize(resizeInputDesc_, jpegOutBufferSize);

   //6. 创建缩放输出图片的描述信息，resizeOutBufferDev_内存中的数据作为模型推理的输入
   resizeOutputDesc_ = acldvppCreatePicDesc();
   acldvppSetPicDescData(resizeOutputDesc_, resizeOutBufferDev_);
   acldvppSetPicDescFormat(resizeOutputDesc_, PIXEL_FORMAT_YUV_SEMIPLANAR_420);
   acldvppSetPicDescWidth(resizeOutputDesc_, modelInputWidth_);
```



```
    acldvppSetPicDescHeight(resizeOutputDesc_, modelInputHeight_);
    acldvppSetPicDescWidthStride(resizeOutputDesc_, resizeOutputWidthStride);
    acldvppSetPicDescHeightStride(resizeOutputDesc_, resizeOutputHeightStride);
    acldvppSetPicDescSize(resizeOutputDesc_, resizeOutBufferSize_);

    //7. 执行异步缩放，再调用aclrtSynchronizeStream接口阻塞Host运行，直到指定Stream中的所有任务都完成
    ret = acldvppVpcResizeAsync(dvppChannelDesc_, resizeInputDesc_,
        resizeOutputDesc_, resizeConfig_, stream_);
    ret = aclrtSynchronizeStream(stream_);

    //8. 释放资源
    //8.1 缩放结束后，释放资源，包括解码的输出内存、缩放输入图片的描述信息、输出图片描述信息
    (void)acldvppFree(decodeOutDevBuffer_);
    acldvppDestroyPicDesc(resizeInputDesc_);
    acldvppDestroyPicDesc(resizeOutputDesc_);

    //8.2 释放其它资源，包括缩放配置、图片数据处理的通道及其描述信息
    acldvppDestroyResizeConfig(resizeConfig_);
    aclError ret = acldvppDestroyChannel(dvppChannelDesc_);
    acldvppDestroyChannelDesc(dvppChannelDesc_);

    //8.3释放通过acldvppMalloc接口申请的数据预处理的输入内存picDevBuffer
    (void)acldvppFree(picDevBuffer);

    //TODO: 模型推理
}
//.....
```

6.3.4 模型推理（单 Batch+固定 shape+静态 AIPP+单模型）

基本原理

同步执行模型推理。模型推理结束后，需及时释放相关资源。

- 关于模型推理

  调用**aclmdlExecute**接口实现同步模型推理，该接口的入参之一是模型ID，在**模型推理资源申请**阶段，如果加载模型成功，则会返回标识模型的ID。

- 关于资源释放
  - 由于数据预处理的输出作为模型推理的输入，在模型推理结束后，需及时调用**acldvppFree**接口释放内存。
  - 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。在模型推理结束后，需及时调用**aclDestroyDataBuffer**接口和**aclmdlDestroyDataset**接口释放描述模型输入的数据，且先调用**aclDestroyDataBuffer**接口，再调用**aclmdlDestroyDataset**接口。如果存在多个输入、输出，需调用多次**aclDestroyDataBuffer**接口。
  - 在模型推理结束后，还需要通过**aclmdlUnload**接口卸载模型。

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。

示例代码如下，您可以从**acl_dvpp_resnet50**样例的“src/sample_process.cpp”、“src/model_process.cpp”文件中查看完整样例代码。

```
#include "acl/acl.h"
//....
```



```
//循环处理每张图片，testPic是自定义的一个结构体，用于描述图片的名称picName、图片宽width、图片高height。
for (size_t index = 0; index < sizeof(testPic) / sizeof(testPic[0]); ++index) {
    //1. 创建模型推理的输入数据
    //创建aclmdlDataset类型的数据，用于描述模型推理时的输入数据（图片数据集），输入的内存地址、内存大小用aclDataBuffer类型的数据来描述
    aclmdlDataset *input_ = aclmdlCreateDataset();
    //示例中的模型只有1个输入，将数据预处理（缩放）的输出内存作为模型推理的输入，inputDataBuffer表示数据预处理的输出内存，bufferSize表示内存大小
    aclDataBuffer* inputData = aclCreateDataBuffer(inputDataBuffer, bufferSize);
    aclError ret = aclmdlAddDatasetBuffer(input_, inputData);

    //2. 执行模型推理
    //modelId_表示模型ID，在“模型推理资源初始化”时成功加载模型后，会返回标识模型的ID
    //output_表示模型推理的输出数据，在“模型推理资源初始化”时已定义
    ret = aclmdlExecute(modelId_, input_, output_)

    //3. 模型推理结束后，释放数据预处理的输出内存dvppOutputBuffer
    acldvppFree(dvppOutputBuffer);

    //4. 释放资源
    //如果模型存在多个输入，则需要多次调用aclDestroyDataBuffer释放描述每个输入的aclmdlDataset类型数据
    aclDestroyDataBuffer(inputData);
    aclmdlDestroyDataset(input_);

    //TODO: 数据后处理
}
//.....
```

6.3.5 数据后处理（调用单算子），回传结果到 Host

数据后处理（图片分类）

模型推理结束后，在数据后处理阶段，处理模型推理的结果。数据后处理结束后，需及时释放相关资源。

当前示例中，使用Cast算子（将推理结果的数据类型从float32转成float16）和ArgMaxD算子（从推理结果中查找最大置信度的类别标识）实现数据后处理。

- Cast算子被封装成ACL接口，因此在使用时，将算子的输入输出Tensor描述、算子输入输出数据的内存等信息作为**aclopCast**的入参，直接调用**aclopCast**接口加载并执行算子。
- ArgMaxD算子没有被封装成ACL接口，因此在使用时，必须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、明确算子类型名称、调用**aclopExecute**接口加载并执行算子。

关于其它单算子调用的方式，请参见**7.7 单算子调用**。

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。

示例代码如下，您可以从**acl_dvpp_resnet50**样例的“src/sample_process.cpp”、“src/singleOp_process.cpp”文件中查看完整样例代码。

```
#include "acl/acl.h"
//.....
//循环处理每张图片，testPic是自定义的一个结构体，用于描述图片的名称picName、图片宽width、图片高height。
for (size_t index = 0; index < sizeof(testPic) / sizeof(testPic[0]); ++index) {
    //1. 在数据后处理前，先获取模型推理的输出，modelOutput表示模型推理的输出
```



```
    aclDataBuffer inputBuffer_[0] = aclmdlGetDatasetBuffer(modelOutput, 0);

    //2. 自定义函数RunSigleOpCast，构造Cast算子的输入输出Tensor描述、申请存放算子输出数据的内存devBufferCast_、调用aclopCast接口加载并执行算子
    Result ret = RunSigleOpCast();

    //3. 自定义函数RunSigleOpArgMaxD，构造ArgMaxD算子的输入输出Tensor、输入输出Tensor描述、算子属性、申请存放算子输出数据的内存devBufferArgMaxD_、调用aclopExecute接口加载并执行算子
    ret = RunSigleOpArgMaxD();

    //4.将ArgMaxD算子的输出数据回传到Host
    //4.1 根据ArgMaxD算子输出数据的大小，申请Host上的内存
    void* hostBuffer = nullptr;
    aclError ret = aclrtMallocHost(&hostBuffer, tensorSizeArgMaxD_);

    //4.2 将ArgMaxD算子的输出数据从Device复制到Host
    ret = aclrtMemcpy(hostBuffer, tensorSizeArgMaxD_, devBufferArgMaxD_,
        tensorSizeArgMaxD_, ACL_MEMCPY_DEVICE_TO_HOST);

    //4.3 在终端窗口显示最大置信度的类别标识
    int32_t* index = static_cast<int32_t*>(hostBuffer);
    INFO_LOG("---> index of classification result is %d", *index);

    //5 释放资源
    //5.1 释放Host的内存
    aclrtFreeHost(hostBuffer);

    //5.2 释放Device上存放算子输出数据的内存
    (void)aclrtFree(devBufferCast_);
    (void)aclrtFree(devBufferArgMaxD_);

    //5.3 释放aclDataBuffer类型数据（用于描述算子输出数据）
    (void)aclDestroyDataBuffer(outputBufferCast_[0]);
    (void)aclDestroyDataBuffer(outputBufferArgMaxD_[0]);
//......
}
//TODO：释放运行管理资源
```

6.3.6 运行管理资源释放与 ACL 去初始化

基本原理

Host、Device上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，需要按顺序释放，先释放Stream，再释放Context，最后再释放Device。

- 如果使用默认Context，则不能通过**aclrtDestroyContext**接口来释放。
- 如果使用默认Stream，则不能通过**aclrtDestroyStream**接口来释放。
- 默认Context、默认Stream，是在调用**aclrtResetDevice**接口后自动释放。
- 调用**aclFinalize**接口实现ACL去初始化。

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。

示例代码如下，您可以从**acl_dvpp_resnet50**样例的“src/sample_process.cpp”文件中查看完整样例代码。

```
#include "acl/acl.h"
//......
```



```
aclError ret = aclrtDestroyStream(stream_);
ret = aclrtDestroyContext(context_);
ret = aclrtResetDevice(deviceId_);
ret = aclFinalize();
//.....
```

6.4 编译运行应用

编译运行应用的步骤，请参见**acl_dvpp_resnet50**样例的**9.4 基于Caffe ResNet-50网络实现图片分类（包含图片解码+缩放）**。

相关注意点如下：

- 模型转换，详细说明请参见《**ATC工具使用指导**》中的约束及参数说明和使用ATC工具转换模型。
- 编译代码时，编译脚本的修改点如下。

  您可以从**acl_dvpp_resnet50**样例中获取编译脚本CMakeLists.txt，在该编译脚本的基础上修改如下参数。

  - include_directories：添加头文件所在的目录。

    示例如下：

    ```
    include_directories(
        directoryPath1
        directoryPath2
    )
    ```

  - link_directories：添加库文件所在的目录。

    示例如下：

    ```
    link_directories(
        directoryPath3
        directoryPath4
    )
    ```

  - add_executable：修改可执行文件的名称（例如：*main*）、添加*.cpp文件所在的目录。

    示例如下：

    ```
    add_executable(main
        directoryPath5
        directoryPath6)
    ```

  - target_link_libraries：修改可执行文件的名称（与 **add_executable：修改可执行文件的...**中保持一致）、添加可执行文件依赖的库文件。

    示例如下：

    ```
    target_link_libraries(main
        ascendcl
        libName1
        libName2)
    ```

    依赖的库文件与接口所在的头文件有关，具体对应关系如下：

    表 6-1 头文件与库文件的对应关系

头文件	库文件
Acllib组件的安装目录/acllib/include/acl/acl_base.h	Acllib组件的安装目录/acllib/lib64/stub/libascendcl.so

头文件	库文件
Acllib组件的安装目录/acllib/include/acl/acl.h	
Acllib组件的安装目录/acllib/include/acl/acl_mdl.h	
Acllib组件的安装目录/acllib/include/acl/acl_op.h	
Acllib组件的安装目录/acllib/include/acl/acl_rt.h	
Acllib组件的安装目录/acllib/include/acl/ops/acl_cblas.h	Acllib组件的安装目录/acllib/lib64/stub/libacl_cblas.so
Acllib组件的安装目录/acllib/include/acl/ops/acl_dvpp.h	Acllib组件的安装目录/acllib/lib64/stub/libacl_dvpp.so
Acllib组件的安装目录/acllib/include/acl/ops/acl_fv.h （当前不支持引用该头文件中的接口。）	Acllib组件的安装目录/acllib/lib64/libacl_retr.so

说明

- 使用“Acllib组件的安装目录/acllib/lib64/stub”目录下的*.so库，是为了编译基于ACL接口的代码逻辑时，不依赖其它组件（例如Driver）的任何*.so库。因此在使用cmake编译命令时，请务必将DCMAKE_SKIP_RPATH设置为TRUE，代表不会将rpath路径（即Acllib组件的安装目录/acllib/lib64/stub）添加到编译生成的可执行文件中去。
- 编译通过后，运行应用时，通过配置环境变量，应用会链接到运行环境上“Acllib组件的安装目录/acllib/lib64”目录下的*.so库，运行时会自动链接到依赖其它组件的*.so库。
- 编译基于ACL接口的代码逻辑时，请按照引用的头文件，依赖对应的so文件，引用多余的so文件可能导致后续版本升级时存在兼容性问题。

– 编译选项，修改可执行文件的名称（与**•add_executable：修改可执行文件的...**中保持一致），以及可执行文件的安装目录。

示例如下，表示*main*安装在${CMAKE_INSTALL_PREFIX}/out目录下，${CMAKE_INSTALL_PREFIX}变量定义的路径是相对路径，相对cmake命令执行的路径：

```
set(CMAKE_INSTALL_PREFIX "../../../")
set(CMAKE_OUTPUT_DIR "out")

install(TARGETS main DESTINATION ${CMAKE_OUTPUT_DIR})
```

说明

关于cmake参数的详细介绍，请参见**https://cmake.org/cmake/help/latest/guide/tutorial/index.html**，选择对应的版本后查看参数。

- 运行可执行文件时，需将ACL初始化配置文件（acl.json）所在的目录、可执行文件所在的目录、测试图片所在的目录、*.om文件所在的目录都上传到Host的同一个目录下。

如果在**ACL初始化**阶段，在**aclInit**接口中传入空指针，则无需将ACL初始化配置文件（acl.json）所在的目录上传到Host。

7 开发关键功能的详细介绍

7.1 Stream 管理

7.1.1 原理介绍

在ACL中，Stream是一个任务队列，应用程序通过Stream来管理任务的并行，一个Stream内部的任务保序执行，即Stream根据发送过来的任务依次执行；不同Stream中的任务并行执行。一个默认Context下会挂一个默认Stream，如果不显式创建Stream，可使用默认Stream。默认Stream作为接口入参时，直接传NULL。

7.1.2 单线程单 Stream

```
#include "acl/acl.h"
//.....
//显式创建一个Stream
aclrtStream stream;
aclrtCreateStream(&stream);

//调用触发任务的接口，传入stream参数
aclrtMemcpyAsync(devPtr, devSize, hostPtr, hostSize, ACL_MEMCPY_HOST_TO_DEVICE, stream);
//调用aclrtSynchronizeStream接口，阻塞应用程序运行，直到指定Stream中的所有任务都完成。
aclrtSynchronizeStream(stream);

//Stream使用结束后，显式销毁Stream
aclrtDestroyStream(stream);
//.....
```


7.1.3 多线程多 Stream

```
#include "acl/acl.h"
//.....
void runThread(aclrtStream stream) {
    int32_t deviceId =0 ;
    aclrtContext context;

    //如果只创建了一个Context，线程默认将这个Context作为线程当前的Context；
    //如果是多个Context，则需要调用aclrtSetCurrentContext接口设置当前线程的Context
    aclrtCreateContext(&context, deviceId);
    aclrtCreateStream(&stream);

    //调用触发任务的接口
    //....

    //释放资源
    aclrtDestroyStream(stream);
    aclrtDestroyContext(context);

}

aclrtStream stream1;
aclrtStream stream2;
//创建2个线程，每个线程对应一个Stream
std::thread t1(runThread, stream1);
std::thread t2(runThread, stream2);
//显式调用join函数确保结束线程
t1.join();
t2.join();
```

7.2 同步等待

7.2.1 原理介绍

ACL提供以下几种同步机制：

- Event的同步等待：调用**aclrtSynchronizeEvent**接口，阻塞应用程序运行，等待Event完成。
- Stream内任务的同步等待：调用**aclrtSynchronizeStream**接口，阻塞应用程序运行，直到指定Stream中的所有任务都完成。
- Stream间任务的同步等待：调用**aclrtStreamWaitEvent**接口，阻塞指定Stream的运行，直到指定的Event完成。支持多个Stream等待同一个Event的场景。
- Device的同步等待：调用**aclrtSynchronizeDevice**接口，阻塞应用程序运行，直到正在运算中的Device完成运算。多Device场景下，调用该接口等待的是当前Context对应的Device。

7.2.2 关于 Event 的同步等待

```
#include "acl/acl.h"
//.....
//创建一个Event
aclrtEvent event;
aclrtCreateEvent(&event);

//创建一个Stream
aclrtStream stream;
aclrtCreateStream(&stream);

//stream末尾添加了一个event
```



```
aclrtRecordEvent(event, stream);

//阻塞应用程序运行，等待event发生，也就是stream执行完成
//stream完成后产生event，唤醒执行应用程序的控制流，开始执行程序
aclrtSynchronizeEvent(event);

//显式销毁资源
aclrtDestroyStream(stream);
aclrtDestroyEvent(event);
//......
```

7.2.3 关于 Stream 内任务的同步等待

```
#include "acl/acl.h"
//......
//显式创建一个Stream
aclrtStream stream;
aclrtCreateStream(&stream);

//调用触发任务的接口，传入stream参数
aclrtMemcpyAsync(devPtr, devSize, hostPtr, hostSize, ACL_MEMCPY_HOST_TO_DEVICE, stream);
//调用aclrtSynchronizeStream接口，阻塞应用程序运行，直到指定Stream中的所有任务都完成。
aclrtSynchronizeStream(stream);

//Stream使用结束后，显式销毁Stream
aclrtDestroyStream(stream);
//......
```

7.2.4 关于 Stream 间任务的同步等待

```
#include "acl/acl.h"
//......
//创建一个Event
aclrtEvent event;
aclrtCreateEvent(&event);

//创建两个Stream
aclrtStream s1;
aclrtStream s2;
aclrtCreateStream(&s1);
aclrtCreateStream(&s2);

//在s1末尾添加了一个event
aclrtRecordEvent(event, s1);

//阻塞s2运行，直到指定event发生，也就是s1执行完成
//s1完成后，唤醒s2，继续执行s2的任务
aclrtStreamWaitEvent(s2, event);

//显式销毁资源
aclrtDestroyStream(s2);
aclrtDestroyStream(s1);
aclrtDestroyEvent(event);
//......
```

7.2.5 关于 Device 的同步等待

```
aclrtSynchronizeDevice(); // 阻塞应用程序运行，直到正在运算中的Device完成运算
```

7.3 数据传输

7.3.1 原理介绍

- 数据传输可以通过内存复制的方式实现，分为同步内存复制、异步内存复制：
  - 同步内存复制：调用**aclrtMemcpy**接口。
  - 异步内存复制：调用**aclrtMemcpyAsync**接口，再调用**aclrtSynchronizeStream**接口实现Stream内任务的同步等待。
- 内存复制前，需要提前申请内存，分为Host上的内存申请、Device上的内存申请：
  - Host上的内存，可以用C++标准库中的new、malloc接口申请内存，也可以使用ACL提供的**aclrtMallocHost**接口申请内存。
  - Device上的内存，使用ACL提供的**aclrtMalloc**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**acldvppMalloc**接口申请内存。
- 当同一个应用的可执行文件既支持在Host执行，也支持在Device上执行时，在编程时需要调用**aclrtGetRunMode**接口获取软件栈的运行模式，根据运行模式来判断后续的内存申请接口调用逻辑。
  a. 如果应用的可执行文件在Host上执行，则可能涉及Host与Device之间的数据传输，需要调用**aclrtMemcpy**接口（同步接口）或**aclrtMemcpyAsync**接口（异步接口）通过内存复制的方式实现数据传输。
  b. 如果应用的可执行文件在Device上执行，则不涉及Host与Device之间的数据传输。

  说明

  Ascend RC场景下，Host和Device都部署在开发者板上，也不涉及Host与Device之间的数据传输。

7.3.2 Host 内的数据传输

示例代码如下：

```
#include "acl/acl.h"
//......

//hostPtrA表示Host上目的内存地址指针，hostSizeA表示目的内存的大小
//hostPtrB表示Host上源内存地址指针，count表示源内存的大小
aclrtMemcpy(hostPtrA, hostSizeA, hostPtrB, count, ACL_MEMCPY_HOST_TO_HOST);

//......
```

7.3.3 Device 内的数据传输

示例代码如下：

```
#include "acl/acl.h"
//......

//devPtrA表示Device上目的内存地址指针，devSizeA表示目的内存的大小
//devPtrB表示Device上源内存地址指针，count表示源内存的大小
aclrtMemcpy(devPtrA, devSizeA, devPtrB, count, ACL_MEMCPY_DEVICE_TO_DEVICE);

//......
```

7.3.4 从 Host 到 Device 的数据传输

示例代码如下：


```
#include "acl/acl.h"
//......

//devPtr表示Device上目的内存地址指针，devSize表示目的内存的大小
//hostPtr表示Host上源内存地址指针，count表示源内存的大小
aclrtMemcpy(devPtr, devSize, hostPtr, count, ACL_MEMCPY_HOST_TO_DEVICE);

//......
```

7.3.5 从 Device 到 Host 的数据传输

示例代码如下：

```
#include "acl/acl.h"
//......

//hostPtr表示Host上目的内存地址指针，hostSize表示目的内存的大小
//devPtr表示源Device上内存地址指针，count表示源内存的大小
aclrtMemcpy(hostPtr, hostSize, devPtr, count, ACL_MEMCPY_DEVICE_TO_HOST);

//......
```

7.4 数据预处理

7.4.1 总体说明

- 关于异步接口

  对于本章介绍的异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功，对于有依赖的接口，为确保能按序执行任务，建议用户在多个接口中指定同一个stream，因为同一个stream中的任务按接口调用顺序执行。

  在调用异步口对图片进行解码、抠图、缩放等操作时，如果任务之间有依赖，一定要调用**aclrtSynchronizeStream**接口确保在同一个Stream中的任务按序执行。

  调用异步接口后，不能马上释放资源，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保Device侧任务执行完成后才能释放。

- 关于内存申请/释放

  实现媒体数据处理的VPC功能、JPEGD功能、JEPGE等功能前，若需要申请Device上的内存存放输入或输出数据，需调用**acldvppMalloc**申请内存、调用**acldvppFree**接口释放内存。

- 支持的媒体数据处理功能如下：
  - VPC（vision preprocessing core）功能：支持对图片做抠图、缩放、叠加、拼接、格式转换等操作，详细描述请参见**7.4.2.1 功能及约束说明**。
  - JPEGD（JPEG decoder）功能：将.jpg、.jpeg、.JPG、.JPEG图片解码成YUV格式图片，详细描述请参见**7.4.3.1 功能及约束说明**。
  - JPEGE（JPEG encoder）功能：将YUV格式图片编码成.jpg图片，详细描述请参见**7.4.4.1 功能及约束说明**。
  - VDEC（video decoder）功能：实现视频的解码，详细描述请参见**7.4.5.1 功能及约束说明**。
  - VENC（video encoder）功能：实现视频的编码，详细描述请参见**7.4.6.1 功能及约束说明**。
- 除**acldvppMalloc**接口、**acldvppFree**接口外，**媒体数据处理**章节中的其它接口只能在Host上调用，不能在Device上调用。

7.4.2 VPC 功能

7.4.2.1 功能及约束说明

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，VPC的处理方式可分为：
    - 非8K缩放，用于处理“**widthStride**在32~4096（包括4096）范围内，**heightStride**在6~4096”的输入图片，不同格式的输入图片，widthStride的取值范围不同，详细描述参见**表7-1**。
    - 8K缩放，用于处理“widthStride在4096~8192范围内或heightStride在4096~8192范围内（不包括4096）”的输入图片。
  - 从是否抠多张图的维度，可分为单图裁剪缩放（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。
- **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。
- **格式转换**
  - 支持RGB格式、YUV格式之间的格式转换，目前的输入图片格式、输出图片格式，请参见**表7-1**。
  - 图像灰度化，对输出图像数据只取Y分量的数据。

约束说明

- 关于VPC输入/输出的约束

表 7-1 关于 VPC 输入/输出的约束

VPC输入/输出	图片分辨率	图片格式	内存要求	宽stride、高stride对齐要求
VPC输入	• 非8K缩放： – YUV400SP/YUV420SP/YUV422SP/yuv444sp：widthStride在32~4096（包括4096）范围内，heightStride在6~4096。 – YUV422packed：widthStride/2在32~4096（包括4096）范围内，heightStride在6~4096。 – YUV444packed/RGB888：	• 非8K缩放：支持acldvppPixelFormat枚举值中的如下枚举项： PIXEL_FORMAT_YUV_400 = 0, // 0, YUV400 8bit PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // 1, YUV420SP NV12 8bit PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // 2, YUV420SP NV21 8bit PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3, // 3, YUV422SP NV12 8bit PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4, // 4, YUV422SP NV21 8bit PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5, // 5, YUV444SP NV12 8bit PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6, // 6, YUV444SP NV21 8bit PIXEL_FORMAT_YUYV_PACKED_422 = 7, // 7, YUV422P YUYV 8bit PIXEL_FORMAT_UYVY_PACKED_422 = 8, // 8, YUV422P UYVY 8bit PIXEL_FORMAT_YVYU_PACKED_422 = 9, // 9, YUV422P YVYU 8bit PIXEL_FORMAT_VYUY_PACKED_422 = 10, // 10, YUV422P VYUY 8bit PIXEL_FORMAT_YUV_PACKED_444 = 11, // 11, YUV444P 8bit PIXEL_FORMAT_RGB_888 = 12, // 12, RGB888 PIXEL_FORMAT_BGR_888 = 13, // 13, BGR888 PIXEL_FORMAT_ARGB_8888 = 14, // 14, ARGB8888 PIXEL_FORMAT_ABGR_8888 = 15, // 15, ABGR8888 PIXEL_FORMAT_RGBA_8888 = 16, // 16, RGBA8888 PIXEL_FORMAT_BGRA_8888 = 17, // 17, BGRA8888 PIXEL_FORMAT_YUV_SEMI_PLANNER_420_10BIT = 18, // 18, YUV420SP 10bit PIXEL_FORMAT_YVU_SEM	• 内存地址起始要求16对齐。 • 输入内存大小与图片数据的格式相关，计算公式如下： – YUV400SP、YUV420SP：widthStride*heightStride*3/2 – YUV422SP：widthStride*heightStride*2 – YUV444SP：widthStride*heightStride*3 – YUV422packed：widthStride*heightStride – YUV444packed、RGB888	• 对于不同图像格式，宽stride的计算公式不同： – YUV400SP、YUV420SP、YUV422SP、YUV444SP：输入图片的宽（width）对齐到16。 – YUV422packed：输入图片的宽（width）对齐到16后，再乘以2（宽16对齐，每个像素占2个字节）。 – YUV444packed、RGB888：输入图片的宽

VPC输入/输出	图片分辨率	图片格式	内存要求	宽stride、高stride对齐要求
	widthStride/3在32~4096（包括4096）范围内，heightStride在6~4096。 – XRGB8888：widthStride/4在32~4096（包括4096）范围内，heightStride在6~4096。 ● 8K缩放：widthStride在4096~8192范围内或heightStride在4096~8192范围内（不包括4096）	I_PLANNER_420_10BIT = 19, // 19, YVU420sp 10bit ● 8K缩放：YUV420SP（NV12、NV21）。	8：widthStride*heightStride – XRGB8888：widthStride*heightStride Device的内存，调用**acldvppMalloc**接口/**acldvppFree**接口申请或释放内存。	（width）对齐到16，再乘以3（宽16对齐，每个像素占3个字节）。 – XRGB8888：输入图片的宽（width）对齐到16后，再乘以4（宽16对齐，每个像素占4个字节）。 ● 高stride：2对齐。

VPC输入/输出	图片分辨率	图片格式	内存要求	宽stride、高stride对齐要求
VPC输出	32*6~4096*4096	支持**acldvppPixelFormat**枚举值中的如下枚举项： PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // 1, YUV420SP NV12 8bit PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // 2, YUV420SP NV21 8bit	• 内存地址起始要求16对齐。 • 输出内存大小与图片数据的格式相关，计算公式如下： – YUV400SP、YUV420SP：widthStride*heightStride*3/2 – YUV422SP：widthStride*heightStride*2 – YUV444SP：widthStride*heightStride*3 – YUV422packed：widthStride*heightStride – YUV444packed、RGB888：widthS	• 宽stride：16对齐 • 高stride：2对齐。

VPC输入/输出	图片分辨率	图片格式	内存要求	宽stride、高stride对齐要求
			tride*heightStride – XRGB8888：widthStride*heightStride Device的内存，调用**acldvppMalloc**接口/**acldvppFree**接口申请或释放内存。	

- 针对缩放功能，宽高缩放比例范围：[1/32, 16]。
- 贴图场景贴图左偏移需要16对齐。

功能示意图及关键概念

图 7-1 VPC 功能示意图（抠图+缩放+叠加）

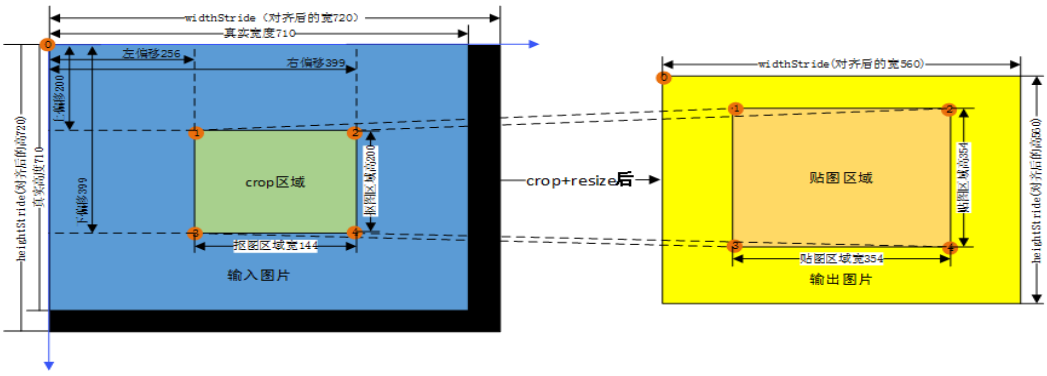


图 7-2 VPC 功能示意图（拼接）

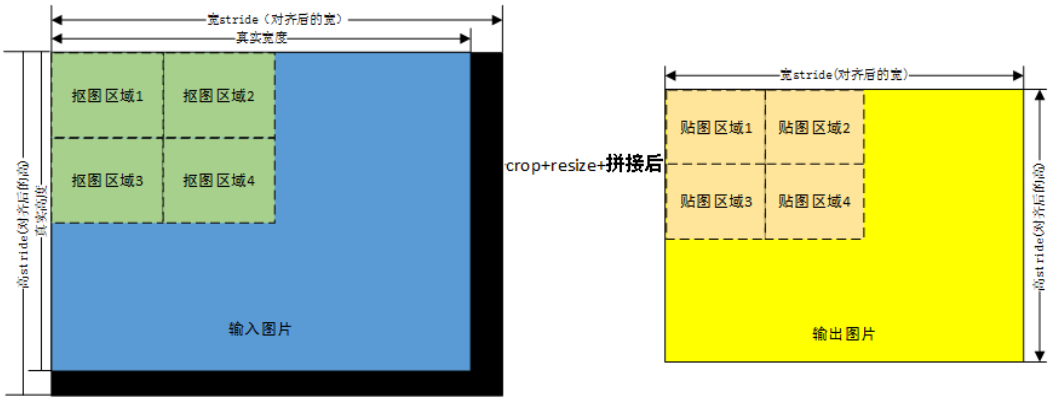


表 7-2 概念解释

概念	描述
宽stride（widthStride）	指一行图像步长，表示输入图片对齐后的宽，RGB格式或YUV格式的widthStride计算方式不一样。 widthStride的对齐要求，请参见表7-1。
高stride（heightStride）	指图像在内存中的行数，表示输入图片对齐后的高。 heightStride的对齐要求，请参见表7-1。
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。 • 左偏移：输入/输出图片中，抠图/贴图区域1、3两个点相对于0点水平向左偏移的值。 • 右偏移：输入/输出图片中，抠图/贴图区域2、4两个点相对于0点水平向左偏移的值。 • 上偏移：输入/输出图片中，抠图/贴图区域1、2两个点相对于0点垂直向上偏移的值。 • 下偏移：输入/输出图片中，抠图/贴图区域3、4两个点相对于0点垂直向上偏移的值。
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。

概念	描述
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。 约束如下： ● 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。 ● 抠图区域不超出输入图片，贴图区域不超出输出图片。 ● 贴图时可直接放置在输出图片的最左侧，即相对输出图片的左偏移为0。 ● 最大贴图个数为256个。 ● 贴图区域相对输出图片的左偏移16对齐。 ● 输出图片的贴图宽度建议16对齐，如果不是16对齐，会多写一段无效数据使其16对齐。

性能指标说明

- 对于**非8K缩放**，基本场景性能指标参考如下：

  对于1080p的图像，若存在Host->Device的图片数据拷贝，由于拷贝带宽限制，最大总帧率约为1000fps。

  对于4K的图像，若存在Host->Device的图片数据拷贝，由于拷贝带宽限制，最大总帧率约为250fps。

场景举例	总帧率
● 输入图像分辨率：1080p（1920*1080） ● 输出图像分辨率：1080p（1920*1080） ● 输入/输出图片格式：YUV420SP ● n路（n<4，1路对应一个线程）	n*360fps
● 输入图像分辨率：1080p（1920*1080） ● 输出图像分辨率：1080p（1920*1080） ● 输入/输出图片格式：YUV420SP ● n路（n≥4，1路对应一个线程）	1440fps
● 输入图像分辨率：4K图像（3840*2160） ● 输出图像分辨率：4K图像（3840*2160） ● 输入/输出图片格式：YUV420SP ● n路（n<4，1路对应一个线程）	n*90fps

场景举例	总帧率
• 输入图像分辨率：4K图像（3840*2160） • 输出图像分辨率：4K图像（3840*2160） • 输入/输出图片格式：YUV420SP • n路（n≥4，1路对应一个线程）	360fps

- 对于**8K缩放**，VPC性能与输出图像分辨率强相关，输出图像分辨率越大，处理耗时越久，性能越低。典型场景性能指标参考如下：

场景举例	总帧率
• 输入图像分辨率：8K图像（7680*4320） • 输出图像分辨率：1080p（1920*1080） • 输入/输出图片格式：YUV420SP • n路（n<4，1路对应一个线程）	n*4fps
• 输入图像分辨率：8K图像（7680*4320） • 输出图像分辨率：1080p（1920*1080） • 输入/输出图片格式：YUV420SP • n路（n≥4，1路对应一个线程）	16fps
• 输入图像分辨率：8K图像（7680*4320） • 输出图像分辨率：4K图像（3840*2160） • 输入/输出图片格式：YUV420SP • n路（n<4，1路对应一个线程）	n*1fps
• 输入图像分辨率：8K图像（7680*4320） • 输出图像分辨率：4K图像（3840*2160） • 输入/输出图片格式：YUV420SP • n路（n≥4，1路对应一个线程）	4fps

7.4.2.2 抠图

基本原理

- 调用**acldvppCreateChannel**接口创建图片数据处理的通道。
- 调用**acldvppVpcCropAsync**异步接口，按指定区域从输入图片中抠图，再将抠的图片存放到输出内存中，作为输出图片。对于异步接口，还需调用**aclrtSynchronizeStream**接口阻塞Host运行，直到指定Stream中的所有任务都完成。
- 输出图片区域与抠图区域cropArea不一致时会对图片再做一次缩放操作。

示例代码

您可以从**acl_vpc_jpege_resnet50**样例的“src/sample_process.cpp”、“src/dvpp_process.cpp”、“src/utils.cpp”文件中查看完整样例代码。

```
#include "acl/acl.h"
#include "acl/ops/acl_dvpp.h"
//1.ACL初始化
const char *aclConfigPath = "../src/acl.json";
aclError ret = aclInit(aclConfigPath);

//2.运行管理资源申请,包括Device、Context、Stream
ret = aclrtSetDevice(deviceId_);
ret = aclrtCreateContext(&context_, deviceId_);
ret = aclrtCreateStream(&stream_);

//3. 指定抠图区域的位置，cropArea_是acldvppRoiConfig类型
cropArea_ = acldvppCreateRoiConfig(550, 749, 480, 679);

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

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

//6. 申请输入内存cropInDevBuffer_，内存大小cropInBufferSize根据计算公式得出
//如果将其它操作（例如，解码）的输出内存作为抠图的输入，则无需单独申请输入内存
ret = acldvppMalloc(&cropInDevBuffer_, cropInBufferSize)
//7. 申请输出内存cropOutBufferDev_，内存大小cropOutBufferSize_根据计算公式得出
ret = acldvppMalloc(&cropOutBufferDev_, cropOutBufferSize_)

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

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

//10. 执行异步缩放，再调用aclrtSynchronizeStream接口阻塞Host运行，直到指定Stream中的所有任务都完成
```



```
ret = acldvppVpcCropAsync(dvppChannelDesc_, cropInputDesc_,
      cropOutputDesc_, cropArea_, stream_);
ret = aclrtSynchronizeStream(stream_);

//11. 解码结束后，释放资源，包括输入/输出图片的描述信息、输入/输出内存、通道描述信息、通道等
acldvppDestroyPicDesc(cropInputDesc_);
acldvppDestroyPicDesc(cropOutputDesc_);
(void)acldvppFree(cropInDevBuffer_);
(void)acldvppFree(cropOutBufferDev_);
acldvppDestroyChannel(dvppChannelDesc_);
(void)acldvppDestroyChannelDesc(dvppChannelDesc_);
dvppChannelDesc_ = nullptr;

//....
```

7.4.2.3 抠图贴图

基本原理

- 调用**acldvppCreateChannel**接口创建图片数据处理的通道。
- 调用**acldvppVpcCropAndPasteAsync**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**aclrtSynchronizeStream**接口阻塞Host运行，直到指定Stream中的所有任务都完成。
- 抠图区域cropArea的宽高与贴图区域pasteArea宽高不一致时会对图片再做一次缩放操作。
- 如果用户需要将目标图片读入内存用于存放输出图片，将贴图区域叠加在目标图片上，则需要编写代码逻辑：在申请输出内存后，将目标图片读入输出内存。

示例代码

您可以从**acl_vpc_jpege_resnet50**样例的“src/sample_process.cpp”、“src/dvpp_process.cpp”、“src/utils.cpp”文件中查看完整样例代码。

```
#include "acl/acl.h"
#include "acl/ops/acl_dvpp.h"
//1.ACL初始化
const char *aclConfigPath = "../src/acl.json";
aclError ret = aclInit(aclConfigPath);

//2.运行管理资源申请,包括Device、Context、Stream
ret = aclrtSetDevice(deviceId_);
ret = aclrtCreateContext(&context_, deviceId_);
ret = aclrtCreateStream(&stream_);

//3. 指定抠图区域的位置、指定贴图区域的位置，cropArea_和pasteArea_是acldvppRoiConfig类型
cropArea_ = acldvppCreateRoiConfig(512, 711, 512, 711);
pasteArea_ = acldvppCreateRoiConfig(16, 215, 16, 215);

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

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

//6. 申请输入内存vpcInDevBuffer_，内存大小vpcInBufferSize根据计算公式得出
//如果将其它操作（例如，解码）的输出内存作为抠图贴图的输入，则无需单独申请输入内存
//ret = acldvppMalloc(&vpcInDevBuffer_, vpcInBufferSize)
//7. 申请输出内存vpcOutBufferDev_，内存大小vpcOutBufferSize_根据计算公式得出
ret = acldvppMalloc(&vpcOutBufferDev_, vpcOutBufferSize_)

//8. 创建输入图片的描述信息，并设置各属性值
```



```
//此处示例将解码的输出内存作为抠图贴图的输入，vpcInputDesc_是acldvppPicDesc类型
vpcInputDesc_ = acldvppCreatePicDesc();
acldvppSetPicDescData(vpcInputDesc_, decodeOutBufferDev_);
acldvppSetPicDescFormat(vpcInputDesc_, PIXEL_FORMAT_YUV_SEMIPLANAR_420);
acldvppSetPicDescWidth(vpcInputDesc_, inputWidth_);
acldvppSetPicDescHeight(vpcInputDesc_, inputHeight_);
acldvppSetPicDescWidthStride(vpcInputDesc_, jpegOutWidthStride);
acldvppSetPicDescHeightStride(vpcInputDesc_, jpegOutHeightStride);
acldvppSetPicDescSize(vpcInputDesc_, jpegOutBufferSize);

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

//10. 执行异步缩放，再调用aclrtSynchronizeStream接口阻塞Host运行，直到指定Stream中的所有任务都完成
ret = acldvppVpcCropAndPasteAsync(dvppChannelDesc_, vpcInputDesc_,
      vpcOutputDesc_, cropArea_, pasteArea_, stream_);
ret = aclrtSynchronizeStream(stream_);

//11. 解码结束后，释放资源，包括输入/输出图片的描述信息、输入/输出内存、通道描述信息、通道等
acldvppDestroyPicDesc(vpcInputDesc_);
acldvppDestroyPicDesc(vpcOutputDesc_);
//(void)acldvppFree(vpcInDevBuffer_);
(void)acldvppFree(vpcOutBufferDev_);
acldvppDestroyChannel(dvppChannelDesc_);
(void)acldvppDestroyChannelDesc(dvppChannelDesc_);
dvppChannelDesc_ = nullptr;

//....
```

7.4.2.4 图片缩放

基本原理

- 调用**acldvppCreateChannel**接口创建图片数据处理的通道。
- 调用**acldvppVpcResizeAsync**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**aclrtSynchronizeStream**接口阻塞Host运行，直到指定Stream中的所有任务都完成。
- 调用**acldvppCreateResizeConfig**接口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”。
- 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

示例代码

您可以从**acl_vpc_jpege_resnet50**样例的“src/sample_process.cpp”、“src/dvpp_process.cpp”、“src/utils.cpp”文件中查看完整样例代码。

```
#include "acl/acl.h"
#include "acl/ops/acl_dvpp.h"
//1.ACL初始化
const char *aclConfigPath = "../src/acl.json";
aclError ret = aclInit(aclConfigPath);

//2.运行管理资源申请,包括Device、Context、Stream
```



```
ret = aclrtSetDevice(deviceId_);
ret = aclrtCreateContext(&context_, deviceId_);
ret = aclrtCreateStream(&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”
//resizeConfig_是acldvppResizeConfig类型
acldvppResizeConfig *resizeConfig_ = acldvppCreateResizeConfig();

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

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

//6. 申请缩放输入内存resizeInDevBuffer_，内存大小resizeInBufferSize根据计算公式得出
ret = acldvppMalloc(&resizeInDevBuffer_, resizeInBufferSize)
//7. 申请缩放输出内存resizeOutBufferDev_，内存大小resizeOutBufferSize_根据计算公式得出
ret = acldvppMalloc(&resizeOutBufferDev_, resizeOutBufferSize_)

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

//9. 创建缩放输出图片的描述信息，并设置各属性值
//如果缩放的输出图片作为模型推理的输入，则输出图片的宽高要与模型要求的宽高保持一致
//resizeOutputDesc_是acldvppPicDesc类型
resizeOutputDesc_ = acldvppCreatePicDesc();
acldvppSetPicDescData(resizeOutputDesc_, resizeOutBufferDev_);
acldvppSetPicDescFormat(resizeOutputDesc_, PIXEL_FORMAT_YUV_SEMIPLANAR_420);
acldvppSetPicDescWidth(resizeOutputDesc_, resizeOutputWidth_);
acldvppSetPicDescHeight(resizeOutputDesc_, resizeOutputHeight_);
acldvppSetPicDescWidthStride(resizeOutputDesc_, resizeOutputWidthStride);
acldvppSetPicDescHeightStride(resizeOutputDesc_, resizeOutputHeightStride);
acldvppSetPicDescSize(resizeOutputDesc_, resizeOutBufferSize_);

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

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

//....
```

7.4.3 JPEGD 功能

7.4.3.1 功能及约束说明

功能及约束说明

JPEGD（JPEG decoder）实现.jpg、.jpeg、.JPG、.JPEG图片的解码，对于硬件不支持的格式，会使用软件解码。

- 关于输入

  - 输入图片分辨率：
    - 最大分辨率：8192 * 8192，最小分辨率：32 * 32。
  - 输入图片格式
    - 只支持Huffman编码，码流的colorspace为YUV，码流的subsample为444/422/420/400；
    - 不支持算术编码；
    - 不支持渐进JPEG格式；
    - 不支持JPEG2000格式；
  - 输入内存：
    - 输入内存的大小就是指实际的输入图片所占用的大小。
    - 输入内存首地址要求128对齐。Device的内存，调用**acldvppMalloc**接口/**acldvppFree**接口申请或释放内存。
- 关于输出
  - 输出图片格式

    针对不同的源图编码格式，解码后，输出如下格式的图片：

    jpeg(YUV444SP) -> YUV444SP V在前U在后 、YUV420 SP V在前U在后、YUV420SP U在前V在后。

    jpeg(YUV422SP) -> YUV422SP V在前U在后、YUV420SP V在前U在后、YUV420SP U在前V在后。

    jpeg(YUV420SP) -> YUV420SP V在前U在后、YUV420SP U在前V在后。

    jpeg(YUV400) -> YUV420SP，UV数据采用0x80填充。
  - 输出内存：
    - 输出内存大小与图片数据的格式相关，计算公式如下：

      YUV420SP：widthStride*heightStride*3/2

      YUV422SP：widthStride*heightStride*2

      YUV444SP：widthStride*heightStride*3
    - 输出内存首地址要求128字节对齐。Device的内存，调用**acldvppMalloc**接口/**acldvppFree**接口申请或释放内存。如果申请大块内存时，内存申请计算应该是（n表示图片数量）：输出内存大小 +(n-1)*AlignTo128(输出内存大小+8)
  - 关于输出图片的宽高对齐要求：
    - 输出图片的widthStride（对齐后的宽度），对齐到128；
    - 输出图片的heightStride（对齐后的高度），对齐到16。
- 关于硬件约束：
  - 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。
- 关于软件约束：
  - 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。

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*1路 15fps。

1080p指分辨率为1920*1080的图片；4K指分辨率为3840*2160的图片。

场景举例	总帧率
1080p * 1路	128fps
1080p * n路（n≥ 2）	256fps
4k * 1路	32fps
4k * n路（n≥ 2）	64fps

7.4.3.2 图片解码

基本原理

- 调用**acldvppCreateChannel**接口创建图片数据处理的通道。
- 调用**acldvppJpegDecodeAsync**异步接口，解码.jpg、.jpeg、.JPG、.JPEG图片。对于异步接口，还需调用**aclrtSynchronizeStream**接口阻塞Host运行，直到指定Stream中的所有任务都完成。
- 可根据存放JPEG图片数据的内存，调用**acldvppJpegPredictDecSize**接口计算出JPEG图片解码后所需的输出内存的大小。

示例代码

您可以从**acl_vpc_jpege_resnet50**样例的“src/sample_process.cpp”、“src/dvpp_process.cpp”、“src/utils.cpp”文件中查看完整样例代码。

```
#include "acl/acl.h"
#include "acl/ops/acl_dvpp.h"
//1.ACL初始化
const char *aclConfigPath = "../src/acl.json";
aclError ret = aclInit(aclConfigPath);

//2.运行管理资源申请,包括Device、Context、Stream
ret = aclrtSetDevice(deviceId_);
ret = aclrtCreateContext(&context_, deviceId_);
ret = aclrtCreateStream(&stream_);

//3.将图片读入内存，inDevBuffer_表示存放输入图片的内存, inDevBufferSize表示内存大小，输入内存要提前申请
ret = acldvppMalloc(&inDevBuffer_, inDevBufferSize);
```

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

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

//6. 申请解码输出内存decodeOutDevBuffer_，内存大小decodeOutBufferSize根据计算公式得出
ret = acldvppMalloc(&decodeOutDevBuffer_, decodeOutBufferSize)

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

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

//9. 解码结束后，释放资源，包括解码输出图片的描述信息、解码输出内存、通道描述信息、通道等
acldvppDestroyPicDesc(decodeOutputDesc_);
(void)acldvppFree(decodeOutDevBuffer_);
acldvppDestroyChannel(dvppChannelDesc_);
(void)acldvppDestroyChannelDesc(dvppChannelDesc_);
dvppChannelDesc_ = nullptr;
//....
```

7.4.4 JPEGE 功能

7.4.4.1 功能及约束说明

功能及约束说明

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。

- 关于输入
  - 输入图片分辨率

    最大分辨率：8192 * 8192，最小分辨率：32 * 32。
  - 输入图片的格式：
    - YUV422 Packed（yuyv,yvyu,uyvy,vyuy）
    - YUV420SP（NV12，NV21）
  - 输入图片的宽高对齐要求：
    - 输入图片的widthStride（对齐后的宽度），对齐到16，兼容对齐到16的倍数如128。对于YUV422packed数据，widthStride应该为输入图片宽度的两倍对齐到16。
    - 输入图片的heightStride，取值：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）。其中后一

种取值的使用场景举例：JPGED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）。

  - 关于输入内存：
    - 输入内存大小与图片数据的格式相关，计算公式如下：
      YUV422packed：widthStride*heightStride
      YUV420SP：widthStride*heightStride*3/2
    - 输入内存首地址要求128对齐。Device的内存，调用**acldvppMalloc**接口/**acldvppFree**接口申请或释放内存。
- 关于输出
  - 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。
  - 关于输出内存：
    - 输出内存的大小就是指实际的编码后图片所占用的大小。
    - 输出内存首地址要求128对齐。Device的内存，调用**acldvppMalloc**接口/**acldvppFree**接口申请或释放内存。

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。

场景举例	总帧率
1080p * n路（n≥1）	64fps
4k * n路（n≥1）	16fps

7.4.4.2 图片编码

基本原理

- 调用**acldvppCreateChannel**接口创建图片数据处理的通道。
- 调用**acldvppJpegEncodeAsync**异步接口，将YUV格式图片编码成.jpg图片。对于异步接口，还需调用**aclrtSynchronizeStream**接口阻塞Host运行，直到指定Stream中的所有任务都完成。
- 调用**acldvppJpegPredictEncSize**接口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。

示例代码

您可以从**acl_vpc_jpege_resnet50**样例的“src/sample_process.cpp”、“src/dvpp_process.cpp”、“src/utils.cpp”文件中查看完整样例代码。

```
#include "acl/acl.h"
#include "acl/ops/acl_dvpp.h"
//1.ACL初始化
const char *aclConfigPath = "../src/acl.json";
aclError ret = aclInit(aclConfigPath);
```

```
//2.运行管理资源申请,包括Device、Context、Stream
ret = aclrtSetDevice(deviceId_);
ret = aclrtCreateContext(&context_, deviceId_);
ret = aclrtCreateStream(&stream_);

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

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

//5.申请内存
//5.1 输入内存
//申请Host内存inputHostBuff，存放YUV格式的图片数据
//申请Device内存inputDevBuff
//将通过aclrtMemcpy接口将Host的图片数据传输到Device，数据传输完成后，需及时调用aclrtFreeHost接口释放Host内存
aclRet = aclrtMallocHost(&inputHostBuff, PicBufferSize);
aclRet = acldvppMalloc(&inputDevBuff, PicBufferSize);
aclRet = aclrtMemcpy(inputDevBuff, PicBufferSize, inputHostBuff, PicBufferSize,
ACL_MEMCPY_HOST_TO_DEVICE);

//5.2 输出内存，申请Device内存encodeOutBufferDev_,存放编码后的输出数据
aclRet = acldvppMalloc(&encodeOutBufferDev_, outBufferSize)

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

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

//8. 执行异步解码，再调用aclrtSynchronizeStream接口阻塞Host运行，直到指定Stream中的所有任务都完成
aclRet = acldvppJpegEncodeAsync(dvppChannelDesc_, encodeInputDesc_, encodeOutBufferDev_,
          &outBufferSize, jpegeConfig_, stream_);
aclRet = aclrtSynchronizeStream(stream_);

//9.申请Host内存hostPtr，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aclrtFreeHost接口释放Host内存
aclRet = aclrtMallocHost(&hostPtr, dataSize);
aclRet = aclrtMemcpy(hostPtr, dataSize, encodeOutBufferDev_, outBufferSize,
ACL_MEMCPY_DEVICE_TO_HOST);

//11. 编码结束后，释放资源，包括编码输入/输出图片的描述信息、编码输入/输出内存、通道描述信息、通道等
acldvppDestroyPicDesc(encodeInputDesc_);
(void)acldvppFree(inputDevBuff);
(void)acldvppFree(encodeOutBufferDev_);
acldvppDestroyChannel(dvppChannelDesc_);
(void)acldvppDestroyChannelDesc(dvppChannelDesc_);
dvppChannelDesc_ = nullptr;
//....
```

7.4.5 VDEC 功能

7.4.5.1 功能及约束说明

VDEC（video decoder）实现视频的解码，VDEC内部经过VPC处理后，输出YUV420SP格式（包括NV12和NV21）的图片。

- 关于输入
  - 输入码流分辨率

    最大分辨率4096 * 4096，最小分辨率128 * 128。
  - 输入码流格式：

    H264 bp/mp/hp level5.1 YUV420编码的码流。

    H265 8/10bit level5.1 YUV420编码的码流。
  - 关于输入内存：

    Device的内存，支持调用**aclrtMalloc**接口/**aclrtFree**接口申请/释放内存，也支持调用**acldvppMalloc**/**acldvppFree**接口申请/释放内存。
- 关于输出
  - VDEC输出图像的格式为（如果不设置输出格式，默认使用YUV420SP NV12）：
    - YUV420SP NV12
    - YUV420SP NV21
  - 关于输出内存：
    - 输出内存大小与图片数据的格式相关，计算公式如下：

      YUV420SP：widthStride*heightStride*3/2
    - 输出的内存首地址要求16对齐。Device的内存，调用**aclrtMalloc**接口/**aclrtFree**接口申请或释放内存。
  - 输出图片的宽高对齐要求为：
    - 输出图片的widthStride（对齐后的宽度），对齐到16
    - 输出图片的heightStride（对齐后的高度），对齐到2
- 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不支持解码。

7.4.5.2 视频解码

基本原理

1. **调用aclvdecCreateChannel接口创建视频码流数据处理的通道。**
   - 创建视频码流数据处理通道前，需先执行以下操作：
     i. 调用**aclvdecCreateChannelDesc**接口创建通道描述信息。
     ii. 调用**aclvdecSetChannelDesc系列接口**设置通道描述信息的属性，包括解码通道号、线程、回调函数、视频编码协议等，其中：
        1) 回调函数需由用户提前创建，用于在视频解码后，获取解码数据，并及时释放相关资源，回调函数的原型请参见**8.12.7.3 aclvdecCallback**。

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

解码结束后，建议用户在回调函数内及时释放VDEC的输入码流内存、输出图片内存以及相应的视频码流描述信息、图片描述信息。

2) 线程需由用户提前创建，并自定义线程函数，在线程函数内调用**aclrtProcessReport**接口，等待指定时间后，触发**1.ii.1)**中的回调函数。

说明

如果不调用**aclvdecSetChannelDescOutPicFormat**接口设置输出格式，则默认使用YUV420SP NV12。

- **aclvdecCreateChannel**接口内部封装了如下接口，无需用户单独调用：
  i. **aclrtCreateStream**接口：显式创建Stream，VDEC内部使用。
  ii. **aclrtSubscribeReport**接口：指定处理Stream上回调函数的线程，回调函数和线程是由用户调用**aclvdecSetChannelDesc系列接口**时指定的。

2. **调用aclvdecSendFrame接口将视频码流解码成YUV420SP格式的图片。**
   - 视频解码前，需先执行以下操作：
     - 调用**acldvppCreateStreamDesc**接口创建输入视频码流描述信息，并调用**acldvppSetStreamDesc系列接口**设置输入视频的内存地址、内存大小、码流格式等属性。
     - 调用**acldvppCreatePicDesc**接口创建输出图片描述信息，并调用**acldvppSetPicDesc系列接口**设置输出图片的内存地址、内存大小、图片格式等属性。
   - **aclvdecSendFrame**接口内部封装了**aclrtLaunchCallback**接口，用于在Stream的任务队列中增加一个需要在Host上执行的回调函数。用户无需单独调用**aclrtLaunchCallback**接口。
3. **调用aclvdecDestroyChannel接口销毁视频处理的通道。**
   - 系统会等待已发送帧解码完成且用户的回调函数处理完成后再销毁通道。
   - **aclvdecDestroyChannel**接口内部封装了如下接口，无需用户单独调用：
     - **aclrtUnSubscribeReport**接口：取消线程注册（Stream上的回调函数不再由指定线程处理）。
     - **aclrtDestroyStream**接口：销毁Stream。
   - 销毁通道后，需调用**aclvdecDestroyChannelDesc**接口销毁通道描述信息。

示例代码

您可以从**acl_vdec_resnet50**样例的“src”目录下查看完整样例代码。

```
#include "acl/acl.h"
#include "acl/ops/acl_dvpp.h"
//1.ACL初始化
const char *aclConfigPath = "../src/acl.json";
aclError ret = aclInit(aclConfigPath);

//2.运行管理资源申请,包括Device、Context、Stream
ret = aclrtSetDevice(deviceId_);
```

```
ret = aclrtCreateContext(&context_, deviceId_);
ret = aclrtCreateStream(&stream_);

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

            // free output vdecOutBufferDev
            aclError ret = acldvppFree(vdecOutBufferDev);
            if (ret != ACL_ERROR_NONE) {
                ERROR_LOG("fail to free output pic desc data");
            }
        }
        // 释放acldvppPicDesc类型的数据，表示解码后输出图片描述数据
        aclError ret = acldvppDestroyPicDesc(output);
        if (ret != ACL_ERROR_NONE) {
            ERROR_LOG("fail to destroy output pic desc");
        }
    }

    // free input vdecInBufferDev and destroy stream desc
    if (input != nullptr) {
        void *vdecInBufferDev = acldvppGetStreamDescData(input);
        if (vdecInBufferDev != nullptr) {
            aclError ret = acldvppFree(vdecInBufferDev);
            if (ret != ACL_ERROR_NONE) {
                ERROR_LOG("fail to free input stream desc data");
            }
        }
        // 释放acldvppStreamDesc类型的数据，表示解码的输入码流描述数据
        aclError ret = acldvppDestroyStreamDesc(input);
        if (ret != ACL_ERROR_NONE) {
            ERROR_LOG("fail to destroy input stream desc");
        }
    }

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

//4.创建视频码流处理通道时的通道描述信息，设置视频处理通道描述信息的属性，其中callback回调函数需要用
户提前创建。
//vdecChannelDesc_是aclvdecChannelDesc类型
vdecChannelDesc_ = aclvdecCreateChannelDesc();
// channelId: 0-15
ret = aclvdecSetChannelDescChannelId(vdecChannelDesc_, 10);
ret = aclvdecSetChannelDescThreadId(vdecChannelDesc_, threadId_);
ret = aclvdecSetChannelDescCallback(vdecChannelDesc_, callback);
//示例中使用的是H265_MAIN_LEVEL视频编码协议
ret = aclvdecSetChannelDescEnType(vdecChannelDesc_, static_cast<acldvppStreamFormat>(enType_));
//示例中使用的是PIXEL_FORMAT_YVU_SEMIPLANAR_420
```



```
ret = aclvdecSetChannelDescOutPicFormat(vdecChannelDesc_, static_cast<acldvppPixelFormat>(format_));

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

//6.申请Device内存dataDev，存放视频解码的输入视频数据
//将通过aclrtMemcpy接口将Host的图片数据传输到Device，数据传输完成后，需及时调用aclrtFreeHost接口释放Host内存
void *dataDev = nullptr;
auto aclRet = acldvppMalloc(&dataDev, dataSize);
//dataHost表示存放Host上输入视频数据的内存，fileLen表示内存大小
aclRet = aclrtMemcpy(dataDev, dataSize, dataHost, fileLen, ACL_MEMCPY_HOST_TO_DEVICE);

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

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

    //7.3 创建输出图片描述信息，设置图片描述信息的属性
    //picOutputDesc_是acldvppPicDesc类型
    picOutputDesc_ = acldvppCreatePicDesc();
    ret = acldvppSetPicDescData(picOutputDesc_, picOutBufferDev_);
    ret = acldvppSetPicDescSize(picOutputDesc_, size);
    ret = acldvppSetPicDescFormat(picOutputDesc_, static_cast<acldvppPixelFormat>(format_));

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

//8.释放资源
ret = aclvdecDestroyChannel(vdecChannelDesc_);
aclvdecDestroyChannelDesc(vdecChannelDesc_);

//......
```

返回码说明

表 7-3 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。	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
AICPU_DVPP_KERNEL_STATE_DVPP_ERROR = 2	ACL内部调用其它模块的接口失败。	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》。
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。	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》。
AICPU_DVPP_KERNEL_STATE_QUEUE_FULL = 6	系统内部队列满。	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》。
AICPU_DVPP_KERNEL_STATE_QUEUE_EMPTY = 7	系统内部队列空。	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》。
AICPU_DVPP_KERNEL_STATE_QUEUE_NOT_EXIST = 8	系统内部队列不存在。	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》。
AICPU_DVPP_KERNEL_STATE_GET_CONTEX_FAILED = 9	获取系统内部上下文失败。	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》。
AICPU_DVPP_KERNEL_STATE_SUBMIT_EVENT_FAILED = 10	提交系统内部事件失败。	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》。
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。

返回码	含义	可能原因及解决方法
AICPU_DVPP_KERNEL_STATE_SEND_NOTIFY_FAILED = 12	发送系统内部通知失败。	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》。
AICPU_DVPP_KERNEL_STATE_VPC_OPERATE_FAILED = 13	系统内部接口处理失败。	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》。
ERR_INVALID_STATE = 0x10001	VDEC解码器状态异常错误	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》。
ERR_HARDWARE = 0x10002	硬件错误，包含解码器启动、执行、停止等异常	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》。
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，暂未使用，预留	-
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。

返回码	含义	可能原因及解决方法
ERR_IMAGE_FORMAT = 0x1000a	输出格式配置错误	请检查输出图像的格式是否为nv12或nv21。
ERR_CALLBACK = 0x1000b	回调函数为空	请检查配置的回调函数是否为空。
ERR_INPUT_BUFFER = 0x1000c	输入内存为空	请检输入内存是否为空。
ERR_INBUF_SIZE = 0x1000d	输入内存大小<=0	请检查输入内存大小是否小于等于0。
ERR_THREAD_CREATE_FBD_FAIL = 0x1000e	系统内部将解码结果通过回调函数返回给用户的线程异常	请检查系统中资源（例如：线程、内存等）是否可用。
ERR_CREATE_INSTANCE_FAIL = 0x1000f	创建解码实例失败	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》。
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）	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》。
ERR_GET_CHANNEL_HANDLE_FAIL = 0x10011	系统内部获取某路视频流的解码句柄失败	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》。
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	请联系华为工程师。

7.4.6 VENC 功能

7.4.6.1 功能及约束说明

功能及约束说明

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。

- 关于输入
  - 输入图片分辨率
    最大分辨率1920 * 1920，最小分辨率128 * 128。
  - 输入图片格式
    YUV420SP NV12/NV21-8bit
  - 关于输入内存
    Device的内存，调用**acldvppMalloc**/**acldvppFree**申请/释放内存。
- 关于输出
  - 输出码流格式
    H264 BP/MP/HP
    H265 MP（仅支持Slice码流）
  - 关于输出内存
    不需要用户管理输出内存，由系统管理内存。

性能指标说明

1080p指分辨率为1920*1080的图片。

场景举例	总帧率
1080p * n路（一个进程对应一路）	30fps

7.4.6.2 视频编码

基本原理

1. **调用aclvencCreateChannel接口创建视频编码处理的通道。**
   - 创建视频编码处理通道前，需先执行以下操作：
     i. 调用**aclvencCreateChannelDesc**接口创建通道描述信息。
     ii. 调用**aclvencSetChannelDesc系列接口**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**8.12.8.3 aclvencCallback**。
     视频编码结束后，建议用户在回调函数内及时释放输入图片内存、以及相应的图片描述信息。视频编码的输出内存由系统管理，不由用户管理，因此无需用户释放。

2) 线程需由用户提前创建，并自定义线程函数，在线程函数内调用**aclrtProcessReport**接口，等待指定时间后，触发**1.ii.1)**中的回调函数。

- **aclvencCreateChannel**接口内部封装了如下接口，无需用户单独调用：
  i. **aclrtCreateStream**接口：显式创建Stream，VENC内部使用。
  ii. **aclrtSubscribeReport**接口：指定处理Stream上回调函数的线程，回调函数和线程是由用户调用**aclvencSetChannelDesc系列接口**时指定的。

2. **调用aclvencSendFrame接口将YUV420SP格式的图片编码成H264/H265格式的视频码流。**
   - 视频编码前，需先执行以下操作：
     - 调用**acldvppCreatePicDesc**接口创建输入图片描述信息，并调用**acldvppSetPicDesc系列接口**设置输入图片的内存地址、内存大小、图片格式等属性。
     - 调用**aclvencCreateFrameConfig**接口创建单帧编码配置数据，并调用**aclvencSetFrameConfig系列接口**设置是否强制重新开始I帧间隔、是否结束帧。
   - **aclvencSendFrame**接口内部封装了**aclrtLaunchCallback**接口，用于在Stream的任务队列中增加一个需要在Host上执行的回调函数。用户无需单独调用**aclrtLaunchCallback**接口。
3. **调用aclvencDestroyChannel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。
   - **aclvencDestroyChannel**接口内部封装了如下接口，无需用户单独调用：
     - **aclrtUnSubscribeReport**接口：取消线程注册（Stream上的回调函数不再由指定线程处理）。
     - **aclrtDestroyStream**接口：销毁Stream。
   - 销毁通道后，需调用**aclvencDestroyChannelDesc**接口销毁通道描述信息。

7.5 模型推理

7.5.1 单 Batch+固定 shape+静态 AIPP+单模型同步推理

1. 若涉及色域转换（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，在模型加载前，需要先参见《**ATC工具使用指导**》中的AIPP配置。
2. 在模型推理前，需要从离线模型文件（适配昇腾AI处理器的离线模型）中加载模型数据到内存中，并创建aclmdlDataset类型的数据描述模型的输出。请参见**6.3.1.4 模型推理资源申请**。
3. 加载模型后，再同步执行模型推理。请参见**6.3.4 模型推理（单Batch+固定shape+静态AIPP+单模型）**。

7.5.2 多模型推理

多模型推理的基本流程与单模型类似，请参见**7.5.1 单Batch+固定shape+静态AIPP+单模型同步推理**。

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。
  加载模型数据分为以下4种方式：
  - **aclmdlLoadFromFile**：从文件加载离线模型数据，由系统内部管理内存。
  - **aclmdlLoadFromMem**：从内存加载离线模型数据，由系统内部管理内存。
  - **aclmdlLoadFromFileWithMem**：从文件加载离线模型数据，由用户自行管理模型运行的内存（包括工作内存和权值内存）。
  - **aclmdlLoadFromMemWithMem**：从内存加载离线模型数据，由用户自行管理模型运行的内存（包括工作内存和权值内存）
- 关于模型推理，如果涉及多个模型，需调用多次模型推理接口。
  调用**aclmdlExecute**接口实现同步模型推理。

7.5.3 多 Batch

多Batch推理的基本流程与单Batch类似，请参见**7.5.1 单Batch+固定shape+静态AIPP+单模型同步推理**。

多Batch推理与单Batch推理的不同点在于，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。

样例代码如下，以8Batch为例：

```
uint32_t batchSize = 8;
uint32_t deviceNum = 1;
uint32_t deviceId = 0;

//获取模型第一个输入的大小
uint32_t modelInputSize = aclmdlGetInputSizeByIndex(modelDesc, 0);
//获取每个Batch输入数据的大小
uint32_t singleBuffSize = modelInputSize / batchSize;

//定义该变量，用于累加Batch数是否达到8Batch
uint32_t cnt = 0;
//定义该变量，用于描述每个文件读入内存时的位置偏移
uint32_t pos = 0;

void* p_batchDst = NULL;
std::vector<std::string>inferFile_vec;

for (int i = 0; i < files.size(); ++i)
    {
        //每8个文件，申请一次Device上的内存，存放8Batch的输入数据
        if (cnt % batchSize == 0)
        {
            pos = 0;
            inferFile_vec.clear();
            //申请Device上的内存
            ret = aclrtMalloc(&p_batchDst, modelInputSize, ACL_MEM_MALLOC_NORMAL_ONLY);
        }

        //TODO: 从某个目录下读入文件，计算文件大小fileSize

        //根据文件大小，申请Host上的内存，存放文件数据
        ret = aclrtMallocHost(&p_imgBuf, fileSize);

        //将Host上的文件数据复制到Device的内存中
        ret = aclrtMemcpy((uint8_t *)p_batchDst + pos, fileSize, p_imgBuf, fileSize,
ACL_MEMCPY_HOST_TO_DEVICE);
        pos += fileSize;
```



```
        //释放Host上的内存
        aclrtFreeHost(p_imgBuf);

        //将第i个文件存入vector中，同时cnt+1
        inferFile_vec.push_back(files[i]);
        cnt++;

        //每8Batch的输入数据送给模型推理进行推理
        if (cnt % batchSize == 0)
        {
            //TODO: 创建aclmdlDataset、aclDataBuffer类型的数据，用于描述模型的输入、输出数据
            //TODO: 调用aclmdlExecute接口执行模型推理
            //TODO: 推理结束后，调用aclrtFree接口释放Device上的内存
        }
    }

//如果最后循环遍历所有的输入数据后，仍不满足多Batch的要求，则直接将剩余数据作为模型推理的输入。
if (cnt % batchSize != 0)
  {
        //TODO: 创建aclmdlDataset、aclDataBuffer类型的数据，用于描述模型的输入、输出数据
        //TODO: 调用aclmdlExecute接口执行模型推理
        //TODO: 推理结束后，调用aclrtFree接口释放Device上的内存
  }
```

7.5.4 动态 Batch

基本原理

动态Batch场景下，关键接口的调用流程如下：

1. 加载模型。模型加载成功后，返回标识模型的ID。

   加载模型数据分为以下4种方式：

   - **aclmdlLoadFromFile**：从文件加载离线模型数据，由系统内部管理内存。
   - **aclmdlLoadFromMem**：从内存加载离线模型数据，由系统内部管理内存。
   - **aclmdlLoadFromFileWithMem**：从文件加载离线模型数据，由用户自行管理模型运行的内存（包括工作内存和权值内存）。
   - **aclmdlLoadFromMemWithMem**：从内存加载离线模型数据，由用户自行管理模型运行的内存（包括工作内存和权值内存）。

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

   a. 调用**aclCreateDataBuffer**接口创建aclDataBuffer类型的数据，用于存放输入/输出数据的内存地址、内存大小，内存需提前调用**aclrtMalloc**接口申请。

      申请动态Batch输入对应的内存前，需要先调用**aclmdlGetInputIndexByName**接口根据输入名称（固定为ACL_DYNAMIC_TENSOR_NAME）获取模型中标识动态Batch输入的index，再调用**aclmdlGetInputSizeByIndex**接口根据index获取内存大小。申请动态Batch输入对应的内存后，无需用户设置内存中的数据（否则可能会导致业务异常），用户调用**3.b**中的接口后，系统会自动向内存中填入数据。

      说明

      ACL_DYNAMIC_TENSOR_NAME是一个宏，宏的定义如下：

      ```
      #define ACL_DYNAMIC_TENSOR_NAME "ascend_mbatch_shape_data"
      ```

   b. 调用**aclmdlCreateDataset**接口创建aclmdlDataset类型的数据，并调用**aclmdlAddDatasetBuffer**接口向aclmdlDataset类型的数据中增加aclDataBuffer类型的数据。

3. 在成功加载模型之后，执行模型之前，设置动态Batch数。
   a. 调用**aclmdlGetInputIndexByName**接口根据输入名称（固定为ACL_DYNAMIC_TENSOR_NAME）获取模型中标识动态Batch输入的index。
   b. 调用**aclmdlSetDynamicBatchSize**接口设置动态Batch数。
      此处设置的Batch数只能是模型转换时通过dynamic_batch_size参数设置的Batch档位中的某一个，模型转换的详细说明请参见《**ATC工具使用指导**》。
      也可以调用**aclmdlGetDynamicBatch**接口获取指定模型支持的Batch档位数以及每一档中的Batch数。
4. 执行模型。
   调用**aclmdlExecute**接口执行模型。

须知

对同一个模型，不能同时调用**aclmdlSetDynamicBatchSize**接口和**aclmdlSetDynamicHWSize**接口。

示例代码

```
//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
int  ModelSetDynamicInfo()
{
        size_t index;
        //2.1 获取动态Batch输入的index，标识动态Batch输入的输入名称固定为ACL_DYNAMIC_TENSOR_NAME
        aclError ret = aclmdlGetInputIndexByName(modelDesc_, ACL_DYNAMIC_TENSOR_NAME, &index);
        //2.2 设置Batch，modelId_表示加载成功的模型的ID，input_表示aclmdlDataset类型的数据，index表示标识动态Batch输入的输入index
        uint64_t batchSize = 8;
        ret = aclmdlSetDynamicBatchSize(modelId_, input_, index, batchSize);
        //.....
}

//4.自定义函数，执行模型
int ModelExecute(int index)
{
        aclError ret;
        //4.1 调用自定义函数，设置动态Batch
    ret = ModelSetDynamicInfo();
        //4.2 执行模型，modelId_表示加载成功的模型的ID，input_和output_分别表示模型的输入和输出
        ret = aclmdlExecute(modelId_, input_, output_);
        //.....
}
//5.处理模型推理结果
//TODO
```

7.5.5 动态分辨率

基本原理

动态分辨率场景下，关键接口的调用流程如下：

1. 加载模型。模型加载成功后，返回标识模型的ID。

加载模型数据分为以下4种方式：

- **aclmdlLoadFromFile**：从文件加载离线模型数据，由系统内部管理内存。
- **aclmdlLoadFromMem**：从内存加载离线模型数据，由系统内部管理内存。
- **aclmdlLoadFromFileWithMem**：从文件加载离线模型数据，由用户自行管理模型运行的内存（包括工作内存和权值内存）。
- **aclmdlLoadFromMemWithMem**：从内存加载离线模型数据，由用户自行管理模型运行的内存（包括工作内存和权值内存）。

2. 创建aclmdlDataset类型的数据，用于描述模型的输入数据、输出数据。
   a. 调用**aclCreateDataBuffer**接口创建aclDataBuffer类型的数据，用于存放输入/输出数据的内存地址、内存大小，内存需提前调用**aclrtMalloc**接口申请。

      申请动态分辨率输入对应的内存前，需要先调用**aclmdlGetInputIndexByName**接口根据输入名称（固定为ACL_DYNAMIC_TENSOR_NAME）获取模型中标识动态分辨率输入的index，再调用**aclmdlGetInputSizeByIndex**接口根据index获取内存大小。申请动态分辨率输入对应的内存后，无需用户设置内存中的数据（否则可能会导致业务异常），用户调用**3.b**中的接口后，系统会自动向内存中填入数据。

      说明

      ACL_DYNAMIC_TENSOR_NAME是一个宏，宏的定义如下：
      ```
      #define ACL_DYNAMIC_TENSOR_NAME "ascend_mbatch_shape_data"
      ```

   b. 调用**aclmdlCreateDataset**接口创建aclmdlDataset类型的数据，并调用**aclmdlAddDatasetBuffer**接口向aclmdlDataset类型的数据中增加aclDataBuffer类型的数据。
3. 在成功加载模型之后，执行模型之前，设置动态分辨率（模型的输入图片的宽和高）。
   a. 根据输入名称（固定为ACL_DYNAMIC_TENSOR_NAME），获取模型中标识动态分辨率输入的index。
   b. 调用**aclmdlSetDynamicHWSize**接口设置动态分辨率。

      此处设置的分辨率只能是模型转换时通过dynamic_image_size参数设置的分辨率档位中的某一个，模型转换的详细说明请参见《**ATC工具使用指导**》。

      也可以调用**aclmdlGetDynamicHW**接口获取指定模型支持的分辨率档位数以及每一档中的宽、高。
4. 执行模型。

   调用**aclmdlExecute**接口执行模型。

须知

对同一个模型，不能同时调用**aclmdlSetDynamicBatchSize**接口和**aclmdlSetDynamicHWSize**接口。

示例代码

```
//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
int  ModelSetDynamicInfo()
{
        size_t index;
                //2.1 获取动态分辨率输入的index，标识动态分辨率输入的输入名称固定为
ACL_DYNAMIC_TENSOR_NAME
        aclError ret = aclmdlGetInputIndexByName(modelDesc_, ACL_DYNAMIC_TENSOR_NAME, &index);
                //2.2 设置输入图片分辨率，modelId_表示加载成功的模型的ID，input_表示aclmdlDataset类型的数
据，index表示标识动态分辨率输入的输入index
                uint64_t height = 224;
        uint64_t width = 224;
        ret = aclmdlSetDynamicHWSize(modelId_, input_, index, height, width);
                //.....
}

//4.自定义函数，执行模型
int ModelExecute(int index)
{
        aclError ret;
        //4.1 调用自定义函数，设置动态分辨率
    ret = ModelSetDynamicInfo();
        //4.2 执行模型，modelId_表示加载成功的模型的ID，input_和output_分别表示模型的输入和输出
        ret = aclmdlExecute(modelId_, input_, output_);
        //.....
}

//5.处理模型推理结果
//TODO
```

7.5.6 动态 AIPP

基本原理

动态AIPP场景下，关键接口的调用流程如下：

1. 加载模型。模型加载成功后，返回标识模型的ID。

   加载模型数据分为以下4种方式：

   - **aclmdlLoadFromFile**：从文件加载离线模型数据，由系统内部管理内存。
   - **aclmdlLoadFromMem**：从内存加载离线模型数据，由系统内部管理内存。
   - **aclmdlLoadFromFileWithMem**：从文件加载离线模型数据，由用户自行管理模型运行的内存（包括工作内存和权值内存）。
   - **aclmdlLoadFromMemWithMem**：从内存加载离线模型数据，由用户自行管理模型运行的内存（包括工作内存和权值内存）。

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

   a. 调用**aclCreateDataBuffer**接口创建aclDataBuffer类型的数据，用于存放输入/输出数据的内存地址、内存大小，内存需提前调用**aclrtMalloc**接口申请。

      申请动态AIPP输入对应的内存前，需要先调用**aclmdlGetInputIndexByName**接口根据输入名称（固定为ACL_DYNAMIC_AIPP_NAME）获取模型中标识动态AIPP输入的index，再调用**aclmdlGetInputSizeByIndex**接口根据index获取内存大小。申请动态AIPP输入对应的内存后，无需用户设置内存中的数据（否则可能会导致业务异常），用户调用**3.c**中的接口后，系统会自动向内存中填入数据。

说明

ACL_DYNAMIC_AIPP_NAME是一个宏，宏的定义如下：

```
#define ACL_DYNAMIC_AIPP_NAME "ascend_dynamic_aipp_data"
```

b. 调用**aclmdlCreateDataset**接口创建aclmdlDataset类型的数据，并调用**aclmdlAddDatasetBuffer**接口向aclmdlDataset类型的数据中增加aclDataBuffer类型的数据。

3. 在成功加载模型之后，执行模型之前，设置**动态AIPP**参数。

a. 根据输入名称（固定为ACL_DYNAMIC_AIPP_NAME），获取模型中标识动态AIPP输入的index。

b. 调用**aclmdlCreateAIPP**接口创建aclmdlAIPP类型。

c. 根据实际需求，调用**8.15.17.2 设置动态AIPP参数**中提供的接口设置动态AIPP参数值。

动态AIPP场景下，**aclmdlSetAIPPInputFormat**接口、**aclmdlSetAIPPSrcImageSize**接口（设置原始图片的宽和高）必须调用。

d. 调用**aclmdlSetInputAIPP**接口设置模型推理时的动态AIPP数据。

e. 及时调用**aclmdlDestroyAIPP**接口销毁aclmdlAIPP类型。

4. 执行模型。

调用**aclmdlExecute**接口执行模型。

示例代码

```
//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_和模型的输出数据output_
//......

//3.自定义函数，设置动态AIPP参数值
int  ModelSetDynamicAIPP()
{
   //3.1 获取标识动态AIPP输入的index
   size_t index;
   aclError ret = aclmdlGetInputIndexByName(modelDesc_, ACL_DYNAMIC_AIPP_NAME, &index);

   //3.2 设置动态AIPP参数值
   uint64_t batchNumber = 1;
   aclmdlAIPP *aippDynamicSet = aclmdlCreateAIPP(batchNumber);
   ret = aclmdlSetAIPPSrcImageSize(aippDynamicSet, 256, 224);
   ret = aclmdlSetAIPPInputFormat(aippDynamicSet, ACL_YUV420SP_U8);
   ret = aclmdlSetAIPPCscParams(aippDynamicSet, 1, 256, 443, 0, 256, -86, -178, 256, 0, 350, 0, 0, 0, 0,
128, 128);
   ret = aclmdlSetAIPPRbuvSwapSwitch(aippDynamicSet, 0);
   ret = aclmdlSetAIPPDtcPixelMean(aippDynamicSet, 0, 0, 0, 0, 0);
   ret = aclmdlSetAIPPDtcPixelMin(aippDynamicSet, 0, 0, 0, 0, 0);
   ret = aclmdlSetAIPPPixelVarReci(aippDynamicSet, 1, 1, 1, 1, 0);
   ret = aclmdlSetAIPPCropParams(aippDynamicSet, 1, 0, 0, 224, 224, 0);
   ret = aclmdlSetInputAIPP(modelId_, input_, index, aippDynamicSet);
   ret = aclmdlDestroyAIPP(aippDynamicSet);

   //......
}


//4.自定义函数，执行模型
int ModelExecute(int index)
{
      aclError ret;
      //4.1 调用自定义函数，设置动态AIPP参数值
```



```
    ret = ModelSetDynamicAIPP();
        //4.2 执行模型，modelId_表示加载成功的模型的ID，input_和output_分别表示模型的输入和输出
        ret = aclmdlExecute(modelId_, input_, output_);
        //.....
}

//5.处理模型推理结果
//TODO
```

7.6 数据后处理

7.6.1 目标分类应用的通用后处理方法

请参见**6.3.5 数据后处理（调用单算子），回传结果到Host**。

7.7 单算子调用

7.7.1 内置算子 GEMM 被封装成 ACL 接口

7.7.1.1 基本原理

目前，ACL已将矩阵-向量乘、矩阵-矩阵乘相关的GEMM算子封装成ACL接口，详细接口请参见**8.11 CBLAS接口**。

单算子执行的基本流程如下：

1. 资源初始化，包括ACL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、申请Device和Host上的内存存放算子数据。
   - 调用**aclInit**接口实现ACL初始化。
   - 单算子模型文件，需要用户提前参见《**ATC工具使用指导**》中的使用ATC工具转换模型将单算子定义文件（*.json）编译成适配昇腾AI处理器的离线模型（*.om文件）。
   - 加载单算子模型文件，有两种方式：

     调用**aclopSetModelDir**接口，设置加载模型文件的目录，目录下存放单算子模型文件（*.om文件）。

     调用**aclopLoad**接口，从内存中加载单算子模型数据，由用户管理内存。单算子模型数据是指“单算子编译成*.om文件后，再将om文件读取到内存中”的数据。
   - 调用**aclrtSetDevice**接口指定运算的Device。
   - 调用**aclrtCreateContext**接口显式创建一个Context，调用**aclrtCreateStream**接口显式创建一个Stream。

     若没有显式创建Stream，则使用默认Stream，默认Stream是在调用**aclrtSetDevice**接口时隐式创建的，默认Stream作为接口入参时，直接传NULL。
2. 将算子输入数据从Host复制到Device上。
   - 调用**aclrtMemcpy**接口实现同步内存复制。
   - 调用**aclrtMemcpyAsync**接口实现异步内存复制。

3. 调用ACL提供的接口执行算子，本章以ACL接口（**aclblasGemmEx**接口）为例。
4. 将算子运算后的输出数据从Device上复制到Host上。
   - 调用**aclrtMemcpy**接口实现同步内存复制。
   - 调用**aclrtMemcpyAsync**接口实现异步内存复制。
5. 按顺序先释放Stream资源，再释放Context资源，最后释放Device资源。
   - 调用**aclrtDestroyStream**接口释放Stream。

     不涉及显式创建Stream，使用默认Stream时，无需调用**aclrtDestroyStream**接口释放Stream。
   - 调用**aclrtDestroyContext**接口释放Context。

     不涉及显式创建Context，使用默认Context时，无需调用**aclrtDestroyContext**接口释放Context。
   - 调用**aclrtResetDevice**接口释放Device。
6. 调用**aclFinalize**接口实现ACL去初始化。

7.7.1.2 资源初始化

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。

示例代码如下，您可以从**acl_execute_gemm**样例的“src/gemm_main.cpp”文件中查看完整样例代码。（没有显式创建Context，使用默认Context。）

```
#include "acl/acl.h"
//......

//1. ACL初始化
//此处是相对路径，相对可执行文件所在的目录
aclInit("test_data/config/acl.json");

//2.设置单算子模型文件所在的目录
//该目录相对可执行文件所在的目录，例如，编译出来的可执行文件存放在run/out目录下，此处就表示run/out/op_models目录
aclopSetModelDir("op_models");

//3. 指定用于运算的设备
int deviceId = 0;
aclrtSetDevice(deviceId);

//4.申请Device上的内存存放算子的输入数据
//对于该矩阵乘示例，sizeA_表示矩阵A数据的大小，sizeB_表示矩阵B数据的大小，sizeC_表示矩阵C数据的大小
aclrtMalloc((void **) &devMatrixA_, sizeA_, ACL_MEM_MALLOC_NORMAL_ONLY)
aclrtMalloc((void **) &devMatrixB_, sizeB_, ACL_MEM_MALLOC_NORMAL_ONLY)
aclrtMalloc((void **) &devMatrixC_, sizeC_, ACL_MEM_MALLOC_NORMAL_ONLY)

//5.申请Host上的内存存放算子的回传结果
//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
hostMatrixA_ = new(std::nothrow) aclFloat16[m_ * k_];
hostMatrixB_ = new(std::nothrow) aclFloat16[k_ * n_];
hostMatrixC_ = new(std::nothrow) aclFloat16[m_ * n_]

//......
```

7.7.1.3 数据传输到 Device

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。

示例代码如下，您可以从**acl_execute_gemm**样例的“src/gemm_main.cpp”、“src/gemm_runner.cpp”文件中查看完整样例代码。

```
#include "acl/acl.h"
//......

//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
auto ret = aclrtMemcpy((void *) devMatrixA_, sizeA_, hostMatrixA_, sizeA_,
ACL_MEMCPY_HOST_TO_DEVICE);
ret = aclrtMemcpy((void *) devMatrixB_, sizeB_, hostMatrixB_, sizeB_, ACL_MEMCPY_HOST_TO_DEVICE);

//......
```

7.7.1.4 执行单算子，回传结果到 Host

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。

示例代码如下，您可以从**acl_execute_gemm**样例的“src/gemm_main.cpp”、“src/gemm_runner.cpp”文件中查看完整样例代码。

```
#include "acl/acl.h"
#include "acl/ops/acl_cblas.h"

//......

//显式创建一个Stream
aclrtStream stream = nullptr;
aclrtCreateStream(&stream);

//对于该示例，调用aclblasGemmEx接口（异步接口）实现矩阵-矩阵的乘法
aclblasGemmEx(ACL_TRANS_N, ACL_TRANS_N, ACL_TRANS_N, m_, n_, k_,
              devAlpha_, devMatrixA_, k_, inputType_, devMatrixB_, n_, inputType_,
              devBeta_, devMatrixC_, n_, outputType_, ACL_COMPUTE_HIGH_PRECISION,
              stream);

//调用aclrtSynchronizeStream接口阻塞Host运行，直到指定Stream中的所有任务都完成
aclrtSynchronizeStream(stream);

//将算子的输出数据从Device复制到Host
aclrtMemcpy(hostMatrixC_, sizeC_, devMatrixC_, sizeC_, ACL_MEMCPY_DEVICE_TO_HOST);

//TODO：将算子输出数据在终端屏幕上显示，同时将算子输出数据写到文件中

//......
```

7.7.1.5 运行管理资源释放与 ACL 去初始化

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。

示例代码如下，您可以从**acl_execute_gemm**样例的“src/gemm_main.cpp”、“src/gemm_runner.cpp”文件中查看完整样例代码。不涉及显式创建Context，使用默认Context时，无需显式释放Context。

```
#include "acl/acl.h"
//......

// 释放显式创建的Stream
(void) aclrtDestroyStream(stream);

//释放Device资源
(void) aclrtResetDevice(deviceId);

//ACL去初始化
```



```
aclFinalize();
//.....
```

7.7.2 内置算子没有封装成 ACL 接口

基本原理

单算子执行的基本流程如下，详细样例请参见**6.3.5 数据后处理（调用单算子），回传结果到Host**：

1. 资源初始化，包括ACL初始化、设置单算子模型文件的加载目录、指定用于运算的Device等。
   - 调用**aclInit**接口实现ACL初始化。
   - 单算子模型文件，需要用户提前参见《**ATC工具使用指导**》中的使用ATC工具转换模型将单算子定义文件（*.json）编译成适配昇腾AI处理器的离线模型（*.om文件）。
   - 加载单算子模型文件，有两种方式：

     调用**aclopSetModelDir**接口，设置加载模型文件的目录，目录下存放单算子模型文件（*.om文件）。

     调用**aclopLoad**接口，从内存中加载单算子模型数据，由用户管理内存。单算子模型数据是指“单算子编译成*.om文件后，再将om文件读取到内存中”的数据。
   - 调用**aclrtSetDevice**接口指定运算的Device。
   - 调用**aclrtCreateContext**接口显式创建一个Context，调用**aclrtCreateStream**接口显式创建一个Stream。

     若没有显式创建Stream，则使用默认Stream，默认Stream是在调用**aclrtSetDevice**接口时隐式创建的，默认Stream作为接口入参时，直接传NULL。
2. 将算子输入数据从Host复制到Device上。
   - 调用**aclrtMemcpy**接口实现同步内存复制。
   - 调用**aclrtMemcpyAsync**接口实现异步内存复制。
3. 执行单算子。

   ACL支持以下两种方式执行单算子：
   - 您需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、调用**aclopExecute**接口加载并执行算子。

     该方式下，每次调用**aclopExecute**接口执行算子，系统内部都会根据算子描述信息匹配内存中的模型。
   - 您需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、调用**aclopCreateHandle**接口创建一个Handle、再调用**aclopExecWithHandle**接口加载并执行算子。

     该方式下，在调用**aclopCreateHandle**接口时，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，每次调用**aclopExecWithHandle**接口执行算子时，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。Handle使用结束后，需调用**aclopDestroyHandle**接口释放。
4. 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**aclrtMemcpy**接口实现同步内存复制。

- 调用**aclrtMemcpyAsync**接口实现异步内存复制。

5. 按顺序先释放Stream资源，再释放Context资源，最后释放Device资源。
   - 调用**aclrtDestroyStream**接口释放Stream。

     不涉及显式创建Stream，使用默认Stream时，无需调用**aclrtDestroyStream**接口释放Stream。
   - 调用**aclrtDestroyContext**接口释放Context。

     不涉及显式创建Context，使用默认Context时，无需调用**aclrtDestroyContext**接口释放Context。
   - 调用**aclrtResetDevice**接口释放Device。
6. 调用**aclFinalize**接口实现ACL去初始化。

示例代码

请参见**6.3.5 数据后处理（调用单算子），回传结果到Host**。

7.7.3 自定义算子

7.7.3.1 固定 Shape 的算子加载与执行

1. 自定义算子的开发，请参见《TBE自定义算子开发指导》中的“算子开发流程”。
2. 完成算子开发后，单算子执行的调用流程，请参见**7.7.2 内置算子没有封装成ACL接口**。

7.7.3.2 动态 Shape 的算子加载与执行

在加载与执行动态Shape算子前，您需要参见《TBE自定义算子开发指导》中的“TIK自定义算子动态Shape专题”中的说明开发自定义算子以及生成对应的二进制文件。

基本原理

动态Shape场景下，算子加载与执行的流程如下：

1. 资源初始化，包括ACL初始化、设置单算子模型文件的加载目录、指定用于运算的Device等。
   - 调用**aclInit**接口实现ACL初始化。
   - 调用ACL接口注册要编译的自定义算子：
     - 调用**aclopRegisterCompileFunc**接口注册算子选择器（即选择Tiling策略的函数），用于在算子执行时，能针对不同Shape，选择相应的Tiling策略。

       算子选择器需由用户提前定义并实现：
       - 函数原型：

         ```
         typedef aclError (*aclopCompileFunc)(int numInputs, const aclTensorDesc *const inputDesc[], int numOutputs, const aclTensorDesc *const outputDesc[], const aclopAttr *opAttr, aclopKernelDesc *aclopKernelDesc);
         ```
       - 函数实现：

         用户自行编写代码逻辑实现Tiling策略选择、Tiling参数生成，并将调用**aclopSetKernelArgs**接口，设置算子Tiling参数、执行并发数等。

        - 调用**aclopCreateKernel**接口将算子注册到系统内部，用于在算子执行时，查找到算子实现代码。
    - 调用**aclrtSetDevice**接口指定运算的Device。
    - 调用**aclrtCreateContext**接口显式创建一个Context，调用**aclrtCreateStream**接口显式创建一个Stream。

      若没有显式创建Stream，则使用默认Stream，默认Stream是在调用**aclrtSetDevice**接口时隐式创建的，默认Stream作为接口入参时，直接传NULL。
2. 用户自行构造算子描述信息（输入输出Tensor描述、算子属性等）、申请存放算子输入输出数据的内存。
3. 将算子输入数据从Host复制到Device上。
    - 调用**aclrtMemcpy**接口实现同步内存复制，内存使用结束后需及时释放。
    - 调用**aclrtMemcpyAsync**接口实现异步内存复制，内存使用结束后需及时释放。
4. 编译单算子。

   调用**aclopUpdateParams**接口编译指定算子，触发算子选择器的调用逻辑。
5. 执行单算子。

   调用**aclopExecute**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**aclrtMemcpy**接口实现同步内存复制，内存使用结束后需及时释放。
    - 调用**aclrtMemcpyAsync**接口实现异步内存复制，内存使用结束后需及时释放。
7. 按顺序先释放Stream资源，再释放Context资源，最后释放Device资源。
    - 调用**aclrtDestroyStream**接口释放Stream。

      不涉及显式创建Stream，使用默认Stream时，无需调用**aclrtDestroyStream**接口释放Stream。
    - 调用**aclrtDestroyContext**接口释放Context。

      不涉及显式创建Context，使用默认Context时，无需调用**aclrtDestroyContext**接口释放Context。
    - 调用**aclrtResetDevice**接口释放Device。
8. 调用**aclFinalize**接口实现ACL去初始化。

示例代码

示例代码如下，您可以从“Acllib组件的安装目录/acllib/sample/acl_execute_op/acl_execute_batchnorm/src”目录下查看完整的样例代码。

```
#include "acl/acl.h"
//.....

//1.资源初始化
//此处是相对路径，相对可执行文件所在的目录
aclError ret = aclInit(nullptr);
aclopRegisterCompileFunc("BatchNorm", SelectAclopBatchNorm);
//将算子Kernel的*.o文件需要用户提前编译好，并调用用户自定义函数该文件加载到内存buffer中，length表示内存大小，如果有多个算子Kernel的*.o文件，需要多次调用该接口
aclopCreateKernel("BatchNorm", "tiling_mode_1__kernel0", "tiling_mode_1__kernel0",
                  buffer, length, ACL_ENGINE_AICORE, Deallocator);
```

```
//-----自定义函数BatchNormTest(n, c, h, w)，执行以下操作-----

//2.构造BatchNorm算子的输入输出Tensor、输入输出Tensor描述，并申请存放算子输入数据、输出数据的内存
aclTensorDesc *input_desc[3];
aclTensorDesc *output_desc[1];
input_desc[0] = aclCreateTensorDesc(ACL_FLOAT16, 4, shape_input, ACL_FORMAT_NCHW);
input_desc[1] = aclCreateTensorDesc(ACL_FLOAT16, 1, shape_gamma, ACL_FORMAT_ND);
input_desc[2] = aclCreateTensorDesc(ACL_FLOAT16, 1, shape_beta, ACL_FORMAT_ND);
output_desc[0] = aclCreateTensorDesc(ACL_FLOAT16, 4, shape_out, ACL_FORMAT_NCHW);

for (int i = 0; i < n * c * h * w; ++i) {
        input[i] = aclFloatToFloat16(1.0f);
    }

    for (int i = 0; i < c; ++i) {
        gamma[i] = aclFloatToFloat16(0.5f);
        beta[i] = aclFloatToFloat16(0.1f);
    }

aclrtMalloc(&devInput, size_input, ACL_MEM_MALLOC_NORMAL_ONLY);
aclrtMalloc(&devInput_gamma, size_gamma, ACL_MEM_MALLOC_NORMAL_ONLY);
aclrtMalloc(&devInput_beta, size_beta, ACL_MEM_MALLOC_NORMAL_ONLY);
aclrtMalloc(&devOutput, size_output, ACL_MEM_MALLOC_NORMAL_ONLY);

//3.将算子输入数据从Host复制到Device上
aclrtMemcpy(devInput, size_input, input, size_input, ACL_MEMCPY_HOST_TO_DEVICE);
aclrtMemcpy(devInput_gamma, size_gamma, gamma, size_gamma, ACL_MEMCPY_HOST_TO_DEVICE);
aclrtMemcpy(devInput_beta, size_beta, beta, size_beta, ACL_MEMCPY_HOST_TO_DEVICE);

//4.调用aclopUpdateParams接口编译算子
aclopUpdateParams("BatchNorm", 3, input_desc, 1, output_desc, nullptr, ACL_ENGINE_AICORE,
ACL_COMPILE_UNREGISTERED, nullptr));

//5.调用aclopExecute接口加载并执行算子
aclopExecute("BatchNorm", 3, input_desc, inputs, 1, output_desc, outputs, nullptr, stream);

//-----自定义函数BatchNormTest(n, c, h, w)，执行以上操作-----

//6.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）
aclrtMemcpy(output, size_output, devOutput, size_output, ACL_MEMCPY_DEVICE_TO_HOST);

//7.按顺序释放资源
//7.1 释放算子的输入输出Tensor描述
for (auto desc : input_desc) {
        aclDestroyTensorDesc(desc);
    }

for (auto desc : output_desc) {
        aclDestroyTensorDesc(desc);
    }
//7.2 释放Host上的内存
delete[]input;
delete[]gamma;
delete[]beta;
delete[]output;
//7.3 释放Device上的内存
aclrtFree(devInput);
aclrtFree(devInput_gamma);
aclrtFree(devInput_beta));
aclrtFree(devOutput);
//7.4 依次释放Stream、Context、Device资源，如果未显式创建Stream、Context，则无需释放
aclrtDestroyStream(stream);
aclrtDestroyContext(context);
aclrtResetDevice(deviceId);
aclFinalize();
```


8 ACL API 参考

8.1 简介

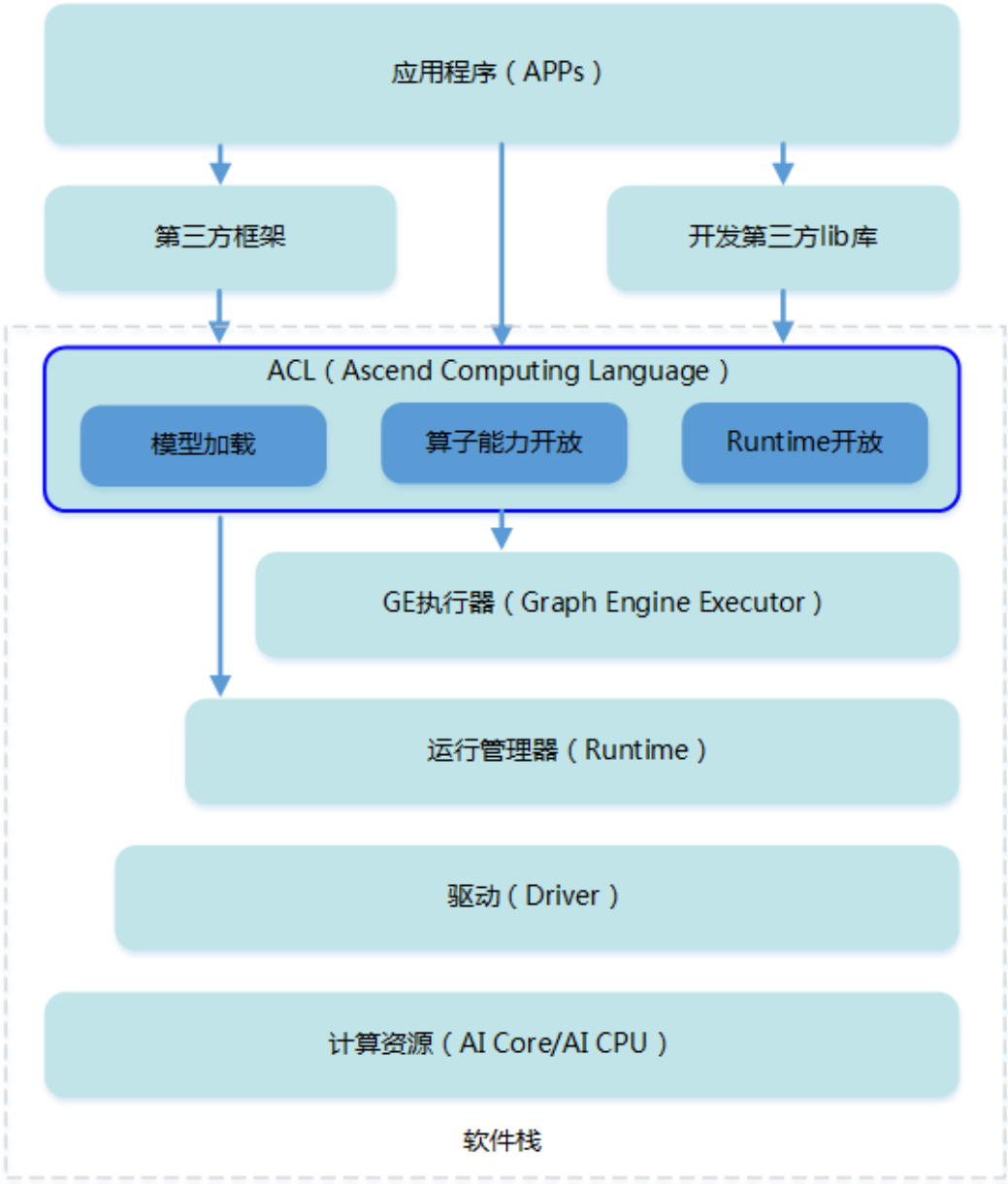
8.1.1 什么是 ACL

ACL（Ascend Computing Language）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++ 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。用户可以通过第三方框架调用ACL接口，以便使用昇腾AI处理器的计算能力；用户还可以使用ACL封装实现第三方lib库，以便提供昇腾AI处理器的运行管理、资源管理能力。

在运行应用时，ACL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。

图 8-1 逻辑架构图



8.1.2 基本概念

表 8-1 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指安装了芯片的硬件设备，利用PCIe接口与Host侧连接，为Host提供NN计算能力。
Context	Context作为一个容器，管理了所有对象（包括Stream、Event、设备内存等）的生命周期。不同Context的Stream、不同Context的Event是完全隔离的，无法建立同步等待关系。 Context分为两种： ● 默认Context：调用**aclrtSetDevice**接口指定用于运算的Device时，系统会自动隐式创建一个默认Context，一个Device对应一个默认Context，默认Context不能通过**aclrtDestroyContext**接口来释放。 ● 显式创建的Context：**推荐，**在进程或线程中调用**aclrtCreateContext**接口显式创建一个Context。
Stream	Stream用于维护一些异步操作的执行顺序，确保按照应用程序中的代码调用顺序在Device上执行。 基于Stream的kernel执行和数据传输能够实现Host运算操作、Host与Device间的数据传输、Device内的运算并行。 Stream分两种： ● 默认Stream：调用**aclrtSetDevice**接口指定用于运算的Device时，系统会自动隐式创建一个默认Stream，一个Device对应一个默认Stream，默认Stream不能通过**aclrtDestroyStream**接口来释放。 ● 显式创建的Stream：**推荐，**在进程或线程中调用**aclrtCreateStream**接口显式创建一个Stream。
Event	支持调用ACL接口同步Stream之间的任务，包括同步Host与Device之间的任务、Device与Device间的任务。 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（AI Preprocessing）用于在AI Core上完成图像预处理，包括色域转换（转换图像格式）、图像归一化（减均值/乘系数）和抠图（指定抠图起始点，抠出神经网络需要大小的图片）等。 AIPP区分为静态AIPP和动态AIPP。您只能选择静态AIPP或动态AIPP方式来处理图片，不能同时配置静态AIPP和动态AIPP两种方式。 ● 静态AIPP：模型转换时设置AIPP模式为静态，同时设置AIPP参数，模型生成后，AIPP参数值被保存在离线模型（*.om）中，每次模型推理过程采用固定的AIPP预处理参数（无法修改）。 如果使用静态AIPP方式，多Batch情况下共用同一份AIPP参数。 ● 动态AIPP：模型转换时仅设置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。设置动态AIPP参数值的接口请参见**8.15.17.2 设置动态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数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数是动态可变的。 ● 动态分辨率H*W: 用户执行推理时,每张图片的分辨率H*W是动态可变的。
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。

8.2 初始化与去初始化

8.2.1 aclInit

函数功能

ACL初始化函数，同步接口。

约束说明

- 一个进程内只能调用一次aclInit接口。
- 使用ACL接口开发应用时，必须先调用aclInit接口，否则可能会导致后续系统内部资源初始化出错，进而导致其它业务异常。

函数原型

aclError aclInit(const char *configPath)

参数说明

参数名	输入/输出	说明
configPath	输入	配置文件所在的路径，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导**》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aclInit接口中传入nullptr。

返回值说明

返回0表示成功，返回其它值表示失败。

配置文件示例

以Caffe ResNet-50网络为例，若需要比对Caffe ResNet-50网络与基于Caffe ResNet-50转换成的适配昇腾AI处理器的离线模型中某些层算子的输出结果，可以在配置文件中配置如下内容：

```
{
    "dump":{
        "dump_list":[
            {
                "model_name":"ResNet-50",
                "layer":[
                    "conv1conv1_relu",
                    "res2a_branch2ares2a_branch2a_relu",
                    "res2a_branch1",
                    "pool1"
                ]
            }
        ],
        "dump_path":"/MyApp20/dump",
            "dump_mode":"output"
    }
}
```

8.2.2 aclFinalize

函数功能

应用的进程退出前，必须显式调用该接口实现ACL去初始化，同步接口。

函数原型

aclError aclFinalize()

参数说明

无

返回值说明

返回0表示成功，返回其它值表示失败。

8.3 Device 管理

8.3.1 aclrtSetDevice

函数功能

指定用于运算的Device，同时隐式创建默认Context，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，同步接口。

如果多次调用aclrtSetDevice接口而不调用**aclrtResetDevice**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aclrtSetDevice接口和**aclrtResetDevice**接口配对使用，在不使用Device上资源时，通过调用**aclrtResetDevice**接口及时释放本进程使用的Device资源。

支持以下使用场景：

- 在不同进程或线程中可指定同一个Device用于运算。
- 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（**aclrtCreateContext**接口）。
- 多Device场景下，可在进程中通过aclrtSetDevice接口切换到其它Device。但利用Context切换（调用**aclrtSetCurrentContext**接口）来切换Device，比使用**aclrtSetDevice**接口效率高。

函数原型

aclError aclrtSetDevice(int32_t deviceId)

参数说明

参数名	输入/输出	说明
deviceId	输入	Device ID。 用户调用**aclrtGetDeviceCount**接口获取可用的Device数量后，这个Device ID的取值范围：[0, (可用的Device数量-1)]

返回值说明

返回0表示成功，返回其它值表示失败。

8.3.2 aclrtResetDevice

函数功能

复位当前运算的Device，释放Device上的资源，包括默认Context、默认Stream以及默认Context下创建的所有Stream，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。

约束说明

若要复位的Device上存在显式创建的Context、Stream、Event，在复位前，建议遵循如下接口调用顺序，否则可能会导致业务异常。

接口调用顺序：调用**aclrtDestroyEvent**接口释放Event/调用**aclrtDestroyStream**接口释放显式创建的Stream-->调用**aclrtDestroyContext**释放显式创建的Context-->调用aclrtResetDevice接口

函数原型

aclError aclrtResetDevice(int32_t deviceId)

参数说明

参数名	输入/输出	说明
deviceId	输入	Device ID。

返回值说明

返回0表示成功，返回其它值表示失败。

8.3.3 aclrtGetDevice

函数功能

获取当前正在使用的Device的ID，同步接口。

约束说明

如果没有调用**aclrtSetDevice**接口显式指定Device或没有调用**aclrtCreateContext**接口隐式指定Device，则调用aclrtGetDevice接口时，返回错误。

函数原型

aclError aclrtGetDevice(int32_t *deviceId)

参数说明

参数名	输入/输出	说明
deviceId	输出	Device ID。

返回值说明

返回0表示成功，返回其它值表示失败。

8.3.4 aclrtGetRunMode

函数功能

获取当前昇腾AI软件栈的运行模式，同步接口。

函数原型

aclError aclrtGetRunMode(aclrtRunMode *runMode)

参数说明

参数名	输入/输出	说明

runMode	输出	表示运行模式。
		typedef enum aclrtRunMode {
		//昇腾AI软件栈运行在Device的Control CPU上
		//昇腾AI软件栈运行在开发者板上
		ACL_DEVICE,
		//昇腾AI软件栈运行在Host CPU上
		ACL_HOST,
		} aclrtRunMode;

返回值说明

返回0表示成功，返回其它值表示失败。

8.3.5 aclrtSetTsDevice

函数功能

设置本次计算需要使用的Task Schedule，同步接口。

函数原型

aclError aclrtSetTsDevice(aclrtTsId tsId)

参数说明

参数名	输入/输出	说明

tsId	输入	指定本次计算需要使用的Task Schedule。如果昇腾AI软件栈中只有AICORE Task Schedule，则设置该参数无效，默认使用AICORE Task Schedule。
		typedef enum aclrtTsId {
		ACL_TS_ID_AICORE = 0, //使用AICORE Task Schedule
		ACL_TS_ID_AIVECTOR = 1, //使用AIVECTOR Task Schedule
		ACL_TS_ID_RESERVED = 2,
		} aclrtTsId;

返回值说明

返回0表示成功，返回其它值表示失败。

8.3.6 aclrtGetDeviceCount

函数功能

获取可用Device的数量，同步接口。

函数原型

aclError aclrtGetDeviceCount(uint32_t *count)

参数说明

参数名	输入/输出	说明
count	输出	Device数量。

返回值说明

返回0表示成功，返回其它值表示失败。

8.4 Context 管理

8.4.1 aclrtCreateContext

函数功能

显式创建一个Context，该Context中包含2个Stream，1个默认Stream和1个执行内部同步的Stream，同步接口。

支持以下使用场景：

- 若不调用aclrtCreateContext接口显式创建Context，那系统会使用默认Context，该默认Context是在调用**aclrtSetDevice**接口时隐式创建的。
  - 隐式创建Context：适合简单、无复杂交互逻辑的应用，但缺点在于，在多线程编程中，执行结果取决于线程调度的顺序。
  - 显式创建Context：**推荐显式**，适合大型、复杂交互逻辑的应用，且便于提高程序的可读性、可维护性。
- 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**8.5.1 aclrtCreateStream**），当前线程在同一时刻内只能使用其中一个Context，建议通过**aclrtSetCurrentContext**接口明确指定当前线程的Context，增加程序的可维护性。

说明

如果在程序中没有调用**aclrtSetDevice**接口，那么在首次调用aclrtCreateContext接口时，系统内部会根据该接口传入的Device ID，为该Device绑定一个默认Stream（一个Device仅绑定一个默认Stream），因此仅在首次调用aclrtCreateContext接口时，会占用3个Stream：Device上绑定的默认Stream、Context内的默认Stream、Context内的用于执行内部同步的Stream。

函数原型

aclError aclrtCreateContext(aclrtContext *context, int32_t deviceId)

参数说明

参数名	输入/输出	说明
deviceId	输入	指定需要创建context的Device的ID。
context	输出	创建的context的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

8.4.2 aclrtDestroyContext

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**aclrtCreateContext**接口创建的Context。

函数原型

aclError aclrtDestroyContext(aclrtContext context)

参数说明

参数名	输入/输出	说明
context	输入	指定需要销毁的context。

返回值说明

返回0表示成功，返回其它值表示失败。

8.4.3 aclrtSetCurrentContext

函数功能

设置线程的Context，同步接口。

支持以下场景：

- 如果在某线程（例如：thread1）中调用**aclrtCreateContext**接口显式创建一个Context（例如：ctx1），则可以不调用aclrtSetCurrentContext接口指定该线程的Context，系统默认将ctx1作为thread1的Context。
- 如果没有调用**aclrtCreateContext**接口显式创建Context，则系统将默认Context作为线程的Context，此时，不能通过**aclrtDestroyContext**接口来释放默认Context。
- 如果多次调用aclrtSetCurrentContext接口设置线程的Context，以最后一次为准。

约束说明

- 若给线程设置的Context所对应的Device已经被复位，则不能将该Context设置为线程的Context，否则会导致业务异常。
- 推荐在某一线程中创建的Context，在该线程中使用。若在线程A中调用**aclrtCreateContext**接口创建Context，在线程B中使用该Context，则需由用户自行保证两个线程中同一个Context下同一个Stream中任务执行的顺序。

函数原型

aclError aclrtSetCurrentContext(aclrtContext context)

参数说明

参数名	输入/输出	说明
context	输入	线程当前的context。

返回值说明

返回0表示成功，返回其它值表示失败。

8.4.4 aclrtGetCurrentContext

函数功能

获取线程的Context，同步接口。

如果用户多次调用aclrtSetCurrentContext接口设置当前线程的Context，则获取的是最后一次设置的Context。

函数原型

aclError aclrtGetCurrentContext(aclrtContext *context)

参数说明

参数名	输入/输出	说明
context	输出	线程当前的context。

返回值说明

返回0表示成功，返回其它值表示失败。

8.5 Stream 管理

8.5.1 aclrtCreateStream

函数功能

创建一个Stream，同步接口。

约束说明

硬件资源最多支持1024个Stream，如果已存在多个默认Stream，只能显式创建N个Stream（ N=1024-默认Stream个数-执行内部同步的Stream个数 ），例如，若已存在一个默认Stream和一个执行内部同步的Stream，则只能显式创建1022个Stream。

每个Context对应一个默认Stream，该默认Stream是调用aclrtSetDevice接口或aclrtCreateContext接口隐式创建的。推荐调用aclrtCreateStream接口显式创建Stream。

- 隐式创建Stream：适合简单、无复杂交互逻辑的应用，但缺点在于，在多线程编程中，执行结果取决于线程调度的顺序。
- 显式创建Stream：**推荐显式**，适合大型、复杂交互逻辑的应用，且便于提高程序的可读性、可维护性。

函数原型

aclError aclrtCreateStream(aclrtStream *stream)

参数说明

参数名	输入/输出	说明
stream	输出	新创建的stream的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

8.5.2 aclrtDestroyStream

函数功能

销毁指定Stream，只能销毁通过**aclrtCreateStream**接口创建的Stream，同步接口。

约束说明

- 在调用**aclrtDestroyStream**接口销毁指定Stream前，需要先调用**aclrtSynchronizeStream**接口确保Stream中的任务都已完成。
- 调用**aclrtDestroyStream**接口销毁指定Stream时，需确保该Stream在当前Context下

函数原型

aclError aclrtDestroyStream(aclrtStream stream)

参数说明

参数名	输入/输出	说明
stream	输入	待销毁的stream。

返回值说明

返回0表示成功，返回其它值表示失败。

8.6 同步等待

8.6.1 aclrtCreateEvent

函数功能

创建一个Event，同步接口。

函数原型

aclError aclrtCreateEvent(aclrtEvent *event)

参数说明

参数名	输入/输出	说明
event	输出	新创建的event的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

8.6.2 aclrtDestroyEvent

函数功能

销毁一个Event，只能销毁通过**aclrtCreateEvent**接口创建的Event，同步接口。销毁某个Event时，用户需确保等待**aclrtSynchronizeEvent**接口或**aclrtStreamWaitEvent**接口涉及的任务都结束后，再销毁。

函数原型

aclError aclrtDestroyEvent(aclrtEvent event)

参数说明

参数名	输入/输出	说明
event	输入	待销毁的event。

返回值说明

返回0表示成功，返回其它值表示失败。

8.6.3 aclrtRecordEvent

函数功能

在Stream中记录一个Event，同步接口。

aclrtRecordEvent接口与**aclrtStreamWaitEvent**接口配合使用时，主要用于多Stream之间同步的场景，在调用**aclrtRecordEvent**接口时，系统内部会申请Event资源，在调用**aclrtStreamWaitEvent**接口之后，请及时调用**aclrtResetEvent**接口释放Event资源。

接口调用顺序：**aclrtCreateEvent**-->**aclrtRecordEvent**-->**aclrtStreamWaitEvent**-->**aclrtResetEvent**

函数原型

aclError aclrtRecordEvent(aclrtEvent event, aclrtStream stream)

参数说明

参数名	输入/输出	说明
event	输入	待记录的event。
stream	输入	将该event记录在指定的stream中。 如果使用默认Stream，此处设置为NULL。

返回值说明

返回0表示成功，返回其它值表示失败。

8.6.4 aclrtResetEvent

函数功能

复位一个Event。用户需确保等待Stream中的任务都完成后，再复位Event，同步接口。

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**aclrtRecordEvent**接口时记录，因此在调用aclrtResetEvent接口前，需要先调用**aclrtRecordEvent**接口。

接口调用顺序：aclrtCreateEvent-->aclrtRecordEvent-->aclrtResetEvent

函数原型

aclError aclrtResetEvent(aclrtEvent event, aclrtStream stream)

参数说明

参数名	输入/输出	说明
event	输入	待复位的event。
stream	输入	指定event所在的stream。

返回值说明

返回0表示成功，返回其它值表示失败。

8.6.5 aclrtQueryEvent

函数功能

查询指定Event的状态，同步接口。

函数原型

aclError aclrtQueryEvent(aclrtEvent event, aclrtEventStatus *status)

参数说明

参数名	输入/输出	说明
event	输入	指定待查询的event。
status	输出	event的状态。

返回值说明

返回0表示成功，返回其它值表示失败。

8.6.6 aclrtSynchronizeEvent

函数功能

阻塞应用程序运行，等待Event完成，同步接口。

函数原型

aclError aclrtSynchronizeEvent(aclrtEvent event)

参数说明

参数名	输入/输出	说明
event	输入	指定需等待的event。

返回值说明

返回0表示成功，返回其它值表示失败。

8.6.7 aclrtEventElapsedTime

函数功能

统计两个Event之间的耗时，同步接口。

约束说明

接口调用顺序：调用**aclrtCreateEvent**接口创建Event-->调用**aclrtRecordEvent**接口在同一个Stream中记录起始Event、结尾Event-->调用**aclrtSynchronizeStream**接口阻塞应用程序运行，直到指定Stream中的所有任务都完成-->调用aclrtEventElapsedTime接口统计两个Event之间的耗时

函数原型

aclError aclrtEventElapsedTime(float *ms, aclrtEvent start, aclrtEvent end)

参数说明

参数名	输入/输出	说明
ms	输出	表示两个Event之间的耗时，单位为毫秒。
start	输入	起始Event。
end	输入	结尾Event。

返回值说明

返回0表示成功，返回其它值表示失败。

8.6.8 aclrtStreamWaitEvent

函数功能

阻塞指定Stream的运行，直到指定的Event完成，同步接口。支持多个Stream等待同一个Event的场景。

aclrtRecordEvent接口与**aclrtStreamWaitEvent**接口配合使用时，主要用于多Stream之间同步的场景，在调用**aclrtRecordEvent**接口时，系统内部会申请Event资源，在调用**aclrtStreamWaitEvent**接口之后，请及时调用**aclrtResetEvent**接口释放Event资源。

接口调用顺序：**aclrtCreateEvent**-->**aclrtRecordEvent**-->**aclrtStreamWaitEvent**-->**aclrtResetEvent**

函数原型

aclError aclrtStreamWaitEvent(aclrtStream stream, aclrtEvent event)

参数说明

参数名	输入/输出	说明
stream	输入	指定需要等待event完成的stream。 如果使用默认Stream，此处设置为NULL。
event	输入	需等待的event。

返回值说明

返回0表示成功，返回其它值表示失败。

8.6.9 aclrtSynchronizeDevice

函数功能

阻塞应用程序运行，直到正在运算中的Device完成运算，同步接口。

多Device场景下，调用该接口等待的是当前Context对应的Device。

函数原型

aclError aclrtSynchronizeDevice(void)

参数说明

无

返回值说明

返回0表示成功，返回其它值表示失败。

8.6.10 aclrtSynchronizeStream

函数功能

阻塞应用程序运行，直到指定Stream中的所有任务都完成，同步接口。

函数原型

aclError aclrtSynchronizeStream(aclrtStream stream)

参数说明

参数名	输入/输出	说明
stream	输入	指定需要完成所有任务的stream。

返回值说明

返回0表示成功，返回其它值表示失败。

8.6.11 aclrtSubscribeReport

函数功能

指定处理Stream上回调函数的线程。同步接口。

函数原型

aclError aclrtSubscribeReport(uint64_t threadId, aclrtStream stream)

参数说明

参数名	输入/输出	说明
threadId	输入	指定线程的ID。
stream	输入	指定Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

8.6.12 aclrtLaunchCallback

函数功能

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。异步接口。

函数原型

aclError aclrtLaunchCallback(aclrtCallback fn, void *userData, aclrtCallbackBlockType blockType, aclrtStream stream)

参数说明

参数名	输入/输出	说明
fn	输入	指定要增加的回调函数。 回调函数的函数原型为： typedef void (*aclrtCallback)(void *userData)
userData	输入	待传递给回调函数的用户数据。

参数名	输入/输出	说明
blockType	输入	指定回调函数调用是否阻塞Device运行。 typedef enum aclrtCallbackBlockType { ACL_CALLBACK_NO_BLOCK, //非阻塞 ACL_CALLBACK_BLOCK, //阻塞 } aclrtCallbackBlockType;
stream	输入	指定Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

8.6.13 aclrtProcessReport

函数功能

等待指定时间后，触发回调处理，由**aclrtSubscribeReport**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aclrtProcessReport接口。

函数原型

aclError aclrtProcessReport(int32_t timeout)

参数说明

参数名	输入/输出	说明
timeout	输入	超时时间，单位为ms。 取值范围： • -1：表示无限等待 • 大于0（不包含0）：表示等待的时间

返回值说明

返回0表示成功，返回其它值表示失败。

8.6.14 aclrtUnSubscribeReport

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。

函数原型

aclError aclrtUnSubscribeReport(uint64_t threadId, aclrtStream stream)

参数说明

参数名	输入/输出	说明
threadId	输入	指定线程的ID。
stream	输入	指定Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

8.6.15 aclrtSetExceptionInfoCallback

函数功能

设置异常回调函数，同步接口。

- 您需要在执行任务之前，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以及线程ID的aclrtExceptionInfo结构体指针，并执行回调函数，用户可以再分别调用**aclrtGetTaskIdFromExceptionInfo**、**aclrtGetStreamIdFromExceptionInfo**、**aclrtGetThreadIdFromExceptionInfo**接口获取产生异常的任务ID、Stream ID以及线程ID，便于定位问题。
- 如果多次设置异常回调函数，以最后一次设置为准。
- 如果想清空回调函数，可调用aclrtSetExceptionInfoCallback接口，将入参设置为空指针。

使用场景举例：例如，在调用**aclopExecute**接口前，调用aclrtSetExceptionInfoCallback接口设置异常回调函数，当算子在Device执行异常时，系统会向用户设置的异常回调函数中传入一个包含任务ID、Stream ID以及线程ID的aclrtExceptionInfo结构体指针，并执行回调函数。

函数原型

aclError aclrtSetExceptionInfoCallback(aclrtExceptionInfoCallback callback)

参数说明

参数名	输入/输出	说明
callback	输入	指定要注册的回调函数。 回调函数的函数原型为： typedef void (*aclrtExceptionInfoCallback)(aclrtExceptionInfo *exceptionInfo);

返回值说明

返回0表示成功，返回其它值表示失败。

8.6.16 aclrtGetTaskIdFromExceptionInfo

函数功能

获取异常信息中的任务ID，同步接口。

函数原型

uint32_t aclrtGetTaskIdFromExceptionInfo(const aclrtExceptionInfo *info)

参数说明

参数名	输入/输出	说明
info	输入	指定异常信息。 在执行异步任务之前调用**aclrtSetExceptionInfoCallback**接口，系统会将产生异常的任务ID、Stream ID以及线程ID存放在aclExceptionInfo结构体中。

返回值说明

返回异常信息中的任务ID，返回值为0xFFFFFFFF（以十六进制为例）时表示异常信息为空。

8.6.17 aclrtGetStreamIdFromExceptionInfo

函数功能

获取异常信息中的Stream ID，同步接口。

函数原型

uint32_t aclrtGetStreamIdFromExceptionInfo(const aclrtExceptionInfo *info)

参数说明

参数名	输入/输出	说明
info	输入	指定异常信息。 在执行异步任务之前调用aclrtSetExceptionInfoCallback接口，系统会将产生异常的任务ID、Stream ID以及线程ID存放在aclExceptionInfo结构体中。

返回值说明

返回异常信息中的Stream ID，返回值为0xFFFFFFFF（以十六进制为例）时表示异常信息为空。

8.6.18 aclrtGetThreadIdFromExceptionInfo

函数功能

获取异常信息中的线程ID，同步接口。

函数原型

uint32_t aclrtGetThreadIdFromExceptionInfo(const aclrtExceptionInfo *info)

参数说明

参数名	输入/输出	说明
info	输入	指定异常信息。 在执行异步任务之前调用aclrtSetExceptionInfoCallback接口，系统会将产生异常的任务ID、Stream ID以及线程ID存放在aclExceptionInfo结构体中。

返回值说明

返回异常信息中的线程ID，返回值为0xFFFFFFFF（以十六进制为例）时表示异常信息为空。

8.7 内存管理

8.7.1 aclrtMalloc

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*devPtr返回已分配内存的指针。

调用**媒体数据处理**的接口前，若需要申请Device上的内存存放输入或输出数据，需调用**acldvppMalloc**申请内存。

约束说明

- 使用aclrtMalloc接口申请的内存，需要通过**aclrtFree**接口释放内存。
- 频繁调用aclrtMalloc接口申请内存、调用**aclrtFree**接口释放内存，会损耗性能，建议用户提前做内存预先分配或二次管理，避免频繁申请/释放内存。
- 调用aclrtMalloc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。

函数原型

aclError aclrtMalloc(void **devPtr, size_t size, aclrtMemMallocPolicy policy)

参数说明

参数名	输入/输出	说明
size	输入	申请内存的大小，单位Byte。 size不能为0。
policy	输入	内存分配规则。 typedef enum aclrtMemMallocPolicy { ACL_MEM_MALLOC_HUGE_FIRST, // 优先申请大页内存，如果大页内存不够，则使用普通页的内存 ACL_MEM_MALLOC_HUGE_ONLY, // 仅申请大页，如果大页内存不够，则返回错误 ACL_MEM_MALLOC_NORMAL_ONLY, // 仅申请普通页 } aclrtMemMallocPolicy;
devPtr	输出	指向Device上已分配内存的指针的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

8.7.2 aclrtFree

函数功能

释放Device上的内存，同步接口。

aclrtFree接口只能释放通过aclrtMalloc接口申请的内存。

函数原型

aclError aclrtFree(void *devPtr)

参数说明

参数名	输入/输出	说明
devPtr	输入	待释放内存的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

8.7.3 aclrtMallocHost

函数功能

申请Host上的内存，同步接口。

Control CPU开放形态下，应用程序运行在Device的Control CPU上时，该接口会返回错误。

约束说明

- 申请的内存不能在Device使用，需要显式拷贝到Device。
- 使用aclrtMallocHost接口申请的内存，需要通过aclrtFreeHost接口释放内存。

函数原型

aclError aclrtMallocHost(void **hostPtr, size_t size)

参数说明

参数名	输入/输出	说明
size	输入	申请内存的大小，单位Byte。 size不能为0。
hostPtr	输出	“已分配内存的指针”的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

8.7.4 aclrtFreeHost

函数功能

释放Host上的内存，同步接口。

aclrtFreeHost接口只能释放通过**aclrtMallocHost**接口申请的内存。

Control CPU开放形态下，应用程序运行在Device的Control CPU上时，该接口会返回错误。

函数原型

aclError aclrtFreeHost(void *hostPtr)

参数说明

参数名	输入/输出	说明
hostPtr	输入	待释放内存的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

8.7.5 aclrtMemset

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。

要初始化的内存都在Host侧或Device侧，系统根据地址判定是Host还是Device。

函数原型

aclError aclrtMemset (void *devPtr, size_t maxCount, int32_t value, size_t count)

参数说明

参数名	输入/输出	说明
devPtr	输入	内存的起始地址。
maxCount	输入	内存的最大长度，单位Byte。
value	输入	设置的值。
count	输入	需要设置为指定值的内存长度，单位Byte。

返回值说明

返回0表示成功，返回其它值表示失败。

8.7.6 aclrtMemsetAsync

函数功能

初始化内存，将内存中的内容设置为指定的值，异步接口。

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**aclrtSynchronizeStream**接口确保内存初始化的任务已执行完成。

要初始化的内存都在Host侧或Device侧，系统根据地址判定是Host还是Device。

函数原型

aclError aclrtMemsetAsync(void *devPtr, size_t maxCount, int32_t value, size_t count, aclrtStream stream)

参数说明

参数名	输入/输出	说明
devPtr	输入	内存的起始地址。
maxCount	输入	内存的最大长度，单位Byte。
value	输入	设置的值。
count	输入	需要设置为指定值的内存长度，单位Byte。
stream	输入	stream ID。

返回值说明

返回0表示成功，返回其它值表示失败。

8.7.7 aclrtMemcpy

函数功能

实现Host内、Host与Device之间、Device内的同步内存复制。

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。

Control CPU开放形态下，应用程序运行在Device的Control CPU上时，该接口仅支持Device内的内存复制。

函数原型

aclError aclrtMemcpy(void *dst, size_t destMax, const void *src, size_t count, aclrtMemcpyKind kind)

参数说明

参数名	输入/输出	说明
dst	输入	目的内存地址指针。
destMax	输入	目的内存地址的最大内存长度，单位Byte。
src	输入	源内存地址指针。
count	输入	内存复制的长度，单位Byte。
kind	输入	内存复制的类型，预留参数，当前不支持。 typedef enum aclrtMemcpyKind { ACL_MEMCPY_HOST_TO_HOST, // Host内的内存复制 ACL_MEMCPY_HOST_TO_DEVICE, // Host到Device的内存复制 ACL_MEMCPY_DEVICE_TO_HOST, // Device到Host的内存复制 ACL_MEMCPY_DEVICE_TO_DEVICE, // Device内的内存复制 } aclrtMemcpyKind;

返回值说明

返回0表示成功，返回其它值表示失败。

8.7.8 aclrtMemcpyAsync

函数功能

实现Host内、Host与Device之间、Device内的异步内存复制。

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**aclrtSynchronizeStream**接口确保内存复制的任务已执行完成。

Control CPU开放形态下，应用程序运行在Device的Control CPU上时，该接口仅支持Device内的内存复制。

约束说明

如果是片内Device到Device的内存复制，源地址和目的地址都必须64字节对齐。

函数原型

aclError aclrtMemcpyAsync(void *dst, size_t destMax, const void *src, size_t count, aclrtMemcpyKind kind, aclrtStream stream)

参数说明

参数名	输入/输出	说明
dst	输入	目的内存地址指针。
destMax	输入	目的内存地址的最大内存长度，单位Byte。
src	输入	源内存地址指针。
count	输入	内存复制的长度，单位Byte。
kind	输入	内存复制的类型。 typedef enum aclrtMemcpyKind { ACL_MEMCPY_HOST_TO_HOST, // Host内的内存复制，当前不支持 ACL_MEMCPY_HOST_TO_DEVICE, // Host到Device的内存复制 ACL_MEMCPY_DEVICE_TO_HOST, // Device到Host的内存复制 ACL_MEMCPY_DEVICE_TO_DEVICE, // Device内的内存复制 } aclrtMemcpyKind;
stream	输入	stream ID。

返回值说明

返回0表示成功，返回其它值表示失败。

8.8 模型加载与执行

8.8.1 aclmdlLoadFromFile

函数功能

从文件加载离线模型数据（适配昇腾AI处理器的离线模型），由系统内部管理内存，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。

函数原型

aclError aclmdlLoadFromFile(const char *modelPath, uint32_t *modelId)

参数说明

参数名	输入/输出	说明
modelPath	输入	离线模型文件的存储路径，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。
modelId	输出	系统完成模型加载后生成的模型ID。

返回值说明

返回0表示成功，返回其它值表示失败。

8.8.2 aclmdlLoadFromMem

函数功能

从内存加载离线模型数据，由系统内部管理模型运行的内存，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。

函数原型

aclError aclmdlLoadFromMem(const void* model, size_t modelSize, uint32_t* modelId)

参数说明

参数名	输入/输出	说明
model	输入	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**aclrtGetRunMode**接口获取。
modelSize	输入	内存中的模型数据长度，单位Byte。
modelId	输出	系统完成模型加载后生成的模型ID。

返回值说明

返回0表示成功，返回其它值表示失败。

8.8.3 aclmdlLoadFromFileWithMem

函数功能

从文件加载离线模型数据（适配昇腾AI处理器的离线模型），由用户自行管理模型运行的内存，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。

函数原型

aclError aclmdlLoadFromFileWithMem(const char *modelPath, uint32_t *modelId, void *workPtr, size_t workSize, void *weightPtr, size_t weightSize)

参数说明

参数名	输入/输出	说明
modelPath	输入	离线模型文件的存储路径，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。
modelId	输出	系统完成模型加载后生成的模型ID。
workPtr	输入	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。
workSize	输入	模型所需工作内存的大小，单位Byte。
weightPtr	输入	Device上模型权值内存（存放权值数据）的指针，由用户自行管理。
weightSize	输入	模型所需权值内存的大小，单位Byte。

返回值说明

返回0表示成功，返回其它值表示失败。

8.8.4 aclmdlLoadFromMemWithMem

函数功能

从内存加载离线模型数据，由用户自行管理模型运行的内存，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。

函数原型

aclError aclmdlLoadFromMemWithMem(const void *model, size_t modelSize, uint32_t *modelId, void *workPtr, size_t workSize, void *weightPtr, size_t weightSize)

参数说明

参数名	输入/输出	说明
model	输入	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aclrtGetRunMode接口获取。
modelSize	输入	模型数据长度，单位Byte。
modelId	输出	系统完成模型加载后生成的模型ID。
workPtr	输入	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，可以为空。
workSize	输入	模型所需工作内存的大小，单位Byte。workPtr为空时无效。
weightPtr	输入	Device上模型权值内存（存放权值数据）的指针，由用户自行管理，可以为空。
weightSize	输入	模型所需权值内存的大小，单位Byte。weightPtr为空时无效。

返回值说明

返回0表示成功，返回其它值表示失败。

8.8.5 aclmdlLoadFromFileWithQ

函数功能

从文件加载离线模型数据（适配昇腾AI处理器的离线模型），模型的输入、输出数据都存放在队列中，同步接口。**当前版本不支持该接口。**

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。

函数原型

aclError aclmdlLoadFromFileWithQ(const char *modelPath, uint32_t *modelId, const uint32_t *inputQ,

size_t inputQNum, const uint32_t *outputQ, size_t outputQNum)

参数说明

参数名	输入/输出	说明
modelPath	输入	离线模型文件的存储路径，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。
modelId	输出	系统完成模型加载后生成的模型ID。
inputQ	输入	队列ID，一个模型的输入对应一个队列ID。
inputQNum	输入	输入队列大小。
outputQ	输入	队列ID，一个模型的输出对应一个队列ID。
outputQNum	输入	输出队列大小。

返回值说明

返回0表示成功，返回其它值表示失败。

8.8.6 aclmdlLoadFromMemWithQ

函数功能

从内存加载离线模型数据（适配昇腾AI处理器的离线模型），模型的输入、输出数据都存放在队列中，同步接口。**当前版本不支持该接口。**

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。

函数原型

aclError aclmdlLoadFromMemWithQ(const void *model, size_t modelSize, uint32_t *modelId,

const uint32_t *inputQ, size_t inputQNum, const uint32_t *outputQ, size_t outputQNum)

参数说明

参数名	输入/输出	说明
model	输入	模型数据的内存地址。
modelSize	输入	内存中的模型数据长度，单位Byte。
modelId	输出	系统完成模型加载后生成的模型ID。

参数名	输入/输出	说明
inputQ	输入	队列ID，一个模型的输入对应一个队列ID。
inputQNum	输入	输入队列大小。
outputQ	输入	队列ID，一个模型的输出对应一个队列ID。
outputQNum	输入	输出队列大小。

返回值说明

返回0表示成功，返回其它值表示失败。

8.8.7 aclmdlExecute

函数功能

执行模型推理，直到返回推理结果，同步接口。

函数原型

aclError aclmdlExecute(uint32_t modelId, const aclmdlDataset *input, aclmdlDataset *output)

参数说明

参数名	输入/输出	说明
modelId	输入	指定需要执行推理的模型的ID。 调用**aclmdlLoadFromFile**接口/**aclmdlLoadFromMem**接口/**aclmdlLoadFromFileWithMem**接口/**aclmdlLoadFromMemWithMem**接口加载模型成功后，会返回模型ID。
input	输入	模型推理的输入数据。 若用户使用**aclrtMalloc**接口申请大块内存并自行划分、管理内存时，用户在管理内存时，模型输入数据的内存有对齐和补齐要求，首地址128字节对齐，对齐后再加32字节，然后再向上对齐到128字节。

参数名	输入/输出	说明
output	输出	模型推理的输出数据。 若用户使用aclrtMalloc接口申请大块内存并自行划分、管理内存时，用户在管理内存时，模型输出数据的内存有对齐和补齐要求，首地址128字节对齐，对齐后再加32字节，然后再向上对齐到128字节。

返回值说明

返回0表示成功，返回其它值表示失败。

8.8.8 aclmdlExecuteAsync

函数功能

执行模型推理，异步接口。

函数原型

aclError aclmdlExecuteAsync(uint32_t modelId, const aclmdlDataset *input, aclmdlDataset *output, aclrtStream stream)

参数说明

参数名	输入/输出	说明
modelId	输入	指定需要执行推理的模型的ID。 调用aclmdlLoadFromFile接口/aclmdlLoadFromMem接口/aclmdlLoadFromFileWithMem接口/aclmdlLoadFromMemWithMem接口加载模型成功后，会返回模型ID。
input	输入	模型推理的输入数据。 若用户使用aclrtMalloc接口申请大块内存并自行划分、管理内存时，用户在管理内存时，模型输入数据的内存有对齐和补齐要求，首地址128字节对齐，对齐后再加32字节，然后再向上对齐到128字节。

参数名	输入/输出	说明
output	输出	模型推理的输出数据。 若用户使用aclrtMalloc接口申请大块内存并自行划分、管理内存时，用户在管理内存时，模型输出数据的内存有对齐和补齐要求，首地址128字节对齐，对齐后再加32字节，然后再向上对齐到128字节。
stream	输入	指定stream。

返回值说明

返回0表示成功，返回其它值表示失败。

8.8.9 aclmdlUnload

函数功能

系统完成模型推理后，可调用该接口卸载模型，释放资源，同步接口。

函数原型

aclError aclmdlUnload(uint32_t modelId)

参数说明

参数名	输入/输出	说明
modelId	输入	指定需卸载的模型的ID。

返回值说明

返回0表示成功，返回其它值表示失败。

8.8.10 aclmdlQuerySize

函数功能

根据模型文件获取模型执行时所需的权值内存大小、工作内存大小，同步接口。

函数原型

aclError aclmdlQuerySize(const char *fileName, size_t *workSize, size_t *weightSize)

参数说明

参数名	输入/输出	说明
fileName	输入	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。
workSize	输出	模型执行时所需的工作内存的大小，单位Byte。
weightSize	输出	模型执行时所需权值内存的大小，单位Byte。

返回值说明

返回0表示成功，返回其它值表示失败。

8.8.11 aclmdlQuerySizeFromMem

函数功能

根据内存中的模型数据获取模型执行时所需的权值内存大小、内存大小，同步接口。

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

函数原型

aclError aclmdlQuerySizeFromMem(const void *model, size_t modelSize, size_t *workSize, size_t *weightSize)

参数说明

参数名	输入/输出	说明
model	输入	需要获取内存信息的模型数据。
modelSize	输入	模型数据长度，单位Byte。
workSize	输出	模型执行时所需的工作内存的大小，单位Byte。
weightSize	输出	模型执行时所需权值内存的大小，单位Byte。

返回值说明

返回0表示成功，返回其它值表示失败。

8.8.12 aclmdlSetDynamicBatchSize

函数功能

在动态Batch场景下，用于设置模型推理时的批量大小Batch（每次处理图片的数量）。

此处设置的Batch数只能是模型转换时通过dynamic_batch_size参数设置的Batch档位中的某一个，模型转换的详细说明请参见《**ATC工具使用指导**》。

也可以调用**aclmdlGetDynamicBatch**接口获取指定模型支持的Batch档位数以及每一档中的Batch数。

函数原型

aclError aclmdlSetDynamicBatchSize(uint32_t modelId, aclmdlDataset *dataset, size_t index, uint64_t batchSize)

参数说明

参数名	输入/输出	说明
modelId	输入	模型ID。 调用**aclmdlLoadFromFile**接口/**aclmdlLoadFromMem**接口/**aclmdlLoadFromFileWithMem**接口/**aclmdlLoadFromMemWithMem**接口加载模型成功后，会返回模型ID。
dataset	输入	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	输入	标识动态Batch输入的输入index，需调用**aclmdlGetInputIndexByName**接口获取，输入名称固定为ACL_DYNAMIC_TENSOR_NAME。
batchSize	输入	指定模型推理时的批量大小Batch。

返回值说明

返回0表示成功，返回其它值表示失败。

8.8.13 aclmdlSetDynamicHWSize

函数功能

设置模型推理时输入图片的高和宽。

此处设置的分辨率只能是模型转换时通过dynamic_image_size参数设置的分辨率档位中的某一个，模型转换的详细说明请参见《**ATC工具使用指导**》。

也可以调用**aclmdlGetDynamicHW**接口获取指定模型支持的分辨率档位数以及每一档中的宽、高。

函数原型

aclError aclmdlSetDynamicHWSize(uint32_t modelId, aclmdlDataset *dataset, size_t index, uint64_t height, uint64_t width)

参数说明

参数名	输入/输出	说明
modelId	输入	模型ID。 调用**aclmdlLoadFromFile**接口/**aclmdlLoadFromMem**接口/**aclmdlLoadFromFileWithMem**接口/**aclmdlLoadFromMemWithMem**接口加载模型成功后，会返回模型ID。
dataset	输入	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	输入	标识动态分辨率输入的输入index，需调用**aclmdlGetInputIndexByName**接口获取，输入名称固定为ACL_DYNAMIC_TENSOR_NAME。
height	输入	需设置的H值。
width	输入	需设置的W值。

返回值说明

返回0表示成功，返回其它值表示失败。

8.8.14 aclmdlSetInputAIPP

函数功能

动态AIPP场景下，设置模型推理时的AIPP参数值，同步接口。

函数原型

aclError aclmdlSetInputAIPP(uint32_t modelId, aclmdlDataset *dataset, size_t index, const aclmdlAIPP *aippParmsSet)

参数说明

参数名	输入/输出	说明
modelId	输入	模型的ID。 调用**aclmdlLoadFromFile**接口/**aclmdlLoadFromMem**接口/**aclmdlLoadFromFileWithMem**接口/**aclmdlLoadFromMemWithMem**接口加载模型成功后，会返回模型ID。
dataset	输入	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	输入	标识动态AIPP输入的输入index，需调用**aclmdlGetInputIndexByName**接口获取，输入名称固定为ACL_DYNAMIC_AIPP_NAME。
aippParmsSet	输入	动态AIPP参数对象。 提前调用**aclmdlCreateAIPP**接口创建aclmdlAIPP类型的数据。

返回值说明

返回0表示成功，返回其它值表示失败。

8.9 算子编译

8.9.1 aclopRegisterCompileFunc

函数功能

动态Shape场景下，注册算子选择器，用于在算子执行时，能针对不同shape，选择相应的Tiling策略。

如果某算子已注册算子选择器，则不允许重新注册，如果需要变更算子选择器，必须先调用**aclopUnregisterCompileFunc**接口取消注册，然后再调用aclopRegisterCompileFunc接口重新注册。

约束说明

无。

函数原型

aclError aclopRegisterCompileFunc(const char *opType, aclopCompileFunc func)

参数说明

参数名	输入/输出	说明
opType	输入	算子类型。
func	输入	算子选择器回调函数，函数定义： typedef aclError (*aclopCompileFunc)(int numInputs, const aclTensorDesc *const inputDesc[], int numOutputs, const aclTensorDesc *const outputDesc[], const aclopAttr *opAttr, aclopKernelDesc *aclopKernelDesc);

返回值说明

返回0表示成功，返回其它值表示失败。

8.9.2 aclopUnregisterCompileFunc

函数功能

动态Shape场景下，取消注册算子选择器。

函数原型

aclError aclopUnregisterCompileFunc(const char *opType)

参数说明

参数名	输入/输出	说明
opType	输入	算子类型。

返回值说明

返回0表示成功，返回其它值表示失败。

8.9.3 aclopCreateKernel

函数功能

动态Shape场景下，将算子注册到系统内部，运行算子时使用。

约束说明

无。

函数原型

aclError aclopCreateKernel(const char *opType,

const char *kernelId,

const char *kernelName,

void *binData,

int binSize,

aclopEngineType enginetype,

aclDataDeallocator deallocator)

参数说明

参数名	输入/输出	说明
opType	输入	算子类型。
kernelId	输入	算子执行时要指定的Kernel ID。
kernelName	输入	算子Kernel名称，和算子二进制文件中的kernelName保持一致。
binData	输入	算子Kernel文件的内存地址。
binSize	输入	算子Kernel文件的内存大小，单位为byte。
enginetype	输入	表示算子执行引擎，该参数只有**aclopUpdateParams**接口的**compileFlag**参数值为ACL_COMPILE_SYS时有效。 typedef enum aclEngineType { ACL_ENGINE_SYS, //不关心具体执行引擎时填写 ACL_ENGINE_AICORE, //将算子编译成AI Core算子 ACL_ENGINE_VECTOR, //将算子编译成Vector Core算子 } aclopEngineType;
deallocator	输入	释放binData内存的回调函数，调用者根据binData的构造方法设置对应的释放函数。如果数据由调用者释放则设置为空。 函数签名为： typedef void (*aclDataDeallocator)(void *data, size_t length);

返回值说明

返回0表示成功，返回其它值表示失败。

8.9.4 aclopSetKernelArgs

函数功能

动态Shape场景下，设置算子Tiling参数、执行并发数。

约束说明

无。

函数原型

aclError aclopSetKernelArgs(aclopKernelDesc *kernelDesc,

const char *kernelId,

uint32_t blockDim,

const void *args,

uint32_t argSize)

参数说明

参数名	输入/输出	说明
kernelDesc	输入	Kernel描述缓存，aclopKernelDesc类型的指针。 typedef struct aclopKernelDesc aclopKernelDesc;
kernelId	输入	算子执行时要指定的Kernel ID，与调用**aclopCreateKernel**时传递的kernelId一致。
blockDim	输入	Kernel执行的并发数。
args	输入	Tiling参数。
argSize	输入	Tiling参数内存大小，单位为Byte。

返回值说明

返回0表示成功，返回其它值表示失败。

8.9.5 aclopSetKernelWorkspaceSizes

函数功能

动态Shape场景下，设置算子Workspace参数。

约束说明

为非必须接口，根据算子情况可选。

函数原型

aclError aclopSetKernelWorkspaceSizes(aclopKernelDesc *kernelDesc, int numWorkspaces, size_t *workspaceSizes)

参数说明

参数名	输入/输出	说明
kernelDesc	输入	Kernel描述缓存，aclopKernelDesc类型的指针。 typedef struct aclopKernelDesc aclopKernelDesc
numWorkspaces	输入	Workspaces个数。
workspaceSizes	输入	Workspaces大小的数组地址。

返回值说明

返回0表示成功，返回其它值表示失败。

8.9.6 aclopUpdateParams

函数功能

在算子动态Shape场景下，根据指定算子，触发调用算子选择器。

约束说明

无。

函数原型

aclError aclopUpdateParams(const char *opType,

int numInputs,

const aclTensorDesc *const inputDesc[],

int numOutputs,

const aclTensorDesc *const outputDesc[],

const aclopAttr *attr)

参数说明

参数名	输入/输出	说明
opType	输入	算子类型名称。

参数名	输入/输出	说明
numInputs	输入	算子输入tensor的数量。
inputDesc	输入	算子输入tensor的描述。
numOutputs	输入	算子输出tensor的数量。
outputDesc	输入	算子输出tensor的描述。
attr	输入	算子属性。

返回值说明

返回0表示成功，返回其它值表示失败。

8.9.7 aclopCompile

须知

该接口在“FwkACLlib组件的安装目录/fwkacllib/include/acl/acl_op_compiler.h”头文件中定义。后续特性的预留接口，当前不推荐使用。

函数功能

编译指定算子。

约束说明

无。

函数原型

aclError aclopCompile(const char *opType,

int numInputs,

const aclTensorDesc *const inputDesc[],

int numOutputs,

const aclTensorDesc *const outputDesc[],

const aclopAttr *attr,

aclopEngineType engineType,

aclopCompileType compileFlag,

const char* opPath)

参数说明

参数名	输入/输出	说明
opType	输入	算子类型名称。
numInputs	输入	算子输入tensor的数量。
inputDesc	输入	算子输入tensor的描述。
numOutputs	输入	算子输出tensor的数量。
outputDesc	输入	算子输出tensor的描述。
attr	输入	算子属性。
engineType	输入	算子执行引擎。 typedef enum aclEngineType { ACL_ENGINE_SYS, //不关心具体执行引擎时填写 ACL_ENGINE_AICORE, //将算子编译成AI Core算子 ACL_ENGINE_VECTOR //将算子编译成Vector Core算子 } aclopEngineType;
compileFlag	输入	算子编译标识。 typedef enum aclCompileType { ACL_COMPILE_SYS, //向GE、FE注册过的算子 ACL_COMPILE_UNREGISTERED //未向GE、FE注册的算子（需要使用py源文件编译算子，当前不支持） } aclopCompileType;
opPath	输入	算子路径。

返回值说明

返回0表示成功，返回其它值表示失败。

8.10 算子加载与执行

8.10.1 aclopSetModelDir

函数功能

设置加载模型文件的目录，该模型文件是由单算子编译成的*.om文件，同步接口。

约束说明

- 一个进程内只能调用一次aclopSetModelDir接口。
- 一个进程内正在执行的算子的最大个数上限是60000。

函数原型

aclError aclopSetModelDir(const char *modelDir)

参数说明

参数名	输入/输出	说明
modelDir	输入	指定模型文件所在的目录。 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

返回0表示成功，返回其它值表示失败。

8.10.2 aclopLoad

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。

单算子模型数据是指“单算子编译成的*.om文件后，再将om文件读取到内存中”的数据。

约束说明

一个进程内正在执行的算子的最大算子个数上限是60000。

函数原型

aclError aclopLoad(const void *model, size_t modelSize)

参数说明

参数名	输入/输出	说明
model	输入	单算子模型数据的内存地址。
modelSize	输入	内存中的模型数据长度，单位Byte。

返回值说明

返回0表示成功，返回其它值表示失败。

8.10.3 aclopExecute

函数功能

异步执行指定的算子。

约束说明

每个算子的输入、输出组织不同，需要应用在调用时严格按照算子输入、输出参数来组织算子。

用户在调用**aclopExecute**时，ACL根据optype、输入tensor的描述、输出tensor的描述、attr等信息查找对应的任务，并下发执行。

函数原型

aclError aclopExecute(const char *opType,

int numInputs,

const aclTensorDesc *const inputDesc[],

const aclDataBuffer *const inputs[],

int numOutputs,

const aclTensorDesc *const outputDesc[],

aclDataBuffer *const outputs[],

const aclopAttr *attr,

aclrtStream stream)

参数说明

参数名	输入/输出	说明
opType	输入	指定算子类型名称。
numInputs	输入	算子输入tensor的数量。
inputDesc	输入	算子输入tensor的描述。
inputs	输入	算子输入tensor。
numOutputs	输入	算子输出tensor的数量。
outputDesc	输入	算子输出tensor的描述。
outputs	输出	算子输出tensor。
attr	输入	算子属性。
stream	输入	该算子需要加载的stream。

返回值说明

返回0表示成功，返回其它值表示失败。

8.10.4 aclopCompileAndExecute

须知

该接口在“FwkACLlib组件的安装目录/fwkacllib/include/acl/acl_op_compiler.h”头文件中定义。后续特性的预留接口，当前不推荐使用。

函数功能

异步编译并执行指定的算子。

约束说明

每个算子的输入、输出组织不同，需要应用在调用时严格按照算子输入、输出参数来组织算子。

用户在调用aclopCompileAndExecute时，ACL根据optype、输入tensor的描述、输出tensor的描述、attr等信息查找对应的任务，再编译、执行算子。

函数原型

```
aclError aclopCompileAndExecute(const char *opType,
int numInputs,
const aclTensorDesc *const inputDesc[],
const aclDataBuffer *const inputs[],
int numOutputs,
const aclTensorDesc *const outputDesc[],
aclDataBuffer *const outputs[],
const aclopAttr *attr,
aclopEngineType engineType,
aclopCompileType compileFlag,
const char *opPath,
aclrtStream stream);
```

参数说明

参数名	输入/输出	说明
opType	输入	指定算子类型名称。

参数名	输入/输出	说明
numInputs	输入	算子输入tensor的数量。
inputDesc	输入	算子输入tensor的描述。
inputs	输入	算子输入tensor。
numOutputs	输入	算子输出tensor的数量。
outputDesc	输入	算子输出tensor的描述。
outputs	输出	算子输出tensor。
attr	输入	算子属性。
engineType	输入	算子执行引擎。 typedef enum aclEngineType { ACL_ENGINE_SYS, //不关心具体执行引擎时填写 ACL_ENGINE_AICORE, //将算子编译成AI Core算子 ACL_ENGINE_VECTOR //将算子编译成Vector Core算子 } aclopEngineType;
compileFlag	输入	算子编译标识。 typedef enum aclCompileType { ACL_COMPILE_SYS, //向GE、FE注册过的算子 ACL_COMPILE_UNREGISTERED //未向GE、FE注册的算子（需要使用py源文件编译算子，当前不支持） } aclopCompileType;
opPath	输入	算子路径。
stream	输入	该算子需要加载的stream。

返回值说明

返回0表示成功，返回其它值表示失败。

8.10.5 aclopExecWithHandle

函数功能

以Handle方式调用一个算子，性能更优，异步接口。

函数原型

aclError aclopExecWithHandle(aclopHandle *handle,

int numInputs,

const aclDataBuffer *const inputs[],

int numOutputs,

aclDataBuffer *const outputs[],

aclrtStream stream);

参数说明

参数名	输入/输出	说明
handle	输入	指定执行算子的handle。 已提前调用**aclopCreateHandle**接口创建aclopHandle类型的数据。
numInputs	输入	算子输入tensor的数量。
inputs	输入	算子输入tensor。 已提前调用**aclCreateDataBuffer**接口创建aclDataBuffer类型的数据。
numOutputs	输入	算子输出tensor的数量。
outputs	输出	算子输出tensor。
stream	输入	该算子需要加载的stream。

返回值说明

返回0表示成功，返回其它值表示失败。

8.11 CBLAS 接口

8.11.1 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 * sizeof(dataTypeA)
  - B矩阵：shape = (k, n)；内存大小 = k * n * sizeof(dataTypeB)
  - C矩阵：shape = (m, n)；内存大小 = m * n * sizeof(dataTypeC)
- A、B、C矩阵是否转置的标记如果设置为ACL_TRANS_NZ，表示采用内部数据格式，矩阵Shape为4维，Shape及内存大小的计算公式如下（假设m,k,n分别为原始轴）：
  - 当矩阵A和矩阵B中数据的类型为**aclFloat16**，在计算实际内存大小时，m、k、n均按16对齐向上取整计算：
    - A矩阵：shape = (⌈k/16⌉, ⌈m/16⌉, 16, 16)；内存大小 = ⌈m/16⌉*16 * ⌈k/16⌉*16 * sizeof(dataTypeA)
    - B矩阵：shape = (⌈n/16⌉, ⌈k/16⌉, 16, 16)；内存大小 = ⌈k/16⌉*16 * ⌈n/16⌉*16 * sizeof(dataTypeB)

- C矩阵：shape = (⌈n/16⌉, ⌈m/16⌉, 16, 16)；内存大小 = ⌈m/16⌉*16 * ⌈n/16⌉*16 * sizeof(dataTypeC)

– 当矩阵A和矩阵B中数据的类型为int8_t，在计算实际内存大小时，reduce轴按32对齐向上取整计算，非reduce轴按16对齐向上取整计算：

- A矩阵：shape = (⌈k/32⌉, ⌈m/16⌉, 16, 32)；内存大小 = ⌈m/16⌉*16 * ⌈k/32⌉*32 * sizeof(dataTypeA)
- B矩阵：shape = (⌈k/32⌉, ⌈n/16⌉, 32, 16)；内存大小 = ⌈k/32⌉*32 * ⌈n/16⌉*16 * sizeof(dataTypeB)
- C矩阵：shape = (⌈n/16⌉, ⌈m/16⌉, 16, 16)；内存大小 = ⌈m/16⌉*16 * ⌈n/16⌉*16 * sizeof(dataTypeC)

说明

⌈ ⌉表示向上对齐。

8.11.2 aclblasGemvEx

函数功能

执行矩阵-向量的乘法，输入数据、输出数据的数据类型通过入参设置，异步接口：

y = αAx + βy

约束说明

A、x、y的数据类型支持仅支持以下组合， α和β的数据类型与y一致。

A的数据类型	x的数据类型	y的数据类型
aclFloat16	aclFloat16	aclFloat16
aclFloat16	aclFloat16	float(float32)
int8_t	int8_t	float(float32)
int8_t	int8_t	int32_t

函数原型

```
aclError aclblasGemvEx(aclTransType transA,
int m,
int n,
const void *alpha,
const void *a,
int lda,
aclDataType dataTypeA,
```

```
const void *x,
int incx,
aclDataType dataTypeX,
const void *beta,
void *y,
int incy,
aclDataType dataTypeY,
aclComputeType type,
aclrtStream stream);
```

参数说明

参数名	输入/输出	说明
transA	输入	A矩阵是否转置的标记。
m	输入	矩阵A的行数，存储矩阵乘数据时，行优先。
n	输入	矩阵A的列数。
alpha	输入	用于执行乘操作的标量α的数据指针。
a	输入	矩阵A的数据指针。
lda	输入	A矩阵的主维，此时选择转置，按行优先，则lda为A的列数。预留参数，当前只能设置为-1。
dataTypeA	输入	矩阵A的数据类型。
x	输入	向量x的数据指针。
incx	输入	x连续元素之间的步长。 预留参数，当前只能设置为-1。
dataTypeX	输入	向量x的数据类型。
beta	输入	用于执行乘操作的标量β的数据指针。
y	输入&输出	向量y的数据指针。
incy	输入	y连续元素之间的步长。 预留参数，当前只能设置为-1。
dataTypeY	输入	向量y的数据类型。
type	输入	计算精度，默认高精度。
stream	输入	执行算子所在的Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

8.11.3 aclblasCreateHandleForGemvEx

函数功能

创建矩阵-向量乘的handle，输入数据、输出数据的数据类型通过入参设置，同步接口。

约束说明

A、x、y的数据类型支持仅支持以下组合：

A的数据类型	x的数据类型	y的数据类型
aclFloat16	aclFloat16	aclFloat16
aclFloat16	aclFloat16	float(float32)
int8_t	int8_t	float(float32)
int8_t	int8_t	int32_t

函数原型

aclError aclblasCreateHandleForGemvEx(aclTransType transA,

int m,

int n,

aclDataType dataTypeA,

aclDataType dataTypeX,

aclDataType dataTypeY,

aclComputeType type,

aclopHandle **handle)

参数说明

参数名	输入/输出	说明
transA	输入	A矩阵是否转置的标记。
m	输入	矩阵A的行数，存储矩阵乘数据时，行优先。

参数名	输入/输出	说明
n	输入	矩阵A的列数。
dataTypeA	输入	矩阵A的数据类型。
dataTypeX	输入	向量x的数据类型。
dataTypeY	输入	向量y的数据类型。
type	输入	计算精度，默认高精度。
handle	输出	“执行算子的handle数据的指针”的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

8.11.4 aclblasHgemv

函数功能

执行矩阵-向量的乘法，输入数据和输出数据的数据类型为**aclFloat16**，异步接口：

$y = \alpha Ax + \beta y$

函数原型

```
aclError aclblasHgemv(aclTransType transA,
int m,
int n,
const aclFloat16 *alpha,
const aclFloat16 *a,
int lda,
const aclFloat16 *x,
int incx,
const aclFloat16 *beta,
aclFloat16 *y,
int incy,
aclComputeType type,
aclrtStream stream)
```

参数说明

参数名	输入/输出	说明
transA	输入	A矩阵是否转置的标记。
m	输入	矩阵A的行数，存储矩阵乘数据时，行优先。
n	输入	矩阵A的列数。
alpha	输入	用于执行乘操作的标量α。
a	输入	矩阵A的数据指针。
lda	输入	A矩阵的主维，此时选择转置，按行优先，则lda为A的列数。预留参数，当前只能设置为-1。
x	输入	向量x的数据指针。
incx	输入	x连续元素之间的步长。 预留参数，当前只能设置为-1。
beta	输入	用于执行乘操作的标量β。
y	输入&输出	向量y的数据指针。
incy	输入	y连续元素之间的步长。 预留参数，当前只能设置为-1。
type	输入	计算精度，默认高精度。
stream	输入	执行算子所在的Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

8.11.5 aclblasCreateHandleForHgemv

函数功能

创建矩阵-向量乘的handle，输入数据和输出数据的数据类型为**aclFloat16**，同步接口。

函数原型

aclError aclblasCreateHandleForHgemv(aclTransType transA,

int m,

int n,

aclComputeType type,

aclopHandle **handle)

参数说明

参数名	输入/输出	说明
transA	输入	A矩阵是否转置的标记。
m	输入	矩阵A的行数，存储矩阵乘数据时，行优先。
n	输入	矩阵A的列数。
type	输入	计算精度，默认高精度。
handle	输出	“执行算子的handle数据的指针”的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

8.11.6 aclblasS8gemv

函数功能

执行矩阵-向量的乘法，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：

y = αAx + βy

函数原型

aclError aclblasS8gemv(aclTransType transA,

int m,

int n,

const int32_t *alpha,

const int8_t *a,

int lda,

const int8_t *x,

int incx,

const int32_t *beta,

int32_t *y,

int incy,

aclComputeType type,

aclrtStream stream)

参数说明

参数名	输入/输出	说明
transA	输入	A矩阵是否转置的标记。
m	输入	矩阵A的行数，存储矩阵乘数据时，行优先。
n	输入	矩阵A的列数。
alpha	输入	用于执行乘操作的标量α。
a	输入	矩阵A的数据指针。
lda	输入	A矩阵的主维，此时选择转置，按行优先，则lda为A的列数。预留参数，当前只能设置为-1。
x	输入	向量x的数据指针。
incx	输入	x连续元素之间的步长。 预留参数，当前只能设置为-1。
beta	输入	用于执行乘操作的标量β。
y	输入&输出	向量y的数据指针。
incy	输入	y连续元素之间的步长。 预留参数，当前只能设置为-1。
type	输入	计算精度，默认高精度。
stream	输入	执行算子所在的Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

8.11.7 aclblasCreateHandleForS8gemv

函数功能

创建矩阵-向量乘的handle，输入数据的数据类型为int8_t，输出数据的数据类型为int32_t，同步接口。

函数原型

aclError aclblasCreateHandleForS8gemv(aclTransType transA,

int m,

int n,

aclComputeType type,

aclopHandle **handle)

参数说明

参数名	输入/输出	说明
transA	输入	A矩阵是否转置的标记。
m	输入	矩阵A的行数，存储矩阵乘数据时，行优先。
n	输入	矩阵A的列数。
type	输入	计算精度，默认高精度。
handle	输出	“执行算子的handle数据的指针”的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

8.11.8 aclblasGemmEx

函数功能

执行矩阵-矩阵的乘法，输入数据、输出数据的数据类型通过入参设置，异步接口：

C = αAB + βC

约束说明

A、B、C的数据类型支持仅支持以下组合， α和β的数据类型与C一致。

A的数据类型	B的数据类型	C的数据类型
aclFloat16	aclFloat16	aclFloat16
aclFloat16	aclFloat16	float(float32)
int8_t	int8_t	float(float32)
int8_t	int8_t	int32_t

函数原型

aclError aclblasGemmEx(aclTransType transA,

aclTransType transB,

aclTransType transC,

int m,

int n,

int k,

const void *alpha,

const void *matrixA,

int lda,

aclDataType dataTypeA,

const void *matrixB,

int ldb,

aclDataType dataTypeB,

const void *beta,

void *matrixC,

int ldc,

aclDataType dataTypeC,

aclComputeType type,

aclrtStream stream)

参数说明

参数名	输入/输出	说明
transA	输入	A矩阵是否转置的标记。
transB	输入	B矩阵是否转置的标记。
transC	输入	C矩阵的标记，当前仅支持ACL_TRANS_N。
m	输入	矩阵A的行数与矩阵C的行数。
n	输入	矩阵B的列数与矩阵C的列数。
k	输入	矩阵A的列数与矩阵B的行数。
alpha	输入	用于执行乘操作的标量α的数据指针。
matrixA	输入	矩阵A的数据指针。
lda	输入	A矩阵的主维，此时选择转置，按行优先，则lda为A的列数。预留参数，当前只能设置为-1。
dataTypeA	输入	矩阵A的数据类型。
matrixB	输入	矩阵B的数据指针。

参数名	输入/输出	说明
ldb	输入	B矩阵的主维，此时选择转置，按行优先，则ldb为B的列数。预留参数，当前只能设置为-1。
dataTypeB	输入	矩阵B的数据类型。
beta	输入	用于执行乘操作的标量β的数据指针。
matrixC	输入&输出	矩阵C的数据指针。
ldc	输入	C矩阵的主维，预留参数，当前只能设置为-1。
dataTypeC	输入	矩阵C的数据类型。
type	输入	计算精度，默认高精度。
stream	输入	执行算子所在的Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

8.11.9 aclblasCreateHandleForGemmEx

函数功能

创建矩阵-矩阵乘的handle，输入数据、输出数据的数据类型通过入参设置，同步接口。

约束说明

A、B、C的数据类型支持仅支持以下组合：

A的数据类型	B的数据类型	C的数据类型
aclFloat16	aclFloat16	aclFloat16
aclFloat16	aclFloat16	float(float32)
int8_t	int8_t	float(float32)
int8_t	int8_t	int32_t

函数原型

aclError aclblasCreateHandleForGemmEx(aclTransType transA,

aclTransType transB,

aclTransType transC,

int m,

int n,

int k,

aclDataType dataTypeA,

aclDataType dataTypeB,

aclDataType dataTypeC,

aclComputeType type,

aclopHandle **handle)

参数说明

参数名	输入/输出	说明
transA	输入	A矩阵是否转置的标记。
transB	输入	B矩阵是否转置的标记。
transC	输入	C矩阵的标记，当前仅支持ACL_TRANS_N。
m	输入	矩阵A的行数与矩阵C的行数。
n	输入	矩阵B的列数与矩阵C的列数。
k	输入	矩阵A的列数与矩阵B的行数。
dataTypeA	输入	矩阵A的数据类型。
dataTypeB	输入	矩阵B的数据类型。
dataTypeC	输入	矩阵C的数据类型。
type	输入	计算精度，默认高精度。
handle	输出	“执行算子的handle数据的指针”的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

8.11.10 aclblasHgemm

函数功能

执行矩阵-矩阵的乘法，输入数据和输出数据的数据类型为**aclFloat16**，异步接口：

C = αAB + βC

函数原型

```
aclError aclblasHgemm(aclTransType transA,
aclTransType transB,
aclTransType transC,
int m,
int n,
int k,
const aclFloat16 *alpha,
const aclFloat16 *matrixA,
int lda,
const aclFloat16 *matrixB,
int ldb,
const aclFloat16 *beta,
aclFloat16 *matrixC,
int ldc,
aclComputeType type,
aclrtStream stream)
```

参数说明

参数名	输入/输出	说明
transA	输入	A矩阵是否转置的标记。
transB	输入	B矩阵是否转置的标记。
transC	输入	C矩阵的标记，当前仅支持ACL_TRANS_N。
m	输入	矩阵A的行数与矩阵C的行数。
n	输入	矩阵B的列数与矩阵C的列数。
k	输入	矩阵A的列数与矩阵B的行数。
alpha	输入	用于执行乘操作的标量α。
matrixA	输入	矩阵A的数据指针。
lda	输入	A矩阵的主维，此时选择转置，按行优先，则lda为A的列数。预留参数，当前只能设置为-1。
matrixB	输入	矩阵B的数据指针。

参数名	输入/输出	说明
ldb	输入	B矩阵的主维，此时选择转置，按行优先，则ldb为B的列数。预留参数，当前只能设置为-1。
beta	输入	用于执行乘操作的标量β。
matrixC	输入&输出	矩阵C的数据指针。
ldc	输入	C矩阵的主维，预留参数，当前只能设置为-1。
type	输入	计算精度，默认高精度。
stream	输入	执行算子所在的Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

8.11.11 aclblasCreateHandleForHgemm

函数功能

创建矩阵-矩阵乘的handle，输入数据和输出数据的数据类型为**aclFloat16**，同步接口。

函数原型

aclError aclblasCreateHandleForHgemm(aclTransType transA,

aclTransType transB,

aclTransType transC,

int m,

int n,

int k,

aclComputeType type,

aclopHandle **handle)

参数说明

参数名	输入/输出	说明
transA	输入	A矩阵是否转置的标记。
transB	输入	B矩阵是否转置的标记。

参数名	输入/输出	说明
transC	输入	C矩阵的标记，当前仅支持ACL_TRANS_N。
m	输入	矩阵A的行数与矩阵C的行数。
n	输入	矩阵B的列数与矩阵C的列数。
k	输入	矩阵A的列数与矩阵B的行数。
type	输入	计算精度，默认高精度。
handle	输出	“执行算子的handle数据的指针”的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

8.11.12 aclblasS8gemm

函数功能

执行矩阵-矩阵的乘法，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：

C = αAB + βC

函数原型

```
aclError aclblasS8gemm(aclTransType transA,
aclTransType transB,
aclTransType transC,
int m,
int n,
int k,
const int32_t *alpha,
const int8_t *matrixA,
int lda,
const int8_t *matrixB,
int ldb,
const int32_t *beta,
int32_t *matrixC,
int ldc,
```

aclComputeType type,

aclrtStream stream)

参数说明

参数名	输入/输出	说明
transA	输入	A矩阵是否转置的标记。
transB	输入	B矩阵是否转置的标记。
transC	输入	C矩阵的标记，当前仅支持ACL_TRANS_N。
m	输入	矩阵A的行数与矩阵C的行数。
n	输入	矩阵B的列数与矩阵C的列数。
k	输入	矩阵A的列数与矩阵B的行数。
alpha	输入	用于执行乘操作的标量α。
matrixA	输入	矩阵A的数据指针。
lda	输入	A矩阵的主维，此时选择转置，按行优先，则lda为A的列数。预留参数，当前只能设置为-1。
matrixB	输入	矩阵B的数据指针。
ldb	输入	B矩阵的主维，此时选择转置，按行优先，则ldb为B的列数。预留参数，当前只能设置为-1。
beta	输入	用于执行乘操作的标量β。
matrixC	输入&输出	矩阵C的数据指针。
ldc	输入	C矩阵的主维，预留参数，当前只能设置为-1。
type	输入	计算精度，默认高精度。
stream	输入	执行算子所在的Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

8.11.13 aclblasCreateHandleForS8gemm

函数功能

创建矩阵-矩阵乘的handle，输入数据的数据类型为int8_t，输出数据的数据类型为int32_t，同步接口。

函数原型

aclError aclblasCreateHandleForS8gemm(aclTransType transA,

aclTransType transB,

aclTransType transC,

int m,

int n,

int k,

aclComputeType type,

aclopHandle **handle)

参数说明

参数名	输入/输出	说明
transA	输入	A矩阵是否转置的标记。
transB	输入	B矩阵是否转置的标记。
transC	输入	C矩阵的标记，当前仅支持ACL_TRANS_N。
m	输入	矩阵A的行数与矩阵C的行数。
n	输入	矩阵B的列数与矩阵C的列数。
k	输入	矩阵A的列数与矩阵B的行数。
type	输入	计算精度，默认高精度。
handle	输出	“执行算子的handle数据的指针”的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

8.11.14 aclopCast

函数功能

转换输入数据的数据类型，异步接口。

函数原型

aclError aclopCast(const aclTensorDesc *srcDesc,

const aclDataBuffer *srcBuffer,

const aclTensorDesc *dstDesc,

aclDataBuffer *dstBuffer,

uint8_t truncate,

aclrtStream stream)

参数说明

参数名	输入/输出	说明
srcDesc	输入	输入tensor的描述。
srcBuffer	输入	输入tensor。
dstDesc	输入	输出tensor的描述。
dstBuffer	输出	输出tensor。
truncate	输入	预留。
stream	输入	执行算子所在的Stream。

返回值说明

返回0表示成功，返回其它值表示失败。

8.11.15 aclopCreateHandleForCast

函数功能

创建数据类型转换的handle，同步接口。

函数原型

aclError aclopCreateHandleForCast(aclTensorDesc *srcDesc,

aclTensorDesc *dstDesc,

uint8_t truncate,

aclopHandle **handle)

参数说明

参数名	输入/输出	说明
srcDesc	输入	输入tensor的描述。
dstDesc	输入	输出tensor的描述。
truncate	输入	预留。
handle	输入	“执行算子的handle数据的指针”的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

8.12 媒体数据处理

媒体数据处理包括图片解码/抠图/缩放、视频解码等。

8.12.1 总体说明

- 关于异步接口

  对于本章介绍的异步接口，调用接口成功仅表示任务下发成功，不表示任务执行成功，对于有依赖的接口，为确保能按序执行任务，建议用户在多个接口中指定同一个stream，因为同一个stream中的任务按接口调用顺序执行。

  在调用异步口对图片进行解码、抠图、缩放等操作时，如果任务之间有依赖，一定要调用**aclrtSynchronizeStream**接口确保在同一个Stream中的任务按序执行。

  调用异步接口后，不能马上释放资源，需调用同步等待接口（例如，**aclrtSynchronizeStream**）确保Device侧任务执行完成后才能释放。

- 关于内存申请/释放

  实现媒体数据处理的VPC功能、JPEGD功能、JEPGE等功能前，若需要申请Device上的内存存放输入或输出数据，需调用**acldvppMalloc**申请内存、调用**acldvppFree**接口释放内存。

- 支持的媒体数据处理功能如下：
  - VPC（vision preprocessing core）功能：支持对图片做抠图、缩放、叠加、拼接、格式转换等操作，详细描述请参见**8.12.4.1 功能及约束说明**。
  - JPEGD（JPEG decoder）功能：将.jpg、.jpeg、.JPG、.JPEG图片解码成YUV格式图片，详细描述请参见**8.12.5.1 功能及约束说明**。
  - JPEGE（JPEG encoder）功能：将YUV格式图片编码成.jpg图片，详细描述请参见**8.12.6.1 功能及约束说明**。
  - VDEC（video decoder）功能：实现视频的解码，详细描述请参见**8.12.7.1 功能及约束说明**。
  - VENC（video encoder）功能：实现视频的编码，详细描述请参见**8.12.8.1 功能及约束说明**。
- 除**acldvppMalloc**接口、**acldvppFree**接口外，**媒体数据处理**章节中的其它接口只能在Host上调用，不能在Device上调用。

8.12.2 内存申请与释放

8.12.2.1 acldvppMalloc

函数功能

该接口主要用于分配内存给Device侧媒体数据处理时使用，申请的大页内存满足数据处理的要求（例如，内存首地址128对齐），同步接口。调用该接口申请内存后，必须使用**acldvppFree**接口释放内存。

- 调用acldvppMalloc接口申请内存时，会对用户输入的size按向上对齐成32整数倍后，再多加32字节申请。

说明

若用户使用acldvppMalloc接口申请大块内存并自行划分、管理内存时，用户在管理内存时，需按每张图片的实际数据大小向上对齐成32整数倍+32字节（ALIGN_UP[len]+32字节）来管理内存。

例如，用户已使用acldvppMalloc接口申请大块内存并自行管理内存，用户需管理n张图片的内存，每张图片大小为len字节，实际应按照n*(ALIGN_UP[len]+32字节)的大小管理内存，每张图片的内存地址按(ALIGN_UP[len]+32字节)为单位偏移。

ALIGN_UP表示向上按32字节对齐：((len-1)/32+1)*32。

- 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。

约束说明

频繁调用acldvppMalloc接口申请内存、调用**acldvppFree**接口释放内存，会损耗性能，建议用户提前做内存预先分配或二次管理，避免频繁申请/释放内存。

函数原型

aclError acldvppMalloc(void **devPtr, size_t size)

参数说明

参数名	输入/输出	说明
devPtr	输出	“Device上已分配内存的指针”的指针。
size	输入	申请内存的大小，单位Byte。

返回值说明

返回0表示成功，返回其它值表示失败。

8.12.2.2 acldvppFree

函数功能

释放通过**acldvppMalloc**接口申请的内存，同步接口。

函数原型

aclError acldvppFree(void *devPtr)

参数说明

参数名	输入/输出	说明
devPtr	输入	待释放内存的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

8.12.3 通道创建与释放

8.12.3.1 VPC/JPEGD/JPEGE 图片处理通道

8.12.3.1.1 acldvppCreateChannel

函数功能

创建图片数据处理的通道，同一个通道可以重复使用，销毁后不再可用，同步接口。

约束说明

通道为非线程安全，即不同线程要求创建不同的通道。

函数原型

aclError acldvppCreateChannel(acldvppChannelDesc *channelDesc)

参数说明

参数名	输入/输出	说明
channelDesc	输入&输出	指定通道描述信息。 调用**acldvppCreateChannelDesc**接口创建acldvppChannelDesc类型的数据。

返回值说明

返回0表示成功，返回其它值表示失败。

8.12.3.1.2 acldvppDestroyChannel

函数功能

销毁图片数据处理的通道，只能销毁通过**acldvppCreateChannel**接口创建的通道，同步接口。

函数原型

aclError acldvppDestroyChannel(acldvppChannelDesc *channelDesc)

参数说明

参数名	输入/输出	说明
channelDesc	输入	指定通道描述信息，与调用**acldvppCreateChannel**接口创建通道时指定的channelDesc保持一致。

返回值说明

返回0表示成功，返回其它值表示失败。

8.12.3.2 VDEC 视频解码通道

8.12.3.2.1 aclvdecCreateChannel

函数功能

创建视频解码处理的通道，同一个通道可以重复使用，销毁后不再可用，同步接口。

aclvdecCreateChannel接口内部封装了**aclrtCreateStream**接口显式创建Stream、**aclrtSubscribeReport**接口指定处理Stream上回调函数的线程，回调函数和线程是由用户调用**aclvdecSetChannelDesc系列接口**时指定的。用户在实现VDEC功能时，无需再单独调用**aclrtCreateStream**接口、**aclrtSubscribeReport**接口。

函数原型

aclError aclvdecCreateChannel (aclvdecChannelDesc *channelDesc)

参数说明

参数名	输入/输出	说明
channelDesc	输入&输出	指定通道描述信息。 调用**aclvdecCreateChannelDesc**接口创建aclvdecChannelDesc类型的数据，调用**aclvdecSetChannelDesc系列接口**设置通道描述信息的属性。

返回值说明

返回0表示成功，返回其它值表示失败。

8.12.3.2.2 aclvdecDestroyChannel

函数功能

销毁视频解码处理的通道，只能销毁通过**aclvdecCreateChannel**接口创建的通道，同步接口。

销毁通道接口会等待已发送帧解码完成且用户的回调函数处理完成后再销毁通道。

aclvdecDestroyChannel接口内部封装了**aclrtUnSubscribeReport**接口取消线程注册（Stream上的回调函数不再由指定线程处理）、**aclrtDestroyStream**接口销毁Stream。用户在实现VDEC功能时，无需再单独调用**aclrtUnSubscribeReport**接口、**aclrtDestroyStream**接口。

函数原型

aclError aclvdecDestroyChannel (aclvdecChannelDesc *channelDesc)

参数说明

参数名	输入/输出	说明
channelDesc	输入	指定通道描述信息，与调用**aclvdecCreateChannel**接口创建通道时指定的channelDesc保持一致。

返回值说明

返回0表示成功，返回其它值表示失败。

8.12.3.3 VENC 视频编码通道

8.12.3.3.1 aclvencCreateChannel

函数功能

创建视频编码处理的通道，同一个通道可以重复使用，销毁后不再可用，同步接口。

函数原型

aclError aclvencCreateChannel(aclvencChannelDesc *channelDesc)

参数说明

参数名	输入/输出	说明
channelDesc	输入&输出	指定通道描述信息。 调用**aclvencCreateChannelDesc**接口创建aclvencChannelDesc类型的数据，调用**aclvencSetChannelDesc系列接口**设置通道描述信息的属性。

返回值说明

返回0表示成功，返回其它值表示失败。

8.12.3.3.2 aclvencDestroyChannel

函数功能

销毁视频编码处理的通道，只能销毁通过**aclvencCreateChannel**接口创建的通道，同步接口。

销毁通道接口会等待用户的回调函数处理完成后再销毁。

函数原型

aclError aclvencDestroyChannel(aclvencChannelDesc *channelDesc)

参数说明

参数名	输入/输出	说明
channelDesc	输入	指定通道描述信息，与调用**aclvencCreateChannel**接口创建通道时指定的channelDesc保持一致。

返回值说明

返回0表示成功，返回其它值表示失败。

8.12.4 VPC 功能

8.12.4.1 功能及约束说明

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，VPC的处理方式可分为：
    - 非8K缩放，用于处理“**widthStride**在32~4096（包括4096）范围内，**heightStride**在6~4096”的输入图片，不同格式的输入图片，widthStride的取值范围不同，详细描述参见**表8-2**。
    - 8K缩放，用于处理“widthStride在4096~8192范围内或heightStride在4096~8192范围内（不包括4096）”的输入图片。
  - 从是否抠多张图的维度，可分为单图裁剪缩放（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。
- **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。
- **格式转换**
  - 支持RGB格式、YUV格式之间的格式转换，目前的输入图片格式、输出图片格式，请参见**表8-2**。
  - 图像灰度化，对输出图像数据只取Y分量的数据。

约束说明

- 关于VPC输入/输出的约束

表 8-2 关于 VPC 输入/输出的约束

VPC输入/输出	图片分辨率	图片格式	内存要求	宽stride、高stride对齐要求
VPC输入	• 非8K缩放： – YUV400SP/YUV420SP/YUV422SP/yuv444sp：widthStride在32~4096（包括4096）范围内，heightStride在6~4096。 – YUV422packed：widthStride/2在32~4096（包括4096）范围内，heightStride在6~4096。 – YUV444packed/RGB888：	• 非8K缩放：支持acldvppPixelFormat枚举值中的如下枚举项： PIXEL_FORMAT_YUV_400 = 0, // 0, YUV400 8bit PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // 1, YUV420SP NV12 8bit PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // 2, YUV420SP NV21 8bit PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3, // 3, YUV422SP NV12 8bit PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4, // 4, YUV422SP NV21 8bit PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5, // 5, YUV444SP NV12 8bit PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6, // 6, YUV444SP NV21 8bit PIXEL_FORMAT_YUYV_PACKED_422 = 7, // 7, YUV422P YUYV 8bit PIXEL_FORMAT_UYVY_PACKED_422 = 8, // 8, YUV422P UYVY 8bit PIXEL_FORMAT_YVYU_PACKED_422 = 9, // 9, YUV422P YVYU 8bit PIXEL_FORMAT_VYUY_PACKED_422 = 10, // 10, YUV422P VYUY 8bit PIXEL_FORMAT_YUV_PACKED_444 = 11, // 11, YUV444P 8bit PIXEL_FORMAT_RGB_888 = 12, // 12, RGB888 PIXEL_FORMAT_BGR_888 = 13, // 13, BGR888 PIXEL_FORMAT_ARGB_8888 = 14, // 14, ARGB8888 PIXEL_FORMAT_ABGR_8888 = 15, // 15, ABGR8888 PIXEL_FORMAT_RGBA_8888 = 16, // 16, RGBA8888 PIXEL_FORMAT_BGRA_8888 = 17, // 17, BGRA8888 PIXEL_FORMAT_YUV_SEMI_PLANNER_420_10BIT = 18, // 18, YUV420SP 10bit PIXEL_FORMAT_YVU_SEM	• 内存地址起始要求16对齐。 • 输入内存大小与图片数据的格式相关，计算公式如下： – YUV400SP、YUV420SP：widthStride*heightStride*3/2 – YUV422SP：widthStride*heightStride*2 – YUV444SP：widthStride*heightStride*3 – YUV422packed：widthStride*heightStride – YUV444packed、RGB888	• 对于不同图像格式，宽stride的计算公式不同： – YUV400SP、YUV420SP、YUV422SP、YUV444SP：输入图片的宽（width）对齐到16。 – YUV422packed：输入图片的宽（width）对齐到16后，再乘以2（宽16对齐，每个像素占2个字节）。 – YUV444packed、RGB888：输入图片的宽

VPC输入/输出	图片分辨率	图片格式	内存要求	宽stride、高stride对齐要求
	widthStride/3在32~4096（包括4096）范围内，heightStride在6~4096。 – XRGB8888：widthStride/4在32~4096（包括4096）范围内，heightStride在6~4096。 ● 8K缩放：widthStride在4096~8192范围内或heightStride在4096~8192范围内（不包括4096）	I_PLANNER_420_10BIT = 19, // 19, YVU420sp 10bit ● 8K缩放：YUV420SP（NV12、NV21）。	8：widthStride*heightStride – XRGB8888：widthStride*heightStride Device的内存，调用**acldvppMalloc**接口/**acldvppFree**接口申请或释放内存。	（width）对齐到16，再乘以3（宽16对齐，每个像素占3个字节）。 – XRGB8888：输入图片的宽（width）对齐到16后，再乘以4（宽16对齐，每个像素占4个字节）。 ● 高stride：2对齐。

VPC输入/输出	图片分辨率	图片格式	内存要求	宽stride、高stride对齐要求
VPC输出	32*6~4096*4096	支持**acldvppPixelFormat**枚举值中的如下枚举项： PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // 1, YUV420SP NV12 8bit PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // 2, YUV420SP NV21 8bit	• 内存地址起始要求16对齐。 • 输出内存大小与图片数据的格式相关，计算公式如下： – YUV400SP、YUV420SP：widthStride*heightStride*3/2 – YUV422SP：widthStride*heightStride*2 – YUV444SP：widthStride*heightStride*3 – YUV422packed：widthStride*heightStride – YUV444packed、RGB888：widthS	• 宽stride：16对齐 • 高stride：2对齐。

VPC输入/输出	图片分辨率	图片格式	内存要求	宽stride、高stride对齐要求
			tride*heightStride – XRGB8888：widthStride*heightStride Device的内存，调用**acldvppMalloc**接口/**acldvppFree**接口申请或释放内存。	

- 针对缩放功能，宽高缩放比例范围：[1/32, 16]。
- 贴图场景贴图左偏移需要16对齐。

功能示意图及关键概念

图 8-2 VPC 功能示意图（抠图+缩放+叠加）

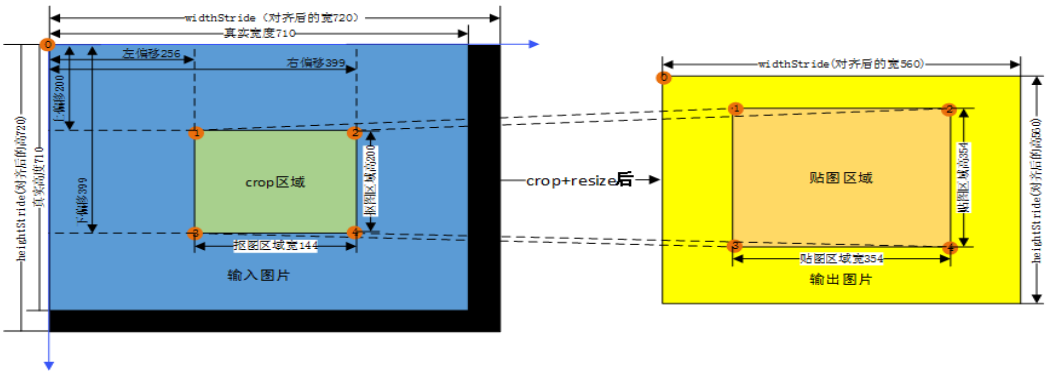


图 8-3 VPC 功能示意图（拼接）

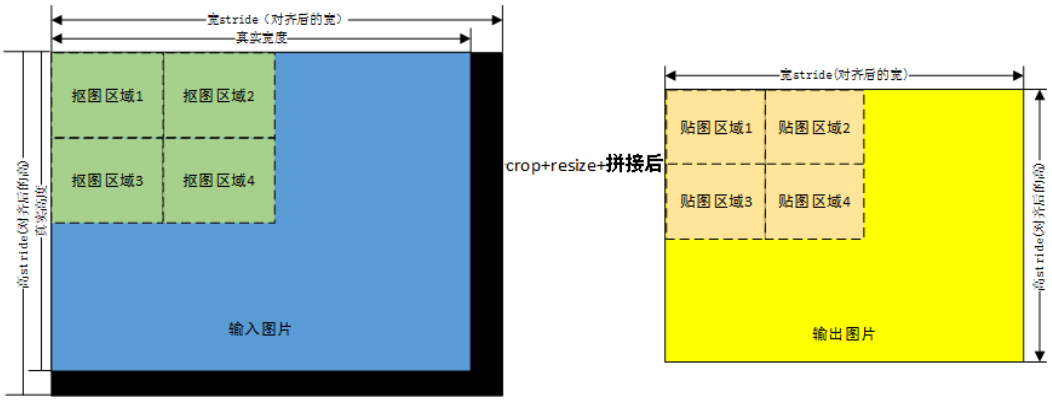


表 8-3 概念解释

概念	描述
宽stride（widthStride）	指一行图像步长，表示输入图片对齐后的宽，RGB格式或YUV格式的widthStride计算方式不一样。 widthStride的对齐要求，请参见表8-2。
高stride（heightStride）	指图像在内存中的行数，表示输入图片对齐后的高。 heightStride的对齐要求，请参见表8-2。
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。 • 左偏移：输入/输出图片中，抠图/贴图区域1、3两个点相对于0点水平向左偏移的值。 • 右偏移：输入/输出图片中，抠图/贴图区域2、4两个点相对于0点水平向左偏移的值。 • 上偏移：输入/输出图片中，抠图/贴图区域1、2两个点相对于0点垂直向上偏移的值。 • 下偏移：输入/输出图片中，抠图/贴图区域3、4两个点相对于0点垂直向上偏移的值。
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。

概念	描述
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。 约束如下： ● 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。 ● 抠图区域不超出输入图片，贴图区域不超出输出图片。 ● 贴图时可直接放置在输出图片的最左侧，即相对输出图片的左偏移为0。 ● 最大贴图个数为256个。 ● 贴图区域相对输出图片的左偏移16对齐。 ● 输出图片的贴图宽度建议16对齐，如果不是16对齐，会多写一段无效数据使其16对齐。

性能指标说明

- 对于**非8K缩放**，基本场景性能指标参考如下：

  对于1080p的图像，若存在Host->Device的图片数据拷贝，由于拷贝带宽限制，最大总帧率约为1000fps。

  对于4K的图像，若存在Host->Device的图片数据拷贝，由于拷贝带宽限制，最大总帧率约为250fps。

场景举例	总帧率
● 输入图像分辨率：1080p（1920*1080） ● 输出图像分辨率：1080p（1920*1080） ● 输入/输出图片格式：YUV420SP ● n路（n<4，1路对应一个线程）	n*360fps
● 输入图像分辨率：1080p（1920*1080） ● 输出图像分辨率：1080p（1920*1080） ● 输入/输出图片格式：YUV420SP ● n路（n≥4，1路对应一个线程）	1440fps
● 输入图像分辨率：4K图像（3840*2160） ● 输出图像分辨率：4K图像（3840*2160） ● 输入/输出图片格式：YUV420SP ● n路（n<4，1路对应一个线程）	n*90fps

场景举例	总帧率
• 输入图像分辨率：4K图像（3840*2160） • 输出图像分辨率：4K图像（3840*2160） • 输入/输出图片格式：YUV420SP • n路（n≥4，1路对应一个线程）	360fps

- 对于**8K缩放**，VPC性能与输出图像分辨率强相关，输出图像分辨率越大，处理耗时越久，性能越低。典型场景性能指标参考如下：

场景举例	总帧率
• 输入图像分辨率：8K图像（7680*4320） • 输出图像分辨率：1080p（1920*1080） • 输入/输出图片格式：YUV420SP • n路（n<4，1路对应一个线程）	n*4fps
• 输入图像分辨率：8K图像（7680*4320） • 输出图像分辨率：1080p（1920*1080） • 输入/输出图片格式：YUV420SP • n路（n≥4，1路对应一个线程）	16fps
• 输入图像分辨率：8K图像（7680*4320） • 输出图像分辨率：4K图像（3840*2160） • 输入/输出图片格式：YUV420SP • n路（n<4，1路对应一个线程）	n*1fps
• 输入图像分辨率：8K图像（7680*4320） • 输出图像分辨率：4K图像（3840*2160） • 输入/输出图片格式：YUV420SP • n路（n≥4，1路对应一个线程）	4fps

8.12.4.2 acldvppVpcResizeAsync

函数功能

将输入图片缩放到输出图片大小，异步接口。

函数原型

aclError acldvppVpcResizeAsync(acldvppChannelDesc *channelDesc,

acldvppPicDesc *inputDesc,

acldvppPicDesc *outputDesc,

acldvppResizeConfig *resizeConfig,

aclrtStream stream)

参数说明

参数名	输入/输出	说明
channelDesc	输入	指定通道描述信息，与调用acldvppCreateChannel接口创建通道时指定的channelDesc保持一致。
inputDesc	输入	输入图片信息。 • 调用acldvppCreatePicDesc接口创建图片描述信息，调用acldvppSetPicDesc系列接口设置图片描述参数（例如，图片格式、宽、高等）。 • 输入图片分辨率、图像格式、widthStride对齐要求、heightStride对齐要求等，请参见约束说明。
outputDesc	输入&输出	输出图片信息。 • 调用acldvppCreatePicDesc接口创建图片描述信息。 • 输出图像分辨率、图像格式、widthStride对齐要求、heightStride对齐要求等，请参见约束说明。
resizeConfig	输入	指定图片缩放配置数据。 调用acldvppCreateResizeConfig接口创建图片缩放配置数据。
stream	输入	指定stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

8.12.4.3 acldvppVpcCropAsync

函数功能

按指定区域从一张输入图片中抠出一张子图，再将抠的图片存放到输出内存中，作为输出图片，异步接口。

输出图片区域与cropArea不一致时会对图片再做一次缩放操作。

函数原型

aclError acldvppVpcCropAsync(acldvppChannelDesc *channelDesc,

acldvppPicDesc *inputDesc,

acldvppPicDesc *outputDesc,

acldvppRoiConfig *cropArea,

aclrtStream stream)

参数说明

参数名	输入/输出	说明
channelDesc	输入	指定通道描述信息，与调用acldvppCreateChannel接口创建通道时指定的channelDesc保持一致。
inputDesc	输入	输入图片信息。 • 调用acldvppCreatePicDesc接口创建图片描述信息，调用acldvppSetPicDesc系列接口设置图片描述参数（例如，图片格式、宽、高等）。 • 输入图片分辨率、图像格式、widthStride对齐要求、heightStride对齐要求等，请参见约束说明。
outputDesc	输入&输出	输出图片信息。 • 调用acldvppCreatePicDesc接口创建图片描述信息。 • 输出图像分辨率、图像格式、widthStride对齐要求、heightStride对齐要求等，请参见约束说明。

参数名	输入/输出	说明
cropArea	输入	抠图区域位置。 调用**acldvppCreateRoiConfig**接口创建区域位置数据。
stream	输入	指定stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

8.12.4.4 acldvppVpcBatchCropAsync

函数功能

对于一张或多张输入图片，支持从每个输入图片中抠出一张或多张子图并一一对应存放到多个输出区域，输出图片区域与抠图区域不一致时会对图片再做一次缩放操作。异步接口。

函数原型

aclError acldvppVpcBatchCropAsync(acldvppChannelDesc *channelDesc,

acldvppBatchPicDesc *srcBatchPicDescs,

uint32_t *roiNums,

uint32_t size,

acldvppBatchPicDesc *dstBatchPicDescs,

acldvppRoiConfig *cropAreas[],

aclrtStream stream)

参数说明

参数名	输入/输出	说明
channelDesc	输入	指定通道描述信息，与调用**acldvppCreateChannel**接口创建通道时指定的channelDesc保持一致。
srcBatchPicDescs	输入	指定批量输入图片描述信息。 调用**acldvppCreateBatchPicDesc**接口创建批量图片描述信息。

参数名	输入/输出	说明
roiNums	输入	表示抠图的数量，当前最大抠图数量为256。 roiNums为数组，数组总和小于等于256，与dstBatchPicDescs结构体中的batchSize值保持一致。 roiNums[0]+...+roiNums[size-1] <= 256
size	输入	表示roiNums数组中的元素个数，个数小于等于256。
dstBatchPicDescs	输入&输出	指定批量输出图片描述信息。 调用**acldvppCreateBatchPicDesc**接口创建批量图片描述信息。
cropArea	输入	抠图区域位置数组。 调用**acldvppCreateRoiConfig**接口创建区域位置数据。
stream	输入	指定stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

8.12.4.5 acldvppVpcCropAndPasteAsync

函数功能

按指定区域从一个输入图片中抠出一张子图，再将抠的子图贴到目标图片的指定位置，作为输出图片。异步接口。

当pasteArea的宽高与cropArea的宽高不一致时会对图片做一次缩放操作。

函数原型

aclError acldvppVpcCropAndPasteAsync(acldvppChannelDesc *channelDesc,

acldvppPicDesc *inputDesc,

acldvppPicDesc *outputDesc,

acldvppRoiConfig *cropArea,

acldvppRoiConfig *pasteArea,

aclrtStream stream)

参数说明

参数名	输入/输出	说明
channelDesc	输入	指定通道描述信息，与调用**acldvppCreateChannel**接口创建通道时指定的channelDesc保持一致。
inputDesc	输入	输入图片信息。 • 调用**acldvppCreatePicDesc**接口创建图片描述信息，调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，图片格式、宽、高等）。 • 输入图片分辨率、图像格式、widthStride对齐要求、heightStride对齐要求等，请参见**约束说明**。
outputDesc	输入&输出	输出图片信息。 • 调用**acldvppCreatePicDesc**接口创建图片描述信息。 • 输出图像分辨率、图像格式、widthStride对齐要求、heightStride对齐要求等，请参见**约束说明**。
cropArea	输入	抠图区域位置。 调用**acldvppCreateRoiConfig**接口创建区域位置数据。
pasteArea	输入	贴图区域位置。 • 调用**acldvppCreateRoiConfig**接口创建区域位置数据。 • 贴图区域左偏移需要16对齐。
stream	输入	指定stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

8.12.4.6 acldvppVpcBatchCropAndPasteAsync

函数功能

按指定区域从一个输入图片中抠出一个或多个子图，再将抠的子图贴到目标图片的指定位置，作为输出图片。异步接口。

当pasteArea的宽高与cropArea的宽高不一致时会对图片做一次缩放操作。

函数原型

aclError acldvppVpcBatchCropAndPasteAsync (acldvppChannelDesc *channelDesc,

acldvppBatchPicDesc *srcBatchPicDesc,

uint32_t *roiNums,

uint32_t size,

acldvppBatchPicDesc *dstBatchPicDesc,

acldvppRoiConfig *cropAreas[],

acldvppRoiConfig *pasteAreas[],

aclrtStream stream)

参数说明

参数名	输入/输出	说明
channelDesc	输入	指定通道描述信息，与调用acldvppCreateChannel接口创建通道时指定的channelDesc保持一致。
srcBatchPicDescs	输入	指定批量输入图片描述信息。 调用acldvppCreateBatchPicDesc接口创建批量图片描述信息。
roiNums	输入	指定每个输入图片的抠图数量，当前最大抠图数量为256。 roiNums为数组，数组总和小于等于256，与dstBatchPicDescs结构体中的batchSize值保持一致。 roiNums[0]+...+roiNums[size-1] <= 256
size	输入	表示roiNums数组中的元素个数，个数小于等于256。
dstBatchPicDescs	输入&输出	指定批量输出图片描述信息。 调用acldvppCreateBatchPicDesc接口创建批量图片描述信息。
cropArea	输入	抠图区域位置数组。 调用acldvppCreateRoiConfig接口创建区域位置数据。 cropAreas、pasteAreas数组中的元素个数与dstBatchPicDescs结构体中的batchSize值相等。

参数名	输入/输出	说明
pasteArea	输入	贴图区域位置。 • 调用**acldvppCreateRoiConfig**接口创建区域位置数据。 • 贴图区域左偏移需要16对齐。 • cropAreas、pasteAreas数组中的元素个数与dstBatchPicDescs结构体中的batchSize值相等。
stream	输入	指定stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

8.12.4.7 acldvppVpcConvertColorAsync

函数功能

对输入图片进行进行色域转换，输出图片的宽高须与输入图片的宽高一致。异步接口。当前版本不支持该接口。

函数原型

aclError acldvppVpcConvertColorAsync(acldvppChannelDesc *channelDesc,

acldvppPicDesc *inputDesc,

acldvppPicDesc *outputDesc,

aclrtStream stream)

参数说明

参数名	输入/输出	说明
channelDesc	输入	指定通道描述信息，与调用**acldvppCreateChannel**接口创建通道时指定的channelDesc保持一致。

参数名	输入/输出	说明
inputDesc	输入	输入图片信息。 • 调用**acldvppCreatePicDesc**接口创建图片描述信息，调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，图片格式、宽、高等）。 • 输入图片分辨率、widthStride对齐要求、heightStride对齐要求等，请参见**约束说明**。 • 支持的输入图片格式如下： PIXEL_FORMAT_YUV_400, //输入图片格式为YUV400时，输出格式必须是YUV400 PIXEL_FORMAT_YVU_SEMIPLANAR_420, PIXEL_FORMAT_YUV_SEMIPLANAR_420, PIXEL_FORMAT_YUV_SEMIPLANAR_422, PIXEL_FORMAT_YVU_SEMIPLANAR_422, PIXEL_FORMAT_YVU_SEMIPLANAR_444, PIXEL_FORMAT_YUV_SEMIPLANAR_444, PIXEL_FORMAT_YUYV_PACKED_422, PIXEL_FORMAT_UYVY_PACKED_422, PIXEL_FORMAT_YVYU_PACKED_422, PIXEL_FORMAT_VYUY_PACKED_422, PIXEL_FORMAT_YUV_PACKED_444, PIXEL_FORMAT_RGB_888, PIXEL_FORMAT_BGR_888, PIXEL_FORMAT_ARGB_8888, PIXEL_FORMAT_ABGR_8888, PIXEL_FORMAT_RGBA_8888, PIXEL_FORMAT_BGRA_8888, PIXEL_FORMAT_YUV_SEMI_PLANNER_420_10BIT,

参数名	输入/输出	说明
		PIXEL_FORMAT_YVU_SEMI_PLANNER_420_10BIT
outputDesc	输入&输出	输出图片信息。 • 调用**acldvppCreatePicDesc**接口创建图片描述信息。 • 输出图像分辨率、widthStride对齐要求、heightStride对齐要求等，请参见**约束说明**。 • 支持的输出图片格式如下： PIXEL_FORMAT_YUV_400, PIXEL_FORMAT_YVU_SEMIPLANAR_420, PIXEL_FORMAT_YUV_SEMIPLANAR_420, PIXEL_FORMAT_YUV_SEMIPLANAR_422, PIXEL_FORMAT_YVU_SEMIPLANAR_422, PIXEL_FORMAT_YUV_SEMIPLANAR_444, PIXEL_FORMAT_RGB_888, PIXEL_FORMAT_BGR_888, PIXEL_FORMAT_ARGB_8888, PIXEL_FORMAT_ABGR_8888, PIXEL_FORMAT_RGBA_8888, PIXEL_FORMAT_BGRA_8888
stream	输入	指定stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

8.12.4.8 acldvppVpcPyrDownAsync

函数功能

对图像进行金字塔缩放，将输出图片缩放为输入图片的一半。异步接口。当前版本不支持该接口。

函数原型

aclError acldvppVpcPyrDownAsync(acldvppChannelDesc *channelDesc,

acldvppPicDesc *inputDesc,

acldvppPicDesc *outputDesc,

void *reserve,

aclrtStream stream)

参数说明

参数名	输入/输出	说明
channelDesc	输入	指定通道描述信息，与调用acldvppCreateChannel接口创建通道时指定的channelDesc保持一致。
inputDesc	输入	输入图片信息。 • 调用acldvppCreatePicDesc接口创建图片描述信息，调用acldvppSetPicDesc系列接口设置图片描述参数（例如，图片格式、宽、高等）。 • 输入图片分辨率：20*12~2048*2048 • 输入图片格式：PIXEL_FORMAT_YUV_400 • widthStride对齐：16对齐 • heightStride对齐：无要求
outputDesc	输入&输出	输出图片信息。 • 调用acldvppCreatePicDesc接口创建图片描述信息。 • 输出图像分辨率：输入图像分辨率/2 • 图像格式：PIXEL_FORMAT_YUV_400 • widthStride对齐：16对齐 • heightStride对齐：无要求
reserve	输入	预留金字塔缩放配置参数。
stream	输入	指定stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

8.12.5 JPEGD 功能

8.12.5.1 功能及约束说明

功能及约束说明

JPEGD（JPEG decoder）实现.jpg、.jpeg、.JPG、.JPEG图片的解码，对于硬件不支持的格式，会使用软件解码。

- 关于输入
  - 输入图片分辨率：
    - 最大分辨率：8192 * 8192，最小分辨率：32 * 32。
  - 输入图片格式
    - 只支持Huffman编码，码流的colorspace为YUV，码流的subsample为444/422/420/400；
    - 不支持算术编码；
    - 不支持渐进JPEG格式；
    - 不支持JPEG2000格式；
  - 输入内存：
    - 输入内存的大小就是指实际的输入图片所占用的大小。
    - 输入内存首地址要求128对齐。Device的内存，调用**acldvppMalloc**接口/**acldvppFree**接口申请或释放内存。
- 关于输出
  - 输出图片格式

    针对不同的源图编码格式，解码后，输出如下格式的图片：

    jpeg(YUV444SP) -> YUV444SP V在前U在后 、YUV420 SP V在前U在后、YUV420SP U在前V在后。

    jpeg(YUV422SP) -> YUV422SP V在前U在后、YUV420SP V在前U在后、YUV420SP U在前V在后。

    jpeg(YUV420SP) -> YUV420SP V在前U在后、YUV420SP U在前V在后。

    jpeg(YUV400) -> YUV420SP，UV数据采用0x80填充。
  - 输出内存：
    - 输出内存大小与图片数据的格式相关，计算公式如下：

      YUV420SP：widthStride*heightStride*3/2

      YUV422SP：widthStride*heightStride*2

      YUV444SP：widthStride*heightStride*3
    - 输出内存首地址要求128字节对齐。Device的内存，调用**acldvppMalloc**接口/**acldvppFree**接口申请或释放内存。如果申请大块内存时，内存申请计算应该是（n表示图片数量）：输出内存大小 +(n-1)*AlignTo128(输出内存大小+8)
  - 关于输出图片的宽高对齐要求：
    - 输出图片的widthStride（对齐后的宽度），对齐到128；

        - 输出图片的heightStride（对齐后的高度），对齐到16。
- 关于硬件约束：
  - 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。
- 关于软件约束：
  - 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。

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*1路 15fps。

1080p指分辨率为1920*1080的图片；4K指分辨率为3840*2160的图片。

场景举例	总帧率
1080p * 1路	128fps
1080p * n路（n≥ 2）	256fps
4k * 1路	32fps
4k * n路（n≥ 2）	64fps

8.12.5.2 acldvppJpegDecodeAsync

函数功能

解码.jpg、.jpeg、.JPG、.JPEG图片，异步接口。

函数原型

aclError acldvppJpegDecodeAsync(acldvppChannelDesc *channelDesc,

const void *data,

uint32_t size,

acldvppPicDesc *outputDesc,

aclrtStream stream)

参数说明

参数名	输入/输出	说明
channelDesc	输入	指定通道描述信息，与调用**acldvppCreateChannel**接口创建通道时指定的channelDesc保持一致。
data	输入	输入图片的内存地址。
size	输入	输入图片的实际数据大小，单位Byte。
outputDesc	输入&输出	输出图片信息。 • 调用**acldvppCreatePicDesc**接口创建图片描述信息。 • JPEG原图的宽、高可通过**acldvppJpegGetImageInfo**接口获取。 • 输出图片的内存大小可提前调用**acldvppJpegPredictDecSize**接口获取。 • 输出图片格式支持设置成jpeg源图编码格式或NV12或NV21格式。 • 输出图片的宽高对齐要求，请参见**8.12.5.1 功能及约束说明**。
stream	输入	指定stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

8.12.5.3 acldvppJpegGetImageInfo

函数功能

从Host上存放JPEG图片数据的内存中读取JPEG图片的宽、高。

函数原型

aclError acldvppJpegGetImageInfo(const void *data,

uint32_t size,

uint32_t *width,

uint32_t *height,

int32_t *components)

参数说明

参数名	输入/输出	说明
data	输入	Host上存放JPEG图片数据的内存的地址。
size	输入	内存大小，单位为byte。
width	输出	图片的宽。
height	输出	图片的高。
components	输出	颜色通道个数。

返回值说明

返回0表示成功，返回其它值表示失败。

8.12.5.4 acldvppJpegPredictDecSize

函数功能

根据Host上存放JPEG图片数据的内存计算出JPEG图片解码后所需的输出内存的大小。

函数原型

aclError acldvppJpegPredictDecSize(const void *data,

uint32_t dataSize,

acldvppPixelFormat outputPixelFormat,

uint32_t *decSize)

参数说明

参数名	输入/输出	说明
data	输入	Host上存放JPEG图片数据的内存的地址。
size	输入	内存大小，单位为byte。
outputPixelFormat	输入	解码后的输出图片的格式。
decSize	输出	JPEG图片解码后所需的输出内存的大小，单位为byte。

返回值说明

返回0表示成功，返回其它值表示失败。

8.12.6 JPEGE 功能

8.12.6.1 功能及约束说明

功能及约束说明

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。

- 关于输入
  - 输入图片分辨率

    最大分辨率：8192 * 8192，最小分辨率：32 * 32。
  - 输入图片的格式：
    - YUV422 Packed（yuyv,yvyu,uyvy,vyuy）
    - YUV420SP（NV12，NV21）
  - 输入图片的宽高对齐要求：
    - 输入图片的widthStride（对齐后的宽度），对齐到16，兼容对齐到16的倍数如128。对于YUV422packed数据，widthStride应该为输入图片宽度的两倍对齐到16。
    - 输入图片的heightStride，取值：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）。其中后一种取值的使用场景举例：JPGED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）。
  - 关于输入内存：
    - 输入内存大小与图片数据的格式相关，计算公式如下：

      YUV422packed：widthStride*heightStride

      YUV420SP：widthStride*heightStride*3/2
    - 输入内存首地址要求128对齐。Device的内存，调用**acldvppMalloc**接口/**acldvppFree**接口申请或释放内存。
- 关于输出
  - 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。
  - 关于输出内存：
    - 输出内存的大小就是指实际的编码后图片所占用的大小。
    - 输出内存首地址要求128对齐。Device的内存，调用**acldvppMalloc**接口/**acldvppFree**接口申请或释放内存。

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。

场景举例	总帧率
1080p * n路（n≥1）	64fps
4k * n路（n≥1）	16fps

8.12.6.2 acldvppJpegEncodeAsync

函数功能

将YUV格式图片编码成.jpg图片，异步接口。

函数原型

aclError acldvppJpegEncodeAsync(acldvppChannelDesc *channelDesc,

acldvppPicDesc *inputDesc,

const void *data,

uint32_t *size,

acldvppJpegeConfig *config,

aclrtStream stream);

参数说明

参数名	输入/输出	说明
channelDesc	输入	指定通道描述信息，与调用**acldvppCreateChannel**接口创建通道时指定的channelDesc保持一致。
inputDesc	输入	输入图片描述信息。 • 调用**acldvppCreatePicDesc**接口创建图片描述信息，调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，图片格式、宽、高等）。 • 输入图片分辨率、图像格式的要求，请参见**8.12.6.1 功能及约束说明**。
data	输入	输出内存地址，存放编码后的数据。

参数名	输入/输出	说明
size	输入&输出	输出内存大小，单位Byte。 size作为输入参数时，可提前调用**acldvppJpegPredictEncSize**接口预估输出内存大小。 size作为输出参数时，表示实际输出内存大小，可能与调用**acldvppJpegPredictEncSize**接口预估的内存大小存在差异，如果用户需要取用编码后的数据，请使用实际输出内存大小。
config	输入	表示图片编码配置数据。 调用**acldvppCreateJpegeConfig**接口创建图片编码配置数据，调用**acldvppSetJpegeConfigLevel**接口设置编码配置数据。
stream	输入	指定stream。

返回值说明

返回0表示任务下发成功，返回非0表示任务下发失败。

8.12.6.3 acldvppJpegPredictEncSize

函数功能

根据输入图片描述信息、图片编码配置数据预估图片编码后所需的输出内存的大小。

函数原型

aclError acldvppJpegPredictEncSize(const acldvppPicDesc *inputDesc, const acldvppJpegeConfig *config, uint32_t *size)

参数说明

参数名	输入/输出	说明
inputDesc	输入	输入图片描述信息。 调用**acldvppCreatePicDesc**接口创建图片描述信息，调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，图片格式、宽、高等）。

参数名	输入/输出	说明
config	输入	图片编码配置数据。 调用**acldvppCreateJpegeConfig**接口创建图片编码配置数据。
size	输出	预估JPEG图片编码后所需的输出内存的大小，单位为byte。 预估的输出内存会大于实际输出内存。

返回值说明

返回0表示成功，返回其它值表示失败。

8.12.7 VDEC 功能

8.12.7.1 功能及约束说明

VDEC（video decoder）实现视频的解码，VDEC内部经过VPC处理后，输出YUV420SP格式（包括NV12和NV21）的图片。

- 关于输入
  - 输入码流分辨率

    最大分辨率4096 * 4096，最小分辨率128 * 128。
  - 输入码流格式：

    H264 bp/mp/hp level5.1 YUV420编码的码流。

    H265 8/10bit level5.1 YUV420编码的码流。
  - 关于输入内存：

    Device的内存，支持调用**aclrtMalloc**接口/**aclrtFree**接口申请/释放内存，也支持调用**acldvppMalloc**/**acldvppFree**接口申请/释放内存。
- 关于输出
  - VDEC输出图像的格式为（如果不设置输出格式，默认使用YUV420SP NV12）：
    - YUV420SP NV12
    - YUV420SP NV21
  - 关于输出内存：
    - 输出内存大小与图片数据的格式相关，计算公式如下：

      YUV420SP：widthStride*heightStride*3/2
    - 输出的内存首地址要求16对齐。Device的内存，调用**aclrtMalloc**接口/**aclrtFree**接口申请或释放内存。
  - 输出图片的宽高对齐要求为：

    - 输出图片的widthStride（对齐后的宽度），对齐到16
    - 输出图片的heightStride（对齐后的高度），对齐到2
- 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不支持解码。

8.12.7.2 aclvdecSendFrame

函数功能

将待解码的输入内存和解码输出内存一起传到解码器进行解码，异步接口。

aclvdecSendFrame接口内部封装了**aclrtLaunchCallback**接口，用于在Stream的任务队列中增加一个需要在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**aclrtLaunchCallback**接口。

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。
- 针对如下典型场景下各分辨率的码流，VDEC解码的路数（n）推荐如下值：

  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。

分辨率	总性能	单路性能（n路）	每路VDEC解码的内存消耗（参考值，以实际码流为准）
3840*2160	120fps	120fps/n（推荐n=4，每路30fps）	约170MB
1920*1080	480fps	480fps/n（推荐n=16，每路30fps）	约86MB
≤1280*720	960fps	960fps/n（推荐n=32，每路30fps）	约70MB

- 关于解码路数与帧率的建议

  注意：下表中给出的规格建议供参考，如果单进程内启动的路数超过下表中的建议，则可能出现内存不足或性能不够的问题，进而导致创建解码通道失败或执行解码缓慢。

  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。

典型分辨率	单进程内各启动路数时的规格建议（依据输入帧率得出）				
-	输入帧率≥25fps	20fps<输入帧率<25fps	15fps<输入帧率≤20fps	10fps<输入帧率≤15fps	输入帧率≤10fps
≤720p	32路	32路	32路	32路	32路
1080p	16路	19路	24路	32路	32路
4K	4路	4路	6路	8路	12路

函数原型

aclError aclvdecSendFrame(aclvdecChannelDesc *channelDesc,

acldvppStreamDesc *input,

acldvppPicDesc *output,

aclvdecFrameConfig *config,

void* userData);

参数说明

参数名	输入/输出	说明
channelDesc	输入	指定通道描述信息，与调用aclvdecCreateChannel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解码的回调函数。
input	输入	输入码流描述信息，输入内存用户需提前申请。 • 调用acldvppCreateStreamDesc接口创建码流描述信息，调用acldvppSetStreamDesc系列接口设置视频码流信息的属性（例如，码流格式）。 • 输入视频码流的分辨率、视频格式的要求，请参见8.12.7.1 功能及约束说明。

参数名	输入/输出	说明
output	输入	输出图片描述信息，输出内存用户需提前申请。 • 调用**acldvppCreatePicDesc**接口创建图片描述信息，调用**acldvppSetPicDesc系列接口**设置图片描述参数（例如，图片格式、宽、高等）。 • 输出图像格式的要求，请参见**8.12.7.1 功能及约束说明**。
config	输入	解码配置，预留，当前可填NULL。
userdata	输入	用户自定义数据。

返回值说明

返回0表示成功，返回非0表示失败。

8.12.7.3 aclvdecCallback

函数功能

视频解码回调函数，同步接口。用户需自定义该回调函数。

约束说明

- 在回调函数中不能执行销毁通道操作，否则会导致程序执行死锁。
- 回调函数处理的时延应满足发帧帧率要求，否则会影响**aclvdecSendFrame**接口进行视频帧处理的实时性。
- 注销回调线程要在所有Callback执行完成后（即注销送码流线程之后）。

函数原型

void (* aclvdecCallback) (acldvppStreamDesc * input, acldvppPicDesc * output, void* userData)

参数说明

参数名	输入/输出	说明
input	输入	与**aclvdecSendFrame**接口中的input一致，表示输入码流描述信息。
output	输入	VDEC解码后的输出图片描述信息，输出内存用户需提前申请。
userData	输入	用户自定义数据。

返回值说明

无

8.12.8 VENC 功能

8.12.8.1 功能及约束说明

功能及约束说明

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。

- 关于输入
  - 输入图片分辨率
    最大分辨率1920 * 1920，最小分辨率128 * 128。
  - 输入图片格式
    YUV420SP NV12/NV21-8bit
  - 关于输入内存
    Device的内存，调用**acldvppMalloc**/**acldvppFree**申请/释放内存。
- 关于输出
  - 输出码流格式
    H264 BP/MP/HP
    H265 MP（仅支持Slice码流）
  - 关于输出内存
    不需要用户管理输出内存，由系统管理内存。

性能指标说明

1080p指分辨率为1920*1080的图片。

场景举例	总帧率
1080p * n路（一个进程对应一路）	30fps

8.12.8.2 aclvencSendFrame

函数功能

将待编码的图片传到编码器进行编码。异步接口。

约束说明

- 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。

函数原型

aclError aclvencSendFrame(aclvencChannelDesc *channelDesc,

acldvppPicDesc *input,

void *reserve,

aclvencFrameConfig *config,

void *userdata)

参数说明

参数名	输入/输出	说明
channelDesc	输入	指定通道描述信息，与调用aclvencCreateChannel接口创建通道时指定的channelDesc保持一致。 在通道描述信息中指定视频编码的回调函数。
input	输入	输入图片描述信息，输入内存用户需提前申请。 • 调用acldvppCreatePicDesc接口创建图片描述信息，调用acldvppSetPicDesc系列接口设置图片描述参数（例如，图片格式、宽、高等）。 • 输入图片的分辨率、格式的要求，请参见8.12.8.1 功能及约束说明。
reserve	输入	预留参数，暂不适用。
config	输入	单帧配置数据。
userdata	输入	用户自定义数据。

返回值说明

返回0表示成功，返回非0表示失败。

8.12.8.3 aclvencCallback

函数功能

视频编码回调函数，同步接口。用户需自定义该回调函数。

约束说明

- 在回调函数中不能执行销毁通道操作，否则会导致程序执行死锁。
- 回调函数处理的时延应满足发帧帧率要求，否则会影响aclvencSendFrame接口进行视频帧处理的实时性。
- 注销回调线程要在所有Callback执行完成后。

函数原型

void (*aclvencCallback)(acldvppPicDesc *input, acldvppStreamDesc *output, void *userdata)

参数说明

参数名	输入/输出	说明
input	输入	表示输入图片描述信息。
output	输入	VENC编码后的输出帧码流描述信息。
userdata	输入	用户自定义数据。

返回值说明

无

8.13 日志管理

8.13.1 aclAppLog

函数功能

将日志记录到日志文件中，日志文件在Host的“/var/log/npu/slog”目录下。

ACL提供ACL_APP_LOG宏，封装了aclAppLog接口，推荐用户调用ACL_APP_LOG宏，传入日志级别、日志描述、fmt中的可变参数。

```
#define ACL_APP_LOG(level, fmt, ...) \
    aclAppLog(level, __FUNCTION__, __FILE__, __LINE__, fmt, ##__VA_ARGS__)
```

函数原型

void aclAppLog(aclLogLevel logLevel, const char *func, const char *file, uint32_t line, const char *fmt, ...)

参数说明

参数名	输入/输出	说明
logLevel	输入	日志级别。 typedef enum { ACL_DEBUG = 0, ACL_INFO = 1, ACL_WARNING = 2, ACL_ERROR = 3, } aclLogLevel;
func	输入	表示用户在哪个接口中调用aclAppLog接口，固定配置为__FUNCTION__
file	输入	表示用户在哪个文件中调用aclAppLog接口，固定配置为__FILE__
line	输入	表示用户在哪一行中调用aclAppLog接口，固定配置为__LINE__
fmt	输入	日志描述。 在调用格式化函数时，fmt中参数的类型、个数必须与实际参数类型、个数保持一致。
...	输入	fmt中的可变参数，根据日志内容添加。

返回值说明

无

调用示例

```
//若fmt中存在可变参数，需提前定义
uint32_t modelId = 1;
ACL_APP_LOG(ACL_INFO, "load model success, modelId is %u", modelId);
```

8.14 特征向量检索

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。

该章节下的接口，昇腾310 AI处理器不支持。

8.14.1 aclfvInit

函数功能

初始化特征检索模块。

函数原型

aclError aclfvInit(uint64_t fsNum)

参数说明

参数名	输入/输出	说明
fsNum	输入	底库特征个数，用于系统内部给特征检索模块申请内存资源。 取值范围：[1,600000000]

返回值说明

返回0表示成功，返回其它值表示失败。

8.14.2 aclfvRelease

函数功能

特征检索模块去初始化，释放内存空间，与**aclfvInit**接口配套使用。

函数原型

aclError aclfvRelease()

参数说明

无

返回值说明

返回0表示成功，返回其它值表示失败。

8.14.3 aclfvRepoAdd

函数功能

添加底库或向已存在底库中添加特征。

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。

函数原型

aclError aclfvRepoAdd(aclfvSearchType type, aclfvFeatureInfo *featureInfo, aclrtStream stream)

参数说明

参数名	输入/输出	说明
type	输入	检索类型。
featureInfo	输入	指定特征描述信息。
stream	输入	指定stream。

返回值说明

返回0表示成功，返回其它值表示失败。

8.14.4 aclfvRepoDel

函数功能

删除底库。

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

函数原型

aclError aclfvRepoDel(aclfvSearchType type, aclfvRepoRange *repoRange, aclrtStream stream)

参数说明

参数名	输入/输出	说明
type	输入	检索类型。
repoRange	输入	指定特征删除范围。
stream	输入	指定stream。

返回值说明

返回0表示成功，返回其它值表示失败。

8.14.5 aclfvDel

函数功能

精确删除底库中某个特征，一次只能删除底库中一个特征。N:M场景下不涉及这个接口。

函数原型

aclError aclfvDel(aclfvFeatureInfo *featureInfo, aclrtStream stream)

参数说明

参数名	输入/输出	说明
featureInfo	输入	指定特征描述信息。
stream	输入	指定stream。

返回值说明

返回0表示成功，返回其它值表示失败。

8.14.6 aclfvModify

函数功能

修改底库中某个特征，一次只能修改底库中一个特征，不允许修改底库中不存在的特征。N:M场景下不涉及这个接口。

函数原型

aclError aclfvModify(aclfvFeatureInfo *featureInfo, aclrtStream stream)

参数说明

参数名	输入/输出	说明
featureInfo	输入	指定特征描述信息。
stream	输入	指定stream。

返回值说明

返回0表示成功，返回其它值表示失败。

8.14.7 aclfvSearch

函数功能

特征1:N或N:M检索。

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。

函数原型

aclError aclfvSearch(

aclfvSearchType type, aclfvSearchInput *searchInput, aclfvSearchResult *searchRst, aclrtStream stream)

参数说明

参数名	输入/输出	说明
type	输入	指定特征描述信息。
searchInput	输入	检索输入信息。
searchRst	输出	检索输出结果。 该结构体中queryCnt参数应与检索输入信息searchInput中的queryCnt参数相同。
stream	输入	指定stream。

返回值说明

返回0表示成功，返回其它值表示失败。

8.15 数据类型及其操作接口

8.15.1 aclError

```
typedef int aclError;
```

表 8-4 返回码列表

返回码	含义	可能原因及解决方法
const int ACL_ERROR_NONE = 0;	执行成功。	-
const int ACL_ERROR_INVALID_PARAM = 100000;	参数校验失败。	请检查接口的入参值是否正确。
const int ACL_ERROR_UNINITIALIZE = 100001;	ACL未初始化。	请检查是否已调用**aclInit**接口进行初始化，请确保已调用**aclInit**接口，且在其它ACL接口之前调用。

返回码	含义	可能原因及解决方法
const int ACL_ERROR_REPEAT_INITIALIZE = 100002;	ACL重复初始化或重复加载。	请检查是否重复调用**aclInit**接口进行初始化或重复调用**aclopSetModelDir**接口进行加载。
const int ACL_ERROR_INVALID_FILE = 100003;	无效的文件。	请检查文件是否存在、文件是否能被访问等。
const int ACL_ERROR_WRITE_FILE = 100004;	写文件失败。	请检查文件路径是否存在、文件是否有写权限等。
const int ACL_ERROR_INVALID_FILE_SIZE = 100005;	无效的文件大小。	请检查文件大小是否符合接口要求。
const int ACL_ERROR_PARSE_FILE = 100006;	解析文件失败。	请检查文件内容是否合法。
const int ACL_ERROR_FILE_MISSING_ATTR = 100007;	文件缺失参数。	请检查文件内容是否完整。
const int ACL_ERROR_FILE_ATTR_INVALID = 100008;	文件参数无效。	请检查文件中参数值是否正确。
const int ACL_ERROR_INVALID_DUMP_CONFIG = 100009;	无效的Dump配置。	请检查**aclInit**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导**》。
const int ACL_ERROR_INVALID_PROFILING_CONFIG = 100010;	无效的Profiling配置。	请检查Profiling配置是否正确。
const int ACL_ERROR_INVALID_MODEL_ID = 100011;	无效的模型ID。	请检查模型ID是否正确、模型是否正确加载。
const int ACL_ERROR_DESERIALIZE_MODEL = 100012;	反序列化模型失败。	模型可能与当前版本不匹配，请参见《**ATC工具使用指导**》重新转换模型。
const int ACL_ERROR_PARSE_MODEL = 100013;	解析模型失败。	模型可能与当前版本不匹配，请参见《**ATC工具使用指导**》重新转换模型。

返回码	含义	可能原因及解决方法
const int ACL_ERROR_READ_MODEL_FAILURE = 100014;	读取模型失败。	请检查模型文件是否存在、模型文件是否能被访问等。
const int ACL_ERROR_MODEL_SIZE_INVALID = 100015;	无效的模型大小。	模型文件无效，请参见《**ATC工具使用指导**》重新转换模型。
const int ACL_ERROR_MODEL_MISSING_ATTR = 100016;	模型缺少参数。	模型可能与当前版本不匹配，请参见《**ATC工具使用指导**》重新转换模型。
const int ACL_ERROR_MODEL_INPUT_NOT_MATCH = 100017;	模型的输入不匹配。	请检查模型的输入是否正确。
const int ACL_ERROR_MODEL_OUTPUT_NOT_MATCH = 100018;	模型的输出不匹配。	请检查模型的输出是否正确。
const int ACL_ERROR_MODEL_NOT_DYNAMIC = 100019;	非动态模型。	请检查当前模型是否支持动态场景，如不支持，请参见《**ATC工具使用指导**》重新转换模型。
const int ACL_ERROR_OP_TYPE_NOT_MATCH = 100020;	单算子类型不匹配。	请检查算子类型是否正确。
const int ACL_ERROR_OP_INPUT_NOT_MATCH = 100021;	单算子的输入不匹配。	请检查算子的输入是否正确。
const int ACL_ERROR_OP_OUTPUT_NOT_MATCH = 100022;	单算子的输出不匹配。	请检查算子的输出是否正确。
const int ACL_ERROR_OP_ATTR_NOT_MATCH = 100023;	单算子的属性不匹配。	请检查算子的属性是否正确。
const int ACL_ERROR_OP_NOT_FOUND = 100024;	单算子未找到。	请检查算子类型是否支持。
const int ACL_ERROR_OP_LOAD_FAILED = 100025;	单算子加载失败。	模型可能与当前版本不匹配，请参见《**ATC工具使用指导**》重新转换模型。

返回码	含义	可能原因及解决方法
const int ACL_ERROR_UNSUPPORTED_DATA_TYPE = 100026;	不支持的数据类型。	请检查数据类型是否存在或当前是否支持。
const int ACL_ERROR_FORMAT_NOT_MATCH = 100027;	Format不匹配。	请检查Format是否正确。
const int ACL_ERROR_BIN_SELECTOR_NOT_REGISTERED = 100028;	使用二进制选择方式编译算子接口时，算子未注册选择器。	请检查是否调用**aclopRegisterSelectKernelFunc**接口注册算子选择器。
const int ACL_ERROR_KERNEL_NOT_FOUND = 100029;	编译算子时，算子Kernel未注册。	请检查是否调用**aclopCreateKernel**接口注册算子Kernel。
const int ACL_ERROR_BIN_SELECTOR_ALREADY_REGISTERED = 100030;	使用二进制选择方式编译算子接口时，算子重复注册。	请检查是否重复调用**aclopRegisterSelectKernelFunc**接口注册算子选择器。
const int ACL_ERROR_KERNEL_ALREADY_REGISTERED = 100031;	编译算子时，算子Kernel重复注册。	请检查是否重复调用**aclopCreateKernel**接口注册算子Kernel。
const int ACL_ERROR_INVALID_QUEUE_ID = 100032;	无效的队列ID。	请检查队列ID是否正确。
const int ACL_ERROR_REPEAT_SUBSCRIBE = 100033;	重复订阅。	请检查针对同一个Stream，是否重复调用**aclrtSubscribeReport**接口。
const int ACL_ERROR_STREAM_NOT_SUBSCRIBE = 100034;	Stream未订阅。	请检查是否已调用**aclrtSubscribeReport**接口。
const int ACL_ERROR_THREAD_NOT_SUBSCRIBE = 100035;	线程未订阅。	请检查是否已调用**aclrtSubscribeReport**接口。

返回码	含义	可能原因及解决方法
const int ACL_ERROR_WAIT_CALLBACK_TIMEOUT = 100036;	等待callback超时。	请检查是否已调用**aclrtLaunchCallback**接口下发callback任务； 请检查**aclrtProcessReport**接口中超时时间是否合理； 请检查callback任务是否已经处理完成，如果已处理完成，但还调用**aclrtProcessReport**接口，则需优化代码逻辑。
const int ACL_ERROR_REPEAT_FINALIZE = 100037;	重复去初始化。	请检查是否重复调用**aclFinalize**接口进行去初始化。
const int ACL_ERROR_NOT_STATIC_AIPP = 100038;	AIPP配置信息不存在。	调用**aclmdlGetFirstAippInfo**接口时，请传入正确的index值。
const int ACL_ERROR_BAD_ALLOC = 200000;	申请内存失败。	请检查硬件环境上的内存剩余情况。
const int ACL_ERROR_API_NOT_SUPPORT = 200001;	接口不支持。	请检查调用的接口当前是否支持。
const int ACL_ERROR_INVALID_DEVICE = 200002;	无效的Device。	请检查Device是否存在。
const int ACL_ERROR_MEMORY_ADDRESS_UNALIGNED = 200003;	内存地址未对齐。	请检查内存地址是否符合接口要求。
const int ACL_ERROR_RESOURCE_NOT_MATCH = 200004;	资源不匹配。	请检查调用接口时，是否传入正确的Stream、Context等资源。
const int ACL_ERROR_INVALID_RESOURCE_HANDLE = 200005;	无效的资源句柄。	请检查调用接口时，传入的Stream、Context等资源是否已被销毁或占用。
const int 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》。

返回码	含义	可能原因及解决方法
const int ACL_ERROR_STORAGE_OVER_LIMIT = 300000;	超出存储上限。	请检查硬件环境上的存储剩余情况。
const int 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》。
const int 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》。
const int 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》。
const int 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》。
const int 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》。
const int ACL_ERROR_PROFILING_FAILURE = 500005;	Profiling相关错误。	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》。

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。

8.15.2 aclDataType

```
typedef enum {
  ACL_DT_UNDEFINED = -1,  //未知数据类型，默认值。
```

```
  ACL_FLOAT = 0,
  ACL_FLOAT16 = 1,
  ACL_INT8 = 2,
  ACL_INT32 = 3,
  ACL_UINT8 = 4,
  ACL_INT16 = 6,
  ACL_UINT16 = 7,
  ACL_UINT32 = 8,
  ACL_INT64 = 9,
  ACL_UINT64 = 10,
  ACL_DOUBLE = 11,
  ACL_BOOL = 12,
} aclDataType;
```

8.15.2.1 aclDataTypeSize

函数功能

获取aclDataType数据的大小，单位Byte，同步接口。

函数原型

size_t aclDataTypeSize(aclDataType dataType)

参数说明

参数名	输入/输出	说明
dataType	输入	指定要获取大小的aclDataType数据。

返回值说明

返回aclDataType数据的大小，单位Byte。

8.15.3 aclFormat

```
typedef enum {
    ACL_FORMAT_UNDEFINED = -1,
    ACL_FORMAT_NCHW = 0,
    ACL_FORMAT_NHWC = 1,
    ACL_FORMAT_ND = 2,
    ACL_FORMAT_NC1HWC0 = 3,
    ACL_FORMAT_FRACTAL_Z = 4,
    ACL_FORMAT_FRACTAL_NZ = 29,
} aclFormat;
```

- UNDEFINED：未知格式，默认值。
- ND：表示支持任意格式，仅有Square、Tanh等这些单输入对自身处理的算子外，其它需要慎用。
- NCHW：NCHW格式。
- NHWC：NHWC格式。
- NC1HWC0：华为自研的5维数据格式。其中，C0与微架构强相关，该值等于cube单元的size，例如16；C1是将C维度按照C0切分：C1=C/C0， 若结果不整除，最后一份数据需要padding到C0。

- FRACTAL_Z：卷积的权重的格式。
- FRACTAL_NZ：内部格式，用户目前无需使用。

8.15.4 acldvppPixelFormat

```
// Supported Pixel Format
enum acldvppPixelFormat {
    PIXEL_FORMAT_YUV_400 = 0, // 0, YUV400 8bit
    PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // 1, YUV420SP NV12 8bit
    PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // 2, YUV420SP NV21 8bit
    PIXEL_FORMAT_YUV_SEMIPLANAR_422 = 3, // 3, YUV422SP NV12 8bit
    PIXEL_FORMAT_YVU_SEMIPLANAR_422 = 4, // 4, YUV422SP NV21 8bit
    PIXEL_FORMAT_YUV_SEMIPLANAR_444 = 5, // 5, YUV444SP NV12 8bit
    PIXEL_FORMAT_YVU_SEMIPLANAR_444 = 6, // 6, YUV444SP NV21 8bit
    PIXEL_FORMAT_YUYV_PACKED_422 = 7, // 7, YUV422P YUYV 8bit
    PIXEL_FORMAT_UYVY_PACKED_422 = 8, // 8, YUV422P UYVY 8bit
    PIXEL_FORMAT_YVYU_PACKED_422 = 9, // 9, YUV422P YVYU 8bit
    PIXEL_FORMAT_VYUY_PACKED_422 = 10, // 10, YUV422P VYUY 8bit
    PIXEL_FORMAT_YUV_PACKED_444 = 11, // 11, YUV444P 8bit
    PIXEL_FORMAT_RGB_888 = 12, // 12, RGB888
    PIXEL_FORMAT_BGR_888 = 13, // 13, BGR888
    PIXEL_FORMAT_ARGB_8888 = 14, // 14, ARGB8888
    PIXEL_FORMAT_ABGR_8888 = 15, // 15, ABGR8888
    PIXEL_FORMAT_RGBA_8888 = 16, // 16, RGBA8888
    PIXEL_FORMAT_BGRA_8888 = 17, // 17, BGRA8888
    PIXEL_FORMAT_YUV_SEMI_PLANNER_420_10BIT = 18, // 18, YUV420SP 10bit
    PIXEL_FORMAT_YVU_SEMI_PLANNER_420_10BIT = 19, // 19, YVU420sp 10bit
    PIXEL_FORMAT_YVU_PLANAR_420 = 20, // 20, YUV420P 8bit
    PIXEL_FORMAT_YVU_PLANAR_422, //YUV422P 8bit
    PIXEL_FORMAT_YVU_PLANAR_444,  //YUV444P 8bit
    PIXEL_FORMAT_RGB_444 = 23,
    PIXEL_FORMAT_BGR_444,
    PIXEL_FORMAT_ARGB_4444,
    PIXEL_FORMAT_ABGR_4444,
    PIXEL_FORMAT_RGBA_4444,
    PIXEL_FORMAT_BGRA_4444,
    PIXEL_FORMAT_RGB_555,
    PIXEL_FORMAT_BGR_555,
    PIXEL_FORMAT_RGB_565,
    PIXEL_FORMAT_BGR_565,
    PIXEL_FORMAT_ARGB_1555,
    PIXEL_FORMAT_ABGR_1555,
    PIXEL_FORMAT_RGBA_1555,
    PIXEL_FORMAT_BGRA_1555,
    PIXEL_FORMAT_ARGB_8565,
    PIXEL_FORMAT_ABGR_8565,
    PIXEL_FORMAT_RGBA_8565,
    PIXEL_FORMAT_BGRA_8565,
    PIXEL_FORMAT_RGB_BAYER_8BPP = 50,
    PIXEL_FORMAT_RGB_BAYER_10BPP,
    PIXEL_FORMAT_RGB_BAYER_12BPP,
    PIXEL_FORMAT_RGB_BAYER_14BPP,
    PIXEL_FORMAT_RGB_BAYER_16BPP,
    PIXEL_FORMAT_BGR_888_PLANAR = 70,
    PIXEL_FORMAT_HSV_888_PACKAGE,
    PIXEL_FORMAT_HSV_888_PLANAR,
    PIXEL_FORMAT_LAB_888_PACKAGE,
    PIXEL_FORMAT_LAB_888_PLANAR,
    PIXEL_FORMAT_S8C1,
    PIXEL_FORMAT_S8C2_PACKAGE,
    PIXEL_FORMAT_S8C2_PLANAR,
    PIXEL_FORMAT_S16C1,
    PIXEL_FORMAT_U8C1,
    PIXEL_FORMAT_U16C1,
    PIXEL_FORMAT_S32C1,
    PIXEL_FORMAT_U32C1,
    PIXEL_FORMAT_U64C1,
    PIXEL_FORMAT_S64C1,
```



```
    PIXEL_FORMAT_YUV_SEMIPLANAR_440 = 1000,
    PIXEL_FORMAT_YVU_SEMIPLANAR_440,
    PIXEL_FORMAT_FLOAT32,
    PIXEL_FORMAT_BUTT,
    PIXEL_FORMAT_UNKNOWN = 10000
};
```

8.15.5 acldvppStreamFormat

```
enum acldvppStreamFormat {
    H265_MAIN_LEVEL = 0,
    H264_BASELINE_LEVEL,
    H264_MAIN_LEVEL,
    H264_HIGH_LEVEL
};
```

8.15.6 aclrtContext

```
typedef void *aclrtContext;
```

8.15.7 aclrtStream

```
typedef void *aclrtStream;
```

8.15.8 aclrtEvent

```
typedef void *aclrtEvent;
```

8.15.9 aclrtEventStatus

```
typedef enum aclrtEventStatus {
    ACL_EVENT_STATUS_COMPLETE  = 0, //完成
    ACL_EVENT_STATUS_NOT_READY = 1, //未完成
    ACL_EVENT_STATUS_RESERVED  = 2, //预留
} aclrtEventStatus;
```

8.15.10 aclTransType

```
typedef enum aclTransType
{
    ACL_TRANS_N,    //不转置，默认为不转置
    ACL_TRANS_T,    //转置
    ACL_TRANS_NZ,   //内部格式，能够提供更好的接口性能，建议使用场景为：某个输入矩阵作为底库，执行一次转NZ格式的操作后，多次使用NZ格式的算子，提升性能。
    ACL_TRANS_NZ_T  //内部格式转置，当前不支持
}aclTransType;
```

8.15.11 aclComputeType

```
typedef enum aclComputeType
{
    ACL_COMPUTE_HIGH_PRECISION,
    ACL_COMPUTE_LOW_PRECISION    //当前不支持
}aclComputeType;
```

8.15.12 aclfvSearchType

该枚举值，昇腾310 AI处理器不支持。

```
enum aclfvSearchType {
    SEARCH_1_N,  // 1:N operation type
    SEARCH_N_M   // N:M operation type
};
```


8.15.13 aclAippInputFormat

```
typedef enum
{
    ACL_YUV420SP_U8 = 1,  //YUV420SP_U8
    ACL_XRGB8888_U8=2,    //XRGB8888_U8
    ACL_RGB888_U8=3,      //RGB888_U8
    ACL_YUV400_U8=4,      //YUV400_U8
    /*上述4个动态AIPP和静态AIPP均支持*/
    ACL_NC1HWC0DI_FP16 = 5;//暂不支持
    ACL_NC1HWC0DI_S8 = 6;//暂不支持
    ACL_ARGB8888_U8 = 7;   //暂不支持
    ACL_YUYV_U8 = 8;     //暂不支持
    ACL_YUV422SP_U8 = 9;  //暂不支持
    ACL_AYUV444_U8 = 10;  //暂不支持
    ACL_RAW10 = 11;  //暂不支持
    ACL_RAW12 = 12;  //暂不支持
    ACL_RAW16 = 13;  //暂不支持
    ACL_RAW24 = 14;  //暂不支持
    ACL_AIPP_RESERVED = 0xffff,
} aclAippInputFormat;
```

8.15.14 aclAippInfo

```
typedef struct aclAippInfo {
    aclAippInputFormat inputFormat;
    int32_t srcImageSizeW;
    int32_t srcImageSizeH;
    int8_t cropSwitch;
    int32_t loadStartPosW;
    int32_t loadStartPosH;
    int32_t cropSizeW;
    int32_t cropSizeH;
    int8_t resizeSwitch;
    int32_t resizeOutputW;
    int32_t resizeOutputH;
    int8_t paddingSwitch;
    int32_t leftPaddingSize;
    int32_t rightPaddingSize;
    int32_t topPaddingSize;
    int32_t bottomPaddingSize;
    int8_t cscSwitch;
    int8_t rbuvSwapSwitch;
    int8_t axSwapSwitch;
    int8_t singleLineMode;
    int32_t matrixR0C0;
    int32_t matrixR0C1;
    int32_t matrixR0C2;
    int32_t matrixR1C0;
    int32_t matrixR1C1;
    int32_t matrixR1C2;
    int32_t matrixR2C0;
    int32_t matrixR2C1;
    int32_t matrixR2C2;
    int32_t outputBias0;
    int32_t outputBias1;
    int32_t outputBias2;
    int32_t inputBias0;
    int32_t inputBias1;
    int32_t inputBias2;
    int32_t meanChn0;
    int32_t meanChn1;
    int32_t meanChn2;
    int32_t meanChn3;
    float minChn0;
    float minChn1;
    float minChn2;
    float minChn3;
```



```
    float varReciChn0;
    float varReciChn1;
    float varReciChn2;
    float varReciChn3;
    aclFormat srcFormat;  //模型转换前，原始模型的输入format
    aclDataType srcDatatype;  //模型转换前，原始模型的输入datatype
    size_t srcDimNum;   //模型转换前，原始模型输入dims
    size_t shapeCount;  //动态Shape（动态Batch或动态分辨率）场景下的档位数
    aclAippDims outDims[ACL_MAX_SHAPE_COUNT];
    aclAippExtendInfo *aippExtend; //预留参数，当前版本用户必须传入空指针
} aclAippInfo;
```

8.15.15 aclAippDims

```
typedef struct aclAippDims {
    aclmdlIODims srcDims; //模型转换前的dims，如[1,3,150,150]
    size_t srcSize; //模型转换前输入数据的大小
    aclmdlIODims aippOutdims; //内部数据格式的dims信息
    size_t aippOutSize; //经过AIPP处理后的输出数据的大小
} aclAippDims;
```

8.15.16 aclmdlIODims

```
const int ACL_MAX_DIM_CNT = 128;
const int ACL_MAX_TENSOR_NAME_LEN = 128;
typedef struct aclmdlIODims {
    char name[ACL_MAX_TENSOR_NAME_LEN]; /**< tensor name */
    size_t dimCount;  /**Shape中的维度个数*/
    int64_t dims[ACL_MAX_DIM_CNT]; /**< 维度信息 */
} aclmdlIODims;
```

8.15.17 aclmdlAIPP

该数据类型下的操作接口都是同步接口。

8.15.17.1 aclmdlCreateAIPP

函数功能

动态AIPP场景下，根据模型支持的Batch数创建的aclmdlCreateAIPP类型的数据，用于存放动态AIPP的参数。

函数原型

aclmdlAIPP *aclmdlCreateAIPP(uint64_t batchSize)

参数说明

参数名	输入/输出	说明
batchSize	输入	模型的Batch数。

返回值说明

aclmdlAIPP地址。

8.15.17.2 设置动态 AIPP 参数

8.15.17.2.1 aclmdlSetAIPPInputFormat

函数功能

动态AIPP场景下，设置原始输入图像的格式，同步接口。

函数原型

aclError aclmdlSetAIPPInputFormat(aclmdlAIPP *aippParmsSet, aclAippInputFormat inputFormat)

参数说明

参数名	输入/输出	说明
aippParmsSet	输入	动态AIPP参数对象。 提前调用aclmdlCreateAIPP接口创建aclmdlAIPP类型的数据。
inputFormat	输入	表示原始输入图像的格式。 取值范围如下： • ACL_YUV420SP_U8 • ACL_XRGB8888_U8 • ACL_RGB888_U8 • ACL_YUV400_U8

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.17.2.2 aclmdlSetAIPPCscParams

函数功能

动态AIPP场景下，设置CSC色域转换相关的参数，若色域转换开关关闭，则调用该接口设置的参数无效。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，您可以参考《ATC工具使用指导》中的**色域转换配置说明**章节中的转换前图片格式、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。

函数原型

aclError aclmdlSetAIPPCscParams(aclmdlAIPP *aippParmsSet, int8_t csc_switch,

int16_t cscMatrixR0C0, int16_t cscMatrixR0C1, int16_t cscMatrixR0C2,

int16_t cscMatrixR1C0, int16_t cscMatrixR1C1,int16_t cscMatrixR1C2,

int16_t cscMatrixR2C0, int16_t cscMatrixR2C1, int16_t cscMatrixR2C2,

uint8_t cscOutputBiasR0, uint8_t cscOutputBiasR1, uint8_t cscOutputBiasR2,

uint8_t cscInputBiasR0, uint8_t cscInputBiasR1, uint8_t cscInputBiasR2)

参数说明

参数名	输入/输出	说明
aippParmsSet	输入	动态AIPP参数对象。 提前调用aclmdlCreateAIPP接口创建aclmdlAIPP类型的数据。
csc_switch	输入	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
cscMatrixR0C0	输入	色域转换矩阵参数。 取值范围：[-32677 ,32676]
cscMatrixR0C1	输入	色域转换矩阵参数。 取值范围：[-32677 ,32676]
cscMatrixR0C2	输入	色域转换矩阵参数。 取值范围：[-32677 ,32676]
cscMatrixR1C0	输入	色域转换矩阵参数。 取值范围：[-32677 ,32676]
cscMatrixR1C1	输入	色域转换矩阵参数。 取值范围：[-32677 ,32676]
cscMatrixR1C2	输入	色域转换矩阵参数。 取值范围：[-32677 ,32676]
cscMatrixR2C0	输入	色域转换矩阵参数。 取值范围：[-32677 ,32676]

参数名	输入/输出	说明
cscMatrixR2C1	输入	色域转换矩阵参数。 取值范围：[-32677 ,32676]
cscMatrixR2C2	输入	色域转换矩阵参数。 取值范围：[-32677 ,32676]
cscOutputBiasR0	输入	RGB转YUV时的输出偏移，默认值为0。 取值范围：[0, 255]
cscOutputBiasR1	输入	RGB转YUV时的输出偏移，默认值为0。 取值范围：[0, 255]
cscOutputBiasR2	输入	RGB转YUV时的输出偏移，默认值为0。 取值范围：[0, 255]
cscInputBiasR0	输入	YUV转RGB时的输入偏移，默认值为0。 取值范围：[0, 255]
cscInputBiasR1	输入	YUV转RGB时的输入偏移，默认值为0。 取值范围：[0, 255]
cscInputBiasR2	输入	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

返回0表示成功，返回其它值表示失败。

8.15.17.2.3 aclmdlSetAIPPRbuvSwapSwitch

函数功能

动态AIPP场景下，设置是否交换R通道与B通道、或者是否交换U通道与V通道，同步接口。

函数原型

aclError aclmdlSetAIPPRbuvSwapSwitch(aclmdlAIPP *aippParmsSet, int8_t rbuvSwapSwitch)

参数说明

参数名	输入/输出	说明
aippParmsSet	输入	动态AIPP参数对象。 提前调用**aclmdlCreateAIPP**接口创建aclmdlAIPP类型的数据。
rbuvSwapSwitch	输入	表示是否交换R通道与B通道、或者是否交换U通道与V通道的开关，取值范围： ● 0：不交换，默认为0 ● 1：交换

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.17.2.4 aclmdlSetAIPPAxSwapSwitch

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。

函数原型

aclError aclmdlSetAIPPAxSwapSwitch(aclmdlAIPP *aippParmsSet, int8_t axSwapSwitch)

参数说明

参数名	输入/输出	说明
aippParmsSet	输入	动态AIPP参数对象。 提前调用**aclmdlCreateAIPP**接口创建aclmdlAIPP类型的数据。
axSwapSwitch	输入	表示RGBA->ARGB或者YUVA->AYUV的交换开关，取值范围： ● 0：不交换，默认为0 ● 1：交换

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.17.2.5 aclmdlSetAIPPSrcImageSize

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。

函数原型

aclError aclmdlSetAIPPSrcImageSize(aclmdlAIPP *aippParmsSet, int32_t srcImageSizeW, int32_t srcImageSizeH)

参数说明

参数名	输入/输出	说明
aippParmsSet	输入	动态AIPP参数对象。 提前调用aclmdlCreateAIPP接口创建aclmdlAIPP类型的数据。
srcImageSizeW	输入	原始图片的宽，对于YUV420SP_U8类型的图像，要求取值是偶数。 取值范围： [0,4096]
srcImageSizeH	输入	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： [0,4096]

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.17.2.6 aclmdlSetAIPPScfParams

函数功能

动态AIPP场景下，设置缩放相关的参数，同步接口。当前版本不支持该接口。

函数原型

aclError aclmdlSetAIPPScfParams(aclmdlAIPP *aippParmsSet, int8_t scfSwitch,

int32_t scfInputSizeW, int32_t scfInputSizeH,

int32_t scfOutputSizeW, int32_t scfOutputSizeH,

uint64_t batchIndex)

约束说明

缩放比例scfOutputSizeW/scfInputSizeW∈[1/16,16]、scfOutputSizeH/scfInputSizeH∈[1/16,16]。

参数说明

参数名	输入/输出	说明
aippParmsSet	输入	动态AIPP参数对象。 提前调用**aclmdlCreateAIPP**接口创建aclmdlAIPP类型的数据。
scfSwitch	输入	是否对图片执行缩放操作，取值范围： • 0：不执行缩放操作，默认为0 • 1：执行缩放操作
scfInputSizeW	输入	缩放前图片的宽。 取值范围：[0,4096]
scfInputSizeH	输入	缩放前图片的高。 取值范围：[0,4096]
scfOutputSizeW	输入	缩放后图片的宽。 取值范围：[0,4096]
scfOutputSizeH	输入	缩放后图片的高。 取值范围：[0,4096]
batchIndex	输入	指定对第几个Batch上的图片执行缩放操作，默认为0。 取值范围：[0,batchSize) batchSize是在调用**aclmdlCreateAIPP**接口创建aclmdlAIPP类型的数据时设置。

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.17.2.7 aclmdlSetAIPPCropParams

函数功能

动态AIPP场景下，设置抠图相关的参数，同步接口。

函数原型

aclError aclmdlSetAIPPCropParams(aclmdlAIPP *aippParmsSet, int8_t cropSwitch,

int32_t cropStartPosW, int32_t cropStartPosH,

int32_t cropSizeW, int32_t cropSizeH,

uint64_t batchIndex)

约束说明

若开启抠图功能，则通过**aclmdlSetAIPPSrcImageSize**接口设置的参数与通过aclmdlSetAIPPCropParams接口设置的参数之间必须满足以下公式：

- srcImageSizeW≥cropSizeW+cropStartPosW
- srcImageSizeH≥cropSizeH+cropStartPosH

参数说明

参数名	输入/输出	说明
aippParmsSet	输入	动态AIPP参数对象。 提前调用**aclmdlCreateAIPP**接口创建aclmdlAIPP类型的数据。
cropSwitch	输入	是否对图片执行抠图操作，取值范围： ● 0：不执行抠图操作，默认为0 ● 1：执行抠图操作
cropStartPosW	输入	抠图时，坐标点起始位置在图中横向的坐标。 对于YUV420SP_U8格式的图像，参数取值要求是偶数。 取值范围：[0,4096]
cropStartPosH	输入	抠图时，坐标点起始位置在图中纵向的坐标。 对于YUV420SP_U8格式的图像，参数取值要求是偶数。 取值范围：[0,4096]
cropSizeW	输入	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：[0,4096]
cropSizeH	输入	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：[0,4096]
batchIndex	输入	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,batchSize) batchSize是在调用**aclmdlCreateAIPP**接口创建aclmdlAIPP类型的数据时设置。

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.17.2.8 aclmdlSetAIPPPaddingParams

函数功能

动态AIPP场景下，设置补边相关的参数，同步接口。

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)

参数说明

参数名	输入/输出	说明
aippParmsSet	输入	动态AIPP参数对象。 提前调用aclmdlCreateAIPP接口创建aclmdlAIPP类型的数据。
paddingSwitch	输入	是否对图片执行补边操作，取值范围： • 0：不执行补边操作，默认为0 • 1：执行补边操作
paddingSizeTop	输入	在图片上方填充的值。 取值范围：[0, 32]
paddingSizeBottom	输入	在图片下方填充的值。 取值范围：[0, 32]
paddingSizeLeft	输入	在图片左方填充的值。 取值范围：[0, 32]
paddingSizeRight	输入	在图片右方填充的值。 取值范围：[0, 32]
batchIndex	输入	指定对第几个Batch上的图片执行补边操作，默认为0。 取值范围：[0,batchSize) batchSize是在调用aclmdlCreateAIPP接口创建aclmdlAIPP类型的数据时设置。

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.17.2.9 aclmdlSetAIPPDtcPixelMean

函数功能

动态AIPP场景下，设置通道的均值，同步接口。

函数原型

aclError aclmdlSetAIPPDtcPixelMean(aclmdlAIPP *aippParmsSet,

int16_t dtcPixelMeanChn0,

int16_t dtcPixelMeanChn1,

int16_t dtcPixelMeanChn2,

int16_t dtcPixelMeanChn3,

uint64_t batchIndex)

参数说明

参数名	输入/输出	说明
aippParmsSet	输入	动态AIPP参数对象。 提前调用aclmdlCreateAIPP接口创建aclmdlAIPP类型的数据。
dtcPixelMeanChn0	输入	通道0的均值。 取值范围：[0, 255]
dtcPixelMeanChn1	输入	通道1的均值。 取值范围：[0, 255]
dtcPixelMeanChn2	输入	通道2的均值。 取值范围：[0, 255]
dtcPixelMeanChn3	输入	通道3的均值。 如果只有3个通道，则该参数传默认值0。 取值范围：[0, 255]

参数名	输入/输出	说明
batchIndex	输入	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,batchSize) batchSize是在调用aclmdlCreateAIPP接口创建aclmdlAIPP类型的数据时设置。

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.17.2.10 aclmdlSetAIPPDtcPixelMin

函数功能

动态AIPP场景下，设置通道的最小值，同步接口。

函数原型

aclError aclmdlSetAIPPDtcPixelMin(aclmdlAIPP *aippParmsSet,

float dtcPixelMinChn0,

float dtcPixelMinChn1,

float dtcPixelMinChn2,

float dtcPixelMinChn3,

uint64_t batchIndex)

参数说明

参数名	输入/输出	说明
aippParmsSet	输入	动态AIPP参数对象。 提前调用aclmdlCreateAIPP接口创建aclmdlAIPP类型的数据。
dtcPixelMinChn0	输入	通道0的最小值。 取值范围：[0, 255]
dtcPixelMinChn1	输入	通道1的最小值。 取值范围：[0, 255]
dtcPixelMinChn2	输入	通道2的最小值。 取值范围：[0, 255]

参数名	输入/输出	说明
dtcPixelMinChn3	输入	通道3的最小值。如果只有3个通道，则该参数传默认值0。 取值范围：[0, 255]
batchIndex	输入	指定对第几个Batch上的图片设置通道最小值，默认为0。 取值范围：[0,batchSize) batchSize是在调用aclmdlCreateAIPP接口创建aclmdlAIPP类型的数据时设置。

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.17.2.11 aclmdlSetAIPPPixelVarReci

函数功能

动态AIPP场景下，设置通道的方差，同步接口。

函数原型

aclError aclmdlSetAIPPPixelVarReci(aclmdlAIPP *aippParmsSet,

float dtcPixelVarReciChn0,

float dtcPixelVarReciChn1,

float dtcPixelVarReciChn2,

float dtcPixelVarReciChn3,

uint64_t batchIndex)

参数说明

参数名	输入/输出	说明
aippParmsSet	输入	动态AIPP参数对象。 提前调用aclmdlCreateAIPP接口创建aclmdlAIPP类型的数据。
dtcPixelVarReciChn0	输入	通道0的方差。 取值范围：[-65504, 65504]
dtcPixelVarReciChn1	输入	通道1的方差。 取值范围：[-65504, 65504]

参数名	输入/输出	说明
dtcPixelVarReciChn2	输入	通道2的方差。 取值范围：[-65504, 65504]
dtcPixelVarReciChn3	输入	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]
batchIndex	输入	指定对第几个Batch上的图片设置通道的方差，默认为0。 取值范围：[0,batchSize) batchSize是在调用aclmdlCreateAIPP接口创建aclmdlAIPP类型的数据时设置。

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.17.3 aclmdlDestroyAIPP

函数功能

销毁aclmdlAIPP类型的数据。

函数原型

aclError aclmdlDestroyAIPP(const aclmdlAIPP *aippParmsSet)

参数说明

参数名	输入/输出	说明
aippParmsSet	输入	待销毁的aclmdlAIPP的指针。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.18 aclopHandle

该数据类型下的操作接口都是同步接口。

8.15.18.1 aclopCreateHandle

函数功能

创建一个执行算子的handle。

函数原型

aclError aclopCreateHandle(const char *opType,

int numInputs,

const aclTensorDesc *const inputDesc[],

int numOutputs,

const aclTensorDesc *const outputDesc[],

const aclopAttr *opAttr,

aclopHandle **handle);

参数说明

参数名	输入/输出	说明
opType	输入	指定算子类型名称。
numInputs	输入	算子输入tensor的数量。
inputDesc	输入	算子输入tensor的描述。
numOutputs	输入	算子输出tensor的数量。
outputDesc	输入	算子输出tensor的描述。
opAttr	输入	算子属性。
handle	输出	aclopHandle数据指针的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.18.2 aclopDestroyHandle

函数功能

销毁一个执行算子的handle。

函数原型

void aclopDestroyHandle(aclopHandle *handle)

参数说明

参数名	输入/输出	说明
handle	输入	待销毁的aclopHandle的指针。

返回值说明

无

8.15.19 aclFloat16

```
typedef uint16_t aclFloat16;
```

该数据类型下的操作接口都是同步接口。

8.15.19.1 aclFloat16ToFloat

函数功能

将**aclFloat16**类型的数据转换为float（指float32）类型的数据。

函数原型

float aclFloat16ToFloat(aclFloat16 value)

参数说明

参数名	输入/输出	说明
value	输入	待转换的数据。

返回值说明

转换后的数据。

8.15.19.2 aclFloatToFloat16

函数功能

将float（指float32）类型的数据转换为**aclFloat16**类型的数据。

函数原型

aclFloat16 aclFloatToFloat16(float value)

参数说明

参数名	输入/输出	说明
value	输入	待转换的数据。

返回值说明

转换后的数据。

8.15.20 aclDataBuffer

您可以通过以下接口获取aclDataBuffer类型的数据、并对aclDataBuffer类型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。

aclDataBuffer主要用于描述内存地址、大小等内存信息。

8.15.20.1 aclCreateDataBuffer

函数功能

创建aclDataBuffer类型的数据。

函数原型

aclDataBuffer *aclCreateDataBuffer(void *data, size_t size)

参数说明

参数名	输入/输出	说明
data	输入	内存地址。 该内存需由用户自行管理，调用aclrtMalloc接口申请内存，调用aclrtFree接口释放内存。
size	输入	内存大小，单位Byte。 如果用户需要使用空tensor，则在申请内存时，内存大小最小为1Byte。

返回值说明

aclDataBuffer地址。

8.15.20.2 aclDestroyDataBuffer

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**aclCreateDataBuffer**接口创建aclDataBuffer类型数据时传入的data的内存需由用户自行释放。

函数原型

aclError aclDestroyDataBuffer(const aclDataBuffer *dataBuffer)

参数说明

参数名	输入/输出	说明
dataBuffer	输入	待销毁的aclDataBuffer的指针。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.20.3 aclGetDataBufferAddr

函数功能

获取aclDataBuffer类型中的数据的内存地址。

函数原型

void *aclGetDataBufferAddr(const **aclDataBuffer** *dataBuffer)

参数说明

参数名	输入/输出	说明
dataBuffer	输入	aclDataBuffer的地址。

返回值说明

无

8.15.20.4 aclGetDataBufferSize

函数功能

获取aclDataBuffer类型中数据的内存大小，单位Byte。

函数原型

size_t aclGetDataBufferSize(const **aclDataBuffer** *dataBuffer)

参数说明

参数名	输入/输出	说明
dataBuffer	输入	aclDataBuffer的地址。

返回值说明

aclDataBuffer类型中数据的内存大小。

8.15.21 aclmdlDataset

您可以通过以下接口获取aclmdlDataset类型的数据、并对aclmdlDataset类型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。

8.15.21.1 aclmdlCreateDataset

函数功能

创建aclmdlDataset类型的数据。

函数原型

aclmdlDataset *aclmdlCreateDataset()

参数说明

无

返回值说明

aclmdlDataset地址。

8.15.21.2 aclmdlDestroyDataset

函数功能

销毁aclmdlDataset类型的数据。

函数原型

aclError aclmdlDestroyDataset(const aclmdlDataset *dataset)

参数说明

参数名	输入/输出	说明
dataset	输入	待销毁的aclmdlDataset的指针。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.21.3 aclmdlAddDatasetBuffer

函数功能

向aclmdlDataset中增加aclDataBuffer。

函数原型

aclError aclmdlAddDatasetBuffer(aclmdlDataset *dataset, aclDataBuffer *dataBuffer)

参数说明

参数名	输入/输出	说明
dataset	输入&输出	待增加aclDataBuffer的aclmdlDataset地址。
dataBuffer	输入	待增加的aclDataBuffer地址

返回值说明

返回0表示成功，返回非0表示失败。

8.15.21.4 aclmdlGetDatasetNumBuffers

函数功能

获取aclmdlDataset中aclDataBuffer的个数。

函数原型

size_t aclmdlGetDatasetNumBuffers(const aclmdlDataset *dataset)

参数说明

参数名	输入/输出	说明
dataset	输入	aclmdlDataset地址。

返回值说明

aclDataBuffer的个数。

8.15.21.5 aclmdlGetDatasetBuffer

函数功能

获取aclmdlDataset中的第n个aclDataBuffer。

函数原型

aclDataBuffer* aclmdlGetDatasetBuffer(const aclmdlDataset *dataset, size_t index)

参数说明

参数名	输入/输出	说明
dataset	输入	aclmdlDataset地址。
index	输入	表明获取的是第几个aclDataBuffer。

返回值说明

- 获取成功，返回aclDataBuffer的地址。
- 获取失败返回空地址。

8.15.22 aclmdlDesc

该数据类型下的操作接口都是同步接口。

8.15.22.1 aclmdlCreateDesc

函数功能

创建aclmdlDesc类型的数据。

函数原型

aclmdlDesc* aclmdlCreateDesc()

参数说明

无

返回值说明

aclmdlDesc地址。

8.15.22.2 aclmdlDestroyDesc

函数功能

销毁aclmdlDesc类型的数据。

函数原型

aclError aclmdlDestroyDesc(aclmdlDesc *modelDesc)

参数说明

参数名	输入/输出	说明
modelDesc	输入	待销毁的aclmdlDesc的指针。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.22.3 aclmdlGetDesc

函数功能

根据模型ID获取该模型的**aclmdlDesc**类型数据。

函数原型

aclError aclmdlGetDesc(aclmdlDesc *modelDesc, uint32_t modelId)

参数说明

参数名	输入/输出	说明
modelDesc	输出	**aclmdlDesc**类型数据的指针。

参数名	输入/输出	说明
modelId	输入	模型ID。 调用**aclmdlLoadFromFile**接口/**aclmdlLoadFromMem**接口/**aclmdlLoadFromFileWithMem**接口/**aclmdlLoadFromMemWithMem**接口加载模型成功后，会返回模型ID。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.22.4 aclmdlGetNumInputs

函数功能

根据aclmdlDesc类型的数据，获取模型的输入个数。

函数原型

size_t aclmdlGetNumInputs(aclmdlDesc *modelDesc)

参数说明

参数名	输入/输出	说明
modelDesc	输入	**aclmdlDesc**类型数据的指针。

返回值说明

模型的输入个数。

8.15.22.5 aclmdlGetNumOutputs

函数功能

根据aclmdlDesc类型的数据，获取模型的输出个数。

函数原型

size_t aclmdlGetNumOutputs(aclmdlDesc *modelDesc)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型数据的指针。

返回值说明

模型的输出个数。

8.15.22.6 aclmdlGetInputSizeByIndex

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。

函数原型

size_t aclmdlGetInputSizeByIndex(aclmdlDesc *modelDesc, size_t index)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型数据的指针。
index	输入	指定获取第几个输入的大小，index值从0开始。

返回值说明

针对动态Batch、动态分辨率（宽高）的场景，返回最大档位对应的输入的大小；静态场景下，返回指定输入的大小。单位是Byte。

8.15.22.7 aclmdlGetOutputSizeByIndex

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。

函数原型

size_t aclmdlGetOutputSizeByIndex(aclmdlDesc *modelDesc, size_t index)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型数据的指针。
index	输入	指定获取第几个输出的大小，index值从0开始。

返回值说明

针对动态Batch、动态分辨率（宽高）的场景，返回最大档位对应的输出的大小；静态场景下，返回指定输出的大小。单位是Byte。

8.15.22.8 aclmdlGetInputDims

函数功能

根据模型描述获取模型的输入tensor的维度信息。

函数原型

aclError aclmdlGetInputDims(const aclmdlDesc *modelDesc, size_t index, aclmdlIODims *dims)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型数据的指针。
index	输入	指定获取第几个输入的Dims，index值从0开始。
dims	输出	获取输入维度信息。 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。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.22.9 aclmdlGetOutputDims

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。

固定Shape场景下，通过该接口获取指定的模型输出tensor的维度信息。

动态Shape（动态Batch或动态分辨率）场景下，通过该接口获取最大档的维度信息。

函数原型

aclError aclmdlGetOutputDims(const aclmdlDesc *modelDesc, size_t index, aclmdlIODims *dims)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型数据的指针。
index	输入	指定获取第几个输出的Dims，index值从0开始。
dims	输出	获取输出维度信息。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.22.10 aclmdlGetInputNameByIndex

函数功能

获取模型中指定输入的输入名称。

函数原型

const char *aclmdlGetInputNameByIndex(const aclmdlDesc *modelDesc, size_t index)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型数据的指针。
index	输入	指定获取第几个输入的名称，index值从0开始。

返回值说明

返回指定输入的输入名称。

8.15.22.11 aclmdlGetOutputNameByIndex

函数功能

获取模型中指定输出的输出名称以及输出边的下标。

函数原型

const char *aclmdlGetOutputNameByIndex(const aclmdlDesc *modelDesc, size_t index)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型数据的指针。
index	输入	指定获取第几个输出的名称及输出边下标，index值从0开始。

返回值说明

返回指定输出的输出名称以及输出边的下标。

8.15.22.12 aclmdlGetInputFormat

函数功能

获取模型中指定输入的Format。

函数原型

aclFormat aclmdlGetInputFormat(const aclmdlDesc *modelDesc, size_t index)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型数据的指针。
index	输入	指定获取第几个输入的Format，index值从0开始。

返回值说明

返回指定输入的Format。

8.15.22.13 aclmdlGetOutputFormat

函数功能

获取模型中指定输出的Format。

函数原型

aclFormat aclmdlGetOutputFormat(const aclmdlDesc *modelDesc, size_t index)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型数据的指针。
index	输入	指定获取第几个输出的Format，index值从0开始。

返回值说明

返回指定输出的Format。

8.15.22.14 aclmdlGetInputDataType

函数功能

获取模型中指定输入的数据类型。

函数原型

aclDataType aclmdlGetInputDataType(const aclmdlDesc *modelDesc, size_t index)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型数据的指针。
index	输入	指定获取第几个输入的数据类型，index值从0开始。

返回值说明

返回指定输入的数据类型。

8.15.22.15 aclmdlGetOutputDataType

函数功能

获取模型中指定输出的数据类型。

函数原型

aclDataType aclmdlGetOutputDataType(const aclmdlDesc *modelDesc, size_t index)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型数据的指针。
index	输入	指定获取第几个输出的数据类型，index值从0开始。

返回值说明

返回指定输出的数据类型。

8.15.22.16 aclmdlGetInputIndexByName

函数功能

根据模型中的输入名称获取对应输入的索引编号。

函数原型

aclError aclmdlGetInputIndexByName(const aclmdlDesc *modelDesc, const char *name, size_t *index)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型数据的指针。
name	输入	指定输入名称。
index	输出	获取对应输入的索引编号。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.22.17 aclmdlGetOutputIndexByName

函数功能

根据模型中的输出名称获取对应输出的索引编号。

函数原型

aclError aclmdlGetOutputIndexByName(const aclmdlDesc *modelDesc, const char *name, size_t *index)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型数据的指针。
name	输入	指定输出名称。
index	输出	获取对应输出的索引编号。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.22.18 aclmdlGetDynamicBatch

函数功能

获取模型支持的动态Batch信息。

函数原型

aclError aclmdlGetDynamicBatch(const aclmdlDesc *modelDesc, aclmdlBatch *batch)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型数据的指针。

batch	输出	const int ACL_MAX_BATCH_NUM = 128;
		typedef struct aclmdlBatch {
		size_t batchCount; /**模型中支持的batch分档数 */
		uint64_t batch[ACL_MAX_BATCH_NUM]; /**模型中支持的具体分档 */
		} aclmdlBatch;
		batchCount等于0时，表示不支持设置档位信息，以模型中的档位为准。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.22.19 aclmdlGetDynamicHW

函数功能

获取模型支持的动态宽高信息。

函数原型

aclError aclmdlGetDynamicHW(const aclmdlDesc *modelDesc, size_t index, aclmdlHW *hw)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型数据的指针。
index	输入	预留参数，当前未使用，固定设置为-1。

hw	输出	const int ACL_MAX_HW_NUM = 128;
		typedef struct aclmdlHW {
		size_t hwCount; /**模型中支持的宽高分档数 */
		uint64_t hw[ACL_MAX_HW_NUM][2]; /**模型中支持的具体分档，每组分档中，数组下标为0代表的是高，下标为1代表的是宽 */
		} aclmdlHW;
		hwCount等于0时，表示不支持设置档位信息，以模型中的档位为准。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.22.20 aclmdlGetCurOutputDims

函数功能

根据模型描述获取指定的模型输出tensor的实际维度信息。

动态Shape（动态Batch或动态分辨率）场景下，如果用户已调用aclmdlSetDynamicBatchSize设置Batch或调用aclmdlSetDynamicHWSize接口设置输入图片的宽高，则可通过该接口获取指定模型输出tensor的实际维度信息；如果用户未调用aclmdlSetDynamicBatchSize接口或aclmdlSetDynamicHWSize接口，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。

函数原型

aclError aclmdlGetCurOutputDims(const aclmdlDesc *modelDesc, size_t index, aclmdlIODims *dims)

参数说明

参数名	输入/输出	说明
modelDesc	输入	aclmdlDesc类型数据的指针。
index	输入	指定获取第几个输出的Dims，index值从0开始。

dims	输出	输出实际维度信息。
		const int ACL_MAX_DIM_CNT = 128;
		const int ACL_MAX_TENSOR_NAME_LEN = 128;
		typedef struct aclmdlIODims {
		char name[ACL_MAX_TENSOR_NAME_LEN]; /**< tensor name */
		size_t dimCount; /**Shape中的维度个数*/
		int64_t dims[ACL_MAX_DIM_CNT]; /**< 维度信息 */
		} aclmdlIODims;

返回值说明

返回0表示成功，返回非0表示失败。

8.15.22.21 aclmdlGetFirstAippInfo

函数功能

获取模型静态AIPP的配置信息。

静态AIPP支持的几种计算方式及其计算顺序如下，详细说明及参数解释，请参考《ATC工具使用指导》。

抠图->色域转换->减均值/归一化->padding

函数原型

aclError aclmdlGetFirstAippInfo(uint32_t modelId, size_t index, aclAippInfo *aippinfo)

参数说明

参数名	输入/输出	说明
modelId	输入	模型ID。 调用aclmdlLoadFromFile接口/aclmdlLoadFromMem接口/aclmdlLoadFromFileWithMem接口/aclmdlLoadFromMemWithMem接口加载模型成功后，会返回模型ID。
index	输入	指定模型的第几个输入。
aippinfo	输出	获取指定输入上静态AIPP的配置信息。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.23 aclTensorDesc

您可以通过以下接口对aclTensorDesc类型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。

8.15.23.1 aclCreateTensorDesc

函数功能

创建aclTensorDesc类型的数据。

函数原型

aclTensorDesc *aclCreateTensorDesc(aclDataType dataType,

int numDims,

const int64_t *dims,

aclFormat format)

参数说明

参数名	输入/输出	说明
dataType	输入	tensor描述的数据类型。

参数名	输入/输出	说明
numDims	输入	tensor描述shape的维度个数。 numDims存在以下约束： • numDims必须大于或等于0； • numDims>0时，numDims的值必须与dims数组的长度保持一致； • numDims=0时，系统不使用dims数组值，dims参数值无效。
dims	输入	tensor描述指定维度的大小。 dims是一个数组，数组中的每个元素表示tensor中每个维度的大小，如果数组中某个元素的值为0，则为空tensor。 如果用户需要使用空tensor，则在申请内存时，内存大小最小为1Byte。
format	输入	tensor描述的format。

返回值说明

aclTensorDesc地址。

8.15.23.2 aclDestroyTensorDesc

函数功能

销毁aclTensorDesc类型的数据。

函数原型

void aclDestroyTensorDesc(const aclTensorDesc *desc)

参数说明

参数名	输入/输出	说明
desc	输入	待销毁的aclTensorDesc的指针。

返回值说明

无

8.15.23.3 aclGetTensorDescType

函数功能

获取tensor描述中的数据类型。

函数原型

aclDataType aclGetTensorDescType(const aclTensorDesc *desc)

参数说明

参数名	输入/输出	说明
desc	输入	aclTensorDesc的指针。

返回值说明

返回指定tensor描述的数据类型。

8.15.23.4 aclGetTensorDescFormat

函数功能

获取tensor描述中的format。

函数原型

aclFormat aclGetTensorDescFormat(const aclTensorDesc *desc)

参数说明

参数名	输入/输出	说明
desc	输入	aclTensorDesc的指针。

返回值说明

返回指定tensor描述的format。

8.15.23.5 aclGetTensorDescSize

函数功能

获取tensor描述占用的空间大小。

函数原型

size_t aclGetTensorDescSize(const aclTensorDesc *desc)

参数说明

参数名	输入/输出	说明
desc	输入	aclTensorDesc的指针。

返回值说明

返回指定tensor描述占用的空间大小。

8.15.23.6 aclGetTensorDescElementCount

函数功能

获取tensor描述中的元素个数。

函数原型

size_t aclGetTensorDescElementCount(const aclTensorDesc *desc)

参数说明

参数名	输入/输出	说明
desc	输入	aclTensorDesc的指针。

返回值说明

返回指定tensor描述中的元素个数。

8.15.23.7 aclGetTensorDescNumDims

函数功能

获取tensor描述中shape的维度个数。

函数原型

size_t aclGetTensorDescNumDims(const aclTensorDesc *desc)

参数说明

参数名	输入/输出	说明
desc	输入	aclTensorDesc的指针。

返回值说明

返回tensor描述中shape的维度个数。

8.15.23.8 aclGetTensorDescDim

函数功能

获取tensor描述中指定维度的大小。

函数原型

int64_t aclGetTensorDescDim(const aclTensorDesc *desc, size_t index)

参数说明

参数名	输入/输出	说明
desc	输入	aclTensorDesc的指针。
index	输入	指定获取第几个维度的大小，index值从0开始。

返回值说明

返回指定tensor描述中指定维度的大小。

8.15.23.9 aclSetTensorDescName

函数功能

设置TensorDesc的名称。

函数原型

void aclSetTensorDescName(aclTensorDesc *desc, const char *name)

参数说明

参数名	输入/输出	说明
desc	输入	aclTensorDesc的指针。
name	输入	TensorDesc的名称。

返回值说明

无

8.15.23.10 aclGetTensorDescName

函数功能

获取TensorDesc的名称。

函数原型

const char *aclGetTensorDescName(aclTensorDesc *desc)

参数说明

参数名	输入/输出	说明
desc	输入	aclTensorDesc的指针。

返回值说明

返回TensorDesc的名称。

8.15.23.11 aclTransTensorDescFormat

函数功能

按指定dstFormat转换源aclTensorDesc中的format，生成新的目标aclTensorDesc，源aclTensorDesc中的format保持不变，当前不支持，预留接口。

函数原型

aclError aclTransTensorDescFormat(const aclTensorDesc *srcDesc, aclFormat dstFormat, aclTensorDesc **dstDesc)

参数说明

参数名	输入/输出	说明
srcDesc	输入	源aclTensorDesc数据指针。
dstFormat	输入	需要设置的目标format。
dstDesc	输出	“目标aclTensorDesc数据指针”的指针。

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.23.12 aclSetTensorStorageShape

函数功能

调用aclCreateTensorDesc接口创建tensor描述信息后，可通过aclSetTensorStorageShape接口设置tensor的实际dims信息，当前主要用于pytorch场景下。

函数原型

aclError aclSetTensorStorageShape(aclTensorDesc *desc, int numDims, const int64_t *dims)

参数说明

参数名	输入/输出	说明
desc	输入	需要设置的aclTensorDesc数据的指针。
numDims	输入	需要设置的dims维度数。
dims	输入	要设置的dims。

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.23.13 aclSetTensorStorageFormat

函数功能

调用aclCreateTensorDesc接口创建tensor描述信息后，可通过aclSetTensorStorageShape设置tensor的实际format信息，当前主要用于pytorch场景下。

函数原型

aclError aclSetTensorStorageFormat(aclTensorDesc *desc, aclFormat format)

参数说明

参数名	输入/输出	说明
desc	输入	需要设置的aclTensorDesc数据的指针。
format	输入	需要设置的format。

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.24 aclopAttr

该类型用于保存算子的属性，您可以通过以下接口设置aclopAttr类型的数据。该数据类型下的操作接口都是同步接口。

8.15.24.1 aclopCreateAttr

函数功能

创建aclopAttr类型的数据。

函数原型

aclopAttr *aclopCreateAttr()

参数说明

无

返回值说明

aclopAttr地址。

8.15.24.2 aclopDestroyAttr

函数功能

销毁aclopAttr类型的数据。

函数原型

void aclopDestroyAttr(const aclopAttr *attr)

参数说明

参数名	输入/输出	说明
attr	输入	待销毁的aclopAttr的指针。

返回值说明

无

8.15.24.3 aclopSetAttrBool

函数功能

设置bool类型标量的属性值。

函数原型

aclError aclopSetAttrBool(aclopAttr *attr, const char *attrName, uint8_t attrValue)

参数说明

参数名	输入/输出	说明
attr	输入	指向aclopAttr的指针。
attrName	输入	属性名。
attrValue	输入	属性值。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.24.4 aclopSetAttrInt

函数功能

设置int64_t类型标量的属性值。

函数原型

aclError aclopSetAttrInt(**aclopAttr** *attr, const char *attrName, int64_t attrValue);

参数说明

参数名	输入/输出	说明
attr	输入	指向aclopAttr的指针。
attrName	输入	属性名。
attrValue	输入	属性值。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.24.5 aclopSetAttrFloat

函数功能

设置float类型标量的属性值。

函数原型

aclError aclopSetAttrFloat(aclopAttr *attr, const char *attrName, float attrValue)

参数说明

参数名	输入/输出	说明
attr	输入	指向aclopAttr的指针。
attrName	输入	属性名。
attrValue	输入	属性值。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.24.6 aclopSetAttrString

函数功能

设置字符串类型标量的属性值。

函数原型

aclError aclopSetAttrString(**aclopAttr** *attr, const char *attrName, const char *attrValue)

参数说明

参数名	输入/输出	说明
attr	输入	指向aclopAttr的指针。
attrName	输入	属性名。
attrValue	输入	属性值。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.24.7 aclopSetAttrListBool

函数功能

设置bool类型列表的属性值。

函数原型

aclError aclopSetAttrListBool(aclopAttr *attr, const char *attrName, int numValues, const uint8_t *values)

参数说明

参数名	输入/输出	说明
attr	输入	指向aclopAttr的指针。
attrName	输入	属性名。
numValues	输入	属性值数目。
values	输入	属性值的数组地址。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.24.8 aclopSetAttrListInt

函数功能

设置int64_t类型列表的属性值。

函数原型

aclError aclopSetAttrListInt(aclopAttr *attr, const char *attrName, int numValues, const int64_t *values)

参数说明

参数名	输入/输出	说明
attr	输入	指向aclopAttr的指针。
attrName	输入	属性名。
numValues	输入	属性值数目。
values	输入	属性值的数组地址。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.24.9 aclopSetAttrListFloat

函数功能

设置float类型列表的属性值。

函数原型

aclError aclopSetAttrListFloat(aclopAttr *attr, const char *attrName, int numValues, const float *values)

参数说明

参数名	输入/输出	说明
attr	输入	指向aclopAttr的指针。
attrName	输入	属性名。
numValues	输入	属性值数目。
values	输入	属性值的数组地址。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.24.10 aclopSetAttrListString

函数功能

设置字符串类型列表的属性值。

函数原型

aclError aclopSetAttrListString(aclopAttr *attr, const char *attrName, int numValues, const char **values)

参数说明

参数名	输入/输出	说明
attr	输入	指向aclopAttr的指针。
attrName	输入	属性名。
numValues	输入	属性值数目。

参数名	输入/输出	说明
values	输入	“属性值的数组指针”的指针。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.24.11 aclopSetAttrListListInt

函数功能

设置int64_t类型列表的列表的属性值。

函数原型

aclError aclopSetAttrListListInt(aclopAttr *attr,

const char *attrName,

int numLists,

const int *numValues,

const int64_t *const values[]);

参数说明

参数名	输入/输出	说明
attr	输入	指向aclopAttr的指针。
attrName	输入	属性名。
numLists	输入	列表数目。
numValues	输入	列表中属性值数目。 numValues是一个数组，数组中的每个元素表示每个列表中的属性值数目。
values	输入	列表中属性值。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.25 acldvppChannelDesc

该类型用于描述图片数据处理的通道信息，您可以通过以下接口创建和销毁acldvppChannelDesc类型的数据。该数据类型下的操作接口都是同步接口。

8.15.25.1 acldvppCreateChannelDesc

函数功能

创建acldvppChannelDesc类型的数据，表示创建图片数据处理通道时的通道描述信息。该接口为同步接口。

函数原型

acldvppChannelDesc *acldvppCreateChannelDesc()

参数说明

无

返回值说明

- 返回acldvppChannelDesc类型，表示成功。
- 返回null，表示失败。

8.15.25.2 acldvppDestroyChannelDesc

函数功能

销毁acldvppChannelDesc类型的数据，只能销毁通过**acldvppCreateChannelDesc**接口创建的acldvppChannelDesc类型。该接口为同步接口。

必须在调用**acldvppDestroyChannel**接口销毁channel通道之后再调用acldvppDestroyChannelDesc接口释放acldvppChannelDesc类型的数据，否则报错。

函数原型

aclError acldvppDestroyChannelDesc(acldvppChannelDesc *channelDesc)

参数说明

参数名	输入/输出	说明
channelDesc	输入	指定需要销毁的channelDesc。

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.25.3 acldvppGetChannelDescChannelId

函数功能

根据通道描述信息获取通道ID。

约束说明

接口调用顺序：acldvppCreateChannelDesc --> acldvppCreateChannel --> acldvppGetChannelDescChannelId

函数原型

uint64_t acldvppGetChannelDescChannelId(const acldvppChannelDesc *channelDesc)

参数说明

参数名	输入/输出	说明
channelDesc	输入	指定通道描述信息。 调用**acldvppCreateChannelDesc**接口创建acldvppChannelDesc类型的数据。

返回值说明

返回通道ID。

8.15.25.4 acldvppSetChannelDescMode

函数功能

指定通道描述信息中的通道模式，明确图片数据处理通道用于实现哪种功能，目前支持**VPC**、**JPEGD**、**JPEGE**功能。

若不调用该接口，则系统默认3种模式的通道都创建，可能会占用资源，推荐用户根据实际功能指定通道模式。

当前版本不支持该接口。

约束说明

接口调用顺序：acldvppCreateChannelDesc --> acldvppSetChannelDescMode --> acldvppCreateChannel

函数原型

aclError acldvppSetChannelDescMode(acldvppChannelDesc *channelDesc, uint32_t mode)

参数说明

参数名	输入/输出	说明
channelDesc	输入	指定通道描述信息。 调用**acldvppCreateChannelDesc**接口创建acldvppChannelDesc类型的数据。

mode	输入	指定通道模式。
		enum acldvppChannelMode {
		DVPP_CHNMODE_VPC = 1,
		DVPP_CHNMODE_JPEGD = 2,
		DVPP_CHNMODE_JPEGE = 4
		};

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.26 acldvppPicDesc

该数据类型下的操作接口都是同步接口。

8.15.26.1 acldvppCreatePicDesc

函数功能

创建图片描述信息。

函数原型

acldvppPicDesc *acldvppCreatePicDesc()

参数说明

无

返回值说明

- 返回acldvppPicDesc类型，表示成功。
- 返回null，表示失败。

8.15.26.2 acldvppSetPicDesc 系列接口

函数功能

设置图片描述参数。

函数原型

aclError acldvppSetPicDescData(acldvppPicDesc *picDesc, void *dataDev);

aclError acldvppSetPicDescSize(acldvppPicDesc *picDesc, uint32_t size);

aclError acldvppSetPicDescFormat(acldvppPicDesc *picDesc, acldvppPixelFormat format);

aclError acldvppSetPicDescWidth(acldvppPicDesc *picDesc, uint32_t width);

aclError acldvppSetPicDescHeight(acldvppPicDesc *picDesc, uint32_t height);

aclError acldvppSetPicDescWidthStride(acldvppPicDesc *picDesc, uint32_t widthStride);

aclError acldvppSetPicDescHeightStride(acldvppPicDesc *picDesc, uint32_t heightStride);

aclError acldvppSetPicDescRetCode(acldvppPicDesc *picDesc,uint32_t retCode)

参数说明

参数名	输入/输出	说明
picDesc	输入	描述图片信息。
dataDev	输入	Device上图片数据内存，必须是使用acldvppMalloc接口申请的内存。
size	输入	内存大小，单位Byte。
format	输入	图片格式。
width	输入	原始图片宽。
height	输入	原始图片高。
widthStride	输入	对齐后的宽。
heightStride	输入	对齐后的高。
retCode	输入	返回码（0成功，非0失败）。

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.26.3 acldvppGetPicDesc 系列接口

函数功能

获取图片描述信息。

函数原型

void *acldvppGetPicDescData(const acldvppPicDesc *picDesc);

uint32_t acldvppGetPicDescSize(const acldvppPicDesc *picDesc);

acldvppPixelFormat acldvppGetPicDescFormat(const acldvppPicDesc *picDesc);

uint32_t acldvppGetPicDescWidth(const acldvppPicDesc *picDesc);

uint32_t acldvppGetPicDescHeight(const acldvppPicDesc *picDesc);

uint32_t acldvppGetPicDescWidthStride(const acldvppPicDesc *picDesc);

uint32_t acldvppGetPicDescHeightStride(const acldvppPicDesc *picDesc);

uint32_t acldvppGetPicDescRetCode(const acldvppPicDesc *picDesc);

参数说明

参数名	输入/输出	说明
picDesc	输入	描述图片信息。

返回值说明

参数名	说明
data	图片数据的内存地址。
size	图片数据的内存的大小。
format	图片格式。
width	原始图片宽。
height	原始图片高。
widthStride	对齐后的宽。
heightStride	对齐后的高。
retCode	返回码（0成功，非0失败）。

8.15.26.4 acldvppDestroyPicDesc

函数功能

销毁图片描述信息，只能销毁通过**acldvppCreatePicDesc**接口创建的图片描述信息。

函数原型

aclError acldvppDestroyPicDesc(acldvppPicDesc *picDesc)

参数说明

参数名	输入/输出	说明
picDesc	输入	指定待销毁的图片描述信息。

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.27 acldvppRoiConfig

该数据类型下的操作接口都是同步接口。

8.15.27.1 acldvppCreateRoiConfig

函数功能

创建用于描述某个区域位置的数据。

函数原型

```
acldvppRoiConfig *acldvppCreateRoiConfig(
uint32_t left,
uint32_t right,
uint32_t top,
uint32_t bottom)
```

参数说明

参数名	输入/输出	说明
left	输入	左偏移。 必须为偶数。 如果用做贴图区域，左偏移需要16对齐。
right	输入	右偏移。 必须为奇数。
top	输入	上偏移。 必须为偶数。
bottom	输入	下偏移。 必须为奇数。

返回值说明

- 返回acldvppRoiConfig类型，表示成功。
- 返回null，表示失败。

8.15.27.2 acldvppSetRoiConfig 系列接口

函数功能

设置某个区域的位置信息。

函数原型

aclError acldvppSetRoiConfig(acldvppRoiConfig *config, uint32_t left, uint32_t right, uint32_t top, uint32_t bottom);

aclError acldvppSetRoiConfigLeft (acldvppRoiConfig *config, uint32_t left);

aclError acldvppSetRoiConfigRight (acldvppRoiConfig *config, uint32_t right);

aclError acldvppSetRoiConfigTop (acldvppRoiConfig *config, uint32_t top);

aclError acldvppSetRoiConfigBottom(acldvppRoiConfig *config, uint32_t bottom)

参数说明

参数名	输入/输出	说明
config	输入	指定区域位置数据。 调用**acldvppCreateRoiConfig**接口创建**acldvppRoiConfig**类型的数据。
left	输入	左偏移。 必须为偶数。 如果用做贴图区域，左偏移需要16对齐。
right	输入	右偏移。 必须为奇数。
top	输入	上偏移。 必须为偶数。
bottom	输入	下偏移。 必须为奇数。

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.27.3 acldvppDestroyRoiConfig

函数功能

销毁通过**acldvppCreateRoiConfig**接口创建的数据。

函数原型

aclError acldvppDestroyRoiConfig(acldvppRoiConfig *roiConfig)

参数说明

参数名	输入/输出	说明
roiConfig	输入	指定待销毁的数据。

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.28 acldvppResizeConfig

该数据类型下的操作接口都是同步接口。

8.15.28.1 acldvppCreateResizeConfig

函数功能

创建图片缩放配置数据，默认缩放算法为“华为自研的最近邻插值算法”。

函数原型

acldvppResizeConfig *acldvppCreateResizeConfig()

参数说明

无

返回值说明

- 返回acldvppResizeConfig类型，表示成功。
- 返回null，表示失败。

8.15.28.2 acldvppSetResizeConfigInterpolation

函数功能

设置图片缩放算法。

函数原型

aclError acldvppSetResizeConfigInterpolation(acldvppResizeConfig *resizeConfig, uint32_t interpolation)

参数说明

参数名	输入/输出	说明
resizeConfig	输入	指定待设置的缩放配置数据。 调用acldvppCreateResizeConfig接口创建acldvppResizeConfig类型的数据。
interpolation	输入	指定缩放算子，取值范围： • 0：默认值，华为自研的最近邻插值算法 • 1：业界通用的Bilinear算法，当前不支持 • 2：业界通用的Nearest neighbor算法，当前不支持

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.28.3 acldvppGetResizeConfigInterpolation

函数功能

获取图片缩放算法。

函数原型

uint32 acldvppGetResizeConfigInterpolation(const acldvppResizeConfig *resizeConfig)

参数说明

参数名	输入/输出	说明
resizeConfig	输入	指定待获取的缩放配置数据。

返回值说明

- 0：默认值，华为自研的最近邻插值算法

8.15.28.4 acldvppDestroyResizeConfig

函数功能

销毁通过**acldvppCreateResizeConfig**接口创建的缩放配置数据。

函数原型

aclError acldvppDestroyResizeConfig(acldvppResizeConfig *resizeConfig)

参数说明

参数名	输入/输出	说明
resizeConfig	输入	指定待销毁的数据。

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.29 acldvppJpegeConfig

该数据类型下的操作接口都是同步接口。

8.15.29.1 acldvppCreateJpegeConfig

函数功能

创建图片编码配置数据。

函数原型

acldvppJpegeConfig *acldvppCreateJpegeConfig()

参数说明

无

返回值说明

- 返回acldvppResizeConfig类型，表示成功。
- 返回null，表示失败。

8.15.29.2 acldvppSetJpegeConfigLevel

函数功能

设置编码配置数据。

函数原型

aclError acldvppSetJpegeConfigLevel(acldvppJpegeConfig *jpegeConfig, uint32_t level)

参数说明

参数名	输入/输出	说明
jpegeConfig	输入&输出	指定编码配置数据。 调用**acldvppCreateJpegeConfig**接口创建**acldvppJpegeConfig**类型的数据。
level	输入	编码质量范围[0, 100]，其中level 0编码质量与level 100差不多，而在[1, 100]内数值越小输出图片质量越差。

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.29.3 acldvppGetJpegeConfigLevel

函数功能

获取编码配置数据。

函数原型

uint32_t acldvppGetJpegeConfigLevel(const acldvppJpegeConfig *jpegeConfig)

参数说明

参数名	输入/输出	说明
jpegeConfig	输入	指定编码配置数据。 调用**acldvppCreateJpegeConfig**接口创建**acldvppJpegeConfig**类型的数据，调用**acldvppSetJpegeConfigLevel**接口设置编码配置数据。

返回值说明

返回编码质量范围[0, 100]，其中level 0编码质量与level 100差不多，而在[1, 100]内数值越小输出图片质量越差。

8.15.29.4 acldvppDestroyJpegeConfig

函数功能

销毁通过**acldvppCreateJpegeConfig**接口创建的编码配置数据。

函数原型

aclError acldvppDestroyJpegeConfig(acldvppJpegeConfig *jpegeConfig)

参数说明

参数名	输入/输出	说明
jpegeConfig	输入	指定待销毁的数据。

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.30 aclvdecChannelDesc

该数据类型下的操作接口都是同步接口。

8.15.30.1 aclvdecCreateChannelDesc

函数功能

创建aclvdecChannelDesc类型的数据，表示创建视频解码处理通道时的通道描述信息。该接口为同步接口。

函数原型

aclvdecChannelDesc *aclvdecCreateChannelDesc()

参数说明

无

返回值说明

- 返回aclvdecChannelDesc类型，表示成功。
- 返回null，表示失败。

8.15.30.2 aclvdecSetChannelDesc 系列接口

函数功能

设置视频解码处理通道描述信息的属性。

函数原型

aclError aclvdecSetChannelDescChannelId(aclvdecChannelDesc *channelDesc, uint32_t channelId);

aclError aclvdecSetChannelDescThreadId(aclvdecChannelDesc *channelDesc, uint64_t threadId);

aclError aclvdecSetChannelDescCallback(aclvdecChannelDesc *channelDesc, aclvdecCallback callback);

aclError aclvdecSetChannelDescEnType(aclvdecChannelDesc *channelDesc, acldvppStreamFormat enType);

aclError aclvdecSetChannelDescOutPicFormat(aclvdecChannelDesc *channelDesc, acldvppPixelFormat outPicFormat);

aclError aclvdecSetChannelDescOutPicWidth(aclvdecChannelDesc *channelDesc, uint32_t outPicWidth);

aclError aclvdecSetChannelDescOutPicHeight(aclvdecChannelDesc *channelDesc, uint32_t outPicHeight);

aclError aclvdecSetChannelDescRefFrameNum(aclvdecChannelDesc *channelDesc, uint32_t refFrameNum);

aclError aclvdecSetChannelDescOutMode(aclvdecChannelDesc *channelDesc, uint32_t outMode)

参数说明

参数名	输入/输出	说明
channelDesc	输入	表示视频解码处理通道的描述信息。
channelId	输入	解码通道号，取值范围[0-31]。
threadId	输入	回调线程ID。
callback	输入	解码回调函数。 请参见**8.12.7.3 aclvdecCallback**。
enType	输入	视频编码协议H265-main level（0）、H264-baseline level（1）、H264-main level(2)、H264-high level（3）。
outPicFormat	输入	YUV图像存储格式，支持如下格式： • YUV420SP NV12 • YUV420SP NV21 如果不设置输出格式，默认使用YUV420SP NV12。
outPicWidth	输入	图片宽度。
outPicHeight	输入	图片高度。

参数名	输入/输出	说明
refFrameNum	输入	参考帧数量，取值范围[0, 16]。
outMode	输入	设置是否实时出帧（即发送一帧解码一帧，无需依赖后续帧的传入），取值范围如下： • 0：默认出帧模式，由于解码过程中存在缓存帧，无法实时输出，因此VDEC需要在收到码流中的多帧数据后，才开始输出解码结果。 • 1：快速出帧模式，VDEC获取码流中的一帧数据后，就开始实时输出解码结果。只支持简单参考关系的H264/H265标准码流（无长期参考帧，无B帧）。

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.30.3 aclvdecGetChannelDesc 系列接口

函数功能

获取视频解码处理通道的描述信息。

函数原型

uint32_t aclvdecGetChannelDescChannelId(const aclvdecChannelDesc *channelDesc);

uint32_t aclvdecGetChannelDescThreadId(const aclvdecChannelDesc *channelDesc);

aclvdecCallback aclvdecGetChannelDescCallback(const aclvdecChannelDesc *channelDesc);

acldvppStreamFormat aclvdecGetChannelDescEnType(const aclvdecChannelDesc *channelDesc);

acldvppPixelFormat aclvdecGetChannelDescOutPicFormat(const aclvdecChannelDesc *channelDesc);

uint32_t aclvdecGetChannelDescOutPicWidth(const aclvdecChannelDesc *channelDesc);

uint32_t aclvdecGetChannelDescOutPicHeight(const aclvdecChannelDesc *channelDesc);

uint32_t aclvdecGetChannelDescRefFrameNum(const aclvdecChannelDesc *channelDesc)

参数说明

参数名	输入/输出	说明
channelDesc	输入	表示视频解码数据处理通道的描述信息。

返回值说明

参数名	说明
channelId	通道号，取值范围[0-31]。
threadId	回调线程ID。
callback	回调函数。 请参见8.12.7.3 aclvdecCallback。
enType	视频编码协议H265-main level（0）、H264-baseline level（1）、H264-main level(2)、H264-high level（3）。
outPicFormat	YUV图像存储格式。
outPicWidth	图片宽度。
outPicHeight	图片高度。
refFrameNum	参考帧数量，取值范围[0, 16]。

8.15.30.4 aclvdecDestroyChannelDesc

函数功能

销毁aclvdecChannelDesc类型的数据，只能销毁通过aclvdecCreateChannelDesc接口创建的aclvdecChannelDesc类型。

函数原型

aclError aclvdecDestroyChannelDesc(aclvdecChannelDesc *channelDesc)

参数说明

参数名	输入/输出	说明
channelDesc	输入	指定需要销毁的channelDesc。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.31 acldvppStreamDesc

该数据类型下的操作接口都是同步接口。

8.15.31.1 acldvppCreateStreamDesc

函数功能

创建acldvppStreamDesc类型的数据，表示视频码流描述信息。

函数原型

acldvppStreamDesc *acldvppCreateStreamDesc()

参数说明

无

返回值说明

- 返回acldvppStreamDesc类型，表示成功。
- 返回null，表示失败。

8.15.31.2 acldvppSetStreamDesc 系列接口

函数功能

设置视频码流信息的属性。

函数原型

aclError acldvppSetStreamDescData(acldvppStreamDesc *streamDesc, void *dataDev);

aclError acldvppSetStreamDescSize(acldvppStreamDesc *streamDesc, uint32_t size);

aclError acldvppSetStreamDescFormat(acldvppStreamDesc *streamDesc, acldvppStreamFormat format);

aclError acldvppSetStreamDescTimestamp(acldvppStreamDesc *streamDesc, uint64_t timestamp);

aclError acldvppSetStreamDescRetCode(acldvppStreamDesc *streamDesc, uint32_t retCode);

aclError acldvppSetStreamDescEos(acldvppStreamDesc *streamDesc, uint8_t eos)

参数说明

参数名	输入/输出	说明
streamDesc	输入	表示要设置的视频码流信息。
dataDev	输入	Device上码流数据内存。
size	输入	内存大小，单位Byte。
format	输入	码流格式（h264/h265）。
timestamp	输入	时间戳。
retCode	输入	返回码（0成功，非0失败）。
eos	输入	是否为最后一帧数据（0否，1是）。

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.31.3 acldvppGetStreamDesc 系列接口

函数功能

获取视频码流信息的属性。

函数原型

void *acldvppGetStreamDescData(const acldvppStreamDesc *streamDesc);

uint32_t acldvppGetStreamDescSize(const acldvppStreamDesc *streamDesc);

acldvppStreamFormat acldvppGetStreamDescFormat(const acldvppStreamDesc *streamDesc);

uint64_t acldvppGetStreamDescTimestamp(const acldvppStreamDesc *streamDesc);

uint32_t acldvppGetStreamDescRetCode(const acldvppStreamDesc *streamDesc);

uint8_t acldvppGetStreamDescEos(const acldvppStreamDesc *streamDesc)

参数说明

参数名	输入/输出	说明
streamDesc	输入	表示要设置的视频码流信息。

返回值说明

参数名	说明
dataDev	Device上码流数据内存。
size	内存大小，单位Byte。
format	码流格式（h264/h265）。
timestamp	时间戳。
retCode	返回码（0成功，非0失败）。
eos	是否为最后一帧数据。

8.15.31.4 acldvppDestroyStreamDesc

函数功能

销毁acldvppStreamDesc类型的数据，只能销毁通过**acldvppCreateStreamDesc**接口创建的acldvppStreamDesc类型。

函数原型

aclError acldvppDestroyStreamDesc(acldvppStreamDesc *streamDesc)

参数说明

参数名	输入/输出	说明
streamDesc	输入	指定需要销毁的streamDesc。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.32 aclvdecFrameConfig

该数据类型下的操作接口都是同步接口。

8.15.32.1 aclvdecCreateFrameConfig

函数功能

创建aclvdecFrameConfig类型的数据，表示创建VDEC解码时的单帧编码配置参数。

函数原型

aclvdecFrameConfig *aclvdecCreateFrameConfig()

参数说明

无

返回值说明

- 返回aclvdecFrameConfig类型，表示成功。
- 返回null，表示失败。

8.15.32.2 aclvdecDestroyFrameConfig

函数功能

销毁aclvdecFrameConfig类型的数据，只能销毁通过**aclvdecCreateFrameConfig**接口创建的aclvdecFrameConfig类型。

函数原型

aclError aclvdecDestroyFrameConfig(aclvdecFrameConfig *vdecFrameConfig)

参数说明

参数名	输入/输出	说明
vdecFrameConfig	输入	指定需要销毁的vdecFrameConfig。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.33 acldvppBatchPicDesc

该数据类型下的操作接口都是同步接口。

8.15.33.1 acldvppCreateBatchPicDesc

函数功能

创建acldvppBatchPicDesc类型的数据，表示批量图片描述信息。

函数原型

acldvppBatchPicDesc *acldvppCreateBatchPicDesc(uint32_t batchSize)

参数说明

参数名	输入/输出	说明
batchSize	输入	图片数量。

返回值说明

- 返回acldvppBatchPicDesc类型，表示成功。
- 返回null，表示失败。

8.15.33.2 acldvppGetPicDesc

函数功能

获取指定图片的图片描述信息。

函数原型

acldvppPicDesc *acldvppGetPicDesc(acldvppBatchPicDesc *batchPicDesc, uint32_t index)

参数说明

参数名	输入/输出	说明
batchPicDesc	输入	指定批量图片描述信息。
index	输入	指定获取第几个图片的描述信息，index值从0开始。

返回值说明

返回指定图片的图片描述信息。

8.15.33.3 acldvppDestroyBatchPicDesc

函数功能

销毁acldvppBatchPicDesc类型的数据，只能销毁通过**acldvppCreateBatchPicDesc**接口创建的acldvppBatchPicDesc类型。

函数原型

aclError acldvppDestroyBatchPicDesc (acldvppBatchPicDesc *batchPicDesc)

参数说明

参数名	输入/输出	说明
batchPicDesc	输入	指定需要销毁的acldvppBatchPicDesc。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.34 aclvencChannelDesc

该数据类型下的操作接口都是同步接口。

8.15.34.1 aclvencCreateChannelDesc

函数功能

创建aclvencChannelDesc类型的数据，表示创建视频编码处理通道时的通道描述信息。

函数原型

aclvencChannelDesc *aclvencCreateChannelDesc()

参数说明

无

返回值说明

- 返回aclvencChannelDesc类型，表示成功。
- 返回null，表示失败。

8.15.34.2 aclvencSetChannelDesc 系列接口

函数功能

设置视频编码处理通道描述信息的属性。

函数原型

aclError aclvencSetChannelDescThreadId(aclvencChannelDesc *channelDesc, uint64_t threadId);

aclError aclvencSetChannelDescCallback(aclvencChannelDesc *channelDesc, aclvencCallback callback);

aclError aclvencSetChannelDescEnType(aclvencChannelDesc *channelDesc, acldvppStreamFormat enType);

aclError aclvencSetChannelDescPicFormat(aclvencChannelDesc *channelDesc, acldvppPixelFormat picFormat);

aclError aclvencSetChannelDescPicWidth(aclvencChannelDesc *channelDesc, uint32_t picWidth);

aclError aclvencSetChannelDescPicHeight(aclvencChannelDesc *channelDesc, uint32_t picHeight);

aclError aclvencSetChannelDescKeyFrameInterval(aclvencChannelDesc *channelDesc, uint32_t keyFrameInterval);

参数说明

参数名	输入/输出	说明
channelDesc	输入	表示视频编码处理通道的描述信息。
threadId	输入	回调线程ID。
callback	输入	编码回调函数。
enType	输入	视频编码协议H265-main level（0）、H264-baseline level（1）、H264-main level(2)、H264-high level（3）。
picFormat	输入	输入图像格式，支持如下格式： • PIXEL_FORMAT_YUV_SEMIPLANAR_420 • PIXEL_FORMAT_YVU_SEMIPLANAR_420
picWidth	输入	图片宽度。
picHeight	输入	图片高度。
keyFrameInterval	输入	关键帧间隔，不能为0。

返回值说明

返回0表示成功，返回其它值表示失败。

8.15.34.3 aclvencGetChannelDesc 系列接口

函数功能

获取视频编码处理通道的描述信息。

函数原型

uint32_t aclvencGetChannelDescChannelId(const aclvencChannelDesc *channelDesc);

uint64_t aclvencGetChannelDescThreadId(const aclvencChannelDesc *channelDesc);

aclvencCallback aclvencGetChannelDescCallback(const aclvencChannelDesc *channelDesc);

acldvppStreamFormat aclvencGetChannelDescEnType(const aclvencChannelDesc *channelDesc);

acldvppPixelFormat aclvencGetChannelDescPicFormat(const aclvencChannelDesc *channelDesc);

uint32_t aclvencGetChannelDescPicWidth(const aclvencChannelDesc *channelDesc);

uint32_t aclvencGetChannelDescPicHeight(const aclvencChannelDesc *channelDesc);

uint32_t aclvencGetChannelDescKeyFrameInterval(const aclvencChannelDesc *channelDesc)

参数说明

参数名	输入/输出	说明
channelDesc	输入	表示视频编码处理通道的描述信息。

返回值说明

参数名	说明
channelId	通道号，默认为0。
threadId	回调线程ID。
callback	编码回调函数。
enType	视频编码协议H265-main level（0）、H264-baseline level（1）、H264-main level(2)、H264-high level（3）。
picFormat	输入图像格式。
picWidth	图片宽度。
picHeight	图片高度。
keyFrameInterval	关键帧间隔，不能为0。

8.15.34.4 aclvencDestroyChannelDesc

函数功能

销毁aclvencChannelDesc类型的数据，只能销毁通过aclvencCreateChannelDesc接口创建的aclvencChannelDesc类型。

函数原型

aclError aclvencDestroyChannelDesc(aclvencChannelDesc *channelDesc)

参数说明

参数名	输入/输出	说明
channelDesc	输入	指定需要销毁的channelDesc。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.35 aclvencFrameConfig

该数据类型下的操作接口都是同步接口。

8.15.35.1 aclvencCreateFrameConfig

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。

函数原型

aclvencFrameConfig *aclvencCreateFrameConfig()

参数说明

无

返回值说明

- 返回aclvencFrameConfig类型，表示成功。
- 返回null，表示失败。

8.15.35.2 aclvencSetFrameConfig 系列接口

函数功能

设置单帧编码配置参数。

函数原型

aclError aclvencSetFrameConfigForceIFrame(aclvencFrameConfig *config, uint8_t forceIFrame);

aclError aclvencSetFrameConfigEos(aclvencFrameConfig *config, uint8_t eos)

参数说明

参数名	输入/输出	说明
config	输入	表示单帧编码配置数据。
forceIFrame	输入	是否强制重新开始I帧间隔： ● 0：不强制 ● 1：强制重新开始I帧
eos	输入	是否为结束帧： ● 0：不是 ● 1：是结束帧

返回值说明

返回0表示成功，返回非0表示失败。

8.15.35.3 aclvencGetFrameConfig 系列接口

函数功能

获取单帧编码配置参数。

函数原型

uint8_t aclvencGetFrameConfigForceIFrame(const aclvencFrameConfig *config);

uint8_t aclvencGetFrameConfigEos(const aclvencFrameConfig *config)

参数说明

参数名	输入/输出	说明
config	输入	表示单帧编码配置数据。

返回值说明

参数名	说明
forceIFrame	是否强制重新开始I帧间隔： ● 0：不强制 ● 1：强制重新开始I帧

参数名	说明
eos	是否为结束帧： • 0：不是 • 1：是结束帧

8.15.35.4 aclvencDestroyFrameConfig

函数功能

销毁aclvencFrameConfig类型的数据，只能销毁通过**aclvencCreateFrameConfig**接口创建的aclvencFrameConfig类型。

函数原型

aclError aclvencDestroyFrameConfig(aclvencFrameConfig *config)

参数说明

参数名	输入/输出	说明
config	输入	指定需要销毁的config。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.36 aclfvRepoRange

该章节下的接口，昇腾310 AI处理器不支持。

8.15.36.1 aclfvCreateRepoRange

函数功能

创建aclfvRepoRange类型的数据，表示创建特征库范围的参数。该接口为同步接口。

函数原型

aclfvRepoRange *aclfvCreateRepoRange(uint32_t id0Min, uint32_t id0Max, uint32_t id1Min, uint32_t id1Max)

参数说明

参数名	输入/输出	说明
id0Min	输入	id0起始值，取值范围：[0,1023]，需小于等于id0Max。
id0Max	输入	id0最大值，取值范围：[0,1023]。
id1Min	输入	id1起始值，取值范围：[0,1023]，需小于等于id1Max
id1Max	输入	id1最大值，取值范围：[0,1023]。

返回值说明

- 返回aclfvRepoRange类型，表示成功。
- 返回null，表示失败。

8.15.36.2 aclfvDestroyRepoRange

函数功能

销毁aclfvRepoRange类型的数据，只能销毁通过**aclfvCreateRepoRange**接口创建的aclfvRepoRange类型。该接口为同步接口。

函数原型

aclError aclfvDestroyRepoRange(aclfvRepoRange *repoRange)

参数说明

参数名	输入/输出	说明
repoRange	输入	指定需要销毁的repoRange。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.37 aclfvFeatureInfo

该章节下的接口，昇腾310 AI处理器不支持。

8.15.37.1 aclfvCreateFeatureInfo

函数功能

创建aclfvFeatureInfo类型的数据，表示创建特征描述信息。该接口为同步接口。

函数原型

aclfvFeatureInfo *aclfvCreateFeatureInfo(uint32_t id0, uint32_t id1, uint32_t offset, uint32_t featureLen,

uint32_t featureCount, uint8_t *featureData, uint32_t featureDataLen)

参数说明

参数名	输入/输出	说明
id0	输入	一级库id，取值范围0-1023，N:M场景默认为0。 通过id0、id1共同确定一个库。
id1	输入	二级库id，取值范围0-1023，N:M场景默认为0。 通过id0、id1共同确定一个库。
offset	输入	添加的首个特征在库中的偏移，N:M场景默认为0。
featureLen	输入	单个特征长度，当前固定为36Byte，系统内部会校验。
featureCount	输入	特征数量，1:N最大100万，N:M最大1000万。
featureData	输入	特征值指针，根据特征长度连续存储。
featureDataLen	输入	featureData指针申请的内存长度，用于校验。

返回值说明

- 返回aclfvFeatureInfo类型，表示成功。
- 返回null，表示失败。

8.15.37.2 aclfvDestroyFeatureInfo

函数功能

销毁aclfvFeatureInfo类型的数据，只能销毁通过**aclfvCreateFeatureInfo**接口创建的aclfvFeatureInfo类型。该接口为同步接口。

函数原型

aclError aclfvDestroyFeatureInfo(aclfvFeatureInfo *featureInfo)

参数说明

参数名	输入/输出	说明
featureInfo	输入	指定需要销毁的featureInfo。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.38 aclfvQueryTable

该章节下的接口，昇腾310 AI处理器不支持。

8.15.38.1 aclfvCreateQueryTable

函数功能

创建aclfvQueryTable类型的数据，表示创建检索输入表信息。该接口为同步接口。

函数原型

aclfvQueryTable *aclfvCreateQueryTable(uint32_t queryCnt, uint32_t tableLen, uint8_t *tableData, uint32_t tableDataLen)

参数说明

参数名	输入/输出	说明
queryCnt	输入	检索请求数量，1:N场景为1，N:M场景最大1024。
tableLen	输入	单个表长度，长度固定为32KB，系统内部会校验。
tableData	输入	特征值列表，根据特征长度连续存储。
tableDataLen	输入	tableData指针申请的内存长度，用于校验。

返回值说明

- 返回aclfvQueryTable类型，表示成功。
- 返回null，表示失败。

8.15.38.2 aclfvDestroyQueryTable

函数功能

销毁aclfvQueryTable类型的数据，只能销毁通过**aclfvCreateQueryTable**接口创建的aclfvQueryTable类型。该接口为同步接口。

函数原型

aclError aclfvDestroyQueryTable(aclfvQueryTable *queryTable)

参数说明

参数名	输入/输出	说明
queryTable	输入	指定需要销毁的queryTable。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.39 aclfvSearchInput

该章节下的接口，昇腾310 AI处理器不支持。

8.15.39.1 aclfvCreateSearchInput

函数功能

创建aclfvSearchInput类型的数据，表示创建检索任务输入信息。该接口为同步接口。

函数原型

aclfvSearchInput *aclfvCreateSearchInput(aclfvQueryTable *queryTable, aclfvRepoRange *repoRange, uint32_t topk)

参数说明

参数名	输入/输出	说明
queryTable	输入	检索的输入表信息。
repoRange	输入	检索的特征库范围。
topk	输入	单个检索请求需要返回的结果数量。

返回值说明

- 返回aclfvSearchInput类型，表示成功。
- 返回null，表示失败。

8.15.39.2 aclfvDestroySearchInput

函数功能

销毁aclfvSearchInput类型的数据，只能销毁通过**aclfvCreateSearchInput**接口创建的aclfvSearchInput类型。该接口为同步接口。

函数原型

aclError aclfvDestroySearchInput(aclfvSearchInput *searchInput)

参数说明

参数名	输入/输出	说明
searchInput	输入	指定需要销毁的searchInput。

返回值说明

返回0表示成功，返回非0表示失败。

8.15.40 aclfvSearchResult

该章节下的接口，昇腾310 AI处理器不支持。

8.15.40.1 aclfvCreateSearchResult

函数功能

创建aclfvSearchResult类型的数据，表示创建检索结果信息。该接口为同步接口。

函数原型

aclfvSearchResult *aclfvCreateSearchResult(uint32_t queryCnt, uint32_t *resultNum, uint32_t resultNumDataLen,

uint32_t *id0, uint32_t *id1, uint32_t *resultOffset, float *resultDistance, uint32_t dataLen)

参数说明

参数名	输入/输出	说明
queryCnt	输入	检索请求个数，1:N场景为1。

参数名	输入/输出	说明
resultNum	输入	每个检索请求的结果个数
resultNumDataLen	输入	resultNum的内存总长度。
id0	输入	一级库id，总个数为topK*queryCnt。
id1	输入	二级库id，总个数为topK*queryCnt。
resultOffset	输入	每个检索请求的检索结果对应底库的偏移，总个数为topK*queryCnt。
resultDistance	输入	每个检索结果与检索请求间的距离，总个数为topK*queryCnt。
dataLen	输入	申请的内存大小，计算公式为：topK*queryCnt*4Byte。

返回值说明

- 返回aclfvSearchResult类型，表示成功。
- 返回null，表示失败。

8.15.40.2 aclfvDestroySearchResult

函数功能

销毁aclfvSearchResult类型的数据，只能销毁通过**aclfvCreateSearchResult**接口创建的aclfvSearchResult类型。该接口为同步接口。

函数原型

aclError aclfvDestroySearchResult(aclfvSearchResult *searchResult)

参数说明

参数名	输入/输出	说明
searchResult	输入	指定需要销毁的searchResult。

返回值说明

返回0表示成功，返回非0表示失败。

9 ACL 样例使用指导

9.1 准备环境（Ascend RC）

- 您需要部署开发环境，请参见《Atlas 200 DK 使用指南》中的**安装开发环境**。
- 您需要部署运行环境，请参见《Atlas 200 DK 使用指南》中的**安装运行环境**。

须知

- 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root用户安装Driver组件，Driver组件的安装路径示例为/usr/local/Ascend，在该路径下，包括“driver”、“add-ons”、“develop”目录。
  - 以非root用户（以HwHiAiUser用户为例）安装ACLlib组件minirc形态安装包，安装路径示例为/home/HwHiAiUser/Ascend/ascend-toolkit/latest/arm64-linux，在该路径下，包括“acllib”目录。
  - 以非root用户（以HwHiAiUser用户为例）安装Toolkit组件，安装路径示例为/home/HwHiAiUser/Ascend/ascend-toolkit/latest，在该路径下，包括“toolkit”目录。
- 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。

9.2 样例使用须知

您可以从开发环境上的“ACLlib组件的安装目录/acllib/sample”目录下获取各样例的代码。

当前样例应用的运行模式如下表所示。

Sample名称	基本功能	在Ascend RC上运行应用
acl_execute_op目录下的**acl_execute_gemm**样例	实现矩阵-矩阵乘运算	√
acl_execute_model目录下的**acl_dvpp_resnet50**样例（调用ACL封装DVPP功能的接口）	基于Caffe ResNet-50网络实现图片分类（包含图片解码+缩放）	√
acl_execute_model目录下的**acl_vpc_jpege_resnet50**样例（调用ACL封装DVPP功能的接口）	基于Caffe ResNet-50网络实现图片分类（包含图片解码+抠图+缩放+编码）	√
acl_execute_model目录下的**acl_vdec_resnet50**样例（调用ACL封装DVPP功能的接口）	基于Caffe ResNet-50网络实现图片分类（包含视频解码）	√
acl_execute_model目录下的**acl_resnet50**样例	基于Caffe ResNet-50网络实现图片分类（同步推理，不含数据预处理）	√

Sample名称	基本功能	在Ascend RC上运行应用
acl_execute_model目录下的acl_resnet50_async样例	基于Caffe ResNet-50网络实现图片分类（异步推理，不含数据预处理）	√

为了提升开发者代码的开发效率，针对昇腾AI处理器的软件组件和推理业务开源了一些代码样例和工具，如表9-1所示。

表 9-1 代码样例介绍

代码样例名称	功能用途	获取链接
API Samples	基于AscendCL架构开发的一系列样例代码，包含简单的编译演示样例。	获取链接
静态目标识别	基于AscendCL架构开发的静态目标识别Demo。	

9.3 实现矩阵-矩阵乘运算

9.3.1 样例介绍

获取样例

您可以从开发环境上的“ACLlib组件的安装目录/acllib/sample/acl_execute_op”目录下获取样例acl_execute_gemm的代码。

功能描述

该样例主要实现矩阵-矩阵相乘的运算：C = αAB + βC，A、B、C都是16*16的矩阵，即：m=16，n=16，k=16。矩阵乘的结果是一个16*16的矩阵。

图 9-1 Sample 示例

$$\text{矩阵 A}=\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1k} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2k} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3k} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mk} \end{pmatrix}, B=\begin{pmatrix} b_{11} & b_{12} & b_{13} & \cdots & b_{1n} \\ b_{21} & b_{22} & b_{23} & \cdots & b_{2n} \\ b_{31} & b_{32} & b_{33} & \cdots & b_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ b_{k1} & b_{k2} & b_{k3} & \cdots & b_{kn} \end{pmatrix}, C=\begin{pmatrix} C_{11} & C_{12} & C_{13} & \cdots & C_{1n} \\ C_{21} & C_{22} & C_{23} & \cdots & C_{2n} \\ C_{31} & C_{32} & C_{33} & \cdots & C_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ C_{m1} & C_{m2} & C_{m3} & \cdots & C_{mn} \end{pmatrix}$$

原理介绍

在该样例中，涉及的关键功能点，如下表所示。API接口的详细介绍请参见《 应用软件开发指南 》中的“AscendCL API参考”。

表 9-2 关键功能点

初始化	● 调用aclInit接口初始化ACL配置。 ● 调用aclFinalize接口实现ACL去初始化。
Device管理	● 调用aclrtSetDevice接口指定用于运算的Device。 ● 调用aclrtGetRunMode接口获取昇腾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aclrtResetDevice接口复位当前运算的Device，回收Device上的资源。
Stream管理	● 调用aclrtCreateStream接口创建Stream。 ● 调用aclrtDestroyStream接口销毁Stream。 ● 调用aclrtSynchronizeStream接口阻塞程序运行，直到指定stream中的所有任务都完成。
内存管理	● 调用aclrtMallocHost接口申请Host上内存。 ● 调用aclrtFreeHost释放Host上的内存。 ● 调用aclrtMalloc接口申请Device上的内存。 ● 调用aclrtFree接口释放Device上的内存。
数据传输	如果在开发者板上运行应用，则无需进行数据传输。
单算子调用	● 调用aclblasGemmEx接口实现矩阵-矩阵相乘的运算，由用户指定矩阵中元素的数据类型。在aclblasGemmEx接口内部封装了系统内置的矩阵乘算子GEMM。 ● 使用ATC（Ascend Tensor Compiler）工具将内置的矩阵乘算子GEMM的算子描述信息（包括输入输出Tensor描述、算子属性等）编译成适配昇腾AI处理器的离线模型（*.om文件），用于验证矩阵乘算子GEMM的运行结果。

目录结构

acl_execute_gemm样例代码结构如下所示。

```
├─── inc
│    ├─── common.h                  //定义公共函数（例如：文件读取函数）的头文件
│    ├─── gemm_runner.h             //定义矩阵乘运算相关函数的头文件

├─── run
│    ├─── out
│    │    ├───test_data
│    │    │    ├─── config
│    │    │    │    ├─── acl.json          //系统初始化的配置文件
│    │    │    │    ├─── gemm.json         //矩阵乘算子的算子描述信息
│    │    │    ├─── data
│    │    │    │    ├─── generate_data.py   //用于生成矩阵A、矩阵B的数据
```



```
├─── src
│    ├─── CMakeLists.txt          //编译脚本
│    ├─── common.cpp               //公共函数（例如：文件读取函数）的实现文件
│    ├─── gemm_main.cpp            //主函数的实现文件
│    ├─── gemm_runner.cpp          //执行矩阵乘运算相关函数的实现文件
```

9.3.2 Ascend RC，编译及运行应用

步骤1 模型转换。

1. 以HwHiAiUser（运行用户）登录开发环境。
2. 参考《**ATC工具使用指导**》中的使用ATC工具转换模型，执行“准备动作”，包括获取工具、设置环境变量。
3. 将矩阵乘算子的算子描述信息（*.json文件）编译成适配昇腾AI处理器的离线模型（*.om文件），运行矩阵乘算子时使用。

   切换到acl_execute_gemm目录，执行如下命令：

   ```
   atc --singleop=run/out/test_data/config/gemm.json --soc_version=Ascend310 --output=run/out/op_models
   ```

   --output参数：生成的om文件必须放在“run/out/op_models”目录下。

   关于各参数的详细解释，请参见《**ATC工具使用指导**》中的“约束及参数说明”。

步骤2 编译代码。

1. 以HwHiAiUser（运行用户）登录开发环境。
2. 切换到“acl_execute_gemm/run/out/test_data/data”目录下，执行generate_data.py脚本，在“run/out/test_data/data”目录下生成矩阵A的数据文件matrix_a.bin、矩阵B的数据文件matrix_b.bin、矩阵C的数据文件matrix_c.bin。

   ```
   python3.7.5 generate_data.py
   ```

   说明

   “run/out/test_data/data”目录下生成的output.bin文件中的数据是generate_data.py脚本中直接通过公式计算出来的矩阵乘结果，不作为该样例的输入数据。

3. 设置环境变量，编译脚本src/CMakeLists.txt通过环境变量所设置的头文件、库文件的路径来编译代码。

   编译脚本CMakeLists.txt中已默认设置编译依赖的头文件路径（/usr/local/Ascend/ascend-toolkit/latest/acllib/include/）、库文件路径（/usr/local/Ascend/ascend-toolkit/latest/acllib/lib64/stub/）。

   如果未设置${DDK_PATH}、${NPU_HOST_LIB}环境变量，则以编译脚本CMakeLists.txt中的路径为准；如果已设置${DDK_PATH}、${NPU_HOST_LIB}环境变量，则编译脚本CMakeLists.txt优先以环境变量为准，按环境变量指向的路径查找编译依赖的头文件和库文件。

   如下为设置环境变量的示例，请将*/home/HwHiAiUser/Ascend/ascend-toolkit/latest*替换为Ascend RC形态的ACLlib安装包的实际安装路径。

   ```
   export DDK_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/arm64-linux
   export NPU_HOST_LIB=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/arm64-linux/acllib/lib64/stub
   ```


说明

使用“*/home/HwHiAiUser/Ascend/ascend-toolkit/latest*/arm64-linux/acllib/lib64/stub”目录下的*.so库，是为了编译基于ACL接口的代码逻辑时，不依赖其它组件（例如Driver）的任何*.so库。

编译通过后，在开发者板上运行应用时，通过配置环境变量，应用会链接到开发者板上“/home/HwHiAiUser/Ascend/acllib/lib64”目录下的*.so库，运行时会自动链接到依赖其它组件的*.so库。

4. 切换到acl_execute_gemm目录，创建目录用于存放编译文件，例如，本文中，创建的目录为“build/intermediates/minirc”。
```
mkdir -p build/intermediates/minirc
```
5. 切换到“build/intermediates/minirc”目录，执行**cmake**生成编译文件。

   “../../../src”表示CMakeLists.txt文件所在的目录，请根据实际目录层级修改。
```
cd build/intermediates/minirc
cmake ../../../src -DCMAKE_CXX_COMPILER=aarch64-linux-gnu-g++ -DCMAKE_SKIP_RPATH=TRUE
```
6. 执行**make**命令，生成的可执行文件execute_gemm_op在“acl_execute_gemm/run/out”目录下。
```
make
```

步骤3 运行应用。

1. 以HwHiAiUser（运行用户）将开发环境的“acl_execute_gemm/run/out”目录下的所有文件上传到开发者板的同一个目录，例如“/home/HwHiAiUser/acl_execute_gemm”。
2. 以HwHiAiUser用户登录开发者板。
3. 设置环境变量。

   如下为设置环境变量的示例，请根据实际安装情况替换路径。
```
export LD_LIBRARY_PATH=/home/HwHiAiUser/Ascend/acllib/lib64
```
4. 切换到可执行文件execute_gemm_op所在的目录，例如“/home/HwHiAiUser/acl_execute_gemm/out”，给该目录下的execute_gemm_op文件加执行权限。
```
chmod +x execute_gemm_op
```
5. 切换到可执行文件execute_gemm_op所在的目录，例如“/home/HwHiAiUser/acl_execute_gemm/out”，运行可执行文件。
```
./execute_gemm_op
```
   执行成功后，会直接在终端窗口显示矩阵A的数据、矩阵B的数据以及矩阵乘的结果，同时在“result_files”目录下，生成存放矩阵乘结果的matrix_c.bin文件。

----结束

9.4 基于 Caffe ResNet-50 网络实现图片分类（包含图片解码+缩放）

9.4.1 样例介绍

获取样例

您可以从开发环境上的“ACLlib组件的安装目录/acllib/sample/acl_execute_model”目录下获取样例acl_dvpp_resnet50的代码。

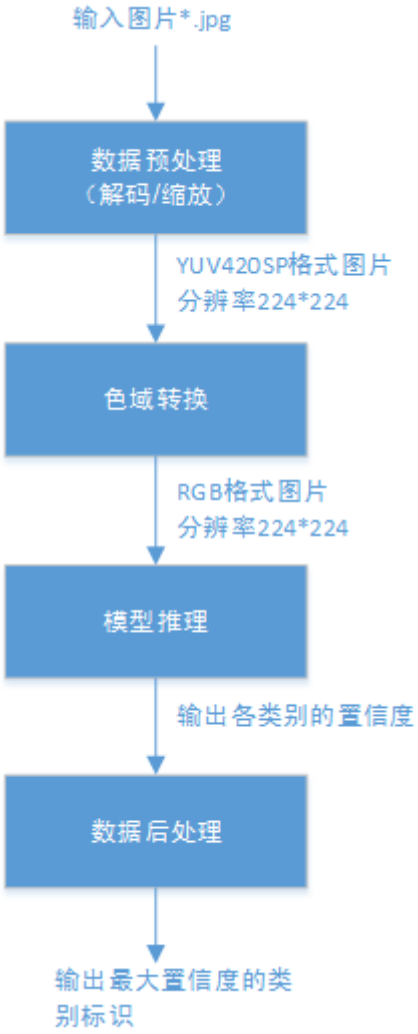
功能描述

该样例主要是基于Caffe ResNet-50网络（单输入、单batch）实现图片分类的功能。

将Caffe ResNet-50网络的模型文件转换为适配昇腾AI处理器的离线模型（*.om文件），在样例中，加载该om文件，对2张*.jpg图片进行解码、缩放、推理，分别得到推理结果后，再对推理结果进行处理，输出最大置信度的类别标识。

转换模型时，需配置色域转换参数，用于将YUV420SP格式的图片转换为RGB格式的图片，才能符合模型的输入要求。

图 9-2 Sample 示例



原理介绍

在该样例中，涉及的关键功能点，如下表所示。API接口的详细介绍请参见《应用软件开发指南》中的“AscendCL API参考”。

初始化	• 调用aclInit接口初始化ACL配置。 • 调用aclFinalize接口实现ACL去初始化。

Device管理	● 调用aclrtSetDevice接口指定用于运算的Device。 ● 调用aclrtGetRunMode接口获取昇腾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aclrtResetDevice接口复位当前运算的Device，回收Device上的资源。
Context管理	● 调用aclrtCreateContext接口创建Context。 ● 调用aclrtDestroyContext接口销毁Context。
Stream管理	● 调用aclrtCreateStream接口创建Stream。 ● 调用aclrtDestroyStream接口销毁Stream。 ● 调用aclrtSynchronizeStream接口阻塞程序运行，直到指定stream中的所有任务都完成。
内存管理	● 调用aclrtMallocHost接口申请Host上内存。 ● 调用aclrtFreeHost释放Host上的内存。 ● 调用aclrtMalloc接口申请Device上的内存。 ● 调用aclrtFree接口释放Device上的内存。 执行数据预处理时，若需要申请Device上的内存存放输入或输出数据，需调用acldvppMalloc申请内存、调用acldvppFree接口释放内存。
数据传输	如果在开发者板上运行应用，则无需进行数据传输。
数据预处理	● 调用acldvppJpegDecodeAsync接口将*.jpg图片解码成YUV420SP格式图片。 ● 调用acldvppVpcResizeAsync接口将YUV420SP格式图片缩小成分辨率为224*224的图片。
模型推理	● 调用aclmdlLoadFromFileWithMem接口从*.om文件加载模型。 ● 调用aclmdlExecute接口执行模型推理。 推理前，通过*.om文件中的色域转换参数将YUV420SP格式的图片转换为RGB格式的图片。 ● 调用aclmdlUnload接口卸载模型。
数据后处理（单算子调用）	处理模型推理的结果，通过调用算子Cast将推理结果的数据类型从float32转成float16，再调用ArgMaxD算子从推理结果中查找最大置信度的类别标识。 通过aclopExecute接口加载与执行算子。

目录结构

acl_dvpp_resnet50样例代码结构如下所示。

```
├acl_dvpp_resnet50
├─── caffe_model
|    ├─── aipp.cfg        //带色域转换参数的配置文件，模型转换时使用
|    ├─── resnet50.prototxt       //resnet50网络的模型文件

├─── data
```

```
|    ├─── dog1_1024_683.jpg           //测试数据
|    ├─── dog2_1024_683.jpg           //测试数据

├─── inc
|    ├─── dvpp_process.h             //声明数据预处理相关函数的头文件
|    ├─── model_process.h            //声明模型处理相关函数的头文件
|    ├─── sample_process.h           //声明资源初始化/销毁相关函数的头文件
|    ├─── singleOp_process.h         //声明单算子执行相关函数的头文件
|    ├─── utils.h                    //声明公共函数（例如：文件读取函数）的头文件

├─── out
|    ├─── op_models
|    ├─── ├───op_list.json           //Cast算子和ArgMaxD算子的算子描述信息

├─── src
|    ├─── acl.json        //系统初始化的配置文件
|    ├─── CMakeLists.txt        //编译脚本
|    ├─── dvpp_process.cpp      //数据预处理相关函数的实现文件
|    ├─── main.cpp             //主函数，图片分类功能的实现文件
|    ├─── model_process.cpp     //模型处理相关函数的实现文件
|    ├─── sample_process.cpp    //资源初始化/销毁相关函数的实现文件
|    ├─── singleOp_process.cpp  //单算子执行相关函数的实现文件
|    ├─── utils.cpp             //公共函数（例如：文件读取函数）的实现文件

├─── .project    //工程信息文件，包含工程类型、工程描述、运行目标设备类型等
├─── CMakeLists.txt    //编译脚本，调用src目录下的CMakeLists文件
```

9.4.2 Ascend RC，编译及运行应用

步骤1 模型转换。

1. 以HwHiAiUser（运行用户）登录开发环境。
2. 参考《**ATC工具使用指导**》中的使用ATC工具转换模型，执行“准备动作”，包括获取工具、设置环境变量。
3. 准备数据。

   从以下链接获取ResNet-50网络的权重文件（*.caffemodel），并以HwHiAiUser（运行用户）将获取的文件上传至开发环境的“acl_dvpp_resnet50/caffe_model”目录下。

   - 从gitee上获取：单击**Link**，查看README.md，查找获取原始模型的链接。
   - 从GitHub上获取：单击**Link**，查看README.md，查找获取原始模型的链接。
4. 将ResNet-50网络转换为适配昇腾AI处理器的离线模型（*.om文件），转换模型时，需配置色域转换参数，用于将YUV420SP格式的图片转换为RGB格式的图片。

   切换到“acl_dvpp_resnet50”目录，执行如下命令：

   ```
   atc --model=caffe_model/resnet50.prototxt --weight=caffe_model/resnet50.caffemodel --framework=0 --output=model/resnet50_aipp --soc_version=Ascend310 --insert_op_conf=caffe_model/aipp.cfg
   ```

   --output参数：生成的resnet50_aipp.om文件存放在“acl_dvpp_resnet50/model”目录下。建议使用命令中的默认设置，否则在编译代码前，您还需要修改sample_process.cpp中的omModelPath参数值。

   ```
   const char* omModelPath = "../model/resnet50_aipp.om";
   ```

   关于各参数的详细解释，请参见《**ATC工具使用指导**》中的“约束及参数说明”。
5. 将Cast和ArgMaxD两个算子的算子描述信息（*.json文件）编译成适配昇腾AI处理器的离线模型（*.om文件），用于运行算子时使用。

   切换到acl_dvpp_resnet50目录，执行如下命令：

   ```
   atc --singleop=out/op_models/op_list.json --soc_version=Ascend310 --output=out/op_models
   ```


--output参数：生成的om文件必须放在“out/op_models”目录下。

关于各参数的详细解释，请参见《**ATC工具使用指导**》中的“约束及参数说明”。

步骤2 编译代码。

1. 以HwHiAiUser（运行用户）登录开发环境。
2. 设置环境变量，编译脚本src/CMakeLists.txt通过环境变量所设置的头文件、库文件的路径来编译代码。

   编译脚本CMakeLists.txt中已默认设置编译依赖的头文件路径（/usr/local/Ascend/ascend-toolkit/latest/acllib/include/）、库文件路径（/usr/local/Ascend/ascend-toolkit/latest/acllib/lib64/stub/）。

   如果未设置${DDK_PATH}、${NPU_HOST_LIB}环境变量，则以编译脚本CMakeLists.txt中的路径为准；如果已设置${DDK_PATH}、${NPU_HOST_LIB}环境变量，则编译脚本CMakeLists.txt优先以环境变量为准，按环境变量指向的路径查找编译依赖的头文件和库文件。

   如下为设置环境变量的示例，请将*/home/HwHiAiUser/Ascend/ascend-toolkit/latest*替换为Ascend RC形态的ACLlib安装包的实际安装路径。

   ```
   export DDK_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/arm64-linux
   export NPU_HOST_LIB=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/arm64-linux/acllib/lib64/stub
   ```

   说明

   使用“*/home/HwHiAiUser/Ascend/ascend-toolkit/latest*/arm64-linux/acllib/lib64/stub”目录下的*.so库，是为了编译基于ACL接口的代码逻辑时，不依赖其它组件（例如Driver）的任何*.so库。

   编译通过后，在开发者板上运行应用时，通过配置环境变量，应用会链接到开发者板上“/home/HwHiAiUser/Ascend/acllib/lib64”目录下的*.so库，运行时会自动链接到依赖其它组件的*.so库。
3. 切换到acl_dvpp_resnet50目录，创建目录用于存放编译文件，例如，本文中，创建的目录为“build/intermediates/minirc”。

   ```
   mkdir -p build/intermediates/minirc
   ```
4. 切换到“build/intermediates/minirc”目录，执行**cmake**生成编译文件。

   “../../../src”表示CMakeLists.txt文件所在的目录，请根据实际目录层级修改。

   ```
   cd build/intermediates/minirc
   cmake ../../../src -DCMAKE_CXX_COMPILER=aarch64-linux-gnu-g++ -DCMAKE_SKIP_RPATH=TRUE
   ```
5. 执行**make**命令，生成的可执行文件main在“acl_dvpp_resnet50/out”目录下。

   ```
   make
   ```

步骤3 运行应用。

1. 以HwHiAiUser（运行用户）将开发环境的“acl_dvpp_resnet50”目录下src目录、out目录、data目录、model目录上传到开发者板的同一个目录下，例如“/home/HwHiAiUser/acl_dvpp_resnet50”。
2. 以HwHiAiUser（运行用户）登录开发者板。
3. 设置环境变量。

   如下为设置环境变量的示例，请根据实际安装情况替换路径。

   ```
   export LD_LIBRARY_PATH=/home/HwHiAiUser/Ascend/acllib/lib64
   ```
4. 切换到可执行文件main所在的目录，例如“/home/HwHiAiUser/acl_dvpp_resnet50/out”，给该目录下的main文件加执行权限。

   ```
   chmod +x main
   ```
5. 切换到可执行文件main所在的目录，例如“/home/HwHiAiUser/acl_dvpp_resnet50/out”，运行可执行文件。

   ```
   ./main
   ```


执行成功后，在屏幕上显示最大置信度的类别标识，示例如下。

```
[INFO]  acl init success
[INFO]  open device 0 success
[INFO]  create context success
[INFO]  create stream success
[INFO]  dvpp init resource success
[INFO]  load model ./test_data/model/resnet50_aipp.om success
[INFO]  create model description success
[INFO]  create model output success
[INFO]  start to process picture:./test_data/data/dog1_1024_683.jpg
[INFO]  Process dvpp success
[INFO]  model execute success
[INFO]  execute sigleOp Cast success
[INFO]  execute ArgMaxD success
[INFO]  sigleOp process success
[INFO]  ---> index of classification result is 161
[INFO]  start to process picture:./test_data/data/dog2_1024_683.jpg
[INFO]  Process dvpp success
[INFO]  model execute success
[INFO]  execute sigleOp Cast success
[INFO]  execute ArgMaxD success
[INFO]  sigleOp process success
[INFO]  ---> index of classification result is 267
[INFO]  unload model success, modelId is 1
[INFO]  execute sample success
[INFO]  end to destroy stream
[INFO]  end to destroy context
[INFO]  end to reset device is 0
```

----结束

9.5 基于 Caffe ResNet-50 网络实现图片分类（包含图片解码+抠图+缩放+编码）

9.5.1 样例介绍

获取样例

您可以从开发环境上的“ACLlib组件的安装目录/acllib/sample/acl_execute_model”目录下获取样例acl_vpc_jpege_resnet50样例的代码。

功能描述

该样例主要是基于Caffe ResNet-50网络（单输入、单batch）实现图片分类的功能。

根据运行应用的入参，该样例可实现以下功能：

- 将一张YUV420SP格式的图片编码为*.jpg格式的图片。
- 将两张*.jpg格式的解码成两张YUV420SP NV12格式的图片，缩放，再进行模型推理，分别得到两张图片的推理结果后，处理推理结果，输出最大置信度的类别标识以及top5置信度的总和。
- 将两张*.jpg格式的解码成两张YUV420SP NV12格式的图片，抠图，再进行模型推理，分别得到两张图片的推理结果后，处理推理结果，输出最大置信度的类别标识以及top5置信度的总和。

- 将两张*.jpg格式的解码成两张YUV420SP NV12格式的图片，抠图贴图，再进行模型推理，分别得到两张图片的推理结果后，处理推理结果，输出最大置信度的类别标识以及top5置信度的总和。

该样例中用于推理的模型文件是*.om文件（适配昇腾AI处理器的离线模型），转换模型时，需配置色域转换参数，用于将YUV420SP格式的图片转换为RGB格式的图片，才能符合模型的输入要求。

原理介绍

在该样例中，涉及的关键功能点，如下表所示。API接口的详细介绍请参见《应用软件开发指南》中的“AscendCL API参考”。

初始化	● 调用aclInit接口初始化ACL配置。 ● 调用aclFinalize接口实现ACL去初始化。
Device管理	● 调用aclrtSetDevice接口指定用于运算的Device。 ● 调用aclrtGetRunMode接口获取昇腾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aclrtResetDevice接口复位当前运算的Device，回收Device上的资源。
Context管理	● 调用aclrtCreateContext接口创建Context。 ● 调用aclrtDestroyContext接口销毁Context。
Stream管理	● 调用aclrtCreateStream接口创建Stream。 ● 调用aclrtDestroyStream接口销毁Stream。 ● 调用aclrtSynchronizeStream接口阻塞程序运行，直到指定stream中的所有任务都完成。
内存管理	● 调用aclrtMallocHost接口申请Host上内存。 ● 调用aclrtFreeHost释放Host上的内存。 ● 调用aclrtMalloc接口申请Device上的内存。 ● 调用aclrtFree接口释放Device上的内存。 执行数据预处理时，若需要申请Device上的内存存放输入或输出数据，需调用acldvppMalloc申请内存、调用acldvppFree接口释放内存。
数据传输	如果在开发者板上运行应用，则无需进行数据传输。

数据预处理	• 图片编码 调用acldvppJpegEncodeAsync接口将YUV420SP格式的图片编码为*.jpg格式的图片。 • 图片解码 调用acldvppJpegDecodeAsync接口将*.jpg图片解码成YUV420SP格式图片。 • 缩放 调用acldvppVpcResizeAsync接口将YUV420SP格式图片缩小成分辨率为224*224的图片。 • 抠图 调用acldvppVpcCropAsync接口按指定区域从输入图片中抠图，再将抠的图片存放到输出内存中，作为输出图片。 • 抠图贴图 调用acldvppVpcCropAndPasteAsync接口按指定区域从输入图片中抠图，再将抠的图片贴到目标图片的指定位置，作为输出图片。
模型推理	• 调用aclmdlLoadFromFileWithMem接口从*.om文件加载模型。 • 调用aclmdlExecute接口执行模型推理。 推理前，通过*.om文件中的色域转换参数将YUV420SP格式的图片转换为RGB格式的图片。 • 调用aclmdlUnload接口卸载模型。

目录结构

acl_vpc_jpege_resnet50样例代码结构如下所示。

```
├acl_vpc_jpege_resnet50
├──── caffe_model
│    ├──── aipp.cfg        //带色域转换参数的配置文件，模型转换时使用
│    ├──── resnet50.prototxt       //resnet50网络的模型文件

├──── data
│    ├──── persian_cat_1024_1536_283.jpg           //测试数据
│    ├──── wood_rabbit_1024_1061_330.jpg            //测试数据
│    ├──── wood_rabbit_1024_1068_nv12.yuv           //测试数据

├──── inc
│    ├──── dvpp_process.h              //声明数据预处理相关函数的头文件
│    ├──── model_process.h             //声明模型处理相关函数的头文件
│    ├──── sample_process.h              //声明资源初始化/销毁相关函数的头文件
│    ├──── utils.h                     //声明公共函数（例如：文件读取函数）的头文件

├──── src
│    ├──── acl.json         //系统初始化的配置文件
│    ├──── CMakeLists.txt         //编译脚本
│    ├──── dvpp_process.cpp       //数据预处理相关函数的实现文件
│    ├──── main.cpp              //主函数，图片分类功能的实现文件
│    ├──── model_process.cpp      //模型处理相关函数的实现文件
│    ├──── sample_process.cpp     //资源初始化/销毁相关函数的实现文件
│    ├──── utils.cpp             //公共函数（例如：文件读取函数）的实现文件

├──── .project     //工程信息文件，包含工程类型、工程描述、运行目标设备类型等
├──── CMakeLists.txt     //编译脚本，调用src目录下的CMakeLists文件
```

9.5.2 Ascend RC，编译及运行应用

步骤1 模型转换。

1. 以HwHiAiUser（运行用户）登录开发环境。
2. 参考《**ATC工具使用指导**》中的使用ATC工具转换模型，执行“准备动作”，包括获取工具、设置环境变量。
3. 准备数据。

   从以下链接获取ResNet-50网络的权重文件（*.caffemodel），并以HwHiAiUser（运行用户）将获取的文件上传至开发环境的“acl_vpc_jpege_resnet50/caffe_model”目录下。

   - 从gitee上获取：单击**Link**，查看README.md，查找获取原始模型的链接。
   - 从GitHub上获取：单击**Link**，查看README.md，查找获取原始模型的链接。
4. 将ResNet-50网络转换为适配昇腾AI处理器的离线模型（*.om文件），转换模型时，需配置色域转换参数，用于将YUV420SP格式的图片转换为RGB格式的图片。

   切换到“acl_vpc_jpege_resnet50”目录，执行如下命令：

   ```
   atc --model=caffe_model/resnet50.prototxt --weight=caffe_model/resnet50.caffemodel --framework=0 --output=model/resnet50_aipp --soc_version=Ascend310 --insert_op_conf=caffe_model/aipp.cfg
   ```

   --output参数：生成的resnet50_aipp.om文件存放在“acl_vpc_jpege_resnet50/model”目录下。建议使用命令中的默认设置，否则在编译代码前，您还需要修改sample_process.cpp中的omModelPath参数值。

   ```
   const char* omModelPath = "../model/resnet50_aipp.om";
   ```

   关于各参数的详细解释，请参见《**ATC工具使用指导**》中的“约束及参数说明”。

步骤2 编译代码。

1. 以HwHiAiUser（运行用户）登录开发环境。
2. 设置环境变量，编译脚本src/CMakeLists.txt通过环境变量所设置的头文件、库文件的路径来编译代码。

   编译脚本CMakeLists.txt中已默认设置编译依赖的头文件路径（/usr/local/Ascend/ascend-toolkit/latest/acllib/include/）、库文件路径（/usr/local/Ascend/ascend-toolkit/latest/acllib/lib64/stub/）。

   如果未设置${DDK_PATH}、${NPU_HOST_LIB}环境变量，则以编译脚本CMakeLists.txt中的路径为准；如果已设置${DDK_PATH}、${NPU_HOST_LIB}环境变量，则编译脚本CMakeLists.txt优先以环境变量为准，按环境变量指向的路径查找编译依赖的头文件和库文件。

   如下为设置环境变量的示例，请将*/home/HwHiAiUser/Ascend/ascend-toolkit/latest*替换为Ascend RC形态的ACLlib安装包的实际安装路径。

   ```
   export DDK_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/arm64-linux
   export NPU_HOST_LIB=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/arm64-linux/acllib/lib64/stub
   ```

   说明

   使用“*/home/HwHiAiUser/Ascend/ascend-toolkit/latest*/arm64-linux/acllib/lib64/stub”目录下的*.so库，是为了编译基于ACL接口的代码逻辑时，不依赖其它组件（例如Driver）的任何*.so库。

   编译通过后，在开发者板上运行应用时，通过配置环境变量，应用会链接到开发者板上“/home/HwHiAiUser/Ascend/acllib/lib64”目录下的*.so库，运行时会自动链接到依赖其它组件的*.so库。

3. 切换到“acl_vpc_jpege_resnet50”目录，创建目录用于存放编译文件，例如，本文中，创建的目录为“build/intermediates/minirc”。
```
mkdir -p build/intermediates/minirc
```
4. 切换到“build/intermediates/minirc”目录，执行**cmake**生成编译文件。

   “../../../src”表示CMakeLists.txt文件所在的目录，请根据实际目录层级修改。
```
cd build/intermediates/minirc
cmake ../../../src -DCMAKE_CXX_COMPILER=aarch64-linux-gnu-g++ -DCMAKE_SKIP_RPATH=TRUE
```
5. 执行**make**命令，生成的可执行文件main在“acl_vpc_jpege_resnet50/out”目录下。
```
make
```

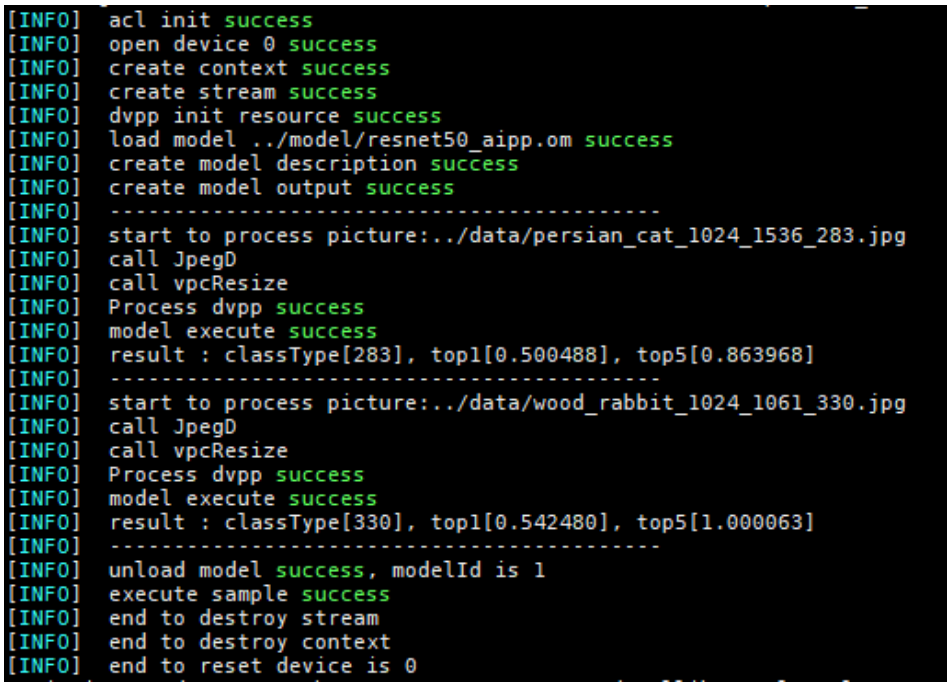
步骤3 运行应用。

1. 以HwHiAiUser（运行用户）将开发环境的“acl_vpc_jpege_resnet50”目录下的src目录、out目录、data目录、model目录上传到开发者板的同一个目录下，例如“/home/HwHiAiUser/acl_vpc_jpege_resnet50”。
2. 以HwHiAiUser（运行用户）登录开发者板。
3. 设置环境变量。

   如下为设置环境变量的示例，请根据实际安装情况替换路径。
```
export LD_LIBRARY_PATH=/home/HwHiAiUser/Ascend/acllib/lib64
```
4. 切换到可执行文件main所在的目录，例如“/home/HwHiAiUser/acl_vpc_jpege_resnet50/out”，给该目录下的main文件加执行权限。
```
chmod +x main
```
5. 切换到可执行文件main所在的目录，例如“/home/HwHiAiUser/acl_vpc_jpege_resnet50/out”，运行可执行文件。

   a. 将两张*.jpg格式的解码成两张YUV420SP NV12格式的图片，缩放，再进行模型推理，分别得到两张图片的推理结果。
```
./main 0
```

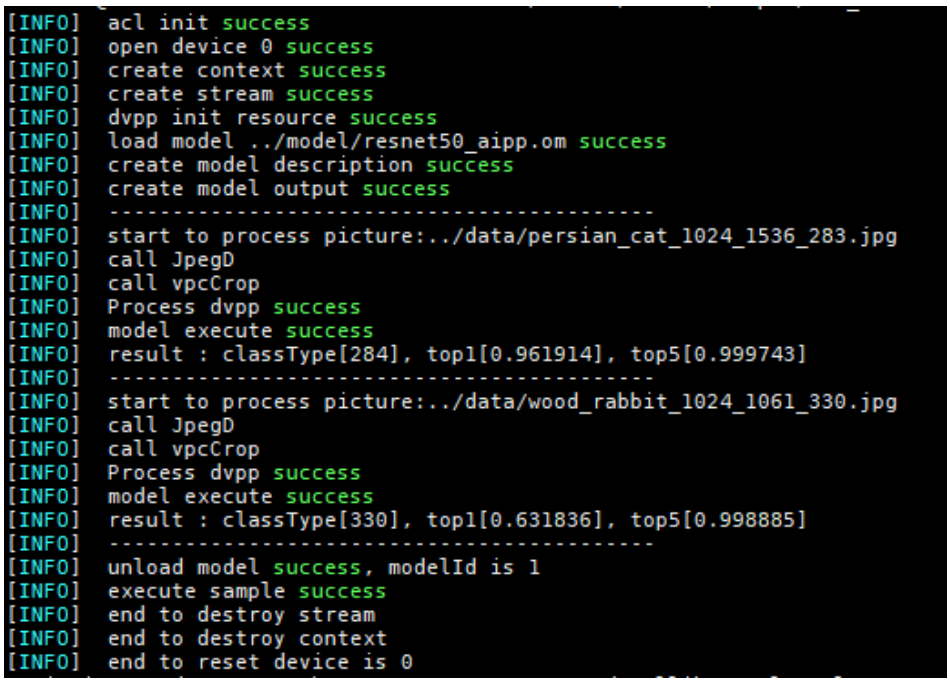
   图 9-3 执行结果示例



   b. 将两张*.jpg格式的解码成两张YUV420SP NV12格式的图片，抠图，再进行模型推理，分别得到两张图片的推理结果。


```
./main 1
```

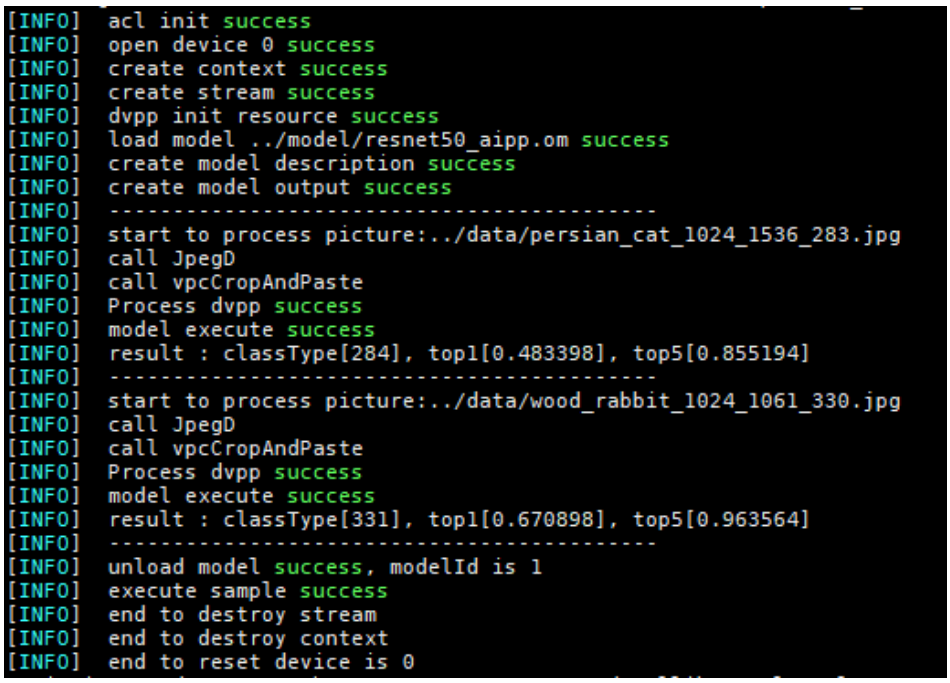
图 9-4 执行结果示例



c. 将两张*.jpg格式的解码成两张YUV420SP NV12格式的图片，抠图贴图，再进行模型推理，分别得到两张图片的推理结果。

```
./main 2
```

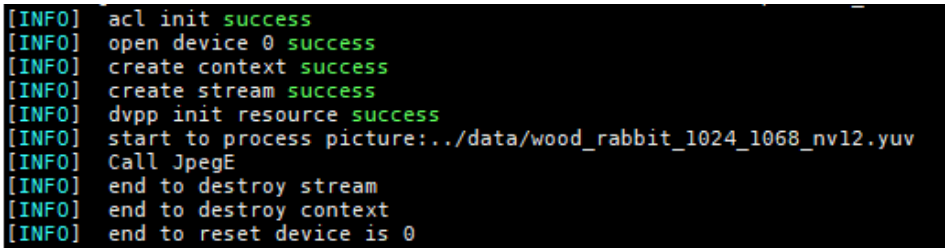
图 9-5 执行结果示例



d. 将一张YUV420SP格式的图片编码为*.jpg格式的图片。

```
./main 3
```


图 9-6 执行结果示例



说明

执行可执行文件成功后，同时会在main文件同级的result目录下生成结果文件，便于后期查看。结果文件如下：

- dvpp_output_0：persian_cat_1024_1536_283.jpg图片经过缩放或抠图或抠图贴图之后的结果图片。
- dvpp_output_1：wood_rabbit_1024_1061_330.jpg图片经过缩放或抠图或抠图贴图之后的结果图片。
- model_output_0：persian_cat_1024_1536_283.jpg图片的模型推理结果，二进制文件。
- model_output_0.txt：persian_cat_1024_1536_283.jpg图片的模型推理结果，txt文件。
- model_output_1：wood_rabbit_1024_1061_330.jpg图片的模型推理结果，二进制文件。
- model_output_1.txt：wood_rabbit_1024_1061_330.jpg图片的模型推理结果，txt文件。
- jpege_output_0.jpg：wood_rabbit_1024_1068_nv12.yuv图片结果编码后的结果图片。

----结束

9.6 基于 Caffe ResNet-50 网络实现图片分类（包含视频解码）

9.6.1 样例介绍

获取样例

您可以从开发环境上的“ACLlib组件的安装目录/acllib/sample/acl_execute_model”目录下获取样例acl_vdec_resnet50样例的代码。

功能描述

该样例主要是基于Caffe ResNet-50网络（单输入、单batch）实现图片分类的功能。

将Caffe ResNet-50网络的模型文件转换为适配昇腾AI处理器的离线模型（*.om文件），在样例中，加载该om文件，将1个*.h265格式的视频码流（仅包含一帧）循环10次解码出10张YUV420SP NV12格式的图片，对该10张图片做缩放，再对10张YUV420SP NV12格式的图片进行推理，分别得到推理结果后，再对推理结果进行处理，输出最大置信度的类别标识以及top5置信度的总和。

转换模型时，需配置色域转换参数，用于将YUV420SP格式的图片转换为RGB格式的图片，才能符合模型的输入要求。

原理介绍

在该样例中，涉及的关键功能点，如下表所示。API接口的详细介绍请参见《 应用软件开发指南 》中的“AscendCL API参考”。

初始化	● 调用aclInit接口初始化ACL配置。 ● 调用aclFinalize接口实现ACL去初始化。
Device管理	● 调用aclrtSetDevice接口指定用于运算的Device。 ● 调用aclrtGetRunMode接口获取昇腾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aclrtResetDevice接口复位当前运算的Device，回收Device上的资源。
Context管理	● 调用aclrtCreateContext接口创建Context。 ● 调用aclrtDestroyContext接口销毁Context。
Stream管理	● 调用aclrtCreateStream接口创建Stream。 ● 调用aclrtDestroyStream接口销毁Stream。 ● 调用aclrtSynchronizeStream接口阻塞程序运行，直到指定stream中的所有任务都完成。
内存管理	调用aclrtMallocHost接口申请Host上内存。 ● 调用aclrtFreeHost释放Host上的内存。 ● 调用aclrtMalloc接口申请Device上的内存。 ● 调用aclrtFree接口释放Device上的内存。 执行数据预处理时，若需要申请Device上的内存存放输入或输出数据，需调用acldvppMalloc申请内存、调用acldvppFree接口释放内存。
数据传输	如果在开发者板上运行应用，则无需进行数据传输。
数据预处理	● 视频解码 调用aclvdecSendFrame接口将视频码流解码成YUV420SP格式图片。 ● 缩放 调用acldvppVpcResizeAsync接口将YUV420SP NV12格式图片缩小成分辨率为224*224的图片。
模型推理	● 调用aclmdlLoadFromFileWithMem接口从*.om文件加载模型。 ● 调用aclmdlExecute接口执行模型推理。 推理前，通过*.om文件中的色域转换参数将YUV420SP格式的图片转换为RGB格式的图片。 ● 调用aclmdlUnload接口卸载模型。

目录结构

acl_vdec_resnet50样例代码结构如下所示。

```
├acl_vdec_resnet50
├──── caffe_model
|    ├──── aipp.cfg        //带色域转换参数的配置文件，模型转换时使用
|    ├──── resnet50.prototxt       //resnet50网络的模型文件

├──── inc
|    ├──── dvpp_process.h             //声明数据预处理相关函数的头文件
|    ├──── model_process.h             //声明模型处理相关函数的头文件
|    ├──── sample_process.h              //声明资源初始化/销毁相关函数的头文件
|    ├──── utils.h                    //声明公共函数（例如：文件读取函数）的头文件
|    ├──── vdec_process.h               //声明视频处理函数的头文件

├──── src
|    ├──── acl.json         //系统初始化的配置文件
|    ├──── CMakeLists.txt         //编译脚本
|    ├──── dvpp_process.cpp       //数据预处理相关函数的实现文件
|    ├──── main.cpp              //主函数，图片分类功能的实现文件
|    ├──── model_process.cpp      //模型处理相关函数的实现文件
|    ├──── sample_process.cpp     //资源初始化/销毁相关函数的实现文件
|    ├──── utils.cpp             //公共函数（例如：文件读取函数）的实现文件
|    ├──── vdec_process.cpp               //声明视频处理函数的实现文件

├──── .project     //工程信息文件，包含工程类型、工程描述、运行目标设备类型等
├──── CMakeLists.txt     //编译脚本，调用src目录下的CMakeLists文件
```

9.6.2 Ascend RC，编译及运行应用

步骤1 模型转换。

1. 以HwHiAiUser（运行用户）登录开发环境。
2. 参考《**ATC工具使用指导**》中的使用ATC工具转换模型，执行“准备动作”，包括获取工具、设置环境变量。
3. 准备数据。

   从以下链接获取ResNet-50网络的权重文件（*.caffemodel），并以HwHiAiUser（运行用户）将获取的文件上传至开发环境的“acl_vdec_resnet50/caffe_model”目录下。

   - 从gitee上获取：单击**Link**，查看README.md，查找获取原始模型的链接。
   - 从GitHub上获取：单击**Link**，查看README.md，查找获取原始模型的链接。
4. 将ResNet-50网络转换为适配昇腾AI处理器的离线模型（*.om文件），转换模型时，需配置色域转换参数，用于将YUV420SP格式的图片转换为RGB格式的图片。

   切换到“acl_vdec_resnet50”目录，执行如下命令：

   ```
   atc --model=caffe_model/resnet50.prototxt --weight=caffe_model/resnet50.caffemodel --framework=0 --output=model/resnet50_aipp --soc_version=Ascend310 --insert_op_conf=caffe_model/aipp.cfg
   ```

   --output参数：生成的resnet50_aipp.om文件存放在“acl_vdec_resnet50/model”目录下。建议使用命令中的默认设置，否则在编译代码前，您还需要修改sample_process.cpp中的omModelPath参数值。

   ```
   const char* omModelPath = "../model/resnet50_aipp.om";
   ```

   关于各参数的详细解释，请参见《**ATC工具使用指导**》中的“约束及参数说明”。

步骤2 编译代码。

1. 准备输入视频码流。

   从**https://gitee.com/HuaweiAscend/tools/tree/master/dvpp_sample_input_data**上获取输入视频码流文件vdec_h265_1frame_rabbit_1280x720.h265，并以HwHiAiUser（运行用户）上传至开发环境的“acl_vdec_resnet50”目录下。

2. 以HwHiAiUser（运行用户）登录开发环境。

3. 设置环境变量，编译脚本src/CMakeLists.txt通过环境变量所设置的头文件、库文件的路径来编译代码。

   编译脚本CMakeLists.txt中已默认设置编译依赖的头文件路径（/usr/local/Ascend/ascend-toolkit/latest/acllib/include/）、库文件路径（/usr/local/Ascend/ascend-toolkit/latest/acllib/lib64/stub/）。

   如果未设置${DDK_PATH}、${NPU_HOST_LIB}环境变量，则以编译脚本CMakeLists.txt中的路径为准；如果已设置${DDK_PATH}、${NPU_HOST_LIB}环境变量，则编译脚本CMakeLists.txt优先以环境变量为准，按环境变量指向的路径查找编译依赖的头文件和库文件。

   如下为设置环境变量的示例，请将*/home/HwHiAiUser/Ascend/ascend-toolkit/latest*替换为Ascend RC形态的ACLlib安装包的实际安装路径。

   ```
   export DDK_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/arm64-linux
   export NPU_HOST_LIB=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/arm64-linux/acllib/lib64/stub
   ```

   说明

   使用“*/home/HwHiAiUser/Ascend/ascend-toolkit/latest*/arm64-linux/acllib/lib64/stub”目录下的*.so库，是为了编译基于ACL接口的代码逻辑时，不依赖其它组件（例如Driver）的任何*.so库。

   编译通过后，在开发者板上运行应用时，通过配置环境变量，应用会链接到开发者板上“/home/HwHiAiUser/Ascend/acllib/lib64”目录下的*.so库，运行时会自动链接到依赖其它组件的*.so库。

4. 切换到“acl_vdec_resnet50”目录，创建目录用于存放编译文件，例如，本文中，创建的目录为“build/intermediates/minirc”。

   ```
   mkdir -p build/intermediates/minirc
   ```

5. 切换到“build/intermediates/minirc”目录，执行**cmake**生成编译文件。

   “../../../src”表示CMakeLists.txt文件所在的目录，请根据实际目录层级修改。

   ```
   cd build/intermediates/minirc
   cmake ../../../src -DCMAKE_CXX_COMPILER=aarch64-linux-gnu-g++ -DCMAKE_SKIP_RPATH=TRUE
   ```

6. 执行**make**命令，生成的可执行文件main在“acl_vdec_resnet50/out”目录下。

   ```
   make
   ```

步骤3 运行应用。

1. 以HwHiAiUser（运行用户）将开发环境的“acl_vdec_resnet50”目录下的src目录、out目录、data目录、model目录上传到开发者板的同一个目录下，例如“/home/HwHiAiUser/acl_vdec_resnet50”。

2. 以HwHiAiUser（运行用户）登录开发者板。

3. 设置环境变量。

   如下为设置环境变量的示例，请根据实际安装情况替换路径。

   ```
   export LD_LIBRARY_PATH=/home/HwHiAiUser/Ascend/acllib/lib64
   ```

4. 切换到可执行文件main所在的目录，例如“/home/HwHiAiUser/acl_vdec_resnet50/out”，给该目录下的main文件加执行权限。

   ```
   chmod +x main
   ```

5. 切换到可执行文件main所在的目录，例如“/home/HwHiAiUser/acl_vdec_resnet50/out”，运行可执行文件。

   ```
   ./main
   ```


可执行文件执行成功后，屏幕提示如下示例信息：

```
......
[INFO]  output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO]  memcopy output data from device to host buffer success.
[INFO]  create output file success, filename=./result/model_output_0, size=4000
[INFO]  start check result file:./result/model_output_0
[INFO]  check result success, file exist
[INFO]  reselut file: [./result/model_output_0.txt]
[INFO]  result:classType[331],top1[0.908203],top5[1.000015]
[INFO]  Process dvpp success
[INFO]  model execute success
[INFO]  output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO]  memcopy output data from device to host buffer success.
[INFO]  create output file success, filename=./result/model_output_1, size=4000
[INFO]  start check result file:./result/model_output_1
[INFO]  check result success, file exist
[INFO]  reselut file: [./result/model_output_1.txt]
[INFO]  result:classType[688],top1[0.596680],top5[0.901611]
[INFO]  Process dvpp success
[INFO]  model execute success
[INFO]  output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO]  memcopy output data from device to host buffer success.
[INFO]  create output file success, filename=./result/model_output_2, size=4000
[INFO]  start check result file:./result/model_output_2
[INFO]  check result success, file exist
[INFO]  reselut file: [./result/model_output_2.txt]
[INFO]  result:classType[331],top1[0.908203],top5[1.000015]
[INFO]  Process dvpp success
[INFO]  model execute success
[INFO]  output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO]  memcopy output data from device to host buffer success.
[INFO]  create output file success, filename=./result/model_output_3, size=4000
[INFO]  start check result file:./result/model_output_3
[INFO]  check result success, file exist
[INFO]  reselut file: [./result/model_output_3.txt]
[INFO]  result:classType[331],top1[0.908203],top5[1.000015]
[INFO]  Process dvpp success
[INFO]  model execute success
[INFO]  output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO]  memcopy output data from device to host buffer success.
[INFO]  create output file success, filename=./result/model_output_4, size=4000
[INFO]  start check result file:./result/model_output_4
[INFO]  check result success, file exist
[INFO]  reselut file: [./result/model_output_4.txt]
[INFO]  result:classType[331],top1[0.908203],top5[1.000015]
[INFO]  Process dvpp success
[INFO]  model execute success
[INFO]  output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO]  memcopy output data from device to host buffer success.
[INFO]  create output file success, filename=./result/model_output_5, size=4000
[INFO]  start check result file:./result/model_output_5
[INFO]  check result success, file exist
[INFO]  reselut file: [./result/model_output_5.txt]
[INFO]  result:classType[331],top1[0.908203],top5[1.000015]
[INFO]  Process dvpp success
[INFO]  model execute success
[INFO]  output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO]  memcopy output data from device to host buffer success.
[INFO]  create output file success, filename=./result/model_output_6, size=4000
[INFO]  start check result file:./result/model_output_6
[INFO]  check result success, file exist
[INFO]  reselut file: [./result/model_output_6.txt]
[INFO]  result:classType[331],top1[0.908203],top5[1.000015]
[INFO]  Process dvpp success
[INFO]  model execute success
[INFO]  output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO]  memcopy output data from device to host buffer success.
[INFO]  create output file success, filename=./result/model_output_7, size=4000
[INFO]  start check result file:./result/model_output_7
```



```
[INFO]  check result success, file exist
[INFO]  reselut file: [./result/model_output_7.txt]
[INFO]  result:classType[331],top1[0.908203],top5[1.000015]
[INFO]  Process dvpp success
[INFO]  model execute success
[INFO]  output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO]  memcopy output data from device to host buffer success.
[INFO]  create output file success, filename=./result/model_output_8, size=4000
[INFO]  start check result file:./result/model_output_8
[INFO]  check result success, file exist
[INFO]  reselut file: [./result/model_output_8.txt]
[INFO]  result:classType[331],top1[0.908203],top5[1.000015]
[INFO]  Process dvpp success
[INFO]  model execute success
[INFO]  output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO]  memcopy output data from device to host buffer success.
[INFO]  create output file success, filename=./result/model_output_9, size=4000
[INFO]  start check result file:./result/model_output_9
[INFO]  check result success, file exist
[INFO]  reselut file: [./result/model_output_9.txt]
[INFO]  result:classType[331],top1[0.908203],top5[1.000015]
[INFO]  unload model success, modelId is 1
[INFO]  execute sample success
......
```

说明

- 可执行文件运行过程中，VDEC解码成功后，模型推理前，会在main文件同级生成outdir目录，用于保存解码后的YUV420SP NV12格式的图片，推理结束后，系统会自动清理outdir目录。
- 可执行文件执行成功后，同时会在main文件同级的result目录下生成结果文件，便于后期查看。*.h265格式的视频码流（仅包含一帧）被解码出一张YUV420SP NV12格式的图片，对该张图片做十次循环处理，包括缩放图片和模型推理，分别得到十张图片的推理结果：
  - model_output_*：模型推理结果的二进制文件。
  - model_output_*.txt：模型推理结果的txt文件。

----结束

9.7 基于 Caffe ResNet-50 网络实现图片分类（同步推理，不含数据预处理）

9.7.1 样例介绍

获取样例

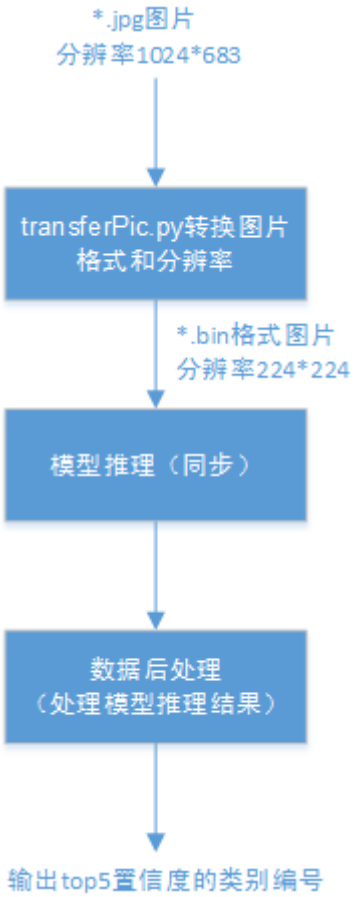
您可以从开发环境上的“ACLlib组件的安装目录/acllib/sample/acl_execute_model”目录下获取样例acl_resnet50的代码。

功能描述

该样例主要是基于Caffe ResNet-50网络（单输入、单batch）实现图片分类的功能。

将Caffe ResNet-50网络的模型文件转换为适配昇腾AI处理器的离线模型（*.om文件），在样例中，加载该om文件，对2张*.jpg图片进行同步推理，分别得到推理结果后，再对推理结果进行处理，输出top5置信度的类别标识。

图 9-7 Sample 示例



原理介绍

在该Sample中，涉及的关键功能点，如下表所示。API接口的详细介绍请参见《应用软件开发指南》中的“AscendCL API参考”。

初始化	• 调用aclInit接口初始化ACL配置。 • 调用aclFinalize接口实现ACL去初始化。
Device管理	• 调用aclrtSetDevice接口指定用于运算的Device。 • 调用aclrtGetRunMode接口获取昇腾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aclrtResetDevice接口复位当前运算的Device，回收Device上的资源。
Context管理	• 调用aclrtCreateContext接口创建Context。 • 调用aclrtDestroyContext接口销毁Context。
Stream管理	• 调用aclrtCreateStream接口创建Stream。 • 调用aclrtDestroyStream接口销毁Stream。

内存管理	● 调用aclrtMalloc接口申请Device上的内存。 ● 调用aclrtFree接口释放Device上的内存。
数据传输	如果在开发者板上运行应用，则无需进行数据传输。
模型推理	● 调用aclmdlLoadFromFileWithMem接口从*.om文件加载模型。 ● 调用aclmdlExecute接口执行模型推理，同步接口。 ● 调用aclmdlUnload接口卸载模型。

数据后处理	提供样例代码，处理模型推理的结果，直接在终端上显示top5置信度的类别编号。
	另外，样例中提供了自定义接口DumpModelOutputResult，用于将模型推理的结果写入文件（运行可执行文件后，推理结果文件在应用可执行文件的同级目录下），默认未调用该接口，用户可在sample_process.cpp中，在调用OutputModelResult接口前，增加如下代码调用DumpModelOutputResult接口：
	// print the top 5 confidence values with indexes.use function DumpModelOutputResult
	// if want to dump output result to file in the current directory
	processModel.DumpModelOutputResult();
	processModel.OutputModelResult();

目录结构

样例代码结构如下所示。

```
├acl_resnet50
├─── caffe_model
|    ├─── resnet50.prototxt       //resnet50网络的模型文件
|
├─── data
|    ├─── dog1_1024_683.jpg           //测试数据
|    ├─── dog2_1024_683.jpg           //测试数据
|
├─── inc
|    ├─── model_process.h             //声明模型处理相关函数的头文件
|    ├─── sample_process.h            //声明资源初始化/销毁相关函数的头文件
|    ├─── utils.h                     //声明公共函数（例如：文件读取函数）的头文件
|
├─── script
|    ├─── transferPic.py              //将*.jpg转换为*.bin，同时将图片从1024*683的分辨率缩放为224*224
|
├─── src
|    ├─── acl.json         //系统初始化的配置文件
|    ├─── CMakeLists.txt         //编译脚本
|    ├─── main.cpp               //主函数，图片分类功能的实现文件
|    ├─── model_process.cpp      //模型处理相关函数的实现文件
|    ├─── sample_process.cpp     //资源初始化/销毁相关函数的实现文件
|    ├─── utils.cpp              //公共函数（例如：文件读取函数）的实现文件
|
├─── .project     //工程信息文件，包含工程类型、工程描述、运行目标设备类型等
├─── CMakeLists.txt    //编译脚本，调用src目录下的CMakeLists文件
```

9.7.2 Ascend RC，编译及运行应用

步骤1 模型转换。

1. 以HwHiAiUser（运行用户）登录开发环境。

2. 参考《**ATC工具使用指导**》中的使用ATC工具转换模型，执行“准备动作”，包括获取工具、设置环境变量。
3. 准备数据。

   从以下链接获取ResNet-50网络的权重文件（*.caffemodel），并以HwHiAiUser（运行用户）将获取的文件上传至开发环境的“acl_resnet50/caffe_model”目录下。

   - 从gitee上获取：单击**Link**，查看README.md，查找获取原始模型的链接。
   - 从GitHub上获取：单击**Link**，查看README.md，查找获取原始模型的链接。
4. 将ResNet-50网络转换为适配昇腾AI处理器的离线模型（*.om文件）。

   切换到“acl_resnet50”目录，执行如下命令：

   ```
   atc --model=caffe_model/resnet50.prototxt --weight=caffe_model/resnet50.caffemodel --framework=0 --output=model/resnet50 --soc_version=Ascend310 --input_format=NCHW --input_fp16_nodes=data -output_type=FP32 --out_nodes=prob:0
   ```

   --output参数：生成的resnet50.om文件存放在“acl_resnet50/model”目录下。建议使用命令中的默认设置，否则在编译代码前，您还需要修改sample_process.cpp中的omModelPath参数值。

   ```
   const char* omModelPath = "../model/resnet50.om";
   ```

   关于各参数的详细解释，请参见《**ATC工具使用指导**》中的“约束及参数说明”。

步骤2 编译代码。

1. 以HwHiAiUser（运行用户）登录开发环境。
2. 切换到“acl_resnet50/data”目录下，执行transferPic.py脚本，将*.jpg转换为*.bin，同时将图片从1024*683的分辨率缩放为224*224。在“acl_resnet50/data”目录下生成2个*.bin文件。

   ```
   python3.7.5 ../script/transferPic.py
   ```

   说明

   如果执行脚本报错“ModuleNotFoundError: No module named 'PIL'”，则表示缺少Pillow库，请使用**pip3.7.5 install Pillow --user**命令安装Pillow库。
3. 设置环境变量，编译脚本src/CMakeLists.txt通过环境变量所设置的头文件、库文件的路径来编译代码。

   编译脚本CMakeLists.txt中已默认设置编译依赖的头文件路径（/usr/local/Ascend/ascend-toolkit/latest/acllib/include/）、库文件路径（/usr/local/Ascend/ascend-toolkit/latest/acllib/lib64/stub/）。

   如果未设置${DDK_PATH}、${NPU_HOST_LIB}环境变量，则以编译脚本CMakeLists.txt中的路径为准；如果已设置${DDK_PATH}、${NPU_HOST_LIB}环境变量，则编译脚本CMakeLists.txt优先以环境变量为准，按环境变量指向的路径查找编译依赖的头文件和库文件。

   如下为设置环境变量的示例，请将*/home/HwHiAiUser/Ascend/ascend-toolkit/latest*替换为Ascend RC形态的ACLlib安装包的实际安装路径。

   ```
   export DDK_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/arm64-linux
   export NPU_HOST_LIB=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/arm64-linux/acllib/lib64/stub
   ```

说明

使用“*/home/HwHiAiUser/Ascend/ascend-toolkit/latest*/arm64-linux/acllib/lib64/stub”目录下的*.so库，是为了编译基于ACL接口的代码逻辑时，不依赖其它组件（例如Driver）的任何*.so库。

编译通过后，在开发者板上运行应用时，通过配置环境变量，应用会链接到开发者板上“/home/HwHiAiUser/Ascend/acllib/lib64”目录下的*.so库，运行时会自动链接到依赖其它组件的*.so库。

4. 切换到acl_resnet50目录，创建目录用于存放编译文件，例如，本文中，创建的目录为“build/intermediates/minirc”。
```
mkdir -p build/intermediates/minirc
```
5. 切换到“build/intermediates/minirc”目录，执行**cmake**生成编译文件。

   “../../../src”表示CMakeLists.txt文件所在的目录，请根据实际目录层级修改。
```
cd build/intermediates/minirc
cmake ../../../src -DCMAKE_CXX_COMPILER=aarch64-linux-gnu-g++ -DCMAKE_SKIP_RPATH=TRUE
```
6. 执行**make**命令，生成的可执行文件main在“acl_resnet50/out”目录下。
```
make
```

步骤3 运行应用。

1. 以HwHiAiUser（运行用户）将开发环境的“acl_resnet50”目录下的src目录、out目录、data目录、model目录上传到开发者板的同一个目录下，例如“/home/HwHiAiUser/acl_resnet50”。
2. 以HwHiAiUser用户登录开发者板。
3. 设置环境变量。

   如下为设置环境变量的示例，请根据实际安装情况替换路径。
```
export LD_LIBRARY_PATH=/home/HwHiAiUser/Ascend/acllib/lib64
```
4. 切换到可执行文件main所在的目录，例如“/home/HwHiAiUser/acl_resnet50/out”，给该目录下的main文件加执行权限。
```
chmod +x main
```
5. 切换到可执行文件main所在的目录，例如“/home/HwHiAiUser/acl_resnet50/out”，运行可执行文件。
```
./main
```

   执行成功后，在屏幕上提示如下示例信息：


```
[INFO]  acl init success
[INFO]  open device 0 success
[INFO]  create context success
[INFO]  create stream success
[INFO]  load model ../model/resnet50.om success
[INFO]  create model description success
[INFO]  create model output success
[INFO]  start to process file:../data/dog1_1024_683.bin
[INFO]  model execute success
[INFO]  top 1: index[161] value[0.900391]
[INFO]  top 2: index[164] value[0.038666]
[INFO]  top 3: index[163] value[0.019287]
[INFO]  top 4: index[166] value[0.016357]
[INFO]  top 5: index[167] value[0.012161]
[INFO]  output data success
[INFO]  start to process file:../data/dog2_1024_683.bin
[INFO]  model execute success
[INFO]  top 1: index[267] value[0.974609]
[INFO]  top 2: index[266] value[0.013062]
[INFO]  top 3: index[265] value[0.010017]
[INFO]  top 4: index[129] value[0.000335]
[INFO]  top 5: index[372] value[0.000179]
[INFO]  output data success
[INFO]  unload model success, modelId is 1
[INFO]  execute sample success
[INFO]  end to destroy stream
[INFO]  end to destroy context
[INFO]  end to reset device is 0
```

----结束

9.8 基于 Caffe ResNet-50 网络实现图片分类（异步推理，不含数据预处理）

9.8.1 样例介绍

获取样例

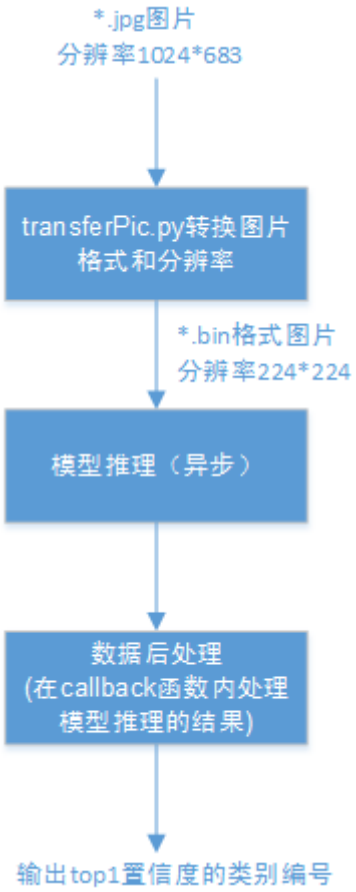
您可以从开发环境上的“ACLlib组件的安装目录/acllib/sample/acl_execute_model”目录下获取样例acl_resnet50_async的代码。

功能描述

该样例主要是基于Caffe ResNet-50网络（单输入、单batch）实现图片分类的功能。

将Caffe ResNet-50网络的模型文件转换为适配昇腾AI处理器的离线模型（*.om文件），在样例中，加载该om文件，对2张*.jpg图片进行n次异步推理（n作为运行应用的参数，由用户配置），分别得到n次推理结果后，再对推理结果进行处理，输出top1置信度的类别标识。

图 9-8 Sample 示例



原理介绍

在该Sample中，涉及的关键功能点，如下表所示。API接口的详细介绍请参见《应用软件开发指南》中的“AscendCL API参考”。

初始化	• 调用aclInit接口初始化ACL配置。 • 调用aclFinalize接口实现ACL去初始化。
Device管理	• 调用aclrtSetDevice接口指定用于运算的Device。 • 调用aclrtGetRunMode接口获取昇腾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aclrtResetDevice接口复位当前运算的Device，回收Device上的资源。
Context管理	• 调用aclrtCreateContext接口创建Context。 • 调用aclrtSetCurrentContext接口设置线程的Context。 • 调用aclrtDestroyContext接口销毁Context。
Stream管理	• 调用aclrtCreateStream接口创建Stream。 • 调用aclrtDestroyStream接口销毁Stream。

内存管理	• 调用aclrtMallocHost接口申请Host上内存。 • 调用aclrtFreeHost释放Host上的内存。 • 调用aclrtMalloc接口申请Device上的内存。 • 调用aclrtFree接口释放Device上的内存。
数据传输	如果在开发者板上运行应用，则无需进行数据传输。
模型推理	• 调用aclmdlLoadFromFileWithMem接口从*.om文件加载模型。 • 创建新线程（例如t1），在线程函数内调用aclrtProcessReport接口，等待指定时间后，触发回调函数（例如CallBackFunc，用于处理模型推理结果）。 • 调用aclrtSubscribeReport接口，指定处理Stream上回调函数（CallBackFunc）的线程（t1）。 • 调用aclmdlExecuteAsync接口执行模型推理，异步接口。 • 调用aclrtLaunchCallback接口，在Stream的任务队列中增加一个需要在Host/Device上执行的回调函数（CallBackFunc）。 • 调用aclrtSynchronizeStream接口，阻塞应用程序运行，直到指定Stream中的所有任务都完成。 • 调用aclrtUnSubscribeReport接口，取消线程注册，Stream上的回调函数（CallBackFunc）不再由指定线程（t1）处理。 • 模型推理结束后，调用aclmdlUnload接口卸载模型。

数据后处理	提供样例代码，处理模型推理的结果，直接在终端上显示top5置信度的类别编号。
	另外，样例中提供了自定义接口DumpModelOutputResult，用于将模型推理的结果写入文件（运行可执行文件后，推理结果文件在应用可执行文件的同级目录下），默认未调用该接口，用户可在sample_process.cpp中，在调用OutputModelResult接口前，增加如下代码调用DumpModelOutputResult接口：
	// print the top 5 confidence values with indexes.use function DumpModelOutputResult
	// if want to dump output result to file in the current directory
	processModel.DumpModelOutputResult();
	processModel.OutputModelResult();

目录结构

样例代码结构如下所示。

```
├acl_resnet50_async
├─── caffe_model
|    ├─── resnet50.prototxt       //resnet50网络的模型文件

├─── data
|    ├─── dog1_1024_683.jpg           //测试数据
|    ├─── dog2_1024_683.jpg           //测试数据

├─── inc
|    ├─── memory_pool.h               //声明内存池处理相关函数的头文件
|    ├─── model_process.h             //声明模型处理相关函数的头文件
|    ├─── sample_process.h            //声明资源初始化/销毁相关函数的头文件
|    ├─── utils.h                   //声明公共函数（例如：文件读取函数）的头文件
```



```
├─── script
│    ├─── transferPic.py              //将*.jpg转换为*.bin，同时将图片从1024*683的分辨率缩放为224*224

├─── src
│    ├─── acl.json         //系统初始化的配置文件
│    ├─── CMakeLists.txt          //编译脚本
│    ├─── main.cpp               //主函数，图片分类功能的实现文件
│    ├─── memory_pool.cpp         //内存池处理相关函数的实现文件
│    ├─── model_process.cpp       //模型处理相关函数的实现文件
│    ├─── sample_process.cpp      //资源初始化/销毁相关函数的实现文件
│    ├─── utils.cpp               //公共函数（例如：文件读取函数）的实现文件

├─── .project     //工程信息文件，包含工程类型、工程描述、运行目标设备类型等
├─── CMakeLists.txt     //编译脚本，调用src目录下的CMakeLists文件
```

9.8.2 Ascend RC，编译及运行应用

步骤1 模型转换。

1. 以HwHiAiUser（运行用户）登录开发环境。
2. 参考《**ATC工具使用指导**》中的使用ATC工具转换模型，执行“准备动作”，包括获取工具、设置环境变量。
3. 准备数据。

   从以下链接获取ResNet-50网络的权重文件（*.caffemodel），并以HwHiAiUser（运行用户）将获取的文件上传至开发环境的“acl_resnet50_async/caffe_model”目录下。

   - 从gitee上获取：单击**Link**，查看README.md，查找获取原始模型的链接。
   - 从GitHub上获取：单击**Link**，查看README.md，查找获取原始模型的链接。
4. 将ResNet-50网络转换为适配昇腾AI处理器的离线模型（*.om文件）。

   切换到“acl_resnet50_async”目录，执行如下命令：

   ```
   atc --model=caffe_model/resnet50.prototxt --weight=caffe_model/resnet50.caffemodel --framework=0 --output=model/resnet50 --soc_version=Ascend310 --input_format=NCHW --input_fp16_nodes=data -output_type=FP32 --out_nodes=prob:0
   ```

   --output参数：生成的resnet50.om文件存放在“acl_resnet50_async/model”目录下。建议使用命令中的默认设置，否则在编译代码前，您还需要修改sample_process.cpp中的omModelPath参数值。

   ```
   const char* omModelPath = "../model/resnet50.om";
   ```

   关于各参数的详细解释，请参见《**ATC工具使用指导**》中的“约束及参数说明”。

步骤2 编译代码。

1. 以HwHiAiUser（运行用户）登录开发环境。
2. 切换到“acl_resnet50_async/data”目录下，执行transferPic.py脚本，将*.jpg转换为*.bin，同时将图片从1024*683的分辨率缩放为224*224。在“acl_resnet50_async/data”目录下生成2个*.bin文件。

   ```
   python3.7.5 ../script/transferPic.py
   ```

   说明

   如果执行脚本报错“ModuleNotFoundError: No module named 'PIL'”，则表示缺少Pillow库，请使用**pip3.7.5 install Pillow --user**命令安装Pillow库。
3. 设置环境变量，编译脚本src/CMakeLists.txt通过环境变量所设置的头文件、库文件的路径来编译代码。

编译脚本CMakeLists.txt中已默认设置编译依赖的头文件路径（/usr/local/Ascend/ascend-toolkit/latest/acllib/include/）、库文件路径（/usr/local/Ascend/ascend-toolkit/latest/acllib/lib64/stub/）。

如果未设置${DDK_PATH}、${NPU_HOST_LIB}环境变量，则以编译脚本CMakeLists.txt中的路径为准；如果已设置${DDK_PATH}、${NPU_HOST_LIB}环境变量，则编译脚本CMakeLists.txt优先以环境变量为准，按环境变量指向的路径查找编译依赖的头文件和库文件。

如下为设置环境变量的示例，请将*/home/HwHiAiUser/Ascend/ascend-toolkit/latest*替换为Ascend RC形态的ACLlib安装包的实际安装路径。

```
export DDK_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/arm64-linux
export NPU_HOST_LIB=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/arm64-linux/acllib/lib64/stub
```

说明

使用“*/home/HwHiAiUser/Ascend/ascend-toolkit/latest*/arm64-linux/acllib/lib64/stub”目录下的*.so库，是为了编译基于ACL接口的代码逻辑时，不依赖其它组件（例如Driver）的任何*.so库。

编译通过后，在开发者板上运行应用时，通过配置环境变量，应用会链接到开发者板上“/home/HwHiAiUser/Ascend/acllib/lib64”目录下的*.so库，运行时会自动链接到依赖其它组件的*.so库。

4. 切换到acl_resnet50_async目录，创建目录用于存放编译文件，例如，本文中，创建的目录为“build/intermediates/minirc”。

```
mkdir -p build/intermediates/minirc
```

5. 切换到“build/intermediates/minirc”目录，执行**cmake**生成编译文件。

“../../../src”表示CMakeLists.txt文件所在的目录，请根据实际目录层级修改。

```
cd build/intermediates/minirc
cmake ../../../src -DCMAKE_CXX_COMPILER=aarch64-linux-gnu-g++ -DCMAKE_SKIP_RPATH=TRUE
```

6. 执行**make**命令，生成的可执行文件main在“acl_resnet50_async/out”目录下。

```
make
```

步骤3 运行应用。

1. 以HwHiAiUser（运行用户）将开发环境的“acl_resnet50_async”目录下的src目录、out目录、data目录、model目录上传到开发者板的同一个目录下，例如“/home/HwHiAiUser/acl_resnet50_async”。
2. 以HwHiAiUser用户登录开发者板。
3. 设置环境变量。

如下为设置环境变量的示例，请根据实际安装情况替换路径。

```
export LD_LIBRARY_PATH=/home/HwHiAiUser/Ascend/acllib/lib64
```

4. 切换到可执行文件main所在的目录，例如“/home/HwHiAiUser/acl_resnet50_async/out”，给该目录下的main文件加执行权限。

```
chmod +x main
```

5. 切换到可执行文件main所在的目录，例如“/home/HwHiAiUser/acl_resnet50_async/out”，运行可执行文件。
   - 运行可执行文件，不带参数时：
     - 执行模型异步推理的次数默认为100次；
     - callback间隔默认为1，表示1次异步推理后，下发一次callback任务；
     - 内存池中的内存块的个数默认为100个。
   - 运行可执行文件，带参数时：

- 第一个参数表示执行模型异步推理的次数；
- 第二个参数表示下发callback间隔，参数值为0时表示不下发callback任务，参数值为非0值（例如m）时表示m次异步推理后下发一次callback任务；
- 第三个参数表示内存池中内存块的个数，内存块个数需大于等于模型异步推理的次数。

```
./main
```

执行成功后，在屏幕上提示如下示例信息：

```
[INFO]  ./main param1 param2 param3, param1 is execute model times(default 100), param2 is callback interval(default 1), param3 is memory pool size(default 100)
[INFO]  execute times = 100
[INFO]  callback interval = 1
[INFO]  memory pool size = 100
[INFO]  acl init success
[INFO]  open device 0 success
[INFO]  create context success
[INFO]  create stream success
[INFO]  get run mode success
[INFO]  load model ../model/resnet50.om success
[INFO]  create model description success
[INFO]  init memory pool success
[INFO]  subscribe report success
[INFO]  top 1: index[267] value[0.889648]
[INFO]  top 1: index[161] value[0.836914]
[INFO]  top 1: index[267] value[0.889648]
[INFO]  top 1: index[161] value[0.836914]
[INFO]  top 1: index[161] value[0.836914]
[INFO]  top 1: index[267] value[0.889648]
[INFO]  top 1: index[161] value[0.836914]
[INFO]  top 1: index[267] value[0.889648]
[INFO]  top 1: index[161] value[0.836914]
[INFO]  top 1: index[267] value[0.889648]
......
[INFO]  top 1: index[161] value[0.836914]
[INFO]  top 1: index[267] value[0.889648]
[INFO]  model execute success
[INFO]  unsubscribe report success
[INFO]  unload model success, modelId is 1
[INFO]  execute sample success
[INFO]  end to destroy stream
[INFO]  end to destroy context
[INFO]  end to reset device is 0
[INFO]  end to finalize acl
```

----结束

A 修订记录

发布日期	修改说明
2020-07-30	第一次正式发布。