Atlas 500

软件开发指导书

文档版本	02
发布日期	2019-10-30

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。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： https://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

目 录

1 前言

概述

本文主要介绍如何使用华为Atlas 500进行业务开发。

本文中提供的参考代码仅供开发者参考，禁止作为商业用途。

读者对象

本文档旨在指导Atlas 500开发工程师进行业务开发并且需要具备如下基础知识：

- 对机器学习、深度学习有一定的了解。
- 对图像处理有一定的了解。
- 具备 C/C++语言进行开发程序能力。
- 对Atlas 500架构有一定的了解，相关知识请参见《Atlas 500 用户指南》。

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
危险	表示如不避免则将会导致死亡或严重伤害的具有高等级风险的危害。
警告	表示如不避免则可能导致死亡或严重伤害的具有中等级风险的危害。
注意	表示如不避免则可能导致轻微或中度伤害的具有低等级风险的危害。
须知	用于传递设备或环境安全警示信息。如不避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。

符号	说明
说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

修订记录

修改记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

文档版本	发布日期	修改说明
02	2019-10-30	修改**4.2.2 软件包介绍**。
01	2019-09-30	第一次正式发布。

2 开发前必读

本章节主要介绍使用Atlas 500进行业务开发时需要了解的基础知识、要求和注意事项。

建议开发人员仔细阅读本章节内容，确保了解各项要求和注意事项之后再启动开发。

使用场景

适用于使用Atlas 500进行推理任务的场景。

关键概念

关键概念

概念	解释
Ascend 310	Ascend 310是一款华为专门为图像识别、视频处理、推理计算及机器学习等领域设计的高性能、低功耗AI芯片。 芯片内置2个AI core，可支持128位宽的LPDDR4x，最高可提供16TOPS（Float16/INT8） 的计算能力。
DDK	数字开发套件（Digital Development Kit），DDK是Mind Studio解决方案提供的开发者套件包，Mind Studio通过安装DDK后获得Mind Studio开发必需的API、库、工具链等开发组件。
Graph	graph是HiAI框架中的概念，而非深度学习框架中计算图的概念。graph在HiAI中是指HiAI框架中用于描述整个业务处理流程的图，由多个engine组成，是一个程序处理流程。
HiAI Engine	HiAI Engine是一个通用业务流程执行引擎，主要包含Agent（运行在Host侧）和Manger（运行在Device侧）两个部分。每个engine完成一个由用户代码实现的功能，即engine的处理程序是由用户实现的。
Host侧	Host侧为Hi3559A CPU的操作系统。
Device侧	Device侧为昇腾310侧的操作系统。
DVPP	数字视觉预处理（Digital Vision Pre-Process），提供对特定格式的视频和图像进行解码、缩放等预处理操作，同时具有对处理后的视频、图像进行编码再输出的能力。

概念	解释
AIPP	AI（AI Pre Process）预处理，支持格式转换、Padding/Crop等处理操作。CSC色域转换：YUV2RGB或者RGB2YUV；padding；Scale UP/Down；通道数据交换等
OMG	离线模型生成（Offline Model Generator），用户使用Caffe/TensorFlow等框架训练好的模型，通过OMG将其转换为华为芯片支持的离线模型，实现算子调度的优化，权值数据重排、压缩，内存使用优化等可以脱离设备完成的模型优化功能。
OME	离线模型执行（Offline Model Inference Executor），已经转换完成的离线模型，使用OME进行模型的加载和推理。
Ctrl CPU	一个Ascend 310芯片中有4个Ctrl CPU，主要负责业务逻辑处理。
AI CPU	一个Ascend 310芯片中有4个AI CPU，主要用于算子任务调度、部分算子的实现。
AI Core	一个Ascend 310芯片中有2个AI Core，主要负责矩阵运算。
IPC	IP摄像机，提供RTSP数据流。

3 Atlas 500 产品及软件介绍

Atlas 500是华为面向广泛边缘应用场景的轻量边缘设备，具有超强计算性能、大容量存储、配置灵活、体积小、工作温度范围宽、环境适应性强、易于维护管理等特点。Atlas 500主要应用在智能视频监控、分析、数据存储等场景，可以广泛部署在各类边缘和中心机房，满足在公安、社区、园区、商场、超市等区域的应用需求。

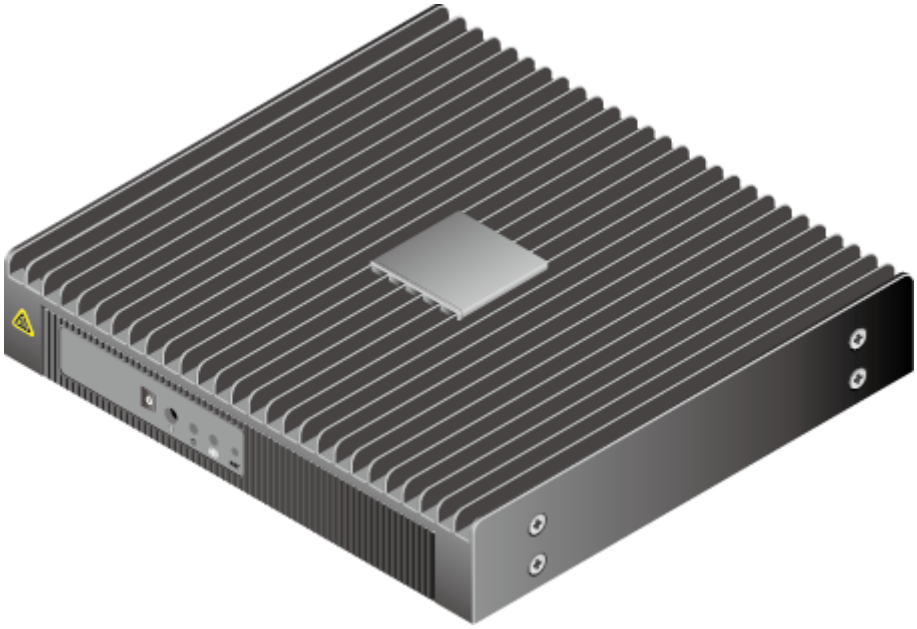
3.1 Atlas 500 硬件介绍

3.1.1 Atlas 500 产品形态

Atlas 500根据硬盘配置，有两种整机形态：

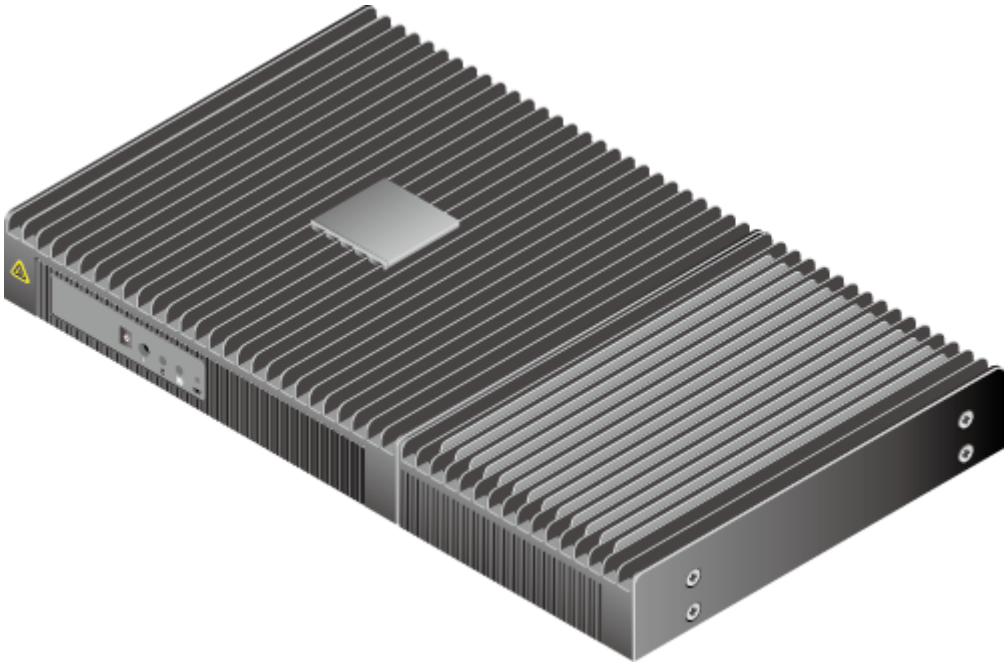
- 当不配置硬盘时，外观如**图3-1**所示。

图 3-1 Atlas 500 外观图（无盘配置）



- 当需要配置3.5英寸硬盘时，在整机右侧扩展硬盘盒，外观如图3-2所示。

图 3-2 Atlas 500 外观图（有盘配置）



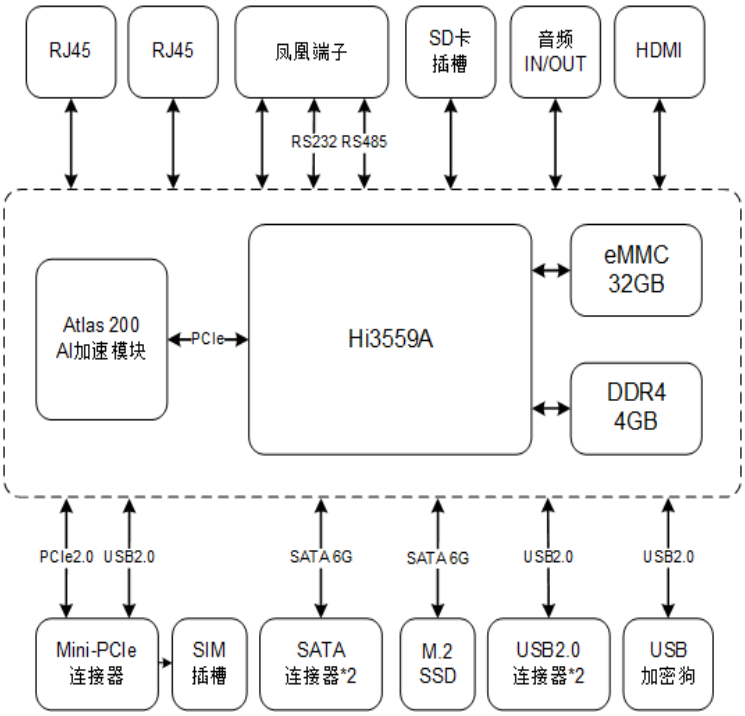
Atlas 500电源规格：

Atlas 500的供电电源为12VDC/5A，推荐使用60W工业级交流电源，接线方式以及推荐的电源型号，请参考《Atlas 500用户指南》。

3.1.2 Atlas 500 硬件系统架构

Atlas 500的系统架构如图3-3所示，处理器为华为自研海思Hi3559A，通过扩展Atlas 200 AI加速模块（可选），最高可提供16T INT8的算力，详情请参考《华为 Atlas 500技术白皮书》。

图 3-3 Atlas 500 系统架构



3.1.3 Atlas 200 AI 加速模块

Atlas 200集成了海思Ascend 310 AI处理器，可实现图像、视频等多种类型的数据分析与推理计算。Atlas 200 AI加速模块的介绍请参考《华为 Atlas 500技术白皮书》。

项目	说明
Atlas 200（可选）	● 2个DaVinci AI Core ● 8核Arm Coretex-A55 1.6GHz CPU ● 乘加计算性能：8TFLOPS/FP16，16TOPS/INT8 ● 内存规格：LPDDR4X，128bit，容量8GB/4GB，接口速率3200 Mbps
Atlas 200编解码能力	● 支持H.264 Decoder硬件解码，16路1080p@30FPS（2路3840 x 2160@60FPS），YUV420 ● 支持H.265 Decoder硬件解码，16路1080p@30FPS（2路3840 x 2160@60FPS），YUV420 ● 支持H.264 Encoder硬件编码，1路1080p@30FPS，YUV420 ● 支持H.265 Encoder硬件编码，1路1080p@30FPS，YUV420 ● JPEG解码能力1080p@256FPS，编码能力1080p@64FPS，最大分辨率：8192 x 8192 ● PNG解码能力1080p 48FPS，最大分辨率：4096 x 4096

3.2 Atlas 500 软件介绍

3.2.1 操作系统介绍

Atlas 500的主控（Hi3559A）上运行了一个基于Linux内核的定制操作系统Euler，用于管理硬件资源。为了管理硬件资源和业务软件，Euler系统内集成了智能管理系统和服务仓等软件，详情请参考《Atlas 智能边缘管理系统 用户指南》。

说明

用户可通过SSH登录到Euler系统的命令行界面，相关的IP地址、用户名和密码请参考《Atlas 智能边缘管理系统 用户指南》中的“默认参数”章节。Euler系统的文件系统和命令与Centos类似，用户可在命令行界面内直接配置网络，查询硬件资源等。

3.2.2 Atlas 200 驱动

Atlas 200作为PCIe从设备，挂载到Euler系统，供业务软件调用。Atlas 200的驱动已经集成到Euler系统，用户无需单独安装与升级。

3.2.3 智能管理系统介绍

Atlas智能边缘管理系统（Intelligent Edge System，简称Atlas IES）主要致力于边缘智能计算硬件的使能与管理，提供安全、易用、可靠的边缘AI硬件平台，使华为Atlas解决方案业务软件或者第三方业务服务软件更简便地使用边缘侧计算能力，支持用户以低成本、安全可靠、灵活多变的方式部署媒体分析处理业务。

Atlas IES支持以下关键特性：

- 支持网络配置
- 支持时间同步
- 支持磁盘分区
- 支持软件安装
- 支持证书管理
- 支持边云协同
- 支持系统维护，包括固件升级、系统重启、日志收集等

关于Atlas IES的详细信息，请参见《Atlas 智能边缘管理系统 用户指南》。

3.2.4 服务仓介绍

华为Atlas 500服务仓（下文统称服务仓）主要是Atlas 500的基础服务，华为发布服务仓基础服务供用户开发使用，从而帮助用户加速构建其解决方案产品。

服务仓主要包括媒体网关服务（MGS）、媒体存储服务（MSS）、媒体中心服务（MCS）、日志管理服务（LMS）、媒体转发服务（MTS）以及DEMO服务车辆检测（VDS）的服务。

服务仓的功能和使用说明请参考《华为 Atlas 500 服务仓 技术白皮书》和《Atlas 500 服务仓 用户指南》。

4 环境准备

4.1 配置 Atlas 500 系统

4.1.1 系统上电

步骤1 根据《Atlas 500 用户指南》中“上电”章节的方法给系统上电。

步骤2 设备通电1-2分钟后，操作系统和设备管理软件启动完成。

步骤3 系统启动完成后，根据《Atlas 500 用户指南 》中的“指示灯和按钮”章节确认Atlas 500前面板指示灯和按钮状态。

- 当使用的是Atlas500（无盘配置）时，安全下电指示灯，健康指示灯均是绿色，绿灯常亮状态表示系统正常。
- 当使用的是Atlas500（有盘配置）时，安全下电指示灯，健康指示灯，硬盘指示灯均是绿色，绿灯常亮状态表示系统正常。否则，系统异常。请参考《Atlas 500 用户指南》。

----结束

4.1.2 确认系统软件版本

根据**4.1.1 系统上电**给系统上电，并且确认系统正常后，进行如下步骤确认软件版本是否正确。

步骤1 准备好一台的PC，给PC添加一个192.168.2.xxx网段的ip地址，“网络掩码”设置为“255.255.255.0”，“网关地址”设置为“192.168.2.1”。

步骤2 使用RJ45接口网线将Atlas 500后面板的GE网口1和PC的网口直连。

步骤3 按照《Atlas 智能边缘管理系统 用户指南》中的“登录Atlas智能边缘管理系统用户界面”章节，登录Atlas 500智能边缘管理系统。

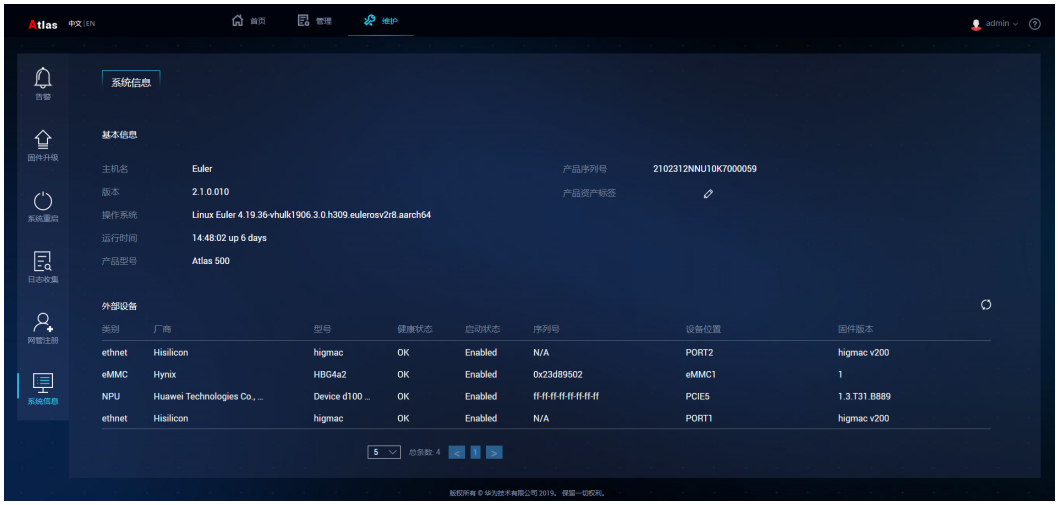
步骤4 点击智能边缘管理系统首页左下角的基本信息栏中按钮“查看更多”进入“系统信息”页面查看系统详细信息。

或者点击智能边缘管理页面的“维护”标签，再点击左侧边栏的按钮“系统信息”查看。

步骤5 在系统信息页面获取系统版本，操作系统版本，NPU驱动版本后，确认是否和获取到的开发软件版本一致。开发软件的获取方法和版本查看方法参考4.2.1 软件版本获取节。

----结束

图 4-1 智能边缘管理系统信息查看



4.1.3 登录和使用 Atlas 500 CLI 管理界面

有以下两种方式可以登录Atlas 500 CLI管理界面

- 使用SSH登录
- 使用串口登录

使用 SSH 登录操作系统

步骤1 准备一台PC，PC上安装SSH客户端软件（PuTTY、Mobaxterm等）。

步骤2 将PC和Atlas 500前面板的GE网口1接入同一个局域网，或者使用网线将Atlas 500直连。

步骤3 在PC上使用ping Atlas 500 IP地址检查Atlas 500和PC网络是否联通

步骤4 在PC上打开SSH客户端软件设置参数登录Atlas 500。

参数示例如下：

- “Host Name”（or IP address）：输入Atlas 500的IP地址。Atlas 500默认ip为“192.168.2.111”。
- “Port”：默认设置为“22”。
- “Connection type”（or protocol）：选择“SSH”。
- “Username”：“admin”

- “Password”：Atlas 500 出厂时admin账户的默认密码为“Huawei12#$”。

  参数设置完成后，启动连接，登录到Atlas 500操作系统。

步骤5 登录完成后，进入Atlas 500 CLI管理界面。命令提示符左侧显示出当前登录设备的主机名。

步骤6 使用Atlas 500 CLI命令。Atlas500 CLI命令和使用方法请参考《Atlas 智能边缘管理系统 用户指南》中的“命令行介绍”章节。

----结束

使用串口登录操作系统

步骤1 准备一台有串口的PC，PC上安装串口客户端软件（PuTTY、Mobaxterm）等，准备好一根凤凰端子转串口线。

步骤2 将凤凰端子转串口线的凤凰端子公头和Atlas 500后面板的凤凰端子母头连接，将串口线和PC连接。根据《Atlas 500 用户指南》中的“后面板”章节查找Atlas 500后面板的凤凰端子母头的位置。

步骤3 打开串口客户端软件，设置登录参数。

参数示例如下：

- “Serial Line to connect to”：“COMn”
- “Speed（baud）”：“115200”
- “Data bits”：“8”
- “Stop bits”：“1”
- “Parity”：“None”
- “Flow control”：“None”

说明

“n”表示不同串口的编号，取值为整数。

参数设置完成后，启动连接。

步骤4 输入用户名和密码。

Atlas 500出厂时默认账号和密码分别为“admin”、“Huawei12#$”。

步骤5 登录完成后，进入到Atlas 500 CLI管理界面。

命令提示符左侧显示出当前登录设备的主机名

步骤6 使用Atlas 500 CLI命令。

Atlas 500 CLI命令和使用方法参照《Atlas 智能边缘管理系统 用户指南》中的“命令行介绍”章节。

----结束

4.1.4 进入 Atlas 500 开发模式

登录Atlas 500 CLI管理界面后，在界面上执行命令**develop**，输入Atlas 500的账户（“root”）和密码（“Huawei@SYS3”）进入Atlas 500开发模式。进入Atlas 500开发模式后，CLI界面显示Euler操作系统CLI界面，此时可以执行Linux通用命令。

说明

建议Atlas 500开发模式只在软件开发调试阶段启用。

4.1.5 配置 Atlas 500 设备网络

如果要将Atlas 500接入到某个非192.168.2.xxx网段的局域网，则需要更改Atls 500的网络。请参考《Atlas 500 用户指南》中“修改初始IP地址”章节的方法设置Atlas 500的网络。

4.2 准备开发软件版本

4.2.1 软件版本获取

Support网站商用版本按照Atlas 200、Atlas 300 AI加速卡（型号 3010）（型号 3000）和Atlas500三种硬件形态分别发布对应版本，请参考如下步骤获取软件和文档。

步骤1 登录华为企业产品技术支持网站。

步骤2 在网站页面上方菜单栏中选择“技术支持 > 产品支持 > 服务器-智能计算 人工智能计算机平台”。

步骤3 在打开的页面中选择“Atlas 500软件”，按需求找到并下载对应软件包。

----结束

4.2.2 软件包介绍

文件列表

请参考表4-1使用对应软件包。

表 4-1 Atlas 500 软件包文件列表

名称	压缩包	说明
Atlas 500软件包文件	Atlas500-ESP.zip	Atlas 500的操作系统固件，开发工具包DDK，自带服务软件包。

包内容介绍

名称	压缩包	说明
Host交叉编译器文件	Euler_compile_env_cross.tar.gz	Atlas 500Host侧（Hi3559A）程序编译使用的编译器。
操作系统源码包文件	Atlas500_EulerOSx.x.x.xxx_64bit_aarch64_basic.tar.gz	Atlas 500Host侧操作系统的内核源码。

名称	压缩包	说明
DDK开发包文件	IES_DDK_Bxxx.tar.gz	Atlas 500开发工具包，包括开发调试工具，头文件，依赖库和Device端编译工具链
系统软件升级包文件	Atlas500-ESP-FIRMWARE-Vx.x.x.xxx.zip	用户可以使用该文件升级Atlas 500操作系统，驱动，智能管理系统软件版本。

说明

以上压缩包名称中的xxx以版本发布后实际获取的包名为准。

4.3 准备文档资料

4.3.1 文档获取方式

步骤1 登录华为企业产品技术支持网站。

步骤2 在搜索框内输入“Atlas 500”，点击“搜索”。

或者在网站页面上方菜单栏中选择“技术支持 > 产品支持 > 服务器-智能计算人工智能计算机平台”。

步骤3 在打开的页面中选择“Atlas 500 > 文档”，按需求找到并下载对应技术文档。

----结束

4.4 获取样例程序

- 获取Sample示例程序
  获取地址：**https://github.com/huaweiatlas/samples**
  使用方法：请参考GitHub各Sample README.md文件。
- 获取Demo程序
  获取地址：**https://github.com/huaweiatlas**
  使用方法：请参考GitHub各Demo README.md文件。

5 Atlas 500 开发环境

本章主要介绍安装后的开发环境，包含头文件、动态库、编译工具、工具等文件分布。

5.1 搭建软件开发环境

按照**4.2.1 软件版本获取**节的方法获取了软件版本，且按照**4.1.2 确认系统软件版本**节确认软件版本一致性后，开始搭建软件开发环境。

步骤1 准备一台x86 CPU的服务器（或者PC）用作开发主机。为开发主机安装好ubuntu 16.04 LTS操作系统（如开发主机已安装好其他版本的操作系统，也可在该主机上安装虚拟机，再安装ubuntu 16.04 LTS操作系统用于开发）。

ubuntu 16.04 LTS操作系统镜像下载地址：**http://old-releases.ubuntu.com/releases/**。

步骤2 配置开发主机网络，确保开发主机和Atlas 500在同一个局域网内。

步骤3 将**4.2.1 软件版本获取**节获取到的软件包文件Atlas500-ESP.zip拷贝到开发主机的某一个目录下。

步骤4 执行命令**unzip Atlas500-ESP.zip**解压“Atlas500-ESP.zip”，得到Euler_compile_env_cross.tar.gz、Atlas500-EulerOSx.x.x.xxx_64bit_aarch64_basic.tar.gz、IES_DDK_Bxxx.tar.gz和Atlas500-ESP-FIRMWARE-Vx.x.x.xxx.zip4个文件。

步骤5 创建Atlas 500 DDK需要安装的目录。

例如：主目录下的Atlas 500_DDK目录，则执行命令**mkdir －p /home/用户名/Atlas500_DDK**来创建安装目录。

步骤6 执行如下命令将Atlas 500 DDK文件安装到创建好的目录。

tar －zxvf IES_DDK_Bxxx.tar.gz －C [Atlas500 DDK安装目录（/home/用户名/Atlas500_DDK）]

IES_DDK_Bxxx.tar.gz的版本号以获取的实际包名为准。

步骤7 执行如下命令将Atlas 500 Host交叉编译工具链安装到已创建的Atlas 500 DDK安装目录。

tar －zxvf Euler_compile_env_cross.tar.gz －C [Atlas500 DDK安装目录（/home/用户名/Atlas500_DDK）]/toolchains

步骤8 执行命令**cd [Atlas500 DDK安装目录（/home/用户名/Atlas500_DDK）]**进入Atlas 500 DDK安装目录。

再执行命令**cat ddk_info**查看安装的Atlas 500 DDK版本，确认DDK版本信息是否正确。

- VERSION：和**4.1.2 确认系统软件版本**节查看的NPU驱动版本一致
- NAME：DDK
- TARGET：ASIC

步骤9 执行命令**vi ~/.bashrc**打开操作系统中当前用户的环境配置文件，并在该文件的结尾添加如下内容：

```
export DDK_HOME= Atlas500 DDK安装目录（/home/用户名/Atlas500_DDK）/
export PATH= $PATH:$DDK_HOME/host/bin/
export LD_LIBRARY_PATH= $LD_LIBRARY_PATH:$DDK_HOME/host/lib/
```

添加完成后，执行命令**source ~/.bashrc**使之生效。

步骤10 如果上述步骤均未报错，并且在操作系统任意目录下执行**omg －h**，输出如下所示表示Atlas 500开发环境搭建成功。否则，Atlas 500开发环境搭建失败，请参考**12 如何获取帮助**。

```
york@huawei-G530-V5:~/Atlas500_DDK$ omg -help
omg: usage: ./omg <args>
example:
./omg --model=./alexnet.prototxt --weight=./alexnet.caffemodel
--framework=0 --output=./domi
aguments explain:
  --model             Model file
  --weight            Weight file. Required when framework is Caffe
  --framework         Framework type(0:Caffe; 3:Tensorflow)
  --output            Output file path&name(needn't suffix, will add .om automatically)
  --encrypt_mode      Encrypt flag. 0: encrypt; -1(default): not encrypt
  --encrypt_key       Encrypt_key file
  --certificate       Certificate file
  --hardware_key      ISV file
  --private_key       Private key file
  --input_shape       Shape of input data. E.g.: "input_name1:n1,c1,h1,w1;input_name2:n2,c2,h2,w2"
  --h/help            Show this help message
  --cal_conf          Calibration config file
  --insert_op_conf    Config file to insert new op
  --op_name_map       Custom op name mapping file
  --plugin_path       Custom op plugin path. Default value is: "./plugin". E.g.:
"path1;path2;path3".
                      Note: A semicolon(;) cannot be included in each path, otherwise the resolved
path will not match the expected one.
  --om                The model file to be converted to json
  --json              The output json file path&name which is converted from a model
  --mode              Run mode. 0(default): model => davinci; 1: framework/davinci model => json;
3: only pre-check
  --target            Target platform. (mini)
  --out_nodes         Output nodes designated by users. E.g.:
"node_name1:0;node_name1:1;node_name2:0"
  --input_format      Format of input data. E.g.: "NCHW"
  --perf_level        Performance level. -1(default): generate a task-sink-model with ub-fuison
and l2-fusion;
                      3: generate task-sink-model without l2-fusion; 4: task-sink-model without ub-
```



```
fusion and l2-fusion.
  --check_report      The pre-checking report file. Default value is: "check_result.json"
  --input_fp16_nodes  Input node datatype is fp16 and format is NCHW. E.g.: "node_name1;node_name2"
  --is_output_fp16    Net output node datatype is fp16 and format is NCHW, or not. E.g.:
"false,true,false,true"
  --ddk_version       The ddk version. E.g.: "x.y.z.Patch.B350"
  --net_format        Set net prior format. ND: select op's ND format preferentially; 5D: select
op's 5D format preferentially
  --output_type       Set net output type. Support FP32 and UINT8
  --fp16_high_prec    FP16 high precision. 0(default): not use fp16 high precision; 1: use fp16
high precision
```

----结束

5.2 Atlas 500 DDK 目录结构

安装好Atlas500 DDK后，Atlas500 DDK安装目录结构如下：

```
.
├─── bin
├─── conf
├─── device
├─── host
├─── include
├─── lib
├─── lib64
├─── packages
├─── sample
├─── scripts
├─── toolchains
    ├─── Euler_compile_env_cross
    ├─── aarch64-linux-gcc6.3
└─── uihost
```

表 5-1 Atlas500 DDK 目录说明

目录名称	目录说明
bin	该目录包含在开发主机上使用的工具，如离线模型转换工具omg。
conf	该目录包含TE算子开发相关头文件。
device	该目录包含Atlas 500 Device侧（昇腾310）的程序依赖的链接库文件，该目录链接到“../lib/aarch64-linux-gcc6.3”目录。
host	该目录包含在开发主机上使用的工具和lib库。
include	该目录包含Atlas 500上运行的软件开发所需要的头文件。
lib	该目录包含Atlas 500 Device侧程序依赖的链接库文件和开发主机上程序依赖的链接库文件。
lib64	该目录包含Atlas 500 Host侧程序依赖的链接库文件。
packages	-
sample	-
scripts	-

目录名称	目录说明
toolchains	该目录包含Atlas 500的交叉编译工具链。
toolchains/Euler_compile_env_cross	该目录包含Atlas 500 Host侧的交叉编译工具链。
toolchains/aarch64-linux-gcc6.3	该目录包含Atlas 500 Device侧交叉编译工具链。
uihost	-

5.3 Atlas 500 程序开发依赖的头文件与链接库

5.3.1 Atlas 500 程序开发的头文件

Atlas 500 Host程序和Device程序在开发时共用一套头文件，所有头文件位于Atlas 500 DDK安装目录下的“include”目录里。“include”目录结构如下：

```
├─── include
│    ├─── inc
│    ├─── libc_sec
│    ├─── log
│    └─── third_party
```

表 5-2 Atlas500 头文件说明

头文件	说明
include/inc	该目录包含Atlas500程序的业务功能头文件。
include/libc_sec	该目录包含安全函数头文件。
include/log	该目录包含日志功能头文件。
include/third_party	该目录包含第三方库头文件。

5.3.2 Atlas 500 程序开发的链接库

Atlas 500程序开发依赖的链接库包括Atlas500 Host侧程序依赖的链接库和Atlas500 Device侧程序依赖的链接库。

- Host侧链接库文件位于Atlas500 DDK安装目录下的“lib64”目录

  lib64目录下无子目录，包含了所有在Atlas 500 host上需要用到的链接库文件及其软链接。
- Device侧链接库文件位于Atlas500 DDK安装目录下的“device/lib”目录

  device/lib目录下无子目录，包含了所有在Atlas 500 device上需要用到的链接库文件及其软链接。

5.4 Atlas 500 软件开发编译工具链

5.4.1 Atlas 500 Host 软件开发编译工具链

Atlas 500 host侧CPU使用的是Hi3559A，Hi3559A是Armv8-a架构的64位CPU。由于Atlas 500没有可以在Hi3559A上运行的编译工具，编译程序时，需要先在x86平台上进行交叉编译，再将编译后的软件拷贝到Atlas 500上的Hi3559A上运行。

Atlas 500 host软件开发编译工具链位于Atlas500 DDK安装目录的“Euler_compile_env_cross/arm/cross_compile/install/bin”路径下。

5.4.2 Atlas 500 Device 软件开发编译工具链

Atlas 500 device侧CPU使用的是昇腾310芯片内的Arm Cortex-A55 CPU。Arm Cortex-A55 CPU是Armv8-a架构的64位CPU。Atlas500没有可以在Arm Cortex-A55 CPU上运行的编译工具，编译程序时，先在x86平台上进行交叉编译，需要将编译后的软件拷贝到Atlas500 host上，程序运行时，再将对应的可执行文件或动态库拷贝到Atlas500 device侧上运行。

Atlas500 device软件开发编译工具链位于Atlas500 DDK安装目录的“toolchains/aarch64-linux-gcc6.3/bin”路径下。

5.5 Atlas500 开发工具

- DDK工具目录：$DDK_HOME/bin/x86_64-linux-gcc5.4
- 工具功能说明请参考表5-3。

表 5-3 工具说明

名称	功能	说明
IDE-daemon-client	IDE Daemon命令工具集	具体说明请参考《Atlas 500 IDE-daemon-client命令参考》。
IDE-daemon-hiai	数据回传工具	● 图片预处理的时候，数据回传。 ● 算子的数据从Device回传到Host侧。
omg	模型转换工具	可将caffe或者tensorflow模型转换为DDK支持的om模型文件，具体可参考《Atlas 500 模型转换指导》。
protoc	第三方库protobuf的工具	用于将proto文件转换为各个语言支持的协议。

6 Atlas 500 业务流程

本章节主要介绍了推理业务的典型业务流程。

6.1 推理软件业务流程

图 6-1 业务流程图

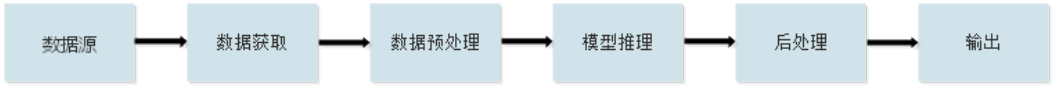


图6-1展示了软件业务流程，具体流程说明请参考表6-1。

表 6-1 业务流程说明

过程	描述	说明
数据源	视频或图片来源	可以使IPC摄像机提供RTSP数据流，或者磁盘离线视频文件，图片文件等。
数据获取	实现数据获取	可以选择开源FFmpeg函数库拉流，或自行实现代码拉流、读文件。运行在HostCPU。
数据预处理	实现图像解码、缩放、色域转换等图片预处理功能	● 选择软解码时，调用开源OpenCV函数库，运行在HostCPU或CtrlCPU。 ● 选择硬解码时，调用DVPP解码，缩放AIPP色域接口，实现预处理，再在Device侧运行对应硬件模块。
推理	实现模型推理功能	运行在Device侧AI Core和AI CPU。

过程	描述	说明
后处理	实现模型结果后处理	可以选择运行在Device侧Ctrl CPU或Host CPU。
输出	实现结果呈现	-

6.2 软件模块与硬件模块对应关系示例

- 预处理和后处理，可以根据实际情况，灵活选择在Host侧或者Device侧执行。
- 预处理可以灵活选择使用硬件实现或软件实现。

图 6-2 软件模块图

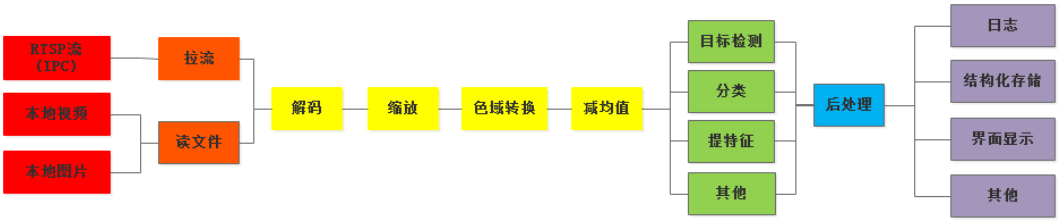
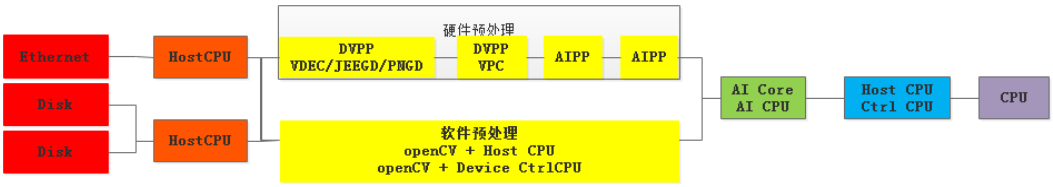


图 6-3 硬件模块图



7 HelloDavinci 程序

本章节主要介绍入门程序HelloDavinci。

用户可以通过这个入门样例了解基于Atlas产品的应用开发的过程，同时验证编译环境和运行环境等配置的正确性。

7.1 HelloDavinci 代码获取

示例代码获取地址：**https://github.com/huaweiatlas/samples**

7.2 HelloDavinci 说明

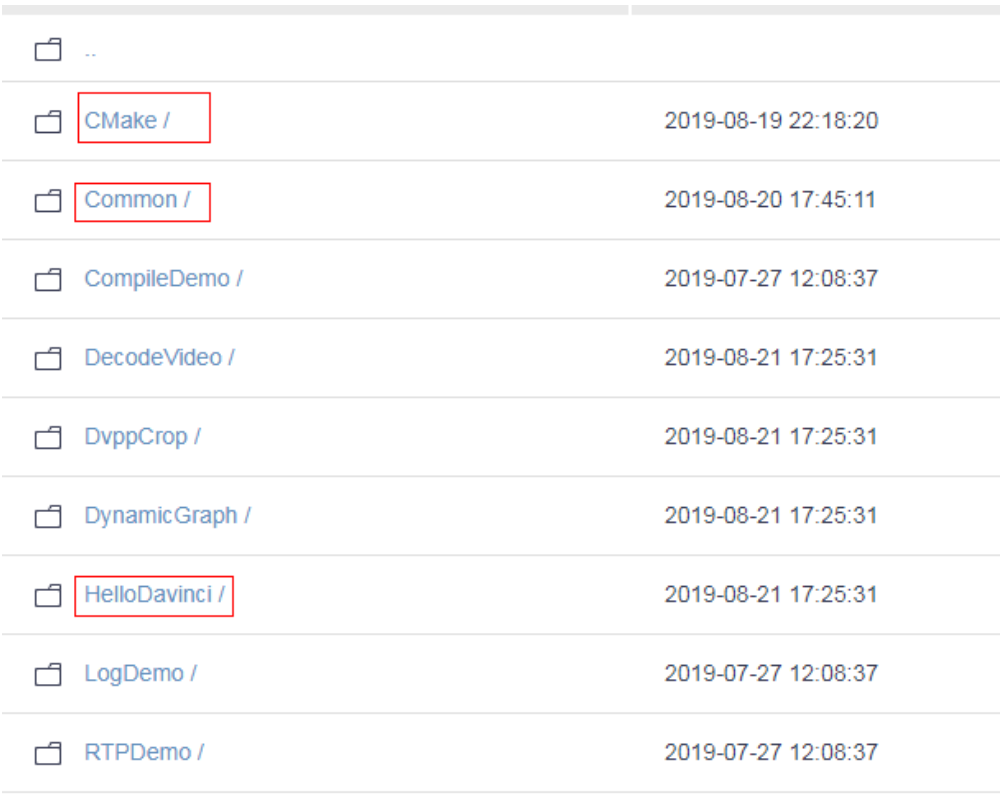
获取的文件samples包含了编译配置和Atlas各个程序样例（HelloDavinci在其中）。如果单独运行HelloDavinci，需要Samples/Cmake， Samples/Common和Samples/HelloDavinci三个文件夹，如**图7-1**所示。

- CMake：存放cmake配置文件。
- Common：存放公共代码。
- HelloDavinci：HelloDavinci工程目录，主要包括build文件、源码文件、graph配置文件及README.md。

说明

请保持这三个文件的相对路径不变。

图 7-1 文件目录



编译工具文件Cmake目录结构如下所示：

```
├———Ascend.cmake    //Device侧编译链
├———Euler.cmake     //Host侧编译链
└———FindDDK.cmake   //cmake寻找DDK模块
```

HelloDavinci的目录结构如下所示：

```
├——— build              //编译文件夹，包括Host和Device侧的编译
|    ├——— CMakeLists.txt
|    ├——— device
|    └——— host
├——— build.sh           //编译脚本
├——— README.md          //README.md
├——— main.cpp           //主函数入口
├——— Common             //HelloDavinci公共模块
|    └——— include
├——— DstEngine          //DstEngine(host侧)
|    ├——— DstEngine.cpp
|    └——— DstEngine.h
├——— graph.config       //graph配置文件
├——— HelloDavinci       //HelloDavinci Engine(device侧)
|    ├——— HelloDavinci.cpp
|    └——— HelloDavinci.h
└——— SrcEngine          //SrcEngine Engine(host侧)
├——— SrcEngine.cpp
└——— SrcEngine.h
```

7.3 HelloDavinci 流程框架

本章节介绍HelloDavinci样例代码的流程。

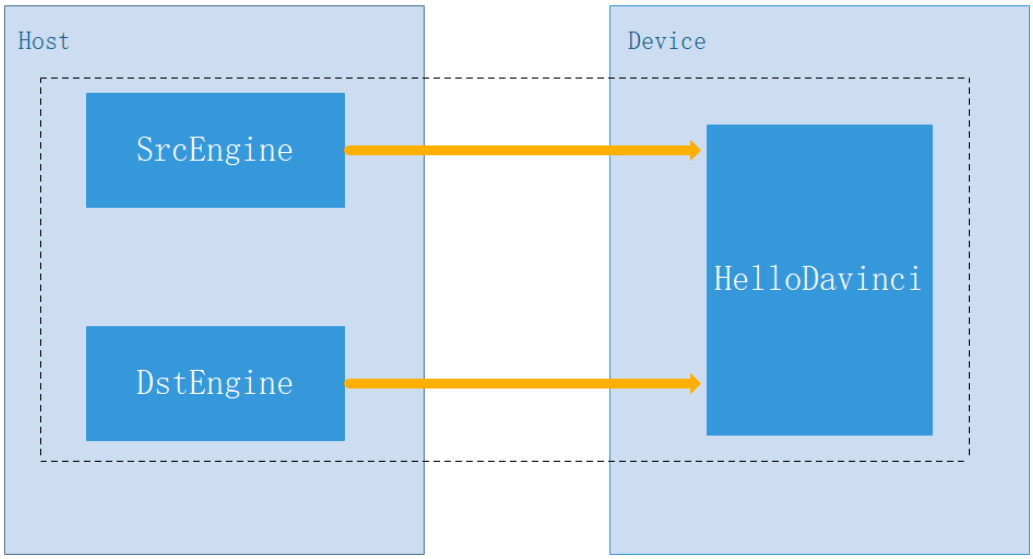
说明

如无需了解本章节内容可跳过本节到**7.4 HelloDavinci编译运行**直接进行编译运行，查看运行结果。

本开发样例主要是演示从Host侧发送数据到Device侧，再从Device侧获取生成的字符串发送回Host侧，保存结果，并且打印到终端。如图7-2所示，整个程序分为两部分运行，Host侧（包括SrcEngine和DstEngine）和Device侧（包括HelloDavinci），运行过程如下：

1. 运行从main开始，向SrcEngine发数据。
2. SrcEngine收到数据之后，转发给HelloDavinci，HelloDavinci在内部生成字符串“This message is from HelloDavinci”并发送到DstEngine。
3. DstEngine收到数据后将信息保存在目录“${workPath}/out/dacvinci_log_info.txt($”{workPath}为工程根目录)下，并向main发送结束信号。
4. main函数收到结束信号后，销毁graph，在终端打印结束信息并退出程序。

图 7-2 HelloDavinci 流程框架图



7.4 HelloDavinci 编译运行

前提条件

- 已安装Atlas 500 DDK。
- 已安装第三方CMake编译工具（2.8.4版本以上）。
  若无CMake编译工具，请参考**8.4.2 使用CMake编译Atlas500软件**安装。

编译步骤

步骤1 执行命令**export DDK_HOME=[Atlas500 DDK安装目录]**。

步骤2 进入HelloDavinci目录。

步骤3 执行命令**bash build.sh A500**进行编译，编译成功后将在“${workPath}/out/”目录下生成可执行文件main和目标动态库文件。

步骤4 将out文件夹拷贝到Atlas 500上，执行命令**./out/main**查看结果输出，如返回如下结果，则表示运行成功。

```
Euler:~ # ./out/main
Hello Davinci!
The sample end!
Euler:~ # cat out/davinci_log_info.txt
This message is from HelloDavinci
```

----结束

8 业务开发

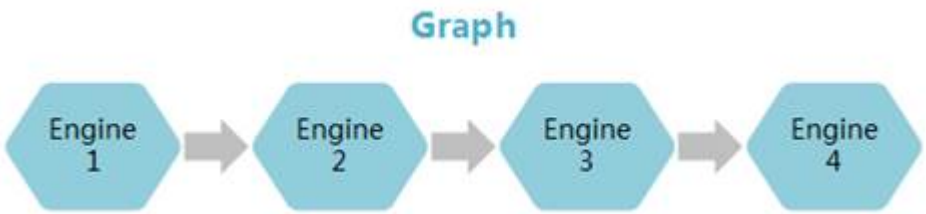
8.1 业务软件框架—Matrix

业务框架

为了高效使用Ascend 310芯片的算力，提供了Matrix框架来完成推理业务迁移，Matrix框架提供的功能如下：

- 流程编排：
  a. 定义Engine为流程的基本功能单元，同时允许用户自定义Engine的实现（输入图片数据、对图片进行分类处理、输出对图片数据的分类预测结果等）。每个Engine在Ascend 310端默认对应一个线程来运行处理。
  b. 定义关于Graph管理若干Engine的流程。每个Graph在Ascend 310端对应一个进程运行处理。Graph与Engine关系如**图8-1**所示。

  图 8-1 Graph 与 Engine 关系

  在Graph配置文件中配置Engine节点间的串接和节点属性（运行该节点所需的参数），节点间数据的实际流向根据具体业务在节点中实现，通过向业务的开始节点灌入数据启动整个Engine计算流程。
- 媒体预处理：

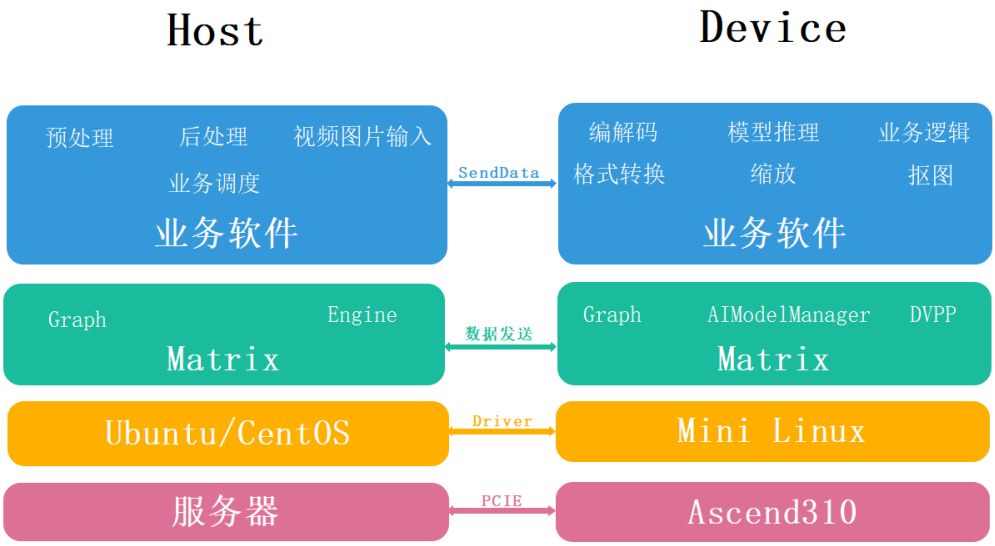
运行在Ascend 310上的Engine可直接调用DVPP提供的API接口实现媒体预处理能力。

- 离线模型加载和运行：

  运行在Ascend 310上的Engine可直接调用模型管家（AIModelManger）提供的API进行离线模型加载和推理功能。

  基于Matrix框架，用户的业务软件结构如下图所示，用户创建自定义的Engine组成业务流（Graph），运行在Device侧的Engine可以调用DVPP和AIModelManager的API，使用Ascend 310的媒体预处理和模型推理的硬件加速功能。Host侧的Engine主要实现业务软件逻辑及与Device侧Engine间的数据传输功能。

图 8-2 业务软件框架



8.1.1 Graph 配置、创建与销毁

Graph主要功能是描述业务包含的Engine及Engine间的数据传输关系，Matrix框架通过Protobuf定义了Graph的数据结构，用户以配置文件的方式来定义Graph配置。

说明

Graph关键字含义请参考**A Graph关键字**，详情请参考文档《Atlas 500 Matrix API参考》。

下文提供了一个简单的graph配置文件，该graph文件会创建一个id为1000的graph业务流，该业务流包含三个Engine，Engine间的数据传输关系如**图8-3**所示。

图 8-3 Engine 间的数据传输关系


```
graphs {
    graph_id: 1000              # 单芯片支持多graph，grpah id需大于0，同芯片不能重
复
    device_id: "0"              # graph运行的芯片id
    priority: 1                 # 优先级，
```

```
    # 用户自定义Engine，可实例化多个Engine，以id区分
    engines {
        id: 2001                                        # 引擎ID
        engine_name: "ObjectDetectionEngine"            # 用户自定义Engine的类名
        side: DEVICE                                    # 指定引擎运行在Host/
Device侧
        so_name: "./libObjectDetectionEngine.so" # 依赖动态连接库名称及在Host
路径，Matrix框架拷贝到Device侧
        # 用户可自定义配置参数
        ai_config{
            items{
                name: "model"                       # 模型名称
                value: "./FaceDetection.om"         # 模型路径在Host的路径，
Matrix框架拷贝到Device侧
            }
            items{
                name: "mode"
                value: "test"
            }
        }
    }
    engines {
        id: 1000
        engine_name: "DecodeEngine"
        side: DEVICE
        so_name: "./libDecodeEngine.so"
    }
    engines {
        id: 2002
        engine_name: "PostProcess"
        side: HOST
    }

    # 描述Engine间的端口连接
    connects {
        src_engine_id: 1000                         #数据传输源引擎ID
        src_port_id: 0                              #数据传输源端口号
        target_engine_id: 2001                      #数据传输目的引擎ID
        target_port_id: 0                           #数据传输目的端口号
    }
    connects {
        src_engine_id: 2001
        src_port_id: 0
        target_engine_id: 2002
        target_port_id: 0
    }
}
```

Matrix提供的Graph接口，常用的三个接口如表8-1所示，详细介绍请参考文档《Atlas 500 Matrix API参考》。

表 8-1 接口说明

接口	说明
HIAI_StatusT HIAI_Init(uint32_t deviceID)	Ascend 310芯片初始化，请注意此处的芯片ID编号是Ascend 310芯片的绝对编号，在Atlas 500上始终为“0”，npu-smi查询的芯片ID是相对编号。

接口	说明
static HIAI_StatusT Graph::CreateGraph(const std::string& configFile)	读取Graph配置文件，初始化Engine，创建线程和数据传输通道，完成业务流初始化。
static HIAI_StatusT Graph::DestroyGraph(uint32_t graphID)	销毁Graph，运行Engine的析构函数等。

8.1.2 Engine

Engine是Matrix框架定义的业务软件基本功能单元，用户可继承Matrix定义的Engine模板类，创建业务中各功能模块的Engine（读取输入文件，图像预处理，神经网络推理，推理结果后处理，host/device数据传输等），详情请参考文档《Atlas 500 Matrix API参考》。每一个Engine定义了函数Init()和Process()，在Graph初始化时，会自动运行Init()，从而实现Engine的参数初始化（包含内存分配和模型加载）。Process()接口实现数据的传输和业务逻辑。

- 常用接口介绍如表8-2所示。

表 8-2 接口说明

接口	描述	说明
`HIAI_DEFINE_PROCESS（inputPortNum, outputPortNum)`	端口（Engine的数据输入和输出数据通道）由此接口指定	Matrix框架为每一端口创建了队列用于缓存数据，Engine间端口的传输关系在Graph配置文件内指定，请参考**A Graph关键字**。
`HIAI_StatusT Engine::Init(const AIConfig &config, const vector<AIModelDescription>&modelDesc)`	Engine初始化	在创建Graph时此接口会被调用，初始化Engine，Graph配置文件中定义的ai_config会传给函数入参config，用户可在配置文件内添加自定义item，并在初始化函数内使用此item。
`HIAI_IMPL_ENGINE_PROCESS(name, engineClass, inPortNum)`	Engine的Process，在Device侧对应线程	由数据驱动，即输入端口收到数据后，由框架启动Process运行，如果设置了多个输入端口，每一个输入端口接收到数据后，都会触发一次Process运行，因此如果业务处理依赖多个输入，用户需自行实现多输入的同步逻辑。框架已将Process封装成宏定义，用户实现该宏定义即可。

- Eingine读取输入端口的数据，框架支持最大16个输入端口（arg0~arg15），用户可直接使用，从业务应用角度看，数据传输的是共享指针，传输代码示例如下：

```
// 发送Engine：传输自定义数据USER_DEFINE_TYPE到目标engine
std::shared_ptr<USER_DEFINE_TYPE > streamData= std::make_shared<
USER_DEFINE_TYPE >();
// 对streamData进行赋值后，调用Senddata发送，发送需要将共享指针转换为void类型
```

```
hiai::Engine::SendData(0, " USER_DEFINE_TYPE ",
std::static_pointer_cast<void>(deviceStreamData));
// 接收Engine：结构数据，并将数据转换为用户自定义类型USER_DEFINE_TYPE
std::shared_ptr< USER_DEFINE_TYPE > inputArg =
std::static_pointer_cast< USER_DEFINE_TYPE >(arg0);
```

8.1.3 数据传输

Matrix框架定义的数据传输，根据业务应用分为以下三类，接口类似，传输的对象都为共享指针，但是根据场景不同，故接口使用方式有明显区别。

- Graph外传输数据到engine的输入端口：请参考文档《Atlas 500 Matrix API参考》中的“Graph::SendData”章节。
- Graph内的engine输出端口传输数据到graph外：请参考文档《Atlas 500 Matrix API参考》中的“Graph::SetDataRecvFunctor”和“Engine::SetDataRecvFunctor”章节。
- Graph内engine间的数据传输：请参考文档《Atlas 500 Matrix API参考》中的“Engine::SendData”章节。

Engine 间的数据传输

Matrix框架将业务软件分为Host侧（X86/ARM服务器）和Device侧（Ascend 310芯片）两部分软件，因此Enigine间的数据传输分为跨侧传输和同侧传输，说明如下：

- 跨侧传输：传输的数据需要序列化为二进制，通过PCIE或DMA等硬件完成数据传输后，反序列为有效数据。因此对于自定义数据结构，需要用户自定义序列化与反序列化函数，Matrix框架主要提供两种序列化/反序列函数定义方式，分别为普通接口和高速接口，普通接口适合256K以下的数据传输，高速接口适合256K以上的数据传输（高速接口传输小内存块，速度与普通接口类似），详情请参考《Atlas 500 Matrix API参考》中的“数据类型序列化和反序列化（C++语言）”章节。
- 同侧传输：传输的数据即为共享指针的地址，并未拷贝，由于Engine是线程运行，该方式即为线程间用共享内存的方式，实现数据传输，需要用户保证不出现非法访问。Matrix框架推荐用于传输的共享指针，在传输给下一个Engine后，本Engine不再修改共享指针。
- 跨侧传输在实现序列化与反序列化函数之后，在业务应用的上采用的接口与同侧传输一致。

```
HIAI_StatusT Engine::SendData(uint32_t portId, const std::string&
messageName,
const shared_ptr<void>& dataPtr,  uint32_t timeOut = TIME_OUT_VALUE);
```

Graph 外传数据到 engine 的输入端口

- 由Engine组成的业务流，由数据驱动，数据从Graph外传输到Graph内，即通过该接口实现。
- 以下接口可实现跨侧传输和同侧传输，要求与Eingine间的数据传输一致，详情请参考文档《Atlas 500 Matrix API参考》中的“Graph::SendData”章节。

```
HIAI_StatusT Graph::SendData(const EnginePortID& targetPortConfig,
const
std::string& messageName, const std::shared_ptr<void>& dataPtr,
const uint32_t
timeOutMs = 500)
```


Graph 内的 Engine 输出端口传输数据到 Graph 外

- Matrix采用了以下两种设置回调函数的方式将Engine输出端口的数据传输到graph外；
  - 业务流节点Engine输出端口回调，请参考《Atlas 500 Matrix API参考》中的“Graph::SetDataRecvFunctor”章节。
  - Engine输出端口回调，请参考《Atlas 500 Matrix API参考》中的“Engine::SetDataRecvFunctor”章节。
- 框架提供了回调函数的模板类---DataRecvInterface。
- 框架要求回调函数与输出Engine运行于同侧，即支持同侧传输，不支持跨侧传输。

8.2 DVPP 接口

DVPP是Ascend 310芯片提供的图像预处理硬件加速模块，该模块集成的六个功能如下所示，接口介绍和使用方法请参考《Atlas 500 DVPP API参考》。

- 格式转换，抠图与缩放（VPC）
- H264/H265视频解码（VDEC）
- H264/H265视频编码 （VENC）
- Jpeg图片解码（JPEGD）
- Jpeg图片编码（JPEGE）
- Png图片解码（PNGD）

8.2.1 DVPP 接口使用

DVPP采用句柄方式提供接口，主要包含三类接口：创建，使用，销毁。

VPC、JPEGE、JPEGD、PNGD共用一套接口，输入的参数有差异。VDEC和VENC各使用一套接口。

- VDEC解码使用的三个接口如下所示，用户以异步方式调用，使用VdecCtl接口传入配置（包含回调函数）、H264/H265数据等，硬件解码后，Matrix框架调用回调函数将结果返回。

  说明

  VDEC解码后的数据为HFBC（内部格式），用户需要使用VPC将HFBC转换为YUV420SP格式，详情请参考《Atlas 500 DVPP API参考》中的“VDEC功能接口”章节中的相关示例。

```
int CreateVdecApi(IDVPPAPI *&pIDVPPAPI, int singleton)
int VdecCtl(IDVPPAPI *&pIDVPPAPI, int CMD, dvppapi_ctl_msg *MSG, int singleton)
int DestroyVdecApi(IDVPPAPI *&pIDVPPAPI, int singleton)
```

- VPC、JPEGE、JPEGD、PNGD使用如下所示相同的接口，不同功能传输的配置参数不同，请用户参考文档《Atlas 500 DVPP API参考》，VPC参考章节“VPC新接口”，JPEGE参考章节“JPEGE”，JPEGD参考章节“JPEGD”，PNGD参考章节"PNGD"。

```
int CreateDvppApi(IDVPPAPI *&pIDVPPAPI)
int DvppCtl(IDVPPAPI *&pIDVPPAPI, int CMD, dvppapi_ctl_msg *MSG)
int DestroyDvppApi(IDVPPAPI *&pIDVPPAPI)
```


- 由于硬件限制，DVPP使用过程中存在部分限制。为了加快读写速度，图片长宽需要对齐到指定大小，但不影响有效区域，采用向左、向下填充0的方式，对齐到指定大小。

  例如：对于300*300的YUV420SP_UV图片，需要对齐到304*300（宽要求16对齐，高要求2对齐），有效区域仍为[0, 0]至[300, 300]，需要用户在图片右侧补零，对齐到304列。

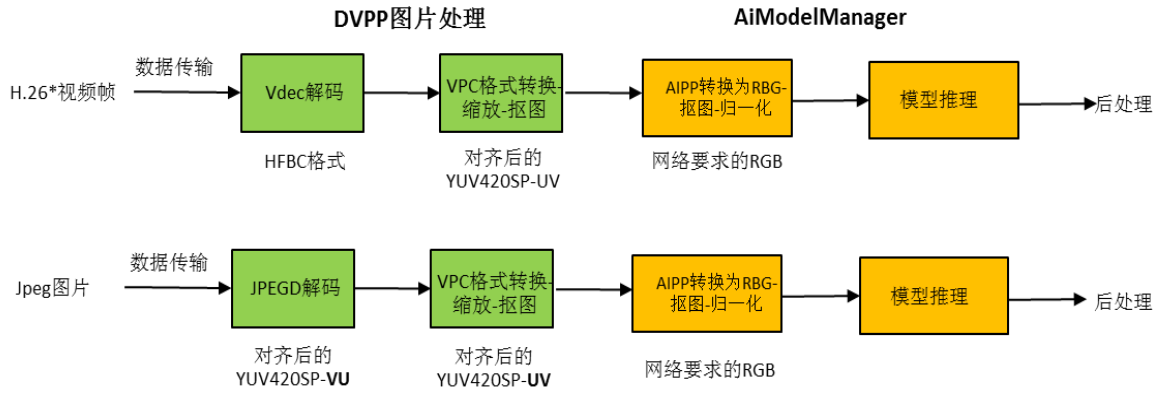
  - 当用户使用DVPP的JPEGD、VDEC、PNGD组件读取输入图片时，解码后的图片需要满足长宽对齐要求（实际长宽），此时用户需要注意，以对齐后的图片大小申请输出图片内存。

    例如：对于300*300的YUV420SP_UV图片，需要申请304*300*3/2Byte（YUV420SP一个像素需要1.5Byte存储）。

    - VPC输入和输出内存地址，16字节对齐。
    - VPC输出图片的宽度，16字节对齐。
    - VPC输出图片的高度，2字节对齐。
    - VPC输入图片的宽度，16字节对齐。
    - VPC输入图片的高度，2字节对齐。
    - JPEGD输出图片的宽度，128字节对齐。
    - JPEGD输出图片的高度，16字节对齐。
  - DVPP各组件基于处理速度和内存占用量的考虑，对输出图片有诸多限制，如输出图片需要长宽对齐，输出格式必须为YUV420SP等，但模型输入通常为RGB或BGR，且输入图片尺寸各异。因此，Acsend310芯片提供AIPP（Ai Preprocess），用于图片格式转换及抠图功能，具体说明请参考《Atlas 500 模型转换指导》。

    例如：Jpeg图片输入和H26*视频输入的处理流程如图8-4所示。

图 8-4 视频与图片输入处理流程



8.2.2 DVPP 的内存申请

Matrix针对DVPP提供了内存申请接口HIAI_DVPP_DMalloc和释放接口HIAI_DVPP_DFree，申请接口用于申请满足DVPP对齐要求的内存。用户在使用时注意该接口需要成对使用，为有效预防内存泄露问题，推荐使用共享指针管理申请的内存，实现代码如下：

```
uint8_t* buffer = nullptr;
HIAI_StatusT ret = hiai::HIAIMemory::HIAI_DVPP_DMalloc(dataSize,
(void*&) buffer);
```



```
std::shared_ptr<uint8_t> dataBuffer =  std::shared_ptr<uint8_t>( buffer,
\
[](std::uint8_t* data) hiai::HIAIMemory::HIAI_DVPP_DFree(data);});
```

此外，该接口仅在Device侧使用，Host侧无该接口。HIAI_DVPP_DMalloc申请的内存可用于Device到Host的高速数据传输，在此方式下，内存由Matrix框架释放，详情请参考sample代码-DvppDecodeResize。

8.3 离线模型推理

Ascend 310芯片有利于加速Caffe和Tensorflow框架的推理。用户在完成训练后，需要先将训练后的模型转换成Ascend 310支持的模型文件（om文件），再编写业务代码，然后调用Matrix框架提供的接口完成业务功能。

Matrix框架将AIPP（Ai Preprocess）和模型推理功能封装至一个模块内， 用户调用推理接口后，Matrix框架会先调用AIPP完成输入图片的预处理，再将预处理后的图片输入到模型推理模块内，最后返回推理后的结果。

8.3.1 AIPP 配置

AIPP是Ascend 310提供的硬件图像预处理功能，包括色域转换，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）等功能。

AIPP区分为静态AIPP和动态AIPP：

- 静态AIPP：在模型转换时设置参数，模型推理过程采用固定的AIPP预处理（无法修改），静态AIPP的关键字含义和配置文件模板请参考《Atlas 500 模型转换指导》中的“配置文件模板”章节。
- 动态AIPP：在模型转换时，设置AIPP为动态模式，每次模型推理前，根据需求，设置AIPP预处理参数。动态AIPP在根据业务要求改变预处理参数的场合下使用（如不同摄像头采用不同的归一化参数，输入图片格式需要兼容YUV420和RGB等）。动态AIPP的使用方法请参考《Atlas 500 Matrix API参考》中的“AIPP配置接口”章节。

说明

- AIPP的输入格式为“YUV420SP_U8”（默认为“YUV420SP_UV”格式），若格式为“YUV420SP_VU”，请修改参数“rbuv_swap_switch”，否则会影响输出结果。
- 模型输入为“RGB_U8”或“BGR_U8”，对应不同的色域转换矩阵。

DVPP模块输出的图片多为对齐后的YUV420SP类型，不支持输出RGB图片。因此，业务流需要使用AIPP模块转换对齐后YUV420SP类型图片的格式，并抠出模型需要的输入图片。

例如：模型要求输入为300*300的RGB图片，用户调用DVPP接口处理（如Jpeg解码和缩放）后，由于对齐要求，输出的图片为384*304（图片有效区域为300*300，向左向下填充零补齐）的YUV420SP_UV格式图片。

静态AIPP的配置如下所示，文件配置了抠图的起始坐标，抠图的长宽默认为模型输入，图像归一化参数，即均值和方差的倒数（减均值后，乘以该系数）。

```
aipp_op{
# AIPP为静态模式
aipp_mode: static
# 使能抠图
crop: true
# 输入图片的格式及大小
```



```
input_format : YUV420SP_U8
src_image_size_w : 384
src_image_size_h : 304
# 抠图的起始坐标，抠图的宽与高默认为模型输入
load_start_pos_h : 0
load_start_pos_w : 0

# 使能格式转换，该转换矩阵将YUV420SP_UV转换为RGB888
csc_switch : true
matrix_r0c0 : 298
matrix_r0c1 : 516
matrix_r0c2 : 0
matrix_r1c0 : 298
matrix_r1c1 : -100
matrix_r1c2 : -208
matrix_r2c0 : 298
matrix_r2c1 : 0
matrix_r2c2 : 409
input_bias_0 : 16
input_bias_1 : 128
input_bias_2 : 128

#开启数据归一化，配置均值和方差的倒数
mean_chn_0 : 125
mean_chn_1 : 125
mean_chn_2 : 125
var_reci_chn_0 : 0.0039
var_reci_chn_1 : 0.0039
var_reci_chn_2 : 0.0039
}
```

8.3.2 离线模型转换

离线模型转换工具包含在DDK工具包内，位于目录“<$DDK_HOME>/uihost/bin/omg”下，omg为命令行工具（可通过**-h**获得参数信息），支持模型加密，用于实现caffe和tensorflow模型转换成Ascend 310支持的om文件。Omg工具的使用说明请参考《Atlas 500 模型转换指导》中的“使用omg命令转换模型”章节。

1. caffe模型的转换：
   ```
   #omg --framework 0 --model <model.prototxt> --weight
   <model.caffemodel> --output <output name> --insert_op_conf <aipp.cfg>
   ```
2. tensorflow模型的转换：
   ```
   #omg --framework 3 --model <model.pb> --input_shape “input_name:
   1,112,112,3” --output <output_name> --insert_op_conf <aipp.cfg>
   ```

表 8-3 参数说明

参数	说明
framework=0	指定为caffe模型，。
framework=3	指定为tensorflow模型。
model	指定模型文件。
weight	指定caffe的权重文件。
output	支持输出om文件的文件名。

参数	说明
input_shape	指定输入层的名字和大小，tensorflow默认为“input_layer_name: n, h, w, c”。
insert_op_conf	指定AIPP的配置文件。

8.3.3 模型推理接口

Matrix框架提供AIModelManager类，实现模型加载和推理功能，详情请参考《Atlas 500 Matrix API参考》。

模型推理初始化

步骤1 在自定义推理模型Enigine的graph配置文件内设置模型路径（在ai_config内，添加items，设置模型在host路径）。

步骤2 使用Matrix框架将模型文件传输到Device侧。

步骤3 用户在自定义Engine内解析自定义items，获得模型在Device侧的路径。

步骤4 调用AIModelManager::Init()完成初始化，具体实现如下：

```
/* 用户自定义Engine的成员变量 */
std::shared_ptr<hiai::AIModelManager> modelManager;
/* 自定义Engine Init 函数内实现 AIModelManager 初始化 */
std::vector<hiai::AIModelDescription> model_desc_vec;
hiai::AIModelDescription model_desc_;
......
/* 从graph配置文件内，ai_config结构内，解析出，模型路径 */
model_desc_.set_path(model_path);// 设置模型路径
model_desc_.set_key("");// 如om文件加密，设置
model_desc_vec.push_back(model_desc_);
ret = modelManager->Init(config, model_desc_vec);// config无意义，将Engine::Init的入参传入即可
```

----结束

设置模型推理输入与输出

- Matrix框架定义IAITensor类，用于管理模型推理的输入与输出矩阵。为了便于使用，Matrix框架基于IAITensor，派生出了AISimpleTensor和AINeuralNetworkBuffer。
- 模型推理的输入和输出采用HIAI_DMalloc接口申请，可减少一次内存拷贝。
- Matrix框架可自动释放的内存AISimpleTensor管理的内存，建议由用户申请和释放，防止出现内存泄露或重复释放。
- 模型转换过程中，如启用了AIPP的抠图、格式转换、图片归一化等功能，则输入数据需经AIPP模块处理，再将处理后的数据进行真正的模型推理。

输入与输出的具体实现

输入输出的具体实现代码如下：

```
/* 获取推理模型的输入输出Tensor描述 */
std::vector<hiai::TensorDimension> inputTensorDims;
```



```
std::vector<hiai::TensorDimension> outputTensorDims;
ret = modelManager->GetModelIOTensorDim(modelName, inputTensorDims,
outputTensorDims);

/* 设置输入，多个输入，依次创建并设置 */
std::shared_ptr<hiai::AISimpleTensor> inputTensor =
std::shared_ptr<hiai::AISimpleTensor>(new hiai::AISimpleTensor());
inputTensor->SetBuffer(<输入数据的内存地址>, <输入数据的长度>);
inputTensorVec.push_back(inputTensor);

/* 设置输出 */
for (uint32_t index = 0; index < outputTensorDims.size(); index++) {
hiai::AITensorDescription outputTensorDesc =
hiai::AINeuralNetworkBuffer::GetDescription();
uint8_t* buf = (uint8_t*)HIAI_DMalloc(outputTensorDims[index].size);
......
std::shared_ptr<hiai::IAITensor> outputTensor =
hiai::AITensorFactory::GetInstance()->CreateTensor(
outputTensorDesc, buf, outputTensorDims[index].size);
outputTensorVec.push_back(outputTensor);
}
```

模型推理

- Matrix框架提供了同步推理和异步推理两种方式，默认为同步推理，可通过设置AIContext配置项，使用回调函数实现异步推理。

  ```
  /* 模型推理 */
  hiai::AIContext aiContext;
  HIAI_StatusT ret = modelManager->Process(aiContext, inputTensorVec,
  outputTensorVec, 0);
  ```

- AIModelManager对象加载多个模型，可通过设置AIContext配置项，指定模型（初始化阶段，指定的模型名），详情请参考《Atlas 500 Matrix API参考》中的“离线模型管家”章节。

模型推理后处理

模型推理的结果矩阵，以内存+描述信息的方式保存在IAITensor对象内，用户需要根据模型的实际输出格式（数据类型和数据顺序），将内存解析成有效的输出。

```
/* 推理结果解析 */
for (uint32_t index = 0; index < outputTensorVec.size(); index++) {
shared_ptr<hiai::AINeuralNetworkBuffer> resultTensor =
std::static_pointer_cast<hiai::AINeuralNetworkBuffer>(outputTensorVec[i])
;
// resultTensor->GetNumber()  -- N
// resultTensor->GetChannel() -- C
// resultTensor->GetHeight()  -- H
// resultTensor->GetWidth()   -- W
// resultTensor->GetSize()    -- 内存大小
// resultTensor->GetBuffer()  -- 内存地址
}
```

常见的分类模型后处理可参考示例代码“InferClassification”，SSD目标检测模型后处理可参考示例代码“InferObjectDetection”。

8.4 Atlas 500 软件编译

一个在Atlas 500上运行的完整程序一般可以分为两部分，一部分运行在Atlas500 host CPU（Hi3559A）上，另一部分运行在Atlas 500 device CPU（Arm Cortex-A55）上。

Atlas 500没有提供在Host CPU和Device CPU上运行的编译工具链，因此需要进行交叉编译。Atlas500 Host软件交叉编译和Device软件交叉编译方法类似，只是交叉编译工具和链接库不同，因而本文档统一说明Host和Device的软件编译方法。为了提高软件编译效率，一般使用自构建工具进行，Atlas 500支持使用Make和Cmake两种使用较多的构建工具进行软件构建，构建工具说明如下：

- Make是Linux下最常用的构建工具，基于Makefile文件所描述的规则进行软件的构建。Atlas 500默认使用Make编译软件。
- Cmake是一款跨平台构建工具，语法简单，可移植性强，Atlas也支持使用Cmake进行软件编译。

8.4.1 使用 Make 编译 Atlas 500 软件

步骤1 在开发主机上执行如下命令安装make，Make的使用方法请参考《GUN make手册》。

sudo apt install make

步骤2 编写Host侧主程序的Makefile文件。

说明

建议基于HelloDavinci工程根目录的Makefile文件修改得到Makefile文件，不建议从头开始编写Makefile文件。

根据Atlas500 DDK里的交叉编译工具链，头文件路径和链接库文件路径，修改该Makefile文件中如下信息，其他内容不做改动。

```
DDK_HOME ?= [Atlas500 DDK所在目录]
CC := $DDK_HOME/toolchains/Euler_compile_env_cross/arm/cross_compile/
install/bin /aarch64-linux-gnu-gcc
CPP := $DDK_HOME/toolchains/Euler_compile_env_cross/arm/cross_compile/
install/bin /aarch64-linux-gnu-g++
LOCAL_MODULE_NAME := [Atlas500主程序的名字]
local_src_files := [工程main函数所有源文件路径]
local_shared_libs_dirs := $(DDK_HOME)/lib64/
local_shared_libs := [main需要链接的库名称] # 库的名称为链接库文件名去掉后缀名和开头的lib字母
```

步骤3 编写Host侧动态库程序Makefile文件。

说明

建议基于HelloDavinci工程SrcEngine目录下的Makefile文件修改获得。

根据Atlas500 DDK里的交叉编译工具链，头文件路径和链接库文件路径，修改“SrcEngine”目录下的Makefile文件中的如下信息，待修改的内容与**步骤2**基本一致，其他内容不做改动。

```
DDK_HOME ?= [Atlas500 DDK所在目录]
CC := $DDK_HOME/toolchains/Euler_compile_env_cross/arm/cross_compile/
install/bin /aarch64-linux-gnu-gcc
CPP := $DDK_HOME/toolchains/Euler_compile_env_cross/arm/cross_compile/
install/bin /aarch64-linux-gnu-g++
LOCAL_MODULE_NAME := [host目标动态库的存储路径]
local_src_files := [host目标动态库所有源文件路径]
local_shared_libs_dirs := $(DDK_HOME)/lib64/
local_shared_libs := [host目标动态库需要链接的库名称] # 库的名称为链接库文件名去掉后缀名和开头的lib字母
```

步骤4 编写Device侧动态库程序Makefile文件。

说明

建议基于HelloDavinci工程HelloDavinci子目录下的Makefile文件修改获得。

根据Atlas 500 DDK里的交叉编译工具链，头文件路径和链接库文件路径，修改“HelloDavinci”目录下Makefile文件中的如下信息，其他内容不做改动。

```
DDK HOME ?= [Atlas500 DDK所在目录]
CC := $DDK HOME/toolchains/aarch64-linux-gcc6.3/bin/aarch64-linux-gnu-gcc
CPP := $DDK_HOME/toolchains/aarch64-linux-gcc6.3/bin/aarch64-linux-gnu-g+
+
LOCAL_MODULE_NAME := [device目标动态库的存储路径]
local_src_files := [device目标动态库所有源文件路径]
local shared libs dirs := $(DDK HOME)/lib64/
local_shared_libs := [device目标动态库需要链接的库名称] # 库的名称为链接库文件名去掉后缀名和开头的lib字母
```

步骤5 编写shell脚本，建议参考HelloDavinci工程根目录下的build.sh文件（通常情况下无需改动）。

build.sh文件默认内容如下所示：

```
#!/bin/sh
if [ -z $DDK_HOME ];then
        echo "[ERROR] DDK_HOME does not exist! Please set environment
variable: export DDK_HOME=<root folder of ddk>"
        echo "eg:  export DDK HOME=/home/HwHiAiUser/tools/che/ddk/ddk/"
        exit 0
fi
export LD_LIBRARY_PATH=${DDK_HOME}/uihost/lib/
TOP_DIR=${PWD}
cd ${TOP_DIR} && make
cp graph.config ./out/
mkdir ./out/result_files
for file in ${TOP_DIR}/*
do
if [ -d "$file" ]
then
  if [ -f "$file/Makefile" ];then
    cd $file && make install
  fi
fi
done
```

步骤6 进入工程根目录，执行如下命令编译程序，编译后在工程根目录下out文件夹中查看生成的目标程序。

export DDK_HOME=[Atlas500 DDK安装目录]&&./build.sh A500

步骤7 行命令**scp**将生成的out目录拷贝至Atlas500 host系统上运行。

----结束

8.4.2 使用 CMake 编译 Atlas500 软件

步骤1 在开发主机上执行**udo apt install cmake**安装Cmake工具。

步骤2 执行**cmake －version**命令查看CMake版本（版本需大于2.8.4）。

步骤3 请参考Atlas500 Sample工程设置搭建Cmake编译工程。

步骤4 将Atlas500 Sample工程根目录下的CMake目录复制到目标工程主目录上一级目录。

步骤5 将Atlas500 Sample工程里某一个子工程（如“HelloDavinci”）目录下的build.sh拷贝到目标工程主目录，并修改该文件。

步骤6 将“source $path_cur/../Common/scripts/build_tools.sh”内容删除。

步骤7 进入工程根目录，执行如下命令编译程序，编译后在工程根目录下out文件夹中查看生成的目标程序。

export DDK_HOME=[Atlas500 DDK安装目录]&&./build.sh A500

步骤8 执行命令**scp**将生成的out目录拷贝至Atlas500 host系统上运行。

----结束

9 Atlas 500 软件调测

本章节主要介绍软件开发过程中常用的调测方法。

9.1 日志系统

本章节介绍Atlas 500平台软件开发过程中如何编写进行日志打印和日志查看的代码，用于软件开发过程中打印调测日志，软件运行过程中记录异常。

9.1.1 日志系统配置

设置日记级别

框架提供了5种日志级别：ERROR > WARNING > INFO > DEBUG >EVENT

EVENT级别的日志，输出全系统最关键日志，需要单独设置。

对于其他级别， 设置了日志对应级别后，此级别及高于此级别的等级都日志都能打印出来。

步骤1 使用**4.1.4 进入Atlas 500开发模式**的方式进入开发者模式。

步骤2 使用vi /etc/slog.conf打开日志配置文件，默认日志配置文件内容如下。

```
# Global log level
global_level=3

# User
user=HwHiAiUser

# Share memory size of node 512k * 32 biggest support
maxNodeSize=524272

# Share memory count of queue
maxQueueCount=40

# log-agent-host #
logAgentMaxFileNum=8
# set host one log file max size, range is (0, 104857600]
logAgentMaxFileSize=10485760
# set host log dir
```



```
logAgentFileDir=/var/dlog
...
```

步骤3 修改日志配置文件中的global_level的值即可以修改日志级别。各个日志等级的对应-global_level值如下所示，修改后保存/etc/slog.conf文件。

0：DEBUG级别。

1：INFO级别。

2：WARNING级别。

3：ERROR级别。

步骤4 重启操作系统，设置生效。

----结束

9.1.2 日志查看

Atlas 500的日志文件位于“/var/dlog”目录下：

- 以host-开头的文件为host（Hi3559A）侧日志文件。
- 以device-开头的文件为device（昇腾310）侧日志文件。

操作步骤

步骤1 根据**4.1.4 进入Atlas 500开发模式**方法进入Atlas500操作系统开发模式。

步骤2 进入“/var/dlog”目录。

步骤3 使用编辑器查看host和device侧的日志。

步骤4 查看日志时需要确认“zip_switch”的值是否为“0”。

若“zip_switch”不为“0”，使用root权限打开“/etc/slog.conf”文件，将“zip_switch”修改为“0”并保存。

步骤5 重启Atlas 500操作系统生效。

设置“zip_switch”之后生成的日志可以直接使用编辑器查看。

----结束

9.1.3 日志 API 使用

定义日志模块

步骤1 定义一个日志模块ID，请注意定义的ID的值需要是唯一的。

```
#define USER_DEFINE_ERROR 0x6001
```

说明

Atlas 500 DDK安装目录下的“include/inc/hiaiengine/status.h”文件中已有部分模块ID定义，请避免和其中定义的ID冲突。

步骤2 定义该日志模块的错误码的枚举，如果有多个错误，就在该枚举定义多个错误码。

```
typedef enum
{
```

```
USER_DEFINE_OK_CODE,
USER_DEFINE_INVALID_CODE
}
USER_DEFINE_CODE;
```

步骤3 以HIAI_DEF_ERROR_CODE（宏定义）注册**步骤2**中定义的错误码。“moduleId”对应**步骤1**定义的模块ID，“loglevel”是注册给该错误码的日志级别，有“HIAI_ERROR”、“HIAI_INFO”、“HIAI_DEBUG”和“HIAI_WARNING”四个级别可以使用。“codeName”对应**步骤2**中错误码的枚举标识（去除“_CODE”）的内容，“codeDesc”为该错误码的描述信息。该宏定义实现和使用示例如下所示：

```
HIAI_DEF_ERROR_CODE(moduleId, logLevel, codeName, codeDesc)
HIAI_DEF_ERROR_CODE(USER_DEFINE_ERROR,HIAI_ERROR,USER_DEFINE_OK,"OK")
```

----结束

输出日志到 dlog 日志文件

Atlas 500业务软件中可以调用HIAI_ENGINE_LOG输出日志到dlog日志文件，具体操作请参考《Atlas 500 Matrix API参考》中的“日志（C++语言）”章节。日志打印格式有八种，本章节结合日志注册的内容介绍其中的一种。其定义方式和使用示例如下所示：

- 函数格式如下，参数说明如**表9-1**所示。

  ```
  #define HIAI_ENGINE_LOG(...) \
  HIAI_ENGINE_LOG_IMPL(__FUNCTION__, __FILE__, __LINE__, ##__VA_ARGS__)
  void HIAI_ENGINE_LOG_IMPL(const char* funcPointer, const char*
  filePath, int lineNumber, const uint32_t  errorCode, const char*
  format, ...);
  ```

  表 9-1 参数说明

参数	说明
errorCode	错误码
format log	日志描述
... format	可变参数，根据日志内容添加

- 日志打印调用实现示例如下所示。

  ```
  HIAI_ENGINE_LOG(HIAI_INVALID_INPUT_MSG, "RUNNING OK");
  ```

10 Atlas 500 软件打包部署

本章节主要介绍如何将已经开发好的软件打包成docker镜像并部署。

10.1 基础镜像导入

操作前提

确保打包环境中已安装Docker程序。

说明

由于dockerfile是docker的内置命令，故建议使用18.09版本的docker程序（同Atlas 500相匹配）。

操作步骤

步骤1 执行如下命令导入欧拉基础镜像：

docker load -i Atlas500-EulerOSx.x.x.xxx_64bit_aarch64_basic.tar.gz

Atlas500-EulerOSx.x.x.xxx_64bit_aarch64_basic.tar.gz的版本号以获取的实际包名为准。

步骤2 执行如下命令查看导入的镜像：

docker images

步骤3 执行如下命令将基础镜像重命名为“euler”：

docker tag atlas500-eulerosx.x.x.xxx_64bit_aarch64_basic euler

----结束

10.2 镜像生成

步骤1 在工程目录下编写Dockerfile，内容如下：

```
FROM euler
#将生成的程序拷贝到/app目录，管理平台部署默认使用HwHiAiUser用户运行，请加--chown=1001:1001
COPY --chown=1001:1001 out /app
WORKDIR /app
ENTRYPOINT [“./main”]
```

步骤2 执行如下命令生成镜像：

docker build –t myapp .

步骤3 执行如下命令将镜像导出：

docker save myapp –o myapp.tar

----结束

10.3 镜像部署

Atlas 500同时支持多种方式部署，本章节介绍了管理平台部署和命令行部署两种方式。

10.3.1 管理平台部署

步骤1 在浏览器输入网址：**https://Atlas 500 IP**，进入Atlas 500的管理平台页面，如图10-1所示。

图 10-1 Atlas 500 管理平台页面

步骤2 输入用户名和密码：

- 默认用户名：admin
- 默认密码：Huawei12#$

步骤3 选择“管理 >服务软件 >服务实例 >创建服务实例”，在界面右侧弹出“创建服务实例”页面。

步骤4 填写基本信息，服“务实例名称”和“服务实例描述”。

步骤5 填写容器信息，设置说明如下：

- “容器镜像文件”：单击 ··· 上传文件。

说明

文件大小（含解压后）不得超过512MB，支持*.tar、*.tar.gz格式。

- “附加配置文件”：单击 上传文件，并挂载到相应容器目录。

说明

附加配置文件指用于容器服务内部的配置、数据等文件所在目录，一般用于存放推理模型文件或图片、视频数据等。

文件大小不得超过512MB，支持*.tar、*.tar.gz格式。

- “资源限额”
  - “CPU”：单击单选框并填写数值，设置允许容器使用的CPU最大值。
  - “内存”：单击单选框并填写数值，设置允许容器使用的内存最大值。
  - “AI算力”：单击单选框，设置允许容器使用AI算力。

步骤6 配置环境变量，设置“变量名称”和“变量值”。

- 需要删除环境变量时，单击“操作”下方的“删除”，可删除环境变量。
- 需要添加环境变量时，单击 ，可添加环境变量。

说明

1. 设置容器运行环境中的系统环境变量，可以在服务实例部署生效后修改，为服务实例提供极大的灵活性。
2. 环境变量会明文展示所输入的信息，请不要填入敏感信息，如涉及敏感信息，请先加密，防止信息泄露。

步骤7 配置磁盘分区。

1. 单击“磁盘分区名称”下方的按钮 选择分区名称，可查看该分区的总容量（GB）及可用容量（GB），设置主机挂载点和容器挂载目录，选择权限。
2. 单击“操作”下方的“删除”，可删除磁盘分区。
3. 单击 ，可挂载磁盘分区。

说明

设置挂载到容器的本地磁盘分区，以实现数据文件的持久化存储。

步骤8 配置容器重启策略和容器服务网络。

1. 可配置的容器重启策略包括：
   a. 失败时重启：当应用容器异常退出时，系统会重新拉起应用容器，正常退出时，则不再拉起应用容器。
   b. 不重启：当应用容器退出时，无论是正常退出还是异常退出，系统都不再重新拉起应用容器。
   c. 总是重启：当应用容器退出时，无论是正常退出还是异常退出，系统都将重新拉起应用容器。
2. 可配置的容器服务网络包括：
   a. 主机网络：容器服务网络配置主机网络。
   b. 端口映射：可编辑“容器端口”和“主机端口”。
3. 删除端口映射：单击“操作”下方的“删除”，可删除端口映射。

4. 添加端口映射：单击按钮  ，可添加端口映射。

步骤9 上述配置设置确认无误之后，点击“确定”系统就会自动开始部署。部署成功后会在列表中显示新添加的服务。

须知

- 部署时间相对耗时，在部署结果返回之前，请不要关闭页面，否则将会导致部署失败。

----结束

10.3.2 命令行部署

步骤1 将myapp.tar文件上传至Atlas 500。

步骤2 在Atlas 500上执行如下命令导入镜像。

docker load –i myapp.tar

步骤3 部署运行。

Atlas 500需要添加多个device参数才能加载AI加速模块，如下所示：

```
docker run \
-it \
--device=/dev/davinci_manager \
--device=/dev/hisi_hdc \
--device=/dev/davinci0 \
myapp
```

----结束

11 Atlas 500 软件升级

11.1 固件升级

步骤1 解压“Atlas500-ESP-FIRMWARE-Vx.x.x.xxx.zip”，得到“Atlas500-firmware.hpm”文件，为了确保软件能正常运行在Atlas 500上，请使用“Atlas500-firmware.hpm”文件将系统升级到与开发包相同的版本。

步骤2 进入Atlas 500的管理平台页面（网址：https://Atlas 500 IP）。

步骤3 输入“用户名”和“密码”。

- 默认用户名：admin
- 默认密码：Huawei12#$

步骤4 选择“维护 > 固件升级”，进入“固件升级”界面。

步骤5 单击“升级文件”后的按钮 ···，选择所需文件。

说明

固件格式必须为“*.hpm”。

步骤6 升级固件。

须知

升级过程中请不要关闭电源，以免损坏设备。

1. 单击“升级”，弹出升级固件的提示框。
2. （可选）勾选“升级完成后立即自动重启生效”。

   说明

   - 勾选该选项，升级完成后系统自动重启生效。
   - 不勾选，则需手动重启生效，参考**步骤7**。
3. 单击“确定”。

完成后可在页面查看升级版本号、升级进度等信息。

步骤7 升级完成后，手动单击“重启生效”。

弹出“系统检测到存在已升级但未生效的固件，确定重启系统以使固件生效？”提示框。

步骤8 单击“确定”。等待10分钟左右，完成升级操作。

----结束

12 如何获取帮助

12.1 联系华为前的准备

为了更好的解决故障，建议在寻求华为技术支持前做好必要的准备工作，包括收集必要的故障信息和做好必要的调试准备。

12.2 联系华为技术支持

故障处理过程中遇到难以确定或解决的问题时，请联系华为技术有限公司客户服务中心（电话：4008229999、网址：**http://enterprise.huawei.com**）。同时，您在向华为工程师反馈问题时，请注意收集以下信息：

- 用户名称、地址。
- 联系人姓名、电话号码。
- 故障发生的具体时间。
- 故障现象的详细描述。
- 设备类型、硬件型号及软件版本。
- 故障后已采取的措施和结果。
- 问题的级别及希望解决的时间。

说明

对于以上介绍的可能在本产品上出现的故障现象，按参考处理建议操作后，如果故障仍无法得到解决，请及时与就近的华为办事处或客户服务中心联系，以便能够快速获取华为公司的技术支持。

12.3 做好必要的调试准备

在寻求华为技术支持时，华为技术支持工程师可能会协助您做一些操作，以进一步收集故障信息或者直接排除故障。

在寻求技术支持前请准备好单板和端口模块的备件、螺丝刀、螺丝、串口线、网线等可能使用到的物品。

12.4 如何使用文档

华为技术有限公司提供全面的随设备发货的指导文档。指导文档能解决您在日常维护或故障处理过程中遇到的常见问题。

为了更好的解决故障，在寻求华为技术支持前，建议充分使用指导文档。

12.5 如何从网站获取帮助

华为技术有限公司通过办事处、公司二级技术支持体系、电话技术指导、远程支持及现场技术支持等方式向用户提供及时有效的技术支持。

技术支持网址

查阅技术支持网站上的技术资料：

- 企业网网址：**http://e.huawei.com**
- 运营商网址：**http://carrier.huawei.com**

获取华为技术支持

如果在设备维护或故障处理过程中，遇到难以确定或难以解决的问题，通过文档的指导仍然不能解决，请通过如下方式获取技术支持：

- 联系华为技术有限公司客户服务中心。

  中国区企业用户请通过以下方式联系华为：
  - 客户服务电话：400-822-9999
  - 客户服务邮箱：**ChinaEnterprise_TAC@huawei.com**

    企业网全球各地区客户服务热线可以通过以下网站查找：**企业用户全球服务热线**

    中国区运营商用户请通过以下方式联系我们：
  - 客户服务电话：400-830-2118
  - 客户服务邮箱：**support@huawei.com**

    运营商全球各地区客户服务热线可以通过以下网站查找：**运营商用户全球服务热线**
- 联系华为技术有限公司驻当地办事处的技术支持人员。

案例库与自助平台

如果您想进一步学习和交流：

- 访问**华为服务器信息自助服务平台**，获取相关服务器产品资料。
- 访问**华为服务器自助问答系统**，快速查询产品问题。
- 访问**华为企业互动论坛（服务器）**，进行学习交流。
- 参阅已有案例进行学习：**华为服务器案例库**。

12.6 联系华为的方法

华为技术有限公司为客户提供全方位的技术支持，用户可与就近的办事处联系，也可直接与公司总部联系。

华为技术有限公司

地址：深圳市龙岗区坂田华为总部办公楼

邮编：518129

网址：**http://enterprise.huawei.com/**

A Graph 关键字

参数		说明	必选（M）/可选（O）
graph_id	-	Graph ID，为正整数。	O
priority	-	优先级，无需调整。	O
device_id	-	设备ID。不同Graph可以运行在一个或多个芯片上，也可以运行在不同PCIe卡的芯片上，如果运行在不同芯片上，需要在graph的配置中增加device_id项，用于指定Graph运行的device id。如果不指定，默认运行在device id为0的芯片上。device id的id值从0开始，到N-1结束（N表示Device个数）。	O
engines **说明** 多路解码时推荐配置多个Engine，一个Engine对应一个线程，如果一个Engine对应多个线程，解码时无法保证顺序。	id	Engine ID。	M
	engine_name	Engine名称。	M
	side	Engine运行target，取值为“HOST”或“DEVICE”，需要注意根据业务需求，配置engines运行在HOST或DEVICE。	M

	so_name	Engine运行时需要从Host侧拷贝动态库so文件名到Device侧。如果FrameworkerEngine运行时，需要依赖第三方库文件或自定义库文件时，依赖的so文件也需要配置在graph文件中（请将以下示例中的“xxx”替换为实际依赖库名。）
		`so_name: "./libFrameworkerEngine.so"`
		`so_name: "./libxxx.so"`
		`so_name: "./libxxx.so"`
			O

	thread_num	线程数量。多路解码时，该参数值推荐设置为“1”，如果“thread_num”的值大于“1”，线程之间的解码将无法保证顺序。	M

参数		说明	必选（M）/可选（O）
	thread_priority	线程优先级。取值范围从“1”到“99”，用于设置engine对应的数据处理线程的优先级，采用SCHED_RR调度策略，再根据策略将对应Engine设置为较高优先级。	O
	queue_size	队列大小，默认为“200”。 用户需要根据业务负载的波动、Engine接收数据大小、系统内存进行合理配置。	O

	ai_config	配置示例：
		`ai_config{`
		`  items{`
		`  name: "model_path"`
		`  value: "./test_data/model/resnet18.om"`
		`  }`
		`}`
		● name值无需配置。
		● value值配置为模型文件所在的路径，包含文件名（名称中只允许包含数字、字母、下划线和小数点），可以将参数值配置为单个模型文件的路径（“./test_data/model/resnet18.om”）；也可以将模型文件打包成tar包后，将参数值配置为tar包所在的路径（“./test_data/model/resnet18.tar”）。若存在多个AIConfigItem，在配置tar包路径时，不允许在同一个目录下同时存在名称相同但格式不同的文件，例如“./test_data/model/test.zip”和“./test_data/model/test.tar”。
			O

	ai_model	需要进行如下示例配置示例：
		`ai_model{`
		`  name: ""  //模型名称`
		`  type: ""  //模型类型`
		`  version: ""  //模型版本`
		`  size: ""  //模型大小`
		`  path: ""  //模型路径`
		`  sub_path: ""  //辅助模型路径`
		`  key: ""  //模型密钥`
		`  sub_key: ""  //辅助模型密钥`
		`  frequency: "UNSET"  //设备频率，取值UNSET或0表示取消设置 ；LOW或1表示低频；MEDIUM或2表示中频；HIGH或3表示高频 。`
		`  device_frameworktype: "NPU"  //模型运行设备类型，取值为NPU/IPU/MLU/CPU。`
		`  framework: "OFFLINE"  //执行框架，取值为OFFLINE/CAFFE/TENSORFLOW。`
		`  }`
			O

参数		说明	必选（M）/可选（O）

	oam_config	配置dump算法数据的方式。当推理不准确时，可以查看某些层或全部层的算法结果。
		配置示例：
		`oam_config{`
		`  items{`
		`  model_name: ""  //相对路径+模型名称+后缀`
		`  is_dump_all: ""  //是否全部dump，取值为"true"或"false"。`
		`  layer: ""  //层`
		`  }`
		`  dump_path: ""  //dump路径`
		`  }`
			O

	internal_so_name	内置在Device侧的动态库文件，用户可直接使用，无需从Host侧拷贝到Device侧。	O
	wait_inputdata_max_time	当前收到数据后等待下一个数据的最大超时时间。 ● 单位：毫秒。 ● 默认为“0”，表示数据不超时。	O
	is_repeat_timeout_flag	针对Engine未收到数据，是否进行循环超时处理（唤醒），与wait_inputdata_max_time配合使用，取值说明如下： ● 0：不重复，默认为0。 ● 1：重复。	O
	holdModelFileFlag	是否保留本Engine的模型文件，取值说明如下： ● 0：不保留。 ● 非0：保留。	O
connects	src_engine_id	源Engine ID。	M

	src_port_id	源Engine的发送端口号。
		对于某个Engine，其端口号从“0”开始顺次增加。端口个数定义一般在相应的头文件中以HIAI_DEFINE_PROCESS宏来定义，代码实现如下：
		`#define SOURCE_ENGINE_INPUT_SIZE  1`
		`#define SOURCE_ENGINE_OUTPUT_SIZE  1`
		`class SrcEngine : public Engine {`
		
		`HIAI_DEFINE_PROCESS(SOURCE_ENGINE_INPUT_SIZE, SOURCE_ENGINE_OUTPUT_SIZE)`
		`}`
			M

	target_engine_id	目的Engine ID。	M

参数		说明	必选（M）/可选（O）
	target_port_id	目的Engine的接收端口号。	M
	target_graph_id	目的Graph ID。 Engine支持跨Graph串接，在该场景下，需要在connects的配置中增加“target_graph_id”项来表示接收端的Graph ID，如不配置，则默认在同一个graph中串接。 在有多个Ascend 310芯片情况下，可以使用跨Graph串接将各个模型运行在不同的Ascend 310芯片上，故尽量将相同的模型放在一个芯片上进行推理，减少不必要的内存消耗。 【场景说明】 ● 如表格下图A-1所示，Engine在以下场景中支持跨Graph串接： 1. Graph A的Host Engine直接给Graph B的Host Engine发送数据。 2. Graph A的Host Engine直接给Graph B的Device Engine发送数据。 3. Graph A的Device Engine直接给Graph B的Device Engine发送数据。 ● Engine在以下场景中不支持跨Graph串接：Graph A的Device Engine直接给Graph B的Host Engine发送数据，如表格下图A-1所示。	O
	receive_memory_without_dvpp	● 参数默认值为“0”，表示Deivce上运行的目标Engine的接收内存需要满足4G地址空间限制； ● 若将参数值配置为“1”，表示Device上运行的目标Engine的接收内存不用满足4G地址空间限制。 ● 用户可在Graph配置文件中将所有运行在Device上的目标Engine的参数“receive_memory_without_dvpp”（在connects属性下）设置为“1”，也可以将所有connects属性下的“receive_memory_without_dvpp”参数都配置为“1”，由于在Host上运行的目标Engine即使设置了该参数也无影响，故Matrix接收内出池就无需使用4G地址空间限制，从而提高系统内存使用率。 **说明** 当前DVPP中，VPC、JPEGE、JPEGD、PNGD功能的输入内存需要满足4G地址空间限制，VDEC、VENC功能的输入内存可以无4G地址空间限制。	O

图 A-1 Engine 跨 Graph 串接场景