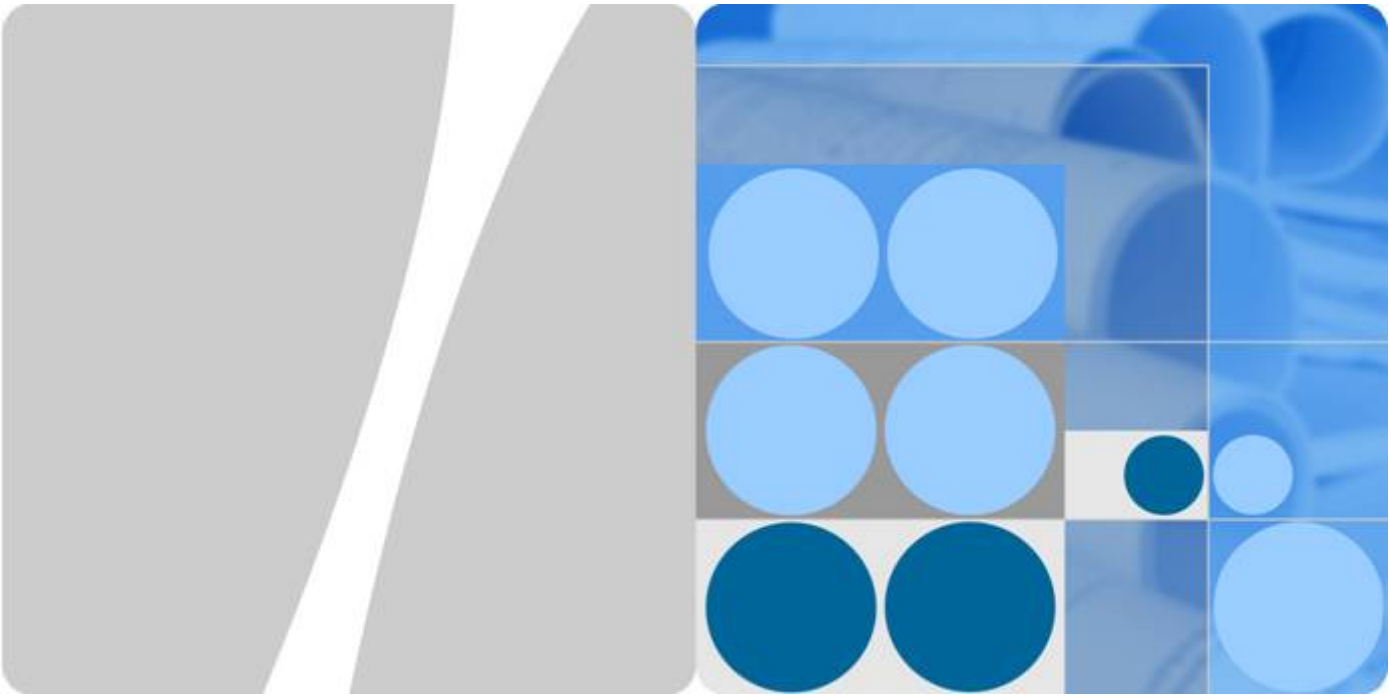


# Ascend 310

# V100R001

# 算法移植指导

文档版本　01

发布日期　2019-03-12

华为技术有限公司

HUAWEI

**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： http://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

# 目 录



# 1 简介

## 1.1 目的

本文以视频监控场景的FasterRCNN检测算法为例，介绍基于神经网络的智能算法从GPU平台使用Mind Studio工具移植到Ascend 310平台的一般流程和注意事项。本文的目标读者是使用Mind Studio工具进行算法移植和开发的高级用户。

## 1.2 算法功能介绍

FasterRCNN检测的算法流程：预处理、神经网络推理、后处理。

- 预处理：将NV12（YUV420SP）的图像（1080P或4K）Resize到1280*720的分辨率，经过YUV2RGB转换、减均值和Scale得到推理的输入。
- 神经网络推理：将预处理后的输入进行推理执行输出结果。
- 后处理：依次是各个类别进行排序、NMS、阈值过滤，然后进行校正得到目标检测位置和置信度。

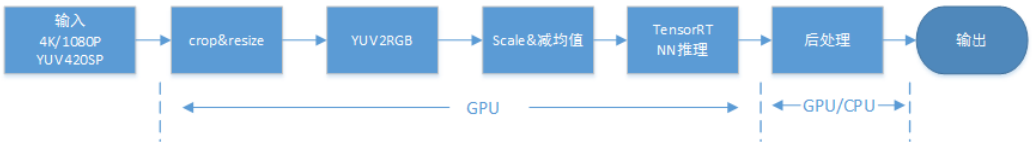
**图 1-1** FasterRCNN 算法的软件应用场景


## 1.3 GPU 与 Ascend 310 Mini 平台算法框架差异

### GPU 平台

由GPU完成Crop/Resize处理、YUV2RGB转换，并进行Scale和减均值处理，执行离线模型推理。推理的结果在CPU做后处理和输出。

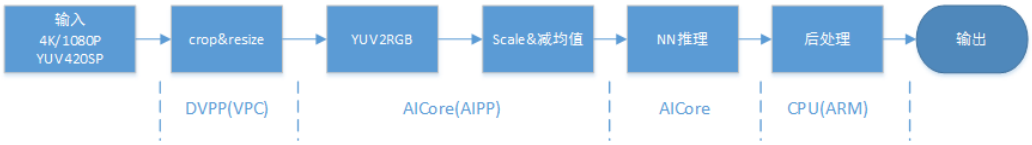
**图 1-2** GPU 平台处理流程



## Ascend 310 平台

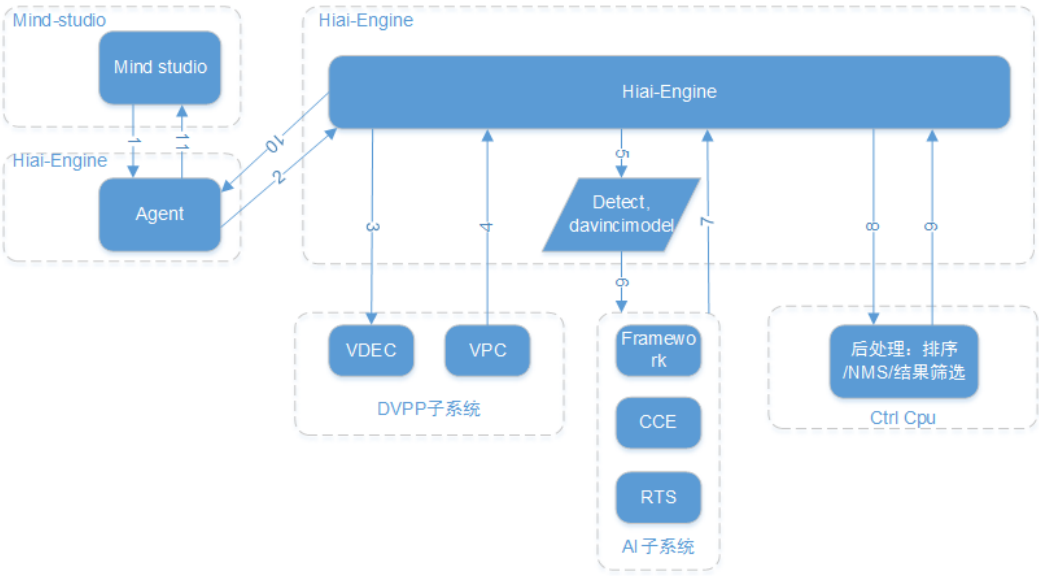
通过HiAI Engine将待处理数据从Host侧发送到Device（Ascend 310芯片），在Device完成所有处理。DVPP子系统实现Crop/Resize，AI子系统完成YUV2RGB、Scale、减均值和神经网络推理，Control CPU实现后处理。

**图 1-3** Device 侧（Ascend310 芯片）处理流程




基于HiAI Engine框架，检测算法从Host到Device的调用处理流程如图1-4所示。

**图 1-4** Host-Device 处理流程




基于上述流程的差异，目标检测模块从GPU移植到Ascend 310平台，主要的修改在于以下几点。

1. 在GPU平台上GPU处理的部分，需要相应移植到DVPP、AI子系统（含AIPP）和CPU上处理。
2. GPU平台NN网络推理的输入图像数据格式通常为BGR Planar INT8 RAW，而Ascend 310平台的NN网络推理可以输入NV12 INT8 RAW（通过AIPP完成转换）或BGR Planar。
3. GPU平台使用的网络离线模型需要转换为Ascend 310平台的网络离线模型；转换时可以通过参数设置，使该网络模型调用AIPP支持YUV2RGB、Scale、减均值等处理。
4. GPU平台模型加载、预测等处理的部分，需要改为使用模型管家AIModelManager，调用Init接口（加载模型）和Process接口（模型预测）。
5. 通过IDE或手动编码方式，将算法模块封装成HiAI Engine接口，提供Init接口（初始化参数、加载模型、分配内存）和Process接口（算法模块业务逻辑）供HiAI Engine框架调用。

注意

将算法封装成HiAI Engine接口、使用模型管家进行模型转化等操作，需要使用DDK中的头文件和so库（头文件、源文件和makefile文件），头文件和so库的路径为：$HOME/tools/che/ddk/ddk/。

# 2 算法移植流程

**表 2-1** 算法移植的整体流程说明

| 关键步骤 | 说明 | 参见章节 |
| --- | --- | --- |
| 环境准备及工具安装 | 包括下载软件包、提供宿主机、准备需要移植的代码、安装Mind Studio工具等。 | **3 前提条件** |
| 工具安装和代码导入 | 安装Mind Studio相关工具，并创建工程、导入代码。 | **4 工程创建及代码导入** |
| 算法移植修改 | 主要介绍移植过程中需要修改的关键代码。 | **5.1 GPU代码移植修改**<br>**5.2 模型转换**<br>**5.3 算法模块的HiAI Engine封装** |
| 平台编译和调试 | 算法修改完成后，进行动态库和可执行文件的编译和调试。 | **6 平台编译和调试** |

# 3 前提条件

## 3.1 环境准备

算法移植前需要准备如表3-1所示的软硬件环境。

**表 3-1** 环境准备

| 类型 | 要求 | 说明 |
| --- | --- | --- |
| 硬件 | Linux系统PC机，并接入网络 | 主机的硬件条件请满足相关要求。<br>请参见《Ascend 310 ReleaseNotes》 |
| 软件 | Ubuntu16.04.3版本、Mind Studio安装包 | 请参见《Ascend 310 ReleaseNotes》 |
| 代码库 | 待移植的GPU代码 | - |
| 模型 | caffe/caffe2/Tensorflow的网络模型文件 | - |

## 3.2 工具安装

请参见相应版本的《Mind Studio工具安装指南》完成Mind Studio工具的安装。

# 4 工程创建及代码导入

## 4.1 创建 Mind Studio 工程

请参见相应版本的《Ascend 310 Mind Studio基本操作》文档中“工程管理 > 操作指南 > 工程创建/删除”完成Mind Studio工程的创建。

**说明**

请根据算法实现语言选择对应的工程类型，例如“File >New Project > C Project”或者“File > New Project > C++ Project”。如果使用图形化界面，可以选择“File > New Project > Mind Project”。

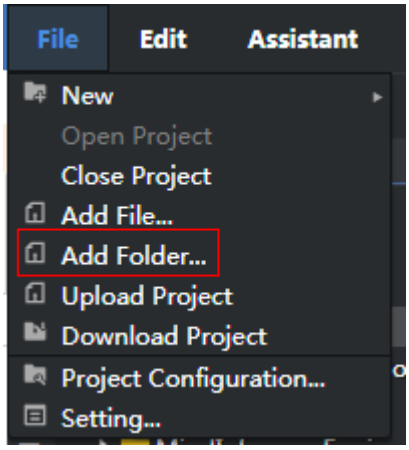
Mind Project同时支持C，C++与Python。

## 4.2 导入代码库

导入代码库有两种方法，如代码库较小，建议使用前台导入方法，如代码库较大，请参见后台导入方法。

### 前台导入

**步骤1** 当进行代码移植的时候，如果已经有了GPU的代码，则首先选中工程目录或子目录，然后选择“File > Add Folder”菜单，进入上传文件窗口。



**步骤2** 在弹出的window窗口里，选择要上传的文件夹，单击**“上传”**。

**----结束**

## 后台导入

**步骤1** 以Mind Studio安装用户登录到Mind Studio服务器。

**步骤2** 进入到“/projects”目录下，可以看到自己创建的工程目录。

**步骤3** 进入自己的工作目录下，如“/projects/demo”，将原来的代码直接拷贝到该目录下，即完成代码的导入。

导入后在Mind Studio界面上可以看到对应目录。

**说明**

代码库较大时，推荐使用后台导入方法。

**----结束**

# 5 算法模块修改

## 5.1 GPU 代码移植修改

在GPU平台上GPU处理的部分，需要相应移植到DVPP、AI子系统（含AIPP）和CPU上处理。

### 5.1.1 GPU 内存处理的函数修改为 C 或 C++函数

需要把Cuda的内存管理的函数如：cudaMalloc()，cudaMemset()，cudaFree()，cudaMemcpy()替换为C语言的接口或C++语言的接口。

- C语言的接口：malloc()，meset_s()，free()，memcpy_s()
- C++语言的接口：new，meset_s()，delete，memcpy_s()

GPU版本代码示例如下所示。

```
/*
* 内存分配
*/
cudaError_t cudaMalloc(void **devPtr, size_t count);

/*
* 内存初始化
*/
cudaError_t cudaMemset(void *devPtr, int value, size_t count);

/*
* 内存释放
*/
cudaError_t cudaFree(void *devPtr);

/*
 * 内存传输
 * enum cudaMemcpyKind{
 *     cudaMemcpyHostToHost,
 *     cudaMemcpyHostToDevice,
 *     cudaMemcpyDeviceToHost,
 *     cudaMemcpyDeviceToDevice,
 *     };
 */
cudaError_t cudaMemcpy(void *dst, const void *src, size_t count, enum cudoMemcpyKind kind);
```

### 5.1.2 GPU 的 Crop 和 Resize 接口替换为 VPC 函数接口

图像的Crop或Resize的GPU处理接口，需要替换为VPC函数的接口，详见**7.2.1 VPC函数接口说明**。

### 5.1.3 GPU 的网络模型处理修改为 AI 网络模型子系统处理

需要将模型的初始化，内存处理，模型输入输出的逻辑处理统一切换为Ascend 310平台AI网络模型子系统的处理方式。详细的接口说明，请参见**7.2.2 AI网络模型子系统函数接口**。

## 5.2 模型转换

模型转化请参见《Ascend 310 Mind Studio基本操作》中的“模型管理”章节，将已有的Caffe模型进行离线模型转换，使之可以适配Ascend 310平台。

也可以参见**7.1 使用命令行进行离线模型转换**通过命令进行模型转化。

## 5.3 算法模块的 HiAI Engine 封装

算法移植到Ascend 310平台时需要进行HiAI Engine封装，以适配HiAI Engine框架。

算法移植到Ascend 310平台时，通常以算法模块为粒度，通过IDE或手工编码的方式进行接口封装，每个算法模块封装为一个Engine。每个Engine需要实现算法模块的Init()接口和Process()接口，其中HiAI Engine框架会在系统启动时调用一次Init()，进行Engine的参数初始化、内存分配、模型加载。

算法模块的Init接口如下所示。

```
#define INPUT_PORT_NUM   3  // 输入端口个数
#define OUTPUT_PORT_NUM  2  // 输出端口个数
#define INPUT_MSGQUEUE_LEN 1024  // 缓存队列长度
class AIEngineDemo : public hiai::Engine {
    public:
        AIEngineDemo()
        : input_que_(INPUT_PORT_NUM, INPUT_MSGQUEUE_LEN)
        {

        }
        ~AIEngineDemo()
        {

        }
        HIAI_StatusT Init(const hiai::AIConfig &config,
            const std::vector<hiai::AIModelDescription>& model_desc)
        {

        }
```

Process接口中实现数据的传输和业务逻辑，如下所示。

```
    /**
    * @ingroup  engine的process处理函数，由框架调用，对外不可见
    * @param  [in]: shared_ptr<void>& arg0,shared_ptr<void>& arg1,shared_ptr<void>& arg2
    **/
    HIAI_IMPL_ENGINE_PROCESS("AIEngineDemo", AIEngineDemo, INPUT_PORT_NUM)
    {
        // 接收输入数据，放入缓冲队列，各个端口数据都拿到后，取出
        input_que_.PushData(0, arg0);
        input_que_.PushData(1, arg1);
```

```
        input_que_.PushData(2, arg2);

        /*************

        进行算法处理

        *************/

        // 发送输出结果到下一个模块
        hiai::Engine::SendData(0, "struct", std::static_pointer_cast<void>(result1));
        hiai::Engine::SendData(1, "string", std::static_pointer_cast<void>(result2));
    }
```

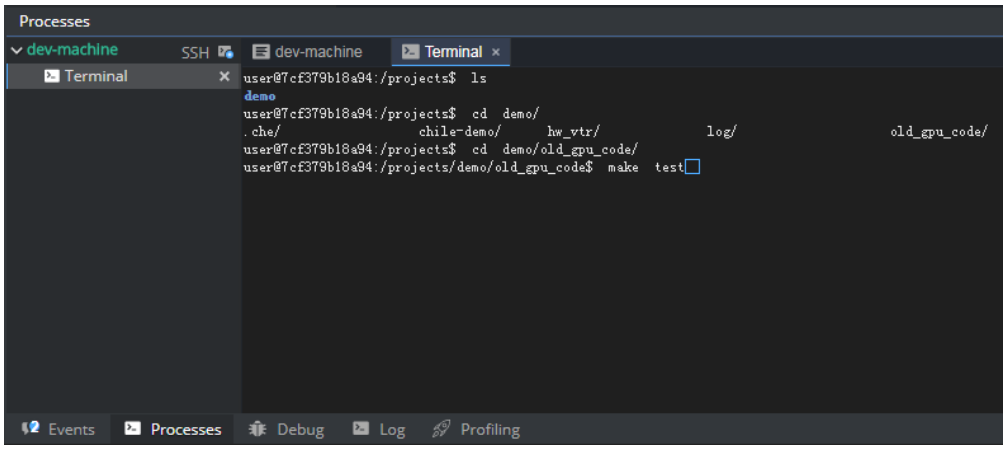
# 6 平台编译和调试

## 6.1 编译算法模块动态库

将算法模块移植完后，需要使用Mind studio将自己的算法模块编译成运行在device侧的动态库。

编译方法有如下:

- 移植的代码，需要自己编写makefile文件，增加DDK的头文件链接路径和相应的so，使用$HOME/tools/che/ddk/ddk/toolchains目录下的编译器，然后在**Terminal**窗口中使用自己的编译框架，编译运行在host侧的动态库，如下图所示。




## 6.2 编译可执行文件

需要基于HiAI Engine框架编写算法运行demo，然后编写makefile，将demo生成为运行在host侧的运行程序。主要完成两部分工作：

1. 实现运行在host侧的src engine和dst engine，用来在host侧读取和接收算法输入/输出数据。
2. 实现graph配置文件，确定不同engine之间的数据传输信息，配置文件中需要指定算法模块的so文件和模型文件，算法so文件和模型文件的路径可以是绝对路径也可以是与执行文件的相对路径。

   graph配置文件示例如下。

```
graphs {
  graph_id: 100
  priority: 1
  engines {
    id: 1000
    engine_name: "HIAISourceEngine"
    side: HOST
    thread_num: 1
  }
  engines {
    id: 1001
    engine_name: "AIPlateDetection"
    so_name:"/projects/demo/old_gpu_code/out/lib_algorithm.so"
    side: HOST
    thread_num: 1
    ai_config {
      items {
        name: "model_path"
        value: "/projects/demo/LP_det.om"
      }
    }
  }
  engines {
    id: 1002
    engine_name: "HIAIDstEngine"
    side: HOST
    thread_num: 1
  }
  connects {
    src_engine_id: 1000
    src_port_id: 0
    target_engine_id: 1001
    target_port_id: 0
  }
  connects {
    src_engine_id: 1000
    src_port_id: 1
    target_engine_id: 1001
    target_port_id: 1
  }
  connects {
    src_engine_id: 1001
    src_port_id: 0
    target_engine_id: 1002
    target_port_id: 0
  }
}
```

**说明**

在Ascend 310 ASIC环境上运行，算法so需要编译成device侧的so，在graph配置文件中，side字段需要配置成“DEVICE”。

# 6.3 逐步确认业务功能

1. 当demo实现之后，在算法模块中的关键位置中加打印或者写文件，如预处理前后、进入网络模型之前、网络模型处理完成时和后处理完成时等。

2. 编译出demo执行文件后，可以在后台服务器路径下直接执行该文件。
3. Demo运行完成后，将收集到的打印信息或者写出来的文件与在gpu环境下对应位置跑出来的结果一一进行比较，比对差异，逐步确认各个子功能模块的正确性。也可以通过host侧的日志文件来定位错误信息。

**说明**

- host侧的日志文件名称：/var/dlog/host-*.log。
- device侧的日志文件名称：/var/dlog/device-*.log。
- 日志文件需要通过Mind studio工具查看，请参见《Ascend 310 Mind Studio 开发辅助工具》。

# 6.4 运行方法

当前版本，如果要在ASIC或者Atlas DK环境下运行执行程序，需要进行如下操作。

1. 工程目录下创建out文件夹，需要将运行的依赖库、算法so和模型文件等都放到out目录下。并在该目录下，创建output文件夹，用于存放运行的结果文件。
2. 在out文件夹下新建一个shell脚本命名为main，内容如下。

```
#!/bin/sh
cur_path=`pwd`
demo_path=$(cd $(dirname $0); pwd)
cd $demo_path
pwd
./demo > demo_log
cd $cur_pat
```

该脚本的意义是指定运行执行程序的名称和路径。

3. 单击界面菜单栏中的“Run > Edit Run Configuration”。
4. 在弹出的窗口中，配置host的IP地址，单击“Run”。
开始运行，可在工具下方的dev-machine窗口中看到运行过程。

# 7 参考

## 7.1 使用命令行进行离线模型转换

### 基于 caffe 给出网络转换示例

**步骤1** 以Mind Studio安装用户进入Mind Studio服务器。

**步骤2** 设置依赖库的环境变量。

**vim ~/.bashrc**

执行如下命令在最后一行添加“/projects/uihost/lib/”的环境变量。

**export LD_LIBRARY_PATH="/home/xxx/tools/che/ddk/ddk/uihost/lib"**

输入**:wq!**保存退出。

执行如下命令使环境变量生效。

**source ~/.bashrc**

**步骤3** 准备好需要转换模型的文件（包括xxx.prototxt, xxx.caffemodel, xxx.cfg）。

**步骤4** 基于uihost下的OMG，将caffe模型转换生成适配Ascend 310的Model。

进入ddk下的**uihost/bin**目录。

执行如下命令。

**./omg --model=/home/usr1/davinci2/PedFaceDetect/PedFaceDetect-add-reshape.prototxt --weight=/home/usr1/davinci2/PedFaceDetect/vgg16_1_4_v5_svd.caffemodel --framework=0 --output=./PedFaceDetect-aipp.davincimodel --aipp_conf=/home/usr1/davinci2/PedFaceDetect/aipp_conf_PedFaceDetect.cfg**

其中，

- model：模型结构文件
- weigth：模型权值文件
- framework：深度学习框架平台
  - 0：caffe

  - 1：caffe2
  - 2：MXNET
  - 3：TENSORFLOW
  - 4：FRAMEWORK_RESERVED
- aipp_conf：AIPP相关配置（YUV2RGB，Scale，减均值等），可选
- output：需要生成模型的路径+名字

----结束

# 7.2 公共函数描述

## 7.2.1 VPC 函数接口说明

VPC模块主要进行图片的Crop和Resize，使用DVPP提供的库，通过命令字来选择VPC功能，对外接口如下。

### 创建 DVPP 处理引擎

```
//************************************
//创建DVPP处理引擎
//************************************
int CreateDvppApi(IDVPPAPI *&pIDVPPAPI);

//************************************
```

### 销毁 DVPP 处理引擎

```
//销毁DVPP处理引擎
//************************************
int DestroyDvppApi (IDVPPAPI *&pIDVPPAPI);
```

### DVPP 引擎处理函数

```
//************************************
//DVPP引擎处理函数
// Parameter: [in] pIDVPPAPI   DVPP处理引擎
// Parameter: [in] CMD        DVPP命令字
// Parameter: [inout] MSG      DVPP处理结构体
//************************************
int DvppCtl(IDVPPAPI *&pIDVPPAPI, int CMD, dvppapi_ctl_msg *MSG);
```

### CMD list

```
CMD命令字：
DVPP_CTL_VPC_PROC     //vpc命令字
DVPP_CTL_PNGD_PROC
DVPP_CTL_JPEGE_PROC
DVPP_CTL_JPEGD_PROC
DVPP_CTL_VDEC_PROC
DVPP_CTL_VENC_PROC
```

## 7.2.2 AI 网络模型子系统函数接口

使用AI ModelManager的公共接口来实现网络模型的加载和处理，其对外接口如下。

### 模型初始化函数

```
//************************************
* @brief 初始化接口，业务在该接口中实现模型加载或其他初始化动作
* @param [in] config  配置信息，根据具体业务添加配置信息
* @param [in] model_desc 模型信息，如果不需要模型，则传入空
//************************************
virtual AIStatus Init(const AIConfig &config,
const std::vector<AIModelDescription> &model_descs = {});
```

### 网络输入数据设置

```
//************************************
@brief 设置数据地址
@param [in] data 输入数据指针
@param [in] size 输入大小
//************************************
void SetBuffer(void *data, const int32_t size, bool isown=false);
```

### 模型处理函数

```
//************************************
* @brief 模型计算接口
* @param [in] context 运行时上下文信息，包含engine运行时的一些可变参数配置
* @param [in] in_data 输入数据
* @param [out] out_data 输出数据
* @param [in] timeout 超时时间，同步调用时无效
//************************************
virtual AIStatus Process(AIContext &context,
const std::vector<std::shared_ptr<IAITensor>> &in_data,
std::vector<std::shared_ptr<IAITensor>> &out_data, uint32_t timeout);
```

## 7.3 缩略语和术语一览

### 缩略语

| | | |
|---|---|---|
| **C** | | |
| **CCE** | Cube-based Compute Engine | 基于Cube的加速计算引擎 |
| **D** | | |
| **DDK** | Digital Development kit | 数字开发套件 |
| **DVPP** | Digital Vision Pre-Process | 数字视觉预处理 |
| **I** | | |
| **IDE** | Integrated Development Environment | 集成开发环境 |
| **O** | | |
| **OME** | Offline Model Inference Engine | 离线模型执行接口引擎 |
| **OMG** | Offline Model Generator | 离线模型生成 |
| **T** | | |

| | | |
|---|---|---|
| **TE** | Tensor Engine | 张量引擎 |
| **TVM** | Tensor Virtual Machine | 张量虚拟机 |

## 术语

**R**

**Runtime**

- 和各种AI Frameworks进行适配，产生计算任务，发送到驱动。
- 接收Driver上报的计算结果，和AI Frameworks进行交互。

**S**

**Scalar** 标量，一般表示一个常数

**T**

**Tensor** 张量