IVS1800&ITS800
9.1.0

算法开发指南(基于昇腾 310)

文档版本 01

发布日期 2021-12-31

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暗示的声明或保证。

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

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.....1
2 开发须知.....2
2.1 目的.....2
2.2 关键概念.....3
2.3 产品及软件介绍.....4
2.3.1 IVS1800 介绍.....4
2.3.2 ITS800 介绍.....7
2.3.3 软件架构.....10
2.4 算法运行时环境.....11
2.4.1 算法生态模式介绍.....11
2.4.2 好望生态模式.....12
2.5 算法打包原理.....14
2.5.1 算法安装包制作流程.....14
2.5.2 算法 APP 目录规范.....15
2.5.3 算法容器镜像制作.....18
2.5.3.1 获取容器基础镜像.....19
2.5.3.2 容器镜像制作.....19
2.5.4 config.xml 配置文件.....24
2.6 算法升级原理.....28
2.6.1 升级方式介绍.....28
2.6.2 算法被动升级.....29
2.6.3 算法主动升级.....29
2.6.3.1 算法包下载.....30
2.6.3.2 算法包升级.....31
2.7 算法启停原理.....33
2.8 端口开放策略.....35
2.9 参数配置原理.....35
2.10 任务管理原理.....37
2.11 分时复用任务原理.....39
2.12 视频拉流原理.....40
2.13 分析结果展示原理.....40
2.14 算法模型.....41
3 算法 DEMO 小试.....42

3.1 DEMO 简介 42
3.2 准备演示环境 42
3.3 算法下载/部署 42
3.3.1 在线部署 42
3.3.2 离线部署 44
3.3.2.1 下载算法及 License 文件 44
3.3.2.2 上传算法及 License 文件 46
3.4 算法演示 47
3.4.1 查看算法状态 47
3.4.2 创建分析任务 47
3.4.3 查看分析结果 49
4 编译环境准备 50
4.1 X86 交叉编译环境 50
4.1.1 硬件要求 50
4.1.2 软件要求 50
4.1.3 搭建开发环境 51
4.2 ARM 直接编译环境 52
4.2.1 硬件要求 53
4.2.2 软件要求 53
4.2.3 开发环境 54
4.3 DDK 目录结构 55
4.4 软件开发依赖的头文件和链接库 56
4.5 软件开发编译工具链 56
4.6 开发工具 57
5 算法 DEMO 编译打包 58
5.1 DEMO 逻辑架构 58
5.2 DEMO 代码目录 59
5.3 DEMO 编译 60
5.3.1 X86 交叉编译方式 61
5.3.2 ARM 直接编译方式 61
5.4 DEMO 打包 61
5.4.1 自定义算法名称 61
5.4.2 打包 62
5.5 认证测试 64
5.6 算法包签名 67
5.7 算法包上线算法商城 67
6 算法功能开发 68
6.1 参数配置 68
6.2 任务管理 69
6.3 视频请流/解码 71
6.4 推理 72

6.5 分析结果回传 72
7 License 开发指南 74
7.1 License 流程 74
7.2 硬件 ID 获取接口 75
7.3 License 控制 76
8 算法调测 78
8.1 命令部署 78
8.2 日志参考 79
9 接口参考 81
9.1 DDK 接口开发指南 81
9.1.1 软件开发指导书 81
9.1.2 模型转换指导书 81
9.1.3 DVPP+API 参考 81
9.1.4 Matrix API 参考 82
9.1.5 参考样例 82
9.2 Restful 接口 83
9.3 eSDK 接口(不再推荐) 85
9.3.1 eSDK 接口使用说明 85
9.3.2 eSDK 业务流程 86
9.3.3 eSDK Demo 介绍 87
9.4 服务化接口 90
9.4.1 参数配置接口 90
9.4.2 任务管理接口 90
9.4.3 分析结果回传接口 90
9.4.4 获取硬件 ID 90
10 附录 91
10.1 网站链接 91
10.2 附件 91

1 前言

读者对象

本文档旨在指导开发工程师如何进行IVS1800或ITS800的智能业务开发。读者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IVS1800或ITS800产品架构有一定的了解。

修订记录

文档版本	发布日期	修改说明
02	2021-11-30	整体改版。 1. 补充相关原理介绍。 2. 新增x86交叉编译能力。 3. 增加参数配置/任务管理/分时复用能力。 4. 提供E2E的DEMO用于更详细地指导算法开发。
01	2020-09-15	第一次正式发布

2 开发须知

2.1 目的
2.2 关键概念
2.3 产品及软件介绍
2.4 算法运行时环境
2.5 算法打包原理
2.6 算法升级原理
2.7 算法启停原理
2.8 端口开放策略
2.9 参数配置原理
2.10 任务管理原理
2.11 分时复用任务原理
2.12 视频拉流原理
2.13 分析结果展示原理
2.14 算法模型

2.1 目的

本文主要介绍如何进行IVS1800或ITS800的智能业务开发。

须知

1. 本文提供的参考代码仅供开发者参考，禁止作为商业用途。
2. IVS1800和ITS800和算法相关的硬件和软件的架构是一致的，所以对于算法开发来说不用区分，在文章中统一用IVS1800描述。

2.2 关键概念

概念	解释
Ascend 310	后文也会称“D芯片”。 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，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Manager（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。
AIPP	AI（AI Pre Process）预处理 支持格式转换、Padding/Crop、CSC色 域转换（YUV2RGB或者RGB2YUV）、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。

概念	解释
Atlas300	也是基于昇腾310芯片及构建算法硬件构架平台的产品。可以复用算法生态。
A算法	基于Hi3559A芯片开发的AI算法。
D算法	基于Ascend 310芯片开发的AI算法。

2.3 产品及软件介绍

2.3.1 IVS1800 介绍

产品定位

IVS1800是华为技术有限公司（以下简称华为）面向园区、普教、社区、小型网点等边缘应用场景的小型盒子类设备，具有超强计算性能、人性化本地人机界面设计、设备即插即用、易于管理运维、高可靠等特点。

IVS1800 硬件介绍

本章节简单介绍IVS1800的硬件信息，详细请参考产品文档。

图 2-1 前面板

前面板口说明如表2-1所示。

表 2-1 接口说明

名称	类型	用途
USB（Universal Serial Bus）接口	USB2.0	用于连接USB设备，常见使用场景为： • 连接U盘。 • 连接USB鼠标。 • 连接移动硬盘。

前面板指示灯说明如表2-2所示。

表 2-2 指示灯说明

名称	状态说明
LAN网口指示灯	• 常亮：设备后面板对应网口连接正常。 • 灭：设备后面板对应网口连接不正常。
硬盘状态指示灯	• 绿灯闪烁：硬盘有数据读写。 • 红灯常亮：至少存在一个硬盘故障。 • 灭：硬盘无数据读写。
电源健康指示灯	• 绿灯常亮：设备正常运行。 • 黄灯5Hz闪烁（0.1s亮、0.1s灭）：非硬盘类一般故障，不影响业务运行。 故障类型包括：温度过高、风扇堵转。 • 红灯5Hz闪烁（0.1s亮、0.1s灭）：致命故障，影响业务运行。 故障类型包括：持续高温、风扇不转、其他器件故障。 • 灭：关机状态。 包括以下场景： – 设备开关未闭合，设备未上电。 – 开机键未打开，软件未运行。

后面板

后面板接口和按钮说明如表2-3所示。

表 2-3 接口和按钮说明

名称	类型	用途
接地端	-	连接地线。
RS485接口	RS485	用于连接外接云台、门禁控制系统等。 硬件预留接口，软件功能暂不支持。
报警输入接口	IO端子	用于连接外部报警输入设备，比如门禁系统。 硬件预留接口，软件功能暂不支持。
报警输出接口	IO端子	用于连接外部报警输出设备，比如警铃系统。
串口	COM	用于设备维护接入。

名称	类型	用途
音频输入接口	RCA	用于音频输入。可以用于语音广播或者与带麦克风的摄像机语音对讲。
音频输出接口	RCA	用于音频输出。可以用于收听摄像机的随路语音。
HDMI（High-Definition Multimedia Interface）接口	HDMI	用于HDMI视频输出，支持双HDMI异源输出。 ● HDMI1.4接口输出分辨率最大支持1080P。 ● HDMI2.0接口输出分辨率最大支持4K。
GE1网口、GE2网口	RJ45	千兆以太网口，用于连接网线。
USB（Universal Serial Bus）接口	USB3.0	用于连接USB设备，常见使用场景为： ● 连接U盘。 ● 连接USB鼠标。 ● 连接移动硬盘。
开机键	-	用于业务软件开关机。 ● 设备电源开关闭合后，设备自动进入运行状态，无需手动按开机键。 ● 设备电源开关闭合状态下，长按开机键10秒，设备进入待机状态；再次按开机键，设备进入运行状态。 设备待机状态下，开机键灯亮且显示蓝色；设备运行状态下，开机键灯灭。
复位键	-	用于恢复出厂设置。
电源线接口	-	连接电源。
电源开关按钮	-	用于设备上下电。

后面板指示灯说明如表2-4所示。

表 2-4 指示灯说明

名称	状态说明
网口ACT指示灯	● 灭：表示当前无数据传输。 ● 闪烁：表示有数据正在传输。
网口LINK指示灯	● 常亮：表示网络连接正常。 ● 灭：表示网络未连接。

名称	状态说明
开机状态灯	• 常亮：表示业务软件已经运行。 • 灭：表示业务软件未启动。

2.3.2 ITS800 介绍

产品定位

ITS800是华为技术有限公司（以下简称华为）全新推出的智能边缘感知计算设备，适用于城市交通十字路口监测、高速路况监测、车路协同等场景，提供车牌、车速、位置、属性、交通事件监测等精准元数据，结合高精度地图可合成多种交通业务数据，比传统方法采集的数据更加精准、稳定。

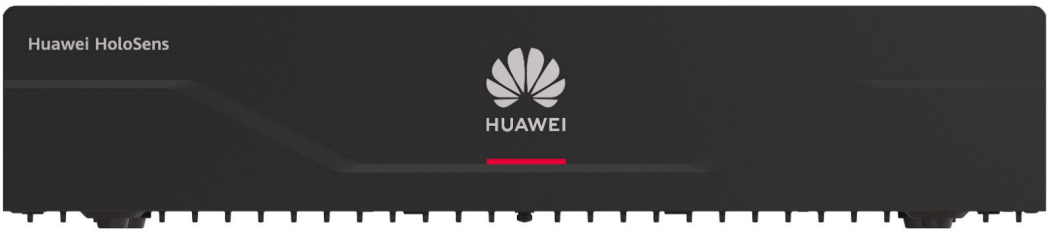
ITS800 硬件介绍

本章节简单介绍ITS800的硬件信息，详细请参考《ITS800 产品文档》。

前面板

ITS800的前面板视图如图2-2所示。

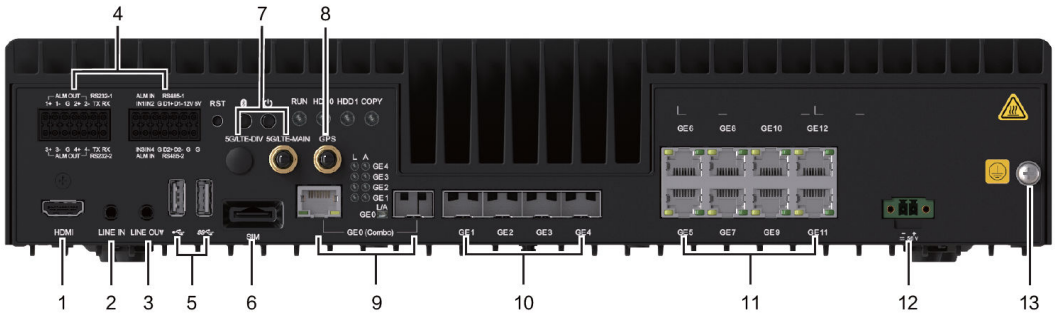
图 2-2 前面板视图



后面板

ITS800的后面板视图如图2-3所示。

图 2-3 后面板视图（接口）

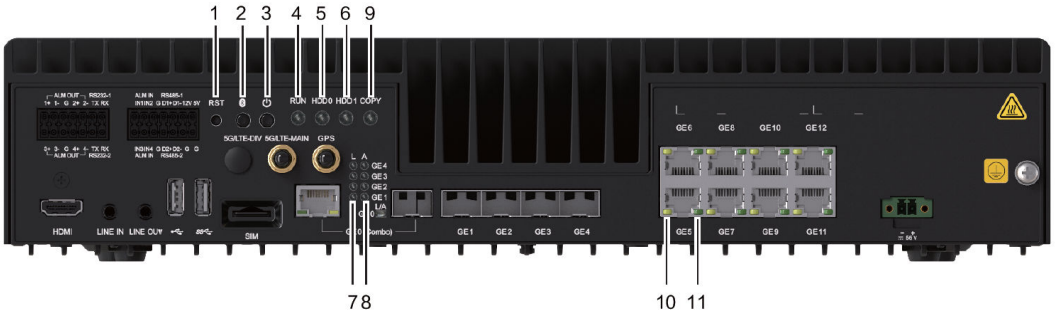


后面板接口说明如表2-5所示。

表 2-5 接口说明

序号	名称	说明
1	HDMI接口	如需输出视频，使用HDMI视频线缆连接视频输出端口和显示器。 HDMI视频线缆不在安装附件包中，需要用户自行购买，请选择长度小于3M的屏蔽线。
2	音频输入接口	如需使用麦克风等音频输入设备，按照标识将音频输入接口连接至音频输入线缆及设备。
3	音频输出接口	如需使用耳机、音响等音频输出设备，按照标识将音频输出接口连接至音频输出线缆及设备。
4	凤凰（告警）端子接口	如需使用烟雾探测器等报警输入设备，或警铃等报警输出设备，将线缆端子与ITS800凤凰端子接口相连接，并确保报警输入设备和ITS800都已经连接地线。 RS232线缆长度建议不超过10m。
5	USB接口	使用USB接口连接移动硬盘、U盘等设备。ITS800后面板有两个USB接口，用户可以选择其中一个接口进行连接。
6	SIM卡接口	5G/LTE模块需配合SIM卡使用，用户自行购买Micro SIM卡。 SIM卡不支持热插拔。根据环境温度要求，必须使用工业级SIM卡。 说明：暂不可用。
7	5G/LTE天线接口	如需使用5G/LTE天线连接网络，将天线设备通过5G/LTE天线接口连接至设备。 说明：5G/LTE-DIV接口为预留，暂不可用。
8	GPS天线接口	如需使用GPS天线连接网络，将天线设备通过GPS天线接口连接至设备。
9	上行GE/SPF combo接口	使用网线连接设备到电交换机或光交换机，2个端口同时只能使用其中1个。 上行GE/SPF comba接口为预留，暂不可用。
10	下行GE光口	连接摄像机。
11	下行GE网口	连接摄像机。
12	电源接口	ITS800电源线缆采用2-PIN直流线缆，一端为2-PIN直流输入端子，另一端为开放式，用户可根据实际使用场景制作线缆，连接至供电系统。2-PIN直流输入端子自带防插反功能，当正负极接反的时候，设备不会上电。
13	接地端	使用十字螺丝刀将保护地线的一端连接到设备的接地端子上，另一端连接到机柜或工作台的接地点上，接地螺钉采用M4螺钉。

图 2-4 后面板视图（指示灯/按钮）



后面板指示灯说明如表2-6所示。

表 2-6 指示灯/按钮说明

序号	名称	说明
1	复位按钮	• 短按1秒，复位系统。 • 长按10秒，恢复出厂设置。
2	蓝牙指示灯/按钮	蓝牙指示灯说明： • 蓝色（常亮）：表示蓝牙正在工作。 • 熄灭：表示未启动蓝牙。 蓝牙按钮说明： • 短按该按钮，系统启动蓝牙功能。 蓝牙指示灯/按钮功能为预留，暂不可用。
3	安全下电指示灯/按钮	安全下电指示灯说明： • 绿色（常亮）：表示正在工作。 • 绿色（闪烁）：表示正在下电过程中。 • 熄灭：表示完成下电，可以断电。 安全下电按钮说明： • 上电状态下短按该按钮，系统启动安全下电流程。 • 上电状态下长按该按钮5秒钟，系统启动强制下电流程。 • 下电状态下短按该按钮，系统启动上电流程。
4	健康指示灯	• 绿色（常亮）：表示设备运转正常。 • 黄色（闪烁）：表示设备中危告警。 • 红色（闪烁）：表示设备高危告警。
5	0号硬盘指示灯	• 绿色（常亮）：表示硬盘运转正常。 • 绿色（闪烁）：表示硬盘正在读写。 • 红色（常亮）：表示硬盘故障。
6	1号硬盘指示灯	

序号	名称	说明
7	GE光口连接状态指示灯	● 熄灭：表示网络未连接。 ● 绿色（常亮）：表示网络连接正常。
8	GE光口传输状态指示灯	● 熄灭：表示无数据传输。 黄色（闪烁）：表示有数据正在传输。
9	COPY指示灯	● 熄灭：处于非备份模式。 ● 绿色（常亮）：表示处于备份模式。 COPY指示灯功能为预留，暂不可用。
10	GE网口传输状态指示灯	● 熄灭：表示无数据传输。 黄色（闪烁）：表示有数据正在传输。
11	GE网口连接状态指示灯	● 熄灭：表示网络未连接。 ● 绿色（常亮）：表示网络连接正常。

2.3.3 软件架构

软件架构总体示意图如图2-5所示：

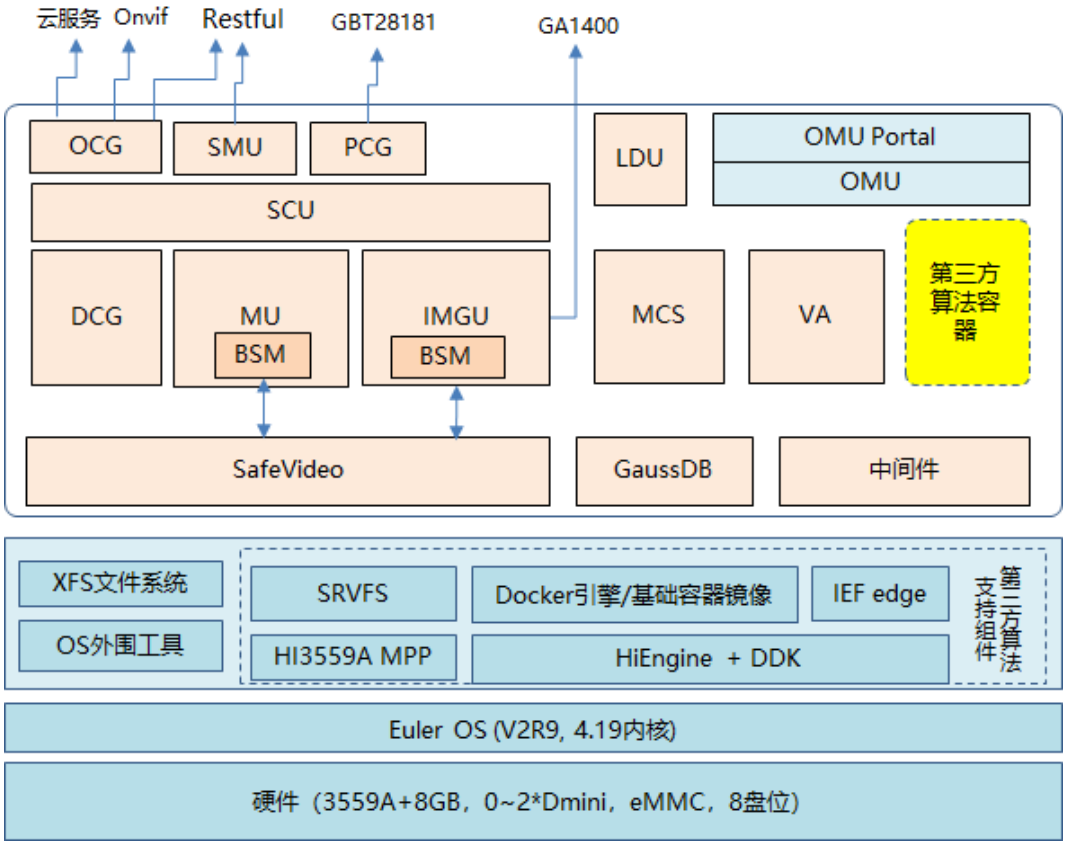


图 2-5 软件架构分层示意图

分为如下4大部分：

- 北向接口软件层：提供Restful/28181/ONVIF/GAT1400北向接口。
- 业务软件能力服务层：SCU/SMU提供用户管理、告警管理、业务存储等。
- 基础硬件能力服务层：IMGU/MU提供媒体能力，包括视频、图片。
- 智能业务层：VA负责获取视频或图片信息，发送给算法进行特征分析。

模块名称	进程名称	模块功能介绍
SafeVideo	smio_xxxx	提供可靠的存储能力。
RIG	ivs_rig	监控进程。用于监控其他进程的运行状态，有进程异常退出时，自动拉起。
VA	va_d0/ va_d1	智能分析进程，分别对应到不同的D卡，运行华为自研算法
MCS	ivs_mcs	媒体检索进程，主要是存储自研算法产生的目标、车辆等特征数据和结构化数据。
MU	ivs_mu	媒体管理进程，主要是用来进行媒体流转发、存储。如果需要录像、实况等业务，这个必须要用到
SMU	ivs_smu	业务管理进程，管理业务，如果需要用到录像配置、查询、创建任务。
SCU	ivs_scu	业务控制进程，业务控制单元，管理业务，如果需要用到录像配置、查询、创建任务。
IMGU	ivs_imgu	图片管理进程，主要用来存储算法产生的图片和北向1400输出。
OMU	ivs_omu	操作维护进程，负责本地运维管理
LDU	ivs_ldu	本地显示单元，接上显示器可以操作、查看。ITS800不支持
SRVFS	ivs_srvfs	本地服务化接口，给第三方算法提供参数配置、任务管理、分析结果回传等能力。

- 设备主芯片为Hi3559A，由海思开发的ARM64指令集芯片。
- 操作系统采用Euler OS，是华为Linux操作系统，系统集成了存储管理和智能分析等软件。

2.4 算法运行时环境

2.4.1 算法生态模式介绍

系统针对D算法，支持IEF生态模式和好望生态模式。

- **IEF生态模式：**是指由华为智能计算产品线运营的Atlas生态，用户可在EI商城对边缘(IVS1800/ITS800/Atlas500等)的算法进行管理。
- **好望生态模式：**是指由华为机器视觉产品线运营的AI生态，算法厂家可开发自己的算法上线到好望商城，用户可在好望商城上购买并管理自己的算法。

IEF生态和好望生态模式的算法都以容器方式运行在IVS1800/ITS800中，但其中有些差别：

- IEF生态模式下，算法开发者将IVS1800/ITS800当做边缘计算盒子，算法运行相对独立，与盒子内本身的视频监控相关业务无耦合关系。
- 好望生态模式结合视频监控的业务，支持通过系统Portal或iClient S100对算法进行安装/卸载/启停管理，支持任务管理、分时复用、分析结果展示等功能，可方便地实现智能业务本地端到端闭环。因而好望生态模式算法除了基于D芯片提供的驱动进行开发外，还建议实现业务管理相关接口。

- IEF生态为华为智能计算产品线主推生态模式。容器可以自己进行制作，也可以通过华为EI商城中的IEF进行制作下发。详细使用方式见链接：

https://support.huaweicloud.com/usermanual-ief/ief_01_0011.html

也可参考对应产品的《 产品文档 》。

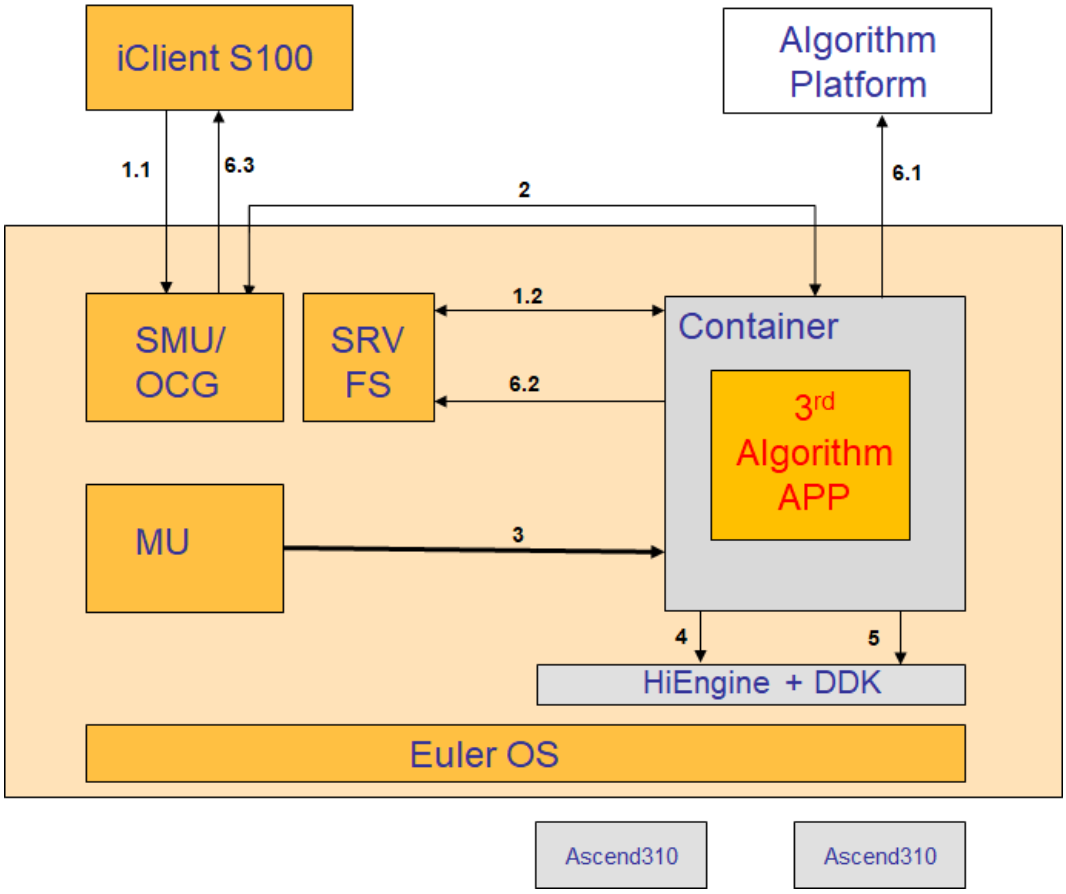
本文将主要针对“好望生态模式”进行算法开发指导。

2.4.2 好望生态模式

系统中第三方算法都要求采用容器方式运行，一个容器运行一个算法插件。

系统中有两块D芯片。一般情况下，一个算法插件使用一个D芯片，不运行系统自带AI算法的情况下，可运行两个第三方算法。

图 2-6 APP 运行环境示意图



1.1- 系统可通过iClient S100客户端或基于北向Restful API(参考**9.2 Restful接口**)开发的第三方客户端对第三方算法进行安装/卸载/启停/参数配置管理。还支持通过该客户端对算法的分析任务进行增删查管理。

1.2- 第三方算法参数配置和任务管理均通过SRVFS提供的服务化接口与三方算法交互(参考**9.4 服务化接口**)，因而三方算法需实现相应接口才能支持通过系统对算法进行参数配置和任务管理。

2- 算法APP可调用系统北向提供的Restful接口(参考**9.2 Restful接口**)向系统请求摄像机媒体流RTSP URL。

3- 算法APP通过RTSP URL向系统发起拉流，系统MU单元转发摄像机的媒体流给算法。

4- 算法APP将获取的媒体流通过DDK的DVPP接口(参考**9.1 DDK接口开发指南**)发送给昇腾310芯片解码。

5- 解码完成，算法APP将通过DDK提供的AI推理接口(参考**9.1 DDK接口开发指南**)实现视频分析(为提高性能，一般会进行跳帧处理)。

6.1-算法APP将分析结果发送给自己的应用平台(由算法自行决定通信协议)。

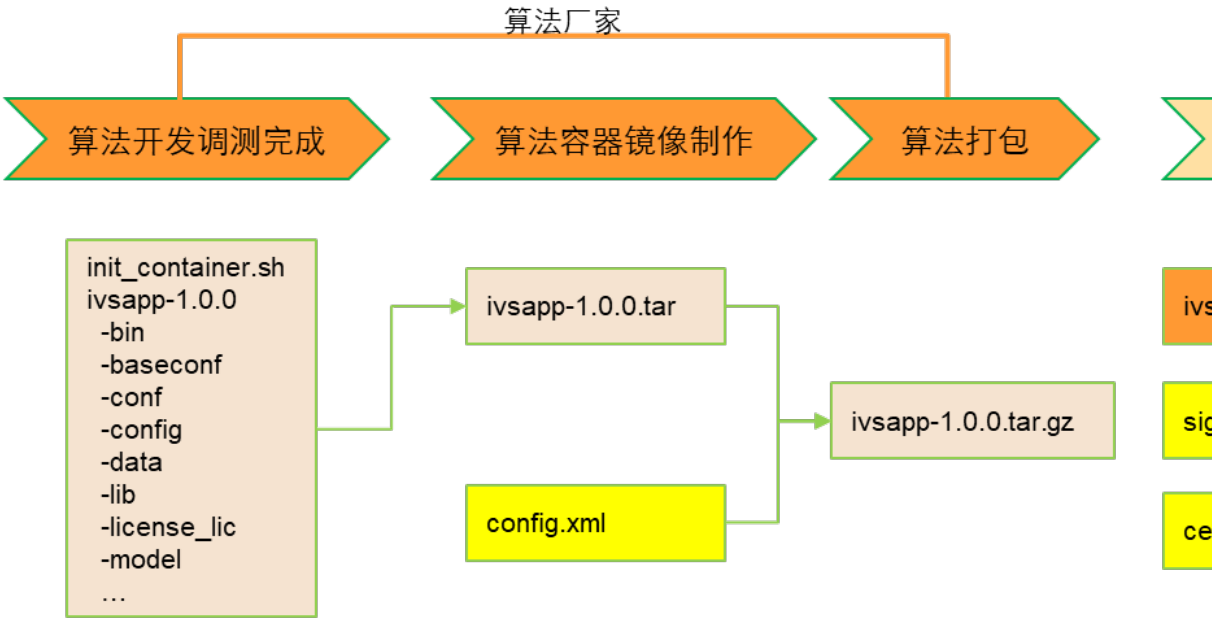
6.2-同时，算法APP也可以使用结果回传接口(参考**9.4 服务化接口**)，将数据推送给系统，系统支持将数据存储/转发。

6.3-系统支持通过北向Restful接口(参考**9.2 Restful接口**)，将分析结果推送出去。

2.5 算法打包原理

2.5.1 算法安装包制作流程

一个可安装部署到设备上的第三方算法插件包，打包流程和结构如下图所示（以算法名称为"ivsapp"为例）：



1. 算法厂家按照目录结构制作容器镜像**ivsapp-1.0.0.tar**。镜像内目录需符合规范，参考**2.5.2 算法APP目录规范**；容器镜像包制作步骤参考**2.5.3 算法容器镜像制作**；
2. 将容器镜像包和**config.xml**配置文件一起打包成算法包**ivsapp-1.0.0.tar.gz**。其中**config.xml**文件是对该算法的资源需求和能力的描述，参考**2.5.4 config.xml配置文件**的说明。打包指令：

   tar zcvf ivsapp-1.0.0.tar.gz ivsapp-1.0.0.tar config.xml
3. 算法包在商城进行数字签名(参考**5.6 算法包签名**)，生成最终的算法安装包**ivsapp-1.0.0.tar.gz.tar**。算法安装包可离线安装到设备上，也可发布到商城进行在线安装。安装部署参考**3.3 算法下载/部署**；算法安装包发布到商城参考**5.7 算法包上线算法商城**。

须知

- 算法包命名规则：**appname-version.tar.gz**。 其中，**appname**为算法名称，字母建议全部小写，否则无法发布到商城；**version**为算法版本号。
- 为防止算法包传输过程中损坏或被恶意篡改，系统会对算法安装包进行数字签名完整性校验，校验不通过的算法无法安装到系统。

2.5.2 算法 APP 目录规范

在进行算法容器镜像制作时，按以下结构组织文件和目录：

须知

- 在算法被动升级时，目前系统只会保留conf, license_lic, data，upgrade目录下的文件，其他文件或目录均会被替换，算法主动升级保留的文件由算法自定义，具体参考**2.6 算法升级原理**。
- 系统上 /opt/third_algorithm目录是算法程序空间，用于存放算法程序，并将对应算法的conf、res、license_lic目录映射到容器中。
- 系统上/opt/third_algorithm_D目录是算法数据空间，用于存放算法分析产生的数据或升级包，对应算法的data和upgrade目录会映射到容器中。

目录/文件	子目录或文件	升级场景被替换	说明
Dockerfile		N/A	容器描述文件。在制作容器镜像时要执行的动作。实际不会安装到容器内。
useradd groupadd usermod		是	容器内用于用户/用户组管理。将被安装到容器内的 **/usr/sbin/**目录下。从**10.2 附件**《用户管理附件.zip》中获取。
init_container.sh		是	算法启动脚本。容器启动后会首先执行该脚本，作为PID为1的进程在容器内驻留，算法的main进程会通过该脚本拉起。算法启停及其注意事项参考**2.7 算法启停原理**。 该文件将被安装到容器内的绝对路径为 **/home/init_container.sh**。
app	-	N/A	算法文件目录，根据实际算法名称修改，同时要和**Dockerfile**中对应。 该目录下的文件或目录将会被安装到容器内的**/home/3rdApp**目录下。

目录/文件	子目录或文件	升级场景被替换	说明
-	baseconf	是	这个是基础的目录，里面存放portal.conf文件和文件夹conf、res。这些是需要拷贝到容器外部方便修改存储。创建容器成功后，将这些文件和文件夹拷贝到容器外部指定目录，供外部访问和修改。 baseconf ----protal.conf ----conf ----res 算法APP启动时，判断conf目录下是否有该文件，如果没有，则需要将配置文件copy到conf目录下。如果涉及到配置文件升级，算法需要在升级后首次启动时，比较conf和baseconf/conf目录下配置文件的差异，自主完成升级。
-	bin	是	执行文件目录。主进程名为main
-	conf	否	配置文件目录，实际是空目录，通过外部映射进行访问。 对外映射到**/opt/third_algorithm/run/ivsapp/conf** 注意： ● 容器安装后**conf目录下的文件将被外部映射的目录屏蔽**，因而算法打包时不要把配置文件放在conf目录，可以放在baseconf/conf目录，算法启动检测到conf目录无必需的配置文件，则从baseconf/conf目录拷贝一份到conf目录，便于在外部对该配置进行修改。 ● conf目录下的配置文件修改建议通过Portal或客户端进行，因而算法要实现配置服务化接口config.paas.sdc(参考**2.9 参数配置原理**)。直接SSH登录设备修改conf目录下配置的方式难以维护，不推荐使用。 ● 升级场景该目录下的文件会保留。因而算法的新版本要尽量兼容老版本的配置文件，但如果确实需要对conf目录下配置文件升级，则应由算法在首次启动时自行检查配置文件，并完成对应配置文件升级处理。
-	res	是	这个目录下存放资源文件，实际是空目录，通过外部映射进行访问。 对外映射到/opt/third_algorithm/run/**ivsapp**/res
-	license_lic	否	这个目录下存放License文件。实际是空目录，license文件从容器外部放入。升级场景该目录的license文件会保留。 对外映射到/opt/third_algorithm/run/**ivsapp**/license_lic

目录/文件	子目录或文件	升级场景被替换	说明
-	data	否	存放数据目录，实际是空目录，用来存放算法产生的数据，算法主动升级时也用于存放升级包(参考**2.6.3 算法主动升级**)。升级场景该目录数据会保留。 对外映射到/opt/third_algorithm_D/3rdApp/**ivsapp**/data
	upgrade	否	存放数据目录，容器安装后该目录实际是空目录，算法主动升级场景下用于存放算法自己生成的升级配置文件，具体参考**2.6.3 算法主动升级**。 对外映射到/opt/third_algorithm_D/3rdApp/**ivsapp**/upgrade
-	model	是	算法存放模型目录
-	config	是	esdk的配置文件放到这里

目录/文件	子目录或文件	升级场景被替换	说明
-	lib	是	动态库目录，包括eSDK动态库、DDK中的动态库 --lib app用到的动态库文件 --eSDK --config --log.xml --mp_log.conf --net.xml --SDKConfig.xml --lib --libivs_esdk_rtsp_client.so --libIVS_eSDK.so --libIVS_NetSource.so --libIVS_Player.so --ddk --libcrypto.so.1.1 --libc_sec.so --libdrvdevdrv.so --libdrvdsmi_host.so --libdrvdsmi.so --libdrvhdc_host.so --libdrvhdc.so --libmatrix.so --libmemory.so --libmmpa.so --libprofilerclient.so --libprofilerserver.so --libprotobuf.so.15 --libprotoc.so.15 --libsafecity_ivs.so --libslog.so --libssl.so.1.1

2.5.3 算法容器镜像制作

2.5.3.1 获取容器基础镜像

步骤1 在网站页面(**https://support.huawei.com/enterprise/zh/index.html**)搜索“IVS1800”，选择“ 机器视觉 > 智能视频云平台 > IVS1800”，打开“软件”页签。

步骤2 在打开的页面中选择IVS18009.0.RC2Full.zip 点击下载

步骤3 按顺序依次解压
IVS18009.0.RC2Full.zip>IVS1800_9.0.RC2Full.zip>IVS1800_9.0.RC2_Full.zip>IVS.zip

请注意每次解压时的文件名

步骤4 找到**vcn5X0_install**文件夹中的**类似EulerOS_Server-docker.aarch64.tar.xz**（不同版本会有差异）。

----结束

说明

获取的基础镜像包与产品软件版本需要匹配，如果身边有设备，可以SSH登录系统，到**/home/package/EulerOS_Server-docker.aarch64.tar.xz**路径下获取。

2.5.3.2 容器镜像制作

前提条件

- 容器镜像制作需要在ARM服务器上进行，如果编译环境是ARM服务器，可以直接在开发用的服务器上进行镜像制作，也可以在IVS1800/ITS800设备上制作；如果编译环境为X86交叉编译环境，则建议将开发好的组件拷贝到IVS1800/ITS800设备上进行容器镜像制作。
- 制作镜像的服务器需要安装对应版本的docker服务(IVS1800和ITS800设备出厂默认携带)。

注意

如果直接在IVS1800/ITS800设备上进行容器镜像制作或进行算法调测，在此之前前，需要先挂载至少一块硬盘，并在portal上对硬盘进行初始化(ITS800自带2块硬盘，只需在portal进行初始化)。否则可能导致算法相关文件占用系统根目录，而导致系统本身的业务进程无法正常运行。

操作步骤

步骤1 运行docker服务

- 启动docker服务
  systemctl restart docker
- 查看运行状态
  systemctl status docker
- 检查容器版本命令
  docker version


```
Euler:/usr/app/container # docker version
Client:
 Version:           18.09.0
 EulerVersion:      18.09.0.72
 API version:       1.39
 Go version:        go1.11
 Git commit:        eab6452281
 Built:             Mon Jul 22 00:00:00 2019
 OS/Arch:           linux/arm64
 Experimental:      false

Server:
 Engine:
  Version:          18.09.0
  EulerVersion:     18.09.0.72
  API version:      1.39 (minimum version 1.12)
  Go version:       go1.11
  Git commit:       eab6452281
  Built:            Mon Jul 22 00:00:00 2019
  OS/Arch:          linux/arm64
  Experimental:     false
```

步骤2 镜像制作

1. 导入基础镜像

   docker images

```
Euler:~ # docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ivsbase             v1.0.0              0de3ed3fcd11        18 hours ago        282MB
```

   如果没有看到基础镜像(ivsbase)，则需要导入基础镜像。

   将**2.5.3.1 获取容器基础镜像**中获取的**EulerOS_Server-docker.aarch64.tar.xz**基础镜像文件复制当前工作目录下。

   执行命令：

   docker import ./EulerOS_Server-docker.aarch64.tar.xz ivsbase:v1.0.0

   docker images

```
Euler:~ # docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ivsbase             v1.0.0              0de3ed3fcd11        18 hours ago        282MB
```

2. 拷贝用户及用户组创建文件

   新建ivsapp文件夹，将附件<用户管理附件>中useradd、groupadd和usermod放到目录下。

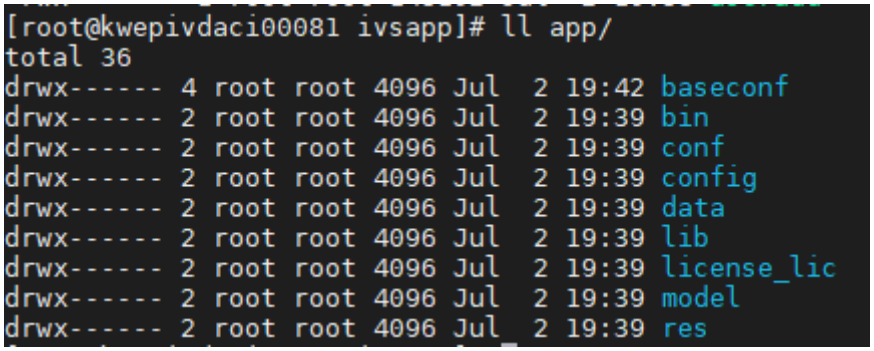
   mkdir ivsapp

   赋上执行权限

   chmod 755 useradd groupadd usermod

```
[root@kwepivdaci00081 ivsapp]# ll
total 292
drwx------ 11 root root   4096 Jul  2 19:39 app
-rw-------  1 root root    581 Jul  2 19:37 Dockerfile
-rwx------  1 root root 142784 Jul  2 19:38 groupadd
-rw-------  1 root root    295 Jul  2 19:38 init_container.sh
-rwx------  1 root root 143192 Jul  2 19:38 useradd
```


3. 准备算法文件和依赖的动态库及配置文件



其中：

- app 目录下的内容参考**2.5.2 算法APP目录规范**。
- **init_container.sh** 为启动脚本，脚本内容参考**2.7 算法启停原理**，可根据实际情况修改。
- **Dockerfile** 为容器描述文件，指定制作镜像时需要执行的动作。可根据实际情况修改。参考如下：

Dockerfile
#根据基础镜像制作 **FROM ivsbase:v1.0.0** #设置工作目录 **WORKDIR /home/** **COPY useradd /usr/sbin/** **COPY groupadd /usr/sbin/** **COPY usermod /usr/sbin/** #创建目录 **CMD mkdir /home/3rdApp/** #copy算法软件到目录下 **COPY app/ /home/3rdApp/** #启动文件 **COPY init_container.sh /home/** #权限和用户配置 **RUN chmod 755 /usr/sbin/useradd /usr/sbin/groupadd ** **&& groupadd -g 1002 ivs ** **&& useradd -u 1001 HwHiAiUser -g ivs ** **&& usermod -aG dialout HwHiAiUser ** **&& chmod 700 init_container.sh ** **&& chown HwHiAiUser:ivs init_container.sh ** **&& chown -R HwHiAiUser:ivs /home/3rdApp/** **HEALTHCHECK --interval=10s --timeout=3s CMD ps -ef|grep bin/main|grep -v grep**

须知

上述红色字体，要确保算法进程启动文件必须是main命名，否则容器会被终止。

4. 制作算法镜像

   执行命令

   docker build -t ivsapp:v1.0.0 .

   注意：最后面的"."不能缺少, 且“.”之前需要一个空格。

   等待完成后，通过镜像命令查看，发现多了一个ivsapp镜像。

```
Euler:/opt/dockertmp # docker images
REPOSITORY                                  TAG                 IMAGE ID            CREATED             SIZE
ivsapp                                      v1.0                6ba7f8b6ace8        2 minutes ago       402MB
ivsbase                                     v1.0.0              b5c2f55c7e32        6 days ago          282MB
```

说明

其中ivsapp，请根据实际算法名称命名。

步骤3 使用镜像创建容器

docker create -it --device=/dev/davinci_manager --device=/dev/hisi_hdc --device=/dev/davinci0 --device=/dev/davinci1 --name=ivsapp --log-opt max-file=2 --log-opt max-size=30KB --memory 500MB --cpus=0.3 --ulimit nofile=65535:65535 --net=host --user HwHiAiUser --privileged=false --cpuset-cpus="2,3" --pids-limit 100 --security-opt apparmor=docker-default,no-new-privileges,label=level:TopSecret --cap-drop NET_RAW --restart=unless-stopped -v /opt/third_algorithm_D/3rdApp/ivsapp/data:/home/3rdApp/data -v /opt/third_algorithm_D/3rdApp/ivsapp/upgrade:/home/3rdApp/upgrade -v /opt/third_algorithm/run/ivsapp/res:/home/3rdApp/res -v /opt/third_algorithm/run/ivsapp/conf:/home/3rdApp/conf -v /opt/third_algorithm/run/ivsapp/license_lic:/home/3rdApp/license_lic -v /var/dlog:/var/dlog -v /usr/slog:/usr/slog -v /mnt/srvfs:/mnt/srvfs -v /tmp:/tmp ivsapp:v1.0.0 /bin/bash -c '/home/init_container.sh'

参数	说明	备注
-it	交互模式运行容器,并重新分配一个伪输入终端	-
--net	网络模式，目前指定为host，和宿主机共网络	目前为必选
--user	使用HwHiAiUser 用户	可以在设备上使用id HwHiAiUser 查看具体id
--memory	指定算法使用的内存	-
--cpus	指定算法使用的CPU阈值	限制使用一个核的0.3
--cpuset-cpus	算法绑核	Hi3559A，一共4个核，其中2，3为大核，1，4为小核
--name	指定容器运行的名称	-

参数	说明	备注
--ulimit nofile=65535:65535	设置容器能够打开的最大文件句柄数量	测试eSDK相关性能时为必选
--device=/dev/davinci_manager --device=/dev/hisi_hdc	D芯片相关参数	-
--device=/dev/davinci0 *--device=/dev/davinci1*:	D芯片相关参数，根据实际需要配置，如果只使用一个D芯片，则选择其中一个，需求两个时，可以都选上	-
--restart=unless-stopped	容器异常退出时自动重启	当使用docker stop停止容器时，容器不会自动重启
-v /var/dlog:/var/dlog -v /usr/slog:/usr/slog	创建这个映射是为了保证算法的日志可以正常输出。	直接在host下的对应路径，可以看到算法日志输出。
-v /opt/third_algorithm/run/**ivsapp**/conf:/home/3rdApp/conf	配置文件映射，需要把算法配置文件拷贝到外部路径，然后修改配置文件。	配置文件需要放到容器外面，方便修改。
-v /opt/third_algorithm/run/**ivsapp**/res:/home/3rdApp/res	资源文件映射，需要把算法资源文件拷贝到外部路径，供nginx访问。	资源文件需要放到容器外面，方便修改
-v /opt/third_algorithm/run/**ivsapp**/license_lic:/home/3rdApp/license_lic	license文件映射，从外部加载license文件，放到这个目录，供算法鉴权。	-
-v /opt/third_algorithm_D/3rdApp/**ivsapp**/data:/home/3rdApp/data	对应的数据目录，其中**ivsapp**为算法名称，这个目录下保存动态的数据，比如产生的图片、数据库数据等。算法自升级时用于存放升级包(参考**2.6.3 算法主动升级**)。 算法升级时该目录数据会保留。	-
-v /opt/third_algorithm_D/3rdApp/**ivsapp**/upgrade:/home/3rdApp/upgrade	升级配置目录。算法自动升级场景用于存放算法自己生成的升级配置，具体参考**2.6.3 算法主动升级**。	-

参数	说明	备注
-v /tmp:/tmp	tmp文件映射到容器，供算法产生Unix Socket文件使用。	-
--log-opt max-file=2 --log-opt max-size=30KB	日志相关选项	根据需要修改

步骤4 启动容器

启动命令：**docker start *-i CONTAINER ID***（通过docker ps查看的容器ID）

```
bash-4.4$ ls -al
total 16
drwxr-xr-x 1 root root 4096 Apr 29 10:09 .
drwxr-xr-x 1 root root 4096 Apr 29 12:39 ..
drwxr-xr-x 6 1001 1002 4096 Apr 29 10:09 3rdApp
-rw------- 1 1001 1002   40 Apr 29 03:28 init_container.sh
```

步骤5 调试容器

进入容器中，运行自己的程序，直到程序能正常运行(参考本章节最后的“容器基本操作指令”)。

步骤6 导出容器镜像

docker export *CONTAINER ID* -o ivsapp-1.0.0.tar

CONTAINER ID：生成的容器ID

ivsapp-1.0.0.tar: 导出的容器镜像文件的名字

本例导出命令为：

docker export c59fca21c131 -o ivsapp-1.0.0.tar

----结束

容器基本操作命令

- 重启命令：**docker restart *CONTAINER ID*（通过ps查看的ID）**
- 停止命令：**docker stop *CONTAINER ID***（通过ps查看的ID）
- 进入容器：**docker exec -it *CONTAINER ID* /bin/bash**
- 退出容器：在容器中输入**exit**
- 查看日志：**docker logs *CONTAINER ID***
- 导出容器：**docker save *IMAGE ID* -o dockerimage.tar**

2.5.4 config.xml 配置文件

算法的config.xml文件是对该算法的资源需求和能力的描述，系统在安装和运行算法容器时，将根据该配置文件进行处理，具体参考如下：

config.xml配置文件

```
<?xml version="1.0" encoding="utf-8"?>
<config>
    <!--appname-->
    <appname>ivsapp</appname>
    <appdesc>Demo<appdesc>
    <!--version-->
    <version>1.0.0</version>
    <!--vendor-->
    <vendor>superAI</vendor>
    <!--占用多少个D卡，默认1个，最多2个，系统根据配置进行分配-->
    <d_num>1</d_num>
    <!--CPU占用情况，如果使用一个D，设置不大于100%，2个D不大于200%-->
    <cpu>30</cpu>
    <!--内存占用情况，如果使用一个D，设置不大于1000M，2个D不大于2000M-->
    <mem>500</mem>
    <!--一共多少个通道，建议不大于16个-->
    <channel>8</channel>

    <!--是否支持任务管理及分析任务状态反馈, 如果实现了task.pass.sdc提供的服务，则填1，否则为0,默认为0 -->
   <mgrEnabled>1</mgrEnabled>

    <!--算法能力，此标签有效字符串长度请勿超过4096 -->
    <capabilities>
        <loopAbility>
            <!--是否支持分时复用，轮询调度，0-不支持，1-支持-->
            <enable>0</enable>
            <!--分时复用支持的单路执行周期范围，默认3到3600秒，最大范围[0.05,36000]-->
            <minInterval>3</minInterval>
            <maxInterval>3600</maxInterval>
        </loopAbility>

        <!--算法分析能力,单个算法限定最大10种分析类型-->
        <analyzeAbilities>
            <analyzeAbility>
                <!--分析类型1, 最大长度64字节(含结束符)-->
                <analyzeType>温度检测</analyzeType>
                <!--ROI类型,0-无，1-线段，2-区域-->
                <roiType>2</roiType>
                <!--该分析类型支持的ROI数量，roitype为非0时有效，最大值8-->
                <roiNumber>2</roiNumber>
        <!--是否支持分析灵敏度配置(0-不支持，1-支持 该值为1则表明该分析类型实现了灵敏度设置，且可设置灵敏度范围为[0,100]-->
        <sensitivity>1</sensitivity>
        <!--可支持最小检测目标大小，填写宽高更小值，不限时填0-->
        <minObjSize>0</minObjSize>
        <!--可支持最大检测目标大小，填写宽高更大值，不限时填0-->
        <maxObjSize>0</maxObjSize>
        <!--需要在场景图上叠加显示的内容。 算法通过TLV上报分析结果(包括自定义key:value)，客户端在图片显示指定key-value的内容。多个要显示的KEY用逗号分隔，字符串最大长度256-->
        <osdKeys>温度,单位</osdKeys>
            </analyzeAbility>
            <analyzeAbility>
                <analyzeType>反光衣检测</analyzeType>
                <roiType>0</roiType>
                <roiNumber>2</roiNumber>
        <sensitivity>1</sensitivity>
        <minObjSize>0</minObjSize>
        <maxObjSize>0</maxObjSize>
```


config.xml配置文件

```
            </analyzeAbility>
        </analyzeAbilities>
    </capabilities>
</config>
```

各个字段含义：

字段名称				字段类型	必选M/可选O	字段描述
appname				string	M	算法名称。系统将以此作为算法的唯一标识。 最大长度：20字节，字母建议全部小写。
appdesc				string	M	算法描述信息。最大长度：256字节
version				string	M	算法版本。建议采用点分模式，如"1.0.1"
vendor				string	M	开发商名称。最大长度：128
d_num				int	M	占用多少个D卡，默认1个，最多2个，系统根据配置进行分配。
cpu				int	M	CPU占用情况，如果使用一个D，设置不大于100%，2个D不大于200%。 单位：百分比。
mem				int	M	内存占用情况，如果使用一个D，设置不大于1000M，2个D不大于2000M。单位：MB
channel				int	M	一共多少个通道，建议不大于16个。
mgrEnabled				int	O	是否支持任务管理及分析任务状态反馈。 如果实现了task.pass.sdc(参考**9.4.2 任务管理接口**)提供的服务，则填1，否则为0。默认为0

字段名称				字段类型	必选M/可选O	字段描述
capabilities				object	O	算法能力，**此标签下的内容有效字符串长度请勿超过4096**字节。 mgrEnabled为1时，系统将解析算法能力，并将算法能力提供给客户端。客户端在进行任务管理时，会基于算法的能力展示不同的任务属性设置界面。任务管理原理参考**9.4.2 任务管理接口**
	loopAbility			object	O	算法分时复用能力。分时复用原理参考**2.11 分时复用任务原理**
		enable		int	M	是否支持分时复用，轮询调度。0-不支持，1-支持
		minInterval		double	M	分时复用支持的单路执行最小周期。 默认3秒，最小0.05秒。
		maxInterval		double	M	分时复用支持的单路执行最大周期。 默认3600秒，最大36000秒。
	analyzeAbilities			object[]	O	算法分析能力列表，单个算法限定最大10种分析类型。 当算法支持多种分析类型时(比如算法内同时集成了反光衣检测和安全帽检测)，可在此标签说明具体支持哪些。客户端进行分析任务管理时，将提供界面给用户选择具体的分析类型(支持多选、单选)。
		analyzeAbility		object	O	算法分析能力对象。
			analyzeType	string	M	分析类型名称。 最大长度64字节(含结束符) 该名称会显示在客户端供用户在下发分析任务时选择具体的分析类型。
			roiType	int	O	ROI类型。0-无，1-线段，2-区域。默认为0。 该值为1或2时，客户端进行分析任务管理是，将显示划线/画框界面。

字段名称				字段类型	必选M/可选O	字段描述
			roiNumber	int	O	该分析类型支持的ROI数量，roitype为非0时有效，最大值8
			sensitivity	int	O	是否支持分析灵敏度配置。0-不支持，1-支持。 该值为1则表明该分析类型实现了灵敏度设置，且可设置灵敏度范围为[0,100]。客户端进行分析任务管理时，将显示灵敏度设置界面。
			minObjSize	int	O	可支持最小检测目标大小，填写宽高更小值，不限时填0。 该值为非0时，客户端进行分析任务管理时，将显示最小目标绘制界面。
			maxObjSize	int	O	可支持最大检测目标大小，填写宽高更大值，不限时填0。 该值为非0时，客户端进行分析任务管理时，将显示最大目标绘制界面。
			osdKeys	string	O	算法分析结果需要显示的内容。 算法通过TLV上报分析结果(包括自定义key:value，参考**2.13 分析结果展示原理**)，客户端在进行分析结果展示时，显示指定key-value的内容。 多个要显示的KEY用逗号分隔，字符串最大长度256。

2.6 算法升级原理

2.6.1 升级方式介绍

第三方算法的升级，系统当前支持以下两种方式：

- **算法被动升级方式：**用户通过设备的Portal/iClient S100客户端/云服务维护Portal等界面触发升级请求，系统根据升级请求主动停止算法，并对算法进行升级处理。
- **算法主动升级方式：**由算法主动检测“算法服务器”上是否有新版本的算法包，检测到有新包后主动下载新版本算法包，并主动触发升级动作。该方式的升级动作实际也由系统完成，只是升级动作由算法主动触发。

算法被动升级只要求算法遵循目录规范，整个升级流程(配置文件升级除外)由系统完成，算法主动升级需要算法厂家实现算法版本管理、算法版本探测、下载等动作。下面对两种升级模式进行分别介绍，算法厂家可根据业务和应用场景选择一种方式。

2.6.2 算法被动升级

用户通过设备的Portal/iClient S100客户端/云服务维护Portal等界面上传新版本算法包，并启动升级后，系统将按如下方式进行升级处理：

1. 检查升级包完整性，检查失败则停止升级。
2. 备份算法的/opt/third_algorithm/run/appname目录下的conf、license_lic子目录内容，并删除/opt/third_algorithm/run/appname目录(/opt/third_algorithm_D/3rdApp/appname为算法数据空间，里面的内容，如data子目录、upgrade子目录不会删除)。
3. 解压算法包，重新安装算法到算法程序空间/opt/third_algorithm，并创建容器，做好容器内和容器外目录映射，映射关系参考**2.5.2 算法APP目录规范**。
4. 将备份的conf、license_lic目录拷回/opt/third_algorithm/run/appname目录下。

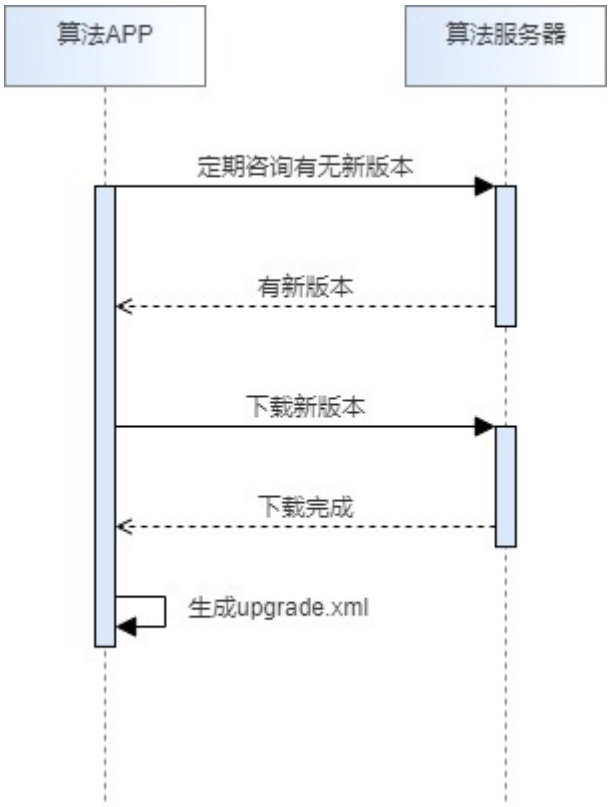
说明

- 从以上升级过程可以看出，升级时系统会保留算法程序空间下的配置文件目录conf、许可文件目录license_lic。也就是容器内conf目录下是老版本配置文件，baseconf目录内是新版本的初始配置文件。新版本算法应尽量兼容老版本的配置文件，如果确实需要对配置文件进行升级处理(如新增配置项)，需要算法升级后首次启动时，自行对配置文件升级(注意，由于原配置项值可能已经做过修改，不应该采用全覆盖模式)。
- /opt/third_algorithm_D/3rdApp/appname下(对应容器内的/home/3rdApp目录)的data和upgrade目录是算法数据空间，升级后里面内容不会变化，因而算法分析产生的数据可放在该目录的子目录下。

2.6.3 算法主动升级

算法开发商可以根据自己的业务实现算法自动升级部署，前提必须有算法服务器用于管理算法版本。

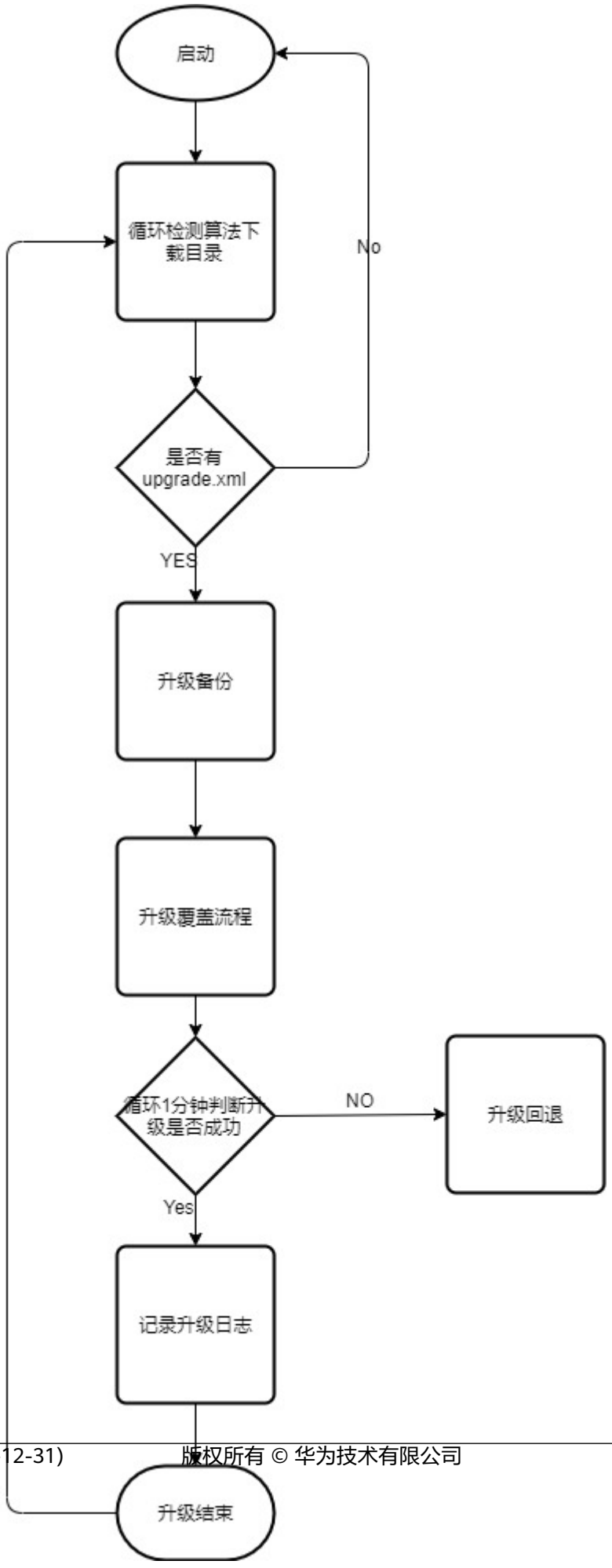
2.6.3.1 算法包下载



第三方算法APP定时向服务器询问是否有新版本需要升级，如果有新版本，则下载到容器内指定的目录（容器内部目录为/home/3rdApp/upgrade（upgrade目录默认由系统创建，如果没有，需要算法自行创建），对应的映射目录为/opt/third_algorithm_D/3rdApp/**appname**/upgrade），第三方算法APP本身对升级包进行MD5校验，校验通过后，生成upgrade.xml文件。微边缘进程OMU进程定时监测目录/opt/third_algorithm_D/3rdApp下所有**appname**目录，如果发现 appname内有upgrade.xml文件后，开始升级流程。

upgrade.xml
<?xml version="1.0" encoding="UTF-8" ?> <Upgrade> <UpgradeMode>1</UpgradeMode> <!-- 升级模式，0表示增量安装，只替换部分，1表示全量升级 --> <ReservedDirs> <!-- 预留的目录，升级后，需要将这个目录的文件再copy回来。--> <DirName>./data</DirName> <DirName>./conf</DirName> </ReservedDirs> <IsClearData>0</IsClearData> <!-- 是否清理数据 --> <DataPath>/home/3rdApp/data</DataPath> <!-- 数据路径 --> </Upgrade>

2.6.3.2 算法包升级



OMU进程启动定时器（10分钟/次），检测指定目录/opt/third_algorithm_D/3rdApp下的**appname**是否满足升级条件，如果需要升级，则先对当前的算法进行备份，然后进行升级覆盖，升级后检测容器运行是否正常，如果不正常，则启动升级回退，否则升级结束。

1. 算法的升级包需要满足固定格式（和安装的格式一致，但不需要进行数字签名）。

   appname-version.tar.gz

2. OMU只保存当前正常运行的版本，不保存历史版本。

说明

1、算法主动升级场景需要算法配合完成，因而需要先通过系统Portal或者iClient S100安装并启动支持主动升级的算法。

2、算法主动升级与算法被动升级场景不同，主动升级由算法在upgrade.xml升级配置文件中指定要保留哪些数据。

2.7 算法启停原理

为实现系统的安全隔离，算法被要求运行在容器中。系统会在必要时对容器进行启停操作(通过docker start/docker stop命令)，以实现对算法的启停。例如用户在iClient S100对算法进行启停操作，或系统为腾出算力自动对算法进行启停操作。

- 算法启动：执行docker start命令，系统将启动对应的docker容器，并执行容器内的**init_container.sh** 脚本。init_container.sh 脚本将作为PID为1的进程运行在容器中，对算法进程进行管理。
- 算法停止：执行docker stop -t=5 命令，系统将向容器内PID为1的进程发送SIGTERM信号，因而init_container.sh 进程将捕获SIGTERM信号，并将信号转发给算法main进程，便于算法进程进行必要的收尾动作，实现优雅退出。若算法在5s内没有完成退出，系统将强制结束该容器，因而**算法的收尾动作请确保在5s内完成**。

按照目录规范(**2.5.2 算法APP目录规范**)：

- 容器内启动脚本路径为： **/home/init_container.sh**
- 容器内算法主进程路径为: **/home/3rdApp/bin/main**

在该规范下，可以直接使用如下内容作为算法的 init_container.sh 启动脚本， 并在自己的**main**进程实现SIGTERM信号捕获及优雅退出处理：

容器内 /home/init_container.sh 启动脚本

```
#!/bin/bash
function has_main () {
    flag=$(ps -ef|grep -v grep | grep main | wc -l)
    if [ ${flag} -eq 0 ]; then
        return 1
    fi
    return 0
}

function clean() {
    local pids=`jobs -p`
    if [[ "$pids" != "" ]]; then
        kill $pids >/dev/null 2>/dev/null
        #最多等待5s，5s后应用还没退出，则PID为1的脚本退出，应用将被强制退出
    local i=0
    while has_main && [ $i -lt 5 ]
    do
        sleep 1
        ((i++))
    done
     fi
     exit
}

#系统执行docker stop时，该脚本会捕获SIGTERM信号，并执行clean操作，clean函数会将SIGTERM信号转给算法应用，以便应用优雅退出。
#应用需要在main进程内捕获SIGTERM信号，进行必要的收尾处理，比如释放AI芯片、停止IO操作等，避免进程强行退出导致资源未释放、文件损坏等问题。
#注意: docker stop的设置的超时时间为5s，应用在5s内还未退出，docker进程及其内部应用都会被系统强行杀死。
trap clean TERM

#设置环境变量，确保软件能运行
export LD_LIBRARY_PATH=/home/3rdApp/lib:$LD_LIBRARY_PATH
#docker启动时，进入算法目录
cd /home/3rdApp
#执行算法运行程序，执行文件名字必须是main。这里需要运行在后台，后面死循环等待。
bin/main &

# 检查算法应用进程状态，应用进程不存在时脚本退出，由于脚本PID为1，因而脚本退出后，docker服务会认为该容器异常，会停止该docker容器
while true
do
    if has_main; then
        sleep 5
    else
        break
    fi
done
```

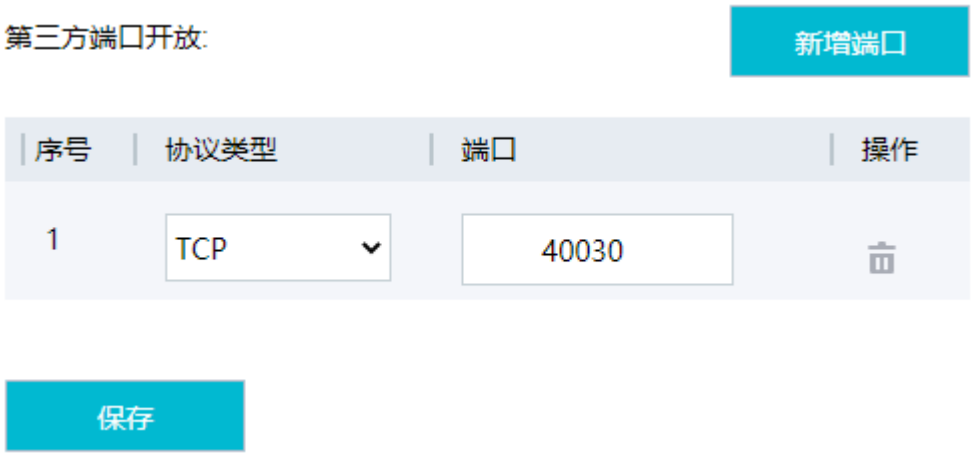

⚠ 注意

- 容器启动后，容器内main进程会被init_container.sh脚本拉起。对于**支持任务管理**的算法（参考**2.10 任务管理原理**），在main进程初始化时**请勿对昇腾310芯片进程初始化处理**，而是延后到收到第一个任务后对昇腾310芯片进程初始化。
- 相应地，当算法上所有分析任务停止后，需要释放昇腾310芯片资源，并重启容器。实现方式可以是先释放昇腾310芯片资源，然后退出main进程，main进程退出后，PID为1的init_container.sh进程也退出，容器退出。由于系统在创建容器时“--restart”参数被设置为“unless-stopped”，即该容器若异常退出，则会被系统自动拉起(用docker stop命令停止的容器除外)。因此，**若是main进程退出，系统会自动重启容器**。
- 算法正在使用昇腾310 AI加速芯片时，算法如果异常退出，很可能因为芯片未释放而导致下一次初始化芯片资源失败，因而在算法进程收到SIGTERM信号后务必进行优雅退出，至少需要结束昇腾310芯片上的任务，并对芯片进行去初始化处理。

2.8 端口开放策略

系统已经通过iptables对端口进行了限制，只开放必要的通信端口。算法如果需要额外开放端口与外界进行通信，则需要将对应的端口放通。

在系统的Portal页面(https://IP:8443/)中，提供了端口开放策略，可以配置40010~40030这个端口段的开放策略，建议算法或者应用也采用这个端口段，方便通过WEB开启防火墙策略。



2.9 参数配置原理

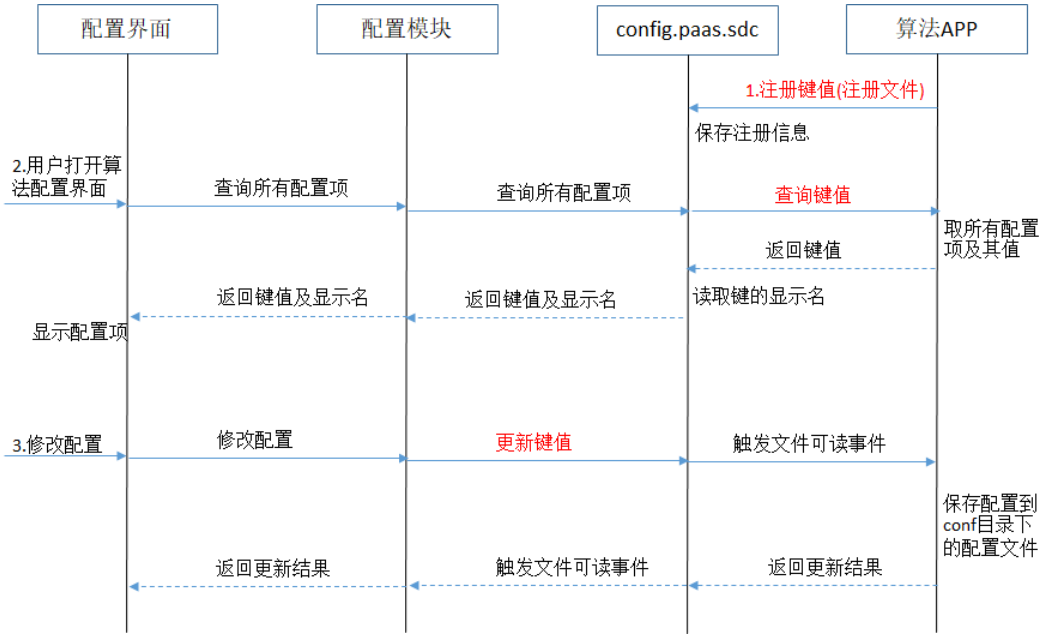
第三方算法安装后，往往需要配置一些运行参数。为了方便参数配置，系统在设备Portal和iClient S100客户端提供界面对算法参数进行配置。

若算法要支持参数配置，则需要实现参数配置服务化接口**config.paas.sdc**(参考**9.4.1 参数配置接口**)。

说明

本章节所述参数配置一般是需要算法对参数进行持久化的全局静态参数，一般在算法安装后进行配置。与分析任务关联的参数配置(如分析类型，ROI)一般是在进行**分析任务管理**时下发。

系统对第三方算法进行参数配置基本流程如下：



注册键值

算法APP在启动时，向配置服务注册键值，即告诉配置服务支持哪些配置项。

注册键值时，需要携带“注册文件”路径。

支持参数配置的算法APP应该包含如下几类配置文件(具体内容参考**config.paas.sdc服务化接口**的“注册键值”消息)：

序号	文件类型	文件内容	文件使用方式	容器内存储放路径	容器外映射路径
1	注册文件	算法配置基本信息。Scheme文件路径，配置项的多语言支持情况	算法“注册键值”时，携带注册文件的路径。config.paas.sdc服务将解析该文件中指向的sheme文件和多语言资源文件，读取所有配置项和配置项在配置界面的显示名称。	/home/3rdApp/conf	/opt/third_algorithm/run/appname/conf
2	Scheme文件	描述算法所支持的所有配置项			
3	多语言资源文件	各个配置项在各个语言环境下所显示的名称			

须知

1. 算法打包时应该将以上配置文件存放在baseconf目录下，算法安装后启动时，算法发现conf目录下没有这些文件的情况下（conf下有这些文件时还进行拷贝会导致原有配置被覆盖），从baseconf目录拷贝到conf目录下。具体原因可参考**2.5.2 算法APP目录规范**。
2. 算法APP调用“注册键值”接口时所携带的“注册文件”路径应该是容器外的路径，即：

   /opt/third_algorithm/run/appname/conf/filename， 其中

   - appname为算法名称
   - filename为注册文件名称

查询键值

用户打开设备portal或iClient S100的配置界面，对第三方算法进行参数配置时，系统会向config.paas.sdc配置服务查询所有的配置项。

配置服务向算法APP查询所有配置项的值。查询成功后，配置服务读取多语言资源文件中的参数显示名称，返回给配置界面。

更新键值

当用户在界面修改第三方算法配置项的值时，会通过配置服务的“更新键值”接口下发给算法APP，算法APP需要将配置项持久化到配置文件中(配置文件格式算法自行定义)，防止算法APP进程重启后配置丢失。

另外：

- 为了防止算法升级后配置丢失，需要把配置文件存放在/home/3rdApp/conf目录下，具体原因可参考**2.5.2 算法APP目录规范**。
- 升级场景下，系统会保留算法/home/3rdApp/conf目录下的配置，也就是配置文件还是老版本的。如果算法升级后配置文件也需要升级，则需要算法在首次启动时，自行检查并升级配置。

2.10 任务管理原理

基本原理

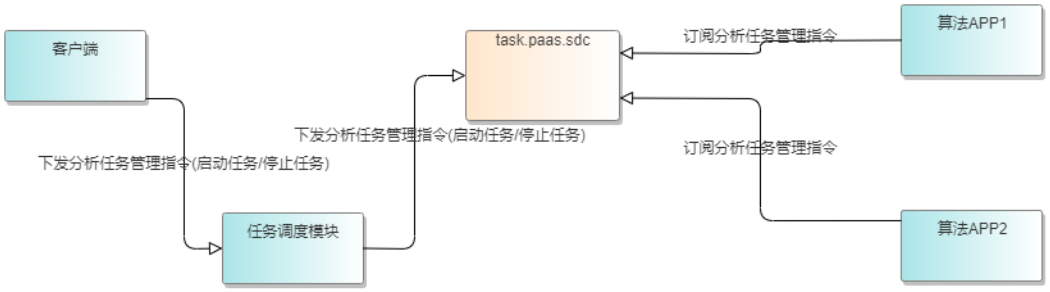
第三方算法分析任务管理是指用户可通过客户端(如iClient S100或基于产品北向Restful API开发的客户端或VMS平台)提供的界面，对第三方算法何时启动分析、对哪路视频进行分析、分析什么、怎么分析等进行管理。

系统任务调度模块和算法APP间通过task.paas.sdc服务接口(参考**9.4.2 任务管理接口**)来进行任务管理和状态同步。

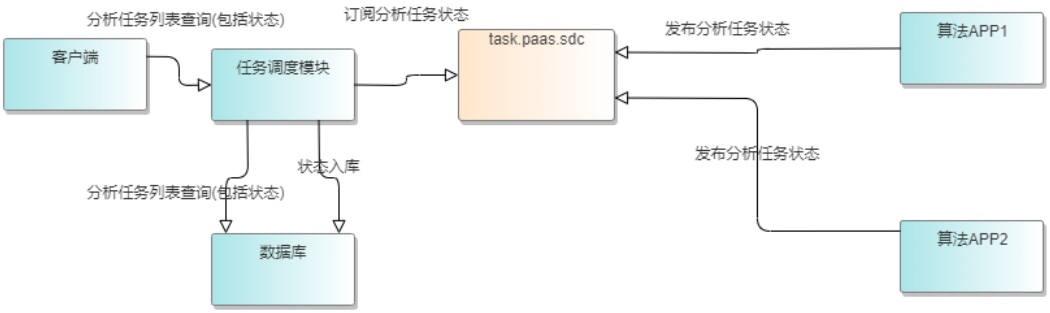
任务管理基本机制如下：

分析任务管理：从客户端下发的分析任务管理指令，最终会发布给task.paas.sdc服务，该服务根据分析任务对应的APP名称，查找到对应PID的算法APP，并将该指令推

送给该APP。因而，APP启动时需要订阅分析任务的管理消息，**并根据消息中的指令实现任务管理**。



分析任务状态同步：系统任务调度模块会订阅task.paas.sdc中的任务状态通知事件，将收取的任务状态维护到任务列表中。客户端会定时查询系统内的任务列表(包括状态)，实时显示任务状态到客户端界面。 因而，算法APP需要根据接口，**发布APP内所有任务状态到task.paas.sdc中**。



分析任务同步机制：任务调度模块将定时(30s)检查数据库中记录的分析任务状态，发现任务状态与预期不一致，则会进行一致性处理：

- 失败或待启动的任务：如果因摄像机离线、码流超时等异常原因导致任务启动失败，则会每5分钟尝试启动，对于状态未知或其他原因导致的失败一般会立即尝试启动。任务失败判断标准：
  - 算法APP在规定周期内明确反馈任务状态为失败。
  - 算法APP长时间(60s)未反馈任务状态，则系统认为状态未知。
- 停止失败的任务，会立即重新下发停止指令给算法APP，停止任务。

基于以上机制：

- 对于已经启动分析的任务也可能重复下发启动请求，已经结束的任务也可能再下发结束任务请求，对此类情况算法APP需做好容错处理。
- 算法APP，在分析任务发生变更时，采用单个上报方式上报该任务状态。另外，为了防止异常导致调度模块和算法间分析状态不同步，算法APP还需要定时(15s)批量上报所有分析任务的状态。

说明

对于支持任务管理的算法，请勿在算法启动时对AI芯片进行初始化操作，可以在收到第一个任务后对AI芯片进行初始化。另外，算法从有任务到无任务状态后，需要去初始化AI芯片，并退出进程。具体原因参考**2.7 算法启停原理**中的说明。

任务管理支持子分析类型设置

虽然当前一个D芯片上只能运行一个第三方算法，但一个算法内部可以支持多种分析类型(需要算法开发商实现)。比如可同时支持“安全帽检测”和“反光衣检测”。算法支持的分析类型可以在**config.xml配置文件**中进行说明。

用户使用客户端创建第三方算法分析任务时，会根据算法的能力，在界面展示出该算法支持的所有分析类型供用户选择。用户可选择进行单项分析(比如对摄像机A进行安全帽检测)，也可选择进行多种分析(比如对摄像机A同时进行安全帽检测和反光衣检测)。

任务管理支持 ROI/最大目标/最小目标/灵敏度设置

有些算法可能需要指定画面中感兴趣的区域(比如入侵检测)来辅助分析，有些算法可能需要划线（比如过线统计）来辅助分析，有些算法分析需要指定检测的最小目标和最大目标，也有些算法支持灵敏度配置。

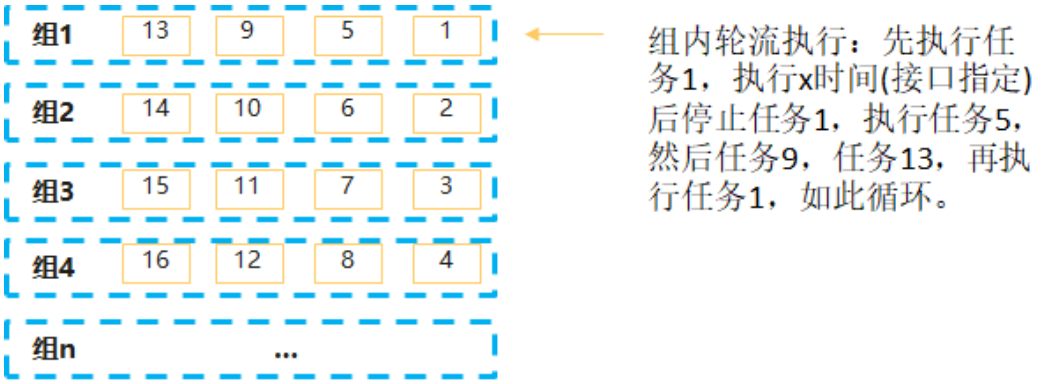
算法开发商可在算法的**config.xml配置文件**中对这些需求或能力进行配置。任务管理客户端会基于这些配置在任务管理时展现相应的界面给用户。比如支持区域设置，则创建分析任务时提供在实时预览画面上画区域的界面，支持划线，则提供在实时预览画面上划线的界面。

说明：对于灵敏度的设置，客户端界面固定支持设置范围为[0-100]， 算法内部如果不是这个范围，则需要算法内部按比例进行转换。

2.11 分时复用任务原理

设备上的AI算力是有限的，因而能支持的并发分析路数有限。一些特定场景下的第三方算法并不需要全程实时检测，例如国防光缆处动工检测，每隔一两分钟检测一次并不会造成漏检。这种场景下，进行全程检测让AI算力未达到最大化利用率。

分时复用任务调度参考CPU分时复用原理，分时间片来处理智能分析任务，用户体验上所有任务都在执行，实际上同一时刻同一组任务里只有一个任务在执行。实现上，系统将智能分析任务分组管理(每组最大4个任务)，组内进行轮询调度，因而智能分析路数可提升到4倍。



算法要支持分时复用任务方式首先要支持任务管理(参考**2.10 任务管理原理**)，并在算法APP安装包的**config.xml配置文件**中使能分时复用功能(loopAbility标签)。

当算法支持分时复用时，客户端在创建分析任务时会展示"轮询组号"和“轮询间隔”的设置，用户设置后，系统任务调度模块将分析任务下发给算法APP，在任务信息中携

带任**务所属的轮询组**和任务的执行时间，算法APP内部需要**自行根据轮询组实现分析任务的轮询**。

2.12 视频拉流原理

算法APP一般从本机拉视频流进行智能分析。系统提供了H.264和H.265视频流转发能力，并提供如下几种请流方式：

序号	拉流接口	描述	接口参考	备注
1	Restful接口+RTSP协议媒体库	通过设备北向自带的Restful接口获取RTSP的URL，然后通过RTSP协议协商建立媒体流。 为保证安全，系统每次返回的URL都不同，且30s内有效。因而每次需要重新拉流时需要重新获取URL，并在30s内发起RTSP协商。	**Restful接口**	**推荐方式。** 通过RTSP协商建立媒体流可使用产品提供的媒体库实现。媒体库可在论坛下载DEMO获取。
2	纯RTSP协议	算法APP直接使用RTSP协议协商建立媒体流。该方式需要每个通道的URL保持不变，因而需要在设备上配置RTSP URL为静态模式，方法如下： 1. 以**admin**用户登录OMU Portal。 2. 选择“系统管理 > 高级配置”。 3. 模块名称选择“MU”，搜索配置项。 4. 将IsConstantUrl 配置项的值修改为1。	-	该方式要求RTSP的URL配置为静态URL，**非安全模式，不推荐**。 RTSP拉流流程与标准流程有些差异： 1. 收流方发送 setup， 发流方回复200 OK。 2. 发流方发送announce，更新SDP媒体信息。收流方回复200 OK。 3. 收流方发送play消息，发流方回复200 OK。
3	eSDK接口	算法集成产品提供的eSDK组件，通过eSDK组件接口请流。	**eSDK接口**	eSDK组件已经不再维护，因而**不再推荐使用该方式**。

2.13 分析结果展示原理

系统提供了分析结果回传的服务化接口**event.paas.sdc**，算法APP可按该接口定义的规范回传分析结果，系统将会保存分析结果。系统还在北向提供了第三方算法智能分析数据的订阅和查询的Restful接口，实现智能业务闭环。

- 算法APP回传分析结果数据格式为TLV格式，具体参考**event.paas.sdc**服务化接口中的定义(**9.4.3 分析结果回传接口**)。

- 系统北向推送分析结果数据格式为Json格式，具体参考产品北向Restful接口(参考**9.2 Restful接口**)。

系统采用透明方式处理算法回传的数据，不会理解其中的内容，不过系统会区分元数据和告警数据(根据TLV中的METATYPE_MASK 字段区分)：

- 元数据：系统支持进行存储/推送/查询。
- 告警数据：除了当做告警进行存储/推送/查询外，还支持告警联动（ 例如告警联动录像、联动IO输出等 ）。

另外，iClient S100客户端已经实现了第三方算法智能分析结果的订阅和展示，可基于该客户端进行算法的调测。系统还支持算法自定义需要展示的字段，具体实现方式为：

- 在**config.xml配置文件**的OSDKeys配置项中指定需要显示的KEY名称(支持多个)。注意KEY名称是在回传的TLV结果中指定的检出目标或图片的属性名(键值对里的KEY)。
- 当系统通过Restful接口推送或查询分析结果时，发现算法回传的结果中有OSDKeys中指定的需要展示的属性，则会在数据中携带“needDisplay”标识。iClient S100客户端会基于needDisplay标识来决定是否显示该属性。

2.14 算法模型

设备携带了最多2块昇腾310芯片，用于AI计算，昇腾310暂时只支持OM算法模型，因而：

- 针对昇腾310训练的算法模型均可在该设备上使用，例如在华为ATLAS系列产品上使用的算法模型，可直接移植。
- 其他类型的算法模型，需要进行转换，转换方法同计算产品线的ATLAS产品，具体参考**9.1.2 模型转换指导书**。

说明

9.1.0及之前版本设备内置的DDK驱动还是Matrix架构，未升级到ACL架构，因而只支持Matrix架构下支持的算子及模型。

3 算法 DEMO 小试

3.1 DEMO简介
3.2 准备演示环境
3.3 算法下载/部署
3.4 算法演示

3.1 DEMO 简介

第三方算法DEMO是为了方便三方算法厂家快速上车，熟悉系统南/北向接口的调用流程，故提供DEMO供三方算法厂家参考。

3.2 准备演示环境

准备一台安装有9.1.0 或者更高版本的IVS1800设备环境

3.3 算法下载/部署

3.3.1 在线部署

前提条件

- 已经在S100 9.1及以上版本添加需安装算法的IVS1800设备。
- 已经申请华为云账号，并进行实名认证，具体操作参见**账号注册&认证**。

限制条件

- IVS1800能与算法商城的网络互连。
- IVS1800能与S100的网络互连。

操作步骤

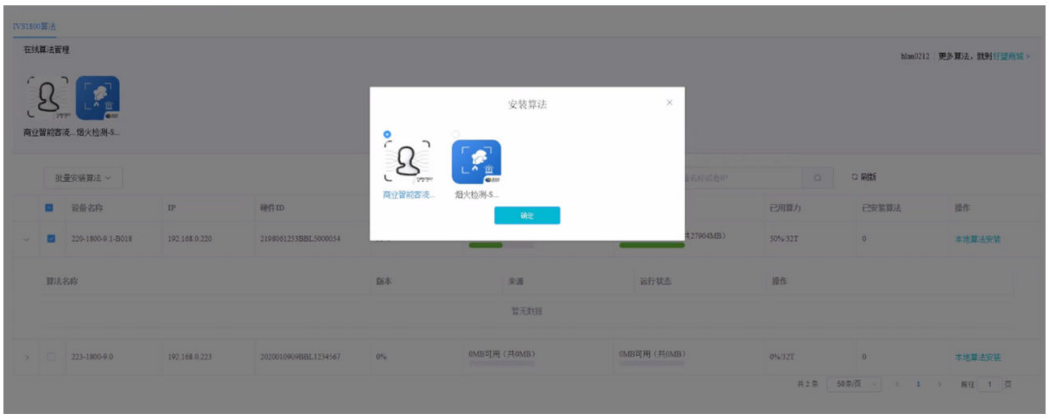
1. 在S100首页中，选择“维护与管理 > 算法管理 ”。

2. 单击“立即登录”，使用华为云账号或IAM用户登录好望商城

3. 登录成功后，“在线算法管理”栏将展示该账号已购买的算法。单击算法图标可查看

4. 勾选设备，单击“批量安装算法”，选择“在线算法”

5. 选择待安装的算法，单击“确定”开始安装。



3.3.2 离线部署

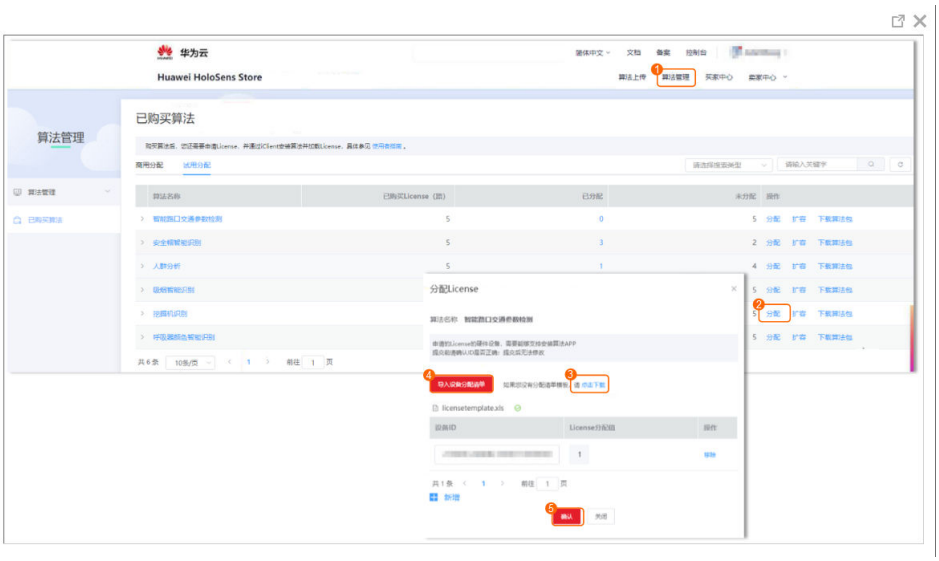
具体请参考《IVS1800 9.0.0 产品文档》。

3.3.2.1 下载算法及 License 文件

操作步骤

1. 使用浏览器访问**算法商城**。
2. 单击界面右上角的“登录”，使用华为云账号和密码登录算法商城。
   如果您尚未申请华为云账号，请先参考**账号注册&认证**进行申请，并完成实名认证。
3. 购买算法。
   a. 单击“进入商城”。
   b. 根据行业分类或关键字搜索所需算法，根据界面提示完成算法购买。
      算法提供免费试用，购买算法前，您可以先安装试用版本算法包，提前体验算法功能。
4. 下载算法包和License。
   a. 在好望商城首页，单击“进入商城”。
   b. 单击“算法管理”，进入算法管理页面。
   c. 单击算法后的“分配”，并配置License信息。

图 3-1 配置 License 信息



设备分配清单中需要配置IVS1800设备的SN号。SN号可以通过如下途径获取：

- 查看贴在设备上的SN标签。
- 登录IVS1800系统进行查询：
  1) 以**root**用户登录操作系统。
  2) 查看SN号。
     elabel get 0x11

d. 下载算法包和License文件，如图2所示。

图 3-2 下载算法包和 License 文件

- 后缀为tar.gz的文件为License文件，如“digger_app_1.0.0_trial.tar.gz”。
- 后缀为tar的文件为算法包，如“digger_app-v1.2-1.aarch64.tar”。

3.3.2.2 上传算法及 License 文件

操作步骤

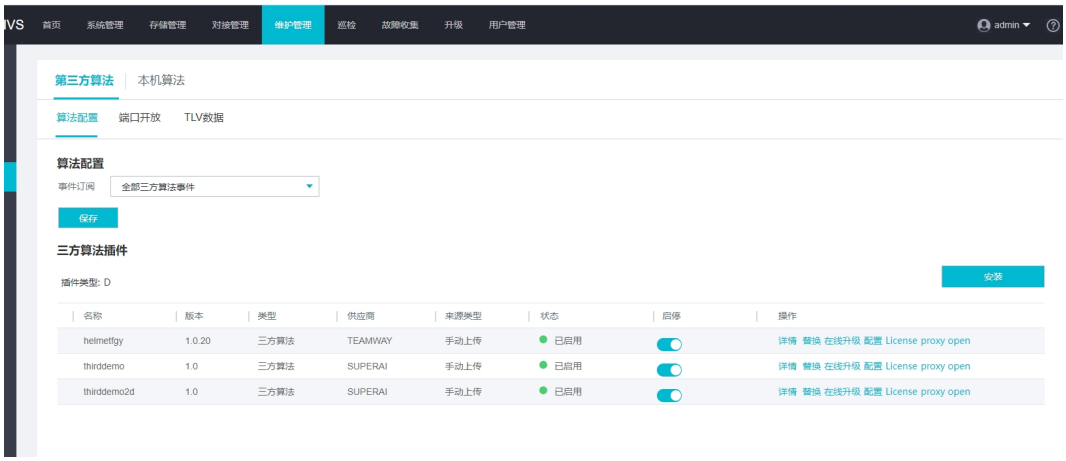
1. 以**admin**用户登录OMU Portal。
2. 安装算法插件。
   a. 选择“系统管理 > 三方算法插件”。
   b. 上传算法插件，如图1所示。
   OMU Portal会自动根据当前设备型号显示待加载插件类型。
   - “插件类型”为“A”时，表示算法加载运行于Hi3559A。
   - “插件类型”为“D”时，表示算法加载运行于昇腾310。

图 3-3 上传算法插件

c. 单击“安装”，安装第三方算法。
安装算法后，“状态”变为“安装成功”。
d. 单击“License”，导入License文件。

3.4 算法演示

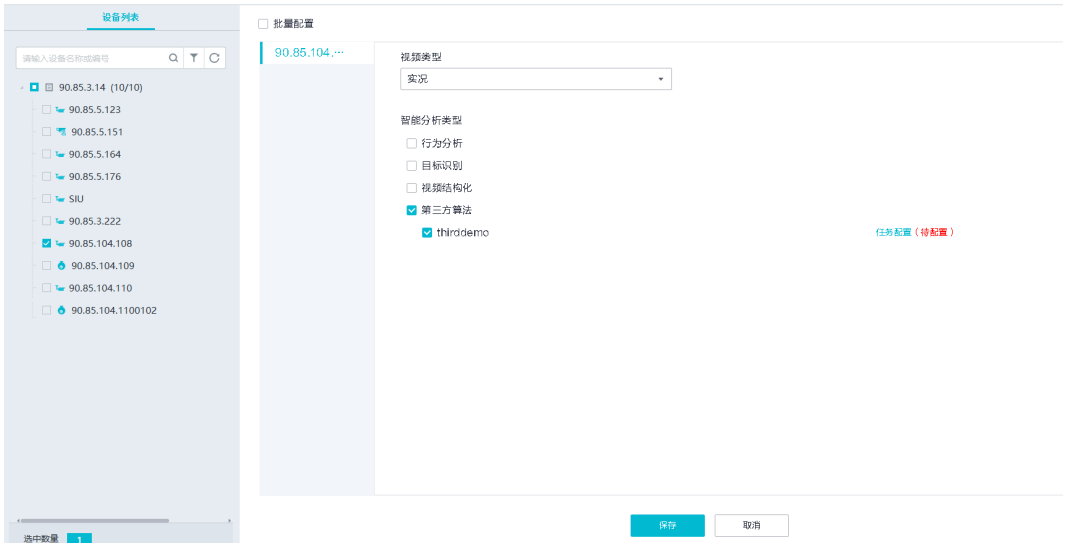
3.4.1 查看算法状态



omu portal上，可以查看已经安装的第三方算法的状态。

3.4.2 创建分析任务

- 在iClient s100上选择相机和指定三方算法



- 指定三方算法分析类型

- 指定算法规则

- 设置时间计划

3.4.3 查看分析结果

在iClient s100的告警中心里，可以看到第三方算法上报的告警抓拍图

4 编译环境准备

设备采用ARM架构芯片，因而编译出的算法二进制程序必须是基于ARM指令集，需要在ARM服务器上进行编译。为了方便开发者，设备还提供了基于X86环境进行交叉编译的工具链，因而算法支持X86交叉编译和ARM直接编译两种方式。

4.1 X86交叉编译环境

4.2 ARM直接编译环境

4.3 DDK目录结构

4.4 软件开发依赖的头文件和链接库

4.5 软件开发编译工具链

4.6 开发工具

4.1 X86 交叉编译环境

4.1.1 硬件要求

项目	规格
CPU	X86服务器的CPU要求64位。
内存	DDR4 32GB以上
硬盘大小	100G以上。

4.1.2 软件要求

内容	项目	版本号	备注
操作系统及编译器	Ubuntu Linux	18.04 LTS	更高的版本未验证，如果有需求，请自行验证。
	GLIBC	2.27	

内容	项目	版本号	备注
	Gcc version	7.5.0	
操作系统及编译器	SUSE Linux	4.12.14-120-default	
	GLIBC	2.22	
	Gcc version	7.3.0	
DDK	DDK/驱动	C32B100	

说明

操作系统内编译要安装有cmake、make相关工具。

- 输入“cmake --version”输出类似cmake version 3.14.1，表示cmake已经安装。若没有，需要自行下载并安装cmake工具。
- 输入“make --version”输出类似GNU Make 4.2.1，表示make已经安装。若没有，需要自行下载并安装make工具。

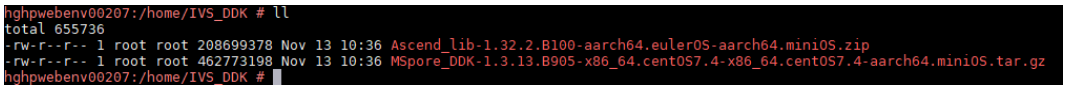
软件包介绍

名称	压缩包	说明
DDK开发包文件	**MSpore_DDK-1.3.13.B905-x86_64.centOS7.4-x86_64.centOS7.4-aarch64.miniOS.tar.gz**	算法开发工具包，包括开发调试工具，头文件，依赖库和Device端编译工具链
驱动文件	**Ascend_lib-1.32.2.B100-aarch64.eulerOS-aarch64.miniOS.zip**	驱动文件

4.1.3 搭建开发环境

1）在编译服务器上创建DDK目录，建议在/home/下创建IVS_DDK。

2）将Ascend_lib-1.32.2.B100-aarch64.eulerOS-aarch64.miniOS.zip和MSpore_DDK-1.3.13.B905-x86_64.centOS7.4-x86_64.centOS7.4-aarch64.miniOS.tar.gz上传到编译服务器上，并且拷贝至/home/IVS_DDK文件夹中。



3）进入到/home/IVS_DDK文件夹内，解压MSpore_DDK-1.3.13.B905-x86_64.centOS7.4-x86_64.centOS7.4-aarch64.miniOS.tar.gz

tar -zxvf MSpore_DDK-1.3.13.B905-x86_64.centOS7.4-x86_64.centOS7.4-aarch64.miniOS.tar.gz

解压后会生成check_sha.sh，install.sh，ddk.tar.gz三个文件

4）继续解压ddk.tar.gz文件

tar -zxvf ddk.tar.gz

解压后，内容如下：

```
hghpwebenv00207:/home/IVS_DDK # ll
total 1128488
-rw-r--r--  1 root root 208699378 Nov 13 10:36 Ascend_lib-1.32.2.B100-aarch64.eulerOS-aarch64.miniOS.zip
-rw-r--r--  1 root root 462773198 Nov 13 10:36 MSpore_DDK-1.3.13.B905-x86_64.centOS7.4-x86_64.centOS7.4-aarch64.miniOS.tar.gz
-r-xr-x---  1 root root       459 May 10  2019 check_sha.sh
drwx------ 13 root root      4096 May 10  2019 ddk
-rw-rw-r--  1 root root 484074066 May 10  2019 ddk.tar.gz
-rwxrwxr-x  1 root root      3073 May 10  2019 install.sh
```

4）解压Ascend_lib-1.32.2.B100-aarch64.eulerOS-aarch64.miniOS.zip。为了避免混乱，新建Ascend_lib文件夹，将Ascend_lib-1.32.2.B100-aarch64.eulerOS-aarch64.miniOS.zip移至Ascend_lib中解压。

hghpwebenv00207:/home/IVS_DDK # mkdir Ascend_lib

hghpwebenv00207:/home/IVS_DDK # mv Ascend_lib-1.32.2.B100-aarch64.eulerOS-aarch64.miniOS.zip Ascend_lib

hghpwebenv00207:/home/IVS_DDK # cd Ascend_lib

hghpwebenv00207:/home/IVS_DDK # unzip Ascend_lib-1.32.2.B100-aarch64.eulerOS-aarch64.miniOS.zip

5)将ddk中host依赖库替换为arm版本。

1、进入ddk/lib路径下，备份原x86_64-linux-gcc4.8.5文件夹（ mv x86_64-linux-gcc4.8.5 x86_64-linux-gcc4.8.5_bk ）；

2、将步骤4中生成的Ascend_lib/EP/host-aarch64.eulerOS/lib文件夹拷贝至ddk/lib路径下，重命名为x86_64-linux-gcc4.8.5（实质是用Ascend_lib/EP/host-aarch64.eulerOS文件夹替换掉ddk/lib/x86_64-linux-gcc4.8.5）(cp -a ../../Ascend_lib/EP/host-aarch64.eulerOS/lib x86_64-linux-gcc4.8.5)

```
hghpwebenv00207:/home/IVS_DDK/ddk/lib # ll
total 12
dr-xr-x--- 2 root root 4096 May 10  2019 aarch64-linux-gcc6.3
dr-xr-x--- 3 root root 4096 Oct 15  2020 x86_64-linux-gcc4.8.5
dr-xr-x--- 3 root root 4096 May 10  2019 x86_64-linux-gcc4.8.5_bk
hghpwebenv00207:/home/IVS_DDK/ddk/lib #
```

6）配置环境变量DDK_HOME

export DDK_HOME=/home/IVS_DDK/ddk

须知

上述设置只会对当前登录有效。需要根据不同的操作系统，将环境变量设置到启动中，确保其他登录有效。比如ubuntu版本中，需要在bashrc文件中增加一行: export DDK_HOME=/home/IVS_DDK/ddk

4.2 ARM 直接编译环境

4.2.1 硬件要求

项目	规格
CPU	ARM服务器的CPU要求64位。
内存	DDR4 32GB以上
硬盘大小	10G以上。

需要提供ARM编译服务器。

服务器	方式
使用华为的硬件服务器	申请购买华为ARM服务器，推荐使用TaiShan100服务器。
申请华为编译云服务器	https://www.huaweicloud.com/devcloud/，购买华为开发平台

4.2.2 软件要求

内容	项目	版本号
操作系统及编译器	EulerOS（ARM）	V2R9
	GLIBC	2.17
	Gcc version	4.8.5
DDK	DDK/驱动	C32B100

说明

操作系统内编译要安装有cmake、make相关工具。

- 输入“cmake --version”输出类似cmake version 3.14.1，表示cmake已经安装。若没有，需要自行下载并安装cmake工具。
- 输入“make --version”输出类似GNU Make 4.2.1，表示make已经安装。若没有，需要自行下载并安装make工具。

软件包介绍

名称	压缩包	说明
DDK开发包文件	Ascend_DDK-1.32.2.B100-1.1.1-aarch64.eulerOS.tar.gz	IVS1800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

名称	压缩包	说明
驱动文件	Ascend_lib-1.32.2.B100-aarch64.eulerOS-aarch64.miniOS.zip	驱动文件

4.2.3 开发环境

前提条件

- 已参考DDK获取方式并获取了软件版本。

平台要求

需要ARM平台，欧拉系统（和host侧欧拉系统保持一样），开通外网，可以SSH远程连接和PIP网络安装。

操作步骤

步骤1 准备一台ARM CPU的服务器（或者PC）用作开发主机，为开发主机安装好EulerOS操作系统。

如果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EulerOS操作系统用于开发。

步骤2 配置开发主机网络，确保开发主机和IVS1800在同一个局域网内。

步骤3 获取如下两个包，如未提供该包，获取方式请咨询相应人员

Ascend_DDK-1.32.2.B100-1.1.1-aarch64.eulerOS.tar.gz

Ascend_lib-1.32.2.B100-aarch64.eulerOS-aarch64.miniOS.zip

步骤4 把两个包放到同一个文件夹。

如/Home/HiAI/(自定义)，进入/Home/HiAI/，然后解压 **Ascend_DDK-1.32.2.B100-1.1.1-aarch64.eulerOS.tar.gz**

解压命令：**tar zxvf Ascend_DDK-1.32.2.B100-1.1.1-aarch64.eulerOS.tar.gz**

步骤5 解压完成后直接在该目录执行如下命令：

python ddk_setup.py install /Home/HiAI/(ddk的安装路径) ./ Ascend_lib-1.32.2.B100-aarch64.eulerOS-aarch64.miniOS.zip

步骤6 成功安装DDK，如图1所示。

如果过程中提示缺少库，比如numpy，直接pip或者yum安装。

图 4-1 安装成功

```
[root@Localhost ddk]# python ddk_setup.py install /home/HwHiAiUser/ /root/source/ddk/Ascend_lib-1.31.T15.B150-aarch64.centOS7.6-aarch64.miniOS.zip
2020-01-20 17:06:26 - [INFO] lib package input check pass
2020-01-20 17:06:26 - [INFO] start to process check_sha_module command
files check pass!
2020-01-20 17:06:27 - [INFO] installing check_sha_module process is finished.
2020-01-20 17:06:27 - [INFO] /home/HwHiAiUser/ddk does not exist, no need to remove
2020-01-20 17:06:41 - [INFO] Decompressing ddk.tar.gz is finished.
2020-01-20 17:06:41 - [INFO] start to process te_module command
2020-01-20 17:06:42 - [INFO] installing te_module process is finished.
2020-01-20 17:06:42 - [INFO] start to process hiai_module command
2020-01-20 17:06:42 - [INFO] installing hiai_module process is finished.
2020-01-20 17:06:42 - [INFO] no need to create symlink for linux.
2020-01-20 17:06:42 - [INFO] lib version check pass!
2020-01-20 17:06:48 - [INFO] start to process check_sha_module_lib command
files check pass!
2020-01-20 17:06:49 - [INFO] installing check_sha_module_lib process is finished.
2020-01-20 17:06:49 - [INFO] import lib finished
Successfully installed the DDK!
```

----结束

4.3 DDK 目录结构

安装好DDK后，DDK安装目录结构如下：

```
.
├─── bin
├─── conf
├─── include
├─── lib
├─── packages
├─── scripts
├─── toolchains
├─── Euler_compile_env_cross
├─── aarch64-linux-gcc7.3.0
└─── uihost
```

表 4-1 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-gcc7.3.0”目录。
host	该目录包含在开发主机上使用的工具和lib库。
include	该目录包含软件开发所需要的头文件。
lib	该目录包含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Host侧程序依赖的链接库文件。
packages	-

目录名称	目录说明
sample	-
scripts	-
toolchains	该目录包含交叉编译工具链。
toolchains/aarch64-linux-gcc7.3.0	该目录包含Device侧的交叉编译工具链。
uihost	-

4.4 软件开发依赖的头文件和链接库

- 头文件

  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DDK安装目录下的“include”目录里。“include”目录结构如表4-2下：

  表 4-2 头文件说明

头文件	说明
include/inc	该目录包含业务功能头文件。
include/libc_sec	该目录包含安全函数头文件。
include/ third_party	该目录包括第三方库头文件。

- 程序开发的链接库

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

  - Host侧链接库文件位于DDK安装目录下的“lib64”目录
    - lib64目录下无子目录，包含了所有在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device侧需要用到的链接库文件及其软链接。

4.5 软件开发编译工具链

- Host 软件开发编译工具链
  - host侧CPU使用的是Hi3559A，Hi3559A是Armv8-a架构的64位CPU。

当前没有可以在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设备上运行。

- Device 软件开发编译工具链
  - 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。

    当前没有可以在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host上，程序运行时，再将对应的可执行文件或动态库拷贝到 device侧上运行。
  - device软件开发编译工具链位于DDK安装目录的“toolchains/ aarch64-linux-gcc7.3.0/bin”路径下。

4.6 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 4-3 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《 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《 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。

5 算法 DEMO 编译打包

Demo下载链接**https://bbs.huaweicloud.com/forum/forum.php?mod=viewthread&tid=168454**

5.1 DEMO逻辑架构

5.2 DEMO代码目录

5.3 DEMO编译

5.4 DEMO打包

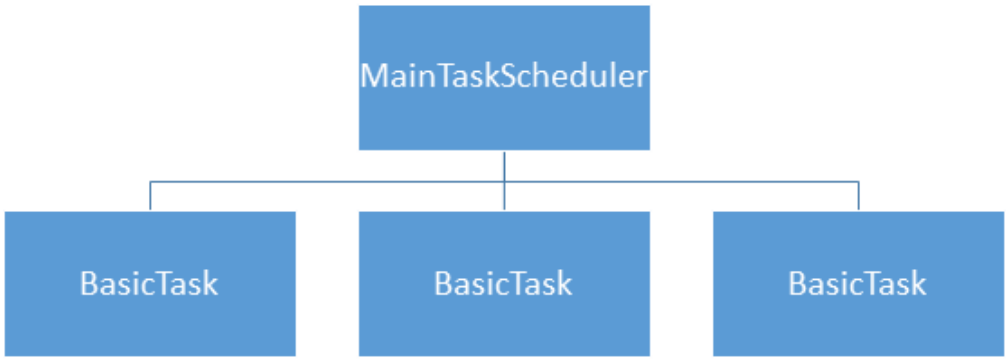
5.5 认证测试

5.6 算法包签名

5.7 算法包上线算法商城

5.1 DEMO 逻辑架构

1. 整个程序的入口在Main.cpp中的main函数；
2. MainTaskScheduler类实现了任务接受和调度的功能；
3. BasicTask类为智能分析任务的功能实现，表示一个从某个相机的拉流智能分析业务；
4. MainTaskScheduler可以管理多个BasicTask，对应于ivs1800智能分析业务的启停动作；



5. CVideoInputBase为媒体输入的抽象类，负责为算法提供视频流输入，其来源可以是存在本地路径下的裸码流文件，也可以是通过系统北向Restful接口获取的rtsp url路径。其中通过camera code获取rtsp url的restful接口参考：**9.2 Restful接口**。
6. AnalyzeInterface目录下，主要是算法的实现。为了尽可能简明扼要地展示三方算法如何与系统对接，我们基于昇腾310芯片开发套件中的InferObjectDetection识别算法demo进行改造，在va_analyze_interface.h头文件中封装了一套通用接口。ISV需要在算法开发中遵循该接口实现，其他业务代码可以不做修改。

   1）VCM_Plugin_Set_LogFunc 设置日志回调函数，可以将日志打印到公共日志目录。

   2）VCM_Plugin_Initialize初始化接口，整个进程起来之后只初始化一次，用于分配算法所需的各种资源，创建graph等。

   3）VCM_Plugin_Create，创建分析通道，每一路分析产生一个通道，框架往该通道送帧数据，算法把分析结果从此通道中回调返回。

   4）VCM_Plugin_Analyze_Stream，通过该接口向算法送帧数据，不同摄像头拉的流通过VCM_Plugin_Create返回的通道号进行区分。

   5）VCM_Plugin_Close，当某一路分析需要关闭时，调用该接口关闭通道，释放相关资源。

   6）VCM_Plugin_Release，整个进程退出时，调用该接口，释放D卡计算资源。
7. SrvfsConfigService目录下，用于任务下发和上报的相关业务代码实现。
8. SrvfsEventSubAndPub 目录下，算法结果回传给系统的相关业务代码实现。

5.2 DEMO 代码目录

一级目录	二级目录	说明
Demo		

	AnalyzeInterface	**基于atlas开源示例代码，做接口适配**
		1、调用昇腾应用层开放api的demo代码。在atlas源代码基础上，结合IVS业务修改适配。
		2、不包括昇腾驱动本身api实现
		3、原始实现已经开源， gitee地址：https://gitee.com/HuaweiAtlas/samples/tree/master/Samples
	AtlasCommon	**基于atlas开源示例代码，做接口适配**
		1、调用昇腾应用层开放api的demo代码。在atlas源代码基础上，结合IVS业务修改适配。
		2、不包括昇腾驱动本身api实现
		3、原始实现已经开源， gitee地址：https://gitee.com/HuaweiAtlas/samples/tree/master/Samples
	build	编译脚本
	commonheaders	开源库的头文件
	commonlibs	依赖的开源库的二进制lib库和平台库头文件
	config	程序所需配置文件
	log	日志模块头文件
	model	昇腾提供的模型转换样例，gitee地址：
		https://gitee.com/HuaweiAtlas/samples/tree/master/Samples/InferObjectDetection/data/models
	out	昇腾matrix框架配置文件
	SrvfsConfigService	实现算法参数配置功能Demo代码
	SrvfsEventSubAndPub	任务下发API，结果上报API 调用的demo代码
	SrvfsInc	系统开放的用于算法开发的服务化接口api头文件
	TaskMgr	基于任务管理API（task.pass.sdc）开发的demo代码
	VideoInput	基于媒体库API（mpsdk）开发的视频流拉取demo代码
Cmake		编译脚本

5.3 DEMO 编译

Demo下载链接**https://bbs.huaweicloud.com/forum/forum.php?mod=viewthread&tid=168454**

5.3.1 X86 交叉编译方式

1. 设置环境变量，export DDK_HOME={DDK的安装环境}
2. 执行编译脚本：sh build.sh x86

5.3.2 ARM 直接编译方式

1. 设置环境变量，export DDK_HOME={DDK的安装环境}
2. 执行编译脚本：sh build.sh arm

5.4 DEMO 打包

5.4.1 自定义算法名称

- 算法名称需要在打包的时候写入config.xml中的<appname>节点中

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <appname>thirddemo</appname>
```

- 用于显示用户可配置内容的资源文件config.ini也要做相应更改：

```
name=thirddemo
desc=config_service
param=./conf/scheme.json

[language]
language.ZH=./conf/langZH.ini
language.EN=./conf/langEN.ini
language.FR=./conf/langFR.ini
language.ES=./conf/langES.ini
```

- 向系统注册config.ini的路径也因算法名称会出现变化：

/opt/third_algorithm/run/thirddemo/conf/config.ini

- 发送结果时TLV的META_NAME = 0x20000002元数据和PRODUCER_NAME = 0x20000003元数据也要做相应修改
- 上报任务状态的event name也需要改成相应算法名称

```
struct sdc_common_head *head = (sdc_common_head *)buf;
head->version = SDC_VERSION;
head->method = SDC_METHOD_CREATE;
head->url = SDC_URL_TASK_STATUS;
head->head_length = sizeof(*head);
struct sdc_task_event *event = (sdc_task_event *)&buf[head->head_length];
strcpy(event->name, VA_APP_NAME); // 消息名称（app名称）      You, 3 months ago • 
event->length = 0;
head->content_length = sizeof(*event) + event->length;

LOGI("InnerReportTaskStatus enter");
int nret = write(fd, buf, uBufBaseLen);
```


5.4.2 打包

算法开发完毕后，可以直接copy到IVS1800产品盒子上进行调试。但是在运行的时候，考虑到和现有的系统进行隔离，需要运行在容器中。

1. 查看镜像

   后台登录IVS1800系统，使用docker指令查看基础镜像是否存在：

   如果没有看到基础镜像(ivsbase)，则需要导入基础镜像。基础镜像的导入参考**“获取容器基础镜像”**。

2. 文件准备

   第三方demo打包依赖的文件夹名称如下：

   ```
   Euler:/opt/WX977453 # ll
   total 296
   drwxr-xr-x 6 root root   4096 Aug  4 15:48 app
   -rw------- 1 root root    596 Aug  4 15:51 Dockerfile
   -rwxr-xr-x 1 root root 142784 Jul 18  2020 groupadd
   -rw-r--r-- 1 root root    379 Aug  4 15:47 init_container.sh
   -rwxr-xr-x 1 root root 143200 Jul 18  2020 useradd
   ```

   其文件内容为：

   1）app：原始工程文件夹，包含算法工程运行依赖的可执行文件，配置文件，库文件等。

   2）Dockerfile：dockerfile文件，用于生成镜像。内容参考**“容器镜像制作”**章节。

   3）groupadd：镜像制作依赖文件，从本文附件中获取。

   4）useradd：镜像制作依赖文件，从本文附件中获取。

   5）init_container.sh：容器启动时的执行脚本，用于启动容器中算法进程等，内容参考**2.7 算法启停原理**章节。

3. 制作流程说明

   a. 镜像制作

      新建thirddemo文件夹，将附件<用户管理附件>中useradd和groupadd放到目录下。

      mkdir thirddemo

      为useradd和groupadd赋上执行权限

      chmod 755 useradd groupadd

      将app，Dockerfile，init_container.sh拷贝到thirddemo文件夹中，进入文件加，执行指令：

      docker build -t thirddemo:v1.0 .

      注意：这里后面的"."不能缺少, 且“.”之前需要一个空格。

      等待完成后，通过镜像命令查看，发现多了一个thirddemo镜像。

b. 容器创建

执行下述指令进行容器创建：

docker create -it --device=/dev/davinci_manager --device=/dev/hisi_hdc --device=/dev/davinci0 --device /dev/cache:/dev/cache --name=thirddemo --log-opt max-file=2 --log-opt max-size=30KB --memory 1024MB --cpus=1.0 --ulimit nofile=65535:65535 --net=host --user HwHiAiUser --privileged=false --cpuset-cpus="2,3" --pids-limit 100 --security-opt apparmor=docker-default,no-new-privileges,label=level:TopSecret --cap-drop NET_RAW -v /var/dlog:/var/dlog -v /usr/slog:/usr/slog -v /mnt/srvfs:/mnt/srvfs -v /tmp:/tmp -e http_proxy_connect=http_proxy_connect.socket -e CREATEBYOMU=1 thirddemo:v1.0 /bin/bash -c '/home/init_container.sh'

具体参数含义可参考**“容器镜像制作”**章节。

c. 调试容器

测试阶段，使用指令启动刚刚创建的容器

启动命令：docker start -i CONTAINER ID（通过ps查看的ID）

启动后，进入容器中进行调试程序，指导程序能正常运行。

进入容器指令：docker exec -ti CONTAINER ID bash

d. 导出容器

docker export CONTAINER ID -o thirddemo-1.0.tar

CONTAINER ID：生成的容器ID

thirddemo-1.0.tar: 导出的容器镜像文件的名字

本例导出命令为：

docker export c59fca21c131 -o thirddemo-1.0.tar

e. 算法打包

将导出的thirddemo-1.0.tar文件与config.xml一起打包。config.xml的作用和详细描述参考**2.5.4 config.xml配置文件**中的说明。

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <appname>thirddemo</appname>
    <version>1.0</version>
    <vendor>superAI</vendor>
    <d_num>1</d_num>
    <cpu>100</cpu>
    <mem>1000</mem>
    <channel>8</channel>
        <!--是否支持任务管理及分析任务状态反馈, 如果实现了task.paas.sdc提供的服务，则填1，否则为0,默认为0 -->
    <mgrEnabled>1</mgrEnabled>
        <!--算法能力，此标签有效字符串长度请勿超过4096 -->
    <capabilities>
        <loopAbility>
                <!--是否支持分时复用，轮询调度，0-不支持，1-支持-->
            <enable>1</enable>
                <!--分时复用支持的单路执行周期范围，默认3到3600秒，最大范围[0.05,36000]-->
            <minInterval>3</minInterval>
            <maxInterval>3600</maxInterval>
        </loopAbility>
            <!--算法分析能力,单个算法限定最大10种分析类型-->
        <analyzeAbilities>
            <analyzeAbility>
                    <!--分析类型1-->
                <analyzeType>移动物体检测1</analyzeType>
                    <!--ROI类型,0-无，1-线段，2-区域-->
                <roiType>2</roiType>
                    <!--该分析类型支持的ROI数量-->
                <roiNumber>2</roiNumber>
                    <!--是否支持分析灵敏度配置(如果支持，范围固定为[0-100]),0-不支持，1-支持-->
                <sensitivity>1</sensitivity>
                    <!--可支持最小检测目标大小，填写宽高更小值，不限时填0-->
                <minObjSize>0</minObjSize>
                    <!--可支持最大检测目标大小，填写宽高更小值，不限时填0-->
                <maxObjSize>0</maxObjSize>
                    <!--需要在场景图上叠加显示的内容-->
                <osdKeys>移动物体检测1</osdKeys>
            </analyzeAbility>
            <analyzeAbility>
                    <!--分析类型2-->
                <analyzeType>移动物体检测2</analyzeType>
                    <!--ROI类型,0-无，1-线段，2-区域-->
                <roiType>2</roiType>
                    <!--该分析类型支持的ROI数量,roitype为非0时有效-->
                <roiNumber>2</roiNumber>
                    <!--是否支持分析灵敏度配置(如果支持，范围固定为[0-100]),0-不支持，1-支持-->
                <sensitivity>1</sensitivity>
                    <!--可支持最小检测目标大小，填写宽高更小值，不限时填0-->
                <minObjSize>0</minObjSize>
                    <!--可支持最大检测目标大小，填写宽高更小值，不限时填0-->
                <maxObjSize>0</maxObjSize>
                    <!--需要在场景图上叠加显示的内容-->
                <osdKeys>移动物体检测2</osdKeys>
            </analyzeAbility>
        </analyzeAbilities>
    </capabilities>
</config>
```

将镜像文件和config.xml文件放到同一目录下，然后执行命令生成ivsapp-1.0.tar.gz算法包。

tar zcvf thirddemo-1.0.tar.gz thirddemo-1.0.tar config.xml

5.5 认证测试

交付件上传格式符合： 一级目录：公司名 二级目录：算法名称+日期 ，更新说明，算法商城测试验收标准3.0 三级目录：License(包含最新的设备ID)、算法包、用户手册、自验证视频（即自测试通过视频）、原始视频素材、病毒扫描报告、场景视频矩阵（若无法在提交验收时自行完成场景用例设计，可在提交后联系算法验收团队协助设计并完成自验证）

使用手册要按照模板要求填写，应用场景描述清晰，场景示例截图与应用场景保持一致，内容不能遗漏，需包含以下几点(规格指标、各特性配置操作指导、RESTful API接口均有使用说明)。备注：检查时注意目标像素大小要求全文一致、语法检查通过，无错词错字
厂商需填写《 物料网络安全需求规格书模板（ 业务应用类软件 ）》文档。 针对安全扫描项 必须修改项： 1、 开源软件漏洞扫描 2、 安全编译选项检查 可选项：对于厂家可控制的代码，需要进行安全整改，对于厂家不可控制的，例如第三方开源代码问题，需要厂家统一通过邮件回复，测试已问题单的形式承载。 1、 安全配置检查 2、 密钥和信息泄露扫描 3、 典型编码问题检查 安全扫描网址：https://secbinarycheck.rnd.huawei.com/（ 该网址不对外开放，可提交给华为认证团队扫描并回传报告 ）。 备注：安全扫描后，算法如果需要修改，由算法团队指导厂商修改。
扫描无风险
所有场景都需要有对应的厂商自验证视频，缺陷修复后也需包含厂商自验证视频。
使用自检工具校验： 1.license通过jar包生成，校验jar包 2.license通过其他方式生成，校验license 将校验成功截图放入checklist中的自检结果，随交付资料一起交付
临时license内容、字段命名、大小写、特殊符号、单词拼写均符合license规范
临时license导入成功，且算法立即生效
临时license导入成功，且运行1h后，目标检测框显示正常
临时license导入成功，频繁启停算法，算法开启后立即生效
临时license导入成功，且试用期到期后，算法不检测，界面无检测框
商用license内容、字段命名、大小写、特殊符号、单词拼写均符合license规范
商用license导入成功，且算法立即生效
商用license导入成功，且运行3*24h后，目标检测框显示正常，没有内存泄露等常见问题
商用license导入成功，频繁启停算法，算法开启后立即生效
多目标画面的检测框不漂移
多目标画面的检测框连续
在算法生效后，画面中应立即出现目标检测框，建议将算法生效时间加入使用手册

在算法频繁启停后，画面中目标检测框显示正常
1. 算法运行日志自动清除机制 2. 无算法运行日志外的数据存储 3.算法日志中无敏感信息存留（密码，车牌等）
建议（具体算法具体分析）： 白天：检测率>=92%；误检率<=5%； 夜晚：检测率>=92%；误检率<=8%； 基线标准： 白天：检测率>=85%；误检率<=10%； 夜晚：检测率>=85%；误检率<=10%；
样本数量 >=20 说明： 参考场景影响因子如下： 1.天气：晴天 2.室内外：室内，室外 3.夜晚：红外 4.光线方向：顺光 5.分辨率：200w（1080P）、400w(2k)、800w(4K) 6.遮挡：无遮挡，轻微遮挡（10%左右），中度遮挡（30%左右） 7.目标密度：低密（<8），中密(8-30) 8.俯仰角度:0-15°，15-45° 9.目标属性：颜色，形态，尺寸等 10.非目标物体：颜色，形态，尺寸等 要求提供的视频素材能覆盖所有可能涉及到的场景影响因子
建议：每个场景（建议侧重场景覆盖度，即单场景尽量包含多个标准场景因子）样本数量>=20 所有设计场景因包含所有场景影响因子，总的样本数量>=100 说明：每一个不同的目标计算一个样本
参考值： 白天：检测率>=75%； 夜晚：检测率>=75%；

样本数 >=10 说明： 参考场景影响因子如下： 1.天气：雾天，雨天，雪天 2.光线方向：逆光 3.遮挡：高度遮挡（>30%） 4.目标密度：高密(30-100) 5.俯仰角度: >45° 要求提供的视频素材能覆盖所有可能涉及到的场景影响因子
建议：每个场景（建议侧重场景覆盖度，即单场景尽量包含多个标准场景因子）样本数量>=20 所有设计场景因包含所有场景影响因子，总的样本数量>=50 说明：每一个不同的目标计算一个样本

5.6 算法包签名

通过商城认证后，有商城负责给算法包签名

5.7 算法包上线算法商城

- ISV厂商与算法验收团队的分工说明

流程角色	厂商自验证阶段	测试验收阶段
ISV算法厂商	完成**生态算法全量测试验证（包含功能，部署，license，可靠性，指标及性能的测试验证）**，且输出测试用例结果。	完成验收问题的澄清与闭环。
算法验收团队	协助ISV算法厂商完成指标场景用例设计。	1. 完成生态算法功能，部署，license，可靠性的全量测试验收； 2. 完成**对ISV厂商指标，性能测试结果的抽样验收**。

- 算法验收团队完成验收工作耗时为3到5天，ISV厂商自验证工作由厂家进度决定，若前期材料准备不全或者算法验证发现问题，导致的延期情况视ISV厂商的整改进度而定，双方验证共同完成无误后可安排上线商城。
- 算法商城测试验收标准请参考**5.5 认证测试**。

6 算法功能开发

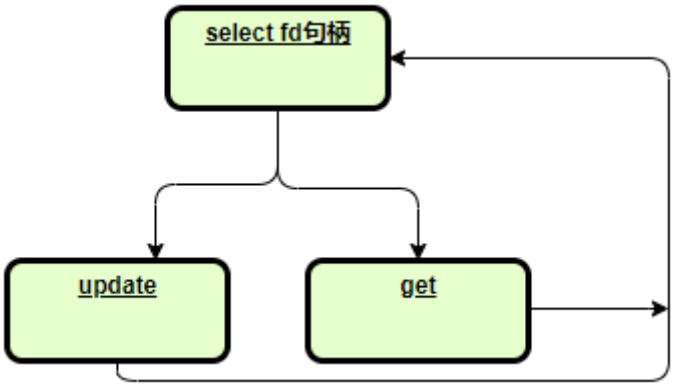
6.1 参数配置
6.2 任务管理
6.3 视频请流/解码
6.4 推理
6.5 分析结果回传

6.1 参数配置

参数配置的技术原理已经在之前的段落进行描述：**2.9 参数配置原理**

srvfs实现了一种特殊的文件系统，可以使用标准的POSIX打开，获取文件句柄。服务端/客户端通过向同一文件写入/读取数据来通信。

1. 当客户打开omu页面时，omu portal服务通过向文件config.paas.sdc发送get命令查询到默认用户配置。
2. 而三方算法程序通过select操作 config.paas.sdc文件句柄，当出现update事件时，将用户设定配置项更新到自己变量。当出现get事件时，将当前配置更新到omu portal服务



3. 伪代码实现如下

```
const char *SDC_CUSTOM_WEB_SERVER = "/mnt/srvfs/config.paas.sdc";
fd = open(SDC_CUSTOM_WEB_SERVER, O_RDWR);
```

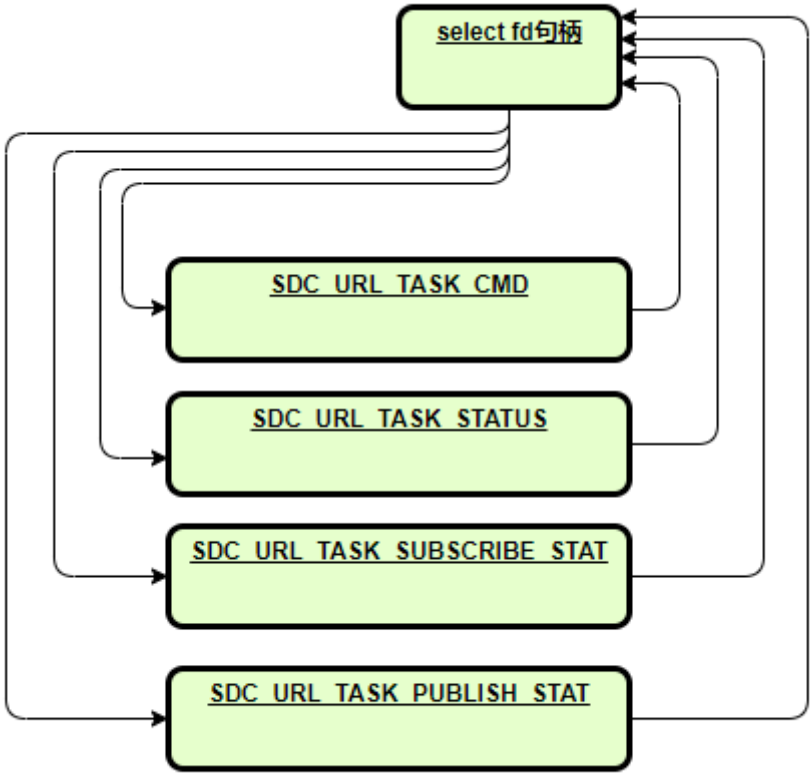
```
while (bRunFlag) {
    struct timeval stTv;
    stTv.tv_sec = 5;//设置5秒超时
    stTv.tv_usec = 0;
    FD_ZERO(&fdRfds);
    FD_SET((uint32_t)fd, &fdRfds);
    int ret = select((int32_t)(fd + 1), &fdRfds, nullptr, nullptr, &stTv);
    char buf[1024] = {0};
    struct sdc_common_head *head = (sdc_common_head *)(buf);//强转成结构体
    int nret = read(fd, buf, sizeof(buf));
    if (head->url != CONFIG_URI_CONFIG_PARAM) {//校验URL类型
        continue;
    }

    switch (head->method) { // 根据method，返回
        case HBTP_METHOD_UPDATE: {
            VA_CONFIG_INFO_ST stCofInfo = { 0 };
            int param_rep_offset = head->head_length;
            CONFIG_APP_PARAM_RESPONCE *pKeyValueArray = (CONFIG_APP_PARAM_RESPONCE *)
(&buf[param_rep_offset]);
            for (int32_t i = 0; i < pKeyValueArray->paramNum; i++) {
                pKeyValueArray->param[i].key //处理config的key值
                pKeyValueArray->param[i].value //处理config的value值
            }
            struct sdc_common_head ret_head = *head;
            //返回200响应码
            ret_head.code = SDC_CODE_200;
            ret_head.head_length = sizeof(sdc_common_head);
            ret_head.content_length = 0;
            nret = write(m_fd, &ret_head, sizeof(ret_head));
        } break;
        case HBTP_METHOD_GET: {
            CONFIG_PARAM stConfigParams[VA_CONF_MAX_NUM] = {0};
            CONFIG_APP_PARAM_RESPONCE keyValueArray = { 0 };
            keyValueArray.paramNum = VA_CONF_MAX_NUM;
            keyValueArray.param = stConfigParams;
            for (int32_t i = 0 ; i < m_stConfInfo.conf_num; i++) {
                keyValueArray.param[i].key = //赋值config的key值
                keyValueArray.param[i].value = //赋值config的value值
            }
            head->content_length = sizeof(CONFIG_APP_PARAM_RESPONCE);
            //返回200响应码，并吧config值放到body中
            head->code = SDC_CODE_200;
            head->response = 1;
            int content_offset = head->head_length;
            void *config_app_param_rep = (void *)&buf[content_offset];
            memcpy(config_app_param_rep, &keyValueArray, sizeof(CONFIG_APP_PARAM_RESPONCE));
            nret = write(fd, buf, head->head_length + head->content_length);
        } break;
        default:
            break;
    }
}
```

6.2 任务管理

任务管理相关介绍**2.10 任务管理原理**，任务管理也是通过srvfs创建的task.paas.sdc文件来串联任务的下发与上报流程。

1. 每当用户通过触发任务下发，fd句柄的select操作就会被触发，然后根据事件类型进行相应事件的处理



任务订阅伪代码：

```
#define FILE_SRVFS_TASK "/mnt/srvfs/task.paas.sdc"
fd = open(FILE_SRVFS_TASK, O_RDWR);
char buf[1024] = {0}; // 要保证足够大
struct sdc_common_head* head = (sdc_common_head*)(buf);
int nret = read(fd, buf, sizeof(buf));
if (head->code != SDC_CODE_200) {
    return 0;
}
switch (head->url) {
    case SDC_URL_TASK_CMD: { // 任务启停
        struct sdc_shm_cache shm_cache = { 0 };
        struct paas_shm_cached_event *shm_event;
        shm_event = (paas_shm_cached_event *)(&buf[head->head_length]);
        shm_cache.addr_phy = shm_event->addr_phy;
        shm_cache.size = shm_event->size;
        shm_cache.cookie = shm_event->cookie;
        nret = ioctl(cache_fd, SDC_CACHE_MMAP, &shm_cache);
        printf("ioctl(SDC_CACHE_MMAP) nret = %d\n", nret);
        if (nret) { // 读取CACHE数据
            break;
        }
        struct sdc_task_event *event = (sdc_task_event *)(shm_cache.addr_virt);
        Process(event); // 处理任务信息
        munmap((void *)shm_cache.addr_phy, shm_cache.size);
    } break;
    case SDC_URL_TASK_STATUS: {
    } break;
    case SDC_URL_TASK_SUBSCRIBE_STAT: {
    } break;
    case SDC_URL_TASK_PUBLISH_STAT: {
    } break;
    default: {
```



```
            printf("unknown response! url=%d\n", head->url);
        }
    }
```

2. 对于每个任务状态而言，需要通过向task.paas.sdc文件写入数据上报任务状态，可以通过定期上报过个任务的状态来提升smu的任务处理效率。

6.3 视频请流/解码

1. 视频拉流的接口描述在 **9.2 Restful接口**中有描述，首先通过restful接口进行登陆

```
    const std::string strLoginData = "{\"userName\": \"" + strUserName + "\",\"password\": \"" +
strPassword + "\"}";//用户名+密码
    const std::string strUrl = "https://" + strHost + ":18531";//IVS1800的ip地址
    const std::string strLoginUrl = strUrl + strLoginUrlPath;
    m_httpClient = make_shared<HttpClient>();
    bool bRet = m_httpClient->Init();
    m_httpClient->SetHeader(HEADER_CONTENT_TYPE, DEFAULT_CONTENT_TYPE);
    m_httpClient->SetHeader(HEADER_CACHE_CONTROL, DEFAULT_NO_CACHE);
    int code = m_httpClient->Post(strLoginUrl, strLoginData);
    LOGI("httpClient->Post() code=%d", code); // CURLE_OK
    if (code != CURLE_OK) {
        return false;
    }
    string responseBody = m_httpClient->GetResponseBody();
    Json::Value rspJson;
    int ret = m_httpClient->GetJsonResponseBody(rspJson);
    if (ret != CURLE_OK || rspJson["resultCode"].isNull() || rspJson["resultCode"] != 0) {
        return false;
    }
    //如果登陆成功，返回body的json字符串中的cookie作为登陆凭据
    const std::string strCookie = m_httpClient->GetResponseCookie();
    m_httpClient->SetHeader(HEADER_COOKIE, strCookie);
```

2. 登陆成功后，通过restful接口获取rtspUrl：

```
    const std::string strUrl = "https://" + m_strHost + ":18531";
    const std::string strGetRtspUrl = strUrl + strGetRtspUrlPath;
    const std::string strGetRtspUrlData = "{\"cameraCode\": \"" + strCameraCode +
    "\",\"mediaURLParam\": " + "{\"broadCastType\": 0,\"packProtocolType\": 1,\"protocolType\": 2," + "
\"serviceType\": 1,\"streamType\": 1,\"transMode\": 0}}";
    //将摄像机sn码拼到请求restful接口URL
    int code = m_httpClient->Post(strGetRtspUrl, strGetRtspUrlData);
    if (code != CURLE_OK) {
        return;
    }
    string responseBody = m_httpClient->GetResponseBody();
    Json::Value rspJson;
    int ret = m_httpClient->GetJsonResponseBody(rspJson);
    if (ret != CURLE_OK || rspJson["resultCode"].isNull() || rspJson["resultCode"] != 0) {
        iResultCode = rspJson["resultCode"].asInt();
        break;
    }
    //返回body的json字符串中包含rtspURL
    strRtspUrl = rspJson["rtspURL"].asString();
```

3. 获取rtspUrl之后通过MPSDK拉取h264\h265裸码流，基本代码逻辑如下

```
    MP_NET_PLAY_PARAM param = { 0 };
    param.bBlocked = MP_TRUE; // 同步方式
    strcpy(param.rtspUrl.cRtspURL, strRtspUrl.c_str());
    param.rtspUrl.uiProtocolType = 2; // 协议类型，1-UDP 2-TCP，默认为1
    param.rtspUrl.uiRtspType = 1;     // RTSP类型，1-RTSP 2-RTSP OVER TLS，默认为1
    param.rtspUrl.rtspOverNorm = 1;   // 是否使用标准rtsp流程:0-否；1-是
    IVS_INT32 iRet = MP_NET_StartRealPlayCBRaw(&param, RawRealCallback, this, &m_ulHandle);
```

4. Atlas芯片具有H264/H265硬件解码能力，matrix框架已经把解码流程固定，在graph.config文件中配置一定数量的解码engine

```
 //1000号解码engine
 engines {
   id: 1000
```



```
  engine_name: "DecodeEngine"
  side: DEVICE
  thread_num: 1
  so_name: "./libDevice.so"
  ai_config{
  }
}
//1001号解码engine
engines {
  id: 1001
  engine_name: "DecodeEngine"
  side: DEVICE
  thread_num: 1
  so_name: "./libDevice.so"
  ai_config{
  }
}
```

5. 在算法进程的host侧，创建hiai::Graph对象，向config文件中配置的解码engine id发送码流，如果发送失败，需要手动释放HIAI_DMalloc申请的内存，并且发送数据不宜过度频繁，根据自研算法的经验，希望每12帧发送一次。

```
//1000号解码engine
uint8_t *allocBuffer = nullptr;
ret = hiai::HIAIMemory::HIAI_DMalloc(uDataBufLen, (void *&)allocBuffer);
streamData->buf.data.reset(allocBuffer, [](uint8_t *data) {});

hiai::EnginePortID engineId;
engineId.graph_id = graphID;
engineId.port_id = 0;
engineId.engine_id = 1000; //与config文件中engine id对应
ret = graph->SendData(engineId, "StreamRawData", std::static_pointer_cast<void>(streamData));
if (ret != HIAI_OK) {
  hiai::HIAIMemory::HIAI_DFree((void *)allocBuffer);
}
```

6. 在算法的Device侧，实现HIAI_IMPL_ENGINE_PROCESS模板函数，处理解码回调函数，解码后的图片可用于算法推理

```
HIAI_IMPL_ENGINE_PROCESS("DecodeEngine", DecodeEngine, DECODE_INPUT_SIZE)
{
    HIAI_ENGINE_LOG(APP_INFO, "[DecodeEngine] start");
    std::shared_ptr<StreamRawData> streamRawData = std::static_pointer_cast<StreamRawData>(arg0);
    g_modelType = streamRawData->modelType;
    if (HIAI_OK != DecodeH26xVideo(streamRawData)) {
        return HIAI_ERROR;
    }

    HIAI_ENGINE_LOG(APP_INFO, "[DecodeEngine] end");
    return HIAI_OK;
}
```

6.4 推理

6.5 分析结果回传

- 推理结果从device向host侧回传

在graph.config配置文件中，定义完engine后使用connects方法将不同engine“连接”

```
connects {
  src_engine_id: 1000
  src_port_id: 1
  target_engine_id: 2002
  target_port_id: 0
}
```


在device侧实现代码中，将结果回传给host侧的Engine

```
//inputArgQueue为注册的回调函数
std::shared_ptr<DeviceStreamData> deviceStreamData = inputArgQueue[j];
//将入参转换为DeviceStreamData类型指针
if (deviceStreamData->detectResult.size() > 0) {
    // crop small image for follow up process
    // 调用matrix框架提供的send方法，即可将分析结果数据发送到host侧
    if (HIAI_OK != hiai::Engine::SendData(0, "DeviceStreamData",
                                          std::static_pointer_cast<void>(deviceStreamData))) {
        HIAI_ENGINE_LOG(APP_ERROR, "ObjectDetectionEngine senddata error!");
        return HIAI_ERROR;
    }
    HIAI_ENGINE_LOG(APP_INFO, "ObjectDetectionEngine senddata successfully!");
 }
```

- 将推理结果通过TLV发送给ivs1800系统

9.4.3 分析结果回传接口文档里面详细说明了分析结果，通过tlv形式回传的原理以及数据组织形式。

代码的具体实现路径在Demo\SrvfsEventSubAndPub\VaEvent.h

```
//VaEvent封装了tlv数据的发送行为
VaEvent vaEvent;
vaEvent.SetVideoUUID(m_refStTaskInfo.szCameraCode);//设置上报摄像机编码
vaEvent.SetMetaType(META_ALARM);//设置上报类型
const unsigned char* objData = static_cast<const unsigned char*>(cmd.u.va_result_data_cmd->data);
vaEvent.VcmPluginAnalyzeCallbackExV2(m_iAnalyzeChannel, cmd.u.va_result_data_cmd->object_id,
    objData, cmd.u.va_result_data_cmd->data_len, iImageNum,
    &(cmd.u.va_result_data_cmd->image_info));
```


7 License 开发指南

算法开发者如果准备将算法上线到商城销售，可能对算法的使用有License控制需求，可参考本章节开发License控制功能。

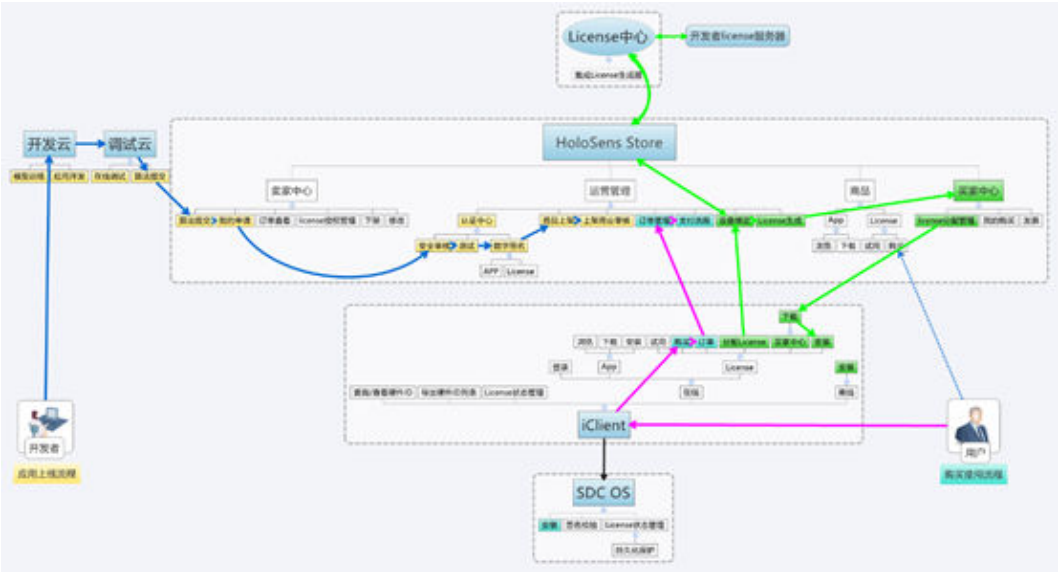
算法开发后自用场景或打包销售场景，可能不涉及License控制，可略过该章节。

7.1 License流程

7.2 硬件ID获取接口

7.3 License控制

7.1 License 流程



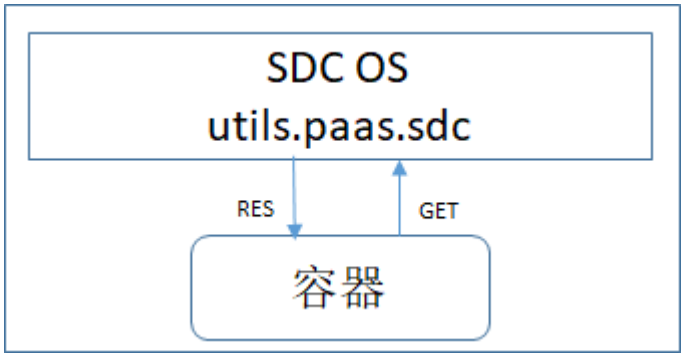
1. 算法厂家可以使用华为License中心产生的License方案，也可以自建License服务器；
2. 客户通过华为的iClient在线方式获取到License，并加载到指定的算法目录；
3. 算法可以到指定的目录读取License，并通过华为提供的接口获取到硬件ID；
   具体参考《 Holosens Store合作伙伴License原则方案 》

7.2 硬件 ID 获取接口

硬件ID是微边缘设备唯一序列号，客户使用ID去申请License。并放到算法指定的目录下。算法通过接口获取到ID，使用这个ID来校验License合法性。

该接口使用的utis.paas.sdc服务化接口，详见**9.4.4 获取硬件ID**。

- **获取流程**



- **请求Common Head方法资源**

方法	URL	VALUE
GET	SDC_URL_HARDWARE_ID	100

- **请求Content**

  无。
- **请求扩展头**

  无。
- **响应码**

  如果功能正常，响应码200， 其他为错误码。
- **响应Content**

  struct sdc_hardware_id

  {

  char id[33];

  };

  硬件标识最多为32个字符，目前微边缘设备的ID是20位。
- **响应扩展头**

  无。
- **参考样例**

```
#include "sdc.h"
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/uio.h>
```

```
int main(int argc,char* argv[])
{
    struct sdc_common_head head = {
        .version = SDC_VERSION,
        .url = SDC_URL_HARDWARE_ID,
        .method = SDC_METHOD_GET,
        .head_length = sizeof(head),
    };
    struct sdc_hardware_id  id;
    struct iovec iov[] = {
        { &head, sizeof(head) },
        { &id, sizeof(id) },
    };
    int nret,fd;
    fd = open("/mnt/srvfs/utils.iaas.sdc", O_RDWR);
    if(fd < 0) {
        printf("open file error\n");
        return -1;
    }
    nret = write(fd, &head, sizeof(head));
    if(nret < 0) {
        printf("write to file error\n");
        close(fd);
        return -1;
    }
    nret = readv(fd, iov, sizeof(iov) / sizeof(iov[0]));
    if(nret < 0 || head.code != SDC_CODE_200 || head.head_length != sizeof(head)) {
        printf("recv from error\n");
        close(fd);
        return -1;
    }
    printf("hardware_id: %s\n", id.id);
    close(fd);
    return 0;
}
```

7.3 License 控制

1. license持久化接口

IVS1800&ITS800提供了license持久化接口，防止临时license被经常加载试用。使用下面的接口进行访问。

持久化信息获取接口：

app 可以向 SDC OS 读取部分持久化 license 信息。服务化接口：appmgr.paas.sdc 通过该接口使得 APP 可以获取到 license 状态信息，方便 APP 进行 license 状态管理。请求 Common Head 方法：

方法 URL VALUE GET SDC_URL_LICENSE_STATE 0x05

请求 Content:

```
struct license_state_op_req{
    char app_name[128];
    char app_version[128];
};
```

app_name: APP 名称；

app_version：APP 对应版本号；

根据 APP 名称和版本信息获取对应的 License 状态信息。

请求扩展头无。

响应码： 如果功能正常，响应码 200，输入错误为 400，服务端错误为 500。

响应 Content：

响应 Content 为下面两个结构的组合，消息体 body 内存布局分别为 appmgr_err_rsp、 get_license_state_info，结构体定义如下：

```
typedef struct err_rsp {
    int err_code; // err_code 等于 0 时, err_str_len 长度为 0；反之，err_str_len 为错误信息字符串 err_str 长度
    int err_str_len; //err_str_len 为错误信息字符串 err_str 长度
    char err_str[0]; // 空终止字符串，该字段不占用内存空间，如果 err_code 不等于 0，则 sizeof(appmgr_err_rsp)偏移位置即为错误字符串，反之，则为 license 持久化信息结构
} appmgr_err_rsp;
typedef struct license_state_info {
    char app_name[128];
    char app_version[128];
    char vendor[64];
    char license_type[16]; //trial\commercial\unkown，分别为试用、商用、未知（未知代表未查到记录）
    int32_t remain_days; //license 试用剩余天数，试用 license 有效
    char license_start_using_time[64]; //license 启用时间
}get_license_state_info;
```

响应 Content 响应解析样例代码：

```
int len = 0;
appmgr_err_rsp* err_rsp = (appmgr_err_rsp*)rsp;
appmgr_app_license_get_rsp* license_rsp = 0;
if(err_rsp == NULL){
    LOG("aync read msg failed.");
    return;
}
if (err_rsp->err_code){
    LOG("err_code:%d.", err_rsp->err_code);
    return;
}
len = sizeof(appmgr_err_rsp);
license_rsp = (appmgr_app_license_get_rsp*)((char*)rsp + len);
```

2. license控制

文件存放到/home/3rdApp/license_lic文件夹下，第三方算法可以直接读取license文件进行license中的路数或者其他选项控制。

8 算法调测

8.1 命令部署
8.2 日志参考

8.1 命令部署

- 算法开发完毕后，可以直接copy到IVS1800设备上进行调试。但是在运行的时候，考虑到和现有的系统进行隔离，需要运行在容器中。
- 开发环境打包好的镜像文件放到IVS1800设备系统中，请放在**/opt/third_algorithm**目录下。
- 确保当前环境docker服务开启，参考**运行docker服务**。

⚠ 注意

如果直接在IVS1800/ITS800设备上进行容器镜像制作或进行算法调测，需要先挂载至少一块硬盘，并在portal上对硬盘进行初始化(ITS800自带2块硬盘，只需在portal进行初始化)。否则可能导致算法相关文件占用系统根目录，而导致系统本身的业务进程无法正常运行。

操作步骤

步骤1 导入镜像

export打包的镜像，执行命令：

docker import ivsapp-1.0.0-1.tar

save打包的镜像，执行命令：

docker load -i ivsapp-1.0.0-1.tar

查看镜像

docker images

```
Euler:/opt/dockertmp # docker images
REPOSITORY                                  TAG                 IMAGE ID            CREATED             SIZE
ivsapp                                      v1.0                6ba7f8b6ace8        2 minutes ago       402MB
ivsbase                                     v1.0.0              b5c2f55c7e32        6 days ago          282MB
```


若导出的镜像REPOSITORY和TAG参数为none，可通过如下命令修改：

docker tag 6ba7f8b6ace8 ivsapp:v1.0

步骤2 生成算法容器

docker create -it --device=/dev/davinci_manager --device=/dev/hisi_hdc --device=/dev/davinci0 --device=/dev/davinci1 --name=test --log-opt max-file=2 --log-opt max-size=30KB --memory 500MB --cpus=0.3 --ulimit nofile=65535:65535 --net=host --user HwHiAiUser --privileged=false --cpuset-cpus="2,3" --pids-limit 100 --security-opt apparmor=docker-default,no-new-privileges,label=level:TopSecret --cap-drop NET_RAW -v /opt/third_algorithm_D/3rdApp/ivsapp/data:/home/3rdApp/data -v /opt/third_algorithm/run/ivsapp/res:/home/3rdApp/res -v /opt/third_algorithm/run/ivsapp/conf:/home/3rdApp/conf -v /opt/third_algorithm/run/ivsapp/license_lic:/home/3rdApp/license_lic -v /var/dlog:/var/dlog -v /usr/slog:/usr/slog -v /mnt/srvfs:/mnt/srvfs -v /tmp:/tmp ivsapp:v1.0 /bin/bash -c '/home/init_container.sh'

步骤3 查看容器

命令：***docker ps –a***

查看STATUS栏，如果显示的是Up表示运行状态，如果Created表示创建状态（未运行），Exited表示运行结束

```
Euler:/opt/dockertmp # docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
ba8bb08ba4b8        ivsapp:v1.0         "/bin/bash"         10 seconds ago      Created                                 jolly_gates
```

步骤4 启动容器

启动命令：**docker start *-i CONTAINER ID***（通过ps查看的ID）

```
bash-4.4$ ls -al
total 16
drwxr-xr-x 1 root root 4096 Apr 29 10:09 .
drwxr-xr-x 1 root root 4096 Apr 29 12:39 ..
drwxr-xr-x 6 1001 1002 4096 Apr 29 10:09 3rdApp
-rw------- 1 1001 1002   40 Apr 29 03:28 init_container.sh
```

----结束

8.2 日志参考

日志在容器 /home/3rdApp/out/log/ivs_app/run 路径下查看

日志中有关业务的关键信息有：

- 表示任务下发的日志：

  ```
  <14>1 2021-08-25T09:32:45.621+08:00 Euler IVS - - - |...Info|PID:0x8|TID:0xFFFF62FFD150|[VaCtrlCmdData.cpp: 54][va_task_excute_cmd_New]|va_task_excute_cmd_New taskId: 1629771717
  ```

  其中包括任务id等消息。

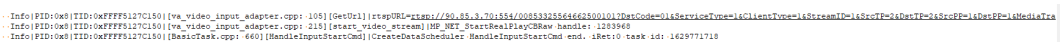
- 任务下发还有带有一些任务属性的json字段，也会进行打印：

  ```
  |TID:0xFFFF62FFD150|[SrvfsSub.cpp: 185][ProcessTaskCmdInfo]|ProcessTaskCmdInfo taskInfo: {"camera":{"cameraCode":"00853325633371601O1","connectCode":"","encodeType":1,"frameInterval":80,"frameRate":25,"height":1080,"ipv4":"90.85
  ```

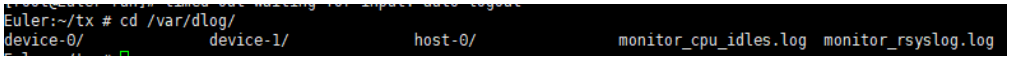
- 表示任务停止的日志：

  ```
  <14>1 2021-08-25T09:33:04.678+08:00 Euler IVS - - - |...Info|PID:0x8|TID:0xFFFF62FFD150|[VaCtrlCmdData.cpp: 126][va_task_stop_cmd_New]|va_task_stop_cmd_New taskId: 1629771716
  ```


- 如果拉视频流成功日志：

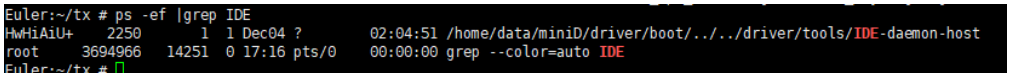


- 由于device侧日志的特殊性，需要在device侧代码调用HIAI_ENGINE_LOG将日志输出到/var/dlog/ 目录下：



  如果运行在0号D卡上，则把日志打印到device-0目录中，如果运行在1号D卡上，则把日志打印打device-1目录中。

- 如/var/dlog/路径下并没有device-"x"格式文件夹，或device-"x"格式文件夹下并没有生成日志文件，则在1800后台用root用户执行指令ps -ef | grep IDE，判断IDE-daemon-host进程是否存在。



  若不存在， D侧日志无法正常录入，需要将该进程拉起来。执行如下指令拉起IDEdaemon-host进程：

  systemctl restart start-minid.service

9 接口参考

9.1 DDK接口开发指南
9.2 Restful接口
9.3 eSDK接口(不再推荐)
9.4 服务化接口

9.1 DDK 接口开发指南

IVS1800、ITS800产品在硬件架构上和Atlas 300一样，在算法开发方面两边资源可以共享，DDK接口开发请参考Atlas 300相关文档。

⚠ 注意

- IVS1800和ITS800设备最大都支持2个昇腾310芯片，按照顺序，分别是0和1。
- 算法厂家默认使用0，也可以同时使用0和1，具体使用哪个芯片，算法必须可配置，防止和其他算法冲突。建议在graph.conf中配置，并在算法说明中加以说明。

9.1.1 软件开发指导书

《Atlas 300 AI加速卡 1.0.0 应用软件开发指南.pdf》**https://support.huawei.com/enterprise/zh/doc/EDOC1100109701/**

9.1.2 模型转换指导书

《Atlas 300 AI加速卡 模型转换工具使用指导（型号+3000,+3010）.pdf》。**https://support.huawei.com/enterprise/zh/doc/EDOC1100109701/fdc40cce**

9.1.3 DVPP+API 参考

《Atlas 300 AI加速卡 DVPP API参考（型号+3000,+3010）.pdf》**https://support.huawei.com/enterprise/zh/doc/EDOC1100109701/262bad7**

9.1.4 Matrix API 参考

《Atlas 300 AI加速卡 Matrix API参考（型号+3000,+3010）.pdf》**https://support.huawei.com/enterprise/zh/doc/EDOC1100109701/c8781278**

9.1.5 参考样例

获取样例程序

- 获取Sample示例程序。

  获取地址：**https://gitee.com/HuaweiAtlas/samples**。

  使用方法：使用个sample前，请一定阅读gitee各Sample README.md文件。编译使用build.sh A300。

- eSDK和DDK融合例子

```
/*
 * 拉取一个摄像机的媒体流，入参是摄像机的编码
 */
int startRealPlay(IVS_CHAR* strCameraCode)
{
    IVS_REALPLAY_PARAM RealplayPara = { 0 };
    RealplayPara.bDirectFirst = 0;        //default
    RealplayPara.bMultiCast = 0;          //default
    RealplayPara.uiProtocolType = 2;   // 1-UDP 2-TCP      // 请使用TCP方式
    RealplayPara.uiStreamType = STREAM_TYPE_MAIN;    // 播放主流

    // 设置回调函数，在eSDK程序中收到媒体后，调用该回调函数处理帧数据
    printf("\r\nstartRealPlay, cameraCode is %s.\n", strCameraCode);
    IVS_INT32 iRet = IVS_SDK_StartRealPlayCBRaw(iSessionID, &RealplayPara, strCameraCode, \
                                        RealPlayCallBackRawFunc, (IVS_VOID *)"test", &ulRealPlayHandle);
    if (IVS_SUCCEED == iRet) {
        printf("IVS_SDK_StartRealPlayCBRaw succeed!\r\n");
    }
    else {
        printf("IVS_SDK_StartRealPlayCBRaw failed!\r\n");
    }
    return iRet;
}

/* 摄像机播放的回调函数 */
void __SDK_CALL RealPlayCallBackRawFunc(IVS_ULONG ulHandle, IVS_RAW_FRAME_INFO*
pRawFrameInfo, IVS_VOID* pBuf, IVS_UINT32 uiBufSize, IVS_VOID* pUserData)
{
    printf("RealPlayCallBackRawFunc start.\n");
    // 码流回调处理, 应用需要在这里对码流进行处理
    static int frameIndex = 0;
    /*
    pRawFrameInfo->uiFrameType
    1：p帧
    2：i帧
    pRawFrameInfo->uiStreamType
    PAY_LOAD_TYPE_H264:99
    */
    printf("pRawFrameInfo->uiFrameType = %d, pRawFrameInfo->uiStreamType = %d.\n"
          , pRawFrameInfo->uiFrameType, pRawFrameInfo->uiStreamType);
    // 只有在I帧的情况下，才发送帧到
    if ((pRawFrameInfo->uiFrameType == 1 || pRawFrameInfo->uiFrameType == 5 || pRawFrameInfo-
>uiFrameType == 7 || pRawFrameInfo->uiFrameType == 8 )
        && pRawFrameInfo->uiStreamType == PAY_LOAD_TYPE_H264 ) {
        printf("*******frameIndex = %d, uiTimeStamp = %d, ullTimeTick = %lld, uiWaterMarkValue = %d
\r\n",
                frameIndex,
                pRawFrameInfo->uiTimeStamp,
                pRawFrameInfo->ullTimeTick,
                pRawFrameInfo->uiWaterMarkValue);
```



```
        fflush(stdout);

        /* 这里是将帧写入到一个文件中，用来调试定位 */
        if (g_pFileFrame != NULL) {
            fwrite(pBuf, uiBufSize, 1, g_pFileFrame);
            fflush(g_pFileFrame);
        }
        // 帧处理函数，最终通过DDK发送给D芯片，请参考DEMO代码
        TestFrame(ulHandle, pRawFrameInfo, pBuf, uiBufSize, pUserData);
    }
    ++frameIndex;

    printf("RealPlayCallBackRawFunc end.\n");
}
```

9.2 Restful 接口

Restful接口是系统北向提供给ISV的应用接口，基本包含了设备所有功能。ISV可基于该接口来开发自己的视频监控应用系统。

下面是第三方算法相关常用的接口(完整的接口及接口详情参考随产品发布的《接口参考(Restful)》文档)：

序号	类别	接口名称	方法及URL	接口功能
1	登录管理	用户登录	POST loginInfo/login/v1.0	登录管理。算法APP拉流前需要先登录系统，并进行保活。 算法运行结束需要登出系统，即注销。 **说明** 若不执行注销直接退出进程，系统会在30分钟后才检测到心跳失败，自动注销，因而建议在算法上的所有任务都结束时，立即进行注销操作。
2		用户保活	GET /common/keepAlive	
3		用户注销	GET /users/logout	
4	视频流	获取视频URL信息	POST /video/rtspurl/v1.0	算法APP可通过此接口拉视频流，然后发给D卡进行解码分析。
以下接口由应用系统(如iClient S100)实现第三方算法相关功能时使用：				
5	算法参数配置	查询支持参数配置的第三方算法列表	GET /thirdalgorithm/config/app	应用系统基于这些接口对第三方算法的参数进行配置。
6		查询第三方算法配置参数	GET /thirdalgorithm/config/appparam	
7		更新第三方算法配置参数	PUT /thirdalgorithm/config/appparam	

序号	类别	接口名称	方法及URL	接口功能
8		查询第三方算法配置国际化信息	GET /thirdalgorithm/config/applang	
9	任务管理	查询算法信息	POST /sdk_service/rest/management/loadAlgorithm	应用系统基于这些接口对第三方算法的分析任务进行管理。包括分析任务管理包括多分析类型的支持、ROI设置、灵敏度设置等。
10		创建智能分析任务	POST /sdk_service/rest/video-analysis/start-intelligent-analysis/v1.1	
11		批量创建智能分析任务	POST /sdk_service/rest/video-analysis/tasks	
12		删除智能分析任务	GET /sdk_service/rest/video-analysis/delete_intelligent_analysis?taskId={taskId}	
13		查询智能分析任务列表	POST /sdk_service/rest/video-analysis/get_intelligent_analysis_list/v1.1	
14		停止智能分析任务	GET /sdk_service/rest/video-analysis/stop_intelligent_by_taskid?taskid={taskId}	
15		重启智能分析任务	GET /sdk_service/rest/video-analysis/restart_intelligent_analysis?taskId={taskId}	
16	智能数据	添加/修改智能数据订阅	POST/PUT /sdk_service/rest/subscribes	应用系统可基于这三个接口订阅第三方算法产生的分析结果。包括第三方算法元数据和第三方算法告警数据。
17		智能元数据推送	POST, URL由订阅者传入	
18		智能告警数据推送	POST, URL由订阅者传入	
19		第三方算法动态库检索	POST /sdk_service/rest/videoanalysis/thirdalgorithm	应用系统可基于此接口查询第三方算法产生的元数据。

序号	类别	接口名称	方法及URL	接口功能
20		查询告警事件列表	POST /alarm/getalarmeventlist/v1.0	应用系统可使用此接口查询到第三方算法分析产生的历史告警列表。
21		获取告警事件信息	GET /alarm/alarmeventinfo/v1.0	应用系统可使用此接口查询到第三方算法分析产生的某一历史告警详细信息。

9.3 eSDK 接口(不再推荐)

须知

eSDK接口用于从系统请求摄像机的媒体流，已经不再推荐使用，推荐使用Restful接口，参考**9.2 Restful接口**。

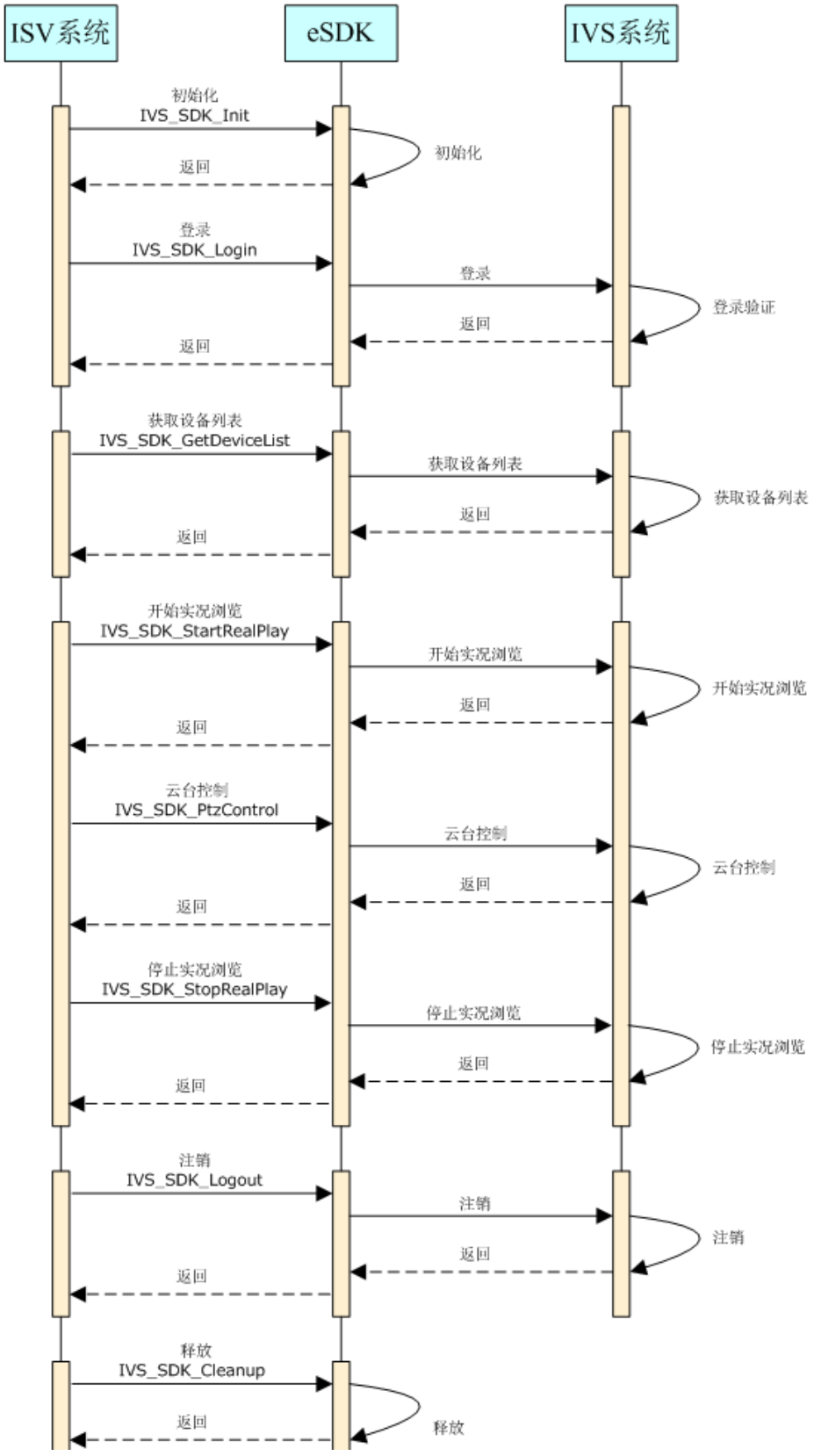
- eSDK可以实现根据摄像机的ID来获取摄像机视频或图片信息，用于给算法提供输入信息。主要接口如**9.3.1 eSDK接口使用说明**所示，用于实现初始化、登录、获取摄像机列表、取实况视频流、登出功能。具体参考《IVS1800 V100R019C50 北向接口参考(eSDK)》。
- eSDK包需要从support上下载，详见**4.2软件**要求介绍。

9.3.1 eSDK 接口使用说明

表 9-1 接口名称

接口名称	接口功能	备注
IVS_SDK_Init	对整个SDK系统进行初始化，内存预分配等操作。	-
IVS_SDK_Login	用户登录，超时时间为20秒。	-
IVS_SDK_GetDeviceList	获取指定类型的子设备列表，包含摄像机设备、告警设备的列表。	-
IVS_SDK_StartRealPlayCBRaw	通过摄像机编码开启实时浏览，以回调方式输出拼帧后的裸码流。	-
IVS_SDK_Logout	用户注销。	-

9.3.2 eSDK 业务流程



9.3.3 eSDK Demo 介绍

eSDK Demo内容详见**第11章节附录**。

eSDK库文件相关内容，打包在**eSDK_IVS_API_VEulerArmEdge_EulerArmEdge(64bit).tar.gz**压缩文件中，目录中的文件内容说明如**表9-2**所示。

表 9-2 内容说明

目录名称	目录说明	备注
lib	eSDK提供的so文件，需要根据实际使用的情况，拷贝到容器中	-
inc	头文件	-
config	配置文件，一般不需要修改，但是需要放到容器中	-

操作步骤

步骤1 修改登录名和密码

在Demo的main0108.cpp文件中，需要配置eSDK的登录名和密码。

```
/*登录用户的账号,密码，需要首先在OMU上进行开户，并修改登录密码*/
std::string g_LoginName = "esdkTest";       //  用户名
std::string g_LoginPwd = "super123";        //   密码
```

步骤2 编译esdkdemo

1. 解压文件：

   在需要编译时，将**eSDK_IVS_API_VEulerArmEdge_EulerArmEdge(64bit).tar.gz**上传到编译服务器解压，然后将demo程序**esdkdemo.zip**放到和esdk的lib同级目录

2. 设置环境变量：

   配置eSDK相关库的环境变量，按照实际路径(修改如下红色路径)添加

   export LD_LIBRARY_PATH=/home/esdk/lib/:${LD_LIBRARY_PATH}

3. 执行make进行编译。

   如下图：

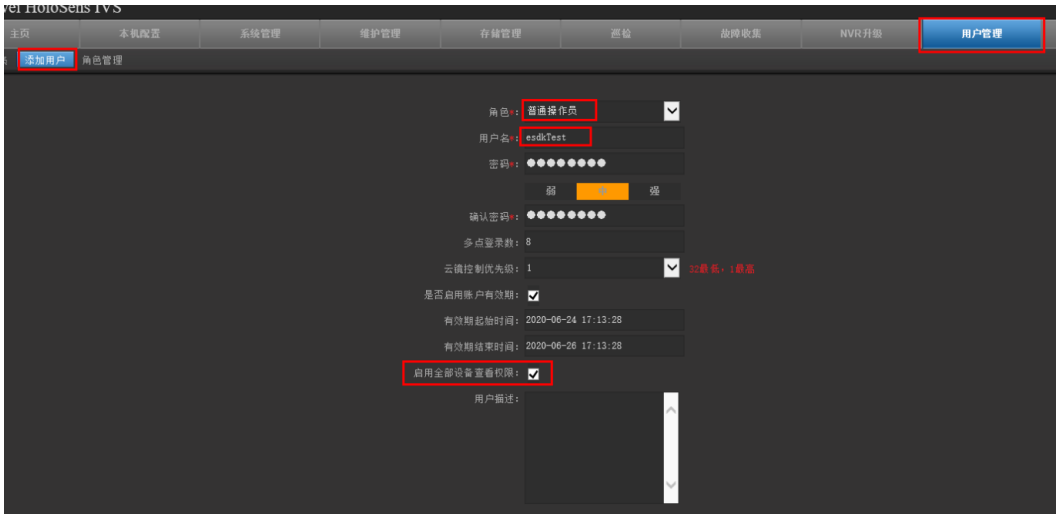
   编译完成后生成testsdk可执行文件

步骤3 微边缘上添加用户

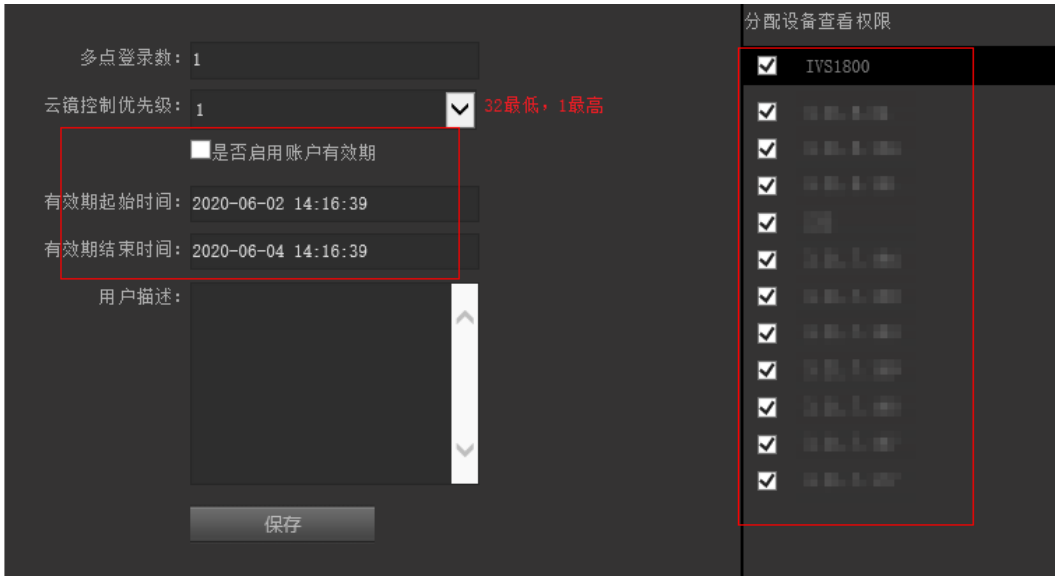
eSDK需要到SMU进程进行鉴权登录，所以需要在OMU Portal上添加用户。

1. 选择“用户管理 > 添加用户”。

添加用户时密码不要使用super123。



2. 重新使用IVS1800新增的用户登录，首次登录将密码修改为super123。同时设置esdkTest用户的账户有效期及对应设备的分配设备查看权限。



说明

- 分配设备查看权限，只需要将测试使用的相机选中即可。
- 初次创建用户后，需要使用esdktest用户登录下Portal，重新设置下密码，密码要和demo中保持一致。

步骤4 运行demo。

编译后的testsdk上传到微边缘设备上，例如放在/home/esdk目录，包括testsdk执行文件和lib库文件。

给testsdk添加可执行权限，执行命令：

chmod +x testsdk

根据需要设置LD_LIBRARY_PATH，如下：

export LD_LIBRARY_PATH=/home/esdk/lib:/home/lib:${LD_LIBRARY_PATH}

如下为Demo运行效果：

```
Euler:/home # ./testsdk
hello from testESdk!
Create Log Root Directory[mkdir -p /opt/log/testESdk///player/],ret=0
IVS_SDK_Init succeed
IVS_SDK_Login succeed,iSessionID = 0 .
Camera Info: CameraCode: 06053338460046640101#eb494ac603064473bff193043d497500, Name: Hwsdk60,Status: 1
Camera Info: CameraCode: 06053338460088910101#eb494ac603064473bff193043d497500, Name: Hwsdk09,Status: 1
Camera Info: CameraCode: 06053338460110150101#eb494ac603064473bff193043d497500, Name: Hwsdk28,Status: 1
Camera Info: CameraCode: 06053338460136680101#eb494ac603064473bff193043d497500, Name: Hwsdk47,Status: 1
Camera Info: CameraCode: 06053338460162820101#eb494ac603064473bff193043d497500, Name: Hwsdk26,Status: 1
Camera Info: CameraCode: 06053338460239550101#eb494ac603064473bff193043d497500, Name: Hwsdk08,Status: 1
Camera Info: CameraCode: 06053338460314740101#eb494ac603064473bff193043d497500, Name: Hwsdk07,Status: 1
Camera Info: CameraCode: 06053338460321590101#eb494ac603064473bff193043d497500, Name: Hwsdk19,Status: 1
Camera Info: CameraCode: 06053338460956070101#eb494ac603064473bff193043d497500, Name: Hwsdk52,Status: 1
Camera Info: CameraCode: 06053338461074240101#eb494ac603064473bff193043d497500, Name: Hwsdk12,Status: 1
Camera Info: CameraCode: 06053338461078600101#eb494ac603064473bff193043d497500, Name: Hwsdk58,Status: 1
Camera Info: CameraCode: 06053338461189590101#eb494ac603064473bff193043d497500, Name: Hwsdk64,Status: 1
Camera Info: CameraCode: 06053338461191780101#eb494ac603064473bff193043d497500, Name: Hwsdk22,Status: 1
Camera Info: CameraCode: 06053338461194500101#eb494ac603064473bff193043d497500, Name: Hwsdk44,Status: 1
Camera Info: CameraCode: 06053338461493200101#eb494ac603064473bff193043d497500, Name: Hwsdk53,Status: 1
Camera Info: CameraCode: 06053338461553880101#eb494ac603064473bff193043d497500, Name: Hwsdk27,Status: 1
Camera Info: CameraCode: 06053338461723370101#eb494ac603064473bff193043d497500, Name: Hwsdk01,Status: 1
Camera Info: CameraCode: 06053338461976630101#eb494ac603064473bff193043d497500, Name: Hwsdk16,Status: 1
Camera Info: CameraCode: 06053338462064230101#eb494ac603064473bff193043d497500, Name: Hwsdk48,Status: 1
Camera Info: CameraCode: 06053338462158250101#eb494ac603064473bff193043d497500, Name: Hwsdk36,Status: 1
Camera Info: CameraCode: 06053338462435740101#eb494ac603064473bff193043d497500, Name: Hwsdk46,Status: 1
```

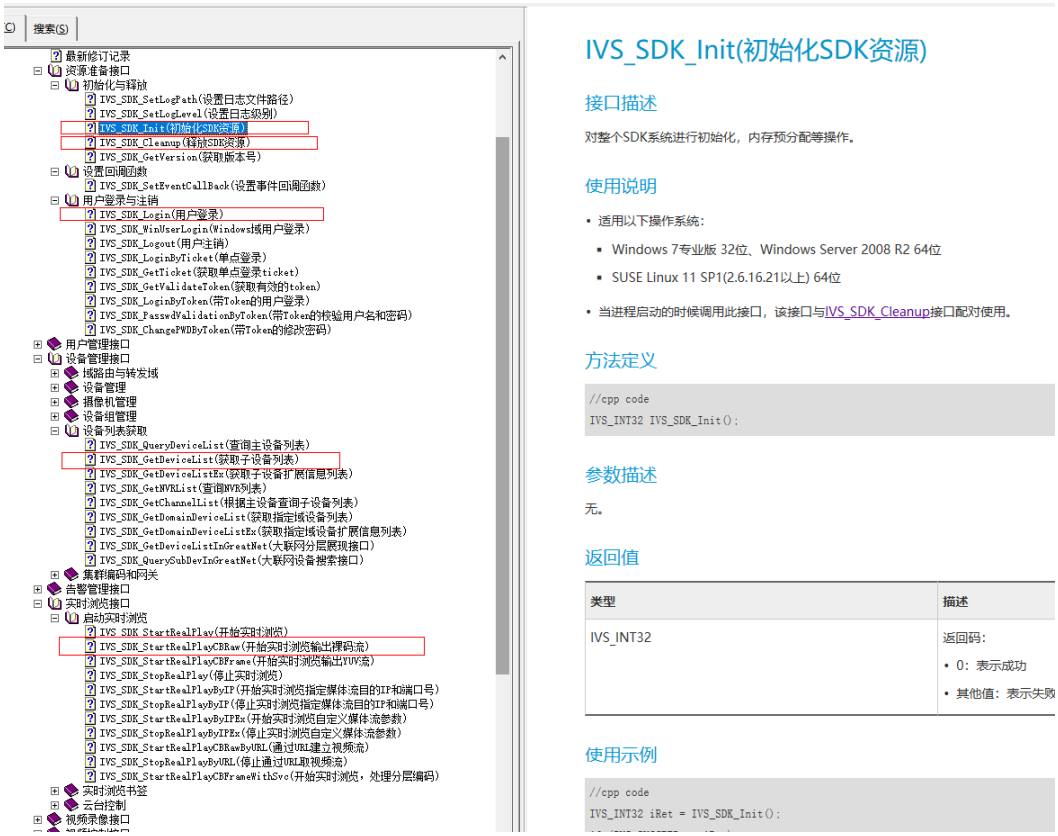
```
Got 64 cameras
begin to get one camera to play

startRealPlay, cameraCode is 06053338460046640101#eb494ac603064473bff193043d497500.
IVS_SDK_StartRealPlayCBRaw succeed!
uiStreamType is 99,uiFrameType is 7,uiGopSequence is 0
uiStreamType is 99,uiFrameType is 8,uiGopSequence is 0
uiStreamType is 99,uiFrameType is 5,uiGopSequence is 1
uiWidth is 3840,uiHeight is 2160
uiStreamType is 99,uiFrameType is 1,uiGopSequence is 1
uiWidth is 3840,uiHeight is 2160
uiStreamType is 99,uiFrameType is 1,uiGopSequence is 1
uiWidth is 3840,uiHeight is 2160
uiStreamType is 99,uiFrameType is 1,uiGopSequence is 1
uiWidth is 3840,uiHeight is 2160
uiStreamType is 99,uiFrameType is 1,uiGopSequence is 1
uiWidth is 3840,uiHeight is 2160
uiStreamType is 99,uiFrameType is 1,uiGopSequence is 1
uiWidth is 3840,uiHeight is 2160
uiStreamType is 99,uiFrameType is 1,uiGopSequence is 1
uiWidth is 3840,uiHeight is 2160
uiStreamType is 99,uiFrameType is 1,uiGopSequence is 1
uiWidth is 3840,uiHeight is 2160
uiStreamType is 99,uiFrameType is 1,uiGopSequence is 1
uiWidth is 3840,uiHeight is 2160
uiStreamType is 99,uiFrameType is 1,uiGopSequence is 1
uiWidth is 3840,uiHeight is 2160
uiStreamType is 99,uiFrameType is 1,uiGopSequence is 1
uiWidth is 3840,uiHeight is 2160
uiStreamType is 99,uiFrameType is 1,uiGopSequence is 1
uiWidth is 3840,uiHeight is 2160
```

```
uiWidth is 3840,uiHeight is 2160
uiStreamType is 99,uiFrameType is 1,uiGopSequence is 2
uiWidth is 3840,uiHeight is 2160
uiStreamType is 99,uiFrameType is 1,uiGopSequence is 2
uiWidth is 3840,uiHeight is 2160
uiStreamType is 99,uiFrameType is 1,uiGopSequence is 2
uiWidth is 3840,uiHeight is 2160
uiStreamType is 99,uiFrameType is 1,uiGopSequence is 2
uiWidth is 3840,uiHeight is 2160
uiStreamType is 99,uiFrameType is 1,uiGopSequence is 2
uiWidth is 3840,uiHeight is 2160
uiStreamType is 99,uiFrameType is 1,uiGopSequence is 2
uiWidth is 3840,uiHeight is 2160
uiStreamType is 99,uiFrameType is 1,uiGopSequence is 2
uiWidth is 3840,uiHeight is 2160
uiStreamType is 99,uiFrameType is 1,uiGopSequence is 2
uiWidth is 3840,uiHeight is 2160
uiStreamType is 99,uiFrameType is 1,uiGopSequence is 2
uiWidth is 3840,uiHeight is 2160
IVS_SDK_StopRealPlay succeed
Logout OK ,iSessionID = 0 .
iRet is 0 .
```

步骤5 其他说明

- 在访问设备内部时，对于IVS_SDK_Login接口，IP请使用127.0.0.1, 端口为9936(代码中已设置)。
- 具体内容，请参考接口文档和开发指南



----结束

9.4 服务化接口

9.4.1 参数配置接口

参考随产品发布的《 算法开发指南(基于HI3559A) 》中**config.paas.sdc**服务化接口定义。

9.4.2 任务管理接口

参考随产品发布的《 算法开发指南(基于HI3559A) 》中**task.paas.sdc**服务化接口定义。

9.4.3 分析结果回传接口

参考随产品发布的《 算法开发指南(基于HI3559A) 》中**event.paas.sdc**服务化接口定义。

9.4.4 获取硬件 ID

参考随产品发布的《 算法开发指南(基于HI3559A) 》中**utils.paas.sdc**服务化接口定义。

10 附录

10.1 网站链接
10.2 附件

10.1 网站链接

https://www.huaweicloud.com/?locale=zh-cn

https://support.huaweicloud.com/usermanual-ief/ief_01_0011.html

https://www.huaweicloud.com/ascend/home.html

10.2 附件

esdkdemo

graph.zip

用户管理附件.zip