

版本日志

版本号	日期	更新说明
1.0.22521010209.1	09/22/2020	1.支持linux arm rv1109系列芯片； 2.口罩检测版本；
1.0.22521010209.2	10/21/2020	1.更新算法库；
1.0.22521010209.3	10/29/2020	1.更新NPU算法库；
1.0.22521010209.4	11/02/2020	1.更新图像质量检测算法库；
1.0.22521010209.5	12/04/2020	1.规避不符合规则的唯一标识； 2.删除离线激活助手，请使用ASFGetActiveDeviceInfo接口采集设备信息保存在文件中上传到开发者中心生成离线授权文件，用于离线激活。

版本日志

1. 简介

1.1 产品概述

1.2 产品功能简介

1.2.1 人脸检测

1.2.2 人脸追踪

1.2.3 图像质量检测

1.2.4 人脸特征提取

1.2.5 人脸特征比对

1.2.6 人脸属性检测

1.2.7 活体检测

1.3 授权方式

1.3.1 在线授权

1.3.2 离线授权

1.4 环境要求

1.4.1 依赖库

1.4.2 兼容性

2. 接入须知

2.1 SDK的获取

2.2 SDK包结构

2.3 业务流程

2.3.1 通用流程概述

2.3.2 活体检测

2.3.2.1 基础原理

2.3.2.2 RGB 单目活体检测

2.3.2.3 IR 双目活体检测

2.3.2.4 RGB+IR 双目活体检测

2.3.2.5 场景及应用方案

2.3.3 人脸 1:1 比对

2.3.3.1 图片vs图片

2.3.3.2 实时视频流vs图片

2.3.4 人脸 1:N 搜索

2.3.5 方案选型

2.4 指标

2.4.1 算法性能

2.4.2 阈值推荐

2.4.3 图像质量要求

3. SDK的详细接入介绍

3.1 工程配置

3.2 支持的颜色空间颜色格式

3.3 数据结构

3.3.1 ASF_VERSION

3.3.2 ASF_ActiveFileInfo

3.3.3 ASF_SingleFaceInfo

3.3.4 ASF_MultiFaceInfo

3.3.5 ASF_FaceFeature

3.3.6 ASF_AgeInfo

3.3.7 ASF_GenderInfo

3.3.8 ASF_Face3DAngle

3.3.9 ASF_LivenessThreshold

3.3.10 ASF_LivenessInfo

3.3.11 ASF_FaceShelter

3.3.12 ASF_MaskInfo

3.3.13 ASF_FaceLandmark

3.3.14 ASF_LandMarkInfo

3.3.15 ASVLOFFSCREEN

3.4 枚举

- 3.4.1 检测模式
- 3.4.2 人脸检测方向
- 3.4.3 检测到的人脸角度
- 3.4.4 检测模型
- 3.4.5 人脸比对可选的模型
- 3.4.6 人脸特征提取可选的模型

3.5 功能接口

- 3.5.1 ASFGetActiveFileInfo
- 3.5.2 ASFOnlineActivation
- 3.5.3 ASFGetActiveDeviceInfo
- 3.5.4 ASFOfflineActivation
- 3.5.5 ASFInitEngine
- 3.5.6 ASFDetectFaces
- 3.5.7 ASFDetectFacesEx
- 3.5.8 ASFImageQualityDetectEx
- 3.5.9 ASFFaceFeatureExtract
- 3.5.10 ASFFaceFeatureExtractEx
- 3.5.11 ASFFaceFeatureCompare
- 3.5.12 ASFSetLivenessParam
- 3.5.13 ASFProcess
- 3.5.14 ASFProcessEx
- 3.5.15 ASFGetAge
- 3.5.16 ASFGetGender
- 3.5.17 ASFGetFace3DAngle
- 3.5.18 ASFGetLivenessScore
- 3.5.19 ASFSetFaceShelterParam
- 3.5.20 ASFGetFaceShelter
- 3.5.21 ASFGetMask
- 3.5.22 ASFGetFaceLandMark
- 3.5.23 ASFProcess_IR
- 3.5.24 ASFProcessEx_IR
- 3.5.25 ASFGetLivenessScore_IR
- 3.5.26 ASFGetVersion
- 3.5.27 ASFUninitEngine

4. 常见问题

4.1 常用接入场景流程

- 4.1.1 常用比对流程
- 4.1.2 常用比对流程 + 活体

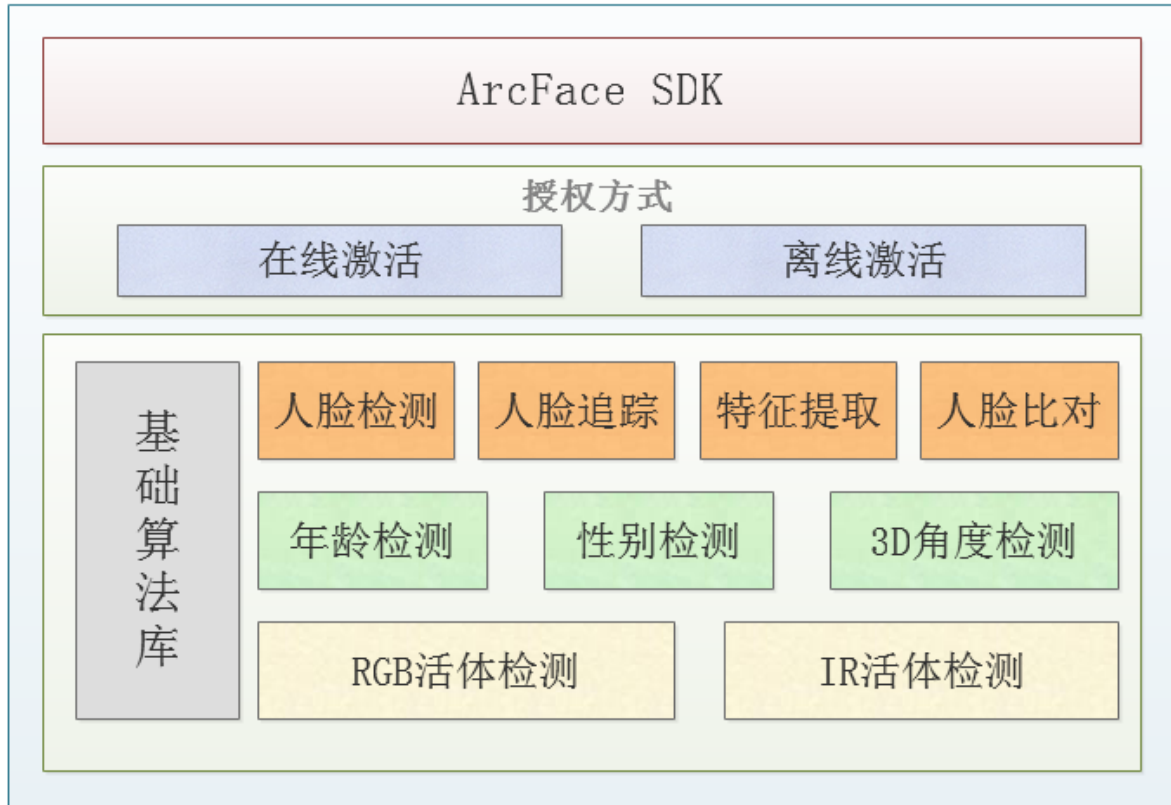
4.2 错误码列表

1. 简介

1.1 产品概述

ArcFace 离线SDK，包含人脸检测、图像质量检测、口罩检测、性别检测、年龄检测、人脸识别、RGB活体检测、IR活体检测等能力，初次使用时需联网激活，激活后即可在本地无网络环境下工作，可根据具体的业务需求结合人脸识别SDK灵活地进行应用层开发。

基础版本暂不支持离线激活，增值版本支持离线激活；



1.2 产品功能简介

1.2.1 人脸检测

对传入的图像数据进行人脸检测，返回人脸的边框以及朝向信息，可用于后续的人脸识别、特征提取、活体检测等操作；

- 支持IMAGE模式和VIDEO模式人脸检测。
- 支持单人脸、多人脸检测，最多支持检测人脸数为50。

1.2.2 人脸追踪

对来自于视频流中的图像数据，进行人脸检测，并对检测到的人脸进行持续跟踪。

1.2.3 图像质量检测

对图像数据中指定的人脸进行图像质量检测。

1.2.4 人脸特征提取

提取人脸特征信息，用于人脸的特征比对。

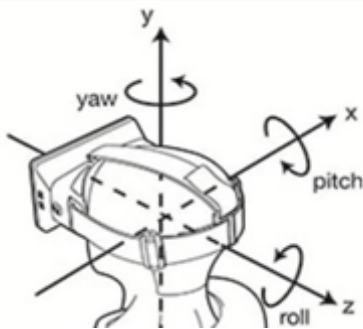
1.2.5 人脸特征比对

对两个人脸特征数据进行比对，返回比对相似度值。

1.2.6 人脸属性检测

人脸属性，支持检测年龄、性别以及3D角度、口罩、遮挡。

人脸3D角度：俯仰角（pitch），横滚角（roll），偏航角（yaw）。



1.2.7 活体检测

离线活体检测，静默式识别，在人脸识别过程中判断操作用户是否为真人，有效防御照片、视频、纸张等不同类型的作弊攻击，提高业务安全性，让人脸识别更安全、更快捷，体验更佳。支持单目RGB活体检测、双目（IR/RGB）活体检测，可满足各类人脸识别终端产品活体检测应用。

1.3 授权方式

SDK授权按设备进行授权，每台硬件设备需要一个独立的授权，此授权的校验是基于设备的唯一标识，被授权的设备在初次授权时需在线进行授权，授权成功后可以离线运行SDK。

1.3.1 在线授权

1. 确保可以正常访问公网；
2. 调用在线激活接口激活SDK；

注意事项：

1. 设备授权后，若设备授权信息被删除（重装系统/应用被卸载等），需联网重新激活；
2. 硬件信息发生变更，需要重新激活；

1.3.2 离线授权

1. 点击【查看激活码】->【离线激活】，进入离线激活操作界面；
2. 生成设备信息文件，使用 `ASFGetActiveDeviceInfo` 接口生成设备信息，保存到文件中；
3. 将设备信息文件上传到开发者中心，生成并获取离线授权文件；
4. 将离线授权文件拷贝到待授权设备上，调用离线激活接口激活SDK；

注意事项：

- 设备授权后，若设备授权信息被删除（重装系统/应用被卸载等），可用原有激活码进行重新激活；
- 激活码失效或硬件信息发生变更，需要使用新的激活码重新激活；

1.4 环境要求

1.4.1 依赖库

- 支持在linux arm32版本
- 仅支持rv1109 cpu指令集

1.4.2 兼容性

- 库依赖GLIBC 2.17及以上
- 库依赖GLIBCXX 3.4.19及以上
- 编译器GCC 4.8.2及以上

2. 接入须知

2.1 SDK的获取

注册开发者账号

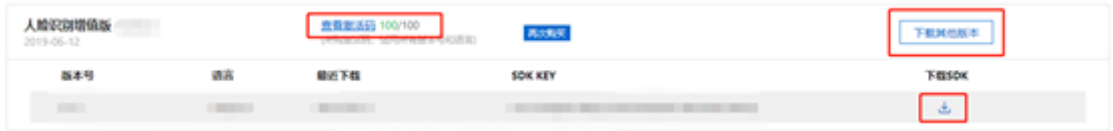
- 访问ArcSoft AI开放平台门户：<https://ai.arcsoft.com.cn>，注册开发者账号并登录。

SDK下载

- 创建对应的应用，并选择需要下载的SDK、对应平台以及版本，确认后即可下载SDK和查看激活码。



- 点击【查看激活码】即可查看所需要APPID、SDKKEY、ActiveKey（基础版本不需要ActiveKey），点击下载图标获取SDK开发包。



2.2 SDK包结构

---doc	
---ARCSOFT_ARC_FACE_DEVELOPER'S_GUIDE.PDF	开发说明文档
---inc	
---amcomdef.h	平台文件
---asvloffscreen.h	平台文件
---arcsoft_face_sdk.h	接口文件
---merror.h	错误码文件
---lib	

--- ---linux_x64	
--- ---libarcsoft_face.so	算法库
--- ---libarcsoft_face_engine.so	引擎库
---samplecode	
---samplecode.cpp	示例代码
---releasenotes.txt	说明文件

2.3 业务流程

2.3.1 通用流程概述

根据实际应用场景以及硬件配置，活体检测和特征提取可选择是否采用并行的方式。



- 如上图所示，人脸识别的核心业务流程可以分为以下四个步骤：

- 人脸检测：采集视频帧或静态图，传入算法进行检测，输出人脸数据，用于后续的检测。
- 活体检测：在人脸识别过程中判断操作用户是否为真人，有效防御照片、视频、纸张等不同类型的作弊攻击，提高业务安全性，可根据应用场景选择是否接入；
- 特征提取：对待比对的图像进行特征提取、人脸库的预提取，用于之后的比对；
- 人脸识别：1：1比对主要是判断「你是你」，用于核实身份的真实性；1：N搜索主要识别「你是谁」，用于明确身份的具体所属。

2.3.2 活体检测

提示：此版本仅支持RGB、IR活体。

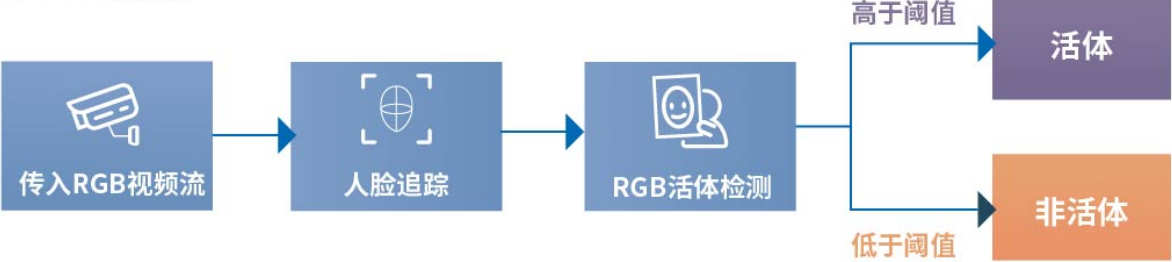
2.3.2.1 基础原理

- RGB可见光活体：主要基于图片破绽，判断目标对象是否为活体。例如图像中的屏幕反光、成像畸形、二次翻拍边框背景等。RGB活体检测基于单目摄像头，采用静默式识别方式，用户体验较好。同时RGB活体受光线以及成像质量影响较大，在强光、暗光等场景，容易影响检测结果，可通过适当的补光、使用宽动态镜头抵消逆光等方式缓解。
- IR红外活体：主要基于红外光线反射成像原理，通过人脸成像甄别是否为活体。基于红外成像特点，对于屏幕类攻击，基本可以达到近100%的活体防御。IR采用静默式识别方式，体验较好，但需要增加红外摄像头成本，同时要结合场景考虑红外成像距离。

提示：在实际业务场景中，需要根据场景特点，灵活组合使用以上几种活体方案。

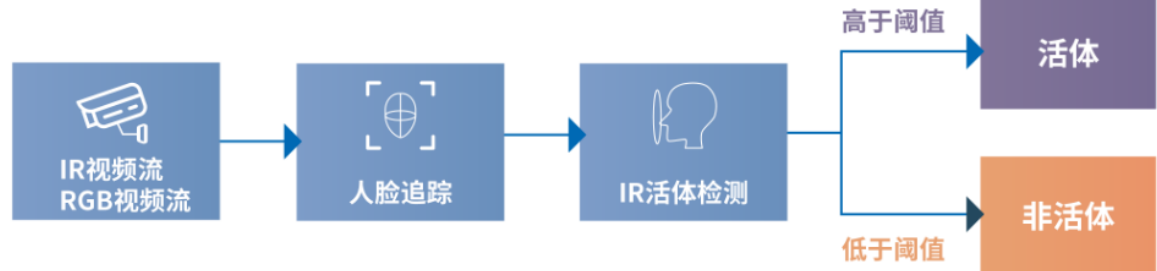
2.3.2.2 RGB 单目活体检测

RGB单目摄像头



2.3.2.3 IR 双目活体检测

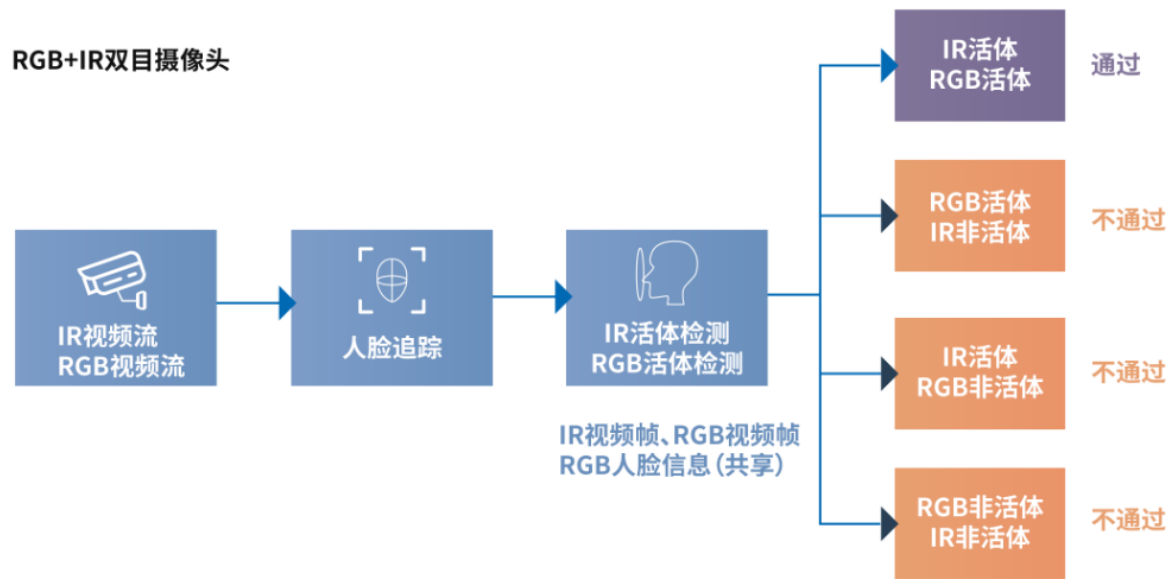
RGB+IR双目摄像头



提示：为了保证取到有效帧，使用RGB视频流检测到人脸信息和IR图像，传入IR活体检测接口进行IR活体检测。

2.3.2.4 RGB+IR 双目活体检测

RGB+IR双目摄像头



2.3.2.5 场景及应用方案

- 通行场景：此场景以考虑通行效率为主，并在此基础上增加活体检测。建议可采用RGB活体检测，体验较好，保障效率的同时仍可保证安全性。
- 身份核验场景：此场景优先确保业务安全性，需提升活体检测安全级别。可考虑采用 RGB+NIR双目活体检测，最大程度防御非法攻击。

使用时尽量避免强光、暗光等场景，可通过补光灯、宽动态摄像头进行缓解。

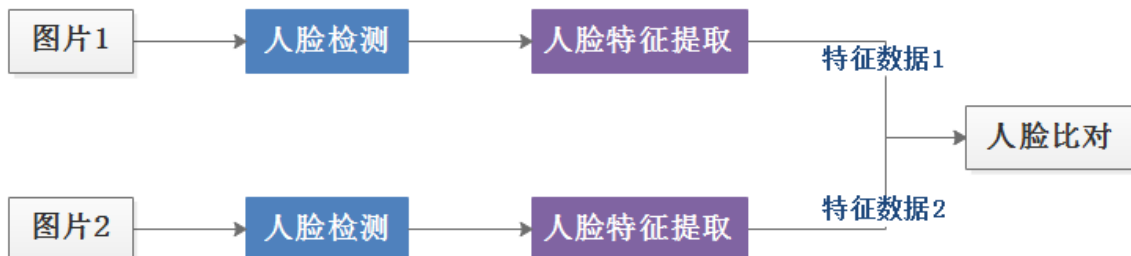
2.3.3 人脸 1:1 比对

1:1 比对常见的应用场景可分为如下两种形式：

2.3.3.1 图片vs图片

两张静态图片分别提取特征，进行比对。

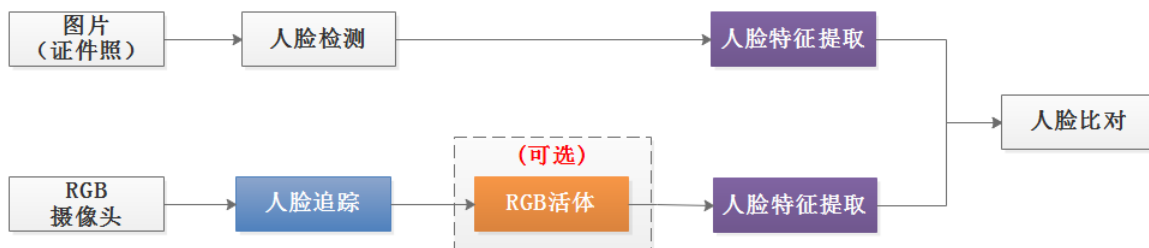
图片VS图片



2.3.3.2 实时视频流vs图片

该场景比较常见，典型的应用场景为人证比对和人脸门禁，本SDK同时支持人证比对和生活照识别，在特征比对时选择对应的特征比对模型即可。

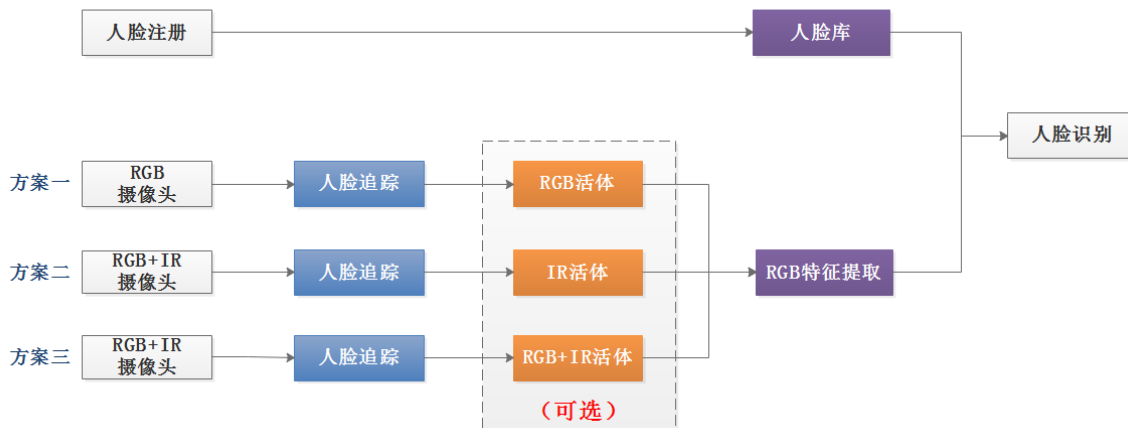
视频流VS图片



2.3.4 人脸 1:N 搜索

将需要识别的人脸图片集注册到本地人脸库中，当有用户需要识别身份时，从视频流中实时采集人脸图片，与人脸库中的人脸集合对比，得到搜索结果。

视频流VS人脸库



2.3.5 方案选型

可结合应用场景进行方案的选择

离线1：1比对

提供本地化的1：1人脸对比功能，证明你是你，可有效应用于车站、机场、大型活动、机关单位、银行、酒店、网吧等人员流动频繁场所或其它重点场景的人证核验，也可用于线上开户的人员身份验证等场景。

离线1：N检索

提供本地化的1：N人脸搜索功能，识别你是谁，可有效应用于社区、楼宇、工地、学校等较大规模的人脸考勤签到、人脸通行等应用场景。该版本人脸库在1万以内效果更佳。

2.4 指标

算法检测指标受硬件配置、图像数据质量、测试方法等因素影响，以下实验数据仅供参考，具体数据以实际应用场景测试结果为准。

2.4.1 算法性能

硬件信息：

- 处理器：ARMv7 Processor rev 5 (v7l)
- 运行内存(RAM)：1.0GB
- 系统类型：ARM Linux 32位

分辨率：1280 x 720

算法	性能(ms)
FaceDetect	< 25
FeatureExtract	< 22
FeatureCompare	< 0.0065
RGB Liveness	< 50
IR Liveness	< 20

2.4.2 阈值推荐

活体取值范围为[0~1]，推荐阈值如下，高于此阈值的即可判断为活体。

- RGB 活体：0.5
- IR 活体：0.7

人脸比对取值范围为[0~1]，推荐阈值如下，高于此阈值的即可判断为同一人。

- 用于生活照之间的特征比对，推荐阈值0.80
- 用于证件照或生活照与证件照之间的特征比对，推荐阈值0.82

2.4.3 图像质量要求

- 建议待检测的图像人脸角度上、下、左、右转向小于30度；
- 图片中人脸尺寸不小于50 x 50像素；
- 图片大小小于5MB；
- 图像清晰；

3. SDK的详细接入介绍

3.1 工程配置

Demo运行环境：

- Linux_x64系统；
- GLIBC 2.17及以上；
- GLIBCXX 3.4.19及以上；
- GCC 4.8.2及以上；
- cmake 3.0及以上；

执行过程：

1. 将ASFTestDemo工程拷贝到Linux系统下；
2. 需要将SDK包目录下中“lib”文件夹中的.so文件拷贝到/ASFTestDemo/linux_so文件目录下；
3. 建议将SDK包目录下中“inc”文件夹中的.h文件替换掉/ASFTestDemo/inc下的文件；
4. 下载SDK时，将从官网中获取的APPID/SDKKEY/ACTIVEKEY更新到samplecode.cpp文件中；
5. 在ASFTestDemo目录下新建一个build文件夹；（在build目录下编译）
6. 进入到/ASFTestDemo/build文件目录下,执行“cmake ..”命令，找到上一级的CMakeLists.txt文件编译，makefile文件会生成在build目录下；
7. 在/ASFTestDemo/build路径下执行“make”命令，生成可执行文件；
8. 在/ASFTestDemo/build路径下执行“./arcsoft_face_engine_test”命令,运行程序（./images文件夹下提供了三张用于测试的图片）；

注意事项：

1. 该demo提供图片为 NV21 格式的裸数据，图片保存在/ASFTestDemo/build/images路径下；
2. 图片宽度需要满足4的倍数，YUYV/I420/NV21/NV12/GRAY格式的图片高度为2的倍数，BGR24格式的图片高度不限制；

3.2 支持的颜色空间颜色格式

提示：当前版本仅支持前六种颜色格式，ASVL_PAF_DEPTH_U16 只是预留。

常量名	常数值	颜色格式说明
ASVL_PAF_NV21	2050	8-bit Y 通道，8-bit 2x2 采样 V 与 U 分量交织通道
ASVL_PAF_NV12	2049	8-bit Y 通道，8-bit 2x2 采样 U 与 V 分量交织通道
ASVL_PAF_RGB24_B8G8R8	513	RGB 分量交织，按 B, G, R, B 字节序排布
ASVL_PAF_I420	1537	8-bit Y 通道，8-bit 2x2 采样 U 通道，8-bit 2x2 采样 V 通道
ASVL_PAF_YUYV	1289	YUV 分量交织，V 与 U 分量 2x1 采样，按 Y0, U0, Y1, V0 字节序排布
ASVL_PAF_GRAY	1793	8-bit IR图像
ASVL_PAF_DEPTH_U16	3074	16-bit IR图像

3.3 数据结构

3.3.1 ASF_VERSION

结构体描述

SDK版本信息。

成员描述

```
typedef char* MPChar;
```

类型	变量名	描述
MPChar	Version	版本号
MPChar	BuildDate	构建日期
MPChar	CopyRight	版权说明

3.3.2 ASF_ActiveFileInfo

结构体描述

激活文件信息。

成员描述

```
typedef char* MPChar;
```

类型	变量名	描述
MPChar	startTime	SDK开始时间
MPChar	endTime	SDK截止时间
MPChar	activeKey	激活码
MPChar	platform	平台版本
MPChar	sdkType	SDK类型
MPChar	appId	APPID
MPChar	sdkKey	SDKKEY
MPChar	sdkVersion	SDK版本号
MPChar	fileVersion	激活文件版本号

3.3.3 ASF_SingleFaceInfo

结构体描述

单人脸信息。

成员描述

```
typedef signed int MInt32;
```

```
typedef struct __tag_rect
{
    MInt32 left;
    MInt32 top;
    MInt32 right;
    MInt32 bottom;
} MRECT, *PMRECT;
```

类型	变量名	描述
MRECT	faceRect	人脸框
MInt32	faceOrient	人脸角度

3.3.4 ASF_MultiFaceInfo

结构体描述

多人脸信息。

成员描述

```
typedef signed int MInt32;
```

```
typedef struct __tag_rect
{
    MInt32 left;
    MInt32 top;
    MInt32 right;
    MInt32 bottom;
} MRECT, *PMRECT;
```

类型	变量名	描述
MRECT*	faceRect	人脸框数组
MInt32*	faceOrient	人脸角度数组
MInt32	faceNum	检测到的人脸数
MInt32*	faceID	一张人脸从进入画面直到离开画面，faceID不变。 在VIDEO模式下有效，IMAGE模式下为空
MFloat*	wearGlasses	戴眼镜的置信度[0-1],推荐阈值：0.5
MInt32*	leftEyeClosed	左眼状态 -1:默认输出 0:未闭眼 1:闭眼
MInt32*	rightEyeClosed	右眼状态 -1:默认输出 0:未闭眼 1:闭眼

3.3.5 ASF_FaceFeature

结构体描述

人脸特征。

成员描述

```
typedef signed int MInt32;
```

```
typedef unsigned char MByte;
```

类型	变量名	描述
MByte*	feature	人脸特征
MInt32	featureSize	人脸特征长度

3.3.6 ASF_AgeInfo

结构体描述

年龄信息。

成员描述

```
typedef signed int MInt32;
```

类型	变量名	描述
MInt32*	ageArray	0:未知; >0:年龄
MInt32	num	检测的人脸数

3.3.7 ASF_GenderInfo

结构体描述

性别信息。

成员描述

```
typedef signed int MInt32;
```

类型	变量名	描述
MInt32*	genderArray	0:男性; 1:女性; -1:未知
MInt32	num	检测的人脸数

3.3.8 ASF_Face3DAngle

结构体描述

3D角度信息。

成员描述

```
typedef signed int MInt32;
```

```
typedef float MFloat;
```

类型	变量名	描述
MFloat*	roll	横滚角
MFloat*	yaw	偏航角
MFloat*	pitch	俯仰角
MInt32*	status	0:正常; 非0:异常
MInt32	num	检测的人脸个数

3.3.9 ASF_LivenessThreshold

结构体描述

活体置信度。

成员描述

```
typedef float MFloat;
```

类型	变量名	描述
MFloat	thresholdmodel_BGR	BGR活体检测阈值设置，默认值0.5
MFloat	thresholdmodel_IR	IR活体检测阈值设置，默认值0.7

3.3.10 ASF_LivenessInfo

结构体描述

活体信息。

成员描述

```
typedef signed int MInt32;
```

类型	变量名	描述
MInt32*	isLive	0:非真人；1:真人； -1：不确定；-2:传入人脸数 > 1；-3: 人脸过小；-4: 角度过大；-5: 人脸超出边界；-6: 深度图错误；-7: 红外图太亮了；
MInt32	num	检测的人脸个数

3.3.11 ASF_FaceShelter

结构体描述

遮挡信息。

成员描述

```
typedef signed int MInt32;
```


类型	变量名	描述
MInt32*	FaceShelter	"1" 表示 遮挡, "0" 表示 未遮挡, "-1" 表示不确定
MInt32	num	检测的人脸个数

3.3.12 ASF_MaskInfo

结构体描述

口罩信息。

成员描述

```
typedef signed int MInt32;
```

类型	变量名	描述
MInt32*	maskArray	"0" 代表没有带口罩, "1"代表带口罩, "-1"表不确定
MInt32	num	检测的人脸个数

3.3.13 ASF_FaceLandmark

结构体描述

特征点信息。

成员描述

```
typedef float Float;
```

类型	变量名	描述
MFloat	x	水平坐标
MFloat	y	垂直坐标

3.3.14 ASF_LandMarkInfo

结构体描述

额头区域点位。

成员描述

```
typedef signed int MInt32;
```

类型	变量名	描述
ASF_FaceLandmark*	point	额头区域点位
MInt32	num	人脸数量

3.3.15 ASVLOFFSCREEN

结构体描述

图像数据信息，该结构体在 asvloffscreen.h 基础的头文件中。

成员描述

```
typedef LPASVLOFFSCREEN LPASF_ImageData;

typedef unsigned int MUInt32;

typedef signed int MInt32;

typedef unsigned char MUInt8;
```

类型	变量名	描述
MUInt32	u32PixelFormat	颜色格式
MInt32	i32Width	图像宽度
MInt32	i32Height	图像高度
MUInt8**	ppu8Plane	图像数据
MInt32*	pi32Pitch	图像步长

3.4 枚举

3.4.1 检测模式

```
enum ASF_DetectMode{
    ASF_DETECT_MODE_VIDEO = 0x00000000,    //VIDEO模式，一般用于多帧连续检测
    ASF_DETECT_MODE_IMAGE = 0xFFFFFFFF     //IMAGE模式，一般用于静态图的单次检测
};
```

3.4.2 人脸检测方向

根据应用场景，推荐选择单一角度，检测效果更优，角度按逆时针方向。

```
enum ArcSoftFace_OrientPriority {
    ASF_OP_0_ONLY = 0x1,    // 常规预览下正方向
    ASF_OP_90_ONLY = 0x2,   // 基于0°逆时针旋转90°的方向
    ASF_OP_270_ONLY = 0x3,  // 基于0°逆时针旋转270°的方向
    ASF_OP_180_ONLY = 0x4,  // 基于0°旋转180°的方向（逆时针、顺时针效果一样）
    ASF_OP_ALL_OUT = 0x5    // 全角度
};
```

3.4.3 检测到的人脸角度

角度按逆时针方向。

```
enum ArcSoftFace_OrientCode {
    ASF_OC_0 = 0x1,    // 0度
    ASF_OC_90 = 0x2,   // 90度
};
```

```

ASF_OC_270 = 0x3,    // 270度
ASF_OC_180 = 0x4,    // 180度
ASF_OC_30  = 0x5,    // 30度
ASF_OC_60  = 0x6,    // 60度
ASF_OC_120 = 0x7,    // 120度
ASF_OC_150 = 0x8,    // 150度
ASF_OC_210 = 0x9,    // 210度
ASF_OC_240 = 0xa,    // 240度
ASF_OC_300 = 0xb,    // 300度
ASF_OC_330 = 0xc     // 330度
};

```

3.4.4 检测模型

根据图像颜色空间选择对应的算法模型进行检测。

```

enum ASF_DetectModel{
    ASF_DETECT_MODEL_RGB = 0x1    //RGB图像检测模型
    //预留扩展其他检测模型
};

```

3.4.5 人脸比对可选的模型

根据应用场景选择对应的模型进行人脸特征比对。

```

enum ASF_CompareModel{
    ASF_LIFE_PHOTO = 0x1,    //用于生活照之间的特征比对，推荐阈值0.80
    ASF_ID_PHOTO   = 0x2     //用于证件照或生活照与证件照之间的特征比对，推荐阈值0.82
};

```

3.4.6 人脸特征提取可选的模型

根据应用场景选择对应的模型进行人脸特征提取。

```

enum ASF_RegisterOrNot{
    ASF_RECOGNITION = 0x0,    //用于识别照人脸特征提取
    ASF_REGISTER    = 0x1     //用于注册照人脸特征提取
};

```

3.5 功能接口

当前版本使用同一个引擎句柄不支持多线程调用同一个算法接口，若需要对同一个接口进行多线程调用需要启动多个引擎。

例如：若需要做多人脸门禁系统，我们希望提升识别速度，一般会使用同一个引擎进行人脸检测，我们可以提前初始化多个引擎用于人脸特征提取，这些引擎只需要初始化 ASF_FACERECOGNITION 属性即可。同时要根据硬件配置来控制最多启动的线程数。

以下接口中示例代码依赖OpenCV实现

3.5.1 ASFGetActiveFileInfo

方法

```
MRESULT ASFGetActiveFileInfo(  
    LPASF_ActiveFileInfo activeFileInfo  
);
```

功能描述

获取激活文件信息。

参数说明

参数	类型	描述
activeFileInfo	out	激活文件信息

返回值

成功返回 MOK，失败详见 [4.2 错误码列表](#)。

示例代码

```
ASF_ActiveFileInfo activeFileInfo = { 0 };  
MRESULT res = ASFGetActiveFileInfo(&activeFileInfo);  
if (res != MOK)  
{  
    printf("ASFGetActiveFileInfo failed: %d\n", res);  
}  
else    //这里只打印了SDK有效期，其他信息打印方式类似  
{  
    printf("startTime: %s\n", activeFileInfo.startTime);  
    printf("endTime: %s\n", activeFileInfo.endTime);  
}
```

3.5.2 ASFOnlineActivation

方法

```
MRESULT ASFOnlineActivation(  
    MPChar AppId,  
    MPChar SDKKey,  
    MPChar ActiveKey  
);
```

功能描述

用于在线激活SDK。

初次使用SDK时需要对SDK先进行激活，激活成功后无需重复调用；
调用此接口时必须为联网状态，激活成功后即可离线使用；

参数说明

参数	类型	描述
AppId	in	官网获取的APPID
SDKKey	in	官网获取的SDKKEY
ActiveKey	in	官网获取的ACTIVEKEY

返回值

成功返回 MOK 或 MERR_ASF_ALREADY_ACTIVATED ，失败详见 [4.2 错误码列表](#)。

示例代码

```
//该组数据无效，仅做示例参考
#define APPID "D617np8jyKt1jN9gMr7ENbT3gjFdaeDet3Pp8yfWxnt"
#define SDKKEY "8FdAtZSffuvS6kuzPP4PrxyxkQMgpsi4yE3Amo61sbF2"
#define ACTIVEKEY "003N-1117-GGGG-FFFF"

MRESULT res = ASFOnlineActivation(APPID, SDKKEY, ACTIVEKEY);
if (MOK != res && MERR_ASF_ALREADY_ACTIVATED != res)
{
    printf("ASFOnlineActivation failed: %d\n", res);
}
```

3.5.3 ASFGetActiveDeviceInfo

方法

```
MRESULT ASFGetActiveDeviceInfo(
    MPChar* deviceInfo
);
```

功能描述

采集当前设备信息（可离线）。

参数说明

参数	类型	描述
deviceInfo	out	采集的设备信息，用于上传到后台生成离线授权文件

返回值

成功返回 MOK、MERR_ASF_ALREADY_ACTIVATED 以及 MERR_ASF_LOCAL_EXIST_USEFUL_ACTIVE_FILE ，失败详见 [4.2 错误码列表](#)。

示例代码

```
MRESULT res = MOK;
char* deviceInfo = NULL;
res = ASFGetActiveDeviceInfo(&deviceInfo);
if (res != MOK) {
    printf("ASFGetActiveDeviceInfo: %d\n", res);
}
```

3.5.4 ASFOfflineActivation

方法

```
MRESULT ASFOfflineActivation(  
    MPChar filePath  
);
```

功能描述

用于离线激活SDK。

注：离线授权方式详见 1.3。

参数说明

参数	类型	描述
filePath	in	许可文件路径(虹软开放平台开发者中心端获取的文件)

返回值

成功返回 MOK、MERR_ASF_ALREADY_ACTIVATED 以及 MERR_ASF_LOCAL_EXIST_USEFUL_ACTIVE_FILE，失败详见 4.2 错误码列表。

示例代码

```
// 原始下载的离线授权文件名和激活码一致  
MRESULT res = ASFOfflineActivation("86612222C1JQJK2W.dat");  
if (MOK != res && MERR_ASF_ALREADY_ACTIVATED != res &&  
MERR_ASF_LOCAL_EXIST_USEFUL_ACTIVE_FILE != res)  
{  
    printf("ASFOfflineActivation failed: %d\n", res);  
}
```

3.5.5 ASFInitEngine

该接口至关重要，清楚的了解该接口参数的意义，可以避免一些问题以及对项目的设计都有一定的帮助。

当前版本仅支持0度角检测。

方法

```
MRESULT ASFInitEngine(  
    ASF_DetectMode      detectMode,  
    ASF_OrientPriority   detectFaceOrientPriority,  
    MInt32               detectFaceScaleVal,  
    MInt32               detectFaceMaxNum,  
    MInt32               combinedMask,  
    MHandle*             hEngine  
);
```

功能描述

初始化引擎。

参数说明

参数	类型	描述
detectMode	in	VIDEO模式:处理连续帧的图像数据 IMAGE模式:处理单张的图像数据
detectFaceOrientPriority	in	人脸检测角度，推荐单一角度检测
detectFaceScaleVal	in	识别的最小人脸比例（图片长边与人脸框长边的比值） VIDEO模式取值范围[2,32]，推荐值为16 IMAGE模式取值范围[2,32]，推荐值为32
detectFaceMaxNum	in	最大需要检测的人脸个数，取值范围[1,50]
combinedMask	in	需要启用的功能组合，可多选
hEngine	out	引擎句柄

重点参数详细说明

• detectMode

宏	值	模式
ASF_DETECT_MODE_VIDEO	0x00000000	VIDEO模式
ASF_DETECT_MODE_IMAGE	0xFFFFFFFF	IMAGE模式

VIDEO 模式

1. 对视频流中的人脸进行追踪，人脸框平滑过渡，不会出现跳框的现象。
2. 使用摄像头是需要做预览显示，每一帧都需要做人脸检测，人脸检测耗时短不会出现显示卡顿的现象。
3. 在视频模式下人脸追踪和带有一个 FaceId 值，标记一张人脸从进入画面直到离开画面，FaceId 值不变，这可用于业务中优化程序性能。

IMAGE 模式

1. 针对单张图片进行人脸检测精度更高。
2. 在注册人脸库时，我们建议使用精度更高的IMAGE模式。

模式选择

1. 从摄像头中获取数据并需要预览显示，推荐选择VIDEO模式；
2. 处理静态图像数据，类似注册人脸库时，推荐使用IMAGE模式；
3. 若同时需要进行IMAGE模式人脸检测和VIDEO模式人脸检测，需要创建一个VIDEO模式的引擎和一个IMAGE模式的引擎；

• detectFaceOrientPriority

注：设置0度方向并不是说只有0度角可以检测到人脸，而是大体在这个方向上的人脸均可以，我们也提供了全角度方向检测，但在应用场景确定的情况下我们还是推荐使用单一角度进行检测，因为单一角相对于全角度性能更好、精度更高。



0度角



90度角



180度角



270度角

• detectFaceScaleVal

识别的最小人脸比例 = 图片长边 / 人脸框长边的比值

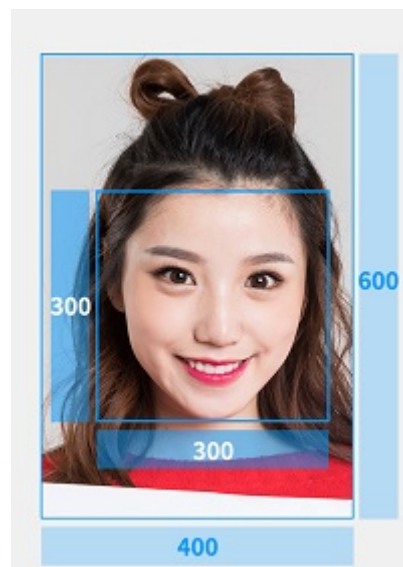
默认推荐值：VIDEO模式推荐16；IMAGE模式推荐32

//如下图所示

//图片尺寸 400 x 600

//人脸尺寸 300 x 300

`minScale = 600 / 300` //若为小数，向上取整（例如：2.53 取值为 3）



设置推荐

根据使用场景适当的设置该值效果会更好，门禁场景下我们推荐使用 VIDEO 模式，`detectFaceScaleVal` 设置为16，但是如果用户使用场景下的成像质量比较高，这我们设置为32，可以在相对较远的位置就可以检测到人脸并比对通过，这样效果更佳，达到无感通行。

• combinedMask

针对算法功能会有常量值与之——对应，可根据业务需求进行自由选择，不需要的属性可以不用初始化，否则会占用多余内存。


```

#define ASF_NONE                0x00000000 //无属性
#define ASF_FACE_DETECT        0x00000001 //人脸检测
#define ASF_FACERECOGNITION    0x00000004 //人脸特征
#define ASF_AGE                0x00000008 //年龄
#define ASF_GENDER             0x00000010 //性别
#define ASF_FACE3DANGLE        0x00000020 //3D角度
#define ASF_FACELANDMARK       0x00000040 //额头区域特征点
#define ASF_LIVENESS           0x00000080 //RGB活体
#define ASF_IR_LIVENESS        0x00000400 //IR活体
#define ASF_FACESHELTER        0x00000800 //人脸遮挡
#define ASF_MASKDETECT         0x00001000 //口罩检测

```

说明：

1. 人脸识别时一般都需要 `ASF_FACE_DETECT` 和 `ASF_FACERECOGNITION` 这两个属性。
2. 需要防止纸张、屏幕等攻击可以传入 `ASF_LIVENESS` 和 `ASF_IR_LIVENESS`，RGB 和 IR 活体检测根据用户的摄像头类型以及实际的业务需求来决定如何选择。
3. `ASF_AGE` / `ASF_GENDER` / `ASF_FACE3DANGLE` 根据业务需求进行选择即可。

这些属性均是以常量值进行定义，可通过 `|` 位运算符进行组合使用。

例如：`MInt32 combinedMask = ASF_FACE_DETECT | ASF_FACERECOGNITION | ASF_LIVENESS;`

返回值

成功返回 `MOK`，失败详见 [4.2 错误码列表](#)。

示例代码

```

MHandle handle = NULL;
MInt32 nScale = 32;
MInt32 faceNum = 5;
MInt32 mask = ASF_FACE_DETECT | ASF_FACERECOGNITION | ASF_AGE | ASF_GENDER |
ASF_FACE3DANGLE | ASF_LIVENESS | ASF_IR_LIVENESS |
ASF_FACELANDMARK | ASF_FACESHELTER | ASF_MASKDETECT;

MRESULT res = ASFInitEngine(ASF_DETECT_MODE_IMAGE, ASF_OP_0_ONLY, nScale,
faceNum, mask, &handle);
if (res != MOK)
{
    printf("ASFInitEngine failed: %d\n", res);
}
else
{
    printf("ASFInitEngine suceeded: %d\n", res);
}

```

3.5.6 ASFDetectFaces

该接口依赖初始化的模式选择，VIDEO模式下调用的人脸追踪功能，IMAGE模式下调用的人脸检测功能。初始化中 `detectFaceOrientPriority`、`detectFaceScaleVal`、`detectFaceMaxNum` 参数的设置，对能否检测到人脸以及检测到几张人脸都有决定性的作用。

方法

```

MRESULT ASFDetectFaces(
    MHandle          hEngine,
    MInt32           width,
    MInt32           height,
    MInt32           format,
    MUInt8*          imgData,
    LPASF_MultiFaceInfo detectedFaces,
    ASF_DetectModel  detectModel = ASF_DETECT_MODEL_RGB
);

```

功能描述

检测人脸信息。

参数说明

参数	类型	描述
hEngine	in	引擎句柄
width	in	图片宽度，为4的倍数
height	in	图片高度，YUYV/I420/NV21/NV12格式为2的倍数； BGR24/GRAY/DEPTH_U16格式无限制；
format	in	图像的颜色格式
imgData	in	图像数据
detectedFaces	out	检测到的人脸信息
detectModel = ASF_DETECT_MODEL_RGB	in	预留字段，当前版本使用默认参数即可

返回值

成功返回 MOK，失败详见 [4.2 错误码列表](#)。

示例代码

这里依赖 `opencv` 对图像进行处理，图像宽度需要四字节对齐（宽度为4的倍数），否则需要裁剪。

```

//opencv方式裁剪图片
void CutIplImage(IplImage* src, IplImage* dst, int x, int y)
{
    cvSize size = cvSize(dst->width, dst->height); //区域大小
    cvSetImageROI(src, cvRect(x, y, size.width, size.height)); //设置源图像ROI
    cvCopy(src, dst); //复制图像
    cvResetImageROI(src); //源图像用完后，清空ROI
}

```

```

IplImage* originalImg = cvLoadImage("1280 x 720.jpg");

```

//图像裁剪，宽度做四字节对齐，若能保证图像是四字节对齐这步可以不用做

```

IplImage* img = cvCreateImage(cvSize(originalImg->width - originalImg->width %
4, originalImg->height), IPL_DEPTH_8U, originalImg->nChannels);
CutIplImage(originalImg, img, 0, 0);

ASF_MultiFaceInfo detectedFaces = { 0 };
if (img)
{
    MRESULT res = ASFDetectFaces(handle, img->width, img->height,
ASVL_PAF_RGB24_B8G8R8, (MUInt8*)img->imageData, &detectedFaces);
    if (MOK != res)
    {
        printf("ASFDetectFaces failed: %d\n", res);
    }
    else
    {
        // 打印人脸检测结果
        for (int i = 0; i < detectedFaces.faceNum; i++)
        {
            printf("Face Id: %d\n", detectedFaces.faceID[i]);
            printf("Face Orient: %d\n", detectedFaces.faceOrient[i]);
            printf("Face Rect: (%d %d %d %d)\n",
                detectedFaces.faceRect[i].left, detectedFaces.faceRect[i].top,
                detectedFaces.faceRect[i].right,
                detectedFaces.faceRect[i].bottom);
        }
    }

    //释放图像内存，这里只是做人脸检测，若还需要做特征提取等处理，图像数据没必要释放这么早
    cvReleaseImage(&img);
}
cvReleaseImage(&originalImg);

```

3.5.7 ASFDetectFacesEx

该接口依赖初始化的模式选择，VIDEO模式下调用的人脸追踪功能，IMAGE模式下调用的人脸检测功能。初始化中 `detectFaceOrientPriority`、`detectFaceScaleVal`、`detectFaceMaxNum` 参数的设置，对能否检测到人脸以及检测到几张人脸都有决定性的作用。

方法

```

MRESULT ASFDetectFacesEx(
    MHandle          hEngine,
    LPASF_ImageData  imgData,
    LPASF_MultiFaceInfo detectedFaces,
    ASF_DetectModel  detectModel = ASF_DETECT_MODEL_RGB
);

```

功能描述

检测人脸信息。

注：该接口与 `ASFDetectFaces` 功能一致，但采用结构体的形式传入图像数据，对更高精度的图像兼容性更好。

参数说明

参数	类型	描述
hEngine	in	引擎句柄
imgData	in	图像数据
detectedFaces	out	检测到的人脸信息
detectModel = ASF_DETECT_MODEL_RGB	in	预留字段，当前版本使用默认参数即可

返回值

成功返回 MOK，失败详见 [4.2 错误码列表](#)。

示例代码

这里依赖 `opencv` 对图像进行处理，图像宽度需要四字节对齐（宽度为4的倍数），否则需要裁剪。

```
//opencv方式裁剪图片
void CutIplImage(IplImage* src, IplImage* dst, int x, int y)
{
    CvSize size = cvSize(dst->width, dst->height); //区域大小
    cvSetImageROI(src, cvRect(x, y, size.width, size.height)); //设置源图像ROI
    cvCopy(src, dst); //复制图像
    cvResetImageROI(src); //源图像用完后，清空ROI
}

IplImage* originalImg = cvLoadImage("1280 x 720.jpg");

//图像裁剪，宽度做四字节对齐，若能保证图像是四字节对齐这步可以不用做
IplImage* img = cvCreateImage(cvSize(originalImg->width - originalImg->width %
4, originalImg->height), IPL_DEPTH_8U, originalImg->nChannels);
CutIplImage(originalImg, img, 0, 0);

//图像数据以结构体形式传入，对更高精度的图像兼容性更好
ASVLOFFSCREEN offscreen = { 0 };
offscreen.u32PixelFormat = ASVL_PAF_RGB24_B8G8R8;
offscreen.i32Width = img->width;
offscreen.i32Height = img->height;
offscreen.pi32Pitch[0] = img->widthStep;
offscreen.ppu8Plane[0] = (MUInt8*)img->imageData;

ASF_MultiFaceInfo detectedFaces = { 0 };

if (img)
{
    MRESULT res = ASFDetectFacesEx(handle, &offscreen, &detectedFaces);
    if (MOK != res)
    {
        printf("ASFDetectFacesEx failed: %d\n", res);
    }
    else
    {
        // 打印人脸检测结果
        for (int i = 0; i < detectedFaces.faceNum; i++)
        {
```

```

        printf("Face Id: %d\n", detectedFaces.faceID[i]);
        printf("Face Orient: %d\n", detectedFaces.faceOrient[i]);
        printf("Face Rect: (%d %d %d %d)\n",
            detectedFaces.faceRect[i].left, detectedFaces.faceRect[i].top,
            detectedFaces.faceRect[i].right,
            detectedFaces.faceRect[i].bottom);
    }
}

//释放图像内存，这里只是做人脸检测，若还需要做特征提取等处理，图像数据没必要释放这么早
cvReleaseImage(&img);
}
cvReleaseImage(&originalImg);

```

3.5.8 ASFImageQualityDetectEx

该接口针对人脸区域的图像进行质量检测，不是针对整张图像。

方法

```

MRESULT ASFImageQualityDetectEx(
    MHandle          hEngine,
    LPASF_ImageData  imgData,
    LPASF_SingleFaceInfo faceInfo,
    MFloat*          confidenceLevel,
    ASF_DetectModel  detectModel = ASF_DETECT_MODEL_RGB
);

```

功能描述

支持多人脸图像质量检测。

参数说明

参数	类型	描述
hEngine	in	引擎句柄
imgData	in	图像数据
faceInfo	in	人脸位置信息
confidenceLevel	out	人脸图像质量检测结果
detectModel = ASF_DETECT_MODEL_RGB	in	预留字段，当前版本使用默认参数即可

返回值

成功返回 MOK，失败详见 [4.2 错误码列表](#)。

示例代码

依赖 ASFDetectFaces 接口成功检测到人脸，将检测到的人脸信息传入到该接口进行图像质量检测。

```

pImage* img = cvLoadImage("1280 x 720.jpg");
ASF_MultiFaceInfo detectedFaces = { 0 };
.....

```

```
// 以上代表检测到人脸

//图像数据以结构体形式传入，对更高精度的图像兼容性更好
ASVLOFFSCREEN offscreen = { 0 };
offscreen.u32PixelFormat = ASVL_PAF_RGB24_B8G8R8;
offscreen.i32Width = img->width;
offscreen.i32Height = img->height;
offscreen.pi32Pitch[0] = img->widthStep;
offscreen.ppu8Plane[0] = (MUInt8*)img->imageData;

ASF_SingleFaceInfo singleDetectedFaces = { 0 };
singleDetectedFaces.faceRect.left = detectedFaces.faceRect[0].left;
singleDetectedFaces.faceRect.top = detectedFaces.faceRect[0].top;
singleDetectedFaces.faceRect.right = detectedFaces.faceRect[0].right;
singleDetectedFaces.faceRect.bottom = detectedFaces.faceRect[0].bottom;
singleDetectedFaces.faceOrient = detectedFaces.faceOrient[0];

MFloat imageQualityConfidenceLevel;
res = ASFImageQualityDetectEx(handle, &offscreen, &singleDetectedFaces,
&imageQualityConfidenceLevel);
if (MOK != res)
{
    printf("ASFImageQualityDetectEx failed: %d\n", res);
}

cvReleaseImage(&img);
```

3.5.9 ASFFaceFeatureExtract

在进行第二次特征提取时，会覆盖第一次特征提取的结果。

例如：1:1的比对分别对两张图片进行特征提取，若使用同一个引擎，第一次特征提取需要拷贝保存，再进行第二次特征提取，否则在比对时输出的结果为1。

方法

```
MRESULT ASFFaceFeatureExtract(
    MHandle          hEngine,
    MInt32           width,
    MInt32           height,
    MInt32           format,
    MUInt8*          imgData,
    LPASF_SingleFaceInfo faceInfo,
    LPASF_FaceFeature feature,
    ASF_RegisterOrNot registerOrNot = ASF_RECOGNITION,
    MInt32           mask = 0
);
```

功能描述

单人脸特征提取。

参数说明

参数	类型	描述
hEngine	in	引擎句柄
width	in	图片宽度，为4的倍数
height	in	图片高度，YUYV/I420/NV21/NV12格式为2的倍数； BGR24/GRAY/DEPTH_U16格式无限制；
format	in	图像的颜色格式
imgData	in	图像数据
faceInfo	in	单人脸信息（人脸框、人脸角度）
feature	out	提取到的人脸特征信息
registerOrNot	in	注册照：ASF_REGISTER 识别照：ASF_RECOGNITION
mask	in	带口罩 1，否则0

返回值

成功返回 MOK，失败详见 [4.2 错误码列表](#)。

示例代码

依赖 ASFDetectFaces 接口成功检测到人脸，将检测到的人脸信息取单张人脸信息和使用的图像信息传入到该接口进行特征提取。

```
pImage* img = cvLoadImage("1280 x 720.jpg");
ASF_MultiFaceInfo detectedFaces = { 0 };
.....
// 以上代表检测到人脸

//这里取检测到的第一张人脸进行特征提取，也可以对人脸大小进行排序，取最大人脸检测，可根据实际应用场景进行选择。
ASF_FaceFeature feature = { 0 };
ASF_SingleFaceInfo singleDetectedFaces = { 0 };
singleDetectedFaces.faceRect.left = detectedFaces.faceRect[0].left;
singleDetectedFaces.faceRect.top = detectedFaces.faceRect[0].top;
singleDetectedFaces.faceRect.right = detectedFaces.faceRect[0].right;
singleDetectedFaces.faceRect.bottom = detectedFaces.faceRect[0].bottom;
singleDetectedFaces.faceOrient = detectedFaces.faceOrient[0];

// 这里以注册照为示例代码，注册照要求不戴口罩
MRESULT res = ASFFaceFeatureExtract(handle, img->width, img->height,
ASVL_PAF_RGB24_B8G8R8, (MUInt8*)img->imageData,
&singleDetectedFaces, &feature, ASF_RECOGNITION, 0);
if (MOK != res)
{
    printf("ASFFaceFeatureExtract failed: %d\n", res);
}

cvReleaseImage(&img);
```

3.5.10 ASFFaceFeatureExtractEx

在进行第二次特征提取时，会覆盖第一次特征提取的结果。

1:1 的比对分别对两张图片进行特征提取，若使用同一个引擎，第一次特征提取需要拷贝保存，再进行第二次特征提取，否则在比对时输出的结果为1。

1:N 搜索可以预先提取N张人脸特征存放在数据库或缓存中，进行比对识别。

方法

```
MRESULT ASFFaceFeatureExtractEx(  
    MHandle          hEngine,  
    LPASF_ImageData  imgData,  
    LPASF_SingleFaceInfo faceInfo,  
    LPASF_FaceFeature feature,  
    ASF_RegisterOrNot registerOrNot = ASF_RECOGNITION,  
    MInt32            mask = 0  
);
```

功能描述

单人特征提取。

注：该接口与 `ASFFaceFeatureExtract` 功能一致，但采用结构体的形式传入图像数据，对更高精度的图像兼容性更好。

参数说明

参数	类型	描述
hEngine	in	引擎句柄
imgData	in	图像数据
faceInfo	in	单人脸信息（人脸框、人脸角度）
feature	out	提取到的人脸特征信息
registerOrNot	in	注册照：ASF_REGISTER 识别照：ASF_RECOGNITION
mask	out	戴口罩 1，否则0

返回值

成功返回 `MOK`，失败详见 [4.2 错误码列表](#)。

示例代码

依赖 `ASFDetectFacesEx` 接口成功检测到人脸，将检测到的人脸信息取单张人脸信息和使用的图像信息传入到该接口进行特征提取。

```
pImage* img = cvLoadImage("1280 x 720.jpg");  
ASF_MultiFaceInfo detectedFaces = { 0 };  
.....  
// 以上代表检测到人脸
```

//这里取检测到的第一张人脸进行特征提取，也可以对人脸大小进行排序，取最大人脸检测，可根据实际应用场景进行选择。


```

ASF_FaceFeature feature = { 0 };
ASF_SingleFaceInfo singleDetectedFaces = { 0 };
singleDetectedFaces.faceRect.left = detectedFaces.faceRect[0].left;
singleDetectedFaces.faceRect.top = detectedFaces.faceRect[0].top;
singleDetectedFaces.faceRect.right = detectedFaces.faceRect[0].right;
singleDetectedFaces.faceRect.bottom = detectedFaces.faceRect[0].bottom;
singleDetectedFaces.faceOrient = detectedFaces.faceOrient[0];

//图像数据以结构体形式传入，对更高精度的图像兼容性更好
ASVLOFFSCREEN offscreen = { 0 };
offscreen.u32PixelFormat = ASVL_PAF_RGB24_B8G8R8;
offscreen.i32Width = img->width;
offscreen.i32Height = img->height;
offscreen.pi32Pitch[0] = img->widthStep;
offscreen.ppu8Plane[0] = (MUInt8*)img->imageData;

//口罩检测
MInt32 processMask = ASF_MASKDETECT;
res = ASFProcessEx(handle, &offscreen2, &detectedFaces2, processMask);
if (res != MOK)
{
    printf("ASFProcessEx failed: %d\n", res);
}

//获取是否戴口罩检测结果
ASF_MaskInfo maskInfo = { 0 };
res = ASFGetMask(handle, &maskInfo);
if (res != MOK)
{
    printf("ASFGetMask failed: %d\n", res);
}
else
{
    printf("ASFGetMask suceeded: %d\n", maskInfo.maskArray[0]);
}

// 这里以预览模式下的识别照为例
MRESULT res = ASFFaceFeatureExtractEx(handle, &offscreen, &singleDetectedFaces,
&feature, ASF_RECOGNITION, maskInfo.maskArray[0]);
if (MOK != res)
{
    printf("ASFFaceFeatureExtractEx failed: %d\n", res);
}

cvReleaseImage(&img);

```

3.5.11 ASFFaceFeatureCompare

方法

```

MRESULT ASFFaceFeatureCompare(
    MHandle          hEngine,
    LPASF_FaceFeature feature1,
    LPASF_FaceFeature feature2,
    MFloat*          confidenceLevel,
    ASF_CompareModel compareModel = ASF_LIFE_PHOTO
);

```

功能描述

人脸特征比对，输出比对相似度。

参数说明

参数	类型	描述
hEngine	in	引擎句柄
feature1	in	人脸特征
feature2	in	人脸特征
confidenceLevel	out	比对相似度
isMask	in	是否佩戴口罩，"1" 戴口罩，其他值未戴口罩
ASF_CompareModel = ASF_LIFE_PHOTO	in	选择人脸特征比对模型，默认为ASF_LIFE_PHOTO。 1. ASF_LIFE_PHOTO：用于生活照之间的特征比对，推荐 阈值0.80； 2. ASF_ID_PHOTO：用于证件照或证件照和生活照之间的 特征比对，推荐阈值0.82；

返回值

成功返回 MOK，失败详见 [4.2 错误码列表](#)。

示例代码

```
// 单人脸特征比对,这里未选择比对模型，默认使用ASF_LIFE_PHOTO
// ASF_LIFE_PHOTO相当于之前ArcFace模型
// ASF_ID_PHOTO相当于之前人证模型
// isMask 参数值一般从ASFGetMask获取结果值传入
MFloat confidenceLevel;
res = ASFFaceFeatureCompare(handle, &feature1, &feature2, 1, &confidenceLevel);
if (res != MOK)
{
    printf("ASFFaceFeatureCompare failed: %d\n", res);
}
else
{
    printf("ASFFaceFeatureCompare suceeded: %lf\n", confidenceLevel);
}
```

3.5.12 ASFSetLivenessParam

方法

```
MRESULT ASFSetLivenessParam(
    MHandle          hEngine,
    LPASF_LivenessThreshold threshold
);
```

功能描述

设置RGB/IR活体阈值，若不设置内部默认RGB：0.5 IR：0.7。

参数说明

参数	类型	描述
hEngine	in	引擎句柄
threshold	in	活体阈值，推荐RGB:0.5 IR:0.7

返回值

成功返回 MOK，失败详见 [4.2 错误码列表](#)。

示例代码

```
ASF_LivenessThreshold threshold = { 0 };
threshold.thresholdmodel_BGR = 0.5;
threshold.thresholdmodel_IR = 0.7;
res = ASFSetLivenessParam(handle, &threshold);
if (res != MOK)
{
    printf("ASFSetLivenessParam failed: %d\n", res);
}
```

3.5.13 ASFProcess

方法

该接口仅支持可见光图像检测。

```
MRESULT ASFProcess(
    MHandle          hEngine,
    MInt32           width,
    MInt32           height,
    MInt32           format,
    MUInt8*          imgData,
    LPASF_MultiFaceInfo detectedFaces,
    MInt32           combinedMask
);
```

功能描述

人脸属性检测（年龄/性别/人脸3D角度/口罩/遮挡/额头区域），最多支持4张人脸信息检测，超过部分返回未知（活体仅支持单张人脸检测，超出返回未知），接口不支持IR图像检测。

参数说明

参数	类型	描述
hEngine	in	引擎句柄
width	in	图片宽度，为4的倍数
height	in	图片高度，YUYV/I420/NV21/NV12格式为2的倍数；BGR24格式无限制；
format	in	支持YUYV/I420/NV21/NV12/BGR24
imgData	in	图像数据
detectedFaces	in	多人脸信息
combinedMask	in	1.检测的属性 (ASF_AGE、ASF_GENDER、ASF_FACE3DANGLE、ASF_LIVENESS、ASF_FACELANDMARK、ASF_FACESHELTER、ASF_MASKDETECT) 支持多选 2.检测的属性须在引擎初始化接口的combinedMask参数中启用

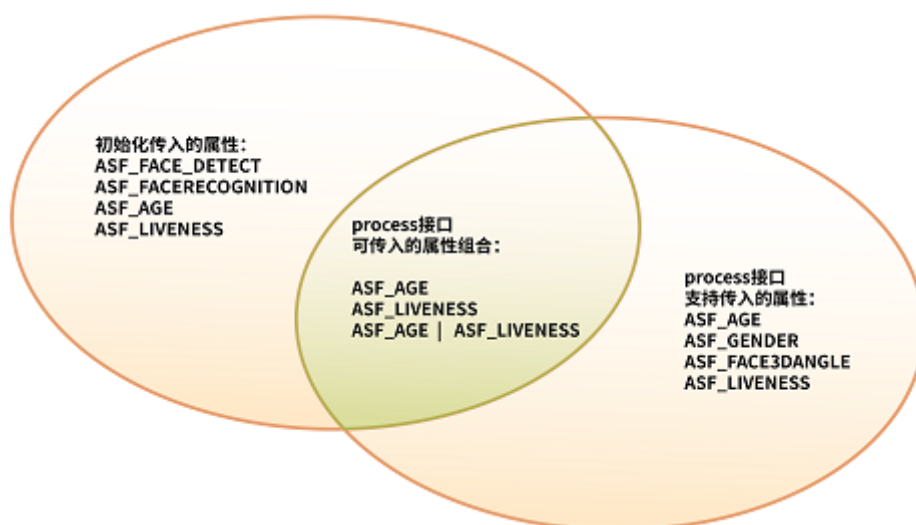
重要参数说明

• combinedMask

process接口中支持检测 ASF_AGE、ASF_GENDER、ASF_FACE3DANGLE、ASF_LIVENESS、ASF_FACELANDMARK、ASF_FACESHELTER、ASF_MASKDETECT 七种属性，但是想检测这些属性，必须在初始化引擎接口中对想要检测的属性进行初始化。

关于初始化接口中 combinedMask 和 ASFProcess 接口中 combinedMask 参数之间的关系，举例进行详细说明，如下图所示：

1. ASFProcess 接口中 combinedMask 支持传入的属性有 ASF_AGE、ASF_GENDER、ASF_FACE3DANGLE、ASF_LIVENESS、ASF_FACELANDMARK、ASF_FACESHELTER、ASF_MASKDETECT。
2. 初始化中传入了 ASF_FACE_DETECT、ASF_FACERECOGNITION、ASF_AGE、ASF_LIVENESS 属性。
3. process可传入属性组合只有 ASF_AGE、ASF_LIVENESS、ASF_AGE、ASF_LIVENESS。



返回值

成功返回 MOK，失败详见 [4.2 错误码列表](#)。

示例代码

下面代码要求初始化引擎时包含了 ASF_AGE | ASF_GENDER | ASF_FACE3DANGLE | ASF_LIVENESS | ASF_FACESHELTER | ASF_MASKDETECT | ASF_FACELANDMARK

```
plImage* img = cvLoadImage("1280 x 720.jpg");
ASF_MultiFaceInfo detectedFaces = { 0 };
.....
// 以上代表检测到人脸

MInt32 processMask = ASF_AGE | ASF_GENDER | ASF_FACE3DANGLE | ASF_LIVENESS |
ASF_FACESHELTER | ASF_MASKDETECT | ASF_FACELANDMARK;
MRESULT res = ASFProcess(handle, img->width, img->height, ASVL_PAF_RGB24_B8G8R8,
(MUInt8*)img->imageData, &detectedFaces, processMask);

if (res != MOK)
{
    printf("ASFProcess failed: %d\n", res);
}
```

3.5.14 ASFProcessEx

方法

该接口仅支持可见光图像检测。

```
MRESULT ASFProcessEx(
    MHandle          hEngine,
    LPASF_ImageData  imgData,
    LPASF_MultiFaceInfo detectedFaces,
    MInt32           combinedMask
);
```

功能描述

人脸属性检测（年龄/性别/人脸3D角度/口罩/遮挡/额头区域），最多支持4张人脸信息检测，超过部分返回未知（活体仅支持单张人脸检测，超出返回未知），接口不支持IR图像检测。

注：该接口与 ASFProcess 功能一致，但采用结构体的形式传入图像数据，对更高精度的图像兼容性更好。

参数说明

参数	类型	描述
hEngine	in	引擎句柄
imgData	in	图像数据
detectedFaces	in	多人脸信息
combinedMask	in	1.检测的属性 (ASF_AGE、ASF_GENDER、ASF_FACE3DANGLE、ASF_LIVENESS、ASF_FACELANDMARK、ASF_FACESHELTER、ASF_MASKDETECT) 支持多选 2.检测的属性须在引擎初始化接口的combinedMask参数中启用

返回值

成功返回 MOK，失败详见 [4.2 错误码列表](#)。

示例代码

下面代码要求初始化引擎时包含了 ASF_AGE | ASF_GENDER | ASF_FACE3DANGLE | ASF_LIVENESS | ASF_FACESHELTER | ASF_MASKDETECT | ASF_FACELANDMARK

```
pImage* img = cvLoadImage("1280 x 720.jpg");
ASF_MultiFaceInfo detectedFaces = { 0 };
.....
// 以上代表检测到人脸

//图像数据以结构体形式传入，对更高精度的图像兼容性更好
ASVLOFFSCREEN offscreen = { 0 };
offscreen.u32PixelFormat = ASVL_PAF_RGB24_B8G8R8;
offscreen.i32Width = img->width;
offscreen.i32Height = img->height;
offscreen.pi32Pitch[0] = img->widthStep;
offscreen.ppu8Plane[0] = (MUInt8*)img->imageData;

MInt32 processMask = ASF_AGE | ASF_GENDER | ASF_FACE3DANGLE | ASF_LIVENESS |
ASF_FACESHELTER | ASF_MASKDETECT | ASF_FACELANDMARK;
MRESULT res = ASFProcessEx(handle, &offscreen, &detectedFaces, processMask);
if (res != MOK)
{
    printf("ASFProcessEx failed: %d\n", res);
}
```

3.5.15 ASFGetAge

方法

```
MRESULT ASFGetAge(
    MHandle          hEngine,
    LPASF_AgeInfo    ageInfo
);
```

功能描述

获取年龄信息。

参数说明

参数	类型	描述
hEngine	in	引擎句柄
ageInfo	out	检测到的年龄信息数组

返回值

成功返回 MOK，失败详见 [4.2 错误码列表](#)。

示例代码

前提：ASFProcess 接口调用成功，且combinedMask参数中包含 ASF_AGE 属性

```
ASF_AgeInfo ageInfo = { 0 };
MRESULT res = ASFGetAge(handle, &ageInfo);
if (res != MOK)
{
    printf("ASFGetAge failed: %d\n", res);
}
else
{
    for (int i = 0; i < ageInfo.num; i++)
    {
        printf("Age %d: %d\n", i, ageInfo.ageArray[i]);
    }
}
```

3.5.16 ASFGetGender

方法

```
MRESULT ASFGetGender(
    MHandle          hEngine,
    LPASF_GenderInfo genderInfo
);
```

功能描述

获取性别信息。

参数说明

参数	类型	描述
hEngine	in	引擎句柄
genderInfo	out	检测到的性别信息数组

返回值

成功返回 MOK，失败详见 [4.2 错误码列表](#)。

示例代码

前提：ASFProcess 接口调用成功，且combinedMask参数中包含 ASF_GENDER 属性

```
ASF_GenderInfo genderInfo = { 0 };
MRESULT res = ASFGetGender(handle, &genderInfo);
if (res != MOK)
{
    printf("ASFGetGender failed: %d\n", res);
}
else
{
    for (int i = 0; i < genderInfo.num; i++)
    {
        printf("Gender %d: %d\n", i, genderInfo.genderArray[i]);
    }
}
```

3.5.17 ASFGetFace3DAngle

方法

```
MRESULT ASFGetFace3DAngle(
    MHandle      hEngine,
    LPASF_Face3DAngle  p3DAngleInfo
);
```

功能描述

获取3D角度信息。

参数说明

参数	类型	描述
hEngine	in	引擎句柄
p3DAngleInfo	out	检测到的3D角度信息数组

返回值

成功返回 MOK，失败详见 [4.2 错误码列表](#)。

示例代码

前提：ASFProcess 接口调用成功，且combinedMask参数中包含 ASF_FACE3DANGLE 属性


```

ASF_Face3DAngle angleInfo = { 0 };
MRESULT res = ASFGetFace3DAngle(handle, &angleInfo);
if (res != MOK)
{
    printf("ASFGetFace3DAngle failed: %d\n", res);
}
else
{
    for (int i = 0; i < angleInfo.num; i++)
    {
        printf("3DAngle %d\n", i);
        printf("roll: %f yaw: %f pitch: %f\n", i,
            angleInfo.roll[i], angleInfo.yaw[i], angleInfo.pitch[i]);
    }
}

```

3.5.18 ASFGetLivenessScore

方法

```

MRESULT ASFGetLivenessScore(
    MHandle          hEngine,
    LPASF_LivenessInfo livenessInfo
);

```

功能描述

获取RGB活体信息。

参数说明

参数	类型	描述
hEngine	in	引擎句柄
livenessInfo	out	检测到的活体信息

返回值

成功返回 `MOK`，失败详见 [4.2 错误码列表](#)。

示例代码

前提：`ASFProcess` 接口调用成功，且 `combinedMask` 参数中包含 `ASF_LIVENESS` 属性

```

ASF_LivenessInfo rgbLivenessInfo = { 0 };
MRESULT res = ASFGetLivenessScore(handle, &rgbLivenessInfo);
if (res != MOK)
{
    printf("ASFGetLivenessScore failed: %d\n", res);
}
else
{
    //只会输出第一张人脸结果，不需要循环打印
    printf("RGB Liveness: %d\n", rgbLivenessInfo.isLive[0]);
}

```

3.5.19 ASFSetFaceShelterParam

方法

```
MRESULT ASFSetFaceShelterParam(  
    MHandle hEngine,  
    MFloat  ShelterThreshold  
);
```

功能描述

设置遮挡算法检测的阈值。

参数说明

参数	类型	描述
hEngine	in	引擎句柄
ShelterThreshold	out	设置遮挡算法检测的阈值

返回值

成功返回 MOK，失败详见 [4.2 错误码列表](#)。

示例代码

```
MFloat ShelterThreshold = 0.8f; //推荐阈值为0.8，可根据实际情况进行调整  
MRESULT res = ASFSetFaceShelterParam(handle, ShelterThreshold);  
if (res != MOK)  
{  
    printf("ASFSetFaceShelterParam failed: %d", res);  
}
```

3.5.20 ASFGetFaceShelter

方法

```
MRESULT ASFGetFaceShelter(  
    MHandle hEngine,  
    LPASF_FaceShelter pFaceShelterInfo  
);
```

功能描述

获取人脸是否被遮挡。

参数说明

参数	类型	描述
hEngine	in	引擎句柄
pFaceShelterInfo	out	检测到人脸是否被遮挡的信息

返回值

成功返回 `MOK`，失败详见 [4.2 错误码列表](#)。

示例代码

前提：`ASFProcessEx` 接口调用成功，且 `combinedMask` 参数中包含 `ASF_FACESHELTER` 属性。

```
ASF_FaceShelter faceShelterInfo = { 0 };
MRESULT res = ASFGetFaceShelter(handle, &faceShelterInfo);
if (res != MOK)
{
    printf("ASFGetFaceShelter failed: %d\n", res);
}
else
{
    printf("ASFGetFaceShelter succeeded: %d\n", faceShelterInfo.FaceShelter[0]);
}
```

3.5.21 ASFGetMask

方法

```
MRESULT ASFGetMask(
    MHandle hEngine,
    LPASF_MaskInfo maskInfo
);
```

功能描述

获取人脸是否戴口罩。

参数说明

参数	类型	描述
hEngine	in	引擎句柄
maskInfo	out	检测到人脸是否戴口罩的信息

返回值

成功返回 `MOK`，失败详见 [4.2 错误码列表](#)。

示例代码

前提：`ASFProcessEx` 接口调用成功，且 `combinedMask` 参数中包含 `ASF_MASKDETECT` 属性。

```
ASF_MaskInfo maskInfo = { 0 };
MRESULT res = ASFGetMask(handle, &maskInfo);
if (res != MOK)
{
    printf("ASFGetMask failed: %d\n", res);
}
else
{
    printf("ASFGetMask succeeded: %d\n", maskInfo.maskArray[0]);
}
```

3.5.22 ASFGetFaceLandMark

方法

```
MRESULT ASFGetFaceLandMark(  
    MHandle          engine,  
    LPASF_LandMarkInfo LandMarkInfo  
);
```

功能描述

获取额头区域位置。

参数说明

参数	类型	描述
hEngine	in	引擎句柄
LandMarkInfo	out	人脸额头点数组，每张人脸额头区域通过四个点表示

返回值

成功返回 MOK，失败详见 [4.2 错误码列表](#)。

示例代码

```
前提：ASFProcessEx 接口调用成功，且combinedMask参数中包含 ASF_FACELANDMARK 属性。  
  
ASF_LandMarkInfo headLandMarkInfo = { 0 };  
MRESULT res = ASFGetFaceLandMark(handle, &headLandMarkInfo);  
if (res != MOK)  
{  
    printf("ASFGetFaceLandMark failed: %d\n", res);  
}  
else  
{  
    printf("ASFGetFaceLandMark sucesseced: %d\n", headLandMarkInfo.num);  
}
```

3.5.23 ASFProcess_IR

方法

```
MRESULT ASFProcess_IR(  
    MHandle      hEngine,  
    MInt32       width,  
    MInt32       height,  
    MInt32       format,  
    MUInt8*      imgData,  
    LPASF_MultiFaceInfo detectedFaces,  
    MInt32       combinedMask  
);
```

功能描述

该接口仅支持单人脸 IR 活体检测，超出返回未知。

参数说明

参数	类型	描述
hEngine	in	引擎句柄
width	in	图片宽度，为4的倍数
height	in	图片高度
format	in	图像颜色格式
imgData	in	图像数据
detectedFaces	in	多人脸信息
combinedMask	in	目前仅支持 ASF_IR_LIVENESS

返回值

成功返回 MOK，失败详见 [4.2 错误码列表](#)。

示例代码

下面代码要求初始化引擎时包含了 ASF_IR_LIVENESS

```
cv::Mat img = cv::imread("1280 x 720.jpg");
ASF_MultiFaceInfo detectedFaces = { 0 };
.....
// 以上代表检测到人脸

//opencv读图时会把灰度图转换成 RGB 颜色格式，需要转换成 GRAY 格式进行IR活体检测
cv::Mat grayMat;
cv::cvtColor(img, grayMat, CV_BGR2GRAY);

MInt32 processIRMask = ASF_IR_LIVENESS;
MRESULT res = ASFProcess_IR(handle, grayMat.cols, grayMat.rows, ASVL_PAF_GRAY,
(MUInt8*)grayMat.data, &detectedFaces, processIRMask);
if (res != MOK)
{
    printf("ASFProcess_IR failed: %d\n", res);
}
```

3.5.24 ASFProcessEx_IR

方法

```
MRESULT ASFProcessEx_IR(
    MHandle          hEngine,
    LPASF_ImageData  imgData,
    LPASF_MultiFaceInfo detectedFaces,
    MInt32           combinedMask
);
```

功能描述

该接口仅支持单人脸 IR 活体检测，超出返回未知。

注：该接口与 `ASFProcess_IR` 功能一致，但采用结构体的形式传入图像数据，对更高精度的图像兼容性更好。

参数说明

参数	类型	描述
hEngine	in	引擎句柄
imgData	in	图像数据
detectedFaces	in	多人脸信息
combinedMask	in	目前仅支持 <code>ASF_IR_LIVENESS</code>

返回值

成功返回 `MOK`，失败详见 [4.2 错误码列表](#)。

示例代码

下面代码要求初始化引擎时包含了 `ASF_IR_LIVENESS`

```
cv::Mat img = cv::imread("1280 x 720.jpg");
ASF_MultiFaceInfo detectedFaces = { 0 };
.....
// 以上代表检测到人脸

//opencv读图时会把灰度图转换成 RGB 颜色格式，需要转换成 GRAY 格式进行IR活体检测
cv::Mat grayMat;
cv::cvtColor(img, grayMat, CV_BGR2GRAY);

MInt32 processIRMask = ASF_IR_LIVENESS;

//图像数据以结构体形式传入，对更高精度的图像兼容性更好
ASVLOFFSCREEN offscreen = { 0 };
offscreen.u32PixelFormat = ASVL_PAF_GRAY;
offscreen.i32Width = grayMat.cols;
offscreen.i32Height = grayMat.rows;
offscreen.pi32Pitch[0] = grayMat.step;
offscreen.ppu8Plane[0] = (MUInt8*)grayMat.data;

MRESULT res = ASFProcessEx_IR(handle, &offscreen, &detectedFaces,
processIRMask);
if (res != MOK)
{
    printf("ASFProcessEx_IR failed: %d\n", res);
}
```

3.5.25 ASFGetLivenessScore_IR

方法

```
MRESULT ASFGetLivenessScore_IR(
    MHandle          hEngine,
    LPASF_LivenessInfo livenessInfo
);
```

功能描述

获取IR活体信息。

参数说明

参数	类型	描述
hEngine	in	引擎句柄
livenessInfo	out	检测到的IR活体信息

返回值

成功返回 MOK，失败详见 [4.2 错误码列表](#)。

示例代码

前提：ASFProcess_IR 接口调用成功，且combinedMask参数中包含 ASF_IR_LIVENESS 属性。

```
ASF_LivenessInfo irLivenessInfo = { 0 };
MRESULT res = ASFGetLivenessScore_IR(handle, &irLivenessInfo);
if (res != MOK)
{
    printf("ASFGetLivenessScore_IR failed: %d\n", res);
}
else
{
    //只会输出第一张人脸结果，不需要循环打印
    printf("RGB Liveness: %d\n", irLivenessInfo.isLive[0]);
}
```

3.5.26 ASFGetVersion

方法

```
const ASF_VERSION ASFGetVersion();
```

功能描述

获取SDK版本信息。

返回值

成功返回版本信息，失败返回MNull。

示例代码

注意：无需激活或初始化即可调用

```
const ASF_VERSION version = ASFGetVersion();
printf("Version:%s\n", version.Version);
printf("BuildDate:%s\n", version.BuildDate);
printf("CopyRight:%s\n", version.CopyRight);
```

3.5.27 ASFUninitEngine

方法

```
MRESULT ASFUninitEngine(  
    MHandle hEngine  
);
```

功能描述

销毁SDK引擎。

参数说明

参数	类型	描述
hEngine	in	引擎句柄

返回值

成功返回 MOK，失败详见 [4.2 错误码列表](#)。

示例代码

```
MRESULT res = ASFUninitEngine(handle);  
if (res != MOK)  
{  
    printf("ALUninitEngine failed: %d\n", res);  
}
```

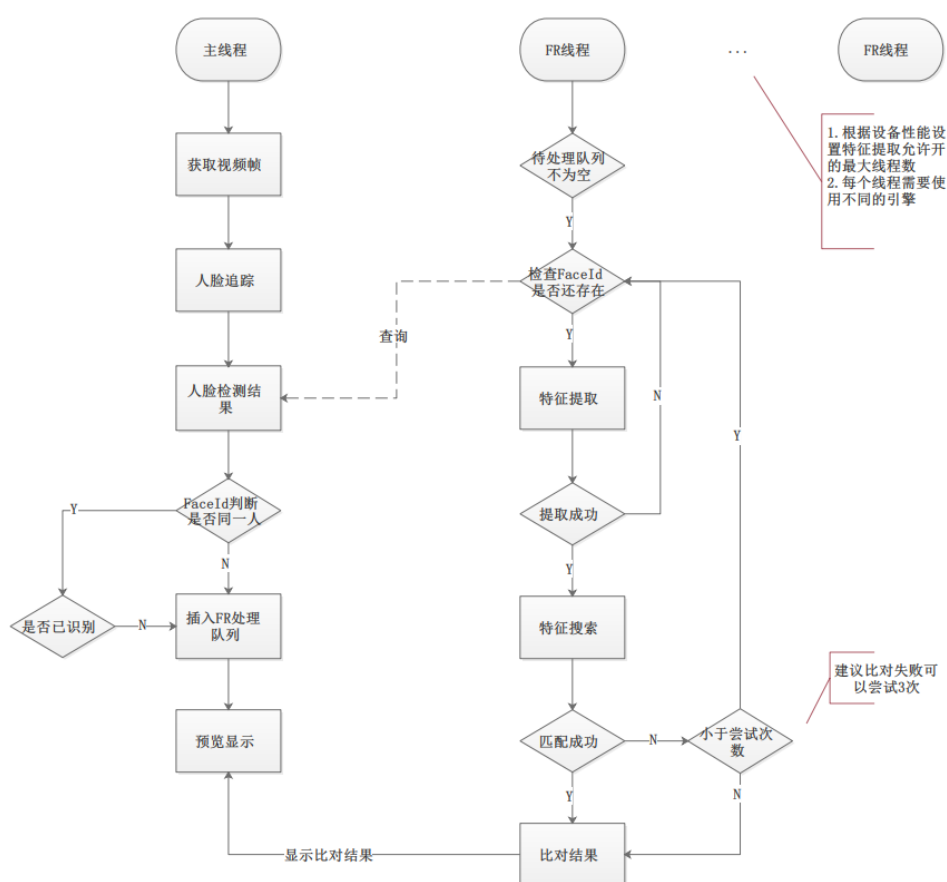

4. 常见问题

4.1 常用接入场景流程

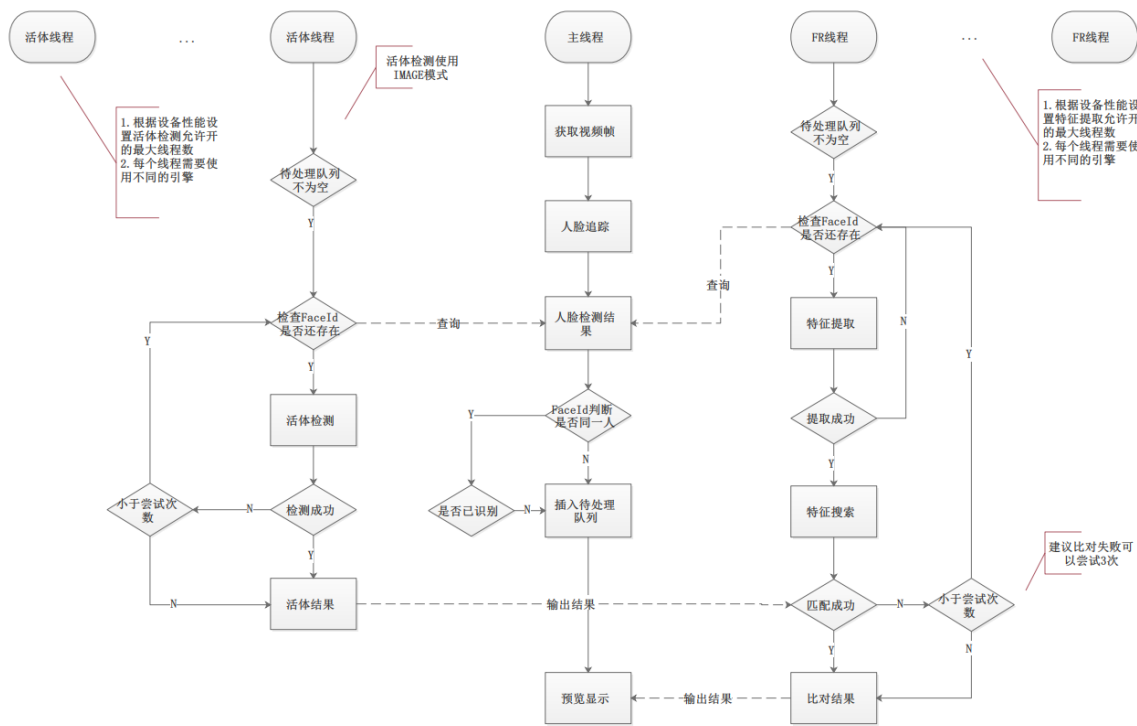
一般情况下使用人脸识别SDK需对每一帧图像数据做人脸检测、特征提取等操作，但这样往往消耗系统资源，这里引入一些优化策略作为参考：

- FaceId 标记一张人脸从进入画面到离开画面这个值不变，可以使用 FaceId 来判断从用户进入画面到离开画面只需做一次人脸比对即可。
- 通过摄像头获取图像数据，人是动态移动的，图像质量不能保证，可以引入尝试策略，连续几次识别失败才认为比对失败，但不能无限制的尝试，在极限情况下可能会影响效果，推荐尝试3-5次即可。

4.1.1 常用比对流程



4.1.2 常用比对流程 + 活体



4.2 错误码列表

错误码名	十六进制	十进制	描述
MOK	0x0	0	成功
MERR_UNKNOWN	0x1	1	错误原因不明
MERR_INVALID_PARAM	0x2	2	无效的参数
MERR_UNSUPPORTED	0x3	3	引擎不支持
MERR_NO_MEMORY	0x4	4	内存不足
MERR_BAD_STATE	0x5	5	状态错误
MERR_USER_CANCEL	0x6	6	用户取消相关操作
MERR_EXPIRED	0x7	7	操作时间过期
MERR_USER_PAUSE	0x8	8	用户暂停操作
MERR_BUFFER_OVERFLOW	0x9	9	缓冲上溢
MERR_BUFFER_UNDERFLOW	0xA	10	缓冲下溢
MERR_NO_DISKSPACE	0xB	11	存储空间不足
MERR_COMPONENT_NOT_EXIST	0xC	12	组件不存在
MERR_GLOBAL_DATA_NOT_EXIST	0xD	13	全局数据不存在
MERRP_IMGCODEC	0xE	14	组件不存在
MERR_FILE_GENERAL	0xF	15	全局数据不存在
MERR_FSDK_INVALID_APP_ID	0x7001	28673	无效的AppId
MERR_FSDK_INVALID_SDK_ID	0x7002	28674	无效的SDKKey
MERR_FSDK_INVALID_ID_PAIR	0x7003	28675	AppId和SDKKey不匹配
MERR_FSDK_MISMATCH_ID_AND_SDK	0x7004	28676	SDKKey和使用的SDK不匹配,请检查入参
MERR_FSDK_SYSTEM_VERSION_UNSUPPORTED	0x7005	28677	系统版本不被当前SDK所支持
MERR_FSDK_LICENCE_EXPIRED	0x7006	28678	SDK有效期过期,需要重新下载更新
MERR_FSDK_FR_INVALID_MEMORY_INFO	0x12001	73729	无效的输入内存
MERR_FSDK_FR_INVALID_IMAGE_INFO	0x12002	73730	无效的输入图像参数
MERR_FSDK_FR_INVALID_FACE_INFO	0x12003	73731	无效的脸部信息
MERR_FSDK_FR_NO_GPU_AVAILABLE	0x12004	73732	当前设备无GPU可用
MERR_FSDK_FR_MISMATCHED_FEATURE_LEVEL	0x12005	73733	待比较的两个人脸特征的版本不一致
MERR_FSDK_FACEFEATURE_UNKNOWN	0x14001	81921	人脸特征检测错误未知
MERR_FSDK_FACEFEATURE_MEMORY	0x14002	81922	人脸特征检测内存错误
MERR_FSDK_FACEFEATURE_INVALID_FORMAT	0x14003	81923	人脸特征检测格式错误
MERR_FSDK_FACEFEATURE_INVALID_PARAM	0x14004	81924	人脸特征检测参数错误
MERR_FSDK_FACEFEATURE_LOW_CONFIDENCE_LEVEL	0x14005	81925	人脸特征检测结果置信度低
MERR_FSDK_FACEFEATURE_EXPIRED	0x14006	81926	人脸特征检测结果操作过期
MERR_FSDK_FACEFEATURE_MISSFACE	0x14007	81927	人脸特征检测人脸丢失
MERR_ASF_EX_FEATURE_UNSUPPORTED_ON_INIT	0x15001	86017	Engine不支持的检测属性
MERR_ASF_EX_FEATURE_UNINITED	0x15002	86018	需要检测的属性未初始化
MERR_ASF_EX_FEATURE_UNPROCESSED	0x15003	86019	待获取的属性未在process中处理过
MERR_ASF_EX_FEATURE_UNSUPPORTED_ON_PROCESS	0x15004	86020	PROCESS不支持的检测属性,例如FR,有自己独立的处理函数
MERR_ASF_EX_INVALID_IMAGE_INFO	0x15005	86021	无效的输入图像
MERR_ASF_EX_INVALID_FACE_INFO	0x15006	86022	无效的脸部信息
MERR_ASF_ACTIVATION_FAIL	0x16001	90113	SDK激活失败,请打开读写权限
MERR_ASF_ALREADY_ACTIVATED	0x16002	90114	SDK已激活
MERR_ASF_NOT_ACTIVATED	0x16003	90115	SDK未激活

错误码名	十六进制	十进制	描述
MERR_ASF_SCALE_NOT_SUPPORT	0x16004	90116	detectFaceScaleVal不支持
MERR_ASF_ACTIVEFILE_SDKTYPE_MISMATCH	0x16005	90117	激活文件与SDK类型不匹配，请确认使用的sdk
MERR_ASF_DEVICE_MISMATCH	0x16006	90118	设备不匹配
MERR_ASF_UNIQUE_IDENTIFIER_ILLEGAL	0x16007	90119	唯一标识不合法
MERR_ASF_PARAM_NULL	0x16008	90120	参数为空
MERR_ASF_LIVENESS_EXPIRED	0x16009	90121	活体已过期
MERR_ASF_VERSION_NOT_SUPPORT	0x1600A	90122	版本不支持
MERR_ASF_SIGN_ERROR	0x1600B	90123	签名错误
MERR_ASF_DATABASE_ERROR	0x1600C	90124	激活信息保存异常
MERR_ASF_UNIQUE_CHECKOUT_FAIL	0x1600D	90125	唯一标识符校验失败
MERR_ASF_COLOR_SPACE_NOT_SUPPORT	0x1600E	90126	颜色空间不支持
MERR_ASF_IMAGE_WIDTH_HEIGHT_NOT_SUPPORT	0x1600F	90127	图片宽高不支持，宽度需四字节对齐
MERR_ASF_READ_PHONE_STATE_DENIED	0x16010	90128	android.permission.READ_PHONE_STATE权限被拒绝
MERR_ASF_ACTIVATION_DATA_DESTROYED	0x16011	90129	激活数据被破坏,请删除激活文件，重新进行激活
MERR_ASF_SERVER_UNKNOWN_ERROR	0x16012	90130	服务端未知错误
MERR_ASF_INTERNET_DENIED	0x16013	90131	INTERNET权限被拒绝
MERR_ASF_ACTIVEFILE_SDK_MISMATCH	0x16014	90132	激活文件与SDK版本不匹配,请重新激活
MERR_ASF_DEVICEINFO_LESS	0x16015	90133	设备信息太少，不足以生成设备指纹
MERR_ASF_LOCAL_TIME_NOT_CALIBRATED	0x16016	90134	客户端时间与服务器时间（即北京时间）前后相差在30分钟以上
MERR_ASF_APPID_DATA_DECRYPT	0x16017	90135	数据校验异常
MERR_ASF_APPID_APPKEY_SDK_MISMATCH	0x16018	90136	传入的AppId和AppKey与使用的SDK版本不一致
MERR_ASF_NO_REQUEST	0x16019	90137	短时间大量请求会被禁止请求,30分钟之后解封
MERR_ASF_ACTIVE_FILE_NO_EXIST	0x1601A	90138	激活文件不存在
MERR_ASF_DETECT_MODEL_UNSUPPORTED	0x1601B	90139	检测模型不支持，请查看对应接口说明，使用当前支持的检测模型
MERR_ASF_CURRENT_DEVICE_TIME_INCORRECT	0x1601C	90140	当前设备时间不正确，请调整设备时间
MERR_ASF_ACTIVATION_QUANTITY_OUT_OF_LIMIT	0x1601D	90141	年度激活数量超出限制，次年清零
MERR_ASF_NETWORK_COULDNT_RESOLVE_HOST	0x17001	94209	无法解析主机地址
MERR_ASF_NETWORK_COULDNT_CONNECT_SERVER	0x17002	94210	无法连接服务器
MERR_ASF_NETWORK_CONNECT_TIMEOUT	0x17003	94211	网络连接超时
MERR_ASF_NETWORK_UNKNOWN_ERROR	0x17004	94212	网络未知错误
MERR_ASF_ACTIVEKEY_COULDNT_CONNECT_SERVER	0x18001	98305	无法连接激活服务器
MERR_ASF_ACTIVEKEY_SERVER_SYSTEM_ERROR	0x18002	98306	服务器系统错误
MERR_ASF_ACTIVEKEY_POST_PARM_ERROR	0x18003	98307	请求参数错误
MERR_ASF_ACTIVEKEY_PARM_MISMATCH	0x18004	98308	ACTIVEKEY与APPID、SDKKEY不匹配
MERR_ASF_ACTIVEKEY_ACTIVEKEY_ACTIVATED	0x18005	98309	ACTIVEKEY已经被使用
MERR_ASF_ACTIVEKEY_ACTIVEKEY_FORMAT_ERROR	0x18006	98310	ACTIVEKEY信息异常
MERR_ASF_ACTIVEKEY_APPID_PARM_MISMATCH	0x18007	98311	ACTIVEKEY与APPID不匹配
MERR_ASF_ACTIVEKEY_SDK_FILE_MISMATCH	0x18008	98312	SDK与激活文件版本不匹配
MERR_ASF_ACTIVEKEY_EXPIRED	0x18009	98313	ACTIVEKEY已过期
MERR_ASF_ACTIVEKEY_DEVICE_OUT_OF_LIMIT	0x1800A	98314	批量授权激活码设备数超过限制
MERR_ASF_LICENSE_FILE_NOT_EXIST	0x19001	102401	离线授权文件不存在或无读写权限
MERR_ASF_LICENSE_FILE_DATA_DESTROYED	0x19002	102402	离线授权文件已损坏
MERR_ASF_LICENSE_FILE_SDK_MISMATCH	0x19003	102403	离线授权文件与SDK版本不匹配

错误码名	十六进制	十进制	描述
MERR_ASF_LICENSE_FILEINFO_SDKINFO_MISMATCH	0x19004	102404	离线授权文件与SDK信息不匹配
MERR_ASF_LICENSE_FILE_FINGERPRINT_MISMATCH	0x19005	102405	离线授权文件与设备指纹不匹配
MERR_ASF_LICENSE_FILE_EXPIRED	0x19006	102406	离线授权文件已过期
MERR_ASF_LOCAL_EXIST_USEFUL_ACTIVE_FILE	0x19007	102407	离线授权文件不可用，本地原有激活文件可继续使用
MERR_ASF_LICENSE_FILE_VERSION_TOO_LOW	0x19008	102408	离线授权文件版本过低，请使用新版本激活助手重新进行离线激活

