<人脸闸机解决集成说明文档—APP 方案>

V1.0 版本

版本	日期	说明
V1.0	2017.07.23	介绍基于 APP 形态的人脸核身集成方案

概念解释

1 本地版活体检测(本地活体)：通过让用户做出指定脸部交互动作，识别当前操作者是否为活体，此功能可离线使用，可设定指定动作是否使用及应用顺序。可有效抵御高清图片、3D 建模、视频等攻击。

2 在线版活体检测(在线活体)：基于本地版有交互活体检测输出的质量合格的图片，请求在线活体检测接口，可以进一步判断用户上传的图片是否为活体，进一步抵御高精度视频、高精度建模等攻击。此接口建议与本地版活体检测结合使用，且输入的照片一定为二次采集图片（即限于产品前端拍照所得，不可为相册图片）。

3 人脸质量检测：判断视频流中的图片帧中，哪些图片质量较佳，即人脸图像特征清晰（满足角度、姿态、光照、模糊度等校验）。

4 人脸图像采集：通过本地 SDK 能力，采集人脸图像，同时经过人脸质量检测，确保采集到的人脸图像符合各条件校验（角度、姿态、光照、模糊度等），为设备前端获取有效可分析人脸的主要功能。

5 在线鉴权：API 接口的权限判断，在后台获取 API Key 和 Secret Key 后生成的 token，用于每次请求时供服务器判断是否可通过此次请求。

6 离线授权：SDK 的授权判断，授权介质也称为 license，在 SDK 使用中，需要通过 license 向授权服务器发起请求，判断 SDK 的使用合法性及使用有效期。

7 人脸识别：人脸识别技术的泛称，通常指识别图片或视频中的人脸，提取人脸特征值，用于进一步的人脸对比、搜索等业务操作。

8 人脸检测：分析并定位图片中的人脸及位置，是人脸识别技术的第一步分析内容。

9 人脸 1：1：目标分析人脸与已知身份人脸进行比对，对比两个图片中的人脸的相似度，根据分值判断两张人脸是否为同一人。

10 人脸 1：N：在一个已知身份的人脸集合中找到相似的人脸，返回内容为人脸集合中每一张脸与目标人脸的相似度分值，找出目标人脸是否属于哪一个已知身份的用户。

11 人脸属性：通过人脸特征抽取，分析人脸的性别、年龄、种族、是否戴眼镜等属性值。

12 ak：对应 API KEY

13 sk：对应 Secret Key

14 LicenseID：应用授权标识

15 包名 packageName：安卓应用包名，用来唯一识别一个应用。

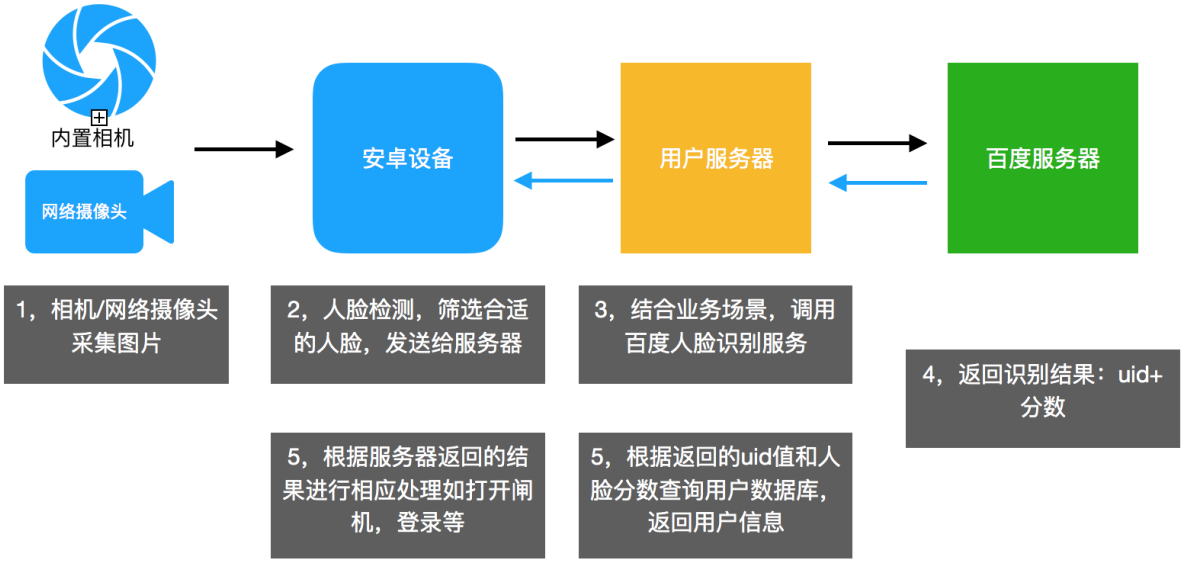
16 license 文件：授权文件，与 ak,sk,安卓包名和签名一一对应。集成的时候请放置在 assets 目录下。

17 签名 MD5：安卓应用签名 jks 文件的 MD5 值。

1.1 名词解释

前端设备：装有安卓系统的设备，如手机，平板，安卓开发板等。
内置相机：手机或平板自带的相机。
网络摄像机：和前端设备分开的，通过网络进行数据传输的摄像机。一般都是基于 RTSP 协议传输 H264 编码的视频。
AndroidSDK：百度人脸检测安卓 SDK;
用户后台服务：用户的后台服务器。
百度人脸服务：百度基于 REST 接口的对外人脸服务。

很多场景的数据流向为：



1. 前端设备通过内相相机或者网络摄像机采集照片/视频，进行解码。获取一帧帧的图片。
2. 调用 AndroidSDK 对图片内的内容进行分析，分析出是否无人脸，人脸在图片中的位置，人的角度等信息。再经过过虑算法，挑选一张质量较好的图片，对人脸区域进行裁剪。
3. 数据上传至用户后台服务器。
4. 用户后台服务接收到照片上传至百度人脸服务进行人脸识别查询。
5. 百度人脸服务返回结果给用户后台服务，用户后台服务进行相应的逻辑处理，再将结果返回给前端 app.

1.2 注意事项

为了跑通整体逻辑，方便理解，安卓 demo 中也集成了，后台服务器的一些功能。实际使用过程中，我们建议把人脸识别接口的功能放置于自己的服务器当中。主要是为了防止 ak,sk，被他人反编译调用，从而造成，经济和数据上的损失 ！！！！！！

1.3 硬件选型：

Android 闸机硬件选型推荐有两种方案，一种是使用现有市场上的成品平板，另外一种是使用定制的硬件平台和自选外设摄像头（USB 摄像头或网络摄像头）。针对三种方案分别做出如

1、平板电脑方案：

名称	要求
CPU	ARM v7(支持 neon 指令集)及以上架构 4 核以上
内存	2G+
屏幕	640*480 以上
摄像头	720p 以上
系统	Android 4.4+

型号推荐：华为 M3

2、USB 摄像头方案

名称	要求
CPU	ARM v7(支持 neon 指令集)及以上架构 4 核以上
内存	2G+
摄像头	USB2.0+ 支持 UVC 协议
系统	Android 4.4+

硬件型号推荐：Firefly-RK3288，Firefly-RK3399

摄像头配置标准：推荐 CMOS 使用 索尼 imx290 cmos

USB 方案：UVC 标准，镜头 3.5MM， 120W 像素以上，宽动态 100dB 以上。
推荐：迪威泰的 3088AC-P32A（自研闸机），施派尔-3220W（深大）
如何使用：将 USB 摄像头与开发板连接，SDK 可以自动识别摄像头。使用方法和内置相机一致。

3、网络摄像头方案。

名称	要求
开发板	rk3399(720p 15+帧，显示和检测异步) rk3288（720p 10+帧 显示和检测异步）
镜头	3.5mm （具体根据检测距离选择）
分辨率	120W 像素以上
宽动态	100dB 以上
快门	小于 1/500s
传输协议	RTSP

如何使用：网络摄像头与开发板连接到同一个局域网内（连接至同一路由器，并使用同一网段如192.168.1.xxx）
根据网络摄像头说明书，配置网络摄像头，**帧率 15，分辨率 720P,码率 12M 左右**，设置 rtsp 协议 url 和用户名和密码。调整焦距至画片清晰。可以用 VLC 等支持 RTSP 协议的软件验证摄像头是否配置成功。如摄像头配置没有问题，可参数 RTSTTestActivity，修改 摄像头地址。其它流程与自带摄像头一致。

2. 集成指南

2.1 准备工作

在正式集成前，需要做一些准备工作，完成一些账号、应用及配置，具体如下：

2.1.1 注册开发者

STEP1：点击百度 AI 开放平台导航右侧的控制台，页面跳转到百度云登录界面，登录完毕后，将会进入到百度云后台，点击「控制台」进入百度云控制台页面；您也可以在官网直接点击免费试用，登录完毕后将自动进入到百度云控制台。
STEP2：使用百度账号完成登录，如您还未持有百度账户，可以点击此处注册百度账户。
STEP3：进入百度云欢迎页面，填写企业/个人基本信息，注册完毕，至此成为开发者。注：(如您之前已经是百度云用户或百度开发者中心用户，STEP3 可略过。)
STEP4：进入百度云控制台，找到人工智能相关服务面板。
STEP5：**点击进入「人脸识别」模块。**

注一：如果您通过百度 AI 开放平台登录到后台，AI 相关服务模块入口，如下图所示：

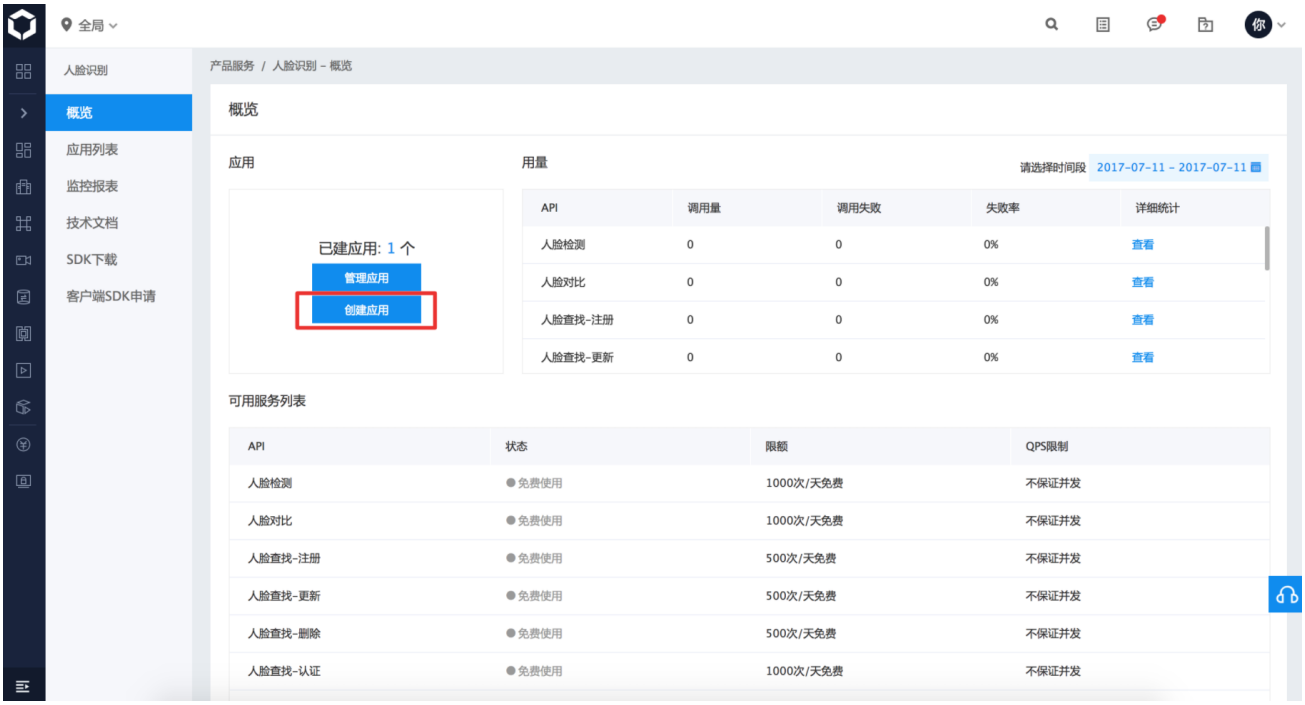
注二：如果您通过百度云直接登录后台，AI 相关服务模块入口，则如下图红框部分所示：

两种入口只是展现形式不同，相关 AI 服务模块内容完全一样。

2.1.2 创建应用

创建好账号后，在正式调用 AI 能力之前，需要您创建一下应用，这个是调用服务的基础能力单元

选择「人脸识别」服务，首先见到此服务的控制台概览页，如下图所示：



如上图所示，点击「创建应用」，即可进入应用创建界面，如下图所示：

2.1.3 获取密钥

	应用名称	AppID	API Key	Secret Key
1	身份证识别	9918631	GOS93OvBoudHeRegLaGpYXGr	******* 显示

在您创建完毕应用后，平台将会分配给您此应用的相关凭证，主要为 **AppID、API Key、Secret Key**，以上三个信息是您应用实际开发的主要凭证，每个应用之间各不相同，请您妥善保管。如上图所示。

注：开发中请注意区分多份 AKSK（API Key、Secret Key）

2.1.4 生成 token

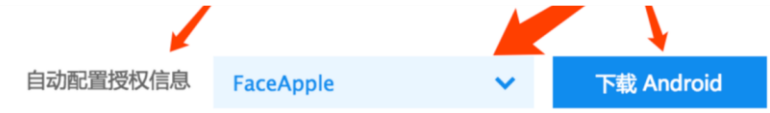
刚才所创建的应用在调用开放平台 API 之前，首先需要获取 Access Token（用户身份验证和授权的凭证）您需要使用创建应用所分配到的 AppID、API Key 及 Secret Key，进行 Access Token 的生成，方法详见 Access Token 获取，我们为您准备了几种常见语言的请求示例代码。

注：Access Token 的有效期为 30 天（以秒为单位），请您集成时注意在程序中定期请求新的 token，或每次请求都获取一次 token。

2.1.5 SDK license（重要重要）

人脸 SDK License:此 license 用于 SDK 离线功能使用,百度AI官网->控制台->产品服务->人脸识别->客户端SDK管理，页面下载两个端 的 license,用于接下来集成使用。

不过我们为您提供了自动化配置脚本,如下图所示:



您可以在下载核身示例工程的同时,直接下载已经配置好授权信息的版本。

2.2 安卓集成

2.2.1 自动配置授权信息集成

如果您是通过自动配置授权信息下载的示例工程,只需要谢欧冠 Config 类和配置签名。您的集成将少很多步骤。

1、修改 Config 类中参数,自动配置已经为您修改好了 licenseID 和 license 文件,您只需要修改 apiKey、secretKey(即 ak/sk), groupId 是自己定义的,用于人脸注册和人脸识别等接口使用。保证注册的人脸 和查找的人脸在同一个 groupId 即可。

```
public class Config {

    // 为了apiKey,secretKey为您调用百度人脸在线接口的，如注册，识别等。
    // 为了的安全，建议放在您的服务端，端把人脸传给服务器，在服务端端进行人脸注册、识别放在示例里面是为了您快速看到效果
    public static String apiKey = 替换为你的apiKey(ak);
    public static String secretKey = 替换为你的secretKey(sk);
    public static String licenseID = 替换为你的licenseID,后台SDK管理界面中,已经生成的licenseID,如:test-face-android;
    public static String licenseFileName = "idl-license.face-android";


    /**
        groupId，标识一组用户（由数字、字母、下划线组成），长度限制128B，可以自行定义，只有注册和识别都是同一个组。

        每个开发者账号只能创建一个人脸库；groupID用于标识人脸库
        每个人脸库下，用户组（group）数量没有限制；
        如闸机场景，可以给一个公司生成一个groupID;用于区分不同公司间的人脸库。
        详情见 http://ai.baidu.com/docs#/Face-API/top

        人脸识别 接口 https://aip.baidubce.com/rest/2.0/face/v2/identify
        人脸注册 接口 https://aip.baidubce.com/rest/2.0/face/v2/faceset/user/add
     */

    public static String groupID = 替换为groupID;

}
```

配置签名(申请 license 时的 md5 为打包签名的文件,所以必须用申请 license 的签名文件)

因为 SDK 运行时会对比 license 里面的 md5 和签名文件的 md5,为了能 debug 也能使用人脸,所以需 要进行下面的配置。实际发布时只要使用申请时关联的签名文件即可,没有下面的配置也行。

app->build.gradle->android->signingConfigs keyAlias 为你创建的打包签名文件的别名

```
signingConfigs {
    debug {
        keyAlias [redacted]
        keyPassword [redacted]
        storeFile file('../[redacted].jks')
        storePassword([redacted])
    }
    release {
        keyAlias [redacted]'
        keyPassword [redacted]'
        storeFile file('../[redacted].jks')
        storePassword([redacted])
    }
}
```

2.2.2 未使用自动配置授权信息集成

1、把申请的 license(idl-license.face-android")放到到项目中 assets 目录中

2、修改 Config 类中的参数 licenseID 为您申请 license 填的授权信息字符串+" -face-android"

groupId 是自己定义的,用于人脸注册和人脸识别等接口使用。保证注册的人脸和查找的人脸在同一个 groupId 即可

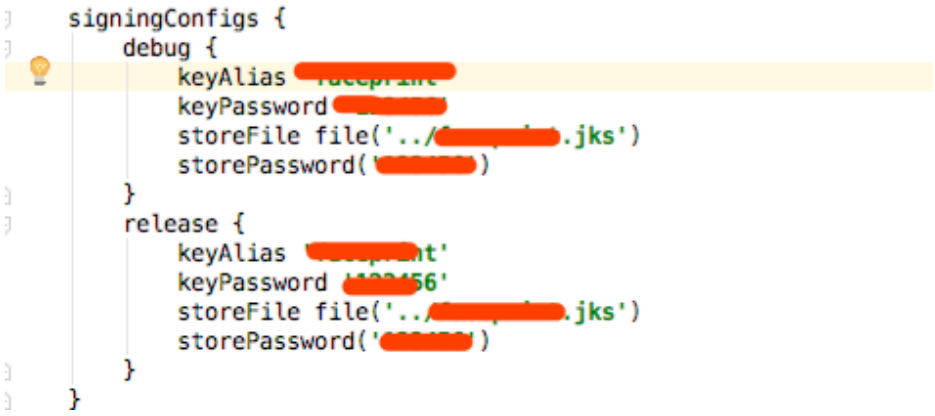
```
public class Config {

    // 为了apiKey,secretKey为您调用百度人脸在线接口的，如注册，识别等。
    // 为了的安全，建议放在您的服务端，端把人脸传给服务器，在服务端端进行人脸注册、识别放在示例里面是为了您快速看到效果
    public static String apiKey = 替换为你的apiKey(ak);
    public static String secretKey = 替换为你的secretKey(sk);
    public static String licenseID = 替换为你的licenseID,后台SDK管理界面中，已经生成的licenseID,如:test-face-android;
    public static String licenseFileName = "idl-license.face-android";


    /**
     groupId，标识一组用户（由数字、字母、下划线组成），长度限制128B，可以自行定义，只有注册和识别都是同一个组。

     每个开发者账号只能创建一个人脸库；groupID用于标识人脸库
     每个人脸库下，用户组（group）数量没有限制；
     如闸机场景，可以给一个公司生成一个groupID;用于区分不同公司间的人脸库。
     详情见 http://ai.baidu.com/docs#/Face-API/top

     人脸识别 接口 https://aip.baidubce.com/rest/2.0/face/v2/identify
     人脸注册 接口 https://aip.baidubce.com/rest/2.0/face/v2/faceset/user/add
     */
    public static String groupID = 替换为groupID;

}
```

3、配置签名(申请 license 时的 md5 为打包签名的文件,所以必须用申请 license 的签名文件)

因为 SDK 运行时会对比 license 里面的 md5 和签名文件的 md5,为了能 debug 也能使用人脸,所以 需要进行下面的配置。实际发布时只要使用申请时关联的签名文件即可,没有下面的配置也行。

app->build.gradle->android->signingConfigs keyAlias 为你创建的打包签名文件的别名 keyPassword 为你创建的打包签名的别名密码 storeFile 为签名文件的路径

storePassword 为签名文件的密码



4、修改包名 app->build.gradle->android->defaultConfig ->您申请 license 时填的包名

```
android {
    compileSdkVersion 25
    buildToolsVersion "25.0.3"
    defaultConfig {
        applicationId "[redacted]"
        minSdkVersion 19
        targetSdkVersion 25
        versionCode 1
        versionName "1.0"

        ndk{
            abiFilters 'armeabi-v7a'
        }

    }
```

```
manifest | application | activity
<?xml version="1.0" encoding="utf-8"?>
<!--
  ~ Copyright (C) 2017 Baidu, Inc. All Rights Reserved.
  -->
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="[redacted]">

    <uses-permission android:name="android.permission.CAMERA"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>

    <uses-feature android:name="android.hardware.camera"/>
    <uses-feature android:name="android.hardware.camera.autofocus"/>
```

2.3 SDK 使用

2.3.1 SDK 初始化

使用人脸检测之前先要对识别库进行初始化。推荐在 applicaton 的 onCreate 方法中调用如下代码片段

```
// 初始化人脸库
FaceDetector.init(this, Config.licenseID, Config.licenseFileName);
// 设置最小人脸，小于此值的人脸不会被识别
FaceDetector.getInstance().setMinFaceSize(100);
FaceDetector.getInstance().setCheckQuality(false);

// 头部的欧拉角，大于些值的不会被识别
FaceDetector.getInstance().setEulerAngleThreshold(45, 45, 45);
```

2.3.2 注册

1、 可以通过 RegDetectActivity 实时检测获取注册的人脸。注册过程是往人脸库中注册照片，因为该照片做为识别标准所以对质量的要求也更高。为了保证人脸注册图片的质量。

Demo 中提供了 RegDetectActivity.该 Activity 在屏幕上指定了一定的区域，要求用户在屏幕中央，脸正对着摄像头。如果用户的姿势达不到要求，会提示用户，直到符合要求为止。

```
private void init() {

    final CameraImageSource cameraImageSource = new CameraImageSource(this);
    // 从相机获取图片
    faceDetectManager.setImageSource(cameraImageSource);
    // 设置预览View
    cameraImageSource.setPreviewView(previewView);
    // 添加PreProcessor对图片进行裁剪。
    faceDetectManager.addPreProcessor(cropProcessor);

    rectView.getViewTreeObserver().addOnGlobalLayoutListener(() -> {
            rectView.getViewTreeObserver().removeOnGlobalLayoutListener(this);
            // 打开相机，映射View到相机之间的坐标
            start();
    });

    faceDetectManager.setOnFaceDetectListener((status, infos, frame) -> {
            handleFace(status, infos, frame.getArgb(), frame.getWidth());
    });

    // 安卓6.0+的系统相机权限是动态获取的，当没有该权限时会回调。
    cameraImageSource.getCameraControl().setPermissionCallback(() -> {
            ActivityCompat
                    .requestPermissions(RegDetectActivity.this, new String[] {Manifest.permission.CAMERA}, 100);
            return true;
    });

    boolean isPortrait = getResources().getConfiguration().orientation == Configuration.ORIENTATION_PORTRAIT;

    if (isPortrait) {
        previewView.setScaleType(PreviewView.ScaleType.FIT_WIDTH);
        cameraImageSource.getCameraControl().setDisplayOrientation(CameraView.ORIENTATION_PORTRAIT);
    } else {
        previewView.setScaleType(PreviewView.ScaleType.FIT_HEIGHT);
        cameraImageSource.getCameraControl().setDisplayOrientation(CameraView.ORIENTATION_HORIZONTAL);
    }
//      // 摄像头的类型，是否是usb摄像头，不设置检测的图片是倒立的，无法检测到人脸
//      cameraImageSource.getCameraControl().setUsbCamera(true);
//      // USB摄像头使用镜面翻转 。相机自带的前置摄像头自己加了镜面翻转处理
//      // 但其它摄像头，如USB摄像头，或者网络摄像头没有做这样的处理。该行代码可以实现预览时的镜面翻转。
//      previewView.getTextureView().setScaleX(-1);
}
```

usb摄像头注意

下面的代码片段是，检测人脸过程中，根据检测结果值，提示用户进行调整，从而得到高质量的照片。

```
private void handleFace(int retCode, FaceInfo[] faces, int[] argb, int width) {
    if (success) {
        return;
    }
    if (faces == null || faces.length == 0) {
        // 获取状态码
        int hint = LivenessDetector.getInstance().getHintCode(retCode, null);
        // 根据状态码获取相应的资源ID
        final int resId = LivenessDetector.getInstance().getTip(hint);
        displayTip(resId);
        lastTipResId = resId;
        return;
    }

    rectView.getGlobalVisibleRect(rect);
    // 屏幕显示坐标对应到，实际图片坐标。
    rectF.set(rect);
    previewView.mapToOriginalRect(rectF);
    LivenessDetector.getInstance().setDetectRect(rectF);

    // 获取状态码
    final int hint = LivenessDetector.getInstance().getHintCode(retCode, faces[0]);
    // 根据状态码获取相应的资源ID
    final int resId = LivenessDetector.getInstance().getTip(hint);

    final Bitmap bitmap = FaceCropper.getFace(argb, faces[0], width);

    handler.post(() -> {
            displayAvatar.setImageBitmap(bitmap);
            // 在主线程，显示。
            displayTip(resId);
            lastTipResId = resId;
    });
```

2、 或者相册获取注册的人脸。

```
// 从相机检测。
private void detect(final String filePath) {

    FileImageSource fileImageSource = new FileImageSource();
    fileImageSource.setFilePath(filePath);
    detectManager.setImageSource(fileImageSource);
    detectManager.setOnFaceDetectListener((status, faces, frame) -> {
            if (faces != null) {
                final Bitmap cropBitmap = FaceCropper.getFace(frame.getArgb(), faces[0], frame.getWidth());
                handler.post(() -> { avatarImageView.setImageBitmap(cropBitmap); });

                try {
                    File file = File.createTempFile(UUID.randomUUID().toString() + "", ".jpg");
                    // 压缩人脸图片至300 * 300，减少网络传输时间
                    ImageUtil.resize(cropBitmap, file, 300, 300);
                    RegisterActivity.this.faceImagePath = file.getAbsolutePath();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
    });
    detectManager.start();
}
```

3、 调用在线 api 进行注册。

```
        APIService.getInstance().reg(new OnResultListener<RegResult>() {
            @Override
            public void onResult(RegResult result) {
                Log.i("wtf", "orientation->" + result.getJsonRes());
                toast("注册成功! ");
                startResultActivity(1, username);
            }

            @Override
            public void onError(FaceError error) {
                toast("注册失败");
                startResultActivity(1, null);
            }
        }, file, uid, username);
//                  } else {
```

2.3.3 人脸识别

创建 FaceDetectManager 对象，该类封装了，人脸检测整体逻辑。
创建 CameraImageSource 用于，从系统相机获取图片。如果是网络相机，创建 RtspImageSource 从网络摄像头获取 图片进行识别。
FaceDetectManager 设置 previewView，进行预览显示。
FaceDetectManager 设置 OnFaceDetectListener,该回调在检测完一帧图片之后会被调用。

OnTrackListener 只有当检测到人脸的时候会被调用。
FaceDetectManager.getFaceFilter()会返回人脸条件过虑器，可以设置过虑角度。
过虑，符合条件的人脸图片上传至服务器进行人脸识别。（也就是 1:N 接口）
正常场景，80 以上可以认为可信。小于 80 的可以采取策略，选择重试。重度逻辑，请自行修改。

1、检测人脸（从 Android 手机自带摄像头和 usb 摄像头获取人脸）（DetectActivity）

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_detected);
    testView = (ImageView) findViewById(R.id.test_view);
    previewView = (PreviewView) findViewById(R.id.preview_view);
    textureView = (TextureView) findViewById(R.id.texture_view);

    // 从系统相机获取图片帧。
    final CameraImageSource cameraImageSource = new CameraImageSource(this);
    // 图片越小检测速度越快，闸机场景640 * 480 可以满足需求。实际预览值可能和该值不同。和相机所支持的预览尺寸有关。
    // 可以通过 camera.getParameters().getSupportedPreviewSizes() 获取支持列表。
    cameraImageSource.getCameraControl().setPreferredPreviewSize(640, 480);
    // 设置最小人脸，该值越大，检测性能越好。范围为80-200
    FaceDetector.getInstance().setMinFaceSize(80);
    FaceDetector.getInstance().setNumberOfThreads(4);
    // 设置预览
    cameraImageSource.setPreviewView(previewView);
    // 设置图片源
    faceDetectManager.setImageSource(cameraImageSource);
    faceDetectManager.getFaceFilter().setAngle(20);
    // 设置回调，回调人脸检测结果。
    faceDetectManager.setOnFaceDetectListener(new FaceDetectManager.OnFaceDetectListener() {
        @Override
        public void onDetectFace(int retCode, FaceInfo[] infos, ImageFrame frame) {
            final Bitmap bitmap =
                    Bitmap.createBitmap(frame.getArgb(), frame.getWidth(), frame.getHeight(), Bitmap.Config
                            .ARGB_8888);
            handler.post(new Runnable() {
                @Override
                public void run() {
                    testView.setImageBitmap(bitmap);
                }
            });
            if (infos == null) {
                // null表示，没有人脸。
                showFrame(null);
                shouldUpload = true;
            }
        }
    });
    // 人脸追踪回调。没有人脸时不会回调。
    faceDetectManager.setOnTrackListener((trackedModel) -> {
            showFrame(trackedModel);
            if (trackedModel.meetCriteria()) {
                // 该帧符合过虑标准，人脸质量较高。上传至服务器，进行识别
                upload(trackedModel);
            }
    });
```

```
        // 不需要屏幕自动变黑。
        textureView.setKeepScreenOn(true);

        boolean isPortrait = getResources().getConfiguration().orientation == Configuration.ORIENTATION_PORTRAIT;
        if (isPortrait) {
            previewView.setScaleType(PreviewView.ScaleType.FIT_WIDTH);
            // 相机竖屏模式
            cameraImageSource.getCameraControl().setDisplayOrientation(CameraView.ORIENTATION_PORTRAIT);
        } else {
            previewView.setScaleType(PreviewView.ScaleType.FIT_HEIGHT);
            // 相机横屏模式
            cameraImageSource.getCameraControl().setDisplayOrientation(CameraView.ORIENTATION_HORIZONTAL);
        }


//      // 摄像头的类型，是否是usb摄像头，不设置检测的图片是倒立的，无法检测到人脸
//      cameraImageSource.getCameraControl().setUsbCamera(true);
//      // USB摄像头使用镜面翻转 。相机自带的前置摄像头自己加了镜面翻转处理
//      // 但其它摄像头，如USB摄像头，或者网络摄像头没有做这样的处理。该行代码可以实现预览时的镜面翻转。
//      previewView.getTextureView().setScaleX(-1);
    }
```

2、从网络摄像头获取人脸（RtspTestActivity）

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_detected);
    previewView = (TexturePreviewView) findViewById(R.id.preview_view);
    textureView = (TextureView) findViewById(R.id.texture_view);

    // rtsp 的图像不是镜面的
    previewView.setMirrored(false);
    // 设置预览View用于预览
    rtspImageSource.setPreviewView(previewView);

    // rtsp网络摄像头地址。具体格式见，摄像头说明书。
    String url = String.format(Locale.ENGLISH,
            "rtsp://admin:Aa123456@%s:554/h264/ch1/main/av_stream", "192.168.1.64");
    rtspImageSource.setUrl(url);

    faceDetectManager.setImageSource(rtspImageSource);
    faceDetectManager.setOnFaceDetectListener((retCode, infos, frame) -> {
            showFrame(frame.getArgb(),infos, frame.getWidth(), frame.getHeight());
            if (retCode == 0) {
                upload(infos,frame.getArgb(),frame.getWidth());
            }
            if (infos == null) {
                shouldUpload = true;
            }
    });

    nameTextView = (TextView) findViewById(R.id.name_text_view);
    textureView.setOpaque(false);
    textureView.setKeepScreenOn(true);
}
```

根据网络摄像头的rtsp url修改

2、人脸比对（调用在线 api）

Demo 中为了方便演示逻辑，直接请求的百度服务器，实际场景可将图片发送给自己的服务器，再由服务器发起识别请求，或者，把 ak,sk 放置在服务器上。客户端调用接口获取 accessToken。保证账户信息安全。

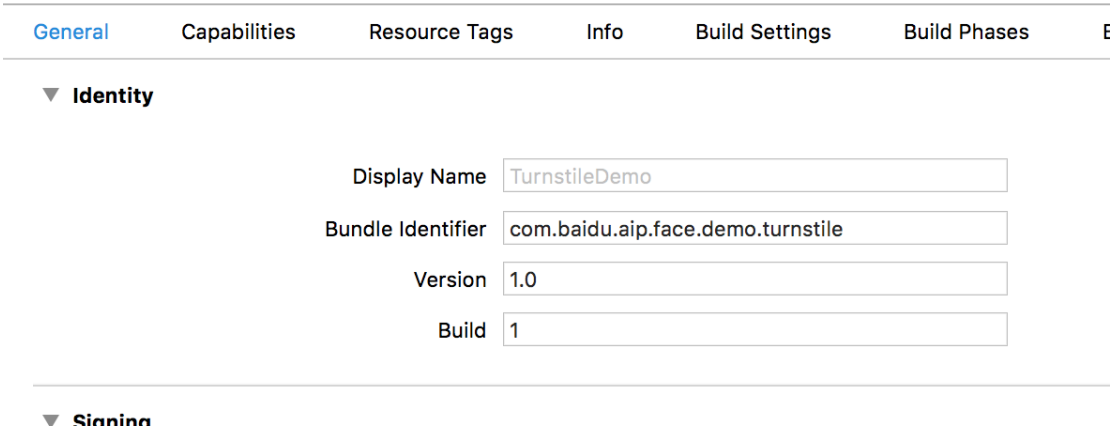
```
// 上传一帧至服务器进行，人脸识别。
private void upload(FaceFilter.TrackedModel model) {
    if (model.getEvent() != FaceFilter.Event.OnLeave) {
        if (!shouldUpload) {
            return;
        }
        shouldUpload = false;
        Bitmap face = model.cropFace();
        try {
            File file = File.createTempFile(UUID.randomUUID().toString() + "", ".jpg");
            // 人脸识别不需要整张图片。可以对人脸区别进行裁剪。减少流量消耗和，网络传输占用的时间消耗。
            ImageUtil.resize(face, file, 200, 200);
            APIService.getInstance().identify(new OnResultListener<FaceModel>() {
                @Override
                public void onResult(FaceModel result) {
                    if (result == null) {
                        return;
                    }
                    // 识别分数小于80，也可能是角度不好。可以选择重试。
                    if (result.getScore() < 80) {
                        shouldUpload = true;
                    }
                    showUserInfo(result);
                }

                @Override
                public void onError(FaceError error) {
                    error.printStackTrace();
                    shouldUpload = true;
                }
            }, file);
        } catch (IOException e) {
            e.printStackTrace();
        }
    } else {
        shouldUpload = true;
    }
}
```

2.3.4 相关类解释

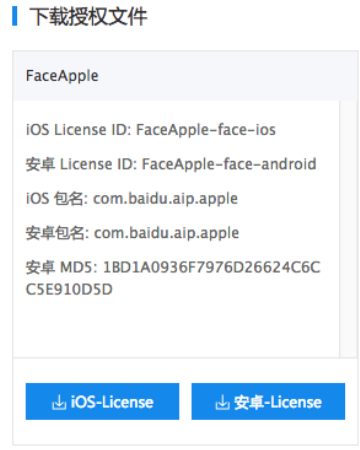
FaceDetector 封装了人脸检测相关底层的各种操作。
ImageFrame:封装了一帧帧的图片。
ImageSource:图片来源，如系统相机，图片文件，网络摄像头等。
FaceDetectManager:封装了图片采集到，检测的一整套逻辑。
PreviewView:显示相机的 View;
FaceCropper:工具类，用于裁剪人脸图片。

iOS 集成

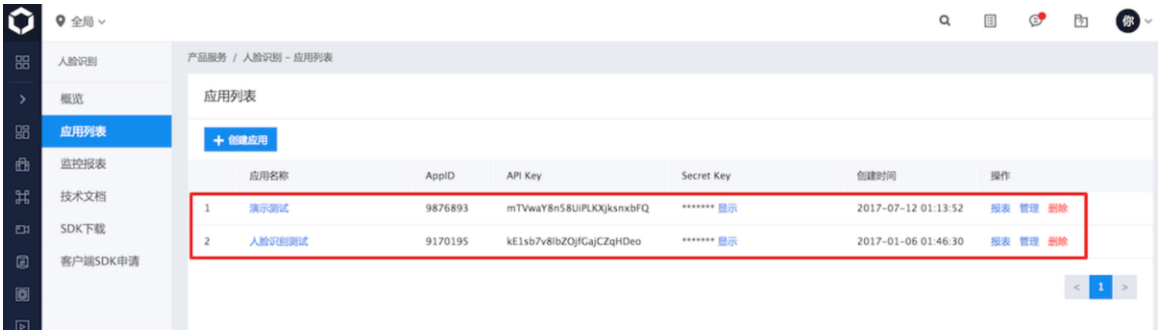
把申请的 idl_license.license 文件添加到工程当中。
1、确定工程的 BundleIdentifier 和申请时填写的 bundleID 相同



App 启动时创建 FaceSDK 实例，注意「APIKEY」,「APIKEY」**对应后台的 LicenseID，如下图示意：**



替换 Config.h 中的 APP_KEY 和 APP_SECRET（对应的后台应用列表中的 API-KEY 和 SECRET-KEY，）



```objc

#define APIKEY @"demo-turnstile-face-ios"

@interface AppDelegate ()

@end

@implementation AppDelegate

- (void)setupFaceSDK {
    NSString* licensePath = [[NSBundle mainBundle] pathForResource:@"idl_license" ofType:@"license"];
    [[FaceSDK sharedInstance]initWithWithAPIKey:APIKEY andLocalLicenceFile:licensePath];

    [[FaceVerifier sharedInstance] setAPIKey:APIKEY andLocalLicenceFile:licensePath];
    // 验证是否FaceSDK是否可用

    BOOL flag = [[FaceSDK sharedInstance] canWork];//
    NSLog(flag ? @"授权成功" : @"授权失败");

    flag = [[FaceVerifier sharedInstance] canWork];//

    NSLog(flag ? @"Yes" : @"No");

    NSString* appKey = APP_KEY;
    NSString* appSecret = APP_SECRECT;

    __weak ViewController* weakSelf = self;
    [[RestAPIService sharedInstance]getTokenWithAppKey:appKey andAppSecret:appSecret callback:^(id result, RequestEr
        //
        NSLog(@"result:%@",result);
        NSLog(@"error:%@",error);

        [SVProgressHUD dismiss];

        if (result) {
            weakSelf.registerButton.hidden = NO;
            weakSelf.detectButton.hidden = NO;
            weakSelf.getTokenButton.hidden = YES;
            NSString* accessToken = result[@"access_token"];
            NSLog(@"token:%@",accessToken);
            [RestAPIService sharedInstance].accessToken = accessToken;
        } else {
            weakSelf.getTokenButton.hidden = NO;
        }
    }];
}
```

设置 SDK 参数，正常情况使用 demo 中提供的参数即可。

```
    // 设置最小检测人脸阈值
    [[FaceVerifier sharedInstance] setMinFaceSize:100];
    // 设置截取人脸图片大小
    [[FaceVerifier sharedInstance] setCropFaceSizeWithWidth:100];
    // 设置人脸遮挡阀值
    [[FaceVerifier sharedInstance] setOccluThr:0.5];
    // 设置亮度阀值
    [[FaceVerifier sharedInstance] setIllumThr:40];
    // 设置图像模糊阀值
    [[FaceVerifier sharedInstance] setBlurThr:0.9];
    // 设置头部姿态角度
    [[FaceVerifier sharedInstance]setEulurAngleThrPitch:10 yaw:10 roll:10];
    // 设置是否进行人脸图片质量检测
    [[FaceVerifier sharedInstance]setIsCheckQuality:NO];
//    [FaceSDKManager.instance setIsCheckQualityWithIsCheck:YES];
    // 设置超时时间
//    [FaceVerifier sharedInstance] set
//    [FaceSDKManager.instance setConditionTimeoutWithTimeout:10];
    // 设置人脸检测精度阀值
    [[FaceVerifier sharedInstance] setNotFaceThr:0.6];
//    [FaceSDKManager.instance setNotFaceThresholdWithThr:0.6];
```

至此，项目配置完成。

注册和识别人脸检测和截取的代码类似，区别在于获取到人脸图片之后调用的 API 不同。注册相关代码说见 CaptureImageViewController

```
    self.cameraControl = [[CameraControl alloc]init];

    self.cameraControl.previewLayer.frame = self.view.bounds;
    self.cameraControl.previewLayer.masksToBounds = YES;
    self.cameraControl.previewDelegate = self;
    [self.view.layer addSublayer:self.cameraControl.previewLayer];
```

创建相机，设置 frame,将相机预览的 layer，添加到 View 的 layer 当中。用于显示相机预览图片。

相机预览过程中会回调- (void)cameraControl:(CameraControl *)camera
didOutputFrame:(UIImage *)image
覆写该方法，调用[[FaceVerifier sharedInstance] prepareDataWithImage:image
andActionType:FaceVerifierActionTypeVerify];对该图片进行人脸检测。检测结果
NSArray* faces = [[FaceVerifier sharedInstance] getTrackedFace];获取检测结果 。

```
- (void)handleFace:(TrackedFaceInfoModel*) face andImage:(UIImage*)image {
    if (!self.shouldHandle) {
        return;
    }

    CGImageRef croped = CGImageCreateWithImageInRect(image.CGImage, face.faceRect);
    UIImage* ui = [UIImage imageWithCGImage:croped];
    CGImageRelease(croped);
    UIImage* faceImage = [ui resizedImageToSize:CGSizeMake(140, 140)];

    self.currentImage = faceImage;

    self.shouldHandle = NO;
    __weak DetectViewController* weakSelf = self;
    [[RestAPIService sharedInstance] identifyWithImage:faceImage callback:^(id result, RequestError *error) {
        if (result) {
            NSLog(@"识别结果:%@",result);

            float score = [result[@"result"][0][@"scores"][0] floatValue];
            if (score > 80) {
                [weakSelf showResultViewController:result[@"result"][0]];
            } else {
                weakSelf.shouldHandle = YES;

            }
        } else {
            [weakSelf showResultViewController:nil];
            NSLog(@"识别错误:%@",error);
//            self.shouldHandle = YES;
        }
    }];
}

- (void)cameraControl:(CameraControl *)camera didOutputFrame:(UIImage *)image {
    //等待相机，否则照出来的图片很容易特别黑
    if ([[NSDate date] timeIntervalSince1970] - self.startTime < 1.5) {
        return;
    }

    [[FaceVerifier sharedInstance] prepareDataWithImage:image andActionType:FaceVerifierActionTypeVerify];
    NSArray* faces = [[FaceVerifier sharedInstance] getTrackedFace];

    [self.facePath removeAllPoints];
```

回调结果上传至人脸识别服务器进行识别。根据返回结果处理相应的业务逻辑。

Demo 中为了方便演示逻辑，直接请求的百度服务器，实际场景可将图片发送给自己的服务器，再由服务器发起识别请求，或者，把 ak,sk 放置在服务器上。客户端调用接口获取 accessToken。保证账户信息安全。

```
- (void)identifyWithImage:(UIImage *)image callback:(RestCallback)callback {
    NSString* groupId = USER_GROUP;

    NSData* data = UIImageJPEGRepresentation(image, 0.8f);
    NSString* imageBase64 = [data base64EncodedStringWithOptions:NSDataBase64EncodingEndLineWithCarriageReturn];
    id params = @{@"group_id" : groupId, @"image" : imageBase64,@"access_token":self.accessToken};
    [self postWithPath:@"identify" params:params andCallback:callback];
}
```

identify 方法比较简单，将 UIImage 转化为 base64 字符串，post 到服务器接口。

常见问题

Q：SDK license 和 ak/sk ?

A：license 是用于本地人脸检测，申请联系商务。ak/sk 用调用百度云服务人脸注册和人脸识别 。

Q：Android 申请时需要填入打包签名的 MD5,该 MD5 如何得来？
A：创建工程时生成签名文件 keystore.jks（该文件会用于最终发布是打包，请认真对待），在命令行工具中使用 keytool -list -v -keystore keystore.jks 得到 md5，删除冒号。

具体可以参考开发者论坛。关于安卓包名、MD5 值的一些讲解
https://ai.baidu.com/forum/topic/show/492039

Q：闸机场景选择多大分辨率？
A：对于闸机场景，分辨率没有必要特别高，通常 640*480 即可满足。分辨率越小，人脸检测的速度就越快。

Q：检测距离太近怎么办？
A：1、可以修改最小检测人脸（可调节范围 80-100），在相同的分辨率下，最小检测人脸越小，就能在更远的距离检测到人脸。当然最小检测人脸越小，越耗性能。

Q：怎么样更快检测到人脸？
A：mFaceTracker.set_detection_frame_interval(10); 设置该参数，参数越小，就会人脸进入屏幕时更快的检测到人脸，单位 ms