ZKLiveFace SDK 开发手册

（Android 版本）

目录

1. ZKLiveFace 算法描述

面部识别系统是从图像或视频中自动识别到人脸的系统。作为最早的生物识别技术之一，面部识别有许多优于其他生物识别技术的地方：与生俱来的，非侵入性的，且易于使用。如今，源于工业技术的快速发展（如数码摄像机，移动网络设备）和日益增强的安全需求，面部识别变得越来越重要，已经被广泛应用于各种系统中，包括考勤，安防，视频监控等。

ZKLiveFace 作为基于可见光光源面部识别算法，是一种快速、准确的 1:1 和 1:N 算法，面向软件开发商和系统集成商全面开放，可根据不同的市场和客户需求，提供定制不同的 SDK 版本；并且不同的 SDK 版本，在不同平台下的面部模板支持比对和识别具备一致性。

ZKLiveFace　SDK 技术说明：

图像

1）为获得高质量面部模板及速度，请根据实际应用场景设置探测距离。

2）面部登记和识别时推荐的最小成像分辨率是 640 x 480 像素。小于这个像素的分辨率也支持，但是会降低面部登记和识别的精准性，影响面部模板的质量。

3) 登记过程中使用多个图像，因为这种方式可以提高面部模板的质量，进而提升面部识别的质量和可靠性。

光照

应考虑可控的和不可控的光照条件。

需注意如下典型的状况：

正面直射光和漫射光对人脸的各个角度和整个人脸区域的阴影有同样的光分布。

某些类型的光照还会造成眼镜或人脸的反光。

面部姿势

面部识别算法可支持多种姿势：

1) 头倾斜±30 度

±25 度是正常的值，对于大多数接近正面的面部图像，这个倾斜度是足够的。

2）俯仰与正面位置偏离±30 度

±25 度是正常的值，如果在人脸登记过程中同一张人脸有几幅不同角度的低头图像，低头的允许公差可以增加到±30 度。

3）摇头与偏离正面位置±30 度

±15 度的偏离是正常的值，对于接近正面的面部图像，这个偏离度也是足够的。

为支持人脸正面±30 度的偏离角度，推荐登记同一人脸的不同图像到数据库中。

面部表情

面部算法最大限度支持一定不自然的面部表情下的人脸识别的准确性。不自然的表情例子如下（允许使用但不推荐使用）

大笑（露出牙齿或嘴里面的部分）

挑眉毛

闭眼

皱眉毛

眼镜，化妆，头发，胡须和小胡子

为保证人脸识别的质量，算法 SDK 支持部分面部被眼镜或头发遮挡的状况：

眼镜—普通的带框眼镜会降低人脸识别的质量，因为它们会遮挡部分面部，导致部分人脸特征不可见。

隐形眼镜—隐形眼镜不影响人脸识别效果。但是戴隐形眼镜的人有时也可能戴普通镜片眼镜。

浓妆可能隐藏或者扭曲面部特征

发型—有些发型可能遮挡部分面部

面部发型改变可能需要额外再登记面部，特别是胡须/小胡子长出来或剃掉的时候。

ZKLiveFace SDK 主要功能：

人脸检测

面部算法 SDK 提供快速、高准确率的人像检测功能。普遍适用于图片与实时视频流，可检出不小于 36*36 像素的人脸。

人脸关键点检测

面部算法 SDK 人脸关键点检测可以精确定位面部的关键区域位置，包括眼睛、尖下巴，脸部轮廓等。支持一定程度遮挡的人脸。

人脸属性分析

分析目标人脸的性别及年龄。

人脸活体检测

提供单帧照片活体检测接口，同时也可以基于人脸姿态估计进行动作配合式活体检测。

人脸识别

1:1 人脸验证：面部算法 SDK 人脸验证技术可被用于登录验证、身份识别等应用场景。帮助用户快速判定两张照片是否为同一个人、判定视频中的人脸是否为目标人脸并支持实时识别认证，还可以实现身份和人脸绑定等功能。

1:N 人脸识别：面部算法 SDK 人脸识别技术可以自动识别出照片、视频流中的人脸身份，识别速度和精度均居世界领先水平。

2. ZKLiveFace SDK 架构及安装

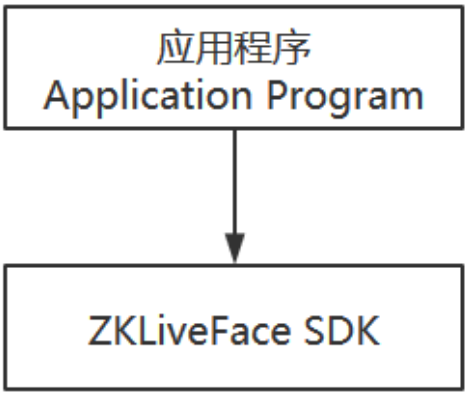
2.1 ZKLiveFace SDK 架构

ZKLiveFace SDK Android 版本主要以 java 接口的方式存在，使用者可以使用 Android 应用的开发语言（java）开发基于可见光人脸识别的应用程序。

Files Included

Operating System	Files	Description
	libs下所有文件	ZKLiveFace 算法库

SDK Architecture

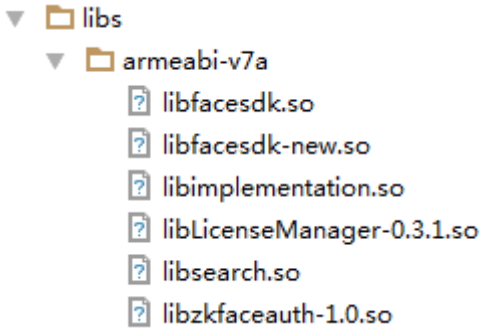


2.2 软件安装

在安装 ZKLiveFace SDK 之前，请确定您的操作系统或移动终端设备符合软件运行的要求。

将 ZKLiveFace 算法库打包到应用程序里面，用户使用不同的开发工具可能会有不同的打包方式。下面以 Android Studio IDE 开发环境为例简述 ZKLiveFace SDK 使用步骤。

1、将.so 及.jar 文件拷贝到 Android 项目的 libs 目录里面，so 库文件根据 CPU 架构
的不同保存在对应不同的目录。



2.3 许可申请及使用事项

本 SDK 使用设备硬件信息或者加密芯片绑定，以下对两种许可方式进行分别阐述。

2.3.1 加密芯片绑定

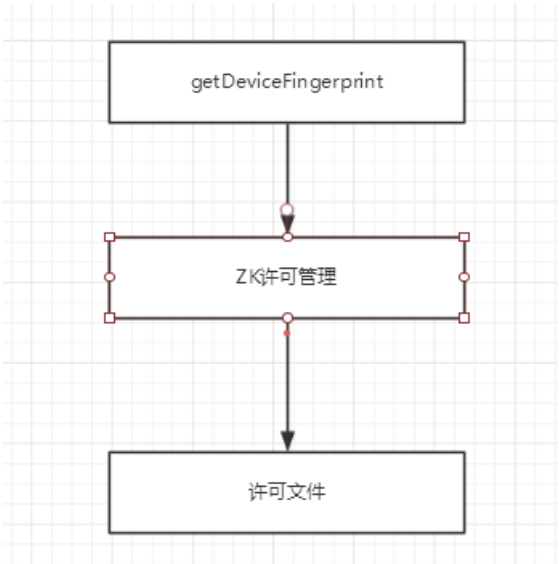
使用加密芯片绑定方式无需绑定其他设备硬件信息。拥有有效加密芯片授权的设备可以直接初始化算法(3.1.8)成功；拥有有效加密芯片授权的设备调用 isAuthorized（3.1.3）接口总是返回 true（拥有合法授权）。

2.3.2 设备硬件信息绑定

- 使用设备硬件信息绑定的方式，由于读取设备硬件信息，读写许可等需求，请至少在清单文件配置包含以下权限：

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

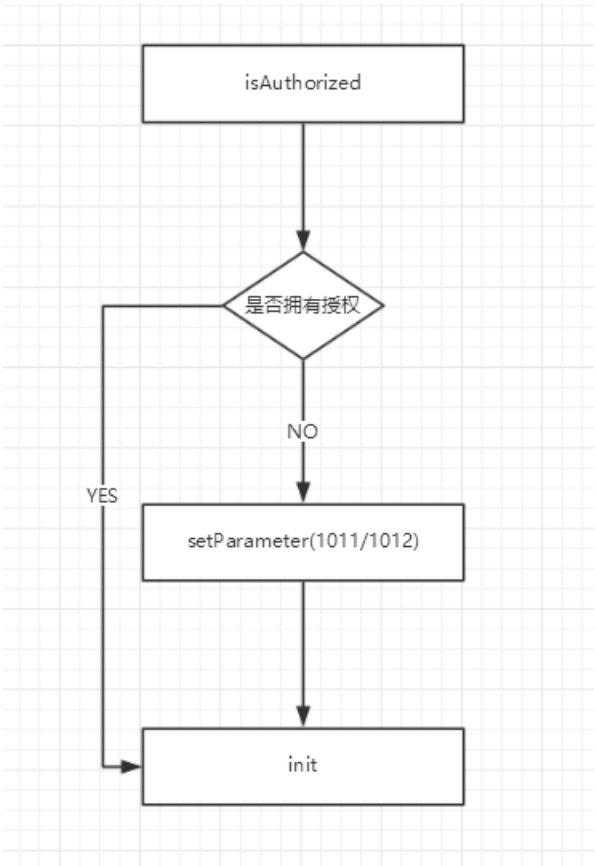
- 许可申请流程



  - 调用 getDeviceFingerprint（3.1.5）获取设备硬件信息

- 用获取到的设备硬件信息申请许可
- 取得许可文件

- 许可激活流程



- 调用 isAuthorized （3.1.3）判断是否已经拥有授权状态(拥有授权状态下可以直接调用 init（3.1.8）初始化)
- 未授权时调用 setParameter（3.1.6）通过 1011/1012(附录 2)参数设置许可文件名或许可数据到 SDK
- 调用 init（3.1.8）初始化算法(初始化算法完成最终整个激活过程)

3. ZKLiveFace SDK 接口类详述

3.1 接口说明

3.1.1 version

[函数]

```
public static int version(byte[] version, int[] size);
```

[功能]

获取版本号

[参数]

version[out]
返回版本号(建议预分配 128 字节以上)
size[in/out]
[in]:version 内存大小(字节数)
[out]:实际返回 version 长度

[返回值]
错误码(见附录 1)

[示例]

```
byte[] version = new byte[256];
int[] size = new int[1];
size[0] = 256;
if (0 == ZKLiveFaceService.version(version, size))
{
    String verStr = new String(version, 0, size[0]);
}
```

3.1.2 getLastError

[函数]

```
public static int getLastError(long context,byte[] lasterror, int[] size);
```

[功能]
返回最近一次的错误信息

[参数]
context[in]
算法实例指针(允许传 NULL)，当传的不为 NULL 时为该实例的最近一次错误(错误码为 11 时可调用该接口获取错误描述)
lasterror[out]
错误信息（建议预分配 256 字节，足够使用）
size[in/out]
[in]:version 内存大小(字节数)
[out]:实际返回 lasterror 长度

[返回值]
错误码(见附录 1)(该接口返回失败，一般错误原因是分配的内存不足)

[备注]
1、当使用的接口不需要传 context 算法实例指针时，调用该接口时 context 参数可以传 0，例如接口： init 等。这些接口在调用 getLastError 时，context 传 0。

[示例]

```
byte[] lasterror = new byte[256];
```

```
int[] size = new int[1];
size[0] = 256;
if (0 == ZKLiveFaceService.getLastError(context, lasterror, size))
{
        String errStr = new String(lasterror, 0, size[0]);
}
```

3.1.3 isAuthorized

[函数]

```
public static boolean isAuthorized();
```

[功能]

当前设备是否已授权

[参数]

[返回]

true 已授权

false 未授权

[备注]

拥有合法加密芯片的设备总是返回 true（已授权状态）

3.1.4 getHardwareId

[函数]

```
public static int getHardwareId(byte[] hwid, int[] size);
```

[功能]

获取机器码

[参数]

hwid[out]

返回机器码(建议分配 256 字节)

size[in/out]

[in]:hwid 内存大小(字节数)

[out]:实际返回 hwid 长度

[返回值]

错误码(见附录 1)

[示例]

```
byte[] hwid = new byte[256];
int[] size = new int[1];
size[0] = 256;
if (0 == ZKLiveFaceService.getHardwareId(hwid, size))
{
```

```
            String hwidStr = new String(hwid, 0, size[0]);
        }
```

[备注]

机器码与实际绑定硬件信息无关，这里仅作为协助用户关联机器和许可文件用。

3.1.5 getDeviceFingerprint

[函数]

```
public static int getDeviceFingerprint(byte[] devFp, int[] size);
```

[功能]

获取设备硬件信息

[参数]

devFp[out]

返回设备硬件信息(建议预分配 32*1024 字节)

size[in/out]

[in]:devFp 内存大小(字节数)

[out]:实际返回 devFp 长度

[返回值]

错误码(见附录 1)

[示例]

```
byte[] devFp = new byte[32*1024];
int[] size = new int[1];
size[0] = 32*1024;
if (0 == ZKLiveFaceService.getDeviceFingerprint(devFp, size))
{
    String devFpStr = new String(devFp, 0, size[0]);
}
```

[备注]

将 devFp 保存成文件或者其他形式发送给商务申请许可。

3.1.6 setParameter

[函数]

```
public static int setParameter(long context, int code, byte[] value, int size);
```

[功能]

设置参数

[参数]

context[in]

算法实例指针

code[in]

参数代码（见附录 2）

value[in]

参数值

size[in]

数据长度(字节数)

[返回值]

错误码(见附录 1)

[备注]

1、设置最大探测人脸数、1：1 比对阀值，参数值为纯数字字符串。例如设置的参数值为："68"。

3.1.7 getParameter

[函数]

public static int getParameter(long context, int code, byte[] value, int[] size);

[功能]

获取参数

[参数]

context[in]

算法实例指针

code[in]

参数代码(见附录 2)

value[out]

参数值

size[in/out]

[in]:value 分配数据长度

[out]:实际返回参数数据长度

[返回值]

错误码(见附录 1)

[备注]

1、获取 1：1 比对阀值，获取到的参数值为纯数字字符串。例如获取 1:1 阀值，返回的参数值为："76"。

3.1.8 init

[函数]

public static int init(long[] context);

[功能]

初始化算法库

[参数]

context[out]
返回算法实例指针(context[0])

[返回值]
错误码(见附录 1)

[示例]

```
long context[] = new long[1];
int ret = ZKFaceService.init(context);
if (0 == ret)
{
    System.out.print("Init succ, context=" + context[0]);
}
else
{
    System.out.print("Init failed, error code=" + ret);
}
```

[备注]
1、调用初始接口成功后可以调用 setParameter 来设置人脸探测识别相关参数，具体设置方法及相关参数说明可以参考 setParameter 接口说明。

3.1.9 terminate

[函数]

```
public static int terminate(long context);
```

[功能]
释放算法资源

[参数]
context[in]
算法实例指针

[返回值]
错误码(见附录 1)

3.1.10 detectFacesFormNV21

[函数]

```
public static int detectFacesFromNV21(long context, byte[] rawImage, int width, int height, int[] detectedFaces);
```

[功能]
探测人脸

[参数]
context[in]

算法实例指针

rawImage[in]

NV21 图像数据

width[in]

图像宽

height[in]

图像高

detectedFaces[out]

探测到人脸数(<=最大探测人脸数(默认最大人脸探测数为 1 个))

[返回]

错误码(见附录 1)

[备注]

由于视频流默认输出 NV21 格式，增加此接口方便使用

3.1.11 detectFacesFromBitmap

[函数]

```
public static int detectFacesFormBitmap(long context, Bitmap bitmap, int[] detectedFaces);
```

[功能]

探测人脸

[参数]

context[in]

算法实例指针

bitmap[in]

图像

detectedFaces[out]

探测到人脸数(<=最大探测人脸数(默认最大人脸探测数为 1 个))

[返回]

错误码(见附录 1)

[备注]

二代证照片可以通过 Bitmap 对象探测。

3.1.12 getFaceContext

[函数]

```
public static int getFaceContext(long context, int faceIdx, long[] faceContext);
```

[功能]

获取指定人脸实例指针

[参数]

context[in]

算法实例指针

faceIdx[in]

人脸索引(见 ZKFace_DetectFaces，0~[detectedFaces-1])

faceContext[out]

返回人脸实例指针

[返回值]

错误码(见附录 1)

3.1.13 getLiveness

[函数]

public static int getLiveness(long faceContext, int[] score);

[功能]

获取人脸活体分数

[参数]

faceContext[in]

人脸实例指针

score[out]

活体检测分数

[返回值]

错误码(见附录 1)

[备注]

活体检测推荐分数:75；<75 识别为假脸

3.1.14 getFacePose

[函数]

public static int getFacePose(long faceContext, float[] yaw,float[] pitch,float[] roll);

[功能]

获取人脸姿态(3 个角度)

[参数]

faceContext[in]

人脸实例指针

yaw[out]

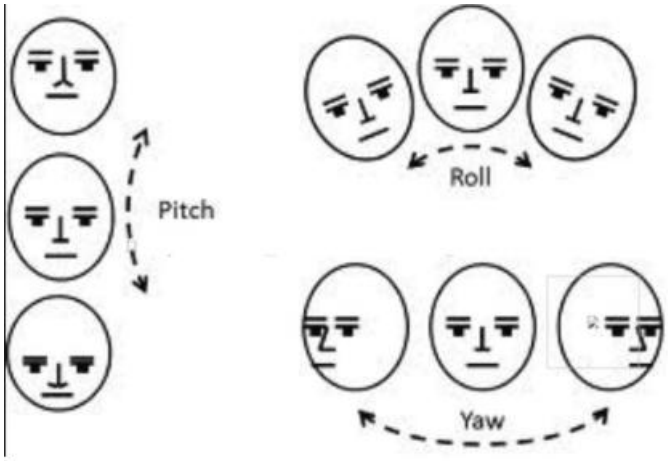
以鼻鼻子子为中心心的旋转角角度值，取值[-90, +90]之间，0 表示正脸，顺时针为正，逆时针 为负，角角度过大大时算法将无无法检测到人人脸，通常需要控制在 45° 之内。

pitch[out]

上下俯仰角角度值，取值[-90, +90]之间，0 表示正脸，抬头为正，低头为负，角角度过大大 时算法将无无法检测到人人脸，通常需要控制在 45° 之内。

roll[out]

左右偏角角角角度值，取值[-90, +90]之间，0 表示正脸，左转为正，右转为负，角角度过大大 时算法将无无法检测到人人脸，通常需要控制在 45° 之内。



[返回值]

错误码(见附录 1)

[备注]

3.1.15 getFaceICaoFeature

[函数]

public static int getFaceICaoFeature(long faceContext, int featureID, int[] score);

[功能]

获取 ICao 特征

[参数]

faceContext[in]

人脸实例指针

featureID[in]

特征 ID

score[out]

返回分数

[返回值]

错误码(见附录 1)

[备注]

1、ZKFACE_ICaoFEATUREID 见附录 3 说明

3.1.16 getFaceRect

[函数]

public static int getFaceRect(long faceContext, int[] points, int cntPx);

[功能]

获取探测到人脸的矩形框

[参数]

faceContext[in]

人脸实例指针

points[out]

矩形框四个坐标点 p0.x p0.y p1.x p1.y p2.x p2.y p3.x p3.y 顺序排列（顺时针方向）

cntPx[in]

points 数组大小(8)

[返回值]

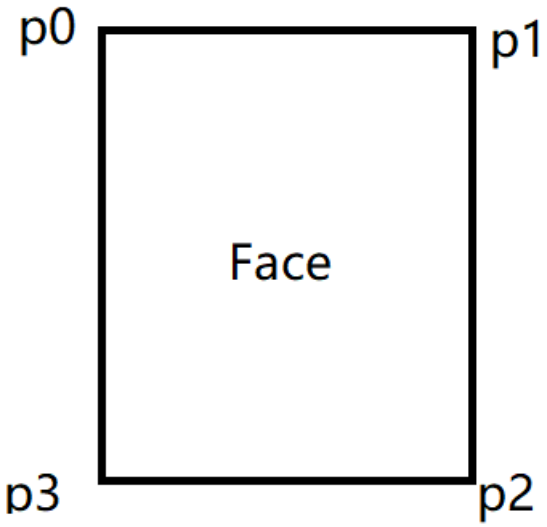
错误码(见附录 1)

[备注]

图片缩放旋转，按实际情况对对坐标转换

[示例]

坐标点示例



3.1.17 extractTemplate

[函数]

public static int extractTemplate(long faceContext, byte[] template, int[] size, int[] resverd);

[功能]

提取人脸模板

[参数]

faceContext[in]

人脸实例指针

temlpate[out]

人脸模板(建议至少分配 8*1024 字节)

size[in/out]

[in]:template 内存分配大小

[out]:实际返回 template 数据长度

resverd[out]

该参数为预留参数

[返回值]

错误码(见附录 1)

[示例]

```
int ret = 0;
byte[] template = new byte[8*1024];
int[] size = new int[1];
int[] resverd = new int[1];
size[0] = 8*1024;
ret = ZKLiveFaceService.extractTemplate(faceContext, template, size, resverd);
```

…

3.1.18 closeFaceContext

[函数]

public static int closeFaceContext(long faceContext);

[功能]

释放人脸实例对象

[参数]

faceContext[in]

人脸实例指针

[返回值]

错误码(见附录 1)

3.1.19 verify

[函数]

public static int verify(long context, byte[] regTemplate, byte[] verTemplate, int[] score)

[功能]

人脸 1:1 比对

[参数]

context[in]

算法实例指针

regTemplate[in]

登记模板

verTemplate[in]

比对模板

score[out]

返回分数

[返回值]

错误码(见附录 1)

[备注]

1、1:1 的比对阀值为 65。超过即为比对成功。

2、比对分数范围：0~100，详见附录 4

[示例]

```
int ret = 0;
int[] score = new int[1];
ret = ZKLiveFaceService.verify(context, regTemplate, verTemplate, score);
```

3.1.20 dbAdd

[函数]

public static int dbAdd(long context, String faceID, byte[] regTemplate)

[功能]

添加人脸模板到 1:N 缓存

[参数]

context[in]

算法实例指针

faceID[in]

人脸 ID

regTemplate[in]

登记模板

[返回值]

错误码(见附录 1)

[备注]

非线程安全接口(注意内存 db 读写保护)

人脸容量，建议 5000~10000 个人脸 ID

3.1.21 dbDel

[函数]

public static int dbDel(long context, String faceID)

[功能]

添加人脸模板到默认的 1:N 高速缓冲区

[参数]

context[in]

算法实例指针

faceID[in]

人脸 ID

[返回值]

错误码(见附录 1)

[备注]

非线程安全接口(注意内存 db 读写保护)

3.1.22 dbClear

[函数]

public static int dbClear(long context)

[功能]

清空默认的 1:N 高速缓冲区

[参数]

context[in]

算法实例指针

[返回值]

错误码(见附录 1)

[备注]

非线程安全接口(注意内存 db 读写保护)

3.1.23 dbCount

[函数]

public static int dbCount(long context, int[] count)

[功能]

获取默认的 1:N 高速缓冲区模板数

[参数]

context[in]

算法实例指针

count[out]

返回的模板个数

[返回值]

错误码(见附录 1)

[备注]

非线程安全接口(注意内存 db 读写保护)

3.1.24 dbIdentify

[函数]

public static int dbIdentify(long context, byte[] verTemplate, byte[] faceIDs, int[] score, int[] maxRetCount, int minScore, int maxScore)

[功能]

取默认的 1:N 高速缓冲区 1:N 识别

[参数]

context[in]

算法实例指针

verTemplate[in]

比对模板

faceID[out]

返回人脸 ID

score[out]

返回比对分数

maxRetCount[in/out]

[in]:最大返回多少个

[out]:实际返回多少个

minScore[in]

最低匹配分数。只有要识别的人脸与数据库中的某一人脸模板的相似度达到该值时，才能识别成功

maxScore[in]

当要识别人脸与数据库中的某一人脸模板的相似度达到该值时，识别成功立即返回

[返回值]

错误码(见附录 1)

[示例]

```
int ret = 0;
int[] score = new int[1];
byte[] faceIDS = new byte[256];
int[] maxRetCount = new int[1];
maxRetCount[0] = 1; //只返回 1 个人脸 ID
ret = ZKLiveFaceService.dbIdentify(context, verTemplate, faceIDS, score, maxRetCount, 76, 100);
...
```

[备注]

1、非线程安全接口(注意内存 db 读写保护)

2、比对分数范围：0~100, 附录 4

3、minScore 和 maxScore 参数详细说明：算法对加载到内存数据库的所有模板进行循环比对,在循环比对过程中,当识别人脸与内存数据库中的某一人脸模板相似度达到 maxScore 时,表示识别成功,立即退出循环比对;识别人脸与内存数据库中的某一人脸模板相似度达到 minScore 时,保存当前人脸 ID 号码,继续进行循环比对,直到循环完成所有模板比对后,按照相似度由高到低,返回参数 maxRetCount 设置的所有人脸 ID。

4. 工作流程说明

4.1 算法授权及流程

4.1.1 授权流程

见 2.3。

4.1.2 重要事项　恢复出厂设置/重新烧写系统

使用设备硬件信息授权的设备切记恢复出厂设置请勿勾选格式化 SD 卡，避免许可丢失。许可丢失需要重新申请。

同样，如果重新烧写系统等操作导致许可文件丢失也是需要重新申请授权。

重新申请授权时设备硬件信息授权需要重新获取。

4.2 算法初始化

4.2.1 初始化接口说明

//初始化人脸识别引擎，若成功则返回零。在调用其他函数之前必须先调用该函数，接口详细说明请见目录 3.1.8 对应的接口说明。

```
public static int init(long[] context);
```

4.2.2 初始化示例程序说明

[示例]

```
long context[] = new long[1];
int ret = ZKFaceService.init(context);
```

```
 if (0 == ret)
{
    System.out.print("Init succ, context=" + context[0]);
}
else
{
    System.out.print("isAuthorized=" + ZKFaceService.isAuthorized());
    System.out.print("Init failed, error code=" + ret);
}
```

4.3 探测人脸

通过 Bitmap/NV21 数据探测人脸个数，成功则返回零，并判断 detectedFaces[0]是否大于 1。

- 调用探测人脸接口
  见 detectFacesFromNV21/detectFacesFromBitmap(3.1.10/3.1.11)函数说明
- 获取人脸实例
  见 getFaceContext（3.1.15）

4.4 提取模板

- 调用[4.3] 探测人脸，获取到人脸实例指针
- 调用 extractTemplate（3.1.17）提取人脸特征。

4.5 登记人脸

4.5.1 人脸登记接口说明

1、[4.2]探测人脸数成功后,即可获取单个人脸的实例指针,并根据该人脸实例指针获

取提取人脸模板。提取成功后，即可登记。

//获取单个人脸实例指针接口，接口详细说明请见目录 3.1.15 对应的接口说明。

public static int getFaceContext(long context, int faceIdx, long[] faceContext);

//提取人脸模板，接口详细说明请见目录 3.1.17 对应的接口说明。

public static int extractTemplate(long faceContext, byte[] template, int[] size, int[] resverd);

2、如果需要进行 1：N 比对则需要将获取到的人脸模板登记到缓存中。接口详细说明请见目录 3.1.20 对应的接口说明。

public static int dbAdd(long context, String faceID, byte[] regTemplate);

4.5.2 人脸登记示例程序说明

1、[4.2]探测人脸数成功后,获取单个人脸的实例指针并提取人脸模板。

[示例]

```
//获取单个人脸实例指针（该示例人脸索引为：0），instanceContext 为算法初始化成功后的实例指针
//人脸索引的范围见：detectFaces，0~detectedFaces-1
long[] faceContext = new long[1];
retCode = ZKLiveFaceService.getFaceContext(instanceContext, 0, faceContext);
if(0 == retCode )
{
//提取人脸模板（人脸索引为：0 的人脸模板）,建议预分配 8192 个字节存放人脸模板。
byte[] template = new byte[8192];
size = new int[1];
size[0] = 8192;
int[] resverd = new int[1];
// faceContext[0]为之前获取成功后的人脸实例指针,获取成功后即可登记人脸
```

模板，

retCode = ZKLiveFaceService.extractTemplate(faceContext[0], template, size, resverd);

……

}

2、如果需要进行 1：N 比对,则需要将获取到的人脸模板登记到 1：N 缓存中。

[示例]

// instanceContext 为算法初始化成功后的实例指针, template 为要添加到 1：N 缓存的人脸模板。"Reg1"为要登记到 1：N 缓存的用户 ID 号。

retCode = ZKLiveFaceService.dbAdd(instanceContext, "Reg1", template);

3、备份登记照片

推荐客户在登记人脸时保存登记照片，算法模型升级时需要重新提取特征。

4.6 比对人脸

4.6.1 人脸比对程序说明

1、1:1 人脸比对

//接口详细说明请见目录 3.1.19 对应的接口说明。

public static int verify(long context, byte[] regTemplate, byte[] verTemplate, int[] score)

2、1:N 人脸比对

//接口详细说明请见目录 3.1.24 对应的接口说明。

public static int dbIdentify(long context, byte[] verTemplate, byte[] faceIDs, int[] score, int[] maxRetCount, int minScore, int maxScore)

4.6.2 人脸比对示例程序说明

1、1：1 人脸比对

int score = 0;

// instanceContext 为算法初始化成功后的实例指针, regTemplate、verTemplate 分别为要比对的人脸模板。

int ret = 0;

//存放返回的比对分数

int[] score = new int[1];

ret = ZKLiveFaceService.verify(instanceContext, regTemplate, verTemplate, score);

2、1：N 人脸比对(返回单个人脸 ID)

int ret = 0;

int[] score = new int[1]; // 存放返回的比对分数

byte[] faceIDS = new byte[256]; //存放返回的人脸 ID 号

int[] maxRetCount = new int[1];

maxRetCount[0] = 1; //只返回 1 个人脸 ID

// instanceContext 为算法初始化成功后的实例指针, verTemplate 为要比对的人脸模板

ret = ZKLiveFaceService.dbIdentify(instanceContext, verTemplate, faceIDS, score, maxRetCount, 76, 100);

[备注]

minScore 和 maxScore 参数详细说明：算法对加载到内存数据库的所有模板进行循环比对,在循环比对过程中,当识别人脸与内存数据库中的某一人脸模板相似度达到 maxScore 时,表示识别成功,立即退出循环比对;识别人脸与内存数据库中的某一人脸模板相似度达到 minScore 时,保存当前人脸 ID 号码,继续进行循环比对,直到循环完成所有模板比对后,按照相似度由高到低,返回参数 maxRetCount 设置的所有人脸 ID。

3、1：N 人脸比对（返回多个人脸 ID）

int ret = 0;

int[] maxRetCount = new int[1];

maxRetCount[0] = 6; //返回 6 个人脸 ID

int[] score = new int[maxRetCount[0]];//存放返回人脸 ID 对应的比对分数

byte[] faceIDS = new byte[4096];//人脸 ID 多条记录以\t 分隔

// instanceContext 为算法初始化成功后的实例指针, verTemplate 为要比对的人脸模板

ret = ZKLiveFaceService.dbIdentify(context, verTemplate, faceIDS, score, maxRetCount, 76, 100);

[备注]

minScore 和 maxScore 参数详细说明：算法对加载到内存数据库的所有模板进行循环比对,在循环比对过程中,当识别人脸与内存数据库中的某一人脸模板相似度达到 maxScore 时,表示识别成功,立即退出循环比对;识别人脸与内存数据库中的某一人脸模板相似度达到 minScore 时,保存当前人脸 ID 号码,继续进行循环比对,直到循环完成所有模板比对后,按照相似度由高到低,返回参数 maxRetCount 设置的所有人脸 ID。

4.7 活体识别

[4.2]探测人脸数成功后,获取单个人脸的实例指针调用活体识别接口，判断是否假脸。

[示例]

```
//获取单个人脸实例指针（该示例人脸索引为：0），instanceContext 为算法初始化成功后的实例指针
//人脸索引的范围见：detectFaces，0~detectedFaces-1
long[] faceContext = new long[1];
retCode = ZKLiveFaceService.getFaceContext(instanceContext, 0, faceContext);
if(0 == retCode )
{
    int[] score = new int[1];
    retCode = ZKLiveFaceService.getLiveness(faceContext[0], score);
```

```
        if (0 == retCode && score[0] < 75)
        {
            //疑似假脸
        }
    }
```

4.8 程序结束

4.8.1 结束程序说明

//释放人脸识别引擎。接口详细说明请见目录 3.1.8 对应的接口说明。

```
public static int terminate(long context);
```

[备注]

如果用使用到 1:N 对应的接口，在调用接口 terminate 前应先调用接口 dbClear 清空 1：N 缓存。

4.8.2 结束示例程序说明

//清空 1:N 缓存(有使用 1:对应的接口)，instanceContext 为算法初始化成功后的实例指针。

```
ret = ZKLiveFaceService.dbClear(instanceContext);
//释放资源，instanceContext[0]为算法初始化成功后的实例指针。
ret = ZKLiveFaceService.terminate(instanceContext);
```

5. 常见问题

5.1 许可授权

请获取设备硬件信息，并发送商务申请许可文件。

详细见 4.1

6. 附录

附录 1

错误码如下表所示

错误码	说明
-1	未知错误
0	成功
1	分配的内存不足
2	参数出错
3	分配内存失败
4	无效句柄
5	无效参数代码
6	获取眼间距出错
7	人脸索引号无效
8	比对分数过低
9	实际人脸的模板长度大于预分配的人脸模板长度
10	接口不支持
11	其它错误
12	无效人脸 ID 号
13	1:N 比对失败，未找到对应的人脸模板
14	加载动态库失败
15	图像类型参数错误

错误码	说明
16	超过 1:N 的最大容量
17	实际人脸缩略图长度大于预分配的人脸缩略图长度

［备注］

返回当错误码 11 时，可以调用 ZKLiveFace_GetLastError 接口来获取错误信息。

附录 2

参数代码说明如下:

参数代码	类型	说明
ZKLIVEFACE_PARAMETER_SET_MIN_EYE_DIST(1005)	string	设置双眼间距，默认间距 60
ZKLIVEFACE_PARAMETER_SET_THRESHOLD_IFCAE_VERIFY(1009)	string	设置(获取)1:1 阀值
ZKLIVEFACE_PARAMETER_GLOBAL_LICENSE_FILENAME(1011)	string	设置许可文件路径
ZKLIVEFACE_PARAMETER_GLOBAL_LICENSE_DATA(1012)	unsigned char*	设置许可文件数据

［备注］

设置(获取)1:1 比对阀值，设置(获取)的参数值为：纯数字字符串。

附录 3

ICAO 特征数据说明

特征代码	说明
ZKFACE_ICAO_FEATURE_ID_AGE(0)	评估年龄

特征代码	说明
ZKFACE_ICAO_FEATURE_ID_GENDER(1)	评估性别
ZKFACE_ICAO_FEATURE_QUALITY(2)	评估面部的质量

[备注]

1、当获取人脸质量时，score 为返回的质量分 0~100。

2、score 返回 1 表示男性，返回 0 表示女性。

附录 4

阈值说明

- 1:1 推荐阈值 65
  主要应用于图像质量比较模糊的场景，比如身份证照片等证件照
- 1:N 推荐阈值 76
  低质量图像不适用于 1:N
- 实际应用可以根据需要调整阈值