金融级身份认证

人脸验证接入

产品文档

腾讯云

文档目录

验证结果

方式一：前端获取结果验证签名

方式二：服务端查询结果

人脸认证多张照片查询接口

人脸验证接入
SDK 接入
生成签名

最近更新时间：2018-07-27 16:05:51

准备步骤

- **前置条件：请合作方确保 NONCE ticket 已经正常获取，获取方式见 NONCE ticket 获取 。**
- 作方为人脸验证服务生成签名，需要具有以下参数：
- 签名的数据需要和使用该签名的 SDK 中的请求参数保持一致。

参数名	说明	来源
appId	腾讯云线下对接分配的 App ID	腾讯云线下对接分配
userId	用户唯一标识	合作方自行分配（和 SDK 里面定义的 userId 保持一致）
version	参数值为：1.0.0	
ticket	合作伙伴服务端缓存的 ticket，注意是 NONCE 类型	获取方式见 NONCE ticket 获取（所用的 userId 参数值需要和 SDK 里面定义 userId 值保持一致）
nonceStr	必须是 32 位随机数	合作方自行生成（和 SDK 里面定义的随机数保持一致）

基本步骤

1. 生成一个 32 位的随机字符串 nonceStr（其为字母和数字，登录时也要用到）。
2. 将 appId、userId、version 连同 ticket、nonceStr 共五个参数的值进行字典序排序。
3. 将排序后的所有参数字符串拼接成一个字符串。
4. 将排序后的字符串进行 SHA1 编码，编码后的 40 位字符串作为签名（sign）。

> **注意：**
>
> 签名算法可参考 签名算法说明 。

参考示例

请求参数：

参数名	参数值
appId	TIDA0001
userId	userID19959248596551
nonceStr	kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7T
version	1.0.0
ticket	XO99Qfxlti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS

字典排序后的参数为：

[1.0.0, TIDA0001, XO99Qfxlti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS ， kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7T, userID19959248596551]

拼接后的字符串为：

1.0.0TIDA0001XO99Qfxlti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMSkHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7TuserID19959248596551

计算 SHA1 得到签名：

4AE72E6FBC2E9E1282922B013D1B4C2CBD38C4BD

该字串就是最终生成的签名（40 位），不区分大小写。

下一步：

Android 动作活体 & 数字活体 + 人脸比对 SDK 接入
Android 光线活体 + 人脸比对 SDK 接入
iOS SDK 接入

Android 动作/数字活体 + 人脸比对 开发准备

最近更新时间：2018-02-01 18:13:12

权限检测

SDK 需要用到相机/录音/读取手机信息权限，在 Android 6.0 以上系统，SDK 对其做了权限的运行时检测。但是由于 Android 6.0 以下系统 Android 并没有运行时权限，检测权限只能靠开关相机/麦克风进行。考虑到 SDK 的使用时间很短，快速频繁开关相机/麦克风可能会导致手机抛出异常，故 SDK 内对 Android 6.0 以下手机没有做权限的检测。为了进一步提高用户体验，在 Android 6.0 以下系统上，**我们建议合作方在拉起 SDK 前，帮助 SDK 做相机/麦克风/读取手机信息权限检测，提示用户确认打开了这三项权限后再进行人脸验证**，可以使整个人脸验证体验更快更好。

CPU 平台设置

目前SDK只支持 armeabi-v7a 平台，为了防止在其他 CPU 平台上 SDK crash，我们建议在您的 App 的 build.gradle 里加上 abiFilter，如下图中红框所示（注意：有且只有 armeabi-v7a 平台）：

```
defaultConfig {
    applicationId "com.webank.testcloud.cloudfacetest"
    minSdkVersion 14
    targetSdkVersion 23
    versionCode 1
    versionName "1.0"

    ndk {
        // 设置支持的SO库架构
        abiFilters 'armeabi-v7a'
    }
}
```

下一步：配置流程

配置流程

最近更新时间：2018-11-01 11:52:34

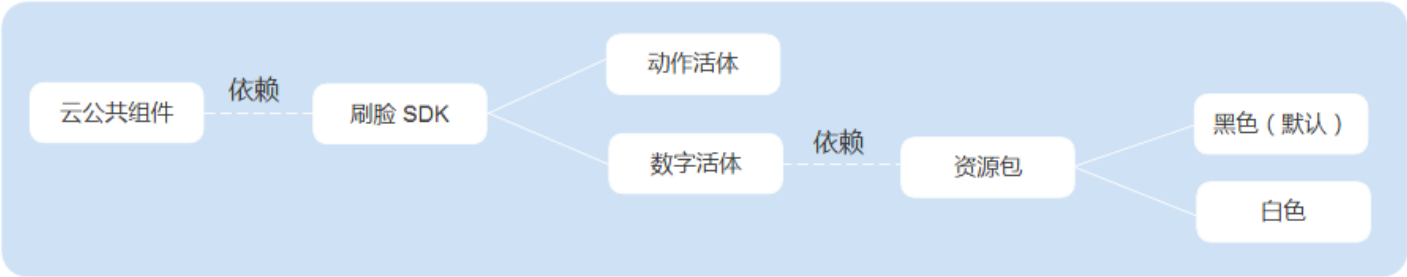
> **注意：**
> 以下文章出现 “刷脸” 一词均可以表示为 “人脸验证”。

1. 接入配置

** 云刷脸 SDK（WbCloudFaceVerify）最低支持到 Android API 14: Android 4.0（ICS）**，请在构建项目时注意。
刷脸 SDK 将以 AAR 文件的形式提供，即 WbCloudFaceVerifySdk，其中数字活体模式需要配合资源包（WbCloudFaceRes）一起使用。

- 刷脸 SDK 分为动作活体和数字活体两个模式。
- 数字活体的资源包分为黑色皮肤和白色皮肤（若要对 SDK 皮肤进行设定，除了接入对应的 AAR，还需要设定相关代码。皮肤的设定是无需格外设置的，默认为黑色皮肤，详情请参见 SDK 样式选择），接入方可自由选择组合配置。

下图为刷脸 SDK 基本构成：



另外刷脸 SDK 同时需要依赖 **云公共组件（WbCloudNormal）**，同样也是以 AAR 文件的形式提供。
为刷脸 SDK 添加依赖的方式如下：
将提供的 AAR 文件加入到 App 工程的 libs 文件夹下面，并且在** build.gradle **中添加下面的配置：

```
android{
//...
repositories {
flatDir {
dirs 'libs' //this way we can find the .aar file in libs folder
}
}
}
//添加依赖
```

```
dependencies {
//0. appcompat-v7
compile 'com.android.support:appcompat-v7:23.0.1'
//1. 云刷脸SDK
compile(name: 'WbCloudFaceVerifySdk', ext: 'aar')
//2. 云normal SDK
compile(name: 'WbCloudNormal', ext: 'aar')
//3. 云刷脸皮肤资源包-可选择黑色/白色 默认黑色（仅数字活体需要设置）
compile(name: 'WbCloudFaceResBlack', ext: 'aar')
//compile(name: 'WbCloudFaceResWhite', ext: 'aar')
}
}
```

2. 混淆配置

云刷脸产品的混淆规则分为三部分，分别是云刷脸 SDK（动作活体/数字活体）的混淆规则、云公共组件的混淆规则以及依赖的第三方库混淆规则。

云刷脸 SDK（动作活体）的混淆规则

```
################云刷脸动作活体混淆规则 faceverify-BEGIN###################
#不混淆内部类
-keepattributes InnerClasses
-keep public class com.webank.faceaction.tools.WbCloudFaceVerifySdk{
public <methods>;
public static final *;
}
-keep public class com.webank.faceaction.tools.WbCloudFaceVerifySdk$*{
*;
}
-keep public class com.webank.faceaction.tools.ErrorCode{
*;
}
-keep public class com.webank.faceaction.ui.FaceVerifyStatus{

}
-keep public class com.webank.faceaction.ui.FaceVerifyStatus$Mode{
*;
}
-keep public class com.webank.faceaction.tools.IdentifyCardValidate{
public <methods>;
}
-keep public class com.tencent.youtulivecheck.**{
*;
```

```
}
-keep public class com.webank.faceaction.Request.*$*{
*;
}
-keep public class com.webank.faceaction.Request.*{
*;
}
##############云刷脸动作活体混淆规则 faceverify-END#####################
```

您可以将如上代码拷贝到您的混淆文件中，也可以将 SDK 中的 webank-cloud-face-action-proguard-rules.pro 拷贝到主工程根目录下,然后通过 -include webank-cloud-face-action-rules.pro 加入到您的混淆文件中。

云刷脸 SDK（数字活体）的混淆规则

```
##############云刷脸混淆规则 faceverify-BEGIN#################
#不混淆内部类
-keepattributes InnerClasses
-keep public class com.webank.facenum.tools.WbCloudFaceVerifySdk{
public <methods>;
public static final *;
}
-keep public class com.webank.facenum.tools.WbCloudFaceVerifySdk$*{
*;
}
-keep public class com.webank.facenum.tools.ErrorCode{
*;
}
-keep public class com.webank.facenum.ui.FaceVerifyStatus{

}
-keep public class com.webank.facenum.ui.FaceVerifyStatus$Mode{
*;
}
-keep public class com.webank.facenum.tools.IdentifyCardValidate{
public <methods>;
}
-keep public class com.tencent.youtulivecheck.**{
*;
}
-keep public class com.webank.facenum.Request.*$*{
*;
}
-keep public class com.webank.facenum.Request.*{
*;
}
################云刷脸混淆规则 faceverify-END#######################
```

您可以将如上代码拷贝到您的混淆文件中，也可以将 SDK 中的 webank-cloud-face-num-proguard-rules.pro 拷贝到主工程根目录下,然后通过 -include webank-cloud-face-num-proguard-rules.pro 加入到您的混淆文件中。

云公共组件的混淆规则

```
############webank normal混淆规则-BEGIN###################
#不混淆内部类
-keepattributes InnerClasses
-keepattributes *Annotation*
-keepattributes Signature

-keep, allowobfuscation @interface com.webank.normal.xview.Inflater
-keep, allowobfuscation @interface com.webank.normal.xview.Find
-keep, allowobfuscation @interface com.webank.normal.xview.BindClick

-keep @com.webank.normal.xview.Inflater class *
-keepclassmembers class * {
@com.webank.normal.Find *;
@com.webank.normal.BindClick *;
}

-keep public class com.webank.normal.net.*$*{
*;
}
-keep public class com.webank.normal.net.*{
*;
}
-keep public class com.webank.normal.thread.*$*{
*;
}
-keep public class com.webank.normal.thread. *{
*;
}
-keep public class com.webank.normal.tools.WLogger{
*;
}

#wehttp混淆规则
-dontwarn com.webank.mbank.okio.**

-keep class com.webank.mbank.wehttp.**{
public <methods>;
}
-keep interface com.webank.mbank.wehttp.**{
public <methods>;
}
-keep public class com.webank.mbank.wehttp.WeLog$Level{
```

```
*;
}
-keep class com.webank.mbank.wejson.WeJson{
public <methods>;
}

#webank normal包含的第三方库bugly
-keep class com.tencent.bugly.webank.**{
*;
}
###########webank normal混淆规则-END#######################
```

您可以将如上代码拷贝到您的混淆文件中，也可以将 SDK 中的 webank-cloud-normal-proguard-rules.pro 拷贝到主工程根目录下,然后通过 -include webank-cloud-normal-rules.pro 加入到您的混淆文件中。

云刷脸依赖的第三方库的混淆规则

```
########云产品依赖的第三方库 混淆规则-BEGIN############

## support:appcompat-v7
-keep public class android.support.v7.widget.** { *; }
-keep public class android.support.v7.internal.widget.** { *; }
-keep public class android.support.v7.internal.view.menu.** { *; }

-keep public class * extends android.support.v4.view.ActionProvider {
public <init>(android.content.Context);
}

##########云产品依赖的第三方库 混淆规则-END############
```

您可以根据您现有的混淆规则，将缺少的第三库混淆规则拷贝到您的混淆文件中。

上一步：开发准备

下一步：接口调用

接口调用

最近更新时间：2018-08-27 16:12:24

> **注意：**
> 以下文章出现“刷脸”一词均可以表示为“人脸验证”。

SDK 接口调用方法

SDK代码调用的入口为 **WbCloudFaceVerifySdk** 这个类。

```
public class WbCloudFaceVeirfySdk {

/**
* 该类为一个单例，需要先获得单例对象再进行后续操作
*/
public static WbCloudFaceVeirfySdk getInstance(){
// ...
}
/**
* 在使用SDK前先初始化，传入需要的数据data，由 FaceVerifyLoginListener返回是否登录SDK成功
* 关于传入数据data见后面的说明
*/
public void init(Context context,Bundle data,FaceVerifyLoginListener loginListerner){
// ...
}
/**
* 登录成功后，调用此函数拉起sdk页面。
* 由 FaceVerifyResultForSecureListener返回刷脸结果。
*/
public void startActivityForSecurity(FaceVerifyResultForSecureListener listener) { // ...
}

/**
* 登录回调接口
*/
public interface FaceVerifyLoginListener {
void onLoginSuccess();
void onLoginFailed(String errorCode, String errorMsg);
}

/**
```

```
 * 刷脸结果回调接口
 */
public interface FaceVerifyResultForSecureListener {
/**
 * @PARAM resultCode 终端自己定义的错误码，详见ErrorCode
 * @PARAM nextShowGuide 下次是否要显示指引，返回第三方，由第三方存储传入
 * @PARAM faceCode 若错误是后台返回的，这里展示后台返回的错误码
 * @PARAM faceMsg 若错误是后台返回的，这里展示后台返回的错误信息;如果错误是终端返回的，这里展示前端错误信息
 * @PARAM sign 供app校验刷脸结果的安全性
 * @RARAM extendData 包含liverate和similarity两个对比分数字段，其中liverate是活体检测分数，similarity是人脸对比分数
 */
void onFinish(int resultCode, boolean nextShowGuide, String faceCode, String faceMsg, String sign, Bundle extendData);
}
```

WbCloudFaceVerifySdk.init() 的第二个参数用来传递数据.可以将参数打包到 data(Bundle) 中，必须传递的参数包括（参考要求详见本页接口参数说明的描述）：

```
//这些都是WbCloudFaceVerifySdk.InputData对象里的字段，是需要传入的数据信息
String userName; //用户姓名
String idType; //用户证件类型，01为身份证
String idNo; //用户身份证号
String agreementNo; //订单号
String clientIp; //用户ip信息,格式为”ip=xxx.xxx.xxx.xxx;”
// 示例：”ip=58.60.124.0”
String gps; //用户gps信息, 格式为”lgt=xxx;lat=xxx;”
//示例：“lgt=22.5044;lat=113.9537“

String openApiAppId; //APP_ID
String openApiAppVersion; //openapi Version
String openApiNonce; //32位随机字符串
String openApiUserId; //user id
String openApiSign; //签名信息

//是否需要显示刷脸指引
boolean isShowGuide;
//刷脸类别：动作活体 FaceVerifyStatus.Mode.MIDDLE
// 数字活体 FaceVerifyStatus.Mode.ADVANCED
FaceVerifyStatus.Mode verifyMode;
String keyLicence; //给合作方派发的licence
```

以上参数被封装在 WbCloudFaceVerifySdk.InputData 对象中（它是一个 Serializable 对象）。

关于接口调用的示例可参考 接入示例

接口参数说明

参数	说明	类型	长度（字节）	是否必填
userName	用户姓名	String	20	是
idType	用户证件类型，当参数值是 01 时代表身份证	String	2	是
userId	用户证件号码，为 18 位身份证号	String	18	是
agreementNo	订单号，合作方订单的唯一标识	String	32	是
clientIp	用户 IP 信息，格式为：”IP=xxx.xxx.xxx.xxx”；示例：”IP=58.60.124.0”。	String	30	是
gps	用户 GPS 信息，格式为：”lgt=xxx;lat=xxx”；示例：“lgt=22.5044;lat=113.9537”	String	30	是
openApiAppId	腾讯云线下对接分配的 App ID	String	腾讯云线下对接决定	是
openApiAppVersion	接口版本号，默认填 1.0.0	String	20	是
openApiNonce	32 位随机字符串，每次请求需要的一次性 nonce	String	32	是
openApiUserId	User Id，每个用户唯一的标识	String	30	是
openApiSign	合作方后台服务器通过 ticket 计算出来的签名信息	String	40	是
isShowGuide	是否需要显示刷脸指引，SDK 每次会返回这个结果，由 App 端存储，下次拉起时再传入	boolean	1	是
verifyMode	刷脸类型： 动作活体是： FaceVerifyStatus.Mode.MIDDLE 数字活体是： FaceVerifyStatus.Mode.ADVANCED	FaceVerifyStatus.Mode		是

参数	说明	类型	长度（字节）	是否必填
keyLicence	腾讯云线下对接分配派发的 Licence	String	腾讯云线下对接决定	是

个性化参数设置（可选）

WbCloudFaceVerifySdk.init() 里 Bundle data，除了必须要传的 InputData 对象之外，还可以由合作方为其传入一些个性化参数，量身打造更契合自己 App 的 SDK。如果合作方未设置这些参数，则以下所有参数按默认值设置。

是否显示刷脸成功页面

合作方可以控制是否显示 SDK 自带的刷脸成功页面。 SDK 默认显示刷脸成功页面，但第三方可以控制关闭不显示，直接回到自己的业务场景或者自定义的成功页面。设置代码如下：

```
# 在 MainActivity 中单击某个按钮的代码逻辑：
//先将必填的 InputData 放入 Bundle 中
data.putSerializable(WbCloudFaceVerifySdk.INPUT_DATA, inputData);
//是否展示刷脸成功页面，如果不设置则默认展示；设置了则以设置为准
//此处设置为不显示刷脸成功页
data.putBoolean(WbCloudFaceVerifySdk.SHOW_SUCCESS_PAGE, false);
```

是否显示刷脸失败原因页

合作方可以控制是否显示 SDK 自带的刷脸失败原因页。 SDK 默认显示刷脸失败原因页，但第三方可以控制关闭不显示，直接回到自己的业务场景或者自定义的失败页面。设置代码如下：

```
# 在 MainActivity 中单击某个按钮的代码逻辑：
//先将必填的 InputData 放入 Bundle 中
data.putSerializable(WbCloudFaceVerifySdk.INPUT_DATA, inputData);
//是否展示刷脸失败原因页，如果不设置则默认展示；设置了则以设置为准
//此处设置为不显示刷脸成功页
data.putBoolean(WbCloudFaceVerifySdk.SHOW_FAIL_PAGE, false);
```

SDK 样式选择

合作方可以选择 SDK 样式，其中数字活体需要和 SDK 资源包一起加载显示。目前 SDK 有黑色模式和白色模式两种，默认显示黑色模式。设置代码如下:

```
# 在 MainActivity 中单击某个按钮的代码逻辑：
//先将必填的 InputData 放入 Bundle中
data.putSerializable(WbCloudFaceVerifySdk.INPUT_DATA, inputData);
//对 sdk 样式进行设置，默认为黑色模式
//此处设置为白色模式（数字活体需要与白色资源包一起配合使用）
data.putString(WbCloudFaceVerifySdk.COLOR_MODE, WbCloudFaceVerifySdk.WHITE);
```

对比类型选择

SDK 提供 **权威库网纹图片比对**、**自带比对源比对** 和 **仅活体检测** 三种对比类型，合作方可以选择传入参数控制对比类型。

权威库网纹图片比对 类型，合作方必须要在 inputData 对象中送入用户的姓名与身份证号信息，供 SDK 与公安网纹图片进行比对。SDK 默认类型为 **权威库网纹图片比对**，不需要额外设置。设置代码如下：

```
# 在MainActivity中单击某个按钮的代码逻辑：
//先将必填的InputData放入Bundle中
//权威库网纹图片比对必须输入用户的姓名与身份证信息
data.putSerializable(WbCloudFaceVerifySdk.INPUT_DATA, inputData);
//设置选择的比对类型 权威库网纹图片对比(无需格外设置，默认选项)
data.putString(WbCloudFaceVerifySdk.COMPARE_TYPE, WbCloudFaceVerifySdk.ID_CARD);
```

自带比对源比对 类型，合作方给 SDK 送上自己提供对比源数据进行对比。合作方可以上送两类照片，一类是网纹照，一类是高清照；照片需要转化为经过 base64 编码后的 String 来上送。图片大小不可超过 2MB，经过编码后的图片 String 大小不可超过 3MB。上送照片类型与上送照片 String 两者缺一不可，否则将不使用上送的数据源。不自带对比源的合作方可以不上送此两个字段。上送的代码设置如下：

```
# 在MainActivity中单击某个按钮的代码逻辑：
//先将必填的InputData放入Bundle中
data.putSerializable(WbCloudFaceVerifySdk.INPUT_DATA, inputData);
//设置选择的比对类型 自带比对源比对
data.putString(WbCloudFaceVerifySdk.COMPARE_TYPE, WbCloudFaceVerifySdk.SRC_IMG);
//自带比对源比对,需要合作方上送数据源信息
//照片类型与照片string缺一不可
//上送照片类型，1是网纹照 2是高清照
data.putString(WbCloudFaceVerifySdk.SRC_PHOTO_TYPE, srcPhotoType);
//比对源照片的BASE64 string
data.putString(WbCloudFaceVerifySdk.SRC_PHOTO_STRING, srcPhotoString);
```

仅活体检测 类型，合作方无需上送任何用户比对源信息，SDK 仅仅对用户进行活体检测。设置的代码如下：

```
# 在MainActivity中单击某个按钮的代码逻辑：
//先将必填的InputData放入Bundle中
data.putSerializable(WbCloudFaceVerifySdk.INPUT_DATA, inputData);
```

```
//设置选择的比对类型 仅活体检测
data.putString(WbCloudFaceVerifySdk.COMPARE_TYPE, WbCloudFaceVerifySdk.NONE);
```

是否录制视频存证

SDK 为了进一步确保刷脸的安全性，默认会录制用户刷脸视频做存证。如果合作方不需要录制视频存证，可以通过该字段进行设置，关闭 SDK 录制视频存证。设置代码如下：

```
# 在 MainActivity 中单击某个按钮的代码逻辑：
//先将必填的 InputData 放入 Bundle 中
data.putSerializable(WbCloudFaceVerifySdk.INPUT_DATA, inputData);
//设置是否录制视频进行存证，默认录制存证。
//此处设置为不录制存证
data.putBoolean(WbCloudFaceVerifySdk.RECORD_VIDEO, false);
```

是否对录制视频进行检查

SDK 为了进一步确保刷脸的安全性，不论是简单还是中级模式都有录制用户刷脸视频做存证。但其实在简单 / 中级模式中，起到识别作用的并不是视频文件。在 SDK 使用过程中，发现视频录制在性能不太好的手机上可能会报错，导致刷脸中断，影响用户体验。

为了减少因为录制视频原因导致的刷脸中断问题，SDK 默认设置对录制的视频不作检测。如果合作方对刷脸安全有进一步的更加严格的要求，可以考虑打开这一选项。但打开这个字段可能导致某些低性能手机上用户刷脸不能进行，请慎重考虑。设置代码如下：

```
# 在 MainActivity 中单击某个按钮的代码逻辑：
//先将必填的 InputData 放入 Bundle 中
data.putSerializable(WbCloudFaceVerifySdk.INPUT_DATA, inputData);
//设置是否对录制的视频进行检测，默认不检测
//此处设置为检测
data.putBoolean(WbCloudFaceVerifySdk.VIDEO_CHECK, true);
```

个性化设置接入示例

```
# 在MainActivity中单击某个按钮的代码逻辑：
//先将必填的InputData放入Bundle中
data.putSerializable(WbCloudFaceVerifySdk.INPUT_DATA, inputData);
//个性化参数设置,此处均设置为与默认相反
//是否显示成功结果页，默认显示，此处设置为不显示
data.putBoolean(WbCloudFaceVerifySdk.SHOW_SUCCESS_PAGE, false);
//是否展示刷脸失败页面，默认展示,此处设置为不显示
data.putBoolean(WbCloudFaceVerifySdk.SHOW_FAIL_PAGE, false);
//sdk样式设置，需要与资源包一起配合使用
//默认黑色模式
```

```
data.putString(WbCloudFaceVerifySdk.COLOR_MODE, WbCloudFaceVerifySdk.BLACK);
//设置选择的比对类型 默认为权威库网纹图片对比
//此处设置自带比对源比对
data.putString(WbCloudFaceVerifySdk.COMPARE_TYPE, WbCloudFaceVerifySdk.SRC_IMG);
//上送自带的数据源信息，照片类型与照片string缺一不可
//不带比对源的可不传这两个字段
//上送照片类型，1是网纹照 2是高清照
data.putString(WbCloudFaceVerifySdk.SRC_PHOTO_TYPE, srcPhotoType);
//比对源照片的BASE64 string
data.putString(WbCloudFaceVerifySdk.SRC_PHOTO_STRING, srcPhotoString);
//是否录制视频进行存证，默认录制存证，此处设置为不录制存证
data.putBoolean(WbCloudFaceVerifySdk.RECORD_VIDEO, false);
//是否对录制视频进行检查,默认不检查，此处设置为检查
data.putBoolean(WbCloudFaceVerifySdk.VIDEO_CHECK, true);
```

上一步：配置流程

更多相关内容可参考：

错误码描述

接入示例

错误码描述

最近更新时间：2018-02-01 18:13:23

注意：
以下文章出现“刷脸”一词均可以表示为“人脸验证”。

登录错误码

错误码	错误描述
FACEVERIFY_LOGIN_ERROR = "-10000";	登录请求错误，没有返回 code
FACEVERIFY_LOGIN_PARAMETER_ERROR = "-20000";	传入参数有误
FACEVERIFY_LOGIN_NO_RESONSE = "-30000";	登录请求未到达后台，非 200

人脸验证错误码

错误码	错误描述
FACEVERIFY_NOERROR = 0;	刷脸成功完成，通过刷脸
FACEVERIFY_ERROR_DEFAULT = 10000;	刷脸完成，但验证失败
FACEVERIFY_ERROR_CANCELED = 20000;	用户取消
FACEVERIFY_ERROR_CANCELED_BEFORE = 21000;	引导页取消
FACEVERIFY_ERROR_CANCELED_DURING = 22000;	刷脸中取消
FACEVERIFY_ERROR_CANCELED_AFTER = 23000;	上传时取消
FACEVERIFY_ERROR_CANCELED_VOICE_LOW=24000;	高级模式音量太低取消
FACEVERIFY_ERROR_NETWORK = 30000;	网络错误 没有网络
FACEVERIFY_ERROR_NETWORK_ACTIVE = 32000;	拉活体模式错误
FACEVERIFY_ERROR_NETWORK_LIPS = 33000;	拉唇语错误
FACEVERIFY_ERROR_NETWORK_UPLOAD = 34000;	上传错误

错误码	错误描述
FACEVERIFY_ERROR_PERMISSION = 40000;	权限异常
FACEVERIFY_ERROR_PERMISSION_CAMERA = 41000;	相机未授权
FACEVERIFY_ERROR_PERMISSION_MIC = 42000;	麦克风未授权
FACEVERIFY_ERROR_PERMISSION_READ_PHONE=43000;	READ_PHONE 未授权
FACEVERIFY_ERROR_CAMERA = 50000;	音视频设备异常
FACEVERIFY_ERROR_MEDIARECORD = 60000;	视频录制出错
FACEVERIFY_ERROR_OUT_OF_TIME = 70000;	验证超时
FACEVERIFY_ERROR_OUT_OF_TIME_FACE_DETECT = 71000;	扫脸验证超时
FACEVERIFY_ERROR_OUT_OF_TIME_ACTIVE_DETECT = 72000;	活体验证超时
FACEVERIFY_ERROR_ACTIVE_DETECT_NOFACE=80000;	活体检测时人脸曾移出框外

接入示例

最近更新时间：2018-08-28 09:12:53

```
# 在MainActivity中单击某个按钮的代码逻辑：
//先填好数据
Bundle data = new Bundle();
WbCloudFaceVerifySdk.InputData inputData = new WbCloudFaceVerifySdk.InputData(
userName,
idType,
idNo,
agreementNo,
clientIp,
gps,
openApiAppId,
openApiAppVersion,
openApiNonce,
userId,
userSign,
isShowGuide,
verifyMode,
keyLicence);
data.putSerializable(WbCloudFaceVerifySdk.INPUT_DATA, inputData);

//个性化参数设置,可以不设置，不设置则为默认选项。
//此处均设置为与默认相反
//是否显示成功结果页，默认显示，此处设置为不显示
data.putBoolean(WbCloudFaceVerifySdk.SHOW_SUCCESS_PAGE, false);
//是否展示刷脸失败页面，默认展示,此处设置为不显示
data.putBoolean(WbCloudFaceVerifySdk.SHOW_FAIL_PAGE, false);
//sdk样式设置，需要与资源包一起配合使用
//默认黑色模式
data.putString(WbCloudFaceVerifySdk.COLOR_MODE, WbCloudFaceVerifySdk.BLACK);
//设置选择的比对类型 默认为权威库网纹图片对比
//权威库网纹图片比对 WbCloudFaceVerifySdk.ID_CRAD
//自带比对源比对 WbCloudFaceVerifySdk.SRC_IMG
//仅活体检测 WbCloudFaceVerifySdk.NONE
//此处设置自带比对源比对
data.putString(WbCloudFaceVerifySdk.COMPARE_TYPE, WbCloudFaceVerifySdk.SRC_IMG);
//上送自带的数据源信息，照片类型与照片string缺一不可
// 不带比对源的可不传这两个字段
//上送照片类型，1是网纹照 2是高清照
data.putString(WbCloudFaceVerifySdk.SRC_PHOTO_TYPE, srcPhotoType);
//比对源照片的BASE64 string
data.putString(WbCloudFaceVerifySdk.SRC_PHOTO_STRING, srcPhotoString);
//是否需要录制视频存证,默认录制，此处设置为不录制
```

```
data.putBoolean(WbCloudFaceVerifySdk.RECORD_VIDEO, false);
//是否对录制视频进行检查,默认不检查，此处设置为检查
data.putBoolean(WbCloudFaceVerifySdk.VIDEO_CHECK, true);


//初始化sdk，得到是否登录sdk成功的结果
WbCloudFaceVerifySdk.getInstance().init(
MainActivity.this,
data,
//由FaceVerifyLoginListener返回登录结果
new WbCloudFaceVerifySdk.FaceVerifyLoginListener() {
@Override
public void onLoginSuccess() {
//登录成功，拉起sdk页面
WbCloudFaceVerifySdk.getInstance().startActivityForSecurity(new WbCloudFaceVerifySdk.FaceVerifyResult
ForSecureListener() {
//由FaceVerifyResultListener返回刷脸结果
@Override
public void onFinish(int resultCode, boolean nextShowGuide, String faceCode, String faceMsg, String sig
n, Bundle extendData) {
//liveRate为活体检测分数
//similarity为人脸对比分数
//userImage为前端返回的人脸识别的图片，仅在刷脸成功时返回
String liveRate = null;
String similarity = null;
String userImage = null;
if (extendData != null) {
liveRate = extendData.getString(WbCloudFaceVerifySdk.FACE_RESULT_LIVE_RATE);
similarity = extendData.getString(WbCloudFaceVerifySdk.FACE_RESULT_SIMILIRATY);
userImage = extendData.getString(WbCloudFaceVerifySdk.FACE_RESULT_USER_IMG);
}
// resultCode为0，则刷脸成功；否则刷脸失败
if (resultCode == 0) {
Log.d(TAG, "刷脸成功！前端可以拿到刷脸照片！liveRate=" + liveRate + "; similarity=" + similarity + "; userIm
g=" + userImage);
} else {
Log.d(TAG, "刷脸失败！");
if (resultCode == ErrorCode.FACEVERIFY_ERROR_DEFAULT) {
Log.d(TAG, "后台对比失败! liveRate=" + liveRate + "; similarity=" + similarity);
if (faceCode.equals("66660004")) {
Log.d(TAG, "但是后台还是可以拿到刷脸图片!");
} else {
Log.d(TAG, "后台拿不到刷脸图片！");
}
} else {
Log.d(TAG, "前端失败！");
}
```

```
}
}
});
}
@Override
public void onLoginFailed(String errorCode, String errorMsg) {
//登录失败
...
}
```

Android 光线活体 + 人脸比对
开发准备

最近更新时间：2018-02-01 18:13:31

权限检测

SDK 需要用到相机/录音/读取手机信息权限，在 Android 6.0 以上系统，SDK 对其做了权限的运行时检测。但是由于 Android 6.0 以下系统 Android 并没有运行时权限，检测权限只能靠开关相机/麦克风进行。考虑到 SDK 的使用时间很短，快速频繁开关相机/麦克风可能会导致手机抛出异常，故 SDK 内对 Android 6.0 以下手机没有做权限的检测。为了进一步提高用户体验，在 Android 6.0 以下系统上，**我们建议合作方在拉起 SDK 前，帮助 SDK 做相机/麦克风/读取手机信息权限检测，提示用户确认打开了这三项权限后再进行人脸验证**，可以使整个人脸验证体验更快更好。

CPU 平台设置

目前SDK只支持 armeabi-v7a 平台，为了防止在其他 CPU 平台上 SDK crash，我们建议在您的 App 的 build.gradle 里加上 abiFilter，如下图中红框所示：

```
defaultConfig {
    applicationId "com.webank.testcloud.cloudfacetest"
    minSdkVersion 14
    targetSdkVersion 23
    versionCode 1
    versionName "1.0"

    ndk {
        // 设置支持的SO库架构
        abiFilters 'armeabi-v7a'
    }
}
```

下一步：配置流程

配置流程

最近更新时间：2018-02-05 17:46:58

> **注意：**
> 以下文章出现“刷脸”一词均可以表示为“人脸验证”。

1. 接入配置

云刷脸 SDK（WbCloudFaceVerify）最低支持到 Android API 14: Android 4.0(ICS)，请在构建项目时注意。

刷脸 SDK 将以 AAR 文件的形式提供，默认黑色皮肤，无需格外设置。另外刷脸 SDK 同时需要依赖 **云公共组件 WbCloudNormal**，同样也是以 AAR 文件的形式提供。

需要添加下面文档中所示的依赖（将提供的 AAR 文件加入到 App 工程的 libs 文件夹下面，并且在 **build.gradle** 中添加下面的配置）：

```
android{
//...
repositories {
flatDir {
dirs 'libs' //this way we can find the .aar file in libs folder
}
}
}
//添加依赖
dependencies {
//0. appcompat-v7
compile 'com.android.support:appcompat-v7:23.0.1'
//1. 云刷脸SDK
compile(name: 'WbCloudFaceVerifySdk', ext: 'aar')
//2. 云normal SDK
compile(name: 'WbCloudNormal', ext: 'aar')
}
}
```

2. 混淆配置

云刷脸产品的混淆规则分为三部分，分别是云刷脸 SDK（光线活体）的混淆规则、云公共组件的混淆规则以及依赖的第三方库混淆规则。

云刷脸 SDK（光线活体）的混淆规则

```
###############云刷脸（光线活体）混淆规则 faceverify-BEGIN##################
#不混淆内部类
-keepattributes InnerClasses

-keep public class com.webank.facelight.tools.WbCloudFaceVerifySdk{
public <methods>;
public static final *;
}
-keep public class com.webank. facelight.tools.WbCloudFaceVerifySdk$*{
*;
}
-keep public class com.webank.record.**{
public <methods>;
public static final *;
}
-keep public class com.webank. facelight.ui.FaceVerifyStatus{

}
-keep public class com.webank. facelight.ui.FaceVerifyStatus$Mode{
*;
}
-keep public class com.webank. facelight.tools.IdentifyCardValidate{
public <methods>;
}
-keep public class com.tencent.youtulivecheck.**{
*;
}
-keep public class com.webank. facelight.contants.**{
*;
}
-keep public class com.webank. facelight.listerners.**{
*;
}
-keep public class com.webank. facelight.Request.*$*{
*;
}
-keep public class com.webank. facelight.Request.*{
*;
}
-keep public class com.webank. facelight.config.FaceVerifyConfig {
public <methods>;
}

-keep public class com.webank. facelight.ui.fragment.FaceLiveUseCase{
*;
```

```
}
-keep public class com.webank. facelight.ui.fragment.FaceLiveUseCase$YTFaceLiveCallback{
*;
}
-keep class com.tencent.youtuface.**{
*;
}
-keep class com.tencent.youtulivecheck.**{
*;
}
-keep class com.tencent.youtufacetrack.**{
*;
}
-keep class com.tencent.youtufacelive.model.**{
*;
}
-keep class com.tencent.youtufacelive.tools.YTUtils{
public <methods>;
}
-keep class com.tencent.youtufacelive.tools.YTFaceLiveLogger{
public <methods>;
}
-keep class com.tencent.youtufacelive.tools.YTFaceLiveLogger$IFaceLiveLogger{
*;
}
-keep class com.tencent.youtufacelive.IYTMaskStateListener{
*;
}
-keep class com.tencent.youtufacelive.YTPreviewHandlerThread{
public static *;
public <methods>;
}
-keep class com.tencent.youtufacelive.YTPreviewHandlerThread$IUploadListener{
*;
}
-keep class com.tencent.youtufacelive.YTPreviewHandlerThread$ISetCameraParameterListener{
*;
}
-keep class com.tencent.youtufacelive.YTPreviewMask{
public <methods>;
}
-keep class com.tencent.youtufacelive.YTPreviewMask$TickCallback{
*;
}
# 保留自定义控件(继承自View)不能被混淆
-keep public class * extends android.view.View {
public <init>(android.content.Context);
```

```
public <init>(android.content.Context, android.util.AttributeSet);
public <init>(android.content.Context, android.util.AttributeSet, int);
public void set*(***);
*** get* ();
}
################云刷脸混淆规则 faceverify-END#########################
```

您可以将如上代码拷贝到您的混淆文件中，也可以将 SDK 中的 webank-cloud-face-light-proguard-rules.pro 拷贝到主工程根目录下,然后通过 -include webank-cloud-face-light-rules.pro 加入到您的混淆文件中。

云公共组件的混淆规则

```
##############webank normal混淆规则-BEGIN###################
#不混淆内部类
-keepattributes InnerClasses
-keepattributes *Annotation*
-keepattributes Signature

-keep, allowobfuscation @interface com.webank.normal.xview.Inflater
-keep, allowobfuscation @interface com.webank.normal.xview.Find
-keep, allowobfuscation @interface com.webank.normal.xview.BindClick

-keep @com.webank.normal.xview.Inflater class *
-keepclassmembers class * {
@com.webank.normal.Find *;
@com.webank.normal.BindClick *;
}

-keep public class com.webank.normal.net.*$*{
*;
}
-keep public class com.webank.normal.net.*{
*;
}
-keep public class com.webank.normal.thread.*$*{
*;
}
-keep public class com.webank.normal.thread. *{
*;
}
-keep public class com.webank.normal.tools.WLogger{
*;
}
-keep public class com.webank.normal.tools.*{
*;
}
```

```
#wehttp混淆规则
-dontwarn com.webank.mbank.okio.**

-keep class com.webank.mbank.wehttp.**{
public <methods>;
}
-keep interface com.webank.mbank.wehttp.**{
public <methods>;
}
-keep public class com.webank.mbank.wehttp.WeLog$Level{
*;
}
-keep class com.webank.mbank.wejson.WeJson{
public <methods>;
}

#webank normal包含的第三方库bugly
-keep class com.tencent.bugly.webank.**{
*;
}
###########webank normal混淆规则-END#########################
```

您可以将如上代码拷贝到您的混淆文件中，也可以将 SDK 中的 webank-cloud-normal-proguard-rules.pro 拷贝到主工程根目录下,然后通过 -include webank-cloud-normal-rules.pro 加入到您的混淆文件中。

云刷脸依赖的第三方库的混淆规则

```
########云产品依赖的第三方库 混淆规则-BEGIN#############

## support:appcompat-v7
-keep public class android.support.v7.widget.** { *; }
-keep public class android.support.v7.internal.widget.** { *; }
-keep public class android.support.v7.internal.view.menu.** { *; }

-keep public class * extends android.support.v4.view.ActionProvider {
public <init>(android.content.Context);
}

#########云产品依赖的第三方库 混淆规则-END#############
```

您可以根据您现有的混淆规则，将缺少的第三库混淆规则拷贝到您的混淆文件中。

上一步：开发准备

下一步：接口调用

接口调用

最近更新时间：2018-12-07 14:50:16

> ⚠ **注意：**
>
> 以下文章出现“刷脸”一词均可以表示为“人脸验证”。

SDK 接口调用方法

SDK 代码调用的入口为：**WbCloudFaceVerifySdk** 这个类。

```
public class WbCloudFaceVeirfySdk {

/**
* 该类为一个单例，需要先获得单例对象再进行后续操作
*/
public static WbCloudFaceVeirfySdk getInstance(){
// ...
}
/**
* 在使用SDK前先初始化，传入需要的数据data
* 由 WbCloudFaceVerifyLoginListener返回是否登录SDK成功
* 关于传入数据data见后面的说明
*/
public void init(Context context,Bundle data,
WbCloudFaceVerifyLoginListener loginListerner){
// ...
}
/**
* 登录成功后，调用此函数拉起sdk页面。
* 由 FaceVerifyResultForSecureListener返回刷脸结果。
*/
public void startWbFaceVerifySdk(Context ctx,
WbCloudFaceVerifyResultListener listener) { // ...
}
```

WbCloudFaceVerifySdk.init() 的第二个参数用来传递数据。可以将参数打包到 data(Bundle) 中，必须传递的参数包括（参考要求详见本页 接口参数说明 ）：

```
//这些都是WbCloudFaceVerifySdk.InputData对象里的字段，是需要传入的数据信息
String userName; //用户姓名
String idType; //用户证件类型，01为身份证
String idNo; //用户身份证号
```

```
String agreementNo; //订单号
String clientIp; //用户ip信息,格式为”ip=xxx.xxx.xxx.xxx;”
// 示例：”ip=58.60.124.0”
String gps; //用户gps信息, 格式为”lgt=xxx;lat=xxx;”
//示例： “lgt=22.5044;lat=113.9537 “

String openApiAppId; //APP_ID
String openApiAppVersion; //openapi Version
String openApiNonce; //32位随机字符串
String openApiUserId; //user id
String openApiSign; //签名信息

//刷脸类别：动作活体 FaceVerifyStatus.Mode.MIDDLE
// 数字活体 FaceVerifyStatus.Mode.ADVANCED
// 光线活体 FaceVerifyStatus.Mode.REFLECT
FaceVerifyStatus.Mode verifyMode;
String keyLicence; //给合作方派发的licence
```

以上参数被封装在 WbCloudFaceVerifySdk.InputData 对象中（它是一个 Serializable 对象）。

而 SDK 的结果分别由以下两个回调返回：

```
/**
* 登录回调接口 返回登录sdk是否成功
*/
public interface WbCloudFaceVerifyLoginListener {
void onLoginSuccess();
void onLoginFailed(WbFaceError error);
}

/**
* 刷脸结果回调接口
*/
public interface WbCloudFaceVerifyResultListener {
void onFinish(WbFaceVerifyResult result);
```

关于接口调用的示例可参考 接入示例。

接口参数说明

InputData 对象说明

InputData 是用来给 SDK 传递一些必须参数所需要使用的对象（WbCloudFaceVerifySdk.init() 的第二个参数），合作方需要往里塞入SDK 需要的一些数据以便启动刷脸 SDK。

其中 InputData 对象中的各个参数定义如下表，请合作方按下表标准传入对应的数据。

参数	说明	类型	长度（字节）	是否必填
userName	用户姓名， 权威库网纹图片比对类型	String	20	是
idType	用户证件类型， 权威库网纹图片比对类型，当参数值是01时代表身份证	String	2	是
userId	用户证件号码，权威库网纹图片比对类型，为18位身份证号	String	18	是
agreementNo	订单号，合作方订单的唯一标识	String	32	是
clientIp	用户 IP 信息，格式为：”IP = xxx.xxx.xxx.xxx”；示例：”IP = 58.60.124.0”。	String	30	是
gps	用户 GPS 信息，格式为：”lgt = xxx;lat=xxx”；示例：“lgt = 22.5044;lat = 113.9537“	String	30	是
openApiAppId	腾讯云线下对接分配的 App ID	String	腾讯云线下对接决定	是
openApiAppVersion	接口版本号，默认填1.0.0	String	20	是
openApiNonce	32位随机字符串，每次请求需要的一次性 nonce	String	32	是
openApiUserId	User Id，每个用户唯一的标识	String	30	是
openApiSign	合作方后台服务器通过 ticket 计算出来的签名信息	String	40	是
verifyMode	刷脸类型： 动作活体是： FaceVerifyStatus.Mode.MIDDLE 数字活体是： FaceVerifyStatus.Mode.ADVANCED 光线活体 FaceVerifyStatus.Mode.REFLECT	FaceVerifyStatus.Mode	-	是

参数	说明	类型	长度（字节）	是否必填
keyLicence	腾讯云线下对接分配派发的 Licence	String	腾讯云线下对接决定	是

WbFaceVerifyResult 对象说明

WbFaceVerifyResult 是 SDK 用来给合作方传递身份识别结果的对象，在 WbCloudFaceVerifyResultListener 回调中作为参数返回给合作方 App。
其中 WbFaceVerifyResult 对象的各个字段意义如下表所示：

字段名	类型	字段含义	说明
isSuccess	boolean	刷脸是否成功	无
sign	String	签名	供 App 校验刷脸结果的安全性
liveRate	String	活体检测分数	无
similarity	String	人脸比对分数	“仅活体检测” 类型不提供此分数
userImageString	String	用户刷脸图片	经过 base64 编码后的用户刷脸图片,仅用户成功通过验证时返回
WbFaceError	自定义对象	刷脸错误	刷脸成功时为null

WbFaceError对象说明

WbFaceError是 SDK 用来给合作方传递刷脸错误信息的对象，在WbCloudFaceVerifyLoginListener 回调和 WbFaceVerifyResult 对象中作为参数返回给合作方 App。
其中 WbFaceError 对象的各个字段意义如下表所示，各个字段的内容取值可以见 错误码描述 文档。

字段名	类型	字段含义	说明
domain	String	错误发生的阶段	无
code	String	错误码	无

字段名	类型	字段含义	说明
desc	String	错误描述	如有需求，可以展示给用户
reason	String	错误信息内容	错误的详细实际原因，主要用于定位问题

个性化参数设置（可选）

WbCloudFaceVerifySdk.init() 里 Bundle data，除了必须要传的 InputData 对象之外，还可以由合作方为其传入一些个性化参数，量身打造更契合自己 App 的 SDK。如果合作方未设置这些参数，则以下所有参数按默认值设置。

是否显示刷脸成功页面

合作方可以控制是否显示 SDK 自带的刷脸成功页面。 SDK 默认显示刷脸成功页面，但第三方可以控制关闭不显示，直接回到自己的业务场景或者自定义的成功页面。设置代码如下：

```
# 在MainActivity中单击某个按钮的代码逻辑：
//先将必填的InputData放入Bundle中
data.putSerializable(WbCloudFaceContant.INPUT_DATA, inputData);
//是否展示刷脸成功页面，如果不设置则默认展示；设置了则以设置为准
//此处设置为不显示刷脸成功页
data.putBoolean(WbCloudFaceContant.SHOW_SUCCESS_PAGE, false);
```

是否显示刷脸失败原因页

合作方可以控制是否显示 SDK 自带的刷脸失败原因页。 SDK 默认显示刷脸失败原因页，但第三方可以控制关闭不显示，直接回到自己的业务场景或者自定义的失败页面。设置代码如下：

```
# 在MainActivity中单击某个按钮的代码逻辑：
//先将必填的InputData放入Bundle中
data.putSerializable(WbCloudFaceContant.INPUT_DATA, inputData);
//是否展示刷脸失败原因页，如果不设置则默认展示；设置了则以设置为准
//此处设置为不显示刷脸成功页
data.putBoolean(WbCloudFaceContant.SHOW_FAIL_PAGE, false);
```

SDK 样式选择

合作方可以选择 SDK 样式。目前 SDK 有黑色模式和白色模式两种，默认显示黑色模式。设置代码如下:

```
# 在 MainActivity 中单击某个按钮的代码逻辑：
//先将必填的 InputData 放入 Bundle中
data.putSerializable((WbCloudFaceContant.INPUT_DATA, inputData);
//对 sdk 样式进行设置，默认为黑色模式
```

```
//此处设置为白色模式
data.putString(WbCloudFaceContant.COLOR_MODE, WbCloudFaceContant.WHITE);
```

对比类型选择

SDK 提供 **权威库网纹图片比对**、**自带比对源比对** 和 **仅活体检测** 三种对比类型，合作方可以选择传入参数控制对比类型。

权威库网纹图片比对 类型，合作方必须要在 inputData 对象中送入用户的姓名与身份证号信息，供 SDK 与公安网纹图片进行比对。SDK 默认类型为 **权威库网纹图片比对**，不需要额外设置。设置代码如下：

```
# 在MainActivity中单击某个按钮的代码逻辑：
//先将必填的InputData放入Bundle中
//权威库网纹图片比对必须输入用户的姓名与身份证信息
data.putSerializable(WbCloudFaceContant.INPUT_DATA, inputData);
//设置选择的比对类型 权威库网纹图片对比(无需格外设置，默认选项)
data.putString(WbCloudFaceContant.COMPARE_TYPE, WbCloudFaceContant.ID_CARD);
```

自带比对源比对 类型，合作方给 SDK 送上自己提供对比源数据进行对比。合作方可以上送两类照片，一类是网纹照，一类是高清照；照片需要转化为经过 base64 编码后的 String 来上送。图片大小不可超过500k，经过编码后的图片 String 大小不可超过1MB。上送照片类型与上送照片 String 两者缺一不可，否则将不使用上送的数据源。不自带对比源的合作方可以不上送此两个字段。上送的代码设置如下：

```
# 在MainActivity中单击某个按钮的代码逻辑：
//先将必填的InputData放入Bundle中
data.putSerializable(WbCloudFaceContant.INPUT_DATA, inputData);
//设置选择的比对类型 自带比对源比对
data.putString(WbCloudFaceContant.COMPARE_TYPE, WbCloudFaceContant.SRC_IMG);
//自带比对源比对,需要合作方上送数据源信息
//照片类型与照片string缺一不可
//上送照片类型，1是网纹照 2是高清照
data.putString(WbCloudFaceContant.SRC_PHOTO_TYPE, srcPhotoType);
//比对源照片的BASE64 string
data.putString(WbCloudFaceContant.SRC_PHOTO_STRING, srcPhotoString);
```

仅活体检测 类型，合作方无需上送任何用户比对源信息，SDK 仅仅对用户进行活体检测。设置的代码如下：

```
# 在MainActivity中单击某个按钮的代码逻辑：
//先将必填的InputData放入Bundle中
data.putSerializable(WbCloudFaceContant.INPUT_DATA, inputData);
//设置选择的比对类型 仅活体检测
data.putString(WbCloudFaceContant.COMPARE_TYPE, WbCloudFaceContant.NONE);
```

是否录制视频存证

SDK 为了进一步确保刷脸的安全性，默认会录制用户刷脸视频做存证。如果合作方不需要录制视频存证，可以通过该字段进行设置，关闭 SDK 录制视频存证。设置代码如下：

```
# 在MainActivity中单击某个按钮的代码逻辑：
//先将必填的InputData放入Bundle中
data.putSerializable(WbCloudFaceContant.INPUT_DATA, inputData);
//设置是否录制视频进行存证，默认录制存证。
//此处设置为不录制存证
data.putBoolean(WbCloudFaceContant.VIDEO_UPLOAD, false);
```

是否对录制视频进行检查

（如果上一节的 **是否录制视频存证** 的设置为录制存证，则目前该字段有效，否则不录制视频的话，也不会对视频进行检查，设置无效）

SDK 为了进一步确保刷脸的安全性，默认都有录制用户刷脸视频做存证。但其实，起到识别作用的并不是视频文件。在 SDK 使用过程中，发现视频录制在性能不太好的手机上可能会报错，导致刷脸中断，影响用户体验。

为了减少因为录制视频原因导致的刷脸中断问题，SDK 默认设置对录制的视频不作检测。如果合作方对刷脸安全有进一步的更加严格的要求，可以考虑打开这一选项。但打开这个字段可能导致某些低性能手机上用户刷脸不能进行，请慎重考虑。设置代码如下：

```
# 在MainActivity中单击某个按钮的代码逻辑：
//先将必填的InputData放入Bundle中
data.putSerializable(WbCloudFaceContant.INPUT_DATA, inputData);
//设置是否对录制的视频进行检测，默认不检测
//此处设置为检测
data.putBoolean(WbCloudFaceContant.VIDEO_CHECK, true);
```

个性化设置接入示例

```
# 在MainActivity中单击某个按钮的代码逻辑：
//先将必填的InputData放入Bundle中
data.putSerializable(WbCloudFaceContant.INPUT_DATA, inputData);
//个性化参数设置,此处均设置为与默认相反
//是否显示成功结果页，默认显示，此处设置为不显示
data.putBoolean(WbCloudFaceContant.SHOW_SUCCESS_PAGE, false);
//是否展示刷脸失败页面，默认展示,此处设置为不显示
data.putBoolean(WbCloudFaceContant.SHOW_FAIL_PAGE, false);
//sdk样式设置，默认为黑色
//此处设置为白色
data.putString(WbCloudFaceContant.COLOR_MODE, WbCloudFaceContant.WHITE);
//设置选择的比对类型 默认为权威库网纹图片对比
//此处设置自带比对源比对
```

```
data.putString(WbCloudFaceContant.COMPARE_TYPE, WbCloudFaceContant.SRC_IMG);
//上送自带的数据源信息，照片类型与照片string缺一不可
//不带比对源的可不传这两个字段
//上送照片类型，1是网纹照 2是高清照
data.putString(WbCloudFaceContant.SRC_PHOTO_TYPE, srcPhotoType);
//比对源照片的BASE64 string
data.putString(WbCloudFaceContant.SRC_PHOTO_STRING, srcPhotoString);
//是否需要录制上传视频 默认需要，此处设置为不需要
data.putBoolean(WbCloudFaceContant.VIDEO_UPLOAD, false);
//是否对录制视频进行检查,默认不检查，此处设置为检查
data.putBoolean(WbCloudFaceContant.VIDEO_CHECK, true);
```

上一步：配置流程

更多相关内容可参考：

错误码描述
接入示例

错误码描述

最近更新时间：2018-07-27 16:21:10

注意：
以下文章出现“刷脸”一词均可以表示为“人脸验证”。

SDK 在登录以及返回人脸服务结果时，如果发生错误或者识别失败会返回 WBFaceError 对象，该对象的字段结构意义见接口参数说明，其中各个字段的内容如下:

WBFaceErrorDomainInputParams

Code（错误码）	Description（描述）	Reason（详细实际原因）
11000	传入参数为空	传入的 xx为空
11001	传入的 keyLicence 不可用	传入的 keyLicence 不可用
11002	身份证格式不正确	身份证格式不正确
11003	使用自带对比源，传入参数错误，非 base64	传入的 srcPhotoString 不是 base64
11004	使用自带对比源，传入参数错误，超过 1 MB	传入的 srcPhotoString 超过 1 MB
11005	SDK 资源引入版本不匹配	没有引入资源包或者引入的资源包版本与当前 SDK 版本不匹配
11006	订单号不能为 0 或者超过 32 位	-
11007	nonce 字符串位数不为 32	-

WBFaceErrorDomainLoginNetwork

Code（错误码）	Description（描述）	Reason（详细实际原因）
21100	网络异常	登录时网络异常（请求未到达后台）
21200	网络异常	登录时后台返回参数有误（请求到达后台）

WBFaceErrorDomainLoginServer

Code（错误码）	Description（描述）	Reason（详细实际原因）

Code（错误码）	Description（描述）	Reason（详细实际原因）
其他错误码	透传后台错误码	例如签名问题等等

WBFaceErrorDomainGetInfo

Code（错误码）	Description（描述）	Reason（详细实际原因）
31100	网络异常	获取数字/活体类型/光线阈值，网络异常（请求未到达后台）
31200	网络异常	获取数字/活体类型/光线阈值，后台返回参数有误（请求到达后台）

WBFaceErrorDomainNativeProcess

Code（错误码）	Description（描述）	Reason（详细实际原因）
41000	用户取消	回到后台/单击 home/左上角/上传时左上角取消
41001	网络环境不满足认证需求	无网络/2G 网络
41002	权限异常，未获取权限	相机/麦克风/read phone/external/storage
41003	相机运行中出错	-
41004	视频录制中出错	不能存/启动失败/结束失败
41005	请勿晃动人脸,保持姿势	未获取到最佳图片
41006	视频大小不满足要求	视频大小不满足要求
41007	超时	预检测/动作活体
41008	检测中人脸移出框外	活体/数字/反光
41009	光线活体本地错误	-
41010	风险控制超出次数	用户重试太多次
41011	没有检测到读数声音	数字活体过程中没有发声

WBFaceErrorDomainCompareNetwork

Code（错误码）	Description（描述）	Reason（详细实际原因）
51100	网络异常	对比时，网络异常（请求未到达后台）
51200	网络异常	对比时，后台返回参数有误（请求到达后台）

WBFaceErrorDomainCompareServer

Code（错误码）	Description（描述）	Reason（详细实际原因）
其他错误码	透传后台错误码	-

接入示例

最近更新时间：2018-11-23 11:18:45

```
# 在MainActivity中单击某个按钮的代码逻辑：
//先填好数据
Bundle data = new Bundle();
WbCloudFaceVerifySdk.InputData inputData = new WbCloudFaceVerifySdk.InputData(
userName,
idType,
idNo,
agreementNo,
clientIp,
gps,
openApiAppId,
openApiAppVersion,
openApiNonce,
userId,
userSign,
verifyMode,
keyLicence);
data.putSerializable(WbCloudFaceContant.INPUT_DATA, inputData);

//个性化参数设置,可以不设置，不设置则为默认选项。
//此处均设置为与默认相反
//是否显示成功结果页，默认显示，此处设置为不显示
data.putBoolean(WbCloudFaceContant.SHOW_SUCCESS_PAGE, false);
//是否展示刷脸失败页面，默认展示,此处设置为不显示
data.putBoolean(WbCloudFaceContant.SHOW_FAIL_PAGE, false);
//sdk样式设置，默认为黑色
//此处设置为白色
data.putString(WbCloudFaceContant.COLOR_MODE, WbCloudFaceContant.WHITE);
//设置选择的比对类型 默认为权威库网纹图片对比
//权威库网纹图片比对 WbCloudFaceContant.ID_CRAD
//自带比对源比对 WbCloudFaceContant.SRC_IMG
//仅活体检测 WbCloudFaceContant.NONE
//此处设置自带比对源比对
data.putString(WbCloudFaceContant.COMPARE_TYPE, WbCloudFaceContant.SRC_IMG);
//上送自带的数据源信息，照片类型与照片string缺一不可
// 不带比对源的可不传这两个字段
//上送照片类型，1是网纹照 2是高清照
data.putString(WbCloudFaceContant.SRC_PHOTO_TYPE, srcPhotoType);
//比对源照片的BASE64 string
data.putString(WbCloudFaceContant.SRC_PHOTO_STRING, srcPhotoString);
//是否需要录制上传视频 默认需要，此处设置为需要
data.putBoolean(WbCloudFaceContant.VIDEO_UPLOAD, false);
```

```
//是否对录制视频进行检查,默认不检查，此处设置为检查
data.putBoolean(WbCloudFaceContant.VIDEO_CHECK, true);


//初始化sdk，得到是否登录sdk成功的结果
WbCloudFaceVerifySdk.getInstance().init(
MainActivity.this,
data,
//由WbCloudFaceVerifyLoginListener返回登录结果
new WbCloudFaceVerifyLoginListener() {
@Override
public void onLoginSuccess() {
//登录成功，拉起sdk页面
WbCloudFaceVerifySdk.getInstance().startWbFaceVerifySdk (MainActivity.this,
new WbCloudFaceVerifyResultListener() {
//由FaceVerifyResultListener返回刷脸结果
@Override
public void onFinish(WbFaceVerifyResult result) {
if(result != null) {
if (result.isSuccess()) {
Log.d(TAG, "刷脸成功！");
} else {
Log.d(TAG, "刷脸失败！);
}
}
});
}
@Override
public void onLoginFailed(WbFaceError error) {
//登录失败
...
}
```

iOS 动作/数字/光线活体 + 人脸比对 配置流程

最近更新时间：2019-01-15 19:05:08

配置流程(一)（IOS动作数字光线活体）- 使用Cocoapod集成

WeBankService SDK 最低支持到 iOS8.0，请在构建项目时候注意。

以下为接入配置的步骤：

1. 将 WBCloudReflectionFaceVerify 文件夹拷贝到自己项目的 podfile 文件所在的同一目录。
2. 在 podfile 使用如下配置：
   target 'WBFacePodDemo' do
   pod 'WBCloudReflectionFaceVerify', :path: './WBCloudReflectionFaceVerify'
   end
3. 使用 pod install 命令
4. SDK 需要使用相机、相册和录音权限，请在 info.plist 中添加：
   Privacy - Microphone Usage Description
   Privacy - Camera Usage Description

配置流程(二)（IOS动作数字光线活体）- 直接引用framework

WeBankService SDK 最低支持到 iOS8.0，请在构建项目时候注意。

以下为接入配置的步骤：

1. 引用以下资源文件到项目：

   WBCloudReflectionFaceVerify.framework

   faceLiveReflect.framework

   NextCV.framework

   rapidnet_ios.framework

   YTCommon.framework

   YTIllumination.framework

   YTPoseDetector.framework

   YTTrackPro.framework

   WBCloudReflectionFaceVerify.bundle

detector.bundle

ufa.bundle

PE.dat

2. SDK 依赖以下系统框架,需要在【BuildPhases】>【Link Binary With Libraries】中添加,可以参考 Demo, 具体依赖的系统库如下:

WebKit.framework

UIKit.framewrok

AVFoundation.framework

MobileCoreServices.framework

CoreVideo.framework

Accelerate.framework

Security.framework

SystemConfiguration.framework

CoreMedia.framework

AudioToolbox.framework

CoreTelephony.framework

ImageIO.framework

libc++.tbd

3. SDK 需要使用相机、相册和录音权限，请在 info.plist 中添加：

Privacy - Microphone Usage Description

Privacy - Camera Usage Description

4. 需要在 【BuildSettings】>【Other Linker Flags】中设置：-ObjC。

下一步：接口调用

接口调用

最近更新时间：2019-01-15 19:05:51

SDK 接口调用方法

SDK 的功能通过 `WBFaceVerifyCustomerService` 这个类的方法进行调用 ，**其中 SDK 中使用的 nonce，sign 等重要信息，需要合作方从自己后台拉取，并且两者不能缓存，只能使用一次即失效**，详细接口说明如下，其他的操作请参考 Demo 中的登录接口的参数说明 ：

```
#import <UIKit/UIKit.h>
#ifndef WBFaceVerifyConst_h
#define WBFaceVerifyConst_h
UIKIT_EXTERN NSString *const WBCloudFaceVerifySDKVersion;// 当前版本3.1.4
/**
具体的活体检测的类型:
- WBFaceVerifyLivingType_Action: 动作活体
- WBFaceVerifyLivingType_Number: 读4个数字活体
- WBFaceVerifyLivingType_Light: 光线活体
*/
typedef NS_ENUM(NSInteger,WBFaceVerifyLivingType){
WBFaceVerifyLivingType_Action,
WBFaceVerifyLivingType_Number,
WBFaceVerifyLivingType_Light,
};
/**
具体的服务类型:
- WBFaceVerifyCompareType_None: 进行活体检测服务, 不进行人脸比对服务
- WBFaceVerifyCompareType_SourceImage: 进行活体检测服务, 使用人脸比对服务: 对比源有合作方传入 SDK 中

- WBFaceVerifyCompareType_IDCard: 进行活体检测服务, 使用人脸比对服务: 对比源由(身份证+姓名)权威数据源照片进行比对
*/
typedef NS_ENUM(NSInteger, WBFaceVerifyCompareType){
WBFaceVerifyCompareType_None,
WBFaceVerifyCompareType_SourceImage,
WBFaceVerifyCompareType_IDCard,
};
/**
SDK使用的主题风格
- WBFaceVerifyThemeDarkness: 暗黑色系主题(当前只支持暗黑色系)
- WBFaceVerifyThemeLightness: 明亮色系主题
*/
typedef NS_ENUM(NSInteger, WBFaceVerifyTheme) {
```

```
WBFaceVerifyThemeDarkness,
WBFaceVerifyThemeLightness,
};
#endif /* WBFaceVerifyConst_h */
```

入口方法说明

（1）活体检测 + 人脸比对服务（身份证的网文照片进行对比）

```
/**
接口服务包含: 活体检测 + 人脸比对(身份证的网文照片进行对比)
@param userid 用户唯一标识, 由合作方自行定义（具体要求，参考word接入文档）
@param nonce 满足接入要求的32位随机数（具体要求，参考word接入文档）
@param sign 满足接入要求的40位签名值（具体要求，参考word接入文档）
@param appid 腾讯服务分配的appid
@param orderNo 每次人脸身份认证请求的唯一订单号: 建议为32位字符串(不超过32位)
@param apiVersion 后台api接口版本号(不是SDK的版本号),默认请填写@"1.0.0"
@param licence 腾讯给合作方派发的前端使用的licence(该licence同app当前使用的bundle id绑定)
@param facetype 人脸身份认证的类型: 请使用枚举传递: WBFaceVerifyLivingType_Action,WBFaceVerifyLiving
Type_Number,WBFaceVerifyLivingType_Light

@param userInfo 需要比对者的身份信息: 18位身份证号 + 姓名 + id类型
@param configure SDK基础配置项目
@param success 服务登录成功回调,登录成功以后开始进行(做动作/读数字/光线反射)活体动作检测
@param failure 服务登录失败回调,具体参考错误码文档(参考word接入文档)
*/
-(void)startLoginLiveCheckAndCompareServiceWithUserid:(NSString *)userid
nonce:(NSString *)nonce
sign:(NSString *)sign
appid:(NSString *)appid
orderNo:(NSString *)orderNo
apiVersion:(NSString *)apiVersion
licence:(NSString *)licence
faceverifyType:(WBFaceVerifyLivingType)facetype
userInfo:(WBFaceUserInfo *)userInfo
configure:(WBFaceVerifySDKConfig *)configure
success:(void (^)())success
failure:(void (^)(WBFaceError *error))failure;
```

（2）活体检测 + 人脸比对服务（合作方提供的比对源图片进行比对）

```
/**
接口服务包含: 活体检测 + 人脸比对(合作方提供的比对源图片进行比对)
@param userid 用户唯一标识, 由合作方自行定义（具体要求，参考word接入文档）
@param nonce 满足接入要求的32位随机数（具体要求，参考word接入文档）
@param sign 满足接入要求的40位签名值（具体要求，参考word接入文档）
@param appid 腾讯服务分配的appid
```

```
@param orderNo 每次人脸身份认证请求的唯一订单号: 建议为32位字符串(不超过32位)
@param apiVersion 后台api接口版本号(不是SDK的版本号),默认请填写@"1.0.0"
@param licence 腾讯给合作方派发的前端使用的licence(该licence同app当前使用的bundle id绑定)
@param facetype 人脸身份认证的类型: 请使用枚举传递 WBFaceVerifyLivingType_Action,WBFaceVerifyLiving
Type_Number,WBFaceVerifyLivingType_Light
@param compareSoureConfig 自带比对源接口配置项目
@param configure SDK基础配置项目
@param success 服务登录成功回调,登录成功以后开始进行(做动作/读数字/光线反射)活体动作检测
@param failure 服务登录失败回调,具体参考错误码文档(参考word接入文档)
*/
-(void)startLoginLiveCheckAndCompareServiceWithUserid:(NSString *)userid
nonce:(NSString *)nonce
sign:(NSString *)sign
appid:(NSString *)appid
orderNo:(NSString *)orderNo
apiVersion:(NSString *)apiVersion
licence:(NSString *)licence
faceverifyType:(WBFaceVerifyLivingType)facetype
compareSourceConfig:(WBFaceVerifyCompareSourceConfig *)compareSoureConfig
configure:(WBFaceVerifySDKConfig *)configure
success:(void (^)())success
failure:(void (^)(WBFaceError *error))failure;
```

（3）活体检测服务接口

```
/*
接口服务包含: 活体检测

@param userid 用户唯一标识, 由合作方自行定义（具体要求，参考word接入文档）
@param nonce 满足接入要求的32位随机数（具体要求，参考word接入文档）
@param sign 满足接入要求的40位签名值（具体要求，参考word接入文档）
@param appid 腾讯服务分配的appid
@param orderNo 每次人脸身份认证请求的唯一订单号: 建议为32位字符串(不超过32位)
@param apiVersion 后台api接口版本号(不是SDK的版本号),默认请填写@"1.0.0"
@param licence 腾讯给合作方派发的前端使用的licence(该licence同app当前使用的bundle id绑定)
@param facetype 人脸身份认证的类型: 请使用枚举传递 WBFaceVerifyLivingType_Action,WBFaceVerifyLiving
Type_Number,WBFaceVerifyLivingType_Light

@param configure SDK基础配置项目
@param success 服务登录成功回调,登录成功以后开始进行(做动作/读数字/光线反射)活体动作检测
@param failure 服务登录失败回调,具体参考错误码文档(参考word接入文档)
*/
-(void)startLoginLiveCheckServiceWithUserid:(NSString *)userid
nonce:(NSString *)nonce
sign:(NSString *)sign
appid:(NSString *)appid
orderNo:(NSString *)orderNo
```

```
apiVersion:(NSString *)apiVersion
licence:(NSString *)licence
faceverifyType:(WBFaceVerifyLivingType)facetype
configure:(WBFaceVerifySDKConfig *)configure
success:(void (^)())success
failure:(void (^)(WBFaceError *error))failure;
```

个性化参数设置

SDK 登录接口 startXXXX 方法中需要传入 WBFaceVerifySDKConfig 字段，通过该对象可以配置 SDK中其他基础配置：包括设置是否展示成功/结果页面，主题风格，资源路径等等，具体参考头文件。

```
/**
人脸识别SDK 基础配置类
*/
@interface WBFaceVerifySDKConfig : NSObject

/**
sdk中拉起人脸活体识别界面中使用UIWindow时的windowLevel配置,默认配置是1 + UIWindowLevelNormal

如果接入放app中有其他自定义UIWindow, 为了防止界面覆盖,可以酌情设置该参数
*/
@property (nonatomic, assign) NSUInteger windowLevel;

/**
人脸识别服务结果页是否展示配置项 - 是否展示人脸对比成功界面 -> 建议关闭
*/
@property (nonatomic, assign) BOOL showSuccessPage;

/**
人脸识别服务结果页是否展示配置项 - 是否展示人脸对比失败界面 -> 建议开启
*/
@property (nonatomic, assign) BOOL showFailurePage;

/**
人脸识别页面中的主题风格, 需要配合不同资源包使用:
WBFaceVerifyThemeDarkness - 暗灰主题
WBFaceVerifyThemeLightness - 明亮主题
*/
@property (nonatomic, assign) WBFaceVerifyTheme theme;

/**
资源包路径位置
*/
```

```
@property (nonatomic, copy) NSString *bundlePath;

/**
默认sdk配置接口
*/
+(instancetype)sdkConfig;
@end
```

上一步：配置流程

下一步：接入流程与说明

接入流程与说明

最近更新时间：2018-12-11 11:38:17

整体工作流程

人脸识别主要流程包括以下几个步骤:

（1) 登录人脸识别 SDK，即 startLoginWBFaceCompareServiceWithUserid:xxx 方法。

（2) 登录结果返回，在第一步登录成功以后，SDK 会自动拉起活体检测界面进行活体检测，进入（3）如果登录失败，则不会拉起活体检测页面，会通过 failure block 回调返回一个 WBFaceError 对象。

（3) 活体识别的具体过程，如果成功进入（4），如果失败进入（5）。

（4) 活体识别前端 SDK 检测结果上送后台进行活体检测以及人脸比对等服务。

（5) 服务结果返回,通过 delegate 回调方法或者注册 NSNotification 的方式获取活体检测/人脸比对的结果，结果封装在 WBFaceVerifyResult。

登录接口说明

iOS 人脸识别接口大部分参数说明请参考头文件注释。其中需要特别说明的是: userid, nonce,sign,orderNo 等参数建议通过接入方后台透传给前端，前端无需在本地生成。

方法功能一：

如果使用活体检测服务 + 人脸比对服务（身份证的网文照片进行对比）方式，则需要传入 WBFaceUserInfo 对象，该对象由需要识别对象的以下信息构成：

- 身份证 idNo - 仅支持18位身份证
- 姓名 name
- 证件类型 idType - 默认18位身份证填写 @"01"

方法功能二：

如果使用活体检测服务 + 人脸比对服务（合作方提供的比对源图片进行比对）。需要合作方传递自带比对源配置对象 WBFaceVerifyCompareSourceConfig，通过 compareSouceConfigWithImage: 创建该对象，并传递比对源图片 UIImage。

方法功能三：

如果使用活体检测服务，仅仅使用活体检测。通过活体检测服务判断是否是真人。

识别界面拉起方式说明

默认情况SDK会创建一个全屏幕大小的 UIWindow，覆盖在当前屏幕上方，并且默认windowLevel = UIWindowLevelNormal + 1.

识别结果回调方式说明

SDK 中，人脸识别结果返回给接入方有两种方式，两种结果回调都在 Main Thread 完成：

1. 通过实现 WBFaceVerifyCustomerServiceDelegate 的 WBFaceVerifyCustomerServiceDelegate 方法,通过该方法返回 WBFaceVerifyResult 对象。

2. 无需实现 delegate 方法，通过注册 WBFaceVerifyCustomerServiceDidFinishedNotification 通知，在接受到该通知时，进行结果处理。

   WBFaceVerifyCustomerServiceDidFinishedNotification 通知中，通过获取该通知的 userInfo，获取字典中 key 为 faceVerifyResult 对应的 value 对象就是 WBFaceVerifyResult。

WBFaceVerifyResult 说明

```
/**
人脸服务返回结果对象
*/
@interface WBFaceVerifyResult : NSObject
/**
人脸比对结果是否通过:
YES: 表示人脸服务通过
NO: 表示人脸服务不通过
*/
@property (nonatomic, assign, readonly) BOOL isSuccess;
@property (nonatomic, copy, readonly) NSString *orderNo;

/**
isSuccess == YES 时, sign有值
isSuccess == NO 时, sign 无意义
*/
@property (nonatomic, copy, readonly) NSString *sign;
/**
isSuccess == YES 时, liveRate 有值:
1. liveRate 可能是 @"分数为空", 这种情况具体咨询合作方
2. float类型的字符串, 请调用 [liveRate floatValue] 获取具体分数
```

```
isSuccess == NO 时, liveRate 无意义
*/
@property (nonatomic, copy, readonly) NSString * liveRate;

/**
isSuccess == YES 时, similarity 有值:
1. similarity 可能是 @"分数为空", 这种情况具体咨询合作方
2. float类型的字符串, 请调用 [similarity floatValue] 获取具体分数
isSuccess == NO 时, similarity 无意义
*/
@property (nonatomic, copy, readonly) NSString * similarity;

/**
isSuccess == YES 时候, error 无意义
isSuccess == NO 时, error中存储的具体错误信息,参考 WBFaceError.h
*/
@property (nonatomic, strong, readonly) WBFaceError *error;

+(instancetype)resultWith:(BOOL)isSuccess error:(WBFaceError *)error;
+(instancetype)resultWith:(BOOL)isSuccess error:(WBFaceError *)error sign:(NSString *)sign;
+(instancetype)resultWith:(BOOL)isSuccess error:(WBFaceError *)error sign:(NSString *)sign liveRate:(NSStr
ing *)liveRate similarity:(NSString *)similarity;

-(NSString *)description;
@end
```

上一步：接口调用

错误码描述

最近更新时间：2018-08-14 17:47:57

SDK 在登录以及返回人脸服务结果时，如果发生错误或者识别失败会返回 WBFaceError 对象，该对象的结构如下：

WBFaceErrorDomainInputParams

Code（错误码）	Description（描述）	Reason（详细实际原因）
12000	传入参数为空	传入的 xx 为空
12001	传入的 keyLicence 不可用	传入的 keyLicence 不可用
12002	身份证格式不正确	身份证格式不正确
12003	使用自带对比源，传入参数错误，非 base64	传入的 srcPhotoString 不是 base64
12004	使用自带对比源，传入参数错误，超过 3MB	传入的 srcPhotoString 超过 3MB
12005	SDK 资源引入版本不匹配	没有引入资源包或者引入的资源包版本与当前 SDK 版本不匹配
12006	订单号不能为 0 或者超过 32 位	
12007	nonce 字符串位数不为 32 位	

WBFaceErrorDomainLoginNetwork

Code（错误码）	Description（描述）	Reason（详细实际原因）
22100	网络异常	登录时网络异常（请求未到达后台）
22200	网络异常	登录时后台返回参数有误（请求到达后台）

WBFaceErrorDomainLoginServer

Code（错误码）	Description（描述）	Reason（详细实际原因）
其他错误码	透传后台错误码	例如签名问题等等

WBFaceErrorDomainGetInfo

Code（错误码）	Description（描述）	Reason（详细实际原因）

Code（错误码）	Description（描述）	Reason（详细实际原因）
32100	网络异常	获取数字/活体类型/光线阈值，网络异常(请求未到达后台)
32200	网络异常	获取数字/活体类型/光线阈值，后台返回参数有误（请求到达后台）

WBFaceErrorDomainNativeProcess

Code（错误码）	Description（描述）	Reason（详细实际原因）
42000	用户取消	回到后台/单击 home/左上角/上传时左上角取消
42001	网络环境不满足认证需求	无网络/2g网络
42002	权限异常，未获取权限	相机/麦克风/read phone/external/storage
42003	相机运行中出错	
42004	视频录制中出错	不能存/启动失败/结束失败
42005	请勿晃动人脸,保持姿势	未获取到最佳图片
42006	视频大小不满足要求	视频大小不满足要求
42007	超时	预检测/动作活体
42008	检测中人脸移出框外	活体/数字/反光
42009	光线活体本地错误	
42010	风险控制超出次数	用户重试太多次
42011	没有检测到读数声音	数字活体过程中没有发声

WBFaceErrorDomainCompareNetwork

Code（错误码）	Description（描述）	Reason（详细实际原因）
52100	网络异常	对比时，网络异常（请求未到达后台）
52200	网络异常	对比时，后台返回参数有误（请求到达后台）

WBFaceErrorDomainCompareServer

Code（错误码）	Description（描述）	Reason（详细实际原因）
其他错误码	透传后台错误码	

其他错误

```
*/
NS_ASSUME_NONNULL_BEGIN
/*
错误domain划分成两类:

出现在登录时, 通过调用startXXXX 方法的failure block进行回调返回:
WBFaceErrorDomainInputParams, WBFaceErrorDomainLoginNetwork, WBFaceErrorDomainLoginServer


人脸服务结果返回(封装在WBFaceVerifyResult中):
WBFaceErrorDomainGetInfo, WBFaceErrorDomainNativeProcess, WBFaceErrorDomainCompareNetwork,
WBFaceErrorDomainCompareServer
*/

/* 登录时传入参数有误 */
UIKIT_EXTERN NSString *const WBFaceErrorDomainInputParams;
/* 登录时网络请求错误 */
UIKIT_EXTERN NSString *const WBFaceErrorDomainLoginNetwork;
/* 登录时后台拒绝登录, 具体参考后台word版本错误码, 这里直接透传 */
UIKIT_EXTERN NSString *const WBFaceErrorDomainLoginServer;
/* 拉取有效信息时候网络错误 */
UIKIT_EXTERN NSString *const WBFaceErrorDomainGetInfo;
/* native本地在活体检测中, 某些原因导致错误 */
UIKIT_EXTERN NSString *const WBFaceErrorDomainNativeProcess;
/* 上送后台比对时,网络错误 */
UIKIT_EXTERN NSString *const WBFaceErrorDomainCompareNetwork;
/* 后台比对完成,返回比对结果的错误原因*/
UIKIT_EXTERN NSString *const WBFaceErrorDomainCompareServer;

@interface WBFaceError: NSObject
/**
错误发生的地方, 具体的发生地方由上面定义的 WBFaceErrorDomainXXXX 指定
*/
@property (nonatomic, readonly, copy) NSString *domain;
/**
每个domain中有相应的错误代码, 具体的错误代码见
*/
@property (nonatomic, readonly, assign) NSInteger code; //错误代码
@property (nonatomic, readonly, copy) NSString *desc; //获取本地化描述
@property (nonatomic, readonly, copy) NSString *reason; // 错误出现的真实原因原因
@property (nonatomic, readonly, copy) NSDictionary * _Nullable otherInfo;// 预留接口

+ (instancetype)errorWithDomain:(NSString *)domain code:(NSInteger)code desc:(NSString *)desc;
+ (instancetype)errorWithDomain:(NSString *)domain code:(NSInteger)code desc:(NSString *)desc reaso
n:(NSString *)reason;
```

```
+ (instancetype)errorWithDomain:(NSString *)domain code:(NSInteger)code desc:(NSString *)desc reaso
n:(NSString *)reason otherInfo:(nullable NSDictionary *)otherInfo;
@end
```

公众号接入
合作方后台上送身份信息

最近更新时间：2018-07-27 16:38:24

1. 生成签名

准备步骤

- **前置条件：请合作方确保 SIGN ticket 已经正常获取，获取方式见 SIGN ticket 获取 。**
- 合作方为人脸验证服务生成签名，需要具有以下参数：
- 合作方根据本次人脸验证的如下参数生成签名,需要签名的参数信息如下：
- 参与签名的数据需要和使用该签名的接口中的请求参数保持一致。

参数	说明	来源
appId	腾讯云线下对接分配的 App ID	腾讯云线下对接分配
orderNo	订单号，本次人脸验证合作伙伴上送的订单号，唯一标识	合作方自行分配
name	姓名	-
idNo	证件号码	-
userId	用户 ID ，用户的唯一标识（不要带有特殊字符）	合作方自行分配
version	1.0.0	-
api ticket	合作伙伴服务端缓存的 tikcet，注意是 SIGN 类型	获取方式见 SIGN ticket 获取

基本步骤

1. 将 appId, orderNo, name, idNo, userId, version, ticket（SIGN 类型）共七个参数的值进行字典序排序。
2. 将排序后的所有参数字符串拼接成一个字符串。
3. 将排序后的字符串进行 SHA1 编码，编码后的 40 位字符串作为签名（sign）。

> **注意：**
>
> 签名算法可参考 签名算法说明 。

参考示例

请求参数：

参数名	参数值
webankAppId	appId001
orderNo	orderNo19959248596551
name	testName
idNo	4300000000000
userId	userID19959248596551
version	1.0.0
ticket	duSz9ptwyW1Xn7r6gYltxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoe

字典排序后的参数为：

```
[1.0.0, 4300000000000, appId001, duSz9ptwyW1Xn7r6gYltxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoe, orderNo19959248596551, testName, userID19959248596551]
```

拼接后的字符串为：

```
1.0.04300000000000appId001duSz9ptwyW1Xn7r6gYltxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoeorderNo19959248596551testNameuserID19959248596551
```

计算 SHA1 得到签名：

```
EE57F7C1EDDE7B6BB0DFB54CD902836B8EB0575B
```

该字串就是最终生成的签名（40位），不区分大小写。

2. 合作方后台上送身份信息

请求

请求 URL：

```
https://idasc.webank.com/api/server/h5/geth5faceid
```

请求方法：POST
报文格式：Content-Type: application/json
请求参数：

参数	说明	类型	长度（字节）	是否必填
webankAppId	腾讯云线下对接分配的 App ID	String	腾讯云线下对接决定	是
orderNo	订单号，由合作方上送，每次唯一，不能超过 32 位	String	不能超过 32 位	是
name	姓名	String	-	是
idNo	证件号码	String	-	是
userId	用户 ID ，用户的唯一标识（不能带有特殊字符）	String	-	是
sourcePhotoStr	比对源照片，**注意：原始图片不能超过 500k，且必须为 JPG 或 PNG 格式。** **参数有值：**使用合作伙伴提供的比对源照片进行比对，必须注照片是正脸可信照片，照片质量由合作方保证。 **参数为空** ：根据身份证号 + 姓名使用权威数据源比对	BASE64String	1048576	否，非必填
sourcePhotoType	比对源照片类型，参数值为1 时是：水纹正脸照。参数值为 2 时是：高清正脸照	String	1	是
version	默认参数值为：1.0.0	String	20	是
sign	签名：使用上面生成的签名	String	40	是

响应

响应示例


```
{
"code": 0,
"msg": "成功",
"result": {
"bizSeqNo":"业务流水号" ,
"orderNo":"合作方订单号",
"h5faceId":"cc1184c3995c71a731357f9812aab988"
}
```


下一步：公众号启动 H5 人脸验证

公众号启动 H5 人脸验证

最近更新时间：2019-01-09 18:08:52

生成签名

准备步骤

- **前置条件：**请合作方确保 NONCE ticket 已经正常获取，获取方式见 NONCE ticket 获取 。
- 合作方根据本次人脸验证的如下参数生成签名，需要签名的参数信息如下：
- 参与签名的数据需要和使用该签名的接口中的请求参数保持一致。

参数	说明	来源
webankAppId	腾讯云线下对接分配的 AppID	腾讯云线下对接分配
orderNo	订单号，本次人脸验证合作伙伴上送的订单号，唯一标识	合作方自行分配
userId	用户 ID ，用户的唯一标识（不要带有特殊字符）	合作方自行分配（和接口中使用的 userId 持一致）
version	参数值为：1.0.0	-
h5faceId	h5/geth5faceid 接口返回的唯一标识	-
api ticket	合作伙伴服务端缓存的 tikcet，注意是 NONCE 类型	获取方式见 NONCE ticket 获取 所用的 userId 和接口里面定的userId参数值保持一致
nonce	随机数：32位随机串（字母+数字组成的随机数）	合作方自行生成（和接口中定义的随机数保持一致）

基本步骤

1. 生成一个32位的随机字符串（字母和数字）nonce（接口请求时也要用到）。
2. 将 webankAppId、userId、orderNo、version、h5faceId 连同 ticket、nonce 共7个参数的值进行字典序排序。
3. 将排序后的所有参数字符串拼接成一个字符串。
4. 将排序后的字符串进行 SHA1 编码，编码后的40位字符串作为签名（sign）。

> ⚠ **注意：**
>
> 签名算法可参考 签名算法说明 。

参考示例

请求参数：

参数名	参数值
webankAppId	appId001
userId	userID19959248596551
nonce	kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7T
version	1.0.0
h5faceId	bwiwe1457895464
orderNo	aabc1457895464
ticket	zxc9QfxIti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS

字典排序后的参数为：

[1.0.0, aabc1457895464, appId001, bwiwe1457895464, kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7T, userID19959248596551, zxc9QfxIti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS]

拼接后的字符串为：

1.0.0aabc1457895464appId001bwiwe1457895464kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7TuserID19959248596551zxc9QfxIti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS

计算 SHA1 得到签名：

该字串就是最终生成的签名（40位），不区分大小写。

4E9DFABF938BF37BDB7A7DC25CCA1233D12D986B

启动 H5 人脸验证

合作方公众号上送 `h5faceId` 以及 `sign` ，后台校验 `sign` 通过之后重定向到 H5 人脸验证。

请求

请求 URL： `https://ida.webank.com/api/h5/login`

请求方法：GET

请求参数：

参数	说明	类型	长度	必填
webankAppId	腾讯云线下对接分配的 AppID	String	腾讯云线下对接决定	是
version	接口版本号，默认参数值：1.0.0	String	20	是
nonce	随机数：32位随机串（字母+数字组成的随机数）	String	32	是
orderNo	订单号，由合作方上送，每次唯一，此信息为本次人脸验证上送的信息，不能超过32位	String	32	是
h5faceId	h5/geth5faceid 接口返回的唯一标识	String	32	是
url	H5 人脸验证完成后回调的第三方 URL，需要第三方提供完整 URL 且做 URL Encode 完整 URL Encode 示例 原 URL： `https://idaop.webank.com` Encode 后： `http://idaop.webank.com`	String	-	是
resultType	是否显示结果页面 参数值为“1”：直接跳转到 url 回调地址 null 或其他值：跳转提供的结果页面	String	-	否
userId	用户 ID ，用户的唯一标识（不要带有特殊字符）	String	-	是
sign	签名：使用上面生成的签名。	String	40	是
liveType	活体模式： 1：使用数字活体 2：使用动作活体 缺省：默认配置	String	-	否
redirectType	跳转模式： 参数值为“1”：刷脸页面使用 replace 方式跳转，不在浏览器 history 中留下记录。 不传或其他值：正常跳转。	String	-	否

H5 人脸验证结果跳转

最近更新时间：2018-07-27 16:49:14

用于实现人脸验证 H5 结果返回跳转第三方 URL 带唯一标识、订单号、验证结果、签名。

请求

请求 URL:

https://xxx.com/xxx?code=xxxx&orderNo=xxxx&h5faceId=xxxx&newSignature=xxxx

注意：

1. xxxx.com 为合作方上送的 URL。
2. 合作方根据 方式一：前端获取结果验证签名 或者 方式二：服务端查询结果 说明进行签名校验，确保返回结果的安全性。

请求方法：GET

响应

响应参数：

参数	说明	类型	长度（字节）
code	人脸验证结果的返回码，0 表示人脸验证成功，其他错误码标识失败。	字符串	
orderNo	订单号 ，由合作方上送，每次唯一，此信息为本次人脸验证上送的信息。	字符串	32
h5faceId	本次请求返回的唯一标识，此信息为本次人脸验证上送的信息。	字符串	32
newSignature	对 URL 参数 App ID、oderNo 和 SIGN ticket、code 的签名。具体见 【 方式二：服务端查询结果】>【生成签名】	字符串	40

上一步：公众号启动 H5 人脸验证

H5 接入
合作方后台上送身份信息

最近更新时间：2018-07-27 16:48:06

1. 生成签名

准备步骤

- **前置条件：请合作方确保 SIGN ticket 已经正常获取，获取方式见 SIGN ticket 获取 。**
- 合作方根据本次人脸验证的如下参数生成签名,需要签名的参数信息如下：
- 参与签名的数据需要和使用该签名的接口中的请求参数保持一致。

参数	说明	来源
appId	腾讯云线下对接分配的 App ID	腾讯云线下对接分配
orderNo	订单号，本次人脸验证合作伙伴上送的订单号，唯一标识	合作方自行分配
name	姓名	-
idNo	证件号码	-
userId	用户 ID ，用户的唯一标识（不要带有特殊字符）	合作方自行分配
version	1.0.0	-
api ticket	合作伙伴服务端缓存的 tikcet，注意是 SIGN 类型	获取方式见 SIGN ticket 获取

基本步骤

1. 将 appId, orderNo, name, idNo, userId, version, ticket（SIGN 类型）共七个参数的值进行字典序排序。
2. 将排序后的所有参数字符串拼接成一个字符串。
3. 将排序后的字符串进行 SHA1 编码，编码后的 40 位字符串作为签名（sign）。

> **注意：**
>
> 签名算法可参考 签名算法说明 。

参考示例

请求参数：

参数名	参数值
webankAppId	appId001
orderNo	orderNo19959248596551
name	testName
idNo	4300000000000
userId	userID19959248596551
version	1.0.0
ticket	duSz9ptwyW1Xn7r6gYItxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoe

字典排序后的参数为：

```
[1.0.0, 4300000000000, appId001, duSz9ptwyW1Xn7r6gYItxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoe, orderNo19959248596551, testName, userID19959248596551]
```

拼接后的字符串为：

```
1.0.043000000000000appId001duSz9ptwyW1Xn7r6gYItxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoeorderNo19959248596551testNameuserID19959248596551
```

计算 SHA1 得到签名：

```
EE57F7C1EDDE7B6BB0DFB54CD902836B8EB0575B
```

该字串就是最终生成的签名（40位），不区分大小写。

2. 合作方后台上送身份信息

请求

请求 URL：

```
https://idasc.webank.com/api/server/h5/geth5faceid
```

请求方法：POST
报文格式：Content-Type: application/json
请求参数：

参数	说明	类型	长度（字节）	是否必填
webankAppId	腾讯云线下对接分配的 App ID	String	腾讯云线下对接决定	是
orderNo	订单号，由合作方上送，每次唯一，不能超过 32 位	String	不能超过 32 位	是
name	姓名	String	-	是
idNo	证件号码	String	-	是
userId	用户 ID ，用户的唯一标识（不能带有特殊字符）	String	-	是
sourcePhotoStr	比对源照片，**注意：原始图片不能超过 500k，且必须为 JPG 或 PNG 格式。** **参数有值：**使用合作伙伴提供的比对源照片进行比对，必须注照片是正脸可信照片，照片质量由合作方保证。 **参数为空** ：根据身份证号 + 姓名使用权威数据源比对	BASE64String	1048576	否，非必填
sourcePhotoType	比对源照片类型，参数值为1 时是：水纹正脸照。参数值为 2 时是：高清正脸照	String	1	是
version	默认参数值为：1.0.0	String	20	是
sign	签名：使用上面生成的签名	String	40	是

响应

响应示例


```
{
"code": 0,
"msg": "成功",
"result": {
"bizSeqNo":"业务流水号" ,
"orderNo":"合作方订单号",
"h5faceId":"cc1184c3995c71a731357f9812aab988"
}
```


下一步：启动 H5 人脸验证

启动 H5 人脸验证

最近更新时间：2018-07-27 16:52:05

前置条件

合作方如果使用 App 内调起 H5 人脸验证，需要 App 平台的 webkit/blink 等组件支持调用摄像头录视频，方可正常使用人脸验证功能。

> **注意：**
> App 内调用 H5 人脸验证或者短信链接调用 H5 人脸验证等场景，若当前设备无法正常调用摄像头进行视频录制，前端将返回特定的错误码（如下）告知合作方，合作方拿到错误码后可选择用备用方案核身处理。

返回码	返回信息	处理措施
3001	该浏览器不支持视频录制	请使用其它验证方案

1. 生成签名

准备步骤

- **前置条件：请合作方确保 NONCE ticket 已经正常获取，获取方式见 NONCE ticket 获取 。**
- 合作方根据本次人脸验证的如下参数生成签名,需要签名的参数信息如下：
- 参与签名的数据需要和使用该签名的接口中的请求参数保持一致。

参数	说明	来源
webankAppId	腾讯云线下对接分配的 App ID	腾讯云线下对接分配
orderNo	订单号，本次人脸验证合作伙伴上送的订单号，唯一标识	合作方自行分配
userId	用户 ID ，用户的唯一标识（不要带有特殊字符）	合作方自行分配
version	参数值为：1.0.0	-
h5faceId	h5/geth5faceid 接口返回的唯一标识	-

参数	说明	来源
api ticket	合作伙伴服务端缓存的 tikcet，注意是 NONCE 类型	获取方式见 NONCE ticket 获取（所用的userid参数值需要和接口里面的 userId 值保持一致）
nonce	随机数：32 位随机串（字母 + 数字组成的随机数）	合作方自行生成（与接口中的随机数保持一致）

基本步骤

1. 生成一个 32 位的随机字符串（字母和数字）nonce（接口请求时也要用到）。
2. 将 webankAppId、userId、orderNo、version、h5faceId 连同 ticket、nonce 共 7 个参数的值进行字典序排序。
3. 将排序后的所有参数字符串拼接成一个字符串。
4. 将排序后的字符串进行 SHA1 编码，编码后的 40 位字符串作为签名（sign）。

> **注意：**
>
> 签名算法可参考 签名算法说明 。

参考示例

请求参数：

参数名	参数值
webankAppId	appId001
userId	userID19959248596551
nonce	kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7T
version	1.0.0
h5faceId	bwiwe1457895464
orderNo	aabc1457895464
ticket	zxc9QfxIti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS

字典排序后的参数为：

[1.0.0, aabc1457895464, appId001, bwiwe1457895464, kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7T, userID19959248596551, zxc9QfxIti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS]

拼接后的字符串为：

```
1.0.0aabc1457895464appId001bwiwe1457895464kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7TuserID19959248596551zxc9QfxIti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS
```

计算 SHA1 得到签名：

```
4E9DFABF938BF37BDB7A7DC25CCA1233D12D986B
```

该字串就是最终生成的签名（40 位），不区分大小写。

2. 启动 H5 人脸验证

合作方公众号上送 h5faceId 以及 sign，后台校验 sign 通过之后重定向到 H5 人脸验证。

请求 URL：

```
https://ida.webank.com/api/web/login
```

请求方法：GET

请求参数：

参数	说明	类型	长度	是否必填
webankAppId	腾讯云线下对接分配的 App ID	String	腾讯云线下对接决定	是
version	接口版本号，默认参数值：1.0.0	String	20	是
nonce	随机数：32 位随机串（字母 + 数字组成的随机数）	String	32	是
orderNo	订单号，由合作方上送，每次唯一，此信息为本次人脸验证上送的信息，不能超过 32 位	String	32	是
h5faceId	h5/geth5faceid 接口返回的唯一标识。	String	32	是
url	H5 人脸验证完成后回调的第三方 URL，**需要第三方提供完整 URL 且做 URL Encode。** **完整 URL Encode 示例:** **原 URL(https://idaop.webank.com)** **Encode 后：** **(http://idaop.webank.com)**	String	-	是
resultType	是否显示结果页面，参数值为"1"时直接跳转到 url 回调地址，null 或其他值跳转提供的结果页面	String	-	否，非必填
userId	用户 ID ，用户的唯一标识（不要带有特殊字符）	String	-	是

参数	说明	类型	长度	是否必填
sign	签名：使用上面生成的签名。	String	40	是
from	browser ：表示在浏览器启动刷脸 app ：表示在 app 里启动刷脸 默认值为 app	String	-	是

上一步：合作方后台上送身份信息

下一步：H5 人脸验证结果跳转

H5 人脸验证结果跳转

最近更新时间：2018-07-27 16:52:38

用于实现人脸验证 H5 结果返回跳转第三方 URL 带唯一标识、订单号、验证结果、签名。

请求

请求 URL:

https://xxx.com/xxx?code=xxxx&orderNo=xxxx&h5faceId=xxxx&newSignature=xxxx

注意：

1. xxxx.com 为合作方上送的 URL。
2. 合作方根据 方式一：前端获取结果验证签名 或者 方式二：服务端查询结果 说明进行签名校验，确保返回结果的安全性。

请求方法：GET

响应

响应参数：

参数	说明	类型	长度（字节）
code	人脸验证结果的返回码，0 表示人脸验证成功，其他错误码标识失败。	字符串	
orderNo	订单号 ，由合作方上送，每次唯一，此信息为本次人脸验证上送的信息。	字符串	32
h5faceId	本次请求返回的唯一标识，此信息为本次人脸验证上送的信息。	字符串	32
newSignature	对 URL 参数 App ID、oderNo 和 SIGN ticket、code 的签名。具体见 【 方式二：服务端查询结果】>【生成签名】	字符串	40

注意：

若 H5 浏览器不支持调用摄像头录制视频，则直接返回错误 code（详见 启动 H5 人脸验证 的前置条件）。

上一步：启动 H5 人脸验证

人脸核身 App 调用 H5 兼容性配置指引

最近更新时间：2019-01-11 16:26:04

概述

1. 请按照下文 **兼容性配置** 指引进行 iOS 及安卓手机的兼容性适配。
2. 兼容性验证：从如下手机列表中选择至少 10 款手机进行兼容性验证，需覆盖安卓系统版本 4.0 - 8.0。

兼容性配置

iOS 接入

iPhone 的兼容性适配，需在配置里加上摄像头和麦克风的使用权限。App 的 info.plist 中加入：

```
.NSMicrophoneUsageDescription
.NSCameraUsageDescription
```

Android 接入

由于 Android 机器碎片化严重，用系统 WebView 调起系统摄像头完成视频录制可能存在很多兼容性问题，如部分机器出现调不起摄像头、调起摄像头无法录制视频等。因此整理了接入指引。
请合作方 **务必** 按照如下步骤顺序，实现兼容性处理：

1. 引入工具类
   将 WBH5FaceVerifySDK.java 文件拷贝到项目中。

2. 申请权限
   a. 在 Manifest.xml 文件中增加申请以下权限

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

b. 动态申请权限

- 如果第三方编译的 targetSdkVersion >= 23，则需要动态申请权限。
- 如果第三方编译的 targetSdkVersion < 23，则不需要动态申请权限。

3. WebSettings 的设置

调用 WebView.loadUrl(String url) 前一行添加如下代码设置 WebSettings。

```
/**
* 对 WebSettings 进行设置：添加 ua 字段和适配 h5 页面布局等
* @param mWebView 第三方的 WebView 对象
* @param context 第三方上下文
*/
WBH5FaceVerifySDK.getInstance().setWebViewSettings(mWebView,getApplicationContext());
```

4. WebChromeClient 的重写

调用 WebView.loadUrl(String url) 前，WebView 必须调用 setWebChormeClient(WebChromeClient webChormeClient)，并重写 WebChromeClient 的如下三个函数：

```
/**
* android端接收H5端发来的请求
For Android >= 3.0
*/
public void openFileChooser(ValueCallback<Uri> uploadMsg, String acceptType) {
if(WBH5FaceVerifySDK.getInstance().recordVideoForApiBelow21(uploadMsg, acceptType,activity))
return;
// TODO: 第三方有调用系统相机处理其他业务的话，将相关逻辑代码放在下面

}
/**
* android端接收H5端发来的请求
For Android >= 4.1
*/
public void openFileChooser(ValueCallback<Uri> uploadMsg, String acceptType, String capture) {
if(WBH5FaceVerifySDK.getInstance().recordVideoForApiBelow21(uploadMsg, acceptType,activity))
return;
// TODO: 第三方有调用系统相机处理其他业务的话，将相关逻辑代码放在下面

}
/**
* android端接收H5端发来的请求
For Lollipop 5.0+ Devices
*/
@Override
public boolean onShowFileChooser(WebView webView, ValueCallback<Uri[]> filePathCallback, FileCho
```

```
oserParams fileChooserParams) {
if(WBH5FaceVerifySDK.getInstance().recordVideoForApi21(webView, filePathCallback,activity, fileChoos
erParams)){
return true;
}
// TODO: 第三方有调用系统相机处理其他业务的话，将相关逻辑代码放在下面

return true;
}
```

注意：

- 如果第三方已重写以上函数，只要将如上述所示的函数体内容添加至第三方的对应函数体首行即可
- 如果第三方没有重写以上函数，则直接按照上述所示重写。

5. Activity 的重写

WebView 所属的 Activity 必须重写如下函数：

```
/**
*返回到WebView所属的Activity的回调
*/
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
super.onActivityResult(requestCode, resultCode, data);
if (WBH5FaceVerifySDK.getInstance().receiveH5FaceVerifyResult(requestCode,resultCode,data))
return;
// TODO: 第三方有其他请求的返回结果要处理的话，将相关逻辑代码放在下面

}
```

注意：

- 如果第三方 WebView 所属的 Activity 已重写以上函数，则将如上图所示的函数体内容添加至第三方的对应函数体首行即可。
- 如果第三方 WebView 所属的 Activity 没有重写以上函数，则直接按照上图所示重写。

兼容性验证

为了验证合作方 Android 端已经兼容了摄像头的处理，建议合作方做如下兼容性测试。

- 从下面的机型列表中选择覆盖 Android4.0 - 8.0 版本号的机型进行测试。
- 每个系统版本号选取至少 2 款不同手机。

手机品牌	手机型号	安卓系统版本
华为	Mate9pro	8.0.0
华为	Mate10	8.0.0
华为	华为荣耀 8	7.0.0
华为	华为 P9 Plus	7.0.0
华为	华为 Nexus 6P	6.0.1
华为	华为 Mate 7	6.0.0
华为	华为荣耀 7i	5.1.1
华为	P8 标配双 4G 版	5.0.1
华为	Mate 7 青春版	4.4.4
华为	华为 G660-L075	4.3.0
OPPO	OPPO R11 Plus	7.1.1
OPPO	OPPO R11	7.1.1
OPPO	OPPO R9s Plus	6.0.1
OPPO	OPPO A57	6.0.1
OPPO	OPPO A53	5.1.1
OPPO	OPPO R7 Plus	5.0.0
OPPO	OPPO R7	4.4.4
OPPO	OPPO R815T	4.2.1
vivo	vivo X20A	7.1.1
vivo	Xplay6	7.1.0
vivo	vivo Y55A	6.0.1
vivo	X9Plus	6.0.0
vivo	vivo Xplay5A	5.1.1

手机品牌	手机型号	安卓系统版本
vivo	vivo Y35A	5.0.2
vivo	vivo X5	4.4.4
vivo	vivo Y613	4.2.2
小米	小米 MI 6	8.0.0
小米	MiNote 3	7.1.1
小米	MIX 2	7.1.0
小米	小米 MI 4	6.0.1
小米	MI 4W	6.0.0
小米	小米 4C	5.1.1
小米	MI NOTE Pro	5.0.2
小米	MI 3W	4.4.4
小米	MI 2	4.1.1
三星	三星 Note 8	7.1.1
三星	三星 S8	7.0.0
三星	GALAXY A8	6.0.1
三星	GALAXY S5	6.0.1
三星	GALAXY On7(全网通)	5.1.1
三星	GALAXY S4（I9508）	5.0.0
三星	GALAXY A3	4.4.4
三星	SM-G3819D	4.1.2
魅族	魅族 PRO 6s	7.1.1
魅族	魅族 PRO 6 Plus	7.0.0
魅族	U20	6.0.0
魅族	M5	6.0.0
魅族	魅蓝 Note	5.1.0

手机品牌	手机型号	安卓系统版本
魅族	MX5	5.1.0
魅族	魅蓝	4.4.4
魅族	魅族 MX3	4.2.1
酷派	大神 Note3	5.1.0
酷派	锋尚 Pro	4.4.4
酷派	酷派 8729	4.3.0
红米	小米 Redmi 4X	7.1.2
红米	Redmi Note 5A	7.1.0
红米	Redmi 4A	6.0.1
红米	Redmi Note 4X	6.0.0
红米	红米 3	5.1.1
红米	红米 Note 3	5.0.2
红米	红米 note	4.4.4
红米	红米 1S	4.4.0
HTC	One A9	7.0.0
HTC	10（国际版/双 4G）	6.0.1
HTC	HTC A9	6.0.0
HTC	HTC D728w	5.1.0
HTC	HTC M9w	5.0.2
HTC	Desire 820s	4.4.4
HTC	HTC X920e	4.1.1

小程序接入
合作方后台上送身份信息

最近更新时间：2018-08-13 17:07:50

1. 生成签名

准备步骤

- **前置条件：请合作方确保 SIGN ticket 已经正常获取，获取方式见 SIGN ticket 获取 。**
- 合作方根据本次人脸验证的如下参数生成签名,需要签名的参数信息如下：
- 参与签名的数据需要和使用该签名的接口中的请求参数保持一致。

参数	说明	来源
appId	腾讯云线下对接分配的 App ID	腾讯云线下对接分配
orderNo	订单号，本次人脸验证合作伙伴上送的订单号，唯一标识	合作方自行分配
name	姓名	-
idNo	证件号码	-
userId	用户 ID ，用户的唯一标识（不要带有特殊字符）	合作方自行分配（与接口中的 userId 保持一致）
version	1.0.0	-
api ticket	合作伙伴服务端缓存的 tikcet，注意是 SIGN 类型	获取方式见 SIGN ticket 获取

基本步骤

1. 将 appId, orderNo, name, idNo, userId, version, ticket（SIGN 类型）共七个参数的值进行字典序排序。
2. 将排序后的所有参数字符串拼接成一个字符串。
3. 将排序后的字符串进行 SHA1 编码，编码后的 40 位字符串作为签名（sign）。

> **注意：**
>
> 签名算法可参考 签名算法说明 。

参考示例

请求参数：

参数名	参数值
webankAppId	appId001
orderNo	orderNo19959248596551
name	testName
idNo	4300000000000
userId	userID19959248596551
version	1.0.0
ticket	duSz9ptwyW1Xn7r6gYItxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoe

字典排序后的参数为：

```
[1.0.0, 4300000000000, appId001, duSz9ptwyW1Xn7r6gYItxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoe, orderNo19959248596551, testName, userID19959248596551]
```

拼接后的字符串为：

```
1.0.04300000000000appId001duSz9ptwyW1Xn7r6gYItxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoeorderNo19959248596551testNameuserID19959248596551
```

计算 SHA1 得到签名：

```
EE57F7C1EDDE7B6BB0DFB54CD902836B8EB0575B
```

该字串就是最终生成的签名（40 位），不区分大小写。

2. 合作方后台上送身份信息

请求

请求 URL：

```
https://idasc.webank.com/api/server/h5/geth5faceid
```

请求方法：`POST`
报文格式：`Content-Type: application/json`

请求参数：

参数	说明	类型	长度（字节）	是否必填
webankAppId	腾讯云线下对接分配的 App ID	String	腾讯云线下对接决定	是
orderNo	订单号，由合作方上送，每次唯一，不能超过 32 位	String	不能超过 32 位	是
name	姓名	String	-	是
idNo	证件号码	String	-	是
userId	用户 ID ，用户的唯一标识（不能带有特殊字符）	String	-	是
sourcePhotoStr	比对源照片，**注意：原始图片不能超过 500k，且必须为 JPG 或 PNG 格式。** **参数有值：**使用合作伙伴提供的比对源照片进行比对，必须注照片是正脸可信照片，照片质量由合作方保证。 **参数为空** ：根据身份证号 + 姓名使用权威数据源比对	BASE64String	1048576	否，非必填
sourcePhotoType	比对源照片类型，参数值为1 时是：水纹正脸照。参数值为 2 时是：高清正脸照	String	1	是
version	默认参数值为：1.0.0	String	20	是
sign	签名：使用上面生成的签名	String	40	是

响应

响应示例：

```
{
"code": 0,
"msg": "成功",
"result": {
"bizSeqNo":"业务流水号" ,
"orderNo":"合作方订单号",
"h5faceId":"cc1184c3995c71a731357f9812aab988"
}
}
```

下一步：启动小程序人脸验证

启动小程序人脸验证

最近更新时间：2018-08-13 17:07:47

1. 生成签名

准备步骤

- **前置条件：请合作方确保 NONCE ticket 已经正常获取，获取方式见 NONCE ticket 获取 。**
- 合作方根据本次人脸验证的如下参数生成签名,需要签名的参数信息如下：
- 参与签名的数据需要和使用该签名的接口中的请求参数保持一致

参数	说明	来源
webankAppId	腾讯云线下对接分配的 App ID	腾讯云线下对接分配
orderNo	订单号，本次人脸验证合作伙伴上送的订单号，唯一标识	合作方自行分配
userId	用户 ID ，用户的唯一标识（不要带有特殊字符）	合作方自行分配（与接口中使用的 userId 一致）
version	参数值为：1.0.0	
h5faceId	h5/geth5faceid 接口返回的唯一标识	
api ticket	合作伙伴服务端缓存的 tikcet，注意是 NONCE 类型	获取方式见 NONCE ticket 获取（所用的 userId 参数值需要和接口里面的 userId 值保持一致）
nonce	随机数：32 位随机串（字母 + 数字组成的随机数）	合作方自行生成（与接口中的随机数保持一致）

基本步骤

1. 生成一个 32 位的随机字符串（字母和数字） nonce（接口请求时也要用到）。
2. 将 webankAppId、userId、orderNo、version、h5faceId 连同 ticket、nonce 共 7 个参数的值进行字典序排序。
3. 将排序后的所有参数字符串拼接成一个字符串。
4. 将排序后的字符串进行 SHA1 编码，编码后的 40 位字符串作为签名（sign）。

> **注意：**
>
> 签名算法可参考 签名算法说明 。

参考示例

请求参数：

参数名	参数值
webankAppId	appId001
userId	userID19959248596551
nonce	kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7T
version	1.0.0
h5faceId	bwiwe1457895464
orderNo	aabc1457895464
ticket	zxc9QfxIti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS

字典排序后的参数为：

```
[1.0.0, aabc1457895464, appId001, bwiwe1457895464, kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7T, userID19959248596551, zxc9QfxIti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS]
```

拼接后的字符串为：

```
1.0.0aabc1457895464appId001bwiwe1457895464kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7TuserID19959248596551zxc9QfxIti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS
```

计算 SHA1 得到签名：

```
4E9DFABF938BF37BDB7A7DC25CCA1233D12D986B
```

该字串就是最终生成的签名（40 位），不区分大小写。

2. 启动小程序

合作方小程序上送 h5faceId 以及 sign ，调用微信打开小程序 API 启动人脸验证小程序。

请求方法： 使用 navigator 组件打开，navigatot 使用方法示例：

```
<navigator target="miniProgram" open-type="navigate" app-id="" path="" extra-data="" version="release">打开绑定的小程序</navigator>
```

此方法是微信小程序提供的 API，详细信息请参考 微信小程序官方文档 。
说明：

1. navigator 组件无回调函数，无法得知跳转结果。且该组件只能触发跳转事件，在跳转前应先获取到 sign 等参数。
2. 由于 sign 存在有效期，若跳转到刷脸小程序后用户长时间未进行操作，合作方小程序应再次获取 sign 后执行跳转（每次调起刷脸小程序都应获取一次）。
3. 由于 navigator 组件目前无回调函数，所以执行跳转后合作方小程序应清除当前缓存，保证异常退出后用户可以再次获取 sign 后跳转。

注意：

- 由于微信修改了小程序调起第三方小程序方式，老的方式 wx.navigateToMiniProgram(OBJECT) 将于 7 月 5 日废弃，届时在基础库 2.0.7 以上（对应微信 6.6.6）的微信将无法使用此接口调起小程序。
- 由于 navigator 组件仅在基础库 2.0.7（对应微信 6.6.6）以上可用，合作方跳转的逻辑仍需兼容旧方式，即低版本的微信。可以使用 wx.getSystemInfo(OBJECT) 获取到当前微信的基础库版本，判断其小于 2.0.7 版本使用旧的跳转方式。
  详细信息请参考 wx.getSystemInfo(OBJECT) 文档。

请求参数：

参数	类型	默认值	说明
target	String		必填，在哪个目标上发生跳转，默认当前小程序
open-type	String	navigate	必填，跳转方式
app-id	String	wx7ccfa42a2a641035	必填，要打开的小程序 AppId
path	String	pages/pre	必填，打开小程序的页面路径，如果为空则打开首页
extra-data	Object		必填，需要传递给目标小程序的数据，目标小程序可在 App.onLaunch()，App.onShow() 中获取到这份数据
version	String	release	必填，要打开的小程序版本，有效值 develop（开发版），trial（体验版），release（正式版），仅在当前小程序为开发版或体验版时此参数有效；如果当前小程序是体验版或正式版，则打开的小程序必定是正式版。

extra-data 中需要传递的参数：

参数	说明	类型	长度	是否必填

参数	说明	类型	长度	是否必填
webankAppId	WebankAppId，由腾讯指定	String	由腾讯指定腾讯服务分配	是
version	接口版本号	String	20	是，默认值：1.0.0
nonce	随机数 32 位随机串（字母+数字组成的随机数）	String	32	是
orderNo	订单号，由合作方上送，每次唯一，此信息为本次人脸验证上送的信息	字符串	32	是
h5faceId	h5 / geth5faceid 接口返回的唯一标识	String	32	是
resultType	是否显示结果页面（值为“1”直接跳转回第三方的小程序，null 或其他值跳转刷脸小程序提供的结果页面）	String	-	否，默认 0
userId	用户 ID ，用户的唯一标识（不要带有特殊字符）	String	-	是
sign	签名：使用上面生成的签名	String	40	是

上一步：合作方后台上送身份信息

下一步：小程序人脸验证结果跳转

小程序人脸验证结果跳转

最近更新时间：2018-08-13 17:07:44

1. 刷脸小程序完成刷脸流程后，会携带唯一标识、订单号、验证结果、签名跳转回第三方小程序。
2. 第三方小程序需在【小程序生命周期函数】>【监听小程序】显示 onShow 方法中监听结果返回，获得刷脸结果。

> **注意：**
>
> i. 第三方小程序在 onShow(options) 中可以通过 options.referrerInfo.extraData 拿到返回的结果参数。此部分为微信小程序提供的 API，详情可以参考 微信小程序官方文档。
> ii. 合作方根据进行签名校验，确保返回结果的安全性。

3. **返回结果参数：**

参数	说明	类型	长度（字节）
code	人脸验证结果的返回码，0 表示人脸验证成功，其他错误码表示失败，具体错误码详见 通用响应码列表 章节	字符串	
orderNo	订单号，由合作方上送，每次唯一，此信息为本次人脸验证上送的信息	字符串	32
h5faceId	本次请求返回的唯一标识，此信息为本次人脸验证上送的信息	字符串	32
newSignature	对 URL 参数 App ID、oderNo 和 SIGN ticket 、code的签名。具体见 签名生成 和校验规则	字符串	40

上一步：启动小程序人脸验证

PC 端 H5 接入
合作方后台上送身份信息

最近更新时间：2019-01-22 10:28:08

生成签名

准备步骤

- **前置条件：**请合作方确保 SIGN ticket 已经正常获取，获取方式见 SIGN ticket 获取 。
- 合作方为人脸验证服务生成签名，需要具有下表中的参数：
- 参与签名的数据需要和使用该签名的接口中的请求参数保持一致。

参数	说明	来源
appId	腾讯云线下对接分配的 AppID	腾讯云线下对接分配
orderNo	订单号，本次人脸验证合作伙伴上送的订单号，唯一标识	合作方自行分配
name	姓名	-
idNo	证件号码	-
userId	用户 ID ，用户的唯一标识（不要带有特殊字符）	合作方自行分配
version	1.0.0	-
api ticket	合作伙伴服务端缓存的 tikcet，注意是 SIGN 类型	获取方式见 SIGN ticket 获取

基本步骤

1. 将 appId、orderNo、name、idNo、userId、version、api ticket（SIGN 类型）共7个参数的值进行字典序排序。
2. 将排序后的所有参数字符串拼接成一个字符串。
3. 将排序后的字符串进行 SHA1 编码，编码后的40位字符串作为签名（sign）。

> ⚠ **注意：**
>
> 签名算法可参考 签名算法说明 。

参考示例

请求参数：

参数名	参数值
webankAppId	appId001
orderNo	orderNo19959248596551
name	testName
idNo	4300000000000
userId	userID19959248596551
version	1.0.0
ticket	duSz9ptwyW1Xn7r6gYItxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoe

字典排序后的参数为：

[1.0.0, 4300000000000, appId001, duSz9ptwyW1Xn7r6gYItxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoe, orderNo19959248596551, testName, userID19959248596551]

拼接后的字符串为：

1.0.043000000000000appId001duSz9ptwyW1Xn7r6gYItxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoeorderNo19959248596551testNameuserID19959248596551

计算 SHA1 得到签名：

该字串就是最终生成的签名（40位），不区分大小写。

EE57F7C1EDDE7B6BB0DFB54CD902836B8EB0575B

合作方后台上送身份信息

请求

请求 URL： https://idasc.webank.com/api/server/h5/geth5faceid

请求方法：POST

报文格式：Content-Type: application/json

请求参数：

参数	说明	类型	长度（字节）	是否必填

参数	说明	类型	长度（字节）	是否必填
webankAppId	腾讯云线下对接分配的 AppID	String	腾讯云线下对接决定	是
orderNo	订单号，由合作方上送，每次唯一，不能超过32位	String	不能超过32位	是
name	姓名	String	-	是
idNo	证件号码	String	-	是
userId	用户 ID ，用户的唯一标识（不能带有特殊字符）	String	-	是
sourcePhotoStr	比对源照片 1. 原始图片不能超过500k，且必须为 JPG 或 PNG 格式。 2. 参数有值：使用合作伙伴提供的比对源照片进行比对，照片是正脸可信照片，照片质量由合作方保证。 3. 参数为空：根据身份证号+姓名使用权威数据源比对	Base64 String	1048576	否
sourcePhotoType	比对源照片类型 参数值为1：水纹正脸照 参数值为2：高清正脸照	String	1	是
version	默认参数值为：1.0.0	String	20	是
sign	签名：使用上面生成的签名	String	40	是

响应

响应示例：


```
{
"code": 0,
"msg": "成功",
"result": {
"bizSeqNo":"业务流水号",
"orderNo":"合作方订单号",
"h5faceId":"cc1184c3995c71a731357f9812aab988"
}
}
```

启动 H5 人脸验证

最近更新时间：2019-01-22 10:28:18

前置条件

- 请合作方确保 NONCE ticket 已经正常获取，获取方式见 NONCE ticket 获取 。
- 人脸识别需要录制视频，因此要求浏览器支持视频录制功能。
  适用浏览器及版本如下：
  - Firefox 浏览器29及以上版本。
  - Chrome 浏览器53及以上版本。

生成签名

准备步骤

- 合作方根据本次人脸验证的如下参数生成签名，需要签名的参数信息如下：
- 参与签名的数据需要和使用该签名的接口中的请求参数保持一致。

参数	说明	来源
webankAppId	腾讯云线下对接分配的 AppID	腾讯云线下对接分配
orderNo	订单号，本次人脸验证合作伙伴上送的订单号，唯一标识	合作方自行分配
userId	用户 ID ，用户的唯一标识（不要带有特殊字符）	合作方自行分配（与接口中使用的 userId 一致）
version	参数值为：1.0.0	-
h5faceId	h5/geth5faceid 接口返回的唯一标识	-
api ticket	合作伙伴服务端缓存的 tikcet，注意是 NONCE 类型	获取方式见 NONCE ticket 获取（所用的 userId 参数值需要和接口里面的 userId 值保持一致）
nonce	随机数：32位随机串（字母+数字组成的随机数）	合作方自行生成（与接口中的随机数保持一致）

基本步骤

1. 生成一个32位的随机字符串（字母和数字）nonce（接口请求时也要用到）。

2. 将 webankAppId、userId、orderNo、version、h5faceId 连同 ticket、nonce 共7个参数的值进行字典序排序。
3. 将排序后的所有参数字符串拼接成一个字符串。
4. 将排序后的字符串进行 SHA1 编码，编码后的40位字符串作为签名（sign）。

> ⚠ **注意：**
>
> 签名算法可参考 签名算法说明 。

参考示例

请求参数：

参数名	参数值
webankAppId	appId001
userId	userID19959248596551
nonce	kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7T
version	1.0.0
h5faceId	bwiwe1457895464
orderNo	aabc1457895464
ticket	zxc9QfxIti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS

字典排序后的参数为：

[1.0.0, aabc1457895464, appId001, bwiwe1457895464, kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7T, userID19959248596551, zxc9QfxIti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS]

拼接后的字符串为：

1.0.0aabc1457895464appId001bwiwe1457895464kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7TuserID19959248596551zxc9QfxIti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS

计算 SHA1 得到签名：

该字串就是最终生成的签名（40位），不区分大小写。

4E9DFABF938BF37BDB7A7DC25CCA1233D12D986B

启动 H5 人脸验证

合作方上送 h5faceId 以及 sign ，后台校验 sign 通过之后重定向到 H5 人脸验证。

请求

请求 URL： https://ida.webank.com/api/pc/login
请求方法：GET
请求参数：

参数	说明	类型	长度	是否必填
webankAppId	腾讯云线下对接分配的 AppID	String	腾讯云线下对接决定	是
version	接口版本号，默认参数值：1.0.0	String	20	是
nonce	随机数：32位随机串（字母+数字组成的随机数）	String	32	是
orderNo	订单号，由合作方上送，每次唯一，此信息为本次人脸验证上送的信息，不能超过32位	String	32	是
h5faceId	h5/geth5faceid 接口返回的唯一标识	String	32	是
url	H5 人脸验证完成后回调的第三方 URL，需要第三方提供完整 URL 且做 URL Encode。 完整 URL Encode 示例 原 URL： https://idaop.webank.com Encode 后： http://idaop.webank.com 默认：0	String	-	否
userId	用户 ID ，用户的唯一标识（不要带有特殊字符）	String	-	是
sign	签名：使用上面生成的签名	String	40	是

PC 端 H5 人脸验证结果跳转

最近更新时间：2019-01-22 10:28:36

用于实现人脸验证 H5 结果返回跳转第三方 URL 带唯一标识、订单号、验证结果、签名。

请求

请求 URL： https://xxx.com/xxx?code=xxxx&orderNo=xxxx&h5faceId=xxxx&newSignature=xxxx
请求方法：GET

> ⚠ **注意：**
>
> - xxxx.com 为合作方上送的 URL。
> - 合作方根据 方式一：前端获取结果验证签名 或者 方式二：服务端查询结果 说明进行签名校验，确保返回结果的安全性。

响应

响应参数：

参数	说明	类型	长度（字节）
code	人脸验证结果的返回码，0表示人脸验证成功，其他错误码标识失败。	String	-
orderNo	订单号 ，由合作方上送，每次唯一，此信息为本次人脸验证上送的信息。	String	32
h5faceId	本次请求返回的唯一标识，此信息为本次人脸验证上送的信息。	String	32
newSignature	对 URL 参数 AppID、oderNo 和 SIGN ticket、code 的签名 详情请参见 方式二：服务端查询结果的 合作方后台生成签名	String	40

验证结果

方式一：前端获取结果验证签名

最近更新时间：2018-07-27 19:03:34

为了确保前端 SDK 和 H5 返回的结果真实性且未被篡改，合作伙伴服务端可以验证结果，我们提供两种结果验证方式，合作伙伴可以根据需求自行选择。

此方式用于：

1. 合作伙伴 App 端或 H5 收到远程人脸验证 SDK 返回以及签名结果。
2. 合作伙伴 App 端或 H5 调用其服务端接口进行签名认证，接口认证成功后继续业务流程。

1. 合作方后台生成签名

准备步骤

- **前置条件：请合作方确保 SIGN ticket 已经正常获取，获取方式见 SIGN ticket 获取 。**
- 合作方为人脸验证服务生成签名，需要具有以下参数：

参数	说明	来源
app_id	腾讯云线下对接分配的 App ID	腾讯云线下对接分配
order_no	订单号，本次人脸验证合作伙伴上送的订单号，唯一标识	合作方自行分配
api ticket	合作伙伴服务端缓存的 tikcet，注意是 SIGN 类型	获取方式见 SIGN ticket 获取
Code	人脸验证结果的返回码，0 表示人脸验证成功，其他错误码标识失败。	-

基本步骤

1. 将 appId, orderNo, ticket（SIGN 类型）、code 共四个参数的值进行字典序排序。
2. 将排序后的所有参数字符串拼接成一个字符串。
3. 将排序后的字符串进行 SHA1 编码，编码后的 40 位字符串作为签名（sign）。

> **注意：**
>
> 签名算法可参考 签名算法说明 。

参考示例

请求参数：

参数名	参数值
app_id	appId001
order_no	test1480921551481
ticket	duSz9ptwyW1Xn7r6gYltxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoe
code	0

字典排序后的参数为：

```
[0，appId001, duSz9ptwyW1Xn7r6gYltxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoe, test1480921551481]
```

拼接后的字符串为：

```
0appId001duSz9ptwyW1Xn7r6gYltxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoetest1480921551481
```

计算 SHA1 得到签名：

```
526365E042766AE27A6E52D2E4829D4C6E156B5D
```

该字串就是最终生成的签名（40位），不区分大小写。

2. 比对签名

合作方服务端生成的签名与 SDK 返回的签名比对，如果相同即可信任 SDK 的人脸验证结果。

> **注意：**
>
> 合作方必须定时刷新 ticket（SIGN）保证远程身份认证后台缓存有该合作方的 ticket（SIGN），否则远程身份认证后台无法生成签名值。

方式二：服务端查询结果

最近更新时间：2018-10-09 14:52:37

此方式用于：

1. 合作伙伴 App 端或 H5 调用其服务端接口查询身份验证结果。
2. 合作伙伴服务端生成签名，并调用远程身份认证服务端查询结果，远程身份认证服务端鉴权完成后返回结果（服务端上送 order_no 和 app_id 查询）。
3. 合作伙伴 App 端或 H5 获取结果后继续后续流程。

1. 合作方后台生成签名

准备步骤

- **前置条件：请合作方确保 SIGN ticket 已经正常获取，获取方式见 SIGN ticket 获取 。**
- 合作方为人脸验证服务生成签名，需要具有以下参数：

参数	说明	来源
app_id	腾讯云线下对接分配的 App ID	腾讯云线下对接分配
order_no	订单号，本次人脸验证合作伙伴上送的订单号，唯一标识	合作方自行分配
version	默认值：1.0.0	
api ticket	合作伙伴服务端缓存的 tikcet，注意是 SIGN 类型	获取方式见 SIGN ticket 获取
nonceStr	32 位随机字符串，字母和数字	合作方自行生成

基本步骤

1. 生成一个 32 位的随机字符串 nonceStr（其为字母和数字，登录时也要用到）。
2. 将 app_id、order_no、version 连同 ticket、nonceStr 共五个参数的值进行字典序排序。
3. 将排序后的所有参数字符串拼接成一个字符串。
4. 将排序后的字符串进行 SHA1 编码，编码后的 40 位字符串作为签名（sign）。

> **注意：**
>
> 签名算法可参考 签名算法说明 。

2. 身份认证查询接口

请求

请求URL：

```
https://idasc.webank.com/api/server/sync
```

请求方法：GET

请求参数：

参数	说明	类型	长度（字节）	是否必填
app_id	腾讯服务分配的app_id	字符串	腾讯服务分配	是
version	版本号，默认值：1.0.0	字符串	20	是
nonce	随机数	字符串	32	是
order_no	订单号，合作方订单的唯一标识	字符串	32	是
sign	签名值，使用本页第一步生成的签名	字符串	40	是
get_file	是否需要获取人脸识别的视频和文件 值为 1 则返回视频和照片 值为 2 则返回照片 值为 3 则返回视频 其他则不返回	字符串	1	否，非必填

请求示例：

```
https://idasc.webank.com/api/server/sync?app_id=xxx&nonce=xxx&order_no=xxx&version=1.0.0&sign=xxx
```

响应

响应参数：

参数	类型	说明
code	String	0：身份验证成功且认证为同一人
msg	String	返回结果描述
bizSeqNo	String	业务流水号
orderNo	String	订单编号

参数	类型	说明
idNo	String	证件号码
idType	String	证件类型
name	String	姓名
liveRate	String	活体检测得分
similarity	String	人脸比对得分
occurredTime	String	进行刷脸的时间
photo	Base64 String	人脸验证时的照片，Base64 位编码
video	Base64 String	人脸验证时的视频，Base64 位编码
app_id	String	腾讯服务分配的 app_id

响应示例：

{"code":"0","msg":"请求成功","bizSeqNo":"1808102000101530021503420085979 2","result":{"bizSeqNo":"180810200010153002150342008597 92","transactionTime":"20180810150342","orderNo":"orderNo19959248596551",
idNo":"***","idType":"01","name":"***","video":"******","photo":"******","liveRate":"100","similarity":"95.0","occurredTime":"20180810150152","success":false},"transactionTime":"20180810150342","app_id":"TIDAlsRT","order_no":"orderNo19959248596551"} }

code 非 0 时，有时不返回图片和视频。

注意：

1. 照片和视频信息作为存证，合作伙伴可以通过此接口拉取视频等文件，需要注意请求参数的 get_file 需要设置为 1；如果不上送参数或者参数为空，默认不返回视频和照片信息。为确保用户操作整体流程顺利完成，部分情况下获取视频和照片会有1秒左右的延迟。
2. 由于照片和视频信息有可能超过 1M，考虑传输的影响，建议合作伙伴在使用时注意，建议获取比对结果用于后续流程处理和存证使用分开调用。避免网络传输带来的影响。
3. 照片和视频均为 base64 位编码，其中照片解码后格式一般为 JPG 和 PNG。视频格式解码后一般为 MP4。

人脸认证多张照片查询接口

最近更新时间：2019-01-22 10:27:48

合作方后台生成签名

准备步骤

- **前置条件：**请合作方确保 SIGN ticket 已经正常获取，获取方式见 SIGN ticket 获取 。
- 合作方为人脸验证服务生成签名，需要具有以下参数：

参数	说明	来源
webankAppId	腾讯云线下对接分配的 AppID	腾讯云线下对接分配
orderNo	订单号，本次人脸验证合作伙伴上送的订单号，唯一标识	合作方自行分配
version	默认值：1.0.0	-
api ticket	合作伙伴服务端缓存的 tikcet，注意是 SIGN 类型	获取方式见 SIGN ticket 获取
nonce	32位随机字符串，字母和数字	合作方自行生成

基本步骤

1. 生成一个32位的随机字符串 nonceStr（其为字母和数字，登录时也要用到）。
2. 将 app_id、order_no、version 连同 ticket、nonceStr 共5个参数的值进行字典序排序。
3. 将排序后的所有参数字符串拼接成一个字符串。
4. 将排序后的字符串进行 SHA1 编码，编码后的40位字符串作为签名（sign）。

> ⚠ **注意：**
>
> 签名算法可参考 签名算法说明 。

人脸认证多张照片查询接口

请求

请求URL： `https://idasc.webank.com/api/v2/base/queryphotoinfo`

请求方法：POST

HTTP 请求 header：

参数名	是否必选	类型	说明
Content-Type	是	String	application/json

请求参数：

参数	说明	类型	长度（字节）	是否必填
webankAppId	腾讯服务分配的 AppID	String	腾讯服务分配	是
version	版本号，默认值：1.0.0	String	20	是
nonce	随机数	String	32	是
orderNo	订单号，合作方订单的唯一标识	String	32	是
sign	签名值，使用本页第一步生成的签名	String	40	是

响应

返回参数说明：

参数名	类型	说明
code	int	0：成功 非0：失败
msg	String	请求结果描述
bizSeqNo	String	请求业务流水号
orderNo	String	订单编号
occurredTime	String	刷脸时间（yyyyMMddHHmmss）
photoList	List	Base64 图像列表（返回1 - 3张照片）

返回示例：


```
{
"code": 0,
"msg": "请求成功",
"bizSeqNo": "业务流水号",
"result": {
"orderNo": "AAAAAA001",
"occurredTime": "20180907142653",
"photoList": ["第一个base64photo字符串", "第二个base64photo字符串", "第三个base64photo字符串"]
```

```
}
}
```