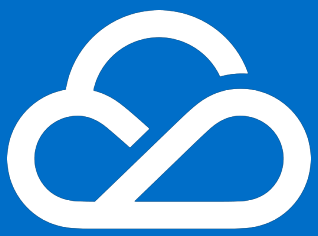
腾讯云天御业务安全防护

金融级身份认证

产品文档



腾讯云

【版权声明】

【商标声明】

【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或模式的承诺或保证。

文档目录

金融级身份认证

接入前必读

为了确保服务接入安全性，整个服务使用的是标准OAuth2.0鉴权认证，整个鉴权分为2部分，签名鉴权和SDK鉴权，具体获取信息如下：

注意：请合作方按照模板填写相关信息后，生成并分配相关鉴权信息。

1.接入信息申请分配

合作方按照模板申请接入的APPID、Secret和SDK license。

按照《整体登录鉴权流程》开始的接入描述获取Token并生成签名。

注意：APPID 和Secret务必保密，一旦泄露，将会导致服务不可用。后果由合作方自行承担。

2.签名鉴权

合作作伙伴服务端根据《整体登录鉴权流程》开始的接入描述获取Token并生成签名。

只有上述步骤成功完成后才能确保能正常调用人脸验证服务端后台接口。

3.SDK 鉴权

用于合作伙伴启动远程身份验证SDK是鉴权，此鉴权的SDK License是根据合作方APP的iOS Bundle id和Android Package name生成。需要合作伙伴按照模板提供后分配生成。

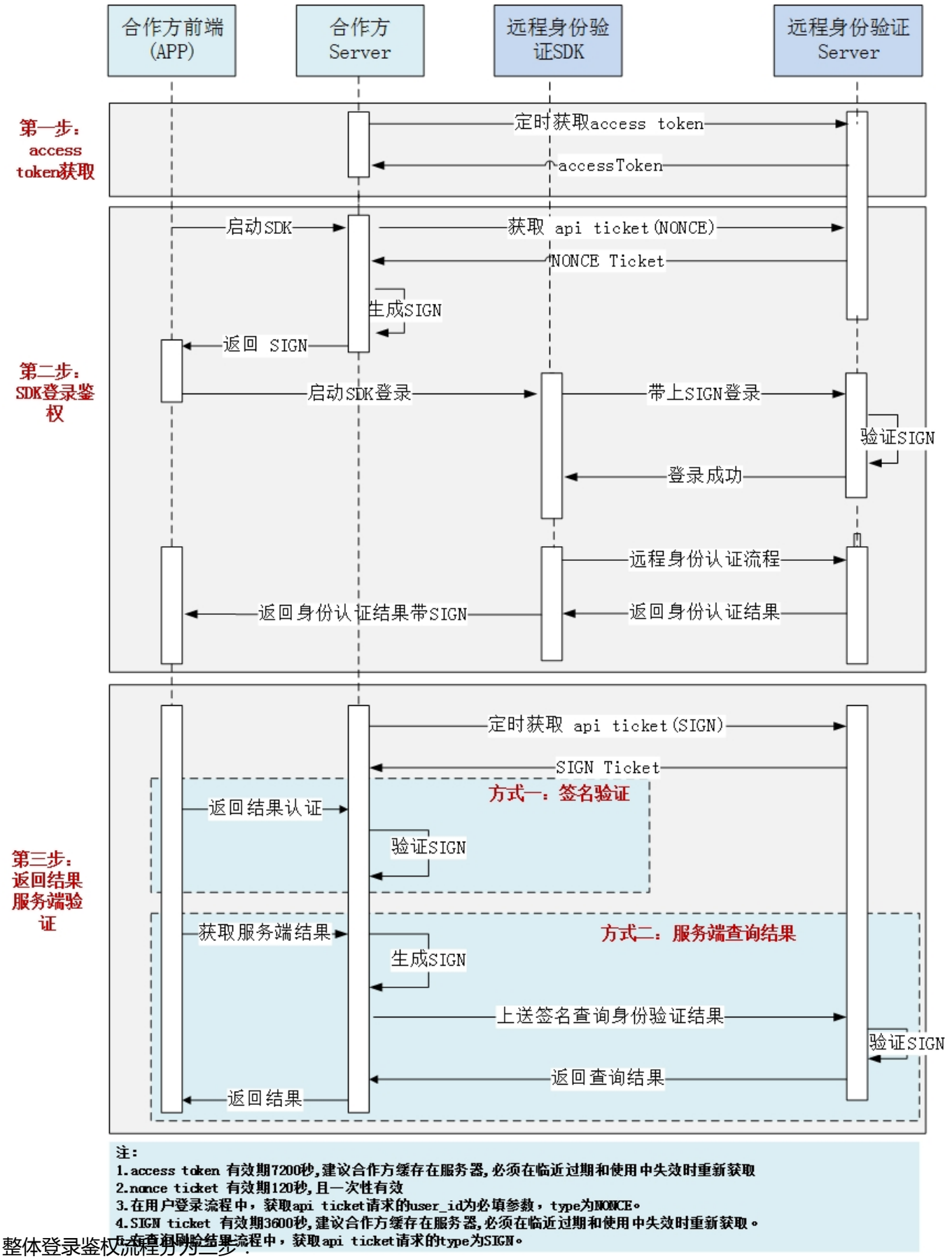
注意：如果合作伙伴不使用SDK人脸验证模块，此处不需要。

鉴权流程

注意事项

整体鉴权流程基于 OAuth2.0 认证。合作伙伴同时使用人脸验证/身份证 OCR 识别服务时，鉴权所需的 token、ticket 及签名算法可以共用。只需实现各自的业务逻辑即可。

人脸验证 SDK 登录鉴权



整体登录鉴权流程分为三步：

第一步 access token 获取

access token 有效期 7200 秒，建议合作方缓存在服务器,必须在临近过期和使用中失效时重新获取。

第二步 SDK 登录鉴权

注意：

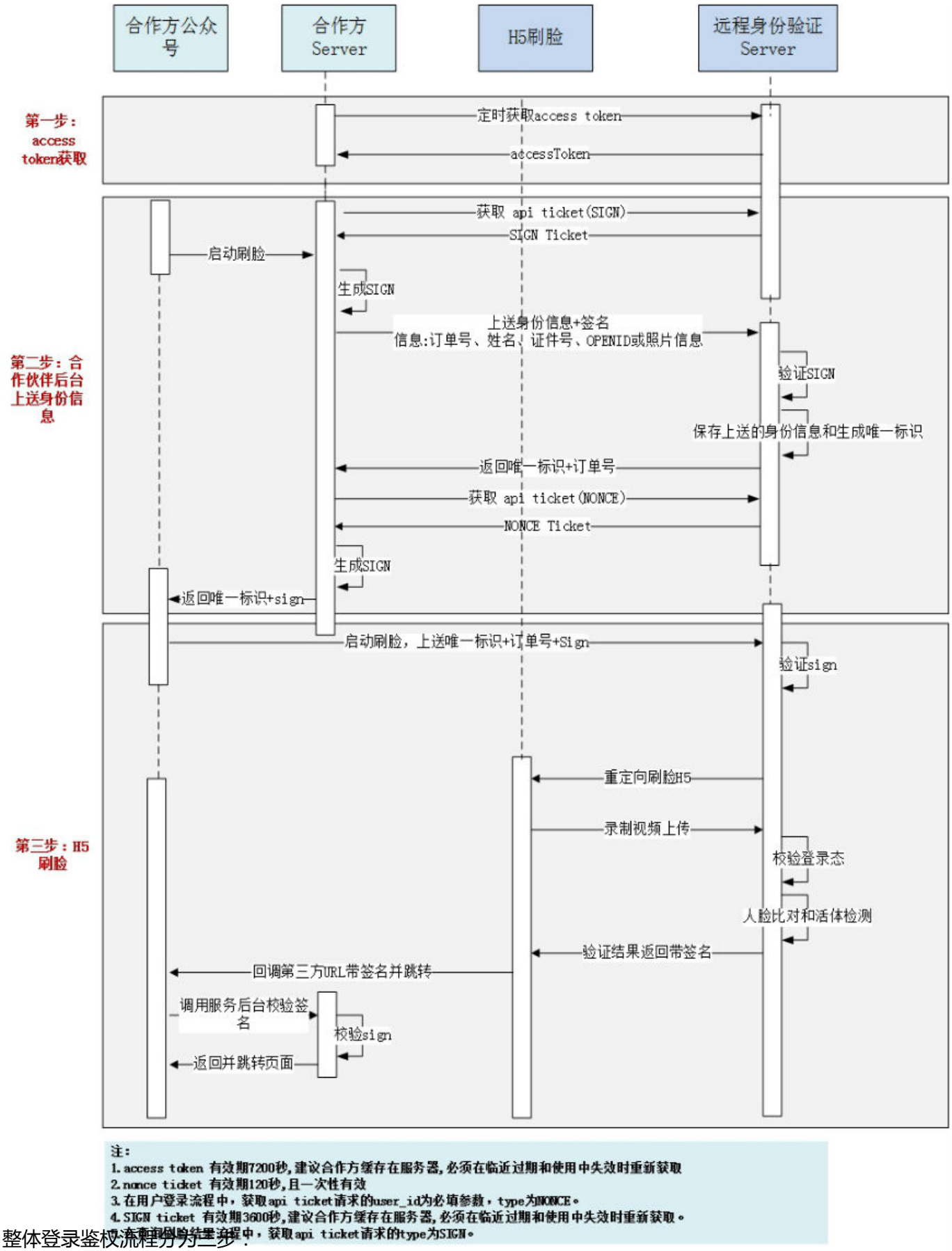
1. nonce ticket 有效期 120 秒,且一次性有效
2. 在用户登录流程中，获取 api ticket 请求的 user_id 为必填参数，type 为 NONCE。

第三步 返回结果服务端验证

SIGN ticket 有效期 3600
秒，建议合作方缓存在服务器，必须在临近过期和使用中失效时重新获取。流程中，获取 api ticket 请求的 type 为 SIGN。

- 方式一：签名验证
  合作伙伴 APP 端调用其服务后台验证签名，验证成功后即可信任前端的返回结果。
- 方式二：服务端查询结果
  合伙伙伴 APP 端调用其服务后台查询身份认证结果，由远程身份认证服务端鉴权并返回最终结果。

人脸验证微信 H5 登录鉴权



整体登录鉴权流程分为三步：

第一步 access token 获取

access token 有效期 7200 秒，建议合作方缓存在服务器,必须在临近过期和使用中失效时重新获取。

第二步 合作伙伴后台上送身份信息

1. 生成签名

合作伙伴获取 SIGN ticket， 有效期 3600
秒,建议合作方缓存在服务器,必须在临近过期和使用中失效时重新获取。流程中，获取 api ticket 请求的 type 为 SIGN。根据规则生成签名。

2. 上送身份信息

合作伙伴上送本次人脸验证的用户身份信息，可以支持合作伙伴提供自带可信比对数据源和使用姓名+身份证获取权威比对数据源两种方式。成功后返回标识信息。

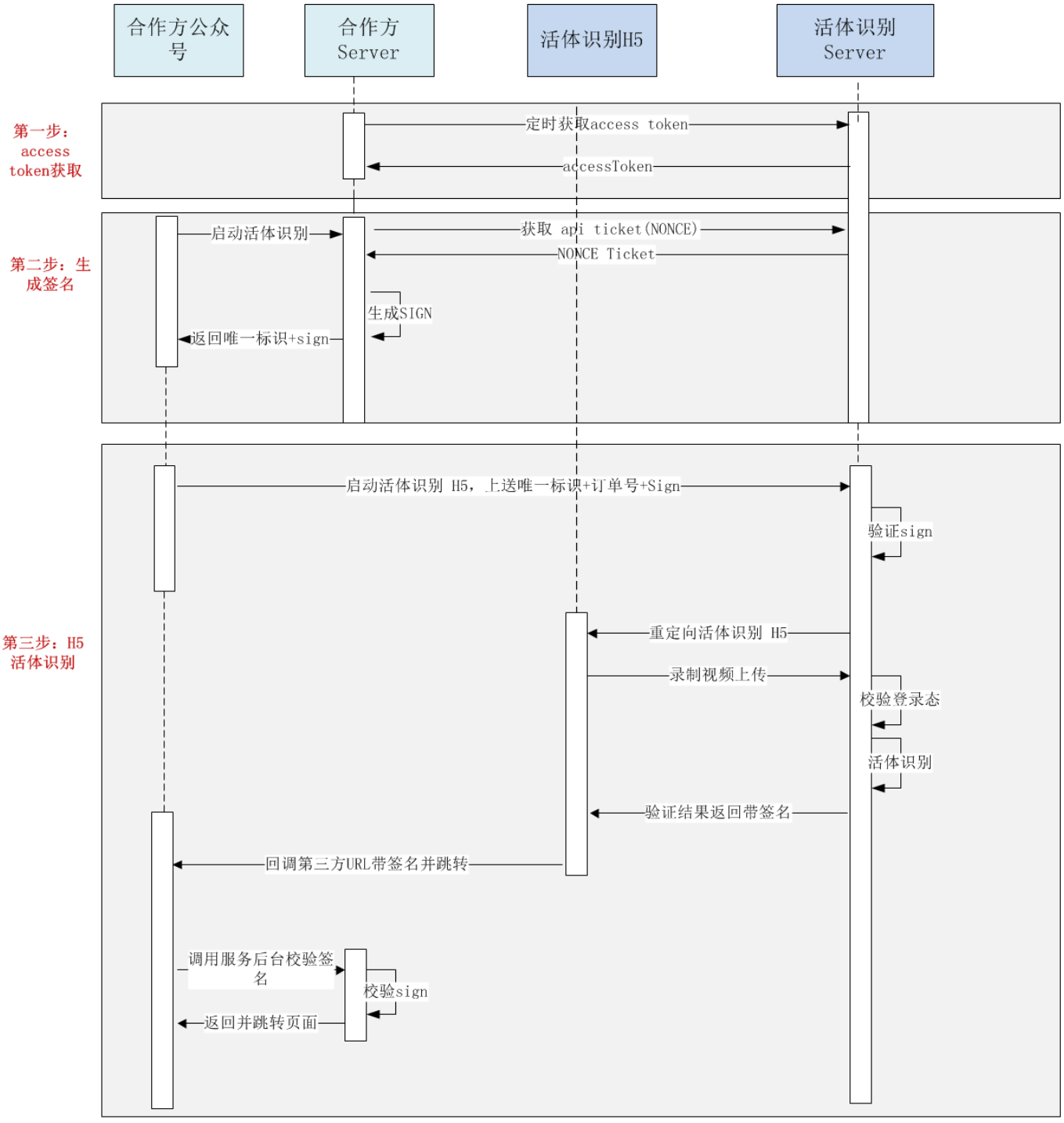
第三步 H5 刷脸

注意：

- NONCE ticket 有效期 120 秒,且一次性有效
- 在用户登录流程中，获取 api ticket 请求的 user_id 为必填参数，type 为 NONCE。

- 方式一：签名验证
  合作伙伴 APP 端调用其服务后台验证签名，验证成功后即可信任前端的返回结果。
- 方式二：服务端查询结果
  合伙伙伴 APP 端调用其服务后台查询身份认证结果，由远程身份认证服务端鉴权并返回最终结果。
  具体接入开发指南参见：人脸验证微信 H5 接入。

活体识别 H5 登录鉴权



注：
1. access token 有效期7200秒，建议合作方缓存在服务器，必须在临近过期和使用中失效时重新获取
2. nonce ticket 有效期120秒，且一次性有效
3. 在用户登录流程中，获取api ticket请求的user_id为必填参数，type为NONCE。
4. SIGN ticket 有效期3600秒，建议合作方缓存在服务器，必须在临近过期和使用中失效时重新获取。
5. 在查询识别结果流程中，获取api ticket请求的type为SIGN。

** 整体登录鉴权流程分为三步：

第一步 access token 获取

access token 有效期 7200 秒，建议合作方缓存在服务器，必须在临近过期和使用中失效时重新获取。

第二步 生成签名

合作伙伴获取 SIGN ticket， 有效期 3600
秒，建议合作方缓存在服务器，必须在临近过期和使用中失效时重新获取。流程中，获取 api ticket 请求的 type 为 SIGN。根据规则生成签名。

第三步 H5 活体识别

注意：

- NONCE ticket 有效期 120 秒,且一次性有效
- 在用户登录流程中，获取 api ticket 请求的 user_id 为必填参数，type 为 NONCE。

- 方式一：签名验证
  合作伙伴 APP 端调用其服务后台验证签名，验证成功后即可信任前端的返回结果。
- 方式二：服务端查询结果
  合伙伙伴H5端调用其服务后台查询身份认证结果，由活体识别服务端鉴权并返回最终结果。
  具体接入开发指南餐见：活体识别微信 H5 接入。

人脸验证小程序登录鉴权

第一步 关联人脸验证小程序

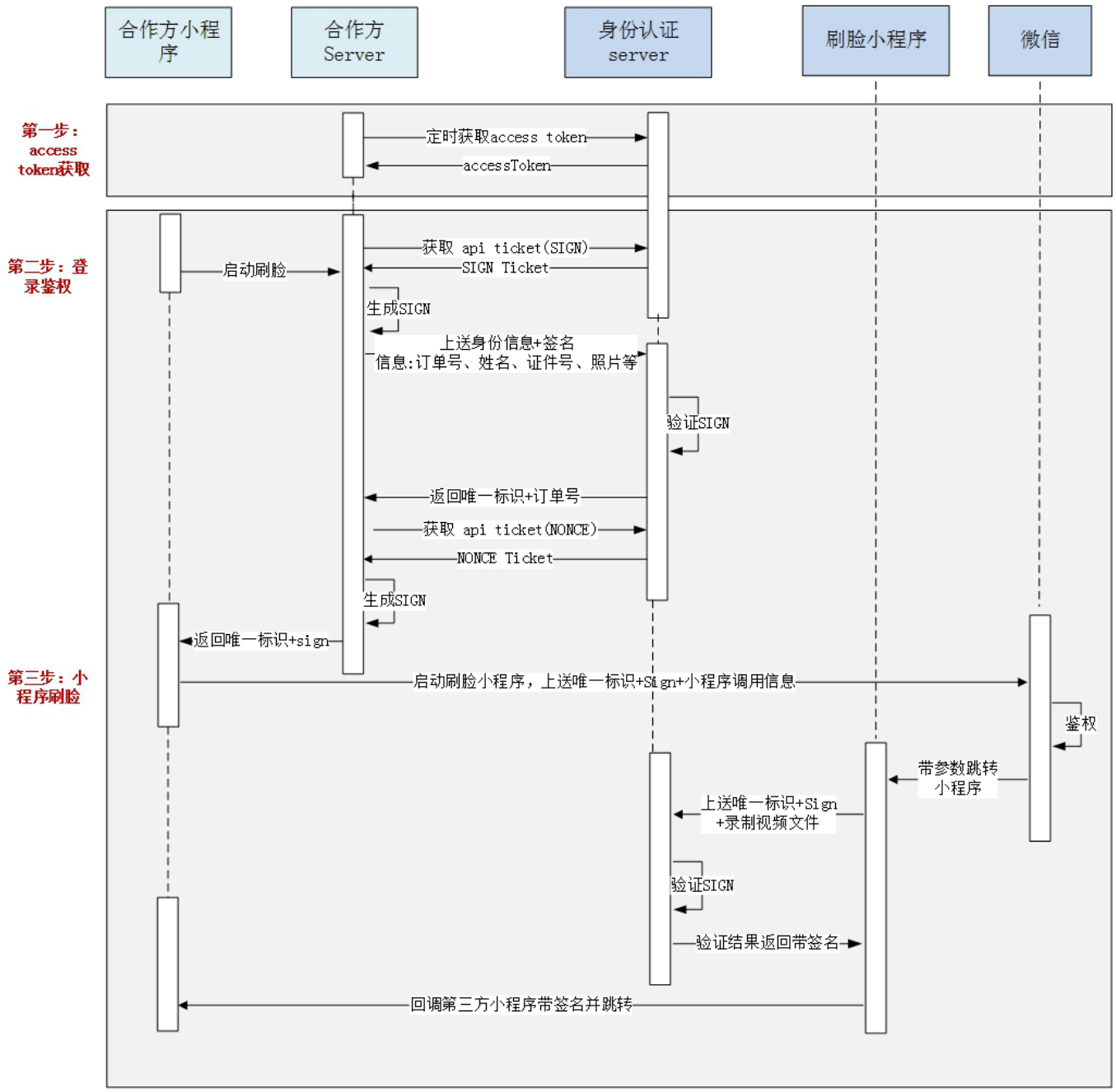
合作方在接入前，需要先将公众号关联到刷脸小程序，授权成功后方可实现刷脸小程序的调用。
合作方小程序对应的公众号关联刷脸小程序的流程请参考 微信官方指南。

关联流程

1. 公众号关联小程序：登录公众号后台-小程序-小程序管理-添加-关联小程序
   通过 appid：wx7ccfa42a2a641035 搜索刷脸小程序并关联。

2. 联系腾讯工作人员授权小程序关联，并确认小程序关联成功。

第二步 小程序登录鉴权整体流程



小程序整体登录鉴权流程分为三步：

1. access token 获取

   access token 有效期 7200

   秒，建议合作方缓存在服务器，必须在临近过期和使用中失效时重新获取。

2. 合作伙伴后台上送身份信息

   1. 生成签名

      合作伙伴获取 SIGN ticket， 有效期 3600

      秒，建议合作方缓存在服务器，必须在临近过期和使用中失效时重新获取。流程中，获取 api ticket 请求的 type 为 SIGN。根据规则生成签名。

   2. 上送身份信息

合作伙伴上送本次人脸验证的用户身份信息，可以支持合作伙伴提供自带可信比对数据源和使用姓名+身份证获取权威比对数据源两种方式。成功后返回标识信息。

3. 小程序刷脸
    1. 合作伙伴小程序发起请求，通过微信鉴权后跳转到刷脸小程序，刷脸完成后将“唯一标识 + sign + 录制视频文件”传给身份认证 server 进行处理
    2. 返回刷脸结果

       合作伙伴可实时获取刷脸的结果：小程序调用其服务后台验证签名，验证成功后即可信任前端的返回结果。
    3. 合作方服务端异步查询更多结果

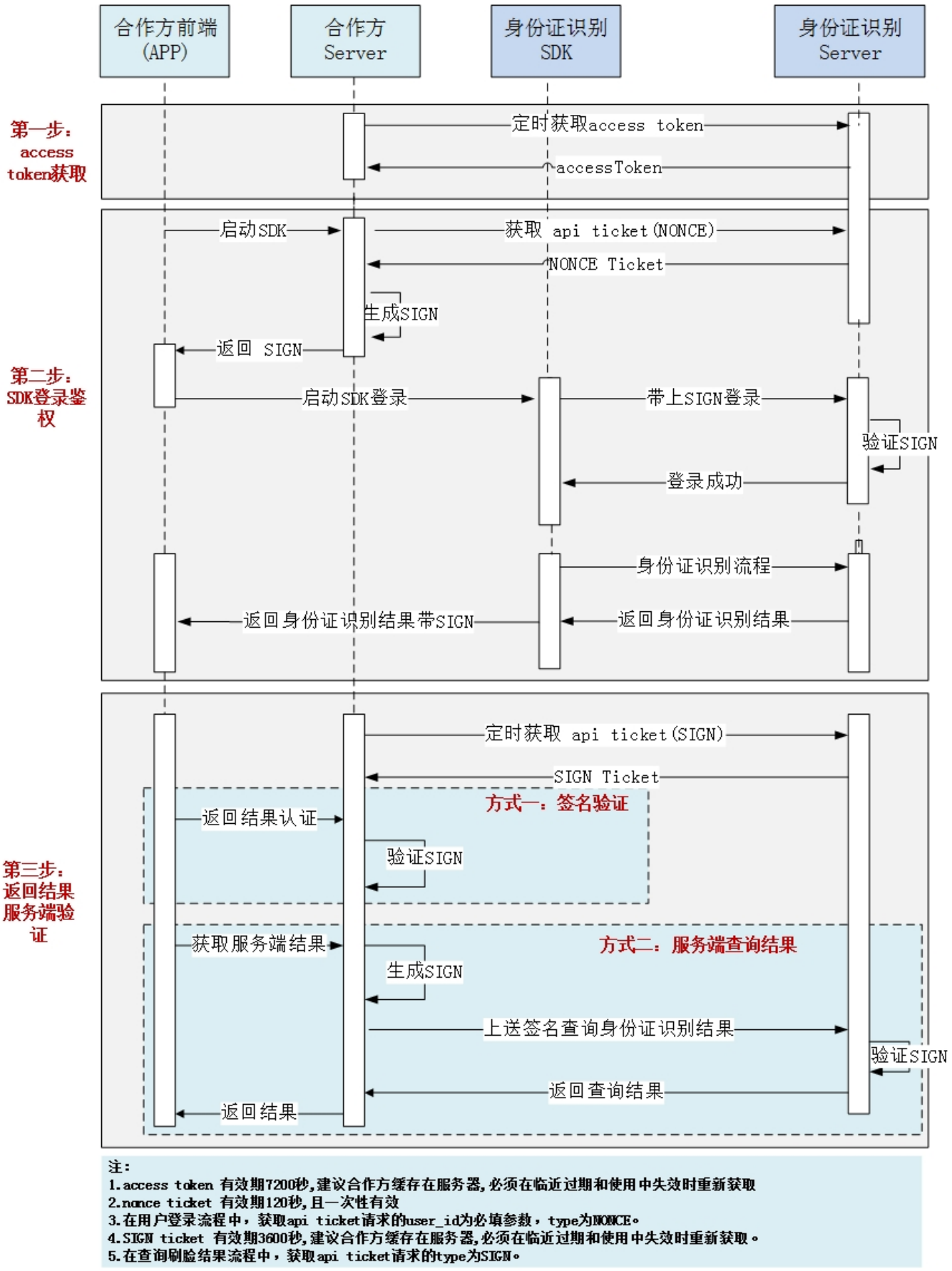
       合伙伙伴小程序调用其服务后台查询身份认证结果，由身份认证服务端鉴权并返回刷脸照片、视频等更多结果。

       具体接入开发指南参见：人脸验证微信小程序接入。

注意：

- NONCE ticket 有效期 120 秒，且一次性有效
- 在用户登录流程中，获取 api ticket 请求的 userId 为必填参数，type 为 NONCE。

OCR 识别 SDK 登录鉴权



整体登录鉴权流程分为三步：

第一步 access token 获取

access token 有效期 7200 秒，建议合作方缓存在服务器，必须在临近过期和使用中失效时重新获取。

第二步 SDK 登录鉴权

注意：

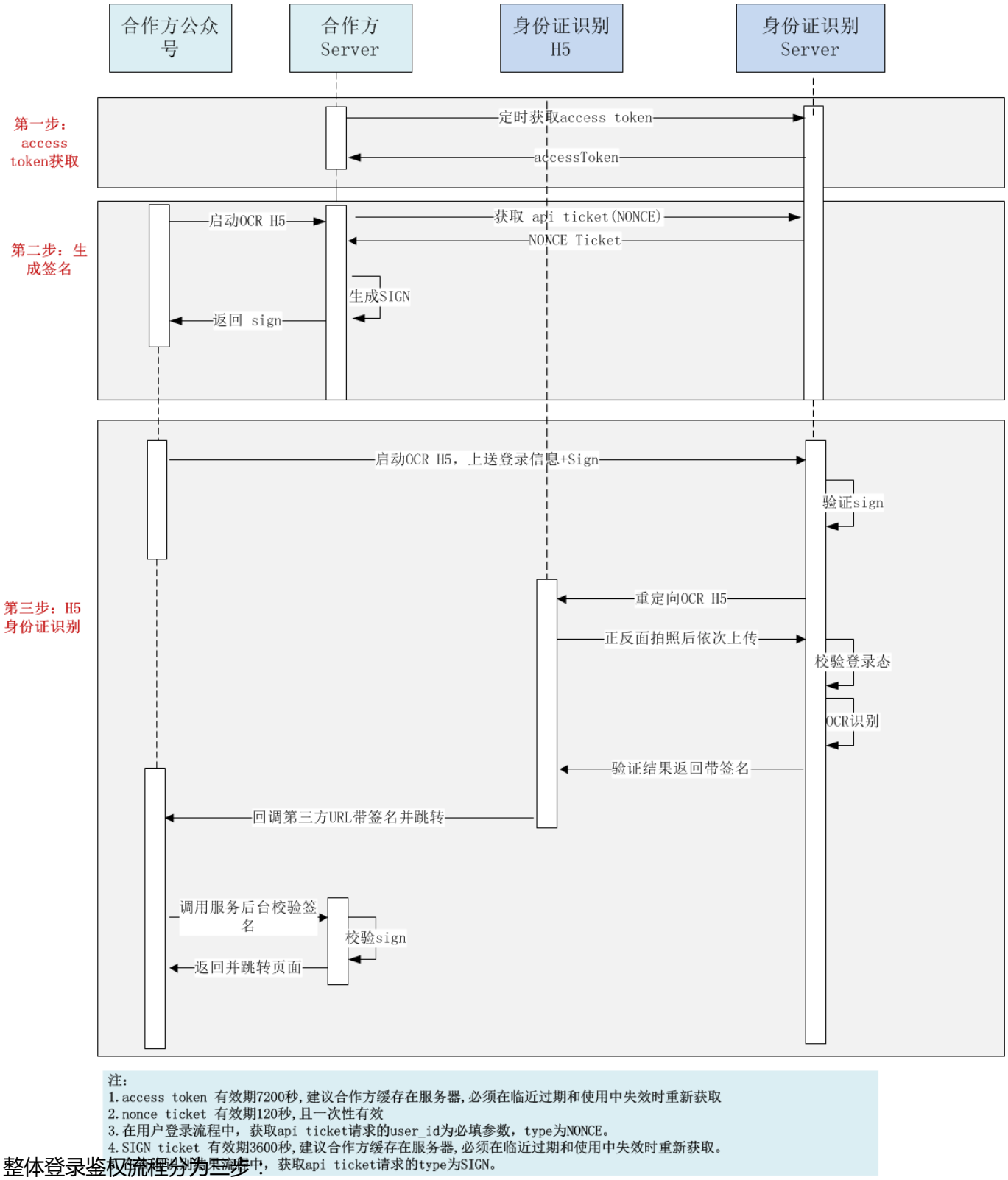
- nonce ticket 有效期 120 秒，且一次性有效
- 在用户登录流程中，获取 api ticket 请求的 user_id 为必填参数，type 为 NONCE。

第三步 返回结果服务端验证

SIGN ticket 有效期 3600
秒，建议合作方缓存在服务，必须在临近过期和使用中失效时重新获取。流程中，获取 api ticket 请求的 type 为 SIGN。

- 方式一：签名验证
  合作伙伴 APP 端调用其服务后台验证签名，验证成功后即可信任前端的返回结果。
- 方式二：服务端查询结果
  合伙伙伴 APP 端调用其服务后台查询识别结果，由 OCR 识别服务端鉴权并返回最终结果。

身份证 OCR 识别 H5 识别登陆鉴权



整体登录鉴权流程分为三步：

第一步 access token 获取

access token 有效期 7200 秒，建议合作方缓存在服务器，必须在临近过期和使用中失效时重新获取。

第二步 生成签名

合作伙伴获取 SIGN ticket， 有效期 3600
秒，建议合作方缓存在服务器，必须在临近过期和使用中失效时重新获取。流程中，获取 api ticket 请求的 type 为 SIGN。根据规则生成签名。

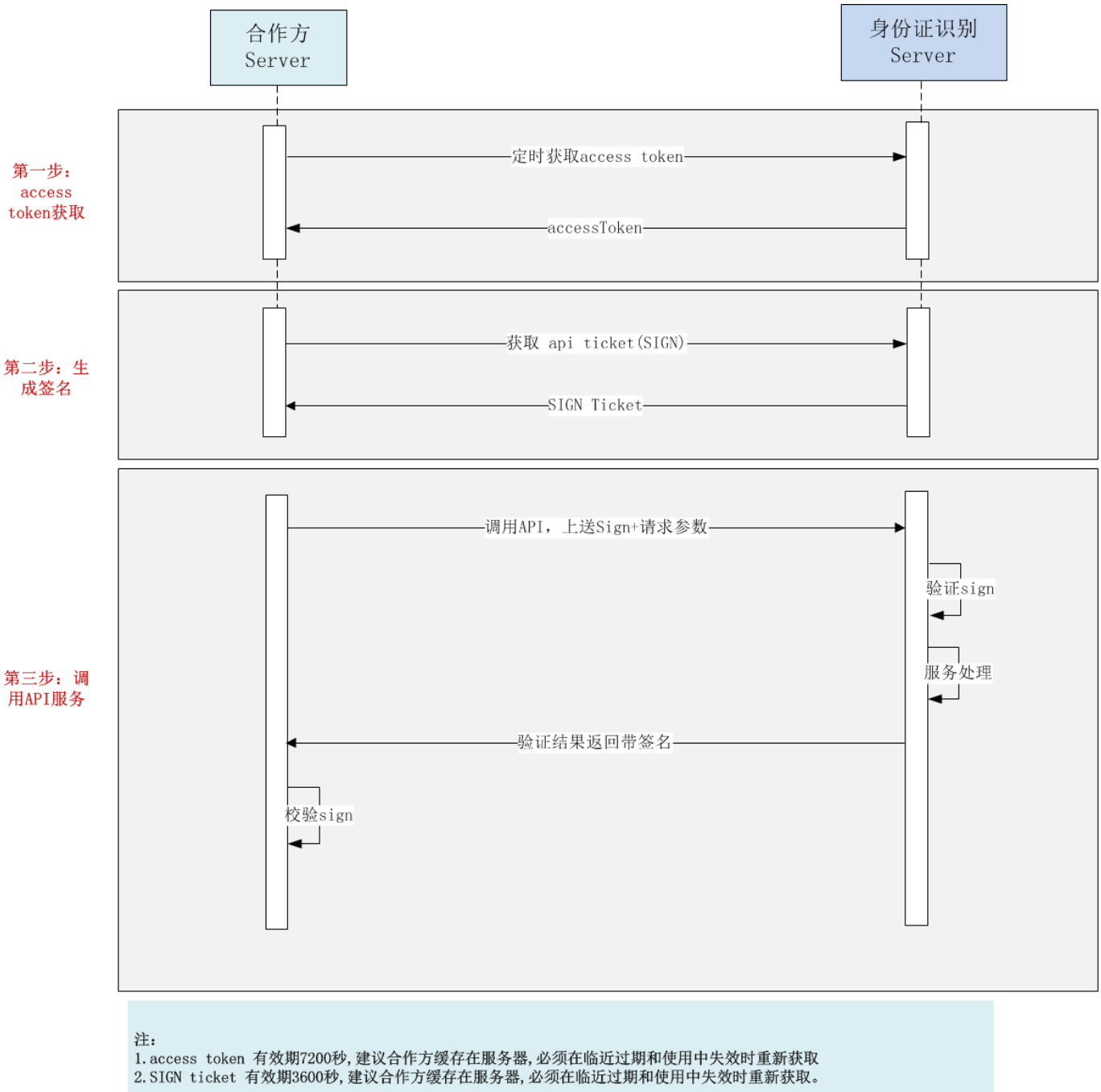
第三步 H5 身份证识别

注意：

- NONCE ticket 有效期 120 秒，且一次性有效
- 在用户登录流程中，获取 api ticket 请求的 user_id 为必填参数，type 为 NONCE。

- 方式一：签名验证
  合作伙伴 H5 端调用其服务后台验证签名，验证成功后即可信任前端的返回结果。
- 方式二：服务端查询结果
  合伙伙伴 H5 端调用其服务后台查询身份认证结果，由身份证识别服务端鉴权并返回最终结果。
  具体接入开发指南参见：身份证识别微信 H5 接入。

API 调用服务登录鉴权



整体登录鉴权流程分为三步：

第一步 access token 获取

access token 有效期 7200 秒，建议合作方缓存在服务器，必须在临近过期和使用中失效时重新获取。

第二步 生成签名

合作伙伴获取 SIGN ticket， 有效期 3600
秒，建议合作方缓存在服务器，必须在临近过期和使用中失效时重新获取。流程中，获取 api ticket 请求的

type 为 SIGN。根据规则生成签名。

第三步 调用 api 服务

注意：

- SIGN ticket 有效期 3600 秒，且一次性有效
- 服务端查询结果：合伙伙伴服务后台校验 SIGN，查询身份认证结果。

Access Token 获取

1.注意事项

1.所有场景默认采用 UTF-8 编码。
2.access token 必须缓存在磁盘，并定时刷新,建议每 1 小时 50 分钟请新的 access token,原 access token 2 小时 (7200S) 失效，期间两个 token 都能使用。
3.每次用户登录时必须重新获取 ticket。

2.Access Token 获取

请求URL：https://idasc.webank.com/api/oauth2/access_token
请求方法:GET
请求参数：

参数	说明	类型	长度	是否必填
app_id	腾讯服务分配的 app_id	字符串	腾讯服务分配	必填，腾讯服务分配的 app_id
secret	腾讯服务 app_id 的密钥	字符串	腾讯服务分配	必填，腾讯服务分配的 secret
grant_type	授权类型	字符串	20	必填，默认值：client_credential (必须小写)
version	版本号	字符串	20	必填，默认值：1.0.0

请求示例：

https://idasc.webank.com/api/oauth2/access_token?app_id=xxx&secret=xxx&grant_type=client_credential&version=1.0.0

响应：

{
"code":"0","msg":"请求成功",
"transactionTime":"20151022043831",
"access_token":"accessToken_string",
"expire_time":"20151022043831",
"expire_in":"7200"
}

注：

1．code 不是 0 是表示获取失败，可以根据 code 和 msg 字段定位和调试。

2．expire_in 为 access_token 的最大生存时间，单位秒，合作伙伴在判定有效期时以此为准。

3．expire_time 为 access_token
失效的绝对时间，由于各服务器时间差异，不能使用作为有效期的判定依据，只展示使用。

4．修改 secret 之后，该 app_id 生成的 access_token 和 ticket 都失效。

API Ticket 获取

SIGN ticket 获取

1.注意事项

1.前置条件：确保 Access Token 已经正常获取，获取规则见登录鉴权流程。
2.主要用于合作方后台服务端业务请求
生成签名鉴权参数之一，用于后台查询验证结果、调用其他业务服务等。
3.api ticket 为 SIGN 类型，有效期为最长为 3600S, 此处 api ticket
的必须缓存在磁盘，并定时刷新,刷新的机制如下：

- 由于 api ticket 的生命周期依赖于 Access Token。最长为 3600S,故为了简单方便，建议 api ticket 的刷新机制与Access Token 定时机制原理一致，严格按照每50分钟请求新的api ticket,原api ticket 1 小时 (3600S) 失效，期间两个 api ticket 都能使用。
- 在获取新的 api ticket，请注意返回的 expire_in 为此 ticket 的最大生存周期，最大为 3600S,具体以实际返回为准，如果 expire_in 大于 600,那么下次刷新时间为 expire_in – 600S; 如果返回的 expire_in 小于等于 600，那么下次取 ticket时需要立刻刷新。

2.获取 SIGN ticket

请求 URL: https://idasc.webank.com/api/oauth2/api_ticket
请求方法:GET
请求参数：

参数	说明	类型	长度	是否必填
app_id	腾讯服务分配的 app_id	字符串	腾讯服务分配	必填，腾讯服务分配的 app_id
access_token	根据《整体登录鉴权流程》获取 access token	字符串	腾讯服务分配	根据《整体登录鉴权流程》获取 access token
type	ticket 类型	字符串	20	必填，默认值：SIGN (必须大写)

参数	说明	类型	长度	是否必填
version	版本号	字符串	20	必填 ，默认值：1.0.0

请求示例：

https://idasc.webank.com/api/oauth2/api_ticket?app_id=xxx&access_token=xxx&type=SIGN&version=1.0.0

响应：

{
"code":"0",
"msg":"请求成功",
"transactionTime":"20151022044027",
"tickets":[
{"value":"ticket_string",
"expire_in":"3600"，
"expire_time":"20151022044027"}
]
}

注:

1.code不是0是表示获取失败，可以根据code和msg字段定位和调试。

2.expire_in为access_token的最大生存时间，单位秒，合作伙伴在判定有效期时以此为准。

3.expire_time为access_token失效的绝对时间，由于各服务器时间差异，不能使用作为有效期的判定依据，只展示使用。

4.access_token失效时，该access_token生成的ticket都失效

NONCE ticket 获取

1.注意事项

1.前置条件：确保 Access Token 已经正常获取，获取规则见登录鉴权流程。
2.主要用于合作方前端包含 APP 和 H5 等生成签名鉴权参数之一，启动 H5 或 SDK 人脸验证。
3.api ticket 为 NONCE 类型，有效期为 120S，且一次性有效, 即每次启动 SDK 刷脸都要重新请求 NONCE ticket。

2.获取 NONCE ticket

ticket 用于对请求数据签名或加密。
请求 URL: https://idasc.webank.com/api/oauth2/api_ticket
请求方法:GET
请求参数：

参数	说明	类型	长度	是否必填
app_id	腾讯服务分配的 app_id	字符串	腾讯服务分配	必填，腾讯服务分配的 app_id
access_token	根据《整体登录鉴权流程》获取 access token	字符串	腾讯服务分配	根据《整体登录鉴权流程》获取 access token
type	ticket 类型	字符串	20	必填，默认值：NONCE (必须大写)
version	版本号	字符串	20	必填，默认值：1.0.0
user_id	当前使用用户的唯一标识。注意合作伙伴必须保证 user_id 的全局唯一。	字符串	30	必填

请求示例：
https://idasc.webank.com/api/oauth2/api_ticket?app_id=xxx&access_token=xxx&type=NONCE&version=1.0.0&user_id=xxx

响应：

{
"code":"0",
"msg":"请求成功",
"transactionTime":"20151022044027",
"tickets":[
{"value":"ticket_string",
"expire_in":"120"，
"expire_time":"20151022044027"}
]
}

注：

1.code不是0是表示获取失败，可以根据code和msg字段定位和调试。

2.expire_in为access_token的最大生存时间，单位秒，合作伙伴在判定有效期时以此为准。

3.expire_time为access_token失效的绝对时间，由于各服务器时间差异，不能使用作为有效期的判定依据，只展示使用。

4.access_token失效时，该access_token生成的ticket都失效

签名算法说明

生成签名算法，合作伙伴可以直接使用，签名参数的要求见各文档描述。

JAVA签名算法

```
public static String sign(List<String> values, String ticket) {
  if (values == null) {
  throw new NullPointerException("values is null");
  }

  values.removeAll(Collections.singleton(null));// remove null
  values.add(ticket);
  java.util.Collections.sort(values);

  StringBuilder sb = new StringBuilder();
  for (String s : values) {
  sb.append(s);
  }

  return Hashing.sha1().hashString(sb, Charsets.UTF_8).toString().toUpperCase();
}
```

注：
Hashing使用的的是com.google.common.hash.Hashing，版本是guava-18.0
用法：首先获取token和ticket再用ticket对数据进行签名
String accessToken = client.getAccessToken();//http请求
String ticket = client.getTicket(accessToken); //http请求
String sign = SignUtils.sign(data, ticket);

人脸验证SDK接入

启动SDK

生成签名

1.前置条件：必须按照获取NONCE ticket.

2.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：

参数	说明
appId	腾讯服务分配的app_id
userId	用户唯一标识
version	1.0.0
ticket	合作伙伴服务端缓存的tikcet,注意是NONCE类型，具体见Access Token获取
nonceStr	必须是32位随机数

生成一个 32 位的随机字符串(字母和数字) nonceStr(登录时也要用到)，将appId、userId、version连同ticket、nonceStr 共五个参数的值进行字典序排序。

1.将排序后的所有参数字符串拼接成一个字符串进行SHA1编码

2.SHA1编码后的40位字符串作为签名(sign)

3.签名算法参考章节7，示例如下：

请求参数：

appId = TIDA0001

userId= userID19959248596551

nonceStr = kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7T (必须为32位)

version = 1.0.0

ticket=XO99QfxIti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS

字典排序后的参数为：

[1.0.0, TIDA0001, XO99QfxIti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS ，kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7T, userID19959248596551]

拼接后的字符串为：

1.0.0TIDA0001XO99QfxIti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMSkHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7TuserID19959248596551

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。

SDK接入（Android）

1.前置条件

1.1 相机/录音/读取手机信息权限检测

SDK 需要用到相机/录音/读取手机信息权限，在 android6.0 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 android6.0 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刷脸，可以使整个刷脸体验更快更好。

1.2 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：

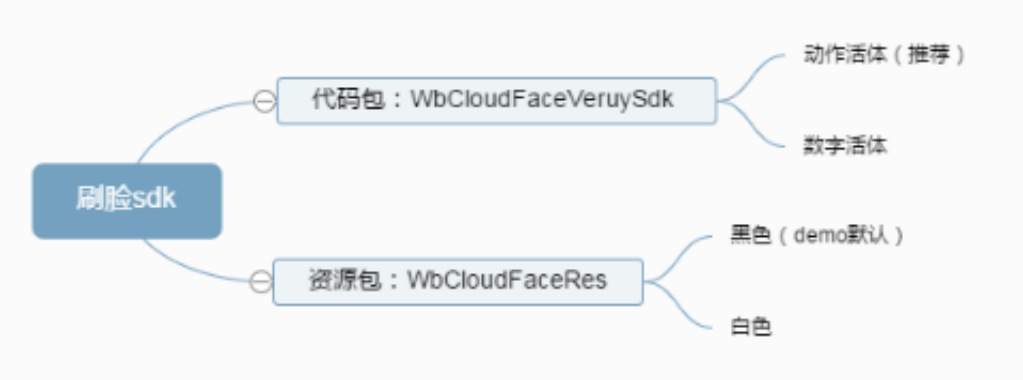
```
defaultConfig {
    applicationId "com.webank.testcloud.cloudfacetest"
    minSdkVersion 14
    targetSdkVersion 23
    versionCode 1
    versionName "1.0"

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

2.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，包括
代码包（WbCloudFaceVerifySdk）和
资源包（WbCloudFaceRes）
两个部分，缺一不可。其中代码包分为动作活体和数字活体两个模式，资源包分为黑色皮肤和白色皮肤(sdk皮

肤的设定，除了接入对应的 aar，还需要设定相关代码。详见 SDK 样式选择
。默认黑色皮肤，无需格外设置)，接入方可自由选择组合四个模式。



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')
 //3. 云刷脸皮肤资源包-可选择黑色/白色 默认黑色
```

```
 compile(name: 'WbCloudFaceResBlack', ext: 'aar')
 //compile(name: 'WbCloudFaceResWhite', ext: 'aar')
}
  }
```

3.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
的混淆规则，云公共组件的混淆规则及依赖的第三方库混淆规则。

3.1 云刷脸 sdk 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.wbcloudfaceverify2.tools.WbCloudFaceVerifySdk{
  public <methods>;
  public static final *;
}
-keep public class com.webank.wbcloudfaceverify2.tools.WbCloudFaceVerifySdk$*{
  *;
}
-keep public class com.webank.wbcloudfaceverify2.tools.ErrorCode{
  *;
}
-keep public class com.webank.wbcloudfaceverify2.ui.FaceVerifyStatus{

}
-keep public class com.webank.wbcloudfaceverify2.ui.FaceVerifyStatus$Mode{
  *;
}
-keep public class com.webank.wbcloudfaceverify2.tools.IdentifyCardValidate{
  public <methods>;
```

```
}
-keep public class com.tencent.youtulivecheck.**{
  *;
}
-keep public class com.webank.wbcloudfaceverify2.Request.*$*{
  *;
}
-keep public class com.webank.wbcloudfaceverify2.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-verify-proguard-rules.pro 拷贝到主工程根目录下,然后通过"-include webank-cloud-face-verify-rules.pro"
加入到您的混淆文件中。

3.2 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。

3.3 云刷脸依赖的第三方库的混淆规则

```
-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。

4.调用 SDK 接口

SDK 代码调用的入口为：
com.webank.wbcloudfaceverify2.tools.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 供后续增加接口字段
  */
  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（参考要求详见下一节描述）：

```
//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（他是是一个 Serializable 对象）。

5.接口参数说明

参数	说明	类型	长度	是否必填
userName	用户姓名	String	20	必填
idType	用户证件类型	String	2	必填，01 为身份证
userId	用户证件号码	String	18	必填，18 位身份证号

参数	说明	类型	长度	是否必填
agreementNo	订单号	String	32	必填，合作方订单的唯一标识
clientIp	用户 ip 信息	String	30	必填,格式为”ip=xxx.xxx.xxx.xxx;”；示例：”ip=58.60.124.0”。
gps	用户 gps 信息	String	30	必填,格式为”lgt= xxx;lat=xxx;”；示例：“lgt=22.5044;lat=113.9537“
openApiAppId	腾讯服务分配的app_id	String	腾讯服务分配	必填，腾讯服务分配的app_id
openApiAppVersion	接口版本号	String	20	必填，默认填 1.0.0
openApiNonce	32 位随机字符串	String	32	必填，每次请求需要的一次性nonce
openApiUserId	User Id	String	30	必填，每个用户唯一的标识
openApiSign	合作方后台服务器通过ticket计算出来的签名信息	String	40	必填
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	必填
verifyMode	刷脸类型： 动作活体 FaceVerifyStatus.Mode.MIDDLE 数字活体 FaceVerifyStatus.Mode.ADVANCED	FaceVerifyStatus.Mode		必填
keyLicence	腾讯给合作方派发的li	String		必填

参数	说明	类型	长度	是否必填
	cence			

6.个性化参数设置

WbCloudFaceVerifySdk.init()里 Bundle data，除了必须要传的 InputData 对象(详情见上节)之外，还可以由合作方方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。

6.1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 中点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);
```

6.2 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 中点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);
```

6.3 SDK 样式选择

合作方可以选择 sdk 样式,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 中点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（需要与白色资源包一起配合使用）
 data.putString(WbCloudFaceVerifySdk.COLOR_MODE, WbCloudFaceVerifySdk.WHITE);
```

6.4 是否自带对比源数据

合作方可以选择给 sdk 送上自带的对比源数据进行对比。合作方可以上送两类照片，一类是水纹照，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 2M，经过编码后的图片 String 大小不可超过 3M。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 中点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);
 //上送自带的数据源信息，照片类型与照片 string 缺一不可
//上送照片类型，1 是水纹照 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);
```

6.5 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 中点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);
```

6.7 接入示例

```
# 在 MainActivity 中点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，默认暗黑模式，此处设置为明亮模式
 data.putString(WbCloudFaceVerifySdk.COLOR_MODE, WbCloudFaceVerifySdk.WHITE);
 //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);
```

7.接入示例

```
# 在 MainActivity 中点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，默认暗黑模式，此处设置为明亮模式
  data.putString(WbCloudFaceVerifySdk.COLOR_MODE, WbCloudFaceVerifySdk.WHITE);
  //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.FaceVerifyResultForSecureListener() {
//由 FaceVerifyResultListener 返回刷脸结果
@Override
public void onFinish(int resultCode, boolean nextShowGuide, String faceCode, String faceMsg, String
sign, Bundle extendData) {
// resultCode为0，则刷脸成功；否则刷脸失败
if (resultCode == 0) {
Log.d(TAG, "刷脸成功！");
} else {
Log.d(TAG, "刷脸失败！);
}
}
});
}
@Override
public void onLoginFailed(String errorCode, String errorMsg) {
//登录失败
...
}
```

8.错误码描述

8.1 刷脸登录错误码

```
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
```

8.2 刷脸错误码

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
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

SDK接入（iOS）

1.接入配置

WeBankService SDK 最低支持到 iOS8.0(iOS7 系统可以编译但是无法使用),请在构建项目时候注意.

1)引用资源文件 WBFaceV2Pics.bundle,ufa.bundle,youtubeauty.bundle 到项目

2)引用 WBCloudFaceVerifySDK.framework, YTFaceSDK.framework, NextCV.framework 到项目

3)SDK 依赖系统的以下框架:

libz.tbd

,

Security.framework

,

MobileCoreServices.framework

,

Accelerate.framework

,

SystemConfiguration.framework

,

libc++.tbd

,

CoreTelephony.framework

,

AVFoundation.framework

,

AudioToolbox.framework

,

CoreMedia.framework

. 需要在

BuildPhases->Link Binary With Libraries

中添加,可以参考 Demo
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,Privacy - Photo Library Usage Description

5)
6)需要在

BuildSettings->Other Linker Flags

中设置

-ObjC

-force_load

$(PROJECT_DIR)/

该 sdk 在项目中的具体路径

/NextCV.framework/NextCV

$(PROJECT_DIR)/

该 sdk 在项目中的具体路径

/YTFaceSDK.framework/YTFaceSDK

2.调用 SDK 接口

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:

```
// SDK版本号
UIKIT_EXTERN NSString *const WBCloudFaceVerifySDKVersion;

// SDK界面风格等配置字段 key 值
UIKIT_EXTERN NSString *const WBFaceExternalIsShowSuccessPageKey;//BOOL
UIKIT_EXTERN NSString *const WBFaceExternalIsShowFailurePageKey;// BOOL
UIKIT_EXTERN NSString *const WBFaceExternalIsLightnessKey; // BOOL
UIKIT_EXTERN NSString *const WBFaceExternalIsShowGuidePageKey; //BOOL

// SDK 自带比对源功能相关配置参数
UIKIT_EXTERN NSString *const WBFaceExternalIsUsingSourceImageKey; //BOOL
```

```
UIKIT_EXTERN NSString *const WBFaceExternalIsUsingHDImageKey; //BOOL
(图像是否为高清图像/或者公安网文图像)
UIKIT_EXTERN NSString *const WBFaceExternalSourceImageKey;
//UIImage(自带比对源图片大小不能超过 2M)

// SDK 在运行结束退出时候会发出通知. 具体的通知内容可以见 delegate 方法
wbfaceVerifyCustomerServiceDidFinished:中的注释
UIKIT_EXTERN NSString *const WBFaceVerifyCustomerServiceDidFinishedNotification;

//登录的错误码
typedef NS_ENUM(NSInteger, WBFaceVerifyLogin) {
  WBFaceVerifyLogin_ERROR = -10000, // 登录请求返回报错
  WBFaceVerifyLogin_PARAMS_ERROR = -20000,// 登录请求传入参数有误
  WBFaceVerifyLogin_NORESPONSE_ERROR = -30000 // 登录请求网络错误
};

typedef NS_ENUM(NSInteger, WBFaceVerifySDKErrorCode) {
  WBFaceVerifySDKErrorCode_SUCESS = 0,// 身份认证成功
  WBFaceVerifySDKErrorCode_FAILURE = 1,// 身份认证出错
  WBFaceVerifySDKErrorCode_CANCELLED = 2,// 用户取消认证
};

typedef NS_ENUM(NSInteger,WBFaceVerifyType){
  WBFaceVerifyTypeMiddle,
  WBFaceVerifyTypeAdvanced,
};

typedef void (^WBFaceLoginSuccessBlock)(void);
//登录过程中 loginCode 细分错误码 --- -2 表示登录时网络错误, -1 表示登录时 webank 后台返回出错, 0
表示登录通用默认码, 其他数字透传 webank 后台错误码(请参考后台文档)
typedef void (^WBFaceLoginFailureBlock)(WBFaceVerifyLogin errorCode, NSString *loginCode,
NSString *message);

@interface WBFaceUserInfo : NSObject
```

```
@property (nonatomic,copy) NSString *orderNo; // 订单号（合作方上送，每次唯一）
@property (nonatomic,copy) NSString *name; // 姓名
@property (nonatomic,copy) NSString *idType; // 证件类型
@property (nonatomic,copy) NSString *idNo; // 证件号码

/**
 判断 UserInfo 信息是否满足要求,内部只判断属性是否有 nil
 */
-(BOOL)isPropertyRight;
@end

/**
  处理刷脸回调
 */
@class WBFaceVerifyCustomerService;
@protocol WBFaceVerifyCustomerServiceDelegate <NSObject>

/**
 注意:
 1. 如果实现该代理方法(wbfaceVerifyServiceGetViewController:),则向 WBFaceVerifyCustomerService SDK 中通过代理传入一个 ViewController,在 sdk 登录成功以后会在该 viewController 通过 presentViewControllerxxxx 方法拉起人脸认证页面.

 2. 如果没有实现该代理方法(wbfaceVerifyServiceGetViewController:), 那么 SDK 会创建一个 UIWindow 覆盖在当前界面,并在新创建的 UIWindow 界面进行人脸认证,并且可以通过实现 wbfaceVerifyServiceGetWindowLevel 代理方法,传入创建的 UIWindow的windowLevel, 传入的 windowLevel 必须是 1~999, 默认情况如果不实现 wbfaceVerifyServiceGetWindowLevel 方法,windowLevel = UIWindowLevelNormal + 1

 */
@optional
- (UIViewController *)wbfaceVerifyServiceGetViewController:(WBFaceVerifyCustomerService *)service;
@optional
- (NSUInteger)wbfaceVerifyServiceGetWindowLevel;
```

```
/**
 * 刷脸身份认证的回调方法,带结果签名 sign 的回调
 *
 * @param errorCode iOS SDK定义的概要错误码 -- 重要信息: 身份认证成功与失败都在这里显示
 * @param faceCode 细分错误码 --- -2 表示网络错误, -1 表示 webank 后台返回出错, 0 表示通用默认码,
其他数字透传 webank 后台错误码(请参考后台文档)
 * @param faceMsg 身份认证错误的相关提示信息
 * @param sign 当前身份认证结果的签名信息


  注意!!!!!!!!!!!!

  可以不实现该 sdk 方法,通过注册 WBFaceVerifyCustomerServiceDidFinishedNotification
这个通知,通过通知的n.userInfo 同样可以拿到 errorCode, faceCode, faceMsg, sign 以及 orderNo(之前传入
sdk 的订单号)等刷脸结果返回的信息.
 */
@optional
-(void)wbfaceVerifyCustomerServiceDidFinished:(WBFaceVerifySDKErrorCode)errorCode
faceCode:(NSString *)faceCode faceMsg:(NSString *)faceMsg sign:(NSString *)sign;

/**
 * SDK 扩展字段
 * WBFaceExternalIsShowSuccessPageKey BOOL 表示是否显示身份认证成功的结果页(默认情况显示 -
YES)
 * WBFaceExternalIsShowFailurePageKey BOOL 表示是否显示身份认证失败的结果页(默认情况显示 - YES)
 * WBFaceExternalIsLightnessKey BOOL
表示是否使用明亮主题的人脸验证界面风格(默认状态是暗黑主题页面风格)
 * WBFaceExternalIsShowGuidePageKey BOOL 表示是否显示身份认证前的引导页(默认情况显示 - YES)
 *
 * 在 delegate 实现中实现该方法
 -(NSDictionary *)wbfaceVerifyServiceGetExternalParams:(WBFaceVerifyCustomerService *)service{
  return @{
  WBFaceExternalIsShowFailurePageKey : @(YES),
```

```
 WBFaceExternalIsShowSuccessPageKey : @(YES),
 WBFaceExternalIsLightnessKey: @(YES),
 WBFaceExternalIsShowGuidePageKey, @(YES),
 WBFaceExternalIsUsingSourceImageKey, @(NO),
 };
}

 注意: 该方法和 -(void)startWBFaceServiceWithUserid:(NSString *)userid nonce:(NSString *)nonce
sign:(NSString *)sign appid:(NSString *)appid userInfo:(WBFaceUserInfo *)userInfo
apiVersion:(NSString *)apiVersion faceverifyType:(WBFaceVerifyType)type licence:(NSString *)licence
success:(WBFaceLoginSuccessBlock)success failure:(WBFaceLoginFailureBlock)failure
externalParams:(NSDictionary *)extenalParams; 方法中的最后一个参数 extenalParams 功能一致,
可以不用设置delegate, 直接使用该StartWBFaceServiceXXXX方法,在最后一个参数 extenalParams
传入配置参数.
 */
@optional
-(NSDictionary *)wbfaceVerifyServiceGetExternalParams:(WBFaceVerifyCustomerService *)service;

/**
 *********************************************************************
 ***************************** 废弃接口 ******************************
 *********************************************************************
- (UIViewController *)getViewController NS_DEPRECATED_IOS(2_0, 2_0,
"该回调方法是前向兼容方法,请使用 wbfaceVerifyServiceGetViewController: 方法");
-(void)wbfaceVerifyCustomerServiceDidFinished:(WBFaceVerifySDKErrorCode)errorCode
faceCode:(NSString *)faceCode faceMsg:(NSString *)faceMsg NS_DEPRECATED_IOS(2_0, 2_0,
"该回调方法是前向兼容方法方法,请使用
wbfaceVerifyCustomerServiceDidFinished:faceCode:faceMsg:sign: 方法");
 */
@end

@interface WBFaceVerifyCustomerService : NSObject
@property (nullable,nonatomic,weak) id<WBFaceVerifyCustomerServiceDelegate> delegate;
```

```
/*
 * SDK全局单例,请使用此单例.不要使用init创建对象
 */
+ (instancetype)sharedInstance;

/**
 * 调用SDK服务核心方法 1
 *
 * @param userid userid 每个用户唯一的标识
 * @param nonce 每次请求需要的一次性 nonce
 * @param sign 对 nonce,userid 等重要信息的签名数据
 * @param appid 每个与 webank 合作分配的 appid
 * @param userInfo 用户信息对象,请参考前面定义的内容
 * @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 type 身份认证的类型- 简单模式,中级模式,高级模式
 * @param licence webank 给合作方派发的 licence
 * @param success 调用 sdk 登录成功回调.请在该回调方法中关闭 loading,并且在 success block
执行以后,sdk 为拉起人脸认证页面
 * @param failure 调用 sdk 登录失败时回调.请在该回调方法中关闭 loading,处理错误逻辑.
 */
-(void)startWBFaceServiceWithUserid:(NSString *)userid
  nonce:(NSString *)nonce
  sign:(NSString *)sign
  appid:(NSString *)appid
  userInfo:(WBFaceUserInfo *)userInfo
  apiVersion:(NSString *)apiVersion
  faceverifyType:(WBFaceVerifyType)type
  licence:(NSString *)licence
  success:(WBFaceLoginSuccessBlock)success
  failure:(WBFaceLoginFailureBlock)failure;


/**
 * 调用SDK服务核心方法2
```

```
 *
 * @param userid userid 每个用户唯一的标识
 * @param nonce 每次请求需要的一次性 nonce
 * @param sign 对 nonce,userid 等重要信息的签名数据
 * @param appid 每个与 webank 合作分配的 appid
 * @param userInfo 用户信息对象,请参考前面定义的内容
 * @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 type 身份认证的类型- 简单模式,中级模式,高级模式
 * @param licence webank 给合作方派发的 licence
 * @param success 调用 sdk 登录成功回调.请在该回调方法中关闭 loading,并且在 success block
执行以后,sdk 为拉起人脸认证页面
 * @param failure 调用 sdk 登录失败时回调.请在该回调方法中关闭 loading,处理错误逻辑.
 * @param extenalParams
 * 参数与"-(NSDictionary *)wbfaceVerifyServiceGetExternalParams:(WBFaceVerifyCustomerService
*)service"接口提供的返回参数功能相同
  传入字典如下:
  @{
  WBFaceExternalIsShowFailurePageKey : @(YES),
  WBFaceExternalIsShowSuccessPageKey : @(YES),
  WBFaceExternalIsLightnessKey: @(YES),
  WBFaceExternalIsShowGuidePageKey, @(YES),
  }
 */
-(void)startWBFaceServiceWithUserid:(NSString *)userid
  nonce:(NSString *)nonce
  sign:(NSString *)sign
  appid:(NSString *)appid
  userInfo:(WBFaceUserInfo *)userInfo
  apiVersion:(NSString *)apiVersion
  faceverifyType:(WBFaceVerifyType)type
  licence:(NSString *)licence
  success:(WBFaceLoginSuccessBlock)success
  failure:(WBFaceLoginFailureBlock)failure
  externalParams:(NSDictionary *)extenalParams;
```

@end

3.接口参数说明

参数	说明	类型	长度	是否必填
userid	用户唯一的标识	NSString	30	必填，必须保证全局唯一
nonce	32 位随机字符串	NSString	32	必填(生成方式参考后台文档)
sign	合作方后台服务器通过ticket计算出来的签名信息	NSString	40	必填(生成方式参考后台文档)
appid	腾讯服务分配的app_id	NSString	腾讯服务分配	必填
apiVersion	接口版本号	NSString	默认填 1.0.0	必填
licence	腾讯给合作方派发的licence	NSString	绑定bundleid	必填
userInfo	用户信息	WBFaceUserInfo	Class类型	必填
orderNo	订单号	NSString	32 位	必填(参考后台文档)
name	客户姓名	NSString	20	必填(身份证姓名)
idType	证件类型	NSString	2	01 为身份证，必填
idNo	18 位身份证号	NSString	18	必填
Type	模式类型	WBFaceVerifyType		中级模式：WBFaceVerifyTypeMiddle, 高级模式：WBFaceVerifyTypeAdvanced,

4.个性化参数设置

WBFaceVerifyCustomerServiceDelegate 的回调方法中,有如下扩展配置选项,直接参考 sdk 头文件注释即可

@optional

```
/**
 * SDK 扩展字段
 * WBFaceExternalIsShowSuccessPageKey BOOL 表示是否显示身份认证成功的结果页
 * WBFaceExternalIsShowFailurePageKey BOOL 表示是否显示身份认证失败的结果页(默认情况显示 – YES)
 * WBFaceExternalIsLightnessKey BOOL
表示是否使用明亮主题的人脸验证界面风格(默认状态是暗黑主题页面风格)
 * WBFaceExternalIsShowGuidePageKey BOOL 表示是否展示引导页页面
 * 在 delegate 实现中实现该方法
 -(NSDictionary *)wbfaceVerifyServiceGetExternalParams:(WBFaceVerifyCustomerService *)service{
  return @{
  WBFaceExternalIsShowSuccessPageKey : @(YES),
  WBFaceExternalIsShowFailurePageKey: @(YES),
  WBFaceExternalIsLightnessKey: @(YES),
  WBFaceExternalIsShowGuidePageKey: @(YES),
  };
 }
 */
-(NSDictionary *)wbfaceVerifySerivceGetExternalParamsWBFaceVerifyCustomerService:)service;
```

人脸验证微信H5接入

合作方后台上送身份信息

1.生成签名

1.1 前置条件：必须按照 SIGN ticket获取.

1.2 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

参数	说明
appId	腾讯服务分配的 app_id
orderNo	订单号，本次人脸验证合作伙伴上送的订单号，唯一标识。
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）
version	1.0.0
api ticket	合作伙伴服务端缓存的 tikcet,注意是 SIGN 类型，具体参考 SIGN ticket获取

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

示例代码及用法：

请求参数：

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 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	是否必填
webankAppId	分配给服务使用方的appId	String	分配给服务使用方	必填
orderNo	订单号，由合作方上送，每次唯一，不能超过 32 位	String	订单号，由合作方上送，每次唯一，不能超过 32 位	必填
name	姓名	String		必填

参数	说明	类型	长度	是否必填
idNo	证件号码	String		必填
userId	用户 ID ，用户的唯一标识（不要带有特殊字符）	String		必填
sourcePhotoStr	比对源照片	BASE64String	比对源照片的 BASE64 String 长度：3145728	非必填， 注意：原始图片不能超过 2M,且必须为 JPG 或 PNG 格式。 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。 为空：根据身份证号 + 姓名使用权威数据源比对
sourcePhotoType	比对源照片类型	String	1	1：水纹正脸照2：高清正脸照
version	1.0.0	String	20	必填，默认值：1.0.0
sign	使用上面生成的签名	String	40	必填

响应返回参数：

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

公众号启动H5刷脸

1. 生成签名

1. 前置条件：必须按照 NONCE ticket 获取.
2. 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：

参数	说明
webankAppId	WebankAppId，由腾讯指定
version	接口版本号 1.0.0
nonce	随机数 32 位随机串（字母+数字组成的随机数）
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。
h5faceId	h5/geth5faceid 接口返回的唯一标识
userId	用户 ID ，用户的唯一标识（不要带有特殊字符）
api ticket	合作伙伴服务端缓存的 tikcet,注意是 NONCE 类型

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

webankAppId= appId001 userId= userID19959248596551 nonce =
kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7T (必须为 32 位) version = 1.0.0 h5faceId = bwiwe1457895464
orderNo = aabc1457895464
ticket=zxc9Qfxlti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS

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 刷脸

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 刷脸。
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	是否必填
webankAppId	WebankAppId，由腾讯指定	String	腾讯服务分配	必填
version	接口版本号	String	20	必填，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	必填
orderNo	订单号，由合作方上送，每次唯一，此信	字符串	32	必填

参数	说明	类型	长度	是否必填
	息为本次人脸验证上送的信息。			
h5faceId	h5/geth5faceid 接口返回的唯一标识。	string	32	必填
url	人脸识别完成后回调第三方 URL	string		必填，H5 刷脸完成后的回调 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%3A%2F%2Fidaop.webank.com)
resultType	是否显示结果页面（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		非必填
userId	用户 ID ，用户的唯一标识（不要带有特殊字符）	string		必填

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	必填

H5刷脸结果跳转

刷脸 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 URL:

https://xxx.com/xxx?code=xxxx&orderNo=xxxx&h5faceId=xxxx&signature=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》说明进行签名校验，确保返回结果的安全性。

   请求方法:GET

请求参数：

参数	说明	类型	长度
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
signature	对 url 参数 appId、oderNo 和 sign ticket 的签名。具体见的签名生成和校验规则	字符串	40

人脸验证微信小程序接入

合作方后台上送身份信息

注：小程序的接入和鉴权整体流程，请参考活体识别 H5 登录鉴权

1.生成签名

1. 前置条件：必须按照必须按照 SIGN ticket获取.
2. 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：

参数	说明
appId	腾讯服务分配的 app_id
orderNo	订单号，本次人脸验证合作伙伴上送的订单号，唯一标识。
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）。
version	1.0.0
api ticket	合作伙伴服务端缓存的 tikcet,注意是 SIGN 类型，具体见SIGN ticket获取来获取规则。

1. 将

webankAppId, orderNo, name, idNo, userId, version，ticket

（SIGN 类型）共七个参数的值进行字典序排序。将排序后的所有参数字符串拼接成一个字符串进行 SHA1 编码。

SHA1 编码后的 40 位字符串作为签名 (sign)

示例代码及用法：

请求参数：

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
.04300000000000appId001duSz9ptwyW1Xn7r6gYItxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNN
DoeorderNo19959248596551testNameuserID19959248596551

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 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	是否必填
webankAppId	分配给服务使用方的 appId	String	分配给服务使用方	必填
orderNo	订单号，由合作方上送，每次唯一，不能超过 32 位	String	订单号，由合作方上送，每次唯一，不能超过32 位	必填
name	姓名	String		必填
idNo	证件号码	String		必填
userId	用户ID ，用户的唯一标识（不要带有特殊字符）	String		必填
sourcePhotoStr	比对源照片	BASE64 String	比对源照片的 BASE64 String 长度：3145728	

启动小程序刷脸

1.生成签名

1. 前置条件：必须按照 NONCE ticket获取来获取规则。
2. 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：

参数	说明
webankAppId	WebankAppId，由腾讯指定
version	接口版本号1.0.0
nonce	随机数 32位随机串（字母+数字组成的随机数）
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。
h5faceId	h5/geth5faceid接口返回的唯一标识
userId	用户ID ，用户的唯一标识（不要带有特殊字符）
api ticket	合作伙伴服务端缓存的 tikcet,注意是 NONCE 类型，具体见 NONCE ticket获取来获取规则。

3.生成一个 32 位的随机字符串(字母和数字) nonce (登录时也要用到)，将

webankAppId 、userId、orderNo、version、h5faceId

连同

ticket、nonce

共 7 个参数的值进行字典序排序。
将排序后的所有参数字符串拼接成一个字符串进行 SHA1 编码
SHA1 编码后的 40 位字符串作为签名 (sign)
示例代码及用法：
请求参数：

webankAppId= appId001
userId= userID19959248596551
nonce = kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7T (必须为32位)

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.启动小程序刷脸

1. 合作方小程序上送

   h5faceId

   以及

sign

，调用微信打开小程序 API 启动刷脸小程序。

2. 请求方法:

wx.navigateToMiniProgram()

此方法是微信小程序提供的 API，详细信息请参考官方文档。

3. 请求参数：

参数	说明	类型	长度	是否必填
appId	刷脸小程序appId	String	20	

小程序刷脸结果跳转

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，拿到刷脸结果。

注意：

1)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，详情可以参考官方文档。

2)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
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
signature	对url 参数 appId、oderNo 和 sign ticket 的签名。具体见10.1的签名生成和校验规则	字符串	40

人脸验证结果

方式一：前端获取结果验证签名

1.合作伙伴APP端或H5收到远程身份验证SDK返回的带签名结果。

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。

1.1 合作方后台生成签名

合作方根据本次人脸验证的如下参数生成签名：

参数	说明
app_id	腾讯服务分配的app_id
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。
api ticket	合作伙伴服务端缓存的tikcet,注意是sign 类型

将app_id、order_no、连同ticket(SIGN)共三个参数的值进行字典序排序

将排序后的所有参数字符串拼接成一个字符串进行SHA1编码

SHA1编码后的40位字符串作为签名(sign)

示例代码及用法

示例：

请求参数：

app_id = appId001

order_no= userID19959248596551

ticket= duSz9ptwyW1Xn7r6gYItxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoe

字典排序后的参数为：

[appId001, duSz9ptwyW1Xn7r6gYItxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoe, test1480921551481]

拼接后的字符串为：

appId001duSz9ptwyW1Xn7r6gYItxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoetest1480921551481

计算 SHA1 得到签名：

B02CEBEB07F792B2F085E8CB1E7BA9EC19284F54

该字串就是最终生成的签名(40位)，不区分大小写。

1.2 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的刷脸结果。
注：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

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

合作方根据本次人脸验证的如下参数生成签名：

参数	说明
app_id	腾讯服务分配的 app_id
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。
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，具体详见 SING ticket获取
nonceStr	32 位随机字符串,字母和数字

生成一个 32 位的随机字符串(字母和数字) nonceStr，将 app_id、order_no、version 连同 ticket、nonceStr 共五个参数的值进行字典序排序再 SHA1 编码生成签名。具体见签名算法说明。

2. 身份认证查询接口

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	是否必填
app_id	腾讯服务分配的app_id	字符串	腾讯服务分配	必填，腾讯服务分配的app_id
version	版本号	字符串	20	必填，默认值：1.0.0
nonce	随机数	字符串	32	必填
order_no	订单号	字符串	32	必填，合作方订单的

参数	说明	类型	长度	是否必填
				唯一标识
sign	签名值	字符串	40	必填，使用上面生成的签名。
get_file	是否需要获取刷脸的视频和文件	字符串	1	非必填，值为1则返回文件；其他则不返回。

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

返回参数：

参数	类型	说明
code	string	“0” 说明身份验证成功
msg	string	返回结果描述
bizSeqNo	string	业务流水号
orderNo	string	订单编号
idNo	string	证件号码
idType	string	证件类型
name	Base 64 string	姓名
liveRate	Base 64 string	活体检测得分
similarity	Base 64 string	人脸比对得分
occurredTime	string	进行刷脸的时间
photo	string	刷脸时的照片，base64 位编码
video	string	刷脸时的视频，base64 编码

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。

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,png.视频格式解码后一般为 mp4。

活体识别微信H5接入

公众号启动 H5 活体识别

一、生成签名

1.前置条件：必须按照 获取NONCE ticket。

2.合作方根据本次活体识别的如下参数生成签名,需要签名的参数信息如下：

参数	说明
appId	腾讯服务分配的app_id
orderNo	订单号，本次人脸验证合作伙伴上送的订单号，唯一标识。
nonce	随机数 32位随机串（字母+数字组成的随机数）
userId	用户ID ，用户的唯一标识（不要带有特殊字符）
version	1.0.0
api ticket	合作伙伴服务端缓存的tikcet,注意是NONCE类型，具体见获取NONCE ticket。

生成一个 32 位的随机字符串(字母和数字)

nonce

(登录时也要用到)，将

webankAppId 、userId、orderNo、version

连同

ticket、nonce

共6个参数的值进行字典序排序。

将排序后的所有参数字符串拼接成一个字符串进行

SHA1

编码

SHA1

编码后的 40 位字符串作为签名 (sign)
示例代码及用法：
请求参数：

webankAppId= appId001
userId= userID19959248596551
nonce = kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7T (必须为32位)
version = 1.0.0
orderNo = aabc1457895464
ticket=zxc9QfxIti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS

字典排序后的参数为：

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

拼接后的字符串为：

1.0.0aabc1457895464appId001kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7T
userID19959248596551zxc9QfxIti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS

计算 SHA1 得到签名：

5E034EF71E90E5F5FB072CDBB259FFF25A938B03
该字串就是最终生成的签名(40位)，不区分大小写。

二、启动H5活体识别

合作方公众号上送sign,后台校验sign通过之后重定向到H5活体识别。
请求URL:

https://ida.webank.com/api/wx/livelogin

请求方法:GET
请求参数：

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	必填			
version	接口版本号	String	20	必填,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	必填
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	必填
url	人脸验证完成后回调第三方URL	string		

H5 活体识别结果跳转

1. 活体识别 h5 结果返回跳转第三方 url 带唯一标识、订单号、识别分数、验证结果、签名。

   请求URL:

   https://xxx.com/xxx?code=xxxx&msg=xxxxx&orderNo=xxxx&live_rate=xxxx&&signature=xxxx

   注意：

   xxxx.com

   为合作方上送的 URL。

合作方根据11.1说明进行签名校验，确保返回结果的安全性。

请求方法:GET

返回参数：

参数	说明	类型	长度
code	活体识别结果的返回码，0表示活体识别成功，其他错误码标识失败，具体错误码信息见章节16。		
msg	活体识别结果描述	字符串	
orderNo	订单号，由合作方上送，每次唯一，此信息为本次活体识别上送的信息。	字符串	32
signature	对url参数appId、oderNo和sign ticket 的签名。		
具体见10.1的签名生成和校验规则	字符串	40	

参数	说明	类型	长度
live_rate	活体识别结果分数	字符串	40

服务端查询结果

1.合作伙伴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H5获取结果后继续后续流程。

一、合作方后台生成签名

合作方根据本次人脸验证的如下参数生成签名：

参数	说明
app_id	腾讯服务分配的app_id
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。
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，具体见6.1获取规则
nonceStr	32位随机字符串,字母和数字

生成一个 32 位的随机字符串(字母和数字) nonceStr，将app_id、order_no、version 连同ticket、nonceStr共五个参数的值进行字典序排序再SHA1编码生成签名。具体签名算法见章节7。

二、活体识别查询接口

请求URL：https://idasc.webank.com/api/server/getLiveResult

请求方法:GET

请求参数：

参数	说明	类型	长度	是否必填
app_id	腾讯服务分配的app_id	字符串	腾讯服务分配	必填，腾讯服务分配的app_id
version	版本号	字符串	20	必填，默认值：1.0.0
nonce	随机数	字符串	32	必填
order_no	订单号	字符串	32	必填，合作方订单的

参数	说明	类型	长度	是否必填
				唯一标识
sign	签名值	字符串	40	必填，使用上面生成的签名。
get_file	是否需要获取活体识别的视频和文件	字符串	1	非必填，值为1则返回文件；其他则不返回。

请求示例：

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

返回参数：

参数	类型	说明
code	string	“0”说明活体成功
msg	string	返回结果描述
bizSeqNo	string	业务流水号
orderNo	string	订单编号
liveRate	Base 64 string	活体检测得分
occurredTime	string	进行活体识别的时间
photo	string	活体识别时的照片，base64位编码
video	string	活体识别时的视频，base64编码

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。

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,png.视频格式解码后一般为mp4。

身份证识别SDK接入

SDK接入（Android）

1. 前置条件

相机/读取手机信息权限检测
SDK 需要用到相机权限/读取手机信息权限。
1)Android6.0 及以上系统
SDK 运行时提示权限，需要用户授权。
2)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进行。考虑到 SDK
的使用时间很短，快速频繁开关相机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 Android6.0 以下系统上，我们建议合作方在拉起 SDK 前，帮助 SDK 做相机/读取手机信息权限检测，提示用户确认打开了这项权限后再进行身份证识别，可以使整个身份证识别体验更快更好。

2. 接入配置

OCR SDK（WbCloudOcr）最低支持到 Android API 14: Android 4.0(ICS)，请在构建项目时注意。
WbCloudOcr将以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-v4
```

```
 compile 'com.android.support:appcompat-v4:23.1.1'
  //1. 云Ocr SDK
  compile(name: 'WbCloudOcrSdk-proRelease-v1.2.4-5cf1801', ext: 'aar')
  //2.云公共组件
compile(name: 'WbCloudNormal-release-v3.0.8-f0cefef', ext: 'aar') }
```

3. 混淆配置

云OCR产品的混淆规则分为三部分，分别是云OCR
SDK的混淆规则，云公共组件的混淆规则及依赖的第三方库混淆规则。

1. 云 OCR SDK 的混淆规则

```
#########################云 ocr 混淆规则 ocr-BEGIN#############################
-keepattributes InnerClasses
-keep public class com.webank.mbank.ocr.WbCloudOcrSDK{
  public <methods>;
  public static final *;
}
-keep public class com.webank.mbank.ocr.WbCloudOcrSDK$*{
  *;
}

-keep public class com.webank.mbank.ocr.tools.ErrorCode{
  *;
}

-keep public class com.webank.mbank.ocr.net.*$*{
  *;
}
-keep public class com.webank.mbank.ocr.net.*{
  *;
}
```

##########云 ocr 混淆规则 ocr-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-ocr-proguard-rules.pro拷贝到主工程根目录下,然后通过"-include webank-cloud-ocr-rules.pro" 加入到您的混淆文件中。

2.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{
  *;
}

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

#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。

3. 云 OCR 依赖的第三方库的混淆规则

云 OCR 依赖的第三方库的混淆规则全部内容变更为：

```
######################云OCR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OCR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。

4. 调用 SDK 接口

SDK 代码调用的入口为
com.webank.mbank.ocr.WbCloudOcrSDK 这个类。

```
public class WbCloudOcrSDK{

/**
* 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 synchronized WbCloudOcrSDK getInstance() {
  // ...
  }
```

```
/**
* 在使用 SDK 前先初始化，传入需要的数据 data，由 OcrLoginListener 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,OcrLoginListener loginListerner){
// ...
}
  /**
  * 登录成功后，调用此函数拉起sdk页面
  * @param context 拉起 SDK 的上下文
  * @param idCardScanResultListener 返回到第三方的接口
  * @param type 进入 SDK 的模式，参数是枚举类型
  */
  public void startActivityForOcr(Context context,IDCardScanResultListener,WBOCRTYPEMODE type){
  // ...
 }
/**
  * 登录回调接口
  */
public interface OcrLoginListener {
  void onLoginSuccess();
  void onLoginFailed(String errorCode, String errorMsg);
  }

/**
  * 退出 SDK,返回第三方的回调,同时返回 ocr 识别结果
  */
public interface IDCardScanResultListener{
  /**
  * @RARAM exidCardResult SDK 返回的识别结果的错误码
* @RARAM exidCardResult SDK 返回的识别结果的错误信息
  */
  void onFinish(String errorCode, String errorMsg);
```

WbCloudOcrSdk.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，参加要求见下一章节描述：

```
//这些都是WbCloudOcrSdk.InputData对象里的字段，是需要传入的数据信息
String orderNo; //订单号
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
```

以上参数被封装在 WbCloudOcrSdk.InputData 对象中（他是一个 Serializable 对象）。
EXIDCardResult 代表 SDK 返回的识别身份证的结果，该类属性如下所示：

```
public int type;//拉起 SDK 的模式所对应的int 值，也就是 startActivityForOcr 方法中 WBOCRTYPEMODE
type 的枚举值 value
  // 识别人像面返回的信息
  public String cardNum; //身份证号码
  public String name;//姓名
  public String sex;//性别
  public String address;//住址
  public String nation;//民族
  public String birth;//出生年月日
  public String frontFullImageSrc;// 人像面图片存放路径

  //识别国徽面返回的信息
  public String office;//签发机关
  public String validDate;//有效期限
  public String backFullImageSrc;//国徽面图片存放路径
//每次网络请求都会返回的信息
public String sign;//签名
public String orderNo; //订单号
public String ocrId;//识别的唯一标识
```

```
public String warning;//识别的警告码
```

4.1 登录接口

```
/**
 * 登录回调接口
 */
public interface OcrLoginListener {
 void onLoginSuccess();//登录成功
 /**
 * @PARAM errorCode 登录失败错误码
 * @PARAM errorMsg 登录失败错误信息
 */
 void onLoginFailed(String errorCode, String errorMsg);
 }
```

4.2 返回第三方接口

```
/**
 * 退出 SDK,返回第三方的回调,同时返回 ocr 识别结果
 */
public interface IDCardScanResultListener{
 /**
 * 退出 SDK,返回第三方的回调,同时返回 ocr 识别结果
 * @param errorCode 返回错误码，识别成功返回 0
 * @param errorMsg 返回错误信息，和错误码相关联 */
 void onFinish(String errorCode, String errorMsg);
}
```

4.3 第三方进入 SDK 的模式

第三方调用startActivityForOcr方法进入 SDK 时，WBOCRTYPEMODE type 参数表示进入 SDK 的模式类型。总共有三种模式，当
type==WBOCRSDKTypeNormal，进入标准模式(进入扫描身份证界面前有个准备界面);当
type==WBOCRSDKTypeFrontSide 时，直接进入扫描身份证人像面界面，进行人像面识别;当
type==WBOCRSDKTypeBackSide 时，直接进入扫描身份证国徽面界面，进行国徽面识别。

5.接口参数说明

参数	说明	类型	长度	是否必填
orderNo	订单号	String	32	必填，合作方订单的唯一标识
openApiAppId	腾讯服务分配的 app_id	String	腾讯服务分配	必填，腾讯服务分配的 app_id
openApiAppVersion	接口版本号	String	20	必填，默认填 1.0.0
openApiNonce	32 位随机字符串	String	32	必填，每次请求需要的一次性 nonce
openApiUserId	User Id	String	30	必填，每个用户唯一的标识
openApiSign	合作方后台服务器通过 ticket 计算出来的签名信息	String	40	必填

6. 个性化参数设置

WbCloudOcrSdk.init() 里 Bundle data，除了必须要传的 InputData 对象(详情见上节)之外，还可以由合作方方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。

6.1 设置 sdk 的界面标题栏背景色

合作方可以设置进入 sdk 的准备界面的标题栏背景色（仅对标准模式此设置才有效）。 SDK 默认显示准备界面的标题栏背景颜色是白色 (#ffffff)，但第三方可对其个性化设置。设置代码如下：

```
# 在 MainActivity 中点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);
//设置标题栏背景色，如果不设置则默认展示；设置了则以设置为准
//此处设置进入 SDK 的第一个界面的标题栏背景色为蓝色(#409eff)
data.putString(WbCloudOcrSDK.TITLE_BAR_COLOR, "#409eff");
```

6.2 设置 sdk 的界面标题栏内容

合作方可以设置进入 sdk 的准备界面的标题栏文字内容（仅对标准模式此设置才有效）。 SDK 默认显示第一个界面的标题栏文字内容是身份证识别，但第三方可对其个性化设置。设置代码如下：

```
# 在 MainActivity 中点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);
//设置标题栏文字内容，如果不设置则默认展示；设置了则以设置为准
//此处设置进入 SDK 的第一个界面的标题栏文字内容为居民身份证识别
data.putString(WbCloudOcrSDK.TITLE_BAR_CONTENT, "居民身份证识别");
```

6.3 设置 sdk 的水印文字内容

合作方可以设置进入 sdk 的第一个界面上的水印文字内容。 SDK 默认显示第一个界面的水印文字内容是仅供内部业务使用，但第三方可对其个性化设置。设置时需要注意：水印文字长度不超过 8，且只支持汉字，若长度超过 8，SDK 会自动截取前 8 个汉字。设置代码如下：

```
# 在MainActivity 中点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);
//设置水印文字内容，如果不设置则默认展示；设置了则以设置为准
//此处设置进入 SDK 的第一个界面的水印文字内容为仅供本次业务使用
data.putString(WbCloudOcrSDK.WATER_MASK_TEXT, "仅供本次业务使用");
```

6.4 设置 sdk 的扫描识别的时间上限

合作方可以设置 sdk 的扫描识别时间的上限。 SDK 打开照相机进行扫描识别的时间上限默认是 20 秒，20秒内若扫描识别成功则返回到 SDK 的第一个界面，否则直到 20
秒直接退出扫描界面。第三方可对其个性化设置，设置的时间上限不能超过 60
秒，建议第三方采用默认值，不要修改这个参数。设置代码如下：

```
# 在 MainActivity 中点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 SDK 扫描识别身份证的时间上限，如果不设置则默认 20 秒；设置了则以设置为准
  //此处设置 SDK 的扫描识别时间的上限为 20 秒
  data.putLong(WbCloudOcrSDK.SCAN_TIME, 20000);
```

6.5 接入示例

```
# 在 MainActivity 中点击某个按钮的代码逻辑：
  //先将必填的 InputData 放入 Bundle 中
  data.putSerializable(WbCloudOcrSDK.INPUT_DATA, inputData);
  //个性化参数设置,此处均设置为与默认不同
  //设置 sdk 标题栏背景颜色，默认白色，此处设置为蓝色（仅对标准模式有效）
data.putString(WbCloudOcrSDK.TITLE_BAR_COLOR, "#409eff");
  //设置 sdk 标题栏文字内容，默认展示身份证识别,此处设置为居民身份证识别（仅对标准模式有效）
data.putString(WbCloudOcrSDK.TITLE_BAR_CONTENT, "居民身份证识别");
  //设置 sdk 水印文字内容，默认仅供内部业务使用，此处设置为仅供本次业务使用（仅对标准模式有效）
  data.putString(WbCloudOcrSDK.WATER_MASK_TEXT, "仅供本次业务使用");
  //设置扫描识别的时间上限,默认 20 秒，此处设置为 20 秒
  data.putLong(WbCloudOcrSDK.SCAN_TIME, 20000);
```

7. 接入示例

```
# 在 MainActivity 中点击某个按钮的代码逻辑：
//先填好数据
  Bundle data = new Bundle();
```

```
WbCloudOcrSDK.InputData inputData = new WbCloudOcrSDK.InputData(
orderNo,
appId,
openApiAppVersion,
nonce,
userId,
sign);
data.putSerializable(WbCloudOcrSDK.INPUT_DATA, inputData);
//个性化参数设置,可以不设置，不设置则为默认选项。
//此处均设置为和默认设置不同
data.putString(WbCloudOcrSDK.TITLE_BAR_COLOR, "#409eff");
//设置 sdk 标题栏文字内容，默认展示身份证识别,此处设置为居民身份证识别data.putString(WbCloudOcr
SDK.TITLE_BAR_CONTENT, "居民身份证识别");
//设置 sdk 水印文字内容，默认仅供内部业务使用，此处设置为仅供本次业务使用
data.putString(WbCloudOcrSDK.WATER_MASK_TEXT, "仅供本次业务使用");
//设置扫描识别的时间上限,默认 20 秒，建议默认
data.putLong(WbCloudOcrSDK.SCAN_TIME, 20000);
//初始化 sdk，得到是否登录 sdk 成功的结果
WbCloudOcrSDK.getInstance().init(MainActivity.this, data, new WbCloudOcrSDK.OcrLoginListener() {
@Override
public void onLoginSuccess() { //登录成功,拉起 SDL 页面
WbCloudOcrSDK.getInstance().startActivityForOcr(MainActivity.this,
new WbCloudOcrSDK.IDCardScanResultListener() { //返退出 SDK 回调接口
@Override
public void onFinish(String resultCode, String resultMsg) {
// resultCode为0，则识别成功；否则识别失败
if ("0".equals(resultCode)) {
// TODO:2017/10/30
WLogger.d(TAG, "识别成功,识别身份证的结果是:"
+WbCloudOcrSDK.getInstance().getResultReturn().toString());
}else{ // TODO:2017/10/30
WLogger.d(TAG, "识别失败:"+resultCode+" --" +resultMsg);
}
}
```

```
},WbCloudOcrSDK.WBOCRTYPEMODE.WBOCRSDKTypeNormal);
}
@Override
public void onLoginFailed(String errorCode, String errorMsg) {
if(errorCode.equals(ErrorCode.IDOCR_LOGIN_PARAMETER_ERROR)) {
Toast.makeText(MainActivity.this, "传入参数有误！" + errorMsg, Toast.LENGTH_SHORT).show();
} else {
Toast.makeText(MainActivity.this, "登录 OCR sdk 失败！" + "errorCode= " + errorCode + "
;errorMsg=" + errorMsg, Toast.LENGTH_SHORT).show();
}
}
})
```

8. 错误码描述

8.1 终端返回错误码

```
 IDOCR_LOGIN_PARAMETER_ERROR = "-20000"; //传入参数有误
IDOCR_USER_CANCEL="200101"; //用户取消操作
IDOCRERROR_USER_NO_NET="100101"; //无网络
IDOCR_USER_2G="100102"; //不支持2G网络
IDOCR_ERROR_PERMISSION_CAMERA="100103"; //无相机权限
IDOCR_ERROR_PERMISSION_READ_PHONE="100103"; //READ PHONE未权限
IDOCR_ERROR_PERMISSION="100103"; //权限异常
  IDOCR_LOGINERROR="-10000"; //登录错误
  SERVER_FAIL="-30000"; //内部服务错误
```

8.2 后台返回错误码

```
INTERNAL_SERVER_ERROR="999999" //网络不给力,请稍后再试
FRONT_INTERNAL_SERVER_ERROR="999998" //网络不给力，请您稍后再试
SERVICE_TIME_OUT="999997" //网络不给力，请您稍后再试
  OAUTH_INVALID_REQUEST="400101" //不合法请求
```

OAUTH_INVALID_LOGIN_STATUS="400102" //不合法请求

OAUTH_ACCESS_DENIED="400103" //服务器拒绝访问此接口

OAUTH_INVALID_PRIVILEGE="400104" //无权限访问此请求

OAUTH_REQUEST_VALIDATE_ERROR="400105" //身份验证不通过

OAUTH_TPS_EXCEED_LIMIT="400501" //请求超过最大限制

OAUTH_INVALID_VERSION="400502" //请求上送版本参数错误

OAUTH_INVALID_FILE_HASH="400503" //文件校验值错误

OAUTH_REQUEST_RATE_LIMIT="400504" //请求访问频率过高

SDK接入（iOS）

说明：接入之前，请详细阅读 SDK 中的 readme 和接入指引！

1. SDK 导入&配置

1. 将 SDK 文件导入到目标 project 中去,并确定“add to target”被勾选.

```
├── WBOCRService.bundle
|
├── WBOCRService.framework
|
├── include
| └── recdetect.h
├── librecdetect.a
└── opencv2.framework
```

2. 在 build phases -> link with libraries 下加入如下依赖.

CoreTelephony.framework
AssetsLibrary.framework
CoreMedia.framework
AVFoundation.framework
libc++.tbd

3. Build Setting --> Enable Bitcode

设置为 NO.

4. Build Setting --> Linking --> other linker flag

设置增加

-ObjC 和 -lz linker flag

5. SDK 中需要使用 camera，需要在

Info.plist

中添加

NSCameraUsageDescription

为 key 的键值对。

2. SDK 调用

调用详情参见 Demo 工程以及头文件 WBOCRService.h

1.在使用 OCR SDK 的类中引入 WBOCRService.h

```
#import <WBOCRService/WBOCRService.h>

@interface ViewController ()

// TODO:

@end
```

2.启动 OCR SDK服务

1) 入口方法

```
/**
* @brief 调起 SDK 入口方法

* @param manageVc SDK 的入口页面对应的 view controller
* @param config 配置参数
* @param version 接口版本号
* @param appId 腾讯服务分配的 app_id
* @param nonce 每次请求需要的一次性 nonce
* @param userId 每个用户唯一的标识
* @param sign 签名信息
* @param orderNo 订单号
* @param startSucceed 启动 SDK 成功回调
* @param recognizeSucceed 识别成功回调
* @param failed SDK 发生异常退出后回调

 */
- (void)startWebOCRServiceWithManageVC:(nonnull UIViewController *)manageVc
  config:(nullable WBWebOCRConfig *)config
  version:(nonnull NSString *)version
  appId:(nonnull NSString *)appId
  nonce:(nonnull NSString *)nonce
  userId:(nonnull NSString *)userId
  sign:(nonnull NSString*)sign
  orderNo:(nonnull NSString *)orderNo
  startSucceed:(nullable WBOCRServiceStartSucceedBlock)startSucceed
  recognizeSucceed:(nullable WBOCRServiceRecognizeSuccessBlock)recognizeSucceed
  failed:(nullable WBOCRServiceFailedBlock)failed;
```

2) WBWebOCRConfig 支持对 SDK 的做个性化配置，字段详情，请阅读 WBWebOCRConfig 头文件部分注释

3 ）WBWebOCRConfig.h

```
/*
```

```
 * @brief WBWebOCRConfig 类定义了 SDK 的配置信息，可以通过 WBWebOCRConfig
单例进行读取、配置，
 * 这个步骤是可选的，如果外部没有配置信息传入，将使用默认的配置参数
 */



/**
 * @brief WBOCRSDKType 定义 SDK 不同的识别模式

 - WBOCRSDKTypeNoraml : 标准模式，SDK 调起成功后，先进入拍摄准备页面，待正反两面识别完成之后
  将识别信息返回到第三方 APP
 - WBOCRSDKTypeFontSide: 人像面识别模式，SDK
调起成功后，直接进入拍摄识别页面，识别身份证人像面
  识别完成之后，将本次识别结果返回第三方 APP
 - WBOCRSDKTypeBackSide: 国徽面识别模式，SDK
调起成功后，直接进入拍摄识别页面，识别身份证国徽面
  识别完成之后，将本次识别结果返回第三方 APP
 */
typedef NS_ENUM(NSInteger, WBOCRSDKType) {
  WBOCRSDKTypeNoraml,
  WBOCRSDKTypeFontSide,
  WBOCRSDKTypeBackSide
};

@interface WBWebOCRConfig : NSObject

+ (instancetype _Nonnull )sharedConfig;

/**
 * @brief 选择 SDK 接入模式，default WBOCRSDKTypeNoraml
 */
@property (nonatomic)WBOCRSDKType SDKType;
/**
 * @brief 设置身份证照片预览页面上的水印信息，default @"仅供本次业务使用"
```

```
 */
@property (nonatomic,copy)NSString * _Nullable waterMarking;

/**
 * @brief 获取 SDK 的登录接口的 baseUrl，readonly
 */
@property (nonatomic,copy,readonly)NSString* _Nonnull baseUrl;

@end
```

3.身份证识别结果,封装在 WBIDCardInfoModel 类中，会返回如下信息

```
/*
 * @brief WBIDCardInfoModel 类封装了身份证的正反面信息
 * SDK会将识别结果包装成一个 WBIDCardInfoModel 实例，通过回调 block 通知第三方

 * @detail 字段含义
 - idcard 公民身份号码
 - name 姓名
 - sex 性别
 - nation 民族
 - address 住址
 - birth 出生
 - authority 签发机关
 - validDate 有效期限
 - frontFullImg 国徽面截图
 - backFullImg 人像面截图
 - orderNo 订单号，和本次业务相关
 - sign 返回的签名信息
- warning 识别结果警告

 */
```

```
@interface WBIDCardInfoModel : NSObject

/// 身份证人像面信息
@property (nonnull,strong,nonatomic) NSString *idcard;
@property (nonnull,strong,nonatomic) NSString *name;
@property (nonnull,strong,nonatomic) NSString *sex;
@property (nonnull,strong,nonatomic) NSString *nation;
@property (nonnull,strong,nonatomic) NSString *address;
@property (nonnull,strong,nonatomic) NSString *birth;

/// 身份证国徽面信息
@property (nonnull,strong,nonatomic) NSString *authority;
@property (nonnull,strong,nonatomic) NSString *validDate;

/// 本次业务相关信息
@property (nonnull,strong,nonatomic) NSString *orderNo;
@property (nonnull,strong,nonatomic) NSString *sign;

/// 正反面识别结果截图信息
@property (nonnull,strong,nonatomic) UIImage* frontFullImg;
@property (nonnull,strong,nonatomic) UIImage* backFullImg;

/// warning，正反面识别结果对应的警告信息
@property (nonnull,strong,nonatomic) NSString *frontWarning;
@property (nonnull,strong,nonatomic) NSString *backWarning;
@end
```

4.具体使用方法参照 demo 工程以及 WBOCRService.h 头文件

5.错误信息通过 fail 回调抛出，错误码列表参见错误码描述章节或头文件。

3.错误码描述

返回码	返回信息	处理措施
100101	无网络，请确认	确认网络正常
100102	不支持 2G 网络	更换网络环境
100103	无相机权限	
200101	用户取消操作	用户主动退出操作
200102	识别超时	用户在身份证正反面识别过程中超过设定的阈值（20S）无法识别，提示超时

生成签名

1. 前置条件：必须按照 Access Token获取 获取 NONCE ticket。

2. 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：

参数	说明
appId	腾讯云分配的 APPID
userId	用户唯一标识
version	1.0.0
ticket	合作伙伴服务端缓存的 tikcet。注意是 NONCE 类型，具体见 Access Token 获取以获取规则。
nonceStr	必须是 32 位随机数

3. 生成一个 32 位的随机字符串（字母和数字）nonceStr，该字符串登录时需要用到。将 appId、userId、version、 ticket、nonceStr 共五个参数的值进行字典序排序。

4. 将排序后的所有参数字符串拼接成一个字符串进行 SHA1 编码。

5. SHA1 编码后的 40 位字符串作为签名（sign）。

6. 签名算法，示例如下（请参考签名算法说明）：
   请求参数：

   appId = TIDA0001
   userId= userID19959248596551
   nonceStr = kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7T (必须为 32 位)
   version = 1.0.0
   ticket=XO99QfxIti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS

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。

身份证识别微信H5接入

合作方生成签名

生成签名

1.前置条件：必须按照说明章节 5.2 获取NONCE ticket.

2.合作方根据本次OCR识别的如下参数生成签名,需要签名的参数信息如下：

参数	说明
appId	腾讯服务分配的app_id
orderNo	订单号，本次人脸验证合作伙伴上送的订单号，唯一标识。
nonce	随机数 32位随机串（字母+数字组成的随机数）
userId	用户ID ，用户的唯一标识（不要带有特殊字符）
version	1.0.0
api ticket	合作伙伴服务端缓存的tikcet,注意是 NONCE 类型，具体见 5.2获取规则.

3.生成一个 32 位的随机字符串(字母和数字) nonce(登录时也要用到)，将webankAppId 、userId、orderNo,version、连同ticket、nonce 共6个参数的值进行字典序排序。
将排序后的所有参数字符串拼接成一个字符串进行SHA1编码
SHA1编码后的40位字符串作为签名(sign)
示例代码及用法：
请求参数：
webankAppId= appId001
userId= userID19959248596551
nonce = kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7T (必须为32位)
version = 1.0.0
orderNo = aabc1457895464
ticket=zxc9Qfxlti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS

字典排序后的参数为：
[1.0.0, aabc1457895464, appId001, kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7T , userID19959248596551,

zxc9QfxIti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS]拼接后的字符串为：

1.0.0aabc1457895464appId001kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7T

userID19959248596551zxc9QfxIti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS

计算 SHA1 得到签名：

5E034EF71E90E5F5FB072CDBB259FFF25A938B03

该字串就是最终生成的签名(40位)，不区分大小写。

公众号启动身份识别H5

启动身份证识别 H5

合作方公众号上送 sign,后台校验 sign 通过之后重定向到身份证识别 H5。

请求 URL: https://ida.webank.com/api/h5/ocrlogin

请求方法:GET

请求参数：

参数	说明	类型	长度	是否必填
webankAppId	WebankAppId，由腾讯指定	String	腾讯服务分配	必填
version	接口版本号	String	20	必填，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	必填
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	必填
url	OCR 识别完成后回调第三方 URL	string		必填，H5 刷脸完成后的回调 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

参数	说明	类型	长度	是否必填
				) Encode 后： (http%3A%2F%2Fidaop.webank.com)
userId	用户 ID ，用户的唯一标识（不要带有特殊字符）	string		必填
sign	签名:使用上面生成的签名。	string	40	必填
ocrFlag	人像面、国徽面识别配置（值为“1”、null时，人像面必须识别，国徽面可选识别；值为“2”时，人像面、国徽面都必须识别）	string		非必填

H5身份证识别结果跳转

H5 身份证 OCR 结果返回跳转第三方 url，带参数：返回码，订单号和签名。合作伙伴接根据返回码判断识别是否成功完成，同时需要根据订单号等信息获取身份证识别结果（识别结果包含姓名、性别、民族、出生日期、身份证号、住址、签发机关、证件有效期）请通过查询接口获取。（见“合作伙伴服务器端查询结果”）

请求URL:

https://xxx.com/xxx?code=xxxx&orderNo=xxxx&signature=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》说明进行签名校验，确保返回结果的安全性。

请求方法:GET

请求参数：

参数	说明	类型	长度
code	身份证ocr识别结果的返回码，0 表示识别成功，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，此信息为本次身份证 ocr 识别上送的信息。	字符串	32
signature	对url 参数 appId、oderNo 和 sign ticket 的签名。具体见的签名生成和校验规则	字符串	40

注：

合作方通过查询身份证识别结果查询接口获取身份证识别信息（姓名、身份证等）及身份证照片，请参

考身份证识别结果查询章节。

身份证识别结果

方式一：前端获取结果验证签名

为了确保前端SDK的结果真实性且未被篡改，合作伙伴服务端可以验证结果，

1.合作伙伴APP端身份证OCR识别SDK返回的带签名结果。

2.合作伙伴APP端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

合作方根据本次人脸验证的如下参数生成签名：

参数	说明
app_id	腾讯服务分配的app_id
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。
api ticket	合作伙伴服务端缓存的tikcet,注意是sign类型，具体见SIGN ticket获取

将app_id、order_no、连同ticket(SIGN)共三个参数的值进行字典序排序

将排序后的所有参数字符串拼接成一个字符串进行SHA1编码

SHA1编码后的40位字符串作为签名(sign)

示例代码及用法

示例：

请求参数：

app_id = appId001

order_no= userID19959248596551

ticket= duSz9ptwyW1Xn7r6gYItxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoe

字典排序后的参数为：

[appId001, duSz9ptwyW1Xn7r6gYItxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoe, test1480921551481]

拼接后的字符串为：

appId001duSz9ptwyW1Xn7r6gYItxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoetest1480921551481

计算 SHA1 得到签名：

B02CEBEB07F792B2F085E8CB1E7BA9EC19284F

54

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的刷脸结果。

注：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)，否则后台无法生成签名值。

方式二：服务端查询结果

合作伙伴服务端生成签名，并调用身份证 OCR 识别服务端查询结果，鉴权完成后返回结果（服务端上送 order_no 和 app_id 查询）。

1. 合作方后台生成签名

合作方根据本次人脸验证的如下参数生成签名：

参数	说明
app_id	腾讯服务分配的 app_id
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。
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，具体见 SIGN ticket获取
nonceStr	32 位随机字符串,字母和数字

生成一个 32 位的随机字符串(字母和数字) nonceStr，将 app_id、order_no、version 连同 ticket、nonceStr 共五个参数的值进行字典序排序再 SHA1 编码生成签名。具体签名算法见签名算法说明。

2. 身份证 OCR 识别结果查询接口

请求URL：https://idasc.webank.com/api/server/getOcrResult
请求方法:GET
请求参数：

参数	说明	类型	长度	是否必填
app_id	腾讯服务分配的 app_id	字符串	腾讯服务分配	必填，腾讯服务分配的 app_id
order_no	订单号	字符串	32	必填，合作方订单的唯一标识
get_file	是否需要获取身份证 OCR 图片文件	字符串	1	非必填，值为1则返回文件；其他则不返回。

参数	说明	类型	长度	是否必填
nonce	随机数	字符串	32	必填
version	版本号	字符串	20	必填，默认值：1.0.0
sign	签名值	字符串	40	必填，使用上面生成的签名。

请求示例：

https://idasc.webank.com/api/server/getOcrResult?app_id=xxx&nonce=xxx&order_no=xxx&version=1.0.0&sign=xxx&get_file=xxxx

返回参数：

参数	类型	说明
frontCode	string	“0”说明人像面识别成功
backCode	string	“0”说明国徽面识别成功
orderNo	string	订单编号
name	string	frontCode为 0 返回:证件姓名
sex	string	frontCode为 0 返回:性别
nation	string	frontCode为 0 返回:民族
birth	string	frontCode为 0 返回:出生日期
address	string	frontCode为 0 返回:地址
idcard	string	frontCode为 0 返回:身份证号
validate	string	backCode为 0 返回:证件的有效期
authority	string	backCode为 0 返回:发证机关
frontPhoto	Base 64 string	人像面照片，转换后为 JPG 格式
backPhoto	Base 64 string	国徽面照片，转换后为 JPG 格式
frontCrop	Base 64 string	人像面切边照片
backCrop	Base 64 string	国徽面切边照片
headPhoto	Base 64 string	身份证头像照片
frontWarnCode	string	人像面警告码，在身份证有遮挡、缺失、信息不全时会返回告警码；当frontCode 为 0 时才会出现告警码，告警码的含义请参考“通用响应码”-“身份证识别响应码”

参数	类型	说明
backWarnCode	string	国徽面警告码，在身份证有遮挡、缺失、信息不全时会返回告警码；当backCode 为 0 时才会出现告警码，告警码的含义请参考“通用响应码”-“身份证识别响应码”
operateTime	string	做 OCR 的操作时间

注意：

1)身份证照片信息作为存证，合作伙伴可以通过此接口拉取识别结果和文件，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，默认不返回照片信息。

2)照片均为 base64 位编码，其中照片解码后格式一般为 jpg。

3)对于身份证识别有部分遮挡、缺失、信息不全的情况，请参考 frontWarnCode 和 backWarnCode 告警码。（见“身份证识别响应码”）

银行卡识别SDK接入

生成签名

1. 前置条件：首先需要按照 Access Token获取 获取 NONCE ticket。
2. 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：

参数	说明
appId	腾讯云分配的app_id
userId	用户唯一标识
version	1.0.0
ticket	合作伙伴服务端缓存的 tikcet。注意是NONCE类型，具体见Access Token 获取以获取规则。
nonceStr	必须是 32 位随机数

3. 生成一个 32
   位的随机字符串（字母和数字）nonceStr（登录时也要用到），将appId、userId、version
   连同ticket、nonceStr 共五个参数的值进行字典序排序。
4. 将排序后的所有参数字符串拼接成一个字符串进行 SHA1 编码
5. SHA1 编码后的 40 位字符串作为签名 （sign）
6. 签名算法，示例如下（请参考签名算法说明）：
   请求参数：

   appId = TIDA0001
   userId= userID19959248596551
   nonceStr = kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7T (必须为 32 位)
   version = 1.0.0
   ticket=XO99QfxIti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS

字典排序后的参数为：

[1.0.0
,TIDA0001,XO99Qfxlti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS，kHoSxvLZGx
SoFsjxlbzEoUzh5PAnTU7T, userID19959248596551]

拼接后的字符串为：

1.0
.0TIDA0001XO99Qfxlti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMSkHoSxvLZGxS
oFsjxlbzEoUzh5PAnTU7TuserID19959248596551

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。

SDK接入（Android）

1.前置条件

1.1 相机/读取手机信息权限检测

- SDK 需要用到相机权限/读取手机信息权限。
- Android6.0 及以上系统
  SDK 运行时提示权限，需要用户授权。
- 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进行。考虑到 SDK 的使用时间很短，快速频繁开关相机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 Android6.0 以下系统上，我们建议合作方在拉起 SDK 前，帮助 SDK 做相机/读取手机信息权限检测，提示用户确认打开了这项权限后再进行银行卡识别，可以使整个银行卡识别体验更快更好。

2.接入配置

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

WbCloudOcr

将以 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-v4
 compile 'com.android.support:appcompat-v4:23.1.1'
  //1. 云Ocr SDK
 compile(name: 'WbCloudOcrSdk-proRelease-v1.2.4-5cf1801
', ext: 'aar')
  //2.云公共组件
compile(name: 'WbCloudNormal-release-v3.0.8-f0cefef', ext: 'aar') }
```

3.混淆配置

云 OCR 产品的混淆规则分为三部分，分别是云 OCR SDK 的混淆规则，云公共组件的混淆规则及依赖的第三方库混淆规则。

3.1 云 OCR SDK 的混淆规则

```
#######################云 ocr 混淆规则 ocr-BEGIN#############################
-keepattributes InnerClasses
-keep public class com.webank.mbank.ocr.WbCloudOcrSDK{
  public <methods>;
  public static final *;
}
-keep public class com.webank.mbank.ocr.WbCloudOcrSDK$*{
  *;
}

-keep public class com.webank.mbank.ocr.tools.ErrorCode{
  *;
```

```
}

-keep public class com.webank.mbank.ocr.net.*$*{
  *;
}
-keep public class com.webank.mbank.ocr.net.*{
  *;
}
########################云 ocr 混淆规则 ocr-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-ocr-proguard-rules.pro

拷贝到主工程根目录下,然后通过

"-include webank-cloud-ocr-rules.pro"

加入到您的混淆文件中。

3.2 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{
  *;
}

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

#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 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。

3.3 云 OCR 依赖的第三方库的混淆规则

```
##########云 OCR 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 OCR 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。

4. 调用 SDK 接口

SDK 代码调用的入口为

com.webank.mbank.ocr.WbCloudOcrSDK

这个类。

```
public class WbCloudOcrSDK{

/**
* 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 synchronized WbCloudOcrSDK getInstance() {
  // ...
  }

/**
* 在使用 SDK 前先初始化，传入需要的数据 data，由 OcrLoginListener 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,OcrLoginListener loginListerner){
// ...
}
  /**
```

```
 * 登录成功后，调用此函数拉起 sdk 页面
 * @param context 拉起 SDK 的上下文
 * @param idCardScanResultListener 返回到第三方的接口
 * @param type 进入 SDK 的模式，参数是枚举类型
 */
 public void startActivityForOcr(Context context,IDCardScanResultListener,WBOCRTYPEMODE type){
 // ...
}
/**
 * 登录回调接口
 */
public interface OcrLoginListener {
 void onLoginSuccess();
 void onLoginFailed(String errorCode, String errorMsg);
 }

/**
 * 退出 SDK,返回第三方的回调,同时返回 ocr 识别结果
 */
public interface IDCardScanResultListener{
 /**
 * @RARAM exidCardResult SDK 返回的识别结果的错误码
* @RARAM exidCardResult SDK 返回的识别结果的错误信息
 */
 void onFinish(String errorCode, String errorMsg);
}
```

WbCloudOcrSdk.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（见 5. 接口参数说明）

```
//这些都是 WbCloudOcrSdk.InputData 对象里的字段，是需要传入的数据信息
String orderNo; //订单号
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
```

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

EXBankCardResult 代表 SDK 返回的识别银行卡的结果，该类属性如下所示：

```
public String ocrId;//识别的唯一标识
public String bankcardNo;//识别的银行卡号
public String bankcardValidDate;//识别的银行卡的有效期
public String orderNo;//订单号
public String warningMsg; //识别的警告信息
public String warningCode; //识别的警告码
public Bitmap bankcardNoPhoto;//识别的银行卡的卡号图片
```

4.1 登录接口

```
/**
 * 登录回调接口
 */
public interface OcrLoginListener {
 void onLoginSuccess();//登录成功
 /**
 * @PARAM errorCode 登录失败错误码
 * @PARAM errorMsg 登录失败错误信息
```

```
 */
 void onLoginFailed(String errorCode, String errorMsg);
 }
```

4.2 返回第三方接口

```
/**
 * 退出 SDK,返回第三方的回调,同时返回ocr识别结果
 */
public interface IDCardScanResultListener{
 /**
 * 退出 SDK,返回第三方的回调,同时返回 ocr 识别结果
 * @param errorCode 返回错误码，识别成功返回 0
 * @param errorMsg 返回错误信息，和错误码相关联 */
 void onFinish(String errorCode, String errorMsg);
}
```

4.3 第三方进入 SDK 的模式

当 type==WBOCRSDKTypeBankSide 时，直接进入扫描银行卡界面，进行银行卡识别。

5. 接口参数说明

参数	说明	类型	长度	是否必填
orderNo	订单号	String	32	必填，合作方订单的唯一标识
openApiAppId	腾讯服务分配的 app_id	String	腾讯服务分配	必填，腾讯服务分配的 app_id
openApiAppVersion	接口版本号	String	20	必填，默认填 1.0.0
openApiNonce	32 位随机字符串	String	32	必填，每次请求需要的一次性 nonce
openApiUserId	User Id	String	30	必填，每个用户唯一

参数	说明	类型	长度	是否必填
				的标识
openApiSign	合作方后台服务器通过 ticket 计算出来的签名信息	String	40	必填

6. 个性化参数设置

WbCloudOcrSdk.init()

里 Bundle data，除了必须要传的 InputData 对象（详情见 5. 接口参数说明）之外，还可以由合作方方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。

6.1 设置 sdk 的扫描识别的时间上限

合作方可以设置 sdk 的扫描识别时间的上限。 SDK 打开照相机进行扫描识别的时间上限默认是 20 秒，20 秒内若识别成功则退出扫描界面，否则一直识别，直到 20 秒后直接退出扫描界面。第三方可对其个性化设置，设置的时间上限不能超过 60 秒，建议第三方采用默认值，不要修改这个参数。设置代码如下：

```
# 在 MainActivity 中点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 SDK 扫描识别证件（身份证、银行卡）的时间上限，如果不设置则默认 20 秒；设置了则以设置为准
  //此处设置 SDK 的扫描识别时间的上限为 20 秒
  data.putLong(WbCloudOcrSDK.SCAN_TIME, 20000);
```

6.2 接入示例

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

```
  //设置扫描识别的时间上限,默认 20 秒，此处设置为 20 秒
  data.putLong(WbCloudOcrSDK.SCAN_TIME, 20000);
```

7. 接入示例

```
# 在 MainActivity 中点击某个按钮的代码逻辑：
//先填好数据
  Bundle data = new Bundle();
  WbCloudOcrSDK.InputData inputData = new WbCloudOcrSDK.InputData(
  orderNo,
  appId,
  openApiAppVersion,
  nonce,
  userId,
  sign);
  data.putSerializable(WbCloudOcrSDK.INPUT_DATA, inputData);
  //个性化参数设置,可以不设置，不设置则为默认选项。
  //设置扫描识别的时间上限,默认 20 秒，建议默认
  data.putLong(WbCloudOcrSDK.SCAN_TIME, 20000);
//初始化 sdk，得到是否登录 sdk 成功的结果
  WbCloudOcrSDK.getInstance().init(MainActivity.this, data, new WbCloudOcrSDK.OcrLoginListener() {
  @Override
  public void onLoginSuccess() { //登录成功,拉起 SDK 页面
WbCloudOcrSDK.getInstance().startActivityForOcr(MainActivity.this,
  new WbCloudOcrSDK.IDCardScanResultListener() { //返退出 SDK 回调接口
  @Override
  public void onFinish(String resultCode, String resultMsg) {
  // resultCode为0，则识别成功；否则识别失败
  if ("0".equals(resultCode)) {
  WLogger.d(TAG, "识别成功，识别银行卡的结果是:"
+WbCloudOcrSDK.getInstance().getBankCardResult().toString());
  } else {
  WLogger.d(TAG, "识别失败"+resultCode+” --” +resultMsg);
```

```
  }

  }
},WbCloudOcrSDK.WBOCRTYPEMODE.WBOCRSDKTypeBackSide);
}
@Override
public void onLoginFailed(String errorCode, String errorMsg) {
if(errorCode.equals(ErrorCode.IDOCR_LOGIN_PARAMETER_ERROR)) {
Toast.makeText(MainActivity.this, "传入参数有误！" + errorMsg, Toast.LENGTH_SHORT).show();
} else {
Toast.makeText(MainActivity.this, "登录OCR sdk失败！" + "errorCode= " + errorCode + " ;errorMsg="
+ errorMsg, Toast.LENGTH_SHORT).show();
}
}
});
```

8. 错误码描述

8.1 终端返回错误码

IDOCR_LOGIN_PARAMETER_ERROR = "-20000"; //传入参数有误
IDOCR_USER_CANCEL="200101"; //用户取消操作
IDOCR__ERROR_USER_NO_NET="100101"; //无网络
IDOCR_USER_2G="100102"; //不支持2G网络
IDOCR_ERROR_PERMISSION_CAMERA="100103"; //无相机权限
IDOCR_ERROR_PERMISSION_READ_PHONE="100103"; //READ PHONE未权限
IDOCR_ERROR_PERMISSION="100103"; //权限异常 IDOCR_LOGIN__ERROR="-10000"; //登录错误
SERVER_FAIL="-30000"; //内部服务错误

8.2 后台返回错误码

INTERNAL_SERVER_ERROR="999999" //网络不给力,请稍后再试

FRONT_INTERNAL_SERVER_ERROR="999998" //网络不给力，请您稍后再试

SERVICE_TIME_OUT="999997" //网络不给力，请您稍后再试

OAUTH_INVALID_REQUEST="400101" //不合法请求

OAUTH_INVALID_LOGIN_STATUS="400102" //不合法请求

OAUTH_ACCESS_DENIED="400103" //服务器拒绝访问此接口

OAUTH_INVALID_PRIVILEGE="400104" //无权限访问此请求

OAUTH_REQUEST_VALIDATE_ERROR="400105" //身份验证不通过

OAUTH_TPS_EXCEED_LIMIT="400501" //请求超过最大限制

OAUTH_INVALID_VERSION="400502" //请求上送版本参数错误

OAUTH_INVALID_FILE_HASH="400503" //文件校验值错误

OAUTH_REQUEST_RATE_LIMIT="400504" //请求访问频率过高

SDK接入（iOS）

说明：接入之前，请详细阅读，SDK 中的 readme 和接入指引！

1. SDK 导入&配置

1. 将 SDK 文件导入到目标 project 中去，并确定“add to target”被勾选。

```
├── WBOCRService.bundle
|
├── WBOCRService.framework
|
├── include
| └── recdetect.h
├── librecdetect.a
└── opencv2.framework
```

2. 在 build phases -> link with libraries 下加入如下依赖。

CoreTelephony.framework
AssetsLibrary.framework
CoreMedia.framework
AVFoundation.framework
libc++.tbd

3. Build Setting --> Enable Bitcode

设置为 NO。

4. Build Setting --> Linking --> other linker flag

   设置 增加

   -ObjC

   和

   -lz linker flag

   。
5. SDK 中需要使用 camera，需要在

   Info.plist

   中添加 N

   SCameraUsageDescription

   为 key 的键值对。

2. SDK 调用

调用详情参见 Demo 工程以及头文件 WBOCRService.h

2.1 在使用 OCR SDK 的类中引入 WBOCRService.h

```
#import <WBOCRService/WBOCRService.h>

@interface ViewController ()

// TODO:
```

```
@end
```

2.2 启动 OCR SDK 服务

1）入口方法

```
/**
 * @brief 调起 SDK 入口方法

 * @param manageVc SDK 的入口页面对应的 view controller
 * @param config 配置参数
 * @param version 接口版本号
 * @param appId 腾讯服务分配的 app_id
 * @param nonce 每次请求需要的一次性 nonce
 * @param userId 每个用户唯一的标识
 * @param sign 签名信息
 * @param orderNo 订单号
 * @param startSucceed 启动 SDK 成功回调
 * @param recognizeSucceed 识别成功回调
 * @param failed SDK 发生异常退出后回调


 * @detail 回调信息 code 和 msg 对照表
 * ## code 退出 code
 * ## msg 退出信息描述

 * @detail code && msg 错误码以及描述信息参考接口文档中“错误码”章节
 *
 */

- (void)startOCRServiceWithManageVC:(nonnull UIViewController *)manageVc
  config:(nullable WBOCRConfig *)config
```

```
 version:(nonnull NSString *)version
 appId:(nonnull NSString *)appId
 nonce:(nonnull NSString *)nonce
 userId:(nonnull NSString *)userId
 sign:(nonnull NSString*)sign
 orderNo:(nonnull NSString *)orderNo
 startSucceed:(nonnull WBOCRServiceStartSucceedBlock)startSucceed
 recognizeSucceed:(nonnull WBOCRServiceRecognizeSuccessBlock)recognizeSucceed
 failed:(nonnull WBOCRServiceFailedBlock)failed;
```

2）WBOCRConfig 支持对 SDK 的做个性化配置，字段详情，请阅读 WBOCRConfig 头文件部分注释。
3）WBOCRConfig.h

```
/*
 * @brief WBOCRConfig 类定义了 SDK 的配置信息，可以通过WBOCRConfig 单例进行读取、配置，
 * 这个步骤是可选的，如果外部没有配置信息传入，将使用默认的配置参数
 */


/**
 * @brief WBOCRSDKType定义 SDK 不同的识别模式

 - WBOCRSDKTypeNoraml : 标准模式，SDK 调起成功后，先进入拍摄准备页面，待正反两面识别完成之后
 将识别信息返回到第三方 APP
 - WBOCRSDKTypeFontSide: 人像面识别模式，SDK
调起成功后，直接进入拍摄识别页面，识别身份证人像面
 识别完成之后，将本次识别结果返回第三方 APP
 - WBOCRSDKTypeBackSide: 国徽面识别模式，SDK
调起成功后，直接进入拍摄识别页面，识别身份证国徽面
 识别完成之后，将本次识别结果返回第三方 APP
 */
typedef NS_ENUM(NSInteger, WBOCRSDKType) {
  WBOCRSDKTypeIDCardNormal,
```

```
  WBOCRSDKTypeIDCardFrontSide,
  WBOCRSDKTypeIDCardBackSide,
  WBOCRSDKTypeBankCard
};
@interface WBOCRConfig : NSObject
+ (instancetype _Nonnull )sharedConfig;
/**
 * @brief 选择 SDK 接入模式为WBOCRSDKTypeBankCard，default WBOCRSDKTypeNoraml
 */
@property (nonatomic)WBOCRSDKType SDKType;

/**
 * @brief 获取 SDK 的登录接口的baseUrl，readonly
 */
@property (nonatomic,copy,readonly)NSString* _Nonnull baseUrl;

@end
```

2.3 银行卡识别结果返回

银行卡识别结果返回，封装在 WBBankCardInfoModel 类中，会返回如下信息：

```
/**
 * @brief 银行卡信息
 * @detail 字段含义
 - bankcardNo 银行卡号
 - bankcardValidDate 银行卡有效期(年 / 月，没有为空)
 - warningCode 警告码
 - warningMsg 警告码描述
 - bankcardNoPhoto 卡号图片
*/
@interface WBBankCardInfoModel : NSObject
```

```
@property (nonnull,strong,nonatomic) NSString *bankcardNo;
@property (nullable,strong,nonatomic) NSString *bankcardValidDate;
@property (nullable,strong,nonatomic) NSString *warningCode;
@property (nullable,strong,nonatomic) NSString *warningMsg;
@property (nonnull,strong,nonatomic) UIImage *bankcardNoPhoto;
@property (nonnull,strong,nonatomic) UIImage *bankcardCropPhoto;

@end
```

2.4 具体使用方法

具体使用方法参照 demo 工程以及 WBOCRService.h 头文件。

2.5 错误信息

错误信息通过 fail 回调抛出，错误码列表参见错误码描述章节或头文件。

3. 错误码描述

返回码	返回信息	处理措施
100101	无网络，请确认	确认网络正常
100102	不支持 2G 网络	更换网络环境
100103	无相机权限	
200101	用户取消操作	用户主动退出操作
200102	识别超时	用户在银行卡识别过程中超过设定的阈值（20S）无法识别，提示超时

银行卡识别结果

方式一：前端获取结果验证签名

为了确保前端 SDK 的结果真实性且未被篡改，合作伙伴服务端可以验证结果。

1. 合作伙伴 APP 端银行卡 OCR 识别 SDK 返回的带签名结果。
2. 合作伙伴 APP 端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

合作方根据本次人脸验证的如下参数生成签名：

参数	说明
app_id	腾讯服务分配的 app_id
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。
api ticket	合作伙伴服务端缓存的 tikcet,注意是 sign 类型，具体见 Access Token获取 获取规则

生成签名步骤：

1. 将 app_id、order_no、连同 ticket（SIGN）共三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进行 SHA1 编码。
3. SHA1 编码后的 40 位字符串作为签名（sign）。

示例代码及用法

示例：
请求参数：
app_id = appId001
order_no= userID19959248596551
ticket= duSz9ptwyW1Xn7r6gYItxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoe

字典排序后的参数为：

[appId001, duSz9ptwyW1Xn7r6gYItxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoe, test1480921551481]

拼接后的字符串为：

appId001duSz9ptwyW1Xn7r6gYItxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoetest1480921551481

计算 SHA1 得到签名：

B02CEBEB07F792B2F085E8CB1E7BA9EC19284F54 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 的刷脸结果。

** 注：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)，否则后台无法生成签名值。

方式二：合作伙伴服务端查询结果

合作伙伴服务端生成签名，并调用银行卡 OCR 识别服务端查询结果，鉴权完成后返回结果（服务端上送 order_no 和 app_id 查询）。

1. 合作方后台生成签名

合作方根据本次人脸验证的如下参数生成签名：

参数	说明
app_id	腾讯服务分配的 app_id
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。
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，具体见Access Token获取获取规则
nonceStr	32位随机字符串，字母和数字

- 生成一个 32 位的随机字符串（字母和数字）nonceStr，将 app_id、order_no、version 连同 ticket、nonceStr 共五个参数的值进行字典序排序再 SHA1 编码生成签名。具体签名算法见签名算法说明。

2. 银行卡 OCR 识别结果查询接口

请求URL：

https://idasc.webank.com/api/server/getBankCardOcrResult

请求方法：GET
请求参数：

参数	说明	类型	长度	是否必填
app_id	腾讯服务分配的 app_id	字符串	腾讯服务分配	必填，腾讯服务分配的 app_id
order_no	订单号	字符串	32	必填，合作方订单的

参数	说明	类型	长度	是否必填
				唯一标识
get_file	是否需要获取银行卡OCR图片文件	字符串	1	非必填，值为1则返回文件；其他则不返回。
nonce	随机数	字符串	32	必填
version	版本号	字符串	20	必填，默认值：1.0.0
sign	签名值	字符串	40	必填，使用上面生成的签名。

请求示例：

https://idasc.webank.com/api/server/getOcrResult?app_id=xxx&nonce=xxx&order_no=xxx&version=1.0.0&sign=xxx&get_file=xxxx

返回参数：

参数	类型	说明
code	string	“0”说明银行卡识别成功
msg	string	返回结果描述
orderNo	string	订单编号
bankCardNo	string	resultCode为 0 返回：银行卡号
bankCardValidDate	string	resultCode 为 0 返回：银行卡有效期
bankcardCropPhoto	Base 64 string	银行卡切边图片
bankcardNoPhoto	Base 64 string	银行卡卡号切边图片
orginBankcardPhoto	Base 64 string	识别原始图片
warnCode	string	银行卡警告码，在银行卡日期失效或者过期会提示；当 frontCode 为 0 时才会出现告警码，告警码的含义请参考“通用响应码”-“银行卡识别响应码”
operateTime	string	做 OCR 的操作时间

** 注意：

- 银行卡照片信息作为存证，合作伙伴可以通过此接口拉取识别结果和文件，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，默认不返回照片信息。
- 照片均为 base64 位编码，其中照片解码后格式一般为 jpg。
- 对于银行卡识别有日期失效或者过期的情况，请参考 frontWarnCode和backWarnCode 告警码。（见银行卡识别响应码码）

API服务接入

通用响应码列表

1. 接入前置服务响应码

接入前置服务过程中的常用错误码返回：

返回码	返回信息	处理措施
400101	不合法的请求	签名校验不通过，请按照章节7确认签名过程。
400102	不合法的请求	登录态失效
400103/400104	服务器拒绝访问此接口	需要仔细核对请求获取token或ticket参数
400502	请求上送版本参数错误	需要仔细核对请求获取 token 或 ticket 参数
400504	请求访问频率过高	同一个 APPID，10 分钟只能成功请求一次 token。
400601	请求参数提过大	检查上送的报文大小，尤其是包含图片和文件的报文，不能超过接口要求大小

2.人脸验证响应码

返回码	返回信息	处理措施
0	人脸验证成功	
66660001	不合法请求	APPID 不存在
66660002	服务已过有效期	服务已经过了有效期，请联系相关人员
66660004	1.请确保光线充足; 2.请确保人脸正对框内; 3.请确保脸部无遮挡	人脸比对相似度得分不够
66660005	无照片信息，无法比对	公安无照片
66660006	1.请确保光线充足; 2.请确保人脸正对框内;	按照提示重新操作

返回码	返回信息	处理措施
	3.请确保脸部无遮挡	
66660008/66660009/66660012	服务处理失败,请稍后再试	联系腾讯处理
66660010	姓名和身份证不一致，请确认	姓名和身份证不匹配
66660011	无人脸验证结果	当前订单或用户无人脸验证结果
66660014	活体检测未通过	活体检测未通过
66660015	姓名或身份证不合法	身份证或姓名非法，后台校验身份证有效性，同时也校验姓名
66660016	不合法请求	文件或视频不合法
66660017	验证次数超限	当日用户验证次数超限
66660018	无此 ID 的用户身份信息	微信 H5 刷脸上送的 FACEID 信息有误
66660019	比对源照片问题,无法解码	比对的照片格式有问题，无法解析
66660022	唇语失效,请重新拉取	获取不到数字唇语
66660023	请确保正脸对框且光线充足	防攻击活体检测不通过(包含翻拍、模型等)
-4006/-4007/-4010/-4015/-5001/-5002/-5005/-5009	视频中自拍照特征提取或检测失败	按照如下提示再次尝试： 1.请确保光线充足; 2.请确保人脸正对框内; 3.请确保脸部无遮挡
-4009/-4017/-4018	比对源照片问题,无法解码	提供的比对源照片编码问题，无法解析，可以尝试如下： 1、如果是合作伙伴自带数据源，请确保数据源照片
-5007/-5008/-5013	视频没有声音或声音太小	请大声跟读屏幕数字
-5010	唇动失败	请大声跟读屏幕数字
-5012	视频中噪声太大	请确保刷脸周围环境安静
-5016	请确保正脸对框且光线充足	人脸或者活体检测失败
-5011/-5014	活体检测失败	唇动、语音等检测失败导致的活体检测失败，需要用户匀速大声跟读数字
-5015	视频像素太低,最小 270*480	视频像素太低不支持，建议换手机再试

3. 身份证识别响应码

返回码	返回信息	处理措施
0	身份证识别成功	
66660020	OCR 结果查询不到	身份证识别无结果，确认订单号是否正确
66661001	识别错误,非身份证件或图像质量问题	未能识别，需重新识别
66661002	识别错误,临时身份证	未能识别，不支持临时身份证
66661003	身份证已失效	未能识别，不支持实效身份证
66661004	非法的日期	未能识别
66661005	图像解码异常	未能识别，需重新识别
66661006	非身份证人像面	未能识别，需重新更换到人像面
66661007	非身份证国徽面	未能识别，需重新更换到国徽面
66661008	姓名或身份证号未能识别	姓名或身份证号识别结果为空，或有效性校验不通过：如身份证号码不满足 18 位、身份证号码奇偶校验不通过、姓名只有一个汉字等情况
警告码	返回信息	处理措施
10000000	身份证图像不全（遮挡或缺角）	
01000000	检测为复印件	

注：

（1）字符串每一位对应一个独立告警。可标识多重告警：比如复印件且信息不符：01001000

（2）对于告警码，默认处理为通过，客户可根据情况自行对此告警码做拒绝识别处理

4. 银行卡识别响应码

返回码	返回信息	处理措施
0	银行卡识别成功	
1304	参数过长	订单号长度为不超过 32 位，确认订单号是否正确
9011	银行卡 OCR 识别失败	未能识别，需重新识别
9012	图片模糊	未能识别，需确保图片对焦成功
9013	未找到银行卡	未能识别，不支持实效身份证
9014	卡号不合法	未能识别，需避免图片反光或遮挡

警告码	返回信息	处理措施
9110	无效的日期信息	
66661009	银行卡已过期	

注：

（1）字符串每一位对应一个独立告警。可标识多重告警：比如复印件且信息不符：01001000

（2）对于告警码，默认处理为通过，客户可根据情况自行对此告警码做拒绝识别处理

5. 二要素验证响应码

返回码	返回信息	处理措施
0	请求成功	
66660001	不合法请求	APPID 不存在
66660002	服务已过有效期	服务已经过了有效期，请联系相关人员
66660008/66660009/66660012	服务处理失败,请稍后再试	联系腾讯处理
66660010	姓名和身份证不一致，请确认	姓名和身份证不匹配
66660015	姓名或身份证不合法	身份证或姓名非法，后台校验身份证有效性，同时也校验姓名

常用错误处理

1. 计算 ticket 阶段

易出现 400101 不合法的请求,ticket 的类型是 NONCE 类型 ticket，ticket 有效期 2 分钟，且一次性有效，即每次启动 SDK 刷脸都要重新请求 NONCE ticket。
腾讯两种 ticket 的区别
NONCE 类型 ticket：用于合作方跳转到腾讯前端时的签名计算，NONCE ticket 有效期为 2 分钟，且一次性有效，合作方使用不需要缓存 NONCE ticket，每次跳转时重新获取。
SIGN 类型 ticket：用于合作方与腾讯后台交互时的签名计算，SIGN ticket 有效期为 1 小时，合作方使用时需要缓存 SIGN ticket，全局唯一。

2. 计算签名 sign 阶段

2.1 参数类型

排序前全部的参数都必须是 string 类型，不能有 int 类型,如果有 timestamp 参数， timestamp 必须为 stirng 类型。

2.2 参数个数

算签名的参数是否按照接口文档的描述计算，启动 SDK 刷脸时合作方后台计算签名的参数值为 5 个：app_id、nonce、version、user_id、ticket(NONCE 类型)。(注释：NONCE ticke 与用户是绑定的，请求 NONCE ticket 传的 user_id 必须与启动 SDK 刷脸时上送的 userId 一致)。其他接口计算签名的参数为 URL 上的参数值、ticket(SIGN 类型)以及整个 post body (编码为 json String)。
如果报 400101 不合法的请求。:app_id
请求参数格式有误，app_id 、version等几个参数是放在 URL 上，如果没有放会报如下错误："code";"400101","msg";"不合法请求:app_id"
解决办法：请求时将 app_id 、version 等几个参数是放在 URL 上，例：

https://xxx.webank.com/api/xxx?app_id=xxx&version=1.0.0

2.3 参数排序

字典序排序，以 ascii 码表的顺序进行排序，各种语言的 sort 方法就可实现。

2.4 参数一致性

合作方后台计算签名的数据与发给腾讯 SDK 的数据是否一致，例如合作方计算签名的 nonce、userId 等必须与发送给腾讯 SDK 的数据一致。提示：计算 sign 的目的就是为了防止数据被篡改，所以请务必计算 sign 签名的参数与发送给腾讯 SDK 的数据保持绝对一致。

2.5 ticket 过期

计算 sign 时有用到 ticket 这个参数。ticket 是否在有效期，ticket 的类型是否搞混，前端跳转时使用 NONCE 类型 ticket，ticket 有效期 2 分钟，且一次性有效，即每次做前端都要重新请求 NONCE ticket，重新计算签名；后台交互使用 SIGN 类型 ticket，1 小时有效。

2.6 content type 为 application/json

所有请求的 content type 为 application/json，否则会报错。

2.7 其他问题

把字典排序后的参数、拼接后的字符串、sha1 生成的签名值发到联调群里找开发人员协助解决。