

目录

目录	1
请求结构	3
公共参数	3
传入方式	3
参数说明	3
返回结果	3
调用成功	4
调用失败	4
签名算法	4
请求	4
计算签名	4
添加	4
签名计算完整样例	4
各语言版本的签名	5
PHP（版本5.5+）	5
Python（版本2.7）	5
活体检测	5
描述	5
HTTP METHOD	5
HTTP BODY编码格式	6
请求参数	6
返回结果	6
示例	6
请求示例	6
人脸核身	6
描述	6
HTTP METHOD	6
HTTP BODY编码格式	7
请求参数	7
返回结果	7
示例	7
请求示例	7
二要素	7
描述	7
HTTP METHOD	8
HTTP BODY编码格式	8
请求参数	8
返回结果	8
示例	8
请求示例	8
OCR	9
描述	9
HTTP METHOD	9
HTTP BODY编码格式	9
请求参数	9
返回结果	9
示例	9
请求示例	9
iOS版活体检测SDK文档	10
下载SDK及相关文档	10

合规性说明	10
SDK集成	10
Android版活体检测SDK开发文档	12
SDK集成	12
FAQ（常见问题解答）	16
历史版本	16

请求结构

客户调用金山云身份认证服务的openAPI接口是通过向指定服务地址发送请求，并按照openAPI文档说明在请求中添加相应的公共参数和接口参数来完成的。

身份认证openAPI的请求结构组成如下：

服务地址

身份认证的服务接入地址为：`identity.api.ksyun.com`

通信协议

支持通过 HTTP 或 HTTPS 两种方式进行请求通信，推荐使用安全性更高的 HTTPS方式发送请求。

请求方法

身份认证的openAPI仅支持POST请求方式。

注意：使用 POST 方式，参数均从 请求Body中取得，需要使用x-www-form-urlencoded方式进行编码。

请求参数

金山云openAPI请求包含两类参数：**公共请求参数**和**接口请求参数**。其中，公共请求参数是每个接口都要用到的请求参数，具体可查看[公共参数](#)；接口请求参数是各个接口所特有的，具体见各个接口的“请求参数”描述。

字符编码

请求及返回结果都使用UTF-8字符集进行编码。

公共参数

传入方式

- GET请求，放在url的query里面。 e.g: `xx.api.ksyun.com?{业务参数}&{公共参数}`。
- POST请求，放在http body里面。 e.g: `{业务参数}&{公共参数}`。

参数说明

公共请求参数(必须)是每个OpenAPI接口都需要使用的请求参数。

名称	类型	是否必须参数	描述
Accesskey	String	是	ak 用户在控制台创建的accesskey
Service	String	是	服务名称，固定值： <code>identity</code>
Action	String	是	操作接口名，与调用的具体openAPI相关
Version	String	是	接口版本号，固定值： <code>2021-10-15</code>
Timestamp	String	是	时间，UTC格式，例如： <code>2021-11-13T17:18:36Z</code>
SignatureVersion	String	是	签名版本号，固定值： <code>1.0</code>
SignatureMethod	String	是	签名算法，固定值： <code>HMAC-SHA256</code>
Signature	String	是	签名 点击查看签名算法
Region	String	否	区域，例如： <code>cn-beijing-6</code>
SecurityToken	String	否	安全令牌，在使用临时ak(STS 角色扮演获取的ak/sk)，需要传该字段，如果使用GET方法，需要对该字段进行urlencode
DryRun	Boolean	否	检查当前调用者是否有权限执行相关操作，而不是真的调用执行相关操作

返回结果

调用金山云的openAPI服务，调用成功，返回的HTTP状态码（Status）为200；调用失败，返回4xx 或5xx的HTTP状态码（Status）。

金山云的身份认证服务的调用返回的数据格式仅支持json。

调用成功

- json格式示例

```
{
  "ErrMsg": "请求成功",
  "Code": "200",
  "Data": {
    "ActionVerify": 1,
    "Score": 0.8
  },
  "RequestId": "0b836e84-f59a-449c-b30f-617e03b56e6a"
}
```

调用失败

调用接口失败，不会返回结果数据；HTTP请求返回一个4xx或5xx的HTTP状态码，返回的HTTP消息体中包含具体的错误代码(code)及错误信息(message)；与调用成功一样还包含请求ID(RequestId)，在调用方找不到错误原因时，可以联系金山云客服，并提供RequestId，以便我们尽快帮您解决问题。

- json格式示例

```
{
  "ErrMsg": "xxx",
  "Code": "10002",
  "Data": {
  },
  "RequestId": "0b836e84-f59a-449c-b30f-617e03b56e6a"
}
```

签名算法

请求

根据请求参数(公共参数和业务参数，不包含公共参数Signature)构造规范化请求字符串：CanonicalizedQueryString。

1. 请求参数排序。排序规则以参数名按照字典排序。
2. 请求参数编码。使用UTF-8字符集对每个请求参数的名称和参数取值进行URLEncode，一般在URLEncode后需对三种字符替换：加号(+)替换成 %20、星号(*)替换成 %2A、 %7E 替换成波浪号(~)。
3. 请求字符串CanonicalizedQueryString。

计算签名

`sign = hash_hmac('sha256', CanonicalizedQueryString, sk)`

sign值为签名算法返回的16进制格式小写字符串。

签名样例：

b28616f50f00380341a647c73101a459d8119c9d4a98fcff5fa4a023f82ef229

计算签名时使用的sk为Accesskey对应的密钥，使用的哈希算法是：HMAC-SHA256。

添加

将签名值作为Signature参数值添加到请求参数中。

签名计算完整样例

1> 构造规范化请求字符串样例

请求参数数组：

```
{
  "AppId": "ftYXXoM1oNmhUKE0gA3xkUQcvCBVL30NV2bcV1qcnIb0EszG3cxK1orXnwAbGMnDHwxJOM8MXkIaWZ9B24LCVOrNXMPGMgGhaYFovNmBUOG4zVQ==",
  "Token": "2fb2b664ea555fb06b312c92b4a9ae11 CM__1__68d04de46704184607095c0ed13c525c__2.1.3.1__1__STsid00000015881406484578yDK1EVivAwBf0wwxHTxZoNUS6WEXHZ0",
  "AuthCode": "123456",
  "Action": "MobileQuery",
  "Version": "2019-05-01",
}
```

```
    "SignatureVersion": "1.0",
    "SignatureMethod": "HMAC-SHA256",
    "Timestamp": "2020-04-15T14:58:22Z",
    "Service": "onepass",
    "Accesskey": "AKxxx"
}
```

1. 请求参数排序：排序结果如下。

```
{
    "Accesskey": "AKxxx",
    "Action": "MobileQuery",
    "AppId": "ftYXXoM1oNmhUKE0gA3xkUQcvCBVL30NV2bcV1qcnIb0EszG3cxK1orXnwAbGMnDHwxJOM8MXkIaWZ9B24LCVorNXMPGMgGhaYFovNmBUOG4zVQ==",
    "AuthCode": "123456",
    "Service": "onepass",
    "SignatureMethod": "HMAC-SHA256",
    "SignatureVersion": "1.0",
    "Timestamp": "2020-04-15T14:58:22Z",
    "Token": "2fb2b664ea555fb06b312c92b4a9ae11CM__1__68d04de46704184607095c0ed13c525c__2.1.3.1__1__STsid00000015881406484578yDK1EVivAwBf0wwxHTxZoNUS6WEXHZ0",
    "Version": "2019-05-01"
}
```

2. 请求参数和参数值URLEncode

3. 字符串拼接如下：

```
Accesskey=AKxxx&Action=MobileQuery&AppId=ftYXXoM1oNmhUKE0gA3xkUQcvCBVL30NV2bcV1qcnIb0EszG3cxK1orXnwAbGMnDHwxJOM8MXkIaWZ9B24LCV
orNXMPGMgGhaYFovNmBUOG4zVQ%3D%3D&AuthCode=123456&Service=onepass&SignatureMethod=HMAC-SHA256&SignatureVersion=1.0&Timestamp=202
0-04-15T14%3A58%3A22Z&Token=2fb2b664ea555fb06b312c92b4a9ae11%20CM__1__68d04de46704184607095c0ed13c525c__2.1.3.1__1__STsid00000
015881406484578yDK1EVivAwBf0wwxHTxZoNUS6WEXHZ0&Version=2019-05-01
```

2> 根据待签串计算签名

```
sk = 'SKxxx'
CanonicalizedQueryString = "Accesskey=AKxxx&Action=MobileQuery&AppId=ftYXXoM1oNmhUKE0gA3xkUQcvCBVL30NV2bcV1qcnIb0EszG3cxK1orXnw
AbGMnDHwxJOM8MXkIaWZ9B24LCVorNXMPGMgGhaYFovNmBUOG4zVQ%3D%3D&AuthCode=123456&Service=onepass&SignatureMethod=HMAC-SHA256&Signatu
reVersion=1.0&Timestamp=2020-04-15T14%3A58%3A22Z&Token=2fb2b664ea555fb06b312c92b4a9ae11%20CM__1__68d04de46704184607095c0ed13c5
25c__2.1.3.1__1__STsid00000015881406484578yDK1EVivAwBf0wwxHTxZoNUS6WEXHZ0&Version=2019-05-01"
//sign = hash_hmac('sha256', CanonicalizedQueryString, sk)
sign: 48d92e6f31d2d7c6b382ae3103105f64cbab3e049bf468a453c49f3b6953f86b
```

各语言版本的签名

PHP（版本5.5+）

```
function sign($params, $secret_key)
{
    ksort($params, SORT_STRING);
    $str_encode = '';
    foreach($params as $k=>$v) {
        $str_encode .= rawurlencode($k).'='.rawurlencode($v).'&';
    }
    $str_encode = substr($str_encode, 0, -1);
    return hash_hmac("sha256", $str_encode, $secret_key);
}
//rawurlencode 实现的urlencode，符合上述签名URLEncode的要求，所以无需做替换
```

Python（版本2.7）

```
#-*- coding:utf-8 -*-
import sys, os, base64, datetime, hashlib, hmac
import urllib
def sign(params, secret_key):
    str_encode = ''
    param_keys = sorted(params.keys())
    for key in param_keys:
        str_encode += urllib.quote(key, '~') + '=' + urllib.quote(str(params[key]), '~') + '&'
    print(str_encode[:-1])
    return hmac.new(secret_key, str_encode[:-1], hashlib.sha256).hexdigest()
```

活体检测

描述

通过动作进行活体检测。

HTTP METHOD

POST|GET

HTTP BODY编码格式

application/x-www-form-urlencoded

请求参数

点击查看 [公共参数列表详情](#)。

名称	类型	必须	描述
Action	String	是	FaceVideo
Version	String	是	2021-10-15
AppId	String	是	金山云 appid
PlatformType	int	是	1-ios 2-android
Video	String	是	1. sdk 中返回 base64 编码的视频数据 2. URL:视频 件URL地址，检测过程中可能下载超时，不推荐
Token	String	是	token sdk 中返回
ExtId	String	否	第三方流水号

返回结果

通过指定 Accept:application/json，返回的数据格式为json。

名称	类型	描述
ActionVerify	Int	1通过 0不通过
Score	float	活体检测的总体打分 范围[0, 1]，分数越 ，则活体的概率越

示例

请求示例

```
curl -X POST \
'http://identity.api.ksyun.com/' \
-H 'Authorization: AWS4-HMAC-SHA256 xxx' \
-H 'accept: application/json' \
-H 'content-type: application/x-www-form-urlencoded' \
-H 'host: identity.api.ksyun.com' \
-d 'Action=FaceVideo&Version=2021-10-15&AppId=J6akuU4YS0icQ_xJ3AVzKA&Token=eyJ0b2t1biI6Ildsfs&PlatformType=1&Video=1af1jasfljadfl'
```

调用成功返回示例

```
{
  "ErrMsg": "请求成功",
  "Code": "200",
  "Data": {
    "ActionVerify": 1,
    "Score": 0.8
  },
  "RequestId": "0b836e84-f59a-449c-b30f-617e03b56e6a"
}
```

调用失败返回示例

```
{
  "ErrMsg": "xxx",
  "Code": "10002",
  "Data": {
  },
  "RequestId": "0b836e84-f59a-449c-b30f-617e03b56e6a"
}
```

人脸核身

描述

对比人脸图片和身份证照片是否一致。

HTTP METHOD

POST|GET

HTTP BODY编码格式

application/x-www-form-urlencoded

请求参数

点击查看 [公共参数列表详情](#)。

名称	类型	必须	描述
Action	String	是	FaceMatch
Version	String	是	2021-10-15
AppId	String	是	金山云 appid
IdCardNo	string	否	身份证号
IdCardName	String	否	身份证姓名
Token	String	是	token sdk 中返回
ExtId	String	否	第三方流水号

返回结果

通过指定 Accept:application/json，返回的数据格式为json。

名称	类型	描述
Match	Int	1通过 0不通过
Desc	string	说明

示例

请求示例

```
curl -X POST \
  'http://identity.api.ksyun.com/' \
  -H 'Authorization: AWS4-HMAC-SHA256 xxx' \
  -H 'accept: application/json' \
  -H 'content-type: application/x-www-form-urlencoded' \
  -H 'host: identity.api.ksyun.com' \
  -d 'Action=FaceMatch&Version=2021-10-15&AppId=J6akuU4YS0icQ_xJ3AVzKA&Token=eyJ0b2t1biI6I1Nuc2lkMDAwM&IdCardNo=121231231123&IdCardName=ladfaf'
```

调用成功返回示例

```
{
  "ErrMsg": "请求成功",
  "Code": "200",
  "Data": {
    "Match": 1,
    "Desc": "说明"
  },
  "RequestId": "0b836e84-f59a-449c-b30f-617e03b56e6a"
}
```

调用失败返回示例

```
{
  "ErrMsg": "xxxx",
  "Code": "10021",
  "Data": {
  },
  "RequestId": "0b836e84-f59a-449c-b30f-617e03b56e6a"
}
```

二要素

描述

验证姓名和身份证是否一致。

HTTP METHOD

POST|GET

HTTP BODY编码格式

application/x-www-form-urlencoded

请求参数

点击查看 [公共参数列表详情](#)。

名称	类型	必须	描述
Action	String	是	IdCardMatch
Version	String	是	2021-10-15
AppId	String	是	金山云 appid
IdCardNo	string	否	身份证号
IdCardName	String	否	身份证姓名
PlatformType	String	是	1-ios; 2-android ; 3-h5 ; 5-其他
ExtId	String	否	第三方流水号

返回结果

通过指定 Accept:application/json，返回的数据格式为json。

名称	类型	描述
Country	String	县区
Gender	int	性别 1:男 2:
Province	String	省份
Birthday	String	生日
Age	int	年龄
City	String	市区
Match	int	1:认证 致（收费）2:认证不 致（收费）3:认证不确定（不收费）4:认证失败（不收费）

示例

请求示例

```
curl -X POST \
'http://identity.api.ksyun.com/' \
-H 'Authorization: AWS4-HMAC-SHA256 xxx' \
-H 'accept: application/json' \
-H 'content-type: application/x-www-form-urlencoded' \
-H 'host: identity.api.ksyun.com' \
-d 'Action=IdCardMatch&Version=2021-10-15&AppId=J6akuU4YS0icQ_xJ3AVzKA&IdCardNo=121231231123&IdCardName=ladfaf&PlatformType=1'
```

调用成功返回示例

```
{
  "Code": "200",
  "Data": {
    "Country": "安县",
    "Gender": "1",
    "Province": "河北省",
    "Birthday": "19970501",
    "Match": 2,
    "Age": "20",
    "City": "xx市"
  },
  "ErrMsg": "请求成功",
  "RequestId": "c267d253-56f1-41d4-9665-accce78c1179"
}
```

调用失败返回示例

```
{
  "Code": "1001",
  "Data": {},
}
```



```
    "ErrMsg": "参数 错误",
    "RequestId": "45cc9f31-3c00-4fb1-94aa-c6d6a112e448"
}
```

OCR

描述

通过图片识别技术提取身份证的姓名和身份证号码信息。

HTTP METHOD

POST|GET

HTTP BODY编码格式

application/x-www-form-urlencoded

请求参数

点击查看 [公共参数列表详情](#)。

名称	类型	必须	描述
Action	String	是	0crSingle
Version	String	是	2021-10-15
AppId	String	是	金山云 appid
Image	string	是	a. sdk 中返回BASE64:图 的base64值 b. URL:图 的 URL地址
Side	String	是	正面: front 背面: back
PlatformType	int	是	1-ios 2-android
ExtId	String	否	第三方流水号

返回结果

通过指定 Accept:application/json，返回的数据格式为json。

名称	类型	描述
Nation	String	族
IssuingDate	String	签发 期
Name	String	姓名
Birthday	String	出 年
IdCardNo	String	证件号
Msg	String	识别信息
Sex	String	性别
Address	String	身份证地址
ExpireDate	String	有效 期
IssuingAuthority	String	签发机构
ImageStatus	int	获取结果状态 1-正常 2-不完整 3-模糊 4-过暗 5-曝光 6-正反面颠倒

示例

请求示例

```
curl -X POST \
  'http://identity.api.ksyun.com/' \
  -H 'Authorization: AWS4-HMAC-SHA256 xxx' \
  -H 'accept: application/json' \
  -H 'content-type: application/x-www-form-urlencoded' \
  -H 'host: identity.api.ksyun.com' \
  -d 'Action=0crSingle&Version=2021-10-15&AppId=J6akuU4YS0icQ_xJ3AVzKA&Image=eyJ0b2t1biI6I1NUc2lkMDAwM&PlatformType=1&Side=f
ront'
```

调用成功返回示例

```
{
  "Code": "200",
  "Data": {
    "Nation": "汉",
    "IssuingDate": null,
    "Name": "xxx",
    "BrithDay": "19990610",
    "IdCardNo": "131736199906100452",
    "Msg": null,
    "Sex": "男",
    "Address": "xxxx",
    "ExpireDate": null,
    "IssuingAuthority": null
  },
  "ErrMsg": "请求成功",
  "RequestId": "c267d253-56f1-41d4-9665-accce78c1179"
}
```

调用失败返回示例

```
{
  "ErrMsg": "xxxx",
  "Code": "10021",
  "Data": {
  },
  "RequestId": "0b836e84-f59a-449c-b30f-617e03b56e6a"
}
```

iOS版活体检测SDK文档

活体检测 SDK 集成文档 V2.0.0.0

名称： 活体检测 iOS SDK 集成文档
版本： V2.0.0.0

下载SDK及相关文档

请在官网下载最新的SDK包（包含Demo+SDK+资源包）[点击下载](#) 非开发人员想体验demo，可直接下载ipa包到iPhone手机 [点击下载](#)

合规性说明

SDK名称	场景描述	收集个人信息的类型	数据是否加密传输
活体检测SDK	活体检测+人证核对	活体检测SDK提供活体检测服务过程中收集和使用的信息包括： 用户的面部识别信息、用户的设备摄像头权限；用于活体检测服务中人脸检测、人脸检索、人脸比对；	是

SDK集成

1、获取SDK

从官网下载活体检测 SDK 包。

2、开发环境搭建

2.1 导入framework

将SDK压缩包中framework中的所有资源添加到工程中，并选择 Copy Items if need 选项

□

2.2 配置Xcode

- Xcode->TARGETS->Build Settings->Linking->Other Linker Flags 添加 -ObjC

□

- Xcode->TARGETS->Info添加相机权限访问描述

□

- 如果在活体中需要麦克风播放提示音则需在Xcode->TARGETS->Info添加麦克风权限访问描述：（该权限为demo所用权限非SDK必须权限，仅供参考）。

3、 SDK 初始化

3.1 初始化

建议在 **Application** 的 `didFinishLaunchingWithOptions` 方法中进行初始化：

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
...
[SRCoreManager initWithAppid:@" appid " withAppsecret:@" appSecret "];
...
return YES;
}
```

注意：在 `initWithAppid:` 中设置 **appId**；和在`withAppsecret:` 中设置**appSecret**。

温馨提示：

登录开发者平台查看相关信息。

4、调用活体检测功能

4.1 回调数据说明

① 会在 `completionHandler`中回调 `CLStatus` 信息，具体的信息如下表所示：

CLStatus	错误信息	备注说明
SROperationTimeout	操作超时，用户未在规定时间内完成动作 /	
SRGetConfFaild	获取配置信息失败 /	
SRRecordFaild	视频录制失败 /	
SRRecordSucess	视频录制完成 /	
SROnlineUploadFaild	网络传输失败 /	
SRSDKError	SDK内部发生错误 /	
SRNetError	无网络连接 /	

温馨提示：

① 调用活体检测方法之前，请确保已经获取到相机！！！！

② 我们提供了默认的声音文件和 GIF图片文件，如不满意可以替换，具体逻辑在demo中实现，可任意更改；

4.2调用流程

如下从4.2.1到4.2.4是一个完整的检测流程

4.2.1 先初始化活体检测对象

```
/// 初始化活体检测对象
/// @param recordView 传入放置检测活体的recordView对象
- (instancetype)initWithRecordView:(UIView *)recordView;
```

4.2.2 在开始活体检测前进行相关参数设置

```
/// 设置活体检测的超时时间（在开始活体检测之前设置）
/// @param timeout 请传入10-120范围内的时间值，单位s
- (void)setTimeoutInterval:(NSTimeInterval)timeout;
/// 设置活体检测每个动作的时长（在开始活体检测之前设置）
/// @param duration 每个动作的时长（1-5s）
```

```

- (void) setActionDuration: (CGFloat) duration;

/// 设置活体检测动作的个数（在开始活体检测之前设置，默认为1-3的随机数）
/// @param number 动作的个数（1-3）

- (void) setActionNumber: (NSInteger) number;

4.2.3 开始活体检测

/// 开始活体检测

/// @param actionsHandler 活体检测动作序列号的回调（非主线程）
/// @param completionHandler 活体检测结果的回调（非主线程），结果状态见CLStatus枚举类型

- (void) startLiveDetectWithActionsHandler: (CLAcitionsHandler) actionsHandler
completionHandler: (CLCompletionHandler) completionHandler;

4.2.4 停止活体检测

/// 停止活体检测

/// 调用时机：

/// 1、在活体检测结果的回调里调用
/// 2、未完成活体检测，需要中止时调用（超时不要调用）

- (void) stopLiveDetect;

```

注意人证核对必须是在活体检测成功的基础上去调用！

5 活体检测和人证核对 5.1开始活体检测

在活体检测成功的回调中，拿到对应的token以及sessionId和 活体检测录制本地视频的url，参照服务端的开发文档去调用自己服务端 活体检测的接口；

5.2开始人证核对

在上一步活体检测回调里面，如果请求成功，在去调用自己服务端 人证核对的服务端接口，最后将结果展示给用户即完成人证核对。

Android版活体检测SDK开发文档

SDK集成

下载SDK及相关文档 请在官网下载最新的SDK包（包含Demo+SDK+资源包）[点击下载](#) 非开发人员想体验demo，可直接下载apk包到手机 [点击下载](#)

1、导入SDK

将获取的 **aar** 文件复制到工程中的 **libs** 文件夹下，然后在项目根目录的**build.gradle** 文件中增加如下代码：

```

repositories {

    flatDir {

        dirs 'libs'

    }

}

```

在 **dependencies** 闭包中增加对 **aar** 包的及 **gson** 的依赖：

```
implementation(name: 'alive-detect-sdk-v3.0.0.0-20211130-release', ext: 'aar')
```

2、 SDK 初始化

2.1、建议在 Application 的 onCreate() 方法中进行初始化：

```
@Override  
  
public void onCreate() {  
  
    super.onCreate();  
  
    // 设置调试模式，可以输出日志，方便调试，生产环境请关闭此开关  
    AliveDetectorApplication.setDebuggable(true);  
  
    // 调用初始化接口，传入 Application Context  
    AliveDetectorApplication.init(getApplicationContext());  
  
}
```

2.2、在 AndroidManifest.xml 文件中设置 appId：

```
<meta-data android:name="ALIVE_DETECT_SDK_APP_ID" android:value="替换为你们自己的appId"/>
```

3、调用活体检测

如果觉得我们提供的界面文案、标题、文字大小、titleBar的背景颜色，活体检测提示音，活体检测时的提示动画（GIF图片）等等不满足你们的需求，或者与你们的App页面风格样式不搭，想自定义这些配置当然也是可以的，我们提供了一个个性化的参数配置类供您灵活地配置各种参数，以尽可能满足您的需求。

在调用之前需要先把 5 个声音文件复制到 **assets** 目录下，把 5 个提示动作的GIF 图片复制到 **drawable** 目录下。

示例代码如下：

```
Map<String, String> soundsMap = new HashMap<>(5);  
  
soundsMap.put(ActionKeys.BLINK_EYES, "online_blink_eyes.mp3");  
soundsMap.put(ActionKeys.TURN_UP, "online_turn_head_to_up.mp3");  
soundsMap.put(ActionKeys.TURN_DOWN, "online_turn_head_to_down.mp3");  
soundsMap.put(ActionKeys.OPEN_MOUTH, "online_open_mouth.mp3");  
soundsMap.put(ActionKeys.SHAKE_HEAD, "online_shake_head.mp3");  
  
Map<String, String> imagesMap = new HashMap<>(5);  
  
imagesMap.put(ActionKeys.BLINK_EYES, String.valueOf(R.drawable.online_blink_eyes));  
imagesMap.put(ActionKeys.TURN_UP, String.valueOf(R.drawable.online_turn_head_to_up));  
imagesMap.put(ActionKeys.TURN_DOWN, String.valueOf(R.drawable.online_turn_head_to_down));  
imagesMap.put(ActionKeys.OPEN_MOUTH, String.valueOf(R.drawable.online_open_mouth));  
imagesMap.put(ActionKeys.SHAKE_HEAD, String.valueOf(R.drawable.online_shake_head));  
  
LayoutInflater inflater = LayoutInflater.from(this);  
  
RelativeLayout customLoading = (RelativeLayout) inflater.inflate(R.layout.custom_loading_layout, null);  
  
RelativeLayout.LayoutParams layoutParams = new  
RelativeLayout.LayoutParams(RelativeLayout.LayoutParams.WRAP_CONTENT,  
RelativeLayout.LayoutParams.WRAP_CONTENT);  
  
layoutParams.addRule(RelativeLayout.CENTER_IN_PARENT);  
  
customLoading.setLayoutParams(layoutParams);  
  
SdkConfiguration configuration = new SdkConfiguration.Builder()
```

```
// 设置活体检测时的提示音（无默认值，必须设置，否则会没有提示音）
.setAliveDetectedSounds(soundsMap)

// 设置活体检测时的 gif 图片集合（无默认值，必须设置，否则会不显示提示动画）
.setAliveDetectedImages(imagesMap)

// 建议从服务端调用获取 authToken 接口获取
.setAuthToken(authToken)

// 设置动作数量1-3分别代表1-3个固定动作，4代表1-3 个动作随机，不设置默认是4
.setActionSize(2)

// 每个动作的执行时间，单位为秒，范围 1-5 秒，默认一个动作 3 秒
.setActionSecond(3)

// 设置自定义 loading
.setCustomLoading(customLoading)

.build();

OnlineAliveDetectorApi.getInstance().start(configuration, new IDetectedListener() {

@Override

public void onSuccess(String result) {

Log.d("TAG", "result: " + result);

}

@Override

public void onFailed(int errorCode, String errorMsg) {

Log.e("TAG", "onFailed() -> errorCode: " + errorCode + ",errorMsg: " + errorMsg);

}

});
```

活体检测 SDK 内部默认不实现上传视频到服务端获取活体检测结果的功能，所以在活体检测视频录制完成后需要将视频上传到服务端进行活体检测，服务端会返回活体检测结果。

当活体检视频录制完成时，onSuccess 的回调数据如下：

```
{

"code": 200,

"data": {

"token": "xxx",

"sessionId": "xxx",

"videoPath": "/sdcard/xxx.mp4"

}

}
```

其他 api:

```
// 显示自定义loading（如果不设置自定义loading调用无效）

OnlineAliveDetectorApi.getInstance().showCustomLoading();

// 隐藏自定义loading（如果不设置自定义loading调用无效）
```

```
OnlineAliveDetectorApi.getInstance().dismissCustomLoading();
```

// 显示弹框

```
OnlineAliveDetectorApi.getInstance().showAlertDialog("温馨提示","活体检测失败");
```

具体的代码可以参考 Demo。

回调数据说明：

- ① 当活体检测视频录制完成时，会在 `onSuccess()` 回调方法中收到回调数据；
- ② 当活体检测有错误的时候，会在 `onFailed()` 回调方法中回调 `errorCode`、`errorMsg` 信息，具体的错误信息如下表所示：

错误码	错误信息	备注说明
10001	缺少相机权限	/
10002	缺少读写SD卡权限	/
20001	context为空，SDK没有初始化或者初始化失败	/
20002	appId为空，SDK没有初始化或者初始化失败	/
20003	网络异常，请检查网络连接	/
20004	请求超时，请检查网络连接或重试！	/
30001	活体检测不通过	/
30002	活体检测超时，请在规定时间内完成提示动作	/
30003	当前网络不稳定，请切换网络后再试，谢谢！	/

温馨提示：

- ① 调用活体检测方法之前，请确保已经获取到相机、读写 SD 卡的权限，SDK内部不做权限申请的处理！！
- ② 我们提供了默认的声音文件和 GIF 图片文件，如果觉得我们的声音不好听或者图片不好看，也可以使用你们自己的，声音文件支持mp3、wav 等常见格式，GIF 图片设计规范：123px * 111px；
- ③ 声音和动作提示的 GIF 图片必须设置，否则会没有提示音和动作动画显示；
- ④ 在调用之前需要获取到authToken，为了安全，建议从服务端请求，具体参考服务端的接口文档，或者参考Demo中的实例代码，Demo 请求是为了方便测试，实际生产环境建议从服务端请求；

4、调用身份证 OCR

传递身份证正、反面照片的 base64 编码

```
IdCardApi.getInstance().doubleSideOcrWithBase64(frontBase64, backBase64, new IDetectedListener() {
    @Override
    public void onSuccess(String result) {
        Log.d("TAG", "onSuccess() -> result:" + result);
    }
    @Override
    public void onFailed(int errorCode, String errorMsg) {
        Log.e("TAG", "onFailed() -> errorCode:" + errorCode + ",errorMsg:" + errorMsg);
    }
});
```

温馨提示：

在转成 base64 编码之前，需要将图片做压缩处理，推荐每张照片压缩小于 1M以内。

这一部分需要看自己公司服务端的接口说明！

1. 调用身份证二要素认证

这一部分需要看自己公司服务端的接口说明！

FAQ（常见问题解答）

1、SDK的混淆规则有哪些？

SDK 内部已经自带了混淆规则，打包的时候会自动将 SDK中的混淆规则和你们自己的混淆规则合并，所以无需手动额外添加混淆规则。

2、添加了网络权限，还是无法进行活体检测

从 **Android P** 开始，默认拦截了非 **HTTPS** 的请求，所以在**AndroidManifest.xml** 文件的\<Application/>标签中添加如下代码即可：

```
android:usesCleartextTraffic="true"
```

在高版本系统中，新建一个文件 **network_security_config.xml** 内容如下：

```
\<?xml version="1.0" encoding="utf-8"?>
\<network-security-config>
\<base-config cleartextTrafficPermitted="true" />
\</network-security-config>
```

然后在 **AndroidManifest.xml** 文件中的 **Application** 标签中引用就可以了。

```
android:networkSecurityConfig="@xml/network_security_config"
```

3、关于 64k 问题

如果在集成的时候遇到 64k 错误，请按如下配置：

1. 在 app 的 build.gradle 中添加 multiDexEnabled true

```
android {
...
defaultConfig {
...
multiDexEnabled true
}
}
```

2. 添加依赖并同步一下工程：

```
implementation 'com.android.support:multidex:1.0.3'
```

如果您使用的是 AndroidX 依赖：

```
implementation "androidx.multidex:multidex:2.0.1"
```

3. 在你们自己的 Application 类中，继承于 MultiDexApplication 类，或者在Application 类中重写如下方法：

```
@Override
protected void attachBaseContext(Context base) {
super.attachBaseContext(base);
MultiDex.install(this);
}
```

历史版本

发布日期	发布版本	更新说明
2021-10-08	V2.0.0	初版发布