# 金融级身份认证

# 银行卡 OCR 识别接入

# 产品文档

腾讯云

# 文档目录

# 银行卡 OCR 识别接入
# SDK 接入
# 生成签名

最近更新时间：2019-01-11 10:14:27

## 准备步骤

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

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

## 基本步骤

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

**注意：**

签名算法可参考 签名算法说明 。

**参考示例**

**请求参数：**

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

**字典排序后的参数为：**

[1.0.0, TIDA0001, XO99Qfxlti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMS ， kHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7T, userID19959248596551]

**拼接后的字符串为：**

1.0.0TIDA0001XO99Qfxlti9iTVgHAjwvJdAZKN3nMuUhrsPdPlPVKlcyS50N6tlLnfuFBPIucaMSkHoSxvLZGxSoFsjxlbzEoUzh5PAnTU7TuserID19959248596551

**计算 SHA1 得到签名：**

4AE72E6FBC2E9E1282922B013D1B4C2CBD38C4BD

**该字串就是最终生成的签名（40 位），不区分大小写。**

下一步：

Android SDK 接入

iOS SDK 接入

# Android SDK 接入
# 开发准备

最近更新时间：2018-05-09 10:42:26

## 权限检测

SDK 需要用到相机 / 读取手机信息权限：

1. Android 6.0 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 Android 6.0 以下系统上，**我们建议合作方在拉起 SDK 前，帮助 SDK 做相机/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银行卡 OCR 识别**，可以使整个银行卡识别体验更快更好。
3. armeabi-v7a 平台
   目前 SDK 只支持 armeabi-v7a 平台，为了防止在其他 CPU 平台上 SDK Crash，我们建议在您的 App 的 build.gradle 里加上 abiFilter，如下图中红框所示：

> **注意：**
> 有且只有 armeabi-v7a 平台。


下一步：配置流程

# 配置流程

最近更新时间：2018-01-24 15:28:59

## 1. 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') }
```

## 2.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。

### 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。

### 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。

### 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。

上一步：开发准备

下一步：接口调用

# 接口调用

最近更新时间：2018-06-29 11:57:50

## SDK 接口调用方法

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：

```
//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
```

### 登录回调接口

```
/**
* 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);
}
```

### 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);
}
```

### 银行卡识别结果类

银行卡识别结果, 封装在 EXBankCardResult 类中，通过 WbCloudOcrSDK.getInstance().getBankCardResult()) 获得，该类属性如下所示：

```
public String ocrId;//ocrId
public String bankcardNo;//卡号
public String bankcardValidDate;//有效期
public String orderNo;//订单号
public String warningMsg;//告警信息
public String warningCode;//告警码
public Bitmap bankcardNoPhoto; //卡号切边图
public String bankcardFullPhoto;//银行卡图片存放路径
```

## 接口参数说明

InputData 是用来给 SDK 传递一些必须参数所需要使用的对象（WbCloudOcrSdk.init() 的第二个参数），合作方需要往里塞入 SDK 需要的一些数据以便启动 OCR SDK。
其中 InputData 对象中的各个参数定义如下表，请合作方按下表标准传入对应的数据。

| 参数 | 说明 | 类型 | 长度 | 是否必填 |
| --- | --- | --- | --- | --- |
| orderNo | 订单号 | String | 32 | 必填，合作方订单的唯一标识 |

| 参数 | 说明 | 类型 | 长度 | 是否必填 |
| --- | --- | --- | --- | --- |
| 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 |
| clientIp | 用户 ip 信息 | String | 30 | 必填,格式为“ip=xxx.xxx.xxx.x xx;”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“ |

# 个性化参数设置

WbCloudOcrSdk.init() 里 Bundle data，除了必须要传的 InputData 对象（详见上节）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。

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

### 个性化设置接入示例

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

上一步：配置流程

# 错误码描述

最近更新时间：2018-01-24 15:30:16

### 终端返回错误码

| 错误码 | 错误描述 |
| --- | --- |
| 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 |

### 后台返回错误码

| 错误码 | 错误描述 |
| --- | --- |
| INTERNAL_SERVER_ERROR="999999" | 网络不给力，请稍后再试 |
| FRONT_INTERNAL_SERVER_ERROR="999998" | 网络不给力，请稍后再试 |
| SERVICE_TIME_OUT="999997" | 网络不给力，请稍后再试 |
| 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 |

| 错误码 | 错误描述 |
| --- | --- |
| 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 |

# 接入示例

最近更新时间：2018-05-31 15:59:39

```
# 在 MainActivity 中单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().startActivityF
orOcr(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();
```

```
}
}
});
```

# iOS SDK 接入
# 错误码描述

最近更新时间：2019-01-08 17:02:44

接入方式请参见 身份证 OCR 识别接入。

| 返回码 | 返回信息 | 处理措施 |
| --- | --- | --- |
| 100100 | 传入SDK参数不合法 | 检查传入参数是否合法 |
| 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 |

# 识别结果

# 方式一：前端获取结果验证签名

最近更新时间：2018-05-10 16:13:11

**此方式用于：**

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

## 1. 合作方后台生成签名

### 准备步骤

- **前置条件：请合作方确保 SIGN ticket 已经正常获取，获取方式见 SIGN ticket 获取 。**
- 合作方为银行卡 OCR 识别服务生成签名，需要具有以下参数：

| 参数 | 说明 | 来源 |
| --- | --- | --- |
| app_id | 腾讯云线下对接分配的 App ID | 腾讯云线下对接分配 |
| order_no | 订单号，本次人脸验证合作伙伴上送的订单号，唯一标识 | 合作方自行分配 |
| api ticket | 合作伙伴服务端缓存的 tikcet，注意是 SIGN 类型 | 获取方式见 SIGN ticket 获取 |

### 基本步骤

1. 将 app_id, order_No, api ticket（SIGN 类型）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。
3. 将排序后的字符串进行 SHA1 编码，编码后的 40 位字符串作为签名（sign）。

> **注意：**
>
> 签名算法可参考 签名算法说明 。

**参考示例**

**请求参数：**

| 参数名 | 参数值 |
| --- | --- |

| 参数名 | 参数值 |
| --- | --- |
| app_id | appId001 |
| order_no | test1480921551481 |
| ticket | duSz9ptwyW1Xn7r6gYltxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoe |

**字典排序后的参数为：**

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

**拼接后的字符串为：**

```
appId001duSz9ptwyW1Xn7r6gYltxz3feMdJ8Na5x7JZuoxurE7RcI5TdwCE4KT2eEeNNDoetest1480921551481
```

**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 的银行卡 OCR 识别结果。

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

# 方式二：合作伙伴服务端查询结果

最近更新时间：2018-10-09 11:18:07

**此方式用于：**

合作伙伴服务端生成签名，并调用银行卡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

### 准备步骤

- **前置条件：请合作方确保 SIGN ticket 已经正常获取，获取方式见 SIGN ticket 获取 。**
- 合作方为银行卡 OCR 识别服务生成签名，需要具有以下参数：

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

### 基本步骤

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

> **注意：**
>
> 签名算法可参考 签名算法说明 。

# 2. 银行卡 OCR 识别结果查询接口

### 请求

**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（字节） | 是否必填 |
| --- | --- | --- | --- | --- |
| app_id | 腾讯服务分配的 App ID | 字符串 | 腾讯服务分配 | 是 |
| order_no | 订单号，合作方订单的唯一标识 | 字符串 | 32 | 是 |
| get_file | 是否需要获取银行卡 OCR 图片文件。<br>值为1则返回文件；其他则不返回 | 字符串 | 1 | 否，非必填 |
| nonce | 随机数 | 字符串 | 32 | 是 |
| version | 版本号，默认值：1.0.0 | 字符串 | 20 | 是 |
| sign | 签名值，使用本页第一步生成的签名 | 字符串 | 40 | 是 |

**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

### 响应

**响应参数：**

| 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 |

| 参数 | 类型 | 说明 |
| --- | --- | --- |
| bankcardCropPhoto | Base 64 String | 银行卡切边图片 |
| bankcardNoPhoto | Base 64 String | 银行卡卡号切边图片 |
| originBankcardPhoto | Base 64 String | 识别原始图片 |
| warnCode | String | 银行卡告警码，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【通用响应码列表 】>【银行卡 OCR 识别响应码】 |
| operateTime | String | 做 OCR 的操作时间 |
| multiWarnCode | String | 多重告警码，含义请参考【通用响应码列表 】 |
| clarity | String | 图片清晰度 |

**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。为确保用户操作整体流程顺利完成，部分情况下获取照片会有1秒左右的延迟。
- 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 OCR 识别有日期失效或者过期的情况，请参考 frontWarnCode 和 backWarnCode 告警码。（参考【通用响应码列表 】>【银行卡 OCR 识别响应码】）