OPPO 游戏中心_SDK 集成说明书（单机）

- OPPO -

文档编号	300001	**文档版本号**	V1.0
归属部门	游戏中心	**文档机密**	中
产品名	游戏 SDK	**产品版本**	v2.0
编写人	卜艳丽	**编写日期**	2018-06-22

修订记录

版本号	修订人	修订日期	修订内容
V1.0	张辉	2015 年 10 月 23 日	修订
V1.0	张辉	2016 年 03 月 24 日	修改初始化方式
V1.0	张辉	2016 年 04 月 22 日	新增支付选项
V1.0	张辉	2016 年 07 月 20 日	简化初始化方式，不再支持分省支付，必须开通全国支付
V1.0	张辉	2018 年 04 月 10 日	去除 onResume 和 onPause 接口
V1.0	张辉	2019 年 01 月 02 日	单机 sdk 增加账号功能（需添加部分权限）
V1.0	张辉	2019 年 09 月 10 日	1、适配 android Q 2、提供 aar 接入 sdk 3、新增 receiver action，去掉短信权限 4、新增上传角色信息接口、实名认证接口
V1.0	张辉	2019 年 12 月 04 日	客户端新增订单查询接口
V1.0	张辉	2020 年 04 月 24 日	支持防沉迷、无接口变动
v1.0	张辉	2020 年 06 月 10 日	1、版本升级 2、新增跳转超休闲专区接口
V2.8	张辉	2020 年 11 月 25 日	1、SDK UI 界面更新 2、增加儿童误充值识别接口
V2.8	张辉	2021 年 01 月 14 日	1、仅更新文档：补充了 androidx 相关的说明，在本文档正文第 2 页。
V3.0	张辉	2021 年 4 月 08 日	1、新增<检查 OPPO 软件商店是否有更新>接口
V3.1	张辉	2021 年 08 月 10 日	1、初始化时权限被拒 48 小时内不再重复申请
V3.3	张辉	2021 年 11 月 23 日	1、 最新的防沉迷版本，修复一些已知 bug。 2、 接口无变动。

V3.4	张辉	2022 年 2 月 15 日	1、 登录、实名认证流程优化。接口无变动。 2、 minSDKversion 变为 21。 3、 去 除 了 READ_PHONE_STATE 、 READ_EXTERNAL_STORAGE 、 WRITE_EXTERNAL_STORAGE 权限
V3.5	张辉	2022 年 3 月 10 日	1、 上线客服自助服务功能，接口无变动
V3.7	张辉	2022 年 5 月 31 日	1、assets 中的文件打包移置 sdk 中。

目录

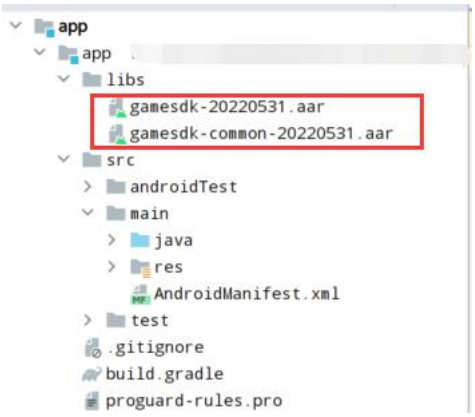
1. SDK 调用

1.1. 环境准备

开发工具可以使用 android studio。

1.1.1. 使用 OPPOsdk 提供的 aar 进行接入

- 将 sdk 压缩包中\resource\aar 接入资源文件夹下的资源对应放入游戏工程的 assets 文件、libs 文件目录下，如下图，具体也可参考 Demo 工程，具体文件以 SDK 压缩包中提供的文件版本为准。



注意：如果以前接入过旧版本，要升级到新版本，请删除原 assets 目录下的 oppo_game_service_***.so，nearme.apk，opay_version 三个文件，因为已经打包在 gamesdk-***.aar 中

包名：以开发者在 OPPO 开放平台创建的游戏包名为准（以.nearme.gamecenter 结尾）。

```
android {
    compileSdkVersion 28
    defaultConfig {
        applicationId "com.nearme.onlinesdk.nearme.gamecenter"
        minSdkVersion 19
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
```

配置 Android SDK 版本

SDK 支持 Android SDK 5.0 (Api Level 21) 及其以上版本,targetSdkVersion 须设置到 26 及以上。

```
<uses-sdk android:minSdkVersion="21"
          android:targetSdkVersion="26"/>
```

AndroidManifest 配置选项

```
<meta-data android:name="debug_mode"
           android:value="false" /> <!-- 调试开关，发布时候设置false -->
<meta-data android:name="is_offline_game"
           android:value="true" /> <!-- true:单机游戏 false:网游 -->
<meta-data android:name="app_key"
           android:value="c5217trjnrmU6gO5jG8VvUFU0" /> <!--appKey,务必换成
游戏自己的参数 -->
<uses-library android:name="org.apache.http.legacy" android:required="false" />
<!--9.0及以上设备可能需要 -->
```

具体请以 Demo 中 Manifest.xml 文件实际配置为准

- Gradle 配置

需要在游戏项目的 build.gradle 中配置，具体以使用的文件版本为准。

```
repositories {
    flatDir {
        dirs 'libs'    // aar 目录
    }
}
android{
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
}
dependencies {
    implementation (name: 'gamesdk-20220310', ext: 'aar')
    implementation (name: 'gamesdk-common-20220310', ext: 'aar')
    implementation 'androidx.legacy:legacy-support-v4:1.0.0'
//如果不方便使用 androidx  v4 的，可以用 SDK 压缩包中提供的 core-1.1.0 代替

}
```

- 混淆配置:

```
-keep class com.nearme.** { *; }
-dontwarn com.nearme.**
```

1.2. SDK 接口

SDK 中所有 API 方法都要求在主线程调用，耗时的 Api 方法 SDK 自行在非 UI 线程中运行，回调函数也将在主线程中执行，CP 不用关心线程的切换问题。GameCenterSDK 是游戏中心的 API 核心类，提供了全部的 API 方法，方法名大部分都是以 doXXX 开头.例如，游戏中心的初始化，请求登录，获取用户信息，支付，分享游戏信息等，SDK 中的所有 API 调用采用回调模式，回调的通用接口为:ApiCallback

```
public interface ApiCallback {
    /**
     * @param resultMsg
     */
    public void onSuccess(String resultMsg);
    /**
     * @param resultMsg
     * @param resultCode
     */
    public void onFailure(String resultMsg, int resultCode);
}
```

1.2.1. 初始化（必接）

游戏需在工程 Application 或者主进程其他位置尽早对 SDK 进行初始化。Appkey 在 manifest 文件中配置。Appsecret 在 java 代码中配置。

SDK 权限申请是在 SDK 初始化时申请。游戏可以自行添加权限说明，之后再调用 SDK 初始化。（本次调整：权限被拒 48 小时之内，不会重复申请）
权限说明：
~~1、运行时权限（弹框）有两个：存储和电话权限。在账号登录、以及实名认证防沉迷等方面都需要用到。~~（已去除）
2、其他敏感权限： GET_ACCOUNT 权限，一加 9 以下机型账号登录时需要获取系统账号进行登录。

```
public class DemoApplication extends Application {
    @Override
    public void onCreate() {
        super.onCreate();
        String appSecret = "e2eCa732422245E8891F6555e999878B";
        GameCenterSDK.init(appSecret, this);
    }
}
```

注意：调用以下任何接口前，请务必保证游戏已经成功初始化.
GameCenterSDK.init(appSecret, this)中的 context 的说明：
在 application 初始化需要传 application 的 context;
在 Activity 初始化需要传 Activity 的 context。

1.2.2. 登录

- 从联运 SDK V3.4 版本开始，SDK 初始化（init）后，将自动登录。跟游戏自己调用登录获取登录结果并无冲突。

```
GameCenterSDK.getInstance().doLogin(context, new ApiCallback() {

    @Override
    public void onSuccess(String resultMsg) {
        // 登录成功
    }

    @Override
    public void onFailure(String resultMsg, int resultCode) {
        // 登录失败
    }
});
```

- 该接口参数为当前 Context 及结果回调
- CP 可在回调接口的 onSuccess 方法中处理登录成功事件，在 onFailure 方法中处理登录失败事件
- 登录成功后，CP 需要调用获取 token 和 ssoid 信息接口拿到 token 和 ssoid，并将结果传给服务端，以方便服务端进行登录验签（具体步骤见《2.1 后台登录验签算法》）！
- CP 可在登录成功后调用获取用户信息接口，以获取当前用户详细信息。

1.2.3. 获取 Token 和 SsoId

```
GameCenterSDK.getInstance().doGetTokenAndSsoid(context, new ApiCallback() {
    @Override
    public void onSuccess(String resultMsg) {
```

```
            try {
                JSONObject json = new JSONObject(resultMsg);
                String token = json.getString("token");
                String ssoid = json.getString("ssoid");
            } catch (JSONException e) {
                e.printStackTrace();
            }
    }
    @Override
    public void onFailure(String resultMsg, int resultCode) {
    }
});
```

- 该接口参数为当前 Context 及结果回调.
- CP 可以调用该接口获取当前用的 token 和 ssoid，并用以获取用户详细信息.
- **注意**：token 中可能包含‘+’，‘/’之类的特殊符号，所以最好先对 token 进行 urlencode 之后再传递给游戏服务端.

1.2.4. 获取用户信息

```
GameCenterSDK.getInstance().doGetUserInfo(context,
    new ReqUserInfoParam(token, ssoid), new ApiCallback() {
    @Override
    public void onSuccess(String resultMsg) {
    }
   @Override
   public void onFailure(String resultMsg, int resultCode) {
    }
});
```

- 该接口参数为当前 Context、请求参数对象 ReqUserInfoParam 及结果回调，其中 ReqUserInfoParam 可以已经在 SDK 中定义，CP 可直接使用.
- CP 可根据需要使用获取到用户详细信息,其中成功获取的 content 内容格式为 JSON.
  eg：{“userName”:”xxx”,”mobile”:”12345678912”,”email”:””,
  ”ssoid”:”123456”,”channel”:”xxx”,adId:”xxx”}
- 注意：1、ssoid 为标记用户所需字段，是唯一保持不变的，其余字段为附加字段，可能会增减，请不要用附加字段作为进入游戏的限制条件。

1.2.5. 支付（必接）

```
GameCenterSDK.getInstance().doSinglePay(this, payInfo,
      new SinglePayCallback() {
   // add OPPO 支付成功处理逻辑~
    public void onSuccess(String resultMsg) {
        Toast.makeText(DemoActivity.this, "支付成功",Toast.LENGTH_SHORT).show();}
  // add OPPO 支付失败处理逻辑~
    public void onFailure(String resultMsg, int resultCode) {
        if (PayResponse.CODE_CANCEL != resultCode) {
        Toast.makeText(DemoActivity.this, "支付失败",Toast.LENGTH_SHORT).show();
           } else {      // 取消支付处理
    Toast.makeText(DemoActivity.this, "支付取消",Toast.LENGTH_SHORT).show();
                        }}
        /* bySelectSMSPay 为true表示回调来自于支付渠道列表选择短信支付，false表示没有网络等非主动选择短信支付时候的回调 */
    public void onCallCarrierPay(PayInfo payInfo, boolean bySelectSMSPay) {
                        // add 运营商支付逻辑~
    Toast.makeText(DemoActivity.this, "运营商支付",Toast.LENGTH_SHORT).show();
                    }
                });
```

其中 PayInfo 对象需要 CP 自己实现

```
private PayInfo createTestPayInfo(int amount){
    PayInfo payInfo = new PayInfo(System.currentTimeMillis() +
    new Random().nextInt(1000) + "", "自定义字段", amount);
    payInfo.setProductDesc("商品描述");
    payInfo.setProductName("300符石");
  // payInfo.setShowCpSmsChannel(true);//sdk支付界面是否显示运营商短信入口，true显示，false不显示
  // payInfo.setUseCachedChannel(true);//设置是否使用上次使用过的支付方式
  // 支付结果服务器回调地址，尽可能使用https，不通过服务端回调发货的游戏可以不用填写~
    payInfo.setCallbackUrl(
            "https://gamecenter.wanyol.com:8080/gamecenter/callback_test_url");
    return payInfo;
}
```

PayInfo 属性详细说明

名称	类型	长度（字符）	是否必填	说明
order	String	100	是	订单号，务必保证唯一
attach	String	200	否	自定义回调字段
amount	int		是	消费总金额，单位为分
productName	String	40	是	商品名（不能有+号等特殊符号）
productDesc	String	120	否	商品描述（不能有+号等特殊符号）
callbackUrl	String		否	支付回调地址,尽可能使用 https，不需要服务端回调的可不填

- SDK 中均按分计算，如若商品价值 5 元，那 amount 应填写 500，
- 调用此接口应保证订单唯一，示例中订单号由时间戳和一个随机值生成，CP 可以沿用，也可以自己定义，但务必保证订单号唯一
- 自定义字段为交易辅助说明字段，建议 CP 填写此次交易相关信息
- 回调地址为 CP 服务端回调地址，服务端接收到我方后台发出的成功回调后即可发放游戏道具

1.2.6. 上传玩家在游戏中的角色信息

- 接入了登录接口的游戏尽量接入此接口，方便后续做活动。

```
GameCenterSDK.getInstance().doReportUserGameInfoData(new ReportUserGameInfoParam(
    "roleId", "roleName", roleLevel, "realmId", "realmName", "chapter", ext),
     new ApiCallback() {
        @Override
        public void onSuccess(String resultMsg) {
             // success
        }
        @Override
        public void onFailure(String resultMsg, int resultCode) {
           // failure
        }
});
```

- 该接口需上传的参数对象为 ReportUserGameInfoParam、上传结果回调 ApiCallback，其中 ReportUserGameInfoParam、ApiCallback 已经定义，CP 可直接创建实例使用。
- 参数说明：

字段名称	类型	长度	是否必填	字段说明
roleId	String	最大 64 字节	必填	角色 id
roleName	String	最大 100 字节	必填	角色昵称
roleLevel	int		必填	角色等级
realmId	String	最大 64 字节	必填	区服 id

realmName	String	最大 100 字节	必填	区服名称
chapter	String	最大 64 字节	必填	关卡章节
ext	Map<String, Number>		必填	扩展字段 字段个数不超过 10 个 名称长度不超过 18 个字节 字段值必须是数字类型

关于扩展字段 ext 的要求如下：

字段	Key 值	类型	是否必填	说明
ext	combatValue	Number	否	战力值，有战力值的网游必须上传，活动工具需要根据战力值评估
	pointValue	Number	否	积分值，捕鱼或棋牌类休闲游戏参与比赛的积分必须上传，如果暂无积分值，游戏可自行设置积分值并上传。

注意：如果游戏没有其中的某个字段，可以传默认值，string 类型字段的默认值 default；int 类型字段的默认值是 0；Map<String, Number>类型字段的默认值是 null。

- CP 可在玩家首次成功登陆游戏服务器，并选定游戏角色信息后，调用该接口，上传玩家初始角色信息；**为了确保即时性和准确性：1、每次登陆进入游戏之后都要调用接口上报数据；2、当游戏中角色信息发生变化时要立即调用接口上报数据（角色名变更、角色等级、扩展字段信息等任意上传字段发生变化都要上报接口）。**

1.2.7. 游戏退出引导（退出游戏时必须调用）

```
GameCenterSDK.getInstance().onExit(Context context, GameExitCallback callback);
```

游戏准备退出时需要调用此方法，并在GameExitCallback的回调exitGame()中实现游戏真正的退出操作(比如:先进行数据存档，然后关闭游戏进程)，关闭游戏进程可以参考接入demo里面使用AppUtil工具类的exitGameProcess方法完成，也可以自己实现。

1.2.8. 实名认证防沉迷接口

```
GameCenterSDK.getInstance().doGetVerifiedInfo(new ApiCallback() {
    @Override
    public void onSuccess(String resultMsg) {
```

```
        try {
            //解析年龄（age）
            int age = Integer.parseInt(resultMsg);
       if (age < 18) {
                //已实名但未成年，CP开始处理防沉迷
            } else {
                //已实名且已成年，尽情玩游戏吧
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onFailure(String resultMsg, int resultCode) {
   if(resultCode == ApiResult.RESULT_CODE_VERIFIED_FAILED_AND_RESUME_GAME){
       //实名认证失败，但还可以继续玩游戏
     }else if(resultCode ==
            ApiResult.RESULT_CODE_VERIFIED_FAILED_AND_STOP_GAME){
       //实名认证失败，不允许继续游戏，CP需自己处理退出游戏
                                }
                }
});
```

- 该接口参数为结果回调。
- CP 可以调用该接口获取当前用户实名认证的年龄，如用户未进行实名认证，会先弹出实名认证的界面，如用户已认证则返回 age（年龄），游戏可以根据 age 来判断用户是否成年.

1.2.9. 订单查询接口

```
    GameCenterSDK.getInstance().checkPayResult("xxx", new ApiCallback() {
          @Override
          public void onSuccess(String resultMsg) {

          }

          @Override
          public void onFailure(String resultMsg, int resultCode) {

          }
    });
```

- 接口中的第一个参数为 CP 订单号，即支付接口中 order 字段中传递的参数，并非 attach 等字段的参数。
- 当 CP 在客户端调用支付接口并未收到订单结果时，可以在下次进入游戏进行查询。
- 失败回调中 `resultCode` 包含以下情况：`10001=没查询到下单记录；10002=订单未支付；10003=请求失败`

1.2.10. 跳转超休闲专区（超休闲游戏必接）

```
GameCenterSDK.getInstance().jumpLeisureSubject();
```

- 该接口无需传参，也没有回调，如果发生错误，OPPOSDK 会有相应的提示，无需游戏处理。
- 如没有特别说明，可以在游戏内的任何界面调用接口，游戏自己决定调用的位置，调用接口之后，将直接跳转到 OPPO 游戏中心超休闲专区。
- 接口供**超休闲游戏**调用且**必须**调用；其它类型的游戏，比如网游、棋牌等不要调用。

1.2.11. 儿童误充值识别

该能力主要是 SDK 用来辅助识别是否是儿童充值。没有交互，也不需要游戏进行测试，只需要调用接口即可。
Cp 可以自己确定是否接入，如果要接入则有两个接口，都需要调用。如下：

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

     GameCenterSDK.getInstance().startChildrenMonitor(this);

}
```

- 上边的接口需要在游戏**主 activity 的 onCreate** 方法中调用。

```
    public boolean dispatchTouchEvent(MotionEvent ev) {
        GameCenterSDK.getInstance().touchMonitor(ev);
        return super.dispatchTouchEvent(ev);
    }
```

- 上边的接口需要在游戏**主 Activity** 的 **dispatchTouchEvent** 方法中调用。

1.2.12. 检查 OPPO 软件商店是否有更新

游戏可以通过调用该接口检查游戏在 OPPO 软件商店的版本是否有更新，如果有更新则会跳到 OPPO 软件商店自动触发下载和安装。

```
GameCenterSDK.getInstance().getNewestVersionCode(new ApiCallback() {

    @Override
    public void onSuccess(String resultMsg) {

    }

    @Override
    public void onFailure(String resultMsg, int resultCode) {

    }
})
    ;
```

- 参数说明：

接口名	参数名	参数说明
getNewestVersionCode	ApiCallback	接收检查更新结果。 onSuccess():有更新且跳转到商店自动更新。 onFailure():无更新。

- 需要在调用 SDK 初始化 init 接口之后调用。

2. 服务端

2.1. CP 后台登录验签算法(非必接)

该说明主要用于 CP 服务端登录验签。部分单机游戏且无服务端登录验签需要的伙伴请忽略此文档.

- OPPO 游戏中心的开放账号采用 oauth1.0 身份认证的方式。
- 游戏客户端可以通过 api 获取用户的信息以及 token 串。
- 游戏可利用用户信息中的**唯一 ssoid** 标示用户(**一定要用 ssoid 标记用户**),游戏服务端可以利用游戏客户端从 SDK 获取的 token 在首次登陆的时候验证用户的身份，以防止客户端被破解后的非法注册
- 网游接入账号的时候，若要求用户安全性高，可以利 token 和 ssoid 在游戏的服务端做 Oauth 验证，单机游戏可以不做 Oauth 验证。

注：该说明均以 java 环境为例

2.1.1. 获取 Token 和 ssoid

用户首次登录时,客户端通过 sdk 登录后获取 token 和 ssoid,并将用户信息传给游戏服务端.获取接口为:

```
GameCenterSDK.getInstance().doGetTokenAndSsoid(Context context , ApiCallback callback);//返回数据为 token 和 ssoid 的 Json 格式合并参数
获取及解析方式如下:
    GameCenterSDK.getInstance().doGetTokenAndSsoid(this, new ApiCallback() {
            @Override
            public void onSuccess(String content, int requestCode) {
                if(requestCode == 1001){
                    try {
                        JSONObject json = new JSONObject(content);
                        String token = json.getString("token");
                        String ssoid = json.getString("ssoid");
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }
            }
            @Override
```

```
            public void onFailure(String content, int resultCode) {
            }
        });
```

注意：获取到 token 之后，先对 token urlencode 之后，再传递给游戏服务端。

2.1.2. 发送 HTTP GET 请求，请求用户信息

游戏服务端利用token和ssoid向游戏中心服务端发送用户信息请求,验证是否有正常返回,用户信息是否一致(验证 ssoid 即可)，一致即认为验签成功。

请求方式：GET

请求地址(有改动，最新地址如下，红色部分)：

https://iopen.game.oppomobile.com/sdkopen/user/fileIdInfo?fileId=ssoid&token=URLEncoder.encode(token,"UTF-8")

注意：url 中的 token 需要 urlencode，但不能进行多次 urlencode

在 GET 方式的 header 中携带请求信息：

请求头名 1：param

请求头值 1：基串 **baseStr**

请求头名 2：oauthSignature

请求头值 2：签名 **Sign**

以下是基串 **baseStr** 以及签名 **Sign** 的生成规则(java)：

baseStr:**oauthConsumerKey=xxx&oauthToken=xxx&oauthSignatureMethod=HMAC-SHA1&oauthTimestamp=xxx&oauthNonce=xxx&oauthVersion=1.0&**

其中：**oauthConsumerKey：即 appkey（在开发者后台创建游戏时生成）**

oauthToken：即由游戏客户端解析之后获取的 token，需要客户端传给服务端，需要 urlencode，但注意不能多次 urlencode

oauthSignatureMethod：HMAC-SHA1

oauthTimestamp： 即时间戳（cp 自己生成）

oauthNonce：即随机数（cp 自己生成）

oauthVersion=1.0

注意：token 本身并不含有空格，对 token encode 之后，即不会产生“+”号，如果 cp encode（token，“UTF-8”）之后，发现 token 中含有“+“号，那可能是在解析 token 的过程中出现了问题，导致解析之后的 token 含有空格，从而在 encode 之后产生了“+”号，总之 encode 之后的 token 中不能出现“+”号。下图为错误 token 示例：

https://iopen.game.oppomobile.com/sdkopen/user/fileIdInfo?fileId=208066711&token=TOKEN_1WDGCqqvW8hDmu92TOhxbCohevAB8Vp7NiddgnamgqFtq9GILvqdhUMqy+huprqU

以下是正确的 token：

https://iopen.game.oppomobile.com/sdkopen/user/fileIdInfo?fileId=208066711&token=TOKEN_1WDGCqqvW8hDmu92TOhxbCohevAB8Vp7NiddgnamgqFtq9GILvqdhUMqy%2Bhuprq U

Sign：即由 **appSecret + &** 作为签名的 **key**，对上边生成的 **baseStr** 进行 **HMAC-SHA1** 加密，其中 **appSecret** 是在开发者后台创建游戏时生成的，和 **appKey** 一样，需要在开发者后台获取。

```
public static final String OAUTH_CONSUMER_KEY = "oauthConsumerKey";
public static final String OAUTH_TOKEN = "oauthToken";
public static final String OAUTH_SIGNATURE_METHOD = "oauthSignatureMethod";
public static final String OAUTH_SIGNATURE = "oauthSignature";
public static final String OAUTH_TIMESTAMP = "oauthTimestamp";
public static final String OAUTH_NONCE = "oauthNonce";
public static final String OAUTH_VERSION = "oauthVersion";
public static final String CONST_SIGNATURE_METHOD = "HMAC-SHA1";
public static final String CONST_OAUTH_VERSION = "1.0";

public static String generateBaseString(String timestamp,String nonce){

    StringBuilder sb = new StringBuilder();
    try {
     sb.append(OAUTH_CONSUMER_KEY).
     append("=").
     append(URLEncoder.encode(Constant.AppKey,"UTF-8")).
     append("&").
        append(OAUTH_TOKEN).
        append("=").
        append(URLEncoder.encode(Constant.Token,"UTF-8")).
        append("&").
        append(OAUTH_SIGNATURE_METHOD).
        append("=").
        append(URLEncoder.encode(CONST_SIGNATURE_METHOD,"UTF-8")).
        append("&").
        append(OAUTH_TIMESTAMP).
        append("=").
        append(URLEncoder.encode(timestamp,"UTF-8")).
        append("&").
        append(OAUTH_NONCE).
        append("=").
        append(URLEncoder.encode(nonce,"UTF-8")).
        append("&").
        append(OAUTH_VERSION).
        append("=").
        append(URLEncoder.encode(CONST_OAUTH_VERSION,"UTF-8")).
        append("&");
    } catch (UnsupportedEncodingException e1) {
         // TODO Auto-generated catch block
         e1.printStackTrace();
    }
    return sb.toString();
      }
```

生成签名 Sign 的示例代码：

```
public static String generateSign(String baseStr){
    byte[] byteHMAC = null;
    try {
        Mac mac = Mac.getInstance("HmacSHA1");
        SecretKeySpec spec = null;
        String oauthSignatureKey =AppSecret + "&";
        spec = new SecretKeySpec(oauthSignatureKey.getBytes(),"HmacSHA1");
        mac.init(spec);
        byteHMAC = mac.doFinal(baseStr.getBytes());
    } catch (InvalidKeyException e) {
        e.printStackTrace();
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    }
    return URLEncoder.encode(String.valueOf(base64Encode(byteHMAC)),"UTF-8");
 }
```

```
public static char[] base64Encode(byte[] data) {
        final char[] alphabet =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/="
                .toCharArray();
        char[] out = new char[((data.length + 2) / 3) * 4];
        for (int i = 0, index = 0; i < data.length; i += 3, index += 4) {
            boolean quad = false;
            boolean trip = false;
            int val = (0xFF & (int) data[i]);
            val <<= 8;
            if ((i + 1) < data.length) {
                val |= (0xFF & (int) data[i + 1]);
                trip = true;
            }
            val <<= 8;
            if ((i + 2) < data.length) {
                val |= (0xFF & (int) data[i + 2]);
                quad = true;
            }
            out[index + 3] = alphabet[(quad ? (val & 0x3F) : 64)];
            val >>= 6;
```

```
            out[index + 2] = alphabet[(trip ? (val & 0x3F) : 64)];
            val >>= 6;
            out[index + 1] = alphabet[val & 0x3F];
            val >>= 6;
            out[index + 0] = alphabet[val & 0x3F];
        }
        return out;
    }
```

2.1.3. 返回值描述

接口请求成功的返回值格式为

```
{"resultCode":"200","resultMsg":"正常","ssoid":123456,"userName": “abc” ,"email":
 “abc1234@163.com” ,"mobileNumber": “18612345678 “}
```

成功获取此信息，且 ssoid 为目标 ssoid，即认为验签成功。

其中：

参数名	类型	说明
resultCode	string	结果返回码，正常：200
resultMsg	string	结果返回描述
ssoid	string	当前账号唯一标识
userName	string	当前账号用户名
email	string	当前账号绑定的邮箱
mobileNumber	int	当前账号绑定的手机号

以下是一些完整的参数请求：

GET 请求：

```
      https://iopen.game.oppomobile.com/sdkopen/user/fileIdInfo?fileId=27352387&t
oken=TOKEN_mpWEc25NDr2HzRXQAAMFB%2Fd77Rhr3PxePY4W0BC%2BlOBQ%2BwWpf8W%2Fvg%3D%3D
```

请求头 param 样式：

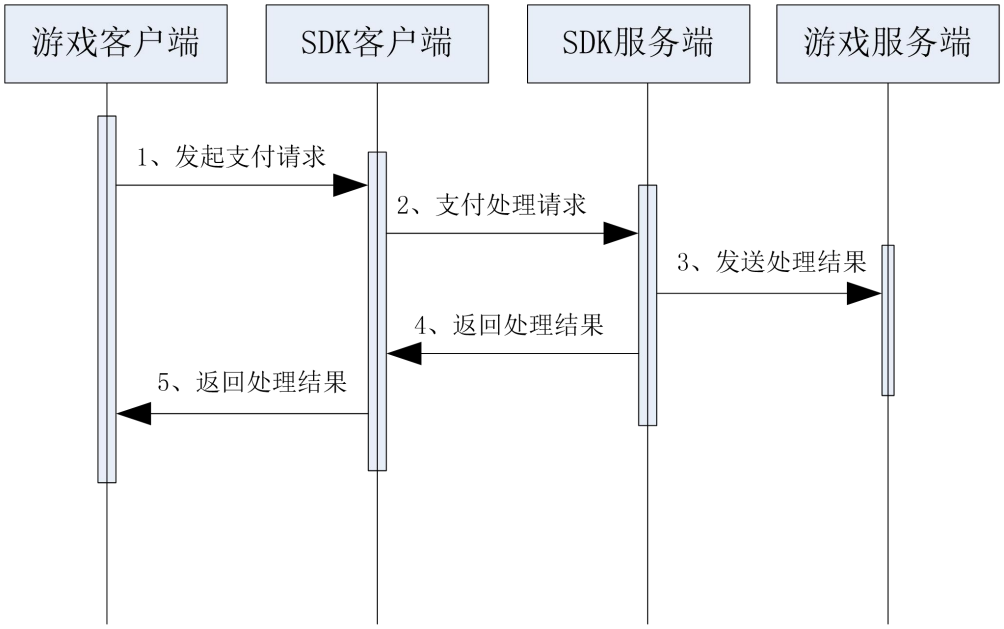
```
param:oauthConsumerKey=93b014fbe9304920ac9d07e50f5eb91b&oauthToken=TOKEN_mpWEc25N
Dr2HzRXQAAMFB%2Fd77Rhr3PxePY4W0BC%2BlOBQ%2BwWpf8W%2Fvg%3D%3D&oauthSignatureMethod
=HMAC-SHA1&oauthTimestamp=1445910766&oauthNonce=-251668506&oauthVersion=1.0&
```

请求头 oauthSignature 样式：

```
oauthSignature:7tzL%2BpBK2Xy9UCCPbhCCpGqQIfE%3D
```

2.2. 支付处理流程（非必接）

2.2.1. 示意图：



支付成功后，SDK 服务器会根据开发上传的回调地址（客户端代码 payinfo->setcallbackurl 中设置）通知开发者订单信息，回调方法为 HTTP POST。Content-type:application/x-www-form-urlencoded。

2.2.2. 回调参数说明：

参数名	类型	长度限制	说明
notifyId	string	50	回调通知 ID（该值使用系统为这次支付生成的订单号）
partnerOrder	string	100	开发者订单号（客户端上传）
productName	string	40	商品名称（客户端上传）

productDesc	string	120	商品描述（客户端上传）
price	int		商品价格(以分为单位)
count	int		商品数量（一般为 1）
attach	string	200	请求支付时上传的附加参数（客户端上传）
sign	string		签名

2.2.3. 验证签名方法&示例

签名步骤

1) 利用回调的参数生成baseString，拼接顺序：
notifyId=xxx&partnerOrder=xxx&productName=xxx&productDesc=xxx&price=xxx&count=xxx&attach=xxx。
为空的也需要参与签名。

```
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        req.setCharacterEncoding("UTF-8");
        NotifyRequestEntity e = new NotifyRequestEntity();
        e.setNotifyId(req.getParameter("notifyId"));
        e.setPartnerOrder(req.getParameter("partnerOrder"));
        e.setProductName(req.getParameter("productName"));
        e.setProductDesc(req.getParameter("productDesc"));
        e.setPrice(Integer.parseInt(req.getParameter("price")));
        e.setCount(Integer.parseInt(req.getParameter("count")));
        e.setAttach(req.getParameter("attach"));
        e.setSign(req.getParameter("sign"));
        String baseString = getBaseString(e);
        }

    private String getBaseString(NotifyRequestEntity ne) {
        StringBuilder sb = new StringBuilder();
        sb.append("notifyId=").append(ne.getNotifyId());
        sb.append("&partnerOrder=").append(ne.getPartnerOrder());
        sb.append("&productName=").append(ne.getProductName());
        sb.append("&productDesc=").append(ne.getProductDesc());
        sb.append("&price=").append(ne.getPrice());
        sb.append("&count=").append(ne.getCount());
        sb.append("&attach=").append(ne.getAttach());
        return sb.toString();
}
```

2） 对baseString进行验证签名（SHA1WithRSA算法）

```
public static boolean doCheck(String content, String sign, String publicKey) {

    try {
        KeyFactory keyFactory = KeyFactory.getInstance("RSA");
        byte[] encodedKey = Base64.base64Decode(publicKey);
        PublicKey pubKey = keyFactory.generatePublic(new
    X509EncodedKeySpec(encodedKey));

       java.security.Signature signature =
            java.security.Signature.getInstance("SHA1WithRSA");

        signature.initVerify(pubKey);
        signature.update(content.getBytes("utf-8"));
        boolean bverify = signature.verify(Base64.base64Decode(sign));
        return bverify;

    } catch (Exception e) {
            logger.error("验证签名出错.",e);
    }

    return false;
}
```

签名公钥

验证签名使用的公钥（在sdk压缩包中）：

2.2.4. 游戏服务端回调处理

CP 收到我方回调后需要回写处理结果。在收到回调后应**即时（200ms 内）**回写结果给我方服务器，游戏发货应采取**异步**方式处理，不能等待游戏发货完成之后再回写结果给我方服务器，等待过程很容易出现超时、请求断掉等情况，如果同一笔订单连续通知 3 次没有收到开发者回写，该订单会从回调队列删除，并将订单状态记入数据库。

返回结果数据格式：

result=arg0&resultMsg=arg1

arg0：值为“OK”或“FAIL”。两者选其一。该值为必填字段

arg1：arg0 为“OK”时该值可以为空字符串，arg0 为“FAIL”时建议提供些有意义的信息，便于查找问题，该值非强制字段。

例如：

result=OK&resultMsg=成功

result=FAIL&resultMsg=网络原因发货失败

注意：收到开发者回写结果，当 result=OK 时，表示 CP 已正常接收到回调，我方不再重复回调；凡是已经处理成功的订单，请务必回复 result=OK；请严格按照以上格式回写结果，避免二次修改。