OPPO 游戏中心_SDK 集成说明书（网游）

- 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**	2015-10-22

修订记录

版本号	修订人	修订日期	修订内容
V1.0	卜艳丽	2015 年 10 月 22 日	创建
V1.0	张辉	2015 年 12 月 17 日	修订登录验证
V1.0	张辉	2016 年 03 月 24 日	调整初始化方式

目录

1. SDK调用

1.1. 环境准备

1.1.1. 资源导入

将 sdk 压缩包中 resource 文件夹下的资源对应放入游戏工程的 assets 文件、values、libs 文件目录下，具体请参考 Demo 工程

1.1.2. Manifest.xml文件配置

包名

游戏包名必须以 .nearme.gamecenter 结尾

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.gcsdk.demo.nearme.gamecenter"
    android:versionCode="200"
    android:versionName="v2.0" >
```

配置 Android SDK 版本

SDK 支持 Android SDK 3.0（Api Level 11）及其以上版本

```
<uses-sdk android:minSdkVersion="11"
          android:targetSdkVersion="19"/>
```

加入所需的 permission 权限

```
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.SEND_SMS" />
    <uses-permission android:name="android.permission.READ_SMS" />
    <uses-permission android:name="android.permission.WRITE_SMS" />
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"
```

```
/>
    <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
    <uses-permission android:name="android.permission.GET_TASKS" />
    <uses-permission android:name="android.permission.GET_ACCOUNTS" />
    <uses-permission android:name="android.permission.USE_CREDENTIALS" />
```

加入所需的组件

具体请以 Demo 中 Manifest.xml 文件实际配置为准

```
<!-- nearme game sdk config goes here -->
<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="false" />   <!-- true:单机游戏   false:网游 -->
<meta-data android:name="app_key"
           android:value="c5217trjnrmU6gO5jG8VvUFU0" />  <!-- appKey -->
<activity
    android:name="com.nearme.game.sdk.component.proxy.ProxyActivity"
    android:theme="@style/Theme_Dialog_Custom"
    android:configChanges="keyboardHidden|orientation|screenSize"
    android:process=":gcsdk">
</activity>
<service
    android:name="com.nearme.game.sdk.component.proxy.ProxyApiService"
    android:process=":gcsdk"
     android:priority="1000">
</service>
<!-- UserCenter SDK Register start -->
<receiver
android:name="com.nearme.game.sdk.component.proxy.ProxyUserCenterOperateReceiver"
    android:exported="true"
     android:process=":gcsdk">
     <intent-filter>
       <action android:name="com.oppo.usercenter.account_login" />
       <action android:name="com.oppo.usercenter.account_logout" />
        <action android:name="com.oppo.usercenter.modify_name" />
      </intent-filter>
</receiver>
     <!-- nearme game sdk config end -->
```

其中，*Theme_Dialog_Custom* 需要 CP 在工程 res/value/style 文件中添加,具体请参考 demo 工程中 res/value/style 文件

1.1.3. 初始化

CP 需在工程 Application 或者主进程其他位置尽早对 SDK 进行初始化

```
public class DemoApplication extends Application {
    @Override
    public void onCreate() {
        super.onCreate();
// 因为sdk插件中service activity跑在独立进程，这里只需要在主进程做一次初始化操作。
    if(getApplicationInfo().packageName.equals(AppUtil
            .getCurrentProcessName(this))){
            String appSecret = "e2eCa732422245E8891F6555e999878B";
            GameCenterSDK.init(appSecret, this);
        }
    }
}
```

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);
}
```

注：调用以下任何接口前，请务必保证游戏已经成功初始化.

1.2.1. 登录

```
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。

1.2.2. 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.

1.2.3. 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”}

1.2.4. 支付

```
GameCenterSDK.getInstance().doPay(context, payInfo, new ApiCallback() {
    @Override
    public void onSuccess(String resultMsg) {
        // 支付成功
    }
    @Override
    public void onFailure(String resultMsg, int resultCode) {
```

```
        // 支付失败
    }
});
```

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.setCallbackUrl(
            "http://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
productDesc	String	120	否	商品描述
callbackUrl	String		是	回调地址

- SDK 中可币均按分计算，如若商品价值 5 可币，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

1.2.5. onResume & onPause（控制浮标的显示和隐藏，需成对调用）

```
@Override
protected void onResume() {
```

```
    super.onResume();
    GameCenterSDK.getInstance().onResume(this);
}

@Override
protected void onPause() {
    GameCenterSDK.getInstance().onPause();
    super.onPause();
}
```

CP 必须在游戏主页面调用这对接口来通知 sdk 做一些状态控制，注意这两个接口必须要成对调用.

1.2.6. 上传玩家在游戏中角色信息

```
GameCenterSDK.getInstance().doReportUserGameInfoData(context,
    new ReportUserGameInfoParam(gameId, service, role, grade),new ApiCallback() {
    @Override
    public void onSuccess(String resultMsg) {
        // success
    }
    @Override
    public void onFailure(String resultMsg, int resultCode) {
        // failure
    }
});
```

- 该接口参数分别为当前 Context、需上传的参数对象 ReportUserGameInfoParam、上传结果回调，其中 ReportUserGameInfoParam 已经定义，CP 可直接使用
- CP 可在玩家成功登陆游戏服务器，并选定游戏角色定信息后，调用该接口，方便我方运营同学同步了解游戏当前情况，以即时制定或调整运营策略

参数说明：gameId（就是 appid，在开发者后台创建游戏的时候生成的）；
service（游戏角色所在的服务器名称或者服务器 id）；
role（角色名称）
grade（角色等级）

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。

2. 服务端

2.1. CP后台登录验签算法

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) {
            }
        });
```

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 id 即可)，一致即认为验签成功。

请求方式：GET

请求地址：

http://i.open.game.oppomobile.com/gameopen/user/fileIdInfo?fileId=URLEncoder.encode(ssoid , "UTF-8")&token= URLEncoder.encode(token,"UTF-8")

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 ,用之前要 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：

```
http://i.open.game.oppomobile.com/gameopen/user/fileIdInfo?fileId=27352387&
token=TOKEN_mpWEc25NDr2HzRXQAAMFB%2Fd77Rhr3PxePY4W0B +LOBQ%2BwWpf8W%2Fvg%3D%3D
```

以下是正确的 token：

```
http://i.open.game.oppomobile.com/gameopen/user/fileIdInfo?fileId=273523873&
token=TOKEN_mpWEc25NDr2HzRXQAAMFB%2Fd77Rhr3PxePY4W0BQ%2B1OBQ%2BwWpf8W%2Fvg%3D%3D
```

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
","loginToken":xxx,"ssoid":123456,"appKey":null,"userName":abc,"email":null,"mobi
leNumber":null,"createTime":null,"userStatus":null}
```

成功获取此信息，且 ssoid 为目标 ssoid，即认为验签成功。

以下是一些完整的参数请求：
GET 请求：

```
      http://i.open.game.oppomobile.com/gameopen/user/fileIdInfo?fileId=27352387&
token=TOKEN_mpWEc25NDr2HzRXQAAMFB%2Fd77Rhr3PxePY4W0BC%2BlOBQ%2BwWpf8W%2Fvg%3D%3D
```

请求头 param 样式：

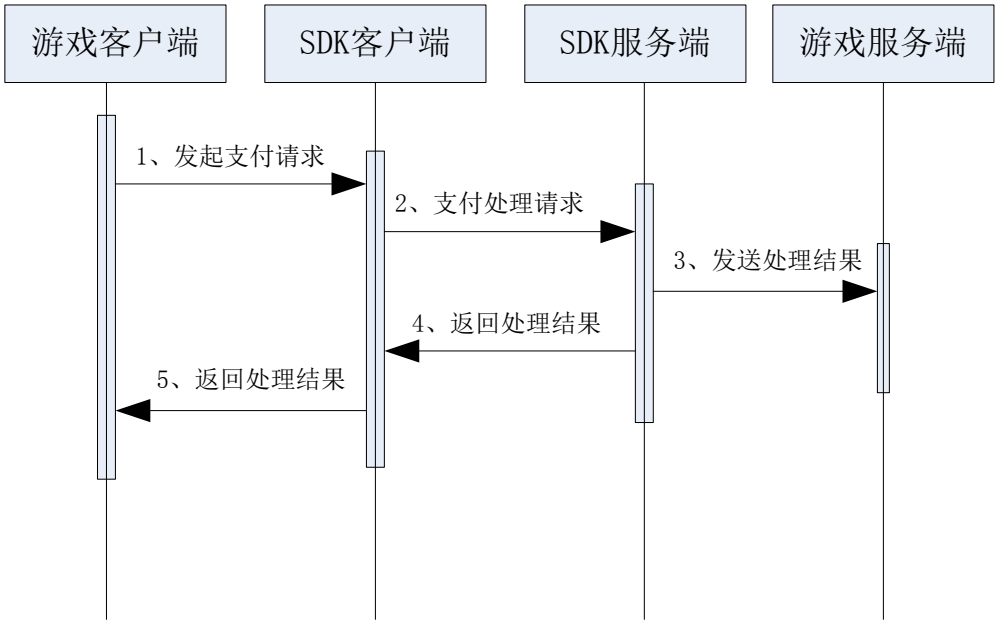
```
param:oauthConsumerKey=93b014fbe9304920ac9d07e50f5eb91b&oauthToken=TOKEN_mpWEc25N
Dr2HzRXQAAMFB%2Fd77Rhr3PxePY4W0BC%2BlOBQ%2BwWpf8W%2Fvg%3D%3D&oauthSignatureMethod
=HMAC-SHA1&oauthTimestamp=1445910766&oauthNonce=-251668506&oauthVersion=1.0&
```

请求头 oauthSignature 样式：

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

2.2. 消耗可币处理流程

2.2.1. 示意图：



消耗可币成功后，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。

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	50	商品名称（客户端上传）
productDesc	string	100	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，方法如下：

```
private static String getKebiContentString(String url){
     final String[] strings = url.split("&");
     final Map<String, String> data = new HashMap<String, String>();
     for(String string : strings){
         final String[] keyAndValue = string.split("=");
         data.put(keyAndValue[0], keyAndValue[1]);
     }
     StringBuilder sb = new StringBuilder();
     sb.append("notifyId=").append(data.get("notifyId"));
     sb.append("&partnerOrder=").append(data.get("partnerOrder"));
     sb.append("&productName=").append(data.get("productName"));
     sb.append("&productDesc=").append(data.get("productDesc"));
     sb.append("&price=").append(data.get("price"));
     sb.append("&count=").append(data.get("count"));
     sb.append("&attach=").append(data.get("attach"));
     return sb.toString();
}
```

2） 对baseString进行验证签名

```
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;
    }
```

相关 java 代码

具体的Java代码在RsaUtil.java中

名称	大小	压缩后大小	类型	修改时间	CRC32
..			文件夹		
oppo_kebi_charge_demo.php	906	607	PHP 文件	2013/12/4 15...	C12CE4E4
pay_rsa_public_key.pem	272	237	PEM 文件	2013/12/12 1...	0901C78B
可币支付回调公钥.txt	216	199	Notepad++ Doc...	2013/11/8 15...	EABA84...
可币支付回调协议.doc	178,227	143,782	Microsoft Word ...	2013/11/11 1...	06A43A...
可币支付回调验签原理说明.txt	560	368	Notepad++ Doc...	2013/12/6 14...	FF97C16F
可币支付回调验证签名Java_Demo.java	4,029	1,723	JAVA 文件	2013/11/11 1...	04FBC1A0
可币支付回调验证签名WebServer_J2EE_Demo.java	4,650	1,597	JAVA 文件	2014/3/13 17...	838FECC6

签名公钥

验证签名使用的公钥：

名称	大小	压缩后大小	类型	修改时间	CRC32
..			文件夹		
oppo_kebi_charge_demo.php	906	607	PHP 文件	2013/12/4 15...	C12CE4E4
pay_rsa_public_key.pem	272	237	PEM 文件	2013/12/12 1...	0901C78B
可币支付回调公钥.txt	216	199	Notepad++ Doc...	2013/11/8 15...	EABA84...
可币支付回调协议.doc	178,227	143,782	Microsoft Word ...	2013/11/11 1...	06A43A...
可币支付回调验签原理说明.txt	560	368	Notepad++ Doc...	2013/12/6 14...	FF97C16F
可币支付回调验证签名Java_Demo.java	4,029	1,723	JAVA 文件	2013/11/11 1...	04FBC1A0
可币支付回调验证签名WebServer_J2EE_Demo.java	4,650	1,597	JAVA 文件	2014/3/13 17...	838FECC6

2.2.4. 游戏服务端回调处理

CP 收到回调之后需要回写处理结果。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：收到开发者回写结果，表示 CP 已正常接收到回调，无论回写的结果是成功还是失败，将改订单从回调队列中删除，并将回调结果信息更新到数据库。