

360 安卓应用接入文档

版本号	修改时间	内容	修改人	确认人
0.2	2012-09-19	初稿	东玮、蔡文华、张玉智、 余帅	
0.3	2012-11-06	第三期文档变更	余帅	
0.3	2012-11-07	第三期文档修正	郭轶尊、何倩、盖祥省	
0.3	2012-11-08	第三期文档修正	余帅	
0.4	2012-11-22	第四期文档变更	郭轶尊	
0.5	2012-12-12	第五期文档变更	孙智彬、余帅	

360 应用开放平台 2012 年 12 月 26 日

目录

1. 概述.....	2
2. 接入准备.....	2
2.1 接入之前.....	2
2.2 使用 SDK.....	2
3. 登录授权流程.....	4
3.1 流程介绍.....	4
3.2 接口介绍.....	5
3.2.1 登录- SDK 接口, 应用客户端调用.....	5
3.2.2 获取 access token-服务器端接口, 应用服务器调用.....	8
3.2.3 获取用户信息-服务器端接口, 建议应用服务器调用.....	9
3.2.4 换新 access token-服务器端接口, 应用服务器调用.....	11
4. 支付流程.....	12
4.1 流程介绍.....	12
4.2 接口介绍.....	13
4.2.1 支付接口- SDK 接口, 应用客户端调用.....	13
4.2.2 支付结果通知接口-应用服务器提供接口, 由 360 服务器回调.....	15
4.2.3 订单核实接口- 服务器端接口, 应用服务器调用.....	16
5. 附录 :	18
5.1 签名算法 :	18
5.2 Demo 工程简介.....	19
5.2.1 App 文件 GameApplication.java.....	19
5.2.2 主界面文件 DemoMain.java.....	20
5.2.3 可配置登录界面 DemoLogin.java.....	20
5.2.4 可配置支付界面 DemoPay.java.....	21

1. 概述

本文档描述安卓手机应用通过 360 开放平台接口接入 360 平台的技术过程. 360 开放平台为安卓应用提供登录服务与支付服务.

本文档面向的读者是安卓应用开发者.

2. 接入准备

2.1 接入之前

安卓应用接入, 走 OAuth2 流程. 开发者可阅读以下文档来了解 OAuth2 流程

http://wiki.dev.app.360.cn/index.php?title=OAuth_2.0%E6%96%87%E6%A1%A3

开发者在调用接口之前, 需要先在应用开放平台 http://open.app.360.cn/?from=open_dt 申请 app_key 和 app_secret. 前者是应用的唯一标识, 后者相当于是应用的密钥.

注意：

app_key 与应用是一对一的关系, 一个 app_key 只能分配给一个应用使用. 若多个应用使用一个 app_key 会导致不能正常计费, 影响收入; 还会导致 SDK 升级失败、各项统计数据出错等严重问题, 影响开发者的收入和用户体验.

为了安全, app_secret 不能写进手机应用里, 只能存储在应用的服务器上. 以防反编译手机应用获取此信息, app_secret 泄漏可能用于伪造支付成功通知消息, 导致合作双方经济受损.

2.2 使用 SDK

配置 SDK 开发环境

1. 配置应用工程的 AndroidManifest.xml

您需要在应用工程 AndroidManifest.xml 里面添加权限：

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
```

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
```

添加activity：

```
<activity
    android:name="com.qihoopay.insdk.activity.ContainerActivity"
    android:configChanges="orientation|keyboardHidden|navigation">
</activity>
```

添加 meta-data，注意必须放入<application>元素区块内：

```
<!-- QHOPENSdk 配置:App Key, 必需-->
<meta-data android:value="8689.....ce26" android:name="QHOPENSdk_APPKEY"/></meta-data>
<!-- QHOPENSdk配置:Private Key, md5(app_secret + "#" + app_key), 全小写, 必需-->
<meta-data android:value="4e04fe9ac.....eb9" android:name="QHOPENSdk_PRIVATEKEY"/></meta-data>
<!-- QHOPENSdk配置:渠道, 非必需 -->
<meta-data android:value="default" android:name="QHOPENSdk_CHANNEL"/></meta-data>
```

2. 将 360SDK.jar 和 360SDK_Common.jar 复制到应用工程 libs 目录下，如果没有就建一个 libs 目录。
3. 将 res.zip 内文件手动解压到应用工程 assets/res/目录下。
4. 将 pro.jar 包复制到应用工程 assets 目录下。
5. LOGO: 在应用工程 assets/res/替换一个 480*129 的游戏 LOGO 图片 qihoo_pay_game_pay_icon.jpg。这个 LOGO 会显示在登录界面之上。
6. 支付: 在 assets/加入 alipay_plugin223_0309.apk 这个包, 用于支付宝支付。
7. SDK 使用
 - a. 在应用 Application 主程序里 **import** com.qihoopay.insdk.matrix.Matrix;
 - b. 应用 Application 类的 onCreate()函数中，执行 Matrix.init(context)
 - c. 应用 Application 类的 onTerminate()函数中，执行 Matrix.onDestroy()

3. 登录授权流程

通过登录授权流程，安卓应用可以获得当前用户的 access token，以此可获得当前用户的登录信息，如用户 id。获得用户 id 后，应用可以接入 360 的支付系统。

3.1 流程介绍

360 开放平台的登录流程有两种，token 模式和 code 模式

Code 模式– 标准 OAuth 流程，用 authorization code 换取 access token，推荐使用

- (1). 应用调用 SDK 登录接口 (3.2.1 节)
- (2). SDK 展示 360 开放平台服务端的页面，引导用户完成登录、授权
- (3). 360 开放平台服务端给 SDK 返回 authorization code (简称 authorization code)
- (4). SDK 返回 authorization code 给应用
- (5). 应用把 authorization code 传给服务器，从服务器调用 360 服务器接口换取 access token (3.2.2 节)
- (6). 应用使用 access token 调用 360 服务器接口获得用户信息 (3.2.3 节)

Code 模式需要应用服务器介入，用 authorization code 换取 access token。由于是应用服务器端向 360 服务器端发起的请求，走 https，所以安全性较好。app_secret 存在应用服务器上也不会泄漏。但流程较为复杂。

Code 模式所获取的 access token 有时间限制。如果超时，可通过 refresh token 换新的 token。换取的过程可见 3.2.4 节。

Token 模式 – [Implicit Grant 模式](#)，应用直接获取 access token，不建议使用

- (1). 应用调用 SDK 登录接口 (3.2.1 节)
- (2). SDK 展示服务端的页面，引导用户完成登录、授权
- (3). 服务端返回 access token 给 SDK
- (4). SDK 返回 access token 给应用
- (5). 应用使用 access token 调用 360 服务器接口获得用户信息 (3.2.3 节)

Token 模式省去了 authorization code 换取 token 的步骤，不需要应用服务器介入，客户端直接获取 token，因此流程简单。但安全性较差。同时，所获取的 token 是一次性的，不能刷新。一旦超时，

就无法再调用 360 开放平台的其他 OAuth 接口, 应用就只能引导用户重新登录。如果想解决这个问题, 只有换到 CODE 模式。

无论是 code 模式还是 token 模式, 都需要用户先输入 360 用户账号密码。SDK 会唤起网页版的登录页面。如果用户已经登录, 会显示快速登录页面。用户可以用保存的账号直接进入游戏, 或者切换到其他账号。如果用户没有 360 账号, 点击 立即注册 按钮就可以注册新账号。



图 1 登录页

图 2 注册页

3.2 接口介绍

3.2.1 登录- SDK 接口, 应用客户端调用

应用调用 SDK 登录接口, SDK 显示登录页面, 引导用户进行登录与授权。最终, code 模式会返回 authorization code, token 模式会直接返回 access token。

推荐应用接入 code 模式, 因为安全性高。不推荐使用 token 模式, 不仅安全性低, 而且 token 有效期过短, 不能刷新, 会导致 360 的其他 OAuth 接口无法使用, 用户必须频繁重新登录。应用可能需要重新换接 code 模式才能解决问题。具体参见 3.1 节的说明

调用登录界面代码示例 (DemoMain.java 代码片段):

```
Intent intent = new Intent(this, ContainerActivity.class);
intent.putExtra(ProtocolKeys.FUNCTION_CODE, ProtocolConfigs.FUNC_CODE_LOGIN);
intent.putExtra(ProtocolKeys.CLIENT_ID, Matrix.getAppkey());
intent.putExtra(ProtocolKeys.RESPONSE_TYPE, "code");// code模式, 更安全
startActivityForResult(intent, ProtocolConfigs.RESULT_CODE_LOGIN);
```

兼容 code 登录模式和 token 登陆模式的返回结果 (DemoLogin.java 代码片段):

```

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (data == null) {
        return;
    }
    switch (requestCode) {
        case ProtocolConfigs.RESULT_CODE_LOGIN:
            String result = data.getStringExtra(ProtocolKeys.LOGIN_RESULT_BACK);
            JSONObject json;
            try {
                json = new JSONObject(result);
                if (json.getBoolean("isCode")) {
                    // code 登陆模式
                    String authorizationCode = json.getString("code");
                    showLoading(this, "获取用户信息", "正在处理数据,请稍后.....");
                    // 通过游戏自身服务器申请access token
                    getAccessToken(authorizationCode);
                } else {
                    // token 登陆模式, 360 服务器直接返回access token
                    String token = json.getString("token");
                    showLoading(this, "获取用户信息", "正在处理数据,请稍后.....");
                    getLoginInfo(token);
                }
            } catch (JSONException e) {
                e.printStackTrace();
            }
            break;
    }
}

/**
 * 用mAuthorizationCode换取mAccessToken
 * 调用应用服务器接口来换取TOKEN
 * @param authorizationCode
 */
private void getAccessToken(String authorizationCode) {
    // 应用服务器端用code来换取token, 应用服务器调用360开放平台access_token接口
    mHttp.post("http://demo.server.app.com/get_token_by_code?code="
        + authorizationCode, null, new HttpCallback() {
        @Override

```

```
public void onload(JSONObject json) {
    if (json != null) {
        try {
            String status = json.getString("status");
            JSONObject data = json.getJSONObject("data");
            if (status.equals("ok")) {
                String accessToken = data.getString("access_token");
                getLoginInfo(accessToken);
            } else {
                Toast.makeText(getApplicationContext(), data.getString("msg"),
                    Toast.LENGTH_LONG).show();
            }
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
});
}

/**
 * 通过mAccessToken获取用户信息
 * 调用应用服务端来获取userinfo
 * @param accessToken
 */
private void getLoginInfo(final String accessToken) {
    if (!TextUtils.isEmpty(accessToken)) {
        mHttp.post(
            "http://demo.server.app.com/get_userinfo_by_token?" +
                accessToken, null, new HttpCallback() {
                @Override
                public void onload(JSONObject json) {
                    if (json != null) {
                        try {
                            String status = json.getString("status");
                            if (status.equals("ok")) {
                                json = json.getJSONObject("data");
                                String userId = json.getString("id");
                                Intent intent = new Intent(DemoLogin.this
                                    .getApplicationContext(), DemoPay.class);
                                intent.putExtra("token", accessToken);
                                intent.putExtra("id", userId);
                                // 自动启动支付页面
                                startActivity(intent);
                            }
                        }
                    }
                }
            });
    }
}
```


返回参数：

参数	必选	参数说明
access_token	Y	Access Token 值
expires_in	Y	Access Token 的有效期 以秒计
refresh_token	Y	用于刷新 Access Token 的 Token, 有效期 14 天
scope	Y	Access Token 最终的访问范围,即用户实际授予的权限列表 (用户在授权页面时,有可能会取消掉某些请求的权限)

返回示例：

```

HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

{
  "access_token": "120652e586871bb6bbcd1c7b77818fb9c95d92f9e0b735873",
  "expires_in": "3600",
  "scope": "basic",
  "refresh_token": "12065961868762ec8ab911a3089a7ebdf11f8264d5836fd41"
}

```

3.2.3 获取用户信息-服务器端接口, 建议应用服务器调用

应用获取 access token 后, 可调用 360 开放平台服务器端接口/user/me, 获取用户信息. 应用可以在服务器端或客户端来调用这个接口. 但建议从服务器端发起请求, 避免将 access token 存在客户端.

参数说明：

参数	必选	参数说明
access_token	Y	授权的 access token
fields	N	允许应用自定义返回字段, 多个属性之间用英文半角逗号作为分隔符. 传递此参数 id,name,avatar 将不再默认返回

返回参数：

参数	必选	参数说明
id	Y	360 用户 ID
name	Y	360 用户名
avatar	Y	360 用户头像
sex	N	360 用户性别, 仅在 fields 中包含时候才返回, 返回值为 :男, 女或者未知
area	N	360 用户地区, 仅在 fields 中包含时候才返回

请求示例：

```
https://openapi.360.cn/user/me.json?access_token=12345678983b38aabcdef387453ac8133ac3263987654321&fields=id,name,avatar,sex,area
```

返回示例：

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store
{
  "id": "201459001",
  "name": "360U201459001",
  "avatar":
    "http://u1.qhimg.com/qhimg/quc/48_48/22/02/55/220255dq9816.3eceac.jpg?f=d140ae40ee93e8b08ed6e9c53543903b",
  "sex": "未知"
  "area": ""
}
```

获取用户信息后，应用需要保存自身账号与 360 账号的绑定关系。并且妥善保存用户信息留待以后使用。例如支付时，就需要使用 360 用户 id

3.2.4 换新 access token—服务器端接口, 应用服务器调用

应用获取的 access token 有时间限制. 如果用户登录时间长于这个时间限制, 就会导致 token 过期, 调用开放平台接口会失败. 此时, 应用应该通过服务器, 用 refresh token 去换新 access token. 同时, refresh token 也会更新.

参数说明：

参数	必选	参数说明
grant_type	Y	定值 refresh_token
refresh_token	Y	用于刷新 access token 用的 refresh token
client_id	Y	app key
client_secret	Y	app secret
scope	Y	定值 basic

请求示例：

```
https://openapi.360.cn/oauth2/access_token?grant_type=refresh_token&refresh_token=12065961868762ec8ab911a3089a7ebdf11f8264d5836fd41&client_id=0fb2676d5007f123756d1c1b4b5968bc&client_secret=8d9e3305c1ab18384f56.....&scope=basic
```

返回参数：

参数同 3.2.2 节获取 access token 时的返回, 只是 access token 和 refresh token 都换了新的.

返回示例：

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store
{
  "access_token": "120652e586871bb6bbcd1c7b77818fb9c95d92f9e0b735873",
  "expires_in": "3600",
  "scope": "basic",
  "refresh_token": "12065961868762ec8ab911a3089a7ebdf11f8264d5836fd41 "
}
```

4. 支付流程

4.1 流程介绍

1. 应用调用 SDK 支付接口(4.2.1 节)
2. SDK 展示支付页面，引导用户完成支付流程
 - a. 若调用接口时指定金额，则显示固定金额支付界面，如图 3 所示
 - b. 若调用接口时不指定金额，则显示充值界面，如图 4 所示
3. 支付成功后，360 服务器回调应用服务器上的通知接口(4.2.2 节)，通知支付结果
4. (可选) 应用服务器调用 360 服务器端订单确认接口(4.2.3 节)，服务端返回确认结果
5. 应用返回 ok 给 360 服务器端，并为用户充值



图 3.固定金额支付界面



图 4. 不定金额充值界面

4.2 接口介绍

4.2.1 支付接口- SDK 接口, 应用客户端调用

应用调用 SDK 支付接口时, SDK 弹出支付选择界面. 用户在界面上完成支付.

参数说明：

QiHooPayInfo 类的参数列表如下表, 代码示例后附

参数	必选	参数说明	数据格式
product name	Y	所购买商品名称, 应用指定. 建议中文	最大 10 中文字
amount	Y	所购买商品金额, 以分为单位. 如果是充值 可不指定	整数
app key	Y	应用 app key (加入签名)	
privatekey	Y	值为 md5(app_secret + “#” + app_key)全小写 用于签名的密钥 不能把 app_secret 写到应用客户端程序里 因此使用这样一个特殊的 KEY 应算出值直接写在 app 中, 而不是写 md5 的计算	
notifyuri	Y	支付结果通知 uri	最大 255 字符
productid	Y	购买商品的商品 id, 应用指定	最大 16 字符
appusername	Y	应用内的用户名 如游戏角色名 若应用内绑定 360 账号和应用账号, 充值不分区服, 充到统一的用户账户, 各区服角色均可使用, 则可用 360 用户名	最大 16 中文字

appuser id	Y	应用内的用户 id 若应用内绑定 360 账号和应用账号, 充值不分区服, 充到统一的用户账户, 各区服角色均可使用, 则可用 360 用户 ID	最大 32 字符
ext1	N	应用扩展信息 1, 原样返回.	最大 255 字符
ext2	N	应用扩展信息 2, 原样返回	最大 255 字符
qihoo user id	Y	360 账号 id	整数
accesstoken	N	用户 access token, 要使用注意过期和刷新问题	最大 32 字符
apporderid	N	应用订单号, 应用内必须唯一	最大 32 字符

注意所有参数字符集都是 UTF-8.使用中文时, 一定要注意字符集问题.

关于应用方订单号的问题：**如果指定了应用方订单号 app_order_id, 则限定应用订单号不能重复提交, 并且, 限定一个应用订单, 不管是否支付成功, 都只能支付一次.** 这样做是为了避免重复支付. 通知应用方加钱时, 会返回应用订单号, 同时, 提供 360 订单号.

Access token 非必选, 接口使用可以大大提高支付安全性, 但要注意 token 的时间期限. 过期后调用支付会失败. 游戏有两种办法可以解决. 一是引导用户重新登录, 二是从服务器端, 调用 refresh token 接口. 具体见 2.4.2 节.

使用说明：

在点击支付按钮加入下面类似代码，参数可以由开发者自己定义（DemoMain.java 代码片段）：

```
Intent intent = new Intent(this, ContainerActivity.class);
intent.putExtra(ProtocolKeys.FUNCTION_CODE, ProtocolConfigs.FUNC_CODE_PAY_CHONGZHI);
intent.putExtra(ProtocolKeys.PRODUCT_NAME, "游戏银两"); // 所购商品名称, 应用指定
intent.putExtra(ProtocolKeys.AMOUNT, "0"); // 不指定金额为充值操作指定金额为购买操作
intent.putExtra(ProtocolKeys.PRODUCT_ID, productId); // 所购商品ID, 应用指定
intent.putExtra(ProtocolKeys.APP_USER_NAME, "用户应用内名称");
intent.putExtra(ProtocolKeys.APP_NAME, "游戏名称");
intent.putExtra(ProtocolKeys.APP_USER_ID, "1888"); // 应用内用户ID
intent.putExtra(ProtocolKeys.NOTIFY_URI, notifyUri);
intent.putExtra(ProtocolKeys.QIHOO_USER_ID, mUserId);
intent.putExtra(ProtocolKeys.APP_KEY, Matrix.getAppkey());
intent.putExtra(ProtocolKeys.PRIVATE_KEY, Matrix.getPrivatekey());
intent.putExtra(ProtocolKeys.RATE, "2"); // 充值兑换比例1元可兑换多少游戏币
intent.putExtra(ProtocolKeys.APP_EXT_1, "可以自行指定1"); // 应用自行指定
intent.putExtra(ProtocolKeys.APP_EXT_2, "可以自行指定2"); // 应用自行制定
intent.putExtra(ProtocolKeys.ACCESS_TOKEN, mAccessToken); // TOKEN可能过期, 服务器端需要做更新
intent.putExtra(ProtocolKeys.APP_ORDER_ID, appOrderId); // 应用订单id
startActivity(intent);
```

4.2.2 支付结果通知接口-应用服务器提供接口, 由 360 服务器回调

应用调用支付接口时, 需指定支付结果的通知回调地址 notify_uri. 支付完成后, 360 服务器会把支付结果以 GET 方式通知到此地址 (建议应用同时支持 GET 和 POST). 应用接收验证参数后, 给用户做游戏内充值.

应用在接收到通知消息后, 需回应 ok, 表示通知已经接收. 如果回应其他值或者不回应, 则被认为通知失败, 360 会尝试多次通知. 这个机制用来避免掉单.

应用应做好接收到多次通知的准备, 防止多次加钱. 同时, 需要特别注意的是, **回应的 ok 表示应用已经正常接到消息, 无需继续发送通知.** 它不表示订单成功与否, 或者应用处理成功与否. 对于重复的通知, 应用可能发现订单已经成功处理完毕, 无需继续处理, 也要返回 ok. 否则, 360 会认为未成功通知, 会继续发送通知.

手机支付的参数如下:

参数	必选	参数说明	是否参与签名
app_key	Y	应用 app key	Y
product_id	Y	所购商品 id	Y
amount	Y	总价,以分为单位	Y
app_uid	Y	应用内用户 id	Y
app_ext1	N	应用扩展信息 1 原样返回	Y
app_ext2	N	应用扩展信息 2 原样返回	Y
user_id	N	360 账号 id	Y
order_id	Y	360 返回的支付订单号	Y
gateway_flag	N	如果支付返回成功, 返回 success 应用需要确认是 success 才给用户加钱	Y
sign_type	Y	定值 md5	Y
app_order_id	N	应用订单号 支付请求时若传递就原样返回	Y
sign_return	Y	应用回传给订单核实接口的参数 不加入签名校验计算	N
sign	Y	签名	N

应用接收到支付平台回调的请求, 参见附录的签名算法对参数进行签名, 然后和平台传递的签名 sign 比较, 从而校验平台请求的合法性.

通知消息样例:

```
order_id=1211090012345678901&app_key=1234567890abcdefghijklmnopqrstuv&product_id=p1&amount=101&app_uid=123456789&app_ext1=XXX201211091985&user_id=987654321&sign_type=md5&gateway_flag=success&sign=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&sign_return=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

样例的签名字段排列 (列出来仅供参考, 请根据实际参数情况用程序排序产生, 不要写死在程序里)

amount, app_ext1, app_key, app_uid, gateway_flag, order_id, product_id, sign_type, user_id

样例的签名串

```
101#XXX201211091985#1234567890abcdefghijklmnopqrstuv#123456789#success#1211090012345678901#p1#md5#987654321#应用 app_secret
```

4.2.3 订单核实接口- 服务器端接口, 应用服务器调用

为了防止伪造的支付成功通知, 应用可以使用本接口做通知数据的校验. 把应用加钱接口(4.2.2 节)收到的通知消息里的参数, 加上 app_key 和 app_secret, 不需要做签名, 直接调用接口, 即可校验数据是否正确.

本接口由 360 服务端提供, 由于需要 app_secret, 必须由应用服务器端调用. 不允许用客户端. 接口支持 POST 和 GET 调用.

接口地址:

https://openapi.360.cn/pay/verify_mobile_notification.json?参数

参数如下:

参数	必选	参数说明
app_key	Y	应用 app key
product_id	Y	所购商品 id
amount	Y	总价,单位: 分
app_uid	Y	应用内用户 id
order_id	Y	360 支付订单号
app_order_id	Y	应用订单号
sign_type	Y	当前仅 md5
sign_return	Y	应用传给订单核实接口的参数 sign_return
client_id	Y	还是应用 app_key
client_secret	Y	应用 app_secret

本表除 client_id 和 client_secret 为应用增加外，其他参数均来自应用加钱接口收到的支付通知消息，原样提供即可。

如果参数提供正确，订单核实接口返回为 json 格式数据。

验证成功返回{"ret":"verified"}

验证不成功返回{"ret":"invalid"}

如果调用参数不全或者有错，则返回的是开放平台接口统一的错误代码。例如：

```
{
  "error_code": "4010102", "error": "oauth_consumer_key 不可用 ( OAuth1.0a ) "
}
```

5.附录：

5.1 签名算法：

签名算法不区分前后端，只要在需要签名的地方，均采用如下的算法

1. 必选参数必须有值，而且参数值必须不为空，不为 0。字符集为 utf-8
2. 所有不为空，不为 0 的参数都需要加入签名，参数必须为做 urlencode 之前的原始数值。如中文 金币，作为参数传输时编码为%E9%87%91%E5%B8%81，做签名时则要用其原始中文值 金币（注意字符集必须是 UTF-8）
3. 对所有不为空的参数按照参数名字母升序排列(如 php 的 ksort 函数)
4. 使用符号#拼装排序后的参数值，最后用#连接应用的 app_secret，整体用 md5 计算签名，就是 sign 参数的值。注意有些语言的 md5 计算结果里字母为大写，需要转化为小写。
5. 拼装 url 进行 WEB 传递，这时参数值要做 urlencode

php 范例如下：

```
// 准备签名参数
$input = array(...);
/* 去掉为空的字段 */
foreach($input as $k=>$v)
{
    if(empty($v)){
        unset($input[$k]);
    }
}
ksort($input); //对参数按照 key 进行排序
$sign_str = implode('#',$input); //第四步
$sign_str = $sign_str.'#'.$sign_key; //拼装密钥（如果是签名，密钥为约定处理后的密钥）
$sign = md5($sign_str);
$input['sign'] = $sign; //得到签名

/* 第五步得到 url 传递即可 */
//这里地址就是一个演示的接口地址
$url = 'http://testapp.com/notify?'.http_build_query($input);
...
```

5.2 Demo 工程简介

0.5 版本的 sdk 提供了新的 Demo 工程，工程结构如图 5 所示：

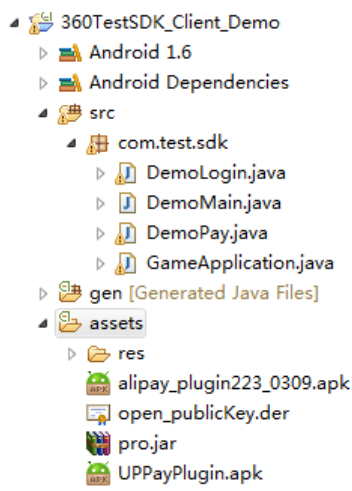


图 5. 支付 SDK 的 Demo 工程

该工程中主要包含 4 个 java 文件：

5.2.1 App 文件 GameApplication.java

请**务必**在 GameApplication 类中调用 Matrix 的初始化函数，请参考 GameApplication.java 中的代码片段：

```
public class GameApplication extends Application {

    private static final String TAG = "GameApplication";

    @Override
    public void onCreate() {
        // SDK初始化，须最先调起
        Matrix.init(this);
    }

    @Override
    public void onTerminate() {
        // 勿忘释放资源，仅供emulated环境
        Matrix.onDestroy();
    }
}
```

5.2.2 主界面文件 DemoMain.java

该文件是 Demo 工程的默认启动界面实现，它包含了“登录”、“定额支付”、“不定额支付”三个独立测试按钮，分别测试 SDK 的三个主要界面。此外，还新增了一个“流程测试”按钮，用来进行“登录->支付”的全流程测试。界面如图 6 所示：



图 6.DemoMain 启动界面

DemoMain 中包含了三个独立测试按钮的代码，开发者可以参考文件中的调用方法。DemoMain 的代码实现都集中在一个文件中，方便参考。唯一的缺点是所有的参数都是硬编码写死的，代码中只能展示“code”模式登录，而无法同时展示“token”登录方式。“流程测试”功能按钮弥补了这一缺憾。

5.2.3 可配置登录界面 DemoLogin.java

DemoLogin 文件提供了“流程测试”中的可配置登录界面，用户可以选择“code”或“token”模式进行登录测试，如图 7 所示：



图 7.可配置登录界面

登录可配参数说明：

可配参数	参数说明
client id	应用的 app key
response type	登录方式选择，可选“code”或“token”

5.2.4 可配置支付界面 DemoPay.java

DemoPay 文件提供了“流程测试”中的可配置支付界面，用户可以指定支付金额、兑换比例等多个参数进行测试，如图 8 所示：

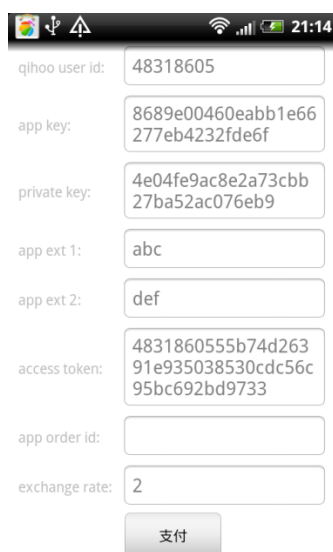


图 8.可配置支付界面

支付可配参数说明：

可配参数	参数说明
product name	出售的产品名称，例如道具名
money amount	支付金额，如果是非 0 值则启动固定支付界面；如果是 0，则启动充值界面
app user name	用户在游戏内的用户名
app name	游戏名称
app user id	游戏中的用户 id
qihoo user id	奇虎用户 id 号
product id	出售的产品 id
notify uri	支付结果的通知回调地址
app key	游戏申请的唯一 key 值
private key	通过 app key 和 secret key 计算出的 md5 值
app ext1	应用需要发送给 sdk 的 extra 信息 1,360 服务器会在支付时把这个字段发送个游戏服务器
app ext2	应用需要发送给 sdk 的 extra 信息 2,360 服务器会在支付时把这个字段发送个游戏服务器

access token	登录成功后获取的访问 token
app order id	订单号，游戏如果没有生成订单号，可不填
exchange rate	人民币与游戏货币兑换比例，例如，2 代表 1 元人民币兑换 2 游戏币。可不填。