易接单机SDK中间件接入使用说明

上海雪鲤鱼计算机科技有限公司

版本	修改日期	作者	修改内容
1.0.0	2014/6/30	田红	初始版本
1.0.1	2014/7/29	李跃年	优化
1.0.2	2014/12/01	刘莹	优化
1.0.3	2015/7/23	田红、吕雪勤 万少锋	增加 LUA 接口； 增加初始化回调的函数； 增加扩展接口； 修改 AndroidManifest.xml 文件的内容； 增加避免模糊的内容
1.0.4	2016/3/07	田红、吕雪勤	增加 U3D 生命周期接口

目录

1 引言

1.1 术语及缩略词

计费点：游戏关于计费的信息，包含：计费点名称、价格、及唯一标识代码等信息。

appId：游戏的唯一标识，用于区分不同游戏的唯一标准。在易接开发者中心游戏管理模块中创建新游戏获取。

付费状态：触发付费行为后的付费结果，状态包含成功、失败、取消、超出限制等。

短代：短信代计费方式。

1.2 适用对象及范围

适用于策划人员、系统设计人员、开发工程师和测试工程师。

2 SDK 概述

2.1 总体描述

易接 Android 单机中间件 SDK 为游戏开发者提供统一调用接口。

易接单机中间件 SDK 接入总体流程图：

易接单机 SDK 中间件接入标准流程

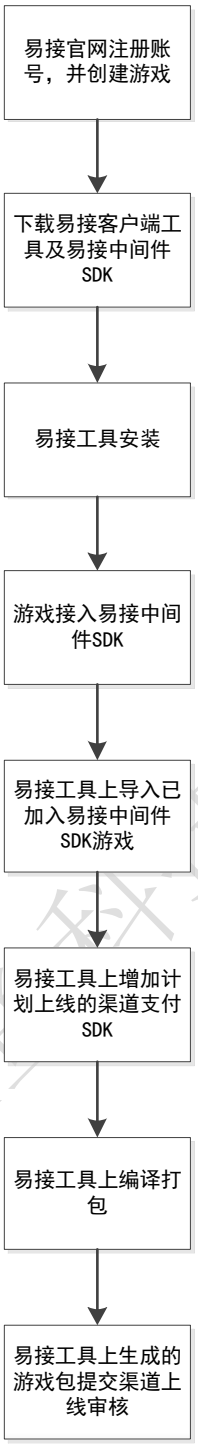


图 2-1

2.2 Java SDK 使用

2.2.1 使用前准备

SDK 开发包包括以下几个文件与目录：

SDK 开发资源包： SDK 目录中包含 SDK 的资源文件，请复制 SDK 目录中的所有目录与文件，并拷贝至各自的游戏工程中，如下：

2.2.1.1 拷贝资源

将“单机\sdk\java\assets”目录下的文件拷贝到游戏对应目录。

2.2.1.2 添加 SDK 提供的 lib 引用

将“单机\sdk\java\libs”目录下的文件拷贝到游戏对应目录下，并做如下关联。

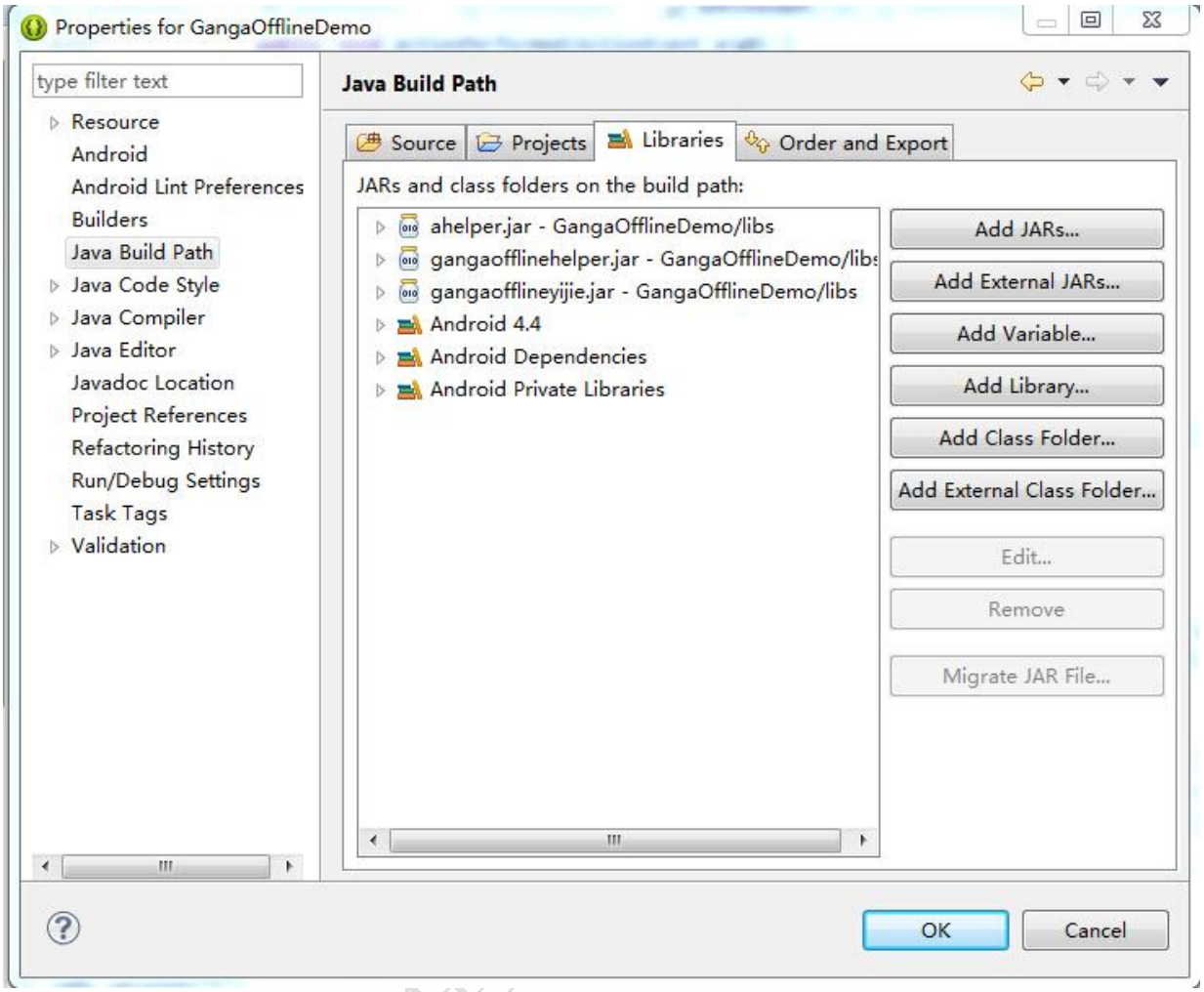


图 2-2

2.2.1.3 修改 Application

1）若无自定义Applicatiion，则修改AndroidManifest.xml的Application如下：

```
<application
android:name="com.snowfish.cn.ganga.offline.helper.SFOfflineApplication
"/>
```

2）若开发者自定义的Application。则自定义Application需要继承*com.snowfish.cn.ganga.offline.helper.SFOfflineApplication*，AndroidManifest.xml修改如下:

```
<application android:name="自定义Application"/>
```

2.2.1.4 添加 SDK 提供的 Activity 引用

将游戏 AndroidManifest.xml 文件的 application 中添加由易接提供如下代码，可参考“单机\sdk\ AndroidManifest.xml "

```
<activity
    android:name="com.snowfish.cn.ganga.offline.helper.SFGameSplashActivity
"
    android:theme="@android:style/Theme.Translucent"
    android:screenOrientation="sensor"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>
<service
    android:name="com.snowfish.a.a.s.ABGSvc"
    android:enabled="true"
    android:process="com.snowfish.a.a.bg">
  <intent-filter>
    <action android:name="com.snowfish.a.a.s.ABGSvc"/>
    <category android:name="android.intent.category.DEFAULT"/>
  </intent-filter>
</service>

<meta-data
    android:name="com.snowfish.appid"
    android:value="KEY">
</meta-data>
<meta-data
    android:name="com.snowfish.customer"
    android:value="SNOWFISH">
</meta-data>
    <meta-data
    android:name="com.snowfish.channel"
    android:value="SNOWFISH">
</meta-data>
    <meta-data
    android:name="com.snowfish.sdk.version"
    android:value="1">
```

```
</meta-data>
```

标红处的 KEY 值需要替换成易接 CP 后台分配的 APPID，

格式如下：{12345678-12345678}。

2.2.1.5 修改启动 Activitiy

在 AndroidManifest.xml 中的主入口 Activity 配置为 *com.snowfish.cn.ganga.offline.helper.SFGameSplashActivity*。

游戏原来的主入口 Activity 请添加到 res/values/strings.xml 中的 *sf_class_name* 字符串中，使得启动完成后即启动游戏

例如：

```
<string name="sf_class_name">com.yijie.cn.sdk.demo.MainActivity</string>
```

2.2.1.6 添加 SDK 需要的 use-permision

将游戏的 AndroidManifest.xml 中添加 use-permision 如下，

可参考“单机\sdk\ AndroidManifest.xml "

```
<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.INTERNET"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
<uses-permission android:name="android.permission.GET_TASKS" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission
    android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
```

2.2.1.7 避免模糊，防止计费时发生异常

两种方式建议使用第二种方式：

一）proguard-project.txt 文件中增加如下代码：

```
-keep class com.snowfish.**{*;}
-dontwarn  com.unity3d.**
-keep class com.unity3d.**{*; }
```

二）android\sdk\tools\proguard\proguard-android.txt 文件中增加：

```
-keep class com.snowfish.**{*;}
-dontwarn  com.unity3d.**
-keep class com.unity3d.**{*; }
```

2.2.2 游戏接口

2.2.2.1 初始化接口（必选接口）

1、以下 2 个接口任选 1 个接入，在游戏主 Activity 中调用。

注意：只需调用一次

（1）`public static void onInit (Activity activity);`

调用用例:

```
SFCommonSDKInterface.onInit(this);
```

（2）`public static void onInit (Activity activity,SFOfflineInitListener initListener)`

调用用例:

```
SFCommonSDKInterface.onInit(this,new SFOfflineInitListener() {
            @Override
            public void onResponse(String tag, String value) {
            if(tag.equalsIgnoreCase("success")){
               //初始化成功的回调
            }else if(tag.equalsIgnoreCase("fail")){
              //初始化失败的回调，value：如果SDK返回了失败的原因，会给value赋值
            }
            }});
```

2、`public static void onDestroy(Activity activity)`

该方法在游戏 Activity 中的 onDestroy 中调用

调用用例: `SFCommonSDKInterface.onDestroy(this);`

3、`public static void onResume(Activity activity)`

该方法在游戏 Activity 中的 onResume 中调用

调用用例: `SFCommonSDKInterface.onResume(this);`

4、`public static void onPause (Activity activity)`

该方法在游戏 Activity 中的 onPause 中调用

调用用例: `SFCommonSDKInterface.onPause (this);`

2.2.2.2 退出接口（必选接口）

```
public static void onExit(Activity activity,SFGameExitListener sfGameExitLi
stener)
```

当游戏退出前必须调用该方法,进行清理工作。如果游戏直接退出,而不调用该方法,可能会出现未知错误,导致程序崩溃，一般游戏在按返回键退出时调用此接口。

调用用例:

```
SFCommonSDKInterface.onExit(this, new SFGameExitListener () {
        @Override
        public void onGameExit(boolean flag) {
            if (flag) {
                //SDK已经退出，此处可处理游戏的退出代码
                MainActivity.this.finish();
            }
        }
    });
```

2.2.2.2 计费接口（必选接口）

```
public static void pay(Activity activity, String id,SFIPayResultListener pay
Listener);
```

该接口用于用户触发计费进行付费行为的入口函数。合作伙伴可以在需要计费的地方调用此接口可以进行计费。

参数描述:

参数名称	类型	意义	注释
activity	Activity	PayActivity	付费页面
id	String	PaymentId	计费点索引,格式如“0”，“1”，“2”
payListener	SFIPayResultListener	Callback	计费结果的回调逻辑。onSuccess:计费成功的回调处理，onFailed：计费失败后的回调处理，onCanceled：用户不进行计费的回调处理。

调用用例：

```
SFCommonSDKInterface.pay(MainActivity.this, "0", new SFIPayResultListener() {
            @Override
            public void onCanceled(String arg0) {
            }

            @Override
```

```
            public void onFailed(String arg0) {
            }

            @Override
            public void onSuccess(String arg0) {
            }
        });
```

2.2.2.3 是否需要打开游戏声音（必选接口）

```
public static boolean isMusicEnabled(Activity activity)
```

判断 SDK 是否需要打开游戏声音，目前只有移动基地需要此接口

游戏开发者需要根据该返回值，设定游戏背景音乐是否开启。

2.2.2.4 更多游戏接口

```
public static void viewMoreGames (Activity activity)
```

更多游戏接口

2.2.2.5 扩展接口

```
public static String extend(Activity activity,String data,Map<String, Object>
callback)
```

扩展接口，有些 SDK，要求必须接入统计接口或者其它特殊的接口，并且有返回值或者回调的函数，用户可以使用此接口调用，具体可以参考易接工具上的 SDK 的参数填写帮助。

参数说明如下：

```
/**
* @param activity                上下文Activity
* @param data                    需要传入的数据
* @param ,Map<String, Object>    回调函数的HashMap,如果没有客户传null
                                 String:回调函数的索引：
                                 callback1、callback2、callback3....以此类推
*                                Object：SFExtendListener类型：回调函数的定义
```

```
*/
```

调用用例：

```
HashMap callback = new HashMap<String, SFExtendListener>();
        SFExtendListener lis1 = new SFExtendListener () {
            @Override
            public void onResponse(String tag, String value) {

            }
        };
         callback.put("callback1", lis1);
       //  callback.put("callback2", lis2);
        // callback.put("callback3", lis3);
String r = SFCommonSDKInterface.extend ((Activity) mContext, "isOneKey",
 callback);
```

2.2.2.6 计费点开发流程

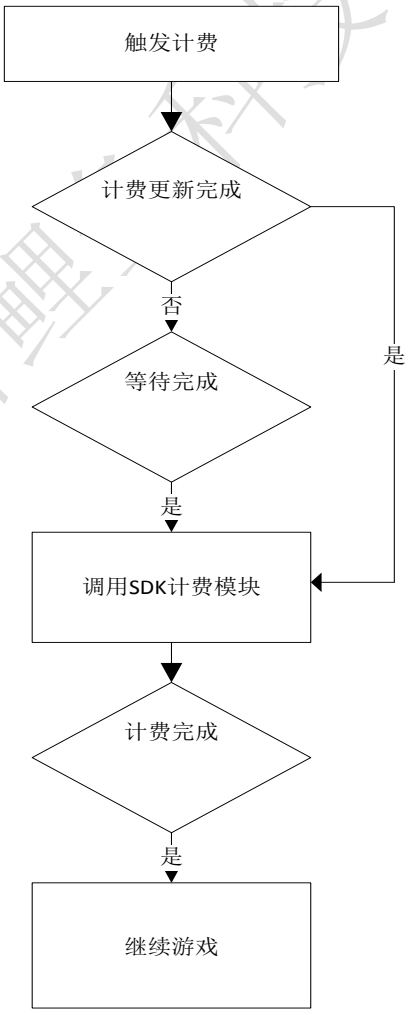


图 2-1 计费流程图

描述：用户触发计费后，对 app 计费更新状态进行判断，若更新完成，则直接进行计费步骤；若未更新则进行等待，等到更新完成后则进行计费步骤。计费完成后用户可继续进行游戏。

2.3 Unity3D 接口

2.3.1 添加 SDK 提供的资源文件

将“单机\sdk\u3d”中的文件 Copy 到 Assets\Plugins\Android\

1. 将游戏 AndroidManifest.xml 文件的 application 中添加由易接提供如下代码，可参考“单机\sdk\ AndroidManifest.xml "

```
<activity
    android:name="com.snowfish.cn.ganga.offline.helper.SFGameSplashActivity
"
    android:theme="@android:style/Theme.Translucent"
    android:screenOrientation="sensor"
    android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
</activity>
<service
    android:name="com.snowfish.a.a.s.ABGSvc"
    android:enabled="true"
    android:process="com.snowfish.a.a.bg" >
  <intent-filter>
    <action android:name="com.snowfish.a.a.s.ABGSvc"/>
    <category android:name="android.intent.category.DEFAULT"/>
  </intent-filter>
</service>

<meta-data
    android:name="com.snowfish.appid"
```

```
    android:value="KEY">
</meta-data>
<meta-data
    android:name="com.snowfish.customer"
    android:value="SNOWFISH">
</meta-data>
    <meta-data
    android:name="com.snowfish.channel"
    android:value="SNOWFISH">
</meta-data>
    <meta-data
    android:name="com.snowfish.sdk.version"
    android:value="1">
</meta-data>
```

标红处的 KEY 值需要替换成易接 CP 后台分配的 APPID，

格式如下：{12345678-12345678}。

2. 添加 SDK 需要的 use-permision

将游戏的 AndroidManifest.xml 中添加 use-permision 如下，

可参考“单机\sdk\ AndroidManifest.xml "

```
<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.INTERNET"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
<uses-permission android:name="android.permission.GET_TASKS" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission
    android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
```

3. 修改 Application

1）若无自定义Applicatiion，则修改AndroidManifest.xml的Application如下：

```
<application
android:name="com.snowfish.cn.ganga.offline.helper.SFOfflineApplication
"/>
```

2）若开发者自定义的Application。则自定义Application需要继承 *com.snowfish.cn.ganga.offline.helper.SFOfflineApplication*，
AndroidManifest.xml修改如下:

```
<application android:name="自定义Application"/>
```

2.3.2 生命周期接口（必选接口）

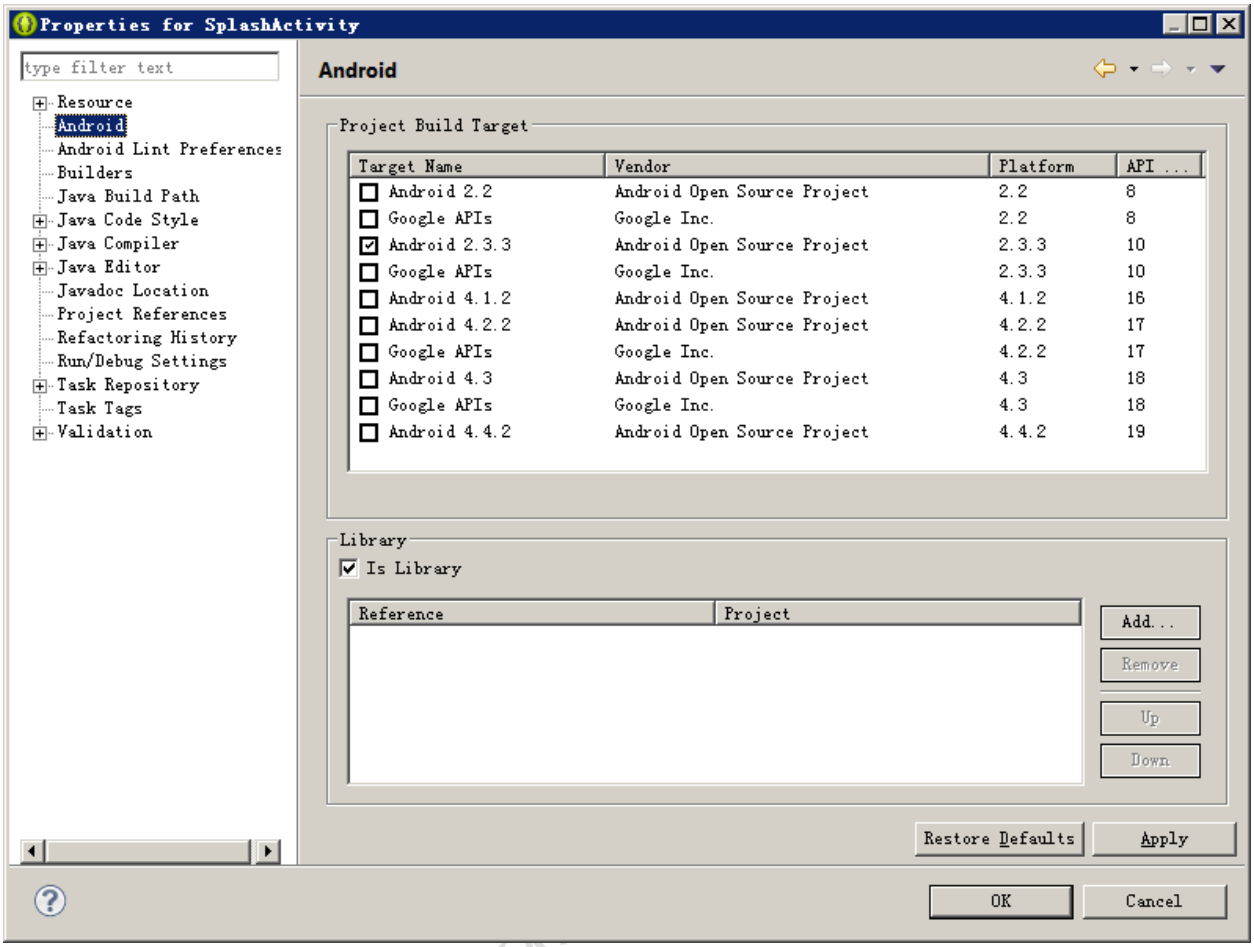
有以下 2 种方式接入

2.3.2.1 游戏入口 Activity 中接入

2.3.2.1.1 接入方法

（1） Eclipse 中建立一个与 unity3d 工程同包名工程，引入 gangaofflinehelper.jar 库，新建一个游戏入口 activity 例如 MainActivity 继承 Unity3D 的 UnityPlayerActivity 或者 UnityPlayerProxyActivity 或者 UnityPlayerNativeActivity（根据游戏的实际情况），可参考 Demo 中的代码，路径如下：单机\demo\u3d\game\Unity3DMainActivity。

（2） 将工程 properties 的 android 选项中 Is Library 选中，如下图



（3） 将工程 bin 目录下生成的 mainactivity.jar 拷贝至 unity 工程 Assets\Plugins\Android\bin 文件夹下；

（4） 修改 AndroidManifest.xml 文件

修改启动 Activity， 在 Assets\Plugins\Android\AndroidManifest.xml 中的主入口 Activity 配置为

com.snowfish.cn.ganga.offline.helper.SFGameSplashActivity。

游戏原来的入口 jar 中定义的 MainActivity 请添加到 res/values/strings.xml 中的 *sf_class_name* 字符串中，使得启动完成后即启动游戏

```
<string name="sf_class_name">com.yijie.cn.sdk.demo.MainActivity</string>
```

在 Assets\Plugins\Android\AndroidManifest.xml 中声明游戏的入口 MainActivity，如下：

```
<activity
      android:name=".MainActivity"
      android:configChanges="orientation|navigation|screenSize
                             |keyboard|keyboardHidden"
      android:label="@string/app_name"
      android:screenOrientation="portrait"
      android:theme="@android:style/Theme.NoTitleBar.Fullscreen" >
      <meta-data
          android:name="unityplayer.UnityActivity"
          android:value="true" />
      <meta-data
          android:name="unityplayer.ForwardNativeEventsToDalvik"
          android:value="true" >
      </meta-data>
  </activity>
```

2.3.2.1.2 接口说明

1. 初始化函数

在 mainactivity.jar 中的 MainActivity.java 的 onCreate 函数中调用，以下 2 个接口任选 1 个接入

注意：只需调用一次

（1）`public static void onInit (Activity activity);`

调用用例:

```
SFCommonSDKInterface.onInit(this);
```

（2）`public static void onInit (Activity activity,SFOfflineInitListener initListener)`

调用用例:

```
/* 初始化完成后可以通过以下方法向U3D中发送消息
* UnityPlayer.UnitySendMessage() 参数1表示发送游戏对象的名称，参数2表示对象绑定的脚本接收该消息的方法（在u3d中定义的函数名称），参数3表示本条消息发送的字符串信息
* 例如：UnityPlayer.UnitySendMessage("MainCamera","messgae",str);
* */
```

```
SFCommonSDKInterface.onInit(this,new SFOfflineInitListener() {
            @Override
            public void onResponse(String tag, String value) {
            if(tag.equalsIgnoreCase("success")){
               //初始化成功的回调
            }else if(tag.equalsIgnoreCase("fail")){
              //初始化失败的回调，value：如果SDK返回了失败的原因，会给value赋值
            }
            }});
```

2. `public static void onDestroy(Activity activity)`

   该方法在游戏 Activity 中的 onDestroy 中调用

   调用用例: `SFCommonSDKInterface.onDestroy(this);`

3. `public static void onResume(Activity activity)`

   该方法在游戏 Activity 中的 onResume 中调用

   调用用例: `SFCommonSDKInterface.onResume(this);`

4. `public static void onPause (Activity activity)`

   该方法在游戏 Activity 中的 onPause 中调用

   调用用例: `SFCommonSDKInterface.onPause (this);`

2.3.2.2 U3D 中接入

1. 初始化接口

   以下 2 个接口任选 1 个接入

   调用该接口完成 SDK 计费流程需要的信息。该接口需在游戏一启动时调用。

注意：只需调用一次

（1） 初始化无回调函数

```
[DllImport ("sfunityoffline")]
```

```
private static extern void onInit(IntPtr context);
```

参数：currentActivity

调用示例：

```
onInit (curActivity.GetRawObject());
```

（2） 初始化有回调完成函数。

```
[DllImport ("sfunityoffline")]
private static extern void onInit_listener(IntPtr context, string
gameObject, string listener);
```

参数名称	类型	注释
context	IntPtr	上下文 Activity
gameObject	string	游戏场景中的对象
listener	string	初始化回调函数，隶属于 gameObject 对象的运行时脚本的方法名称，该方法会在收到通知后触发

调用示例：

```
onInit_listener(curActivity.GetRawObject(),"Main Camera","initResult");
```

```
2. [DllImport ("sfunityoffline")]
private static extern void onPause(IntPtr context);
```

参数：currentActivity

调用示例：

```
onPause (curActivity.GetRawObject())
```

```
3. [DllImport ("sfunityoffline")]
private static extern void onResume (IntPtr context);
```

参数：currentActivity

调用示例：

```
onResume(curActivity.GetRawObject());
```

```
4. [DllImport ("sfunityoffline")]
private static extern void onDestroy (IntPtr context);
```

参数：currentActivity

调用示例：

```
onDestroy (curActivity.GetRawObject());
```

2.3.3 在游戏启动后，判断用户是否开启音效接口（必选接口）

```
[DllImport ("sfunityoffline")]
```

```
private static extern int isMusicEnabled(IntPtr context);
```

判断 SDK 是否需要打开游戏声音，目前只有移动基地需要此接口

参数名称	类型	注释
context	IntPtr	上下文 Activity

返回值： 0 关闭， 1 开启

游戏开发者需要根据该返回值，设定游戏背景音乐是否开启。

调用示例见 SDK 提供的 Demo 脚本 APaymentHelperDemo。

2.3.4 在游戏需要弹出计费界面的场景调用计费接口（必选接口）

```
[DllImport ("sfunityoffline")]
private static extern void pay (IntPtr context, string id, string gameObject,
string runtimeScriptMethod);
```

该接口用于定额支付的接口函数。合作伙伴在需要使用定额支付的时候使用该接口

参数名称	类型	注释
context	IntPtr	上下文 Activity
id	string	计费点索引
gameObject	string	游戏场景中的对象，SDK 内部完成计费逻辑后，并把计费结果通过 Unity 内部 API(com.unity3d.player.UnityPlayer.UnitySendMessage(String gameObject,StringruntimeScriptMethod,Stringargs)通知到 Unity，故游戏开发者需要指定一个游戏对象和该对象的运行脚本，用于侦听 SDK 的计费结果。
runtimeScriptMethod	string	支付监听函数，隶属于 gameObject 对象的运行时脚本的方法名称，该方法会在收到通知后触发

支付回调：

名称	值	注释
PayResult. SUCCESS	"0"	支付成功
PayResult. FAILURE	"1"	支付失败
PayResult. CANCELED	"2"	支付取消

调用示例见 SDK 提供的 Demo 脚本 APaymentHelper.cs。

2.3.5 更多游戏接口

```
[DllImport ("sfunityoffline")]
private static extern void viewMoreGames(IntPtr context);
```

参数：currentActivity

调用该接口查看更多游戏。

调用示例见 SDK 提供的 Demo 脚本

2.3.6 扩展接口

```
[DllImport ("sfunityoffline")]
private static extern void extend(IntPtr context，string data，string gameObject，
string listener);
```

扩展接口，有些SDK， 要求必须接入统计接口或者其它特殊的接口，并且有返回值或者回调的函数，用户可以使用此接口调用，具体可以参考易接工具上的SDK的参数填写帮助。

参数说明：

参数名称	类型	注释
context	IntPtr	上下文 Activity
data	String	需要传入的数据
gameObject	string	游戏场景中的对象
listener	string	回调函数，隶属于 gameObject 对象的运行时脚本的方法名称，该方法会在收到通知后触发

调用示例：

```
SFJSONObject temp =new SFJSONObject();
temp.put("callbackcount","2");//扩展接口回调函数的数量，没有可以传“0”
```

```
temp.put("callback1","callback1");
temp.put("callback2","callback2");
string ss = temp.toInlineString();
SFJSONObject temp1 =new SFJSONObject();
temp1.put("callbackmap",ss);
temp1.put("extendcallback","extendCallback");//返回值，没有可以不传
string ll = temp1.toString();
extend (curActivity.GetRawObject(), "data" , "Main Camera", ll);
```

具体调用示例详见 SDK 提供的 Demo 脚本

2.3.7 游戏退出接口（必选接口）

在游戏退出的时候调用

```
[DllImport ("sfunityoffline")]
private static extern void onExitWithUI(IntPtr context, string gameObject,
string runtimeScriptMethod);
```

参数说明：

参数名称	类型	注释
context	IntPtr	上下文 Activity
gameObject	string	游戏场景中的对象
runtimeScriptMethod	string	退出监听函数，隶属于 gameObject 对象的运行时脚本的方法名称，该方法会在收到通知后触发

主要是释放 SDK 的资源。

调用示例见 SDK 提供的 Demo 脚本 APaymentHelperDemo。

2.4 Flash Air 接口

2.4.1 使用前准备

1) 将“单机\sdk\flashAir\assets\"下的文件 copy 到游戏的 assets\中

2) 把 “单机\sdk\flashAir\libs\ ganga.android.ane" 拷贝到游戏项目的 libs 目录下；

3) 在项目属性的 Flex Build Path - Native Extension 下，添加对 ganga.android.ane 的引用；

4) 在项目属性的 Flex Build Packaging - Google Android - Native Extensions 下，在 Package 列为 ganga.android.ane 打上勾；

5) 在游戏项目的 xxx-app.xml 里添加 SDK 提供的 Activity 引用，

可参考“单机\sdk\ AndroidManifest.xml "

```
<service
    android:name="com.snowfish.a.a.s.ABGSvc"
    android:enabled="true"
    android:process="com.snowfish.a.a.bg" >
  <intent-filter>
    <actionandroid:name="com.snowfish.a.a.s.ABGSvc"/>
    <categoryandroid:name="android.intent.category.DEFAULT"/>
  </intent-filter>
</service>

<meta-data
    android:name="com.snowfish.appid"
    android:value="KEY">
</meta-data>
<meta-data
    android:name="com.snowfish.customer"
    android:value="SNOWFISH">
</meta-data>
    <meta-data
    android:name="com.snowfish.channel"
    android:value="SNOWFISH">
</meta-data>
    <meta-data
    android:name="com.snowfish.sdk.version"
    android:value="1">
</meta-data>
```

标红处的 KEY 值需要替换成易接后台分配的 APPID，

格式如下：{12345678-12345678}。

6) 添加应用权限

将游戏的 AndroidManifest.xml 中添加 use-permision 如下，

可参考“单机\sdk\ AndroidManifest.xml "

```
<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.INTERNET"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
<uses-permission android:name="android.permission.GET_TASKS" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission
    android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
```

7） 修改 Application

(1）若无自定义Applicatiion，则修改AndroidManifest.xml的Application如下：

```
<application
android:name="com.snowfish.cn.ganga.offline.helper.SFOfflineApplication
"/>
```

(2）若开发者自定义的Application。则自定义Application需要继承 *com.snowfish.cn.ganga.offline.helper.SFOfflineApplication*，AndroidManifest.xml修改如下:

```
<application android:name="自定义Application"/>
```

8）在游戏项目的 xxx-app.xml 里添加

```
<extensions>
<extensionID>cn.ganga.ane.android</extensionID>
</extensions>
```

2.4.2 在游戏初启动界面，调用接口（必选接口）

以下 2 个接口任选一个接入，注意：只需调用一次

1、 `public function initSDK():void`

该方法用于 SDK 的初始化，调用用例：

```
SFCommonSDKInterface.getInstance().initSDK();
```

```
    SFCommonSDKInterface.getInstance().onResume();
    SFCommonSDKInterface.getInstance().addEventListener(Constants.EVENT_TYP
E_GANGA_CALLBACK,onCallbackEvent);
    NativeApplication.nativeApplication.addEventListener(Event.EXITING,
onDestroy);
    NativeApplication.nativeApplication.addEventListener(Event.ACTIVATE ,
onActivate);
    NativeApplication.nativeApplication.addEventListener(Event.DEACTIVATE ,
onDeActivate );

    if(stage){
            stage.addEventListener(KeyboardEvent.KEY_DOWN, onKey);
    }
```

onCallbackEvent 函数是对登陆、支付的回调的处理，具体用法参考 demo。

2、`public function initSDKExtend():void`

该方法用于 SDK 的初始化，带有初始化完成的回调函数

调用用例：

```
SFCommonSDKInterface.getInstance().initSDKExtend();
SFCommonSDKInterface.getInstance().onResume();
SFCommonSDKInterface.getInstance().addEventListener(Constants.EVENT_TYPE_GA
        NGA_CALLBACK,onCallbackEvent);
NativeApplication.nativeApplication.addEventListener(Event.EXITING,
        onDestroy);
NativeApplication.nativeApplication.addEventListener(Event.ACTIVATE ,
        onActivate);
NativeApplication.nativeApplication.addEventListener(Event.DEACTIVATE ,
        onDeActivate );
if(stage){
    stage.addEventListener(KeyboardEvent.KEY_DOWN, onKey);
    }
```

onCallbackEvent 函数是对初始化、登陆、支付的回调的处理，具体用法参考 demo。

2.4.3 在游戏启动后，判断用户是否开启音效接口

```
SFCommonSDKInterface.getInstance().isMusicEnable();
```

返回值： 0 关闭， 1 开启

判断 SDK 是否需要打开游戏声音，目前只有移动基地需要此接口

游戏开发者需要根据该返回值，设定游戏背景音乐是否开启。

2.4.4 在游戏需要弹出计费界面的场景调用计费接口（必选接口）

```
SFCommonSDKInterface.getInstance().pay(paymentIdString: String);
```

参数： paymentIdString:计费点索引

2.4.5 游戏退出接口（必选接口）

在游戏退出的时候调用

```
SFCommonSDKInterface.getInstance().exitSDK();
```

2.4.6 获取 Meta 信息接口（可选接口）

该接口用于获取 android 中 Meta 信息接口，结果在 onCallbackEvent 中处理
调用用例：

```
SFCommonSDKInterface.getInstance().getMetaData("com.snowfish.appid");
```

2.4.7 扩展接口（可选接口）

```
public function extend(data:String,callbackCount:String):void
```

扩展接口，有些SDK， 要求必须接入统计接口或者其它特殊的接口，并且有返回值或者回调的函数，用户可以使用此接口调用，具体可以参考易接工具上的SDK的参数填写帮助。
参数说明：

```
/** 扩展接口
 * data:需要传入的数据
 * callbackCount:扩展接口回调函数的数量，没有可以传“0”
 */
```

调用用例：

```
SFCommonSDKInterface.getInstance().extend ("isOneKey","3");
```

2.4.8 更多游戏接口

```
SFCommonSDKInterface.getInstance().viewMoreGame();
```

调用该接口查看更多游戏。

调用示例见 SDK 提供的 Demo 脚本，

2.4.9 获取 Meta 信息接口（可选接口）

该接口用于获取 android 中 Meta 信息接口，结果在 onCallbackEvent 中处理

```
SFCommonSDKInterface.getInstance().getMetaData(key: String);
```

2.4.10 回调处理（必选接口）

```
public function onCallbackEvent(event:CallbackEvent):void
```

回调统一在此函数中处理，具体用法参考 demo。

回调的类型如下：

名称	注释
Constants.CALLBACKTYE_InitResponse	初始化结果的回调
Constants.CALLBACKTYE_MusicEnable	用户是否开启音效接口的回调
Constants.CALLBACKTYE_PaySuccess	支付成功的回调
Constants.CALLBACKTYE_PayFailed	支付失败的回调
Constants.CALLBACKTYE_PayCancel	支付取消的回调
Constants.CALLBACKTYPE_Exit	退出的回调
Constants.CALLBACKTYPE_Expand_Return	扩展接口的返回值
Constants.CALLBACKTYPE_Expand_Calback	扩展接口的回调
Constants.CALLBACKTYPE_GetMetaData	获取meta-data的返回值

2.5 C++游戏接口

2.5.1 开发前准备

2.5.1.1 拷贝资源

SDK 开发资源包： SDK 目录中包含 SDK 的资源文件，请复制 SDK 目录中的所有目录与文件，并拷贝至各自的游戏工程中，如下：

将“单机\sdk\cocos\assets”目录下的文件拷贝到游戏对应目录。

将“单机\sdk\cocos\libs”目录下的文件拷贝到游戏对应目录下，并做如下关联。

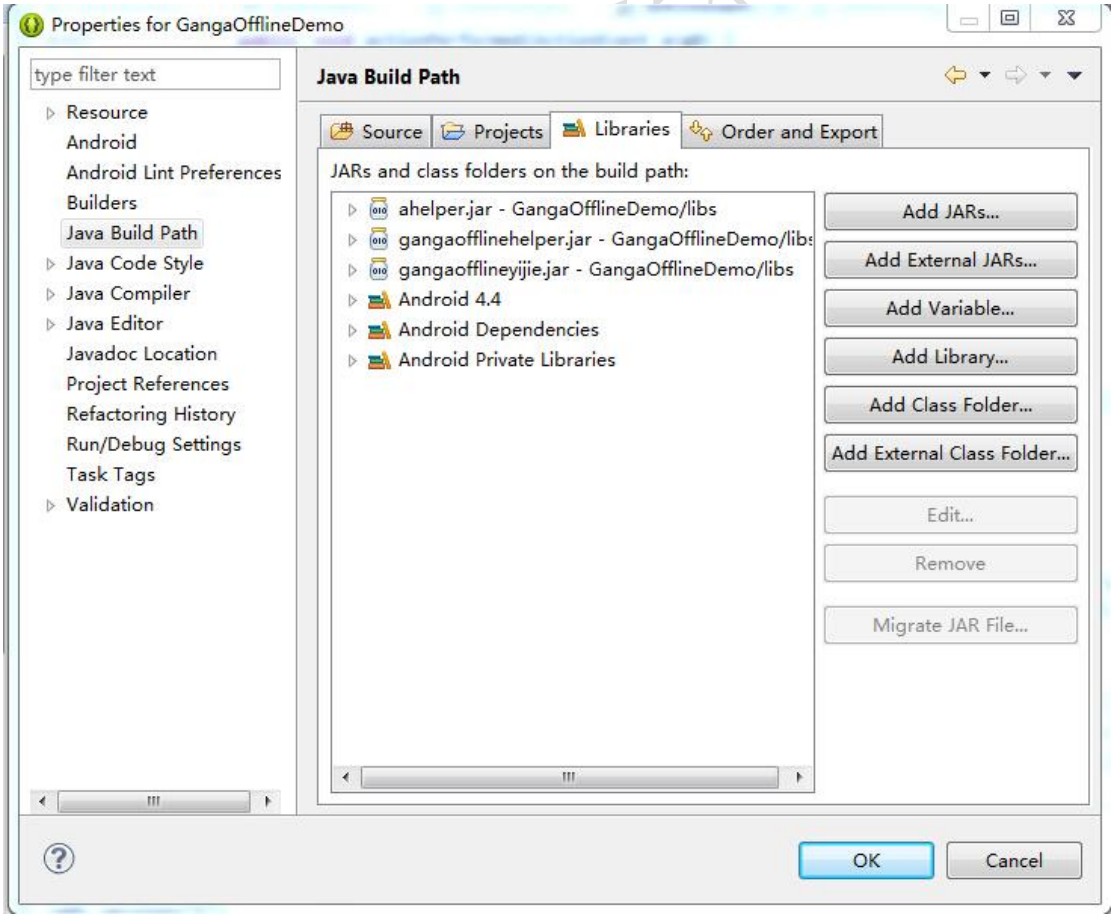


图 2-2

将 Classes/SFGameNativeInterface.h 文件拷贝到游戏工程中 C++头文件

存放位置，如 Cocos2dx 项目放入 Classes 目录。并且在使用接口文件中，加入 include 引用：`#include "SFGameNativeInterface.h"`

2.5.1.2 添加 SDK 提供的 Activity 引用

将游戏 AndroidManifest.xml 文件的 application 中添加由易接提供如下代码，可参考“单机\sdk\ AndroidManifest.xml "

```
<activity
    android:name="com.snowfish.cn.ganga.offline.helper.SFGameSplashActivity
"
    android:theme="@android:style/Theme.Translucent"
            android:screenOrientation="sensor"
    android:label="@string/app_name" >
        <intent-filter>
            <actionandroid:name="android.intent.action.MAIN"/>
            <categoryandroid:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
</activity>
<service
    android:name="com.snowfish.a.a.s.ABGSvc"
    android:enabled="true"
    android:process="com.snowfish.a.a.bg" >
  <intent-filter>
    <actionandroid:name="com.snowfish.a.a.s.ABGSvc"/>
    <categoryandroid:name="android.intent.category.DEFAULT"/>
  </intent-filter>
</service>

<meta-data
    android:name="com.snowfish.appid"

    android:value="KEY">
</meta-data>
<meta-data
    android:name="com.snowfish.customer"
    android:value="SNOWFISH">
</meta-data>
    <meta-data
    android:name="com.snowfish.channel"
```

```
    android:value="SNOWFISH">
</meta-data>
    <meta-data
    android:name="com.snowfish.sdk.version"
    android:value="1">
</meta-data>
```

标红处的 KEY 值需要替换成由易接 CP 后台分配的 APPID，

格式如下：{12345678-12345678}。

2.5.1.3 修改启动 Activitiy

在 AndroidManifest.xml 中的主入口 Activity 配置为 *com.snowfish.cn.ganga.offline.helper.SFGameSplashActivity*。游戏原来的主入口 Activity 请添加到 res/values/strings.xml 中的 *sf_class_name* 字符串中，使得启动完成后即启动游戏

例如：

```
<string name="sf_class_name">com.yijie.cn.sdk.demo.MainActivity</string>
```

2.5.1.4 添加 SDK 需要的 use-permision

将游戏的 AndroidManifest.xml 中添加 use-permision 如下，

可参考“单机\sdk\ AndroidManifest.xml "

```
<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.INTERNET"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
<uses-permission android:name="android.permission.GET_TASKS" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission
```

```
android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
```

2.5.1.5 避免模糊，防止计费时发生异常

两种方式建议使用第二种方式：

一）proguard-project.txt 文件中增加如下代码：

```
-keep class com.snowfish.**{*;}
-dontwarn  com.unity3d.**
-keep class com.unity3d.**{*; }
```

二）android\sdk\tools\proguard\proguard-android.txt 文件中增加：

```
-keep class com.snowfish.**{*;}
-dontwarn  com.unity3d.**
-keep class com.unity3d.**{*; }
```

2.5.1.6 修改 Application

1）若无自定义Applicatiion，则修改AndroidManifest.xml的Application如下：

```
<application
android:name="com.snowfish.cn.ganga.offline.helper.SFOfflineApplication
"/>
```

2）若开发者自定义的Application。则自定义Application需要继承 *com.snowfish.cn.ganga.offline.helper.SFOfflineApplication*，AndroidManifest.xml修改如下:

```
<application android:name="自定义Application"/>
```

2.5.1.7 其它修改

1、 将\sdk\cocos\libs\armabi\libs\armabi\libsfunityoffline.so 拷贝到游戏项目的 libs\armabi 目录和 jni 目录下。

2、 将\sdk\cocos\Classes\SFGameNativeInterface.h 文件拷贝到游戏工程中 C++头文件存放位置，如 Cocos2dx 项目放入 Classes 目录。

- 在调用 C++接口的文件中，加入 include 引用：

```
#include "SFGameNativeInterface.h"
```

3、 修改 android 工程的 jni/Android.mk 文件，添加 libsfunityoffline.so：

```
include $(CLEAR_VARS)
LOCAL_MODULE := sfunityoffline
LOCAL_SRC_FILES := libsfunityoffline.so
include $(PREBUILT_SHARED_LIBRARY)
```

并且添加为游戏 C++模块的依赖共享库：

```
LOCAL_SHARED_LIBRARIES += sfunityoffline
```

4、游戏主 Activity 中加载库文件，如需添加其他库文件，请将此库文件第一个加载:

```
static {
    System.loadLibrary("sfunityoffline");
    }
```

具体参考demo

2.5.2 初始化接口（必选接口）

1、以下2种形式任选1种接入，只需要调用一种，在游戏主Activity的onCreate中调用。

1） 初始化无回调

```
SFCommonSDKInterface.onInit(Activity activity);
public static void init(Activity activity, SFActionCallback callback)
```

调用用例：

```
SFCommonSDKInterface.onInit(this);
SFNativeAdapter.init(AppActivity.this, new SFActionCallback() {
        @Override
        public void callback(Runnable run) {
            runOnGLThread(run);
        }
    });
```

2）初始化有回调

```
public static void init_listener(Activity activity,SFActionCallback callback)
```

调用用例：

```
SFNativeAdapter.init_listener(this, new SFActionCallback() {

            @Override
            public void callback(Runnable run) {
                runOnGLThread(run);
            }
        });
```

2、`public static void onDestroy(Activity activity)`

该方法在游戏 Activity 中的 onDestroy 中调用

调用用例：　`SFCommonSDKInterface.onDestroy(this);`

3、`public static void onResume(Activity activity)`

该方法在游戏 Activity 中的 onResume 中调用

调用用例：　`SFCommonSDKInterface.onResume(this);`

4、`public static void onPause (Activity activity)`

该方法在游戏 Activity 中的 onPause 中调用

调用用例：　`SFCommonSDKInterface.onPause (this);`

2.5.3　在游戏启动后，判断用户是否开启音效接口

```
static bool SFGameNativeInterface::isMusicEnabled();
```

返回值：　false　关闭，　true　开启

判断 SDK 是否需要打开游戏声音，目前只有移动基地需要此接口

游戏开发者需要根据该返回值，设定游戏背景音乐是否开启。

2.5.4　在游戏需要弹出计费界面的场景调用计费接口（必选接口）

```
static void SFGameNativeInterface:: pay(const char* id);
```

参数： id:计费点索引

2.5.5 扩展接口

```
const static char* extend(const char* data, int count);
```

扩展接口，有些SDK， 要求必须接入统计接口或者其它特殊的接口，并且有返回值或者回调的函数，用户可以使用此接口调用，具体可以参考易接工具上的SDK的参数填写帮助。

参数说明：

```
/** 扩展接口
 * data:需要传入的数据
 * count:扩展接口回调函数的数量，没有可以传 0
 */
```

调用用例：

```
SFGameNativeInterface::extend("data",2);
```

2.5.6 初始化回调类

```
class SFNativeInitResulBack {
public:
    SFNativeInitResulBack() {};
    virtual ~SFNativeInitResulBack() {};
    virtual void onResponse(const char* tag,const char* value) = 0;
};
```

2.5.7 退出回调类

```
class SFNativeGameExitCallBack {
public:
    SFNativeGameExitCallBack() {};
    virtual ~SFNativeGameExitCallBack() {};
    //如果b为TRUE，代表游戏退出
    virtual void onGameExit(bool b) = 0;
};
```

2.5.8 支付回调处理类

```
    class SFNativeIPayResulBack {
    public:
        SFNativeIPayResulBack() {};
        virtual ~SFNativeIPayResulBack() {};
```

```
        //取消支付
        virtual void onCanceled(const char* remain) = 0;
        //支付失败
        virtual void onFailed(const char* remain) = 0;
        //支付成功
        virtual void onSuccess(const char* remain) = 0;
    };
```

2.5.9 扩展回调处理类

```
class SFNativeIExtendResulBack {
public:
    SFNativeIExtendResulBack() {};
    virtual ~SFNativeIExtendResulBack() {};
    virtual void onResponse(int index,const char* tag,const char* value) = 0;
};
```

2.5.10 更多游戏接口

```
static void SFGameNativeInterface:: viewMoreGames ();
```

调用该接口查看更多游戏。

2.5.11 游戏退出接口（必选接口）

```
static void SFGameNativeInterface::onExit();
```

调用示例：

```
SFGameNativeInterface::setSFGameExitCallBack(&exitCallback);
SFGameNativeInterface:: onExit ();
```

2.6 Constructs2 JS 游戏接口

2.6.1 Constructs2 android Java 工程准备

1. 将 Constructs2 工程导出 Android 工程。
2. 创建 Crosswalk cordova 工程。

3. 将第 2 步创建的 Crosswalk cordova 工程中的 cordova.js 文件（路径为 assets/www）复制到第 1 步创建的工程的根目录下。

4. 将 Constructs2 导出的 Android 工程复制到前面创建的 Crosswalk cordova 工程的 assets 目录下，移除原来的 www 文件夹，并将 Constructs2 导出的 Android 工程的目录名修改为 www。

经过以上步骤，则将成功创建一个 Android Java app 工程。

2.6.2 易接 Java SDK 使用准备

请参考本文档 2.2.1 节的说明，完成易接 Java SDK 的使用前准备。

2.6.3 游戏接口

以下接口可以参考 yijieconstructs2demo。

2.6.3.1 易接 android Java 部分初始化

找到由 Crosswalk cordova 自动创建的游戏工程的主 Activity，主 Activity 是继承自 org.apache.cordova.CordovaActivity 的 Activity。并在主 Activity 的 onCreate()方法中增加如下代码来初始化易接 SDK。比如，src/com/yijie/ghostshooter/GhotstShooterActivity.java 文件：

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    super.init();

    SFCommonSDKInterface.onInit(this);
    boolean b = SFCommonSDKInterface.isMusicEnabled(this);

    appView.addJavascriptInterface(new SFJsInterface(this, appView),
```

```
"SFJsInterface");

        // Set by <content src="index.html" /> in config.xml
        super.loadUrl(Config.getStartUrl());
        // super.loadUrl("file:///android_asset/www/index.html");
    }

    @Override
    public void onResume() {
        super.onResume();
    }

    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_BACK) {

            SFCommonSDKInterface.onExit(this, new SFGameExitListener() {

                @Override
                public void onGameExit(boolean flag) {
                    if (flag) {
                        System.exit(0);
                    }
                }
            });
            // System.exit(0);
            return true;
        }
        return super.onKeyDown(keyCode, event);
    }
```

2.6.3.2 易接 Java SDK 接口说明

请参考本文档 2.2.2 节，了解易接 Java SDK 接口。

2.6.3.3 添加 JS 接口

需要创建一个连通 JS 和 Java 层的接口，以便于在 Constructs2 的 html 代码中可以执行支付、退出等功能。并需要在游戏工程主 Activity

的 onCreate()方法中，super.loadUrl(Config.*getStartUrl*())执行之前，通过调用 appView.addJavascriptInterface(new SFJsInterface(this, appView), "SFJsInterface")注册该接口。JS 接口的一个示例实现如下：

```
public class SFJsInterface {
    private static final String TAG = "SFJsInterface";

    private Activity mActivity;
    private CordovaWebView mAppView;

    public SFJsInterface(Activity activity, CordovaWebView appView) {
        mActivity = activity;
        mAppView = appView;
    }

    @JavascriptInterface
    public void pay(String billingIndex) {
        Log.i(TAG, "billingIndex = " + billingIndex);
        SFCommonSDKInterface.pay(mActivity, billingIndex, new PayListener());
    }

    @JavascriptInterface
    public void exit() {
        SFCommonSDKInterface.onExit(mActivity, new SFGameExitListener() {

            @Override
            public void onGameExit(boolean flag) {
                if (flag) {
                    System.exit(0);
                }
            }
        });
    }

    class PayListener extends SFIPayResultListener {

        @Override
        public void onCanceled(String remain) {
            mAppView.evaluateJavascript("(function() { return { var1:
\"variable3\", var2: \"variable4\" }; })();",
```

```
                new ValueCallback<String>() {

                    @Override
                    public void onReceiveValue(String value) {
                        Log.i(TAG, "receive in failed, value = " + value);
                    }
                });
        }

        @Override
        public void onFailed(String remain) {
            mAppView.evaluateJavascript("(function() { return { var1:
\"variable3\", var2: \"variable4\" }; })();",
                new ValueCallback<String>() {

                    @Override
                    public void onReceiveValue(String value) {
                        Log.i(TAG, "receive in failed, value = " + value);
                    }
                });
        }

        @Override
        public void onSuccess(String remain) {
            mAppView.post(new Runnable() {

                @Override
                public void run() {
                    mAppView.evaluateJavascript(
                        "(function() { return { var1: \"variable1\", var2:
\"variable2\" }; })();",
                        new ValueCallback<String>() {

                            @Override
                            public void onReceiveValue(String value) {
                                Log.i(TAG, "receive in success, value = " +
value);

                            }
                        });
                }
            });
        }
    }
```

```
}
```

在此 JS 接口中，需要实现支付和退出的逻辑来给 JS 层代码调用。在支付回调方法中，可以借助于 CordovaWebView 的功能来执行 JS 代码，以便于将支付结果传回给 JS 层。

2.6.3.4 在 Cosntructs2 JS 代码中执行支付、退出动作

在 JS 层，可以通过前面注册的 JS 接口，来执行支付、退出等动作。如采用类似下面的方法：

```
<a href="#" onclick="pay0()">Pay billing index 0</a>
<a href="#" onclick="pay1()">Pay billing index 1</a>
<a href="#" onclick="pay2()">Pay billing index 2</a>
<a href="#" onclick="exit()">Exit</a>

<script>
function pay0() {
    SFJsInterface.pay("0");
}
function pay1() {
    SFJsInterface.pay("1");
}
function pay2() {
    SFJsInterface.pay("2");
}
function exit() {
    SFJsInterface.exit();
}
</script>
```

2.7 Cocos Lua 接口

2.7.1 易接 cocos lua 使用准备

2.7.1.1 资源拷贝

1、将 SDK\lua 目录下 frameworks\runtime-src\proj.android 中的 libs 和 yijie 文件夹拷贝到对应目录中。

2、将 SDK\lua 目录下 src\YijieInterface.lua 文件拷贝到对应目录中。

2.7.1.2 修改 Application

1）若无自定义Applicatiion，则修改AndroidManifest.xml的Application如下：

```
<application
android:name="com.snowfish.cn.ganga.offline.helper.SFOfflineApplication
"/>
```

2）若开发者自定义的Application。则自定义Application需要继承*com.snowfish.cn.ganga.offline.helper.SFOfflineApplication*，AndroidManifest.xml修改如下:

```
<application android:name="自定义Application"/>
```

2.7.1.3 添加 SDK 提供的 Activity 引用

将游戏 AndroidManifest.xml 文件的 application 中添加由易接提供如下代码，可参考“单机\sdk\ AndroidManifest.xml "

```
<activity
    android:name="com.snowfish.cn.ganga.offline.helper.SFGameSplashActivity
"
```

```
    android:theme="@android:style/Theme.Translucent"
    android:screenOrientation="sensor"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>
<service
    android:name="com.snowfish.a.a.s.ABGSvc"
    android:enabled="true"
    android:process="com.snowfish.a.a.bg">
  <intent-filter>
    <action android:name="com.snowfish.a.a.s.ABGSvc"/>
    <category android:name="android.intent.category.DEFAULT"/>
  </intent-filter>
</service>

<meta-data
    android:name="com.snowfish.appid"
    android:value="KEY">
</meta-data>
<meta-data
    android:name="com.snowfish.customer"
    android:value="SNOWFISH">
</meta-data>
    <meta-data
    android:name="com.snowfish.channel"
    android:value="SNOWFISH">
</meta-data>
    <meta-data
    android:name="com.snowfish.sdk.version"
    android:value="1">
</meta-data>
```

标红处的 KEY 值需要替换成易接 CP 后台分配的 APPID，

格式如下：{12345678-12345678}。

2.7.1.4 修改启动 Activitiy

在 AndroidManifest.xml 中的主入口 Activity 配置为

com.snowfish.cn.ganga.offline.helper.SFGameSplashActivity。

游戏原来的主入口 Activity 请添加到 res/values/strings.xml 中的 *sf_class_name* 字符串中，使得启动完成后即启动游戏

例如：

```
<string name="sf_class_name">org.cocos2dx.lua.AppActivity</string>
```

2.7.1.5 添加 SDK 需要的 use-permision

将游戏的 AndroidManifest.xml 中添加 use-permision 如下，

可参考“单机\sdk\ AndroidManifest.xml "

```
<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.INTERNET"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
<uses-permission android:name="android.permission.GET_TASKS" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission
    android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
```

2.7.1.6 拷贝易接 SDK 资源

修改 frameworks\runtime-src\proj.android\build-cfg.json 文件增加 "copy_resources":数组中增加

```
{
    "from": "yijie/asssets/Sonnenblume",
    "to": "Sonnenblume"
}
```

修改后如下图。

```
    ],
    "copy_resources": [
        {
            "from": "../../../src",
            "to": "src"
        },
        {
            "from": "../../../res",
            "to": "res"
        },
        {
            "from": "../../cocos2d-x/cocos/scripting/lua-bindings/script"
            "to": ""
        },
        {
            "from": "yijie/asssets/Sonnenblume",
            "to": "Sonnenblume"
        }
    ]
```

2.7.2 接口调用说明

以下接口可以参考易接 lua demo。

2.7.2.1 易接 Android　Java 部分初始化

在游戏 activity 的 oncreate 添加如下代码，初始化 lua 接口:

```
SFLuaAdapter.init(this, new SFActionCallback() {

        @Override
        public void callback(Runnable run) {
            AppActivity.this.runOnGLThread(run);
        }
    });
```

如下图所示：

```
import org.cocos2dx.lib.Cocos2dxActivity;

public class AppActivity extends Cocos2dxActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);


        SFLuaAdapter.init(this, new SFActionCallback() {
            @Override
            public void callback(Runnable run) {
                AppActivity.this.runOnGLThread(run);
            }
        });

    }
}
```

2.7.2.2 实例化易接 SDK 接口

```
--获取易接接口
require "YijieInterface"
local yijie = YijieInterface.new()
```

2.7.2.3 计费接口

function YijieInterface:pay(payid, callbackFunc)

payid 为计费点 id，callbackFunc 为支付回调函数，例如：

```
yijie:pay("1", payCallback)
function payCallback(jPrama)
    require "json"
    local data = json.decode(jPrama);
    if data["result"] == "success" then--支付成功
    elseif data["result"] == "fail" then--支付失败
    elseif data["result"] == "cancel" then--支付取消
    end
```

```
end
```

2.7.2.4 是否打开游戏声音

function YijieInterface:isMusicEnable()

使用示例:

```
local musicEnable = yijie:isMusicEnable()
if musicEnable then --音乐开
else --音乐关
end
```

2.7.2.5 扩展接口

function YijieInterface:extend(data, count, callbackFunc)

data 是需要传入接口的数据，如果参数比较多，可以传 json 数据

count 需要注册多少个回调函数

callbackFunc lua 回调函数

使用示例：

```
yijie:extend (“data”, 3, extendCallback)
function extendCallback (jPrama)
        require "json"
        local data = json.decode(jPrama);
        local index = data["result"]
        local tag = data["tag"]
```

```
        local value = data["value"]
    end
```

2.7.2.6 退出接口

function YijieInterface:onExit(exitCallback)

exitCallback lua 退出回调函数

使用示例：

```
yijie:onExit(exitCallback)
function exitCallback(jPrama)
    require "json"
    local data = json.decode(jPrama);
    if data["result"] == "exit" then--退出游戏
        os.exit()
    elseif data["result"] == "noExit" then--用户取消退出，继续游戏
    end
end
```

2.7.2.7 更多游戏接口

function YijieInterface:viewMoreGames()

调用示例：

```
yijie: viewMoreGames ()
```

3 常见问题解决方案

1、获取配置失败

解决方案：请检查 appId 是否正确、计费点 ID 是否存在，还有网络是否连接正常。

2 如何接入易接单机支付？

操作步骤：

A. 联系易接商务和运营人员确认合作关系，并提交给运营人员游戏计费点信息。

B. 等待运营人员审核通过。

C. 审核通过后，游戏开发人员接入易接 SDK。

D. 使用易接客户端工具，打入易接支付 SDK，完成后提交给易接测试。