91 移动开发平台 SDK(完全版)参考手册

V3.2.6.3

2013-11-04

时间	版本	更新说明
2011-10-18	3.1.2	新建参考手册
2011-11-8	3.1.3	增加代币充值，修改设置为调试模式
2012-01-05	3.1.5	修改接口获取好友头像(自定义大小),AndroidManifest.xml 配置，提交排行榜分数返回状态码描述，代币充值参数
2012-2-10	3.1.6	去掉“导入 nd_res.zip”描述，增加释放平台内存。
2012-3-14	3.1.7	增加接口：查询当前账号是否已经绑定手机号，请求下发绑定手机号的短信验证码，请求绑定手机号，版本检查更新数据接口
2012-3-31	3.1.8	增加接口：当前登录状态，登录(支持游客登录)，游客注册，设置会话(Session)无效监听，以及切换账号相关接口：切换账号登录，设置切换帐号监听通知，设置切换帐号时重新启动，新增使用场景案例
2012-06-12	3.2.0	增加软件互推相关接口
2012-08-09	3.2.1	去掉切换账号功能描述，改为注销并合并到注销功能介绍里面； 增加账号管理功能 增加指定服务器充值说明
2012-08-22	3.2.2	增加进入推广应用列表接口
2012-11-14	3.2.3	增加捕获个人信息修改的通知
2013-1-25	3.2.3.2	调整进入推广应用列表接口
2013-4-23	3.2.5	调整初始化接口，增加暂停页接口，退出页接口
2013-7-12	3.2.6	增加悬浮工具栏接口，增加 AndroidManifest.xml 的 Activity 配置
2013-08-29	3.2.6.1	调整暂停页显示内容
2013-11-04	3.2.6.3	修复注册页部分 BUG，修改 WEB 风格，优先根据代码设置界面方向

目录

一、 91SDK 构成

当您下载完 91SDK 的 rar 包后，解压出该包，里面包含的文件如图：

其中：

- 91Logo　使用 91 手机平台，请在你的 app 的图标上打上 91 平台的标志，资源文件是开发包中的 91Logo 目录。
- SDK 文档及 Demo　91 移动开发平台相关的文档及 DEMO。
- Sdk_lib　包含 91SDK 的 lib 工程。
- readme.txt　91SDK 包的描述文件。

其中 Sdk_lib 文件包括如图：

- 91SDK_LibProject　91SDK 的 lib 工程
- AndroidManifest.xml 配置说明.txt　添加到 AndroidManifest.xml 的配置声明文档

注意：

- 91 移动开发平台资源即 91SDK_LibProject 工程里面的资源（包括 drawable,id,string,color,layout 等）均以“nd_”或“nd3_”为开头的命名，所以希望用户在接入该平台时自己工程的其他资源命名不要以“nd_”和“nd3_”为开头，以避免资源命名冲突。
- 支持 Android SDK1.6 及以上版本。Java JDK1.6 及以上版本。编译时请用 Android SDK4.0 以上版本编译。

二、 SDK 接入流程简要描述

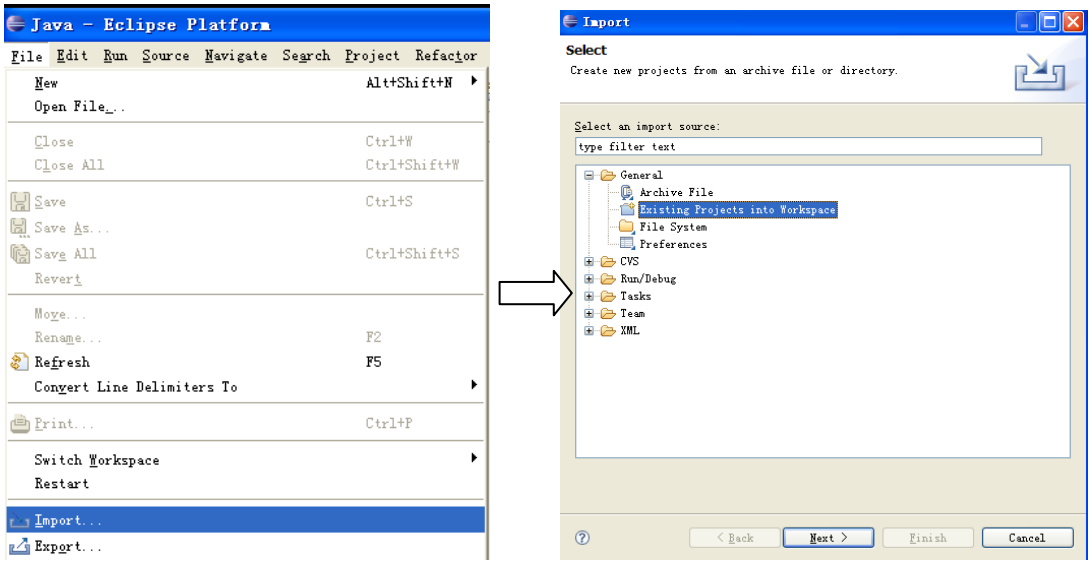
1 搭建 SDK 的环境，导入 SDK 的必要文件，参见 SDK 开发环境搭建
2 初始化 SDK，参见初始化。（初始化 API 中已包含版本更新检测）
3 如果需要的话，设置 SDK 的平台方向，原则要求设置平台方向与游戏方向一致
3 捕捉到初始化完成的通知后，根据需要调用登录，参见登录/注销
4 支付流程请参照应用内购买
5 在游戏主界面需要调用悬浮工具栏接口

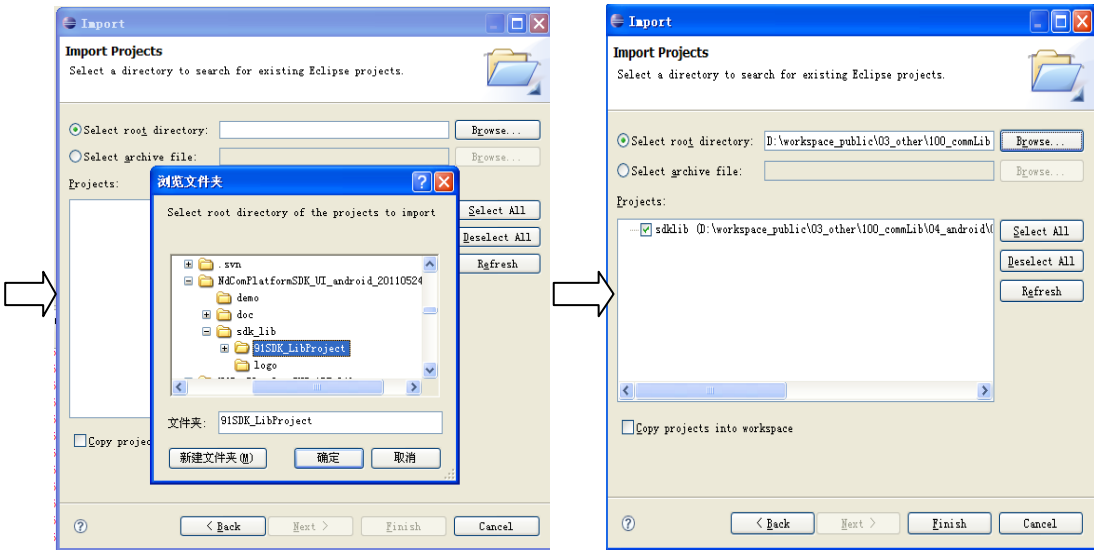
6 在游戏进入暂停或者游戏从后台恢复的时候，需要调用暂停页接口，参见显示暂停页
7 在退出游戏的时候，需要调用退出页接口，参见显示退出页
8 应用中需要有明显入口进入平台中心，详细 API 参见进入平台中心

三、 91SDK 环境搭建

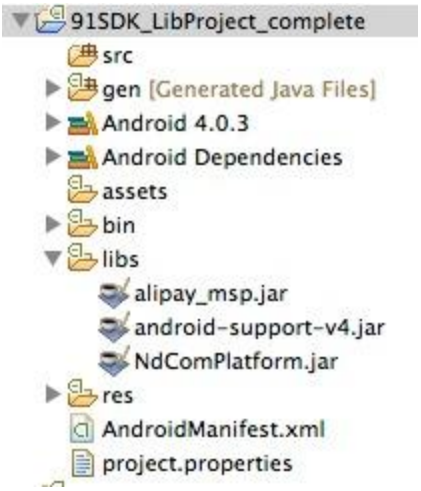
1、 导入 91SDK_LibProject 工程

在您的工程所在的工作空间下，导入 sdk_lib 下的 91SDK_LibProject 工程，如图：

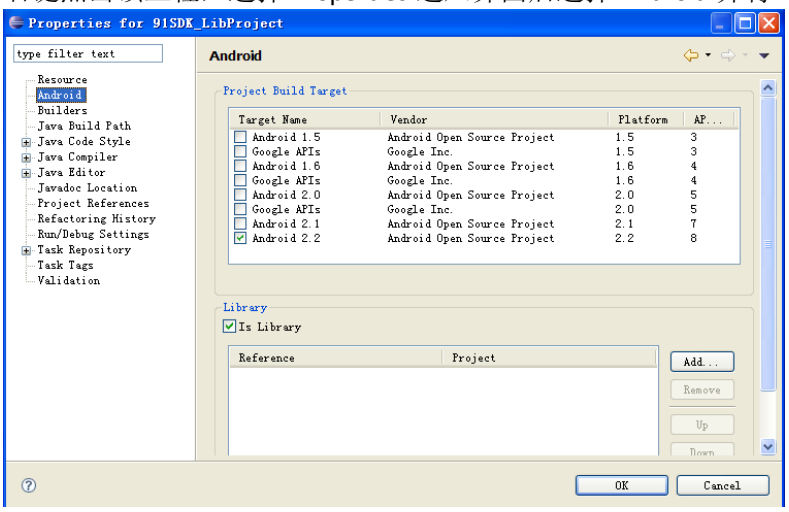




点 Finish 后就把工程引进来了，如图：



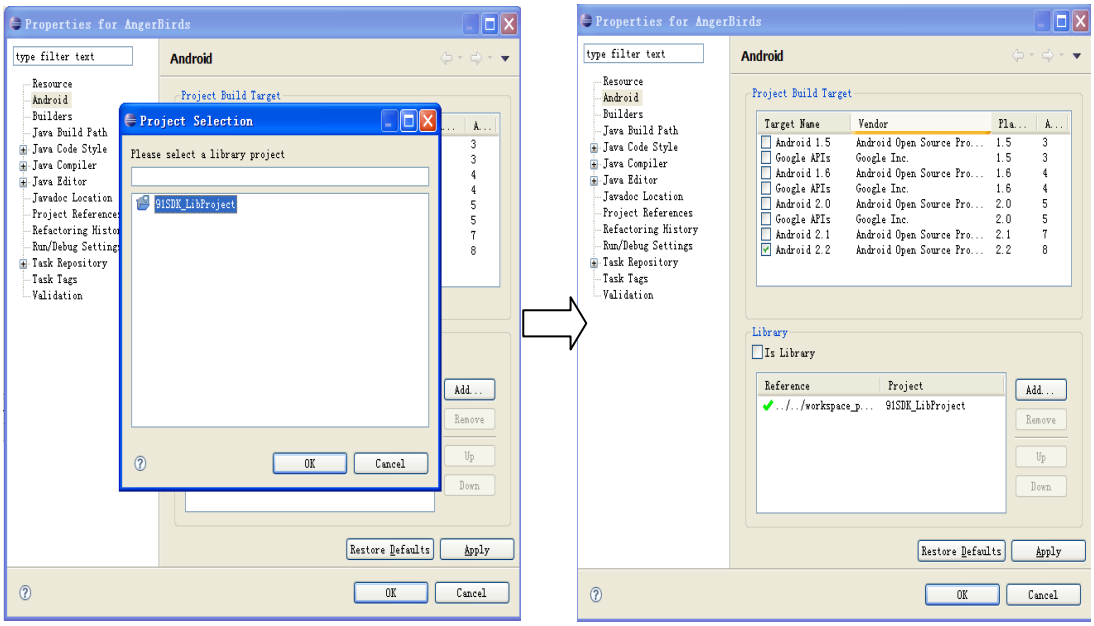
右键点击该工程，选择 Properties 进入界面后选择 Android 并将 Is Library 打勾，如图：



点击 OK 后就成功将 91SDK_LibProject 工程引入并将该工程标识为 library。

2、 为您的工程引进 91SDK_LibProject

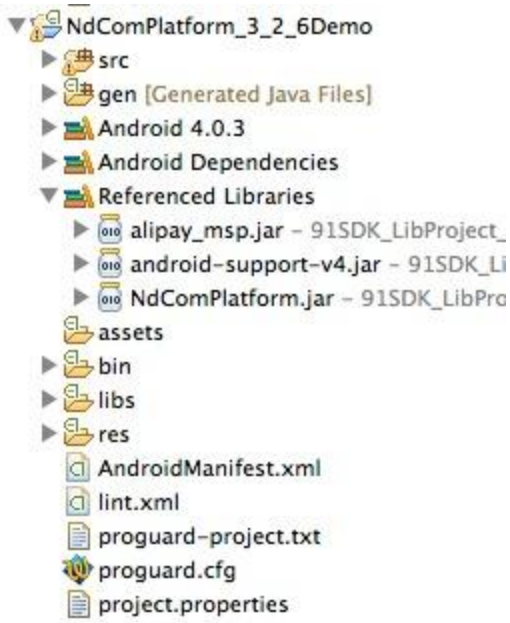
当成功 import 进 91SDK_LibProject 工程后您就可以为您的工程引入 91SDK 了，步骤如下：右键点击您的工程，选择 Properties 后选择 Android，在面板上点击 Add 将 91SDK_LibProject 工程引进来，如图：



选择 OK 就把 91SDK_LibProject 工程的资源引到您的工程里了。

如果您的工程里没有 android-support-v4.jar 包的话，需要将该 jar 包拷贝至 libs 目录下，并添加至构建路径。

最终结果如图：



3、 配置 AndroidManifest.xml

您需要将 AndroidManifest.xml 配置说明.txt 里面的权限，如图：

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

```
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
    <uses-permission android:name="android.permission.CALL_PHONE" />
   <uses-permission android:name="android.permission.GET_TASKS" />
   <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
```

activity，service，receiver 如图：

```
<activity android:name="com.nd.commplatform.activity.SNSControlCenterActivity"
            android:configChanges="orientation|keyboardHidden|navigation|screenSize"
            android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
            android:windowSoftInputMode="adjustPan" android:windowBackground="@null"
            android:launchMode="singleTask"/>

<activity android:name="com.nd.commplatform.activity.SNSLoginActivity"
           android:configChanges="orientation|keyboardHidden|navigation|screenSize"
            android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
            android:windowSoftInputMode="adjustPan"
            android:screenOrientation="landscape"
            android:windowBackground="@null"
            android:launchMode="singleTask"/>

<activity android:name="com.nd.commplatform.gc.activity.ForumActivity"
           android:screenOrientation="landscape"
           android:configChanges="orientation"/>

<service  android:name="com.nd.commplatform.service.NdDownloadService"
                android:process=":com.nd.commplatform.download"
                android:enabled="true"
              android:exported="false"/>

<receiver android:name="com.nd.commplatform.versionupdate.ND2VersionUpdateNotify"
                android:process="android.intent.nd.sns.commplatform.versionupdate"
         android:exported="false"/>

<service android:name="com.nd.commplatform.versionupdate.ND2VersionUpdateService"
                android:process="android.intent.nd.sns.commplatform.versionupdate"
         android:exported="false"/>

<service  android:name="com.nd.commplatform.service.NdNewMessageNotification"
                android:enabled="true"
              android:exported="false"/>
```

拷贝到您的工程的 AndroidManifest.xml 文件里面。

四、 91SDK 基础功能

1、 消息通知

数据通知(ndCallbackListener.class)

该类主要用于将异步请求的结果信息通知给用户，用户只要实现抽象方法 callback 即可捕获请求结果信息，该类包括了如下 API：

回调实现(callback)

```
abstract void callback(int responseCode, T t)
```

您可以通过实现该抽象方法捕捉相关的状态信息及数据，其中：

- responseCode 为返回的错误码或状态码(具体可查看上面对 NdErrorCode 的说明)
- t 是接口返回值泛型，具体的类型会在各个接口的回调传入时声明。

调用方法举例：

```
NdCommplatform.getInstance().ndGetLeaderBoard(ctx,
        new NdCallbackListener<NdPageList<NdRankInfo>>(){

    @Override
    public void callback(int responseCode, NdPageList<NdRankInfo> arg1) {
        // TODO Auto-generated method stub
        if(responseCode == NdErrorCode.ND_COM_PLATFORM_SUCCESS){
            //success
            NdPageList<NdRankInfo> ndPageList = arg1;
        }else{
            //failure
        }
    }

});
```

通知销毁(destroy)

```
void destroy()
```

用于销毁通知。当用户在实现 callback 时有涉及到操作 UI 线程，并且可能由于手机系

统性能或者网络等原因使的程序在执行 callback 前用户就退出该 UI 线程而导致的异常，用户可在退出该 UI 线程时调用该 destroy()方法销毁该通知以避免发生异常。
调用举例：

```
private NdCallbackListener callbackListener;

private void initNdCallbackListener(){
    callbackListener = new NdCallbackListener(){

        @Override
        public void callback(int responseCode, Object t) {
            // TODO Auto-generated method stub
            mMsgCountTextView.setText("91SDK demo");
        }

    };
}

@Override
public void finish() {
    // TODO Auto-generated method stub
    super.finish();
    callbackListener.destroy();
}
```

判断该通知是否销毁(isDestroy)

```
boolean isDestroy()
```

状态通知(NdMiscCallbackListener.class)

该类包括的状态通知有“登录通知”，“支付通知”，“退出平台界面通知”，下面是各个接口的描述：

登录通知(OnLoginProcessListener)

该接口主要用于登录结果通知，接口在取消登录（注册）、完成登录（注册）及自动登录及登录失败后发出通知，您可以通过捕捉回调获取登录状态。通知接口的具体代码如下：

```
interface OnLoginProcessListener {
        void finishLoginProcess(int code);
}
```

其中 code 为登录结果的状态码，有三种状态如下：

对应的状态码(NdErrorCode 中定义)	描述

NdErrorCode.*ND_COM_PLATFORM_SUCCESS*	登录成功
NdErrorCode.*ND_COM_PLATFORM_ERROR_CANCEL*	取消操作
其他状态码	其他错误码则为登录失败

调用举例：

```
NdCommplatform.getInstance().ndLogin(context, new OnLoginProcessListener() {
            @Override
            public void finishLoginProcess(int code) {
                // TODO Auto-generated method stub
                switch(code){
                case NdErrorCode.ND_COM_PLATFORM_SUCCESS:
                    //登录成功
                    break;
                case NdErrorCode.ND_COM_PLATFORM_ERROR_LOGIN_FAIL:
                    //登录失败
                    break;
                case NdErrorCode.ND_COM_PLATFORM_ERROR_CANCEL:
                    //取消登录
                    break;
                default:
                    //登录失败
                }
            }
    });
```

支付通知(OnPayProcessListener)

该通知主要用于捕获支付购买结果的一些状态，不同的支付购买业务接口返回的结果状态不同，具体哪些支付购买业务接口对应哪些结果状态我们会在下面对平台各接口介绍时详细描述。

接口具体代码如下：

```
interface OnPayProcessListener {
        void finishPayProcess(int code);
}
```

如何结合该支付通知的使用会在介绍相关接口进行举例。

退出平台界面通知 (OnPlatformBackground)

通过该消息接口可以捕获关闭平台界面的动作：

```
interface OnPlatformBackground {
    public void onPlatformBackground();
```

```
}
```

注意：该接口是全局性的，适合平台所有界面。如果您有多处调用则会自动覆盖成最后调用的那个。当您关闭平台界面时可以通过这个通知来侦听您的关闭动作。

调用方法举例：

```
NdCommplatform.getInstance().setOnPlatformBackground(
                new OnPlatformBackground() {
            @Override
            public void onPlatformBackground() {
                // TODO Auto-generated method stub
                //do something
            }
        });
```

该通知在 91SDK 所有通知里面优先级是最低的。即会迟于其它通知接口的执行。

2、 初始化 91SDK

函数：

```
void ndInit(Activity activity, NdAppInfo ndAppInfo,
                                        OnInitCompleteListener mOnInitCompleteListener)
```

- activity: 开发者游戏启动页的 Activity
- ndAppInfo: AppId 和 AppKey 等相关的参数设置
- mOnInitCompleteListener: 初始化完成后的通知

调用结束后返回：

对应的状态码(OnInitCompleteListener 中定义)	描述
OnInitCompleteListener.*FLAG_NORMAL*	正常初始化
OnInitCompleteListener.FLAG_FORCE_CLOSE	强制退出游戏

首先您需要您的游戏启动页先调用该初始化函数，来设置您从 91 移动开放平台申请的 AppId 和 AppKey 等参数，并在初始化完成的通知监听接口里实现您自己的游戏逻辑。所以在接入 91 移动开放平台之前，您需要向开发者后台申请可接入的 AppId 和 AppKey。所有的 SDK 的操作都需要设置这两个参数才能够正常工作。**请务必将客户端用的 AppId 和 AppKey 转告服务器端开发人员，确保服务器端用的 AppId 和 AppKey 和客户端的保持一致。**

NdAppInfo.setNdVersionCheckStatus(int flag)方法为可选配置：

对应的状态码(NdAppInfo 中定义)	描述
NdAppInfo.ND_VERSION_CHECK_LEVEL_STRICT	value=0，当获取更新信息失败时强制退出游戏（此为默认选项）
NdAppInfo. ND_VERSION_CHECK_LEVEL_NORMAL	value=1，当获取更新信息失败时继续游戏

举例如下：

```
    private OnInitCompleteListener mOnInitCompleteListener;
```

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    this.setContentView(R.layout.welcome);
    init91SDK();
}

/**
 * 初始化91SDK
 */
private void init91SDK(){
     mOnInitCompleteListener = new OnInitCompleteListener(){
          @Override
          protected void onComplete(int ndFlag) {
               switch (ndFlag) {
               case OnInitCompleteListener.FLAG_NORMAL:
                    initGame();  // 初始化自己的游戏
                    break;
               case OnInitCompleteListener.FLAG_FORCE_CLOSE:
               default:
                    // 如果还有别的Activity或资源要关闭的在这里处理
                    //彻底关闭游戏的处理
                    break;
               }
          }
     };
     NdAppInfo appInfo = new NdAppInfo();
     appInfo.setAppId(Constant.appID_91Bean);//应用ID
     appInfo.setAppKey(Constant.appKEY_91Bean);//应用Key
    //单机或弱联网游戏必须关注setNdVersionCheckStatus的设置，详见上面说明
    appInfo.setNdVersionCheckStatus(NdAppInfo.ND_VERSION_CHECK_LEVEL_STRICT);
    appInfo.setCtx(context);
     //初始化91SDK
     NdCommplatformMisc.getInstance().ndInit(this, appInfo,
               mOnInitCompleteListener);
}

private void initGame(){
     /* 初始化游戏 */
}

@Override
protected void onDestroy() {
     // TODO Auto-generated method stub
```

```
        super.onDestroy();
        if(mOnInitCompleteListener != null){
            mOnInitCompleteListener.destroy();
        }
    }
```

3、 暂停页

函数：

```
void ndPause(OnPauseCompleteListener mOnPauseCompleteListener)
```

- mOnPauseCompleteListener:

游戏暂停时调用该函数。

举例如下：

```
NdCommplatform.getInstance().ndPause(new OnPauseCompleteListener(mContext){
        @Override
        public void onComplete() {
            // TODO Auto-generated method stub
            Toast.makeText(BaseActivity.this, "退出暂停页", Toast.LENGTH_LONG).show();
        }
});
```

4、 退出页

函数：

```
void ndExit(OnExitCompleteListener mOnExitCompleteListener)
```

- mOnExitCompleteListener:

游戏退出时调用该函数。

举例如下：

```
NdCommplatform.getInstance().ndExit(new OnExitCompleteListener(mContext){


        @Override
        public void onComplete() {
            // TODO Auto-generated method stub
            Toast.makeText(ctx, "退出DEMO", Toast.LENGTH_LONG).show();
            //游戏退出逻辑处理
        }


});
```

5、 浮动工具栏

简介

浮动工具栏是一个悬浮在游戏上方的一个可拖动的图标，点击可以展开工具栏，集成了91SDK 的一些常用功能。

浮动工具栏是非单例的，在每调用一次 create()方法时都将生成一个单独的实例。

1) 创建浮动工具栏

函数：

```
static NdToolBar create(Context context, int place)
```

- place: 浮动工具栏初始位置，仅第一次进入时有效，之后工具栏会显示在用户最后一次停留的位置。共有 6 个值可以设置，如下：

对应的状态码(NdToolBarPlace 中定义)	描述
NdToolBarPlace.*NdToolBarTopLeft*	value=1；左上角
NdToolBarPlace.*NdToolBarTopRight*	value=2；右上角
NdToolBarPlace.*NdToolBarLeftMid*	value=3；左边中间
NdToolBarPlace.*NdToolBarRightMid*	value=4；右边中间
NdToolBarPlace.*NdToolBarBottomLeft*	value=5；左下角
NdToolBarPlace.*NdToolBarBottomRight*	value=6；右下角

在要显示浮动工具栏的 Activty 的 onCreate 方法中调用该函数，并将浮动工具栏对象保存为成员变量。

举例如下：

```
private NdToolBar toolBar;


@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
    ....
    ...
    toolBar = NdToolBar.create(this, NdToolBarPlace.NdToolBarBottomRight);
    ...
}
```

2) 显示浮动工具栏

函数：

```
void show()
```

在要显示浮动工具栏的 Activty 中的合适时机调用该函数，例如 onCreate 方法中。

注意：调用该接口前请确保调用过**创建浮动工具栏**接口。

举例如下：

```
private NdToolBar toolBar;


@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
    ....
    ...
    toolBar = NdToolBar.create(this, NdToolBarPlace.NdToolBarBottomRight);
    toolBar.show();
    ...
}
```

3) 隐藏浮动工具栏

函数：

```
void hide()
```

如果有需要隐藏浮动工具栏可以调用此接口。

举例如下：

```
toolBar.hide();
```

4) 回收浮动工具栏

函数：

```
void recycle()
```

在 Activity 销毁的时候调用。回收当前 Activty 中的浮动工具栏实例。

举例如下：

```
@Override
protected void onDestroy() {
    if(toolbar != null) {
        toolBar.recycle();
        toolBar  = null;
    }
    super.onDestroy();
}
```

5) 清除浮动工具栏参数

函数：

```
static void clear()
```

清除浮动工具栏的静态参数，在退出游戏时调用。

注意：在通过退出页退出游戏时已经调用过该接口，有设置退出页的游戏无需调用该接口

举例如下：

```
NdToolBar.clear();
```

6、 设置为调试模式(ndSetDebugMode)

设定为调试模式的支付功能和升级功能

函数：

```
void ndSetDebugMode(int nFlag)
```

nFlag:该参数保留，暂时不用，默认为 0。

注意：该接口要在您调用 SDK 的其他 API 之前调用。

其中的支付功能和升级功能具体包括：

- 游戏版本的检测升级
- 91 豆余额查询和支付
- 代币充值中的 91 豆兑换
- 虚拟商店自定义虚拟币充值中的 91 豆兑换

开发者调用该接口后，SDK 将这些功能转换为测试环境。开发者需要到 http://dev.91.com 进行配置相关的测试数据（测试账号和余额，游戏升级的版本等）。

图示：

注意：这个方法只是您在开发阶段用户测试上面描述的功能用的，当您的游戏或者应用准备正式发布时切记将该方法的调用去掉。否则用户将无法进行支付及升级。

1：配置游戏升级版本

开发者配置后调用接口时，检测升级时将会后台配置的版本号进行比较，低于后台配置的版本时，将会返回一个默认升级包的下载地址。该测试功能，主要用于你的软件中使用了平台的版本检测与升级模块功能。 我们提供这样的功能， 主要是模拟已经有一个新版本发布， 用于测试当前的版本，将来可以正确的升级到新版本。

后台配置如图： 通常测试版本号，需要比本地版本号要高， 才能模拟版本升级

2：配置支付功能的测试账户和余额

支付功能的支付余额默认为 5000 个 91 豆，开发者需要在后台配置测试帐号，即可在测试环境使用。该功能仅提供支付测试，及支付结果通知，用于测试你的软件中的购买流程。支付功能不会产生支付流水，不产生消费记录，也不产生任何收益。后台配置的测试帐号，仅在测试模式下生效，在正式环境中，是作为正常的普通用户。

开发者可以在 dev 后台中，配置测试帐号和查阅测试订单。

7、 检查更新

简介

当开发者要发布新的客户端版本时，为了区分某些版本更新的重要性，可以在后台进行更新属性配置，有“强制更新”和“普通更新”两种。默认是普通更新

强制更新：当客户端发生了重要变更（如修复了某个严重 BUG）或者开发者想强行推广新版本时，可以在开发者后台里把新版本属性设置为强制更新。用户必须更新到最新版本，才允许使用。

普通更新：非强制更新的情况。用户不需要更新到最新版本，即在当前版本也能使用。

开发者可以在后台的【应用管理】中设置是否强制更新。

SDK 仅是告知是哪种形式的更新。开发者需要根据接收到的标识符进行相应的处理。因此我们建议开发者可以按以下方式处理版本更新。

- 当版本检测失败时，按无版本更新处理
- 当收到用户取消普通更新时，允许其登录等后续相关流程
- 当收到用户取消强制更新时，不允许其登录等后续相关流程，同时提示用户需要更新后才能使用。

开发者接入 SDK，在设置完 AppId 和 AppKey 后，应该先检查软件自身版本更新，检查更新完成后才能够进行登录和开始应用的相关操作。这样可以避免您的应用在出现重要更新时无法及时的更新客户端。

同时，如果您是首次接入的开发者或者需要验证更新的流程，您可以通过设置调试模式来进行版本升级的功能测试。具体方法请参见设置调试模式一节。

为了保证版本比较的准确性，需要严格按照标准定义版本号. 具体参见本文档的版本号设定规则章节。

注意：从 3.2.5 开始，NdInit 在初始化时将自动完成检查更新的工作。

8、 捕获退出平台界面的通知(setOnPlatformBackground)

接口：

```
void setOnPlatformBackground(NdMiscCallbackListener.OnPlatformBackground
onPlatformBackground)
```

通过该接口设置捕获退出平台界面通知。详情及调用举例请参考消息通知里的退出平台界面通知

9、 设置平台界面横竖屏方向(ndSetScreenOrientation)

在不调用本接口的情况下，平台界面默认为竖屏方向。

接口：

```
void ndSetScreenOrientation(int orientation)
```

参数 orientation 值为：

对应的状态码（NdCommplatform 中定义）	描述
SCREEN_ORIENTATION_PORTRAIT	竖屏
SCREEN_ORIENTATION_LANDSCAPE	横屏
SCREEN_ORIENTATION_AUTO	自动（由重力感应自动选择）

10、 登录/注销

检查更新完成后，您就可以开始调用登录接口了。

平台提供了两个登录接口，开发者应根据自身需求在其中选择一种进行接入，我们不建议在应用中同时使用两种类型的登录模式：

- 普通登录
  由用户提供账户和密码进行登录和注册，包括第三方类型的登录，这种类型的账户支持平台提供的各类功能，包括好友，支付，充值，成就和排行榜。
- 登录（支持游客登录）
  为没有历史登录账号的设备提供一种游客模式的快速登录，平台为玩家预分配一个 uin 进行，开发者通过这个 uin 进行玩家游戏数据的保持。同时开发者需要及时提醒玩家进行账户转正，成为正式 91 用户。

处于游客登录状态的玩家无法进行包括好友，支付，充值，成就和排行榜等相关操作。

使用支持游客登录的接口需要处理用户账户的判断用户登录状态和游客账户转正式账户。

1) 普通登录(ndLogin)

如果用户是第一次登录，则系统将进到登录界面，如果用户已经登录过则系统将根据前一次的设置信息判断是否自动登录，如果是自动登录则系统将进行自动登录，如果不是自动登录则系统将进入登录界面，用户可以选择输入已有的用户名和密码进行登录，也可以进入注册界面重新注册新账号，登录及注册界面的具体功能和操作可以进入登录和注册界面查看。
接口：

```
void ndLogin(Context ctx,
            NdMiscCallbackListener.OnLoginProcessListener onLoginProcessListener)
```

接口的调用举例请参考登录通知。

2) 游客登录

如果您想让游戏新手玩家，免去注册或登录时输入账号和密码操作，而快速体验游戏，您可以使用游客登录。游客登录只是系统为游客玩家预分配了一个 UIN，开发者可以通过以这个 UIN 为标识来保存玩家的游戏数据，等到玩家需要购买道具、升到一定级别或者退出游戏时，再提醒玩家进行转正注册，成为 91 用户，可查看使用场景案例。
处于游客登录状态的玩家无法进行支付、充值、相关数据的获取等操作。
使用游客登录功能需要配套组合使用登录（支持游客登录），判断用户登录状态，游客账户转正式账户等接口，具体如下接口介绍：

a) 登录(支持游客登录)(ndLoginEx)

本函数在登录(ndLogin)的基础上增加了游客登录的支持。所谓游客登录就是不需要玩家输入账号和密码注册或登录，系统会快速为未登录过 91SDK 的设备分配一个 UIN。开发者开通过这个 UIN 保存玩家的相关游戏数据，等玩家需要注册时再通过引导游客注册设置账号和密码将分配的 UIN 转正。
如果玩家设备已经有登录过 91 账号，则本函数的调用结果与普通登录(ndLogin)是一致的，即走普通登录(ndLogin)登录流程。

函数：

```
void ndLoginEx(Context ctx,
            NdMiscCallbackListener.OnLoginProcessListener onLoginProcessListener)
```

- onLoginProcessListener：登录过程返回的状态码如下：

状态码：

对应的状态码(NdErrorCode 中定义)	描述
NdErrorCode.*ND_COM_PLATFORM_SUCCESS*	登录成功
NdErrorCode.*ND_COM_PLATFORM_ERROR_CANCEL*	取消操作
其他状态码	其他错误码则为登录失败

登录成功后可通过 NdCommplatform.*getInstance*().ndGetLoginStatus()来判断当前是游客登录状态还是 91 账号登录状态，示例代码如下：

调用举例：

```
NdCommplatform.getInstance().ndLoginEx(ctx, new
                NdMiscCallbackListener.OnLoginProcessListener() {


            @Override
            public void finishLoginProcess(int code) {
                String tip = "";
                    // 登录的返回码检查
                if (code == NdErrorCode.ND_COM_PLATFORM_SUCCESS) {
                    finish();
                    HomeActivity.show(ctx);
                    if(NdCommplatform.getInstance().ndGetLoginStatus()
                            == NdLoginStatus.AccountLogin){//账号登录
                        //账号登录成功，此时可用初始化玩家游戏数据
                        tip = "账号登录成功";
                    }else if(NdCommplatform.getInstance().ndGetLoginStatus()
                            == NdLoginStatus.GuestLogin){//游客登录
    //游客登录成功，此时可获取玩家的游客UIN做为保存游戏数据的标识，玩家游客账号转正后该UIN不变。
                        tip = "游客登录成功";
                    }
                } else if (code == NdErrorCode.ND_COM_PLATFORM_ERROR_CANCEL) {
                    tip = "取消账号登录";
                } else  {
                    tip = "登录失败，错误代码：" + code;
                }

                Toast.makeText(ctx, tip, Toast.LENGTH_SHORT).show();
                }
            });
```

b) 获取当前登录状态(ndGetLoginStatus)

函数：

```
NdLoginStatus ndGetLoginStatus()
```

NdLoginStatus 是个枚举，包含三个枚举状态如下：

- NdLoginStatus.NotLogin ： 未登录状态
- NdLoginStatus.AccountLogin：账号登录状态
- NdLoginStatus.GuestLogin：游客登录状态

调用举例：

```
if(NdCommplatform.getInstance().ndGetLoginStatus() != NdLoginStatus.GuestLogin){
        Toast.makeText(ctx, "当前非游客登录模式", Toast.LENGTH_SHORT).show();
        return;
}
```

c) 游客账户转正式账户(ndGuestRegist)

本函数是用于游客登录 Uin 转正（游客注册）。调用后会进入引导游客 Uin 转正界面。

在游客注册界面，游客输入用户名和密码后点注册 91 通行证注册成功后，这时系统的 Uin 与您调用本函数前的 Uin 一致，开发者可以视为该用户正式登录成功；但如果游客选择使用已有 91 通行证登录并登录成后，此时游客登录的 Uin 会变成已有 91 通行证的 Uin。这种情况下，由于 Uin 发生变化，开发者可以视为用户切换帐号，业务层需要使用新的 Uin 进行初始化。

本函数调用的前提是：当前处于游客登录状态，即调用完登录(支持游客登录)后发现当前的登录状态是游客登录状态（可通过获取当前登录状态来判断）。

函数：

```
int ndGuestRegist(Context ctx, String tips,
            NdMiscCallbackListener.OnLoginProcessListener onLoginProcessListener)
```

- 返回值 int：如果当前为非游客登录则返回-1（不允许调用），否则返回 0。
- 参数 tips：可定制的游客转正提示。
- 参数 onLoginProcessListener：游客注册过程返回的状态码如下：

游客转正注册：

对应的状态码(NdErrorCode 中定义)	描述
NdErrorCode.*ND_COM_GUEST_OFFICIAL_SUCCESS*	游客转正注册成功
NdErrorCode.*ND_COM_PLATFORM_ERROR_CANCEL*	取消操作
其他状态码	其他错误码则为登录失败

游客使用已有账号登录：

对应的状态码(NdErrorCode 中定义)	描述
NdErrorCode.*ND_COM_PLATFORM_SUCCESS*	登录成功
NdErrorCode.*ND_COM_PLATFORM_ERROR_CANCEL*	取消操作
其他状态码	其他错误码则为登录失败

调用举例：

```
if(NdCommplatform.getInstance().ndGetLoginStatus() != NdLoginStatus.GuestLogin){
    Toast.makeText(ctx, "当前非游客登录模式", Toast.LENGTH_SHORT).show();
    return;
}

//游客注册
NdCommplatform.getInstance().ndGuestRegist(ctx,
                    "您现在处于游客模式，为了更好地保护您的数据，请注册91通行证",
                    new NdMiscCallbackListener.OnLoginProcessListener() {
                @Override
                public void finishLoginProcess(int code) {
                    String tip = "";
                    hideLoading();
                    switch(code){// 登录的返回码检查
                    case NdErrorCode.ND_COM_GUEST_OFFICIAL_SUCCESS:
                        tip = "游客转正注册成功";
                        break;
                    case NdErrorCode.ND_COM_PLATFORM_SUCCESS:
                        //游客使用已有账户登录成功，此时UIN变成了已有91通行证的UIN
                         //根据UIN重新保存游戏数据
                        tip = "游客使用已有账号登陆成功";
                        break;
                    case NdErrorCode.ND_COM_PLATFORM_ERROR_CANCEL:
                        tip = "取消转正或者登录操作";
                        break;
                    default:
                        tip = "转正或者登录失败，错误代码: " + code;
                    }
                    Toast.makeText(ctx, tip, Toast.LENGTH_SHORT).show();
                }
    });
```

3) 注销登录

注销账号分两种，一种是由开发者在任何场景调用，用来注销当前的登录账号；另一种是由玩家在 91 社区界面里面选择注销当前的账号。下面对两种方式分别作介绍：

a) 开发者触发账号注销

接口：

```
void ndLogout(int nFlag, Context ctx)
```

该接口用于注销当前会话，其中 nFlag 值为：

对应的状态码（NdCommplatform 中定义）	描述
LOGOUT_TO_NON_RESET_AUTO_LOGIN_CONFIG	注销会话
LOGOUT_TO_RESET_AUTO_LOGIN_CONFIG	注销会话以及取消自动登录

调用举例：

```
NdCommplatform.getInstance().ndLogout(NdCommplatform.LOGOUT_TO_NON_RESET_AUTO_LOGIN_CONFIG, context);
```

b) 玩家触发账号注销

玩家触发是指玩家在“91 社区->首页”或者“91 社区->更多”界面点击“切换账号”触发的切换账号事件。此时开发者可以通过设置切换账号时重新启动来设置玩家切换账号时，游戏是否重新启动。及通过设置切换账号监听通知来监听切换账号的过程。

i. 设置切换账号时重新启动(setRestartWhenSwitchAccount)

函数：

```
void setRestartWhenSwitchAccount(boolean restart)
```

- restart：true:游戏重启，false:弹出登录界面切换账号

ii. 设置切换帐号监听通知(setOnSwitchAccountListener)

本函数用于设置玩家在“91 社区->首页”或者“91 社区->更多”界面，点击“切换账号”时，所触发的切换账号过程监听。

本函数为全局函数，多处调用后面设置会覆盖前面设置。

函数：

```
void setOnSwitchAccountListener(
                        NdMiscCallbackListener.OnSwitchAccountListener listener)
```

- NdMiscCallbackListener.OnSwitchAccountListene：账号切换过程监听接口。该接口在登录通知(OnLoginProcessListene)的基础上增加了“用户将要切换账号”和“用户重新启动游戏”状态的监听，开发者需要实现改监听接口以捕捉玩家切换账号过程的通知。通过该监听接口能捕捉到的消息状态码如下：
  1. 当开发者设置设置切换账号时重新启动为重新启动时，账号切换过程能捕捉到的状态码为：

状态码（(NdErrorCode 中定义)）	含义
NdErrorCode.ND_COM_PLATFORM_ERROR_USER_RESTART	用户重新启动应用

2. 当开发者设置设置切换账号时重新启动为弹出登录界面切换账号时，账号切换过程能捕捉到的状态码为：

状态码（(NdErrorCode 中定义)）	含义
NdErrorCode.ND_COM_PLATFORM_ERROR_USER_SWITCH_ACCOUNT	用户将要切换帐号
NdErrorCode.*ND_COM_PLATFORM_SUCCESS*	登录成功
NdErrorCode.*ND_COM_PLATFORM_ERROR_CANCEL*	取消操作
其他状态码	其他错误码则为登录失败

调用举例：

```
NdCommplatform.getInstance().setOnSwitchAccountListener(
            new OnSwitchAccountListener() {// 设置玩家在社区界面选择注销账号时的监听事件
        @Override
        public void onSwitchAccount(int code) {
            if (code == NdErrorCode.ND_COM_PLATFORM_ERROR_USER_SWITCH_ACCOUNT) {
            // 玩家点击社区注销账号并确定注销账号时，捕捉该状态，此时可保存被注销玩家的游戏数据
            //有捕捉到该状态说明游戏不会重启，而只是单纯的切换到登录界面由玩家用新的账号登录
                Toast.makeText(ctx, "玩家将要注销帐号", Toast.LENGTH_SHORT).show();
            } else if (code == NdErrorCode.ND_COM_PLATFORM_ERROR_USER_RESTART) {
            // 玩家点击社区注销账号并确定注销账号时，捕捉该状态，此时可保存被注销玩家的游戏数据
            //有捕捉到该状态说明游戏接下去会进行重启
                Toast.makeText(ctx, "游戏将重新启动", Toast.LENGTH_SHORT).show();
            } else if (code == NdErrorCode.ND_COM_PLATFORM_SUCCESS) {
            // 玩家点击社区注销账号按钮输入新的账号并登录成功时可捕捉该状态，此时可初始化新玩家
               的游戏数据
                Toast.makeText(ctx, "新帐号登录成功", Toast.LENGTH_SHORT).show();
            } else if (code == NdErrorCode.ND_COM_PLATFORM_ERROR_CANCEL) {
            // 玩家点击社区注销账号按钮但未输入新账号登录，而是退出界面可捕捉此状态
                Toast.makeText(ctx, "取消帐号登录", Toast.LENGTH_SHORT).show();
            } else {
            // 玩家点击社区注销账号按钮输入新的账号并登录失败时可捕捉此状态
                Toast.makeText(ctx, "帐号登录失败", Toast.LENGTH_SHORT).show();
            }
        }
});
```

4) 判断是否已经登录(isLogined)

接口：

```
boolean isLogined()
```

返回值为 true 表示已经登录，false 表示未登录。

调用举例：

```
if(NdCommplatform.getInstance().isLogined()){
    //已经是登录状态
}else{
    //未登录状态
}
```

5) 获取会话 ID(getSessionId)

获取本次登录的会话 ID，只有账号登录才能取到。

接口：

```
String getSessionId()
```

11、 用户反馈 (ndUserFeedback)

用户反馈的内容，将在开发者后台呈现．开发者可以通过开发者后台,对用户的反馈进行回复．用户会在平台的系统通知里,收到你回复的内容．因此建议加入该模块。

接口：

```
int ndUserFeedback(Context ctx)
```

12、 进入平台中心 (ndEnterPlatform)

进入平台中心的首页界面。

接口：

```
int ndEnterPlatform(int nFlag,Context ctx)
```

其中 nFlag 为保留参数，暂时不用。

13、 应用内购买

平台提供两种应用内购买模式：同步购买(ndUniPay)和异步购买(ndUniPayAsyn)。您可以根据自己的需求选择使用其中的一种方式。相关应用内购买的信息请参见 SDK 包中

的【91 移动开发平台支付功能接入规范 V2.1.pdf】

1) 如何使用同步购买(ndUniPay)

同步购买接口(ndUniPay)有三种消息状态：

状态码（NdErrorCode 中定义）	描述
ND_COM_PLATFORM_SUCCESS	支付成功
ND_COM_PLATFORM_ERROR_PAY_FAILURE	支付失败
ND_COM_PLATFORM_ERROR_PAY_CANCEL	取消支付

本接口支持调试模式。

调用该 API 前应先判断用户状态，用户未登陆则应先登陆，确保用户在登陆成功后进行购买。

a) 发起购买请求

```
NdBuyInfo buyInfo = new NdBuyInfo();
serial = UUID.randomUUID().toString();//订单号唯一(不能为空)
buyInfo.setSerial(UUID.randomUUID().toString());//订单号唯一(不能为空)
buyInfo.setProductId("680254");/商品ID，厂商也可以使用固定商品ID 例如“1”
buyInfo.setProductName("苹果");//产品名称
buyInfo.setProductPrice(2.60);//产品现价 (不能小于0.01个91豆)
buyInfo.setProductOrginalPrice(2.60);//产品原价，同上面的价格
buyInfo.setCount(3);//购买数量(商品数量最大10000，最新1)
buyInfo.setDesription("gamezoon1");//服务器分区，不超过20个字符，只允许英文或数字

int aError = NdCommplatform.getInstance().ndUniPay(buyInfo, this,
                    new NdMiscCallbackListener.OnPayProcessListener() {
    @Override
    public void finishPayProcess(int code) {
        switch(code){
        case NdErrorCode.ND_COM_PLATFORM_SUCCESS:
            Toast.makeText(ctx, "购买成功", Toast.LENGTH_SHORT).show();
            break;
        case NdErrorCode.ND_COM_PLATFORM_ERROR_PAY_FAILURE:
            Toast.makeText(ctx, "购买失败", Toast.LENGTH_SHORT).show();
            break;
        case NdErrorCode.ND_COM_PLATFORM_ERROR_PAY_CANCEL:
            Toast.makeText(ctx, "取消购买", Toast.LENGTH_SHORT).show();
            break;
        default:
            Toast.makeText(ctx, "购买失败", Toast.LENGTH_SHORT).show();
        }
    }
});
```

```
if(aError != 0){
    Toast.makeText(ctx, "您输入参数有错，无法提交购买请求", Toast.LENGTH_SHORT).show();
}
```

b) 漏单处理

在购买过程中，可能出现以下情况：购买已经完成，而应用程序却没有收到购买结果的通知，例如，购买过程中，用户退出了应用程序，或者网络出现了问题，导致购买成功的消息无法到达客户端。对于这种情况，建议采取以下措施。

在发起购买请求后，立即记录下该请求的订单号，在购买结果到达时，删除该条记录。而在每次应用程序启动时，检查是否有未收到购买结果的订单，如果有，向服务器发起验证，并根据验证结果处理该记录。

必须在账号登陆成功的情况下调用漏单处理，建议调用时机在用户登陆成功的回调中。

客户端向服务器发起验证的接口为：

```
void ndCheckPaySuccess(String orderSerial, Context ctx, NdCallbackListener<Boolean>
callback )
```

其中 orderSerial 订单号。验证结果为 Boolean 型通过数据通知接口通知用户，true 表示购买成功，false 表示购买失败。

该验证接口通过数据通知接口通知返回的状态码为：

状态码（NdErrorCode 中定义）	描述
ND_COM_PLATFORM_SUCCESS	查询成功（注：查询成功不代表支付成功，只有当 NdCallbackListener<Boolean>里的 Boolean 为 true 才表示购买成功）
ND_COM_PLATFORM_ERROR_UNEXIST_ORDER	无此订单
其他错误码	查询失败（注：查询失败不代表支付失败）

调用举例：

```
NdCallbackListener<Boolean> callbackCheckPay = new NdCallbackListener<Boolean>(){
    @Override
    public void callback(int responseCode, Boolean isPay) {
        // TODO Auto-generated method stub
        switch(responseCode){
        case NdErrorCode.ND_COM_PLATFORM_SUCCESS://查询成功
            if(isPay){
                //购买成功
            }else{
                //购买失败
            }
            break;
        case NdErrorCode.ND_COM_PLATFORM_ERROR_UNEXIST_ORDER:
            //无此订单
```

```
                break;
            default :
                //查询失败
            }
        }
};
NdCommplatform.getInstance().ndCheckPaySuccess(serial, context, callbackCheckPay );
```

2) 如何使用异步购买(ndUniPayAsyn)

异步购买接口（ndUniPayAsyn）有五种消息状态：

状态码（NdErrorCode 中定义）	描述
ND_COM_PLATFORM_SUCCESS	购买成功
ND_COM_PLATFORM_ERROR_PAY_FAILURE	购买失败
ND_COM_PLATFORM_ERROR_PAY_CANCEL	取消购买
ND_COM_PLATFORM_ERROR_PAY_ASYN_SMS_SENT	订单已提交，充值短信已发送
ND_COM_PLATFORM_ERROR_PAY_REQUEST_SUBMITTED	*订单已提交*

调用该 API 前应先判断用户状态，用户未登陆则应先登陆，确保用户在登陆成功后进行购买。

异步购买除了支持余额充足的购买以外，还支持另外一种模式：在余额不足情况下购买某物品时，先进行充值，并在充值金额到账后，由服务器自动为您购买该物品。

在余额充足时，异步购买的行为与同步购买一致，在购买结束后回调购买结束消息告知购买结果。客户端可以收到该消息，并确认购买成功时，从服务器更新购买物品的消息。

在余额不足并且未进入充值或者好友代付界面时退出平台界面，此时通知的购买状态为“取消购买”。而在余额不足并且已经进入了充值或者好友代付界面，此时通知的购买状态为“订单已经提交”。必须注意的是：这时，物品很有可能还未到账，无法获取到该物品的购买结果消息。如果已经成功发送了短信则此时通知的购买状态为“订单已提交，充值短信已发送”。

调用举例：

```
NdBuyInfo buyInfo = new NdBuyInfo();
serial = UUID.randomUUID().toString();
buyInfo.setSerial(UUID.randomUUID().toString());//订单号唯一(不能为空)
buyInfo.setProductId("680254");//商品ID，厂商也可以使用固定商品ID 例如“1”
buyInfo.setProductName("苹果");//产品名称
buyInfo.setProductPrice(2.60);//产品现价（不能小于0.01个91豆)
buyInfo.setProductOrginalPrice(2.60);//产品原价，同上面的价格
buyInfo.setCount(3);//购买数量(商品数量最大10000，最新1)
buyInfo.setDesription("gamezoon1");//服务器分区，不超过20个字符，只允许英文或数字

int aError = NdCommplatform.getInstance().ndUniPayAsyn(buyInfo, this,
```

```
                    new NdMiscCallbackListener.OnPayProcessListener() {
    @Override
    public void finishPayProcess(int code) {
        switch(code){
        case NdErrorCode.ND_COM_PLATFORM_SUCCESS:
            Toast.makeText(ctx, "购买成功", Toast.LENGTH_SHORT).show();
            break;
        case NdErrorCode.ND_COM_PLATFORM_ERROR_PAY_FAILURE:
            Toast.makeText(ctx, "购买失败", Toast.LENGTH_SHORT).show();
            break;
        case NdErrorCode.ND_COM_PLATFORM_ERROR_PAY_CANCEL:
            Toast.makeText(ctx, "取消购买", Toast.LENGTH_SHORT).show();
            break;
        case NdErrorCode.ND_COM_PLATFORM_ERROR_PAY_ASYN_SMS_SENT:
            Toast.makeText(ctx, "订单已提交，充值短信已发送", Toast.LENGTH_SHORT).show();
            break;
        case NdErrorCode.ND_COM_PLATFORM_ERROR_PAY_REQUEST_SUBMITTED:
            Toast.makeText(ctx, "订单已提交", Toast.LENGTH_SHORT).show();
            break;
        default:
            Toast.makeText(ctx, "购买失败", Toast.LENGTH_SHORT).show();
        }
    }
});
if(aError != 0){
    Toast.makeText(ctx, "您输入参数有错，无法提交购买请求", Toast.LENGTH_SHORT).show();
}
```

最后，使用异步购买请注意以下几点：

1）再次强调，异步购买要求应用有自己的业务服务器，同时虚拟物品必须通过业务服务器获取。

2）应用程序客户端只能从服务器获取用户所拥有的虚拟物品的信息，不能本地缓存。例如：用户通过异步购买，买了一个虚拟物品“战神斧”。购买结束后，用户打开“背包”查看自己的物品信息，背包里的物品信息必须是从服务器获取的。客户端不能在购买行为结束后，为用户加入“战神斧”物品，必须从自己的业务服务器获取！只有自己的业务服务器确认并发回用户拥有了战神斧物品的信息，应用程序才能认为用户确实获得了该物品。

3）如果客户端想在发起购买请求后刷新物品消息，可以在收到购买消息时，判断如下三种状态时进行刷新：

a）NdErrorCode.*ND_COM_PLATFORM_SUCCESS* 购买成功

b）NdErrorCode.*ND_COM_PLATFORM_ERROR_PAY_ASYN_SMS_SENT* 订单已提交，充值短信已发送

c）NdErrorCode.*ND_COM_PLATFORM_ERROR_PAY_REQUEST_SUBMITTED* *订单已提*

交
但需要再次强调的是，第二和第三种状态时，由于充值到账存在延时，服务端可能不能及时得到该物品的状态信息。

3) 支付结果通知

使用异步购买，支付结果将以消息的方式通知到您的业务服务器或者服务端虚拟商店中。应用接入方需要在开发者后台 配置支付回传通知地址。您的业务服务器需要处理用户的支付结果通知。

a) 开发者需要在后台配置支付结果通知地址

开发者需要在后台配置支付回传地址，接收由 91 移动开发平台服务器发送给各个应用服务商的支付购买结果。服务端对接方式和相应的数据格式具体参见服务端接口规范。

4) 代币充值(ndUniPayForCoin)

代币充值是 SDK 为拥有自己业务服务器的开发者提供的一种便利快捷的充值支付渠道，可以和 SDK 内部的 91 豆充值一样直接对应用内代币进行充值操作，无须在后台配置虚拟物品再进行购买。开发者使用时需要在后台配置使用代币的名称，单位，以及同人民币的汇率，同时开发者还需要配置支付结果通知地址。

SDK 提供的接口如下,其中 cooOrderSerial 是合作商订单号，必须保证唯一，这是双方对账的唯一标记，应该使用 GUID 生成的 32 位的字符串。

调用该 API 前应先判断用户状态，用户未登陆则应先登陆，确保用户在登陆成功后进行购买。

函数：

```
int ndUniPayForCoin(String cooOrderSerial, int needPayCoins, String payDescription, Context ctx)
```

- cooOrderSerial：订单号，32 位 GUID
- needPayCoins：需支付代币个数，改参数如果为 0 或者小于 0 则会进入玩家自己选择充值多少代币界面（此时玩家所充的代币不一定足够购买所需的商品）；如果大于 0 则会进入限制玩家最少要充多少代币界面（需充的代币数开发者可根据玩家的代币余额和购买商品所需代币计算）。
- payDescription：服务器分区，发送支付通知时，原样返回给开发者，长度限制 20 字以内，允许英文或数字。
- 返回值 int 型表示：

状态码（NdErrorCode 中定义）	描述
ND_COM_PLATFORM_ERROR_HAS_NOT_LOGIN	未登录
ND_COM_PLATFORM_SUCCESS	成功进入界面

本接口支持调试模式。

使用场景示例：

《星际迷航 Demo》中使用“神马币”作为游戏内流通的应用代币，其与人民币的的兑换汇率为 1:100，后台配置完成后。在游戏中，玩家 A 的账上还有 30 个神马币的余额，玩家 A 在购买一个单价为 50 个神马币的“战斗机”时，还需支付 20 个神马币，此时玩家使用 API 进行充值时 needPayCoins 应为 20。API 调用示例代码如下所示：

```
String cooOrderSerial = UUID.randomUUID().toString();
int needPayCoins = 20;
String payDescription = "gamezoon1";
NdCommplatform.getInstance().ndUniPayForCoin(cooOrderSerial,
                                        needPayCoins, payDescription, context);
```

调用接口后会进入到 SDK 的代币充值界面，充值结果 SDK 会按后台配置的地址通知开发者的业务服务器，开发者应该及时处理并更新数据，充值成功应该为用户发放相应的代币。客户端在离开代币充值界面会发送退出平台的消息，开发者可以通过捕获退出平台界面的通知来刷新数据，但是由于网络等原因，充值不一定会到账。

使用代币充值需要注意的：

- 开发者应该拥有自己的业务服务器，在后台准确配置消息通知地址，同时实现接收充值结果通知的逻辑。
- 由于网络等原因服务器可能无法及时通知到业务服务器充值结果，开发者可以适当延时再进行查询获取。
- 捕获到退出平台的消息只表示客户端退出充值界面，具体是否充值，充值是否成功未知，客户端需要和业务服务器进行进一步确认。

5) 指定服务器充值

开发者的游戏中，如果存在多个服务器，且在充值时，需要区分充值到哪个服务器，开发者可以按照一下方法进行处理。

每种充值模式的 API 传入参数中，都有一个 payDescription，这个字符串字段，不超过 20 个字符，只允许英文或数字。开发者可以使用这个字段来区分充值到哪个服务器。例如 payDescription 传入 1003，表示游戏服务器 ID。

在游戏服务器与 91 服务器的通信接口中，例如:【接收支付购买结果】,【查询支付购买结果】，这两个接口中的 Note，就是客户端购买时，提交 payDescription,字段。游戏服务器通过处理 Note 字段，即可区分那个游戏服务器。详情查看【91 移动开发平台服务端与应用服务端接口规范 1.00.pdf】

五、 91SDK 扩展功能

该部分为 91SDK 的扩展功能，开发者可以根据自身软件的特性，来参考使用这些扩展功能。

1. 分享到第三方平台 (ndShareToThirdPlatform)

进入分享到第三方平台的界面，如果没有绑定指定的平台，则会跳到绑定界面。
接口：

```
void ndShareToThirdPlatform(Context ctx, String content, Bitmap mPhoto)
```

参数：

- content：预分享的内容（140个字符），第三方平台可能会对重复内容进行屏蔽处理（如新浪微博禁止发重复内容）；
  mPhoto：分享图片到新浪微博，预分享的图片。

2. 捕捉会话过期的通知

可以通过设置本监听接口捕捉会话无效的通知。本监听接口为全局函数，多处调用时后面设置为覆盖前面设置。
函数：

```
void setOnSessionInvalidListener(NdMiscCallbackListener.OnSessionInvalidListener
listener)
```

调用举例：

```
NdCommplatform.getInstance().setOnSessionInvalidListener(
                                        new OnSessionInvalidListener() {
            @Override
            public void onSessionInvalid() {
                // TODO Auto-generated method stub
                Toast.makeText(ctx, "会话无效，请重新登录", Toast.LENGTH_SHORT).show();
            }
    });
```

3. 发送渠道 ID(ndSendChannel)

本函数用于发送渠道到移动开放平台服务端。如果您有调用本函数那么请告诉我们并将您的 APK 安装包发给我们打渠道 ID。
接口：

```
void ndSendChannel(Context ctx,NdCallbackListener callback)
```

数据通知接口通知返回的状态码为：

状态码（(NdErrorCode 中定义)）	含义
ND_COM_PLATFORM_SUCCESS	查询成功
ND_COM_PLATFORM_ERROR_SERVER_RETURN_ERROR	发送失败

4. 释放平台内存(destory)

本函数用于退出游戏或者应用时释放平台资源。

接口：

```
void destory()
```

5. 进入平台

1) 进入好友中心 (ndEnterFriendCenter)

进入好友中心的界面。

接口：

```
int ndEnterFriendCenter(int nFlag, Context ctx)
```

其中 nFlag 为保留参数，暂时不用。

2) 进入指定用户的空间(ndEnterUserSpace)

进入指定用户的空间，可以查看用户的详情信息，排行榜游戏信息。

接口：

```
int ndEnterUserSpace(Context ctx, String strUin)
```

其中 strUin 为用户的 uin。

如果 uin 是好友，则可以进行好友备注修改、发送消息、查看他的新鲜事、他的好友、删除好友操作。

如果 uin 不是好友，则可以进行添加好友操作。

如果 uin 为空，返回参数错误。

接口返回码为：

对应的状态码（NdErrorCode 中定义）	描述
ND_COM_PLATFORM_ERROR_HAS_NOT_LOGIN	未登录
ND_COM_PLATFORM_ERROR_PARAM	参数错误
ND_COM_PLATFORM_SUCCESS	进入成功

3) 进入游戏大厅(ndEnterAppCenter)

接口：

```
int ndEnterAppCenter(int nFlag, Context ctx)
```

其中 nFlag 为保留参数，暂时不用。

4) 进入指定应用的主页（ndEnterAppCenter)

接口：

```
int ndEnterAppCenter(int nFlag, Context ctx, int appId)
```

其中 nFlag 为保留参数，暂时不用。appId 为指定应用的应用 ID。

5) 进入设置界面（ndEnterUserSetting)

接口：

```
int ndEnterUserSetting(int nFlag, Context ctx)
```

其中 nFlag 为保留参数，暂时不用。

6) 进入邀请好友界面（ndInviteFriend)

进入平台的批量邀请好友的界面。

接口：

```
int ndInviteFriend(String inviteContent, Context ctx)
```

- inviteContent 是邀请的内容，可为空。

接口返回码为：

对应的状态码（NdErrorCode 中定义）	描述
ND_COM_PLATFORM_SUCCESS	打开界面
ND_COM_PLATFORM_ERROR_HAS_NOT_LOGIN	未登录

调用举例：

```
NdCommplatform.getInstance().ndInviteFriend("一起来玩吧！", context);
```

7) 进入应用论坛界面（ndEnterAppBBS)

接口：

```
int ndEnterAppBBS(Context ctx,int nFlag)
```

其中 nFlag 为保留参数，暂时不用。

8) 进入账号管理界面（ndEnterAccountManage)

调用该接口，如果已经登录时则进入账号管理界面，如果是处于游客登录状态则返回：NdErrorCode.*ND_COM_PLATFORM_ERROR_HAS_NOT_LOGIN*。如果是未登录则会调用普通登录接口进行登录。

接口：

```
int ndEnterAccountManage(Context ctx,
            NdMiscCallbackListener.OnLoginProcessListener onLoginProcessListener)
```

- onLoginProcessListener：回调参考登录通知。

6. 虚拟商店

1) 简介

首先需要在开发者后台配置相关的虚拟商品信息。

虚拟商店可以配置使用 91 豆购买虚拟商品或者是自定义游戏币购买虚拟商品。

商品计费类型：非消费型商品，订阅型商品，消费型商品，总共 3 种。非消费型商品一次购买可以终身使用。订阅型商品购买后使用有时间限制，超过时间段就失效不能用。消费型商品按照剩余可用次数表示商品的可用性。

商品计费类型：

商品计费类型（NdFeeInfo 中定义）	描述
FEE_TYPE_NON_CONSUMABLE	非消费型(一次购买，终身使用)
FEE_TYPE_SUBSCRIBE	订阅型(按时段消费、同时可能按次/数量)
FEE_TYPE_CONSUMABLE	消费型(按次/数量)

商品类别：可以对商品进行归类，指定类别名称，避免商品过多引起的杂乱无章。比如农场游戏，指定“水果”、“蔬菜”等类别的虚拟商品。该属性不是必需的。

商品促销信息：后台还可以进行促销信息的定制。

注：需要登录后才能调用该模块接口。为了减少冗余的示例代码，下面的示例代码假设用户已登录了平台。

关于虚拟商品的展示，购买与使用流程等详细信息，可以参照【91 移动开发平台支付功能接入规范 V2.1.pdf】

2) 进入虚拟商店

平台有提供虚拟商品展示界面，进去后用户可以查看商品的详细信息，可以进行购买相关操作，对所有商品一目了然。

开发者可以对商店里展示的虚拟商品进行商品计费类型、商品类别的过滤。

函数接口：

```
void ndEnterVirtualShop(String cateId, int feeType, int packageId, Context ctx)
```

- 参数cateId是商品类别ID。如果要指定展示某一类的虚拟商品，需要知道类别ID，可以通过获取虚拟商品类别信息接口获得。
- 参数feeType是商品计费类型，可以同时查询多种计费类型。
- 参数packageId是用于客户端不同版本间区分虚拟商品包选取，可以通过后台配置，0

代表忽略参数。

示例代码：

```
Int nFeeType = NdFeeInfo.FEE_TYPE_NON_CONSUMABLE
        | NdFeeInfo.FEE_TYPE_SUBSCRIBE
        | NdFeeInfo.FEE_TYPE_CONSUMABLE;     //所有计费类型
NdCommplatform sdk = NdCommplatform.getInstance();
sdk .ndEnterVirtualShop(null, nFeeType,  0,  context);    //所有类别
```

3) 获取虚拟商品类别

获取后台配置的所有虚拟商品类别信息。开发者可以使用接口返回数据自定义类别展示界面。

函数接口：

```
void ndGetCategoryList(Context ctx, NdCallbackListener<List<NdCateInfo>> callback)
```

通过数据通知接口通知返回的状态码为：

返回码（NdErrorCode中定义）	**含义**
ND_COM_PLATFORM_SUCCESS	成功
ND_COM_PLATFORM_ERROR_SERVER_RETURN_ERROR	服务器处理失败
ND_COM_PLATFORM_ERROR_CLIENT_APP_ID_INVALID	应用Id无效

示例代码：

```
NdCommplatform sdk = NdCommplatform.getInstance();
NdCallbackListener<List<NdCateInfo>> callback = new NdCallbackListener<List<NdCateInfo>>() {
        @Override
    public void callback(int responseCode, List<NdCateInfo> t) {
        //91SDK  获取商品列表结果
        if (responseCode == NdErrorCode.ND_COM_PLATFORM_SUCCESS && t != null) {
            //TODO:  成功
        }

        if (responseCode != NdErrorCode.ND_COM_PLATFORM_SUCCESS) {
            //TODO:  失败
        }
    }
};
sdk . ndGetCategoryList (context, callback );
```

4) 获取应用促销信息

如果后台有配置促销信息，开发者想自己展示促销信息内容，可以使用该方法。注：虚拟商店界面有展示促销信息。

函数接口：

```
void ndGetAppPromotion(Context ctx,
                       NdCallbackListener<String> callback)
```

(1) 如果没有促销信息，文本为空字符串。

通过数据通知接口通知返回的状态码为：

返回码（NdErrorCode中定义）	含义
ND_COM_PLATFORM_SUCCESS	成功
ND_COM_PLATFORM_ERROR_SERVER_RETURN_ERROR	服务器处理失败
ND_COM_PLATFORM_ERROR_CLIENT_APP_ID_INVALID	应用Id无效

示例代码：

```
NdCommplatform sdk = NdCommplatform.getInstance();
NdCallbackListener<String>        callback = new NdCallbackListener<String>() {
            @Override
            public void callback(int responseCode, String t) {
                if (responseCode == NdErrorCode.ND_COM_PLATFORM_SUCCESS ) {
                    //TODO: 成功
                } else {
                    //TODO: 失败
                }
            };
}
//91SDK  获取商品促销信息
sdk.ndGetAppPromotion(context, callback);
```

5) 获取商店里的商品信息列表

开发者可以指定商品计费信息与商品种类，获取虚拟商品信息列表原始页数据，并自定义展示界面。

函数接口：

```
void ndGetCommodityList(String cateId,
                        int feeType,
                        int packageId,
                        NdPagination pagination,
                        Context ctx,
```

```
NdCallbackListener<NdPageList<NdProductInfo>> callback)
```

(1) 位置1-3参数参考进入虚拟商店的参数说明。

通过数据通知接口通知返回的状态码为：

返回码（NdErrorCode中定义）	含义
ND_COM_PLATFORM_SUCCESS	成功
ND_COM_PLATFORM_ERROR_SERVER_RETURN_ERROR	服务器处理失败
ND_COM_PLATFORM_ERROR_CLIENT_APP_ID_INVALID	应用Id无效
ND_COM_PLATFORM_ERROR_INVALID_PRODUCT_CATE	商品类别无效
ND_COM_PLATFORM_ERROR_INVALID_FEE_TYPE	商品计费类型无效

示例代码：

```
//分页获取虚拟商品信息
NdPagination pagination = new NdPagination();
pagination.setPageIndex(1); //设置页索引，从 1 开始
pagination.setPageSize(10); //设置页大小，是 5 的倍数，不要超过 50
NdCallbackListener<NdPageList<NdProductInfo>>  callback =
                                    new NdCallbackListener<NdPageList<NdProductInfo>>() {
        @Override
        public void callback(int responseCode, NdPageList<NdProductInfo>  t) {
            if (responseCode == NdErrorCode.ND_COM_PLATFORM_SUCCESS) {
                //TODO: 成功
                // NdPageList<NdProductInfo> 包含页索引以及对应的数据列表。此外还包含总数
                //可以根据总数与页大小计算页索引, 具体查看NdPageList<NdProductInfo>定义
            } else {
                //TODO: 失败
            }
        }
};
NdCommplatform sdk = NdCommplatform.getInstance();
sdk.ndGetCommodityList(cateId,  nFeeType,  0,  pagination, context, callback);
```

6) 购买虚拟商品

购买指定的虚拟商品，购买成功或者失败，接口会弹窗提示，开发者无需再做界面提示。购买流程结束会抛消息通知开发者。

a)如果虚拟商品是 91 豆支付模式，会先请求该商品是否可购买，如果可购买则进入异步支付的购买界面；

b)如果虚拟商品是游戏币支付模式，会直接请求支付。如果余额不足会引导用户进入虚拟币直充界面。

函数接口：

```
int ndBuyCommodity(String productId,
```

```
                    String payDesc,
                    int count,
                    Context ctx,
                    NdCallbackListener<NdVirtualPayResult> callback) {
```

- 参数productId是虚拟商店的商品id，获取商品信息列表后，可以取得该ID。
- 参数payDesc是购买时的描述备注。
- 参数count代表消费型商品的购买数量，单次购买数量不能超过10000。购买的商品类型是非消费型或者订阅型时，参数值必须是1。
- 函数返回非0时，表示参数错误。
- NdVirtualPayResult包含了本次支付结果，该参数返回本次购买的订单号、错误码及错误描述，错误码包括（请参考CHM文档）：

错误码	错误描述
1	商品不存在
2	商品不在销售期内
3	商品已下架
4	商品已售完
5	此商品已购买，无需重复购买(非消费)
6	此商品已购买，无需重复购买(订阅)
7	消费型商品购买数量超过限制
8	此商品不在购买时间段内
9	此商品正在购买支付中，无需重复购买(非消费)
10	此商品正在购买支付中，无需重复购买(订阅)
11	虚拟币余额不足，请充值

数据通知回调返回的状态码：

状态码(NdErrorCode中定义)	**含义**
ND_COM_PLATFORM_SUCCESS	购买*成功*
ND_COM_PLATFORM_ERROR_PAY_CANCEL	用户取消购买 注： 游戏币支付模式下，余额不足且用户取消充值后，返回该错误码 91豆支付模式下，用户在确定购买或余额不足界面未做任何操作，直接退出，返回该错误码
ND_COM_PLATFORM_ERROR_PAY_FAILURE	购买失败
ND_COM_PLATFORM_ERROR_PAY_ASYN_SMS_SENT	异步支付成功发送短信 注： 用户进入充值页面，并且选择短信充值方式发送充值短信
ND_COM_PLATFORM_ERROR_PAY_REQUEST_SUBMITTED	订单已经提交 注： 91豆支付模式，用户进入充值页面或者请求好友代付
ND_COM_PLATFORM_ERROR_QUERY_BALANCE_FAIL	余额查询失败

ND_COM_PLATFORM_ERROR_REQUEST_SERIAL_FAIL	获取虚拟商品订单号失败 注： 91豆支付模式下，获取虚拟商品订单号失败
ND_COM_PLATFORM_ERROR_EXIT_FROM_RECHARGE	退出充值界面 注： 游戏币支付模式下， 用户进入充值页面后返回，用户可能进行过充值操作

示例代码：

```
NdCallbackListener callbackz = new NdCallbackListener<NdVirtualPayResult>() {
        @Override
        public void callback(int responseCode, NdVirtualPayResult t) {
            //91SDK 支付结果回调
            if (responseCode == NdErrorCode.ND_COM_PLATFORM_SUCCESS) { // 购买成功
                //从NdVirtualPayResult获取订单号
                //TODO:购买成功
                 //获取订单号：t.getOrderSerial()
            } else { // 购买失败
                //判断NdVirtualPayResult是否是null，从NdVirtualPayResult中可以获取失败信息
                if (t != null) {
                    //TODO： 购买失败
                    //获取错误描述：t. getErrDesc ()
                 } else {
                    switch(responseCode){
                    case NdErrorCode.ND_COM_PLATFORM_ERROR_PAY_CANCEL:// 取消购买
                        //游戏币支付模式下，余额不足且用户取消充值后，返回该错误码
                    //91豆支付模式下，用户在确定购买或余额不足界面未做任何操作，直接退出，返回该错误码
                        //TODO： 取消购买
                        break;
                    case NdErrorCode.ND_COM_PLATFORM_ERROR_PAY_FAILURE:// 购买失败
                        //TODO： 购买失败
                        break;
                    case NdErrorCode.ND_COM_PLATFORM_ERROR_PAY_ASYN_SMS_SENT:
                         //用户进入充值页面，并且选择短信充值方式发送充值短信
                          //TODO： 订单已提交，充值短信已发送
                        break;
                    case NdErrorCode.ND_COM_PLATFORM_ERROR_PAY_REQUEST_SUBMITTED:
                        //91豆支付模式下， 用户进入充值页面或者请求好友代付
                         //TODO： 订单已提交
                        break;
                    case NdErrorCode.ND_COM_PLATFORM_ERROR_QUERY_BALANCE_FAIL:
                        //虚拟币余额查询失败
```

```
                            //TODO:  虚拟币余额查询失败
                        break;
                    case NdErrorCode.ND_COM_PLATFORM_ERROR_REQUEST_SERIAL_FAIL:
                        //91豆支付模式下，获取虚拟商品订单号失败
                        //TODO:  获取虚拟商品订单号失败
                        break;
                    case NdErrorCode.ND_COM_PLATFORM_ERROR_EXIT_FROM_RECHARGE:
                        //游戏币支付模式下， 用户进入充值页面后返回，用户可能进行过充值操作
                        //TODO:  退出充值界面
                        break;
                    default:// 其他错误码，均为购买失败
                        //TODO:  购买失败
                    }
                }
            }
        }
};
//91SDK 购买虚拟商品
NdCommplatform.getInstance().ndBuyCommodity(productId,"购买描述", 1, context, callbackz);
//购买数量：1
```

附：

1) 非消费型商品买了一次就不需要再买，因此在不知道指定的虚拟商品是否已经购买的情况下，可以使用查询指定的虚拟商品授权信息查看其所有权。同样的对与其它计费类型的商品也可以查询其所有权，决定是否还需要购买。

7) 获取已购买的商品信息列表

获取用户已经购买过的商品列表页数据。开发者可以使用返回数据订制自己的已购买商品列表界面。

函数接口：

```
void ndGetUserProductList(NdPagination pagination,
            Context ctx,
             NdCallbackListener<NdPageList<NdPurchasedProductInfo>> callback)
```

通过数据通知接口通知返回的状态码为：

返回码(NdErrorCode中定义)	含义
ND_COM_PLATFORM_SUCCESS	成功
ND_COM_PLATFORM_ERROR_SERVER_RETURN_ERROR	服务器处理失败
ND_COM_PLATFORM_ERROR_CLIENT_APP_ID_INVALID	应用Id无效

示例代码：

```
NdCallbackListener<NdPageList<NdPurchasedProductInfo>> callback =
                    NdCallbackListener<NdPageList<NdPurchasedProductInfo>>() {

                        @Override
                        public void callback(int responseCode, NdPageList<NdPurchasedProductInfo> t) {

                            if (responseCode == NdErrorCode.ND_COM_PLATFORM_SUCCESS) {
                                //TODO:  获取成功
                            }else {
                                  //TODO:获取已购买商品列表失败"
                               }
                            }
                    };
}
NdCommplatform sdk = NdCommplatform.getInstance();
//页索引： pagination，包含页码，页大小
NdPagination pagination = new NdPagination();
pagination.setPageIndex(1);    //获取第1页
pagination.setPageSize(10);    //每页10个数据项，5的倍数，不超过50
sdk.ndGetUserProductList(pagination, context, callback);
```

8) 查询指定虚拟商品授权信息

商品授权包含以下信息：(1)商品 Id;(2)商品计费类型;(3)是否已经购买了商品;(4)如果已经购买，它的剩余的可用次数;(5)如果已经购买，它是否可以在本机使用;(6)如果已经购买，且该商品是订阅型商品，它的可用起始与终止时间。

该接口可以用于判断某个商品已经购买，比如进入自定义虚拟商店准备购买商品，在不知道某个商品是否已经购买的情况下，可以使用该接口查询。

函数接口：

```
void ndGetUserProduct(String productId,
                      Context ctx,
                       NdCallbackListener<NdProductOwnershipInfo> callback)
```

通过数据通知接口通知返回的状态码为：

返回码(NdErrorCode中定义)	**含义**
ND_COM_PLATFORM_SUCCESS	成功
ND_COM_PLATFORM_ERROR_SERVER_RETURN_ERROR	服务器处理失败
ND_COM_PLATFORM_ERROR_CLIENT_APP_ID_INVALID	应用Id无效

示例代码：

```
NdCallbackListener callback = new NdCallbackListener<NdProductOwnershipInfo>() {
                @Override
                public void callback(int responseCode, NdProductOwnershipInfo t) {
                    if (responseCode == NdErrorCode.ND_COM_PLATFORM_SUCCESS) {
                        if (t.hasOwnership()) {
                            //TODO:已经买了这个商品

                            //判断商品是否在当前机器可以使用
                            NdProductAuthInfo authInfo = t.getProductAuthInfo()
                            if (!authInfo.isUsableOnCurrentDevice()) {
                                //TODO:该商品仅不允许在本机使用,
                            } else {
                                //TODO:该商品仅允许在本机使用,
                            }
                        } else {
                            int errorCode = t.getErrorCode();
                            if (errorCode == NdProductOwnershipInfo.Error_Product_Not_Exist ||
                                errorCode == NdProductOwnershipInfo.Error_Other) {
                                //TODO: 未购买，同时不能买的原因, t.getErrorDesc()
                            } else if (errorCode ==
        NdProductOwnershipInfo.Error_Product_Not_Own_Or_Expired) {
                                //TODO: 未购买或者已经失效
                            }
                        }
                    } else {
                        //TODO: 未购买，获取商品授权信息失败
                    }
                }
            };
        }
NdCommplatform sdk = NdCommplatform.getInstance();
sdk.ndGetUserProduct(productId, context, callback);
```

附：

(1) 非消费型商品可以使用ndProductIsPayed接口判断是否已经购买，返回值判断流程与上面的示例一样。

(2) 订阅型商品可以使用ndProductIsExpired接口判断是否已经购买，返回值判断流程与上面的示例一样。

9) 使用已购买的虚拟商品

使用指定的虚拟商品，该商品应该是已购买并可用的。对于有使用次数/数量限制的商品，在调用该接口后，服务端会对该商品进行可用次数/数量进行相应的消减操作。比如有

使用次数限制的订阅型商品，具备数量属性的消费型商品。

注意：已经购买的商品不一定就能在本机使用，某些非消费型的商品可以配置成只在购买机器使用，因此使用前必须判断是否可以在本机用。

函数接口：

```
void ndUseHolding(String productId, int useCnt, String extParam, Context ctx,
                  NdCallbackListener<NdProductUseResult> callback)
```

- productId 获取开发者商品编号(商品在应用中的Id)
- useCnt 本次使用数量
- 参数extParam保留，目前未使用。
- 参数useCnt代表使用商品本次的使用次数。

通过数据通知接口通知返回的状态码为：

返回码(NdErrorCode中定义)	含义
ND_COM_PLATFORM_SUCCESS	成功
ND_COM_PLATFORM_ERROR_SERVER_RETURN_ERROR	服务器处理失败
ND_COM_PLATFORM_ERROR_CLIENT_APP_ID_INVALID	应用Id无效

示例代码：

```
NdCallbackListener mUseGoodsCallback = new NdCallbackListener<NdProductUseResult>() {
        @Override
        public void callback(int responseCode, NdProductUseResult t) {
            //91SDK 商品使用结果返回代码
            String errDesc = null;
            if (responseCode == NdErrorCode.ND_COM_PLATFORM_SUCCESS) {
                if (t.isSuccess()) {
                    //使用成功
                    Toast.makeText(GoodsPurchasedDetailActivity.this, "使用成功",
Toast.LENGTH_LONG).show();
                    updateAuthInfo(t.getFeeType(), t.getProductAuthInfo());
                } else {
                    //使用失败
                    errDesc = t.getErrDesc();
                }
            } else {
                //使用失败
                errDesc = "网络请求失败";
            }

            if (errDesc != null) {
                Toast.makeText(GoodsPurchasedDetailActivity.this,
                                errDesc, Toast.LENGTH_LONG).show();
            }
        }
```

```
};


NdCommplatform sdk = NdCommplatform.getInstance();
// 91SDK 使用商品
sdk.ndUseHolding(mProduct.getProductId(), useCount, null, this, mUseGoodsCallback);
```

10) 查询游戏币余额

该接口用来查询虚拟商店中**自定义（非 91 豆）**的游戏币余额，如果虚拟商店配置的不是 91 豆，不要调用该接口。

函数接口：

```
void ndGetVirtualBalance(Context ctx, NdCallbackListener<Double> callback)
```

通过数据通知接口通知返回的状态码为：

返回码(NdErrorCode中定义)	**含义**
ND_COM_PLATFORM_SUCCESS	成功
ND_COM_PLATFORM_ERROR_QUERY_BALANCE_FAIL	查询余额失败

示例代码：

```
NdCallbackListener<Double> callback = new NdCallbackListener<Double>() {


        @Override
        public void callback(int responseCode, Double t) {
            if (responseCode == NdErrorCode.ND_COM_PLATFORM_SUCCESS) {
                //TODO: 获取余额t.doubleValue());
            }else {
                 //TODO: 获取余额失败
            }
        }
};
NdCommplatform sdk = NdCommplatform.getInstance();
//91SDK 查询自定义的游戏币余额
sdk.ndGetVirtualBalance(context, callback);
```

7. 获取平台数据信息

1) 数据结构介绍

- **NdPagination**

NdPagination 为封装分页信息的类，内容包括：

```
pageIndex;  // 要获取的第几页记录(从1开始)
pageSize;   // 每页记录的个数(5 的倍数，最多 50 个)
```

这两个数据项数据可以通过 set 和 get 方法设置或者获取。在未设置这两个数据项的情况下，平台默认为 pageIndex = 1; pageSize = 5。

- **NdPageList<T>**

NdPageList<T>为封装某一分页数据信息的类，内容包括：

```
NdPagination  pagination;   // 分页索引(上面已介绍)
int           totalCount;   // 所有数据总数（不是单页数据数）
List<T>       list;         // 页数据集合(泛型为数据结构体)
```

2) 获取当前应用的玩家列表(ndGetAppUserList)

接口：

```
void ndGetAppUserList(NdPagination pagination,Context ctx,
            NdCallbackListener<NdPageList<NdStrangerUserInfo>> callback)
```

NdPagination 与 NdPageList 请查看数据结构介绍。

数据 NdStrangerUserInfo（查看 CHM 文档）通过数据通知回调返回。该接口的 NdCallbackListener 错误码为：

状态码（NdErrorCode 中定义）	描述
ND_COM_PLATFORM_SUCCESS	*查询成功*
ND_COM_PLATFORM_ERROR_SERVER_RETURN_ERROR	服务器处理发生错误，请求无法完成
ND_COM_PLATFORM_ERROR_APP_NOT_EXIST	该应用不存在

调用举例

```
pagination = new NdPagination();//分页信息
pagination.setPageIndex(1);//初始化分页信息
pagination.setPageSize(5);//初始化分页信息
NdCallbackListener callback1 = new NdCallbackListener<NdPageList<NdStrangerUserInfo>>(){
        @Override
        public void callback(int arg0, NdPageList<NdStrangerUserInfo> pageList) {
            // TODO Auto-generated method stub
            switch(arg0){
            case NdErrorCode.ND_COM_PLATFORM_SUCCESS://查询成功
                if(pageList != null){
                    pagination = pageList.getPagination();//当前的分页信息
                    totalCount = pageList.getTotalCount();//所有数据总数
                    strangerUserInfoList = pageList.getList();//当前页数据集合
                }
                break;
            case NdErrorCode.ND_COM_PLATFORM_ERROR_SERVER_RETURN_ERROR://服务器错误
                //do something
```

```
                break;
            case NdErrorCode.ND_COM_PLATFORM_ERROR_APP_NOT_EXIST://该应用不存在
                //do something
                break;
            }
        }
    };
    NdCommplatform.getInstance().ndGetAppUserList(pagination, ctx, callback1);
```

3) 获取当前应用的我的好友列表(ndGetAppMyFriendList)

接口：

```
void ndGetAppMyFriendList(NdPagination pagination,Context ctx,
            NdCallbackListener<NdPageList<NdFriendUserInfo>> callback)
```

NdPagination 与 NdPageList 请查看数据结构介绍。
数据 NdFriendUserInfo（查看 CHM 文档）通过数据通知回调返回。该接口的 NdCallbackListener 错误码为：

状态码（NdErrorCode 中定义）	含义
ND_COM_PLATFORM_SUCCESS	*查询成功*
ND_COM_PLATFORM_ERROR_SERVER_RETURN_ERROR	服务器处理发生错误，请求无法完成
ND_COM_PLATFORM_ERROR_APP_NOT_EXIST	该应用不存在

调用举例请参获取当前应用的玩家列表(ndGetAppUserList)

4) 获取我的好友列表(ndGetMyFriendList)

接口：

```
void ndGetMyFriendList(NdPagination pagination,Context ctx,
            NdCallbackListener<NdPageList<NdFriendUserInfo>> callback)
```

NdPagination 与 NdPageList 请查看数据结构介绍。
数据 NdFriendUserInfo（查看 CHM 文档）通过数据通知回调返回。该接口的 NdCallbackListener 错误码为：

状态码（NdErrorCode 中定义）	含义
ND_COM_PLATFORM_SUCCESS	*查询成功*
ND_COM_PLATFORM_ERROR_SERVER_RETURN_ERROR	服务器处理发生错误，请求无法完成

调用举例请参获取当前应用的玩家列表(ndGetAppUserList)

5) 获取登录用户昵称(ndLoginNickName)

接口：

```
String getLoginNickName()
```

6) 获取登录用户 uin(ndLoginUin)

接口：

```
String getLoginUin()
```

7) 获取当前应用名称(getAppName)

接口：

```
String getAppName()
```

8) 获取我的信息(ndGetMyInfo)

调用该接口可获取登录用户的基本信息包括 UIN，昵称，头像的 checksum。

接口：

```
NdMyUserInfo ndGetMyInfo()
```

NdMyUserInfo 请查看 chm 文档定义。

9) 获取我的详细信息(ndGetMyInfoDetail)

接口：

```
int ndGetMyInfoDetail(Context ctx, NdCallbackListener<NdMyUserInfoDetail> callback)
```

详细信息 NdMyUserInfoDetail 通过数据通知回调返回。

该接口的 NdCallbackListener 错误码为：

状态码（NdErrorCode 中定义）	含义
ND_COM_PLATFORM_SUCCESS	*成功*
ND_COM_PLATFORM_ERROR_SERVER_RETURN_ERROR	服务器处理发生错误，请求无法完成
ND_COM_PLATFORM_ERROR_USER_NOT_EXIST	该用户不存在
ND_COM_PLATFORM_ERROR_PERMISSION_NOT_ENOUGH	*权限不足*

调用举例：

```
NdCallbackListener<NdMyUserInfoDetail> callbackMyInfoDetail = new
NdCallbackListener<NdMyUserInfoDetail>(){


    @Override
    public void callback(int responseCode, NdMyUserInfoDetail userInfoDetail) {
```

```
            // TODO Auto-generated method stub
            switch(responseCode){
            case NdErrorCode.ND_COM_PLATFORM_SUCCESS:
                //查询成功
                break;
            case NdErrorCode.ND_COM_PLATFORM_ERROR_SERVER_RETURN_ERROR:
                //查询失败
                break;
            case NdErrorCode.ND_COM_PLATFORM_ERROR_USER_NOT_EXIST:
                //该用户不存在
                break;
            case NdErrorCode.ND_COM_PLATFORM_ERROR_PERMISSION_NOT_ENOUGH:
                //权限不足
                break;
            default :
                //查询失败
            }
        }
};
NdCommplatform.getInstance().ndGetMyInfoDetail(context, callbackMyInfoDetail);
```

10) 获取用户详细信息(ndGetUserInfoDetail)

接口：

```
void ndGetUserInfoDetail(String uin, int flag, Context ctx,
                NdCallbackListener<NdUserInfo> callback)
```

其中 uin 为用户的 id；flag 为获取标识，按位组合，flag 的值如下：

状态码（NdCommplatform 中定义）	含义
0	无效
GET_USER_BASE_INFO	基本信息
GET_USER_POINT	积分
GET_USER_EMOTION	心情
举例： 如flag = NdCommplatform.GET_USER_BASE_INFO | NdCommplatform.GET_USER_POINT | NdCommplatform.GET_USER_EMOTION 则获取对应用户的基本信息,积分,心情。	

用户详细信息 NdUserInfo 通过数据通知回调返回。该接口的 NdCallbackListener 错误码为：

状态码（NdErrorCode 中定义）	含义

ND_COM_PLATFORM_SUCCESS	*成功*
ND_COM_PLATFORM_ERROR_SERVER_RETURN_ERROR	服务器处理发生错误，请求无法完成
ND_COM_PLATFORM_ERROR_USER_NOT_EXIST	该用户不存在
ND_COM_PLATFORM_ERROR_PERMISSION_NOT_ENOUGH	*权限不足*

调用举例：

```
NdCallbackListener<NdUserInfo> callbackUserInfo = new NdCallbackListener<NdUserInfo>(){
    @Override
    public void callback(int responseCode, NdUserInfo userInfo) {
        // TODO Auto-generated method stub
        switch(responseCode){
        case NdErrorCode.ND_COM_PLATFORM_SUCCESS:
            //查询成功
            break;
        case NdErrorCode.ND_COM_PLATFORM_ERROR_SERVER_RETURN_ERROR:
            //查询失败
            break;
        case NdErrorCode.ND_COM_PLATFORM_ERROR_USER_NOT_EXIST:
            //该用户不存在
            break;
        case NdErrorCode.ND_COM_PLATFORM_ERROR_PERMISSION_NOT_ENOUGH:
            //权限不足
            break;
        default :
            //查询失败
        }
    }
};
NdCommplatform.getInstance().ndGetUserInfoDetail("123456789",
        NdCommplatform.GET_USER_BASE_INFO
        | NdCommplatform.GET_USER_EMOTION
        | NdCommplatform.GET_USER_POINT, context, callbackUserInfo);
```

11) 捕获个人信息修改的通知

接口：

```
void setOnUserInfoChangeListener(
            NdMiscCallbackListener.OnUserInfoChangeListener listener)
```

当用户修改了头像时，可通过调用该通知设置获取修改信息； 回调信息通过实现：

```
void onUserInfoChanged(int value)
```

方法捕获，其中value值表示如下：

- OnUserInfoChangeListener.USER_ICON ：用户头像发生变化
- OnUserInfoChangeListener.USER_INFO ：用户信息发生变化
- OnUserInfoChangeListener.USER_ALL ：头像及用户信息均发生变化

8. 好友操作

1) 给好友发送消息(ndSendFriendMsg)

该接口只进行网络数据传输，不进入平台界面。
接口：

```
void ndSendFriendMsg(String uin, String msg, Context ctx,
            NdCallbackListener<String> callback)
```

其中 uin 为好友的 uin，msg 为消息内容(1-140 个字符)。消息发送成功后对应的消息 ID 通过数据通知回调返回。

该接口的 NdCallbackListener 状态码为：

状态码（NdErrorCode 中定义）	含义
ND_COM_PLATFORM_SUCCESS	*成功*
ND_COM_PLATFORM_ERROR_SERVER_RETURN_ERROR	服务器处理发生错误，请求无法完成
ND_COM_PLATFORM_ERROR_CONTENT_LENGTH_INVALID	内容长度不合法
ND_COM_PLATFORM_ERROR_EMOTION_CONTENT_INVALID	内容不合法
ND_COM_PLATFORM_ERROR_USER_NOT_EXIST	用户不存在
ND_COM_PLATFORM_ERROR_NOT_ALLOWED_TO_SEND_MSG	发送者被禁止发消息
ND_COM_PLATFORM_ERROR_CAN_NOT_SEND_MSG_TO_SELF	不能给自己发送短消息
ND_COM_PLATFORM_ERROR_PERMISSION_NOT_ENOUGH	权限不足

2) 添加/删除好友

如果想进行好友的“添加/删除”操作，可以调用该接口，进入指定用户的空间界面，那里有提供添加/删除好友的操作。该 uin 不能为空，不能是自己。

```
int ndEnterUserSpace(Context ctx, String uin)
```

使用方式请参见进入指定用户的空间接口。

9. 获取头像/图标

1) 简介

1.1)、获取头像和图标尺寸都可以指定分辨率大小：

NdIcon.ICON_DIMENSION_TINY　　16 * 16 像素
NdIcon.ICON_DIMENSION_SMALL　　48 * 48 像素
NdIcon.ICON_DIMENSION_MIDDLE　120 * 120 像素
NdIcon.ICON_DIMENSION_BIG　　200 * 200 像素

1.2)、开发者无需自己再做图片文件缓存。

因为对于头像和图标，平台里采用 checksum 统一的缓存机制。如果缓存图片的 checksum 与新的 checksum 比对不一致，则会删除缓存，重新从服务器下载新的图片并缓存；如果新的 checksum 与缓存 checksum 一致，或者新的 checksum 为空，那将使用缓存图片不再下载（如果初始没有缓存则需要从服务器下载并缓存）。对缓存的图片如果长期不再使用，那将会定期删除这些旧的图片，以免占用空间。

1.3)、关于 checksum

该参数一般是在获取用户的信息中包含该字段，用于及时更新用户的头像。该字段在应用信息，排行榜成就榜信息，虚拟商品信息等这些结构信息中一般都会包含 checksum，都是用来校验图标是否变化以便及时更新。如果开发者不关心图标的更新问题，在相应的 API 接口该字段可以传空。

2) 获取好友头像(默认大小)(ndGetPortrait)

本接口根据分辨率大小返回图片，分辨率大于 480 返回 120*120，否则返回 48*48。

接口：

```
boolean ndGetPortrait(String uin, String checksum, Context ctx,
            NdCallbackListener<NdIcon> callback)
```

返回的 boolean 如果为 true 表示图片从本地缓存获取，false 表示从网络获取。

返回的图片 NdIcon 通过数据通知回调返回。

该接口的 NdCallbackListener 状态码为：

状态码（NdErrorCode 中定义）	含义
ND_COM_PLATFORM_SUCCESS	*成功*
ND_COM_PLATFORM_ERROR_SERVER_RETURN_ERROR	服务器处理发生错误，请求无法完成
ND_COM_PLATFORM_ERROR_USER_NOT_EXIST	该用户不存在
ND_COM_PLATFORM_ERROR_NO_CUSTOM_PHOTO	没有自定义头像

3) 获取好友头像(自定义大小)(ndGetPortraitEx)

本接口用户可以根据图片四种尺寸类型根据自己的需求获取图片。

接口：

```
boolean ndGetPortraitEx(String uin, int imgType,String checksum, Context ctx,
            NdCallbackListener<NdIcon> callback)
```

返回的 boolean 如果为 true 表示图片从本地缓存获取，false 表示从网络获取。
返回的图片 NdIcon 通过数据通知回调返回。

参数：

- Uin　用户 UIN（ID）
- imgType　图片尺寸类型（参考上面图片简介）
- checksum　（参考上面图片简介）

返回的图片 NdIcon 通过数据通知回调返回。该接口的 NdCallbackListener 状态码为：

状态码（NdErrorCode 中定义）	含义
ND_COM_PLATFORM_SUCCESS	*成功*
ND_COM_PLATFORM_ERROR_SERVER_RETURN_ERROR	服务器处理发生错误，请求无法完成
ND_COM_PLATFORM_ERROR_USER_NOT_EXIST	该用户不存在
ND_COM_PLATFORM_ERROR_NO_CUSTOM_PHOTO	没有自定义头像

4) 获取好友头像路径(ndGetPortraitPath)

调用该接口直接 SDCard 获取头像的路径。
当 SDCard 时返回 null，这时用户可以通过

- R.drawable.nd3_default_portrait 对应的图片类型：
  NdIcon.ICON_DIMENSION_TINY
  NdIcon.ICON_DIMENSION_SMALL
- R.drawable.nd3_default_portrait_big 对应的图片类型：
  NdIcon.ICON_DIMENSION_MIDDLE
  NdIcon.ICON_DIMENSION_BIG

获取默认头像。
当获取图片路径失败时返回 SDCard 里面

- nd3_default_portrait.png（对应的图片类型：NdIcon.ICON_DIMENSION_TINY， NdIcon.ICON_DIMENSION_SMALL）
- nd3_default_portrait_big.png （对应的图片类型：NdIcon.ICON_DIMENSION_MIDDLE，NdIcon.ICON_DIMENSION_BIG）

的路径。
接口：

```
boolean ndGetPortraitPath(String uin, int imgType,String checksum, Context ctx,
            NdCallbackListener<String> callback)
```

参数：

- uin　用户 UIN（ID）
- imgType　图片尺寸类型（参考上面图片简介）
- checksum　（参考上面图片简介）

返回的图片路径通过数据通知回调返回。该接口的 NdCallbackListener 错误码为：

状态码（NdErrorCode 中定义）	含义
ND_COM_PLATFORM_SUCCESS	*成功*
ND_COM_PLATFORM_ERROR_ICON_NO_EXIST	图标不存在

5) 获取默认头像、默认应用图标 (ndGetDefaultPhoto)

当玩家未定义头像，或者系统未定义应用图标时，可根据本接口获取默认的头像或者应用图标，分辨率大于 480 返回 120*120，否则返回 48*48。

接口：

```
Bitmap ndGetDefaultPhoto(int type,Context ctx)
```

图片 Bitmap 直接从返回值获取。

其中 type 为：

状态码（NdCommplatform 中定义）	含义
DEFAULT_ICON_TYPE_HEAD	取默认头像
DEFAULT_ICON_TYPE_HEAD	取默认应用图标

6) 获取排行榜图标(ndGetLeaderboardIcon)

接口：

```
boolean  ndGetLeaderboardIcon(String leaderboardId,  String checksum,
                    int dimension,  Context ctx, NdCallbackListener<NdIcon> callback)
```

返回值 boolean 型为 true 表示从本地缓存取，false 为从网络取。

参数：

- leaderboardId　排行榜 ID
- dimension　图片尺寸类型（参考上面图片简介）
- checksum　（参考上面图片简介）

返回的图片 NdIcon 通过数据通知回调返回。该接口的 NdCallbackListener 状态码为：

状态码（NdErrorCode 中定义）	含义
ND_COM_PLATFORM_SUCCESS	*成功*
ND_COM_PLATFORM_ERROR_SERVER_RETURN_ERROR	服务器处理发生错误，请求无法完成
ND_COM_PLATFORM_ERROR_ICON_NO_EXIST	该图标不存在
ND_COM_PLATFORM_ERROR_ICON_NOT_CHANGED	图标没有变更
ND_COM_PLATFORM_ERROR_NO_CUSTOM_APP_ICON	无自定义图标

7) 获取成就榜图标(ndGetAchievementIcon)

接口：

```
boolean ndGetAchievementIcon(String achievementId, String checksum, int dimension,
                Context ctx, NdCallbackListener<NdIcon> callback)
```

返回值 boolean 型为 true 表示从本地缓存取，false 为从网络取。
参数：

- achievementId　成就榜 ID
- dimension　图片尺寸类型（参考上面图片简介）
- checksum　（参考上面图片简介）

返回的图片 NdIcon 通过数据通知回调返回。该接口的 NdCallbackListener 状态码为：

状态码（NdErrorCode 中定义）	含义
ND_COM_PLATFORM_SUCCESS	*成功*
ND_COM_PLATFORM_ERROR_SERVER_RETURN_ERROR	服务器处理发生错误，请求无法完成
ND_COM_PLATFORM_ERROR_ICON_NO_EXIST	该图标不存在
ND_COM_PLATFORM_ERROR_ICON_NOT_CHANGED	图标没有变更
ND_COM_PLATFORM_ERROR_NO_CUSTOM_APP_ICON	无自定义图标

8) 获取虚拟商品图标(ndGetProductIcon)

接口：

```
public boolean ndGetProductIcon(String productId, String checksum,int type,
            Context ctx, NdCallbackListener<NdIcon> callback)
```

返回值 boolean 型为 true 表示从本地缓存取，false 为从网络取。
参数：

- productId　虚拟商品 ID
- type　图片尺寸类型（参考上面图片简介）
- checksum　（参考上面图片简介）

返回的图片 NdIcon 通过数据通知回调返回。该接口的 NdCallbackListener 状态码为：

状态码（NdErrorCode 中定义）	含义
ND_COM_PLATFORM_SUCCESS	*成功*
ND_COM_PLATFORM_ERROR_SERVER_RETURN_ERROR	服务器处理发生错误，请求无法完成
ND_COM_PLATFORM_ERROR_ICON_NO_EXIST	该图标不存在
ND_COM_PLATFORM_ERROR_ICON_NOT_CHANGED	图标没有变更
ND_COM_PLATFORM_ERROR_NO_CUSTOM_APP_ICON	无自定义图标

9) 获取应用图标(ndGetAppIcon)

接口：

```
public boolean ndGetAppIcon (int appId, String checksum,int type,
            Context ctx, NdCallbackListener<NdIcon> callback)
```

返回值 boolean 型为 true 表示从本地缓存取，false 为从网络取。

参数：

- appId 应用 ID
- type　图片尺寸类型（参考上面图片简介）
- checksum　（参考上面图片简介）

返回的图片 NdIcon 通过数据通知回调返回。该接口的 NdCallbackListener 状态码为：

状态码（NdErrorCode 中定义）	含义
ND_COM_PLATFORM_SUCCESS	*成功*
ND_COM_PLATFORM_ERROR_SERVER_RETURN_ERROR	服务器处理发生错误，请求无法完成
ND_COM_PLATFORM_ERROR_ICON_NO_EXIST	该图标不存在
ND_COM_PLATFORM_ERROR_ICON_NOT_CHANGED	图标没有变更
ND_COM_PLATFORM_ERROR_NO_CUSTOM_APP_ICON	无自定义图标

10.　手机号绑定

1) 查询当前账号是否已绑定手机号

函数：

```
void ndHasBindPhoneNumber(Context ctx,  NdCallbackListener<Boolean> callback)
```

查询结果通过数据通知接口 NdCallbackListener<Boolean>返回：

- true：已绑定
- false：未绑定
- null：查询出错

数据通知接口通知返回的状态码为：

返回码（NdErrorCode中定义）	含义
ND_COM_PLATFORM_SUCCESS	成功
ND_COM_PLATFORM_ERROR_SERVER_RETURN_ERROR	服务器处理失败

2) 请求下发绑定手机号的短信验证码

函数：

```
Void ndSendBindSMSCode(String phoneNo,Context ctx,NdCallbackListener callback)
```

- phoneNo：待绑定的手机号码

数据通知接口通知返回的状态码为：

返回码（NdErrorCode中定义）	含义

ND_COM_PLATFORM_SUCCESS	成功
ND_COM_PLATFORM_ERROR_SERVER_RETURN_ERROR	服务器处理失败

3) 请求绑定手机号

函数：

```
void ndBindPhoneNumber(String verifyCode,
                String phoneNo,Context ctx, NdCallbackListener<String> callback)
```

- verifyCode：短信验证码
- phoneNo：待绑定的手机号码

绑定结果错误信息通过 NdCallbackListener<String>回调返回，如果绑定成功则返回的字符串为 null。

数据通知接口通知返回的状态码为：

返回码（NdErrorCode中定义）	**含义**
ND_COM_PLATFORM_SUCCESS	成功
ND_COM_PLATFORM_ERROR_SERVER_RETURN_ERROR	服务器处理失败
ND_COM_PHONENO_INVALID	手机号码格式无效
ND_COM_SMSCODE_ERROR	短信验证码错误
ND_COM_SMSCODE_EXPIRED	短信验证码过期
ND_COM_REBIND	重复绑定，账号已经绑定手机号
ND_COM_INCONSISTENT_BRFORE	手机号码前后不一致
ND_COM_HAS_BIND	手机号已经绑定其他账号

六、 版本号设定规则

用户软件版本更新是用于对您自己开发的软件进行版本检查和升级。版本更新是根据应用的 AndroidManifest.xml 文件中 manifest 节点属性 android:versionCode 来判断的，位置如图：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      android:versionCode="1"
```

该字段为 int 型，数字大的为高版本。

注意：版本更新升级是更新升级您自己所开发的软件而不是 91 移动开发平台的 SDK，版本更新升级的前提是您要有更高版本的应用放在 91 移动开发平台的应用中心。

七、 混淆

91SDK 包是以 jar 包及资源文件提供给用户的，其中 jar 包已经半混淆状态，您在混淆自己 APK 包的时候请不要将 91SDK 的 jar 包一起混淆，因为里面有些自定义 UI 控件，若被混淆后会因为无法找到相关类而抛异常。

您可以在用 ant 构建混淆包的 build.xml 里面对应位置或者在 proguard.cfg 里加入：

```
-libraryjars libs/*.jar
```

以避免 91SDK 的相关的 jar 包被混淆。

八、 FAQ

- **为什么同一框应用或游戏在我的机子会出现输入框被输入法覆盖，而别的机子不会。**

  对于某些机型可能会出现这样的问题，这个是机子本身的问题，用户可在 AndroidManifest.xml 增加：

```
<supports-screens android:anyDensity="true"
    android:smallScreens="true"
    android:normalScreens="true"
    android:largeScreens="true"/>
```

- **为什么我在调用带 NdCallbackListener 回调的 API 后，关闭界面有时候会报异常。**

  带回调的 API 一般都是异步的网络请求，请求完成后才会执行回调，如果您在回调执行前关闭界面而且在您的回调实现方法里有操作 UI 资源的话，那么有可能出现在执行回调的时候找不到资源。所以建议您在关闭界面的时候调用该回调的 destroy()方法销毁该回调。

- **为什么 91SDK 与我的工程代码一起混淆编译后运行会出现异常？**

  因为 91SDK 的 jar 里面有一部分代码为 UI 控件，如果被混淆后 layout 就无法找到对应的控件，以致运行出现异常。如果您需要混淆您的工程代码时请不要将 91SDK 的 NdComPlatform.jar 加进去混淆。如果您是用 Android SDK 自带的 proguard 混淆的话那么您可以在 proguard.cfg 添加以下代码：

```
-ignorewarning
-keep public class com.nd.commplatform.**
```

- **为什么按照上面的第 3 点将那两行代码放到 proguard.cfg 了，但在导出 apk 包时还会报异常，如下：**

```
Proguard returned with error code 1. See console
Warning: com.nd.commplatform.friend.B: can't find referenced class com.nd.commplatform.friend.B$com.nd.comm
Warning: there were 1 unresolved references to classes or interfaces.
         You may need to specify additional library jars (using '-libraryjars'),
         or perhaps the '-dontskipnonpubliclibraryclasses' option.
java.lang.ArrayIndexOutOfBoundsException: 2
  at proguard.classfile.editor.VariableRemapper.remapVariable(VariableRemapper.java:151)
  at proguard.classfile.editor.VariableRemapper.visitLocalVariableInfo(VariableRemapper.java:107)
  at proguard.classfile.attribute.LocalVariableTableAttribute.localVariablesAccept(LocalVariableTableAttril
  at proguard.classfile.editor.VariableRemapper.visitLocalVariableTableAttribute(VariableRemapper.java:82)
  at proguard.classfile.attribute.LocalVariableTableAttribute.accept(LocalVariableTableAttribute.java:63)
  at proguard.classfile.attribute.CodeAttribute.attributesAccept(CodeAttribute.java:199)
  at proguard.classfile.editor.VariableRemapper.visitCodeAttribute(VariableRemapper.java:75)
  at proguard.optimize.ParameterShrinker.visitCodeAttribute(ParameterShrinker.java:143)
  at proguard.classfile.attribute.CodeAttribute.accept(CodeAttribute.java:101)
  at proguard.classfile.ProgramMethod.attributesAccept(ProgramMethod.java:79)
```

这个是因为您的 Android SDK 下的 tools 下 proguard 版本比较旧，您可以下载个最新的 proguard 然后将里面的 lib 替换替换掉 Android SDK 下的 tools\proguard 的 lib。

- **调用自动登录接口后，还没等登录结果回调运行我就退出程序，有时候会抛异常，怎么处理？**

  您可以在您的程序退出时，如重写 Activity 的 finish()方法里调用 NdMiscCallbackListener.*setOnLoginProcessListener*(**null**); 方法将登录结果通知回调设置为 null。

- **如何查看当前程序所接入的 91SDK 版本？**

  在 91SDK3.1 版本之后，平台界面增加了查看所接的 SDK 版本等信息。查看方式：进入平台中心的“更多”界面，进入“关于我们”列表项，里面展示了所接 91SDK 的版本信息。

- **在非 UI 线程里调用 SDK 的函数出现如下异常如何处理：**

```
Uncaught handler: thread Thread-9 exiting due to uncaught exception
java.lang.RuntimeException: Can't create handler inside thread that has not called Looper.prepare()
```

您可以通过 Handler 获取 UI 线程执行 SDK，如：

```
new Handler(Looper.getMainLooper()).post(new Runnable(){

    @Override
    public void run() {
        // TODO Auto-generated method stub
        //调用SDK
    }

});
```

九、 场景案例

1. 游客登录

场景：

1. 玩家的设备未登录过 91 账号，并只是想先快速看下游戏好不好玩，再决定是否注册成为 91 用户继续玩。
2. 开发者想在玩家玩游戏的过程中通过 UIN 标识来保存玩家的游戏积分，等级等信息。
3. 开发者想在玩家处于游客登录状态且玩了 10 分钟后，或者玩家达到游戏等级 3 级时、或者玩家购买道具时、充值游戏币时再提醒玩家是否进行游客账户转正式账户。

流程图：