中国移动应用商场

应用内计费

开发指南

本文档主要描述了中国移动应用商场(MM)的应用内计费基本机制，以及指导开发者如何在应用中使用应用内计费功能。文档中提供了开发者需要做哪些准备、如何获取所需资源及如何使用 SDK 等的指引，同时也提供了相关的范例供开发者学习使用。

V 3.5.4

2017-09

目　　录

1. 开发应用.....1

1.1 准备开发环境.....1

1.2 下载和导入 SDK.....1

1.3 开发者开发应用注意事项.....2

2. 应用内计费 SDK 使用手册.....3

2.1 重载 Application 类，添加加载 sdk 代码.....3

2.2 关于 mmiap.xml 的说明（重要，重要，重要）.....4

2.2.1 获取方法.....4

2.2.2 集成方法.....5

2.3 SDK 组成和接口说明.....6

2.3.1 Purchase API 说明.....7

2.3.2 OnPurchaseListener.....10

2.3.3 OnPurchaseListener 中返回数据说明.....11

2.3.4 AndroidManifest 设置（开发者必须要注意的地方）.....12

2.4 SDK 调用时序.....14

2.4.1 构造及初始化.....14

2.4.2 查询.....15

2.4.3 订购.....15

2.5 示例代码与说明.....16

2.5.1 示例代码.....16

2.5.1.1 APPID，APPKEY，PayCode 设置.....16

2.5.1.2 OnPurchaseListener 接口实现.....16

2.5.1.3 SDK 初始化.....20

2.5.2 接口的调用.....22

2.5.2.1 订购接口调用.....22

2.5.2.2 查询接口调用.....23

2.5.2.3 退订接口调用.....23

2.5.3 获取渠道 ID.....24

Step1：读取 mmiap.xml 文件.....24

Step2：解析上面函数返回的 string，最终得到 channelid.....25

2.6 应用混淆.....26

3. 应用内计费中使用有数能力.....29

1. 开发应用

1.1 准备开发环境

在使用应用内计费接口之前，请确认 Eclipse、JDK、Android SDK 已经安装，并正常使用。如果尚未安装，请参考以下资源，安装过程不再赘述。

Eclipse：
http://www.eclipse.org/downloads/

JDK:
http://www.oracle.com/technetwork/java/javase/downloads/index.html

Android SDK:
http://developer.android.com/SDK/index.html

1.2 下载和导入 SDK

＊ SDK 可以在创建应用页面进行下载

SDK 以 jar 文件的形式提供给开发者在程序中使用，同时提供 HTML 格式的 API 文档供查阅相关类、方法、常量等说明。

以下内容将说明如何在 Eclipse 中，将 jar 文件加入到应用工程中去。

1. 将 lib/mmbilling3.5.4.jar 放到 libs；
2. 将 lib/armeabi/**.so 放到 libs/armeabi；
3. 由于运行时权限用到了 v4 包中的类，此版本需将 android.support-v4-library 放到 libs 下。
   a. v4 包版本需 23.0.1 及以上；
   b. 亦可依赖 android-v7-support-library，因为 v7 包括 v4
   c. v4 包请自行下载，或用 demo 工程 Libs 目录下的。

最后，检查 Private Libraries 中是否可以看到 jar 文件，如下图 1 所示。如果可以，则表示配置成功，否则，请检查上述步骤是否执行成功。

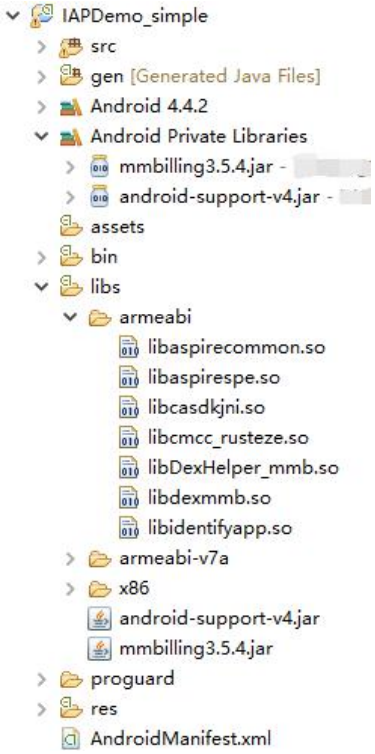


图 1

1.3 开发者开发应用注意事项

1) 本版计费 SDK 增加了安全机制，可能与常用的 App 加固产品不兼容，请开发者谨慎使用 App 加固。
2) 为保证自身敏感数据(APPID、APPKEY、PAYCODE)的安全性，请尽量采用加密等手段保存，避免以常量字符串形式出现于源码。
3) 为保护自身数据和付费点，开发完成后，建议对 APK 做混淆处理。
4) 应用中不能同时发起两起或者以上的订购操作，比如不能同时启动两个查询订单的线程。
5) **可批量购买的计费点，在两次订购之间，目前有时间限制(目前定义 30 秒钟)。**
6) **可批量购买的计费点，一次订购数量不能超过 10 个。**
7) 应用升级，开发如果需要升级目前已投入商用的 APP，需要重新上传后，用户通过 MM 商城或者其他与移动有关渠道升级。如果采用自升级，可能会因为 APP 数据与移动服务器中数据不一致，导致 APP 中无法正常发

起交易业务。

8) 中国移动部分省份已经开始销售 147 号段的 SIM 卡，该号段同样可以使用 IAP 进行计费。请应用开发者注意判断此号段的 SIM 卡，以免造成不必要的麻烦。

9) 如果应用中同时有 armeabi 和 armeabi-v7a 等多个文件夹，请将计费需要的 so 库同时添加到这两个文件夹中。如果存在 x86 文件夹，操作方式同上。

10) 升级 SDK 时，是否能仅升级 jar？不行，sdk 包括 jar 和 so，在使用新版本时，请同时替换 so 库。

11) 从 3.5.2 版本开始 SDK 不区分强弱联网，原先的弱联网计费点将以强联网渠道优先，弱联网渠道为补充的原则发起计费订购，强联网计费点不受影响。

12) 关于 mmiap.xml 文件需要用户自己在开发者网站上下载并集成到 apk 中，具体方法见本文 2.2 节。

13) SDK3.5 (3.5.0，3.5.1，3.5.2)版本的 mmiap.xml 可共用，3.5 版本和 3.1.9，3.1.10 版本的 mmiap.xml 不可共用。

14) 此版本发起订购请求时，需要透传交易流水号，流水号长度为小于或等于 64 位字符串。

15) 此版本需要开发者在调用 SDK 前，获取手机存储权限，以免影响正常功能使用。

16) 此版本去除支付宝 SDK，支付宝支付方式改为 H5 页面支付。

2. 应用内计费 SDK 使用手册

SDK 组成和接口说明

2.1 重载 Application 类，添加加载 sdk 代码

文件内容如下：

```
package com.aspire.demo;//你自己的包名

import android.app.Application;

import com.secneo.mmb.Helper;
```

```
import android.content.Context;

public class MyApplication extends Application {

    protected void attachBaseContext(Context ctx) {

        super.attachBaseContext(ctx);

        Helper.install(this);

    }

}
```

说明：MyApplication 是你自定义的名称。此处以 MyApplication 举例.

2.2 关于 mmiap.xml 的说明（重要，重要，重要）

本版本的自测试流程与 mmiap.xml 文件密切相关，自测试流程也需要集成对应版本的 mmiap.xml 文件到程序包中。 sdk 本身不集成 mmiap.xml 文件，因此需要 AP 自行在开发者网站获取。

ProgramId 对应着下一个自测试包的 ID, channel 为各个分发的渠道默认为 0000000000。获取后集成进 apk 即可进行自测试。将程序包上传申请商用并通过测试后，此文件中对应该的 ProgramId 即为商用的 ID，将不能再用于自测试，若需要再进行自测试的，需要重新获取 mmiap.xml 文件。

此 mmiap 文件不可与 3.1.10 及以前的共用。

2.2.1 获取方法

此版本的 mmiap.xml 文件需要自行从开发者网站获取。获取方式如下：

如果您是添加新应用：

开发者社区网站->登录->管理中心->应用维护->能力配置->下一步->拉到下面配置服务器接口那里选择 3.5 及以后版本的 SDK.

如果您是在已有的应用上添加的新的程序包:

1. 进入上传程序包页面，点击添加程序包

2. 进入页面，“使用 SDK 版本”选择“**3.5 及以后版本的 SDK**”后，点击下载配置文件的超链接即可下载新的 mmiap.xml 文件

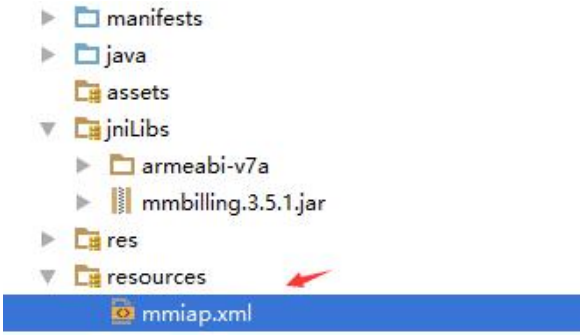
2.2.2 集成方法

mmiap.xml 文件放到 apk 根目录下，与 AndroidManifest.xml 同级。如下图所示：

具体方法有以下二种：

1、eclipse 工程中可放入 src 文件夹下，mmiap 文件需要在最终你生成的 apk 解压后的根目录下(非工程目录根目录),如下图。

2、AndroidStudio 工程放入 resources 文件夹中。

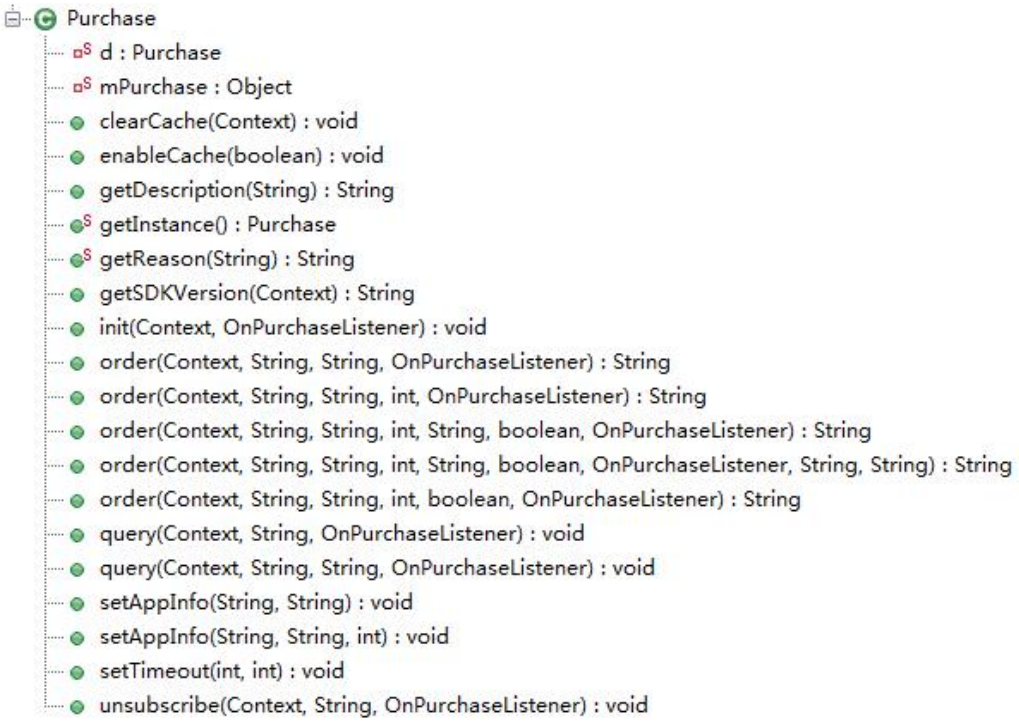


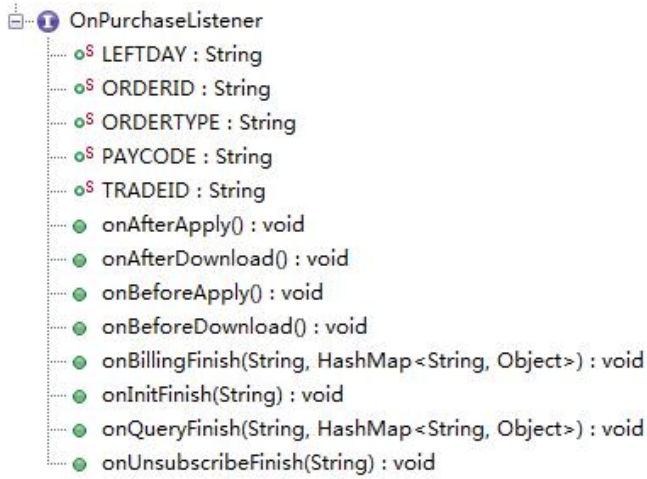
Build.gradle 中如果没有以下配置，在 src/main/下添加 resources 文件夹并将 mmiap.xml 放入；有的话则将 mmiap.xml 放入 src 目录下。

```
sourceSets {
    main {
        manifest.srcFile 'AndroidManifest.xml'
        java.srcDirs = ['src']
        resources.srcDirs = ['src']
        aidl.srcDirs = ['src']
        jniLibs.srcDirs = ['libs']
        renderscript.srcDirs = ['src']
        res.srcDirs = ['res']
        assets.srcDirs = ['assets']
    }
```

2.3 SDK 组成和接口说明

SDK 公开的接口包括：





2.3.1 Purchase API 说明

Purchase 对象是 SDK 提供给开发者发起订购，查询的接口。

开发者在实例化该对象后，调用其中的函数可以处理相应的业务。

1) 构造实例：

Purchaser 对象的创建使用了单例模式，不需要重复创建：

```
purchase = Purchase.getInstance();
```

2) 各参数设置：

Purchase.setAppInfo(appid, appkey); // 设置计费应用 ID 和 Key (必须)

Purchase.setTimeout(15000, 15000); // 设置超时时间(可选)，可不设置，缺省都是 15s

3) 初始化：

Init()，初始化函数，必须调用 1 次且仅 1 次。此函数主要实现用户身份数字证书申请，**开发者在 APP 初始化中调用**，这样可减少用户在订购，查询业务中的等待时间。

Purchase.init(context, listener); //初始化，传入监听器

调用后，请等待 onInitFinish()完成后，再发起其他业务请求，否则其他业务不会处理。

4) 订购：

调用 Purchase 对象中的 order 函数，一共有 4 种，传入相应的参数：

- payCode，计费点。
- **tradID，交易流水号。此版本发起订购请求时，需要透传交易流水号，流水号长度为小于或等于 64 位字符串，必须数字和字母。**
- orderCount,订购数量。（包月、约定租期和不可重复订购计费点只能传入 1，可重复订购计费点可以传入 10 以下数值）
- nextCycle，对于租赁类业务，可以预订下一期租赁周期。
- Listener，本参数是开发实现 OnPurchaseListener 对象的实例，主要用于监听各个业务的结果。
- **data，本参数是可透传到开发者服务器的自定义数据，长度：所有计费点必须在 16 位以内。必须数字和字母。**

```
// 订购一个商品
Purchase.order(context, paycode, tradID,listener);
// 订购 5 个商品
Purchase.order(context, paycode, tradID,5, listener);
```

// 租赁当前周期

Purchase.order(context, month_paycode, tradID,1, false[true], listener);

// 调用购买接口并传入自定义数据

Purchase.order(context, paycode, tradID,1, data, true, listener);

订购具体的结果在 OnPurchaseListener 中的 onBillingFinish()中获得,可以在查询接口中传入 onBillingFinish()返回的交易 ID 查询交易有无成功.

5) 查询：

调用 Purchase 对象中的 query 函数，传入相应参数：

- payCode，计费点
- TradeID，调用 order()接口返回的交易 ID，用于查询交易是否成功
- Listener，本参数是开发实现 OnPurchaseListener 对象的实例，主要用于监听各个业务的结果。

// 查询单次或者租赁类商品是否订购成功

Purchase.query(context, paycode, listener);

// 带交易 ID 查询重复类商品是否交易成功

Purchase.query(context, paycode, tradID, listener);

6) 退订：

调用 Purchase 对象中的 unsubscribe 函数，传入相应参数：

- payCode，计费点
- Listener，本参数是开发实现 OnPurchaseListener 对象的实例，主要用于监听各个业务的结果。

// 退订包月业务

Purchase.unsubscribe(context, Paycode, Listener);

注意：目前只有包月业务允许退订。其他类型业务均不允许退订。

7) 获取 SDK 的版本号：

调用 Purchase 对像中的 getSDKversion 函数,可以获取当前的 SDK 的版本号,传入参数:

- Context，Activity 的实例

Purchase.getSDKVersion(Context context) throws Exception

8) 获取状态码：

8.1 调用 Purchase 对像中的 getDescription 函数,获取对应状态码的描述信息,传入参数:

- code，OnPurchaseListener 返回的错误码

Purchase.getDescription(String code)

8.2 调用 Purchase 对像中的 getReason 函数,获取该状态码对应的描述信息,传入参数:

- code，OnPurchaseListener 返回的状态码

Purchase.getReason(String code)

2.3.2 OnPurchaseListener

应用内计费各种操作(查询，订购)监听器。开发者通过实现该接口中各个接口来监听各种业务操作的状态:

```
// 初始化返回接口
void onInitFinish(final String returnCode)
// 查询返回接口
void onQueryFinish(final String returnCode, final HashMap<String, Object> returnObject)
// 订购返回接口:
void onBillingFinish(final String returnCode, final HashMap<String, Object> returnObject)
//退订返回接口：
public void onUnsubscribeFinish(final String returnCode)
```

Returncode 的定义在 PurchaseCode 类中，具体含义可以通过

getDescription()获取。

2.3.3 OnPurchaseListener 中返回数据说明

正如前面所描述的一样，初始化，查询，订购接口的返回值在 OnPurchaseListenr 中得到。

returnObject 中定义的数据主要有几种：

```
// 订单号
public final static String ORDERID = "OrderId";
// 计费点代码
public final static String PAYCODE = "Paycode";
// 租赁剩余时间
public final static String LEFTDAY = "LeftDay";
// 交易ID
public final static String TRADEID = "TradeID";
//订购类型
public final static String ORDERTYPE = "OrderType";
```

上面这些值包含在 onXXFinish 中参数 Hashmap 中。各个接口返回的数据不一样。

上面这些值所代表的意义如下：

OrderId，表示此次订单，mm 平台形成的订单流水号

Paycode，表示此次交易的商品 id

LeftDay，表示此次交易商品的有效期。

TradeID，表示此次交易的交易 ID，供查询用。

OrderType，表示此次交易的类型。如果返回 0，则表示是生成测试订单；如果返回 1，则表示生成正式订单。

1) 初始化接口

初始化接口不返回任何数据。

2) 订购接口

订购接口在订购成功后，onBillingFinish()返回上面 5 个值。如果订购失败，则不返回上面任何值。

3) 查询接口

查询接口在查询成功后，返回上面的 OrderId，Paycode ,LeftDay 这三个值。失败则不返回任何值。

2.3.4 AndroidManifest 设置（开发者必须要注意的地方）

此版本需要开发者在 AndroidManifest.xml 中增加如下声明。

1) **增加权限声明**

```
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission
android:name="android.permission.READ_PHONE_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.ACCESS_COARSE_LOCATION' />
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission
android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION' />
```

2）增加 application 声明(参考文档最下面的 4 点，派生 Application 类)

```
<application
        android:name="com.aspire.demo.MyApplication"
        android:icon="@drawable/icon"
```

```
        android:label="@string/app_name" >
        <activity
            android:name="com.aspire.demo.Demo"
            android:configChanges="orientation|screensize"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <!-- 6.0 运行时权限所需 -->
            <activity
android:name="mm.purchasesdk.core.PermissionsGrantActivity" />
    </application>
```

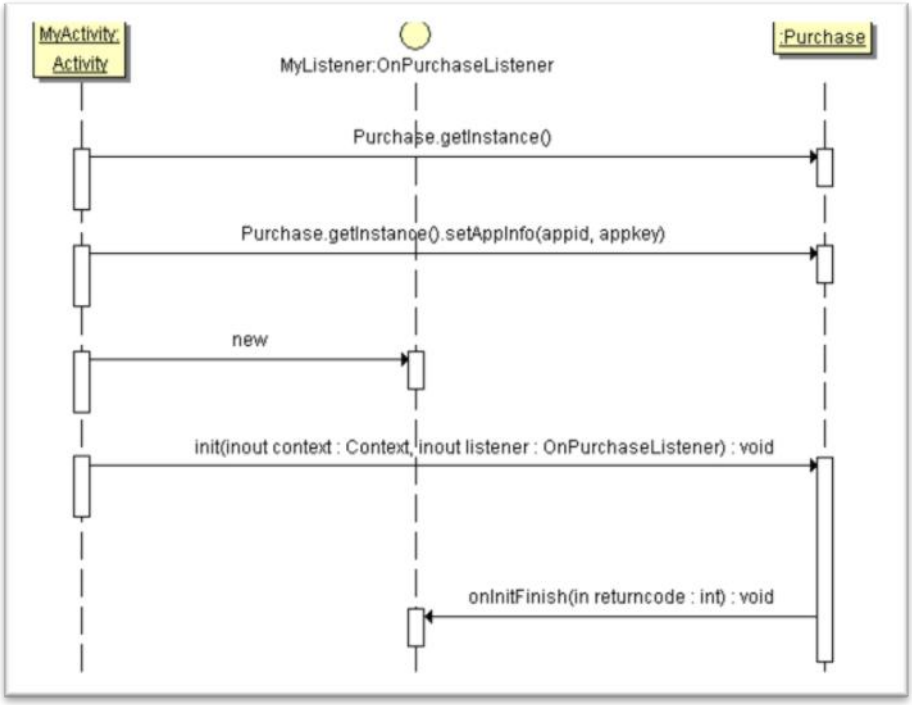
说明：

1、MyApplication 是你自定义的名称。此处以 MyApplication 举例；

2、6.0 以上机器获取运行时权限需要配置 PermissionsGrantActivity。

2.4 SDK 调用时序

2.4.1 构造及初始化

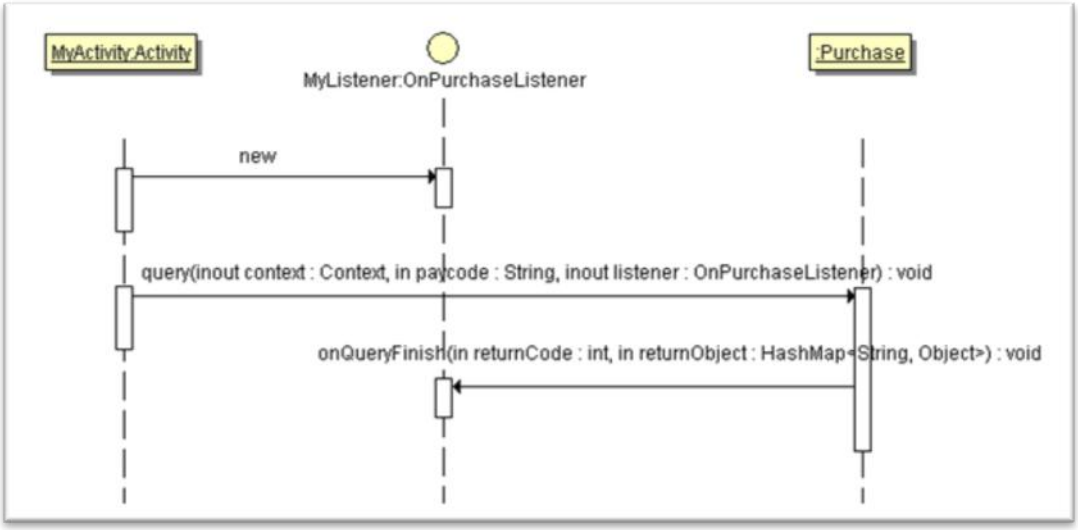


注意：

1．**开发者需要在游戏或者程序启动时调用初始化接口 init（），比如界面进入时，调用初始化方法 init()，**这样可以避免在计费过程中申请用户身份数字证书，以节省一些时间

2．一旦调用了初始化方法 init()，在 init()没有完成之前，调用其他的业务接口(查询、订购、退订)是不允许的，必须先调 init()一次且仅 1 次即可。所以应用最好在监听器中等到初始化结束(不管成功失败)，再允许订购。

2.4.2 查询

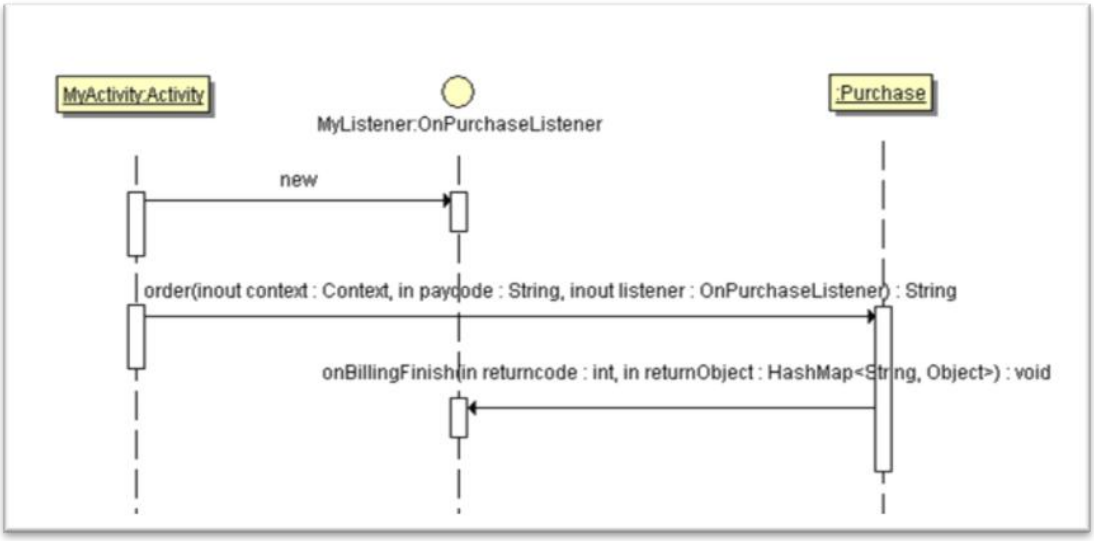


注意：

1. 由于版权文件已被弃用，所以接口类 OnPurchaseListener 中与版权文件相关的接口都被停用，即以下接口：onBeforeDownload，onAfterDownload，onBeforeApply，onAftreApply。

2. 查询到已成功的订单，可以拿到 orderID

2.4.3 订购



注意：

1. 本期 SDK 在触发订购之后，接管全部 UI，所以应用在调用 order 接口之

后不需要做任何处理，所有 UI 处理均已由 SDK 处理。

2. onBillingFinish()方法在订购界面退出后，才会返回给应用。

3. 鉴权成功或者订购成功，均会返回 OrderID

2.5 示例代码与说明

下面将通过 demo 中的代码具体说明如何使用本 SDK。

2.5.1 示例代码

2.5.1.1 APPID，APPKEY，PayCode 设置

在 demo 程序中，APPID，APPKEY，PayCode 默认赋值为 00000000000。请根据在 2.3 节中操作，将生成的 APPID，APPKEY，PayCode 分别填入。

```
// 计费应用信息(demo 测试)
private static final String APPID = "00000000000";//填入你的 appid
private static final String APPKEY = "00000000000";// 填入你的 appkey
// 计费点信息
private static final String LEASE_PAYCODE = "00000000000"; // 单次，填入你的 paycode
```

2.5.1.2 OnPurchaseListener 接口实现

开发者使用 SDK，需要自定义实现一个接口 OnPurchaseListener，该接口用来将订购，查询，退订各业务的结果或者状态返回给 APP。Demo 中具体实现如下：

```
public class IAPListener implements OnPurchaseListener {

private final String TAG = "IAPListener";
    private Demo context;
```

```
    private IAPHandler iapHandler;

public IAPListener(Context context, IAPHandler iapHandler) {
        this.context = (Demo) context;
        this.iapHandler = iapHandler;
    }
    @Override
    public void onAfterApply() {}
     @Override
    public void onAfterDownload() {}
    @Override
    public void onBeforeApply() {}

    @Override
    public void onBeforeDownload() {}

    @Override
    public void onInitFinish(String code) {
        Log.d(TAG, "Init finish, status code = " + code);
        Message message = iapHandler
.obtainMessage(IAPHandler.INIT_FINISH);
        String result = "初始化结果：" + Purchase.getReason(code);
        message.obj = result;
        message.sendToTarget();
    }

    @Override
    public void onBillingFinish(String code, HashMap arg1) {
        Log.d(TAG, "billing finish, status code = " + code);
        String result = "订购结果：订购成功";
        iapHandler.obtainMessage(IAPHandler.BILL_FINISH);
        // 此次订购的 orderID
        String orderID = null;
        // 商品的 paycode
```

```
            String paycode = null;
            // 商品的有效期(仅租赁类型商品有效)
            String leftday = null;
            // 商品的交易 ID，用户可以根据这个交易 ID，查询商品是否已经
交易
            String tradeID = null;
            String ordertype = null;
            if ( PurchaseCode.BILL_ORDER_OK.equalsIgnoreCase( code )
                    || PurchaseCode.AUTH_OK.equalsIgnoreCase( code )
                    ||
PurchaseCode.WEAK_ORDER_OK.equalsIgnoreCase( code ) ) {

                /**
                 * 商品购买成功或者已经购买。 此时会返回商品的 paycode，
orderID,以及剩余时间(租赁类型商品)
                 */
                if (arg1 != null) {
                    leftday                    =                    (String)
arg1.get(OnPurchaseListener.LEFTDAY);
                    if (leftday != null && leftday.trim().length() != 0) {
                        result = result + ",剩余时间 ： " + leftday;
                    }
                    orderID                    =                    (String)
arg1.get(OnPurchaseListener.ORDERID);
                    if (orderID != null && orderID.trim().length() != 0) {
                        result = result + ",OrderID ： " + orderID;
                    }
                    paycode                    =                    (String)
arg1.get(OnPurchaseListener.PAYCODE);
                    if (paycode != null && paycode.trim().length() != 0) {
                        result = result + ",Paycode:" + paycode;
                    }
                    tradeID                    =                    (String)
arg1.get(OnPurchaseListener.TRADEID);
```

```
                    if (tradeID != null && tradeID.trim().length() != 0) {
                        result = result + ",tradeID:" + tradeID;
                    }
                    ordertype = (String)
arg1.get(OnPurchaseListener.ORDERTYPE);
                    if (tradeID != null && tradeID.trim().length() != 0) {
                        result = result + ",ORDERTYPE:" + ordertype;
                    }
                }
            } else {
                //表示订购失败。
                result = "订购结果：" + Purchase.getReason(code);
            }
            context.dismissProgressDialog();
            System.out.println(result);
        }

        @Override
        public void onQueryFinish(String code, HashMap arg1) {
            Log.d(TAG, "license finish, status code = " + code);
            iapHandler.obtainMessage(IAPHandler.QUERY_FINISH);
            String result = "查询成功,该商品已购买";
            // 此次订购的 orderID
            String orderID = null;
            // 商品的 paycode
            String paycode = null;
            // 商品的有效期(仅租赁类型商品有效)
            String leftday = null;
            if (code.compareTo(PurchaseCode.QUERY_OK) != 0) {
                //查询不到商品购买的相关信息/
                result = "查询结果：" + Purchase.getReason(code);
            } else {
                //查询到商品的相关信息。此时你可以获得商品的 paycode，
orderid，以及商品的有效期 leftday（仅租赁类型商品可以返回）
```

```
            leftday = (String) arg1.get(OnPurchaseListener.LEFTDAY);
            if (leftday != null && leftday.trim().length() != 0) {
                result = result + ",剩余时间 ： " + leftday;
            }
            orderID = (String) arg1.get(OnPurchaseListener.ORDERID);
            if (orderID != null && orderID.trim().length() != 0) {
                result = result + ",OrderID ： " + orderID;
            }
            paycode = (String) arg1.get(OnPurchaseListener.PAYCODE);
            if (paycode != null && paycode.trim().length() != 0) {
                result = result + ",Paycode:" + paycode;
            }
        }
        System.out.println(result);
        context.dismissProgressDialog();
    }

    @Override
    public void onUnsubscribeFinish(String code) {
        String result = "退订结果：" + Purchase.getReason(code);
        System.out.println(result);
        context.dismissProgressDialog();
    }
```

2.5.1.3 SDK 初始化

本 SDK 初始化很简单，只需要实例化 SDK 中 Purchase 类即可，再根据 APP 的实际情况设置相应的参数。Demo 中的代码如下：

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        //IAP组件初始化.包括下面3步。
        /**
        * step1.实例化PurchaseListener。实例化传入的参数与您实现
PurchaseListener接口的对象有关。
        * 例如，此Demo代码中使用IAPListener继承PurchaseListener，其
```

```
构造函数需要Context实例。
            */
            listener = new IAPListener(this, iapHandler);
            /**
            * step2.实例化Purchase对象。在实例化Purhcase对象后，必须为
purchase实例setAppInfo
            *接口。该接口函数需要传入APPID，APPKEY。
            */
            purchase = Purchase.getInstance();
            try {
                Purchase.setAppInfo(APPID, APPKEY);
            } catch (Exception e1) {
                e1.printStackTrace();
            }
            /**
            * step3.IAP组件初始化开始，
            * 参数PurchaseListener，初始化函数需传入step1时实例化的
onPurchaseListener。
            */
            Purchase.init(listener);
        }
        private void showProgressDialog(String text) {
            if (mProgressDialog == null) {
                mProgressDialog = new ProgressDialog(Demo.this);
                mProgressDialog.setIndeterminate(true);
                LayoutInflater inflater = getLayoutInflater();
                View view = inflater.inflate(R.layout.layout, null);
                mProgressDialog.setView(view);
                mProgressDialog.setMessage("请稍后.....");
            }
            if (!mProgressDialog.isShowing()) {
                mProgressDialog.show();
            }
        }
```

Demo 中 Purhcase 类实例化后，设置了网络超时和 SDK 的 init 接口。

如果不设置网络超时，则默认网络超时为 15s，建议按默认值，如果设置的话，建议设置 7s 以上，15s 以内。如果不调用 init 接口，则用户在订购解密将等待较长时间，建议在 APP 初始化或者数据加载过程中调用此函数。

2.5.2 接口的调用

2.5.2.1 订购接口调用

订购分为租赁，永久性购买，可重复购买这三种类型的订购，各类型根据 paycode 来区分。此版本应用内计费 SDK 包含有 4 个订购接口

1. 简化版订购接口

public String order(Context context, String paycode, OnPurchaseListener listener)

需要传入 Activity 实例，paycode，以及 OnPurchaseListener 的实例，此接口可以订购单件商品。而且此接口将会返回此次交易的交易 ID。用户或者开发者可以通过此交易 ID 去查询交易是否成功。

注：此接口不能用作租赁类型的续订接口。

2. 一般性订购接口

public String order(Context context, String paycode, int orderCount,OnPurchaseListener listener)

此接口和简化版接口相比，增加了一个 int 型参数，也就是指此接口支持一次订购多件商品。

3. 完整版订购接口

public String order(Context context, String paycode, int orderCount,boolean nextCycle, OnPurchaseListener listener)

此接口和一般性接口相比，增加了一个 boolean 型参数，也就是指此接口支持商品续订。可以用于租赁类型商品的续订。

4. 用户可透传数据的订购接口

public String order(Context context, String paycode, int orderCount,String data, boolean nextCycle, OnPurchaseListener listener)

此接口增加了一个 String 型参数，是给用户透传到服务器的自定义数据，字符串数据类型，中间不能有空格。

2.5.2.2 查询接口调用

此版本应用内计费的查询分为两个接口。一个是根据交易 ID 查询，查询该交易是否存在。另一个是根据 paycode 查询，查询当前商品是否已经订购。

1. 根据交易 ID 查询

public void query(Context context, String paycode, String tradeID, OnPurchaseListener listener)

使用交易 ID 查询，主要目的是查询该笔交易是否成功。例如，如果您记录以前某次交易的交易 ID，您可以根据此接口查询是否交易成功。

2. 根据 paycode 查询

public void query(Context context, String paycode, OnPurchaseListener listener)

此版本与之前版本接口保持一致，用于查询当前商品状态是否已经订购。

注意：可重复计费的商品订购成功 30 秒以后使用此查询接口，将查不到已订购状态。

2.5.2.3 退订接口调用

根据计费点进行退订，仅对包月类型计费点有效

public void unsubscribe(Context context, String paycode,

OnPurchaseListener listener)

2.5.3 获取渠道 ID

应用内计费 sdk 中包含有渠道配置文件 mmiap.xml。开发者可以读取这个文件取得当前程序包的渠道 id。

调用方法如下：

Step1：读取 mmiap.xml 文件

```
private static final String CHANNEL_FILE = "mmiap.xml";
public static String getResFileContent(String filename, Context context) {
    InputStream is = context.getClass().getClassLoader()
            .getResourceAsStream(filename);

    if (is == null) {
        return null;
    }

    StringBuilder builder = new StringBuilder();
    String content = "";
    BufferedReader bufferedReader = new BufferedReader(
            new InputStreamReader(is));
    try {
        while (bufferedReader.ready()) {
            content = bufferedReader.readLine();
            builder.append(content);
        }
        bufferedReader.close();
    } catch (IOException e) {
        return null;
    }
    return builder.toString();
```

```
    }
```

Step2：解析上面函数返回的 string，最终得到 channelid

```
    public static String LoadChannelID(Context context) {
        if (channelID != null) {
            return channelID;
        }

        // 载入渠道字符串
        String channleStr = getResFileContent(context, CHANNEL_FILE);

        // 解析文件
        XmlPullParserFactory factory;
        try {
            factory = XmlPullParserFactory.newInstance();
            XmlPullParser parser = factory.newPullParser();
            byte[] data = channleStr.getBytes();
            ByteArrayInputStream bais = new
ByteArrayInputStream(data);

            parser.setInput(bais, "utf-8");
            int event = parser.getEventType();
            while (event != XmlPullParser.END_DOCUMENT) {
                switch (event) {
                    case XmlPullParser.START_DOCUMENT:
                        break;
                    case XmlPullParser.START_TAG:
                        String tag = parser.getName();
                        if ("channel".equals(tag)) {
                            channelID = parser.nextText();
                        }
                        break;
                    case XmlPullParser.END_TAG:
```

```
                    break;
                }
                event = parser.next();
            }
        }
        catch (XmlPullParserException e) {
            channelID = null;
            return null;
        }
        catch (IOException e) {
            channelID = null;
            return null;
        }
        return channelID;
        }
```

2.6 应用混淆

如果您的应用需要混淆，请参照下面的格式修改 proguard.cfg。然后使用 ProGuard 工具对您的应用进行混淆。

```
-optimizationpasses 5
-dontusemixedcaseclassnames
-dontskipnonpubliclibraryclasses
-dontpreverify
-dontwarn
-dontnote
-verbose
-optimizations !code/simplification/arithmetic,!field/*,!class/merging/*
-libraryjars libs/mmbilling.jar

-keep public class * extends android.app.Activity
-keep public class * extends android.app.Application
-keep public class * extends android.app.Service
```

```
-keep public class * extends android.content.BroadcastReceiver
-keep public class * extends android.content.ContentProvider
-keep public class * extends android.app.backup.BackupAgentHelper
-keep public class * extends android.preference.Preference
-keep public class com.android.vending.licensing.ILicensingService

-keepclasseswithmembernames class * {
    native <methods>;
}
-keepclasseswithmembernames class * {
    public <init>(android.content.Context, android.util.AttributeSet);
}

-keepclasseswithmembernames class * {
    public <init>(android.content.Context, android.util.AttributeSet, int);
}

-keepclassmembers enum * {
    public static **[] values();
    public static ** valueOf(java.lang.String);
}

-keep class * implements android.os.Parcelable {
  public static final android.os.Parcelable$Creator *;
}
-keep class mm.purchasesdk.**
-keep class mm.purchasesdk.** {*;}
-keep class com.secneo.mmb.**
-keep class com.secneo.mmb.** {*;}
-keep class com.chinaMobile.**
-keep class com.chinaMobile.** {*;}
```

具体修改如下，如果要混淆某个包，则在 proguard.cfg 中增加:

–keep class 应用某个包名.**

注意：您在混淆时，请务必在 cfg 文件中添加红色的代码，否则可以会导致您的程序运行出错。蓝色部分的代码，如果您使用 eclipse 工具混淆则需要添加，如果是 ant 编写的混淆脚本则不需要添加。

如果您使用 ant 脚本编译，可以参照 build.xml 中的混淆脚本修改您的脚本

```
    <target name="optimize" depends="compile">
        <echo>optimize classes are put to
"${out.absolute.dir}"     .</echo>
        <jar basedir="${out.classes.absolute.dir}"
destfile="${out.absolute.dir}/temp.jar"/>
       <java jar="${jar.proguard}\proguard.jar" fork="true"
    failonerror="true">
            <jvmarg value="-Dmaximum.inlined.code.length=32"/>
            <arg value="@${jar.proguard}\proguard.cfg"/>
            <arg value="-injars ${out.absolute.dir}/temp.jar"/>
            <arg value="-outjars ${out.absolute.dir}/optimized.jar"/>
            <arg value="-libraryjars
    ${SDK.dir}/platforms/android-4/android.jar"/>
            <arg value="-libraryjars ./libs/mmbilling.jar"/>

       </java>
        <delete file="${out.absolute.dir}/temp.jar"/>
        <delete dir="${out.classes.dir}" failonerror="false" />
        <mkdir dir="${out.classes.dir}"/>
        <unzip src="${out.absolute.dir}/optimized.jar"
dest="${out.classes.absolute.dir}"/>
        <delete file="${out.absolute.dir}/optimized.jar"/>
    </target>
```

Proguard 的路径在 local.properties 中进行修改。Demo 中有个文件夹 proguard 中存放的是 proguard.jar 文件和 proguard.cfg。

3. 应用内计费中使用有数能力

1、有数（http://dev.10086.cn/datau/）是中国移动开发者社区提供的数据采集和分析能力，通过嵌入应用中的有数 SDK，可采集用户使用应用的相关数据，并通过有数平台提供给开发者应用分析服务。

2、应用内计费 SDK 2.2 以上版本已默认经嵌入了有数 SDK，在用户使用应用内计费的过程中，有数 SDK 已经自动采集应用启动、计费流程等基础数据。

3、如需使用有数平台提供的其他数据采集和分析功能（例如用户终端信息、页面路径信息、自定义事件分析、错误事件分析等），可参考有数平台相关介绍和有数开发指南。

4、有数平台：http://dev.10086.cn/datau/

5、有数开发指南：

http://dev.10086.cn/datau/modules/views/introduction/index.html?menu=sdk