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 3.5.10

2018-03

目　　录

1. 准备开发与集成

1.1 准备开发环境

在使用应用内计费接口之前，请确认 Eclipse/Android Studio、JDK、Android SDK 已经安装，并可正常使用。如果尚未安装，请自行下载资源安装。

1.2 下载 SDK 资源

1.2.1 应用内计费 SDK 下载流程

参考步骤：

(1) 中国移动开发者社区（http://dev.10086.cn）首页中查找 “能力平台” 模块，并选择 “应用内计费能力” 进入应用内计费 SDK 页面；

(2) 在应用内计费 SDK 页面中，查找 “下载专区” ，即可下载当前最新 SDK 版本与对应开发者指南。

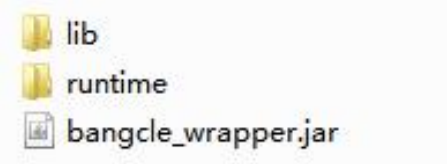
1.2.2 SDK 资源包组成

SDK 以 jar 文件的形式提供给开发者在程序中使用，同时提供 pdf 格式的开发者指南文档供查阅相关类、方法、常量等使用说明。

资源压缩包内容：

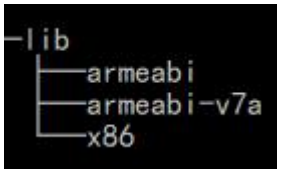
(1) **mmbillingxx.xx.xx**：提供开发者集成的 SDK jar 文件与 So 文件；

mmbillingxx.xx.xx 文件内容：



a. lib：提供开发者集成的 so 文件；

lib 文件内容（包括 armeabi、armeabi-v7a、x86 三种指令集）：



b. bangcle_wrapper.jar：提供开发者集成的 SDK jar 文件；

c. runtime/sdk.jar：提供开发者集成的 SDK jar 文件；

(2) **IAP_Eclipse**：提供开发者学习集成使用的 eclipse 结构 Demo 工程；

(3) **IAP_AS**：提供开发者学习集成使用的 AS 结构 Demo 工程；

(4) **ReadMe.txt**：当期应用内计费 SDK 更新说明、重点注意事项；

(5) **应用内计费-开发指南 xx.xx.xx.pdf：**本文档，帮助开发者集成应用内计费 SDK 开发者指南

1.3 开发注意事项

1. 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. 建议将 AndroidMannifest.xml 文件中的 android:allowBackup 属性值设为 false。
5. 应用中不能同时发起两次或以上的订购操作，比如不能同时启动两个查询订单的线程。

6. 可批量购买的计费点，在两次订购之间有时间间隔限制，(目前定义 30 秒钟)。

7. 可批量购买的计费点，一次订购数量不能超过 10 个。

8. 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。

9. 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。

10. 如果应用中同时有 armeabi 和 armeabi-v7a 等多个指令集文件夹，请将计费 SDK 需要的 so 文件同时添加到这两个文件夹中。

11. 升级更新 SDK 时，不可仅更新 SDK jar 文件，应同时更新替换 jar 和 so 文件。在使用新版本 SDK 时，请注意 so 文件变更，移除废弃 so 文件，再更新替换 so 库。

12. 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。

13. 关于 mmiap.xml 文件需要开发者在中国移动开发者社区管理中心上下载并集成到 apk 中，具体方法见本文 2.2 节。

14. 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。

15. **（重要！！！）发起订购请求时，需要透传订单交易流水号，流水号必须保证唯一性，否则导致 APP 中无法正常发起交易业务。**

    订单交易流水号格式要求如下：

    a. **交易流水号类型限制：数字、字母、数字字母组合字符串；**
    b. **交易流水号长度限制：小于或等于 64 位字符串；**
    c. **必须保证订单交易流水号唯一性，即每次透传流水号不可重复。**

16. 需要开发者在调用 SDK 前，确保已获取手机存储权限，以免影响正常功能使用。

17. SDK 中支付宝支付方式为 H5 页面支付。

18. 此版本对外 API（Purchase_、PurchaseCode_）增加下划线，集成步骤稍微增加、改变（集成步骤详见 1.4 节导入 SDK）。

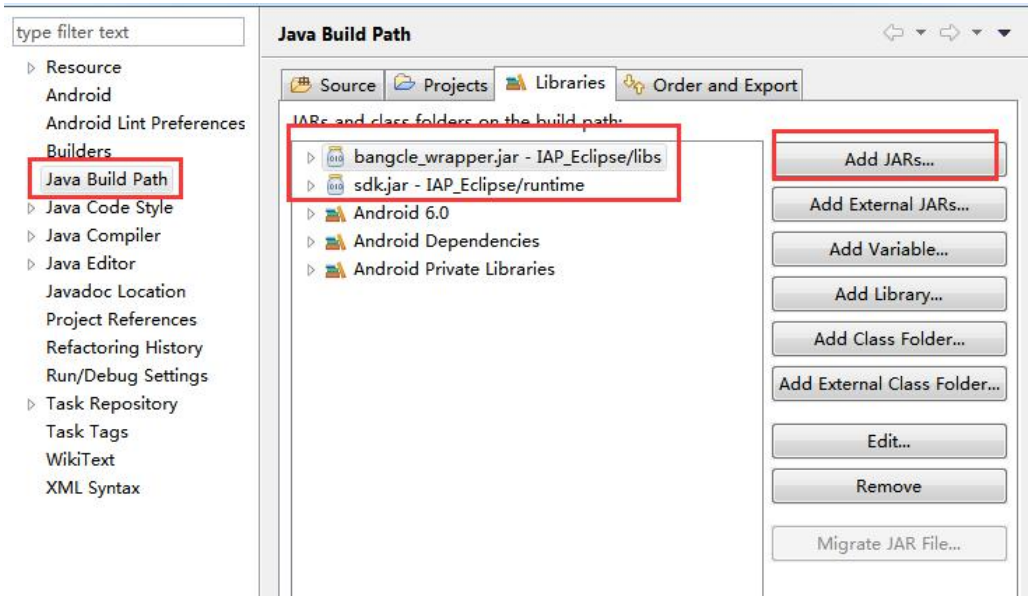
19. 有数计费融合版对外 API MobileAgent 类增加下划线，如 MobileAgent_

1.4 导入 SDK

1.4.1 Eclipse 环境

以下内容将说明如何将 SDK 资源导入到 Eclipse 工程应用中。

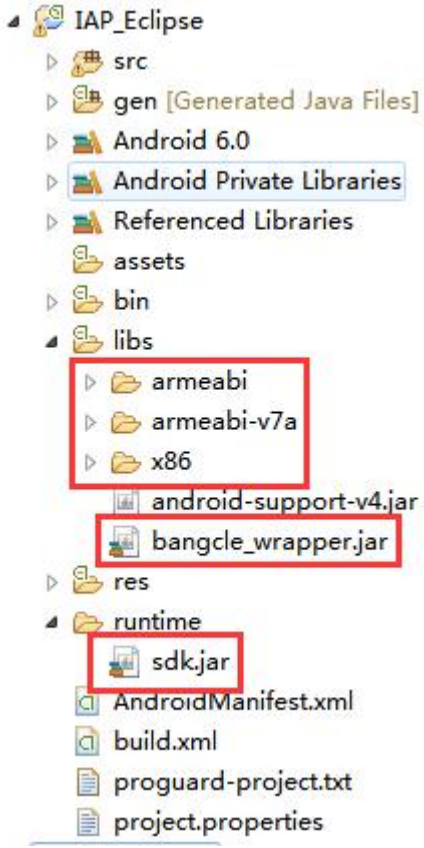
1. 将 SDK 资源包 mmbillingxx.xx.xx 文件夹 / bangcle_wrapper.jar 放到 Eclipse 项目中 libs；
2. 将 SDK 资源包 lib 文件夹 / armeabi / **.so 放到 Eclipse 项目 libs/armeabi，其他指令集 so 类似
3. 将 SDK 资源包 runtime 文件夹 / sdk.jar 并放到 Eclipse 项目中手动新建在项目根目录的 runtime 文件夹中
4. 右键项目 -> 属性 -> java build path -> library -> add jars，将 bangcle_wrapper.jar 与 sdk.jar 添加到项目依赖。 如图:



5. 由于运行时权限使用到 android.support-v4 包中的类 ,此版本需将 android.support-v4.jar 放到 libs 下。

a. v4 包版本需 23.0.1 及以上；
b. 亦可依赖 android-v7-support.jar，因为 v7 包括 v4 包；
c. v4 包请自行下载 ,或使用 SDK Demo 工程 libs 目录下的 v4 包。

最后，检查 Eclipse 项目中 Android Private Libraries 中是否可以看到上述步骤导入的 jar 文件，如（图 1， Eclipse 版本 demo 导入样例）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。



图一

1.4.2 Android Studio 环境

以下内容将说明如何将 SDK 资源导入到 Android Studio 工程应用中。

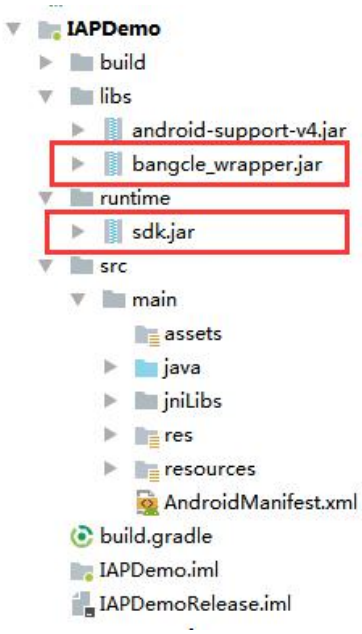
1. 将 SDK 资源包 mmbillingxx.xx.xx 文件夹 / bangcle_wrapper.jar 放到 Android Studio 项目中 libs；
2. 在 Android Studio 项目中的 app 目录下新建 runtime 文件夹,并将 SDK 资源包 mmbillingxx.xx.xx 文件夹 / runtime / sdk.jar
3. 将 SDK 资源包 lib 文件夹 / armeabi / **.so 放到 Android Studio

项目 src / main / jniLibs / armeabi，其他指令集 so 类似；

4. 由于运行时权限使用到 android.support-v4 包中的类 ,此版本需将 android.support-v4.jar 放到 libs 下。

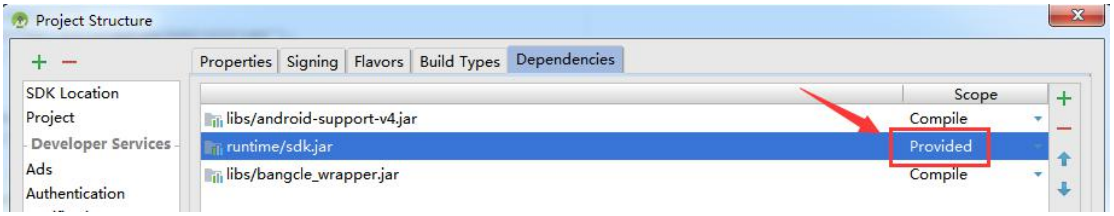
   a. v4 包版本需 23.0.1 及以上；
   b. 亦可依赖 android-v7-support.jar，因为 v7 包括 v4 包；
   c. v4 包请自行下载 ,或使用 SDK Demo 工程 libs 目录下的 v4 包。

最后，Android Studio 项目中导入 SDK 资源后，如（图 2）所示。



右键工程：选择 Open Module Settings，选择工程模块>Dependencies->+(添加 runtime 中的 sdk.jar)，并选择“Scope”为“Provided”，**否则编译报错！！！**

成功添加如图：



AS 环境导入注意事项

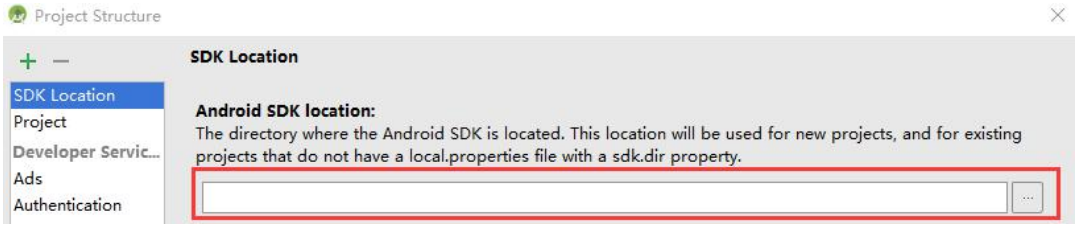
（重要）注意：由于 Android studio 中集成 NDK 编译环境，可致使

项目集成计费 SDK 后运行初始化崩溃。

解决方案：

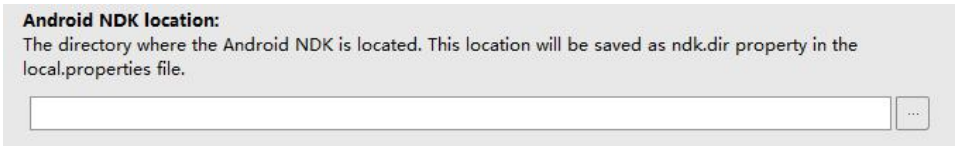
1. 删除或移除 AS 关联的本地 Android SDK 环境中的 ndk 环境包，参考路径：本地 Android SDK 环境路径/ndk-bundle；

Ps： 可在项目 Project Structure 的 SDK location 查看本地 Android SDK 路径



2. 重启 AS；

Ps： AS 重启后，确认是否已移除 AS 项目中 Project Structure 的 Android NDK location 关联；



1.5 导入 mmiap.xml 文件

1.5.1 mmiap.xml 说明（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，因此需要开发者自行在开发者网站获取，并集成。

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。

1.5.2 mmiap.xml 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”选择“使用的 SDK 版本”：“**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，“使用的 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

1.5.3 mmiap.xml 集成方法

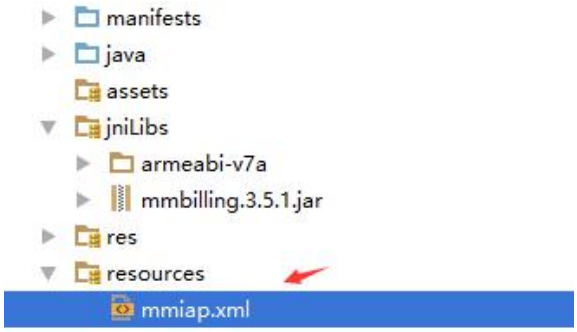
Eclipse 集成

1. Eclipse 工程中可导入到 src 文件夹下，与 AndroidManifest.xml

同级。mmiap 文件最终生成的 apk 解压后的根目录下(非工程目录根目录)，如下图。

Android Studio 集成

1. AndroidStudio 工程放入 resources 文件夹中。



其中 resources 可自行在 src/main/下创建，与 res 文件夹同级。

或者 build.gradle 中配置 resources.srcDirs（如下图），则可将 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. 应用内计费 SDK 集成

2.1 重载 Application 类，添加加载 SDK 代码（重要！）

（重要）必须在调用 SDK 初始化 init( )方法前，在应用工程的自定义 Application 中调用 Helper.install(Context context);

Helper.install()此方法必须调用 1 次且仅 1 次！

Demo 示例：

```
import android.app.Application;
import android.content.Context;

import com.secneo.mmb.Helper;

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 SDK 初始化与接口

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。

2.2.1 构造实例

Purchase_对象的创建使用了单例模式，不需要重复创建：

getInstance 函数说明	
函数名	getInstance()
说明	构造 Purchase_实例
返回值	Purchase_

示例代码：

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

2.2.2 配置参数

setAppInfo 函数说明		
函数名	setAppInfo(String appID, String appkey)	
说明	设置计费应用 ID 和 Key （必须）	
返回值	void	
参数	**类型**	**描述**
appid	String	开发者社区创建应用申请的 APPID
appkey	String	开发者社区创建应用申请的 APPKEY

示例代码：

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

setTimeout 函数说明		
函数名	setTimeout(int connTimeout, int dataTimeout)	
说明	设置超时时间(可选)，可不设置，缺省都是 15s	
返回值	void	
参数	**类型**	**描述**
connTimeout	int	连接超时时间，缺省是 15s
dataTimeout	int	读取超时时间，缺省是 15s

示例代码：

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

注意：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。

2.2.3 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。

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 查询交易是否成功。

注意：

1．**开发者可在游戏或者 APP 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，以减少等待时间

2．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。

3．初始化 init( )调用后，请等待 onInitFinish( )回调后，再发起其他业务请求。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。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。

init 函数说明	
函数名	init(Context context, OnPurchaseListener listener)
说明	初始化函数，**必须调用 1 次且仅 1 次**

返回值	void	
参数	**类型**	**描述**
context	Context	上下文
listener	OnPurchaseListener	计费 SDK 各种业务操作状态(查询，订购)监听器

示例代码：

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

2.2.4 订购

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，调用 **Purchase_**类中的 order 函数，传入相应的参数。

注意：

1. 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。

2.2.4.1 简化版订购接口

order 函数说明（一）	
函数名	order(Context context, String paycode,String tradID,OnPurchaseListener listener)
说明	订购接口，默认订购数量为 1

返回值	void	
参数	**类型**	**描述**
context	Context	上下文
paycode	String	计费点代码
tradID	String	**交易流水号，具体限制请查看 《1.3 节开发注意事项》 第 14 点**
listener	OnPurchaseListener	计费 SDK 各种业务操作状态(查询，订购)监听器

示例代码：

```
// 订购一个商品
Purchase_.order(context, paycode, tradID,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。

2.2.4.2 一般性订购接口

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。

order 函数说明（二）	
函数名	order(Context context, String paycode,String tradID, int orderCount,OnPurchaseListener listener)
说明	订购接口，可自定义订购数量
返回值	void

参数	类型	描述
context	Context	上下文
paycode	String	计费点代码
tradID	String	**交易流水号，具体限制请查看 《1.3 节开发注意事项》 第 14 点**
orderCount	int	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**）
listener	OnPurchaseListener	计费 SDK 各种业务操作状态(查询，订购)监听器

示例代码：

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

2.2.4.3 完整版订购接口

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。

order 函数说明（三）	
函数名	order(Context context, String paycode,String tradID, int orderCount,boolean nextCycle, OnPurchaseListener listener)
说明	订购接口，可自定义订购数量，是否续订
返回值	void

参数	类型	描述
context	Context	上下文
paycode	String	计费点代码
tradID	String	**交易流水号，具体限制请查看 《1.3 节开发注意事项》 第 14 点**
orderCount	int	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	boolean	对于租赁类业务，可预订下一期租赁周期
listener	OnPurchaseListener	计费 SDK 各种业务操作状态(查询，订购)监听器

示例代码：

```
// 租赁当前周期
Purchase_.order(context, month_paycode, tradID,1, false[true], listener);
```

2.2.4.4 用户可透传数据的订购接口

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。

order 函数说明（四）	
函数名	order(Context context, String paycode,String tradID, int orderCount,String data, boolean nextCycle, OnPurchaseListener listener)
说明	订购接口，可自定义订购数量，是否续订，自定义

	透传数据	
返回值	void	
参数	**类型**	**描述**
context	Context	上下文
paycode	String	计费点代码
tradID	String	**交易流水号，具体限制请查看 《1.3 节开发注意事项》 第 14 点**
orderCount	int	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**）
data	String	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。**
nextCycle	boolean	对于租赁类业务，可预订下一期租赁周期
listener	OnPurchaseListener	计费 SDK 各种业务操作状态(查询，订购)监听器

示例代码：

```
// 调用购买接口并传入自定义数据
Purchase_.order(context, paycode, tradID,1, data, true, listener);
```

2.2.5 查询

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。

调用 **Purchase_**对象中的 query 函数，传入相应参数。

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，可以获取到 orderID。

2.2.5.1 根据交易 ID 查询

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。

query 函数说明（一）		
函数名	query(Context context, String paycode,OnPurchaseListener listener)	
说明	查询接口	
返回值	void	
参数	**类型**	**描述**
context	Context	上下文
paycode	String	计费点代码
listener	OnPurchaseListener	计费 SDK 各种业务操作状态(查询，订购)监听器

示例代码：

```
// 查询单次或者租赁类商品是否订购成功
```

```
Purchase_.query(context, paycode, listener);
```

2.2.5.2 根据 paycode 查询

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。

query 函数说明（二）		
函数名	query(Context context, String paycode, String tradeID, OnPurchaseListener listener)	
说明	查询接口	
返回值	void	
参数	**类型**	**描述**
context	Context	上下文
paycode	String	计费点代码
tradID	String	调用 order()接口返回的交易 ID，用于查询交易是否成功
listener	OnPurchaseListener	计费 SDK 各种业务操作状态(查询，订购)监听器

示例代码：

```
// 带交易 ID 查询重复类商品是否交易成功
Purchase_.query(context, paycode, tradID, listener);
```

2.2.6 退订

根据计费点进行退订，仅对包月类型计费点有效。调用 **Purchase_**对象中的 unsubscribe 函数，传入相应参数。

unsubscribe 函数说明		
函数名	unsubscribe(Context context, String paycode, OnPurchaseListener listener)	
说明	退订接口	
返回值	void	
参数	**类型**	**描述**
context	Context	上下文
paycode	String	计费点代码
listener	OnPurchaseListener	计费 SDK 各种业务操作状态(查询，订购)监听器

示例代码：

```
// 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。

2.2.7 获取 SDK 版本号

调用 Purchase_对象中的 getSDKversion 函数，可获取当前的 SDK 的版本号。

getSDKVersion 函数说明		
函数名	getSDKVersion(Context context)	
说明	获取当前 SDK 版本号接口	
返回值	String，当前 SDK 版本号	
参数	**类型**	**描述**
context	Context	上下文

示例代码：

```
// 退订包月业务
 Purchase_.getSDKVersion(context);
```

2.2.8 获取对应状态码描述信息

调用 Purchase_对像中的 getDescription 函数 ,获取对应状态码的描述信息。

getDescription 函数说明		
函数名	getDescription(String code)	
说明	获取对应状态码描述信息	
返回值	String，对应状态码的描述信息	
参数	**类型**	**描述**
code	String	OnPurchaseListener 返回的错误码

示例代码：

```
// 获取对应状态码描述信息
Purchase_.getDescription(code);
```

2.2.9 获取对应状态码的错误原因

调用 Purchase_对像中的 getReason 函数，获取对应状态码的错误原因。

getReason 函数说明		
函数名	getReason(String code)	
说明	获取对应状态码的错误原因	
返回值	String，对应状态码的错误原因	
参数	**类型**	**描述**

code	String	OnPurchaseListener 返回的错误码

示例代码：

```
// 获取对应状态码的错误原因
Purchase_.getReason(code);
```

2.3 SDK 回调说明

计费 SDK 各种业务操作状态(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。

2.3.1 OnPurchaseListener

具体回调如下：

```
// 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：
void onUnsubscribeFinish(final String returnCode)
```

Returncode 的定义在 PurchaseCode_类中，具体含义可以通过 getDescription( )获取。

具体说明如下：

onInitFinish 说明

函数名	onInitFinish(String returnCode)	
说明	SDK 初始化回调接口	
返回值	Void	
参数	类型	描述
returnCode	String	初始化结果状态码

onQueryFinish 说明		
函数名	onQueryFinish(String returnCode, HashMap<String, Object> returnObject)	
说明	查询回调接口	
返回值	Void	
参数	类型	描述
returnCode	String	查询结果状态码
returnObject	HashMap<String, Object>	返回订单数据，详情见 2.3.2

onBillingFinish 说明		
函数名	onBillingFinish(String returnCode, HashMap<String, Object> returnObject)	
说明	订购回调接口	
返回值	Void	
参数	类型	描述
returnCode	String	订购结果状态码
returnObject	HashMap<String, Object>	返回订单数据，详情见 2.3.2

onUnsubscribeFinish 说明		
函数名	onUnsubscribeFinish(String returnCode)	
说明	退订回调接口	
返回值	Void	
参数	**类型**	**描述**
returnCode	String	退订结果状态码

2.3.2 返回数据 ReturnObject 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。

2.3.3 AndroidManifest 设置

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（开发者必须要注意的地方）

```
<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.WRITE_EXTERNAL_STORAGE" />
<uses-permission
android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
```

```
    <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <!-- 新增以下权限-->
  <uses-permission
android:name="android.permission.CHANGE_NETWORK_STATE" />
  <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
  <uses-permission android:name="android.permission.WRITE_SETTINGS" />
```

2) 增加 Application 声明

```
    <application
        android:name="com.cmic.iapdemo.MyApplication"
        android:allowBackup="false"
        android:icon="@drawable/icon"
        android:label="@string/app_name" >

        ...

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

说明：

1、MyApplication 是开发者自定义的名称。此处以 Demo 中 MyApplication 举例；

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

3、Android API Level 8 及其以上 Android 系统提供了为应用程序数据的备份和恢复功能，此功能的开关决定于该应用程序中 AndroidManifest.xml 文件中的 allowBackup 属性值，其属性值默认是 True。当 allowBackup 标志为 true 时，用户即可通过 adb backup 和 adb restore 来进行对应用数据的备份和恢复，这可能会带来一定的安全风险。

建议将 AndroidMannifest.xml 文件中的 android:allowBackup 属性值设为 false。

2.4 SDK 示例代码说明

以下是说明资源包中的 Demo 使用注意事项。

2.4.1 使用注意事项

在 Demo 工程中，APPID、APPKEY、PayCode 默认赋值为 00000000000。请填入在移动开发者社区创建应用生成的 APPID，APPKEY，PayCode。

2.5 获取渠道 ID

应用内计费 SDK 中包含有渠道配置文件 mmiap.xml。开发者可以读取此文件取得当前程序包的渠道 ID。

调用方法如下：

2.5.1 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();
        }
```

2.5.2 解析上面函数返回的 string，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 com.secneo.mmb.**
-keep class com.secneo.mmb.** {*;}
```

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。

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

注意：有数计费融合版对外 API MobileAgent 类增加下划线，如 MobileAgent_ 。