**阅读本文的基础**

ActionScript 3.0编程基础  
Apple iOS开发者权限

**前言**

记得在写Flash开发iOS应用全攻略的时候，不少感兴趣的朋友询问Flash技术是否能实现App Store的应用内付费功能（In-App Purchase，以下简称IAP）。在那个时候我只能说，很抱歉，目前还不支持，但是在半年后的今天，在我提笔写这篇教程的时候，我刚刚完成了一个用Flash技术实现IAP的例子。确切的说，是用AIR 3.0的原生扩展功能（AIR Native Extension，以下简称ANE）实现的。ANE作为AIR 3.0的一项重要特性，为Flash平台往系统底层的功能延伸奠定了基础，为Flash开发者在移动设备上开发商业应用提供了技术保证。它的出现，使AIR更具开放性，随即推动的是使用AIR和原生技术共同开发工作流，AS开发者与Native开发者的关系也将从以往的竞争转为合作。

在这个系列的文章中，我会对ANE做一个详细的介绍，同样介绍的还有苹果IAP的相关知识，最后通过一个具体的例子来讲解从流程到代码，从思路到技巧等各方面的知识点。希望各位朋友在阅读之后可以全面掌握ANE和IAP，并在实际的应用开发中派上用场。

[AIR面向iOS设备的原生扩展](http://jamesli.cn/blog/?p=1164)

**本文的主要内容如下**

 AIR Native Extension介绍

 ANE的组成部分

 ActionScript 3.0扩展

 Objective-C 扩展

 使用ADT打包ANE

 使用ADT打包IPA

————————————————————————————  
**AIR Native Extension介绍**

AIR Native Extension (ANE)是AIR 3.0的一项重要特性，简单的说，它允许AIR应用程序通过扩展文件与原生应用程序类库相互通讯，从而让AIR应用实现一些只有原生程序才可以做到的功能。  
在ANE出现以前，移动平台上的AIR对系统的访问非常有限，功能的实现都是封装在封闭的，由Adobe定义好的ActionScript 3.0 API内，比如Accelerometer, GeoLocator等AS类。ANE则将AIR彻底开放出来，AIR不再针对具体的功能提供封闭的API，而是允许开发者通过AIR的扩展机制自由调用使用原生语言开发的类库。这样可以让AIR应用程序享有与原生应用程序同等的机会，其意义对Flash技术来说是划时代的。  
————————————————————————————  
**ANE的组成部分**

ANE支持向Windows、Mac OSX、Android和iOS各个平台原生应用程序的扩展，本文只针对iOS平台进行介绍。在iOS平台中，ANE的组成部分基本分为ActionScript 3.0扩展类库和Objective-C原生扩展类库两个部分，这两个部分打包后生成AIR扩展文件(.ane)，最后和AIR应用程序一起打包成iOS原生应用IPA文件。如下图所示。

  
图1 ANE的组成部分  
————————————————————————————  
**ActionScript 3.0扩展**

ANE的AS扩展部分是一个SWC，AIR 3.0 SDK里为flash.external.ExtensionContext类添加了新的方法。如下例所示：

|  |  |
| --- | --- |
| 1  2  3  4  5 | import flash.external.ExtensionContext;  ...  private var ext:ExtensionContext;  ...  ext = ExtensionContext.createExtensionContext("com.adobe.appPurchase",""); |

在这个例子里，ExtensionContext通过静态方法createExtensionContext()来获得一个实例，**参数com.adobe.appPurchase是这个扩展的ID，它非常重要，在扩展的配置文件里和应用程序描述文件中都需要用这个ID进行配对。**

调用原生类中定义的方法可以用方法call()来实现，由于是同步调用，所以函数可以有返回值。如在原生类中定义的方法finish，可以用下面的代码来调用。

|  |  |
| --- | --- |
| 1 | var result:Object = ext.call("finish"); |

我们还可以给ExtensionContext类添加事件侦听，用来获取从原生类中派发回来的事件。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | ext.addEventListener(StatusEvent.STATUS,onStatus);  public function onStatus(e:StatusEvent):void{  switch(e.code){  case "removeTransaction":  ...  }  ｝ |

————————————————————————————  
**Objective-C 扩展**  
接下来是原生类的部分，如果你注册成为苹果iOS开发者，那么你可以在[苹果开发者网站](http://developer.apple.com/devcenter/ios/index.action)上免费下载Object-C的开发工具XCode。关于如何注册成为苹果iOS开发者，请参考我的这篇文章，[如何成为一个合法的iOS开发者](http://jamesli.cn/blog/?p=966)。

总的来说，Objective-C 虽然语法比较奇怪，但只要掌握了基本的规则，还是和ActionScript一样易懂。OBJC扩展类需要引入一个FlashRuntimeExtension.h类包，它实现了和ActionScript沟通的接口。

引入FlashRuntimeExtension.h之后，可以用下面的代码定义一个FREObject方法，FREObject是接口类型。**这里要注意，与AS的接口包括函数返回值，都要定义成FREObject类型**，比如代码中的retVal。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | FREObject finishTransaction1(FREContext ctx, void\* funcData, uint32\_t argc, FREObject argv[]) {  NSLog(@"Finish Transaction Called");  BOOL matchFound = NO;  const uint8\_t\* str = nil;  uint32\_t len = -1;  ......  FREObject retVal;  if(FRENewObjectFromBool(matchFound, &retVal) == FRE\_OK){  return retVal;  }else{  return nil;  }  } |

要把FREObject方法定义成接口，还需要在ContextInitializer方法内进行配置，如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | //这里是需要定义的接口的数量  \*numFunctionsToTest = 6;    //定义一个FRENamedFunction类型的实例func，初始化函数的个数  FRENamedFunction\* func = (FRENamedFunction\*)malloc(sizeof(FRENamedFunction)\*6);    //定义一个接口，name是字符串"getProducts"，函数体是getProducts  func[0].name = (const uint8\_t\*)"getProducts";  func[0].functionData = NULL;  func[0].function = &getProducts;    func[1].name = (const uint8\_t\*)"startPayment";  func[1].functionData = NULL;  func[1].function = &startAppPayment;    func[2].name = (const uint8\_t\*)"finish";  func[2].functionData = NULL;  func[2].function = &finishTransaction1;    func[3].name = (const uint8\_t\*)"muted";  func[3].functionData = NULL;  func[3].function = &muted;    func[4].name = (const uint8\_t\*)"restore";  func[4].functionData = NULL;  func[4].function = &restoreTrans;    func[5].name = (const uint8\_t\*)"trans";  func[5].functionData = NULL;  func[5].function = &getTrans;    \*functionsToSet = func;  .... |

而ContextInitializer方法，是在原生扩展类的初始化函数ExtInitializer中指定的：

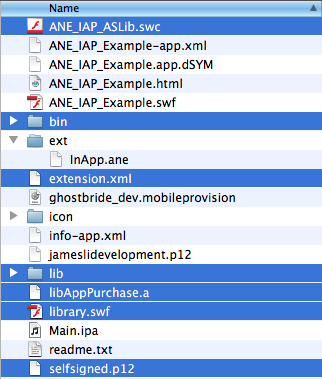
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | void ExtInitializer(void\*\* extDataToSet, FREContextInitializer\* ctxInitializerToSet,  FREContextFinalizer\* ctxFinalizerToSet) {  NSLog(@"Extension Initialized");  \*extDataToSet = NULL;  \*ctxInitializerToSet = &ContextInitializer;  \*ctxFinalizerToSet = &ContextFinalizer;  } |

ExtInitializer是原生扩展的程序入口，它可以通过扩展配置文件extension.xml来定义：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <?xml version="1.0" encoding="utf-8" standalone="yes"?>  <extension xmlns="http://ns.adobe.com/air/extension/2.5">  <id>com.adobe.appPurchase</id>  <versionNumber>1</versionNumber>  <platforms>  <platform name="iPhone-ARM">  <applicationDeployment>  <nativeLibrary>libAppPurchase.a</nativeLibrary>  <initializer>ExtInitializer</initializer>  <finalizer>ExtFinalizer</finalizer>  </applicationDeployment>  </platform>  </platforms>  </extension> |

我介绍的这个顺序，实际上就是实际程序编写的思路，先确定接口，再实现连接。 也许有朋友和我一开始接触OBJC的时候一样，对这些代码一头雾水。没有关系，在这篇教程里我只是对流程做简短的介绍，具体的代码解析会在本系列的最后一篇教程里做更详细的讲解。那么接下来让我来介绍下一个部分，打包扩展。  
————————————————————————————  
**使用ADT打包ANE**

在图1中，我介绍了.ane文件的组成，它包括了AS类库(.swc)和原生类(.a)两个部分，以及刚才我们介绍的这个扩展配置文件extension.xml。那么要打包ane我们还需要哪些文件呢？

  
图2 打包ANE所需要的文件

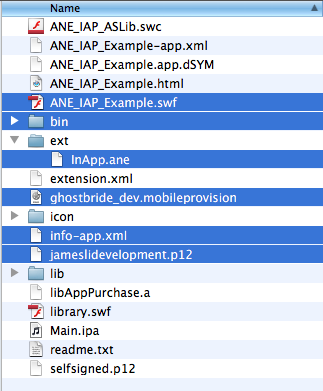
如图2所示，所选择的文件以及文件夹就是打包ANE所需要的所有文件，它包括：  
1，AIR SDK打包应用程序和类库(bin,lib)  
2，ActionScript扩展类包.swc，如图ANE\_IAP\_ASLib.swc  
3，ActionScript扩展类包.swf，如图library.swf，可以通过将SWC的文件扩展名改成ZIP后解压缩得到。  
4，Objective-C扩展类包.a，如图libAppPurchase.a，可以通过在Xcode中编译项目得到。  
5，扩展配置文件XML，如图extension.xml  
6，一个打包证书，如图selfsigned.p12，可以通过Flash CS5的AIR发布设置生成。

一切就绪后便可以使用命令行进行打包，注意路径，下例路径为当前文件夹。

|  |  |
| --- | --- |
| 1 | bin/adt -package -storetype pkcs12 -keystore selfsigned.p12 -storepass 1234 -target ane ext/InApp.ane extension.xml -swc ANE\_IAP\_ASLib.swc -platform iPhone-ARM library.swf libAppPurchase.a |

————————————————————————————  
**使用ADT打包IPA**

.ane文件打包成功后，便可以用来打包IPA文件，也就是iOS应用程序包。如果你对开发iOS应用的必要流程还不很清楚，请参阅我的这篇教程，[如何使用iOS开发者授权以及如何申请证书](http://jamesli.cn/blog/?p=995)。我以前介绍过如何用Flash Professional CS5打包IPA，今天主要介绍如何用AIR SDK的打包工具ADT来生成含有ANE扩展的IPA。

  
图3 使用ADT生成含有ANE扩展的IPA所需要的文件

如图3所示，所选择的文件就是生成IPA的必要文件：  
1，应用程序文件SWF，如图是ANE\_IAP\_Example.swf。  
2，开发者设备授权文件.mobileprovision，如图是ghostbride\_dev.mobileprovision。  
3，开发者签名证书文件.p12，如图是jameslidevelopment.p12。  
4，应用程序描述文件XML，如图是info-app.xml。  
5，扩展包路径，如图是ext  
6，如果应用程序有图标图片，还需要图标文件夹，如图是icon

在应用描述文件XML中，需要对扩展追加一个定义：

|  |  |
| --- | --- |
| 1  2  3 | <extensions>  <extensionID>com.adobe.appPurchase</extensionID>  </extensions> |

这里可以看到，在AS扩展类、扩展配置文件extension.xml和应用描述文件info-app.xml中都指定了一个统一扩展的ID: com.adobe.appPurchase。

利用下面的命令行可以打包生成Main.ipa:

|  |  |
| --- | --- |
| 1 | bin/adt -package -target ipa-test-interpreter -provisioning-profile ghostbride\_dev.mobileprovision -storetype pkcs12 -keystore jameslidevelopment.p12 -storepass 1234 Main.ipa info-app.xml ANE\_IAP\_Example.swf -extdir ext icon |

## [iOS应用内付费简介和准备流程](http://jamesli.cn/blog/?p=1174)

**本文的主要内容如下：**

 IAP简介

 商品与交易

 测试IAP的准备流程

————————————————————————————  
**IAP简介**

IAP的全称是In-App Purchase，应用内付费。这种业务模式允许用户免费下载试用，对应用内提供的商品选择消费，比如购买游戏道具，购买游戏等级等等。相比完全收费的应用而言，应用内付费给用户试用的机会，不会让优秀的应用因为缺乏用户的认知而丧失消费者；而且对于开发商，也不需要为了让用户试用而单独发布一款免费的精简版本。  
————————————————————————————  
**商品与交易**

苹果官方的关于IAP的帮助文档是只对注册开发者开放的，所以我不方便把它转载在这里，但我可以对IAP里的关键因素做一个通俗易懂的介绍。

IAP里有两个关键的词，商品(Product)和交易(Transaction)。  
商品可以是一种虚拟的道具，一个隐藏的关卡或者地图，但必须是直观的，可以让用户购买后直接获益的产品。IAP的商品从消费性质上分为四种：

1，**消耗型商品**，比如游戏道具，子弹，药品等等。由于这类商品可以被消耗，所以支持重复购买。苹果应用商店不保存此类商品的购买记录，如果要保存则需要开发者同步到自己的服务器上。

2，**非消耗型商品**，比如游戏关卡，隐藏地图等等。这类商品只要购买一次便可以了，苹果应用商店里每一个用户对非消耗型商品的购买都有记录，可以在不同的设备上恢复购买状态，这个恢复的过程叫做Restore。

3，**自动重置型订阅**，比如电子杂志，读物等。消费者购买这类商品时会从列表中选择一个有效期限，卖家在定义商品的时候从一群固定的选项中选择添加一个有效期，比如7天，一个月，两个月。过了有效期之后，商品的购买状态会被自动重置成未购买，要想继续获得内容则需要再次订阅。这种类型的商品和非消耗型商品一样，会在苹果商店内保存购买记录。

4，**非自动重置型订阅**，比如用户订阅电子杂志和读物报刊时需要从自定义的期限列表中选择期限，而不是苹果提供的固定选项，比如9天，一个半月或任意时间。在这种情况下，苹果商店无法根据期限来控制订阅的到期行为，所以一切都需要开发商自己编写相应的逻辑来实现。

交易是指用户对一个商品的购买行为，当用户点击购买一件商品时，一个新的付费行为就被添加到付费队列中，付费队列是一条系统线程，即使应用程序终止仍然会继续执行。  
————————————————————————————  
**测试IAP的准备流程**

**如果你是一个注册的苹果开发者，接受了最新的苹果开发者协议，并且签署了iOS付费应用合同，那么你就可以继续IAP的测试了。**

如果此时你对iOS的开发流程不是很熟悉，在阅读下面的内容之前，我强烈建议你先看看我的这几篇文章：  
[如何成为一个合法的iOS开发者](http://jamesli.cn/blog/?p=966)  
[如何使用iOS开发者授权以及如何申请证书](http://jamesli.cn/blog/?p=995)  
[如何为iTunes Connect准备应用](http://jamesli.cn/blog/?p=1045)

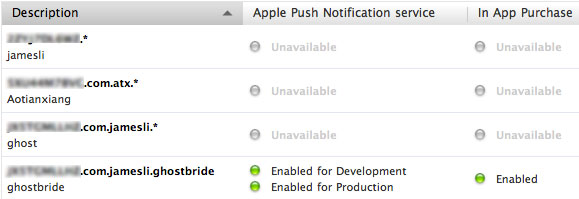
如果你已经注册成为苹果开发者，并且熟知了如何给设备授权，如何申请证书，如何发布应用，那么请继续阅读我下面的内容：

**使用未越狱的设备**

测试IAP的项目不能使用越狱的设备，否则会出现无法连接到应用商店的错误。恢复设备到未越狱的系统后，登陆[Provisioning Portal](http://developer.apple.com/ios/manage/overview/index.action)添加设备的UID。

**使用没有通配符的App ID**

在定义App的Bundle ID的时候，我曾经介绍过可以使用类似 com.jamesli.\* 这样的值来覆盖多个应用的ID。这种定义方式不能用在打算使用IAP的应用上面，定义IAP的应用必须使用唯一的ID，如com.jamesli.ghostbride。如果正确定义了，应用的In-App Purchase的功能是默认开启的，如下图：



**在iTunesConnect创建应用**

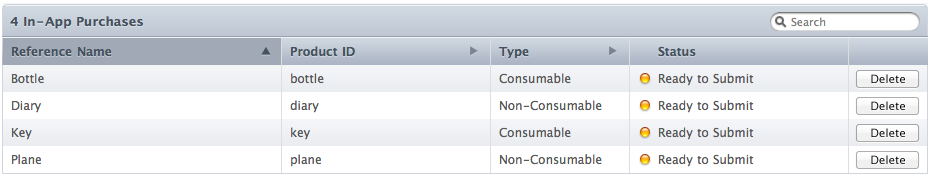
登陆[iTunesConnect](http://itunesconnect.apple.com/)，创建一个新的应用，即使该应用尚未开发，也可以用一些假的文字和图片来代替，**创建好之后切记要点击Ready to Upload binary将应用的状态变为Waiting for upload。**

**管理In-App Purchase商品**

在应用列表中点击新创建的应用图标，进入应用首页，在右面的一行按钮中选择Manage In-App Purchase，进入内付费商品管理页面。通过点击左上角的Create New按钮可以进入商品页面选择创建一个新的商品。页面中显示的四种商品分别是我在本文介绍过的四种商品，消耗型商品（Consumable），非消耗型商品（Non-Consumable），自动重置型订阅（Auto-Renewable Subscriptions），非自动重置型订阅（Non-Renewing Subscription）。

以消耗型商品为例，点击Select进入创建页面。  
Reference Name是商品名字，这不是最终用户会看到的名字，而是会在内付费管理的商品列表中显示的字符，类似于变量名。  
Product ID是商品的唯一标识，这个ID十分重要，在编写应用程序的时候会用它来识别改商品。

接下来是为不同的语言定义该商品的显示名称，最终用户看到的就是这个名称。定义好名称后是为商品定价以及上传缩略图，这个商品就算是定义完了。如下图，定义完成的商品会显示在内付费管理的商品列表中。每一个内付费商品的创建和修改都需要提交审核，但这里需要注意的是，**在一个新的应用版本内创建的内付费商品，必须和这个应用版本一起提交审核，而在该应用版本通过审核之后再为它创建的内付费商品，可以通过这个列表中的Ready to submit按钮来提交。**



刚刚创建好的内付费应用，已经可以用来调试了。

**使用测试帐号调试应用**

苹果应用商店是一个交易环境，任何用户可以在这个环境内购买应用，但如果要测试正在开发过程中的应用内付费，我们不能在真正的苹果商店里进行。苹果给开发者提供了一个用于调试购买行为的测试沙箱，它完全复制了应用商店的交易环境，但在沙箱环境中我们不能用平常的苹果帐号，而是需要用测试帐号。

在iTunesConnect的首页可以点击Manage Users进入用户管理页面，然后选择Test User来创建测试帐号。根据苹果开发者的最新谢意，创建测试帐号必须使用一个真实的Email地址，而且密码必须是符合规范的，测试账号需要在邮件里激活后才可以使用。这里创建的帐号可以用来购买开发过程中的应用内付费，**但必须记住，测试帐号不能用来登陆真正的应用商店并在产品环境中进行购买行为，否则你的iTunes帐号将有可能被停用。**

## [ANE面向IAP的测试和开发](http://jamesli.cn/blog/?p=1179)

**本文的内容如下：**

 在Native扩展中使用StoreKit框架

 ActionScript扩展

 ANE－IAP开发实例分享

**本文用到的工具和设备如下：**  
[Flash Builder 4.5](http://www.adobe.com/products/flash-builder-family.html)  
[Flex SDK 4.5.1](http://opensource.adobe.com/wiki/display/flexsdk/Download+Flex+4.5)  
[Flash Professional CS5.5](http://www.adobe.com/products/flash.html)  
[AIR SDK 3.0](http://www.adobe.com/special/products/air/sdk/)  
[Xcode 4.1 + iOS SDK 4.3](https://developer.apple.com/devcenter/ios/index.action)  
iPad 1

**本文例子中的Objective-C部分和AS扩展类库部分的代码作者为我的同事**[**Saumitra Bhave**](http://saumitrabhave.com/)  
———————————————————————————–  
**在Native扩展中使用StoreKit框架**

下载安装XCode 4和iOS SDK之后，让我们先来了解一下苹果原生类库是如何处理应用内付费功能的。iOS SDK中有一个框架叫做StoreKit，它负责应用程序和应用商店的业务流程。StoreKit中有一些负责具体功能的类，商品的请求，请求结果，结果回调，购买，购买队列等都由不同的类来完成。

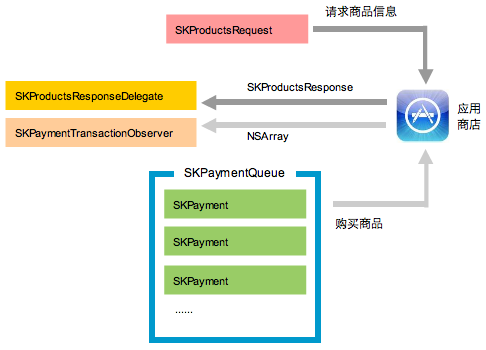


图1 StoreKit的业务流程

如上图所示，StoreKit解决内付费业务的大体流程可以概括成这样，SKProductsRequest向商店发出请求获得商品信息，商店通过回调函数SKProductsRequestDelegate把请求的结果SKProductsReponse传了回来，如果用户选择购买商品，则创建一个SKPayment实例到购买队列SKPaymentQueue中，然后通过回调SKPaymentTransactionObserver来返回购买的结果。

我现在拿请求商品信息的部分来举例说明一下：

|  |  |
| --- | --- |
| 1  2  3 | SKProductsRequest\* req = [[SKProductsRequest alloc] initWithProductIdentifiers:pids];  req.delegate = observer;  [req start]; |

上面的Objective-C的代码如果转译成ActionScript 3.0，相当于这个意思：

|  |  |
| --- | --- |
| 1  2  3 | var req:SKProductsRequest = SKProductsRequest.initWithProductIdentifiers(pids);  req.delegate = observer;  req.start(); |

SKProductsRequest继承于父类SKRequest，通过静态方法initWithProductIdentifiers和参数pids创建了一个实例req，参数pids是一个数组，列出了需要请求的内付费商品ID。req通过start方法向商店发出请求，并在发出请求的动作之前给自己注册了一个回调函数observer。

observer是SKProductsRequestDelegate的实例，回调接口如下：

|  |  |
| --- | --- |
| 1 | - (void)productsRequest:(SKProductsRequest \*)request didReceiveResponse:(SKProductsResponse \*)response; |

转译成AS是如下的代码：

|  |  |
| --- | --- |
| 1 | function productsRequest(request:SKProductsRequest, response:SKProductsResponse):void; |

其中response就是请求的结果。

下面是一个完整的Objective-C函数体：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | - (void)productsRequest:(SKProductsRequest \*)request didReceiveResponse:(SKProductsResponse \*)response{  NSLog(@"Products Received");  NSMutableString\* retXML = [[NSMutableString alloc] initWithString:@"<products>"];  for (SKProduct\* p in response.products) {  [retXML appendFormat:@"<product><title>%@</title><desc>%@</desc><price>%@</price><locale>%@</locale><id>%@</id></product>",p.localizedTitle,p.localizedDescription,p.price,[p.priceLocale localeIdentifier],p.productIdentifier];  }  [retXML appendFormat:@"<invalid>"];  for(NSString\* s in response.invalidProductIdentifiers){  [retXML appendFormat:@"<id>%@</id>",s];  }  [retXML appendFormat:@"</invalid></products>"];  FREDispatchStatusEventAsync(g\_ctx, (const uint8\_t\*)"productsReceived", (const uint8\_t\*)[retXML UTF8String]);  [retXML release];  [request release];  } |

这里主要实现的是将结果格式化成一个XML流，然后通过派发事件传回给AS扩展类，注意这里派发事件用的是FREDispatchStatusEventAsync方法，它定义在FlashRuntimeExtension.h内部，前文我介绍过，这是负责与AS扩展类通信的类。

在AS类包中，通过注册的StatusEvent来侦听这个事件，并从e.level中取得XML流里的商品信息。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | ext.addEventListener(StatusEvent.STATUS,onStatus);  ...  function onStatus(e:StatusEvent):void{  switch(e.code){  case "productsReceived":  var xml:XML = new XML(e.level);  ......  }  } |

购买的业务流程和请求信息的流程十分相似，这里我不一一介绍，大家可以在最后下载项目的代码来查看。  
———————————————————————————  
**ActionScript扩展**

在Saumitra提供的ANE扩展类中，AS部分的结构是这样的：  
com.adobe.nativeExtensions.AppPurchase;  
com.adobe.nativeExtensions.AppPurchaseEvent;  
com.adobe.nativeExtensions.Base64;  
com.adobe.nativeExtensions.Product;  
com.adobe.nativeExtensions.Transaction;

其中AppPurchase负责业务流程以及与Native扩展的接口，AppPurchaseEvent定义了各种StatusEvent的状态，Product和Transaction定义了数据模型，Base64负责为参数转码。整个扩展类库简单易懂，我会在下面的项目实例中介绍其中用到的一些方法。  
———————————————————————————–  
**ANE－IAP开发实例分享**

最后总结本系列教程所有的知识来做一个例子。  
我首先做的准备是在iTunesConnect中创建了一个新的应用，并新建了四个内付费商品：plane（非消耗型），diary（非消耗型），bottle（消耗型），key（消耗型）。我希望在iPad的沙箱环境中测试以下的业务流程：  
请求商品信息，购买消耗型和非消耗性商品，恢复非消耗型商品的购买状态，如果都成功则我的应用调试成功。

分析一下这个应用的UI需求，我需要一个按钮来触发请求商品信息的动作，然后需要一个列表来显示商品信息，接下来需要给每个商品添加一个购买的按钮。由于iPad上无法用trace等debug方法调试，所以我还需要一个监测的窗口来打印所有的流程信息。

下面是该应用的截屏：

  
图2 请求商品列表

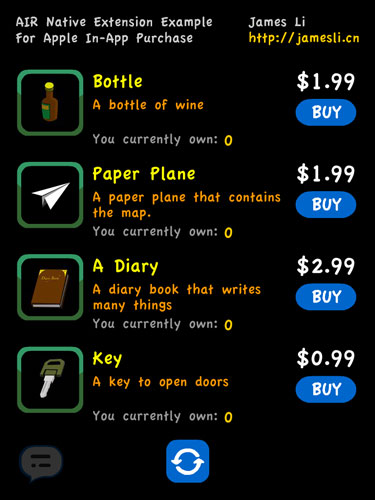
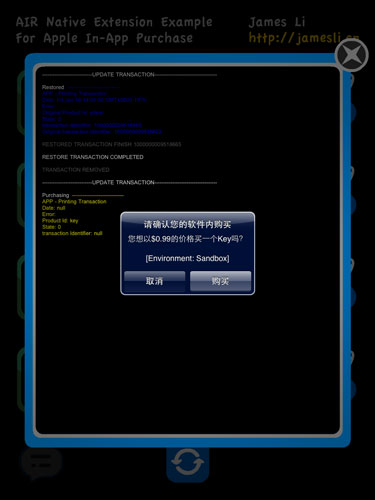
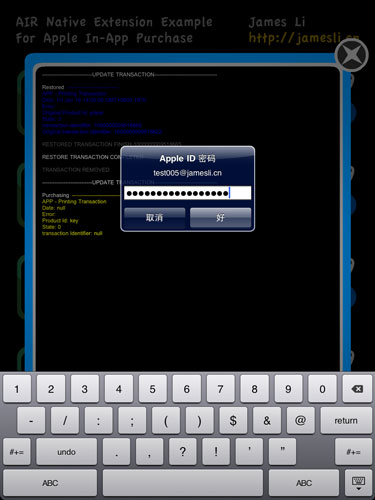
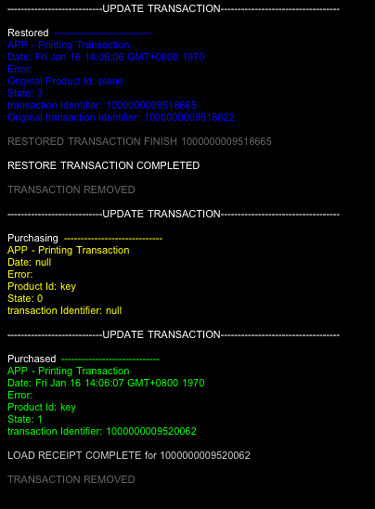
  
图3 显示商品信息

  
图4 恢复非消耗型商品的购买状态

  
图5 购买新的消耗型商品

  
图6 输入测试用户密码

  
图7 购买成功

**资源：**  
[实例ANE-IAP（包括AS扩展类，Native扩展类）下载](http://jamesli.cn/uploads/ANE_IAP.zip)

**如何使用本例：**  
开发者需要使用自己的证书和设备来打包和发布，有关如何打包ANE和如何发布IPA，我已经在这个系列教程的前文提到过了，本文不做重复介绍。

**注意事项：**  
1，本例使用Flash Professional作为Compiler，使用Flash Builder作为代码编辑IDE，请在Flash Professional的发布设置中选择Flash Player，并在发布的时候忽略这个错误： VerifyError: Error #1014: Class flash.external::ExtensionContext could not be found。直接使用生成的SWF即可。

2，在iPad上测试之前务必要先注销已经登陆的苹果帐号，注销方法为，进入系统偏好设置，在左边列表内点击Store图标，然后点击在右侧出现的帐号，再在弹出的窗口中点击”注销“。

3，如果你没有用过XCode，不知道如何发布OBJC项目，请在安装XCode之后打开下载的Native扩展包里的AppPurchase.xcodeproj文件，项目打开后按Command+B，在左边项目资源列表的Products文件夹内会生成一个.a文件，右键点击后可以在Finder中找到这个文件。

总结：  
本文所提供的例子，只是应用内付费的入门，如果要做真正的产品，你需要搭建自己的服务器用来验证购买商品的收据，以及在本地记录商品的使用状态并与服务器同步等等。这些知识不是本文的重点，这里不做详细介绍，感兴趣的朋友可以和我做进一步的探讨。谢谢各位！

摘自：<http://jamesli.cn/blog/>