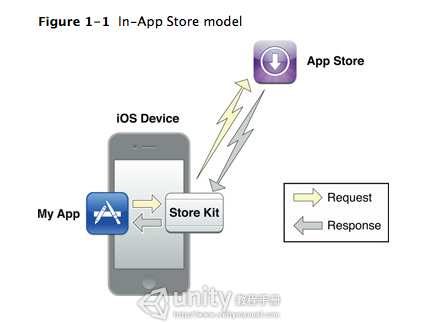
**Unity3D教程：Unity3D游戏内的付费**

Posted on 2013年01月25日 by U3d / [Unity3D 基础教程](http://www.unitymanual.com/category/manual/unity3d-%e5%9f%ba%e7%a1%80%e6%95%99%e7%a8%8b)/被围观 334 次

**一、搭建号沙盒环境**

**二、IAP付费流程图：**

总体流程图如下：

[](http://www.unitymanual.com/wp-content/uploads/2013/01/143909smglu2jl46i69902.png)

Unity3D教程：Unity3D游戏内的付费

详细流程图分为带服务端验证和不带服务端验证，本文研究的是带服务端验证，流程图如下：

在Unity中制作IAP主要思想和OC是一样的，只需更改输入接口和输出接口，所以本文主要围绕如何通过C#以插件的形式，在OC跟C#之间建立连接，本质是非托管跟托管之间的连接（托管是可以再公共语言运行库（CLR）上运行的）。

**三、接来下我以代码的形式，简短的将整个过程贯穿起来。**

首先点击付费按钮之后，调用StoreKit.Install(产品的部分ID);//完整这样com.XXX.XXXX.iap.50，此处填com.XXX.XXXX.iap。StoreKit.Install(产品的部分ID)会调用插件里\_StoreKitInstall(productIdPrefix)，\_StoreKitInstall(productIdPrefix)跟OC建立起了连接，调用相应的OC函数，最后会在OC一个变量中保存产品的部分ID信息。

其次当用户点了某一个购买按钮，向OC发送一次请求，当OC受到请求后，会向App store发送请求，验证当前产品ID是否合法，合法的话，会返回BaseKey，productID，OrderId信息。 UnitySendMessage(“Config”, “BuyComplate\_CallBack”, [json UTF8String]);通过这个函数，完成OC和C#一次回调。以json的形式返回给C#产品的订单信息。（UnitySendMessage函数中Config是放置购买脚本的GameObject，BuyComplate\_CallBack是购买脚本里面的回调函数）

最后，当客户端收到产品订单后，传给本地服务器，本地服务器拿到产品订单后，再跟App store进行一次验证，返回给客户端验证结果，客户端在更新虚拟货币信息。

**四、核心代码**

StoreKitPluginEntry.mm和StoreKit.cs是连接OC和C#的桥梁，具体代码如下：

StoreKitPluginEntry.mm static IAPTransactionObserver \*observer; static NSString\* CreateNSString (const char\* string) { return [NSString stringWithUTF8String:(string ? string : "")]; } extern “C” void \_StoreKitInstall(const char \*productIdPrefix) { if (observer == nil) { observer = [[IAPTransactionObserver alloc] initWithProductIdPrefix:CreateNSString(productIdPrefix)]; } } extern “C” void \_StoreKitBuy(const char \*productName) { [observer queuePayment:CreateNSString(productName)]; }Source code

StoreKit.cs static string productIdPrefix\_; public static void Install(string productIdPrefix) { productIdPrefix\_ = productIdPrefix; #if UNITY\_IPHONE && !UNITY\_EDITOR \_StoreKitInstall(productIdPrefix); #endif } public static void Buy(string productName) { #if UNITY\_IPHONE && !UNITY\_EDITOR \_StoreKitBuy(productName); #endif } #if UNITY\_IPHONE [DllImport("\_\_Internal")] private static extern void \_StoreKitInstall(string productIdPrefix); [DllImport ("\_\_Internal")] private static extern void \_StoreKitBuy(string productName); #endif [DllImport ("\_\_Internal")] 是托管跟非托管的桥梁。以下是Mono官网对 [DllImport ("\_\_Internal")] 的说明 To make the runtime lookup the symbol in the current executable, use the special library name \_\_Internal like this, in your DllImport attribute:

using System.Runtime.InteropServices; [DllImport ("\_\_Internal", EntryPoint="DoSomething")]static extern void DoSomething ();The “\_\_Internal” library name will instruct Mono not to look this up in an external library, but to try to satisfy the symbol referenced (DoSomething) in the current executable image.

Buy.cs购买代码

Source code

public void BuyComplate\_CallBack(string result){ string url=”"; print(“result:”+ result); url+=”m=XXX&a=XXX&uid=”+player.PlayerID; Hashtable json=(Hashtable)MiniJSON.JsonDecode(result);//json解析器 productInfo=json["productID"].ToString().Substring(productInfo.Length+1);//截取购买的类型 WWWForm resultPost=new WWWForm();//由于json字节过长，不能采用get方式提交，所以选用Post方式提交 resultPost.AddField(“basyKey”,json["BaseKey"].ToString()); resultPost.AddField(“OrderId”,json["OrderId"].ToString()); resultPost.AddField(“productID”,json["productID"].ToString()); StartCoroutine(BuyComplate(url,str,resultPost)); } IEnumerator BuyComplate(string url,string productId,WWWForm buyInfo)// { WWW productInfo=new WWW(url,buyInfo); yield return productInfo; //print(“data:”+productInfo.text); if(productInfo.error==null) { Hashtable result=(Hashtable)MiniJSON.JsonDecode(productInfo.text); if(result["status"].ToString()==”ok”) { switch(productId) { case “tier1″:player.Gemstone+=50;break; } } } }

到此，Unity之IAP讲述完毕，以下附上原工程和对应的Json解析器。ECPurchase和 testIap下载。