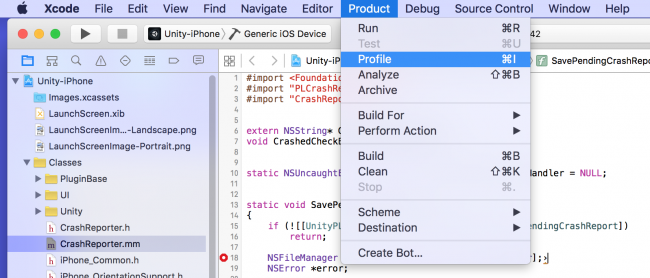
**使用Instruments进行性能探查**

在对企业做技术支持服务的过程中，我们经历过许多iOS项目。在每一个iOS开发过程中，开发者们总会遇见，一边运行游戏一边纳闷“这玩意儿为什么跑这么慢？”的时刻。其实有许多很不错的性能分析工具集，今天我们将会为大家介绍Instruments，这就是其中的佼佼者。

要使用Instruments，或任何其他的XCode调试工具，您必须构建一个以iOS为目标的Unity项目（同时取消对Development Build以及Script Debugging选项的勾选）。然后使用XCode以Release模式，将生成的XCode项目编译后发布到一个已连接的iOS设备上。

启动Instruments后（通过长按Play按钮，或选择 Products>Profile），选择Time Profiler。接着选择生成应用，并按下红色的Record按钮，便可启动探查过程。此时，iOS设备上会开始运行已连接了Instruments的应用，而Time Profiler则开始自动记录远程传回的信息。回传信息在Instruments的时间线上以蓝色折线图表示。



P.S.  Call Tree窗口右侧Details面板上的Settings子菜单中有Flatten Recursion 以及Hide System Libraries两个选项，可用于调整Call Tree窗口中调用的显示层次。

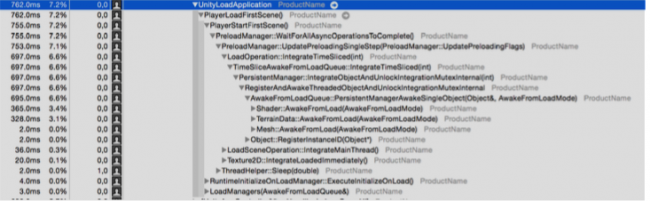
Instruments窗口中的详细区域会列出一系列方法调用，它们中的每个最顶级方法调用都代表了应用中的一个线程。

通常来说，因为main方法里包含所有托管代码，所以它成了理所当然的万众焦点。

展开main方法后，会出现一个非常庞大的方法调用树，其主要的分支会是以下两者之一：

* [startUnity] 以及UnityLoadApplication（这些方法名有时会全大写显示）。
* PlayerLoop

[startUnity]因为包含了初始化Unity引擎所需的所有时间，所以值得留意。在它下面是一个名为UnityLoadApplication的方法，在此方法之后方可进行启动时间探查。



当您对应用进行了一段时间的探查后，暂停Profiler，展开调用树。您会发现左栏以毫秒为单位的时间值沿树状结构向下递减。而您要做的是寻找那些上下差值巨大的时间点，即性能热点。随后您只要据此返回到对应代码，就可找到这时间黑洞的成因。

这原因有可能的确是某项必要的操作，也有可能是潜伏于您生产项目中的某些古老异类代码，或者……呃……原因真的是不胜枚举。怎样来（是否要）解决这个问题完全取决与您，因为没人比您更了解您的代码 :D。

Instruments还可以用于寻找分散的性能损耗点——即那些单独来说并没有造成很大时间消耗，但却在代码的许多不同位置不停吞噬少量时间的损耗点。

具体做法是，在Instruments中Call Tree窗口右上方的符号搜索框中输入部分或完全的函数名。如果正在探查的对象是一个游戏，则展开PlayerLoop并收起其下的其他方法。在某个特定操作上消耗的总毫秒数可以通过将PlayerLoop或UnityLoadApplication上所用的总时间减去self栏中对应的毫秒数进行粗略估计。

要查找的常见方法：

– “Box(“, “box” 以及 “box” — 这些代表了正发生C#值装箱操作；大多数的装箱实例都能轻易修正。

 – “Concat” — 字符串连接操作通常可以很容易进行优化

 – “CreateScriptingArray” — 所有返回数组的Unity API会分配数组的新副本。减少对这些方法的调用。

 – “Reflection” — 反射（Reflection）很慢。用此来预估在反射上的时间损失，并尽可能去掉它。

 – “FindObjectOfType” — 用此来定位重复的或非必要的FindObjectOfType调用，或其他已知超慢的Unity API。

– “Linq” — 根据时间损耗来确定创建或移除Linq查询；建议将热点对象替换为手工优化的方法。

Instruments除了能进行CPU时间探查之外，还可以用于探查内存使用率。Instruments的Allocations探查器提供了两个可以显示应用程序详细内存使用率的探针。Allocations探针能对特定时间段内驻留于内存中的对象进行检查。VM Tracker探针可对脏内存堆大小进行监视。脏内存堆是iOS确定是否要强制关闭一个应用的主要依据。

当在Instruments中选择Allocations探查器时，这两个探针都会同时运行。同样的，按红色的Record按钮开始探查过程。

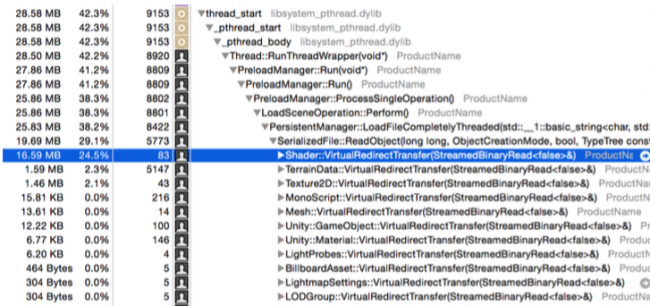
要正确设置Allocations探针，必须确保Instruments右边Detail选项卡上的以下这些设定都正确。确保Display Settings (中间选项)下的Allocation Lifespan被设为Created & Persistent。确保Record Settings (左侧选项)已勾选 Discard events for freed memory 选项。

Allocations探针的默认视图Statistics是内存行为诊断的最佳利器。它显示为一条时间线。当使用推荐设置时，图表上会实时显示代表时间及内存分配程度的蓝色线条。通过这个图表，您可以仅通过简单重复测试场景，监控长时间占用或泄露的内存，并确保每次运行之间没有蓝线产生。

Call Tree是另一个有用的视图，它可以显示发生内存分配操作的代码行，以及该代码行所负责的内存量。

您可以看到下图中大约25%的内存使用量由着色器消耗。既然这些着色器都位于加载线程中，那它们必定是在应用启动时就加载的Unity项目捆绑的标准着色器。

从下图您可以看到，单单着色器就消耗掉了测试应用总内存使用量的25%左右。既然这些着色器都位于启动加载的线程中，那它们必定是在每次应用启动时就加载的，与默认Unity项目捆绑的标准着色器。



与前面一样，当您定位到热点位置时，后续的动作完全取决于您的项目本身。

好了，这就是一个Instruments简要介绍。1000多字，没有任何“天龙特攻队（A-Team）”的引用，[我们可不想再陷入上次那样的麻烦了](http://blogs.unity3d.com/2015/12/14/enterprise-support-solving-hard-problems/)

。侵犯版权真的一点都不好玩。

Unity企业支持团队正在创作更多的类似指南，我们将在几个月内发布完整版的最佳实践指南。