第17章 游戏性能分析与优化

在开发游戏产品的时候，有一个难题是无法回避的，即如何提高游戏的运行性能。相对于一般应用，游戏中往往使用了大量的美术和音效素材，绚丽的粒子效果、逼真的物理模拟、人工智能算法等。虽然所有这些元素都可以增强游戏的互动性和代入感，但过多的资源负荷不可避免地会影响游戏的运行速度，也就是帧数，从而影响了玩家的实际体验。

本章将简要介绍如何使用Xcode开发工具包中的Instruments来发现游戏的性能瓶颈；此外还将介绍Cocos2D内置的性能分析工具CCProfiler和CCProfilingTimer，以及如何在项目中使用性能分析工具；最后对如何优化Cocos2D游戏的性能提出一些建议。

## 如何发现性能瓶颈

性能分析（Profiling，又称为Performance Analysis）是以收集程序运行时的信息为手段来研究程序行为的分析方法。性能分析与静态代码分析是相对的。性能分析的作用在于决定程序中的哪个部分应该被优化，从而提高程序的运行速度或者内存、CPU等硬件资源的使用效率。

有过iOS项目开发经验的朋友或许曾经接触过Instruments。这是一款由苹果官方随同Xcode一起提供的优秀性能分析工具。Instruments非常强大，使用它可以动态跟踪和分析Mac OS X以及iOS应用和游戏代码的性能。

接下来将对如何Instruments进行Cocos2D项目的性能分析加以介绍。

### Instruments基本功能

Instruments的基本功能包括：

* 检查单个或多个进程的行为；
* 记录一系列的用户行为并回放；
* 创建定制化的DTrace工具分析系统和程序的行为；
* 将用户界面记录和工具配置保存为模板，从而可以直接使用Xcode来读取。

使用Instruments工具可以实现以下功能：

* 跟踪代码中难以复制的问题；
* 对程序执行性能分析；
* 自动测试代码；
* 对程序碎片执行压力测试；
* 执行系统级的问题修复；
* 更深入地理解代码运作原理。

Instruments应用使用Instrument（分析工具）收集数据。每个分析工具都会收集并显示不同类型的信息，比如文件访问、内存使用等。Instruments应用内置一个标准化的分析工具库，可以对代码的不同方面进行测试。Instruments所完成的工作都会以跟踪文件（trace document）的形式存在。

注意 在使用Instruments进行测试之前，不要在虚拟机上运行程序。当在虚拟机上运行程序时，除了使用Leaks工具来捕捉内存泄露外，其他的性能分析都是无效的。

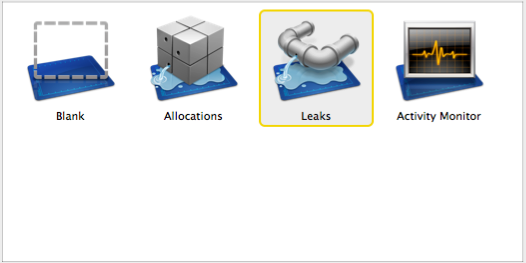
此外，在进行性能分析的时候，不要选择Debug。由于Debug需要增加一系列的检查，执行效率相对较低。在Xcode中，确保从Product菜单选择Build for Profiling，这一点非常重要。

### Instruments中内置的分析工具

Instruments内置了4种性能分析工具模板，分别涉及内存管理、CPU使用、I/O操作、图形图像。

#### 内存管理分析工具

和内存管理相关的工具模板如图17-1所示。



1. 与内存管理相关的分析工具模板

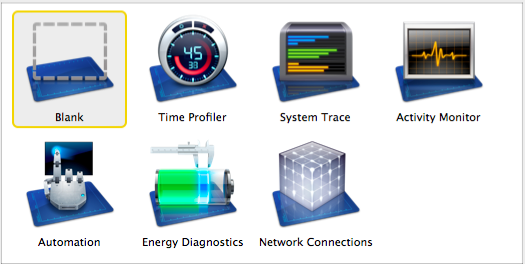
Allocations工具模板通过跟踪内存分配，包括特殊类对象的分配来获取内存使用信息。使用该工具还可以记录虚拟内存统计信息。

Leaks工具模板测量常规的内存使用状况、检查内存泄露情况，并按类提供对象分配的统计信息，所有活动分配的内存地址历史以及和内存泄露相关的块。

Activity Monitor工具模板可以检测系统的总体活动和统计信息，包括CPU、内存、磁盘和网络等。此外，还可以检测当前所有的进程，并将新的分析工具指定到特定的进程中，检测父进程和子进程的层级关系，以及退出当前运行的进程。

#### CPU使用分析工具

和CPU使用相关的分析工具如图17-2所示。



1. 与CPU使用相关的分析工具模板

中Time Profiler工具模板主要用于获取代码CPU上执行所占用的时间。

System Trace工具模板提供系统行为的全面信息，包括线程何时执行，以及通过系统调用或内存操作将用户代码转换为系统级代码。

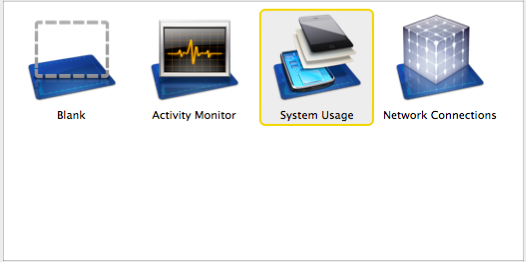
Automation工具模板将对从Instruments运行的iOS应用运行一个脚本，以模拟UI交互。

Energy Diagnostics工具模板提供与电源使用相关的分析信息，包括主要设备部件的开关状态。

Network Connections工具模板提供与使用TCP/IP、UDP/IP连接相关的分析信息。

#### I/O操作分析工具

与I/O操作相关的分析工具如图17-3所示。

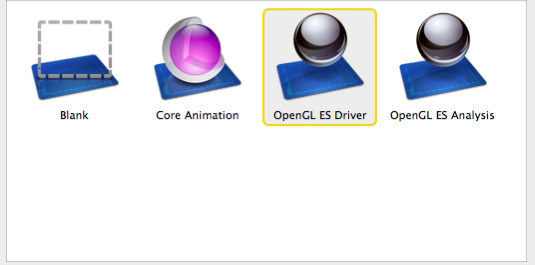


1. 与I/O操作相关的分析工具模板

System Usage工具模板用于记录与文件、sockets和共享内存相关的I/O系统活动，以及每次调用的输入、输出、持续时间、backtrace（函数调用栈）、calltree等。

#### 图形图像分析工具

与图形图像相关的分析工具如图17-4所示。



1. 与图形图像处理相关的分析工具模板

Core Animation工具模板用于测量应用的图形性能，以及进程的CPU占用率。

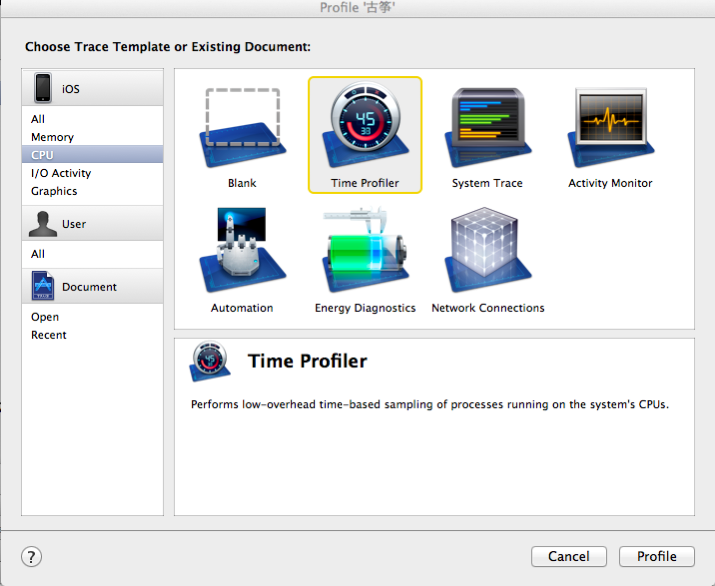
OpenGL ES Driver工具模板用于了解设备GPU的运作，有助于发现游戏性能的不合理之处，后面会详细介绍使用方法。

OpenGL ES Analysis工具模板用于分析OpenGL ES活动，以检测OpenGL ES的正确性和性能问题，同时它还可以提供相关问题的解决建议。

### 如何使用Time Profiler工具模板

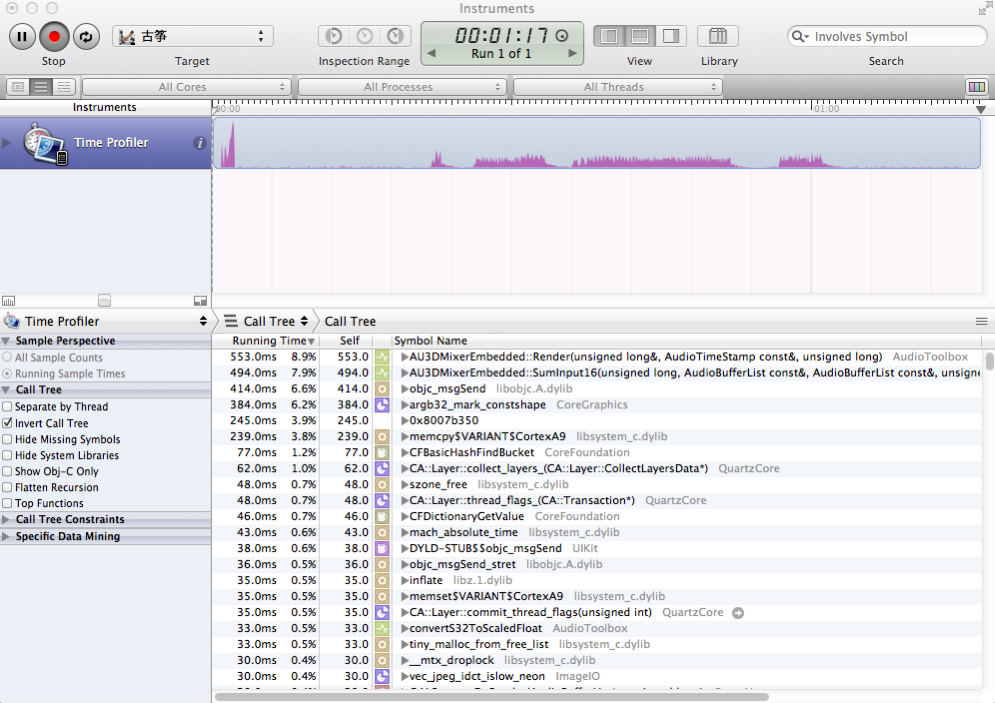
使用Time Profiler模板可以获取设备的CPU利用率，以及各部分代码在CPU执行上所占用的时间。Time Profiler会跟踪代码以及代码所调用的框架或系统类库。默认情况下，信息会显示在Invert Call Tree（反转命令树）上，在顶部显示所调用的代码，并在下方显示调用者。当选中反转命令树时，如果方法A调用了方法B，则会在上方看到B，下方看到A。

如要使用Time Profiler，在Xcode的菜单中选中Product→Profile，此时会弹出如图17-5所示的界面。



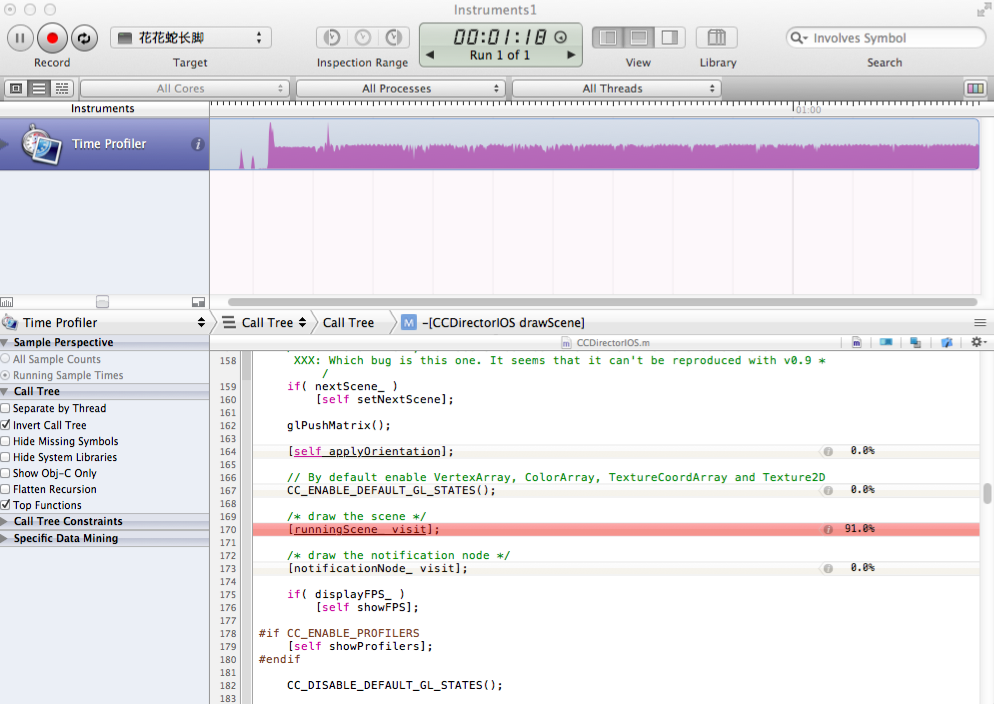
1. 选择Time Profiler界面

选中Time Profiler，单击Profile，此时会在设备上运行应用，并在Time Profiler中显示分析的结果，如图17-6所示。



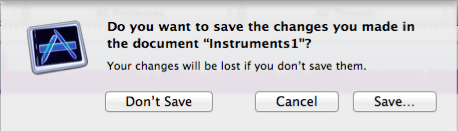
1. 在Time Profiler中显示分析的结果

勾选左侧的Top Functions，右侧的Call Tree就会按CPU占用率来排序。如果对某一处的CPU占用率有疑问，可以双击该行，系统将自动进入相关的代码行，如图17-7所示。



1. 使用Time Profiler查看相关的代码行

在退出Time Profiler的界面时，系统会有如下提示，如图17-8所示。



1. 退出Time Profiler界面时的系统提示

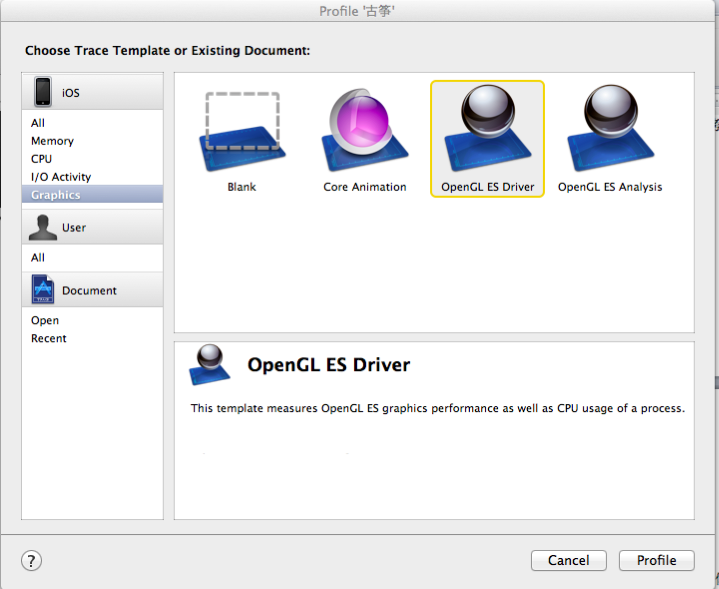
如果需要，可以选择一个路径进行保存，以便后续参考。

通过这样的方式可以有效定位可能影响游戏运行性能的代码段，然后思考如何改进。

### 如何使用OpenGL ES Driver工具模板

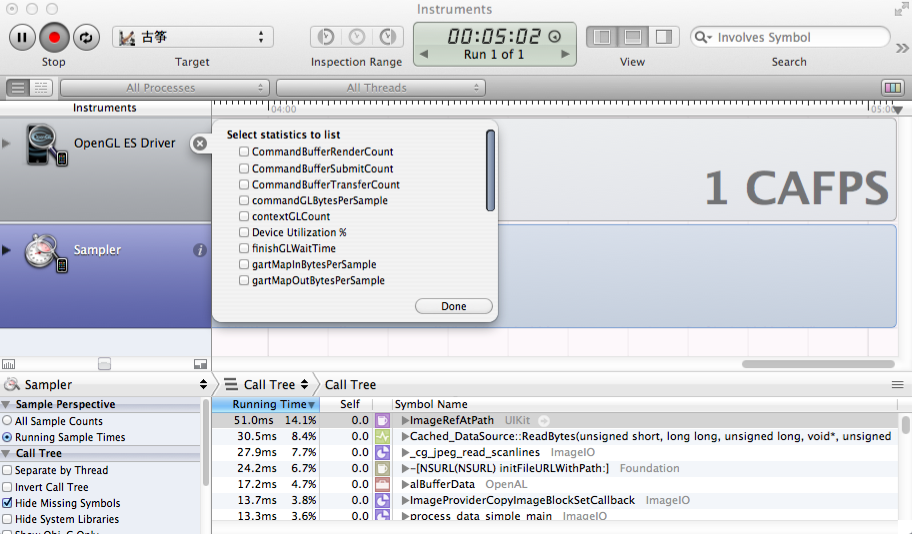
为了方便了解设备GPU的运作，Xcode中提供了一款优秀的OpenGL ES分析工具模板，即OpenGL ES Driver。

在Xcode的菜单中选中Product→Profile，此时会弹出下面的界面，如图17-9所示。



1. 选择OpenGL ES Driver界面

单击OpenGL ES Driver旁边的小i符号，并单击Configure，会看到如图17-10所示的界面。



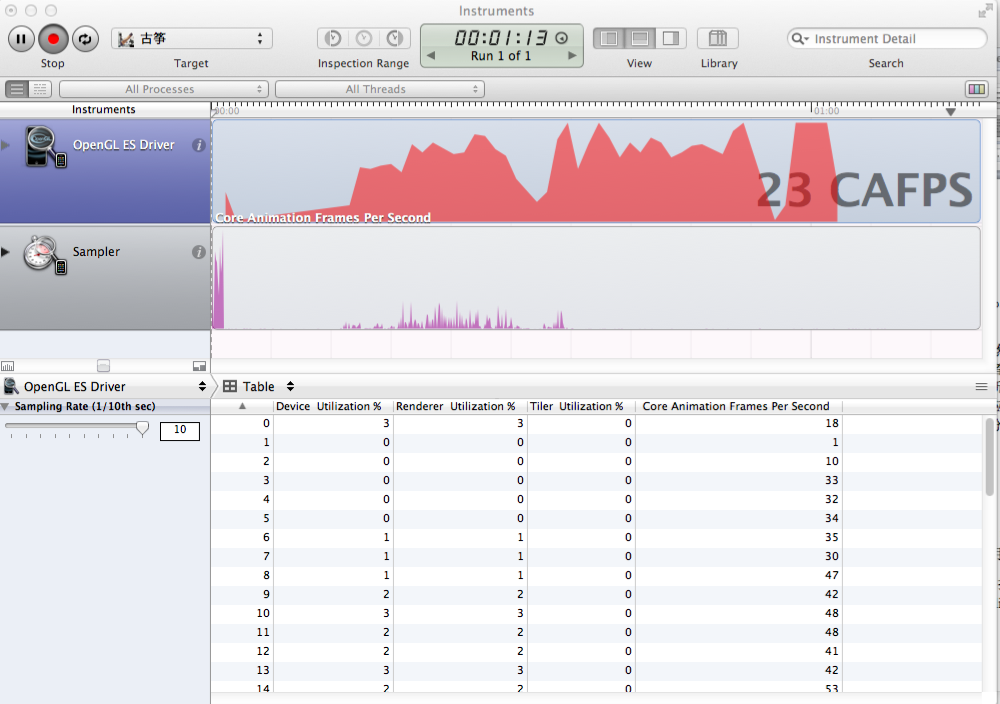
1. 设置OpenGL ES Driver的分析选项界面

其中包含多个选项，这里对其中常用的选项做简要的解释：

* CommandBufferRenderCount：由GPU渲染的3D帧数。
* CommandBufferTransferCount：由GPU处理的图像操作数量。
* CommandBufferSwapCount：由Driver所处理的显示交换命令数。
* CommandBufferSubmitCount：由Driver所处理的命令缓冲。一个命令缓冲可能包含多个渲染和图像操作。
* ContextGLCount： 所有OpenGL context的数量。
* Renderer Utilization：GPU用于像素渲染处理的时间百分比。
* Tiler Utilization：GPU用于处理顶点和栅格的时间百分比。
* Device Utilization：GPU用于处理栅格或渲染工作的时间百分比。
* SplitSceneCount：某个组件需要进入分屏模式的次数。
* CoreAnimationFramesPerSecond：Core Animation每秒钟所形成的新帧数。

通常需要选中Render Utilization、Tiler Utilization、Device Utilization和CoreAnimationFramesPerSecond几个选项。

此时分析结果类似图17-11所示。



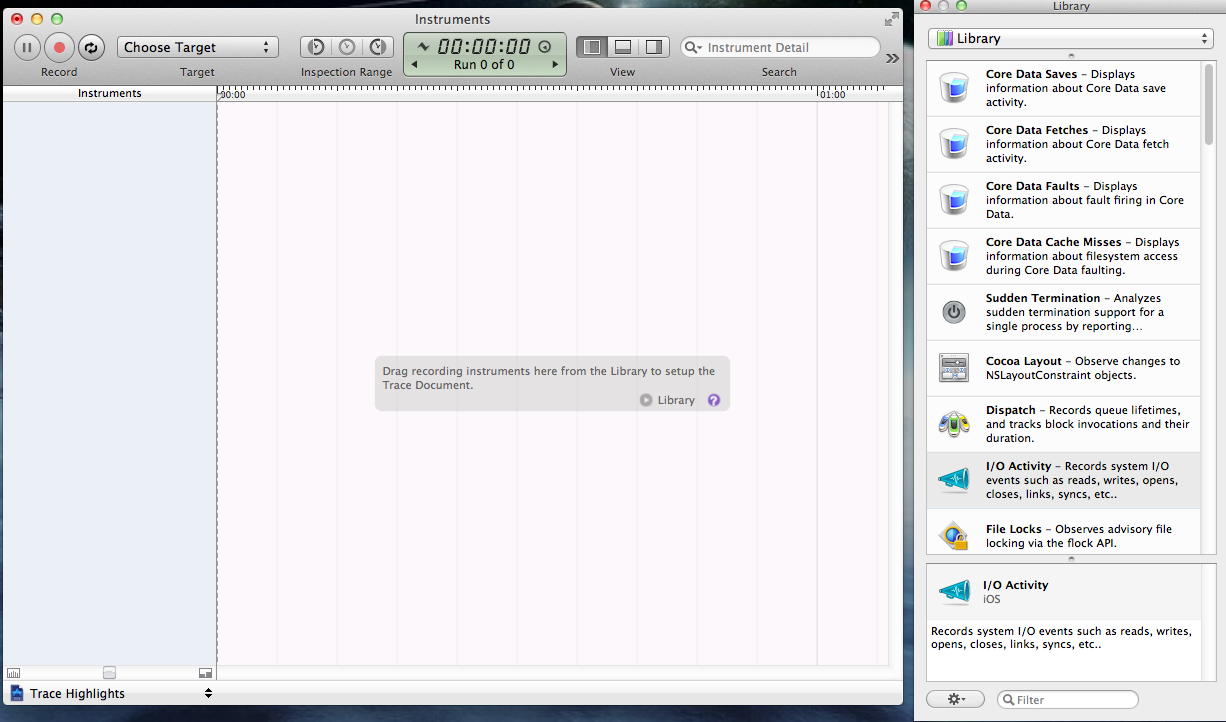
1. OpenGL ES Driver的分析结果显示

通过对相关数据的分析，有助于发现游戏性能的不合理之处。

### 使用空白模板创建定制分析工具

除了使用Instruments内置性能分析工具模板，还可以使用空白模板创建定制化的分析工具。

在Xcode的菜单中选中Product→Profile，选择Blank。在菜单中选择Window，然后单击Library显示性能分析工具库，如图17-12所示。



1. Library中显示的内置分析工具

此时可以从Library中选择需要分析的相关分析工具，并拖动到空白区域。需要注意的是，其中一些分析工具是针对Mac平台软件的，并不适用于iOS设备。

限于篇幅，这里无法对Instruments各个方面的细节都详细介绍，感兴趣的读者可以参考苹果官方的文档：

http://developer.apple.com/library/ios/#documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/Introduction/Introduction.html

## Cocos2D内置的性能分析工具

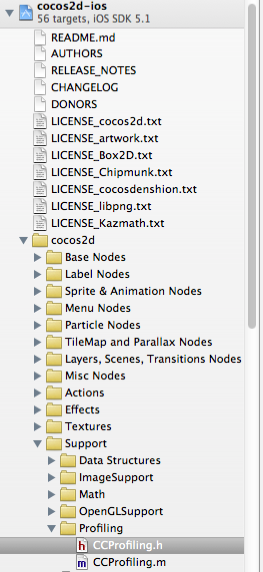
Cocos2D虽然只是一个开源的框架，但仍然内置了一个迷你的性能分析工具，可以帮助开发者判断在 特定方法的执行上所耗费的时间。在进行游戏测试的时候，可能会需要判断某个人工智能方法是否耗费了过多的计算时间，或者寻路算法让游戏中的人物反应缓慢。

本节将要学习如何在Cocos2D的项目中使用内置的性能分析工具。

1. CCProfiler和CCProfilingTimer

Cocos2D中提供了两个类CCProfiler和CCProfilingTimer，用于分析性能。在学习如何使用之前，先看看这两个类的相关定义。

在Xcode中打开cocos2d-ios-xcodeproj，在左侧展开cocos2d，在Support文件夹下展开Profiling，会看到CCProfiling.h和CCProfiling.m两个文件，如图17-13所示。



1. 在项目中找到CCProfiling.h和CCProfiling.m

对CCProfiler和CCProfilingTimer的定义就在其中。CCProfiler的定义如下：

/\*\* CCProfiler

cocos2d builtin profiler.

To use it, enable set the CC\_ENABLE\_PROFILERS=1 in the ccConfig.h file

\*/

@interface CCProfiler : NSObject {

@public

NSMutableDictionary\* activeTimers;

}

/\*\* shared instance \*/

+ (CCProfiler\*)sharedProfiler;

/\*\* Creates and adds a new timer \*/

- (CCProfilingTimer\*) createAndAddTimerWithName:(NSString\*)timerName;

/\*\* releases a timer \*/

- (void)releaseTimer:(NSString\*)timerName;

/\*\* releases all timers \*/

- (void) releaseAllTimers;

/\*\* display the timers \*/

- (void)displayTimers;

@end

可以看到，CCProfiler直接继承自NSObject，类中定义了一个静态类方法，从而可以使用该类的单例对象。同时，它还定义了4个实例方法，其作用分别如下：

1. 创建并添加一个新的计时器：

- (CCProfilingTimer\*) createAndAddTimerWithName:(NSString\*)timerName;

1. 释放一个计时器：

- (void)releaseTimer:(NSString\*)timerName;

1. 释放所有的计时器：

- (void) releaseAllTimers;

1. 显示计时器：

- (void)displayTimers;

CCProfilingTimer的定义如下：

/\*\* CCProfilingTimer

Profiling timers used by CCProfiler

\*/

@interface CCProfilingTimer : NSObject {

@public

NSString \*name;

struct timeval startTime;

double averageTime;

double minTime;

double maxTime;

double totalTime;

NSUInteger numberOfCalls;

}

/\*\* resets the timer properties \*/

-(void) reset;

@end

从定义中可以看到，它是被CCProfiler所使用的性能分析计时器。

该类只有一个实例方法，就是重新设置计时器的属性。同时还类提供了多个公用的实例变量，可以供其他类的对象直接调用。

在CCProfiling.m中可以看到CCProfiler和CCProfilingTimer中方法的实现代码。

1. 如何使用性能分析工具

本节重点学习CCProfiler和CCProfilingTimer这两个类的工作原理，以及如何在项目中使用它们来分析游戏的性能，。

在Cocos2D中使用性能分析工具其实并非难题：

1. 需要对每一条要计算其执行时间的代码创建一个CCProfilingTimer实例，此时需要通过CCProfiler的单例对象来创建。
2. CCProfiler会在控制台每秒输出每个计时器的平均时间。

整个过程就这么简单。当然，这里无法得到命令堆栈的信息，或是任何所耗时间的百分比。但可以得到的信息包括每个时间碎片所包含代码的实际执行时间。比如设置的游戏运行速度是60 FPS（帧每秒），那么每帧最多可以耗费16.67毫秒，而实际的耗用时间对游戏开发者是非常有参考价值的。

在Cocos2D之前的版本中，使用CCProfiler和CCProfilingTimer来进行性能分析相对比较麻烦。首先要在.h文件中声明计时器实例变量，在.m文件中创建并初始化实例变量，然后启动某个实例的计时，接下来需要终止计时，最后需要释放这些实例变量。如果需要在程序中使用不同的CCProfilingTimer，以上步骤将令人无所适从。

在Cocos2D最新的版本中，对CCProfiling.h和CCProfiling.m做了修改。使用NSMutableDictionary记录活跃的计时器，并更改了几个方法。这样一来，使用Cocos2D的内部性能分析工具变得直观明了。

在项目中使用内置的性能分析工具步骤如下。

1. 首先在Xcode中切换到ccConfig.h，在文件中找到CC\_ENABLE\_PROFILES 的相关代码，并修改如下：

#ifndef CC\_ENABLE\_PROFILERS

#define CC\_ENABLE\_PROFILERS 1

#endif

1. 在Xcode中进入项目，找到要进行性能分析的代码段。只需按照以下方式输入相关的代码即可：

#if CC\_ENABLE\_PROFILERS

[[CCProfiler sharedProfiler]createAndAddTimerWithName:@"First Timer"];

CCProfilingBeginTimingBlock(@"First Timer");

#endif

//这里是要进行性能分析的代码段

[self YourMethodsYouWantToBeTimed];

[self YourOtherMethodsIncludedInTheTiming];

#if CC\_ENABLE\_PROFILERS

CCProfilingEndTimingBlock(@"First Timer");

#endif

编译运行项目，CCProfiler将在控制台中每秒输出代码段的平均耗用时间。在Cocos2D开发的应用和游戏中，充分使用内置性能分析工具，将有助于开发者对代码进行针对性的优化。

## 优化Cocos2D游戏性能

为了提升游戏的运行性能，可以使用Instruments和CCProfiler等性能分析工具对代码进行深入的分析。此外，还有一些常用的技巧。本节将介绍一些常用的技巧，以帮助开发者了解如何优化Cocos2D游戏的性能。

1. 在Cocos2D中开启自动引用计数

iOS 5中支持全新特性自动引用计数 （Automatic Reference Counting，ARC），首次在iOS系统中提供了自动内存管理，从而避免因为对retain、release、autorelease这些命令的错误调用而导致内存泄露。实际上，Apple还成功地使得ARC可以向下兼容iOS 4的项目。因此，即便不用iOS 5中的新特性，也必定希望在下一个项目中用到ARC，特别是游戏项目中。

要说明的是，除非Zynga及Cocos2D的原作者从头开始改写Cocos2D，否则就必须在Cocos2D的源代码中禁用ARC。也就是说，Cocos2D自身不支持ARC，但并不意味着使用Cocos2D开发的项目不兼容ARC。

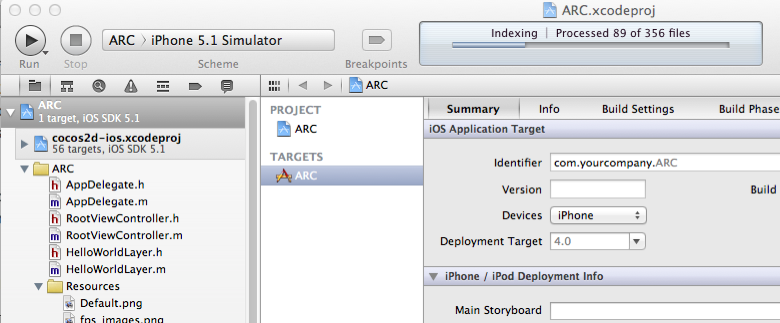
如果使用Cocos2D的模板创建项目，或者仅仅把一些源代码文件拖到Xcode项目中，那么必须通过设置编译器flag的方式对每个.m的Cocos2D源文件禁用ARC，方法如下：

1. 在Xcode中选中项目名称，单击TARGETS下的项目名，然后切换到Build Phases，打开Compile Sources的下三角。
2. 选中（可以多选）想要禁用ARC的文件名，按回车键，输入“–fno-objc-arc”这样的编译器标识。

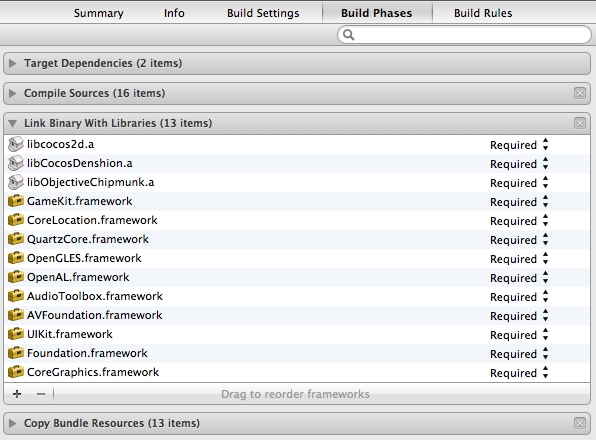
更多关于ARC的信息可以参考：http://clang.llvm.org/docs/AutomaticReferenceCounting.html

当然，还有一个比较简单的方法，就是跨项目引用。在Cocos2D中很容易进行设置。

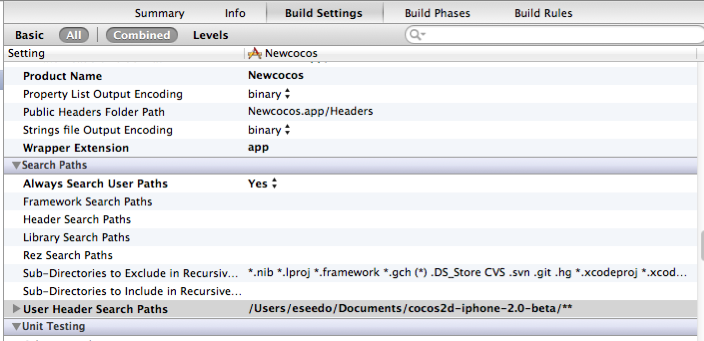
1. 使用Xcode打开所创建的项目，按住键盘上的Ctrl键，并单击选择libs文件夹，然后单击Delete，接着再次单击Delete，以彻底删除这些文件。通过这种方式，就可以把Cocos2D的相关库文件从项目中删除。
2. 在Cocos2D的源文件夹中，找到cocos2d-ios.xcodeproj项目。把这个项目拖进自己创建的Xcode项目，此时它将显示为项目中的项目。如图17-14所示。



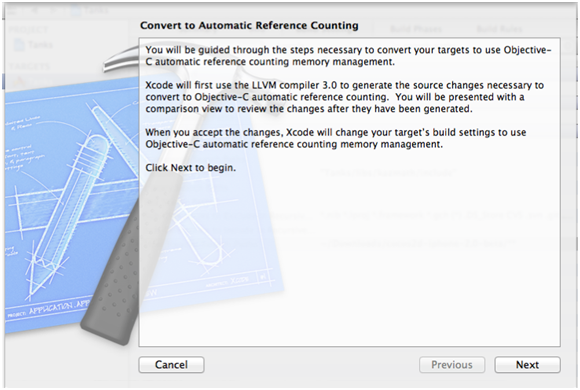
1. 将cocos2d-ios.xcodeproj项目拖进当前项目
2. 进入所创建项目的Build Phases 设置，并在Link Binary With Libraries中添加cocos2d libs，如图17-15所示。



1. 在Link Binary With Libraries中添加cocos2d libs
2. 进入项目的Build Settings，将Always Search User Paths设置为YES，然后将Cocos2D的源文件路径添加到User Header Search Paths中，并确保其标记为Recursive。如图17-16所示。



1. 设置Always Search User Paths和User Header Search Paths
2. 从Xcode的主菜单中选择File→Refactor→Convert to Objective-C ARC，选中下拉列表中的所有文件，并按照转换向导的提示进行操作。如图17-17所示。



1. 使用转换向导让选中的项目文件支持ARC

这样，不必再对所有的Cocos2D源文件设置禁用ARC的编译器flag，只需对Cocos2D项目（以及它所创建的静态库）禁用ARC即可，而项目的其他部分仍然可以继续使用ARC。因为Cocos2D项目默认是禁用ARC的，所以只需确保母项目支持ARC即可。

此时编译项目，仍然会提示ARC相关的问题，无法通过编译。问题在于，开启ARC编译项目时会对所有的头文件检查是否支持ARC。可以按照以下方式手动修改（由于新版本有变化，因此不建议手动修改），也可以直接从这里下载并直接使用修改后的Cocos2D模板（支持ARC）。

使用修改后的Cocos2D项目，在创建Cocos2D对象的时候，编译器将自动处理retain/release这些事情。与此同时，Cocos2D的对象也有自己的内存管理机制。两者有机结合，可以让编写代码更快更轻松。

此外，以上方法对其他的第三方类库同样适用。如上面的截图中有Objective-Chipmunk（Chipmunk物理类库的Objective-C语言版本），只需对某个头文件做一些小的调整，同样可以和ARC完美协作。

1. 使用CCSpriteBatchNode 替代CCSprite

本书2.5节曾对CCSpriteBatchNode做过介绍。本节首先回顾与图像相关的一些基本概念。

#### 图像文件

在Cocos2D中，所有的游戏角色都是以图像的形式存储在iPhone/iPad设备的内存中。通常使用的格式是PNG或JPEG。一旦这些图像被加载入内存，它们将以一种未压缩的纹理格式来存储。PNG是苹果官方推荐的iOS设备使用的图像存储格式。

#### 纹理图

游戏角色的图像文件在使用前必须解压缩，并转换成iPhone和iPad的GPU可以理解的格式，并被加载进RAM（随机存储器）。被加载进RAM的图像称为纹理。GPU可以原生支持一系列的压缩格式，如PVRTC，其他格式则必须存储为未压缩的图像数据。这些数据可以被OpenGL ES用于在屏幕上绘制图像。PNG图像文件虽然在闪存中占用的空间较少，但是因为要解压缩，所以会在内存中占用更大的空间。

#### 纹理图集与精灵表单

为了节省内存空间，并减少纹理中的浪费空间，需要把这些纹理拼合在一起成为一个大的纹理图，称为纹理图集。纹理图集只是一个大的纹理图而已，里面包含了所有的图像。

使用OpenGL ES处理图像时，如果使用纹理图集把所有的图像一次性交给OpenGL ES来处理，那么比把单个图像逐个交给OpenGL ES处理要高效得多。在Cocos2D中通过CCSpriteBatchNode（精灵表单）来使用纹理图集。

游戏开发者最容易遇到的问题就是，当有大量的精灵出现在屏幕上时，游戏的运行变得奇慢无比。如果使用单独的CCSprite精灵对象，当屏幕上有了15或20个精灵的时候，会发现游戏帧速率急剧下降。添加的图像元素越多，帧速率下降的越快。即便不提供任何游戏逻辑机制，只是简单地移动这些CCSprite精灵，随着精灵数量的增加，游戏还是会变得越来越慢。导致这一问题的原因很多，最重要的就是在导入图像时，开发者使用了单独的CCSprite,而不是CCSpriteBatchNode。

简而言之，加载数量众多的精灵对象时会调用过多的OpenGL ES命令，而把每个图像作为单独的纹理来处理已经超过了GPU所能处理的上限。为了解决这一问题，最简单的方法就是使用纹理图集。

仔细看CCSprite的相关代码（CCSprite.h文件）会发现，游戏的每一帧都会调用-(void)draw方法。在draw方法中，每次在屏幕中绘制一个精灵时，都会调用真实的OpenGL ES命令（gl为前缀的方法，详见gl.h）。对于每个精灵，OpenGL ES都需要将纹理图绑定在这个CCSprite上，然后将其绘制在屏幕上（也称为渲染）。

在屏幕上真实显示图像的像素前，还有一件事要做：iOS提供的OpenGL ES Driver将把OpenGL ES的命令转换成GPU可以理解的硬件编码，从而让GPU将图像显示出来。作为一个游戏开发者无需了解更多的驱动细节，只需要明白一点：每一次调用OpenGL ES的命令都将耗费OpenGL ES驱动的CPU时钟。尽可能减少OpenGL ES的调用，游戏的运行将更为顺畅。

CCSpriteBatchNode是Cocos2D中一个特殊的类，其中可包含多个CCSprite。该类处于CCLayer和CCSprite之间。如果想了解更多的知识，可以参考2.5节的相关内容，以及Cocos2D官方文档中所对应的API说明。

1. 提升游戏性能的其他技巧

为了充分提高游戏的运行性能，可以多多尝试以下一些基本技巧：

1. 尽可能减少纹理图集中的空白空间。

拥有一个尺寸很大但空白空间很多的纹理图集和使用单独的精灵一样耗费资源。在程序中使用纹理图集时，要使得纹理图集的尺寸最可能的小，而且尽量减少空白空间。

1. 尽可能使用最小位深。

在Cocos2D中，默认情况下所使用的PNG文件的最小位深是RGBA8888。如果可以让图片使用更小的位深，当然更好不过，同时也需要确保在使用Zwoptex或TexturePacker创建纹理图集时使用更小的位深来保存。

在Cocos2D中设置位深及纹理格式的方法是：在Xcode中切换到AppDelegate.m，找到applicationDidFinishLaunching方法，在以下代码设置不同的值：

// Default texture format for PNG/BMP/TIFF/JPEG/GIF images

// It can be RGBA8888, RGBA4444, RGB5\_A1, RGB565

// You can change anytime.

[CCTexture2D setDefaultAlphaPixelFormat:kCCTexture2DPixelFormat\_RGBA8888];

1. 尽可能使用PVR图像格式。

由PVRTC的纹理可以以压缩格式直接存储在iPhone和iPad GPU中，而PNG、JPG和其他格式的图像文件在存储到GPU的内存前必须是未压缩的。PVRTC图片采用PVRTC协议工作，可以很好地处理真实世界图像和写实风格的美术图片，但是对像素风格的美术图片（比如MineCraft使用的就是像素风格的美术图）和卡通图片的处理并不尽如人意。虽然PVRTC图片会占用更多的磁盘（或闪存）空间，但相对PNG或JPG格式的图片所占用的RAM要少的多。PVRTC纹理图通常每像素的色彩数值是4位或2位。Cocos2D还支持一种特殊格式的PVR图像，称为PVR.CCZ，这种格式的突破已经过gzip压缩，因此占用的磁盘（或闪存）空间更少。Cocos2D将在加载这种格式图片的时候将其解压缩，然后再交给GPU来处理。

1. 使用尺寸更小的图片。

如果项目中所使用的美术资源是矢量图，那么最好在载入时将其尺寸设置为较小，然后在实际使用的使用将其放大。相关代码如下：

CCSprite \*mySprite = [CCSprite spriteWithFile:@"myImage.png"];

mySprite.scale = 2.0f;

在以上代码中，首先加载原始大小的图片，然后将其比例设置为原大小的两倍。

1. 使用FlipX和FlipY翻转图片。

如果在项目中需要横向或纵向翻转图片，请尽量使用FlipX或FlipY，而不要分别存储两个图片，相关代码如下：

mySprite.flipX = YES;

mySprite.flipY = YES;

在以上代码中，通过将原图片的横向翻转和纵向翻转属性设置为YES，获得了翻转后的图片效果。

1. 在游戏初始化的时候就加载美术和音乐资源。

如果在游戏中要用到大量的美术和音乐资源，那么最好在相关的游戏层/场景或关卡初始化的时候就为这些资源分配内存。这样一来，绝大多数的内存分配工作会在游戏开始时完成。尽量不要在游戏进行的过程中加载这些资源。当然，这样做也需要把握一种平衡。如果相关资源太多，那么内存占用问题就会取代游戏运行的性能问题，甚至令开发者更为头疼。

1. 清除未使用的纹理。

在编写游戏的时候，对于不同的游戏场景和关卡，可能会使用不同的纹理图或纹理图集。Cocos2D会对所有已加载的图片或纹理保存一个纹理缓存，这样当需要在项目中使用的时候，无需浪费时间加载。但这样带来的问题是，大量的可用内存被用作保存纹理缓存。

可以考虑在某个游戏场景或关卡结束后手动清除未使用的纹理图。Cocos2D中提供了一个单例CCTextureCache可以帮忙完成此工作，相关的代码如下：

[[CCTextureCache sharedTextureCache]removeUnusedTextures];

在以上代码中，使用CCTextureCache的单例对象清除了所有未使用的纹理图。

1. 重用精灵对象。

在项目中所加载的精灵对象有时候会需要重复使用。某种敌人在被打死后可能会稍后在新的位置出现，某个小物体在跌落出屏幕后可能会需要从屏幕上方再次出现等。如果每次都先清除这些精灵对象，然后再重新生成，会大大影响游戏的运行性能。在Cocos2D中可以将这些可能需要重用的精灵对象重设为最初的状态，停止所有的动作，然后在适当的时候重新使用。相关代码如下所示：

[mySprite stopAllActions];

当然，也可以尝试传统平台游戏开发中的一些性能优化方法，这里不再赘述。

## 本章小结

本章介绍了如何使用Xcode中的Instruments工具进行游戏性能分析；此外还介绍了Cocos2D内置的性能分析工具CCProfiler和CCProfilingTimer，以及如何使用以上工具来发现游戏中的性能瓶颈；最后对如何优化Cocos2D游戏的性能提出了一些建议。

第18章将带领读者学习如何进行用户测试、如何在App Store中发布产品，以及如何让更多的用户接受发布的产品。