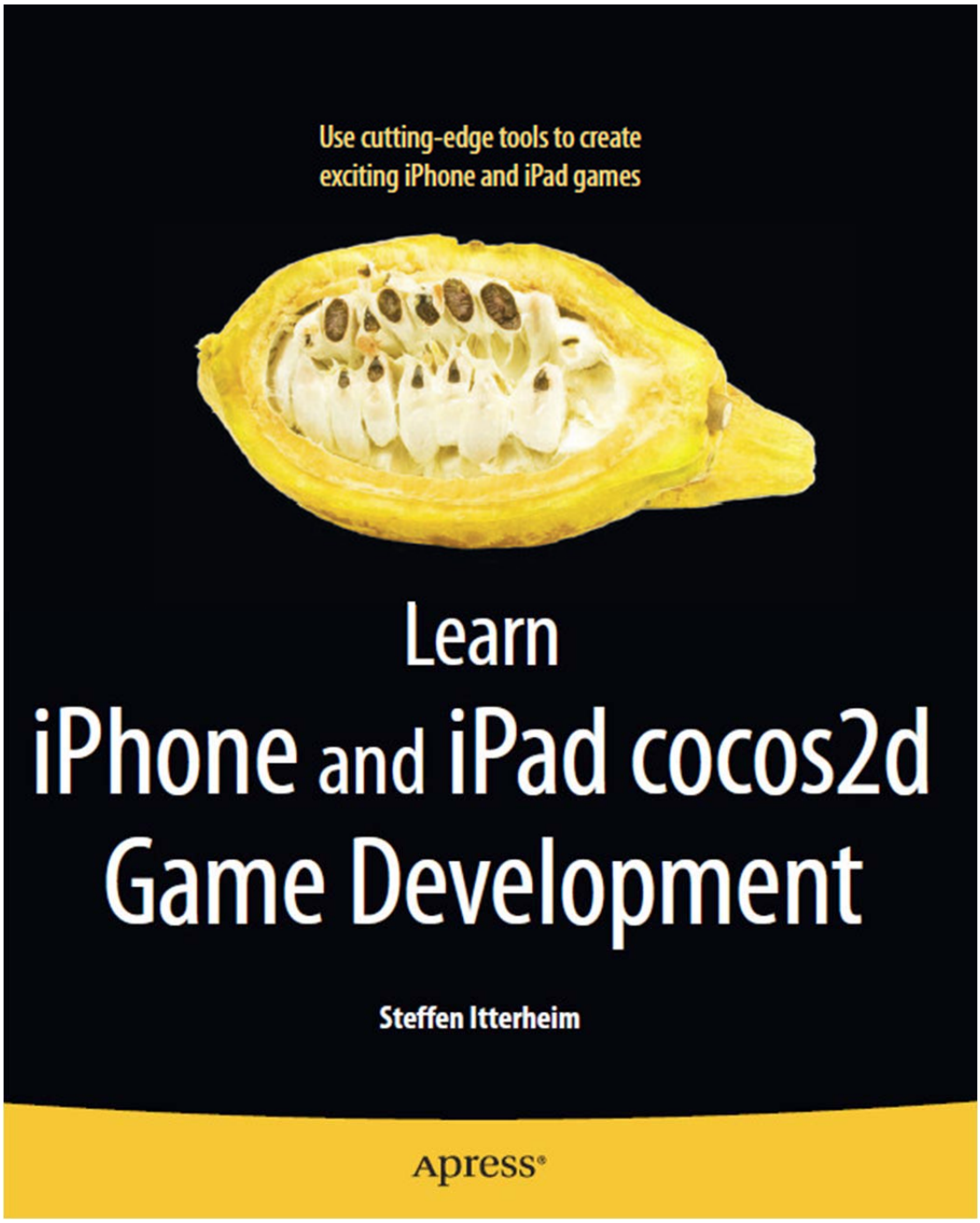

译者：杨栋
邮箱：yangdongmy@gmail.com

第二章
开始学习

我想帮你尽快开始编写 cocos2d 游戏。在本章结束时，你将能够用 Xcode 的 cocos2d 项目模板创建新的项目。我也会介绍一些重要的游戏开发知识。因为许多人对内存管理感到迷惑，所以我将会解释内存管理在 cocos2d 里是如何工作的。希望这样可以帮助你避免一些常见的陷阱。在本章结束时，你将会有一个基于项目模板的，可以正常运行的 cocos2d 项目。

你需要些什么

在这一节，我将告诉你需要哪些东西和通过哪些步骤以开始开发。关于如何注册成为 iOS 开发者和得到 Provisioning Profiles，苹果提供了非常详尽的文档，我就不在这里重复了。

系统要求

以下是开发 iOS 程序对软硬件的最低要求：

1. 基于英特尔 CPU 的苹果电脑，至少要有 1GB 内存。
2. Mac OS X 10.6 (Snow Leopard)或者更高版本。
3. 任何一种 iOS 设备。

任何基于英特尔苹果电脑都可用于开发。甚至Mac mini都可以。不过我建议你配备2GB或者更多的内存。更多的内存会让电脑运行的更流畅，特别是很多游戏开发工具比别的工具都要耗费更多的内存。你将在开发过程中处理大量的图片，声音文件和程序代码，而且很可能所有这些工具将同时运行。

注：在2010年6月推出iPhone OS SDK 4以后，用于iOS开发的操作系统必须是Mac OS X 10.6或以上的版本。如果你还在使用低于10.6版本的Mac OS X版本，请参照Mac OS X技术说明网站（http://www.apple.com/macosx/specs.html）以便知道是否你的Mac符合开发程序的各种要求和如何购买Mac OS X 10.6。

注册成为 iOS 开发者

如果你还没有注册过，第一步就是完成苹果 iPhone 开发者账号的注册。要参加 iPhone 开发者计划，你需要每年支付 99 美元会费。成为付费会员后，你就可以使用 iPhone 开发者门户网站得到 iPhone SDK，并且在门户网站配置开发设备和获取资料用于开发。你也可以使用 iTunes Connect 来管理你的合同和发布应用。

你可以在 iPhone 开发中心（iPhone Dev Center）注册成为 iOS 开发者：
http://developer.apple.com/iphone

证书和 Provisioning Profiles

你最终要把做好的游戏发布到你的 iOS 设备上做测试。要这样做的话你就必须创建一份 iPhone 开发证书（iPhone Development Certificate），注册你的 iOS 设备，然后激活设备用于开发。最后，你要创建 Development 或者 Distribution Provisioning Profiles，将它们下载到本地电脑，然后设置每一个 Xcode 项目以使用它们。所有这些步骤在 iPhone Provisioning Portal 都有详细解释。苹果在每一个 Provisioning Portal 区里的 How To 标签里都有相应的文档解释这些步骤。注册好的 iOS 开发者可以通过以下网址使用 iPhone Provisioning Portal：
https://developer.apple.com/iphone/manage/overview/index.action

下载和安装 iPhone SDK

作为注册的 iPhone 开发者，你可以从开发中心网站下载最新的 iPhone SDK。SDK 文件大概有 2GB 大，安装需要好几分钟。所以在下载之前准备一些咖啡或者热巧克力。因为开发用的 IDE Xcode 也已安装完毕，所以安装完 iPhone SDK 以后你就可以开始开发 iOS 应用了。如果你从来没有用过 Xcode，我建议你先熟悉一下这个开发工具。我推荐 Ian Piper 的“Learn Xcode Tools for Mac OS X and iPhone Development”（Apress，2010）。

注：你很可能想一直处于 iPhone SDK 开发的最前沿。时不时的，Beta 版 iPhone SDK 会被发布出来。除非有很充分的理由，我不建议使用 Beta 版的 iPhone SDK 来开发应用。Beta 版可能有很多问题，他们也可能和目前的 cocos2d 版本不兼容。Beta 版 SDK 都带有 NDA（保密协议），这意味着任何人都不允许在公开场合谈论 Beta 版的 SDK，这让你很难找到解决碰到的问题的途径。

而且，你必须在你的设备上安装 Beta 版的 iOS 操作系统，一旦安装，你将不能回到之前的 iOS 版本。因为在 iOS 设备上安装的应用通常要等到正式的 iPhone SDK 发布以后才会得到更新，所以这些应用很有可能与 Beta 版的 iOS 操作系统不兼容。如果你所做的工作依赖于某些应用，那就不要升级设备的 iOS 操作系统。

下载和安装 cocos2d

下一步是获取 cocos2d。你可以通过以下网址下载：http://www.cocos2d-iphone.org/download。

我建议下载稳定版本。虽然不稳定版本并不意味着崩溃，但是你要把它当成 Beta 版。不稳定版本可能会有一些不完善的地方和未经测试的功能。在你考虑使用不稳定版本之前，请先参考发行说明以确定此版本确实包含你需要的功能。如果没有，请下载稳定版本。

双击下载好的文件，将文件解压缩到你 Mac 里的任何一个地方。取决于你下载的 cocos2d 的版本号，将会生成一个子文件夹叫做“cocos2d-iphone-0.993”或者类似的名称。

安装 cocos2d Xcode 项目模板

现在打开“终端”（Terminal）程序－你可以在 Mac 的应用程序文件夹下的“实用工具”文件夹里找到。或者直接在 Mac 右上角的搜索框里输入“Terminal.app”直接找到它。cocos2d 的 Xcode 项目模板安装需要用到一个 shell 脚本，所以要用到终端。

首先，在终端窗口输入 sudo，紧跟着输入一个空白。打开 Finder 窗口，在 cocos2d 文件夹中找到 install-templates.sh，将其拖拽到终端窗口。这将把文件的路径和文件名直接添加到 sudo 命令的后面。现在的终端窗口中将会有以下的命令：sudo /book/cocos2d-iphone-0.99.3/install-templates.sh

按下回车键，因为脚本要求 root 权限才能运行安装程序，所以终端会要求你输入系统密码。如果没有出错的话，你将看到终端窗口会打印出几行反馈文字。大多数文字以“…copying”开头。如果你看到这样的反馈文字，说明模板已被成功安装。

如果你遇到任何的出错信息，请先确认 sudo 和文件路径之间留有空白。并且确保 install-template.sh 脚本的路径是正确的。如果脚本反馈告诉你模板已经被安装，就按照脚本建议的那样在命令行后面加上-f 参数。这会覆盖之前安装的可能已经过时的 Xcode 项目模板。它不会影响任何已存在的基于 cocos2d 模板的项目。

创建一个 cocos2d 应用

打开 Xcode，选择 File ➤ New Project。在 User Templates 里，你会看到 cocos2d 项目模板（Figure 2-1）。

注：Box2d 和 Chipmunk 应用模板将在第 13 章讨论。如果你现在就想感受一下物理效果的话，你也可以试着用这几个物理引擎模板来创建项目。

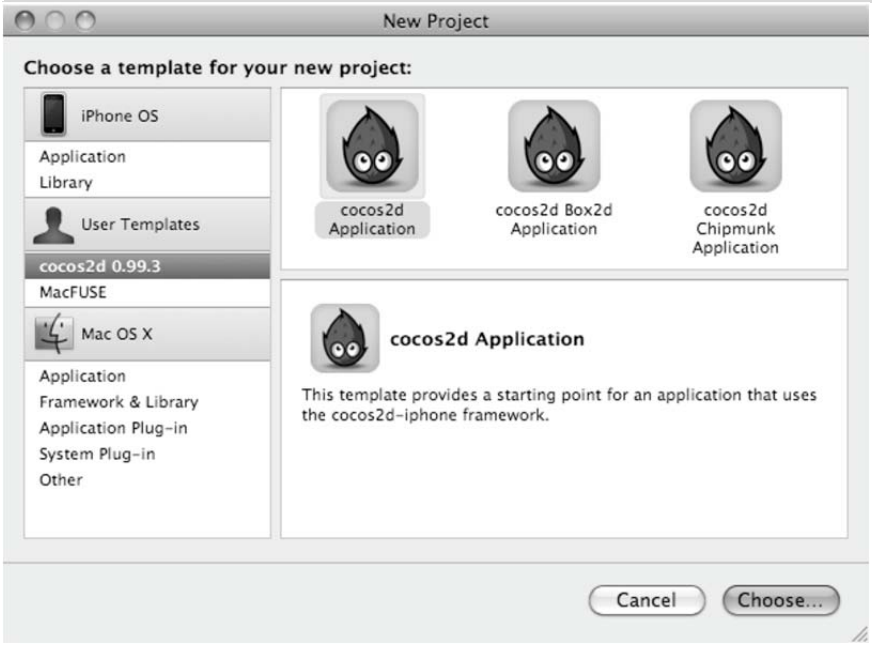

图 2-1. cocos2d Xcode 项目模板

选择“cocos2d Application”模板，项目命名为“HelloWorld”。

注:项目名称中尽量不要使用空格。虽然 Xcode 不介意，但是你用的一些其它工具可能不允许空格。

避免在项目名中使用空格是防守性的措施，以避免任何潜在的问题。过去很长的一段时间里，编写操作系统和应用程序的程序员们依赖不带空格的文件名。虽然在过去的 10 年里，现代操作系统已经允许使用带空格的文件名，但是即使到了今天，有些问题依然与文件名中的空格和特殊字符有关。在处理与代码相关的命名时，我总是避免使用空格或特殊字符。无论是项目命名，源代码文件命名，还是资源的命名。对于开发者使用的文件名，只有数字，减号和下划线是最安全的。

Xcode 将会生成基于模板的项目。下图中的 Xcode 项目窗口将会打开：

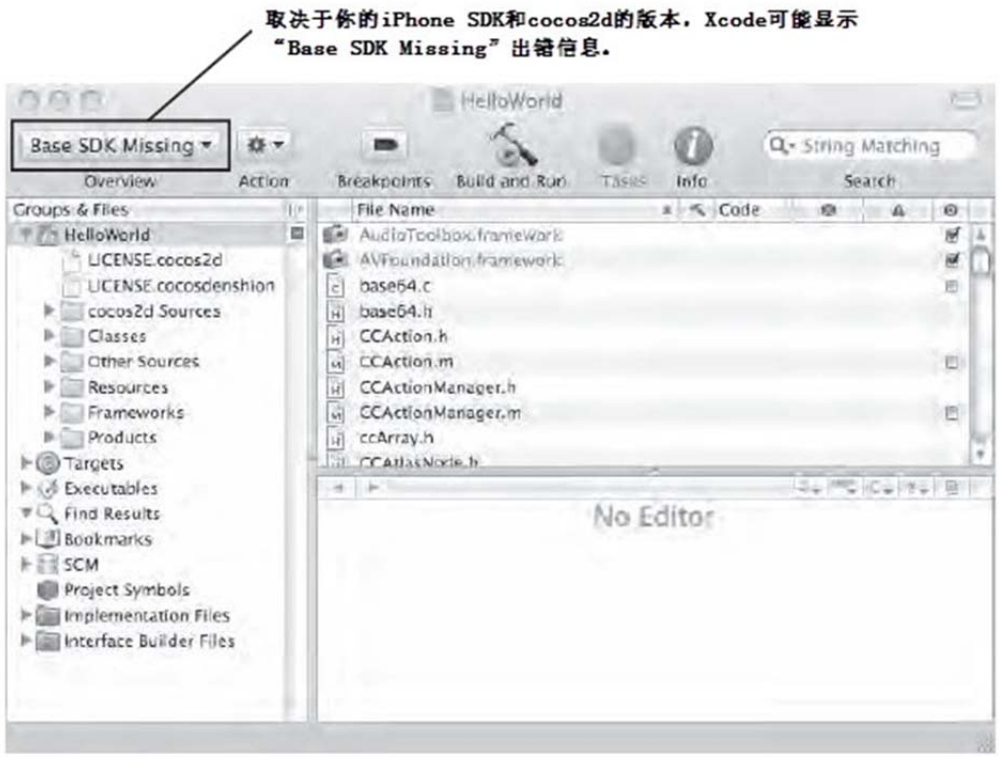

图2-2. HelloWorld项目。取决于你的iPhone SDK和cocos2d的版本，Xcode可能显示“Base SDK Missing”出错信息。

不用太紧张“Base SDK Missing”出错信息。我碰到这个出错信息是因为我使用的iPhone SDK版本是4.0，而当时cocos2d的稳定版本只支持到iPhone SDK 3。这里的“Base SDK Missing”指的是项目模板指向了iPhone SDK 3，而我使用的却是4.0版本。

为了解决这个问题，你可以打开Xcode的“Project”菜单，选择“Edit Project Settings”。如**图2-3**所示，Xcode将会打开一个名为“Project

“HelloWorld” Info”的弹窗。在此弹窗的底部，找到下拉框“Base SDK for All Configurations”。目前下拉框的选项可能是“iPhone Device 3.0 (missing)”，在此选择最新的SDK版本。

注：把Base SDK设置为某个特定的版本，并不代表你的应用只能运行在特定版本的iOS上。对于后者的设置由名叫“iPhone OS Deployment Target”这个“Build Setting”决定。你可以在“Get Info”弹窗里的“Build”标签页中找到这个设置。在那里你可以选择应用程序可以运行的iOS版本，从iOS 2.0到iOS 4.0。

图 2-3. 选择一个存在的 Base SDK 以解决“Base SDK Missing”错误。

现在你可以编译和运行项目了。默认情况下，iPhone 模拟器会被启动，如**图 2-4** 所示。

图 2-4. 成功！模板项目编译成功并且在 iPhone 模拟器里运行显示“Hello World”标签。

HelloWorld 应用

我们不费很多力气就创建了一个可以运行的 cocos2d 应用程序。但是现在你想知道这个程序是怎样工作的，对吧？我没有想过你会那么容易放过我，因为即使我讲的多么深入，你还是想知道更多的事情。这是很好的学习精神！

让我们看一下到底 Hello World 这个 Xcode 项目是如何工作的，这样你就可以了解这些代码是如何连接起来的。

找到 HelloWorld 的项目文件

首先，简单介绍一下Xcode。默认情况下，如**图2-5**所示，你会看到Xcode项目窗口的左边栏是“Groups&Files”。Xcode在这里显示所有的项目文件，还有一些其他的东西像目标（Targets）和可执行文件（Executables）。现在我们只要关注HelloWorld项目下的Groups&Files就可以了。

图 2-5. Xcode 的 Gropus&Files 边栏。展开的组里面包含我们将要讨论的项目文件。

在 cocos2d Sources 组里，你将看到所有组成 cocos2d 游戏引擎的文件。虽然你不需要知道 cocos2d 游戏引擎的细节，但是可以随时查看游戏引擎的源代码对 Debug 很有好处的。你也可以通过这些源代码了解一下引擎的工作原理 。

注： Xcode 的 Groups&Files 边栏看起来像 Finder 里的文件夹和文件。但是不要误解 Xcode 里面群组（Groups）的意思，它和 Finder 里的文件夹是两个不同的概念。你可以在 Xcode 里面随意安排群组，但是这些文件在 Finder 面还是在同一个文件夹里面。这就是为什么称它们为群组：你可以在 Xcode 里任意地安排文件，而不用理会这些文件到底是存放在硬盘的哪个位置。

资源

在左边栏“Groups&Files”中，让我们从下往上看。在资源组（Resources group），你可以看到所有非源代码文件，例如图片和音频文件。

iOS 加载应用程序的时候会使用“Default.png”，“Icon.png”则用作应用程序的图标。“fps_images.png”是 cocos2d 用来显示帧率的，即每秒显示的帧数：你不能删除或者修改此图片。

“info.plist”文件包含了一些与应用程序相关的配置信息。你只需要在快要发布应用时才修改这些配置信息。

其它源代码

如果你熟悉 C 或类似的编程语言，你可能在“Other Sources”中认出应用程序的起始程序 main.m。

Main.m

任何在主函数和HelloWorldAppDelegate类之间做的处理，属于iPhone SDK的后台魔法，你没有控制的权限。因为你几乎不需要改变main.m，所以你可以忽略它。不过，看一看也无妨。

简单点说，主函数创建了一个NSAutoreleasePool（自动释放池），然后调用UIApplicationMain启动程序，程序使用HelloWorldAppDelegate类来执行UIApplicationDelegate的协议（protocol）。

```
int main(int argc, char *argv[]) {
     NSAutoreleasePool *pool = [NSAutoreleasePool new];
     int retVal = UIApplicationMain(argc, argv, nil,
                                          @"HelloWorldAppDelegate");

     [pool release];
     return retVal;
}
```

上述代码中使用了一个NSAutoreleasePool来帮助你管理内存。说的简单一点就是通过使用autorelease（自动释放）信息，你不再需要担心忘记发送释放信息。Autorelease pool（自动释放池）可以确保内存中的自动释放对象最终会被释放。

如果你现在不明白我在说些什么东西，不用紧张。我会在本章的后面部份介绍 cocos2d 的内存管理机制。你会明白为什么每个 iOS 应用程序都被包含在 NSAutoreleasePool 里面。

预编译前缀头文件

“HelloWorld_prefix.pch”头文件的作用是给编译过程加速。你应该把不常变化的框架（Frameworks）头文件添加到前缀头文件（prefix header）中。这样的话，在编译的时候，框架的代码会被预先编译，所有的类都将可以使用这些头文件。不幸的是，这样做也有一个缺点：如果前缀头文件里其中一个头文件发生了变化，你的所有代码将会重新编译。这就是为什么你应该只添加那些极少或者从来都不变化的头文件到前缀头文件中。

例如，就像我在**列表2-1**中所做的那样，cocos2d.h头文件可以被添加到前缀头文件中，因为它很少改变。不过，只有复杂一些的项目才会感受到编译速度的

变化。不过，即使只为了不再需要在其他源代码里写 #import “cocos2d.h”，也是值得一开始就在前缀头文件里写进 cocos2d.h 的。

列表2-1：把cocos2d.h头文件加到前缀头文件中

```
#ifdef __OBJC__
    #import <Foundation/Foundation.h>
    #import <UIKit/UIKit.h>
    #import "cocos2d.h"
#endif
```

如果你想了解更多关于如何使用前缀头文件减少编译时间的知识，可以参考苹果开发者文档：
http://developer.apple.com/mac/library/documentation/DeveloperTools/Conceptual/XcodeBuildSystem/800-Reducing_Build_Times/bs_speed_up_build.html

类

有两个类组成了HelloWorld项目的核心。“HelloWorldAppDelegate”类：用于处理程序的全局事件和状态变化。“HelloWorldScene”类：包含了所有用于显示Hello World标签的代码。

HelloWorldAppDelegate

每个 iOS 程序都有一个 AppDelegate 类用于实现 UIApplicationDelegate 协议（protocol）。在我们的 HelloWorld 项目中叫做 HelloWorldAppDelegate。这个命名规则对每个新项目都是一样的：项目名加上 AppDelegate。从现在开始，我会简称它为 AppDelegate，因为你可以在任何一个 iOS 程序里找到它。

AppDelegate 通过在某些时间点从 iOS 接收信息来跟踪程序的状态变化。例如，你可以用它确认是否有打进来的电话，或者可用的系统内存已经不够用。程序开始运行后收到的第一个信息是 applicationDidFinishLaunching，在这个方法中，你可以放置启动代码，cocos2d 就是在这里初始化的。

如果你想学习更多 AppDelegate 相关的方法，怎样使用它们和 iPhone SDK 在什么时候发送信息，你可以参考苹果官方文档中的 UIApplicationDelegate 协议：
http://developer.apple.com/iphone/library/documentation/uikit/reference/UIApplicationDelegate_Protocol

注:既然我们现在讲的是程序启动，我也顺便讲一下程序关闭。你会注意到 AppDelegate 中 dealloc 方法存在一个奇怪的地方：这个方法不会被调用！任何在 AppDelegate 的 dealloc 方法中设置的断点都不起作用！这是正常的。当 iOS 关闭一个程序时，它只是简单的把内存清空，以加快关闭的速度。这也是为什么 AppDelegate 的 dealloc 方法中的任何代码都不会被运行。你并不需要手动调用 dealloc 方法以“解决这个问题”。如果你确实需要在程序关闭之前在 AppDelegate 中运行代码，你可以在 applicationWillTerminate 方法中运行

代码。如果你的目标 iOS 是 4 或者更高的版本，你应该使用 applicationDidEnterBackground。

大多数情况下，在cocos2d的初始化过程中，只有三个设置你可能会改变：

```
[[CCDirector sharedDirector] setDeviceOrientation:CCDeviceOrientationLandscapeLeft];
[[CCDirector sharedDirector] setAnimationInterval:1.0/60];
[[CCDirector sharedDirector] setDisplayFPS:YES];
```

在接下去的内容中，我会提供上述三个设置的一些细节。

设备方向

最重要的一个设置是设备方向的设置。HelloWorld程序使用了横向设置，这意味着你将会横着拿iOS设备来看屏幕。如果你把设置从CCDeviceOrientationLandscapeLeft改为CCDeviceOrientationLandscapeRight,你会发现“Hello World”标签文字将以头朝下的方向显示。

以下是iOS支持的设备方向。每一个都试一遍，看看它们是如何显示Hello World文字的：

```
CCDeviceOrientationPortrait
CCDeviceOrientationPortraitUpsideDown
CCDeviceOrientationLandscapeLeft
CCDeviceOrientationLandscapeRight
```

注：你可以在以后设置设备的方向，甚至在玩游戏的时候。例如，你可以把方向作为一个用户可以改变的游戏设置。如果从一种横向模式换到另一种横向模式，或者从一种纵向模式转换到另一种纵向模式，你不需要修改代码。允许用户选择两种横向模式中的一种，或者两种纵向模式中的一种，实现起来很简单。因为每个人都有不同的习惯，所以允许用户选择普通的或者上下颠倒的方向是很有必要的。

动画间隔（Animation Interval）

动画间隔决定了cocos2d更新屏幕的频率。此设置影响游戏可以达到的最大帧率。不过，动画间隔并不等于多少帧每秒。正相反，它表示的是cocos2d更新屏幕的频率。这就是为什么它的参数是1.0/60 - 因为1除以60等于0.0167秒，这是在两次屏幕更新之间的时间间隔。当然，如果你的游戏很复杂，CPU或者GPU将需要更多的时间来显示下一帧，你的游戏就不可能保证从始至终都达到60帧每秒的帧率。实际上，保持游戏运行在一个高的帧率上是你的责任。贯穿本书，我将为你介绍各种提高游戏运行性能的技术。

在某些时候，把帧率设为30帧每秒可能更合适。对于那些很复杂的游戏，它们的帧率可能在30和60帧每秒之间上下浮动的很厉害，设置一个低一点的帧率会对这样的游戏有所帮助。当你设置了一个低一点的帧率，而且游戏可以稳定的保持在这个帧率，用户的体验会比使用一个高一些但是不稳定的帧率要好很多。

人的感觉是很复杂的东西。

注： iOS设备不支持超过60帧每秒的帧率，它的屏幕刷新率被锁定在60帧每秒（Hz）。如果强迫cocos2d以超过60帧每秒的速度进行渲染，在最好的情况下，可能会不产生任何效果。在最差的情况下，你的帧率可能反而会下降。如果你想让cocos2d运行在最快的帧率下，把动画间隔设置为1.0/60。

显示帧每秒（FPS）

启用 FPS 显示后，在屏幕的左下角将会显示一个小数字。这是你程序的运行帧率，也就是每秒屏幕会被刷新的帧数。理想状态下，你的游戏应该运行在 60 帧每秒的帧率，特别是那些动作游戏。有一些游戏，比如大多数的益智游戏，30 帧每秒就可以满足要求。FPS 显示可以帮助你跟踪当前的帧率和任何可能的问题。

注： 如果你想调节 FPS 显示的反应速度，你可以修改“ccConfig.h”中的 CC_DIRECTOR_FPS_INTERVAL 。你可以在 Xcode Sources/cocos2d 组找到这个文件。默认情况下是 0.1，这表明帧率的显示每秒会被更新 10 次。如果你增加这个值，FPS 显示的更新将会变慢。但是你将不会看到任何突然的改变，虽然还是可以看到一些变化。

HelloWorldScene

cocos2d使用HelloWorldScene类来显示Hello World标签。在我详细解释之前，你应该首先了解cocos2d是使用了多个层级的CCNode对象来决定在什么地方显示什么内容的。

所有节点的基类都是CCNode类。它包含了位置信息，但是没有显示信息。它是所有其他节点类的父类，包括两个最基本的类：CCScene和CCLayer。

CCScene是一个抽象的概念，它的功能是根据像素坐标把物体放置在场景里相应的地方。所以任何cocos2d场景都会用一个CCScene作为父对象。

CCLayer类本身并不做什么，它的功能是允许触摸和加速计的输入。因为大多数游戏会接受基本的触摸输入，所以CCLayer通常是第一个被加入CCScene的类。

如果你打开HelloWorldScene.h头文件，你可以看到HelloWorld类是继承自CCLayer类的。

因为CCScene只是一个抽象的概念，默认的设置场景的方法是在你的类里面使用一个静态初始化方法（static initializer）+(id) scene。此方法会生成一个CCScene对象，并且将HelloWorld的对象添加到场景节点中。几乎在任何情况下，CCScene都是在这里创建和使用的。以下是一个通用的+(id) scene方法：

```
+(id) scene
{
```

```
	CCScene *scene = [CCScene node];
	id node = [HelloWorld node];
	[scene addChild:node];
	return scene;
}
```

首先，CCScene类的静态初始化方法+(id) node生成一个CCScene对象。接下来，同样的+(id) node方法生成了HelloWorld节点，并被添加到场景中。然后场景被返回给调用者。

我们接着来看看**列表2-2**的-(id) init方法。你可能会注意到一个有些奇怪的地方：self = [super init]这个调用中，发送给super对象的init信息所返回的值被赋给了self。如果你有C++的编程经验，你可能会对此很不理解。不需要沮丧！这是因为Objective-C必须手动调用super类的init方法。不存在对父类的自动调用。而且我们必须把[super init]的返回值赋给self，因为我们有可能得到一个空值（nil）：

列表2－2. init方法生成Hello World的标签对象，并将它添加到场景层中

```
-(id) init
{
	if ((self = [super init])) {
		// 生成并初始化标签对象
		CCLabel* label = [CCLabel labelWithString:@"Hello World"
                                          fontName:@"Marker Felt" fontSize:64];
		// 从CCDirector得到窗口的尺寸
		CGSize size = [[CCDirector sharedDirector] winSize];

		// 将标签放在屏幕中央
		label.position = CGPointMake(size.width / 2, size.height / 2);

		// 将标签作为子节点添加到场景层中
		[self addChild: label];
	  }
	  return self;
}
```

如果你很介意上述[super init]的写法，以下是另一种写法。它的作用和上面的写法完全一样。

```
-(id) init {
	self = [super init];
	if (self != nil) {
		//在此添加init方法的代码
	}
	return self;
}
```

现在我来解释 Hello World 标签是如何添加到场景中的。如果你再看一遍**列表2-2** 中的 init 方法，你会看到 CCLabel 对象是通过 CCLabel 的 init 静态初始化方法生成的。这个静态方法会返回一个新的 CCLabel 对象，而且此对象是一个自动释放的对象。在 init 方法运行完成以后，为了不丢失内存中已经存在的 CCLabel 对象，你必须把此对象添加到 self 中，在此使用的方法是[self

addChild:label]。同时标签也被赋予了位置信息－屏幕的中央。有一点值得注意的是：你可以在调用 addChild 之前，也可以在调用之后赋予位置信息。

cocos2d 的内存管理

接下去我们此讨论内存管理和自动释放信息（autorelease message）。

通常，当你在 Objective-C 里生成一个对象时，你会调用 alloc。如果这样做的话，你就要在不再需要此对象时释放生成的对象。以下是一个典型的生成/初始化（alloc/init）和释放（release）的循环：

```
// 生成一个NSObject实例
NSObject* myObject = [[NSObject alloc] init];

// 用myObject做些事情…
// 释放myObject占用的内存
// 如果你不释放的话，此对象占用的内存将泄漏，并且那部份内存将永远都不会被释放
[myObject release];
```

因为iOS程序总是会使用一个自动释放池（autorelease pool）和自动释放信息，所以你可以避免发送任何释放信息。以下是用自动释放方式重写的例子：

```
//生成一个NSObject实例
NSObject* myObject = [[[NSObject alloc] init] autorelease];
//用myObject做些事情…
// 不需要调用释放，实际上你不应该在这里发送释放信息，因为会让程序崩溃。
```

你可以看到，这样做简化了内存管理，你也不再需要记着发送释放信息给对象。自动释放池会记着给你的对象在晚一些时候发送释放信息。你所要做的只是在生成对象时加上自动释放信息。现在看一下以下代码，这些代码是传统的CCNode对象释放模式：

```
// 分配一个新的CCNode实例
CCNode* myNode = [[CCNode alloc] init]];
// 用myNode做些事情 …
[myNode release];
```

这不是我们喜欢的生成cocos2d对象的方式。使用静态初始化方法来生成可自动释放的cocos2d对象会更简单。和苹果官方的建议相反，cocos2d引擎通过把 [[[NSObject alloc] init] autorelease] 这样的调用放进类本身的静态方法中，来达到使用自动释放对象的目的。这实际上是件好事！因为你不用再记住那些需要释放的对象了。因为忘记释放内存，要么会导致过度释放对象而引起的程序崩溃，或者因为没有释放所有应该释放的内存而导致内存泄漏。

以CCNode类为例，它的静态初始化方法是 +(id) node。以下代码会将分配信息（alloc）发送给self，这和在CCNode里面使用[CCNode alloc]的效果是一样的：

```
+(id) node
{
        return [[[self alloc] init] autorelease];
}
```

只是上述代码在这样的情况下更加普遍。

现在我们可以用静态初始化方法来重写CCNode的生成方法。不出所料，代码变得很简洁：

```
//分配一个新的CCNode实例
CCNode* myNode = [CCNode node];
//用myNode做些事情 …
```

这就是使用自动释放对象的好处。你不再需要给对象发送释放信息。每一次cocos2d进入下一帧，那些不再用到的自动释放对象将被自动地释放。但是这样做也有一个缺点。如果你用了这里的代码，然后在下一帧或者以后想要访问myNode对象，你会发现它已经不存在于内存里了。如果这时你发送信息给它，将会导致程序出现EXC_BAD_ACCESS错误而崩溃。

简单地把CCNode* myNode变量当作你的类成员变量，并不意味着对象用到的内存会被自动保留下来。如果你想在下一帧或者以后的帧里面访问自动释放对象，你必须保留（retain）它，并且如果你没有将它作为子节点（child node）添加到场景中的话，你还是要手动释放它。

有一个方法可以更好地使用自动释放对象而不需要明确地调用保留（retain）方法：你可以通过将生成的CCNode对象作为子节点添加到另一个继承自CCNode的对象，以达到添加到场景层级（scene hierarchy）中的目的。

你甚至可以不用成员变量，而直接依赖cocos2d来为你保存对象。

```
// 生成一个自动释放的CCNode实例
-(void) init
{
        myNode = [CCNode node];
        myNode.tag = 123;
        // 将此实例作为子节点添加给self（假设self继承自CCNode）
        [self addChild:myNode];
}
-(void) update:(ccTime)delta
{
        // 之后我们就可以在这里访问和使用myNode对象了
        CCNode* myNode = [self getChildByTag:123];
        //用myNode做些事情 …
}
```

这里的 addChild 将生成的 CCNode 对象添加到了一个集合中，这里是使用了 CCArray。CCArray 和 iPhone SDK 的 NSMutableArray 类似，但是速度上快一些。CCArray 和 NSMutableArray，还有任何其它 iPhone SDK 集合会给每一个添加进来的对象发送 retain 信息，也会给没有个删除的对象发送 release 信息。所以，这样生成的对象可以一直存在，直到从集合里被删除。在集合里被删除以后，对象会被自动释放。

通过上述方法给 cocos2d 对象做内存管理是最好的方式。有的开发者可能会告诉你自动释放不好或者速度很慢。你不该被它们说服。

注：苹果开发者文档建议减少使用自动释放对象。但是大多数 cocos2d 对象都是自动释放对象。这样做的好处是让内存管理变的简单。

如果你用传统的 alloc/init 和 release 来管理每一个 cocos2d 对象的话，那将是自找麻烦，而且得不到任何好处。这也不是说你永远都不会用到 alloc/init；有时候它们可能是必须的。但是对于 cocos2d 对象来说，你应该依赖于静态自动释放初始化方法（static autorelease initializers）。

自动释放对象只有一个缺点，那就是直到游戏进入下一帧，这些对象将一直占用内存。这就意味着，如果你在每一帧都生成许多很快就要被丢弃的自动释放对象，你可能会浪费很多内存。不过这样的情况很少发生。

我们在这里结束cocos2d内存管理的快速介绍。Objective－C的内存管理由以下两条规律约束着：

1. 如果你拥有（通过alloc，copy或retain得到）一个对象，你必须在用完之后释放它。
2. 如果你已经给一个对象发送了自动释放信息，你将不能release它。

如果你想了解更多的内存管理方面的知识，请参考苹果的“内存管理编程指南”（Memory Management Programming Guide）：
http://developer.apple.com/iphone/library/documentation/Cocoa/Conceptual/MemoryMgmt/MemoryMgmt.html

改变世界

如果我们只是停留在由模板生成的HelloWorld，那就太没有意思了。我们来改点代码让它变的有意思一些。

首先，在init方法中做两处修改：一是启用触摸输入，二是设置一个以后可以调用标签对象的tag。在**列表2-3**中标出了这两处修改：

列表2-3. 启用触摸输入和给标签对象设置tag

```
-(id) init
{
        if ((self = [super init])) {
                // 生成和初始化一个标签对象
                CCLabel* label = [CCLabel labelWithString:@"Hello World"
                fontName:@"Marker Felt" fontSize:64];

                // 从CCDirector得到当前屏幕的尺寸
                CGSize size = [[CCDirector sharedDirector] winSize];

                // 将标签定位在屏幕中央
                label.position = CGPointMake(size.width / 2, size.height / 2);

                // 将标签添加到层
```

```
            [self addChild: label];

            // 我们的标签需要一个tag，这样以后才能找到它
            // 你可以使用任何数字
            label.tag = 13;

            // 如果你想收到触摸事件的话，你必须在这里启用它！
            self.isTouchEnabled = YES;
        }
        return self;
}
```

标签对象的tag属性被设定为13。这样你就可以在以后用这个数字来访问标签对象。tag数字必须为正整数，并且每个对象的tag不能是一样的。如果出现了两个一样数字的标签，系统就不知道该拿哪一个对象了。

贴士：你应该使用常量而不是像13这样的具体数字作为tag值。同kTagForLabel这样的变量名相比，你很难记住13这个tag到底意味着什么。我将在第5章具体讨论。

其次，self.isTouchEnabled被设为YES。这是CCLayer的一个属性，它告诉系统你的程序想接受触摸信息。只有这样ccTouchBegan才会被调用：

```
-(void) ccTouchesBegan:(NSSet*)touches withEvent:(UIEvent*)event;
{
        CCLabel* label = (CCLabel*)[self getChildByTag:13];
        label.scale = CCRANDOM_0_1();
}
```

通过使用[self getChildByTag:13]，你可以使用tag属性访问你在init方法中设置的CCLabel对象,然后开始使用此对象。在这个例子里，我们使用了cocos2d的CCRANDOM_0_1()宏来改变标签的大小属性，使它在0和1之间变化。每次你点击屏幕都会让标签尺寸发生变化。

因为getChildByTag总是会返回标签，我们可以将它转换成一个 CCLabel* 对象。不过，如果返回的不是继承自CCLabel的标签对象，你的游戏会崩溃。如果你不小心将另一个对象的tag值设为13的话，这样的事情很容易发生。因为这个原因，我们应该在编程过程中总是验证得到的对象。你可以使用NSAssert方法来验证是否得到了你想要的对象：

```
-(void) ccTouchesBegan:(NSSet*)touches withEvent:(UIEvent*)event;
{
        CCNode* node = [self getChildByTag:13];

        //防御性编程：验证返回的节点是CCLable类的对象
        NSAssert([node isKindOfClass:[CCLabel class]], @"node is not a CCLabel!");

        CCLabel* label = (CCLabel*)node;
        label.scale = CCRANDOM_0_1();
}
```

在上述例子中，我们希望getChildByTag返回一个继承自CCLabel的对象，但是我们不能确定。所以通过使用NSAssert来验证返回的对象，以及时发现错误避免程序崩溃。

这样做虽然多写了两行代码，但是代码的执行效率是一样的。因为在最终的发布版本中，NSAssert将被删除。这两个版本的执行效率是完全一样的，但是第二个版本的好处是：如果你没有得到希望的CCLabel对象，你会得到EXC_BAD_ACCESS报错而不是程序崩溃。

其他一些你需要知道的事情

因为本章是开篇章节，我想借这个机会介绍一些iOS游戏开发中非常重要但是又经常被忽略的方面。特别是，人们经常不知道到底有多少内存可以被使用，从而做出错误的判断。实际上你只能安全地使用设备上很少一部分的内存。

我也想让你知道,iPhone模拟器（Simulator）可以被用于测试游戏，但是它不能测试运行效率，内存使用情况和其它的一些功能。在模拟器上的运行体验和真实iOS设备上的体验是有很大区别的。不要把在iPhone模拟器上的游戏体验等同于真实设备上的游戏体验。

iOS设备

当你为iOS设备开发游戏时，你需要考虑不同设备之间的区别。大多数独立和兴趣型的游戏开发者没有足够资金买齐所有型号的iOS设备。这些设备之间的区别不是太大。在写这本书的时候，市场上一共有8个型号的iOS设备。而且每年大概会发布两个新的型号。虽然你不可能拥有所有型号的iOS设备，你至少需要了解它们之间的一些重要区别。

以下链接列出了所有iOS设备，包括iPhone，iPod Touch和iPad的设备说明：
http://support.apple.com/specs/#iphone
http://support.apple.com/specs/#ipodtouch
http://support.apple.com/specs/#ipad

表格2-1列出了游戏开发者需要关心的一些最重要的硬件方面的区别。iPod Touch设备以代来表示各代的设备，因为苹果没有给iPod Touch像iPhone那样用“3G”等来命名。从这个表格上，你可以看到iOS设备之间还是有些区别的。

表格2-1：iOS设备硬件区别

设备	CPU	图形芯片	屏幕分辨率	内存
一代设备	412 MHz	PowerVR MBX	480x320	128 MB
iPhone 3G(二代)	412 MHz	PowerVR MBX	480x320	128 MB
iPod Touch(二代)	532 MHz	PowerVR MBX	480x320	128 MB
三代设备	600 MHz	PowerVR SGX535	480x320	256 MB
iPad	1000 MHz	PowerVR SGX535	1024x768	256 MB
iPhone 4	1000 MHz	PowerVR SGX535	960x640	512 MB

可以看到，每推出一代新的iOS设备，CPU会变的更快，图形芯片会更加强劲，内存和屏幕分辨率也都会增加。如果你想通过iOS游戏赚钱，要记住市场上还有很多旧的型号存在，用户更新换代的速度永远要比新产品出来的速度慢很多。甚至到了今天，如果你在设计游戏时不考虑二代设备的话，你实际上放弃了很大一块市场。

通常，游戏开发者关注的硬件功能集中在CPU速度和图形芯片。他们用这两项来评估技术的可行性。但是，作为移动设备，直到最新的iPhone 4，iOS设备的瓶颈大多数是在可用的内存上。

注：不要将内存与储存MP3，视频，应用和照片的闪存相混淆。对后者来说，即使目前为止最小的iOS设备都有8GB。闪存等同于台式电脑的硬盘。内存则是用于为正在运行的应用程序储存代码，数据和贴图的。

关于内存使用量

iOS设备安装有128，256，或512MB的内存。但是，程序并不能使用所有这些内存。iOS操作系统本身要占用很大一部分内存，而且还要考虑到iOS 4的多任务操作。运行着iOS 4的设备可能同时运行好多个后台任务，这些后台任务会使用很多内存，而且事先无法确定。iOS开发者总结出了理论上一个应用程序可以使用的内存量。**表格2-2**列出了各种内存配备下可以为应用程序所用的内存量。理想状态下，你应该把你的内存使用量保持在“内存报警临界值”之下。特别是那些只有128MB内存的设备，大概只有20-25MB的内存可以被使用。在临界值附近，你的程序有可能开始接收到内存警告通知（Memory Warning notifications）。你可以忽略一级内存警告通知（Memory Warning Level 1 notifications）。但是如果程序继续使用更多的内存，你可能收到第二级内存警告通知。这时候操作系统其实是在告诉你：如果你再不释放一些内存的话，程序将被关闭。就像你妈威胁你，如果你不马上清理你的房间，她就不给你买新电脑！你还是遵从这些指令为妙。

表格2-2：你并不能用到所有安装着的内存

安装着的内存	可用内存	内存报警临界值
128 MB	35-40 MB	20-25 MB
256 MB	120-150 MB	80-90 MB
512 MB	340-370 MB	260-300 MB （估计）

cocos2d可以通过调用清除方法帮助你释放一些内存。通过在AppDelegate中的

applicationDidReceiveMemoryWarning方法里添加purgeCachedData方法，你可以让cocos2d帮着释放一些不再需要用到的内存：

```
- (void)applicationDidReceiveMemoryWarning:(UIApplication *)application {
        [[CCTextureCache sharedTextureCache] removeUnusedTextures];
        [[CCDirector sharedDirector] purgeCachedData];
}
```

但是当你的游戏变得复杂以后，你可能需要设计自己的方式来应对内存报警（Memory Warnings）。如果你单纯依靠cocos2d的机制来处理内存报警的话，可能会从内存中删除以后还需要用到的资源，比如精灵的贴图。如果贴图被删除，而几帧以后你的游戏又需要它，那么就会导致贴图在游戏进行的过程中被再次加载，这样的加载速度很慢。因为你可以用自己的代码区分不再需要的贴图和之后还需要的贴图，所以设计自己的方式来处理内存警报会更好。不过，没有单一的方法可以处理所有可能出现的情况。因为要是有的话，我就在这里提供给你了。

如果你在为配备有256或512MB内存的设备开发游戏的话，你要考虑到目前大多数的iOS设备只有128MB内存。除非你只针对第三代和第四代的iOS设备开发游戏，否则你应该买个便宜点的二手一代或者二代iOS设备，用于测试超出内存使用后可能产生的问题。问题发现的越早越好，因为越早发现就越容易解决问题，特别是涉及到游戏设计的时候。通常来说，我们建议使用硬件能力最弱的设备来做开发。这可以帮助你尽早发现任何运行在低内存环境下可能产生的问题。

注：我们也建议在运行iOS 4系统的设备上测试你的应用。在最坏的情况下测试：在打开多个背景任务，内存被占用很大的情况下。多任务只有在第三代和更新的设备上可以运行。这意味着只有配备256MB内存的设备允许运行多任务。这是个好消息：如果你为第一代和第二代设备设计游戏的话，因为它们只有128MB内存，你就不需要担心其它程序会占用内存。

在配备128MB内存的设备上，你最多能使用35到40MB内存。这还只是理论上的极限；具体内存大小取决于设备甚至是在运行你游戏之前运行过的应用程序。这也是为什么当你遇到程序崩溃时，开发者会建议你重启设备。重启设备可以释放一些内存。应用程序发生突然退出的最主要原因是设备内存耗尽。因此，你要对你应用程序的内存使用量提高警惕。

你可以使用Instruments来衡量内存使用量。请参考苹果的Instruments用户指南：
http://developer.apple.com/iphone/library/documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/Introduction/Introduction.html

模拟器

iPhone SDK 允许你通过安装在 Mac 上的 iPhone 模拟器和 iPad 模拟器来运行和测试 iPhone 和 iPad 应用程序。模拟器的首要作用是让你可以很快地测试你的应用程序，因为当你的游戏变的越来越大时，发布到 iOS 设备上做测试的过程将会需要越来越多的时间。特别是那些要用到很多图片和道具文件的游戏，发布过程将会很慢，因为传输这些图片和道具文件需要很多时间。

不过，模拟器有好几个缺点。以下我们列出了模拟器无法为你做的事情。出于这些原因，我们建议你尽早在真实设备上测试，而且要经常测试。至少在每一个大的改动之后或者在结束一天的开发工作之前，你应该在 iOS 设备上测试你的游戏以确保运行正常。

不能评估性能

在模拟器上的游戏运行性能安全依靠你电脑的 CPU。图形渲染过程甚至没有使用 Mac 图形芯片的硬件加速功能。这也是为什么通过模拟器得到的帧率（framerate）没有任何实际意义的原因。在你修改代码之后，你甚至无法确定模拟器上得到的前后帧率比较结果是否会和设备上得到的比较结果一致。在某些极端情况下，模拟器可能显示帧率上升，而在设备上却显示帧率下降。所以一定要在设备上做性能测试，并且使用发行版本（Release build）的配置。

不能评估内存使用量

模拟器可以使用所有你电脑上配置的内存，所以模拟器比 iOS 设备有更多的可用内存。这意味着你不会在模拟器上收到内存报警，你的游戏在模拟器上会运行流畅，但是当你发布到 iOS 设备上时，你可能发现游戏在第一次运行时就崩溃了。

不过你可以用模拟器评估游戏当前使用的内存大小。

无法使用所有 iOS 设备的功能

有一些 iOS 设备的功能，比如设备转向（Device Orientation），可以用程序菜单或者键盘快捷键来模拟。但是模拟的体验和真实的体验相差甚远。而有一些功能，像多点触摸，加速计，震动或者位置信息获取，则完全不能通过模拟器来测试。因为电脑的硬件无法模拟这些功能。即使你摇晃 Mac 或者点击屏幕也没有用。不信的话你可以试一下。

运行时的表现可能不一样

有时候你会碰到很棘手的问题：游戏在模拟器上运行良好，但是放到 iOS 设备上就崩溃，或者是没有理由的变慢。还有可能有些图形方面的问题只出现在模拟器上或者只出现在设备上。在你进入代码寻找可能存在的问题之前，如果问题出现在模拟器上，就在设备上运行你的游戏；如果问题出现在设备上，那就在模拟器上运行。有时候，出现的问题可能就自动消失。不过即使没有小时，通过这样做，你也可以了解问题可能出现在哪里。

关于日志

默认情况下，你的 cocos2d 项目会有两个构建配置（build configurations）：调试（Debug）和发布（Release）。他们之间的主要区别是：只有在调试时，某些函数，比如 CCLOG 才会编译和被游戏代码所使用。这是影响调试和发布构建两个配置之间影响运行性能最主要的因素。

注：CCLOG 宏将苹果的 NSLog 方法进行了封装，所以 CCLOG 只在调试构建时会被编译，在发布构建时会被删除。我建议在使用 NSLog 的地方用 CCLOG 代替，因为日志只是给你自己看的。NSLog 会让发布的游戏变慢，因为它即使在发布构建里也会运行！

即使一条 NSLog 或者 CCLOG 也会让 Debugger Console 的窗口填满日志信息，导致程序运行速度变慢。如果你觉得你的游戏在调试构建中的运行性能很差，你应该看一下 Debugger Console 里面是不是有多余的日志活动。从 Xcode 的 Run 菜单里，选择 Console 以显示 Debugger Console 窗口。

日志也是为什么要用发布构建来测试游戏性能的主要原因。

结语

本章我们讨论了从下载到设置所有需要的工具，然后到具体用 cocos2d 的模板生成你的第一个项目。

接着，我详细介绍了在理论上，cocos2d 程序是怎样工作的，外加一些细节。我也详细介绍了内存管理，因为内存管理很重要。它很容易被误解，甚至完全被忽略。如果忽略内存管理，你的游戏就会建立在一个很不稳固的基础之上。

我演示了如何给 cocos2d 程序添加触摸输入和 cocos2d 对象是如何被保存和取回的。

最后，我介绍了各种不同的 iOS 设备之间的区别，游戏可以在各个设备上使用的内存量。我也讨论了模拟器和真实设备的区别，还有在模拟器和真实机器上做测试的区别。

在下一章里，你将学习 cocos2d 的基础知识。理解了这些基础知识以后，我们就可以开始设计 cocos2d 游戏了。