iOS 5 Programming Cookbook

第十五章
图形和动画

版本 1.0

翻译时间：2012-08-20

DevDiv 翻译： kyelup cloudhsu 耐心摩卡
wangli2003j3 xiebaochun dymx101
Jimmylianf BeyondVincent

DevDiv 校对： laigb kyelup

DevDiv 编辑： BeyondVincent

写在前面

目前，移动开发被广大的开发者们看好，并大量的加入移动领域的开发。

鉴于以下原因：

- 国内的相关中文资料缺乏
- 许多开发者对 E 文很是感冒
- 电子版的文档利于技术传播和交流

DevDiv.com 移动开发论坛特此成立了翻译组，翻译组成员具有丰富的移动开发经验和英语翻译水平。组员们利用业余时间，把一些好的相关英文资料翻译成中文，为广大移动开发者尽一点绵薄之力，希望能对读者有些许作用，在此也感谢组员们的辛勤付出。

关于 DevDiv

DevDiv 已成长为国内最具人气的综合性移动开发社区

更多相关信息请访问 DevDiv 移动开发论坛。

技术支持

首先 DevDiv 翻译组对您能够阅读本文以及关注 DevDiv 表示由衷的感谢。

在您学习和开发过程中，或多或少会遇到一些问题。DevDiv 论坛集结了一流的移动专家，我们很乐意与您一起探讨移动开发。如果您有什么问题和技术需要支持的话，请访问 DevDiv 移动开发论坛 或者发送邮件到 BeyondVincent@DevDiv.com，我们将尽力所能及的帮助你。

关于本文的翻译

感谢 kyelup、cloudhsu、耐心摩卡、wangli2003j3、xiebaochun、dymx101、jimmylianf 和 BeyondVincent 对本文的翻译，同时非常感谢 laigb 和 kyelup 在百忙中抽出时间对翻译初稿的认真校验，指出了文章中的错误。才使本文与读者尽快见面。由于书稿内容多，我们的知识有限，尽管我们进行了细心的检查，但是还是会存在错误，这里恳请广大读者批评指正，并发送邮件至 BeyondVincent@devdiv.com，在此我们表示衷心的感谢。

读者查看下面的帖子可以持续关注章节翻译更新情况

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译各章节汇总

更多学习资料

我们也为大家准备了 iOS6 新特征的相关内容，请参考下面的帖子：

iOS6 新特征：参考资料和示例汇总------持续更新中

目录

前言

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_前言

第 1 章　基础入门

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第一章_基础入门

第 2 章　使用控制器和视图

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第二章_使用控制器和视图(上)

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第二章_使用控制器和视图(下)

第 3 章　构造和使用 Table View

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第三章_构造和使用 Table View

第 4 章 Storyboards

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第四章_Storyboards

第 5 章　并发

参考帖子

[DEVDIV 翻译] iOS 5 Programming Cookbook 翻译 第五章 并发

第 6 章　定位核心与地图

参考帖子

[DEVDIV 翻译] iOS 5 Programming Cookbook 翻译_第六章_核心定位与地图

第 7 章　实现手势识别的功能

参考帖子

[DEVDIV 翻译] iOS 5 Programming Cookbook 翻译_第七章_实现手势识别

第 8 章　网络，JSON，XML 以及 Twitter

参考帖子

[DEVDIV 翻译] iOS 5 Programming Cookbook 翻译_第八章_网络，JSON，XML 以及 Twitter

第 9 章　音频和视频

参考帖子

[DEVDIV 翻译]iOS 5 Programming Cookbook 中文翻译 第九章 音频和视频

第 10 章 通讯录

参考帖子

[DEVDIV 翻译]iOS 5 Programming Cookbook 中文翻译_第十章_通讯录

第 11 章 照相机和图片库

参考帖子

[DEVDIV 翻译]iOS 5 Programming Cookbook 中文翻译_第十一章_照相机和图片库

第 12 章 多任务

参考帖子

[DEVDIV 翻译]iOS 5 Programming Cookbook 中文翻译_第十二章_多任务

第 13 章 Core Data

参考帖子

[DEVDIV 翻译]iOS 5 Programming Cookbook 中文翻译 第十三章 Core Data

第 14 章　日期，日程表和事件

参考帖子

[DEVDIV 翻译]iOS 5 Programming Cookbook 中文翻译_第十四章_日期，日程表和事件

第 15 章 图形和动画

15.0. 简介

你一定在 iPhone 或 iPad 上见过带有美丽的图形效果的应用。也可能会在游戏或其他应用中遇到过令人印象深刻的动画。当 iOS 运行时和 Cocoa 编程框架组合时，让使用相对简单的代码产生出各种迷人的图形和动画效果成为可能。当然，这些图形和动画的质量部分的依赖于程序员对数学的敏感以及美术合作者。但在本章中，你将看到如何使用适当的编程技巧来完成。

在后面的学习过程中，我会不断介绍一些背景概念，介绍诸如颜色空间，变换和图形环境之类的概念。在冲进代码之前，我会介绍一点基础知识。

在 Cocoa Touch 中，程序是由“窗口”和“视图”组成。一个带有 UI 的程序至少有一个窗口，这个窗口至少包括一个到多个视图。在 Cocoa Touch 中，一个窗口就是一个 UIWindow 的实例。一般的，程序会打开到主窗口，然后程序员会向这个窗口添加视图，来展示 UI 的不同的部分：例如按钮，文本，图像和自定义控件。所有这些 UI 相关的组件是由 UIKit 来进行处理和绘制的。

某些概念听起来可能不容易理解，但我保证随着我们深入本章，你将从我给你的许多例子中逐步的理解。

Apple 为开发者提供了强有力的框架，来处理 iOS 和 OS X 中的图形和动画。下面是这些框架和技术：

UIKit
高级别框架，允许程序员创建视图，窗口，按钮，和其他 UI 相关的控件。它也将低层的 API 组合到一个易于使用的高级别 API 中。

Quartz 2D
运行内部的用于 iOS 画图的主引擎；UIKit 使用了 Quarz。

Core Graphics
支持图形环境（后面会介绍），加载图片，绘制图片等等的框架。

Core Animation
顾名思义，iOS 上的动画框架。

当在屏幕上绘图时，需要掌握的最重要的概念之一是点和像素的关系。我确信你熟悉像素，但是点是什么呢？他们是像素和设备相关的伙伴。例如，比较一下 iPhone3GS 和 iPhone4，他么都有 3.5 英寸的屏幕。不过，iPhone 3GS 上在竖屏模式下可以绘制的像素数量是 320 X 480。相同尺寸的 iPhone 4 上的竖屏模式下，却能够绘制两倍的像素，是 640 X 960。

现在想像一下，你在开发一个 iPhone 应用，它仅有一个屏幕，你可以使用绿色简单的

填充整个屏幕。假设你天真的指定了填充区域为 320 x 480 像素。当 iPhone 3GS 用户运行你的程序时他们会相当开心，因为“它干了它说要干的事” --- 将整个屏幕填充为绿色。但是，iPhone 4 用户会相当不开心：他们看到了非常不同的画面，如图 15-1 所示。

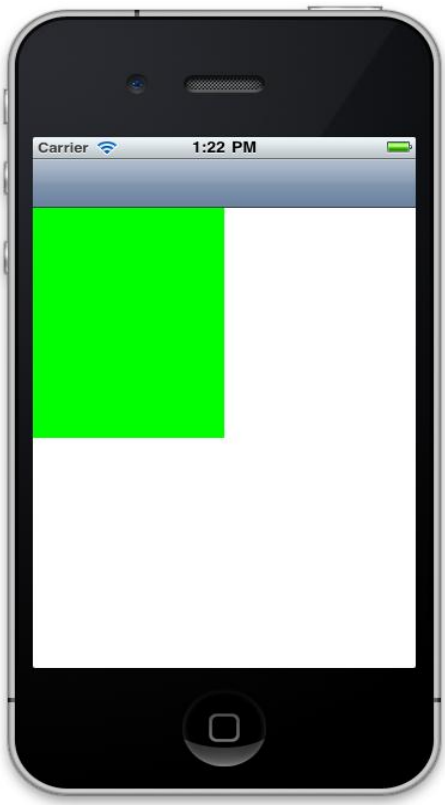

图 15-1. 依赖于设备的像素渲染在不同设备上显示不同结果

为了补救这个问题，Apple 引入了设备独立的绘制方法，来帮助开发者关注他们的图形和图像如何显示到设备上，而不是担心在不同设备上运行相同的代码时的屏幕尺寸和分辨率的问题。为了解决我们在图 15-1 中看到的问题，应用的开发者只要使用相应的 API，以点而不是像素为单位来指定绿色矩形。这能让同样的代码在 iPhone 3GS 和 iPhone 4 上运行，确保了 iPhone4 的屏幕会被矩形完全填充。因此，你在本章中看到的许多方法将是以点（或者像苹果一样，称他们为逻辑点）为基础的。

在 iOS 设备上屏幕的原点在左上角。以左上角为绘制原点的屏幕也被称为 Upper Left Origin 屏， 或者 ULO 屏。这意味着(0,0)点位于屏幕的最左和最上边的位置。并且 x 轴从这点向右为正向，y 轴向下为正向。也就是说 x 轴的 20 比 10 要更靠右一些。y 轴上，20 比 10 要更靠下一些。

在本章中，会使用 UIView 类型那个的视图对象来绘制图形，字符串，以及其他在屏幕上可见的东西。

我假设你有最新的 XCode。如果没有，请到 Xcode 的网站上下载它。

为了将本章代码片段加入到一个应用程序中，首先我要告诉你在 Xcode 中创建工程以及编写 UIView 子类的必要步骤：

1. 打开 Xcode。
2. 从 File 菜单，选择 New->Project。
3. 在屏幕的左侧，确定 iOS 分类被选中。在该分类下选择 Application（见图 15-2）。

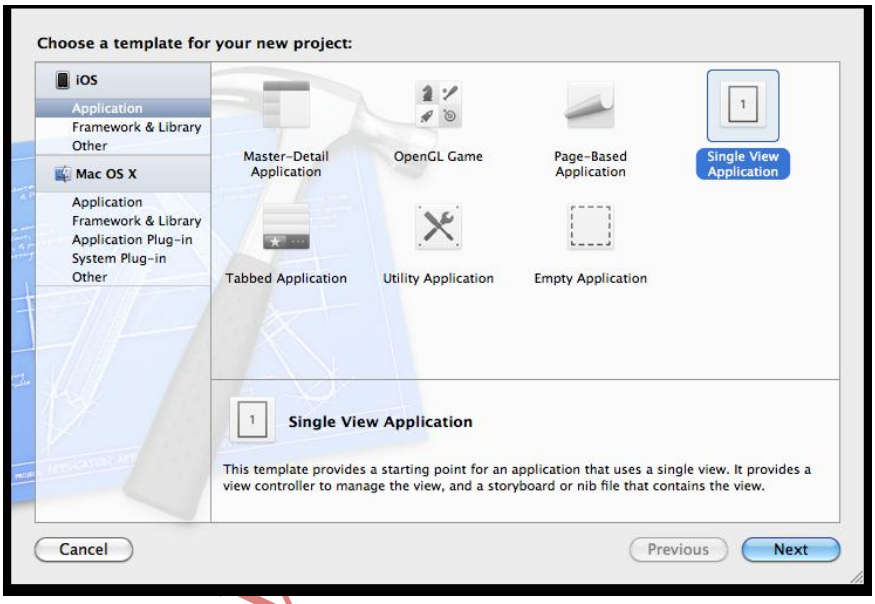

图 15-2， 在 Xcode 中为 iOS 创建一个单视图程序

4. 在屏幕右边，选择 View-based Application，点击 Next（参见图 15-2）。
5. 在 Product Name 输入框，为你的工程选择一个名字。我输入了 Graphics，并且推荐你使用相同的名字，避免之后感到混乱。
6. 在 Company Identifier 输入框中，输入一个目录标识前缀，应该优先使用你选择的 Product Name。通常是 com.company。我选择了 com.pixolity。Xcode 会自动为你选择它。
7. 在 Device Falmily 中，选择 iPhone，然后点击 Next。
8. 在下一个屏幕(图 15-4)，选择你要保存工程的位置。我选择了 Desktop。点击 Create。

现在你的 Xcode 工程已经打开。在 Xcode 的左边，展开 Graphics 组，显示在我们创建工程时 Xcode 为我们创建的文件。现在我们应该为我们的视图控制器创建一个视图对象了。请按照以下步骤来做：

1. 在 Xcode 的左手边选择 Graphics 组。
2. 右键点击 Graphics 主，选择 New File…
3. 在 New File 对话框中，确保 iOS 被选中，并选 Cocoa Touch 作为子分类（参见图 15-5）.

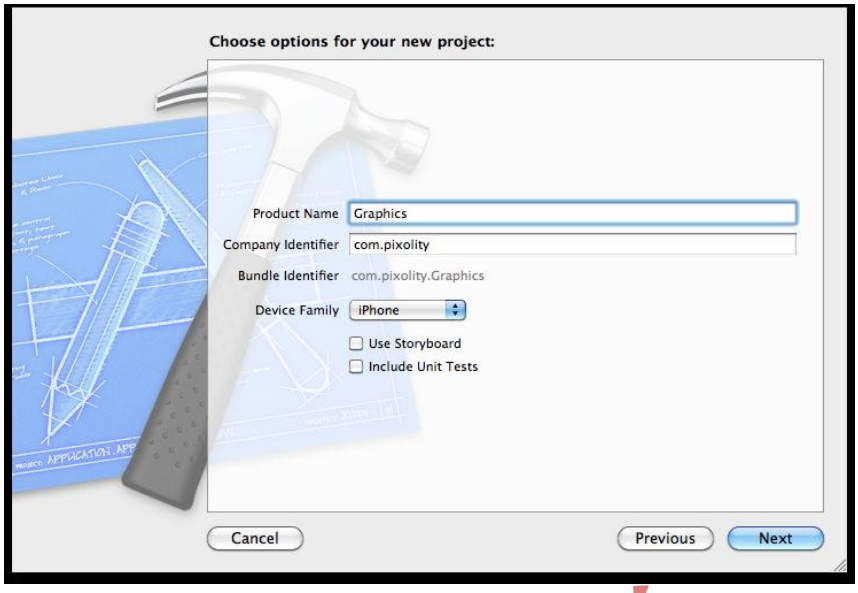

图 15-3. 在 XCode 中对一个新建工程的选项进行设置

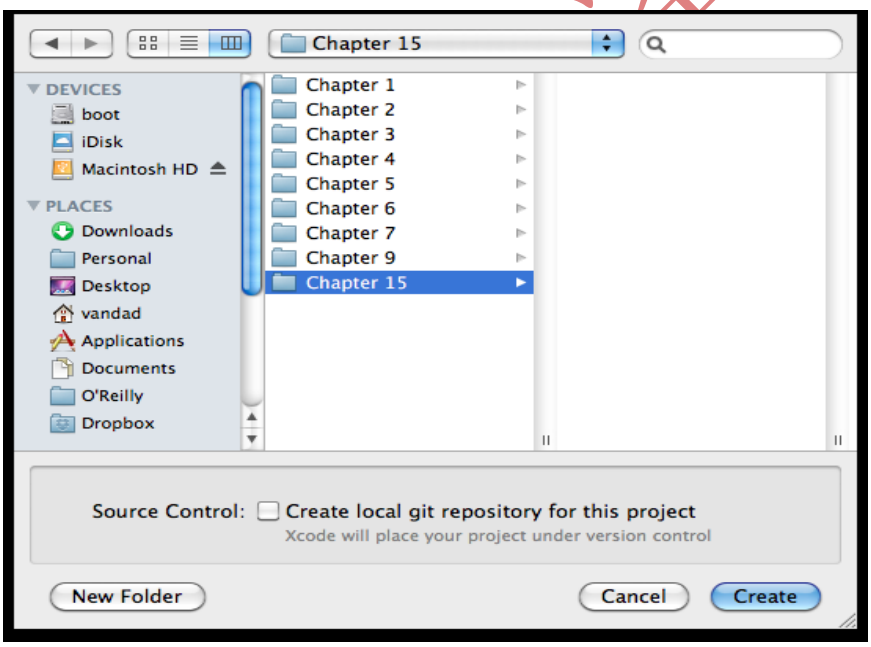

图 15-4. 将 Xcode 工程保存到磁盘

4.在右边，选择 Objective-C 类，点击 Next（参见图 15-5）.

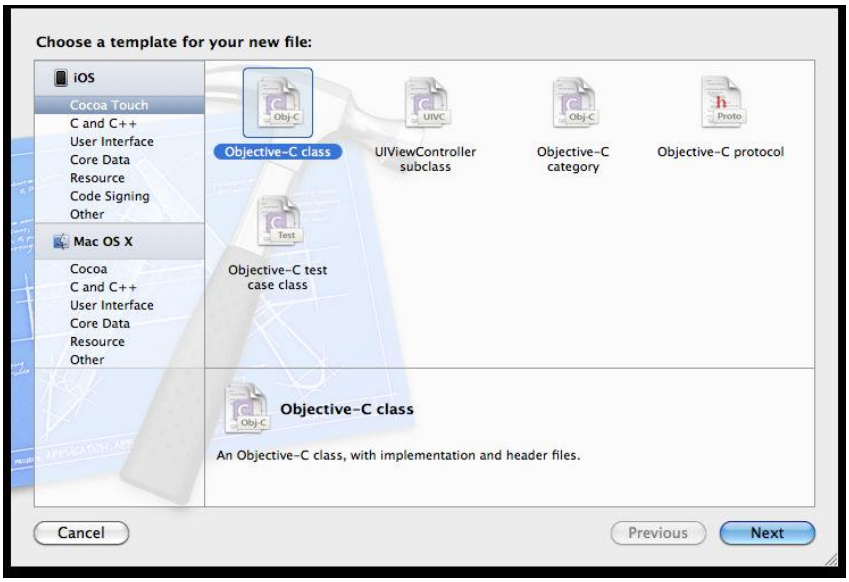

图 15-5. 在 Xcode 中创建一个新的 Objective-C 类

5. 在下一个屏幕（图 15-6），确定 Subclass 框中选择了 UIView，然后点击 Next。

6.在 Save As 对话框中，设置文件名为 GraphicsViewControllerView.m.

7.在 Group 下拉框中选择 Graphics（参见图 15-7）.

8.确定"Add to targets"勾选了之前创建的工程，点击 Save（见图 15-7）.

9.在 XCode 主窗口的左侧，点击 GraphicsViewController.xib 文件，Interface Builder 将在 Xcode 屏幕的右侧出现，如图 15-8 所示。现在我们还不会使用.xib 文件。

10.从 Xcode 菜单，选择 View->Utilities->File Inspector。文件检查器将被显示，默认位于 Xcode 窗口的右侧。

11.在 Interface Builder 中点击你之前创建的灰色视图的内部。在文件检查器（位于右侧）中会反映出你选择的变化。

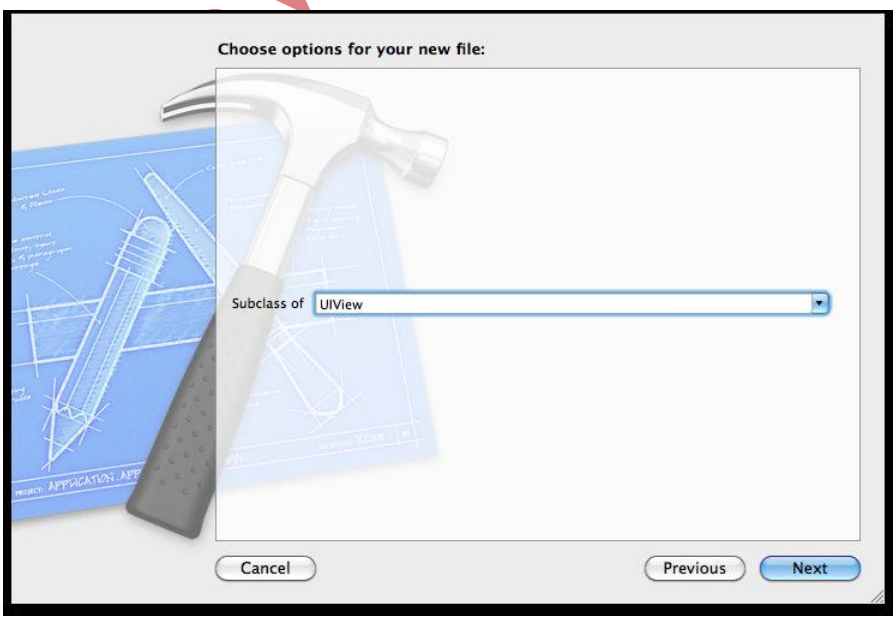

图 15-6. 创建 UIView 的子类

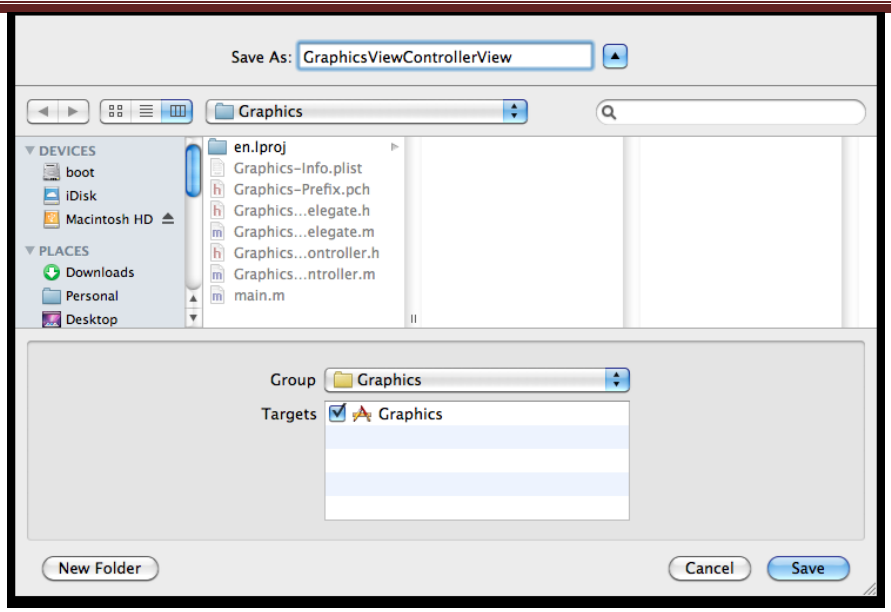

图 15-7. 将 UIView 的子类保存到磁盘

12.在文件检查器中，选择顶部的身份检查器（Identity Inspector）（见图 15-10）.

13.在 Custom Class 区域的 Class 框中，输入 GraphicsViewControllerView(我们之前创建的视图对象)，并点击键盘上的回车。

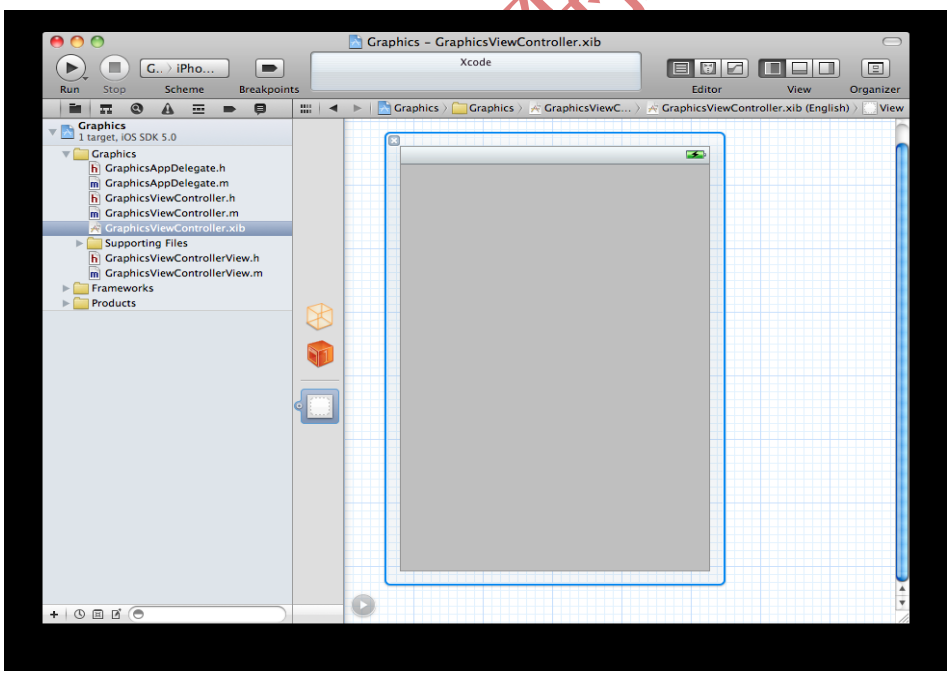

图 15-8. 选择视图控制器的 xib 文件

现在我们准备好了，可以开始编码了。我们要做的只是创建一个 UIView 类型的视图类，这样在本章后面，可以修改这个类的代码。然后我们使用 Interface Builder 将视图控制器的视图类，设置为我们创建的那个视图对象。这意味着现在我们的视图控制器的视图将是一个 GraphicsViewControllerView 类的实例。

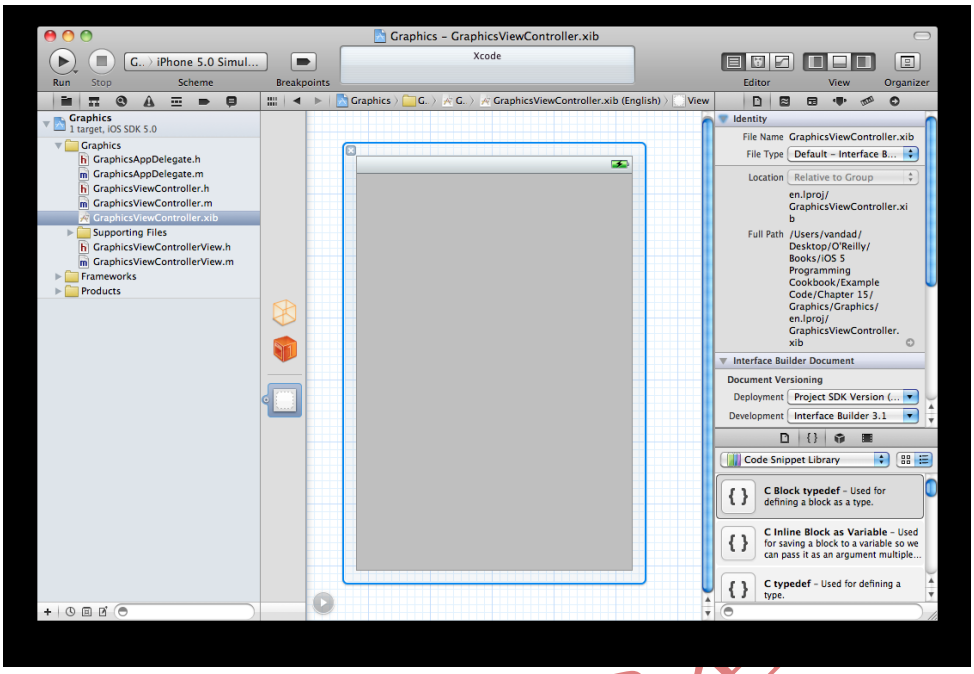

图 15-9. Interface Builder 中的文件检查器

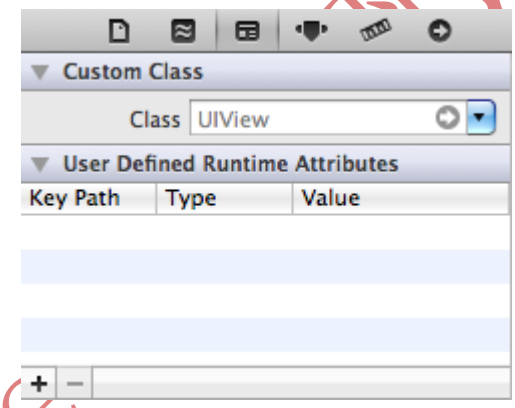

图 15-10. Identity Inspector 显示了我们视图对象的信息

你可能已经看过 Xcode 产生的视图对象的内容了。这个对象中最重要的方法之一就是 drawRect:。Cocoa Touch 不论何时要绘制这个视图，都会自动调用这个方法，用它请求视图对象将其内容绘制到 Cocoa Touch 自动为此视图准备的图形环境之上。图形环境可以被当作一个画布，它提供了大量的属性，比如画笔颜色，画笔厚度等等。给定了环境，你可以在 drawRect:方法中直接开始画画，Cocoa Touch 会确保图形环境的属性应用到你的绘制中。我们后面会讨论这个，现在，让我们开始更有趣的主题吧。

15.1. 枚举和加载字体

15.1.1. 问题

为了渲染屏幕上的某些文字，你想使用 iOS 设备上预装的字体。

15.1.2. 方案

使用 UIFont 类

15.1.3. 讨论

字体是在图形用户界面上显示文字的基础。UIKIt 框架为程序员提供了便于枚举，加载，和使用字体的高级别 API。字体被封装于 Cocoa Touch 中的 UIFont 类中。每个 iOS 设备自身都有内建的系统字体。字体被组织到 family 中，每个 family 中包含 face。比如，Helvetica 是一个字体 family，Helvetica Bold 是 Helvetica 家庭的一个 face。为了能加载字体，你必须知道字体的 face（也就是他的名字）--- 要想知道它的 face，就必须知道 family，所以，我们首先要枚举出安装到设备上的所有的字体 family，使用 UIFont 类的 familyNames 类方法：

```
- (void) enumerateFonts{

    for (NSString *familyName in [UIFont familyNames]){
      NSLog(@"Font Family = %@", familyName);
    }
}
```

在 iOS 模拟器上运行这个程序，我得到了类似下面的结果：

```
Font Family = Heiti TC
Font Family = Sinhala Sangam MN
Font Family = Kannada Sangam MN
Font Family = Georgia
Font Family = Heiti J
Font Family = Times New Roman
Font Family = Snell Roundhand
Font Family = Geeza Pro
Font Family = Helvetica Neue ...
```

在得到字体 family 之后，我们可以在每个 family 内部枚举字体名字。我们使用 UIFont 类的 fontNamesForFamilyName:类方法，可以得到以 family 名字作为参数得到的字体名字数组：

```
- (void) enumerateFonts{

    for (NSString *familyName in [UIFont familyNames]){
      NSLog(@"Font Family = %@", familyName);
      for (NSString *fontName in
           [UIFont fontNamesForFamilyName:familyName]){
           NSLog(@"\t%@", fontName);
       }
    }
}
```

在 iOS 设备上运行上面的代码，我们得到如下结果：

```
...
Font Family = Geeza Pro
   GeezaPro
   GeezaPro-Bold
Font Family = Helvetica Neue
   HelveticaNeue-Italic
   HelveticaNeue-Bold
   HelveticaNeue-BoldItalic
   HelveticaNeue
   ...
```

你可以看到，Helvetica Neue 是字体 family，HelveticaNeue-Bold 是这个 family 中的一个字体名字。现在我们知道了字体名字，我们就可以使用 UIFont 的 fontWithName:size:类方法将字体加载到 UIFont 类型对象中：

```
UIFont *helveticaBold = [UIFont fontWithName:@"HelveticaNeue-Bold" size:12.0f];
```

如果 UIFont 类的 fontWithName:size:类方法的返回结果为 nil，指定的字体名字不能被找到。首先枚举出所有的字体 family 和该 family 中的所有字体名字，可以确保你提供的字体名字在系统中可用。

你也可以使用 UIFont 类的 systemFontOfSize:类方法（译者：原文是 instance method，但实际上是类方法）（或者它的粗体方法，boldSystemFontOfSize:）从你代码运行的设备上加载系统字体，不管这些字体是什么。iOS 设备的默认系统字体是 Helvetica。

在你加载了字体之后，你可以继续学习 15.2 小节，我们会用这里加载的字体，在图形环境中绘制文本。

15.1.4. 参见

XXX

15.2. 绘制文本

15.2.1. 问题

你希望能够在 iOS 设备上绘制文本。

15.2.2. 方案

使用 NSString 的 drawAtPoint:withFont:方法

15.2.3. 讨论

为了绘制文本，我们可以使用构建在 NSString 中的一些相当便利的方法，比如 drawAtPoint:withFont:。在我们继续之前，确保你遵照了 15.0 章中的说明。你现在应该有一个视图对象，继承自 UIView，名称为 GraphicsViewControllerView。打开该文件。如果视图对象的 drawRect:实例方法被注释了，取消那些注释，让你的视图对象里有这个方法：

```
#import "GraphicsViewControllerView.h"

@implementation GraphicsViewControllerView

- (id)initWithFrame:(CGRect)frame{
    self = [super initWithFrame:frame];
    if (self) {
        // Initialization code
    }
    return self;
}
-(void)drawRect:(CGRect)rect{
}
@end
```

drawRect:方法是我们要进行绘制的地方，如前面所提到的那样。在此，我们可以开始加载字体，然后在屏幕上 x 轴的 40 及 y 轴 180 处以 40 点的字体画出一个简单的字符串（图 15-11）：

```
- (void)drawRect:(CGRect)rect{

UIFont *helveticaBold = [UIFont fontWithName:@"HelveticaNeue-Bold" size:40.0f];

NSString *myString = @"Some String";
[myString drawAtPoint:CGPointMake(40, 180)
withFont:helveticaBold];

}
```

在上面的代码中，我见仅仅是加载了一个 40 点尺寸的粗体 Helvetica 字体，然后使用它来在点(40, 180)画出了文本“Some String”。

图 15-11. 在一个视图的图形环境上绘制的一个随机字符串

15.2.4. 参考

XXX

15.3. 构造，设置和使用颜色

15.3.1. 问题

你希望能够获得颜色对象的引用，以便你在视图上绘制诸如文本，矩形，三角形和直线片段这些不同形式时，能够使用他们。

15.3.2. 方案

使用 UIColor 类

15.3.3. 讨论

UIKit 为程序员提供了对颜色的高级别抽象，封装在 UIColor 对象中。这个类有一些相当方便的类方法，例如 redColor，blueColor，brownColor，和 yellowColor。但是，如果你在找的颜色不是上述 UIColor 的显式命名了的方法所提供的话，你总是可以使用 UIColor 的 colorWithRed:green:blue:alpha:类方法来加载你要找的颜色。这个类方法的返回值是一个

UIColor 类型的值。此方法的参数如下：

red

在颜色中用到的红色数量。这个值可以取 0.0f 到 1.0f 之间的任何值，0.0f 没有任何红色，而 1.0f 让红色分量尽可能的多。

green

在颜色中同红色混合的绿色值。这个值的范围也是 0.0f 到 1.0f。

blue

颜色中和红色及绿色混合的蓝色值的量。这个值的范围也是 0.0f 到 1.0f。

alpha

颜色的不透明度。这个值可以在 0.0f 到 1.0f 间取值，取 1.0f 时颜色完全不透明，0.0f 时完全透明（也就是说，不可见）。

在你获得了 UIColor 类型的对象之后，你可以用它的 set 实例方法，使当前的图形环境在接下来的绘图中使用这个颜色了。

你可以使用 UIColor 类的 colorWithRed:green:blue:alpha: 类方法来加载元色（primary colors），比如要加载红色的话，可以传递 1.0f 作为 red 参数，0.0f 作为 green 和 blue 参数，alpha 参数随你的便。

如果你查看图 15-11，你会注意到我们刚创建的视图对象的背景色是一种相当丑陋的灰色。。让我们改一下吧？只要找到你的视图控制器的 viewDidLoad 实例方法，将视图的背景色改为白色，如下：

```
- (void)viewDidLoad{
    [super viewDidLoad];
    self.view.backgroundColor = [UIColor whiteColor];
}
```

我们将使用 NSString 类的实例方法来将文本绘制到当前的图形环境上，就像下面要讨论的那样。

现在让我们将品红色加载到 UIColor 对象上，然后用 30 号的 粗体 Helvetica 字体将文本“I Learn Really Fast”绘制到当前视图的图形环境上（加载字体请参考 15.1 小节）：

```
- (void)drawRect:(CGRect)rect{
// Drawing code

/* Load the color */
UIColor *magentaColor =[UIColor colorWithRed:0.5f
                                       green:0.0f
```

```
                                                blue:0.5f
                                               alpha:1.0f];
/* Set the color in the graphical context */
  [magentaColor set];

/* Load the font */
UIFont *helveticaBold = [UIFont fontWithName:@"HelveticaNeue-Bold"
                                                size:30.0f];

/* Our string to be drawn */
NSString *myString = @"I Learn Really Fast";

/* Draw the string using the font. The color has already been set */
[myString drawAtPoint:CGPointMake(25, 190)
                withFont:helveticaBold];
}
```

结果显示在图 15-12 中。

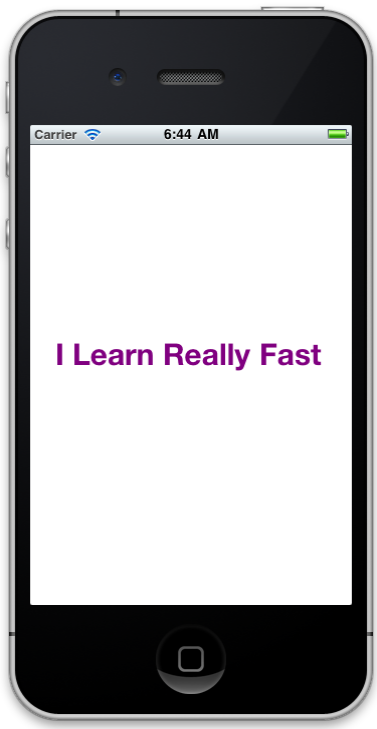

图 15-12. 用某种颜色在一个图形环境上绘制文本

我们也使用 NSString 类的 drawInRect:withFont:实例方法，将文本绘制在指定矩形空间中。文本为了适配矩形会被拉伸。UIKit 甚至会在文本不能适配给定矩形时将它隐藏。矩形边界被包裹在 CGRect 结构中。你可以使用 CGRectMake 函数来创建一个矩形边界。这个函数接受 4 个参数：

x

矩形相对于图形环境的坐标的 x 轴位置。在 iOS 中，这是从矩形的左边开始向右的点数。

y

矩形相对于图形环境的坐标的 y 轴位置。在 iOS 中，这是从矩形的顶部开始向下的点数。

width
矩形的宽度（以点为单位）。

height
矩形的高度（以点为单位）。

```
- (void)drawRect:(CGRect)rect{
// Drawing code

/* Load the color */
UIColor *magentaColor = [UIColor colorWithRed:0.5f
                                        green:0.0f
                                         blue:0.5f
                                        alpha:1.0f];
/* Set the color in the graphical context */
[magentaColor set];

/* Load the font */
UIFont *helveticaBold = [UIFont boldSystemFontOfSize:30];

/* Our string to be drawn */
NSString *myString = @"I Learn Really Fast";

/* Draw the string using the font. The color has already been set */
[myString drawInRect:CGRectMake(100,/* x */
                                120, /* y */
                                100, /* width */
                                200) /* height */
            withFont:helveticaBold];
}
```

输出如图 15-13 所示。

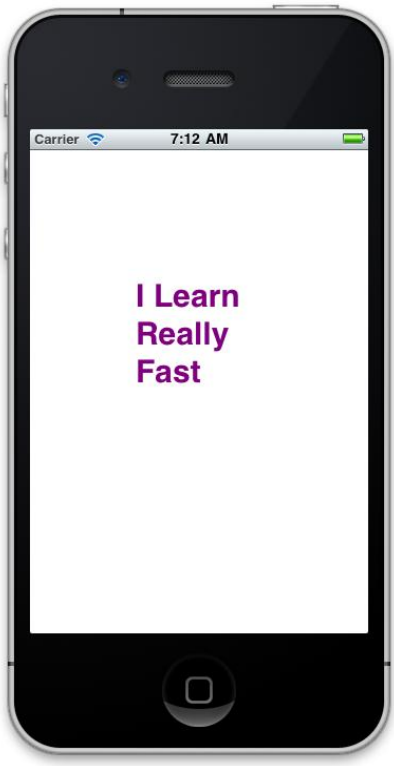

图 15-13. 在矩形控件中绘制一个字符串

UIColor 其实是 UIKit 对于 Core Grahpics 的 CGColor 类的包装。当我们得到 Core Graphics 这样低级别的对象时，我们立刻对于如何使用颜色对象有了更多的控制，甚至可以决定颜色的组成部分。我们假设有一些代码，传给你一个 UIColor 类型对象，你想检查他的 red,green,blue 和 alpha 组成部分。为了得到 UIColor 的组成部分，你可以按下面的步骤做：

1.使用 UIColor 类实例的 CGColor 实例方法。这会返回一个 CGColorRef 类型的颜色对象，这是一个 Core Graphics 颜色引用（Color Reference）对象。

2.使用 CGColorGetComponents 函数来得到构成颜色对象的组成部分。

3.使用 CGColorGetNumberOfComponents 函数来确定我们用来构造颜色（red+green+其他）的组件数量，如果有必要的话。

下面是一个例子：

```
/* Load the color */
UIColor *steelBlueColor = [UIColor colorWithRed:0.3f
                                          green:0.4f
                                           blue:0.6f
                                          alpha:1.0f];

CGColorRef colorRef = [steelBlueColor CGColor];

const CGFloat *components = CGColorGetComponents(colorRef);
 NSUInteger componentsCount = CGColorGetNumberOfComponents(colorRef);

 NSUInteger counter = 0;
 for (counter = 0;
     counter < components
     Count; counter++){
 NSLog(@"Component %lu = %.02f",
 }
```

在我们运行上面的代码后，控制台窗口的输出为：

```
Component 1 = 0.30
Component 2 = 0.40
Component 3 = 0.60
Component 4 = 1.00
```

15.3.4. 参见

XXX

15.4. 绘制图像

15.4.1. 问题

你希望能够向 iOS 设备的屏幕绘制图像。

15.4.2. 方案

使用 UIImage 类加载一个图像，然后用图像的 drawInRect:方法将它绘制到图形环境上。

15.4.3. 讨论

UIKit 能帮你轻松画出图像。你只要将图像加载到 UIImage 实例中即可。UIImage 类提供了不同的类方法和实例方法，用于加载你的图像。下面是 iOS 中的一些比较重要的方法：

imageNamed: 类方法
加载图片（如果加载成功还会缓存图像）。参这个方法的参数是 bundle 中的图像名字，比如 Tree Texture.png。

imageWithData:类方法
从 NSData 对象实例中包裹的数据中加载图片，NSData 对象是此方法的参数传入的。

initWithContentsOfFile:实例方法(用于初始化)
使用指定参数作为路径来加载一个图像，并用来初始化图像对象。


路径应该是在 bundle 中图像的完整路径。

initWithData:实例方法（用于初始化）
使用 NSData 类型的指定参数来初始化图像。这个数据应该属于一个有效图像。

请遵照以下步骤，来向你的 Xcode 工程添加图像文件：
1.在你的电脑上找到图片的位置。
2.将图片文件拖放到 XCode（放到左边你的工程文件保存之处）。
3.屏幕上会显示一个对话框，如果你想将图片文件拷贝到你的工程结构中，勾选“Copyitemsintodestination group's folder (if needed)”，如果你不想将图片拷贝到你的工程文件中，而是让 Xcode 从你拖放的原始位置读取的话，就不要勾选。
4.在 Folder 部分，确保“Create groups for any added folders”单选按钮被选中。
5.在“Add to targets”部分，确保你勾选了你要添加图片的目标。


你可以在 Google 图片中搜一搜 xcode filetype:png，找到 PNG 格式的 XCode 图标文件。

我们会把这张图绘制到图形环境上，来演示如何绘制图片。

我已经找到了这个文件并拖放它到我的 iOS 应用中。现在在我的应用 bundle 中有一个名为 xcode.png 的图像。该图像如图 15-14 所示

。

图 15-14. XCode 的图标，通过 Google 中搜索得到

下面是绘制图像的代码：

```
- (void)drawRect:(CGRect)rect{

UIImage *image = [UIImage imageNamed:@"Xcode.png"];
   if (image != nil){
   NSLog(@"Successfully loaded the image.");
   } else {
   NSLog(@"Failed to load the image.");
   }
}
```

如果在你程序的 bundle 中有 Xcode.png 这个图片，运行上面的代码将在控制台中输出 "Successfully loaded the image"。如果你没有图片，则输出“Failed to load the image”。下面，我将假设你在程序目录下有这张图片。你可以随便用什么别的图替换 XCode.png，我们将在示例代码中使用它。

在图形环境上绘制 UIImage 类型的图片的两种最简单的方法是：

drawAtPoint:UIImage 的实例方法

将图片以原始尺寸绘制到指定坐标点。使用 CGPointMake 函数来构造坐标点。

drawInRect:UIImage 的实例方法

在指定的矩形空间绘制图片，要构造这个矩形空间，请使用 CGRectMaker 函数：

```
- (void)drawRect:(CGRect)rect{

    /* Assuming the image is in your app bundle and we can load it */
    UIImage *xcodeIcon = [UIImage imageNamed:@"Xcode.png"];

    [xcodeIcon drawAtPoint:CGPointMake(0.0f, 20.0f)];
    [xcodeIcon drawInRect:CGRectMake(50.0f,
                                     10.0f,
                                     40.0f,
                                     35.0f)];
}
```

以上对 drawAtPoint:的调用将按照图像的完整尺寸将其绘制在(0,20)这一点，而 drawInRect:会将图像以 40X35 点在(50,10)位置进行绘制，如图 15-15 所示：

图 15-15. 在图形环境中绘制图像，可以用两种不同方法完成

宽高比（Aspect ratio）是图像或者电脑屏幕的宽度和高度之比。假设你有一个 100X100 的图像，如果你将它_以（100，200）的尺寸绘制在(0,0)这一点，你立刻可以看到图像在高度上被拉伸了（以 200 像素替换了 100 像素）。UIImage 的 drawInRect：实例方法让你来决定要怎么绘制。也就是，是你指定了 x, y,宽度和高度，让他在屏幕上显示。

15.4.4. 参见

XXX

15.5. 画线

15.5.1. 问题

你想在图形环境上画线。

15.5.2. 方案

得到你的图形环境，然后用 CGContextMoveToPoint 和 CGContextAddLineToPoint 函数来画线。

15.5.3. 讨论

当我们谈到 iOS 或 OS X 上的形状时，我们其实是在谈论路径(paths)。什么是路径，你会问？路径是由屏幕上的一组或多组点的集合构成的。在路径和线之间有个很大的区别。就是，路径可以包含多条线，但是线不能包含多个路径。可以将路径理解为多组点的集合 ---就那么简单。

线必须使用路径来绘制。指定起点和终点，然后请求 Core Graphics 来为你填充路径。Core Graphics 会意识到你在那个路径上创建了一条线，并会用你指定的颜色为你绘制路径（参见 15.3 小节）。

后面我们会更深入的介绍路径（参见 15.6 小节），但现在，让我们关注如何使用路径创建直线。为此，请遵照以下步骤：

1.在你的图形环境选择一个颜色（参见 15.3 小节）。
2.使用 UIGraphicsGetCurrentContext 函数，获得图形环境的句柄。
3.使用 CGContextMoveToPoint 函数设置线的起点。
4.使用 CGContextAddLineToPoint 函数移动图形环境的画笔，来指定线的终点。
5.使用 CGContextStrokePath 函数创建你的路径。这个过程将使用图形环境当前设置的颜色来绘制路径。


在 iOS 中，线的宽度以逻辑点为单位计量。

下面是一个示例：

```
- (void)drawRect:(CGRect)rect{

/* Set the color that we want to use to draw the line */
  [[UIColor brownColor] set];

/* Get the current graphics context */
CGContextRef currentContext = UIGraphicsGetCurrentContext();
```

```
/* Set the width for the line */
CGContextSetLineWidth(currentContext,
                      5.0f);

/* Start the line at this point */
CGContextMoveToPoint(currentContext,
                     50.0f,
                     10.0f);

/* And end it at this point */
CGContextAddLineToPoint(currentContext,
                        100.0f,
                        200.0f);

/* Use the context's current color to draw the line */
  CGContextStrokePath(currentContext);

}
```

在 iOS 模拟器上运行此代码，结果如图 15-16。

图 15-16. 在当前图形环境上绘制一条线

让我们看另外一个例子。前面提到，CGContextAddLineToPoint 过程指定了当前线的终点。现在，如果我们已经从(20,20)大盘（100，100）画了一条线，怎么再从（100，100）向（30，100）继续画线呢？你可能会想到，在画完第一条线后，我们已经用 CGContextMoveToPoint 将画笔移动到（100，100）了，然后可以使用 CGContextAddLineToPoint 过程向（300，100）继续画线。这样是没问题的，但是有一个更高效的方法。在你调用了 CGContextAddLineToPoint 过程指定了线的终点之后，你的画笔会

移动到这个方法指定的位置。也就是说，在你调用完使用画笔的方法后，它会将画笔留在它绘制的终点，你只需要用另一个终点，再次调用 CGContextAddLineToPoint 过程，下面是示例代码：

```
- (void)drawRect:(CGRect)rect{
// Drawing code

/* Set the color that we want to use to draw the line */
  [[UIColor brownColor] set];

/* Get the current graphics context */
CGContextRef currentContext = UIGraphicsGetCurrentContext();

/* Set the width for the lines */
CGContextSetLineWidth(currentContext, 5.0f);

/* Start the line at this point */
CGContextMoveToPoint(currentContext,
                     20.0f,
                     20.0f);

/* And end it at this point */
CGContextAddLineToPoint(currentContext,
                        100.0f,
                        100.0f);

/* Extend the line to another point */
  CGContextAddLineToPoint(currentContext,
                          300.0f,
                          100.0f);

/* Use the context's current color to draw the lines */
  CGContextStrokePath(currentContext);
}
```

结果如图 15-17 所示。你可以看到，两条线都成功的被画出来了，而不需要为第二条线重新移动画笔。

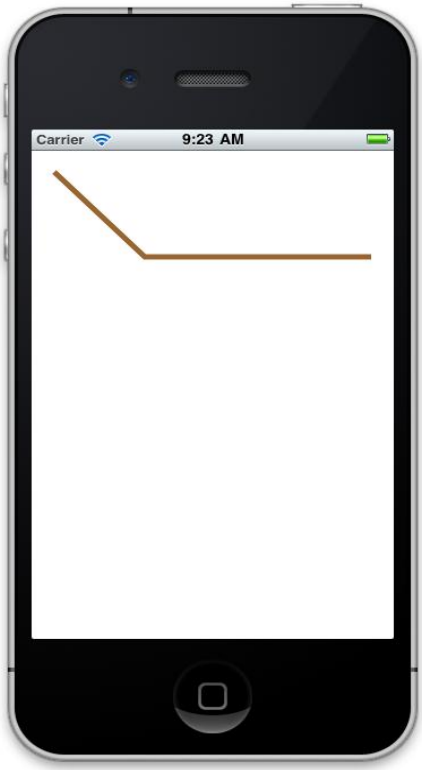

图 15-17. 一次画两条线

两条线相遇的点，毫不意外，被称为接合点（join）。使用 Core Graphics，你可以指定线和线之间接合点的类型。要设置自己的接合点类型，你必须调用 CGContextSetLineJoin 过程。它接受两个参数：图形环境和接合点类型，类型必须是 CGLineJoin 类型。CGLineJoin 有以下枚举值：

kCGLineJoinMiter
接合点为尖角。这是默认的接合类型。

kCGLineJoinBevel
接合点为斜角

kCGLineJoinRound
接合点为圆角

让我们看一个例子。假设我们想写个程序在图形环境上画“屋顶”（画 3 个，每个接合类型 1 个），同时在每个屋顶下画出接合点使用的类型的文字。结果会像图 15-18 所示那样。

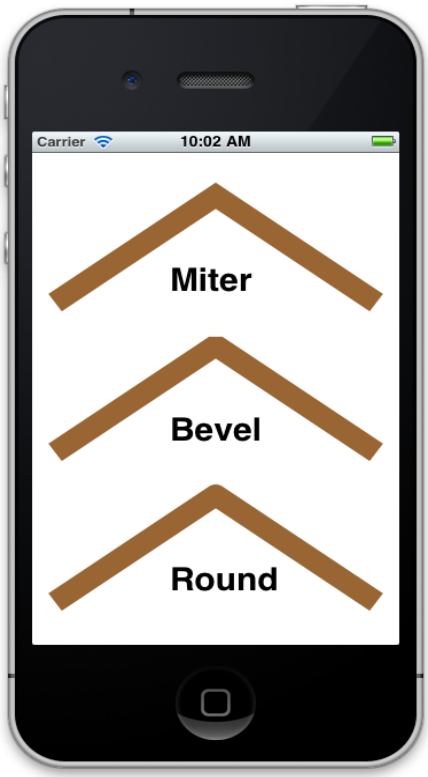

图 15-18.core Graphics 的三种线的接合类型

为了完成这个，我写了一个名为 drawRooftopAtTopPointof:text ToDisplay:lineJoin:的方法，它接受三个参数：

1.一个点，屋顶的顶部在这一点

2.屋顶内显示的文字

3.要使用的接合类型

下面是代码：

```
- (void) drawRooftopAtTopPointof:(CGPoint)paramTopPoint
                    textToDisplay:(NSString *)paramText
                        lineJoin:(CGLineJoin)paramLineJoin{

    /* Set the color that we want to use to draw the line */
    [[UIColor brownColor] set];

    /* Get the current graphics context */
    CGContextRef currentContext = UIGraphicsGetCurrentContext();

    /* Set the line join */
    CGContextSetLineJoin(currentContext, paramLineJoin);

    /* Set the width for the lines */
    CGContextSetLineWidth(currentContext,
                          20.0f);

    /* Start the line at this point */
    CGContextMoveToPoint(currentContext,
                         paramTopPoint.x - 140,
                         paramTopPoint.y + 100);
```

```
    /* And end it at this point */
    CGContextAddLineToPoint(currentContext,
                            paramTopPoint.x,
                            paramTopPoint.y);

    /* Extend the line to another point to make the rooftop */
    CGContextAddLineToPoint(currentContext,
                            paramTopPoint.x + 140,
                            paramTopPoint.y + 100);

    /* Use the context's current color to draw the lines */
    CGContextStrokePath(currentContext);

    /* Draw the text in the rooftop using a black color */
    [[UIColor blackColor] set];

    /* Now draw the text */
    CGPoint drawingPoint = CGPointMake(paramTopPoint.x - 40.0f,
                                       paramTopPoint.y + 60.0f);
     [paramText drawAtPoint:drawingPoint
                   withFont:[UIFont boldSystemFontOfSize:30.0f]];
}
```

现在，我们在视图对象的 drawRect:方法中，调用这个方法：

```
- (void)drawRect:(CGRect)rect{

    [self drawRooftopAtTopPointof:CGPointMake(160.0f, 40.0f)
                    textToDisplay:@"Miter"
                         lineJoin:kCGLineJoinMiter];

    [self drawRooftopAtTopPointof:CGPointMake(160.0f, 180.0f)
                    textToDisplay:@"Bevel"
                         lineJoin:kCGLineJoinBevel];

    [self drawRooftopAtTopPointof:CGPointMake(160.0f, 320.0f)
                    textToDisplay:@"Round"
                         lineJoin:kCGLineJoinRound];
}
```

15.5.4. 参见

XXX

15.6. 构造路径

15.6.1. 问题

你希望能够在图形环境上画任意形状

15.6.2. 方案

构造和绘制路径。

15.6.3. 讨论

一系列的放在一起的点构成了形状。一系列形状放在一起构成了路径。路径可以被 Core Graphics 轻易的管理。在 15.5 小节，我们用 CGContext 间接的操作了路径。不过 Core Graphics 还有直接操作路径的函数，下面我们将会看到它们。

路径属于他们被绘制的图形环境。路径没有边界或者指定的形状，而不像在其上绘制的形状那样。不过路径确实有边界矩形（bounding boxes）。请记住，范围（boundaries）不同于边界矩形。范围是一种限制，超出了它你就不能往画布上绘图，而路径的边界矩形是包含已经被画到指定路径上的所有的形状，点和其他对象的最小矩形。你可以把路径看做邮票，而把图形环境当成信封。每次你给朋友写信的时候，信封可以是相同的，但是你往环境上放置的东西（邮票或路径）可以不同。

在你完成一条路径的绘制后，你可以将它画到图形环境上。熟悉游戏编程的开发者知道缓冲(buffers)的概念，他们绘制场景并且在适当的时间，将图像输出（flush）到屏幕。路径就是这些缓冲。他们就像可以在恰当的时间，被绘制到图形环境上的空画布。

直接操作路径的第一步就是创建他们。创建路径的方法返回一个句柄，你可以在你想往路径上画点东西的时候使用它，将句柄传给 core graphics 可以得到路径的引用。在你创建路径后，你可以往上面加不同的点，线和形状，然后绘制这条路径。你可以在图形环境上填充路径，或者用画笔画出它。下面是你要使用的一些方法：

CGPathCreateMutable 函数
创建一个新的 CGMutablePathRef 类型的可变路径并返回其句柄。我们应该在操作完成后处理它，你后面会看到。

CGPathMoveToPoint 过程
将路径上当前画笔位置移动到 CGPoint 类型的参数指定的点。

CGPathAddLineToPoint 过程
从画笔当前位置向指定位置绘制一条线段。

CGContextAddPath 过程
向图形环境上添加一个路径（由一个路径句柄指定），该路径已经准备好被绘制。

CGContextDrawPath 过程
在图形环境上绘制指定路径

CGPathRelease 过程
释放为路径句柄所分配的内存。

CGPathAddRect 过程

向路径添加一个矩形。这个矩形由 CGRect 结构指定。

你可以要求 CGContextDrawPath 过程执行三种重要的绘制方法：

kCGPathStroke
画线来标记路径的边界或边缘，使用选中的绘图色。

kCGPathFill
用选中的填充色，填充被路径包围的区域。

kCGPathFillStroke
组合绘图和填充。用当前填充色填充路径，并用当前绘图色绘制路径边界。下面我们会看到一个使用此方法的例子。

让我们看个例子。我们要从左上角向右下角，以及右上角向左下角，分别绘制一条蓝色的线，这在屏幕上创建了一个巨大的 X。

此例中，我要隐藏程序的状态条。如果你不想这么干，请继续去看示例代码。如果有状态条，屏幕上显示的就有点不一样了。要隐藏它，请找到 Info.plist，并增加一个 UIStatusBarHidden 键，将其值设置为 YES，如图 15-19 所示。这将强制隐藏状态条。

Key	Type	Value
Localization native development region	String	en
Bundle display name	String	${PRODUCT_NAME}
Executable file	String	${EXECUTABLE_NAME}
Icon file	String	
Bundle identifier	String	com.pixolity.${PRODUCT_NAME:rfc1034identifier}
InfoDictionary version	String	6.0
Bundle name	String	${PRODUCT_NAME}
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle creator OS Type code	String	????
Bundle version	String	1.0
Application requires iPhone environme	Boolean	YES
Main nib file base name	String	MainWindow
Status bar is initially hidden	Boolean	YES
▶ Supported interface orientations	Array	(3 items)

图 15-19. 用 info.plist 文件隐藏 iOS 应用的状态条

```
- (void)drawRect:(CGRect)rect{

    /* Create the path */
    CGMutablePathRef path = CGPathCreateMutable();

    /* How big is our screen? We want the X to cover the whole screen */
    CGRect screenBounds = [[UIScreen mainScreen] bounds];

    /* Start from top-left */
```

```
    CGPathMoveToPoint(path, NULL,
                      screenBounds.origin.x,
                      screenBounds.origin.y);

    /* Draw a line from top-left to bottom-right of the screen */
    CGPathAddLineToPoint(path,
                         NULL,
                         screenBounds.size.width,
                         screenBounds.size.height);

    /* Start another line from top-right */
    CGPathMoveToPoint(path,
                      NULL,
                      screenBounds.size.width,
                      screenBounds.origin.y);

    /* Draw a line from top-right to bottom-left */
    CGPathAddLineToPoint(path,
                         NULL,
                         screenBounds.origin.x,
                         screenBounds.size.height);

    /* Get the context that the path has to be drawn on */
    CGContextRef currentContext = UIGraphicsGetCurrentContext();

    /* Add the path to the context so we candraw it later */
    CGContextAddPath(currentContext,
                     path);

    /* Set the blue color as the stroke color */
    [[UIColor blueColor] setStroke];

    /* Draw the path with stroke color */
    CGContextDrawPath(currentContext, kCGPathStroke);

    /* Finally release the path object */
    CGPathRelease(path);
}
```

传给 CGPath MoveToPoint 之类的过程的 NULL 参数，代表了在给定路径上绘制形状和线时可能用到的变换。关于变换的更多信息，参见 15.10，15.11，15.12 小节。

你可以看到向环境上绘制一个路径有多简单。你要记住的只是创建一个可变路径(CGPathCreateMutable)，向你的图形环境上添加该路径(CGContextAddPath)，并将它绘制到图形环境上(CGContextDraw Path)。如果你运行了上面的代码，可以得到类似于如图 15-20 所示的输出。

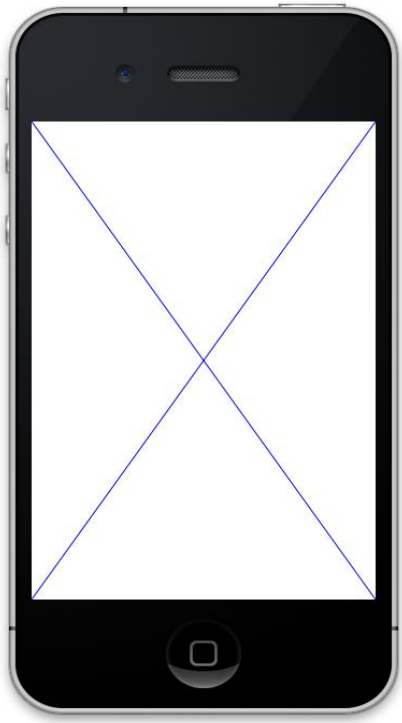

图 15-20. 使用路径在图形环境上绘图

15.6.4. 参见

XXX

15.7. 绘制矩形

15.7.1. 问题

你想在图形环境上绘制矩形

15.7.2. 方案

用 CGPathAddRect 向路径中添加一个矩形，然后在图形环境上绘制这条路径。

15.7.3. 讨论

像我们在 15.6 小节中学到的那样，路径的构造和使用相当简单。core graphics 中你可以用于路径的过程之一是 CGPathAddRect，它能让你将矩形作为路径的一部分进行绘制，如下：

```
- (void)drawRect:(CGRect)rect{
```

```
    /* Create the path first. Just the path handle. */
    CGMutablePathRef path = CGPathCreateMutable();

    /* Here are our rectangle boundaries */
    CGRect rectangle = CGRectMake(10.0f,
                                  10.0f,
                                  200.0f,
                                  300.0f);

    /* Add the rectangle to the path */
     CGPathAddRect(path,
                   NULL,
                   rectangle);

    /* Get the handle to the current context */
    CGContextRef currentContext = UIGraphicsGetCurrentContext();

    /* Add the path to the context */
    CGContextAddPath(currentContext, path);

    /* Set the fill color to cornflower blue */
    [[UIColor colorWithRed:0.20f
                     green:0.60f
                      blue:0.80f
                     alpha:1.0f] setFill];

    /* Set the stroke color to brown */
    [[UIColor brownColor] setStroke];

    /* Set the line width (for the stroke) to 5 */
    CGContextSetLineWidth(currentContext,
                          5.0f);

    /* Stroke and fill the path on the context */
    CGContextDrawPath(currentContext, kCGPathFillStroke);
    /* Dispose of the path */
    CGPathRelease(path);
}
```

这里，我们在路径上画了个矩形，用矢车菊蓝填充它，并用棕色描出了矩形的边缘。图 15-21 显示了程序运行时的样子。

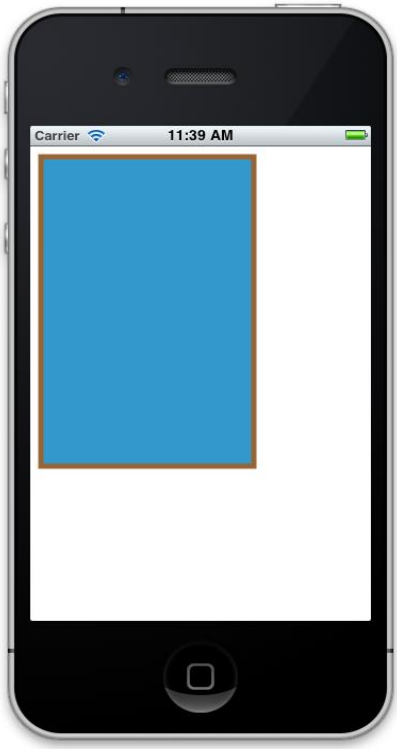

图 15-21. 用路径绘制矩形

如果你有多个矩形要绘制，你可以向 CGPathAddRects 过程传递一个 CGrect 对象的数组，如下：

```
- (void)drawRect:(CGRect)rect{
    /* Create the path first. Just the path handle. */
    CGMutablePathRef path = CGPathCreateMutable();

    /* Here are our first rectangle boundaries */
    CGRect rectangle1 = CGRectMake(10.0f,
                                   10.0f,
                                   200.0f,
                                   300.0f);

    /* And the second rectangle */
    CGRect rectangle2 = CGRectMake(40.0f,
                                   100.0f,
                                   90.0f,
                                   300.0f);

    /* Put both rectangles into an array */
    CGRect rectangles[2] = {
                R ectangle1, rectangle2 };

    /* Add the rectangles to the path */
    CGPathAddRects(path, NULL,
     (const CGRect *)&rectangles, 2);

    /* Get the handle to the current context */
    CGContextRef currentContext = UIGraphicsGetCurrentContext();

    /* Add the path to the context */
    CGContextAddPath(currentContext,
```

```
                                path);

    /* Set the fill color to cornflower blue */
    [[UIColor colorWithRed:0.20f
                     green:0.60f
                      blue:0.80f
                     alpha:1.0f] setFill];

    /* Set the stroke color to black */
    [[UIColor blackColor] setStroke];

    /* Set the line width (for the stroke) to 5 */
    CGContextSetLineWidth(currentContext,
                          5.0f);

    /* Stroke and fill the path on the context */
    CGContextDrawPath(currentContext,
                      kCGPathFillStroke);

    /* Dispose of the path */
    CGPathRelease(path);
}
```

图 15-22 显示了这段代码在模拟器上的输出。我们传给 CGPathAddRects 过程的参数有：

1.要添加矩形的路径句柄

2.变换，如果有的话，应用到矩形上。（关于变换的信息，参见 15.10，15.11 和 15.12 小节）

3.CGRect 数组的引用

4.前一个参数中数组的大小。你传入的这个参数一定要和矩形数相等，以避免此过程出现未知的行为。

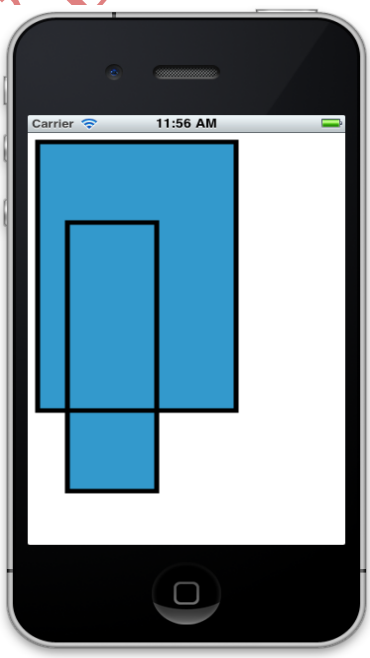

图 15-22. 一次绘制多个矩形

15.7.4. 参见

XXX

15.8. 为形状增加阴影

15.8.1. 问题

你想要为绘制在图形环境上的形状应用阴影。

15.8.2. 方案

使用 CGContextSetShadow 过程。

15.8.3. 讨论

用 core graphics 绘制阴影很容易。图形环境是容纳阴影的元素。就是说，你要先对环境应用阴影，再绘制需要阴影的形状，最后要将阴影从环境上移除（或者设置一个新环境）。我们很快能看到例子。

在 core graphics 中，我们可以使用下列两个过程来为图形环境应用阴影：

CGContextSetShadow 过程

这个过程会创建黑色或灰色的阴影，它接受三个参数：

1.要使用阴影的图形环境

2.阴影的位移，由 CGSize 类型值指定，从每个形状要应用阴影的右下部分开始。位移的 x 值越大，形状右边的阴影就扩散得越远。位移的 y 值越大，下部的阴影就越低。

3.阴影的模糊值，以浮点值(CGFloat)来指定。指定 0.0f 将导致阴影成为固态形状。这个值越高，阴影就越模糊。我们很快能看到例子。

CGContextSetShadowWithColor 过程

这个过程接受的参数和 CGContextSetShadow 完全相同，不过加了一个 CGColorRef 类型的参数，用于设置阴影的颜色。

在此前我提到，图形环境会保留阴影属性，直到我们显式的移除阴影。我们看一个例子来让这一点更清楚吧。下面我们编写代码来绘制两个矩形，第一个有阴影，第二个没有。我们这样绘制第一个矩形：

```
- (void) drawRectAtTopOfScreen{

    /* Get the handle to the current context */
    CGContextRef currentContext = UIGraphicsGetCurrentContext();
    CGContextSetShadowWithColor(currentContext,
                                CGSizeMake(10.0f, 10.0f),
                                20.0f,
                                [[UIColor grayColor] CGColor]);
```

```
    /* Create the path first. Just the path handle. */
    CGMutablePathRef path = CGPathCreateMutable();

    /* Here are our rectangle boundaries */
    CGRect firstRect = CGRectMake(55.0f,
                                  60.0f,
                                 150.0f,
                                 150.0f);

    /* Add the rectangle to the path */
    CGPathAddRect(path,
                  NULL,
                  firstRect);

    /* Add the path to the context */
    CGContextAddPath(currentContext, path);

    /* Set the fill color to cornflower blue */
    [[UIColor colorWithRed:0.20f
                     green:0.60f
                      blue:0.80f
                     alpha:1.0f] setFill];

    /* Fill the path on the context */
    CGContextDrawPath(currentContext,
                      kCGPathFill);

    /* Dispose of the path */
    CGPathRelease(path);
}
```

如果我们在视图对象的 drawRect:方法里调用上面的方法，我们会看到屏幕上画出了一个带有漂亮阴影的矩形，如图 15-23 所示。

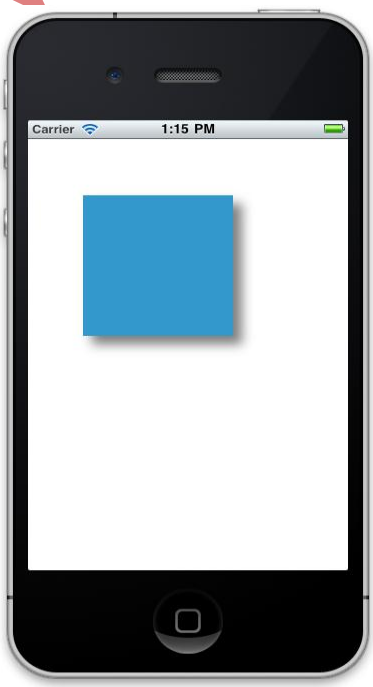

图 15-23. 对矩形使用阴影

下面我们画第二个矩形，我们没有要求有阴影，但是我们保持了在绘制第一个矩形时图形环境的阴影属性：

```
- (void) drawRectAtBottomOfScreen{

    /* Get the handle to the current context */
    CGContextRef currentContext = UIGraphicsGetCurrentContext();

    CGMutablePathRef secondPath = CGPathCreateMutable();
                        CGRect secondRect = CGRectMake(150.0f,
                                                       250.0f,
                                                       100.0f,
                                                       100.0f);
    CGPathAddRect(secondPath,
                  NULL,
                  secondRect);

    CGContextAddPath(currentContext, secondPath);

    [[UIColor purpleColor] setFill];

    CGContextDrawPath(currentContext, kCGPathFill);
    CGPathRelease(secondPath);
}

- (void)drawRect:(CGRect)rect{
    [self drawRectAtTopOfScreen];
    [self drawRectAtBottomOfScreen];
}
```

drawRect:方法首先调用了 drawRectAtTopOfScreen 方法，此后，调用 drawRectAtBottomOfScreen 方法。我们在 drawRectAtBottomOfScreen 中没有要求一个阴影，但是如果你跑一下程序，你会看到类似图 15-24 所示的结果。

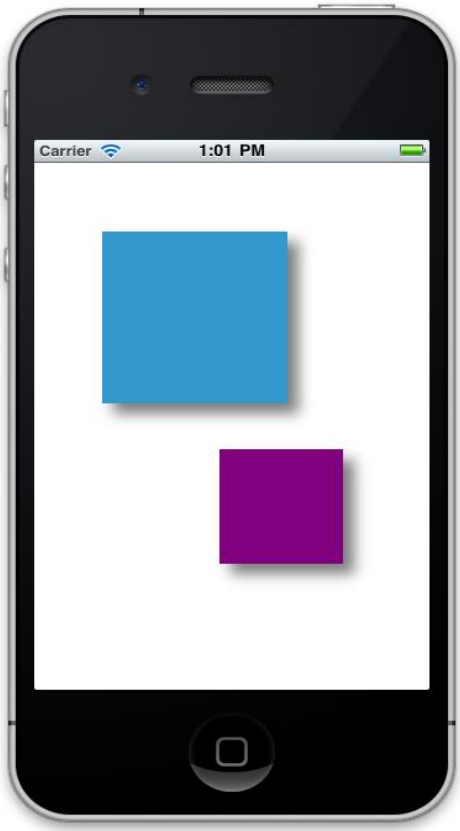

图 15-24. 应用到第二个矩形的阴影不是我们想要的

你马上观察到屏幕下方的第二个矩形被应用了阴影效果。为了避免这个，我们将在对图形环境应用阴影效果之前保存它的状态，并在我们想去掉阴影效果时，恢复之前的状态。

广而言之，图形环境状态的存取不仅限于阴影。恢复图形环境的状态会恢复一切（填充色，字体，线宽，等等）到你之前的设置。所以如果你同时也设置了填充和画笔颜色，这些颜色也会被重置。

你可以通过 CGContextSaveGState 过程来保存图形环境的状态，而通过 CGContextRestoreGState 过程恢复之前的状态。所以如果修改 drawRectAtTopOfScreen 过程，在应用阴影前保存图形环境状态，并在绘制路径后恢复其状态，我们将得到不同的结果，如图 15-25 所示：

```
- (void) drawRectAtTopOfScreen{

/* Get the handle to the current context */
CGContextRef currentContext = UIGraphicsGetCurrentContext();

CGContextSaveGState(currentContext);

CGContextSetShadowWithColor(currentContext,
                            CGSizeMake(10.0f, 10.0f), 20.0f,
                                [[UIColor grayColor] CGColor]);
  /* Create the path first. Just the path handle. */
  CGMutablePathRef path = CGPathCreateMutable();

  /* Here are our rectangle boundaries */
```

```
    CGRect firstRect = CGRectMake(55.0f,
                                  60.0f,
                                  150.0f,
                                  150.0f);

    /* Add the rectangle to the path */
    CGPathAddRect(path,
                  NULL,
                  firstRect);

    /* Add the path to the context */
    CGContextAddPath(currentContext,
                     path);

    /* Set the fill color to cornflower blue */
    [[UIColor colorWithRed:0.20f
                     green:0.60f
                      blue:0.80f
                     alpha:1.0f] setFill];

    /* Fill the path on the context */
    CGContextDrawPath(currentContext,
                      kCGPathFill);

    /* Dispose of the path */
    CGPathRelease(path);

    /* Restore the context to how it was when we started */
    CGContextRestoreGState(currentContext);
}
```

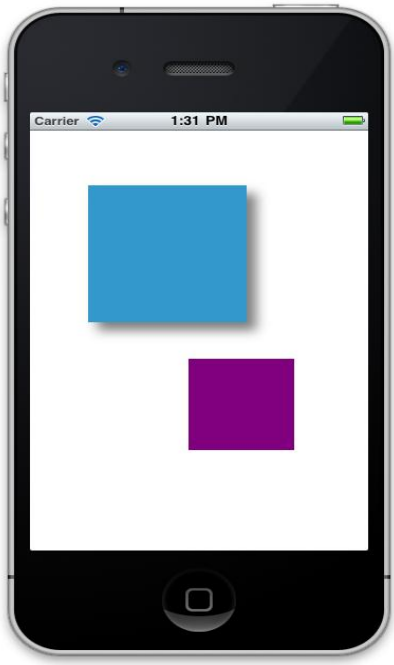

图 15-25. 保存图形环境状态来准确的使用阴影

15.8.4. 参见

XXX

15.9. 绘制渐变

15.9.1. 问题

你想使用不同的颜色在图形环境上绘制渐变。

15.9.2. 方案

使用 CGGradientCreateWithColor 函数。

15.9.3. 讨论

在 15.3 节学完关于颜色的部分后，现在我们可以更好的利用这些技术，而不仅限于画几个简单的矩形和带色文字了。

core graphics 允许程序员创建两类渐变：轴向的（axial）和放射状的（radial）。（这本书里只讨论轴向渐变。）轴向渐变是以一种颜色为起点，而以另一种颜色为终点的渐变（虽然可以在起点和终点用一种颜色，但是那不会产生渐变）。“轴向”的意思是和某个轴有关。两个点（起点和终点）形成一条线段，这就是渐变要被绘制的轴。关于轴向渐变的例子如图 15-26 所示。

图 15-26. 一个开始于蓝色结束于绿色的轴向渐变

为了创建一个轴向渐变，你必须调用 CGGradientCreateWithColorComponents 函数。函数的返回值是一个 CGGradientRef 类型的新渐变。它是渐变的句柄。当你用完渐变后，你必须调用 CGGradientRelease 过程，并传入你之前从 CGGradientCreateWithColorComponents 获得的渐变的句柄。

CGGradientCreateWithColorComponents 函数接受四个参数：

一个颜色空间
这是一个颜色范围的容器，这个参数必须是 CGColorSpaceRef 类型的，我们只要传入 CGColorSpaceCreateDeviceRGB 函数的返回值即可，该函数会给我们一个 RGB 颜色空间。

颜色组件的数组（详情参见 15.3 小节）
这个数组必须包含红，绿，蓝和 alpha 值，都以 CGFloat 表示。数组中元素的个数和下面两个参数紧密相连。你必须在这个数组中包含足够的值来指定第四个参数中的位置数量。所以，如果你请求两个位置（起点和终点），你就必须在数组中提供两种颜色。因为每种颜色都是由红、绿、蓝和 alpha 组成，这个数组必须有 2X4 项：两种颜色分别 4 个。别担心你不能接受这些规定，通过后面的例子你最终会理解。

颜色数组中的颜色位置
这个参数控制了渐变过度的迅速程度。元素的个数一定要和第 4 个参数一样。如果我请求 4 个颜色，比如说，我们希望第 1 种颜色为起始色，最后一种颜色为终止色，这时我们就要提供包含两个 CGFloat 类型元素的数组，第一个元素设置为 0.0f（颜色数组的第一个），第二个元素设置为 3.0f（颜色数组的第 4 个）。中间的两个颜色的值决定了在起点和终点间如何插入颜色。同样不需要担心这些太难，我会给你很多例子帮助你完全理解这个概念。

位置数量
指定我们想要有多少种颜色和位置

我们来看例子。假设我们想绘制如图 15-26 所示的渐变，下面是步骤：

1.选择渐变的开始和结束点 --- 它变换的轴。这里，我们选择了从左向右移动。想像你在沿着一条假想的水平线移动时改变颜色。沿着那条线，我们会传播这些颜色使得每条垂直于这条水平线的线上只包含一种颜色。这里，这些垂直的线将会是图 15-26 中的垂线。近距离观察那些垂直线，其中的每一条从上到下都只包含一种颜色。这就是轴向渐变的原理。OK，现在理论讲了很多---让我们开始第二步吧。

2.如前所述，现在我们要创建一个颜色空间来作为 CGGradientCreateWithColorComponents 函数的第一个参数：

```
CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();
```


在我们用完这个颜色空间后，我们会释放它。

3.根据图 15-26 所选择的颜色，我们选择蓝色作为起点（左边），绿色作为终点（右

边）。我选择的变量名（startColorComponents 和 endColorComponents）是特意选定来帮助我们记住我们对每个颜色在做的事情。实际上我们会使用数组位置来指定起点和终点：

```
UIColor *startColor = [UIColor blueColor];
   CGFloat *startColorComponents =
   (CGFloat *)CGColorGetComponents([startColor CGColor]);

   UIColor *endColor = [UIColor greenColor];
   CGFloat *endColorComponents =
   (CGFloat *)CGColorGetComponents([endColor CGColor]);
```

如果你不记得颜色组件背后的概念，我建议在你继续之前，看看 15.3 小节。

4.在你获得每个颜色的组件后，我们将他们放到一个简单的数组中，以传入 CGGradientCreateWithColorComponents 函数：

```
CGFloat colorComponents[8] = {

      /* Four components of the blue color (RGBA) */
      startColorComponents[0],
      startColorComponents[1],
      startColorComponents[2],
      startColorComponents[3], /* First color = blue */

      /* Four components of the green color (RGBA) */

      endColorComponents[0],
      endColorComponents[1],
      endColorComponents[2],
      endColorComponents[3], /* Second color = green */
   };
```

5. 因为这个数组只有两种颜色，我们需要指定第一个是渐变的最开始(位置 0.0)而第二个在最后(位置 1.0)。让我们将这些索引放到数组中作为参数传递到 CGGradientCreateWithColorComponents 函数：

```
CGFloat colorIndices[2] = {
      0.0f, /* Color 0 in the colorComponents array */
      1.0f, /* Color 1 in the colorComponents array */
      };
```

6.现在我们要做的只剩下用之前创建的参数实际调用 CGGradientCreateWithColor Components 函数：

```
CGGradientRef gradient =
CGGradientCreateWithColorComponents
(colorSpace,
   (const CGFloat *)&colorComponents,
   (const CGFloat *)&colorIndices,
```

```
2);
```

7.非常棒！现在我们在 gradient 变量中保存了我们的渐变对象。在我们忘记之前，还必须释放之前在 CGColorSpaceCreateDeviceRGB 函数中创建的颜色空间：

```
CGColorSpaceRelease(colorSpace);
```

现在我们将使用 CGContextDrawLinearGradient 来往图形环境上面绘制轴向渐变。这个过程接受 5 个参数：

图形环境

指定轴向渐变被绘制到的图形环境

轴向渐变

轴向渐变对象的句柄。我们在 CGGradientCreateWithColorComponents 函数中创建这个渐变对象。

起点

图形环境中的一点，由 CGPoint 指定，它指定了渐变的起点。

终点

图形环境中的一点，由 CGPoint 指定，它指定了渐变的终点。

渐变绘制选项

指定了当你指定的起点或终点不是图形环境的边缘时的行为。你可以使用你的开始色或结束色来填充渐变之外的区域。指定下列参数值之一：

kCGGradientDrawsAfterEndLocation

在渐变终点之后将渐变扩展到所有点

kCGGradientDrawsBeforeStartLocation

在渐变起点之前将渐变扩展到所有点

0

不会以任何方式扩展渐变

为了在两端都扩展颜色，可以用逻辑 OR（使用|操作符）同时指定“之后”和“之前”参数。我们看下面的例子：

```
CGRect screenBounds = [[UIScreen mainScreen] bounds];

CGPoint startPoint, endPoint;

startPoint = CGPointMake(0.0f,
                         screenBounds.size.height / 2.0f);

endPoint = CGPointMake(screenBounds.size.width,
                       startPoint.y);

CGContextDrawLinearGradient (currentContext,
                             gradient,
                             startPoint,
                             endPoint,
                             0);
CGGradientRelease(gradient);
```

代码结束位置所释放的渐变句柄是我们在之前的示例代码中创建的。

这段代码的输出显然如图 15-26 所示。因为我们从最左边的点开始将渐变拖拽到最右边的点，所以我们不能利用 CGContextDrawLinearGradient 过程最后的“渐变绘制选项”参数。让我们补救一下吧？绘制一个如图 15-27 所示的渐变如何？

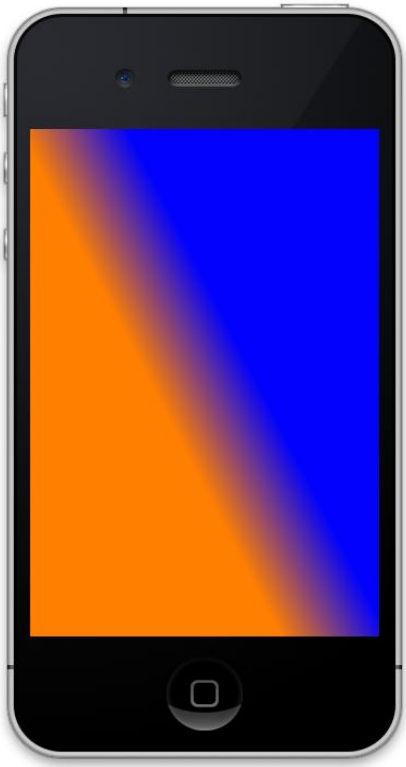

图 15-27. 一个对起点和终点进行颜色扩展的渐变

我们会使用这部分早先用过的相同的代码来得到结果：

```
- (void)drawRect:(CGRect)rect{

CGContextRef currentContext = UIGraphicsGetCurrentContext();

CGContextSaveGState(currentContext);
CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();

UIColor *startColor = [UIColor orangeColor];
CGFloat *startColorComponents =
(CGFloat *)CGColorGetComponents([startColor CGColor]);

UIColor *endColor = [UIColor blueColor];
CGFloat *endColorComponents =
(CGFloat *)CGColorGetComponents([endColor CGColor]);

CGFloat colorComponents[8] = {
```

```
        /* Four components of the orange color (RGBA) */
        startColorComponents[0],
        startColorComponents[1],
        startColorComponents[2],
        startColorComponents[3], /* First color = orange */

        /* Four components of the blue color (RGBA) */
        endColorComponents[0],
        endColorComponents[1],
        endColorComponents[2],
        endColorComponents[3], /* Second color = blue */
    };
CGFloat colorIndices[2] = {
    0.0f, /* Color 0 in the colorComponents array */
    1.0f, /* Color 1 in the colorComponents array */
};

CGGradientRef gradient = CGGradientCreateWithColorComponents
  (colorSpace,
    (const CGFloat *)&colorComponents,
    (const CGFloat *)&colorIndices,
    2);

CGColorSpaceRelease(colorSpace);

CGPoint startPoint, endPoint;
startPoint = CGPointMake(120,
                         260);

endPoint = CGPointMake(200.0f,
                         220);

CGContextDrawLinearGradient (currentContext,
                             gradient,
                            startPoint,
                             endPoint,
                             kCGGradientDrawsBeforeStartLocation |
                             kCGGradientDrawsAfterEndLocation);

CGGradientRelease(gradient);
CGContextRestoreGState(currentContext);
}
```

你可能难以理解将 kCGGradientDrawsBeforeStartLocation 和 kCGGradientDrawsAfterEndLocation 值混合传入 CGContextDrawLinearGradient 过程，是如何产生如图 15-27 所示的对角线效果的。那么让我们将这些值从 CGContextDrawLinearGradient 过程的那个参数中去掉，仅仅像之前一样传入 0。图 15-28 显示了结果。

图 15-28. 不使用拉伸颜色（stretched colors）的轴向渐变

很容易得出结论，图 15-28 和图 15-27 中使用的是相同的渐变。但是，图 15-27 中的渐变扩展了起点和终点的颜色，使渐变充满整个图形环境，这就是为什么你能看见整个屏幕被颜色所覆盖。

15.9.4. 参见

XXX

15.10. 移动图形环境上所绘制的形状

15.10.1. 问题

你想移动被绘制到图形环境上的一切，到一个新的位置，而不需要修改你的绘图代码。或者你只是想轻易的移动你的图形环境。

15.10.2. 方案

使用 CGAffineTransformMakeTranslation 函数创建一个仿射位移变换（affine translation

transformation）。

15.10.3. 讨论

15.7 小节提到了变换。顾名思义：就是改变图形显示的方式。变换在 core graphics 中是图形在绘制前被应用的一些对象。比如，你可以创建一个位移变换。移动什么，你可能会问？一个位移变换是一种让你能移动形状或图形环境的机制。

其他类型的变换包含旋转（参见 15.12）和缩放（参见 15.11）这些都是仿射变换的例子，仿射变换会在最终版本中将原来的每个点映射到另一个点。本书讨论的所有变换都将是仿射变换。

位移变换能将路径或图形环境上的形状的当前位置移动到另一个相对的位置。比如说，你在（10，20）画了一个点，对其使用了（30，40）的仿射变换，然后再画它，这个点会被画到（40，60），因为 10+30 的和为 40，20+40 的和为 60。

为了创建一个新的位移变换，我们必须使用 CGAffineTransformMakeTranslation 函数，它会返回一个 CGAffineTransform 类型的仿射变换。此函数的两个参数指定了以点为单位的 x 和 y 方向上的位移。

在 15.7 小节中，我们看见了 CGPathAddRect 过程的第二个参数是一个 CGAffineTransform 类型的变换对象。为了从原始位置向另一个位置移动矩形，你只要创建一个放射变换，并指定你希望的 x 和 y 轴的位移，并将变换作为第二个参数传入 CGPathAddRect，如下：

```
- (void)drawRect:(CGRect)rect{

    /* Create the path first. Just the path handle. */
    CGMutablePathRef path = CGPathCreateMutable();

    /* Here are our rectangle boundaries */
    CGRect rectangle = CGRectMake(10.0f,
                                  10.0f,
                                  200.0f,
                                  300.0f);

    /* We want to displace the rectangle to the right by 100 points but want to keep the y position untouched */
    CGAffineTransform transform = CGAffineTransformMakeTranslation(100.0f, 0.0f);

    /* Add the rectangle to the path */
    CGPathAddRect(path,
                  &transform,
                  rectangle);

    /* Get the handle to the current context */
    CGContextRef currentContext = UIGraphicsGetCurrentContext();

    /* Add the path to the context */
    CGContextAddPath(currentContext,
                     path);

    /* Set the fill color to cornflower blue */
    [[UIColor colorWithRed:0.20f
```

```
                        green:0.60f
                         blue:0.80f
                        alpha:1.0f] setFill];

    /* Set the stroke color to brown */
    [[UIColor brownColor] setStroke];

    /* Set the line width (for the stroke) to 5 */
    CGContextSetLineWidth(currentContext,
                          5.0f);

    /* Stroke and fill the path on the context */
    CGContextDrawPath(currentContext,
                      kCGPathFillStroke);

    /* Dispose of the path */
    CGPathRelease(path);
}
```

图 15-29 显示了这点代码位于某个视图对象内部的输出结果。

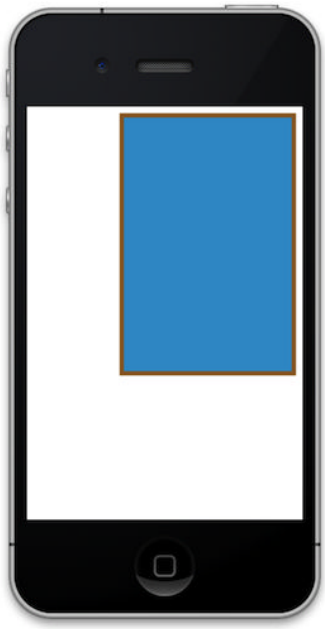

图 15-29. 一个应用了仿射位移变换的矩形

比较图 15-29 和图 15-21.你能发现不同点吗？检查这两个图形的代码，你会发现每个代码块中的矩形所指定的 x 和 y 点是相同的。只不过在图 15-29 中，我们在将矩形加到路径中时，对矩形应用了仿射位移变换。

除了向绘制到路径中的形状应用变换之外，我们还可以使用 CGContextTranslateCTM 过程对图形环境应用变换。这会在当前变换矩阵（CTM）上应用位移变换。当前的变换矩阵，虽然听起来很复杂，其实很容易理解。你可以将 CTM 当作你的图形环境的中心如何被设置，而你绘制的每个点又是如何投射到屏幕上的工具。比如说，当你要求 core graphics 在（0，0）点绘图时，core graphics 查看 CTM 来得到屏幕的中心。CTM 会做一些计算然后告诉 core graphics（0，0）这点确实是在屏幕的左上角。使用如 CGContextTranslateCTM 这样的过程函数，你可以修改 CTM 的配置，从而改变图形环境上的每个形状，使他们绘制到画

布上的另一个位置。下面的例子中，我们能通过对 CTM 应用位移变换而不是直接对矩形操作，而得到和图 15-29 完全相同的屏幕：

```
- (void)drawRect:(CGRect)rect{

    /* Create the path first. Just the path handle. */
    CGMutablePathRef path = CGPathCreateMutable();

    /* Here are our rectangle boundaries */
    CGRect rectangle = CGRectMake(10.0f,
                                  10.0f,
                                  200.0f,
                                  300.0f);

    /* Add the rectangle to the path */
    CGPathAddRect(path,
                  NULL,
                  rectangle);

    /* Get the handle to the current context */
    CGContextRef currentContext = UIGraphicsGetCurrentContext();

    /* Save the state of the context to revert back to how it was at this state, later */
    CGContextSaveGState(currentContext);

    /* Translate the current transformation matrix to the right by 100 points */
    CGContextTranslateCTM(currentContext, 100.0f,
                          0.0f);

    /* Add the path to the context */
    CGContextAddPath(currentContext, path);

    /* Set the fill color to cornflower blue */
    [[UIColor colorWithRed:0.20f
                     green:0.60f
                      blue:0.80f
                     alpha:1.0f] setFill];

    /* Set the stroke color to brown */

    [[UIColor brownColor] setStroke];

    /* Set the line width (for the stroke) to 5 */
    CGContextSetLineWidth(currentContext,
                          5.0f);

    /* Stroke and fill the path on the context */
    CGContextDrawPath(currentContext, kCGPathFillStroke);

    /* Dispose of the path */
    CGPathRelease(path);

    /* Restore the state of the context */
    CGContextRestoreGState(currentContext);
}
```

在运行上述代码后，你会发现结果同图 15-29 一模一样。

15.10.4. 参见

XXX

15.11. 缩放绘制到图形环境上的形状

15.11.1. 问题

你想动态的缩放你图形环境上的形状。

15.11.2. 方案

使用 CGAffineTransformMakeScale 函数创建一个仿射缩放变换。

15.11.3. 讨论

15.10 解释了什么是变换，以及如何对图形环境和形状应用变换。而另一种可用的变换就是缩放。你能很容易的请求 core graphics 来缩放一个形状，比如圆形，让他放大 100 倍。

为了创建一个仿射缩放变换，可以用 CGAffineTransformMakeScale 函数，此函数会返回一个 CGAffineTransform 类型的变换对象。如果你想直接对图形环境应用缩放变换，可以使用 CGContextScaleCTM 过程函数来缩放当前变换矩阵（CTM）。关于 CTM 的更多信息，参见 15.10 小节。

缩放变换函数接受两个参数：一个缩放 x 轴，另一个缩放 y 轴。再看一眼图 15-21 中的矩形。如果我们想将这个矩形的长和宽缩小到一半，我们只要将 x 和 y 轴都设为 0.5 即可（原始值的一半），如下：

```
/* Scale the rectangle to half its size */
CGAffineTransform transform =
    CGAffineTransformMakeScale(0.5f, 0.5f);

/* Add the rectangle to the path */
CGPathAddRect(path,
              &transform,
              rectangle);
```

图 15-30 显示了应用了这个缩放变换之后的结果。

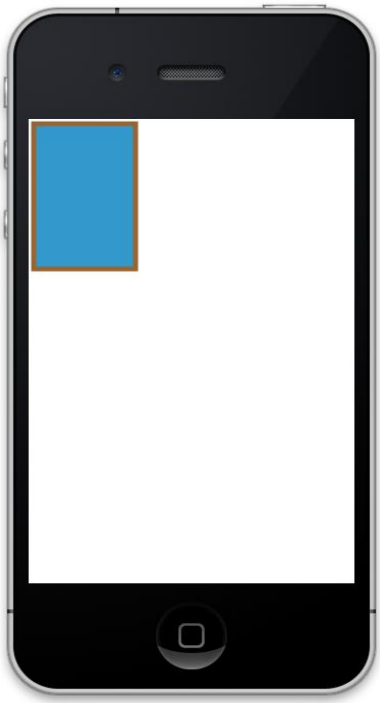

图 15-30. 缩放一个矩形

除了使用 CGAffineTransformMakeScale 函数，你还可以用 CGContextScaleCTM 过程函数将缩放变换应用到一个图形环境。下面的代码将会得到同前面的例子相同的结果，如图 15-30 所示：

```
- (void)drawRect:(CGRect)rect{

    /* Create the path first. Just the path handle. */
    CGMutablePathRef path = CGPathCreateMutable();

    /* Here are our rectangle boundaries */
    CGRect rectangle = CGRectMake(10.0f,
                                  10.0f,
                                  200.0f,
                                  300.0f);

    /* Add the rectangle to the path */
    CGPathAddRect(path,
                  NULL,
                  rectangle);

    /* Get the handle to the current context */
    CGContextRef currentContext = UIGraphicsGetCurrentContext();

    /* Scale everything drawn on the current
      graphics context to half its size */
      CGContextScaleCTM(currentContext,
                        0.5f,
                        0.5f);

    /* Add the path to the context */
    CGContextAddPath(currentContext, path);
    /* Set the fill color to cornflower blue */
```

```
    [[UIColor colorWithRed:0.20f
                     green:0.60f
                      blue:0.80f
                     alpha:1.0f] setFill];

    /* Set the stroke color to brown */
    [[UIColor brownColor] setStroke];

    /* Set the line width (for the stroke) to 5 */
    CGContextSetLineWidth(currentContext,
                          5.0f);

    /* Stroke and fill the path on the context */
    CGContextDrawPath(currentContext,
                      kCGPathFillStroke);

    /* Dispose of the path */
    CGPathRelease(path);
}
```

15.11.4. 参见

XXX

15.12. 旋转绘制在图形环境上的形状

15.12.1. 问题

你想要旋转图形环境上的内容，而不改变绘图代码。

15.12.2. 方案

使用 CGAffineTransformMakeRotation 函数来创建一个仿射旋转变换。

15.12.3. 讨论

在开始这部分之前，我强烈建议你阅读 15.10 和 15.11 中的内容。为了避免内容重复，我已经将前面部分讲过的内容从本小节中剔除。

就像缩放和平移一样，你可以对绘制于路径上的图形，或者图形环境，应用旋转变换。你可以使用 CGAffineTransformMakeRotation 函数，并传入以弧度为单位的旋转值来得到一

个 CGAffineTransform 类型的旋转变换。然后你就可以将它应用到路径和形状上了。如果你想以指定角度旋转整个图形环境，就必须使用 CGContextRotateCTM 过程函数。

让我们对图 15-21 中的那个矩形实施 45 度顺时针的旋转（如图 15-31）.你提供给旋转的值必须是弧度值。正值导致顺时针旋转，而负值导致逆时针旋转：

```
/* Rotate the rectangle 45 degrees clockwise */
CGAffineTransform transform = CGAffineTransformMakeRotation((45.0f * M_PI) / 180.0f);

/* Add the rectangle to the path */
CGPathAddRect(path,
              &transform,
              rectangle);
```

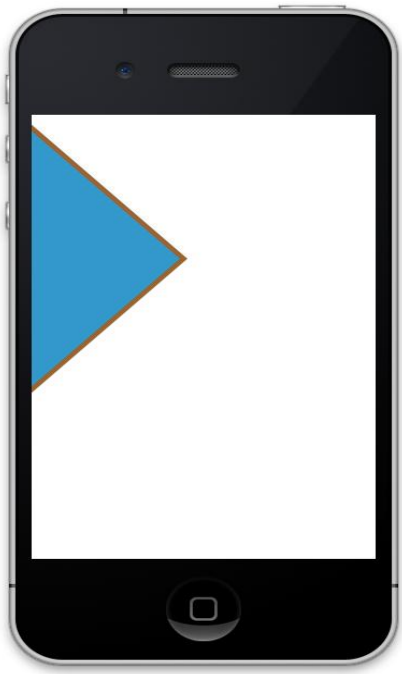

图 15-31. 旋转一个矩形

如我们在 15.11 小节所见，我们可以使用 CGContextRotateCTM 过程函数对整个图形环境直接使用变换。

15.12.4. 参见

XXX

15.13. 动画之视图的移动

15.13.1. 问题

如果你想要移动视图

15.13.2. 方案

使用 UIView 的动画方法来移动你的视图。

15.13.3. 讨论

在 iOS 中有几种途径来执行动画：这种动画的能力可以是由底层 API 提供的，也可以是高级别的。我们可以通过 UIKit 来获取最高级别的实现，这也是我们在本节要讨论的内容。UIKit 包含一些底层核心动画方法并为我们呈现了一套干净的 API 以供我们使用。

在 UIKit 中一个动画的执行开始于 UIView 的类方法：beginAnimations:context:它的第一个参数是你为这个动画设置的名字，这个参数是可选的，第二个为你晚些可以取回并传给该动画的委托方法的一个上下文环境参数，这个参数同样也是可选的。

在你调用 beginAnimations:context:方法来启动一个动画后，动画并不会立即被执行，直到你调用 UIView 类的 commitAnimations 类方法。你对一个视图对象执行的介于 beginAnimations:context:方法跟 commitAnimations 方法之间的操作（例如移动）会在 commitAnimations 被执行后才会生效。下面让我们来看一个例子。

就像我们在 15.4 小节看到的，我在我的应用资源中引入了一张叫做 Xcode.png 的图片。这是一张 Xcode 的图标图片，我是在谷歌图片搜索中找到的（参见 15-14）。现在，在我的视图控制器中（参见 15.0 小节），我想要将这张图片放在一个 UIImageView 类型的图片视图中，然后把这张图片从屏幕的左上角移动到右下角。

下面是实现上述任务的步骤：

1、打开你的视图控制器的 .h 文件。

2、在你的视图控制器中定义一个 UIImageView 属性，取名为 xcodeImageView，就像这样

```
#import <UIKit/UIKit.h>

@interface Animating_and_Moving_ViewsViewController : UIViewController

@property (nonatomic, strong) UIImageView *xcodeImageView;

@end
```

3、在你的视图控制器的 .m 文件里，为上一步中定义的图片视图属性声明 synthesize，并确保在恰当的时候将其销毁：

```
#import "Animating_and_Moving_ViewsViewController.h"

@implementation Animating_and_Moving_ViewsViewController
@synthesize xcodeImageView;

- (void)viewDidUnload{
 [super viewDidUnload];
 self.xcodeImageView = nil;
}
...
```

4、当你的视图加载时，将 Xcode.png 图片加载到一个 UIImage 实例中：

```
- (void) viewDidLoad{
[super viewDidLoad];

 UIImage *xcodeImage = [UIImage imageNamed:@"Xcode.png"];

 self.xcodeImageView = [[UIImageView alloc]
                        initWithImage:xcodeImage];

 /* Just set the size to make the image smaller */
 [self.xcodeImageView setFrame:CGRectMake(0.0f,
                                          0.0f,
                                          100.0f,
                                          100.0f)];
 self.view.backgroundColor = [UIColor whiteColor];
 [self.view addSubview:self.xcodeImageView];
}
```

5、图 15-32 向我们展示了当我们在 iOS 运行这个程序时它的样子：

图 15-32 为一个视图对象添加一张图片

6、现在当我们的视图看起来像上面的界面那样时，在我们的视图控制器的 viewDidAppear:实例方法中，我们将为我们的图片视图启动动画块并执行将图片从它的初始位置—屏幕左上角，移动到右下角的动作。我们将确保这个动画的运行时间为 5 秒：

```
- (void) viewDidAppear:(BOOL)paramAnimated{

   [super viewDidAppear:paramAnimated];

   /* Start from top left corner */
      [self.xcodeImageView setFrame:CGRectMake(0.0f,
                                               0.0f,
                                               100.0f,
                                               100.0f)];
```

```
    [UIView beginAnimations:@"xcodeImageViewAnimation"
                    context:(__bridge void *)self.xcodeImageView];

    /* 5 seconds animation */
    [UIView setAnimationDuration:5.0f];

    /* Receive animation delegates */
     [UIView setAnimationDelegate:self];

     [UIView setAnimationDidStopSelector:
     @selector(imageViewDidStop:finished:context:)];

    /* End at the bottom right corner */
       [self.xcodeImageView setFrame:CGRectMake(200.0f,
                                                350.0f,
                                                100.0f,
                                                100.0f)];
       [UIView commitAnimations];
}
```

API 为我们的视图控制器提供了 imageViewDidStop:finished:context:代理方法，这样 UIKit 就可以在动画结束时调用它。这个操作是可选的，我们的例子中我只是简单的打印了一些消息日志来证明该方法被调用了。晚些会有一个例子向我们展示了当动画结束时，如何使用该方法来结束（剔除）其它的活动：

现在当你运行这个应用时，你会注意到只要你的视图一显示，在图 15-32 中的那张图片就会开始向屏幕右下角移动，就像图 15-33 展示的那样，历时 5 秒。

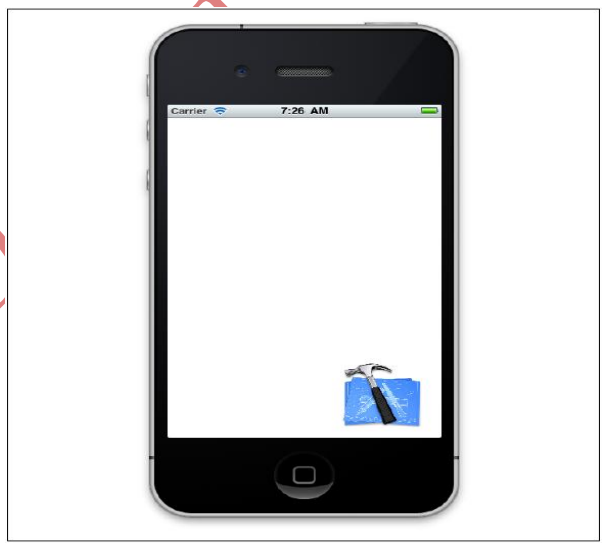

图 15-33 图片移动到屏幕的右下角

另外，在控制台的输出中，当你在等动画结束时你会看到一些类似这样的信息：

```
Animation finished.
Animation ID = xcodeImageViewAnimation
Image View = <UIImageView: 0x68221a0;
                        frame = (200 350; 100 100);
                        opaque = NO;
                        userInteractionEnabled = NO;
```

```
layer = <CALayer: 0x68221d0>>
```

现在，让我们看看一些概念以及我们是如何移动这个图片视图的。下面是当你使用 UIKit 来执行动画时需要知道的 UIView 类的一些重要的类方法：

beginAnimations:context:

启动一个动画块。任何在此类方法调用后你提交给视图的动画属性的改变会在动画提交后得到执行。

setAnimationDuration:

设置动画的历时长度，单位为秒。

setAnimationDelegate:

设置对象来接收，可能发生在动画执行前，执行中或者执行后发生的各种事件的委托对象。设置一个委托对象并不会马上触发动画委托。你必须对视图对象使用不同的设置类方法来告知 UIKit 你会用你的委托对象中的哪些选择器来接收哪些委托消息。

setAnimationDidStopSelector:

在委托对象中设置动画结束时要调用的方法。这个方法按照下列顺序接收 3 个参数：

1、一个 NSString:类型的动画标识，该参数包含的标识会在动画开始时传给 UIView 的类方法 beginAnimations:context:。

2、一个 NSNumber: 类型的“结束”指示器， 这个参数会在 NSNumber 中放一个 bool 值，当动画在被代码结束掉前若动画已经完全完成，运行时会将其设置为 YES；要是值为 NO,则意味着动画在完成之前被中断了。

3、一个 void *:类型的上下文环境。这就是当动画开始时传递给 UIView 类的 beginAnimations:context:类方法的的上下文环境。

setAnimationWillStartSelector:

设置动画要启动时委托对象中要调用的选择器。传给这个类方法的选择器有两个参数，顺序如下：

1、一个 NSString:类型的动画标识，运行时会把该标识设置给在动画开始时要传给 UIView 的类方法 beginAnimations:context:的动画标识。

2、一个 void *:类型的上下文环境。这就是当动画开始时传递给 UIView 类的 beginAnimations:context:类方法的的上下文环境。

setAnimationDelay:

设置动画开始的延迟时间，单位为秒。例如，当这个值设为 3.0f 时，动画会在它被提交后的 3 秒开始执行。

setAnimationRepeatCount:

设置一个动画块要重复该动画的次数。

现在，我们已经了解了一些有用的 UIView 的类方法来帮助我们移动视图，接下来让我们看另外一个动画。在这个例子的代码中，我想要两个图片视图，显示的是相同的图片，同时显示在屏幕上：一张在屏幕左上角，一张在右下角，就像图 15-34 所展示的

图 15-34 动画的启动位置

本节中，我会调用左上角的图片 1 和右下角的图片 2

我们在代码中要做的是像上面提到的创建两个图片，一个在左上角一个在右下角。接着，我们希望图 1 向图二移动，历时 3 秒，然后逐渐消失。当图 1 接近图 2 时，我们希望图 2 开始启动它的动画并左上角开始移动，即图 1 开始所在的位置。我们同样希望图 2 经过 3 秒完成它的动画然到达终点逐渐消失。当你在终端或者 iOS 模拟器上跑这个应用的时候，它看起来棒极了。下面我们会展示下如何编写这些代码：

1、在你的视图控制器的 .h 文件中，定义两个图片视图：

```
#import <UIKit/UIKit.h>

@interface Animating_and_Moving_ViewsViewController : UIViewController

@property (nonatomic, strong) UIImageView *xcodeImageView1;
@property (nonatomic, strong) UIImageView *xcodeImageView2;

@end
```

2、在你的视图控制器的 .m 文件中，确保你为上述两个视图声明了 synthesize，因为它们是属性

```
#import "Animating_and_Moving_ViewsViewController.h"

@implementation Animating_and_Moving_ViewsViewController

@synthesize xcodeImageView1;
@synthesize xcodeImageView2;
...
```

3、确保当视图卸载时将两个图片视图销毁掉

```
- (void)viewDidUnload{
 [super viewDidUnload];
 self.xcodeImageView1 = nil;
 self.xcodeImageView2 = nil;
}
```

4、在你的视图控制器的 viewDidLoad 实例方法中，将两个图片视图都初始化并将其都放到你的视图中：

```
- (void) viewDidLoad{
    [super viewDidLoad];
    UIImage *xcodeImage = [UIImage imageNamed:@"Xcode.png"];
    self.xcodeImageView1 = [[UIImageView alloc]
    initWithImage:xcodeImage];

    self.xcodeImageView2 = [[UIImageView alloc]
    initWithImage:xcodeImage];

   /* Just set the size to make the images smaller */
    [xcodeImageView1 setFrame:CGRectMake(0.0f,
                                         0.0f,
                                         100.0f,
                                         100.0f)];

    [xcodeImageView2 setFrame:CGRectMake(220.0f,
                                         350.0f,
                                         100.0f,
                                         100.0f)];

    self.view.backgroundColor = [UIColor whiteColor];
    [self.view addSubview:self.xcodeImageView1];
    [self.view addSubview:self.xcodeImageView2];
}
```

5、为你的视图控制器实现一个 startTopLeftImageViewAnimation 实例方法。这个方法就像它的名字描述的，会为图 1 执行动画，从屏幕左上角移动到右下角并逐渐消失。消失操作只用简单的将 appha 值设为 0 就可以了：

```
- (void) startTopLeftImageViewAnimation{

    /* Start from top left corner */
    [self.xcodeImageView1 setFrame:CGRectMake(0.0f,
                                              0.0f,
                                              100.0f,
                                              100.0f)];

    [self.xcodeImageView1 setAlpha:1.0f];

    [UIView beginAnimations:@"xcodeImageView1Animation"
                    context:(__bridge void *)self.xcodeImageView1];

    /* 3 seconds animation */
    [UIView setAnimationDuration:3.0f];
```

```
        /* Receive animation delegates */
        [UIView setAnimationDelegate:self];

        [UIView setAnimationDidStopSelector:
        @selector(imageViewDidStop:finished:context:)];

        /* End at the bottom right corner */
        [self.xcodeImageView1 setFrame:CGRectMake(220.0f,
                                                  350.0f,
                                                  100.0f,
                                                  100.0f)];
        [self.xcodeImageView1 setAlpha:0.0f];

        [UIView commitAnimations];
    }
```

6、当图片视图中的任意一个停止时，我们需要将其从它的父视图中移除，因为它已经毫无用处了。就像我们在 startTopLeftImageViewAnimation 方法中看到的，我们需要给 UIView 类的类方法 setAnimationDidStopSelector:传递一个委托选择器，这个选择器会在图 1 的动画（就像我们看到的那样）以及图 2 的动画（我们即将看到的那样）结束时被调用。下面是这个委托选择器的代码实现：

```
    - (void)imageViewDidStop:(NSString *)paramAnimationID
                    finished:(NSNumber *)paramFinished
                     context:(void *)paramContext{

    UIImageView *contextImageView = (__bridge UIImageView *)paramContext;
        [contextImageView removeFromSuperview];
}
```

7、我们还需要一个方法来执行图 2 的动画。我们在写图 2 的动画方法时跟图 1 的方法比较会有些不同。我希望在图 1 的动画快要结束时再启动图 2 的动画。因此当图 1 在 3 秒的时间段内执行它的动画时，我希望图 2 快要到屏幕右下角并消失时开始启动它自己的动画。要实现这些想法，我们让两个动画在同一时间启动，但为图 2 设置 2 秒的延迟。这样一来假设当我们在下午 1 点整同时启动两个动画，图 1 会在 13:00:00 开始执行动画并在 13:00:03 结束，而图 2 会在 13:00:02 启动它的动画并在 13:00:05 结束。下面是我们如何执行图 2 的动画的：

```
    - (void) startBottomRightViewAnimationAfterDelay:(CGFloat)paramDelay{

        /* Start from bottom right corner */
        [self.xcodeImageView2 setFrame:CGRectMake(220.0f,
                                                  350.0f,
                                                  100.0f,
                                                  100.0f)];

        [self.xcodeImageView2 setAlpha:1.0f];
        [UIView beginAnimations:@"xcodeImageView2Animation"
        context:(__bridge void *)self.xcodeImageView2];

        /* 3 seconds animation */
        [UIView setAnimationDuration:3.0f];

        [UIView setAnimationDelay:paramDelay];
```

```
    /* Receive animation delegates */
    [UIView setAnimationDelegate:self];

    [UIView setAnimationDidStopSelector:
    @selector(imageViewDidStop:finished:context:)];

    /* End at the top left corner */
    [self.xcodeImageView2 setFrame:CGRectMake(0.0f,
                                              0.0f,
                                              100.0f,
                                              100.0f)];
    [self.xcodeImageView2 setAlpha:0.0f];

    [UIView commitAnimations];
}
```

8、最后，我们要在我们的视图变的可见时同时触发 startTopLeftImageViewAnimation 与 startBottomRightViewAnimationAfterDelay:方法：

```
- (void) viewDidAppear:(BOOL)paramAnimated{

    [super viewDidAppear:paramAnimated];
    [self startTopLeftImageViewAnimation];
    [self startBottomRightViewAnimationAfterDelay:2.0f];
}
```

15.13.4. 参考

无

15.14. 动画之视图的缩放

15.14.1. 问题

如果你想要放大或者缩小你的视图

15.14.2. 方案

为你的视图创建一个仿射缩放变换并使用 UIView 的动画方法来执行缩放变换。

15.14.3. 讨论

烈建议你在进行本节内容前先去看下 15.13 小节的内容。

要在执行时缩放一个视图，你可以在一个动画块中向视图提交一个缩放变换（参见 15.11），或者仅仅改变视图的宽和（或）高。

让我们来看一下如何通过向视图提交缩放变换来缩放图像视图：

```
- (void) viewDidAppear:(BOOL)paramAnimated{
```

```
    [super viewDidAppear:paramAnimated];

    /* Place the image view at the center of the view of this view controller */
    self.xcodeImageView.center = self.view.center;

    /* Make sure no translation is applied to this image view */
    self.xcodeImageView.transform = CGAffineTransformIdentity;

    /* Begin the animation */
    [UIView beginAnimations:nil
                    context:NULL];

    /* Make the animation 5 seconds long */
    [UIView setAnimationDuration:5.0f];

    /* Make the image view twice as large in
      width and height */
    self.xcodeImageView.transform = CGAffineTransformMakeScale(2.0f,
                                                               2.0f);

    /* Commit the animation */
    [UIView commitAnimations];
}
```

上面的代码使用了一个仿射缩放变换来缩放这个图片视图，让其变成原来的两倍大小。将缩放变换提交给一个视图的最大的好处是缩放的宽和高使用的是视图的中心来作为它缩放的中心。假设你视图在屏幕上的中心坐标是(100, 100)，将视图（宽，高都）放大为原来的两倍。放大后的视图中心还是在屏幕的(100, 100)坐标处，只是各个方向的宽度变为原先的两倍。但如果你通过显式的增加视图边框的宽高来来放大视图，可能最后的结果是你的视图会位于屏幕的另外一个位置。因为通过改变图片视图的边框来缩放视图时，无论你愿意与否，你同时也改变了边框的 x 跟 y 值。正因如此，你的图形视图不会从它的中心进行放大。

这个问题的解决方法不在本书的讨论范围内，但你可以自由地作一些尝试，或许你会找到解决方案。这里我可以给你个提示，你可以同时并行的运行两个动画：一个用来改变宽高，另外一个则用来改变图形视图的中心位置。

15.14.4. 参考

无

15.15. 动画之视图的旋转

15.15.1. 问题

如果你想要旋转你的视图。

15.15.2. 方案

创建一个旋转仿射变换并使用 UIView 类的动画方法来执行旋转动作。

15.15.3. 讨论

我强烈建议你在进行本节内容前先去看下 15.13 小节的内容。

为了在执行时旋转一个视图，你必须在一个动画块中向视图提交一个旋转变换（参见 15.11）。让我们来看些例子能让我们对此做一个更清晰的了解。那么我们看到我们有一张叫做 Xcode.png 的图片（参见 15-14 小节），并且我们希望在屏幕的正中间显示它。当图片显示出来后，我们希望它在 5 秒内旋转 90 度，然后再转会它原来的样子。因此当我们的视图显示在屏幕上后，让我们来将它顺时针旋转 90 度吧：

```
- (void) viewDidAppear:(BOOL)paramAnimated{
    [super viewDidAppear:paramAnimated];

    self.xcodeImageView.center = self.view.center;

    /* Begin the animation */
    [UIView beginAnimations:@"clockwiseAnimation"
                    context:NULL];

    /* Make the animation 5 seconds long */
    [UIView setAnimationDuration:5.0f];
    814 | Chapter 15: Graphics and Animations

    [UIView setAnimationDelegate:self];

    [UIView setAnimationDidStopSelector:
    @selector(clockwiseRotationStopped:finished:context:)];

    /* Rotate the image view 90 degrees */
    self.xcodeImageView.transform =
    CGAffineTransformMakeRotation((90.0f * M_PI) / 180.0f);

    /* Commit the animation */
    [UIView commitAnimations];
}
```

当顺时针动画完成后我们选择调用 clockwiseRotationStopped:finished:context:选择器。在这个方法中，我们会将图片视图 5 秒内逆时针转回 0 度（就像最开始的时候的样子）：

```
- (void)clockwiseRotationStopped:(NSString *)paramAnimationID
                        finished:(NSNumber *)paramFinished
                         context:(void *)paramContext{

    [UIView beginAnimations:@"counterclockwiseAnimation"
                    context:NULL];

    /* 5 seconds long */
    [UIView setAnimationDuration:5.0f];

    /* Back to original rotation */
    self.xcodeImageView.transform = CGAffineTransformIdentity;

    [UIView commitAnimations];
```

```
}
```

就像你在 15.13 和 15.14 小节中看到的，有几种途径能让视图动起来（直接或者间接继承 UIView），在执行你的动画时有多个属性可以改变。要让自己富有创造力起来并去查看一下 UIView 类的其它你之前不知道的一些属性。或者你还可能还想在 Xcode 的 Organizer 中去查看一下 UIView 的文档资料。

15.15.4. 参考

无

点击这里访问：DevDiv.com 移动开发论坛