目　　录

本书的目标读者

本书面向新的 iPhone 开发人员，他们马上要开发实际项目，但面对着一个全新的 SDK。虽然每位程序员创建表格的目标和经验并不相同，但大多数人都需要在其开发工作中解决类似的任务：如何构建表格，如何创建安全的密钥链条目，如何搜索地址簿，如何在视图之间切换，以及如何使用 Core Location。

本书针对的正好是刚开始接触 iPhone 编程的学习者。通过清晰、内容完备的示例，读者可以迅速起步并开始高效率的开发。书中给出了经过测试的、现成可用的解决方案，程序员可以专注于应用程序的具体部分，而不必为样本式的任务劳神。

本书的组织结构

本书针对 iPhone 开发新手面对的大多数常见问题逐一提供了解决方案：布置界面元素，响应用户操作，访问本地数据源并连接到因特网。本书按示例进行组织，因此代码拿来就可以使用，非常方便。程序员可以在自己的项目中使用书中的源代码，然后根据需要进行定制。每章都将相关的任务归类在一起。读者可以直接到所需的问题类别查找解决方案，而不需要总去琢磨哪些类或框架最适合当前问题。

下面概述了本书中各章的内容。

第1章：iPhone SDK 简介

第1章介绍 iPhone SDK 并将 iPhone 作为交付平台进行研究，包括平台的限制等。它对标准 iPhone 应用程序进行了细化的分类，并指导你构建第一个 Hello World 样式的示例。

第2章：视图

第2章介绍屏幕上的 iPhone 视图及对象。你将了解如何对视图进行布置、创建和排序，以创建 iPhone 应用程序的骨架。还会了解视图层次结构、几何方法和动画，以及用户如何通过触摸与视图进行交互。

第3章：视图控制器

iPhone 范型具体而言就是：小屏幕和大虚拟世界。在第3章中，你将探索各种 UIView- Controller

类，使用这些类可以对用户所交互的虚拟空间进行扩大和排序。你将学习如何在 iPhone 应用程序屏幕之间导航时让这些强大的对象执行所有繁重的任务。

第4章：警告用户

iPhone 为用户提供了多种警告方式，从弹出对话框和进度栏到音频提示和状态栏更新。第4章展示如何将这些指示功能构建到应用程序中，并扩展用户警告词汇库。

第5章：基本表格

表格提供了在小型受限设备上获得出色运行效果的交互类。iPhone 和 iPod touch 随带的许多乃至大部分应用程序都以表格为中心，包括 Settings、YouTube、Stocks 和 Weather。第5章展示 iPhone 表格的工作方式，哪些表格对于开发人员可用，以及如何在自己的程序中使用表格特性。

第6章：高级表格

iPhone 表格并不仅限于简单的滚动列表。你可以构建分为多个部分（各部分都有自己的标题）、带多个滚动栏的表格。你可以添加开关之类的控件，创建半透明单元背景，以及添加自定义字体。第6章以第5章的内容为基础，介绍可在 iPhone 程序中使用的高级表格秘诀。

第7章：媒体

不负众望，iPhone 可以加载并显示各种格式的媒体。它可以播放音乐和电影，处理图像和 Web 页面。也可以呈现 PDF 文档和相册。第7章介绍用多种方法将数据导入或下载到程序中，并使用 iPhone 的多点触摸界面显示这些数据。

第8章：控件

UIControl 类为许多 iPhone 交互式元素提供了基础，包括按钮、文本字段、滑块和开关。第8章通过已经或尚未用文档细致记录的 SDK 调用来介绍控件及其用法。

第9章：人物、地点和事件

除了在任意计算机上都可看到的标准用户界面控件和媒体组件，iPhone SDK 还提供了大量特定于 iPhone 和 iPod touch 交付的专门的开发人员解决方案。第9章介绍其中最有用的解决方案，包括地址簿访问（人物）、Core Location（地点）和传感器（事件）。

第10章：连接服务

作为一种可与因特网连接的设备，iPhone 非常适用于订阅基于 Web 的服务。苹果公司通过其在各类网络计算服务方面的坚实基础以及支持技术丰富了这种平台。iPhone SDK 可处理套接字、密码密钥链、SQL 访问、XML 处理等。第10章将探讨常用的网络计算技术，并提供能简化日常工作的方法。

第11章：Cover Flow 编程

虽然 Cover Flow 并未正式包含在 iPhone SDK 中，但它仍然提供了 iPhone 体验中最优秀的特性之一。使用 Cover Flow，能为用户提供极为出色的视觉选择体验，这一点令标准的滚动列表望尘莫及。第11章介绍 Cover Flow 并展示如何在应用程序中使用它。

4.1 通过警告直接与用户对话

可以通过 UIActionSheet 和 UIAlertView 对象与用户对话。它们通过弹出或在其他视图上方滚动来发送消息。这些轻量级类向应用程序中添加双向对话。警告直观地与用户“对话”并且可以提示用户回答。应用程序在屏幕上显示警告，获取用户确认，然后关闭警告继续进行其他任务。

如果你认为警告只不过是附带 OK 按钮的消息（如图4-1所示），那么这样的结论值得反思。UIAlertSheet 对象提供了丰富的功能（假设苹果公司继续让你访问这些丰富的功能）。借助警告表，你可以实际构建菜单、文本输入、查询以及更多内容。遗憾的是，此行为多半都被归类为文档中未记录或几乎未记录的类别。在本章的秘诀中，你将了解如何创建可以在自己程序中使用的各种实用的警告。

本章中涉及的大部分功能已从正式的 SDK 中删除，但在公共框架中仍然存在。由于本章更多地依赖于“非正式”调用，因此应该找到可靠性和功能之间的最佳平衡点。苹果公司不鼓励开发人员使用私有例程，因为开发人员会随意更改这些例程。但其本质与 Mac OS X 相同。如果你知道如何在公共架构中使用私有例程的方法，则可以使用它们。在应用程序中访问私有框架则是错误的做法。

说明　在早期版本的 iPhone 固件中，UIActionSheet 和 UIAlertView 实际上由相同的 UIAlertSheet 类实现。该类不仅提供弹出警告，而且还提供菜单功能。然后苹果公司将警告表替换为 UIModalView，并对来自该基类的这些新对象进行子类化。之后，苹

果公司删除了 UIModalView，在新版本的 SDK 中，UIActionSheet 和 UIAlertView 不再由此类派生。（它们都来自于 UIView。）与其前身一样，它们在行为方面仍然属于同类，并且使用相似的底层技术在屏幕上呈现自己。

4.1.1 记录结果

由于 printf 非常简单，因此大部分秘诀都使用 printf 以可以查看的格式输出它们的结果。在最初的 SDK Beta 版本期间，printf 提供了检查内部程序数据的最可靠的方法。现在，在最新的 iPhone SDK 版本中，NSLog()得到了完全实现，并且非常稳定。此外，下面给出了几个实用的技巧。

1. 重定向 stderr

标准的 freopen()函数将 stderr 输出重定向到你选择的日志文件，为其指定一个本地路径。此处，logPath 是一个标准的 NSString（如@"/tmp/mylog"），它已被转换为本地文件系统表示。完成此调用之后，所有 stderr 结果（包括 NSLog）将直接进入你的日志文件。当你的程序运行时，使用"w"（写）而不是使用"a"（附加）来重新启动日志：

```
freopen([logPath fileSystemRepresentation], "a", stderr);
```

2. 构建自定义日志函数

构建你自己的日志函数，可以添加除标准 NSLog 之外的功能。下面是模拟 NSLog 功能的一个基本日志函数。通过此函数，可以方便地生成警告通知或文本视图更新，而不必输出到 stderr：

```
#include <stdarg.h>
void dolog(id formatstring,...)
{
    va_list arglist;
    if (formatstring)
    {
      va_start(arglist, formatstring);
      id outstring = [[NSString alloc] initWithFormat:formatstring
      ➥ arguments:arglist];
      fprintf(stderr, "%s\n", [outstring UTF8String]);
      [outstring release];
      va_end(arglist);
    }
      }
```

4.1.2 构建警告

若要创建警告表，请分配一个 UIAlertView 对象。通过初始化为其分配一个标题、一个按钮数组和若干其他选项。该标题是一个 NSString，按钮数组包含多个 NSString，其中每个字符串表示一个按钮。代码清单4-1创建了一个非常简单的警告，如图4-1所示。

使用警告时，通常空间非常珍贵。将所有按钮放在一行上可以最大程度地减小警告高度，并且这也是默认表示。当定义更多按钮时，它们会一个挨一个地添加到该行中。因此，任何时候都要将添加的按钮数量限制为不超过3个或4个。按钮越少，效果越好，最好只有一两个。

UIAlertView 对象没有提供可视的“默认”按钮突出显示。在 iPhone 上下文中，按钮突出显示不但不能提供帮助，而且还容易造成混淆。（例如，用户应该按浅色按钮还是深色按钮？）没有办法使界面符合用户期望，因为此表示不明确。因此，与 OS X 不同的是，苹果公司在较新的 iPhone 固件中删除了突出显示。

说明　　Cancel 按钮出现在菜单的底部，并且位于弹出警告的左侧。

代码清单4-1　创建基本警告

```
UIAlertView *baseAlert = [[UIAlertView alloc]
                          initWithTitle:@"Alert" message:@""
                          delegate:self cancelButtonTitle:nil
                          otherButtonTitles:@"OK", nil];
```

图4-1　使用 UIAlertView 类构建简单的弹出警告

除非万不得已，否则应将委托设置为 UIViewController 主对象。委托实现了 UIAlertView-Delegate 协议。UIAlertView 实例至少需要此委托支持才能处理按钮轻击。

通过委托方法，你可以自定义当按下不同的按钮时的响应。实际上，如果仅需要通过一个 OK 按钮抛出一些消息，则可以忽略此委托支持。

用户看到警告并与之交互之后，他们会发起以下委托方法调用：alertView: clickedButtonAtIndex:。通过此调用，你可以确定按下了哪个按钮并且恢复在此表上设置的任何选项。按钮编号从零开始。

4.1.3 显示警告

使用 show 方法通知警告显示在屏幕上（即[myAlert show]）。显示时，警告采用模态方式工作。即，它使其后的屏幕暗淡并阻止用户与应用程序交互。此模式交互将一直继续，直到用户按钮轻击（通常是选择 OK 或 Cancel）确认该警告为止。

说明　创建警告表之后，你可以通过更新其 message 来自定义警告。该可选文本位于警告标题下方，警告按钮下方。其他可自定义的属性包括 title、delegate 和 visible。

4.2 秘诀：创建多行按钮显示

默认情况下，iPhone 在一行上显示其按钮。图4-2在左侧显示了这个标准的外观。你可以使用文档中未记录的调用覆盖一行显示。此调用创建一个多行警告，如右侧所示。使用 setNumberOfRows:构建单行按钮警告，如代码清单4-2所示。其中提供一个参数和一个整数，用于指定所请求的行数。

图4-2　通过文档中未记录的 UIKit 调用，你可以选择是将按钮放置在一行还是一列

说明　　有关在程序中使用文档中未记录的调用和特性的详细讨论，请参见第1章。

早期版本 iPhone 软件的工作方式恰好相反。最初的标准采用多行模式，并且必须使用 setNumberOfRows:让电话的按钮移动到单行上。

秘诀4-1创建的示例在屏幕中央弹出其警告，其周围的屏幕将变暗。由于默认情况下它采用模态方式运行，因此在用户轻击 OK 按钮之前，任何事情都处于等待状态。轻击之后，警告关闭，然后将控制传递给 modalView: clickedButtonAtIndex:委托方法。该方法正如此处所定义的，除了打印轻击按钮的编号之外，什么也没有做。

说明　　你可以使用 setDimsBackground:NO 禁止背景变暗淡。这是另一个文档中未记录的调用。我建议不要将它用作常规用途。它违背了苹果公司的人机界面标准，并且可能使用户弄不清楚是否可以在警告生存期内轻击背景。

秘诀4-1　创建带单行按钮的 UIAlertView

```
- (void) presentSheet
{
```

```
    UIAlertView *baseAlert = [[UIAlertView alloc]
                              initWithTitle:@"Alert"
                              message:@"Please select a button"
                              delegate:self cancelButtonTitle:nil
                              otherButtonTitles:@"One", @"Two", @"Three", nil];
    [baseAlert setNumberOfRows:3];
    [baseAlert show];
}

- (void)alertView:(UIAlertView *)alertView
➥clickedButtonAtIndex:(NSInteger)buttonIndex
{
    printf("User Pressed Button %d\n", buttonIndex + 1);
    [alertView release];
}
```

4.3　秘诀：自动计时的无按钮警告

无按钮警告提出了一个特殊的挑战，原因是它们无法正确地回调委托方法。它们无法自动关闭，即使在轻击时也是如此。图4-3显示了一个简单的无按钮警告。从此屏幕快照中可以看出，删除按钮会使警告看起来比较笨拙。你可以通过调整消息文本来改进此显示。添加回车（@"\n"），通过顶部的空间来平衡底部（正常情况下是按钮所在的位置）。

图4-3　使用文档中未记录的 dismiss 方法在显示一个无按钮的 UIAlertView 之后，将控制返回你的程序。在 message 属性中添加回车有助于平衡底部因缺少按钮而多出来的空间

由于警告以模态方式运行，因此你需要通过一项安全措施来确保警告在特定时间点消失，并且用户可以继续使用他们的 iPhone。一个简单的 NSTimer 可在一段时间（必须将控制返回到标准 GUI 的时间段）之后关闭警告。秘诀4-2创建了一个警告并使用计时器将其关闭，要么使用文档中未记录的 dismiss 方法，要么使用 UIAlertView 类的

dismissWithClickedButtonIndex: animated:方法。

秘诀4-2　创建带计时器自动防故障装置的无按钮 UIAlertView

```
- (void) performDismiss: (NSTimer *)timer
{
    [baseAlert dismissWithClickedButtonIndex:0 animated:NO];
}

- (void) presentSheet
{
    baseAlert = [[UIAlertView alloc]
                            initWithTitle:@"Alert"
                            message:@"Message to user with asynchronous
                            information"
                            delegate:self cancelButtonTitle:nil
                            otherButtonTitles: nil];
    [NSTimer scheduledTimerWithTimeInterval:3.0f target:self selector:
    ➥@selector(performDismiss:)
                                    userInfo:nil repeats:NO];
    [baseAlert show];
}
```

4.4　秘诀：请求用户的文本输入

暂且将正式 SDK 调用放在一边，警告视图还提供了一个非常简单的方法，用于提示用户输入文本。UIAlertView 类提供显示和关闭关联键盘的全部命令。你只需要添加文本输入字段，告诉警告如何处理它们，以及让它注意其他事项。

若要添加文本字段，请使用文档中未记录的 addTextFieldWithValue: label:调用。将默认文本作为第一个参数，将 otherwiseempty 字段中显示的文本作为第二个参数传递给该调用。图4-4显示了已填充字段和空字段。

图4-4　细致地管理空间并忽略标题和正文文本，你一次最多可向

UIAlertSheet 中添加3个文本输入字段。你可能希望将 UIAlertSheets

限制为1个或2个文本字段

你可以通过调用[alert textFieldAtIndex:0]，然后从带 text 属性的文本字段获取文本来恢复每个字段。由于每个字段都是可寻址的，因此你可以在显示警告之前设置其属性。自定义自动标题、自动纠正以及首选的键盘，如秘诀4-3所示。

空间非常重要。如果要细致地管理空间，你可以将警告放在屏幕上，同时留出足够的空间用于键盘显示。通过跳过标题和正文文本，你可以一次在屏幕上放置3个（而不是4个）文本输入字段。两个字段是符合实际的最大值，如图4-4所示。

秘诀4-3　使用 UIAlertView 请求用户的文本

```
- (void)alertView:(UIAlertView *)alertView
clickedButtonAtIndex:(NSInteger)buttonIndex
{
    printf("User Pressed Button %d\n", buttonIndex + 1);
    printf("Text Field 1: %s\n", [[[modalView textFieldAtIndex:0] text]
    ➥cStringUsingEncoding:NSUTF8StringEncoding]);
    printf("Text Field 2: %s\n", [[[modalView textFieldAtIndex:1] text]
   ➥cStringUsingEncoding:1]);
    [alertView release];
}

- (void) presentSheet
{
    UIAlertView *alert = [[UIAlertView alloc]
                          initWithTitle: @"Enter Information"
                          message:@"Specify the Name and URL"
                          delegate:self
                          cancelButtonTitle:@"Cancel"
                          otherButtonTitles:@"OK", nil];
    [alert addTextFieldWithValue:@"" label:@"Enter Name"];
    [alert addTextFieldWithValue:@"http://" label:@"Enter URL"];

    // Name field
    UITextField *tf = [alert textFieldAtIndex:0];
    tf.clearButtonMode = UITextFieldViewModeWhileEditing;
    tf.keyboardType = UIKeyboardTypeAlphabet;
    tf.keyboardAppearance = UIKeyboardAppearanceAlert;
    tf.autocapitalizationType = UITextAutocapitalizationTypeWords;
    tf.autocorrectionType = UITextAutocorrectionTypeNo;

    // URL field
    tf = [alert textFieldAtIndex:1];
    tf.clearButtonMode = UITextFieldViewModeWhileEditing;
    tf.keyboardType = UIKeyboardTypeURL;
    tf.keyboardAppearance = UIKeyboardAppearanceAlert;
    tf.autocapitalizationType = UITextAutocapitalizationTypeNone;
    tf.autocorrectionType = UITextAutocorrectionTypeNo;

    [alert show];
}
```

4.5　秘诀：显示简单菜单

图 4-5 使用 showInView:创建简单的菜单演示。菜单滑动到视图的底部。尽管此处 Delete File 菜单按钮显示为灰色，但它在 iPhone 上显示为红色，表示永久的操作，这些操作可能对用户造成不利影响

谈到菜单，UIActionSheet 实例可满足 iPhone 的需要。它们将选择对象（主要是表示可能操作的按钮列表）滑动到屏幕上并且等待用户响应。操作页不同于弹出菜单。弹出菜单与界面相分离，并且比较适合用于请求注意。菜单滑入视图中，并且可以更好地与当前应用程序工作集成。Cocoa Touch 提供了两种显示菜单的方法。

l showInView。在视图中显示表是使用菜单的理想方式，此处使用的就是这个方法。该方法将视图底部的菜单向上滑动（参见图4-5）。

l showFromToolBar:和 showFromTabBar。在使用工具栏、选项卡栏或提供水平分组按钮的任何其他栏（在很多应用程序的底部你可以看到这些栏）时，这些方法将该菜单与栏顶部对齐并且完全滑出它原来所在的位置。

秘诀4-4显示了初始化和显示简单的 UIAction- Sheet 实例的方法。它的初始化方法引入了一个 UIAlertView:破坏性按钮中缺少的概念。红色的破坏性按钮表示没有返回的操作，如永久删除某个文件（参见图4-5）。明亮的红色向用户发出关于这个选择的警告。很明显，该选项应该谨慎使用。

说明 操作表缺少的就是 UIAlertView 的消息（至少对于正式 SDK 是如此）。除了显示标

题之外，操作表实例未正式支持添加消息。实际上，消息虽然得到了实现，但效果却不是非常理想。使用文档中未记录的 setMessage:调用向菜单中添加消息。使用时，消息显示在表标题下方，文本稍微有点大。

秘诀4-4　显示菜单

```
-(void)actionSheet:(UIActionSheet *)actionSheet
clickedButtonAtIndex:(NSInteger)buttonIndex
{
    printf("User Pressed Button %d\n", buttonIndex + 1);
    [actionSheet release];
}

- (void) presentSheet
{
    UIActionSheet *menu = [[UIActionSheet alloc]
                           initWithTitle: @"File Management"
                           delegate:self
                           cancelButtonTitle:@"Cancel"
                           destructiveButtonTitle:@"Delete File"
                           otherButtonTitles:@"Rename File", @"Email File", nil];
    [menu showInView:contentView];
}
```

4.6　“请稍候”：向用户显示进度

等待是计算体验中固有的一部分，在可预见的将来仍然如此。作为一名开发人员，将这个事实告诉用户是你的工作。Cocoa Touch 提供了许多类，用于通知用户等待进程完成。这些进度指示器采用两种形式：进度轮，它在其演示期间一直存在；进度条，在整个进程中，它从左侧向右侧填充。提供这些指示的类如下所示。

l　UIActivityIndicatorView。该进度指示器提供一个旋转的圆，通知用户等待，而不提供有关其完成程度的具体信息。iPhone 活动指示器非常小，但它的活动动画会吸引用户的眼球并且最适合用于正常应用程序中的快速中断。

l　UIProgressView。该视图显示一个进度条。该条提供有关已经完成的工作量以及剩余但仍然占用相对较小数量的屏幕空间的工作量的具体反馈。它显示为一个狭长、水平的矩形，随着进度的前进它从左向右填充。这种经典的用户界面元素对于长时间延迟来说效果最佳，因为在这期间用户想知道作业已经完成了多少。

从文档中未记录的特性的角度来看，你拥有以下内容。

l　UIProgressHUD。文档中未记录的“小心显示”版本的进度指示器浮在窗口中的所有其他视图上方，并且向基本进度指示器添加一个状态消息。HUD 在进度条和简单指示器之间提供一个中间立场。你不需要采用进度条的方式量化你的进度，但可以利用关联文本描绘进度。当你想指出伴随某个进程所发生的情况时，进度 HUD 工作特别好（例如，“联系服务器”、“身份验证”、“请求数据”等）。

4.7　秘诀：调用基本的文档中未记录的 UIProgressHUD

UIProgressHUD 类使用一个状态对话框来覆盖窗口。该对话框包含一个旋转的进度指示器和一个可指定的简短消息。就编程而言，你只需要实例化对象，使用主窗口初始化它，并通过发送 show:YES 告诉它执行的操作。图4-6显示了一个实际运行的 UIProgressHUD 显示，秘诀4-5显示了创建该显示的代码。

通过 UIProgressHUD，你可以根据需要更新其文本。随时都可发送一个新的 setText:消息。完成后，使用 show:NO 关闭它。如果需要，你可以通过 setFontSize:调整字体大小，但是我建议你确保消息简短扼要。

图4-6　UIProgressHUD 的名称表示它是一个 HUD。它添加了一个覆盖在窗口之上的进度条。你告诉它什么时候显示，什么时候隐藏以及内容是什么

说明　令人失望的是，苹果公司并没有将它包含在标准 SDK 类中。这里，我已经提出了解决办法。到这本书出版时，我真心地希望苹果公司将它添加回 Cocoa Touch。

使用带数据密集型代码的 HUD 显示时，你会希望让它在其自己的线程中运行，以便 UI 更新不会阻止。为此，可以使用 NSThread 的 detatchThreadSelector: toTarget: withObject:方法。使用单独的线程使其启动，更新 HUD 的文本并在完成后关闭 HUD。

秘诀4-5　为窗口创建一个 UIProgressHUD

```
@interface UIProgressHUD : NSObject
- (void) show: (BOOL) yesOrNo;
- (UIProgressHUD *) initWithWindow: (UIView *) window;
- (void) setText: (NSString *) theText;
@end

- (void) killHUD: (id) aHUD
{
    [aHUD show:NO];
    [aHUD release];

}
- (void) presentSheet
{
    id HUD = [[UIProgressHUD alloc] initWithWindow:[contentView superview]];
    [HUD setText:@"Downloading File. Please wait."];
    [HUD show:YES];


    [self performSelector:@selector(killHUD:)
               withObject:HUD afterDelay:5.0];
}
```

4.8　秘诀：使用 UIActivityIndicatorView

UIActivityIndicatorView 实例提供轻型视图，这些视图显示一个标准的旋转进度轮。当使用这些视图时，最重要的一个关键词是小。20×20像素是大多数指示器样式获得最清楚显示效果的大小。只要稍大一点，指示器都会变得模糊。图4-7显示了一个40像素的版本。

你需要在屏幕上将该指示器居中。将其放置在最方便操作的位置。作为背面清晰的视图，指示器将混合位于其后的背景视图。该背景的主要颜色帮助选择要使用的指示器样式。

对于一般用途，只需将活动指示器作为子视图添加到你要覆盖的窗口、视图、工具栏或导航栏。分配该指示器并使用一个框来初始化它，最好位于当前所使用的父视图中央。

通过将指示器的 animating 属性更新为 YES 来启动它。若要停止，将该属性设置为 NO。Cocoa Touch 会负责完成其余工作，在视图不使用时隐藏视图。

iPhone 提供了几种不同样式的 UIActivityIndicatorView 类。UIActivityIndicator-ViewStyleWhite 和 UIActivityIndicatorViewStyleGray 是最简洁的。黑色背景下最适合白色版本的外观，白色背景最适合灰色外观（如图4-7所示）。它非常瘦小，而且采用夏普风格。选择白色还是灰色时要格外注意。全白显示在白色背景下将不能显示任何内容。而 UIActivityIndicatorViewStyleWhiteLarge 只能用于深色背景。它提供最大、最清晰的指示器。

秘诀4-6显示了创建这些简单的 UIActivityIndicatorView 实例的代码。

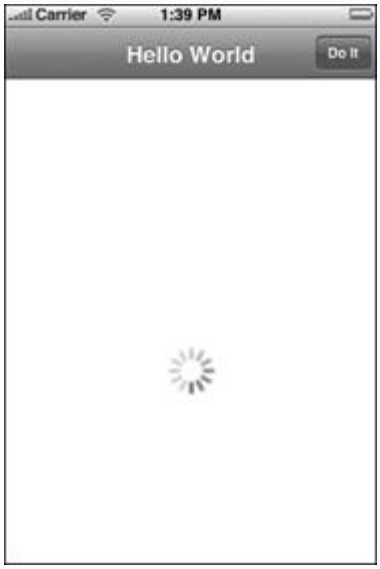

图4-7　UIActivityIndicatorView 类提供了一个简单的旋转轮，这意味着将以相对较小的尺寸显示

秘诀4-6　向程序中添加 UIActivityIndicatorView

```
- (void) performAction
{
        if (progressShowing) [activityIndicator stopAnimating];
           else [activityIndicator startAnimating];
        progressShowing = !progressShowing;
}
- (void)loadView
{
```

```
    contentView = [[UIView alloc] initWithFrame:[[UIScreen mainScreen]
    ➥applicationFrame]];
    self.view = contentView;
    contentView.backgroundColor = [UIColor whiteColor];
    [contentView release];
    // Add an action button
    self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
                                              initWithTitle:@"Do It"
                                              style:UIBarButtonItemStylePlain
                                              target:self
                                              action:@selector(performAction)]
                                              autorelease];

    // Add the progress indicator but do not start it
    progressShowing = NO;
    activityIndicator = [[UIActivityIndicatorView alloc]
    ➥initWithFrame:CGRectMake(0.0f, 0.0f, 32.0f, 32.0f)];
    [activityIndicator setCenter:CGPointMake(160.0f, 208.0f)];
    [activityIndicator
    ➥setActivityIndicatorViewStyle:UIActivityIndicatorViewStyleGray];
    [contentView addSubview:activityIndicator];
    [activityIndicator release];
}
```

4.9　秘诀：构建 UIProgressView

进度视图可以让用户在执行任务时了解任务进度，而不仅仅是告诉用户“请稍候”。会显示一条从左向右填充的进度条，表示任务完成的程度。进度条最适合长时间的等待，它提供状态反馈，从而使用户能够保持控制感。

若要创建进度视图，请分配该视图并设置它的框。若要使用进度条，请调用 setProgress:。它只带一个参数，一个范围介于0.0（无进度）和1.0（已完成）之间的浮点数字。进度视图条有两种样式：基本白色或浅灰色。可以通过 setStyle:方法选择你喜欢的种类。

与其他种类的进度指示器不同的是，显示和隐藏进度条的视图将完全由你来决定。它没有 setVisible:方法。我喜欢向操作表中添加进度条。这样既可以将它们带到屏幕上，又能关闭它们。另一个好处是当显示警告表时，屏幕的其他部分会变暗。这样便可在你的任务进行时强制采用模式显示。用户无法与 GUI 交互，直到关闭警告为止。秘诀4-7显示了一个 UIActionSheet/　UIProgressView 示例，该示例生成的显示如图4-8所示。

图4-8　使用 UIProgressView 实例跟踪长时间延迟的进度。将它们添加到一个 UIActionSheet 简化它们的显示和关闭

该秘诀使用文档中未记录的 setNumberOfRows:调用。若要停留在 SDK 标准之内，就向标题中添加几个新行（\n）。

说明　　在操作表中嵌入进度条时，你需要实现 UIActionSheet 委托协议。添加一个按钮单击方法符合该协议所需的方法，即使该方法本身未执行任何操作。

秘诀4-7　向警告表中添加进度条

```
@interface UIActionSheet (extended)
- (void) setNumberOfRows: (NSInteger) rows;
@end

@implementation HelloController

// This callback fakes progress via setProgress:
- (void) incrementBar: (id) timer
{
    amountDone += 1.0f;
    [progbar setProgress: (amountDone / 20.0)];
    if (amountDone > 20.0) {[baseSheet dismissWithClickedButtonIndex:0
    ➥animated:YES]; [timer invalidate];}
}

// Dismiss the action sheet backdrop used here
```

```
- (void)actionSheet:(UIActionSheet *)actionSheet
➥clickedButtonAtIndex:(NSInteger)buttonIndex
{
    [actionSheet release];
}
// Load the progress bar onto an actionsheet backing
- (void) presentSheet
{
if (!baseSheet) {
        baseSheet = [[UIActionSheet alloc]
                     initWithTitle:@"Please Wait"
                     delegate:self
                     cancelButtonTitle:nil
                     destructiveButtonTitle: nil
                     otherButtonTitles: nil];
        [baseSheet setNumberOfRows:5];
        [baseSheet setMessage:@"Updating Internal Databases"];

        progbar = [[UIProgressView alloc] initWithFrame:CGRectMake(50.0f, 70.0f,
        ➥220.0f, 90.0f)];
        [progbar setProgressViewStyle: UIProgressViewStyleDefault];

        [baseSheet addSubview:progbar];
        [progbar release];
    }

    // Create the demonstration updates
    [progbar setProgress:(amountDone = 0.0f)];
    [NSTimer scheduledTimerWithTimeInterval: 0.5 target: self selector:
➥@selector(incrementBar:) userInfo: nil repeats: YES];
    [baseSheet showInView:contentView];
}

- (void)loadView
{
    contentView = [[UIView alloc] initWithFrame:[[UIScreen mainScreen]
    ➥applicationFrame]];
    self.view = contentView;
    contentView.backgroundColor = [UIColor whiteColor];
    [contentView release];

    // Add an action button
    self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
                                               initWithTitle:@"Do It"
                                               style:UIBarButtonItemStylePlain
                                               target:self
                                               action:@selector(presentSheet)]
                                               autorelease];
}
```

.10　秘诀：添加自定义、可轻击的覆盖层

图 4-9　结合透明度和动画，创建可弹出的视图覆盖层

当与用户交流时，你需要限制使用标准的（甚至是文档中未记录的）UIKit 对象。通过在视图上添加覆盖层，你可以创造无限的可能。你可以抛出文本、图像、动画以及你所拥有的任何内容。可以让覆盖层具有交互性（如此秘诀中所示），或者允许通过轻击进入下层界面。可以添加简单的状态信息（如透明背景上白色、灰色或黑色的文本），或者可以构建详细的显示。这些选择由你自己决定。

图4-9显示了在秘诀4-8中构建的自定义“请稍候”覆盖层。添加 UIView 动画，如第2章中所述，使覆盖层的入口变得平滑，并退出显示。通过在覆盖层上轻击来关闭它，从而返回到下层界面。可以看到，该覆盖层非常繁忙，并且其作用主要是提供一个干净、明亮的背景。若要使用这种类型的覆盖层以及复杂的界面，你可能需要用半透明的白色来填充透明区域，这样会部分隐藏界面并添加一个仍然能看见任何消息的背景。

如果你想添加一个类似于此的覆盖层并提供与 UIProgressHUD 相似的背景，应尽量与苹果公司的整体图形设计一致。在自定义界面元素中，用户对错误尤为敏感。

秘诀4-8　创建自定义动画 UIView 覆盖层

```
@interface PleaseWaitView : UIImageView
@end

@implementation PleaseWaitView
- (PleaseWaitView *) initWithFrame: (CGRect) rect
{
    self = [super initWithFrame:rect];
    [self setImage:[UIImage imageNamed:@"PlsWait.png"]];

    UIImageView *imgView = [[UIImageView alloc] initWithFrame:CGRectMake(40.0f,
    ➥300.0f, 60.0f, 60.0f)];
    [self addSubview:imgView];
    [imgView release];

    // load in the animation cells for the butterfly

    NSMutableArray *bflies = [[NSMutableArray alloc] init];
    for (int i = 1; i <= 17; i++) {
        NSString *cname = [NSString stringWithFormat:@"bf_%d.png", i];
        UIImage *img = [UIImage imageNamed:cname];

        if (img) [bflies addObject:img];
    }

    // begin the animation
    [imgView setAnimationImages:bflies];
    imgView.animationDuration = 0.75f;
    [imgView startAnimating];
    [bflies release];

    return self;
}

- (void)touchesBegan: (NSSet *) touches withEvent:(UIEvent *)event
{
    // fade away the overlay
    CGContextRef context = UIGraphicsGetCurrentContext();
    [UIView beginAnimations:nil context:context];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
    [UIView setAnimationDuration:1.0];

    [self setAlpha:0.0f];

    // Complete the animation
    [UIView commitAnimations];
    [pleaseWait setUserInteractionEnabled:YES];
}

@end

@interface HelloController : UIViewController
{
    UIView *contentView;
    PleaseWaitView *pleaseWait;
}
@end

@implementation HelloController

CGPoint randomPoint() { return CGPointMake(random() % 256, random() % 256); }

- (void) moveButterfly: (NSTimer *) timer
```

```
{
    if ([pleaseWait alpha] == 0.0f) {[timer invalidate]; return;}

    // Create an animation block around moving the butterfly to a new origin
    CGContextRef context = UIGraphicsGetCurrentContext();
    [UIView beginAnimations:nil context:context];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
    [UIView setAnimationDuration:1.0];

    // Move the butterfly
    UIImageView *iv = [[pleaseWait subviews] objectAtIndex:0];
    CGRect frame = [iv frame];
    frame.origin = randomPoint();
    [iv setFrame:frame];

    // Complete the animation
    [UIView commitAnimations];
}

- (void) presentSheet
{
    if ([pleaseWait alpha] != 0.0f) return;

    // Create an animation block around fading in the overlay
    CGContextRef context = UIGraphicsGetCurrentContext();
    [UIView beginAnimations:nil context:context];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
    [UIView setAnimationDuration:1.0];

    [pleaseWait setAlpha:1.0f];

    // Complete the animation
    [UIView commitAnimations];

    // Give life to the butterfly
    [NSTimer scheduledTimerWithTimeInterval: 4.0f target: self selector:
    ➥@selector(moveButterfly:)
                                    userInfo: nil repeats: YES];

}

- (id) init
{
    if (self = [super init]) self.title = @"Hello World";
    return self;
}

- (void)loadView
{
    contentView = [[UIView alloc] initWithFrame:[[UIScreen mainScreen]
    ➥applicationFrame]];
    self.view = contentView;
    contentView.backgroundColor = [UIColor whiteColor];
    [contentView release]; // reduce retain count by one

    // Add an action button
    self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
```

```
                                        initWithTitle:@"Do It"
                                        style:UIBarButtonItemStylePlain
                                        target:self
                                        action:@selector(presentSheet)]
                                        autorelease];

    // Create and hide the "Please Wait" view
    pleaseWait = [[PleaseWaitView alloc] initWithFrame:[[UIScreen mainScreen]
    ➥applicationFrame]];
    [contentView addSubview:pleaseWait];
    [pleaseWait setAlpha:0.0f];
}

-(void) dealloc
{
    [pleaseWait release];
    [contentView release];
    [super dealloc];
}
@end
```

4.11 秘诀：构建下滑式警告

在构建之前介绍的动画块时，你可以采用相同的方法来创建一个比较传统的下滑式警告。秘诀4-9让某视图框的位置有了动画效果，从而使视图看上去具有滚动的屏幕。该框刚开始不在屏幕中，然后以动画的方式显示在屏幕上，如图4-10所示。当用户单击 OK 时，进程反向。你可以使用该方法构建任何种类的警告。

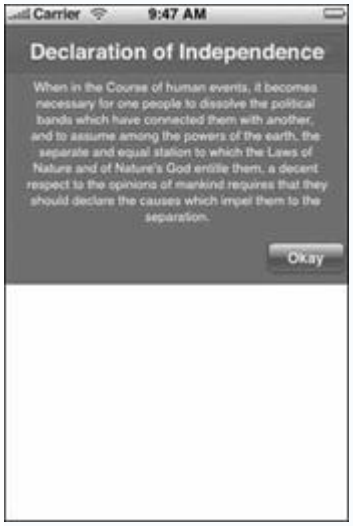

图4-10 使用 UIViewAnimation 创建自定义下滑式警告表

该秘诀中的视图包含一个标题、一个文本块以及一个简单的按钮。向你自己的实现中添加任何所需的 UIView 元素或控件。第5章中的一个示例就采用了本秘诀的方法，它在向下滑动的视图中添加用户可以选择的滚动表。

秘诀4-9　构建自定义下滑式警告

```
// Create a color of blue that mimics the official gray highlighting
UIColor *sysBlueColor(float percent) {
    float red = percent * 255.0f;
    float green = (red + 20.0f) / 255.0f;
    float blue = (red + 45.0f) / 255.0f;
    if (green > 1.0) green = 1.0f;
    if (blue > 1.0f) blue = 1.0f;

    return [UIColor colorWithRed:percent green:green blue:blue alpha:1.0f];
}
@interface TopAlert : UIView
{
    UILabel *title, *message;
}
- (void) setTitle: (NSString *)titleText;
- (void) setMessage: (NSString *)messageText;
@end

@implementation TopAlert

- (void) setTitle: (NSString *)titleText
{

    [title setText:titleText];
}
- (void) setMessage: (NSString *)messageText
{
    [message setText:messageText];
}

- (TopAlert *) initWithFrame: (CGRect) rect
{
    rect.origin.y = 20.0f - rect.size.height; // Place above status bar
    self = [super initWithFrame:rect];

    [self setAlpha:0.9];
    [self setBackgroundColor: sysBlueColor(0.4f)];

    // Add button
    UIButton *button = [[UIButton buttonWithType:UIButtonTypeCustom]
    ➥initWithFrame:CGRectMake(220.0f, 200.0f, 80.0f, 32.0f)];
    [button setBackgroundImage:[UIImage imageNamed:@"whiteButton.png"]
    ➥forState:UIControlStateNormal];
    [button setTitle:@"Okay" forState: UIControlStateHighlighted];
```

```
    [button setTitle:@"Okay" forState: UIControlStateNormal];
    [button setFont:[UIFont boldSystemFontOfSize:14.0f]];
    [button addTarget:self action:@selector(removeView)
    ➥forControlEvents:UIControlEventTouchUpInside];
    [self addSubview:button];

    // Add title
    title = [[UILabel alloc] initWithFrame:CGRectMake(0.0f, 8.0f, 320.0f, 32.0f)];
    title.text = @"Declaration of Independence";
    title.textAlignment = UITextAlignmentCenter;
    title.textColor = [UIColor whiteColor];
    title.backgroundColor = [UIColor clearColor];
    // Alternative framed title:
    // title.backgroundColor = [UIColor colorWithRed:1.0f green:1.0f blue:1.0f
    ➥alpha:0.25f];
    title.font = [UIFont boldSystemFontOfSize:20.0f];
    [self addSubview:title];
    [title release];

    // Add message
    message = [[UILabel alloc] initWithFrame:CGRectMake(20.0f, 40.0f, 280.0f,
    ➥200.0f - 48.0f)];
    message.text = @"When in the Course of human events, it becomes necessary for
    ➥one people to dissolve the political bands which have connected them with
    ➥another, and to assume among the powers of the earth, the separate and equal

    ➥which the Laws of Nature and of Nature's God entitle them, a decent respect
    ➥to the opinions of mankind requires that they should declare the causes
    ➥which impel them to the separation.";
    message.textAlignment = UITextAlignmentCenter;
    message.numberOfLines = 999;
    message.textColor = [UIColor whiteColor];
    message.backgroundColor = [UIColor clearColor];
    message.lineBreakMode = UILineBreakModeWordWrap;
    message.font = [UIFont systemFontOfSize:[UIFont smallSystemFontSize]];
    [self addSubview:message];
    [message release];

    return self;
}

- (void) removeView
{
    // Scroll away the overlay
    CGContextRef context = UIGraphicsGetCurrentContext();
    [UIView beginAnimations:nil context:context];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
    [UIView setAnimationDuration:0.5];

    CGRect rect = [self frame];
    rect.origin.y = -10.0f - rect.size.height;
```

```
    [self setFrame:rect];

    // Complete the animation
    [UIView commitAnimations];
}

- (void) presentView
{
    // Scroll in the overlay
    CGContextRef context = UIGraphicsGetCurrentContext();
    [UIView beginAnimations:nil context:context];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
    [UIView setAnimationDuration:0.5];

    CGRect rect = [self frame];
    rect.origin.y = 0.0f;
    [self setFrame:rect];

    // Complete the animation
    [UIView commitAnimations];
}
-(void) dealloc
{
    [title release];
    [message release];
    [super dealloc];
}
@end
```

4.12　秘诀：添加状态栏图像

位于 iPhone 顶部的栏提供了丰富的状态信息。这些应用程序触发的状态符号位于时间显示的右侧。比较常用的图标包括暂停/播放指示器、电池使用等。UIApplication 实例可以使用文档中未记录的方法向状态栏中添加现有图像，但是我无法让状态栏调用使用来自应用程序包自身的数据。所有状态栏图像都存储在 SpringBoard.app 中的 /System/Library/CoreServices/SpringBoard.app 下。因此不难发现，这非常符合 App Store 协议的边界条件。

此限制有两个意义深远的结果。首先，你能使用的图像非常有限（它们只是标准的 PNG，使用透明度）；其次，需要向用户的 System 文件夹中添加自定义图像，这会带来安全性问题并影响操作系统的整体性能。这种方式不好，而且不安全，并且开发人员和苹果公司之间签有协议和沙盒。

秘诀4-10显示了如何加载和删除这些图像。若要设置图标，请调用 addStatusBar-ImageNamed:，并传递一个参数（不带扩展的图标名）。作为一项规则，每个图像有两个版

本，一个用于黑色的状态栏（每个名称都以 FSO_开头），一个用于白色/灰色状态栏（名称以 Default_开头）。所有更改在退出程序之后都会持续存在。这也表明你应该谨慎地使用状态栏更新。

图 4-11　使用状态栏图标，可以与用户交流模式信息

例如，在传递时，@"Airplane"（飞机模式图标）、SpringBoard 显示 Default_Airplane.png（针对黑色状态栏）。你没有传递扩展名或前缀。使用 removeStatusBarImageNamed:删除图标。图4-11显示了飞机图标，如图所示。

在应用程序结束之后，状态栏图像会仍然存在。在状态栏中添加一个飞机，然后退出，飞机仍然存在。如果你决定使用状态栏图像，则考虑在挂起或终止程序之前清除状态栏。按照定义，这些图像的目的是指出超过任何应用程序会话范围的行为。由于苹果公司坚决要求“一次一个应用程序”，因此在该限制之内应尽量少用这种技术。

秘诀4-10　添加和删除状态栏图标

```
@interface UIApplication (extended)
- (void) addStatusBarImageNamed:(NSString *)aName;
- (void) removeStatusBarImageNamed:(NSString *)aName;
@end

- (void) performAction
{
    isShowing = !isShowing;

    if (isShowing)
        [[UIApplication sharedApplication] addStatusBarImageNamed:@"Airplane"];
    else
        [[UIApplication sharedApplication] removeStatusBarImageNamed:@"Airplane"];
}
```

4.13　添加应用程序标记

如果曾经使用过 iPhone 或 iPod touch，你可能在主屏幕的应用程序上面看到过红色的小标记（badge）。这可能表示自从用户上次打开电话或电子邮件以来累计的未接电话数量或未读的电子邮件数量。

实际上，可采用3种方法来标记应用程序：一种是非常简单的 UIApplication 调用；另一种有

点复杂，需要在 UIKit 中建立通道；最后一种是使用从 Web 服务器发布的 SDK push 功能。若要在应用程序内部设置标记，则可以使用文档中未记录的 setApplicationBadge:调用或设置正式的 applicationIconBadgeNumber 属性。将 NSString 作为常规标记参数传递给该调用，将字符串大小限制为4或5（最多）个字符，或将标记数量属性限制为 NSInteger。例如，你可以用当前月份的3个字母缩写来标记应用程序，如代码清单4-2所示。

说明　　若要隐藏标记，将 applicationIconBadgeNumber 设置为0（数字零），或将 setApplicationBadge:设置为@""（空字符串）。

代码清单4-2　使用当前月份标记应用程序

```
NSDate *now = [NSDate dateWithTimeIntervalSinceNow:0];
NSString *caldate = [[now
       dateWithCalendarFormat:@"%b"
       timeZone:nil] description];
[[UIApplication sharedApplication] setApplicationBadge:caldate];
```

若要删除应用程序标记，请传递空字符串（如@""）。这将从图标中删除任何现有标志。如果想获得一个“空”标记，则传递一个空格字符串@""。

UIApplication 方法的问题在于，若要使用它，必须将你的请求直接放置在应用程序下，然后只能以数值表示。你可能希望标记图标来自应用程序外部。这是开发人员将希望了解的事情，尤其是你不打算通过应用程序存储分发某些应用程序时。代码清单4-3演示了如何使用公用 UIKit 框架的动态链接来标记应用程序。该代码不要求应用程序自身完成标记设置。而是调用 SpringBoard 来执行琐碎的工作。

此函数依赖动态链接。对 UIKit 框架进行反向工程展示了 UIApplication 的 setApplicationBadge 的工作原理。它调用具有一个字符串和一个应用程序标识符的 SBSetApplicationBadge。通常，在日常编程中，我不赞成使用动态链接方法。图4-12显示了使用该功能标记几个图标的结果。

图4-12　应用程序标记显示为位于应用程序图标右上角的红色小容器

注意，最新的 iPhone 固件版本行为与标记不一致。第二方标志继续可靠地为/Applications 文件夹中的项工作，但不能为位于沙盒中的那些应用程序可靠地工作。

说明　　就 App Store 提交而言，仍然不确定 SpringBoard 调用是属于“文档中未记录但可以接受”类别还是属于“文档中未记录且不受支持”类别。应谨慎使用动态链接。

代码清单4-3　通过反向工程和动态链接实现常规标记

```
#include <dlfcn.h>
void badge(char *appid, char *badge)
{
   // link to UIKit
   void *uikit = dlopen("/System/Library/Frameworks/UIKit.framework/UIKit",
                 RTLD_LAZY);

   // Recover the SpringBoard Port
   int (*SBSSpringBoardServerPort)() =
                 dlsym(uikit, "SBSSpringBoardServerPort");
   mach_port_t *p;
   p = SBSSpringBoardServerPort();

   // Perform the badging
   int (*doBadge)(mach_port_t* port, char* x, char *y) =
                 dlsym(uikit, "SBSetApplicationBadge");
   doBadge(p, appid, badge);
   dlclose(uikit);
}
```

4.14　秘诀：简单的音频警告

音频警告直接对用户“说话”。它们产生即时反馈（假设用户的听力没有问题）。幸运的是，苹果公司通过系统音频服务将基本声音播放构建到 Cocoa Touch SDK 中。它的工作方式与 Macintosh 上的系统音频非常相似。或者也可以使用 Audio Queue 调用来创建音频警告，但

价格非常高昂。Audio Queue 播放对于程序来说“非常昂贵”，它涉及的内容比所需的简单的警告声音更加复杂。相比之下，你可以下载系统音频，然后仅使用几行代码来播放它。

苹果公司表示，让警告声音保持简短，两秒或更少时间最合适。虽是这么说，但我使用长度高达一分钟的声音进行了测试，它们使用系统音频服务播放起来非常不错。系统音频只播放 PCM 和 IMA 音频。这意味着将你的声音限制为 AIFF、WAV 和 CAF 格式。

若要构建系统声音，请通过一个指向声音文件的文件 URL 来调用 AudioServicesCreate-SystemSoundID。该调用返回一个初始化的系统声音对象，然后你可以随意播放该对象。只需借助声音对象来调用 AudioServicesPlaySystemSound。此单一调用完成了所有工作。

你可以添加一个可选的系统声音完成回调，在声音完成播放（调用 AudioServicesAdd-SystemSoundCompletion）之后通知程序，但实际上，你真的不需要这样做，尤其在使用苹果公司认可的简短声音时更是如此。

若要清除声音，请在使用它们之后，抛弃它们。可以使用正在讨论的声音对象调用 AudioServicesDisposeSystemSoundID。

说明　若要使用这些系统声音服务，请确保在代码中包含 Audio Toolbox/AudioServices.h，并且链接到音频工具箱框架。

震动

与音频警告相同，震动会立即引起用户的注意。而且，震动几乎适合所有用户，包括听力或视力受损的用户。使用秘诀4-11中的系统音频服务，你可以在播放声音的同时震动。你只需要通过下面这行调用来实现它：

```
AudioServicesPlaySystemSound (kSystemSoundID_Vibrate);
```

你无法改变震动参数。每个调用都会产生一个简短的一到两秒的嗡嗡声。在不支持震动的平台上（如 iPod touch），该调用不执行任何操作，但也不会产生错误。

秘诀4-11　播放简单的音频警告

```
@interface HelloController : UIViewController
{
    UIView *contentView;
    SystemSoundID pmph;
}
@end

@implementation HelloController

- (void) playSound
{
    AudioServicesPlaySystemSound (pmph);
}

- (void)loadView
{
    contentView = [[UIView alloc] initWithFrame:[[UIScreen mainScreen]
    ➥applicationFrame]];
    self.view = contentView;
    contentView.backgroundColor = [UIColor whiteColor];
    [contentView release];

    // Add an action button
    self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
                                               initWithTitle:@"Do It"
                                               style:UIBarButtonItemStylePlain
                                               target:self
                                               action:@selector(playSound)]
                                               autorelease];

    // Load the sound
    id sndpath = [[NSBundle mainBundle] pathForResource:@"pmph1" ofType:@"wav"
    ➥inDirectory:@"/"];
    CFURLRef baseURL = (CFURLRef)[[NSURL alloc] initFileURLWithPath:sndpath];
    AudioServicesCreateSystemSoundID (baseURL, &pmph);
}

-(void) dealloc
{
    if (pmph) AudioServicesDisposeSystemSoundID(pmph);
    [contentView release];
    [super dealloc];
}
@end
```

4.15 小结

本章介绍了直接与用户交互的方式。你已经学习了如何构建警告（视觉、听觉和触觉）以引起用户的注意，并且可以请求立即反馈。使用这些示例增强程序的交互性，并利用仅 iPhone 支持的独特特性。下面给出了本章的一些要点。

l 本章中的很多秘诀都依赖于文档中未记录的 UIKit 调用或动态链接（较差的方式）。我已经将这些秘诀包含在一本书中，该书侧重于正式的 SDK 开发，因为它们的功能不仅实用，而且经过了大部分 iPhone 开发人员的测试。本章中的所有调用都仅限于公用的 SDK 框架，

并且没有违背苹果公司所声明的策略（至少理论上是这样）。

l　无论何时何任务占用大量时间时，一定要对用户有礼貌并且显示一种进度反馈。iPhone 提供了许多可完成此操作的方法，例如弹出显示和状态栏指示器。

l　警告立刻引起用户注意。它们的作用是在交流信息时引起响应。而且，正如本章所述，它们具有极佳的可定制性。可以围绕简单的 UIAlertSheet 构建整个应用程序。

l　标记和状态栏图标都允许将应用程序状态扩展到当前应用程序之外。保守地使用这些特性可以提供最大量的信息。

l　音频反馈就像单击和嘟嘟声一样，可以改进程序并丰富交互性。使用系统声音调用意味着使用 iPod 功能可以出色地播放声音，不会破坏当前的听觉体验。

IControl 类为 iPhone 的许多交互式元素，包括按钮、文本字段、滑块和开关，提供了基础。与它们的祖先类相比，这些控件有更多共同点。控件都使用相似的布局和目标操作（target-action）方法。本章介绍控件及其用法，探索如何采用多种方式构建和定制控件。你将了解已在文档中细致记录的 SDK 调用和未在文档中细致记录的 SDK 调用。你还会学习如何通过分解 UIControl 视图结构（hierarchy）来访问和修改控件元素。从简单到复杂，本章介绍了大量可以在程序中重新使用的控件秘诀。

8.1　秘诀：构建简单的按钮

iPhone 支持若干种按钮类型。两个最常使用的按钮是从头开始创建的独立的 UIButton 实例，和专门在栏（包括导航栏、标签栏和工具栏）上使用的 UIBarButtonItem 实例。

UIBarButtonItem 提供了一个极其简单的方法，用最少的代码向程序添加按钮操作功能。代码清单8-1显示了一个典型的调用，你可以在 UIViewController loadView 方法中找到该调用。使用一个标题、一种样式、一个目标和一个操作来创建按钮。父视图，不管是导航栏、标签栏还是工具栏，都会为你处理按钮的位置和大小问题。

代码清单8-1　向导航栏添加 UIBarButtonItem

```
// Add a randomize button
UIBarButtonItem *randomButton = [[[UIBarButtonItem alloc]
                                 initWithTitle:@"Randomize"
                                 style:UIBarButtonItemStylePlain
                                 target:self
                                 action:@selector(randomize)] autorelease];
self.navigationItem.rightBarButtonItem = randomButton;
```

UIBarButtonItem 使用目标操作。与所有控件相同，当用户与它们进行交互时，它们允许添加至少一个目标和回调操作。栏按钮条目（bar button item）不指定相关事件的类型，它唯一关注的事件是按钮点击。

使用栏按钮条目的缺点是不能把它们随意地放置在屏幕上。它们不是视图。因此，不能设置它们的结构并把它们添加到其他视图中。拥有它们的栏会构建和管理它们的外观。要把按钮放置在任意视图中，需要使用 UIButton 类。

说明　按钮由其父视图构建完成之后，就可以访问栏按钮条目的真实视图了。调用文档中未记录的 UIButton view 方法。在没有遇到极其复杂问题的情况下，不能将该视图从其父视图中移除并把它看成一个独立的对象。而应使用视图添加像（不受文档支持）UIToolbarButtonBadge 类这样的子视图，如本章后面的内容所示。

8.1.1　UIButton 类

UIButton 实例向 iPhone 应用程序添加交互式按钮。它们提供更传统的控件，包括用于目标操作调用的边框（frame）和事件规范。点击这些按钮时，它们向指定的目标发送消息。IPhone 提供了构建 UIButton 的两种方法。你可以使用一个预置按钮类型或从头开始构建一个自定义按钮。

当前的 iPhone SDK 提供了下面这些预置按钮类型。和你看到的一样，按钮类型并不是十分普遍。它们被添加到 SDK 主要是为了方便苹果公司，而不是方便你。虽然如此，你也可以在需要时在程序中使用它们。

l　Detail Disclosure。当你向表单元格添加细节展开附件时，会看到圆形的、蓝色的带 V 形的圆圈。

l Info Light 和 Info Dark。这两个按钮提供了一个小的圆形的 i，和你在 Macintosh 的 Dashboard 小配件中看到的一样。这些按钮在 Weather 和 Stocks 应用程序中使用，用来将视图从一边翻到另一边。

l Contact Add。这个圆形的蓝色圆圈中心有一个白色的“+”，在联系人应用程序中向地址簿添加新联系人时可以看到它。

l Rounded Rectangle。这个按钮提供了一个简单的环绕按钮文本的圆角矩形。它并不是特别有魅力的按钮（就是说，它并不十分像“苹果风格”的外观），不过在应用程序中编写和使用它非常简单。

要使用预置按钮，需要分配它、设置其边框并添加一个目标。不要为添加自定义样式或创建按钮的整体外观而担心，因为 SDK 负责所有这些工作。代码清单8-2展示了如何构建一个简单的圆角矩形按钮。要构建其他标准按钮类型，请忽略标题行。圆角矩形是唯一使用标题的预置按钮类型。

说明 使用 UIControlEventTouchUpInside 事件获取用户对按钮的点击。

代码清单8-2 构建“预置”圆角矩形按钮

```
UIButton *button = [UIButton buttonWithType:UIButtonTypeRoundedRect];
[button setFrame:CGRectMake(0.0f, 0.0f, 80.0f, 30.0f)];
[button setCenter:CGPointMake(160.0f, 208.0f)];
[button setTitle:@"Beep" forState:UIControlStateNormal];
[button addTarget:self action:@selector(playSound:)
➥forControlEvents:UIControlEventTouchUpInside];
[contentView addSubview:button];
```

8.1.2 构建自定义按钮

使用 UIButtonTypeCustom 类型时，你需要提供所有的按钮样式。图像的数量取决于你希望按钮如何工作。对于一个简单的按钮，可以添加一个背景图像，并在按钮被按下的时候把标签颜色改为突出显示。对于一个开关类型的按钮，需要使用4个图像：正常显示下的“关”状态，突出显示下的“关”状态（即按钮按下时），还有“开”状态对应的两个图像。由你自己来选择和设计交互细节。

秘诀8-1构建了一个切换开关按钮，展示了有关构建自定义按钮的详细信息。点击按钮时，按钮会把它的样式从绿色切换到红色或者从红色切换到绿色。这样，你的用户（非色盲）能立刻识别当前状态。图8-1（左边）显示了使用此秘诀创建的按钮。

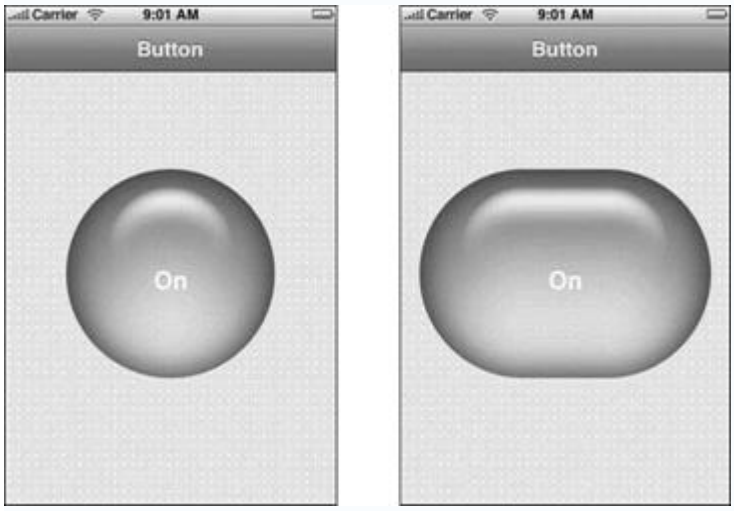

图8-1　使用 UIImage 拉伸来调整任意按钮宽度。设置左边罩（cap）的宽度以便指定从哪个位置进行拉伸

此秘诀中的 UIImage 可拉伸图像调用在按钮创建过程中起着重要作用。通过将圆形样式转换为菱形按钮，可拉伸图像使你能够创建任意宽度的按钮。你将指定任意一个边缘的罩（cap）（即不应该被拉伸的部分）。在本例中，罩为110像素宽。如果你打算将此秘诀中使用的220像素按钮宽度改为300，就在罩结束的那一点中间的位置拉伸按钮，如图8-1所示（右边）。

秘诀8-1　构建切换开关的 UIButton

```
@interface HelloController : UIViewController
{
    UIImageView            *contentView;
    BOOL                isOn;
}
@end
```

```
@implementation HelloController
- (UIButton *) buildButton: (NSString *) aTitle
{
    // Create a button sized to our art
    UIButton *button = [[UIButton alloc] initWithFrame:CGRectMake(0.0f, 0.0f,
    ➥220.0f, 233.0f)];

    // The cap width indicates the location where the stretch occurs
    [button setBackgroundImage:[[UIImage imageNamed:@"green.png"]
    ➥stretchableImageWithLeftCapWidth:110.0 topCapHeight:0.0]
    ➥forState:UIControlStateNormal];
    [button setBackgroundImage:[[UIImage imageNamed:@"green2.png"]
    ➥stretchableImageWithLeftCapWidth:110.0 topCapHeight:0.0]
    ➥forState:UIControlStateHighlighted];

    // Set up the button aligment properties
    button.contentVerticalAlignment = UIControlContentVerticalAlignmentCenter;
    button.contentHorizontalAlignment = UIControlContentHorizontalAlignmentCenter;

    // Set the title, font and color. The color switches from
    // white to gray for highlights
    [button setTitle:aTitle forState:UIControlStateNormal];
    [button setTitle:aTitle forState:UIControlStateHighlighted];

    [button setTitleColor:[UIColor whiteColor] forState:UIControlStateNormal];
    [button setTitleColor:[UIColor lightGrayColor]
    ➥forState:UIControlStateHighlighted];

    [button setFont:[UIFont boldSystemFontOfSize:24.0f]];

    isOn = NO;
    return [button autorelease];
}

- (void) toggleButton: (UIButton *) button
{
    // Swap the art when the state changes
    if (isOn = !isOn)
    {
        [button setTitle:@"On" forState:UIControlStateNormal];
        [button setTitle:@"On" forState:UIControlStateHighlighted];
        [button setBackgroundImage:[[UIImage imageNamed:@"green.png"]
        ➥stretchableImageWithLeftCapWidth:110.0 topCapHeight:0.0]
        ➥forState:UIControlStateNormal];
        [button setBackgroundImage:[[UIImage imageNamed:@"green2.png"]
        ➥stretchableImageWithLeftCapWidth:110.0 topCapHeight:0.0]
        ➥forState:UIControlStateHighlighted];
    }
    else
    {
        [button setTitle:@"Off" forState:UIControlStateNormal];
        [button setTitle:@"Off" forState:UIControlStateHighlighted];
        [button setBackgroundImage:[[UIImage imageNamed:@"red.png"]
        ➥stretchableImageWithLeftCapWidth:110.0 topCapHeight:0.0]
```

```
        ➥forState:UIControlStateNormal];
        [button setBackgroundImage:[[UIImage imageNamed:@"red2.png"]
        ➥stretchableImageWithLeftCapWidth:110.0 topCapHeight:0.0]
        ➥forState:UIControlStateHighlighted];
    }
}
- (void)loadView
{
    // Load an application image and set it as the primary view
    contentView = [[UIImageView alloc] initWithFrame:[[UIScreen mainScreen]
    ➥applicationFrame]];
    [contentView setImage:[UIImage imageNamed:@"bluedots.png"]];
    [contentView setUserInteractionEnabled:YES];
    self.view = contentView;
    [contentView release];

    // Add the button
    UIButton *button = [self buildButton:@"On"];
    [button setCenter:CGPointMake(160.0f, 200.0f)];
    [contentView addSubview: button];
    [button addTarget:self action:@selector(toggleButton:)
    ➥forControlEvents: UIControlEventTouchUpInside];

    isOn = YES;
}
-(void) dealloc
{

    [contentView release];
    [super dealloc];
}
@end
```

8.1.3 玻璃按钮（glass button）

在 SDK Beta 阶段，苹果公司取消了对其简单 UIGlassButton 类的官方支持。这个类提供了比圆角矩形更漂亮的外观，并支持对控件进行着色。在编写本书时，UIGlassButton 类依然存在于模拟器的 UIKit 框架中，不过苹果公司并没有在 iPhone 的 UIKit 框架中完全实现它

8.2 秘诀：向按钮添加动画元素

你不能向按钮添加子视图，不过可以创造性地在按钮的前面或后面放置样式。使用标准 UIView 结构来达到此目的，要确保对任何视图禁用用户交互，否则会让按钮变得模糊难懂（setUserInteractionEnabled:NO）。图 8-2显示了当你把半透明按钮样式和它后面的动画 UIImage 结合起来后会发生什么。图像视图内容“泄漏”给观察者，使你能够向按钮添加动画元素。

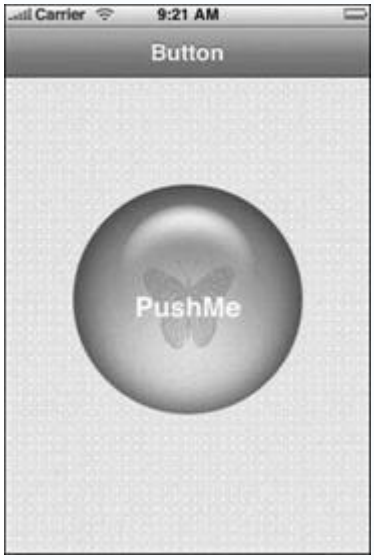

图8-2　将半透明的按钮样式和动画 UIImageViews 结合起来，以构建引人注目的 UI 元素。在秘诀8-2中，蝴蝶在按钮“内部”拍动翅膀

秘诀8-2　在 UIButton 后面添加动画元素

```
- (void)loadView
{
    // Load an application image and set it as the primary view
    contentView = [[UIImageView alloc] initWithFrame:[[UIScreen mainScreen]
    ➥applicationFrame]];
    [contentView setImage:[UIImage imageNamed:@"bluedots.png"]];
    [contentView setUserInteractionEnabled:YES];
    self.view = contentView;
    [contentView release];

    // load in the animation cells for the butterfly
    NSMutableArray *bflies = [[NSMutableArray alloc] init];
    for (int i = 1; i <= 17; i++) {
        NSString *cname = [NSString stringWithFormat:@"bf_%d.png", i];
        UIImage *img = [UIImage imageNamed:cname];
        if (img) [bflies addObject:img];
    }

    // create the image view and add it

    UIImageView *imageView = [[UIImageView alloc] initWithFrame:CGRectMake(0.0f,
    ➥0.0f, 80.0f, 80.0f)];
    [imageView setAnimationImages:bflies];
    [imageView setAnimationDuration:1.2f];
    [imageView startAnimating];
    [bflies release];
```

```
    [contentView addSubview:imageView];
    [imageView release];
    [imageView setCenter:CGPointMake(160.0f, 200.0f)];
    [imageView setUserInteractionEnabled:NO];

    // Add the button and set it to play the sound
    UIButton *button = [self buildButton:@"PushMe"];
    [button setCenter:CGPointMake(160.0f, 200.0f)];
    [contentView addSubview: button];
    [button addTarget:self action:@selector(playSound:) forControlEvents:
    ➥UIControlEventTouchDown];
}
```

8.3 秘诀：为按钮响应制作动画效果

UIControl 除了边框和目标操作外还有更多内容。所有的控件都继承自 UIView 类。这就意味着当你处理控件时，可以像处理标准视图时一样使用 UIView 动画块。秘诀8-3构建了一个切换开关，它使用 UIViewAnimationTransitionFlipFromLeft 旋转按钮，同时更改状态。

与秘诀8-1不同，此代码不转换样式，而是转换按钮。共有两个按钮：一个开按钮和一个关按钮，它们都基于清晰的 UIView。通过为这两个按钮提供一个透明的父视图，你能够仅为这些按钮应用翻转，而不包含用户界面的其余元素。忽略清晰的背景，并最终旋转整个窗口——这并不是好的 UI 选择。

由于此秘诀使用了和上一个秘诀相同的半透明样式，因此任何时候在屏幕上只出现一个按钮是很重要的。要达到此目的，在动画块（animation block）中时，当前按钮应该把自己从该清晰的超视图中移除。相反状态的按钮会取代它的位置。图8-3显示了翻转过程中的翻转按钮。

图8-3　使用 UIView 动画块在控件状态之间翻转。这里，按钮正在旋转，在关状态与开状态之间进行切换

秘诀8-3　将 UIView 动画块添加到控件

```
- (void) toggleButton: (UIButton *) button
{
    [UIView beginAnimations:nil context:NULL];

    [UIView setAnimationDuration:1.0f];
    [UIView setAnimationTransition:
     UIViewAnimationTransitionFlipFromLeft forView:clearView cache:YES];

    // swap out the button views during the flip
    [button removeFromSuperview];
    if (isOn = !isOn)[clearView addSubview:onButton];
    else [clearView addSubview:offButton];

    [UIView commitAnimations];
}
```

8.4　秘诀：定制开关

重要的 UIControl 对象通常基于一系列子视图构建。通过导航某个控件的子视图树，你可以公开和定制通常不能从标准 SDK 中访问的对象。以开关为例，SDK 只提供一种类型的开关：一个开/关按钮。要编辑该按钮使其转换为如图8-4所示（左边）的 Yes/No 按钮，只需要编写几行代码。

图8-4　（左边）通过向下导航对象（dumpster diving），你能够公开和自定义底层 UISwitch 视图。（右边）通过访问警告的子视图来为 UIAlertView 按钮着色

这种定制依赖于对控件视图树的理解。代码清单8-3显示了如何探索该结构。向 explode: level: 发送一个控件对象，初始级别为0。该方法以递归的方式遍历树，依次呈现每个子视图。

代码清单8-3　探索 UIControl 视图结构

```
#define outstring(anObject) [[anObject description] UTF8String]

- (void) explode: (id) aView level: (int) aLevel
{
    // indent to show the current level
    for (int i = 0;  i < aLevel; i++) printf("—", aLevel);

    // show the class and superclass for the current object
    printf("%s:%s\n", outstring([aView class]), outstring([aView superclass]));

    // recurse for all subviews
    for (UIView *subview in [aView subviews])
        [self explode:subview level:(aLevel + 1)];
}
```

发送开关时，explode 会在 Xcode 控制台中创建以下结果。正如你所看到的，一个开关有一个子视图，即一个自定义 UISwitchSlider 对象。这里有4个子视图。3个图像用于创建背景和开关样式。一个视图拥有开关上使用的两个标签。因为这些标签是标准的 UILabel 实例，因此可以允许它们使用你提供的任何文本：

```
UISwitch:UIControl
 --_UISwitchSlider:UISlider
 ----UIImageView:UIView
 ----UIImageView:UIView
 ----UIView:UIResponder
 ------UILabel:UIView
 ------UILabel:UIView
 ----UIImageView:UIView
```

知道如何设置视图后，就能够构建一个访问这些条目的 UISwitch 子类。秘诀8-4显示了添加标签文本定制的 UICustomSwitch 类实现。

说明　秘诀8-4中扩展的 UISwitch 类定义还显示了文档中未记录的 setAlternateColors:调用，使你能够使用桔黄色背景而不是蓝色背景。有关在应用程序中使用文档中未记录的特性的讨论，请参见第1章。

秘诀8-4　向 iPhone 开关添加文本定制

```
// Add the undocumented alternate colors
@interface UISwitch (extended)
- (void) setAlternateColors:(BOOL) boolean;
@end

// Expose the _UISwitchSlider class
@interface _UISwitchSlider : UIView
@end

// UICustomSwitch allows you to set the left and right label text
@interface UICustomSwitch : UISwitch
- (void) setLeftLabelText: (NSString *) labelText;
- (void) setRightLabelText: (NSString *) labelText;
@end

@implementation UICustomSwitch
- (_UISwitchSlider *) slider {
    return [[self subviews] lastObject];
}
- (UIView *) textHolder {
    return [[[self slider] subviews] objectAtIndex:2];
}
- (UILabel *) leftLabel {
    return [[[self textHolder] subviews] objectAtIndex:0];
}
- (UILabel *) rightLabel {
    return [[[self textHolder] subviews] objectAtIndex:1];
}
- (void) setLeftLabelText: (NSString *) labelText {
    [[self leftLabel] setText:labelText];
}
- (void) setRightLabelText: (NSString *) labelText {
    [[self rightLabel] setText:labelText];
}
@end
```

8.4.1 定制 UIAlertView 按钮

通过访问子视图，你可以定制许多类。图8-4（右边）显示了一个带有自定义按钮的UIAlertView。被着色为红色时，它指示可以在 UIActionSheet 破坏性按钮中找到的相同类型的危险。这是危险信号。

着色过程是不一致的。因为 UIAlertView 是一个文档中未记录的按钮类，它不响应标准的 tintColor 调用。你必须为按钮设置背景颜色。这就意味着颜色会渗过按钮边缘。幸运的是，渗漏很轻微，只有挂饰（pendant）才需要避免此方法。这里的教训是使用非标准 SDK 类时要小心。它们中的许多类都可以使用 drawRect：调用来呈现，但不允许对自身的完全定制。

代码清单8-4使用了 willPresentAlertView:委托方法。此方法使警告能够在定制按钮之前完成自身的构建。警告视图和动作表（以及大多数栏类）不会完全地构建它们的组件视图，直到

它们出现在屏幕上之前。等待使得子视图能够在被修改之前和在屏幕上被呈现给用户之前完成对自身的构建。

代码清单8-4　为 UIActionSheet 按钮着色

```
- (void)willPresentAlertView:(UIAlertView *)alertView
{

    [[[alertView subviews] objectAtIndex:2] setBackgroundColor:[UIColor
    ➥colorWithRed:0.5 green:0.0f blue:0.0f alpha:1.0f]];
}
```

8.5　秘诀：添加自定义滑块缩略图

要保持修改控件元素的主题，需要知道许多控件都提供了公共定制。例如，通过 UISlider 类，能够更改其缩略图组件的外观，而不需要搜索整个子视图结构。通过调用 setThumbImage:forState:方法将此缩略图设置为你喜欢的任何图像。下一个秘诀将重点从搜索正确的视图转到了动态定制上。它将动态地构建滑块样式。

秘诀8-5构建了一个滑块，其缩略图可以动态地改变。在本例中，缩略图的亮度直接和当前滑块的值联系在一起。当用户操作滑块时，CreateImage 例程在运行时构建滑块。Core Graphics 调用构建图8-5所示的样例图像。内部圆形的亮度被设置为滑块的值。

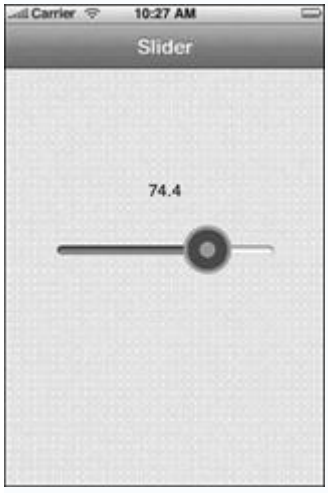

图8-5　Core Graphics 调用使此滑块的缩略图能根据当前滑块的值变暗或变亮

这种类型的反馈可以基于任何类型的数据。你可以从机载传感器中获取值或向因特网发出呼叫。不管你使用的是什么样的在线更新方案，动态更新显然是图形密集的，不过它不会太昂

贵。Core Graphics 调用的速度很快，缩略图对内存的需求也是最小的。秘诀8-5使用苹果公司的示例代码创建了可定制的位图上下文。然后，它根据传递给 createImage 函数的浮点值（percentage）的目标类型样式填充上下文。当用户与滑块进行交互时，此函数返回一个新的缩略图。

秘诀8-5　构建动态滑块缩略图

```
// MyCreateBitmapContext: Source based on Apple Sample Code
CGContextRef MyCreateBitmapContext (int pixelsWide,
                                    int pixelsHigh)
{

    CGContextRef    context = NULL;
    CGColorSpaceRef colorSpace;
    void *          bitmapData;
    int             bitmapByteCount;
    int             bitmapBytesPerRow;

    bitmapBytesPerRow   = (pixelsWide * 4);
    bitmapByteCount     = (bitmapBytesPerRow * pixelsHigh);

    colorSpace = CGColorSpaceCreateDeviceRGB();
    bitmapData = malloc( bitmapByteCount );
    if (bitmapData == NULL)
    {
        fprintf (stderr, "Memory not allocated!");
        return NULL;
    }
    context = CGBitmapContextCreate (bitmapData,
                                     pixelsWide,
                                     pixelsHigh,
                                     8,
                                     bitmapBytesPerRow,
                                     colorSpace,
                                     kCGImageAlphaPremultipliedLast);
    if (context== NULL)
    {
        free (bitmapData);
```

```
        fprintf (stderr, "Context not created!");
        return NULL;
    }
    CGColorSpaceRelease( colorSpace );

    return context;
}

// Build a thumb image based on the grayscale percentage
id createImage(float percentage)
{
    CGRect aRect = CGRectMake(0.0f, 0.0f, 48.0f, 48.0f);
    CGContextRef context = MyCreateBitmapContext(48, 48);
    CGContextClearRect(context, aRect);

    // Outer gray circle
    CGContextSetFillColorWithColor(context, [[UIColor lightGrayColor] CGColor]);
    CGContextFillEllipseInRect(context, aRect);

    // Inner circle with feedback levels
    CGContextSetFillColorWithColor(context, [[UIColor colorWithRed:percentage
    ➥green:0.0f blue:0.0f alpha:1.0f] CGColor]);
    CGContextFillEllipseInRect(context, CGRectInset(aRect, 4.0f, 4.0f));

    // Inner gray circle
    CGContextSetFillColorWithColor(context, [[UIColor lightGrayColor] CGColor]);
    CGContextFillEllipseInRect(context, CGRectInset(aRect, 16.0f, 16.0f));

    CGImageRef myRef = CGBitmapContextCreateImage (context);
    free(CGBitmapContextGetData(context));
    CGContextRelease(context);

    return [UIImage imageWithCGImage:myRef];
}

@interface HelloController : UIViewController
{
    UIImageView *contentView;
    UILabel *valueLabel;
    UIImage *knob;
}
@end

@implementation HelloController
- (void) updateValue: (UISlider *) slider
{
```

```
    [valueLabel setText:[NSString stringWithFormat:@"%3.1f", [slider value]]];

    CGImageRef oldknobref = [knob CGImage];
    UIImage *knob = createImage([slider value] / 100.0f);
    [slider setThumbImage:knob forState:UIControlStateNormal];
    [slider setThumbImage:knob forState:UIControlStateHighlighted];
    CFRelease(oldknobref);
}

- (void)loadView
{
    contentView = [[UIView alloc] initWithFrame:[[UIScreen mainScreen]
    ➥applicationFrame]];
    [contentView setImage:[UIImage imageNamed:@"bluedots.png"]];
    [contentView setUserInteractionEnabled:YES];
    self.view = contentView;
    [contentView release];

    // Add a text label to show the current slider scale
    valueLabel = [[UILabel alloc] initWithFrame:CGRectMake(50.0f, 100.0f, 220.0f,
    ➥40.0f)];
    valueLabel.textAlignment = UITextAlignmentCenter;
    valueLabel.text = @"50.0";
    valueLabel.backgroundColor = [UIColor clearColor];
    [contentView addSubview:valueLabel];
    [valueLabel release];

    // Build the slider

    UISlider *slider = [[UISlider alloc] initWithFrame:CGRectMake(50.0f, 160.0f,
    ➥220.0f, 40.0f)];
    slider.backgroundColor = [UIColor clearColor];
    slider.minimumValue = 0.0f;
    slider.maximumValue = 100.0f;
    slider.continuous = YES;
    slider.value = 50.0f;

    // Add the custom knob
    knob = createImage(0.5);
    [slider setThumbImage:knob forState:UIControlStateNormal];
    [slider setThumbImage:knob forState:UIControlStateHighlighted];

    [slider addTarget:self action:@selector(updateValue:)
    ➥forControlEvents:UIControlEventValueChanged];

    [contentView addSubview: slider];
    [slider release];
}
@end
```

8.5.1 向滑块添加文本

不幸的是，添加文本的代价相对来说比较高。可以向滑块添加文本，如代码清单8-5所示，不过对于普通的交互都需要付出大量成本。此代码清单展示了如何将标签放入到核心图形上下文中。最好预先计算这些小图像，并从那些预先计算的图像中加载它们，而不是立即生成它们。虽然此例程在 Macintosh 上模拟得很好，但对于处理那些文本处理给实时图像创建带来的额外负担来说，iPhone 的芯片有些功率不足。相比之下，预先计算的图像比较便宜，无

论是在计算方面还是在内存存储量方面。

记住，Quartz 坐标系从左下角开始，而不是左上角。将文本添加到图像中的一个点使用了 Quartz 系统。

代码清单8-5　添加略缩文本

```
id createImage(float percentage)
{
    CGRect aRect = CGRectMake(0.0f, 0.0f, 48.0f, 48.0f);
    CGContextRef context = MyCreateBitmapContext(48, 48);
    CGContextClearRect(context, aRect);

    CGContextSetFillColorWithColor(context, [[UIColor lightGrayColor] CGColor]);
    CGContextFillEllipseInRect(context, aRect);

    CGContextSetFillColorWithColor(context, [[UIColor colorWithRed:percentage
    ➥green:0.0f blue:0.0f alpha:1.0f] CGColor]);
    CGContextFillEllipseInRect(context, CGRectInset(aRect, 4.0f, 4.0f));

    CGContextSetFillColorWithColor(context, [[UIColor whiteColor] CGColor]);

    CGContextSelectFont(context, "Georgia", 14.0f, kCGEncodingMacRoman);
    CGContextSetTextDrawingMode(context, kCGTextFill);
    CGContextSetShouldAntialias(context, true);
    NSString *outString = [NSString stringWithFormat:@"%4.2f", percentage];
    CGContextShowTextAtPoint(context, 10, 20, [outString UTF8String], [outString
    ➥length]);

    CGImageRef myRef = CGBitmapContextCreateImage (context);
    free(CGBitmapContextGetData(context));
    CGContextRelease(context);

    return [UIImage imageWithCGImage:myRef];
}
```

8.6　秘诀：关闭 UITextField 键盘

有关 UITextField 控件最常问到的一个问题是“如何关闭键盘？”没有内置方法可以自动解决这个问题。当用户结束对 UITextField 内容的编辑时，键盘就应该自动消失。

所幸，响应编辑结束还需要一些工作。通过监视 Return 键，可以放弃首要响应者状态。这会把键盘移出视线之外，如秘诀8-6所示。下面是有关上述操作的一些要点。

图8-6　捕获按下 Return 键的动作，以了解你的用户什么时候结束编辑一个文本字段，这样你就可以关闭键盘

l　　将 Return 键类型设置为 UIReturnKeyDone。使用一个“Done”类型的 Return 键告知用户如何完成编辑。图8-6显示了一个使用 Done 键类型的键盘。

l　　实现 textFieldShouldReturn:。此方法捕获所有按下回车键的动作——不管它们是如何被命名的。使用此方法放弃首要响应者。这会隐藏键盘，直到用户触摸另一个文本字段或文本视图。

l　　确保你的视图控制器实现了 UITextFieldDelegate 协议，并确保将字段委托设置为指向控制器。

代码需要处理这些要点，以便为 UITextField 实例创建平滑的交互过程。图8-6实际地显示了 UITextField 控件，以及字段编辑过程中位于适当位置的键盘。

秘诀8-6　使用 Done 键关闭文本字段键盘

```
// This method catches the Return action
- (BOOL)textFieldShouldReturn:(UITextField *)textField
{
```

```
    [textField resignFirstResponder];
    return YES;
}
- (void)loadView
{
    contentView = [[UIView alloc] initWithFrame:[[UIScreen mainScreen]
    ➥applicationFrame]];
    contentView.backgroundColor = [UIColor whiteColor];
    self.view = contentView;
    [contentView release];

    // Create a field with a Done return key
    namefield = [[[UITextField alloc] initWithFrame:CGRectMake(120.0f, 40.0f,
    ➥150.0f, 30.0f)] retain];
    [namefield setBorderStyle:UITextBorderStyleRoundedRect];

    namefield.placeholder = @"name";
    namefield.returnKeyType = UIReturnKeyDone;
    namefield.clearButtonMode = UITextFieldViewModeWhileEditing;
    namefield.delegate = self;
    [contentView addSubview:namefield];
    [namefield release];
}
```

8.7　秘诀：关闭 UITextView 键盘

当关闭键盘时，UITextView 实例需要的方法和 UITextField 实例稍微有些不同。用户应该能够在文本视图中点击 Return，在不关闭键盘的情况下添加回车。当文本视图有效时，向普通界面添加一个 Done 按钮，如图8-7所示。当用户完成他或她的编辑时，可以使用这个键放弃首要响应者状态。

为了理解文本视图的活动，视图控制器必须实现 UITextViewDelegate 协议，它必须被设置为文本视图的委托。当用户点击视图时，就触发 textViewDidBeginEditing:委托方法。通过查明此问题，你能够添加或启用 Done 按钮。然后，用户可以在完成编辑之后点击 Done 按钮。Done 按钮为完成编辑和关闭键盘提供了一个明了的方法。

秘诀8-7展示了如何在委托方法调用中添加导航条目按钮，以及当用户完成编辑时如何移除它。当视图变为有效时就显示 Done 按钮，当放弃视图的首要响应者状态时就隐藏它。

图8-7　当用户开始与文本视图进行交互时，就向导航栏添加一个 Done 键。这为用户完成编辑和关闭键盘提供了一个明了的方法

秘诀8-7　向有效的 UITextView 会话添加一个 Done 按钮

```
@interface HelloController : UIViewController <UITextViewDelegate>
{
    UITextView *contentView;
}
@end

@implementation HelloController

// Reveal a done button when editing starts
- (void) textViewDidBeginEditing: (UITextView *) textView
{
    self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
                                               initWithTitle:@"Done"
                                               style:UIBarButtonItemStyleDone
                                               target:self
                                               action:@selector(doneEditing:)]
                                               autorelease];
}

// The done button forces the text view to resign its first-responder status
- (void) doneEditing: (id) sender
{
    [contentView resignFirstResponder];
    self.navigationItem.rightBarButtonItem = NULL;
}
- (void)loadView
{
    // Load an application image and set it as the primary view
```

```
    contentView = [[UITextView alloc] initWithFrame:[[UIScreen mainScreen]
    ➥applicationFrame]];
    [contentView setDelegate:self];
    self.view = contentView;
    [contentView release];
}

-(void) dealloc
{
    [contentView release];
    [super dealloc];
}
@end
```

8.8　秘诀：向文本视图添加一个撤销（Undo）按钮

秘诀8-7显示了如何在一个文本视图内部捕获用户交互。UITextViewDelegate 协议也可以理解用户对显示文本所做的更改。当用户在视图中键入或删除文本时，就会触发 textView: shouldChangeTextInRange: replacementText:方法。通过返回 YES 并显示一个撤销按钮，可以创建支持恢复到最近保存状态的基本结构。

秘诀8-8向秘诀8-7中构建的基本结构添加了持久性数据。现在，当用户点击 Done 按钮以取消键盘时，应用程序会把文本的一个副本保存到一个文件中。在编辑过程中，如果用户点击了 Undo 按钮，程序会用该文件的内容代替文本视图的内容，并重新隐藏 Undo 按钮。

这些更改确保了只有当可见文本相比存储文本有了变化时才显示 Undo 按钮。图8-8显示了使用此秘诀构建的界面，即用户文本更改触发 Undo 按钮时的界面。

图8-8　屏幕左上角的 Undo 按钮只有在用户更改了文本视图中最近存储状态下的文本时才出现。点击 Done 按钮会把所有更改保存到

文件中（并隐藏 Undo 按钮）。点击 Undo 按钮会把视图内容恢复到最近存储的状态

秘诀8-8　向文本视图添加撤销支持

```
@interface HelloController : UIViewController <UITextViewDelegate>
{
    UITextView *contentView;
    BOOL readyToUndo;
}
@end

@implementation HelloController

#define FILEPATH [NSHomeDirectory()
➥stringByAppendingPathComponent:@"Documents/MyText.txt"]
// Reveal a Done button when starting to edit
- (void) textViewDidBeginEditing: (UITextView *) textView
{
    self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
                                               initWithTitle:@"Done"
                                               style:UIBarButtonItemStyleDone
                                               target:self
                                               action:@selector(doneEditing:)]
                                               autorelease];
    readyToUndo = YES;

}

// Only reveal the undo after text has actually been typed or deleted
- (BOOL)textView:(UITextView *)textView shouldChangeTextInRange:(NSRange)range
➥replacementText:(NSString *)text
{
    if (readyToUndo)
        self.navigationItem.leftBarButtonItem = [[[UIBarButtonItem alloc]
                                                  initWithTitle:@"Undo"
                                                  style:UIBarButtonItemStylePlain
                                                  target:self
                                                  action:@selector(undoEditing:)]
                                                  autorelease];
    readyToUndo = NO;
    return YES;
}

// The Done button forces the text view to resign its first-responder status and
➥saves changes
- (void) doneEditing: (id) sender
{
    NSError *error;

    [contentView resignFirstResponder];
    self.navigationItem.rightBarButtonItem = NULL;
    self.navigationItem.leftBarButtonItem = NULL;
    [[contentView text] writeToFile:FILEPATH atomically: YES
    ➥encoding:NSUTF8StringEncoding error: &error];
}
```

```
// Undo means revert to last saved state
- (void) undoEditing: (id) sender
{
    if ([[NSFileManager defaultManager] fileExistsAtPath:FILEPATH])
        [contentView setText:[NSString stringWithContentsOfFile:FILEPATH]];
    else
        [contentView setText:@""];

    readyToUndo = YES;
    self.navigationItem.leftBarButtonItem = NULL;
}

- (void)loadView
{
    // Load an application image and set it as the primary view
    contentView = [[UITextView alloc] initWithFrame:[[UIScreen mainScreen]
    ➥applicationFrame]];
    [contentView setDelegate:self];
    self.view = contentView;
    [contentView release];

    // Preload with text if any has been saved
    if ([[NSFileManager defaultManager] fileExistsAtPath:FILEPATH])
        [contentView setText:[NSString stringWithContentsOfFile:FILEPATH]];
}
@end
```

8.9　秘诀：创建一个基于文本视图的 HTML 编辑器

虽然苹果公司正式从 UITextView 类中删除了 HTML 支持，但它仍然隐藏在 UIKit 框架中。当希望添加简单的富文本扩展时，可以访问这项文档中未记录的特性。显然，苹果公司希望你使用 UIWebView 而不是 UITextView 来进行 HTML 显示，不过 UITextView 提供了更吸引人的特性。

要访问此 HTML 显示，需要声明 setContentToHTMLString:方法，如秘诀8-9中所示。这个文档中未记录的 UITextView 方法告知文本视图将一个字符串解释为 HTML 源。使用 UITextView 有以下两个好处。

l　UITextView 可以被编辑。你可以使用 HTML 文本初始化视图，并允许用户编辑结果。文本会选择周围元素的属性作为自己的属性。例如，如果向一个粗体标题行添加文本，新的文本也是粗体的。

l　第二，UITextView 可以很容易地被重新加载。例如，如果你允许用户在基于文本的源的模式中编辑文本，可以在 HTML 模式中把这些更改重新加载到相同的视图中。

秘诀8-9显示了如何构建一个简单的 HTML 编辑器。图8-9显示了相同的 UITextView 中的文本源和 HTML 表示。这并不是技术的通常用法（或实际用法），不过它突出了一项以后会用到的功能。通常，你只希望用一些富文本功能（如粗体的标题行）来初始化文本，然后允许用户直接编辑该文本。

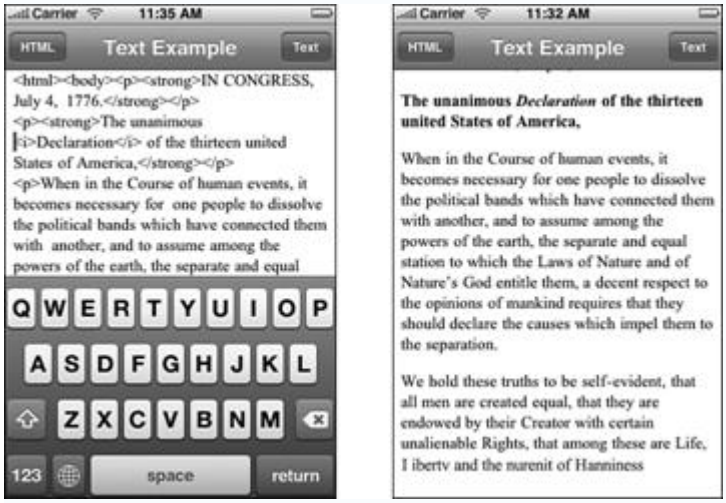

图8-9　这个基于 UITextView 的 HTML 编辑器允许用户在同一个视图中编辑源和查看结果

说明　　可以使用 NSLocalizeString 使文本国际化。向它传递来自语言文件夹的 Localizable.strings 文件（例如，从 French.lproj 或 English.lproj 文件）的任意键。NSLocalized String (@"Title", nil)在 strings 文件中查找 Title 键，并返回与该键匹配的本地化字符串。通过常规国际设置，iPhone 知道使用哪种语言，也知道要访问哪个 lproj 文件夹。为了使应用程序名国际化，需要向每个 lproj 文件夹添加一个 InfoPlist.strings 文件。在这些文件中设置 CFBundleDisplayName 键。SDK 发布之后，有关最新的详细信息可以参考苹果公司的文档。

秘诀8-9　使用文档中未记录的 UITextView HTML 特性

```
- (void) presentText: (id) sender
{
    // dismiss keyboard if it exists
    [contentView resignFirstResponder];
    // Store text only if previous editor was text-based
    if ([contentView tag]) self.sourceText = [contentView text];

    // Text-based Editor
    [contentView setText:self.sourceText];
    [contentView setEditable:YES];
    [contentView setTag:YES];
}

- (void) presentHTML: (id) sender
{

    // dismiss keyboard if it exists
    [contentView resignFirstResponder];

    // Store text only if previous editor was text-based
    if ([contentView tag]) self.sourceText = [contentView text];

    // HTML-based Presentation-only
    [contentView setContentToHTMLString:self.sourceText];
    [contentView setEditable:NO];
    [contentView setTag:NO];
}

- (void)loadView
{
    // Load an application image and set it as the primary view
    contentView = [[UITextView alloc] initWithFrame:[[UIScreen mainScreen]
    ➥applicationFrame]];
    [contentView setAutocorrectionType:UITextAutocorrectionTypeNo];
    [contentView setTag:NO];
    [contentView setEditable:NO];
    self.view = contentView;
    [contentView release];

    // Load the HTML source
    self.sourceText = [NSString stringWithContentsOfFile:[[NSBundle mainBundle]
    ➥pathForResource:@"decl" ofType:@"html"]];
    [contentView setContentToHTMLString:self.sourceText];

    // Add Text and HTML buttons
    self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
                                              initWithTitle:@"Text"
                                              style:UIBarButtonItemStylePlain
                                              target:self
                                              action:@selector(presentText:)]
                                              autorelease];
    self.navigationItem.leftBarButtonItem = [[[UIBarButtonItem alloc]
                                             initWithTitle:@"HTML"
                                             style:UIBarButtonItemStylePlain
                                             target:self
                                             action:@selector(presentHTML:)]
                                             autorelease];

}
```

8.10 秘诀：构建一个交互搜索栏

UISearchBar 类提供了一个简单的搜索主题文本字段，它显示在应用程序的导航栏中正好合适。图8-10实际地显示了一个搜索栏，在颜色画笔名中搜索匹配的项。搜索是实时的。当用户键入文本时，下面的列表会自动更新。

秘诀8-10使用 searchbar:textDidChange:方法捕获搜索栏文本更改。在用户键入新字符时，此方法重新构建用于填充屏幕列表的数组。它使搜索字段的文本与数组中的条目相匹配。

注意图8-10中使用的 Return 键换出。通常，搜索栏会延迟搜索，直到用户键入了一个词语。这就是它们为默认的 Return 键显示 Search 的原因。秘诀8-10的实时交互不使用上述搜索模型。取消键盘后，用户通过 Done 键可以检验他或她已经执行的当前搜索的结果。

图8-10　搜索栏提供了出色的交互控件，用于在冗长的列表中放置条目

要代替 Return 键，需要访问 UISearchBar 实例的搜索字段子视图，并为该字段设置文本特性。这时，文本视图是最前端（即最后一个）视图。如果在以后 SDK 发布中更改了此内容，应对应地调整秘诀代码。

秘诀8-10　使用带有实时反馈的搜索栏

```
// create an array by matching to the search string
- (void) buildSearchArrayFrom: (NSString *) matchString
{
    NSString *upString = [[matchString uppercaseString] autorelease];
    if (searchArray) [searchArray release];

    searchArray = [[NSMutableArray alloc] init];
    for (NSString *word in colorArray)
    {

        if ([matchString length] == 0)
        {
            [searchArray addObject:word];
            continue;
        }

        NSRange range = [[[[word componentsSeparatedByString:@" #"]
        ➥objectAtIndex:0] uppercaseString] rangeOfString:upString];
        if (range.location != NSNotFound) [searchArray addObject:word];
    }

    [self.tableView reloadData];
}

// When the search text changes, update the array
- (void)searchBar:(UISearchBar *)searchBar textDidChange:(NSString *)searchText
{
    [self buildSearchArrayFrom:searchText];
    if ([searchText length] == 0) [searchBar resignFirstResponder];
}

// When the search ("done") button is clicked, hide the keyboard
- (void)searchBarSearchButtonClicked:(UISearchBar *)searchBar
{
    [searchBar resignFirstResponder];
}
// Prepare the Table View
- (void)loadView
{
    [super loadView];

    // Retrieve the text and colors from file
    NSString *pathname = [[NSBundle mainBundle]  pathForResource:@"crayons"
    ➥ofType:@"txt" inDirectory:@"/"];
    NSString *wordstring = [NSString stringWithContentsOfFile:pathname];
    colorArray = [[wordstring componentsSeparatedByString:@"\n"] retain];
    [self buildSearchArrayFrom:@""];

    search = [[UISearchBar alloc] initWithFrame:CGRectMake(0.0f, 0.0f, 280.0f,
    ➥44.0f)];
    search.delegate = self;
    search.placeholder = @"color match text";
    search.autocorrectionType = UITextAutocorrectionTypeNo;
    search.autocapitalizationType = UITextAutocapitalizationTypeNone;
    self.navigationItem.titleView = search;
    [search release];

    // "Search" is the wrong key usage here. Replacing it with "Done"

    UITextField *searchField = [[search subviews] lastObject];
    [searchField setReturnKeyType:UIReturnKeyDone];
}
```

8.11　秘诀：添加标注（callout）视图

如果你在 iPhone 上使用过 Google Map，你可能在实战中见过 UICalloutView 实例。虽然它们的名称如此，但是它们是一种 UIControl 实例。它们是文档中未记录的，但在 UIKit 框架中可用。

标注视图指向屏幕上的某些内容。它们在使用附加的扩展按钮移动到另一个消息之前，可以显示一个临时消息。图8-11显示了带有几个标注视图的屏幕，其中有些标注视图显示它们的源（临时）消息，其他标注视图显示最终消息。秘诀8-11提供了构建此屏幕的代码。

图8-11　UICalloutView 实例指向某些内容。它们可以显示一个初始消息（这里是“You Tapped Here！”）和一个最终消息（这里是“More Info…”），以及一个附加的扩展按钮。苹果公司的 Google Map 应用程序中使用了标注

由于类是文档中未记录的，因此需要定义头文件。代码清单8-6显示了 UICalloutView 类定义。它还不完整，不过它提供了足够的功能使你能够在程序中有效地使用这个类。

代码清单8-6　定义 UICalloutView 界面

```
@class UIImageView, UILabel, UIPushButton;

@interface UICalloutView : UIControl
- (UICalloutView *)initWithFrame:(struct CGRect)aFrame;
- (void)fadeOutWithDuration:(float)duration;
- (void)setTemporaryTitle:(NSString *)fp8;
- (NSString *)temporaryTitle;
- (void)setTitle:(NSString *)fp8;
- (NSString *)title;
- (void)addTarget:(id)target action:(SEL)selector;
- (void)removeTarget:(id)target;
- (void)setAnchorPoint:(struct CGPoint)aPoint boundaryRect:(struct CGRect)aRect
➥animate:(BOOL)yorn;
@end
```

要创建一个标注，需要传递一个定位点——即标注指向的坐标——和一个广义（generalized）边界矩形。据我所知，这些限制完全被忽略了。

标注使用两个标题：一个临时标题和一个普通标题。当添加一个临时标题时，此标题显示大约一秒钟的时间，然后会变为普通标题，即带有扩展按钮的标题。还可以向标注添加子标题，不过那样看起来很可怕，这里不推荐使用。

通过调用 addTarget: action:方法，为扩展按钮设置目标。当扩展按钮被点击时，就调用传递的选择程序。这里不为标注上的用户点击计数，iPhone 忽略了它们。因此，如果希望在出现一个新标注时除去现有标注，应该由你来消除它。可以快速地使用 fadeOutWithDuration:方法并指定渐没时间。

秘诀8-11　使用 UICalloutView

```
@interface TapView : UIImageView
@end

@implementation TapView
- (void) hideDisclosure: (UIPushButton *) calloutButton
{
    UICalloutView *callout = (UICalloutView *)[calloutButton superview];
    [callout fadeOutWithDuration:1.0f];
}

- (UICalloutView *) buildDisclosure
{
    UICalloutView *callout = [[[UICalloutView alloc] initWithFrame:CGRectZero]
    ➥autorelease];
    callout.contentVerticalAlignment = UIControlContentVerticalAlignmentCenter;
```

```
    callout.contentHorizontalAlignment =
    ➥UIControlContentHorizontalAlignmentCenter;

    [callout setTemporaryTitle:@"You Tapped Here!"];
    [callout setTitle:@"More Info..."];
    [callout addTarget:self action:@selector(hideDisclosure:)];

    return callout;
}

- (void) showDisclosureAt:(CGPoint) aPoint
{
    UICalloutView *callout = [self buildDisclosure];
    [self addSubview:callout];
    [callout setAnchorPoint:aPoint boundaryRect:CGRectMake(0.0f, 0.0f, 320.0f,
    ➥100.0f) animate:YES];
}

- (void) touchesBegan:(NSSet*)touches withEvent:(UIEvent*)event
{
    [self showDisclosureAt:[[touches anyObject] locationInView:self]];
}

@end
```

8.12 添加一个页面指示器控件

令人伤心的是，UIPageControl 类并不能真正满足我们的期望。它和在 SpringBoard 中看到的出色类不一样。它们看起来相同，但执行起来不同。官方 SDK 中可用的 UIPageControl 类很难使用，通常会给用户带来许多烦恼。因此，在使用它时，要确保添加了可选择的导航选项（比如第2章中介绍的滑动），以便页面控件更像一个指示器，而不是一个控件。

图8-12实际地显示了一个页面控件。点击当前颜色鲜艳的页面指示器的左边或右边，会触发 UIControlEventValueChanged 事件，并启动设置为控件动作的任何方法。你可以通过调用 currentPage 查询控件的新值，并通过调整 numberOfPage 属性设置可用的页面数。

秘诀8-12显示了图像的5个独特页面，不过实际上，它只使用了两个图像视图。在任意给定的时间上，显示一个视图，另一个视图准备加载下一个图像 ——不管在数值上该图像在当前图像之前还是之后。当用户移动到一个新页面时，视图控制器将新的图像加载到未使用的图像视图中，并调用 UIViewAnimation 块以便在两个图像视图之间进行交换。两个“页面”之间的数值关系确定了交换会把图像从右边推到左边还是从左边推到右边。

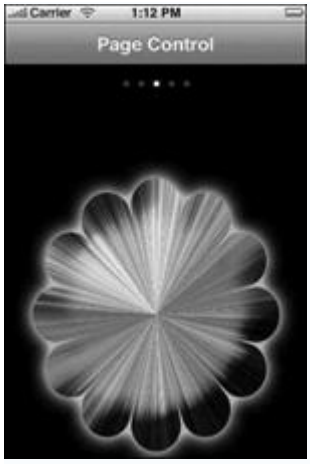

图8-12　UIPageControl 类提供了一个用于多页面表示的交互指示器。通过点击活动点的左边或右边，用户能够选择新的页面。至少理论上是这样。页面控件很难点击，而且其响应性能也很差

秘诀8-12　使用 UIPageControl 指示器

```
@interface HelloController : UIViewController
{
    SwipeView *contentView;
    int currentPage;
}
@end

@implementation HelloController

- (id)init
{
    if (!(self = [super init])) return self;
    self.title = @"Page Control";
    return self;
}

// Build and return an  animation with the given direction
```

```
- (CATransition *) getAnimation:(NSString *) direction
{
    CATransition *animation = [CATransition animation];
    [animation setDelegate:self];
    [animation setType:kCATransitionPush];
    [animation setSubtype:direction];
    [animation setDuration:1.0f];
    [animation setTimingFunction:[CAMediaTimingFunction
    ➥functionWithName:kCAMediaTimingFunctionEaseInEaseOut]];
    return animation;
}

// Perform the page turn animation, in this case a push
- (void) pageTurn: (UIPageControl *) pageControl
{
    CATransition *transition;
    int secondPage = [pageControl currentPage];
    if ((secondPage - currentPage) > 0)
        transition = [self getAnimation:@"fromRight"];
    else
        transition = [self getAnimation:@"fromLeft"];
    UIImageView *newView = (UIImageView *)[[contentView subviews]
    ➥objectAtIndex:0];
    [newView setImage:[UIImage imageNamed:[NSString stringWithFormat:@"f%d.png",
    ➥secondPage + 1]]];
    [contentView exchangeSubviewAtIndex:0 withSubviewAtIndex:1];
    [[contentView layer] addAnimation:transition
    ➥forKey:@"transitionViewAnimation"];

    currentPage = [pageControl currentPage];
}

- (void)loadView
{
    UIView *baseView = [[[UIView alloc] initWithFrame:[[UIScreen mainScreen]
    ➥applicationFrame]] autorelease];

    // Add the content view with its two images
    contentView = [[[SwipeView alloc] initWithFrame:[[UIScreen mainScreen]
    ➥applicationFrame]] autorelease];
    [contentView setHost:self];


    [contentView addSubview:[[UIImageView alloc] initWithFrame:[[UIScreen
    ➥mainScreen] applicationFrame]]];
    [contentView addSubview:[[UIImageView alloc] initWithFrame:[[UIScreen
    ➥mainScreen] applicationFrame]]];
```

```
    for (UIView *view in [contentView subviews])  [view
    ➥setUserInteractionEnabled:NO];

    // initialize with "page" 3
    [[[contentView subviews] lastObject] setImage:[UIImage imageNamed:@"f3.png"]];

    [baseView addSubview:contentView];

    // Add the page control
    UIPageControl *pageControl = [[UIPageControl alloc]
    ➥initWithFrame:CGRectMake(0.0f, 10.0f, 320.0f, 20.0f)];
    [pageControl setNumberOfPages:5];
    [pageControl setCurrentPage:(currentPage = 2)];
    [pageControl addTarget:self action:@selector(pageTurn:)
    ➥forControlEvents:UIControlEventValueChanged];
    [baseView addSubview:pageControl];
    self.view = baseView;
}

-(void) dealloc
{
    [contentView release];
    [super dealloc];
}
@end
```

8.13　秘诀：定制工具栏

UITabBar 类拥有对定制的官方支持，而 UIToolBar 类没有。你可以把视图控制器从它的 More 屏幕拖到或拖出标签栏（tab bar）。而对于工具栏，你不能这么做。至少不能正式地这么做。

非正式情况下，UIToolbar 提供了一个相似的（不过是文档中未记录的）交互类型。虽然不能添加新的条目或换出条目，但可以对工具栏重新排序。它并不是非常灵活的交互类型，不过你可以使用它让用户定制工具栏的显示外观或让用户重新排列 Scrabble 信件。用法由你来决定。和公共框架中的所有未记录的调用一样，你可以在程序中自由地使用它们，不过它们会在不预先通知的情况下做出更改。图8-13实际地显示了工具栏定制屏幕。

工具栏中的所有对象都出现在定制屏幕中。也就是说，如果你添加固定或可变空间，它们会像图标一样在那里显示。必须为完成编辑工具栏添加一个外部方法，因为一旦进入定制模式，触发的动作在工具栏按钮上就不起作用了。这正是屏幕右上角 Done 按钮的任务。

秘诀8-13显示了如何添加工具栏定制。它扩展了 UIToolbar 类，以显示文档中未记录的开始和结束定制方法。

图8-13　文档中未记录的 UIToolbar 定制屏幕允许用户对工具栏条目重新排列

秘诀8-13　定制工具栏

```
// Expose the customizing calls
@interface UIToolbar (extended)
- (void)beginCustomizingItems:(id)fp8;
- (BOOL)endCustomizingAnimated:(BOOL)fp8;
@end;

@interface HelloController : UIViewController
{
    UIImageView *contentView;
    UIToolbar *toolbar;
    NSMutableArray *allitems;
}
@end

@implementation HelloController

// When finished customizing, remove the Done button
- (void) endEditing
{
    [toolbar endCustomizingAnimated:YES];
    self.navigationItem.rightBarButtonItem = NULL;
}

// Reveal the Done button and begin customizing
- (void) doAction: (id) sender
{
    [toolbar beginCustomizingItems:allitems];
    self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
```

```
                                        initWithTitle:@"Done"
                                        style:UIBarButtonItemStylePlain
                                        target:self
                                        action:@selector(endEditing)]
                                        autorelease];
}

- (void)loadView
{

    // Load an application image and set it as the primary view
    contentView = [[UIView alloc] initWithFrame:[[UIScreen mainScreen]
    ➥applicationFrame]];
    [contentView setImage:[UIImage imageNamed:@"bluedots.png"]];
    [contentView setUserInteractionEnabled:YES];
    self.view = contentView;
    [contentView release];

    toolbar = [[UIToolbar alloc] initWithFrame:CGRectMake(0.0f, 416.0f - 44.0f,
    ➥320.0f, 44.0f)];

    // Add a whole bunch of arbitrary system items with icons
    allitems = [[NSMutableArray alloc] init];
    [allitems addObject:[[[UIBarButtonItem alloc]
        ➥initWithBarButtonSystemItem:UIBarButtonSystemItemAdd
        ➥target:self action:@selector(doAction:)] autorelease]];
    [allitems addObject:[[[UIBarButtonItem alloc]
        ➥initWithBarButtonSystemItem:UIBarButtonSystemItemBookmarks
        ➥target:self action:@selector(doAction:)] autorelease]];
    [allitems addObject:[[[UIBarButtonItem alloc]
        ➥initWithBarButtonSystemItem:UIBarButtonSystemItemCamera
        ➥target:self action:@selector(doAction:)] autorelease]];
    [allitems addObject:[[[UIBarButtonItem alloc]
        ➥initWithBarButtonSystemItem:UIBarButtonSystemItemCompose
        ➥target:self action:@selector(doAction:)] autorelease]];
    [allitems addObject:[[[UIBarButtonItem alloc]
        ➥initWithBarButtonSystemItem:UIBarButtonSystemItemOrganize
        ➥target:self action:@selector(doAction:)] autorelease]];

    [allitems addObject:[[[UIBarButtonItem alloc]
        ➥initWithBarButtonSystemItem:UIBarButtonSystemItemRefresh
        ➥target:self action:@selector(doAction:)] autorelease]];
    [allitems addObject:[[[UIBarButtonItem alloc]
        ➥initWithBarButtonSystemItem:UIBarButtonSystemItemReply
        ➥target:self action:@selector(doAction:)] autorelease]];
    [allitems addObject:[[[UIBarButtonItem alloc]
        ➥initWithBarButtonSystemItem:UIBarButtonSystemItemSearch
        ➥target:self action:@selector(doAction:)] autorelease]];
    [allitems addObject:[[[UIBarButtonItem alloc]
        ➥initWithBarButtonSystemItem:UIBarButtonSystemItemStop
        ➥target:self action:@selector(doAction:)] autorelease]];
    [allitems addObject:[[[UIBarButtonItem alloc]
        ➥initWithBarButtonSystemItem:UIBarButtonSystemItemTrash
        ➥target:self action:@selector(doAction:)] autorelease]];
```

```
    [toolbar setItems:allitems];
    [contentView addSubview:toolbar];
    [toolbar release];
}

-(void) dealloc
{
    [toolbar release];
    [allitems release];
    [contentView release];
    [super dealloc];
}
@end
```

工具栏提示

使用工具栏时，下面是一些将来会用到的技巧。

l 固定空间可以拥有宽度。

在所有 UIBarButtonItems 中，只有 UIBarButtonSystemItemFixedSpace 条目可以被分配一个宽度。因此，创建空间条目，设置其宽度，然后只把它添加到条目数组中。

```
UIBarButtonItem *spacer = [[[UIBarButtonItem alloc]
initWithBarButtonSystemItem:UIBarButtonSystemItemFixedSpace
target:NULL action:NULL] autorelease];
spacer.width = 20;
[items addObject:spacer];
```

l 使用一个灵活空间进行左对齐或右对齐。

在条目列表开始添加一个 UIBarButtonSystemItemFlexibleSpace 会使所有剩余条目右对齐。在末尾添加一个 UIBarButtonSystemItemFlexibleSpace 会使所有剩余条目左对齐。使用两个 UIBarButtonSystemItemFlexibleSpace，一个添加在开头，一个添加在末尾，会使剩余条目居中对齐。

l 考虑遗漏的条目。

根据上下文隐藏栏按钮条目时，不要只使用灵活的空间分配来除去它们。而是使用一个与条目原来大小相匹配的固定宽度的空间代替该条目。那样做会在条目消失之前和消失之后保存布局并把所有其他图标放在原来的位置。

l 栏按钮条目不是视图。

如果你需要向一个栏按钮条目添加子视图，应该为它提供一个合适的父视图并让该父视图（不管是工具栏、标签栏还是导航栏）在屏幕上构建它。然后，可以使用文档中未记录的 view 调用来恢复栏按钮条目的视图。

8.14 小结

本章介绍了许多与控件进行交互和在程序中使控件发挥最大作用的非标准（有些是非官方和文档中未记录的）方法。在继续下一章的内容之前，先回顾一下本章的一些要点。

l 不要害怕查看控件，你会发现它们是如何组织在一起的。公开了像标签和按钮这样的标准 SDK 类之后，就可以根据自己的规范和期望来定制它们。

l 如果只是一个条目属于 UIControl 类，这并不意味着你可以把它当作一个 UIView 来对待。可以为它提供子视图，调整其大小，为它制作动画，在屏幕周围移动它或在以后点击它 。

l 利用 Core Graphic 可以根据需要构建可见元素。使用 SDK 类的舒适感和随时发出的叫好声会为外观呈现增加砝码。

l 和搜索栏键盘示例一样，不要让苹果公司有关应该如何使用控件的预想概念妨碍你以自己希望的方式使用它。如果“search”没有描述在交互结束时用户需要做什么，就把 Return 按钮转换为一个能完成此任务的短语。

l 标注视图只是 UIKit 载有的许多未在文档中记录的类中的一个，它丰富了应用程序。

l 本章介绍了许多文档中未记录的类和调用。和所有未在文档中记录的和非官方的资料一样，在软件中应该小心地使用它们。