

介绍

请注意：本文档之前命名为 *iPhone OS 编程指南*。

[文章出处：

<http://www.apple.com.cn/developer/iphone/library/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/Introduction/Introduction.html>]

iPhone SDK 为创建 iPhone 的本地应用程序提供必需的工具和资源。在用户的 Home 屏幕上，iPhone 的本地应用程序表示为图标。它们和运行在 Safari 内部的 web 应用程序不同，在基于 iPhone OS 的设备上，它们作为独立的执行程序来运行。本地应用程序可以访问 iPhone 和 iPod Touch 的所有特性，比如加速计、位置服务、和多点触摸接口，正是这些特性使设备变得更加有趣。本地应用程序还可以将数据保存在本地的文件系统中，甚至 可以通过定制的 URL 类型来和安装在设备上的其它程序进行通讯。



为 iPhone OS 开发本地应用程序需要使用 UIKit [框架](#)。利用该框架提供的基础设施和缺省行为，您可以在几分钟内创建一个具有一定功能的应用程序。UIKit 框架（和系统中的其它框架）不但提供大量的缺省行为，而且提供了一些挂钩，开发者可以通过这些挂钩来定制和扩展它的行为。

谁应该阅读本文？

本文的目标读者是希望创建 iPhone 本地应用程序的新老 iPhone OS 开发者，目的是向您介绍 iPhone 应用程序的架构，展示 UIKit 和其它重要系统框架中的一些关键的定制点。在介绍这些内容的同时，本文还将提供一些有助于正确设计的指导意见。文中还指出一些为特定主题提供建议和进行进一步讨论的其它文档。

虽然本文描述的很多框架也存在于 Mac OS X 系统中，但阅读本文并不需要熟悉 Mac OS X 及其技术。

先决条件

在开始阅读本文之前，您必须至少对下面这些 Cocoa 概念有基本的理解：

有关 Xcode 和 Interface Builder 的基本信息及其在应用程序开发中的作用。

如何定义新的 [Objective-C](#) 类。

如何[管理内存](#)包括如何[创建](#)和释放 Objective-C 对象。

[委托](#)对象在管理应用程序行为中的作用。

目标-动作范式在用户界面管理中的作用。

不熟悉 Cocoa 和 Objective-C 的开发者可以在 [Cocoa 基本原理指南](#)中得到相应的信息。

iPhone 应用程序的开发需要在运行 Mac OS X v10.5或更高版本系统以及基于 Intel 的 Macintosh 电脑上进行，还必须下载和安装 iPhone SDK。有关如何得到 iPhone SDK 的信息，请访问 <http://www.apple.com.cn/developer/iphone/>网站。

核心应用程序

所有的 iPhone 应用程序都是基于 UIKit 框架构建而成的，因此，它们在本质上具有相同的核心架构。UIKit 负责提供运行应用程序和协调用户输入及屏幕显示所需要的关键对象。应用程序之间不同的地方在于如何配置缺省对象，以及如何通过定制对象来添加用户界面和行为。

虽然应用程序的界面和基本行为的定制发生在定制代码的内部，但是，还有很多定制需要在应用程序的最高级别上进行。这些高级的定制会影响应用程序和系统、以及 和设备上的其它程序之间的交互方式，因此，理解何时需要定制、何时缺省行为就已经足够是很重要的。本章将概要介绍核心应用程序架构和高级别的定制点，帮助 您确定什么时候应该定制，什么时候应该使用缺省的行为。

核心应用程序架构

从应用程序启动到退出的过程中，UIKit 框架负责管理大部分关键的基础设施。iPhone 应用程序不断地从系统接收事件，而且必须响应那些事件。接收事件是 [UIApplication](#) 对象的工作，但是，响应事件则需要您的定制代码来处理。为了理解事件响应需要在哪里进行，我们有必要对 iPhone 应用程序的整个生命周期和事件周期有一些理解。本文的下面部分将描述这些周期，同时还对 iPhone 应用程序开发过程中使用的一些关键设计模式进行总结。

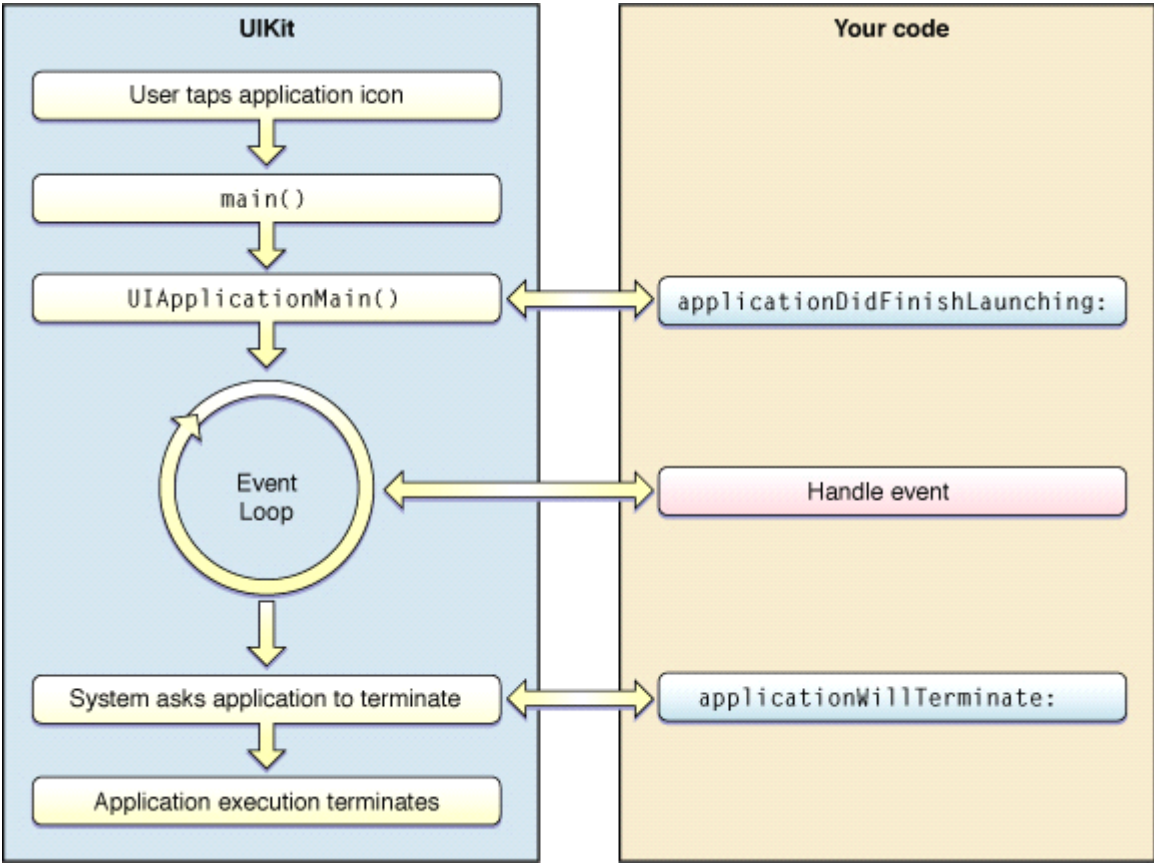
应用程序的生命周期

应用程序的生命周期是由发生在程序启动到终止期间的一序列事件构成的。在 iPhone OS 中，用户可以通过轻点 Home 屏幕上的图标来启动应用程序。在轻点图标之后的不久，系统就会显示一个过渡图形，然后调用相应的 main 函数来启动应用程序。从这个点之后，大量的初始化工作就会交给 UIKit，由它装载应用程序的[用户界面](#)和准备事件循环。在事件循环过程中，UIKit 会将事件分发给您的[定制对象](#)及响应应用程序发出的命令。当用户进行退出应用程序的操作时，UIKit 会通知应用程序，并开始应用程序的终止过程。

图1-1显示了一个简化了的 iPhone 应用程序生命周期。这个框图展示了发生在应用程序启动

到退出过程中的事件序列。在应用程序初始化和终止的时候，UIKit 会向应用程序委托对象发送特定的消息，使其知道正在发生的事件。在事件循环中，UIKit 将事件派发给应用程序的定制事件处理器。有关初始化和终止事件的如何处理的信息，将在随后的“[初始化和终止](#)”部分进行讨论；事件处理的过程则在“[事件处理周期](#)”部分介绍，在后面的章节也还有更为详细的讨论。

图1-1 应用程序的生命周期



主函数

在 iPhone 的应用程序中，main 函数仅在最小程度上被使用，应用程序运行所需的大多数实际工作由 [UIApplicationMain](#) 函数来处理。因此，当您在 Xcode 中开始一个新的应用程序工程时，每个工程模板都会提供一个 main 函数的标准实现，该实现和“处理关键的应用程序任务”部分提供的实现是一样的。main 例程只做三件事：创建一个 [自动释放池](#)，调用 UIApplicationMain 函数，以及使用自动释放池。除了少数的例外，您永远不应该改变这个函数的实现。

程序清单1-1 iPhone 应用程序的 main 函数

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

int main(int argc, char *argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc]
```

```
init];  
    int retVal = UIApplicationMain(argc, argv, nil, nil);  
    [pool release];  
    return retVal;  
}
```

请注意：自动释放池用于内存管理，它是 Cocoa 的一种机制，用于延缓释放具有一定功能的代码块中创建的对象。有关自动释放池的更多信息，请参见 [Cocoa 内存管理编程指南](#)；如果需要了解与自动释放池有关的具体内存管理规则，则请参见[“恰当地分配内存”](#)部分。程序清单的核心代码是 UIApplicationMain 函数，它接收四个参数，并将它们用于初始化应用程序。传递给该函数的缺省值并不需要修改，但是它们对于应用程序启动的作用还是值得解释一下。除了传给 main 函数的 argc 和 argv 之外，该函数还需要两个字符串参数，用于标识应用程序的首要类（即应用程序对象所属的类）和应用程序委托类。如果首要类字符串的值为 nil，UIKit 就缺省使用 [UIApplication](#) 类；如果应用程序委托类为 nil，UIKit 就会将应用程序主 nib 文件（针对通过 Xcode 模板创建的应用程序）中的某个对象假定为应用程序的委托对象。如果您将这些参数设置为非 nil 值，则在应用程序启动时，UIApplicationMain 函数会创建一个与传入值相对应的类实例，并将它用于既定的目的。因此，如果您的应用程序使用了 UIApplication 类的定制子类（这种做法是不推荐的，但确实是可能的），就需要在第三个参数指定该定制类的类名。

应用程序的委托

监控应用程序的高级行为是应用程序[委托](#)对象的责任，而应用程序委托对象是您提供的定制类实例。委托是一种避免对复杂的 UIKit 对象（比如缺省的 UIApplication 对象）进行子类化的机制。在这种机制下，您可以不进行子类化和方法重载，而是将自己的定制代码放到委托对象中，从而避免对复杂对象进行修改。当您感兴趣的事件发生时，复杂对象会将消息发送给您定制的委托对象。您可以通过这种“挂钩”执行自己的定制代码，实现需要的行为。

重要提示：委托模式的目的是使您在创建应用程序的时候省时省力，因此是非常重要的设计模式。如果您需要概要了解 iPhone 应用程序中使用的重要设计模式，请参见[“基本设计模式”](#)部分；如果需要对委托和其它 UIKit 设计模式的详细描述，则请参见 [Cocoa 基本原理指南](#)部分。

应用程序的委托对象负责处理几个关键的系统消息。每个 iPhone 应用程序都必须有应用程序委托对象，它可以是您希望的任何类的实例，但需要遵循 [UIApplicationDelegate](#) 协议，该协议的方法定义了应用程序生命周期中的某些挂钩，您可以通过这些方法来实现定制的行为。虽然您不需要实现所有的方法，但是每个应用程序委托都应该实现[“处理关键的应用程序任务”](#)部分中描述的方法。

有关 UIApplicationDelegate 协议方法的更多信息请参见 [UIApplicationDelegate 协议参考](#)。

主 Nib 文件

初始化的另一个任务是装载应用程序的主 [nib 文件](#)。如果应用程序的信息[属性列表](#)(Info.plist)文件中含有 NSMainNibFile 键，则作为初始化过程的一个部分，UIApplication 对象会装载该键指定的 nib 文件。主 nib 文件是唯一一个自动装载的 nib 文件，其它的 nib 文件可以在稍后根据需要进行装载。

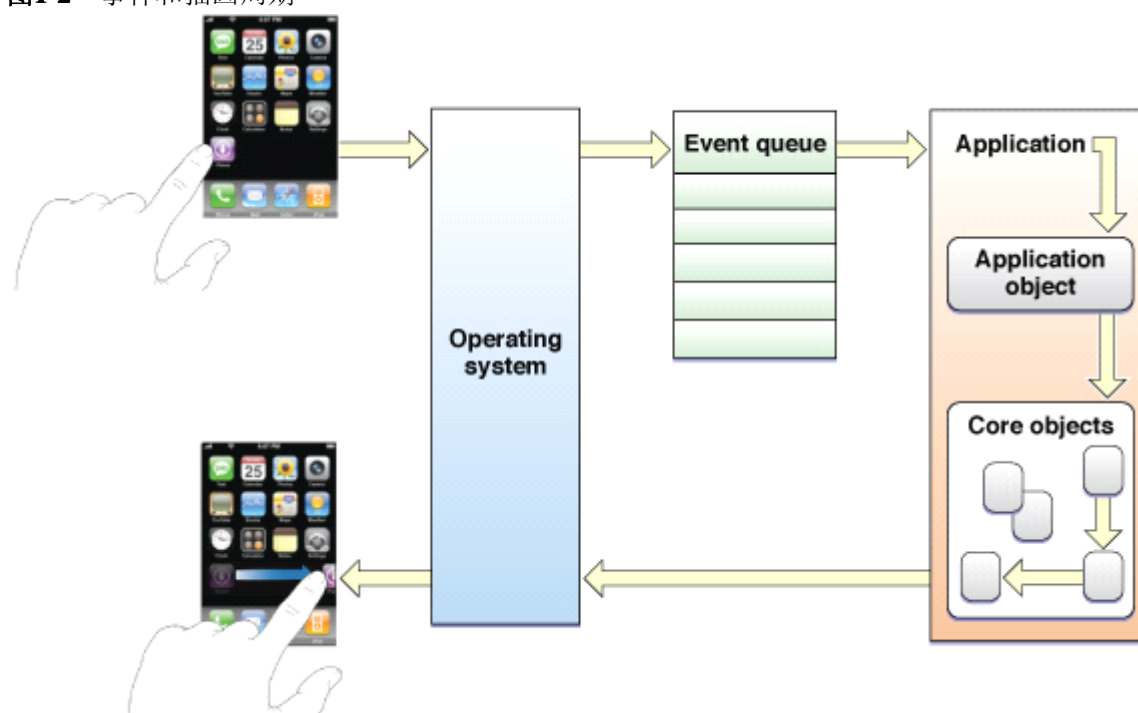
Nib 文件是基于磁盘的资源文件，用于存储一或多个对象的快照。iPhone 应用程序的主 nib 文件通常包含一个窗口对象和一个应用程序委托对象，还可能包含一个或多个管理窗口的其它重要对象。装载一个 nib 文件会使该文件中的对象被重新构造，从而将每个对象的磁盘表示转化为应用程序可以操作的内存对象。从 nib 文件中装载的对象和通过编程方式创建的对象之间没有区别。然而，对于用户界面而言，以图形的方式（使用 Interface Builder 程序）创建与用户界面相关联的对象并将它们存储在 nib 文件中通常比以编程的方式进行创建更加方便。

有关 nib 文件及其在 iPhone 应用程序中如何使用的更多信息，请参见[“Nib 文件”](#)部分，有关如何为应用程序指定主 nib 文件的信息则请参见[“信息属性列表”](#)部分。

事件处理周期

在应用程序初始化之后，UIApplicationMain 函数就会启动管理应用程序事件和描画周期的基础组件，如图1-2所示。在用户和设备进行交互的时候，iPhone OS 会检测触摸事件，并将事件放入应用程序的事件队列。然后，UIApplication 对象的事件处理设施会从队列的上部逐个取出事件，将它分发到最适合对其进行处理的对象。举例来说，在一个按键上发生的触摸事件会被分发到对应的按键对象。事件也可以被分发给[控制器对象](#)和应用程序中不直接负责处理触摸事件的其它对象。

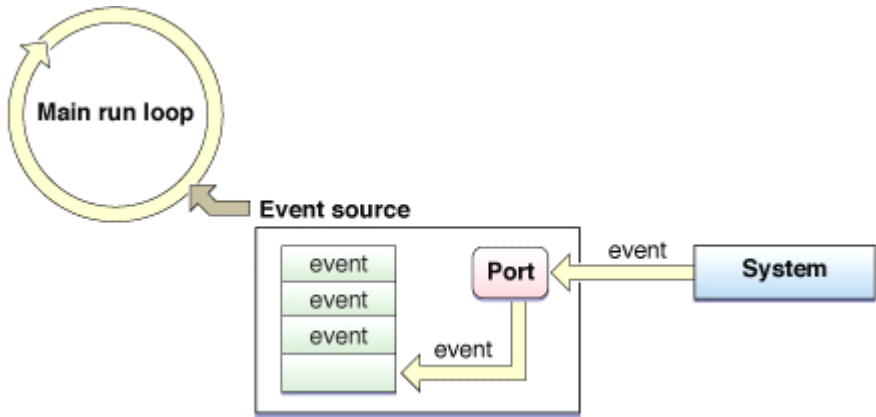
图1-2 事件和描画周期



在 iPhone OS 的多点触摸事件模型中，触摸数据被封装在事件对象（[UIEvent](#)）中。为了跟踪触摸动作，事件对象中包含一些触摸对象（[UITouch](#)），每个触摸对象都对应于一个正在触摸屏幕的手指。当用户把手指放在屏幕上，然后四处移动，并最终离开屏幕的时候，系统通过对应的触摸对象报告每个手指的变化。

在启动一个应用程序时，系统会为该程序创建一个进程和一个单一的线程。这个初始线程成为应用程序的主线程，UIApplication 对象正是在这个线程中建立主运行循环及配置应用程序的事件处理代码。图1-3显示了事件处理代码和主运行循环的关系。系统发送的触摸事件会在队列中等待，直到被应用程序的主运行循环处理。

图1-3 在主运行循环中处理事件



请注意：运行循环负责监视指定执行线程的输入源。当输入源有数据需要处理的时候，运行循环就唤醒相应的线程，并将控制权交给输入源的处理器代码。处理器在完成任务后将控制权交回运行循环，然后，运行循环就处理下一个事件。如果没有其它事件，运行循环会使线程进入休眠状态。您可以通过 Foundation 框架的 [NSRunLoop](#) 类来安装自己的输入源，包括端口和定时器。更多有关 NSRunLoop 和运行循环的一般性讨论，请参见[线程编程指南](#)。UIApplication 对象用一个处理触摸事件的输入源来配置主运行循环，使触摸事件可以被派发到恰当的响应者对象。响应者对象是继承自 [UIResponder](#) 类的对象，它实现了一或多个事件方法，以处理触摸事件不同阶段发生的事件。应用程序的响应者对象包括 UIApplication、[UIWindow](#)、[UIView](#)、及所有 UIView 子类的实例。应用程序通常将事件派发给代表应用程序主窗口的 UIWindow 对象，然后由窗口对象将事件传送给它的**第一响应者**，通常是发生触摸事件的视图对象（UIView）。

除了定义事件处理方法之外，UIResponder 类还定义了**响应者链**的编程结构。响应者链是为实现 Cocoa 协作事件处理而设计的机制，它由应用程序中一组链接在一起的响应者对象组成，通常以第一响应者作为链的开始。当发生某个事件时，如果第一响应者对象不能处理，就将它传递给响应者链中的下一个对象。消息继续在链中传递——从底层的响应者对象到诸如窗口、应用程序、和应用程序委托这样的高级响应者对象一直到事件被处理。如果事件最终没有被处理，就会被丢弃。

进行事件处理的响应者对象可能发起一系列程序动作，结果导致应用程序重画全部或部分用户界面（也可能导致其它结果，比如播放一个声音）。举例来说，一个按键对象（也就是一个 [UIControl](#) 的子类对象）在处理事件时向另一个对象（通常是控制器对象，负责管理当前活动的视图集合）发送动作消息。在处理这个动作消息时，控制器可能以某种方式改变用户界面或者视图的位置，而这又要求某些视图对自身进行重画。如果这种情况发生，则视图和图形基础组件会接管控制权，尽可能以最有效的方式处理必要的重画事件。

更多有关事件、响应者、和如何在定制对象中处理事件的信息，请参见[“事件处理”](#)部分；更多有关窗口及视图如何与事件处理机制相结合的信息，请参见[“视图交互模型”](#)部分；有关图形组件及视图如何被更新的更多信息，则请参见[“视图描画周期”](#)部分。

基本设计模式

UIKit 框架的设计结合了很多在 Mac OS X Cocoa 应用程序中使用的设计模式。理解这些设计模式对于创建 iPhone 应用程序是很关键的，我们值得为此花上几分钟时间。下面部分将简要概述这些设计模式。

表1-1 iPhone 描述

应用程序使用的设计模式设计模式

[模型-视图-控制器](#)

模型-视图-控制器(MVC)模式将您的代码分割为几个独立的部分。**模型**部分定义应用程序的数据引擎，负责维护数据的完整性；**视图**部分定义应用程序的用户界面，对显示在用户界面上的数据出处则没有清楚的认识；**控制器**部分则充当模型和控制器的桥梁，帮助实现数据和显示的更新。

[委托](#)

委托模式可以对复杂对象进行修改而不需要子类化。与子类化不同的是，您可以照常使用复杂对象，而将对其行为进行修改的定制代码放在另一个对象中，这个对象就称为委托对象。复杂对象需要在预先定义好的时点上调用委托对象的方法，使其有机会运行定制代码。

[目标-动作](#)

控件通过**目标-动作**模式将用户的交互通知给您的应用程序。当用户以预先定义好的方式（比如轻点一个按键）进行交互时，控件就会将消息（动作）发送给您指定的对象（目标）。接收到动作消息后，目标对象就会以恰当的方式进行响应（比如在按动按键时更新应用程序的状态）。

[委托内存模型](#)

Objective-C 使用引用计数模式来确定什么时候应该释放内存中的对象。当一个对象刚刚被创建时，它的引用计数是1。然后，其它对象可以通过该对象的 [retain](#)、[release](#)、或 [autorelease](#) 方法来增加或减少引用计数。当对象的引用计数变为0时，Objective-C 运行环境会调用对象的清理例程，然后解除分配该对象。

有关这些设计模式更为详尽的讨论请参见 [Cocoa 基本原理指南](#)。

应用程序运行环境

iPhone OS 的运行环境被设计为快速而安全的程序执行环境。下面的部分这个运行环境的关键部分，并就如何在这个环境中进行操作提供一些指导。

启动过程快，使用时间短

iPhone OS 设备的优势是它们的便捷性。用户通常从口袋里掏出设备，用上几秒或几分钟，就又放回口袋中了。在这个过程中，用户可能会打电话、查找联系人、改变正在播放的歌曲、或者取得一片信息。

在 iPhone OS 中，每次只能有一个前台应用程序。这意味着每次用户在 **Home** 屏幕上轻点您的应用程序图标时，您的程序必须快速启动和初始化，以尽可能减少延迟。如果您的应用程序花很长时间来启动，用户可能就不喜欢了。

除了快速启动，您的应用程序还必须做好快速退出的准备。每次用户离开您的应用程序时，无论是按下 **Home** 键还是通过软件提供的功能打开了另一个应用程序，iPhone OS 会通知您的应用程序退出。在那个时候，您需要尽快将未保存的修改保存到磁盘上。如果您的应用程序退出的时间超过5秒，系统可能会立刻终止它的运行。

当用户切换到另一个应用程序时，虽然您的程序不是运行在后台，但是我们鼓励您使它看起来好像是在后台运行。当您的程序退出时，除了对未保存的数据进行保存之外，还应该保存当前的状态信息；而在启动时，则应该寻找这些状态信息，并将程序恢复到最后一次使用时的状态。这样可以使用户回到最后一次使用时的状态，使用户体验更加一致。以这种

方式保存用户的当前位置还可以避免每次启动都需要经过多个屏幕才能找到需要的信息，从而节省使用的时间。

应用程序沙箱

由于安全的原因，iPhone OS 将每个应用程序（包括其偏好设置信息和数据）限制在文件系统的特定位置上。这个限制是安全特性的一部分，称为应用程序的“沙箱”。沙箱是一组细粒度的控制，用于限制应用程序对文件、偏好设置、网络资源、和硬件等的访问。在 iPhone OS 中，应用程序和它的数据驻留在一个安全的地方，其它应用程序都不能进行访问。在应用程序安装之后，系统就通过计算得到一个不透明的标识，然后基于应用程序的根目录和这个标识构建一个指向应用程序家目录的路径。因此，应用程序的家目录具有如下结构：

/ApplicationRoot/ApplicationID/

在安装过程中，系统会创建应用程序的家目录和几个关键的子目录，配置应用程序沙箱，以及将应用程序的程序包拷贝到家目录上。将应用程序及其数据放在一个特定的地方可以简化备份-并-恢复操作，还可以简化应用程序的更新及卸载操作。有关系统为每个应用程序创建的专用目录、应用程序更新、及备份-并-恢复操作的更多信息，请参见[“文件和数据管理”](#)部分。

重要提示：沙箱可以限制攻击者对其它程序和系统造成的破坏，但是不能防止攻击的发生。换句话说，沙箱不能使您的程序避免恶意的直接攻击。举例来说，如果在您的输入处理代码中有一个可利用的缓冲区溢出，而您又没有对用户输入进行正当性检查，则攻击者可能仍然可以使您的应用程序崩溃，或者通过这种漏洞来执行攻击者的代码。

虚拟内存系统

在本质上，iPhone OS 使用与 Mac OS X 同样的虚存系统。在 iPhone OS 中，每个程序都仍然有自己的虚拟地址空间，但其可用的虚拟内存受限于现有的物理内存的数量（这和 Mac OS X 不同）。这是因为当内存用满的时候，iPhone OS 并不将非永久内存页面（volatile pages）写入到磁盘。相反，虚拟内存系统会根据需要释放永久内存（nonvolatile memory），确保为正在运行的应用程序提供所需的内存。内存的释放是通过删除当前没有正在使用或包含只读内容（比如代码页面）的内存页面来实现的，这样的页面可以在稍后需要的时候重新装载到内存中。

如果内存还是不够，系统也可能向正在运行的应用程序发出[通告](#)，要求它们释放额外的内存。所有的应用程序都应该响应这种通告，并尽自己所能减轻系统的内存压力。有关如何在应用程序中处理这种通告的更多信息，请参见[“观察低内存警告”](#)部分。

自动休眠定时器

iPhone OS 试图省电的一个方法是使用自动休眠定时器。如果在一定的时间内没有检测到触摸事件，系统最初会使屏幕变暗，并最终完全关闭屏幕。大多数开发者都应该让这个定时器打开，但是，游戏和不使用触摸输入的应用程序开发者可以禁用这个定时器，使屏幕在应用程序运行时不会变暗。将共享的 [UIApplication](#) 对象的 [idleTimerDisabled](#) 属性设置为 YES，就可以禁用自动休眠定时器。

由于禁用休眠定时器会导致更大的电能消耗，所以开发者应该尽一切可能避免这样做。只

有地图程序、游戏、以及不依赖于触摸输入而又需要在设备屏幕上显示内容的 应用程序才应该考虑禁用休眠定时器。音频应用程序不需要禁用这个定时器，因为在屏幕变暗之后，音频内容可以继续播放。如果您禁用了定时器，请务必尽快重新 激活它，使系统可以更省电。有关应用程序如何省电的其它贴士，请参见[“减少电力消耗”](#)部分。

应用程序的程序包

当您连编 iPhone 程序时，Xcode 会将它组织为[程序包](#)。**程序包**是文件系统中的 一个目录，用于将执行代码和相关资源集合在一个地方。iPhone 应用程序包中包含应用程序的执行文件 和应用程序需要用到的所有资源（比如应用程序图标、其它图像、和本地化内容）。表1-2 列出了一个典型的 iPhone 应用程序包中的内容（为了便于说明，我们称之为 MyApp）。这个例子只是为了演示，表中列出的一些文件可能并不出现在您自己的应用程序包中。

表1-2 一个典型的 应用程序包文件	描述
MyApp	包含应用程序代码的执行文件，文件名是略去.app 后缀的应用程序名。这个文件是必需的。
Settings.bundle	设置程序包是一个文件包，用于将应用程序的偏好设置加入到 Settings 程序中。这种程序包中包含一些 属性列表 和其它资源文件，用于配置和显示您的偏好设置。更多信息请参见 “显示应用程序的偏好设置” 部分。
Icon.png	这是个57 x 57像素的图标，显示在设备的 Home 屏幕上，代表您的应用程序。这个图标不应该包含任何光亮效果。系统会自动为您加入这些效果。这个文件是必须的。更多有关这个图像文件的信息，请参见 “应用程序图标和启动图像” 部分。
Icon-Settings.png	这是一个29 x 29像素的图标，用于在 Settings 程序中表示您的应用程序。如果您的应用程序包含设置程序包，则在 Settings 程序中，这个图标会显示在您的应用程序名的边上。如果您没有指定这个图标文件，系统会将 Icon.png 文件按比例缩小，然后用做代替文件。有关这个图像文件的更多信息，青参见 “显示应用程序的偏好设置” 部分。
MainWindow.nib	这是应用程序的主 nib 文件 ，包含应用程序启动时装载的缺省用户界面对象。典型情况下，这个 nib 文件包含应用程序的主窗口对象和一个应用程序 委托 对象实例。其它界面对象则或者从其它 nib 文件装载，或者在应用程序中以编程的方式创建（主 nib 文件的名称可以通过 Info.plist 文件中的 NSMainNibFile 键来指定，进一步的信息请参见 “信息属性列表” 部分）。
Default.png	这是个480 x 320像素的图像，在应用程序启动的时候显示。系统使用这个文件作为临时的背景，直到应用程序完成窗口和用户界面的装载。有关这个图像文件的信息请参见 “应用程序图标和启动图像” 部分。
iTunesArtwork	这 是个512 x 512的图标，用于通过 ad-hoc 方式发布的 应用程序。这个图标通常由 App Store 来提供，但是通过 ad-hoc 方式分发的应用程序并不经由 App Store，所以在程序包必须包含这个文件。iTunes 用这个图标来代表您的程序（如果您的应用程序在 App Store 上发布，则在这个属性上指定的文件应该和提交到 App Store 的文件保持一致（通常是个 JPEG 或 PNG 文件），文件名必须和左边显示的一样，而且不带文件扩

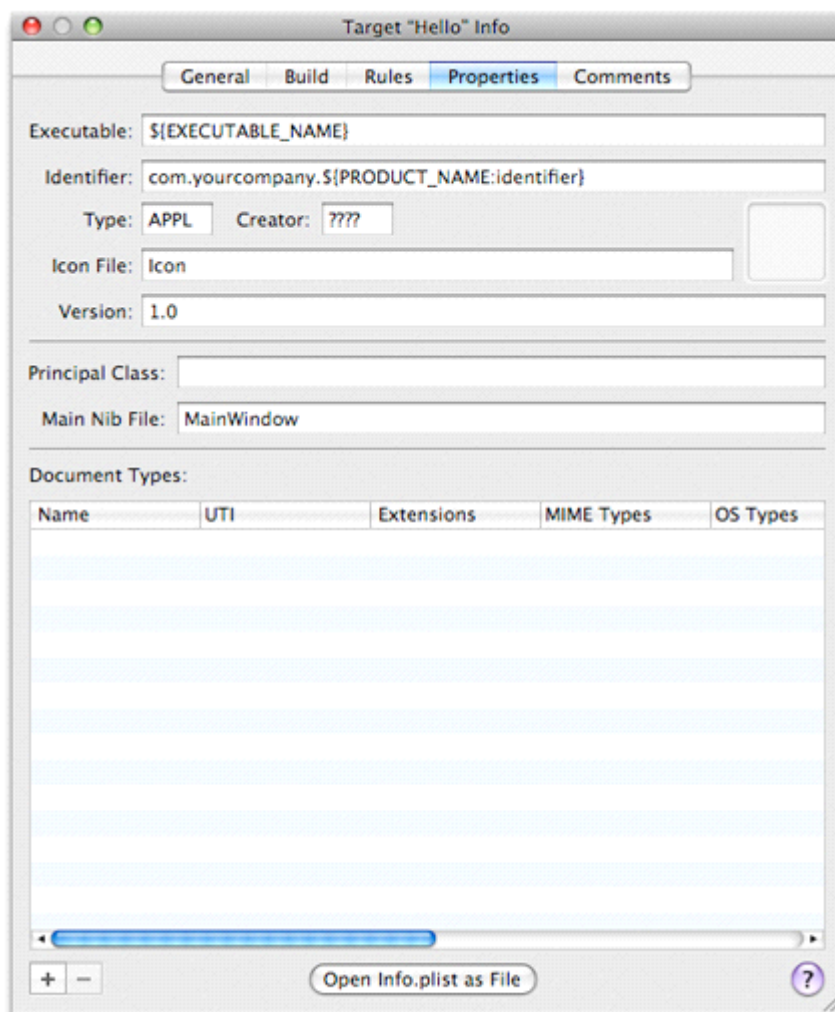
	展名)。
Info.plist	这个文件也叫信息 属性列表 ，它是一个定义应用程序键值的属性列表，比如程序包 ID、版本号、和显示名称。进一步的信息请参见 “信息属性列表” 部分。这个文件是必需的。
sun.png (或其它资源文件)	非本地化资源放在程序包目录的最上层（在这个例子中，sun.png 表示一个非本地化的图像）。应用程序在使用非本地化资源时，不需要考虑用户选择的语言设置。
en.lproj fr.lproj es.lproj 其它具体语言的工程目录	本地化资源 放在一些子目录下，子目录的名称是 ISO 639-1 定义的语言缩写加上.lproj 后缀组成的（比如 en.lproj、fr.lproj、和 es.lproj 目录分别包含英语、法语、和西班牙语的本地化资源）。更多信息请参见 “国际化您的应用程序” 部分。
iPhone 应用程序应该是 国际化 的。程序支持的每一种语言都有一个对应的语言.lproj 文件夹。除了为应用程序提供定制资源的本地化版本之外，您还可以本地化您的应用程序图标（Icon.png）、缺省图像（Default.png）、和 Settings 图标（Icon-Settings.png），只要将同名文件放到具体语言的工程目录就可以了。然而，即使您提供了本地化的版本，也还是应该在应用程序包的最上层包含这些文件的缺省版本。当某些的本地化版本不存在的时候，系统会使用缺省版本。	
您可以通过 NSBundle 类的方法或者与 CFBundleRef 类型相关联的函数来获取应用程序包中本地化和非本地化图形及声音资源的路径。举例来说，如果您希望得到图像文件 sun.png（显示在 “响应中断” 部分中）的路径并通过它创建一个图像文件，则需要下面两行 Objective-C 代码：	
<pre>NSString* imagePath = [[NSBundle mainBundle] pathForResource:@"sun" ofType:@"png"]; UIImage* sunImage = [[UIImage alloc] initWithContentsOfFile:imagePath];</pre>	
代码中的 mainBundle 类方法用于返回一个代表应用程序包的对象。有关资源装载的信息请参见 资源编程指南 。	

信息属性列表

信息[属性列表](#)是一个名为 Info.plist 的文件，通过 Xcode 创建的每个 iPhone 应用程序都包含一个这样的文件。属性列表中的键值对用于指定重要的应用程序运行时配置信息。信息属性列表的元素被组织在一个层次结构中，每个结点都是一个实体，比如数组、字典、字符串、或者其它数值类型。

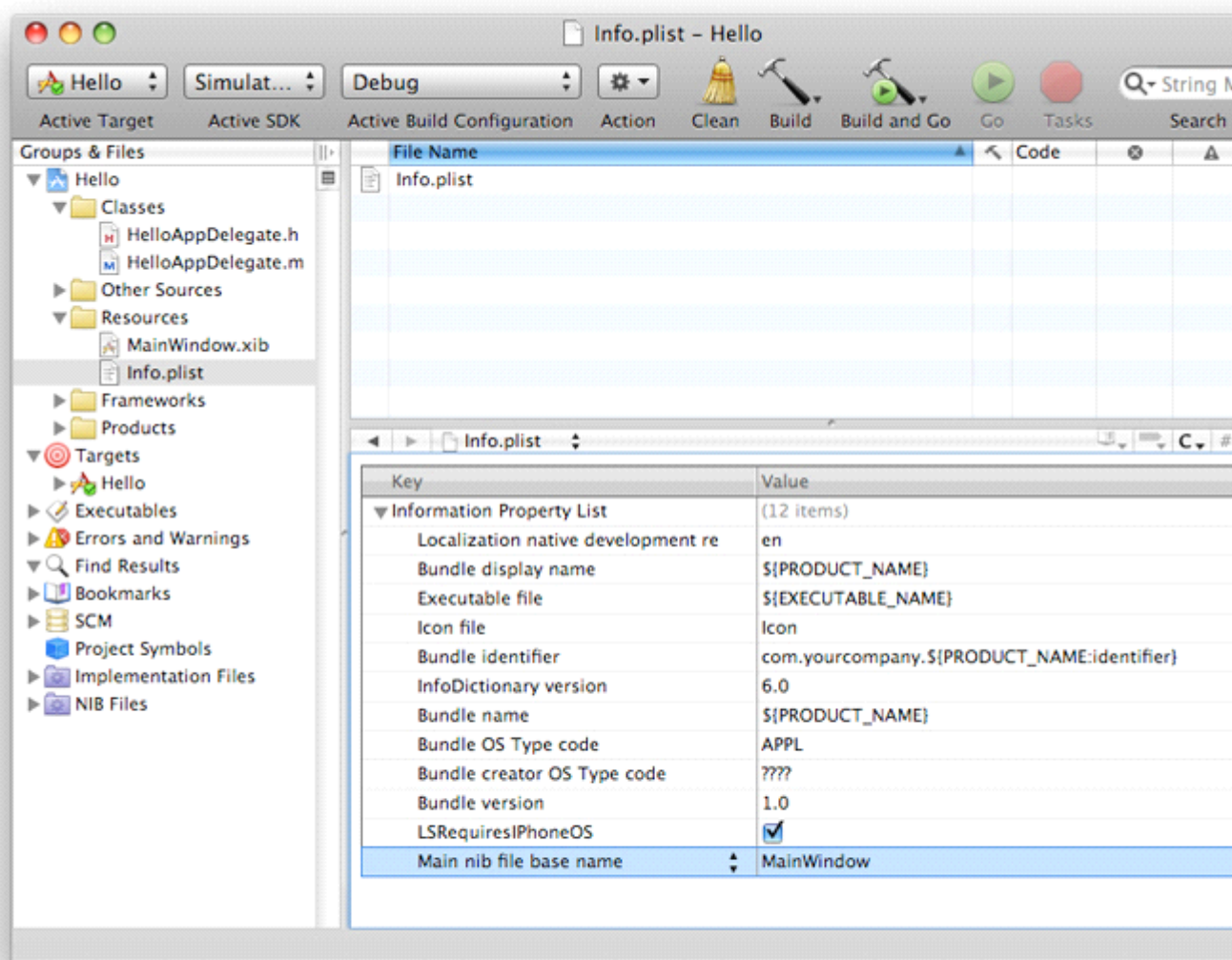
在 Xcode 中，您可以通过在 Project 菜单中选择 Edit Active Target *TargetName* 命令、然后在目标的 Info 窗口中点击 Properties 控件来访问信息属性列表。Xcode 会显示如图1-4所示的信息面板。

图1-4 目标 Info 窗口的属性面板



属性面板显示的是[程序包](#)的一些属性，但并不是所有属性都显示在上面。当您选择“Open Info.plist as File” 按键或在 Xcode 工程中选择 Info.plist 文件时，Xcode 会显示如图1-5所示的属性列表编辑器窗口，您可以通过这个窗口来编辑属性值和添加键-值对。您还可以查看添加到 Info.plist 文件中的实际键名，具体操作是按住 Control 键的同时点击编辑器中的信息属性列表项目，然后选择上下文菜单中的 Show Raw Keys/Values 命令。

图1-5 信息属性列表编辑器



Xcode 会自动设置某些属性的值，其它属性则需要显式设置。表 1-3 列出了一些重要的键，供您使用在自己的 Info.plist 文件中（在缺省情况下，Xcode 不会直接显示实际的键名，因此，下表在括号中列出了这些键在 Xcode 中显示的字符串。您可以查看所有键的实际键名，具体做法是按住 Control 键的同时点击编辑器中的信息属性列表项目，然后选择上下文菜单中的 Show Raw Keys/Values 命令）。有关属性列表文件可以包含的完整属性列表及系统如何使用这些属性的信息，请参见[运行环境配置指南](#)。

表1-3 Info.plist 文件中重要的键值

CFBundleDisplayName (程序包显示名)	显示在应用程序图标下方的名称。这个值应该本地化为所有支持的语言。 这是由您提供的标识字符串，用于在系统中标识您的应用程序。这个字符串必须是一个统一的类型标识符 (UTI)，仅包含字母数字 (A-Z、a-z、0-9)，连字符 (-)，和句号 (.)；且应该使用反向 DNS 格式。举例来说，如果您的公司的域名为 Ajax.com，且您创建的应用程序名为 Hello，则可以将字符串 com.Ajax.Hello 作为应用程序包的标识。
CFBundleIdentifier (程序包标识)	

CFBundleURLTypes (URL 类型)	<p>程序包的标识用于验证应用程序的签名。</p> <p>这是应用程序能够处理的 URL 类型数组。每个 URL 类型都是一个字典，定义一种应用程序能够处理的模式（如 http 或 mailto）。应用程序可以通过这个属性来注册定制的 URL 模式。</p>
CFBundleVersion (程序包版本号)	<p>这是一个字符串，指定程序包的连编版本号。它的值是单调递增的，由一或多个句号分隔的整数组成。这个值不能被本地化。</p>
LSRequiresIPhoneOS	<p>这是一个 Boolean 值，用于指示程序包是否只能运行在 iPhone OS 系统上。Xcode 自动加入这个键，并将它的值设置为 true。您不应该改变这个键的值。</p>
NSMainNibFile (主 nib 文件的名称)	<p>这是一个字符串，指定应用程序主 nib 文件的名称。如果您希望使用其它的 nib 文件（而不是 Xcode 为工程创建的缺省文件）作为主 nib 文件，可以将该 nib 文件名关联到这个键上。nib 文件名不应该包含 .nib 扩展名。</p>
UIStatusBarStyle	<p>这是个字符串，标识程序启动时状态条的风格。这个键的值基于 UIApplication.h 头文件中声明的 UIStatusBarStyle 常量。缺省风格是 UIStatusBarStyleDefault。在启动完成后，应用程序可以改变状态条的初始风格。</p>
UIStatusBarHidden	<p>这个一个 Boolean 值，指定在应用程序启动的最初阶段是否隐藏状态条。将这个键值设置为 true 将隐藏状态条。缺省值为 false。</p>
UIInterfaceOrientation	<p>这是个字符串，标识应用程序用户界面的初始方向。这个键的值基于 UIApplication.h 头文件中声明的 UIInterfaceOrientation 常量。缺省风格是 UIInterfaceOrientationPortrait。</p> <p>有关将应用程序启动为景观模式的更多信息，请参见“以景观模式启动”部分。</p>
UIPrerenderedIcon	<p>这个一个 Boolean 值，指示应用程序图标是否已经包含发光和斜面效果。这个属性缺省值为 false。如果您不希望系统在您的原图上加入这些效果，则将它设置为 true。</p>
UIRequiredDeviceCapabilities	<p>这是个信息键，作用是使 iTunes 和 App Store 知道应用程序运行需要依赖于哪些与设备相关的特性。iTunes 和移动 App Store 程序使用这个列表来避免将应用程序安装到不支持所需特性的设备上。</p> <p>这个键的值可以是一个数组或者字典如果您使用的是数组，则数组中存在某个键就表示该键对应的特性是必需的；如果您使用的是字典，则必须为每个键指定一个 Boolean 值，表示该键是否需要。无论哪种情况，不包含某个键表示该键对应的特性不是必需的。</p> <p>如果您需要可包含在这个字典中的键列表，请参见表1-4。这个键在 iPhone OS 3.0及更高版本上才被支持。</p>
UIRequiresPersistentWiFi	<p>这是个 Boolean 值，用于通知系统应用程序是否使用 Wi-Fi 网络进行通讯。如果您的应用程序需要在一段时间内使用 Wi-Fi，则应该将这个键值设置为 true；否则，为了省电，设备会在30分钟内关闭 Wi-Fi 连接。设置这个标志还可以让系统在 Wi-Fi 网络可用但未被使用的时候显示网络选择对话框。这个键的缺省值是 false。</p>

UISupportedExternalAccessoryProtocols	<p>请注意，当设备处于闲置状态（也就是屏幕被锁定的状态）时，这个属性的值为 <code>true</code> 是没有作用的。这种情况下，应用程序会认为是不活动的，虽然它可能在某些级别上还可以工作，但是没有 Wi-Fi 连接。</p> <p>这是个字符串数组，标识应用程序支持的配件协议。配件协议是应用程序和连接在 iPhone 或 iPod touch 上的第三方硬件进行通讯的协议。系统使用这个键列出的协议来识别当配件连接到设备上时可以打开的应用程序。</p> <p>有关配件和协议的更多信息，请参见“和配件通讯”部分。这个键只在 iPhone OS 3.0和更高版本上支持。</p>
UIViewGroupOpacity	<p>这是个 Boolean 值，用于指示 Core Animation 子层是否继承其超层的不透明特性。这个特性使开发者可以在仿真器上进行更为复杂的渲染，但是对性能会有显著的影响。如果属性列表上没有这个键，则其缺省值为 NO。</p> <p>这个键只在 iPhone OS 3.0和更高版本上支持。</p>
UIViewEdgeAntialiasing	<p>这是个 Boolean 值，用于指示在描画不和像素边界对齐的层时，Core Animation 层是否进行抗锯齿处理。这个特性使开发者可以在仿真器上进行更为复杂的渲染，但是对性能会有显著的影响。如果属性列表上没有这个键，则其缺省值为 NO。</p> <p>这个键只在 iPhone OS 3.0和更高版本上支持。</p>

如果信息属性文件中的属性值是显示在用户界面上的字符串，则应该进行本地化，特别是当 Info.plist 中的字符串值是与本地化语言子目录下 InfoPlist.strings 文件中的字符串相关联的键时。更多信息请参见[“国际化您的应用程序”](#)部分。

表1-4列出了和 UIRequiredDeviceCapabilities 键相关联的数组或字典中可以包含的键。您应该仅包含应用程序确实需要的键。如果应用程序可以通过不执行某些代码路径来适应设备特性不存在的情况，则不需要使用对应的键。

表1-4 UIRequiredDeviceCapabilities 键的字典键	
键	描述
telephony	如果您的应用程序需要 Phone 程序，则包含这个键。如果您的应用程序需要打开 tel 模式的 URL，则可能需要这个特性。
sms	如果您的应用程序需要 Messages 程序，则包含这个键。如果您的应用程序需要打开 sms 模式的 URL，则可能需要这个特性。
still-camera	如果您的应用程序使用 UIImagePickerController 接口来捕捉设备照相机的图像时，需要包含这个键。
auto-focus-camera	如果您的应用程序需要设备照相机的自动对焦能力，则需要包含这个键。虽然大多数开发者应该不需要，但是如果您的应用程序支持微距摄影，或者需要更高锐度的图像以进行某种处理，则可能需要包含这个键。
video-camera	如果您的应用程序使用 UIImagePickerController 接口来捕捉设备摄像机的视频时，需要包含这个键。
wifi	当您的应用程序需要设备的网络特性时，包含这个键。
accelerometer	如果您的应用程序使用 UIAccelerometer 接口来接收加速计事件，则需要包含

r	这个键。如果您的程序仅需要检测设备的方向变化，则不需要。
location-services	如果您的应用程序使用 Core Location 框架来访问设备的当前位置，则需要包含这个键（这个键指的是一般的位置服务特性。如果您需要 GPS 级别的精度，则还应该包含 gps 键）。
gps	如果您的应用程序需要 GPS（或者 AGPS）硬件，以获得更高精度的位置信息，则包含这个键。如果您包含了这个键，就应该同时包含 location-services 键。如果您的程序需要更高精度的位置数据，而不是由蜂窝网络或 Wi-fi 信号提供的数据，则应该要求只接收 GPS 数据。
magnetometer	如果您的应用程序使用 Core Location 框架接收与方向有关的事件时，则需要包含这个键。
microphone	如果您的应用程序需要使用内置的麦克风或支持提供麦克风的外设，则包含这个键。
opengles-1	如果您的应用程序需要使用 OpenGL ES 1.1 接口，则包含这个键。
opengles-2	如果您的应用程序需要使用 OpenGL ES 2.0 接口，则包含这个键。

应用程序图标和启动图像

显示在用户 Home 屏幕上的图标文件的缺省文件名为 `Icon.png`（虽然通过 `Info.plist` 文件中的 `CFBundleIconFile` 属性可以进行重命名）。它应该是一个位于[程序包](#)最上层目录的 PNG 文件。应用程序图标应该是一个 57 x 57 像素的图像，不带任何刨光和圆角斜面效果。典型情况下，系统在显示之前会将这些效果应用到图标上。然而，在应用程序的 `Info.plist` 文件中加入 `UIPrerenderedIcon` 键可以重载这个行为，更多信息请参见[表1-3](#)。

请注意：如果您以 ad-hoc 的方式（而不是通过 App Store）将应用程序发布给本地用户，则程序包中还应该包含一个 512 x 512 像素版本的应用程序图标，命名为 `iTunesArtwork`。在分发您的应用程序时，iTunes 需要显示这个文件提供的图标。

应用程序的启动图像文件的文件名为 `Default.png`。这个图像应该和应用程序的初始界面比较相似；系统在应用程序准备好显示用户界面之前显示启动文件，使用户觉得启动速度很快。启动图像也应该是 PNG 图像文件，位于应用程序包的顶层目录。如果应用程序是通过 URL 启动的，则系统会寻找名为 `Default-scheme.png` 的启动文件，其中 *scheme* 是 URL 的模式。如果该文件不存在，才选择 `Default.png` 文件。

将一个图像文件加入到 Xcode 工程的具体做法是从 Project 菜单中选择 Add to Project 命令，在浏览器中定位目标文件，然后点击 Add 按钮。

请注意：除了程序包顶层目录中的图标和启动图像，您还可以在应用程序中[具体语言的工程子目录](#)下包含这些图像文件的本地化版本。更多有关应用程序本地化资源的信息请参见[“国际化您的应用程序”](#)部分。

Nib 文件

[nib 文件](#)是一种数据文件，用于存储可在应用程序需要时使用的一些“冻结”的对象。大多数情况下，应用程序使用 nib 文件来存储构成用户界面的窗口和视图。当您把 nib 文件载入应用程序时，nib 装载代码会将文件中的内容转化为应用程序可以操作的真正对象。通过这个机制，nib 文件省去了用代码创建那些对象的工作。

Interface Builder 是一个可视化的设计环境，您可以用它来创建 nib 文件。您可以将标准对象（比如 UIKit 框架中提供的窗口和视图）和 Xcode 工程中的定制对象放到 nib 文件中。在

Interface Builder 中创建视图层次相当简单，只需要对视图对象进行简单拖拽就可以了。您也可以通过查看器窗口来配置每个对象的属性，以及通过创建对象间的连接 来定义它们在运行时的关系。您所做的改变最终都会作为 nib 文件的一部分存储到磁盘上。

在运行时，当您需要 nib 文件中包含的对象时，就将 nib 文件装载到程序中。典型情况下，装载 nib 文件的时机是当用户界面发生变化和需要在屏幕上显示某些新视图的时候。如果您的应用程序使用视图控制器，则视图控制器会自动处理 nib 文件的装载过程，当然，您可以通过 [NSBundle](#) 类的方法自行装载。

有关如何设计应用程序用户界面的更多信息，请参见 [iPhone 用户界面指南](#)。有关如何创建 nib 文件的信息则参见 [Interface Builder 用户指南](#)。

处理关键的应用程序任务

本部分将描述几个所有 iPhone 应用程序都应该处理的任务。这些任务是整个应用程序生命周期的一部分，因此也是将应用程序集成到 iPhone OS 系统的重要方面。在最坏的情况下，没有很好地处理其中的某些任务甚至可能会导致应用程序被操作系统终止。

初始化和终止

在初始化和终止过程中，[UIApplication](#) 类会向应用程序的[委托](#)发送恰当的消息，使其执行必要的任务。虽然系统并不要求您的应用程序响应这些消息，但是，几乎所有的 iPhone 应用程序都应该处理这些消息。初始化是您为 应用程序准备用户界面及使其进入初始运行状态的阶段。类似地，在终止阶段，您应该把未保存的数据和关键的应用程序状态写入磁盘。由于一个 iPhone 应用程序必须在其它应用程序启动之前退出，所以花在初始化和终止阶段的执行时间要尽可能少。初始化阶段并不适合装载大的、却又不需要马上使用的数据结构。在开始阶段，您的目标应该是尽可能快地显示应用程序的用户界面，最好是使它进入最后一次退出的状态。如果您的应用程序在启动过程中需要 更多的时间来装载网络数据，或者执行一些可能很慢的任务，则应该首先显示出用户界面并运行起来，然后在后台线程中执行速度慢的任务。这样，您就有机会向用 户显示进度条和其它反馈信息，指示应用程序正在装载必要的数 据，或者正在执行重要的任务。

表1-5列举出 [UIApplicationDelegate](#) 协议定义的方法，您在应用程序委托中需要实现这些协议方法，以处理初始化和终止的事务。表中还列出了您在每个方法中应该执行的关键事务。

表1-5 应用程序委托的责任委托方法	描述
applicationDidFinishLaunching:	使用这个方法将应用程序恢复到上一个会话的状态。您也可以在这个方法中执行应用程序数据结构和用户界面的定制初始化。
applicationWillTerminate:	使用这个方法将未存数据或关键的应用程序状态存入磁盘。您也可以在这个方法中执行额外的清理工作，比如删除临时文件。

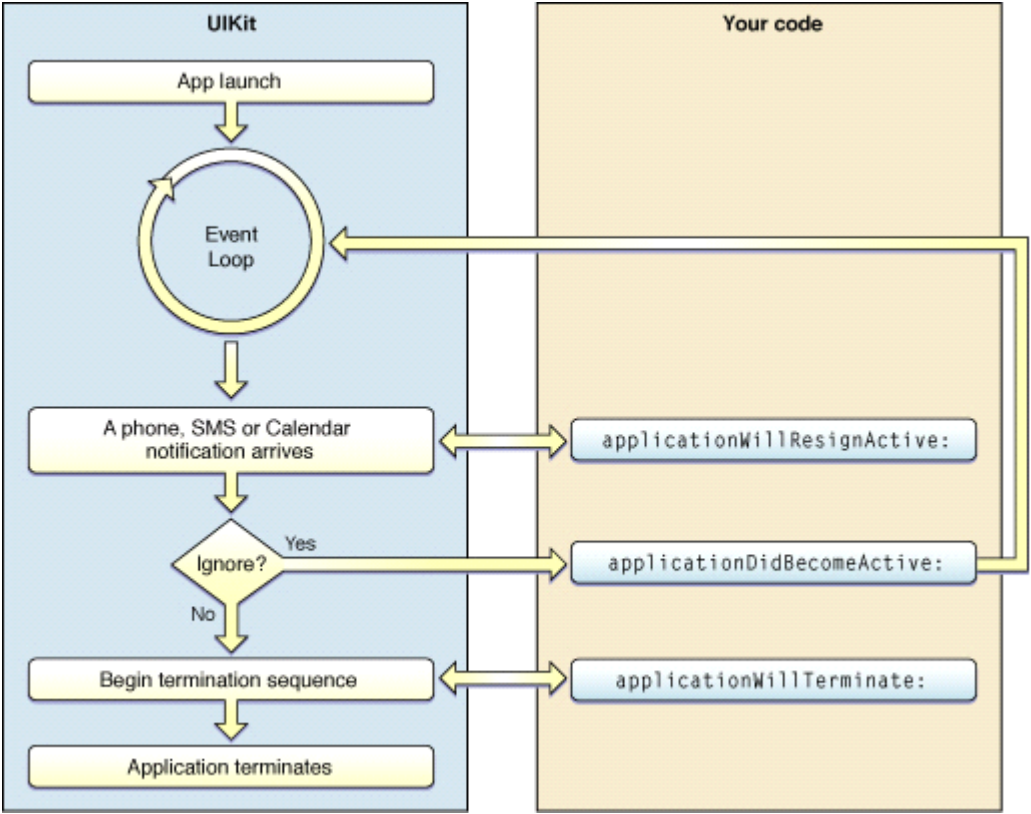
响应中断

除了 Home 按键可以终止您的应用程序之外，系统也可以暂时中断您的应用程序，使用户得以响应一些重要的事件。举例来说，应用程序可能被呼入的电话、SMS 信息、日历警告、

或者设备上的 Sleep 按键所打断。按下 Home 按键会终止您的应用程序，而上述这些中断则只是暂时的。如果用户忽略这些中断，您的应用程序可以象之前那样继续运行；然而，如果用户决定接电话或回应 SMS 信息，系统就会开始终止您的程序。

图1-6显示了在电话、SMS 信息、或者日历警告到来时发生的事件序列。紧接在图后面的步骤说明更为详细地描述了事件序列的关键点，包括您在响应每个事件时应该做的事项。这个序列并不反映当用户按下 Sleep/Wake 按键时发生的情景；该场景的事件序列在步骤说明之后的部分进行描述。

图1-6 中断过程的事件流程



系统检测到有电话、SMS 信息、或者日历警告发生。

系统调用应用程序委托的 [applicationWillResignActive:](#) 方法，同时禁止将触摸事件发送给您的应用程序。

中断会导致应用程序暂时失去控制权。如果控制权的丢失会影响程序的行为或导致不好的用户体验，您就应该在委托方法中采取恰当的步骤进行规避。举例来说，如果您的程序是个游戏，就应该暂停。您还应该禁用定时器、降低 OpenGL 的帧率（如果正在使用 OpenGL 的话），通常还应该使应用程序进行休眠状态。在这休眠状态下，您的应用程序继续运行，但是不应该做任何重要的工作。

系统显示一个带有事件信息的警告窗口。用户可以选择忽略或响应该事件。

- 1 如果用户忽略该事件，系统就调用应用程序委托的 [applicationDidBecomeActive:](#) 方法，并重新开始向应用程序传递触摸事件。

您可以在这个方法中重新激活定时器、提高 OpenGL 的帧率、以及将应用程序从休眠状态唤醒。对于处于暂停状态的游戏，您应该考虑使它停在当时的状态上，等待用户做好重新玩的准备。举例来说，您可以显示一个警告窗口，而窗口中带有重新开始的控制件。

如果用户选择响应该事件（而不是忽略），则系统会调用应用程序委托的 [applicationWillTerminate:](#) 方法。您的应用程序应该正常终止，保存所有必要的上下文信息，

使应用程序在下一次启动的时候可以回到同样的位置。

在您的应用程序终止之后，系统就开始启动负责中断的应用程序。

根据用户对中断的不同响应，系统可能在中断结束之后再次启动您的应用程序。举例来说，如果用户接听一个电话并在完成后挂断，则系统会重新启动您的应用程序；如果用户在接听电话过程中回到 Home 屏幕或启动另一个程序，则系统就不再启动您的应用程序了。

重要提示：当 用户接听电话并在通话过程中重新启动您的应用程序时，状态条的高度会变大，以反映当前用户正在通话中。类似地，当用户结束通话的时候，状态条的高度会缩回 正常尺寸。您的应用程序应该为状态条高度的变化做好准备，并据此调整内容区域的尺寸。视图控制器会自动处理这个行为，然而，如果您通过代码进行用户界面的 布局，就需要在视图布局以及通过 [layoutSubviews](#) 方法处理动态布局变化时考虑状态条的高度。

在运行您的应用程序时，如果用户按下设备的休眠/唤醒按键，系统会调用应用程序委托的 [applicationWillResignActive:](#)方法，停止触摸事件的派发，然后使设备进入休眠状态。之后，当用户唤醒设备时，系统会调用应用程序委托的 [applicationDidBecomeActive:](#)方法，并再次开始向应用程序派发事件。如同处理其它中断一样，您应该使用这些方法来使应用程序进入休眠状态（或者暂停游戏）及再次唤醒它们。在休眠时，您的应用程序应该尽可能少用电力。

观察低内存警告

当系统向您的应用程序发送低内存警告时，您需要加以注意。当可用内存的数量降低到安全阈值以下时，iPhone OS 会通知最前面的应用程序。如果您的应用程序收到这种警告，就必须尽可能多地释放内存，即释放不再需要的对象或清理易于在稍后进行重建的缓存。

UIKit 提供如下几种接收低内存警告的方法：

在应用程序委托中实现 [applicationDidReceiveMemoryWarning:](#)方法。

在您的 UIViewController 子类中实现 [didReceiveMemoryWarning](#) 方法。

注册 [UIApplicationDidReceiveMemoryWarningNotification](#) 通告。

一旦收到上述的任何警告，您的处理代码就应该立即响应，释放所有不需要的内存。视图控制器应该清除当前离屏的视图对象，您的应用程序委托则应该释放尽可能多的数据结构，或者通知其它应用程序对象释放其拥有的内存。

如果您的定制对象知道一些可清理的资源，则可以让该对象注册 [UIApplicationDidReceiveMemoryWarningNotification](#) 通告，并在通告处理器代码中直接释放那些资源。如果您通过少数对象来管理大多数可清理的资源，且适合清理所有的这些资源，则同样可以让这些对象进行注册。但是，如果您有很多可清理的对象，或者仅希望释放这些对象的一个子集，则在您的应用程序委托中进行释放可能更好一些。

重要提示：和 系统的应用程序一样，您的应用程序总是需要处理低内存警告，即使在测试过程中没有收到那些警告，也一样要进行处理。系统在处理请求时会消耗少量的内存。在 检测到低内存的情况时，系统会将低内存警告发送给所有正在运行的进程（包括您的应用程序），而且可能终止某些后台程序（如果必要的话），以减轻内存的压 力。如果释放后内存仍然不够—可能因为您的应用程序发生泄露或消耗太多内存—系统仍然可能会终止您的应用程序。

定制应用程序的行为

有几种方法可以对基本的应用程序行为进行定制，以提供您希望的用户体验。本文的下面部

分将描述一些必须在应用程序级别进行的定制。

以景观模式启动

为了配合 Home 屏幕的方向，iPhone OS 的应用程序通常以肖像模式启动。如果您的应用程序既可以以景观模式运行，也可以以肖像模式运行，那么，一开始应该总是以纵向模式启动，然后由视图控制器根据设备的方向旋转用户界面。但是，如果您的应用程序只能以景观模式启动，则必须执行下面的步骤，使它一开始就以景观模式启动。

在应用程序的 Info.plist 文件中加入 `UIInterfaceOrientation` 键，并将它的值设置为景观模式。您可以将这个键值设置为 [UIInterfaceOrientationLandscapeLeft](#) 或者 [UIInterfaceOrientationLandscapeRight](#)。

以景观模式布局您的视图，并确保正确设置视图的自动尺寸调整选项。

重载视图控制器的 [shouldAutorotateToInterfaceOrientation:](#) 方法，使它仅在期望的景观方向时返回 YES，而在肖像方向时返回 NO。

重要提示：上面描述的步骤假定您的应用程序使用视图控制器来管理视图层次。视图控制器为处理方向改变和复杂的视图相关事件提供了大量的基础设施。如果您的应用程序不使用视图控制器—游戏和其它基于 OpenGL ES 的应用程序可能是这样的—就必须根据需要旋转绘图表面（或者调整绘图命令），以便将您的内容以景观模式展示出来。

`UIInterfaceOrientation` 属性提示 iPhone OS 在启动时应该配置应用程序状态条（如果有的话）的方向，就象配置[视图控制器](#)管理下的视图方向一样。在 iPhone OS 2.1 及更高版本的系统中，视图控制器会尊重这个属性，将视图的初始方向设置为指定的方向。使用这个属性相当于在 `applicationDidFinishLaunching:` 方法的一开始执行 `UIApplication` 的 [setStatusBarOrientation:animated:](#) 方法。

请注意：在 v2.1 之前的 iPhone OS 系统中，如果要以景观模式启动基于视图控制器的应用程序，需要在上文描述的所有步骤的基础上对应用程序根视图的转换矩阵进行一个 90 度的旋转。在 iPhone OS 2.1 之前，视图控制器并不会根据 `UIInterfaceOrientation` 键的值自动进行旋转，当然在 iPhone OS 2.1 及更高版本的系统中不需要这个步骤。

和其它应用程序进行通讯

如果一个应用程序支持一些已知类型的 URL，您就可以通过对应的 URL 模式和该程序进行通讯。然而，在大多数情况下，URL 只是用于简单地启动一个应用程序并显示一些和调用方有关的信息。举例来说，对于一个用于管理地址信息的应用程序，您就可以在发送给它的 URL 中包含一个 Maps 程序可以处理的地址，以便显示相应的位置。这个级别的通讯为用户创造一个集成度高得多的环境，减少应用程序重新实现设备上其它程序已经实现的功能的必要性。

苹果内置支持 `http`、`mailto`、`tel`、和 `sms` 这些 URL 模式，还支持基于 `http` 的、指向 Maps、YouTube、和 iPod 程序的 URL。应用程序也可以自己注册定制的 URL 模式。您的应用程序可以和其它应用程序通讯，具体方法是用正确格式的内容创建一个 [NSURL](#) 对象，然后将它传给共享 `UIApplication` 对象 [openURL:](#) 方法。`openURL:` 方法会启动注册接收该 URL 类型的应用程序，并将 URL 传给它。当用户最终退出该应用程序时，系统通常会重新启动您的应用程序，但并不总是这样。系统会考虑用户在 URL 处理程序中的动作及在用户看来返回您的应用程序是否合理，然后做出决定。

下面的代码片断展示了一个程序如何请求另一个程序提供的服务（假定这个例子中的

“todolist”是由应用程序注册的定制模式)：

```
NSURL *myURL = [NSURL URLWithString:@"todolist://www.acme.com?Quarterly%20Report#200806231300"];
[[UIApplication sharedApplication] openURL:myURL];
```

重要提示：如果您的 URL 类型包含的模式和苹果定义的一样，则启动的是苹果提供的程序，而不是您的程序。如果有多个第三方的应用程序注册处理同样的 URL 模式，则该类型的 URL 由哪个程序处理是没有定义的。

如果您的应用程序定义了自己的 URL 模式，则应该实现对该模式进行处理的方法，具体信息在[“实现定制的 URL 模式”](#)部分中进行描述。有关系统支持的 URL 处理，包括如何处理 URL 的格式，请参见[苹果的 URL 模式参考](#)。

实现定制的 URL 模式

您可以为自己的应用程序注册包含定制模式的 URL 类型。定制的 URL 模式是第三方应用程序和其它程序及系统进行交互的机制。通过定制的 URL 模式，应用程序可以将自己的服务提供给其它程序。

注册定制的 URL 模式

在为您的应用程序注册 URL 类型时，必须指定 CFBundleURLTypes 属性的子属性，我们已经在[“信息属性列表”](#)部分中介绍过这个属性了。CFBundleURLTypes 属性是应用程序的 Info.plist 文件中的一个字典数组，每个字典负责定义一个应用程序支持的 URL 类型。表1-6 描述了 CFBundleURLTypes 字典的键和值。

表1-6	
CFBundleURLTypes 属性的键和值	值
CFBundleURLName	这是个字符串，表示 URL 类型的抽象名。为了确保其唯一性，建议您使用反向 DNS 风格的标识，比如 com.acme.myscheme。
CFBundleURLSchemes	这里提供的 URL 类型名是一个指向本地化字符串的键，该字符串位于本地化语言包子目录中的 InfoPlist.strings 文件中。本地化字符串是人类可识别的 URL 类型名称，用相应的语言来表示。
	这是个 URL 模式的数组，表示归属于这个 URL 类型的 URL。每个模式都是一个字符串。属于指定 URL 类型的 URL 都带有它们的模式组件。

图1-7显示了一个正在用内置的 Xcode 编辑器编辑的 Info.plist 文件。在这个图中，左列中的 URL 类型入口相当于您直接加入到 Info.plist 文件的 CFBundleURLTypes 键。类似地，“URL identifier”和“URL Schemes”入口相当于 CFBundleURLName 和 CFBundleURLSchemes 键。

图1-7 在 Info.plist 文件中定义一个定制的 URL 模式

Key	Value
▼ Information Property List	(12 items)
Localization native develo	en
Bundle display name	#{PRODUCT_NAME}
Executable file	#{EXECUTABLE_NAME}
Icon file	
Bundle identifier	com.acme.#{PRODUCT_NAME}
InfoDictionary version	6.0
Bundle name	#{PRODUCT_NAME}
Bundle OS Type code	APPL
Bundle creator OS Type co	????
Bundle version	1.0
Main nib file base name	MainWindow
▼ URL types	(1 item)
▼ Item 1	(2 items)
URL identifier	com.acme.ToDoList
▼ URL Schemes	(1 item)
Item 1	todolist

您在对 CFBundleURLTypes 属性进行定义，从而注册带有定制模式的 URL 类型之后，可以通过下面的方式来进行测试：

连编、安装、和运行您的应用程序。

回到 Home 屏幕，启动 Safari（在 iPhone 仿真器上，在菜单上选择 Hardware > Home 命令就可以回到 Home 屏幕）。

在 Safari 的地址栏中，键入使用定制模式的 URL。

确认您的应用程序是否启动，以及应用程序委托是否收到 [application:handleOpenURL:](#) 消息。

处理 URL 请求

应用程序委托在 [application:handleOpenURL:](#) 方法中处理传递给应用程序的 URL 请求。如果您已经为自己的应用程序注册了定制的 URL 模式，则务必在 [委托](#) 中实现这个方法。

基于定制模式的 URL 采用的协议是请求服务的应用程序能够理解的。URL 中包含一些注册模式的应用程序期望得到的信息，这些信息是该程序在处理或响应 URL 请求时需要的。传递给 [application:handleOpenURL:](#) 方法的 [NSURL](#) 对象表示的是 Cocoa Touch 框架中的 URL。NSURL 遵循 RFC 1808 规范，该类中包含一些方法，用于返回 RFC 1808 定义的各个 URL 要素，包括用户名、密码、请求、片断、和参数字符串。与您注册的定制模式相对应的“协议”可以使用这些 URL 要素来传递各种信息。

在程序清单 1-2 显示的 [application:handleOpenURL:](#) 方法实现中，传入的 URL 对象在其请求和片断部分带有具体应用程序的信息。应用程序委托抽出这些信息——在这个例子中，是指一个 to-do 任务的名称和到期日——并根据这些信息创建应用程序的模型对象。

程序清单 1-2 处理基于定制模式的 URL 请求

```
- (BOOL)application:(UIApplication *)application handleOpenURL:(NSURL *)url {
    if ([[url scheme] isEqualToString:@"todolist"]) {
        ToDoItem *item = [[ToDoItem alloc] init];
```

```

NSString *taskName = [url query];
if (!taskName || ![self isValidTaskString:taskName]) { // must have a task name
    [item release];
    return NO;
}
taskName = [taskName
stringByReplacingPercentEscapesUsingEncoding:NSUTF8StringEncoding];

```

```

item.toDoTask = taskName;
NSString *dateString = [url fragment];
if (!dateString || [dateString isEqualToString:@"today"]) {
    item.dateDue = [NSDate date];
} else {
    if (![self isValidDateString:dateString]) {
        [item release];
        return NO;
    }
    // format: yyymmddhhmm (24-hour clock)
    NSString *curStr = [dateString substringWithRange:NSMakeRange(0, 4)];
    NSInteger yeardigit = [curStr integerValue];
    curStr = [dateString substringWithRange:NSMakeRange(4, 2)];
    NSInteger monthdigit = [curStr integerValue];
    curStr = [dateString substringWithRange:NSMakeRange(6, 2)];
    NSInteger daydigit = [curStr integerValue];
    curStr = [dateString substringWithRange:NSMakeRange(8, 2)];
    NSInteger hourdigit = [curStr integerValue];
    curStr = [dateString substringWithRange:NSMakeRange(10, 2)];
    NSInteger minutedigit = [curStr integerValue];

    NSDateComponents *dateComps = [[NSDateComponents alloc] init];
    [dateComps setYear:yeardigit];
    [dateComps setMonth:monthdigit];
    [dateComps setDay:daydigit];
    [dateComps setHour:hourdigit];
    [dateComps setMinute:minutedigit];
    NSCalendar *calendar = [NSCalendar currentCalendar];
    NSDate *itemDate = [calendar dateFromComponents:dateComps];
    if (!itemDate) {
        [dateComps release];
        [item release];
        return NO;
    }
    item.dateDue = itemDate;
    [dateComps release];
}

```

```

    }

    [(NSMutableArray *)self.list addObject:item];
    [item release];
    return YES;
}
return NO;
}

```

请务必对传入的 URL 输入进行验证。如果您希望了解如何避免 URL 处理的相关问题，请参见[安全编码指南](#)文档中的[验证输入](#)部分。如果要了解苹果定义的 URL 模式，请参见[苹果的 URL 模式参考](#)。

显示应用程序的偏好设置

如果您的应用程序通过偏好设置来控制其行为的不同方面，那么，以何种方式向用户提供偏好设置就取决于它们是否为程序的必需部分。

如果偏好设置是程序使用的必需部分（且直接实现起来足够简单），那么应该直接通过应用程序的定制界面来呈现。

如果偏好设置不是必需的，且要求相对复杂的界面，则应该通过系统的 **Settings** 程序来呈现。在 确定一组偏好设置是否为程序的必需部分时，请考虑您为程序设计的使用模式。如果您希望用户相对频繁地修改偏好设置，或者这些偏好设置对程序的行为具有相对 重要的影响，则可能就是必需部分。举例来说，游戏中的设置通常都是玩游戏的必需部分，或者是用户希望快速改变的项目。然而，由于 **Settings** 程序是一个独立的程序，所以只能用于处理用户不频繁访问的偏好设置。

如果您选择在应用程序内进行偏好设置管理，则可以自行定义用户界面及编写代码来实现。但是，如果您选择使用 **Settings** 程序，则必须提供一个设置包（**Settings Bundle**）来进行管理。

设置包是位于应用程序的程序包目录最顶层的定制资源，它是一个封装了的目录，名字为 **Settings.bundle**。设置包中包含一些具有特别格式的数据文件（及其支持资源），其作用是告诉 **Settings** 程序如何显示您的偏好设置。这些文件还告诉 **Settings** 程序应该把结果值存储在偏好设置数据库的什么位置上，以便应用程序随后可以通过 [NSUserDefaults](#) 或 [CFPreferences API](#) 来进行访问。

如果您通过设置包来实现偏好设置管理，则还应该提供一个定制的图标。**Settings** 程序会在您的应用程序包的最顶层寻找名为 **Icon-Settings.png** 的图像文件，并将该图像显示在应用程序名称的边上。该文件应该是一个 29 x 29 像素的 PNG 图像文件。如果您没有在应用程序包的最顶层提供这个文件，则 **Settings** 程序会缺省使用缩放后的应用程序图标（**Icon.png**）。

有关如何为应用程序创建设置包的更多信息，请参见[“应用程序的偏好设置”](#)部分。

关闭屏幕锁定

如 果一个基于 iPhone OS 的设备在某个特定时间段中没有接收到触摸事件，就会关闭屏幕，并禁用触摸传感器。以这种方式锁定屏幕是省电的重要方法。因此，除非您确实需要在应用程序中避免无意的行为，否则应该总是打开屏幕锁定功能。举例来说，如果您的应用程序不接收屏幕事件，而是使用其它特性（比如加速计）来进行输入，则可能需要 禁用屏幕锁

定功能。

将共享的 UIApplication 对象的 [idleTimerDisabled](#) 属性设置为 YES，就可以禁止屏幕锁定。请务必在程序不需要禁止屏幕锁定功能时将该属性重置为 NO。举例来说，您可能在用户玩游戏的时候禁止了屏幕锁定，但是，当用户处于配置界面或没有处于游戏活跃状态时，应该重新打开这个功能。

国际化您的应用程序

理想情况下，iPhone 应用程序显示给用户的文本、图像、和其它内容都应该本地化为多种语言。比如，警告对话框中显示的文本就应该以用户偏好的语言显示。为工程准备特定语言的本地化内容的过程就称为[国际化](#)。工程中需要本地化的候选组件包括：

代码生成的文本，包括与具体区域设置有关的日期、时间、和数字格式。

静态文本—比如装载到 web 视图、用于显示应用程序帮助的 HTML 文件。

图标（包括您的应用程序图标）及其它包含文本或具体文化意义的图像。

包含发声语言的声音文件。

[Nib 文件](#)。

通过 Settings 程序，用户可以从 Language 偏好设置视图（参见图1-8）中选择希望在用户界面上看到的语言。您可以访问 General 设置，然后在 International 组中找到该视图。

图1-8 语言偏好设置视图



用户选择的语言和[程序包](#)中的一个子目录相关联，该子目录名由两个部分组成，分别是 ISO 639-1定义的语言码和.lproj 后缀。您还可以对语言码进行修改，使之包含具体的地区，方法是在后面（在下划线之后）加入 ISO 3166-1定义的区域指示符。举例来说，如果要指定美国英语的本地化资源，程序包中的子目录应该命名为 en_US.lproj。我们约定，本地化语言子目录称为 lproj 文件夹。

请注意：您也可以使用 ISO 639-2语言码，而不一定使用 ISO 639-1的定义。有关语言和区域

代码的信息，请参见[国际化编程主题](#)文档中的“语言和地域的指定”部分。

一个 lproj 文件夹中包含所有指定语言（还可能包含指定地区）的本地化内容。您可以用 [NSBundle](#) 类或 [CFBundleRef](#) 封装类型提供的工具来（在应用程序的 lproj 文件夹）定位当前选定语言的本地化资源。列表1-3给出一个包含英语（en）本地化内容的目录。

列表1-3 本地化语言子目录的内容

en.lproj/

InfoPlist.strings

Localizable.strings

sign.png

这个例子目录有下面几个项目：

InfoPlist.strings 文件，包含与 Info.plist 文件中特定键（比如 CFBundleDisplayName）相关联的本地化字符串值。比如，一个英文名称为 Battleship 的应用程序，其 CFBundleDisplayName 键在 fr.lproj 子目录的 InfoPlist.strings 文件中有如下的入口：

```
CFBundleDisplayName = "Cuirassé";
```

Localizable.strings 文件，包含应用程序代码生成的字符串的本地化版本。

本例子中的 sign.png，是一个包含本地化图像的文件。

为了本地化，我们需要国际化代码中的字符串，具体做法是用 [NSLocalizedString](#) 宏来代替字符串。这个宏的定义如下：

```
NSString *NSLocalizedString(NSString *key, NSString *comment);
```

第一个参数是一个唯一的键，指向给定 lproj 文件夹中 Localizable.strings 文件里的一个本地化字符串；第二个参数是一个注释，说明字符串如何使用，因此可以为翻译人员提供额外的上下文。举例来说，假定您正在设置用户界面中一个标签（UILabel 对象）的内容，则下面的代码可以国际化该标签的文本：

```
label.text = NSLocalizedString(@"City", @"Label for City text field");
```

然后，您就可以为给定语言创建一个 Localizable.strings 文件，并将它加入到相应的 lproj 文件夹中。对于上文例子中的键，该文件中应该有如下入口：

```
"City" = "Ville";
```

请注意：另一种方法是在代码中恰当的地方插入 NSLocalizedString 调用，然后运行 genstrings 命令行工具。该工具会生成一个 Localizable.strings 文件的模板，包含每个需要翻译的键和注释。更多有关 genstrings 的信息，请参见 [genstrings\(1\)](#) 的 man 页面。

更多有关国际化的信息，请参见[国际化编程主题](#)。

性能和响应速度的调优

在应用程序开发过程的每一步，您都应该考虑自己所做的设计对应用程序总体性能的影响。由于 iPhone 和 iPod touch 设备的移动本质，iPhone 应用程序的操作环境受到更多的限制。本文的下面部分将描述在开发过程中应该考虑哪些因素。

不要阻塞主线程

您 应该认真考虑在应用程序主线程上执行的任务。主线程是应用程序处理触摸事件和其它

用户输入的地方。为了确保应用程序总是可以响应用户，我们不应该在主线程 中执行运行时间很长或可能无限等待的任务，比如访问网络的任务。相反，您应该将这些任务放在后台线程。一个推荐的方法是将每个任务都封装在一个操作对象 中，然后加入操作队列。当然，您也可以自己创建显式的线程。

将任务转移到后台可以使您的主线程继续处理用户输入，这对于应 用程序的启动和退出尤其重要。在这些时候，系统期望您的应用程序及时响应事件。如果应用程序的主线程在启动过程中被阻塞住了，系统甚至可能在启动完成之前 将它杀死；如果主线程在退出时被阻塞了，则应用程序可能来不及保存关键用户数据就被杀死了。

更多有关如何使用操作对象和线程的信息，请参见[线程编程指南](#)。

有效地使用内存

由于 iPhone OS 的虚存模型并不包含磁盘交换区空间，所以应用程序在更大程度上受限于可供使用的内存。对内存的大量使用会严重降低系统的性能，可能导致应用程序被终止。因此，在设计阶段，您应该把减少应用程序的内存开销放在较高优先级上。

应 用程序的可用内存和相对性能之间有直接的联系。可用内存越少，系统在处理未来的内存请求时就越可能出问题。如果发生这种情况，系统总是先把代码页和其它非 易失性资源从内存中移除。但是，这可能只是暂时的修复，特别是当系统在短时间后又再次需要那些资源的时候。相反，您需要尽可能使内存开销最小化，并及时清 除自己使用的内存。

本文的下面部分将就如何有效使用内存和在只有少量内存时如何反应方面提供更多的指导。

减少应用程序的内存印迹

表1-7列出一些如何减少应用程序总体内存印迹的技巧。在开始时将内存印迹降低了，随后就可以有更多的空间用于需要操作的数据。

表1-7 减少应用程序内存印迹的技巧

采取的措施	技巧
消除内存泄露	由于内存是 iPhone OS 的关键资源，所以您的应用程序不应该有任何的内存泄露。存在内存泄露意味着应用程序在之后可能没有足够的内存。您可以用 Instruments 程序来 跟踪代码中的泄露，该程序既可以用于仿真器，也可以用于实际的设备。有关如何使用 Instruments 的更多信息，请参见 Instruments 用户指南 。
使资源文件尽可能小	文件驻留在磁盘中，但在使用时需要载入内存。 属性列表 文件和图像文件是通过简单的处理就可以节省空间的两种资源类型。您可以通过 NSPropertyListSerialization 类将属性列表文件存储为二进制格式，从而减少它们的使用空间；对于图像，可以将所有图像文件压缩得尽可能小(PNG 图像是 iPhone 应用程序的推荐图像格式，可以用 pngcrush 工具来进行压缩)。
使用 Core Data 或 SQLite 来处理大的数据集	如果您的应用程序需要操作大量的结构化数据，请将它存储在 Core Data 的持久存储或 SQLite 数据库，而不是使用扁平文件。Core Data 和 SQLite 都提供了管理大量数据的有效方

延缓装载资源	<p>法，不需要将整个数据一次性地载入内存。</p> <p>Core Data 的支持是在 iPhone OS 3.0系统上引入的。</p> <p>在真正需要资源文件之前，永远不应该进行装载。预先载入资源文件表面看好象可以节省时间，但实际上会使应用程序很快变慢。此外，如果您最终没有用到那些资源，预先载入将只是浪费内存。</p>
将程序连编为 Thumb 格式	<p>加入-mthumb 开关可以将代码的尺寸减少最多达35%。但是，对于具有大量浮点数运算的代码模块，请务必将这个选项关闭，因为对那样的模块使用 Thumb 反而会导致性能的下降。</p>

恰当地分配内存

iPhone 应用程序使用[委托内存模式](#)，因此，您必须显式保持和释放内存。表1-8列出了一些在程序中分配内存的技巧。

表1-8 分配内存的技巧	采取的措施
减少自动释放对象的使用	<p>通过 autorelease 方法释放的对象会留在内存中，直到显式清理自动释放池或者程序再次回到事件循环。在任何可能的时候，请避免使用 autorelease 方法，而是通过 release 方法立即收回对象占用的空间。如果您必须创建一定数量的自动释放对象，则请创建局部的自动释放池，以便在返回事件循环之前定期对其进行清理，回收那些对象的内存。</p>
为资源设置尺寸限制	<p>避免装载大的资源文件，如果有更小的文件可用的话。请用适合于 iPhone OS 设备的恰当尺寸图像来代替高清晰度的图像。如果您必须使用大的资源文件，需要考虑仅装载当前需要的部分。举例来说，您可以通过 <code>mmap</code> 和 <code>munmap</code> 函数来将文件的一部分载入内存或从内存卸载，而不是操作整个文件。有关如何将文件映射到内存的更多信息，请参见文件系统性能指南。</p>
避免无边界的问题集	<p>无边界的问题集可能需要计算任意大量的数据。如果该集合需要的内存比当前系统能提供的还要多，则您的应用程序可能无法进行计算。您的应用程序应该尽可能避免处理这样的集合，而将它们转化为内存使用极限已知的问题。</p>
有关如何在 iPhone 应用程序中 分配内存 及使用自动释放池的详细信息，请参见 Cocoa 基本原理指南 文档的 Cocoa 对象 部分。	

浮点数学运算的考虑

iPhone-OS 设备上的处理器有能力在硬件上处理浮点数计算。如果您目前的程序使用基于软件的定点数数学库进行计算，则应该考虑对代码进行修改，转向使用浮点数数学库。典型情况下，基于硬件的浮点数计算比对应的基于软件的定点数计算快得多。

重要提示：当然，如果您的代码确实广泛地使用浮点数计算，请记住不要使用-mthumb 选项来编译代码。Thumb 选项可以减少代码模块的尺寸，但是也会降低浮点计算代码的性能。

减少电力消耗

移动设备的电力消耗一直是个问题。iPhone OS 的电能管理系统保持电能的方法是关闭当前未被使用的硬件功能。此外，要避免 CPU 密集型和高图形帧率的操作。您可以通过优化如下组件的使用来提高电池的寿命：

CPU

Wi-Fi 和基带(EDGE, 3G)无线信号

Core Location 框架

加速计

磁盘

您的优化目标应该是以尽可能有效的方式完成大多数的工作。您应该总是采用 Instruments 和 Shark 工具对应用程序的算法进行优化。但是，很重要的一点是，即使最优化的算法也可能对设备的电池寿命造成负面的影响。因此，在写代码的时候应该考虑如下的原则：

避免需要轮询的工作，因为轮询会阻止 CPU 进入休眠状态。您可以通过 [NSRunLoop](#) 或者 [NSTimer](#) 类来规划需要做的工作，而不是使用轮询。

尽一切可能使共享的 [UIApplication](#) 对象的 [idleTimerDisabled](#) 属性值保持为 NO。当设备处于不活动状态一段时间后，空闲定时器会关闭设备的屏幕。如果您的应用程序不需要设备屏幕保持打开状态，就让系统将它关闭。如果关闭屏幕给您的应用程序的体验带来负面影响，则需要通过修改代码来消除那些影响，而不是不必要地关闭空闲定时器。

尽可能将任务合并在一起，以便使空闲时间最大化。每隔一段时间就间歇性地执行部分任务比一次性完成相同数量的所有任务开销更多的电能。间歇性地执行任务会阻止系统在更长时间内无法关闭硬件。

避免过度访问磁盘。举例来说，如果您需要将状态信息保存在磁盘上，则仅当该状态信息发生变化时才进行保存，或者尽可能将状态变化合并保存，以避免短时间频繁进行磁盘写入操作。

不要使屏幕描画速度比实际需求更快。从电能消耗的角度看，描画的开销很大。不要依赖硬件来压制应用程序的帧率，而是应该根据程序实际需要的帧率来进行帧的描画。

如果你通过 [UIAccelerometer](#) 类来接收常规的加速计事件，则当您不再需要那些事件时，要禁止这些事件。类似地，请将事件传送的频率设置为满足应用程序需要的最小值。更多信息请参见“[访问加速计事件](#)”部分。

您向网络传递的数据越多，就需要越多的电能来进行无线发射。事实上，访问网络是您所能进行的最耗电的操作，您应该遵循下面的原则，使网络访问最小化：

仅在需要的时候连接外部网络，不要对服务器进行轮询。

当您需要连接网络时，请仅传递完成工作所需要的最少数据。请使用紧凑的数据格式，不要包含可被简单忽略的额外数据。

尽可能快地以群发 (in burst) 方式传递数据包，而不是拉长数据传输的时间。当系统检测到设备没有活动时，就会关闭 Wi-Fi 和蜂窝无线信号。您的应用程序以较长时间传输数据比以较短时间传输同样数量的数据要消耗更多的电能。

尽可能通过 Wi-Fi 无线信号连接网络。Wi-Fi 耗电比基带无线少，是推荐的方式。

如果您通过 Core Location 框架收集位置数据，则请尽可能快地禁止位置更新，以及将位置过滤器和精度水平设置为恰当的值。Core Location 通过可用的 GPS、蜂窝、和 Wi-Fi 网络来确定用户的位置。虽然 Core Location 已经努力使无线信号的使用最小化了，但是，设置恰当的精度和过滤器的值可以使 Core Location 在不需要位置服务的时候完全关闭硬件。更多信息请参见“[获取用户的当前位置](#)”部分。

代码的优化

和 iPhone OS 一起推出的还有几个应用程序的优化工具。它们中的大部分都运行在 Mac OS X 上，适合于调整运行在仿真器上的代码的某些方面。举例来说，您可以通过仿真器来消除内存泄露，确保总的内存开销尽可能小。借助这些工具，您还可以排除代码中可能由低效算法或已知瓶颈引起的计算热点。

在仿真器上进行代码优化之后，还应该在设备上用 Instruments 程序进行进一步优化。在实际设备上运行代码是对其进行完全优化的唯一方式。因为仿真器运行在 Mac OS X 上，而运行 Mac OS X 的系统具有更快的 CPU 和更多的可用内存，所以其性能通常比实际设备的性能好很多。在实际设备上用 Instruments 跟踪代码可能会发现额外的性能瓶颈，您需要进一步优化。

更多有关 Instruments 的使用信息，请参见 [Instruments 用户指南](#)。

窗口和视图

窗口和视图是为 iPhone 应用程序构造用户界面的可视组件。窗口为内容显示提供背景平台，而视图负责绝大部分的内容描画，并负责响应用户的交互。虽然本章讨论的概念和窗口及视图都相关联，但是讨论过程更加关注视图，因为视图对系统更为重要。

视图对 iPhone 应用程序是如此的重要，以至于在一个章节中讨论视图的所有方面是不可能的。本章将关注窗口和视图的基本属性、各个属性之间的关系、以及在应用程序中如何创建和操作这些属性。本章不讨论视图如何响应触摸事件或如何描画定制内容，有关那些主题的更多信息，请分别参见[“事件处理”](#)和[“图形和描画”](#)部分。

什么是窗口和视图？

和 Mac OS X 一样，iPhone OS 通过窗口和视图在屏幕上展现图形内容。虽然窗口和视图对象之间在两个平台上有很多相似性，但是具体到每个平台上，它们的作用都有轻微的差别。

UIWindow 的作用

和 Mac OS X 的应用程序有所不同，iPhone 应用程序通常只有一个窗口，表示为一个 [UIWindow](#) 类的实例。您的应用程序在启动时创建这个窗口（或者从 [nib 文件](#) 进行装载），并往窗口中加入一或多个视图，然后将它显示出来。窗口显示出来之后，您很少需要再次引用它。

在 iPhone OS 中，窗口对象并没有像关闭框或标题栏这样的视觉装饰，用户不能直接对其进行关闭或其它操作。所有对窗口的操作都需要通过其编程接口来实现。应用程序可以借助窗口对象来进行事件传递。窗口对象会持续跟踪当前的第一响应者对象，并在 [UIApplication](#) 对象提出请求时将事件传递它。

还有一件可能让有经验的 Mac OS X 开发者觉得奇怪的事是 UIWindow 类的继承关系。在 Mac OS X 中，NSWindow 的父类是 NSResponder；而在 iPhone OS 中，UIWindow 的父类是 [UIView](#)。因此，窗口在 iPhone OS 中也是一个视图对象。不管其起源如何，您通常可以将 iPhone OS 上的窗口和 Mac OS X 的窗口同样对待。也就是说，您通常不必直接操作 UIWindow 对象中与视图有关的[属性变量](#)。

在创建应用程序窗口时，您应该总是将其初始的边框尺寸设置为整个屏幕的大小。如果您的窗口是从 nib 文件装载得到，Interface Builder 并不允许创建比屏幕尺寸小的窗口；然而，如果您的窗口是通过编程方式创建的，则必须在创建时传入期望的边框矩形。除了屏幕矩形之外，没有理由传入其它边框矩形。屏幕矩形可以通过 [UIScreen](#) 对象来取得，具体代码如下所示：

```
UIWindow* aWindow = [[[UIWindow alloc] initWithFrame:[UIScreen mainScreen] bounds]] autorelease];
```

虽然 iPhone OS 支持将一个窗口叠放在其它窗口的上方，但是您的应用程序永远不应创建多个窗口。系统自身使用额外的窗口来显示系统状态条、重要的警告、以及位于应用程序窗口上方的其它消息。如果您希望在自己的内容上方显示警告，可以使用 UIKit 提供的警告视图，而不应创建额外的窗口。

UIView 是作用

视图是 [UIView](#) 类的实例，负责在屏幕上定义一个矩形区域。在 iPhone 的应用程序中，视图在展示用户界面及响应用户界面交互方面发挥关键作用。每个视图对象都要负责渲染视图矩形区域中的内容，并响应该区域中发生的触碰事件。这一双重行为意味着视图是应用程序与用户交互的重要机制。在一个基于[模型-视图-控制器](#)的应用程序中，视图对象明显属于视图部分。

除了显示内容和处理事件之外，视图还可以用于管理一或多个子视图。**子视图**是指嵌入到另一视图对象边框内部的视图对象，而被嵌入的视图则被称为父视图或**超视图**。视图的这种布局方式被称为**视图层次**，一个视图可以包含任意数量的子视图，通过为子视图添加子视图的方式，视图可以实现任意深度的嵌套。视图在视图层次中的组织方式决定了在屏幕上显示的内容，原因是子视图总是被显示在其父视图的上方；这个组织方法还决定了视图如何响应事件和变化。每个父视图都负责管理其直接的子视图，即根据需要调整它们的位置和尺寸，以及响应它们没有处理的事件。

由于视图对象是应用程序和用户交互的主要途径，所以需要在很多方面发挥作用，下面是其中的一小部分：

描画和动画

视图负责对其所属的矩形区域进行描画。

某些视图[属性变量](#)可以以动画的形式过渡到新的值。

布局和子视图管理

视图管理着一个子视图列表。

视图定义了自身相对于其父视图的尺寸调整行为。

必要时，视图可以通过代码调整其子视图的尺寸和位置。

视图可以将其坐标系统下的点转换为其它视图或窗口坐标系统下的点。

事件处理

视图可以接收触摸事件。

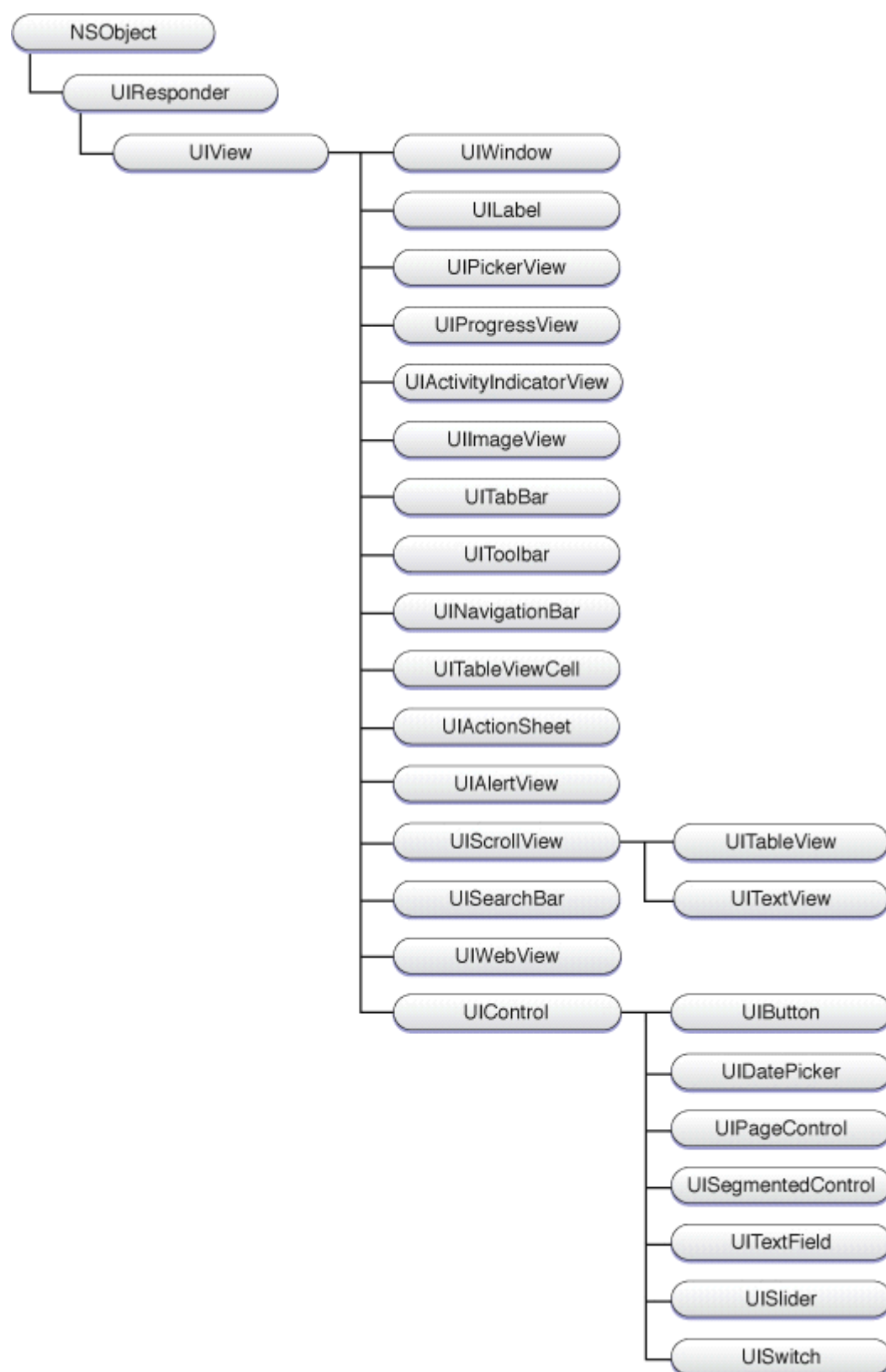
视图是响应者链的参与者。

在 iPhone 应用程序中，视图和视图控制器紧密协作，管理若干方面的视图行为。视图控制器的作用是处理视图的装载与卸载、处理由于设备旋转导致的界面旋转，以及和用于构建复杂用户界面的高级导航对象进行交互。更多这方面的信息请参见[“视图控制器的作用”](#)部分。本章的大部分内容都着眼于解释视图的这些作用，以及说明如何将您自己的定制代码关联到现有的 UIView 行为中。

UIKit 的视图类

[UIView](#) 类定义了视图的基本行为，但并不定义其视觉表示。相反，UIKit 通过其子类来为像文本框、按键、及工具条这样的标准界面元素定义具体的外观和行为。图2-1显示了所有 UIKit 视图类的层次框图。除了 [UIView](#) 和 [UIControl](#) 类是例外，这个框图中的大多数视图都设计为可直接使用，或者和[委托对象](#)结合使用。

图2-1 视图的类层次



这个视图层次可以分为如下几个大类：

容器

容器视图用于增强其它视图的功能，或者为视图内容提供额外的视觉分隔。比如，[UIScrollView](#)类可以用于显示因内容太大而无法显示在一个屏幕上的视图。[UITableView](#)类是 [UIScrollView](#) 类的子类，用于管理数据列表。表格的行可以支持选择，所以通常也用于层

次数据的导航—比如用于挖掘一组有层次结构的对象。

[UIToolbar](#) 对象则是一个特殊类型的容器，用于为一或多个类似于按键的项提供视觉分组。工具条通常出现在屏幕的底部。Safari、Mail、和 Photos 程序都使用工具条来显示一些按键，这些按键代表经常使用的命令。工具条可以一直显示，也可以根据应用程序的需要进行显示。

控件

控件用于创建大多数应用程序的用户界面。控件是一种特殊类型的视图，继承自 [UIControl](#) 超类，通常用于显示一个具体的值，并处理修改这个值所需要的所有用户交互。控件通常使用标准的系统范式（比如目标-动作模式和委托模式）来通知应用程序发生了用户交互。控件包括按键、文本框、滑块、和切换开关。

显示视图

控件和很多其它类型的视图都提供了交互行为，而另外一些视图则只是用于简单地显示信息。具有这种行为的 UIKit 类包括 [UIImageView](#)、[UILabel](#)、[UIProgressView](#)、[UIActivityIndicatorView](#)。

文本和 web 视图

文本和 web 视图为应用程序提供更为高级的显示多行文本的方法。[UITextView](#) 类支持在滚动区域内显示和编辑多行文本；而 [UIWebView](#) 类则提供了显示 HTML 内容的方法，通过这个类，您可以将图形和高级的文本格式选项集成到应用程序中，并以定制的方式对内容进行布局。

警告视图和动作表单

警告视图和动作表单用于即刻取得用户的注意。它们向用户显示一条消息，同时还有一或多个可选的按键，用户通过这些按键来响应消息。警告视图和动作表单的功能类似，但是外观和行为不同。举例来说，[UIAlertView](#) 类在屏幕上弹出一个蓝色的警告框，而 [UIActionSheet](#) 类则从屏幕的底部滑出动作框。

导航视图

页签条和导航条和[视图控制器](#)结合使用，为用户提供从一个屏幕到另一个屏幕的导航工具。在使用时，您通常不必直接创建 [UITabBar](#) 和 [UINavigationController](#) 的项，而是通过恰当的控制器接口或 Interface Builder 来对其进行配置。

窗口

窗口提供一个描画内容的表面，是所有其它视图的根容器。每个应用程序通常都只有一个窗口。更多信息请参见[“UIWindow 的作用”](#)部分。

除了视图之外，UIKit 还提供了视图控制器，用于管理这些对象。更多信息请参见[“视图控制器的作用”](#)部分。

视图控制器的作用

运行在 iPhone OS 上的应用程序在如何组织内容和如何将内容呈现给用户方面有很多选择。含有很多内容的应用程序可以将内容分为多个屏幕。在运行时，每个屏幕的背后都是一组视图对象，负责显示该屏幕的数据。一个屏幕的视图后面是一个[视图控制器](#)其作用是管理那些视图上显示的数据，并协调它们和应用程序其它部分的关系。

[UIViewController](#) 类 负责创建其管理的视图及在低内存时将它们从内容中移出。视图控制器还为某些标准的系统行为提供自动响应。比如，在响应设备方向变化时，如果应用程序支持该 方向，视图控制器可以对其管理的视图进行尺寸调整，使其适应新的方向。您也可以通过视图控制器来将新的视图以模式框的方式显示在当前视图的上方。

除了基础的 UIViewController 类之外，UIKit 还包含很多高级子类，用于处理平台共有的某

些高级接口。特别需要提到的是，导航控制器用于显示多屏具有一定层次结构的内容；而页签条控制器则支持用户在一组不同的屏幕之间切换，每个屏幕都代表应用程序的一种不同的操作模式。

有关如何通过视图控制器管理用户界面上视图的更多信息，请参见 [iPhone OS 的视图控制器编程指南](#)。

视图架构和几何属性

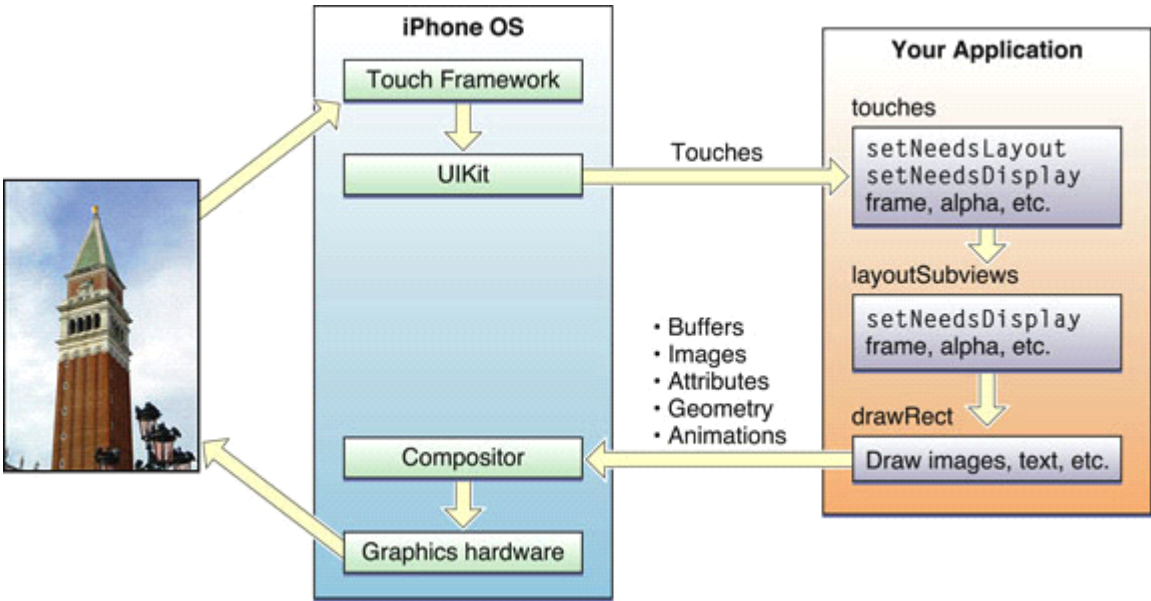
由于视图是 iPhone 应用程序的焦点对象，所以对视图与系统其它部分的交互机制有所了解是很重要的。UIKit 中的标准视图类为应用程序免费提供相当数量的行为，还提供了一些定义良好的集成点，您可以通过这些集成点来对标准行为进行定制，完成应用程序需要做的工作。

本文的下面部分将解释视图的标准行为，并说明哪些地方可以集成您的定制代码。如果需要特定类的集成点信息，请参见该类的参考文档。您可以从 [UIKit 框架参考](#) 中取得所有类参考文档的列表。

视图交互模型

任 何时候，当用户和您的程序界面进行交互、或者您的代码以编程的方式进行某些修改时，UIKit 内部都会发生一个复杂的事件序列。在事件序列的一些特定的点上，UIKit 会调用您的视图类，使它们有机会代表应用程序进行事件响应。理解这些调用点是很重要的，有助于理解您的视图对象和系统在哪里进行结合。图2-2显示了从用户触击屏幕到图形系统更新屏幕内容这一过程的基本事件序列。以编程方式触发事件的基本步骤与此相同，只是没有最初的用户交互。

图2-2 UIKit 和您的视图对象之间的交互



下面的步骤说明进一步剖析了图2-2中的事件序列，解释了序列的每个阶段都发生了什么，以及应用程序可能如何进行响应。

用户触击屏幕。

硬件将触击事件报告给 UIKit 框架。

UIKit 框架将触击信息封装为一个 [UIEvent](#) 对象，并派发给恰当的视图（有关 UIKit 如何将事件递送给您的视图的详细解释，请参见[“事件的传递”](#)部分）。

视图的事件处理方法可以通过下面的方式来响应事件：

调整视图或其子视图的[属性变量](#)（边框、边界、透明度等）。

将视图（或其子视图）标识为需要修改布局。

将视图（或其子视图）标识为布局需要重画。

将数据发生的变化通报给[控制器](#)。

当然，上述的哪些事情需要做及调用什么方法来完成是由视图来决定的。

如果视图被标识为需要重新布局，UIKit 就调用视图的 [layoutSubviews](#) 方法。

您可以在自己的定制视图中重载这个方法，以便调整子视图的尺寸和位置。举例来说，如果一个视图具有很大的滚动区域，就需要使用几个子视图来“平铺”，而不是 创建一个内存很可能装不下的大视图。在这个方法的实现中，视图可以隐藏所有不需显示在屏幕上的子视图，或者在重新定位之后将它们用于显示新的内容。作为这 个过程的一部分，视图也可以将用于“平铺”的子视图标识为需要重画。

如果视图的任何部分被标识为需要重画，UIKit 就调用该视图的 [drawRect:](#)方法。

UIKit 只对那些需要重画的视图调用这个方法。在这个方法的实现中，所有视图都应该尽可能快地重画指定的区域，且都应该只重画自己的内容，不应该描画子视图的内容。在这个调用点上，视图不应该尝试进一步改变其属性或布局。

所有更新过的视图都和其它可视内容进行合成，然后发送给图形硬件进行显示。

图形硬件将渲染完成的内容转移到屏幕。

请注意：上述的更新模型主要适用于采纳内置视图和描画技术的应用程序。如果您的应用程序使用 OpenGL ES 来描画内容，则通常要配置一个全屏的视图，然后直接在 OpenGL 的图形上下文中进行描画。您的视图仍然需要处理触碰事件，但不需要对子视图进行布局或者实现 [drawRect:](#)方法。有关 OpenGL ES 的更多信息，请参见[“用 OpenGL ES 进行描画”](#)部分。

基于上述的步骤说明可以看出，UIKit 为您自己定制的视图提供如下主要的结合点：

下面这些事件处理方法：

[touchesBegan:withEvent:](#)

[touchesMoved:withEvent:](#)

[touchesEnded:withEvent:](#)

[touchesCancelled:withEvent:](#)

[layoutSubviews](#) 方法

[drawRect:](#)方法

大多数定制视图通过实现这些方法来得到自己期望的行为。您可能不需要重载所有方法，举例来说，如果您实现的视图是固定尺寸的，则可能不需要重载 [layoutSubviews](#) 方法。类似地，如果您实现的视图只是显示简单的内容，比如文本或图像，则通常可以通过简单地嵌入 [UIImageView](#) 和 [UILabel](#) 对象作为子视图来避免描画。

重要的是要记住，这些是主要的结合点，但不是全部。[UIView](#) 类中有几个方法的设计目的就是让子类重载的。您可以通过查阅 [UIView 类参考](#)中的描述来了解哪些方法可以被重载。

视图渲染架构

虽然您通过视图来表示屏幕上的内容，但是 [UIView](#) 类自身的很多基础行为却严重依赖于另一个对象。UIKit 中每个视图对象的背后都有一个 Core Animation 层对象，它是一个 [CALayer](#)

类的实例，该类为视图内容的布局和渲染、以及合成和动画提供基础性的支持。

和 Mac OS X（在这个平台上 Core Animation 支持是可选的）不同的是，iPhone OS 将 Core Animation 集成到视图渲染实现的核心。虽然 Core Animation 发挥核心作用，但是 UIKit 在 Core Animation 上面提供一个透明的接口层，使编程体验更为流畅。这个透明的接口使开发者在大多数情况下不必直接访问 Core Animation 的层，而是通过 UIView 的方法和[属性声明](#)取得类似的行为。然而，当 UIView 类没有提供您需要的接口时，Core Animation 就变得重要了，在那种情况下，您可以深入到 Core Animation 层，在应用程序中实现一些复杂的渲染。本文的下面部分将介绍 Core Animation 技术，描述它通过 UIView 类为您提供的一些功能。有关如何使用 Core Animation 进行高级渲染的更多信息，请参见[Core Animation 编程指南](#)。

Core Animation 基础

Core Animation 利用了硬件加速和架构上的优化来实现快速渲染和实时动画。当视图的 `drawRect:` 方法首次被调用时，层会将描画的结果捕捉到一个位图中，并在随后的重画中尽可能使用这个缓存的位图，以避免调用开销很大的 `drawRect:` 方法。这个过程使 Core Animation 得以优化合成操作，取得期望的性能。

Core Animation 把和视图对象相关联的层存储在一个被称为**层树**的层次结构中。和视图一样，层树中的每个层都只有一个父亲，但可以嵌入任意数量的子层。缺省情况下，层树中对象的组织方式和视图在视图层次中的组织方式完全一样。但是，您可以在层树中添加层，而不同时添加相应的视图。当您希望实现某种特殊的视觉效果、而又不需要在视图上保持这种效果时，就可能需要这种技术。

实际上，层对象是 iPhone OS 渲染和布局系统的推动力，大多数视图属性实际上是其层对象属性的一个很薄的封装。当您（直接使用 CALayer 对象）修改层树上层对象的属性时，您所做的改变会立即反映在层对象上。但是，如果该变化触发了相应的动画，则可能不会立即反映在屏幕上，而是必须随着时间的变化以动画的形式表现在屏幕上。为了管理这种类型的动画，Core Animation 额外维护两组层对象，我们称之为**表示树**和**渲染树**。

表示树反映的是层在展示给用户时的当前状态。假定您对层值的变化实行动画，则在动画开始时，表示层反映的是老的值；随着动画的进行，Core Animation 会根据动画的当前帧来更新表示树层的值；然后，渲染树就和表示树一起，将变化渲染在屏幕上。由于渲染树运行在单独的进程或线程上，所以它所做的工作并不影响应用程序的主运行循环。虽然层树和表示树都是公开的，但是渲染树的接口是私有。

在视图后面设置层对象对描画代码的性能有很多重要的影响。使用层的好处在于视图的大多数几何变化都不需要重画。举例来说，改变视图的位置和尺寸并需要重画视图的内容，只需简单地重用层缓存的位图就可以了。对缓存的内容实行动画比每次都重画内容要有效得多。使用层的缺点在于层是额外的缓存数据，会增加应用程序的内存压力。如果您的应用程序创建太多的视图，或者创建多个很大的视图，则可能很快就会出现内存不够用的情形。您不用担心在应用程序中使用视图，但是，如果有现成的视图可以重用，就不要创建新的视图对象。换句话说，您应该设法使内存中同时存在的视图对象数量最小。

有关 Core Animation 的进一步概述、对象树、以及如何创建动画，请参见[Core Animation 编程指南](#)。

改变视图的层

在 iPhone OS 系统中，由于视图必须有一个与之关联的层对象，所以 `UIView` 类在初始化时会自动创建相应的层。您可以通过视图的 [layer](#) 属性访问这个层，但是不能在视图创建完成后改变层对象。

如果您希望视图使用不同类型的层，必须重载其 [layerClass](#) 类方法，并在该方法中返回您希望使用的层对象。使用不同层类的最常见理由是为了实现一个基于 OpenGL 的应用程序。为了使用 OpenGL 描画命令，视图下面的层必须是 [CAEAGLLayer](#) 类的实例，这种类型的层可以和 OpenGL 渲染调用进行交互，最终在屏幕上显示期望的内容。

重要提示：您永远不应修改视图层的 `delegate` 属性，该属性用于存储一个指向视图的指针，应该被认为是私有的。类似地，由于一个视图只能作为一个层的[委托](#)，所以您必须避免将它作为其它层对象的委托，否则会导致应用程序崩溃。

动画支持

iPhone OS 的每个视图后面都有一个层对象，这样做的好处之一是使视图内容更加易于实现动画。请记住，动画并不一定是为了在视觉上吸引眼球，它可以将应用程序界面变化的上下文呈现给用户。举例来说，当您在屏幕转移过程中使用过渡时，过渡本身就向用户指示屏幕之间的联系。系统自动支持了很多经常使用的动画，但您也可以为界面上的其它部分创建动画。

[UIView](#) 类的很多属性都被设计为可动画的（animatable）。可动画的属性是指当属性从一个值变为另一个值的时候，可以半自动地支持动画。您仍然必须告诉 UIKit 希望执行什么类型的动画，但是动画一旦开始，Core Animation 就会全权负责。UIView 对象中支持动画的属性有如下几个：

`frame`

`bounds`

`center`

[transform](#)

[alpha](#)

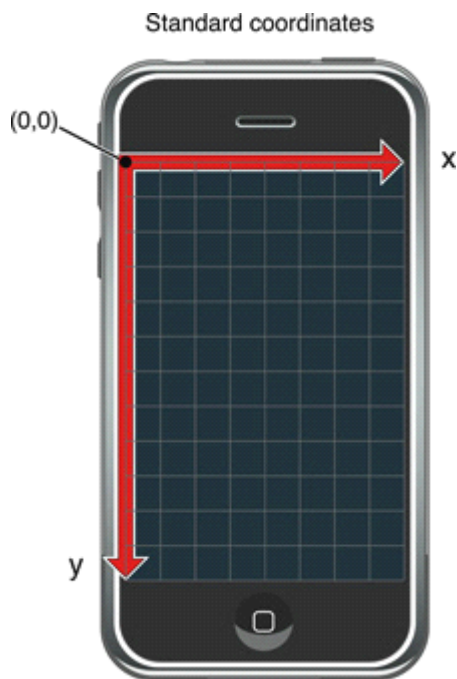
虽然其它的视图属性不直接支持动画，但是您可以为其中的一部分显式创建动画。显式动画要求您做很多管理动画和渲染内容的工作，通过使用 Core Animation 提供的基础设施，这些工作仍然可以得到良好的性能。

有关如何通过 `UIView` 类创建动画的更多信息，请参见“[实现视图动画](#)”部分；有关如何创建显式动画的更多信息，则请参见 [Core Animation 编程指南](#)。

视图坐标系统

UIKit 中的坐标是基于这样的坐标系统：以左上角为坐标的原点，原点向下和向右为坐标轴正向。坐标值由浮点数来表示，内容的布局和定位因此具有更高的精度，还可以支持与分辨率无关的特性。[图2-3](#)显示了这个相对于屏幕的坐标系统，这个坐标系统同时也用于 [UIWindow](#) 和 [UIView](#) 类。视图坐标系统的方向和 Quartz 及 Mac OS X 使用的缺省方向不同，选择这个特殊的方向是为了使布局用户界面上的控件及内容更加容易。

图2-3 视图坐标系统



您在编写界面代码时，需要知道当前起作用的坐标系统。每个窗口和视图对象都维护一个自己本地的坐标系统。视图中发生的所有描画都是相对于视图本地的坐标系统。但是，每个视图的边框矩形都是通过其父视图的坐标系统来指定，而事件对象携带的坐标信息则是相对于应用程序窗口的坐标系统。为了方便，`UIWindow` 和 `UIView` 类都提供了一些方法，用于在不同对象之间进行坐标系统的转换。

虽然 Quartz 使用的坐标系统不以左上角为原点，但是对于很多 Quartz 调用来说，这并不是问题。在调用视图的 `drawRect:` 方法之前，UIKit 会自动对描画环境进行配置，使左上角成为坐标系统的原点，在这个环境中发生的 Quartz 调用都可以正确地在视图中描画。您唯一需要考虑不同坐标系统之间差别的场合是当您自行通过 Quartz 建立描画环境的时候。

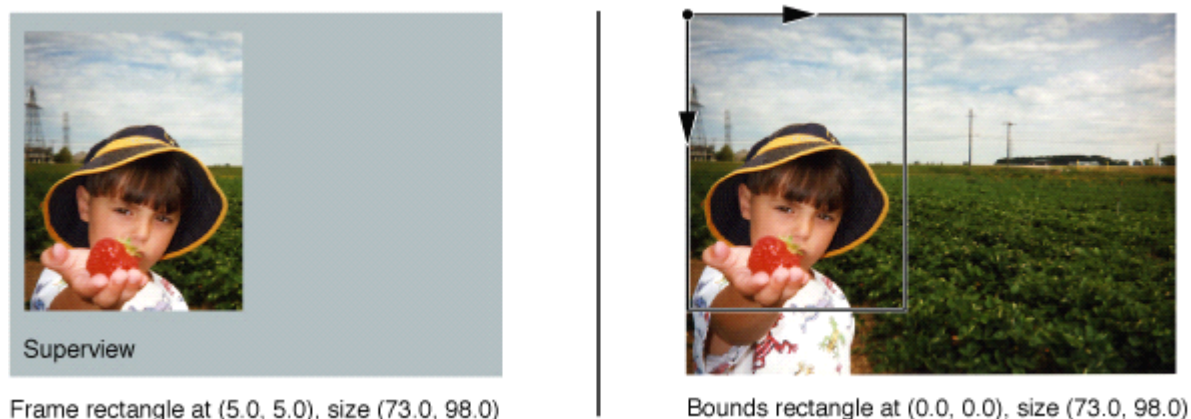
更多有关坐标系统、Quartz、和描画的一般信息，请参见[“图形和描画”](#)部分。

边框、边界、和中心的关系

视图对象通过 [frame](#)、[bounds](#)、和 [center 属性声明](#) 来跟踪自己的大小和位置。`frame` 属性包含一个矩形，即**边框矩形**，用于指定视图相对于其父视图坐标系统的位置和大小。`bounds` 属性也包含一个矩形，即**边界矩形**，负责定义视图相对于本地坐标系统的位置和大小。虽然边界矩形的原点通常被设置为 (0, 0)，但这并不是必须的。`center` 属性包含边框矩形的**中心点**。在代码中，您可以将 `frame`、`bounds`、和 `center` 属性用于不同的目的。边界矩形代表视图本地的坐标系统，因此，在描画和事件处理代码中，经常借助它来取得视图中发生事件或需要更新的位置。中心点代表视图的中心，改变中心点一直是移动视图位置的最好方法。边框矩形是一个通过 `bounds` 和 `center` 属性计算得到的便利值，只有当视图的变换属性被设置恒等变换时，边框矩形才是有效的。

图2-4显示了边框矩形和边界矩形之间的关系。右边的整个图像是从视图的(0, 0)开始描画的，但是由于边界的大小和整个图像的尺寸不相匹配，所以位于边界矩形之外的图像部分被自动裁剪。在视图和它的父视图进行合成的时候，视图在其父视图中的位置是由视图边框矩形的原点决定的。在这个例子中，该原点是(5, 5)。结果，视图的内容就相对于父视图的原点向下向右移动相应的尺寸。

图2-4 视图的边框和边界之间的关系



如果没有经过变换，视图的位置和大小就由上述三个互相关联的属性决定的。当您在代码中通过 [initWithFrame:](#) 方法创建一个视图对象时，其 `frame` 属性就会被设置。该方法同时也将 `bounds` 矩形的原点初始化为(0.0, 0.0)，大小则和视图的边框相同。然后 `center` 属性会被设置为边框的中心点。

虽然您可以分别设置这些属性的值，但是设置其中的一个属性会引起其它属性的改变，具体关系如下：

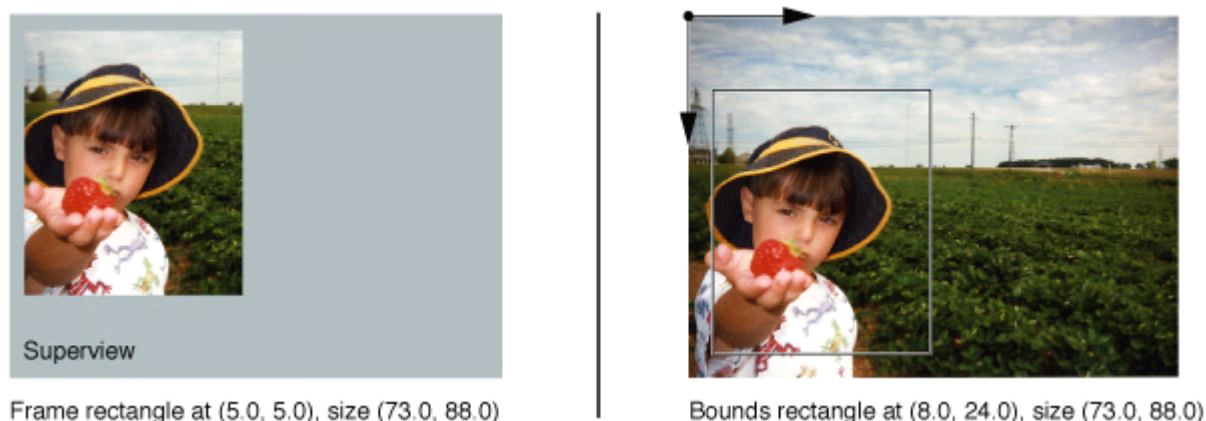
当您设置 `frame` 属性时，`bounds` 属性的大小会被设置为与 `frame` 属性的大小相匹配的值，`center` 属性也会被调整为与新的边框中心点相匹配的值。

当您设置 `center` 属性时，`frame` 的原点也会随之改变。

当您设置 `bounds` 矩形的大小时，`frame` 矩形的大小也会随之改变。

您可以改变 `bounds` 的原点而不影响其它两个属性。当您这样做时，视图会显示您标识的图形部分。在图2-4中，边界的原点被设置为(0.0, 0.0)。在图2-5中，该原点被移动到(8.0, 24.0)。结果，显示出来的是视图图像的不同部分。但是，由于边框矩形并没有改变，新的内容在父视图中的位置和之前是一样的。

图2-5 改变视图的边界



请注意：缺省情况下，视图的边框并不会被父视图的边框裁剪。如果您希望让一个视图裁剪其子视图，需要将其 [clipsToBounds](#) 属性设置为 YES。

坐标系统变换

在视图的 `drawRect:` 方法中常常借助坐标系统变换来进行描画。而在 iPhone OS 系统中，您

还可以用它来实现视图的某些视觉效果。举例来说，[UIView](#) 类中包含一个 [transform 属性声明](#)，您可以通过它来对整个视图实行各种类型的平移、比例缩放、和变焦缩放效果。缺省情况下，这个属性的值是一个恒等变换，不会改变视图的外观。在加入变换之前，首先要得到该属性中存储的 [CGAffineTransform](#) 结构，用相应的 Core Graphics 函数实行变换，然后再将修改后的变换结构重新赋值给视图的 transform 属性。

请注意：当您将变换应用到视图时，所有执行的变换都是相对于视图的中心点。

平移一个视图会使其所有的子视图和视图本身的内容一起移动。由于子视图的坐标系统是继承并建立在这些变化的基础上的，所以比例缩放也会影响子视图的描画。有关如何控制视图内容缩放的更多信息，请参见[“内容模式和比例缩放”](#)部分。

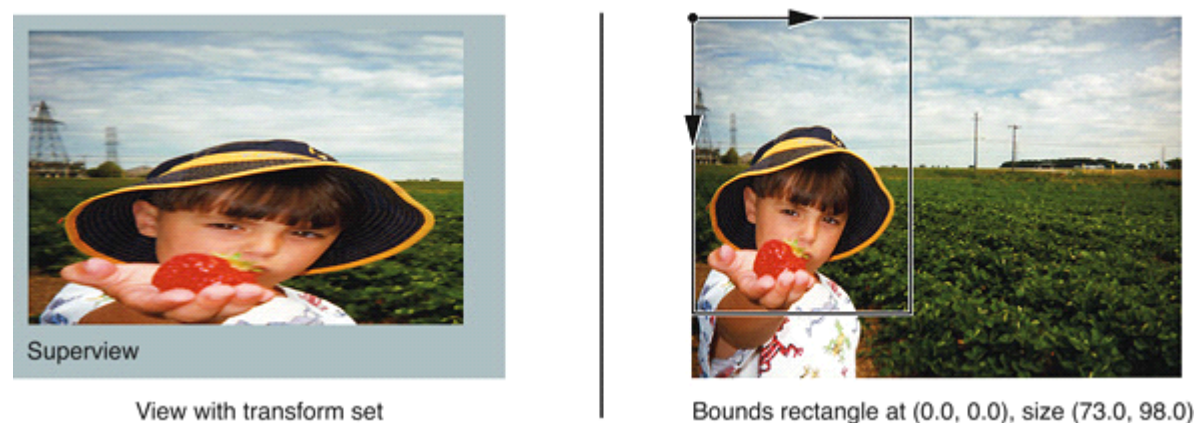
重要提示：如果 transform 属性的值不是恒等变换，则 frame 属性的值就是未定义的，必须被忽略。在设置变换属性之后，请使用 bounds 和 center 属性来获取视图的位置和大小。

有关如何在 drawRect:方法中使用变换的信息，请参见[“坐标和坐标变换”](#)部分；有关用于修改 [CGAffineTransform](#) 结构的函数，则请参见 [CGAffineTransform 参考](#)。

内容模式与比例缩放

当您改变视图的边界，或者将一个比例因子应用到视图的 [transform 属性声明](#)时，边框矩形会发生等量的变化。根据内容模式的不同，视图的内容也可能被缩放或重新定位，以反映上述的变化。视图的 [contentMode](#) 属性决定了边界变化和缩放操作作用到视图上产生的效果。缺省情况下，这个属性的值被设置为 [UIViewContentModeScaleToFill](#)，意味着视图内容总是被缩放，以适应新的边框尺寸。作为例子，图2-6显示了当视图的水平缩放因子放大一倍时产生的效果。

图2-6 使用 scale-to-fill 内容模式缩放视图



视图内容的缩放仅在首次显示视图的时候发生，渲染后的内容会被缓存在视图下面的层上。当边界或缩放因子发生变化时，UIKit 并不强制视图进行重画，而是根据其内容模式决定如何显示缓存的内容。图2-7比较了在不同的内容模式下，改变视图边界或应用不同的比例缩放因子时产生的结果。

图2-7 内容模式比较

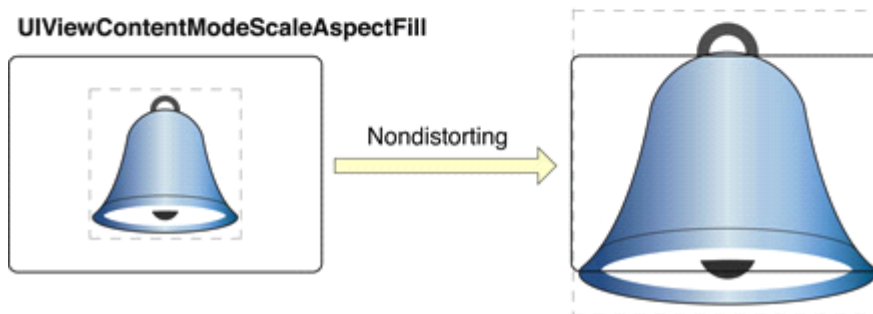
UIViewContentModeScaleToFill



UIViewContentModeScaleAspectFit



UIViewContentModeScaleAspectFill



对视图应用一个比例缩放因子总是会使其内容发生缩放，而边界的改变在某些内容模式下则不会发生同样的结果。不同的 [UIViewContentMode](#) 常量（比如 [UIViewContentModeTop](#) 和 [UIViewContentModeBottomRight](#)）可以使当前的内容在视图的不同角落或沿着视图的不同边界显示，还有一种模式可以将内容显示在视图的中心。在这些模式的作用下，改变边界矩形只会简单地将现有的视图内容移动到新的边界矩形中对应的位置上。

当您希望在应用程序中实现尺寸可调整的控件时，请务必考虑使用内容模式。这样做可以避免控件的外观发生变形，以及避免编写定制的描画代码。按键和分段控件（segmented control）特别适合基于内容模式的描画。它们通常使用几个图像来创建控件外观。除了有两个固定尺寸的盖帽图像之外，按键可以通过一个可伸展的、宽度只有一个像素的中心图像来实现水平方向的尺寸调整。它将每个图像显示在自己的图像视图中，而将可伸展的中间图像的内容模式设置为 [UIViewContentModeScaleToFill](#)，使得在尺寸调整时两端的外观不会变形。更为重要的是，每个图像视图的关联图像都可以由 Core Animation 来缓存，因此不需要编写描画代码就可以支持动画，从而使大大提高了性能。

内容模式通常有助于避免视图内容的描画，但是当您希望对缩放和尺寸调整过程中的视图外观进行特别的控制时，也可以使用 [UIViewContentModeRedraw](#) 模式。将视图的内容模式设置为这个值可以强制 Core Animation 使视图的内容失效，并调用视图的 `drawRect:` 方法，而不是自动进行缩放或尺寸调整。

自动尺寸调整行为

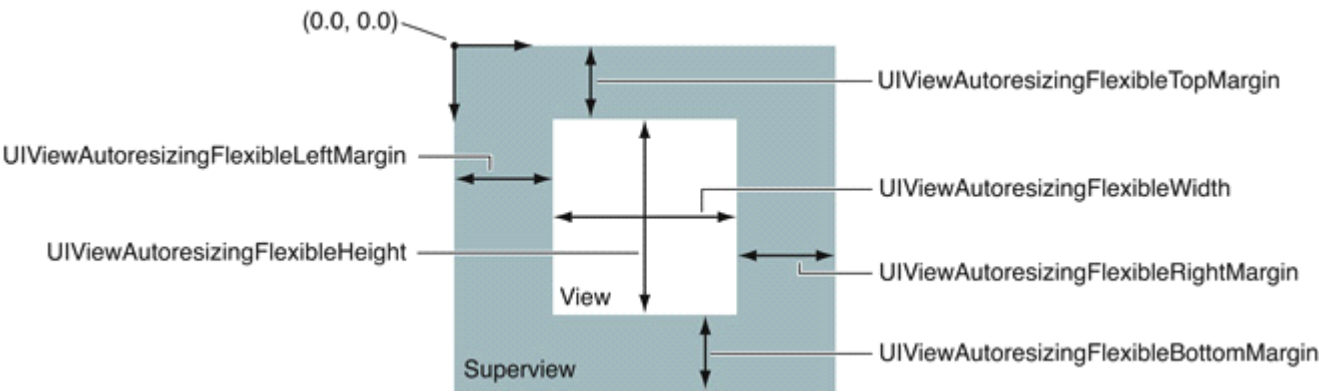
当您改变视图的边框矩形时，其内嵌子视图的位置和尺寸往往也需要改变，以适应原始视图的新尺寸。如果视图的 [autoresizesSubviews 属性声明](#) 被设置为 YES，则其子视图会根据 [autoresizingMask](#) 属性的值自动进行尺寸调整。简单配置一下视图的自动尺寸调整掩码常常就能使应用程序得到合适的行为；否则，应用程序就必须通过重载 [layoutSubviews](#) 方法来提供自己的实现。

设置视图的自动尺寸调整行为的方法是通过位 OR 操作符将期望的自动尺寸调整常量连结起来，并将结果赋值给视图的 `autoresizingMask` 属性。表2-1列举了自动尺寸调整常量，并描述这些常量如何影响给定视图的尺寸和位置。举例来说，如果要使一个视图和其父视图左下角的相对位置保持不变，可以加入 `UIViewAutoresizingFlexibleRightMargin` 和 `UIViewAutoresizingFlexibleTopMargin` 常量，并将结果赋值给 `autoresizingMask` 属性。当同一个轴向有多个部分被设置为可变时，尺寸调整的裕量会被平均分配到各个部分上。

表2-1 自动尺寸调整掩码常量自动尺寸调整掩码	描述
UIViewAutoresizingNone	这个常量如果被设置，视图将不进行自动尺寸调整。
UIViewAutoresizingFlexibleHeight	这个常量如果被设置，视图的高度将和父视图的高度一起成比例变化。否则，视图的高度将保持不变。
UIViewAutoresizingFlexibleWidth	这个常量如果被设置，视图的宽度将和父视图的宽度一起成比例变化。否则，视图的宽度将保持不变。
UIViewAutoresizingFlexibleLeftMargin	这个常量如果被设置，视图的左边界将随着父视图宽度的变化而按比例进行调整。否则，视图和其父视图的左边界的相对位置将保持不变。
UIViewAutoresizingFlexibleRightMargin	这个常量如果被设置，视图的右边界将随着父视图宽度的变化而按比例进行调整。否则，视图和其父视图的右边界的相对位置将保持不变。
UIViewAutoresizingFlexibleBottomMargin	这个常量如果被设置，视图的底边界将随着父视图高度的变化而按比例进行调整。否则，视图和其父视图的底边界的相对位置将保持不变。
UIViewAutoresizingFlexibleTopMargin	这个常量如果被设置，视图的上边界将随着父视图高度的变化而按比例进行调整。否则，视图和其父视图的上边界的相对位置将保持不变。

图2-8为这些常量值的位置提供了一个图形表示。如果这些常量之一被省略，则视图在相应方向上的布局就被固定；如果某个常量被包含在掩码中，在该方向的视图布局就就灵活的。

图2-8 视图的自动尺寸调整掩码常量



如 果您通过 Interface Builder 配置视图，则可以用 Size 查看器的 Autosizing 控制来设置每个

视图的自动尺寸调整行为。上图中的灵活宽度及高度常量和 Interface Builder 中位于同样位置的弹簧具有同样的行为，但是空白常量的行为则是正好相反。换句话说，如果要将灵活右空白的自动尺寸调整行为应用到 Interface Builder 的某个视图，必须使相应方向空间的 Autosizing 控制为空，而不是放置一个支柱。幸运的是，Interface Builder 通过动画显示了您的修改对视图自动尺寸调整行为的影响。

如果视图的 `autoresizesSubviews` 属性被设置为 NO，则该视图的直接子视图的所有自动尺寸调整行为将被忽略。类似地，如果一个子视图的自动尺寸调整掩码被设置为 [UIViewAutoresizingNone](#)，则该子视图的尺寸将不会被调整，因而其直接子视图的尺寸也不会被调整。

请注意：为了使自动尺寸调整的行为正确，视图的 `transform` 属性必须设置为恒等变换；其它变换下的尺寸自动调整行为是未定义的。

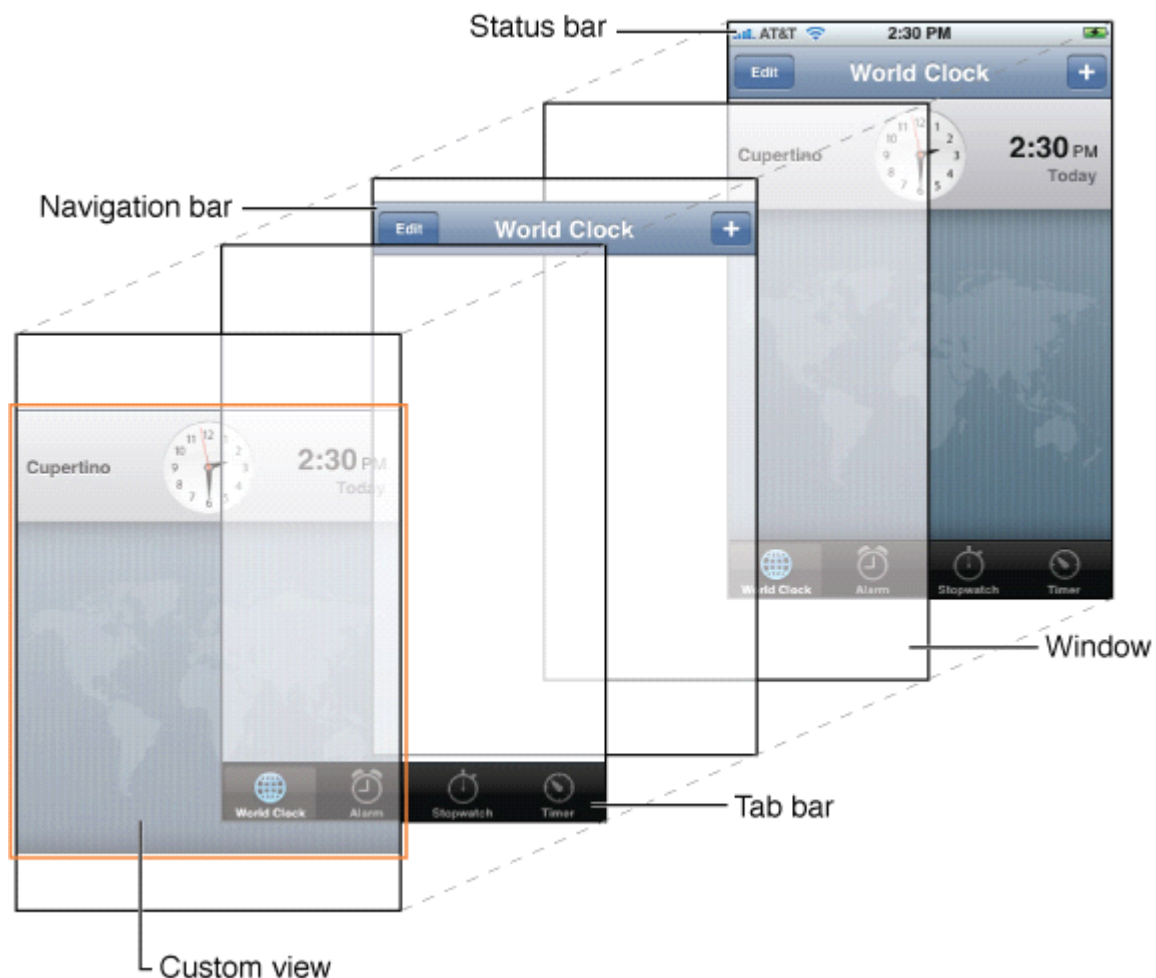
自动尺寸调整行为可以适合一些布局的要求，但是如果您希望更多地控制视图的布局，可以在适当的视图类中重载 `layoutSubviews` 方法。有关视图布局管理的更多信息，请参见[“响应布局的变化”](#)部分。

创建和管理视图层次

管理用户界面的视图层次是开发应用程序用户界面的关键部分。视图的组织方式不仅定义了应用程序的视觉外观，而且还定义了应用程序如何响应变化。视图层次中的父-子关系可以帮助我们定义应用程序中负责处理触摸事件的对象链。当用户旋转设备时，父-子关系也有助于定义每个视图的尺寸和位置是如何随着界面方向的变化而变化的。

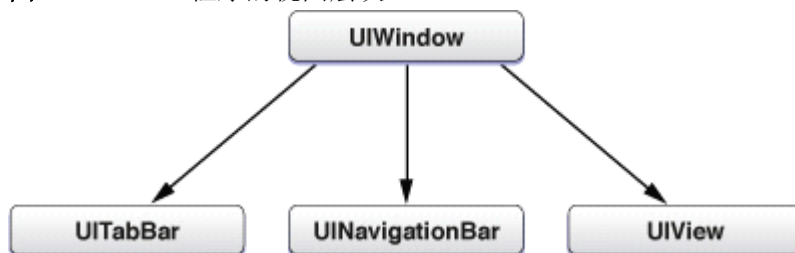
图2-9显示了一个简单的例子，说明如何通过视图的分层来创建期望的视觉效果。在 Clock 程序中，页签条和导航条视图，以及定制视图混合在一起，实现了整个界面。

图2-9 Clock 程序的视图层



如果您探究 Clock 程序中视图之间的关系，就会发现它们很像“改变视图的层”部分中显示的关系，窗口对象是应用程序的页签条、导航条、和定制视图的根视图。

图2-10 Clock 程序的视图层次



在 iPhone 应用程序的开发过程中，有几种建立视图层次的方法，包括基于 Interface Builder 的可视化方法和通过代码编程的方法。本文的下面部分将向您介绍如何装配视图层次，以及如何在建立视图层次之后寻找其中的视图，还有如何在不同的视图坐标系统之间进行转换。

创建一个视图对象

创建视图对象的最简单方法是使用 Interface Builder 进行制作，然后将视图对象从制成的 [nib 文件](#) 载入内存。在 Interface Builder 的图形环境中，您可以将新的视图从库中拖出，然后放到窗口或另一个视图中，以快速建立需要的视图层次。Interface Builder 使用的是活的视图对象，因此，当您用这个图形环境构建用户界面时，所看到的就是运行时装载的外观，而且不需要为视图层次中的每个视图编写单调乏味的 [内存分配和初始化](#) 代码。

如果您不喜欢 Interface Builder 和 nib 文件，也可以通过代码来创建视图。创建一个新的视图对象时，需要为其分配内存，并向该对象发送一个 [initWithFrame:](#) 消息，以对其进行初始化。举例来说，如果您要创建一个新的 UIView 类的实例作为其它视图的容器，则可以使用下面的代码：

```
CGRect viewRect = CGRectMake(0, 0, 100, 100);
UIView* myView = [[UIView alloc]
initWithFrame:viewRect];
```

请注意：虽然所有系统提供的视图对象都支持 [initWithFrame:](#) 消息，但是其中的一部分可能有自己偏好的 [初始化](#) 方法，您应该使用那些方法。有关定制初始化方法的更多信息，请参见相应的类参考文档。

您在视图初始化时指定的边框矩形代表该视图相对于未来父视图的位置和大小。在将视图显示于屏幕上之前，您需要将它加入到窗口或其它视图中。在这个时候，UIKit 会根据您指定的边框矩形将视图放置到其父视图的相应位置中。有关如何将视图添加到视图层次的信息，请参见“[添加和移除子视图](#)”部分。

添加和移除子视图

Interface Builder 是建立视图层次的最便利工具，因为它可以让您看到视图在运行时的外观。在界面制作完成后，它将视图对象及其层次关系保存在 nib 文件中。在运行时，系统会按照 nib 文件的内容为应用程序重新创建那些对象和关系。当一个 nib 文件被装载时，系统会自动调用重建视图层次所需要的 UIView 方法。

如果您不喜欢通过 Interface Builder 和 nib 文件来创建视图层次，则可以通过代码来 [创建](#)。如果一个视图必须具有某些子视图才能工作，则应该在其 [initWithFrame:](#) 方法中进行对其创建，以确保子视图可以和视图一起被显示和 [初始化](#)。如果子视图是应用程序设计的一部分（而不是视图工作必需的），则应该在视图的初始化代码之外进行创建。在 iPhone 程序中，有两个地方最常用于创建视图和子视图，它们是应用程序 [委托](#) 对象的 [applicationDidFinishLaunching:](#) 方法和 [视图控制器](#) 的 [loadView](#) 方法。

您可以通过下面的方法来操作视图层次中的视图对象：

调用父视图的 [addSubview:](#) 方法来添加视图，该方法将一个视图添加到子视图列表的最后。

调用父视图的 [insertSubview:...:](#) 方法可以在父视图的子视图列表中间插入视图。

调用父视图的 [bringSubviewToFront:](#)、[sendSubviewToBack:](#)、或 [exchangeSubviewAtIndex:withSubviewAtIndex:](#) 方法可以对父视图的子视图进行重新排序。使用这些方法比从父视图中移除子视图并再次插入要快一些。

调用子视图（而不是父视图）的 [removeFromSuperview](#) 方法可以将子视图从父视图中移除。在添加子视图时，UIKit 会根据子视图的当前边框矩形确定其在父视图中的初始位置。您可以随时通过修改子视图的 [frame 属性声明](#) 来改变其位置。缺省情况下，边框位于父视图可视边界外部的子视图不会被裁剪。如果您希望激活裁剪功能，必须将父视图的 [clipsToBounds](#) 属性设置为 YES。

程序清单 2-1 显示了一个应用程序委托对象的 [applicationDidFinishLaunching:](#) 方法示例。在这个例子中，应用程序委托在启动时通过代码创建全部的用户界面。界面中包含两个普通的 UIView 对象，用于显示基本颜色。每个视图都被嵌入到窗口中，窗口也是 UIView 的一个子类，因此可以作为父视图。父视图会保持它们的子视图，因此这个方法释放了新创建的视图对象，以避免重复保持。

程序清单2-1 创建一个带有视图的窗口

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    // Create the window object and assign it to the
    // window instance variable of the application delegate.
    window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen]
bounds]];
    window.backgroundColor = [UIColor whiteColor];

    // Create a simple red square
    CGRect redFrame = CGRectMake(10, 10, 100, 100);
    UIView *redView = [[UIView alloc] initWithFrame:redFrame];
    redView.backgroundColor = [UIColor redColor];

    // Create a simple blue square
    CGRect blueFrame = CGRectMake(10, 150, 100, 100);
    UIView *blueView = [[UIView alloc] initWithFrame:blueFrame];
    blueView.backgroundColor = [UIColor blueColor];

    // Add the square views to the window
    [window addSubview:redView];
    [window addSubview:blueView];

    // Once added to the window, release the views to avoid the
    // extra retain count on each of them.
    [redView release];
    [blueView release];

    // Show the window.
    [window makeKeyAndVisible];
}
```

重要提示：在内存管理方面，可以将子视图考虑为其它的集合对象。特别是当您通过 `addSubview:` 方法将一个视图作为子视图插入时，父视图会对其进行保持操作。反过来，当您通过 `removeFromSuperview` 方法将子视图从父视图移走时，子视图会被自动释放。在将视图加入视图层次之后释放该对象可以避免多余的保持操作，从而避免内存泄露。

有关 Cocoa 内存管理约定的更多信息，请参见 [Cocoa 内存管理编程指南](#)。

当您为某个视图添加子视图时，UIKit 会向相应的父子视图发送几个消息，通知它们当前发生的状态变化。您可以在自己的定制视图对诸如 [willMoveToSuperview:](#)、[willMoveToWindow:](#)、[willRemoveSubview:](#)、[didAddSubview:](#)、[didMoveToSuperview:](#) 和 [didMoveToWindow:](#) 这样的方法进行重载，以便在事件发生的前后进行必要的处理，并根据发生的变化更新视图的状态信息。

在视图层次建立之后，您可以通过视图的 [superview](#) 属性来取得其父视图，或者通过 [subviews](#) 属性取得视图的子视图。您也可以通过 [isDescendantOfView:](#) 方法来判定一个视图是否在其父视图的视图层中。一个视图层次的根视图没有父视图，因此其 `superview` 属性被设置为 `nil`。对于当前被显示在屏幕上的视图，窗口对象通常是整个视图层次的根视图。

您可以通过视图的 [window](#) 属性来取得指向其父窗口（如果有的话）的指针，如果视图还没有被链接到窗口上，则该属性会被设置为 `nil`。

视图层次中的坐标转换

很多时候，特别是处理事件的时候，应用程序可能需要将一个相对于某边框的坐标值转换为相对于另一个边框的值。例如，触摸事件通常使用基于窗口指标系统的坐标值来报告事件发生的位置，但是视图对象需要的是相对于视图本地坐标的位置信息，两者可能是不一样的。

[UIView](#) 类定义了下面这些方法，用于在不同的视图本地坐标系统之间进行坐标转换：

[convertPoint:fromView:](#)

[convertRect:fromView:](#)

[convertPoint:toView:](#)

[convertRect:toView:](#)

`convert...:fromView:` 方法将指定视图的坐标值转换为视图本地坐标系统的坐标值；`convert...:toView:` 方法则将视图本地坐标系统的坐标值转换为指定视图坐标系统的坐标值。如果传入 `nil` 作为视图引用参数的值，则上面这些方法会将视图所在窗口的坐标系统作为转换的源或目标坐标系统。

除了 [UIView](#) 的转换方法之外，[UIWindow](#) 类也定义了几个转换方法。这些方法和 [UIView](#) 的版本类似，只是 [UIView](#) 定义的方法将视图本地坐标系统作为转换的源或目标坐标系统，而 [UIWindow](#) 的版本则使用窗口坐标系统。

[convertPoint:fromWindow:](#)

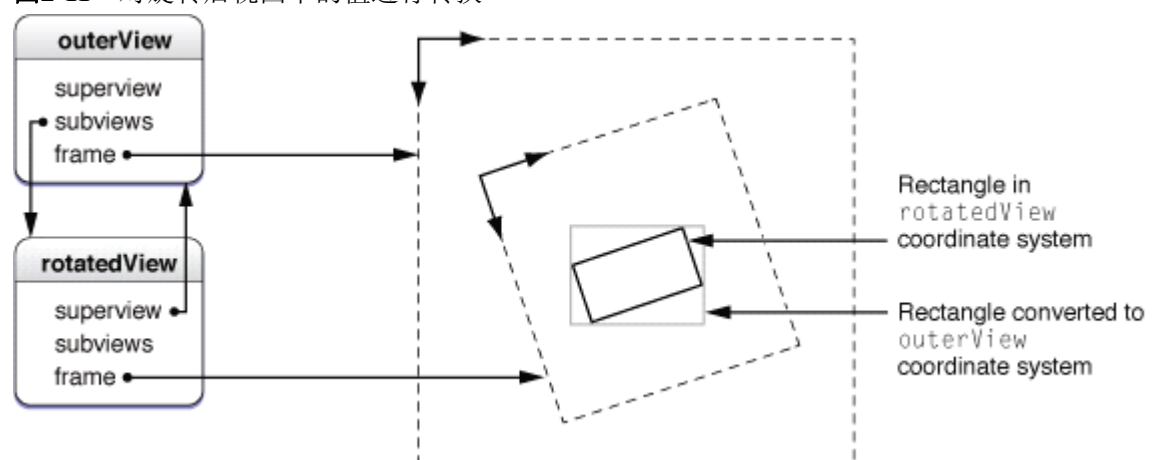
[convertRect:fromWindow:](#)

[convertPoint:toWindow:](#)

[convertRect:toWindow:](#)

当参与转换的视图没有被旋转，或者被转换的对象仅仅是点的时候，坐标转换相当直接。如果是在旋转之后的视图之间转换矩形或尺寸数据，则其几何结构必须经过合理的改变，才能得到正确的结果坐标。在对矩形结构进行转换时，[UIView](#) 类假定您希望保证原来的屏幕区域被覆盖，因此转换后的矩形会被放大，其结果是使放大后的矩形（如果放在对应的视图中）可以完全覆盖原来的矩形区域。图2-11显示了将 `rotatedView` 对象的坐标系统中的矩形转换到其超类（`outerView`）坐标系统的结果。

图2-11 对旋转后视图中的值进行转换



对于尺寸信息，[UIView](#) 简单地将它处理为分别相对于源视图和目标视图(0.0, 0.0)点的偏移量。虽然偏移量保持不变，但是相对于坐标轴的差额会随着视图的旋转而移动。在转换尺寸

数据时，UIKit 总是返回正的数值。

标识视图

UIView 类中包含一个 [tag](#) 属性。借助这个属性，您可以通过一个整数值来标识一个视图对象。您可以通过这个属性来唯一标识视图层次中的视图，以及在运行时进行视图的检索（基于 tag 标识的检索比您自行遍历视图层次要快）。tag 属性的缺省值为0。

您可以通过 UIView 的 [viewWithTag:](#) 方法来检索标识过的视图。该方法从消息的接收者自身开始，通过深度优先的方法来检索接收者的子视图。

在运行时修改视图

应用程序在接收用户输入时，需要通过调整自己的用户界面来进行响应。应用程序可能重新排列界面上的视图、刷新屏幕上模型数据已被改变的视图、或者装载一组全新的视图。在决定使用哪种技术时，要考虑您的用户界面，以及您希望实现什么。但是，如何初始化这些技术对于所有应用程序都是一样的。本章的下面部分将描述 这些技术，以及如何通过这些技术在运行时更新您的用户界面。

请注意：如果您需要了解 UIKit 如何在框架内部和您的定制代码之间转移事件和消息的背景信息，请在继续阅读本文之前查阅[“视图交互模型”](#)部分。

实现视图动画

动画为用户界面在不同状态之间的迁移过程提供流畅的视觉效果。在 iPhone OS 中，动画被广泛用于视图的位置调整、尺寸变化、甚至是 alpha 值的变化（以实现淡入淡出的效果）。动画支持对于制作易于使用的应用程序是至关重要的，因此，UIKit 直接将它集成到 [UIView](#) 类中，以简化动画的创建过程。

UIView 类定义了几个内在支持动画的[属性声明](#)——也就是说，当这些属性值发生变化时，视图为其变化过程提供内建的动画支持。虽然执行动画所需要的工作由 UIView 类自动完成，但您仍然必须在希望执行动画时通知视图。为此，您需要将改变给定属性的代码包装在一个动画块中。

动画块从调用 UIView 的 [beginAnimations:context:](#) 类方法开始，而以调用 [commitAnimations](#) 类方法作为结束。在这两个调用之间，您可以配置动画的参数和改变希望实行动画的属性值。一旦调用 [commitAnimations](#) 方法，UIKit 就会开始执行动画，即把给定属性从当前值到新值的变化过程用动画表现出来。动画块可以被嵌套，但是在最外层的动画块提交之前，被嵌套的动画不会被执行。

表2-2列举了 UIView 类中支持动画的属性。

表2-2

支持动画的属性

描述

frame	视图的边框矩形，位于父视图的坐标系中。
bounds	视图的边界矩形，位于视图的坐标系中。

center 边框的中心，位于父视图的坐标系中。
transform 视图上的转换矩阵，相对于视图边界的中心。
alpha 视图的 alpha 值，用于确定视图的透明度。

配置动画的参数

除了在动画块中改变属性值之外，您还可以对其它参数进行配置，以确定您希望得到的动画行为。为此，您可以调用下面这些 `UIView` 的类方法：

用 [setAnimationStartDate:](#) 方法来设置动画在 `commitAnimations` 方法返回之后的发生日期。缺省行为是使动画立即在动画线程中执行。

用 [setAnimationDelay:](#) 方法来设置实际发生动画和 `commitAnimations` 方法返回的时间点之间的间隔。

用 [setAnimationDuration:](#) 方法来设置动画持续的秒数。

用 [setAnimationCurve:](#) 方法来设置动画过程的相对速度，比如动画可能在启示阶段逐渐加速，而在结束阶段逐渐减速，或者整个过程都保持相同的速度。

用 [setAnimationRepeatCount:](#) 方法来设置动画的重复次数。

用 [setAnimationRepeatAutoreverses:](#) 方法来指定动画在到达目标值时是否自动反向播放。您可以结合使用这个方法和 `setAnimationRepeatCount:` 方法，使各个属性在初始值和目标值之间平滑切换一段时间。

`commitAnimations` 类方法在调用之后和动画开始之前立刻返回。`UIKit` 在一个独立的、和应用程序的主事件循环分离的线程中执行动画。`commitAnimations` 方法将动画发送到该线程，然后动画就进入线程中的队列，直到被执行。缺省情况下，只有在当前正在运行的动画块执行完成后，`Core Animation` 才会启动队列中的动画。但是，您可以通过向动画块中的 [setAnimationBeginsFromCurrentState:](#) 类方法传入 `YES` 来重载这个行为，使动画立即启动。这样做会停止当前正在执行的动画，而使新动画在当前状态下开始执行。

缺省情况下，所有支持动画的属性在动画块中发生的变化都会形成动画。如果您希望让动画块中发生的某些变化不产生动画效果，可以通过 [setAnimationsEnabled:](#) 方法来暂时禁止动画，在完成修改后才重新激活动画。在调用 `setAnimationsEnabled:` 方法并传入 `NO` 值之后，所有的改变都不会产生动画效果，直到用 `YES` 值再次调用这个方法或者提交整个动画块时，动画才会恢复。您可以用 `areAnimationsEnabled` 方法来确定当前是否激活动画。

配置动画的委托

您可以为动画块分配一个[委托](#)，并通过该委托接收动画开始和结束的消息。当您需要在动画开始前和结束后立即执行其它任务时，可能就需要这样做。您可以通过 `UIView` 的 [setAnimationDelegate:](#) 类方法来设置委托，并通过 [setAnimationWillStartSelector:](#) 和 [setAnimationDidStopSelector:](#) 方法来指定接收消息的选择器方法。消息处理方法的形式如下：

```
- (void)animationWillStart:(NSString *)animationID context:(void *)context;  
- (void)animationDidStop:(NSString *)animationID finished:(NSNumber *)finished context:(void *)context;
```

上面两个方法的 `animationID` 和 `context` 参数和动画块开始时传给 `beginAnimations:context:` 方法的参数相同：

`animationID` - 应用程序提供的字符串，用于标识一个动画块中的动画。

context - 也是应用程序提供的对象，用于向委托对象传递额外的信息。

`setAnimationDidStopSelector`: 选择器方法还有一个参数—即一个布尔值。如果动画顺利完成，没有被其它动画取消或停止，则该值为 YES。

响应布局的变化

任何时候，当视图的布局发生改变时，UIKit 会激活每个视图的自动尺寸调整行为，然后调用各自的 [layoutSubviews](#) 方法，使您有机会进一步调整子视图的几何尺寸。下面列举的情形都会引起视图布局的变化：

视图边界矩形的尺寸发生变化。

滚动视图的内容偏移量—也就是可视内容区域的原点—发生变化。

和视图关联的转换矩阵发生变化。

和视图层相关联的 Core Animation 子层组发生变化。

您的应用程序调用视图的 [setNeedsLayout](#) 或 [layoutIfNeeded](#) 方法来强制进行布局。

您的应用程序调用视图背后的层对象的 [setNeedsLayout](#) 方法来强制进行布局。

子视图的初始布局由视图的自动尺寸调整行为来负责。应用这些行为可以保证您的视图接近其设计的尺寸。有关自动尺寸调整行为如何影响视图的尺寸和位置的更多信息，请参见“[自动尺寸调整行为](#)”部分。

有些时候，您可能希望通过 `layoutSubviews` 方法来手工调整子视图的布局，而不是完全依赖自动尺寸调整行为。举例来说，如果您要实现一个由几个子视图元素组成的定制控件，则可以通过手工调整子视图来精确控制控件在一定尺寸范围内的外观。还有，如果一个视图表示的滚动内容区域很大，可以选择将内容显示为一组平铺的子视图，在滚动过程中，可以回收离开屏幕边界的视图，并在填充新内容后将它重新定位，使它成为下一个滚入屏幕的视图。

请注意：您也可以使用 `layoutSubviews` 方法来调整作为子层链接到视图层的定制 [CALayer](#) 对象。您可以通过对隐藏在视图后面的层层次进行管理，实现直接基于 Core Animation 的高级动画。有关如何通过 Core Animation 管理层次的信息，请参见 [Core Animation 编程指南](#)。

在编写布局代码时，请务必在应用程序支持的每个方向上都进行测试。对于同时支持景观方向和肖像方向的应用程序，必须确认其是否能正确处理两个方向上的布局。类似地，您的应用程序应该做好处理其它系统变化的准备，比如状态条高度的变化，如果用户在使用您的应用程序的同时接打电话，然后再挂断，就会发生这种变化。在挂断时，负责管理视图的[视图控制器](#)可能会调整视图的尺寸，以适应缩小的状态条。之后，这样的变化会向下渗透到应用程序的其它视图。

重画视图的内容

有些时候，应用程序数据模型的变化会影响到相应的用户界面。为了反映这些变化，您可以将相应的视图标识为需要刷新（通过调用 [setNeedsDisplay](#) 或 [setNeedsDisplayInRect](#) 方法）。和简单创建一个图形上下文并进行描画相比，将视图标识为需要刷新的方法使系统有机会更有效地执行描画操作。举例来说，如果您在某个运行周期中将一个视图的几个区域标识为需要刷新，系统就会将这些需要刷新的区域进行合并，并最终形成一个 `drawRect` 方法的调用。结果，只需要创建一个图形上下文就可以描画所有这些受影响的区域。这个做法比连续快速创建几个图形上下文要有效得多。

实现 `drawRect:` 方法的视图总是需要检查传入的矩形参数，并用它来限制描画操作的范围。因为描画是开销相对昂贵的操作，以这种方式来限制描画是提高性能的好方法。

缺省情况下，视图在几何上的变化并不自动导致重画。相反，大多数几何变化都由 `Core Animation` 来自动处理。具体来说，当您改变视图的 `frame`、`bounds`、`center`、或 `transform` 属性时，`Core Animation` 会将相应的几何变化应用到与视图层相关联的缓存位图上。在很多情况下，这种方法是完全可以接受的，但是如果您发现结果不是您期望得到的，则可以强制 `UIKit` 对视图进行重画。为了避免 `Core Animation` 自动处理几何变化，您可以将视图的 `contentMode` 属性声明设置为 `UIViewContentModeRedraw`。更多有关内容模式的信息，请参见“[内容模式和比例缩放](#)”部分。

隐藏视图

您可以通过改变视图的 [hidden 属性声明](#) 来隐藏或显示视图。将这个属性设置为 YES 会隐藏视图，设置为 NO 则可以显示视图。对一个视图进行隐藏会同时隐藏其内嵌的所有子视图，就好像它们自己的 `hidden` 属性也被设置一样。

当您隐藏一个视图时，该视图仍然会保留在视图层次中，但其内容不会被描画，也不会接收任何触摸事件。由于隐藏视图仍然存在于视图层次中，所以会继续参与自动尺寸调整和其它布局操作。如果被隐藏的视图是当前的第一响应者，则该视图会自动放弃其自动响应者的状态，但目标为第一响应者的事件仍然会传递给隐藏视图。有关响应者链的更多信息，请参见“[响应者对象和响应者链](#)”部分。

创建一个定制视图

`UIView` 类为在屏幕上显示内容及处理触摸事件提供了潜在的支持，但是除了在视图区域内描画带有 `alpha` 值的背景色之外，`UIView` 类的实例不做其它描画操作，包括其子视图的描画。如果您的应用程序需要显示定制的内容，或以特定的方式处理触摸事件，必须创建 `UIView` 的定制子类。

本章的下面部分将描述一些定制视图对象可能需要实现的关键方法和行为。有关子类化的更多信息，请参见 [UIView 类参考](#)。

初始化您的定制视图

您定义的每个新的视图对象都应该包含 [initWithFrame:初始化](#) 方法。该方法负责在创建对象时对类进行初始化，使之处于已知的状态。在通过代码创建您的视图实例时，需要使用这个方法。

程序清单2-2显示了标准的 `initWithFrame:` 方法的一个框架实现。该实现首先调用继承自超类的实现，然后初始化类的实例变量和状态信息，最后返回初始化完成的对象。您通常需要首先执行超类的实现，以便在出现问题时可以简单地终止自己的初始化代码，返回 `nil`。

程序清单2-2 初始化一个视图的子类

```
- (id)initWithFrame:(CGRect)aRect {  
    self = [super initWithFrame:aRect];
```

```

        if (self) {
            // setup the initial properties of
the view
            ...
        }
        return self;
    }
}

```

如果您从 [nib 文件](#) 中装载定制视图类的实例，则需要知道：在 iPhone OS 中，装载 nib 的代码并不通过 `initWithFrame:` 方法来实例化新的视图对象，而是通过 [NSCoding 协议](#) 定义的 `initWithCoder:` 方法来进行。

即使您的视图采纳了 NSCoding 协议，Interface Builder 也不知道它的定制属性，因此不知道如何将那些属性编码到 nib 文件中。所以，当您从 nib 文件装载定制视图时，`initWithCoder:` 方法不具有进行正确初始化所需要的信息。为了解决这个问题，您可以在自己的类中实现 [awakeFromNib](#) 方法，特别用于从 nib 文件装载的定制类。

描画您的视图内容

当您改变视图内容时，可以通过 [setNeedsDisplay](#) 或 [setNeedsDisplayInRect:](#) 方法来将需要重画的部分通知给系统。在应用程序返回运行循环之后，会对所有的描画请求进行合并，计算界面中需要被更新的部分；之后就开始遍历视图层次，向需要更新的视图发送 [drawRect:](#) 消息。遍历的起点是视图层次的根视图，然后从后往前遍历其子视图。在可视边界内显示定制内容的视图必须实现其 `drawRect:` 方法，以便对该内容进行渲染。

在调用视图的 `drawRect:` 方法之前，UIKit 会为其配置描画的环境，即创建一个图形上下文，并调整其坐标系统和裁剪区，使之和视图的坐标系统及边界相匹配。因此，在您的 `drawRect:` 方法被调用时，您可以使用 UIKit 的类和函数、Quartz 的函数、或者使用两者相结合的方法来直接进行描画。需要的话，您可以通过 [UIGraphicsGetCurrentContext](#) 函数来取得当前图形上下文的指针，实现对它的访问。

重要提示：只有当定制视图的 `drawRect:` 方法被调用的期间，当前图形上下文才是有效的。UIKit 可能为该方法的每个调用创建不同的图形上下文，因此，您不应该对该对象进行缓存并在之后使用。

程序清单2-3显示了 `drawRect:` 方法的一个简单实现，即在视图边界描画一个10像素宽的红色边界。由于 UIKit 描画操作的实现也是基于 Quartz，所以您可以像下面这样混合使用不同的描画调用来得到期望的结果。

程序清单2-3 一个描画方法

```

- (void)drawRect:(CGRect)rect {
    CGContextRef          context          =
UIGraphicsGetCurrentContext();
    CGRect                myFrame = self.bounds;

    CGContextSetLineWidth(context, 10);

    [[UIColor redColor] set];
    UIRectFrame(myFrame);
}

```

```
}
```

如果您能确定自己的描画代码总是以不透明的内容覆盖整个视图的表面，则可以将视图的 [opaque 属性声明](#) 设置为 YES，以提高描画代码的总体效率。当您视图标识为不透明时，UIKit 会避免对该视图正下方的内容进行描画。这不仅减少了描画开销的时间，而且减少内容合成需要的工作。然而，只有当您能确定视图提供的内容为不透明时，才能将这个属性设置为 YES；如果您不能保证视图内容总是不透明，则应该将它设置为 NO。

提高描画性能（特别是在滚动过程）的另一个方法是将视图的 [clearsContextBeforeDrawing](#) 属性设置为 NO。当这个属性被设置为 YES 时，UIKit 会在调用 `drawRect:` 方法之前，把即将被该方法更新的区域填充为透明的黑色。将这个属性设置为 NO 可以取消相应的填充操作，而由应用程序负责完全重画传给 `drawRect:` 方法的更新矩形中的部分。这样的优化在滚动过程中通常是一个好的折衷。

响应事件

UIView 类是 [UIResponder](#) 的一个子类，因此能够接收用户和视图内容交互时产生的触摸事件。触摸事件从发生触摸的视图开始，沿着响应者链进行传递，直到最后被处理。视图本身就是响应者，是响应者链的参与者，因此可以收到所有关联子视图派发给它们的触摸事件。处理触摸事件的视图通常要实现下面的所有方法，更多细节请参见“[事件处理](#)”部分：

[touchesBegan:withEvent:](#)

[touchesMoved:withEvent:](#)

[touchesEnded:withEvent:](#)

[touchesCancelled:withEvent:](#)

请记住，在缺省情况下，视图每次只响应一个触摸动作。如果用户将第二个手指放在屏幕上，系统会忽略该触摸事件，而不会将它报告给视图对象。如果您希望在视图的事件处理器方法中跟踪多点触摸手势，则需要重新激活多点触摸事件，具体方法是将视图的 [multipleTouchEnabled 属性声明](#) 设置为 YES。

某些视图，比如标签和图像视图，在初始状态下完全禁止事件处理。您可以通过改变视图的 [userInteractionEnabled](#) 属性值来控制视图是否可以对事件进行处理。当某个耗时很长的操作被挂起时，您可以暂时将这个属性设置为 NO，使用户无法对视图的内容进行操作。为了阻止事件到达您的视图，还可以使用 [UIApplication](#) 对象的 [beginIgnoringInteractionEvents](#) 和 [endIgnoringInteractionEvents](#) 方法。这些方法影响的是整个应用程序的事件分发，而不仅仅是某个视图。

在处理触摸事件时，UIKit 会通过 UIView 的 [hitTest:withEvent:](#) 和 [pointInside:withEvent:](#) 方法来确定触摸事件是否发生在指定的视图上。虽然很少需要重载这些方法，但是您可以通过重载来使子视图无法处理触摸事件。

视图对象的清理

如果您的视图类[分配](#)了任何内存、存储了任何对象的引用、或者持有在释放视图时也需要[被释放](#)的资源，则必须实现其 [dealloc](#) 方法。当您的视图对象的保持数为零、且视图本身即将被解除分配时，系统会调用其 [dealloc](#) 方法。您在这个方法的实现中应该释放视图持有的对象和资源，然后调用超类的实现，如程序清单2-4所示。

程序清单2-4 实现 dealloc 方法


```
- (void)dealloc {  
    // Release a retained UIColor object  
    [color release];  
  
    // Call the inherited implementation  
    [super dealloc];  
}
```

事件处理

本章将描述 iPhone OS 系统中的事件类型，并解释如何处理这些事件。文中还将讨论如何在应用程序内部或不同应用程序间通过 [UIPasteboard](#) 类提供的设施进行数据的拷贝和粘贴，该类是 iPhone OS 3.0 引入的。

iPhone OS 支持两种类型的事件：即触摸事件或运动事件。在 iPhone OS 3.0 中，[UIEvent](#) 类已经被扩展为不仅可以包含触摸事件和运动事件，还可以容纳将来可能引入的其它事件类型。每个事件都有一个与之关联的事件类型和子类型，可以通过 UIEvent 的 [type](#) 和 [subtype](#) [属性声明](#) 进行访问，类型既包括触摸事件，也包括运动事件。在 iPhone OS 3.0 上，子类型只有一种，即摇摆-运动子类型（[UIEventSubtypeMotionShake](#)）。

触摸事件

iPhone OS 中的触摸事件基于多点触摸模型。用户不是通过鼠标和键盘，而是通过触摸设备的屏幕来操作对象、输入数据、以及指示自己的意图。iPhone OS 将一个或多个和屏幕接触的手指识别为 **多点触摸序列** 的一部分，该序列从第一个手指碰到屏幕开始，直到最后一个手指离开屏幕结束。iPhone OS 通过一个多点触摸序列来跟踪与屏幕接触的手指，记录每个手指的触摸特征，包括手指在屏幕上的位置和发生触摸的时间。应用程序通常将特定组合的触摸识别为手势，并以用户直觉的方式来进行响应，比如对收缩双指距离的手势，程序的响应是缩小显示的内容；对轻拂屏幕的手势，则响应为滚动显示内容。

请注意：手指在屏幕上能达到的精度和鼠标指针有很大的不同。当用户触击屏幕时，接触区域实际上是椭圆形的，而且比用户想像的位置更靠下一点。根据触摸屏幕的手指、手指的尺寸、手指接触屏幕的力量、手指的方向、以及其它因素的不同，其“接触部位”的尺寸和形状也有所不同。底层的多点触摸系统会分析所有的这些信息，为您计算出单一的触点。很多 UIKit 类对多点触摸事件的处理方式不同于它的对象实例，特别是像 [UIButton](#) 和 [UISlider](#) 这样的 [UIControl](#) 的子类。这些子类的对象—被称为控件对象—只接收特定类型的手势，比如触击或向特定方向拖拽。控件对象在正确配置之后，会在某种手势发生后将动作消息发送给目标对象。其它的 UIKit 类则在其它的上下文中处理手势，比如 [UIScrollView](#) 可以为表格视图和具有很大内容区域的文本视图提供滚动行为。

某些应用程序可能不需要直接处理事件，它们可以依赖 UIKit 类实现的行为。但是，如果您创建了 [UIView](#) 的 [定制子类](#)—这是 iPhone OS 系统开发的常见模式—且希望该视图响应特定的触摸事件，就需要实现处理该事件所需要的代码。而且，如果您希望一个 UIKit 对象以不同的方式响应事件，就必须创建框架类的子类，并重载相应的事件处理方法。

事件和触摸

在 iPhone OS 中，**触摸**动作是指手指碰到屏幕或在屏幕上移动，它是一个**多点触摸序列**的一部分。比如，一个 pinch-close 手势就包含两个触摸动作：即屏幕上的两个手指从相反方向靠近对方。一些单指手势则比较简单，比如触击、双击、或 轻拂（即用户快速碰擦屏幕）。应用程序也可以识别更为复杂的手势，举例来说，如果一个应用程序使用具有转盘形状的定制控件，用户就需要用多个手指来“转动”转盘，以便进行某种精调。

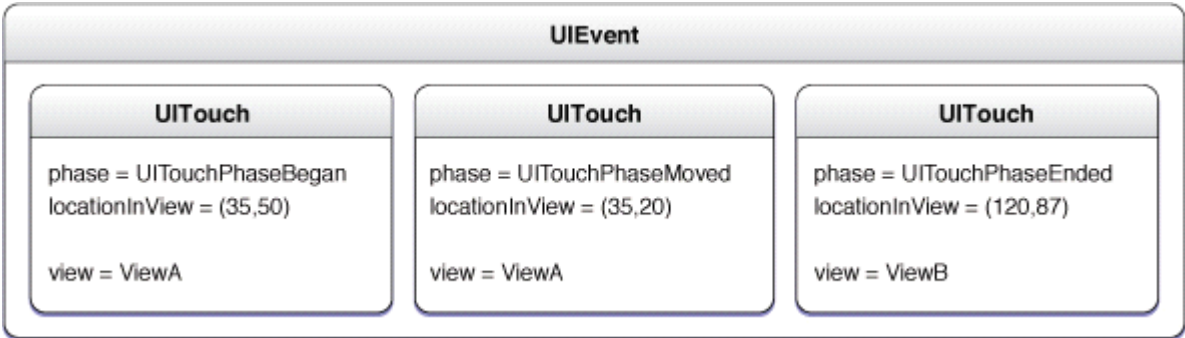
事件是当用户手指触击屏幕及在屏幕上移动时，系统不断 发送给应用程序的对象。事件对象为一个多点触摸序列中所有触摸动作提供一个快照，其中最重要的是特定视图中新发生或有变化的触摸动作。一个多点触摸序列从 第一个手指碰到屏幕开始，其它手指随后也可能触碰屏幕，所有手指都可能在屏幕上移动。当最后一个手指离开屏幕时，序列就结束了。在触摸的每个阶段，应用程序都会收到事件对象。

触摸信息有时间和空间两方面，时间方面的信息称为**阶段（phrase）**，表示触摸是否刚刚开始、是否正在移动或处于静止状态，以及何时结束——也就是手指何时从屏幕举起（参见图 3-1）。 触摸信息还包括当前在视图或窗口中的位置信息，以及之前的位置信息（如果有的话）。当一个手指接触屏幕时，触摸就和某个窗口或视图关联在一起，这个关联在 事件的整个生命周期都会得到维护。如果有多个触摸同时发生，则只有和同一个视图相关联的触摸会被一起处理。类似地，如果两个触摸事件发生的间隔时间很短， 也只有当它们和同一个视图相关联时，才会被处理为多触击事件。

图3-1 多点触摸序列和触摸阶段

在 iPhone OS 中，一个 [UITouch](#) 对象表示一个触摸，一个 [UIEvent](#) 对象表示一个事件。事件对象中包含与当前多点触摸序列相对应的所有触摸对象，还可以提供与特定视图或窗口相关联的触摸对象（参见图3-2）。在一个触摸序列发生的过程中，对应于特定手指的触摸对象是持久的，在跟踪手指运动的过程中，UIKit 会对其进行修改。发生改变的触摸**属性变量**有触摸阶段、触摸在视图中的位置、发生变化之前的位置、以及时间戳。事件处理代码通过检查这些属性的值来确定如何响应事件。

图3-2 UIEvent 对象及其 UITouch 对象间的关系



系统可能随时取消多点触摸序列，进行事件处理的应用程序必须做好正确响应的准备。事件的取消可能是由于重载系统事件引起的，电话呼入就是这样的例子。

事件的传递

系统将事件按照特定的路径传递给可以对其进行处理的对象。如“[核心应用程序架构](#)”部分描述的那样，当用户触摸设备屏幕时，iPhone OS 会将它识别为一组触摸对象，并将它们封装在一个 UIEvent 对象中，放入当前应用程序的事件队列中。事件对象将特定时刻的多点触摸

序列封装为一些触摸对象。负责管理应用程序的 [UIApplication](#) 单件对象将事件从队列的顶部取出，然后派发给其它对象进行处理。典型情况下，它会将事件发送给应用程序的键盘焦点窗口——即拥有当前用户事件焦点的窗口，然后代表该窗口的 [UIWindow](#) 对象再将它发送给第一响应者进行处理（第一响应者在“响应者对象和响应者链”部分中描述）。

应用程序通过触碰测试（hit-testing）来寻找事件的第一响应者，即通过递归调用视图层次中视图对象的 [hitTest:withEvent:](#) 方法来确认发生触摸的子视图。触摸对象的整个生命周期都和该视图互相关联，即使触摸动作最终移动到该视图区域之外也是如此。[“事件处理技巧”](#) 部分对触碰测试在编程方面的一些隐含意义进行讨论。

[UIApplication](#) 对象和每个 [UIWindow](#) 对象都在 [sendEvent:](#) 方法（两个类都声明了这个方法）中派发事件。由于这些方法是事件进入应用程序的通道，所以，您可以从 [UIApplication](#) 或 [UIWindow](#) 派生出子类，重载其 [sendEvent:](#) 方法，实现对事件的监控或执行特殊的事件处理。但是，大多数应用程序都不需要这样做。

响应者对象和响应者链

响应者对象是可以响应事件并对其进行处理的对象。[UIResponder](#) 是所有响应者对象的基类，它不仅为事件处理，而且也为常见的响应者行为定义编程接口。[UIApplication](#)、[UIView](#)、和所有从 [UIView](#) 派生出来的 [UIKit](#) 类（包括 [UIWindow](#)）都直接或间接地继承自 [UIResponder](#) 类。

第一响应者是应用程序中当前负责接收触摸事件的响应者对象（通常是一个 [UIView](#) 对象）。[UIWindow](#) 对象以消息的形式将事件发送给第一响应者，使其有机会首先处理事件。如果第一响应者没有进行处理，系统就将事件（通过消息）传递给响应者链中的下一个响应者，看看它是否可以进行处理。

响应者链是一系列链接在一起的响应者对象，它允许响应者对象将处理事件的责任传递给其它更高级别的对象。随着应用程序寻找能够处理事件的对象，事件就在响应者链中向上传递。响应者链由一系列“下一个响应者”组成，其顺序如下：

第一响应者将事件传递给它的视图控制器（如果有的话），然后是它的父视图。

类似地，视图层次中的每个后续视图都首先传递给它的视图控制器（如果有的话），然后是它的父视图。

1 最上层的容器视图将事件传递给 [UIWindow](#) 对象。

[UIWindow](#) 对象将事件传递给 [UIApplication](#) 单件对象。

如果应用程序找不到能够处理事件的响应者对象，则丢弃该事件。

响应者链中的所有响应者对象都可以实现 [UIResponder](#) 的某个事件处理方法，因此也都可以接收事件消息。但是，它们可能不愿处理或只是部分处理某些事件。如果是那样的话，它们可以将事件消息转送给下一个响应者，方法大致如下：

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    UITouch* touch = [touches anyObject];
    NSUInteger numTaps = [touch tapCount];
    if (numTaps < 2) {
        [self.nextResponder touchesBegan:touches
withEvent:event];
    } else {
        [self handleDoubleTap:touch];
    }
}
```

```
}  
}
```

请注意：如果一个响应者对象将一个多点触摸序列的初始阶段的事件处理消息转发给下一个响应者（在 [touchesBegan:withEvent:](#) 方法中），就应该同样转发该序列的其它事件处理消息。动作消息的处理也使用响应者链。当用户对诸如按键或分页控件这样的 [UIControl](#) 对象进行操作时，控件对象（如果正确配置的话）会向目标对象发送动作消息。但是，如果目标对象被指定为 nil，应用程序就会像处理事件消息那样，把该动作消息路由给第一响应者。如果第一响应者没有进行处理，再发送给其下一个响应者，以此类推，将消息沿着响应者链向上传递。

调整事件的传递

UIKit 为应用程序提供了一些简化事件处理、甚至完全关闭事件流的编程接口。下面对这些方法进行总结：

关闭事件的传递。缺省情况下，视图会接收触摸事件。但是，您可以将其 [userInteractionEnabled](#) 属性声明设置为 NO，关闭事件传递的功能。隐藏或透明的视图也不能接收事件。

在一定的时间内关闭事件的传递。应用程序可以调用 UIApplication 的 [beginIgnoringInteractionEvents](#) 方法，并在随后调用 [endIgnoringInteractionEvents](#) 方法来实现这个目的。前一个方法使应用程序完全停止接收触摸事件消息，第二个方法则重启消息的接收。某些时候，当您的代码正在执行动画时，可能希望关闭事件的传递。

打开多点触摸的传递。缺省情况下，视图只接收多点触摸序列的第一个触摸事件，而忽略所有其它事件。如果您希望视图处理多点触摸，就必须使它启用这个功能。在代码或 Interface Builder 的查看器窗口中将视图的 [multipleTouchEnabled](#) 属性设置为 YES，就可以实现这个目标。

将事件传递限制在某个单独的视图上。缺省情况下，视图的 [exclusiveTouch](#) 属性被设置为 NO。将这个属性设置为 YES 会使相应的视图具有这样的特性：即当该视图正在跟踪触摸动作时，窗口中的其它视图无法同时进行跟踪，它们不能接收到那些触摸事件。然而，一个标识为“独占触摸”的视图不能接收与同一窗口中其它视图相关联的触摸事件。如果一个手指接触到一个独占触摸的视图，则仅当该视图是窗口中唯一一个跟踪手指的视图时，触摸事件才会被传递。如果一个手指接触到一个非独占触摸的视图，则仅当窗口中没有其它独占触摸视图跟踪手指时，该触摸事件才会被传递。

将事件传递限制在子视图上。一个定制的 UIView 类可以通过重载 [hitTest:withEvent:](#) 方法将多点触摸事件的传递限制在它的子视图上。这个技巧的讨论请参见“[事件处理技巧](#)”部分。

处理多点触摸事件

为了处理多点触摸事件，[UIView](#) 的定制子类（比较不常见的还有 [UIApplication](#) 或 [UIWindow](#) 的定制子类）必须至少实现一个 [UIResponder](#) 的事件处理方法。本文的下面部分将对这些方法进行描述，讨论处理常见手势的方法，并展示一个处理复杂多点触摸事件的响应者对象实例，以及就事件处理的某些技术提出建议。

事件处理方法

在一个多点触摸序列发生的过程中，应用程序会发出一系列事件消息。为了接收和处理这些消息，响应者对象的类必须至少实现下面这些由 `UIResponder` 类声明的方法之一：

- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event;
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event;
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event;
- (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event

在给定的触摸阶段中，如果发生新的触摸动作或已有的触摸动作发生变化，应用程序就会发送这些消息：

当一个或多个手指触碰屏幕时，发送 [touchesBegan:withEvent:](#) 消息。

当一个或多个手指在屏幕上移动时，发送 [touchesMoved:withEvent:](#) 消息。

当一个或多个手指离开屏幕时，发送 [touchesEnded:withEvent:](#) 消息。

当触摸序列被诸如电话呼入这样的系统事件所取消时，发送 [touchesCancelled:withEvent:](#) 消息。

上面这些方法都和特定的触摸阶段（比如 [UITouchPhaseBegan](#)）相关联，该信息存在于 [UITouch](#) 对象的 [phase 属性声明](#) 中。

每个与事件处理方法相关联的消息都有两个参数。第一个参数是一个 [UITouch](#) 对象的集合，表示给定阶段中新的或者发生变化的触摸动作；第二个参数是一个 [UIEvent](#) 对象，表示这个特定的事件。您可以通过这个事件对象得到与之相关联的所有触摸对象（[allTouches](#)），或者发生在特定的视图或窗口上的触摸对象子集。其中的某些触摸对象表示自上次事件消息以来没有发生变化，或虽然发生变化但处于不同阶段的触摸动作。

为了处理给定阶段的事件，响应者对象常常从传入的集合参数中取得一或多个 [UITouch](#) 对象，然后考察这些对象的属性或取得它们的位置（如果需要处理所有触摸对象，可以向该 [NSSet](#) 对象发送 [anyObject](#) 消息）。[UITouch](#) 类中有一个名为 [locationInView:](#) 的重要方法，如果传入 `self` 参数值，它会给出触摸动作在响应者坐标系统中的位置（假定该响应者是一个 [UIView](#) 对象，且传入的视图参数不为 `nil`）。另外，还有一个与之平行的方法，可以给出触摸动作之前位置（[previousLocationInView:](#)）。[UITouch](#) 实例的属性还可以给出发生多少次触碰（[tapCount](#)）、触摸对象的创建或最后一次变化发生在什么时间（[timestamp](#)）、以及触摸处于什么阶段（[phase](#)）。

响应者类并不是必须实现上面列出的所有三个事件方法。举例来说，如果它只对手指离开屏幕感兴趣，则只需要实现 [touchesEnded:withEvent:](#) 方法就可以了。

在一个多点触摸序列中，如果响应者在处理事件时创建了某些持久对象，则应该实现 [touchesCancelled:withEvent:](#) 方法，以便当系统取消该序列的时候对其进行清理。多点触摸序列的取消常常发生在应用程序的事件处理遭到外部事件——比如电话呼入——破坏的时候。请注意，响应者对象同样应该在收到多点触摸序列的 [touchesEnded:withEvent:](#) 消息时清理之前创建的对象（“[事件处理技巧](#)”部分讨论了如何确定一个序列中的最后一个 touch-up 事件）。

处理单个和多个触碰手势

iPhone 应用程序中一个很常见的手势是触击：即用户用手指触碰一个对象。响应者对象可以以一种方式响应单击，而以另外一种方式响应双击，甚至可能以第三种方式响应三次触击。您可以通过考察 [UITouch](#) 对象的 [tapCount 属性声明](#) 值来确定用户在一个响应者对象上的触击次数，

取得这个值的最好地方是 [touchesBegan:withEvent:](#) 和 [touchesEnded:withEvent:](#) 方法。在很多情况下，我们更倾向于后者，因为它与用户手指离开屏幕的阶段相对应。在触摸结束阶段（

[UITouchPhaseEnded](#)) 考察触击的次数可以确定手指是真的触击，而不是其它动作，比如手指接触屏幕后拖动的动作。

程序清单3-1展示了如何检测某个视图上是否发生双击。

程序清单3-1 检测双击手势

```
- (void) touchesEnded:(NSSet*)touches withEvent:(UIEvent*)event
{
    UITouch          *touch = [touches anyObject];

    if ([touch tapCount] == 2) {
        CGPoint tapPoint = [theTouch locationInView:self];
        // Process a double-tap gesture
    }
}
```

当一个响应者对象希望以不同的方式响应单击和双击事件时，就会出现复杂的情况。举例来说，单击的结果可能是选定一个对象，而双击则可能是显示一个编辑视图，用于编辑被双击的对象。那么，响应者对象如何知道一个单击不是另一个双击的起始部分呢？我们接下来解释响应者对象如何借助上文刚刚描述的事件处理方法来处理这种情况：

在 `touchesEnded:withEvent:` 方法中，当触击次数为一时，响应者对象就向自身发送一个 [performSelector:withObject:afterDelay:](#) 消息，其中的 [选择器](#) 标识由响应者对象实现的、用于处理单击手势的方法；第二个参数是一个 [NSValue](#) 或 [NSDictionary](#) 对象，用于保存相关的 `UITouch` 对象；时延参数则表示单击和双击手势之间的合理时间间隔。

请注意：使用一个 `NSValue` 对象或字典来保存触摸对象是因为它们会保持传入的对象。然而，您自己在进行事件处理时，不应该对 `UITouch` 对象进行保持。

在 `touchesBegan:withEvent:` 方法中，如果触击次数为二，响应者对象会向自身发送一个 [cancelPreviousPerformRequestsWithTarget:](#) 消息，取消当前被挂起和延期执行的调用。如果触碰次数不为二，则在指定的延时之后，先前步骤中由选择器标识的方法就会被调用，以处理单击手势。

在 `touchesEnded:withEvent:` 方法中，如果触碰次数为二，响应者会执行处理双击手势的代码。

检测碰擦手势

水平和垂直的碰擦 (Swipe) 是简单的手势类型，您可以简单地自己的代码中进行跟踪，并通过它们执行某些动作。为了检测碰擦手势，您需要跟踪用户手指在期望的坐标轴方向上的运动。碰擦手势如何形成是由您自己来决定的，也就是说，您需要确定用户手指移动的距离是否足够长，移动的轨迹是否足够直，还有移动的速度是否足够快。您可以保存初始的触碰位置，并将它和后续的 `touch-moved` 事件报告的位置进行比较，进而做出这些判断。程序清单3-2展示了一些基本的跟踪方法，可以用于检测某个视图上发生的水平碰擦。在这个例子中，视图将触摸的初始位置存储在名为 `startTouchPosition` 的成员变量中。随着用户手指的移动，清单中的代码将当前的触摸位置和起始位置进行比较，确定是否为碰擦手势。如果触摸在垂直方向上移动得太远，就会被认为不是碰擦手势，并以不同的方式进行处理。但是，如果手指继续在水平方向上移动，代码就继续将它作为碰擦手势来处理。一旦碰擦手势在水平方向移动得足够远，以至于可以认为是完整的手势时，处理例程就会触发相应的动作。检测垂直方向上的碰擦手势可以用类似的代码，只是把 `x` 和 `y` 方向的计算互换一下就

可以了。

程序清单3-2 在视图中跟踪碰擦手势

```
#define HORIZ_SWIPE_DRAG_MIN 12
#define VERT_SWIPE_DRAG_MAX 4

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

- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [touches anyObject];
    CGPoint currentTouchPosition = [touch locationInView:self];

    // If the swipe tracks correctly.
    if (fabsf(startTouchPosition.x - currentTouchPosition.x) >= HORIZ_SWIPE_DRAG_MIN
    &&
        fabsf(startTouchPosition.y - currentTouchPosition.y) <=
    VERT_SWIPE_DRAG_MAX)
    {
        // It appears to be a swipe.
        if (startTouchPosition.x < currentTouchPosition.x)
            [self myProcessRightSwipe:touches withEvent:event];
        else
            [self myProcessLeftSwipe:touches withEvent:event];
    }
    else
    {
        // Process a non-swipe event.
    }
}
```

处理复杂的多点触摸序列

触击和碰擦是简单的手势。如何处理更为复杂的多点触摸序列——实际上是解析应用程序特有的手势——取决于应用程序希望完成的具体目标。您可以跟踪所有阶段的所有触摸动作，记录触摸对象中发生变化的[属性变量](#)，并正确地改变内部的状态。

说明如何处理复杂的多点触摸序列的最好方法是通过实例。程序清单 3-3 展示一个定制的 [UIView](#) 对象如何通过动画移动“Welcome”标语牌来响应用户手指的移动，以及如何通过改变欢迎标语的语言来响应用户的双击手势（例子中的代码来自一个名为 [MoveMe](#) 的示例工程，进一步考察该工程可以更好地理解事件处理的上下文）。

程序清单3-3 处理复杂的多点触摸序列

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [[event allTouches] anyObject];
    // Only move the placard view if the touch was in the placard view
    if ([touch view] != placardView) {
        // On double tap outside placard view, update placard's display string
        if ([touch tapCount] == 2) {
            [placardView setupNextDisplayString];
        }
        return;
    }
    // "Pulse" the placard view by scaling up then down
    // Use UIView's built-in animation
    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDuration:0.5];
    CGAffineTransform transform = CGAffineTransformMakeScale(1.2, 1.2);
    placardView.transform = transform;
    [UIView commitAnimations];

    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDuration:0.5];
    transform = CGAffineTransformMakeScale(1.1, 1.1);
    placardView.transform = transform;
    [UIView commitAnimations];

    // Move the placardView to under the touch
    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDuration:0.25];
    placardView.center = [self convertPoint:[touch locationInView:self]
fromView:placardView];
    [UIView commitAnimations];
}

- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [[event allTouches] anyObject];

    // If the touch was in the placardView, move the placardView to its location
    if ([touch view] == placardView) {
        CGPoint location = [touch locationInView:self];
        location = [self convertPoint:location fromView:placardView];
        placardView.center = location;
    }
}
```

```

        return;
    }
}

- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [[event allTouches] anyObject];

    // If the touch was in the placardView, bounce it back to the center
    if ([touch view] == placardView) {
        // Disable user interaction so subsequent touches don't interfere with animation
        self.userInteractionEnabled = NO;
        [self animatePlacardViewToCenter];
        return;
    }
}

```

请注意：对于通过描画自身的外观来响应事件的定制视图，在事件处理方法中通常应该只是设置描画状态，而在 `drawRect:` 方法中执行所有的描画操作。如果需要了解更多关于描画视图内容的方法，请参见[“图形和描画”](#)部分。

事件处理技巧

下面是一些事件处理技巧，您可以在自己的代码中使用。

跟踪 `UITouch` 对象的变化

在事件处理代码中，您可以将触摸状态的相关位置保存下来，以便在必要时和变化之后的 [UITouch](#) 实例进行比较。作为例子，假定您希望将每个触摸对象的最后位置和其初始位置进行比较，则在 [touchesBegan:withEvent:](#) 方法中，您可以通过 [locationInView:](#) 方法得到每个触摸对象的初始位置，并以 `UITouch` 对象的地址作为键，将它们存储在 [CFDictionaryRef](#) 封装类型中；然后，在 [touchesEnded:withEvent:](#) 方法中，可以通过传入 `UITouch` 对象的地址取得该对象的初始位置，并将它和当前位置进行比较（您应该使用 `CFDictionaryRef` 类型，而不是 [NSDictionary 对象](#)，因为后者需要对其存储的项目进行拷贝，而 `UITouch` 类并不采纳 [NSCopying](#) 协议，该协议在对象拷贝过程中是必须的）。

对子视图或层上的触摸动作进行触碰测试

定制视图可以用 `UIView` 的 [hitTest:withEvent:](#) 方法或 `CALayer` 的 [hitTest:](#) 方法来寻找接收触摸事件的子视图或层，进而正确地处理事件。下面的例子用于检测定制视图的层中的“Info”图像是否被触碰。

```

- (void)touchesEnded:(NSSet*)touches withEvent:(UIEvent*)event {
    CGPoint location = [[touches anyObject] locationInView:self];
    CALayer *hitLayer = [[self layer] hitTest:[self convertPoint:location
fromView:nil]];

    if (hitLayer == infoImage) {
        [self displayInfo];
    }
}

```

```
}  
}
```

如果您有一个携带子视图的定制视图，就需要明确自己是希望在子视图的级别上处理触摸事件，还是在父视图的级别上进行处理。如果子视图没有实现 [touchesBegan:withEvent:](#)、[touchesEnded:withEvent:](#)、或者 [touchesMoved:withEvent:](#) 方法，则这些消息就会沿着响应者链被传播到父视图。然而，由于多次触碰和多点触摸事件与发生这些动作所在的子视图是互相关联的，所以父视图不会接收到这些事件。为了保证能接收到所有的触摸事件，父视图必须重载 [hitTest:withEvent:](#) 方法，并在其中返回其本身，而不是它的子视图。

确定多点触摸序列中最后一个手指何时离开

当您希望知道一个多点触摸序列中的最后一个手指何时从视图离开时，可以将传入的集合参数中包含的 [UITouch](#) 对象数量和 [UIEvent](#) 参数对象中与该视图关联的触摸对象数量相比较。请看下面的例子：

```
- (void)touchesEnded:(NSSet*)touches withEvent:(UIEvent*)event {  
    if ([touches count] == [[event touchesForView:self] count]) {  
        // last finger has lifted....  
    }  
}
```

运动事件

当用户以特定方式移动设备，比如摇摆设备时，iPhone 或者 iPod touch 会产生运动事件。运动事件源自设备加速计。系统会对加速计的数据进行计算，如果符合某种模式，就将它解释为手势，然后创建一个代表该手势的 [UIEvent](#) 对象，并发送给当前活动的应用程序进行处理。

请注意：在 iPhone 3.0 上，只有摇摆设备的动作会被解释为手势，并形成运动事件。

运动事件比触摸事件简单得多。系统只是告诉应用程序动作何时开始及何时结束，而不包括在这个过程中发生的每个动作的时间。而且，触摸事件中包含一个触摸对象的[集合](#)及其相关的状态，而运动事件中除了事件类型、子类型、和时间戳之外，没有其它[状态](#)。系统以这种方式来解析运动手势，避免和方向变化事件造成冲突。

为了处理运动事件，[UIResponder](#) 的子类必须实现 [motionBegan:withEvent:](#) 或 [motionEnded:withEvent:](#) 方法之一，或者同时实现这两个方法。举例来说，如果用户希望赋予水平摆动和垂直摆动不同的意义，就可以在 [motionBegan:withEvent:](#) 方法中将当前加速计轴的值缓存起来，并将它们和 [motionEnded:withEvent:](#) 消息传入的值相比较，然后根据不同的结果进行动作。响应者还应该实现 [motionCancelled:withEvent:](#) 方法，以便响应系统发出的运动取消的事件。有些时候，这些事件会告诉您整个动作根本不是一个正当的手势。

应用程序及其键盘焦点窗口会将运动事件传递给窗口的第一响应者。如果第一响应者不能处理，事件就沿着响应者链进行传递，直到最终被处理或忽略，这和触摸事件的处理相类似（详细信息请参见“[事件的传递](#)”部分）。但是，摆动事件和触摸事件有一个很大的不同，当用户开始摆动设备时，系统就会通过 [motionBegan:withEvent:](#) 消息的方式向第一响应者发送一个运动事件，如果第一响应者不能处理，该事件就在响应者链中传递；如果摆动持续的时间小于1秒左右，系统就会向第一响应者发送 [motionEnded:withEvent:](#) 消息；但是，如果摆动时间持续更长，如果系统确定当前的动作不是摆动，则第一响应者会收到一个 [motionCancelled:withEvent:](#) 消息。

如果摆动事件沿着响应者链传递到窗口而没有被处理，且 [UIApplication](#) 的

applicationSupportsShakeToEdit 属性被设置为 YES，则 iPhone OS 会显示一个带有撤消（Undo）和重做（Redo）的命令。缺省情况下，这个属性的值为 NO。

拷贝、剪切、和粘贴操作

在 iPhone OS 3.0 之后，用户可以在一个应用程序上拷贝文本、图像、或其它数据，然后粘贴到当前或其它应用程序的不同位置上。比如，您可以从某个电子邮件中拷贝一个地址，然后粘贴到 Contacts 程序的地址域中。目前，UIKit 框架在 [UITextView](#)、[UITextField](#)、和 [UIWebView](#) 类中实现了拷贝-剪切-粘贴支持。如果您希望在自己的应用程序中得到这个行为，可以使用这些类的对象，或者自行实现。

本文的下面部分将描述 UIKit 中用于拷贝、剪切、和粘贴操作的编程接口，并解释其用法。

请注意：与拷贝和粘贴操作相关的使用指南，请参见 [iPhone 人机界面指南](#) 文档中的“支持拷贝和粘贴”部分。

UIKit 中支持拷贝-粘贴操作的设施

UIKit 框架提供几个类和一个 [非正式协议](#)，用于为应用程序中的拷贝、剪切、和粘贴操作提供方法和机制。具体如下：

[UIPasteboard](#) 类提供了粘贴板的接口。粘贴板是用于在一个应用程序内或不同应用程序间进行数据共享的受保护区域。该类提供了读写剪贴板上数据项目的方法。

[UIMenuController](#) 类可以在选定的拷贝、剪切、和粘贴对象的上下方显示一个编辑菜单。编辑菜单上的命令可以有拷贝、剪切、粘贴、选定、和全部选定。

[UIResponder](#) 类声明了 [canPerformAction:withSender:](#) 方法。响应者类可以实现这个方法，以根据当前的上下文显示或移除编辑菜单上的命令。

[UIResponderStandardEditActions](#) 非正式协议声明了处理拷贝、剪切、粘贴、选定、和全部选定命令的接口。当用户触碰编辑菜单上的某个命令时，相应的 [UIResponderStandardEditActions](#) 方法就会被调用。

粘贴板的概念

粘贴板是同一应用程序内或不同应用程序间交换数据的标准化机制。粘贴板最常见的用途是处理拷贝、剪贴、和粘贴操作：

当用户在一个应用程序中选定数据并选择拷贝（或剪切）菜单命令时，被选择的数据就会被放置在粘贴板上。

当用户选择粘贴命令时（可以在同一或不同应用程序中），粘贴板上的数据就会被拷贝到当前应用程序上。

在 iPhone OS 中，粘贴板也用于支持查找（Find）操作。此外，还可以用于在不同应用程序间通过定制的 URL 类型传输数据（而不是通过拷贝、剪切、和粘贴命令，关于这个技巧的信息请参见“[和其它应用程序间的通讯](#)”部分。

无论是哪种操作，您通过粘贴板执行的基本任务是读写粘贴板数据。虽然这些任务在概念上很简单，但是它们屏蔽了很多重要的细节。复杂的原因主要在于数据的表现方式可能有很多种，而这个复杂性又引入了效率的考虑。本文的下面部分将对这些以及其它的问题进行讨论。

命名粘贴板

粘贴板可能是公共的，也可能是私有的。公共粘贴板被称为**系统粘贴板**；私有粘贴板则由应用程序自行创建，因此被称为**应用程序粘贴板**。粘贴板必须有唯一的名字。UIPasteboard 定义了两个系统粘贴板，每个都有自己的名字和用途：

[UIPasteboardNameGeneral](#) 用于剪切、拷贝、和粘贴操作，涉及到广泛的数据类型。您可以通过该类的 [generalPasteboard](#) 类方法来取得代表通用（General）粘贴板的**单件**对象。

[UIPasteboardNameFind](#) 用于检索操作。当前用户在检索条（[UISearchBar](#)）键入的字符串会被写入到这个粘贴板中，因此可以在不同的应用程序中共享。您可以通过调用 [pasteboardWithName:create:类方法](#)，并在名字参数中传入 [UIPasteboardNameFind](#) 值来取得代表检索粘贴板的对象。

典型情况下，您只需使用系统定义的粘贴板就够了。但在必要时，您也可以通过 [pasteboardWithName:create:](#) 方法来创建自己的应用程序粘贴板。如果您调用 [pasteboardWithUniqueName](#) 方法，UIPasteboard 会为您提供一个具有唯一名称的应用程序粘贴板。您可以通过其 [name 属性声明](#)来取得这个名称。

粘贴板的持久保留

您可以将粘贴板标识为持久保留，使其内容在当前使用的应用程序终止后继续存在。不持久保留的粘贴板在其创建应用程序退出后就会被移除。系统粘贴板是持久保留的，而应用程序粘贴板在缺省情况下是不持久保留的。将其应用程序粘贴板的 [persistent](#) 属性设置为 YES 可以使其持久保留。当持久粘贴板的拥有者程序被用户卸载时，其自身也会被移除。

粘贴板的拥有者和数据项

最后将数据放到粘贴板的对象被称为该粘贴板的**拥有者**。放到粘贴板上的每一片数据都称为一个粘贴板**数据项**。粘贴板可以保有一个或多个数据项。应用程序可以放入或取得期望数量的数据项。举例来说，假定用户在视图中选择的内容包含一些文本和一个图像，粘贴板允许您 将文本和图像作为不同的数据项进行拷贝。从粘贴板读取多个数据项的应用程序可以选择只读取被支持的数据项（比如只是文本，而不支持图像）。

重要提示：当一个应用程序将数据写入粘贴板时，即使只是单一的数据项，该数据也会取代粘贴板的当前内容。虽然您可能使用 UIPasteboard 的 [addItem:](#)方法来添加项目，但是该写入方法并不会将那些项目加入到粘贴板当前内容之后。

数据的表示和 UTI

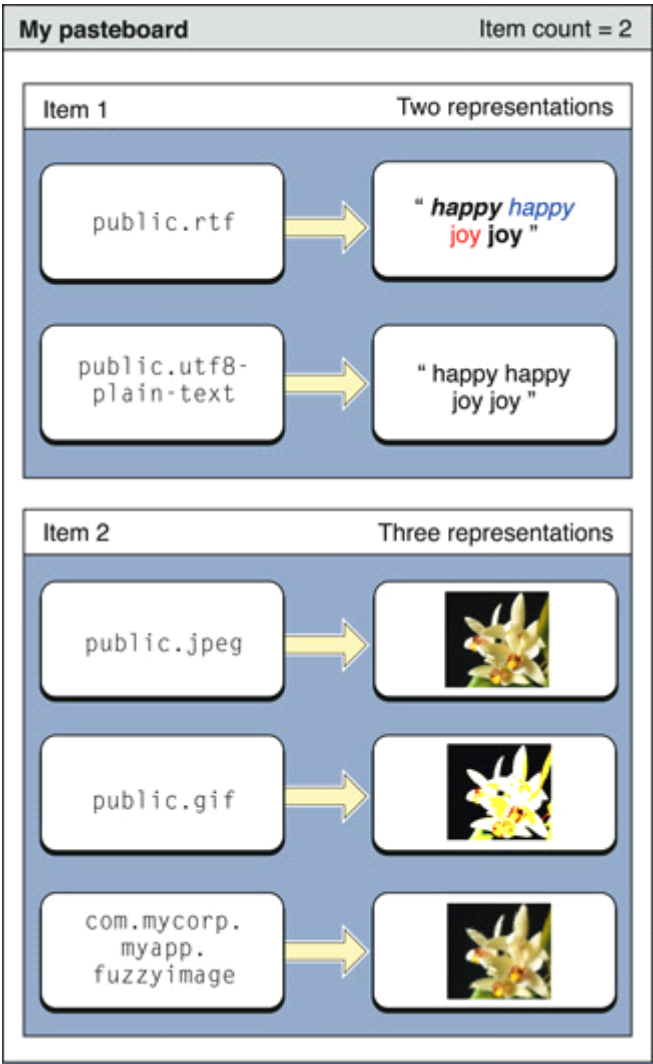
粘贴板操作经常在不同的应用程序间执行。系统并不要求应用程序了解对方的信息，包括对方可以处理的数据种类。为了最大化潜在的数据分享能力，粘贴板可以保留同一个数据项的多种**表示**。例如，一个富文本编辑器可以提供被拷贝数据的 HTML、PDF、和纯文本表示。粘贴板上的一个数据项包括应用程序可为该数据提供的所有表示。

粘贴板数据项的每种表示通常都有一个唯一类型标识符（Unique Type Identifier，缩写为 UTI）。UTI 简单定义为一个唯一标识特定数据类型的字符串。UTI 提供了一个标识数据类型

的常用手段。如果您希望支持一个定制的数据类型，就必须为其创建一个唯一的标识符。为此，您可以用反向 DNS 表示法来定义类型标识字符串，以确保其唯一性。例如，您可以用 `com.myCompany.myApp.myType` 来表示一个定制的类型标识。更多有关 UTI 的信息请参见[统一类型标识符概述](#)。

作为例子，假定一个应用程序支持富文本和图像的选择，它可能希望将富文本和 Unicode 版本的选定文本，以及选定图像的不同表示放到粘贴板上。在这样的场景下，每个数据项的每种表示都和它自己的数据一起保存，如图3-3所示。

图3-3 粘贴板及其表示



一般情况下，为了最大化潜在的共享可能性，粘贴板数据项应该包括尽可能多的表示。粘贴板的读取程序必须找到最适合自身能力（如果有的话）的数据类型。通常情况下，这意味着选择内涵最丰富的可用类型。举例来说，一个文本编辑器可能为被拷贝的数据提供 HTML（富文本）和纯文本表示，支持富文本的应用程序应该选择 HTML 表示，而只支持纯文本的应用程序则应该选择纯文本的表示。

变化记数

变化记数是每个粘贴板都有的变量，它随着每次粘贴板内容的变化而递增——特别是发生增加、修改、或移除数据项的时候。应用程序可以通过考察变化记数（通过 [changeCount](#) 属性）

来确定粘贴板的当前数据是否和最后一次取得的数据相同。每次变化记数递增时，粘贴板都会向对此感兴趣的观察者发送通告。

选择和菜单管理

在拷贝或剪切视图中的某些内容之前，必须首先选择“某些内容”。它可能是一些文本、一个图像、一个 URL、一种颜色、或者其它类型的数据，包括[定制对象](#)。为了在定制视图中实现拷贝-和-粘贴行为，您必须自行管理该视图中对象的选择。如果用户通过特定的触摸手势（比如双击）来选择视图中的对象，您就必须处理 该事件，即在程序内部记录该选择（同时取消之前的选择），可能还要在视图中指示新的选择。如果用户可以在视图选择多个对象，然后进行拷贝-剪切-粘贴操作，您就必须实现多选的行为。

请注意：触摸事件及其处理技巧在[“触摸事件”](#)部分进行讨论。

当应用程序确定用户请求了编辑菜单时——可能就是一个选择的动作——您应该执行下面的步骤来显示菜单：

调用 [UIMenuController](#) 的 [sharedMenuController](#) 类方法来取得[全局的](#)，即菜单控制器实例。计算选定内容的边界，并用得到的边界矩形调用 [setTargetRect:inView:](#)方法。系统会根据选定内容与屏幕顶部和底部的距离，将编辑菜单显示在该矩形的上方或下方。

调用 [setMenuVisible:animated:](#)方法（两个参数都传入 YES），在选定内容的上方或下方动画显示编辑菜单。

程序清单3-4演示了如何在 [touchesEnded:withEvent:](#)方法的实现中显示编辑菜单（注意，例子中省略了处理选择的代码）。在这个代码片段中，定制视图还向自己发送一个[becomeFirstResponder](#)消息，确保自己在随后的拷贝、剪切、和粘贴操作中是第一响应者。

程序清单3-4 显示编辑菜单

```
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
    UITouch *theTouch = [touches anyObject];

    if ([theTouch tapCount] == 2 && [self becomeFirstResponder]) {

        // selection management code goes here...

        // bring up editing menu.
        UIMenuController *theMenu = [UIMenuController sharedMenuController];
        CGRect selectionRect = CGRectMake(currentSelection.x, currentSelection.y, SIDE,
SIDE);
        [theMenu setTargetRect:selectionRect inView:self];
        [theMenu setMenuVisible:YES animated:YES];

    }
}
```

初始的菜单包含所有的命令，因此第一响应者提供了相应的 [UIResponderStandardEditActions](#) 方法的实现（copy:、paste:等）。但是在菜单被显示之前，系统会向第一响应者发送一个[canPerformAction:withSender:](#)消息。在很多情况下，第一响应者就是定制视图的本身。在该方法的实现中，响应者考察给定的命令（由第一个参数传入的[选择器](#)表示）是否适合当前的

上下文。举例来说，如果该选择器是 `paste:`，而粘贴板上没有该视图可以处理的数据，则响应者应该返回 `NO`，以便禁止粘贴命令。如果第一响应者没有实现 `canPerformAction:withSender:` 方法，或者没有处理给定的命令，该消息就会进入响应者链。程序清单3-5展示了 `canPerformAction:withSender:` 方法的一个实现。该实现首先寻找和 `copy:`、`copy:`、及 `paste:` 选择器相匹配的消息，并根据当前选择的上下文激活或禁用拷贝、剪切、和粘贴菜单命令。对于粘贴命令，还考虑了粘贴板的内容。

程序清单3-5 有条件地激活菜单命令

```
- (BOOL)canPerformAction:(SEL)action withSender:(id)sender {
    BOOL retVal = NO;
    ColorTile *theTile = [self colorTileForOrigin:currentSelection];

    if (action == @selector(paste:))
        retVal = (theTile == nil) &&
            [[UIPasteboard
                generalPasteboard]
 containsPasteboardTypes:
            [NSArray arrayWithObject:ColorTileUTI]];
    else if (action == @selector(cut:) || action == @selector(copy:))
        retVal = (theTile != nil);
    else
        retVal = [super canPerformAction:action
 withSender:sender];
    return retVal;
}
```

请注意，这个方法的最后一个 `else` 子句调用了超类的实现，使超类有机会处理子类忽略的命令。

还要注意，操作一个菜单命令可能会改变其它菜单命令的上下文。比如，当用户选择视图中的所有对象时，拷贝和剪切命令就应该被包含在菜单中。在这种情况下，虽然菜单仍然可见，但是响应者可以调用菜单控制器的 [update](#) 方法，使第一响应者的 `canPerformAction:withSender:` 再次被调用。

拷贝和剪切选定的内容

当用户触碰编辑菜单上的拷贝或剪切命令时，系统会分别调用响应者对象的 `copy:` 或 `cut:` 方法。通常情况下，第一响应者——也就是您的定制视图——会实现这些方法，但如果没有实现的话，该消息会按正常的方式进入响应者链。请注意，[UIResponderStandardEditActions 非正式协议](#) 声明了这些方法。

请注意：由于 `UIResponderStandardEditActions` 是非正式协议，应用程序中的任何类都可以实现它的方法。但是，为了使命令可以按缺省的方式在响应者链上传递，实现这些方法的类应该继承自 [UIResponder](#) 类，且应该被安装到响应者链中。

在 `copy:` 或 `cut:` 消息的响应代码中，您需要把和选定内容相对应的对象或数据以尽可能多的表示形式写入到粘贴板上。这个操作涉及到如下这些步骤（假定只有一个的粘贴板数据项）：标识或取得和选定内容相对应的对象或二进制数据。

二进制数据必须封装在 [NSData 对象](#) 中。其它可以写入到粘贴板的对象必须是 [属性列表](#) 对象

—也就是说，必须是下面这些类的对象：[NSString](#)、[NSArray](#)、[NSDictionary](#)、[NSDate](#)、[NSNumber](#)、或者 [NSURL](#)（有关属性列表对象的更多信息，请参见[属性列表编程指南](#)）。可能的话，请为对象或数据生成一或多个其它的表示。

举例来说，在之前提到的为选定图像创建 [UIImage](#) 对象的步骤中，您可以通过 [UIImageJPEGRepresentation](#) 或 [UIImagePNGRepresentation](#) 函数将图像转换为不同的表示。取得粘贴板对象。

在很多情况下，使用通用粘贴板就可以了。您可以通过 [generalPasteboard](#) 类方法来取得该对象。

为写入到粘贴板数据项的每个数据表示分配一个合适的 UTI。

这个主题的讨论请参见“[粘贴板的概念](#)”部分。

将每种表示类型的数据写入到第一个粘贴板数据项中：

向粘贴板对象发送 [setData:forPasteboardType:](#)消息可以写入数据对象。

向粘贴板对象发送 [setValue:forPasteboardType:](#)消息可以写入属性列表对象。

- 2 对于剪切（cut:方法）命令，需要从应用程序的[数据模型](#)中移除选定内容所代表的对象，并更新视图。

程序清单3-6展示了 copy:和 cut:方法的一个实现。cut:方法调用了 copy:方法，然后从视图和数据模型中移除选定的对象。注意，copy:方法对[定制对象](#)进行归档，目的是得到一个 NSData 对象，以便作为参数传递给粘贴板的 setData:forPasteboardType:方法。

程序清单3-6 拷贝和剪切操作

```
- (void)copy:(id)sender {
    UIPasteboard *gpBoard = [UIPasteboard generalPasteboard];
    ColorTile *theTile = [self colorTileForOrigin:currentSelection];
    if (theTile) {
        NSData *tileData = [NSKeyedArchiver
archivedDataWithRootObject:theTile];
        if (tileData)
            [gpBoard setData:tileData forPasteboardType:ColorTileUTI];
    }
}

- (void)cut:(id)sender {
    [self copy:sender];
    ColorTile *theTile = [self colorTileForOrigin:currentSelection];

    if (theTile) {
        CGPoint tilePoint = theTile.tileOrigin;
        [tiles removeObject:theTile];
        CGRect tileRect = [self rectFromOrigin:tilePoint inset:TILE_INSET];
        [self setNeedsDisplayInRect:tileRect];
    }
}
```

粘贴选定内容

当用户触碰编辑菜单上的粘贴命令时，系统会调用响应者对象的 `paste:` 方法。通常情况下，第一响应者——也就是您的定制视图——会实现这些方法，但如果没有实现的话，该消息会按正常的方式进入响应者链。`paste:` 方法在 [UIResponderStandardEditActions 非正式协议](#) 中声明。在 `paste:` 消息的响应代码中，您可以从粘贴板中读取应用程序支持的表示，然后将被粘贴对象加入到应用程序的 [数据模型](#) 中，并将新对象显示在用户指定的视图位置上。这个操作涉及到如下这些步骤（假定只有单一的粘贴板数据项）：

取得粘贴板对象。

在很多情况下，使用通用粘贴板就可以了，您可以通过 [generalPasteboard 类方法](#) 来取得该对象。

确认第一个粘贴板数据项是否包含应用程序可以处理的表示，这可以通过调用 [containsPasteboardTypes:](#) 方法，或者调用 [pasteboardTypes](#) 方法并考察其返回的类型数组来实现。

请注意，您在 [canPerformAction:withSender:](#) 方法的实现中应该已经执行过这个步骤。

如果粘贴板的第一个数据项包含应用程序可以处理的数据，则可以调用下面的方法来读取：[dataForPasteboardType:](#)，如果要读取的数据被封装为 [NSData 对象](#)，就可以使用这个方法。[valueForPasteboardType:](#)，如果要读取的数据被封装为 [属性列表](#) 对象，请使用这个方法（请参见“[拷贝和剪切选定的内容](#)”部分）。

将对象加入到应用程序的数据模型中。

3 将对象的表示显示在用户界面中用户指定的位置上。

程序清单3-7是 `paste:` 方法的一个实现实例，该方法执行与 `cut:` 及 `copy:` 方法相反的操作。示例中的视图首先确认粘贴板是否包含自身支持的定制表示数据，如果是的话，就读取该数据并将它加入到应用程序的数据模型中，然后将视图的一部分——当前选定区域——标识为需要重画。

程序清单3-7 将粘贴板的数据粘贴到选定位置上

```
- (void)paste:(id)sender {
    UIPasteboard *gpBoard = [UIPasteboard generalPasteboard];
    NSArray *pbType = [NSArray arrayWithObject:ColorTileUTI];
    ColorTile *theTile = [self colorTileForOrigin:currentSelection];
    if (theTile == nil && [gpBoard containsPasteboardTypes:pbType]) {

        NSData *tileData = [gpBoard dataForPasteboardType:ColorTileUTI];
        ColorTile *theTile = (ColorTile *)[NSKeyedUnarchiver
unarchiveObjectWithData:tileData];
        if (theTile) {
            theTile.tileOrigin = self.currentSelection;
            [tiles addObject:theTile];
            CGRect tileRect = [self rectFromOrigin:currentSelection inset:TILE_INSET];
            [self setNeedsDisplayInRect:tileRect];
        }
    }
}
```

消除编辑菜单

在您实现的 `cut:`、`copy:`、或 `paste:` 命令返回后，编辑菜单会被自动隐藏。通过下面的代码使它保持可见：

```
[UIMenuController setMenuController].menuVisible = YES;
```

系统可能在任何时候隐藏编辑菜单，比如当显示警告信息或用户触碰屏幕其它区域时，编辑菜单就会被隐藏。如果您有某些状态或屏幕显示需要依赖于编辑菜单是否显示的话，就应该侦听 `UIMenuControllerWillHideMenuNotification` 通告，并执行恰当的动作。

图形和描画

高质量的图形是应用程序用户界面的重要组成部分。提供高质量的图形不仅会使应用程序具有好的的外观，还会使它看起来像是系统的自然扩展。iPhone OS 为创建高质量的图形提供两种路径：即通过 OpenGL 进行渲染，或者通过 Quartz、Core Animation、和 UIKit 进行渲染。

OpenGL [框架](#) 主要适用于游戏或要求高帧率的应用程序开发。它是一组基于 C 语言的接口，用于在桌面电脑上创建 2D 和 3D 内容。iPhone OS 通过 OpenGL ES 框架来支持 OpenGL 描画，该框架同时支持 OpenGL ES 2.0 和 OpenGL ES v1.1。OpenGL ES 是特别为嵌入式硬件系统设计的，和桌面版本的 OpenGL 有很多不同。

对于希望采用更为面向对象的方法进行描画的开发 者，iPhone OS 提供了 Quartz、Core Animation、还有 UIKit 中的图形支持。Quartz 是主要的描画接口，支持基于路径的描画、抗锯齿渲染、渐变填充模式、图像、颜色、坐标空间 变换、以及 PDF 文档的创建、显示、和分析。UIKit 为 Quartz 的图像和颜色操作提供了 Objective-C 的封装。Core Animation 为很多 UIKit 的视图 [属性声明](#) 的动画效果提供底层支持，也可以用于实现定制的动画。

本章将为 iPhone 应用程序的描画过程提供一个概览，同时介绍描画技术的一些具体描画技巧。本章还为如何优化 iPhone OS 平台的描画代码提供一些指导原则和小贴士。

UIKit 的图形系统

在 iPhone OS 上，所有的描画—无论是否采用 OpenGL、Quartz、UIKit、或者 Core Animation—都发生在 [UIView](#) 对象的区域内。[视图](#) 定义描画发生的屏幕区域。如果您使用系统提供的视图，描画工作会自动得到处理；然而，如果您定义自己的定制视图，则必须自行提供描画代码。对于使用 OpenGL 进行描画的应用程序，一旦建立了渲染表面，就必须使用 OpenGL 指定的描画模型。

对于 Quartz、Core Animation、和 UIKit，您需要使用本文下面部分描述的概念。

视图描画周期

[UIView](#) 对象的基本描画模型涉及到如何按需更新视图的内容。通过收集您发出的更新请求，并在最适合的时机将它们发送给您的描画代码，UIView 类使内容更新过程变得更为简单和高效。

任何时候，当视图的一部分需要重画时，[UIView](#) 对象内置的描画代码就会调用其 [drawRect:](#) 方法，并向它传入一个包含需要重画的视图区域的矩形。您需要在定制视图子类中重载这个方法，并在这个方法中描画视图的内容。在首次描画视图时，UIView 传递给 `drawRect:` 方法的矩形包含视图的全部可见区域。但在随后的调用中，该矩形只代表实际需要被描画的部分。

触发视图更新的动作有如下几种：

对遮挡您的视图的其它视图进行移动或删除操作。

将视图的 [hidden 属性声明](#) 设置为 NO，使其从隐藏状态变为可见。

将视图滚出屏幕，然后再重新回到屏幕上。

显式调用视图的 [setNeedsDisplay](#) 或者 [setNeedsDisplayInRect:](#) 方法。

在调用 [drawRect:](#) 方法之后，视图会将自己标志为已更新，然后等待新的更新动作触发下一个更新周期。如果您的视图显示的是静态内容，则只需要在视图的可见性发生变化时进行响应就可以了，这种变化可能由滚动或其它视图是否被显示引起的。然而，如果您需要周期性地更新视图内容，就必须确定什么时候调用 [setNeedsDisplay](#) 或 [setNeedsDisplayInRect:](#) 方法来触发更新。举例来说，如果您需要每秒数次地更新内容，则可能要使用一个定时器。在响应用户交互或生成新的视图内容时，也可能需要更新视图。

坐标和坐标变换

如“[视图坐标系](#)”部分描述的那样，窗口或视图的坐标原点位于左上角，坐标的值向下向右递增。当您编写描画代码时，需要通过这个坐标系来指定描画内容中点的位置。

如果您需要改变缺省的坐标系，可以通过修改当前的转换矩阵来实现。**当前转换矩阵（CTM）**是一个数学矩阵，用于将视图坐标系上的点映射到设备的屏幕上。在视图的 [drawRect:](#) 方法首次被调用时，就需要建立 CTM，使坐标系的原点和视图的原点互相匹配，且将坐标轴的正向分别处理为向下和向右。然而，您可以通过加入缩放、旋转、和转换因子来改变 CTM，从而改变缺省坐标系相对于潜在视图或窗口的尺寸、方向、和位置。

修改 CTM 是在视图内容描画的标准技术，因为它需要的工作比其它方法少得多。如果您希望在当前描画系统中坐标为 (20, 20) 的位置上画出一个 10 x 10 的方形，可以首先创建一个路径，将它的起始点移动到坐标为 (20, 20) 的位置上，然后再画出组成方形的几条线。然而，如果您在之后希望将方形移动到坐标为 (10, 10) 的位置上，就必须用新的起始点重新创建路径。事实上，每次改变原点，您都必须重新创建路径。创建路径是开销相对较大的操作，相比之下，创建一个起始点为 (0, 0) 的方形，然后通过修改 CTM 来匹配目标描画原点的开销就少一些。

在 Core Graphics [框架](#) 中，有两种修改 CTM 的方法。您可以通过 [CGContext 参考](#) 定义的 CTM 操控函数来直接修改 CTM，也可以创建一个 [CGAffineTransform](#) 结构，将您希望的转换应用到该结构上，然后将它连结到 CTM 上。使用仿射变换可以将各种变换组合在一起，然后一次性地应用到 CTM 上。您也可以通过修改和恢复仿射变换来调整点、尺寸、和矩形的值。有关仿射变换的更多信息，请参见 [Quartz 2D 编程指南](#) 和 [CGAffineTransform 参考](#)。

图形上下文

在调用您提供的 [drawRect:](#) 方法之前，视图对象会自动配置其描画环境，使您的代码可以立即进行描画。作为这些配置的一部分，[UIView](#) 对象会为当前描画环境创建一个图形上下文（对应于 [CGContextRef](#) 封装类型）。该图形上下文包含描画系统执行后续描画命令所需要的信息，定义了各种基本的描画属性，比如描画使用的颜色、裁剪区域、线的宽度及风格信息、字体信息、合成选项、以及几个其它信息。

当您希望在视图之外的其它地方进行描画时，可以创建定制的图形上下文对象。在 Quartz 中，当您希望捕捉一系列描画命令并将它们用于创建图像或 PDF 文件时，就需要这样做。您可以用 [CGBitmapContextCreate](#) 或 [CGPDFContextCreate](#) 函数来创建上下文。有了上下文

对象之后，您可以将它传递给创建内容时需要调用的描画函数。

您 创建的定制图形上下文的坐标系统和 iPhone OS 使用的本地坐标系统是不同的。与后者的坐标原点位于左上角不同的是，前者的坐标原点位于左下角，其坐标值向上向右递增。您在描画命令中指定的坐标必须 对此加以考虑，否则，结果图像或 PDF 文件在渲染时就可能会发生错误。

重要提示：由于在位图或 PDF 上下文中进行描画时使用的是左下原点，所以在将描画结果渲染到视图上的时候，必须对坐标系统进行补偿。换句话说，如果您创建一个图像，并调用 `CGContextDrawImage` 函数来进行描画，则该图像在缺省情况下是上下颠倒的。为了纠正这个问题，您必须将 CTM 的 y 轴进行翻转（即将该值乘以-1），使其原点从左下角移动到视图的左上角。

如果使用 `UIImage` 对象来包装您所创建的 `CGImageRef` 类型，则不需要修改 CTM。`UIImage` 对象会自动对 `CGImageRef` 类型的坐标系统进行翻转补偿。

有关图形上下文、如何修改图形状态信息、以及如何用图形上下文来创建定制内容的更多信息，请参见 [Quartz 2D 编程指南](#)。如果需要与图形上下文结合使用的函数列表，则请参见 [CGContext 参考](#)、[CGBitmapContext 参考](#)、以及 [CGPDFContext 参考](#)。

点和像素的不同

Quartz 描画系统使用基于向量的描画模型，这不同于基于栅格的描画模型。在栅格描画模型中，描画命令操作的是每个独立的像素，而 Quartz 的描画命令则是通过固定比例的描画空间来指定，这个描画空间就是所谓的**用户坐标空间**。然后，由 iPhone OS 将该描画空间的坐标映射为设备的实际像素。这个模型的优势在于，使用向量命令描画的图形在通过仿射变换放大或缩小之后仍然显示良好。

为了维持基于向量的描画系统固有的精度，Quartz 描画系统使用浮点数（而不是定点数）作为坐标值。使用浮点类型的坐标值可以非常精确地指定描画内容的位置。在大多数情况下，您不必担心这些值最终如何映射到设备的屏幕。

用户坐标空间是您发出的所有描画命令的工作环境。该空间的单位由点来表示。**设备坐标空间**指的是设备内在的坐标空间，由像素来表示。缺省情况下，用户坐标空间上的一个点等于设备坐标空间的一个像素，这意味着一个点等于1/160英寸。然而，您不应该假定这个比例总是1:1。

颜色和颜色空间

iPhone OS 支持 Quartz 中具有的所有颜色空间，但是，大多数应用程序应该只需要 RGB 颜色空间，因为 iPhone OS 是为嵌入式硬件设计的，而且只在一个屏幕上显示，在这种场合下，RGB 颜色空间是最合适的。

[UIColor](#) 对象提供了一些便利方法，用于通过 RGB、HSB、和灰度值指定颜色值。以这种方式创建颜色不需要指定颜色空间，`UIColor` 对象会自动为您指定。

您也可以使用 Core Graphics 框架中的 [CGContextSetRGBStrokeColor](#) 和 [CGContextSetRGBFillColor](#) 函数来创建和设置颜色。虽然 Core Graphics 框架支持用其它的颜色空间来创建颜色，还支持创建定制的颜色空间，但是我们不推荐在描画代码中使用那些颜色。您的描画代码应该总是使用 RGB 颜色。

支持的图像格式

表4-1列出了 iPhone OS 直接支持的图像格式。在这些格式中，我们优先推荐 PNG 格式。

表4-1 支持的图像格式	格式	文件扩展名
可移植网络图像格式(PNG)		.png
标记图像文件格式(TIFF)		.tiff, .tif
联合影像专家组格式(JPEG)		.jpeg, .jpg
图形交换格式(GIF)		.gif
视窗位图格式(DIB)		.bmp, .BMPf
视窗图标格式		.ico
视窗光标		.cur
XWindow 位图		.xbm

描画贴士

本文的下面部分将为您提供一些贴士，讨论如何在编写高质量描画代码的同时确保应用程序外观对最终用户具有吸引力。

确定何时使用定制的描画代码

根据您创建的应用程序类型，不使用或使用很少的定制代码进行描画是可能的。虽然沉浸式的应用程序通常广泛使用定制的描画代码，但是工具型和效率型的应用程序则可以使用标准的视图和控件来显示内容。

定制描画代码的使用应该限制在当显示在屏幕上的内容需要动态改变的场合。比如，用于跟踪用户描画命令的应用程序需要使用定制描画代码；还比如，游戏程序也需要经常更新屏幕，以反映游戏环境的改变。在那些情况下，您需要选择合适的描画技术，以及创建定制的视图类来正确处理事件和更新屏幕。

另一方面，如果应用程序中大量的用户界面是固定的，则可以事先将那些界面渲染到一或多个图像文件中，然后在运行时通过 [UIImageView](#) 对象显示出来。您可以根据自己的需要，将图像视图和其它内容组合在一起。比如，您可以用 [UILabel](#) 对象来显示需要配置的文本，用按键或其它控件来进行交互。

提高描画的性能

在任何平台上，描画的开销都比较昂贵，对描画代码进行优化一直都是开发过程的重要步骤。表4-2列举了几个贴士，用于确保您的描画代码得到尽可能的优化。除了这些贴士，您还应该用现有的性能工具对代码进行测试，消除描画热点和多余的描画操作。

表4-2 提高描画性能的贴士	Action
Tip	
使描画工作最小化	在每个更新周期中，您应该只更新视图中真正发生变化的部分。如果您使用 UIView 的 drawRect: 方法来进行描画，则要通过传给该方法的更新矩形来限制描画的范围。对于基于 OpenGL 的描画，您必须自行跟踪更新区域。

尽可能将视图标识为不透明	合成不透明的视图所需要的开销比合成部分透明的视图要少得多。一个不透明的视图必须不包含任何透明的内容，且视图的 opaque 属性必须设置为 YES。
删除不透明的 PNG 文件中的 alpha 通道	如果一个 PNG 图像的每个像素都是不透明的，则将其 alpha 通道删除可以避免对包含该图像的图层进行融合操作，从而很大程度上简化了该图像的合成，提高描画的性能。
在滚动过程中重用表格单元和视图	应该避免在滚动过程中创建新的视图。创建新视图的开销会减少用于更新屏幕的时间，因而导致滚动不平滑。
避免在滚动过程中清除原先的内容	缺省情况下，在调用 drawRect: 方法对视图的某个区域进行更新之前，UIKit 会清除该区域对应的上下文缓冲区。如果您对视图的滚动事件进行响应，则在滚动过程中反复清除缓冲区的开销是很大的。为了禁止这种行为，可以将 clearsContextBeforeDrawing 属性设置为 NO。
在描画过程中尽可能不改变图形状态	改变图形状态需要窗口服务器的参与。如果您要描画的内容使用类似的图形状态，则尽可能将这些内容一起描画，以减少需要改变的状态。

保持图像的质量

为用户界面提供高品质的图像应该是设计工作中的重点之一。图像是一种合理而有效的显示复杂图形的方法，任何合适的地方都可以使用。在为应用程序创建图像的时候，请记住下面的原则：

使用 PNG 格式的图像。 PNG 格式可以提供高品质的图像内容，是 iPhone OS 系统上推荐的图像格式。另外，iPhone OS 对 PNG 图像的描画路径是经过优化的，通常比其它格式具有更高的效率。

创建大小合适的图像，避免在显示时调整尺寸。 如果您计划使用特定尺寸的图像，则在创建图像资源时，务必使用相同的尺寸。不要创建一个大的图像，然后再缩小，因为缩放需要额外的 CPU 开销，而且需要进行插值。如果您需要以不同的尺寸显示图像，则请包含多个版本的图像，并选择与目标尺寸相对接近的图像来进行缩放。

用 Quartz 和 UIKit 进行描画

Quartz 是 iPhone OS 的窗口服务器和描画技术的一般叫法。Core Graphics [框架](#) 是 Quartz 的核心，也是内容描画的基本接口。该框架提供的数据类型和函数用于操作如下对象：

图形上下文

路径

图像和位图

透明层

颜色、图案颜色、和颜色空间

渐变和阴影

字体

PDF 内容

UIKit 在 Quartz 基本特性的基础上提供了一组专门的类，用于与图形相关的操作。UIKit 的

图形类并不是为了向您提供一个全面的描画工具箱 — 因为这样的工具 在 Core Graphics 框架中已经有了，而是为了向其它 UIKit 类提供描画支持。UIKit 包括下面的类和函数：

[UIImage](#), 一个不可变类，用于图像显示。

[UIColor](#), 为设备颜色提供基本的支持。

[UIFont](#), 为需要字体的类提供字体信息。

[UIScreen](#), 提供屏幕的基本信息。

生成 UIImage 对象的 JPEG 或 PNG 表示的函数。

描画矩形和对描画区域进行裁剪的函数。

改变和获取当前图形上下文的函数

有关 UIKit 包含的类和方法的信息，请参见 [UIKit 框架参考](#)，有关组成 Core Graphics 框架的封装类型和函数，请参见 [Core Graphics 框架参考](#)。

配置图形上下文

在您的 [drawRect:](#)方法被调用时，视图对象的内置描画代码已经为您创建并配置好了一个缺省图形上下文。您可以通过调用 [UIGraphicsGetCurrentContext](#) 函数来取得当前上下文的指针，该函数返回一个类型为 [CGContextRef](#) 的引用，您可以将它传给 Core Graphics 函数，以修改当前的图形状态。表4-3列出了负责设置各种图形状态的一些主要函数，如果需要完整的函数列表，请参见 [CGContext 参考](#)。该表还列出了 UIKit 中和这些函数对应的组件，如果有的话。

表 4-3 修改图形状态的 Core Graphics 函数	Core Graphics 函数	UIKit 对应组件
当前转换矩阵(CTM)	CGContextRotateC	无
	TM	
	CGContextScaleC	
	T	
	CGContextTranslat	
	eCTM	
裁剪区域	CGContextConcatC	无
	TM	
	CGContextClipToR	
	ect	
	CGContextSetLine	
	Width	
线: 宽度，线间链接，线端点，破折号，斜角限制	CGContextSetLineJ	无
	oin	
	CGContextSetLine	
	Cap	
	CGContextSetLine	
	Dash	
曲线拟合的精度(平滑度)	CGContextSetMiter	无
	Limit	
	CGContextSetFlatn	
	ess	

抗锯齿设置	CGContextSetAntialiasing	无
颜色：填充和笔划设置	CGContextSetRGBFillColor CGContextSetRGBStrokeColor	UIColor 类
Alpha 值(透明度)	CGContextSetAlpha	无
渲染意图	CGContextSetRenderingIntent	无
颜色空间：填充和笔划设置	CGContextSetFillColorSpace CGContextSetStrokeColorSpace	无
文本：字体，字体尺寸，字符间隔， 文本描画模式	CGContextSetFont CGContextSetFontSize CGContextSetCharacterSpacing	UIFont 类
混合模式	CGContextSetBlendMode	您可以为 UIImage 类和各种描画函数指定混合模式

图形上下文中包含一个保存过的图形状态堆栈。在 Quartz 创建图形上下文时，该堆栈是空的。[CGContextSaveGState](#) 函数的作用是将当前图形状态推入堆栈。之后，您对图形状态所做的修改会影响随后的描画操作，但不影响存储在堆栈中的拷贝。在修改完成后，您可以通过 [CGContextRestoreGState](#) 函数把堆栈顶部的状态弹出，返回到之前的图形状态。这种推入和弹出的方式是回到之前图形状态的快速方法，避免逐个撤消所有的状态修改；这也是将某些状态（比如裁剪路径）恢复到原有设置的唯一方式。

有关图形上下文及如何用它来配置描画环境的一般信息，请参见 [Quartz 2D 编程指南](#) 的 [图形上下文](#) 部分。

创建和描画图像

iPhone OS 同时支持通过 UIKit 和 Core Graphics [框架](#) 装载和显示图像。到底选择哪些类和函数描画图像取决于具体的应用场合。但是，我们推荐您尽可能使用 UIKit 来表示图像。表4-4 列举了一些使用场景及处理这些场景的推荐方法。

表 4-4 图像使用场景	推荐用法
将图像作为视图的内容	使用 UIImageView 类装载和显示图像。这种方法假定视图的内容就是一个图像，但您仍然可以在图像视图上面放置其它视图，用于描画其它控件或内容。
将图像作为部分视图的装饰	用 UIImage 类装载和描画图像。
将某些位图数据保存到图像对象中	使用 UIGraphicsBeginImageContext 函数创建一个新的、基于图像的图形上下文。在这之后，您就可以将图像内容描画在上面，然

后用 [UIGraphicsGetImageFromCurrentImageContext](#) 函数生成一个图像（如果需要的话，您甚至可以继续描画并生成其它的图像）。在图像创建完成后，可以用 [UIGraphicsEndImageContext](#) 函数来关闭图形上下文。

如果您更喜欢使用 Core Graphics，则可以用 [CGBitmapContextCreate](#) 函数创建一个位图的图形上下文，并在上面描画您的图像内容。画完之后，用 [CGBitmapContextCreateImage](#) 函数把位图上下文中的内容创建为一个 CGImageRef 类型的图像。您可以直接描画 Core Graphics 图像，或者用它来初始化一个 UIImage。

将图像保存为 JPEG 或 PNG 文件

基于原始的图像数据创建一个 UIImage 对象。通过 [UIImageJPEGRepresentation](#) 或 [UIImagePNGRepresentation](#) 函数取得一个 NSData 对象，并使用该方法将数据保存为文件。

下面的例子将展示如何从应用程序的[程序包](#)中装载一个图像。在该图像装载完成后，您可以将它用于初始化 UIImageView 对象、将它保存到磁盘、或者在视图的 drawRect: 方法中进行显式描画。

```
NSString* imagePath = [[NSBundle mainBundle] pathForResource:@"myImage" ofType:@"png"];
```

```
UIImage* myImageObj = [[UIImage alloc] initWithContentsOfFile:imagePath];
```

在视图的 drawRect: 方法中，您可以使用 UIImage 类提供的任何描画方法。您可以指定希望在视图的什么位置描画图像，从而避免在描画之前进行位置的转换。假定您将之前装载的图像存储在一个名为 anImage 的成员变量中，下面的代码会将该图像画在视图的(10, 10) 坐标位置上：

```
- (void)drawRect:(CGRect)rect
{
    // Draw the image
    [anImage drawAtPoint:CGPointMake(10, 10)];
}
```

重要提示：如果您使用 [CGContextDrawImage](#) 函数来直接描画位图，则在缺省情况下，图像数据会上下倒置，因为 Quartz 图像假定坐标系统的原点在左下角，且坐标轴的正向是向上和向右。虽然您可以在描画之前对其进行转换，但是将 Quartz 图像包装为一个 UIImage 对象是更简单的方法，这样可以自动补偿坐标空间的差别。有关如何用 Core Graphics 创建和描画图像的更多信息，请参见 [Quartz 2D 编程指南](#)。

创建和描画路径

路径用于描述由一序列线和 Bézier 曲线构成的2D 几何形状。UIKit 中的 [UIRectFrame](#) 和 [UIRectFill](#) 函数（以及其它函数）的功能是在视图中描画象矩形这样的简单路径。Core Graphics 中也有一些用于创建简单路径（比如矩形和椭圆形）的便利函数。对于更为复杂的路径，必须用 Core Graphics [框架](#)提供的函数自行创建。

在创建路径时，需要首先通过 [CGContextBeginPath](#) 函数配置一个接收路径命令的图形上下文。调用该函数之后，就可以使用与路径相关的函数来设置路径的起始点，描画直线和曲线，加入矩形和椭圆形等等。路径的几何形状指定完成后，就可以直接进行描画，或者将其引用

存储在 [CGPathRef](#) 或 [CGMutablePathRef](#) 数据类型中，以备后用。

在视图上描画路径时，可以描画轮廓，也可以进行填充，或者同时进行这两种操作。路径轮廓可以用像 [CGContextStrokePath](#) 这样的函数来画，即用当前的笔划颜色画出以路径为中心位置的线。路径的填充则可以用 [CGContextFillPath](#) 函数来实现，它的功能是用当前的填充颜色或样式填充路径线段包围的区域。

有关如何描画路径的更多信息，包括如何为复杂路径元素指定点的信息，请参见 [Quartz 2D 编程指南](#) 的 [路径](#) 部分。有关路径创建函数的信息，则请参见 [CGContext](#) 参考和 [CGPath](#) 参考。

创建样式、渐变、和阴影

Core Graphics [框架](#) 还包含一些用于创建样式、渐变、和阴影类型的函数。基于这些类型，您可以创建复杂的颜色，并用它们来填充自己创建的路径。样式是从重复出现的图像或内容创建而来的，渐变和阴影则是不同颜色之间平滑过渡的方式。

有关创建样式、渐变、和阴影的详细信息，在 [Quartz 2D 编程指南](#) 中进行讨论。

用 OpenGL ES 进行描画

开放图形库（**Open Graphics Library**，即 **OpenGL**）是一个跨平台的、基于 C 语言的接口，用于在桌面系统中创建 2D 和 3D 内容。游戏或需要以高帧率进行描画的开发者通常需要使用这个接口。您可以用 OpenGL 函数来指定图元结构，比如点、线、多边形和纹理，以及增强这些结构外观的特殊效果。您调用的函数会将图形命令发送给底层的硬件，然后由硬件进行渲染。由于大多数渲染工作是由硬件来完成，所以 OpenGL 的描画速度通常很快。

OpenGL 的嵌入式系统版本是 OpenGL 的精简版本，是专门为移动设备设计的，可以充分利用现代图形硬件的优势。如果您希望为基于 iPhone OS 的设备——也就是 iPhone 或 iPod Touch——创建 OpenGL 内容，就要使用 OpenGL ES。iPhone OS 系统提供的 OpenGL ES [框架](#) (OpenGL ES.framework) 同时支持 OpenGL ES v1.1 和 OpenGL ES v2.0 规范。

有关 iPhone OS 系统上的 OpenGL ES 的更多信息，请参见 [iPhone OpenGL ES 编程指南](#)。

应用 Core Animation 的效果

Core Animation 是一个 Objective-C 语言的[框架](#)，其目的是为快速创建实时动画提供基础设施。Core Animation 本身并不是一个描画技术，因为它并不提供创建形状、图像、或其它内容的基本例程；相反，它是一种操作和显示由其它技术创建的内容的技术。

在 iPhone OS 上，大多数程序都会以某种形式受益于 Core Animation 技术。动画可以将当前正在发生的事情呈现给用户。比如，在用户使用 Settings 程序时，屏幕会根据用户是向预置的更深层次移动还是返回根结点而滑入或滑出视图。这种反馈是很重要的，可以为用户提供上下文的信息。动画还可以增强应用程序的视觉效果。

大多数情况下，您通过很少的工作就可以得到 Core Animation 的好处。举例来说，您可以对 UIView 类的几个[属性声明](#)（其中包括视图的边框、中心、颜色、和透明度等）进行配置，使得当它们的值发生变化时，可以触发动画效果。您需要通过少量的工作让 UIKit 知道您希望执行哪些动画，但动画的创建和运行都是自动的。有关如何触发内置视图动画的更多信息，请参见[“视图动画”](#)部分。

如果您要超越基本的动画效果，就必须直接和 Core Animation 的类及方法进行更多的交互。本文的下面部分将进一步提供有关 Core Animation 的信息，向您展示如何用它提供的类和方法创建 iPhone OS 上的典型动画。更多有关 Core Animation 及其用法的信息，请参见 [Core Animation 编程指南](#)。

关于层

Core Animation 的关键技术是层对象。层是一种轻量级的对象，在本质上类似于[视图](#)，但实际上是[模型对象](#)，负责封装显示内容的几何属性、显示时机、和视觉[属性变量](#)。内容本身可以通过如下三种方式来提供：

您可以将一个 CGImageRef 类型的数据赋值给层对象的 contents [属性变量](#)。

您可以为层分配一个委托，让它负责描画工作。

您可以从 [CALayer](#) 派生出子类，并对其显示方法进行重载。

当您操作层对象的属性时，您真正操作的是模型级别的数据，该数据决定了与之关联的内容应该如何被显示，而实际的渲染则由您的代码之外的模块来处理，系统对这个过程进行了大量的优化，确保渲染工作能快速完成。您需要做的只是设置层的内容和配置动画属性，然后让 Core Animation 接管剩下的工作。

更多有关层及如何使用层的信息，请参见 [Core Animation 编程指南](#)。

关于动画

对于具有动画效果的层，Core Animation 使用独立的动画对象来控制动画的时机和行为。[CAAnimation](#) 类及其子类实现了不同类型的动画行为，供您在代码中使用。您可以创建简单的动画，将某个[属性变量](#)从一个值变为另一个值；也可以创建复杂的关键帧动画，通过您自己提供的值和时间函数来跟踪动画。

Core Animation 还可以将多个动画组合为一个单独的单元，称为事务。[CATransaction](#) 对象负责将一组动画组合成一个单元来管理，您也可以用它提供的方法来设置动画的持续时间。

如果您需要如何创建定制动画的实例，请参见[动画类型和时机的编程指南](#)。

关于文本和 Web 的支持

iPhone OS 的文本系统提供了大量的功能，同时又非常简单易用。UIKit [框架](#)中包含几个高级类，负责管理文本的显示和输入。该框架还含有一个更为高级的类，用于显示 HTML 和基于 JavaScript 的内容。

本文的下面部分将描述 iPhone OS 对文本和 web 内容的基本支持。如果您需要这里列举的各个类的更多信息，请参见 [UIKit 框架参考](#)。

文本视图

UIKit 框架提供三个显示文本内容的基本类：

[UILabel](#) 显示静态文本字符串

[UITextField](#) 显示单行可编辑文本

[UITextView](#) 显示多行可编辑文本

虽然标签和文本编辑框通常用于显示相对少量的文本，但实际上这些类可以显示任意数量的文本。然而，基于 iPhone OS 的设备的屏幕比较小，为了使显示在屏幕上的文本便于阅读，这些类不支持像 Mac OS X 这样的桌面操作系统上常见的高级格式功能。另一方面，考虑到可能的需要，这三个类仍然支持指定字体信息，包括字体的尺寸和风格选项，只是指定的字体将应用到对象中显示的所有文本。

图5-1显示了这些文本类在屏幕上的显示实例。这些例子来自 [UICatalog](#) 示例程序，该程序演示了 UIKit 框架中的很多视图和控件。左图显示的是几个不同风格的文本输入框，右图则显示一个文本视图。灰色背景中显示的说明文字所在的视图是一些 UILabel 对象，它们被嵌入到负责显示各种视图的表格单元中。左图的屏幕底部还有一个 UILabel 对象，显示内容为“Left View”。

图5-1 UICatalog 应用程序的文本类



在使用可编辑的文本视图时，您必须提供一个[委托](#)对象，负责管理编辑会话。文本视图会向委托对象发送几个不同的[通告](#)，让它知道编辑何时开始，何时结束，并使它有机会重载某些编辑动作。举例来说，委托可以决定当前文本是否包含有效的值，还可以在需要的时候防止编辑会话被终止。在编辑过程最终结束的时候，您可以通过委托取得编辑结果，更新应用程序的数据模型。

由于各种文本视图的用法有轻微的不同，所以它们的委托方法也有所不同。为 UITextField 类提供支持的委托需要实现 [UITextFieldDelegate](#) 协议定义的方法。类似地，为 UITextView 类提供支持的委托需要实现 [UITextViewDelegate](#) 协议定义的方法。对于上述两种情形，系统并没有要求您一定要实现协议中的任何方法，但是如果没有实现必要的方法，文本输入框就没有什么用处了。有关这两个协议的更多信息，请参见 [UITextFieldDelegate 协议参考](#)和 [UITextViewDelegate 协议参考](#)。

Web 视图

[UIWebView](#) 类使您可以将一个微型 web 浏览器集成到应用程序的用户界面上。UIWebView 类充分使用了 iPhone OS 上的 web 技术, 同样的这些技术也用于实现 iPhone OS 上的 Safari, 实现对 HTML、CSS、和 JavaScript 内容的全面支持。UIWebView 还支持很多用户在 Safari 中已经熟悉了的手势, 比如通过双击和双指捏夹 (pinch) 的手势来放大和缩小页面, 还有通过手指拖动来滚动页面。

除了显示内容, 您还可以用 web 视图对象来显示 web 表单, 收集用户输入。和 UIKit 的其它文本类相似, 如果您在 web 页面的表单中有可编辑的文本框, 则轻触该文本框就会弹出键盘, 用户可以通过键盘输入文本。这是 web 浏览整体体验的一部分, web 视图会自行管理键盘的显示和消除。

图5-2显示了一个 UIWebView 对象的例子, 它来自 [UICatalog](#) 示例程序, 该程序演示了 UIKit 框架中的很多视图和控件。这个例子只是显示 HTML 内容, 如果您希望用户可以象使用 web 浏览器那样在网页之间进行漫游, 需要加入一些控件。比如, 图中的 web 视图只是占用 URL 文本框下面的空间, 而不包含文本框的本身。

图5-2 web 视图



web 视图通过其关联的[委托](#)对象提供有关网页何时被装载、及装载过程是否发生错误的信息。web 委托是指实现一个或多个 [UIWebViewDelegate](#) 协议方法的对象。您可以通过实现委

托方法来响应装载错误或处理一些与装载有关的其它任务。更多有关 `UIWebViewDelegate` 协议方法的信息请参见 [UIWebViewDelegate 协议参考](#)。

键盘和输入法

每 当用户触击一个可以接受文本输入的对象时，该对象就会请求系统显示一个合适的键盘。根据用户程序的需要和偏好的语言，系统可以显示几种不同的键盘。您的应用程序虽然不能控制用户的偏好语言（因此也不能控制键盘的输入法），但可以控制键盘的使用属性，比如特殊键的配置及其行为。

您可以直接通过应用程序中的文本对象来配置键盘的属性。`UITextField` 和 `UITextView` 类都遵循 `UITextInputTraits` 协议，该协议定义了一些配置键盘的属性。在程序或 Interface Builder 的查看器窗口中设置这些属性就可以使系统显示指定类型的键盘。

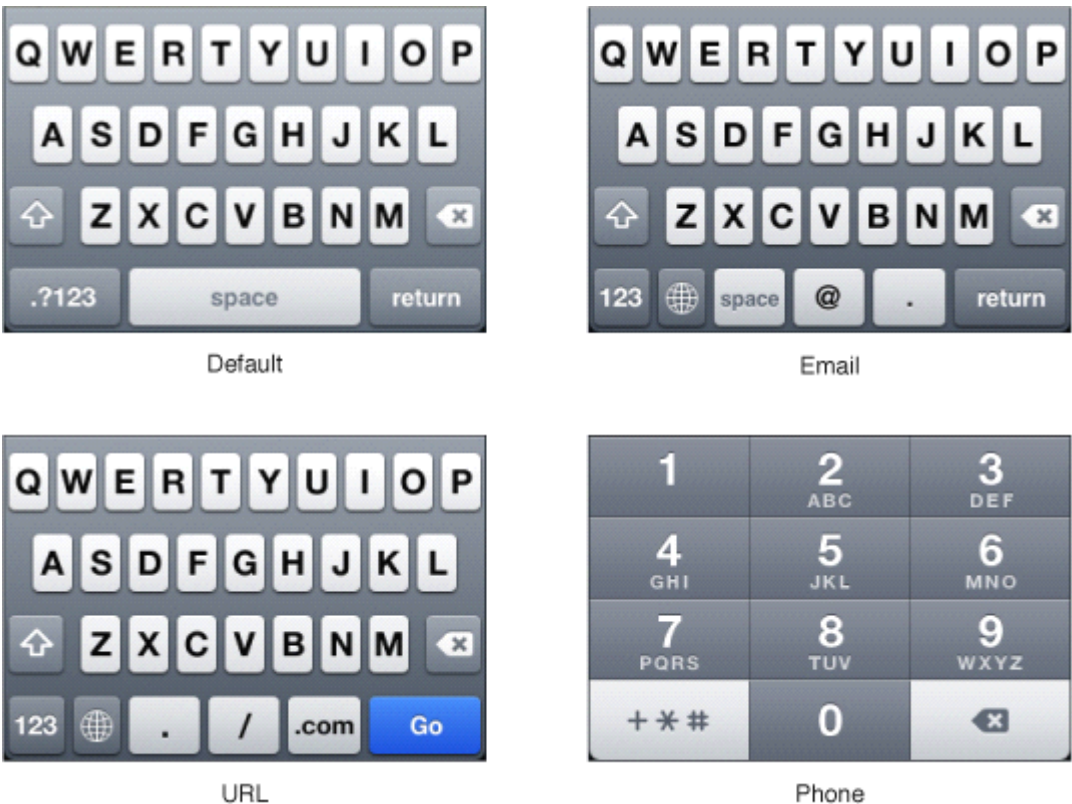
请注意：虽然 `UIWebView` 类并不直接支持 `UITextInputTraits` 协议，但您还是可以配置文本输入元素的一些键盘属性。特别值得一提的是，您可以在输入元素的定义中包含 `autocorrect` 和 `autocapitalization` 属性，通过这些属性来指定键盘的行为，如下面的例子所示：

```
<input type="text" size="30" autocorrect="off" autocapitalization="on">
```

您不能在输入元素中指定键盘的类型。web 视图显示的是缺省的键盘，但包含一些额外的控制，可以进行表单元素之间漫游。

缺省的键盘配置是为一般的文本输入设计的。图5-3显示了缺省的和其它的几个键盘配置。缺省键盘显示的是一个字母键盘，用户可以将它切换为数字和标点符号键盘。大多数其它键盘在都提供与缺省键盘类似的功能，同时又提供一些适合于特定任务的其它按键。但是，电话和数字键盘的布局显著不同，它们是特别为数字输入设计的。

图5-3 几个不同的键盘类型



为了实现不同的语言偏好，iPhone OS 还支持与不同语言相对应的输入法和键盘布局，图

5-4显示了部分输入法和布局。输入法和键盘布局是由用户语言偏好设置决定的。

图5-4 几个不同的键盘和输入法



管理键盘

虽然很多 UIKit 对象在响应用户交互时会自动显示键盘，但您的程序仍然需要配置和管理键盘。本文的下面部分将描述应用程序在键盘管理方面应该承担的责任。

接收键盘通告

当键盘被显示或隐藏的时候，iPhone OS 会向所有经过注册的观察者对象发出如下[通告](#)：

[UIKeyboardWillShowNotification](#)

[UIKeyboardDidShowNotification](#)

[UIKeyboardWillHideNotification](#)

[UIKeyboardDidHideNotification](#)

当键盘首次出现或者消失，以及键盘的所有者或应用程序的方向发生变化的任何时候，系统都会发出键盘通告。在上述的各种情况下，系统只发送与具体场景相关的消息集合。举例来说，如果键盘的所有者发生变化，系统只向当前的拥有者发送 `UIKeyboardWillHideNotification` 消息，但不发送 `UIKeyboardDidHideNotification` 消息，因为这个变化不会导致键盘最终被隐藏。`UIKeyboardWillHideNotification` 消息只是简单地通知键盘当前的所有者即将失去键盘焦点。而改变键盘的方向则会使系统发出上述的两种消息，因为每个方向的键盘是不同的，在显示新的键盘之前，必须先隐藏原来的键盘。

每个键盘通告都包含键盘在屏幕上的位置和尺寸。您应该使用通告中的信息来确定键盘的尺寸和位置，而不是假定键盘具有某个特定的尺寸或处于某个特定的位置。键盘在使用不同输入法时并不一定总是一样的，在不同版本的 iPhone OS 上也可能会发生变化。另外，即使对于特定的某种语言和某个系统版本，键盘的尺寸也会因为应用程序方向的不同而不同。作为例子，请看图5-5显示了 URL 键盘在肖像模式和景观模式下的相对尺寸。使用键盘通告中的信息可以确保得到正确的尺寸和位置信息。

图5-5 在肖像模式和景观模式下的相对键盘尺寸



请注意：info 字典中的 [UIKeyboardBoundsUserInfoKey](#) 键包含的矩形只能用于取得尺寸信息，不要将该矩形的原点（它的值总是为 {0.0, 0.0}）用于矩形计算。由于键盘是以动画的形式出现在它的位置上的，其实际的边界尺寸会随着时间的不同而不同，因此，info 字典中有 [UIKeyboardCenterBeginUserInfoKey](#) 和 [UIKeyboardCenterEndUserInfoKey](#) 两个键，用于保存键盘的起始和终止的位置，您可以根据这些位置计算出键盘的原点。

使用键盘通告的一个原因是为了重新定位被键盘遮掩的内容。有关如何进行重新定位的信息，请参见[“移动键盘下面的内容”](#)部分。

显示键盘

当 用户触击一个视图时，系统就会自动将该视图作为第一响应者。而当这种场景发生在包含可编辑文本的视图时，该视图就会启动一个文本编辑会话。如果当前键盘不可见，该视图会在编辑会话刚开始时请求系统显示键盘。如果键盘已经显示在屏幕上了，第一响应者的改变会导致来自键盘的文本输入被重定向到用户刚刚触击的视图上。

键盘是在视图变为第一响应者时自动被显示的，因此，您通常不需要为了显示它而做什么工作。但是，您可以通过调用视图对象的 [becomeFirstResponder](#) 方法来为可编辑的文本视图显示键盘。调用这个方法可以使目标视图成为第一响应者，并开始编辑过程，其效果和用户触击该视图是一样的。

如果您的应用程序在一个屏幕上管理几个基于文本的视图，则需要跟踪当前哪个视图是第一响应者，以便在需要的时候取消键盘的显示。

取消键盘

虽然键盘通常是自动显示的，但它并不自动取消。相反，您的应用程序需要在恰当的时机取消键盘。通常情况下，您在响应用户动作的时候进行这样的操作，比如当用户触击键盘上的 **Return** 或 **Done** 按键、或者触击应用程序界面上的其它按键时。根据键盘配置的不同，您可能需要在用户界面上加入额外的控件来取消键盘。

您可以调用作为当前第一响应者的文本视图的 [resignFirstResponder](#) 方法来取消键盘。当文本视图失去第一响应者的状态时，就会结束其当前的编辑会话，将这个变化通知它的委托对象，并取消键盘。换句话说，如果您有一个名为 `myTextField` 的变量，指向一个 `UITextField` 对象，假定该对象是当前的第一响应者，则可以简单地通过下面的代码来取消键盘：

```
[myTextField resignFirstResponder];
```

从这个点之后的所有操作都由文本对象自动处理。

移动键盘下面的内容

当系统收到显示键盘的请求时，就从屏幕的底部滑出键盘，并将它放在应用程序内容的上方。由于键盘位于您的内容的上面，所以有可能遮掩住用户希望编辑的文本对象。如果这种情况发生，就必须对内容进行调整，使目标对象保持可见。

需要做的调整通常包括暂时调整一或多个视图的尺寸和位置，从而使文本对象可见。管理带有键盘的文本对象的最简单方法是将它们嵌入到一个 [UIScrollView](#)（或其子类，如 [UITableView](#)）对象。当键盘被显示出来时，您需要做的只是调整滚动视图的尺寸，并将目标文本对象滚动到合适的位置。为此，在 [UIKeyboardDidShowNotification](#) 通告的处理代码中需要进行如下操作：

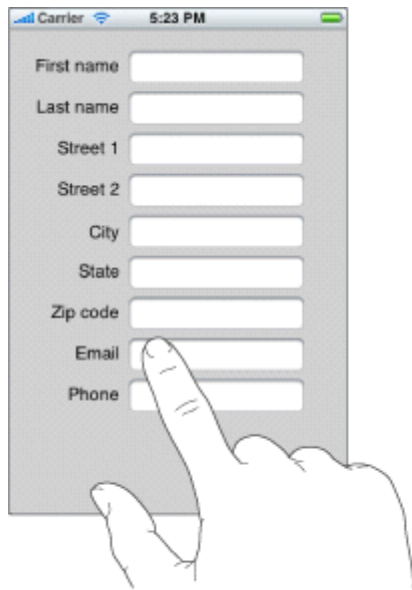
取得键盘的尺寸。

将滚动视图的高度减去键盘的高度。

将目标文本框滚动到视图中。

图5-6演示了一个简单的应用程序如何处理上述的几个步骤。该程序将几个文本输入框嵌入到 `UIScrollView` 对象中，当键盘出现时，通告处理代码首先调整滚动视图的尺寸，然后用 `UIScrollView` 类的 [scrollRectToVisible:animated:](#) 方法将被触击的文本框滚动到视图中。

图5-6 调整内容的位置，使其适应键盘



1. User taps email field



2. Scroll view resized to new height



3. Email field scrolled into view

请注意：在配置滚动视图时，请务必为所有的内容视图配置恰当的自动尺寸调整规则。在之前的图中，文本框实际上是一个 [UIView](#) 对象的子视图，该 [UIView](#) 对象又是 [UIScrollView](#) 对象的子视图。如果该 [UIView](#) 对象的 [UIViewAutoresizingFlexibleWidth](#) 和 [UIViewAutoresizingFlexibleHeight](#) 选项被设置了，则改变滚动视图的边框尺寸会同时改变它的边框，因而可能导致不可预料的结果。禁用这些选项可以确保该视图保持尺寸不变，并正确滚动。

程序清单5-1显示了如何注册接收键盘[通告](#)和如何实现相应的处理器方法。这段代码是由负责滚动视图管理的[视图控制器](#)实现的，其中 `scrollView` 变量是一个指向滚动视图对象的插座变量。每个处理器方法都从通告的 `info` 对象取得键盘的尺寸，并根据这个尺寸调整滚动视图的高度。此外，`keyboardWasShown:` 方法的任务是将当前活动的文本框矩形滚入视图，该文本框对象存储在一个定制变量中（在本例子中名为 `activeField`），该变量是视图控制器的一个成员变量，在 [textFieldDidBeginEditing:](#) 委托方法中进行赋值，委托方法本身的代码显示在[程序清单5-2](#)中（在这个例子中，视图控制器同时也充当所有文本输入框的委托）。

程序清单5-1 处理键盘通告

// Call this method somewhere in your view controller setup code.

```
- (void)registerForKeyboardNotifications
{
    [[NSNotificationCenter defaultCenter] addObserver:self
        selector:@selector(keyboardWasShown:)
        name:UIKeyboardDidShowNotification object:nil];

    [[NSNotificationCenter defaultCenter] addObserver:self
        selector:@selector(keyboardWasHidden:)
        name:UIKeyboardDidHideNotification object:nil];
}
```

// Called when the UIKeyboardDidShowNotification is sent.

```

- (void)keyboardWasShown:(NSNotification*)aNotification
{
    if (keyboardShown)
        return;

    NSDictionary* info = [aNotification userInfo];

    // Get the size of the keyboard.
    NSValue* aValue = [info
objectForKey:UIKeyboardBoundsUserInfoKey];
    CGSize keyboardSize = [aValue CGRectValue].size;

    // Resize the scroll view (which is the root view of the window)
    CGRect viewFrame = [scrollView frame];
    viewFrame.size.height -= keyboardSize.height;
    scrollView.frame = viewFrame;

    // Scroll the active text field into view.
    CGRect textFieldRect = [activeField frame];
    [scrollView scrollRectToVisible:textFieldRect animated:YES];

    keyboardShown = YES;
}

// Called when the UIKeyboardDidHideNotification is sent
- (void)keyboardWasHidden:(NSNotification*)aNotification
{
    NSDictionary* info = [aNotification userInfo];

    // Get the size of the keyboard.
    NSValue* aValue = [info
objectForKey:UIKeyboardBoundsUserInfoKey];
    CGSize keyboardSize = [aValue CGRectValue].size;

    // Reset the height of the scroll view to its original value
    CGRect viewFrame = [scrollView frame];
    viewFrame.size.height += keyboardSize.height;
    scrollView.frame = viewFrame;

    keyboardShown = NO;
}

```

上面程序清单中的 `keyboardShown` 变量是一个布尔值，用于跟踪键盘是否可见。如果您的用户界面有多个文本输入框，则用户可能触击其中的任意一个进行编辑。发生这种情况时，虽

然键盘并不消失，但是每次开始编辑新的文本框时，系统都会产生 `UIKeyboardDidShowNotification` 通告。您可以通过跟踪键盘是否确实被隐藏来避免多次减少滚动视图的尺寸。

程序清单5-2显示了一些额外的代码，视图控制器用这些代码来设置和清理之前例子中的 `activeField` 变量。在初始化时，界面中的每个文本框都将视图控制器设置为自己的委托。因此，当文本编辑框被激活的时候，这些方法就会被调用。更多关于文本框及其委托通告的信息，请参见 [UITextField 类参考](#)。

程序清单5-2 跟踪活动文本框的方法

```
- (void)textFieldDidBeginEditing:(UITextField *)textField
{
    activeField = textField;
}

- (void)textFieldDidEndEditing:(UITextField *)textField
{
    activeField = nil;
}
```

描画文本

除了显示和编辑文本的 `UIKit` 类之外，iPhone OS 还包含几个直接在屏幕上描画文本的方法。描画简单字符串的最简单有效的方法是使用 `NSString` 类的 `UIKit` 扩展，该扩展包含一些在屏幕上描画字符串的方法，并且可以描画时使用多种属性。还有一些方法，可以在真正描画之前计算渲染字符串所需要的尺寸，这些方法有助于更加精确布局应用程序的内容。

重要提示：由于性能上的考虑，您应该尽可能避免直接描画文本。对于静态文本，通过一或多个 `UILabel` 对象进行描画比使用定制描画例程要高效得多。类似地，`UITextField` 类也支持不同的风格，这些风格使您更加易于将可编辑的文本区域集成到您的内容中。

当您在界面上描画定制文本字符串时，请使用 `NSString` 方法。`UIKit` 包含一些对基本 `NSString` 类的扩展，用于在视图中描画字符串。这些方法使您可以精确调整文本的位置，以及将文本和视图内容进行融合；这个类的方法还可以根据指定的字体和风格属性计算文本的包围矩形。更多信息请参见 [NSString UIKit 扩展参考](#)。

如果您需要对描画过程中用到的字体有更多的控制，还可以使用 `Core Graphics` 框架中的函数来进行描画。`Core Graphics` 框架提供的方法可以对字形和文本进行精确描画和定位。有关这些函数及其用法的更多信息，请参见 [Quartz 2D 编程指南](#)和 [Core Graphics 框架参考](#)。

在 Web 视图中显示内容

如果您的用户界面包含 `UIWebView` 对象，就可以显示本地或网络上的内容。对于本地的内容，您可以动态创建，也可以使用文件，然后调用 [loadData:MIMETYPE:textEncodingName:baseURL:](#)或 [loadHTMLString:baseURL:](#)方法；如果要从网络加载，则需要创建一个 `NSURLRequest` 对象，然后传递给 web 视图对象的 [loadRequest:](#)

方法。

在发起一个基于网络的请求后，如果由于某种原因必须[释放](#) web 视图，则必须在释放之前取消待处理的请求。为此，您可以调用 web 视图的 [stopLoading](#) 方法。通常情况下，您可以在 web 视图的视图控制器的 [viewWillDisappear](#) 方法中执行这些代码。如果需要确定一个请求是否处于等待状态，可以通过 web 视图的 [loading](#) 属性来判断。

文件和网络

运行在 iPhone OS 系统上的应用程序可以通过各种 Core OS 和 Core Services [框架](#)来访问本地的文件系统和网络。读写本地文件系统的能力使您可以保存用户数据和应用程序状态，以备后用；而访问网络的能力则使您可以和网络服务器进行交流，进而实现远程操作的执行和数据的收发。

文件和数据管理

iPhone OS 系统上的文件和用户的媒体数据及个人文件共享闪存上的空间。出于安全的目的，您的应用程序被放在其自己的目录下，并且只能对该目录进行读写。本章的下面部分将描述应用程序本地文件系统的结构及几个读写文件的技术。

常用目录

出于安全的目的，应用程序只能将自己的数据和偏好设置写入到几个特定的位置上。当应用程序被安装到设备上时，系统会为其创建一个家目录。表6-1列出了应用程序家目录下的一些重要子目录，您的程序可能需要对其进行访问。表中还描述了每个目录的设计目的和访问限制，以及 iTunes 是否对该目录下的内容进行备份。有关备份和恢复过程的更多信息，请参见[“备份和恢复”](#)部分；有关应用程序家目录本身的信息，则请参见 [“应用程序沙箱”](#)部分。

表 6-1 iPhone 应用程序的目录目录

描述	描述
<code><Application_Home>/AppName.app</code>	这是 程序包 目录，包含应用程序的本身。由于应用程序必须经过签名，所以您在运行时不能对这个目录中的内容进行修改，否则可能会使应用程序无法启动。 在 iPhone OS 2.1及更高版本的系统，iTunes 不对这个目录的内容进行备份。但是，iTunes 会对在 App Store 上购买的应用程序进行一次初始的同步。
<code><Application_Home>/Documents/</code>	您应该将所有的应用程序数据文件写入到这个目录下。这个目录用于存储用户数据或其它应该定期备份的信息。有关如何取得这个目录路径的信息，请参见 “获取应用程序目录的路径” 部分。 iTunes 会备份这个目录的内容。
<code><Application_Home>/Library/Preferences</code>	这个目录包含应用程序的偏好设置文件。您不应该直接创建偏好设置文件，而是应该使用 NSUserDefaults 类或 CFPreferences API 来取得和设置应用程序的偏好，详情请参见 “添加 Settings 程序包” 部分。 iTunes 会备份这个目录的内容。

<Application_Home>/Library/Caches

这个目录用于存放应用程序专用的支持文件，保存应用程序再次启动过程中需要的信息。您的应用程序通常需要负责添加和删除这些文件，但在对设备进行完全恢复的过程中，iTunes 会删除这些文件，因此，您应该能够在必要时重新创建。您可以使用[“获取应用程序目录的路径”](#)部分描述的接口来获取该目录的路径，并对其进行访问。

在 iPhone OS 2.2及更高版本，iTunes 不对这个目录的内容进行备份。

<Application_Home>/tmp/

这个目录用于存放临时文件，保存应用程序再次启动过程中不需要的信息。当您的应用程序不再需要这些临时文件时，应该将其从这个目录中删除（系统也可能在应用程序不运行的时候清理留在这个目录下的文件）。有关如何获得这个目录路径的信息，请参见[“获取应用程序目录的路径”](#)部分。

在 iPhone OS 2.1及更高版本，iTunes 不对这个目录的内容进行备份。

备份和恢复

您不需要在应用程序中为备份和恢复操作做任何准备。在 iPhone OS 2.2及更高版本的系统中，当设备被连接到计算机并完成同步时，iTunes 会对除了下面这些目录之外的所有文件进行增量式的备份：

<Application_Home>/AppName.app

<Application_Home>/Library/Caches

<Application_Home>/tmp

虽然 iTunes 确实对应用程序的[程序包](#)本身进行备份，但并不是在每次同步时都进行这样的操作。通过设备上的 App Store 购买的应用程序在下次设备和 iTunes 同步时进行备份。而在之后的同步操作中，应用程序并不进行备份，除非应用程序包本身发生了变化（比如由于应用程序被更新了）。

为了避免同步过程花费太长时间，您应该有选择地往应用程序家目录中存放文件。

<Application_Home>/Documents 目录应该用于存放用户数据文件或不容易在应用程序中重新创建的文件。存储临时数据的文件应该放在 *Application Home/tmp* 目录，而且应该在不需要的时候将其删除。如果您的应用程序需要创建用于下次启动的数据文件，则应该将那些文件放到 *Application Home/Library/Caches* 目录下。

请注意：如果您的应用程序需要创建数据量大或频繁变化的文件，则应该考虑将它们存储在 *Application Home/Library/Caches* 目录下，而不是 <Application_Home>/Documents 目录。备份大数据文件会使备份过程显著变慢，备份频繁变化（因此必须频繁备份）的文件也同样如此。将这些文件放到 Caches 目录下可以避免每次同步都对其进行备份（在 iPhone OS 2.2及更高版本）。

有关如何在应用程序中使用目录的更多信息，请参见[表6-1](#)。

在应用程序更新过程中被保存的文件

更新应用程序就是将用户下载的新版应用程序代替之前的版本。在这个过程中，iTunes 会将更新过后的应用程序安装到新的应用程序目录下，并在删除老版本之前，将用户数据文件转移

到新的应用程序目录下。在更新的过程中，iTunes 保证如下目录中的文件会得以保留：

<Application_Home>/Documents

<Application_Home>/Library/Preferences

虽然其它用户目录下的文件也可能被转移，但是您不应该假定更新之后该文件还仍然存在。

Keychain 数据

keychain 是一个安全、经过加密保护的容器，用于保存密码和其它秘密信息。应用程序的 keychain 数据存储在应用程序沙箱之外。如果应用程序被卸载，则该数据会自动被删除。当用户通过 iTunes 备份应用程序数据时，keychain 数据也会被备份。然而，keychain 数据只能被恢复到之前做备份的设备上。应用程序的更新并不影响其 keychain 数据。

有关 iPhone OS keychain 的更多信息，请参见 [Keychain 服务编程指南](#) 文档中的“[Keychain 服务的概念](#)”部分。

获取应用程序目录的路径

系统在各个级别上都提供了用于获取应用程序沙箱目录路径的编程方法。然而，取得这些路径的推荐方式还是使用 Cocoa 编程接口。[NSHomeDirectory](#) 函数（在 Foundation 框架中）负责返回顶级家目录的路径——也就是包含应用程序、Documents、Library、和 tmp 目录的路径。除了这个函数，您还可以用 [NSSearchPathForDirectoriesInDomains](#) 和 [NSTemporaryDirectory](#) 函数来取得 Documents、Caches、和 tmp 目录的准确路径。

[NSHomeDirectory](#) 和 [NSTemporaryDirectory](#) 函数都通过 [NSString](#) 对象返回正确格式的路径。您可以通过 [NSString](#) 类提供的与路径相关的方法来修改路径信息或创建新的路径字符串。举例来说，在取得临时的目录路径之后，您可以附加一个文件名，并用结果字符串在临时目录下创建给定名称的文件。

请注意：如果您使用带有 ANSI C 编程接口的框架——包括那些接受路径参数的接口——请记住 [NSString](#) 对象和其在 Core Foundation 框架中的等价类型之间是“免费桥接”的。这意味着您可以将一个 [NSString](#) 对象（比如上述某个函数的返回结果）强制类型转换为一个 [CFStringRef](#) 类型，如下面的例子所示：

```
CFStringRef homeDir =  
(CFStringRef)NSHomeDirectory();
```

有关免费桥接的更多信息，请参见 *Carbon-Cocoa 集成指南* 文档。

Foundation 框架中的 [NSSearchPathForDirectoriesInDomains](#) 函数用于取得几个应用程序相关目录的全路径。在 iPhone OS 上使用这个函数时，第一个参数指定正确的搜索路径常量，第二个参数则使用 [NSUserDomainMask](#) 常量。表6-2列出了大多数常用的常量及其返回的目录。

表6-2 常用的搜索路径常量

常量	目录
NSDocumentDirectory	<Application_Home>/Documents
NSCachesDirectory	<Application_Home>/Library/Caches
NSApplicationSupportDirectory	<Application_Home>/Library/Application Support

由于 [NSSearchPathForDirectoriesInDomains](#) 函数最初是为 Mac OS X 设计的，而 Mac OS X 上可能存在多个这样的目录，所以它的返回值是一个路径数组，而不是单一的路径。在 iPhone OS 上，结果数组中应该只包含一个给定目录的路径。程序清单6-1显示了这个函数的典型用法。

程序清单6-1 取得指向应用程序 Documents 目录的文件系统路径

```
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,  
NSUserDomainMask, YES);
```

```
NSString *documentsDirectory = [paths objectAtIndex:0];
```

在调用 `NSSearchPathForDirectoriesInDomains` 函数时，您可以使用 `NSUserDomainMask` 之外的其它域掩码参数，或者使用表6-2之外的其它目录常量，但是应用程序不能向其返回的目录写入数据。举例来说，如果您指定 [NSApplicationDirectory](#) 作为目录参数，同时指定 [NSSystemDomainMask](#) 作为域掩码参数，则可以返回（设备上的）`/Applications` 路径，但是，您的应用程序不能往该位置写入任何文件。

另外一个需要记住的考虑是，不同平台的目录位置是不一样的。`NSSearchPathForDirectoriesInDomains`、[NSHomeDirectory](#)、[NSTemporaryDirectory](#)、和其它类似函数的返回路径取决于应用程序运行在设备还是仿真器上。作为例子，[程序清单6-1](#)上显示的函数调用在设备上返回的路径（`documentsDirectory`）大致如下：

```
/var/mobile/Applications/30B51836-D2DD-43AA-BCB4-9D4DADFED6A2/Documents
```

但是，它在仿真器上返回的路径则具有如下的形式：

```
/Volumes/Stuff/Users/johnDoe/Library/Application Support/iPhone  
Simulator/User/Applications/118086A0-FAAF-4CD4-9A0F-CD5E8D287270/Documents
```

在读写用户偏好设置时，请使用 [NSUserDefaults](#) 类或 `CFPreferences` API。这些接口使您免于构造 `Library/Preferences/` 目录路径和直接读写偏好文件。有关使用这些接口的更多信息，请参见“[添加 Settings 程序包](#)”部分。

如果应用程序的程序包中包含声音、图像、或其它资源，则应该使用 [NSBundle](#) 类或 [CFBundleRef](#) 封装类型来装载那些资源。程序包知道应用程序内部资源应该在什么位置上，此外，它还知道用户的语言偏好，能够自动选择本地化的资源。有关程序包的更多信息，请参见“[应用程序的程序包](#)”部分。

文件数据的读写

iPhone OS 提供了如下几种读、写、和管理文件的方法：

Foundation 框架：

如果您可以将应用程序数据表示为一个属性列表，则可以用 [NSPropertyListSerialization](#) API 来将属性列表转换为一个 [NSData](#) 对象，然后通过 `NSData` 类的方法将数据对象写入磁盘。

如果应用程序的模型对象采纳了 [NSCoding](#) 协议，则可以通过 [NSKeyedArchiver](#) 类、特别是它的 [archivedDataWithRootObject:](#) 方法将模型对象图进行归档。

Foundation 框架中的 [NSFileHandle](#) 类提供了随机访问文件内容的方法。

Foundation 框架中的 [NSFileManager](#) 类提供了在文件系统中创建和操作文件的方法。

Core OS 调用：

诸如 `fopen`、`fread`、和 `fwrite` 这些调用可以用于对文件进行顺序或随机读写。

`mmap` 和 `munmap` 调用是将大文件载入内存并访问其内容的有效方法。

请注意：上面的 Core OS 调用列表只是列举一些较为常用的例子。更完全的可用函数列表请参见 [iPhone OS 手册](#) 的第三部分中的函数列表。

本章的下面部分将描述如何使用一些高级技术来进行文件的读写。有关 Foundation 框架中与文件相关类的更多信息，请参见 [Foundation 框架参考](#)。

属性列表数据的读写

[属性列表](#)是一种数据表示形式，用于封装几种 Foundation（及 Core Foundation）的数据类型，包括[字典](#)、[数组](#)、[字符串](#)、[日期](#)、[二进制数据](#)、[数值及布尔值](#)。属性列表通常用于存储结构化的配置数据。举例来说，每个 Cocoa 和 iPhone 应用程序中都有一个 Info.plist 文件，它就是用于存储应用程序本身配置信息的属性列表。您自己也可以用属性列表来存储其它信息，比如应用程序退出时的状态等。

在代码中，属性列表的构造通常从构造一个字典或数组、并将它作为容器对象开始，然后在容器中加入其它的属性列表对象，（可能）包含其它的字典和数组。字典的键必须是字符串对象，键的值则是 [NSDictionary](#)、[NSArray](#)、[NSString](#)、[NSDate](#)、[NSData](#)、和 [NSNumber](#) 类的实例。

对于可以将数据表示为属性列表对象的应用程序（比如 NSDictionary 对象），您可以用程序清单6-2所示的方法将属性列表写入磁盘。该方法将属性列表序列化为 NSData 对象，然后调用 writeApplicationDataToFile:方法（其实现如[程序清单6-4](#)所示）将数据写入磁盘。

程序清单6-2 将属性列表对象转换为 NSData 对象并写入存储

```
- (BOOL)writeApplicationPlist:(id)plist toFile:(NSString *)fileName {
    NSString *error;
    NSData *pData = [NSPropertyListSerialization dataFromPropertyList:plist
format:NSPropertyListBinaryFormat_v1_0 errorDescription:&error];
    if (!pData) {
        NSLog(@"%@@", error);
        return NO;
    }
    return ([self writeApplicationData:pData toFile:(NSString *)fileName]);
}
```

在 iPhone OS 系统上保存属性列表文件时，采用二进制格式进行存储是很重要的。在编码时，可以通过为 [dataFromPropertyList:format:errorDescription:](#) 方法的 *format* 参数指定 [NSPropertyListBinaryFormat_v1_0](#)值来实现。二进制格式比其它基于文本的格式紧凑得多，这种紧凑不仅使属性列表在用户设备上占用的空间最小，还可以减少读写属性列表的时间。程序清单6-3的代码展示了如何从磁盘装载属性列表，并重新生成属性列表中的对象。

程序清单 6-3 从应用程序的 Documents 目录读取属性列表对象

```
- (id)applicationPlistFromFile:(NSString *)fileName {
    NSData *retData;
    NSString *error;
    id retPlist;
    NSPropertyListFormat format;

    retData = [self applicationDataFromFile:fileName];
    if (!retData) {
        NSLog(@"Data file not returned.");
        return nil;
    }
}
```

```

retPlist = [NSPropertyListSerialization propertyListFromData:retData
mutabilityOption:NSPropertyListImmutable format:&format errorDescription:&error];
if (!retPlist){
    NSLog(@"Plist not returned, error: %@", error);
}
return retPlist;
}

```

有关属性列表和 `NSPropertyListSerialization` 类的更多信息，请参见[属性列表编程指南](#)。

用归档器进行数据读写

[归档器](#)的作用是将任意的对象集合转换为字节流。这听起来像是 `NSPropertyListSerialization` 类采用的过程，但它们之间有一个重要的区别。属性列表序列化只能转换一个有限集合的数据类型（大多数是数量类型），而归档器可以转换任意的 Objective-C 对象、数量类型、数组、结构、字符串、及更多其它类型。

归档过程的关键在于目标对象的本身。归档器操作的对象必须遵循 [NSCoding 协议](#)，该协议定义了读写对象状态的接口。归档器在编码一组对象时，会向每个对象发送一个 `encodeWithCoder:` 消息，目标对象则在这个方法中将自身的关键状态信息写入到对应的档案中。解档过程的信息流与此相反，在解档过程中，每个对象都会接收到一个 `initWithCoder:` 消息，用于从档案中读取当前状态信息，并基于这些信息进行初始化。解档过程完成后，字节流就被重新组成一组与之前写入档案时具有相同状态的新对象。

Foundation 框架支持两种归档器—顺序归档和基于键的归档。基于键的归档器更加灵活，是应用程序开发中推荐使用的归档器。下面的例子显示如何用一个基于键的归档器对一个对象图进行归档。`_myDataSource` 对象的 `representation` 方法返回一个单独的对象（可能是一个数组或字典），指向将要包含到档案中的所有对象，之后该数据对象就被写入由 `myFilePath` 变量指定路径的文件中。

```

NSData *data = [NSKeyedArchiver archivedDataWithRootObject:[_myDataSource
representation]];
[data writeToFile:myFilePath atomically:YES];

```

请注意：您还可以向 `NSKeyedArchiver` 对象发送 [archiveRootObject:toFile:](#) 消息，以便在一个步骤中完成档案的创建和将档案写入存储。

您可以简单地通过相反的流程来装载磁盘上的档案内容。在装载磁盘数据之后，可以通过 [NSKeyedUnarchiver](#) 类及其 [unarchiveObjectWithData:](#) 方法来取回[模型对象](#)图。例如，您可以用下面的代码来解档之前例子中的数据：

```

NSData* data = [NSData dataWithContentsOfFile:myFilePath];
id rootObject = [NSKeyedUnarchiver unarchiveObjectWithData:data];

```

更多如何使用归档器和如何使对象支持 `NSCoding` 协议的信息，请参见 [Cocoa 的归档和序列化编程指南](#)。

将数据写到 Documents 目录

有了封装应用程序数据的 `NSData` 对象（或者是档案，或者是序列化了的属性列表）之后，您就可以调用程序清单6-4所示的方法将数据写到应用程序的 Documents 目录中。

程序清单6-4 将数据写到应用程序的 Documents 目录

```
- (BOOL)writeApplicationData:(NSData *)data toFile:(NSString *)fileName {
    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
    NSUserDomainMask, YES);
    NSString *documentsDirectory = [paths objectAtIndex:0];
    if (!documentsDirectory) {
        NSLog(@"Documents directory not found!");
        return NO;
    }
    NSString *appFile = [documentsDirectory stringByAppendingPathComponent:fileName];
    return ([data writeToFile:appFile atomically:YES]);
}
```

从 Documents 目录读取数据

为了从应用程序的 Documents 目录读取文件，您首先需要根据文件名构建相应的路径，然后以期望的方法将文件内容读入内存。对于相对较小的文件——也就是尺寸小于几个内存页面的文件——您可以用程序清单6-5中的代码来取得文件内容。该代码首先为 Documents 目录下的文件构建一个全路径，并为这个路径创建一个数据对象，然后返回。

程序清单6-5 从应用程序的 Documents 目录读取数据

```
- (NSData *)applicationDataFromFile:(NSString *)fileName {
    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
    NSUserDomainMask, YES);
    NSString *documentsDirectory = [paths objectAtIndex:0];
    NSString *appFile = [documentsDirectory stringByAppendingPathComponent:fileName];
    NSData *myData = [[[NSData alloc] initWithContentsOfFile:appFile] autorelease];
    return myData;
}
```

对于载入时需要多个内存页面的文件，应该避免一次性地装载整个文件。如果您只是计划使用部分文件，这一点就尤其重要。对于大文件，您应该考虑用 `mmap` 函数或 [NSData](#) 的 [initWithContentsOfFile](#) 方法来将文件映射到内存。

到底是采用映射文件还是直接装载取决于您的考虑。如果只需要少量（3-4）内存页面，则将整个文件载入内存相对安全一些。但是，如果您的文件需要数十或上百个页面，则将文件映射到内存可能更为有效一些。当然，无论采用什么方法，您都应该测量应用程序的性能，确定装载文件和为其分配必要内存需要多长时间。

文件访问的指导原则

在您创建文件或写入文件数据时，请记住下面这些指导原则：

使写入磁盘的数据量尽可能少。文件操作速度相对较慢，且涉及到 Flash 盘的写操作，有一定的寿命限制。下面这些具体的小贴士可以帮助您最少化与文件相关的操作：

只写入发生变化的文件部分，但要尽可能对变化进行累计，避免在只有少数字节发生改变时对整个文件进行写操作。

在定义文件格式时，将频繁变化的内容放在一起，以便使每次需要写入磁盘的总块数最少。如果您的数据是需要随机访问的结构化内容，则可以将它们存储在 Core Data 持久仓库或 SQLite 数据库中。如果您处理的数据量可能增长到数兆以上，这一点尤其重要。

避免将缓存文件写入磁盘。这个原则的唯一例外是：在应用程序退出时，您需要写入某些状态信息，使程序在下次启动时可以回到之前的状态。

保存状态信息

当用户按下 Home 键时，iPhone OS 会退出您的应用程序，返回到 Home 屏幕。类似地，如果您的应用程序打开一个由其它应用程序处理的 URI 模式，iPhone OS 也会退出您的应用程序，在相应的应用程序上打开该 URI。换句话说，在 Mac OS X 上引起应用程序挂起或转向后台的动作，在 iPhone OS 上都会使其退出。这些动作在移动设备上经常发生，因此，您的应用程序必须改变管理可变数据和程序状态的方式。

大多数桌面应用程序由用户手工选择将文件存入磁盘的时机，与此不同的是，iPhone 应用程序应该在工作流的关键点上自动保存已发生的变化。究竟何时保存数据由您自己来决定，但是有两个潜在的时间点：或者在用户做出改变之后马上进行保存；或者将同一页面上的变化累计成批，然后在退出该页面、显示新页面、或者应用程序退出的时候进行保存。在任何情况下，您不应该让用户漫游到新的页面而不保存之前页面的内容。

当您的应用程序被要求退出时，应该将当前状态保持到临时的缓存文件或偏好数据库中。在用户下次启动应用程序时，可以根据这些信息将程序恢复到之前的状态。您保持的状态信息应该尽可能少，但同时又足够使应用程序恢复到恰当的点。您不必一定要显示用户上次退出时操作的页面，如果那样做并不合理的话。比如，如果一个用户在编辑某个联系人的时候离开了 Phone 程序，那么在下次运行时，Phone 程序显示的是联系人的顶级列表，而不是该联系人的编辑屏幕。

大小写敏感性

iPhone OS 设备的文件系统是大小写敏感的。在处理文件名的任何时候，您都应该确保大小写准确匹配，否则可能不能打开或访问文件。

网络

iPhone OS 的网络栈中包含几个基于（iPhone 和 iPod touch 设备上的）无线通讯硬件的编程接口。主编程接口是 CFNetwork 框架，该框架在 BSD 套接字和 Core Foundation 框架的封装类型之上，实现了网络实体间的通讯。您也可以使用 Foundation 框架的 [NSSStream](#) 类和位于系统 Core OS 层中的 BSD 套接字来进行通讯。

本文的下面部分将为需要集成网络功能的开发者提供一些专门针对 iPhone 的贴士。有关如何通过 CFNetwork 框架实现网络通讯的信息，请参见 [CFNetwork 编程指南](#)和 [CFNetwork 框架参考](#)；有关如何使用 NSSStream 类的信息，则请参见 [Foundation 框架参考](#)。

有效进行网络通讯的贴士

在实现收发网络数据的代码时，请记住这是设备上最耗电的操作之一。最少化收发数据的时间有助于提高电池的使用寿命。为此，您在编写与网络相关的代码时需要考虑如下贴士：对于您自己控制的协议，请将数据格式定义得尽可能紧凑。

避免使用聊天式的协议进行通讯。

在任何可能的时候，将数据包成群传输。

蜂窝网和 Wi-Fi 无线网都被设计为在没有数据传输活动时关闭电源。然而，根据无线网络的不同，这样做可能需要花几秒钟的时间。如果您的应用程序每隔数秒就发送少量的数据，则即使无线装置实际上并没做什么，也会一直保持电源打开，持续耗电。相比于经常性地传输少量数据，一次性传递所有数据或间隔时间较长但每次传递数据量较大是更好的选择。在进行网络通讯时，意识到数据包在任何时候都可能丢失是很重要的。在编写网络通讯代码时，请务必在出现错误时进行处理，使程序尽可能强壮。实现响应网络条件变化的处理程序是完全合理的，但如果这些处理程序始终没有被调用，也不要觉得奇怪。举例来说，在网络服务消失时，Bonjour 的网络回调函数并不总是立即被调用。当接收到某个服务即将消失的通告时，Bonjour 系统服务确实立即调用浏览回调函数（browsing callbacks），然而，网络服务可能没有通告就消失了，如果设备提供的网络服务意外地丢掉网络连接，或者通告在传递中丢失，就可能出现这种情况。

使用 Wi-Fi

如果您的应用程序通过 Wi-Fi 无线信号访问网络，则必须将这个事实通知系统，即在应用程序的 Info.plist 文件中包含 UIRequiresPersistentWiFi 键。包含这个键使系统知道在检测到活动的 Wi-Fi 热区时应该弹出网络选择框，同时还使系统知道在您的应用程序运行时不应试图关闭 Wi-Fi 硬件。

为了防止 Wi-Fi 硬件消耗太多的电能，iPhone OS 内置一个定时器，如果在30分钟内没有应用程序通过 UIRequiresPersistentWiFi 键请求使用 Wi-Fi，就会完全关闭该硬件。如果用户启动某个包含该键的应用程序，则在该程序的生命周期中，iPhone OS 会有效地禁用该定时器。但是一旦该程序退出，系统就会重新启用该定时器。

请注意：即使 UIRequiresPersistentWiFi 键的值为 true，在设备空闲（也就是处于屏幕锁定状态）时也是没有效果的。在那种情况下，应用程序被认为是不活动的，虽然它可能在某些级别上还在工作，但没有 Wi-Fi 连接。

有关 UIRequiresPersistentWiFi 键及 Info.plist 文件中其它键的更多信息，请参见[“信息属性列表”](#)部分。

飞行模式警告

当应用程序启动时，如果设备处于飞行模式，系统可能会显示一个对话框通知用户。系统仅下面的所有条件都满足时才会显示这个通知对话框：

应用程序的信息[属性列表](#)(Info.plist) 文件包含 UIRequiresPersistentWiFi 键，且该键的值被设置为 true。

应用程序启动的同时设备处于飞行模式。

在切换到飞行模式后设备上的 Wi-Fi 还没有被手工激活。

多媒体支持

无论多媒体功能在您的应用程序中是处于中心地位，还是偶尔被使用，iPhone 用户都期望有很高的品质。视频应该充分利用设备携带的高分辨率屏幕和高帧率，而引人注目的音频也会对应应用程序的总体用户体验有不可估量的增强作用。

您可以利用 iPhone OS 的多媒体[框架](#)来为应用程序加入下面这些功能：

高品质的音频录制和回放

生动的游戏声音

实时的声音聊天

用户 iPod 音乐库内容的回放

在支持的设备上进行视频的回放和录制

本章将介绍 iPhone OS 上为应用程序添加音视频功能的多媒体技术。

在 iPhone OS 上使用声音

iPhone OS 为应用程序提供一组丰富的声音处理工具。根据功能的不同，这些工具被安排到如下的框架中：

如果希望用简单的 Objective-C 接口进行音频的播放和录制，可以使用 AV Foundation 框架。如果要播放和录制带有同步能力的音频、解析音频流、或者进行音频格式转换，可以使用 Audio Toolbox 框架。

如果要连接和使用音频处理插件，可以使用 Audio Unit 框架。

如果希望在游戏和其它应用程序中回放位置音频，需要使用 OpenAL 框架。iPhone OS 对 OpenAL 1.1 的支持是建立在 Core Audio 基础上的。

如果希望播放 iPod 库中的歌曲、音频书、或音频播客，需要使用 Media Player 框架中的 iPod 媒体库访问接口。

Core Audio 框架（和其它音频框架对等）中提供所有 Core Audio 服务需要使用的数据类型。本部分将就如何着手实现各种音频功能提供一些指导，如下表所示：

播放用户 iPod 库中的歌曲、音频播客、以及音频书，请参见[“用 iPod 媒体库访问接口播放媒体项”](#)部分。

播放警告及用户界面声音效果，或者使具有震动功能的设备发生震动，可以使用系统声音服务，具体请参见[“使用系统声音服务播放短声音及激活震动”](#)部分。

如果要用最少量的代码播放和录制音频，可以使用 AV Foundation 框架，具体参见[“通过 AVAudioPlayer 类轻松播放声音”](#)及 [“用 AVAudioRecorder 类进行音频录制”](#)部分。

如果需要提供全功能的音频回放，包括立体声定位、音量控制、和同期声（simultaneous sounds），可以使用 OpenAL，具体参见[“使用 OpenAL 播放和定位声音”](#)部分。

如果要提供最低延迟的音频，特别是需要同时进行音频输入输出（比如 VoIP 应用程序）时，请使用 I/O 音频单元，具体请参见[“iPhone OS 中的音频单元支持”](#)部分。

如果您播放的声音需要精确的控制（包括同步），可以使用音频队列服务，具体参见[“用音频队列服务播放和控制声音”](#)部分，音频队列服务还支持音频录制，具体请见[“用音频队列服务进行音频录制”](#)部分的描述。

如果需要解析来自网络连接的音频流，请使用音频文件流服务，具体参见[“解析音频流”](#)部分。请务必阅读本文接下来的部分，即[“基础：硬件编解码器、音频格式、和音频会话”](#)部分，以了解在基于 iPhone OS 的设备上音频工作机制的关键信息；而且也请您阅读[“iPhone 音频的最佳实践”](#)部分，该部分提供了一些指导原则，并列举了一些能得到最好性能和最佳用户体验的音频和文件格式。

当您准备好进一步学习时，请访问 [iPhone Dev Center](#)。这个开发者中心包含各种指南文档、实例代码、及更多其它信息。有关如何执行常见音频任务的贴士，请参见 [音频&视频编程的 How-To's](#) 部分；如果需要 iPhone OS 音频开发的深入解释，则请参见 [Core Audio 概述](#)、[音频队列服务编程指南](#)、和 [音频会话编程指南](#)。

基础：硬件编解码器、音频格式、和音频会话

在开始 iPhone 音频开发之前，了解 iPhone OS 设备的一些硬软件架构知识是很有帮助的。

iPhone 音频硬件编解码

iPhone OS 的应用程序可以使用广泛的音频数据格式。从 iPhone OS 3.0 开始，这些格式中的大多数都可以支持基于软件的编解码。您可以同时播放多路各种格式的声音，虽然出于性能的考虑，您应该针对给定的场景选择最佳的格式。通常情况下，硬件解码带来的性能影响比软件解码要小。

下面这些 iPhone OS 音频格式可以利用硬件解码进行回放：

AAC

ALAC (Apple Lossless)

MP3

通过硬件，设备每次只能播放这些格式中的一种。举例来说，如果您正在播放的是 MP3 立体声，则第二个同时播放的 MP3 声音就只能使用软件解码。类似地，您不能通过硬件同时播放一个 AAC 声音和一个 ALAC 声音。如果 iPod 应用程序正在后台播放 AAC 声音，则您的应用程序只能使用软件解码来播放 AAC、ALAC、和 MP3 音频。

为了以最佳性能播放多种声音，或者为了在 iPod 程序播放音乐的同时能更有效地播放声音，可以使用线性 PCM（无压缩）或者 IMA4（有压缩）格式的音频。

如果需要了解如何检测设备硬软件编解码器是否可用，请查阅 [音频格式服务参考](#) 中有关 [kAudioFormatProperty_HardwareCodecCapabilities](#) 常量的讨论。

音频回放和录制格式

下面是一些 iPhone OS 支持的音频回放格式：

AAC

HE-AAC

AMR (Adaptive Multi-Rate，是一种语音格式)

ALAC (Apple Lossless)

iLBC (互联网 Low Bitrate Codec，另一种语音格式)

IMA4 (IMA/ADPCM)

- 线性 PCM (无压缩)

μ-law 和 a-law

MP3 (MPEG-1 音频第3层)

下面是一些 iPhone OS 支持的音频录制格式：

ALAC (Apple Lossless)

iLBC (互联网 Low Bitrate Codec, 用于语音)

IMA/ADPCM (IMA4)

线性 PCM

μ-law 和 a-law

下面的列表总结了 iPhone OS 如何支持单路或多路音频格式：

线性 PCM 和 IMA4 (IMA/ADPCM) 在 iPhone OS 上，您可以同时播放多路线性 PCM 或 IMA4 声音，而不会导致 CPU 资源的问题。这一点同样适用于 AMR 和 iLBC 语音品质格式，以及 μ-law 和 a-law 压缩格式。在使用压缩格式时，请检查声音的品质，确保满足您的需要。

AAC、MP3、和 ALAC (Apple Lossless) AAC、MP3、和 ALAC 声音的回放可以使用 iPhone OS 设备上高效的硬件解码，但是这些编解码器共用一个硬件路径，通过硬件，设备每次只能播放上述格式的一种。

AAC、MP3、和 ALAC 的回放共用同一硬件路径的事实会对“合作播放”风格的应用程序（比如虚拟钢琴）产生影响。如果用户在 iPod 程序上播放上述三种格式之一的音频，则您的应用程序—如果要和该音频一起播放声音—需要使用软件解码。

音频会话

Core Audio 的音频会话接口（具体描述请见[音频会话服务参考](#)）使应用程序可以为自己定义一般的音频行为，并在更大的音频上下文良好工作。您能够影响的行为有：

您的音频在 Ring/Silent 切换过程中是否变为无声

在屏幕锁定状态时您的音频是否停止

当您的音频开始播放时，iPod 音频是继续播放，还是变为无声

更大的音频上下文包括用户所做的改变，比如用户插入耳机，处理 Clock 和 Calendar 这样的警告事件，或者处理呼入的电话。通过音频会话，您可以对这样的事件做出恰当的响应。

音频会话服务提供了三种编程特性，如表7-1所述。

表7-1 音频会

话接口提供的

特性

音频会话

特性

描述

范畴

范畴是标识一组应用程序音频行为的键。您可以通过范畴的设置来指示自己希望得到的音频行为，比如希望在屏幕锁定状态时继续播放音频。

中断和路由变化

当您的音频发生中断或中断结束，以及当硬件音频路由发生变化时，音频会话会发出[通告](#)，使您可以优雅地响应发生在更大音频环境中的变化—比如由于电话呼入而导致的中断。

硬件特征

您可以通过查询音频会话来了解应用程序所在的设备的特征，比如硬件采样率，硬件通道数量，以及是否有音频输入。

[AVAudioSession 类参考](#)和 [AVAudioSessionDelegate 协议参考](#)描述了一个管理音频会话的精简接口。如果要使音频会话支持中断，则可以直接使用基于 C 语言的音频会话服务接口，该接口的描述请见[音频会话服务参考](#)。在应用程序中，这两个接口的代码可以混用及互相匹配。音频会话带有一些缺省的行为，可以作为开发的起点。但是，除了某些特殊的情况之外，采用缺省行为的音频应用程序并不适合发行。您需要通过配置和使用音频会话来表达自己使用音频的意图，响应 OS 级别的音频变化。

举例来说，在使用缺省的音频会话时，如果出现 Auto-Lock 超时或屏幕锁定，应用程序的音频就会停止。如果您希望在屏幕被锁定时继续播放音频，则必须将下面的代码包含到应用程

序的[初始化](#)代码中：

```
[[AVAudioSession sharedInstance] setCategory: AVAudioSessionCategoryPlayback error: nil];  
[[AVAudioSession sharedInstance] setActive: YES error: nil];
```

[AVAudioSessionCategoryPlayback](#) 范畴确保音频的回放可以在屏幕锁定时继续。激活音频会话会使指定的范畴也被激活。范畴的详细信息请参见[音频会话编程指南](#)中的[音频会话范畴](#)部分。

如何处理呼入电话或时钟警告引起的中断取决于您使用的音频技术，如表7-2所示。

表7-2 处理音频中断音频技术	中断如何工作
系统声音服务	当中断开始时，系统声音和警告声音会变为无声。如果中断结束—当用户取消警告或选择忽略呼入电话时，会发生这种情况—它们就又自动变为可用。使用这种技术的应用程序无法影响声音中断的行为。
音频队列服务、OpenAL、I/O 音频单元	这些技术为中断的处理提供最大的灵活性。您需要编写一个中断监听回调函数，具体描述请参见 音频会话编程指南 中的“ 响应音频中断 ”部分。
AVAudioPlayer 类	AVAudioPlayer 类为中断的开始和结束提供了 委托 方法。根据实际的需要，您可以在 audioPlayerBeginInterruption:方法中更新用户界面，音频播放器对象会负责暂停回放。您也可以利用 audioPlayerEndInterruption:方法来重启音频的回放，并在必要时更新用户界面。音频播放器会负责重新激活您的音频会话。

每个 iPhone OS 应用程序—除了很少的例外—都应该采纳音频会话服务。如果需要了解具体的用法，请阅读[音频会话编程指南](#)。

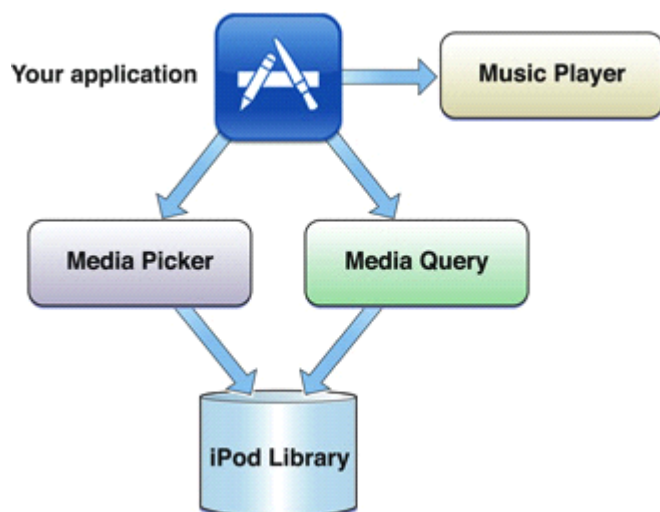
播放音频

本部分将介绍如何用 iPod 媒体库访问接口、系统声音服务、音频队列服务、AV Foundation 框架、和 OpenAL 来播放 iPhone OS 上的声音。

通过 iPod 媒体库访问接口播放媒体项

从 iPhone OS 3.0开始，iPod 媒体库访问接口使应用程序可以播放用户的歌曲、音频书，和音频播客。这个 API 的设计使基本回放变得非常简单，同时又支持高级的检索和回放控制。如图7-1所示，您的应用程序有两种方式可以取得媒体项，一种是通过媒体项选择器，如图左所示，它是个易于使用、预先封装好的视图控制器，其行为和内置 iPod 程序的音乐选择接口类似。对于很多应用程序，这种方式就够用了。如果媒体选择器没有提供您需要的某种访问控制，则可以使用媒体查询接口，该接口支持以基于断言（predicate）的方式指定 iPod 媒体库中的项目。

图7-1 使用 iPod 媒体库访问接口



如上图所示，位于右边的应用程序在取得媒体项之后，可以通过这个 API 提供的音乐播放器进行播放。

有关如何在应用程序中加入媒体项回放功能的完整解释，请参见 *iPod* 媒体库访问接口指南。

使用系统声音服务播放短声音及触发震动

当您需要播放用户界面声音效果（比如触击按键）或警告声音，或者使支持震动的设备产生震动时，可以使用系统声音服务。这个简洁接口的描述请参见[系统声音服务参考](#)。您可以在 [iPhone Dev Center](#) 中找到 *SysSound* 实例代码。

请注意：通过系统声音服务播放的声音不受音频会话配置的控制。因此，您无法使系统声音服务的音频行为 and 应用程序的其它音频行为保持一致。这也是需要避免使用系统声音服务播放音频的最重要原因，除非您有意为之。

[AudioServicesPlaySystemSound](#) 函数使您可以非常简单地播放短声音文件。使用上的简单也带来一些限制。您的声音文件必须是：

- 长度小于30秒

- 采用 PCM 或者 IMA4 (IMA/ADPCM) 格式

- 包装为.caf、.aif、或者.wav 文件

此外，当您使用 [AudioServicesPlaySystemSound](#) 函数时：

- 声音会以当前系统音量播放，且无法控制音量

- 声音立即被播放

- 不支持环绕和立体效果

[AudioServicesPlayAlertSound](#) 是一个类似的函数，用于播放一个短声音警告。如果用户在声音设置中将设备配置为震动，则这个函数在播放声音文件之外还会产生震动。

请注意：系统和用户界面的声音效果并不提供给您应用程序。举例来说，将 `kSystemSoundID_UserPreferredAlert` 常量作为参数传递给 [AudioServicesPlayAlertSound](#) 函数将不会播放任何声音。

在用 [AudioServicesPlaySystemSound](#) 或 [AudioServicesPlayAlertSound](#) 函数时，您需要首先创建一个声音 ID 对象，如程序清单7-1所示。

程序清单7-1 创建一个声音 ID 对象


```

        // Get the main bundle for the app
        CFBundleRef          mainBundle          =
CFBundleGetMainBundle ();

        // Get the URL to the sound file to play. The file in this
case
        // is "tap.aiff"
        soundFileURLRef          =
CFBundleCopyResourceURL (
                                mainBundle,
                                CFSTR ("tap"),
                                CFSTR ("aif"),
                                NULL
                                );

        // Create a system sound object representing the sound
file
        AudioServicesCreateSystemSoundID (
            soundFileURLRef,
            &soundFileObject
        );

```

然后再播放声音，如清单7-2所示。

程序清单7-2 播放一个系统声音

```

- (IBAction) playSystemSound {
    AudioServicesPlaySystemSound
(self.soundFileObject);
}

```

这片代码经常用于偶尔或者反复播放声音。如果您希望反复播放，就需要保持声音 ID 对象，直到应用程序退出。如果您确定声音只用一次——比如程序启动的声音——则可以在播放完成后立即销毁声音 ID，释放其占用的内存。

如果 iPhone OS 设备支持振动，则运行在该设备上的应用程序可以通过系统声音服务触发振动，振动的选项通过 `kSystemSoundID_Vibrate` 标识符来指定。`AudioServicesPlaySystemSound` 函数可以用于触发振动，具体如程序清单7-3所示。

程序清单7-3 触发振动

```

#import <AudioToolbox/AudioToolbox.h>
#import <UIKit/UIKit.h>
- (void) vibratePhone {
    AudioServicesPlaySystemSound
(kSystemSoundID_Vibrate);
}

```

如果您的应用程序运行在 iPod touch 上，则上面的代码不执行任何操作。

通过 AVAudioPlayer 类轻松播放声音

AVAudioPlayer 类提供了一个简单的 [Objective-C](#) 接口，用于播放声音。如果您的应用程序不需要立体声或精确同步，且不播放来自网络数据流的音频，则我们推荐您使用这个类来回放声音。

通过音频播放器可以实现如下任务：

播放任意长度的声音

播放文件或内存缓冲区中的声音

循环播放声音

同时播放多路声音（虽然不能精确同步）

控制每个正在播放声音的相对音量

跳到声音文件的特定点上，这可以为需要快进和反绕的应用程序提供支持

取得音频强度数据，用于测量音量

AVAudioPlayer 类可以播放 iPhone OS 上有的所有音频格式，具体描述请参见[“音频回放和录制格式”](#)部分。或者。如果您需要该类接口的完整描述，请参见 [AVAudioPlayer 类参考](#)。

为了使音频播放器播放音频，您需要为其分配一个声音文件，使其做好播放的准备，并为其指定一个委托对象。程序清单7-4中的代码通常放在应用程序控制器类的初始化方法中。

程序清单7-4 配置 AVAudioPlayer 对象

```
// in the corresponding .h file:
```

```
// @property (nonatomic, retain) AVAudioPlayer *player;
```

```
@synthesize player; // the player object
```

```
NSString *soundFilePath =
```

```
    [[NSBundle mainBundle] pathForResource:@"sound"
                                           ofType:
```

```
@"wav"];
```

```
NSURL *fileURL = [[NSURL alloc] initWithURLWithPath: soundFilePath];
```

```
AVAudioPlayer *newPlayer =
```

```
    [[AVAudioPlayer alloc] initWithContentsOfURL:
```

```
fileURL
```

```
error:
```

```
nil];
```

```
[fileURL release];
```

```
self.player = newPlayer;
```

```
[newPlayer release];
```

```
[player prepareToPlay];
```

```
[player setDelegate: self];
```

您可以通过[委托](#)对象（可能是您的[控制器对象](#)）来处理中断，以及在声音播放完成后更新用

户界面。有关 AVAudioPlayer 类的委托对象的具体描述请参见 [AVAudioPlayerDelegate 协议参考](#)。程序清单7-5显示了一个委托方法的简单实现，其中的代码在声音播放完成时更新了播放/暂停切换按键的标题。

程序清单7-5 实现 AVAudioPlayer 类的委托方法

```
- (void) audioPlayerDidFinishPlaying: (AVAudioPlayer *) player
    successfully: (BOOL) flag {
    if (flag == YES) {
        [self.button setTitle: @"Play" forState:
        UIControlStateNormal];
    }
}
```

调用回放控制方法可以使 AVAudioPlayer 对象执行播放、暂停、或者停止操作。您可以通过 `playing` 属性来检测当前是否正在播放。程序清单7-6显示了播放/暂停切换方法的基本实现，其功能是控制回放和更新 UIButton 对象的标题。

程序清单7-6 控制 AVAudioPlayer 对象

```
- (IBAction) playOrPause: (id) sender {

    // if already playing, then pause
    if (self.player.playing) {
        [self.button setTitle: @"Play" forState:
        UIControlStateHighlighted];
        [self.button setTitle: @"Play" forState: UIControlStateNormal];
        [self.player pause];

        // if stopped or paused, start playing
    } else {
        [self.button setTitle: @"Pause" forState:
        UIControlStateHighlighted];
        [self.button setTitle: @"Pause" forState:
        UIControlStateNormal];
        [self.player play];
    }
}
```

AVAudioPlayer 类使用 Objective-C 的[属性声明](#)来管理声音信息——比如取得声音时间线上的回放点和访问回放选项（如音量和是否重复播放的设置）。举例来说，您可以通过如下的代码设置一个音频播放器的回放音量：

```
[self.player setVolume: 1.0];    // available range is 0.0 through
1.0
```

有关 AVAudioPlayer 类的更多信息，请参见 [AVAudioPlayer 类参考](#)。

用音频队列服务播放和控制声音

音频队列服务（Audio Queue Services）加入了一些 AVAudioPlayer 类不具有的回放能力。通过音频队列服务进行回放可以：

精确计划声音的播放，支持声音的同步。

精确控制音量—基于一个个的缓冲区。

通过音频文件流服务（Audio File Stream Services）来播放从流中捕捉的音频。

音频队列服务可以播放 iPhone OS 支持的所有音频格式，具体描述请见[“音频回放和录制格式”](#)部分；还支持录制，详见[“录制音频”](#)部分。

有关如何使用这个技术的详细信息，请参见[音频队列服务编程指南](#)和[音频队列服务参考](#)。如果需要实例代码，请见 [iPhone Dev Center](#) 网站的 [SpeakHere](#) 实例（Mac OS X 系统上的实现则见 Core Audio SDK 的 AudioQueueTools 工程，在 Mac OS X 上安装 Xcode 工具之后，在 /Developer/Examples/CoreAudio/SimpleSDK/AudioQueueTools 路径下可以找到 AudioQueueTools 工程）。

创建一个音频队列对象

创建一个音频队列对象需要下面三个步骤：

创建管理音频队列所需的数据结构，比如您希望播放的音频格式。

定义管理音频队列缓冲区的回调函数。在回调函数中，您可以使用音频文件服务来读取希望播放的文件（在 iPhone OS 2.1 及更高版本中，您还可以用扩展音频文件服务来读取文件）。

1 通过 [AudioQueueNewOutput](#) 函数实例化回放音频队列。

程序清单 7-7 是上述步骤的 ANSI C 代码。[SpeakHere](#) 示例工程中也有同样的步骤，只是它们位于 [Objective-C](#) 程序的上下文中。

程序清单 7-7 创建一个音频队列对象

```
static const int kNumberBuffers = 3;
// Create a data structure to manage information needed by the audio queue
struct myAQStruct {
    AudioFileID                mAudioFile;
    CAStreamBasicDescription    mDataFormat;
    AudioQueueRef               mQueue;
    AudioQueueBufferRef         mBuffers[kNumberBuffers];
    SInt64                      mCurrentPacket;
    UInt32                     mNumPacketsToRead;
    AudioStreamPacketDescription *mPacketDescs;
    bool                        mDone;
};
// Define a playback audio queue callback function
static void AQTestBufferCallback(
    void                *inUserData,
    AudioQueueRef        inAQ,
    AudioQueueBufferRef  inCompleteAQBuffer
) {
    myAQStruct *myInfo = (myAQStruct *)inUserData;
```

```

if (myInfo->mDone) return;
UInt32 numBytes;
UInt32 nPackets = myInfo->mNumPacketsToRead;

AudioFileReadPackets (
    myInfo->mAudioFile,
    false,
    &numBytes,
    myInfo->mPacketDescs,
    myInfo->mCurrentPacket,
    &nPackets,
    inCompleteAQBuffer->mAudioData
);
if (nPackets > 0) {
    inCompleteAQBuffer->mAudioDataByteSize = numBytes;
    AudioQueueEnqueueBuffer (
        inAQ,
        inCompleteAQBuffer,
        (myInfo->mPacketDescs ? nPackets : 0),
        myInfo->mPacketDescs
    );
    myInfo->mCurrentPacket += nPackets;
} else {
    AudioQueueStop (
        myInfo->mQueue,
        false
    );
    myInfo->mDone = true;
}
}
// Instantiate an audio queue object
AudioQueueNewOutput (
    &myInfo.mDataFormat,
    AQTestBufferCallback,
    &myInfo,
    CFRRunLoopGetCurrent(),
    kCFRunLoopCommonModes,
    0,
    &myInfo.mQueue
);

```

控制回放音量

音频队列对象为您提供两种控制回放音量的方法。

您可以通过调用 [AudioQueueSetParameter](#) 函数并传入 kAudioQueueParam_Volume 参数来直接设置回放的音量，如程序清单7-8所示，音量的变化会立即生效。

程序清单7-8 直接设置回放的音量

```
Float32 volume = 1;    // linear scale, range from 0.0 through
1.0
AudioQueueSetParameter (
    myAQstruct.audioQueueObject,
    kAudioQueueParam_Volume,
    volume
);
```

您还可以通过 [AudioQueueEnqueueBufferWithParameters](#) 函数来设置音频队列缓冲区的回放音量。这个函数可以指定音频队列缓冲区进入队列时携带的音频队列设置。通过这个函数做出的改变在音频队列缓冲区开始播放的时候生效。

在上述的两种情况下，对音频队列的音量所做的修改都会一直保持下来，直到再次被改变。

指示回放音量

您可以通过下面的方式得到音频队列对象的当前回放音量：

启用音频队列对象的音量计，具体方法是将其 kAudioQueueProperty_EnableLevelMetering 属性设置为 true。

查询音频队列对象的 kAudioQueueProperty_CurrentLevelMeter 属性。

这个属性的值是一个 AudioQueueLevelMeterState 结构的数组，每个声道都有一个相对应的结构。程序清单7-9显示了这个结构的内容：

程序清单7-9 AudioQueueLevelMeterState 结构

```
typedef struct AudioQueueLevelMeterState {
    Float32    mAveragePower;
    Float32    mPeakPower;
}; AudioQueueLevelMeterState;
```

同时播放多路声音

为了同时播放多路声音，需要为每路声音创建一个回放音频队列对象，并对每个音频队列调用 [AudioQueueEnqueueBufferWithParameters](#) 函数，将第一个音频缓冲区排入队列，使之开始播放。

在基于 iPhone OS 的设备中同时播放声音时，音频格式是很关键的。如果要同时播放，您需要使用线性 PCM (无压缩) 音频格式或特定的有压缩音频格式，具体描述请参见[“音频回放和录制格式”](#)部分。

使用 OpenAL 播放和定位声音

开源的 OpenAL 音频 API 位于 iPhone OS 系统的 OpenAL 框架中，它提供了一个优化接口，用于定位正在回放的立体声场中的声音。使用 OpenAL 进行声音的播放、定位、和移动是很简单的——其工作方式和其它平台一样。此外，OpenAL 还可以进行混音。OpenAL 使用 Core Audio 的 I/O 单元进行回放，从而使延迟最低。

由于所有的这些原因，OpenAL 是 iPhone OS 设备中游戏程序的最好选择。当然，OpenAL 也是一般的 iPhone OS 应用程序进行音频播放的良好选择。

iPhone OS 对 OpenAL 1.1 的支持是构建在 Core Audio 之上的。更多的信息请参见 [iPhone OS 系统的 OpenAL FAQ](#)。如果需要有关 OpenAL 的文档，请参见 <http://openal.org> 的 OpenAL 网站；如果需要演示如何播放 OpenAL 音频的示例程序，请参见 [oalTouch](#)。

录制音频

在 iPhone OS 系统上，可以通过 [AVAudioRecorder](#) 类和音频队列服务来进行音频录制，而 Core Audio 则为其提供底层的支持。这些接口所做的工作包括连接音频硬件、管理内存、以及在需要时使用编解码器。您可以录制“[音频的回放和录制格式](#)”部分列出的所有格式的音频。

本部分将介绍如何通过 AVAudioRecorder 类和音频队列服务在 iPhone OS 系统上录制音频。

通过 AVAudioRecorder 类进行录制

iPhone OS 上最简单的录音方法是使用 AVAudioRecorder 类，类的具体描述请参见 [AVAudioRecorder 类参考](#)。该类提供了一个高度精简的 Objective-C 接口。通过这个接口，您可以轻松实现诸如暂停/重启录音这样的功能，以及处理音频中断。同时，您还可以对录制格式保持完全的控制。

进行录制时，您需要提供一个声音文件的 URL、建立音频会话、以及配置录音对象。进行这些准备工作的一个良好时机就是应用程序启动的时候，如程序清单 7-10 所示。诸如 soundFilePath 和 recording 这样的变量都在类接口文件中进行声明。

程序清单 7-10 建立音频会话和声音文件的 URL

```
- (void) viewDidLoad {  
  
    [super viewDidLoad];  
  
    NSString *tempDir = NSTemporaryDirectory ();  
    NSString *soundFilePath = [tempDir stringByAppendingString:  
@"sound.caf"];  
  
    NSURL *newURL = [[NSURL alloc] initWithFileURLWithPath: soundFilePath];  
    self.soundFileURL = newURL;  
    [newURL release];  
}
```

```

    AVAudioSession *audioSession = [AVAudioSession sharedInstance];
    audioSession.delegate = self;
    [audioSession setActive: YES error: nil];

    recording = NO;
    playing = NO;
}

```

您需要在接口声明中加入 AVAudioSessionDelegate、AVAudioRecorderDelegate、AVAudioPlayerDelegate（如果同时支持声音回放的话）协议。然后，就可以实现如程序清单7-11所示的录制方法。

程序清单7-11 一个基于 AVAudioRecorder 类的录制/停止方法

```

-(IBAction) recordOrStop: (id) sender {

    if (recording) {

        [soundRecorder stop];
        recording = NO;
        self.soundRecorder = nil;

        [recordOrStopButton setTitle: @"Record" forState: UIControlStateNormal];
        [recordOrStopButton setTitle: @"Record" forState: UIControlStateHighlighted];

        [[AVAudioSession sharedInstance] setActive: NO error: nil];

    } else {

        [[AVAudioSession sharedInstance] setCategory: AVAudioSessionCategoryRecord error:
nil];

        NSDictionary *recordSettings =
            [[NSDictionary alloc] initWithObjectsAndKeys:
                [NSNumber numberWithInt: 44100.0],
AVSampleRateKey,
                [NSNumber numberWithInt: kAudioFormatAppleLossless], AVFormatIDKey,
                [NSNumber numberWithInt: 1],
AVNumberOfChannelsKey,
                [NSNumber numberWithInt: AVAudioQualityMax],
AVEncoderAudioQualityKey,
                nil];

        AVAudioRecorder *newRecorder = [[AVAudioRecorder alloc] initWithURL:
soundFileURL
                                settings:

```

```

recordSettings

error: nil];

[recordSettings release];
self.soundRecorder = newRecorder;
[newRecorder release];

soundRecorder.delegate = self;
[soundRecorder prepareToRecord];
[soundRecorder record];
[recordOrStopButton setTitle: @"Stop" forState: UIControlStateNormal];
[recordOrStopButton setTitle: @"Stop" forState: UIControlStateHighlighted];

recording = YES;
}
}

```

有关 AVAudioRecorder 类的更多信息，请参见 [AVAudioRecorder 类参考](#)。

用音频队列服务进行录制

用音频队列服务进行录制时，您的应用程序需要配置音频会话、实例化一个录音音频队列对象，并为其提供一个回调函数。回调函数负责将音频数据存入内存以备随时使用，或者写入文件进行长期存储。

声音的录制发生在 iPhone OS 的系统定义级别（system-defined level）。系统会从用户选择的音频源取得输入——比如内置的麦克风、耳机麦克风（如果连接到 iPhone 上的话）、或者其它输入源。

和声音的回放一样，您可以通过查询音频队列对象的 `kAudioQueueProperty_CurrentLevelMeter` 属性来取得当前的录制音量，具体描述请见“[指示回放音量](#)”部分。

有关如何通过音频队列服务录制音频的详细实例，请参见 [音频队列服务编程指南](#)的[录制音频](#)部分，实例代码则请见 [iPhone Dev Center](#) 网站上的 [SpeakHere](#)。

解析音频流

为了播放音频流内容，比如来自网络连接的音频流，可以结合使用音频文件流服务和音频队列服务。音频文件流服务负责从常见的、采用网络位流格式的音频文件容器中解析出音频数据和元数据。您也可以用它来解析磁盘文件中的数据包和元数据。

iPhone OS 可以解析的音频文件和位流格式和 Mac OS X 相同，具体如下：

- MPEG-1 Audio Layer 3，用于.mp3文件
- MPEG-2 ADTS，用于.aac 音频数据格式
- AIFC
- AIFF
- CAF
- MPEG-4，用于.m4a、.mp4、和.3gp 文件

NeXT
WAVE

在取得音频数据包之后，您就可以以任何 iPhone OS 系统支持的格式进行播放，这些格式在[“音频回放和录制格式”](#)部分中列出。

为了获得最好的性能，处理网络音频流的应用程序应该仅使用来自 Wi-Fi 连接的数据。您可以通过 iPhone OS 提供的 System Configuration [框架](#)及其 SCNetworkReachability.h 头文件定义的接口来确定什么网络是可到达和可用的。如果需要实例代码，请参见 [iPhone Dev Center](#) 网站的 [Reachability](#) 工程。

为了连接网络音频流，可以使用 iPhone OS 系统中的 Core Foundation 框架中的接口，比如 CFHTTPMessage 接口，具体描述请见 [CFHTTPMessage 参考](#)。通过音频文件流服务解析网络数据包，将它恢复为音频数据包，然后放入缓冲区，发送给负责回放的音频队列对象。音频文件流服务依赖于音频文件服务定义的接口，比如 AudioFramePacketTranslation 结构和 AudioFilePacketTableInfo 结构，具体描述请见 [音频文件服务参考](#)。

有关如何使用流的更多信息，请参见 [音频文件流服务参考](#)。实例代码则请参见位于 <Xcode>/Examples/CoreAudio/Services/ 目录下的 AudioFileStream 例子工程，其中 <Xcode> 是开发工具所在的目录。

iPhone OS 系统上的音频单元支持

iPhone OS 提供一组音频插件，称为音频单元，可以用于所有的应用程序。您可以通过 Audio Unit [框架](#)提供的接口来打开、连接、和使用音频单元；还可以定义定制的音频单元，在自己的应用程序内部使用。由于应用程序必须静态连接定制的音频单元，所以 iPhone OS 系统上的其它应用程序不能使用您开发的音频单元。

表7-3列出了 iPhone OS 提供的音频单元。

表7-3 系统提

供的音频单元

描述

音频单元

转换器单元

转换器单元，类型为 kAudioUnitSubType_AUConverter，用于音频数据的格式转换。

iPod 均衡器单元

iPod EQ 单元，类型为 kAudioUnitSubType_AUiPodEQ，提供一个简单的、基于预设的均衡器，可以在应用程序中使用。

3D 混音器单元

3D 混音器单元，类型为 kAudioUnitSubType_AU3DMixerEmbedded，用于混合多个音频流，指定立体声输出移动，操作采样率，等等。

多通道混音器单元

多通道混音器单元，类型为 kAudioUnitSubType_MultiChannelMixer，用于将多个音频流混合成为单一的音频流。

一般输出单元

一般输出单元，类型为 kAudioUnitSubType_GenericOutput，支持和线性 PCM 格式互相转换，可以用于开始或结束一个音频单元图。

I/O 单元

I/O 单元，类型为 kAudioUnitSubType_RemoteIO，用于连接音频输入和输入硬件，支持实时 I/O。如何使用音频单元的实例代码请见 [aurioTouch](#) 工程。

语音处理 I/O 单元

语音处理 I/O 单元，类型为 kAudioUnitSubType_VoiceProcessingIO，具有 I/O 单元的特征，同时为了支持双向交流，加入了回响抑制功能。

有关系统音频单元的更多信息，请参见[系统音频单元访问指南](#)。

iPhone 音频的最佳实践

操作音频的贴士

在操作 iPhone OS 系统上的音频内容时，您需要记住表7-4列出的基本贴士。

表7-4 音频贴士贴士	动作
正确地使用压缩音频	对于 AAC、MP3、和 ALAC (Apple Lossless) 音频，解码过程是由硬件来完成的，虽然比较有效，但同时只能解码一个音频流。如果您需要同时播放多路声音，请使用 IMA4 (压缩) 或者线性 PCM (无压缩) 格式来存储那些文件。
将音频转换为您需要的数据格式和文件格式	Mac OS X 的 afconvert 工具可以进行很多数据格式和文件类型的转换。请参见 “iPhone OS 偏好的音频格式” 部分和 afconvert 工具的手册页面。
评价音频的内存使用问题	当您使用音频队列服务播放音频时，需要编写一个回调函数，负责将较短的音频数据片断发送到音频队列的缓冲区。在某些情况下，将整个音频文件载入内存是最佳的选择，这样可以使播放时的磁盘访问尽最少；而在另外一些情况下，最好的方法则是每次只载入足够填满缓冲区的数据。请测试和评价哪种策略对您的应用程序最好。
限制音频的采样率和位深度，减少音频文件的尺寸	采样率和每个样本的位深度对无压缩音频的尺寸有直接的影响。如果您需要播放很多这样的声音，则应该考虑降低这些指标，以减少音频数据的内存开销。举例来说，相对于使用采样率为44.1 kHz 的音频作为声音效果，您可以使用采样率为32 kHz（或可能更低）的音频，仍然可以得到很合理的品质。
选择恰当的技术	使用 Core Audio 的系统声音服务来播放警告和用户界面声音效果。当您希望使用便利的高级接口来定位立体声场中的声音，或者要求很低的回放延迟时，则应该使用 OpenAL。如果需要从文件或网络数据流中解析出音频数据，可以使用音频文件服务接口。如果只是简单回放一路或多路声音，则应该使用 AVAudioPlayer 类。对于具有其它音频功能的应用程序，包括音频流的回放和音频录制，可以使用音频队列服务。
低延迟编码	如果需要尽可能低的回放延迟，可以使用 OpenAL，或者直接使用 I/O 单元。

iPhone OS 偏好的音频格式

对于无压缩（最高品质）音频，请使用封装在 CAF 文件中的、16位、低位在前（little endian）的线性 PCM 音频数据。您可以用 Mac OS X 的 afconvert 命令行工具来将音频文件转换为上述格式：

```
/usr/bin/afconvert -f caff -d LEI16 {INPUT}
```

{OUTPUT}

afconvert 工具可以进行广泛的音频数据格式和文件类型转换。您可以通过 afconvert 的手册页面，以及在 shell 提示符下键入 afconvert -h 命令获取更多信息。

对于压缩音频，当每次只需播放一个声音，或者当不需要和 iPod 同时播放音频时，适合使用 AAC 格式的 CAF 或 m4a 文件。

当您需要在同时播放多路声音时减少内存开销时，请使用 IMA4 (IMA/ADPCM) 压缩格式，这样可以减少文件尺寸，同时在解压缩过程中对 CPU 的影响又最小。和线性 PCM 数据一样，请将 IMA4 数据封装在 CAF 文件中。

在 iPhone OS 使用视频

录制视频

从 iPhone OS 3.0 开始，您可以在具有录制支持的设备上录制视频，包括当时的音频。显示视频录制界面的方法是创建和推出一个 [UIImagePickerController](#) 对象，和显示静态图片照相机界面完全一样。

在录制视频时，您必须首先检查是否存在照相机源类型 ([UIImagePickerControllerSourceTypeCamera](#))，以及照相机是否支持电影媒体类型 (kUTTypeMovie)。根据您为 mediaTypes 属性分配的媒体类型的不同，选择器对象可以直接显示静态图像照相机，或者视频摄像机，还可以显示一个选择界面，让用户选择。

使用 [UIImagePickerControllerDelegate](#) 协议，注册为图像选择器的委托。在视频录制完成时，您的委托对象的 `imagePickerController:didFinishPickingMediaWithInfo:` 方法会备调用。

对于支持录制的设备，您也可以从用户照片库中选择之前录制的视频。

有关如何使用图像选择器的更多信息，请参见 [UIImagePickerController 类参考](#)。

播放视频文件

在 iPhone OS 系统上，应用程序可以通过 Media Player 框架 (MediaPlayer.framework) 来播放视频文件。视频的回放只支持全屏模式，需要播放场景切换动画的游戏开发者或需要播放媒体文件的其它开发者可以使用。当应用程序开始播放视频时，媒体播放器界面就会接管，将屏幕渐变为黑色，然后渐渐显示视频内容。视频播放界面上可以显示或者不显示调整回放的控件。您可以通过部分或全部激活这些控件（如图7-2所示），使用户可以改变音量、改变回放点、开始或停止视频的播放。如果禁用所有的控件，视频会一直播放，直到结束。

图7-2 带有播放控制的媒体播放器界面



在开始播放前，您必须知道希望播放的 URL。对于应用程序提供的文件，这个 URL 通常是指向应用程序包中某个文件的指针；但是，它也可以是指向远程服务器文件的指针。您可以用这个 URL 来[实例化](#)一个新的 `MPMoviePlayerController` 类的实例。这个类负责视频文件的回放和管理用户交互，比如响应用户对播放控制（如果显示的话）的触击动作。简单调用控制器的 [play](#) 方法，就可以开始播放了。

程序清单 7-12 显示一个实例方法，功能是播放位于指定 URL 的视频。`play` 方法是异步的调用，在电影播放时会把控制权返回给调用者。电影控制器负责将电影载入一个全屏的视图，并通过动画效果将电影放到应用程序现有内容的上方。在视频回放完成后，电影控制器会向委托对象发出一个[通告](#)，该委托对象负责在不再需要时释放电影控制器。

程序清单 7-12 播放全屏电影

```
-(void)playMovieAtURL:(NSURL*)theURL
{
    MPMoviePlayerController*    theMovie    =    [[MPMoviePlayerController    alloc]
initWithContentURL:theURL];

    theMovie.scalingMode = MPMovieScalingModeAspectFill;
    theMovie.movieControlMode = MPMovieControlModeHidden;

    // Register for the playback finished notification.
    [[NSNotificationCenter defaultCenter] addObserver:self
        selector:@selector(myMovieFinishedCallback:)
        name:MPMoviePlayerPlaybackDidFinishNotification
        object:theMovie];

    // Movie playback is asynchronous, so this method returns immediately.
    [theMovie play];
}

// When the movie is done, release the controller.
-(void)myMovieFinishedCallback:(NSNotification*)aNotification
{
    MPMoviePlayerController* theMovie = [aNotification object];

    [[NSNotificationCenter defaultCenter] removeObserver:self
        name:MPMoviePlayerPlaybackDidFinishNotification
```

```

        object:theMovie];

    // Release the movie instance created in playMovieAtURL:
    [theMovie release];
}

```

有关 Media Player 框架的各个类的更多信息，请参见 [Media Player 框架参考](#)。有关它支持的视频格式列表，请参见 [iPhone OS 技术概览](#)。

设备支持

iPhone OS 支持很多使移动计算的用户体验更具吸引力的特性。通过 iPhone OS，应用程序可以访问诸如加速计和照相机这样的硬件特性，也可以访问像用户照片库这样的软件特性。本文的下面部分将描述这些特性，并向您展示如何将它们集成到您的应用程序中。

确定硬件支持是否存在

为 iPhone OS 设计的应用程序必须能够运行在具有不同硬件特性的多种设备上。虽然像加速计和 Wi-Fi 连网这样的特性在所有设备上都是支持的，但是一些设备不包含照相机或 GPS 硬件。如果您的应用程序要求设备具有这样的特性，应该在用户购买之前通知他们。对于那些不是必需、但如果存在就希望支持的特性，则必须在试图使用之前检测它们是否存在。

重要提示：如果应用程序运行的前提是某个特性一定要存在，则应该在应用程序的 Info.plist 文件中对 UIRequiredDeviceCapabilities 键进行相应的设置，以避免将需要某种特性的应用程序安装在不具有该特性的设备上。但是，如果您的应用程序在给定特性存在或不存在时都可以运行，则不应该包含这个键。更多有关如果配置该键的信息，请参见“[信息属性列表](#)”部分。表8-1列出了确定某种硬件是否存在的方法。如果您的应用程序在缺少某个特性时可以工作，而在该特性存在时又可以加以利用，则应该使用这些技术。

表8-1 识别可用的硬件特性特性

选项	
确定网络是否存在...	使用 Software Configuration 框架的可达性（reachability）接口检测当前的网络连接。有关如何使用 Software Configuration 框架的例子请参见 可达性 部分。
确定静态照相机是否存在...	使用 UIImagePickerControllerController 类的 isSourceTypeAvailable: 方法来确定照相机是否存在。更多信息请参见“ 使用照相机进行照相 ”部分。
确定音频输入（麦克风）是否存在...	在 iPhone OS 3.0 及之后的系统上，可以用 AVAudioSession 类来确定音频输入是否存在。该类考虑了 iPhone OS 设备上的很多不同的音频输入设备，包括内置的麦克风、耳机插座、和连接的配件。更多信息请参见 AVAudioSession 类参考 部分。
确定 GPS 硬件是否存在...	在配置 CLLocationManager 对象、使应用程序可以获取位置变化时，指定高精度级别。Core Location 框架 并不指定硬件是否存在的直接信息，而是使用精度值来提供您所需要的数据。如果一系列位置事件报告的精度都不够高，您可以通知用户。更多信息请参见“ 获取用户的当前位置 ”部分。

确定特定的配件是否存在 使用 External Accessory 框架的类来寻找合适的附近对象，并在... 行连接。更多信息请参见[“和配件进行通讯”](#)部分。

和配件进行通讯

在 iPhone OS 3.0及之后的系统上，External Accessory [框架](#)（ExternalAccessory.framework）提供了一种管道机制，使应用程序可以和 iPhone 或 iPod touch 设备的配件进行通讯。通过这种管道，应用程序开发者可以将配件级别的功能集成到自己的程序中。

请注意：下面部分将向您展示 iPhone 应用程序如何连接配件。如果您有兴趣成为 iPhone 或 iPod touch 配件的开发者，可以在 <http://developer.apple.com> 网站上找到相应的信息。

为了使用 External Accessory 框架的接口，您必须将 ExternalAccessory.framework 加入到 Xcode 工程，并连接到相应的目标中。此外，还需要在相应的源代码文件的顶部包含一个 `#import <ExternalAccessory/ExternalAccessory.h>` 语句，才能访问该框架的类和头文件。有关如何为工程添加框架的更多信息，请参见 [Xcode 工程管理指南](#) 中的 [工程中的文件](#) 部分；有关 External Accessory 框架中类的一般信息，请参见 [External Accessory 框架参考](#)。

配件的基础

在和配件进行通讯之前，需要与配件的制造商紧密合作，理解配件提供的服务。制造商必须在配件的硬件中加入显式的支持，才能和 iPhone OS 进行通讯。作为这种支持的一部分，配件必须支持至少一种命令[协议](#)，也就是支持一种定制的通讯模式，使配件和应用程序之间可以进行数据传输。苹果并不维护一个协议的注册表，支持何种协议及是否使用其他制造商支持的定制或标准协议是由制造商自行决定的。

作为和配件制造商通讯的一部分，您必须找出给定的配件支持什么协议。为了避免名字空间发生冲突，协议的名称由反向的 DNS 字符串来指定，形式是 `com.apple.myProtocol`。这使得每个配件制造商都可以根据自己的需要定义协议，以支持不同的配件产品线。

应用程序通过打开一个使用指定协议的会话来和配件进行通讯。打开会话的方法是创建一个 [EASession](#) 类的实例，该类中包含 [NSInputStream](#) 和 [NSOutputStream](#) 对象，可以和配件进行通讯。通过这些流对象，应用程序可以向配件发送未经加工的数据包，以及接收来自配件的类似数据包。因此，您必须按照期望的协议来理解每个数据包的格式。

声明应用程序支持的协议

能够和配件通讯的应用程序应该在其 Info.plist 文件中声明支持的协议，使系统知道在相应的配件接入时，该应用程序可以被启动。如果当前没有应用程序可以支持接入的配件，系统可以选择启动 App Store 并指向支持该设备的应用程序。

为了声明支持的协议，您必须在应用程序的 Info.plist 文件中包含 `UISupportedExternalAccessoryProtocols` 键。该键包含一个字符串[数组](#)，用于标识应用程序支持的通讯协议。您的应用程序可以在这个列表中以任意顺序包含任意数量的协议。系统并不使用这个列表来确定应用程序应该选择哪个协议，而只是用它来确定应用程序是否能够和相应的配件进行通讯。您的代码需要在开始和配件进行对话时选择适当的通讯协议。

在运行时连接配件

在配件接入系统并做好通讯准备之前，通过 External Accessory 框架无法看到配件。当配件变为可见时，您的应用程序就可以获取相应的配件对象，然后用其支持的一或多个协议打开会话。

共享的 [EAAccessoryManager](#) 对象为应用程序寻找与之通讯的配件提供主入口点。该类包含一个已经接入的配件对象的数组，您可以对其进行[枚举](#)，看看是否存在应用程序支持的配件。[EAAccessory](#) 对象中的绝大多数信息（比如名称、制造商、和型号信息）都只是用于显示。如果您要确定应用程序是否可以连接一个配件，必须看配件的协议，确认应用程序是否支持其中的某个协议。

请注意：多个配件对象支持同一协议是可能的。如果发生这种情况，您的代码必须负责选择使用哪个配件对象。

对于给定的配件对象，每次只能有一个指定协议的会话。EAAccessory 对象的 [protocolStrings](#) 属性包含一个[字典](#)，字典的键是配件支持的协议。如果您试图用一个已经在使用的协议创建会话，External Accessory 框架就会产生错误。

程序清单8-1展示了如何检查接入配件的列表并从中取得应用程序支持的第一个配件。它为指定的协议创建一个会话，并对会话的输入和输出流进行配置。在这个方法返回会话对象时，已经完成和配件的连接，并可以开始发送和接收数据了。

程序清单8-1 创建和配件的通讯会话

```
- (EASession *)openSessionForProtocol:(NSString *)protocolString
{
    NSArray *accessories = [[EAAccessoryManager sharedAccessoryManager]
                             connectedAccessories];

    EAAccessory *accessory = nil;
    EASession *session = nil;

    for (EAAccessory *obj in accessories)
    {
        if ([[obj protocolStrings] containsObject:protocolString])
        {
            accessory = obj;
            break;
        }
    }

    if (accessory)
    {
        session = [[EASession alloc] initWithAccessory:accessory
                                                  forProtocol:protocolString];

        if (session)
        {
            [[session inputStream] setDelegate:self];
            [[session inputStream] scheduleInRunLoop:[NSRunLoop
currentRunLoop]
```

```

forMode:NSDefaultRunLoopMode];
    [[session inputStream] open];
    [[session outputStream] setDelegate:self];
    [[session outputStream] scheduleInRunLoop:[NSRunLoop
currentRunLoop]

forMode:NSDefaultRunLoopMode];
    [[session outputStream] open];
    [session autorelease];
    }
}

return session;
}

```

在配置好输入输出流之后，最好一步就是处理和流相关的数据了。程序清单8-2展示了在[委托](#)方法中处理流事件的基本代码结构。清单中的方法可以响应来自配件输入输出流的事件。当配件向应用程序发送数据时，事件发生表示有数据可供读取；类似地，当配件准备好接收应用程序数据时，也通过事件来表示（当然，您并不一定要等到这个事件发生才向流写出数据，应用程序也可以调用流的[hasBytesAvailable](#)方法来确认配件是否还能够接收数据）。有关流及如何处理流事件的更多信息，请参见[Cocoa 流编程指南](#)。

程序清单8-2 处理流事件

// Handle communications from the streams.

```

- (void)stream:(NSSStream*)theStream handleEvent:(NSStreamEvent)streamEvent
{
    switch (streamEvent)
    {
        case NSStreamHasBytesAvailable:
            // Process the incoming stream data.
            break;

        case NSStreamEventHasSpaceAvailable:
            // Send the next queued command.
            break;

        default:
            break;
    }
}

```

监控与配件有关的事件

当配件接入或断开时，External Accessory 框架都可以发送[通告](#)。但是这些通告并不自动发送，如果您的应用程序感兴趣，必须调用 EAAccessoryManager 类的 [registerForLocalNotifications](#) 方法来显式请求。当配件接入、认证、并准备好和应用程序进行交互时，框架可以发出一个 [EAAccessoryDidConnectNotification](#) 通告；而当配件断开时，框架则可以发送一个 [EAAccessoryDidDisconnectNotification](#) 通告。您可以通过缺省的 [NSNotificationCenter](#) 来注册接收这些通告。两种通告都包含受影响的配件的信息。

除了通过缺省的通告中心接收通告之外，当前正在和配件进行交互的应用程序可以为相应的 EAAccessory 对象分配一个[委托](#)，使它在发生变化时得到通知。委托对象必须遵循 [EAAccessoryDelegate](#) 协议，该协议目前包含名为 [accessoryDidDisconnect](#) 的可选方法，您可以通过这个方法接收配件断开通告，而不需要事先配置通告观察者。

有关如何注册接收通告的更多信息，请参见 [Cocoa 通告编程主题](#)。

访问加速计事件

加速计以时间为轴，测量速度沿着给定线性路径发生的变化。每个 iPhone 和 iPod touch 都包含三个加速计，分别负责设备的三个轴向。这种加速计的组合使得我们可以检测设备在任意方向上的运动。您可以用这些数据来跟踪设备突然发生的运动，以及当前相对于重力的方向。

请注意：在 iPhone OS 3.0 及之后的系统，如果您希望检测特定类型的运动，比如摇摆设备，应该考虑通过运动事件来进行，而不是使用加速计的接口。运动事件为检测特定类型的加速计运动提供一致的接口，更多的细节请参见[“运动事件”](#)部分。

每个应用程序都可以通过 [UIAccelerometer](#) 的[单件](#)对象来接收加速计数据。您可以通过 UIAccelerometer 的 [sharedAccelerometer](#) 类方法来取得该类的实例。之后，您就可以设置加速计数据更新的间隔时间及负责取得数据的自定义[委托](#)。数据更新的间隔时间的最小值是 10 毫秒，对应于 100Hz 的刷新频率。对于大多数应用程序来说，可以使用更大的时间间隔。您一旦设置了委托对象，加速计就会开始发送数据。而委托对象也会在您请求的时间间隔之后收到数据。

程序清单 8-3 展示了配置加速计的基本步骤。在这个例子中，更新频率设置为 50Hz，对应于 20 毫秒的时间间隔。myDelegateObject 是您定义的定制对象，必须支持 [UIAccelerometerDelegate](#) 协议，该协议定义了接收加速计数据的方法。

程序清单 8-3 配置加速计

```
#define kAccelerometerFrequency      50 //Hz
-(void)configureAccelerometer
{
    UIAccelerometer*      theAccelerometer    =    [UIAccelerometer
sharedAccelerometer];
    theAccelerometer.updateInterval = 1 / kAccelerometerFrequency;

    theAccelerometer.delegate = self;
    // Delegate events begin immediately.
}
```

全局共享的加速计会以固定频率调用委托对象的 [accelerometer:didAccelerate:](#) 方法，通过它传

送事件数据，如清单8-4所示。在这个方法中，您可以根据自己的需要处理加速计数据。一般地说，我们推荐您使用一些过滤器来分离您感兴趣的数据成分。

程序清单8-4 接收加速计事件

```
- (void)accelerometer:(UIAccelerometer *)accelerometer didAccelerate:(UIAcceleration *)acceleration
{
    UIAccelerationValue x, y, z;
    x = acceleration.x;
    y = acceleration.y;
    z = acceleration.z;

    // Do something with the values.
}
```

将全局共享的 `UIAccelerometer` 对象的委托设置为 `nil`，就可以停止加速计事件的递送。将委托对象设置为 `nil` 的操作会向系统发出通知，使其在需要的时候关闭加速计硬件，从而节省电池的寿命。

在委托方法中收到的加速计数据代表的是来自加速计硬件的实时数据。即使设备完全处于休息状态，加速计硬件报告的数据也可能产生轻微的波动。使用这些数据时，务必通过取平均值或对收到的数据进行调整的方法，来平抑这种波动。作为例子，`Bubble Level` 示例程序提供了一些控制，可以根据已知的表面调整当前的角度，后续读取的数据则是相对于调整后的角度进行调整。如果您的代码需要类似级别的精度，也应该在程序界面中包含一些调整的选项。

选择恰当的更新频率

在配置加速计事件的更新频率时，最好既能满足应用程序的需求，又能使事件发送次数最少。需要系统以每秒100次的频率发送加速计事件的应用程序是很少的。使用较低的频率可以避免应用程序过于繁忙，从而提高电池的寿命。表8-2列出了一些典型的更新频率，以及在该频率下产生的加速计数据适合哪些应用场合。

常 用的加速计 事件更新频 率事件频率 (Hz)	用途
10-20	适合用于确定代表设备当前方向的向量。
30-60	适合用于游戏和使用加速计进行实时输入的应用程序。
70-100	适合用于需要检测设备高频运动的应用程序，比如检测用户快速触击或摆动设备。

从加速计数据中分离重力成分

如果您希望通过加速计数据来检测设备的当前方向，就需要将数据中源于重力的部分从源于设备运动的部分中分离开来。为此，您可以使用低通滤波器来减少加速计数据中剧烈变化部

分的权重，这样过滤之后的数据更能反映由重力产生的较为稳定的因素。

程序清单8-5展示了一个低通滤波器的简化版本。清单中的代码使用一个低通滤波因子生成一个由当前的滤波前数据的10%和前一个滤波后数据的90%组成的值。前一个加速计数值存储在类的 `accelX`、`accelY`、和 `accelZ` 成员变量中。由于加速计数据以固定的频率进入您的应用程序，所以这些数值会很快稳定下来，但过滤后的数据对突然而短暂的运动响应缓慢。

程序清单8-5 从加速计数据中分离出重力的效果

```
#define kFilteringFactor 0.1
```

```
- (void)accelerometer:(UIAccelerometer *)accelerometer didAccelerate:(UIAcceleration *)acceleration {  
    // Use a basic low-pass filter to keep only the gravity component of each axis.  
    accelX = (acceleration.x * kFilteringFactor) + (accelX * (1.0 - kFilteringFactor));  
    accelY = (acceleration.y * kFilteringFactor) + (accelY * (1.0 - kFilteringFactor));  
    accelZ = (acceleration.z * kFilteringFactor) + (accelZ * (1.0 - kFilteringFactor));  
  
    // Use the acceleration data.  
}
```

从加速计数据中分离实时运动成分

如果您希望通过加速计数据检测设备的实时运动，则需要将突然发生的运动变化从稳定的重力效果中分离出来。您可以通过高通滤波器来实现这个目的。

程序清单8-6展示了一个简化版的高通滤波器算法。从前一个事件得到的加速计数值存储在类的 `accelX`、`accelY`、和 `accelZ` 成员变量中。清单中的代码首先计算低通滤波器的值，然后从当前加速计数据中减去该值，得到仅包含实时运动成分的数据。

程序清单8-6 从加速计数据中分离出实时运动成分

```
#define kFilteringFactor 0.1
```

```
- (void)accelerometer:(UIAccelerometer *)accelerometer didAccelerate:(UIAcceleration *)acceleration {  
    // Subtract the low-pass value from the current value to get a simplified high-pass filter  
    accelX = acceleration.x - ( (acceleration.x * kFilteringFactor) + (accelX * (1.0 - kFilteringFactor)) );  
    accelY = acceleration.y - ( (acceleration.y * kFilteringFactor) + (accelY * (1.0 - kFilteringFactor)) );  
    accelZ = acceleration.z - ( (acceleration.z * kFilteringFactor) + (accelZ * (1.0 - kFilteringFactor)) );  
  
    // Use the acceleration data.  
}
```


取得当前设备的方向

如果您需要知道的是设备的大体方向，而不是精确的方向向量，则应该通过 [UIDevice](#) 类的相关方法来取得。使用 [UIDevice](#) 接口比较简单，不需要自行计算方向向量。

在取得当前方向之前，您必须调用 [beginGeneratingDeviceOrientationNotifications](#) 方法，使 [UIDevice](#) 类开始产生设备方向[通告](#)。对该方法的调用会打开加速计硬件（否则为了省电，加速计硬件处于关闭状态）。

在打开方向通告的很短时间后，您就可以从 [UIDevice](#) 对象 [orientation](#) 属性声明得到当前的方向。您也可以通过注册接收 [UIDeviceOrientationDidChangeNotification](#) 通告来得到方向信息，当设备的大体方向发生改变时，系统就会发出该通告。设备的方向由 [UIDeviceOrientation](#) 常量来描述，它可以指示设备处于景观模式还是肖像模式，以及设备的正面是朝上还是朝下。这些常量指示的是设备的物理方向，不一定和应用程序的用户界面相对应。

当您不再需要设备的方向信息时，应该调用 [UIDevice](#) 的 [endGeneratingDeviceOrientationNotifications](#) 方法来关闭方向通告，使系统有机会关闭加速计硬件，如果其它地方也不使用的话。

使用位置和方向服务

[Core Location](#) [框架](#)为定位用户当前位置和方向（Heading）提供支持，它负责从相应的设备硬件收集信息，并以异步的方式报告给您的应用程序。数据是否可用取决于设备的类型以及所需的硬件当前是否打开，如果设备处于飞行模式，则某些硬件可能不可用。

在使用 [Core Location](#) 框架的接口之前，必须将 [CoreLocation.framework](#) 加入到您的 Xcode 工程中，并在相关的目标中进行连接。要访问该框架的类和头文件，还需要在相应的源代码文件的顶部包含 `#import <CoreLocation/CoreLocation.h>` 语句。更多有关如何在工程中加入框架的信息，请参见 [Xcode 工程管理指南](#) 文档中的[工程中的文件](#)部分。

有关 [Core Location](#) 框架的类的一般性信息请参见 [Core Location 框架参考](#)。

取得用户的当前位置

[Core Location](#) 框架使您可以定位设备的当前位置，并将这个信息应用到程序中。该框架利用设备内置的硬件，在已有信号的基础上通过三角测量得到固定位置，然后将它报告给您的代码。在接收到新的或更为精确的信号时，该框架还对位置信息进行更新。

如果您确实需要使用 [Core Location](#) 框架，则务必控制在最小程度，且正确地配置位置服务。收集位置数据需要给主板上的接收装置上电，并向基站、Wi-Fi 热点、或者 GPS 卫星查询，这个过程可能要花几秒钟的时间。此外，请求更高精度的位置数据可能需要让接收装置更长时间地处于打开状态，而长时间地打开这个硬件会耗尽设备的电池。如果位置信息不是频繁变化，通常可以先取得初始位置，然后每隔一段时间请求一次更新就可以了。如果您确实需要定期更新位置信息，也可以为位置服务设置一个最小的距离阈值，从而最小化代码必须处理的位置更新。

取得用户当前位置首先要[创建 CLLocationManager](#) 类的实例，并用期望的精度和阈值参数进行配置。开始接收通告则需要为该对象分配一个[委托](#)，然后调用 [startUpdatingLocation](#) 方法来确定用户当前位置。当新的位置数据到来时，位置管理器会通知它的委托对象。如果位置更新通告已经发送完成，您也可以直接从 [CLLocationManager](#) 对象获取最新的位置数据，而

不需要等待新的事件。

程序清单 8-7 展示了定制的 `startUpdates` 方法和 [locationManager:didUpdateToLocation:fromLocation:](#) 委托方法的一个实现。`startUpdates` 方法创建一个新的位置管理器对象（如果尚未存在的话），并用它启动位置更新事件的递送（在这个实例中，`locationManager` 变量是 `MyLocationGetter` 类中声明的成员变量，该类遵循 [CLLocationManagerDelegate](#) 协议）。事件处理方法通过事件的时间戳来确定其延迟的程度，对于太过时的事件，该方法会直接忽略，并等待更为实时的事件。在得到足够实时的数据后，即关闭位置服务。

程序清单8-7 发起和处理位置更新事件

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

```
@implementation MyLocationGetter
- (void)startUpdates
{
    // Create the location manager if this object does not
    // already have one.
    if (nil == locationManager)
        locationManager = [[CLLocationManager alloc] init];

    locationManager.delegate = self;
    locationManager.desiredAccuracy = kCLLocationAccuracyKilometer;

    // Set a movement threshold for new events
    locationManager.distanceFilter = 500;

    [locationManager startUpdatingLocation];
}

// Delegate method from the CLLocationManagerDelegate protocol.
- (void)locationManager:(CLLocationManager *)manager
    didUpdateToLocation:(CLLocation *)newLocation
    fromLocation:(CLLocation *)oldLocation
{
    // If it's a relatively recent event, turn off updates to save power
    NSDate* eventDate = newLocation.timestamp;
    NSTimeInterval howRecent = [eventDate timeIntervalSinceNow];
    if (abs(howRecent) < 5.0)
    {
        [manager stopUpdatingLocation];

        printf("latitude %+.6f, longitude %+.6f\n",
```

```

        newLocation.coordinate.latitude,
        newLocation.coordinate.longitude);
    }
    // else skip the event and process the next one.
}
@end

```

对时间戳进行检查是推荐的做法，因为位置服务通常会立即返回最后缓存的位置事件。得到一个大致的固定位置可能要花几秒钟的时间，更新之前的数据只是反映最后一次得到的数据。您也可以通过精度来确定是否希望接收位置事件。位置服务在收到精度更高的数据时，可能返回额外的事件，事件中的精度值也会反映相应的精度变化。

请注意：Core Location 框架在位置请求的一开始（而不是请求返回的时候）记录时间戳。由于 Core Location 使用几个不同的技术来取得固定位置，位置请求返回的顺序有时可能和时间戳指示的顺序不同。这样，新事件的时间戳有时会比之前的事件还要老一点，这是正常的。Core Location 框架致力于提高每个新事件的位置精度，而不考虑时间戳的值。

获取与方向有关的事件

Core Location 框架支持两种获取方向信息的方法。包含 GPS 硬件的设备可以提供当前移动方向的大致信息，该信息和经纬度数据通过同一个位置事件进行传递。包含磁力计的设备可以通过方向对象提供更为精确的方向信息，方向对象是 [CLHeading](#) 类的实例。

通过 GPS 硬件取得大致方向的过程和“[取得用户的当前位置](#)”部分的描述是一样的，框架会向您的应用程序委托传递一个 [CLLocation](#) 对象，对象中的 [course](#) 和 [speed 属性声明](#) 包含相关的信息。这个接口适用于需要跟踪用户移动的大多数应用程序，比如实现汽车导航系统的导航程序。对于基于指南针或者可能需要了解用户静止时朝向的应用程序，可以请求位置管理器提供方向对象。

您的程序必须运行在包含磁力计的设备上才能接收方向对象。磁力计可以测量地球散发的磁场，进而确定设备的准确方向。虽然磁力计可能受到局部磁场（比如扬声器的永磁铁、马达、以及其它类型电子设备发出的磁场）的影响，但是 Core Location 框架具有足够的智能，可以过滤很多局部磁场的影响，确保方向对象包含有用的数据。

请注意：如果路线或方向信息对于您的应用程序是必须的，则应该在程序的 Info.plist 文件中正确地包含 `UIRequiredDeviceCapabilities` 键。这个键用于指定应用程序正常工作需要具备的设备特性，您可以用它来指定设备必须具有 GPS 和磁力计硬件。更多有关这个键值设置的信息请参见“[信息属性列表](#)”部分。

为了接收方向事件，您需要创建一个 [CLLocationManager](#) 对象，为其分配一个委托对象，并调用其 [startUpdatingHeading](#) 方法，如程序清单 8-8 所示。然而，在请求方向事件之前，应该检查一下位置管理器的 [headingAvailable](#) 属性，确保相应的硬件是存在的。如果该硬件不存在，应用程序应该回退到通过位置事件获取路线信息的代码路径。

程序清单 8-8 发起方向事件的传送

```

CLLocationManager* locationManager = [[CLLocationManager alloc] init];
if (locationManager.headingAvailable)
{
    locationManager.delegate = myDelegateObject; // Assign your custom delegate
    object

```

```

locManager.headingFilter = 5;
[locManager startUpdatingHeading];
}
else

```

// Use location events instead

您赋值给 `delegate` 属性的对象必须遵循 [CLLocationManagerDelegate 协议](#)。当一个新的方向事件到来时，位置管理器会调用 [locationManager:didUpdateHeading:](#) 方法，将事件传递给您的应用程序。一旦收到新的事件，应用程序应该检查 [headingAccuracy](#) 属性，确保刚收到的数据是有效的，具体做法如清单8-9。

程序清单8-9 处理方向事件

```

- (void)locationManager:(CLLocationManager*)manager
didUpdateHeading:(CLHeading*)newHeading
{
    // If the accuracy is valid, go ahead and process the event.
    if (newHeading.headingAccuracy > 0)
    {
        CLLocationDirection theHeading = newHeading.magneticHeading;

        // Do something with the event data.
    }
}

```

CLHeading 对象的 [magneticHeading](#) 属性包含主方向数据，且该数据一直存在。这个属性给出了相对于磁北极的方向数据，磁北极和北极不在同一个位置上。如果您希望得到相对于北极（也称为地理北极）的方向数据，则必须在 `startUpdatingHeading` 之前调用 `startUpdatingLocation` 方法来启动位置更新，然后通过 CLHeading 对象的 [trueHeading](#) 属性取得相对于地理北极的方向。

显示地图和注解

iPhone OS 3.0引入了 Map Kit [框架](#)。通过这个框架可以在应用程序的窗口中嵌入一个全功能的地图界面。Maps 程序中的很多常见功能都包含在这个框架提供的地图支持中，您可以通过它来显示标准的街道地图、卫星图像，或两者的组合；还可以通过代码来缩放和移动地图。该框架还自动支持触摸事件，用户可以用手指缩放或移动地图。您还可以在地图中加入自己定制的注释信息，以及用框架提供的反向地理编码功能寻找和地图坐标关联的地址。在使用 Map Kit 框架的功能之前，必须将 `MapKit.framework` 加入到 Xcode 工程中，并且在相关的目标中加以连接；在访问框架的类和头文件之前，需要在相应的源代码文件的顶部加入 `#import <MapKit/MapKit.h>` 语句。有关如何将框架加入工程的更多信息，请参见 [Xcode 工程管理指南](#) 中的 [工程中的文件](#) 部分；有关 Map Kit 框架类的一般性信息，则请参见 [MapKit 框架参考](#)。

重要提示： Map Kit 框架使用 Google 的服务来提供地图数据。框架及其相关接口的使用必须遵守 Google Maps/Google Earth API 的服务条款，具体条款信息位于 <http://code.google.com/apis/maps/iphone/terms.html>。

在用户界面中加入地图视图

为应用程序加入地图之前，需要在应用程序的视图层次中嵌入一个 [MKMapView](#) 类的实例，该类为地图信息的显示和用户交互提供支持。您可以通过代码来为该类型[创建实例](#)，并通过 [initWithFrame:](#) 方法来对其进行[初始化](#)，或者用 Interface Builder 将它加入到 [nib 文件](#) 中。

地图视图也是个视图，因此您可以通过它的 [frame 属性声明](#) 随意调整它的位置和尺寸。虽然地图视图本身没有提供任何控件，但是您可以在它的上面放置工具条或其它视图，使用户可以和地图内容进行交互。您在地图视图中加入的所有子视图的位置是不变的，不会随着地图内容的滚动而滚动。如果您希望在地图上加入定制的内容，并使它们跟着地图滚动，则必须创建注解，具体描述请参见[“显示注解”](#)部分。

MKMapView 类有很多属性，可以在显示之前进行配置，其中最重要的是 [region](#) 属性，负责定义最初显示的地图部分及如何缩放和移动地图内容。

缩放和移动地图内容

MKMapView 类的 [region](#) 属性控制着当前显示的地图部分。当您希望缩放和移动地图时，需要做的只是正确改变这个属性的值。这个属性包含一个 [MKCoordinateRegion](#) 类型的结构，其定义如下：

```
typedef struct {
    CLLocationCoordinate2D
center;
    MKCoordinateSpan span;
} MKCoordinateRegion;
```

改变 *center* 域可以将地图移动到新的位置；而改变 *span* 域 的值则可以实现缩放。这些域的值需要用地图坐标来指定，地图坐标用度、分、和秒来度量。对于 *span* 域，您需要通过经纬度距离来指定它的值。虽然纬度距离 相对固定，每度大约111公里，但是经度距离却是随着纬度的变化而变化的。在赤道上，经度距离大约每度111公里；而在地球的极点上，这个值则接近于零。当然，您总是可以通过 [MKCoordinateRegionMakeWithDistance](#) 函数来创建基于公里值（而不是度数）的区域。

如果您希望在更新地图时不显示过程动画，可以直接修改 *region* 或 *centerCoordinate* 属性的值；如果需要动画过程，则必须使用 [setRegion:animated:](#) 或 [setCenterCoordinate:animated:](#) 方法。[setCenterCoordinate:animated:](#) 方法可以移动地图，且避免在无意中触发缩放，而 [setRegion:animated:](#) 方法则可以同时缩放和移动地图。举例来说，如果您要使地图向左移动，移动距离为当前宽度的一半，则可以通过下面的代码找到地图左边界的坐标，然后将它用于中心点的设置，如下所示：

```
CLLocationCoordinate2D mapCenter = myMapView.centerCoordinate;
mapCenter = [myMapView convertPoint:
              CGPointMake(1,
(myMapView.frame.size.height/2.0))
              toCoordinateFromView:myMapView];
```

```
[myMapView setCenterCoordinate:mapCenter animated:YES];
```

缩放地图则应该修改 *span* 属性的值，而不是中心坐标。减少 *span* 属性值可以使视图缩小；

相反，增加该属性值可以使视图放大。换句话说，如果当前的 `span` 值是一度，将它指定为两度会使地图跨度放大两倍：

```
MKCoordinateRegion theRegion = myMapView.region;
```

```
// Zoom out
theRegion.span.longitudeDelta *= 2.0;
theRegion.span.latitudeDelta *= 2.0;
[myMapView setRegion:theRegion animated:YES];
```

显示用户的当前位置

Map Kit 框架内置支持将用户的当前位置显示在地图上，具体做法是将地图视图对象的 [showsUserLocation](#) 属性值设置为 YES 就可以了。进行这个设置会使地图视图通过 Core Location 框架找到用户位置，并在地图上加入类型为 [MKUserLocation](#) 的注解。

在地图上加入 [MKUserLocation](#) 注解对象的事件会通过 [委托](#) 对象进行报告，这和定制注解的报告方式是一样的。如果您希望在用户位置上关联一个定制的注解视图，应该在委托对象的 [mapView:viewForAnnotation:](#) 方法中返回该视图。如果您希望使用缺省的注解视图，则应该在该方法中返回 nil。

坐标和像素之间的转换

您通常通过经纬度值来指定地图上的点，但有些时候也需要在经纬度值和地图视图对象中的像素之间进行转换。举例来说，如果您允许用户在地图表面拖动注解，定制注解视图的事件处理器代码就需要将边框坐标转换为地图坐标，以便更新关联的注解对象。[MKMapView](#) 类中几个例程，用于在地图坐标和地图视图对象的本地坐标系之间进行转换，这些例包括：

[convertCoordinate:toPointToView:](#)

[convertPoint:toCoordinateFromView:](#)

[convertRegion:toRectToView:](#)

[convertRect:toRegionFromView:](#)

有关如何处理定制注解事件的更多信息，请参见[“处理注解视图中的事件”](#)部分。

显示注解

注解是您定义并放置在地图上面的信息片段。Map Kit [框架](#)将注解实现为两个部分，即注解对象和用于显示注解的视图。大多数情况下，您需要负责提供这些定制对象，但框架也提供一些标准的注解和视图供您使用。

在地图视图上显示注解需要两个步骤：

[创建注解对象](#)并将它加入到地图视图中。

- 1 在自己的 [委托对象](#) 中实现 [mapView:viewForAnnotation:](#) 方法，并在该方法中创建相应的注解视图。

注解对象是指遵循 [MKAnnotation 协议](#) 的任何对象。通常情况下，注解对象是相对小的数据对象，存储注解的坐标及相关信息，比如注解的名称。注解是通过协议来定义的，因此应

用程序中的任何对象都可以成为注解对象。然而，在实践上，注解对象应该是轻量级的，因为在显式删除注解对象之前，地图视图会一直保存它们的引用。注意，同样的结论并不一定适用于 注解视图。

在将注解显示在屏幕上时，地图视图负责确保注解对象具有相关联的注解视图，具体的方法是在注解坐标即将变为可见时调用其委托对象的 `mapView:viewForAnnotation:` 方法。但是，由于注解视图的量级通常总是比其对应的注解对象更重，所以地图对象尽可能不在内存中同时保存很多注解视图。为此，它实现了注解视图的回收机制。这个机制和表视图在滚动时回收表单元使用的机制相类似，即当一个注解视图移出屏幕时，地图视图就解除其与注解对象之间关联，将它放入重用队列。而在创建新的注解视图之前，委托的 `mapView:viewForAnnotation:` 方法应该总是调用地图对象的 [dequeueReusableAnnotationViewWithIdentifier:](#) 方法来检查重用队列中是否还有可用的视图对象。如果该方法返回一个正当的视图对象，您就可以对其进行再次初始化，并将它返回；否则，您再创建和返回一个新的视图对象。

添加和移除注解对象

您不应直接在地图上添加注解视图，而是应该添加注解对象，注解对象通常不是视图。注解对象可以是应用程序中遵循 [MKAnnotation](#) 协议的任何对象。注解对象中最重要的部分是它的 [coordinate](#) 属性声明，它是 MKAnnotation 协议必需实现的属性，用于为地图上的注解提供锚点。

往地图视图加入注解所需要的工作就是调用地图视图对象的 [addAnnotation:](#) 或 [addAnnotations:](#) 方法。何时往地图视图加入注解以及何时为加入的注解提供用户界面由您自己来决定。您可以提供一个工具条，由用户通过工具条上的命令来创建注解，或者也可以自行编码创建注解，注解信息可能来自本地或远程的数据库信息。

如果您的应用程序需要删除某个老的注解，则在删除之前，应该调用 [removeAnnotation:](#) 或 [removeAnnotations:](#) 方法将它从地图中移除。地图视图会显示它知道的所有注解，如果您不希望某些注解被显示在地图上，就需要显式地将它们删除。例如，如果您的应用程序允许用户对餐厅或本地风景点进行过滤，就需要删除与过滤条件不匹配的所有注解。

定义注解视图

Map Kit 框架提供了两个注解视图类：`MKAnnotationView` 和 `MKPinAnnotationView`。[MKAnnotationView](#) 类是一个具体的视图，定义了所有注解视图的基本行为。[MKPinAnnotationView](#) 类则是 `MKAnnotationView` 的子类，用于在关联的注解坐标点上显示一个标准的系统大头针图像。

您可以将 `MKAnnotationView` 类用于显示简单的注解，也可以从该类派生出子类，提供更多的交互行为。在直接使用该类时，您需要提供一个定制的图像，用于在地图上表示您希望显示的内容，并将它赋值给注解视图的 [image](#) 属性。如果您显示的内容不需要动态改变，而且不需要支持用户交互，则这种用法是非常合适的。但是，如果您需要支持动态内容和用户交互，则必须定义定制子类。

在一个定制的子类中，有两种方式可以描画动态内容：可以继续使用 [image](#) 属性来显示注解图像，这样或许需要设置一个定时器，负责定时改变当前的图像；也可以重载视图的 [drawRect:](#) 方法来显示描画您的内容，这种方法也需要设置一个定时器，以定时调用视图的

[setNeedsDisplay](#) 方法。

如果您通过 `drawRect:` 方法来描画内容，则必须记住：要在注解视图初始化后不久为其指定尺寸。注解视图的缺省[初始化](#)方法并不包含边框矩形参数，而是在初始化后通过您分配给 `image` 属性的图像来设置边框尺寸。如果您没有设置图像，就必须显式设置边框尺寸，您渲染的内容才会被显示。

有关如何在注解视图中支持用户交互的信息，请参见[“处理注解视图的事件”](#)部分；有关如何设置定时器的信息，则请参见 [Cocoa 定时器编程主题](#)。

创建注解视图

您应该总是在[委托对象](#)的 `mapView:viewForAnnotation:` 创建注解视图。在创建新视图之前，您应该总是调用 `dequeueReusableAnnotationViewWithIdentifier:` 方法来检查是否有可重用的视图，如果该方法返回非 `nil` 值，就应该将地图视图提供的注解分配给重用视图的 `annotation` 属性，并执行其它必要的配置，使视图处于期望的状态，然后将它返回；如果该方法返回 `nil`，则应该创建并返回一个新的注解视图对象。

程序清单8-10是 `mapView:viewForAnnotation:` 方法的一个例子实现，展示了如何为定制注解对象提供大头针注解视图。如果队列中已经存在一个大头针注解视图，该方法就将它和相应的注解对象相关联；如果重用队列中没有视图，该方法则创建一个新的视图，对其基本属性进行配置，并为插图符号配置一个附加视图。

程序清单8-10 创建注解视图

```
-(MKAnnotationView *)mapView:(MKMapView *)mapView
    viewForAnnotation:(id <MKAnnotation>)annotation
{
    // If it's the user location, just return nil.
    if ([annotation isKindOfClass:[MKUserLocation class]])
        return nil;

    // Handle any custom annotations.
    if ([annotation isKindOfClass:[CustomPinAnnotation class]])
    {
        // Try to dequeue an existing pin view first.
        MKPinAnnotationView* pinView =
            (MKPinAnnotationView*)[mapView
                dequeueReusableAnnotationViewWithIdentifier:@"CustomPinAnnotation"];

        if (!pinView)
        {
            // If an existing pin view was not available, create one
            pinView = [[[MKPinAnnotationView alloc]
                initWithAnnotation:annotation
                reuseIdentifier:@"CustomPinAnnotation"]
```

```

        autorelease];
pinView.pinColor = MKPinAnnotationColorRed;
pinView.animatesDrop = YES;
pinView.canShowCallout = YES;

// Add a detail disclosure button to the callout.
UIButton* rightButton = [UIButton buttonWithType:
                        UIButtonTypeDetailDisclosure];
[rightButton addTarget:self
action:@selector(myShowDetailsMethod:)]

forControlEvents:UIControlEventTouchUpInside];
pinView.rightCalloutAccessoryView = rightButton;
}
else
pinView.annotation = annotation;

return pinView;
}

return nil;
}

```

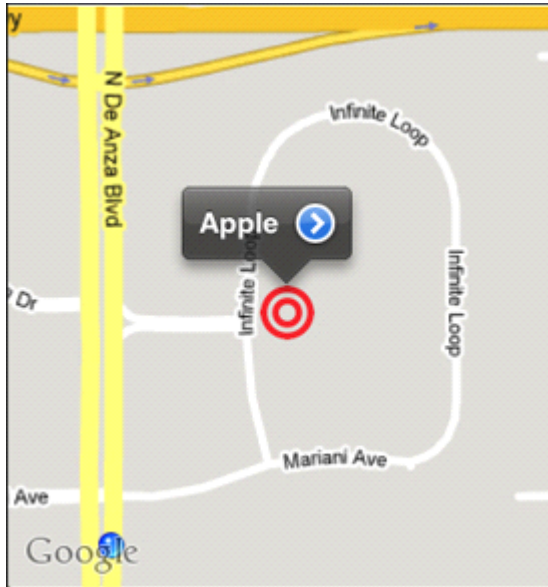
处理注解视图中的事件

虽然注解视图位于地图内容上面的特殊层中，但它们也是功能完全的视图，能够接收触摸事件。您可以通过这些事件来实现用户和注解之间的交互。比如，您可以通过视图中的触摸事件来实现注解在地图表面的拖拽行为。

请注意：由于地图被显示在一个滚动界面上，所以，在用户触击定制视图和事件最终被派发之间往往有一个小的延迟。滚动视图可以利用这个延迟来确定触摸事件是否为某种滚动手势的一部分。

随后的一系列示例代码将向您展示如何实现一个支持用户拖动的注解视图。例子中的注解视图直接在注解坐标点上显示一个公牛眼图像，并包含一个定制的附加视图，用以显示目的地的详细信息。图8-1显示注解视图的一个实例以及其包含的气泡符号。

图8-1 公牛眼注解视图



程序清单8-11显示了 BullseyeAnnotationView 类的定义。类中包含一些正确跟踪视图移动需要的其它成员变量，以及一个指向地图视图本身的指针，指针的值是在 mapView:viewForAnnotation:方法中设置的，该方法是创建或再次初始化注解视图的地方。在事件跟踪完成后，代码需要调整注解对象的地图坐标，这时需要用到地图视图对象。

程序清单8-11 BullseyeAnnotationView 类

```
@interface BullseyeAnnotationView : MKAnnotationView
{
    BOOL isMoving;
    CGPoint startLocation;
    CGPoint originalCenter;

    MKMapView* map;
}

@property (assign, nonatomic) MKMapView* map;

- (id)initWithAnnotation:(id <MKAnnotation>)annotation;

@end

@implementation BullseyeAnnotationView
@synthesize map;
- (id)initWithAnnotation:(id <MKAnnotation>)annotation
{
    self = [super initWithAnnotation:annotation
                      reuseIdentifier:@"BullseyeAnnotation"];
    if (self)
    {
```



```

        UIImage* theImage = [UIImage
imageNamed:@"bullseye32.png"];
        if (!theImage)
            return nil;

        self.image = theImage;
        self.canShowCallout = YES;
        self.multipleTouchEnabled = NO;
        map = nil;

        UIButton* rightButton = [UIButton buttonWithType:
UIButtonTypeDetailDisclosure];
        [rightButton addTarget:self
action:@selector(myShowAnnotationAddress:)
forControlEvents:UIControlEventTouchUpInside];
        self.rightCalloutAccessoryView = rightButton;
    }
    return self;
}
@end

```

当触击事件首次到达公牛眼视图时, 该类的 [touchesBegan:withEvent:](#) 方法会记录事件的信息, 作为初始信息, 如清单8-12所示。 [touchesMoved:withEvent:](#) 方法会利用这些信息来调整视图位置。所有的位置信息都存储在父视图的坐标空间中。

程序清单8-12 跟踪视图的位置

```

@implementation BullseyeAnnotationView (TouchBeginMethods)
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    // The view is configured for single touches only.
    UITouch* aTouch = [touches anyObject];
    startLocation = [aTouch locationInView:[self superview]];
    originalCenter = self.center;

    [super touchesBegan:touches withEvent:event];
}

- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch* aTouch = [touches anyObject];
    CGPoint newLocation = [aTouch locationInView:[self superview]];
    CGPoint newCenter;

    // If the user's finger moved more than 5 pixels, begin the drag.
    if ( (abs(newLocation.x - startLocation.x) > 5.0) ||

```

```

        (abs(newLocation.y - startLocation.y) > 5.0) )
        isMoving = YES;

    // If dragging has begun, adjust the position of the view.
    if (isMoving)
    {
        newCenter.x = originalCenter.x + (newLocation.x -
startLocation.x);
        newCenter.y = originalCenter.y + (newLocation.y -
startLocation.y);
        self.center = newCenter;
    }
    else // Let the parent class handle it.
        [super touchesMoved:touches withEvent:event];
}
@end

```

当用户停止拖动注解视图时，您需要调整原有注解的坐标，确保视图位于新的位置。清单8-13显示了 `BullseyeAnnotationView` 类的 [touchesEnded:withEvent:](#) 方法，该方法通过地图成员变量将基于像素的点转化为地图坐标值。由于注解的 [coordinate](#) 属性通常是只读的，所以例子中的注解对象实现了一个名为 `changeCoordinate` 的定制方法，负责更新它在本地存储的值，而这个值可以通过 `coordinate` 属性取得。如果触摸事件由于某种原因被取消，[touchesCancelled:withEvent:](#) 方法会使注解视图回到原来的位置。

程序清单8-13 处理最后的触摸事件

```

@implementation BullseyeAnnotationView (TouchEndMethods)
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
    if (isMoving)
    {
        // Update the map coordinate to reflect the new position.
        CGPoint newCenter = self.center;
        BullseyeAnnotation* theAnnotation = self.annotation;
        CLLocationCoordinate2D newCoordinate = [map
convertPoint:newCenter
toCoordinateFromView:self.superview];

        [theAnnotation changeCoordinate:newCoordinate];

        // Clean up the state information.
        startLocation = CGPointZero;
        originalCenter = CGPointZero;
        isMoving = NO;
    }
    else

```

```

        [super touchesEnded:touches withEvent:event];
    }

- (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event
{
    if (isMoving)
    {
        // Move the view back to its starting point.
        self.center = originalCenter;

        // Clean up the state information.
        startLocation = CGPointZero;
        originalCenter = CGPointZero;
        isMoving = NO;
    }
    else
        [super touchesCancelled:touches withEvent:event];
}
@end

```

通过反向地理编码器获取地标信息

Map Kit 框架主要处理地图坐标值。地图坐标值由经度和纬度组成的，比较易于在代码中使用，但却不是用户最容易理解的描述方式。为使用户更加易于理解，您可以通过 [MKReverseGeocoder](#) 类来取得与地图坐标相关联的地标信息，比如街道地址、城市、州、和国家。

[MKReverseGeocoder](#) 类负责向潜在的地图服务查询指定地图坐标的信息。由于需要访问网络，反向地理编码器对象总是以异步的方式执行查询，并将结果返回给相关联的[委托对象](#)。委托对象必须遵循 [MKReverseGeocoderDelegate](#) 协议。

启动反向地理编码器的具体做法是首先创建一个 [MKReverseGeocoder](#) 类的实例，并将恰当的对象赋值给该实例的 [delegate](#) 属性，然后调用 [start](#) 方法。如果查询成功完成，您的委托就会收到带有一个 [MKPlacemark](#) 对象的查询结果。[MKPlacemark](#) 对象本身也是注解对象——也就是说，它们采纳了 [MKAnnotation 协议](#)——因此如果您愿意的话，可以将它们添加到地图视图的注解列表中。

用照相机照相

通过 UIKit 的 [UIImagePickerController](#) 类可以访问设备的照相机。该类可以显示标准的系统界面，使用户可以通过现有的照相机拍照，以及对拍得的图像进行裁剪和尺寸调整；该类还可以用于从用户照片库中选取照片。

照相机界面是一个模式视图，由 [UIImagePickerController](#) 类来管理。具体使用时，您不应从代码中直接访问该视图，而是应该调用当前活动的[视图控制器](#)的 [presentModalViewController:animated:](#) 方法，并向其传入一个 [UIImagePickerController](#) 对象作

为新的视图控制器。一旦被安装，选取控制器就会自动将照相机界面滑入屏幕，并一直保持活动，直到用户确认或取消图像选取的操作。如果用户做出选择，选取控制器会将这个事件通知其委托对象。

[UIImagePickerController](#) 类管理的界面可能并不适用于所有的设备。在显示照相机界面之前，您应该调用 [UIImagePickerController](#) 类的 [isSourceTypeAvailable:类方法](#)，确认该界面是否可用。您应该总是尊重该方法的返回值，如果它返回 NO，意味着当前设备没有照相机，或者照相机由于某种原因不可用；如果返回 YES，则可以通过下面的步骤显示照相机界面：

[创建](#)一个新的 [UIImagePickerController](#) 对象。

为该对象分配一个[委托对象](#)。

大多数情况下，您可以让当前的视图控制器充当选取控制器的委托，但也可以根据自己的喜好使用完全不同的对象。委托对象必须遵循 [UIImagePickerControllerDelegate](#) 和 [UINavigationControllerDelegate](#) 协议。

请注意：如果您的委托不遵循 [UINavigationControllerDelegate](#) 协议，在编译时就会看到警告信息。然而，由于该协议的方法是可选的，所以不会对代码带来什么影响。如果要消除该警告信息，需要将 [UINavigationControllerDelegate](#) 协议加入委托类支持的协议列表中。

将选取控制器的类型设置为 [UIImagePickerControllerSourceTypeCamera](#)。

为 [allowsImageEditing](#) 属性声明设置恰当的值，以便激活或者禁用图片编辑控制。这是个可选步骤。

调用当前视图控制器的 [presentModalViewController:animated:](#)方法，显示选取控制器。

程序清单8-14的代码实现了上述步骤。在调用 [presentModalViewController:animated](#) 方法之后，选取控制器随即接管控制权，将照相机界面显示出来，并负责响应所有的用户交互，直到退出该界面。而从用户照片库中选取现有照片需要做的只是将选取控制器的 [sourceType](#) 属性的值改为 [UIImagePickerControllerSourceTypePhotoLibrary](#) 就可以了。

程序清单8-14 显示照相界面

```
-(BOOL)startCameraPickerFromViewController:(UIViewController*)controller
usingDelegate:(id<UIImagePickerControllerDelegate>)delegateObject
{
    if ( (![UIImagePickerController
isSourceTypeAvailable:UIImagePickerControllerSourceTypeCamera])
        || (delegateObject == nil) || (controller == nil))
        return NO;

    UIImagePickerController* picker = [[UIImagePickerController alloc] init];
    picker.sourceType = UIImagePickerControllerSourceTypeCamera;
    picker.delegate = delegateObject;
    picker.allowsImageEditing = YES;

    // Picker is displayed asynchronously.
    [controller presentModalViewController:picker animated:YES];
    return YES;
}
```

当用户触击相应的按键关闭照相机界面时，[UIImagePickerController](#) 会将用户的动作通知委托对象，但并不直接实施关闭操作。选取器界面的关闭由委托对象负责（您的应用程序还必

须负责在不需要选取器对象时将它释放，这个工作也 可以在委托方法中进行)。由于这个原因，委托对象实际上应该是将选取器显示出来的视图控制器对象。一旦收到委托消息，视图控制器会调用其 [dismissModalViewControllerAnimated:](#)方法来关闭照相机界面。

程序清单8-15展示了关闭照相机界面的委托方法，该界面是由[程序清单8-14](#)的代码显示出来的。这些方法是由一个名为 `MyViewController` 的定制类实现的，它是 `UIViewController` 的一个子类。在这个例子中，执行这些代码和显示选取器的应该是同一个对象。`useImage:`方法是一个空壳，应该被您的定制代码代替，您可以在这个方法中使用用户选取的图像。

程序清单8-15 图像选取器的委托方法

@implementation MyViewController (ImagePickerDelegateMethods)

```
- (void)imagePickerController:(UIImagePickerController *)picker
    didFinishPickingImage:(UIImage *)image
    editingInfo:(NSDictionary *)editingInfo
{
    [self useImage:image];

    // Remove the picker interface and release the picker object.
    [[picker
                                     parentViewController]
 dismissModalViewControllerAnimated:YES];
    [picker release];
}

- (void)imagePickerControllerDidCancel:(UIImagePickerController *)picker
{
    [[picker
                                     parentViewController]
 dismissModalViewControllerAnimated:YES];
    [picker release];
}

// Implement this method in your code to do something with the image.
- (void)useImage:(UIImage*)theImage
{
}

@end
```

如果图像编辑功能被激活，且用户成功选取了一张图片，则 [imagePickerController:didFinishPickingImage:editingInfo:](#)方法的 `image` 参数会包含编辑后的图像，您应该将这个图像作为用户选取的图像。当然，如果用户希望存储原始图像，可以从 `editingInfo` 参数的字典中得到（同时还可以得到编辑用的裁剪矩形）。

从照片库中选取照片

UIKit 通过 [UIImagePickerController](#) 类为访问用户照片库提供支持。这个[控制器](#)可以显示照

片选取器界面，用户可以通过该界面漫游用户照片库，选取某个图像，并将它返回给应用程序。您也可以打开用户编辑功能，使用户可以移动和裁剪返回的图像。这个类也可以用于显示一个照相机界面。

[UIImagePickerController](#) 类既可以显示照相机界面，也可以显示用户照片库，两种显示方式的使用步骤几乎一样。唯一的区别是是否将选取器对象的 [sourceType](#) 属性值设置为 [UIImagePickerControllerSourceTypePhotoLibrary](#)。显示照相机选取器的具体步骤请参见“[用照相机照相](#)”部分的讨论。

请注意：当您使用照相机选取器时，应该总是调用 [UIImagePickerController](#) 类的 [isSourceTypeAvailable](#) 类方法，并尊重其返回值，而不应假定给定的设备总是具有照片库功能。即使设备支持照片库，该方法仍然可能在照片库不可用时返回 NO。

使用邮件编辑界面

在 iPhone OS 3.0 及之后的系统中，您可以通过 [MFMailComposeViewController](#) 类 在应用程序内部显示一个标准的邮件发送界面。在显示该界面之前，您可以用该类的方法来配置邮件的接受者、主题、和希望包含的附件。当邮件在界面显示出来（通过标准的视图控制器技术）之后和提交给 Mail 程序进行发送之前，用户可以对邮件的内容进行编辑。用户也可以将整个邮件取消。

请注意：在所有版本的 iPhone OS 中，您可以通过创建和打开一个 `mailto` 类型的 URL 来制作邮件，这种类型的 URL 会自动传递给 Mail 程序进行处理。有关如何打开这种类型的 URL 的更多信息，请参见“[和其它应用程序间的通讯](#)”部分。

在使用邮件编辑界面之前，您必须首先把 `MessageUI.framework` 加入到工程中，并在相应的目标中进行连接。为了访问该框架中的类和头文件，还必须在相应的源代码文件的顶部包含 `#import <MessageUI/MessageUI.h>` 语句。有关如何在工程中加入框架的信息，请参见 [Xcode 工程管理指南](#) 文档中的 [工程中的文件](#) 部分。

应用程序在使用 `MFMailComposeViewController` 类时，必须首先 [创建一个实例](#)，并使用该方法设置初始的电子邮件数据；还必须为 [视图控制器](#) 的 [mailComposeDelegate](#) 属性声明分配一个对象，负责在用户接收或取消邮件发送时退出界面。您指定的 [委托对象](#) 必须遵循 [MFMailComposeViewControllerDelegate](#) 协议

在指定电子邮件地址时，应该使用纯字符串对象。如果您希望使用通讯录用户列表中的邮件地址，可以通过 Address Book 框架来实现。更多有关如何通过该框架获取电子邮件及其它数据的信息，请参见 [iPhone OS 的 Address Book 编程指南](#)。

程序清单 8-16 展示了如何在应用程序中创建 `MFMailComposeViewController` 对象，并用模式视图显示邮件编辑接口的代码。您可以将清单中的 `displayComposerSheet` 方法包含到定制的视图控制器中，并在需要时通过它来显示邮件编辑界面。在这个例子中，父视图控制器将自身作为委托，并实现了 [mailComposeController:didFinishWithResult:error:](#) 方法。该委托方法只是退出邮件编辑界面，没有进行更多的操作。在您自己的应用程序中，可以在委托方法中考察 `result` 参数的值，确定用户是否发送或取消了邮件。

程序清单 8-16 显示邮件编辑界面

@implementation WriteMyMailViewController (MailMethods)

-(void)displayComposerSheet

```

{
    MFMailComposeViewController *picker = [[MFMailComposeViewController alloc]
init];
    picker.mailComposeDelegate = self;

    [picker setSubject:@"Hello from California!"];

    // Set up the recipients.
    NSArray *toRecipients = [NSArray arrayWithObjects:@"first@example.com",
                                                    nil];
    NSArray *ccRecipients = [NSArray arrayWithObjects:@"second@example.com",
                                                    @"third@example.com", nil];
    NSArray *bccRecipients = [NSArray arrayWithObjects:@"four@example.com",
                                                    nil];

    [picker setToRecipients:toRecipients];
    [picker setCcRecipients:ccRecipients];
    [picker setBccRecipients:bccRecipients];

    // Attach an image to the email.
    NSString *path = [[NSBundle mainBundle] pathForResource:@"ipodnano"
                                                    ofType:@"png"];
    NSData *myData = [NSData dataWithContentsOfFile:path];
    [picker addAttachmentData:myData mimeType:@"image/png"
                               fileName:@"ipodnano"];

    // Fill out the email body text.
    NSString *emailBody = @"It is raining in sunny California!";
    [picker setMessageBody:emailBody isHTML:NO];

    // Present the mail composition interface.
    [self presentViewController:picker animated:YES];
    [picker release]; // Can safely release the controller now.
}

// The mail compose view controller delegate method
- (void)mailComposeController:(MFMailComposeViewController *)controller
    didFinishWithResult:(MFMailComposeResult)result
    error:(NSError *)error
{
    [self dismissModalViewControllerAnimated:YES];
}
@end

```

有关如何通过标准视图控制器技术显示界面的更多信息，请参见 [iPhone OS 视图控制器编程](#)

[指南](#)：有关 Message UI 框架中包含的类信息，则请参见 [Message UI 框架参考](#)。

应用程序偏好设置

在传统的桌面应用程序中，偏好设置是一些专门面向应用程序 的设置，用于配置应用程序的行为和外观。iPhone OS 也支持应用程序偏好设置，但并不将它作为应用程序整体的一部分。在 iPhone OS 上，应用程序级别的偏好设置并不由各个程序本身的定制界面来显示，而是由系统提供的 Settings 程序统一显示。

为了将定制的应用程序偏好设置集成到 Settings 程序中，您必须在应用程序包的顶级目录中包含一个特殊格式的 Settings [程序包](#)，由它负责将应用程序的偏好设置信息提供给 Settings 程序，而 Settings 程序则负责对其进行显示，并将用户提供的值写入偏好设置数据库。在运行时，您的应用程序可以通过标准的 API 取得这些偏好设置的值。本章的下面部分将描述 Settings 程序包的格式，以及用于取得偏好设置值的 API。

偏好设置的指导原则

将 偏好设置加入到 Settings 程序的做法最适合于效率工具类型的应用程序，以及偏好设置值配置完成后很少再改变的程序。Mail 程序就是一个例子，它通过这种形式的偏好设置来存储用户账户信息及消息检查设置。由于 Settings 程序可以按层次进行显示，所以当您有大量的偏好设置时，通过 Settings 程序来进行操作也是比较合适的，在自己的应用程序中提供同样的偏好设置集合可能需要太多屏幕，而且可能造成用户的混淆。

当 您的应用程序只需要少数的选项，或者用户需要经常改变这些选项时，应该认真考虑是否用 Settings 程序来管理。举例来说，工具程序更适合在主视图的背面提供定制的配置选项，即在视图上通过一个特殊的控件翻转视图，显示应用程序的选项，再通过另一个控件将视图翻转回来。对于简单的应用程序，这种方式使用户可以立即访问应用程序选项，比使用 Settings 程序方便得多。

对于游戏和其它全屏程序的预置，可以使用 Settings 程序或自行实现定制的屏幕。定制屏幕通常更适合游戏程序，因为偏好设置可以处理为游戏设置的一部分。当然，您也可以使用 Settings 程序，如果您认为那样对游戏的使用流程更好的话。

请注意：永远不要使偏好设置同时存在于 Setting 程序和自定义的应用程序屏幕上。举例来说，如果工具类应用程序在主视图的背面有偏好设置，则在 Settings 程序中就不应该再有可配置的设置。如果您的应用程序需要进行偏好设置，则请仅选择和使用一种方案。

偏好设置的接口

Settings 程序实现了一组有层次的页面，用于访问应用程序的偏好设置。Settings 程序的主视图显示了可以进行偏好设置的系统程序及第三方应用程序，用户选择一个第三程序后会进入该程序的偏好设置页面。

每个应用程序都至少有一个偏好设置页面，我们称为主页面。如果您的应用程序只有少数几个偏好设置，则一个主页面可能就够了。然而，如果偏好设置太多，在主页面上放不下，也可以加入更多页面。这些额外的页面就成为主页面的子页面，用户通过轻触特定类型的偏好设置来访问这些页面。

您显示的每一个偏好设置都必须具有特定的类型。偏好设置的类型定义了 **Settings** 程序如何对其进行显示。大多数偏好设置类型都和某种类型的、用于进行设置的控件相关联，而另外一些类型则提供一种偏好设置的组织方式。表9-1列出了 **Settings** 程序支持的各种元素类型，以及如何用这些类型来实现自己的偏好设置页面。

表 9-1 偏好设置元素的类型 元素类型	描述
文本框	文本框类型显示一个可选的标题和一个可编辑的文本输入框，适用于需要用户输入自定义字符串的偏好设置。 这个类型的键是 <code>PSTextFieldSpecifier</code> 。
标题	标题类型显示一个只读的字符串，适用于显示只读字符串的偏好设置（如果偏好设置包含隐含或非直接的值，这个类型可以将可能的值映射为字符串）。 这个类型的键是 <code>PSTitleValueSpecifier</code> 。
拨 动 开 关	拨动开关类型显示一个 ON/OFF 拨动按钮，适用于配置值为二选一的偏好设置。 这个类型通常用于表示包含布尔值的偏好设置，但也可以用于表示包含非布尔值的偏好设置。 这个类型的键是 <code>PSToggleSwitchSpecifier</code> 。
滑块	滑块类型显示一个滑块控件，适用于值为一个范围的偏好设置。这个类型的值是一个实数，值的最小和最大值由您来指定。 这个类型的键是 <code>PSSliderSpecifier</code> 。
值列表	值列表类型使用户可以从一个值的列表中选择其一，适用于支持多个互斥值的偏好设置，这些值的类型可以是任意的。 这个类型的键是 <code>PSMultiValueSpecifier</code> 。
组	组类型使您可以将几组不同的偏好设置组织到一个页面上。组类型并不表示一个可配置的偏好设置，而只是包含一个标题字符串，显示在一或多个可配置的偏好设置之前。 这个类型的键是 <code>PSGroupSpecifier</code> 。
子页面	子页面类型使用户可以访问新的偏好设置页面，适用于实现多层次的偏好设置。 有关如何配置和使用这个类型的更多信息，请参见 “多层次的偏好设置” 。这个类型的键是 <code>PSChildPaneSpecifier</code> 。

各种偏好设置类型的详细格式信息请参见 [Settings 程序的结构参考](#)。如果要了解如何创建和编辑 Setting 程序的页面文件，则请参见[“添加和修改 Settings 程序包”](#)部分。

Settings 程序包

在 iPhone OS 中，开发者通过一种特殊的 **Settings 程序包**来指定应用程序的偏好设置，这种程序包命名为 `Settings.bundle`，驻留在应用程序程序包的顶级目录上。该程序包中包含一或多个 **Settings** 页面文件，用于定义应用程序偏好设置的详细信息；还可以包含显示偏好设置需要的其它支持文件，比如图像或本地化文件。表9-2列出了一个典型 **Settings** 程序包的内容。

Settings.bundle 目录下的内容项目名称	描述
Root.plist	这个 Settings 页面文件包含根页面的偏好设置，它的内容在 “Settings 页面文件的格式” 部分有更详细的描述。
其它.plist 文件	如果您需要通过多个子面板来构建一组有层次结构的偏好设置，则每个子面板的内容都分别存储在不同的 Settings 页面文件中。您需要负责命名这些文件，并将它们关联到正确的子面板上。
一或多个.lproj 目录	这些目录用于存储 Settings 页面文件的本地化字符串资源。每个目录都包含一个字符串文件，文件的标题在 Settings 页面中指定。这些字符串文件为偏好设置提供可以直接显示给用户的本地化内容。
其它图像	如果您使用滑块控件，则可以将滑块的图像存储在程序包的顶级目录下。

除了 Settings 程序包之外，应用程序的程序包中还可以包含应用程序设置的定制图标。如果应用程序包的顶级目录含有名为 Icon-Settings.png 的文件，则该文件包含的图标会被 Settings 程序用于标识应用程序的偏好设置。如果不存在这样的文件，Settings 程序会转而采用应用程序的图标文件（缺省为 Icon.png），并进行必要的缩放处理。您的 Icon-Settings.png 文件必须是29 x 29像素的图像。

在启动时，Settings 程序会检查每一个定制的应用程序是否包含 Settings 程序包，并对其进行装载，然后将相应的应用程序名称和图标显示在 Settings 程序的主页面上。当用户轻触您的应用程序对应的行时，Settings 程序会装载 Settings 程序包的 Root.plist 页面文件，并根据该文件的定义显示应用程序的主设置页面。

除了装载程序包的 Root.plist 页面文件之外，Settings 程序还会在必要时装载与该文件相关联的语言资源。每个 Settings 页面文件都可以有一个关联的.strings 文件，用于包含可见字符串的本地化值。在准备显示偏好设置信息时，Settings 程序会根据用户偏好的语言来寻找相应的字符串资源，并在显示之前替换偏好设置页面中对应的内容。

Settings 页面文件的格式

Settings 程序包中的每个 Settings 页面文件都以 iPhone 设置[属性列表](#)的文件格式（它是一种结构化的文件格式）进行存储。编辑 Settings 页面文件的最简单方法，就是使用 Xcode 内置的编辑器组件，具体做法请参见[“为 Settings 页面的编辑做准备”](#)部分；您也可以用属性列表编辑器程序来进行编辑，它是 Xcode 的工具之一。

请注意：在连编时，Xcode 会将工程中基于 XML 的属性文件自动转换为二进制格式，转换过程是连编时自动完成的，目的是节省磁盘空间。

每个 Settings 页面文件的根元素都包含表9-3列出的键。事实上，只有一个键是必须的，但我们推荐包含所有的两个键。

表9-3 Settings 页面文件中的根键键	类型	值
PreferenceSpecifiers (必须包含)	数组	这个键的值是一个 字典数组 ，数组中的每个字典都包含一个偏好设置元素的信息。有关元素类型列表请参见 表9-1 ，与元素类型相关联的键的描述，则请参见 Settings 程序的结构参考 。
StringsTable	字符串	和这个页面文件相关联的字符串文件的名称。程序包中专用于语言的工程目录应该包含这个字符串文件的一个拷贝（带有相应的

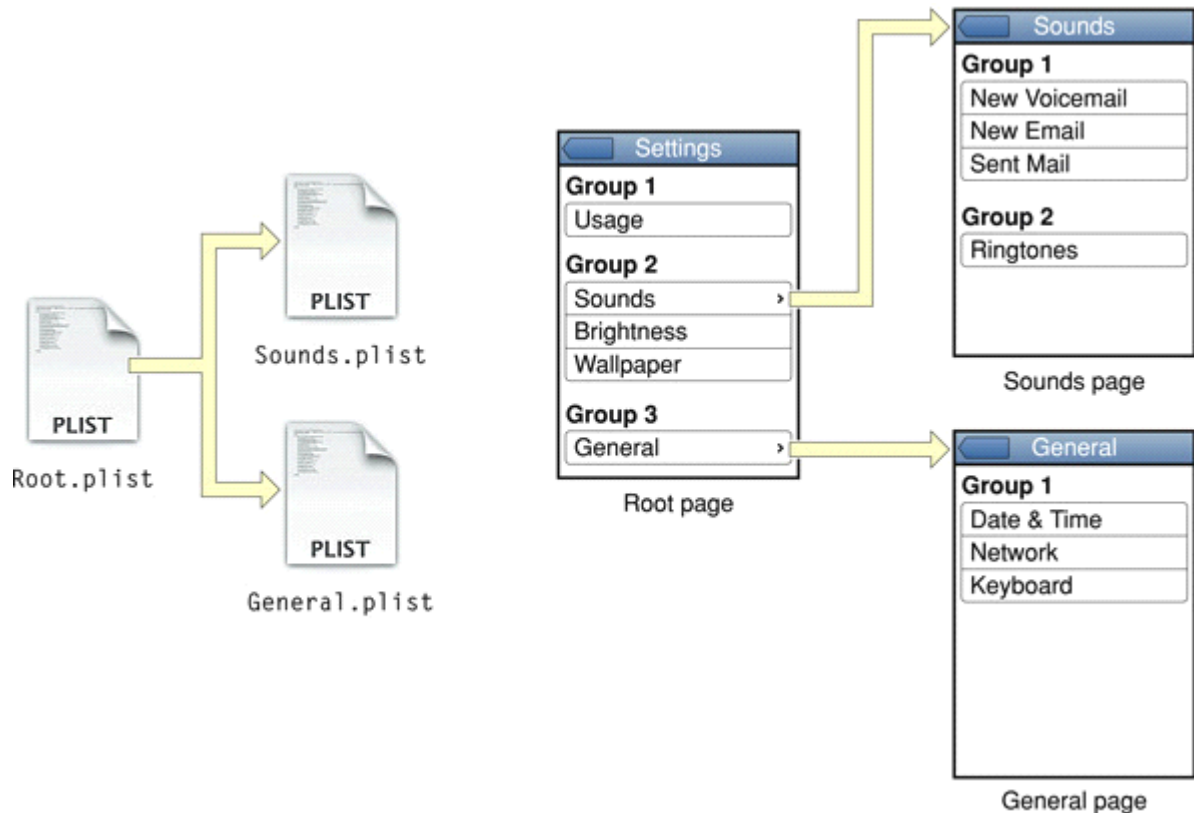
串 本地化字符串)。如果您没有包含这个键,则表示页面文件中的字符串没有被本地化。有关如何使用这些字符串的信息,请参见[“本地化资源”](#)部分。

多层次的偏好设置

如果您希望以一定的层次结构组织偏好设置,则您定义的每个页面都必须有它自己的.plist文件,每个.plist文件包含一组仅在该页面显示的偏好设置。应用程序偏好设置的主页面总是存储在 Root.plist 文件中,其它页面则可以根据自己的喜好进行命名。

为了建立父子页面之间的连接,您需要在父页面中包含一个子面板元素。子面板元素负责占据一行,在用户触击时显示一个新的设置 Settings 页面。子面板元素的 File 键标识一个.plist文件的名称,该文件负责定义子页面的内容;Title 键则标识子页面的标题,该标题也作为子面板元素行的文本。**Settings** 程序会自动提供子页面的漫游控制,使用户可以回到父页面。图9-1展示了一组多层次的页面是如何工作的。图的左边显示了.plist文件,右边则显示各个页面之间的关系。

图9-1 用子面板组织偏好设置



有关子面板元素及其关联键的更多信息,请参见[Settings 程序的结构参考](#)。

本地化资源

由于偏好设置中包含用户可见的字符串,所以您应该在 **Settings** 程序包中为那些字符串提供本地化版本。对于程序包支持的每种本地化语言,偏好设置页面都可以有一个.strings文件与之对应。当 **Settings** 程序碰到一个支持本地化的键时,就会在相应本地化版本的.strings文件中寻找匹配的键,如果找到了,就显示与之关联的值。

在寻找诸如.strings 文件这样的本地化资源时，Settings 程序遵循和 Mac OS X 程序一样的规则，即首先寻找与用户偏好语言相匹配的本地化资源，如果该版本的资源不存在，再选择缺省语言的版本。

有关字符串文件的格式、语言工程目录、以及如何从程序包中取得特定语言资源的相关信息，请参见[国际化编程主题](#)。

添加和修改 Settings 程序包

Xcode 提供了一个为当前工程添加 Settings [程序包](#)的模板。缺省的 Settings 程序包中包含一个 Root.plist 文件，以及一个用于存放本地化资源的缺省语言目录。您可以在这个基础上进行扩展，加入 Settings 程序包需要的其它属性列表文件和资源。

添加 Settings 程序包

通过如下步骤可以为您的 Xcode 工程添加一个 Settings 程序包：

选择 File > New File.

选择 iPhone OS > Settings > Settings Bundle template.

将文件命名为 Settings.bundle.

除了在工程中添加一个新的 Settings 程序包之外，Xcode 还自动将该程序包加入到应用程序目标的 Copy Bundle Resources 连编阶段中。这样，您需要做的就只是修改 Settings 程序包中的[属性列表](#)文件和添加其它资源了。

新添加的 Settings.bundle 程序包具有如下结构：

Settings.bundl

e/

Root.plist

en.lproj/

Root.strings

为 Settings 页面的编辑做准备

用 Settings 程序包模板创建 Settings 程序包之后，您可以将结构文件（schema file）的内容进行格式化，使它们更容易编辑。下面的步骤向您展示如何格式化 Settings 程序包的 Root.plist 文件，这些步骤同样适用于您创建的其它结构文件。

显示 Settings 程序包中 Root.plist 文件的内容。

在 Groups & Files 列表中，展开 Settings.bundle，查看程序包的内容。

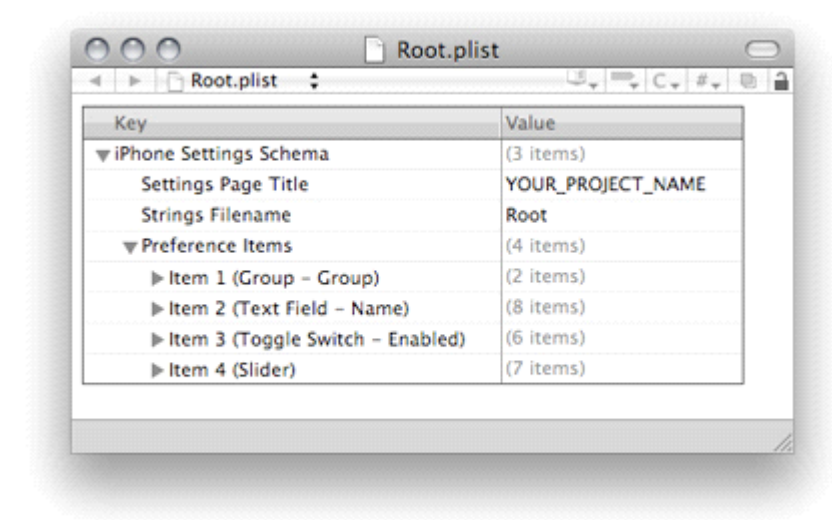
选择 Root.plist 文件，其内容就会显示在 Detail 视图中。

在 Detail 视图中，选择 Root.plist 文件的 Root 键。

选择 View > Property List Type > iPhone Settings plist.

这个命令会将 Detail 视图中的属性列表内容进行格式化。Xcode 不是直接显示属性列表的键和值，而是将它们显示为可读的字符串（如图9-2所示），使我们更加易于理解和编辑文件的内容。

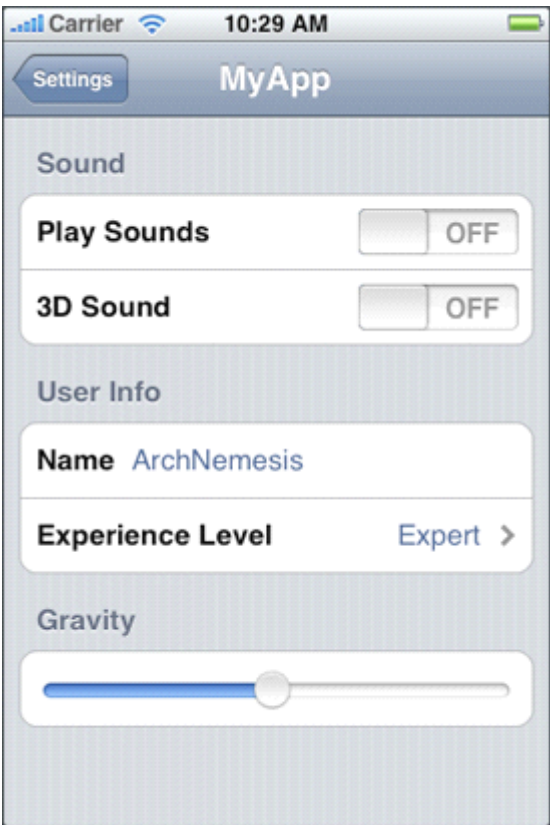
图9-2 格式化过的 Root.plist 文件内容



配置一个 **Settings** 页面：一个教程

这个部分包含一个教程，目的是向您展示如果配置一个 **Settings** 页面，使它显示您需要的内容。教程的目标是创建一个像图9-2这样的页面，如果您之前还没有为自己的工程创建 **Settings** 程序包，则在执行下面这些步骤之前，应该按照[“为 **Settings** 页面的编辑做好准备”](#)部分的描述进行准备。

图9-3 一个根 **Settings** 页面



将 **Settings Page Title** 键的值改为您的应用程序名称。

双击 YOUR_PROJECT_NAME 文本并将它改为 MyApp。

展开 Preference Items 键，显示模板包含的缺省项目。

将 Item 1的标题改为 Sound:

展开 Preference Items 的 Item 1。

将 Title 键的值由 Group 改为 Sound。

保持 Type 键的值不变，仍然为 Group。

为新命名的 Sound 组创建第一个拨动开关。

选中 Preference Items 的 Item 3项，并选择 Edit > Cut 命令。

选中 Item 1， 并选择 Edit > Paste 命令（这会将拨动开关项移到文本框项的前面）。

展开拨动开关项，显示其配置键。

将 Title 键的值改为 Play Sounds。

将 Identifier 键的值改为 play_sounds_preference。现在，这个项目的配置应该如下图所示：

▼ Item 2 (Toggle Switch – Play Sounds) ▼	(6 items)
Type	Toggle Switch
Title	Play Sounds
Identifier	play_sounds_preference
Default Value	<input checked="" type="checkbox"/>
Value for ON	YES
Value for OFF	NO

为 Sound 组创建第二个拨动开关。

选中 Item 2（即 Play Sounds 拨动开关）。

选择 Edit > Copy 命令。

选择 Edit > Paste 命令， 将拨动开关的拷贝放到第一个的下面。

展开新的拨动开关项，显示其配置键。

将其 Title 键的值改为3D Sound。

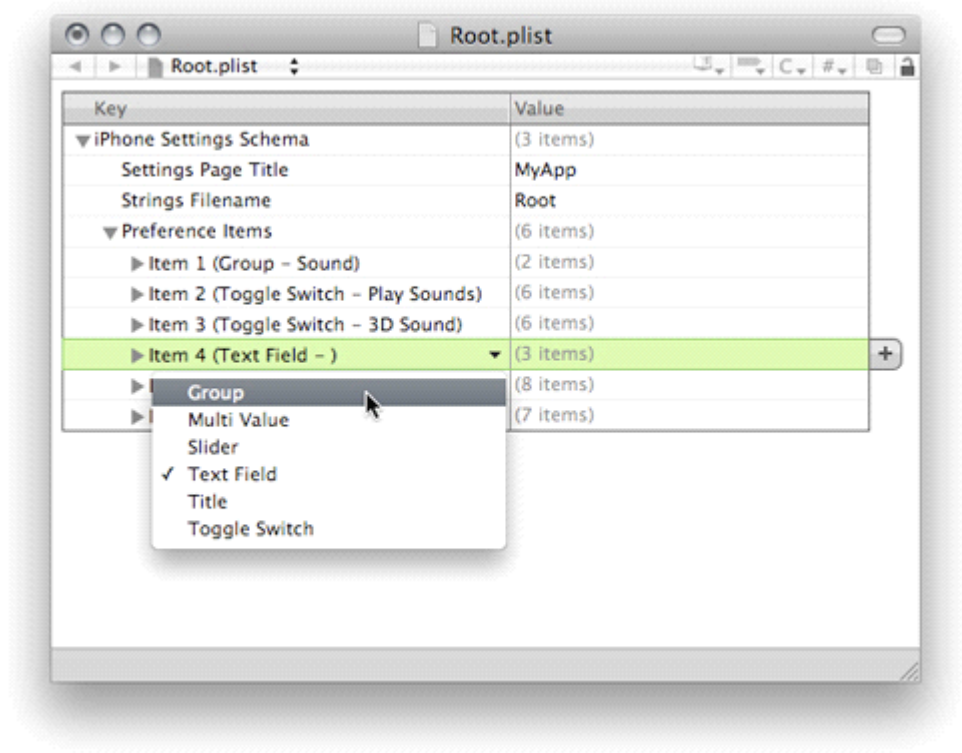
将其 Identifier 键的值改为3D_sound_preference。

现在，您已经完成了第一组设置，可以开始创建 User Info 组了。

将 Item 4改为 Group 类型的元素，并命名为 User Info。

在 Preferences Items 中点击 Item 4， 显示一个项目类型列表的下拉菜单。

从下拉菜单中，选择 Group 元素类型。



展开 Item 4的内容。

将 Title 键的值设置为 User Info。

创建 Name 域。

选择 Preferences Item 中的 Item 5。

使用下拉菜单，将其类型改为 Text Field。

将 Title 键的值改为 User Info。

将 Identifier 键的值改为 user_name。

合上展开按键，隐藏这个项目的内容。

创建 Experience Level 设置。

选择 Item 5并点击加号 (+) 键 (或者按下回车键)，创建一个新的项目。

点击这个新创建的项目，将其类型设置为 Multi Value。

展开项目的内容，将其标题设置为 Experience Level，标识设置为 experience_preference，缺省值设置为0。

选中 Default Value 键，点击加号键加入一个 Titles 数组。

通过展开键打开 Titles 数组，点击表格右侧的项目按键。点击这个键可以为 Titles 添加一个新的子项目。

- 选中新添加的子项目，点击两次加号键，创建总共三个子项目。

将子项目的值设置为 Beginner、Expert、和 Master。

再次选择 Titles 键，点击其展开键，将子项目隐藏起来。

点击加号键，创建 Values 数组。

在 Values 数组中加入三个子项目，将它们的值分别设置为0、1、和2。

点击 Item 6的展开按键，隐藏其内容。

添加设置页面的最好一组。

创建一个新项目，将其类型设置为 Group，标题设置为 Gravity。

- 再次创建一个新项目，将其类型设置为 Slider，标识设置为 gravity_preference，缺省值设置为1，最大值设置为2。

创建额外的 Settings 页面文件

Settings 程序包模板包含一个 Root.plist 文件，用于定义应用程序的顶级 Settings 页面。您如果要定义额外的 Settings 页面，必须在 Settings 程序包中加入额外的属性列表文件，您可以在 Finder 或 Xcode 中进行添加。

在 Xcode 中为 Settings 程序包添加属性列表的步骤如下：

在 Groups & Files 面板中，打开 Settings 程序包，选中 Root.plist 文件。

选择 File > New 命令。

选择 Other > Property List 命令。

选中新生成的文件，并选择 View > Property List Type > iPhone Settings plist 命令，将它配置为一个设置文件。

往 Settings 程序包加入新的 Settings 页面之后，就可以按照[“配置一个 Settings 页面：一个教程”](#)部分描述的那样，在页面中显示设置。您必须通过一个子面板元素对其进行引用，详情请参见[“多层次的偏好设置”](#)部分的描述。

访问您的偏好设置

iPhone 应用程序可以通过 Foundation 或者 Core Foundation [框架](#)来读写偏好设置的值。在 Foundation 框架中，您可以通过 NSUserDefaults 类来读写偏好设置的值；而在 Core Foundation 框架中，您则可以使用几个与偏好设置相关的函数。

程序清单 9-1 展示一个如何在应用程序中读取偏好设置的简单实例，例子中通过 NSUserDefaults 类取得一个在[“配置一个 Settings 页面：一个教程”](#)部分中创建的偏好设置值，并将它赋值给应用程序的一个实例变量。

程序清单9-1 访问应用程序偏好设置的值

```
- (void)applicationDidFinishLaunching:(UIApplication *)application
{
    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
    [self                                setShouldPlaySounds:[defaults
boolForKey:play_sounds_preference]];

    // Finish app initialization...
}
```

有关 NSUserDefaults 类中用于读写偏好设置值的方法的更多信息，请参见 [NSUserDefaults 类参考](#)；有关读写偏好设置的 Core Foundation 函数，请参见[偏好设置工具参考](#)。

在仿真器中调试应用程序的偏好设置

在运行您的应用程序时，iPhone Simulator 会将所有偏好设置的值保存在~/Library/Application

Support/iPhone Simulator/User/Applications/(<APP_ID>)/Library/Preferences 目录下，这里的 <APP_ID>是一个由程序生成的目录名，iPhone OS 用它来标识您的应用程序。

每次重新安装应用程序时，iPhone OS 都会执行一次干净的安装，将之前所有的偏好设置删除。换句话说，在 Xcode 中连编或运行应用程序会导致老版本的所有内容被新版本所代替。如果您要测试应用程序在两次运行之间偏好设置发生的变化，则必须直接从仿真器界面上运行，而不应该通过 Xcode 运行。