介绍

iOS 操作系统是 iPhone、 iPod touch 以及 iPad 设备的核心。

构建 iOS 平台的知识与 Mac OS X 系统同出一辙，iOS 平台的许多开发工具和开发技术也源自 Mac OS X。但开发者无须具备 Mac OS X 开发经验就可以编写 iOS 应用程序。iPhone 软件开发包（SDK）为着手创建 iOS 应用程序提供所需要的一切。

谁应阅读本文档?

iOS 技术概述是针对iOS平台新手的介绍指南。它简要介绍一些对开发过程有影响的技术和工具，并且提供相关文档和其他信息来源链接。通过阅读本文档，您将可以：

- 熟悉iOS平台。
- 了解iOS软件技术、知道为何使用这些技术以及何时使用。
- 了解该平台的发展机遇。
- 获得从其他平台转移到 iOS平台用到的技巧和指南。
- 找到和您感兴趣的技术相关的关键文档。

本文档只提供和软件开发过程有关的信息，不提供和用户级别的系统功能有关的信息。

本文档非常适合开发新手用于熟悉iOS平台。有经验的开发人员可以将本文档作为路线图，根据它来钻研某些特定技术或开发技术。

本文档的组织方式

本文档包含下述章节和附录：

- “ iOS开发相关信息”概要介绍iOS平台以及如何使用iPhone SDK进行iOS应用程序开发。
- “Cocoa Touch 层” 介绍 iOS的 Cocoa Touch层以及它为应用程序提供的功能。
- “Media层” 介绍iOS的Media层以及它为应用程序提供的功能。
- “Core Services层” 介绍iOS的Core Service层以及它为应用程序提供的功能。
- “Core OS 层” 介绍iOS的Core OS层以及它为应用程序提供的功能。
- “将Cocoa应用软件移植到iOS平台” 为希望将现有Cocoa应用程序移植到iOS平台的开发者提供一些起步建议。
- “iOS框架 ” 对开发软件使用到的框架进行描述。您可以通过这些信息找到其他您所感兴趣的技术，或者了解某个框架是在何时被引入iOS平台。
- “iOS开发者工具”概要介绍您在创建iOS软件时可以使用的工具。

获取iPhone SDK

iPhone SDK包含设计、创建、调试以及优化iOS软件所需要的工具。同时，它也包含一些头文件、样例代码以及平台技术文档。您可以从iPhone开发中心的会员区下载iPhone SDK，您可以通过下面的链接进入会员区http://www.apple.com.cn/developer/。

如需更多MAC OS X可用工具及技术的信息, 请阅读 “iOS开发者工具”

提供反馈

如希望为该文档提供反馈，请使用内建于页面底部的反馈表格。

我们提倡您向苹果公司报告在使用苹果软件或者文档过程中遇到的错误。您也可以向我们提交增强功能请求，告知我们某个产品或文档未来版本应具备什么功能。苹果开发者网站的错误报告页面可供您提交错误报告或增强功能的请求：

http://developer.apple.com/bugreporter/

您必须注册成为苹果开发者才能提交错误报告。按照苹果开发者注册页面的指导，您可以免费获得一个登陆名称。

其他参考资料

下述文档提供和 iOS开发相关的关键信息：

- Cocoa基础指南 为iOS应用程序开发使用的设计模式和实践提供起步信息。
- iOS 应用程序编程指南 提供iOS应用程序的架构概览及创建iOS应用程序的实践准则。
- iPhone人机接口指南 和 iPad人机接口指南 为如何设计应用程序的用户接口提供重要的信息。
- iOS开发指南 从工具方面为iOS开发过程提供重要信息。从配置设备到使用Xcode（以及其他工具）构建、运行并测试软件，俱都涵盖其中。
- Objective-C编程语言 介绍Objective-C以及Objective-C运行时系统。Objective-C运行时系统是许多动态行为和iOS 扩展性的基础。
-

Cocoa Touch 层

Cocoa Touch层包含创建 iOS应用程序所需的关键框架。上至实现应用程序可视界面，下至与高级系统服务交互，都需要该层技术提供底层基础。在开发应用程序的时候，请尽可能不要使用更底层的框架，尽可能使用该层的框架。

高级特性

下面章节描述一些常见特性，也许您也正打算在应用程序支持这些特性。

多任务

如果应用程序构建于iPhone SDK 4.0及其后续版本（且运行于iOS 4.0及后续版本操作系统），则点击Home键的时候，应用程序不会结束，而是切换到后台。对于大多数应用程序来说，进入后台，它们就会进入挂起状态。让应用程序驻留在后台可以避免以后的重新启动过程，应用程序可以直接将自己激活，这在很大程度上改善了整体用户体验。另外，将应用程序挂起也可以改善系统性能，因为挂起应用程序可以最小化电能使用，并可让前台应用程序获得更多的执行时间。

尽管应用程序进入后台就会被挂起，但可以通过下述的技术让其在后台继续运行：

- 应用程序可以请求一定的时间完成某些重要的任务。
- 应用程序可以声明自身支持的某种服务需要获得定期后台执行时间。
- 应用程序可以使用本地通告在指定时间向用户发通知。这种方式对于应用程序是否运行没有要求。

不管应用程序是被挂起还是在后台运行，支持多任务不需要付出额外的工作。但是在某些情况（例如内存不足）下，应用程序可能会被结束运行。因此，应用程序应该可以在任何时候退出。这就意味着许多在退出应用程序时需要执行的任务必须改为在应用程序切换到后台的时候执行。这就要求您在应用程序委托中实现一些新的方法以响应程序的状态切换。

如果需要进一步了解如何对后台状态切换进行处理以及如何让应用程序在后台继续运行，请查看iOS应用程序编程指南。

数据保护

和敏感用户数据打交道的应用程序可以使用设备内建加密功能（有些设备可能不提供内建加密功能）对数据进行保护。如果应用程序指定某个文件受保护，系统会以加密格式将该文件保存在磁盘。当设备锁住的时候，您的应用程序以及其他潜在的闯入者都不能访问该文件，而当用户解锁设备后，系统会生成一份密钥以便您的应用程序就访问该文件。

如需实现数据保护，应用程序应在待保护数据的创建和管理方式下一些功夫。它必须能够在数据创建之时保证其安全，而且需要适配设备上锁或未上锁造成的文件可访问性的变化。

如需进一步了解应用程序如何为文件添加数据保护，请查看iOS应用程序编程指南中的实现标准的应用程序行为一章。

苹果推送通知服务

iOS 3.0及后续版本的系统中，不管应用程序是否运行，苹果推送通知服务可用于通知用户某个应用程序具有新信息。利用这项服务，您可以向系统推送文本通知，可以触发声音提醒或者在应用程序图标上添加一个数字化标记。这样用户就知道他们应该打开应用程序接收相关信息。

从设计角度看，让应用程序支持推送通知包含两个部分。首先，iOS应用程序需要请求系统向其发送通知，然后要合理配置应用程序委托使其可以对通知进行恰当处理。这些工作可以通过应用程序委托以及`UIApplication`对象合作完成。第二，您需要提供一个服务器端进程用于产生最初的通知。该进程运行在您自己的本地服务器，它和苹果推送通知服务协同工作以产生最初的通知。

如需进一步了解如何配置应用程序以便使用远程通知，请查看本地通知及推送通知编程指南。

本地通知

iOS 4.0引入了本地通知。本地通知是对已有推送通知的补充，通用程序可以通过它在本地生成通知，不再需要依赖外部服务器。当有重要的事件发生时，后台应用程序可以利用本地通知获得用户关注。举个例子，运行于后台的导航应用程序可以使用本地通知提醒用户要转弯。应用程序也可以安排在未来的某个时刻向用户发送本地通知，而且发送这些通知并不要求应用程序处于运行状态。

本地通知的优点是它独立于您的应用程序。一旦某个通知被安排好后，系统会负责通知发送。而且在发送通知的时候，您的应用程序无需处于运行状态

如需进一步了解本地通知的使用方式，请查看本地通知及推送通知编程指南。

手势识别器

iOS 3.2引入了手势识别器。手势识别器是一个绑定到视图的对象，用于检测常见的手势类型。将手势识别器绑定到视图后，您可以告诉它某个手势发生的时候执行何种动作。之后，手势识别器就可以对原始事件进行跟踪，根据系统定义的试探方式识别手势。在引入手势识别器前，如果要识别一个手势，您需要跟踪视图的原始触摸事件流，然后再使用复杂的试探方法来判断这些事件是否表示某种手势。

现在，UIKit框架中包含一个`UIGestureRecognizer` 类，它定义了所有手势识别器的基本行为。您可以使用自定义的手势识别器子类或者系统定义的某个子类处理下面这些标准手势：

- 拍击 (任意次数的拍击)
- 向里或向外捏 (用于缩放)
- 摇动或者拖拽
- 擦碰 (以任意方向)
- 旋转 (手指朝相反方向移动)
- 长按

如需进一步了解您可以使用的手势识别器，请查看iOS事件处理指南。

文件共享支持

应用程序可以使用文件共享让用户访问程序的用户数据文件。文件共享允许应用程序通过iTunes向用户显露应用程序`/Documents`目录的内容。这样，用户就可以在iPad和桌面计算机来回移动文件。但是，该功能不允许应用程序和同一设备上的其他应用程序共享文件。如果希望在程序间共享文件，请使用剪贴板或者文档交互控制器对象。

您可以通过如下步骤来让应用程序支持文件共享：

1. 在应用程序的Info.plist文件中添加UIFileSharingEnabled键，并将键值设置为YES。
2. 将您希望共享的文件放在应用程序的Documents目录。
3. 一旦设备插入到用户计算机，iTunes 9.1就会在选中设备的Apps标签中显示一个File Sharing区域。
4. 此后，用户就可以向该目录添加文件或者将文件移动到桌面计算机中。

如果应用程序支持文件共享，当文件添加到Documents目录后，应用程序应该能够识别并做出适当响应。例如说，应用程序可以将新文件的内容显示界面上。请不要向用户展现目录的文件列表并询问他们希望对文件执行什么操作。

如需进一步了解 UIFileSharingEnabled键，请查看信息属性列表参考。

点对点服务

在iOS 3.0 及后续版本，Game Kit框架支持经由蓝牙进行点对点连接。您可以使用点对点连接启动与某个邻近设备的通讯会话，也可用它实现多种多玩家游戏的常见特性。虽然点对点连接主要应用于游戏，但您也可将之应用于其他类型应用程序。

如需进一步了解如何在应用程序中使用点对点连接的特性，请查看Game Kit 编程指南。如果需要概要了解Game Kit框架，请查看“Game Kit 框架”。

标准系统视图控制器

Cocoa Touch层许多框架含有展现标准系统界面的视图控制器。我们提倡您在应用程序中使用这些视图控制器，这样可以让您的程序和系统具有一致的用户体验。如果您需要执行下述任务，请从相应的框架中选择一个视图控制器使用:

- **显示或者编辑联系人信息** – 请使用Address Book UI框架中的视图控制器。
- **创建或者编辑日历事件** – 请使用Event Kit UI框架的视图控制器。
- **创建email或者SMS消息** – 请使用Message UI框架中的视图控制器。
- **打开或者预览一份文件的内容** – 请使用UIKit框架中的UIDocumentInteractionController类。
- **从用户的照片库选取一张照片** – 请使用UIKit框架中的 UIImagePickerController 类。
- **拍摄视频片段** – 请使用UIKit框架中的UIImagePickerController类。

如需了解如何选择使用视图控制器，请查看iOS视图控制器编程指南。如需了解特定视图控制器所展现的界面，请查看相应的框架参考。

外部设备支持

安装iOS 3.2的设备可通过一组设备支持的线缆连接一个外部显示设备。当外部设备连上后，应用程序就可以使用其显示屏显示内容。屏幕的信息，包括屏幕能够支持的分辨率都可通过UIKit框架的接口获取。您也可以通过该框架将应用程序窗口关联到某个屏幕。

- UIScreen 类用于获取当前所有屏幕（包括设备主屏幕）的屏幕对象。屏幕对象包含屏幕的属性信息（包括将屏幕的尺寸和像素比例也考虑在内的屏幕特征）。
- 您可以从UIScreenMode 类获得某个具有特定尺寸像素比例的屏幕的信息。
- 您可以将窗口(使用UIWindow类来表示)指定到某个特定的屏幕。如果您需要对内容作镜像显示，则需要提供两个独立的窗口，然后再在其中显示相同内容。

如需进一步了解上述类可提供的支持，请查看UIKit框架参考中相应的类描述。

Cocoa Touch 层包含的框架

下面部分描述Cocoa Touch层包含的框架以及这些框架提供的服务。

Address Book UI 框架

Address Book UI 框架(AddressBookUI.framework)是一套Objective-C的编程接口，可以显示创建或者编辑联系人的标准系统界面。该框架简化了应用程序显示联系人信息所需的工作，另外它也可以确保应用程序使用的界面和其他应用程序相同，进而保证跨平台一致性。

如果需要进一步了解Address Book UI框架以及如何使用该框架，请查看iOS地址簿编程指南以及iOS地址簿框架参考。

Event Kit UI 框架

iOS 4.0引入了Event Kit UI框架（EventKitUI.framework），它提供一个视图控制键可以展现查看并编辑事件的标准系统界面。Event Kit框架（查看“Event Kit框架”可获得该框架的进一步信息）的事件数据是该框架的构建基础。

如需进一步了解Event Kit UI框架类和方法，请查看Event Kit UI框架参考。

Game Kit 框架

iOS 3.0引入了**Game Kit**框架(GameKit.framework)。该框架支持点对点连接及游戏内语音功能，您可以通过该框架为应用程序增加点对点网络功能。点对点连接以及游戏内语音功能在多玩家的游戏中非常普遍，不过您也可以考虑将其加入到非游戏应用程序。此框架通过一组建构于Bonjour之上的简单而强大的类提供网络功能，这些类将许多网络细节抽象出来，从而让没有网络编程经验的开发者可以更加容易地将网络功能整合到应用程序。

如需进一步了解Game Kit框架， 请查看Game Kit 编程指南以及Game Kit 框架参考 。

iAd 框架

iOS 4.0引入了iAd框架 (iAd.framework)。您可以通过该框架在应用程序中发布横幅广告。广告会被放入到标准视图，您可以将这些视图加入到用户界面，并在合适的时机向用户展现。这些视图和苹果的公告服务相互协作，自动处理广告内容的加载和展现，同时也可以响应用户对广告的点击。

如需进一步了解如何在应用程序当中使用iAd，请查看iAd 框架参考。

Map Kit 框架

iOS 3.0导入了 **Map Kit**框架 (MapKit.framework) ，该框架供一个可被嵌入到应用程序的地图界面，该界面包含一个可以滚动的地图视图。您可以在视图中添加定制信息，并可将其嵌入到应用程序视图，通过编程的方式设置地图的各种属性（包括当前地图显示的区域以及用户的方位）。您也可以使用定制标注或标准标注（例如使用测针标记）突出显示地图中的某些区域或额外的信息。

在iOS 4.0系统中，该框架开始支持可拖动标注以及定制覆盖层。可拖动标注允许您通过编程方式或通过用户交互方式重定位某个标注的位置。覆盖层可用于创建多个点组成的复杂地图标注。地图表面诸如公交路线、选举地图、公园边界或者气象信息（例如雷达数据）等可以使用覆盖层进行显示。

如需进一步了解Map Kit框架中的类，请查看Map Kit 框架参考。

Message UI 框架

iOS 3.0引入了Message UI框架 (MessageUI.framework)。您可以利用该框架撰写电子邮件，并将其放入到用户的发件箱排队等候发送。该框架提供一个视图控制器界面，您可以在应用程序中展现该界面，让用户通过该界面撰写邮件。界面的字段可以根据待发送信息的内容生成。例如您可

以设置接收人、主题、邮件内容并可以在邮件中包含附件。这个界面允许用户先对邮件进行编辑，然后再选择接受。在用户接受邮件内容后，相应的邮件就会放入用户的发件箱排队等候发送。

在iOS 4.0及其后续的系统中，该框架提供一个SMS撰写面板控制器。您可以通过它在应用程序中直接创建并编辑SMS信息（无需离开应用程序）。和电子邮件撰写界面一样，该界面也允许用户先编辑SMS信息再发送。

如需进一步了解Message UI框架中的类，请参考Message UI 框架参考。

UIKit 框架

UIKit框架 (`UIKit.framework`)的Objective-C编程接口为实现iOS应用程序的图形及事件驱动提供关键基础。iOS系统所有程序都需要通过该框架实现下述核心功能：

- 应用程序管理
- 用户界面管理
- 图形和窗口支持
- 多任务支持
- 处理触摸及移动事件。
- 代表标准系统视图和控件的对象
- 文本和web内容相关操作
- 剪切、复制以及粘贴
- 使用动画显示用户界面内容
- 通过URL方式将其他应用程序整合到系统
- 苹果推送通知服务支持，请查看“苹果推送通知服务 ”
- 为残疾用户提供辅助功能
- 本地通知的调度和发送
- 创建PDF
- 使用定制输入视图（其行为类似系统键盘）
- 创建和系统键盘进行交互的定制文本视图

除了链编至应用程序的基础代码，UIKit还为下述和设备紧密相连的功能提供支持：

- 加速器数据
- 内置相机（存在相机的设备）
- 用户的图片库
- 设备名称和模型信息
- 电池状态信息
- 距离感应器信息
- 来自绑定听筒的远程控制信息

如需进一步了解UIKit框架的信息，请查看UIKit 框架参考。

媒体层

媒体层包含图形技术、音频技术和视频技术，这些技术相互结合就可为移动设备带来最好的多媒体体验，更重要的是，它们让创建外观音效俱佳的应用程序变得更加容易。您可以使用iOS的高级框架更快速地创建高级的图形和动画，也可以通过底层框架访问必要的工具，从而以某种特定的方式完成某种任务。

图形技术

高质量的图形是iOS应用程序的重要组成部分。创建应用程序最简单最有效的方法是使用事先渲染过的图片，搭配上标准视图以及UIKit框架的控件，然后把绘制任务交给系统来执行。但是在某些情况下，您可能需要一些UIKit所不具有的功能，而且需要定制某些行为。在这种情况下，您可以使用下述技术管理应用程序的图形内容：

- Core Graphics (也被称为Quartz)，用于处理本地2D向量渲染和图片渲染。
- Core Animation (Quartz Core框架的一部分)，为动画视图和其他内容提供更高级别支持。
- OpenGL ES，为使用硬件加速接口的2D和3D渲染提供支持。
- Core Text，提供一个精密的文本布局和渲染引擎。
- Image I/O，提供读取及编写大多数图形格式的接口。
- 资产库框架（Assets Library framework），可用于访问用户照片库中的照片和视频。

大多数应用程序应该无需改动，或者只需做很少修改，便可运行在具备高分辨率屏幕的设备。因为在绘图或者操作视图的时候，您所指定的座标值会被映射到逻辑座标系统，它和底层屏幕分辨率没有关联。而且绘制的内容会自动根据需要按比例缩放，以此来支持高分辨率屏幕。对基于向量进行绘制的代码来说，系统框架会自动使用额外的像素来改善图画的内容，使其变得更清晰。如果应用程序中使用了图片，则可以利用UIKit自动加载现有图片的高分辨率版本。如您需进一步了解如何支持高分辨率屏幕，请查看iOS应用程序编程指南中的“支持高分辨率屏幕”。

如需进一步了解图像相关框架，请查看”媒体层框架“中相应的内容。

音频技术

iOS音频技术可帮助您为用户提供丰富多彩的音响体验。您可以使用音频技术来播放或录制高质量的音频，也可以用于触发发设备的震动功能（具有震动功能的设备）。

iOS系统提供数种播放或录制音频的方式供您选用。在选择音频技术的时候，请记住，要尽可能地选取高级框架，因为它们可以简化播放音频所需的工作。下面列出的框架从高级到低级排列，，媒体播放器框架（Media Player　framwork）提供的是最高级的接口：

- 媒体播放器框架。该框架可以让访问用户的iTume库变得很容易，并且支持播放曲目和播放列表。
- AV Foundation框架。它提供一组简单易用的Objective-C接口，可用于管理音频的播放或录制。.
- OpenAL框架。它提供一组跨平台，用于发布方位音频的接口。
- Core Audio框架。它提供的接口简单而精密，可用于播放或录制音频内容。您可以使用这些接口播放系统的警报声音、触发备的震动功能、管理多声道的缓冲和播放、对音频内容进行流化处理。

iOS音频技术支持下述音频格式：

- AAC
- Apple Lossless (ALAC)
- A-law
- IMA/ADPCM (IMA4)

- Linear PCM
- μ-law
- DVI/Intel IMA ADPCM
- Microsoft GSM 6.10
- AES3-2003

如需进一步了解上述音频框架，请查看"媒体层框架"中相应的内容。

视频技术

iOS有数种技术可用于播放应用程序包的电影文件以及来自网络的数据流内容。如果设备具有合适的视频硬件，这些技术也可用于捕捉视频，并可将捕获到的视频集成到应用程序。

系统提供也提供多种方法用于播放或录制视频内容，您可以根据需要选择。选择视频技术的时候，请尽可能选择高级框架，因为高级框架可以简化为提供对某种功能的支持所需的工作。下面列出的框架由高级到低级排列。其中，媒体播放器框架提供最高级的接口：

- 媒体播放器框架，它提供一组易于使用的接口，可用于播放应用程序中全屏或部分屏的电影。
- AV Foundation框架，它提供一组Objective-C接口，可以对电影的捕捉和播放进行管理。
- Core Media框架，它对较高级框架使用的底层类型进行描述，同时也提供一些底层接口，它们用于对媒体进行处理。

iOS视频技术支持播放的电影文件应具有.mov、.mp4、.m4v以及.3gp文件扩展名，而且文件应使用下述的压缩标准：

- H.264视频，多达1.5 Mbps， 640x480像素，每秒30帧。 H.264 Baseline Profile 的 Low-Complexity 版本支持 AAC-LC 音频（ .m4v、.mp4以及.mov文件格式中高达160Kbps ,48KHz 的立体音频）。
- H.264视频，高达68 Kbps，320x240像素，每秒30帧。达到Level 1.3的 Baseline Profile支持AAC-LC音频（ .m4v、.mp4以及.mov文件格式中高达160Kbps ,48KHz 的立体音频）。
- MPEG-4视频，高达 2.5 Mbps， 640 x 480像素，每秒30帧。Simple Profile支持AAC-LC 音频（ .m4v、.mp4以及.mov文件格式中高达160Kbps ,48KHz 的立体音频）。
- 各种音频格式，包括 "音频技术"列出的清单。

如需进一步了解上述视频框架，请查看"媒体层框架"中相应的内容。

媒体层包含的框架

后续部分对媒体层的框架和框架所提供的服务进行说明。

资产库框架

iOS 4.0引入了资产库框架(AssetsLibrary.framework)，该框架提供一个查询界面，您可以通过它查找用户照片和数据。通过使用该框架，您可以访问Photos管理的资产，包括用户保存的相册以及导入到设备中的图片或视频。而且您也可以将照片或者视频保存到用户的相册。

如需进一步了解该框架，请查看资产库框架参考。

AV Foundation 框架

iOS 2.2引入了AV Foundation 框架 (AVFoundation.framework)，该框架包含的Objective-C 类可用于播放音频内容。通过使用该框架，您可以播放声音文件或播放内存中的音频数据，也可以同时播放多个声音，并对各个声音的播放特定进行控制。

在 iOS 4.0及后续版本中，该框架提供的服务得到很大的扩展，下述的服务现在也包含在框架中：

- 媒体资产管理
- 媒体编辑
- 电影捕捉
- 电影播放
- 曲目管理
- 媒体项的元数据管理
- 立体声淘选
- 不同声音的精确同步
- 用于判断声音文件详细信息的Objective-C接口，例如判断数据格式、采样率和声道数。

AV Foundation框架是iOS中录制播放音频和视频的唯一框架，该框架还支持对媒体项进行管理和处理。

如需进一步了解AV Foundation框架，请查看AV Foundation 框架参考。

Core Audio

表3-1列出来的Core Audio框架家族为音频提供本地支持。**Core Audio**框架提供C语言接口，可用于操作立体声音频。通过iOS系统Core Audio框架，您可以在应用程序中生成、录制、混合或播放音频，您也可通过该框架访问设备的震动功能（支持震动功能的设备）。

表 3-1 Core Audio框架

框架	服务
CoreAudio.framework	定义Core Audio框架家族使用的音频数据类型。
AudioToolbox.framework	播放或录制音频文件或数据流，也可用于管理音频文件、播放系统警告声音、触发某些设备的震动功能。
AudioUnit.framework	为内置音频单元服务，内置音频单元是指音频处理模块。

如需进一步了解Core Audio，请参考Core Audio 概述。如需了解如何使用Audio Toolbox 框架播放声音，请查看音频队列服务编程指南r以及Audio Toolbox框架参考。

Core Graphics 框架

Core Graphics框架 (CoreGraphics.framework)包含Quartz 2D绘图API接口 。**Quartz** 是Mac OS X系统使用的向量绘图引擎，它支持基于路径绘图、抗锯齿渲染、渐变、图片、颜色、坐标空间转换、PDF文件的创建、显示和解析。虽然API基于C语言，但是它使用基于对象的抽象以表示

基本绘图对象，这样可以让开发者可以更方便地保存并复用图像内容。

如需进一步了解如何使用Quartz绘制内容，请查看 Quartz 2D 编程指南 以及Core Graphics框架参考 。

Core Text 框架

iOS 3.2引入了Core Text框架(`CoreText.framework`)，该框架包含一组简单高效的C接口，可用于对文本进行布局以及对字体进行处理。Core Text框架提供一个完整的文本布局引擎，您可以通过它管理文本在屏幕上的摆放。所管理的文本也可以使用不同的字体和渲染属性。

该框架专为诸如字处理程序这类需要具有精密文本处理功能的应用程序而设计。如果您的应用程序只需要一种文本输入和显示，则应使用UIKit框架中已有的类。

如需进一步了解 Core Text接口的使用方式，请参考Core Text 编程指南和Core Text 参考集。

Core Video 框架

iOS 4.0引入了 Core Video框架 (`CoreVideo.framework`) ，该框架为Core Media提供缓存和缓存池的支持。大多数应用程序都不应该直接使用该框架。

Image I/O 框架

iOS 4.0引入 Image I/O 框架 (`ImageIO.framework`)，该框架的接口可用于导入或导出图像数据及图像元数据。该框架建构于 Core Graphics数据类型和函数之上，能够支持iOS 上所有的标准图像类型。

如需进一步了解该框架的数据类型和函数，请查看mage I/O 参考集。

媒体播放器框架

媒体播放器框架 (`MediaPlayer.framework`)为应用程序播放视频和音频内容提供高级支持。通过该框架，您就可以使用标准系统界面播放视频。iOS 3.0增加了对访问用户iTune库的支持。因此，您可以利用该框架播放音乐曲目、播放列表、搜索歌曲并向用户显示媒体选取界面。

在 iOS 3.2系统中，该框架发生了变化，开始支持在可改变尺寸的视图中播放视频（之前只支持全屏）。另外还新增数个界面用于支持配置和管理电影播放。

如需进一步了解媒体播放器框架中的类，请查看媒体播放器框架参考 。如需了解如何使用该框架访问用户的 iTunes库，请查看iPod 库访问编程指南。

OpenAL 框架

除了Core Audio之外，，iOS 还支持 **Open Audio Library (OpenAL)**。OpenAL接口是在应用程序中发布方位音频的跨平台标准。通过使用该框架，您可以在游戏或者要求有方位音频输出的程序中实现高性能、高质量的音频。OpenAL是跨平台的标准，iOS平台使用OpenAL编写的代码模块可以移植到许多其他的平台运行。

如需了解OpenAL及其使用方式，请查看http://www.openal.org。

OpenGL ES 框架

OpenGL ES框架 (`OpenGLES.framework`)提供的工具可用于绘制2D及3D内容。该框架基于C语言，能够和设备硬件紧密协作，为全屏游戏类型的应用程序提供很高的帧速率。

OpenGL框架需要和EAGL接口结合使用。这些接口是 OpenGL ES 框架的一部分，它们是OpenGL ES绘图代码及应用程序中的窗口对象的接口。

在 iOS 3.0及其后续版本的系统中，OpenGL ES 框架同时支持 OpenGL ES 2.0 及OpenGL ES 1.1 接口规范。2.0规范支持分段和点着色，只有运行iOS 3.0及其后续版本的设备才支持2.0。所有版本的iOS及iOS设备都支持OpenGL ES 1.1规范。

如需了解如何在应用程序中使用OpenGL ES，请查看iOS OpenGL ES编程指南。如果需要参考信息，请查看OpenGL ES框架参考。

Quartz Core 框架

Quartz Core框架 (`QuartzCore.framework`)包含Core Animation接口。**Core Animation**是高级动画制作和混合技术，它使用经过优化的渲染路径实现复杂的动画和视觉效果。它提供的高级Objective-C接口可对动画效果进行配置，然后在设备硬件中进行渲染，以此来提高程序的性能。Core Animation框架被整合到iOS的许多部分（包括UIKit框架中的许多类（如`UIView`）），可以为多种系统行为提供动画效果。您也可以使用该框架中的Objective-C接口直接创建定制动画。

如需进一步了解如何在应用程序中使用Core Animation，请查看Core Animation Programming Guide and Core Animation参考集。

Core Services 层

Core Services层为所有的应用程序提供基础系统服务。可能应用程序并不直接使用这些服务，但它们是系统很多部分赖以建构的基础。

高阶特性

下面的部分描述一些比较常见特性，也许您正打算让您的应用程序支持这些特性。

块对象

iOS 4.0引入了块对象。块对象是C级别的构造，您可以在C或Objective-C代码中使用快对象。从本质上说，块对象本质上是一个匿名函数加上该函数的伴随数据。有些时候，其他语言也称块对象为 closure或者lambda。块对象非常适用于回调函数。如果您需要有很便捷的方法将执行代码和相关数据组合在一起，块对象也是很好的选择。

在 iOS系统中，块对象通常用于下述场合：

- 作为委托或委托方法的替代品。
- 作为回调函数的替代品。
- 用于实现一次性操作的完成处理器。
- 简化在群体所有子项上迭代执行某种任务的操作。
- 配合分发队列。可用于执行异步任务。

如需了解块对象及其使用方式，请查看块对象简短实践指南 。如需进一步了解块对象， 请查看块对象编程论述 。

Grand Central Dispatch

iOS 4.0引入了 Grand Central Dispatch (GCD)，它是BSD级别的技术，可用于在应用程序内管理多个任务的执行。GCD技术将异步编程模型和高度优化内核结合在一起，可作为多线程的便捷（且更高效）替代。同时，它也为许多种底层任务（ 例如读写文件描述符、实现定时器、监视信号和处理事件等）提供替代方案。

如需进一步了解在应用程序内使用GCD的方式，请查看并发编程指南 。如果需要了解特定GCD函数的信息，请查看Grand Central Dispatch (GCD)参考 。

应用程序内购买（In App Purchase）

iOS 3.0引入了应用程序内购买功能。通过该功能，您可以在应用程序内出售内容或服务。该功能使用Store Kit框架来实现，它可以为使用iTunes账户进行的财务交易的处理提供基础支持，应用程序只需处理用户体验及待售内容或服务的展现。

如果需要进一步了解iOS如何支持应用程序内购买功能，请参考应用程序内购买编程指南 。如需进一步了解 Store Kit框架,请参考"Store Kit框架" 。

定位服务

应用程序可使用Core Location框架提供的接口追踪用户位置。此框架利用当前可用的硬件无线电波（包括Wi-Fi、蜂窝无线或者GPS）定位用户的当前位置。应用程序可以对框架提供的信息进行裁剪，然后再将其发送给客户， 或是用于实现某些特定功能。举个例子，社交应用程序允许您找到附近其他应用程序用户，然后再与之进行通讯。

如需进一步了解如何使用定位服务，请阅读方位感知编程指南 。如需进一步了解Core Locatio框架，请阅读"Core Location框架 "。

SQLite

SQLite库允许开发者将一个轻量级SQL数据库嵌入到应用程序，而且开发者不需要运行独立的远程数据库服务器进程。在此之后，开发者可以在应用程序中创建本地数据库文件，管理文件中的表和记录。虽然SQLite数据库出于通用目的而设计，但它还是针对数据库记录的快速访问做过优化。

用于访问SQLite库的头文件位于 <iOS_SDK>`/usr/include/sqlite3.h`。 在该路径中， <iOS_SDK>是Xcode安装目录中目标SDK的路径。如果需要更多如何使用SQLite的信息， 请访问http://www.sqlite.org网站。

XML 支持

Foundation框架支持使用`NSXMLParser` 类从XML文档中解析元素，而`libXML2`库则为操作XML内容提供支持。 `libXML2`库是开源的，它可以让您快速地解析或写入任意的XML数据，也可将XML内容转化为HTML文件。

用于访问`libXML2`库的头文件位于 <iOS_SDK>`/usr/include/libxml2/`。 在该路径中， <iOS_SDK>是Xcode安装目录中目标SDK的路径。如要更多`libXML2`的使用信息， 请访问http://xmlsoft.org/index.htm 网站。

Core Services 框架

下述部分描述Core Services层的框架以及这些框架提供的服务。

Address Book 框架

Address Book框架 (`AddressBook.framework`)支持编程访问存储于用户设备中的联系人信息。如果应用程序使用到联系人信息，则可通过该框架访问并修改用户联系人数据库的记录。举个例子，通过使用该框架，聊天程序可以获取一个联系人列表，利用此列表初始化聊天会话，并在联系人视图显示列表的联系人。

如果需要进一步了解 Address Book框架的功能，请访问Address Book 框架参考 。

CFNetwork 框架

CFNetwork框架 (`CFNetwork.framework`)提供一组高性能基于C语言的接口，它们为使用网络协议提供面向对象抽象。通过这些抽象，您可以对协议栈进行更精细的控制，而且可以使用诸如 BSD socket这类底层结构。您也可以通过该框架简化诸如与FTP或HTTP服务器通讯以及DNS主机解析这类任务。下面列举一些可以使用 CFNetwork框架执行的任务：

- 使用BSD sockets
- 使用SSL或TLS创建加密连接
- 解析DNS主机
- 使用HTTP，校验HTTP以及HTTPS服务器。
- 使用FTP服务器
- 发布、解析并浏览 Bonjour服务。

CFNetwork理论及实现都以 BSD socket为基础。如需更多如何使用CFNetwork框架的信息， 请访问CFNetwork编程指南以及CFNetwork框架参考。

Core Data 框架

iOS 3.0引入**Core Data框架(`CoreData.framework`)**。**Core Data框架**是一种管理模型-视图-控制器应用程序数据模型的技术，它适用于数据模型已经高度结构化的应用程序。通过此框架，您再也不需要通过编程定义数据结构，而是通过Xcode提供的图形工具构造一份代表数据模型的图表。在程序运行的时候，Core Data框架就会创建并管理数据模型的实例，同时还对外提供数据模型访问接口。

通过Core Data管理应用程序的数据模型，可以极大程度减少需编写的代码数量。除此之外，Core Data还具有下述特征：

- 将对象数据存储在SQLite数据库以获得性能优化。
- 提供`NSFetchedResultsController` 类用于管理表视图的数据。
- 管理undo/redo操作。
- 属性值校验支持。
- 支持对数据变化进行传播，并且不会改变对象间的关联。
- 支持对数据进行归类，过滤，并支持对内存数据进行管理。

如果您正在开发新应用程序或打算对某个现有的程序进行大幅度更新，请考虑使用Core Data。如果需要了解如何在iOS应用程序中使用Core Data，请参考iOS Core Data 教程 。如果需要进一步了解Core Data框架中的类，请参考Core Data框架参考 。

Core Foundation 框架

Core Foundation框架 (`CoreFoundation.framework`) 是一组C语言接口，它们为iOS应用程序提供基本数据管理和服务功能。下面列举该框架支持进行管理的数据以及可提供的服务：

- 群体数据类型 (数组、集合等)
- 程序包

- 字符串管理
- 日期和时间管理
- 原始数据块管理
- 偏好管理
- URL及数据流操作
- 线程和RunLoop
- 端口和soket通讯

Core Foundation框架和Foundation框架紧密相关，它们为相同功能提供接口，但Foundation框架提供Objective-C接口。如果您将Foundation对象和Core Foundation类型掺杂使用，则可利用两个框架之间的 “toll-free bridging”。所谓的**Toll-free bridging**是说您可以在某个框架的方法或函数同时使用Core Foundatio和Foundation 框架中的某些类型。很多数据类型支持这一特性，其中包括群体和字符串数据类型。每个框架的类和类型描述都会对某个对象是否为 toll-free bridged，应和什么对象桥接进行说明。

如需进一步信息，请阅读Core Foundation 框架参考。

Core Location 框架

Core Location框架 (`CoreLocation.framework`)可用于定位某个设备当前经纬度。它可以利用设备具备的硬件，通过附近的GPS、蜂窝基站或者WiFi信号等信息计算用户方位。Maps应用程序就是利用此功能在地图上显示用户当前位置。您可以将此技术结合到应用程序，以此向用户提供方位信息。例如，应用程序可根据用户当前位置搜索附近饭店、商店或其他设施 。

在iOS 3.0系统中，该框架开始支持访问iOS设备（具有相应硬件的设备）的方向信息。

在iOS 4.0系统中，该框架开始支持低能耗的方位监视服务，该服务利用蜂窝基站跟踪用户方位。

如需了解Core Location框架中的类，请参考Core Location 框架参考 。

Core Media 框架

iOS 4.0引入了Core Media框架 (`CoreMedia.framework`)。此框架提供AV Foundation框架使用的底层媒体类型。只有少数需要对音频或视频创建及展示进行精确控制的应用程序才会涉及该框架，其他大部分应用程序应该都用不上。

如需进一步了解此框架的函数和数据类型，请阅读Core Media 框架参考。

Core Telephony 框架

iOS 4.0引入了Core Telephony框架(`CoreTelephony.framework`)。此框架为访问具有蜂窝无线的设备上的电话信息提供接口，应用程序可通过它获取用户蜂窝无线服务的提供商信息。如果应用程序对于电话呼叫感兴趣，也可以在相应事件发生时得到通知。

如需进一步了解如何使用该框架的类和方法，请阅读Core Telephony框架参考。

Event Kit 框架

iOS 4.0引入了 Event Kit框架 (`EventKit.framework`)。此框架为访问用户设备的日历事件提供接口。您可以通过该框架访问用户日历中现有事件，可以增加新事件。日历事件可包含闹铃，而且可以配置闹铃激活规则。

如需进一步了解如何使用该框架的类和方法，请阅读Event Kit 框架参考 ，也可参考Event Kit UI 框架。

Foundation 框架

Foundation框架 (`Foundation.framework`)为 Core Foundation框架的许多功能提供Objective-C封装。您可以参考Core Foundation框架了解前面对Core Foundation框架的描述。 Foundation框架为下述功能提供支持：

- 群体数据类型 (数组、集合等)
- 程序包
- 字符串管理
- 日期和时间管理
- 原始数据块管理
- 偏好管理
- URL及数据流操作
- 线程和RunLoop
- Bonjour
- 通讯端口管理
- 国际化
- 正则表达式匹配
- 缓存支持

如需进一步了解如何使用该框架的类和方法，请阅读Foundation框架参考 。

Mobile Core Services 框架

iOS 3.0引入了**Mobile Core Services框架** (`MobileCoreServices.framework`)。此框架定义统一类型标识符 (UTIs)使用的底层类型。

如需进一步了解此框架定义的类型，请查看统一类型标识符参考 。

Quick Look 框架

iOS 4.0引入Quick Look框架(`QuickLook.framework`），应用程序可以用过该框架预览无法直接支持查看的文件内容。如果应用程序从网络下载文件或者需处理来源未知的文件，则非常适合使用此框架。因为应用程序只要在获得文件后，调用框架提供的视图控制器就可以直接在界面中显示文件的内容。

如需进一步了解该框架的类和方法，请参考Quick Look框架参考。

Store Kit 框架

iOS 3.0引入**Store Kit** 框架(`StoreKit.framework`)，此框架为iOS应用程序内购买内容或服务提供支持。例如，开发者可以利用此框架允许用户解锁应用程序的额外功能。或者假设您是一名游戏开发人员，则可使用此特性向玩家出售附加游戏级别。在上述的两种情况中，Store Kit 框架会处于交易过程中和财务相关的事件，包括处理用户通过 iTunes Store账号发出的支付请求并且向应用程序提供交易相关信息。

Store Kit框架主要关注交易过程中和财务相关的事务，目的是为了确保交易安全准确。应用程序需要处理交易事物的其他因素，包括购买界面和下载（或者解锁）恰当的内容。通过这种任务划分方式，您就拥有购买内容的控制权，可以决定希望展示给用户的购买界面以及何时向用户展示这些界面，同时也可以决定和应用程序最匹配的交付机制。

如需进一步了解Store Kit框架的使用方式，请查看应用程序内购买编程指南以及Store Kit 框架参考 。

System Configuration 框架

System Configuration框架(SystemConfiguration.framework) 可用于确定设备的网络配置。您可以使用该框架判断Wi-Fi或者蜂窝连接是否正在使用中，也可以用于判断某个主机服务是否可以使用。

如需进一步了解此框架提供的接口，请查看System Configuration框架参考。如需通过此框架获取网络信息的示例，请访问 Reachability样例工程。

Core OS 层

Core OS层的底层功能是很多其他技术的构建基础。通常情况下，这些功能不会直接应用于应用程序，而是应用于其他框架。但是，在直接处理安全事务或和某个外设通讯的时候，则必须要应用到该层的框架。

Accelerate 框架

iOS 4.0引入了Accelerate框架（`Accelerate.framework`）。该框架的接口可用于执行数学、大数字以及DSP运算。和开发者个人编写的库相比，该框架的优点在于它根据现存的各种iOS设备的硬件配置进行过优化。因此，您只需一次编码就可确保它在所有设备高效运行。

如需要进一步了解Accelerate框架，请查看Accelerate框架参考。

External Accessory 框架

iOS 3.0引入了External Accessory框架（`ExternalAccessory.framework`），通过它来支持iOS设备与绑定附件通信。附件可以通过一个30针的基座接口和设备相连，也可通过蓝牙连接。通过External Accessory框架，您可以获得每个外设的信息并初始化一个通讯会话。通讯会话初始化完成之后，您可以使用设备支持的命令直接对其进行操作。

如需进一步了解External Accessory框架的使用方式，请查看External Accessory编程概论。如需了解External Accessory框架中相关类的信息，请查看External Accessory框架参考。如需了解如何开发iOS设备附件，请访问 http://www.apple.com.cn/developer/页面。

Security 框架

iOS系统不但提供内建的安全功能，还提供**Security**框架（`Security.framework`）用于保证应用程序所管理之数据的安全。该框架提供的接口可用于管理证书、公钥、私钥以及信任策略。它支持生成加密的安全伪随机数。同时，它也支持对证书和Keychain密钥进行保存，是用户敏感数据的安全仓库。

CommonCrypto接口另外还支持对称加密、HMAC以及Digests。实际上，Digests的功能实和OpenSSL库常用的功能兼容，但是iOS无法使用OpenSSL库。

在 iOS 3.0及其后续版本的系统中， 您可以让所创建的多个应用程序共享某些Keychain项，这样可以让相同套件内的应用程序的互用更流畅。举个例子，您可以在应用程序间共享用户密码和及其他元素。通过这种方法，您就不需要在每个应用程序单独对用户作出提示。如应用程序需要共享数据，则每个应用程序的Xcode工程必须配备恰当的资格。

如需要进一步了解Security框架的功能和特征，请查看Security 框架参考 。如需了解如何访问 Keychain，请查看Keychain服务编程指南。如需了解如何在Xcode工程中设置应用程序的资格，请查看iOS 开发指南。如需了解您可以对哪些应用程序资格进行配置，请查看Keychain服务参考中的`SecItemAdd`函数。

System

系统层包括内核环境、驱动及操作系统底层UNIX 接口。内核以Mach为基础，它负责操作系统的各个方面，包括管理系统的虚拟内存、线程、文件系统、网络以及进程间通讯。这一层包含的驱动是系统硬件和系统框架的接口。出于安全方面的考虑，内核和驱动只允许少数系统框架和应用程序访问。

应用程序可以使用iOS提供的`LibSystem`库访问多种操作系统底层功能。`LibSystem`库的接口基于C语言，可为下述功能提供支持：

- 线程 (POSIX 线程)
- 网络 (BSD sockets)
- 文件系统访问
- 标准 I/O
- Bonjour和 DNS服务
- 区域信息
- 内存分配
- 数学计算

许多 Core OS技术的头文件位于<iOS_SDK>`/usr/include/`目录，<iOS_SDK>是 Xcode安装目录中目标SDK的路径。如果需要了解这些技术相关联的功能，请访问iOS手册页面。

从 Cocoa 迁移到 iOS

如果您是一名Cocoa开发者，则会对iOS中的许多框架感到熟悉。虽然iOS的基础技术栈在许多方面都等同于Mac OS X，但有些框架则非如此。本章针对创建iOS应用程序可能碰到的差异进行描述，同时也说明如何应对某些重要变化：

请注意：本章专为已熟悉Cocoa术语和编程技术的开发者设计。如果您需要进一步了解Cocoa应用程序（以及iOS应用程序）使用的基本设计模式，请查看Cocoa基础指南。

通用的迁移注意事项

如果您的Cocoa应用程序已经使用模型-视图-控制器模式，则很容易将程序的关键部分迁移到iOS系统。如需了解如何设计iOS应用程序，请查看

iOS应用程序编程指南。

迁移数据模型

如果Cocoa应用程序的数据模型以Foundation或者Core Foundation框架的类为基础，则将数据模型迁移到iOS只需做少量修改，甚至无需修改。因为iOS也支持这两个框架。并且彼此差异不大。大多数差异要么相对次要，要么和iOS应用程序无法具有的功能相关。例如iOS应用程序所不支持AppleScript。如需此类差异的详细列表，请查看“Foundation框架差异”。

如果Cocoa应用程序使用Core Data，则只能将其数据模型迁移到iOS3.0及后续版本，因为早期的iOS不支持Core Data。iOS的Core Data框架支持二进制和SQLite数据存储（不支持ＸＭＬ数据存储），同时也支持从现有的Cocoa应用程序迁移数据。如果是受支持的数据存储，您可以把Core Data资源文件复制到iOS应用程序工程，然后就可以直接使用这些文件。如需进一步了解如何在Xcode工程中使用Core Data，请查看Core Data编程指南。

如果Cocoa应用程序在屏幕上显示很多数据，则在将其迁移到iOS的时候，您可能需要对数据模型进行简化。虽说iOS系统上也可创建数据量大且内容丰富的应用程序，但是这样做并不符合用户需求，因为移动用户通常希望在最短的时间里获得最重要的信息。另外，一次性提供给用户太多数据也行不通，毕竟屏幕的空间有限。而且这样做可能会让应用程序变慢，因为加载数据也需要额外的工作。如果对Cocoa的数据结构进行重构可以提供更好的性能及更优秀的iOS用户体验，那就非常值得一试。

迁移用户界面

iOS用户界面的结构及实现和Mac OS X有很大区别。以 Cocoa表示视图和窗口的对象为例。虽然iOS和Cocoa都有用于表示视图和窗口的对象，但是不同平台对象的工作方式稍有差异。另外，在iOS系统中，视图显示的内容需要更精心地挑选，因为屏幕尺寸有限，而视图又必须足够大以便给用户的手指提供足够多的操作目标。

除了视图对象自身的差异，在程序运行时，视图显示方式也有巨大差别。举个例子，如果您希望在Cocoa应用程序中显示很多数据，则可以增加窗口的尺寸，使用多个窗口或者使用标签窗口来管理数据。而在iOS应用程序中，只存在一个窗口且窗口的尺寸固定不变。因此，应用程序必须按合理尺寸对信息进行分块，并把数据块呈现在不同的视图。对信息进行分块是为了将其划分成多个屏幕内容，然后您可以根据屏幕内容设计相应的应用程序视图。举个例子，如果要在Cocoa显示分层列表数据，您可能会使用一个`NSBrowser`对象。但是在iOS系统中，您就需要创建一组大相径庭的视图用于显示不同层的信息。虽然这种方式会导致更复杂的界面设计，但是它的确是非常重要的显示信息的方式。因此，iOS系统为这种组织方式提供很多支持。

Mac OS X v10.5系统Cocoa才开始引入视图控制器，对于视图控制器的使用可能还未普及。但在iOS应用程序中，视图控制器是用户界面管理基础架构的关键部分。视图控制器管理用户界面的展现，而且它还与系统相互协作，可以保证应用程序的资源不会占用太多内存，因而可防止程序性能下降。总之，理解视图控制器的角色及其在应用程序的使用方式是设计用户界面的关键。

如需iOS通用的用户界面设计原则，请查看iPhone人机接口准则。如需了解用户界面的窗口和视图及其底层架构，请参考iOS 应用程序编程指南。如需了解视图控制器及其创建用户界面的方式，请查看iOS视图控制器编程指南。

内存管理

iOS系统不支持垃圾收集，您需要使用内存管理模型保持、释放或自动释放对象。

和Macintosh计算机相比， iOS设备内存非常有限。因此，您需调整自动释放池的使用，避免创建多个自动释放池对象。另外，请尽可能直接释放对象，不要自动释放。如果您在一个紧凑的循环中分配了很多对象，要么就直接释放那些对象，要么就在循环代码中的恰当的位置创建自动释放池，并在规则的间隔内释放自动释放对象。等到循环结束再释放可能会导致内存不足的警告或导致应用程序被系统杀死。

框架差异

虽然iOS的大多数框架同样存在于 Mac OS X系统，但不同平台框架具有不同的实现方式和使用方式。下面收集了一些Mac OS X 开发者开发iOS应用程序需要注意的重要差别：

UIKit 与 AppKit 的对比

在 iOS系统中，创建图形应用程序、管理事件循环以及执行其他界面相关的任务都离不开UIKit提供的基础结构。UIKit和AppKit具有非常显著的区别，在设计iOS应用程序的时候，应该特别注意这一点。也正是因为这个原因，在将Cocoa应用程序迁移到iOS系统的时候，您必须提供和界面相关的类和逻辑。表6-1列出了框架之间的特定的差异，它可帮助您理解iOS中的应用程序应该具有什么特征：

表 6-1 界面技术的差异

差异	讨论
文档支持	在iOS系统中，文档角色的重要性有所降低，简单内容模型则变的越来越重要。因为iOS系统的应用程序通常只拥有一个窗口（在不连接外部显示的情况下），主窗口是创建及编辑所有应用程序内容的唯一环境。更重要的是，所有和文档相关的操作，包括文件的创建和管理，现在都由应用程序在幕后完成，不再需要用户干预。
视图类	UIKit为您提供一组非常有针对性的视图和控件。AppKit框架有许多视图和控件无法在iOS设备上工作，其他一些视图则被更具iOS特色的视图替代。例如，在显示分层信息的时候，iOS不使用 `NSBrowser`类，而是使用完全不同的样式（导航控制器）。如需了解iOS中的视图和控件及其使用方式，请查看iPhone人机接口准则。
视图座标系统	iOS系统Quartz和UIKit内容的绘画模型和Mac OS X的基本相同，只有一处例外。在 Mac OS X绘画模型坐标系统中，窗口和视图的原点默认位于左下角，坐标轴向上向右延伸。但在iOS系统中，默认的原点位置是左上角，坐标轴向下向右延伸。Mac OS X的座标系统称为“被翻转”的座标系统，iOS则是缺省座标系统。如需进一步了解图形和座标系统，请查看iOS视图编程指南。
窗口即视图	从概念上来看，iOS系统的窗口和视图Mac OS X的具有相同含义。但从实现的角度来看，区别很大。在Mac OS X系统中，`NSWindow`类是`NSResponder`类的子类，但在iOS系统中，`UIWindow`实际是`UIView`的子类。继承关系上的改变表明窗口将会使用Core Animation层来绘制外表。之所以有这样的改变，主要是为了在操作系统级别支持窗口分层。举个例子，系统可以在一个独立的窗口中显示状态栏，并让该窗口浮动于应用程序窗口之上。 iOS系统和Mac OS X系统另外一个差异和窗口的使用方式相关。Mac OS X应用程序可以用于任意数量的窗口，但大多数iOS应用程序只能有一个窗口。在iOS应用程序中显示不同屏幕的数据不是通过改变窗口实现，而是通过在应用程序窗口中切换定制视图来完成。
事件处理	UIKit的事件处理模型和Mac OS X的事件处理模型区别很大。UIKit框架不向视图发送鼠标和键盘事件，而是发送触摸和移动事件。这些事件不但要求您实现一组不同的方法，同时也要求您修改整个事件处理代码。举个例子，本地跟踪循环的排队事件不能包含触摸事件，您的代码也据此做相应调整。如需进一步了解iOS应用程序的事件处理，请参考iOS事件处理指南。
目标–动作模型	UIKit支持三种形式的动作，AppKit仅支持一种。UIKit的控件可以在不同的交互阶段调用唤醒不同动作，而且一个交互过程可以指定多个目标。因此，在UIKit中，一个控件可以在一次交互过程中向多个目标发送多个不同的动作。如需进一步了解iOS应用程序的目标–动作模型，请查看iOS事件处理指南。

绘画及打印支持	为支持UIKit渲染需要，UIKit的绘画能力经过适当的调节。它支持图片的加载和显示、字符串显示、颜色管理、字体管理以及多个用于渲染矩阵和获取图形上下文的函数。UIKit不包含通用目的的绘图类，因为iOS系统使用其他方式完成此类功能（即Quartz和OpenGL ES）。 iOS系统部支持打印功能，iOS设备不能连接打印机或其他相关的打印硬件。 如需进一步了解图形和绘图方面的信息，请查看iOS视图编程指南。
文本支持	撰写电子邮件和记事本是iOS系统提供的主要的文本支持。UIKit类可以让应用程序显示并编辑简单的字符串和稍微复杂点的HMTL内容。 在iOS 3.2及后续系统中，Core Text框架 和UIKit框架提供更加精密的文本处理能力，您可以通过这它们实现更精密的文本编辑及展现视图，也可通过它们定制视图提供的输入方法。如需进一步了解文本支持相关的信息，请查看iOS文本和Web编程指南。
存取方法的使用和属性对比	UIKit在其类声明中大量使用属性。属性由Mac OS X在10.5版本引入，是AppKit框架大量的类创建出来以后才出现。属性不是对AppKit框架getter和setter方法的简单模仿，而是被UIKit用于简化类接口。如需了解属性的使用方式，请查看Objective-C编程语言中的“属性声明”。
控件和单元	UIKit控件不使用单元。单元被Mac OS X作为视图的轻量级替代物。但是UIKit视图本身就是非常轻量的对象，因此单元派不上用场。虽然在命名约定上，`UITableView`类也用到了单元这个词，但是此处的单元实际上是`UITableView`的子类。
表视图	iOS系统的 `UITableView` 类可以看成是`AppKit`框架中`NSTableView`和`NSOutlineView`的折中物。它结合Appkit框架中者两个类的特征，更适合在小屏幕上显示。`UITableView`一次显示一列数据，而且您将相关的行组合成一个区段。`UITableView`也可用于显示并编辑分层列表数据。如需进一步了解`UITableView`类，请查看UITableView类参考。
菜单	几乎所有iOS应用程序的命令集都比类似的Mac OS X应用程序小得多，因此，iOS不支持菜单，通常也用不到菜单。对于需要少数的命令的场合，使用工具栏或者一组按键更加合适。对于需要数据菜单的场合，使用拾取器或导航控制器界面通常更合适，而如需对上下文敏感的菜单，则可其中的菜单项显示在Edit菜单，用它们替代或补充剪切、复制或者粘贴等命令。
Core Animation层	在 iOS系统中，所有外表的绘制都由Core Animation层实现。该框架还隐式为许多视图相关属性提供的动画支持。由于这种内建的动画支持，您就不需要在代码中显示使用Core Animation层，只需更改一下视图的某些属性即可实现大多数动画。只有当需要对分层进行精确控制或者不想将某些特征暴露于视图层，您才需要直接使用Core Animation。如需了解将Core Animation层整合到iOS绘图模型的方式，请查看iOS视图编程指南。

如需了解UIKit的类信息，请查看UIKit框架参考。

Foundation 框架的差异

Mac OS X和 iOS都有Foundation框架，大多数您所预期的类都可以在该框架中找到。 两个平台的框架都支持数值管理、字符串、集合、线程以及许多其他常见的数据类型。表6-2列出一些不包含于iOS框架的重要功能以及相关类不存在的缘由，同时也尽量列出有哪些技术可作为替代。

表 6-2 iOS的Foundation不具有的技术

技术	注意事项
元数据和预测管理	iOS不支持Spotlight 元数据和搜索预测，因为iOS不支持Spotlight。
分布式对象和端口名称服务管理	iOS不存在分布式对象技术，但是您可以使用`NSPort`家族类和端口（及socket）进行交互交互。您也可以使用Core Foundation和CFNetwork框架处理网络需求。
Cocoa绑定	iOS不支持Cocoa绑定，而是使用经过少量修改的目标-动作模型。因为这种方式可以让代码对动作的处理方式有更多的灵活性。
Objective-C垃圾收集	iOS不支持垃圾收集，您必须使用内存管理模型。您需要通过保持对象来宣告对对象的拥有权，并在不需要对象的时候释放对象。
AppleScript支持	iOS不支持AppleScript。

iOS系统的Foundation框架提供对XML的支持，您可以通过 `NSXMLParser` 类解析XML文件，其他解析类（包括`NSXMLDocument`、`NSXMLNode`）不受支持。除了`NSXMLParser`之外，您还可以使用`libXML2`库，这是C语言的XML解析接口。

如果需要了解哪些类存在于Mac OS X而不存在于iOS，请查看位于Foundation框架参考中的“Foundation 框架”的类层次图。

其他框架的改变

表 6-3列出iOS其他框架的关键差异。

表 6-3 同时存在于iOS和Mac OS X的框架之间的差异

框架	差异
`AddressBook.framework`	该框架接口可用于访问用户的联系人信息。虽然名称相同，但是此框架的iOS版本和Mac OS X版本却有很大的区别。 在iOS系统中，除了访问联系人数据的C接口，您还可以使用Address Book UI框架提供的类展现标准联系人挑选和编辑界面。 如需进一步的信息，请查看Address Book框架参考。

`AudioToolbox.framework`
`AudioUnit.framework`
`CoreAudio.framework`
	在iOS系统中，这些框架支持音频录制、播放以及单声道和多声道的音频内容混合，但不支持更高级的音频处理功能和定制音频单元插件。不过iOS系统增加了一个功能，即触发iOS设备（具有相应硬件）的震动功能。如果需要了解如何使用音频支持，请查看iOS应用程序编程指南中的多媒体支持 。

`CFNetwork.framework`	该框架包含Core Foundation Network接口。在iOS系统中，CFNetwork框架是顶层框架，它没有子框架。该框架的接口大部分保持不变。如需进一步信息，请查看CFNetwork框架参考。
`CoreGraphics.framework`	该框架包含Quartz接口。在iOS系统中，Core Graphics框架是顶层框架，它没有子框架。使用Quartz创建路径、渐变、阴影、图案、图像以及位图的方式和Mac OS X系统完全相同。不过有一些Quartz的功能（包括PostScript支持、图像来源和去向、Quartz显示服务支持、Quartz事件服务支持）不存在于iOS系统。如需进一步信息，请查看Core Graphics框架参考。

`OpenGLES.framework`	OpenGL ES 是专为嵌入式系统设计的OpenGL版本。如果您是OpenGL开发人员，则应该会很熟悉OpenGL ES接口。不过，OpenGL ES接口还是有几点较大差别。首先，它是一套更加小巧的接口，仅支持可以在现有图形硬件有效执行的功能。第二，许多桌面OpenGL可以使用的扩展并不存在于OpenGL ES。虽然如此，您应该还是能够执行大多数和桌面OpenGL相同的操作。但如果你是在迁移现有的OpenGL代码，则可能需要重写一部分代码，需要使用iOS系统的渲染技术（不同于Mac OS X）。如需了解iOS对OpenGL ES的支持，请查看iOS OpenGL ES编程指南。
`QuartzCore.framework`	该框架包含Core Animation接口。iOS大部分 Core Animation接口和Mac OS X相同。但是iOS系统没有用于管理布局约束的类，也不支持使用Core Image过滤器。另外，iOS也没有Core Image和Core Video接口（两者都包含于Mac OS X版本的QuartzCore框架）。如需进一步信息，请查看Quartz Core框架参考。
`Security.framework`	该框架包含安全接口。在iOS系统中，该框架通过加解密、伪随机数生成以及Keychain保护应用程序数据安全。该框架不包含身份验证或身份验证接口，也不支持显示证书内容。Keychain接口也是Mac OS X版本的简化。如需了解iOS的安全支持，请查看iOS应用程序编程指南。
`SystemConfiguration.framework`	该框架包含和网络相关的接口。在iOS系统中，您可以使用这些接口来决定设备如何与网络连接，是通过EDGE、GPRS或是通过Wi-Fi。

从 Cocoa 迁移到 iOS

如果您是一名Cocoa开发者，则会对iOS中的许多框架感到熟悉。虽然iOS的基础技术栈在许多方面都等同于Mac OS X，但有些框架则非如此。本章针对创建iOS应用程序可能碰到的差异进行描述，同时也说明如何应对某些重要变化：

请注意：本章专为已熟悉Cocoa术语和编程技术的开发者设计。如果您需要进一步了解Cocoa应用程序（以及iOS应用程序）使用的基本设计模式，请查看Cocoa基础指南。

通用的迁移注意事项

如果您的Cocoa应用程序已经使用模型-视图-控制器模式，则很容易将程序的关键部分迁移到iOS系统。如需了解如何设计iOS应用程序，请查看iOS应用程序编程指南。

迁移数据模型

如果Cocoa应用程序的数据模型以Foundation或者Core Foundation框架的类为基础，则将数据模型迁移到iOS只需做少量修改，甚至无需修改。因为iOS也支持这两个框架。并且彼此差异不大。大多数差异要么相对次要，要么和iOS应用程序无法具有的功能相关。例如iOS应用程序所不支持AppleScript。如需此类差异的详细列表，请查看“Foundation框架差异”。

如果Cocoa应用程序使用Core Data，则只能将其数据模型迁移到iOS3.0及后续版本，因为早期的iOS不支持Core Data。iOS的Core Data框架支持二进制和SQLite数据存储（不支持ＸＭＬ数据存储），同时也支持从现有的Cocoa应用程序迁移数据。如果是受支持的数据存储，您可以把Core Data资源文件复制到iOS应用程序工程，然后就可以直接使用这些文件。如需进一步了解如何在Xcode工程中使用Core Data，请查看Core Data编程指南。

如果Cocoa应用程序在屏幕上显示很多数据，则在将其迁移到iOS的时候，您可能需要对数据模型进行简化。虽说iOS系统上也可创建数据量大且内容丰富的应用程序，但是这样做并不符合用户需求，因为移动用户通常希望在最短的时间里获得最重要的信息。另外，一次性提供给用户太多数据也行不通，毕竟屏幕的空间有限。而且这样做可能会让应用程序变慢，因为加载数据也需要额外的工作。如果对Cocoa的数据结构进行重构可以提供更好的性能及更优秀的iOS用户体验，那就非常值得一试。

迁移用户界面

iOS用户界面的结构及实现和Mac OS X有很大区别。以 Cocoa表示视图和窗口的对象为例。虽然iOS和Cocoa都有用于表示视图和窗口的对象，但是不同平台对象的工作方式稍有差异。另外，在iOS系统中，视图显示的内容需要更精心地挑选，因为屏幕尺寸有限，而视图又必须足够大以便给用户的手指提供足够多的操作目标。

除了视图对象自身的差异，在程序运行时，视图显示方式也有巨大差别。举个例子，如果您希望在Cocoa应用程序中显示很多数据，则可以增加窗口的尺寸，使用多个窗口或者使用标签窗口来管理数据。而在iOS应用程序中，只存在一个窗口且窗口的尺寸固定不变。因此，应用程序必须按合理尺寸对信息进行分块，并把数据块呈现在不同的视图。对信息进行分块是为了将其划分成多个屏幕内容，然后您可以根据屏幕内容设计相应的应用程序视图。举个例子，如果要在Cocoa显示分层列表数据，您可能会使用一个`NSBrowser`对象。但是在iOS系统中，您就需要创建一组大相径庭的视图用于显示不同层的信息。虽然这种方式会导致更复杂的界面设计，但是它的确是非常重要的显示信息的方式。因此，iOS系统为这种组织方式提供很多支持。

Mac OS X v10.5系统Cocoa才开始引入视图控制器，对于视图控制器的使用可能还未普及。但在iOS应用程序中，视图控制器是用户界面管理基础架构的关键部分。视图控制器管理用户界面的展现，而且它还与系统相互协作，可以保证应用程序的资源不会占用太多内存，因而可防止程序性能下降。总之，理解视图控制器的角色及其在应用程序的使用方式是设计用户界面的关键。

如需iOS通用的用户界面设计原则，请查看iPhone人机接口准则。如需了解用户界面的窗口和视图及其底层架构，请参考iOS 应用程序编程指南。如需了解视图控制器及其创建用户界面的方式，请查看iOS视图控制器编程指南。

内存管理

iOS系统不支持垃圾收集，您需要使用内存管理模型保持、释放或自动释放对象。

和Macintosh计算机相比， iOS设备内存非常有限。因此，您需调整自动释放池的使用，避免创建多个自动释放池对象。另外，请尽可能直接释放对象，不要自动释放。如果您在一个紧凑的循环中分配了很多对象，要么就直接释放那些对象，要么就在循环代码中的恰当的位置创建自动释放池，并在规则的间隔内释放自动释放对象。等到循环结束再释放可能会导致内存不足的警告或导致应用程序被系统杀死。

框架差异

虽然iOS的大多数框架同样存在于 Mac OS X系统，但不同平台框架具有不同的实现方式和使用方式。下面收集了一些Mac OS X 开发者开发iOS应用程序需要注意的重要差别：

UIKit 与 AppKit 的对比

在 iOS系统中，创建图形应用程序、管理事件循环以及执行其他界面相关的任务都离不开UIKit提供的基础结构。UIKit和AppKit具有非常显著的区

别，在设计iOS应用程序的时候，应该特别注意这一点。也正是因为这个原因，在将Cocoa应用程序迁移到iOS系统的时候，您必须提供和界面相关的类和逻辑。表6-1列出了框架之间的特定的差异，它可帮助您理解iOS中的应用程序应该具有什么特征：

表 6-1 界面技术的差异

差异	讨论
文档支持	在iOS系统中，文档角色的重要性有所降低，简单内容模型则变的越来越重要。因为iOS系统的应用程序通常只拥有一个窗口（在不连接外部显示的情况下），主窗口是创建及编辑所有应用程序内容的唯一环境。更重要的是，所有和文档相关的操作，包括文件的创建和管理，现在都由应用程序在幕后完成，不再需要用户干预。
视图类	UIKit为您提供一组非常有针对性的视图和控件。AppKit框架有许多视图和控件无法在iOS设备上工作，其他一些视图则被更具iOS特色的视图替代。例如，在显示分层信息的时候，iOS不使用 `NSBrowser`类，而是使用完全不同的样式（导航控制器）。如需了解iOS中的视图和控件及其使用方式，请查看iPhone人机接口准则。
视图座标系统	iOS系统Quartz和UIKit内容的绘画模型和Mac OS X的基本相同，只有一处例外。在 Mac OS X绘画模型坐标系统中，窗口和视图的原点默认位于左下角，坐标轴向上向右延伸。但在iOS系统中，默认的原点位置是左上角，坐标轴向下向右延伸。Mac OS X的座标系统称为“被翻转”的座标系统，iOS则是缺省座标系统。如需进一步了解图形和座标系统，请查看iOS视图编程指南。
窗口即视图	从概念上来看，iOS系统的窗口和视图Mac OS X的具有相同含义。但从实现的角度来看，区别很大。在Mac OS X系统中，`NSWindow`类是`NSResponder`类的子类，但在iOS系统中，`UIWindow`实际是`UIView`的子类。继承关系上的改变表明窗口将会使用Core Animation层来绘制外表。之所以有这样的改变，主要是为了在操作系统级别支持窗口分层。举个例子，系统可以在一个独立的窗口中显示状态栏，并让该窗口浮动于应用程序窗口之上。 iOS系统和Mac OS X系统另外一个差异和窗口的使用方式相关。Mac OS X应用程序可以用于任意数量的窗口，但大多数iOS应用程序只能有一个窗口。在iOS应用程序中显示不同屏幕的数据不是通过改变窗口实现，而是通过在应用程序窗口中切换定制视图来完成。
事件处理	UIKit的事件处理模型和Mac OS X的事件处理模型区别很大。UIKit框架不向视图发送鼠标和键盘事件，而是发送触摸和移动事件。这些事件不但要求您实现一组不同的方法，同时也要求您修改整个事件处理代码。举个例子，本地跟踪循环的排队事件不能包含触摸事件，您的代码也据此做相应调整。如需进一步了解iOS应用程序的事件处理，请参考iOS事件处理指南。
目标-动作模型	UIKit支持三种形式的动作，AppKit仅支持一种。UIKit的控件可以在不同的交互阶段调用唤醒不同动作，而且一个交互过程可以指定多个目标。因此，在UIKit中，一个控件可以在一次交互过程中向多个目标发送多个不同的动作。如需进一步了解iOS应用程序的目标-动作模型，请查看iOS事件处理指南。
绘画及打印支持	为支持UIKit渲染需要，UIKit的绘画能力经过适当的调节。它支持图片的加载和显示、字符串显示、颜色管理、字体管理以及多个用于渲染矩阵和获取图形上下文的函数。UIKit不包含通用目的的绘图类，因为iOS系统使用其他方式完成此类功能（即Quartz和OpenGL ES）。 iOS系统部支持打印功能，iOS设备不能连接打印机或其他相关的打印硬件。 如需进一步了解图形和绘图方面的信息，请查看iOS视图编程指南。
文本支持	撰写电子邮件和记事本是iOS系统提供的主要的文本支持。UIKit类可以让应用程序显示并编辑简单的字符串和稍微复杂点的HMTL内容。 在iOS 3.2及后续系统中，Core Text框架 和UIKit框架提供更加精密的文本处理能力，您可以通过这它们实现更精密的文本编辑及展现视图，也可通过它们定制视图提供的输入方法。如需进一步了解文本支持相关的信息，请查看iOS文本和Web编程指南。
存取方法的使用和属性对比	UIKit在其类声明中大量使用属性。属性由Mac OS X在10.5版本引入，是AppKit框架大量的类创建出来以后才出现。属性不是对AppKit框架getter和setter方法的简单模仿，而是被UIKit用于简化类接口。如需了解属性的使用方式，请查看Objective-C编程语言中的“属性声明”。
控件和单元	UIKit控件不使用单元。单元被Mac OS X作为视图的轻量级替代物。但是UIKit视图本身就是非常轻量的对象，因此单元派不上用场。虽然在命名约定上，`UITableView`类也用到了单元这个词，但是此处的单元实际上是`UITableView`的子类。
表视图	iOS系统的 `UITableView` 类可以看成是`AppKit`框架中`NSTableView`和`NSOutlineView`的折中物。它结合Appkit框架中者两个类的特征，更适合在小屏幕上显示。`UITableView`一次显示一列数据，而且您将相关的行组合成一个区段。`UITableView`也可用于显示并编辑分层列表数据。如需进一步了解`UITableView`类，请查看UITableView类参考。
菜单	几乎所有iOS应用程序的命令集都比类似的Mac OS X应用程序小得多，因此，iOS不支持菜单，通常也用不到菜单。对于需要少数的命令的场合，使用工具栏或者一组按键更加合适。对于需要数据菜单的场合，使用拾取器或导航控制器界面通常更合适，而如需对上下文敏感的菜单，则可其中的菜单项显示在Edit菜单，用它们替代或补充剪切、复制或者粘贴等命令。
Core Animation层	在 iOS系统中，所有外表的绘制都由Core Animation层实现。该框架还隐式为许多视图相关属性提供的动画支持。由于这种内建的动画支持，您就不需要在代码中显示使用Core Animation层，只需更改一下视图的某些属性即可实现大多数动画。只有当需要对分层进行精确控制或者不想将某些特征暴露于视图层，您才需要直接使用Core Animation。如需了解将Core Animation层整合到iOS绘图模型的方式，请查看iOS视图编程指南。

如需了解UIKit的类信息，请查看UIKit框架参考。

Foundation 框架的差异

Mac OS X和 iOS都有Foundation框架，大多数您所预期的类都可以在该框架中找到。 两个平台的框架都支持数值管理、字符串、集合、线程以及许多其他常见的数据类型。表6-2列出一些不包含于iOS框架的重要功能以及相关类不存在的缘由，同时也尽量列出有哪些技术可作为替代。

表 6-2 iOS的Foundation不具有的技术

技术	注意事项
元数据和预测管理	iOS不支持Spotlight 元数据和搜索预测，因为iOS不支持Spotlight。
分布式对象和端口名称服务管理	iOS不存在分布式对象技术，但是您可以使用`NSPort`家族类和端口（及socket）进行交互交互。您也可以使用Core Foundation和CFNetwork框架处理网络需求。

Cocoa绑定	iOS不支持Cocoa绑定，而是使用经过少量修改的目标-动作模型。因为这种方式可以让代码对动作的处理方式有更多的灵活性。
Objective-C垃圾收集	iOS不支持垃圾收集，您必须使用内存管理模型。您需要通过保持对象来宣告对对象的拥有权，并在不需要对象的时候释放对象。
AppleScript支持	iOS不支持AppleScript。

iOS系统的Foundation框架提供对XML的支持，您可以通过 `NSXMLParser` 类解析XML文件，其他解析类（包括`NSXMLDocument`、`NSXMLNode`）不受支持。除了`NSXMLParser`之外，您还可以使用`libXML2`库，这是C语言的XML解析接口。

如果需要了解哪些类存在于Mac OS X而不存在于iOS，请查看位于Foundation框架参考中的"Foundation 框架"的类层次图。

其他框架的改变

表 6-3列出iOS其他框架的关键差异。

表 6-3 同时存在于iOS和Mac OS X的框架之间的差异

框架	差异
`AddressBook.framework`	该框架接口可用于访问用户的联系人信息。虽然名称相同，但是此框架的iOS版本和Mac OS X版本却有很大的区别。 在iOS系统中，除了访问联系人数据的C接口，您还可以使用Address Book UI框架提供的类展现标准联系人挑选和编辑界面。 如需进一步的信息，请查看Address Book框架参考。

`AudioToolbox.framework`
`AudioUnit.framework`
`CoreAudio.framework`
	在iOS系统中，这些框架支持音频录制、播放以及单声道和多声道的音频内容混合，但不支持更高级的音频处理功能和定制音频单元插件。不过iOS系统增加了一个功能，即触发iOS设备（具有相应硬件）的震动功能。如果需要了解如何使用音频支持，请查看iOS应用程序编程指南中的多媒体支持 。

`CFNetwork.framework`	该框架包含Core Foundation Network接口。在iOS系统中，CFNetwork框架是顶层框架，它没有子框架。该框架的接口大部分保持不变。如需进一步信息，请查看CFNetwork框架参考。
`CoreGraphics.framework`	该框架包含Quartz接口。在iOS系统中，Core Graphics框架是顶层框架，它没有子框架。使用Quartz创建路径、渐变、阴影、图案、图像以及位图的方式和Mac OS X系统完全相同。不过有一些Quartz的功能（包括PostScript支持、图像来源和去向、Quartz显示服务支持、Quartz事件服务支持）不存在于iOS系统。如需进一步信息，请查看Core Graphics框架参考。
`OpenGLES.framework`	OpenGL ES 是专为嵌入式系统设计的OpenGL版本。如果您是OpenGL开发人员，则应该会很熟悉OpenGL ES接口。不过，OpenGL ES接口还是有几点较大差别。首先，它是一套更加小巧的接口，仅支持可以在现有图形硬件有效执行的功能。第二，许多桌面OpenGL可以使用的扩展并不存在于OpenGL ES。虽然如此，您应该还是能够执行大多数和桌面OpenGL相同的操作。但如果你是在迁移现有的OpenGL代码，则可能需要重写一部分代码，需要使用iOS系统的渲染技术（不同于Mac OS X）。如需了解iOS对OpenGL ES的支持，请查看iOS OpenGL ES编程指南。
`QuartzCore.framework`	该框架包含Core Animation接口。iOS大部分 Core Animation接口和Mac OS X相同。但是iOS系统没有用于管理布局约束的类，也不支持使用Core Image过滤器。另外，iOS也没有Core Image和Core Video接口（两者都包含于Mac OS X版本的QuartzCore框架）。如需进一步信息，请查看Quartz Core框架参考。
`Security.framework`	该框架包含安全接口。在iOS系统中，该框架通过加解密、伪随机数生成以及Keychain保护应用程序数据安全。该框架不包含身份验证或身份验证接口，也不支持显示证书内容。Keychain接口也是Mac OS X版本的简化。如需了解iOS的安全支持，请查看iOS应用程序编程指南。
`SystemConfiguration.framework`	该框架包含和网络相关的接口。在iOS系统中，您可以使用这些接口来决定设备如何与网络连接，是通过EDGE、GPRS或是通过Wi-Fi。

从 Cocoa 迁移到 iOS

如果您是一名Cocoa开发者，则会对iOS中的许多框架感到熟悉。虽然iOS的基础技术栈在许多方面都等同于Mac OS X，但有些框架则非如此。本章针对创建iOS应用程序可能碰到的差异进行描述，同时也说明如何应对某些重要变化：

请注意：本章专为已熟悉Cocoa术语和编程技术的开发者设计。如果您需要进一步了解Cocoa应用程序（以及iOS应用程序）使用的基本设计模式，请查看Cocoa基础指南。

通用的迁移注意事项

如果您的Cocoa应用程序已经使用模型-视图-控制器模式，则很容易将程序的关键部分迁移到iOS系统。如需了解如何设计iOS应用程序，请查看iOS应用程序编程指南。

迁移数据模型

如果Cocoa应用程序的数据模型以Foundation或者Core Foundation框架的类为基础，则将数据模型迁移到iOS只需做少量修改，甚至无需修改。因为iOS也支持这两个框架。并且彼此差异不大。大多数差异要么相对次要，要么和iOS应用程序无法具有的功能相关。例如iOS应用程序所不支持AppleScript。如需此类差异的详细列表，请查看"Foundation框架差异"。

如果Cocoa应用程序使用Core Data，则只能将其数据模型迁移到iOS3.0及后续版本，因为早期的iOS不支持Core Data。iOS的Core Data框架支持二进制和SQLite数据存储（不支持ＸＭＬ数据存储），同时也支持从现有的Cocoa应用程序迁移数据。如果是受支持的数据存储，您可以把Core Data资源文件复制到iOS应用程序工程，然后就可以直接使用这些文件。如需进一步了解如何在Xcode工程中使用Core Data，请查看Core Data编程指南。

如果Cocoa应用程序在屏幕上显示很多数据，则在将其迁移到iOS的时候，您可能需要对数据模型进行简化。虽说iOS系统上也可创建数据量大且内容丰富的应用程序，但是这样做并不符合用户需求，因为移动用户通常希望在最短的时间里获得最重要的信息。另外，一次性提供给用户太多数据也行不通，毕竟屏幕的空间有限。而且这样做可能会让应用程序变慢，因为加载数据也需要额外的工作。如果对Cocoa的数据结构进行重构可以提供更好的性能及更优秀的iOS用户体验，那就非常值得一试。

迁移用户界面

iOS用户界面的结构及实现和Mac OS X有很大区别。以 Cocoa表示视图和窗口的对象为例。虽然iOS和Cocoa都有用于表示视图和窗口的对象，但是不同平台对象的工作方式稍有差异。另外，在iOS系统中，视图显示的内容需要更精心地挑选，因为屏幕尺寸有限，而视图又必须足够大以便给用户的手指提供足够多的操作目标。

除了视图对象自身的差异，在程序运行时，视图显示方式也有巨大差别。举个例子，如果您希望在Cocoa应用程序中显示很多数据，则可以增加窗口的尺寸，使用多个窗口或者使用标签窗口来管理数据。而在iOS应用程序中，只存在一个窗口且窗口的尺寸固定不变。因此，应用程序必须按合理尺寸对信息进行分块，并把数据块呈现在不同的视图。对信息进行分块是为了将其划分成多个屏幕内容，然后您可以根据屏幕内容设计相应的应用程序视图。举个例子，如果要在Cocoa显示分层列表数据，您可能会使用一个`NSBrowser`对象。但是在iOS系统中，您就需要创建一组大相径庭的视图用于显示不同层的信息。虽然这种方式会导致更复杂的界面设计，但是它的确是非常重要的显示信息的方式。因此，iOS系统为这种组织方式提供很多支持。

Mac OS X v10.5系统Cocoa才开始引入视图控制器，对于视图控制器的使用可能还未普及。但在iOS应用程序中，视图控制器是用户界面管理基础架构的关键部分。视图控制器管理用户界面的展现，而且它还与系统相互协作，可以保证应用程序的资源不会占用太多内存，因而可防止程序性能下降。总之，理解视图控制器的角色及其在应用程序的使用方式是设计用户界面的关键。

如需iOS通用的用户界面设计原则，请查看iPhone人机接口准则。如需了解用户界面的窗口和视图及其底层架构，请参考iOS 应用程序编程指南。如需了解视图控制器及其创建用户界面的方式，请查看iOS视图控制器编程指南。

内存管理

iOS系统不支持垃圾收集，您需要使用内存管理模型保持、释放或自动释放对象。

和Macintosh计算机相比， iOS设备内存非常有限。因此，您需调整自动释放池的使用，避免创建多个自动释放池对象。另外，请尽可能直接释放对象，不要自动释放。如果您在一个紧凑的循环中分配了很多对象，要么就直接释放那些对象，要么就在循环代码中的恰当的位置创建自动释放池，并在规则的间隔内释放自动释放对象。等到循环结束再释放可能会导致内存不足的警告或导致应用程序被系统杀死。

框架差异

虽然iOS的大多数框架同样存在于 Mac OS X系统，但不同平台框架具有不同的实现方式和使用方式。下面收集了一些Mac OS X 开发者开发iOS应用程序需要注意的重要差别：

UIKit 与 AppKit 的对比

在 iOS系统中，创建图形应用程序、管理事件循环以及执行其他界面相关的任务都离不开UIKit提供的基础结构。UIKit和AppKit具有非常显著的区别，在设计iOS应用程序的时候，应该特别注意这一点。也正是因为这个原因，在将Cocoa应用程序迁移到iOS系统的时候，您必须提供和界面相关的类和逻辑。表6-1列出了框架之间的特定的差异，它可帮助您理解iOS中的应用程序应该具有什么特征：

表 6-1 界面技术的差异

差异	讨论
文档支持	在iOS系统中，文档角色的重要性有所降低，简单内容模型则变的越来越重要。因为iOS系统的应用程序通常只拥有一个窗口（在不连接外部显示的情况下），主窗口是创建及编辑所有应用程序内容的唯一环境。更重要的是，所有和文档相关的操作，包括文件的创建和管理，现在都由应用程序在幕后完成，不再需要用户干预。
视图类	UIKit为您提供一组非常有针对性的视图和控件。AppKit框架有许多视图和控件无法在iOS设备上工作，其他一些视图则被更具iOS特色的视图替代。例如，在显示分层信息的时候，iOS不使用 `NSBrowser`类，而是使用完全不同的样式（导航控制器）。如需了解iOS中的视图和控件及其使用方式，请查看iPhone人机接口准则。
视图座标系统	iOS系统Quartz和UIKit内容的绘画模型和Mac OS X的基本相同，只有一处例外。在 Mac OS X绘画模型坐标系统中，窗口和视图的原点默认位于左下角，坐标轴向上向右延伸。但在iOS系统中，默认的原点位置是左上角，坐标轴向下向右延伸。Mac OS X的座标系统称为“被翻转”的座标系统，iOS则是缺省座标系统。如需进一步了解图形和座标系统，请查看iOS视图编程指南。
窗口即视图	从概念上来看，iOS系统的窗口和视图Mac OS X的具有相同含义。但从实现的角度来看，区别很大。在Mac OS X系统中，`NSWindow`类是`NSResponder`类的子类，但在iOS系统中，`UIWindow`实际是`UIView`的子类。继承关系上的改变表明窗口将会使用Core Animation层来绘制外表。之所以有这样的改变，主要是为了在操作系统级别支持窗口分层。举个例子，系统可以在一个独立的窗口中显示状态栏，并让该窗口浮动于应用程序窗口之上。 iOS系统和Mac OS X系统另外一个差异和窗口的使用方式相关。Mac OS X应用程序可以用于任意数量的窗口，但大多数iOS应用程序只能有一个窗口。在iOS应用程序中显示不同屏幕的数据不是通过改变窗口实现，而是通过在应用程序窗口中切换定制视图来完成。
事件处理	UIKit的事件处理模型和Mac OS X的事件处理模型区别很大。UIKit框架不向视图发送鼠标和键盘事件，而是发送触摸和移动事件。这些事件不但要求您实现一组不同的方法，同时也要求您修改整个事件处理代码。举个例子，本地跟踪循环的排队事件不能包含触摸事件，您的代码也据此做相应调整。如需进一步了解iOS应用程序的事件处理，请参考iOS事件处理指南。
目标–动作模型	UIKit支持三种形式的动作，AppKit仅支持一种。UIKit的控件可以在不同的交互阶段调用唤醒不同动作，而且一个交互过程可以指定多个目标。因此，在UIKit中，一个控件可以在一次交互过程中向多个目标发送多个不同的动作。如需进一步了解iOS应用程序的目标–动作模型，请查看iOS事件处理指南。
绘画及打印支持	为支持UIKit渲染需要，UIKit的绘画能力经过适当的调节。它支持图片的加载和显示、字符串显示、颜色管理、字体管理以及多个用于渲染矩阵和获取图形上下文的函数。UIKit不包含通用目的的绘图类，因为iOS系统使用其他方式完成此类功能（即Quartz和OpenGL ES）。 iOS系统部支持打印功能，iOS设备不能连接打印机或其他相关的打印硬件。 如需进一步了解图形和绘图方面的信息，请查看iOS视图编程指南。
文本支持	撰写电子邮件和记事本是iOS系统提供的主要的文本支持。UIKit类可以让应用程序显示并编辑简单的字符串和稍微复杂点的HMTL内容。 在iOS 3.2及后续系统中，Core Text框架 和UIKit框架提供更加精密的文本处理能力，您可以通过这它们实现更精密的文本编辑及展现视图，也可通过它们定制视图提供的输入方法。如需进一步了解文本支持相关的信息，请查看iOS文本和Web编程指南。

存取方法的使用和属性对比	UIKit在其类声明中大量使用属性。属性由Mac OS X在10.5版本引入，是AppKit框架大量的类创建出来以后才出现。属性不是对AppKit框架getter和setter方法的简单模仿，而是被UIKit用于简化类接口。如需了解属性的使用方式，请查看Objective-C编程语言中的“属性声明”。
控件和单元	UIKit控件不使用单元。单元被Mac OS X作为视图的轻量级替代物。但是UIKit视图本身就是非常轻量的对象，因此单元派不上用场。虽然在命名约定上，UITableView类也用到了单元这个词，但是此处的单元实际上是UITableView的子类。
表视图	iOS系统的 UITableView 类可以看成是AppKit框架中NSTableView和NSOutlineView的折中物。它结合Appkit框架中者两个类的特征，更适合在小屏幕上显示。UITableView一次显示一列数据，而且您将相关的行组合成一个区段。UITableView也可用于显示并编辑分层列表数据。如需进一步了解UITableView类，请查看UITableView类参考。
菜单	几乎所有iOS应用程序的命令集都比类似的Mac OS X应用程序小得多，因此，iOS不支持菜单，通常也用不到菜单。对于需要少数的命令的场合，使用工具栏或者一组按键更加合适。对于需要数据菜单的场合，使用拾取器或导航控制器界面通常更合适，而如需对上下文敏感的菜单，则可其中的菜单项显示在Edit菜单，用它们替代或补充剪切、复制或者粘贴等命令。
Core Animation层	在 iOS系统中，所有外表的绘制都由Core Animation层实现。该框架还隐式为许多视图相关属性提供的动画支持。由于这种内建的动画支持，您就不需要在代码中显示使用Core Animation层，只需更改一下视图的某些属性即可实现大多数动画。只有当需要对分层进行精确控制或者不想将某些特征暴露于视图层，您才需要直接使用Core Animation。如需了解将Core Animation层整合到iOS绘图模型的方式，请查看iOS视图编程指南。

如需了解UIKit的类信息，请查看UIKit框架参考。

Foundation 框架的差异

Mac OS X和 iOS都有Foundation框架，大多数您所预期的类都可以在该框架中找到。 两个平台的框架都支持数值管理、字符串、集合、线程以及许多其他常见的数据类型。表6-2列出一些不包含于iOS框架的重要功能以及相关类不存在的缘由，同时也尽量列出有哪些技术可作为替代。

表 6-2 iOS的Foundation不具有的技术

技术	注意事项
元数据和预测管理	iOS不支持Spotlight 元数据和搜索预测，因为iOS不支持Spotlight。
分布式对象和端口名称服务管理	iOS不存在分布式对象技术，但是您可以使用NSPort家族类和端口（及socket）进行交互交互。您也可以使用Core Foundation和CFNetwork框架处理网络需求。
Cocoa绑定	iOS不支持Cocoa绑定，而是使用经过少量修改的目标-动作模型。因为这种方式可以让代码对动作的处理方式有更多的灵活性。
Objective-C垃圾收集	iOS不支持垃圾收集，您必须使用内存管理模型。您需要通过保持对象来宣告对对象的拥有权，并在不需要对象的时候释放对象。
AppleScript支持	iOS不支持AppleScript。

iOS系统的Foundation框架提供对XML的支持，您可以通过 NSXMLParser 类解析XML文件，其他解析类（包括NSXMLDocument、NSXMLNode）不受支持。除了NSXMLParser之外，您还可以使用libXML2库，这是C语言的XML解析接口。

如果需要了解哪些类存在于Mac OS X而不存在于iOS，请查看位于Foundation框架参考中的“Foundation 框架”的类层次图。

其他框架的改变

表 6-3列出iOS其他框架的关键差异。

表 6-3 同时存在于iOS和Mac OS X的框架之间的差异

框架	差异
AddressBook.framework	该框架接口可用于访问用户的联系人信息。虽然名称相同，但是此框架的iOS版本和Mac OS X版本却有很大的区别。 在iOS系统中，除了访问联系人数据的C接口，您还可以使用Address Book UI框架提供的类展现标准联系人挑选和编辑界面。 如需进一步的信息，请查看Address Book框架参考。
AudioToolbox.framework AudioUnit.framework CoreAudio.framework	在iOS系统中，这些框架支持音频录制、播放以及单声道和多声道的音频内容混合，但不支持更高级的音频处理功能和定制音频单元插件。不过iOS系统增加了一个功能，即触发iOS设备（具有相应硬件）的震动功能。如果需要了解如何使用音频支持，请查看iOS应用程序编程指南中的多媒体支持 。
CFNetwork.framework	该框架包含Core Foundation Network接口。在iOS系统中，CFNetwork框架是顶层框架，它没有子框架。该框架的接口大部分保持不变。如需进一步信息，请查看CFNetwork框架参考。
CoreGraphics.framework	该框架包含Quartz接口。在iOS系统中，Core Graphics框架是顶层框架，它没有子框架。使用Quartz创建路径、渐变、阴影、图案、图像以及位图的方式和Mac OS X系统完全相同。不过有一些Quartz的功能（包括PostScript支持、图像来源和去向、Quartz显示服务支持、Quartz事件服务支持）不存在于iOS系统。如需进一步信息，请查看Core Graphics框架参考。
OpenGLES.framework	OpenGL ES 是专为嵌入式系统设计的OpenGL版本。如果您是OpenGL开发人员，则应该会很熟悉OpenGL ES接口。不过，OpenGL ES接口还是有几点较大差别。首先，它是一套更加小巧的接口，仅支持可以在现有图形硬件有效执行的功能。第二，许多桌面OpenGL可以使用的扩展并不存在于OpenGL ES。虽然如此，您应该还是能够执行大多数和桌面OpenGL相同的操作。但如果你是在迁移现有的OpenGL代码，则可能需要重写一部分代码，需要使用iOS系统的渲染技术（不同于Mac OS X）。如需了解iOS对OpenGL ES的支持，请查看iOS OpenGL ES编程指南。

QuartzCore.framework	该框架包含Core Animation接口。iOS大部分 Core Animation接口和Mac OS X相同。但是iOS系统没有用于管理布局约束的类，也不支持使用Core Image过滤器。另外，iOS也没有Core Image和Core Video接口（两者都包含于Mac OS X版本的QuartzCore框架）。如需进一步信息，请查看Quartz Core框架参考。
Security.framework	该框架包含安全接口。在iOS系统中，该框架通过加解密、伪随机数生成以及Keychain保护应用程序数据安全。该框架不包含身份验证或身份验证接口，也不支持显示证书内容。Keychain接口也是Mac OS X版本的简化。如需了解iOS的安全支持，请查看iOS应用程序编程指南。
SystemConfiguration.framework	该框架包含和网络相关的接口。在iOS系统中，您可以使用这些接口来决定设备如何与网络连接，是通过EDGE、GPRS或是通过Wi-Fi。

从 Cocoa 迁移到 iOS

如果您是一名Cocoa开发者，则会对iOS中的许多框架感到熟悉。虽然iOS的基础技术栈在许多方面都等同于Mac OS X，但有些框架则非如此。本章针对创建iOS应用程序可能碰到的差异进行描述，同时也说明如何应对某些重要变化：

请注意：本章专为已熟悉Cocoa术语和编程技术的开发者设计。如果您需要进一步了解Cocoa应用程序（以及iOS应用程序）使用的基本设计模式，请查看Cocoa基础指南。

通用的迁移注意事项

如果您的Cocoa应用程序已经使用模型-视图-控制器模式，则很容易将程序的关键部分迁移到iOS系统。如需了解如何设计iOS应用程序，请查看iOS应用程序编程指南。

迁移数据模型

如果Cocoa应用程序的数据模型以Foundation或者Core Foundation框架的类为基础，则将数据模型迁移到iOS只需做少量修改，甚至无需修改。因为iOS也支持这两个框架。并且彼此差异不大。大多数差异要么相对次要，要么和iOS应用程序无法具有的功能相关。例如iOS应用程序所不支持AppleScript。如需此类差异的详细列表，请查看“Foundation框架差异”。

如果Cocoa应用程序使用Core Data，则只能将其数据模型迁移到iOS3.0及后续版本，因为早期的iOS不支持Core Data。iOS的Core Data框架支持二进制和SQLite数据存储（不支持ＸＭＬ数据存储），同时也支持从现有的Cocoa应用程序迁移数据。如果是受支持的数据存储，您可以把Core Data资源文件复制到iOS应用程序工程，然后就可以直接使用这些文件。如需进一步了解如何在Xcode工程中使用Core Data，请查看Core Data编程指南。

如果Cocoa应用程序在屏幕上显示很多数据，则在将其迁移到iOS的时候，您可能需要对数据模型进行简化。虽说iOS系统上也可创建数据量大且内容丰富的应用程序，但是这样做并不符合用户需求，因为移动用户通常希望在最短的时间里获得最重要的信息。另外，一次性提供给用户太多数据也行不通，毕竟屏幕的空间有限。而且这样做可能会让应用程序变慢，因为加载数据也需要额外的工作。如果对Cocoa的数据结构进行重构可以提供更好的性能及更优秀的iOS用户体验，那就非常值得一试。

迁移用户界面

iOS用户界面的结构及实现和Mac OS X有很大区别。以 Cocoa表示视图和窗口的对象为例。虽然iOS和Cocoa都有用于表示视图和窗口的对象，但是不同平台对象的工作方式稍有差异。另外，在iOS系统中，视图显示的内容需要更精心地挑选，因为屏幕尺寸有限，而视图又必须足够大以便给用户的手指提供足够多的操作目标。

除了视图对象自身的差异，在程序运行时，视图显示方式也有巨大差别。举个例子，如果您希望在Cocoa应用程序中显示很多数据，则可以增加窗口的尺寸，使用多个窗口或者使用标签窗口来管理数据。而在iOS应用程序中，只存在一个窗口且窗口的尺寸固定不变。因此，应用程序必须按合理尺寸对信息进行分块，并把数据块呈现在不同的视图。对信息进行分块是为了将其划分成多个屏幕内容，然后您可以根据屏幕内容设计相应的应用程序视图。举个例子，如果要在Cocoa显示分层列表数据，您可能会使用一个NSBrowser对象。但是在iOS系统中，您就需要创建一组大相径庭的视图用于显示不同层的信息。虽然这种方式会导致更复杂的界面设计，但是它的确是非常重要的显示信息的方式。因此，iOS系统为这种组织方式提供很多支持。

Mac OS X v10.5系统Cocoa才开始引入视图控制器，对于视图控制器的使用可能还未普及。但在iOS应用程序中，视图控制器是用户界面管理基础架构的关键部分。视图控制器管理用户界面的展现，而且它还与系统相互协作，可以保证应用程序的资源不会占用太多内存，因而可防止程序性能下降。总之，理解视图控制器的角色及其在应用程序的使用方式是设计用户界面的关键。

如需iOS通用的用户界面设计原则，请查看iPhone人机接口准则。如需了解用户界面的窗口和视图及其底层架构，请参考iOS 应用程序编程指南。如需了解视图控制器及其创建用户界面的方式，请查看iOS视图控制器编程指南。

内存管理

iOS系统不支持垃圾收集，您需要使用内存管理模型保持、释放或自动释放对象。

和Macintosh计算机相比， iOS设备内存非常有限。因此，您需调整自动释放池的使用，避免创建多个自动释放池对象。另外，请尽可能直接释放对象，不要自动释放。如果您在一个紧凑的循环中分配了很多对象，要么就直接释放那些对象，要么就在循环代码中的恰当的位置创建自动释放池，并在规则的间隔内释放自动释放对象。等到循环结束再释放可能会导致内存不足的警告或导致应用程序被系统杀死。

框架差异

虽然iOS的大多数框架同样存在于 Mac OS X系统，但不同平台框架具有不同的实现方式和使用方式。下面收集了一些Mac OS X 开发者开发iOS应用程序需要注意的重要差别：

UIKit 与 AppKit 的对比

在 iOS系统中，创建图形应用程序、管理事件循环以及执行其他界面相关的任务都离不开UIKit提供的基础结构。UIKit和AppKit具有非常显著的区别，在设计iOS应用程序的时候，应该特别注意这一点。也正是因为这个原因，在将Cocoa应用程序迁移到iOS系统的时候，您必须提供和界面相关的类和逻辑。表6-1列出了框架之间的特定的差异，它可帮助您理解iOS中的应用程序应该具有什么特征：

表 6-1 界面技术的差异

差异	讨论

文档支持	在iOS系统中，文档角色的重要性有所降低，简单内容模型则变的越来越重要。因为iOS系统的应用程序通常只拥有一个窗口（在不连接外部显示的情况下），主窗口是创建及编辑所有应用程序内容的唯一环境。更重要的是，所有和文档相关的操作，包括文件的创建和管理，现在都由应用程序在幕后完成，不再需要用户干预。
视图类	UIKit为您提供一组非常有针对性的视图和控件。AppKit框架有许多视图和控件无法在iOS设备上工作，其他一些视图则被更具iOS特色的视图替代。例如，在显示分层信息的时候，iOS不使用 `NSBrowser`类，而是使用完全不同的样式（导航控制器）。如需了解iOS中的视图和控件及其使用方式，请查看iPhone人机接口准则。
视图座标系统	iOS系统Quartz和UIKit内容的绘画模型和Mac OS X的基本相同，只有一处例外。在 Mac OS X绘画模型坐标系统中，窗口和视图的原点默认位于左下角，坐标轴向上向右延伸。但在iOS系统中，默认的原点位置是左上角，坐标轴向下向右延伸。Mac OS X的座标系统称为“被翻转”的座标系统，iOS则是缺省座标系统。如需进一步了解图形和座标系统，请查看iOS视图编程指南。
窗口即视图	从概念上来看，iOS系统的窗口和视图Mac OS X的具有相同含义。但从实现的角度来看，区别很大。在Mac OS X系统中，`NSWindow`类是`NSResponder`类的子类，但在iOS系统中，`UIWindow`实际是`UIView`的子类。继承关系上的改变表明窗口将会使用Core Animation层来绘制外表。之所以有这样的改变，主要是为了在操作系统级别支持窗口分层。举个例子，系统可以在一个独立的窗口中显示状态栏，并让该窗口浮动于应用程序窗口之上。 iOS系统和Mac OS X系统另外一个差异和窗口的使用方式相关。Mac OS X应用程序可以用于任意数量的窗口，但大多数iOS应用程序只能有一个窗口。在iOS应用程序中显示不同屏幕的数据不是通过改变窗口实现，而是通过在应用程序窗口中切换定制视图来完成。
事件处理	UIKit的事件处理模型和Mac OS X的事件处理模型区别很大。UIKit框架不向视图发送鼠标和键盘事件，而是发送触摸和移动事件。这些事件不但要求您实现一组不同的方法，同时也要求您修改整个事件处理代码。举个例子，本地跟踪循环的排队事件不能包含触摸事件，您的代码也据此做相应调整。如需进一步了解iOS应用程序的事件处理，请参考iOS事件处理指南。
目标-动作模型	UIKit支持三种形式的动作，AppKit仅支持一种。UIKit的控件可以在不同的交互阶段调用唤醒不同动作，而且一个交互过程可以指定多个目标。因此，在UIKit中，一个控件可以在一次交互过程中向多个目标发送多个不同的动作。如需进一步了解iOS应用程序的目标-动作模型，请查看iOS事件处理指南。
绘画及打印支持	为支持UIKit渲染需要，UIKit的绘画能力经过适当的调节。它支持图片的加载和显示、字符串显示、颜色管理、字体管理以及多个用于渲染矩阵和获取图形上下文的函数。UIKit不包含通用目的的绘图类，因为iOS系统使用其他方式完成此类功能（即Quartz和OpenGL ES）。 iOS系统部支持打印功能，iOS设备不能连接打印机或其他相关的打印硬件。 如需进一步了解图形和绘图方面的信息，请查看iOS视图编程指南。
文本支持	撰写电子邮件和记事本是iOS系统提供的主要的文本支持。UIKit类可以让应用程序显示并编辑简单的字符串和稍微复杂点的HMTL内容。 在iOS 3.2及后续系统中，Core Text框架 和UIKit框架提供更加精密的文本处理能力，您可以通过这它们实现更精密的文本编辑及展现视图，也可通过它们定制视图提供的输入方法。如需进一步了解文本支持相关的信息，请查看iOS文本和Web编程指南。
存取方法的使用和属性对比	UIKit在其类声明中大量使用属性。属性由Mac OS X在10.5版本引入，是AppKit框架大量的类创建出来以后才出现。属性不是对AppKit框架getter和setter方法的简单模仿，而是被UIKit用于简化类接口。如需了解属性的使用方式，请查看Objective-C编程语言中的“属性声明”。
控件和单元	UIKit控件不使用单元。单元被Mac OS X作为视图的轻量级替代物。但是UIKit视图本身就是非常轻量的对象，因此单元派不上用场。虽然在命名约定上，`UITableView`类也用到了单元这个词，但是此处的单元实际上是`UITableView`的子类。
表视图	iOS系统的 `UITableView` 类可以看成是`AppKit`框架中`NSTableView`和`NSOutlineView`的折中物。它结合Appkit框架中者两个类的特征，更适合在小屏幕上显示。`UITableView`一次显示一列数据，而且您将相关的行组合成一个区段。`UITableView`也可用于显示并编辑分层列表数据。如需进一步了解`UITableView`类，请查看UITableView类参考。
菜单	几乎所有iOS应用程序的命令集都比类似的Mac OS X应用程序小得多，因此，iOS不支持菜单，通常也用不到菜单。对于需要少数的命令的场合，使用工具栏或者一组按键更加合适。对于需要数据菜单的场合，使用拾取器或导航控制器界面通常更合适，而如需对上下文敏感的菜单，则可其中的菜单项显示在Edit菜单，用它们替代或补充剪切、复制或者粘贴等命令。
Core Animation层	在 iOS系统中，所有外表的绘制都由Core Animation层实现。该框架还隐式为许多视图相关属性提供的动画支持。由于这种内建的动画支持，您就不需要在代码中显示使用Core Animation层，只需更改一下视图的某些属性即可实现大多数动画。只有当需要对分层进行精确控制或者不想将某些特征暴露于视图层，您才需要直接使用Core Animation。如需了解将Core Animation层整合到iOS绘图模型的方式，请查看iOS视图编程指南。

如需了解UIKit的类信息，请查看UIKit框架参考。

Foundation 框架的差异

Mac OS X和 iOS都有Foundation框架，大多数您所预期的类都可以在该框架中找到。 两个平台的框架都支持数值管理、字符串、集合、线程以及许多其他常见的数据类型。表6-2列出一些不包含于iOS框架的重要功能以及相关类不存在的缘由，同时也尽量列出有哪些技术可作为替代。

表 6-2 iOS的Foundation不具有的技术

技术	注意事项
元数据和预测管理	iOS不支持Spotlight 元数据和搜索预测，因为iOS不支持Spotlight。
分布式对象和端口名称服务管理	iOS不存在分布式对象技术，但是您可以使用`NSPort`家族类和端口（及socket）进行交互交互。您也可以使用Core Foundation和CFNetwork框架处理网络需求。
Cocoa绑定	iOS不支持Cocoa绑定，而是使用经过少量修改的目标-动作模型。因为这种方式可以让代码对动作的处理方式有更多的灵活性。
Objective-C垃圾收集	iOS不支持垃圾收集，您必须使用内存管理模型。您需要通过保持对象来宣告对对象的拥有权，并在不需要对象的时候释放对象。

AppleScript支持	iOS不支持AppleScript。

iOS系统的Foundation框架提供对XML的支持，您可以通过 `NSXMLParser` 类解析XML文件，其他解析类（包括`NSXMLDocument`、`NSXMLNode`）不受支持。除了`NSXMLParser`之外，您还可以使用`libXML2`库，这是C语言的XML解析接口。

如果需要了解哪些类存在于Mac OS X而不存在于iOS，请查看位于Foundation框架参考中的"Foundation 框架"的类层次图。

其他框架的改变

表 6-3列出iOS其他框架的关键差异。

表 6-3 同时存在于iOS和Mac OS X的框架之间的差异

框架	差异
`AddressBook.framework`	该框架接口可用于访问用户的联系人信息。虽然名称相同，但是此框架的iOS版本和Mac OS X版本却有很大的区别。 在iOS系统中，除了访问联系人数据的C接口，您还可以使用Address Book UI框架提供的类展现标准联系人挑选和编辑界面。 如需进一步的信息，请查看Address Book框架参考。

`AudioToolbox.framework`
`AudioUnit.framework`
`CoreAudio.framework`
	在iOS系统中，这些框架支持音频录制、播放以及单声道和多声道的音频内容混合，但不支持更高级的音频处理功能和定制音频单元插件。不过iOS系统增加了一个功能，即触发iOS设备（具有相应硬件）的震动功能。如果需要了解如何使用音频支持，请查看iOS应用程序编程指南中的多媒体支持 。

`CFNetwork.framework`	该框架包含Core Foundation Network接口。在iOS系统中，CFNetwork框架是顶层框架，它没有子框架。该框架的接口大部分保持不变。如需进一步信息，请查看CFNetwork框架参考。
`CoreGraphics.framework`	该框架包含Quartz接口。在iOS系统中，Core Graphics框架是顶层框架，它没有子框架。使用Quartz创建路径、渐变、阴影、图案、图像以及位图的方式和Mac OS X系统完全相同。不过有一些Quartz的功能（包括PostScript支持、图像来源和去向、Quartz显示服务支持、Quartz事件服务支持）不存在于iOS系统。如需进一步信息，请查看Core Graphics框架参考。
`OpenGLES.framework`	OpenGL ES 是专为嵌入式系统设计的OpenGL版本。如果您是OpenGL开发人员，则应该会很熟悉OpenGL ES接口。不过，OpenGL ES接口还是有几点较大差别。首先，它是一套更加小巧的接口，仅支持可以在现有图形硬件有效执行的功能。第二，许多桌面OpenGL可以使用的扩展并不存在于OpenGL ES。虽然如此，您应该还是能够执行大多数和桌面OpenGL相同的操作。但如果你是在迁移现有的OpenGL代码，则可能需要重写一部分代码，需要使用iOS系统的渲染技术（不同于Mac OS X）。如需了解iOS对OpenGL ES的支持，请查看iOS OpenGL ES编程指南。
`QuartzCore.framework`	该框架包含Core Animation接口。iOS大部分 Core Animation接口和Mac OS X相同。但是iOS系统没有用于管理布局约束的类，也不支持使用Core Image过滤器。另外，iOS也没有Core Image和Core Video接口（两者都包含于Mac OS X版本的QuartzCore框架）。如需进一步信息，请查看Quartz Core框架参考。
`Security.framework`	该框架包含安全接口。在iOS系统中，该框架通过加解密、伪随机数生成以及Keychain保护应用程序数据安全。该框架不包含身份验证或身份验证接口，也不支持显示证书内容。Keychain接口也是Mac OS X版本的简化。如需了解iOS的安全支持，请查看iOS应用程序编程指南。
`SystemConfiguration.framework`	该框架包含和网络相关的接口。在iOS系统中，您可以使用这些接口来决定设备如何与网络连接，是通过EDGE、GPRS或是通过Wi-Fi。

iOS 开发者工具

如果您希望在iOS系统开发应用程序，则需要配备一台运行Xcode工具的Mac OS X计算机。XCode是苹果公司的开发工具套件，它可用于管理工程，编辑代码，构建可执行文件，进行源码级调试，进行源代码仓库管理，进行性能调节等。套件的核心是Xcode应用程序本身，它用于提供基本的源代码开发环境。但是Xcode并非唯一可以使用的工具，下面的章节将向您介绍开发iOS软件将会使用到的关键应用程序：

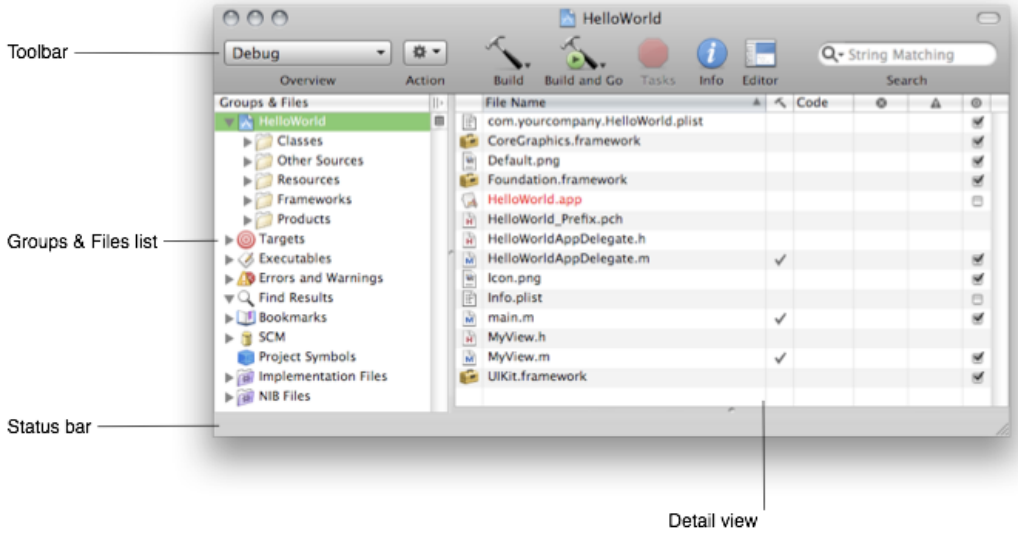
Xcode

您的开发经验应当是集中在Xcode应用程序上。Xcode 是一个集成开发环境(IDE)，从创建及管理iOS工程和源文件到将源代码链编程可执行文件，并在设备运行代码或者在iPhone模拟器上调试代码所需的各种工具，尽皆包含其中。总之，Xcode将下面这一系列的功能整合在一起，可以让iOS应用程序开发变得更加容易：

- 用于对软件产品进行定义的工程管理系统。
- 代码编辑环境，包括为文法显示不同颜色、代码补全以及符号指示等多种功能。
- 高级文档阅读工具 ，可用于阅读搜索苹果文档。
- 对上下文敏感的检查工具，可用于查看选定代码符号的信息。
- 高级链编系统，具有依赖检查及链编规则计算功能。
- GCC编译器，此编译器支持对C、C++、Objective-C、Objective-C++以及Objective-C 2.0和其他语言进行编译。
- 集成源码级的调试功能，此功能使用GDB来实现。
- 分布式计算，此功能可以让您将巨大的工程分布到数台联网的机器上运 行。
- 预测编译，此功能可以加速单个文件的编译周转时间。
- 高级调试功能，例如停顿和继续运行，而且可以定制数据格式化方式。
- 高级重构工具，这些工具可以让您在不改变整体行为的前提下对代码进行全局性的修改。
- 工程快照的支持。工程快照是一种轻量级的本地源代码管理形式。
- 支持启动性能工具对软件进行分析。
- 支持源代码管理集成
- 支持使用AppleScript实现链编过程自动化。
- 可以生成DWARF和Stabs调试信息（所有的新工程都会默认生成DWARF调试信息）

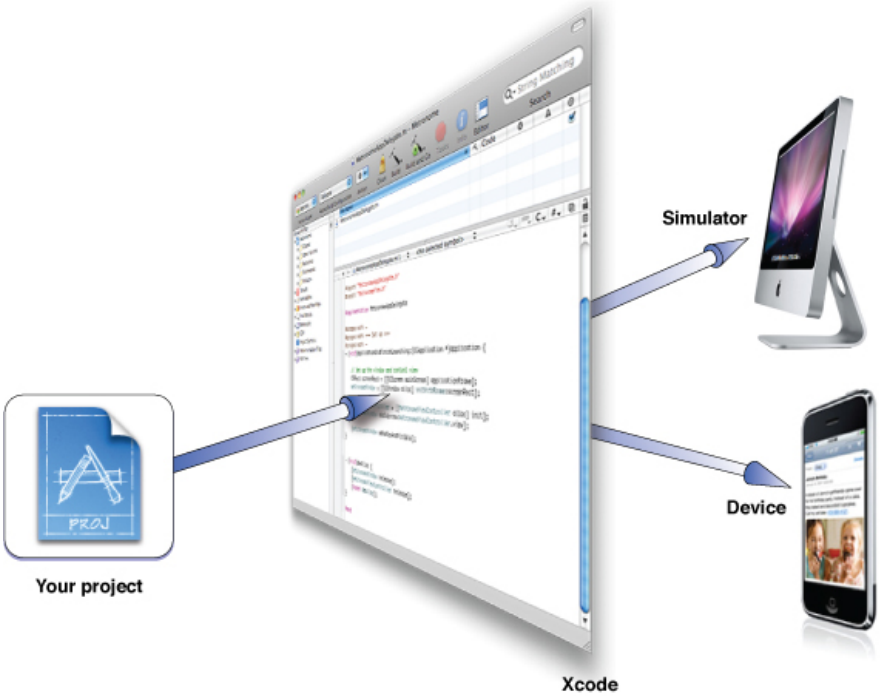
如欲创建一个新的iOS应用程序，您需先在Xcode中创建一个新的工程。所有和应用程序相关的信息，包括源文件、链编设置以及将所有这些事物集成在一起的规则都由该工程管理。Xcode工程的中心部分是一个工程窗口，如图 A-1所示。此窗口为应用程序的关键元素提供快速访问。Groups &Files 列表可以对工程文件（包括源文件以及由源文件生成的链编目标）进行管理。工具栏可以访问常用的工具和命令。详情面板可以配置出一块区域用于对文件进行各种操作。工程窗口的其他部分为您提供一些工程上下文信息。

图 A-1　Xcode工程窗口



通过Xcode链编应用程序的时候，您可将其链编至iPhone 模拟器 或设备。模拟器为应用程序测试提供本地环境，您可以通过它测试应用程序是否具有正确行为。当应用程序的基本行为符合预期后，再通过Xcode将其链编到设备上，然后在已连接至计算机的iOS设备上运行程序。在设备运行应用程序是最终测试环境。在这一测试过程中，Xcode允许您将内建调试器绑定至设备上运行的代码，直接在设备上进行调试。

图 A-2　从Xcode中运行一个工程



如需进一步了解如何在iOS系统中链编运行一个工程，请查看OS 开发指南 。如需进一步了解整个Xcode环境，请查看Xcode教程 。

Interface Builder

Interface Builder 以所见即所得方式组装用户界面。通过Interface Builder, 您可以把事先配置好的组件拖拽到应用程序窗口，并最终组装出应用程序的用户界面（如图A-3所显示）。这里所说的组件既包括标准系统控件，例如切换控件、文本字段及按键，也包括一些定制视图（用于表现应用程序特有的外观）。将控件放在窗口表面后，您还可以拽着它在四周移动，为其寻找合适的位置。同时，您可以使用inspector配置组件属性，并在对象和代码之间建立正确关联。当用户界面达到要求后，您可以将这些界面的内容保存到nib文件（一种定制的资源文件格式）。

图 A-3　使用Interface Builder创建iOS界面

您在Interface Builder创建的nib文件包含UIKit在运行时为应用程序重建对象所需的一切信息。在加载nib文件的时候，系统会为保存在文件中每个对象创建一份运行时版本，然后再对其进行配置，使之和Interface Builder中的状态保持一致。另外，系统还将根据您制定的关联信息为新建对象和应用程序已有对象建立关联。这些关联可以为代码提供指向nib文件包含的对象的指针，同时也为这些对象与代码中的用户动作进行通讯提供必要信息。

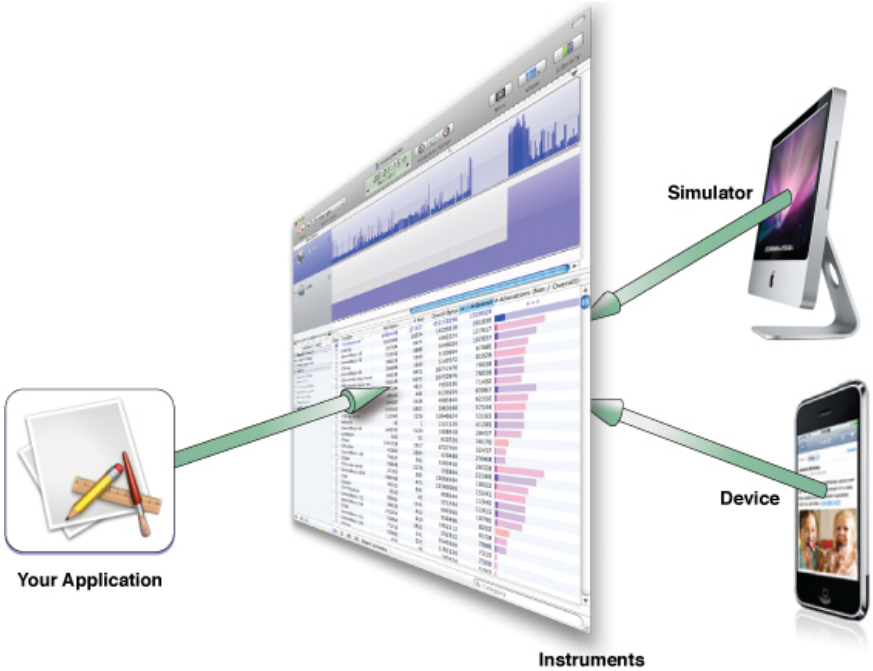
总而言之，在创建应用程序用户界面的时候，使用Interface Builder可以节省大量的时间。使用Interface Builder之后，在创建、配置及摆放界面对象的时候就无需再编写定制代码，因为它是一种可视化的编辑器，编辑时所见的界面即运行时所得。

如需进一步了解使用Interface Builder的信息，请查看Interface Builder 用户指南。

Instruments

为确保软件具有最佳的用户体验，在iOS应用程序运行于模拟器或设备上时，您可以利用Instruments环境分析其性能。Instruments会收集运行程序的数据，并以时间线方式展现数据。可以采集应用程序数据包括应用程序内存使用情况、磁盘活动、网络活动以及图形性能。时间线视图可以同时显示不同类型的信息，这样，您就可以把整个应用程序的行为相互关联起来，而非仅看到某一特定方面的行为。如果还需要更加详细的信息，则可以查看Instruments收集的精细采样。

图 A-4 使用Instruments调整应用程序



除了时间线视图，Instruments 还提供一些工具帮助您对不同时间的应用程序行为进行分析。举个例子，Instruments 窗口允许您将多次运行的数据保存起来，这样您就可以看到应用程序的行为是否确实有所改善，或仍需调整。您也可以把这些数据保存在在一份Instruments文档中以备随时查看。

如需深入了解如何使用Instruments分析iOS应用程序，请查看 OS 开发指南。 如需使用Instruments的一般性信息，请查看Instruments用户指南 。

Shark

Shark是请打的分析iOS应用程序性能的工具。 当程序运行在iOS设备时，您可以通过Shark从几个不同方面对代码进行剖析。剖析结果可认为是应用程序运行时行为的统计采样，您可以通过Shark的数据采集和图表化工具对它进行分析。这些工具可以帮助您直观地了解应用程序的运行时行为，进而找到潜在产生问题之处。

如需进一步了解如何在iOS设备上使用Shark，请查看Shark 用户指南。

iOS 的框架

本附录介绍iOS系统包含的框架，它们为编写iOS平台的软件提供必要的接口。下面的表格尽可能地列出框架中的类、方法、函数、类型以及常量使用的关键前缀，请避免在您的符号名称中使用这些前缀。

设备中的框架

表B-1描述iOS设备提供的框架，它们位于<Xcode>`/Platforms/iPhoneOS.platform/Developer/SDKs/`<iOS_SDK>`/System/Library/Frameworks`目录。路径中的<Xcode>表示Xcode的安装目录， <iOS_SDK>则表示目标SDK版本。表中标题为“最先引入”的那一列表示首次引入相关框架的的iOS系统版本。

表 B-1 设备中的框架

名称	最先引入	前缀	描述
`Accelerate.framework`	4.0	`cblas,vDSP`	包含加速数学和DSP函数。 请查看 加速框架参考 。
`AddressBook.framework`	2.0	`AB`	包含直接访问用户联系人数据库的函数。请查看地址簿框架参考。
`AddressBookUI.framework`	2.0	`AB`	包含显示系统定义的联系人挑选界面和编辑界面的类。请查看iOS地址簿UI框架参考 。
`AssetsLibrary.framework`	4.0	`AL`	包含显示用户照片和视频的类，请查看资源库框架参考 。
`AudioToolbox.framework`	2.0	`AU,Audio`	包含处理音频流数据以及播放或录制音频的接口。请查看音频工具箱框架参考。
`AudioUnit.framework`	2.0	`AU,Audio`	包含加载并使用音频单元的接口。请查看音频单元框架参考.
`AVFoundation.framework`	2.2	`AV`	包含播放或录制音频的Objective-C接口。请查看AV Foundation框架参考.。
`CFNetwork.framework`	2.0	`CF`	包含通过WiFi或者蜂窝无线访问网络的接口。请查看CFNetwork框架参考。
`CoreAudio.framework`	2.0	`Audio`	包含Core Audio框架使用的各种数据类型。请查看Core Audio 框架参考 。
`CoreData.framework`	3.0	`NS`	包含管理应用程序数据模型的接口。请查看Core Data 框架参考 。
`CoreFoundation.framework`	2.0	`CF`	提供一些基本软件服务，包括常见数据类型抽象、字符串实用工具、群体类型实用工具、资源管理以及偏好设置。请查看rovCore Foundation框架参考。
`CoreGraphics.framework`	2.0	`CG`	包含Quartz 2D接口。请查看Core Graphics 框架参考 。
`CoreLocation.framework`	2.0	`CL`	包含确定用户方位信息的接口。请查看Core Location 框架参考。
`CoreMedia.framework`	4.0	`CM`	包含操作音频和视频的底层例程。请查看Core Media 框架参考。
`CoreMotion.framework`	4.0	`CM`	包含访问加速度计以及陀螺仪的数据的接口。请查看Core Motion 框架参考。
`CoreTelephony.framework`	4.0	`CT`	包含访问电话相关的信息的例程。请查看Core Telephony 框架参考。
`CoreText.framework`	3.2	`CT`	包含一个文本的布局渲染引擎。请查看Core Text参考集.。
`CoreVideo.framework`	4.0	`CV`	包含操作音频和视频的底层例程。请不要直接使用该框架。
`EventKit.framework`	4.0	`EK`	包含访问用户日历事件数据的接口。请查看Event Kit 框架参考。
`EventKitUI.framework`	4.0	`EK`	包含显示标准系统日历界面的类。请查看Event Kit UI 框架参考。
`ExternalAccessory.framework`	3.0	`EA`	包含与外设进行通讯的接口。请查看External Accessory 框架参考。
`Foundation.framework`	2.0	`NS`	包含Cocoa Foundation层的类和方法。请查看Foundation 框架参考。
`GameKit.framework`	3.0	`GK`	包含点对点连接管理接口。请查看Game Kit 框架参考。
`iAd.framework`	4.0	`AD`	包含在应用程序中显示广告的类。请查看iAd 框架参考。
`ImageIO.framework`	4.0	`CG`	包含读取或写入图像数据的类。请查看Image I/O 参考集。

IOKit.framework	2.0	N/A	包含设备所使用的接口。请不要直接使用此框架。
MapKit.framework	3.0	MK	包含将地图界面嵌入到应用程序的类，也可以用于查找地理编码反向坐标。请查看Map Kit框架参考。
MediaPlayer.framework	2.0	MP	包含显示全屏视频的接口。请查看Media Player 框架参考。
MessageUI.framework	3.0	MF	包含撰写和排队发送电子邮件信息的界面。请查看Message UI 框架参考。
MobileCoreServices.framework	3.0	UT	定义系统支持的统一类型标识符（UTIs）。
OpenAL.framework	2.0	AL	包含OpenAL接口。OpenAL是一个跨平台的方位音频库。如需进一步了解，请访问 http://www.openal.org。
OpenGLES.framework	2.0	EAGL, GL	包含OpenGL ES接口。OpenGL ES框架是OpenGL跨平台2D和3D渲染库的跨平台版本。请查看OpenGL ES 框架参考。
QuartzCore.framework	2.0	CA	包含Core Animation接口。请查看Quartz Core 框架参考。
QuickLook.framework	4.0	QL	包含预览文件接口。请查看Quick Look 框架参考。
Security.framework	2.0	CSSM,Sec	包含管理证书、公钥私钥以及信任策略的接口。请查看Security框架参考。
StoreKit.framework	3.0	SK	包含用于处理与应用程序内购买相关的财务交易。请查看 Store Kit 框架参考。
SystemConfiguration.framework	2.0	SC	包含用于处理设备网络配置的接口。请查看System Configuration框架参考。
UIKit.framework	2.0	UI	包含iOS应用程序用户界面层使用的类和方法。请查看UIKit 框架参考。

模拟器的框架

虽然编写代码应该面向设备框架，但是在测试的过程中，您也需要针对模拟器编译代码。设备和模拟器的框架稍有区别。模拟器将几个Mac OS X框架作为其自身实现的一部分。另外，由于系统的限制，设备框架的确切接口有可能和模拟器框架稍有不同。如果您需要这些框架的列表以及设备和模拟器框架之间的差异信息，请查看iOS 开发指南。

系统库

请注意，iOS系统可能没有将Core OS和Core Services层某些特殊的库打包成框架，而是将其作为动态库放在系统的/usr/lib 目录。动态共享库通过.dylib扩展名标识，其相应的头文件位于/usr/include目录。

所有版本的iPhone SDK都包含一份安装在系统的动态共享库本地副本。这些副本被安装在您的开发系统，您可以从XCode工程进行链接。如果您需要查看某个版本的动态库列表，请查看<Xcode>/Platforms/iPhoneOS.platform/Developer/SDKs/<iOS_SDK>/usr/lib。在这个路径中，<Xcode>表示Xcode的安装目录，<iOS_SDK>表示您当时正在使用某个版本的SDK。举个例子，iOS 3.0 SDK的动态库位于 /Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS3.0.sdk/usr/lib目录，相应的头文件则位于/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS3.0.sdk/usr/include目录。

iOS使用符号链接来指向程序库的最近版本。在链接某个动态共享库的时候，请使用符号链接而不要使用动态库特定版本链接。因为在将来的iOS版本中，库的版本可能会发生改变。如果您的软件连接到某个特定的版本，而那个版本可能已经不存在于用户的系统当中，那就会出问题。