互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

第3章

使用Apple的开发工具

在被 Apple 买下之前，NeXT 公司的生意几乎全部都是卖开发工具。它的硬件产品线中止了，操作系统也没有得到广泛应用。然而它的开发工具，买得起的人都很喜欢。

OS X 最初发布的时候就包含了 NeXT 开发环境的两大基石：Project Builder 和 Interface Builder，而且几乎没做什么修改。虽然 Apple 计算机出了名的贵，但即便是最贵的 Mac，其实也只不过跟 NeXT 出售的 Windows 版本 OpenStep 及其开发工具的价格差不多。

这些年来 Apple 重写了其中的很多代码。Interface Builder 在 OS X 10.5 时被完全重写，而更早以前，Project Builder 就已演变成了 Xcode。Xcode 最近的版本里，Apple 开始使用新的 LLVM C、Objective-C 和 C++ 前端——Clang，不仅用于编译，而且用于重构和静态分析，以及语法高亮。Clang 的设计目标之一就是对源代码进行足够快地解析和语义分析，从而可在代码编辑器后台周期性运行，提供实时的诊断信息和语法高亮。

大多数的代码编辑器实际上做的并不是语法高亮，而是词法高亮。它们切分（tokenize）输入文件，给关键字和注释加上颜色。而 Xcode 中的功能全面的语法高亮器可以给类名不同于其他类型的颜色，根据作用域范围给变量不同的颜色。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

这种方法的一个很大优势是具有能完全理解 C 预处理语句的完整解析器，这一点很多代码编辑器都做得不好。

3.1 获取Apple的开发工具

Mac 附带的安装 DVD 包含的往往是一个老版本的开发工具。虽然在创建这个 DVD 时它是最新版本的，但是 Apple 会经常发布更新的版本。因为这些工具的更新机制是下载全新版本然后安装，所以安装之前最好先检查其是否最新版本。

开发工具可以从 Apple 开发者网站下载，地址是 http://developer.apple.com/mac/。标有 Xcode 的包就包含了 OS X 开发所需的全部工具。可能需要注册一个免费的 Apple Developer Connection（ADC）账号。这一过程中需要同意一些条款，可以接受或者不接受。如果不愿意接受，还可以（虽然可能更难些）使用自由软件工具在 OS X 上面构建软件，参见 3.3.4 节。

3.2 Interface Builder

Interface Builder 可能是 OS X 的程序中名字起得最差的一个。如果 Apple 在别处也遵循这一命名方式，那么 iTunes 可能会叫做 CD Ripper，OS X 自身可能会叫做 TextEdit Launcher。

虽然 Interface Builder 能够并且擅长构建界面，但这仅仅是其一小部分功能。GNUstep 提供了略为粗糙的替代品，叫做 Graphical Object Relationship Modeller（GORM），这个名字更确切，虽然比较烦琐。GORM 可以在其他平台为 OS X 构建界面，如果可以使用 OS X，Interface Builder 的界面要更友好些。

Interface Builder 生成的是 nib 文件。这是从 NeXT 版本 Interface Builder 继承来的原始扩展名，在 Cocoa 中用得很多。nib 文件包含了一个序列化的对象图。这些对象通常都是用户界面对象，但不全是。即便在用户界面 nib 文件内，通常除了视图对象也还有控制器对象。OS X 10.5 前，Interface Builder 内部也使用 nib 文件，最新版本改为使用 xib 文件，以简单的 XML 格式来展现相同的信息，在程序构建时再转换成 nib 文件。

整个 nib 文件都可以由程序来解析展现。于是可以非常简单方便地创建一组常见对象集的副本。这一点在文档驱动的程序中最常见，nib 文件可以包含构成单个文档对象实例所需的所有对象。然后，文档对象可以用一行代码创建出来。

互动出版网
China-Pub.com
专业图书网上第一专营店
www.china-pub.com
独家提供样章

图 3.1：打开了一个新应用程序项目的 Interface Builder

模型—视图—控制器模式

模型—视图—控制器（Model-View-Controller，MVC）模式在 Cocoa 中广为使用。视图是用来表现用户界面的对象，包含很少的一些状态。模型是用来表现抽象数据的，如一个文档或者文档的一部分。控制器用来连接以上两者。在 Cocoa 最近的版本中绑定（binding）的使用在很多情况下避免了对控制器的需求。

遵循 MVC 模式意味着程序逻辑和用户界面将完全分离。这样程序移植到别的平台时就会更加容易。例如，把一个设计良好的 Cocoa 程序移植到 iPhone 平台，只需针对新设备的小屏幕写些优化的界面代码，大多数代码都无须改动。

NSNib 类是 nib 文件的主要接口。用这个类载入 nib 文件有两个步骤。创建该类的实例，把 nib 文件载入内存备用，但并不真的创建包含的对象。接下来调用这个对象，创建 nib 存储的对象的新副本。可以用这一方法快速创建一组对象。

大多数时候，不需要直接使用 NSNib。应用程序对象会载入程序的主 nib 文件，并实例化里面包含的所有对象。NSWindowController 类提供了载入对应窗口 nib 文件的接口。在用 nib 文件创建 NSWindowController 实例时，它会把自己设为 nib 的拥有者，并创建 nib 中所有对象的副本。

每个 nib 文件都有一个伪对象，在 IB 中叫做文件属主（file's owner）。这个对象不是 nib 文件的一部分，在创建对象图的时候，它必须在 nib 文件外创建，然后传递给 NSNib。一般情况下，nib 包含一个窗口、窗口内的一组视图，以及和

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

这些视图相关的控制器，而文件属主一般会采用 NSWindowController 的子类作为窗口控制器。

3.2.1 Outlet 与 Action

可以在 nib 中创建任何对象，不论它是否支持归档。对象由两种接口连接：outlet 和 action。outlet 就是实例变量，在 nib 文件载入时指向对象。action 是消息，某些动作发生时就会发出。

视图通知控制器自己的变化有两种方法：一种是通过 action；另一种是通过 delegate。action 相当简单，由 NSControl 类实现，提供一个简单的机制通知控制器有些事情正在发生。nib 文件载入系统在创建视图时会对其发送两个消息：-setTarget: 和 -setAction:。第一个调用告诉视图应该给哪些对象发送 action 消息，第二个则告诉它应该发送那些对象的哪些消息。这些在 Interface Builder 中可以可视化地连接。图 3.2 显示了用来选择连接 outlet 或者 action 的检视器。

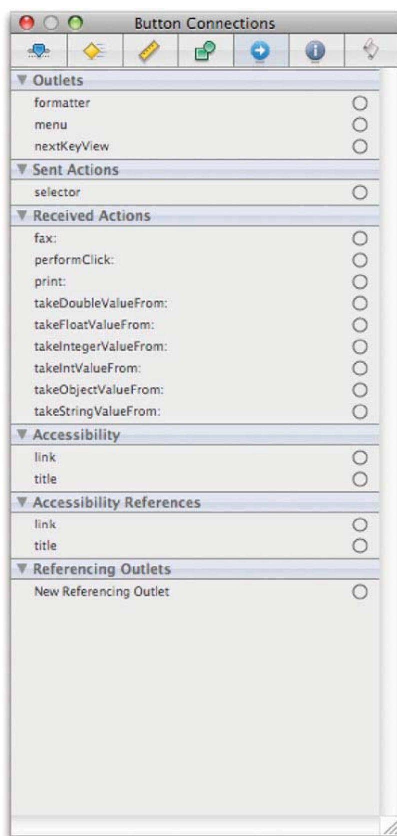

图 3.2：Interface Builder 中的 outlet 和 action 检视器

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

这一机制用来传递按钮按下之类简单的事件。action 消息将发送者作为其参数。对于按钮按下这种简单的事件这就足够了，但对复杂的视图，尤其是有很多不同的交互可能发生时就远远不够了。

这个时候就要使用 delegate 了。delegate 是控制器对象，用于响应一组由视图定义的消息。因为 Objective-C 很轻松地支持了迟绑定及自省，所以并不总是需要实现 delegate 定义的全部消息。Java 程序员可能熟悉的一幕是，在实现 Listener 族接口的类里，有着大量不包含任何代码的方法，它们的存在只是为了满足接口规范的要求。在 Cocoa 中，这种事情不会发生；视图会检查 delegate 响应哪些消息，然后只发送这些消息。

delegate 是 outlet 的一例。这也是借助 Objective-C 的自省能力实现的。可以在运行时自省一个类，找出其包含的实例变量、它们的类型，以及相对对象起点的偏移。这就是载入 nib 文件时发生的事情。命名实例变量的偏移被检查，然后被设置为那个对象。这也是松散耦合（louse coupling）的一个例子。可以重新编译对象，改变其内存布局，但是不需要改变 nib 文件。只要实例变量的名字和类型正确，nib 文件就可以继续正常工作。

这种类型的松耦合在 Cocoa 中比比皆是。这有两个优势。首先，它意味着修改了程序的一个部分，很少需要重新编译其他部分，从而令快速的编写—编译—测试循环得以实现。当然这一点不像当年运行在 25MHz 系统时那么明显了，但是对大的项目依然重要。另外一个优势就是，Cocoa 的代码很容易写得通用和可重用，从而节省大量开发时间。

3.2.2 Cocoa 绑定

OS X 10.3 版本引入了两种相关的机制。第一个是 key-value coding（KVC）。这一相当简单的机制提供了对对象属性统一的访问接口。它提供了两个方法：一个用来设置值，另一个用来获取值。在底层提供了直接访问实例变量、调用 set 和 get 方法，以及调用回退机制等机制。

KVC 是一种便捷抽象机制，真正酷的是 key-value observing（KVO）机制。它让其他对象可以请求在指定的 key 关联的值发生变化时得到通知。有很多 Cocoa 类支持这两个机制，NSObject 这个根类中包含了在自己的类中支持它们所需的基本构架。

KVC 和 KVO 的结合，再加上一点 Interface Builder，叫做 Cocoa 绑定（binding）。绑定利用了 KVC 的组合实现，当一个对象的属性变化时，就设置另外一个对象的属性。这个机制使得可以省去大量的控制器对象。如果模型对象支持 KVO，视

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

图支持 KVC，那么视图可以在模型变化时自动更新。同样，如果模型支持 KVC，视图支持 KVO，那么模型也可以根据视图的改变而更新。在 Delicious Library 程序开始使用绑定时，用可重用的通用绑定代码代替自定义的控制器对象，使作者得以一天内删除了数千行代码，结果不仅功能更强，还解决了一些 bug。

绑定通常也会带来些开销。它们比用直接的消息传递机制来操作明显慢多了，而且还需要一些额外的内存空间。但是对于用户界面代码，它们几乎不可能是瓶颈所在。在窗口上描绘一行简单的文本也比绑定执行的操作慢得多。

指令缓存性能

现在做一个公平的基准测试非常困难的原因之一，是代码的性能高度取决于缓存的使用。访问内存远远慢于访问寄存器或者缓存，需要数百个 CPU 循环。微基准测试中慢的通用代码，可能在实际使用中会稍快一些。如果通用代码在很多地方都用，那么它们可能会待在 CPU 的指令缓存中，所以也许在 CPU 等待高度优化的代码从内存载入的时间内，它们就已经执行完了。如果想手工优化特化代码请记住这些，最终代码可能耗内存更多、更慢，这可不是打发时间的好方法。

3.2.3 绘制简单的应用程序

Objective-C 的松耦合令其可以创建简单的，以及某些非常复杂的行为，而不需要写任何代码。打开 Interface Builder，会看到一个空窗口和组件面板。可以从组件面板拖拽一个视图到窗口，然后改变其大小。

图 3.3 展示了一组视图。面板左下角的按钮，可用来修改视图列表的表现形式。学习的时候，可能觉得同时显示图标和描述会更加方便，可以方便地看到描述。但是在熟悉这些对象以后，会发现只显示图标更为快捷方便。

对于这个简单的例子，拖拽一个文本框和水平滑选器到窗口上。

对于窗口中的这两个视图，可以把它们连接到一起。按住 Control 键，用鼠标从其中一个拖动到另外一个。松开鼠标按钮后，会显示一个弹出的半透明窗口，窗口内包含了 outlet 和 action 的列表，如图 3.4 所示。从接收 action 的列表中选择 takeIntegerValueFrom:，然后反过来拖动做同样的选择。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

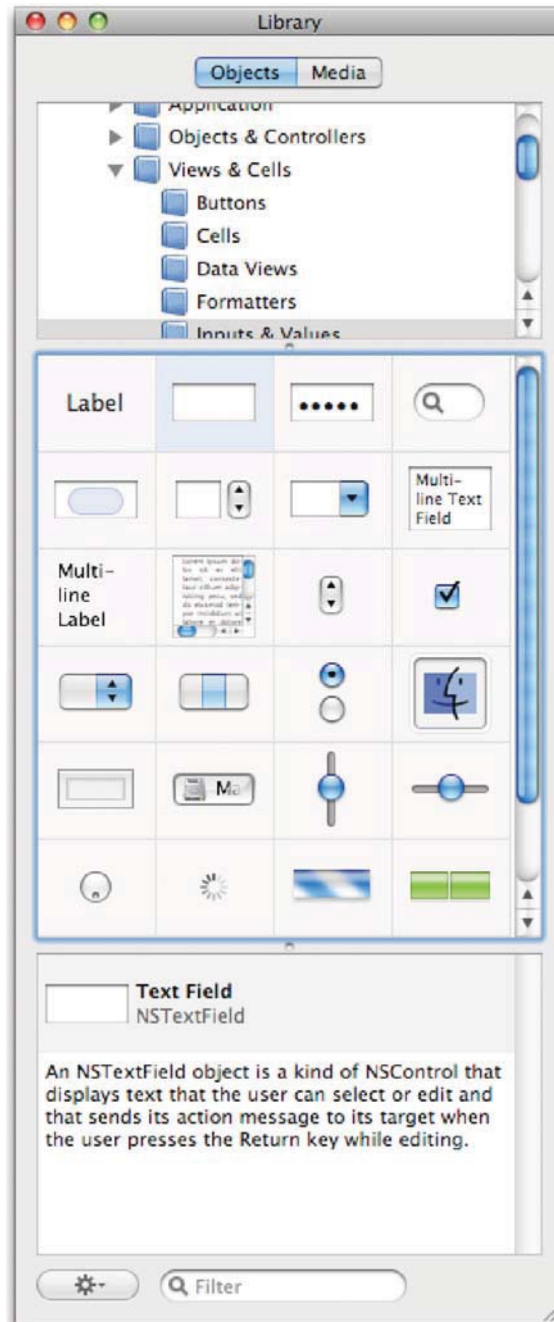

图 3.3：Interface Builder 的 Views 面板

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

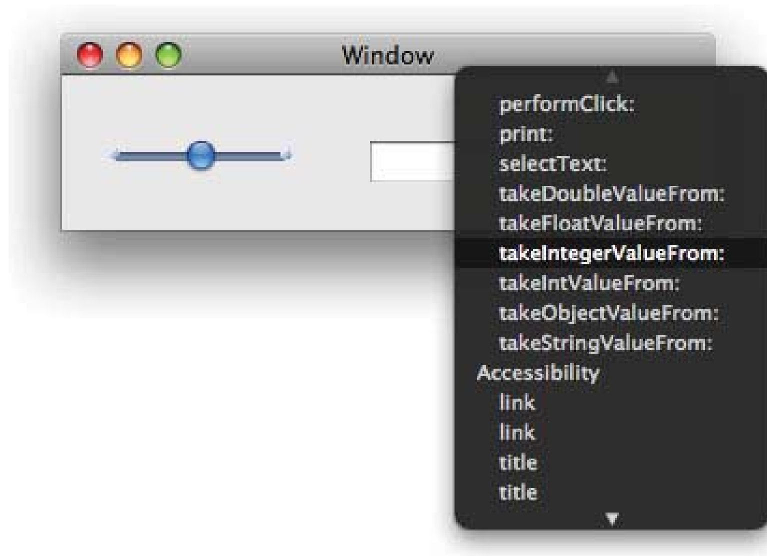

图 3.4：连接一个 action

现在，不管哪个视图变化，都会发送 takeIntegerValueFrom: 消息给另外一个。因为这是一个 action 消息，它会把发送者作为参数。接收者会发送一个 integerValue 消息给发送者，并把自己的值设置为返回的值。

两个对象都不用了解对方的任何细节。action 消息不是硬编码在它们之上的，而是在 nib 载入时设置的。它们也不知道对方的内部会如何展现其数据。尽管滑动条用浮点值来保存其位置，而文本框用字符串存储其中的内容，但由于它们都实现了 integerValue 方法，因此都可以获得其值所对应的整数值。

在 Interface Builder 中按下 Command+R 键，可以测试界面，如图 3.5 所示。移动滑块，文本框里面的数字会发生变化。而在文本框中输入一个数值，滑块的位置也会发生变化。

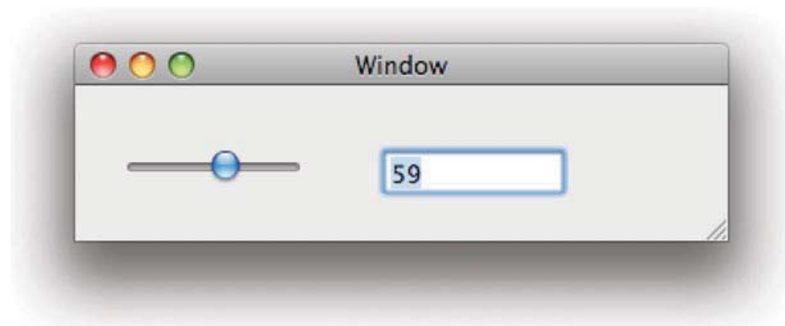

图 3.5：运行一个简单的界面

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

这说明了两件事。首先，它表明在 Cocoa 中经常可能不用写任何代码就实现功能。这是好事儿，因为任何新代码都可能包含 bug，而老代码往往经过了更全面的测试，尤其是这些老代码来自 Cocoa 框架，在 OS X 中到处都在用。

其次，这是真实的对象集合。当nib文件载入时，真实的Cocoa视图对象被创建，并被连接起来。虽然没有写任何代码，但是这些对象是 Cocoa 类的实例。它们之间的连接由 Objective-C 来控制，而不是由更上层的什么对象语言来控制。这也充分说明了 Objective-C 促进代码重用的能力。两个类都不需要任何特殊代码就可以跟对方通信。可以轻松地用自己的视图对象替换其中任何一个，而仍使用相同的机制。

3.3 Xcode

OS X 10.3 中包含了对开发环境的大量改善。其中包括对 Objective-C 语言的扩展、对 Cocoa 对象的很多改善，以及新的开发工具。特别是，用 Xcode 集成开发环境替换掉了 Project Builder。

Xcode 提供了编译构建系统、代码编辑器和调试器。Interface Builder 还保持为独立程序，虽然两者的交互非常紧密。调试工具构建在 GNU Debugger（GDB）之上。这是一个命令行程序，Xcode 用管道跟它交互。因为 Xcode 的图形界面没有暴露 GDB 所有的能力，所以你可以直接发命令给它，这很有用。花点时间阅读 GDB 的手册可以让你以后省很多力气。

3.3.1 创建简单的项目

在创建一个新项目的时候，会看到如图 3.6 显示的欢迎窗口。它提供了很多模板项目。大多数情况下，可以选择 Cocoa Application 或者 Cocoa Document-based Application。系统会询问保存文件的位置，然后在该目录中生成骨架项目。

骨架项目仅包含所有项目都需要的一些文件。包含一个 MainMenu.xib 文件，这个文件包含了程序的主菜单和窗口，构建程序的时候会编译为 MainMenu.nib 文件。可以用 Interface Builder 来定制它。

由 Cocoa Application 模板创建的唯一源文件叫做 main.m，提供了程序的入口点。这一文件很少需要修改。可以创建另外的类，并在主 nib 文件内把它们和你的程序对象连接起来。选择菜单 New File，可看到如图 3.7 所示的选项。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

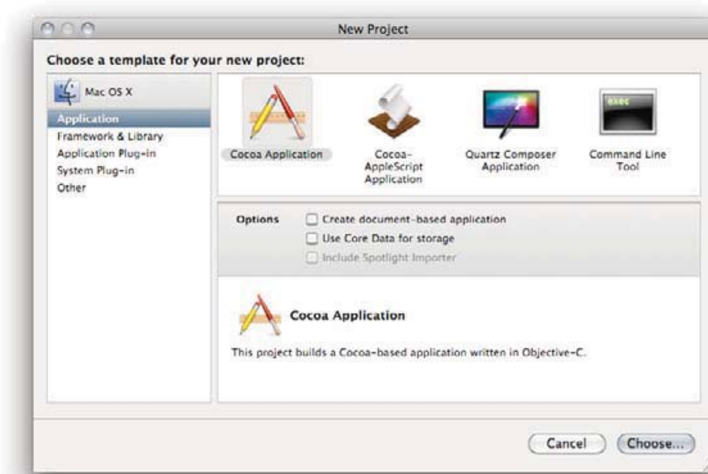

图 3.6：在 Xcode 中创建新项目

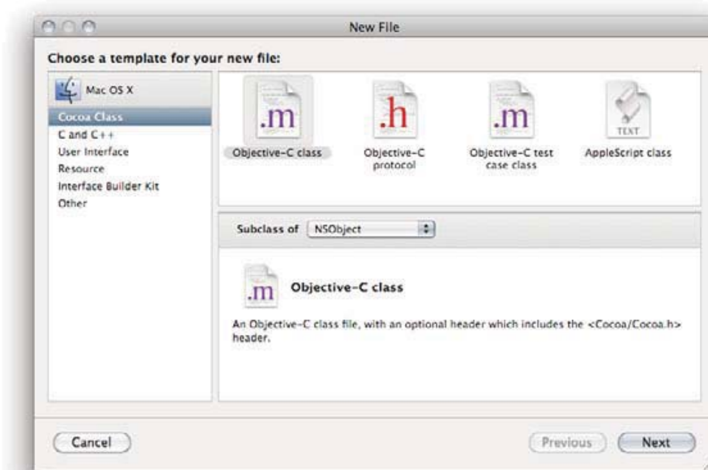

图 3.7：在 Xcode 中创建新文件

如果创建新 Objective-C 类，可以选择是否包含相应的头文件。通常都希望这样。Cocoa 会在 .m 文件中放入类的骨架实现，而在 .h 文件中放入类的骨架接口。

Xcode 的主窗口如图 3.8 所示。包含三个面板。左边的面板包括了当前项目包含的所有东西，不全是源代码。程序可以有多个不同的 target，它们包含了用来构建 product 的规则，target 通常是程序或者框架。可以在这里创建智能分组，来包含项目中匹配用户自定义条件的所有文件。选择左侧面板中的分组，相应内

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

容就会显示在右上方的面板内。

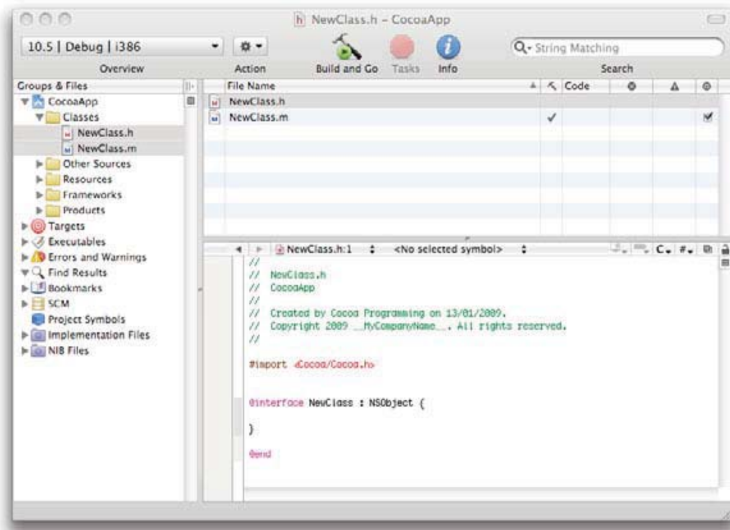

图 3.8：Xcode 的主窗口

有两种使用 Xcode 的方式，整个项目使用一个编辑器，或者每个文件使用一个。双击某个文件，该文件将在新编辑器中打开。单击文件，则该文件会在窗口内嵌的编辑器中打开。Xcode 提供的其他窗口，如调试器和构建窗口，也是如此。

如果不使用内嵌的编辑器面板，可以将其折叠，给其他的面板留出更多空间。使用 Cocoa 代码编辑器的时候，有几个非常方便的键盘快捷方式应该记住。如果按住 Command 和 Option 键，然后按下一个方向键，就会打开另一个文件。按下上键会在同名的接口和实现文件之间切换。如果正在编辑 MyObject.m 然后按下 Command+Option+ 上键，就会切换到 MyObject.h。每个文件都有独立的撤销（undo）缓冲区，所以切换不会对其他文件的撤销操作构成影响。编辑器还维护有其中编辑过的所有文件的历史记录。按住 Command 和 Option 键，再按键盘左、右键，可以在历史记录中前后切换要编辑的文件。

3.3.2 OpenStep Bundle

传统 MacOS 的自定义特性之一是文件系统的分支（fork）概念。一开始，HFS 文件系统支持每个文件两个分支：数据分支和资源分支。数据分支包含了在其他系统中文件也同样会包含的普通的流数据，而资源分支则包含了一个资源的结构体。通常用来保存图标或者其他资源。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

从用户角度来看分支是一个非常好的模型，因为这意味着程序（或者其他类型的文档）对用户来说就是单个文件，即使它们包含了很多其他的组件。可以直接将其拖到其他硬盘，而不用担心落下任何部分。

NeXT 考虑过这一模型，但是最终放弃了，因为这太依赖文件系统的特别支持。如果把一个程序复制到 DOS 硬盘，或者通过 NFS 共享，资源分支就会丢失。于是它们用文件包（bundle）来替代。从文件系统的角度看，文件包就是个目录。即便在不提供支持的系统上对其进行复制或者修改，也不会丢掉任何东西。

不过，HFS+ 文件系统还是包含了对文件包的一些特殊支持。使用一比特元数据来表明某个目录是不是一个包。可以用 GetFileInfo 工具查看 HFS+ 文件系统的标志。

```
$ GetFileInfo example.key/
directory: "/Users/CocoaProgramming/example.key"
attributes: avBstclinmEdz
```

上面显示了一个 keynote 包的属性。小写字母代表标志未设置，大写字母代表标志已设置。对于这个包，B 和 E 标志已设置，表示这个目录是包，而且其扩展名（.key）在 Finder 程序中会隐藏。你可以注意到这些标志对程序则没有设置。

```
$ GetFileInfo TextEdit.app/
directory: "/Applications/TextEdit.app"
attributes: avbstclinmedz
```

这是因为 OS X 设计为可以运行在其他的文件系统上，所以所有标准包格式，如 .app，在 Finder 程序中有特殊的逻辑来处理。如果目录有已知的包扩展名，或者设置了对应的标志，那么 Finder 程序就会把它显示为文件，尝试打开的时候就会用工作区功能，而不是在双击后直接显示其包含的内容。

文件包在 OS X 中应用广泛，如果要创建文档驱动的程序，你可能也想定义自己的包类型。

作为开发者，最常遇见的文件包应该是 framework。这是包含库和头文件的文件包。它们也可以包含资源，例如声音和图像，在代码中可以使用 NSBundle 类来访问。

3.3.3 开发样例

在安装了 Xcode 开发工具之后，可在 Developer/Examples 目录下发现很多例子，这些例子的授权非常宽松，可以任何方式处理，包括将其加入私有或者开源软件中，只要不把修改后的代码作为 Apple 例子发布即可。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

这一文件夹下的每个子目录都包含了跟 OS X 某个特定部分有关的例子。最重要的一些在 AppKit 目录下，包含了跟 Cocoa 应用程序工具包框架相关的例子，这一框架几乎是所有 Cocoa 程序的核心。

还有 OS X 下其他语言的例子。Java、Perl、Python 和 Ruby 目录包含了这几种语言的例子。在 wxWidgets 目录下，包含了 Perl 和 Python 语言使用跨平台 wxWidgets 框架开发 OS X 程序的例子。

其中部分例子非常底层。例如，IOKit 目录包含了如何在 OS X 下编写设备驱动的例子。其他的与特定的框架有关，如 WebKit——为 Safari 开发的 HTML 渲染框架，在 OS X 的很多部分都在应用。

花点时间玩玩这些例子是很值得的。仅仅编译并执行，就可以从中获得启发，在 OS X 下可以做些什么样的东西。注意，这些例子的权限都设置为只有管理员账号才能编译，因为需要在 Build 目录写入内容。如果不是管理员账号，那么先把它们复制到别的目录，然后再编译。

3.3.4 不用 Xcode 来编译

NeXT 系统上的 Project Builder 和 Xcode 的早期版本都使用 GNU Compiler Collection（GCC）编译程序。[1] 伴随 OS X 发行的版本里面的 GCC 和自由软件基金会发行的版本不完全一样，虽然它是从基金会的代码管理系统中的分支发展而来的。这两者经常分别被称为 Apple GCC 和 GNU GCC。某些特性会先出现在其中一个里，所以在两者间迁移代码时，如果使用版本宏判断来测试编译器特性，可能会出问题。

从 Xcode 3 开始，Apple 还加入了新的编译器——LLVM GCC。它使用一个 GCC 的修改版本来做代码分析，用 Low-Level Virtual Machine（LLVM）来做代码生成。这是最初版本的 SDK 里，唯一支持做 iPhone 平台软件开发的编译器。GCC 并非设计为可以被分开，被这样使用的，所以 Apple 在制作一个新的前端——Clang，它可以分析 C 语言、Objective-C，以及 C++ 代码。这一切换部分的原因是技术，部分则是法律。GNU GCC 4.3 版本基于 GNU General Public License（GPL）版本 3 分发。Apple 没有基于这个授权分发任何代码，所以它的分支将无法合并来自主线版本中的变化。

这些编译器都可以用于命令行。如果你来自 UNIX 平台，你可能很熟悉 GCC，以及它的命令行选项。OS X 加上了一个 -framework。它指明了一个程序

1 与Linux和BSD系统不同，OS X包含有自己的链接器，不使用GNU ld。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

链接所需要的框架。框架可以包含头文件和库文件，所以这个参数可以跟 -I、-L 以及 -l 组合。

虽然你可以直接用 make 命令构建你的 OS X 程序，但是那跟 Xcode 很难结合。[1] 而使用 xcodebuild 工具，可以使用命令行构建 Xcode 项目。这个工具也被 Xcode 用来构建你的项目；它的输出就是你在构建窗口看到的那些东西，如果你选择查看细节就会看到它们。

与 make 不同，xcodebuild 会区分目标和动作。目标是一组构建出的产品，而动作是对目标做的一些事情。三个最常见的动作是 build、install 和 clean。它们分别创建目标，安装目标，删除中间结果。

你可以在包含单独 Xcode 文件的目录中不用任何选项地运行 xcodebuild。

```
$ xcodebuild
=== BUILDING NATIVE TARGET Presenter OF PROJECT Presenter WITH THE
DEFAULT CONFIGURATION (Release) ===

Checking Dependencies...

{lots of output}

** BUILD SUCCEEDED **
```

这样做会根据 Xcode 中的默认配置构建项目。这个工具有很多命令行选项，全部文档化在 man 中。如果你想自动进行每夜构建，应该考虑用它。但对大多数人而言，在 Xcode 中点 build 按钮会更方便。

3.4 Objective-C

Objective-C 是 C 语言的纯超集，设计上允许使用 Smalltalk 式的面向对象编程，支持代码重用。这个语言反映出来的几大核心原则是：

兼容性。C 语言合法的代码在 Objective-C 中就是合法的。这令重用已有代码变得很简单。你不需要 extern "C" 声明，直接调用或者嵌入 C 语言代码即可。所有新关键字都是由 @ 符号开头的，不会跟现有的任何标识符混淆。

没有魔法。Objective-C 的编译器非常简单，仅仅是替换对运行时库的调用。所有的机制对程序员都是开放的，编译器可以做的事情你都可以用自己的代码来做。语言中的一切都可以在库中实现。

1 如果你希望在OS X下使用make构建程序，GNUStep Make有方便这么做的预定义规则。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

没有不需要的复杂度。Objective-C 易学易用。不同于 C++，这个语言很小，一个 C 程序员可以在一个下午内学会。

新的语义采用新的语法。Objective-C 没有重载任何 C 语言原有的语法。加入 C 语言不能表达的功能时，将使用新的关键字或者新的语法元素。

“没有魔法”原则的一个效果是，这个语言中全部支持自省。编译器知道的事情，程序员都可以知道。你可以枚举一个类的全部方法和实例变量，只要你愿意，甚至可以在运行时创建一个新类。

3.4.1 为何学习 Objective-C

Objective-C 是一组基于 C 语言的 Smalltalk 式的面向对象语法的拓展。这个语言很简单，只有一项主要的新增语法（消息传递），和一些很有用的关键字。与广为流传的看法不同，使用 Objective-C 不会把你锁定在 Apple 平台上。在 OS X 10.5 中，Apple 发布了两个 Objective-C 编译器：gcc 和 llvm-gcc。它们都是开源的，虽然 llvm-gcc 仅能在 OS X 下编译 Objective-C。第三个编译器 clang，在 OS X 10.6 中的 Xcode 3.2 版中引入。Clang 是目前唯一可以在非 Apple 平台上，编译使用 Objective-C 新特性的代码的编译器。

设计 Objective-C 是为了在廉价的机器上，给 C 语言程序员带来 Smalltalk 的灵活性。这个语言的创造者 Brad Cox，在 20 世纪 80 年代早期写了很多关于组件化软件设计的论文，设计 Objective-C 就是为了鼓励独立组件的开发。

Steve Jobs 在离开 Apple 后，发现了这个语言的潜力。他创建 NeXT 公司的目的，是要利用当前科技生产最接近完美的计算机，而一个动态的、模块化的系统是这个目标的主要组成部分。NeXT 公司，基于它们曾经与 Sun 公司的合作，设计了叫做 Cocoa 的 API。这个 API 在 OS X 上由 Apple 实现，此外还有其他项目，如 GNUstep 和 Cocotron，在其他平台实现了这个 API。

尽管 Objective-C 不是唯一可以使用 Cocoa 的语言，但却是 Cocoa 设计支持的语言。大多数其他语言的支持都是通过桥接实现的，或者其他不够完美的映射。Objective-C 还是唯一支持所有实现 Cocoa API 平台的语言。

3.4.2 对 C 的添加

作为 C 语言的纯超集，Objective-C 可以用它对 C 语言的扩展来定义。大多数部分对见过 Smalltalk 的人来说都很眼熟，但是有些是 Objective-C 所独有的，用来帮助 Smalltalk 对象模型和 C 语言语法相结合。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

这个章节介绍 Objective-C 的概况，以及大多数特性。

Objective-C 提供的扩展有的不是加在 C 语言之上的。具体地说，有加到预处理器上的。如 #import 指令是 #include 的一个变体，可以确保头文件在编译单元内只被包含一次。这意味着 Objective-C 的头文件不需要用 #ifdef 块来保护，它们总是只被包含一次。

Objective-C 还引入了很多新的类型。它们都是用 typedef 定义在头文件中的，所以实际上算不得语言的扩展，但是很重要。id 类型用来表示一个对象的指针。你也可以使用一个具体的类型，如 NSObject* 来表示一个具有特定类型的对象指针。

这还包括了一些附加的转换规则。id 类型大致上等于 C 语言中的 void* 类型，只是指向的必须是一个对象的指针。你可以隐式地把 id 转换成，或者转换自其他对象类型。此外，你还可以隐式地把对象的指针转换为其任何一个父类的指针。任何其他类型的转换，就都需要显式地转换了，就像 C 语言那样。

加入的另外两个类型是 Class，表示对象结构的指针，以及 SEL，表示 selector。selector 是一个消息名字的抽象形式。在 Objective-C 中传递消息的时候，消息查找的形式是基于 selector 的。在编译期间，可以用 @selector() 语句生成 selector，如下：

```
SEL newsel = @selector(new);
```

这就给出了消息名为“new”的 selector。运行时，你可以使用 NSStringFromSelector() 函数获得一个 selector 的消息名。

> **哪个 C？**
>
> C 语言有三种常用的方言。原始的、未标准化的 C 语言，通常叫做 K&R C，名字来自语言的设计者，主要用于遗留代码中，新项目要避免使用。有两种标准化的 C 语言方言，C89 和 C99，名字来自标准化的年份。还有一个新版本，C1X，即将成为标准。
>
> Objective-C 是 C 语言的超集，但是具体采用哪个 C 语言的方言取决于你自己。Xcode 允许你自行选择，但是一般没有必要选择 C99。

消息传递

Objective-C 最重要的部分是消息传递。Objective-C 对象间的交互都通过消息传递来进行。默认情况下，消息分发是同步的，就像函数调用一样，不过某些情况下，它们也可以异步进行。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

消息传递是高阶版本的函数调用。不同的是，消息传递是迟绑定的。这就是说，消息发送之前，用来响应一个消息（方法（method））的代码是不确定的。这和 C 函数调用形成了鲜明对比，C 语言中调用的代码在载入或者编译期间就决定了。

> **方法和消息**
>
> 讨论 Objective-C 代码的时候，“发送一个消息”和“调用一个方法”经常互换使用。大多数情况下，它们大致相同。发送一个消息的结果是同名的方法被调用。但也有例外，有时候，发送一个消息的结果是，消息被保存，或在网络中传递，或调用一个完全不同对象的方法。

因为消息传递语义不同于函数调用，所以它采用了新的语法。粗看起来，这让 Objective-C 的代码看起来比 C++ 还要怪异，但是从长期看，它令代码变得更加易读。例如下面的 C++ 代码：

```
object.doSomething();
```

这有三种可能性：

1. 可能 object 是个 C 语言的结构体，那么它会调用 doSomething 字段包含的函数指针所指函数。

2. 如果 object 是 C++ 对象，doSomething 没有声明为 virtual，那么调用 object 所属类的 doSomething() 方法，传递隐含的 this 指针指向 object。

3. 如果 object 是 C++ 对象，doSomething 声明为 virtual，那么动态查找 object 的类，调用 doSomething() 方法，传递隐含的 this 指针指向 object。

要知道到底是哪种情况需要读很多行的代码。更严重的是，如果 object. 前缀不存在，那么就既有可能是对当前对象的那两种调用，也有可能是一个 C 函数。Objective-C 没有 C++ 的非 virtual 方法的等价物，因为那语义上等同于 C 函数。下面两行在 C++ 中实现方式是一样的（如果 doSomething() 不是 virtual 的）：

```
doSomething(object);
object.doSomething();
```

起初第二行的写法被当做语法糖引入，但是要多敲一个字符（如果 object 是指针就要多两个），更精确地说应该算做语法盐吧。Objective-C 的消息传递使用 Smalltalk 语法。上面例子可以写为

```
[object doSomething];
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

消息的参数和消息名连在一起，如下：

```
[dict setObject: anObject forKey: aKey];
[dict removeObjectForKey: aKey];
```

方括号是 Smalltalk 语法的变种。通过它们编译器可以轻松地分清纯 C 语法和 Smalltalk 式的语法。它们还清晰地表明了消息的边界，你可以很清楚地看到 setObject: 和 forKey: 是同一个消息的组成部分。

方括号语法是 Objective-C 的一个最初设计者 Tom Love 强烈提倡的。他甚至把这叫做“面向对象领域的离合器”，意为外面的一切是 C 语言语法，而里面的是 Smalltalk 式的语法。所以 Objective-C 代码比 C++ 更加易读；Objective-C 中你可以轻松地分清楚消息传递和函数调用，而在 C++ 中你需要非常了解代码和库才可以。

消息传递有三个组成部分：接收者，选择器和（可选的）参数。接收者就是接收消息的对象，在这个例子里面就是 dict。选择器是方法名的唯一版本。Objective-C 加入了一个新的关键字，@selector，用来在编译期间通过字符串获得一个选择器。在运行时你可以使用 NSSelectorFromString() 函数来获得选择器。

消息传递有一个特殊的情况。如果对 nil（或者其他的 0 指针）传递消息，结果是 nil，0 或者一个填满 0 的结构体，取决于调用者希望获得的返回值。

类

Objective-C 对 C 语言的两大扩展是类和消息传递。在 Objective-C 中，对象是一个结构体，第一个元素是指向类的指针。在 10.5 之前，类结构是公有的。在代码中，你可以看到，如果使用 Objective-C 1 来编译，它还是可见的，但是用 Objective-C 2 来编译就不会这样。

从 Objective-C 2 开始，类结构不再透明。你仍旧可以操作它，但是现在必须通过运行时函数来操作。这样未来 Apple 给类加入新的字段时，就不用担心破坏 ABI（应用程序二进制接口）。

虽然这个结构未来注定要改变，但是仍旧给了我们类数据如何存储的概况信息。其中三个结构类型名以 _list 结尾。类的三种类型的属性：方法，协议和实例变量（ivar），它们都是列表类型的。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

代码清单 3.1：Objective-C 的类结构（取自 /usr/include/objc/runtime.h）

```
43 struct objc_class {
44     Class isa;
45 
46 #if !__OBJC2__
47     Class super_class OBJC2_UNAVAILABLE;
48     const char *name OBJC2_UNAVAILABLE;
49     long version OBJC2_UNAVAILABLE;
50     long info OBJC2_UNAVAILABLE;
51     long instance_size OBJC2_UNAVAILABLE;
52     struct objc_ivar_list *ivars OBJC2_UNAVAILABLE;
53     struct objc_method_list **methodLists OBJC2_UNAVAILABLE;
54     struct objc_cache *cache OBJC2_UNAVAILABLE;
55     struct objc_protocol_list *protocols OBJC2_UNAVAILABLE;
56 #endif
57 
58 } OBJC2_UNAVAILABLE;
```

isa 指针指向类的元类（metaclass）。正如对象可以接受的信息是由类定义的一样，类可以理解的信息由元类定义。Objective-C 类也是对象，你可以像给对象发送消息那样发送消息给一个类。不要把 isa 指针和 super_class 指针弄混。super_class 指针指向的是一个类的父类。如果你给一个对象发消息，首先会在它的方法中查找，然后会在父类的方法中查找，依此类推。而元类只跟发往类的消息有关，跟类的实例没有任何关系。

其他的字段大多是类的元数据。最重要的是 instance_size 字段，定义了类实例的尺寸。当你实例化一个类时，这个字段用于确保分配足够的空间。

在 Leopard 之前直接创建这些结构体是可能的，而 OS X 10.5 下可以用 objc_allocateClassPair() 函数构建，但是不那么方便。大多数时候你都可以用 Objective-C 语法来定义它们，而用编译器、运行时库来创建。

这由两个阶段完成。首先是描述对象的接口。通常是在头文件中，但如果不需要公布它们的接口，也可以放在实现文件中。代码清单 3.2 展示了一个简单类的接口。

代码清单 3.2：一个 Objective-C 接口（取自 examples/SimpleObject/simpleObject.m）

```
3 @interface SimpleObject : NSObject {
4     int counter;
5 }
6 - (void) addToCounter:(int) anInteger;
7 - (int) counter;
8 @end
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

接口括在 @interface 和 @end 关键字内。第一行给出类的名字（SimpleObject）和它的父类（NSObject）。因为 Objective-C 不支持多重继承，所以这是它唯一的父类。在 OS X 下指定 NSObject 为父类是多余的，因为不指定父类的时候就会使用它，但是这是一种好的编码风格。

下面的代码块包含了定义在类中的实例变量的列表。类的每个实例都有个结构体包含了类定义的所有实例变量，以及父类包含的全部实例变量，作为它的字段。值得注意的是类的指针（isa 指针）没有任何特殊的表现；它就是 NSObject 定义的一个简单的实例变量而已。

后面的 6、7 两行是方法声明。减号前缀说明它们是实例方法，而不是类方法，类方法使用加号前缀。实例方法是附加到类上的，而类方法是附加到类的元类上的。也就是说类的实例会响应实例方法，而类本身会响应类方法。

定义类的下一步是定义它的实现。代码清单 3.3 是对应的实现代码。在接口中声明的每个方法都必须在实现中声明，但是没有在接口中声明的方法也可以有。

代码清单 3.3：一个 Objective-C 类的实现（取自 examples/SimpleObject/simpleObject.m）

```
10 @implementation SimpleObject
11 - (void) addToCounter:(int) anInteger
12 {
13     counter += anInteger;
14 }
15 - (int) counter
16 {
17     return counter;
18 }
19 @end
```

Objective-C 没有运行时私有方法的概念；但是，如果试图调用一个没有在接口中声明过的方法，在编译期间会返回一个警告。

你可以看到 counter 没有初始化过。这是因为在 Objective-C 对象内的每个实例变量在对象创建时，都会自动被初始化为 0。如果这是合理的初值，那么你无须自己设置它。

协议

Java 的对象模型和 Objective-C 的很接近，而 Java 的接口（interface）概念非常像 Objective-C 的协议（protocol）。有一点不同的是，Java 的接口不是对象，而 Objective-C 的协议是，它是 Protocol 类的实例。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

协议就是一组消息，遵循协议的类必须实现这些消息。代码清单 3.4 是代码清单 3.2 的改版，区别在于现在方法声明在协议里，被类继承下来。类要遵循的协议写在父类后面，放在尖括号内，多个协议间用逗号分隔。

代码清单 3.4：使用协议的 Objective-C 接口（取自 examples/SimpleObject/simpleObject2.m）

```
3  @protocol SimpleObject
4  - (void) addToCounter:(int) anInteger;
5  - (int) counter;
6  @end
7
8  @interface SimpleObject : NSObject <SimpleObject> {
9      int counter;
10 }
11 @end
```

运行时你可以发送 -conformsToProtocol: 消息来检测对象是否遵循某个协议，例如：

```
[obj conformsToProtocol:@protocol(SimpleObject)];
```

这里系统的实现有点小问题。此时编译器无须了解 SimpleObject 协议的定义。协议可能还没链接到当前的编译单元。如果编译到这里时协议定义出现，系统就会创建一个 Protocol 对象，把名字设为“SimpleObject”。

如果两个协议名字相同，在运行时就会认为它们是同一个协议。它们的方法不会被检测。所以，注意在同一个程序里，一定不要出现两个名字一样，定义不同的协议。

category

Objective-C 最有意思的一个特性就是 category。一个 category 就是一组方法。像类一样，它可以包含接口和实现。与类不同的是，category 只是一组方法，不能包含实例变量，而且它是附加到一个现有类上去的。

category 提供了一种机制，让你可以扩展一个你没有源代码的类。它们也可以用来帮你从概念上把一个对象分成几个不同的部分。Objective-C 类的所有方法必须放在一个 @implementation...@end 块内，实现分散于不同文件的话，必须用预处理器整合到一起。

category 提供了一个替代方案。核心方法可以实现在类中，其他的方法可以在 category 中定义。代码清单 3.5 是一个简单的例子。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

代码清单 3.5：一个 Objective-C 类分成两个部分（取自 examples/SimpleObject/simpleObject3.m）

```
3  @interface SimpleObject : NSObject {
4      int counter;
5  }
6  - (void) addToCounter:(int) anInteger;
7  @end
8  @interface SimpleObject (GetMethod)
9  - (int) counter;
10 @end
11
12 @implementation SimpleObject
13 - (void) addToCounter:(int) anInteger
14 {
15     counter += anInteger;
16 }
17 @end
18 @implementation SimpleObject (GetMethod)
19 - (int) counter
20 {
21     return counter;
22 }
23 @end
```

这个实现跟前面两个 SimpleObject 没有本质区别。在这个版本里，-counter 的声明（8~10 行）和定义（18~23 行）都在 category 里。category 的名字，在声明和定义中都有，主要是为了归档。接口和实现对 category 来说都是可选的。

category 里没有办法实现对象的“私有”方法。

隐含协议

category 有时候用来实现隐含协议（informal protocol），方法是在 NSObject 上加入那些方法的空实现。这样就可以对任何对象发送消息，而无须检测它们是否遵循协议。这种做法能免则免，因为它会降低所有对象方法查找的速度，而且当两个不同的库都想加入一个相同方法时会造成问题。

3.4.3 异常与同步

从 OS X 10.3 起，Apple 在 Objective-C 中引入了一些 Java 式的扩展。本着库可以做的东西就绝不在语言层面实现的原则，Objective-C 没有引入任何异常处理

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

的元语。

在 OpenStep 里的异常处理是由一组宏和 NSException 类实现的。宏支持如下的代码块：

```
NS_DURING
    // Code throwing an exception.
    NS_VALUERETURN(YES, BOOL);
NS_HANDLER
    return NO;
NS_ENDHANDLER
```

这些宏是用 goto 语句，以及 setjmp()/longjmp() 调用的复杂组合实现的。setjmp() 保存当前的寄存器集合，而 longjmp() 则用以恢复。

这种做法有两大缺点。首先，setjmp() 没有办法在中间堆栈帧调用清除代码。这就是说在异常抛出和被捕获之间的栈帧，必须了解异常抛出的可能性，在每次调用前确保自己在一个可以突然返回的状态上。第二个问题就是慢。

本来异常仅应该发生在异常自己的环境内。一个理想的异常机制应该仅在异常真的抛出时才有开销。而这个实现正好相反；抛出一个异常成本很低，但是每个 NS_DURING 宏里的 setjmp() 调用开销很大。

还有一些小问题。只有 NSException 实例可以抛出，在处理块中由 localException 来识别。而且，在异常块内不能使用 return 语句，也不能从异常处理块中去掉，这样经常在后续执行中带来问题。必须用 NS_VALUERETURN 和 NS_VOIDRETURN 来代替。

OS X 10.3 中，Apple 引入了三个异常处理的关键字：@try，@catch 和 @finally。行为和 Java 中的对应关键字完全一致。现在异常块看起来是这样的：

```
@try {
    // Something that might throw an exception
} @catch(NSException *e) {
    // Handle the exception
} @finally {
    // Clean up
}
```

老的异常处理宏都用新关键字重新实现了，如代码清单 3.6 所示。新增的 @finally() 块大大简化了清理代码的写法，可以指定多个 @catch() 块的能力，可以支持同时处理不同异常类型。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

代码清单 3.6：OS X 10.6 以后的异常宏（取自 /System/Library/Frameworks/Foundation.framework/Headers/NSException.h）

```
66
67 #define NS_DURING        @try {
68 #define NS_HANDLER       } @catch (NSException *localException) {
69 #define NS_ENDHANDLER       }
70 #define NS_VALUERETURN(v,t) return (v)
```

大的问题，@try 块的开销还是没有解决。它还是要编译为 setjmp()/longimp() 调用，以兼容老代码。OS X 10.5 时，Apple 终于有机会打破 ABI 的限制，而不用担心有人抱怨。在编译 64 位代码，或者面向 Objective-C 2 运行时库的代码时，使用的就是新的异常模型。

这基于安腾处理器的“零开销”异常处理系统。简单可执行文件里面的函数就是指令流。如果编译时打开调试支持，会有些 DWARF 数据描述这些函数的数据布局，令调试器可以用名字访问变量，找到源代码文件中的当前行。零开销异常系统扩展了这些调试信息，对 @try 块中每个调用，提供了地址和 @catch 语句相关的类型信息。

当抛出一个异常时，unwinding 库恢复栈，对每个帧调用一个个性化函数 (personality function)。有些是用于任何支持 unwinding 的语言的。也包括 C 语言，虽然 C 语言版本只是执行下一帧而已。个性化函数读取栈帧的调试信息，检查是否有清除代码需要运行，检查它是否知道如何处理这个异常。

你可能会觉得这是开销很大的操作。每个栈帧都会调用个性化函数，它还需要调用额外的函数解析 DWARF 数据。但它的优点在于，只有异常抛出的时候才会执行这些操作。没有异常抛出时，什么都不做。unwinding 信息是静态包含的，可执行文件的尺寸会变得略大，但是 @try 块完全不会影响执行速度。

在新模型下，如果你的代码抛出的异常多到会拖慢程序，那么代码多半本来就错了。

这个模型的另外一个好处是它使用了和 C++ 一样的 unwinding 代码。这两种语言的个性化函数和类型都不同，你不能在一个语言中捕捉另一个语言的异常。但是如果 C++ 代码中调用了些 Objective-C 代码，而这些 Objective-C 代码又调用了些 C++ 代码，那么 C++ 代码中的异常抛出和捕获会引发 Objective-C 栈帧中的清除代码。

这一机制的底层实现提供了对外来异常的支持，所以也许在未来的版本会实现一个 NSCXXException 类，或者类似的什么东西用来在 Objective-C 中捕获 C++ 异常。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

异常处理的新关键字之外，Apple 还增加了 @synchronized() 语句。用来支持资源互斥占有，和 Java 的对应关键字很类似。在现代 Java VM 里，大量的工作都得到了这个机制的优化，例如，可以彻底清除锁定，因为它可以保证不会有两个 CPU 同时锁定一个对象。这需要精确的垃圾回收，以及重写本地二进制文件的能力，Objective-C 编译器这两件事情都做不了。所以，这个关键字非常低效。

当你这样锁定一个对象时，它会为对象寻找一个 POSIX 递归排他锁（recursive mutex），在块的开头锁定，在块的结尾解锁。这个查找过程开销可以很大，而且递归排他锁是 POSIX 线程库提供的开销最大的锁定元语，创建它开销就已经很大了。

@synchronized 的主要好处是它与异常处理机制集成。如果一个异常在块中抛出，它会被捕获，锁解开，然后异常重新抛出。而临界区中是无法抛出异常的，虽然它是 Objective-C 中最高效的锁定机制。

3.4.4 自省

Objective-C 提供了大量的自省信息。Objective-C 中的每个对象都是类的实例，而且包含了一个指针指向表现类结构的结构体。这与 C++ 不同，C++ 里的对象就是一个结构体，对象对自己的结构一无所知。这个指针对 Objective-C 的工作方式很重要。因为它的存在，每个对象自己知道如何处理发送给它的消息。同时每个对象都知道自己的结构。

反射

人们经常用术语反射来代表自省。这会带来混乱。自省指的是对象可以查看自己，了解自己的结构和能力。反射包括这点，也包括对象可以修改自己结构的能力。内省是反射的子集。Objective-C 的运行时库的 API 提供了反射的能力。更多相关信息参见第 25 章。

每个类有两个主要属性：方法列表和实例变量的列表。这些实例变量描述了对象的结构。每个实例变量有三个属性：

- 实例变量的名字。
- 实例变量的类型。
- 实例变量在对象（二进制结构）中的偏移位置。给定一个对象指针，这个值可以用来获得实例变量的指针。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

在随 Leopard 发行的新运行时库以前，这些属性都在编译期间绑定。到了 Leopard，偏移值在装载的时候才会确定。这有两个不好的副作用。首先，造成实例变量的访问变慢，因为访问是间接的。其次，让 @defs() 语句失效。在其他的运行时库下，你可以定义如下的结构体：

```
struct MyObjectStruct
{
    @defs(MyObject);
};
```

你可以把 MyObject 实例对象的指针转换成一个 struct MyObjectStruct* 指针，用 C 语言访问里面的实例变量。现在不行了，因为类的布局不再是在编译期间确定的了。这一改变带来的巨大优点是给类加入一个新的实例变量，不会造成全部子类的重新编译了。

新运行时库的另外一个改变是把类结构体变成了一个不透明的类型。现在要获得实例变量的偏移位置需要两步：

```
Ivar ivar = class_getInstanceVariable([obj class], "aVariable");
ptrdiff_t offset = ivar_getOffset(ivar);
```

Ivar 是新运行时库带来的另外一种不透明类型，老的类型其实就是一个 struct 包含了三种属性的列表。这个类型使用 Objective-C 类型编码形式（Objective-C type encoding）。它是一个字符串，由表 3.1 中的字符构成。

数组、结构体和联合体也由它们构建。数组编码为方括号，里面是元素的个数，然后是一个元素的类型标识。数组 int[12] 表现为 [12i]。结构体用花括号和包含的类型标识符构成。编码 {Point=ff} 可能对应的结构体如下：

```
struct Point
{
    float x;
    float y;
};
```

注意结构体的名字保留了，但是字段的名字被省略了。Objective-C 类型编码被广泛使用。使用 @encode() 编译语句你可以确保在编译期间使用类型编码。例如，@encode(int) 在运行时等于常量字符串“i”。你可以在与使用 sizeof() 相同的场景使用 @encode()。

用在方法中的类型编码使用更复杂一点的形式。前面例子里的 addToCounter: 方法的类型编码结果可能是“v12@0:4i8”或者“v20@0:8i16”，这取决于在 32 位还是 64 位平台编译。第一个字符表示函数的返回值类型，这里是 void。后面的数字表示全部参数的尺寸。然后跟着的是三个参数，以及他们在参数块里面的位置。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

它们是 self、_cmd 和 anInteger。self 和 _cmd 是隐藏参数，每个 Objective-C 方法都包含它们，所以不论一个方法有没有显式声明的参数，它们都会出现在方法的编码中。

表 3.1 Objective-C 类型编码

Type Encoding	C Type
c	char
C	unsigned char
s	short
S	unsigned short
i	int
I	unsigned int
l	long
L	unsigned long
q	long long
Q	unsigned long long
f	float
d	double
B	bool (C++) or _Bool (C99)
v	void
*	char *
@	id
#	Class
:	SEL
^type	type*

运行时库提供了一些底层的内省函数，技术上讲也可以算做 Objective-C 的一部分，但是它们是用 C 语言写的，而 C 语言是 Objective-C 的子集，所以它们用起来并不方便。Objective-C 内置了两个类：Protocol 和 Object。Object 类实现了一些对内省的包装。在 Cocoa 中，我们一般使用 NSObject，它提供了类似的功能。特别是你可以查询一个对象实现了什么方法和协议，还可以查询它在类层次中的位置。

你可以传递 -isMemberOfClass: 消息给一个对象，询问它是否是某个类的实例。这个消息用 Class 对象作为参数，如果接收者确实是这个类的实例，则返回 YES。-isKindOfClass: 方法的限制更宽松一些，只要接收者是参数中 Class 对象的任一个子类的实例即会返回 YES。

你可以发送 -conformsToProtocol: 消息，来确认类是否遵循某个协议。最有用的就是检测对象是否能处理某消息的能力。在使用委托对象的时候非常有用。Cocoa 中的一个常见模式如下：

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
if ([delegate respondsToSelector: @selector(delegateMessage:)])
{
    [delegate delegateMessage: anArgument];
}
```

在实际工作时，你最好考虑缓存委托对象是否响应对应的选择器的信息，不要每次运行的时候都去重新检测，这里只是展示基础的思路。由于这种能力，所以很多委托对象和事件处理方法无须协议，它们可以实现也可以不实现任何传递给它们的消息，检查它们是否可以响应这些消息是调用者的责任。

3.4.5 C 中的 Objective-C

Objective-C 的没有魔法的策略带来的一个好处是，任何可以在 Objective-C 里面做的事情都可以在纯 C 语言中做。在 OS X 下发送一个消息给对象，其实调用的是 objc_msgSend() 函数。Objective-C 方法会编译成带有两个隐藏参数的 C 函数。下面两行生成的代码是相同的 ：

```
- (int) aMethod
...
int aFunction(id self, SEL _cmd)
...
```

方法会被加到声明它的类中，但是执行部分是一样的。你可以发送 -methodForSelector: 消息给一个对象，获得特定方法对应的函数。它会掉用相应的运行时库函数，在 Leopard 中为 class_getMethodImplementation()。

代码清单 3.7 展示了如何用 C 语言写一个简单的 Objective-C 程序。7、8 两行查询要用的类。10~12 行查询要用的选择器。编译成 Objective-C 时，这些都会被模块载入函数代替，在每个编译单元中缓存。

真正的代码从第 14 行开始，创建了一个 autorelease pool。第 16 行创建了一个 NSObject 对象的实例。这些都通过发送 +new 消息给类对象实现的。注意，在实现级别和抽象级别，传递消息给对象和类都是没有区别的。在第 21 行，同一函数用于发送消息给一个 NSString 对象的实例。

18 和 19 两行展示了调用方法的另外一种机制。首先查询实例方法指针（instance method pointer，IMP），一个指向实现该方法的函数的指针，然后调用这个函数。这样做在 OS X 下很慢，而 Objective-C 生成代码的时候，是在 objc_msgSend() 函数里使用一些优化过的汇编代码，然后内联它们。但是这样的好处是你可以缓存结果。这被叫做 IMP 缓存，你可以只查询一次，以后每次调用的时候速度就和 C 语言函数调用一样快了。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

代码清单 3.7：用 C 语言写的简单 Objective-C 程序（取自 examples/ObjCinC/objc.c）

```
1  #include <objc/Runtime.h>
2  #include <stdio.h>
3
4  int main(void)
5  {
6      // Look up the classes we need
7      Class NSObject= (Class)objc_getClass("NSObject");
8      Class NSAutoreleasePool = (Class)objc_getClass("NSAutoreleasePool");
9      // Cache some selectors.
10     SEL new = sel_getUid("new");
11     SEL description = sel_getUid("description");
12     SEL UTF8String = sel_getUid("UTF8String");
13     // Create the autorelease pool
14     objc_msgSend(NSAutoreleasePool, new);
15     // Create the NSObject instance in two steps.
16     id obj = (id)objc_msgSend(NSObject, new);
17     // id descString = [obj description];
18     IMP descMethod = class_getMethodImplementation(obj->isa, description);
19     id descString = descMethod(obj, description);
20     // char *desc = [descString UTF8String];
21     char *desc = (char*)objc_msgSend(descString, UTF8String);
22     printf("Created object: %s\n", desc);
23     return 0;
24 }
```

编译运行这个程序，我们将得到如下输出：

```
$ gcc objc.c -framework Foundation  && ./a.out
Created object: <NSObject: 0x10032a0>
```

写了这么一堆才这点结果，所以通常大家不会想这么做。但是这些方法在你想用 C 语言给 Objective-C 写些模块的时候用得上。

注意这些例子只适于 Leopard Objective-C 运行时库。在 GNU Objective-C 或者 NeXT/Apple 的早期版本运行时库上不可行。这些库提供了相同的功能，但是使用了不同的接口。

3.4.6 Objective-C 2.0

在 OS X 10.5 中，Apple 引入了 Objective-C 的新版本。这个版本引入了大量新特性。同时包含了新的 Objective-C 运行时库，经完全重写，与老版本二进制不兼容。自省功能的接口更清晰，便于 Apple 未来改变实现方式。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

垃圾回收

引入的最大新功能是垃圾回收（garbage collection）。Objective-C 1 使用 Foundation 框架的引用计数功能，完全手工进行内存管理。

Objective-C 2 中，当你把对象指针赋值给实例或全局变量时，将加入内存写屏蔽。它会调用运行时库的函数更新垃圾回收器的状态。

随垃圾回收一起带来的最大改变是加入了弱引用（weak reference）。弱引用会被垃圾回收器忽略。只被弱引用的对象是可以被释放的，而被强引用的对象则不会。默认情况下，所有对象指针都是强引用。弱引用的指针由前缀 __weak 确定，如下：

```
__weak id delegate;
```

被弱引用的对象如果被释放，那么它的弱引用会被设置为 Nil。（实际上它们将在下次访问的时候被设置为 nil。）也就是说，它们可以用于任何指针，如果你不想指针指向无效内存，而且不在意对象是否被释放。这常用于委托对象以及集合中。

有了垃圾回收，对象的清除代码也略有改变。大多数对象实现 -dealloc 方法，用于释放对象。大多数时候，这个方法什么都不做，只是减少要被释放的对象引用的那些对象的引用数量而已。这对垃圾回收程序没有任何用处。取而代之的是，可以实现一个 -finalize 方法，在对象将要被垃圾回收器释放时调用。它可以用来关闭文件句柄，释放 C/C++ 数据等。

说起 Cocoa 里的垃圾回收，你有三个选择。第一个选项，很简单，不用就是了。这对兼容性来说是最好的选择，因为有垃圾回收的代码无法运行在 OS X 10.4 或者更早版本上，而且现在很难移植到其他 Cocoa API 实现上去。第二个选项是打开，到处都用。如果你刚开始 Cocoa 编程或者不在乎兼容性，这是最简单的选择。

还有条中间道路。即使你已经在老风格的引用计数机制下写好了代码，你仍旧可以从垃圾回收中得到好处。使用 GCC 的选项 -fobjc-gc，或者在 Xcode 工程属性中选择“Supported”垃圾回收，就可以在不是为垃圾回收所写的代码上打开垃圾回收功能。这将使所有引用计数消息被忽略，但是仍旧可以使用它们。这个选项对框架尤其有用，这样不管程序是否使用垃圾回收，框架都可以被链接进去。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

跟踪与引用计数

有两种方法可以实现垃圾回收，它们由相同的基础算法构成。一种是自动引用计数。仅使用它是不够的，因为遇到两个对象互相引用的情况永远不会被释放。要把引用技术变成完善的垃圾回收，你需要加入循环检测器。它会访问一些对象，然后移除因循环形成的额外引用。

另外一种是跟踪（tracing），Apple 在 Objective-C 2 中提供的 autozone 回收器就是这种。两种方法各有千秋。引用计数在每次赋值时代价更高点，而且消耗更多的内存。跟踪的运行次数更难确定，使得代码推理更加困难，而且在产生交换的低内存环境下表现不佳（由于跟踪代码需要依次访问每个对象，甚至是已被换出的对象），与不同地址空间的对象，如分布对象，也不太协调。

说不好为什么 Apple 要用跟踪，而不是加入一个自动的引用计数器以及循环检测器到已有的手动引用计数机制里去。

属性

Leopard 中 Objective-C 的另外一个新特性是属性（property）。也就是说，可以用一种统一的方式访问实例变量。它实际上是很薄的一层，用于声明和调用 setter 和 getter 方法。属性首先在类接口中声明，如下：

```
@property int counter;
```

这声明了一个叫做 counter 的属性，它的类型是整型（integer）。等同于声明如下两个方法：

```
- (int) counter;
- (void) setCounter: (int)anInteger;
```

你可以不用任何特殊语法来访问属性，发送相应的消息即可。也可以选择新语法，让它们看起来更像一个结构体的成员。下面两对代码显示了相同操作的两种不同语法形式：

```
// Using property syntax
obj.counter = 12;
int c = obj.counter;
// Using ObjC 1 syntax
[obj setCounter: 12];
int c = [obj counter];
```

新语法打破了一些 Objective-C 的既有规则，重用了已有的访问结构体成员的语法，但是行为不同。但是，当访问很深层次嵌套的属性时非常方便。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

属性的一大好处是当需要隐藏实现细节的时候方便多了。@synthesize 语句用来创建属性的实现。可以为已有的实例变量创建存取方法，或者加入一个新的实例变量。（后一种只能用于 64 位系统或 Objective-C 2 运行时库下。）你也可以使用 @dynamic 语句说明你将自己提供属性的存取代码。

属性提供了很多关于其实现的线索。在它们的声明中用一个逗号分隔的列表说明，如下：

```
@property(nonatomic, readonly) id object;
```

第一个往往最重要。默认情况下，属性使用 atomic 操作符修饰，使其线程安全。如果不需要线程安全，用 nonatomic 修饰可以使其更快。

下一个，readonly，表明只有读取方法会被生成。在这个例子中不会有方法 -setObject:。如果你不想用存取方法的默认名，还可以在属性声明中，使用 setter= 和 getter= 来指定想要的方法名。最好要确保属性遵循 key-value coding 规范的要求，例如布尔值的访问方法由“is”打头。

你还可以提部分 assign、retain、copy 等修饰放在属性声明里，来控制 @synthesize 生成的设置方法的语义。第一个通常是默认值，生成一个方法进行简单的赋值。这是非对象类型唯一可用选项。对于对象，如果不用垃圾回收，则没有默认值，必须明确指明一个。你很少会希望使用 assign，因为它很容易造成当你还指向一个对象时，它却被释放了，从而造成你间接地引用了一个无效指针。retain 和 copy 选项都会导致老的值被释放（引用计数减少），而新的值得到一个 retain 或者 copy 消息。前者增加引用计数，后者返回一个 copy（对不可变对象则返回原始对象）。

3.4.7 Block

严格说来 block 不是 Objective-C 的一部分，它是 Apple 设计的 C 语言的扩展，由 OS X 10.6 引入。很早以前，GCC 就支持嵌套函数。允许你定义函数里的函数，可以访问外圈函数作用域内的变量，它使你可以写如下的代码：

```
void function(void)
{
    int count=0
    id nested(id object)
    {
        count++;
        return [object permute];
    }
    array = mapArray(array, nested);
    return count;
}
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

做过函数式编程或者动态语言编程的人可能会觉得很熟悉。里面的函数的指针传递给 mapArray() 函数，它（假定）会针对集合里面的每个对象调用它，然后返回生成的数组。

Apple 在 OS X 中的版本里默认禁用了这种功能。原因是它们的实现上可能有安全隐患。代码编译时，内外函数先会被编译成两个分离的函数。当你取内部函数的指针时，实际上得到的是栈上一个跳板的地址。这是一段很小的机器代码，载入跳板代码后紧跟的栈帧的指针，然后跳到真实的函数上。

因为需要在栈上运行代码，所以需要栈既可写又可以执行。从安全角度来看这是一个坏主意，这样很容易让栈溢出，然后执行任何恶意代码。

不幸的是，嵌套函数非常有用，很多人都抱怨 Apple 禁用了它们。Apple 的解决方案是创建了一个新的语言特性：block。引入了一个新的 C 语言调用约定和指针类型。函数指针用 ^ 声明，而不是用普通指针的 *，这是一个胖指针，包含一个函数指针和一个数据指针。数据指针是传递给函数的隐含参数。这里用了和一些 C++ 编译器里方法指针一样的机制。使用 block，你可以写如下的代码：

Block 和 Block

block 这个术语很容易混淆。在 C 语言里，通常把一个作用域范围——花括号间的区域叫做 block。新的语言特性得名自 Smalltalk 里的类似特性。在 Smalltalk 里，block 兼有二者的角色。作用域在 Smalltalk 里实际上是闭包，跟这里的 block 一样。

这段代码将在屏幕上打印 12。call_a_block 的参数是一个 block，新型函数包含了一个隐含的参数。参数指针指向了包含 X 副本的结构体。这个 block 从创建起就可以访问 X 的值，但是不能给它赋值。

如同新的指针类型和函数声明方法一样，还加入了一个新变量声明修饰符，如下：

```
__block int shared = 12;
void (^nested)(int) =
    ^(int y)
    {
        shared += y;
    };
```

因为变量 shared 声明为 __block，它将被复制到一个引用计数的结构里，被跟 nested() 相关联的数据引用。每次调用 nested() 将增加它的值，栈帧也可以引用

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

它。从程序员的视角来看，变量 shared 在 block 里外会保持一致，都可以被修改。

这些 block 是一级闭包。它们不能和函数指针互换，但是可以被用来做同样的事情。它们使你可以快速生成引用某个作用域变量的函数，也可以用于实现对集合的映射、归并等操作。

block 实现了类似 Objective-C 的 retain/release 机制。如果你想保留一个 block 的指针，应该这样做：

```
savedBlock = Block_copy(aBlock);
// 一段时间以后：
Block_release(savedBlock);
```

这将复制 block 或者增加它的引用计数，具体取决于 block 是否还在栈上。

未来 Cocoa 肯定会在集合类及其他方面大量使用 block。OS X 10.6 中加入了大量的这类方法，我们将在下一章对其中的一部分进行介绍。block 补充了 Objective-C 的模型，当一个对象上仅定义一个方法时，使用 blocks 写一个全须全影的对象高效多了。你可以用 block 字典来实现完整的对象模型，当然这就比真的对象慢多了。

如同 Smalltalk，Objective-C 的 block 也是对象。不用调两个函数来实现引用计数，发送 -retain 和 -release 消息即可。你可以在任何可以使用对象的场景使用 block。这样流程控制就变得非常灵活。例如，可以在一个字典里面保存很多 block，然后根据得到的输入来调用相应的 block。

把 block 加入对象中作为方法也是可以的。这使 Objective-C 得以支持 Lieberman 原型，就像 JavaScript 和 Self 语言那样。这是快速定制类的一种非常方便的方法，尤其是针对用户界面类的工作。甚至不需要 Apple 的直接支持，可以使用第二次机会转发机制来调用 block，虽然这样会非常慢，也可以用一个跳板函数来做。

3.4.8 Objective-C++

Objective-C 是对 C 语言的一组扩展。这些扩展大都与 C 语言正交。还有一组对 C++ 的扩展，它们叫做 Objective-C++。并非每个 Objective-C 程序都是合法的 Objective-C++ 程序，就像并非每个 C 语言程序都是合法的 C++ 程序一样；然而每个 C++ 程序都是合法的 Objective-C++ 程序。

Objective-C++ 几乎没什么用处，因为 C++ 比起 Objective-C 也没什么好处。主要用处是在需要接入一个非常复杂的已有 C++ 代码时。最知名的例子就是

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

WebKit 框架。最早它是 KDE 项目的 KHTML 库的一个分支，使用 C++ 编写。所有在 OS X 上的公开接口都是用 Objective-C 写的，而 Objective-C++ 用于两种语言之间的桥接。

如果你在把 C++ 程序移植到 OS X 上，那么 Objective-C++ 可以让你使用 Cocoa 但无须在 C++ 和 Objective-C 代码之间写 C 接口。

3.5 Cocoa规约

约定对 API 的一致性很重要。学会 Cocoa 所使用的模式，比学会每个类每个函数的细节实现方式容易得多。Cocoa API 非常巨大，对大多数程序员来说每件事情都记住是不可能的。最重要的事情是能够轻松找到正确的类和方法，可以使用 Xcode 内置的自动完成，或者查询文档。

3.5.1 命名

一个好的语言是友好 API 的良好开端，但是命名约定也是非常重要的。对 Java 1.0 的最大抱怨就是函数间参数顺序的不统一。约定的统一使人们容易记住 API 如何使用，因为记住常用模式容易，记住每个细节难。

常常会激怒初学者的一点是，Cocoa 的命名怎么那么冗长。类和方法名常常都是十几个字符，某些方法名还会更长。但是如果用支持自动完成的代码编辑器这就不会成为一个问题。在 Xcode 下你一般只需要键入一个名字的前几个字符，编辑器就会帮你完成后面的。

前缀

Objective-C 中每个框架内的类都带有一个短前缀。所有的 Cocoa 标准类都有 NS 前缀，这是 NeXTStep 的遗迹。这是 Objective-C 缺少名称空间概念造成的。所有链接到程序里的 Objective-C 类都在一个名称空间内。这意味着实现了同名类的两个框架是不能链接到一起的。

解决方案是框架内的类的类名都包含一个标明框架提供者的短前缀。Apple 大量使用 NS，但是某些框架有自己的前缀，比如 AB 是 AddressBook 框架的前缀。

这些前缀不仅用于类名。函数、常量，以及任何在全局名称空间下的东西都应该使用它们。方法除外，因为方法总是附加在指定的类上的。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

类名

简单类的类名通常是一个名词加上前缀构成的。例如 NSObject 或者 ABPerson。子类经常是在前缀和名词之间加上形容词，如 NSMutableArray 是 NSArray 的子类，在 NSArray 中加入了可以修改的行为。

类名应该展现类的功能。大多数类封装某种数据，它们的名字应该反映这点。例如，NSDate 或者 PDFPage 表现的是什么数据是非常明显的。

协议的命名略有区别。协议描述的是一种行为，而类是描述一种对象。所以，协议通常用动词命名，如 NSCopying 或者 NSLocking。例外是当协议描述类的重要方法时。例如，NSObject 协议，它描述的是 NSObject 类要实现的方法，以及所有在 Cocoa 中其他基类要实现的方法。这个协议被 Cocoa 中的另一基类 NSProxy 所采用。

如果定义了某种通用想法的默认实现类，这个模式也适用。在其他的代码，你应该要求采用这个协议的对象，而不是这个类的实例，允许其他人创建自己的版本，而不需要继承或者显式地类型转化。

方法名

方法名应该由小写字母打头。通常由一个动词开始，每个参数一个名词。没有任何参数的方法，如果要执行某种操作，如 -retain 或者 -log，那么用动词即可。如果它们有一个参数，应该用动词 - 名词对，如 addObject:。

对于有多个参数的方法，规则略微复杂一点。每个参数应该是一个名词，或形容词 - 名词对。例如，setObject:forKey:，动词 - 名词对用于第一个参数，形容词 - 名词对用于第二个。形容词也常被忽略，如 -initWithString:calendarFormat:，因为有时候显得多余。反之，方法 -rangeOfUnit:inUnit:forDate: 的每个参数都有形容词，甚至第一个。因为这是在查询对象中的数据，而不是执行任何动作。调用 -rangeOfUnit:unit:date: 方法的话，无法起到自文档的作用。

大多数返回对象部分成员的方法的名字是一个单词，如 -count 或者 -length。例外的是返回 BOOL 类型的方法，通常由 is 打头，如 -isEnabled。这样方法名更易懂。你可以朗读下面的代码，读起来就像英语的句子一样：

```
if ([anObject isEnabled])
```

方法名通常都很长，你可能需要把它们折成多行。Objective-C 的约定是把冒号排成一列。在 Étoilé 里，我们采用了这个约定，缩进后面的空白应该用空格，而缩进本身应该用 tab。这就是下面的布局：

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
    [NSAlert alertWithMessageText: @"Example"
                    defaultButton: @"Continue"
                  alternateButton: @"Abort"
                      otherButton: nil
        informativeTextWithFormat: @"An example long line"];
```

这个方法的好处是，允许任何人读你的代码，不管他们的缩进设置是如何的，也不会打乱参数。不幸的是，Xcode 不区分缩进和对齐，所以不会自动缩进这种风格的代码。

需要注意的是在 Apple 实现的 Objective-C 中 selector 是没有类型的。假设你在两个不同的类，定义了两个名字相同的方法，但是参数类型不同。当其他代码有一个指向某对象的指针，发送这个名字的消息给这个对象，编译器会生成调用的代码。如果指针指向特定对象类型，那么这个对象版本（假设编译器知道）的方法声明将会被选择。否则，可能选择任何一个。

名字中的“Get”

很多 toolkit 都使用 get 作为返回属性的方法前缀，而使用 set 作为设置属性的方法前缀。Cocoa 使用 set 前缀，但不用 get。只有在方法或者函数接受指针作为参数，并用它通过引用的方式返回值时，才会在名字里包含 get。

如果对象的类型不同，那么也没问题。参数响应当前消息，或者抛出一个运行时异常。但是，如果它们有不同的原始类型，或者其中一个是原始类型，一个是对象类型，那么栈帧会结束在错误的尺寸上。这会导致栈失效、安全漏洞，以及崩溃。

为了避免这个问题，很多方法在它们的名字上带上了参数的类型，例如 -addObject: 或者 -setIntValue:。这是一个好习惯。如果你的对象放在框架中，和其他框架链接在一起，它们实现了冲突的选择器名，那么这个问题会一直延续到框架重写才能得以解决。

函数名

函数名遵循类名和方法名约定的混合体。如果它们不是某个公开 API 的组成部分，那么应该声明为 static。这样可以阻止它们进入全局名称空间，于是它们的名字就没那么重要了。对于那些不能声明为 static 的函数，总是应该由一个前缀打头。Cocoa 函数和 Cocoa 类一样，都由 NS 打头。

通常，Cocoa 有两种函数：一些做点什么，一些执行类型转换。具体来

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

说，有些被调用是因为它们的作用，有些是因为其返回值。后面一种通常命名为 NSTypeFromOtherType()。它们返回参数的其他类型展示形式，或者是参数的属性。例如，NSClassFromString() 和 NSEventMaskFromType()。

因为作用而被调用的函数的名字通常是前缀加上动词，如 NSAllocateObject()。

指向不透明类型的函数通常函数名都包含类型的名字，往往紧跟前缀。例如 NSMapInsert() 和 NSDecimalAdd()。有些在前面加上了动词，如 NSFreeMapTable() 和 NSMakePoint()。不幸的是，在 Foundation 库中函数的命名就有很多的不一致。

3.5.2 内存管理

传统（NeXT 之前的）Objective-C 使用手工内存管理，和 C/C++ 类似。你可以在 Objective-C 运行时库中的 Object 类里看到，接口定义在 /usr/include/objc/Object.h，现在很少用。这个类提供了 +new 和 -free 方法，直接对应 C 语言的 malloc() 和 free() 函数。

问题在于，对象经常被别名引用，很多对象都指向它们，非常难以判断何时释放一个对象。通常的解决方案是，给每个对象分配一个宿主，制定复杂的规则来决定哪个对象拥有其他的某个对象，然后在你无法确定的时候，尽量多地复制对象。

引用计数

Cocoa 和 Objective-C 1 使用引用计数。如果想保持一个对象的引用，就发一个 -retain。绝大多数情况，这将返回这个对象。但偶尔也会返回一个新对象，所以永远要像下面这样返回对象：

```
obj = [obj retain];
```

这可能发生在对象保存在多变的环境里，很快会失效时。-retain 的对立面是 -release。如果不再使用对象，应该发送 -release 消息给它。

有一件麻烦的事情要记住。在实现实例变量的设置方法时，你可能想显然应该像下面这么做：

```
- (void) setFoo:(id) anObject
{
    [foo release];
    foo = [anObject retain];
}
```

这在大多数情况下没问题，但是当 anObject == foo 而且引用计数为 1 时会怎

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

样呢？在这种情况下，release 消息会导致对象被释放；retain 消息会被发送给非法的内存地址。如果对象的内存还没有归还给操作系统，那么 foo 的新值将和老值一样，只是指针失效了，会造成后面出现微妙的 bug。如果已经归还，则会马上造成段失效错误。解决方案是先 -retain，如下：

```
id tmp = [anObject retain];
[foo release];
foo = tmp;
```

有点麻烦。GNUstep 提供了 ASSIGN() 宏来做这个操作。你可以考虑在自己的代码里实现类似的逻辑。可以像下面这样：

```
#define ASSIGN(var, obj) do {\
    id _tmp = [obj retain];\
    [var release];\
    var = _tmp;\
} while(0)
```

Autorelease Pool

简单的引用计数在大多数情况下足够了，但是也有问题。最常见的是，函数或者方法返回一个临时对象时，该怎么办？你不希望保持它的引用，所以你想释放它；但是这样一来调用者还没有机会 retain 它，它就被你摧毁了。

Cocoa 给出的解决方案是 autorelease pool。Cocoa 程序中的任何一点，都必须有 NSAutoreleasePool 的合法实例。如果你发送 -autorelease 消息给对象，而不是 -release 消息，那么它会查询当前的 autorelease pool，把自己加进去。当前 autorelease pool 销毁时，它会发送 -release 消息给每个注册在其上的对象。

autorelease 一个对象意味着你现在不需要它了，但是别人也许会需要，所以先别摧毁它。通常在函数或者方法里创建一个对象后，马上就要 autorelease 它，而不需要等到返回的时候。

如果使用垃圾回收环境，那么你不需要考虑 retain 和 release 对象。本书会用引用计数风格编写例子，因为这样的代码可以在垃圾回收的环境下编译，而反之则不可。如果你希望支持 OS X 10.5 之前的版本、iPhone、使用尚未编译成垃圾回收版本的框架的用户，以及其他的 OpenStep 实现，那么你的代码就不要依赖于垃圾回收。

GNUstep 很多年前就通过 Boehm 垃圾回收器支持了垃圾回收，使用 RETAIN() 和 RELEASE() 宏，和垃圾回收一起编译的时候预处理器会自动移除这些代码。你可以学着使用这种不错的方式。未来继续支持没有垃圾回收的系统是没有意义的，而这提供了一个简单的方式来移除所有多余的 retain 和 release 消息。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

3.5.3 构造器与初始化器

传统的 Objective-C（NeXT 以前）的对象使用 Smalltalk 风格的 +new 消息创建，而 +new 消息就是 malloc() 的简单封装。为了支持更复杂的内存管理系统，NeXT 把它分成两个方法。当你发送 +new 消息给一个类，它会调用 +alloc 方法，然后对结果调用 -init 方法。

+alloc 方法声明在 NSObject 之中，它调用 +allocWithZone:，把对象的实例分配在一个 NSZone 内。在 Cocoa 里，这是一个用来描述一个内存区域的不透明类型。目前只支持两种内存分配方式：堆式和栈式。后者必须一次性全部释放，在创建大量短生命周期的对象时很有用。

NeXTSTEP 中大量使用 NSZone 去充分攫取当时慢速机器的性能。而在 Cocoa 中很少用。不幸的是，NSZone 的好处只有在全面使用它的时候才能显现，而这很难再出现了。同时，在使用垃圾回收的时候，NSZone 是没用的。

主要的遗产是你几乎永远无法绕过 +alloc，但是可以轻松地绕过 -init。比你想象的要难点。简单地调用父类的 -init 方法永远不够用。我们不需要对象通过 -init 方法返回它本身，而是需要一个单件（singleton）或者其他值，或者初始化会失败，返回 nil。代码清单 3.8 展现了实现一个 -init 方法正确的方式。

代码清单 3.8：对象初始化方法

```
1  - (id) init
2  {
3      self = [super init];
4      if (self == nil)
5      {
6          return nil;
7      }
8      // Do initialization specific to this class.
9      if (someInitializationFailureCondition)
10     {
11         [self release]
12         self = nil;
13     }
14     return self;
15 }
```

第一步是发送一个 -init 消息到父类，然后把结果赋值给 self。这就实现了父类要做的初始化工作。给 self 赋值看起来很奇怪，但是记住 self 只是 Objective-C 里的一个参数而已，遵循其他函数参数要遵循的规则。

如果给 self 赋值为 nil，那么任何设置实例变量的企图都会导致段失效错误，

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

程序会崩溃。然后你应该做初始化。大多数类没有什么可失败的初始化操作。但在某些情况下，初始化需要一些资源，如文件或者网络连接，则可能会失败。这种情况下，对象会释放自己，初始化函数会返回 nil。

-init 以外，类还可以实现 +initialize 方法。这个方法会在运行时，当第一个消息发给类的时候被自动调用。这是实现文件内静态变量的延迟初始化的好方法，可以改善 Objective-C 程序的启动时间，因为它减少了载入时需要执行的操作。+initialize 会在类的任何实例创建前调用。一般来说，发送给类的消息会是 +alloc，所以类的初始化可以在第一个实例创建前开始。

注意，子类可以调用父类的 +initialize 方法。有时候你可能想对每个子类执行这个操作，但是更多时候将只想执行一次。代码清单 3.9 展示了该如何做。

代码清单 3.9：类初始化方法

```
1 + (void) initialize
2 {
3     if (self == [MyClass class])
4     {
5         // Do initialization
6     }
7     [super initialize];
8 }
```

因为这是一个类方法，self 将是类本身，而不是类的一个实例。第 3 行的判断只会在执行 MyClass 的 +initialize 时成功一次，而在任何子类的 +initialize 都不会成功。最后一行调用父类的 +initialize，而父类的 +initialize 内也可能有类似的代码块来忽略这一消息。

注意这个方法与对象初始化方法的不同，它返回 void。这是因为它被运行时库调用，而不是用户的代码。就算它返回什么值，对用户也没什么意义。

Objective-C 运行时库还为 +load 方法提供了特殊的行为。它会在类载入的时候被调用。因为库载入的顺序是没有严格定义的，也就是说它依赖于此时父类已经被载入，而不是其他的类。实际应用中，通常可以认为 Cocoa 类比你自己的载入更早，但是这是不好的习惯。

实现 +load 方法一般用来进行类替换（class posing），就是在运行时用一个类替换另外一个。这是 category 的替代技术，方法的新实现可以调用老的实现。新的运行时库废弃了这种方法，虽然用替换方法实现的方式仍旧可以实现相同的效果。这需要用 +load 代替 +initialize，因为这必须发生在子类实例化之前。

因为在载入期间执行，+load 会延长程序的启动时间，所以从用户体验角度

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

看不是个好主意。因为执行的状态没有良好的定义，对很多初始化任务这也不是一个好选项。笔者总共写过两次 +load 方法，都是很底层的代码，改变了大量的基础功能。

除了默认的初始化方法，有很多对象还实现了自己的初始化和构造方法。例如 NSArray 的 +arrayWithObject: 和 -initWithObject: 方法。第一个在分配对象内存后调用第二个。

每个类都有指定初始化方法（designed initializer），在子类初始化时必须调用父类的这个方法。通常是 -init，但不总是它。有些类族（class cluster）没有指定的初始化方法；子类在初始化期间不应给父类发任何消息。

调用命名的构造方法，返回的对象会被 autorelease。下面两行是等同的：

```
[NSArray arrayWithObject: anObject];
[[[NSArray alloc] initWithObject: anObject] autorelease];
```

前面一种通常是后面一种的包装，但是某些类族的初始化方法里，会避免生成子类实例会替换掉的占位对象。某些时候，只有其中一种可以用。

3.6 小结

本章介绍了 Objective-C 和 Apple 提供的开发工具。想熟悉它们只有多练习。Xcode 遵循很多与其他 Mac 程序一样的约定，所以相对易学。具体来说，它大量使用检视器（inspector），大多数你可以点的东西都可以被检视。花点时间浏览各种检视器，看看开发工具带的例子，以及 Interface Builder 面板里面的各种对象。

如果你已经懂 C 语言，你可以试着不用多少 Cocoa 库写点简单的 Objective-C 程序。如果你用简单的 C 库实现过自己的对象模型，试着用 NSObject 代替它。这不需要太多 Cocoa 知识，只是用 Objective-C @interface 替换你的每个类型定义而已，使用 -retain 和 -release 方法做内存管理，把函数改成方法。但是不要花太多时间在这里。在本书后面的章节，我们将会看到 Cocoa 提供的标准对象，它们让 Objective-C 变得更加简单。