第1章

欢迎使用 Swift

苹果公司在 2014 年推出 Swift 编程语言，该语言是一门性能可调整且类型安全的现代编程语言，拟在取代 Objective-C 语言。这门新语言包括面向协议编程、类型推断、自动引用计数、泛型、优秀的函数对象、重载、可为空特性、可选类型等。Swift 具有详细检查控制的机制和结构，增强了某些特性，比如 Cocoa 和 Cocoa Touch 项目中的 Switch 和枚举类型，使其更为强大但又不失灵活性。

首席开发者 Chris Lattner 在 2010 年开始研发 Swift 语言。该项目一直在进行，并在 2013 年成为 Apple 工具开发组的主要焦点。2014 年推出后，Chris Lattner 在他的博客中详述了设计 Swift 语言所付出的努力：“Swift 语言的诞生是大家不懈努力的结果，其中包括语言专家、参考资料的专家、编译程序的优化者，以及内部的 dogfooding 组，他们所提供的反馈建议帮助更好地测试和优化产品。当然，也从该领域的其他语言中吸取了许多宝贵的经验，其中包括 Objective-C、Rust、Haskell、Ruby、Python、C#、CLU 等，远远要比上面列举的多得多。”

在 WWDC 2015 之前，Swift 语言都没有发挥其真正的潜力。在那时，Swift 发展到了 2.0 版本，重新设计了错误处理系统，更好地与原有 API 进行整合，并且把功能进行了升级，让语言与功能更为接近。这并不是说 Swift 时代已经来临。Swift 2.0 虽然已经推出，但 Swift 语言本身并没有完成。在 WWDC 演讲表达的主题词汇中会反复出现诸如“在未来更新”、“在未来的版本中”和“当我们有时间会这样去实现”等字样。虽然 WWDC 上 Swift 发生了巨大变化，但是几乎不影响支撑你编写每一行代码的基本概念。

自从苹果公司推出 Swift 语言之后，它的更新就一直没有间断。从一个 beta 版到另一个 beta 版，从一个 release 版再到另一个 release 版，这说明在源代码中添加的新功能和新语法将有可能在下一个语言版本更新时就不适用了。笔者预测在未来的几年里，至少每六个月 Swift 就会更新一次，之后更新速度就开始减慢，直到它的稳定版本出现。

现在 Swift 的处境有些尴尬。它虽然不够精致，但是它的势头非常迅猛，同时对苹果开

发者社区也非常重要，以至于它不能被忽略。如果你还没有加入，那么是时候加入了。请接受这个现实。这是一个漫长、艰难且令人沮丧的工作环境。

我很乐意投身于一个关于 Swift 的图书项目，原因有两点。首先，Swift 最基本的特征是融合。虽然很多小功能不断更新，但是结构体、枚举和类的地位不太可能被设计师所撼动。其次，苹果致力于提供该语言的版本迁移工具，使你可以把源代码升级到最新的语言版本。当 Swift 将其基本的打印函数从 println()修改成 print()时，Xcode 的迁移工具对此改变提供了很大帮助。

开发者不再使用早期的 Swift 测试版，因为测试版语言中的一些东西已不再适用。苹果公司主要是更新 Swift 中的技术和程序，而不是核心语言概念，确切地说是不更新大部分核心语言概念。早期测试版中的编码有些滑稽可笑。现如今 Swift 虽然有些令人沮丧，但它还是可以被大加赞赏的。Swift 语言的更新虽然还没有停止，但现在的版本是值得你花费时间和精力去学习和使用的。

苹果公司一直致力于打造 Swift 的未来。现在是跳上 Swift 这趟列车的最佳时机，去看一下它究竟能把你带向何方。不要看着 Swift 2.0 说“我已经太晚了，不能登上这列火车了”或者“Swift 2.0 版本怎么依然是个测试版？”请你使用 Swift 中基本的数学计算将最新版本转换为它的真实版本号：

```
Official Swift version N = Unofficial Swift(version: N * 0.1)
```

这种简单而有趣的方式说明“2.0 版本离真正发布的 1.0 版本还有很长的路要走”。如果 Swift 语言继续加快更新速度的话，那么这个数字常量 0.1 可能会上升。

1.1 代码迁移

在本书开头就讨论代码迁移似乎有些奇怪，但本章致力于讨论 Swift 的可爱性和高价值，这稍微会让你有点措手不及。了解如何在一个不断更新的语言中迁移代码是一项很实用的技能。

Xcode 中的 Swift 代码迁移功能在 Swift 1.2 版本中首次引入。这是不断更新的语言的一个重要功能，该功能可以帮助定位过时的语法，并提供了一条顺利更新到最新语言版本的途径。这个工具的确切细节可能会随着时间的推移而改变，但苹果公司已经承诺为每个 release 版本提供一个迁移工具，以支持开发人员。

1.1.1 如何迁移

在 Edit 菜单下找到 Convert 选项。这个菜单在 Xcode 7 的 beta 版中如图 1-1 所示。一旦选择了 Convert，Xcode 就会一步步将所有的代码迁移到最新的 Swift 版本中。

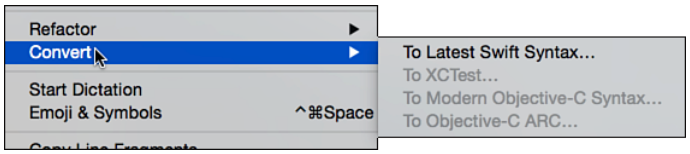

图 1-1　Xcode 可以将 Swift 源码更新为最新的语法

选择要升级的目标，如图 1-2 所示。你可能会被提示是否启用存储快照。无论你是使用 git(可以使用喜欢的版本控制系统)还是使用 Xcode 自带的存储仓库，都强烈建议你提交原有的项目代码，这样就可以把项目恢复到升级前的版本。

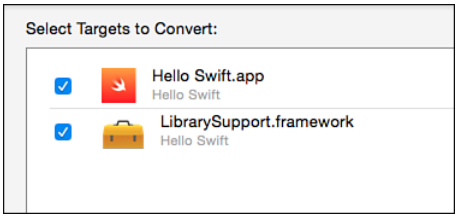

图 1-2　选择项目中你想转换的目标

Xcode 会扫描所有文件，并高亮标记出所修改的文件(见图 1-3)。可以通过单击 Save 或 Cancel 来允许或取消更新。

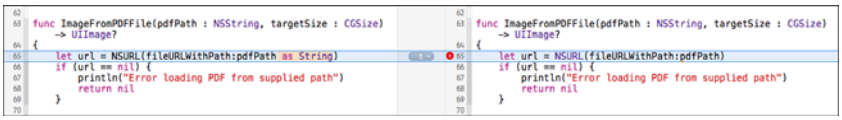

图 1-3　Xcode 提供的一边对一边的比较方式，展示了新语法(转换后)和旧语法(转换前)的不同之处。左边是更新后的代码，右边是更新前的代码

每个升级修改的地方会有一个红点，如图 1-3 所示。红点的旁边是一个带有数字编号的灰色小椭圆。单击每个红点可以查看更新的详细说明信息。单击灰色椭圆形按钮则会放弃更新。

左上方列表中的内容是 Review Changes 窗格。在 Review Changes 窗格中，可以选择单独的文件进行审查。在 Xcode 中，转换后(After Conversion)的版本位于左窗格，转换前(Before Conversion)的版本位于右窗格。如果你不确定哪边的窗格具体对应着什么类型的代码，可以查看每个窗格底部的标题。Xcode 显式地添加了转换角色以及文件名 。完成审查后，就可以单击 Save 进行保存。Xcode 如果有新的版本，就可以使用最新的编译器进行代码编辑。

1.1.2　迁移课程

Swift 2.0 beta 版发布后，开发人员开始升级苹果公司提供的 playgrounds 示例，例如在 2015 年 3 月份提供的 Mandelbrot 示例(https://developer.apple.com/swift/blog/?id=26)。一些开发人员试图让示例正常工作，从而浪费了几个小时。尽量不要手动升级代码，Xcode 的代码迁移工

具(migrator)能够帮助你更好地完成此项工作。它机械地执行代码升级，并且对迁移工作没有任何偏见。那些通过重新下载旧项目并选择自动升级的开发人员在几分钟内就可以运行该示例代码。

升级程序并不是万能的。迁移工具并不会发现逻辑错误或编码缺陷，它不能转变编程范式。例如，使用一个涉及 Swift 的相当复杂的概念，考虑一种 Swift 应用处理失败条件的方式。Swift 中的代码迁移不能把 Swift 1.2 中的 if-let 转变为 Swift 2.0 中的 guard 语句。你需要手动地对这些类型进行转换。虽然范式变了，但 Xcode 调整的只是语法。它不考虑最佳做法或模式更新。

Xcode 不能升级版本太旧的代码。最初的 1.1 到 1.2 版本都不能处理 Swift beta 1.0 的代码。你要保持代码是最新版本，要时刻为下一个语言版本的更新做好准备。这虽然需要花费大量的时间，但如果你要继续投身于 Swift 语言的话，这也是必要的。

迁移意味着代码复审。在图 1-3 中可以看到并排式的比较图。这就是在升级后必须要对代码进行检查的原因。这是经验之谈。这并没有减少语言版本迁移所需要的时间成本和所造成的失控风险。

重要的是，你可以从 Swift 版本迁移中学到一些东西。花时间检查代码是非常重要的，因为它是保持当前库和应用最新的重要组成部分。一个完整的套件提供完整的代码覆盖，并确保你的功能模块可以通过最小的验证集合。如果你的测试代码是使用 Swift 实现的，就需要花费一定的时间对测试进行迁移。然后自动化测试升级后的程序，正如 Xcode 自动迁移一样。

1.2 使用 Swift

Swift 是一门通用语言，它可以被当成脚本来编译或运行，也可以在命令行中使用，或者在独立的应用中使用。从编译的应用到脚本，这里简单总结了一些 Swift 生成可执行代码的方法。

1.2.1 编译应用

最典型的案例就是可以使用 Swift 为 iOS、OS X、tvOS 和 watchOS 构建应用和扩展程序(见图 1-4)。Swift 是一门现代的、类型安全的语言，它提供了快速高效的代码生成方式。它是一套完整的开发方案，整合了苹果全系列的开发人员 API。你可以使用纯 Swift 编码来创建应用, 或者混合使用 Swift、C 和 Objective-C 来创建应用。

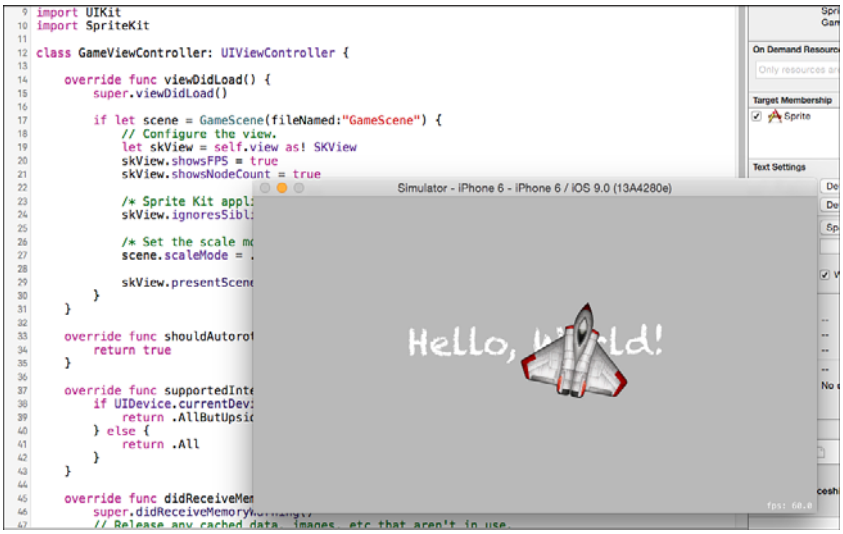

图 1-4　Swift 是 Apple 应用开发的首选语言

1.2.2　框架和库

Swift 并不仅仅限于构建带有传统用户界面的应用，它使你能够创建 frameworks、bundles、services 和命令行程序(Swift 目前还不支持创建静态库)。Xcode 提供了一个宽泛的且支持 Swift 语言的内置模板，所以可以从它开始构建一个大型项目(见图 1-5)。

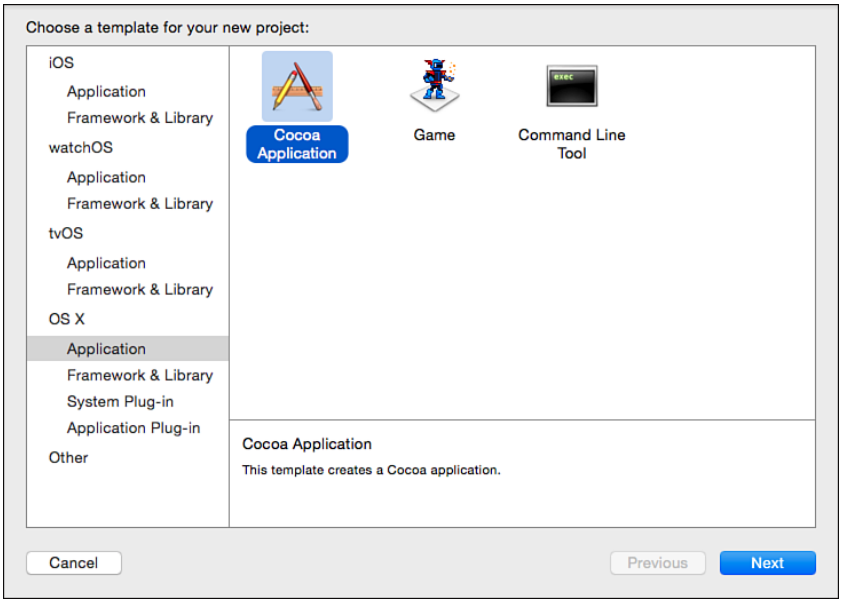

图 1-5　Swift 构建类似于命令行工具和框架的对象

1.2.3 脚本

Swift 还是一门脚本语言。可以使用它快速构建 shell 工具。下面的示例通过 iTunes 查找一个特定 App Store ID 项目的价格。除了第一行的内容外(哈希签名后是感叹号)，这个脚本与用于命令行工具编译的源代码没有区别：

```
#!/usr/bin/xcrun swift
import Cocoa
var arguments = Process.arguments
for appID in arguments.dropFirst() {
    let urlString = "https://itunes.apple.com/lookup?id=\(appID)"
    guard let url = NSURL(string: urlString) else {continue}
    guard let data = NSData(contentsOfURL: url) else {continue}
    if let json =
        try NSJSONSerialization.JSONObjectWithData(data, options: [])
            as? NSDictionary,
        resultsList = json["results"] as? NSArray,
        results = resultsList.firstObject as? NSDictionary,
        name = results["trackName"] as? String,
        price = results["price"] as? NSNumber {
            let words = name.characters.split(
                isSeparator:{$0 == ":" || $0 == "-"}).map(String.init)
            let n = words.first!
            print ("\(n): \(price)")
    }
}
```

1.2.4 REPL

REPL 表示 Read Eval Print Loop，它是一个位于交互层顶级的 shell，读取用户的表达式，进行逻辑运算，并打印结果。在 Swift 中，REPL 提供了一个即时的测试和开发环境。只要在命令行上运行 Swift，就进入了一个交互式环境。

Swift REPL 在创建函数时会一直保持着输入状态，这一点可以与后面要介绍的结构体和类进行对比。只需要输入表达式进行即时运算即可：

```
% swift
Welcome to Apple Swift version 2.0 (700.0.42.1 700.0.53).
Type :help for assistance.
```

```
  1> print("hello world")
hello world
  2> func greet() {
  3.     print("hello world")
  4. }
  5> greet()
hello world
  6> 1 + 2 + 3 + 4
$R0: Int = 10
```

1.2.5　Playground

Playground 是 Swift REPL 的加强版。从 Playground 中可以得到 REPL 的期望值，但是 Playground 的可视化程度更高，并提供了文档工具(见图 1-6)。Playground 可以让你探索 Swift 语言，测试程序，快速实现解决方案，钻研 Cocoa/Cocoa Touch API 以及创建灵活的交互式文档。

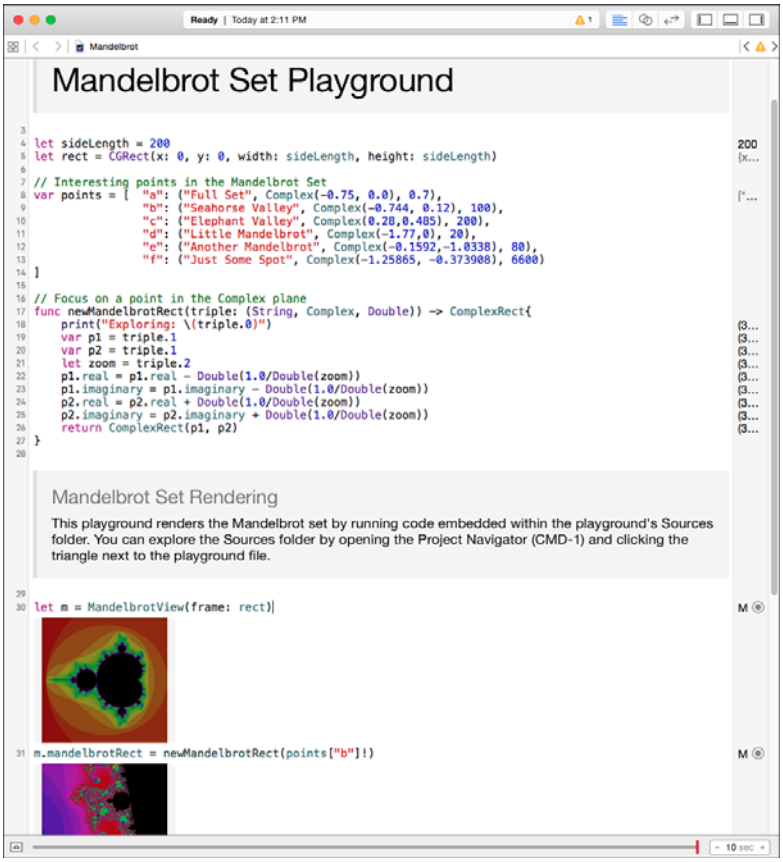

图 1-6　Apple 提供的该 playground 展示了 Mandelbrot 集合，显示了带有美丽的不规则碎片形分界的复杂数字

1.2.6 其他

在 WWDC 2015 上，苹果公司宣布对 Swift 语言进行开源，并为 iOS、OS X 以及 Linux 平台提供源码编译器。那些希望从其他平台上探索 Swift 语言的开发人员可以使用在线解释程序。Swiftstub(http://swiftstub.com)使从任何网络浏览器进行 Swift 编程成为可能(见图 1-7)。

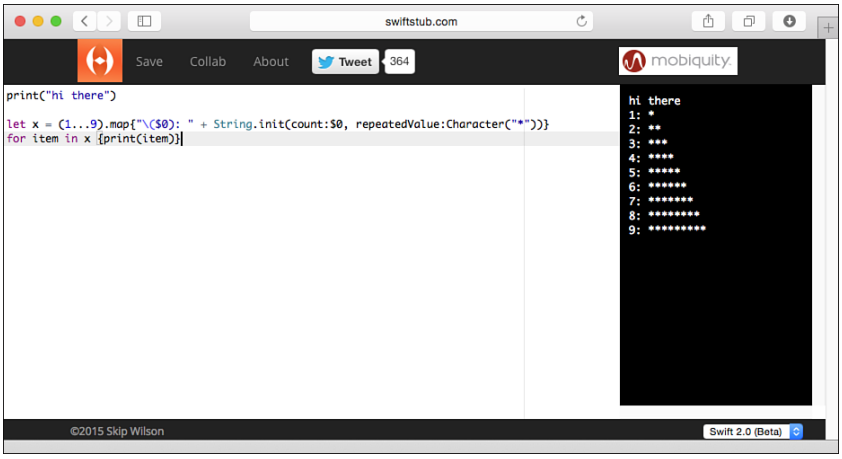

图 1-7　从任何网络浏览器与 Swift 进行交互

1.3 学习 Swift

本书既不是入门教程，也不是详细的参考指南。Apple 的 iBooks 中提供了免费的电子书 *The Swift Programming Language*，下载该书，可以在工作中参考书中的章节。一个全面的 Revision History 部分，可以让你更好地探索语言的变化。

第二本苹果公司的电子书是 *Using Swift with Cocoa and Objective-C*，该书也是非常重要的。它概述了两种语言互操作性的主题和调用 API 的细节。由于该书的紧凑性，该书的篇幅比 *The Swift Programming Language* 一书短得多。

这两本书都可以在苹果公司的网站以及 iBooks 中找到。与 iBooks 有限的书籍相比，笔者更喜欢网络搜索引擎。网络搜索除了使用精确的短语匹配外，还可以使用布尔子句。与此相反，iBooks 搜索仅限于文字和页码。常见的单词匹配(例如 for 或 Switch)在 iBooks 中的处理令人非常不满意。与流行的搜索引擎相比，它并不支持上下文关联匹配。

苹果公司在线的 Swift Standard Library Reference(从网页中搜索它，因为它的网址会定期更改)提供了对 Swift 中不可或缺的基本功能层的概述。它提供了基本数据类型、常见数据结构、函数、方法和协议的概述。如果你停止了对 Swift 语言的基础学习，那么就错过了语言表现力的核心部分。SwiftDoc(http://swiftdoc.org)为 Swift 的标准库提供了自动生成的文档。这与在 Xcode 中按着 Command 键单击标识符所提示的内容相同，只不过现在是更易于阅读的网页。这是一个非常棒的资源。

苹果公司的 Swift 博客(https://developer.apple.com/swift/blog/)大约一个月更新一次，但它的覆盖范围包括不可错过的与语言特点及案例研究相关的主题。其官方声称要窥探 Swift 设计背后的场景，但其关注焦点则是如何让文章更为实用。博客的资源页面(https://developer.apple.com/swift/resources/)还提供了一套 Swift 基础的链接和 Xcode 资料，包括 iTunes U 课程、视频、示例代码，以及到官方的 Swift Standard Library Reference 的链接。

苹果开发者论坛 Apple Developer Forums(https://forums.developer.apple.com/community/xcode/swift)被重新设计，它可以供 Swift 开发工程师访问，工程师们在这里展开激烈的讨论，并提供最新的信息。在老论坛网站(https://devforums.apple.com/index.jspa)上可以通过 Developer Tools | Language | Swift 导航找到已经被归档的 Swift 语言的重要信息。

ASCII WWDC(http://asciiwwdc.com)也是一个令人惊奇的网站。你可以通过输入关键字来搜索往年的 WWDC 上的话题。这个网站可以帮助你追踪 Swift 语言的具体报告和支持 Swift 开发的 Xcode 工具。

如果你使用同行们推荐的互联网聊天工具 Relay Chat(IRC)，在 Freenode(chat.freenode.net)网站的#swift-lang 聊天室中会提供专业级的编码建议。它是一个为语言专门创建的聊天室，如果你需要询问有关操作系统 API 的问题，请访问#iphonedev 或#macdev 聊天室。最后一个聊天室是#cocoa-init，它是一个专门帮助指导 iOS 和 OS X 开发新手的聊天室。

1.4　小结

关于 Swift，苹果公司将致力于重新定义它的开发者工具套件。Swift 2.0 代表这一旅程的开始，而不是结束。它的工具、模块和语言本身将会在可预见的将来继续增长。此时，Swift 停留在一个没有终点的地方。

作为最佳实践，结合设计模式，现在是时候将代码转换到有光明未来的新技术上去。本书接下来的几章中总结了 Swift 开发的重点领域。这些章节将指导你选择需要学习的那些领域，并为你提供来之不易的技术，让你融入代码中去。

祝你好运！