国内第一本含金量超过 Swift 官方文档的原创图书

Swift 权威指南

国内第一本含金量超过 Swift 官方文档的原创图书
第一本将 Swift 和最新的 SpriteKit 游戏引擎深度结合的原创图书
实战性地讲解了 Swift 的开发技术和诀技巧；精彩游戏项目 Flappybird 让读者一览 Swift
项目开发全过程；推出了国内首套 Swift 视频课程；随时提供答疑和完整资源下载

李宁 编著

人民邮电出版社
POSTS & TELECOM PRESS

内 容 提 要

本书共分 20 章，专门介绍了 Swift 的基础语法及进行应用和游戏开发的技术。主要内容包括运算符、字符串、集合类、控制流、函数、枚举类型、类、结构体、属性、方法、下标、泛型、扩展、协议等内容，以及使用 Swift 语言开发 iOS 平台的应用和游戏。最后给出了一个 Flappybird 游戏综合案例，让读者了解使用 Swift 语言开发游戏的完整步骤。

本书适合 iOS 程序员、Swift 初学者学习用书，也可作为大专院校及培训学校的教学用书。

前　言

为什么要写这本书

由于苹果公司一直以来以生产硬件闻名，在 2014 年的 WWDC 上竟然发布了一种新的编程语言—— Swift。这一举动引起了业界不小的震动。在不到 1 个月的时间里，Swift 就挤进流行语言前列，这在以前从未发生过。

Swift 目前可用于开发 iOS 和 OS X 平台的应用和游戏程序。但由于 Swift 刚诞生不久，中文资料还不多，而且由于 Swift 语言具有功能强大和效率开发高的特点，很有可能在将来取代 Objective-C，成为 iOS 和 OS X 平台上的主流开发语言。所以，为了让国内广大程序员能尽早掌握 Swift 开发技术，特意撰写了本书，以便可以让更多的人对 Swift 语言有所了解，更希望让更多的人成为国内乃至世界上第一批 Swift 语言专家。

本书的内容

Swift 语言基础部分（第 1 章~第 17 章）主要介绍了 Swift 语言的基本语法，尤其是和其他语言不同的地方。项目实战部分（第 18 章~第 20 章）主要介绍了如何使用 Swift 语言开发 iOS 平台的应用和游戏，在最后一章还给出了一个 Flappybird 游戏以供大家学习 Swift 项目开发的全过程。

本书适合我吗

当您走进书店，看到书的标题中熟悉的字眼“Swift”，想了解这本书是

否适合自己时，下面的提示对您的选购很有帮助：

- 您听说过 iOS 吗？
- 您知道 App Store 吗？
- 您听说过擅长做硬件的苹果公司居然推出了 Swift 开发语言了吗？

如果上述问题中有一个以上是肯定的，可以很高兴地告诉您，拿在手中的这本书确实是这个方向上的，下面需要进一步确认：

- 您对软件开发有经验或者有兴趣吗？

- 您对开发语言有了解吗？
- 您做过手机应用开发吗？
- 您是 iOS 或移动开发爱好者吗？

如果上述问题，您的回答中有肯定的，那么您已经具备了阅读本书需要的基础，不用担心读不懂了，那么：

- 您想快速了解并进入 Swift 应用开发吗？
- 您想找到一本系统介绍 Swift 开发的参考资料吗？
- 您想选择一本有原理剖析又有真实例子演示的教材吗？
- 您想选一本通俗易懂，符合自己阅读习惯的图书吗？

如上问题中，如果您有大多数回答都是肯定的，那么非常恭喜您，现在拿着的这本书差不多正是您需要的，可以放心地带回去开始自己的 Swift 之旅了。

如果还在犹豫，那么让下面几个提示告诉您，尽早开始学习的重要性：

- IT 界中移动开发的热潮推动了移动互联网的快速发展，而 Swift 是一个非常强大的开发语言，其让您可以快速切入无线互联网领域；
- 在 App Store 发布应用的数量在快速增长，早日发布自己的 App 可以体现自己的开发价值和乐趣；
- 掌握了 Swift 开发就可以很快开发出供全球 iOS 用户使用的应用，有人已经在 App Store 上赚到许多钱了！

本书的特点

●国内第一本含金量超过 Swift 官方文档的原创图书。

●第一本将 Swift 和最新的 SpriteKit 游戏引擎深度结合的原创图书。

●实战性地讲解了 Swift 的开发技术和和技巧。

●精彩游戏应用 Flappybird 让读者一览 Swift 项目开发全过程。

●不仅介绍了 Swift 语言方面的知识，还结合了 iOS 应用和游戏开发进行讲解。尤其是讨论了基于 SpriteKit 的 2D 游戏开发技术。

●推出了国内首套 Swift 视频课程：http://edu.51cto.com/course/course_id-1387.html。

●随时提供答疑和完整资源下载：http://blog.csdn.net/nokiaguy。

读者对象

- 从事 iOS 平台应用和游戏开发的程序员。
- 对 Swift 语言感兴趣的程序员。
- 以前使用 Objective-C，但想摆脱 Objective-C 繁琐的程序员。
- 所有对新知识感兴趣的程序员。

源代码和工具下载

读者可以到作者的 Blog：http://blog.csdn.net/nokiaguy 下载相关的源程序和相关开发工具。

其他学习资源

由于目前 Swift 语言仍然是测试版，所以在读者拿到本书时，Swift 的某些语法或 API 可能会有变化。为此，作者在 51CTO 上开了关于 Swift 的视频课程，这套教材会随着 Swift 的更新而不断更新。

视频地址：http://edu.51cto.com/course/course_id-1387.html。

勘误和支持

由于作者的水平有限，编写时间仓促，书中难免会出现一些错误或不准确的地方，恳请读者批评指正。如有问题或建议，请发送至 techcast@126.com 或在新浪微博（http://weibo.com/638012593）上留言。非常期待能够得到你们的真挚反馈。编辑联系邮箱为 zhangtao@ptpress.com.cn。

致谢

感谢所有在本书写作过程中给予我指导、帮助和鼓励的朋友，尤其是人民邮电出版社的编辑，他们不仅对本书提出了宝贵的写作建议，而且还对本书进行了仔细的审阅。

感谢一直以来信任、鼓励、支持我的家人和朋友。

谨以此书献给我最亲爱的家人，以及众多热爱移动开发的朋友们！

目　　录

第 1 章　未来的 iOS 开发语言

Swift 语言介绍

苹果（Apple）公司最近动作还是比较多的，除了即将推出的 iWatch 等新硬件产品外，还推出了一种新的编程语言 Swift。可能初次接触苹果软件开发的程序员对 Swift 还很陌生，当然，这也不奇怪，因为在写这本书时，Swift 才刚刚推出。不过，由于 Swift 出身贵族，这也注定了 Swift 将拥有一个美好的未来，很可能成为苹果软件开发体系的中坚力量。既然 Swift 如此重要，作为求知欲极强的程序员们怎能放过这么一个成为国内，不！应该说世界上首批 Swift 专家的机会呢！现在就让我们开启 Swift 语言的开发之旅吧！

本章要点

- Swift 语言简介
- Swift 开发环境搭建
- 创建 Swift 工程
- 编写一段简单的 Swift 语言代码
- 所见即所得的 Playground

1.1 Swift 语言的前世今生

在 2014 年的苹果 WWDC 大会上，最大的亮点当属 Swift 的出现了。因为苹果一贯以硬件为主，这次突然弄出了软件，而且还是生产软件的软件：Swift 语言。自然会引起各方的广泛关注。国内外在 24 小时内推出了大量关于 Swift 语言的学习资料，甚至视频[①]。我就从来没看过一种新技术被如此关注过，因为当年我赶上了微软 C#的首发、还有 Google 的 Go 语言首发，关注度都没有 Swift 语言高，看来业界对这个一直玩硬件的苹果突然推出一种新编程语言还是很看好的。这么说当然是有证据的，就在 Swift 语言刚推出不到一个月的时间里，已经排到了编程语言的第 16 位，这在以前从未发生过。

[①] 为了赶上时代的脉搏，我也在 24 小时内推出了 Swift 的学习视频。感兴趣的读者可以关注 http://edu.51cto.com/course/course_id-1387.html

既然说到 Swift 语言，那么就必须要提一下 Swift 的发明者 Chris Lattner（可以叫他克里斯），Chris 博士毕业，是一个全面发展的好学生。

据说 Chris 最喜欢看的 IT 著作是龙书[①]，还喜欢在旅游时带这本书。别人旅游时都看小说或看 Video，这家伙却看编译原理的书，的确高大上。高手就是与别人不一样（就在写这本书时，我已经将龙、虎、鲸 3 本书的英文电子版放到平板电脑里了，准备旅游时看）。

下面主要来谈谈 Chris 的光荣事迹。Chris 在硕士毕业时提出了一套完整的运行时编译思想，奠定了 LLVM[②]的发展基础。在博士期间继续领导 LLVM 编译框架向前发展，并取得了长足的进步。LLVM 已经可以基于 GCC 前端编译器的语义分析结果进行编译优化和代码生成，所以，Chris 在 2005 年毕业时已经是业界知名的编译器专家了。

苹果在 2005 年雇佣了 Chris。Chris 在苹果的几年中不仅大幅度优化和改进 LLVM 以适应 Objective-C 的语法变革和性能要求，同时发起了 CLang 项目，旨在全面替换 GCC，现在这个目标已经实现了。从 OS X 10.9 和 XCode 5 开始，LLVM+GCC 已经被替换成了 LLVM+CLang。

在 2010 年，Chris 接到了一个不同寻常的任务，就是为 iOS 和 OS X 平台开发下一代的编程语言，这就是现在看到的 Swift。最初 Swift 完全是由 Chris 开发的。只是在一年后（2011），才陆续有若干编译器专家加入了 Swift 团队。终于在 4 年后的 2014 年，Swift 的第一个版本在苹果的 2014 年 WWDC 大会上向我们展示了她的魅力。

1.2 Swift 到底是怎样的一种语言

Swift 是一门博采众长的现代语言，在设计的过程中，Chris 参考了 Objective-C、Rust、Haskell、Ruby、Python、C#等优秀语言的特点，最终形成了目前 Swift 的语法特性。这也是为什么使用各种语言的程序员都能从 Swift 中找到自己熟悉的影子的原因。那么，Swift 语言到底是一种怎样的语言。可以从下面几方面初步了解一下 Swift 语言。

[①] 不知道龙书是什么东东的读者自己上网查。总之，有位古人说得好：精通龙、虎、鲸，玩转编译器。也想像 Chris 一样发明编程语言的朋友赶紧囤这 3 本书！

[②] LLVM 是一套架构编译器（Compiler）的框架系统，用 C++编写而成。提供了在编译、连接和运行期间的优化处理，并可以生成相应的汇编代码。目前支持 ActionScript、Ada、D、Fortran、GLSL、Haskell、Java bytecode、C#、Objective-C、Python、Ruby、Rust、Scala 等语言。著名的跨平台 2D/3D 游戏引擎 libgdx（主要使用 Java 语言开发）在底层正因为使用了 LLVM，才可以开发 iOS 平台的游戏。

（1）Swift 是面向对象的、编译型语言。编译时底层需要通过 LLVM 生成本地代码才能执行，所以效率还是很高的。

（2）Swift 可以使用 Cocoa 和 Cocoa Touch 中的 API。这也就意味着 Swift 与 Objective-C 一样，拥有了一个强大的 Framework Library。

（3）Swift 吸取了很多编程语言的优点，同时 Swift 又具备了很多动态语言的语法特性和交互方式，当然，Swift 本身是静态语言。所以，Swift 尽可能地在静态语言和动态语言之间寻找平衡点。

（4）既然说 Swift 是一种拥有动态特性的静态语言，那么 Swift 自然就是一门类型安全的语言。编译器可以在编译过程中检测出类型异常。例如，如果你期望为一个字符串变量赋值，那么类型安全机制会阻止你为这个变量设置整数。正是由于类型安全机制的存在，使开发者可以更早地发现并修复错误。

（5）支持各种高级语言特性，包括闭包、泛型、面向对象、多返回值、类型接口、元组、集合等。

（6）Swift 能与 Objective-C 进行混合编程[①]，但代码分属不同的文件。

（7）全面支持 Unicode 编码。也就是说，可以用任何想用的字符作为变量名，例如，一个笑脸字符或汉字。图 1-1 就是使用笑脸图标和汉字作为变量名的一个典型例子。

```
var 李宁  = "hello world";
var 😃 = "你好吗？"

println(😃)

println("x1=\(x1),李宁=\(李宁),😃=\(😃)");
```

图 1-1　使用笑脸图标和汉字作为变量名

（8）使分号（;）变成了可选的符号。通常的静态语言，如 Java、C#。每条语句结束后，都会在最后加上“;”，而 Swift 的每条语句不需要加“;”，当然，加上也没问题。只有在两条或多条语句写在同一行时才必须加“;”。

（9）简化和增强了集合数据类型。用过 Java 和 C#的读者知道。在这两种语言中，各种集合类型不可谓不全，但太多也有它不好的地方，就是不知道使用哪个。而且这些集合数据类型的功能也不够强大。在 Swift 语言中只提供了数组（Array）和字典（Dictionary）两个集合数据类型。其中 Array 类似 List 的功能，可以修改、添加、替换和删除数组元素。Dictionary

[①] 这一点与 Objective-C 和 C++的混合编程不同。这两种语言的混合编程，代码可以混合放在同一个文件中（.mm）。

类似 Map 的功能，用于存储 Key-Value 风格的值。

（10）Swift 可以通过元组实现函数返回多个值。这一功能在其他语言中需要通过返回一个对象或结构体（指针）来实现。

（11）提供了优雅的闭包解决方案。例如，在排序函数 sort 中可以将函数作为参数值传递。下面的代码是一种典型的写法：

```
let array1 = ["X", "A", "1", "2", "Z"]
func backwards(s1: String, s2: String) -> Bool
{
return s1 > s2
}
var array2 = sort(array1, backwards)
```

当然，更简洁的写法是 var array2 = sort(["X", "A", "1", "2", "Z"]) { $0 > $1 }，如果读者不明白这么写是什么意思，那么就继续往后看吧！

（12）Swift 语言中提供了一种可选变量（Optional）。主要是为了应对一个变量可能存在，也可以是 nil 的情况。例如，将一个字符串（String）转换为整数（Int），但这个字符串是否可以成功转换为整数呢？如果不确定，就返回一个可选变量。如果成功转换，则返回正常的整数值；如果转换失败，则返回 nil。实现的代码如下：

```
let str = "126CB5"
let value = str.toInt()  //  value 是一个可选常量（用 let 声明是常量，用 var 声明是变量）
```

这时，value 就是一个可选变量。要想判断转换是否成功，可以使用下面的代码：

```
if value
{
   //  如果 value 是可选变量，引用时需要加上“!”,
   // 表示该选项变量中肯定有一个值
println(value!)
}
```

可选变量的引入解决了大部分需要显式处理的异常，这部分工作也扔给编译器去做了。

（13）拥有不同意义的 nil。在 Swift 中的 nil 和 Objective-C 中的 nil 不同。在 Objective-C 中，nil 是指向不存在对象的指针，而在 Swift 里，nil 不是指针，它表示特定类型的值不存在。所有类型的可选值都可以被设置为 nil，不仅是对象类型。

（14）Swift 中没有从语言层面支持异步和多核，不过可以直接在 Swift 中复用 GCD（Grand Central Dispatch）的 API 实现异步功能。

（15）Swift 没有一部处理机制。可能是认为有了可选变量，异常会很少使用，所以未加

入异常处理。

1.3 Swift 开发环境搭建

尽管 Swift 语言拥有独立的编译器，不过，目前 Apple 并没有单独发布 Swift 开发包。所以只能连同 XCode 6 一起安装来使用 Swift 语言。由于 XCode 6 仍然是 Beta 版，所以，Apple 并未对所有的人开放下载 XCode 6，只有拥有苹果开发者账号[①]的用户才允许下载 XCode 6 Beta 版。如果读者不舍得那 99 美金也不要紧，我已经将 XCode 6 Beta 的安装文件上传到的网盘，下载地址是 http://pan.baidu.com/s/1hq7mAOk，下载后是一个 dmg 文件（2.33G 左右），直接在 Mac OS X 下双击安装即可。XCode 6 是可以和 Xcode 5 或其他版本共存的，所以不必担心，尽管安装。

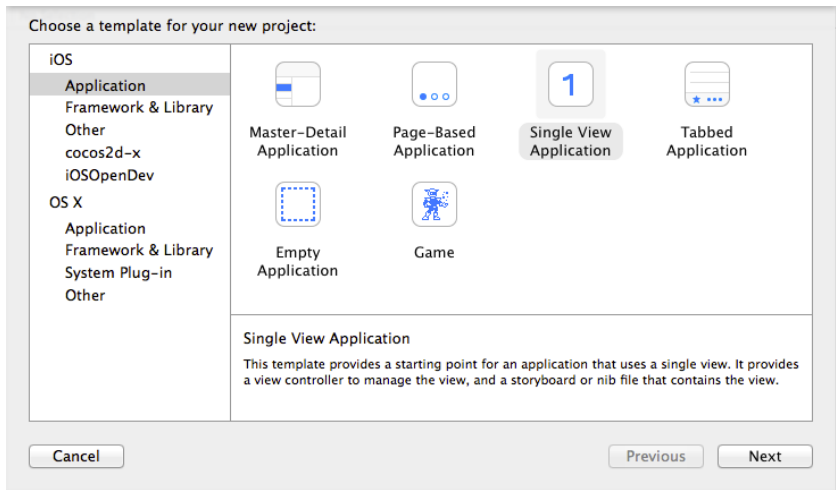
安装完后，在 Mac 的“应用程序”列表中多了一个“Xcode6-Beta.app”，直接双击运行即可。成功运行后，点击 XCode6 中的“Xcode”>“About Xcode”菜单项，如果能看到如图 1-2 所示的窗口，并且 Version 显示的是 6.0，则说明 XCode6 已经安装成功了。

图 1-2　XCode 6 的 About 窗口

[①] 苹果开发者账号 99$/年。现在可以使用支付宝付款，大概一年 600 多人民币，感兴趣的读者可以到苹果官网去购买。因为只有拥有苹果开发者账号，才允许在 iPhone、iPad 设备上运行程序，并且可以下载任何处于 Beta 版本的开放工具（包括 XCode6）。

1.4 创建 Swift 工程（OS X 和 iOS 平台）

Swift 语言可以开发 iOS 和 OS X 两个平台的程序，读者可以选择自己喜欢的平台学习 Swift 语言。如果有苹果开发者账号的，最好选择 iOS 平台，因为可以在 iPhone 上测试全部的功能。如果没有开发者账号，可以考虑选择 OS X 平台，否则就只能在 iOS 模拟器上测试 Swift 程序了。不管是创建哪个类型的工程，首先需要点击“File”>“New”>“Project”菜单项打开如图 1-3 所示的工程模板选择窗口。

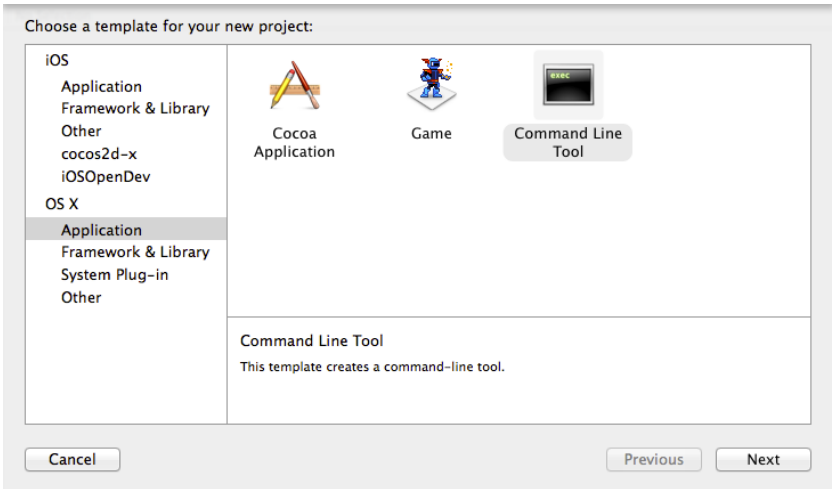


▲图 1-3　创建 iOS 工程

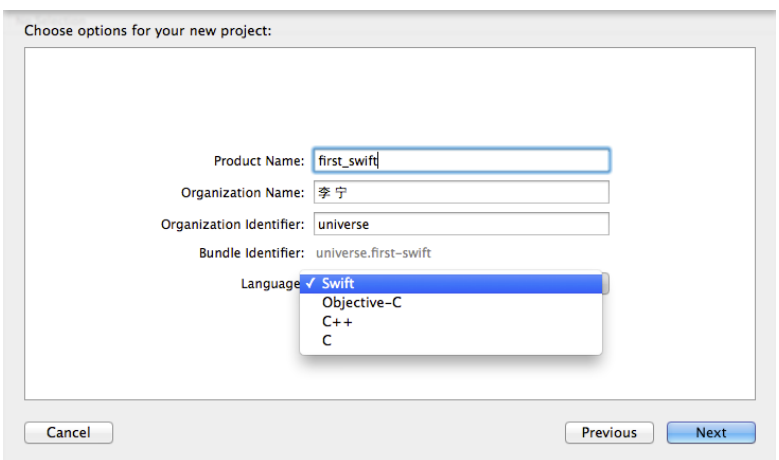
如果选择 iOS 平台，通常选择“Application”，然后在右侧工程模板列表中选择一个，如“Single View Application”，如图 1-3 所示。

如果读者选择 OS X 平台，可以选择“Application”，并在右侧的工程模板列表中选择“Command Line Tool”（一个终端工程模板），如图 1-4 所示。

接下来可以点击“Next”按钮进入下一步的设置。在下一步中需要输入工程名和工程支持的语言(为了测试方便，这里选择了 OS X 平台，等涉及 iOS 部分时，再创建 iOS 工程)。当然，我们需要选择“Swift”，如图 1-5 所示，然后点击“Next”按钮进入下一步。在下一步中需要选择一个用于存放工程的目录，选择完成后，点击“Create”按钮，就会创建一个 OS X 工程。



▲图 1-4　创建 Mac 工程



▲图 1-5　建立 iOS 工程，选择 Swift 语言

图 1-5 是 OS X 工程的机构，其中新创建了一个 main.swift 文件。该文件用于输入 Swift 语言的代码。

在 main.swift 文件中默认生成了如下的代码：

```
import Foundation
println("Hello, World!")
```

直接运行该工程，就会在输出视图中输出“Hello, World!”。

如果读者要建立 iOS for Swift 工程，这里假设选择了图 1-3 所示的“Single View Application”模板，在下一个设置页面输入工程名（如 first_swift_ios）后再进入下一个设置页面，选择存储目录后可创建 iOS 工程，工程结构如图 1-7 所示。

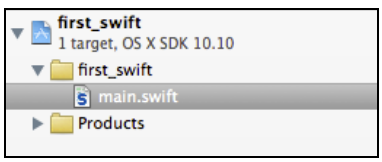


图 1-6　OS X 工程结构

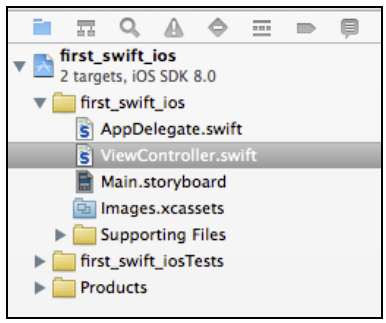


图 1-7　iOS 工程结构

如果运行这个工程，首先会启动 iOS 模拟器（要选择 iOS8 模拟器），然后在模拟器中会运行一个 iOS 程序，该程序并没有任何 UI，背景为白色。本书后面的章节会介绍如何使用 Swift 开发 iOS 应用和游戏。

对于只想学习一下 Swift 语言的读者，建议还是使用 OS X 终端程序为好，因为 OS X 终端程序运行快速简单，可以排除其他的干扰。

1.5 瞧一瞧 Swift 到底长啥样

可能很多读者看到前面的介绍后跃跃欲试，想试一试 Swift 语言，不过先别忙，在正式编写 Swift 语言之前，先了解一下 Swift 语言的语法和样式是非常必要的。就像搞对象，起码得先看看对方长啥样，然后才能进行下一步。现在就让我们看看 Swift 到底长啥样。

Swift 语言和其他语言（如 Java、C#、Objective-C）类似，也支持一些常规的操作和语法元素，例如，运算符（+、−、*、/、%、++、&&、||等）、数据类型（字符串、数值等）、控制语句（if、switch、for）、类、方法（函数）等。不过 Swift 添加了很多特有的语法元素（至少是大多数语言都不具备的技术），例如，类扩展、增强的 switch 语句、小标、元组类型、返回多种的方法（函数）等。下面是一段标准的 Swift 语言代码，尽管没有枚举 Swift 中所有的新特性，但足够让我们充分了解 Swift 语言的基本语法规则。如果读者对某些表达式不理解也没关系，在后面的章节会深入介绍这些特有的语法规则。

```
import Foundation
```

```
var value1 = 20    //  定义并初始化一个整型变量
value1 += 35
//  输出：value1=55
println("value1=\(value1)")
let value2 = 42    //  定义并初始化一个整型常量
//  输出：97
println(value1 + value2)
//  定义并初始化一个数组变量
var array1 = ["abc","bbb","ccc"]
//  枚举数组中的每一个元素值
//  输出：abc  bbb  ccc
for element in array1
{
   print(element + "  ")
}
//  定义函数，返回值类型是 String
func getPerson(name: String, age:Int) -> (String)
{
   return "name:" + name + "  age:" + toString(age)
}
println()
//  调用 getPerson 函数
let person = getPerson("bill", 50)
//  输出：person:name:bill  age:50
println("person:\(person)")
//  定义一个可以返回多个值的函数（返回两个值：name 和 price）
func getProduct() -> (name:String, price:Float)
{
   return ("iPhone6", 6666.66)
}
//  输出：(iPhone6, 6666.66015625)
//  调用 getProduct 函数，并输出其返回值
println(getProduct())
//  输出：product name: iPhone6
//  输出 getProduct 函数返回 name 值
println("product name: \(getProduct().name)")
//  定义 Country 类
class Country
{
   var name = "China"
   let area: Float = 960
   //  定义一个成员方法
   func getDescription() -> String {
      return "Country:\(name)  Area:\(area)"
   }
}
```

```
//  创建一个 Country 类型的常量
let country = Country()
//  输出：Country:China  Area:960.0
//  调用 Country.getDescritpion 方法，并输出该方法的返回值
println(country.getDescription())

//  定义一个类扩展，相当于将一个类的某一部分拿出来单独定义
extension Country
{
    //  为 Country 类添加一个方法
    func getAddress() -> (planet: String, location:String)
    {
        return ("Earth","Asia")
    }
}
//  输出：Planet:Earth
//  调用扩展方法 getAddress，并输出返回值中的 planet
println("Planet:\(country.getAddress().planet)")
//  输出：Location:Asia
//  调用扩展方法 getAddress，并输出返回值中的 location
println("Location:\(country.getAddress().location)")
```

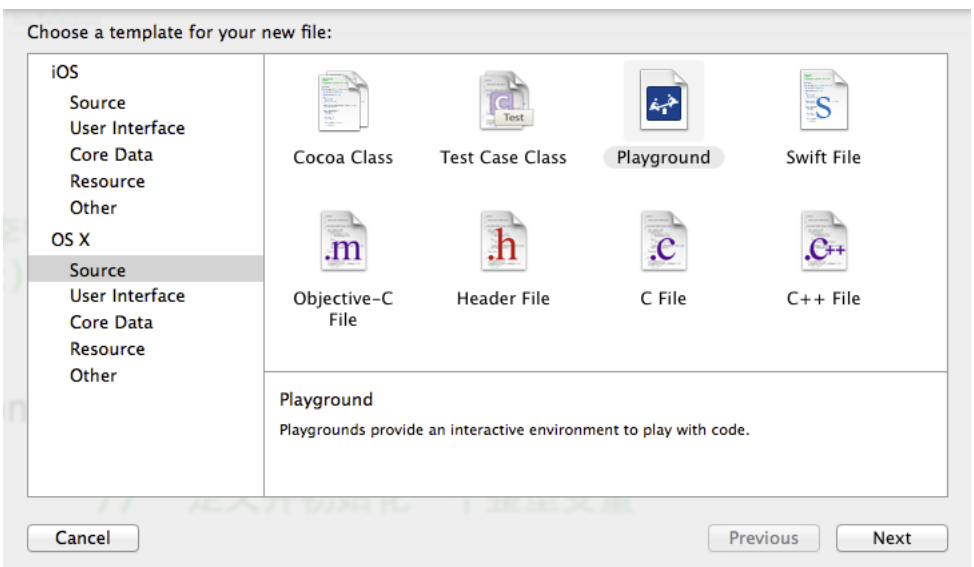
如果执行上面这段代码，会输出如下的信息。

```
value1=55
97
abc    bbb    ccc
person:name:bill    age:50
(iPhone6, 6666.66015625)
product name: iPhone6
Country:China    Area:960.0
Planet:Earth
Location:Asia
```

1.6 所见即所得的 Playground

XCode 6 新增加了一个 Playground，在 iOS 和 OS X 工程中都可以使用 Playground。这个东西其实就是一个所见即所得的写代码的文件。这里的所见即所得就是编写代码后，不需要运行，立刻会显示出结果，并且还可以显示出代码中某个变量/常量的当前值。Playground 比较适合于测试 Swift 语言的代码。但 Playground 在输出结果时可能会有些慢，而且写代码时会有些迟钝。这可能是因为需要实时解释 Swift 代码的原因！

要想使用 Playground，需要先创建 Swift 工程（iOS 和 OS X 工程都可以），然后在当前工程中点击“File”>“New”>“File”菜单项，会弹出一个如图 1-8 所示的窗口。在“iOS”和“OS X”中都有一个“source”节点，在该节点中都有一个“Playground”模板。不管是什么工程，选择哪个“source”节点下的“Playground”都可以。



▲图 1-8　选择“Playground”模板

选择“Playground”模板后，进入下一个设置页面。在该设置页面中需要选择 playground 文件的存储目录，通常和 Swift 文件放在同一个目录中。如果下方的“Targets”列表中的工程未选择，需要选择该工程，效果如图 1-9 所示。

如果成功创建了 Playground 文件，在工程目录中会多了一个文件扩展名为“playground”的文件。默认是 MyPlayground.playground。如果读者将上一节的代码都复制到

MyPlayground.playground 文件中，就会在代码右侧输出相应的结果，并且在输出结果和代码之间的部分还会显示当前变量/常量的当前值，效果如图 1-10 所示。

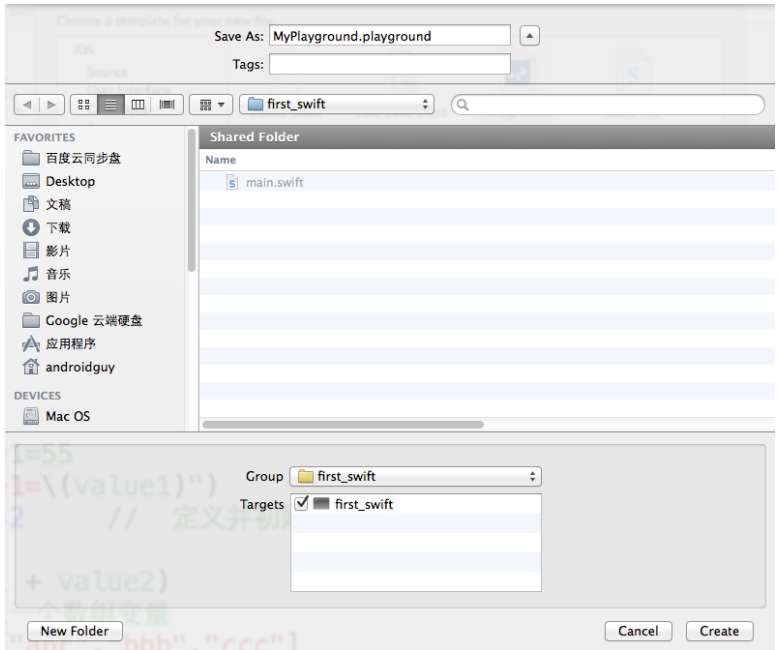


图 1-9　创建 Playground 文件

图 1-10　Playground 的效果

1.7 小结

相信看完本章后，读者已经对 Swift 语言有了一个大致的了解。也学会了如何搭建实验环境。不过真正的 Swift 开发之旅才刚刚开始。后面还有 Swift 的语法、内置函数、iOS 应用程序开发、基于 SpriteKit 游戏引擎的游戏程序开发等知识要学习。下面就让我们整装待发，去欣赏 Swift 给我们的一次又一次震撼吧！

第 6 章　丰富多彩的功能——函数

Swift 语言为函数提供了丰富多彩的功能。但从功能上来说，Swift 函数和其他语言的函数相比是非常强大的，在后面章节要介绍的方法与函数基本上一致。不过在本章先不讨论方法的细节，只介绍函数的各种功能和使用方法。

本章要点

- 函数的定义和调用
- 返回多种的函数
- 扩展参数
- 扩展参数和内部参数使用同一个名字
- 默认参数值
- 可变参数
- 常量和变量参数
- 输入输出参数
- 函数类型
- 嵌套函数

6.1 函数的定义和调用

源代码文件：src/ch06/function/function/main.swift

Swift 并不是纯的面向对象语言，所以和 C++一样，支持函数。函数和方法的区别就是函数是全局的，而方法的作用域仅限于方法内。要想引用方法，必须先要引用包含该方法的对象。也可以将方法看成是定义在类中的函数。因此，函数和方法的定义

规则基础一致。在后面讲到类时再详细讨论方法的细节。本节先来讨论如何定义一个函数。

不管是什么语言，函数都必须由下面几部分组成。

- 函数名。
- 返回值类型。
- 函数的参数列表，在列表中包含参数名（形参）和参数类型。
- 函数体。

定义 Swift 语言的函数也逃不出这几项。下面看一下 Swift 函数的语法形式。

```
func functionName(paramName1:paramType1, paramName2,paramType2,...) -> returnType
{
   function body
}
```

很明显，Swift 函数和 C 语言的函数在定义上差异很大。首先，Swift 函数必须以 func 开头，然后跟着函数名，接下来是函数参数列表，最后是返回值类型。其中函数参数列表和返回值类型之间需要用“→”分隔。最后需要用一对花括号（{...}）将函数体括起来，这里面函数参数类型列表和返回值类型都是可选的。如果不指定函数参数列表，则函数没有参数，但必须在函数名后面指定一对圆括号。如果不指定返回值类型，则函数没有返回值，相当于 C 语言函数前面指定了 void。

下面是一个标准的 Swift 函数的代码，该函数接收一个 String 类型的参数，返回一个 String 类型的值，最后调用了 sayHello 函数，并输出了函数的返回值。

```
func sayHello(personName: String) -> String
{
   let greeting = "hello " + personName + "!"
   return greeting
}
//  调用 sayHello 函数
println(sayHello("李宁"))
```

执行这段代码后，会输出如下内容。

hello 李宁!

下面是一些其他形式的函数（多个参数、没有参数、没有返回值）。

```
//  多个参数的函数
func add(a:Int, b:Int) -> Int
{
   return a + b
```

```
}
//  调用 add 函数
println(add(20, 30))
//  没有参数，但又返回值的函数
func process() -> Float
{
   return 3*20
}
//  调用 process 函数
println(process())
//  既没有参数，也没有返回值的函数
func method()
{
   println("hello world")
}
//  调用 method 方法
method()
```

执行这段代码后，会输出如下内容。

```
50
60.0
hello world
```

6.2 返回多值的函数

源代码文件：src/ch06/function/function/main.swift

不管是数学上的定义，还是各种语言中的实现，函数都只能返回一个值，如果非要返回多个值，就返回一个对象，然后将要返回的值以字段、属性或方法形式体现，但这也是返回一个值。不过在 Swift 函数中，却彻底颠覆了我们对函数的印象。Swift 函数是真真正正地可以返回多个值。

可能有的读者会想到，在前面学习数据类型时，有一个元组类型，这个类型可以同时表示多个值，难道函数返回的是这个类型的值？没错，Swift 函数就是通过元组类型实现返回多个值的功能的。

第 19 章　开启游戏开发之旅——SpriteKit 游戏引擎

本章详细分析了 SpriteKit 默认游戏工程的基本结构，读者通过本章的学习，可以初步掌握一个基于 SpriteKit 的游戏工程的结构，以及编写游戏的基本步骤。

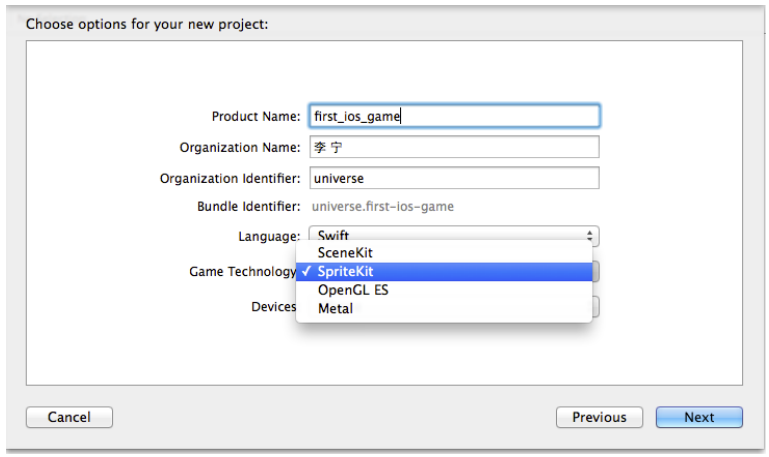
本章要点

- 创建游戏工程
- 游戏工程的结构
- 运行游戏工程
- 默认游戏工程原理分析

19.1 创建游戏工程

创建 iOS 游戏工程和应用工程类似，只是需要选择 game 工程模板，在工程信息设置页面中，多了一个 Game Tchnology 列表，如图 19-1 所示。在该列表中可以选择多种游戏技术。默认是 SpriteKit。这是一个内置的 2D 游戏引擎。如果想开发 3D 游戏，可以选择 SceneKit。当然，也可以选择 OpenGL ES 或 Metal。不过，这几个不在本章的讨论范围内。本章只讨论 2D 游戏引擎 SpriteKit 的用法。

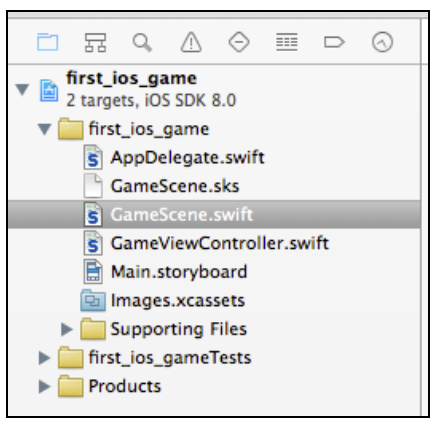
当输入完游戏工程信息后，剩下的工作就和创建应用工程完全相同了。一切都搞定后，最后单击“Create”按钮就会创建游戏工程。



▲图 19-1　创建 SpriteKit 游戏工程

19.2 游戏工程的结构

创建完游戏工程后，会看到如图 19-2 所示的工程结构。



▲图 19-2　游戏工程结构

我们可以看到，游戏工程和应用工程的目录结构类似。只是多了一个 swift 文件和 sks 文件。其中 sks 文件是可视化场景编辑器。不过除非是往窗口上放控件，否则通常是直接通过程序在窗口上放置各种游戏元素，因为这样更灵活。

现在看一看工程中 3 个主要的 swift 文件的含义。

AppDelegate.swift 文件与应用工程中的同名文件的功能完全一样，用于处理整个游

戏程序的事件，如游戏启动、游戏切换到后台（一般在这个事件中关掉游戏的背景音乐和其他声音）、游戏切换到前台（一般在这个事件中重新播放游戏的背景音乐和其他声音）等。

GameViewController.swift 文件和应用工程中的 ViewController.swift 文件的功能类似，不过前者用于显示场景[①]，后者用于显示窗口。

GameScene.swift 文件中定义了一个场景类 GameScene。在 GameViewController 类中会显示该场景。所有的游戏元素都会添加到这个场景中。

19.3 运行游戏工程

默认生成的游戏工程是一个非常简单的程序。在屏幕中心显示一个很大的“Hello World”，然后当鼠标光标或手指触摸屏幕时，就会在屏幕的触摸点显示一个不断选择的小飞机，效果如图 19-3 所示。

▲图 19-3　默认游戏工程的运行效果

[①] 场景是游戏中的重要概念，可以理解为游戏中的窗口。

Swift 权威指南

本书最佳学习流程：

千里之行始于足下——Swift 语言基础
万丈高楼平地起——基本操作符
此字典非彼字典——数组和字典
千变万化的程序——控制流
丰富多彩的功能——函数
代码之美的诠释——闭包
特殊的数据——枚举类型
Swift 语言的核心——类和结构体
容易犯错的地方——类的继承
内存管理机制——ARC
让程序不抛出异常顺畅运行——可选链
Swift 语言的魔法——类型转换
为程序增添无限动力——扩展
开发大型程序必备元素——协议
增强代码的灵活性——泛型
私人定制—高级操作符
做未来的 iOS 程序员——Swift 开发 iOS 应用技术
开启游戏开发之旅——SpriteKit 游戏引擎
综合应用实战——Flappybird 游戏开发

本书支持社区

51CTO学院
为梦想喝彩！

封面设计：董志桢

分类建议：计算机／Swift
人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-36847-8