Swift3.0从入门到发疯

目录

第一章 前言 1

1. 作者简介 1

2. 本书初衷 1

3. 本书内容简介 1

第二章 Swift3.0语言概述 2

2.1 Swift语言概述 2

2.1.1 什么是Swift语言 2

2.1.2 语言特点 2

2.1.3 开发框架 2

2.1.4 Swift与Objective-C语言对比 2

2.2 Swift开发环境和工具 2

2.2.1 开发环境 2

2.2.2 Xcode概述 2

2.2.3 安装Xcode 8开发工具 2

2.2.4 Swift项目结构 2

2.2.5 帮助文档 2

2.2.6 学习工具—playground 2

2.3 第一个Swift程序 2

2.4 本章小结 2

2.5 本章习题 2

第三章 Swift3.0基本语法 2

3.1 关键字和标识符 2

3.1.1 关键字 2

3.1.2 标识符 3

3.2 常量和变量 3

3.2.1 常量和变量的声明 3

3.2.2 类型推断和类型安全 3

3.2.3 输出常量和变量 3

3.3 简单数据类型 3

3.3.1 整型 3

3.3.2 浮点型 3

3.3.3 布尔类型（Bool） 3

3.4 元组类型 3

3.4.1 元组的声明 3

3.4.2 元组变量的访问 3

3.5 基本运算符 3

3.5.1 赋值运算符 3

3.5.2 单目负运算符 3

3.5.3 算术运算符 3

3.5.4 比较运算符 3

3.5.5 三目运算符 3

3.5.6 逻辑运算符 3

3.6 区间运算符 3

3.6.1 闭区间运算符 4

3.6.2 半闭区间运算符 4

3.7 Optional可选类型 4

3.7.1 可选类型的声明 4

3.7.2 解包（Unwrapping） 4

3.7.3 隐式解析可选类型 4

3.8 本章小结 4

3.9 本章习题 4

第四章 Swift3.0控制流 4

4.1 条件语句 4

4.1.1 if条件语句 4

4.1.2 if-let语句 4

4.1.3 guard语句 4

4.1.4 switch语句 4

4.2 循环语句 4

4.2.1 for-in循环 4

4.2.2 while循环 4

4.2.3 repeat-while循环 4

4.3 本章小结 4

4.4 本章习题 4

第五章 Swift3.0字符串和集合 5

5.1 字符 5

5.1.1 字符概述 5

5.1.2 转义字符 5

5.2 字符串 5

5.2.1 初始化字符串 5

5.2.2 字符串的基本操作 5

5.2.3 字符串的高级操作 5

5.3 集合（Collection） 5

5.3.1 创建数组（Array） 5

5.3.2 数组的常见操作 5

5.3.3 Set 5

5.3.4 字典（Dictionary） 5

5.4 本章小结 5

5.5 本章习题 5

第六章 Swift3.0函数、闭包和枚举 5

6.1 函数 5

6.1.1 函数的定义和调用 5

6.1.2 函数的参数和返回值 5

6.1.3 局部参数名和外部参数名 6

6.1.4 函数参数的其他用法 6

6.1.5 嵌套函数 6

6.2 闭包 6

6.2.1 闭包的概念和定义 6

6.2.2 使用尾随闭包 6

6.2.3 使用闭包表达式 6

6.2.4 捕获 6

6.3 枚举 6

6.3.1 枚举的定义和访问 6

6.3.2 使用Switch语句匹配枚举值 6

6.3.3 原始值 6

6.4 本章小结 6

6.5 本章习题 6

第七章 Swift3.0面向对象（上） 6

7.1 面向对象概述 6

7.2 类和结构体 6

7.2.1 类和结构体的定义 6

7.2.2 类和结构体的实例 6

7.2.3 类和结构体对比 6

7.3 属性 7

7.3.1 存储属性 7

7.3.2 懒存储属性 7

7.3.3 计算属性 7

7.3.4 属性观察器 7

7.3.5 类型属性 7

7.4 方法 7

7.4.1 实例方法 7

7.4.2 类型方法 7

7.5 构造函数 7

7.5.1 构造函数基础 7

7.5.2 重载构造函数 7

7.5.3 指定构造函数与便利构造函数 7

7.6 析构函数 7

7.7 下标脚本 7

7.7.1 下标脚本语法 7

7.7.2 下标脚本的使用 7

7.8 本章小结 7

7.9 本章习题 7

第八章 Swift3.0面向对象（下） 7

8.1 面向对象的三大特性 8

8.2 继承和重写 8

8.2.1 继承的概念 8

8.2.2 继承的实现 8

8.2.3 重写 8

8.2.4 final关键字的使用 8

8.2.5 super关键字的使用 8

8.3 构造函数的继承和重写 8

8.3.1 构造函数的调用规则 8

8.3.2 构造过程的安全检查 8

8.3.3 构造函数的自动继承 8

8.3.4 构造函数的重写 8

8.4 封装和多态 8

8.4.1 封装 8

8.4.2 多态 8

8.5 可选链 8

8.5.1 可选链与强制展开 8

8.5.2 可选链访问属性、方法和下标 8

8.6 类型检查和转换 8

8.6.1 类型检查（is操作符） 8

8.6.2 类型转换（as操作符） 9

8.6.3 Any和AnyObject的类型转换 9

8.7 嵌套类型 9

8.8 本章小结 9

8.9 本章习题 9

第九章 Swift3.0扩展Swift内存管理 9

9.1 扩展 9

9.1.1 扩展概述 9

9.1.2 扩展计算型属性 9

9.1.3 扩展构造函数 9

9.1.4 扩展方法 9

9.1.5 扩展下标 9

9.2 协议 9

9.2.1 协议概述 9

9.2.2 协议的要求 9

9.2.3 协议作为类型使用 9

9.2.4 协议的继承 9

9.2.5 检查协议一致性 9

9.2.6 代理模式 9

9.3 扩展和协议的结合 9

9.3.1 通过扩展采纳协议 10

9.3.2 协议扩展 10

9.4 本章小结 10

9.5 本章习题 10

第十章 Swift3.0 内存管理 10

10.1 Swift内存管理机制 10

10.1.1 自动引用计数工作机制 10

10.1.2 自动引用计数示例 10

10.1.3 类实例之间的循环强引用 10

10.1.4 解决类实例之间的循环强引用 10

10.2 闭包引起的循环强引用 10

10.2.1 闭包引起的循环强引用 10

10.2.2 解决闭包引起的循环强引用 10

10.3 本章小结 10

10.4 本章习题 10

第十一章 Swift3.0的其他高级特性 10

11.1 泛型 10

11.1.1 泛型函数 10

11.1.2 泛型类型 10

11.1.3 类型约束 11

11.1.4 关联类型 11

11.1.5 where子句 11

11.2 错误处理机制 11

11.2.1 错误的表示 11

11.2.2 错误处理 11

11.2.3 清理操作 11

11.3 访问控制 11

11.3.1 模块、源文件及访问级别 11

11.3.2 类型的访问级别 11

11.3.3 变量常量属性下标及构造函数的访问控制 11

11.3.4 协议扩展的访问控制 11

11.4 命名空间 11

11.4.1 查看和修改命名空间 11

11.4.2 使用命名空间 11

11.5 高级运算符 11

11.5.1 位运算符 11

11.5.2 溢出运算符 11

11.5.3 优先级和结合性 11

11.5.4 运算符函数 11

11.5.5 自定义运算符 12

11.6 本章小结 12

11.7 本章习题 12

第十二章 Swift与Objective-C的相互操作 12

12.1 Swift项目中调用Objective-C类 12

12.1.1 实现原理分析 12

12.1.2 创建Swift项目 12

12.1.3 新建Objective-C类 12

12.1.4 在Swift项目中调用Objective-C代码 12

12.2 Objective-C项目中调用Swift类 12

12.2.1 实现原理分析 12

12.2.2 创建Objective-C项目 12

12.2.3 新建Swift类 12

12.2.4 在Objective-C项目中调用Swift代码 12

12.3 Objective-C项目到Swift项目的迁移 12

12.3.1 准备工作 12

12.3.2 迁移到Swift项目 12

12.4 本章小结 12

12.5 本章习题 12

第十三章 Swift3.0 IOS UI/GUI开发 13

第十四章 Swift3.0项目实战 13

第十五章 尾语 13

## 前言

### 作者简介

作者姓名:王衡杰

邮箱: hundredsir@gmail.com

博客名:香蕉大大(简书)

微信公众号:大大家的iOS说

微信号:hundreda

毕业于:广东某工业大学

一枚一直在路上的开发者,希望能用自己微小的力量给大家带来一些借鉴以及经验参考!

### 本书初衷

想了很久,一方面以前一直看大神的书成长,但是觉得感觉还是那一句,人在江湖走,出来混迟早要还的,所以我想法很简单,在如今蔚然大观的人潮中,怎么总结一套自己的资料并且供大家传阅,可能就是我今年年末最后的目标.

如果说本书的想法,不知道有没有机会录制成视频,但是鉴于如今参考资料繁多,所以我还是站在一个点,希望能和大家分享,并且通过开源的方式一起把此书做好,也希望大家能给我提点建议,给大家多一些正确的参考.

本书于2016年12月26日完结第一稿,很多不足的地方希望大家能多给点建议,再次谢谢大家拜读,也希望大家能通过鄙人短浅的认识,开始swift3.0的世界!

### 本书内容简介

## Swift3.0语言概述

## 2.1 Swift语言概述

Swift是Apple在WWDC2014所发布的一门编程语言，用来撰写OS X和iOS应用程序。在设计Swift时．就有意和Objective-C共存，Objective-C是Apple操作系统在导入Swift前使用的编程语言,所以说如果之前做IOS开发使用OC的朋友,学习Swift简直小菜一碟.

### 2.1.1 什么是Swift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。

### 2.1.2 语言特点

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）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[ 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）。]，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。

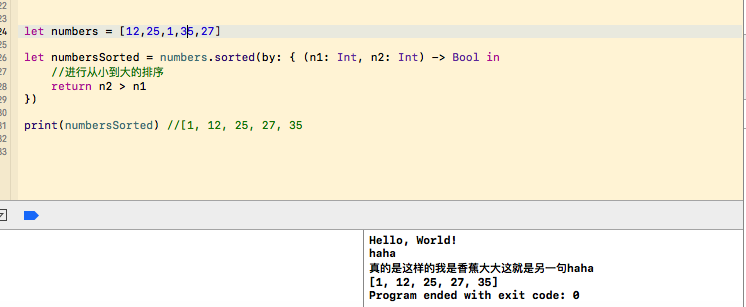


（8）使分号（;）变成了可选的符号。通常的静态语言，如Java、C#。每条语句结束后，都会在最后加上“;”，而Swift的每条语句不需要加“；”，当然，加上也没问题。只有在两条或多条语句写在同一行时才必须加“;”。

（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：



（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：

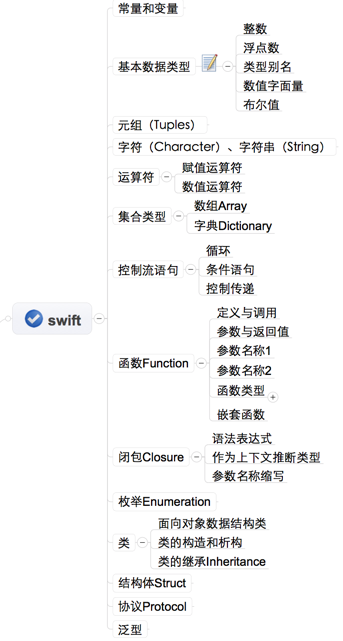


(13)拥有不同意义的nil。在Swift 中的 nil 和 Objective-C 中的 nil 不同。在 Objective-C 中，nil 是指向不存在对象的指针，而在 Swift 里，nil 不是指针，它表示特定类型的值不存在。所有类型的可选值都可以被设置为nil，不仅仅是对象类型。

(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。

### 2.1.3 开发框架

我觉得这里只需要给大家两幅图,简单明了

iOS学习框架 swift学习框架

### 2.1.4 Swift与Objective-C语言对比

WWDC 2014上苹果再次惊世骇俗的推出了新的编程语言Swift 雨燕, 这个消息会前没有半点风声的走漏。消息发布当时，会场一片惊呼，相信全球看直播的码农们当时也感觉脑袋被敲了一记闷棍吧。于是熬夜学习了Swift大法， 越看越想高呼 ” Swift大法好！“

程序员，最讲究的就是实事求是和客观，下面就开始对比两种语言。

首先要强调的是，Swift绝对不是解释性语言，更不是脚本语言，它和Objective-C，C++一样，编译器最终会把它翻译成C语言，也就是 说编译器最终面对的其实都是C语言代码（ 这是千真万确，不容置疑的！！！所以不要看它长的想脚本语言，其实它是比Java, C#要高效的多的C语言！！！），但是Swift的强大之处在于它站在所有语言的肩膀上，吸取所有语言的精华。

这个系列我们先谈谈几个最基本的语法变化：

1. Swift终于放弃了Objective-C那幺蛾子般的 [ obj method:x1 with:x2] 的语法，终于跟随了大流，变成了obj.method( )的顺眼模式。虽然对于Objective-C的程序员来说，这些[ ]看上去特显酷 ， 你们知道就是这个中括弧吓跑了多少c++, java , c#的程序员嘛？所以说这个小小的变化，可以让苹果的开发更平易近人，对有其他开发语言基础的人来说更友好。

但苹果不会这么自甘平庸，我们知道Objective-C里方法的调用有种语法是其他主流语言没有的，那就是标签。我们在使用java, C++, C, C#等语言时，如果使用 rect.set( 10, 20, 100, 500 ), 虽然在写set方法的时候，IDE有提示四个形参的含义，但写完后，这句代码中10， 20， 100， 500是社么意思? 所以代码的可读性就变的很差， 而Objective-C很完美的解决了这个问题 ：

1. ［ rect setX:10 y:20 width:100 height:500 ］

看看！多棒！Swift当然不会丢弃这么好的传统， 在Swift中是这个样子的

1. rect.set( 10, y:20, width:100, height:500 )

对于调用方法时，标签的使用需要注意以下两点：

**1）全局函数的调用，不可以使用标签**

1. fun( 11, 22 , 33 ) //正确
2. fun( n1:11, n2: 22, n3: 33 ) //错误

**2 ) 类的函数，第一个参数不可以加标签**

1. rect.set( 100, y:100, width:300, height: 200 ) //正确
2. rect.set( x:100, y:100, width:300, height: 200 ) //错误

其实Swift中对类的定义和java, c#几乎一样，再也不分头文件和.m文件了。

一个类的定义语法如下：

1. **class** Weapon
2. {
3. var name:NSString
4. var power:**int**
5. init( name:NSString, power:**int** )
6. {
7. self.name = name
8. self.power = power
9. }
10. func shoot( )
11. }

**注意**：Swift中的语句不需要分号结束.其次，终于有构造函数和析构函数了！！！Objective-C也有？no no no!!!

Objective-C中没有构造函数，真正的构造函数是系统自动调用的，而不是强迫程序员去调用。以前要强迫程序员[ [ Obj alloc ] init ]，现在终于终于终于系统自动调用了！

1. Weapon weapon = Weapon( name:“人间大炮”, power: 100000000000 )

我没有写错， 对的！现在和java,C#一样！虽然weapon是一个指针，但不要写那颗星号！！因为这颗星号吓死过好多人！“ 啥？指针？！！啊。。。。。”

C、 C++的程序员注意了，这个weapon对象不是分配在栈内存上的， 仍然是alloc出来的， 在堆上呢。

1. 最期待的语法终于加入了！

对于override在Java,C++,Objective-C中都有问题，举个栗子：

1. @interface Weapon
2. -(**void**)shoot;
3. @end
4. @interface Gun : Weapon
5. -(**void**)Shoot;
6. @end

在大项目中经常会遇到这个问题，程序员的本意是希望覆盖父类的shoot ,结果手潮。。。。写成了Shoot ， 这既没有语法错误，也没有逻辑错误，结果在

1. Weapon\*  currentWeapon = [ Gun **new** ];

[currentWeapon shoot ] 中调用的却是父类的shoot方法（ 因为子类根本没有覆盖啦，子类大小写不小心写错了 ）， 这种小错误如果出现在超大型项目种还真是很难找呢！！现在，Swift终于解决这个问题啦！ 子类覆盖父类方法的时候，一定要在方法前写上override ：

## 2.2 Swift开发环境和工具

### 2.2.1 开发环境

### 2.2.2 Xcode概述

### 2.2.3 安装Xcode 8开发工具

### 2.2.4 Swift项目结构

### 2.2.5 帮助文档

### 2.2.6 学习工具—playground

## 2.3 第一个Swift程序

## 2.4 本章小结

## 2.5 本章习题

## Swift3.0基本语法

## 关键字和标识符

**任何一种计算机语言都离不开标识符和关键字，下面我们将详细介绍[Swift](http://lib.csdn.net/base/swift" \o "Swift知识库" \t "_blank)标识符和关键字**。

### 3.1.1 关键字

关键字是类似于标识符的保留字符序列，除非用重音符号（`）将其括起来，否则不能用作标识符。关键字是对编译器具有特殊意义的预定义保留标识符。常见的关键字有以下4种。

1. 与声明有关的关键字：class、deinit、enum、extension、func、import、init、let、protocol、static、struct、subscript、typealias和var。

2.与语句有关的关键字：break、case、continue、default、do、else、fallthrough、if、in、for、return、switch、where和while。

3.表达式和类型关键字：as、dynamicType、is、new、super、self、Self、Type、\_\_COLUMN\_\_、\_\_FILE\_\_、\_\_FUNCTION\_\_和\_\_LINE\_\_。

4.在特定上下文中使用的关键字：associativity、didSet、get、infix、inout、left、mutating、none、nonmutating、operator、override、postfix、precedence、prefix、rightset、unowned、unowned(safe)、unowned(unsafe)、weak和willSe t。

对于上述关键字，目前我们没有必要全部知道它们的含义，但是要记住：在Swift中，关键字是区分大小写的，因此class和Class是不同的，当然Class不是Swift的关键字。

### 3.1.2 标识符

标识符就是给变量、常量、方法、函数、枚举、结构体、类、协议等指定的名字。构成标识符的字母均有一定的规范，Swift语言中标识符的命名规则如下：

区分大小写，Myname与myname是两个不同的标识符；

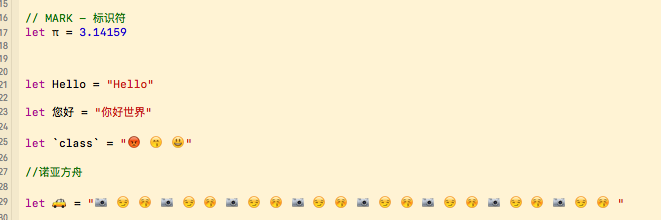
标识符首字符可以以下划线（\_）或者字母开始，但不能是数字；

标识符中其他字符可以是下划线（\_）、字母或数字。

例如，identifier、userName、User\_Name、\_sys\_val、身高等为合法的标识符，而2mail、room#和class为非法的标识符。其中，使用中文“身高”命名的变量是合法的。

注意    Swift中的字母采用的是Unicode编码[[1]](file:///C:/Users/Administrator/Desktop/%E3%80%8ASwift%E5%BC%80%E5%8F%91%E6%8C%87%E5%8D%97%E3%80%8B%E6%A0%B7%E7%AB%A0-%E7%AC%AC%E4%B8%89%E7%AB%A0.doc" \l "_ftn1" \o "" \t "_blank)。Unicode叫做统一编码制，它包含了亚洲文字编码，如中文、日文、韩文等字符，甚至是我们在聊天工具中使用的表情符号，如等，这些符号事实上也是Unicode，而非图片。这些符号在Swift中都可以使用。

如果一定要使用关键字作为标识符，可以在关键字前后添加重音符号（`），例如：

其中class是关键字，事实上重音符号（`）不是标识符的一部分，它也可以用于其他标识符，如π和`π`是等价的。因此使用关键字作为标识符是一种很不好的编程习惯。

## 3.2 常量和变量

### 3.2.1 常量和变量的声明

### 3.2.2 类型推断和类型安全

### 3.2.3 输出常量和变量

## 3.3 简单数据类型

### 3.3.1 整型

### 3.3.2 浮点型

### 3.3.3 布尔类型（Bool）

## 3.4 元组类型

### 3.4.1 元组的声明

### 3.4.2 元组变量的访问

## 3.5 基本运算符

### 3.5.1 赋值运算符

### 3.5.2 单目负运算符

### 3.5.3 算术运算符

### 3.5.4 比较运算符

### 3.5.5 三目运算符

### 3.5.6 逻辑运算符

## 3.6 区间运算符

### 3.6.1 闭区间运算符

### 3.6.2 半闭区间运算符

## 3.7 Optional可选类型

### 3.7.1 可选类型的声明

### 3.7.2 解包（Unwrapping）

### 3.7.3 隐式解析可选类型

## 3.8 本章小结

## 3.9 本章习题

## Swift3.0控制流

## 4.1 条件语句

### 4.1.1 if条件语句

### 4.1.2 if-let语句

### 4.1.3 guard语句

### 4.1.4 switch语句

## 4.2 循环语句

### 4.2.1 for-in循环

### 4.2.2 while循环

### 4.2.3 repeat-while循环

## 4.3 本章小结

## 4.4 本章习题

## Swift3.0字符串和集合

## 5.1 字符

### 5.1.1 字符概述

### 5.1.2 转义字符

## 5.2 字符串

### 5.2.1 初始化字符串

### 5.2.2 字符串的基本操作

### 5.2.3 字符串的高级操作

## 5.3 集合（Collection）

### 5.3.1 创建数组（Array）

### 5.3.2 数组的常见操作

### 5.3.3 Set

### 5.3.4 字典（Dictionary）

## 5.4 本章小结

## 5.5 本章习题

## Swift3.0函数、闭包和枚举

## 6.1 函数

### 6.1.1 函数的定义和调用

### 6.1.2 函数的参数和返回值

### 6.1.3 局部参数名和外部参数名

### 6.1.4 函数参数的其他用法

### 6.1.5 嵌套函数

## 6.2 闭包

### 6.2.1 闭包的概念和定义

### 6.2.2 使用尾随闭包

### 6.2.3 使用闭包表达式

### 6.2.4 捕获

## 6.3 枚举

### 6.3.1 枚举的定义和访问

### 6.3.2 使用Switch语句匹配枚举值

### 6.3.3 原始值

## 6.4 本章小结

## 6.5 本章习题

## Swift3.0面向对象（上）

## 7.1 面向对象概述

## 7.2 类和结构体

### 7.2.1 类和结构体的定义

### 7.2.2 类和结构体的实例

### 7.2.3 类和结构体对比

## 7.3 属性

### 7.3.1 存储属性

### 7.3.2 懒存储属性

### 7.3.3 计算属性

### 7.3.4 属性观察器

### 7.3.5 类型属性

## 7.4 方法

### 7.4.1 实例方法

### 7.4.2 类型方法

## 7.5 构造函数

### 7.5.1 构造函数基础

### 7.5.2 重载构造函数

### 7.5.3 指定构造函数与便利构造函数

### 7.6 析构函数

### 7.7 下标脚本

### 7.7.1 下标脚本语法

### 7.7.2 下标脚本的使用

## 7.8 本章小结

## 7.9 本章习题

## Swift3.0面向对象（下）

## 8.1 面向对象的三大特性

## 8.2 继承和重写

### 8.2.1 继承的概念

### 8.2.2 继承的实现

### 8.2.3 重写

### 8.2.4 final关键字的使用

### 8.2.5 super关键字的使用

## 8.3 构造函数的继承和重写

### 8.3.1 构造函数的调用规则

### 8.3.2 构造过程的安全检查

### 8.3.3 构造函数的自动继承

### 8.3.4 构造函数的重写

## 8.4 封装和多态

### 8.4.1 封装

### 8.4.2 多态

## 8.5 可选链

### 8.5.1 可选链与强制展开

### 8.5.2 可选链访问属性、方法和下标

## 8.6 类型检查和转换

### 8.6.1 类型检查（is操作符）

### 8.6.2 类型转换（as操作符）

### 8.6.3 Any和AnyObject的类型转换

## 8.7 嵌套类型

## 8.8 本章小结

## 8.9 本章习题

## Swift3.0扩展Swift内存管理

## 9.1 扩展

### 9.1.1 扩展概述

### 9.1.2 扩展计算型属性

### 9.1.3 扩展构造函数

### 9.1.4 扩展方法

### 9.1.5 扩展下标

## 9.2 协议

### 9.2.1 协议概述

### 9.2.2 协议的要求

### 9.2.3 协议作为类型使用

### 9.2.4 协议的继承

### 9.2.5 检查协议一致性

### 9.2.6 代理模式

## 9.3 扩展和协议的结合

### 9.3.1 通过扩展采纳协议

### 9.3.2 协议扩展

## 9.4 本章小结

## 9.5 本章习题

## Swift3.0 内存管理

## 10.1 Swift内存管理机制

### 10.1.1 自动引用计数工作机制

### 10.1.2 自动引用计数示例

### 10.1.3 类实例之间的循环强引用

### 10.1.4 解决类实例之间的循环强引用

## 10.2 闭包引起的循环强引用

### 10.2.1 闭包引起的循环强引用

### 10.2.2 解决闭包引起的循环强引用

## 10.3 本章小结

## 10.4 本章习题

## Swift3.0的其他高级特性

## 11.1 泛型

### 11.1.1 泛型函数

### 11.1.2 泛型类型

### 11.1.3 类型约束

### 11.1.4 关联类型

### 11.1.5 where子句

## 11.2 错误处理机制

### 11.2.1 错误的表示

### 11.2.2 错误处理

### 11.2.3 清理操作

## 11.3 访问控制

### 11.3.1 模块、源文件及访问级别

### 11.3.2 类型的访问级别

### 11.3.3 变量常量属性下标及构造函数的访问控制

### 11.3.4 协议扩展的访问控制

## 11.4 命名空间

### 11.4.1 查看和修改命名空间

### 11.4.2 使用命名空间

## 11.5 高级运算符

### 11.5.1 位运算符

### 11.5.2 溢出运算符

### 11.5.3 优先级和结合性

### 11.5.4 运算符函数

### 11.5.5 自定义运算符

## 11.6 本章小结

## 11.7 本章习题

## Swift与Objective-C的相互操作

## 12.1 Swift项目中调用Objective-C类

### 12.1.1 实现原理分析

### 12.1.2 创建Swift项目

### 12.1.3 新建Objective-C类

### 12.1.4 在Swift项目中调用Objective-C代码

## 12.2 Objective-C项目中调用Swift类

### 12.2.1 实现原理分析

### 12.2.2 创建Objective-C项目

### 12.2.3 新建Swift类

### 12.2.4 在Objective-C项目中调用Swift代码

## 12.3 Objective-C项目到Swift项目的迁移

### 12.3.1 准备工作

### 12.3.2 迁移到Swift项目

## 12.4 本章小结

## 12.5 本章习题

## Swift3.0 IOS UI/GUI开发

## Swift3.0项目实战

## 尾语