第5章

Swift函数、闭包与内存管理

5.1 Swift 函数

5.1.1 Swift 函数概述

Swift 函数是具有固定格式，用来解决某个问题或实现一定功能的代码块，在 Swift 中用关键字 func 表示。

通过函数能够实现代码的复用。Swift 函数可通过元组类型返回多个数据。Swift 函数可以通过 inout 关键字使参数变为引用传递。Swift 函数支持可变参数。

5.1.2 Swift 函数的定义

Swift 函数通过 func 关键字进行定义，具体定义格式如下。

(1) 定义无参数无返回值函数的语法格式

```
func 函数名(){
    语句块
}
```

注：Swift 函数不用 void 表示无返回值。

(2) 定义无参数有返回值函数的语法格式

```
func 函数名()->返回值类型{
    语句块
    return 值
}
```

注：Swift 函数通过“—>数据类型”定义返回值的类型；return 后数据的类型要与返回值类型一致。

(3) 定义有参数无返回值函数的语法格式

```
func 函数名(参数名1:数据类型,……){
    语句块
}
```

(4) 定义有参数有返回值函数的语法格式

```
func 函数名(参数名1:数据类型,……)->返回值类型{
    语句块
}
```

5.1.3 Swift 函数的调用


视频讲解

Swift 函数通过“.”运算符进行调用，有返回值的函数需要使用常量或变量接受返回值。

(1) 无返回值函数调用的语法格式

```
函数名([参数标签:参数值, ……])
```

(2) 有返回值函数调用的语法格式

```
常量/变量 = 函数名([参数标签:参数值, ……])
```

例 5.1 Swift 函数的定义。

程序代码：

```
/**
 * 功能:Swift 函数的定义
 * 作者:罗良夫
 */
import Foundation
//无参数无返回值的函数
func showName(){
    print("你好,罗良夫。")
}
showName()
//无参数有返回值的函数
func showDate() -> String{
    let dateFormatter = DateFormatter()
    dateFormatter.dateFormat = "yyyy 年 mm 月 dd 日 HH 时 mm 分 ss 秒"
    return dateFormatter.string(from:Date())
}
var now = showDate()
print(now)
//有参数无返回值的函数
func helloSomeone(name:String){
    print("\(name)你好,欢迎登录系统!")
}
helloSomeone(name:"罗良夫")
//有参数有返回值的函数
func add(num1:Int,num2:Int) -> Int{
    return num1 + num2
}
print("124 + 591 = \(add(num1:124,num2:591))")
```

执行结果：

```
你好,罗良夫。
2022 年 07 月 29 日 13 时 07 分 01 秒
罗良夫你好,欢迎登录系统!
124 + 591 = 715
```

5.1.4 可变参数


视频讲解

函数编写的过程中有时会遇到参数个数不确定的情况，Swift 提供了可变参数来解决这个问题。

可变参数的语法格式：

```
func 函数名(参数名:数据类型…)->返回值类型{
    语句块
    return 语句
}
```

注：参数可以有零到多个；当有多个参数时，可变参数必须为最后一个参数；可变参数中所有参数的数据类型必须相同。

例 5.2 Swift 可变参数的使用。

程序代码：

```
/**
 * 功能:Swift 可变参数的使用
 * 作者:罗良夫
 */
//可变参数的使用
func generalIncome(salary:Double...)->Double{
    var income:Double = 0.0
    for tmp in salary{
        income += tmp
    }
    return income
}
print("总收入为:\(generalIncome())元.")
print("总收入为:\(generalIncome(salary:1000.05,2846.5,3864.0))元.")
//多参数时可变参数的使用
func showLike(name:String,like:String...){
    print("\(name)的爱好有:",terminator:"")
    var index = 1
    for tmp in like{
        if index == like.count{
            print("\(tmp).")
        }else{
            print(tmp,terminator:"")
        }
        index += 1
    }
}
showLike(name:"罗良夫",like:"骑行","羽毛球")
```

执行结果：

```
总收入为:0.0 元.
总收入为:7710.55 元.
罗良夫的爱好有:骑行羽毛球.
```


视频讲解

5.1.5 参数默认值

Swift 可以给函数参数添加默认值，在函数调用过程中未给参数赋值时，自动以默认值填充。

参数默认值的语法格式：

```
func 函数名( 参数标签 参数名:数据类型 = 默认值)
```

例 5.3 Swift 参数默认值的使用。

程序代码：

```
/**
    * 功能:Swift 参数默认值的使用
    * 作者:罗良夫
 */
//参数的默认值
func isLeapFebruary(year:Int = 2022) -> Int{
    if year % 4 == 0 && year % 100 != 0 || year % 400 == 0{
        return 29
    }else{
        return 28
    }
}
var year = 2024
var februryDay = isLeapFebruary(year:year)
print("\(year)\(februryDay == 29 ? "是闰年,2 月份有 29 天." : "不是闰年,2 月份有 28 天.")")
februryDay = isLeapFebruary()
print("2022 年\(februryDay == 29 ? "是闰年,2 月份有 29 天." : "不是闰年,2 月份有 28 天.")")
```

执行结果：

```
2024 是闰年,2 月份有 29 天.
2022 年不是闰年,2 月份有 28 天.
```

5.1.6 参数标签


视频讲解

Swift 函数中每个参数有两个名称，即参数标签与参数名，参数标签用于函数调用过程中；参数名用于函数定义过程中。当只写参数名时，表示参数标签与参数名相同。

参数标签定义的语法格式：

```
func 函数名(参数标签 参数名:数据类型) ->返回值类型{
    使用参数名的语句块
    return 语句
}
函数名(参数标签:值)
```

注：在参数标签定义时可以使用“_”作为占位符，在函数调用时可以省略参数标签的书写。

例 5.4 Swift 参数标签的使用。

程序代码：

```
/**
 * 功能:Swift 参数标签的使用
 * 作者:罗良夫
 */
//参数标签
func showName(user name:String){
    print("欢迎\(name)登录!")
}
showName(user:"罗良夫")
//参数标签与参数名同名
```

```
func isLeapYear(year:Int){
    if year % 4 == 0 && year % 100 != 0 || year % 400 != 0{
        print("\(year)是闰年!")
    }else{
        print("\(year)不是闰年!")
    }
}
isLeapYear(year:2022)
//_占位符的使用
func averageAge(_ age:Int...) -> Double{
    var sum:Int = 0
    for tmp in age{
        sum += tmp
    }
    return Double(sum)/Double(age.count)
}
print("学生的平均年龄:\(averageAge(18,23,19,20,22))岁.")
```

执行结果:

```
欢迎罗良夫登录!
2022 是闰年!
学生的平均年龄:20.4 岁.
```


视频讲解

5.1.7 输入输出参数

Swift 函数参数采用的是值传递方式,当需要在函数调用结束后保留参数值,则需要使用输入输出类型的参数,对应的关键字是 inout,语法格式如下。

输入输出参数使用的语法格式:

```
func 函数名(参数名:inout 数据类型) ->返回值类型{
    语句块
    return 语句
}
函数名( & 参数名 )
```

例 5.5 Swift 输入输出参数的使用。

程序代码:

```
/**
 * 功能:Swift 输入输出参数的使用
 * 作者:罗良夫
 */
//输入输出参数的使用
func swapValue(num1:inout Int , num2:inout Int){
    var tmp:Int
    tmp = num1
    num1 = num2
    num2 = tmp
}
var num1:Int = 100 , num2 = 150
print("交换前:num1 = \(num1),num2 = \(num2)")
swapValue(num1:&num1,num2:&num2)
print("交换后:num1 = \(num1),num2 = \(num2)")
```

执行结果：

```
交换前:num1 = 100,num2 = 150
交换后:num1 = 150,num2 = 100
```

5.1.8 函数类型


视频讲解

Swift 函数可以作为数据类型定义常量或变量，可以作为函数参数及返回值类型。

(1) 函数作为数据类型的语法格式

```
let/var 常量/变量名:(参数类型 1, ……) ->返回值类型
```

(2) 函数作为参数类型的语法格式

```
func 函数名(参数名:(参数类型 1, ……) ->返回值类型){
    语句块
}
```

注：函数作为参数类型时，需要写返回值类型，无返回值时写“—> Void”。

(3) 函数作为返回值类型的语法格式

```
func 函数名(参数名:数据类型) ->(参数类型 1, …… ) ->返回值类型{
    语句块
}
```

注：函数作为返回值类型时，需要使用小括号将其包含起来。

例 5.6 函数类型示例。

程序代码：

```
/**
 * 功能:函数类型示例
 * 作者:罗良夫
 */
//函数类型
func add(num1:Int , num2:Int) -> Int{
    return num1 + num2
}
func div(num1:Int , num2:Int) -> Int{
    return num1 - num2
}
var cal:(Int,Int) -> Int = add
let res1 = cal(730,35012)
print("res1 = \(res1)")
cal = div
let res2 = cal(730,35012)
print("res2 = \(res2)")
//函数作参数类型
func showMsg(user:String , message:String) -> Void{
    print("\(user)say: \(message).")
}
func caller(funcName:(String,String) -> Void,v1:String,v2:String){
   funcName(v1,v2)
}
caller(funcName:showMsg,v1:"罗良夫",v2:"iOS 综合应用开发")
//函数类型作为返回值类型
```

```
func selfInspection(){
    print("机器自检……")
}
func startUp() -> (() -> Void){
    print("机器开机……")
    return selfInspection
}
var si = startUp()
si()
```

执行结果：

```
res1 = 35742
res2 = -34282
罗良夫 say: iOS 综合应用开发.
机器开机……
机器自检……
```


视频讲解

5.1.9 函数嵌套

Swift 允许在函数中定义内部函数，外部函数可以调用内部函数。

函数嵌套的语法格式：

```
func 函数名(参数名 1;参数类型, ……) ->返回值类型{
func 函数名(参数名 1:参数类型, ……) ->返回值类型{
    语句块
    return 语句
}
语句块
return 语句
}
```

注：内部函数可以使用外部函数的参数。

例 5.7 函数嵌套示例。

程序代码：

```
/**
 * 功能:函数嵌套示例
 * 作者:罗良夫
 */
//函数嵌套
func average(values:Double...) -> Double{
    func sum() -> Double{
         var sum:Double = 0
        for n in values{
         sum += n
        }
        return sum
    }
    return sum()/Double(values.count)
}
print("result = \(average(values:1.3,3.14,6.43))")
```

执行结果：

```
result = 3.6233333333333335
```

5.1.10 多返回值函数

每个函数只能返回一个值，Swift 可以通过元组类型间接实现返回多个值。

多返回值函数的语法格式：

```
函数名(参数名 1:参数类型, ……) ->(元组类型){
    语句块
    return 语句
}
```

例 5.8 多返回值函数示例。

程序代码：

```
/**
 * 功能:多返回值函数示例
 * 作者:罗良夫
 */
//多返回值函数
func teacherInfo(name:String,age:Int,research:String) ->(String,Int,String){
    return (name,age,research)
}
var teacher1 = teacherInfo(name:"罗良夫",age:37,research:"AI")
print("教师 1 信息:\(teacher1)")
```

执行结果：

```
教师 1 信息:("罗良夫", 37, "AI")
```

5.2 Swift 闭包

5.2.1 Swift 闭包概述

Swift 可以通过字符串和整数形式使用函数，函数的这种用法称为闭包。Swift 闭包是自包含代码块，可以在代码中被传递和使用。闭包可以捕获与存储其所在上下文中任意常量和变量的引用，也就是所谓“闭合并包裹”着这些常量和变量。

Swift 闭包的特点如下：

- Swift 闭包可以通过上下文推断其参数和返回值类型；
- 隐式返回单表达式闭包时，可以省略 return 关键字；
- 参数名称可以缩写；
- 尾随(Trailing)闭包语法。

闭包的三种形式如下。

(1) 全局函数：有名字但不会捕获任何值的闭包。

(2) 嵌套函数：有名字并可以捕获其封闭函数域内值的闭包。

(3) 闭包表达式：没有名字但可以捕获上下文中变量和常量值的闭包。

5.2.2 Swift 闭包表达式


视频讲解

闭包表达式的语法格式：

```
{(参数名:参数类型……) ->返回值类型 in
    语句块
}
```

注：闭包表达式的参数可以为常量和变量，也可以使用 inout 类型，但不能提供默认值；在参数列表的最后可以使用可变参数；可以使用元组作为参数和返回值。

例 5.9 闭包表达式示例。

程序代码：

```
/**
 * 功能:闭包表达式示例
 * 作者:罗良夫
 */
var cityName:[String] = ["wuhan","beijing","shanghai","guangzhou"]
var citySorted = cityName.sorted(by:{(s1:String,s2:String) -> Bool in return s1 < s2})
print("升序排列:\(citySorted)")
```

执行结果：

```
升序排列:["beijing", "guangzhou", "shanghai", "wuhan"]
```


视频讲解

5.2.3 Swift 闭包的简写形式

Swift 闭包简写形式 1：

```
{参数名 1, …… in return 返回值表达式}
```

注：Swift 闭包可以省略参数类型和返回值类型，Swift 通过上下文可以推断出参数类型。

Swift 闭包简写形式 2：

```
{参数名 1, …… in 返回值表达式}
```

注：Swift 闭包中只有一条语句时，可以省略 return 关键字。

Swift 闭包简写形式 3：

```
{ $n 语句}
```

注：$n 是参数的简写形式，n 表示实际参数序号，$0 代表第一个参数；使用 $简写参数时，可以省略参数的定义，Swift 会自动推断参数的数据类型；in 关键字可以省略，闭包表达式由函数体组成。

Swift 闭包简写形式 4：

```
{运算符}
```

注：Swift 闭包只有一条语句时，可以省略参数名。

尾随闭包的语法格式：

```
函数名(……){闭包表达式}
```

注：尾随闭包是将闭包表达式写在函数参数列表之后，函数支持将其作为最后一个参数调用。

例 5.10 Swift 闭包的简写形式。

程序代码：

```
/**
 * 功能:Swift 闭包的简写
 * 作者:罗良夫
 */
func isLeapYear(year:Int , mth:(Int) -> Bool){
    if mth(year) == true{
        print("\(year)是闰年!")
    }else{
        print("\(year)不是闰年!")
    }
}
//闭包简写形式 1
var y = 2022
isLeapYear(year:y , mth:{(p1) in  return (y % 4 == 0 && y % 100 != 0 || y % 400 == 0) ? true :
false})
//闭包简写形式 2
y = 2024
isLeapYear(year:y , mth:{(p1) in  (y % 4 == 0 && y % 100 != 0 || y % 400 == 0) ? true : false})
//闭包简写形式 3
func compareBigOrSmall(n1:Int , n2:Int, fun:(Int,Int) -> Int) -> Int{
    return fun(n1,n2)
}
var v1 = 192 , v2 = 841
var max = compareBigOrSmall(n1:v1,n2:v2,fun:{ $0 > $1 ? $0 : $1})
print("\(v1)与\(v2)中较大的数是\(max).")
//闭包简写形式 4
func subtraction(n1:Int,n2:Int,fun:(Int,Int) -> Int) -> Int{
    return fun(n1,n2)
}
var n1 = -930 , n2 = 1293
var res = subtraction(n1:n1,n2:n2,fun:-)
print("\(n1) - \(n2) = \(res)")
//尾随闭包
y = 2028
isLeapYear(year:y){(p1) in  (y % 4 == 0 && y % 100 != 0 || y % 400 == 0) ? true : false}
```

执行结果:

```
2022 不是闰年!
2024 是闰年!
192 与 841 中较大的数是 841.
-930 - 1293 = -2223
2028 是闰年!
```

5.3 Swift 内存管理

5.3.1 Swift 内存管理概述

Swift 使用自动引用计数(Auto Reference Count,ARC)机制来解决应用程序的内存管理问题。一般来说,应用不需要进行内存管理,系统会自动管理类的实例,并在适当的时候释放其占用的内存。

当类创建实例化对象时,系统会分配指定大小的内存空间存储实例的相关数据,当该实

例不再被使用时,ARC 会自动释放实例所占用的内存空间。


视频讲解

5.3.2 强引用

实例对象在使用中时,ARC 将通过跟踪和计算实例的所有引用数(即该实例被多少属性、常量与变量引用)来确保该实例的内存空间不会被释放,引用数大于 0 时,实例对象所占内存不会被释放。

属性、常量与变量对实例对象的引用为强引用。Swift 中可以通过 CFGetRetainCount 获取实例对象的引用计数。

当两个类同时使用对方作为属性时,会产生两个类之间的循环引用,导致两个类的实例对象无法被 ARC 销毁,造成内存泄漏。

例 5.11 Swift 强引用示例。

程序代码:

```
/**
 * 功能:Swift 强引用示例
 * 作者:罗良夫
 */
import UIKit

class Staff{
    var name:String
    var age:Int
    var manager:Manager?
    init(name:String,age:Int){
        print("Staff instance is created.")
        self.name = name
        self.age = age
    }
    deinit{
        print("Staff instance is destroyed.")
    }
}
class Manager{
    var name:String
    var age:Int
    var staff:Staff?
    init(name:String,age:Int){
        print("Manager instance is created.")
        self.name = name
        self.age = age
    }
    deinit{
        print("Manager instance is destroyed.")
    }
}
var sta:Staff? = Staff(name: "one", age: 19)
var man:Manager? = Manager(name: "two", age: 20)
sta?.manager = man
man?.staff = sta
print("sta:\(CFGetRetainCount(sta))")
```

```
sta = nil
man = nil
```

执行结果：

```
Staff instance is created.
Manager instance is created.
sta:3
```

5.3.3 弱引用


视频讲解

Swift 可以通过弱引用来解决强引用产生的循环引用问题，弱引用不会对其引用的实例保持强引用，因而不会阻止 ARC 销毁被引用的实例，ARC 会在引用的实例被销毁后自动将其弱引用赋值为 nil，所以对实例对象的引用需要设置成可选类型，Swift 中用关键字 weak 表示弱引用类型。

例 5.12 弱引用示例。

程序代码：

```
/**
 * 功能:Swift 弱引用示例
 * 作者:罗良夫
 */
import UIKit

class Staff{
    var name:String
    var age:Int
    var manager:Manager?
    init(name:String,age:Int){
        print("Staff instance is created.")
        self.name = name
        self.age = age
    }
    deinit{
        print("Staff instance is destroyed.")
    }
}
class Manager{
    var name:String
    var age:Int
    //弱引用:当 Staff 引用为 nil 时,staff 的值自动为 nil
    weak var staff:Staff?
    init(name:String,age:Int){
        print("Manager instance is created.")
        self.name = name
        self.age = age
    }
    deinit{
        print("Manager instance is destroyed.")
    }
}
var sta:Staff? = Staff(name: "one", age: 19)
var man:Manager? = Manager(name: "two", age: 20)
```

```
sta?.manager = man
man?.staff = sta
print("sta:\(CFGetRetainCount(sta))")
sta = nil
```

执行结果：

```
Staff instance is created.
Manager instance is created.
sta:2
Staff instance is destroyed.
```


视频讲解

5.3.4 无主引用

Swift 无主引用允许循环引用中的一个实例引用另一个实例而不保持强引用，Swift 中用关键字 unowned 表示无主引用。无主引用不会在被引用实例销毁后自动赋值为 nil。无主引用必须确保引用始终指向一个未销毁的实例，如果在赋给无主引用的实例被销毁后访问无主引用，会触发运行时错误。

例 5.13 无主引用示例。

程序代码：

```
/**
 * 功能:Swift 无主引用示例
 * 作者:罗良夫
 */
import UIKit

class Staff{
    var name:String
    var age:Int
    var manager:Manager?
    init(name:String,age:Int){
        print("Staff instance is created.")
        self.name = name
        self.age = age
    }
    deinit{
        print("Staff instance is destroyed.")
    }
}
class Manager{
    var name:String
    var age:Int
    //无主引用
    unowned var staff:Staff
    init(name:String,age:Int,staff:Staff){
        print("Manager instance is created.")
        self.name = name
        self.age = age
        self.staff = staff
    }
    deinit{
        print("Manager instance is destroyed.")
```

```
    }
}
var sta:Staff? = Staff(name: "one", age: 19)
var man:Manager? = Manager(name: "two", age: 20,staff: sta!)
sta?.manager = man
man?.staff = sta!
print("sta:\(CFGetRetainCount(sta))")
sta = nil
```

执行结果：

```
Staff instance is created.
Manager instance is created.
sta:2
Staff instance is destroyed.
```

5.4 小结

Swift 函数是具有固定格式，用来解决某个问题或实现一定功能的代码块，在 Swift 中用关键字 func 表示。Swift 函数支持可变参数的使用，适合于参数数量可变的情况；Swift 允许给函数的参数指定默认值；Swift 函数参数有两个名称，即参数标签与参数名；Swift 函数具有输入输出参数功能，允许使用引用方式调用参数。

Swift 闭包是自包含代码块，可以在代码中被传递和使用。Swift 闭包可以通过上下文推断其参数和返回值类型。Swift 闭包具有全局函数、嵌套函数、闭包表达式三种形式。

习题

一、单选题

1. Swift 中函数通过关键字(　　)进行定义。

A. meth　　B. param　　C. class　　D. func

2. Swift 可变参数对应的关键字是(　　)。

A. ...　　B. ->　　C. # #　　D. //

3. Swift 的内存管理通过(　　)机制来实现。

A. 垃圾回收　　B. 循环引用　　C. ARC　　D. 链式指针

4. Swift 闭包的形式不包括(　　)。

A. 嵌套函数　　B. 全局函数　　C. 闭包表达式　　D. 复合函数

二、填空题

1. Swift 闭包以_____和_____形式使用函数。
2. Swift 函数的每个参数有两个名称，即_____与参数名。
3. 属性、常量与变量对实例对象的引用为_____引用。
4. 输入输出类型的参数的关键字是_____。
5. Swift 闭包中隐式返回单表达式闭包时，可以省略_____关键字。

实训　函数与闭包

1. 函数

```
//加法函数
func add(v1:Int,2:Int) -> Int {
    let result = v1 + v2
    return result
}
var result = 0
result = add(v1: 2, v2: 3)
print("2 + 3 = \(result) ")
```

2. 闭包

```
let clo1:(String, String) -> String = { (str1: String, str2: String) -> String in
    return str1 + str2
}
print(clo1("simple", " closure."))
```