# 第五章 函数与闭包

## 5.1函数

函数是人们为了减少代码编写量和时间，以及使代码结构鲜明、清晰易懂而提出的概念。使用函数可以把有特定功能的代码封装起来，然后在多处使用，这样方便快捷且代码简洁。本节将介绍使用swift语言如何来编写及调用函数。

### 5.1.1函数的声明和使用

函数的基本语法形式如下：

func 函数名（参数列表...）->返回值类型...{

    语句...

} //其中...表示其数量可以为任意非负整数

函数名是一个标识符，一般表示此函数的功能；参数列表由参数名和数据类型组成，使用参数是为了接受调用函数时传递的参数；’->’后面跟着的是函数的返回值类型，函数可以有返回值，也可以没有返回值。函数如果以参数的角度分类可分为无参函数和有参函数；如果以返回值的角度来分类的话可分为无返回值的函数和有返回值的函数。下面会依次讨论不同种类函数的声明和使用。

（1）无参函数

无参函数就是没有参数列表的函数。无参函数声明定义的一般形式如下：

|  |
| --- |
| func 函数名（）->返回值类型{      语句  } |

函数调用的形式如下：

|  |
| --- |
| 函数名（） |

【例 5-1】以下定义一个无参函数printHelloSwift（）并调用它，此函数用来输出字符串“ Hello Swift!”,代码如下：

|  |
| --- |
| func printHelloSwift(){              //声明定义函数  print(“Hello Swift!”)  }  printHelloSwift()            //调用函数 |

运行结果为：

|  |
| --- |
| Hello Swift! |

（2） 有参函数

所谓有参函数，就是函数列表不为空的函数，参数数量可以是一个或多个。有参函数声明定义的一般形式如下：

|  |
| --- |
| func 函数名（参数名1：数据类型，参数名2：数据类型…）->返回值类{      语句  } |

其中参数名和数据类型用‘：’号来隔开，不同的参数用‘，’号来隔开。其调用的形式如下：

函数名（参数名1：参数值1，参数名2：参数值2…）

其中参数值和参数名的类型一定要一致。

【例 5-2】 以下定义一个sum函数来实现2个数的相加，代码如下：

|  |
| --- |
| func sum(value1:Int,value2:Int){      print(value1+value2)  }  sum(value1:20,value2:10) |

运行结果为：

|  |
| --- |
| 30 |

（3）无返回值的函数

函数是可以没有返回值的，其声明定义的形式如下：

|  |
| --- |
| func 函数名（参数列表...）{      语句...  } |

函数调用的形式如下：

|  |
| --- |
| 函数名（参数列表） |

（4） 具有一个返回值的函数

在函数中返回一个值是最常见的。函数返回值的类型必须在函数声明定义时为函数设定，并用return语句进行返回。这种函数的声明定义形式如下：

|  |
| --- |
| func 函数名（参数表）->返回值类型{  语句  return 表达式  } |

返回的表达式类型必须与函数的返回值类型一致。

【例 5-3】 以下定义一个sum1函数来实现2个数的相加，代码如下：

|  |
| --- |
| func sum1(value1:Int,value2:Int)->Int{      return value1+value2  }  print(sum1(value1:20,value2:10))  //直接调用  let d=sum1(value1:20,value2:20)   //赋值  print(d) |

运行结果为：

|  |
| --- |
| 30  40 |

（5）具有多个返回值的函数

函数返回多个返回值，就需要使用到元组类型。这种函数的声明定义形式如下：

|  |
| --- |
| func 函数名（参数列表）->(数据类型1，数据类型2，数据类型3...){  语句  return （表达式1，表达式2，表达式3...）  } |

数据类型之间用‘，’号来隔开，return中的表达式的类型要与返回值的数据类型一一对应。

【例 5-4】 以下定义一个sum2函数来实现2个数和2个字符串的的相加，代码如下：

|  |
| --- |
| func sum2(value1:Int,value2:Int,str1:String,str2:String) -> (Int,String){  //2个数和2个字符串分别相加      let k=value1+value2      let l=str1+str2      return (k,l)  }  print(sum2(value1: 5, value2: 1, str1: "hello", str2: "swift")) |

运行结果为：

|  |
| --- |
| (6, "helloswift") |

### 5.1.2 关于函数参数的使用

如果开发者使用的函数为有参函数时，对于参数需要了解以下几点。

（1）关于函数参数名

有时参数的参数名会有两个，这两个函数参数名分为外部参数名和本地参数名。本地参数名用于声明函数时在函数内部使用，外部参数名用于调用该参数时使用。外部参数名是为了让函数中的参数明确和便于理解。定义函数时，外部参数名需写在本地参数名之前，并用空格来隔开。声明定义的一般形式如下：

|  |
| --- |
| func 函数名（外部参数名 本地参数名：数据类型）->返回值类型{  ……  } |

调用的一般形式如下：

|  |
| --- |
| 函数名（外部参数名：参数值） |

（2）忽略参数名

如果开发者想在调用函数时忽略参数名，可以使用下划线 ’\_’ 代替外部参数名来实现。它的一般形式如下：

|  |
| --- |
| func 函数名（\_ 参数名1：数据类型，\_ 参数名2：数据类型…）->返回值类型{  语句  } |

【例 5-5】 以下定义一个sum3函数来实现2个数的相加，代码如下：

|  |
| --- |
| func sum3(\_ value1:Int,\_ value2:Int)->Int{      return value1+value2  }  print(sum3(10,20)) //省略参数名 |

运行结果为：

|  |
| --- |
| 30 |

### 5.1.3 为参数设置默认值

开发者可以在函数体中为每个参数定义默认值。当默认值定义后，调用这个函数时可以忽略这个参数，系统会把默认值传给此参数。

【例 5-6】 以下定义一个sum4函数来实现求和运算，代码如下：

|  |
| --- |
| func sum4(value1:Int=20,value2:Int)->Int{ //参数有默认值      return value1+value2  }  print(sum4(value2:20)) |

运行结果为：

|  |
| --- |
| 40 |

### 5.1.4 可变参数

可变参数可以接受零个或多个指定类型的值。函数调用时，可以用可变参数来指定函数参数，其数量是不确定的。

可变参数通过在数据类型名后面加入 ‘...’ 的方式来定义。

【例 5-7】 以下定义一个sum5函数来实现求和运算，代码如下：

|  |
| --- |
| func sum5(numbers:Double...){       //可变参数      var total:Double=0.0      //求和      for number in numbers{          total+=number      }      print(total)  }  sum5(numbers: 1.1,2.22,4) |

运行结果为：

|  |
| --- |
| 7.32 |

5.1.5输入输出参数

一般默认在函数中定义的参数都是常量参数，也就是这个参数只可以查询使用，不能改变它的值。 如果想要改变参数的值，且参数在函数调用后其改变仍有效，可以在参数名后加 ‘inout’ 关键字，这样就可以改变这个参数的值了。它的一般形式如下：

|  |
| --- |
| func 函数名（参数名：inout 数据类型，...）->返回值类型{      语句  } |

注意：使用inout关键字的参数必须是已经声明过的变量。函数的调用形式如下：

|  |
| --- |
| 函数名（&参数，...） |

【例 5-8】 以下定义一个sum6函数来实现求和运算，代码如下：

|  |
| --- |
| func sum6(value1:inout Int,\_ value2:Int){  //输入输出参数      value1 = value2+value1  }  var s = 20  print("没加任何值的s为\(s)")  sum6(value1: &s, 20)  print("加了值后的s为\(s)") |

运行结果为：

|  |
| --- |
| 没加任何值的s为20  加了值后的s为40 |

### 5.1.6函数类型

在swift中，每一个函数都有一个特定的类型，这种类型称为函数类型，函数类型由函数的参数类型和返回类型组成。例如【例 5-5】中的函数：

|  |
| --- |
| func sum3(\_ value1:Int,\_ value2:Int)->Int{      return value1+value2  } |

此函数的类型就是（int,int）->int。对于没有返回值的无参函数来说，它的类型为()->()。

Swift语言中可以使用函数类型来定义其他常量/变量，就像使用其他类型一样。函数类型使用的一般形式如下：

|  |
| --- |
| let/var 常量名/变量名：函数类型=函数名 |

或者：

|  |
| --- |
| let/var 常量名/变量名=函数名 |

【例 5-9】 以下把sum7函数的类型赋值给变量来实现求和运算，代码如下：

|  |
| --- |
| func sum7(value1:Int,value2:Int)->Int{      return value1+value2  }  let add:(Int,Int)->Int=sum7  print(add(20, 30))  print(sum7(value1: 40, value2:20)) |

运行结果为：

|  |
| --- |
| 50  60 |

函数类型可以作为函数参数来使用。

【例 5-10】 以下把上例sum7函数的类型作为函数参数来使用，代码如下：

|  |
| --- |
| func sum8(value1:Int,value2:Int)->Int{      return value1+value2  }  func adder(fun:(Int,Int)->Int,a:Int,b:Int)->Int{      let x=fun(a,b)      return x  }  print(adder(fun: sum8, a: 20, b: 60)) |

运行结果为：

|  |
| --- |
| 80 |

其中adder函数中的fun只是一个参数名，其类型为(Int,Int)->Int，所有这个类型的函数都能成为adder函数的参数。从这个例子可以看出，函数类型作为函数参数与数据类型作为函数参数是一样的。

函数类型还可以作为返回类型来使用。

【例 5-11】 以下把函数类型作为返回值并使用，代码如下：

|  |
| --- |
| func mult(a:Int,b:Int)->Int{      return a\*b  }  func sum9(value1:Int,value2:Int)->Int{      return value1+value2  }  func numchange(c:Int,d:Int)->(Int,Int)->Int{      return c>d ? sum9:mult  }  let num=numchange(c: 40, d: 50)  print(num(5,5)) |

运行结果为：

|  |
| --- |
| 25 |

### 5.1.7函数的嵌套

函数的嵌套分为嵌套函数和函数的嵌套调用2种。嵌套函数指的是函数内定义的一个新的函数，定义它的函数可以调用该函数。嵌套调用的形式有在一个函数中调用其他函数；或者调用自身函数。调用自身的嵌套调用我们称之为递归调用。

1.嵌套函数

默认情况下，嵌套函数只能在父函数中调用，在父函数外部无法被调用。定义嵌套函数的父函数也可以返回它的某一个嵌套函数，使得这个函数可以在其他域中被使用。

【例 5-12】 以下在函数中声明定义嵌套函数并调用，代码如下：

|  |
| --- |
| func isDivisible (\_ a:Int,\_ b:Int)->()->String{  func divisibled()->String{return "\(b)可以被\(a)整除。"}  func undivisibled()->String{return "\(b)不可以被\(a)整除。"}      return b%a==0 ? divisibled:undivisibled  }  let x=isDivisible(3, 46) //x是一个函数  let result=x() //调用x函数  print(result) |

运行结果为：

|  |
| --- |
| 46不可以被3整除。 |

2.嵌套调用

嵌套调用是一个函数调用一个或多个函数（不是在自己内部定义的）。其基本形式如下：

|  |
| --- |
| func 函数名0（参数列表）->返回值数据类型{  ......  }  func 函数名1（参数列表）->返回值数据类型{  ......  调用函数0  ......  }  func 函数名2（参数列表）->返回值数据类型{  ......  调用函数1  ......  } |

递归调用属于嵌套调用的一种特殊方式。它在调用函数时调用了该函数本身。其形式如下：

|  |
| --- |
| func 函数名1（参数列表）->返回值数据类型{  ......  调用函数1  ......  } |

递归调用是嵌套调用的一种特殊情况。

【例 5-13】 以下通过递归调用来计算5！的值：

|  |
| --- |
| func fac(value:Int)->Int{       //递归调用      if(value==1){          return value      }else{          return value\*fac(value: value-1)      }  }  print(fac(value: 5)) |

运行结果为：

|  |
| --- |
| 120 |

## 5.2 闭包

闭包可以看做无参数名的函数，但它不是函数。函数实际上是特殊的闭包。闭包能够像函数一样根据语句实现某种特定功能，也可以捕获和存储上下文中变量的引用。

本章主要讲解swift语言中闭包的声明定义和使用、尾随闭包和捕获值等的相关知识。

### 5.2.1闭包表达式

闭包表达式的基本语法形式如下：

|  |
| --- |
| {(参数列表) -> 返回值类型 in  语句  } |

定义的语法形式如下：

|  |
| --- |
| let/var 闭包表达式常量名称/闭包表达式常量名称=具有参数的闭包表达式 |

调用形式如下:

|  |
| --- |
| 闭包表达式常量名称/闭包表达式变量名称（参数值1，参数值2...） |

其中参数列表和函数的一样可以是常量、变量、输入输出函数，但是不能有默认值。参数列表和返回值类型和函数一样可有可无，多个参数和返回值类型的声明定义的形式也和函数一样。关键字 ‘in’ 后面跟着的是闭包表达式的语句，相当于函数的语句部分。闭包表达式声明定义的同时一定要赋值给一个常量或变量，这就相当于给了一个函数名。其调用形式就如函数一样了。

【例 5-14】 以下实现两个整数的相除和输出字符串“Swift”，代码如下：

|  |
| --- |
| //有参有返回值闭包  let divide = {(val1: Int, val2: Int) -> Int in      return val1 / val2  }  let res = divide(200, 20)  print (res)  //无参无返回值闭包  let printSwift={() in      print("Swift")  }  printSwift() |

运行结果为：

|  |
| --- |
| 10  Swift |

闭包表达式可以用于其他函数中，可以在函数中先声明其类型，在调用该函数时再定义闭包。

【例 5-15】 以下将闭包表达式作为函数的一部分，实现判断在数组中是否存在某个值的公倍数。代码如下：

|  |
| --- |
| func hasMultiple(arr:[Int],\_ value:Int,\_ cm:(\_ num:Int,\_ value:Int) ->Bool)->Bool{      //闭包作为参数      for item in arr{          if(cm(item,value)){              return true              }          }      return false  }  let ray=[20,80,60,100]  var v=hasMultiple(arr: ray,16,{(num,value)->Bool in return num%value==0  })  print(v) |

运行结果为：

|  |
| --- |
| true |

### 5.2.2闭包表达式的优化

以下都以【例5-13】为例：

**(1)推断类型和**省略return

hasMultiple()函数的第三个参数是闭包表达式，它的类型为 ’(Int,Int) ->Bool’。闭包可以根据上下文推断其参数和返回值的类型，所以->和数值类型以及返回值类型都可以省略；如果闭包表达式的语句的代码只有一行，那么也可以省略return，如以下代码所示：

|  |
| --- |
| var v=hasMultiple(arr: ray,16,{(num,value)  in num%value ==0}) |

（2）简写或省略参数名

开发者可以通过$0,$1,$2来顺序的调用闭包的参数。

|  |
| --- |
| var v=hasMultiple(arr: ray,16,{$0%$1==0}) |

闭包的函数体很短时可以写成一行，所以写成一行的写法并没有错。

### 5.2.3尾随闭包

当闭包作为一个函数的参数，且是最后一个参数时，可以使用尾随闭包。尾随闭包（又称Trailing闭包）是一个书写在函数括号之后的闭包表达式，这是因为有的闭包表达式很长，用尾随闭包可以增强函数的可读性。以【例 5-13】为例，其函数的调用可写成：

|  |
| --- |
| var v=hasMultiple(arr: ray, 16){$0%$1==0} |

当函数只有闭包这一个参数时，可以把括号也省略掉。

【例 5-16】用sorted函数配合闭包完成排序。

|  |
| --- |
| let number=[1,55,26,8,46,20]  var reverse = number.sorted() { $0 > $1 }  var reverse1 = number.sorted { $0 > $1 }  print(reverse)  print(reverse1) |

运行结果为：

|  |
| --- |
| [55, 46, 26, 20, 8, 1]  [55, 46, 26, 20, 8, 1] |

### 5.2.4 捕获值

闭包可以捕获其被定义的上下文的常量或变量的值，实际上相当于使用闭包把这些常量和变量存起来。在常量或变量的原作用域不存在的情况下，仍然能够在闭包函数体内修改这些值。闭包捕获值最简单的应用形式就是嵌套函数。嵌套函数可以捕获其父函数定义的所有参数、常量和变量。

【例 5-17】以下定义一个嵌套函数，该函数将会捕获其父函数的参数和变量，代码如下：

|  |
| --- |
| func numberOfMultiply(multiplier num:Int) -> () -> Int {      var result = 3      func multiply() -> Int {          result \*= num          return result      }      return multiply  }  let mul = numberOfMultiply(multiplier: 2)  print(mul())  print(mul()) |

运行结果为：

|  |
| --- |
| 6  12 |

## 5.3 小结

本章主要介绍了函数和闭包的有关内容，重点介绍了函数的生声明和使用以及闭包的生命和使用。本章需要重点掌握函数的声明和调用方式，闭包表达式的声明和调用，熟练应用函数与闭包的各种功能。