5.1 函数的介绍

函数是用来完成特定任务的独立的代码块。你给一个函数起一个合适的名字，用来标识函数做什么，并且当函数需要执行的时候，这个名字会被“调用”。

Swift 统一的函数语法足够灵活，可以用来表示任何函数。参数可以提供默认值，以简化函数调用。参数也可以既当做传入参数，也当做传出参数，也就是说，一旦函数执行结束，传入的参数值可以被修改。在 Swift 中，每个函数都有一种类型，包括函数的参数值类型和返回值类型。你可以把函数类型当做任何其他普通变量类型一样处理，这样就可以更简单地把函数当做别的函数的参数，也可以从其他函数中返回函数。函数的定义可以写在在其他函数定义中，这样可以在嵌套函数范围内实现功能封装。

一个完整的函数由func关键字、函数名、参数表以及函数的返回值类型组成。它的语法形式如下：

Func 函数名（参数表）->返回值类型{

语句}

}

5.2 无参函数的使用

无参函数实质没有参数列表的函数。无参函数的声明的一般形式如下：

Func 函数名（）->返回值类型{

语句

}

其中，语句可以是一条或多条语句；函数名同样是一个标识符，用来表示函数要实现的功能：->返回值类型表示函数可能返回的值的类型。返回值可以省略，这时表示函数无返回值。这相当于C语言中函数的返回值为void。函数的语句也可以没有，函数定义没有语句，就是空语句。函数定义好后，就可以使用该函数，称为函数的调用。函数调用的形式如下：

函数名（）

以下定义一个无参函数printSwift（），此函数用来输出字符串“Swift”,代码如下：

Func printSwift(){ //声明定义函数

Print(“Swift”)

}

PrintSwift() //调用函数

此时运行程序，会看到如下结果。

Swift

所谓有参函，也就是函数头的参数表不为空。其声明定义的形式如下：

Func 函数名（参数名1：数据类型，参数名2：数据类型，参数名3：数据类型…）->返回值类型{

语句

}

参数列表由函数名和数据类型组成，其中使用冒号“：”隔开。参数列表的参数可以是一个，可以是多个，其中用“，”号隔开。定义时使用参数是为了接收调用时传递的数据。其调用形式如下：

函数名（参数名1：参数值，参数名2：参数值，参数名3：参数值…）

示例 以下将实现向某人打招呼的功能。代码如下。

Func hello(name:Sting){ //声明函数

Print(“hello,\(name)”)

}

let name1=”Tom”

lello(name:name1) //调用函数

let name2=”Jane”

hello(name:name2) //调用函数

运行程序，输出结果为

Hello，Tom

Hello,Jane

5.3 函数参数的注意事项

在使用有参函数时，要注意以下一些事项

1. 函数参数名

函数参数名分为外部参数名和本地参数名。外部参数名用来标记传递给函数调用的参数，本地参数名在实现函数时使用。外部参数名是为了让函数中的参数明确和便于理解。外部参数名需写在本地参数名之前，并用空格来隔开。其一般形式如下：

Func 函数名（外部参数名 内部参数名：数据类型）->返回值类型{

……

}

对于有外部参数名的函数，调用形式如下。

函数名（外部参数名：参数值）

对于无外部参数名的函数，调用方式如5.3中的有参函数。

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，\_ 参数名3：数据类型…）->返回值类型{

语句

}

示例 以下将实现求和运算。代码如下

func sum(\_ value1:Int,\_ value2:Int)->{

return value1+value2

}

print(sum(10,20))

运行该程序，会看到如下结果

30

3. 为参数设置默认值

开发者可以在函数体中为每个函数定义默认值。当默认值定义后，调用这个函数时可以忽略这个函数。

示例 以下将实现求和运算。代码如下

func sum1(value1:Int=20,value2:Int)->Int{

return value1+value2

}

print(sum1(20))

运行该程序，会看到如下结果

40

4. 可变参数

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

可变参数通过在变量类型名后面加入（...）的方式来定义。

示例 以下将实现求和运算。代码如下

func sum2(numbers:Double…){ //可变参数

var total:Double=0.0

//求和

for number in numbers{

total+=number

}

Print(total)

}

sum2(1.1,2.22,4)

运行该程序，会看到如下结果

7.32

5.常量，变量及 I/O 参数

一般默认在函数中定义的参数都是常量参数，也就是这个参数你只可以查询使用，不能改变它的值。

如果想要声明一个变量参数，可以在参数定义前加 inout 关键字，这样就可以改变这个参数的值了。

例如：

func getName(\_ name: inout String)

此时这个 name 值可以在函数中改变。

一般默认的参数传递都是传值调用的，而不是传引用。所以传入的参数在函数内改变，并不影响原来的那个参数。传入的只是这个参数的副本。

当传入的参数作为输入输出参数时，需要在参数名前加 & 符，表示这个值可以被函数修改。

示例

func swapTwoInts(\_ a: inout Int, \_ b: inout Int) {

let temporaryA = a

a = b

b = temporaryA

}

var x = 1

var y = 5

swapTwoInts(&x, &y)

print("x 现在的值 \(x), y 现在的值 \(y)")

swapTwoInts(\_:\_:) 函数简单地交换 a 与 b 的值。该函数先将 a 的值存到一个临时常量 temporaryA 中，然后将 b 的值赋给 a，最后将 temporaryA 赋值给 b。

需要注意的是，someInt 和 anotherInt 在传入 swapTwoInts(\_:\_:) 函数前，都加了 & 的前缀。

以上程序执行输出结果为：

x 现在的值 5, y 现在的值 1

5.4函数的返回值

在函数中可以有返回值，也可以没有返回值。函数是否有返回值以及返回值的数据类型都是与函数声明定义有关。本小结主要讲解函数返回值的情况。

1. 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

}

注：返回的表达式类型必须与函数的返回值类型一致。

1. 具有一个返回值的函数

函数不仅可以返回一个返回值，还可以返回多个，这时就需要使用到元组类型。其语法形式如下：

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...）

}

示例 以下将计算两个整数和两个字符串的最大值，并返回。代码如下：

func maxvalue(\_ value1:Int,\_ value2:Int,\_ str1:String,\_ str2:String)->(Int,String){

    var maxInt=0;

    var maxString:String

    if(value1>value2){

        maxInt=value1

    }else{

        maxInt=value2

    }

    if(str1>str2){

        maxString=str1

    }else{

        maxString=str2

    }

    return (maxInt,maxString)

}

print(maxvalue(5, 1, "hello", "swift"))

运行程序，会看到如下结果：

**(5, "swift")**

注：在->后面的返回值类型的个数要和return中返回值个数一致,对应的类型也要一致，否则程序会报错。

## 5.5函数类型

在swift中，每一个函数都是一个特定的类型，这种类型称为函数类型，由函数的参数类型和返回类型组成。例如以下函数：

func sum(\_ value1:Int,\_ value2:Int,\_ v:Int)->Int{

    return value1+value2+v

}

这个函数的类型就是（Int,Int）->Int，开发者可以理解为函数类型有2个Int整形参数，并返回一个Int整型值。对于不带参数和返回值的函数，其函数类型为（）->（）。

在 Swift 中，使用函数类型就像使用其他类型一样。其一般形式如下：

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

示例 以下将使用一个变量引用函数。代码如下：

func sum(\_ value1:Int,\_ value2:Int,\_ v:Int)->Int{

    return value1+value2+v

}

var addtion:(Int,Int,Int)->Int=sum

print(addtion(10,20,5))

此时运行程序，会看到如下结果：

35

## 5.6 函数类型作为参数类型

我们可以将函数作为参数传递给另外一个参数：

func sum4(a: Int, b: Int) -> Int {

    return a + b

}

var addition: (Int, Int) -> Int = sum4

print("输出结果: \(addition(40, 89))")

func another(addition: (Int, Int) -> Int, a: Int, b: Int) {

    print("输出结果: \(addition(a, b))")

}

another(addition: sum4, a: 10, b: 20)

此时运行程序，会看到如下结果：

输出结果: 129

输出结果: 30

5.7 函数类型作为返回类型

开发者还可以用函数类型作为另一个函数的返回类型。需要做的是在返回箭头(->)后写一个完整的函数类型，其语法形式如下：

func 函数名（参数列表）->函数类型{

......

}

示例 以下通过给定的值，输出一系列值。代码如下：

func stepForward(input: Int) -> Int {

    return input + 1

}

func stepBackward(input: Int) -> Int {

    return input - 1

}

func chooseStepFunction(backwards: Bool) -> (Int) -> Int {

    return backwards ? stepBackward : stepForward

}

var currentValue = 3

let moveNearerToZero = chooseStepFunction(backwards: currentValue > 0)

print("Counting to zero:")

// Counting to zero:

while currentValue != 0 {

    print("\(currentValue)... ")

    currentValue = moveNearerToZero(currentValue)

}

print("zero!")

此时运行程序，会看到如下结果：

**Counting to zero:**

**3...**

**2...**

**1...**

**zero!**

**5.8函数的嵌套**

函数的嵌套分为嵌套函数和函数的嵌套调用。嵌套函数指的是函数内定义的一个新的函数，外部的函数可以调用函数内定义的函数。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，嵌套函数是对外界不可见的，但是可以被定义这个函数的函数来调用。定义嵌套函数的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。

示例 以嵌套函数的形式重写5.7的最后一个示例：

func chooseStepFunction(backwards: Bool) -> (Int) -> Int {

    func stepForward(input: Int) -> Int { return input + 1 }

    func stepBackward(input: Int) -> Int { return input - 1 }

    return backwards ? stepBackward : stepForward

}

var currentValue = -4

let moveNearerToZero = chooseStepFunction(backwards: currentValue > 0)

// moveNearerToZero now refers to the nested stepForward() function

while currentValue != 0 {

    print("\(currentValue)... ")

    currentValue = moveNearerToZero(currentValue)

}

print("zero!")

此时运行程序，会看到如下结果：

**Counting to zero:**

**3...**

**2...**

**1...**

**zero!**

1. 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。

示例 以下通过递归调用来计算5！的值：

func facto(value:Int)->Int{       //递归调用

    if(value==1){

        return value

    }else{

        return value\*facto(value: value-1)

    }

}

print(facto(value: 5))

此时运行程序，会看到如下结果：

120