Swift 字典 Swift 闭包

Swift 函数

Swift 函数用来完成特定任务的独立的代码块。

Swift使用一个统一的语法来表示简单的C语言风格的函数到复杂的Objective-C语言风格的方法。

- 函数声明: 告诉编译器函数的名字，返回类型及参数。
- 函数定义: 提供了函数的实体。

Swift 函数包含了参数类型及返回值类型：

函数定义

Swift 定义函数使用关键字 **func**。

定义函数的时候，可以指定一个或多个输入参数和一个返回值类型。

每个函数都有一个函数名来描述它的功能。通过函数名以及对应类型的参数值来调用这个函数。函数的参数传递的顺序必须与参数列表相同。

函数的实参传递的顺序必须与形参列表相同，**->** 后定义函数的返回值类型。

语法

```
func funcname(形参) -> returntype
{
   Statement1
   Statement2
   ……
   Statement N
   return parameters
}
```

实例

以下我们定义了一个函数名为 runoob 的函数，形参的数据类型为 String，返回值也为 String：

```
import Cocoa

func runoob(site: String) -> String {
    return (site)
}
print(runoob(site: "www.runoob.com"))
```

以上程序执行输出结果为：

```
www.runoob.com
```

函数调用

我们可以通过函数名以及对应类型的参数值来调用函数，函数的参数传递的顺序必须与参数列表相同。

以下我们定义了一个函数名为 runoob 的函数，形参 site 的数据类型为 String，之后我们调用函数传递的实参也必须 String 类型，实参传入函数体后，将直接返回，返回的数据类型为 String。

```
import Cocoa

func runoob(site: String) -> String {
    return (site)
}
print(runoob(site: "www.runoob.com"))
```

以上程序执行输出结果为：

```
www.runoob.com
```

函数参数

函数可以接受一个或者多个参数，这些参数被包含在函数的括号之中，以逗号分隔。

以下实例向函数 runoob 传递站点名 name 和站点地址 site：

```
import Cocoa

func runoob(name: String, site: String) -> String {
    return name + site
}
print(runoob(name: "菜鸟教程: ", site: "www.runoob.com"))
print(runoob(name: "Google: ", site: "www.google.com"))
```

以上程序执行输出结果为：

```
菜鸟教程: www.runoob.com
Google: www.google.com
```

不带参数函数

我们可以创建不带参数的函数。

语法：

```
func funcname() -> datatype {
   return datatype
}
```

实例

```
import Cocoa

func sitename() -> String {
    return "菜鸟教程"
}
print(sitename())
```

以上程序执行输出结果为：

```
菜鸟教程
```

元组作为函数返回值

函数返回值类型可以是字符串，整型，浮点型等。

元组与数组类似，不同的是，元组中的元素可以是任意类型，使用的是圆括号。

你可以用元组（tuple）类型让多个值作为一个复合值从函数中返回。

下面的这个例子中，定义了一个名为minMax(_:)的函数，作用是在一个Int数组中找出最小值与最大值。

```
import Cocoa

func minMax(array: [Int]) -> (min: Int, max: Int) {
    var currentMin = array[0]
    var currentMax = array[0]
    for value in array[1..<array.count] {
        if value < currentMin {
            currentMin = value
        } else if value > currentMax {
            currentMax = value
        }
    }
    return (currentMin, currentMax)
}

let bounds = minMax(array: [8, -6, 2, 109, 3, 71])
print("最小值为 \(bounds.min) ，最大值为 \(bounds.max)")
```

minMax(_:)函数返回一个包含两个Int值的元组，这些值被标记为min和max，以便查询函数的返回值时可以通过名字访问它们。

以上程序执行输出结果为：

```
最小值为 -6 ，最大值为 109
```

如果你不确定返回的元组一定不为nil，那么你可以返回一个可选的元组类型。

你可以通过在元组类型的右括号后放置一个问号来定义一个可选元组，例如(Int, Int)?或(String, Int, Bool)?

> *注意*
> *可选元组类型如`(Int, Int)?`与元组包含可选类型如`(Int?, Int?)`是不同的.可选的元组类型，整个元组是可选的，而不只是元组中的每个元素值。*

前面的`minMax(_:)`函数返回了一个包含两个`Int`值的元组。但是函数不会对传入的数组执行任何安全检查，如果`array`参数是一个空数组，如上定义的`minMax(_:)`在试图访问`array[0]`时会触发一个运行时错误。

为了安全地处理这个"空数组"问题，将`minMax(_:)`函数改写为使用可选元组返回类型，并且当数组为空时返回`nil`：

```
import Cocoa

func minMax(array: [Int]) -> (min: Int, max: Int)? {
    if array.isEmpty { return nil }
    var currentMin = array[0]
    var currentMax = array[0]
    for value in array[1..<array.count] {
        if value < currentMin {
            currentMin = value
        } else if value > currentMax {
            currentMax = value
        }
    }
    return (currentMin, currentMax)
}
if let bounds = minMax(array: [8, -6, 2, 109, 3, 71]) {
    print("最小值为 \(bounds.min)，组大值为 \(bounds.max)")
}
```

以上程序执行输出结果为：

```
最小值为 -6，组大值为 109
```

没有返回值函数

下面是 runoob(_:) 函数的另一个版本，这个函数接收菜鸟教程官网网址参数，没有指定返回值类型，并直接输出 String 值，而不是返回它：

```
import Cocoa

func runoob(site: String) {
    print("菜鸟教程官网: \(site)")
}
runoob(site: "http://www.runoob.com")
```

以上程序执行输出结果为：

```
菜鸟教程官网: http://www.runoob.com
```

函数参数名称

函数参数都有一个外部参数名和一个局部参数名。

局部参数名

局部参数名在函数的实现内部使用。

```
func sample(number: Int) {
   println(number)
}
```

以上实例中 number 为局部参数名，只能在函数体内使用。

```
import Cocoa

func sample(number: Int) {
    print(number)
}
sample(number: 1)
sample(number: 2)
sample(number: 3)
```

以上程序执行输出结果为：

```
1
2
3
```

外部参数名

你可以在局部参数名前指定外部参数名，中间以空格分隔，外部参数名用于在函数调用时传递给函数的参数。

如下你可以定义以下两个函数参数名并调用它：

```
import Cocoa

func pow(firstArg a: Int, secondArg b: Int) -> Int {
   var res = a
   for _ in 1..<b {
      res = res * a
   }
   print(res)
   return res
}
pow(firstArg:5, secondArg:3)
```

以上程序执行输出结果为：

```
125
```

> *注意*
> *如果你提供了外部参数名，那么函数在被调用时，必须使用外部参数名。*

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。

```
import Cocoa

func vari<N>(members: N...){
    for i in members {
        print(i)
    }
}
vari(members: 4,3,5)
vari(members: 4.5, 3.1, 5.6)
vari(members: "Google", "Baidu", "Runoob")
```

以上程序执行输出结果为：

```
4
3
5
```

```
4.5
3.1
5.6
Google
Baidu
Runoob
```

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。

实例

```
import Cocoa

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
```

函数类型及使用

每个函数都有种特定的函数类型，由函数的参数类型和返回类型组成。

```
func inputs(no1: Int, no2: Int) -> Int {
   return no1/no2
}
```

inputs 函数类型有两个 Int 型的参数(no1、no2)并返回一个 Int 型的值。

实例如下：

```
import Cocoa

func inputs(no1: Int, no2: Int) -> Int {
    return no1/no2
}
print(inputs(no1: 20, no2: 10))
print(inputs(no1: 36, no2: 6))
```

以上程序执行输出结果为：

```
2
6
```

以上函数定义了两个 Int 参数类型，返回值也为 Int 类型。

接下来我们看下如下函数，函数定义了参数为 String 类型，返回值为 String 类型。

```
func inputstr(name: String) -> String {
   return name
}
```

函数也可以定义一个没有参数，也没有返回值的函数，如下所示：

```
import Cocoa

func inputstr() {
   print("菜鸟教程")
   print("www.runoob.com")
}
inputstr()
```

以上程序执行输出结果为：

```
菜鸟教程
www.runoob.com
```

使用函数类型

在 Swift 中，使用函数类型就像使用其他类型一样。例如，你可以定义一个类型为函数的常量或变量，并将适当的函数赋值给它：

```
var addition: (Int, Int) -> Int = sum
```

解析:

"定义一个叫做 `addition` 的变量，参数与返回值类型均是 `Int` ，并让这个新变量指向 `sum` 函数"。

`sum` 和 `addition` 有同样的类型，所以以上操作是合法的。

现在，你可以用 addition 来调用被赋值的函数了：

```
import Cocoa

func sum(a: Int, b: Int) -> Int {
   return a + b
}
var addition: (Int, Int) -> Int = sum
print("输出结果: \(addition(40, 89))")
```

以上程序执行输出结果为：

```
输出结果: 129
```

函数类型作为参数类型、函数类型作为返回类型

我们可以将函数作为参数传递给另外一个参数：

```
import Cocoa

func sum(a: Int, b: Int) -> Int {
    return a + b
}
var addition: (Int, Int) -> Int = sum
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: sum, a: 10, b: 20)
```

以上程序执行输出结果为：

```
输出结果: 129
输出结果: 30
```

函数嵌套

函数嵌套指的是函数内定义一个新的函数，外部的函数可以调用函数内定义的函数。

实例如下：

```
import Cocoa

func calcDecrement(forDecrement total: Int) -> () -> Int {
   var overallDecrement = 0
   func decrementer() -> Int {
      overallDecrement -= total
      return overallDecrement
   }
   return decrementer
}
let decrem = calcDecrement(forDecrement: 30)
print(decrem())
```

以上程序执行输出结果为：

```
-30
```

Swift 字典 Swift 闭包