使用func和closure加工数据

返回视频 (/series/functions-and-closure)

Swift 3关于函数类型的一项重要提议 (https://www.boxueio.com/series/functions-and-closure/ebook/149)

Back to series (/series/functions-and-closure)

字号 字号 默认主题 金色主题 暗色主题

函数的返回值以及灵活多变的参数

函数，作为另外一项几乎任何编程语言中都有的特性，我们在之前的内容中其实已经反复使用过了。那为什么又要把它单独拿出来作为一个系列呢？因为，除了封装逻辑、传参、调用、获取结果这种最基础的用法之外，Swift中的函数还有很多设计上的特性以及使用经验，值得我们理解和掌握。

作为开始，在这一节中，我们就快速回顾一下Swift中函数的最基本要素，这将是我们接下来所有内容的基础。如果你认为自己已经比较熟悉Swift函数的基本用法，大可以跳过这一节；否则，就跟着我们开始吧。

一个最简单的函数

一个最简单的函数，看上去是这样的：

```
func printName() {
    print("My name is Mars")
}
```

其中：

- `func` 是定义函数的关键字，后面是函数名；
- `()` 中是可选的参数列表，既然是最简单的函数，自然我们可以让它留空；
- `()` 后面，是函数的返回值，同样，简单起见，我们也没有定义返回值；
- `{}` 中是函数要封装的逻辑，其实，在这里，我们调用的 `print` ，也是一个函数，只不过，它是一个定义在标准库中的函数，并且带有一个参数罢了；

定义好函数之后，我们就可以直接像这样，来调用它：

```
printName() // My name is Mars
```

就可以在控制台看到*My name is Mars*的打印结果了。

向函数传递参数

当然，上面这个最简单的函数除了演示用法外，并没有任何实际意义。我们定义函数当然不是为了反复在控制台打印同样的内容。为了使用同样的逻辑加工不同的数据，第一个要做的事情，就是定义函数参数。例如，为了计算两个整数的乘积：

```
func mul(m: Int, n: Int) {
    print(m * n)
}
```

然后，我们通过下面这样来使用 `mul` ：

```
mul(m: 2, n: 3) // 6
```

就可以在控制台看到打印的结果6了。

理解参数的两种名称

在Swift里，函数的参数实际上有两个名字，一个用于在定义函数的时候使用，叫做argument name，一个用于在调用函数时使用，叫做argument label。

但是，其实我并不是很喜欢这个称呼，因为我经常搞混name和label，谁用在定义，谁用在调用。因此，我更喜欢管定义的时候使用的名称叫做 `internal name` ，表示在函数内部使用；而管调用的时候使用的名称叫做 `external name` ，表示在函数外部使用。而在下面的例子里，我也会使用这两个名字。

在我们的 `mul` 例子中，m和n，就是internal name，默认情况下，如果不特别定义external name，它和internal name则是相等的。或者，我们也可以像这样，来自定义external name：

```
func mul(multiplicand m: Int, of n: Int) {
    print(m * n)
}
```

然后，我们就必须使用 mul 的external name来调用它了：

```
mul(multiplicand: 2, of: 3) // 6
```

或者，如果你不需要在函数调用的时候，使用external name，就要在定义函数的时候，在external name的位置，明确使用 _ 表示忽略：

```
func mul(_ m: Int, of n: Int) {
    print(m * n)
}
```

然后我们就可以这样使用 mul 了：

```
mul(2, of: 3)
```

> 在Swift 3里，函数的第一个参数不再默认缺省忽略external name，它和函数的其他参数是一样的。

所以，除了站在语法的角度来理解这两个名称之外，我们也可以从一个更实际的角度来理解它们：

- External name为了让函数在调用的时候，呈现更好的语义；
- Internal name为了让函数在实现的时候，呈现更好的实现逻辑；

如果internal name可以兼顾两者，你也就无须再单独定义external name了。

为参数设置默认值

除了参数名之外，另一个我们会经常处理的问题，是参数的默认值。它可以用来约束函数的默认行为，或者简化绝大多数时候都会传递的值。例如：

```
func mul(_ m: Int, of n: Int = 1) {
    print(m * n)
}
```

当我们这样使用 mul 时：

```
mul(2) // 2
```

就会计算 2 * 1 的结果，并在控制台看到2。

> 拥有默认值的函数参数必须从右向左依次排列，有默认值的参数不能出现在无默认值的参数的左边。

定义可变长参数

接下来，如果我们要计算不确定个数参数的乘积该怎么办呢？Swift还允许我们通过下面的方式，定义可变长度的参数列表：

```
func mul(_ numbers: Int ...) {
    let arrayMul = numbers.reduce(1, *)
    print("mul: \(arrayMul)")
}
```

在上面的例子中，我们用 numbers: Int... 的形式，表示函数可以接受的 Int 参数的个数是可变的。实际上， numbers 的类型，是一个 Array<Int> ，因此，为了计算乘积，我们直接使用 Array 类型的 reduce 方法就好了。

定义好之后，我们可以这样调用它：

```
mul(2, 3, 4, 5, 6, 7) // 5040
```

就能在控制台看到打印的计算结果了。

定义inout参数

在Swift里，函数的参数有一个性质：默认情况下，参数是只读的，这也就意味着：

- 你不能在函数内部修改参数值；
- 你也不能通过函数参数对外返回值；

先来看第一条：

```
func mul(result: Int, _ numbers: Int ...) {
    result = numbers.reduce(1, *) // !!! Error here !!!
    print("mul: \(result)")
}
```

在上面的实现里，函数的参数默认是个常量，因此编译器会提示你不能在函数内部对常量赋值。然后再来看第二条：如果我们希望参数可以被修改，并且把修改过的结果返回给传递进来的参数，该怎么办呢？

其实，很简单，我们需要用 inout 关键字修饰一下参数的类型，明确告诉Swift编译器我们要修改这个参数的值：

```
func mul(result: inout Int, _ numbers: Int ...) {
    result = numbers.reduce(1, *) // !!! Error here !!!
    print("mul: \(result)")
}
```

然后，就可以这样来使用 mul 了：

```
var result = 0
mul(result: &result, 2, 3, 4, 5, 6, 7)
result // 5040
```

注意到了没？对于 inout 类型的参数，我们在调用函数的时候，也需要在参数前明确使用 & 。这样，mul 执行结束后，就可以看到 result 的值，变成了5040。

通过函数返回内容

当然，通过参数来获取返回值只能算函数的某种副作用，更“正统”的做法，应该是把返回值放在函数的定义里，像这样：

```
func mul(_ numbers: Int ...) -> Int {
    return numbers.reduce(1, *)
}
```

我们通过 -> Type 的方式，在参数列表后面定义返回值。然后，就可以用 mul 的返回值，来定义变量了：

```
let result = mul(2, 3, 4, 5, 6, 7) // 5040
```

What's next?

以上，就是和函数相关的最基本的内容，对于每一个函数来说，把这些基本元素融合在一起，就形成了这个函数的签名，也就是函数自身的类型。在下一节中，我们就来单独谈谈和函数类型相关的话题。

◀ 返回视频 (/series/functions-and-closure)

Swift 3关于函数类型的一项重要提议 ▶ (https://www.boxueio.com/series/functions-and-closure/ebook/149)

猎云网采访报道泊学 (http://www.lieyunwang.com/archives/144329)
Dec 31, 2015

What most schools do not teach (https://www.boxueio.com/what-most-schools-do-not-teach)
Dec 21, 2015

一个工作十年PM终创业的故事（一） (https://www.boxueio.com/founder-story)
May 8, 2015

泊学相关

关于泊学 ›

加入泊学 ›

泊学用户隐私以及服务条款 (HTTPS://WWW.BOXUEIO.COM/TERMS-OF-SERVICE)

版权声明 (HTTPS://WWW.BOXUEIO.COM/COPYRIGHT-STATEMENT)

联系泊学

Email: 10[AT]boxue.io (mailto:10@boxue.io)
QQ: 2085489246

友情链接 SwiftV (http://www.swiftv.cn) | Seay信息安全博客 (http://www.cnseay.com) | Swift.gg (http://swift.gg/) | Laravist (http://laravist.com/) | SegmentFault (https://segmentfault.com) | 靛青K的博客 (http://blog.dianqk.org/)