使用func和closure加工数据

函数的返回值以及灵活多变的参数
(https://www.boxueio.com/series/functions-and-closure/ebook/148)

函数和Closure真的是不同的类型么？
(https://www.boxueio.com/series/functions-and-closure/ebook/150)

Swift 3关于函数类型的一项重要提议

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

字号
字号
默认主题
金色主题
暗色主题

对于一个函数来说，把它的参数和返回值放在一起，就形成了它的签名，这是用于描述函数自身属性的一种类型。为什么要把这个话题单独拿出来呢？这是因为在Swift 3的一项修正案里，对函数参数名称在签名中的地位进行了调整（详见SE-0111 Remove type system significance of function argument labels (https://github.com/apple/swift-evolution/blob/master/proposals/0111-remove-arg-label-type-significance.md)）。而这项调整，存在着一些让人容易困惑的地方。

参数名称不再是函数类型的一部分

不过没关系，我们从一个简单的例子开始。对于上一节中，我们实现的 `mul`：

```
func mul(m: Int, of n: Int) -> Int {
    return m * n
}
```

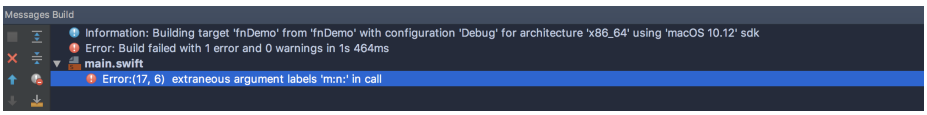
如果我们用type inference定义一个和 `mul` 类型相同的函数变量：

```
let fnMul = mul
```

之后，我们该如何使用 `fnMul` 呢？**在Swift 2里**，是这样的：

```
// ONLY 6 in Swift 2
// WRONG in Swift 3
fnMul(m: 2, n: 3)
```

而到了Swift 3里，编译器会报错：



我们只能这样使用 `fnMul`：

```
fnMul(2, 3) // 6
```

这说明什么呢？从结果上来看 `m:n:` 已经在不再是 `mul` 签名的一部分了。在Swift 3里， `mul` 和 `fnMul` 的类型，实际上都是：`(Int, Int) -> Int`。

然而，为什么要做这样的改动呢？我们继续来看一个在Swift 2中因为函数签名带来的诡异的问题。

假设，我们还有一个用于计算整数除法的函数：

```
func div(a: Int, b: Int) -> Int {
    return a / b
}
```

然后，对于下面这段代码，你期望最后 `fnDiv` 得到什么结果呢?

```
// !!! Swift 2 ONLY !!!
var fnDiv: (a: Int, b: Int) -> Int = div
fnDiv = mul

fnDiv(a: 2, b: 3) // This will call mul(m: 2, n: 3)
// !!! Swift 2 ONLY !!!
```

结果会是6， `fnDiv(a: 2, b: 3)` 实际上调用了 `mul`。造成这个结果的原因有2：

1. 首先， `mul` 和 `div` 都接受2个 `Int` 作为参数，并返回一个 `Int`，因此，尽管它们的参数名不同，但它们的类型是兼容的，我们可以让 `fnDiv = mul`；
2. 其次，由于参数名又是函数签名的一部分，我们使用 `fnDiv` 的时候，必须带上 `a:b:`；

因此，就造成了这种完全是 div 的用法，而暗地里执行了 mul 的错觉。如果这段代码隐藏在你的源文件中，你能轻易发现问题的原因么？

正是由于这种原因，在Swift 3里，参数名不再是函数签名的一部分了， mul 和 div 的类型都是 (Int, Int) -> Int 。这样：

- 之前我们定义的 fnMul ，只能用 fnMul(2, 3) 这样的形式调用；
- 无论 fnMul 被赋值成任何函数，调用方式都是统一的： fnMul(num1, num2) ，这样就不会有因为参数名带来的调用错觉了；

为什么管函数类型叫做“一等公民”?

理解了函数类型之后，接下来我们来聊另外一个话题。你也许经常会听到类似“函数在Swift中是一等公民”这样的说法。然而，对一门编程语言来说，“一等公民”意味着什么呢？

简单来说，就是**函数这种类型，和Swift其它类型有着完全相同的语法功能**。它们主要包括：

- 可以用来定义变量，例如我们之前定义的 fnMul 和 fnDiv ，这种类型的变量同样可以当作函数来调用；
- 可以当成函数参数；
- 可以被函数返回；

而对于后两种情况之所以是很重要的语言特性，是因为它们构成了函数式编程的基础。例如，我们定义一个表达计算概念的函数：

```
func calc<T>(_ first: T,
             _ second: T,
             _ fn: (T, T) -> T) -> T {
    return fn(first, second)
}
```

在这个函数的定义里， calc<T> 就接受一个函数类型的参数，然后，我们可以像传递数字一样，来传递函数变量：

```
calc(2, 3, mul) // 6
calc(2, 3, div) // 0
```

类似的，我们也可以把函数类型当成返回值，例如下面这个有点儿特殊的 mul 方法：

```
func mul(_ a: Int) -> (Int) -> Int {
    func innerMul(_ b: Int) -> Int {
        return a * b
    }

    return innerMul
}
```

它只接受一个参数，并返回一个函数，在返回的函数里，接受另外一个参数，并最终完成相乘的运算。因此，我们可以用两种不同的方式来调用它：

```
let mul2By = mul(2)
mul2By(3) // 6

mul(2)(3) // 6
```

第一种方式，我们先用一个变量保存 mul 返回的函数，然后通过再传递一个单参数来最终完成运算。或者，我们可以直接通过 () 把两次函数调用串联起来。因此，尽管currying function在Swift 3中被废除了（详见SE-0002 Removing currying func declaration syntax (https://github.com/apple/swift-evolution/blob/master/proposals/0002-remove-currying.md)），但我们还是可以通过这样的方法来实现同样的效果。

What's next?

以上，就是Swift 3中，和函数类型相关的话题。理解了Swift对这种类型的改进，以及它在Swift中的地位之后，下一节，我们将讨论和函数相关的另外一个话题——Closure，这又是一个在很多现代化编程语言中都具有的语言特性。有人说，Closure就是匿名的函数，也有人说，Closure就是可以捕获变量的函数，那么，究竟在Swift里，Closure和函数之间有什么关系呢？

(https://www.boxueio.com/series/functions-and-closure/ebook/148) (https://www.boxueio.com/series/functions-and-closure/ebook/150)

职场漂泊的你，每天多学一点。

从开发、测试到运维，让技术不再成为你成长的绊脚石。我们用打磨产品的精神去传播知识，把最新的移动开发技术，通过简单的图表， 清晰的视频，简明的文字和切实可行的例子一一向你呈现。让学习不仅是一种需求，也是一种享受。

泊学动态

一个工作十年PM终创业的故事（二） (https://www.boxueio.com/after-the-full-upgrade-to-swift3)
Mar 4, 2017

人生中第一次创业的"10有" (https://www.boxueio.com/founder-chat)
Jan 9, 2016

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/)