What and Why in Swift 3.1

SE-0045 Sequence中新添加的两个筛选元素的方法 (https://www.boxueio.com/series/what-is-new-in-swift-31/ebook/208)

SE-0141 通过available约束Swift版本 (https://www.boxueio.com/series/what-is-new-in-swift-31/ebook/210)

# SE-0103 临时转换成escaping的closure

Back to series (/series/what-is-new-in-swift-31)

字号
字号
默认主题
金色主题
暗色主题

在这一节里，我们来看和函数的closure参数默认属性相关的话题。

## SE-0103和SR-4188

就像SE-0103 (https://github.com/apple/swift-evolution/blob/master/proposals/0103-make-noescape-default.md)中描述的一样，在Swift 3里，函数的closure类型参数默认从escaping变成了non-escaping。这很好理解，因为大多数用于函数式编程的closure参数的确都以non-escaping的方式工作。

但这份提议也提到了一个问题，就是有时候，我们需要把non-escaping属性的closure，传递给需要escaping属性closure的函数。来看个例子：

```
func increaseValue(in array: [Int], with: () -> Int) {
    // !!! The following code won't compile !!!
    let increasedArray = array.lazy.map { $0 + with() }

    // Some processing here

    print(increasedArray[0])
    print(increasedArray[1])
}
```

我们通过 `increaseValue` 给 `array` 中的元素增加固定的值 `with` ，为了演示刚才提到的场景，我们特意使用了一个 `closure` 来获取要增加的值。在 `increaseValue` 的实现里，我们使用了 `array.lazy` 来推迟对数组值实际的增加动作。在完成一些特定的处理之后，最后，我们打印了 `increasedArray[0]` 和 `increasedArray[1]` ，因此，实际的增加动作只发生了两次。

这个想法看似很好，但实际上并不能通过编译。因为， `lazy.map` 的closure参数，是带有 `@escaping` 属性的，而我们的 `increaseValue` 参数中， `with` 默认是non-escaping属性的。

那么，有必要在函数签名上，让 `with` 带上 `@escaping` 么？想一下就知道，这并不合理。为什么指定加数的参数会可能escape呢？Escape之后，它和谁相加呢？何况，在 `increaseValue` 中使用 `lazy` 属性完全是一个实现细节，无论如何让实现细节去影响公开签名应该不是个好主意。

怎么办呢？为此，Swift提供了一个把non-escaping函数临时变成escaping函数的方法 `withoutActuallyEscaping` ，于是，我们之前的例子可以改成这样：

```
func increaseValue(in array: [Int], with: () -> Int) {
    withoutActuallyEscaping(with) { escapedWith in
        let increasedArray =
            array.lazy.map { $0 + escapedWith() }

        // Some processing here

        print(increasedArray[0])
        print(increasedArray[1])
    }
}
```

`withoutActuallyEscaping` 有两个参数，第一个参数表示转换前的non-escaping closure，第二个参数也是一个closure，用来执行需要escaping closure的逻辑，它有一个参数，就是转换后的closure。因此，在我们的例子里， `escapedWith` 就是转换后的 `with` 。

然后，我们就可以这样来使用 `increaseValue` 了：

```
increaseValue(in: [1, 2, 3, 4, 5], with: { return 2 })
// 3
// 4
```

SR-4188

这就是SE-0103中，关于closure类型参数默认属性的内容的补充。但如果你对Swift足够熟悉，就会发现之前的例子还有一点改进的空间，我们可以让 `with` 参数是一个 `@autoclosure` ，这样，我们就可以直接传递一个值来替代closure这种形式了。

```
func increaseValue(in array: [Int],
    with: @autoclosure () -> Int) { // !!! This will not compile !!!
    withoutActuallyEscaping(with) { escapedWith in
        let increasedArray =
            array.lazy.map { $0 + escapedWith() }

        // Some processing here

        print(increasedArray[0])
        print(increasedArray[1])
    }
}

increaseValue(in: [1, 2, 3, 4, 5], with: 2)
```

但遗憾的是，至少在现在的Swift 3.1版本中，还不能这样。 `withoutActuallyEscaping` 并不支持 `@autoclosure` 修饰的closure参数，上面的代码会导致编译错误。而Swift官方也提供了一个临时方案，就是再通过一个内嵌函数把 `@autoclosure` 参数过渡成普通closure参数，然后在这个内嵌函数里，使用 `withoutActuallyEscaping` ：

```
func increaseValue(in array: [Int],
                   with: @autoclosure () -> Int) {

    func increaseValueTmp(in array: [Int], with: () -> Int) {
        withoutActuallyEscaping(with) { escapedWith in
            let increasedArray = array.lazy.map { $0 + escapedWith() }

            // Some processing here

            print(increasedArray[0])
            print(increasedArray[1])
        }
    }

    increaseValueTmp(in: array, with: with)
}

increaseValue(in: [1, 2, 3, 4, 5], with: 2)
```

这样，就完美了。

SE-0045 Sequence中新添加的两个筛选元素的方法 (https://www.boxueio.com/series/what-is-new-in-swift-31/ebook/208)

SE-0141 通过available约束Swift版本 (https://www.boxueio.com/series/what-is-new-in-swift-31/ebook/210)

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/)