

☰ 使用func和closure加工数据

◀ 被绝大多数人误会的inout参数

Closure什么时候需要escaping? ▶

(<https://www.boxueio.com/series/functions-and-closure/ebook/157>)

(<https://www.boxueio.com/series/functions-and-closure/ebook/159>)

什么时候需要把参数自动转化为closure?

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

在编程语言的组合逻辑判断中，存在着一个逻辑，叫做short circuit。简单来说，就是：

- 对于多个逻辑与 && 串联的情况，如果某一个表达式的值为 false 就不再继续评估后续的表达式了；
- 对于多个逻辑或 || 串联的情况，如果某一个表达式的值为 true 就不再继续评估后续的表达式了；

一个由表达式评估方式引发的问题

通常，我们还会依赖这个特性来编写代码，例如，判断数组第一个元素是否大于0：

```
let numbers: [Int] = [1, 2, 3]

if !numbers.isEmpty && numbers[0] > 0 {
    //
}
```

在上面的代码里，只有当 !numbers.isEmpty 为 true 时，才会评估后面的表达式，因此，我们并不会因为数组越界导致程序崩溃。当如果我们自己定义一个执行逻辑或运算的函数：

```
func logicAnd(_ l: Bool, _ r: Bool) -> Bool {
    guard l else { return false }

    return r
}
```

就没有这种福利了。我们把 numbers 改成一个空数组，然后这样来调用 logicAnd：

```
let numbers: [Int] = []

if !numbers.isEmpty && numbers[0] > 0 {
    // This works
}

if logicAnd(!numbers.isEmpty, numbers[0] > 0) {
    // This failed
}
```

然后就悲剧了，Swift原生的 && 依旧可以正常工作，但我们的 logicAnd 却阵亡了。道理很简单，函数在执行前，要先评估它的所有参数，在评估到 numbers[0] 的时候，发生了运行时异常。此时，我们在 logicAnd 内部通过 guard 模拟的short circuit完全派不上用场。

怎么办呢？

让函数来救场

思路很简单，我们把通过第二个参数获取 Bool 值的过程，封装在一个函数里。在评估 logicAnd 参数的时候，会评估到一个函数类型。我们把真正获取 Bool 的动作，推后到函数执行的时候。

这样不仅解决了评估问题，也真正模拟了short circuit的效果：

```
func logicAnd(_ l: Bool, _ r: () -> Bool) -> Bool {
    guard l else { return false }

    return r()
}
```

但这样修改之后，也有一个副作用，就是我们要修改一下之前进行判断的代码：

- 🔊 字号
- 🔊 字号
- 🖌️ 默认主题
- 🖌️ 金色主题
- 🖌️ 暗色主题

```
if logicAnd(!numbers.isEmpty, { numbers[0] > 0 }) {  
  
}
```

这样,就完全没问题了,除了写法不同之外, `logicAnd` 和 `&&` 的行为简直如出一辙。唯一一点不方便的就是,我们要时刻记着第二个 `Bool` 表达式要通过一个closure来表示。这显然,是一个有违直觉的事情。

有办法么?

靠@autoclosure来自动化

当然有办法,Swift里允许我们通过 `@autoclosure` 来修饰参数类型:

```
func logicAnd(_ l: Bool, _ r: @autoclosure () -> Bool) -> Bool {  
    guard l else { return false }  
  
    return r()  
}
```

这样,我们就可以只写上用于得到返回值的表达式,Swift会自动把这个表达式变成一个closure:

```
if logicAnd(!numbers.isEmpty, numbers[0] > 0) {  
  
}
```

怎么样,现在用起来就更自然了吧。借助于 `@autoclosure`, 我们完美的模拟了带有short circuit性质的 `&&` 操作符。

What's next?

理解了这个过程之后,相信你就可以理解 `@autoclosure` 的主要应用场景了。简单来说,当你需要延后评估某个表达式的值的时候,你就可以考虑这个工具了。接下来,我们来了解另外一个参数的修饰:

`@escaping`, 虽然我们之前也用过几次,大致提及了它的用法。但是,如你已经看过的种种,Swift中的诸多语法设施,都有着鲜为人知故事。

被绝大多数人误会的inout参数

(<https://www.boxueio.com/series/functions-and-closure/ebook/157>)

Closure什么时候需要escaping?

(<https://www.boxueio.com/series/functions-and-closure/ebook/159>)



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](#)

>

[泊学用户隐私及服务条款 \(HTTPS://WWW.BOXUEIO.COM/TERMS-OF-SERVICE\)](https://www.boxueio.com/terms-of-service)[版权声明 \(HTTPS://WWW.BOXUEIO.COM/COPYRIGHT-STATEMENT\)](https://www.boxueio.com/copyright-statement)

联系泊学

Email: 10[AT]boxue.io (<mailto:10@boxue.io>)

QQ: 2085489246

2017 © Boxue, All Rights Reserved. 京ICP备15057653号-1 (<http://www.miibeian.gov.cn/>) 京公网安备 11010802020752号 (<http://www.beian.gov.cn/portal/registerSystemInfo?recordcode=11010802020752>)

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