理解值语义的自定义类型

返回视频 (/series/understand-value-types)

定义更复杂的值 - struct (https://www.boxueio.com/series/understand-value-types/ebook/170)

都是修改对象属性惹的祸

Back to series (/series/understand-value-types)

字号
字号
默认主题
金色主题
暗色主题

修改一个自定义类型的属性，这听起来是个非常平常的事情。但是，随着程序业务逻辑以及计算环境日趋复杂，对象的这种特性却逐渐成为了各种莫名其妙bug的一个主要来源。不信？你来看下面这个例子。

一个可修改的引用语义常量

```
let numbers: NSMutableArray = [1, 2, 3, 4, 5]

for _ in numbers { // Will CRASH when iterating
    numbers.removeLastObject()
}
```

对于上面这段代码，如果你越熟悉Swift，就越会觉得不那么容易理解。 numbers 是个常量，为什么我们可以调用 removeLastObject 方法删除元素呢?

因为在Swift里， NSMutableArray 是个类对象， let 约定的是 numbers 不可以再引用其它的类对象，而并不约定类对象的属性是否可以修改。当然，我们可以把这个现象作为Swift常量类对象的一个性质勉强记住。

但当你执行上面这段的代码的时候，就会发现程序直接崩溃了。也就是说，我们编写了一段表面看上去合法的代码，但却完全无法执行。这次，你应该无论如何都无法接受这个现状了，我们还能好好的写代码么?

之所以会造成上面的问题，是因为在Swift里，通过 for 遍历数组是通过 while 和 Iterator 模拟出来的，换句话说，我们可以把之前的 for 改写成这样：

```
var numberIter = numbers.makeIterator()

while let number = numberIter.next() {
    numbers.removeLastObject()
}
```

对于 numberIter 来说，它的实现直接访问了 NSMutableArray 的底层数据存储，而当我们从 numbers 中删除元素的同时，也会破坏掉 numberIter 内部用于遍历数组的状态。因此，删掉一个元素之后，再调用 numberIter.next() ，就发生运行时错误了。这也就意味着，所有调用了 removeLastObject() 方法的API都会有上面类似的问题。

怎么样，你现在是不是更有如履薄冰的感觉？好在我们有机会了解到这些类型内部的实现，才能对这类问题一探究竟。对于我们不熟悉的类型呢？研究这类bug的过程只能用苦不堪言来形容。也许之前你从没想过，原来修改常量类对象的属性是件如此危险的事情。但现在，你应该有点儿相信我们在这一节开始提到的观点了吧。

一个不可修改的值语义常量

但是，当你把之前的例子换成Swift“原装”的 Array 时，就完全没这个问题了，因为，你无法修改一个 Array 常量：

```
let nubmers = [1, 2, 3, 4, 5]

for _ in numbers {
    // Compile time ERROR
    numbers.removeLast()
}
```

对于这段代码，编译器就会为你守住关卡。它会提示你： numbers 是一个常量。而当你把 numbers 改成一个变量之后，这个循环会在执行5次之后，正常结束。

这是因为，在之前集合的视频中我们提到过，Swift中 Array 的 Iterator 在内部保存了一个 Array 的副本，这个副本和 numbers 是分开独立的，而 for 循环迭代的，实际上是这个副本。因此，尽管我们在循环里不断的在 numbers 末尾删除元素，也不会造成运行时错误。

所以，你看，一个获取拷贝的小操作，居然解决了之前我们的一个大麻烦。然而事情还远不止于此，即便没有 NSMutableArray 和 Iterator 的暗道机关，在多线程环境里，修改一个类对象的属性仍旧容易引发各种bug，我们继续看下面的例子。

对多线程环境里修改共享的类对象保持谨慎

为了演示这个问题，我们定义一个没有“秘密”的 class：

```
class Queue {
    var position = 0
    var array: [Int] = []

    init(_ array: [Int]) {
        self.array = array
    }

    func next() -> Int? {
        guard position < array.count else {
            return nil
        }

        position += 1

        return array[position - 1]
    }
}
```

这就是一个通过 Array 模拟的单次遍历队列，然后，我们写一个遍历 Queue 的函数：

```
func traverseQueue(_ queue: Queue) {
    while let item = queue.next() {
        print(item)
    }
}
```

当我们在主线程中执行下面的代码时，不会有任何问题：

```
let q = Queue([1, 2, 3, 4, 5])
traverseQueue(q)
// 1 2 3 4 5
```

控制台上会依次打印出数字1到5。但是，在多线程环境里，这段代码却有很大概率无法正常工作：

```
for _ in 0..<1000 {
    let q = Queue([1, 2, 3, 4, 5])

    DispatchQueue.global().async {
        traverseQueue(q) // May crash here
    }

    traverseQueue(q) // Or here
}
```

其中 for 循环执行的次数越多，发生异常的概率就越大，而当循环次数较小的时候，通常又不会发生问题。所以，这类问题大概是我们编程中最难发现和调试的一类。而问题的原因，仍旧是我们修改了共享对象的属性。

由于 Queue 是一个引用类型，当 q 分别传递给主线程和全局队列时，传递的都是同一个 Queue 对象的引用，在 next() 的判断里，当 guard position < array.count 判断后，如果发生线程切换，在另外的线程里把 position + 1，再回到之前的线程里读取 Queue.array 的时候，就会发生异常了。

What's next?

通过这些例子，现在你应该相信了吧，一个可以修改的类对象会在很多时候，偷偷给我们带来意想不到的麻烦。所以，越来越多的言论在鼓励我们尽可能使用常量，当然，这里也就包含着值不可被修改的对象。因为这的确可以为我们带来更安全和可维护的代码。

因此，在Swift里，提到自定义类型时，除了传统意义上的 `class` 之外，你会看到更多拥有值语义的类型，例如： `struct` 和 `enum` 。在这个系列接下来的几个视频中，我们就来了解它们的一些惯用方法。

返回视频 (/series/understand-value-types)

定义更复杂的值 - struct (https://www.boxueio.com/series/understand-value-types/ebook/170)

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/)