使用func和closure加工数据

为什么delegate模式不适用于struct类型?
(https://www.boxueio.com/series/functions-and-closure/ebook/155)

被绝大多数人误会了的inout参数
(https://www.boxueio.com/series/functions-and-closure/ebook/157)

是delegate protocol，还是callback?

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

字号
字号
默认主题
金色主题
暗色主题

继续上一节的讨论，如果 `struct` 类型不能当作delegate，当我们要通过它设置某些外部自定义的方法时，该怎么办呢？一个替代方案当然就是**直接在这些类型内定义个函数类型的属性作为callback**。相比 `protocol` 的方式，这样做甚至还更灵活。但是，嗯，你也得牺牲一些特性，我们来看个例子。

通过struct实现函数类型的callback

还是之前的 `FinishAlertView`，这次，我们把统计按钮点击次数的delegate，改成一个属性：

```
class FinishAlertView {
    var buttons: [String] = [ "Cancel", "The next" ]
    var buttonPressed: ((Int) -> Void)?

    func goToTheNext() {
        buttonPressed?(1)
    }
}
```

这样，在 `goToTheNext` 的实现里，我们只要直接这样 `buttonPressed?(1)` 调用它就好了。似乎还比delegate的方式简单了一些。唯一不足的地方，就是你无法为 `buttonPressed` 设置参数label了。所以，这是一个便利性加分，但表意上减分的妥协。

接下来，我们用一个 `struct` 类型，来实现这个callback：

```
struct PressCounter {
    var count = 0

    mutating func buttonPressed(at Index: Int) {
        self.count += 1
    }
}
```

如你所见，这也更简单了，我们都不用声明它遵循某个 `protocol`，直接定义一个和callback签名一样的函数就好了。然后，我们分别定义一个 `FnishAlertView` 和 `PressCounter` 对象：

```
let fav = FinishAlertView()
var counter = PressCounter()
```

该如何设置 `counter.buttonPressed(at:)` 是 `fav` 的回调函数呢？这样肯定不行：

```
fav.buttonPressed = counter.buttonPressed
```

Swift编译器会给你这么个不太容易明白的提示：

```
30
31 fav.buttonPressed = counter.buttonPressed   Partial application of 'mutating' method is not allowed
```

什么叫“Partial application of mutating method is not allowed”呢？简单来说，就是直接从代码的字面值上，我们无法确定一些行为，例如：

- 我们应该拷贝 `counter` 对象么？
- 还是我们应该让 `fav.buttonPressed` 捕获 `counter` 呢？

既然无法确定，编译器只好给我们一个错误，让我们自己明确做一个选择。显然，为了让 `counter` 对象执行计数，我们必须捕获它，而不是拷贝它，否则，就又变成了上一节中我们提到的那个错误场景了。

怎么办呢？很简单，把 `counter.buttonPressed` 封装在一个closure里就好了：

```
fav.buttonPressed = { counter.buttonPressed(at: $0) }
```

这样，编译器就会对你的代码予以放行了。测试一下：

```
fav.goToTheNext()
fav.goToTheNext()
fav.goToTheNext()
fav.goToTheNext()
fav.goToTheNext()
fav.goToTheNext()

counter.count // 6
```

模拟了六次点击之后， `counter.count` 的值就是6了。

感觉怎么样？除了无法在callback签名中指定参数label之外，似乎callback与之前的 `protocol` 相比毫不逊色，甚至看上去方便和灵活：

- 我们无须让类型遵循 `protocol` ；
- 无需在实现callback的类型中使用与callback属性同名的方法；
- 我们甚至可以直接就用一个closure来设置这个callback：

```
fav.buttonPressed = { _ in print("OK, go to the next episode") }
```

怎么样？是不是感觉还不错。但故事至此还没结束，当我们通过一个 `class` 类型来实现callback的时候，事情会更简单。

通过class实现函数类型的callback

我们先把之前的 `PressCounter` 改成一个 `class` ：

```
class PressCounter {
    var count = 0

    func buttonPressed(at Index: Int) {
        self.count += 1
    }
}
```

然后，我们居然就可以使用之前会导致编译失败的方式来设置callback了：

```
fav.buttonPressed = counter.buttonPressed
```

这看似方便，实则危险。或者说，我们就应该避免这种用法。经过上面这种赋值之后，我们就创建了一个 `fav` 到 `counter` 的strong reference。在我们这个例子里，这没问题，因为 `counter` 并没有引用 `fav` 对象。

但想象一下，如果我们让一个view controller来实现这个callback，就很容易写出下面的逻辑：

1. 创建view controller；
2. 在view controller中创建view，这样就创建了controller到view的strong reference；
3. 把view controller设置为view的callback，这样就创建了view到controller的strong reference；

然后呢？然后当然就reference cycle了。

所以，不要贪图上面这样的“便利”写法，甚至，还要比 `struct` 版本写的更复杂：

```
fav.buttonPressed = { [weak counter] index in
    counter?.buttonPressed(at: index)
}
```

我们通过capture list，限定了 `fav` 对 `counter` 是一个弱引用，这样，无论在任何环境里，都不会有引用循环的问题了。

What's next?

以上就是这一节的内容，如同你看到的一样，使用callback替代 `protocol` ，是一个有利也有弊的方案。Callback带来了更大的灵活性和更简洁的代码，却也在使用引用类型时，埋下了引用循环的隐患。

当然，技术细节上的差异只是你在选择实现方案的一部分考量，另一部分，则来自于代码呈现的语义。通常，如果你有若干功能非常相关的回调函数，你还是应该把它们归拢到一起，通过一个 `protocol` 来约束他们。这样，实现这些回调函数的类型，也就变成一个遵从了 `protocol` 的类型（毕竟你无法让一个对象的delegate同时等于多个对象），这一定是比散落在各处的callback要好多了。

在下一节中，我们将讨论一个被很多人都误会了的话题，当我们要通过函数参数返回值的时候，使用 `inout` 真的会让参数按引用传递么？

为什么delegate模式不适用于struct类型?
(https://www.boxueio.com/series/functions-and-closure/ebook/155)

被绝大多数人误会了的inout参数
(https://www.boxueio.com/series/functions-and-closure/ebook/157)

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/)