使用func和closure加工数据

在复杂排序中处理optional (https://www.boxueio.com/series/functions-and-closure/ebook/154)

是delegate protocol，还是callback? (https://www.boxueio.com/series/functions-and-closure/ebook/156)

为什么delegate模式不适用于struct类型?

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

字号
字号
默认主题
金色主题
暗色主题

如果你有过Objective-C的编程经验，一定会对UIKit中的各种delegate再熟悉不过了。UIKit负责定义 `protocol`，对应的Controller负责实现它们，并把自己注册为某个控件的delegate。这样，当特定的事件发生时，Controller中定义的 `protocol` 方法就会被调用了。

虽然这听起来耳熟能详，但在Swift里，却也有一些小细节，制约了这种形式在Swift中的应用。为了理解它们，我们看个例子。

通过class实现的delegate

假设，我们有一个提示用户视频已播放完的 `FinishAlertView` 以及 `FinishAlertViewDelegate`：

```
protocol FinishAlertViewDelegate: class {
    func buttonPressed(at index: Int)
}

class FinishAlertView {
    var buttons: [String] = [ "Cancel", "The next" ]
    weak var delegate: FinishAlertViewDelegate?

    func goToTheNext() {
        delegate?.buttonPressed(at: 1)
    }
}
```

然后，如果我们要使用 `FinishAlertView`，通常的做法，就是在一个view controller里：

1. 初始化 `FinishAlertView` 对象；
2. 把自己注册成 `FinishAlertView` 对象的delegate；
3. 实现 `FinishAlertViewDelegate`；

```
class EpisodeViewController: FinishAlertViewDelegate {
    var episodeAlert: FinishAlertView!

    init() {
        // 1. Init
        self.episodeAlert = FinishAlertView()
        // 2. Register itself
        self.episodeAlert.delegate = self
    }

    // 3. Implement interface
    func buttonPressed(at index: Int) {
        print("Go to the next episode...")
    }
}
```

这看起来没什么问题，我们应该对这个套路很熟悉了。但这里，仍有两个细节是我们需要关注的：

首先，为什么 `protocol FinishAlertViewDelegate` 必须要约束只能通过 `class` 来实现呢?

其次，正是因为我们对 `protocol` 的实现进行了约束，在 `FinishAlertView` 里，我们还要把 `delegate` 修饰为 `weak`；这样，即便 `FinishAlertViewDelegate` 中存在对 `FinishAlertView` 的strong reference，也不会造成引用循环；

但究竟是为什么要如此呢？作为delegate的 `protocol` 为什么不能由 `struct` 类型实现呢?

为什么delegate不能通过struct实现

为了彻底搞清楚这个问题，我们可以试着去掉 `FinishAlertViewDelegate` 的 `class` 约束：

```
protocol FinishAlertViewDelegate {
    mutating func buttonPressed(at Index: Int)
}
```

并且，为了让 struct 类型可以通过 buttonPressed(at:) 修改自身的属性，我们还要用 mutating 来修饰这个方法；

这时，我们就不能在 FinishAlertView 里，使用 weak 修饰 delegate 属性了，因为实现 FinishAlertViewDelegate 的，不一定是一个引用类型，还有可能是值类型。

```
class FinishAlertView {
    var delegate: FinishAlertViewDelegate?

    // omit for simplicity...
}
```

至此，你可以已经有点儿感觉不好了。因为只要我们对 delegate 赋值，就创建了一个到 FinishAlertViewDelegate 对象的strong reference，你得时刻记得，不要在 FinishAlertViewDelegate 的实现里再创建到 FinishAlertView 的strong reference，否则，我们就创造了一个循环引用。

但无论如何，至少Swift没有明令禁止这样做。所以，我们就不妨先继续下去，让另外一个 struct 来实现这个 protocol：

```
struct PressCounter: FinishAlertViewDelegate {
    var count = 0

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

它有一个属性 counter，无论哪个按钮被按下，它都会把计数器+1。然后，在之前的 EpisodeViewController 里，我们把delegate设置成新创建的 PressCounter 对象：

```
class EpisodeViewController {
    var episodeAlert: FinishAlertView!
    var counter: PressCounter!

    init() {
        self.episodeAlert = FinishAlertView()
        self.counter = PressCounter()
        self.episodeAlert.delegate = self.counter
    }
}
```

完成后，我们用下面的代码试一下：

```
let evc = EpisodeViewController()

evc.episodeAlert.goToTheNext()
evc.episodeAlert.goToTheNext()
evc.episodeAlert.goToTheNext()
evc.episodeAlert.goToTheNext()
evc.episodeAlert.goToTheNext()
evc.episodeAlert.goToTheNext()

evc.counter.count // Still 0
```

在我们模拟了6次按钮点击事件之后，我们期望 evc.counter.count 是多少呢？你一定会想：6呗。

但实际上，我们会得到0。要想获得按钮点击次数的统计，我们得这样：

```
(evc.episodeAlert.delegate as! PressCounter).count // 6
```

这是因为 PressCounter 是一个值类型，当我们执行 self.episodeAlert.delegate = self.counter 时， delegate 实际上是 self.counter 的拷贝，它们引用的并不是同一个对象，因此调用 goToTheNext() 的时候，增加的只是 self.episodeAlert.delegate，而不是 self.counter。

通过这个例子，你就知道了，去掉delegate protocol 的 class 约束，并不是一个好主意，这不仅让 class 类型在实现 protocol 的时候引入了strong reference；而对于 struct 类型来说，哈，它原来根本就不配做个delegate。

What's next?

既然 `struct` 类型不能当作delegate，当我们要为它设置某些外部自定义的方法时，该怎么办呢？其实，在很多时候，定义一个回调函数属性就好了。那么，protocol和callback究竟具体的差别在哪呢？我们应该在什么时候使用它们？下一节，我们就来讨论这个话题。

◀ 在复杂排序中处理optional
(https://www.boxueio.com/series/functions-and-closure/ebook/154)

是delegate protocol，还是callback? ▶
(https://www.boxueio.com/series/functions-and-closure/ebook/156)