使用func和closure加工数据

如何通过类型系统模拟OC的运行时特性?
(https://www.boxueio.com/series/functions-and-closure/ebook/153)

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

在复杂排序中处理optional

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

字号
字号
默认主题
金色主题
暗色主题

在上一节我们提到的 SortDescriptor 方案中，还有一个细节我们没有处理，如果要排序的属性是个optional怎么办呢？来看个例子。

为optional类型“提升”比较函数

假设，我们有一个表示1到5的 [String]：

```
let numbers = ["Five", "4", "3", "2", "1"]
```

现在，要按照整数比较它们的大小，怎么做呢？显然，用下面的方法创建 SortDescriptor<String> 是不行的：

```
let intDescriptor: SortDescriptor<String> =
    makeDescriptor(key: { Int($0) }, <)
```

我们会得到下面的错误：

```
90 let intDescriptor: SortDescriptor<String> =
```

这是因为， Int($0) 返回了 Int? ，而在Swift 3中，Apple去掉了对optional类型的比较操作，它们认为大家对于optional中，值和 nil 的比较会引起歧义。但没关系，这并不妨碍我们自己来实现比较optional的过程，一个最直接的手段，就是把比较逻辑直接写在 makeDescriptor 里：

```
let intDescriptor: SortDescriptor<String> =
    makeDescriptor(key: { Int($0) }, {
        return { l, r in
            switch (l, r) {
            case (nil, nil):
                return false
            case (nil, _):
                return false
            case (_, nil):
                return true
            case let (l?, r?):
                return l < r
            default:
                fatalError()
            }
        }
    })
```

这个实现没什么难度，我们只是通过若干个 case 定义了当参与比较的一方或双方都为 nil 时的比较规则。简单来说就是：

- 当参与比较的两个值都不为 nil 时，就用 < 比较它们的大小；
- 我们把所有的 nil 结果放在数组的最后，并且，不对同为 nil 的结果进行比较；

但是，这样的代码还是无法通过编译，编译器会告诉我们无法推导closure expression的类型：

这是因为，我们的 makeDescriptor 使用的比较函数，不能接受两个Optional类型的参数。我们需要定义一个方法，把一个 (T, T) -> Bool ，“提升”成一个 (T?, T?) -> Bool 就好了：

```
func shift<T: Comparable>(
    _ compare: @escaping (T, T) -> Bool) -> (T?, T?) -> Bool {
    return { l, r in
        switch (l, r) {
        case (nil, nil):
            return false
        case (nil, _):
            return false
        case (_, nil):
            return true
        case let (l?, r?):
            return compare(l, r)
        default:
            fatalError()
        }
    }
}
```

这段代码和我们之前在 makeDescriptor 中使用的逻辑，99%都是相同的，只不过，在 l 和 r 都不为 nil 时，我们使用了 compare 参数来比较它们的大小。

有了这个“提升”函数之后，我们就可以这样来定义 intDescriptor：

```
let intDescriptor: SortDescriptor<String> =
    makeDescriptor(key: { Int($0) }, shift(<))
```

这明显要比之前的版本直观多了，而排序的结果，应该和之前是一样的。

自定义组合排序规则的操作符

在上一节中，我们遗留的第二个问题，是合并多个 SortDescriptor 的 combine 方法，暴露了我们本不需要的实现细节。当然，这是我们采取的实现方案不可避免的弊端，但为了让它语法上好看一些，我们可以做一些改进。

例如，用一个自定义的操作符，来表现这种逐层递进的排序关系：

```
episodes.sorted(by: typeDescriptor |> lengthDescriptor)
```

这样，我们就不用先定义成数组，然后再 combine 了。至于 |> 操作符的实现，核心思想和 combine 几乎是一样的。

首先，我们要自定义 |> 操作符：

```
infix operator |>: LogicalDisjunctionPrecedence
```

由于要用在两个操作数中间，所以，我们把它定义为了 infix operator 。另外，由于我们要定义的操作符在语义上，有逻辑上逐层深入的含义，我们还把它的优先级定义为了 LogicalDisjunctionPrecedence ，也就是说，它和 || 以及 && 的优先级是相同的。如果我们不指定优先级，Swift会为它设置默认的 DefaultPrecedence 。

关于Swift 3中，操作符优先级更多的讨论，大家可以去查看SE0077-operator-precedence (https://github.com/apple/swift-evolution/blob/master/proposals/0077-operator-precedence.md)，我们就不过多展开了。

定义好这个操作符之后，我们就可以来实现它了：

```
func |><T>(
    l: @escaping SortDescriptor<T>,
    r: @escaping SortDescriptor<T>) -> SortDescriptor<T> {

    return {
        if l($0, $1) {
            return true
        }

        if l($1, $0) {
            return false
        }

        // $0 and $1 is the same, try the second descriptor
        if r($0, $1) {
            return true
        }

        return false
    }
}
```

看到了吧， |> 的核心实现逻辑和上一节中的 combine 唯一的不同，就是在左操作数比较相等后，我们继续使用了右操作数继续进行比较。

定义好了这个操作符之后，之前串联descriptor的代码也就可以正常工作了：

```
episodes.sorted(by: typeDescriptor |> lengthDescriptor)
```

而得到的结果，应该和之前使用 combine 是一样的。

What's next?

以上，就是我们对 SortDescriptor 方案的补充和优化。至此，关于用函数类型模拟OC运行时特性的内容，基本就结束了。在这三节的内容里，我们始终在贯彻的一个思想，就是把函数类型自身当成一种数据，一方面用类型名称增强代码的语义，另一方面，利用编译器的类型系统在编译期提供类型安全。掌握了这个核心思想，函数就不再只是简单的数据加工机器了，它能在很多领域，有效改进代码质量。

在下一节，我们将讨论另外一个常见的函数应用场景：delegate。在iOS开发中，这是一个我们几乎每天都会与之打交道的模式。但是，在Swift里，我们只能用 class 类型来完成这个工作。为什么会这样呢?

◀ 如何通过类型系统模拟OC的运行时特性?
(https://www.boxueio.com/series/functions-and-closure/ebook/153)

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

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/)