使用func和closure加工数据

通过Local function捕获变量共享资源

(https://www.boxueio.com/series/functions-and-closure/ebook/151)

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)

OC中水土不服的运行时特性

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的开发经验，一定会对它提供的各种运行时特性印象深刻。基于这些特性提供的功能更是灵活强大，可以帮助我们处理一些复杂的任务。但这一切，都是有代价的。没错，它们大多用起来都不直观，如果你不去刨一下文档，总不那么容易理解相关API的正确用法。

并且，既然这些特性是基于运行时的，因此，编译器仅可以对它们执行非常有限的检查。一旦你稍有疏忽，就得承担App闪退的严重后果。

用OC运行时特性进行排序

我们来看个和搜索有关的例子。首先，定义一个表示视频信息的类：

```
final class Episode: NSObject {
    var title: String
    var type: String
    var length: Int

    override var description: String {
        return title + "\t" + type + "\t" + String(length)
    }

    init(title: String, type: String, length: Int) {
        self.title = title
        self.type = type
        self.length = length
    }
}
```

其实，在Swift里，这类内容定义成 `struct` 更合适，但为了演示OC的运行时特性，我们把它定义成了一个派生自 `NSObject` 的类。并且，通过关键字 `final` 限制了它不能继续被继承。

`Episode` 有三个属性，分别表示视频的标题、类型和长度。然后，我们重载了 `description` 属性，以便后面通过 `print` 直接打印 `Episode` 对象。

这一切都很简单，然后，我们定义一些测试数据：

```
let episodes = [
    Episode(title: "title 1", type: "Free", length: 520),
    Episode(title: "title 4", type: "Paid", length: 500),
    Episode(title: "title 2", type: "Free", length: 330),
    Episode(title: "title 5", type: "Paid", length: 260),
    Episode(title: "title 3", type: "Free", length: 240),
    Episode(title: "title 6", type: "Paid", length: 390),
]
```

接下来，我们要先按 `type` 排序，并在排序后的结果里，继续按照 `length` 排序，该怎么办呢？Apple在开发者文档里介绍了一种叫做NSSortDescriptor (https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/SortDescriptors/Articles/Creating.html#//apple_ref/doc/uid/TP40002366-SW1)的用法，这就是一个典型的功能强大，但是又必须要看文档才能掌握的技能。

为了排序 `type` ，首先，我们定义一个 `typeDescriptor`：

```
let typeDescriptor = NSSortDescriptor(
    key: #keyPath(Episode.type),
    ascending: true,
    selector: #selector(NSString.localizedCompare(_:)))
```

其中：

- `key`： 表示要排序的属性；

- ascending： 表示是否按升序排序；
- selector： 表示要进行比较的方法；

其次，定义一个 Array<NSDescriptor>：

```
let descriptors = [ typeDescriptor ]
```

最后，把 episodes 转型成 NSArray ，调用 sortedArray(using:) 方法，把 descriptors 传递给它：

```
let sortedEpisodes = (episodes as NSArray).sortedArray(using: descriptors)
```

这样，就完成排序了，但我们会得到一个 Array<Any> 的结果，为了查看它的内容，我们得这样：

```
sortedEpisodes.forEach { print($0 as! Episode) }
```

然后，我们就可以在控制台看到下面的结果了：

```
title 1 Free    520
title 2 Free    330
title 3 Free    240
title 4 Paid    500
title 5 Paid    260
title 6 Paid    390
```

此时，我们就完成了按 Type 进行排序，接下来，我们还要在这个排序结果里，把 Free 和 Paid 的视频按时间排序。理解了上面的套路之后，就很简单了，我们继续定义一个 lengthDescriptor：

```
let lengthDescriptor = NSSortDescriptor(
    key: #keyPath(Episode.length),
    ascending: true)
```

这次，我们使用系统默认的整数比较操作符就好了，可以不明确指定要使用的selector。定义好之后，直接把它添加到之前创建的 descriptors 数组里：

```
let descriptors = [ typeDescriptor, lengthDescriptor ]
```

这样，重新执行一次， sortedArray(using:) 方法就会返回这样的结果：

```
title 3 Free    240
title 2 Free    330
title 1 Free    520
title 5 Paid    260
title 6 Paid    390
title 4 Paid    500
```

看到了吧，现在，每一类视频里，就是按照时长进行排序的了，这就是 NSSortDescriptor 的用法。当你理解了这个过程之后，就能体会到它的功能强大，我们可以在 descriptors 数组中，包含任意多个不同的 NSSortDescriptor 对象，来实现复杂的搜索功能。但是，如果你不看文档，Hmmmm...，估计你也很难理解它的使用方法。

除了不怎么好学之外，上面的方法在Swift里还有个先天不足，就是我们使用了OC的两个运行时特性：一个是Key-Value coding，用来读取属性中的值，一个是selector，用来表示排序时使用的算法。编译器对这些当然一无所知，只要语法上正确，就会开绿灯。但是，显然，调试运行时错误要比编译错误麻烦的多。

那Swift的方式呢?

显然，尽管 NSSortDescriptor 的思想并不难掌握，但把它用在Swift里，还是显的有点儿水土不服，这主要表现在：

- 首先，从定义之初，就限制了我们必须使用 class ，必须从 NSObject 派生。但显然，这样的信息在Swift更适合定义成 struct ；
- 其次，我们要在使用API的时候，把 Array bridge到 NSArray ，从 NSArray 再bridge回来的时候，类型变成了 Any ，我们还要手工找回类型信息；
- 最后，Key-Value coding和selector都没有利用编译器提供足够充分的类型检查；

所以，对于Swift原生类型来说， NSSortDescriptor 并不是复杂排序规则的最佳解决方案。那就究竟该怎么办呢？你可能会想， Array 不是有一个接受函数参数的 sorted 方法么：

```
episodes.sorted {
    // Complex sorting code here
}
```

但这并不是一个好主意，相比之前 NSSortDescriptor 的方式，不仅我们无法有效表达要排序的规则，而且，把这些规则统统塞进一个排序函数中也并不利于维护。想象一下，如果现在我们又要对 title 和 length 排序了该怎么办呢?

What's next?

为了在Swift中找到更易用和安全的解决方案，我们还是得从 NSSortDescriptor 的思路入手，把排序规则的表达、排序规则的组合，以及执行排序的动作独立分开。只是，为了充分利用Swift编译器提供类型检查，我们要避免使用运行时机制来识别要访问的内容。具体该怎么办呢？大体的思路是这样的，对于比较元素时执行的函数，我们可以去掉selector，直接用Swift中的函数类型。而对于获取要排序的属性，我们也要通过一个函数类型来表达，而不要通过Key-Value coding。在下一节中，我们就基于这个思路，用Swift自身的方式，来模拟OC的这两个运行时特性。

通过Local function捕获变量共享资源
(https://www.boxueio.com/series/functions-and-closure/ebook/151)

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)