Filter / Reduce / FlatMap的实现和扩展

Back to series (/series/collection-types)

字号
字号
默认主题
金色主题
暗色主题

理解了 Array 中使用closure参数化对数组元素操作的核心思想之后，在这一节中我们着重了解三个比较重要的 Array API， filter / reduce / flatMap ，它们和我们在上一节中实现的 map 一起，形成了各种 Array 操作的基础。

filter和与filter类似的语义

之前，我们提到过 filter 的用法，用于在 Array 中，过滤满足特定条件的元素。而这个条件，就是通过 filter 的closure参数来确定的：

```
var fibonacci = [0, 1, 1, 2, 3, 5]

// [0, 2]
fibonacci.filter { $0 % 2 == 0 }
```

按照上一节中实现 map 的思路，我们可以自己来实现一个 filter：

```
extension Array {
    func myFilter(_ predicate: (Element) -> Bool) -> [Element] {
        var tmp: [Element] = []

        for value in self where predicate(value) {
            tmp.append(value)
        }

        return tmp
    }
}
```

在上面的实现里，最核心的环节就是通过带有 where 条件的 for 循环找到原数组中符合条件的元素，然后把它们一一添加到 tmp 中，并最终返回给函数的调用者。然后，我们测试下 myFilter：

```
fibonacci.myFilter { $0 % 2 == 0 } // [0, 2]
```

结果，应该是和标准库中自带的 filter 是一样的。理解了 filter 之后，我们就可以自行定义一些标准库中没有的方法。例如：

剔除掉数组中满足条件的元素：

```
extension Array {
    func reject(_ predicate: (Element) -> Bool) -> [Element] {
        return filter { !predicate($0) }
    }
}
```

我们只要把调用转发给 filter ，然后把指定的条件取反就好了。这样，剔除元素的代码语义上就会更好看一些：

```
fibonacci.reject { $0 % 2 == 0 } // [1, 1, 3, 5]
```

另一个基于 filter 语义的常用操作是判断数组中是否存在满足条件的元素。下面的代码可以完成任务：

```
fibonacci.filter { $0 % 2 == 0 }.count > 0 // true
```

但这样做在性能上并不理想，因为即便找到了满足条件的元素，也要遍历完整个数组，这显然是没必要的。Swift标准库中，提供了一个更方便的方法：

```
fibonacci.contains { $0 % 2 == 0 } // true
```

contains 的一个好处就是只要遇到满足条件的元素，函数的执行就终止了。基于这个 contains ，我们还可以给 Array 添加一个新的方法，用来判断 Array 中所有的元素是否满足特定的条件：

```
extension Array {
    func allMatch(_ predicate: (Element) -> Bool) -> Bool {
        return !contains { !predicate($0) }
    }
}
```

在 allMatch 的实现里，只要没有不满足条件的元素，也就是所有元素都满足条件了。我们可以用下面的代码测试一下：

```
let evens = [2, 4, 6, 8]
evens.allMatch { $0 % 2 == 0 } // true
```

reduce和与reduce相关的语义

除了用一个数组生成一个新的数组，有时，我们会希望把一个数组变成某种形式的值。例如，之前我们提到的求和：

```
fibonacci.reduce(0, +) // 12
```

了解 reduce 的进一步用法之前，我们先来自己实现一个：

```
extension Array {
    func myReduce<T>(_ initial: T, _ next: (T, Element) -> T) -> T {
        var tmp = initial

        for value in self {
            tmp = next(tmp, value)
        }

        return tmp
    }
}
```

从上面的实现就可以看出， reduce 的实现也没有什么神奇之处。无非就是把 for 循环迭代相加的过程封装了起来。然后，用下面的代码测试一下，就会发现和标准库中的 reduce 一样了。

```
fibonacci.myReduce(0, +) // 12
```

除了求和之外，我们还可以把 fibonacci reduce成一个字符串：

```
let str = fibonacci.myReduce("") { str, num in
    return str + "\(num) "
}
// "0 1 1 2 3 5 "
```

甚至，我们还可以用 reduce 模拟 map 和 filter 的实现：

```
extension Array {
    func myMap2<T>(_ transform: (Element) -> T) -> [T] {
        return reduce([], { $0 + [transform($1)] })
    }

    func myFilter2(_ predicate: (Element) -> Bool) -> [Element] {
        return reduce([], { predicate($1) ? $0 + [$1] : $0 })
    }
}
```

然后，简单测试一下：

```
// [0, 1, 1, 4, 9, 25]
fibonacci.myMap2 { $0 * $0 }
// [0, 2]
fibonacci.myFilter2 { $0 % 2 == 0 }
```

它们的结果和标准库中的 map 和 filter 是一样的。但是，这种看似优雅的写法却没有想象中的那么好。在它们内部的 reduce 调用中，每一次 $0 的参数都是一个新建的数组，因此整个算法的复杂度是 $O(n^2)$，而不再是 for 循环版本的O(n)。所以，这样的实现方法最好还是用来作为理解 reduce 用法的例子。

flatMap

最后，我们来了解 `flatMap` 。简单来说，如果你用在 `map` 中的closure参数不返回一个数组元素，而是也返回一个数组，这样，你就会得到一个数组的数组，但如果你只需要一个一维数组， `flatMap` 就可以派上用场了，而这，也就是flat的含义。先来看一个例子：

```
let animals = ["🐱", "🐶", "🐰", "🐼"]
let ids = [1, 2, 3, 4]
```

假设，我们要给 `animals` 数组中的动物都使用 `ids` 中的数字进行编号。一开始，可能我们会写下这样的代码：

```
animals.map { animal in
    return ids.map { id in (animal, id) }
}
```

但如果是这样，由于 `animals.map` 使用的closure参数返回的是一个 `Array` ，而不是单一元素，最终，我们会得到一个“数组的数组”：

```
[[("🐱", 1), ("🐱", 2), ("🐱", 3), ("🐱", 4)], [("🐶", 1), ("🐶", 2), ("🐶", 3), ("🐶", 4)],
[("🐰", 1), ("🐰", 2), ("🐰", 3), ("🐰", 4)], [("🐼", 1), ("🐼", 2), ("🐼", 3), ("🐼", 4)]]
```

但这并不是我们想要的，我们只是需要一个一维数组表示所有的 `(animal, id)` 。此时，就可以让 `flatMap` 派上用场了：

```
animals.flatMap { animal in
    return ids.map { id in (animal, id) }
}
```

这样，我们就能得到期望的内容了：

```
[("🐱", 1), ("🐱", 2), ("🐱", 3), ("🐱", 4), ("🐶", 1), ("🐶", 2), ("🐶", 3), ("🐶", 4),
("🐰", 1), ("🐰", 2), ("🐰", 3), ("🐰", 4), ("🐼", 1), ("🐼", 2), ("🐼", 3), ("🐼", 4)]
```

实际上， `flatMap` 的实现很简单，只要在 `map` 内部的 `for` 循环里，不断把closure参数生成的数组的内容，添加到要返回的结果里就好了：

```
extension Array {
    func myFlatMap<T>(_ transform: (Element) -> [T]) -> [T] {
        var tmp: [T] = []

        for value in self {
            tmp.append(contentsOf: transform(value))
        }

        return tmp
    }
}
```

用下面的代码测试，得到的结果，应该和之前使用 `flatMap` 是一样的：

```
animals.myFlatMap { animal in
    return ids.map { id in (animal, id) }
}
```

What's next?

至此，我们对Swift中 `Array` 的讨论，就结束了。从最基本的用法，到基于函数式编程的各种实践，我们应该对 `Array` 有一个比较全面的认识了。在下一节中，我们开始和大家介绍Swift中的另一类集合： `Dictionary` ，它是一个用来表示“键值对”的无序集合。

⏮ 通过closure参数化对数组元素的变形操作
(https://www.boxueio.com/series/collection-types/ebook/127)

和Dictionary相关的基础知识 ⏭
(https://www.boxueio.com/series/collection-types/ebook/129)

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/)