Swift 3 Collections

理解Array和NSArray的差异
(https://www.boxueio.com/series/collection-types/ebook/125)

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)

# 用Swift的方式使用Array

Back to series (/series/collection-types)

字号
字号
默认主题
金色主题
暗色主题

尽管Swift Array 尽可能为你提供了所有你可能习惯的使用数组的方式。但这不意味着所有的方式都是被Swift开发者认同或喜爱的方法。特别是，如果你仔细观察 Array 提供的API，就能从中发现一些有意思的事情。首先，我们就从通过下标访问数组元素说起。

## 绝大多数时候，其实你不需要[]

对于下标访问数组元素这种老旧的形式，Swift的开发者应该是不太喜欢的。为了避免你这么做，他们甚至在Swift语言中去掉了传统C风格 `for initial; condition; step` 循环。的确，当你对数组使用这种循环时，下标是一定会在循环内出场的。

另外一个他们不喜欢下标操作符的理由是，对于 `array[index]` 这样的访问，甚至都没有使用optional来保护越界的情况。通过下面的代码可以看到：

```
let a = [1, 2, 3]
type(of: a[1]) // Int.type
```

`a[1]` 的类型是 `Int` ，而不是 `Optional<Int>` ，这说明什么呢？你必须小心翼翼的使用index来访问 `Array` 中的元素，一旦index的值不正确，你就需要承担运行崩溃的严重后果。

那么，为什么要对 [] 如此冷漠呢？因为当我们把基于连续内存中的一组值进一步抽象成一个数组集合之后，下标这种方式带着太多C语言中和内存访问相关的历史气息。而我们应该把注意力更多的放在我们要解决的各种问题上。例如：

当我们想访问数组中的每一个元素时：

```
a.forEach { print($0) }
// or
for value in a {}
```

当我们要获得数组中每一个元素的索引和值时：

```
for (index, value) in a.enumerated() {}
```

当我们要查找数组中元素的位置时（例如，查找等于1的元素的索引）：

```
a.index { $0 == 1 }
```

`index` 会返回一个 `Optional<Int>` ，当要查找的元素存在时，就返回该元素的索引，否则，就返回 `nil` 。

当我们要过滤数组中的某些元素时（例如，去掉所有偶数）：

```
a.filter { $0 % 2 == 0 }
```

当然，在下个视频中我们会专门和大家分享Swift `Array` 常用操作的惯用形式。但至少现在，你应该已经感受到了，当你要完成特定的操作时，Swift一定有比直接使用下标更具表现力和安全的写法。

话又说回来，给 [] 添加optional保护也不能解决安全问题，因为一旦你force unwrapping一个optional，就有可能会带来一连串的force unwrapping。这不仅看上去不美观，从代码表现的含义上来说，既然已经准备好要为结果全权负责了，又何必要再让你多执行一步force unwrapping呢。

## 一些安全周到的方法

和 [] 的高风险形成鲜明对比的是，对于那些可以生成优秀代码的方法，Swift则考虑的面面俱到。例如：

访问数组中第一个和最后一个元素的 `first` 和 `last` 属性，当 `Array` 为空时，它们的值都是 `nil` ：

```
a.first // 1
a.last  // 3
type(of: a.first) // Optional<Int>.Type
```

另外一个值得一提的是在 Array 末尾删除元素。Swift为这个动作提供了两个API：

- removeLast ，你需要自行确保数组中有元素，否则会引发运行时错误；
- popLast ，如果数组为空，会返回 nil ；

为什么要如此呢？一个最通俗的解释就是，为了表意更清晰的代码。

当你基于 Array 实现诸如栈这样后入先出的数据结构时，弹出一个元素并判断是否为空是一个常规的操作，所以 popLast 返回了一个optional。而对于更一般的“删除数组中最后一个元素”这样的行为，Swift认为，这没有任何更具体的使用场景，你应该自己对这样的“低级操作”负责。

## What's next?

在这一节，我们通过一些API的设计和行为模式，感性的体验了Swift在让 Array “更容易用对，而不容易用错”这个原则上的良苦用心。在接下来的几部分内容中，我们将通过一些更具体的例子，来了解如何用更现代化的方式，使用Swift中的 Array 对象。

◀ 理解Array和NSArray的差异
(https://www.boxueio.com/series/collection-types/ebook/125)

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)