Swift 3 Collections

和Array相关的基础知识 (https://www.boxueio.com/series/collection-types/ebook/124)

用Swift的方式使用Array (https://www.boxueio.com/series/collection-types/ebook/126)

理解Array和NSArray的差异

Back to series (/series/collection-types)

字号
字号
默认主题
金色主题
暗色主题

同样是数组类型，Swift中的 Array 和Foundation中的 NSArray 有着截然不同的语义和用法。在这一节中，我们就来简单了解下这些区别。

按值语义实现的Array

在Swift中， Array 是按照值语义实现的，当我们复制一个 Array 对象时，会拷贝整个 Array 的内容：

```
var a = [1, 2, 3] // [1, 2, 3]
let copyA = a     // [1, 2, 3]

a.append(4)
// a  [1, 2, 3, 4]
// copyA [1, 2, 3]
// copyA.append(4)   // Compile error
```

上面的代码中，有两点值得说明。

首先，Swift数组是否可以被修改完全是通过 var 和 let 关键字来决定的， Array 类型自身并不解决它是否可以被修改的问题；

其次，从结果可以看到，复制 a 并向 a 添加内容之后， copyA 的内容并不会修改。但是，Swift在复制 Array 时，同样对 Array 的性能有所考量，它使用了copy on write的方式。即如果你仅仅复制了 Array 而不对它修改时，真正的复制是不会发生的，两个数组仍旧引用同一个内存地址。只有当你修改了其中一个 Array 的内容时，才会真正让两个 Array 对象分开。为了看到这个过程，我们先来实现一个方法，把保存 Array 内容的地址变成一个字符串：

```
func getBufferAddress<T>(of array: [T]) -> String {
    return array.withUnsafeBufferPointer { buffer in
        return String(describing: buffer.baseAddress)
    }
}
```

其中， withUnsafeBufferPointer 是 Array 的一个方法，它可以把保存 Array 内容的地址，传递给它的closure参数。在我们的例子里，这个closure只是把 Array 的地址，变成了一个 String 对象。

然后，我们在 a.append(4) 前后，分别观察 a 和 copyA 的内容：

```
getBufferAddress(of: a)
getBufferAddress(of: copyA)

a.append(4)

getBufferAddress(of: a)
getBufferAddress(of: copyA)
```

```
52 getBufferAddress(array: a)                  "Optional(0x0000610000007d160)"
53 getBufferAddress(array: copyA)              "Optional(0x0000610000007d160)"
54
55 a.append(4)                                 [1, 2, 3, 4]
56
57 getBufferAddress(array: a)                  "Optional(0x0000610000280f20)"
58 getBufferAddress(array: copyA)              "Optional(0x0000610000007d160)"
```

就如同图中显示的，只有在给 a 添加内容后，它才被重新分配了内存地址。

了解了Swift Array 之后，我们再来看Foundation中 NSArray 的情况。

按引用语义实现的NSArray

在Foundation中，数组这个类型有两点和Swift Array 是不同的：

- 数组是否可以被修改是通过 NSArray 和 NSMutableArray 这两个类型来决定的；

- NSArray 和 NSMutableArray 都是类对象，复制它们执行的是引用语义；

当把这两个因素放在一起的时候，Foundation中的“常量数组”这个概念就会在一些场景里表现的很奇怪。因为你还可以通过对一个常量数组的非常量引用去修改它，来看下面的例子：

```
// Mutable array [1, 2, 3]
let b = NSMutableArray(array: [1, 2, 3])
// Const array [1, 2, 3]
let copyB: NSArray = b

// [0, 1, 2, 3]
b.insert(0, at: 0)
// [0, 1, 2, 3]
copyB
```

从上面的代码可以看到，尽管我们在创建 copyB 时，使用了 NSArray ，表明我们不希望它的值被修改，由于这个赋值执行的是引用拷贝，因此，实际上它和 b 指向的是同一块内存空间。因此，当我们修改 b 的内容时， copyB 也就间接受到了影响。

为了在拷贝 NSArray 对象时，执行值语义，我们必须使用它的 copy 方法复制所有的元素：

```
let b = NSMutableArray(array: [1, 2, 3])
let copyB: NSArray = b
let deepCopyB = b.copy() as! NSArray

b.insert(0, at: 0) // [0, 1, 2, 3]
copyB              // [0, 1, 2, 3]
deepCopyB          // [1, 2, 3]
```

从注释中的结果，你就能很容易理解deep copy的含义了。

> 当我们使用 NSArray 和 NSMutableArray 时，Swift中的 var 和 let 关键字就和数组是否可以被修改没关系了。它们只控制对应的变量是否可以被赋值成新的 NSArray 或 NSMutableArray 对象。

What's next?

了解了 Array 和 NSArray 的差别之后，下一节中，我们结合一些常用的场景，来看一下Swift是如何在API的设计层面，向我们推荐 Array 的各种更合理用法的。

◀ 和Array相关的基础知识
(https://www.boxueio.com/series/collection-types/ebook/124)

用Swift的方式使用Array ▶
(https://www.boxueio.com/series/collection-types/ebook/126)

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/)