理解引用语义的自定义类型

返回视频 (/series/understand-ref-types)

理解class类型的各种init方法 (https://www.boxueio.com/series/understand-ref-types/ebook/175)

差异于语法之外的struct和class

作为Swift中的另外一种自定义类型，从语法上来说， `class` 和 `struct` 有很多相似的地方，例如：

```
struct PointValue {
    var x: Int
    var y: Int
}

class PointRef {
    var x: Int
    var y: Int

    init(x: Int, y: Int) {  class 必须实现 init
        self.x = x
        self.y = y
    }
}
```

字号
字号
默认主题
金色主题
暗色主题

你可以看到，它们都可以用来自定义类型、都可以有properties，也都可以有methods。因此，单纯从语法上来理解 `class` 是个没什么意义的事情。在之前我们也说过，作为Swift中的引用类型， `class` 表达的是一个具有明确生命周期的对象，我们需要关注这类内容的“生死存亡”，而值类型，我们更多关注的，就真的只是它的值而已。

接下来，作为这一章的开始，我们就通过一些例子，来感受下引用类型和值类型的差异。

引用类型必须明确指定init方法

首先，Swift并不会为 `class` 自动生成默认的 `init` 方法。如果我们不定义它，Swift编译器会报错。因此，无论多么简单的 `class` ，我们至少要为它定义一个初始化其所有属性的 `init` 方法。虽然有时候这样做很无聊，但是我们没有其它的选择。

为什么要如此呢？刚才我们说过了， `class` 并不简单表达一个“值”的概念。Swift要求我们明确通过 `init` 方法说明“打造”一个对象的过程。相反， `struct` 表达一个自定义的“值”，在没有特别说明的情况下，一个值的初始化当然是把它的每一个member都按顺序初始化。

引用类型关注的是对象本身

其次， `class` 和 `struct` 对“常量”的理解是不同的。我们分别定义一个 `PointRef` 和 `PointValue` 的常量：

```
let p1 = PointRef(x: 0, y: 0)
let p2 = PointValue(x: 0, y: 0)
```

同样是常量，当我们修改 `p2` 的属性时，编译器会报错： `p2 is a let constant` ：

```
p2.x = 10 // Compile time error
```

但是，我们却可以修改 `p1` ：

```
p1.x = 10 // OK
```

这是因为， `p2` 作为一个值类型，常量的意义当然是：“它的值不能被改变”。但是 `p1` 作为一个引用类型，常量的意义则变成了，它可以修改自身的属性，但不能再引用其他的 `PointRef` 对象。如果我们尝试让 `p1` 引用另外一个 `PointRef` 对象，就会发生下面的错误：

```
p1 = PointRef(x: 1, y: 1) // Compile time error
```

以上就是引用类型代表的“对象”和值类型代表的“值本身”在语义上的差别。而这种差别，还体现在了对它们各自进行赋值之后的表现上：

```
var p3 = p1
var p4 = p2
```

这之后，当我们使用 === 比较 p1 和 p3 的时候，得到的结果是 true：

```
p1 === p3 // true
```

并且，当我们修改了 p3 之后， p1 的值，会一并修改：

```
p3.x = 10
p1.x // 10
```

但是，当我们修改了一个值类型时，却并不会这样：

```
p4.x = 10
p2.x // 0
```

了解了引用和值在语义上的差别之后，我们继续来看这个差别在其各自的方法中，带来的差异。

引用类型默认是可以修改的

由于引用类型关注的是其引用的对象，而不是对象的值。因此，它的方法默认是可以修改对象属性的。例如：

```
class PointRef {
    // ...

    func move(to: PointRef) {
        self.x = to.x
        self.y = to.y
    }
}
```

但是，在之前我们讨论值类型的内容里，已经提到过了，对于 PointValue 来说 move 必须用 mutating 来修饰：

```
struct PointValue {
    // ...

    mutating func move(to: PointValue) {
        self.x = to.x
        self.y = to.y
    }
}
```

> 所以，修改一个 struct 的本意，实际上是你需要一个全新的值。

最后，还有一点要说明的是，在 PointValue 里，我们可以直接给 self 赋值：

```
mutating func move(to: PointValue) {
    self = to
}
```

编译器知道对一个值类型赋值就是简单的内存拷贝，因此，他会自动用 to 的每一个属性设置 self 的对应属性。但是，对于一个引用类型来说，你却不能这样：

```
class PointRef {
    // ...
    func move(to: PointRef) {
        self = to // !! Compile time error !!
    }
}
```

在 class 的方法里， self 自身是一个常量，我们不能直接让它引用其它的对象。

What's next?

以上就是我们这一节的内容。理解 `class` 这种类型要表达的语义，是可以正确使用它的前提。作为一个有明确生命周期的对象，创建它的 `init` 方法要比 `struct` 类型复杂的多。而理解这套略显复杂的规则，又是我们可以正确使用 `class` 类型的基础。在接下来的两个话题中，我们就深入其中，去了解可以用于 `class` 类型的 `init` 家族方法。

返回视频 (/series/understand-ref-types)

理解class类型的各种init方法 (https://www.boxueio.com/series/understand-ref-types/ebook/175)

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/)