理解引用语义的自定义类型

继承和多态并不是解决问题的唯一方式
(https://www.boxueio.com/series/understand-ref-types/ebook/181)

使用unowned和weak处理reference cycle
(https://www.boxueio.com/series/understand-ref-types/ebook/183)

Reference cycle是如何发生的?

Back to series (/series/understand-ref-types)

字号
字号
默认主题
金色主题
暗色主题

虽然Swift使用ARC（Automatic Reference Counting）为我们打理内存，这并不代表它面对任何情况都足够聪明。尤其是当对象之间存在相互引用的时候，更是容易由于reference cycle导致内存无法释放。当然，这并非我们本意，只是有时这样的问题发生的不甚明显。作为这个话题的开始，我们先用最直观的方式，来了解reference cycle是如何发生的。

从值类型的内存管理说起

让我们先从Swift中的值类型说起，这类对象用起来很安全，无论如何，值类型对象之间都不会发生引用循环。例如，我们定义一个表示人的 `struct`，它有两个属性，一个表示姓名，一个表示他的所有朋友：

```
struct Person {
    var name: String
    var friends: [Person]
}
```

然后，下面的代码会发生引用循环么？

```
var mars = Person(name: "Mars", friends: [])
mars.friends = [mars]
```

答案是并不会，由于 `struct` 是一个值类型，我们放到数组中的，实际上是 `mars` 的一个拷贝，而并不是 `mars` 对象自身，也就是说，此时 `mars` 的内容是这样的：

```
// name: "Mars", friends: [ name: "Mars", friends: [] ]
```

因此，我们无须关心值类型对象的引用循环问题，这类问题只发生在引用类型的对象之间。只有引用类型的对象，才接受ARC机制的管束。那么，ARC究竟是如何工作的呢？

ARC是如何工作的?

首先，把之前的 `struct Person` 改成 `class Person`：

```
class Person {
    let name: String

    init(name: String) {
        self.name = name
        print("\(name) is being initialized.")
    }

    deinit {
        print("\(name) is being deinitialized.")
    }
}
```

为了观察到对象的初始化和回收，我们在 `init` 以及 `deinit` 方法中，向控制台打印了一条消息。然后，观察下面代码的的执行过程：

```
var ref1: Person? = Person(name: "Mars")
// count = 1
// Mars is being initialized.
```

这时， `Person` 对象就有了一个strong reference `ref1`，它的引用计数是1，并且，我们会在控制台看到*Mars is being initialized.*的消息。然后，我们把它赋值给另外一个 `Person?`：

```
var ref2: Person? = ref1
// count = 2
```

此时， ref1 和 ref2 就指向了相同的 Person 对象，这个对象的引用计数就变成了2。然后，我们让 ref1 为 nil：

```
ref1 = nil
// count = 1
```

Person 对象的引用计数就又变回了1，因此它还不会被ARC回收掉。最后，我们让 ref2 也等于 nil：

```
ref2 = nil
// count = 0
// Mars is being deinitialized.
```

这时， Person 对象就没有任何strong reference了。它的引用计数会降为0，一旦如此，ARC就会立即回收掉 Person 对象。于是，我们就能在控制台看到*Mars is being deinitialized.*的提示了。

了解了ARC的机制之后，我们来看引用循环究竟是如何发生的。

理解循环引用是如何发生的

为了演示这个过程，我们给 Person 类新添加一个属性，表示他租住的公寓：

```
class Person {
    let name: String
    var apartment: Apartment?

    init(name: String) {
        self.name = name
        print("\(name) is being initialized.")
    }

    deinit {
        print("\(name) is being deinitialized.")
    }
}
```

由于并不是所有人都住公寓，于是，它是一个 Apartment? 。然后，我们来实现这个 Apartment：

```
class Apartment {
    let unit: String
    var tenant: Person?

    init(unit: String) {
        self.unit = unit
        print("Apartment \(unit) is being initialized.")
    }

    deinit {
        print("Apartment \(unit) is being deinitialized.")
    }
}
```

这里， unit 表示房间号， tenant 表示房客，由于不一定每个房间都出租，所有它是一个 Person? 。然后，我们分别创建一个 Person 和 Apartment：

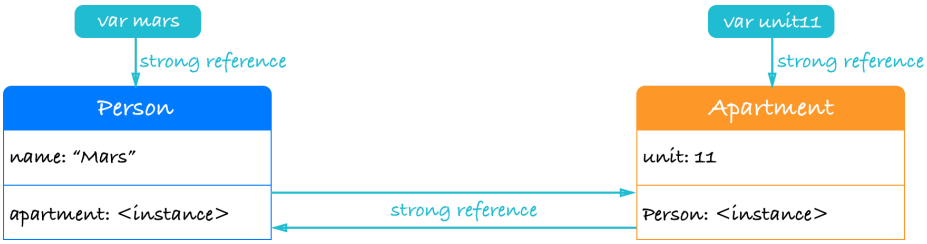
```
// Mars is being initialized
// count = 1
var mars: Person? = Person(name: "Mars")
// Apartment 11 is being initialized
// count = 1
var unit11: Apartment? = Apartment(unit: "11")
```

可以看到，此时 mars 和 unit11 的引用计数都是1。

然后，当 mars 决定租下 unit11 的时候：

```
mars?.apartment = unit11
unit11?.tenant = mars
```

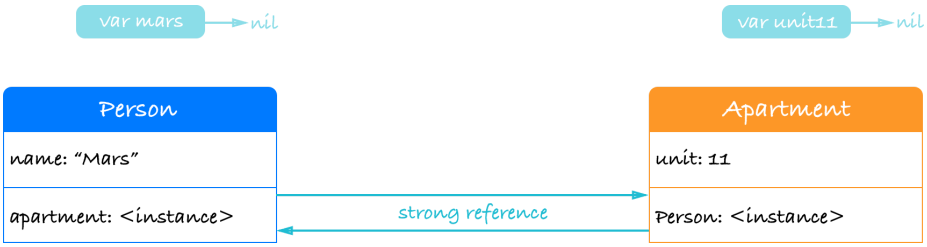
mars 和 unit11 这两个对象就同时多了一个strong reference，此时，它们的引用计数都变成了2。



接下来，当 mars 和 unit11 离开作用域之后，这里我们用 nil 来模拟这个场景：

```
// count = 1
mars = nil
// count = 1
unit11 = nil
```

这时， mars 和 unit11 对象的引用计数会减1，但它们仍旧会被 mars?.apartment 和 unit11?.tenant 保持在内存里。因此它们各自的 deinit 方法并不会被调用，只是此时，我们也再没机会去解决这个问题了。



最终，留给我们的，就是这样个类对象间引用循环的结局。

What's next?

了解了引用循环的成因之后，在接下来的几节中，我们将和大家分享解决的办法。基于引用类型的不同，以及引用类型之间的关系不同，我们采取的方案也不同。

◀ 继承和多态并不是解决问题的唯一方式
(https://www.boxueio.com/series/understand-ref-types/ebook/181)

使用unowned和weak处理reference cycle ▶
(https://www.boxueio.com/series/understand-ref-types/ebook/183)

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/)