理解引用语义的自定义类型

Reference cycle是如何发生的?
(https://www.boxueio.com/series/understand-ref-types/ebook/182)

容易让人犯错的closure内存管理 - I
(https://www.boxueio.com/series/understand-ref-types/ebook/184)

使用unowned和weak处理reference cycle

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

字号
字号
默认主题
金色主题
暗色主题

为了打破我们在上一节中提到的引用循环，根据属性是否可以为 nil ，我们要采取不同的方式。但它们的思路都是一致的，不要让某种形式的引用增加对象的引用计数就好了。

一方属性可以为nil时，使用weak

在我们的 Person 和 Apartment 例子里，由于 Person.apartment 和 Apartment.tenat 都可以为 nil ，我们只要选择其中一个属性，把它指定为 weak 就可以了。当然，你也可以把它们都指定为 weak ，但这并不必要。

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

    // ...
}
```

例如，我们选择把 tenant 设置为 weak 。然后，同样先创建一个 Person 和 Apartment 对象。然后，让 mars “租下” unit11：

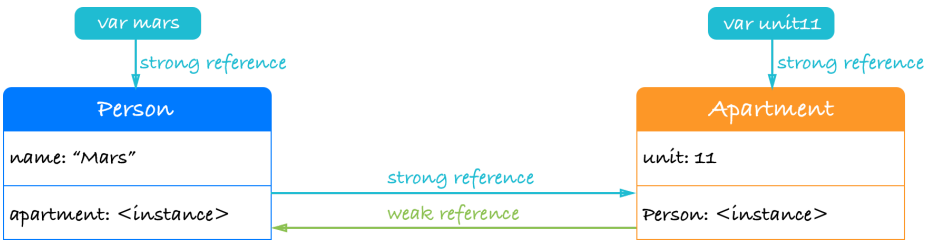
```
var mars: Person? = Person(name: "Mars")
var unit11: Apartment? = Apartment(unit: "11")

mars?.apartment = unit11 // strong reference
```

接下来，当我们把 unit11 的房客设置为 mars 的时候，创建的就不再是strong reference了，而是一个weak reference：

```
unit11?.tenant = mars // weak reference
```

此时，对象间的关系是这样的：



tenant 变成了一个指向 Person 对象的弱引用。这意味着什么呢？我们看下面的代码：

```
mars = nil // Mars is being deinitialized.
unit11?.tenant // nil
```

此时，对象的关系是这样的：

看到了没？当 mars 为 nil 之后，发生了一系列故事：

- 首先， Person 对象就不再有任何strong reference了，于是它会立即被回收掉。于是我们可以在控制台看到*Mars is being deinitialized.*的提示；

- 其次， Person 对象被回收掉之后， apartment 这个strong reference也不存在了， Apartment 对象的引用计数会减1；
- 第三，由于 tenant 是一个weak reference，它的值会被自动设置为 nil 。我们可以访问 unit11?.tenant 来确认这个结果；

于是，只要我们再把 unit11 设置为 nil ，所有的对象就都可以正常回收了：

```
unit11 = nil // Apartment 11 is being deinitialized.
```

一方属性不可以为nil时，使用unowned

虽然我们用 weak 解决了引用循环，但是你一定会想，不一定所有的属性都可以为 nil 啊。如果存在一个不可以为 nil 的属性造成了引用循环，该怎么办呢？为了演示这个过程，我们先给 Person 添加一个属性：

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

    // ...
}
```

当然，并不是所有人都有车，所以 car 的类型是个optional，然后我们来定义 Car ：

```
class Car {
    var owner: Person

    init(owner: Person) {
        self.owner = owner
        print("Car is being initialized.")
    }

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

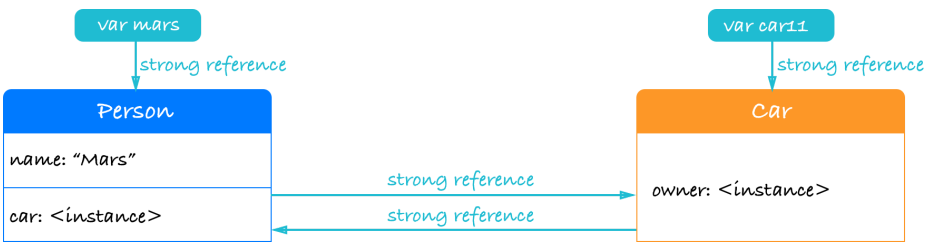
这里，每辆车一定都有主人，所以， owner 是一个普通的类对象属性，它不是一个optional。然后，继续看下面的代码：

```
var mars: Person? = Person(name: "Mars")
// Mars is being initialized.
var car11 = Car(owner: mars!)
// Mars's car is being initialized.
```

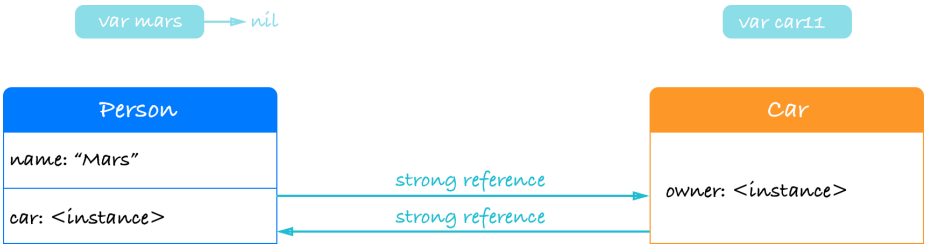
在这里，我们的重点是解决非 nil 对象的循环引用问题，因此，我们去掉了 Apartment 的部分。现在， car11 的主人已经被设置成了 mars 。但是，当 mars 把车“领走”的时候：

```
mars?.car = car11 // Reference cycle
```

我们就创建了一个引用循环。此时， mars 和 car11 的关系是这样的：



即便我们让 mars 为 nil ， car11 变量离开它的作用域， Person.car 和 Car.owner 之间的引用仍旧会把这两个对象保持在内存里。

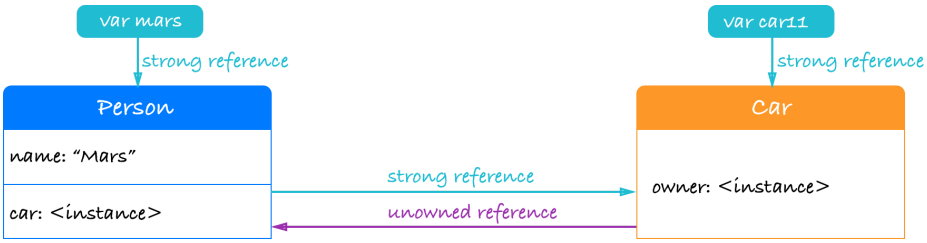


那么，仿照之前的例子，让 car 属性变成一个 weak optional当然也可以解决问题。但这里我们要介绍的，是Swift中的另外一种方式。把不可以为 nil 的属性定义为 unowned 。

```
class Car {
    unowned var owner: Person

    // ...
}
```

此时的对象关系，就变成了这样：



然后，当我们把之前的测试代码放在一个由 if 创造出来的作用域里：

```
if true {
    var mars: Person? = Person(name: "Mars")
    var car11 = Car(owner: mars!)

    mars?.car = car11
    mars = nil
}
// Mars is being deinitialized.
// Car is being deinitialized.
```

当代码离开作用域之后，就可以在控制台看到对象被销毁的提示了。

双方属性都不可以为nil时

了解了 unowned 的用法之后，我们来处理最后一种情况。如果涉及引用循环的两个属性都不可以为 nil 该怎么办呢？和之前的两种情况不同，由于Swift不会把这种属性自动设置为 nil ，因此我们得采取一些曲线救国的办法。

首先我们要明白一件事情，所谓两个属性都不能为 nil 这个假设，是针对类的使用者的，在他们看来，这是两个一定会访问成功的对象。言外之意，我们只要让它们用起来是这个感觉就好了。为了解决这类对象的引用循环问题，我们只能把其中一个属性，用implicit unwrapped optional伪装起来，让它还是一个optional，而另一个属性定义为 unowned 就好了。这类问题的本质，和我们刚才解决的有一方不可以为 nil 是相同的。

而这，也是implicit unwrapped optional另外一个典型的用法（其它的用法在这里(https://boxueio.com/series/optional-is-not-an-option/ebook/147)）。由于类属性一定会在 init 里初始化，所以，在其它的代码里访问这个implicit unwrapped optional包含的对象也是安全的。

What's next?

以上，就是解决类对象之间引用循环问题的场景和方法。大多数时候，只要理解了它们，我们就不会再引发同样的错误。但Swift中的引用类型可不止 `class` 一种，之前我们提到过，closure也是一种引用类型，而我们往往在这种类型是否会引发引用循环的问题上会感到犹豫不决。在下一节里，我们就专门来讨论下closure和内存管理有关的话题。

⏮ Reference cycle是如何发生的?
(https://www.boxueio.com/series/understand-ref-types/ebook/182)

容易让人犯错的closure内存管理 - I ⏭
(https://www.boxueio.com/series/understand-ref-types/ebook/184)

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/)