它叫Optional, 却必不可少

Optional关键实现技术模拟

(https://www.boxueio.com/series/optional-is-not-an-option/ebook/139)

使用guard简化optional unwrapping

(https://www.boxueio.com/series/optional-is-not-an-option/ebook/141)

有哪些常用的optional使用范式

Back to series (/series/optional-is-not-an-option)

字号
字号
默认主题
金色主题
暗色主题

既然optional类型表达了有可能失败这样含义，因此，它最频繁出现的场景当然就是各种条件分支和循环语句。为了简化optional类型在这些场景中的应用，Swift在语法上进行了诸多简化，让optional用起来更加自然。在这一节中，我们就来看一些最常用的optional使用范式。

if let

如果我们要表达“当optional不等于 `nil` 时，则执行某些操作”这样的语义，最朴素的写法，是这样的：

```
let number: Int? = 1

if number != nil {
    print(number!)
}
```

其中， `number!` 这样的写法叫做force unwrapping，用于强行读取optional变量中的值，此时，如果optional的值为 `nil` 就会触发运行时错误。所以，通常，我们会事先判断optional的值是否为 `nil` 。

但这样写有一个弊端，如果我们需要在 `if` 代码块中包含多个访问 `number` 的语句，就要在每一处使用 `number!` ，这显得很啰嗦。我们明知此时 `number` 的值不为 `nil` ，应该可以直接使用它的值才对。为此，Swift提供了 `if let` 的方式，像这样：

```
if let number = number {
    print(number)
}
```

在上面的代码里，我们使用 `if let` 直接在 `if` 代码块内部，定义了一个新的变量 `number` ，它的值是之前 `number?` 的值。然后，我们就可以在 `if` 代码块内部，直接通过新定义的 `number` 来访问之前 `number?` 的值了。

这里用了一个小技巧，就是在 `if let` 后面新定义变量的名字，和之前的optional是一样的。这不仅让代码看上去就像是访问optional自身一样，而且，通常为一个optional的值另取一个新的名字，也着实没什么必要。

除了可以直接在 `if let` 中绑定optional的value，我们还可以通过布尔表达式进一步约束optional的值，这也是一个常见的用法，例如，我们希望number为奇数：

```
if let number = number, number % 2 != 0 {
    print(number)
}
```

我们之前讲到过逗号操作符在 `if` 中的用法，在这里， `number % 2 != 0` 中的number，指的是在 `if` 代码块中新定义的变量，理解了这点，上面的代码就不存在任何问题了。

有了optional的这种用法之后，对于那些**需要一连串有可能失败的行为都成功时才执行的动作**，只要这些行为都返回optional，我们就有了一种非常漂亮的解决方法。

例如，为了从某个url加载一张jpg的图片，我们可以这样：

```
if let url = URL(string: imageUrl), url.pathExtension == "jpg",
    let data = try? Data(contentsOf: url),
    let image = UIImage(data: data) {

    let view = UIImageView(image: image)
}
```

在上面的例子里，从生成 `URL` 对象，到根据url创建 `Data` ，到用 `data` 创建一个 `UIImage` ，每一步的继续都依赖于前一步的成功，而每一步调用的方法又都返回一个optional，因此，通过串联多个 `if let` ，我们就把每一步成功的结果绑定在了一个新的变量上并传递给下一步，这样，比我们在每一步不断的去判断optional是否为 `nil` 简单多了。

while let

除了在条件分支中使用 `let` 绑定optional，我们也可以在循环中，使用类似的形式。例如，为了遍历一个数组，我们可以这样：

```
let numbers = [1, 2, 3, 4, 5, 6]
var iterator = numbers.makeIterator()

while let element = iterator.next() {
    print(element)
}
```

在这里， `iterator.next()` 会返回一个 `Optional<Int>` ，直到数组的最后一个元素遍历完之后，会返回 `nil` 。然后，我们用 `while let` 绑定了数组中的每一个值，并把它们打印在了控制台上。

看到这里，你可能会想，直接用一个 `for...in...` 数组不就好了么？为什么要使用这种看上去有点儿麻烦的 `while` 呢?

实际上，通过这个例子，我们要说明一个重要的问题：**在Swift里， `for...in` 循环是通过 `while` 模拟出来的**，这也就意味着， `for` 循环中的循环变量在每次迭代的时候，都是一个全新的对象，而不是对上一个循环变量的修改：

```
for element in numbers {
    print(element)
}
```

在上面这个 `for...in` 循环里，每一次迭代， `element` 都是一个全新的对象，而不是在循环开始创建了一个 `element` 之后，不断去修改它的值。用 `while` 的例子去理解，每一次 `for` 循环迭代中的 `element` ，就是一个新的 `while let` 绑定。

然而，为什么要这样做呢?

因为这样的形式，可以弥补由于closure捕获变量带来的一个不算是bug，却也有违直觉的问题。首先，我们来看一段JavaScript代码：

```
var fnArray = [];

for (var i in [0, 1, 2]) {
    fnArray[i] = () => { console.log(i); };
}

fnArray[0](); // 2
fnArray[1](); // 2
fnArray[2](); // 2
```

对于末尾的三个 `fnArray` 调用，你期望会返回什么结果呢？我们在每一次 `for...in` 循环中，定义了一个打印循环变量 `i` 的箭头函数。当它们执行的时候，也许你会不假思索的脱口而出：当然是输出0, 1, 2啊。

但实际上，由于循环变量 `i` 自始至终都是同一个变量，在最后调用 `fnArray` 中保存的每一个函数时，它们在真正执行时访问的，也都是同一个变量 `i` 。因此，这三个调用打印出来的值，都是2。类似这样的问题，稍不注意，就会在代码中，埋下Bug的隐患。

因此，在Swift的 `for` 循环里，每一次循环变量都是一个“新绑定”的结果，这样，无论任何时间调用这个closure，都不会出现类似JavaScript中的问题了。

我们把之前的那个例子，用Swift重写一下：

```
var fnArray: [()->()] = []

for i in 0...2 {
    fnArray.append({ print(i) })
}

fnArray[0]() // 0
fnArray[1]() // 1
fnArray[2]() // 2
```

这里，由于变量 `i` 在每次循环都是一个新绑定的结果，因此，每一次添加到 `fnArray` 中的closure捕获到的变量都是不同的对象。当我们分别调用它们的时候，就可以得到捕获到它们的时候，各自的值了。

What's next?

通过这节，我们了解了当把optional用在条件分支和循环中时，Swift在语法上，对optional类型进行的简化。并深入了解了Swift中 `for...in` 循环的实现方式，以及如何通过optional value binding解决closure捕获循环变量带来的一个“小问题”。在下一个节中，我们将对value binding的用法进行一些扩展，来了解如何通过 `guard` 在函数内部去掉多余的unwrapping操作。

Optional关键实现技术模拟

(https://www.boxueio.com/series/optional-is-not-an-option/ebook/139)

使用guard简化optional unwrapping

(https://www.boxueio.com/series/optional-is-not-an-option/ebook/141)

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/)