它叫Optional, 却必不可少

有哪些常用的optional使用范式

(https://www.boxueio.com/series/optional-is-not-an-option/ebook/140)

Chaining and Nil coalescing

(https://www.boxueio.com/series/optional-is-not-an-option/ebook/142)

使用guard简化optional unwrapping

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

通常情况下，我们只能在optional被unwrapping的作用域内，来访问它的值。

理解optional unwrapping的作用域

例如，在下面这个 `arrayProcess` 函数里：

```
func arrayProcess(array: [Int]) {
    if let first = array.first {
        print(first)
    }
}
```

我们只能在 `if` 代码块内部，访问被unwrapping之后的值。但这样做有一个麻烦，就是如果我们要在函数内部的多个地方使用 `array.first` ，就要在每个地方都进行某种形式的unwrapping，这不仅写起来很麻烦，还会让代码看上去非常凌乱。

实际上，面对这种在多处访问同一个optional的情况，更多的时候，我们需要的是一个确保optional一定不为 `nil` 的环境。如果，我们能在一个地方统一处理optional为 `nil` 的情况，就可以在这个地方之外，安全的访问optional的值了。

好在，Swift在语法上，对这个操作进行了支持，这就是 `guard` 的用法：

```
func arrayProcess(array: [Int]) {
    guard let first = array.first else {
        return
    }

    print(first)
}
```

在上面的例子里，我们使用 `guard let` 绑定了 `array.first` 的非 `nil` 值。如果 `array.first` 为 `nil` ，就会转而执行 `else` 代码块里的内容。这样，我们就可以在 `else` 内部，统一处理 `array.first` 为 `nil` 的情况。在这里，我们可以编写任意多行语句，唯一的要求，就是 `else` 的最后一行必须离开当前作用域，对于函数来说，就是从函数返回，或者调用 `fatalError` 表示一个运行时错误。

而这，也是为数不多的，我们可以在value binding作用域外部，来访问optional value的情况。

一个特殊情况

在Swift里，有一类特殊的函数，它们返回 `Never` ，表示这类方法直到程序执行结束都不会返回。Swift管这种类型叫做*uninhabited type*。

什么情况会使用 `Never` 呢？其实并不多，一种是崩溃前，例如，使用 `fatalError` 返回一些用于排错的消息；另一种，是类似 `dispatchMain` 这样，在进程生命周期中一直需要执行的方法。

当我们在返回 `Never` 的函数中，使用 `guard` 时， `else` 语句并不需要离开当前作用域，而是最后一行必须调用另外一个返回 `Never` 的函数就好了。例如下面的例子：

```
func toDo(item: String?) -> Never {
    guard let item = item else {
        fatalError("Nothing to do")
    }

    fatalError("Implement \(item) later")
}
```

在 toDo 的实现里，如果我们没有指定要完成的内容，就在 else 里调用 fatalError 显示一个错误。在这里， fatalError 也是一个返回 Never 的函数。

一个伪装的optional

除了使用真正的optional变量之外，有时，我们还是利用编译器对optional的识别机制来为变量的访问创造一个安全的使用环境。例如，为了把数组中第一个元素转换为 String ，我们可以这样：

```
func arrayProcess(array: [Int]) -> String? {
    let firstNumber: Int

    if let first = array.first {
        firstNumber = first
    } else {
        return nil
    }

    // `firstNumber` could be used here safely
    return String(firstNumber)
}
```

在上面的代码里，有两点值得说明：

首先，我们使用了Swift中延迟初始化的方式，在 if let 中，才初始化常量 firstNumber ；
其次，从程序的执行路径分析，对于 firstNumber 来说，要不我们已经在 if let 中完成了初始化；要不，我们已经从 else 返回。因此，只要程序的执行逻辑来到了 if...else... 之后，访问 firstNumber 就一定是安全的了。

实际上，Swift编译器也可以识别这样的执行逻辑。 firstNumber 就像一个伪装的optional一样，在 if let 分支里被初始化成具体的值，在 else 分支里，被认为值是 nil 。因此，在 else 代码块之后，就像在之前 guard 语句之后一样，我们也可以认为 firstNumber 一定是包含值的，因此安全的访问它。

What's next?

以上，就是在各种不同的作用域，访问optional unwrapping结果的话题。在下一节，我们来看另外一种常见的场景：如何把会返回optional的多个方法调用串联起来。

◀ 有哪些常用的optional使用范式
(https://www.boxueio.com/series/optional-is-not-an-option/ebook/140)

Chaining and Nil coalescing ▶
(https://www.boxueio.com/series/optional-is-not-an-option/ebook/142)

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/)