它叫Optional, 却必不可少

为什么"哨兵值"没有解决错误处理问题

(https://www.boxueio.com/series/optional-is-not-an-option/ebook/138)

有哪些常用的optional使用范式

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

Optional关键实现技术模拟

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"哨兵值"不是一个好办法，又改怎么办呢？其实我们在上一节已经无意间提到了一个思路：让编译器强制我们处理可能发生错误的情况。为了做到这点，我们得满足下面这几个条件：

- 首先，作为一个函数的返回值，它仍旧得是一个独立的类型；
- 其次，对于所有成功的情况，这个类型得有办法包含正确的结果；
- 最后，对于所有错误的情况，这个类型得有办法用一个和正确情况类型不同的值来表达；

做到这些，当我们把一个错误情况的值用在正常的业务逻辑之后，编译器就可以由于类型错误，给我们予以警告了。

让编译器强制你处理错误的情况

说到这，我们应该就有思路了，一个包含两个 case 的 enum 正是解决这个问题的完美方案：

```
enum Optional<T> {
    case some(T)
    case none
}
```

由于我们并不知道函数要返回的正确结果的类型，因此，我们把 Optional 定义成了一个泛型类型。这样：

- 我们就可以让有可能发生错误的函数统一返回一个 Optional ，它仍旧是一个独立的类型；
- 对于所有成功的情况，我们用 case some ，并且把成功的结果保存在associated value里；
- 对于所有错误的情况，我们用 case none 来表示；

然后，我们可以给 Array 添加一个和 std::find 类似的方法：

```
extension Array where Element: Equatable {
    func find(_ element: Element) -> Optional<Index> {
        var index = startIndex

        while index != endIndex {
            if self[index] == element {
                return .some(index)
            }

            formIndex(after: &index)
        }

        return .none
    }
}
```

在 find 的实现里，它有两个退出函数的路径。当在 Array 中找到参数时，就把对应的 Index 作为 .some 的associated value并返回；否则，当 while 循环结束时，就返回 .none 。这样，当我们用 find 查找元素位置时：

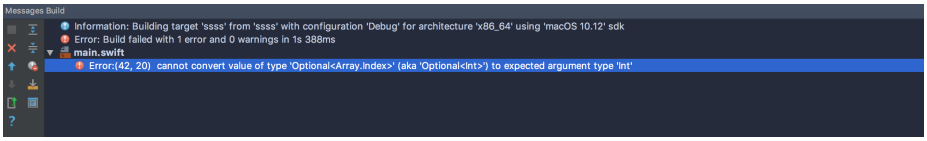
```
var numbers = [1, 2, 3]
let index = numbers.find(4)

print(type(of: index)) // Optinal<Int>
```

index 的类型就会变成 Optional<Int> ，于是，当我们尝试把这个类型传递给 remove(at:) 时：

```
numbers.remove(at: index) // !!! Compile time error !!!
```

就会直接看到一个编译器错误：



为了使用 index 中的值，我们只能这样：

```
switch index {
    case .some(let index):
        numbers.remove(at: index)
    case .none:
        print("Not exist")
}
```

看到了么？只要会发生错误的函数返回 Optional ，编译器就会强制我们对调用成功和失败的情况明确分开处理。并且，当你看到一个函数返回了 Optional ，从它的签名就可以知道，Hmmm，调用它有可能会发生错误，我得小心处理。

实际上，你并不需要自己定义这样的 Optional 类型，Swift中的optional变量就是如此实现的，因此，当让 find 直接返回一个 Index optional时：

```
func find(_ element: Element) -> Index? {
    // ...
}
```

代码一样是可以通过编译的。

理解Swift对optional类型进行的简化处理

Optional作为Swift中最重要的语言特性之一，为了避免让你每次都通过 .some 和 .none 来处理不同的情况（毕竟，这是optional的实现细节），Swift在语法层面对这个类型做了诸多改进。

首先，optional包含的 .some 关联值会在必要的时候，被自动升级成optional；而 nil 字面值则会被转换成 .none 。因此，我们之前的 find 可以被实现成这样：

```
func find(_ element: Element) -> Index? {
    var index = startIndex

    while index != endIndex {
        if self[index] == element {
            return index // Simplified for .some(index)
        }

        formIndex(after: &index)
    }

    return nil // Simplified for .none
}
```

注意 find 中的两个 return 语句，你就能理解从字面值自动升级到optional的含义了。实际上 Array 你也无需自己实现这样的 find ， Array 中自带了一个 index(of:) 方法，它的功能和实现方式，和 find 是一样的。

其次，在 switch 中使用optional值的时候，我们也不用明确使用 .some 和 .none ，Swift同样做了类似的简化：

```
switch index {
    case let index?:
        numbers.remove(at: index)
    case nil:
        print("Not exist")
}
```

我们可以用 case let index? 这样的形式来简化读取 .some 的关联值，用 case nil 来简化 case .none 。

What's next?

以上就是这节的内容，我们从 enum 在类型上的多样性开始，利用编译器强制我们对函数返回的成功和错误结果分开处理。并以此，模拟了 Optional<T> 类型的实现。最后，我们还了解了Swift中，字面值和 nil 自动提升的规则。在下一节中，我们将进一步了解optional类型在Swift中的常用使用范式。

⏮ 为什么"哨兵值"没有解决错误处理问题

(https://www.boxueio.com/series/optional-is-not-an-option/ebook/138)

有哪些常用的optional使用范式 ⏭

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

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/)