Swift中的异常和错误处理

Swift中的错误是如何映射到NSError的?
(https://www.boxueio.com/series/error-handling/ebook/201)

使用defer以及串联either type
(https://www.boxueio.com/series/error-handling/ebook/203)

如何处理closure参数会发生的错误?

Back to series (/series/error-handling)

字号
字号
默认主题
金色主题
暗色主题

在Swift里，通过一个函数类型的参数来定制函数的行为我们已经很熟悉了，它可以帮助我们对不同层次的业务逻辑进行更清晰的分层和抽象。但一直以来，我们使用的closure都有一个特点，就是它们不会“抛出”错误。但情况并不总是如此，当函数类型的参数有可能返回错误时，针对不同的应用场景，会有不同的处理方法。

同步执行的closure参数

为了了解它们的用法，我们先给 Car 一个表示编号的属性：

```
struct Car {
    var fuelInLitre: Double
    var no: String

    // ...
}
```

然后，把燃料不足的错误添加一些详细信息：

```
enum CarError: Error {
    case outOfFuel(no: String, fuelInLitre: Double)
}
```

这样，我们就可以知道车辆具体的燃料情况了。最后，我们把车辆启动的方法进行相应的修改，并把检测项目改成调用 start()：

```
/// - Throws: `CarError` if the car is out of fuelInLitre
func start() throws -> String {
    guard fuelInLitre > 5 else {
        throw CarError.outOfFuel(no: no, fuelInLitre: fuelInLitre)
    }

    return "Ready to go"
}

func selfCheck() throws -> Bool {
    _ = try start()

    return true
}
```

现在，假设我们有1000台车等待启动检测：

```
var vwGroup: [Car] = []

(1...1000).forEach {
    let amount = Double(arc4random_uniform(70))
    vwGroup.append(Car(fuelInLitre: amount, no: "Car-\($0)"))
}
```

这里，我们生成了一个包含1000个 Car 对象的 Array ，并用随机数给其中的每台车加了油。接下来，为了逐台启动车辆进行检测，我们给 Sequence 添加了一个 extension：

```
extension Sequence {
    func checkAll(by rule:
        (Iterator.Element) -> Bool) -> Bool {

        for element in self {
            guard rule(element) else { return false }
        }

        return true
    }
}
```

想法很简单，通过一个接受 Array 元素类型，并返回 Bool 的函数检查 Array 中的每一个元素，只有所有元素都返回 true 时，最终结果才是 true ，否则就返回 false 。在这个例子里，函数参数 rule 对 checkAll 来说，就是我们说到的同步执行的closure。

但当我们尝试执行 checkAll 的时候，却会遇到点儿小麻烦：

```
_ = vwGroup.checkAll(by: {
    try $0.selfCheck()
})
```

编译器会提示我们不能把一个 throws 方法用在一个不会“抛出”错误的closure里，怎么办呢？我们得把 checkAll 方法做一些修改。第一步，当然是让closure参数可以“抛出错误”：

```
func checkAll(by rule:
    (Iterator.Element) throws -> Bool) -> Bool {

    for element in self {
        // Still error here
        guard try rule(element) else { return false }
    }

    return true
}
```

但这样仍旧不行，编译器会提示我们没有在 checkAll 的实现里，处理 try rule(element) 返回的错误。因此，我们还需要一种方法，表达“只有当 rule 抛出错误时， checkAll 才会抛出错误”这样的概念，而这，就是Swift中 rethrows 关键字的用途：

```
 func checkAll(by rule:
    (Iterator.Element) throws -> Bool) rethrows -> Bool {

    for element in self {
        guard try rule(element) else { return false }
    }

    return true
}
```

这样就表示， checkAll 自身的实现是安全的，调用它是否安全，取决于传递给它的 closure 参数。至此，处理同步调用closure参数返回错误的问题就搞定了，我们可以用下面代码进行检测下：

```
do {
    _ = try vwGroup.checkAll(by: {
        try $0.selfCheck()
    })
}
catch let CarError.outOfFuel(no, fuelInLitre) {
    print("\(no) is out of fuel. Current: \(fuelInLitre)L")
}
```

就会在控制台看到类似"Car-5 is out of fuelInLitre. Current: 1.0L"这样的提示了。

异步执行的closure参数

接下来，我们看另外一个场景，函数的closure参数是被异步调用的，在这种情况下，我们之前的方案就不那么好用了。为了演示这个场景，我们给 Car 添加一个 osUpdate 方法：

```
struct Car {
    // ...
    func osUpdate(postUpdate: @escaping (Int) -> Void) {
        DispatchQueue.global().async {
            // Some update staff
            let checksum = 200

            postUpdate(checksum)
        }
    }
}
```

它有一个closure参数 `postUpdate` ， `postUpdate` 接受一个整数参数，表示更新后的校验码。这里为了示意，我们只是硬编码了200。由于更新涉及到下载，解压缩和文件IO，我们把它放在了一个单独的线程中完成，等操作完成之后，我们我们调用 `postUpdate` 方法通知调用者。因此，在这个例子里， `postUpdate` 就是我们提到的异步执行的closure。

然后，我们用下面的代码尝试更新一台车的OS：

```
vwGroup[0].osUpdate(postUpdate: {
    if $0 == 200 {
        print("Starting Car OS...")
    }
})

sleep(1)
```

如果没有错误，我们就能在控制台看到 `Starting Car OS...` 的提示了。但通常，实际的情况并不这么简单，在更新的过程中，网络有可能中断、下载有可能失败、文件IO有可能发生错误。因此，我们不一定可以得到一个校验码。为了表达错误，一个最简单的方式，就是让 `postUpdate` 接受一个 `Int?` ：

```
func osUpdate(postUpdate: @escaping (Int?) -> Void) {
    // ...
}
```

然后，我们就要这样来更新系统：

```
vwGroup[0].osUpdate(postUpdate: {
    if let checksum = $0, checksum == 200 {
        print("Starting Car OS...")
    }
})
```

但显然，一个 `nil` 远不足以表达在更新过程中可能发生的各种错误。但这时，如果我们像之前一样把closure参数变成 `throws` ，也同样不解决问题：

```
func osUpdate(postUpdate: @escaping (Int) throws -> Void) {
    // ...
}
```

这表示， `postUpdate` 自身会“抛出”错误，而并不是说它接受一个可以表示错误结果的参数。于是，你可能会想，那我就把 `throws` 再往里挪一层呗：

```
enum CarError: Error {
    case outOfFuel(no: String, fuelInLitre: Double)
    case updateFailed
}

func osUpdate(postUpdate:
    @escaping (() throws -> Int) -> Void) {
        DispatchQueue.global().async {
            // Some update staff
            let checksum = 200

            postUpdate {
                if checksum != 200 {
                    throw CarError.updateFailed
                }

                return checksum
            }
        }
    }
```

虽然在语法上这并无不妥，当你把他写出来之后，自己可能都会觉得并不那么容易理解。然而，对于 osUpdate 的使用者来说，他们的感受同样糟糕：

```
vwGroup[0].osUpdate(postUpdate: {
    (getResult: (() throws -> Int)) in
    do {
        let checksum = try getResult()
    }
    catch CarError.updateFailed {
        print("Update failed")
    }
    catch {
    }
})
```

想必你从未在Apple官方的API中有过如此的开发体验。因此，在异步回调函数中处理错误，也是Swif原生的错误处理机制目前还无法优雅处理的问题。但是，如果我们去掉错误处理机制的语法糖，用一开始的 Result 封装一下执行结果，你立刻就能找回似曾相识的感觉：

```
func osUpdate(postUpdate: @escaping (Result<Int>) -> Void) {
    DispatchQueue.global().async {
        // Some update staff
        let checksum = 400

        if checksum != 200 {
            postUpdate(.failure(CarError.updateFailed))
        }
        else {
            postUpdate(.success(checksum))
        }
    }
}
```

相比之前的实现， Result 版本的逻辑要简单直观的多。这时， osUpdate 用起来，也会有似曾相识的感觉：

```
vwGroup[0].osUpdate(postUpdate: {
    switch $0 {
    case let .success(checksum):
        print("Update success: \(checksum)")
    case let .failure(error):
        print(error.localizedDescription)
    }
})
```

因此，对于异步回调函数的错误处理方式，这样的解决方案也得到了Swift开源社区的认同。很多第三方框架都使用了类似的解决方案。对于 Result<T> ，由于包含了两类不同的值，它也有了一个特别的名字，叫做either type。

What's next?

至此，我们对于如何在Swift中精确的区分和处理每一类不同的错误，就已经有一个比较全面的认识和了解了。下一节，我们来看如何串联调用多个返回either type的函数，以及如何对有可能“抛出”错误的函数调用进行“收尾”工作。

⏮ Swift中的错误是如何映射到NSError的?
(https://www.boxueio.com/series/error-handling/ebook/201)

使用defer以及串联either type ⏭
(https://www.boxueio.com/series/error-handling/ebook/203)

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/)