

☰

错误处理

4.2 ▾

Swift 编程语言 / 错误处理

这是一篇社区协同翻译的文章，你可以点击右边区块信息里的『改进』按钮向译者提交改进建议。

错误处理是响应错误以及从错误中恢复的过程。Swift 提供了在运行时对可恢复错误的抛出、捕获、传递和操作的一等公民支持。

我们不能保证所有的操作都可以全部被执行，也不能保证都可以生成有用的结果。当操作失败时，可以使用可选类型表示缺省值，但是了解导致失败的原因通常是很有用的，这样您的代码就可以做出相应的响应。

举个例子，假设有个任务是从磁盘读取并处理数据。可能有很多种方法导致这个任务失败，比如指定路径下的文件不存在，没有文件的读取权限，或者文件的编码格式不兼容。区分导致错误的不同情况可以让程序解决这些错误，并把解决不了的错误反馈给用户。

注意

Swift 中的错误处理与在 Cocoa 和 Objective-C 中使用的 `NSError` 的错误处理模式互操作。有关 `NSError` 的更多信息，请参阅 [Handling Cocoa Errors in Swift](#)。

表示和抛出错误

在 Swift 中，错误是遵循 `Error` 协议的值。`Error` 是一个空协议，表明遵循该协议的类型可以用于错误处理。

Swift 中的枚举特别适合对一组相关的错误条件进行建模，关联值允许传递有关错误的附加信息。例如，以下是您可能在游戏中操作自动售货机时出现的错误情况：

```
enum VendingMachineError: Error {
    case invalidSelection
    case insufficientFunds(coinsNeeded: Int)
    case outOfStock
}
```

100%

翻译进度

10

分块数量

4

参与人数



Jett 翻译于 5个月前

👍 0

重译

由 Summer 审阅



Jett 翻译于 5个月前

👍 0

重译

由 Aufree 审阅



```
}
```

您可以通过抛出错误表示发生了意外情况，导致正常的执行流程不能继续执行下去。您可以使用 `throw` 语句抛出错误。例如，以下代码通过抛出错误表明自动售货机还需要 5 个硬币：

```
throw VendingMachineError.insufficientFunds(coinsNeeded: 5)
```

处理错误

当抛出错误时，它周围的代码必须负责处理错误。例如，通过尝试替换方案或将错误通知用户来纠正错误。

在 Swift 中有四种处理错误的方式。您可以将错误从函数传递给调用该函数的代码，可以使用 `do` - `catch` 语句处理错误，可以通过可选值处理错误，或者通过断言保证错误不会发生。在下面章节会对每种方式进行详细讲解。

当函数抛出错误时，它会改变程序的流程，因此快速定位问题显得尤为重要。在调用可能引发错误的函数、方法或初始化程序的代码之前，使用 `try` 关键字，或者使用它的变体 `try?` 或 `try!`，在您的代码中定位错误的位置。这些关键字在下面的章节中有详细描述。

注意

在 Swift 中使用 `try`、`catch` 和 `throw` 关键字进行错误处理的语法和其他语言的异常处理差不多。不同于 Objective-C 等语言的异常处理，Swift 中的错误处理不会展开调用堆栈，因为这个过程的计算成本很高。因此，`throw` 语句的性能特性和 `return` 语句的性能特性相差无几。

使用抛出函数传递错误

为了让函数、方法或者初始化程序可以抛出错误，您需要在函数声明的参数后面添写 `throws` 关键字。标有 `throws` 的函数称为抛出函数。如果函数指定了返回类型，则在返回箭头 (`->`) 之前添写 `throws` 关键字。

```
func canThrowErrors() throws -> String
```

```
func cannotThrowErrors() -> String
```



Jett 翻译于 5个月前

👍 0

重译

由 Aufree 审阅



Jett 翻译于 5个月前

👍 0

重译

由 Aufree 审阅

抛出函数将函数中的错误传递给调用该函数的代码。

注意

只有抛出函数才能传递错误。任何在非抛出函数中抛出错误都必须在函数内部进行处理。

在下面的示例中， `VendingMachine` 类有一个 `vend(itemNamed:)` 方法，如果请求的对象不可用，缺货或成本超出当前存款金额，则抛出适当的 `VendingMachineError` 错误。

```
struct Item {
    var price: Int
    var count: Int
}

class VendingMachine {
    var inventory = [
        "Candy Bar": Item(price: 12, count: 7),
        "Chips": Item(price: 10, count: 4),
        "Pretzels": Item(price: 7, count: 11)
    ]
    var coinsDeposited = 0

    func vend(itemNamed name: String) throws {
        guard let item = inventory[name] else {
            throw VendingMachineError.invalidSelection
        }

        guard item.count > 0 else {
            throw VendingMachineError.outOfStock
        }

        guard item.price <= coinsDeposited else {
            throw VendingMachineError.insufficientFunds(coinsNeeded: item.price)
        }

        coinsDeposited -= item.price

        var newItem = item
        newItem.count -= 1
        inventory[name] = newItem

        print("Dispensing \(name)")
    }
}
```

```
}  
}
```

当无法满足购买零食的要求，`vend(itemNamed:)` 方法将使用 `guard` 语句提前退出方法，并抛出适当的错误。因为 `throw` 语句会立即转移程序控制，所以只有当满足所有要求时程序才会出售物品。

因为 `vend(itemNamed:)` 方法会传播它抛出的任何错误，因此调用此方法的任何代码都必须处理错误 --- 使用 `do` - `catch` 语句，`try?` 或 `try!` --- 或继续传播它们。例如，下面示例中的 `buyFavoriteSnack(person:vendingMachine:)` 也是一个可抛出函数，所以 `vend(itemNamed:)` 方法抛出的任何错误都将传播到调用 `buyFavoriteSnack(person: vendingMachine:)` 函数的地方。

```
let favoriteSnacks = [  
    "Alice": "Chips",  
    "Bob": "Licorice",  
    "Eve": "Pretzels",  
]  
  
func buyFavoriteSnack(person: String, vendingMachine: VendingMachine) throws {  
    let snackName = favoriteSnacks[person] ?? "Candy Bar"  
    try vendingMachine.vend(itemNamed: snackName)  
}
```

在这个例子中，`buyFavoriteSnack(person: vendingMachine:)` 函数查找某个人最喜欢的零食并尝试通过调用 `vend(itemNamed:)` 方法购买它。因为 `vend(itemNamed:)` 方法可能抛出错误，所以需要在它前面用 `try` 关键字调用它。

可抛出构造器可以像抛出函数一样传播错误。例如，下面清单中 `PurchasedSnack` 结构的构造器将可抛出函数作为初始化过程的一部分调用，该构造器通过将错误传播给调用者来处理它遇到的任何错误。

```
struct PurchasedSnack {  
    let name: String  
    init(name: String, vendingMachine: VendingMachine) throws {  
        try vendingMachine.vend(itemNamed: name)  
        self.name = name  
    }  
}
```

使用 Do-Catch 处理错误



jihongboo 翻译于 5个月

👍 0

重译

由 Summer 审阅

`do` - `catch` 语句通过运行代码块来处理错误。如果 `do` 子句中的代码抛出错误，它将与 `catch` 子句一一匹配，以确定它们中的哪一个子句可以处理错误。

这是 `do` - `catch` 语句的一般形式：

```
do {  
    try expression  
    statements  
} catch pattern 1 {  
    statements  
} catch pattern 2 where condition {  
    statements  
} catch {  
    statements  
}
```

在 `catch` 之后写一个模式来表明该子句可以处理的错误。如果 `catch` 子句没有模式，则该子句将会匹配所有错误并将错误绑定到名为 `error` 的本地常量。有关模式匹配的更多信息，请参阅 [模式](#)。

例如，以下代码匹配 `VendingMachineError` 枚举的所有三种情况。

```
var vendingMachine = VendingMachine()  
vendingMachine.coinsDeposited = 8  
do {  
    try buyFavoriteSnack(person: "Alice", vendingMachine: vendingMachine)  
    print("Success! Yum.")  
} catch VendingMachineError.invalidSelection {  
    print("Invalid Selection.")  
} catch VendingMachineError.outOfStock {  
    print("Out of Stock.")  
} catch VendingMachineError.insufficientFunds(let coinsNeeded) {  
    print("Insufficient funds. Please insert an additional \$(coinsNeeded) coins.")  
} catch {  
    print("Unexpected error: \$(error).")  
}  
  
// 打印 "Insufficient funds. Please insert an additional 2 coins."
```

在上面的例子中，`buyFavoriteSnack(person:vendingMachine:)` 函数在 `try` 表达式中调用，因为它可能会抛出错误。如果抛出错误，执行会被立即转移到匹配的 `catch` 子句，该子句决定是否允许继续执行。如果没有匹配的模式，则错误会被最终的 `catch` 子句捕获并绑定到本地 `error` 常量。如果没有抛出错误，则执行 `do` 语句中的其余语句。



jihongboo 翻译于 5个月

👍 0

重译

由 Summer 审阅



jihongboo 翻译于 5个月

👍 0

重译

由 Summer 审阅



jihongboo 翻译于 5个月

👍 0

重译

由 Summer 审阅

`catch` 子句不必处理 `do` 子句中抛出的所有可能错误。如果所有 `catch` 子句都没能处理错误，错误会向周围传播。但是，传播的错误必须能被附近的一些作用域处理。在非抛出函数中，封闭的 `do` - `catch` 子句必须处理错误。在可抛出函数中，封闭的 `do` - `catch` 子句或调用者必须处理错误。如果错误传播到顶级作用域而未被处理，则会出现运行时错误。

例如，之前的例子中，任何非 `VendingMachineError` 的错误都会被调用函数捕获：

```
func nourish(with item: String) throws {
    do {
        try vendingMachine.vend(itemNamed: item)
    } catch is VendingMachineError {
        print("Invalid selection, out of stock, or not enough money.")
    }
}

do {
    try nourish(with: "Beet-Flavored Chips")
} catch {
    print("Unexpected non-vending-machine-related error: \(error)")
}

// 打印 "Invalid selection, out of stock, or not enough money."
```

在 `nourish(with:)` 函数中，如果 `vend(itemNamed:)` 抛出一个错误，且这个错误是 `VendingMachineError` 枚举中的其中一个，那么 `nourish(with:)` 将通过打印消息来处理错误。否则，`nourish(with:)` 将错误传播到其调用栈，然后通过通用 `catch` 子句捕获错误。

将错误转换为可选值

你可以使用 `try?` 将错误转换为可选值来处理错误。如果在执行 `try?` 表达式时抛出错误，表达式的值将为 `nil`。例如，在下面的代码中，`x` 和 `y` 具有相同的值和行为：

```
func someThrowingFunction() throws -> Int {
    // ...
}

let x = try? someThrowingFunction()

let y: Int?
do {
```



```
y = try someThrowingFunction()
} catch {
    y = nil
}
```

如果 `someThrowingFunction()` 抛出错误，则 `x` 和 `y` 的值为 `nil`。否则，`x` 和 `y` 的值是函数返回的值。请注意，`x` 和 `y` 是 `someThrowingFunction()` 返回的任何类型的可选项。这里函数返回一个整数，因此 `x` 和 `y` 是可选的整数类型。

当你想以同样的方式处理所有错误时，使用 `try?` 可以帮助你编写出简洁的错误处理代码。例如，以下代码使用多种途径来获取数据，如果所有方法都失败则返回 `nil`。

```
func fetchData() -> Data? {
    if let data = try? fetchDataFromDisk() { return data }
    if let data = try? fetchDataFromServer() { return data }
    return nil
}
```

禁用错误传播

有时你知道可抛出函数或方法实际上不会在运行时抛出错误。在这种情况下，你可以在表达式之前添加 `try!` 来禁用错误传播，并把调用过程包装在运行时断言中，从而禁止其抛出错误。而如果实际运行时抛出了错误，你将收到运行时错误。

例如，以下代码使用 `loadImage(atPath:)` 函数，该函数会加载指定路径下的资源，如果无法加载图像则抛出错误。在这种情况下，由于图片随应用程序一起提供，因此运行时不会抛出任何错误，因此禁用错误传播是合适的。

```
let photo = try! loadImage(atPath: "./Resources/John Appleseed.jpg")
```

指定清理操作

当代码执行到即将离开当前代码块之前，可以使用 `defer` 语句来执行一组语句。无论是因为错误而离开 --- 抑或是因为诸如 `return` 或 `break` 等语句而离开，`defer` 语句都可以让你执行一些必要的清理。例如，你可以使用 `defer` 语句来关闭文件描述符或释放手动分配的内存。

`defer` 语句会推迟执行，直到退出当前作用域。该语句由 `defer` 关键字和稍后要执行的语句组成。延迟语句可能不包含任何将控制转移出语句的代码，例如 `break` 或



jihongboo 翻译于 5个月



0

重译

由 Summer 审阅



jihongboo 翻译于 5个月



0

重译

由 Summer 审阅

`return` 语句，或抛出错误。延迟操作的执行顺序与它们在源代码中编写的顺序相反。也就是说，第一个 `defer` 语句中的代码最后一个执行，第二个 `defer` 语句中的代码倒数第二个执行，依此类推。源代码中的最后一个 `defer` 语句最先执行。

```
func processFile(filename: String) throws {
    if exists(filename) {
        let file = open(filename)
        defer {
            close(file)
        }
        while let line = try file.readline() {
            // Work with the file.
        }
        // 在此关闭（文件），位于语句的末端。
    }
}
```

上面的例子使用 `defer` 语句来确保 `open(_:)` 函数有相应的 `close(_:)` 函数调用。

注意

即使没有涉及错误处理代码，也可以使用 `defer` 语句。

本文中的所有译文仅用于学习和交流目的，转载请务必注明文章译者、出处、和本文链接

我们的翻译工作遵照 [CC 协议](#)，如果我们的工作有侵犯到您的权益，请及时联系我们。

← 上一篇

下一篇 →

👍 点赞





更多职位





讨论数量: 0

发起讨论

☐ 只看当前版本讨论

暂无话题~

兄弟社区

-  Laravel China
-  PythonCaff.com
-  GolangCaff.com
-  VuejsCaff.com

资源推荐

资源推荐

其他信息

-  软件外包
-  商务合作
-  联系站长

由 Summer 设计和编码 ❤️