它叫Optional, 却必不可少

什么时候需要force unwrapping

(https://www.boxueio.com/series/optional-is-not-an-option/ebook/145)

到底该在什么地方使用implicit optional

(https://www.boxueio.com/series/optional-is-not-an-option/ebook/147)

两个调试optional的小技巧

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

尽管前面我们提到了很多使用optional的正确方式，以及列举了诸多不要使用force unwrapping的理由，但现实中，你还是或多或少会跟各种使用了force unwrapping的代码打交道。使用这些代码，就像拆弹一样，稍不留神它就会让我们的程序崩溃。因此，我们需要一些简单易行的方式，让它在跟我们翻脸前，至少留下些更有用的内容。

改进force unwrapping的错误消息

得益于Swift可以自定义操作符的特性，一个更好的主意是我们自定义一个force unwrapping操作符的加强版，允许我们自定义发生运行时错误的消息。既然一个 ! 表示force unwrapping，那我们暂且就定义一个 !! 操作符就好了。它用起来，像这样：

```
var record = ["name": "11"]
record["type"] !! "Do not have a key named type"
```

怎么做呢?

首先，在上面的例子里， !! 是一个中序操作符（infix operator），也就是说，它位于两个操作数中间，我们这样来定义它：

```
infix operator !!
```

其次，我们把它定义为一个泛型函数，因为我们并不知道optional中包含的对象类型。这个函数有两个参数，第一个参数是左操作数，表示我们要force unwrapping的optional对象，第二个参数是右操作数，表示我们要在访问到 nil 时显示的错误消息：

```
func !!<T>(optional: T?,
    errorMsg: @autoclosure () -> String) -> T {
    // TODO: implement later
}
```

最后， !!<T> 的实现就很简单了，成功unwrapping到，就返回结果，否则，就用 fatalError 打印运行时错误：

```
func !!<T>(optional: T?,
    errorMsg: @autoclosure () -> String) -> T {

    if let value = optional { return value }
    fatalError(errorMsg)
}
```

这样，我们上面的 record["type"] 就会得到下面的运行时错误：

```
fatal error: Do not have a key named type: file /var/folders/_l/lkj1p80s7v517p50cg4m0str0000gn/
T/./lldb/9406/playground306.swift, line 109
Current stack trace:
0    libswiftCore.dylib                0x0000000106ed5cc0 swift_reportError + 132
1    libswiftCore.dylib                0x0000000106ef2f50 _swift_stdlib_reportFatalErrorInFile
```

于是，即便发生意外，至少我们也还能够让程序“死个明白”。

进一步改进force unwrapping的安全性

当然，除了在运行时死的明白之外，我们还可以把调试日志只留在debug mode，并在release mode，为force unwrapping到 nil 的情况提供一个默认值。就像之前我们提到过的 ?? 类似，我们来定义一个 !? 操作符来实现这个过程：

```
infix operator !?

func !?<T: ExpressibleByStringLiteral>(
        optional: T?,
        errorMsg: @autoclosure () -> String) -> T {
    assert(optional != nil, errorMsg())
    return optional ?? ""
}
```

在上面的代码里，我们使用 `ExpressibleByStringLiteral` 这个protocol约束了类型T必须是一个 `String`，之所以要做这个约束，是因为我们要为 `nil` 的情况提供一个默认值。

在 `!?` 的实现里， `assert` 仅在debug mode生效，它的执行的逻辑，和我们实现 `!!` 操作符时是一样的。而在release mode，我们直接使用了 `??` 操作符，为 `String?` 提供了一个空字符串默认值。

于是，当我们这样使用 `record["type"]` 的时候：

```
record["type"] !? "Do not have a key named type"
```

我们就只会在debug mode得到和之前同样的运行时错误，而在release mode，则会得到一个空字符串。或者，基于这种方法，我们还可以有更灵活的选择。例如，借助Tuple，我们同时可以自定义 `nil` 时使用的默认值和运行时错误：

```
func !?<T: ExpressibleByStringLiteral>(
    optional: T?,
    nilDefault: @autoclosure () -> (errorMsg: String, value: T)) -> T {

    assert(optional != nil, nilDefault().errorMsg)
    return optional ?? nilDefault().value
}
```

然后，我们的 `record["Type"]` 就可以改成：

```
record["type"] !? ("Do not have a key named type", "Free")
```

这样，在release mode， `record["type"]` 的值，就是“Free”了。理解了这个方式的原理之后，我们就可以使用Swift标准库中提供了Expressible家族，来对各种类型的optional进行约束了：

- ExpressibleByNilLiteral
- ExpressibleByArrayLiteral
- ExpressibleByFloatLiteral
- ExpressibleByStringLiteral
- ExpressibleByIntegerLiteral
- ExpressibleByBooleanLiteral
- ...

这些类型的含义很好理解，我们就不一一举例了。

最后，我们再来看一种特殊的情况，当我们通过optional chaining得到的结果为 `Void?` 时，例如这样：

```
record["type"]?.write(" account")
```

由于Swift并没有提供类似 `ExpressibleByVoidLiteral` 这样的protocol，为了方便调试 `Optional<Void>`，我们只能再手动重载一个非泛型版本的 `!?`：

```
func !?(optional: Void?, errorMsg: @autoclosure () -> String) {
    assert(optional != nil, errorMsg())
}
```

然后，就可以在debug mode调试 `Optional<Void>` 了：

```
record["type"]?
    .write(" account")
    !? "Do not have a key named type"
```

What's next?

以上，就是当我们不得不面对force unwrapping的时候，可以采取的一些折衷的办法。它们至少可以帮我们在收拾残局的时候，提供一些更有用的信息，或者，为app提供更为可靠的运行时环境。但是，我们还是应该尽量避免使用force unwrapping。

接下来，我们要讨论另外一个容易引发困惑的话题，Swift中还有一类可以自动unwrapping的optional，叫做implicitly unwrapped optional，它用上去就跟一个普通的变量一样，但是一旦它的内容为 `nil` ，就会立即跟你翻脸。既然如此危险，为什么还要存在这种特殊的类型呢?

⏮ 什么时候需要force unwrapping

(https://www.boxueio.com/series/optional-is-not-an-option/ebook/145)

到底该在什么地方使用implicit optional ⏭

(https://www.boxueio.com/series/optional-is-not-an-option/ebook/147)