Swift中的异常和错误处理

理解Swift中的错误处理机制
(https://www.boxueio.com/series/error-handling/ebook/199)

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

# NSError是如何桥接到Swift原生错误处理的?

Back to series (/series/error-handling)

字号
字号
默认主题
金色主题
暗色主题

使用Swift编程，一个不可避免的问题，就是和使用Objective-C编写的各种API打交道，这些API大多通过返回 NSError 表达错误信息。当我们进行混编的时候， NSError 是如何与Swift原生的 Error 类型进行交互的呢?

## NSError是如何桥接到Error的?

，我们给之前的项目添加一个用OC编写的类为了了解Foundation中的API是如何桥接到Swift的 Sensor ，表达汽车的传感器：

```
// In Sensor.h
extern NSString *carSensorErrorDomain;

NS_ENUM(NSInteger, CarSensorError) {
    overHeat = 100
};

@interface Sensor: NSObject {
}

+ (BOOL)checkTemperature: (NSError **)error;
@end
```

我们只添加了一个检测水温的方法 checkTemperature:error ，为了模拟不同水温的情况，它的实现里，我们只是随机生成了一个10-130之间的随机数，并在温度超过100摄氏度时，返回 NSError ：

```
// In Sensor.m
NSString *carSensorErrorDomain = @"CarSensorErrorDomain";

@implementation Sensor {
}

+ (BOOL)checkTemperature: (NSError **)error {
    double temp = 10 + arc4random_uniform(120);

    if ((error != NULL) && (temp >= 100)) {
        NSDictionary *userInfo = @{
            NSLocalizedDescriptionKey: NSLocalizedString(
                @"The radiator is over heat", nil),
        };

        *error = [NSError errorWithDomain: carSensorErrorDomain
                                     code: overHeat
                                 userInfo: userInfo];
        return NO;
    }
    else if (temp >= 100) {
        return NO;
    }

    return YES;
}
@end
```

实际上， checkTemperature 的这种声明：

```
+ (BOOL)checkTemperature: (NSError **)error
```

是很多Foundation API都会采取的“套路”。通过一个 BOOL 搭配 NSError ** 来表达API可能返回的各种错误。当 checkTemperature 桥接到Swift后，根据SE-0112 (https://github.com/apple/swift-evolution/blob/master/proposals/0112-nserror-bridging.md)中的描述，它的签名会变成这样：

```
func checkTemperature() throws {
    // ...
}
```

这里要特别说明的是，**只有返回 BOOL 或 nullable 对象，并通过 NSError ** 参数表达错误的OC函数，桥接到Swift时，才会转换成Swift原生的 throws 函数**。并且，由于 throws 已经足以表达失败了，因此，Swift也不再需要OC版本的 BOOL 返回值，它会被去掉，改成 Void 。

理解了这个桥接过程后，我们给 Car 添加一个自检的方法：

```
struct Car {
    // ...

    func selfCheck() throws {
        try Sensor.checkTemperature()
    }
}
```

这里，由于 checkTemperature 是一个 throws 方法，我们调用的时候要使用 try 关键字。并且，由于没有使用 do...catch ， checkTemperature 返回的错误就会被“扔”到 selfCheck 里，因此，它也得是一个 throws 方法，我们可以用这种方式不断向上一级“抛出”错误。

> 如果我们一直这样“抛出”错误，最终，错误就会被传到Swift运行时默认的错误处理方法，结果，当然就是你的App闪退了。

接下来，在调用 start() 方法前，我们先对 Car 对象进行自检：

```
do {
    try vw.selfCheck()

    // ...
}
```

现在，问题来了。既然 selfCheck() 有可能“抛出”错误，但这个错误是 NSError 桥接而来的，我们应该如何 catch 呢?

这个问题的答案，从某种程度上说，取决于API返回的 NSError 是如何在OC中定义的。而按照我们现在这样的定义方式， selfCheck() 会返回一个 NSError ，我们只能这样来 catch ：

```
do {
    try vw.selfCheck()
} catch let error as NSError
    where error.code == CarSensorError.overHeat.rawValue {
    // CarSensorErrorDomain
    print(error.domain)
    // The radiator is over heat
    print(error.userInfo["NSLocalizedDescription"] ?? "")
}
```

虽然可以正常工作，但你看到了，为了匹配到 selfCheck 的错误，我们得先进行一步类型转换，再检查转换结果的 code 是否相等，这显然太啰嗦了。为什么不能把 CarSensorError 变成一个可以直接 catch 的对象，并让它包含OC中所有错误信息呢?

按照SE-0112 (https://github.com/apple/swift-evolution/blob/master/proposals/0112-nserror-bridging.md)中的设计， CarSensorError 的确应该是可以直接 catch 的。为此，Objective-C中还专门加了一个宏： NS_ERROR_ENUM 。按照这份proposal中的定义，这个宏会在Swift中引入一个和OC ENUM同名的结构，这个结构中包含了和 NSError 中相同的信息。

但在Xcode 8.2.1中，这个宏却还不能正常使用。为了试验(SE-0112) (https://github.com/apple/swift-evolution/blob/master/proposals/0112-nserror-bridging.md)中定义的行为，我们得自己把它添加进来。

在Sensor.h中，添加下面的宏定义：

```
// In Sensor.h
#if __has_attribute(ns_error_domain)
    #define NS_ERROR_ENUM(type, name, domain) \
        _Pragma("clang diagnostic push") \
        _Pragma("clang diagnostic ignored \"-Wignored-attributes\"") \
        NS_ENUM(type, __attribute__((ns_error_domain(domain))) name) \
        _Pragma("clang diagnostic pop")
#else
    #define MY_ERROR_ENUM(type, name, domain) NS_ENUM(type, name)
#endif
```

由于Swift 2.3中， `ns_error_domain` 并没有任何语义，因此，我们先进行了兼容性判断，仅在Swift 3的环境中定义了 `NS_ERROR_DOMAIN` ，本质上，这就是一个特殊的 `NS_ENUM` 。在 `NS_ENUM` 前后的三个 `_Pragma` 用于在处理宏定义时，临时关闭警告，它们并不影响编译结果。

我们定义的 `NS_ERROR_ENUM` 有三个参数：

- `type`： `ENUM` 中值的类型；
- `name`： `ENUM` 类型的名字；
- `domain`： 指定error domain的值；

这样，我们就可以定义一个专门表示 `NSError` code的 `ENUM`：

```
// In Sensor.h
NS_ERROR_ENUM(NSInteger, CarSensorError, carSensorErrorDomain) {
    overHeat = 100
};
```

然后，Swift就会导入一个叫做 `CarSensorError` 的 `struct` ，它兼容了 `NSError` 中的所有信息。这样，我们就能用下面两种方式来“捕获” `Car.selfCheck` 返回的错误了。

第一种方式只能识别对应的 `NSError` ，这种方式语法上最简单，但会丢掉一些信息：

```
do {
    try vw.selfCheck()
} catch CarSensorError.overHeat {
    print("The radiator is over heat")
}
```

可以看到，这样和Swift原生的 `Error` 类型用起来完全没区别。但是，除了code之外，我们就无法读到对应的 `domain` 和 `userInfo` 信息了。

第二种方式，是直接把 `error` 转型为 `CarSensorError` ，这样不但可以直接识别错误，还可以保留所有的 `NSError` 信息：

```
do {
    try vw.selfCheck()
} catch let error as CarSensorError {
    print(error._domain)
    print(error.errorCode)
    print(error.userInfo["NSLocalizedDescription"] ?? "")
}
```

> 对于Swift引入的 `CarSensorError` 结构的定义，大家去看下SE-0112 (https://github.com/apple/swift-evolution/blob/master/proposals/0112-nserror-bridging.md)中关于实现细节的描述就明白了。

## What's next?

以上，就是让 `NSError` 适配Swift原生错误处理的方式。接下来，我们来看Swift中的 `Error` 移植到Objective-C后，是如何变成一个 `NSError` 对象的。

⏮ 理解Swift中的错误处理机制 (https://www.boxueio.com/series/error-handling/ebook/199)

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

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/)