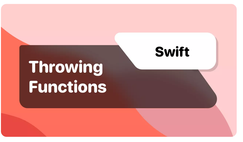
Nguyen Tuan Anh N @anhnnt Theo dõi
122 2 14
Đã đăng vào khoảng 14 giờ trước · 4 phút đọc
26 0 0

Swift: Throwing Functions

> Với các throwing function, bạn có thể dễ dàng xử lý các lỗi.



Trong Swift 2.0, Apple đã giới thiệu từ khóa throw trong Swift. Từ khóa này rất hữu ích cho chúng ta trong việc *xử lý lỗi*. Về cơ bản, từ khóa throws cho phép chúng ta xử lý các lỗi bên trong một function property, class, hoặc struct.\

Sau khi async / await ra mắt với Swift 5.5, các throw function trở nên phổ biến hơn. Thành thật mà nói, tôi thích các throwing function vì chúng rất đơn giản, dễ hiểu và trông dễ đọc hơn. Trong bài này, chúng ta sẽ tìm hiểu tất cả các trường hợp sử dụng của throwing functions.\

Sử dụng từ khóa throws

Tất cả những gì cần thiết để tạo một throwing function là viết từ khóa throws vào một function ngay trước câu lệnh return.

```
func exampleMethod(_ first: String, _ second: String) throws {
    print(first, second)
}
```

Để gọi method này, chúng ta cần sử dụng từ khóa try trước khi viết method.

```
try exampleMethod("First","Second")
```

Throwing Errors

Chúng ta sử dụng từ khóa "throw" để loại bỏ các lỗi bên trong các method, classes, structs và properties. Để throw một lỗi, chúng ta phải xác định các loại Error của chúng ta như sau:

```
enum ExampleError: Error {
    case invalid
    case uncorrect
}
```

Sau đó, chúng ta có thể sử dụng kiểu ExampleError này vào exampleMethod như sau:

```
func exampleMethod(_ first: String, _ second: String) throws {
    if first == second {
        throw ExampleError.invalid
    } else if first == "" || second == "" {
        throw ExampleError.uncorrect
    }

    print("First string: \(first), second string: \(second).")
}
```

Vì vậy, nếu bạn cung cấp các String values giống nhau cho các tham số thứ nhất và thứ hai, bạn sẽ thấy lỗi không hợp lệ trên terminal.

Throwing Initializer in Swift

Giống như bất kỳ method, chúng ta có thể tạo throwing initializers. Bằng cách này, chúng ta có thể throw một lỗi theo giá trị của các tham số mong muốn sau khi xác định một đối tượng.
Ví dụ: bạn có thể muốn xác thực email trước khi tạo đối tượng User.

```
class User {

    // MARK: - Enumerations
    enum EmailError: Error {
        case invalid
        case uncorrect
    }

    // MARK: - Properties
    let email: String
    let storedEmails = ["tcook@apple.com", "sjobs@apple.com", "cfe

    // MARK: - Life Cycle
    init(email: String) throws {
        if storedEmails.contains(email) {
            throw EmailError.invalid
        } else if email.count < 5 || email == "" {
            throw EmailError.uncorrect
        }

        self.email = email
    }
}

let user = try User(email: "contact.canbalkaya@gmail.com")
```

do-catch Statement

Để bắt một lỗi được throw, chúng ta cần sử dụng câu lệnh do-catch. Bằng cách này, mặc dù một lỗi được throw, ứng dụng của chúng ta không bị crash, nhưng chúng ta có thể hiểu rằng một lỗi đã được throw.

```
do {
    let user = try User(email: "tcook@apple.com")
    print("Created user with name \(user.email)")
} catch {
    print("User creation failed with error: \(error)")
}
```

Khi bạn chạy ví dụ trên, bạn sẽ nhận thấy rằng khi một lỗi được throw, các code được viết trong block catch sẽ chạy.
Ví dụ trên là cách sử dụng cơ bản nhất của câu lệnh do-catch. Vì vậy, chúng ta có thể làm nhiều hơn: Chúng ta có thể catch lỗi trong các block riêng biệt để hiển thị các cảnh báo khác nhau.

```
do {
    let user = try User(email: "tcook@apple.com")
    print("Created user with name \(user.email)")
} catch User.EmailError.uncorrect {
    print("Email is uncorrect.")
} catch User.EmailError.invalid {
    print("Email is invalid.")
} catch {
    print("An another error is appeared.")
}
```

Chúng ta cũng có thể catch hai hoặc nhiều loại lỗi cụ thể. Trong trường hợp này, bạn có thể sử dụng danh sách trong catch statements của mình:

```
do {
    let user = try User(email: "tcook@apple.com")
    print("Created user with name \(user.email)")
} catch User.EmailError.uncorrect, User.EmailError.invalid {
    print("Email is uncorrect or invalid.")
} catch {
    print("An another error is appeared.")
}
```

try? & try!

Nếu bạn không muốn lưu vào bộ nhớ cache bất kỳ lỗi nào được đưa ra, bạn có thể sử dụng thử ?. Bằng cách này, ứng dụng không bị treo ngay cả khi có lỗi.

```
let user = try? User(email: "sjobs@apple.com")
print(user?.email)
```

Nếu lỗi được đưa ra liên quan đến thuộc tính email của đối tượng User, thuộc tính email sẽ là nil.
Nếu bạn muốn ứng dụng bị treo mỗi khi gặp lỗi, bạn nên sử dụng try !. Điều này sẽ làm ứng dụng của bạn không thành công giống như fatalError method.

```
let user = try! User(email: "sjobs@apple.com")
print(user.email)
```

Tổng kết

Throwing functions có thể rất hữu ích trong việc viết các bài kiểm tra và các phương thức không đồng bộ. Sau Swift 5.5, dường như bây giờ nó sử dụng nhiều throwing functions hơn trước.

Nguồn tham khảo: Swift: Throwing Functions

Xcode Swift iOS

MỤC LỤC

CÁC TỔ CHỨC ĐƯỢC ĐỀ XUẤT

Sun* AI Research Team
281 35 451

Sun* Cyber Security Team
174 10 263

Avengers Group
55 16 119

Bài viết liên quan

3 lỗi nghiêm trọng của lập trình viên iOS
Phuong VNC
13 phút đọc
1033 5 0 6

Tìm hiểu về Optional trong Swift
hungbv
7 phút đọc
4378 4 1 4

[iOS] [Swift] Tổng hợp tất cả từ khóa trong ngôn ngữ Swift (Part 3)
Nguyễn Quốc Tinh
1 phút đọc
878 4 0 1

Bắt đầu với Swift, những khái niệm cơ bản
hungbv
6 phút đọc
2303 3 0 2

Bài viết khác từ Nguyen Tuan Anh N

Các Modifier và View mới trong SwiftUI iOS 15
Nguyen Tuan Anh N
4 phút đọc
53 0 0 0

Tạo thư viện Public trên CocoaPods
Nguyen Tuan Anh N
4 phút đọc
40 0 2 2

Swift: Tạo custom phép toán tử (Operator) của riêng bạn!
Nguyen Tuan Anh N
4 phút đọc
59 1 1 2

Sub-modules cho Xcode
Nguyen Tuan Anh N
2 phút đọc
97 1 1 5

Bình luận

Viết Xem trước

Viết phản hồi...

Bình luận

Chưa có bình luận nào.