Error Handling

G4ENG

29 JUL 19

Agenda

- Error Type
- Error Throw
- Do try catch

Error Type

- 스위프트에서 오류를 처리하는 방법
  - 옵셔널
  - 오류 처리 구문

Error Type

- 스위프트 초기에는 옵셔널을 통해 오류를 충분히 처리할 수 있었다
- 단점: 오류가 발생했을 때 오류에 대한 정보를 외부로 전달할 방법이 없음

Error Type

- 오류 타입 정의하기
- 오류 처리를 위해서는 오류 정보를 담아 함수나 메소드 외부로 던질 오류 타입 객체가 필요
- 이 객체는 하나의 일관된 오류 쩨에 소속된 여러 가지 오류를 정의할 수 있어야 하므로, 보통 열거형 타입으로 정의

Error Type

- 예를 들어 [yyyy-mm-dd] 형태를 보이는 문자열을 분석하여 연도, 월, 일 형식의 데이터로 각각 변환하는 함수가 있다고 해 보자
- 이 함수를 분석하는 과정에서 다음과 같은 다양한 오류를 만날 수 있다

Error Type

1. 입력된 문자열의 길이가 필요한 크기와 맞지 않는 오류
2. 입력된 문자열의 형식이 YYYY-MM-DD 형태가 아닌 오류
3. 입력된 문자열의 값이 날짜와 맞지 않는 오류

Error Type

Protocol

Error

A type representing an error value that can be thrown.

Declaration

```
protocol Error
```

Overview

Any type that declares conformance to the `Error` protocol can be used to represent an error in Swift's error handling system. Because the `Error` protocol has no requirements of its own, you can declare conformance on any custom type you create.

Error Type

```
enum DateParseError: Error {
    case overSizeString // 입력된 데이터의 길이가 크다
    case underSizeString // 입력된 데이터의 길이가 작다
    case incorrectFormat(part: String)   // 형식 안맞음
    case incorrectData(part: String)     // 데이터 값이 안맞음
}
```

Error Throw

- 우리가 작성한 오류 타입 객체는 함수나 메소드를 실행하는 과정에서 필요에 따라 외부로 던져 실행 흐름을 옮겨 버릴 수 있다
- 이때 함수나 메소드는 오류 객체를 외부로 던질 수 있다는 것을 컴파일러에 알려주기 위해 정의 구문을 작성할 때는 throws 키워드를 추가한다

Error Throw

```
// throws 키워드는 반환 타입을 표시하는 ->보다 앞에 작성해야 한다. 이는 오류를 던지면 값이 반환되지 않는다는 의미. 함수나 메소드 또는 클로저까지 모두 throws를
    사용할 수 있지만, 명시적으로 throws를 추가하지 않으면 오류를 던질 수 없다.

func canThrowErrors() throws -> String { return "a" }
func cannotThrowErrors() -> String { return "a" }
```

Error Throw

```
{() throws -> String in
    ...
}
```

Error Throw

```
import Foundation

struct Date {
   var year: Int, month: Int, date: Int
}
   func parseDate(param: NSString) throws -> Date {
       // 입력된 문자열의 길이가 10이 아닐 경우 분석이 불가능하므로 오류
       guard param.length == 10 else {
           if param.length > 10 {
               throw DateParseError.overSizeString
           }
           else {
               throw DateParseError.underSizeString
           }
       }

       // 반환할 객체 타입 선언
       var dateResult = Date(year: 0, month: 0, date: 0)

       // 연도 정보 분석
       if let year = Int(param.substring(with: NSRange(location: 0, length: 4))) {
           dateResult.year = year
       }
       else {
           // 연도 분석 오류
           throw DateParseError.incorrectFormat(part: "year")
       }
```

Error Throw

```
    // 월 정보 분석
    if let month = Int(param.substring(with: NSRange(location: 5, length: 2))) {
        // 월에 대한 값은 1~12까지만 가능하므로 그 이외의 범위는 잘못된 값으로 처리
        guard month > 0 && month < 13 else {
            throw DateParseError.incorrectData(part: "month")
        }
        dateResult.month = month
    }
    else {
        // 월 분석 오류
        throw DateParseError.incorrectFormat(part: "month")
    }

    // 일 정보 분석
    if let date = Int(param.substring(with: NSRange(location: 8, length: 2))) {
        // 일에 대한 값은 1~31
        guard date > 0 && date < 32 else {
            throw DateParseError.incorrectData(part: "date")
        }
        dateResult.date = date
    }
    else {
        // 일 분석 오류
        throw DateParseError.incorrectFormat(part: "date")
    }

    return dateResult
}
```

Error Throw

```
try parseDate(param: "2020-02-28")

// 이 값을 다른 변수나 상수에 할당할 때에도 try를 붙여서 사용하자
let date = try parseDate(param: "2020-02-21")
```

Error Throw

```
do {
  try <오류를 던질 수 있는 함수>
}
catch <오류 타입1> {
  // 오류 타입1에 대한 대응
}
catch <오류 타입2> {
// 오류 타입2에 대한 대응
}
catch <오류 타입3> {
// 오류 타입3에 대한 대응
}
catch ...
```

Error Throw

```swift
func getPartsDate(date: NSString, type: String) {
    do {
        let date = try parseDate(param: date)

        switch type {
        case "year":
            print("\(date.year)")
        case "month":
            print("\(date.month)")
        case "date":
            print("\(date.date)")
        default:
            print("no data")
        }
    }
    catch DateParseError.overSizeString {
        print("oversize")
    }
    catch DateParseError.underSizeString {
        print("undersize")
    }
    catch DateParseError.incorrectData(let part) {
        print("\(part) error")
    }
    catch DateParseError.incorrectFormat(let part) {
        print("\(part) error")
    }
    catch {
        print("unknown error")
    }
}

// 가장 마지막에 오류 타입이 작성되지 않은 catch 구문은 앞의 catch 구문에서 잡히지 않은 모든 오류를 잡아주는 와일드 카드 역할을 한다. 오류를 던지도록 설계된 함수나 메소드이지만, 필요에 의해 오류를 던지지 않게 하고 싶을 때는 try 대신 try!를 사용한다.
```

QNA