和JSON处理相关的常见错误

Back to series (/series/what-is-new-in-swift-4)

字号
字号
默认主题
金色主题
暗色主题

一直以来，我们都忽略了在编码和解码JSON时可能发生的各种错误。在了解了各种正常情况的处理方法之后，这一节，我们集中来看错误处理的问题。

处理不合法的JSON格式

第一个要处理的情况，是要解码的JSON格式不合法，这可能是多种原因造成的，例如服务端的程序有bug，或者网络传输问题等。为了演示，我们人为创建一个非法的JSON：

```
let response = """
{
    "1":{
        "title": "Episode 1"
    },
    "2": {
        "title": "Episode 2"
    },
    "3": {
        "title": "Episode 3"

}
"""
```

还是上一节中的内容，只不过，我们在最后故意去掉了一个 } 。然后直接调用上一节的全局 decode：

```
try decode(response: response, of: Episodes.self)
```

就会在控制台看到一个 DecodingError.dataCorrupted 异常，为了处理它，我们得修改一下这个 decode 函数：

```
func decode<T>(response: String, of: T.Type) throws -> T where T: Codable
 {
    let data = response.data(using: .utf8)!
    let decoder = JSONDecoder()

    do {
        let model = try decoder.decode(T.self, from: data)

        return model
    }
    catch DecodingError.typeMismatch(let type, let context) {
        dump(type)
        dump(context)
        exit(-1)
    }
}
```

这次，我们捕获了 DecodingError.dataCorrupted 异常，它有一个关联值，类型是 DecodingError.Context ，我们把这个值 dump 出来，就能看到下面的结果了：

```
▿ Swift.DecodingError.Context
  - codingPath: 0 elements
  - debugDescription: "The given data was not valid JSON."
  ▿ underlyingError: Optional(Error Domain=NSCocoaErrorDomain Code=3840 "U
nexpected end of file while parsing object."
```

其中， debugDescription 给出了错误信息的概述，而 underlyingError 则是一个 Error? 对象，包含了更具体的错误信息，这里，它提示我们*Unexpected end of file while parsing object*。

处理不存在的key

第二种错误情况，是访问了不存在的key，例如，我们先定义一个 CodingKey：

```
enum DemoKey: String, CodingKey {
    case demo
}
```

接下来，在解码的时候，我们使用这个 Demokey：

```
init(from decoder: Decoder) throws {
    let container = try decoder.container(
        keyedBy: EpisodeInfo.self)

    var v = [Episode]()
    for key in container.allKeys {
        let innerContainer = try container.nestedContainer(
            keyedBy: DemoKey.self, forKey: key)

        /// !!! Error
        let title = try innerContainer.decode(
            String.self, forKey: .demo)
        /// !!! Error

        let episode = Episode(id: Int(key.stringValue)!, title: title)
        v.append(episode)
    }

    self.episodes = v
}
```

在上面这个例子中，当我们在子容器中尝试读取key "demo"对应的值时，就会发生运行时错误。我们可以在 decode 全局函数中通过捕获 keyNotFound 出来key不存在的异常：

```
do {
    _ = try decoder.decode(Episodes.self, from: data)
}
catch DecodingError.keyNotFound(let codingPath, let context) {
    dump(codingPath)
    dump(context)
}
```

执行一下，就会看到它们的值了：

```
- Codable.DemoKey.demo
▿ Swift.DecodingError.Context
  - codingPath: 0 elements
  - debugDescription: "No value associated with key demo (\"demo\")."
  - underlyingError: nil
```

处理JSON值和model类型不匹配

最后一类错误，是JSON中包含的值和model中对应属性的类型不同。为了看到这个错误，我们把 init 方法改成这样：

```
init(from decoder: Decoder) throws {
        let container =
            try decoder.container(keyedBy: EpisodeInfo.self)

        var v = [Episode]()
        for key in container.allKeys {
            let innerContainer = try container.nestedContainer(
                keyedBy: EpisodeInfo.self, forKey: key)

            let title = try innerContainer.decode(
                Int.self, forKey: .title)
        }

        self.episodes = v
    }
```

这次，我们在解码的时候，把 title 对应的类型，改成了 Int ，但JSON中包含的是 String ，因此重新执行就会发生异常。对此，我们可以通过捕获 DecodingError.typeMismatch 来处理：

```
do {
    _ = try decoder.decode(Episodes.self, from: data)
}
catch DecodingError.typeMismatch(let type, let context) {
    dump(type)
    dump(context)
}
```

这个异常也有两个关联值，分别表示了编码错误的类型，和我们熟悉的 context 对象，执行一下，就可以看到下面的结果了：

```
- Swift.Int #0
▿ Swift.DecodingError.Context
  ▿ codingPath: 1 element
    ▿ Codable.Episodes.EpisodeInfo
      - stringValue: "title"
  - debugDescription: "Expected to decode Int but found a string/data inst
ead."
  - underlyingError: nil
```

以上，就是把JSON映射到Swift model时三种常见的错误处理。除了解码之外，当我们编码一个Swift model时，只有可能遇到一种错误，就是model属性的类型，无法编码到指定的容器中。大家可以自己了解下 enum EncodingError ，道理是一样的，我们就不重复了。

◀ 如何让model兼容多个版本的API (https://www.boxueio.com/series/what-is-new-in-swift-4/ebook/299)

返回视频 ▶ (/series/what-is-new-in-swift-4)