What's new in Swift 4

如何自定义model对象的编码过程

(https://www.boxueio.com/series/what-is-new-in-swift-4/ebook/296)

如何编码和解码带有派生关系的model

(https://www.boxueio.com/series/what-is-new-in-swift-4/ebook/298)

如何自定义JSON的解码过程

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

欢迎回来，我们继续来看如何自定义JSON解码的过程，其实绝大部分思想是自定义编码是一样的。因此，这一节的内容，既是对新内容的介绍，也是对容器概念的一个复习。

如何自定义Decoding

为了可以从一个JSON自动获得与之对应的Swift model，我们必然要自定义一个 `init` 方法：

```
struct Episode: Codable {
    var title: String
    var createdAt: Date
    var comment: String?
    var duration: Int
    var slices: [Float]

    init(from decoder: Decoder) throws {

    }

    /// ...
}
```

但在实现这个 `init` 方法之前，别忘了先添加一个memberwise init，因为一旦我们自定义了 `init` ，编译器便不再为我们合成memberwise init方法了：

```
init(title: String,
     createdAt: Date,
     comment: String?,
     duration: Int,
     slices: [Float]) {
    self.title = title
    self.createdAt = createdAt
    self.comment = comment
    self.duration = duration
    self.slices = slices
}
```

接下来，为了从 `Decoder` 中得到对应的Value，我们还是要借用容器的概念：

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

这样，我们就可以从 `container` 中解码出 `title` ， `createdAt` 和 `duration` ：

```
let title = try container.decode(
    String.self, forKey: .title)
let createdAt = try container.decode(
    Date.self, forKey: .createdAt)
let duration = try container.decode(
    Int.self, forKey: .duration)
```

然后，用 `decodeIfPresent` 解码出 `comment` ：

```
let comment = try container.decodeIfPresent(
    String.self, forKey: .comment)
```

用 `nestedUnKeyedContainer` 把百分比反推回来：

```
var unkeyedContainer =
    try container.nestedUnkeyedContainer(forKey: .slices)
var percentages: [Float] = []

while (!unkeyedContainer.isAtEnd) {
    let sliceDuration = try unkeyedContainer.decode(Float.self)
    percentages.append(sliceDuration / Float(duration))
}
```

在上面的例子里，和编码时不同的是，这次，我们直接使用了 `.slice` 得到了用于保存数组的容器。然后，只要不断的从容器中解码出元素，直到 `unkeyedContainer.isAtEnd` 为 `true` 就好了。

现在，从JSON中集齐了所有要用于初始化 `Episode` 的值之后，我们直接调用memberwise init方法：

```
self.init(title: title,
    createdAt: createdAt,
    comment: comment,
    duration: duration,
    slices: slices)
```

这样，自定义的 `decode` 方法就实现好了。然后，我们假设服务器返回的JSON是这样的：

```
let response = """
{
    "title": "How to parse a json - IV",
    "comment": "null",
    "created_at": "2017-08-24 00:00:00 +0800",
    "duration": 500,
    "slices": [125, 250, 375]
}
"""
```

这时，如果我们用下面的代码进行解码：

```
let data = response.data(using: .utf8)!
let decoder = JSONDecoder()
let episode = try! decoder.decode(Episode.self, from: data)

dump(episode)
```

执行一下就会看到一个运行时错误，提示我们*Expected to decode Double but found a string/data instead*。这是由于 `created_at` 默认需要的，是一个表示时间的浮点数，就像我们编码的时候形成的结果那样。但这里，服务器返回的是一个字符串。因此，我们也要自定义这个日期的解码方式，这和编码是类似的：

```
let data = response.data(using: .utf8)!
let decoder = JSONDecoder()

decoder.dateDecodingStrategy = .custom({ (decoder) -> Date in
    let data = try decoder
        .singleValueContainer()
        .decode(String.self)

    let formatter = DateFormatter()
    formatter.dateFormat = "yyyy-MM-dd HH:mm:ss Z"

    return formatter.date(from: data)!
})
```

在自定义 `Date` 解码的时候，closure只接受一个参数就是 `Decoder` 本身，然后，返回一个解码后的 `Date` 对象。在这个clousre的实现里，我们用Single Value Container得到JSON中表示日期的字符串。然后，根据字符串的形式定义好 `DateFormatter` ，最后，返回根据 `formatter` 生成的 `Date` 对象就好了。

这样，重新执行一下，就能看到下面的结果了：

```
▿ Codable.Episode
  - title: "How to parse a json - IV"
  ▿ createdAt: 2017-08-24 06:00:00 +0000
    - timeIntervalSinceReferenceDate: 525247200.0
  ▿ comment: Optional("null")
    - some: "null"
  - duration: 500
  - slices: 0 elements
```

扁平化JSON的编码和解码

至此，我们就已经掌握绝大多数情况下JSON的编码和解码场景了。但是，我们处理的这些场景都有一个共性，就是JSON和model在信息结构上，是一一对应的。但很多时候，情况也并不完全如此。来看个例子：

```
{
    "title": "How to parse a json - IV",
    "comment": "null",
    "created_at": "2017-08-24 00:00:00 +0800",
    "meta": {
        "duration": 500,
        "slices": [125, 250, 375]
    }
}
```

假设服务器返回的JSON是上面这样的，把 `duration` 和 `slices` 嵌套在了 `meta` 里。但我们的model的结构保持不变，还是“扁平”的。在这种情况下，如何自动完成编码和解码呢?

首先，我们要为嵌套在内层的Key单独定义一个 `enum`：

```
enum MetaCodingKeys: String, CodingKey {
    case duration
    case slices
}
```

并在之前的 `CodingKeys` 中去掉 `duration` 和 `slices`，添加这个 `meta`：

```
enum CodingKeys: String, CodingKey {
    case title
    case createdAt = "created_at"
    case comment
    case meta
}
```

接下来，当解码的时候，我们要把自定义的 `init(from decoder: Decoder)` 修改一下：

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

    /// ...

    let meta = try container.nestedContainer(
        keyedBy: MetaCodingKeys.self, forKey: .meta)

    let duration = try meta.decode(
        Int.self, forKey: .duration)

    var unkeyedContainer =
        try meta.nestedUnkeyedContainer(forKey: .slices)

    /// ...
}
```

这里，我们使用了 `nestedContainer` 方法，为 `.meta` 按照 `MetaCodingKeys` 中指定的规格，创建了一个内嵌的容器。于是，meta key中的所有内容，都通过这个内嵌的容器进行解码就好了。

另外，由于现在model和JSON的格式不对应了，我们也要自定义model的编码方法。这个过程的关键部分，和解码是类似的：

```
extension Episode {
    func encode(to encoder: Encoder) throws {
        var container = encoder.container(keyedBy: CodingKeys.self)

        /// ...

        var meta = container.nestedContainer(
            keyedBy: MetaCodingKeys.self, forKey: .meta)
        try meta.encode(duration, forKey: .duration)

        var unkeyedContainer =
            meta.nestedUnkeyedContainer(forKey: .slices)

        /// ...
    }
}
```

如果你理解了解码的过程，上面这段代码就应该没有任何难度了。全部完成后，我们来试一下。先来看解码：

```
var data = response.data(using: .utf8)!

let decoder = JSONDecoder()
decoder.dateDecodingStrategy = .custom({
    (decoder) -> Date in
    let data = try decoder
        .singleValueContainer()
        .decode(String.self)

    let formatter = DateFormatter()
    formatter.dateFormat = "yyyy-MM-dd HH:mm:ss Z"

    return formatter.date(from: data)!
})

let episode =
    try! decoder.decode(Episode.self, from: data)

dump(episode)
```

执行一下就会看到解码出来的 Episode 对象了：

```
▿ Codable.Episode
  - title: "How to parse a json - IV"
  ▿ createdAt: 2017-08-24 06:00:00 +0000
    - timeIntervalSinceReferenceDate: 525247200.0
  ▿ comment: Optional("null")
    - some: "null"
  - duration: 500
  ▿ slices: 3 elements
    - 0.25
    - 0.5
    - 0.75
```

然后再来看编码：

```
let encoder = JSONEncoder()
encoder.outputFormatting = .prettyPrinted
let encodedData = try encoder.encode(episode)

print(String(data: encodedData, encoding: .utf8)!)
```

再重新执行下， episode 对象就会变回服务器发送的JSON格式，要注意的是，这里，我们没有自定义 Date 的编码方式，因此，在编码后的结果里， created_at 只是默认的浮点数形式：

```
{
  "meta" : {
    "duration" : 500,
    "slices" : [
      125,
      250,
      375
    ]
  },
  "title" : "How to parse a json - IV",
  "comment" : "null",
  "created_at" : 525247200
}
```

What's next?

以上，就是这一节的内容，至此，我们就可以处理绝大多数常见的JSON了。但一直以来，我们使用的model都是单一类型，在下节里，我们就来看当model之间存在继承关系的时候，它们与JSON进行转换的过程。

⏮ 如何自定义model对象的编码过程

(https://www.boxueio.com/series/what-is-new-in-swift-4/ebook/296)

如何编码和解码带有派生关系的model ⏭

(https://www.boxueio.com/series/what-is-new-in-swift-4/ebook/298)