What's new in Swift 4

如何处理常见的JSON嵌套结构 (https://www.boxueio.com/series/what-is-new-in-swift-4/ebook/295)

如何自定义JSON的解码过程 (https://www.boxueio.com/series/what-is-new-in-swift-4/ebook/297)

如何自定义model对象的编码过程

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

虽然绝大多数时候，我们让一个类型遵从 Codable 就可以享受它提供的编码和解码便利，但终有一些时候，我们需要改变其中一些默认的规则，例如我们在最初看到的关于日期和时间的编码。为此，在这一节，我们将走到 Codable 内部，来定制其中的方法。

自定义Encoding

为了演示这个过程，我们先来看在第一个视频中编码日期时间的问题。这里，我们对 Episode 进行了一些简化：

```
struct Episode: Codable {
    let title: String
    let createdAt: Date

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

然后，当我们编码一个 Episode 对象的时候：

```
let episode = Episode(
    title: "How to parse a JSON - III",
    createdAt: Date())
let encoder = JSONEncoder()

let data = try! encoder.encode(episode)

dump(String(data: data, encoding: .utf8)!)
```

就会得到这样的结果：

```
"{\"title\":\"How to parse a JSON - III\",\"created_at\":525248629.55017
7}"
```

为了自定义 Episode 的编码过程，我们要实现下面这个方法：

```
extension Episode {
    func encode(to encoder: Encoder) throws {
        // Todo: Add the custom encoding here
    }
}
```

接下来， Encoder 中包含了几种不同的容器，我们要做的，就是根据要编码的内容，选择对应的容器，并把对应的值编码进去。例如：

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

在 encode 的实现里，我们使用的容器叫做**Keyed Container**。因为，要编码的内容同时包含了key和value，我们仍旧通过 CodingKeys.self 告诉容器model中的属性和JSON中的key的对应规则。另外，由于我们要向 container 中添加内容，它应该必须是一个变量。

有了容器之后，就是把每个属性编码到容器里：

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

    try container.encode(title, forKey: .title)
    try container.encode(createdAt, forKey: .createdAt)
}
```

现在重新执行一下，会发现，结果和之前是相同的。这很正常，因为当我们什么都不写的时候，`Codable` 中默认的 `encode` 实现，和我们自己写的是一样的。

现在，要定制 `Date` 的编码方式，为此，我们得使用另外一种容器，叫做**Single Value Container**：

```
encoder.dateEncodingStrategy = .custom({ (date, encoder) in
    let formatter = DateFormatter()
    formatter.dateFormat = "yyyy-MM-dd hh:mm:ss"
    let stringData = formatter.string(from: date)
    var container = encoder.singleValueContainer()

    try container.encode(stringData)
})
```

当我们把 `.dateEncodingStrategy` 改成 `.custom` 之后，它的关联值，便是一个closure。这个closure的第一个参数，表示要编码的 `Date` 对象，第二参数，是 `Encoder` 对象，在这个clousre的实现里，最关键的，是最后两行：

```
var container = encoder.singleValueContainer()
try container.encode(stringData)
```

这样，我们就用**Single Value Container**自定义了 `Date` 对象的编码格式。重新执行一下，就会看到这样的结果：

```
- "{\"title\":\"How to parse a JSON - III\",\"created_at\":\"2017-08-24 0
8:14:56\"}"
```

自定义Optional对象的编码

了解了 `encode` 的实现之后，我们来看一个特殊情况，当处理 `Optional` 类型的属性时，如果属性为 `nil`，就不会出现在默认的结果里了。例如，我们把 `Episode` 改成这样：

```
struct Episode: Codable {
    let title: String
    let createdAt: Date
    let comment: String?

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

然后，创建一个 `duration` 为 `nil` 的对象：

```
let episode = Episode(
    title: "How to parse a JSON - III",
    createdAt: Date(),
    comment: nil)

/// ...

let data = try! encoder.encode(episode)
dump(String(data: data, encoding: .utf8)!)
```

这是，我们可以先把自定义的 `encode` 注释掉，重新执行就会发现， `comment` 并不在里面：

```
"{\"title\":\"How to parse a JSON - III\",\"created_at\":\"2017-08-24 09:4
0:58\"}"
```

这是因为，在默认的实现里，对于Optional类型，默认是用 `encodeIfPresent` 进行编码的，只有Optional不为 `nil`，值才会编码到结果里。但通常，这个行为不是我们想要的，为此，我们可以在自定义的 `encode` 里强行把它编码进来：

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

    try container.encode(title, forKey: .title)
    try container.encode(createdAt, forKey: .createdAt)
    try container.encode(comment, forKey: .comment)
}
```

重新执行一下，就能看到结果了：

```
"{\"title\":\"How to parse a JSON - III\",\"created_at\":\"2017-08-24 10:1
1:40\",\"comment\":null}"
```

至此，我们就已经了解了两种“容器”了，接下来，我们看如何自定义数组类型的编码，并以此了解第三种容器的用法。

自定义数组类型的编码

之前我们提到过，在JSON里，value除了一个简单值之外，还可以是一个数组。例如，我们给 Episode 添加一个包含视频切片时间的数组 slices 以及视频的总时长 duration：

```
struct Episode: Codable {
    /// ...
    let duration: Int
    let slices: [Float] = [0.25, 0.5, 0.75]

    enum CodingKeys: String, CodingKey {
        /// ...
        case slices
    }
}
```

其中， slices 表示在 duration 的25% / 50% / 75%的位置，对视频进行切片。当我们默认对 slices 编码的时候，这些百分比数字就会直接包含在JSON里。

如果我们希望提交到服务器的JSON中包含的是切片实际的时长，就可以在 encode 里自定义数组的编码过程：

```
func encode(to encoder: Encoder) throws {
    /// ...

    var unkeyedContainer =
        container.nestedUnkeyedContainer(forKey: .slices)
    try slices.forEach {
        try unkeyedContainer.encode(Float(duration) * $0)
    }
}
```

这里，由于 slices 是JSON的一部分，因此，我们使用了 nestedUnkeyedContainer 方法，创建了一个Nested Unkeyed Container。然后，只要遍历数组中的每个成员，根据 duration 计算对应的时间，并把时间编码进Unkeyed Container就好了。

然后：

```
let episode = Episode(
    title: "How to parse a JSON - III",
    createdAt: Date(),
    comment: nil,
    duration: 500,
    slices: [0.25, 0.5, 0.75)

let data = try! encoder.encode(episode)
let encoder = JSONEncoder()
encoder.outputFormatting = .prettyPrinted

/// ...

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

重新执行下，编码过的 slices 中，包含的就是对应切片的时长了：

```
{
  "slices" : [
    125,
    250,
    375
  ],
  "title" : "How to parse a JSON - III",
  "comment" : null,
  "created_at" : "2017-09-03 10:51:43"
}
```

What's next?

以上，就是和自定义对象编码有关的内容。其中最重要的，就是要理解容器的概念，以及三种不同容器（keyed container / unkeyed container / single value container）的应用场景。在下一节，我们来看，如何自定义JSON解码的过程。

⏮ 如何处理常见的JSON嵌套结构
(https://www.boxueio.com/series/what-is-new-in-swift-4/ebook/295)

如何自定义JSON的解码过程 ⏭
(https://www.boxueio.com/series/what-is-new-in-swift-4/ebook/297)