What's new in Swift 4

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)

和JSON处理相关的常见错误

(https://www.boxueio.com/series/what-is-new-in-swift-4/ebook/300)

如何让model兼容多个版本的API

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

在前面的内容中，我们都在讨论JSON和model转换时用到的各种语言层面的内容。这一节，我们来看一些更偏业务层面的场景。

有时，我们接手的任务并不是从零开始的。例如，我们有两个版本返回视频信息的API，老版本中视频创建日期的格式是这样的：

```
{
  "created_at": "Oct-24-2017"
}
```

而新版本API中日期的格式是这样的：

```
{
  "created_at" : "2017-08-28T00:24:10+0800"
}
```

现在，如何让我们的model在保证兼容性的前提下，过度到新的API呢？为了更好的演示这一节的内容，我们把之前使用的 Episode 对象进行了一些简化，让它只保留一个表示时间的字段：

```
struct Episode: Codable {
    var createdAt: Date

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

现在，为了兼容老版本的API，我们可以这样。

首先，定义一个包含版本信息的结构 EpisodeCodingOptions：

```
struct EpisodeCodingOptions {
    enum Version {
        case v1
        case v2
    }

    let apiVersion: Version
    let dateFormatter: DateFormatter

    static let infoKey = CodingUserInfoKey(
        rawValue: "io.boxue.episode-coding-options")!
}
```

其中：

- apiVersion 用于区分API版本；
- dateFormatter 用于表示不同版本的日期格式；
- infoKey 是一个 CodingUserInfoKey 对象，它的作用，有点儿类似 Dictionary 中的key。稍后我们就会看到，每个 encoder 都可以通过这个类型的对象包含一些关于编码的额外信息，它的参数只用于标识不同的key，没有其他的含义。由于所有的 EpisodeCodingOptions 都应该使用相同的标识符，因此，我们把它定义成class attribute就可以了；

其次，定义一个表示老版本API的 EpisodeCodingOptions 对象：

```
let formatter = DateFormatter()
formatter.dateFormat = "MMM-dd-yyyy"

let options = EpisodeCodingOptions(
    apiVersion: .v1, dateFormatter: formatter)
```

第三，为了适配老版本的API，我们修改一下在上一节中实现的全局 encode 函数。先给它添加一个参数，用于传递版本信息：

```
func encode<T>(of model: T,
    options: [CodingUserInfoKey: Any]!) throws where T: Codable {
    // ...
}
```

并且，当 options 不为 nil 的时候，我们用它设置 encoder：

```
func encode<T>(of model: T,
    options: [CodingUserInfoKey: Any]!) throws where T: Codable {
    // ...
    if options != nil {
        encoder.userInfo = options
    }
    // ...
}
```

第四，我们就可以在编码的时候得到要使用的版本信息了。为了使用这个信息，我们得自定义 encode 方法：

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

        if let options =
            encoder.userInfo[EpisodeCodingOptions.infoKey] as?
                EpisodeCodingOptions {
            let date = options.dateFormatter.string(from: createdAt)
            try! container.encode(date, forKey: .createdAt)
        }
        else {
            fatalError("Can not read coding options.")
        }
    }
}
```

在上面的代码里，我们先读取了 encoder.userInfo[EpisodeCodingOptions.infoKey] 并尝试把它转型成一个 EpisodeCodingOptions 。如果转型成功了，就表示我们得到版本信息了。这里，我们直接读取 options.dateFormatter 生成对应的字符串，并把这个字符串编码到 createdAt 对应的值就好了。当然，这里，我们也可以通过读取 options.apiVersion 做一些针对API版本的特别动作。

这样，所有的工作就都完成了，重新执行一下，就会看到下面这样的结果：

```
{
  "created_at" : "Aug-28-2017"
}
```

当我们要编码到新API时，只要定义 .v2 版本的 EpisodeCodingOptions 就好了：

```
let formatter = DateFormatter()
formatter.dateFormat = "yyyy-MM-dd'T'HH:mm:ssZ"

let options = EpisodeCodingOptions(
    apiVersion: .v2, dateFormatter: formatter)
```

重新执行下，就会得到下面的结果：

```
{
  "created_at" : "2017-08-28T00:24:10+0800"
}
```

理解了 encode 的方法之后，大家可以试着自己写一下用于解码的 init 方法，原理是完全一样的，我们就不重复了。

处理Key的个数不确定的JSON

解码Key不确定的JSON

除了兼容性之外，另外一类我们还没提过的场景，就是JSON中key的个数是不确定的，例如，用视频id作为key：

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

面对这种情况，显然我们无法把所有的id值都通过model属性一一对应起来。怎么办呢?

首先，我们为这个新的JSON格式定义一个model：

```
struct Episodes: Codable {

}
```

其次，把之前表示视频的 Episode 修改成下面这个样子，让它只包含表示视频 id 和 title 的属性：

```
struct Episodes {
    /// ...
    struct Episode: Codable {
        let id: Int
        let title: String
    }
}
```

第三，在 Episodes 里，我们要定义一个更灵活的 CodingKey 类型来表示JSON和model的对应关系：

```
struct Episodes {
    struct EpisodeInfo: CodingKey {
        var stringValue: String
        init?(stringValue: String) {
            self.stringValue = stringValue
        }

        var intValue: Int? { return nil }
        init?(intValue: Int) { return nil }

        static let title = EpisodeInfo(stringValue: "title")!
    }
}
```

这里，对于一个遵从了 CodingKey 的类型来说， **stringValue 和 intValue 属性，以及接受 String 和 Int 为参数的 init 方法是 protocol 强制要求的，我们必须定义它们**。稍后就会看到，由于我们可以从JSON的key中读到 id ，因此在 EpisodeInfo 里，我们只要在最后，定义 title 在model中的映射规则就好了。

最后，我们用一个 Array<Episode> 存储JSON中的所有内容：

```
struct Episodes {
    /// ...
    var episodes: [Episode] = []
}
```

这样，model的部分就完成了，我们最终还是用了一个 Array ，解决了JSON key个数不确定的问题。接下来，为了从JSON自动生成model，我们只要重写 Episodes 的 init 方法就好了：

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

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

        let title = try innerContainer.decode(
            String.self, forKey: .title)
        let episode = Episode(id: Int(key.stringValue)!,
            title: title)

        v.append(episode)
    }

    self.episodes = v
}
```

在上面的代码里：

首先，用 EpisodeInfo 定义的规格创建了解码用的容器；

其次，用一个 for 循环，遍历了JSON中的所有key。在这个循环内部，我们先用 EpisodeInfo 读到了和没一个key对应的子容器，在这个子容器中，通过解码 title 得到了对应的值，并且，通过 Int(key.stringValue)! 得到了对应的视频id。这样，创建 Episode 需要的所有值就都准备好了；

第三，我们创建 Episode 对象，并把它添加到保存结果的临时数组里：

最后，用临时变量更新 self.episodes 的值。之所以这样做，是为了避免JSON中存在非法数据而破坏之前的历史数据，我们只有在所有值都转换成功之后，才更新原始值。

这样，所有的代码就完成了。我们定义一个 decode 全局函数来观察下解码的结果：

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

    dump(model)
}
```

和之前的全局 encode 类似，我们只是封装了之前的解码代码。现在我们试着解码一下之前的 response：

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

执行一下，就能看到下面这样的结果了：

```
▿ 3 elements
  ▿ Codable.Episodes.Episode
    - id: 2
    - title: "Episode 2"
  ▿ Codable.Episodes.Episode
    - id: 1
    - title: "Episode 1"
  ▿ Codable.Episodes.Episode
    - id: 3
    - title: "Episode 3"
```

可以看到，这和我们在 Episodes 中设计的数据结构是一样的。

编码Key不确定的JSON

了解了解码之后，我们再来看如何编码 Episodes 对象，基本思路其实是一样的，直接来看 encode 的源代码：

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

    for episode in episodes {
        let id = EpisodeInfo(
            stringValue: String(episode.id))!
        var nested = container.nestedContainer(
            keyedBy: EpisodeInfo.self, forKey: id)

        try nested.encode(episode.title, forKey: .title)
    }
}
```

首先，我们还是用 EpisodeInfo 约定的规格创建了用于编码的容器；

其次，我们遍历了 episodes 数组，用 id 创建了JSON中的每一个key，并用这个key创建了子容器；

最后，在这个子容器里，我们编码进了 title 的值；

这样，编码过程就完成了。为了方便利用之前的解码结果，我们把刚才实现的全局 decode 改一下：

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

    return model
}
```

然后，把之前解码后的结果再编码回来：

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

执行一下，可以看到下面这样的结果了：

```
{
  "2" : {
    "title" : "Episode 2"
  },
  "1" : {
    "title" : "Episode 1"
  },
  "3" : {
    "title" : "Episode 3"
  }
}
```

和最初，我们在 response 中的内容是一样的。

What's next?

以上，就是这一节的内容，在了解了如何处理版本过渡，以及Key不确定的JSON之后，下一节，我们来看和JSON处理相关的最后一部分内容，如何处理解码和编码JSON时的常见错误。

◀ 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)

和JSON处理相关的常见错误 ▶
(https://www.boxueio.com/series/what-is-new-in-swift-4/ebook/300)

从开发、测试到运维，让技术不再成为你成长的绊脚石。我们用打磨产品的精神去传播知识，把最新的移动开发技术，通过简单的图表， 清晰的视频，简明的文字和切实可行的例子一一向你呈现。让学习不仅是一种需求，也是一种享受。

泊学动态

一个工作十年PM终创业的故事（二） (https://www.boxueio.com/after-the-full-upgrade-to-swift3)
Mar 4, 2017

人生中第一次创业的"10有" (https://www.boxueio.com/founder-chat)
Jan 9, 2016

猎云网采访报道泊学 (http://www.lieyunwang.com/archives/144329)
Dec 31, 2015

What most schools do not teach (https://www.boxueio.com/what-most-schools-do-not-teach)
Dec 21, 2015

一个工作十年PM终创业的故事（一） (https://www.boxueio.com/founder-story)
May 8, 2015

泊学相关

关于泊学 ›

加入泊学 ›

泊学用户隐私以及服务条款 (HTTPS://WWW.BOXUEIO.COM/TERMS-OF-SERVICE)

版权声明 (HTTPS://WWW.BOXUEIO.COM/COPYRIGHT-STATEMENT)

联系泊学

Email: 10[AT]boxue.io (mailto:10@boxue.io)

QQ: 2085489246

友情链接 SwiftV (http://www.swiftv.cn) | Seay信息安全博客 (http://www.cnseay.com) | Swift.gg (http://swift.gg/) | Laravist (http://laravist.com/) | SegmentFault (https://segmentfault.com) | 靛青K的博客 (http://blog.dianqk.org/)