Protocol和泛型的台前幕后

具象函数和泛型函数的解析顺序
(https://www.boxueio.com/series/protocol-and-generic/ebook/190)

如何通过泛型编程简化网络请求？
(https://www.boxueio.com/series/protocol-and-generic/ebook/192)

我们的网络请求代码是怎么变成乱七八糟的?

Back to series (/series/protocol-and-generic)

字号
字号
默认主题
金色主题
暗色主题

现如今，网络通信几乎已经是每一个app的标配功能了。但是处理服务器返回的JSON通常不是个让人愉快的事情。这种代码通常有三个特点：

1. 网络通信代码绝大部分的逻辑相同；
2. JSON序列化成Model会包含大量的类型转换；
3. 为了处理各个环节可能发生的错误，会引入大量的optional操作；

于是，这部分功能很容易写出复杂臃肿的函数，但仔细看看却又觉得没什么更好的办法。这种函数不仅看上去不美观，也难以进行单元测试。为了了解如何改进这个过程，在这一节里，我们就先来重现这个过程。

测试场景

假设，我们要设计一个API，获取泊学所有的视频列表：

```
https://api.boxue.io/v1/episodes
```

它返回的JSON是这样的：

- `level` 表示视频的难度，从1到3分别表示初级、中级和高级；
- `title` 表示视频标题；
- `urls` 是一个对象，表示视频的三种不同格式的URL；

为了方便在app中访问这些信息，我们定义一个 `class Episode` 表示视频信息的model：

```
enum Level: Int {
    case beginner = 1, intermediate, advanced
}

class Episode {
    typealias EpisodeInfo = [String: String]

    var level: Level
    var title: String
    var urls: EpisodeInfo

    init(level: Level, title: String, urls: EpisodeInfo) {
        self.level = level
        self.title = title
        self.urls = urls
    }
}
```

除了这些“传统”得像模板一样的代码之外，我们知道从服务器获取的JSON在Swift中会被表示成一个 `[String: Any]` 的字典，于是，我们最好给 `Episode` 再添加一个convenience init，帮助我们从一个 `[String: Any]` 直接生成 `Episode` 对象。由于并不是所有的 `[String: Any]` 都可以生成 `Episode` 对象，我们得定义一个failable init：

```
class Episode {
    // ...

    convenience init?(response: [String: Any]) {
        guard let levelValue = response["level"] as? Int,
            let level = Level(rawValue: levelValue),
            let title = response["title"] as? String,
            let urls = response["urls"] as? EpisodeInfo
        else {
            return nil
        }

        self.init(level: level, title: title, urls: urls)
    }
}
```

在上面的代码里，我们在 guard 条件中串联了多个value binding表达式。这样，只有我们成功读取了JSON所有字段之后，才会创建 Episode 对象，否则就返回 nil 。这样写，要比我们使用多个if判断optional是否为 nil ，或者使用force unwrapping简洁和安全的多。

这样，所有的准备工作就完成了。接下来，就是完成网络通信的部分了。

一个传统的HTTP请求实现

我们来看一个最“朴素”的版本，把业务逻辑和网络请求统统塞进一个函数里。首先是第一部分，从服务器请求数据：

```
func getEpisodes() {
    // Phase 1: Network request
    let episodesData = try? Data(contentsOf: episodesUrl)

    let jsonRoot = episodesData.flatMap {
        try? JSONSerialization.jsonObject(with: $0)
    }

    // ...
}
```

在上面的代码里，我们通过一个同步网络请求从服务器获取了JSON结果，最终 jsonRoot 是一个 Any? 类型的对象。接下来，我们就要根据HTTP API返回的JSON规则，来解析这个 Any? 对象：

```
func getEpisodes() {
    // Phase 1: Network request

    // Phase 2: Parse the response
    var episodes: [Episode]? = nil

    if let jsonRoot = (jsonRoot as? JSONObj),
       let episodeInfo = (jsonRoot["episodes"] as? [JSONObj]) {
        episodes = episodeInfo.map {
            Episode(response: $0)
        }
        .filter { $0 != nil }
        .map { $0! }
    }
}
```

由于我们之前为 Episode 已经定义好了通过 [String: Any] 直接生成 Episode 对象的方法，这一步我们要做的，仅仅是在返回结果里通过类型转换，一层层的把需要的字段扒出来，然后生成 Episode 对象就好了。

这一步，我们会得到一个 [Episode]? 。最后，我们就可以根据得到的结果编写进一步的处理逻辑了，把它持久化也好，显示一些内容也好这就是我们比较熟悉的操作了，我们可以把这些操作用一个 ([Episode]?) -> Void 类型的函数表示。最终，我们的 getEpisode 就会被实现成这样：

```
func getEpisodes(callback: ([Episode]?) -> Void) {
    // Phase 1: Network request
    // Phase 2: Parse the response

    // Phase 3: Business logic
    callback(episodes)
}
```

怎么样，如果你跟着写下来就会发现，不知不觉 getEpisodes 就变成了一个看上去很复杂并难以测试的方法。更何况，接下来，如果要获取所有的文档信息呢？我们现在的做法只能是复制一个 getEpisode 实现，然后修改其中很少一部分的细节。但你知道，面对这种绝大多数逻辑都相同的工作，这样做显然是不科学的。

What's next?

怎么办呢？在下一节中，我们就通过泛型编程，把 getEpisode 中的这三个过程真正抽象出来，以达到进一步解耦网络请求和业务逻辑代码的目的。

◀ 具象函数和泛型函数的解析顺序
(https://www.boxueio.com/series/protocol-and-generic/ebook/190)

如何通过泛型编程简化网络请求？ ▶
(https://www.boxueio.com/series/protocol-and-generic/ebook/192)

职场漂泊的你，每天多学一点。

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