Protocol和泛型的台前幕后

我们的网络请求代码是怎么变成乱七八糟的?
(https://www.boxueio.com/series/protocol-and-generic/ebook/191)

编译器是如何理解面向protocol编程的?
(https://www.boxueio.com/series/protocol-and-generic/ebook/193)

如何通过泛型编程简化网络请求?

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

字号
字号
默认主题
金色主题
暗色主题

其实，绝大多数请求HTTP API的方法，它们的执行流程都可以按照我们之前的方式分成三个不同的阶段。不同的，仅仅是请求的细节、对结果的解析方法，以及要执行的具体业务逻辑。

因此，不难想象，我们可以把解析结果的方法也作为一个函数参数。更进一步，我们可以让这个函数参数是一个泛型函数，这样，就可以处理任意规则的API返回结果了。按照这个思路，我们可以把 `getEpisodes` 改成 `getResource`，像这样：

```
func getResource<T>(at path: URL,
                    parse: (Any) -> T?,
                    callback: (T?) -> Void) {

}
```

其中，`parse` 用来解析服务器返回结果，`callback` 用来处理业务逻辑。然后，第一阶段的网络请求，和之前基本是一样的：

```
func getResource<T>(at path: URL,
                    parse: (Any) -> T?,
                    callback: (T?) -> Void) {
    // Phase 1: Network request
    let resourceData = try? Data(contentsOf: path)

    let jsonRoot = resourceData.flatMap {
        try? JSONSerialization.jsonObject(with: $0, options: [])
    }

    // ...
}
```

接下来，我们可以把第二阶段和第三阶段合并起来，先读取 `jsonRoot` 的非 `nil` 值传递给 `parse`，并把 `parse` 的返回值传递给 `callback`：

```
func getResource<T>(at path: URL,
                    parse: (Any) -> T?,
                    callback: (T?) -> Void) {
    // ...
    callback(jsonRoot.flatMap(parse))
}
```

这样，一个“通用”的HTTP API请求的代码就写好了。为了解析之前的视频响应，我们需要自己写一个 `parse` 方法：

```
func parseEpisodes(jsonRoot: Any) -> [Episode]? {
    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! }
    }

    return episodes
}
```

这和我们之前在第二阶段实现的代码是一样的。最后，我们之前实现的 `getEpisodes` 就可以变成这样：

```
func getEpisodes() {
    getResource(at: URL("https://api.boxue.io/v1/episodes")!,
        parse: parseEpisodes,
        callback: { print($0 ?? "") })
}
```

并且，当我们再要请求其他API的时候，只要自定义服务器返回的解析方法和对应的业务逻辑就好了。这要比我们一开始的“大杂烩”版本清晰和方便很多。

但如果我们稍微仔细想下就会发现，其实 at 和 parse 这两个参数的关系是非常密切的，因为不同的API一定会有不同的 parse 方法，它们应该被封装在一起，共同表示一个“网络资源”。

通过泛型类型进一步解耦代码

假设，这个类型叫做 Resource ，由于 parse 是个泛型函数， Resource 也应该是一个泛型类型，像这样：

```
struct Resource<T> {
    let path: URL
    let parser: (Any) -> T?
}
```

然后，我们可以把之前同步请求的方法，添加到 Resource<T> 的 extension 里：

```
extension Resource {
    func syncLoad(callback: (T?) -> Void) {
        let resourceData = try? Data(contentsOf: path)

        let jsonRoot = resourceData.flatMap {
            try? JSONSerialization.jsonObject(with: $0, options: [])
        }

        callback(jsonRoot.flatMap(parse))
    }
}
```

这样，我们之前读取视频信息的代码就可以写成：

```
let episodeResource: Resource<[Episode]> =
    Resource(
        path: URL("https://api.boxue.io/v1/episodes")!,
        parser: parseEpisodes)

episodeResource.syncLoad(
    callback: { print($0 ?? "") })
```

并且，给 Resource<T> 加上异步加载的功能也是举手之劳的事情：

```
extension Resource {
    func asyncLoad(
        callback: @escaping (T?) -> Void) {
        let session = URLSession.shared

        session.dataTask(with: path) {
            resourceData, _, _ in
            let jsonRoot = resourceData.flatMap {
                try? JSONSerialization.jsonObject(with: $0)
            }

            callback(jsonRoot.flatMap(self.parse))
        }.resume()
    }
}
```

可以看到，我们只是使用 URLSession 替代了同步的 Data(contentsOf:) 方法。这里由于 callback 是被异步回调的，因此这个回调函数很大可能会离开 asynchronouslyLoad 方法存活，因此，它必须是一个 @escaping 方法。

这样，我们就通过若干泛型方法和一个泛型类型，把网络请求、解析返回结果以及对应的业务逻辑实现这三部分功能成功分开了。在最后，即便我们要使用不同的HTTP方法，或者添加不同的HTTP header，都是很简单的事情，我们只要给 Resource 添加不同的属性就好，而无须因为这种事情去修改绑定着业务逻辑的代码。这种功能独立、相互隔离的方法无疑给开发、测试和维护，都带来了极大的便利。

What's behind?

通过这几节的内容我们可以发现，在Swift里，泛型编程和 `protocol` 的关系是非常密切的，如果我们不使用 `prototol` 对泛型类型做出约束，几乎很难写出操作泛型类型的代码。为什么会这样呢？在接下来的两节里，我们就从编译器实现的角度出发，和大家分享编译器是如何理解 `protocol` 和泛型编程的。

我们的网络请求代码是怎么变成乱七八糟的?
(https://www.boxueio.com/series/protocol-and-generic/ebook/191)

编译器是如何理解面向protocol编程的?
(https://www.boxueio.com/series/protocol-and-generic/ebook/193)

泊 .I HD

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

2017 © Boxue, All Rights Reserved. 京ICP备15057653号-1 (http://www.miibeian.gov.cn/) 京公网安备 11010802020752号 (http://www.beian.gov.cn/portal/registerSystemInfo?recordcode=11010802020752)

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/)