

04 | 工整与自由的风格之争：SOAP和REST

2019-09-18 四火

全栈工程师修炼指南

[进入课程 >](#)



讲述：四火

时长 21:07 大小 14.50M



你好，我是四火。

今天我要邀请两位风格迥异的主角登上舞台，一位西装革履，另一位随性洒脱。前面那位，代表着工整、严谨和细致；后面那位，代表着自由、灵活和简约。

它们来自两个不同的时代，却同时活跃于当今的互联网，并担当着重量级的角色，影响了一批新技术的诞生。今天，就让我们来认识下它们，它们的名字，分别叫做 SOAP 和 REST。

概念

SOAP, Simple Object Access Protocol, 即简单对象访问协议, 定义了数据对象传输的格式, 以便在网络的节点之间交换信息。你可能会问, HTTP 不就是干这事的吗? 其实, 它

们都在 OSI 7 层模型的应用层上，但却互不冲突，SOAP 并不依赖于 HTTP 而存在，而且它们可以互相配合。

HTTP 负责信息的传输，比如传递文本数据，关心的是消息的送达，却并不关心这个数据代表着什么。这个数据可能本来是一个内存里的对象，是一篇文章，或者是一张图片。但是 SOAP 恰恰相反，它关心的就是怎样把这个数据给序列化成 XML 格式的文本，在传输到对端以后，再还原回来。

用一个形象的比喻就是，**消息传输就像快递，HTTP 主要关心的是信封，而 SOAP 主要关心的是信封里的物件**。今天我们讨论的 SOAP，不仅仅是协议本身，更主要的是它的风格。


REST, Representational State Transfer, 即表现层状态转换，指的是一种为了信息能在互联网上顺利传递而设计的软件架构风格。对，请注意，**SOAP 是协议，但 REST 是风格，而非协议或标准**，至于 HTTP，它是 REST 风格的重要载体。重要，但不是唯一，因为载体并不只有 HTTP 一个，比如还有 HTML 和 XML，它们恰当地互相配合，组合在一起，来贯彻和体现 REST 的风格。

SOAP 和 REST，由于概念层次上的不同，其实原本是无法放到一起比较的，但是当我们旨在风格层面上讨论 SOAP 和 REST 的时候，这件事却变得可行而有趣了。

现在让我们用一个实际例子来进一步认识它们。这个例子很常见，假设我们要设计一个图书馆，馆中的书可以增删改查（CRUD），特别是要添加一本书的时候，我们分别来看看，应用 SOAP 该怎么做，应用 REST 又该怎么做。

SOAP

这是一个最简单的给图书馆添加一本书的 XML 消息体：

 复制代码

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <soap:Envelope
3   xmlns:soap="https://www.w3.org/2003/05/soap-envelope/"
4   soap:encodingStyle="https://www.w3.org/2003/05/soap-encoding">
5   <soap:Body xmlns:b="...">
6     <b:CreateBook>
7       <b:Name>...</m:Name>
8       <b:Author>...</m:Author>
```

```
9      ...
10    </b:CreateBook>
11  </soap:Body>
12 </soap:Envelope>
```

让我来简单解释一下：

1. 第一行指明了这个消息本身格式是 XML，包括它的版本号和编码格式。
2. 这里的很多标签都带有 “soap:” 的前缀，其实，这里的 “soap” 就是 XML 的命名空间（其中 “xmlns” 就是指 “xml namespace” ），并且通过 XML schema 的方式预先定义好了如下两个 SOAP 消息必须要遵从的规则：


一个是代码片段第 3 行的 [soap-envelope](#)，它定义了基本的语法规则，比如标签的从属关系，比如同级某标签的个数限制等等，举例来说，你可以看到例子中有一个 Body 标签位于 Envelope 内部，这就是它定义的；

另一个是代码片段第 4 行的 [soap-encoding](#)，它定义了编码和数据类型等规则。

3. 在 Body 标签内部，有一个 CreateBook 标签，这是我们的业务标签，命名空间 b 也是我们自己定义的。通过在内部封装姓名（Name）和作者（Author）等书本信息，实现了在图书馆中添加书本的需求。

上面是一个最简单的例子，实际在 Envelope 中还可以添加 Head 标签，用于存放头部信息，在 Body 中可以添加 Fault 标签，用于存放错误信息。关于这些，都在 XML schema 中做了严格的定义，通过它可以帮助验证一个 XML 是否符合格式，从而可以在最短的时间内验证并发现消息中的格式问题。

SOAP 通常是通过 HTTP POST 的方式发送到对端的，这个图书馆对书本的增删改查操作，URL 可以是同一个，这是因为 SOAP 消息的具体内容写明了具体要干什么（上述的 CreateBook 标签）。比如下面这个例子，请注意其中的 Content-Type，它是令响应具备自我描述特性的重要组成部分：

 复制代码

```
1 POST /books HTTP/1.1
2 Host: xxx
```

```
3 Content-Type: application/soap+xml; charset=utf-8
4 Content-Length: xxx
5
6 ... (省略前述的 SOAP 消息体)
```

最后，谈谈经常和 SOAP 放在一起谈论的 WSDL，Web Service Description Language。

WSDL 用于描述一个 Web Service，说白了，就是用来说明某个 Web 服务该怎样使用，有怎样的接口方法，支持怎样的参数，会有怎样的返回。由于支持 SOAP 的服务端接口是经常使用 WSDL 来描述，因此我们才看到它们总被放在一起讨论，于是在这种情况下，**WSDL 常常被形容成 SOAP 服务的使用说明书**，但是请注意，本质上它们之间不存在依赖关系。

这种将服务和服务的描述解耦开设计的方式非常普遍，希望你可以去类比和联想。在软件的世界里，我们经常谈论这个“描述”的行为，以及描述者和被描述者。比如元属性描述数据，方法签名描述方法，类描述对象等等。

REST


现在，我们再来看 REST 的做法。**REST 的核心要素包括资源、表现层和状态转换这三个部分**。我们把前面客户端发送请求的过程使用 REST 风格再来实现一遍，你将看到这三个要点是怎样体现出来的：

1. 协议

我们将使用 HTTP 协议，在加密的情况下，协议是 HTTPS，但这对我们的实现来说没有什么区别。

2. URL

通常来说，这个 URL 要包括域名、版本号以及实体名称，而这个 URL 整体，代表了 REST 中的一类或一项“资源”。比如说：

 复制代码

1 <https://xxx/v1/books>

请注意其中的实体名称，它往往是一个单纯的名词，并且以复数形式出现。

这里提到了 URL，我想给经常混用的 URL、URI 做个简要的说明：URL 指的是 Uniform Resource Locator，URI 指的是 Uniform Resource Identifier，前者是统一资源定位符，后者是统一资源标识符。**Identifier 可以有多种形式，而 locator 只是其中一种，因此 URI 更宽泛，URL 只是 URI 的其中一种形式。**

当我们提到一个完整的地址，例如 <https://www.google.com>，它就是 URL，因为它可以被“定位”，它当然也是 URI；但是如果我们只提到上面地址的一个片段，例如 www.google.com，那么由于缺少了具体协议，我们无法完成完整的定位，因此这样的东西只能被叫做一个标识符，故而只能算 URI，而非 URL。

3. 方法

HTTP 的方法反映了这个接口上将执行的行为，如果用自然语言描述，它将是一个动词。比如说，给图书馆添加一本图书，那么这个行为将是“添加”。在以 REST 风格主导的设计中，我们将使用这样的 HTTP 方法来代表增删改查（CRUD）的不同动作：

HTTP 方法	含义（CRUD）	URI 例子（123 表示资源唯一 ID）	幂等的（Idempotent）	安全的（Safe）
GET	获取资源（Retrieve）	/books/123 单个资源 /books 多个资源 /books/123/price 资源属性	Yes	Yes
POST	创建一个资源单位（Create）	/books	No	No
PUT	更新一个资源单位（Update）	/books/123	Yes	No
DELETE	删除一个资源单位（Delete）	/books/123	Yes	No

重点解释下表格的最后两列：

幂等性指的是对服务端的数据状态，执行多次操作是否和执行一次产生的结果一样。从表格中你可以看到，创建资源单位不是幂等的，执行多次就意味着创建了多个资源单位，而其它操作都是幂等的。通常来说，**幂等操作是可以被重试而不产生副作用的。**

安全性指的是该操作是否对服务端的数据状态产生了影响。从表格中可以看到，获取资源的操作是安全的，不会对资源数据产生影响，但是其它操作都是不安全的。一定条件下，**安全操作是可以被缓存的**，而不安全的操作，必定对服务端的状态产生了影响，这体现了 REST 的“状态转换”这一要素。

全栈系统的设计和优化都需要紧密围绕幂等性和安全性进行，这是两个非常重要的概念，在我们后续的学习中，你还会反复见到它们，并和它们打交道。

你看，通过这样的办法，就把 HTTP 的方法和实际对资源的操作行为绑定起来了。当然，还有一些其它方法，比较常见的有：

PATCH：和 PUT 类似，也用于资源更新，但支持的是资源单位的部分更新，并且在资源不存在的时候，能够自动创建资源，这个方法实际使用比较少。


HEAD：这个方法只返回资源的头部，避免了资源本身获取和传输的开销。这种方法很有用，经常用来检查资源是否存在，以及有无修改。

OPTIONS：这个方法常用来返回服务器对于指定资源所支持的方法列表。

4. 正文

POST 和 PUT 请求都是有 HTTP 正文的，正文的类型和 Content-Type 的选取有关，比如 JSON 就是最典型的一种格式。请不要轻视这里的 Content-Type，从本质上说，它代表了资源的表现形式，从而体现了 REST 定义中的“表现层”这一要素。

最后，回到我们实际的图书馆添加图书的问题。SOAP 添加一本书的消息，用 REST 风格的 POST 请求实现就会变成这样：

 复制代码

```
1 POST /v1/books HTTP/1.1
2 HOST: ...
3 Content-Type: application/json
4
5 {
6   "name": "...",
7   "author": "...",
8   ...
9 }
```

风格之争

看到这儿，你应该已经感受到了，SOAP 和 REST 代表了两种迥异的风格。在我们取舍技术的时候，如果没有给出具体场景和限制，我们往往是很难讲出谁更“好”，而是需要进行比较，权衡利弊的。

SOAP 明显是更“成熟”的那一个。它在数据传输的协议层面做了更多的工作，藉由 XML schema，它具备更严格的检查和校验，配合 WSDL，在真正发送请求前，几乎就对所有远程接口事无巨细的问题有了答案。但是，它的复杂度令人望而生畏，也是它最受人诟病的地方。

REST 则相反，新接口的学习成本很低，只需要知道资源名称，根据我们熟知的规约，就可以创建出 CRUD 的请求来。但是直到真正发送请求去测试为止，并没有办法百分百确定远程接口的调用是否能工作，或者说，并不知道接口定义上是否有不规范、不合常规的坑在里面。

对于互联网来说，SOAP 已经是一项“古老”的技术了，晚辈 REST 似乎更切合互联网的潮流。在大多数情况下，REST 要易用和流行得多，于是很多人都不喜欢繁琐的 SOAP 协议。**技术的发展总是有这样的规律，一开始无论问题还是办法都很简单，可是随着需求的进一步增加，解决的方法也缓慢演化，如 SOAP 一般强大而复杂，直到某一天突然掉到谷底，如 REST 一般返璞归真。**

但是别忘了，有利必有弊。首先，正是因为 REST 只是一个缺乏限制的风格，而非一个严谨的规范，有太多不明确、不一致的实现导致的问题，这些问题轻者给接口调用者带来困惑，重者导致接口调用错误，甚至服务端数据错误。

其次，REST 通过 HTTP 方法实现本身，也受到了 HTTP 方法的局限性制约。比如最常见的 GET 请求，有时需要一个复杂的查询条件集合，因此参数需要结构化，而 GET 只支持一串键值对组合的参数传递，无法适应业务需要。对于这样的问题，有一些 workaround，比如使用 POST 消息体来传递查询条件的结构体，但那已经偏离了 REST 的最佳实践，丢失了 GET 本身的优势，也带来了实现不一致等一系列问题。

最后，REST 还存在的一个更本质的问题，资源是它的核心概念，这原本带来了抽象和简洁的优势，但如今也成为了它的桎梏。或者说，前面反复提到的增删改查是它最拿手的本事，可是互联网的需求是千变万化的，远不只有简单的增删改查。有时需要一个复杂的多步操作，有时则需要一个异步事务（需要回调或者二次查询等等方式来实现），这些都没有一个完美统一的 REST 风格的解决方案。即便实现出来了，也可谓五花八门，同时失去了以往我们所熟知的 REST 的简洁优势。

互联网总在变复杂，但矛盾的是，人们却希望互联网上的交互会不断变简单。于是这引发了 REST 的流行，可即便 REST 再流行，依旧有它不适合的场景；SOAP 虽古老，依然有它的

用武之地。


对于全栈工程师或者期待修炼全栈技能的你我来说，trade-off 是永恒的话题。另外，除了 SOAP 和 REST，其实我们还有其它选择。我将在下一讲，结合实例具体介绍如何选择技术，并设计和实现远程接口。

总结思考

今天我们认识并学习了 SOAP 和 REST 这样两种截然不同的风格，前者工整、严谨和细致，后者自由、灵活和简约。两道思考题如下：

在做技术比较的时候，文中已经简单介绍了 REST 和 SOAP 的优劣，你觉得，它们各自适合怎样的业务场景呢？

有位程序员朋友在应用 RESTful 风格设计用户管理系统的接口时，“删除单个用户”功能的 URL 举例如下，你觉得有哪些问题？

 复制代码

```
1 http://xxx/deleteUser?userName=James
```

今天的内容就到这里，希望你已经享受到了技术学习的快乐，如果你还有余力，请继续学习下面的选修课堂和扩展阅读。最后，对于上面的问题，或者你对今天的学习有什么感受，欢迎在留言区和我讨论！

选修课堂：动手调用 RESTful API

学习全栈怎么能不动手实践呢，现在就让我们开始吧。有一些在线工具，预置了 REST 风格的接口服务，我们可以使用命令行去指定不同的 HTTP 方法，发送一些不同的 HTTP 请求，观察返回，通过实际的练习，相信你能够更好地理解 REST。这样的工具有很多，你可以自行搜索，也可以直接选择 [REQ | RES](#)：

Request		Response
/api/users?page=2		200
GET	LIST USERS	<pre>{ "page": 2, "per_page": 6, "total": 12, "total_pages": 2, "data": [{ "id": 7, "email": "michael.lawson@reqres.in", "first_name": "Michael", "last_name": "Lawson", "avatar": "https://s3.amazonaws.com/reqres.in/img/fake/7/avatar.png" }, { "id": 8, "email": "michael.lawson@reqres.in", "first_name": "Michael", "last_name": "Lawson", "avatar": "https://s3.amazonaws.com/reqres.in/img/fake/8/avatar.png" }, { "id": 9, "email": "michael.lawson@reqres.in", "first_name": "Michael", "last_name": "Lawson", "avatar": "https://s3.amazonaws.com/reqres.in/img/fake/9/avatar.png" }, { "id": 10, "email": "michael.lawson@reqres.in", "first_name": "Michael", "last_name": "Lawson", "avatar": "https://s3.amazonaws.com/reqres.in/img/fake/10/avatar.png" }, { "id": 11, "email": "michael.lawson@reqres.in", "first_name": "Michael", "last_name": "Lawson", "avatar": "https://s3.amazonaws.com/reqres.in/img/fake/11/avatar.png" }, { "id": 12, "email": "michael.lawson@reqres.in", "first_name": "Michael", "last_name": "Lawson", "avatar": "https://s3.amazonaws.com/reqres.in/img/fake/12/avatar.png" }] }</pre>
GET	SINGLE USER	
GET	SINGLE USER NOT FOUND	
GET	LIST <RESOURCE>	
GET	SINGLE <RESOURCE>	
GET	SINGLE <RESOURCE> NOT FOUND	

你可以使用网站上预置的请求，但我更推荐你自己写 curl 命令。比如发送一个 GET 请求，列出所有用户：

复制代码

```
1 curl -v https://reqres.in/api/users | jq
```

其中的 -v 参数可以帮助输出详尽的信息，包括请求和响应的完整信息，当然也可以不用；后面的 “|jq” 是为了让返回的 JSON 数据展示更美观，当然，你需要安装 jq。如果你没有安装，不使用 jq 管道也是完全可以的。

再比如，使用 POST 请求创建一个用户：

复制代码

```
1 curl -X POST -d '{"name":"xxx", "job":"yyy"}' -H "Content-Type: application/json" https://reqres.in/api/users
```

这里使用了 -x 参数指定其为 POST 请求，之后的 Content-Type 是必不可少的，而 JSON 形式的 user 对象则通过参数 -d 传了过去。

最后得到了这样的结果：

复制代码

```
1 ... (省略请求统计信息)
```

```
2 {  
3   "name": "xxx",  
4   "job": "yyy",  
5   "id": "585",  
6   "createdAt": "2019-07-20T22:19:49.825Z"  
7 }
```

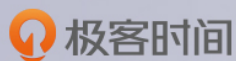
扩展阅读

W3Cschool 上的 [SOAP 教程](#)，如果你对 SOAP 不够熟悉，那么你可以参考这个简明扼要的教程。

【基础】[REST API Tutorial](#)，REST 的教程很多，这是我觉得非常简洁和清晰的一个。

ProgrammableWeb 上的 [Web API 列表](#)，排名最靠前的 10 个 API，其中有 9 个的架构风格都是 REST，这也从侧面应证了 REST 在互联网的趋势。

[REST 和 SOAP：谁更好，或者都好？](#) 这是一篇内容精悍的译文，分别介绍了适合 REST 和 SOAP 的场景。



全栈工程师修炼指南

从全栈入门到技能实战

熊燚

Oracle 首席软件工程师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

精选留言 (9)

写留言



八哥 置顶

2019-09-18

早期后台接口之间调用Web Service时用的SOAP协议多，比如订单和供应商之间接口调用等。后续RESTful风格更轻量级，开发者工程师更愿意使用。配合api文档，解决了联调过程中，接口没有定义规范等问题。

userName不是唯一的，实际无法完成删除，delete是一个删除操作，不应该暴露在URL...
展开

作者回复: 感谢回答！回答正确。

第一个问题属于开放性的，当然，也有很多可供比较的其它方面。

对于删除用户的接口设计，回答得非常全面：

- (1) 动作要以 HTTP 方法体现出来，而不是放在 URL 里面；
- (2) 要使用 userId，而不是 name，userId 才是唯一的。
- (3) 资源使用复数 “users” ，完全正确。



3



leslie

2019-09-18

RESTFUL设计删除单个用户"http://xxx/deleteUser?userName=James"的问题如下：1)get/put/post/delete不是应当接完后再加/再去跟具体的么？至少应当改成"http://xxx/delete/username/James"

2)其实这种具体的值不应当直接放到这里：我记得当年自己做开发的时候就不会允许暴露账号信息的...

展开

作者回复: 感谢回答！

严格来说，行为应当放到 HTTP 方法中去，而不是 URL 中。还有使用 userName 本身的问题，你可以看一下另一位朋友的回答。当然，现在不少接口其实并不是很符合 REST 风格的，方法放在 URL 中就是其中一个方面。至于不允许暴露账号信息是特殊的业务需要，能想到这一点非常好，但它并非 REST 的要求。



2



靠人品去赢

2019-09-20

(1) 两种都用过，之前在银行类使用过webservice，就是很典型的，就是约定好字段一个不多一个不少按照WSDL上来。后面公司用的是Rest，看文档反正你要的我都给你了，需求变更也不怕反正能满足你就是了。

(2) 参数选择不对，命名不规范，用户名称不是唯一的很可能重复，即使是唯一的，博大精深的汉字也能教育你，可能某个系统编码库不全，一些少见的字造成不必要的麻烦，...

展开 ∨

作者回复: 接口命名方面，没有统一的标准，但你可以看看第五讲扩展阅读的 AnyAPI，看看其他人是怎么做的，大多数接口服务的命名风格是一致的。



1



松松

2019-09-18

米用过，但如果米理解错的话，RESTful风格是用URI标识要操作的资源，用HTTP请求行为表示要对资源执行的动作，所以delete这个“行为”不应该出现在URI里吧，总不能get这个用户和del这个用户操作的还是不同的资源？

展开 ∨

作者回复: 对，行为不能出现在 URI 里面，你可以看一下另一位朋友 的回答。



1



Gopher

2019-09-20

RESTful 风格的API：（系统接口：正交化，统一化/标准化，结构化）

1.正交化资源（名词）和动作（动词）

2.资源的描述要统一（URI），先模块/位置，再资源/标识符，最后可以附加简单的参数（?x="y"#label）。复杂的参数应当结构化，通过Header传递

3.动作要标准化（AEMR等有限集合），个性化需求可以通过API查询接口获取/协同...

展开 ∨

作者回复: 🙏





OnFire
2019-09-19

目前外包在银行做供应链业务，soap与restful混用，新接口基本都不用soap了，像soap这种结构相对复杂的方案会逐渐被淘汰吧，restful更简单明了

展开 ▾



anginiit
2019-09-19

之前一个项目 ws和rest都有用到，rest是自己项目前后台请求使用，ws是请求第三方的接口使用。只是停留在使用阶段 还没有深入的了解过，借着这一课 认真学习下。



许童童
2019-09-18

项目中用的基本上都是REST，我觉得用SOAP的只有以前遗留下来的代码了。

作者回复: SOAP 还是常有使用的，特别是一些电信软件、传统软件等领域。当然，确实 REST 更常见。



sky
2019-09-18

项目中基本上用的都是restful。soap用着别扭

展开 ▾

作者回复: 为什么别扭? :)

