

05 | 权衡的艺术：漫谈Web API的设计

2019-09-20 四火

全栈工程师修炼指南

[进入课程 >](#)



讲述：四火

时长 21:02 大小 14.46M



你好，我是四火。

今天，我们该根据之前所学，来谈谈具体怎样设计 Web API 接口了。我们围绕的核心，是“**权衡**” (**trade-off**) 这两个字，事实上，它不只是 Web API 接口设计的核心，还是软件绝大多数设计问题的核心。

我们说“没有银弹”，是因为没有一种技术可以百搭，没有一种解决方案是完美的，但一个优秀的全栈工程师，是可以从琳琅满目的同类技术中，因地制宜地选择出最适合的那一个。

概念

在一切开始之前，我们先来明确概念。什么是 Web API？

你应该很熟悉 API，即 Application Programming Interface，应用程序的接口。它指的就是一组约定，不同系统之间的沟通必须遵循的协议。使用者知道了 API，就知道该怎样和它沟通，使用它的功能，而不关心它是怎么实现的。

Web API 指的依然是应用程序接口，只不过它现在暴露在了 Web 的环境里。并且，我们通常意义上讲 Web API 的时候，无论是在 B/S（浏览器 / 服务器）模型还是 C/S（客户端 / 服务器）模型下，往往都心照不宣地默认它在服务端，并被动地接受请求消息，返回响应。

通常一个 Web API 需要包括哪些内容呢？

回答这个问题前，让我们先闭上眼睛想一想，如果没有“Web”这个修饰词，普通的 API 要包括哪些内容呢？嗯，功能、性能、入参、返回值.....它们都对，看起来几乎是所有普通 API 的特性，在 Web API 中也全都存在。而且，因为 Web 的特性，它还具备我们谈论普通 API 时不太涉及的内容：

比如承载协议。这里可以有多个协议，因为协议是分层的。HTTP 协议和 TCP 协议就是并存的。

再比如请求和响应格式。Web API 将普通 API 的方法调用变成了网络通信，因此参数的传入变成了请求传入，结果返回变成了响应传出。

正是有了 Web API，网络中的不同应用才能互相协作，分布式系统才能正常工作，互联网才能如此蓬勃发展。而我们，不能只停留在“知道”的层面，还要去深入了解它们。

Web API 的设计步骤

关于 Web API 的设计步骤，不同人有不同的理解，争论不少，涉及到的内容也非常广泛。这里我综合了自己的经验和观点进行介绍，希望你能有所启发。

第一步：明确核心问题，确定问题域

和普通的 API 设计、程序的库设计一样，Web API 并不是东打一枪，西打一炮的。想想写代码的时候，我们还要让同类型的方法，以某种方式组织在类和对象中，实现功能上的内聚呢，一个类还要遵循单一职责的原则呢。

因此，一组 Web API，就是要专注于一类问题，核心问题必须是最重要的一个。

在上一讲中我举了个图书管理系统的例子，那么可以想象，图书的增删改查 API 就可以放到一起，而如果有一个新的 API 用于查询图书馆内部员工的信息，那么它显然应该单独归纳入另外的类别中，甚至是另外的系统中。

第二步：结合实际需求和限制，选择承载技术

这里有两件事情需要你考虑，一个是需求，一个是限制。我们虽然经常这样分开说，但严格来说，限制也是需求的一种。比方说，如果对网络传输的效率要求很高，时延要求很短，这就是需求，而且是非功能性的需求。

大多数功能性的需求大家都能意识到，但是一些非功能性的需求，或者一些“限制”就容易被忽略了。比如说，向前的兼容性，不同版本同时运行，鉴权和访问控制，库依赖限制，易测试性和可维护性，平滑发布（如新老接口并行），等等。

再来说说承载技术。承载技术指的是实现接口，以及它的请求响应传输所需要使用到的技术集合，比如 HTTP + JSON。我们前面提到的要求网络传输效率高、时延短，[Protobuf](#) 就是一个值得考察的技术；但有时候，我们更需要消息直观、易读，那么显然 Protobuf 就不是一个适合的技术。这里我们通过分析技术优劣来做选择，这就是权衡。

虽说 Web API 主要的工作在服务端，但在技术分析时还需要考虑客户端。特别是一些技术要求自动生成客户端，而有些技术则允许通过一定方式“定制”客户端（例如使用 DSL，Domain Specific Language，领域特定语言）。

第三步：确定接口风格

技术的选择将很大程度地影响接口的风格。

还记得我在上一讲介绍的 SOAP 和 REST 的例子吗？那就是接口风格比较的一个典型示例。请不要小看这两个字，“风格”包含的内容很多，大到怎样划分功能，小到接口的命名，都包括在内。在实际设计中，我们很少正面地去谈论具体的风格，但我们都有意无意地将其考虑在内。这里我举几个比较重要的例子，通过它，你会了解到权衡其实无处不在。

角度一：易用性和通用性的平衡，或者说是设计“人本接口”还是“最简接口”。

比如一个图书管理的接口，一种设计是让其返回“流行书籍”，实际的规则是根据出版日期、借阅人数、引进数量等等做了复杂的查询而得出；而另一种设计则是让用户来自行决定

和传入这几个参数，服务端不理解业务含义，接口本身保持通用。

前者偏向“易用”，更接近人的思维；后者偏向“通用”，提供了最简化的接口。虽说多数情况下我们还是会见到后者多一些，但二者却不能说谁对谁错，它们实际代表了不同的风格，各有优劣。

角度二：接口粒度的划分。

比如用户还书的过程包括：还书排队登记、检查书本状况、图书入库，这一系列过程是设计成一个大的接口一并完成，还是设计成三个单独的接口分别调用完成？

其实，这二者各有优劣。**设计成大接口往往可以增加易用性，便于内部优化提高性能（而且只需调用一次）；设计成小接口可以增加可重用性，便于功能的组合。**

你可能会想，两种方式都保留，让用户去选择不行吗？

行，但那样给双方带来好处的同时，也带来了更多的问题，除了风格的不一致，接口也不再是正交的，而是有一定重叠性的，并且更多的接口意味着更多的开发和维护工作。这些接口要像是一个人设计出来的，而不是简单的组合添加，**风格统一也是一致性的一种表现**。因此，多数情况下我们不那么做。你看，这又是权衡。

但是，我说的是“多数情况下”我们不那么做。在一些极端情况下，我们是会牺牲掉一致性，保留冗余的。


我举一个 JDK 的例子。JDK 的 HashTable 有一个 containsValue 方法，还有一个 contains 方法，二者功能上完全一样，之所以搞这样两个完全一样的方法，正是由于历史原因造成的。JDK 1.2 才正式引入 Java Collections Framework，抽象了 Map 接口，才有了 containsValue 方法，而之前的方法因为需要保持向下兼容而无法删除，也是无可奈何。同样，这也是权衡。

第四步：定义具体接口形式

在上面这三步通用和共性的步骤完成之后，我们就可以正式跳进具体的接口定义中，去确定 URL、参数、返回和异常等通用而具体的形式了。还记得上一讲中对 REST 请求发送要点的分解吗？在它的基础上，我们将继续以 REST 风格为例，进行更深刻的讨论。

1. 条件查询


我们在上一讲的例子中使用 HTTP GET 请求从图书馆获取书本信息，从而完成增删改查中的“查”操作：

 复制代码

```
1 /books/123
2 /books/123/price
```

分别查询了 ID 为 123 的图书的全部属性，和该图书的价格信息。


但是，实际的查所包含的内容可远比这个例子多，比如不是通过 ID 查询，而是通过条件查询：

 复制代码

```
1 /books?author=Smith&page=2&pageSize=10&sortBy=name&order=desc
```

你看条件查询书籍，查询条件通过参数传入，指定了作者，要求显示第二页，每页大小为 10 条记录，按照书名降序排列。

除了使用 Query String（问号后的参数）来传递查询条件，多级路径也是一种常见的设计，这种设计让条件的层级关系更清晰。比如：

 复制代码

```
1 /category/456/books?author=Smith
```

它表示查询图书分类为“艺术”（编号为 456）的图书，并且作者是 Smith。看到这里，你可能会产生这样两个疑问。


疑问一：使用 ID 多不直观啊，我们能使用具体名称吗？

当然可以！可以使用具备业务意义的字段来代替没有可读性的 ID，但是这个字段不可重复，也不宜过长，比如例子中的 category 就可以使用名称，而图书，则可以使用国际标准书号 ISBN。于是 URI 就变成了：

 复制代码

```
1 /category/Arts/books?author=Smith
```

疑问二：category 可以通过 Query String 传入吗？比如下面这样：

 复制代码


```
1 /books?author=Smith&category=Arts
```

当然可以！“category”可以放置在路径中，也可以放置在查询参数串中。这是 REST 设计中的一个关于设计上合理冗余的典型例子，可以通过不同的方式来完成相同的查询。如果你学过 Perl，你可能听过 “There’s more than one way to do it” 这样的俗语，这是一样的道理，也是 REST 风格的一部分。

当然，从这也可以看出上一讲我们提到过的，REST 在统一性、一致性方面的约束力较弱。

2. 消息正文封装

有时候我们还需要传递消息正文，比如当我们使用 POST 请求创建对象，和使用 PUT 请求修改对象的时候，我们可以选择使用一种技术来封装它，例如 JSON 和 XML。通常来说，既然我们选择了 REST 风格，我们在相关技术的选择上也可以继续保持简约的一致性，因此 JSON 是更为常见的那一个。

 复制代码

```
1 {  
2   "name": "...",  
3   "category": "Arts",  
4   "authorId": 999,  
5   "price": {  
6     "currency": "CNY",  
7     "value": 12.99  
8   },
```



```
9   "ISBN": "...",
10  "quantity": 100,
11  ...
12 }
```

上面的消息体内容就反映了一本书的属性，但是，在设置属性的时候，往往牵涉到对象关联，上面这个小小的例子就包含了其中三种典型的方式：

传递唯一业务字段：例如上面的 `category` 取值是具备实际业务意义的 “Arts” ；

传递唯一 id：例如上面的 `authorId`，请注意，这里不能传递实际作者名，因为作者可能会重名；

传递关联对象：例如上面的 `price`，这个对象通常可以是一个不完整的对象，这里指定了货币为人民币 CNY，也指定了价格数值为 12.99。

3. 响应和异常设计

HTTP 协议中规定了返回的状态码，我想你可能知道一些常见的返回码，大致上，它们分为这样五类：

1xx：表示请求已经被接受，但还需要继续处理。这时你可能还记得在 [\[第 03 讲\]](#) 中，我们将普通的 HTTP 请求升级成为 WebSocket 的过程，101 就是确认连接升级的状态码。


2xx：表示请求已经被接受和成功处理。最常见的就是 204，表示请求成功处理，且返回中没有正文内容。

3xx：表示重定向，请客户端使用重定向后的新地址继续请求。其中，301 是永久重定向，而 302 是临时重定向，新地址一般在响应头 “Location” 字段中指定。

4xx：表示客户端错误。服务端已经接到了请求，但是处理失败了，并且这个锅服务端不背。这可能是我们最熟悉的返回码了，比如最常见的 404，表示页面不存在。常见的还有 400，表示请求格式错误，以及 401，鉴权和认证失败。

5xx：表示服务端错误。这回这个处理失败的锅在服务端这边。最常见的是 500，通用的和未分类的服务端内部错误，还有 503，服务端暂时不可用。

错误处理是 Web API 设计中很重要的一部分，我们需要告知用户是哪个请求出错了，什么时间出错了，以及为什么出错。比如：

 复制代码

```
1 {  
2     "errorCode": 543,  
3     "timeStamp": 12345678,  
4     "message": "The requested book is not found.",  
5     "detailedInfomation": "...",  
6     "reference": "https://...",  
7     "requestId": "..."  
8 }
```

在这个例子中，你可以看到上面提到的要素都具备了，注意这里的 `errorCode` 不是响应中的 HTTP 状态码，而是一个具备业务意义的内部定义的错误码。在有些设计里面，也会把 HTTP 状态码放到这个正文中，以方便客户端处理，这种冗余的设计当然也是可以的。

总结思考

还记得我们是通过怎样的步骤来设计 Web API 的吗？其实可以总结为八个字：**问题、技术、风格和定义**，由问题到实现，由概要到细节。

问题域往往比较好确定，技术选型在需求和限制分析清楚的情况下也不难做出选择，但是接口风格往往就考验 API 的设计功底了。在这部分中，易用性和通用性的平衡，接口粒度的控制，是非常重要的两个方面，这是需要通过不断地“权衡”来确定的。至于在接口定义的步骤中，细节很多，更多的内容需要我们在实践中多参考一些优秀的接口实现案例，逐渐积累经验。

这一讲通篇都不断地提到了“权衡”，现在我来提一个关于权衡的小问题：

在介绍 REST 的参数传递的时候，我们讲了 `category` 参数传递的两种方式，一种是通过路径传递，一种是通过 Query String 的参数传递。你觉得哪些参数适合使用第一种，哪些参数更适合使用第二种？

如果你还有余力，那我再提一个接口设计方面的问题：

我们提到了 REST 风格下，我们使用 HTTP 的不同方法来应对增删改查这样不同的行为。但是，互联网的业务是很复杂的，有时候操作并非简单的增删改查，这种情况会考验我们的 REST 设计功底。比如说，我们要给银行转账，即钱从一个人的账下转移到另一个人的账下，这样的复杂行为不属于增删改查中的任何一项，我们是否能使用 REST 风格来设计这样的转账接口呢？

到今天为止，第一章，也就是“网络协议和 Web 接口”的内容我们就讲完了。网络协议部分，我们以 HTTP 为核心，介绍了它的特性和发展进程，展示了 TLS 连接建立和证书验证的原理，深入了 Comet 和 WebSocket 等服务端消息推送技术，并通过抓包分析等实践，进一步加深了理解。Web 接口部分，我们结合图书馆的实例，学习和比较了 SOAP 和 REST 的实现和风格，并一步一步梳理了 Web 接口设计的过程。

最后，对于上面的问题你有什么答案，或是对于这一章的内容有什么思考和疑问，欢迎你在留言区中畅所欲言，我们一起探讨，相信能碰撞出很多新的火花。

扩展阅读

【基础】[HTTP 状态码](#) 和 [HTTP 头字段](#)，我们在工作中会反复和各式各样的状态码和请求、响应中的头部字段打交道，因此通读并熟知一些常见的状态码是很有必要的。关于 HTTP 状态码，有人把一些常见的状态码形象地对应到[猫的照片](#)，或许能帮助你记忆，当然，如果你喜欢狗，那你可以看看[这个](#)。

[Any API](#)，我们不能光闭门造车，还要去学习其它网站的 Web API 设计，了解互联网上大家都是怎么做的。我们学习它们的实现，但请不要盲目，有不少接口由于种种原因，设计有一些亟待商榷的地方，请带上你批判的眼光。

[Richardson Maturity Model](#)，这篇会有一些深度，大名鼎鼎的马丁·福勒（Martin Fowler）的文章，讲 REST 的一种成熟度模型，里面划分了从 0 级到 3 级这样 4 种成熟度级别，这种分级方式被一些人奉为圭臬。

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。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 04 | 工整与自由的风格之争：SOAP和REST

下一篇 06 | 特别放送：北美大厂如何招聘全栈工程师？

精选留言 (8)

写留言



panlatent 置顶

2019-09-21

结合上一讲内容，想问问老师对 graphql 的看法，以及如何权衡 graphql / rest 两种api。

作者回复：这其实是个很好的问题。

我对 GraphQL 的理解简单来说是这样的，仅供参考。它本质上是一种声明式的 DSL，把接口逻辑从服务端拿到客户端来，客户端来决定做什么查询，执行什么操作，资源的概念被彻底弱化了。

和基于资源的 REST 相比，因为可以更加细粒度地控制需要什么数据，减少了多次调用或者是不需要的数据返回造成的开销。当然，它也有许多弱点，比如复杂性更高（客户端需要理解复杂的业务数据模型），不容易使用缓存等等。

我觉得它应该是 REST 风格的一种补充，而不是绝对的替代。有很多场景，比如复杂的数据查询，使用 GraphQL 可以做得很灵活，且有比较高的效率。而多数业务场景，REST 确是更好的选择。



3



tt 置顶

2019-09-20

1、突然有个想法，通过路径传递参数相当于非web系统设计的定长报文或固定分隔符报文，可读性比较差，可以更多的实现对外部的信息隐藏。

而通过query参数传递相当于URL承载的键值对报文，可读性比较好。

...

展开

作者回复: 感谢回答。

关于 1，我觉得你说得很好。路径传递“隐含”了这个 key，而使用 Query String 的键值对的方式，则显式指定了 key。从这个角度来说，确实后者更为“明确”。

但是，“可读性”并不一定是指定了 key 的更好，可读性毕竟是一个和个人理解密切相关的判断。通常在路径层次较短，且路径的定义符合人的认知的时候（比如“资源/分类/唯一ID”，这样由大到小的递进），路径传递参数的方式也具有很高的可读性。

在使用框架处理的难度方面，我认为这两种方式并没有太大区别。

另外，使用 Query String 的方式灵活性上要更高，比如可以通过合理的配置自动构造和注入一个复杂的参数对象，这方面我们会在下一章学到。

关于 2，说得非常好，这种把一个复杂行为转变为可进行 CRUD 的“资源”，就是一种很有效的处理方法。

对于你最后关于鉴权和授权的问题，我计划在第 5 章谈到。



2



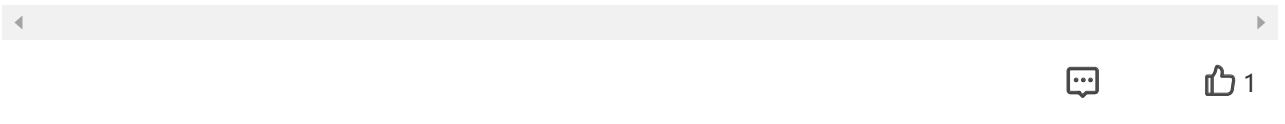
alan

2019-09-22

老师，为啥说204是最常见的返回码？不是200吗？

展开

作者回复: 看场景了。对于 Web API 来说, 如果是一个命令接口, 常常见到返回 204, 表示操作成功, 但不需要消息体。



jxs1211

2019-09-23

sql建议使用pk进行查询, 如果使用name这样的业务相关的字段进行查询, 可以提高理解性, 是否也应该加索引, 提高查询效率

展开 ∨



leslie

2019-09-23

问题、技术、风格和定义: 这个似乎不止在web api的设计中会有这个问题, 我觉得在现在的数据系统/存储中间件 的设计中其实现在典型的出现了; 毕竟现在一旦设计软件就不是过去传统的C/S或B/S。

昨天一个同行问我一个问题: 我们有前端开发, 我们需要架构师; 我就反问了一句 “你们需要的是什么架构?” 分析“问题”、寻找 “技术”、选择 “风格”、定义“需求”。架构...

展开 ∨



饭团

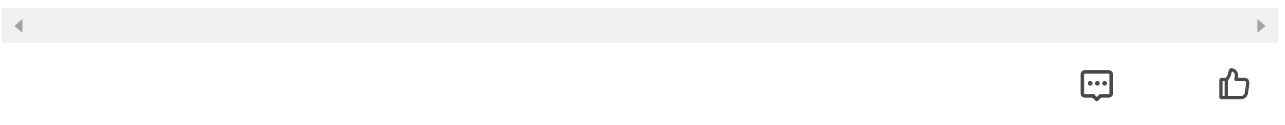
2019-09-21

我感觉路径选择不够灵活, 因为他默认了各级参数所对应的含义! 在需要加入参数或者调整参数顺序的时候就会带来诸多不便! 我看到的大部分对外接口都是在指定到特定目录后 (应该是该功能的功能模块), 参数通过或则query_string的形式传输过来!

也就是说如果一个接口使用纯路径选择, 是不是就适合于功能较为简单, 参数偏少的情况! 而大部分情况都是混合使用

展开 ∨

作者回复: 🍡, 说得很好。



anginiit

2019-09-20

一直只关心技术 对于设计思想层面的考虑为0, 这篇文章对我来说读的一头雾水。学习,

学习。。。



编程爱好者

2019-09-20

全栈工程师不是技术方面的问题，而是灵魂是否有趣的问题，很喜欢作者结尾分享的链接，这门课程是告诉我们如何成为一个有趣的程序员

展开 ∨

作者回复: 感谢评价!

