链滴

api 接口有必要 restful 吗?

作者：ghostsf
原文链接：https://ld246.com/article/1587629717195
来源网站：链滴

<h3 id="REST-的起源-">REST 的起源？ </h3>
REST 流行起来是因为 Roy Fielding 他 2000 年的博士论文 Architectural Styles and the Design of Network-based Software Architectures 中介绍并荐了它。Roy 因为他对 web 开发，特别是 HTTP 规范的贡献而闻名。
<h3 id="什么是-RESTful-APIs--">什么是 RESTful APIs？ </h3>
REST 即表述性状转移，是一种用于构建可扩展 web 服务的架构风格。Roy 提倡使用他在 HTTP 标准 中参与制定的请方法来赋予 HTTP 请求语义。
因此当使用 REST 时，下面的 HTTP 请求都有不同的语义：

<code>GET /object/list</code>
<code>POST /object/list</code>
<code>PUT /object/list</code>

这只是一部分 HTTP 请求方法。完整的列表如下：<code>CONNECT</code>, <code>DELET</code>, <code>GET</code>, <code>HEAD</code>, <code>OPTIONS</code>, <code>PTCH</code>, <code>POST</code>, <code>PUT</code>, <code>TRACE</code>。可以原你从没有听说过其中的某些方法，因为它们中的一部分也一直都没得到客户端或服务端的支持。
Roy 同样主张使用 他参与制定的 HTTP 响应状态码来传达响应的语义。共有 38 种 HTTP 响应状态。
<table>
<thead>
<tr>
<th>Method</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 OK</td>
<td>201 Created</td>
</tr>
<tr>
<td>202 Accepted</td>
<td>203 Not authorized</td>
</tr>
<tr>
<td>204 No content</td>
<td>205 Reset content</td>
</tr>
<tr>
<td>206 Partial content</td>
<td></td>
</tr>
<tr>
<td>300 Multiple choice</td>
<td>301 Moved permanently</td>
</tr>
<tr>
<td>302 Found</td>

```
<td>303 See other</td>
</tr>
<tr>
<td>304 Not modified</td>
<td>306 (unused)</td>
</tr>
<tr>
<td>307 Temporary redirect</td>
<td></td>
</tr>
<tr>
<td>400 Bad request</td>
<td>401 Unauthorized</td>
</tr>
<tr>
<td>402 Payment required</td>
<td>403 Forbidden</td>
</tr>
<tr>
<td>404 Not found</td>
<td>405 Method not allowed</td>
</tr>
<tr>
<td>406 Not acceptable</td>
<td>407 Proxy auth required</td>
</tr>
<tr>
<td>408 Timeout</td>
<td>409 Conflict</td>
</tr>
<tr>
<td>410 Gone</td>
<td>411 Length required</td>
</tr>
<tr>
<td>412 Preconditions failed</td>
<td>413 Request entity too large</td>
</tr>
<tr>
<td>414 Requested URI too long</td>
<td>415 Unsupported media</td>
</tr>
<tr>
<td>416 Bad request range</td>
<td>417 Expectation failed</td>
</tr>
<tr>
<td>500 Server error</td>
<td>501 Not implemented</td>
</tr>
<tr>
<td>502 Bad gateway</td>
<td>503 Service unavailable</td>
</tr>
```

<tr>
<td>504 Gateway timeout</td>
<td>505 Bad HTTP version</td>
</tr>
</tbody>
</table>
由于各种原因，RESTful API 可能比上面显示的更复杂。
<h3 id="RESTful-APIs-已知的主要问题">RESTful APIs 已知的主要问题</h3>
在许多方面，例如内容交付上，REST 都是一个不错的机制，它也很好得为我们服务了二十多年但是是时候打破沉默了，承认 RESTful API 可能是在 web 软件中广泛使用的最糟糕的想法之一。</p

我们很快将看到更好的构建互联网 API 的方式，但在完全领会这个解决方案之前，我们首先得知 RESTful APIs 的 5 个主要问题，它们使 API 昂贵，冗余，并且容易出错。
<h3 id="问题--1--关于-RESTful-API-是什么-几乎达不成共识">问题 #1: 关于 RESTful API 是什么几乎达不成共识</h3>
没有人能够在全部的请求方法，实体和响应状态码的真正含义上达成一致。
设想一下，例如，我们应该用 <code>200 OK</code> 表示成功的更新了一条记录还是要用 <code 201 Created</code>？看上去我们应该用 <code>250 Updated</code>，但是这个状态码并不在。而且谁能向我解释一下 <code>417 Expectation failed</code> 的真正含义。当然指除了 Roy 之外的人。
HTTP 方法和响应状态码的词汇过于模糊和不完整，无法就其语义达成一致。至少在我所知道的没有任何管理机构能召集大家一起把事情讲清楚。一个公司定义的 <code>200 OK</code> 必然与一家公司定义的有细微且令人讨厌的区别。这意味着一个 RESTful 模式不能像我们想要的那样可以预。
<h3 id="问题--2--REST-词汇的支持不完全">问题 #2: REST 词汇的支持不完全</h3>
即使就 REST 的各个语义达成一致，我们还会遇到另一个实际的问题：大部分客户端和服务器程不能完全支持 HTTP 协议 里定义的动词或响应状态码。例如，大部分浏览器对 <code>PUT</code> 或 <code>DELETE</code> 仅提供有限的支持。并且很多服务端程序通常也不能正确地实现这些方。
那么我们如何突破这些限制呢？一种常用的方法是在浏览器的表单里嵌入预期的请求方法。这意着 REST 请求现在包含：

一个 HTTP 请求方法, 如 <code>POST</code>
一个请求地址, 如 <code>/object/list</code>
一个我们实际期望的方法，嵌入在请求的实体中，如 <code>DELETE</code>
请求实体本身，如表单字段数据

响应状态码的处理也好不到哪去。不同的 web 浏览器 (或者服务器) 对响应状态码的解释也经常不一样。例如，如果遇到 <code>307 Temporary redirect</code> 状态码，一个浏览器可能允许户端 JavaScript 在重定向之前处理响应并取消它，另一个浏览器可能就禁止客户端 JavaScript 处理应。所有程序中真正可靠的响应状态码只有 <code>200 OK</code> 和 <code>500 Internal serve error</code>。除此之外，支持程度一个比一个糟糕。因此，我们也常常可以看到在返回的的实体嵌入有实际想要的响应状态码。
当然，即使我们能让每个人都对什么是 REST 达成一致，并神奇地修复所有已连入互联网的程序对 RET 支持的缺陷。
<h3 id="问题--3--REST-词汇对-APIs-来说不够丰富">问题 #3: REST 词汇对 APIs 来说不够丰富</h>
REST 的方法和响应状态码太过于局限，无法有效地表示所有应用程序所需的各种请求和响应。一下，我们创建了一个应用程序，并要向 HTTP 客户端 发送一个 “render complete” 响应。我们能使用 HTTP 响应状态码，因为它不存在并且 HTTP 不能扩展。
还有另外一个问题：HTTP 响应状态码 (200, 201, 202 等) 是和 HTTP 请求方法 (<code>GET</ode>, <code>POST</code> 等)没有直接联系的数字，而我们的实体载荷通常是 JSON 格式。所执行 REST 请求就好像发送一个目的地是斯瓦希里却包含英语内容的包裹，并通过打鼓声来获得交付认。这种复杂性往往会造成巨大的困惑和错误。它给我们带来了下一个问题：调试。

<h3 id="问题--4--RESTful-APIs-非常难调试">问题 #4: RESTful APIs 非常难调试</h3>
如果你曾经用过 RESTful API，你大概知道他们几乎难以调试。这是因为我们必须查看 7 个不同地方来整理出一个完整的事务中发生了什么：

HTTP 请求方法, 如 <code>POST</code>
请求地址, 如 <code>/object/list</code>
请求实体中嵌入的实际期望的方法，如 <code>DELETE</code>
请求实体中的实际数据，如 表单数据
响应状态码，如 <code>200 OK</code>
响应实体中嵌入的实际期望的响应状态码，如 <code>206 Partial content</code>
响应实体中的实际数据

我们的问题不仅有两个 REST 词汇上的限制，大家就语义达不成一致。现在还有需要查找 7 个不的地方才可能完全理解和调试一个事务。唯一还能使这更糟的事是，REST 完全绑定到了一个协议上而不适用于任何其他的协议。当然，这也是我们的下一个问题。
<h3 id="问题--5--RESTful-APIs-通常绑定在-HTTP-上">问题 #5: RESTful APIs 通常绑定在 HTTP</h3>
RESTful API 打破了良好通信的一个基本原则：消息内容应该独立于传输通道。将两者混合来迷读者是一个历史悠久的方法。
将 HTTP 协议 与事务的意义混合使得 RESTful API 完全不可移植。将 RESTful API 从 HTTP 迁移到其他传输协议上需要从 7 个不同的地方解构和重组信息，我们再使用这些信息对 RESTful 请求的全含义进行编码。