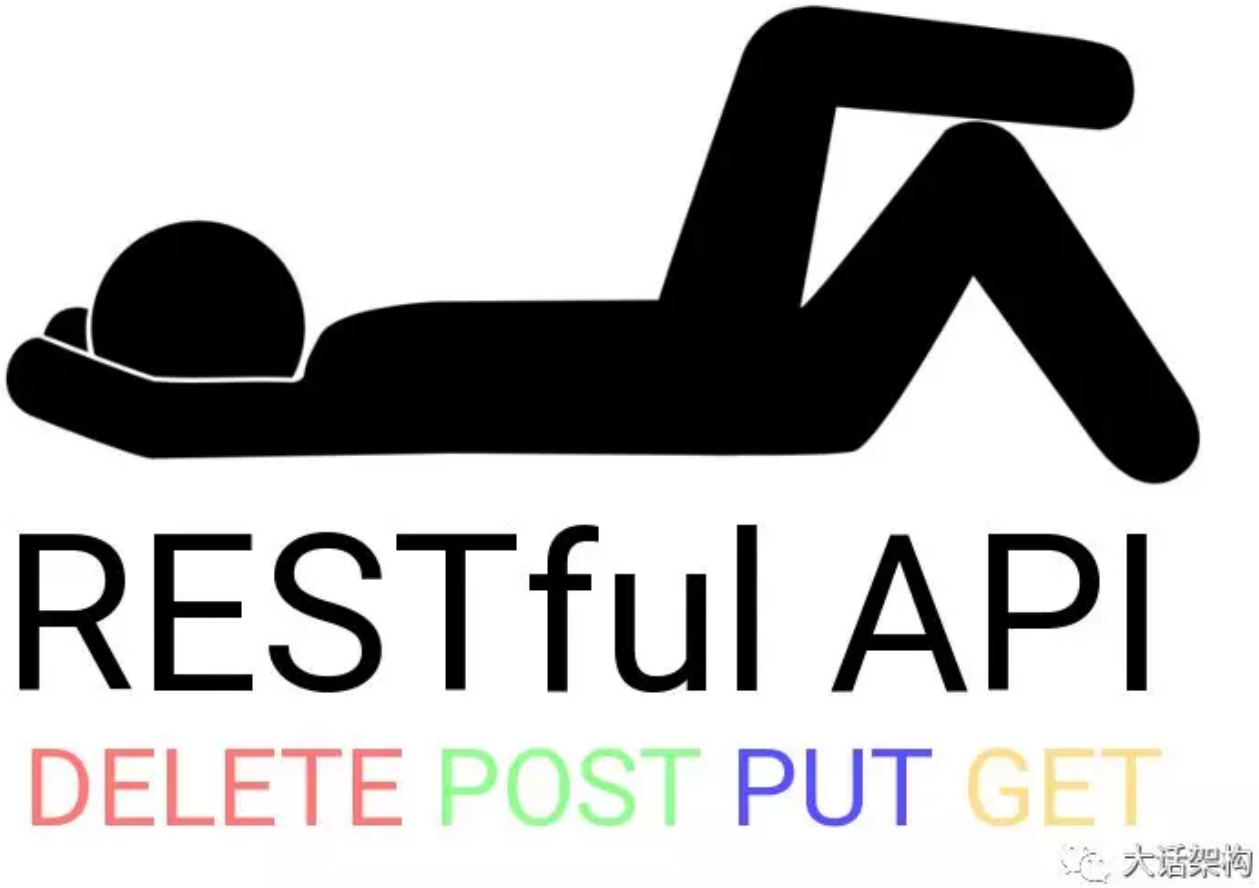
RESTful API 设计最佳实践

原创：Hanson 大话架构 3天前



前言

互联网 + 时代，RESTful API成为越来越重要的客户端和服务器端交互的形式。 但是，在人类完全能够理解HTML文档的情况下，这不适用于需要特定信息的机器。 这就是为什么每个体面的互联网公司的Web服务都有一个API，并可能将它用于他们的网站、应用程序、集成和外部服务。 但不幸的是，太多的 API 隐晦不清，难以使用。 本文将提供一些关于设计遵循Web约定的RESTful，超媒体API的实用建议。

使用URL作为ID

API中的每个概念都应该有自己的URL。 URL应该既作为标识符又作为定位器：它是一个东西的标识，它提供了一种获取有关该事物的信息的方法。

首先，URL使您的响应更容易导航。当表示社交媒体帖子的JSON对象引用某个具有18EA91FB19标识的作者时，您不知道在哪里可以找到该作者。您需要阅读API文档，发现作者的端点并撰写您的请求。如果ID是URL，您将立即知道将请求发送到何处。这不仅适用于人类，也适用于机器，因为它们无法读取您的API文档 - 但它们可以导航URL。

其次，URL不仅是单个系统中的唯一标识符，而且在不同系统中也是唯一的。域名负责处理。这意味着您可以跨多个系统使用数据。这是使链接数据非常棒的属性之一。

确保您的网址（和ID）稳定。酷URI不会改变。如果他们确实需要更改，请确保将旧网址重定向到新网址。没有人喜欢断链。

您的API端点就是您的网站

您不需要API的子域，例如 api.example.com 或子路径，例如example.com/api 。您的端点应该是您网页的根目录：example.com。

这很有用，因为如上所述，URL应该是作为单个资源的定位符的标识符。无论是某人正在寻找HTML版本还是例如资源的JSON表示，他都应该能够使用相同的URL。这使您的API更易于使用，因为浏览您网站的人可以随时了解如何以其他格式访问相同的资源。

但是，如果URL不会在格式上发生变化，那么您如何申请正确的URL？这就是HTTP内容协商派上用场的地方。客户端可以在Accept HTTP标头中发送有关要接收的内容类型的首选项。 Web浏览器的默认标头是text / HTML，但对于大多数API，诸如application / json之类的机器可读设置更合适。

但是API版本怎么样？我们希望我们的网址不会更改，因此我们不应对不同的API版本使用不同的网址。解决方案同样是使用HTTP标头。在请求中使用api-version标头或特定的Mime类型。

在URL路径中使用合理的层次结构

拥有合理的URL层次结构不仅对您的网站很重要，对您的API也很重要 - 特别是如果您的API结构类似于您的网站结构。 尝试提出合理的URL策略，与您的同事讨论，并在开发过程的早期完成所有这些。

一些比较简洁的事情：

- 从大到小，从通用到特定。
- 用户应该能够删除URL的最后部分并到达父资源。
- 让层次结构反映导航网站的用户体验。
- 尽量保持网址尽可能短。
- 人类可读的URL更容易理解和分享，它们非常适合SEO。
- 酷URI不会改变。 遗漏任何可能发生变化的内容，例如作者，文件扩展名，状态或主题。

正确使用查询参数

URI规范告诉我们仅将查询参数用于非分层数据。

不要使用查询参数来标识资源; 使用路径。

- 坏的实践：example.com/posts?id=123
- 好的实践：example.com/posts/123

将查询参数用于限制，排序，过滤和其他修饰符等可选项：

- 好的实践：example.com/posts?limit=30
- 好的实践：example.com/posts/123?show_hidden=true

使用HTTP方法

不是为各种类型的操作设置一堆端点，而是为应用程序中的每个资源使用一个URL。使用HTTP方法区分操作。

- 坏的实践：GET example.com/showPost/123
- 坏的实践：GET example.com/removePost/123
- 好的实践：GET example.com/posts/123
- 好的实践：DELETE example.com/posts/123

旨在阅读内容，创建内容或编辑内容的请求之间存在很大差异。确保正确使用GET，POST，PUT和PATCH HTTP方法。GET和PUT操作是幂等的，这意味着请求可以重复多次而没有副作用。这种区别很重要，因为它告诉客户端是否可以在发生错误时再次尝试。它还有助于缓存，因为只有GET请求可以缓存。

如果要提供表单来删除或编辑资源，该表单将与资源本身不同，因此需要一个单独的URL。一个很好的约定是将表单资源嵌套在资源本身下面。这样，如果用户想要编辑该资源，则用户只需添加/编辑该URL。

- 好的实践：GET example.com/posts/123/remove
- 好的实践：GET example.com/posts/123/edit

使用HTTP状态代码

几乎所有类型的错误消息都可以在现有的HTTP状态代码中进行分类。 这些不仅对人类有用，尤其对机器有用。 状态代码的解析速度远远快于正文。 另一个优点是它们是标准化的，因此客户端库可能知道状态代码代表什么。 您不必支持每一个，但至少要确保使用以下五个类别：

- 1xx：信息 - 只是让你知道
- 2xx：成功 - 一切都好
- 3xx：重定向 - 您的内容在其他地方
- 4xx：客户端错误 - 你做错了什么
- 5xx：服务器错误 - 我们做错了什么

为JSON添加上下文

假设您使用JSON作为序列化格式，您可以使用@context。 @context对象是一个非常小的想法，可以使您的API更具自我描述性。 @context对象描述了JSON中各种键实际代表的内容。 它提供了可以找到定义的链接。 确保您的所有ID都是实际链接，并包含您的上下文。 现在你所有的JSON都变成了JSON-LD，它是链接数据。 这意味着您的JSON数据现在可以转换为RDF（Turtle，N3，N-triples等），这意味着它变得更加可重用。 请记住，您使用的链接最好解析为一些解释您的概念所代表的内容的文档。 找到相关概念的一个很好的起点是 schema.org。

提供各种序列化选项

在序列化选项中尽可能灵活。 对于许多MVC框架，添加新序列化程序所需的工作量并不是那么糟糕。 例如，我们为 Ruby on Rails 编写了一个库，以序列化为JSON-LD，RDF/XML，N3，N-triples和Turtle。 使用上述HTTP接受标头来处理内容协商。

标准化索引页面和分页功能

您可能需要使用分页的索引页面。 怎么处理？ 分页不是一个微不足道的问题，但幸运的是，你并不是第一个遇到它的人。 不要试图重新发明轮子并使用已存在的东西，例如W3C活动流集合或Hydra集合。

不需要API密钥

您的默认API（HTML版）不需要，因此您的JSON API也不需要。 使用速率限制来确保您的服务器不会爆掉。 当然，您仍然可以使用API密钥或身份验证来访问API的特殊部分。

为API文档指定 doc. 子域

这是一个聪明的小主意：在doc.example.com上提供您的API文档。 如果用户想知道你的api如何适用于某个页面，他只需添加doc。 在他当前的URL前面。 向用户显示一个页面，告诉我们如何在该路径上使用API。

定义一套标准的返回体数据结构

对于所有的RESTful HTTP请求定义一套标准的返回结构体，前端可以根据这样的固定格式做标准化的解析，对于系统的可维护性起到很大的帮助。这个结构体里应该包含返回的具体资源，结果状态码和错误原因（如果有的话）。对于返回的资源，数据类型也尽量做到统一，比如日期，枚举类型都返回统一的数据类型避免不同的API对同一种数据有不同的处理方式。资源属性尽量做到可读也能大大减少前后端的沟通成本。

RESTful API的版本控制

一个简单的做法是直接在URL中插入版本号，这样可以允许多个版本的API同时运行。在已经发布的版本中尽量做到向后兼容，包括URL和参数，对于返回值也是尽量增加新的冗余参数以兼容不同客户端不同的升级频率。等到所有的客户端升级以后再去除冗余的过程。

使用您自己的API

最后，也许最重要的是：吃自己的狗粮。 通过使用它作为从该系统访问信息的唯一方法，使您的API成为一等公民。 API驱动的开发迫使您使API实际运行。 它可以帮助您正确记录您的API，因为您的同事也需要使用它。 此外，它有助于使您的应用程序更加模块化，并逐步帮助您实现微服务架构，它具有自己的一系列优势。

参考文章：

1. Best practices for RESTful API design

2. RESTful API 设计最佳实践