# RESTful 学习笔记

**维基百科：**

**可重新表达的状态迁移**（REST，英文：**Representational State Transfer**）是[Roy Thomas Fielding](https://zh.wikipedia.org/w/index.php?title=Roy_Thomas_Fielding&action=edit&redlink=1)博士于2000年在他的博士论文[[1]](https://zh.wikipedia.org/wiki/%E5%85%B7%E8%B1%A1%E7%8A%B6%E6%80%81%E4%BC%A0%E8%BE%93" \l "cite_note-Fielding-Ch5-1) 中提出来的一种[万维网](https://zh.wikipedia.org/wiki/%E4%B8%87%E7%BB%B4%E7%BD%91" \o "万维网)[软件架构](https://zh.wikipedia.org/wiki/%E8%BD%AF%E4%BB%B6%E6%9E%B6%E6%9E%84)风格，目的是便于不同软件/程序在网络（例如互联网）中互相传递信息。

特征：

1. 客户端-服务器分离（Client - Server）
2. 无状态性（Stateless）
3. 可缓存的（Cacheable）
4. 统一接口（Uniform Interface）
5. 分层的系统（Layered System）
6. 按需编写代码（Code on Demand）

API表达的是你的数据和你的数据使用者之间的契约。

RESTful架构，就是目前最流行的一种互联网软件架构。——阮一峰

满足 REST 架构特性的 Web API 称为 RESTful API。它从以下三个方面对资源进行定义，

* 直观简短的资源地址：URI，比如：http://example.com/resources/。
* 传输的资源：Web服务接受与返回的[互联网媒体类型](https://zh.wikipedia.org/wiki/%E4%BA%92%E8%81%94%E7%BD%91%E5%AA%92%E4%BD%93%E7%B1%BB%E5%9E%8B" \o "互联网媒体类型)，比如：[JSON](https://zh.wikipedia.org/wiki/JSON)，[XML](https://zh.wikipedia.org/wiki/XML" \o "XML)，[YAML](https://zh.wikipedia.org/wiki/YAML" \o "YAML)等。
* 对资源的操作：Web服务在该资源上所支持的一系列[请求方法](https://zh.wikipedia.org/wiki/%E8%B6%85%E6%96%87%E6%9C%AC%E4%BC%A0%E8%BE%93%E5%8D%8F%E8%AE%AE" \l "%E8%AF%B7%E6%B1%82%E6%96%B9%E6%B3%95" \o "超文本传输协议)（比如：POST，GET，PUT或DELETE）。

实现RESTful API时HTTP请求方法的典型用途：

* GET (选择)：从服务器上获取一个具体的资源或者一个资源列表。
* POST （创建）： 在服务器上创建一个新的资源。
* PUT （更新）：以整体的方式更新服务器上的一个资源。
* PATCH （更新）：只更新服务器上一个资源的一个属性。
* DELETE （删除）：删除服务器上的一个资源。

还有两个不常用的HTTP动词：

* HEAD ： 获取一个资源的元数据，如数据的哈希值或最后的更新时间。
* OPTIONS：获取客户端能对资源做什么操作的信息。

不像基于[SOAP](https://zh.wikipedia.org/wiki/SOAP" \o "SOAP)的Web服务，RESTful Web服务并没有“正式”的标准[[4]](https://zh.wikipedia.org/wiki/%E5%85%B7%E8%B1%A1%E7%8A%B6%E6%80%81%E4%BC%A0%E8%BE%93" \l "cite_note-4)。这是因为REST是一种架构，而SOAP只是一个协议。虽然REST不是一个标准，但大部分RESTful Web服务实现会使用[HTTP](https://zh.wikipedia.org/wiki/HTTP" \o "HTTP)、[URI](https://zh.wikipedia.org/wiki/URI" \o "URI)、[JSON](https://zh.wikipedia.org/wiki/JSON" \o "JSON)和[XML](https://zh.wikipedia.org/wiki/XML" \o "XML)等各种标准。(来自维基百科)。

**Protocal**

总是使用 HTTPS

**Version**

一个好的RESTful API会在URL中包含版本信息。

**API Root URL**

API的根地址很重要。让你的API根入口点保持尽可能的简单是很重要的。一个好的RESTful API总是基于HTTPS来发布的。应该部署在专用域名下。

**EndPoints**

一个端点就是指向特定资源或资源集合的URL。

每个网址代表一种资源（resource），所以网址中不能有动词，只能有名词，而且所用的名词往往与数据库的表格名对应。一般应使用复数形式，因为数据库的数据都是集合。

**Filtering**

* ?limit=10: 减少返回给客户端的结果数量（用于分页）
* ?offset=10: 发送一堆信息给客户端（用于分页）
* ?animal\_type\_id=1: 使用条件匹配来过滤记录
* ?sortby=name&order=asc:  对结果按特定属性进行排序

**Status Code**

对于一个RESTful API来说很重要的一点就是要使用HTTP的状态码，因为它们是HTTP的标准。

1xx

范围的状态码是保留给底层HTTP功能使用的，并且估计在你的职业生涯里面也用不着手动发送这样一个状态码出来。

2xx

范围的状态码是保留给成功消息使用的，你尽可能的确保服务器总发送这些状态码给用户。

* 200 OK - [GET]：服务器成功返回用户请求的数据，该操作是幂等的（Idempotent）。
* 201 CREATED - [POST/PUT/PATCH]：用户新建或修改数据成功。
* 202 Accepted - [\*]：表示一个请求已经进入后台排队（异步任务）
* 204 NO CONTENT - [DELETE]：用户删除数据成功。

3xx

范围的状态码是保留给重定向用的。大多数的API不会太常使用这类状态码，但是在新的超媒体样式的API中会使用更多一些。

4xx

范围的状态码是保留给客户端错误用的。例如，客户端提供了一些错误的数据或请求了不存在的内容。这些请求应该是幂等的，不会改变任何服务器的状态。

* 400 INVALID REQUEST - [POST/PUT/PATCH]：用户发出的请求有错误，服务器没有进行新建或修改数据的操作，该操作是幂等的。
* 401 Unauthorized - [\*]：表示用户没有权限（令牌、用户名、密码错误）。
* 403 Forbidden - [\*] 表示用户得到授权（与401错误相对），但是访问是被禁止的。
* 404 NOT FOUND - [\*]：用户发出的请求针对的是不存在的记录，服务器没有进行操作，该操作是幂等的。
* 406 Not Acceptable - [GET]：用户请求的格式不可得（比如用户请求JSON格式，但是只有XML格式）。
* 410 Gone -[GET]：用户请求的资源被永久删除，且不会再得到的。
* 422 Unprocesable entity - [POST/PUT/PATCH] 当创建一个对象时，发生一个验证错误。

5xx

范围的状态码是保留给服务器端错误用的。这些错误常常是从底层的函数抛出来的，并且开发人员也通常没法处理。发送这类状态码的目的是确保客户端能得到一些响应。收到5xx响应后，客户端没办法知道服务器端的状态，所以这类状态码是要尽可能的避免。

* 500 INTERNAL SERVER ERROR - [\*]：服务器发生错误，用户将无法判断发出的请求是否成功。

**Expected Return Documents**

当使用不同的HTTP动词向服务器请求时，客户端需要在返回结果里面拿到一系列的信息。下面的列表是非常经典的RESTful API定义：

* GET /collection: 返回一系列资源对象
* GET /collection/resource: 返回单独的资源对象
* POST /collection: 返回新创建的资源对象
* PUT /collection/resource: 返回完整的资源对象
* PATCH /collection/resource: 返回完整的资源对象
* DELETE /collection/resource: 返回一个空文档

请注意当一个客户端创建一个资源时，她们常常不知道新建资源的ID（也许还有其他的属性，如创建和修改的时间戳等）。这些属性将在随后的请求中返回，并且作为刚才POST请求的一个响应结果。

**Content Type**

把使用方式放在响应数据的接收头里面。通过接收头来指示返回内容类型（这也是HTTP标准的一部分）

**Hypermedia APIs**

超媒体API很可能就是RESTful API设计的将来。在非超媒体RESTful API的情景中，URL端点是服务器与客户端契约的一部分。这些端点必须让客户端事先知道，并且修改它们也意味着客户端可能再也无法与服务器通信了。

超媒体API概念的运作跟人们的行为类似。通过请求API的根来获得一个URL的列表，这个列表里面的每一个URL都指向一个集合，并且提供了客户端可以理解的信息来描述每一个集合。是否为每一个资源提供ID并不重要（或者不是必须的），只要提供URL即可。

**Documentation**

* 让你的文档对那些未经认证的开发者也可用
* 不要使用文档自动化生成器，即便你用了，你也要保证自己审阅过并让它具有更好的版式。
* 不要截断示例中请求与响应的内容，要展示完整的东西。并在文档中使用高亮语法。
* 文档化每一个端点所预期的响应代码和可能的错误消息，和在什么情况下会产生这些的错误消息

**Others**

（1）API的身份认证应该使用[OAuth 2.0](http://www.ruanyifeng.com/blog/2014/05/oauth_2_0.html" \t "_blank)框架。

（2）服务器返回的数据格式，应该尽量使用JSON，避免使用XML。

最后

究竟什么是 REST ？

用最简单的话说，在 REST 架构风格中，数据和功能都被视为是资源，并通过统一资源定位符（URI）被访问。这些资源通过一系列简单的、有良好的定义的操作被使用。客户端和服务器通过使用标准化的接口和协议来交换这些资源表述，典型的协议如 HTTP。

将资源与其表述解耦，使其内容可以以多种不同的形式被访问，如 HTML，XML, 普通文本，PDF， JPEG， JSON，及其他。关于资源的元数据也是可利用的，例如，用来控制缓存，侦探传输错误，协商合适的表述性格式，以及参与认证及访问控制。最重要的是，与资源的每一次交互都是无状态的。

**幂等性**

从定义上看，HTTP方法的幂等性是指一次和多次请求某一个资源应该具有同样的副作用。

幂等性并不属于特定的协议，它是分布式系统的一种特性；所以，不论是SOA还是RESTful的Web API设计都应该考虑幂等性。

下面将介绍HTTP GET、DELETE、PUT、POST四种主要方法的语义和幂等性。

* GET方法用于获取资源，不应有副作用，所以是幂等的。
* DELETE方法用于删除资源，有副作用，但它应该满足幂等性。调用一次和N次对系统产生的副作用是相同的。
* POST所对应的URI并非创建的资源本身，而是资源的接收者。两次相同的POST请求会在服务器端创建两份资源，它们具有不同的URI；所以，POST方法不具备幂等性。
* PUT所对应的URI是要创建或更新的资源本身。对同一URI进行多次PUT的副作用和一次PUT是相同的；因此，PUT方法具有幂等性。

**参考文献**

<https://github.com/aisuhua/restful-api-design-references>

<http://www.cnblogs.com/moonz-wu/p/4211626.html>

<https://restfulapi.net/>

<http://www.ruanyifeng.com/blog/2014/05/restful_api.html>