RESTful及RESTful API设计规范简介

目录

[什么是REST 1](#_Toc15571)

[什么是RESTful 2](#_Toc27938)

[RESTful API设计规范 3](#_Toc13615)

[URL命名原则 5](#_Toc6858)

[其他补充说明 7](#_Toc5799)

[举例说明 8](#_Toc11062)

# 什么是REST

REST这个词，是Roy Thomas Fielding在他2000年的博士论文中提出的。

REST的全称是Representational State Transfer（表现层状态转化）。

”资源”是一种信息实体，它可以有多种外在表现形式。我们把”资源”具体呈现出来的形式，叫做它的”表现层”（Representation）。

比如，文本可以用txt格式表现，也可以用HTML格式、XML格式、JSON格式表现，甚至可以采用二进制格式；图片可以用JPG格式表现，也可以用PNG格式表现。

访问一个网站，就代表了客户端和服务器的一个互动过程。在这个过程中，势必涉及到数据和状态的变化。

互联网通信协议HTTP协议，是一个无状态协议。这意味着，所有的状态都保存在服务器端。因此，如果客户端想要操作服务器，必须通过某种手段，让服务器端发生”状态转化”（State Transfer）。而这种转化是建立在表现层之上的，所以就是”表现层状态转化”。—— [ 阮一峰的网络日志：理解RESTful架构 ]

# 什么是RESTful

RESTful是一种软件架构风格、设计风格，而不是标准，只是提供了一组设计原则和约束条件。它主要用于客户端和服务器交互类的软件。基于这个风格设计的软件可以更简洁，更有层次，更易于实现缓存等机制。

—— [ 百度百科 ]

优点：

   RESTful结构清晰、符合标准、易于理解、扩展方便，所以正得到越来越多网站的采用。相比于SOAP（Simple Object Access protocol，简单对象访问协议）以及XML-RPC更加简单明了，无论是对URL的处理还是对Payload的编码，REST都倾向于用更加简单轻量的方法设计和实现。

# RESTful API设计规范

详细的可以参考下面这几篇文章：

Restful API 的设计规范

REST接口设计规范

RESTful API 设计指南

跟着 Github 学习 Restful HTTP API 设计 (Github的API设计是RESTful的一个教科书式的典范)好RESTful API的设计原则

这里简单总结一下，有些没有说到的可以参考上面的文章或其他一些资料。

1. URI是对大小写敏感的，所以为了避免歧义，我们尽量用小写字符URL的设计应该以资源为中心
2. 尽量使用复数来表示资源（例外：文档(Document)类型的资源用名词(短语)单数命名；控制器(Controller)类型的资源用动词(短语)命名，可参看上面第二个链接）
3. URI中应该放置版本信息，如： /v1/users/1(也可以放在Accept Header中，如：Accept: application/json+v1；或者自定义 Header如：X-Api-Version: 1)
4. URL中应该只有名词，不能使用动词（对资源的不同操作用http的请求方法来表示）应该使用连字符”-“来提高URL的可读性，而不是使用下划线”\_”
5. “/”分隔符一般用来对资源层级的划分，为了避免混淆，”/”不应该出现在URL的末尾
6. CRUD的操作不要体现在URI中，HTTP协议中的操作符（Verbs）已经对CRUD做了映射:
7. GET：获取资源
8. POST：添加或创建资源
9. PUT：更新（或替换）资源的所有属性
10. PATCH：更新资源的部分属性
11. DELETE：删除资源
12. HEAD：只获取某个资源的头部信息。比如只想了解某个文件的大小，某个资源的修改日期等
13. OPTIONS：获取信息，关于资源的哪些属性是客户端可以改变的。
14. 返回结果：
15. GET /collection：返回资源对象的列表（数组）
16. GET /collection/resource：返回单个资源对象
17. POST /collection：返回新生成的资源对象
18. PUT /collection/resource：返回完整的资源对象
19. PATCH /collection/resource：返回完整的资源对象
20. DELETE /collection/resource：返回一个空文档
21. 返回状态码推荐标准HTTP状态码:
22. 200 (“OK”) 用于一般性的成功返回
23. 200 (“OK”) 不可用于请求错误返回
24. 201 (“Created”) 资源被创建
25. 202 (“Accepted”) 用于Controller控制类资源异步处理的返回，仅表示请求已经收到。对于耗时比较久的处理，一般用异步处理来完成
26. 204 (“No Content”) 此状态可能会出现在PUT、POST、DELETE的请求中，一般表示资源存在，但消息体中不会返回任何资源相关的状态或信息。
27. 301 (“Moved Permanently”) 资源的URI被转移，需要使用新的URI访问
28. 302 (“Found”) 不推荐使用，此代码在HTTP1.1协议中被303/307替代。我们目前对302的使用和最初HTTP1.0定义的语意是有出入的，应该只有在GET/HEAD方法下，客户端才能根据Location执行自动跳转，而我们目前的客户端基本上是不会判断原请求方法的，无条件的执行临时重定向
29. 303 (“See Other”) 返回一个资源地址URI的引用，但不强制要求客户端获取该地址的状态(访问该地址)
30. 304 (“Not Modified”) 有一些类似于204状态，服务器端的资源与客户端最近访问的资源版本一致，并无修改，不返回资源消息体。可以用来降低服务端的压力
31. 307 (“Temporary Redirect”) 目前URI不能提供当前请求的服务，临时性重定向到另外一个URI。在HTTP1.1中307是用来替代早期HTTP1.0中使用不当的302
32. 400 (“Bad Request”) 用于客户端一般性错误返回, 在其它4xx错误以外的错误，也可以使用400，具体错误信息可以放在body中
33. 401 (“Unauthorized”) 在访问一个需要验证的资源时，验证错误
34. 403 (“Forbidden”) 一般用于非验证性资源访问被禁止，例如对于某些客户端只开放部分API的访问权限，而另外一些API可能无法访问时，可以给予403状态
35. 404 (“Not Found”) 找不到URI对应的资源
36. 405 (“Method Not Allowed”) HTTP的方法不支持，例如某些只读资源，可能不支持POST/DELETE。但405的响应header中必须声明该URI所支持的方法
37. 406 (“Not Acceptable”) 客户端所请求的资源数据格式类型不被支持，例如客户端请求数据格式为application/xml，但服务器端只支持application/json
38. 409 (“Conflict”) 资源状态冲突，例如客户端尝试删除一个非空的Store资源
39. 412 (“Precondition Failed”) 用于有条件的操作不被满足时
40. 415 (“Unsupported Media Type”) 客户所支持的数据类型，服务端无法满足
41. 500 (“Internal Server Error”) 服务器端的接口错误，此错误于客户端无关
42. 503 Service Unavailable 服务器因为负载过高或者维护，暂时无法提供服务。服务器端应该返回 Retry-After 头部，告诉客户端过一段时间再来重试
43. 返回数据的类型在http响应报文的header中用Content-Type指定
44. 避免层级过深的URI

/在URI中表示层级，用于按实体关联关系进行对象导航，一般跟进id导航;

过深的导航容易导致url膨胀，不易维护，如 GET /categorys/1/areas/3/foods/4，尽量使用查询参数代替路径中的实体导航，如GET /foods?category=1&area=3;

# URL命名原则

1. URL请求采用小写字母，数字，部分特殊符号（非制表符）组成。
2. URL请求中不采用大小写混合的驼峰命名方式，尽量采用全小写单词，如果需要连接多个单词，则采用连接符“\_”连接单词

# 其他补充说明

URI(Uniform Resource Identifiers) 统一资源标示符

URL(Uniform Resource Locator) 统一资源定位符

URI的格式定义如下：

URI = scheme “://” authority “/” path [ “?” query ] [ “#” fragment ]

URL是URI的一个子集(一种具体实现)，对于REST API来说一个资源一般对应一个唯一的URI(URL)。在URI的设计中，我们会遵循一些规则，使接口看起透明易读，方便使用者调用。

端点(Endpoint)：这个API在服务器上的URL用于表达一个资源或者一个集合,如：https://api.example.com/v1/categorys、https://api.example.com/v1/foods

安全性与幂等性

安全性：不会改变资源状态，可以理解为只读的；

幂等性（Idempotent）：执行1次和执行N次，对资源状态改变的效果是等价的，无边际效应，多次操作得到相同的结果。

  安全性和幂等性均不保证反复请求能拿到相同的response。以 DELETE 为例，第一次DELETE返回200表示删除成功，第二次返回404提示资源不存在，这是允许的。

# 举例说明

假设数据库中有两张表food\_category和food，表结构如下

food\_category（菜品分类）表

| **字段名** | **类型** |
| --- | --- |
| id | int |
| name | varchar |

food（菜品）表

| **字段名** | **类型** |
| --- | --- |
| id | int |
| name | varchar |
| price | float |
| description | varchar |
| category\_id | int |

food表中的category\_id是一个外键，用来和food\_category表关联，则相关的CRUD 接口路由用RESTful风格来设计可以如下这样（前面是请求方式，后面是路由）：

1. 获取所有菜品分类信息(url中包含版本号，资源类型为复数，获取数据请求方式用GET)

GET /v1/categorys

1. 分页获取菜品分类信息（url中带参数，实际url可能是这样子/v1/categorys?page=1&size=10）

GET /v1/categorys 参数列表中接收page,size

1. 获取指定菜品分类的信息(其中{cid}是一个url参数，指定获得单个资源，实际请求时url可能是这样子的 /v1/categorys/1)

GET /v1/categorys/{cid}

1. 添加菜品分类信息(创建用POST)

POST /v1/categorys 参数列表中可以用一个对象接收参数{categoryName:”“,address:”“}

1. 修改某个菜品分类信息（全量更新，用PUT）

PUT /v1/categorys/{cid} 参数列表中可以用一个对象接收参数{categoryName:”“,address:”“}

1. 修改某个菜品分类所在的地址（只更新部分内容,用PATCH）

PATCH /v1/categorys/{cid} 参数列表接收参数address

1. 删除某个菜品分类信息（删除用DELETE）

DELETE /v1/categorys/{cid}

1. 删除所有菜品分类信息

DELETE /v1/categorys

1. 获取某个菜品分类下的所有菜品信息（表现层级关系）

GET /v1/categorys/{cid}/foods

1. 获取某个菜品分类下的某只菜品信息（表现层级关系）

GET /v1/categorys/{cid}/foods/{fid}