RESTful API规范

REST即表述性状态传递（英文：Representational State Transfer，简称REST）是Roy Fielding博士在2000年他的博士论文中提出来的一种软件架构风格。它是一种针对网络应用的设计和开发方式，可以降低开发的复杂性，提高系统的可伸缩性。

RESTful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。

做出一个好的API设计很难，你的API越容易使用，那么就会有越多的人去用它。规划好API的外观要先于开发它实际的功能。

**https://www.fenjin.com:8080/api/v1/fjtmsusers/users/1**

**https://www.fenjin.com:8080/api/v1/fjtmsusers/users?username=admin&fields=id,username,password&sort=-createdtime,username&page=1,15**

|  |  |  |
| --- | --- | --- |
| GET | 请求方法类型 | GET，POST，PUT，DELETE |
| https | 协议类型 | http，https（建议） |
| www.fenjin.com | 域名 | 可以是IP地址或主机名 |
| 8080 | API访问端口 | 可以没有，默认为80 |
| api | API root | 可以使用子域名，解决跨域风险 |
| v1 | 版本 | 确保API的兼容性，避免跨域引发发送多次请求 |
| fenjin-users | 微服务 | Getway映射微服务名称 |
| users | 路由 | 控制器路由名称 |
| 1 | 查询条件 | /{id}或者?username=admin |
| fields | 过滤条件 | 输出内容，英文逗号“,”分隔 |
| sort | 排序条件 | 英文逗号“,”分隔，“-”降序，默认升序 |
| page | 分页信息 | 指定第几页及每页记录数，用英文逗号“,”分隔 |

在Restful之前的操作：

http://127.0.0.1/user/query/1 GET  根据用户id查询用户数据

http://127.0.0.1/user/save POST 新增用户

http://127.0.0.1/user/update POST 修改用户信息

http://127.0.0.1/user/delete GET/POST 删除用户信息

RESTful用法：

http://127.0.0.1/user/1 GET  根据用户id查询用户数据

http://127.0.0.1/user  POST 新增用户

http://127.0.0.1/user  PUT 修改用户信息

http://127.0.0.1/user  DELETE 删除用户信息



# URL设计

一个好的RESTful API只允许第三方调用者使用这四个动词进行数据交互，并且在URL段里面不出现任何其他的动词。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 资源 | GET查询 | POST创建 | PUT修改 | DELETE删除 |
| /users | 返回users集合 | 创建新的user | 批量更新users | 删除所有users |
| /users/1 | 返回指定user | 不允许 | 更新指定user | 删除指定user |
| /users?query | 返回指定条件的users集合 | 不允许 | 不允许 | 删除指定条件的users |

## 1.1动词+宾语

RESTful的核心思想就是，客户端发出的数据操作指令都是“动词+宾语”的结构。比如，GET /users这个命令，GET是动词，/users是宾语。

动词通常就是五种，对应CRUD操作。

|  |  |  |
| --- | --- | --- |
| 动词 | 对应操作 | 说明 |
| GET | Read | 查询，从服务器上获取一个具体的资源或者一个资源列表 |
| POST | Create | 创建，在服务器上创建一个新的资源 |
| PUT | Update | 更新，以整体的方式更新服务器上的一个资源 |
| DELETE | Delete | 删除，删除服务器上的一个资源或者一个资源集合 |

根据HTTP规范，动词一律大写。

## 1.2宾语必须是名词，且建议使用小写的名词复数

宾语就是API的URL，是HTTP动词作用的对象。应该是名字，不能是动词。

|  |  |
| --- | --- |
| 推荐 | 错误 |
| /users | /getAllUsers |
| /users/1 | /getById/1 |
| /users/username/admin | /getByUsername/admin |
| /users?page=1,15 | /getAllUsers/1/15 |

## 宾语名词尽量领域模型和数据库表名一致

名词命名与领域模型或数据库表名一致，方便对接口、服务、数据访问、数据库的统一，有利于后续程序跟踪调试，也方便API调用者与后台开发者之间沟通。

|  |  |  |  |
| --- | --- | --- | --- |
| 推荐 | 避免 | 领域模型或数据库表名 | 名词解释（多种解释） |
| /users | /account | User或user | 用户或账户 |
| /logs | /systemlog | Log或log | 系统日志或异常日志 |
| /activitylogs | /operationlog | Activitylog  或activitylog | 用户活动日志或操作日志 |

## 1.4避免多级URL，可选的、复杂的参数使用查询字符串（？）

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。尽量使用查询参数代替路径中的实体导航。

|  |  |
| --- | --- |
| 推荐 | 避免 |
| /animals?zoo=1&area=3 | /zoos/1/areas/3/animals/4 |
| /roles?deleted=false | /roles/deleted |

## 1.5避免在URL的最后出现“/”

|  |  |
| --- | --- |
| 推荐 | 避免 |
| /users | /Users/ |
| /users/1 | /users/1/ |

## 1.6提供分页、排序及过滤的功能

当客户端创建了一个请求来获取一个列表时，返回给他们的一个符合查询条件的所有对象的列表。这个列表可能会很大。但你不能随意给返回数据的数量做限制。可以让客户端自己对结果做一些具体的过滤或限制。这么做最重要的一个圆形是可以最小化网络传输，让客户端尽可能快的得到查询结果。如果客户端能对结果做一些过滤或分页就更好了。

|  |  |
| --- | --- |
| /users?page=1,10&count=true | 使用page限定返回数据量，count是否返回总数 |
| /users?username=admin | 使用条件匹配来过滤记录 |
| /users?fields=id,username,password | 使用fields限定返回数据内容 |
| /users?sort=username,-createdtime | 使用sort限定返回数据的排序 |

## 1.7避免使用下划线“\_”，可以使用连接符“-”代替

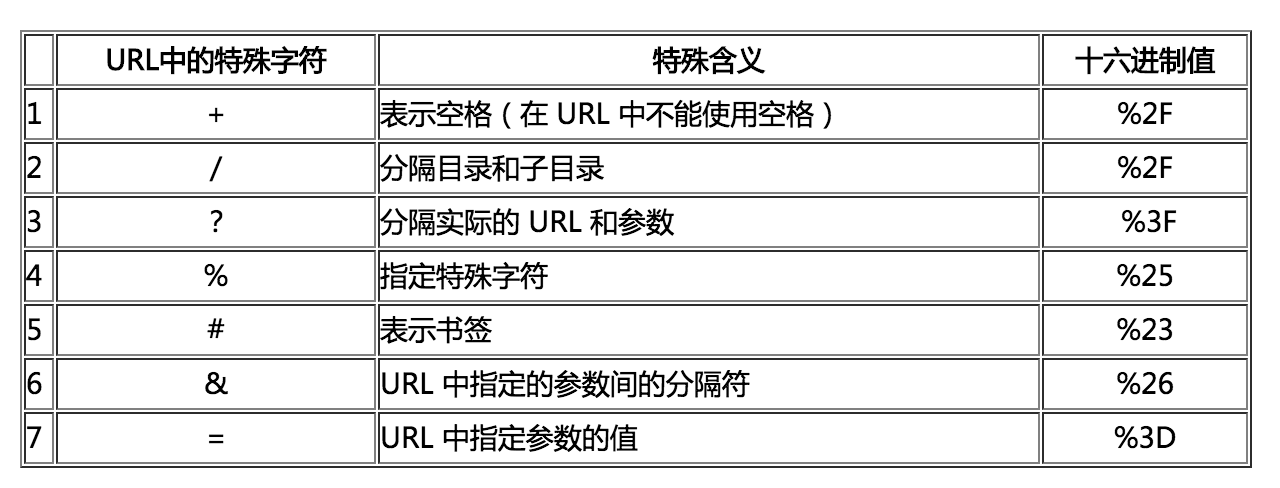
RESTful API应该具备良好的可读性，可以将多个单词用“-”分割，不采用下划线“\_”主要是因为与浏览器超链接显示效果重叠，影响可读性。

|  |  |
| --- | --- |
| 推荐 | 避免 |
| /fjtmsusers/users | /fenjin\_fjtms\_users/users |
| /fjtmsusers/users | /fenjin-fjtms-users/users |

路由禁止使用“-”，参数部分可以，但尽量避免使用“-”

## ~~1.8参数列表要encode(可选)~~

1. 不要对Unreserved Characters做percent encode编码。
2. 除了保留字符和非保留字符外的所有字符，必须使用percent encode进行编码。
3. 保留字符不用于URI分隔符，而是用于其它位置，比如query部分的value时，要对这时用到的保留字符做percent encode编码。



## 1.9路由重复的情况下将参数较少的方法改为路由方式

根据RESTful规范，查询列表和较少条件查询的路由发生冲突，推荐使用一下的方式进行区分。

|  |  |
| --- | --- |
| 作用 | URL推荐 |
| 多条件过滤查询用户列表 | /users?username=admin&deleted=false |
| 通过用户id查询用户信息 | /users/{id} |
| 通过用户名username查询用户信息 | /users/username/{username} |
| 通过电子邮箱email查询用户信息 | /users/email/{email} |
| 修改用户信息 | /users/{id} |
| 修改用户账号密码 | /users/{id}/password |

# 状态码

## 2.1状态码必须精确

客户端每一次请求，服务器都必须给出回应。回应包括HTTP状态码，消息内容和具体数据三部分。HTTP状态码就是一个三位数，分成五个类别。

|  |  |
| --- | --- |
| 1xx | 相关信息 |
| 2xx | 操作成功 |
| 3xx | 重定向 |
| 4xx | 客户端错误 |
| 5xx | 服务器错误 |

这五大类总共包含[100多种](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes)状态码，覆盖了绝大部分可能遇到的情况。每一种状态码都有标准的（或者约定的）解释，客户端只需查看状态码就可以判断出发生了什么情况，所以服务器应该返回尽可能精确的状态码。

|  |  |  |
| --- | --- | --- |
| HTTP Method | 详细解释 | 返回状态码 |
| GET | 获取对象或集合 | 200成功、401没有授权、403访问禁止、404没有资源、参数错误、406请求格式不正确、500服务器内部错误 |
| POST | 新增一条数据 | 200创建成功、422新增数据验证错误、（401、403、404、406、500） |
| DELETE | 删除一条数据 | 200删除成功、（401、403、404、406、500） |
| PUT | 更新实体所有信息除ID外 | 200更新成功、422（401、403、404、406、500） |

# 服务器回应

合法的请求应该统一返回数据格式，包括：

|  |  |
| --- | --- |
| Code | 整数类型的http响应状态码 |
| Message | 显示成功或失败信息 |
| Data | 返回成功响应的Json |

例如

|  |  |
| --- | --- |
| 返回成功的响应Json | 返回失败的响应Json |
| {  "code": 200,  "message": "success",  "data": {  "userName": "123456",  "age": 16,  "address": "beijing"  }  } | {  "code": 401,  "message": "error message",  "data": null  } |

|  |
| --- |
| */\*\*  \* 服务器返回  \*/* @Data public class Result {  *//操作成功* public static final int *SUCCESS* = 200;  *//未认证* public static final int *UNAUTHORIZED* = 401;  *//未授权* public static final int *FORBIDDEN* = 403;  *//请求资源找不到* public static final int *NOT\_FOUND* = 404;  *//请求格式不正确* public static final int *NOT\_ACCEPTABLE* = 406;  *//参数校验失败* public static final int *VALIDATE\_FAILED* = 422;  *//操作失败* public static final int *FAILED* = 500;    private int code;  private String message;  private Object data;   */\*\*  \* 普通成功返回  \*  \* @param data 获取的数据  \*/* public Result success(Object data) {  this.code = *SUCCESS*;  this.message = "操作成功";  this.data = data;  return this;  }   */\*\*  \* 成功返回List数据  \*  \* @param data 获取的数据  \*/* public Result success(List data) {  Map<String, Object> result = new HashMap<>();  result.put("list", data);  this.code = *SUCCESS*;  this.message = "操作成功";  this.data = data;  return this;  }   */\*\*  \* 返回分页成功数据  \*/* public Result pageSuccess(int pageIndex, int pageSize, int total, List data) { *// PageInfo pageInfo = new PageInfo(data);* Map<String, Object> result = new HashMap<>();  result.put("pageSize", pageSize); *// result.put("totalPage", pageInfo.getPages());* result.put("total", total);  result.put("pageIndex", pageIndex);  result.put("list", data);  this.code = *SUCCESS*;  this.message = "操作成功";  this.data = result;  return this;  }   */\*\*  \* 普通失败提示信息  \*/* public Result failed() {  this.code = *FAILED*;  this.message = "操作失败";  return this;  }   */\*\*  \* 请求资源找不到  \*/* public Result notFound() {  this.code = *NOT\_FOUND*;  this.message = "请求资源不存在";  return this;  }   */\*\*  \* 请求格式不正确  \*/* public Result notAcceptable() {  this.code = *NOT\_ACCEPTABLE*;  this.message = "请求格式不正确";  return this;  }   */\*\*  \* 自定义失败提示信息  \*/* public Result failed(String message) {  this.code = *FAILED*;  this.message = message;  return this;  }   */\*\*  \* 参数验证失败使用  \* @param message 错误信息  \*/* public Result validateFailed(String message) {  this.code = *VALIDATE\_FAILED*;  this.message = message;  return this;  }   */\*\*  \* 未登录时使用  \* @param message 错误信息  \*/* public Result unauthorized(String message) {  this.code = *UNAUTHORIZED*;  this.message = "暂未登录或token已经过期";  this.data = message;  return this;  }   */\*\*  \* 未授权时使用  \* @param message 错误信息  \*/* public Result forbidden(String message) {  this.code = *FORBIDDEN*;  this.message = "没有相关权限";  this.data = message;  return this;  }   */\*\*  \* 参数验证失败使用  \* @param result 错误信息  \*/* public Result validateFailed(BindingResult result) {  validateFailed(result.getFieldError().getDefaultMessage());  return this;  } } |

## 4.1只返回Json格式的数据

API返回的数据格式，不应该是纯文本，而应该是一个JSON对象，因为这样才能返回标准的结构化数据。所以，服务器回应的HTTP头的Content-Type属性要设置为application/json。

## 4.2总是使用SSL（技术研究）

API与用户的通信协议，总是使用HTTPS协议，简化认证过程，访问令牌不用让每个API请求签署。不加密的通信，便于窃听或伪造，如身份验证凭证被劫持。

## 4.3返回结果应符合以下原则

GET /collection: 返回资源对象的列表 (数组)

GET /collection/resource: 返回单个资源对象

POST /collection: 返回新生成的资源对象

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