

RESTful API 设计规范，看这篇就够了！

Java后端 前天

点击上方 Java后端，选择 设为星标
优质文章，及时送达



RESTful 是目前最流行的 API 设计规范，用于 Web 数据接口的设计。

它的大原则容易把握，但是细节不容易做对。本文总结 RESTful 的设计细节，介绍如何设计出易于理解和使用的 API。



一 URL 设计

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 /articles 这个命令，GET 是动词，/articles 是宾语，动词通常就有 5 种 HTTP 请求方法，对应 CRUD 操作，根据 HTTP 规范，动词一律大写。

```
# GET: 读取 (Read)
# POST: 新建 (Create)
# PUT: 更新 (Update)
# PATCH: 更新 (Update)，通常是部分更新
# DELETE: 删除 (Delete)
```

动词的覆盖

有些客户端只能使用 GET 和 POST 这两种方法。服务器必须接受 POST 模拟其他三个方法（PUT、PATCH、DELETE）。这时，客户端发出的 HTTP 请求，要加上 X-HTTP-Method-Override 属性，告诉服务器应该使用哪一个动词，覆盖 POST 方法。

```
POST /api/Person/4 HTTP/1.1
X-HTTP-Method-Override: PUT
```

上面代码中，X-HTTP-Method-Override 指定本次请求的方法是 PUT，而不是 POST。

宾语必须是名词

宾语就是 API 的 URL，是 HTTP 动词作用的对象。它应该是名词，不能是动词。比如，/articles 这个 URL 就是正确的，而下面的 URL 不是名词，所以都是错误的。

```
# /getAllCars
# /createNewCar
# /deleteAllRedCars
```

复数 URL

既然 URL 是名词，那么应该使用复数，还是单数？这没有统一的规定，但是常见的操作是读取一个集合，比如 GET /articles（读取所有文章），这里明显应该是复数。

为了统一起见，建议都使用复数 URL，比如 GET /articles/2 要好于 GET /article/2。

避免多级 URL

常见的情况是，资源需要多级分类，因此很容易写出多级的 URL，比如获取某个作者的某一类文章。

```
# GET /authors/12/categories/2
```

这种 URL 不利于扩展，语义也不明确，往往要想一会，才能明白含义。

更好的做法是，除了第一级，其他级别都用查询字符串表达。

```
# GET /authors/12?categories=2
```

下面是另一个例子，查询已发布的文章。你可能会设计成下面的 URL。

```
# GET /articles/published
```

查询字符串的写法明显更好

```
# GET /articles?published=true
```



二、状态码

状态码必须精确

客户端的每一次请求，服务器都必须给出回应。回应包括 HTTP 状态码和数据两部分。

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。

API 不需要1xx状态码，下面介绍其他四类状态码的精确含义。

2XX状态码

200状态码表示操作成功，但是不同的方法可以返回更精确的状态码。

```
# GET: 200 OK
# POST: 201 Created
# PUT: 200 OK
# PATCH: 200 OK
# DELETE: 204 No Content
```

上面代码中，POST返回201状态码，表示生成了新的资源；DELETE返回204状态码，表示资源已经不存在。

此外，202 Accepted状态码表示服务器已经收到请求，但还未进行处理，会在未来再处理，通常用于异步操作。下面是一个例子。

```
HTTP/1.1 202 Accepted

{
  "task": {
    "href": "/api/company/job-management/jobs/2130040",
    "id": "2130040"
  }
}
```

3xx 状态码

API 用不到301状态码（永久重定向）和302状态码（暂时重定向，307也是这个含义），因为它们可以由应用级别返回，浏览器会直接跳转，API 级别可以不考虑这两种情况。

API 用到的3xx状态码，主要是303 See Other，表示参考另一个 URL。它与302和307的含义一样，也是“暂时重定向”，区别在于302和307用于GET请求，而303用于POST、PUT和DELETE请求。收到303以后，浏览器不会自动跳转，而会让用户自己决定下一步怎么办。

下面是一个例子。

```
HTTP/1.1 303 See Other
Location: /api/orders/12345
```

4xx 状态码

4xx状态码表示客户端错误，主要有下面几种。

- **400 Bad Request:** 服务器不理解客户端的请求，未做任何处理。
- **401 Unauthorized:** 用户未提供身份验证凭据，或者没有通过身份验证。
- **403 Forbidden:** 用户通过了身份验证，但是不具有访问资源所需的权限。
- **404 Not Found:** 所请求的资源不存在，或不可用。
- **405 Method Not Allowed:** 用户已经通过身份验证，但是所用的 HTTP 方法不在他的权限之内。
- **410 Gone:** 所请求的资源已从这个地址转移，不再可用。
- **415 Unsupported Media Type:** 客户端要求的返回格式不支持。比如，API 只能返回JSON 格式，但是客户端要求返回XML 格式。
- **422 Unprocessable Entity:** 客户端上传的附件无法处理，导致请求失败。
- **429 Too Many Requests:** 客户端的请求次数超过限额。

5xx 状态码

5xx状态码表示服务端错误。一般来说，API 不会向用户透露服务器的详细信息，所以只要两个状态码就够了。

- **500 Internal Server Error:** 客户端请求有效，服务器处理时发生了意外。
- **503 Service Unavailable:** 服务器无法处理请求，一般用于网站维护状态。



三、服务器回应

不要返回纯文本

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属性要设为 application/json。

客户端请求时，也要明确告诉服务器，可以接受 JSON 格式，即请求的 HTTP 头的ACCEPT 属性也要设成application/json。下面是一个例子。

```
GET /orders/2 HTTP/1.1
Accept: application/json
```

发生错误时，不要返回 200 状态码

有一种不恰当的做法是，即使发生错误，也返回200状态码，把错误信息放在数据体里面，就像下面这样。

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "status": "failure",
  "data": {
    "error": "Expected at least two items in list."
  }
}
```

上面代码中，解析数据体以后，才能得知操作失败。

这种做法实际上取消了状态码，这是完全不可取的。正确的做法是，状态码反映发生的错误，具体的错误信息放在数据体里面返回。下面是一个例子。

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
  "error": "Invalid payload.",
  "detail": {
    "surname": "This field is required."
  }
}
```

提供链接

API 的使用者未必知道，URL 是怎么设计的。一个解决方法就是，在回应中，给出相关链接，便于下一步操作。这样的话，用户只要记住一个 URL，就可以发现其他的 URL。这种方法叫做 HATEOAS。

举例来说，GitHub 的 API 都在 api.github.com 这个域名。访问它，就可以得到其他 URL。

```
{
  ...
  "feeds_url": "https://api.github.com/feeds",
  "followers_url": "https://api.github.com/user/followers",
  "following_url": "https://api.github.com/user/following/{target}",
  "gists_url": "https://api.github.com/gists/{gist_id}",
  "hub_url": "https://api.github.com/hub",
  ...
}
```

上面的回应中，挑一个 URL 访问，又可以得到别的 URL。对于用户来说，不需要记住 URL 设计，只要从 api.github.com 一步步查找就可以了。

HATEOAS 的格式没有统一规定，上面例子中，GitHub 将它们与其他属性放在一起。更好的做法应该是，将相关链接与其他属性分开。

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "status": "In progress",
  "links": [{
    { "rel": "cancel", "method": "delete", "href": "/api/status/12345" },
    { "rel": "edit", "method": "put", "href": "/api/status/12345" }
  ]
}
```

-END-

如果看到这里，说明你喜欢这篇文章，请 [转发](#)、[点赞](#)。同时 [标星（置顶）](#) 本公众号可以第一时间间接受到博文推送。

推荐阅读

[f. 今年工作不好找，一面试 Redis 直接给问懵逼了](#)

2. 五分钟带你读懂 TCP 协议 (图文并茂)

3. Git 从入门到精通

4. 阿里官方Redis开发规范!



微信搜一搜

Java后端

最近整理一份面试资料《[Java技术栈学习手册](#)》，覆盖了Java技术、面试题精选、Spring全家桶、Nginx、SSM、微服务、数据库、数据结构、架构等等。

获取方式：点 “[在看](#)”，关注公众号 [Java后端](#) 并回复 [777](#) 领取，更多内容陆续奉上。

喜欢文章，点个在看