# RESTful API 规范 v1.0

[HTTP网络](http://imweb.io/tab/network)

本文作者：IMWeb 梁伟盛 原文出处：[IMWeb社区](http://imweb.io/topic/5707561f06f2400432c139a5) **未经同意，禁止转载**

# RESTful API 规范 v1.0

[toc]

## URI

### URI规范

* 不要用大写
* 单词间使用下划线'\_'
* 不使用动词，资源要使用名词复数形式，如：user、rooms、tickets
* 层级 >= 三层，则使用'?'带参数

 (bad) /citys?users=1&address=2; (good)

## Request

### Method

* GET：查询资源
* POST：创建资源
* PUT/PATCH
  + PUT：全量更新资源（提供改变后的完整资源）
  + PATCH：局部更新资源（仅提供改变的属性）
* DELETE：删除资源

### 安全性与幂等性

* 安全性：任意多次对同一资源操作，都不会导致资源的状态变化
* 幂等性：任意次对同一资源操作，对资源的改变是一样的 |Method|安全性|幂等性| |------|:---:|:---:| |GET|√|√| |POST|×|×| |PUT|×|√| |PATCH|×|√| |DELETE|×|√|

### 兼容

很多客户只支持GET/POST请求，一般有两种方式模拟PUT等请求

* 添加\_method参数
* /users/1?\_method=put&name=111
* 添加X-HTTP-Method-Override请求头 (我们使用这种方式)
* X-HTTP-Method-Override: PUT

### 参数

#### Method

##### GET

* 非id的参数使用'?'方式传输
* /users/1?state=closed

##### POST、PATCH、PUT、DELETE

* 非id的参数使用body传输，并且应该encode

#### 过滤

?type=1&state=closed

#### 排序

* +升序，如?sort=+create\_time，根据id升序
* -降序，如?sort=-create\_time，根据id降序

#### 分页

?limit=10&offset=10

* limit：返回记录数量
* offset：返回记录的开始位置

#### 单参数多字段

使用, 分隔，如

/users/1?fields=name,age,city

## 版本控制

三种方案：

1. 在uri中加入版本： /v1/room/1
2. Accept Header：Accept: v1
3. 自定义 Header：X-Imweb-Media-Type: imweb.v1 (我们使用此方案)

自定义Media-Type参考资料[github](https://developer.github.com/v3/media/#request-specific-version)

## 状态码

### 成功

| **Code** | **Method** | **Describe** |
| --- | --- | --- |
| 200 | ALL | 请求成功并返回实体资源 |
| 201 | POST | 创建资源成功 |

### 客户端错误

| **Code** | **Method** | **Describe** |
| --- | --- | --- |
| 400 | ALL | 一般是参数错误 |
| 401 | ALL | 一般用户验证失败（用户名、密码错误等） |
| 403 | ALL | 一般用户权限校验失败 |
| 404 | ALL | 资源不存在（github在权限校验失败的情况下也会返回404，为了防止一些私有接口泄露出去） |
| 422 | ALL | 一般是必要字段缺失或参数格式化问题 |

### 服务器错误

| **CODE** | **METHOD** | **DESCRIBE** |
| --- | --- | --- |
| 500 | ALL | 服务器未知错误 |

以上是常见的状态码，完整的状态码列表在这[状态码](http://www.restapitutorial.com/httpstatuscodes.html)

## HATEOAS

在介绍HATEOAS之前，先介绍一下REST的成熟度模型

在介绍 HATEOAS 之前，先介绍一下 Richardson 提出的 REST 成熟度模型。该模型把 REST 服务按照成熟度划分成 4 个层次：

* 第一个层次（Level 0）的 Web 服务只是使用 HTTP 作为传输方式，实际上只是远程方法调用（RPC）的一种具体形式。
* 第二个层次（Level 1）的 Web 服务引入了资源的概念。每个资源有对应的标识符和表达。
* 第三个层次（Level 2）的 Web 服务使用不同的 HTTP 方法来进行不同的操作，并且使用 HTTP 状态码来表示不同的结果。如 HTTP GET 方法来获取资源，HTTP DELETE 方法来删除资源。
* 第四个层次（Level 3）的 Web 服务使用 HATEOAS。在资源的表达中包含了链接信息。客户端可以根据链接来发现可以执行的动作。

### 简述

HATEOAS（Hypermedia as the engine of application state）是 REST 架构风格中最复杂的约束，也是构建成熟 REST 服务的核心。它的重要性在于客户端和服务器之间的解耦。

### 例子

#### 分页

request请求，查询user，每页显示10条，从第10条开始显示（第二页）

/users?limit=10&offset=10

response

{

data: {

xxxx

},

meta: {

\_link: [

{rel: 'self', href: 'xxx/users?limit=10&offset=10'},

{rel: 'first', href: 'xxx/users?limit=10&offset=0', title: 'first page'},

{rel: 'last', href: 'xxx/users?limit=10&offset=50', title: 'last page'},

{rel: 'prev', href: 'xxx/users?limit=10&offset=0', title: 'prev page'},

{rel: 'next', href: 'xxx/users?limit=10&offset=20', title: 'next page'}

]

}

}

\_link返回了5个资源

* rel: 'self'，资源本身
* rel: 'first'，第一页资源
* rel: 'last'，最后一页资源
* rel: 'prev'，上一页资源
* rel: 'next'，下一页资源

### 权限相关

如用户查询一个订单

#### 普通用户

request

/orders/1

response

{

data: {

xxx

},

meta: {

\_link: [

{rel: 'self', href: 'xxx/orders/1'},

{rel: 'related', href: 'xxx/orders/1/payment', title: 'pay the order'}

]

}

}

\_link返回两个资源

* rel: 'self'，资源本身
* rel: 'related'，与当前资源相关的资源，/order/1/payment用户可以使用此资源进行支付

#### 权限用户

request

/orders/1

response

{

data: {

xxx

},

meta: {

\_link: [

{rel: 'self', href: 'xxx/orders/1'},

{rel: 'edit', href: 'xxx/orders/1', title: 'edit the order'},

{rel: 'delete', href: 'xxx/orders/1', title: 'delete the order'}

]

}

}

此用户拥有修改与删除订单的权限，因此返回了3个资源

* rel: 'self'，资源本身
* rel: 'edit'，此用户可修改该资源
* rel: 'delete'，此用户可删除该资源

### 常用rel

| **rel** | **describe** |
| --- | --- |
| self | 资源本身，每个资源表述都一个包含此关系 |
| edit | 指向一个可以编辑当前资源的链接 |
| delete | 指向一个可以删除当前资源的链接 |
| item | 如果当前资源表示的是一个集合，则用来指向该集合中的单个资源 |
| collection | 如果当前资源包含在某个集合中，则用来指向包含该资源的集合 |
| related | 指向一个与当前资源相关的资源 |
| first、last、prev、next | 分别用来指向第一个、最后一个、上一个和下一个资源 |

### HATEOAS总结

由以上例子可以看出\_link就是以Hyperlink表述资源与资源之间的关系，这种方式使客户端与服务端能很好的分离开来，只要接口的定义不变，客户端与服务端就可以独立的开发和演变。