

# 目录

前言	1.1
RESTful API简介	1.2
举例	1.2.1
RESTful API通用设计规则	1.3
通用设计规则	1.3.1
RESTful API工具和库	1.4
开发测试工具	1.4.1
框架和库	1.4.2
RESTful API如何写接口文档	1.5
用Markdown写API文档	1.5.1
用Postman生成API文档	1.5.2
用Swagger生成API文档	1.5.3
RESTful API心得和经验	1.6
不好的设计风格	1.6.1
返回数据的格式和风格	1.6.2
其他心得	1.6.3
附录	1.7
参考资料	1.7.1

# HTTP后台端： RESTful API接口设计

## 简介

整理过[HTTP知识总结](#)后，继续去整理 HTTP 的后台相关的技术。在服务器后台进行设计API接口时，目前最流行的风格（原则/标准/规范）就是[RESTful](#)。

## 源码+浏览+下载

本书的各种源码、在线浏览地址、多种格式文件下载如下：

### Gitook源码

- [crifan/http\\_restful\\_api: HTTP后台端： RESTful API接口设计](#)

### 在线浏览

- [HTTP后台端： RESTful API接口设计 book.crifan.com](#)
- [HTTP后台端： RESTful API接口设计 crifan.github.io](#)

### 离线下载阅读

- [HTTP后台端： RESTful API接口设计 PDF](#)
- [HTTP后台端： RESTful API接口设计 ePub](#)
- [HTTP后台端： RESTful API接口设计 Mobi](#)

crifan.com, 使用[知识署名-相同方式共享4.0协议](#)发布 all right reserved, powered by Gitbook该文件修订时间： 2017-12-14 11:24:23

# RESTful API简介

服务器后台设计API接口时，目前最流行的风格（原则/标准/规范）就是 RESTful。

其中 REST = REpresentational State Transfer

- REST 直译：表现层状态转移
- REST 核心含义：无状态的资源
  - 资源的变化（CURD）都是通过操作去实现的
    - 资源可以用 URI 表示
    - 用不同的URI和方法，表示对资源的不同操作
      - 典型的：
        - GET：获取资源
        - POST：新建资源
        - DELETE：删除资源

## RESTful的通俗理解

借用某人的总结：

- 看 url 就知道要什么
- 看 http method 就知道干什么
- 看 http status code 就知道结果如何

## 其他类型的接口设计风格(含RESTful)

- ROA = Resource Oriented Architecture
- RPC = Remote Procedure Call
- SOA = Simple Object Access Protocol
- REST = REpresentational State Transfer

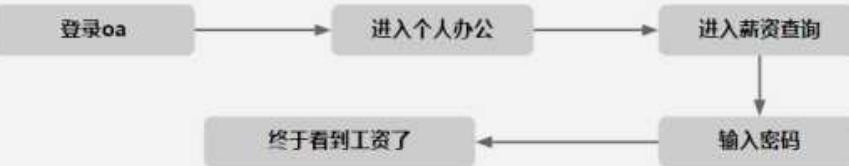
## 关于无状态的解释

### 有状态

## 有状态

- 无状态是相对于『有状态』而言的
- 我们平常接触到的网站，都是『有状态』的

如，在oa中查看基本工资：



**后面的每一个状态，都依赖于前面的状态  
没有一个url，能够直接定位到『张三』的『工资』**

Copyright© Gevin

## 无状态

### 无状态

对每个资源的请求，都不依赖于其他资源或其他请求  
每个资源，都是可寻址的，都有至少一个url能对其定位

- Application State
- Resource Stateless

RESTful 架构下，工资可以通过以下url查询：

张三工资 <http://oa.company.com/salary/zhangsan>  
李四工资 <http://oa.company.com/salary/lee4>

无状态更加方便客户端使用服务器的资源或服务

Copyright© Gevin

## RESTful API常见形式举例

下面找些常见的RESTful的API供参考和有个直观的概念：

### RESTful的订单的API

HTTP 方法	行为	示例
GET	获取资源的信息	<a href="http://example.com/api/orders">http://example.com/api/orders</a>
GET	获取某个特定资源的信息	<a href="http://example.com/api/orders/123">http://example.com/api/orders/123</a>
POST	创建新资源	<a href="http://example.com/api/orders">http://example.com/api/orders</a>
PUT	更新资源	<a href="http://example.com/api/orders/123">http://example.com/api/orders/123</a>
DELETE	删除资源	<a href="http://example.com/api/orders/123">http://example.com/api/orders/123</a>

### RESTful的客户的API

```
POST http://www.example.com/customers
POST http://www.example.com/customers/12345/orders

GET http://www.example.com/customers/12345
GET http://www.example.com/customers/12345/orders
GET http://www.example.com/buckets/sample

PUT http://www.example.com/customers/12345
PUT http://www.example.com/customers/12345/orders/98765
PUT http://www.example.com/buckets/secret_stuff

DELETE http://www.example.com/customers/12345
DELETE http://www.example.com/customers/12345/orders
DELETE http://www.example.com/bucket/sample
```

### RESTful的待办事项TodoList的API

HTTP 方法	URL	动作
GET	<a href="http://[hostname]/todo/api/v1.0/tasks">http://[hostname]/todo/api/v1.0/tasks</a>	检索任务列表
GET	<a href="http://[hostname]/todo/api/v1.0/tasks/[task_id]">http://[hostname]/todo/api/v1.0/tasks/[task_id]</a>	检索某个任务
POST	<a href="http://[hostname]/todo/api/v1.0/tasks">http://[hostname]/todo/api/v1.0/tasks</a>	创建新任务
PUT	<a href="http://[hostname]/todo/api/v1.0/tasks/[task_id]">http://[hostname]/todo/api/v1.0/tasks/[task_id]</a>	更新任务
DELETE	<a href="http://[hostname]/todo/api/v1.0/tasks/[task_id]">http://[hostname]/todo/api/v1.0/tasks/[task_id]</a>	删除任务

更多细节详见： [【整理】 TodoList待办事项：常被用于解释一个概念和框架如何应用](#))

crifan.com, 使用[知识署名-相同方式共享4.0协议](#)发布 all right reserved, powered by Gitbook该文件修订时间: 2017-12-20 10:29:20

# RESTful API通用设计规则

对于RESTful的API设计，有些通用的设计规则。其实也可以叫做HTTP的各种方法的典型用法。

crifan.com, 使用[知识署名-相同方式共享4.0协议](#)发布 all right reserved, powered by Gitbook该文件修订时间: 2017-12-08 11:34:27

# 通用设计规则

资源：往往对应着后台系统中对象，比如一个用户User，一个待办事项todo item，一个任务Task等等

用对应的接口表示要对资源进行何种操作，想要实现什么目的：

HTTP Verb=HTTP方法	操作类型=CRUD	返回	说明
POST	Create创建	<ul style="list-style-type: none"> <li>正常           <ul style="list-style-type: none"> <li>最常用：200 (OK)</li> <li>不太常用：201 (Created)</li> </ul> </li> <li>异常           <ul style="list-style-type: none"> <li>404 (Not Found)</li> <li>409 (Conflict)</li> </ul> </li> </ul>	
GET	Read读取	<ul style="list-style-type: none"> <li>正常：           <ul style="list-style-type: none"> <li>200 (OK)</li> </ul> </li> <li>异常：           <ul style="list-style-type: none"> <li>404 (Not Found)</li> </ul> </li> </ul>	
PUT/UPDATE/PATCH	Update/Replace 更新/替换	<ul style="list-style-type: none"> <li>正常：           <ul style="list-style-type: none"> <li>200 (OK)</li> </ul> </li> <li>异常           <ul style="list-style-type: none"> <li>405 (Method Not Allowed)</li> <li>204 (No Content)</li> <li>404 (Not Found)</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>PUT, UPDATE, PATCH 都常被用于更新（已有的）某个资源</li> <li>最最常用：PUT，另外也有用POST用作PUT去更新资源的</li> </ul>
DELETE	Delete删除	<ul style="list-style-type: none"> <li>正常：           <ul style="list-style-type: none"> <li>200 (OK)</li> <li>204 (No Content)</li> </ul> </li> <li>异常：           <ul style="list-style-type: none"> <li>404 (Not Found)</li> <li>405 (Method Not Allowed)</li> </ul> </li> </ul>	

## 举例：RESTful的某个类似于外卖的项目的API

此处给出之前做过一个项目的RESTful的API，供参考：

- 用户
  - 获取用户ID：支持多个参数，根据参数不同返回对应的值
    - GET /v1.0/open/userId?type=phone&phone={phone}

- GET /v1.0/open/userId?type=email&email={email}
- GET /v1.0/open/userId?type=facebook&facebookUserId={facebookUserId}
- 获取用户信息
  - GET /v1.0/users/{userId}
- 修改用户信息
  - PUT /v1.0/users/{userId}/info
- 修改密码
  - PUT /v1.0/users/{userId}/password
- 订单
  - 获取订单任务信息
    - GET /v1.0/tasks/{taskId}/users/{userId}
  - 发布任务
    - POST /v1.0/tasks/users/{userId}
  - 发单人确认任务信息
    - PUT /v1.0/tasks/{taskId}/users/{userId}/confirmInfo

crifan.com, 使用[知识署名-相同方式共享4.0协议](#)发布 all right reserved, powered by Gitbook该文件修订时间: 2017-12-14 13:51:12

# RESTful API工具和库

下面介绍和RESTful API开发、设计时相关的一些工具、库。

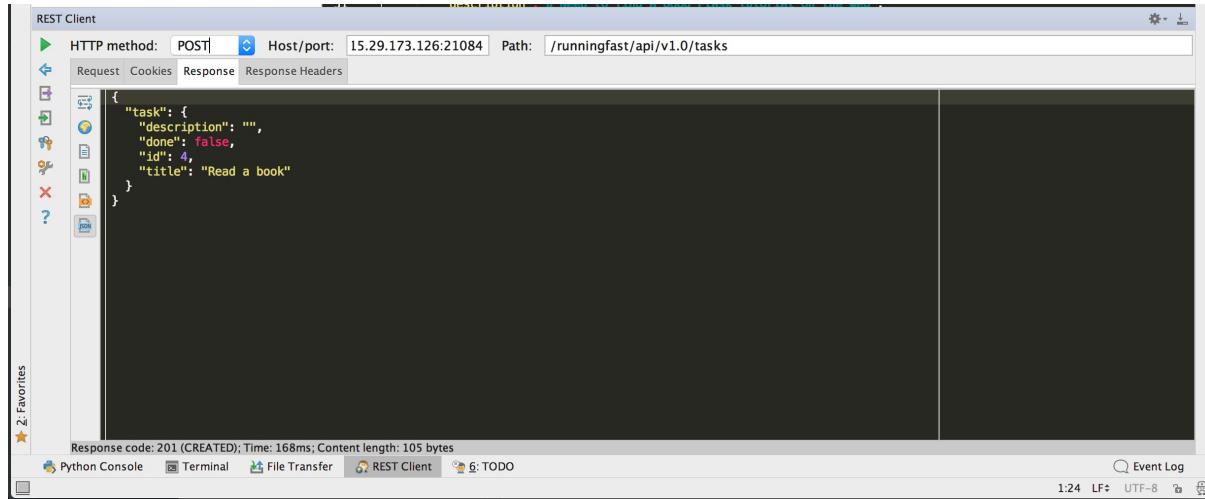
crifan.com, 使用[知识署名-相同方式共享4.0协议](#)发布 all right reserved, powered by Gitbook该文件修订时间: 2017-12-14 09:59:54

## RESTful的API测试工具

### Postman

详见另外的教程：[API开发利器：Postman](#)

### PyCharm中的Restful API测试工具



详见：[【整理】flask restful api 测试工具](#)

### Chrome插件： Advanced REST client

详见：[\[记录\] chrome的websocket插件：Advanced REST client](#)

crifan.com, 使用[知识署名-相同方式共享4.0协议](#)发布 all right reserved, powered by Gitbook该文件修订时间： 2017-12-29 11:15:07

# Restful的API开发设计工具和框架

## Python语言

[Flask-RESTful](#)

## Javascript

[NodeJS的ExpressJS](#)

## PHP

[Slim](#)

## Ruby

[Sinatra](#)

## .NET

[Nancy - Lightweight Web Framework for .net](#)

crifan.com, 使用[知识署名-相同方式共享4.0协议](#)发布 all right reserved, powered by Gitbook该文件修订时间: 2017-12-30 22:55:05

# 如何写API接口文档

如果你是后台API开发人员，往往会为了写清晰的API接口文档而发愁 此处，自己的建议和经验是：

- 方法1: 用 `markdown` 写API文档
- 方法2: 用 [Postman](#) 生成API文档
- 方法3: 用 [Swagger](#) 写（设计API接口的同时就可以生成出）API文档
  - 并可生成对应的后台和前端的代码
  - 剩下只需要编写业务逻辑代码即可

后来发现其他还有一些API文档工具，比如：

- [docute](#)
- [ShowDoc](#)
- [小么鸡 接口文档管理工具](#)
  - 也支持API接口调试

下面详细介绍这些写API文档的不同方法。

crifan.com, 使用[知识署名-相同方式共享4.0协议](#)发布 all right reserved, powered by Gitbook该文件修订时间： 2017-12-29 11:17:31

# 用Markdown写API文档

举例：一个GET方法，用于获取验证码的接口：在postman中已经调试完毕：

The screenshot shows the Postman interface with the following details:

- History:** All, Me, Team, Collections (highlighted).
- Collections:** runningfast (23 requests).
- Request URL:** /open/smscode
- Method:** GET
- URL Parameters:** type=register&phone=13511113333
- Headers:** None
- Body:** None
- Params:** type: register, phone: 13511113333
- Buttons:** Send, Save

然后去（推荐）有道云笔记中编写markdown：

```

# API接口
## 注册
#### 获取验证码

目前有4种短信验证码，对应的type是：
- 注册短信验证码：register
- 修改密码短信验证码：changePassword
- 修改手机号短信验证码：changePhoneNumber
- 验证手机号短信验证码：verifyPhoneNumber

##### Request
- Method: *`GET`*
- URL: ```/v1.0/open/smscode?type={type}&phone={phone}```
  - register for new user: ```/v1.0/open/smscode?type=register&phone=13811119999```
  - forgot password: ```/v1.0/open/smscode?type=changePassword&phone=13822224444```
- Headers:
- Body:
```

#####
##### Response
- Body
```
{
  "code": 200,
  "data": "730781",
  "message": "OK"
}
```

```

注意：为了防止短信验证码被滥用，短信如果发送后，需要隔60s才能重新发送。

对应的效果：

```

53
54  ### 获取验证码
55
56
57  目前有4种短信验证码，对应的type是：
58  - 注册短信验证码: register
59  - 修改密码短信验证码: changePassword
60  - 修改手机短信验证码: changePhoneNumber
61  - 验证手机号短信验证码: verifyPhoneNumber
62
63  ##### Request
64  - Method: **GET**
65  - URL: ``/v1.0/open/smscode?type={type}&phone={phone}```
66    - register for new user:
67      ``/v1.0/open/smscode?type=register&phone=13811119999```
68    - forgot password:
69      ``/v1.0/open/smscode?type=changePassword&phone=13822224444```
70  - Headers:
71  - Body:
72  ...
73
74  ##### Response
75  - Body
76

```

## 获取验证码

目前有4种短信验证码，对应的type是：

- 注册短信验证码: register
- 修改密码短信验证码: changePassword
- 修改手机短信验证码: changePhoneNumber
- 验证手机号短信验证码: verifyPhoneNumber

### Request

- Method: GET
- URL: /v1.0/open/smscode?type={type}&phone={phone}
  - register for new user: /v1.0/open/smscode?type=register&phone=13811119999
  - forgot password: /v1.0/open/smscode?type=changePassword&phone=13822224444
- Headers:
- Body:

```

64  Method: GET
65  - URL: ``/v1.0/open/smscode?type={type}&phone={phone}```
66    - register for new user:
67      ``/v1.0/open/smscode?type=register&phone=13811119999```
68    - forgot password:
69      ``/v1.0/open/smscode?type=changePassword&phone=13822224444```
70  - Headers:
71  - Body:
72  ...
73
74  ##### Response
75  - Body
76
77  {
78    "code": 200,
79    "data": "730781",
80    "message": "OK"
81  }
82
83
84  注意：为了防止短信验证码被滥用，短信如果发送后，需要隔60s才能重新发送。
85
86  ##### 忘记密码 / 修改密码

```

### Request

- Method: GET
- URL: /v1.0/open/smscode?type={type}&phone={phone}
  - register for new user: /v1.0/open/smscode?type=register&phone=13811119999
  - forgot password: /v1.0/open/smscode?type=changePassword&phone=13822224444
- Headers:
- Body:

### Response

- Body

```
{
  "code": 200,
  "data": "730781",
  "message": "OK"
}
```

注意：为了防止短信验证码被滥用，短信如果发送后，需要隔60s才能重新发送。

另外，再举个有request也有response的POST的例子：

```

##### 创建新用户
##### Request

- Method: **POST**
- URL: ``/v1.0/open/register``
- Headers: Content-Type:application/json

```

```
- Body:  
  ``  
 {  
   "phone" : "13511112222",  
   "smsCode" : "730781",  
   "email" : "crifan@webonn.com",  
   "firstName" : "crifan",  
   "lastName" : "Li",  
   "password" : "654321",  
   "facebookUserId" : "123907074803456"  
 }  
 ``  
  
##### Response  
- Body  
  ``  
 {  
   "code": 200,  
   "data": {  
     "avatarUrl": "",  
     "createdAt": "2016-10-24T20:39:46",  
     "curRole": "IdleNoRole",  
     "email": "crifan@webonn.com",  
     "errandOrRating": 0,  
     "facebookUserId": "123907074803456",  
     "firstName": "crifan",  
     "id": "user-4d51faba-97ff-4adf-b256-40d7c9c68103",  
     "isOnline": false,  
     "lastName": "Li",  
     "location": {  
       "createdAt": null,  
       "fullStr": null,  
       "id": null,  
       "latitude": null,  
       "longitude": null,  
       "shortStr": null,  
       "updatedAt": null  
     },  
     "locationId": null,  
     "password": "654321",  
     "phone": "13511112222",  
     "shareCodeCount": 0,  
     "updatedAt": "2016-10-24T20:39:46"  
   },  
   "message": "new user has created"  
 }  
 ``
```

markdown生成文档的效果：

◀ 返回 RunningFast API 20161226

---

## 创建新用户

### Request

- Method: POST
- URL: /v1.0/open/register
- Headers: Content-Type:application/json
- Body:

```
{  
    "phone" : "13511112222",  
    "smsCode" : "730781",  
    "email" : "crifan@webonn.com",  
    "firstName" : "crifan",  
    "lastName" : "Li",  
    "password" : "654321",  
    "facebookUserId" : "123907074803456"  
}
```

### Response

- Body

```
{  
    "code": 200,  
    "data": {  
        "avatarUrl": "",  
        "createdAt": "2016-10-24T20:20:46"  
    }  
}
```

◀ 返回 RunningFast API 20161226

## Response

- Body

```
{
  "code": 200,
  "data": {
    "avatarUrl": "",
    "createdAt": "2016-10-24T20:39:46",
    "curRole": "IdleNoRole",
    "email": "crifan@webonn.com",
    "errandorRating": 0,
    "facebookUserId": "123907074803456",
    "firstName": "crifan",
    "id": "user-4d51faba-97ff-4adf-b256-40d7c9c68103",
    "isOnline": false,
    "lastName": "Li",
    "location": {
      "createdAt": null,
      "fullStr": null,
      "id": null,
      "latitude": null,
      "longitude": null,
      "shortStr": null,
      "updatedAt": null
    },
    "locationId": null,
    "password": "654321",
    "phone": "13511112222",
    "shareCodeCount": 0,
    "updatedAt": "2016-10-24T20:39:46"
  },
  "message": "new user has created"
}
```

所以后续其他接口，均可参考上面的GET/POST等接口的写法，去写出对应的markdown的源文件，生成API文档后，效果还是不错的。

当然，也可以用其他Markdown编辑器去写md文件，去生成对应API文档。

另外，再附上，在写具体单个API接口之前的声明的部分：

```
# 文档说明
## 服务器API地址
前缀：
```http://115.29.173.126:21084/runningfast/api```

完整的API地址为：```前缀```+```具体接口路径```

比如，获取验证码都接口为：
```http://115.29.173.126:21084/runningfast/api``` + ```/v1.0/open/smscode```
->
```http://115.29.173.126:21084/runningfast/api/v1.0/open/smscode```
```

```
## 调用接口说明
- 如果参数格式是==JSON==的话：提交request请求时必须添加header头： ==Content-Type:application/json==
- 请求中是需要包含头信息： ==Authorization:{accesstoken}==
    - 接口中==包含==``/open/``的：不需要添加
    - 接口中==不包含==``/open/``：需要添加
        - 说明该接口都需要对应的权限才可以访问，所以需要在请求中包含头信息：```Authorization:{accesstoken}```
        - 当access token无效或者已过期时，返回：
```
{
    "code": 401,
    "message": "invalid access token: wrong or expired"
}
```

- 所有的接口的返回形式都是统一为：
    - 正常返回
```
{
    "code": 200,
    "message": "OK",
    "data": 某种类型的数据，比如字符串，数字，字典等等
}
```

    - 错误返回
```
{
    "code": 具体的错误码,
    "message": "具体的错误信息字符串"
}
```
``
```

文档效果：

&lt; 返回 RunningFast API 20161226

 编辑

## 文档说明

### 服务器API地址

前缀: `http://115.29.173.126:21084/runningfast/api`

完整的API地址为: 前缀 + 具体接口路径

比如, 获取验证码都接口为: `http://115.29.173.126:21084/runningfast/api + /v1.0/open/smscode`

->

```
http://115.29.173.126:21084/runningfast/api/v1.0/open/smscode
```

### 调用接口说明

- 如果参数格式是JSON的话: 提交request请求时必须添加header头: `Content-Type:application/json`
- 请求中是否要包含头信息: `Authorization:{accesstoken}`
  - 接口中包含 `/open/` 的: 不需要添加
  - 接口中不包含 `/open/` : 需要添加
    - 说明该接口都需要对应的权限才可以访问, 所以需要在请求中包含头信息: `Authorization:{accesstoken}`
  - 当access token无效或者已过期时, 返回:

```
{
```

&lt; 返回 RunningFast API 20161226

 编辑

- 接口中包含 `/open/` 的: 不需要添加
- 接口中不包含 `/open/` : 需要添加
  - 说明该接口都需要对应的权限才可以访问, 所以需要在请求中包含头信息: `Authorization:{accesstoken}`
- 当access token无效或者已过期时, 返回:

```
{
  "code": 401,
  "message": "invalid access token: wrong or expired"
}
```

- 所有的接口的返回形式都是统一为:
  - 正常返回

```
{
  "code": 200,
  "message": "OK",
  "data": 某种类型的数据, 比如字符串, 数字, 字典等等
}
```

- 错误返回

```
{  
    "code": "具体的错误码,  
    "message": "具体的错误信息字符串"  
}
```

- 优点：简单易上手
- 缺点：后续API更新后，需要及时更新markdown的文档内容

crifan.com, 使用[知识署名-相同方式共享4.0协议](#)发布 all right reserved, powered by Gitbook该文件修订时间: 2017-12-14 14:19:25

# 用Postman生成API文档

步骤：

1. Collection
2. 鼠标移动到某个Collection
3. 点击三个点
4. Publish Docs
5. Publish
6. Public URL
7. 别人打开这个Public URL即可查看API文档

效果：

The screenshot shows the Postman interface with the '奶牛云' collection selected. The left sidebar lists various API endpoints with their methods and URLs. The main content area displays two API endpoints: 'GET index/TodoNum' and 'GET index/CurMonthTodoNum'. Each endpoint has its URL and a 'Sample Request' section showing a curl command. The top right corner shows the 'Publish' button, the environment 'No environment', and the user '李茂 (crifan)'.

详见教程：[API开发利器：Postman](#)

- 优点：
  - 方便
    - 因为本身往往已用Postman调试接口，调试完毕后，即可发布
  - 及时更新文档
    - 同理，在后台代码更新后，用Postman调试无误后，即可再次点击发布即可，无须手动修改API文档
  - 美观
    - Postman生成的在线的API文档已足够清晰和美观
- 缺点：
  - 必须依赖于在Postman中调试接口



# 用Swagger写（设计API接口的同时就可以生成出）API文档

效果： API Development Tools | Swagger Editor | Swagger

The screenshot shows the Swagger Editor interface. On the left, the OpenAPI specification for the Uber API is displayed in YAML format. On the right, the generated API documentation is shown under the heading "Uber API". The documentation includes sections for "Paths", "Responses", and "Parameters". A specific endpoint, "/products", is highlighted with a blue background, showing its details: a GET method returning an array of products. The "Summary" field is "Product Types", and the "Description" field is "The Products endpoint returns information about the \*Uber\* products offered at a given location. The response includes the display name and other details about each product, and lists the products in the proper display order." The "Parameters" section shows two query parameters: "latitude" and "longitude", both of type number. The "Responses" section shows two possible outcomes: a successful 200 response with a schema of "An array of products" (which is expanded to show a "Product" schema) and a default response for unexpected errors.

```

1: # this is an example of the Uber API
2: # as a demonstration of an API spec in YAML
3: swagger: '2.0'
4: info:
5:   title: Uber API
6:   description: Move your app forward with the Uber API
7:   version: 1.0.0
8:   termsOfService:
9:     host: api.uber.com
10:    # array of all schemes that your API supports
11:   schemes:
12:     - https
13:    # will be prefixed to all paths
14:   basePath: /v1
15:   produces:
16:     - application/json
17:   paths:
18:     /products:
19:       get:
20:         summary: Product Types
21:         description: |
22:           The Products endpoint returns information about the *Uber* products
23:           offered at a given location. The response includes the display name
24:           and other details about each product, and lists the products in the
25:           proper display order.
26:         parameters:
27:           - name: latitude
28:             in: query
29:             description: Latitude component of location.
30:             required: true
31:             type: number
32:             format: double
33:           - name: longitude
34:             in: query
35:             description: Longitude component of location.
36:             required: true
37:             type: number
38:             format: double
39:         tags:
40:           - Products
41:         responses:
42:           200:
43:             description: An array of products
44:             schema:
45:               type: array
46:               items:
47:                 $ref: '#/definitions/Product'
48:             default:
49:               description: Unexpected error
50:             schema:

```

详见：【整理】swagger OpenAPI

- 优点：
  - 设计API接口的同时就是编写好了API文档
    - 因为有对应的工具可以直接生成API文档
  - 另外可以同时生成服务器端和客户端的代码
    - 剩下的只需要自己编写业务逻辑即可，支持N多种编程语言
  - 美观
    - 生成的API文档层次够清晰，够美观
- 缺点：
  - 必须用swagger去设计和编写API文档

crifan.com, 使用[知识署名-相同方式共享4.0协议](#)发布 all right reserved, powered by Gitbook该文件修订时间： 2017-12-14 11:34:53

# RESTful API的心得和经验

此处把之前折腾过的RESTful的一些心得和经验整理出来，供参考。

crifan.com, 使用[知识署名-相同方式共享4.0协议](#)发布 all right reserved, powered by Gitbook该文件修订时间: 2017-12-08 11:56:53

## 不好的API设计风格

### 在接口中添加GET/UPDATE等动词

比如：

- GET /getUser
- POST /updateUser
- POST /cowfarm/cowfarmemp/new
- POST /cowfarm/cowfarmemp/update
- POST /cowfarm/cowfarmemp/delete/{id}
  - 且返回值中包含了data和对应的字段

实际上不应该在接口中加这些动词，而应该通过接口的HTTP方法，GET/UPDATE，来表示接口的含义，比如改为：

- GET /user
  - 获取一个用户的信息
- PUT /user
  - 更新用户的信息
- POST /cowfarm/employee
  - 表示 新建一个农场的雇员/员工
- PUT /cowfarm/employee
  - 表示 更新农场雇员/员工的信息
- DELETE /cowfarm/employee
  - body中包含json参数
    - {"id": xxx}
  - 表示删除用户
  - 且返回值中， message, code应该正常返回， data就没必要返回了。

### 非改动资源的操作却设计为POST/PUT等方法

对于没有新增/更新/删除等去改动和影响资源的操作，HTTP的方法却设计为POST/PUT等

- 很多公司的后台开发人员，为了偷懒省事，所有的接口都用POST，包括本应该用GET的接口
- 或者是，对API接口设计规范不了解，把仅仅是获取、查询资源，不会改动资源的接口设计成POST

比如，不好的做法：

- 通过id获取task信息： POST /task
  - body参数： { "id": "1234" }
- 查询出符合条件的任务： POST /task/query
  - 参数放在body中 { "keyword": "xxx", "start": 0, "limit": 10 } 应该改为正常的做法：
- GET /task/{id}
  - 或： GET /task?id=1234
- GET /task/query?keyword=xxxx&start=0&limit=10
  - 其中GET的query string在客户端调用API接口时，往往不是手动加上去
  - 而是传递一个字典变量，然后用相关的encode函数去编码出来的
  - 详见：[HTTP知识总结](#)
  - 这样才能确保参数中包含特殊字符，服务器端也能正常接受，比如：
    - 空格 -> %20

- 中文"李茂" -> 被UTF-8编码为 -> %e6%9d%8e%e8%8c%82

crifan.com, 使用[知识署名-相同方式共享4.0协议](#)发布 all right reserved, powered by Gitbook该文件修订时间: 2017-12-20 10:27:29

# RESTful API接口返回数据的格式和风格

常见返回的数据一般用JSON。

对应返回的内容，常见的做法是：

- `code` : http的status code
  - 如果有自己定义的额外的错误，那么也可以考虑用自己定义的错误码
- `message` : 对应的文字描述信息
  - 如果是出错，则显示具体的错误信息
  - 否则操作成功，一般简化处理都是返回OK
- `data`
  - 对应数据的json字符串
    - 如果是数组，则对应最外层是[]的 list
    - 如果是对象，则对应最外层是{}的 dict

比如之前某项目中设计的返回的数据格式：

1. `code`是200 创建用户 `POST /v1.0/open/register` 返回：

```
{  
    "code": 200,  
    "message": "new user has created",  
    "data": {  
        "id": "User-4d51faba-97ff-4adf-b256-40d7c9e68103",  
        "firstName": "crifan",  
        "lastName": "Li",  
        "password": "654321",  
        "phone": "13511112222",  
        "createdAt": "2016-10-24T20:39:46",  
        "updatedAt": "2016-10-24T20:39:46"  
        ....  
    }  
}
```

2. `code`是401

```
{  
    "code": 401,  
    "message": "invalid access token: wrong or expired"  
}
```

## 其他RESTful API心得

### api中是否一定要加版本号?

如果是为了设计长期稳定的API接口，则最好是加上版本号v1.0这种写法 `http://[hostname]/todo/api/v1.0/` 但是往往中小型项目不需要这么长期维护和不需要迭代太多版本，则可以考虑不需要版本号，则可以写成：

`http://[hostname]/todo/api/` 即可。

另外了解到：有些的做法是把API的版本号v1，放到request header中。->github就是这么做的：[Media Types | GitHub Developer Guide](#)

### 设计Restful的接口时，尽量用复数，且统一

即，用 `/artists` 而不要用 `/artist`

### 如果有多个对象，用模块化逻辑，嵌套资源去设计接口

举例：

获取某个（内部id为8的）歌手的所有专辑：`GET /artists/8/albums`

### 当查询返回的数据多了则：paging分页

如上，如果一个歌手的专辑太多，则应该使用分页 `paging`

设计分页的API时返回的数据的格式，可参考之前某项目中的返回的格式：

```
{
  "code": 200,
  "message": "get task/orders ok",
  "data": {
    "curPageNum": 2,
    "hasNext": false,
    "hasPrev": true,
    "numPerPage": 10,
    "tasks": [
      "task-10b01105-ec53-41bb-810e-720ab468bdf7": {.....},
      "task-da013992-e7aa-4ae9-8b6f-bdf621b9fbba": {.....},
      "task-f3c0c660-e7f5-4583-bab2-23c7006dadc4": {.....},
      "task-f7a4d0df-3142-444b-a962-83660acd447f": {.....}
    ],
    "totalNum": 14,
    "totalpageNum": 2
  }
}
```

具体的解释：Pagination的对象中，最常用到的属性就是：

- `items`：具体有多少个对象，是个列表
  - 然后就可以通过`items`去获取每个对象的详细信息了。其他还有一些常用属性：
- `page`：当前的页数
- `pages`：总的页数
- `per_page`：每一页的个数

- `has_prev` : 是否有前一页
- `has_next` : 是否有后一页
- `total` : (符合当前分页查询的) 总 (的目的) 个数

具体的Python的 Flask+SQLAlchemy 的API的代码，供参考：

```

curPageTaskList = None
taskPagination = None

if curRole == UserRole.Initiator:
    taskPagination = Task.query.filter_by(initiatorId=userId).paginate(
        page=curPageNum,
        per_page=numberPerPage,
        error_out=False)
elif curRole == UserRole.Errandor:
    taskPagination = Task.query.filter_by(errandorId=userId).paginate(
        page=curPageNum,
        per_page=numberPerPage,
        error_out=False)

paginatedTaskList = taskPagination.items

paginatedTaskDict = {}
for curIdx, eachTask in enumerate(paginatedTaskList):
    paginatedTaskDict[eachTask.id] = marshal(eachTask, task_fields)

respPaginatedTaskInfoDict = {
    "curPageNum": taskPagination.page,
    "totalPageNum": taskPagination.pages,
    "numPerPage": taskPagination.per_page,
    "hasPrev": taskPagination.has_prev,
    "hasNext": taskPagination.has_next,
    "totalTaskNum": taskPagination.total,
    "tasks": paginatedTaskDict
}

```

详见：【已解决】Flask-Restful中如何设计分页的API

## 是否一定要严格按照Restful的规则，用对应的http的method实现对应的功能？

一般来说，不用非常严格的依照规则，尤其是

- UPDATE/PATCH 去更新修改资源
  - 往往为了简化，用PUT表示 更新修改资源即可

不过，有些项目，对方本身就要求设计接口时，严格按照Restful的规则来设计，这时最好用UPDATE/PATCH去更新修改资源。

其他的，见上面的表格总结，典型的是：

- GET 获取资源
- POST 新建资源
- PUT 更新/修改 资源
- DELETE 删除资源

## POST时body中无参数时不应该添加 Content-Type:application/json 的Header

比如POST时,如果没有Body内容参数传递时，Header中就不要包含 `Content-Type:application/json`

否则，某些服务器就会返回错误。

比如Flask的Flask-Restful的接口就会自动返回：

```
code = 0 message = Failed to decode JSON object: No JSON object could be decoded
```

-》其意思是，你指定了

```
Content-Type:application/json
```

所以框架就会去尝试从Body中找JSON字符串，去解析参数

但是发现Body是空的，没有可用的JSON字符串待解析，所以报这个错误。

crifan.com, 使用[知识署名-相同方式共享4.0协议](#)发布 all right reserved, powered by Gitbook该文件修订时间：2017-12-14 14:23:30

## 附录

下面列出相关参考资料。

crifan.com, 使用[知识署名-相同方式共享4.0协议](#)发布 all right reserved, powered by Gitbook该文件修订时间: 2017-12-14 11:21:48

## 参考资料

- 怎样用通俗的语言解释什么叫 REST, 以及什么是 RESTful? - 计算机网络 - 知乎
- 使用 Python 和 Flask 设计 RESTful API — Designing a RESTful API with Python and Flask 1.0 documentation
- restful update put When to use PUT or POST - The RESTful cookbook
- PUT Versus POST > RESTful APIs in the Real World Course 1 | KnpUniversity
- http - PUT vs. POST in REST - Stack Overflow
- HTTP Methods for RESTful Services
- RFC 5789 - PATCH Method for HTTP
- RESTful Services Quick Tips
- RESTful API 设计指南 - 阮一峰的网络日志
- 设计一个务实的RESTful API
- RESTful API 编写指南
- API Creation - Full Stack Python
- REST best practices

crifan.com, 使用[知识署名-相同方式共享4.0协议](#)发布 all right reserved, powered by Gitbook该文件修订时间: 2017-12-14 11:00:40