RESTful API

设计与实现规范

文档类型：	标准规范	**当前版本：**	v1.0
文档状态：	[√] 草　稿	**文档编制：**	数字化研究院架构组
	[　] 讨论稿	**完成日期：**	2020.03.06
	[　] 正式稿		

文档信息

项目名称	N/A		
文档名称	北控水务 RESTful API 设计规范		
文档编制	数字化研究院架构组	**当前版本**	v1.0
相关人员		**完成日期**	2020.03.06
审核批准		**审核日期**	

文档修改历史

版本号	版本日期	编制人	审核人/批准人	修改说明
v0.1	2020-03-06	段立功		初稿

目 录

1. 概述

1.1. 目的

本规范各条标准制定的目的：提高系统REST API的可用性、一致性、健壮性、安全性，避免设计RESTful API时易犯的错误，有利于系统及服务之间的解耦以及未来的系统维护与升级，便于自动化测试。

本软件技术规范的内容包括：RESTful API的Protocol(协议)、Root URL、版本、Endpoint、HTTP动词、过滤、认证、响应及安全。

1.2. 适用范围

本规范适用于从事本集团IT项目的开发与设计人员。

1.3. 说明

为了避免歧义，文档大量使用了「能愿动词」，对应的解释如下：

- 必须 (MUST)：绝对，严格遵循，请照做，无条件遵守；
- 一定不可 (MUST NOT)：禁令，严令禁止；
- 应该 (SHOULD)：强烈建议这样做，但是不强求；
- 不该 (SHOULD NOT)：强烈不建议这样做，但是不强求；
- 可以 (MAY) 和 可选 (OPTIONAL)：选择性高一点，在这个文档内，此词语使用较少。

2. 设计原则

- API优先设计。各业务系统或服务应该提供对外的RESTful API，在系统及服务实现之前设计API；
- 易用且不易误用。API设计一定不可太复杂，要简单易用，而且还不能容易用错；

- API响应符合预期(Least Astonishment)。API后台的行为及返回结果应该符合多数人的预期，能够自解释；
- 围绕用例设计。API设计要围绕User Story或者Use Case来进行，在一个业务场景下提供完整的闭环操作。

3. 规范

3.1. Protocol

客户端在通过 API 与后端服务通信的过程中，应该使用 HTTPS 协议。

3.2. API Root URL

API 的根入口点应尽可能保持足够简单，这里有两个常见的 URL 根例子：

```
- api.example.com/*
- example.com/api/*
```

如果你的应用很庞大或者你预计它将会变的很庞大，那应该将 API 放到子域下（api.example.com）。这种做法可以保持某些规模化上的灵活性。

3.3. Versioning

所有的API必须保持向后兼容，必须在引入新版本API的同时确保旧版本API仍然可用。所以应该为其提供版本支持。

目前比较常见版本形式是在URL中嵌入版本编号

```
api.example.com/v1/*
```

这种做法使得版本号直观、易于调试。

3.4. Endpoints

端点就是指向特定资源或资源集合的URL。在端点的设计中，你必须遵守下列约定：

- URL 的命名必须全部小写

- URL 中资源（resource）的命名必须是名词，并且必须是复数形式
- 必须优先使用Restful类型的 URL
- URL必须是易读的
- URL一定不可暴露服务器架构

至于 URL 是否必须使用连字符（`-`） 或下划线（`_`），不做硬性规定，但必须根据产品情况统一一种风格。

错误的例子：

```
- https://api.example.com/getUserInfo?userid=1
- https://api.example.com/getusers
- https://api.example.com/sv/u
- https://api.example.com/cgi-bin/users/get_user.php?userid=1
```

正确的例子：

```
- https://api.example.com/zoos
- https://api.example.com/animals
- https://api.example.com/zoos/{zoo}/animals
- https://api.example.com/animal_types
- https://api.example.com/employees
```

3.5. **HTTP 动词**

对于资源的具体操作类型，由HTTP动词表示。常用的HTTP动词有下面五个:

- GET（Select）：从服务器端取出资源（一项或多项）；
- POST（Create）：在服务器端新建一个资源，该方法不具备幂等性；

- PUT（Replace（Create or Update））：在服务器端更新资源（客户端提供改变后的完整资源），该方法具备幂等性；
- PATCH（UPDATE）：在服务器端更新资源局部（客户端提供改变的属性）；
- DELETE（DELETE）：从服务器端删除资源。

其中，

- 删除资源必须用DELETE方法
- 创建新的资源必须使用POST方法
- 更新资源应该使用PUT方法
- 更新资源部分属性应该使用PATCH方法
- 获取资源信息必须使用GET方法

针对每一个端点来说，下面列出所有可行的HTTP动词和端点的组合

请求方法	URL	描述
GET	/zoos	列出所有的动物园(ID和名称，不要太详细)
POST	/zoos	新增一个新的动物园
GET	/zoos/{zoo}	获取指定动物园详情
PUT	/zoos/{zoo}	更新指定动物园(整个对象)
PATCH	/zoos/{zoo}	更新动物园(部分对象)
DELETE	/zoos/{zoo}	删除指定动物园
GET	/zoos/{zoo}/animals	检索指定动物园下的动物列表(ID和名称，不要太详细)
GET	/animals	列出所有动物(ID和名称)。
POST	/animals	新增新的动物
GET	/animals/{animal}	获取指定的动物详情
PUT	/animals/{animal}	更新指定的动物(整个对象)

PATCH	/animals/{animal}	更新指定的动物(部分对象)
GET	/animal_types	获取所有动物类型(ID和名称，不要太详细)
GET	/animal_types/{type}	获取指定的动物类型详情
GET	/employees	检索整个雇员列表
GET	/employees/{employee}	检索指定特定的员工
GET	/zoos/{zoo}/employees	检索在这个动物园工作的雇员的名单(身份证和姓名)
POST	/employees	新增指定新员工
POST	/zoos/{zoo}/employees	在特定的动物园雇佣一名员工
DELETE	/zoos/{zoo}/employees/{employee}	从某个动物园解雇一名员工

超出Restful端点的，应该模仿上表的方式来定义端点。

3.6. Filtering

如果记录数量很多，服务器不可能都将它们返回给用户。API应该提供参数，过滤返回结果。下面是一些常见的参数。

- ?limit=10：指定返回记录的数量
- ?offset=10：指定返回记录的开始位置。
- ?page=2&per_page=100：指定第几页，以及每页的记录数。
- ?sortby=name&order=asc：指定返回结果按照哪个属性排序，以及排序顺序。
- ?animal_type_id=1：指定筛选条件

所有URL参数必须是全小写，必须使用下划线类型的参数形式。

分页参数必须固定为page、per_page。

经常使用的、复杂的查询应该标签化，降低维护成本。如

```
GET /trades?status=closed&sort=sortby=name&order=asc
```

可为其定制快捷方式：

```
GET /trades/recently_closed
```

3.7. **Authentication**

应该使用OAuth2.0的方式为API调用者提供登录认证。必须先通过登录接口获取 Access Token后再通过该token调用需要身份认证的API。

OAuth的端点设计示列：

- RFC 6749 /token
- Twitter /oauth2/token
- Fackbook /oauth/access_token
- Google /o/oauth2/token
- Github /login/oauth/access_token
- Instagram /oauth/authorize

客户端在获得access token的同时必须在响应中包含一个名为expires_in的数据，它表示当前获得的token会在多少秒后失效。

```
{
    "access_token": "token....",
    "token_type": "Bearer",
    "expires_in": 3600
}
```

客户端在请求需要认证的API时，必须在请求头Authorization中带上access_token。

```
Authorization: Bearer token...
```

当超过指定的秒数后， access token就会过期，再次用过期/或无效的token访问时，服务端应该返回invalid_token的错误或401错误码。

```
HTTP/1.1 401 Unauthorized
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache

{
    "error": "invalid_token"
}
```

3.8. **Response**

所有的API响应，必须遵守HTTP设计规范，必须选择合适的HTTP状态码。

一定不可所有接口都返回状态码为200的HTTP响应，如：

```
HTTP/1.1 200 ok
Content-Type: application/json
Server: example.com

{
    "code": 0,
    "msg": "success",
    "data": {
        "username": "username"
    }
```

```
}
```

或

```
HTTP/1.1 200 ok
Content-Type: application/json
Server: example.com

{
    "code": -1,
    "msg": "该活动不存在",
}
```

下表列举了常见的HTTP状态码

状态码	描述
1xx	代表请求已被接受，需要继续处理
2xx	请求已成功，请求所希望的响应头或数据体将随此响应返回
3xx	重定向
4xx	客户端原因引起的错误
5xx	服务端原因引起的错误

只有来自客户端的请求被正确的处理后才能返回2xx的响应，所以当 API 返回2xx类型的状态码时，前端必须认定该请求已处理成功。

必须强调的是，所有API一定不可返回1xx类型的状态码。当API发生错误时，必须返回出错时的详细信息。目前常见返回错误信息的方法有两种：

1、将错误详细放入 HTTP响应首部；

```
X-MYNAME-ERROR-CODE: 4001
X-MYNAME-ERROR-MESSAGE: Bad authentication token
X-MYNAME-ERROR-INFO: http://docs.example.com/api/v1/authentication
```

2、直接放入响应实体中；

```
HTTP/1.1 401 Unauthorized
Server: nginx/1.11.9
Content-Type: application/json
Transfer-Encoding: chunked
Cache-Control: no-cache, private
Date: Sun, 24 Jun 2018 10:02:59 GMT
Connection: keep-alive

{"error_code":40100,"message":"Unauthorized"}
```

考虑到易读性和客户端的易处理性，必须把错误信息直接放到响应实体中，并且错误格式应该满足如下格式：

```
{
    "message": "您查找的资源不存在",
    "error_code": 404001
}
```

其中错误码（error_code）必须和HTTP状态码对应，也方便错误码归类，如：

```
HTTP/1.1 429 Too Many Requests
Server: nginx/1.11.9
Content-Type: application/json
```

```
Transfer-Encoding: chunked
Cache-Control: no-cache, private
Date: Sun, 24 Jun 2018 10:15:52 GMT
Connection: keep-alive

{"error_code":429001,"message":"你操作太频繁了"}
```

```
HTTP/1.1 403 Forbidden
Server: nginx/1.11.9
Content-Type: application/json
Transfer-Encoding: chunked
Cache-Control: no-cache, private
Date: Sun, 24 Jun 2018 10:19:27 GMT
Connection: keep-alive

{"error_code":403002,"message":"用户已禁用"}
```

应该在返回的错误信息中，同时包含面向开发者和面向用户的提示信息，前者可方便开发人员调试，后者可直接展示给终端用户查看，如：

```
{
    "message": "直接展示给终端用户的错误信息",
    "error_code": "业务错误码",
    "error": "供开发者查看的错误信息",
    "debug": [
        "错误堆栈，必须开启 debug 才存在"
```

```
        ]
    }
```

下面详细列举了各种情况 API 的返回说明。

- 200 ok

200状态码是最常见的HTTP状态码，在所有成功的GET请求中，必须返回此状态码。HTTP响应实体部分必须直接就是数据，不要做多余的包装。

错误示例：

```
HTTP/1.1 200 ok
Content-Type: application/json
Server: example.com

{
    "user": {
        "id":1,
        "nickname":"fwest",
        "username": "example"
    }
}
```

正确示例：

1、获取单个资源详情

```
{
    "id": 1,
    "username": "godruoyi",
    "age": 88,
```

```
}
```

2、获取资源集合

```
[
    {
        "id": 1,
        "username": "godruoyi",
        "age": 88,
    },
    {
        "id": 2,
        "username": "foo",
        "age": 88,
    }
]
```

3、额外的媒体信息

```
{
    "data": [
        {
            "id": 1,
            "avatar": "https://lorempixel.com/640/480/?32556",
            "nickname": "fwest",
            "last_logined_time": "2018-05-29 04:56:43",
            "has_registed": true
        },
        {
```

```
            "id": 2,
            "avatar": "https://lorempixel.com/640/480/?86144",
            "nickname": "zschowalter",
            "last_logined_time": "2018-06-16 15:18:34",
            "has_registed": true
        }
    ],
    "meta": {
        "pagination": {
            "total": 101,
            "count": 2,
            "per_page": 2,
            "current_page": 1,
            "total_pages": 51,
            "links": {
                "next": "http://api.example.com?page=2"
            }
        }
    }
}
```

其中，分页和其他额外的媒体信息，必须放到 `meta` 字段中。

- 201 Created

当服务器创建数据成功时，应该返回此状态码。常见的应用场景是使用POST提交用户信息，如：

- 添加了新用户

- 上传了图片

- 创建了新活动

等，都可以返回201状态码。需要注意的是，你可以选择在用户创建成功后返回新用户的数据

```
HTTP/1.1 201 Created
Server: nginx/1.11.9
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sun, 24 Jun 2018 09:13:40 GMT
Connection: keep-alive

{
    "id": 1,
    "avatar": "https:\/\/lorempixel.com\/640\/480\/?32556",
    "nickname": "fwest",
    "last_logined_time": "2018-05-29 04:56:43",
    "created_at": "2018-06-16 17:55:55",
    "updated_at": "2018-06-16 17:55:55"
}
```

也可以返回一个响应实体为空的HTTP Response，如：

```
HTTP/1.1 201 Created
Server: nginx/1.11.9
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
```

```
Date: Sun, 24 Jun 2018 09:12:20 GMT
Connection: keep-alive
```

这里我们应该采用第二种方式，因为大多数情况下，客户端只需要知道该请求操作成功与否，并不需要返回新资源的信息。

- 202 Accepted

该状态码表示服务器已经接受到了来自客户端的请求，但还未开始处理。常用短信发送、邮件通知、模板消息推送等这类很耗时需要队列支持的场景中；

返回该状态码时，响应实体必须为空。

```
HTTP/1.1 202 Accepted
Server: nginx/1.11.9
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Date: Sun, 24 Jun 2018 09:25:15 GMT
Connection: keep-alive
```

- 204 No Content

该状态码表示响应实体不包含任何数据，其中：

- 在使用DELETE方法删除资源成功时，必须返回该状态码
- 使用PUT、PATCH方法更新数据成功时，也应该返回此状态码

```
HTTP/1.1 204 No Content
Server: nginx/1.11.9
Date: Sun, 24 Jun 2018 09:29:12 GMT
```

```
Connection: keep-alive
```

■ 3xx 重定向

所有API不该返回3xx类型的状态码。因为3xx类型的响应格式一般为下列格式：

```
HTTP/1.1 302 Found
Server: nginx/1.11.9
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Cache-Control: no-cache, private
Date: Sun, 24 Jun 2018 09:41:50 GMT
Location: https://example.com
Connection: keep-alive

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8" />
        <meta http-equiv="refresh" content="0;url=https://example.com" />
        <title>Redirecting to https://example.com</title>
    </head>
    <body>
        Redirecting to <a href="https://example.com">https://example.com</a>.
    </body>
</html>
```

所有API一定不可返回纯HTML结构的响应；若一定要使用重定向功能，可以返回一个响应实体为空的3xx响应，并在响应头中加上Location字段:

```
HTTP/1.1 302 Found
Server: nginx/1.11.9
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Date: Sun, 24 Jun 2018 09:52:50 GMT
Location: https://godruoyi.com
Connection: keep-alive
```

- 400 Bad Request

由于明显的客户端错误（例如，请求语法格式错误、无效的请求、无效的签名等），服务器应该放弃该请求。

当服务器无法从其他 4xx 类型的状态码中找出合适的来表示错误类型时，都必须返回该状态码。

```
HTTP/1.1 400 Bad Request
Server: nginx/1.11.9
Content-Type: application/json
Transfer-Encoding: chunked
Cache-Control: no-cache, private
Date: Sun, 24 Jun 2018 13:22:36 GMT
Connection: keep-alive


{"error_code":40000,"message":"无效的签名"}
```

- 401 Unauthorized

该状态码表示当前请求需要身份认证，以下情况都必须返回该状态码。

- 未认证用户访问需要认证的 API
- access_token 无效/过期

客户端在收到401响应后，都应该提示用户进行下一步的登录操作。

```
HTTP/1.1 401 Unauthorized
Server: nginx/1.11.9
Content-Type: application/json
Transfer-Encoding: chunked
WWW-Authenticate: JWTAuth
Cache-Control: no-cache, private
Date: Sun, 24 Jun 2018 13:17:02 GMT
Connection: keep-alive

{"message":"Token Signature could not be verified.","error_code": "40100"}
```

■ 403 Forbidden

该状态码可以简单的理解为没有权限访问该请求，服务器收到请求但拒绝提供服务，如当普通用户请求操作管理员用户时，必须返回该状态码。

```
HTTP/1.1 403 Forbidden
Server: nginx/1.11.9
Content-Type: application/json
Transfer-Encoding: chunked
Cache-Control: no-cache, private
Date: Sun, 24 Jun 2018 13:05:34 GMT
Connection: keep-alive
```

```
{"error_code":40301,"message":"权限不足"}
```

- 404 Not Found

该状态码表示用户请求的资源不存在，如

- 获取不存在的用户信息 （get /users/9999999）
- 访问不存在的端点

都必须返回该状态码，若该资源已永久不存在，则应该返回410响应。

- 405 Method Not Allowed

当客户端使用的HTTP请求方法不被服务器允许时，必须返回该状态码，如客户端调用了`POST`方法来访问只支持 GET 方法的 API。

该响应必须返回一个Allow头信息用以表示出当前资源能够接受的请求方法的列表。

```
HTTP/1.1 405 Method Not Allowed
Server: nginx/1.11.9
Content-Type: application/json
Transfer-Encoding: chunked
Allow: GET, HEAD
Cache-Control: no-cache, private
Date: Sun, 24 Jun 2018 12:30:57 GMT
Connection: keep-alive

{"message":"405 Method Not Allowed","error_code": 40500}
```

- 406 Not Acceptable

API在不支持客户端指定的数据格式时，应该返回此状态码。如支持JSON和XML输出的API被指定返回YAML格式的数据时。

HTTP协议一般通过请求首部的 Accept 来指定数据格式

- 408 Request Timeout

客户端请求超时时必须返回该状态码，需要注意的时，该状态码表示客户端请求超时，在涉及第三方API调用超时时，一定不可返回该状态码。

- 409 Confilct

该状态码表示因为请求存在冲突无法处理。如通过手机号码提供注册功能的API，当用户提交的手机号已存在时，必须返回此状态码。

```
HTTP/1.1 409 Conflict
Server: nginx/1.11.9
Content-Type: application/json
Transfer-Encoding: chunked
Cache-Control: no-cache, private
Date: Sun, 24 Jun 2018 12:19:04 GMT
Connection: keep-alive


{"error_code":40900,"message":"手机号已存在"}
```

- 410 Gone

和404类似，该状态码也表示请求的资源不存在，只是410状态码进一步表示所请求的资源已不存在，并且未来也不会存在。在收到410状态码后，客户端应该停止再次请求该资源。

- 413 Request Entity Too Large

该状态码表示服务器拒绝处理当前请求，因为该请求提交的实体数据大小超过了服务器愿意或者能够处理的范围。

此种情况下，服务器可以关闭连接以免客户端继续发送此请求。

如果这个状况是临时的，服务器应该返回一个Retry-After的响应头，以告知客户端可以在多少时间以后重新尝试。

- 414 Request-URI Too Long

该状态码表示请求的URI长度超过了服务器能够解释的长度，因此服务器拒绝对该请求提供服务。

- 415 Unsupported Media Type

通常表示服务器不支持客户端请求首部Content-Type指定的数据格式。如在只接受 JSON格式的API中放入XML类型的数据并向服务器发送，都应该返回该状态码。

该状态码也可用于如：只允许上传图片格式的文件，但是客户端提交媒体文件非法或不是图片类型，这时应该返回该状态码：

```
HTTP/1.1 415 Unsupported Media Type
Server: nginx/1.11.9
Content-Type: application/json
Transfer-Encoding: chunked
Cache-Control: no-cache, private
Date: Sun, 24 Jun 2018 12:09:40 GMT
Connection: keep-alive
```

```
{"error_code":41500,"message":"不允许上传的图片格式"}
```

- 429 Too Many Requests

该状态码表示用户请求次数超过允许范围。如API设定为60次/分钟，当用户在一分钟内请求次数超过 60 次后，都应该返回该状态码。并且也应该在响应首部中加上下列头部：

```
X-RateLimit-Limit: 10 请求速率（由应用设定，其单位一般为小时/分钟等，这里是 10次/5分钟）
X-RateLimit-Remaining: 0 当前剩余的请求数量
X-RateLimit-Reset: 1529839462 重置时间
Retry-After: 120 下一次访问应该等待的时间（秒）
```

例如：

```
HTTP/1.1 429 Too Many Requests
Server: nginx/1.11.9
Content-Type: application/json
Transfer-Encoding: chunked
X-RateLimit-Limit: 10
X-RateLimit-Remaining: 0
X-RateLimit-Reset: 1529839462
Retry-After: 290
Cache-Control: no-cache, private
Date: Sun, 24 Jun 2018 11:19:32 GMT
Connection: keep-alive
```

```
{"message":"You have exceeded your rate limit.","error_code":42900}
```

必须为所有的 API 设置 Rate Limit 支持。

- 500 Internal Server Error

该状态码必须在服务器出错时抛出，对于所有的500错误，都应该提供完整的错误信息支持，也方便跟踪调试。

- 503 Service Unavailable

该状态码表示服务器暂时处理不可用状态，当服务器需要维护或第三方API请求超时/不可达时，都应该返回该状态码，其中若是主动关闭 API 服务，应该在返回的响应首部加上Retry-After头部，表示多少秒后可以再次访问。

```
HTTP/1.1 503 Service Unavailable
Server: nginx/1.11.9
Content-Type: application/json
Transfer-Encoding: chunked
Cache-Control: no-cache, private
Date: Sun, 24 Jun 2018 10:56:20 GMT
Retry-After: 60
Connection: keep-alive

{"error_code":50300,"message":"服务维护中"}
```

其他 `HTTP` 状态码请参考 [HTTP 状态码 - 维基百科](https://zh.wikipedia.org/zh-hans/HTTP状态码)。

3.9. 安全

以下是当你在设计, 测试以及发布你的 API 的时候所需要核对的重要安全措施

3.9.1. 身份认证

不要使用Basic Auth使用标准的认证协议 (如 JWT, OAuth).

加密所有的敏感数据。

- JWT (JSON Web Token)
  - 使用随机复杂的密钥 (JWT Secret) 以增加暴力破解的难度.
  - 不要在请求体中直接提取数据, 要对数据进行加密 (HS256或RS256).
  - 使 token 的过期时间尽量的短 (TTL, RTTL).
  - 不要在 JWT 的请求体中存放敏感数据, 它是[可破解的](https://jwt.io/#debugger-io).

- OAuth 授权或认证协议
  - 始终在后台验证redirect_uri, 只允许白名单的 URL.
  - 每次交换令牌的时候不要加 token (不允许response_type=token).
  - 使用state参数并填充随机的哈希数来防止跨站请求伪造(CSRF).
  - 对不同的应用分别定义默认的作用域和各自有效的作用域参数.

3.9.2. 访问

- 限制流量来防止 DDoS 攻击和暴力攻击.

- 在服务端使用 HTTPS 协议来防止 MITM 攻击.
- 使用 `HSTS` 协议防止 SSLStrip 攻击.

3.9.3. 输入

- 使用与操作相符的 HTTP 操作函数：GET (读取), POST (创建), PUT (替换/更新)以及 DELETE (删除记录), 如果请求的方法不适用于请求的资源则返回405 Method Not Allowed.
- 在请求头中的content-type字段使用内容验证来只允许支持的格式 (如application/xml, application/json等等) 并在不满足条件的时候返回406 Not Acceptable.
- 验证content-type的发布数据和你收到的一样，如：

  application/x-www-form-urlencoded,

  multipart/form-data,

  application/json

  等等
- 验证用户输入来避免一些普通的易受攻击缺陷，如XSS, SQL注入, 远程代码执行等等。
- 不要在 URL 中使用任何敏感的数据，如credentials, Passwords, security tokens, or API keys), 而是使用标准的认证请求头.
- 使用 API Gateway 服务来启用缓存、访问速率限制

4. 参考资料

能愿动词的定义与使用：[RFC 2119](http://www.ietf.org/rfc/rfc2119.txt)

restful-api-design-references：https://github.com/duanlg/restful-api-design-references

Principles of good RESTful API Design：

http://www.cnblogs.com/moonz-wu/p/4211626.html

HTTP 状态码- 维基百科：https://zh.wikipedia.org/zh-hans/HTTP状态码