HTTP API 设计指南

> 翻译自 `HTTP API Design Guide` https://github.com/interagent/http-api-design

前言

这篇指南介绍了大量的 http + json 的api设计风格，最初摘录自heroku平台的api设计指引 Heroku 平台 API 指引.

这篇指南除了当前那些API，同时也适用于heroku平台新集成的api,我们希望那些在heroku之外的api设计者也感兴趣!

我们的目标是一致性，专注业务逻辑同时避免设计上的空想，我们一直在寻找一种良好的，统一的，显而易见的API设计方式，未必只有一种方式。

我们假设你熟悉基本的 http+json api 设计方式，但是，不一定包含在此篇指南里面的所有内容。

内容

返回合适的状态码

为每一次的响应返回合适的HTTP状态码. 成功的HTTP响应应该使用如下的状态码:

- `200`：`GET` 请求成功, 以及 `DELETE` 或 `PATCH` 同步请求完成
- `201`：`POST` 同步请求完成
- `202`：`POST`，`DELETE`，或 `PATCH` 异步请求将要完成
- `206`：`GET` 请求成功, 这里只是一部分状态码: 更多

对于用户请求的错误情况，及服务器的异常错误情况，请查阅完整的HTTP状态码HTTP response code spec。

提供全部可用的资源

提供全部可显现的资源 (例如. 这个对象的所有属性) 不用考虑全部可能响应的状态码，总是在响应码为200或是201时返回所有可用资源, 包含 `PUT` / `PATCH` 和 `DELETE` 请求, 例如:

```
$ curl -X DELETE \
  https://service.com/apps/1f9b/domains/0fd4

HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8
...
{
  "created_at": "2012-01-01T12:00:00Z",
  "hostname": "subdomain.example.com",
  "id": "01234567-89ab-cdef-0123-456789abcdef",
  "updated_at": "2012-01-01T12:00:00Z"
}
```

当请求状态码为202时，不返回所有可用资源, e.g.:

```
$ curl -X DELETE \
  https://service.com/apps/1f9b/dynos/05bd

HTTP/1.1 202 Accepted
Content-Type: application/json;charset=utf-8
...
{}
```

在请求的body体使用JSON数据

在 `PUT` / `PATCH` / `POST` 请求的body体使用JSON格式数据, 而不是使用 form 表单形式的数据. 这里我们使用JSON格式的body请求创建对称的格式数据, 例如.:

```
$ curl -X POST https://service.com/apps \
    -H "Content-Type: application/json" \
    -d '{"name": "demoapp"}'

{
  "id": "01234567-89ab-cdef-0123-456789abcdef",
  "name": "demoapp",
  "owner": {
    "email": "username@example.com",
    "id": "01234567-89ab-cdef-0123-456789abcdef"
  },
  ...
}
```

提供资源的唯一标识

在默认情况给每一个资源一个 `id` 属性. 用此作为唯一标识除非你有更好的理由不用.不要使用那种在服务器上或是资源中不是全局唯一的标识，比如自动增长的id标识。

返回的唯一标识要用小写字母并加个分割线格式 `8-4-4-4-12`, 例如.:

```
"id": "01234567-89ab-cdef-0123-456789abcdef"
```

提供标准的时间戳

提供默认的资源创建时间，更新时间 `created_at` and `updated_at` , e例如:

`json { ... "created_at": "2012-01-01T12:00:00Z", "updated_at": "2012-01-01T13:00:00Z", ... }`

这些时间戳可能不适用于某些资源，这种情况下可以忽略省去。

使用ISO8601的国际化时间格式

在接收的返回时间数据时只使用UTC格式. 查阅ISO8601时间格式, 例如:

```
"finished_at": "2012-01-01T12:00:00Z"
```

使用统一的资源路径

资源命名

使用复制形式为资源命名，而不是在系统中有争议的一个单个资源 (例如，在大多数系统中，给定的用户帐户只有一个). 这种方式保持了特定资源的统一性.

形为

好的末尾展现形式不许要指定特殊的资源形为，在某些情况下，指定特殊的资源的形为是必须的,用一个标准的 `actions` 前缀去替代他, 清楚的描述他:

```
/resources/:resource/actions/:action
```

例如.

```
/runs/{run_id}/actions/stop
```

路径和属性要用小写字母

使用小写字母并用 `-` 短线分割路径名字,并且紧跟着主机域名 e.g:

```
service-api.com/users
service-api.com/app-setups
```

同样属性也要用小写字母, 但是属性名字要用下划线 `_` 分割，以至于这样在javaScript语言中不用输入引号。 例如.:

```
service_class: "first"
```

嵌套外键关系

序列化的外键关系通常建立在一个有嵌套关系的对象之上, 例如.:

`json { "name": "service-production", "owner": { "id": "5d8201b0..." }, ... }`

而不是这样 例如:

`json { "name": "service-production", "owner_id": "5d8201b0...", ... }`

这种方式尽可能的把相关联的资源信息内联在一起，而不用改变响应资源的结构,或者展示更高一级的响应区域, 例如:

`json { "name": "service-production", "owner": { "id": "5d8201b0...", "name": "Alice", "email": "alice@`

支持方便的无id间接引用

在某些情况下，为了方便用户使用接口，在末尾提供用id标识资源,例如，一个用户想到了他在heroku平台app的名字，但是这个app的唯一标识是id,这种情况下，你想让接口通过名字和id都能访问，例如:

```
$ curl https://service.com/apps/{app_id_or_name}
$ curl https://service.com/apps/97addcf0-c182
$ curl https://service.com/apps/www-prod
```

不要只接受使用名字而剔除了使用id。

构建错误信息

在网络请求响应错误的时候，返回统一的，结构化的错误信息。要包含一个机器可读的错误 `id` ,一个人类能识别的错误信息 `message` , 根据情况可以添加一个 `url` ，告诉客户端关于这个错误的更多信息以及如何去解决它。 例如:

```
HTTP/1.1 429 Too Many Requests
```

`json { "id": "rate_limit", "message": "Account reached its API rate limit.", "url": "https://docs.serv`

把你的错误信息格式文档化，以及这些可能的错误信息 `id` s 让客户端能获取到.

支持Etags缓存

在所有请求响应中包含一个 `ETag` 头,比如，标识出返回资源的指定版本. 用户能够根据他们提供的头信息 `If-None-Match` ，在随后的请求中检查出那些无效的。

用id来跟踪每次的请求

在每一个API响应中要包含一个 `Request-Id` 头信息, 通常用唯一标识UUID. 如果服务器和客户端都打印出他们的 `Request-Id` , 这对我们的网络请求调试和跟踪非常有帮助.

按范围分页

对于服务器响应的大量数据我们应该为此分页。 使用 `Content-Range` 头传递分页请求的数据.这里有个例子详细的说明了请求和响应头、状态码，限制条件、排序以及分页处理：Heroku Platform API on Ranges.

显示速度限制状态

客户端的访问速度限制可以维护服务器的良好状态，进而为其他客户端请求提供高性的服务.你可以使用token bucket algorithm技术量化请求限制.

为每一个带有 `RateLimit-Remaining` 响应头的请求，返回预留的请求tokens。

指定可接受头信息的版本

在开始的时候指定API版本，使用 `Accepts` 头传递版本信息,也可以是一个自定义的内容, 例如:

```
Accept: application/vnd.heroku+json; version=3
```

最好不要给出一个默认的版本, 而是要求客户端明确指明他们要使用特定的版本.

减少路径嵌套

在一些有父路径/子路径嵌套关系的资源数据模块中, 路径可能有非常深的嵌套关系, 例如:

```
/orgs/{org_id}/apps/{app_id}/dynos/{dyno_id}
```

推荐在根(root)路径下指定资源来限制路径的嵌套深度。使用嵌套指定范围的资源，例如在下面的情况下，dyno属性app范围下，app属性org范围下:

```
/orgs/{org_id}
/orgs/{org_id}/apps
/apps/{app_id}
/apps/{app_id}/dynos
/dynos/{dyno_id}
```

提供机器可读的JSON概要

提供一个机器可读的概要恰当的表现你的API.使用 prmd管理你的概要, 并且确保用 `prmd verify` 验证是有效的。

提供人类可读的文档

提供人类可读的文档让客户端开发人员可以理解你的API。

如果你用prmd创建了一个概要并且按上述要求描述，你可以很容易的生成所有结节使用 `prmd doc` 的Markdown文件。

除此之在详细信息的结尾，提供一个关于如下信息的API摘要:

- 验证授权,包含获取及使用验证tokens.
- API 稳定性及版本控制, 包含如何选择所需要的版本.
- 一般的请求和响应头信息.
- 错误信息序列格式.

- 不同语言客户端使用API的例子.

提供可执行的示例

提供可执行的示例让用户可以直接在终端里面看到API的调用情况,最大程度的让这些示例可以逐字的使用，以减少用户尝试使用API的工作量。例如:

```
$ export TOKEN=... # acquire from dashboard
$ curl -is https://$TOKEN@service.com/users
```

如果你使用prmd生成Markdown文档, 你会自动获取一些示例在结点处。

描述稳定性

描述您的API的稳定性或是它在各种各样节点环境中的完备性和稳定性 例如. 带有 原型/开发版/产品 等标签.

更多关于可能的稳定性和改变管理的方式，查看 Heroku API compatibility policy

一旦你的API宣布产品正式版本及稳定版本时, 不要在当前API版本回退做一些不兼容的改变。如果你需要回退做一些不兼容的改变,请创建一个新的版本的API。

要求传输通道安全

要求在传输通道安全的情况下访问API,这点没什么可解释的了. 因为这不值得去争辩在什么时候使用传输通道安全访问API是好的，什么时候是不好的，只需要记住在访问API时，使用安全传输通道就行了。

使用友好的JSON输出

用户在第一次看到你的API使用情况，很可能是在命令行下使用 `curl` 进行的。友好的输出会让他们非常容易的理解你的API，为好开发者们的方便,打印友好的JSON输出, 例如:

```
json { "beta": false, "email": "alice@heroku.com", "id": "01234567-89ab-cdef-0123-456789abcdef", "last
```

而不是这样 例如:

```
json {"beta":false,"email":"alice@heroku.com","id":"01234567-89ab-cdef-0123-456789abcdef","last_login"
```

记住要在每行尾部包含一个换行，这样那些使用终端测试API的输出不会被遮挡住。

对于大多数的API友好的响应数据输出总会有更好的表现，你可能受到过不友好打印数据API的影响(例如:非常高的流量占用) 或者是在一些客户端上不能工作 (例如：一个随意编写的程序).

译者注

- 更新时间： `2014-06-11`
- 此为本人第一篇翻译文档，翻译不好的地方，还望读者见谅.
- 如果您觉得此文档对您帮助非常大，可以慰劳一下译者，支付宝