



HTTPAPI设计指南

极客学院出版

前言

这篇指南介绍描述了 HTTP+JSON API 的一种设计模式，最初摘录整理自 Heroku 平台的 API 设计指引 [Heroku 平台 API 指引 \(https://devcenter.heroku.com/articles/platform-api-reference\)](https://devcenter.heroku.com/articles/platform-api-reference)。

这篇指南除了详细介绍现有的 API 外，Heroku 将来新加入的内部 API 也会符合这种设计模式，我们希望非 Heroku 员工的 API 设计者也能感兴趣。

我们的目标是保持一致性，专注业务逻辑同时避免过度设计。我们一直试图找出一种良好的、一致的、显而易见的 API 设计方法，而并不是所谓的"最终/理想模式"。

学习前提

我们假设你熟悉基本的 HTTP+JSON API 设计方法，所以本篇指南并不包含所有的 API 设计基础。

鸣谢： [http-api-design \(http://github.com/ZhangBohan/http-api-design-ZH_CN\)](http://github.com/ZhangBohan/http-api-design-ZH_CN)

目录

前言	1
第 1 章 基础	4
隔离关注点	5
强制使用安全连接（ Secure Connections ）	6
强制头信息 Accept 中提供版本号	7
支持 Etag 缓存	8
为内省而提供 Request-Id	9
通过请求中的范围（ Range ）拆分大的响应	10
第 2 章 请求（ Requests ）	11
返回合适的状态码	12
提供全部可用的资源	13
在请求的 body 体使用 JSON 格式数据	14
使用统一的资源路径格式	15
路径和属性要小写	16
支持方便的无 id 间接引用	17
最小化路径嵌套	18
第 3 章 响应（ Responses ）	19
提供资源的(UU)ID	20
提供标准的时间戳	21
使用 UTC（世界标准时间）时间，用 ISO8601 进行格式化	22
嵌套外键关系	23
生成结构化的错误	24
显示频率限制状态	25
保证响应 JSON 最小化	26

第 4 章	工件 (Artifacts)	27
	可读性	28
	提供可执行的例子	29
	描述稳定性	30
	良好打印的默认 JSON	31



T



基础



隔离关注点

设计时通过将请求和响应之间的不同部分隔离来让事情变得简单。保持简单的规则让我们能更关注在一些更大的更困难的问题上。

请求和响应将解决一个特定的资源或集合。使用路径（path）来表明身份，body 来传输内容（content）还有头信息（header）来传递元数据（metadata）。查询参数同样可以用来传递头信息的内容，但头信息是首选，因为他们更灵活、更能传达不同的信息。

强制使用安全连接（Secure Connections）

所有的访问 API 行为，都需要用 TLS 通过安全连接来访问。没有必要搞清或解释什么情况需要 TLS 什么情况不需要 TLS，直接强制任何访问都要通过 TLS。

理想状态下，通过拒绝所有非 TLS 请求，不响应 http 或 80 端口的请求以避免任何不安全的数据交换。如果现实情况中无法这样做，可以返回 403 Forbidden 响应。

把非 TLS 的请求重定向(Redirect)至 TLS 连接是不明智的，这种含混/不好的客户端行为不会带来明显好处。依赖于重定向的客户端访问不仅会导致双倍的服务器负载，还会使 TLS 加密失去意义，因为在首次非 TLS 调用时，敏感信息就已经暴露出去了。

强制头信息 Accept 中提供版本号

制定版本并在版本之间平缓过渡对于设计和维护一套 API 是个巨大的挑战。所以，最好在设计之初就使用一些方法来预防可能会遇到的问题。

为了避免 API 的变动导致用户使用中产生意外结果或调用失败，最好强制要求所有访问都需要指定版本号。请避免提供默认版本号，一旦提供，日后想要修改它会相当困难。

最适合放置版本号的位置是头信息(HTTP Headers)，在 `Accept` 段中使用自定义类型(content type)与其他元数据(metadata)一起提交。例如：

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


支持 Etag 缓存

, identifying the specific version of the returned resource. The user should be able to check for staleness in their subsequent requests by supplying the value in the If-None-Match header.

在所有的请求中带上 ETag 头，用于识别特定版本的返回资源。用户可以在随后的请求中通过提供 If-None-Match 头的值来检查内容是否过期。

为内省而提供 Request-Id

为每一个请求响应包含一个 `Request-Id` 字段，并使用 UUID 作为该值。通过在客户端、服务器或任何支持服务上记录该值，它能主我们提供一种机制来跟踪、诊断和调试请求。

通过请求中的范围（Range）拆分大的响应

一个大的响应应该通过多个请求使用 Range 头信息来拆分，并指定如何取得。详细的请求和响应的头信息（header），状态码(status code)，范围(limit)，排序(ordering)和迭代(iteration)等，参考[Heroku Platform API discussion of Ranges \(https://devcenter.heroku.com/articles/platform-api-reference#ranges\)](https://devcenter.heroku.com/articles/platform-api-reference#ranges)。



请求 (Requests)



返回合适的状态码

为每一次的响应返回合适的HTTP状态码。好的响应应该使用如下的状态码:

- 200 : GET 请求成功, 及 DELETE 或 PATCH 同步请求完成, 或者 PUT 同步更新一个已存在的资源
- 201 : POST 同步请求完成, 或者 PUT 同步创建一个新的资源
- 202 : POST , PUT , DELETE , 或 PATCH 请求接收, 将被异步处理
- 206 : GET 请求成功, 但是只返回一部分

使用身份认证 (authentication) 和授权 (authorization) 错误码时需要注意:

- 401 Unauthorized : 用户未认证, 请求失败
- 403 Forbidden : 用户无权限访问该资源, 请求失败

当用户请求错误时, 提供合适的状态码可以提供额外的信息:

- 422 Unprocessable Entity : 请求被服务器正确解析, 但是包含无效字段
- 429 Too Many Requests : 因为访问频繁, 你已经被限制访问, 稍后重试
- 500 Internal Server Error : 服务器错误, 确认状态并报告问题

对于用户错误和服务器错误情况状态码, 参考: [HTTP response code spec \(https://tools.ietf.org/html/rfc7231#section-6\)](https://tools.ietf.org/html/rfc7231#section-6)

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), 当响应码为 200 或是 201 时返回所有可用资源, 包含 **P** **UT** / **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, 例如:

```
$ 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 格式数据

在 PUT / PATCH / POST 请求的正文 (request bodies) 中使用JSON格式数据，而不是使用 form 表单形式的数据。这与我们使用JSON格式返回请求相对应，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"
  },
  ...
}
```

使用统一的资源路径格式

资源名称

使用复数来命名资源，除非该资源在系统中是单件（比如，在绝大多数系统中，一个用户只能拥有一个账户）。这样在你引用特定资源时可以保持一致性。

对独有的资源使用不需要特定动作的 endpoint 格式。这样当需要特定的动作，只需要把它们放到标准的 actions 前缀后边，就可以清晰的描述它们：

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

如：

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


路径和属性要小写

为了和域名命名规则保持一致，使用小写字母并用 `-` 分割路径名字，例如：

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

属性也使用小写字母，但是属性名要用下划线 `_` 分割，以便在 Javascript 中省略引号。例如：

```
service_class: "first"
```

支持方便的无 id 间接引用

在某些情况下，让用户提供 ID 去定位资源是不方便的。例如，一个用户想取得他在 Heroku 平台 app 信息，但是这个 app 的唯一标识是 UUID。这种情况下，你应该支持接口通过名字和 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放弃了使用 id。

最小化路径嵌套

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。在上述例子中，dyno 属于 app，app 属于 org 可以表示为：

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



3

响应 (Responses)



提供资源的(UU)ID

在默认情况给每一个资源一个 `id` 属性。除非有更好的理由，否则请使用 UUID。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，尤其是自动增长的 `id`。

生成小写的UUID格式 `8-4-4-4-12`，例如：

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

提供标准的时间戳

为资源提供默认的创建时间 `created_at` 和更新时间 `updated_at`，例如：

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

有些资源不需要使用时间戳那么就忽略这两个字段。

使用 UTC (世界标准时间) 时间, 用 ISO8601 进行格式化

在接收和返回时都只使用 UTC 格式。ISO8601 格式的数据, 例如:

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

嵌套外键关系

使用嵌套对象序列化外键关联，例如：

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

而不是像这样：

```
{
  "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，而不用改变资源的结构，或者引入更多的字段，例如：

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


生成结构化的错误

响应错误的时，生成统一的、结构化的错误信息。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` 来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
```

```
{
  "id": "rate_limit",
  "message": "Account reached its API rate limit.",
  "url": "https://docs.service.com/rate-limits"
}
```

文档化客户端可能遇到的错误信息格式，以及这些可能的错误信息 `id`。

显示频率限制状态

客户端的访问速度限制可以维护服务器的良好状态，保证为其他客户端请求提供高性的服务。你可以使用[token bucket algorithm](http://en.wikipedia.org/wiki/Token_bucket_algorithm) ([http://en.wikipedia.org/wiki/Token_bucket](http://en.wikipedia.org/wiki/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。

保证响应 JSON 最小化

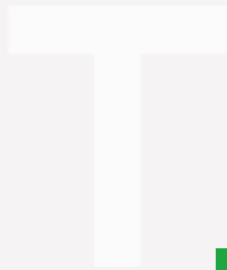
请求中多余的空格会增加响应大小，而且现在很多的 HTTP 客户端都会自己输出可读格式 ("pretty") 的 JSON。所以最好保证响应 JSON 最小化，例如：

```
{"beta":false,"email":"alice@heroku.com","id":"01234567-89ab-cdef-0123-456789abcdef","last_login":"2012-01-01T12:00:00Z"}
```

而不是这样：

```
{
  "beta": false,
  "email": "alice@heroku.com",
  "id": "01234567-89ab-cdef-0123-456789abcdef",
  "last_login": "2012-01-01T12:00:00Z",
  "created_at": "2012-01-01T12:00:00Z",
  "updated_at": "2012-01-01T12:00:00Z"
}
```

你可以提供可选的方式为客户端提供更详细可读的响应，使用查询参数（例如： `?pretty=true` ）或者通过 `Accept` 头信息参数（例如： `Accept: application/vnd.heroku+json; version=3; indent=4;` ）。



工件 (Artifacts)



可读性

机器可读的 JSON 模式

提供一个机器可读的模式可以精确的指定你的 API。使用 prmd 来管理你的模式，并确保它能被 prmd verify 验证。

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，你可以为所有节点很容易的使用 `prmd doc` 生成 Markdown 文档。

除了节点信息，提供一个 API 概述信息：

- 验证授权，包含如何取得和如何使用 token。
- API 稳定及版本管理，包含如何选择所需要的版本。
- 一般情况下的请求和响应的头信息。
- 错误的序列化格式。
- 不同编程语言客户端使用 API 的例子。

提供可执行的例子

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简单的使用，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](https://github.com/interagent/prmd) (<https://github.com/interagent/prmd>) 生成 Markdown 文档，每个节点都会自动获取一些示例。

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：加上 原型版 (prototype) / 开发版 (development) / 产品版 (production) 等标记。

更多关于可能的稳定性和改变管理的方式，查看 [Heroku API compatibility policy \(https://devcenter.heroku.com/articles/api-compatibility-policy\)](https://devcenter.heroku.com/articles/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 版本中做一些不兼容的改变。如果你需要，请创建一个新的版本的 API。

良好打印的默认 JSON

用户第一次查看你的 API 很可能是在使用 curl 的命令行里。如果 API 的响应有良好的打印格式，那在命令行里它们会很容易理解。为了给这些开发者提供方便，良好打印格式的 JSON 如下：

```
{
  "beta": false,
  "email": "alice@heroku.com",
  "id": "01234567-89ab-cdef-0123-456789abcdef",
  "last_login": "2012-01-01T12:00:00Z",
  "created_at": "2012-01-01T12:00:00Z",
  "updated_at": "2012-01-01T12:00:00Z"}
```

而不是：

```
{"beta":false,"email":"alice@heroku.com","id":"01234567-89ab-cdef-0123-456789abcdef","last_login":"2012-01-01T12:00:00Z","created_at":"2012-01-01T12:00:00Z","updated_at":"2012-01-01T12:00:00Z"}
```

要确保在 JSON 结尾有换行，以防止阻塞用户的终端界面。

对于大部分 API 的响应，性能考虑要优先于良好打印。在某些结点（例如高流量结点）或为某些特定用户（例如无 GUI 界面的程序）使用时，你可能会考虑使用高性能而非良好打印的 API。

注：headless program 译为“无显示界面的程序”，参考自这篇文章。

极客学院

jikexueyuan.com

中国最大的IT职业在线教育平台



更多信息请访问 

<http://wiki.jikexueyuan.com/project/http-api/>