RESTful API 设计规范

从字面可以看出，他是Rest式的接口，所以我们先了解下什么是Rest

什么是 REST API?

REST API 也称为 RESTful API，是遵循 REST 架构规范的应用编程接口（API 或 Web API），支持与 RESTful Web 服务进行交互。REST全称 **RE**presentational **S**tate **T**ransfer。 由Roy Thomas Fielding博士在2000年于其论文 *Architectural Styles and the Design of Network-based Software Architectures* 中提出的。 是一种分布式超媒体架构风格。

> ### 什么是 API?
>
> API 由一组定义和协议组合而成，可用于构建和集成应用软件。有时我们可以把它们当做信息提供者和信息用户之间的合同——建立消费者（呼叫）所需的内容和制作者（响应）要求的内容。例如，天气服务的 API 可指定用户提供邮编，制作者回复的答案由两部分组成，第一部分是最高温度，第二部分是最低温度。
>
> 换言之，如果你想与计算机或系统交互以检索信息或执行某项功能，API 可帮助你将你需要的信息传达给该系统，使其能够理解并满足你的请求。
>
> 可以把 API 看做是用户或客户端与他们想要的资源或 Web 服务之间的传递者。它也是企业在共享资源和信息的同时保障安全、控制和身份验证的一种方式，即确定哪些人可以访问什么内容。
>
> API 的另一个优势是你无需了解缓存的具体信息，即如何检索资源或资源来自哪里。

> ### 如何理解 REST 的含义?
>
> **REST 是一组架构规范，并非协议或标准。API 开发人员可以采用各种方式实施 REST。**
>
> 当客户端通过 RESTful API 提出请求时，它会将资源状态表述传递给请求者或终端。该信息或表述通过 HTTP 以下列某种格式传输：JSON（Javascript 对象表示法）、HTML、XLT、Python、PHP 或纯文本。JSON 是最常用的编程语言，尽管它的名字英文原意为“JavaScript 对象表示法”，但它适用于各种语言，并且人和机器都能读。
>
> **头和参数在 RESTful API HTTP 请求的 HTTP 方法中也很重要，因为其中包含了请求的元数据、授权、统一资源标识符（URI）、缓存、cookie 等重要标识信息。有请求头和响应头，每个头都有自己的 HTTP 连接信息和状态码。**

如何实现 RESTful API?

API 要被视为 RESTful API，必须遵循以下标准：

- **客户端-服务器架构由客户端、服务器和资源组成，并且通过 HTTP 管理请求。**
- 无状态**客户端-服务器通信，即 get 请求间隔期间，不会存储任何客户端信息，并且每个请求都是独立的，互不关联。**客户端到服务端的所有请求必须包含了所有信息，不能够利用任何服务器存储的上下文。 这一约束可以保证绘画状态完全由客户端控制

  这一点在你写一个接口的时候需要独立思考一下，如果每个请求都是独立的，互不关联的，那么他们怎么配合着实现一整套的功能，
- **可缓存性数据**：可简化客户端-服务器交互。
- **组件间的统一接口：使信息以标准形式传输。这要求：**
  - Identification of resources 资源标识符**所请求的资源可识别并与发送给客户端的表述分离开。**

- Manipulation of resources through representations

  通过“representation”来操作资源

- Self-descriptive messages 自我描述

  客户端可通过接收的表述操作资源，因为表述包含操作所需的充足信息。返回给客户端的自描述消息包含充足的信息，能够指明客户端应该如何处理所收到的信息。

- 超文本/超媒体可用，是指在访问资源后，客户端应能够使用超链接查找其当前可采取的所有其他操作。

- **组织各种类型服务器（负责安全性、负载平衡等的服务器）的分层系统会参与将请求的信息检索到对客户端不可见的层次结构中**。

  系统是分层的，客户端无法知道也不需要知道与他交互的是否是真正的终端服务器。 这也就给了系统在中间切入的可能，提高了安全性和伸缩性。

Resource 资源

在了解了REST API的约束后，REST最关键的概念就是资源。 任何的信息在REST架构里都被抽象为资源：图像、文档、集合、用户，等等。 （这在某些场景是和直觉相悖的，后文会详述） REST通过资源标识符来和特定资源进行交互。

资源在特定时间戳下的状态称之为资源表示(Resource Representation)，由**数据**，**元数据**和**超链接**组成。 资源的格式由媒体类型(media type)指定。(我们熟悉的JSON即是一种方式)

一个真正的REST API看上去就像是超文本一样。 除了数据本身以外还包含了其他客户端想了解的信息以描述自己， 比如一个典型的例子是在获取分页数据时，服务端同时还会返回页码总数以及下一页的链接。

REST vs HTTP

从上面的概念我们就可以知道，REST和任何具体技术无关。 我们会认为REST就是HTTP，主要是因为HTTP是最广为流行的客户端服务端通信协议。 但是HTTP本身和REST无关，你可以通过其他协议构建RESTful服务； 你用HTTP构建的服务也很有可能不是RESTful的。

REST vs JSON

与通信协议一样，REST与任何具体的数据格式无关。 无论你用XML，JSON或是HTML，都可以构建REST服务。

更进一步的，JSON甚至不是一种超媒体格式，只是一种数据格式。 比如JSON并没有定义超链接发现的行为。 真正的REST需要的是有着清楚规范的超媒体格式，比较标准的JSON-base超媒体格式有 JSON-LD 和 HAL

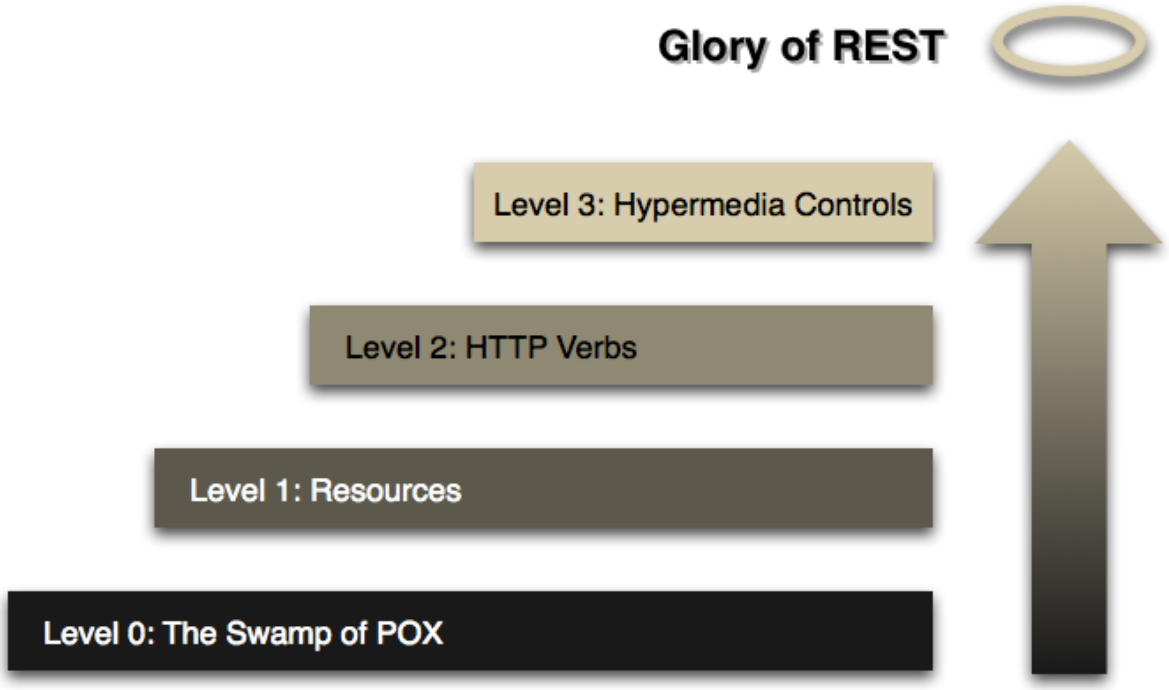
个人最想分享的部分！！！

Richardson Maturity Model

> steps toward the glory of REST
>
> *A model (developed by Leonard Richardson) that breaks down the principal elements of a REST approach into three steps. These introduce resources, http verbs, and hypermedia controls.*
>
> *一个模型（由 Leonard Richardson 开发）将 REST 方法的主要元素分解为三个步骤。这些介绍了资源、http 动词和超媒体控件。*

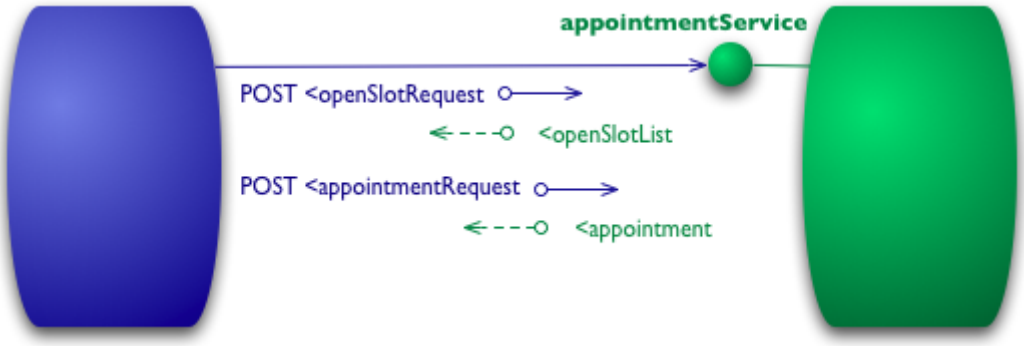
核心是这样一个概念，即网络是一个运行良好的大规模可扩展分布式系统的存在证明，我们可以从中汲取灵感来更轻松地构建集成系统。

走向 REST 的步骤



级别 0

该模型的出发点是使用 HTTP 作为远程交互的传输系统，但不使用任何 Web 机制。本质上，在这里所做的是使用 HTTP 作为自己的远程交互机制的隧道机制，通常基于Remote Procedure Invocation。



0 级交互示例

假设我想和我的医生预约。我的预约软件首先需要知道我的医生在给定日期有哪些空档，因此它会向医院预约系统发出请求以获取该信息。在 0 级场景中，医院将在某个 URI 处公开服务端点。然后，我将包含我的请求详细信息的文档发布到该端点。

```
POST /appointmentService HTTP/1.1
[various headers]

<openSlotRequest date = "2010-01-04" doctor = "mjones"/>
然后服务器将返回一个文件给我这个信息
HTTP/1.1 200 OK
[various headers]

<openSlotList>
  <slot start = "1400" end = "1450">
    <doctor id = "mjones"/>
```

```
  </slot>
  <slot start = "1600" end = "1650" >
    <医生 id = "mjones"/>
  </slot>
</openSlotList>
```

我在这里使用 XML 作为示例，但内容实际上可以是任何内容：JSON、YAML、键值对或任何自定义格式。

我的下一步是预约，我可以再次通过将文档发布到端点来进行预约。

```
POST /appointmentService HTTP/1.1
[various headers]

<appointmentRequest>
  <slot doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
</appointmentRequest>
```

如果一切顺利，我会收到回复说我的约会已预订。

```
HTTP/1.1 200 OK
[various headers]

<appointment>
  <slot doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
</appointment>
```

如果有问题，说其他人在我之前进入，那么我会在回复正文中收到某种错误消息。

```
HTTP/1.1 200 OK
[various headers]

<appointmentRequestFailure>
  <slot doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
  <reason>Slot not available</reason>
</appointmentRequestFailure>
```

到目前为止，这是一个简单的 RPC 样式系统。这很简单，因为它只是来回传输普通的旧 XML (POX)。如果您使用 SOAP 或 XML-RPC，它基本上是相同的机制，唯一的区别是您将 XML 消息包装在某种信封中。

级别 1 - 资源

在 RMM 中实现REST的荣耀的第一步是引入资源。因此，现在我们不再向单个服务端点发出所有请求，而是开始与单个资源进行对话。

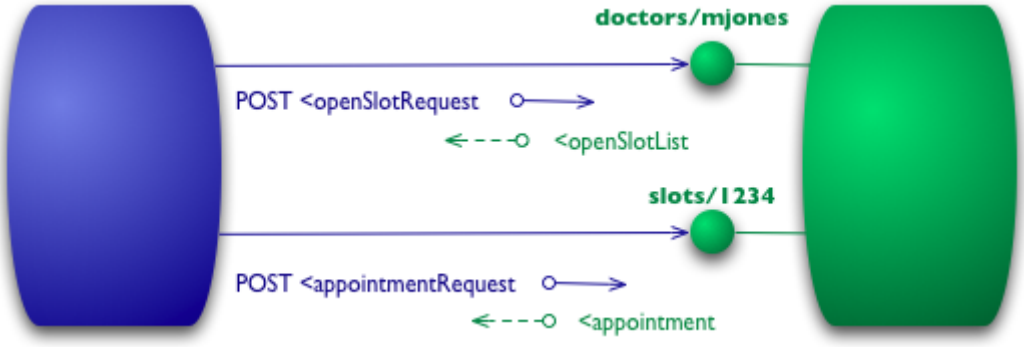


图 3：1 级添加资源

因此，对于我们的初始查询，我们可能有给定医生的资源。

```
POST /doctors/mjones HTTP/1.1
[various headers]

<openSlotRequest date = "2010-01-04"/>
```
回复带有相同的基本信息，但现在每个插槽都是可以单独寻址的资源。

```
HTTP/1.1 200 OK
[various headers]


<openSlotList>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <slot id = "5678" doctor = "mjones" start = "1600" end = "1650"/>
</openSlotList>
```
使用特定资源预约意味着发布到特定位置。

```
POST /slots/1234 HTTP/1.1
[各种其他标头]

<appointmentRequest>
  <患者 id = "jsmith"/>
</appointmentRequest>
```
如果一切顺利，我会收到与之前类似的回复。

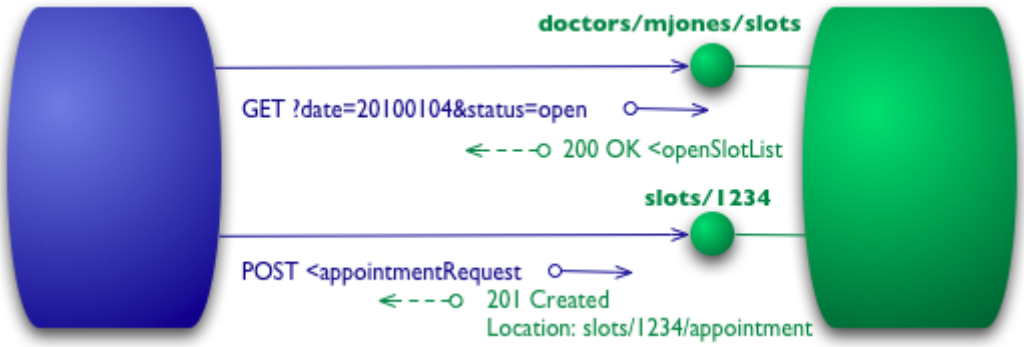
```
HTTP/1.1 200 OK
[various headers]

<appointment>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
</appointment>
```

区别是我们不是调用某个函数并传递参数，而是在一个特定对象上调用一个方法，为其他信息提供参数。

第 2 级 - HTTP 动词

在 0 级和 1 级的所有交互中都使用了 HTTP POST 动词，但有些人使用 GET 代替或附加使用。在这些级别上并没有太大区别，它们都被用作隧道机制，允许我们通过 HTTP 隧道交互。级别 2 远离这一点，使用 HTTP 动词尽可能接近它们在 HTTP 本身中的使用方式



对于我们的插槽列表，这意味着我们要使用 GET。

```
GET /doctors/mjones/slots?date=20100104&status=open HTTP/1.1
主机: royalhope.nhs.uk
```

回复与 POST 的回复相同

```
HTTP/1.1 200 OK
[various headers]

<openSlotList>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <slot id = "5678" doctor = "mjones" start = "1600" end = "1650"/>
</openSlotList>
```

在第 2 级，对这样的请求使用 GET 至关重要。HTTP 将 GET 定义为一种安全操作，即它不会对任何事物的状态进行任何重大更改。这允许我们以任何顺序安全地调用 GET 多次，并且每次都获得相同的结果。这样做的一个重要结果是，**它允许请求路由中的任何参与者使用缓存**，这是使 Web 性能与它一样好的关键因素。HTTP 包括各种支持缓存的措施，通信中的所有参与者都可以使用这些措施。通过遵循 HTTP 的规则，我们能够利用该功能。

为了预约，我们需要一个改变状态的 HTTP 动词，一个 POST 或一个 PUT。我将使用与之前相同的 POST。

即使我使用与级别 1 相同的帖子，远程服务的响应方式也存在另一个显着差异。如果一切顺利，该服务会回复一个响应代码 201，表示世界上有一个新资源。

```
HTTP/1.1 201 Created
Location: slots/1234/appointment
[various headers]

<appointment>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
</appointment>
```

201 响应包含一个带有 URI 的 location 属性，客户端可以使用该 URI 来获取该资源的当前状态。此处的响应还包括该资源的表示，以立即为客户端节省额外的调用。

如果出现问题，例如其他人预订会话，则还有另一个区别。

```
HTTP/1.1 409 Conflict
[various headers]

<openSlotList>
  <slot id = "5678" doctor = "mjones" start = "1600" end = "1650"/>
</openSlotList>
```

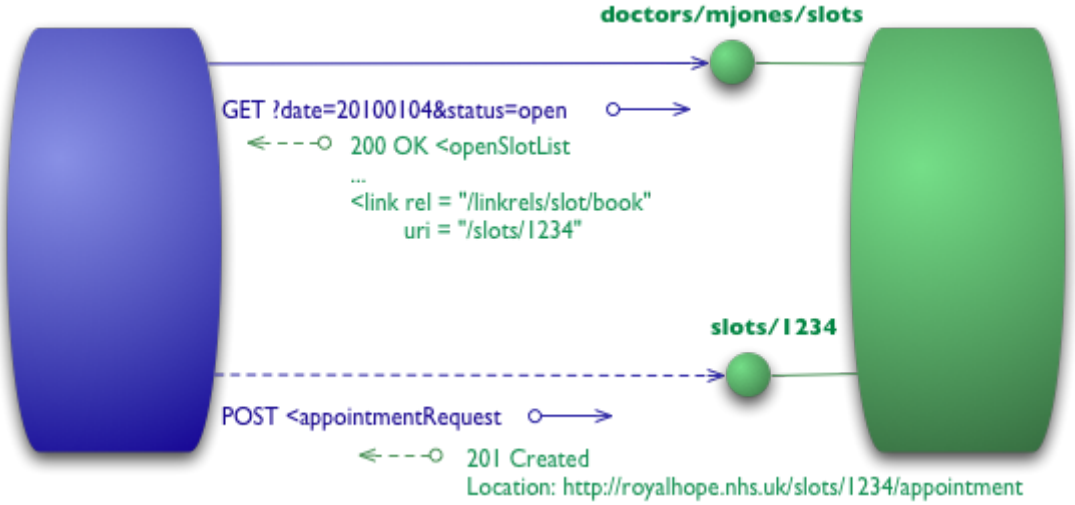
此响应的重要部分是使用 HTTP 响应代码来指示出现问题。在这种情况下，409 似乎是一个不错的选择，表明其他人已经以不兼容的方式更新了资源。不是使用返回码 200 而是包含错误响应，在第 2 级，我们明确地使用了类似这样的某种错误响应。由协议设计者决定使用什么代码，但如果出现错误，应该有一个非 2xx 响应。第 2 级介绍了使用 HTTP 动词和 HTTP 响应代码。

这里有一个不一致的地方。REST 倡导者谈论使用所有 HTTP 动词。他们还通过说 REST 试图从 Web 的实际成功中学习来证明他们的方法是正确的。但是万维网在实践中很少使用 PUT 或 DELETE。更多地使用 PUT 和 DELETE 有合理的理由，但网络的存在证明不是其中之一。

Web 存在支持的关键元素是安全（例如 GET）和非安全操作之间的强分离，以及使用状态代码来帮助传达遇到的各种错误。

3 级 - 超媒体控制

最后一层介绍了一些你经常听到的东西，它被称为 HATEOAS（超文本作为应用程序状态的引擎）它解决了如何从列表中获取空缺职位以了解如何进行预约的问题。




```
我们从在级别 2 中发送的相同初始 GET 开始

GET /doctors/mjones/slots?date=20100104&status=open HTTP/1.1
Host: royalhope.nhs.uk
但回应有一个新的元素
HTTP/1.1 200 OK
[various headers]

<openSlotList>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450">
     <link rel = "/linkrels/slot/book"
           uri = "/slots/1234"/>
  </slot>
  <slot id = "5678" doctor = "mjones" start = "1600" end = "1650">
     <link rel = "/linkrels/slot/book"
           uri = "/slots/5678"/>
  </slot>
</openSlotList>
```

每个插槽现在都有一个链接元素，其中包含一个 URI，告诉我们如何预约。

超媒体控件的重点是它们告诉我们下一步可以做什么，以及我们需要操作的资源的 URI。我们不必知道在哪里发布我们的预约请求，响应中的超媒体控件会告诉我们如何去做。

POST 将再次复制 2 级的

```
POST /slots/1234 HTTP/1.1
[various other headers]

<appointmentRequest>
  <patient id = "jsmith"/>
</appointmentRequest>

HTTP/1.1 201 Created
Location: http://royalhope.nhs.uk/slots/1234/appointment
[various headers]
回复包含许多超媒体控件，用于接下来要做的不同事情

<appointment>
```

```
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
  <link rel = "/linkrels/appointment/cancel"
        uri = "/slots/1234/appointment"/>
  <link rel = "/linkrels/appointment/addTest"
        uri = "/slots/1234/appointment/tests"/>
  <link rel = "self"
        uri = "/slots/1234/appointment"/>
  <link rel = "/linkrels/appointment/changeTime"
        uri = "/doctors/mjones/slots?date=20100104&status=open"/>
  <link rel = "/linkrels/appointment/updateContactInfo"
        uri = "/patients/jsmith/contactInfo"/>
  <link rel = "/linkrels/help"
        uri = "/help/appointment"/>
</appointment>
```

我应该强调，虽然 RMM 是一种思考 REST 元素的好方法，但它并不是 REST 本身级别的定义。Roy Fielding 明确表示，3 级 RMM 是 REST 的先决条件。与软件中的许多术语一样，REST 有很多定义，但自从 Roy Fielding 创造了这个术语，他的定义应该比大多数人更重要。

我发现这个 RMM 的有用之处在于它提供了一个很好的循序渐进的方式来理解restfulness.思维背后的基本思想。因此，我认为它是帮助我们了解概念的工具，而不是应该在某种评估机制中使用的东西。我认为我们还没有足够的示例来真正确定 restful 方法是集成系统的正确方法，我确实认为这是一种非常有吸引力的方法，并且在大多数情况下我会推荐这种方法。

这个模型的吸引力在于它与常见设计技术的关系。

- 级别 1 通过使用分而治之，将大型服务端点分解为多个资源来解决处理复杂性的问题。
- Level 2 引入了一组标准的动词，以便我们以相同的方式处理类似的情况，消除不必要的变化。

局限：

不是所有业务都可以被表示为资源

这在构建REST API时是经常会碰到的，我们不能正确表示资源，所以被迫采用了其他实际。

例如，一个简单的用户登入登出，如果抽象为资源可能变成了创建一个会话， 即`POST /api/session`，这其实远不如`POST /login`来的直观。

又比如，一个播放器资源，当我们要播放或停止时，一个典型的设计肯定是`POST /player/stop`， 而如果要满足REST规范，停止这个动作将不复存在，取而代之的是`播放器状态`，API形如 `POST /player {state:"stop"}`。

以上两例都展示了，REST在某些场景下可能并不能提供良好的表现力。

基于 HTTP+JSON 的类 REST API 设计

http://www.ruanyifeng.com/blog/2014/05/restful_api.html

一、协议

API与用户的通信协议，总是使用HTTPs协议。

二、域名

应该尽量将API部署在专用域名之下。

```
https://api.example.com
```

如果确定API很简单，不会有进一步扩展，可以考虑放在主域名下。

```
https://example.org/api/
```

三、版本（Versioning）

应该将API的版本号放入URL。

```
https://api.example.com/v1/
```

另一种做法是，将版本号放在HTTP头信息中，但不如放入URL方便和直观。Github采用这种做法。

四、路径（Endpoint）

路径又称"终点"（endpoint），表示API的具体网址。

在RESTful架构中，每个网址代表一种资源（resource），所以网址中不能有动词，只能有名词，而且所用的名词往往与数据库的表格名对应。一般来说，数据库中的表都是同种记录的"集合"（collection），所以API中的名词也应该使用复数。

举例来说，有一个API提供动物园（zoo）的信息，还包括各种动物和雇员的信息，则它的路径应该设计成下面这样。

- https://api.example.com/v1/zoos
- https://api.example.com/v1/animals
- https://api.example.com/v1/employees

五、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（括号里是对应的SQL命令）。

- GET（SELECT）：从服务器取出资源（一项或多项）。
- POST（CREATE）：在服务器新建一个资源。
- PUT（UPDATE）：在服务器更新资源（客户端提供改变后的完整资源）。
- PATCH（UPDATE）：在服务器更新资源（客户端提供改变的属性）。
- DELETE（DELETE）：从服务器删除资源。

六、过滤信息（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

参数的设计允许存在冗余，即允许API路径和URL参数偶尔有重复。比如，GET /zoo/ID/animals 与 GET /animals?zoo_id=ID 的含义是相同的。

七、状态码（Status Codes）

八、错误处理（Error handling）

如果状态码是4xx，就应该向用户返回出错信息。一般来说，返回的信息中将error作为键名，出错信息作为键值即可。

```
{
 error: "Invalid API key"
}
```

九、返回结果

针对不同操作，服务器向用户返回的结果应该符合以下规范。

- GET /collection：返回资源对象的列表（数组）
- GET /collection/resource：返回单个资源对象
- POST /collection：返回新生成的资源对象
- PUT /collection/resource：返回完整的资源对象
- PATCH /collection/resource：返回完整的资源对象
- DELETE /collection/resource：返回一个空文档

十、Hypermedia API

RESTful API最好做到Hypermedia，即返回结果中提供链接，连向其他API方法，使得用户不查文档，也知道下一步应该做什么。

比如，当用户向api.example.com的根目录发出请求，会得到这样一个文档。

```
{"link": {
"rel":   "collection https://www.example.com/zoos",
"href":  "https://api.example.com/zoos",
"title": "List of zoos",
"type":  "application/vnd.yourformat+json"
}}
```

上面代码表示，文档中有一个link属性，用户读取这个属性就知道下一步该调用什么API了。rel表示这个API与当前网址的关系（collection关系，并给出该collection的网址），href表示API的路径，title表示API的标题，type表示返回类型

十一、其他

（1）API的身份认证应该使用OAuth 2.0框架。

（2）服务器返回的数据格式，应该尽量使用JSON，避免使用XML。