1. 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/
```

2. 版本（Versioning）

应该将API的版本号放入URL。

```
http://www.example.com/app/1.0/foo

http://www.example.com/app/1.1/foo

http://www.example.com/app/2.0/foo
```

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就采用了这种做法。

因为不同的版本，可以理解成同一种资源的不同表现形式，所以应该采用同一个URL。版本号可以在HTTP请求头信息的Accept字段中进行区分（参见Versioning REST Services）：

```
Accept: vnd.example-com.foo+json; version=1.0

Accept: vnd.example-com.foo+json; version=1.1

Accept: vnd.example-com.foo+json; version=2.0
```

3. 路径（Endpoint）

路径又称"终点"（endpoint），表示API的具体网址，每个网址代表一种资源（resource）

(1) 资源作为网址，只能有名词，不能有动词，而且所用的名词往往与数据库的表名对应。

举例来说，以下是不好的例子:

```
/getProducts
/listOrders
/retreiveClientByOrder?orderId=1
```

对于一个简洁结构，你应该始终用名词。 此外，利用的HTTP方法可以分离网址中的资源名称的操作。

```
GET /products ：将返回所有产品清单
POST /products ：将产品新建到集合
GET /products/4 ：将获取产品 4
PATCH（或）PUT /products/4 ：将更新产品 4
```

(2) API中的名词应该使用复数。无论子资源或者所有资源。

举例来说，获取产品的API可以这样定义

```
获取单个产品：http://127.0.0.1:8080/AppName/rest/products/1
获取所有产品：http://127.0.0.1:8080/AppName/rest/products
```

3. HTTP动词

对于资源的具体操作类型，由HTTP动词表示。

常用的HTTP动词有下面四个（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）。
- DELETE（DELETE）：从服务器删除资源。

还有三个不常用的HTTP动词。

- PATCH（UPDATE）：在服务器更新(更新)资源（客户端提供改变的属性）。
- HEAD：获取资源的元数据。
- OPTIONS：获取信息，关于资源的哪些属性是客户端可以改变的。

下面是一些例子。

```
GET /zoos：列出所有动物园
POST /zoos：新建一个动物园（上传文件）
GET /zoos/ID：获取某个指定动物园的信息
PUT /zoos/ID：更新某个指定动物园的信息（提供该动物园的全部信息）
PATCH /zoos/ID：更新某个指定动物园的信息（提供该动物园的部分信息）
DELETE /zoos/ID：删除某个动物园
GET /zoos/ID/animals：列出某个指定动物园的所有动物
DELETE /zoos/ID/animals/ID：删除某个指定动物园的指定动物
```

4. 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。query_string 查询字符串,地址栏后面问号后面的数据,格式: name=xx&sss=xxx

```
?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 /zoos/ID/animals 与 GET /animals?zoo_id=ID 的含义是相同的。

6. 状态码（Status Codes）

服务器向用户返回的状态码和提示信息，常见的有以下一些（方括号中是该状态码对应的HTTP动词）。

> - 200 OK - [GET]：服务器成功返回用户请求的数据
> - 201 CREATED - [POST/PUT/PATCH]：用户新建或修改数据成功。
> - 202 Accepted - [*]：表示一个请求已经进入后台排队（异步任务）
> - 204 NO CONTENT - [DELETE]：用户删除数据成功。
> - 400 INVALID REQUEST - [POST/PUT/PATCH]：用户发出的请求有错误，服务器没有进行新建或修改数据的操作
> - 401 Unauthorized - [*]：表示用户没有权限（令牌、用户名、密码错误）。
> - 403 Forbidden - [*] 表示用户得到授权（与401错误相对），但是访问是被禁止的。
> - 404 NOT FOUND - [*]：用户发出的请求针对的是不存在的记录，服务器没有进行操作，该操作是幂等的。
> - 406 Not Acceptable - [GET]：用户请求的格式不可得（比如用户请求JSON格式，但是只有XML格式）。
> - 410 Gone -[GET]：用户请求的资源被永久删除，且不会再得到的。
> - 422 Unprocesable entity - [POST/PUT/PATCH] 当创建一个对象时，发生一个验证错误。
> - 500 INTERNAL SERVER ERROR - [*]：服务器发生错误，用户将无法判断发出的请求是否成功。

状态码的完全列表参见这里或这里。

7. 错误处理（Error handling）

如果状态码是4xx，服务器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"
}
```

8. 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/ID：返回单个资源对象(json)
- POST /collection：返回新生成的资源对象(json)
- PUT /collection/ID：返回完整的资源对象(json)
- DELETE /collection/ID：返回一个空文档(空字符串)

9. 超媒体（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，Github的API就是这种设计，访问api.github.com会得到一个所有可用API的网址列表。

```
{
"current_user_url": "https://api.github.com/user",
"authorizations_url": "https://api.github.com/authorizations",
// ...
}
```

从上面可以看到，如果想获取当前用户的信息，应该去访问api.github.com/user，然后就得到了下面结果。

```
{
  "message": "Requires authentication",
  "documentation_url": "https://developer.github.com/v3"
}
```

上面代码表示，服务器给出了提示信息，以及文档的网址。

10. 其他

服务器返回的数据格式，应该尽量使用JSON，避免使用XML。