# RESTful架构风格的特点

## 资源郭冬霞—测试i

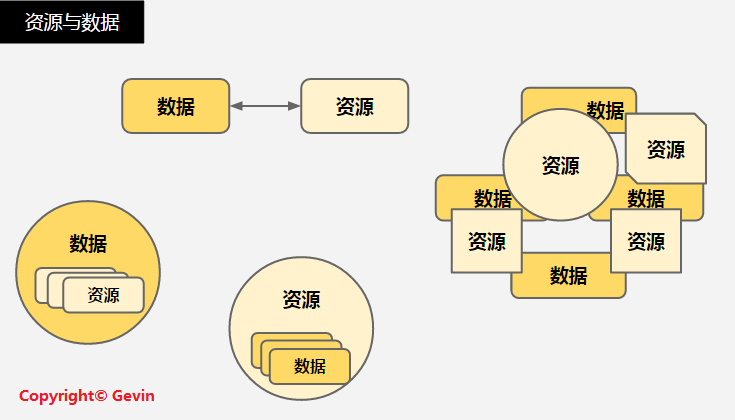
所谓“资源”，就是网络上的一个实体，或者说是网络上的一个具体信息。它可以是一短文本、一张图片、一首歌曲、一种服务、总之就是一个具体的是在。资源总是要通过某种载体反应其内容，文本可以同txt格式来表现，也可以用HTML格式、XML格式表现，甚至可以采用二进制格式；图片可以使用JPG格式表现，也可以使用PNG格式表现；JSON是现在最常用的资源表示格式。

资源是以json（或者其他）为载体的、面向用户的一组数据集，资源对信息的表达倾向于模型概念中的数据：

* 资源总是以某种Representation为载体显示的，即序列化的信息
* 常用的Representation是json(推荐)或者xml（不推荐）等
* Representation 是REST架构的表现层

相对而言，数据（尤其是数据库）是一种更加抽象的、对计算机更高效和友好的数据表现形式，更多存在于逻辑模型中

资源和数据关系如下



## 统一接口

RESTful架构风格规定，数据的元操作，即CRUD操作，分别对应HTTP  
方法的：GET获取资源，POST用来新建资源（也可以用来进行更行资源），PUT用来更新资源，DELETE用来删除资源，这样就统一了数据操作的接口，仅通过HTTP方法，就可以完成对数据的所有增删改查工作。

即：

* 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（客户端提供完整资源数据）。
* PATCH（UPDATE）：在服务器更新资源（客户端提供需要修改的资源数据）。
* DELETE（DELETE）：从服务器删除资源。

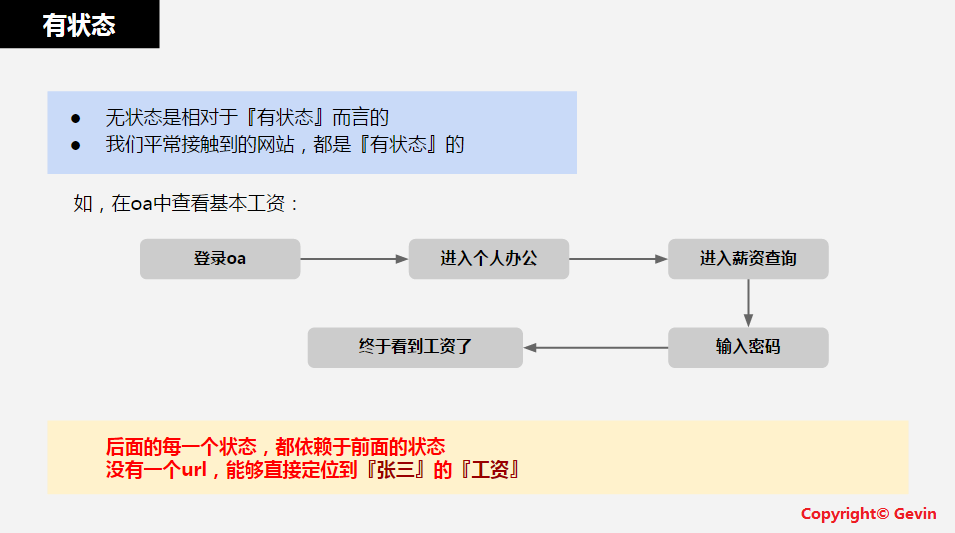
## URI

可以用一个URI（统一资源标识符；URL（统一资源定位符））指向资源，即每一个URI都对应一个特定的资源。要获取这个资源，访问它的URI就可以，因此URI就成了每一个资源的地址或识别符。

一般的，每个资源至少有一个URI与之对应，最典型的URI就是URL。

## 无状态

所谓的无状态，及所有资源都可以通过URI来定位，而且这个定位于其他资源无关，也不会因为其他资源的变化而产生变化。有状态和无状态的区别，举个简单的例子说明一下。如查询员工的工资，如果查询工资需要登录系统，进入查询相关的操作后，获取工资的多少，则这种情况就是**有状态**的，因为查询工资的每一步都需要依赖于前一步的操作，只要前置操作不成功，后续操作就无法执行录入过输入一个url就可以获得指定员工的工资，则这种情况是**无状态**的，因为获取工作不依赖于其他的资源或者状态，且这种情况下，员工工资是一个资源，由一个url与之对应，可以通过HTTP中的GET方法得到资源，这就是典型的RESTful风格。





# 2.1 RESTful简单用例

## 2.1.1 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。

还有两个不常用的HTTP动词

* 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。

下面试一些例子。

* GET /zoos：列出所有动物园
* POST /zoos：新建一个动物园
* 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

## 2.1.2 过滤信息（Filtering）

如果记录数量很多，服务器不可能将他们全部返回给用户。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。

## 2.1.3 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，该操作是幂等的（Idempotent）。
* 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，该操作是幂等的。
* 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。

## 2.1.4 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” }

## 2.1.5 返回结构

针对不同的操作，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

## 2.1.6 Hypermedia API

RESTful API最好做到Hypermedia API，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标识这个API与当前网站的关系，href标识API的路径，title标识API的标题，type标识返回类型。

Hypermedia API的设计被称为HATEOAS。Github的API就是折后在哪个设计，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>",

// ...

}

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

{

"message": "Requires authentication",

"documentation\_url": "<https://developer.github.com/v3>"

}

上面代码标识，服务器给出了提示信息，以及文档的网址。

# 3.1 设计规范

## 3.1.1 URL设计

### 3.1.1.1 动词 + 宾语

RESTful 的核心思想就是，客户端发出的数据指令都是“动词 + 宾语”的结构。比如，**GET /articles** 这个命令，GET 是动词，/articles 是宾语。

动词通常就是五种HTTP 方法，对应CRUD操作。

* GET：读取（Read）
* POST：新建（Create）
* PUT：更新（Update）
* PATCH：更新（Update），通常是部分更新
* DELETE：删除（Delete）

根据HTTP规范，动词一律大写。

### 3.1.1.2 动词的覆盖

有些客户端只能使用GET和POST这两种方法。服务器必须接受POST模拟其他三个方法（PUT、PATCH、DELETE）。

这时，客户端发出的 HTTP 请求，要加上X-HTTP-Method-Override属性，告诉服务器应该使用哪一个动词，覆盖POST方法。

POST /api/Person/4 HTTP/1.1

X-HTTP-Method-Override: PUT

上面代码中，X-HTTP-Method-Override 指定本次请求的方法是PUT，而不是POST。

### 3.1.1.3 宾语必须是名词

宾语就是 API 的 URL，是 HTTP 动词作用的对象。它应该是名词，不能是动词。比如，/articles这个 URL 就是正确的，而下面的 URL 不是名词，所以都是错误的。

* /getAllCars
* /createNewCar
* /deleteAllRedCars

### 3.1.1.4 复数URL

既然 URL 是名词，那么应该使用复数，还是单数？

这没有统一的规定，但是常见的操作是读取一个集合，比如GET /articles（读取所有文章），这里明显应该是复数。

为了统一起见，建议都使用复数 URL，比如GET /articles/2要好于GET /article/2。

# 4.1 RESTful的优点

1. 轻量，直接基于http，不再需要任何别的诸如消息协议。get/post/put/delete为CRUD操作

2. 面向资源，一目了然，具有自解释性。

3. 数据描述简单，一般以xml，json做数据交换。

4. 无状态，在调用一个接口（访问、操作资源）的时候，可以不用考虑上下文，不用考虑当前状态，极大的降低了复杂度。

5. 简单、低耦合

---------------------------------------------------------------------

可更高效利用缓存来提高响应速度

通讯本身的无状态性可以让不同的服务器的处理一系列请求中的不同请求，提高服务器的扩展性

浏览器即可作为客户端，简化软件需求

相对于其他叠加在HTTP协议之上的机制，REST的软件依赖性更小

不需要额外的资源发现机制

在软件技术演进中的长期的兼容性更好

# 5.1 参考网址

## 5.1.1 <http://www.ruanyifeng.com/blog/2014/05/restful_api.html>

## 5.1.2

<https://blog.igevin.info/posts/restful-architecture-in-general/>

## 5.1.3

<http://www.ruanyifeng.com/blog/2018/10/restful-api-best-practices.html>

## 5.1.4

<https://blog.csdn.net/acmant/article/details/44492395>