RESTful 架构详解

- RESTful 约束条件
- 什么是资源
- 什么是表述层
- 表述层状态变化
- 状态变化结果

RESTful 全称 **Representational State Transfer**， 表述性(表现层)状态转移。软件架构风格。遵守REST架构约束的Web API被称为RESTful API

6条约束条件

Uniform interface(同一接口)

> 严格定义系统内部的将要暴露给消费者的资源的API接口，系统内部一个资源必须只有一个逻辑URL，，并且提供一种方式，这个方式可以通过和这个URL获取到资源。
>
> 一个资源的表现层必须包含这个资源的全部信息，也就是这个资源通过"表现层"表现出来。
>
> *Once a developer becomes familiar with one of your API, he should be able to follow the similar approach for other APIs.*

Client–server(客户端 - 服务器)

> 客户端和服务端独立，没有任何依赖，客户端只需要知道资源的url。

Stateless(无状态)

> 客户端和服务端无状态，服务端不会保存来自客户端的HTTP请求历史或者其他上下文cookie等，将每一个请求视作新的第一次的请求(No session, no history)
>
> 无状态通信原则，并不是说客户端应用不能有状态，而是指服务端不应该保存客户端状态。

Cacheable

> 缓存，缓存的时候，必须要表明数据是可缓存的(Cacheable)， 缓存并不和无状态冲突(无状态是指客户端和应用有关的 session、上下文等)
>
> *Well-managed caching partially or completely eliminates some client-server interactions, further improving scalability and performance.*

Layered system 分层系统(微服务东西，一个服务干一件事情)

> 使用分层系统架构，如API部署在A服务，数据存储在B服务，授权在C服务

Code on demand (optional)

> 这个约束是可选的。大多数情况下，您将以XML或JSON的形式发送资源的静态表示。但当你需要的时候，你可以自由地返回可执行代码来支持你的应用程序的一部分

REST原则，围绕**资源**展开讨论：

- 资源与URI
- 资源的表述
- 状态的转移

URL定位资源，用HTTP动词（GET,POST,DELETE,DETC）描述操作。

资源

任何事物，有被引用的必要(*Any information that can be named can be a resource*)，那就是个资源，资源可以是实体，也可以是一个抽象概念，如：

- 某人使用的手机号码
- 某个用户的个人信息
- 文档、照片等。

资源的表述(representation)

HTTP方法可以获取资源，确切的说，客户端获取的只是资源的表述而已。 资源在外界的具体呈现，可以有多种表述(或成为表现、表示)形式，在客户端和服务端之间传送的也是资源的表述，而不是资源本身。 例如文本资源可以采用html、xml、json等格式，图片可以使用PNG或JPG展现出来。

资源的表述包括数据和描述数据的元数据，例如，HTTP头"Content-Type" 就是这样一个元数据属性

URL

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。

要让一个资源可以被识别，那么需要有一个标识，在WEB 中这个标识就是 URL，URL 可以看做是资源的地址，也可以看做是资源的名称，如果资源不能使用URL表示，那么它就不能算是一个资源，只能说是一些资源的信息而，已，**URL应该是能被人识别，给人直觉上的关联，让人一看就明白这是一个什么样的资源**。

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**

URL的内容必须**全部小写**

代表的是一种资源，而不是一个动作，**所以URL 中不可以出现动词**

- **MUST:** URL中不出现动词。
- **MUST:** 以复数名词或词组来命名集合。
- **MUST:** URL中的所有内容必须都采用小写。
- **MUST:** URL中不出现组织信息。

一些不好的URL 命名

> - {baseURL}/v1/product/1234* *(noun does not use plural)*
> - *{baseURL}/v1/ProductGroup/Update (case used)*
> - *{baseURL}/v1/prdocuts/query (verbs used)*
> - *{baseURL}/v1/pcsdproducts/1234 (IT organization information appears)*

- 使用 - 或者_ 隔开单词： `http://www.oschina.net/news/38119/oschina-translate-reward-plan。`
- / 表示资源的层级关系
- ? 用于表示过滤资源

  `/devices?startIndex=0&size=20`

HTTP谓词 -> State Transfer：状态变化(资源的变化)

这种转化是建立在表现层之上的(发送Json 数据改变增加数据库数据。put更新数据库等)，所以就是"表现层状态转化"。

Url指向了一个资源 , 但是对资源的操作，如更新，删除等操作都是动作的操作，而这些动作是根据 http 谓词实现

无论什么样的资源都可以通过相同的接口进行资源的访问，接口应该使用HTTP 的**GET/PUT/POST**等，

- GET： **获取**资源，可以是单个或者是多个
- POST：**创建**资源
- PUT **更新**资源，客户端需要提供，改变后完整的资源
- patch **更新**资源，客户端只需要提供，改变的那个属性
- DELETE **删除**资源
- HEAD 获取资源的元数据
- OPTIONS 获取信息

URL 提供资源，HTTP 谓词，提供对这些资源的操作

以上有了 资源 -> 资源的表述 -> 资源的变化(操作)

如何设计

URL `https://restfulapi.net/rest-api-design-tutorial-with-example/`

设计REST api 的前提是，确定那些被 `resentted` 为资源的对象(object)

我们假设在应用中只有一下两类资源

- Devices
- Configurations

二者两个资源的关系是，configuration是device 的下级资源，一个device 可以有很多的configuration

我们可能会设计的url为：

```
/devices
/devices/{id}

/configurations
/configurations/{id}

/devices/{id}/configurations
/devices/{id}/configurations/{id}
```

观察上面的url可以发现，设计url，很重要的一点事：`不使用任何的动词或者操作性的词语`，还有一点就是如：/devices 使用资源的时候一般都是使用到复数，代表是资源这一类(很多资源)，当用于获取单个资源的时候，如：/devices/{id}，这里也可以很清楚的表述出，我们要获取的资源是devices 这类资源中的第id个资源。

如 `/devices/{id}/configurations` 是用于获取第 id 个 device 下的所有 configurations资源

同理，也可以很好的解释 /devices/{id}/configurations/{id} 设计意图。

如下URL：

```
HTTP GET /devices
HTTP GET /configurations
```

很清楚的需要获取的是 所有的 device 或者 configuration 而不是一个

获取资源

```
HTTP GET /devices?startIndex=0&size=20
HTTP GET /configurations?startIndex=0&size=20
```

采用过滤，? 是过滤的意思，我们也需要时在获取买足过滤条件的 device 和configurations， 获取的也肯定不是一个。

删除某个资源：

```
HTTP DELETE /devices/{id}
HTTP DELETE /configurations/{id}
```

更新某个资源

```
HTTP PUT /devices/{id}
HTTP PUT /configurations/{id}
```

创建资源

```
HTTP POST /devices
HTTP POST /configurations
```

这里也使用复数的形式，从这个可以理解，在url 中使用复数是说明者一类资源，而不是说是由多个资源。(一类资源中可能会有多个这种资源，也可能只有一个，或者一个都没有)，这里创建的url说明的就是创建这种资源，表示的都创建这种资源，而并没有指明是创建多少个，一般在使用的时候，只是创建一个。

```
//Apply Configuration on a device,将 configurations 资源应用到(更新) devices 资源
HTTP PUT /devices/{id}/configurations

//Remove Configuration on a device 删除第 id 个 device 下的第 id 个 configuration
HTTP DELETE /devices/{id}/configurations/{id}
```

操作是否正确呢？操作状态码

2XX 成功状态码

客户端发起请求时， 这些请求通常都是成功的。服务器有一组用来表示成功的状态码，分别对应于不同类型的请求。

状态码	状态消息	含义	实例
200	OK	服务器成功处理了请求（这个是我们见到最多的）	HTTP协议详解-200
201	Created（已创建）	对于那些要服务器创建对象的请求来说，资源已创建完毕。	
202	Accepted（已接受）	请求已接受， 但服务器尚未处理	
203	Non-Authoritative Information（非权威信息）	服务器已将事务成功处理，只是实体Header包含的信息不是来自原始服务器，而是来自资源的副本。	
204	No Content(没有内容)	Response中包含一些Header和一个状态行， 但不包括实体的主题内容（没有response body）	状态码204
205	Reset Content(重置内容)	另一个主要用于浏览器的代码。意思是浏览器应该重置当前页面上所有的HTML表单。	
206	Partial Content（部分内容）	部分请求成功	状态码206

4XX客户端错误状态码

有时客户端会发送一些服务器无法处理的东西，比如格式错误的Request, 或者最常见的是， 请求一个不存在的URL。

状态码	状态消息	含义	实例
400	Bad Request（坏请求）	告诉客户端，它发送了一个错误的请求。	状态码400
401	Unauthorized（未授权）	需要客户端对自己认证	HTTP协议之基本认证-401
402	Payment Required（要求付款）	这个状态还没被使用， 保留给将来用	
403	Forbidden（禁止）	请求被服务器拒绝了	状态码403
404	Not Found（未找到）	未找到资源	HTTP协议详解-404
405	Method Not Allowed（不允许使用的方法）	不支持该Request的方法。	状态码405
406	Not Acceptable（无法接受）		
407	Proxy Authentication Required(要求进行代理认证)	与状态码401类似， 用于需要进行认证的代理服务器	HTTP协议之代理-407
408	Request Timeout（请求超时）	如果客户端完成请求时花费的时间太长， 服务器可以回送这个状态码并关闭连接	
409	Conflict（冲突）	发出的请求在资源上造成了一些冲突	
410	Gone（消失了）	服务器曾经有这个资源，现在没有了， 与状态码404类似	
411	Length Required（要求长度指示）	服务器要求在Request中包含Content-Length。	状态码411
412	Precondition Failed（先决条件失败）		
413	Request Entity Too Large（请求实体太大）	客户端发送的实体主体部分比服务器能够或者希望处理的要大	状态码413

5xx(服务器错误)

这些状态码表示服务器在处理请求时发生内部错误。这些错误可能是服务器本身的错误，而不是请求出错。

500(服务器内部错误)服务器遇到错误，无法完成请求。

501(尚未实施)服务器不具备完成请求的功能。例如，服务器无法识别请求方法时可能会返回此代码。

502(错误网关)服务器作为网关或代理，从上游服务器收到无效响应。

503(服务不可用)服务器目前无法使用(由于超载或停机维护)。通常，这只是暂时状态。

504(网关超时)服务器作为网关或代理，但是没有及时从上游服务器收到请求。

505(HTTP 版本不受支持)服务器不支持请求中所用的 HTTP 协议版本。

RESTful 教程

菜鸟教程

组织的故事性、连贯性、结构(总分总/总分)