RESTful 学习报告

目录

一、提出原因

互联网将传统单机环境中的软件与系统之间的通信融合，“互联网软件”即互联网环境中使用的软件的出现，激发了发开人员对结构清晰、符合标准、易于理解、扩展方便的互联网软件架构的需求。

Roy Thomas Fielding 在《Architectural Styles and the Design of Network-based Software Architectures》中提到，随着软件和网络的交叉，“My work is motivated by the desire to understand and evaluate the architectural design of network-based application software through principled use of architectural constraints, therby obtaining the functional, performance, and social properties desired of an architecture.”在这样的发展背景下，Roy 提出了 RESTful 架构。

二、基本概念

RESTful 全称为 Representational State Transfer, 为一种 Web 应用程式架构风格，而不是标准。

（1）RESTful 框架的主体为资源即 resources。每一个 URI 代表一种资源；

（2）客户端和服务器之间，传递这种资源的某种表现层；

（3）客户端通过四个 HTTP 动词，对服务器端资源进行操作，实现表现层状态转化；

1. 资源 Resources

资源就是网络上的一个实体，可以是文本、图片、歌曲、服务等。在 RESTful 框架中，使用一个 URI 指向它，每种资源对应一个特定的 URI，即每个资源都有独一无二对应的识别符。当需要与资源进行交互的时候，即调用其 URI 对资源进行操作。

2. 表现层 Representation

URI 代表资源的实体，但是不代表它的形式，因此表现层是将资源具体呈现出来的形式。比如，文本可以用 txt 格式表现，也可以用 HTML 格式、XML 格式、JSON 格式表现，甚至可以采用二进制格式；图片可以用 JPG 格式表现，也可以用 PNG 格式表现。

3. 状态转化 State Transfer

当用户访问网站时，浏览器作为客户端和服务器之间进行交互，从而产生数据和状态的变化。但是 HTTP 是一个无状态协议，因此所有的状态都应该保存在服务器端。当客户端想要进行操作时，通过不同的动作，使服务器端发生不同的状态转化，从而完成整个操作。比如常用的方法，GET 来获取资源，POST 来新建资源，PUT 来更新资源，DELETE 来删除资源。

三、API 设计

1. API 与用户的通信协议，使用 HTTP/HTTPs 协议。

2. 每个 URL 代表一种资源，因此没有动词，只有名词。具体的操作方式即动词应该放在相关协议中。当某些动词不能直接使用 HTTP 协议的动词时，将其整体作为一种资源。

non-RESTful	RESTful
/posts/show/1	GET /posts/1
POST /accounts/1/transfer/500/to/2	POST /transaction HTTP/1.1 Host: 127.0.0.1 from=1&to=2&amount=500.00

3. 版本可以放在 URL 中也可以放在 HTTP/HTTPs 头的信息中。如果放在 URL 中，更加直观；如果放在头中，由于不同版本的资源本质上是同一个资源，更加符合每个资源一个 URL 的风格。

4. 常用的操作：

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

5. 状态码 Status Codes

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

四、实践学习

在学习了相关的理论知识后，我对 RESTful 框架的理解并不深入，因此找到了一个基于 RESTful 的实践教程学习从而加深理解。以下链接为我学习教程后完成的 todoList 的源代码

https://github.com/laylalaisy/RESTful_todoList

在本项目中，基于 RESTful API 服务器完成一个 todoList。Endpoint 是指网址最后的部分，例如，POST 的网址会是：localhost:3000/todo 。

操作	操作方式	Endpoint
建立一个待办事项	POST	/todo
读取一个待办事项	GET	/todo/:id
删除一个待办事项	DELETE	/todo/:id
读取整个待办事项列表	GET	/todo
更新一个待办事项	PUT	/todo/:id

```
var Http = require( 'http' ),
    Router = require( 'router' ),
    server,
    router;
var BodyParser = require('body-parser');
var counter = 0,
    todoList = {};

/* router */
router = new Router();

/* server */
server = Http.createServer( function( request, response ) {
  router( request, response, function( error ) {
    if ( !error ) {
      response.writeHead( 404 );
    } else {
      // Handle errors
      console.log( error.message, error.stack );
      response.writeHead( 400 );
    }
    response.end( 'RESTful API Server is running!\n' );
  });
});

server.listen( 3000, function() {
  console.log( 'Listening on port 3000' );
});

router.use(BodyParser.text());

/* POST: create item */
```

```
function createItem(request, response){
  var id = counter += 1,
      item = request.body;

  console.log('Create item', id, item);
  todoList[id] = item;
  response.writeHead(201, {
    'Content-Type' : 'text/plain',
    'Location' : '/todo/' + id
  });
  response.end('Item');
}

router.post('/todo', createItem);

// GET: read item
function readItem(request, response){
  var id = request.params.id,
      item = todoList[id];
  if(typeof item !== 'string'){
    console.log('Item not found', id);
    response.writeHead(404);
    response.end('\n');
    return;
  }
  console.log('Read item', id, item);
  response.writeHead(200, {
    'Content-Type': 'text/plain'
  });
  response.end(item + '\n');
}

router.get('/todo/:id', readItem);

/* DELETE: delete item */
function deleteItem(request, response){
  var id = request.params.id;

  if(typeof todoList[id] !=='string'){
    console.log('Item not found', id);
    response.writeHead(404);
    response.end('\n');
    return;
  }
```

```
  console.log('Delete item', id);
  todoList[id] = undefined;   // delete item
  response.writeHead(204, {
    'Content-Type' : 'text/plain'
  });
  response.end('');           // response nothing
}

router.delete('/todo/:id', deleteItem);

/* GET: read item list */
function readList(request, response){
  var item,
      itemList = [],
      listString;

  for(id in todoList){
    if(!todoList.hasOwnProperty(id)){   // test if this id is existed
      continue;
    }
    item = todoList[id];
    if(typeof item !== 'string'){       // test if this id is valid
      continue;
    }
    itemList.push(item);
  }
  console.log('Read List: \n', JSON.stringify(
    itemList,
    null,
    ' '
  ));
  listString = itemList.join('\n');
  response.writeHead(200, {
    'Content-Type' : 'text/plain'
  });
  response.end(listString + '\n');
}
router.get('/todo', readList);

/* PUT: update item */
function updateItem(request, response){
  var id = request.params.id,
      item = request.body;
```

```
  if(typeof todoList[id] !== 'string'){
    console.log('Item not found', id);
    response.writeHead(404);
    response.end('\n');
    return;
  }

  console.log('Update item', id, item);

  todoList[id] = item;
  response.writeHead(201, {
    'Content-Type' : 'test/plain',
    'Location' : '/todo/' + id
  });
  response.end(item + '\n');
}

router.put('/todo/:id', updateItem);
```

五、优点

1. 满足浏览器作为客户端的需求，从而简化了软件需求；
2. 不需要额外的资源发现机制；
3. 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；
4. 可更高效利用缓存来提高响应速度

参考：

《Architectural Styles and the Design of Network-based Software Architectures》：www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

理解 RESTful 架构：

http://www.ruanyifeng.com/blog/2011/09/restful.html?bsh_bid=1717507328

Node.js RESTful API 实例教学：

https://nodejust.com/node-js-restful-api-tutorial/