restful_00001

restful restful https gzip 对称加密

---

restful理论

什么是WEB？

> 要了解REST那么必须要了解WEB，WEB（World Wide Web）是连接所有单机资源、把所有资源网络化、共享化；

WEB 主要有以下组成部分：

> 1. URI：统一资源定位符
> 2. HTTP：超文本传输协议
> 3. HyperText：超文本
> 4. MIME：多用途互联网邮件扩展

WEB的几个发展历程：

> 静态内容阶段（HTML）—> 脚本语言阶段（JSP、ASP、JS）—> RIA（Rich Internet Application）—> 移动WEB阶段（ANDROID/IOS/WINP）；
> 通过这几个阶段的发展可以发现资源越来越动态化、复杂化；那么对于资源的管理是尤为重要的。

restful 定义

> 定义：REST（Representation Status Transfer）表述性状态转移

核心点：

1. Resource 与URI：资源
    1. 资源是看待服务器的一种方式、将服务器看待多个离散点资源的整合；
    2. URI是对资源的一种抽象，通过URI就能够定位服务器端的具体资源；对于URI的设计是可寻址性原则，具有自描述性，需要在形式上给人以直觉上的关联；
2. Representations ：资源表述性
    1. 通过URI可以拿到资源抽象信息描述以及元数据并不是具体的资源；资源表述有多种方式：JSON、XML等；
3. Status Transfer：状态转移
    1. 状态分为客户端状态（客户端维护）和资源状态（服务端维护），状态转移是针对于资源状态来说的；服务端通过超媒体形式向客户端提供资源的状态信息，客户端通过操作来达到资源状态的转移
4. Uniform Interfece：统一接口
    1. 统一接口包含了一组受限的预定义的操作，不论什么样的资源，都是通过使用相同的接口进行资源的访问；接口应该使用标准的HTTP方法如GET，PUT和POST，并遵循这些方法的语义；基本组件：
    2. 7个HTTP方法
    3. HTTP头信息
    4. HTTP响应状态码
    5. 内容协商机制（数据传输格式
    6. 缓存机制
    7. 客户端身份认证机制
5. HyperText Driven：超文本驱动
    1. 将超媒体作为应用状态的引擎（Hypermedia As The Engine Of Application State，来自Fielding博士论文中的一句话，缩写为HATEOAS）
    2. 将Web应用看作是一个由很多状态（应用状态）组成的有限状态机。资源之间通过超链接相互关联，超链接既代表资源之间的关系，也代表可执行的状态迁移。在超媒体之中不仅仅包含数据，还包含了状态迁移的语义。以超媒体作为引擎，驱动Web应用的状态迁移。通过超媒体暴露出服务器所提供的资源，服务器提供了哪些资源是在运行时通过解析超

> 媒体发现的，而不是事先定义的。从面向服务的角度看，超媒体定义了服务器所提供服务的协议。客户端应该依赖的是超媒体的状态迁移语义，而不应该对于是否存在某个URI或URI的某种特殊构造方式作出假设。一切都有可能变化，只有超媒体的状态迁移语义能够长期保持稳定。

REST6大特性：

> 1. Resource Oriented(面向资源)：一切皆资源ROA架构
> 2. Addressability(可寻址性)：URI资源定位
> 3. Connectedness(连通性)：资源互通 /customs/order/customs
> 4. Statelessness(无状态性)：状态分离，服务端不需要维护客户端状态对于扩展、高可用适应更加
> 5. Uniform Interface(统一接口)：Hypertext Driven：超文本驱动

规范

> restful是前后端交互规范的一种基于资源定义的主流规范，规范的合理与否对于api的设计至关重要；所以在下面列举出主要的规范列表

规范要点

1. 域名：api.项目名称.com
2. API是通过名词来命名、ACTION是通过HTTP1.1提供的标准操作来定义
3. 路径应该使用名词的复数形式
4. 资源之间存在连通性
    1. GET /tickets/12/messages - Retrieves list of messages for ticket #12
    2. GET /tickets/12/messages/5 - Retrieves message #5 for ticket #12
    3. POST /tickets/12/messages - Creates a new message in ticket #12
    4. PUT /tickets/12/messages/5 - Updates message #5 for ticket #12
    5. PATCH /tickets/12/messages/5 - Partially updates message #5 for ticket #12
    6. DELETE /tickets/12/messages/5 - Deletes message #5 for ticket #12
5. 文档化
6. 版本化
    1. 版本存放位置：header、url中；存储在url中更加合理因为资源能够通过浏览器更容易映射出来
7. 结果过滤、排序、搜索
    1. 过滤表示通过唯一查询参数获取想要的结果，例如想查询已经开售的tickets:GET /tickets?status=open
    2. 排序表示通过DESC/ASC对结果集进行排序，例如想查询优先级最高的tickets：GET /tickets?sort=-priority
    3. 有时候通过filter满足不了条件，那么通过匹配查询（es/lucence/like）方式，例如查询具有返程的tickets:GET /tickets?q=return&sort=-priority
    4. 限制资源的返回字段，例如想查询tickets的id，subject：GET /tickets/id?fileds=id,subject
8. 资源的DML都应该返回资源的表述，存放在location header
9. HATEOAS：超媒体即应用状态引擎的使用
10. 幂等操作：客户端操作不会随操作的次数而产生不同的效果
11. 响应的使用JSON方式

    1. 传输格式可以存放在header、url中；为了让浏览器更易于检索建议存放在url中
    2. 使用snake_case更易于解析和json lib库的解析
12. 默认打印，确保gzip支持：意思是去掉json的格式不易于查看而且节省流量不够明显；但是通过gzip压缩会更加明显
13. 不要使用信封的方式包裹消息体
14. 分页
    1. 分页连接放在Header links里面
    Link:
    https://api.github.com/user/repos?page=3&per_page=100; rel="next",
    https://api.github.com/user/repos?page=50&per_page=100; rel="last"
    总数量放在header X-Total-Count里面
15. 自动关联相关资源
    1. GET /tickets/12?embed=customer.name,assigned_user
    2. 返回结果：`{ "id" : 12, "subject" : "I have a question!", "summary" : "Hi, ....", "customer" : { "name" : "Bob" }, assigned_user: { "id" : 42, "name" : "Jim", } }`
16. 限制IP流量
    1. X-Rate-Limit-Limit - The number of allowed requests in the current period
    2. X-Rate-Limit-Remaining - The number of remaining requests in the current period
    3. X-Rate-Limit-Reset - The number of seconds left in the current period
    4. 为什么在X-Rate-Limit-Reset使用的是最后的剩余的秒数而不是时间错；因为 1. 时间包含了不需要的额外信息例如时间和时区；2. 最小的处理耗时和时间偏移问题
17. HTTP缓存:提供了两种标识缓存的方式
    1. http ETAG方式：客户端在请求后台资源，后台会给每个资源设置一个TAG标签，假如资源有修改那么TAG的值会进行相应的更新呢操作；下次请求过来服务端进行TAG的对比操作，决定是否需要进行新资源的数据返回

2. http LAST-MODIFIED方式：客户端在请求后台资源，后台会给每个资源设置一个过期时间（1.倒计时、2.过期日期）,客户端发现达到了过期时间进行重新资源的请求操作
3. 最好的方式通过两种标识缓存方式的结合

18. HTTP安全
    1. access_token
    2. Oauth2
19. 错误ERROR
    1. 错误信息的展现在HTML中是通过普通的错误页面展现的，但是对于API的错误方式更应该是具体、详细、有正对性的`{ "code" : 1234, "message" : "Something bad happened :(", "description" : "More details about the error here" }`
20. 对于DML操作的错误信息可以进行分层处理:`{ "code" :1024, "message" :"Validation Failed", "errors" :[ { "code" : 5432, "field" : "first_name", "message" : "First name cannot have fancy characters" }, { "code" : 5622, "field" : "password", "message" : "Password cannot be blank" } ]}`
21. HTTP状态码的定义：参照org.apache.http.HttpStatus

示例代码

```java
package com.rest.restapi.controller;

/**
 * Created by lennylv on 2017-1-3.
 * <p>
 *     api.dev.hawk.com/v1/users
 *     api.test.hawk.com/v1/users
 *     api.pro.hawk.com/v1/users
 * <p>
 * v1版本API
 */
@RestController
@RequestMapping(path = "/v1/users")
public class UserController {

    /**
     * 创建用户
     * 并且资源放在 LocationHeader 中
     *
     * @param userVo
     */
    @RequestMapping(method = RequestMethod.POST)
    @ResponseStatus(HttpStatus.CREATED)
    public void insertUser(@RequestBody(required = tru
e) UserVo userVo) {
        System.out.println(userVo);
    }

    /**
     * 删除用户
     *
     * @param id
     */
    @RequestMapping(value = "/{id}", method = RequestMe
thod.DELETE)
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void deleteUser(@PathVariable(value = "id")
int id) {
        System.out.println(id);
    }
```

```
    /**
     * 更新用户
     *
     * @param userVo
     */
    @RequestMapping(method = RequestMethod.PUT)
    @ResponseStatus(HttpStatus.CREATED)
    public void updateUser(@RequestBody(required = tru
e) UserVo userVo) {
        System.out.println(userVo);
    }

    /**
     * 通过id查询用户
     *
     * @param id
     * @return
     */
    @RequestMapping(path = "/{id}", method = RequestMet
hod.GET, consumes = MediaType.APPLICATION_JSON_VALUE, p
roduces = MediaType.APPLICATION_JSON_VALUE)
    public Object queryUserById(@PathVariable(value =
"id") int id) {
        return new UserVo(id, "zhangsan", "zhangsan@tc
l.com", "1234567890");
    }

    /**
     * 通过username过滤用户
     *api.dev.clean.com/v1/users?username=zhangsan/ 直接
映射RequestParam 对应参数;然后进行参数校验
     *
     * @param username
     * @return
     */
    @RequestMapping(method = RequestMethod.GET)
    public Object queryUserByFilter(@RequestParam(value
= "username") String username) {
        if ("zhangsan".equalsIgnoreCase(username)) {
```

```
            return new UserVo(0, "zhangsan", "zhangsa
n@tcl.com", "1234567890");
        } else {
            return null;
        }
    }

    /**
     * 通过username过滤用户
     * ?sort=-id,username
     *
     * @return
     */

    @RequestMapping(params = {QueryConstants.SORT}, met
hod = RequestMethod.GET)
    public Object queryUserSort(@RequestParam(QueryCons
tants.SORT) String sorts) {

        // 转换成排序列表
        return QueryOrderUtil.parseSort(sorts);
    }

    /**
     * 通过username过滤用户
     * ?q=zhangsan
     *
     * @param searches
     * @return
     */

    @RequestMapping(params = {QueryConstants.Q_PARAM},
method = RequestMethod.GET)
    public Object queryUserSearch(@RequestParam(QueryCo
nstants.Q_PARAM) final String searches) {

        // 转换成搜索列表
        return QuerySearchUtil.parseSearch(searches);
    }
```

```
    /**
     * 通过username过滤用户
     * ?page=3&size=100
     *
     * @param page
     * @param size
     * @return
     */

    @RequestMapping(params = {QueryConstants.PAGE, QueryConstants.SIZE}, method = RequestMethod.GET)
    public Object queryUserPaging(@RequestParam(QueryConstants.PAGE) final int page, @RequestParam(QueryConstants.SIZE) final int size) {
        return null;
    }

    /**
     * 通过用户id查询订单
     *
     * @param id
     * @return
     */
    @RequestMapping(path = "/{id}/orders", method = RequestMethod.GET)
    public Object queryOrderByUserId(@PathVariable(value = "id") int id) {
        ArrayList orders = new ArrayList<OrderVo>();
        orders.add(new OrderVo(1, "order", id, null));
        System.out.println(orders);
        return orders;
    }

}
```