# dada-单点登录系统实现

## dada-sso

单点登录系统开发

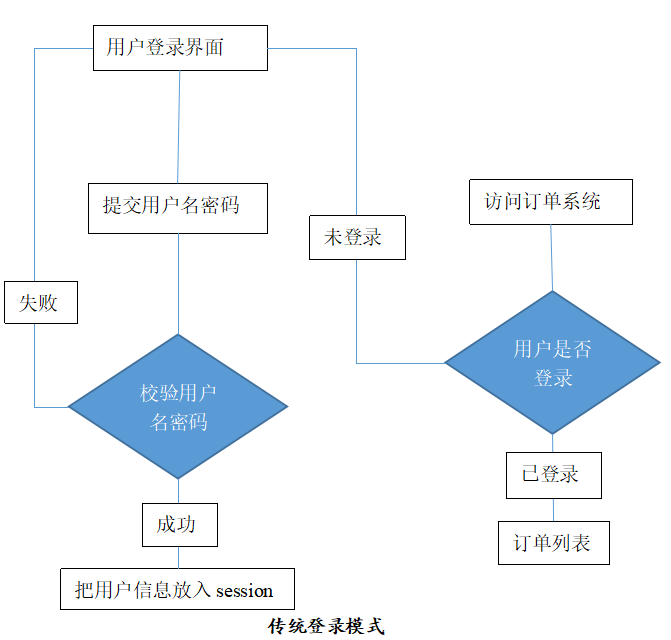
* 创建一个单点登录系统
* 发布服务
* 报表平台、后台管理系统调用单点登录系统

session共享

SSO英文全称Single Sign On，单点登录。SSO是在多个应用系统中，用户只需要登录一次就可以访问所有相互信任的应用系统。它包括可以将这次主要的登录映射到其他应用中用于同一个用户的登录的机制。它是目前比较流行的企业业务整合的解决方案之一。

## 单点登录系统相关概念

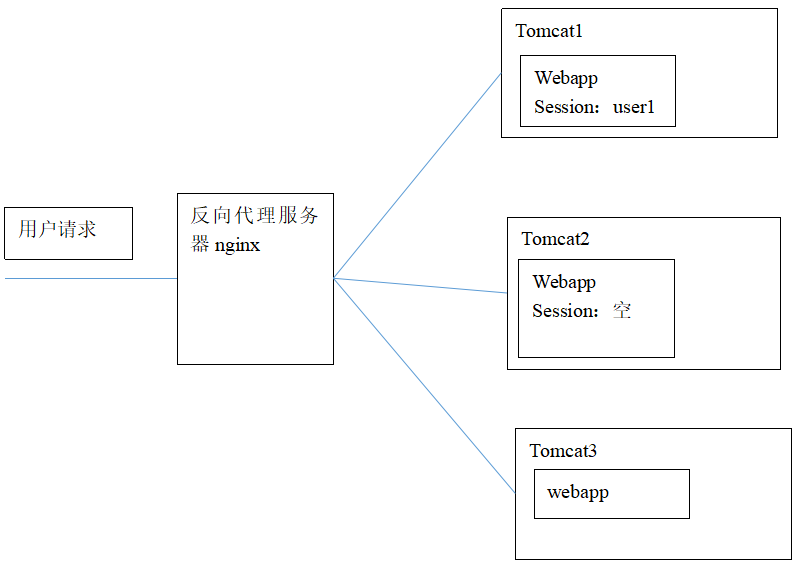
### 传统的登录流程说明



### 并发量高的情况下

2000~3000并发

一个tomcat 不能满足业务需求，需要做集群。

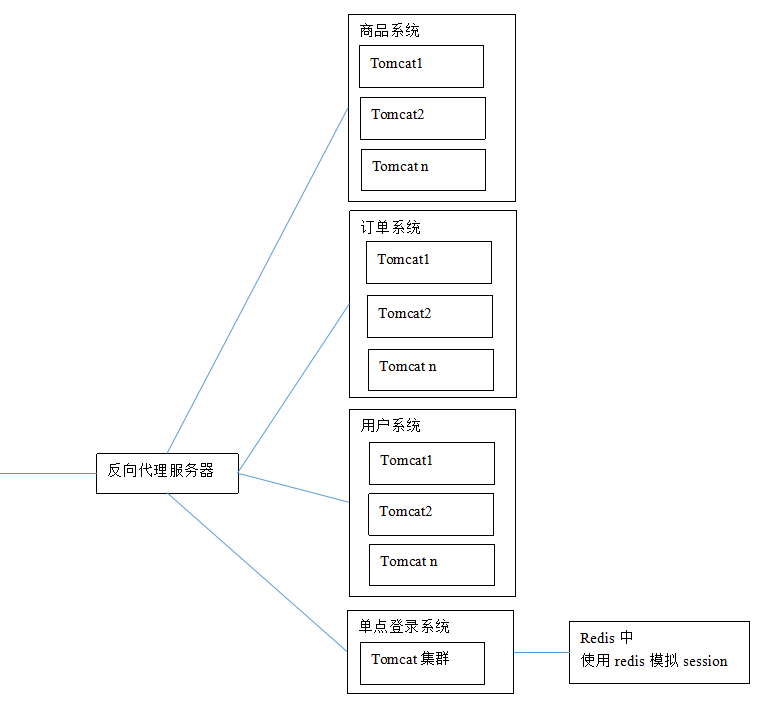


Session共享的问题：

tomcat做集群配置session复制。如果集群中节点很多，会形成网络风暴。推荐节点数量不要超过5个。

分布式架构。拆分成多个子系统。

### 分布式架构



## 单点登录系统处理流程



## 系统架构



### 问题分析

Cookie缓存问题：

在实际的应用场景中，在客户端只能存储一个用户信息相关的cookie，再次登录的用户会将之前的用户挤掉

否则出现多个不同角色用户混乱切换的情况

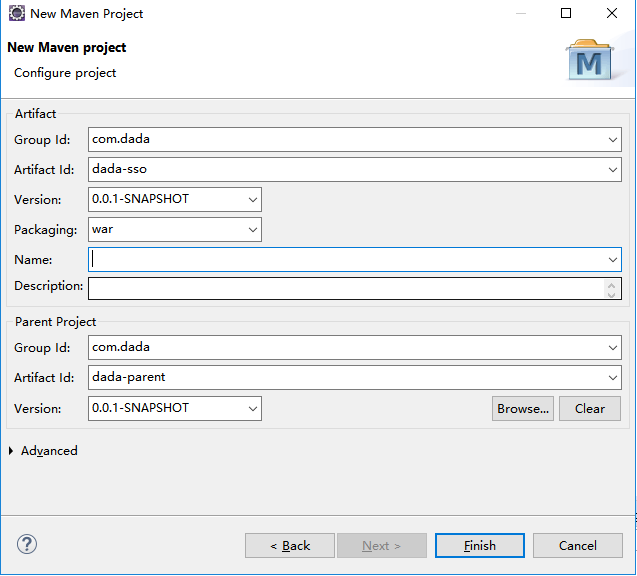
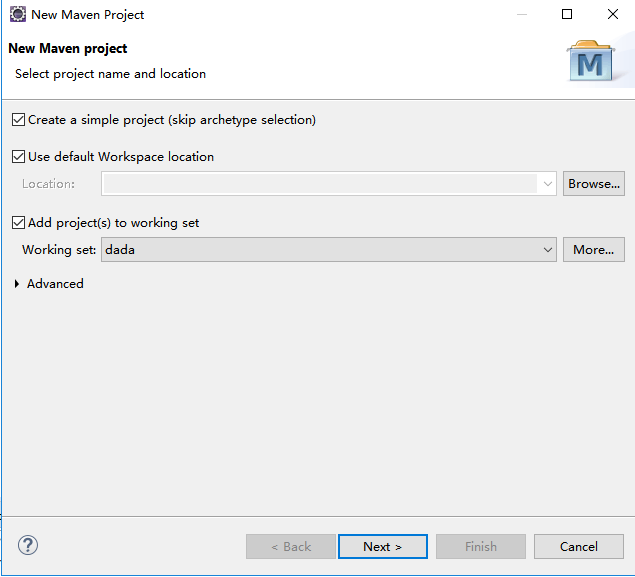
针对重复登录的账号信息则考虑是否需要注销当前的页面操作？

## 创建单点登录系统

### 创建工程

使用的技术：

1. Mybatis
2. Spring
3. Springmvc
4. Jedis（redis缓存待定）



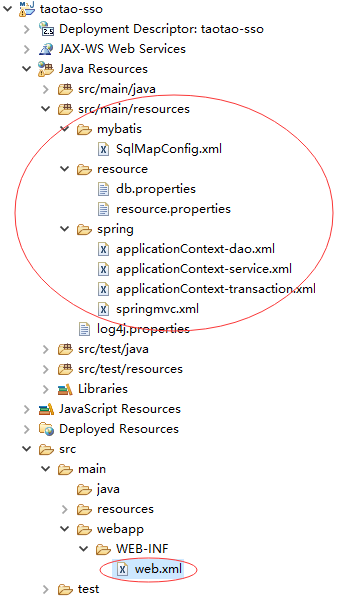
### 依赖的jar包

Pom文件

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* |

### 框架整合

整合ssm，具体配置可以参考dada-rest工程。



需要修改的地方内容说明：

SqlMapConfig.xml（mapper配置）、db.properties（数据库连接配置）、resource.properties（引用变量配置）、

Springmvc整合文件配置：applicationContext-dao.xml、applicationContext-service.xml、applicationContext-transaction.xml、springmvc.xml

## 单点登录系统的实现

登录、注册（数据验证）

### 注册接口的实现

#### 1.数据校验接口(待定)

##### 分析

要查询的表tb\_user。根据查询条件到数据库中进行查询，如果查询到结果返回false，查询结果为空返回true。使用taotaoResult包装一下返回。需要支持jsonp，根据callback判断。

##### Dao层

可以使用逆向工程。

##### Service层

接收两个参数，一个是要校验的数据，一个是数据类型。根据不同的数据类型生成不同的查询条件，到tb\_user中进行查询如果查询到结果返回false，查询结果为空返回true。

参数：

String param

Int type

返回值：

TaotaoResult

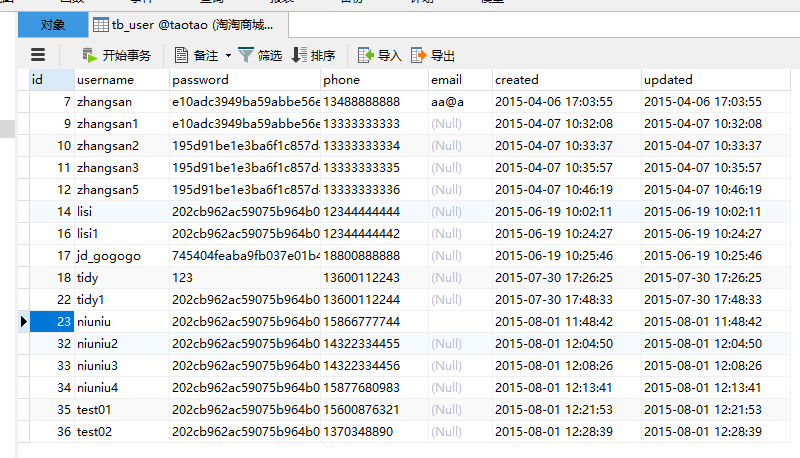
|  |
| --- |
| @Service  **public** **class** RegisterServiceImpl **implements** RegisterService {  @Autowired  **private** TbUserMapper userMapper;  @Override  **public** TaotaoResult checkData(String param, **int** type) {  //根据数据类型检查数据  TbUserExample example = **new** TbUserExample();  Criteria criteria = example.createCriteria();  //1、2、3分别代表username、phone、email  **if** (1 == type) {  criteria.andUsernameEqualTo(param);  } **else** **if** ( 2 == type) {  criteria.andPhoneEqualTo(param);  } **else** **if** ( 3 == type ) {  criteria.andEmailEqualTo(param);  }  //执行查询  List<TbUser> list = userMapper.selectByExample(example);  //判断查询结果是否为空  **if** (list == **null** || list.isEmpty()) {  **return** TaotaoResult.*ok*(**true**);  }  **return** TaotaoResult.*ok*(**false**);  }  } |

##### Controller

Controller只是发布服务。接收三个参数，一个是要校验的数据，一个数据类型，一个是callback。调用Service校验。返回json数据。需要支持jsonp，需要判断callback。

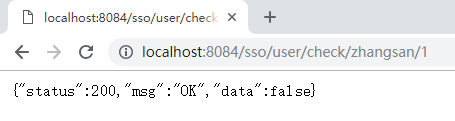
|  |
| --- |
| @Controller  @RequestMapping("/user")  **public** **class** RegisterController {  @Autowired  **private** RegisterService registerService;  @RequestMapping("/check/{param}/{type}")  @ResponseBody  **public** Object checkData(@PathVariable String param, @PathVariable Integer type, String callback) {  **try** {  TaotaoResult result = registerService.checkData(param, type);  **if** (StringUtils.*isNotBlank*(callback)) {  //请求为jsonp调用，需要支持  MappingJacksonValue mappingJacksonValue = **new** MappingJacksonValue(result);  mappingJacksonValue.setJsonpFunction(callback);  **return** mappingJacksonValue;  }  **return** result;  } **catch** (Exception e) {  e.printStackTrace();  **return** TaotaoResult.*build*(500, ExceptionUtil.*getStackTrace*(e));  }  }  } |

测试：启动taotao-sso进行测试

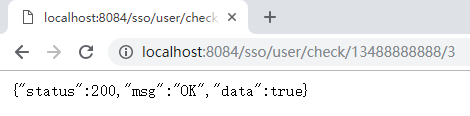


通过不同的数据测试数据校验接口，对多个字段和类型进行匹配测试

数据库中已存在相关数据则返回结果为false



数据库中不存在相关的数据则返回结果为true



#### 用户注册接口

##### 分析

使用的表user\_info，向user\_info表中插入数据。插入之前还需要进行数据校验，如果校验失败返回state为400，如果校验成功执行插入操作，完成后返回相应的结果

##### Dao层

可以使用逆向工程。

##### Service

参数：TbUser

返回值：TaotaoResult。

接收参数，对数据进行校验，校验成功，插入数据，返回结果。

|  |
| --- |
| @Override  **public** TaotaoResult register(TbUser user) {  // 校验数据  //校验用户名、密码不能为空  **if** (StringUtils.*isBlank*(user.getUsername())  || StringUtils.*isBlank*(user.getPassword())) {  **return** TaotaoResult.*build*(400, "用户名或密码不能为空");  }  //校验数据是否重复  //校验用户名  TaotaoResult result = checkData(user.getUsername(), 1);  **if** (!(**boolean**) result.getData()) {  **return** TaotaoResult.*build*(400, "用户名重复");  }  //校验手机号  **if** (user.getPhone() != **null**) {  result = checkData(user.getPhone(), 2);  **if** (!(**boolean**) result.getData()) {  **return** TaotaoResult.*build*(400, "手机号重复");  }  }  //校验手机号  **if** (user.getEmail() != **null**) {  result = checkData(user.getEmail(), 3);  **if** (!(**boolean**) result.getData()) {  **return** TaotaoResult.*build*(400, "邮箱重复");  }  }  //插入数据  user.setCreated(**new** Date());  user.setUpdated(**new** Date());  // 进行MD5密码加密  user.setPassword(DigestUtils.*md5DigestAsHex*(user.getPassword().getBytes()));  userMapper.insert(user);  **return** TaotaoResult.*ok*();  } |

##### Controller

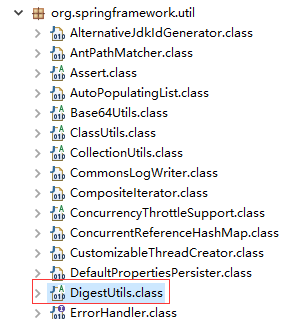
接收一个表单，请求的方法为post。使用TbUser接收表单的内容。调用Service插入数据，返回。

|  |
| --- |
| @RequestMapping(value="/register", method=RequestMethod.***POST***)  @ResponseBody  **public** TaotaoResult register(TbUser user) {  **try** {  TaotaoResult result = registerService.register(user);  **return** result;    } **catch** (Exception e) {  e.printStackTrace();  **return** TaotaoResult.*build*(500, ExceptionUtil.*getStackTrace*(e));  }  } |

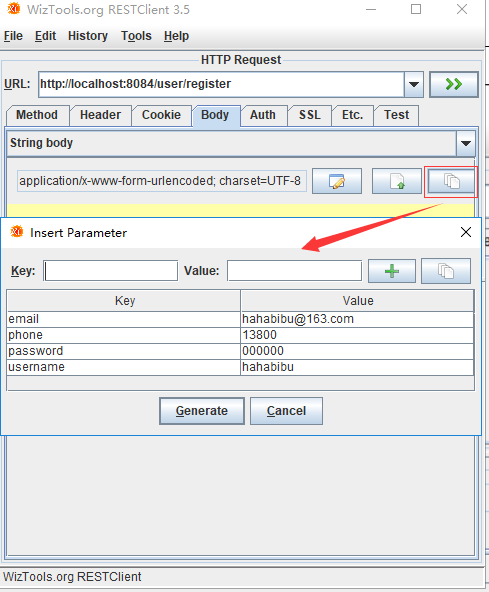
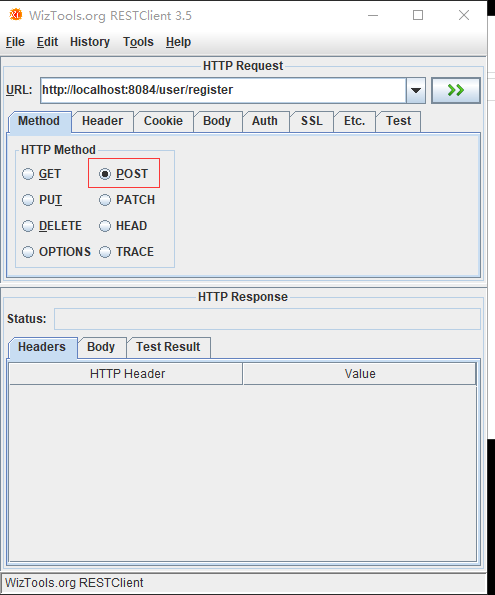
账号密码加密：spring提供了MD5加密，也可借助工具类

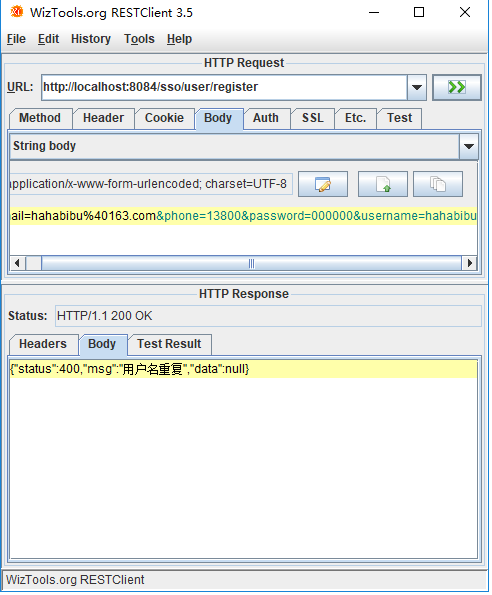
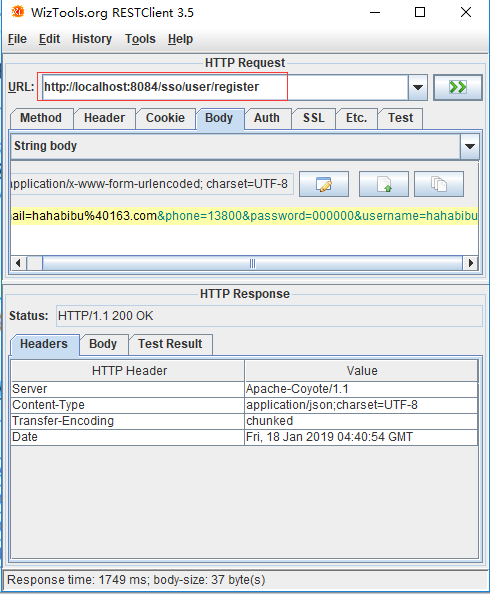


E:\repo\org\springframework\spring-core\4.1.3.RELEASE\spring-core-4.1.3.RELEASE.jar



测试接口：访问路径：http://localhost:8084/sso/user/register，重复访问测试报相应的错误





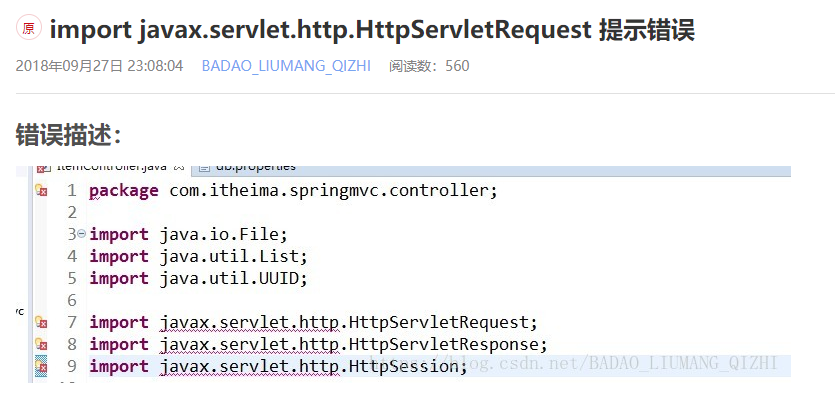
### 用户登录接口

#### 分析

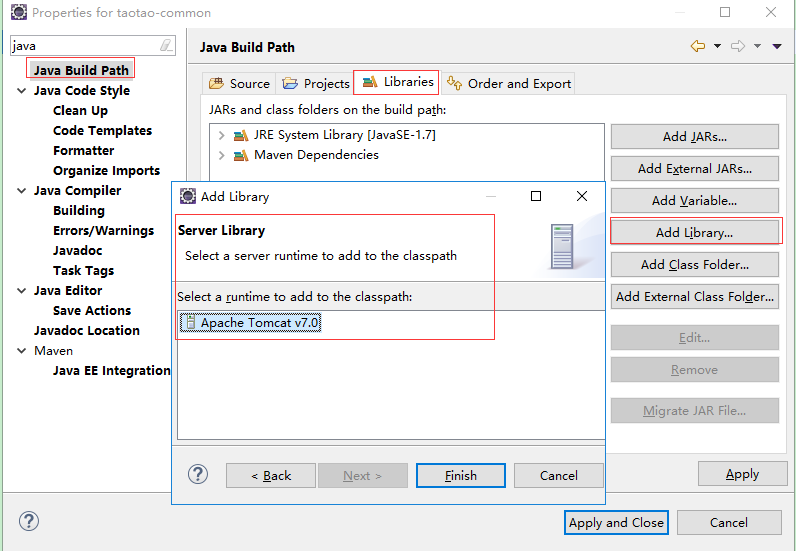
1. 接收用户名和密码
2. 校验用户名密码
3. 生成token，可以使用UUID
4. 把用户信息写入redis，key就是token
5. 把token写入cookie。
6. 返回登录成功需要把token返回给客户端。

#### 工具类参考

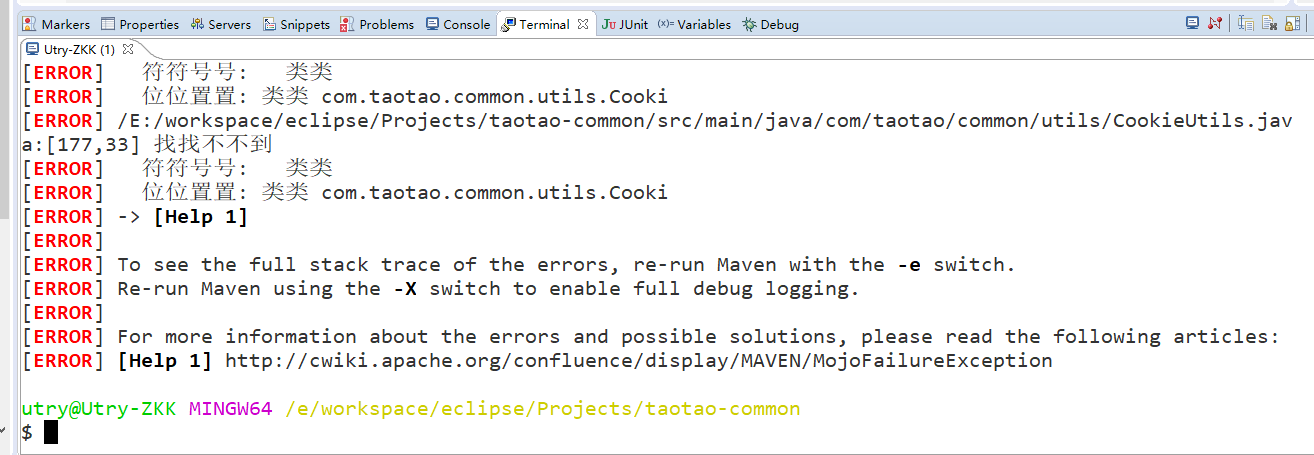
借助CookieUtils进行操作，在taotao-common下引入CookieUtils工具类，可能会出现如下问题，提供解决方案

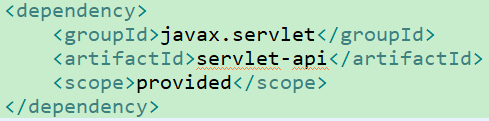


右键工程，选择Properties，搜索“Java Build Path”，在Libraries选项卡中选择“Add Library”添加Server Runtime，选择外部安装的tomcat环境，添加完成选择应用，刷新工程，问题得以解决

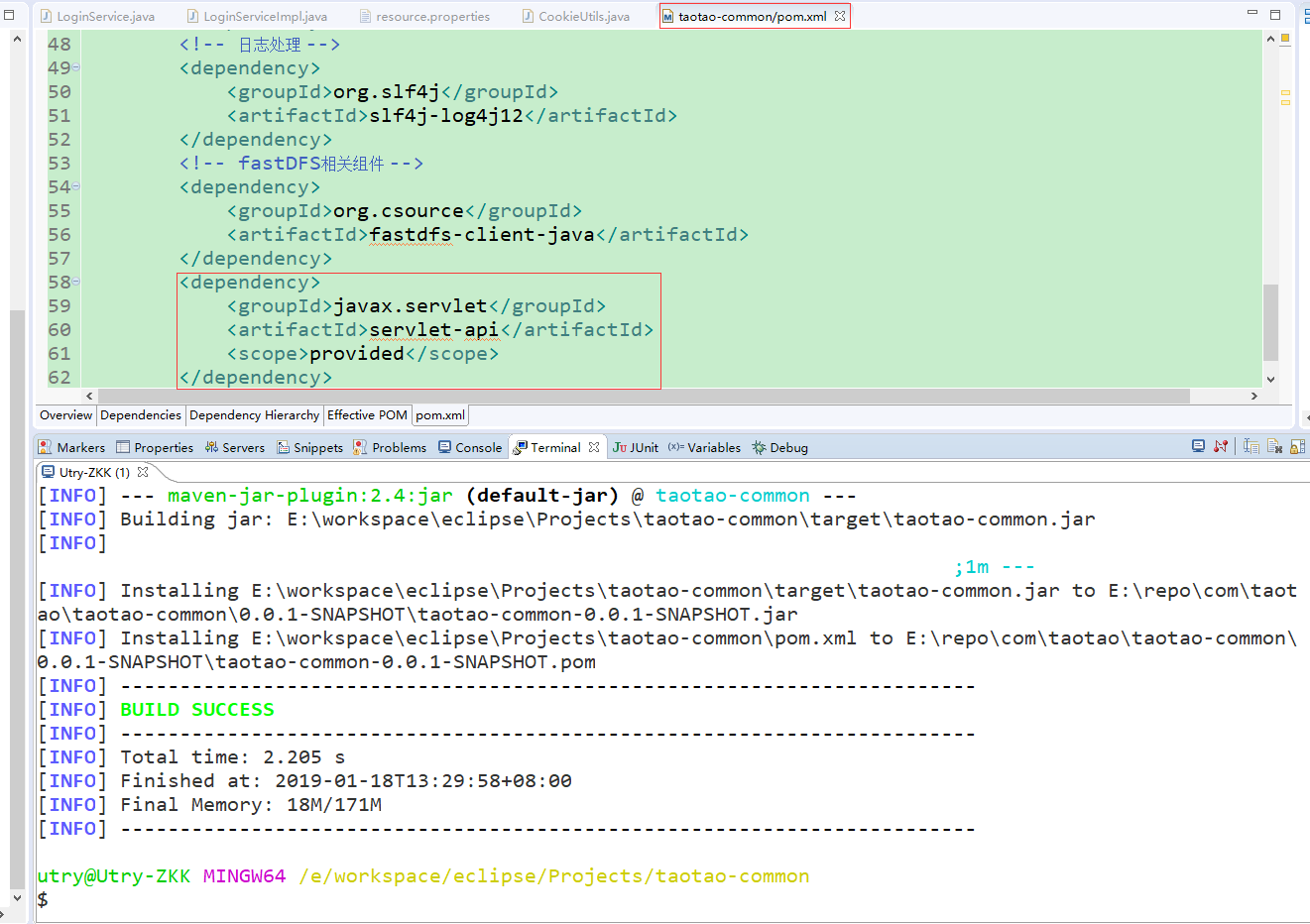


添加完成之后需要重新将taotao-common导入到本地maven仓库，存在问题，虽然能够正常导入相关包，但实际上打包的时候没有办法将相关包引入，此处需要将所需的jar加载到taotao-common工程中，即通过pom.xml添加依赖，随后再进行打包测试





依赖添加完成，再次测试，问题得以解决



#### Dao层

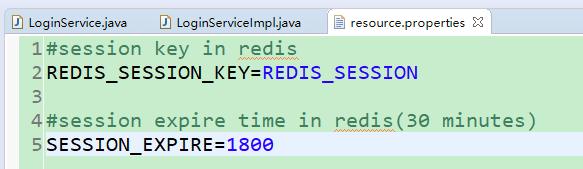
可以使用逆向工程生成的代码。

#### Service层

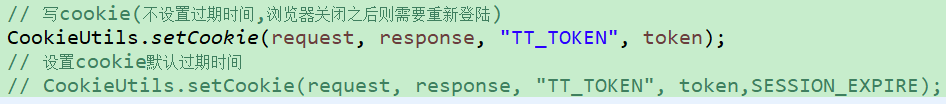
接收参数：用户名、密码。校验密码是否正确，生成token，向redis中写入用户信息，把token写入cookie，返回TaotaoResult包含token。

参数：用户名、密码、HttpServletResponse、HttpServletRequest

返回值：TaotaoResult



|  |
| --- |
| **package** com.taotao.sso.service.impl;  **import** java.util.List;  **import** java.util.UUID;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.beans.factory.annotation.Value;  **import** org.springframework.stereotype.Service;  **import** org.springframework.util.DigestUtils;  **import** com.taotao.common.result.TaotaoResult;  **import** com.taotao.common.utils.CookieUtils;  **import** com.taotao.common.utils.JsonUtils;  **import** com.taotao.mapper.TbUserMapper;  **import** com.taotao.pojo.TbUser;  **import** com.taotao.pojo.TbUserExample;  **import** com.taotao.pojo.TbUserExample.Criteria;  **import** com.taotao.sso.component.JedisClient;  **import** com.taotao.sso.service.LoginService;  @Service  **public** **class** LoginServiceImpl **implements** LoginService {  @Autowired  **private** TbUserMapper userMapper;  @Autowired  **private** JedisClient jedisClient;    @Value("${REDIS\_SESSION\_KEY}")  **private** String REDIS\_SESSION\_KEY;  @Value("${SESSION\_EXPIRE}")  **private** Integer SESSION\_EXPIRE;    @Override  **public** TaotaoResult login(String username, String password, HttpServletRequest request,  HttpServletResponse response) {  //校验用户名密码是否正确  TbUserExample example = **new** TbUserExample();  Criteria criteria = example.createCriteria();  criteria.andUsernameEqualTo(username);  List<TbUser> list = userMapper.selectByExample(example);  //取用户信息  **if** (list == **null** || list.isEmpty()) {  **return** TaotaoResult.*build*(400, "用户名或密码错误");  }  TbUser user = list.get(0);  //校验密码  **if**(!user.getPassword().equals(DigestUtils.*md5DigestAsHex*(password.getBytes()))) {  **return** TaotaoResult.*build*(400, "用户名或密码错误");  }  //登录成功  //生成token  String token = UUID.*randomUUID*().toString();  //把用户信息写入redis  //key:REDIS\_SESSION:{TOKEN}  //value:user转json  user.setPassword(**null**);  jedisClient.set(REDIS\_SESSION\_KEY + ":" + token, JsonUtils.*objectToJson*(user));  //设置session的过期时间  jedisClient.expire(REDIS\_SESSION\_KEY + ":" + token, SESSION\_EXPIRE);  //写cookie  CookieUtils.*setCookie*(request, response, "TT\_TOKEN", token);  **return** TaotaoResult.*ok*(token);  }  } |



#### Controller

请求的url：/user/login

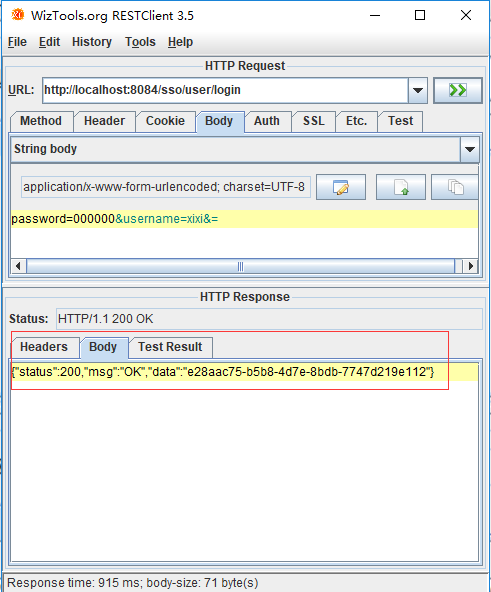
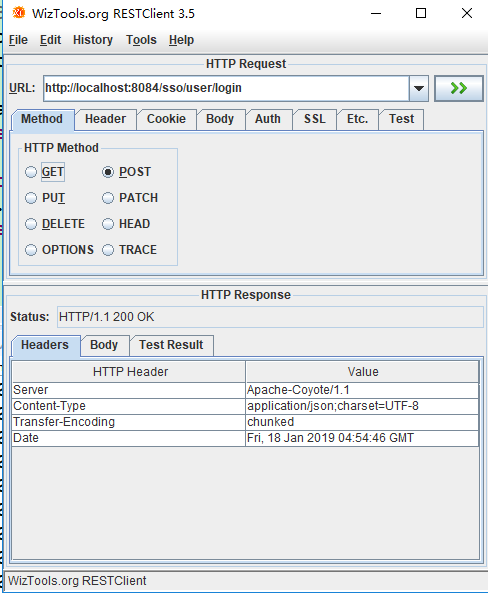
接收参数：username、password

调用Service，返回taotaoResult对象。

响应json数据。

|  |
| --- |
| **package** com.taotao.sso.controller;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RequestMethod;  **import** org.springframework.web.bind.annotation.ResponseBody;  **import** com.taotao.common.result.TaotaoResult;  **import** com.taotao.common.utils.ExceptionUtil;  **import** com.taotao.sso.service.LoginService;  @Controller  **public** **class** LoginController {  @Autowired  **private** LoginService loginService;  @RequestMapping(value="/user/login", method=RequestMethod.***POST***)  @ResponseBody  **public** TaotaoResult login(String username, String password,  HttpServletRequest request, HttpServletResponse response) {  **try** {  TaotaoResult result = loginService.login(username, password, request, response);  **return** result;  } **catch** (Exception e) {  e.printStackTrace();  **return** TaotaoResult.*build*(500, ExceptionUtil.*getStackTrace*(e));  }  }  } |

测试：



访问测试链接：

<http://localhost:8083/sso/page/login>

### 通过token查询用户信息

#### 分析

根据token到redis查询用户信息，如果用户信息不存在说明session已经过期，返回400并提示用户session已经过期。如果查询到用户，返回用户信息，并且更新一下用户的过期时间。

请求url：/sso/login/token/{token}

需要支持jsonp

返回：UserInfo

#### Dao

使用JedisClient实现

#### Service

参数：String token

根据token查询redis，查询到结果返回用户对象，更新过期时间。如果查询不到结果，返回Session已经过期，状态码400.

返回值：TaotaoResult

|  |
| --- |
| @Override  **public** TaotaoResult getUserByToken(String token) {  // 根据token取用户信息  String json = jedisClient.get(REDIS\_SESSION\_KEY + ":" + token);  //判断是否查询到结果  **if** (StringUtils.*isBlank*(json)) {  **return** TaotaoResult.*build*(400, "用户session已经过期");  }  //把json转换成java对象  TbUser user = JsonUtils.*jsonToPojo*(json, TbUser.**class**);  //更新session的过期时间  jedisClient.expire(REDIS\_SESSION\_KEY + ":" + token, SESSION\_EXPIRE);  **return** TaotaoResult.*ok*(user);  } |

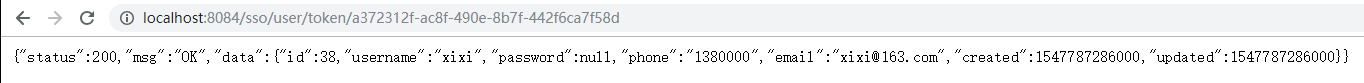
#### Controller

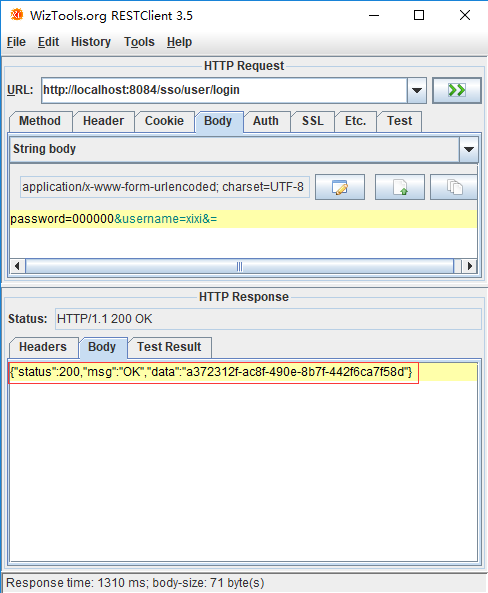
请求的url：/user/token/{token}

从url中取token的内容，调用Service取用户信息，返回TaotaoResult。（json数据）

|  |
| --- |
| @RequestMapping("/user/token/{token}")  @ResponseBody  **public** Object getUserByToken(@PathVariable String token, String callback) {  **try** {  TaotaoResult result = loginService.getUserByToken(token);  //支持jsonp调用  **if** (StringUtils.*isNotBlank*(callback)) {  MappingJacksonValue mappingJacksonValue = **new** MappingJacksonValue(result);  mappingJacksonValue.setJsonpFunction(callback);  **return** mappingJacksonValue;  }  **return** result;    } **catch** (Exception e) {  e.printStackTrace();  **return** TaotaoResult.*build*(500, ExceptionUtil.*getStackTrace*(e));  }  } |

测试：先随便登录一个账号信息（http://localhost:8083/sso/page/login），随后通过<http://localhost:8083/sso/login/token/2>访问测试





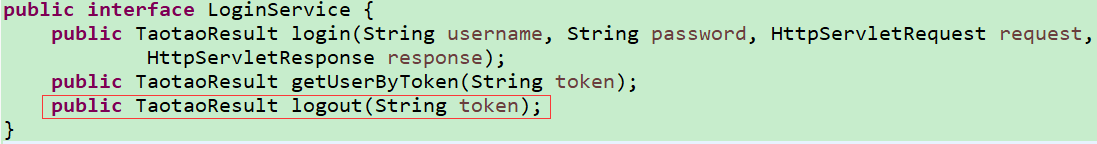
### 安全退出

接收token，删除token对应的数据。返回ok。

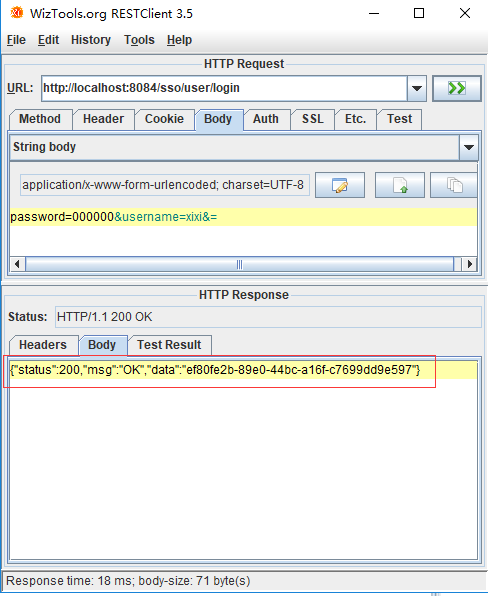
#### Dao层

参考mybatis逆向工程生成的内容

#### Service层



#### Controller层

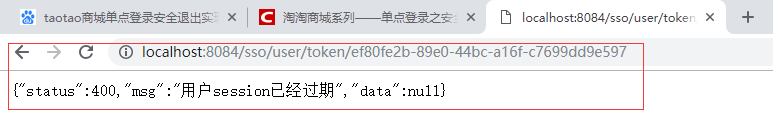


测试说明，此处先随机使用已有账号信息模拟登录，随后获取到相应的token，调用http://localhost:8084/sso/user/logout/ef80fe2b-89e0-44bc-a16f-c7699dd9e597模拟登出操作



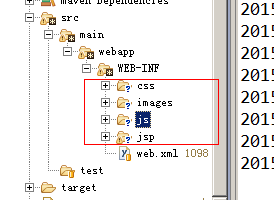
执行成功之后再次通过获取token，查看当前登录信息是否有效

http://localhost:8084/sso/user/token/ef80fe2b-89e0-44bc-a16f-c7699dd9e597



## 注册功能实现

把静态页面添加到sso系统。



Springmvc设置拦截，需要配置资源映射

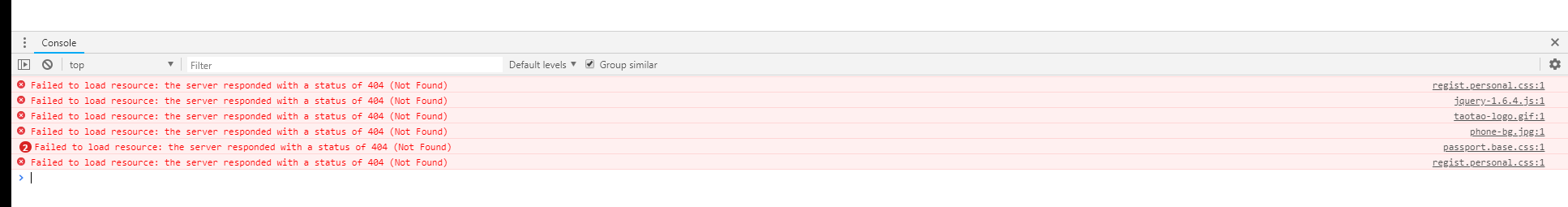


### 展示注册和登录页面

#### Controller

|  |
| --- |
| **package** com.taotao.sso.controller;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  @Controller  **public** **class** PageController {  /\*\*  \* 展示登录页面  \*/  @RequestMapping("/page/login")  **public** String showLogin() {  **return** "login";  }  /\*\*  \* 展示注册页面  \*/  @RequestMapping("/page/register")  **public** String showRegister() {  **return** "register";  }  } |

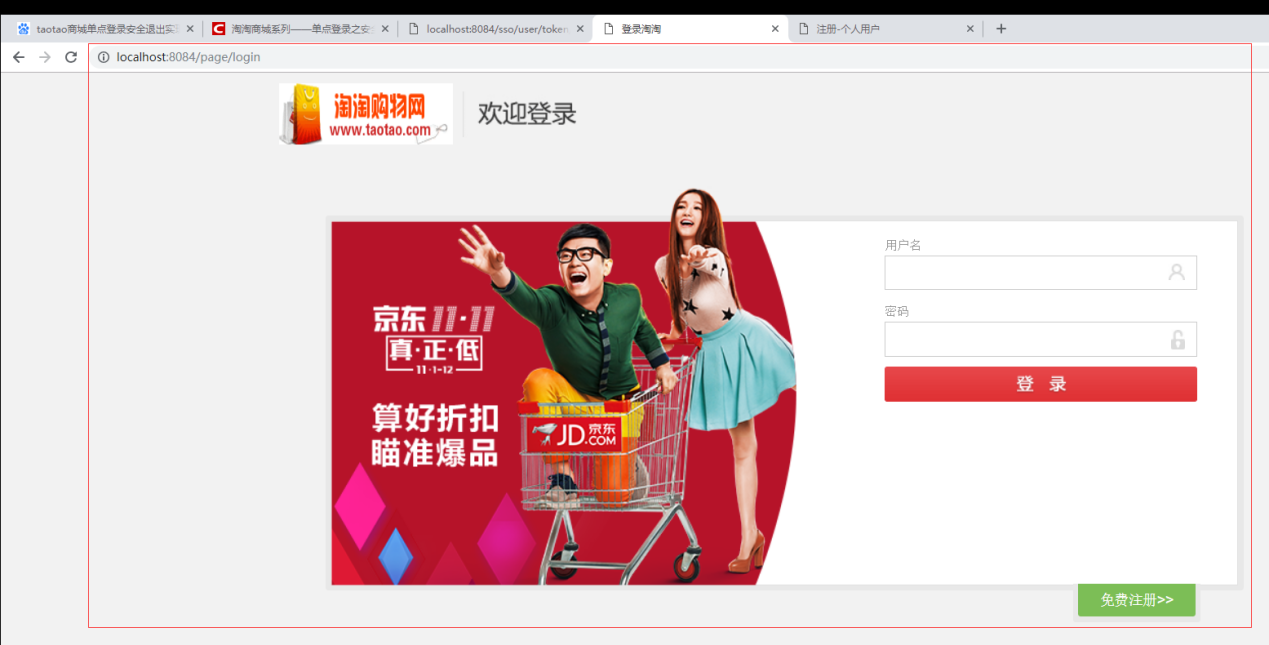
测试：访问链接，出现页面访问css样式失效的问题

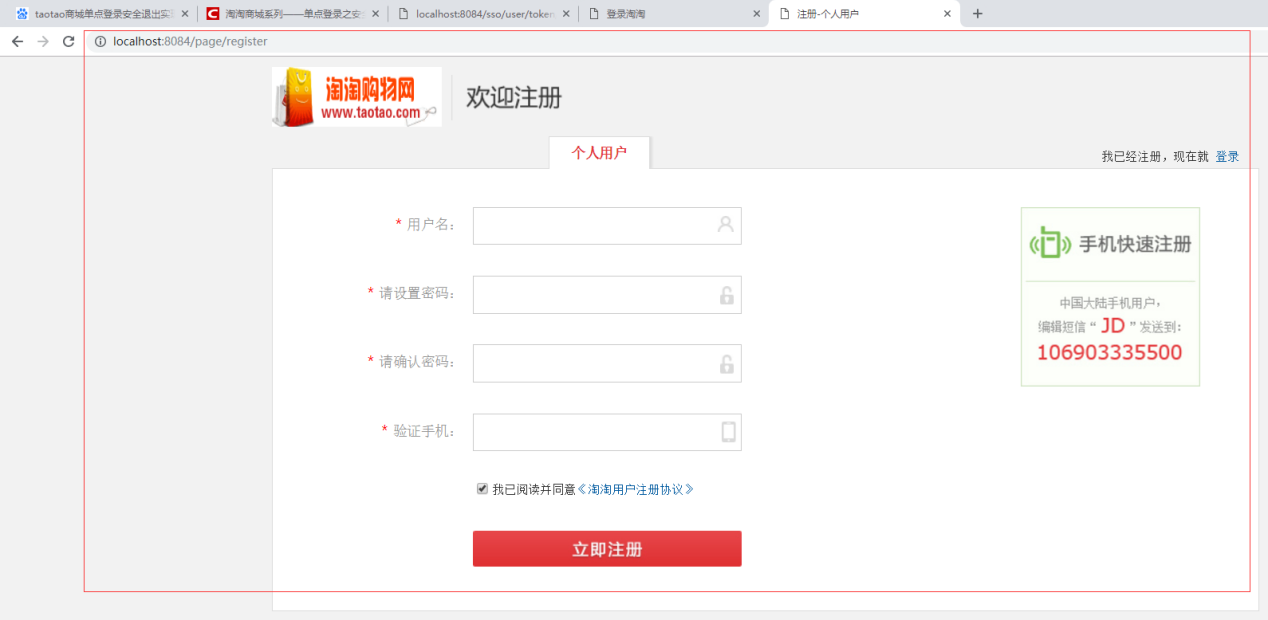


考虑可能是拦截配置的问题导致出错：此处设置taotao-sso配置拦截url的请求，并在springmvc.xml中配置静态资源映射，访问测试，数据正常显示：

此处做了修改，相应其他controller路径访问也需要做相应的变动！





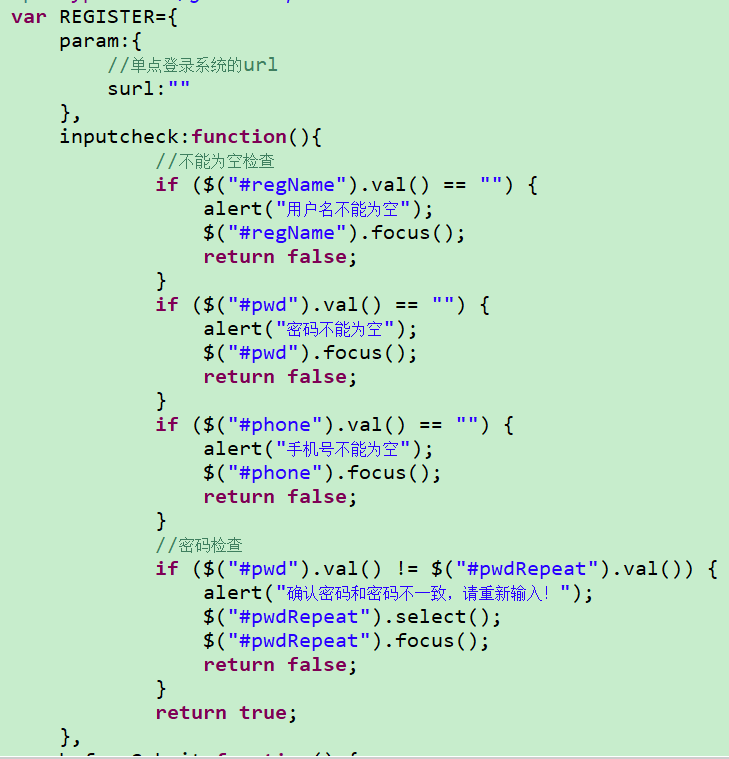


### 用户注册分析

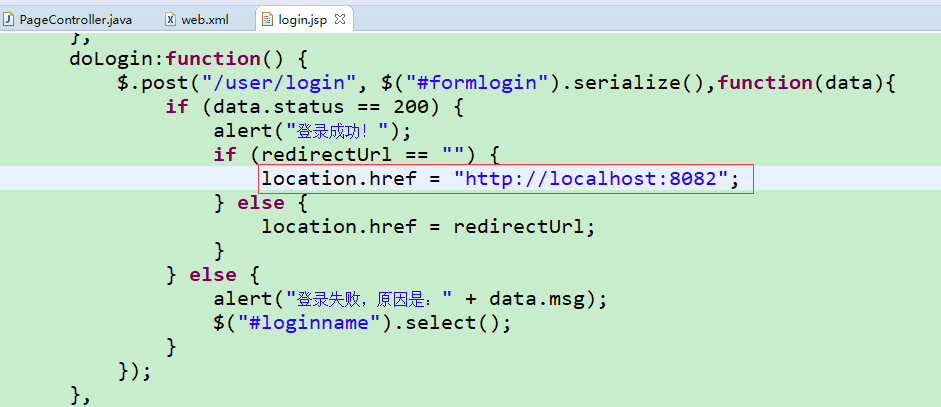
1. 用户页面输入注册信息，要对数据的有效性进行检查。可以调用sso系统的接口判断数据的有效性。
2. 提交表单，提交到用户注册接口。

**参考jsp实现。**

数据注册校验：

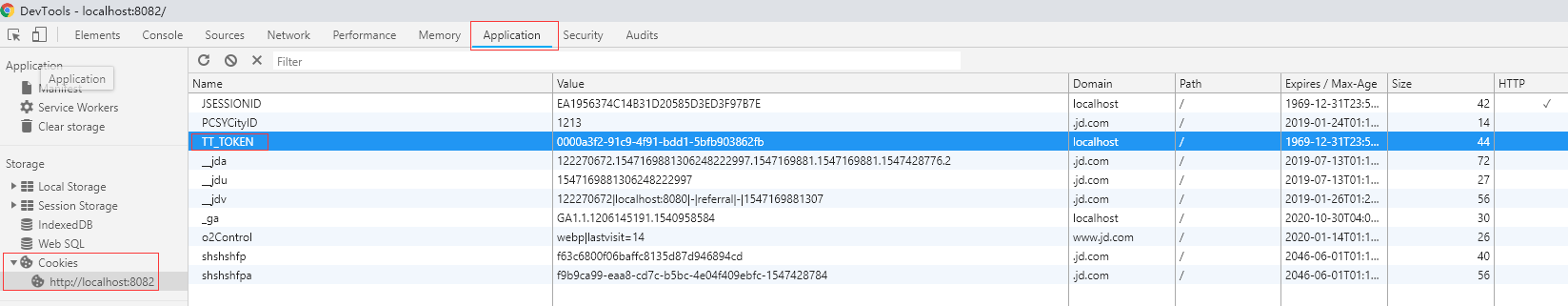


登录：

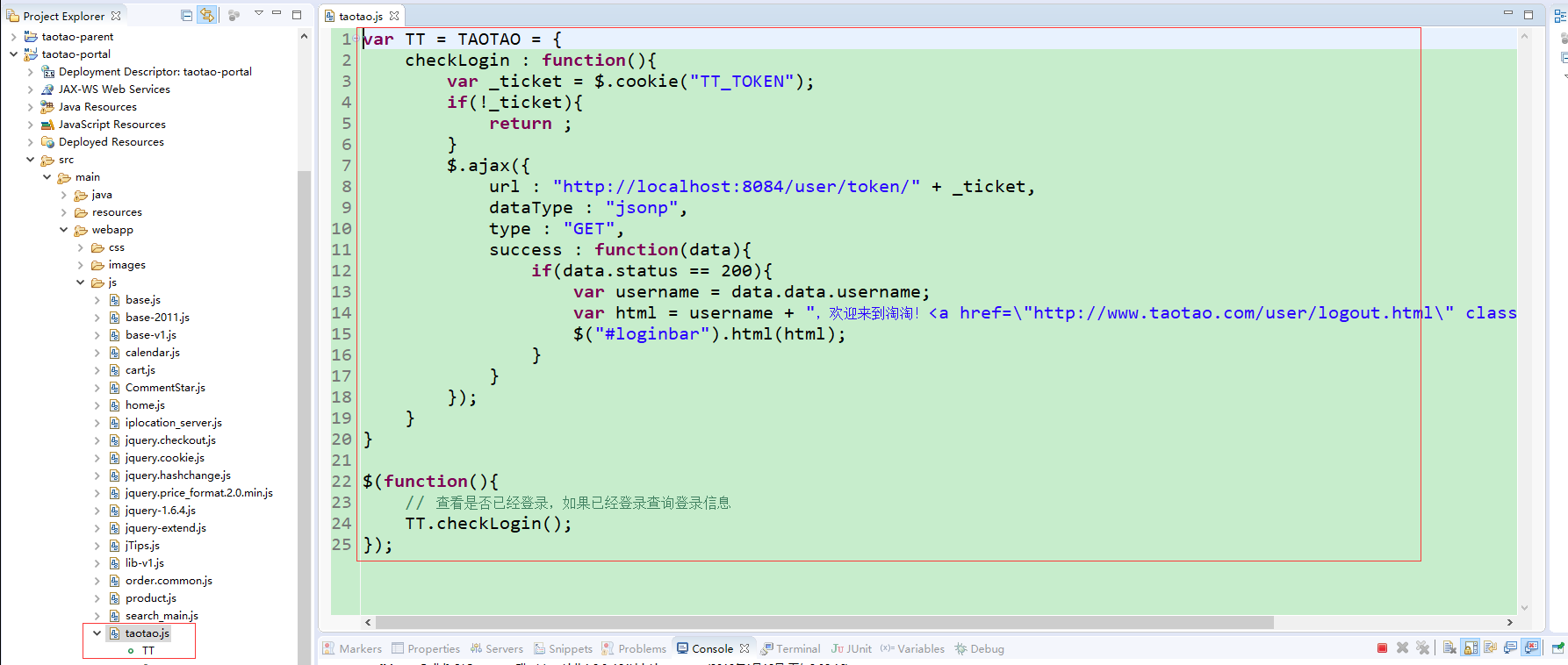


## dada-portal、dada-manager整合sso

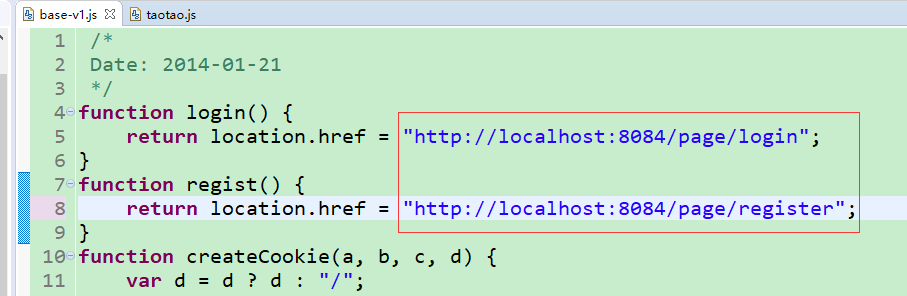
当用户在首页点击登录或者注册的时候需要跳转到sso系统。进行相应的操作。登录成功跳转到首页。首页应该显示当前登录的用户。



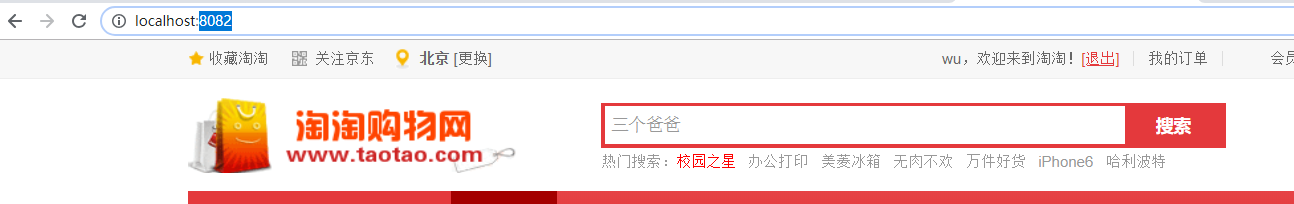
### 首页展示用户



启动taotao-rest、taotao-portal、taotao-sso测试登录、注册，查看存储的数据信息







参考：

https://blog.csdn.net/qq\_40369829/article/details/80024973

## 登录拦截器

### 分析

当用户提交订单时此时必须要求用户登录，可以使用拦截器来实现。

拦截器的处理流程：

1. 拦截请求url
2. 从cookie中取token
3. 如果没有toke跳转到登录页面。
4. 取到token，需要调用sso系统的服务查询用户信息。
5. 如果用户session已经过期，跳转到登录页面
6. 如果没有过期，放行。

### Service

根据token取用户信息，如果取到返回TbUser对象，如果取不到，返回null。

参数：Request、Response

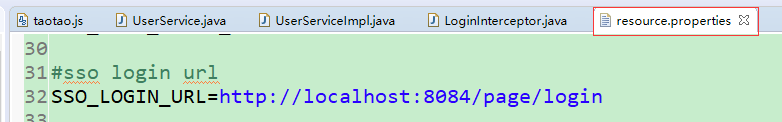
返回值：TbUser



|  |
| --- |
| **package** com.taotao.portal.service.impl;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** org.apache.commons.lang3.StringUtils;  **import** org.springframework.beans.factory.annotation.Value;  **import** org.springframework.stereotype.Service;  **import** com.taotao.common.pojo.HttpClientUtil;  **import** com.taotao.common.result.TaotaoResult;  **import** com.taotao.common.utils.CookieUtils;  **import** com.taotao.pojo.TbUser;  **import** com.taotao.portal.service.UserService;  @Service  **public** **class** UserServiceImpl **implements** UserService {  @Value("${SSO\_BASE\_URL}")  **private** String SSO\_BASE\_URL;  @Value("${SSO\_USER\_TOKEN\_SERVICE}")  **private** String SSO\_USER\_TOKEN\_SERVICE;  @Override  **public** TbUser getUserByToken(HttpServletRequest request, HttpServletResponse response) {  **try** {  //从cookie中取token  String token = CookieUtils.*getCookieValue*(request, "TT\_TOKEN");  //判断token是否有值  **if** (StringUtils.*isBlank*(token)) {  **return** **null**;  }  //调用sso的服务查询用户信息  String json = HttpClientUtil.*doGet*(SSO\_BASE\_URL + SSO\_USER\_TOKEN\_SERVICE + token);  //把json转换成java对象  TaotaoResult result = TaotaoResult.*format*(json);  **if** (result.getStatus() != 200) {  **return** **null**;  }  //取用户对象  result = TaotaoResult.*formatToPojo*(json, TbUser.**class**);  TbUser user = (TbUser) result.getData();  **return** user;  } **catch** (Exception e) {  e.printStackTrace();  **return** **null**;  }  }  } |

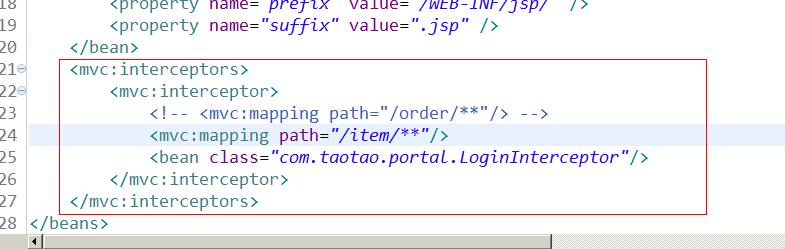
### 拦截器

在springmvc中需要实现HandlerInterceptor接口。



|  |
| --- |
| **package** com.taotao.portal.interceptor;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.beans.factory.annotation.Value;  **import** org.springframework.web.servlet.HandlerInterceptor;  **import** org.springframework.web.servlet.ModelAndView;  **import** com.taotao.pojo.TbUser;  **import** com.taotao.portal.service.UserService;  **public** **class** LoginInterceptor **implements** HandlerInterceptor {  @Autowired  **private** UserService userService;  @Value("${SSO\_LOGIN\_URL}")  **private** String SSO\_LOGIN\_URL;  @Override  **public** **boolean** preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)  **throws** Exception {  // 1、拦截请求url  // 2、从cookie中取token  // 3、如果没有toke跳转到登录页面。  // 4、取到token，需要调用sso系统的服务查询用户信息。  TbUser user = userService.getUserByToken(request, response);  // 5、如果用户session已经过期，跳转到登录页面  **if** (user == **null**) {  response.sendRedirect(SSO\_LOGIN\_URL);  **return** **false**;  }  // 6、如果没有过期，放行。  **return** **true**;  }  @Override  **public** **void** postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,  ModelAndView modelAndView) **throws** Exception {  }  @Override  **public** **void** afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex)  **throws** Exception {  }  } |

### Springmvc中配置拦截器



**访问的时候出现问题，说明拦截器属性没有被扫描到，考虑父子容器的关系，需要在springmvc.xml中配置扫描属性文件**

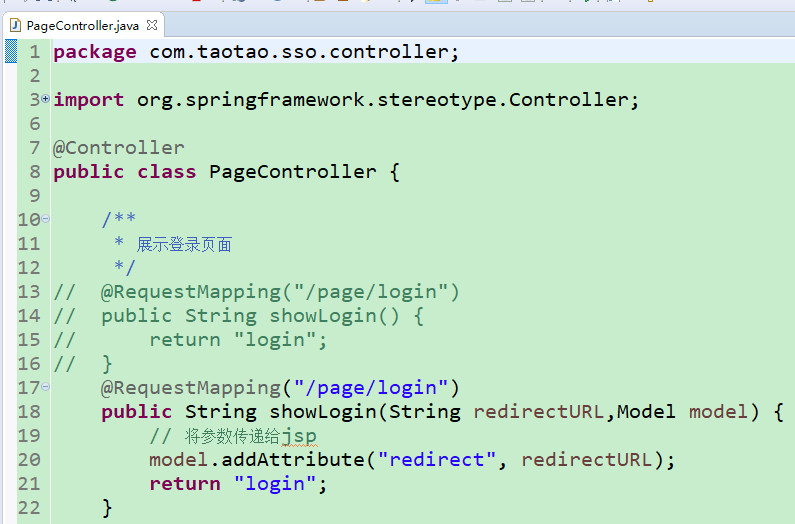




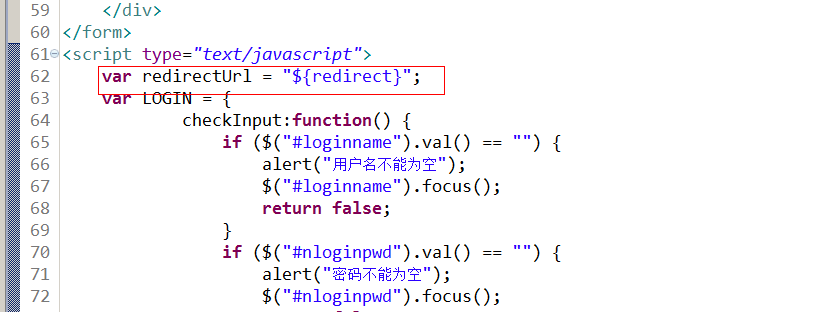
## 处理登录回调

登录的url中包含回调的参数，参数就是回调的url，sso登录Controller接收参数，当登录成功后跳转到回调的url

### 登录页面Controller



### Jsp分析





### 拦截器设置



首次未登录直接访问商品详情页面被拦截，需要强制登录才能操作，登录完成之后返回被拦截的url



## 问题分析

Json数据转换异常问题：



日期你注解问题：

**import** com.fasterxml.jackson.annotation.JsonFormat;

在实体类上添加注解

@JsonFormat(pattern = "yyyy-MM-dd hh:mm:ss", timezone = "GMT+8")

fastjson相关jar包引入：

<dependency>    
    <groupId>net.sf.json-lib</groupId>    
    <artifactId>json-lib</artifactId>    
    <version>2.4</version>    
    <classifier>jdk15</classifier>    
</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

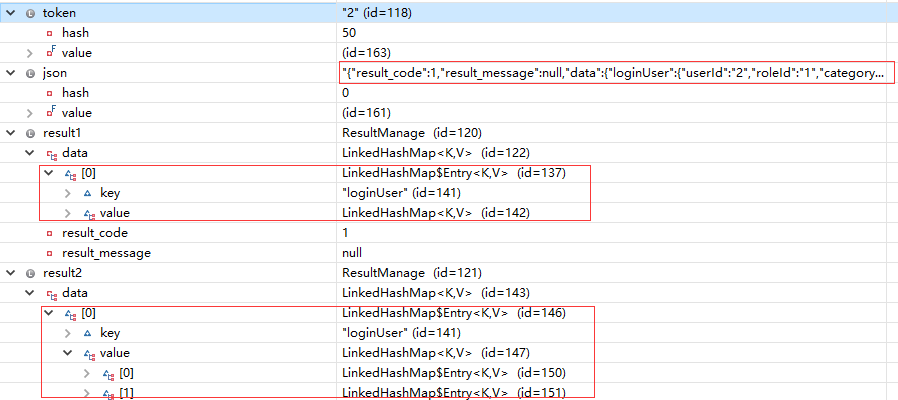
<version>1.2.56</version>

</dependency>

ResultManage result1 = ResultManage.*format*(json);

ResultManage result2= JsonUtils.*jsonToPojo*(json, ResultManage.**class**);

ResultManage result3 = JSON.*parseObject*(json, ResultManage.**class**);



ResultManage result = JSON.*parseObject*(json, ResultManage.**class**);

Map<String, Object> resultMap = result.getData();

UserInfo userInfo = JSON.*toJavaObject*((JSONObject)resultMap.get("loginUser"),UserInfo.**class**);

使用fastjson处理数据，与jackjson、自定义的JsonUtils处理的数据返回结果不同，因此在转化的时候报类型转化错误，且由JSONObject转化为相应的实体时也要遵守相应的规则，否则包类型转化错误或其他问题（JSONObject类型为com.alibaba.fastjson.JSONObject;）

