淘淘商城第十天

# 内容回顾

展示商品详情页面

1. 第一种方式：使用jsp展示。
   1. 商品基本信息(正常加载)
   2. 商品描述（延迟加载）
   3. 商品规格参数（按需加载）
   4. 调用服务获得的数据。
2. 第二种方式：网页静态化使用freemarker
   1. 创建一个模板，类似于jsp的东西
   2. 创建一个数据集把模板需要的数据准备好。
   3. Freemarker加载模板根据数据集生成html
   4. 使用nginx访问html

# 课程计划

1. 单点登录系统开发
   1. 创建一个单点登录系统
   2. 发布服务
   3. 前台系统调用单点登录系统。
2. session共享

# 单点登录系统

## 什么是单点登录系统

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。

## 为什么要有单点登录系统

### 传统的登录

已登录

订单列表

未登录

用户是否登录

访问订单系统

失败

成功

把用户信息放入session

提交用户名密码

校验用户名密码

用户登录界面

### 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。

Tomcat2

Webapp

Session：空

Tomcat1

Webapp

Session：user1

用户请求

反向代理服务器nginx

webapp

Tomcat3

Session共享的问题：

1. tomcat做集群配置session复制。这种是采用广播的方式给其他的服务器发送广播，所有服务器之间都要进行广播来同步session数据，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。
2. 分布式架构。拆分成多个子系统。

### 分布式架构

Redis中

使用redis模拟session

反向代理服务器

Tomcat集群

单点登录系统

Tomcat n

Tomcat2

Tomcat1

用户系统

订单系统

Tomcat1

Tomcat2

Tomcat n

Tomcat n

Tomcat2

Tomcat1

商品系统

## 单点登录系统处理流程



## 系统架构

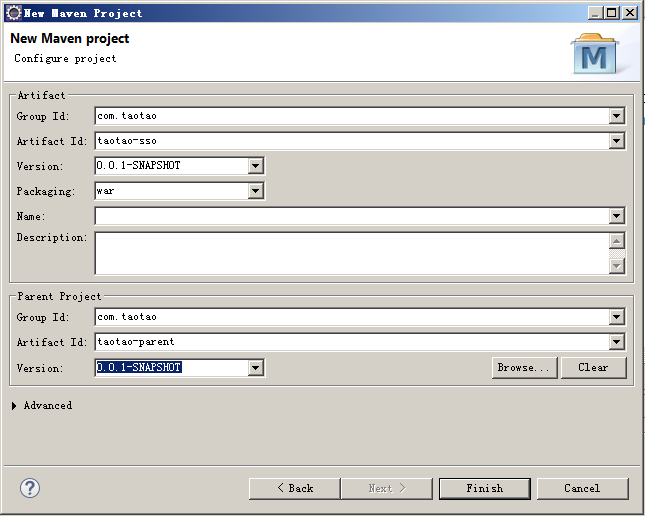


# 创建单点登录系统

## 创建工程

使用的技术：

1. Mybatis
2. Spring
3. Springmvc
4. Jedis



## 依赖的jar包

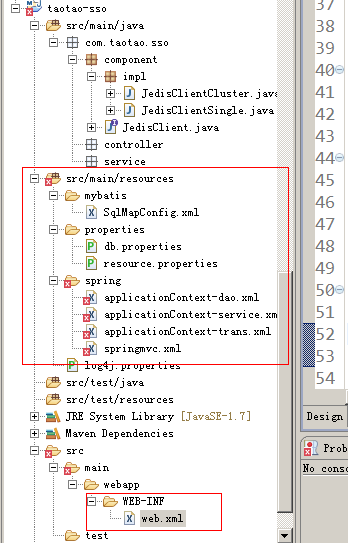
Pom文件

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.taotao</groupId>  <artifactId>taotao-parent</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <groupId>com.taotao</groupId>  <artifactId>taotao-sso</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>war</packaging>  <dependencies>  <dependency>  <groupId>com.taotao</groupId>  <artifactId>taotao-manager-dao</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  <!-- Spring -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-beans</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aspects</artifactId>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context-support</artifactId>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>servlet-api</artifactId>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jsp-api</artifactId>  <scope>provided</scope>  </dependency>  <!-- Redis客户端 -->  <dependency>  <groupId>redis.clients</groupId>  <artifactId>jedis</artifactId>  </dependency>  </dependencies>  <!-- 添加tomcat插件 -->  <build>  <plugins>  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <configuration>  <port>8084</port>  <path>/</path>  </configuration>  </plugin>  </plugins>  </build>  </project> |

## 框架整合

整合ssm。

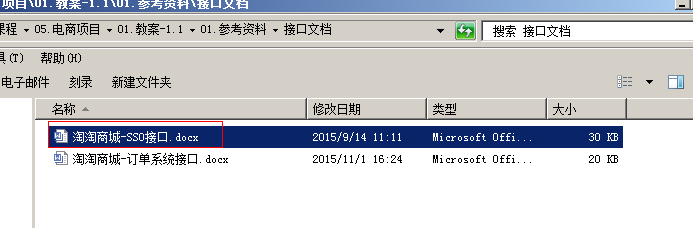
可以参考taotao-rest工程。



# 单点登录系统的实现

## 接口文档

注册功能、登录功能、查询功能。



## 注册接口的实现

### 数据校验接口

#### 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));  }  }    } |

### 用户注册接口

#### 分析

使用的表tb\_user，向tb\_user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TaotaoResult.ok

#### 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。

|  |
| --- |
| /\*\*  \* 用户注册  \* <p>Title: register</p>  \* <p>Description: </p>  \* **@param** user  \* **@return**  \* **@see** com.taotao.sso.service.RegisterService#register(com.taotao.pojo.TbUser)  \*/  @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());  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));  }  } |

## 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。

### 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

|  |
| --- |
| @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。

|  |
| --- |
| @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));  }  }  } |

## 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：/user/token/{token}

需要支持jsonp

返回：TaotaoResult

### 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));  }  } |

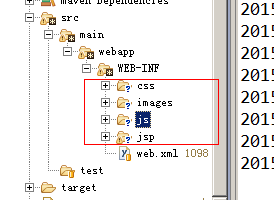
## 安全退出

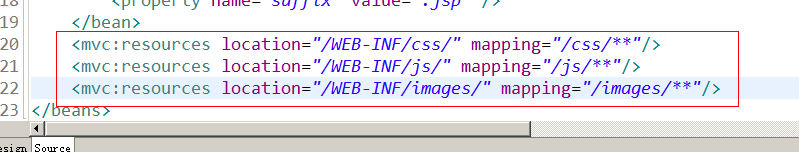
接收token，删除token对应的数据。返回ok。

作业！！！

# 注册功能实现

把静态页面添加到sso系统。





## 展示注册和登录页面

### Controller

|  |
| --- |
| @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";  }  } |

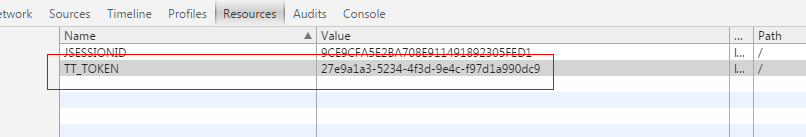
## 用户注册分析

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。

# 门户系统整合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



# 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

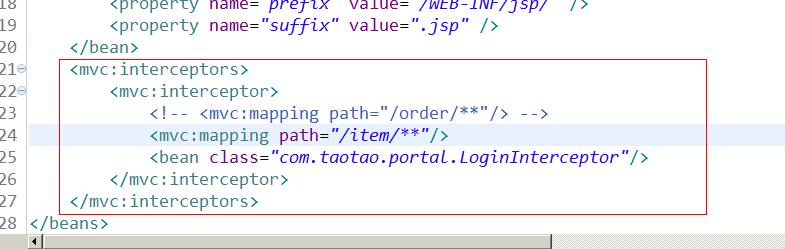
|  |
| --- |
| @Service  **public** **class** UserServiceImpl **implements** UserServic {    @Value("${SSO\_BASE\_URL}")  **private** String SSO\_BASE\_URL;  @Value("${SSO\_USER\_TOKEN\_SERVICE}")  **private** String SSO\_USER\_TOKEN\_SERVICE;    @Override  **public** TbUser getUserByToken(HttpServletRequest 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。

|  |
| --- |
| **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 {  }  } |

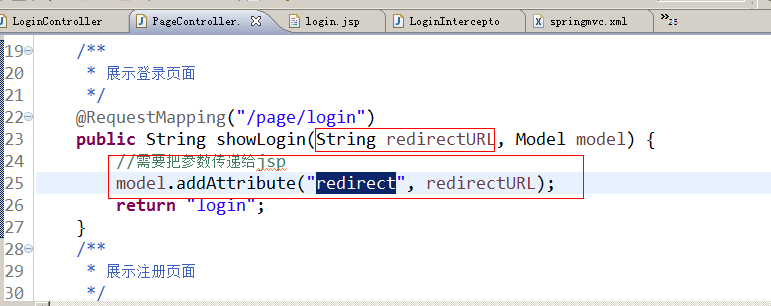
## Springmvc中配置拦截器



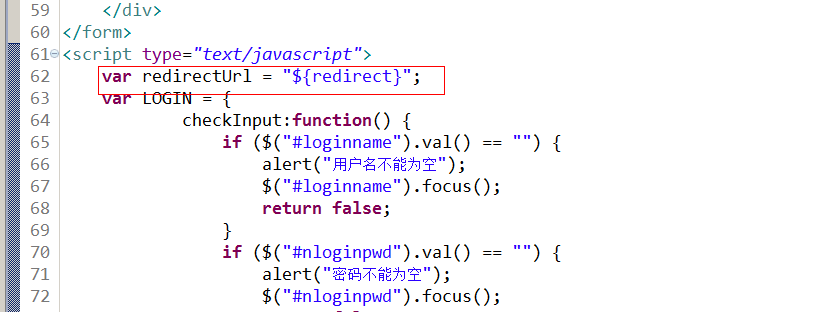
# 处理登录回调

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





## 拦截器

