# 淘淘商城-购物车、订单系统实现

### 内容回顾

1. 单点登录系统
   1. 分布式环境下session共享。
   2. Session放到redis中。
   3. 使用uuid生成一个token
   4. Token放到cookie中。
   5. 根据token或用户信息。
2. sso系统集成portal
   1. 用户登录注册功能都跳转到sso系统。
   2. 登录完成后跳转首页。
   3. 首页需要在加载完成后，需要从cookie中取token，调用sso系统的服务（jsonp）根据token查询用户信息，把用户名显示到首页。
   4. 使用拦截器实现用户强制登录。
   5. 回调页面的实现

### 课程计划

1. 购物车的实现
2. 订单系统的实现。

## 购物车

### 分析

1、添加购物车不需要用户登录。购物车的数据应该放到cookie中。

2、当向购物车添加同一款商品时，购物车中商品的数量增加。

3、购物车中可以删除商品

4、购物车中可以修改商品数量。商品的总价需要重新计算。

5、点击“结算”按钮要求用户登录。

### 添加购物车

#### 分析

请求的url：/cart/add/${itemId}.html

1、接收商品id，根据商品id取商品的图片、title、单价，需要包含商品的数量。并且把数据保存到列表中。

2、把商品数据转换成json数据，保存到cookie中。对购物车数据进行转码。

3、展示添加购物车成功页面。

#### Service

参数：

* 商品id
* 商品数量

业务逻辑：

* 接收商品id
* 从cookie中购物车商品列表
* 从商品列表中查询列表是否存在此商品
* 如果存在商品的数量加上参数中的商品数量
* 如果不存在，调用rest服务，根据商品id获得商品数据。
* 把商品数据添加到列表中
* 把购物车商品列表写入cookie
* 返回TaotaoResult

返回值：TaotaoResult

购物车商品POJO：

|  |
| --- |
| **public** **class** CartItem {  **private** Long id;  **private** String title;  **private** Long price;  **private** Integer num;  **private** String image;  } |

|  |
| --- |
| **package** com.taotao.portal.service.impl;  **import** java.util.ArrayList;  **import** java.util.List;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** org.apache.commons.lang3.StringUtils;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.beans.factory.annotation.Value;  **import** org.springframework.stereotype.Service;  **import** com.taotao.common.result.TaotaoResult;  **import** com.taotao.common.utils.CookieUtils;  **import** com.taotao.common.utils.JsonUtils;  **import** com.taotao.pojo.TbItem;  **import** com.taotao.portal.pojo.CartItem;  **import** com.taotao.portal.service.CartService;  **import** com.taotao.portal.service.ItemService;  @Service  **public** **class** CartServiceImpl **implements** CartService {  @Autowired  **private** ItemService itemService;  @Value("${COOKIE\_EXPIRE}")  **private** Integer COOKIE\_EXPIRE;  @Override  **public** TaotaoResult addCart(Long itemId, Integer num,  HttpServletRequest request, HttpServletResponse response) {  // 1、接收商品id  // 2、从cookie中购物车商品列表  List<CartItem> itemList = getCartItemList(request);  // 3、从商品列表中查询列表是否存在此商品  **boolean** haveFlg = **false**;  **for** (CartItem cartItem : itemList) {  //如果商品存在数量相加  // 4、如果存在商品的数量加上参数中的商品数量  **if** (cartItem.getId().longValue() == itemId) {  cartItem.setNum(cartItem.getNum() + num);  haveFlg = **true**;  **break**;  }  }  // 5、如果不存在，调用rest服务，根据商品id获得商品数据。  **if** (!haveFlg) {  TbItem item = itemService.getItemById(itemId);  //转换成CartItem  CartItem cartItem = **new** CartItem();  cartItem.setId(itemId);  cartItem.setNum(num);  cartItem.setPrice(item.getPrice());  cartItem.setTitle(item.getTitle());  **if** (StringUtils.*isNotBlank*(item.getImage())) {  String image = item.getImage();  String[] strings = image.split(",");  cartItem.setImage(strings[0]);  }  //添加到购物车商品列表  // 6、把商品数据添加到列表中  itemList.add(cartItem);  }  // 7、把购物车商品列表写入cookie  CookieUtils.*setCookie*(request, response, "TT\_CART", JsonUtils.*objectToJson*(itemList), COOKIE\_EXPIRE, **true**);  // 8、返回TaotaoResult  **return** TaotaoResult.*ok*();  }  /\*\*  \* 取购物车商品列表  \* <p>Title: getCartItemList</p>  \* <p>Description: </p>  \* **@param** request  \* **@return**  \*/  **private** List<CartItem> getCartItemList(HttpServletRequest request) {  **try** {  //从cookie中取商品列表  String json = CookieUtils.*getCookieValue*(request, "TT\_CART", **true**);  //把json转换成java对象  List<CartItem> list = JsonUtils.*jsonToList*(json, CartItem.**class**);  **return** list==**null**?**new** ArrayList<CartItem>():list;  } **catch** (Exception e) {  **return** **new** ArrayList<CartItem>();  }  }  } |

#### Controller

请求的url：/cart/add/{itemId}.html?num=1

接收两个参数：

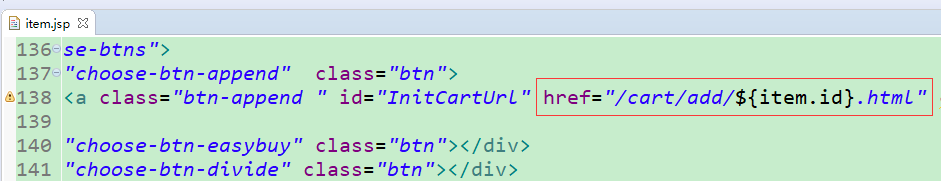
1. 商品id
2. 商品数量

调用Service添加到购物车

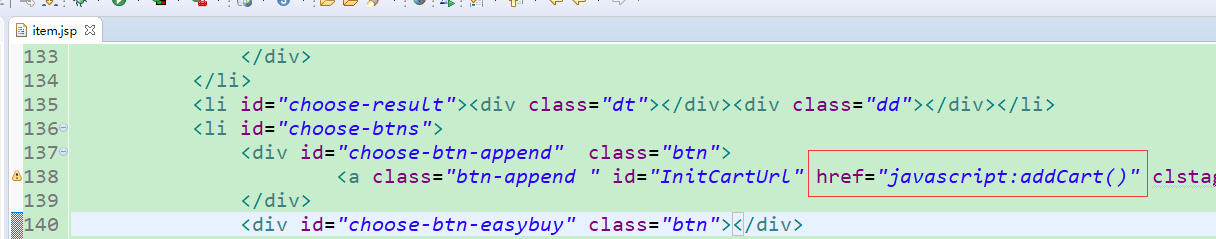
返回jsp页面，添加成功页面。

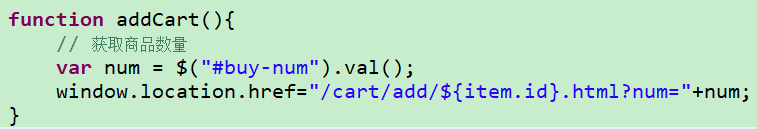
|  |
| --- |
| **package** com.taotao.portal.controller;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.PathVariable;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** com.taotao.common.result.TaotaoResult;  **import** com.taotao.portal.service.CartService;  @Controller  **public** **class** CartController {  @Autowired  **private** CartService cartService;  @RequestMapping("/cart/add/{itemId}")  **public** String addCart(@PathVariable Long itemId, Integer num,  HttpServletResponse response, HttpServletRequest request) {  TaotaoResult result = cartService.addCart(itemId, num, request, response);  **return** "cartSuccess";  }  } |

#### Jsp



修改为如下所示：





#### 测试

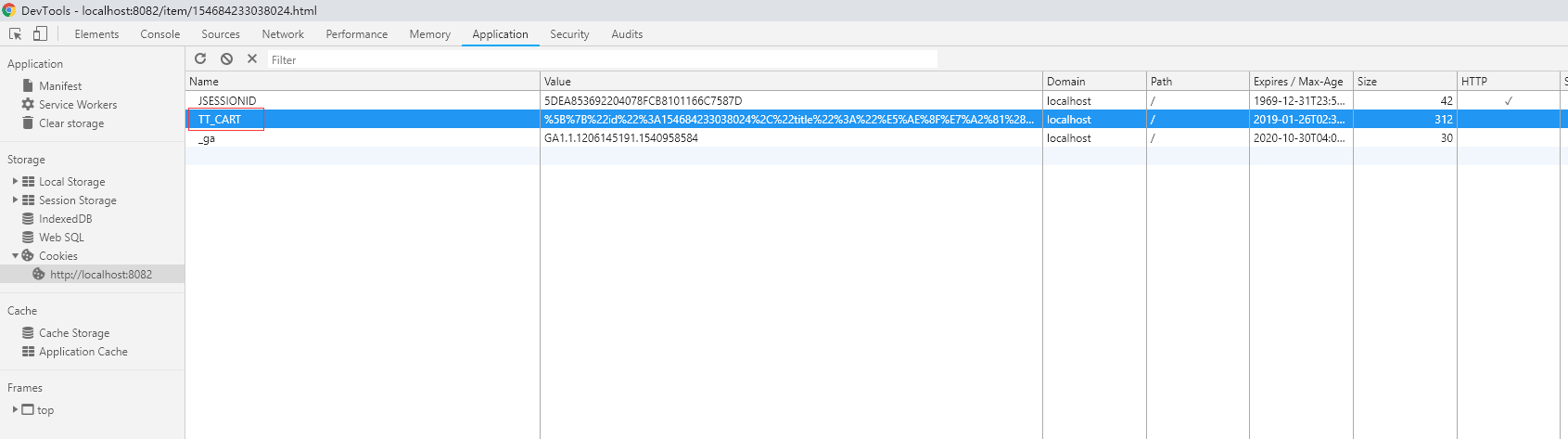
随意访问一个商品详情，点击添加入购物车，测试



选择数量，添加至购物车：



查看生成的cookies信息



### 展示购物车商品列表

#### 分析

请求的url：/cart/cart

没有参数。

从cookie中取购物车商品列表，在页面展示。

Jsp页面需要一购物车商品列表即可。

#### Service

没有参数，直接从cookie中取购物车商品列表返回。

返回值：List<CartItem>

|  |
| --- |
| Impl：    @Override  **public** List<CartItem> getCartItems(HttpServletRequest request) {  List<CartItem> list = getCartItemList(request);  **return** list;  } |

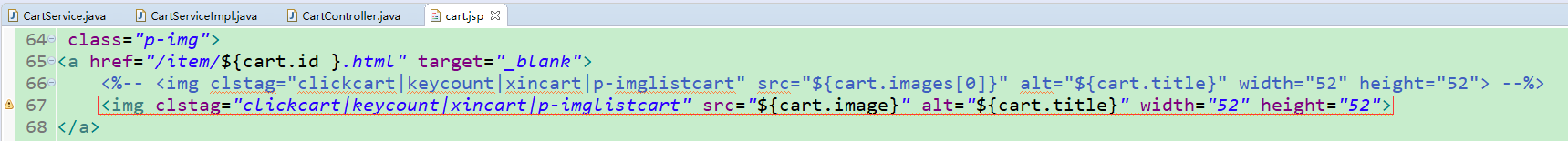
#### Controller

接收请求，请求的url：/cart/cart

调用Service取商品列表，传递给jsp。

|  |
| --- |
| @RequestMapping("/cart/cart")  **public** String showCartList(HttpServletRequest request, Model model) {  List<CartItem> list = cartService.getCartItems(request);  //把商品列表传递给jsp  model.addAttribute("cartList", list);  **return** "cart";  } |

查看购物车对应的jsp文件：cart.jsp



#### 测试



### 修改购物车商品数量

#### 分析

修改商品数量js：



请求的url：/cart/update/num/{itemId}/{num}.action

1. 把购物车商品列表从cookie中取出来
2. 根据itemId找到对应的商品
3. 使用参数中的数量更新数量。
4. 需要把购物车商品列表写回cookie。
5. 返回json数据

#### Service层

接收两个参数：

1、商品id

2、商品数量

业务逻辑：

1、把购物车商品列表从cookie中取出来

2、根据itemId找到对应的商品

3、使用参数中的数量更新数量。

4、需要把购物车商品列表写回cookie。

5、返回TaotaoResult

|  |
| --- |
| @Override  **public** TaotaoResult updateCartItem(**long** itemId, Integer num, HttpServletRequest request,  HttpServletResponse response) {  // 从cookie中取购物车商品列表  List<CartItem> itemList = getCartItemList(request);  //根据商品id查询商品  **for** (CartItem cartItem : itemList) {  **if** (cartItem.getId() == itemId) {  //更新数量  cartItem.setNum(num);  **break**;  }  }  //写入cookie  CookieUtils.*setCookie*(request, response, "TT\_CART", JsonUtils.*objectToJson*(itemList), COOKIE\_EXPIRE, **true**);  **return** TaotaoResult.*ok*();  } |

#### Controller

请求的url：/cart/update/num/{itemId}/{num}.action

从url中取两个参数，调用Service更新商品数量。

返回TaotaoResult。（返回json格式的数据）

|  |
| --- |
| @RequestMapping("/cart/update/num/{itemId}/{num}")  @ResponseBody  **public** TaotaoResult updateCartItemNum(@PathVariable Long itemId, @PathVariable Integer num,  HttpServletResponse response, HttpServletRequest request) {  TaotaoResult result = cartService.updateCartItem(itemId, num, request, response);  **return** result;  } |

#### 测试

### 删除购物车商品

#### 分析

需要的参数是商品id，根据商品id删除cookie中的商品列表中对应的商品。重新写入cookie。

返回逻辑视图，购物车页面。

#### Service

1. 接收商品id
2. 从cookie中取购物车商品列表
3. 找到对应id的商品
4. 删除商品。
5. 再重新把商品列表写入cookie。
6. 返回成功

参数：商品ID

返回值：TaotaoResult

|  |
| --- |
| @Override  **public** TaotaoResult deleteCartItem(**long** itemId, HttpServletRequest request, HttpServletResponse response) {  // 1、接收商品id  // 2、从cookie中取购物车商品列表  List<CartItem> itemList = getCartItemList(request);  // 3、找到对应id的商品  **for** (CartItem cartItem : itemList) {  **if** (cartItem.getId() == itemId) {  // 4、删除商品。  itemList.remove(cartItem);  **break**;  }  }  // 5、再重新把商品列表写入cookie。  CookieUtils.*setCookie*(request, response, "TT\_CART", JsonUtils.*objectToJson*(itemList), COOKIE\_EXPIRE, **true**);  // 6、返回成功  **return** TaotaoResult.*ok*();  } |

#### Controller

接收商品id，调用Service删除商品，返回购物车页面。

请求的url：/cart/delete/{itemId}

|  |
| --- |
| @RequestMapping("/cart/delete/{itemId}")  **public** String deleteCartItem(@PathVariable Long itemId,  HttpServletResponse response, HttpServletRequest request) {  TaotaoResult result = cartService.deleteCartItem(itemId, request, response);  **return** "redirect:/cart/cart.html";  } |

### 购物车畅想

1、未登录的情况下，可以把购物车写入cookie。

2、已经登录的情况下，需要把购物车写入redis

3、登录时判断cookie中有购物车商品，应该把cookie中的购物车商品列表转移到redis中。

Key:用户id

Value：购物车商品列表

1. 如果想redis中转移商品时，redis的购物车中已经有商品，此时，需要把商品合并。如果是同一款商品数量叠加，如果新商品就合并商品。

## 订单系统的实现

### 分析

1. 订单确认页面，确认购买的商品、使用优惠券或积分、确认配送地址。
2. 确认页面有一个订单提交按钮，点击按钮生成订单。
3. 生成订单需要调用服务。需要发布一个生成订单服务。

需要创建一订单系统

### 创建订单系统

需要直接使用关系数据库。

使用的技术：

Mybatis

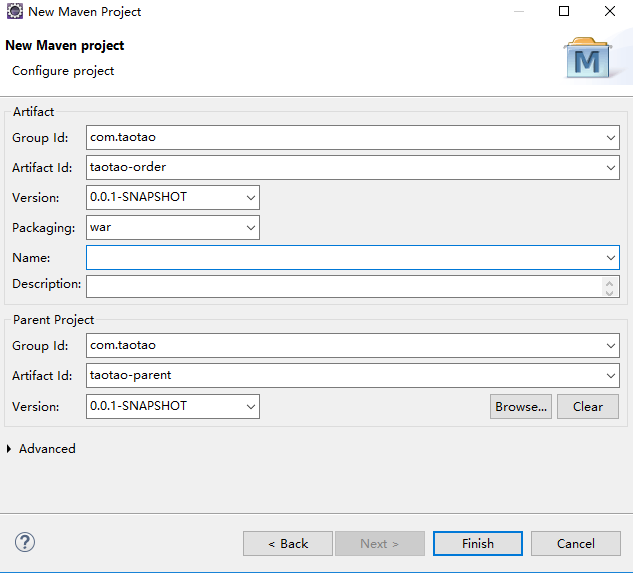
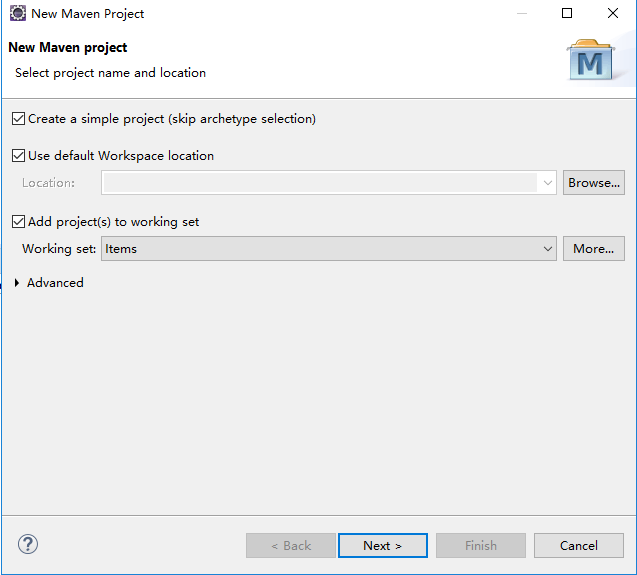
Spring

Springmvc

Redis

可以参考taotao-rest创建。

#### 创建工程

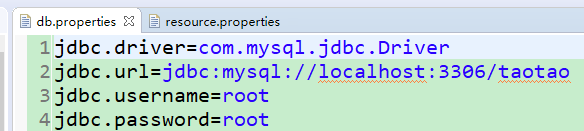
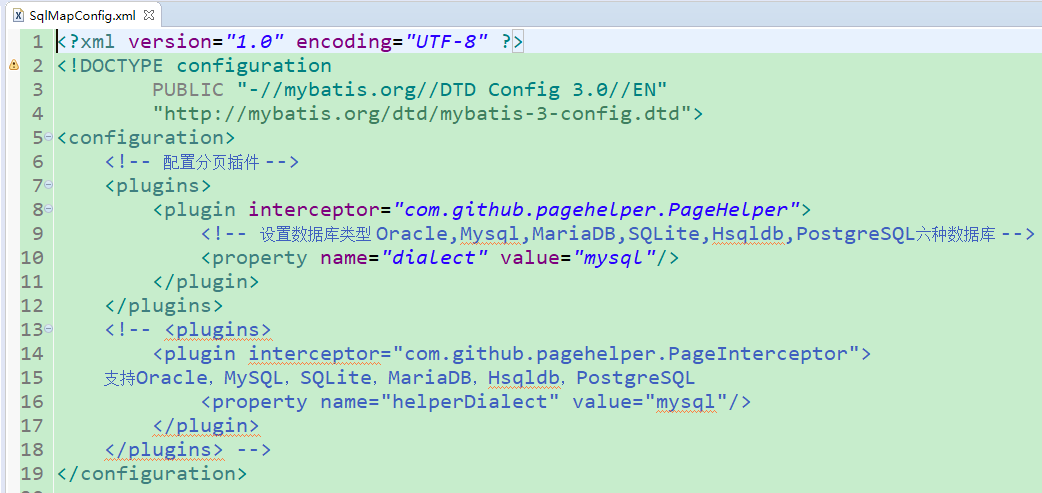


#### 依赖的jar包

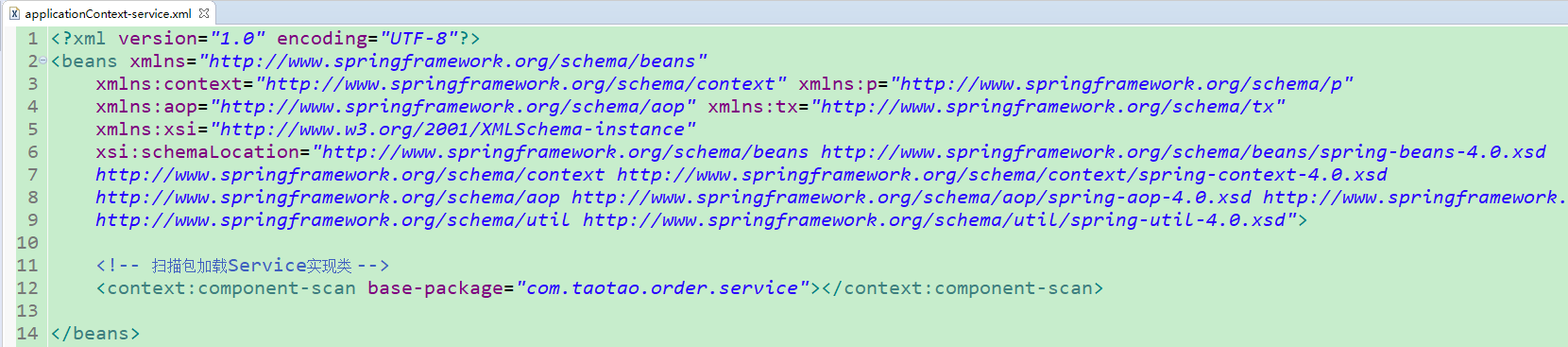
Pom.xml

|  |
| --- |
| <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-order</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>war</packaging>  <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-order</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>war</packaging>  <dependencies>  <dependency>  <groupId>com.taotao</groupId>  <artifactId>taotao-manager-mapper</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>8085</port>  <path>/</path>  </configuration>  </plugin>  </plugins>  </build>  </project> |

#### 框架整合

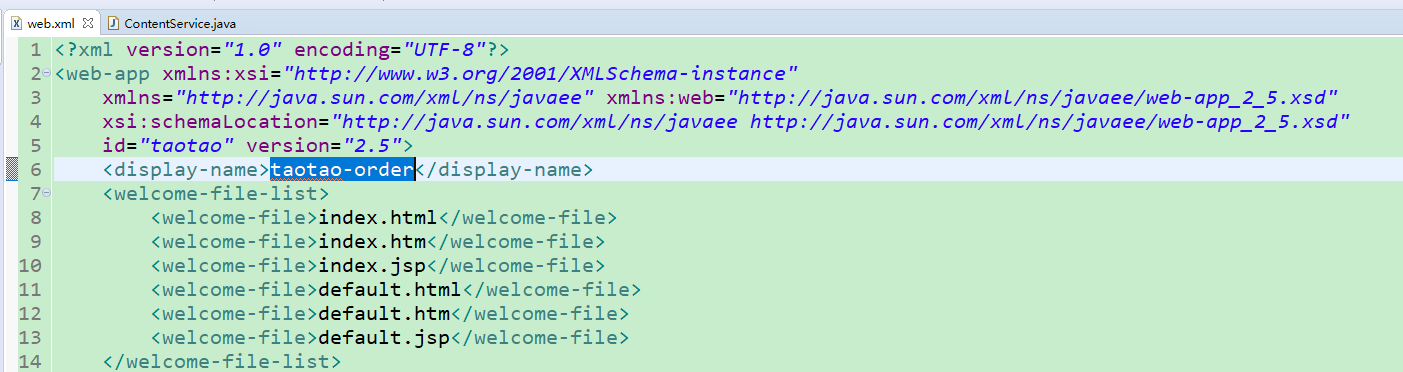












### 创建订单服务

#### 分析

1、接收json数据，需要使用@RequestBody注解。对应提交的json数据格式创建一个pojo接收。

需要使用的表：

tb\_order 订单表

tb\_order\_item 订单明细表

tb\_order\_shipping 物流表

可以使用订单相关的三个表对应的pojo封装一个pojo接收提交的json数据。

|  |
| --- |
| **public** **class** OrderInfo **extends** TbOrder{  **private** List<TbOrderItem> orderItems;  **private** TbOrderShipping orderShipping;  **public** List<TbOrderItem> getOrderItems() {  **return** orderItems;  }  **public** **void** setOrderItems(List<TbOrderItem> orderItems) {  **this**.orderItems = orderItems;  }  **public** TbOrderShipping getOrderShipping() {  **return** orderShipping;  }  **public** **void** setOrderShipping(TbOrderShipping orderShipping) {  **this**.orderShipping = orderShipping;  }  } |

1. 接收json数据后，把数据插入到响应的表中。

tb\_order 订单表

tb\_order\_item 订单明细表

tb\_order\_shipping 物流表

订单号的生成：要求可读性号，不能太长应该是纯数字的。

1. Uuid
2. 取毫秒
3. **使用redis的incr命令（redis是单线程，不会重复）**

补全订单的相关字段

插入到响应的表中

1. 返回TaotaoResult其中包含订单号

#### Dao层

向tb\_order 订单表tb\_order\_item 订单明细表tb\_order\_shipping 物流表插入数据，可以使用逆向工程。

#### Service

一、插入订单表

1. 接收数据OrderInfo
2. 生成订单号
3. 补全字段
4. 插入订单表
5. 插入订单明细
6. 生成订单明细id，使用redis的incr命令生成。
7. 补全字段
8. 插入数据
9. 插入物流表
10. 补全字段
11. 插入数据
12. 返回TaotaoResult包装订单号。

参数：OrderInfo

返回值：TaotaoResult

|  |
| --- |
| **package** com.taotao.order.service.impl;  **import** java.util.Date;  **import** java.util.List;  **import** org.apache.commons.lang3.StringUtils;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.beans.factory.annotation.Value;  **import** org.springframework.stereotype.Service;  **import** com.taotao.common.result.TaotaoResult;  **import** com.taotao.mapper.TbOrderItemMapper;  **import** com.taotao.mapper.TbOrderMapper;  **import** com.taotao.mapper.TbOrderShippingMapper;  **import** com.taotao.order.component.JedisClient;  **import** com.taotao.order.pojo.OrderInfo;  **import** com.taotao.order.service.OrderService;  **import** com.taotao.pojo.TbOrderItem;  **import** com.taotao.pojo.TbOrderShipping;  @Service  **public** **class** OrderServiceImpl **implements** OrderService {  @Autowired  **private** TbOrderMapper orderMapper;  @Autowired  **private** TbOrderItemMapper orderItemMapper;  @Autowired  **private** TbOrderShippingMapper orderShippingMapper;  @Autowired  **private** JedisClient jedisClient;    @Value("${REDIS\_ORDER\_GEN\_KEY}")  **private** String REDIS\_ORDER\_GEN\_KEY;  @Value("${ORDER\_ID\_BEGIN}")  **private** String ORDER\_ID\_BEGIN;  @Value("${REDIS\_ORDER\_DETAIL\_GEN\_KEY}")  **private** String REDIS\_ORDER\_DETAIL\_GEN\_KEY;    @Override  **public** TaotaoResult createOrder(OrderInfo orderInfo) {  // 一、插入订单表  // 1、接收数据OrderInfo  // 2、生成订单号  //取订单号  String id = jedisClient.get(REDIS\_ORDER\_GEN\_KEY);  **if** (StringUtils.*isBlank*(id)) {  //如果订单号生成key不存在设置初始值  jedisClient.set(REDIS\_ORDER\_GEN\_KEY, ORDER\_ID\_BEGIN);  }  Long orderId = jedisClient.incr(REDIS\_ORDER\_GEN\_KEY);  // 3、补全字段  orderInfo.setOrderId(orderId.toString());  //状态：1、未付款，2、已付款，3、未发货，4、已发货，5、交易成功，6、交易关闭  orderInfo.setStatus(1);  Date date = **new** Date();  orderInfo.setCreateTime(date);  orderInfo.setUpdateTime(date);  // 4、插入订单表  orderMapper.insert(orderInfo);  // 二、插入订单明细  // 2、补全字段  List<TbOrderItem> orderItems = orderInfo.getOrderItems();  **for** (TbOrderItem orderItem : orderItems) {  // 1、生成订单明细id，使用redis的incr命令生成。  Long detailId = jedisClient.incr(REDIS\_ORDER\_DETAIL\_GEN\_KEY);  orderItem.setId(detailId.toString());  //订单号  orderItem.setOrderId(orderId.toString());  // 3、插入数据  orderItemMapper.insert(orderItem);  }  // 三、插入物流表  TbOrderShipping orderShipping = orderInfo.getOrderShipping();  // 1、补全字段  orderShipping.setOrderId(orderId.toString());  orderShipping.setCreated(date);  orderShipping.setUpdated(date);  // 2、插入数据  orderShippingMapper.insert(orderShipping);  // 返回TaotaoResult包装订单号。  **return** TaotaoResult.*ok*(orderId);  }  } |

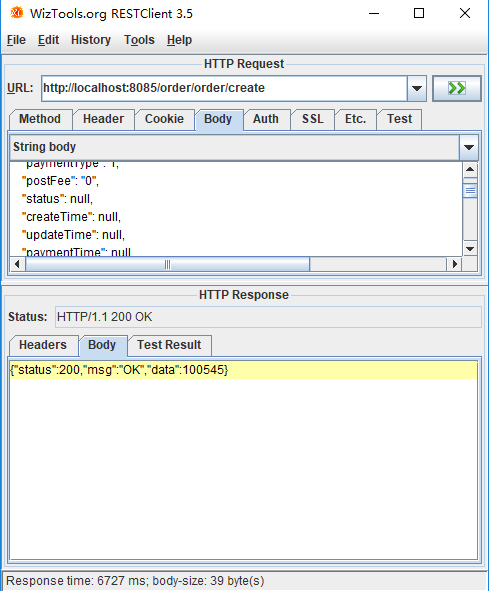
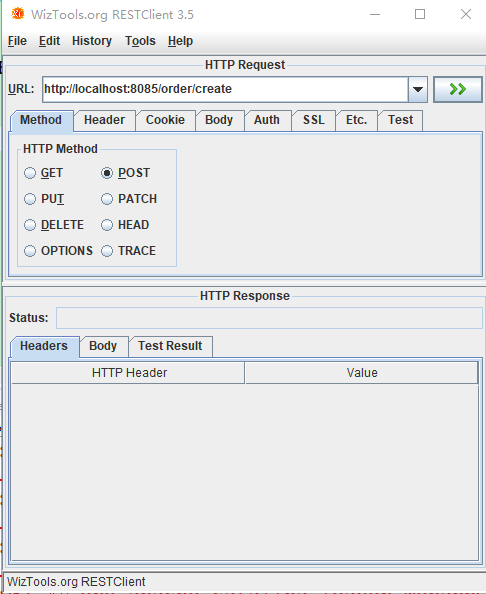
#### Controller

发布服务。

接收客户端发送的数据，接收json格式的数据使用@RequestBody OrderInfo 来接收数据，调用Service生成订单，返回TaotaoResult。

|  |
| --- |
| **package** com.taotao.order.controller;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestBody;  **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.order.pojo.OrderInfo;  **import** com.taotao.order.service.OrderService;  @Controller  **public** **class** OrderController {  @Autowired  **private** OrderService orderService;  @RequestMapping(value="/order/create", method=RequestMethod.***POST***)  @ResponseBody  **public** TaotaoResult createOrder(@RequestBody OrderInfo orderInfo) {  **try** {  TaotaoResult result = orderService.createOrder(orderInfo);  **return** result;    } **catch** (Exception e) {  e.printStackTrace();  **return** TaotaoResult.*build*(500, ExceptionUtil.*getStackTrace*(e));  }  }  } |

#### 测试



{

"orderId": null,

"payment": "20996.00",

"paymentType": 1,

"postFee": "0",

"status": null,

"createTime": null,

"updateTime": null,

"paymentTime": null,

"consignTime": null,

"endTime": null,

"closeTime": null,

"shippingName": null,

"shippingCode": null,

"userId": 38,

"buyerMessage": "empty",

"buyerNick": "xixi",

"buyerRate": null,

"orderItems": [

{

"id": null,

"itemId": "154684233038024",

"orderId": null,

"num": 3,

"title": "宏碁(Acer)暗影骑士3游戏本",

"price": 549900,

"totalFee": 1649700,

"picPath": "http://192.168.187.128/group1/M00/00/00/wKi7gFwvbO-ASU1GAAAF22vMnHo825.jpg"

},

{

"id": null,

"itemId": "154684383340439",

"orderId": null,

"num": 1,

"title": "宏碁(Acer)蜂鸟Swift3微边框金属轻薄本14英寸笔记本电脑SF314(i5-8250U 8G 128G SSD+1T IPS 指纹)小超银",

"price": 449900,

"totalFee": 449900,

"picPath": "http://192.168.187.128/group1/M00/00/00/wKi7gFwvczeABAmSAAAF22vMnHo859.jpg"

}

],

"orderShipping": {

"orderId": null,

"receiverName": "小白",

"receiverPhone": null,

"receiverMobile": "15891588888",

"receiverState": "北京",

"receiverCity": "北京",

"receiverDistrict": "昌平区",

"receiverAddress": "西三旗 xxxxxxxxx",

"receiverZip": null,

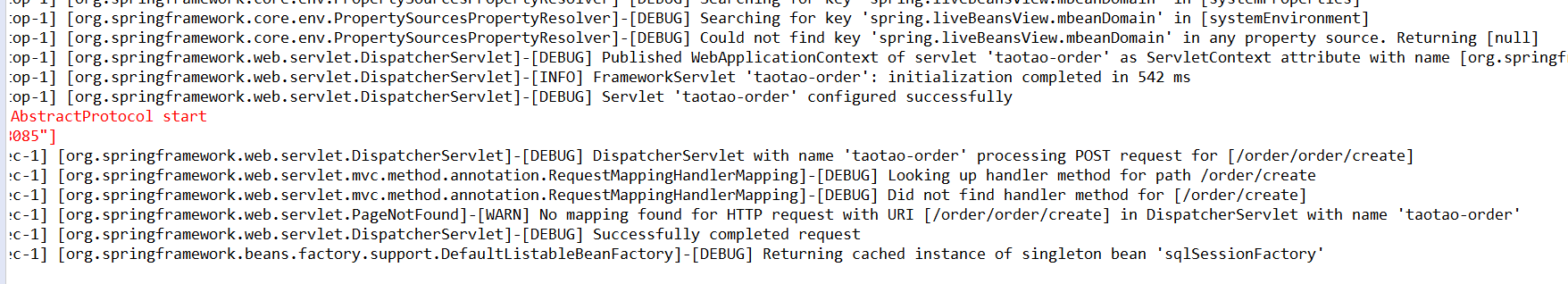
"created": null,

"updated": null

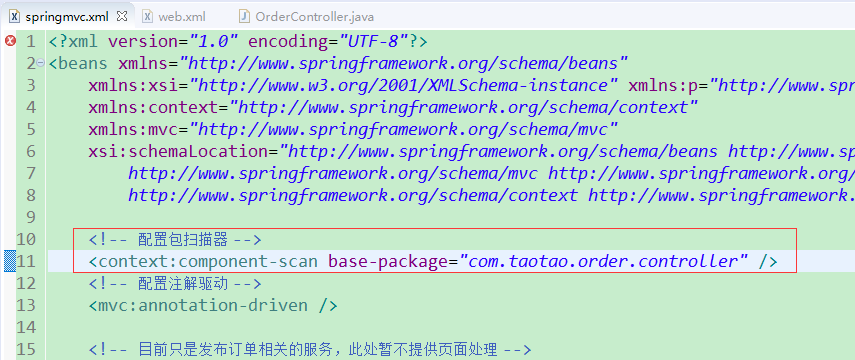
}

}

访问的时候出现如下问题，tomcat启动不报错，实际访问数据的时候报404问题



在配置文件springmvc.xml中没有指定controller扫描包路径：



## 创建订单portal的处理

### 分析

1. 显示订单确认页面
2. 提交订单处理

### 显示订单确认页面

请求的url：/order/order-cart

1、商品数据从购物车中获得。

2、访问此页面要求用户登录。

3、需要根据用户id查询常用地址列表（待实现）。

#### Service

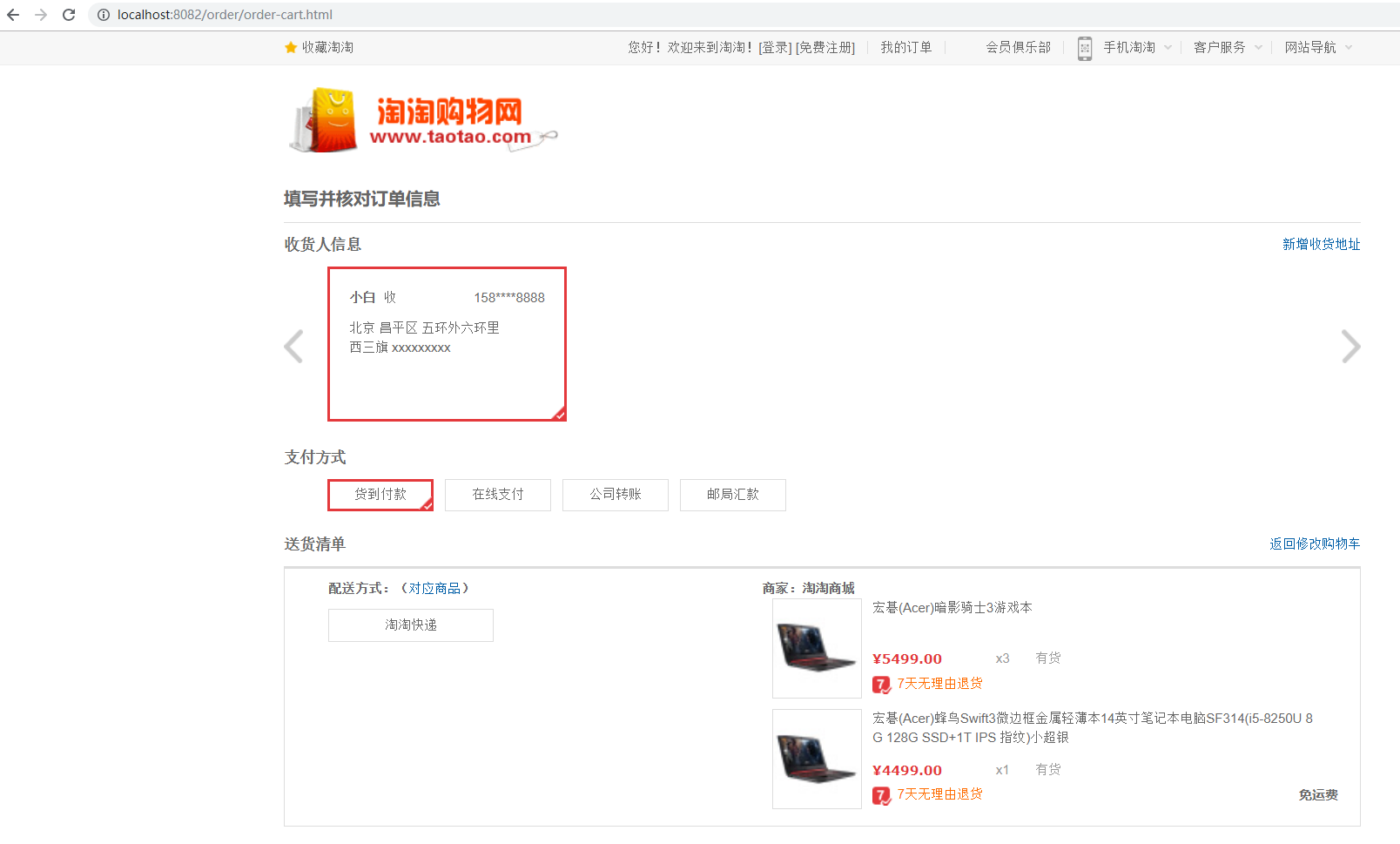
可以使用购物车Service

#### Controller

取购物车商品列表，传递给jsp。

|  |
| --- |
| **package** com.taotao.portal.controller;  **import** java.util.List;  **import** javax.servlet.http.HttpServletRequest;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.ui.Model;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** com.taotao.portal.pojo.CartItem;  **import** com.taotao.portal.service.CartService;  @Controller  @RequestMapping("/order")  **public** **class** OrderController {  @Autowired  **private** CartService cartService;  @RequestMapping("/order-cart")  **public** String showOrderCart(Model model, HttpServletRequest request) {  //取购物车商品列表  List<CartItem> list = cartService.getCartItems(request);  model.addAttribute("cartList", list);  **return** "order-cart";  }  } |

购物车结算：

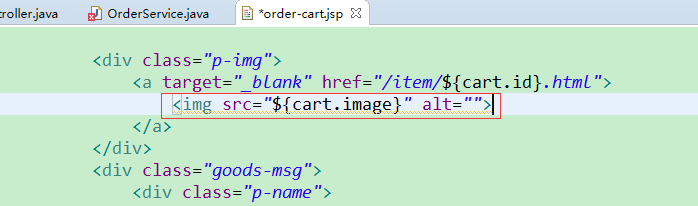


### 提交订单

#### 处理分析

在订单确认页面点击“提交订单按钮”进行提交订单的处理。把表单的内容提交给taotao-portal的后台，后台调用taotao-order发布的服务生成订单。展示订单提交成功页面，并且展示订单号。





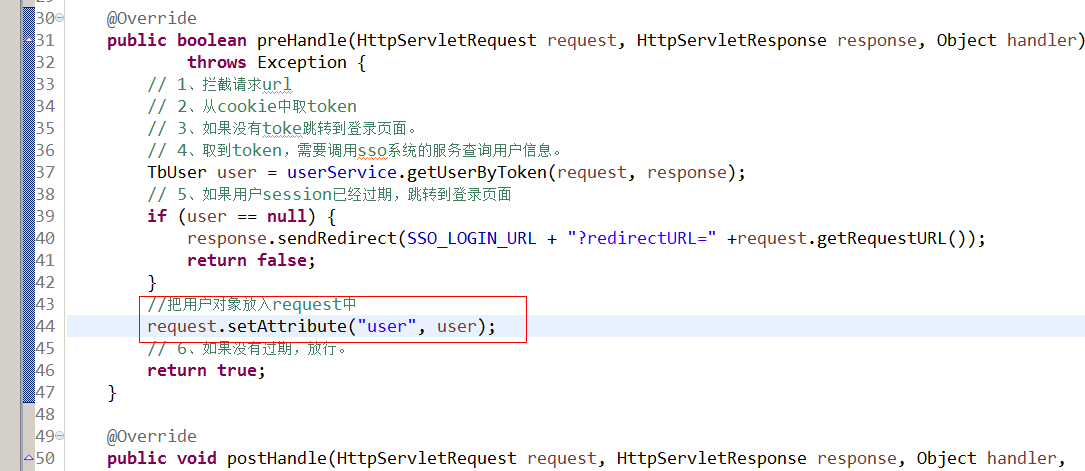
接收表单的内容：

使用OrderInfo来接收。

需要通过token取用户对象，使用userid和昵称



配置拦截器，在拦截器中把用户对象放入request。



#### Service

接收表单的内容OrderInfo，调用taotao-order的服务，生成订单。提交数据格式是json格式。

返回TaotaoResult，其中包含订单号，取出订单号，并返回。

参数：OrderInfo

返回值：String 订单号

|  |
| --- |
| @Service  **public** **class** OrderServiceImpl **implements** OrderService {  @Value("${ORDER\_BASE\_URL}")  **private** String ORDER\_BASE\_URL;  @Value("${ORDER\_CREATE\_URL}")  **private** String ORDER\_CREATE\_URL;  @Override  **public** String createOrder(OrderInfo orderInfo) {  //把OrderInfo转换成json  String json = JsonUtils.*objectToJson*(orderInfo);  //提交订单数据  String jsonResult = HttpClientUtil.*doPostJson*(ORDER\_BASE\_URL + ORDER\_CREATE\_URL, json);  //转换成java对象  TaotaoResult taotaoResult = TaotaoResult.*format*(jsonResult);  //取订单号  String orderId = taotaoResult.getData().toString();  **return** orderId;  }  } |

#### Controller

接收提交的订单信息，使用OrderInfo接收。调用Service创建订单，跳转到成功页面，并且把订单号传递给页面。

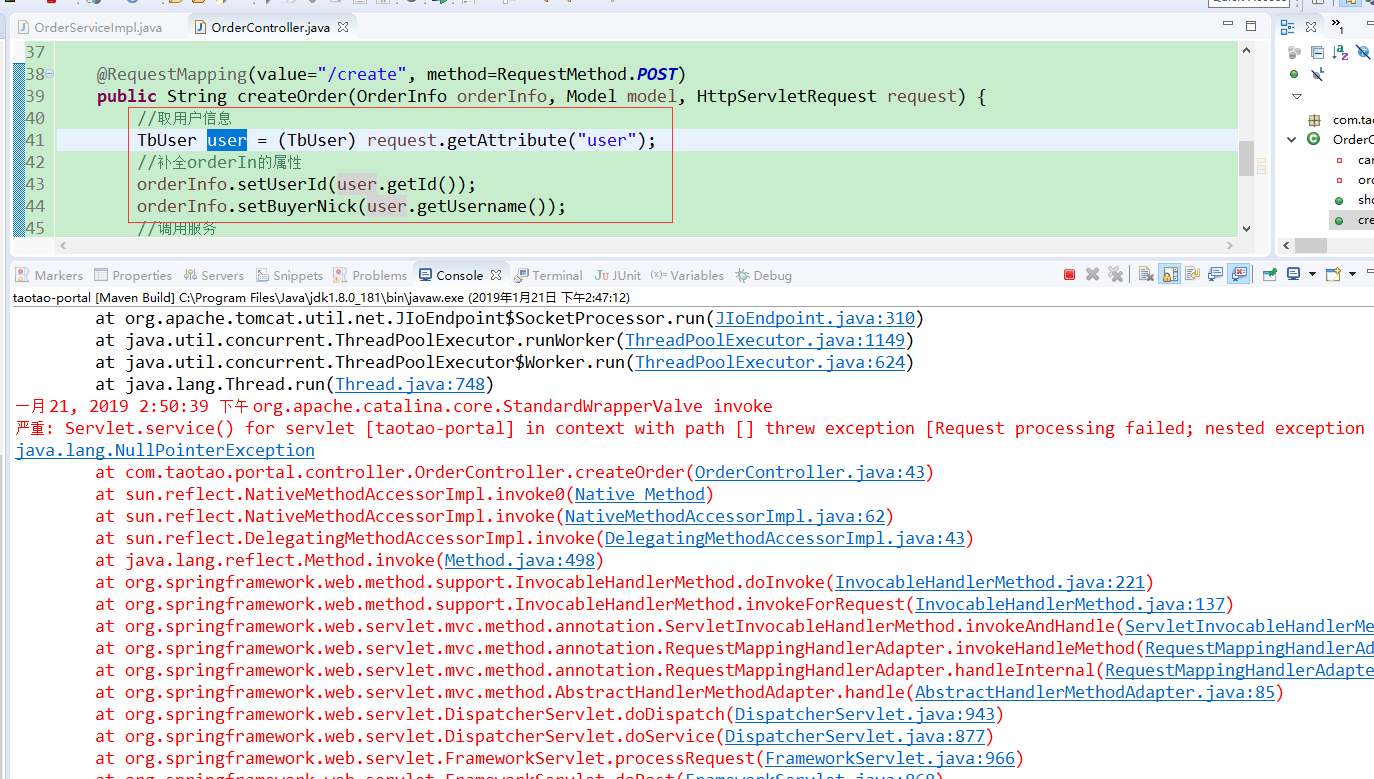
请求的url：/order/create

|  |
| --- |
| @Controller  @RequestMapping("/order")  **public** **class** OrderController {  @Autowired  **private** CartService cartService;    @Autowired  **private** OrderService orderService;    @RequestMapping("/order-cart")  **public** String showOrderCart(Model model, HttpServletRequest request) {  //取购物车商品列表  List<CartItem> list = cartService.getCartItems(request);  model.addAttribute("cartList", list);  **return** "order-cart";  }    @RequestMapping(value="/create", method=RequestMethod.***POST***)  **public** String createOrder(OrderInfo orderInfo, Model model, HttpServletRequest request) {  //取用户信息  TbUser user = (TbUser) request.getAttribute("user");  //补全orderIn的属性  orderInfo.setUserId(user.getId());  orderInfo.setBuyerNick(user.getUsername());  //调用服务  String orderId = orderService.createOrder(orderInfo);  //把订单号传递个页面  model.addAttribute("orderId", orderId);  model.addAttribute("payment", orderInfo.getPayment());  DateTime dateTime = **new** DateTime();  dateTime = dateTime.plusDays(3);  model.addAttribute("date", dateTime.toString("yyyy-MM-dd"));  //返回逻辑视图  **return** "success";  }  } |

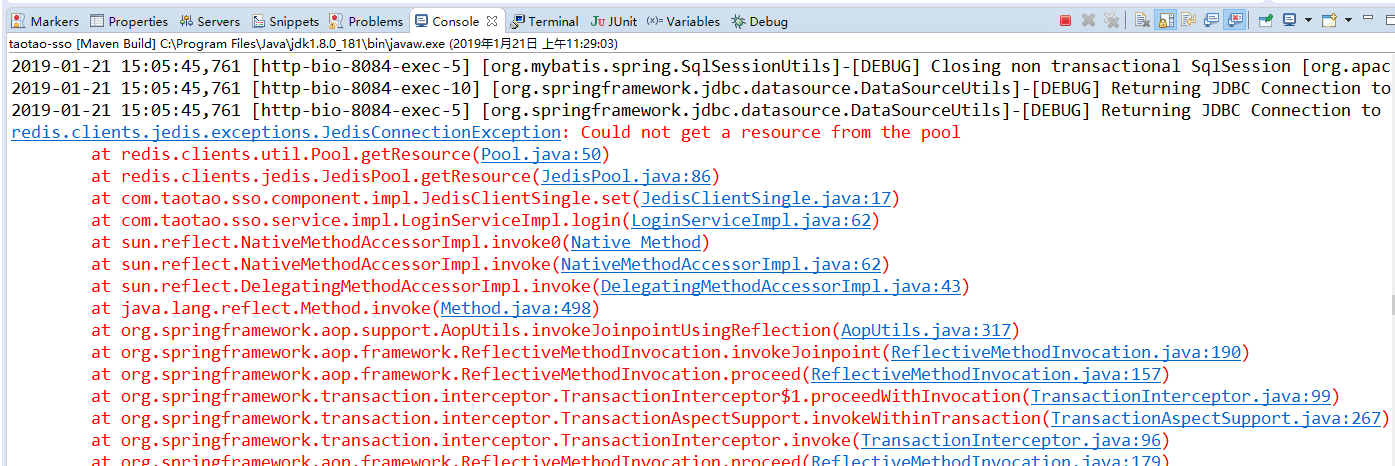
提交订单



可能出现的问题分析：空指针异常，指向对应的user对象，可能是由于拦截器在之前的测试过程中被注释了，导致此处没有将当前登录用户传给相应的jsp文件，也无法从jsp文件中获取该内容，导致出错报空指针异常



Jedis连接问题：



问题解决思路：

* 查看相应的redis是否启动？
* 查看服务器防火墙是否关闭？
* 查看redis服务器访问的ip、端口号是否正确？
* Jedis对象使用完后需要释放，否则导致其一直占用而无法获取新的资源
* 当jedispool中的jedis被取完，而等待时间超过你设置的 MaxWaitMillis 就会抛出Could not get a resource from the pool
* 连接池中空闲的连接过一阵子就会自动断开，但是连接池还以为连接正常