**淘淘商城**

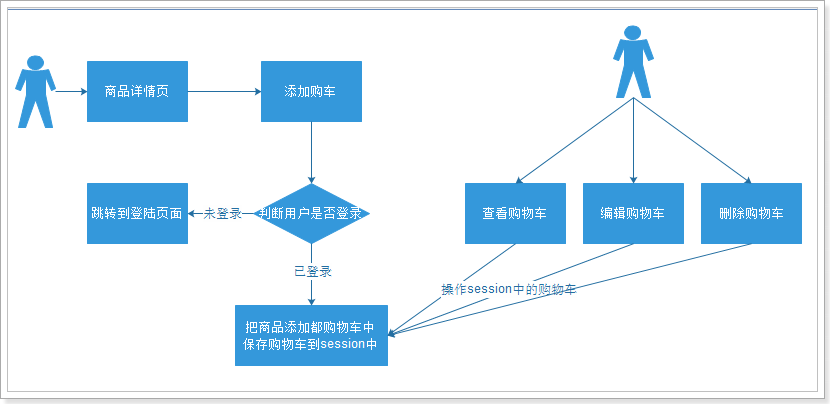
**第十二天**

# 课程计划

1. 购物车流程
2. 实现购物车功能
   1. 创建购物车系统
   2. 登录状态的购物车实现
   3. 未登录状态的购物车实现

# 购物车流程

## 以前的购车流程

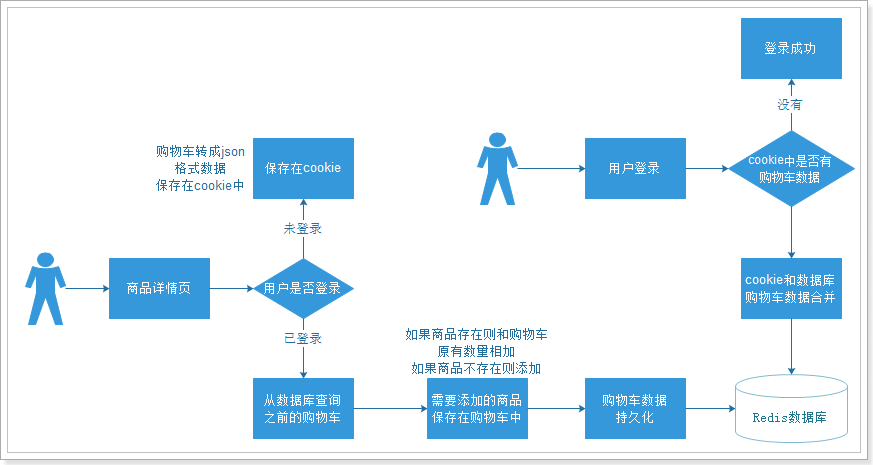


存在的问题：

1. 购物车保存在session，如果session销毁，购物车就没了
   1. 给购物车做持久化操作
2. 用户未登录的时候，不能添加购物车
   1. 未登录状态，可以把购物车数据保存在cookie
3. 购物车使用session，session是使用服务器的内存，使用了服务器的内存资源，这样很不好
   1. MySQL数据库（数据的完整性）
   2. redis数据库（使用redis，I/O读写速度快）
4. session无法共享，无法行水平扩展（集群）

## 现在的购物车实现

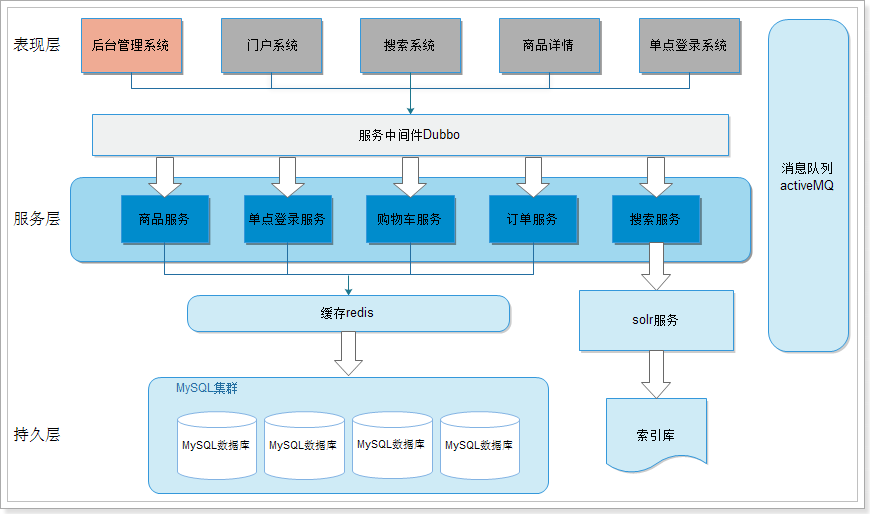
1. 实现用户登录状态和未登录状态的购车操作（添加，查看，修改，删除）



# 购物车系统搭建

## 创建购物车系统

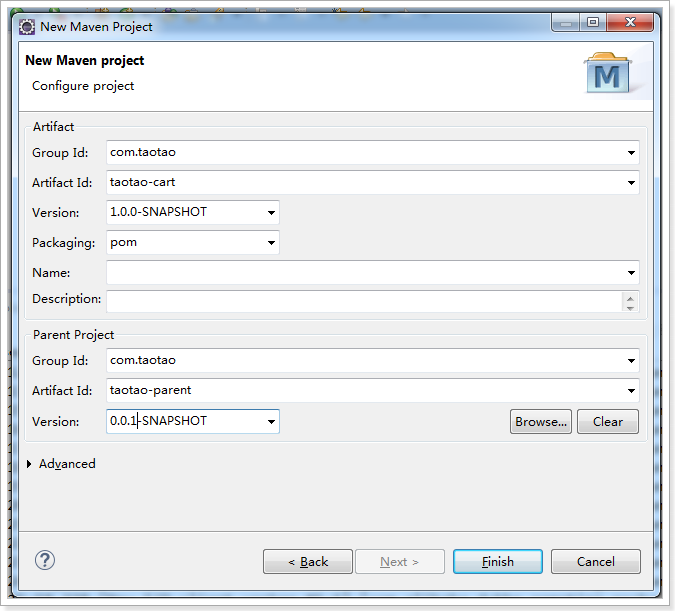
需要创建购物服务层，提供购物车服务



### 创建taotao-cart

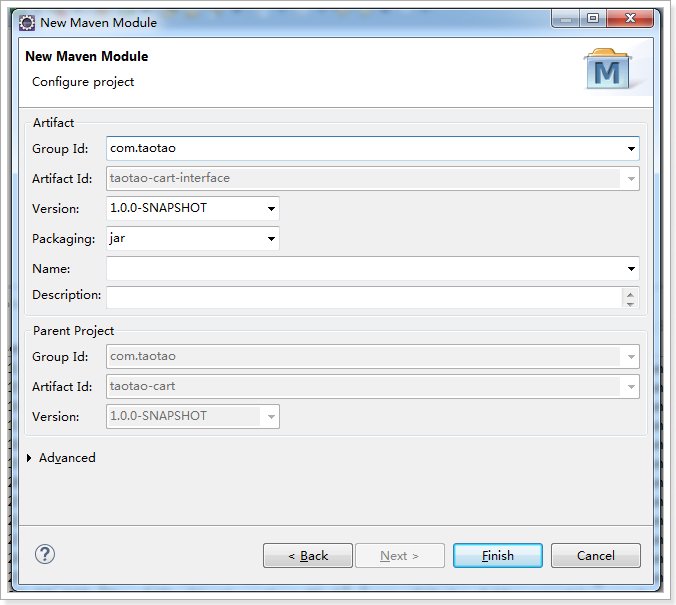
taotao-cart服务层和taotao-sso类似

聚合工程的父工程



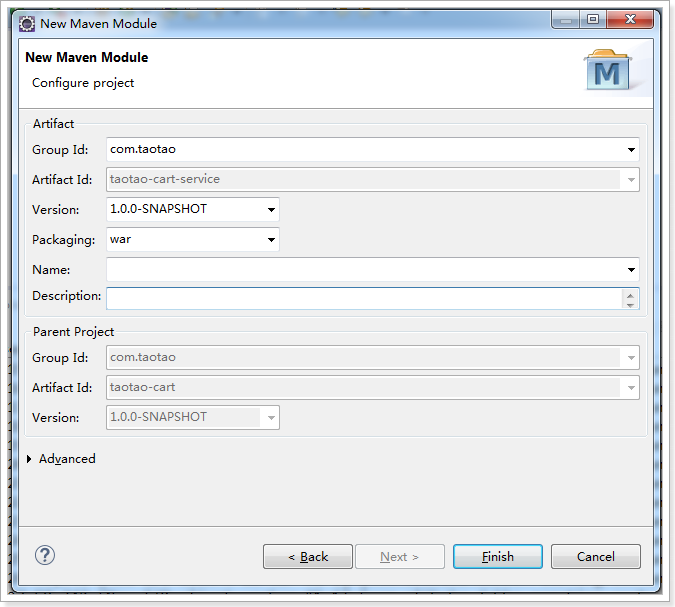
### 创建taotao-cart-interface

聚合工程子工程



### 创建taotao-cart-service

聚合工程子工程



## 加入依赖

### taotao-cart

<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-cart</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>pom</packaging>

<modules>

<module>taotao-cart-interface</module>

<module>taotao-cart-service</module>

</modules>

<dependencies>

<!-- 单元测试 -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<scope>test</scope>

</dependency>

<!-- 日志 -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

</dependency>

<!-- Apache通用工具组件 -->

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-lang3</artifactId>

</dependency>

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-io</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<!-- 配置Tomcat插件 -->

<plugin>

<groupId>org.apache.tomcat.maven</groupId>

<artifactId>tomcat7-maven-plugin</artifactId>

<configuration>

<port>8086</port>

<path>/</path>

</configuration>

</plugin>

</plugins>

</build>

</project>

### taotao-cart-interface

<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-cart</artifactId>

<version>1.0.0-SNAPSHOT</version>

</parent>

<artifactId>taotao-cart-interface</artifactId>

<dependencies>

<!-- 加入taotao-manager-pojo依赖 -->

<dependency>

<groupId>com.taotao</groupId>

<artifactId>taotao-manager-pojo</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

</dependencies>

</project>

### taotao-cart-service

<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-cart</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<artifactId>taotao-cart-service</artifactId>

<version>1.0.0-SNAPSHOT</version>

<packaging>war</packaging>

<dependencies>

<!-- 加入taotao-cart-interface依赖 -->

<dependency>

<groupId>com.taotao</groupId>

<artifactId>taotao-cart-interface</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

<!-- 加入taotao-manager-mapper依赖 -->

<dependency>

<groupId>com.taotao</groupId>

<artifactId>taotao-manager-mapper</artifactId>

<version>1.0.0-SNAPSHOT</version>

</dependency>

<!-- Spring -->

<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>

<!-- Jackson Json处理工具包 -->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

</dependency>

<!-- dubbo相关 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>dubbo</artifactId>

<exclusions>

<exclusion>

<groupId>org.springframework</groupId>

<artifactId>spring</artifactId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<groupId>org.apache.zookeeper</groupId>

<artifactId>zookeeper</artifactId>

</dependency>

<dependency>

<groupId>com.github.sgroschupf</groupId>

<artifactId>zkclient</artifactId>

</dependency>

<!-- Redis客户端 -->

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

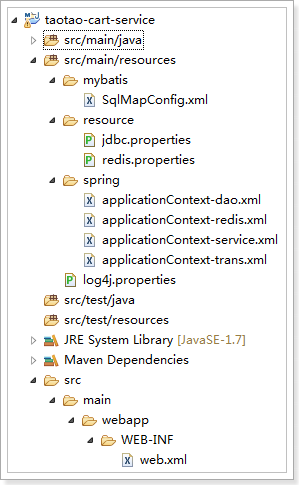
</dependency>

</dependencies>

</project>

## 配置文件

taotao-cart-service添加配置文件



### web.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns=*"http://java.sun.com/xml/ns/javaee"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*

id=*"MyWebApp"* version=*"2.5"*>

<display-name>taotao-cart</display-name>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

<!-- spring -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring/applicationContext\*.xml</param-value>

</context-param>

<!-- spring监听器 -->

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

</web-app>

### applicationContext-dao.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"*>

<!-- 配置 读取properties文件 jdbc.properties -->

<context:property-placeholder location=*"classpath:resource/\*.properties"* />

<!-- 配置数据源 -->

<bean id=*"dataSource"* class=*"com.jolbox.bonecp.BoneCPDataSource"*

destroy-method=*"close"*>

<!-- 数据库驱动 -->

<property name=*"driverClass"* value=*"${jdbc.driver}"* />

<!-- 相应驱动的jdbcUrl -->

<property name=*"jdbcUrl"* value=*"${jdbc.url}"* />

<!-- 数据库的用户名 -->

<property name=*"username"* value=*"${jdbc.username}"* />

<!-- 数据库的密码 -->

<property name=*"password"* value=*"${jdbc.password}"* />

<!-- 检查数据库连接池中空闲连接的间隔时间，单位是分，默认值：240，如果要取消则设置为0 -->

<property name=*"idleConnectionTestPeriod"* value=*"60"* />

<!-- 连接池中未使用的链接最大存活时间，单位是分，默认值：60，如果要永远存活设置为0 -->

<property name=*"idleMaxAge"* value=*"30"* />

<!-- 每个分区最大的连接数 -->

<property name=*"maxConnectionsPerPartition"* value=*"150"* />

<!-- 每个分区最小的连接数 -->

<property name=*"minConnectionsPerPartition"* value=*"5"* />

</bean>

<!-- 配置SqlSessionFactory -->

<bean class=*"org.mybatis.spring.SqlSessionFactoryBean"*>

<!-- 配置Mybatis核心配置文件 -->

<property name=*"configLocation"* value=*"classpath:mybatis/SqlMapConfig.xml"* />

<!-- 配置数据源 -->

<property name=*"dataSource"* ref=*"dataSource"* />

</bean>

<!-- 配置Mapper扫描 -->

<bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>

<!-- 配置Mapper扫描包 -->

<property name=*"basePackage"* value=*"com.taotao.manager.mapper"* />

</bean>

</beans>

### applicationContext-redis.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"* xmlns:dubbo=*"http://code.alibabatech.com/schema/dubbo"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd*

*http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"*>

<!-- 配置JedisPool -->

<bean class=*"redis.clients.jedis.JedisPool"*>

<!-- 配置redis单机版的ip地址 -->

<constructor-arg name=*"host"* value=*"${redis.host}"* />

<!-- 配置redis单机版的端口号 -->

<constructor-arg name=*"port"* value=*"${redis.port}"* />

</bean>

<!-- 配置单机版实现 -->

<bean class=*"com.taotao.cart.redis.RedisPool"* />

<!-- 配置集群版的连接对象jedisCluster -->

<bean class=*"redis.clients.jedis.JedisCluster"*>

<constructor-arg name=*"nodes"*>

<!-- 配置集群节点信息 -->

<set>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"${cluster1.host}"* />

<constructor-arg name=*"port"* value=*"${cluster1.port}"* />

</bean>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"${cluster2.host}"* />

<constructor-arg name=*"port"* value=*"${cluster2.port}"* />

</bean>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"${cluster3.host}"* />

<constructor-arg name=*"port"* value=*"${cluster3.port}"* />

</bean>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"${cluster4.host}"* />

<constructor-arg name=*"port"* value=*"${cluster4.port}"* />

</bean>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"${cluster5.host}"* />

<constructor-arg name=*"port"* value=*"${cluster5.port}"* />

</bean>

<bean class=*"redis.clients.jedis.HostAndPort"*>

<constructor-arg name=*"host"* value=*"${cluster6.host}"* />

<constructor-arg name=*"port"* value=*"${cluster6.port}"* />

</bean>

</set>

</constructor-arg>

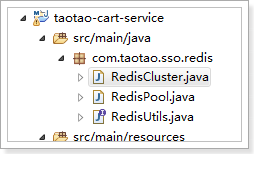
</bean>

<!-- 配置集群版的实现 -->

<!-- <bean class="com.taotao.cart.redis.RedisCluster" /> -->

</beans>

复制taotao-sso-service的Redis工具类到taotao-cart-service



### applicationContext-service.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"* xmlns:dubbo=*"http://code.alibabatech.com/schema/dubbo"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd*

*http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"*>

<!-- 配置service扫描 -->

<context:component-scan base-package=*"com.taotao.cart.service"* />

<!-- 提供方应用信息，用于计算依赖关系 -->

<dubbo:application name=*"taotao-cart-service"* />

<!-- 使用multicast广播注册中心暴露服务地址 -->

<!-- <dubbo:registry address="multicast://224.5.6.7:1234" /> -->

<!-- <dubbo:registry address="N/A" /> -->

<dubbo:registry protocol=*"zookeeper"* address=*"192.168.37.161:2181"* />

<!-- 用dubbo协议在20880端口暴露服务 -->

<dubbo:protocol name=*"dubbo"* port=*"20884"* />

<!-- 声明需要暴露的服务接口 -->

<!-- <dubbo:service interface="com.taotao.content.service.ContentCategoryService" -->

<!-- ref="contentCategoryServiceImpl" /> -->

</beans>

### applicationContext-trans.xml

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*

*http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*

*http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>

<!-- 定义事务管理器 -->

<bean id=*"transactionManager"*

class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>

<property name=*"dataSource"* ref=*"dataSource"* />

</bean>

<!-- 定义事务策略 -->

<tx:advice id=*"txAdvice"* transaction-manager=*"transactionManager"*>

<tx:attributes>

<!-- 传播行为 -->

<tx:method name=*"save\*"* propagation=*"REQUIRED"* />

<tx:method name=*"insert\*"* propagation=*"REQUIRED"* />

<tx:method name=*"add\*"* propagation=*"REQUIRED"* />

<tx:method name=*"create\*"* propagation=*"REQUIRED"* />

<tx:method name=*"delete\*"* propagation=*"REQUIRED"* />

<tx:method name=*"update\*"* propagation=*"REQUIRED"* />

<tx:method name=*"find\*"* propagation=*"SUPPORTS"* read-only=*"true"* />

<tx:method name=*"select\*"* propagation=*"SUPPORTS"* read-only=*"true"* />

<tx:method name=*"get\*"* propagation=*"SUPPORTS"* read-only=*"true"* />

<tx:method name=*"query\*"* propagation=*"SUPPORTS"* read-only=*"true"* />

</tx:attributes>

</tx:advice>

<aop:config>

<!--pointcut元素定义一个切入点，execution中的第一个星号 用以匹配方法的返回类型， 这里星号表明匹配所有返回类型。 com.abc.dao.\*.\*(..)表明匹配cn.itcast.mybatis.service包下的所有类的所有

方法 -->

<aop:pointcut id=*"myPointcut"*

expression=*"execution(\* com.taotao.cart.service.\*.\*(..))"* />

<!--将定义好的事务处理策略应用到上述的切入点 -->

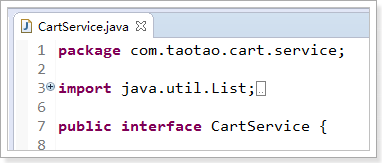
<aop:advisor advice-ref=*"txAdvice"* pointcut-ref=*"myPointcut"* />

</aop:config>

</beans>

## 创建服务并发布

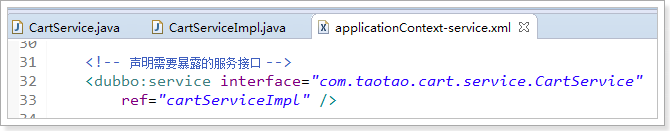
### 创建购物车服务接口



### 创建购物车接口实现类



### 发布服务



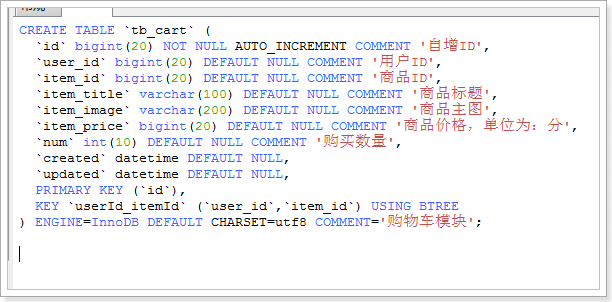
# 实现登录状态购物车功能

## 创建pojo

购物车原来是放在MySQL中。

因为购物车功能用户频繁访问，需要提高其性能，所以进行改造，在redis中进行保存。

原来的购物车表



id是购物车id，是主键

user\_id是用户id，关联用户表，一个用户的购物车可以添加多个商品

item\_id是商品id，关联商品表

根据原来的购物车表创建购物车pojo

在taotao-manager-pojo创建

**public** **class** Cart **extends** BasePojo {

**private** Long id;

**private** Long userId;

**private** Long itemId;

**private** String itemTitle;

**private** String itemImage;

**private** Long itemPrice;

**private** Integer num;

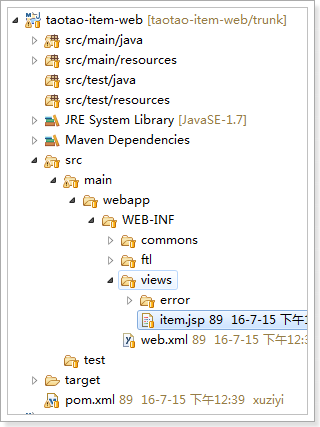
get/set。。。。。。

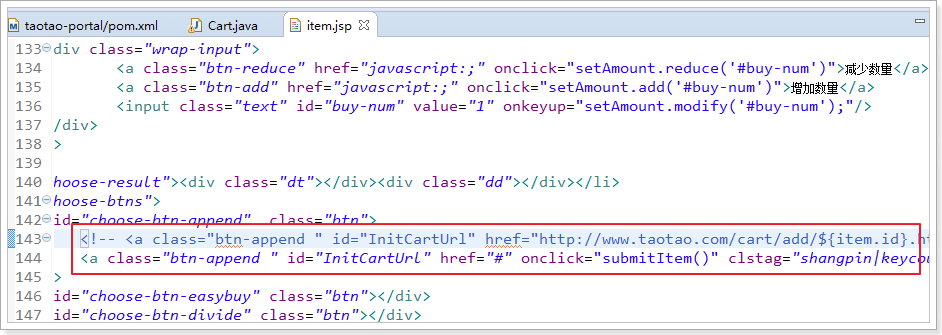
## 实现添加购物车

### 页面分析

在页面发现可以修改商品数量，但商品数量无法提交，需要改造页面请求

在taotao-item-web修改item.jsp页面







### 需求分析

用户进入商品详情页可以点击“加入购物车”按钮，

然后跳转到购物车详情页，可以查看到新添加的商品和之前添加的商品

首先完成登录状态下的购物车功能

实现：

需要完成两件事：

1. 添加商品到购物车
   1. 获取用户信息
   2. 调用购物车服务，把商品数据保存在redis的购物车中的
      1. 参数包含：用户id，商品id，商品数量
2. 展示购物车详情页
   1. 获取用户信息，需要用户的id
   2. 调用购物车服务，把该用户的所有购物车数据查询数来，根据用户id查询
   3. 跳转到购物车页面（cart.jsp），显示数据

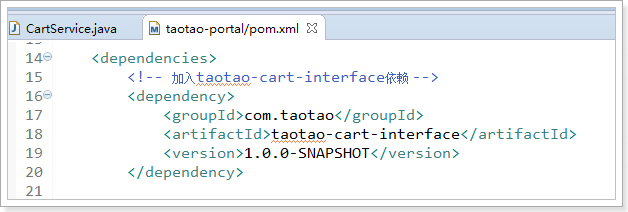
前台系统实现的购物车功能，所以在前台系统的controller接受请求

前台系统调用购物车服务，实现把商品添加到购物车中

### 功能实现

#### taotao-portal添加依赖

taotao-portal需要调用购物车服务，需要添加接口依赖

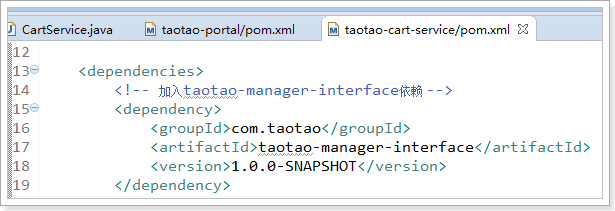


#### taotao-portal声明服务调用



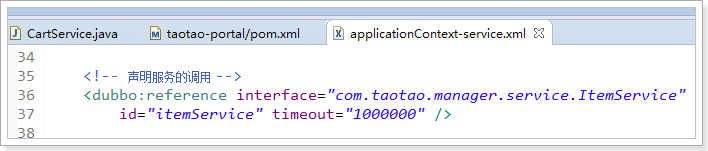
#### taotao-cart-service添加依赖

taotao-cart-service需要使用商品服务，需要添加taotao-manager-interface



#### taotao-cart-service声明服务调用

taotao-cart-service使用商品服务



#### 实现Controller

在taotao-portal编写

@Controller

@RequestMapping("cart")

**public** **class** CartController {

@Value("${TT\_TICKET}")

**private** String TT\_TICKET;

@Autowired

**private** CartService cartService;

@Autowired

**private** UserService userService;

// http://www.taotao.com/cart/1082433.html?num=4

/\*\*

\* 添加商品到购物车

\*

\* **@param** itemId

\* **@param** num

\* **@return**

\*/

@RequestMapping(value = "{itemId}", method = RequestMethod.***GET***)

**public** String saveItemByCart(@PathVariable("itemId") Long itemId, Integer num,

HttpServletRequest request) {

// 获取用户登录信息

String ticket = CookieUtils.*getCookieValue*(request, **this**.TT\_TICKET);

User user = **this**.userService.queryUserByTicket(ticket);

// 判断用户是否登录

**if** (user != **null**) {

// 用户已登录

**this**.cartService.saveItemByCart(user.getId(), itemId, num);

} **else** {

// 用户未登录

}

// 重定向到购物车详情页

**return** "redirect:/cart/show.html";

}

/\*\*

\* 显示购物车详情页

\*

\* **@param** model

\* **@param** request

\* **@return**

\*/

@RequestMapping(value = "show", method = RequestMethod.***GET***)

**public** String showCart(Model model, HttpServletRequest request) {

// 获取用户登录信息

String ticket = CookieUtils.*getCookieValue*(request, **this**.TT\_TICKET);

User user = **this**.userService.queryUserByTicket(ticket);

List<Cart> cartList = **null**;

// 判断用户是否登录

**if** (user != **null**) {

// 用户已登录

cartList = **this**.cartService.queryCartByUserId(user.getId());

} **else** {

// 用户未登录

}

// 保存购物车到模型中

model.addAttribute("cartList", cartList);

// 显示购物车详情页

**return** "cart";

}

}

#### 实现Service接口

在taotao-cart-interface编写

**public** **interface** CartService {

/\*\*

\* 添加商品到购物车

\*

\* **@param** userId

\* **@param** itemId

\* **@param** num

\*/

**void** saveItemByCart(Long userId, Long itemId, Integer num);

/\*\*

\* 根据用户查询购物车数据

\*

\* **@param** userId

\* **@return**

\*/

List<Cart> queryCartByUserId(Long userId);

}

#### Service接口实现

在taotao-cart-service编写

@Service

**public** **class** CartServiceImpl **implements** CartService {

**private** **final** **static** ObjectMapper ***MAPPER*** = **new** ObjectMapper();

@Value("${TAOTAO\_CART\_KEY}")

**private** String TAOTAO\_CART\_KEY;

@Autowired

**private** RedisUtils redisUtils;

@Autowired

**private** ItemMapper itemMapper;

@Override

**public** **void** saveItemByCart(Long userId, Long itemId, Integer num) {

// 从redis中查询购物车数据。redis中保存的是购物车的json格式数据

List<Cart> list = **this**.queryCartByUserId(userId);

// 遍历购物车，商品是否存在

Cart cart = **null**;

**for** (Cart c : list) {

**if** (c.getItemId().longValue() == itemId.longValue()) {

cart = c;

}

}

// 判断商品在购物车是否存在

**if** (cart != **null**) {

// 如果存在,商品数量相加

cart.setNum(cart.getNum() + num);

cart.setUpdated(**new** Date());

} **else** {

// 如果不存在,使用商品服务获取商品数据

Item item = **this**.itemMapper.selectByPrimaryKey(itemId);

// 创建购物车数据

cart = **new** Cart();

cart.setUserId(userId);

cart.setItemId(itemId);

cart.setItemTitle(item.getTitle());

cart.setItemPrice(item.getPrice());

cart.setItemImage(item.getImage());

cart.setNum(num);

cart.setCreated(**new** Date());

cart.setUpdated(cart.getCreated());

list.add(cart);

}

**try** {

// 把添加好的购物车保存在redis中

**this**.redisUtils.set(**this**.TAOTAO\_CART\_KEY + userId, ***MAPPER***.writeValueAsString(list));

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

@Override

**public** List<Cart> queryCartByUserId(Long userId) {

// 从redis中查询购物车数据

String json = **this**.redisUtils.get(**this**.TAOTAO\_CART\_KEY + userId);

// 判断json数据不为空

**if** (StringUtils.*isNotBlank*(json)) {

// 把json反序列化为list集合

**try** {

// json格式数据序列化为集合

List<Cart> list = ***MAPPER***.readValue(json,

***MAPPER***.getTypeFactory().constructCollectionType(List.**class**, Cart.**class**));

// 返回购物车数据

**return** list;

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

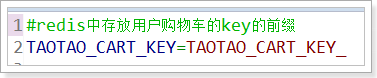
// 如果有异常或者查询为空，返回空集合，其他方法调用时，不会因为遍历报错

**return** **new** ArrayList<Cart>();

}

}

加入配置文件



## 实现编辑购物车

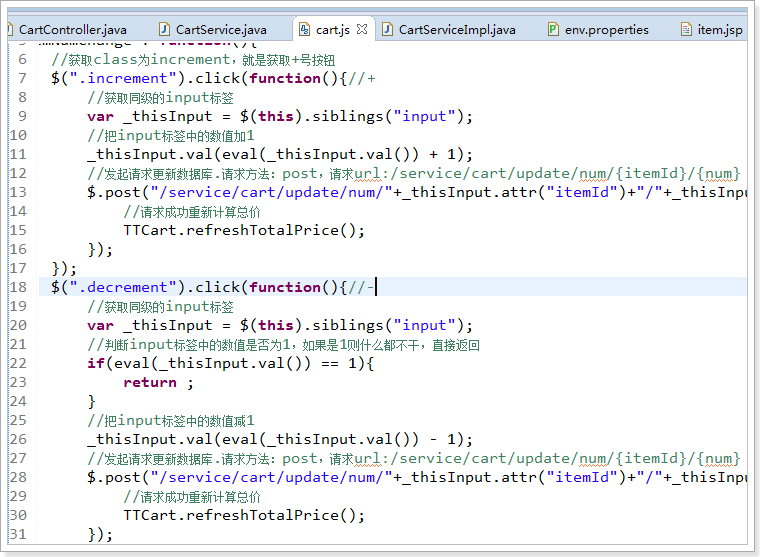
### 购物车编辑前端实现

页面可以通过三种方式修改：

1. 按-号数量减一
2. 按+号数量加一
3. 直接在输入框编辑



taotao-portal的webapp\js\cart.js文件



### 实现Controller

在taotao-portal编写

// 请求方法：post

// 请求url:/service/cart/update/num/{itemId}/{num}

/\*\*

\* 更新购物车商品数量

\*

\* **@param** itemId

\* **@param** num

\* **@param** request

\*/

@RequestMapping(value = "update/num/{itemId}/{num}", method = RequestMethod.***POST***)

@ResponseBody

**public** **void** updateNumByCart(@PathVariable("itemId") Long itemId, @PathVariable("num") Integer num,

HttpServletRequest request) {

// 获取用户登录信息

String ticket = CookieUtils.*getCookieValue*(request, **this**.TT\_TICKET);

User user = **this**.userService.queryUserByTicket(ticket);

// 判断用户是否登录

**if** (user != **null**) {

// 用户已登录

**this**.cartService.updateNumByCart(user.getId(), itemId, num);

} **else** {

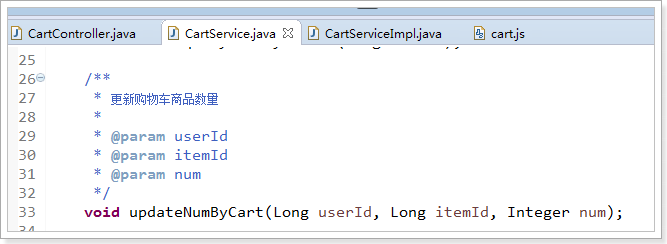
// 用户未登录

}

}

### 实现Service接口

在taotao-cart-interface编写



### Service接口实现

在taotao-cart-service编写

@Override

**public** **void** updateNumByCart(Long userId, Long itemId, Integer num) {

// 从redis中查询购物车数据。redis中保存的是购物车的json格式数据

List<Cart> list = **this**.queryCartByUserId(userId);

// 遍历购物车，商品是否存在,存在则更新，不存在什么都不做

**for** (Cart cart : list) {

**if** (cart.getItemId().longValue() == itemId.longValue()) {

// 如果存在，则更新购物车商品数量。这里是更新，不是相加

cart.setNum(num);

cart.setUpdated(**new** Date());

**try** {

// 把添加后的购物车保存在redis中

**this**.redisUtils.set(**this**.CART\_TAOTAO\_CART\_KEY + userId, ***MAPPER***.writeValueAsString(list));

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

## 实现删除购物车

点击删除按钮发起请求

http://www.taotao.com/cart/delete/1082433.html

应该删除redis中的购物车数据后，再刷新购物车详情页

### 实现Controller

在taotao-portal编写

// http://www.taotao.com/cart/delete/1082433.html

/\*\*

\* 删除购物车的商品

\*

\* **@param** itemId

\* **@param** request

\* **@return**

\*/

@RequestMapping(value = "delete/{itemId}", method = RequestMethod.***GET***)

**public** String deleteItemByCart(@PathVariable("itemId") Long itemId, HttpServletRequest request) {

// 获取用户登录信息

String ticket = CookieUtils.*getCookieValue*(request, **this**.TT\_TICKET);

User user = **this**.userService.queryUserByTicket(ticket);

// 判断用户是否登录

**if** (user != **null**) {

// 用户已登录

**this**.cartService.deleteItemByCart(user.getId(), itemId);

} **else** {

// 用户未登录

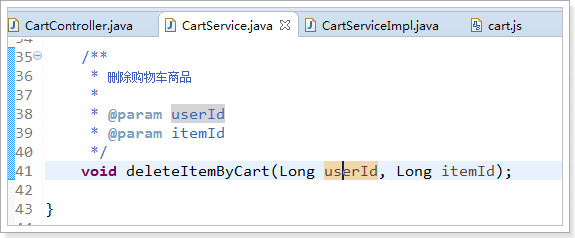
}

**return** "redirect:/cart/show.html";

}

### 实现Service接口

在taotao-cart-interface编写



### Service接口实现

在taotao-cart-service编写

@Override

**public** **void** deleteItemByCart(Long userId, Long itemId) {

// 从redis中查询购物车数据。redis中保存的是购物车的json格式数据

List<Cart> list = **this**.queryCartByUserId(userId);

// 遍历购物车，商品是否存在,存在则更新，不存在什么都不做

**for** (Cart cart : list) {

**if** (cart.getItemId().longValue() == itemId.longValue()) {

list.remove(cart);

**try** {

// 把删除后的购物车保存在redis中

**this**.redisUtils.set(**this**.CART\_TAOTAO\_CART\_KEY + userId, ***MAPPER***.writeValueAsString(list));

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

// 跳出循环

**break**;

}

}

}

# 实现未登录状态购物车功能

未登录的用户也可以实现添加购物车功能

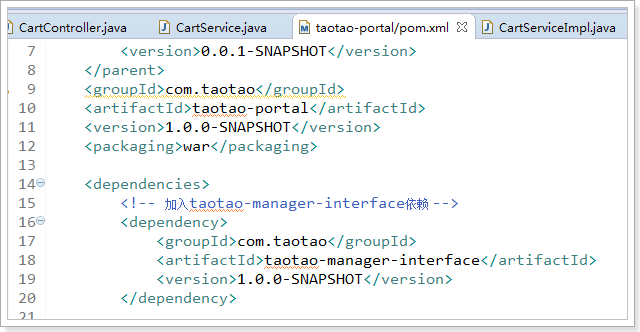
把购物车数据保存在cookie中。

操作cookie不需要调用购物车服务，taotao-portal创建CartCookieService实现即可

但是需要商品服务，获取商品信息

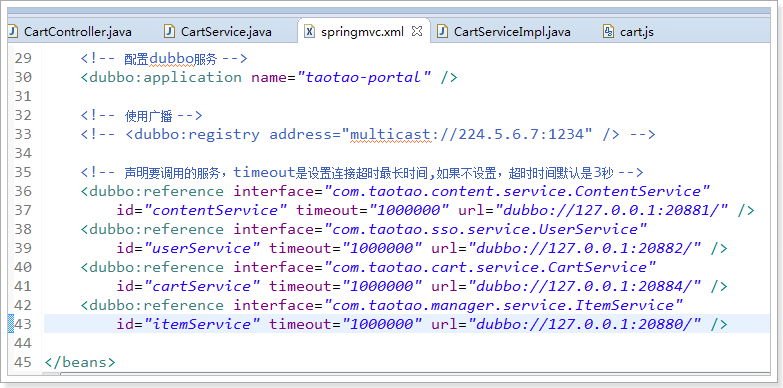
## 添加依赖

taotao-portal需要商品的服务，添加依赖



## 声明服务调用

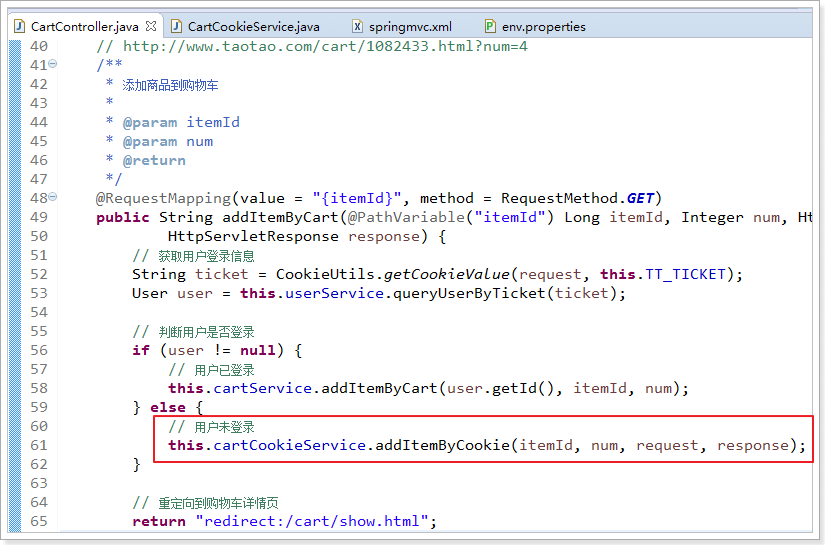
taotao-portal声明商品服务ItemService的调用



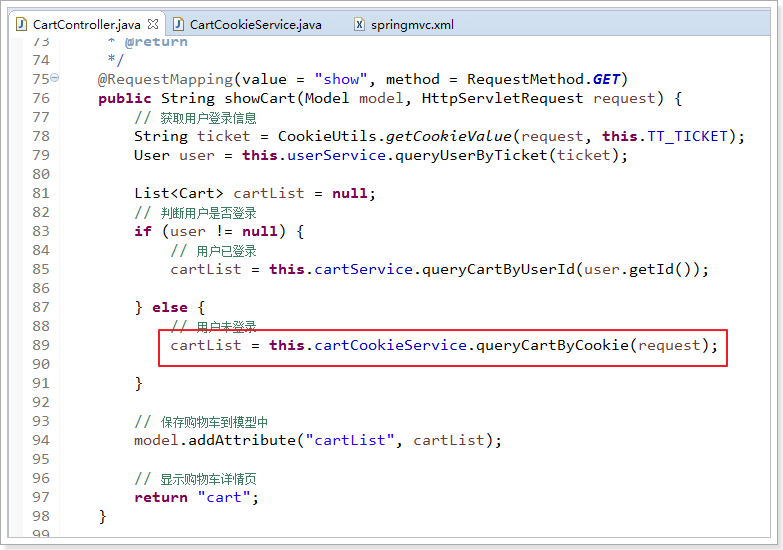
## 实现添加购物车

### 实现Controller

在taotao-portal编写，添加商品到购物车



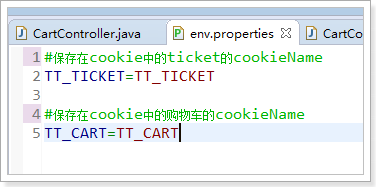
显示购物车详情页



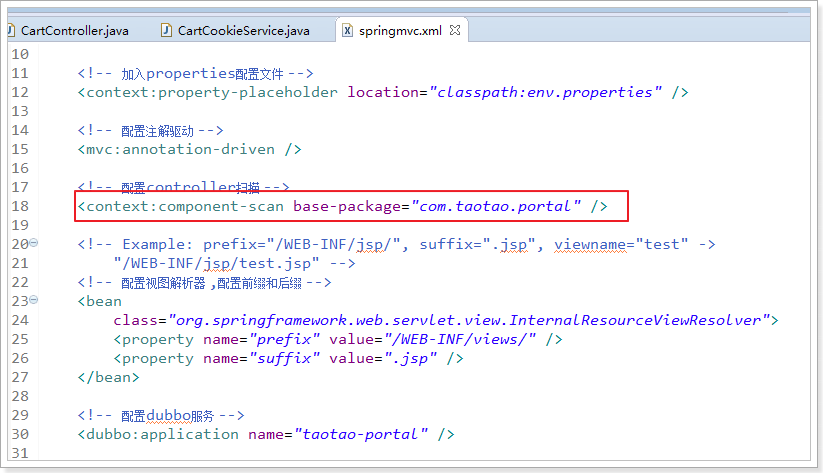
### Service实现

在taotao-portal编写

配置文件



修改SpringMVC的扫描配置



实现service

@Service

**public** **class** CartCookieService {

**private** **final** **static** ObjectMapper ***MAPPER*** = **new** ObjectMapper();

@Value("${TT\_CART}")

**private** String TT\_CART;

@Autowired

**private** ItemService itemService;

/\*\*

\* 添加商品到cookie的购物车中

\*

\* **@param** itemId

\* **@param** num

\* **@param** request

\* **@param** response

\*/

**public** **void** addItemByCookie(Long itemId, **int** num, HttpServletRequest request,

HttpServletResponse response) {

// 从cookie中获取购物车数据

List<Cart> list = **this**.queryCartByCookie(request);

Cart cart = **null**;

// 遍历购物车数据

**for** (Cart c : list) {

**if** (c.getItemId().longValue() == itemId.longValue()) {

cart = c;

}

}

**if** (cart != **null**) {

// 如果存在则商品数量相加

cart.setNum(cart.getNum() + num);

cart.setUpdated(**new** Date());

} **else** {

// 如果不存在则把商品加入到购物车中

// 获取商品数据

Item item = **this**.itemService.queryById(itemId);

// 设置购物车数据

cart = **new** Cart();

cart.setId(**null**);

cart.setUserId(**null**);

cart.setItemId(itemId);

cart.setItemTitle(item.getTitle());

// 设置商品图片url

**if** (item.getImages() != **null**) {

cart.setItemImage(item.getImages()[0]);

} **else** {

cart.setItemImage(**null**);

}

cart.setItemPrice(item.getPrice());

cart.setNum(num);

cart.setCreated(**new** Date());

cart.setUpdated(cart.getCreated());

// 把购物车放在list中

list.add(cart);

}

// 把修改后的购物车数据放到cookie中

**try** {

CookieUtils.*setCookie*(request, response, **this**.TT\_CART, ***MAPPER***.writeValueAsString(list),

60 \* 60 \* 24 \* 3, **true**);

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

/\*\*

\* 查询cookie中的购物车数据

\*

\* **@param** request

\* **@return**

\*/

**public** List<Cart> queryCartByCookie(HttpServletRequest request) {

// 从cookie获取购物车的json格式数据

String json = CookieUtils.*getCookieValue*(request, **this**.TT\_CART, **true**);

// 判断json数据不为空

List<Cart> list = **null**;

**if** (StringUtils.*isNotBlank*(json)) {

// 把json反序列化为list集合

**try** {

// json格式数据序列化为集合

list = ***MAPPER***.readValue(json,

***MAPPER***.getTypeFactory().constructCollectionType(List.**class**, Cart.**class**));

**return** list;

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

// 如果json为空，返回空的list集合

**return** **new** ArrayList<Cart>();

}

}

## 实现编辑购物车

### 实现Controller

在taotao-portal编写

// 请求方法：post

// 请求url:/service/cart/update/num/{itemId}/{num}

/\*\*

\* 更新购物车商品数量

\*

\* **@param** itemId

\* **@param** num

\* **@param** request

\* **@return**

\*/

@RequestMapping(value = "update/num/{itemId}/{num}", method = RequestMethod.***POST***)

@ResponseBody

**public** String updateNumByCart(@PathVariable("itemId") Long itemId, @PathVariable("num") Integer num,

HttpServletRequest request, HttpServletResponse response) {

// 获取用户登录信息

String ticket = CookieUtils.*getCookieValue*(request, **this**.TT\_TICKET);

User user = **this**.userService.queryUserByTicket(ticket);

// 判断用户是否登录

**if** (user != **null**) {

// 用户已登录

**this**.cartService.updateNumByCart(user.getId(), itemId, num);

} **else** {

// 用户未登录

**this**.cartCookieService.updateNumByCookie(itemId, num, request, response);

}

**return** "ok";

}

### Service实现

在taotao-portal编写

/\*\*

\* 修改购物车中的商品数量

\*

\* **@param** itemId

\* **@param** num

\* **@param** request

\* **@param** response

\*/

**public** **void** updateNumByCookie(Long itemId, **int** num, HttpServletRequest request,

HttpServletResponse response) {

// 从cookie获取购物车数据

List<Cart> list = **this**.queryCartByCookie(request);

// 遍历购物车

**for** (Cart cart : list) {

**if** (cart.getItemId().longValue() == itemId.longValue()) {

// 如果找到对应的商品，修改商品数量，注意:不是相加

cart.setNum(num);

cart.setUpdated(**new** Date());

// 把修改后的购物车数据保存在cookie中

**try** {

CookieUtils.*setCookie*(request, response, **this**.TT\_CART, ***MAPPER***.writeValueAsString(list),

60 \* 60 \* 24 \* 3, **true**);

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**break**;

}

}

}

## 实现删除购物车

### 实现Controller

在taotao-portal编写

// http://www.taotao.com/cart/delete/1082433.html

/\*\*

\* 删除购物车的商品

\*

\* **@param** itemId

\* **@param** request

\* **@return**

\*/

@RequestMapping(value = "delete/{itemId}", method = RequestMethod.***GET***)

**public** String deleteItemByCart(@PathVariable("itemId") Long itemId, HttpServletRequest request,

HttpServletResponse response) {

// 获取用户登录信息

String ticket = CookieUtils.*getCookieValue*(request, **this**.TT\_TICKET);

User user = **this**.userService.queryUserByTicket(ticket);

// 判断用户是否登录

**if** (user != **null**) {

// 用户已登录

**this**.cartService.deleteItemByCart(user.getId(), itemId);

} **else** {

// 用户未登录

**this**.cartCookieService.deleteItemByCookie(itemId, request, response);

}

**return** "redirect:/cart/show.html";

}

### Service实现

在taotao-portal编写

/\*\*

\* 删除购物车中的商品

\*

\* **@param** itemId

\* **@param** request

\* **@param** response

\*/

**public** **void** deleteItemByCookie(Long itemId, HttpServletRequest request, HttpServletResponse response) {

// 从cookie获取购物车数据

List<Cart> list = **this**.queryCartByCookie(request);

// 遍历购物车

**for** (Cart cart : list) {

**if** (cart.getItemId().longValue() == itemId.longValue()) {

// 如果找到对应的商品，删除商品

list.remove(cart);

// 把删除后的购物车数据保存在cookie中

**try** {

CookieUtils.*setCookie*(request, response, **this**.TT\_CART, ***MAPPER***.writeValueAsString(list),

60 \* 60 \* 24 \* 3, **true**);

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**break**;

}

}

}