谷粒商城

版本：V 1.0

购物车

# 一、 购物车业务简介



**当前的位置**

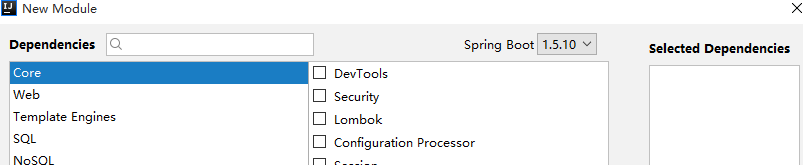
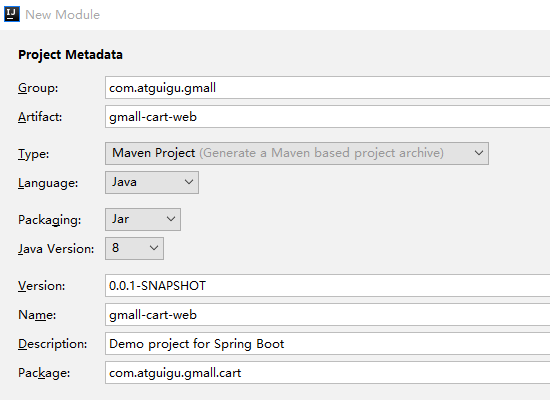
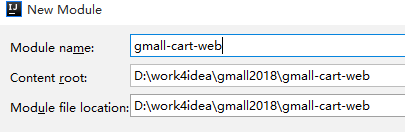
购物车模块要能过存储顾客所选的的商品，记录下所选商品，还要能随时更新，当用户决定购买时，用户可以选择决定购买的商品进入结算页面。

功能要求：

1. 要持久化，保存到数据库中。Mysql -- cartInfo
2. 利用缓存提高性能。Redis
3. 未登录状态也可以存入购物车，一旦用户登录要进行合并操作。Cookie --【Redis】

# 二、 购物车模块搭建

## 1 gmall-cart-web

pom.xml

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <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>   <groupId>com.atguigu.gmall</groupId>  <artifactId>gmall-cart-web</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>jar</packaging>   <name>gmall-cart-web</name>  <description>Demo project for Spring Boot</description>   <parent>  <groupId>com.atguigu.gmall</groupId>  <artifactId>gmall-parent</artifactId>  <version>1.0-SNAPSHOT</version>  </parent>    <dependencies>   <dependency>  <groupId>com.atguigu.gmall</groupId>  <artifactId>gmall-interface</artifactId>  <version>1.0-SNAPSHOT</version>  </dependency>   <dependency>  <groupId>com.atguigu.gmall</groupId>  <artifactId>gmall-web-util</artifactId>  <version>1.0-SNAPSHOT</version>  </dependency>  </dependencies>   <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>   </project> |

application.properties

|  |
| --- |
| server.port=8088 spring.thymeleaf.cache=false spring.thymeleaf.mode=LEGACYHTML5  spring.dubbo.application.name=cart-web spring.dubbo.registry.protocol=zookeeper spring.dubbo.registry.address=192.168.67.202:2181 spring.dubbo.base-package=com.atguigu.gmall spring.dubbo.protocol.name=dubbo spring.dubbo.consumer.timeout=100000 spring.dubbo.consumer.check=false |

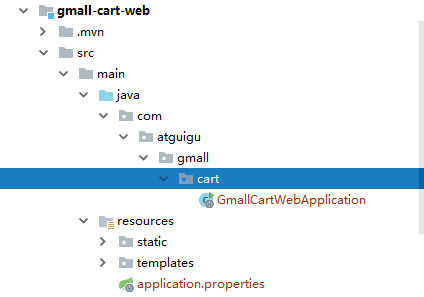
host文件

|  |
| --- |
| # gmall  192.168.67.163 cart.gmall.com passport.atguigu.com item.gmall.com list.gmall.com manage.gmall.com www.gmall.com resource.gmall.com |

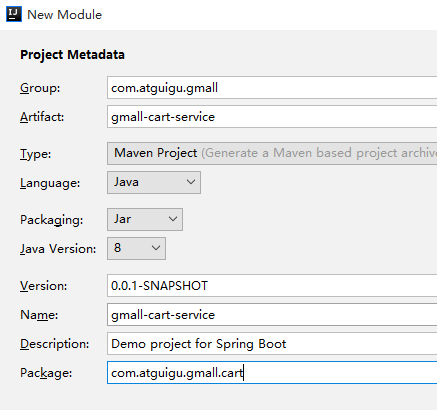
nginx.conf

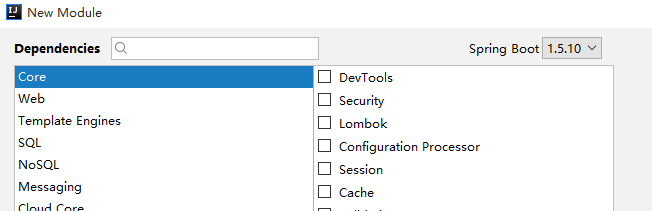
|  |
| --- |
| upstream cart.gmall.com{  server 192.168.67.1:8088;  }  server {  listen 80;  server\_name cart.gmall.com;  location / {  proxy\_pass http://cart.gmall.com;  proxy\_set\_header X-forwarded-for $proxy\_add\_x\_forwarded\_for;  }  } |

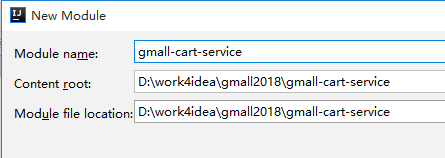
拷入静态文件和页面



## 2 gmall-cart-service







pom.xml

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <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>   <groupId>com.atguigu.gmall</groupId>  <artifactId>gmall-cart-service</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>jar</packaging>   <name>gmall-cart-service</name>  <description>Demo project for Spring Boot</description>   <parent>  <groupId>com.atguigu.gmall</groupId>  <artifactId>gmall-parent</artifactId>  <version>1.0-SNAPSHOT</version>  </parent>  <dependencies>   <dependency>  <groupId>com.atguigu.gmall</groupId>  <artifactId>gmall-interface</artifactId>  <version>1.0-SNAPSHOT</version>  </dependency>   <dependency>  <groupId>com.atguigu.gmall</groupId>  <artifactId>gmall-service-util</artifactId>  <version>1.0-SNAPSHOT</version>  </dependency>   </dependencies>   <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>   </project> |

application.properties

|  |
| --- |
| spring.datasource.url=jdbc:mysql://localhost:3306/gmall?characterEncoding=UTF-8 spring.datasource.username=root spring.datasource.password=root  logging.level.root=debug  server.port=8089  spring.dubbo.application.name=cartservice spring.dubbo.registry.protocol=zookeeper spring.dubbo.registry.address=192.168.67.202:2181 spring.dubbo.base-package=com.atguigu.gmall spring.dubbo.protocol.name=dubbo  spring.redis.host=192.168.67.202 spring.redis.port=6379 spring.redis.database=0 |

# 三、功能—添加入购物车

商品详情页面,item.html.

## 1 功能解析：

1. 根据**skuId**查询出商品详情skuInfo
2. 把skuInfo信息对应保存到购物车[购物车的实体类]
3. 返回成功页面

## 2 设计购物车的数据结构



**下单才会真正锁定价格！**

## 3 实体bean—CartInfo

|  |
| --- |
| **public class** CartInfo **implements** Serializable{   @GeneratedValue(strategy = GenerationType.***IDENTITY***)  @Id  @Column  String **id**;  @Column  String **userId**;  @Column  String **skuId**;  @Column  BigDecimal **cartPrice**;  @Column  Integer **skuNum**;  @Column  String **imgUrl**;  @Column  String **skuName**;   // 实时价格  @Transient  BigDecimal **skuPrice**;  // 下订单的时候，商品是否勾选  @Transient  String **isChecked**=**"0"**;  } |

## 4 Redis中的结构

购物车数据存放利用Hash结构存储：

String ,List,Hash,Set,ZSet

Set(key,value)

Demo:购物车中有两件商品，如果使用String 类型，要想修改第一个商品，则必须将所有的value全部进行反序列化。设计到Io操作，性能降低。

Hset(key,field,value): field = skuId

要想修改第一个商品，hget(key,001,cartInfoJson); 需要修改哪件商品，则反序列化哪一个商品即可！

hset(user:[userId]:cart,skuId,CartInfoJson)

|  |
| --- |
| key： “user:[userId]:cart”  field: [skuId]  value: CartInfo (Json) |

## 5 gmall-cart-service 模块业务方法

### 1 思路：

1. 先检查该用户的购物车里是否已经有该商品
2. 如果有商品，只要把对应商品的数量增加上去就可以，同时更新缓存
3. 如果没有该商品，则把对应商品插入到购物车中，同时插入缓存。

## 别忘记写mapper

CartInfoMapper

### 2 CartServiceImpl

接口

|  |
| --- |
| **public interface** CartService {  **public void** addToCart(String skuId,String userId,Integer skuNum); } |

|  |
| --- |
| @Service **public class** CartServiceImpl **implements** CartService {   @Autowired  **private** CartInfoMapper **cartInfoMapper**;   @Reference  **private** ManageService **manageService**;   @Autowired  **private** RedisUtil **redisUtil**;  @Override  **public void** addToCart(String skuId,String userId,Integer skuNum) {  *// 先查cart中是否* CartInfo cartInfo = **new** CartInfo();  cartInfo.setSkuId(skuId);  cartInfo.setUserId(userId);  CartInfo cartInfoExist = **cartInfoMapper**.selectOne(cartInfo);  **if** (cartInfoExist!=**null**){  *// 更新商品数量* cartInfoExist.setSkuNum(cartInfoExist.getSkuNum()+skuNum);  *// 给实时价格赋值*  cartInfoExist.setSkuPrice(cartInfoExist.getCartPrice());  **cartInfoMapper**.updateByPrimaryKeySelective(cartInfoExist);  }**else**{  *// 如果不存在，保存购物车* SkuInfo skuInfo = **manageService**.getSkuInfo(skuId);  CartInfo cartInfo1 = **new** CartInfo();  cartInfo1.setSkuId(skuId);  cartInfo1.setCartPrice(skuInfo.getPrice());  cartInfo1.setSkuPrice(skuInfo.getPrice());  cartInfo1.setSkuName(skuInfo.getSkuName());  cartInfo1.setImgUrl(skuInfo.getSkuDefaultImg());  cartInfo1.setUserId(userId);  cartInfo1.setSkuNum(skuNum);  *// 插入数据库* **cartInfoMapper**.insertSelective(cartInfo1);  cartInfoExist = cartInfo1;  }  *// 构建key user:userid:cart* String userCartKey = CartConst.***USER\_KEY\_PREFIX***+userId+CartConst.***USER\_CART\_KEY\_SUFFIX***;  *// 准备取数据* Jedis jedis = **redisUtil**.getJedis();  *// 将对象序列化* String cartJson = JSON.*toJSONString*(cartInfoExist);  jedis.hset(userCartKey,skuId,cartJson);  *// 更新购物车过期时间* String userInfoKey = CartConst.***USER\_KEY\_PREFIX***+userId+CartConst.***USERINFOKEY\_SUFFIX***;  Long ttl = jedis.ttl(userInfoKey);  jedis.expire(userCartKey,ttl.intValue());  jedis.close();  } } |
| **定义常量**  **package** com.atguigu.gmall.cart.constant;  **public class** CartConst {   **public static final** String ***USER\_KEY\_PREFIX*** =**"user:"**;   **public static final** String ***USER\_CART\_KEY\_SUFFIX*** =**":cart"**;   **public static final** String ***USER\_CHECKED\_KEY\_SUFFIX*** =**":checked"**;   **public static final** String ***USERINFOKEY\_SUFFIX***=**":info"**;  } |

## 6 Web模块业务方法（gmall-cart-web）

### 1 思路：

1. 获得参数：skuId 、skuNum
2. 判断该用户是否登录，用userId判断
3. 如果登录则调用后台的service的业务方法
4. 如果未登录，要把购物车信息暂存到cookie中。
5. 实现利用cookie保存购物车的方法。

### 2 CartController

|  |
| --- |
| @Controller **public class** CartController {   @Reference  **private** CartService **cartService**;  @Autowired  **private** CartCookieHandler **cartCookieHandler**;  @Reference  **private** ManageService **manageService**;   @RequestMapping(**"addToCart"**)  @LoginRequie(autoRedirect = **false**)  **public** String addToCart(HttpServletRequest request, HttpServletResponse response){  *// 获取userId，skuId，skuNum* String skuId = request.getParameter(**"skuId"**);  String skuNum = request.getParameter(**"skuNum"**);   String userId = (String) request.getAttribute(**"userId"**);  *// 判断用户是否登录* **if** (userId!=**null**){  *// 说明用户登录* **cartService**.addToCart(skuId,userId,Integer.*parseInt*(skuNum));  }**else**{  *// 说明用户没有登录没有登录放到cookie中* **cartCookieHandler**.addToCart(request,response,skuId,userId,Integer.*parseInt*(skuNum));  }  *// 取得sku信息对象* SkuInfo skuInfo = **manageService**.getSkuInfo(skuId);  request.setAttribute(**"skuInfo"**,skuInfo);  request.setAttribute(**"skuNum"**,skuNum);  **return "success"**;  } } |

## 7 实现利用cookie保存购物车的CartCookieHandler

### 1 思路：

和service模块的方法类似

1. 先查询出来在cookie中的购物车，反序列化成列表。
2. 通过循环比较有没有该商品
3. 如果有，增加数量
4. 如果没有，增加商品
5. 然后把列表反序列化，利用之前最好的CookieUtil保存到cookie中。

### 2 CartCookieHandler

|  |
| --- |
| @Component **public class** CartCookieHandler {  *// 定义购物车名称* **private** String **cookieCartName** = **"CART"**;  *// 设置cookie 过期时间* **private int COOKIE\_CART\_MAXAGE**=7\*24\*3600;  @Reference  **private** ManageService **manageService**;   *// 未登录的时候，添加到购物车* **public void** addToCart(HttpServletRequest request, HttpServletResponse response, String skuId, String userId, Integer skuNum){  *//判断cookie中是否有购物车 有可能有中文，所有要进行序列化* String cartJson = CookieUtil.*getCookieValue*(request, **cookieCartName**, **true**);  List<CartInfo> cartInfoList = **new** ArrayList<>();  **boolean** ifExist=**false**;  **if** (cartJson!=**null**){  cartInfoList = JSON.*parseArray*(cartJson, CartInfo.**class**);  **for** (CartInfo cartInfo : cartInfoList) {  **if** (cartInfo.getSkuId().equals(skuId)){  cartInfo.setSkuNum(cartInfo.getSkuNum()+skuNum);  *// 价格设置* cartInfo.setSkuPrice(cartInfo.getCartPrice());  ifExist=**true**;  **break**;  }  }  }  *// //购物车里没有对应的商品 或者 没有购物车* **if** (!ifExist){  *//把商品信息取出来，新增到购物车* SkuInfo skuInfo = **manageService**.getSkuInfo(skuId);  CartInfo cartInfo=**new** CartInfo();   cartInfo.setSkuId(skuId);  cartInfo.setCartPrice(skuInfo.getPrice());  cartInfo.setSkuPrice(skuInfo.getPrice());  cartInfo.setSkuName(skuInfo.getSkuName());  cartInfo.setImgUrl(skuInfo.getSkuDefaultImg());   cartInfo.setUserId(userId);  cartInfo.setSkuNum(skuNum);  cartInfoList.add(cartInfo);  }  *// 把购物车写入cookie* String newCartJson = JSON.*toJSONString*(cartInfoList);  CookieUtil.*setCookie*(request,response,**cookieCartName**,newCartJson,**COOKIE\_CART\_MAXAGE**,**true**);  }  } |

### 3 添加成功后的展示页面

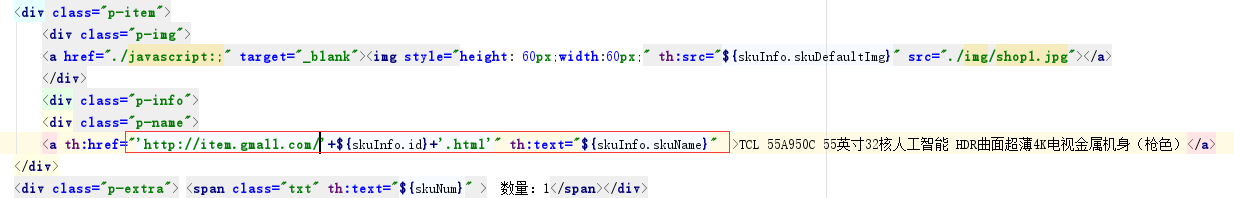
#### succuess.html

|  |
| --- |
| <div class="p-item">  <div class="p-img">  <a href="/javascript:;" target="\_blank"><img style="height: 60px;width:60px;" th:src="${skuInfo.skuDefaultImg}"></a>  </div>  <div class="p-info">  <div class="p-name">  <a th:href="'http://item.gmall.com/'+${skuInfo.id}+'.html'" th:title="${skuInfo.skuName}" th:text="${skuInfo.skuName}"> 商品名称</a> </div> <div class="p-extra"> <span class="txt" th:text="'数量:'+${num}"> 数量：XX</span></div> </div> |

需要注意的是：item.html 中表单提交需要给skuId 添加一个值

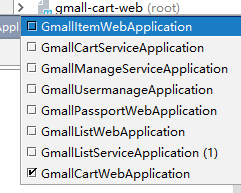
<**form id="itemForm" action="http://cart.gmall.com/addToCart" method="post"**>  
<**input type="text" name="skuNum" id="" value="1"** />  
<**input type="hidden" name="skuId" th:value="${skuInfo.id}"** />

</form>

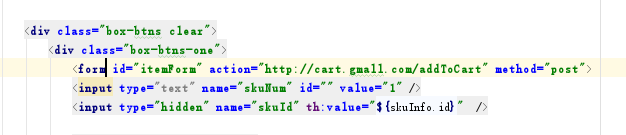


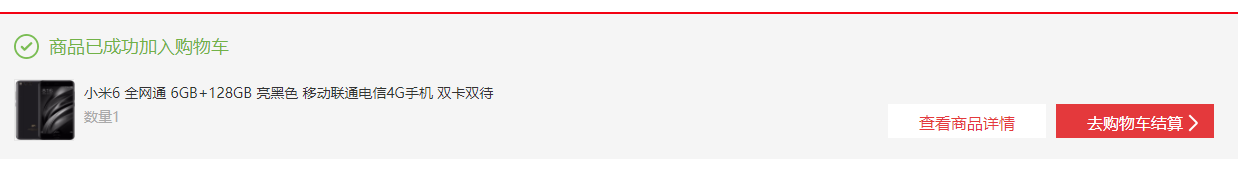
## 8 测试效果

List,manage,item,cart服务+页面都要启动



点击添加购物车的时候，修改表单中的值 item.html





# 四、功能—展示购物车列表

## 1 功能解析

1. 展示购物中的信息
2. 如果**用户已登录**从缓存中取值，如果缓存没有，加载数据库。
3. 如果**用户未登录**从cookie中取值。

## 2 CartController

|  |
| --- |
| @RequestMapping(**"cartList"**) @LoginRequie(autoRedirect = **false**) **public** String cartList(HttpServletRequest request,HttpServletResponse response){  *// 判断用户是否登录，登录了从redis中，redis中没有，从数据库中取  // 没有登录，从cookie中取得* String userId = (String) request.getAttribute(**"userId"**);  **if** (userId!=**null**){  *// 从redis中取得，或者从数据库中* List<CartInfo> cartList = **cartService**.getCartList(userId);  request.setAttribute(**"cartList"**,cartList);  }**else**{  List<CartInfo> cartList = **cartCookieHandler**.getCartList(request);  request.setAttribute(**"cartList"**,cartList);  }  **return "cartList"**; } |

cartCookieHandler中getCartList

|  |
| --- |
| */\*\*  \* 查询cookie 中购物车列表  \** ***@param request*** *\** ***@param userId*** *\** ***@return*** *\*/* **public** List<CartInfo> getCartList(HttpServletRequest request){  String cartJson = CookieUtil.*getCookieValue*(request, **cookieCartName**, **true**);  List<CartInfo> cartInfoList = JSON.*parseArray*(cartJson, CartInfo.**class**);  **return** cartInfoList; } |

## 3 CartServiceImpl

注意点：

1、redis中取出来要进行反序列化

2、redis的hash结构是无序的，要进行排序（可以用时间戳或者主键id，倒序排序）

3、如果redis中没有要从数据库中查询，要连带把最新的价格也取出来，默认要显示最新价格而不是当时放入购物车的价格，如果考虑用户体验可以把两者的差价提示给用户。

4、加载入缓存时一定要设定失效时间，保证和用户信息的失效时间一致即可。

接口

|  |
| --- |
| **// 缓存中没有数据，则从 数据库中加载**  **public** List<CartInfo> loadCartCache(String userId);  // 查询购物车集合列表 **public** List<CartInfo> getCartList(String userId); |

实现类

|  |
| --- |
| **public** List<CartInfo> getCartList(String userId){  *// 从redis中取得，* Jedis jedis = **redisUtil**.getJedis();  String userCartKey = CartConst.***USER\_KEY\_PREFIX***+userId+CartConst.***USER\_CART\_KEY\_SUFFIX***;  List<String> cartJsons = jedis.hvals(userCartKey);   **if** (cartJsons!=**null**&&cartJsons.size()>0){  List<CartInfo> cartInfoList = **new** ArrayList<>();  **for** (String cartJson : cartJsons) {  CartInfo cartInfo = JSON.*parseObject*(cartJson, CartInfo.**class**);  cartInfoList.add(cartInfo);  }  *// 排序* cartInfoList.sort(**new** Comparator<CartInfo>() {  @Override  **public int** compare(CartInfo o1, CartInfo o2) {  **return** Long.*compare*(Long.*parseLong*(o2.getId()),Long.*parseLong*(o1.getId()));  }  });  **return** cartInfoList;  }**else**{  *// 从数据库中查询，其中cart\_price 可能是旧值，所以需要关联sku\_info 表信息。* List<CartInfo> cartInfoList = loadCartCache(userId);  **return** cartInfoList;  } }  */\*\*  \* 购物车查询，在数据库中查找  \** ***@param userId*** *\** ***@return*** *\*/* **public** List<CartInfo> loadCartCache(String userId){  List<CartInfo> cartInfoList = **cartInfoMapper**.selectCartListWithCurPrice(userId);  **if** (cartInfoList==**null** && cartInfoList.size()==0){  **return null**;  }  String userCartKey = CartConst.***USER\_KEY\_PREFIX***+userId+CartConst.***USER\_CART\_KEY\_SUFFIX***;  Jedis jedis = **redisUtil**.getJedis();  Map<String,String> map = **new** HashMap<>(cartInfoList.size());  **for** (CartInfo cartInfo : cartInfoList) {  String cartJson = JSON.*toJSONString*(cartInfo);  *// key 都是同一个，值会产生重复覆盖！* map.put(cartInfo.getSkuId(),cartJson);  }  *// 将java list - redis hash* jedis.hmset(userCartKey,map);  jedis.close();  **return** cartInfoList; } |
| **Mapper接口**  **package** com.atguigu.gmall.cart.mapper;  **import** com.atguigu.gmall.bean.CartInfo; **import** tk.mybatis.mapper.common.Mapper;  **import** java.util.List;  **public interface** CartInfoMapper **extends** Mapper<CartInfo>{  List<CartInfo> selectCartListWithCurPrice(String userId); } |

CartInfoMapper.xml

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <!DOCTYPE mapper SYSTEM "http://mybatis.org/dtd/mybatis-3-mapper.dtd" *>* <mapper namespace="com.atguigu.gmall.cart.mapper.CartInfoMapper">  <select id ="selectCartListWithCurPrice" resultMap="cartMap">  SELECT c.\*,s.price FROM cart\_info c  INNER JOIN sku\_info s ON c.sku\_id=s.id WHERE c.user\_id=#{userId}  order by c.id desc  </select>  <resultMap id="cartMap" type="com.atguigu.gmall.bean.CartInfo" autoMapping="true">  <result property="id" column="id" ></result>  <result property="skuPrice" column="price" ></result>  </resultMap> </mapper> |

因为要取mapper.xml所以

application.properties中要加入

|  |
| --- |
| **mybatis.mapper-locations**=**classpath:mapper/\*Mapper.xml mybatis.configuration.mapUnderscoreToCamelCase**=**true** |

## 4 显示购物车列表页面

|  |
| --- |
| <**li th:each="cartInfo:${cartList}"**>  <**div**> </**div**>  <**div**>  <**ol**>  <**li**><**input type="checkbox" class="check" onchange="***checkSku*(**this**)**"** /></**li**>  <**li**>  <**dt**><**img th:src="${cartInfo.imgUrl}" src="img\shop1.jpg" alt=""**></**dt**>  <**dd th:onclick="'toItem('+${cartInfo.skuId}+')'" onclick="***toItem*(1)**"**>  <**p**>  <**span th:text="${cartInfo.skuName}"** >TCL 55A950C 55英寸32核</**span**>  </**p**>  </**dd**>  </**li**>  <**li**>  <**p class="dj" th:text="'￥'+${cartInfo.cartPrice}"** >￥4599.00</**p**>  </**li**>  <**li**>  <**p**>  <**span**>-</**span**>  <**span th:text="${cartInfo.skuNum}"**>5</**span**>  <**span**>+</**span**>  </**p**>  </**li**>  <**li style="font-weight**:**bold"**><**p class="zj" th:text="${cartInfo.skuNum}\*${cartInfo.cartPrice}"** >￥22995.00</**p**></**li**>  <**li**>  <**p**>删除</**p**>  </**li**>  </**ol**>  </**div**> </**li**> |

## 5 测试效果



# 五、功能--合并购物车

由于加入购物车时，用户可能存在登录和未登录两种情况，登录前在cookie中保存了一部分购物车信息，如果用户登录了，那么对应的要把cookie中的购物车合并到数据库中，并且刷新缓存。

思路：用数据库中的购物车列表与传递过来的cookie里的购物车列表循环匹配。

能匹配上的数量相加

匹配不上的插入到数据库中。

最后重新加载缓存

## 1 CartController

增加判断如果用户是登录状态的，但是cookie里却还有购物车，说明需要把cookie中的购物车合并进来，同时把cookie中的清空。

|  |
| --- |
| @RequestMapping(**"cartList"**) @LoginRequie(autoRedirect = **false**) **public** String cartList(HttpServletRequest request,HttpServletResponse response){  *// 判断用户是否登录，登录了从redis中，redis中没有，从数据库中取  // 没有登录，从cookie中取得* String userId = (String) request.getAttribute(**"userId"**);  **if** (userId!=**null**){  ***// 从cookie中查找购物车* List<CartInfo> cartListFromCookie = cartCookieHandler.getCartList(request);  List<CartInfo> cartList = null;  if (cartListFromCookie!=null && cartListFromCookie.size()>0){  *// 开始合并* cartList=cartService.mergeToCartList(cartListFromCookie,userId);  *// 删除cookie中的购物车* cartCookieHandler.deleteCartCookie(request,response);  }else{  *// 从redis中取得，或者从数据库中* cartList= cartService.getCartList(userId);  }**  request.setAttribute(**"cartList"**,cartList);  }**else**{  List<CartInfo> cartList = **cartCookieHandler**.getCartList(request);  request.setAttribute(**"cartList"**,cartList);  }  **return "cartList"**; } |

在CartCookieHandler中加入

|  |
| --- |
| **public void** deleteCartCookie(HttpServletRequest request,HttpServletResponse response){  CookieUtil.*deleteCookie*(request,response,**cookieCartName**); } |

## 2 CartServiceImpl

|  |
| --- |
| **public** List<CartInfo> mergeToCartList(List<CartInfo> cartListFromCookie, String userId){  List<CartInfo> cartInfoListDB = **cartInfoMapper**.selectCartListWithCurPrice(userId);  *// 循环开始匹配* **for** (CartInfo cartInfoCk : cartListFromCookie) {  **boolean** isMatch =**false**;  **for** (CartInfo cartInfoDB : cartInfoListDB) {  **if** (cartInfoDB.getSkuId().equals(cartInfoCk.getSkuId())){  cartInfoDB.setSkuNum(cartInfoCk.getSkuNum()+cartInfoDB.getSkuNum());  **cartInfoMapper**.updateByPrimaryKeySelective(cartInfoDB);  isMatch = **true**;  }  }  *// 数据库中没有购物车，则直接将cookie中购物车添加到数据库* **if** (!isMatch){  cartInfoCk.setUserId(userId);  **cartInfoMapper**.insertSelective(cartInfoCk);  }  }  *// 从新在数据库中查询并返回数据* List<CartInfo> cartInfoList = **loadCartCache(userId);**  **return** cartInfoList; } |

# 六、选中状态的变更

用户每次勾选购物车的多选框，都要把当前状态保存起来。由于可能会涉及更频繁的操作，所以这个勾选状态不必存储到数据库中。保留在缓存状态即可。

## 1 页面动作的js

注意jquery1.6版本以后要用prop(“checked”)取得多选框选中状态。

|  |
| --- |
| **function** *checkSku*( chkbox){  **console**.log(**$**(chkbox));  **var** skuId= **$**(chkbox).attr(**"value"**);  **var** checked=**$**(chkbox).prop(**"checked"**);  **var** isCheckedFlag=**"0"**;  **if**(checked){  isCheckedFlag=**"1"**;  }  **var** param=**"isChecked="**+isCheckedFlag+**"&"**+**"skuId="**+skuId;  **console**.log(param);  **$**.post(**"checkCart"**,param,**function** (data) {  *sumSumPrice*();  }); } |

在页面89行添加

<**li**><**input type="checkbox" class="check" th:value="${cartInfo.skuId}" onchange="***checkSku*(**this**)**"** /></**li**>

## 2 CartController

同样这里要区分，用户登录和未登录状态。

如果登录，修改缓存中的数据，如果未登录，修改cookie中的数据。

|  |
| --- |
| @RequestMapping(**"checkCart"**) @ResponseBody @LoginRequire(autoRedirect = **false**) **public void** checkCart(HttpServletRequest request,HttpServletResponse response){  String skuId = request.getParameter(**"skuId"**);  String isChecked = request.getParameter(**"isChecked"**);  String userId=(String) request.getAttribute(**"userId"**);  **if** (userId!=**null**){  **cartService**.checkCart(skuId,isChecked,userId);  }**else**{  **cartCookieHandler**.checkCart(request,response,skuId,isChecked);  } } |

## 3 CartServiceImpl

把对应skuId的购物车的信息从redis中取出来，反序列化，修改isChecked标志。

再保存回redis中。

同时保存另一个redis的key 专门用来存储用户选中的商品，方便结算页面使用。

|  |
| --- |
| **public void** checkCart(String skuId,String isChecked,String userId){  *// 更新购物车中的isChecked标志* Jedis jedis = **redisUtil**.getJedis();  *// 取得购物车中的信息* String userCartKey = CartConst.***USER\_KEY\_PREFIX***+userId+CartConst.***USER\_CART\_KEY\_SUFFIX***;  String cartJson = jedis.hget(userCartKey, skuId);  *// 将cartJson 转换成对象* CartInfo cartInfo = JSON.*parseObject*(cartJson, CartInfo.**class**);  cartInfo.setIsChecked(isChecked);  String cartCheckdJson = JSON.*toJSONString*(cartInfo);  jedis.hset(userCartKey,skuId,cartCheckdJson);  *// 新增到已选中购物车* String userCheckedKey = CartConst.***USER\_KEY\_PREFIX*** + userId + CartConst.***USER\_CHECKED\_KEY\_SUFFIX***;  **if** (isChecked.equals(**"1"**)){  jedis.hset(userCheckedKey,skuId,cartCheckdJson);  }**else**{  jedis.hdel(userCheckedKey,skuId);  }  jedis.close(); } |

CartCookieHandler中的checkCart

|  |
| --- |
| **public void** checkCart(HttpServletRequest request,HttpServletResponse response,String skuId,String isChecked){  *// 取出购物车中的商品* List<CartInfo> cartList = getCartList(request);  *// 循环比较* **for** (CartInfo cartInfo : cartList) {  **if** (cartInfo.getSkuId().equals(skuId)){  cartInfo.setIsChecked(isChecked);  }  }  *// 保存到cookie* String newCartJson = JSON.*toJSONString*(cartList);  CookieUtil.*setCookie*(request,response,**cookieCartName**,newCartJson,**COOKIE\_CART\_MAXAGE**,**true**); } |

## 4 修改页面isChecked的状态

在88行添加

|  |
| --- |
| <**li**><**input type="checkbox" class="check" th:value="${cartInfo.skuId}" onchange="***checkSku*(**this**)**" th:checked="(${cartInfo.isChecked}==1)?'true':'false'"** /></**li**> |

# 七、点击结算要做的收尾工作

要解决用户在未登录且购物车中有商品的情况下，直接点击结算。

所以不能直接跳到结算页面，要让用户强制登录后，检查cookie并进行合并后再重定向到结算页面

## 1 CartController

|  |
| --- |
| @RequestMapping(**"toTrade"**) @LoginRequire(autoRedirect = **true**) **public** String toTrade(HttpServletRequest request,HttpServletResponse response){  String userId = (String) request.getAttribute(**"userId"**);  List<CartInfo> cookieHandlerCartList = **cartCookieHandler**.getCartList(request);  **if** (cookieHandlerCartList!=**null** && cookieHandlerCartList.size()>0){  **cartService**.mergeToCartList(cookieHandlerCartList, userId);  **cartCookieHandler**.deleteCartCookie(request,response);  }  **return "redirect://order.gmall.com/trade"**; } |

合并购物车的时候，勾选状态可能会丢掉，因为DB中没有存储！

|  |
| --- |
| *在*mergeToCartList方法中新增  *// 从新在数据库中查询并返回数据* List<CartInfo> cartInfoList = loadCartCache(userId);  **for** (CartInfo cartInfo : cartInfoList) {  **for** (CartInfo info : cartListFromCookie) {  **if** (cartInfo.getSkuId().equals(info.getSkuId())){  *// 只有被勾选的才会进行更改* **if** (info.getIsChecked().equals(**"1"**)){  cartInfo.setIsChecked(info.getIsChecked());  *// 更新redis中的isChecked* checkCart(cartInfo.getSkuId(),info.getIsChecked(),userId);  }  }  } } **return** cartInfoList; |