**详细设计**

**1.编写目的：**

在前一阶段（概要设计说明书）中，已解决了实现该系统需求的程序模块设计问题。包括如何把该系统划分成若干个模块、决定各个模块之间的接口、模块之间传递的信息，以及数据结构、模块结构的设计等。在以下的详细设计报告中将对在本阶段中对系统所做的所有详细设计进行说明。

在本阶段中，确定应该如何具体地实现所要求的系统，从而在编码阶段可以把这个描述直接翻译成用具体的程序语言书写的程序。主要的工作有：根据在需求分析中所描述的数据、功能、运行、性能需求，并依照概要设计说明书所确定的处理流程、总体结构和模块外部设计，设计软件系统的结构设计、逐个模块的程序描述（包括各模块的功能、性能、输入、输出、算法、程序逻辑、接口等等），解决如何

1. 接受：管理员和用户登录信息；
2. 输出：订单信息、信息通知、商品信息；
3. 判断信息的正误并采取相应的处理步骤；
4. 进行数据库的查询、修改工作；
5. 接受并判断错误，输出相应的出错消息；

**2. 体系结构设计**

**（1）MVC框架**

         MVC(model,view,controller)，一种将业务逻辑、数据、界面分离的方法组织代码的框架。在改进界面及用户交互的同时，不需要重写业务逻辑。MVC将传统的输入、处理和输出分离。

Model：模型，逻辑部分，也是表示应用程序的核心，比如说数据库的表和记录

View：视图，数据显示，也是表示界面，是用于显示的，比如说显示数据库的记录

Controller：控制器，用户交互，也是表示处理的，比如说处理/增删查改数据库的记录

MVC中，视图可以有多个，在视图中没有真正的处理，视图只作为一种数据输出的形式，一个允许用户操作的平台；模型中的代码可以被多个视图使用；控制器通过接受用户的输入、调用模型、调用视图去完成用户的请求，控制器也不做处理，只做接收请求然后通知相关的模型，再确定用哪个视图来返回数据。

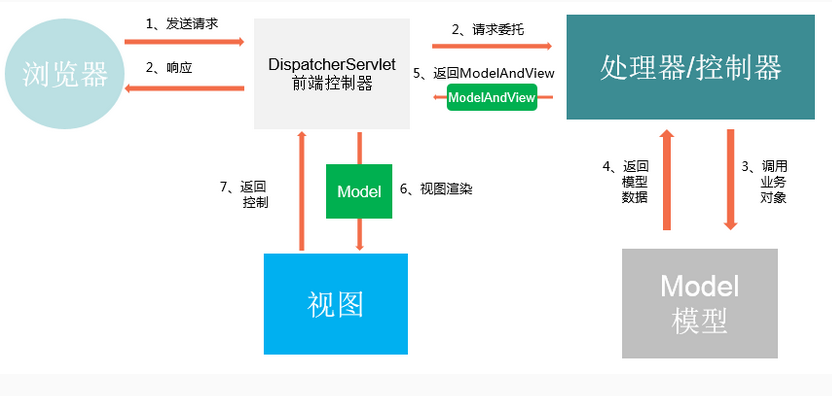
   MVC将视图层和业务层分离，很容易改变应用程序的数据层和业务规则，耦合性低且可维护性高。

   MVC中的模型可以被不同的视图使用，重用性高。

本项目主要用到了maven管理项目、spring-boot框架、以及mybatis框架，数据库用到了mysql。

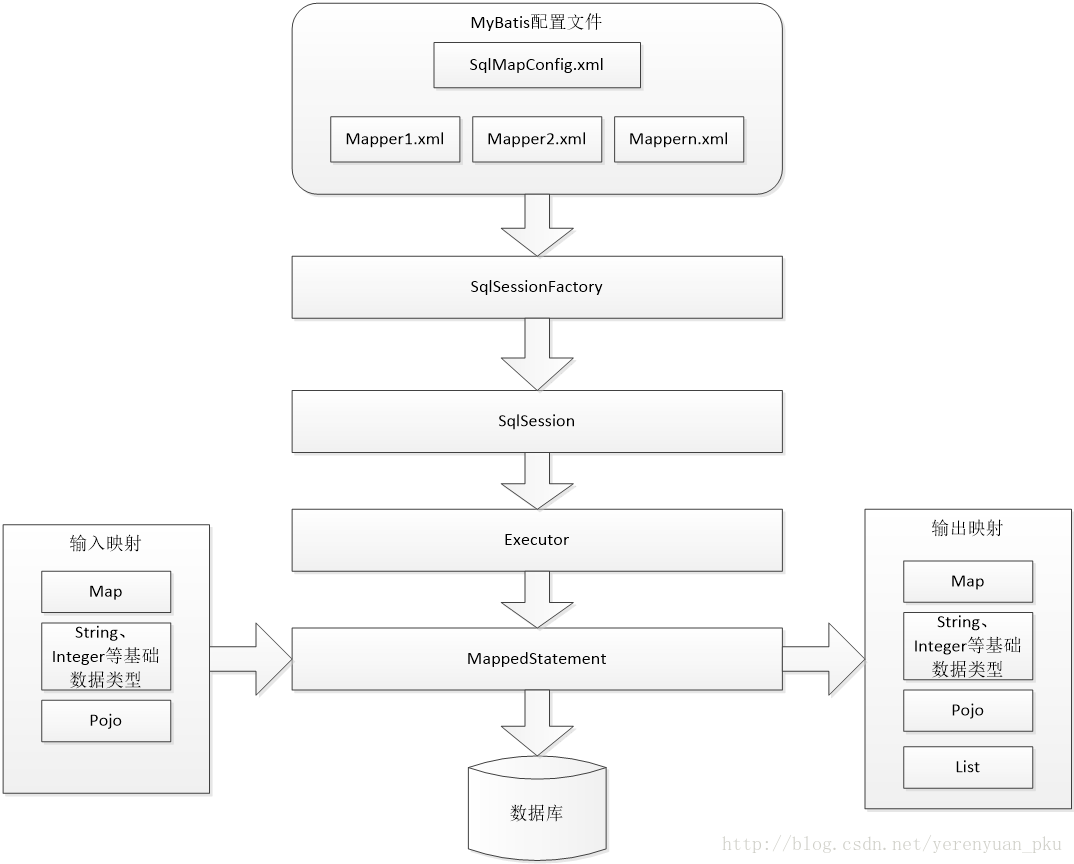
**（2）Spring Boot框架**

Spring Boot相比于spring-mvc封装了很多常用的依赖，并且内置了tomcat容器，启动生成的jar包即可启动项目，也是目前流行的微服务常用框架。，其设计目的是用来简化新Spring应用的初始搭建以及开发过程。该框架使用了特定的方式来进行配置，从而使开发人员不再需要定义样板化的配置。用我的话来理解，就是spring boot其实不是什么新的框架，它默认配置了很多框架的使用方式，就像maven整合了所有的jar包，spring boot整合了所有的框架。



**（3）mybatis框架**

MyBatis 是一款优秀的持久层框架，它支持定制化 SQL、存储过程以及高级映射。MyBatis 避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。MyBatis 可以使用简单的 XML 或注解来配置和映射原生信息，将接口和 Java 的 POJOs(Plain Old Java Objects,普通的 Java对象)映射成数据库中的记录。



**3.识别设计类**

（1）将边界类映射的设计类：

|  |  |
| --- | --- |
| 分析类 | 设计类 |
| AddressController | “设计类”AddressController |
| AdminController | “设计类”AdminController |
| CartController | “设计类”CartController |
| ChangeUserInfoController | “设计类”ChangeUserInfoController |
| FileController | “设计类”FileController |
| LoginController | “设计类”LoginController |
| MessageController | “设计类”MessageController |
| OrderController | “设计类”OrderController |
| ProductController | “设计类”ProductController |
| RegisterController | “设计类”RegisterController |
| ShowProductController | “设计类”ShowProductController |

（2）将控制类映射的设计类：

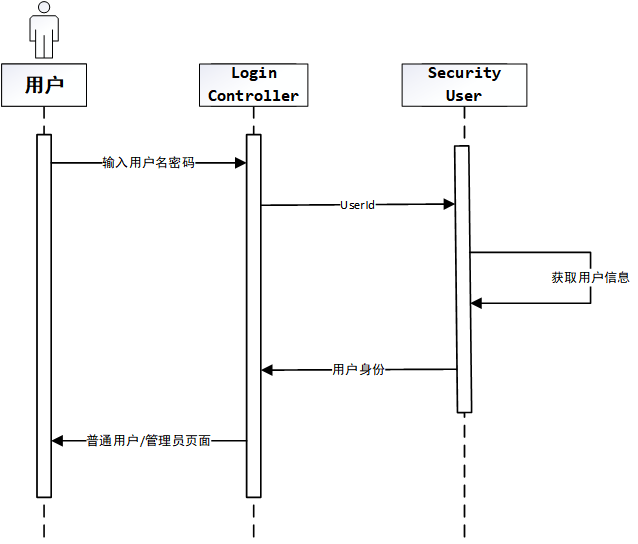
|  |  |
| --- | --- |
| 分析类 | 设计类 |
| AddressService | “设计类”AddressService |
| CartService | “设计类”CartService |
| CategoryService | “设计类”CategoryService |
| ChangeUserInfoService | “设计类”ChangeUserInfoService |
| CommenService | “设计类”CommenService |
| CustomUserDetailsService | “设计类”CustomUserDetailsService |
| FileService | “设计类”FileService |
| ManagerService | “设计类”ManagerService |
| MessageService | “设计类”MessageService |
| OrderService | “设计类”OrderService |
| ProductService | “设计类”ProductService |
| RegisterService | “设计类”RegisterService |

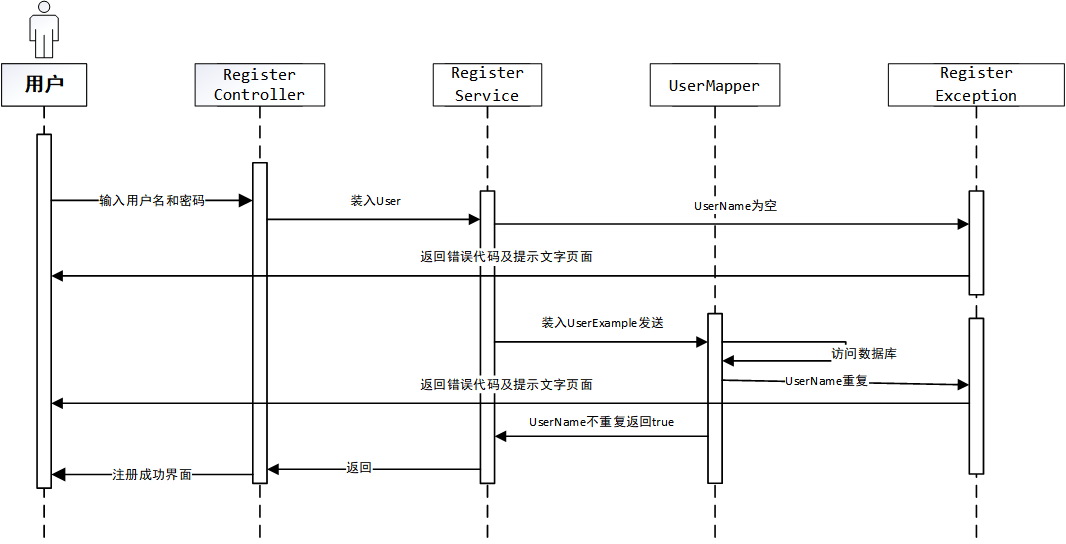
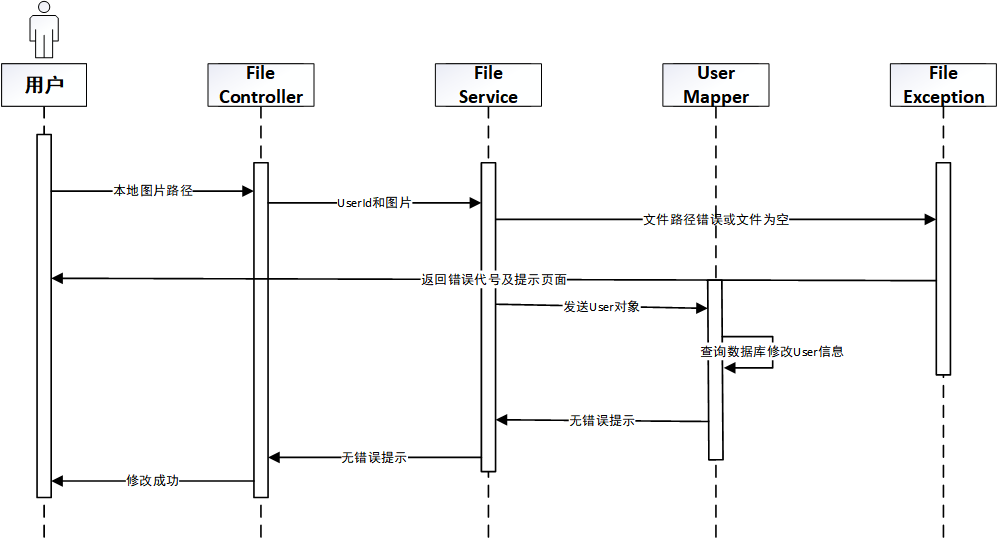
（3）将实体类映射的设计类：

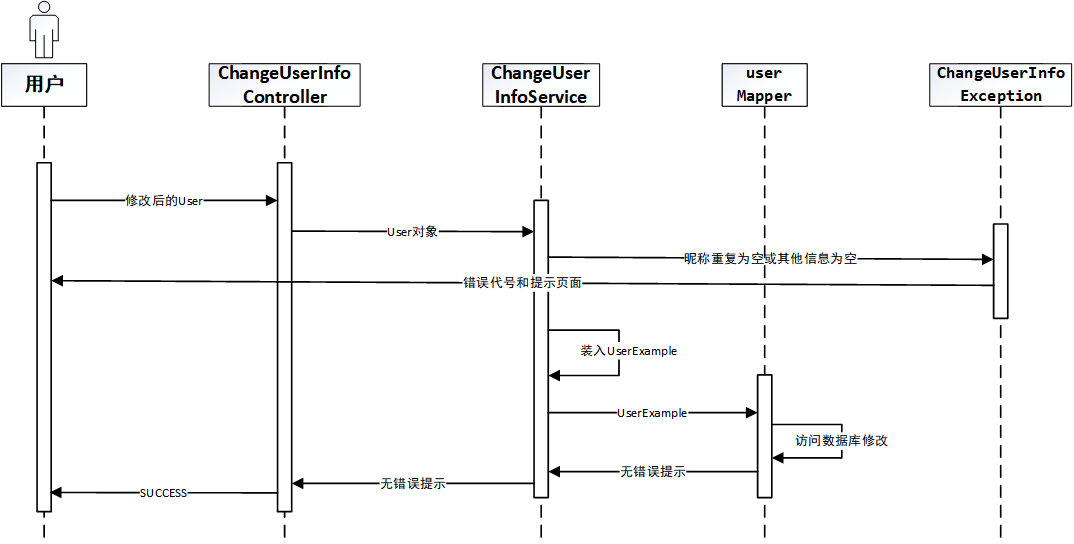
|  |  |
| --- | --- |
| 分析类 | 设计类 |
| Address | “设计类”Address |
| AddressExample | “设计类”AddressExample |
| Cart | “设计类”Cart |
| CartExample | “设计类”CartExample |
| Comment | “设计类”Comment |
| CommentExample | “设计类”CommentExample |
| FirstCategory | “设计类”FirstCategory |
| FirstCategoryExample | “设计类”FirstCategoryExample |
| Image | “设计类”Image |
| ImageExample | “设计类”ImageExample |
| Message | “设计类”Message |
| MessageExample | “设计类”MessageExample |
| Order | “设计类”Order |
| OrderExample | “设计类”OrderExample |
| OrderItem | “设计类”OrderItem |
| OrderItemExample | “设计类”OrderItemExample |
| OrderItemKey | “设计类”OrderItemKey |
| Product | “设计类”Product |
| ProductExample | “设计类”ProductExample |
| ProductWithImage | “设计类”ProductWithImage |
| SecondCategory | “设计类”SecondCategory |
| SecondCategoryExample | “设计类”SecondCategoryExample |
| SecurityUser | “设计类”SecurityUser |
| User | “设计类”User |
| UserExample | “设计类”UserExample |

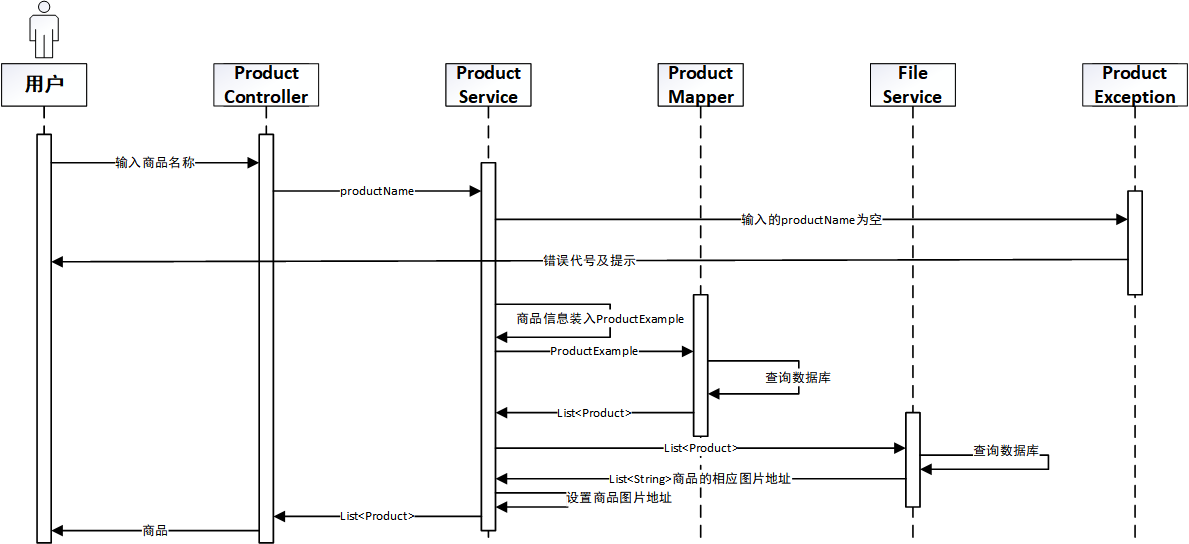
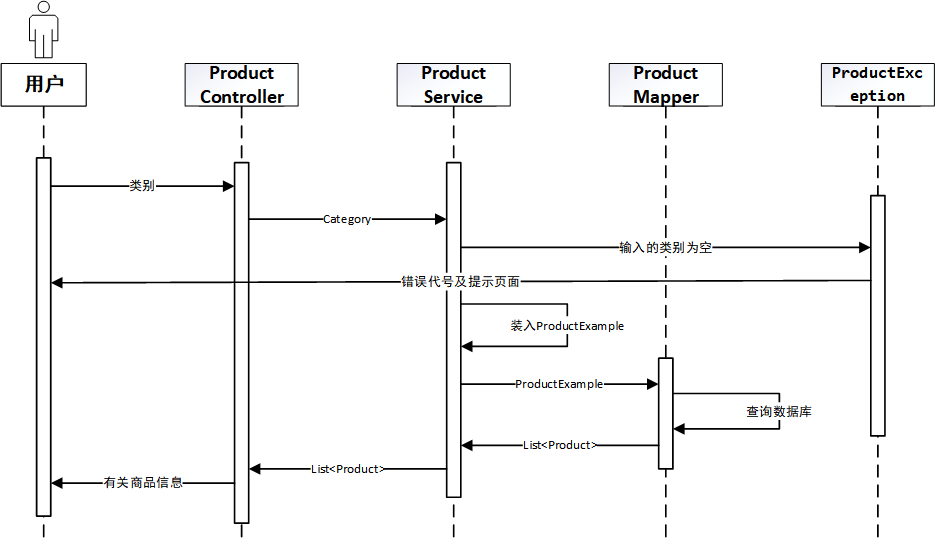
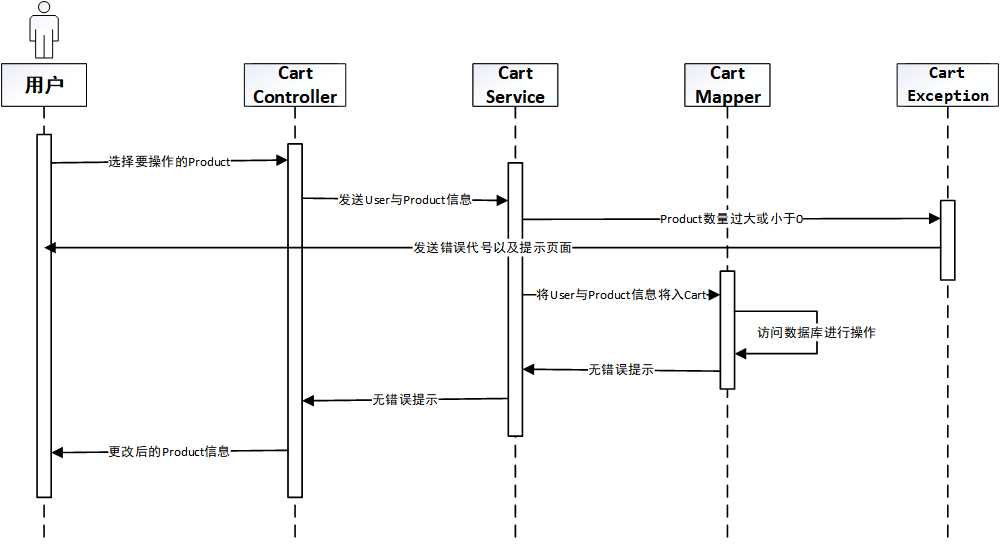
**4、各流程模块**

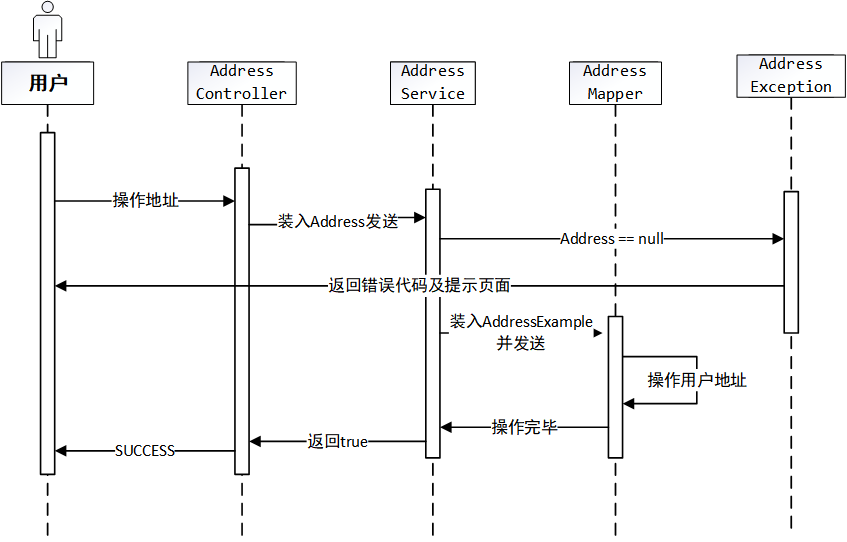
（1）用户登录



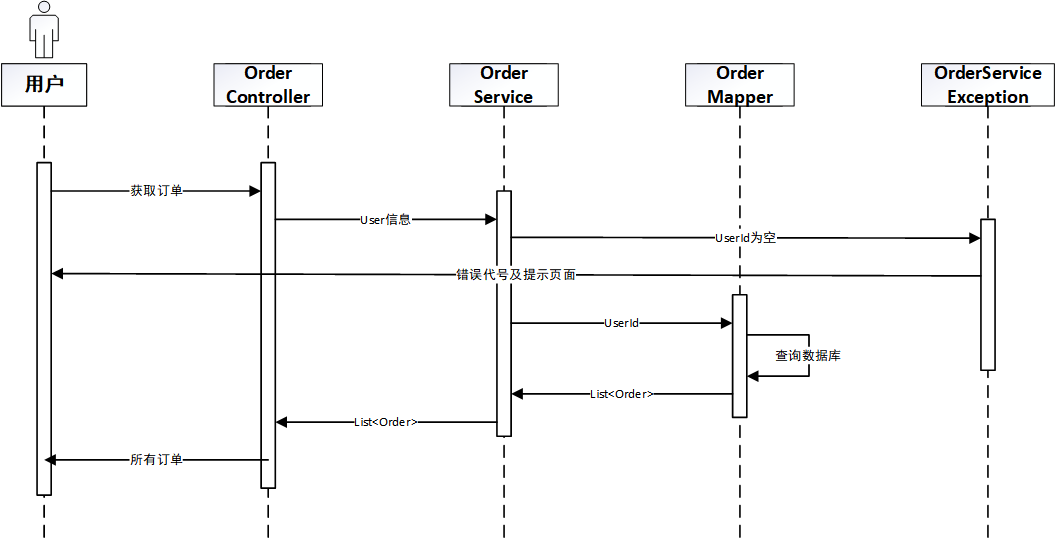
1. 用户注册
2. 用户上传或修改头像
3. 用户修改信息



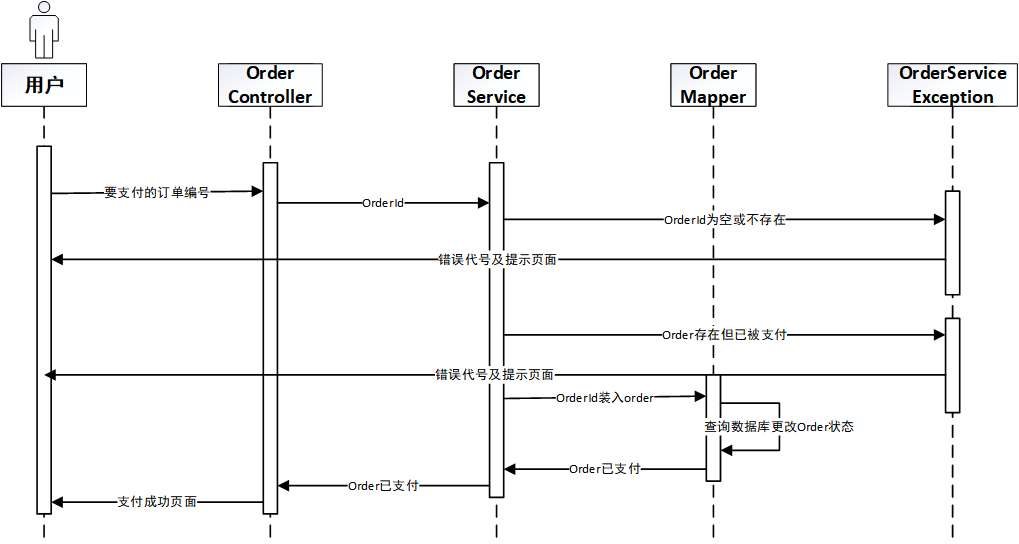
1. 根据商品名字查询商品
2. 根据商品类别查询商品
3. 添加购物车与修改商品数量
4. 用户修改或添加地址



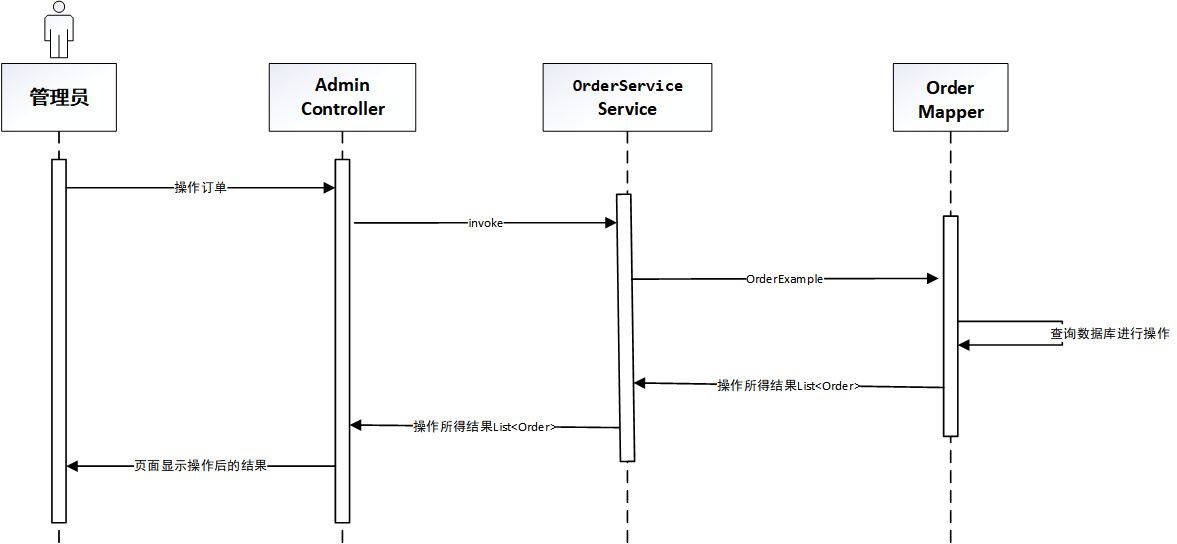
1. 用户获取订单

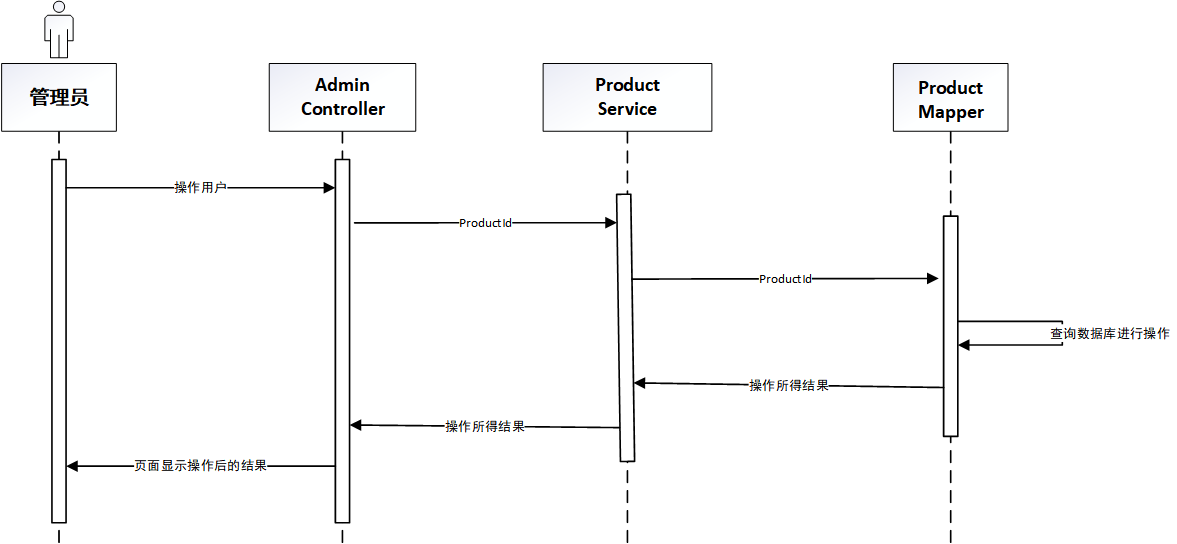


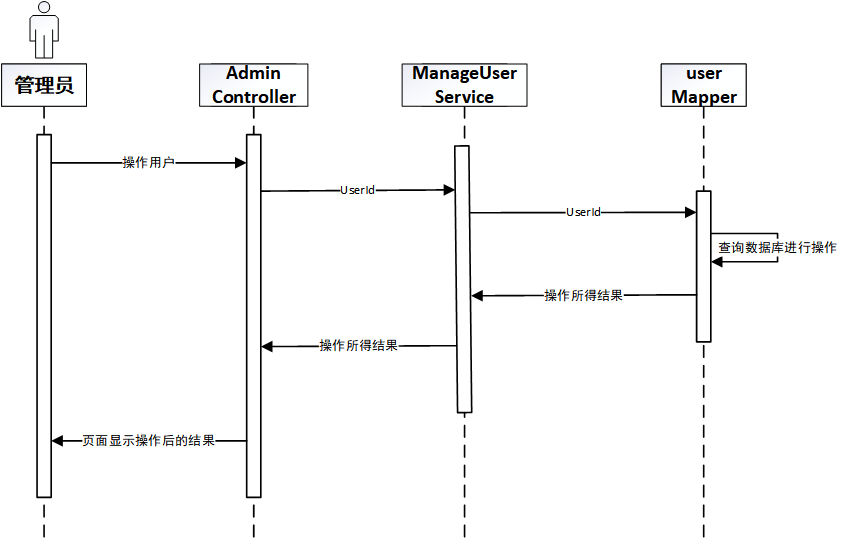
（10）订单支付



（11）获取所有订单



1. 商品管理（增删改查）
2. 用户管理（增删改查）



**5.关键代码**

**（1）用户注册Service：**

1. @Service
2. **public** **class** RegisterService {

5. /\*\*
6. \* 密码加密工具类
7. \*/
8. **private** PasswordEncoder passwordEncoder = **new** BCryptPasswordEncoder(10);
10. @Autowired
11. **private** SqlSessionFactory sqlSessionFactory;
13. /\*\*
14. \*
15. \* @param user 要注册的新用户，不能传入为null
16. \*/
17. **public** **void** register(User user) {
19. //如果传入user.user\_id为空
20. **if** (**null** == user || **null** == user.getUserId()||**null**==user.getUserPassword()||**null**==user.getUserName()){
22. **throw** **new** RegisterException(RegisterException.INVALID\_INPUT\_MESSAGE,RegisterException.INVALID\_INPUT);
24. }
26. SqlSession sqlSession = sqlSessionFactory.openSession();
27. UserMapper userMapper = sqlSession.getMapper(UserMapper.**class**);
29. //看看用户email是否已经存在,如果已经存在
30. **if** (**null** != userMapper.selectByPrimaryKey(user.getUserId())) {
32. **throw** **new** RegisterException(RegisterException.EMAIL\_HAS\_EXIST\_MESSAGE,RegisterException.EMAIL\_HAS\_EXIST);
33. }
35. //看看userName是否已经存在
36. UserExample userExample = **new** UserExample();
37. userExample.createCriteria().andUserNameEqualTo(user.getUserName());
38. **if** (!userMapper.selectByExample(userExample).isEmpty()){
39. **throw** **new** RegisterException(RegisterException.USERNAME\_HAS\_EXIST\_MESSAGE,RegisterException.USERNAME\_HAS\_EXIST);
40. }
42. //设置角色为普通用户
43. user.setRole(**false**);
44. //设置头像路径为
45. user.setHeadImg("boy.jpg");
46. //将用户密码进行加密后，保存到数据库中
47. user.setUserPassword(passwordEncoder.encode(user.getUserPassword()));
48. userMapper.insert(user);

51. sqlSession.close();
53. **return**;
54. }
56. }

**（2）订单管理service**

1. @Service
2. **public** **class** OrderService {
4. **public** **static** Integer STATUS\_COUNT = 5;
6. **public** **static** String ORDER\_STATUS\_NO\_PAYMENT = "待付款";
7. **public** **static** String ORDER\_STATUS\_HAS\_PAYMENT = "已付款";
8. **public** **static** String ORDER\_STATUS\_ON\_WAY = "正在运送";
9. **public** **static** String ORDER\_STATUS\_CANCELED = "已取消";
10. **public** **static** String ORDER\_STATUS\_FINISH = "已完成";
11. @Autowired
12. **private** SqlSessionFactory sqlSessionFactory;
14. @Autowired
15. **private** ProductService productService;
17. /\*\*
18. \* 将选中的商品放入订单中，并对购物车和库存进行相应操作
19. \*
20. \* @param userId
21. \* @param productIds
22. \*/
23. @Transactional
24. **public** **void** createOrderByProductId(String userId, String[] productIds, String addressId) {
26. //检查输入
27. **if** (**null** == userId || **null** == productIds || **null** == addressId) {
28. **throw** **new** OrderServiceException(INVALID\_INPUT\_MESSAGE, OrderServiceException.INVALID\_INPUT);
29. }
31. //已经选中的商品id
32. List<String> listProductsId = Arrays.asList(productIds);
34. SqlSession sqlSession = sqlSessionFactory.openSession();
35. CartMapper cartMapper = sqlSession.getMapper(CartMapper.**class**);
37. //获取选中的购物车中的商品信息
38. CartExample cartExample = **new** CartExample();
39. //构建条件
40. cartExample.createCriteria().andUserIdEqualTo(userId)
41. .andProductIdIn(listProductsId);
42. List<Cart> carts = cartMapper.selectByExample(cartExample);
44. ProductMapper productMapper = sqlSession.getMapper(ProductMapper.**class**);
45. //更新数据库中商品的数量
46. Product product;
47. **for** (Cart cart :
48. carts) {
50. //查找商品表中对应的商品
51. product = productMapper.selectByPrimaryKey(cart.getProductId());
52. //判断是否还有库存
53. **if** (**null** == product || product.getProductLeftTotals() < cart.getTotals()) {
54. //抛出库存不足异常
55. **throw** **new** OrderServiceException(OrderServiceException.NO\_ENOUGH\_PRODUCT\_MESSAGE, OrderServiceException.NO\_ENOUGH\_PRODUCT);
56. }
57. //更新商品数量
58. product.setProductLeftTotals(product.getProductLeftTotals() - cart.getTotals());
59. productMapper.updateByPrimaryKeySelective(product);
61. }
63. //生成订单并存入数据库
64. OrderMapper orderMapper = sqlSession.getMapper(OrderMapper.**class**);
65. Order order = **new** Order();
66. order.setAddressId(addressId);
67. order.setOrderDate(**new** Date());
68. String orderId = UUID.randomUUID().toString();
69. order.setOrderId(orderId);//订单Id，由UUID生成
70. order.setOrderStatus(ORDER\_STATUS\_NO\_PAYMENT);//支付状态，默认待付款
71. order.setUserId(userId);
72. orderMapper.insert(order);
74. //存入订单商品
75. OrderItem orderItem;
76. OrderItemMapper orderItemMapper = sqlSession.getMapper(OrderItemMapper.**class**);
77. **for** (Cart cart :
78. carts) {
80. orderItem = **new** OrderItem();
81. orderItem.setProductCount(cart.getTotals());
82. orderItem.setOrderId(orderId);
83. orderItem.setProductId(cart.getProductId());
85. orderItemMapper.insert(orderItem);
86. }
88. //删除购物车中已经选中的商品
89. cartMapper.deleteByExample(cartExample);
91. sqlSession.close();
92. }

95. /\*\*
96. \* 普通用户取消自己的订单，如果订单已经运送、取消或者完成，则不能取消
97. \*
98. \* @param userId
99. \* @param orderId
100. \*/
101. @Transactional
102. **public** **void** cancelOrderByOrderIdandUserId(String userId, String orderId) {
104. //判断数据的合法性
105. **if** (**null** == userId || **null** == orderId) {
106. **throw** **new** OrderServiceException(INVALID\_INPUT\_MESSAGE, OrderServiceException.INVALID\_INPUT);
107. }
109. SqlSession sqlSession = sqlSessionFactory.openSession();
110. OrderMapper orderMapper = sqlSession.getMapper(OrderMapper.**class**);
111. //查询订单
112. Order order = orderMapper.selectByPrimaryKey(orderId);
114. //判断订单是否可以取消，在运送中或者已经取消、完成的订单不能取消
115. **if** (ORDER\_STATUS\_CANCELED.equals(order.getOrderStatus()) ||
116. ORDER\_STATUS\_ON\_WAY.equals(order.getOrderStatus()) ||
117. ORDER\_STATUS\_FINISH.equals(order.getOrderStatus())) {
118. **throw** **new** OrderServiceException(OrderServiceException.ORDER\_CANNT\_CANCEL\_MESSAGE, OrderServiceException.ORDER\_CANNT\_CANCEL);
119. }
121. //如果userId不一致，则订单不可以取消，因为没有权限
122. **if** (!userId.equals(order.getUserId())) {
123. **throw** **new** OrderServiceException(OrderServiceException.NO\_AUTHORITY\_MESSAGE, OrderServiceException.NO\_AUTHORITY);
124. }
126. //如果已经付款
127. **if** (ORDER\_STATUS\_HAS\_PAYMENT.equals(order.getOrderStatus())) {
128. //FIXME:退款
129. System.out.println("退款");
130. }
132. //设置订单状态为取消
133. order.setOrderStatus(ORDER\_STATUS\_CANCELED);
134. orderMapper.updateByPrimaryKeySelective(order);
136. sqlSession.close();
138. **return**;
139. }

142. /\*\*
143. \* 根据用户id查找对应状态的订单
144. \*
145. \* @param userId
146. \* @param statuses
147. \* @return
148. \*/
149. **public** List<Order> searchAllOrderByUserIdAndOrderStatus(String userId, String[] statuses) {
151. //检查输入
152. **if** (**null** == userId || **null** == statuses) {
153. **throw** **new** OrderServiceException(INVALID\_INPUT\_MESSAGE, OrderServiceException.INVALID\_INPUT);
154. }
156. List<String> arrayStatuses = Arrays.asList(statuses);
158. SqlSession sqlSession = sqlSessionFactory.openSession();
159. OrderMapper orderMapper = sqlSession.getMapper(OrderMapper.**class**);
161. //查找对应的订单
162. OrderExample orderExample = **new** OrderExample();
163. orderExample.createCriteria().andUserIdEqualTo(userId).andOrderStatusIn(arrayStatuses);
164. List<Order> orders = orderMapper.selectByExample(orderExample);
166. //判断结果是否为空
167. **if** (orders.isEmpty()) {
168. **throw** **new** OrderServiceException(OrderServiceException.NO\_ORDERS\_MESSAGE, OrderServiceException.NO\_ORDERS);
169. }
171. sqlSession.close();
172. **return** orders;
173. }
175. /\*\*
176. \* 管理员接收用户的订单（当前仅当订单已付款），订单状态变为“正在运送”状态
177. \*
178. \* @param orderId
179. \*/
180. **public** **void** acceptOrderByOrderId(String orderId) {
182. //检查输入
183. **if** (**null** == orderId) {
185. **throw** **new** OrderServiceException(INVALID\_INPUT\_MESSAGE, OrderServiceException.INVALID\_INPUT);
186. }
188. SqlSession sqlSession = sqlSessionFactory.openSession();
189. OrderMapper orderMapper = sqlSession.getMapper(OrderMapper.**class**);
191. //查询订单
192. Order order = orderMapper.selectByPrimaryKey(orderId);
193. **if** (**null** == order) {
194. **throw** **new** OrderServiceException(OrderServiceException.NO\_ORDERS\_MESSAGE, OrderServiceException.NO\_ORDERS);
195. }
197. //如果订单状态不是已付款状态，则抛出异常
198. **if** (!ORDER\_STATUS\_HAS\_PAYMENT.equals(order.getOrderStatus())) {
200. **throw** **new** OrderServiceException(OrderServiceException.NO\_PAY\_MESSAGE, OrderServiceException.NO\_PAY);
201. }
203. //更新订单状态为正在运送
204. order.setOrderStatus(ORDER\_STATUS\_ON\_WAY);
205. orderMapper.updateByPrimaryKeySelective(order);
207. sqlSession.close();
208. **return**;
209. }
211. /\*\*
212. \* 管理员取消用户订单，订单状态变为"已取消"状态
213. \*
214. \* @param orderId TODO:未完成
215. \*/
216. **public** **void** cancelOrderByOrderId(String orderId) {
217. //检查输入
218. **if** (**null** == orderId) {
219. **throw** **new** OrderServiceException(INVALID\_INPUT\_MESSAGE, OrderServiceException.INVALID\_INPUT);
220. }
222. SqlSession sqlSession = sqlSessionFactory.openSession();
223. OrderMapper orderMapper = sqlSession.getMapper(OrderMapper.**class**);
225. //查询订单
226. Order order = orderMapper.selectByPrimaryKey(orderId);
227. **if** (**null** == order) {
228. **throw** **new** OrderServiceException(OrderServiceException.NO\_ORDERS\_MESSAGE, OrderServiceException.NO\_ORDERS);
229. }
231. //判断订单是否可以取消，在运送中或者已经取消、完成的订单不能取消
232. **if** (ORDER\_STATUS\_CANCELED.equals(order.getOrderStatus()) ||
233. ORDER\_STATUS\_ON\_WAY.equals(order.getOrderStatus()) ||
234. ORDER\_STATUS\_FINISH.equals(order.getOrderStatus())) {
235. **throw** **new** OrderServiceException(OrderServiceException.ORDER\_CANNT\_CANCEL\_MESSAGE, OrderServiceException.ORDER\_CANNT\_CANCEL);
236. }
238. //如果已经付款
239. **if** (ORDER\_STATUS\_HAS\_PAYMENT.equals(order.getOrderStatus())) {
240. //FIXME:退款
241. System.out.println("退款");
242. }
244. //设置订单状态为取消
245. order.setOrderStatus(ORDER\_STATUS\_CANCELED);
246. orderMapper.updateByPrimaryKeySelective(order);
247. sqlSession.close();
248. }
250. /\*\*
251. \* 根据订单id获取订单项
252. \*
253. \* @param orderId
254. \* @return
255. \*/
256. **public** List<Product> getOrderItemByOrderId(String orderId) {
258. //检查输入
259. **if** (**null** == orderId) {
260. **throw** **new** OrderServiceException(INVALID\_INPUT\_MESSAGE, INVALID\_INPUT);
261. }
263. SqlSession sqlSession = sqlSessionFactory.openSession();
264. OrderItemMapper orderItemMapper = sqlSession.getMapper(OrderItemMapper.**class**);
266. OrderItemExample orderItemExample = **new** OrderItemExample();
267. orderItemExample.createCriteria().andOrderIdEqualTo(orderId);
269. List<OrderItem> orderItems = orderItemMapper.selectByExample(orderItemExample);
271. //查找商品
272. List<Product> products = **new** ArrayList<>();
273. Product product;
274. **for** (OrderItem orderItem :
275. orderItems) {
277. product = productService.searchProductByID(orderItem.getProductId());
278. product.setProductLeftTotals(orderItem.getProductCount());
280. products.add(product);
281. }

284. sqlSession.close();
286. **return** products;

289. }
291. /\*\*
292. \* 根据用户id查找订单
293. \*
294. \* @param userId
295. \* @return
296. \*/
297. **public** List<Order> searchAllOrderByUserId(String userId) {
299. //检查输入
300. **if** (**null** == userId) {
301. **throw** **new** OrderServiceException(INVALID\_INPUT\_MESSAGE, OrderServiceException.INVALID\_INPUT);
302. }
304. SqlSession sqlSession = sqlSessionFactory.openSession();
305. OrderMapper orderMapper = sqlSession.getMapper(OrderMapper.**class**);
307. //查找对应的订单
308. OrderExample orderExample = **new** OrderExample();
309. orderExample.createCriteria().andUserIdEqualTo(userId);
310. List<Order> orders = orderMapper.selectByExample(orderExample);
312. sqlSession.close();
313. **return** orders;
314. }
316. /\*\*
317. \* 管理员用于获取所有订单
318. \*
319. \* @return
320. \*/
321. **public** List<Order> getAllOrder() {
323. SqlSession sqlSession = sqlSessionFactory.openSession();
324. OrderMapper orderMapper = sqlSession.getMapper(OrderMapper.**class**);
326. List<Order> orders = orderMapper.selectByExample(**new** OrderExample());
328. sqlSession.close();
329. **return** orders;
330. }
332. /\*\*
333. \* 根据传入订单状态数组搜索符合状态的订单
334. \*
335. \* @param statuses
336. \* @return
337. \*/
338. **public** List<Order> searchOrderByOrderStatus(String[] statuses) {

341. //检查输入
342. **if** (**null** == statuses || 0 == statuses.length) {
344. **throw** **new** OrderServiceException(INVALID\_INPUT\_MESSAGE, INVALID\_INPUT);
345. }
347. List<String> statusList = Arrays.asList(statuses);
349. SqlSession sqlSession = sqlSessionFactory.openSession();
350. OrderMapper orderMapper = sqlSession.getMapper(OrderMapper.**class**);
352. OrderExample orderExample = **new** OrderExample();
353. orderExample.createCriteria().andOrderStatusIn(statusList);
355. List<Order> orders = orderMapper.selectByExample(orderExample);

358. sqlSession.close();
359. **return** orders;
361. }
363. /\*\*
364. \* 传入orderId，将订单变为已支付状态
365. \*
366. \* @param orderId
367. \*/
368. **public** **void** payOrderByOrderId(String orderId) {
370. //检查输入
371. **if** (**null** == orderId) {
373. **throw** **new** OrderServiceException(INVALID\_INPUT\_MESSAGE, INVALID\_INPUT);
374. }
376. SqlSession sqlSession = sqlSessionFactory.openSession();
377. OrderMapper orderMapper = sqlSession.getMapper(OrderMapper.**class**);
379. Order order = orderMapper.selectByPrimaryKey(orderId);
381. //判断订单是否存在
382. **if** (**null** == order) {
383. **throw** **new** OrderServiceException(OrderServiceException.NO\_ORDERS\_MESSAGE, OrderServiceException.NO\_ORDERS);
384. }
386. //判断订单是否未支付
387. **if** (!ORDER\_STATUS\_NO\_PAYMENT.equals(order.getOrderStatus())){
388. **throw** **new** OrderServiceException(OrderServiceException.NO\_UNPAY\_MESSAGE,OrderServiceException.NO\_UNPAY);
389. }
391. //更新订单状态为已支付
392. order.setOrderStatus(ORDER\_STATUS\_HAS\_PAYMENT);
393. orderMapper.updateByPrimaryKey(order);
395. sqlSession.close();
397. **return**;
398. }
399. }

**（3）购物车service**

1. @Service
2. **public** **class** CartService {
4. **private** **final** SqlSessionFactory sqlSessionFactory;
6. @Autowired
7. **public** CartService(SqlSessionFactory sqlSessionFactory) {
8. **this**.sqlSessionFactory = sqlSessionFactory;
9. }
11. /\*\*
12. \* 函数会检查所有参数是否为空
13. \* 若为空，则不合法
14. \* 用户必须可以在database中被找到
15. \*
16. \* @param user 用户
17. \*/
18. **private** **void** checkParams(User user) {
19. // 数据验证
20. **if** (user == **null**) {
21. **throw** **new** CartException(USER\_EMPTY\_MESSAGE, USER\_EMPTY);
22. }
24. SqlSession sqlSession = sqlSessionFactory.openSession();
25. UserMapper userMapper = sqlSession.getMapper(UserMapper.**class**);
27. // 检查用户是否合法
28. String userID = user.getUserId();
29. **if** (userID == **null** || userID.equals("")) {
30. sqlSession.close();
31. **throw** **new** CartException(USER\_EMPTY\_MESSAGE, USER\_EMPTY);
32. }
33. User userInDB = userMapper.selectByPrimaryKey(userID);
34. **if** (userInDB == **null**) {
35. sqlSession.close();
36. **throw** **new** CartException(USER\_EMPTY\_MESSAGE, USER\_EMPTY);
37. }
39. sqlSession.close();
40. }
42. /\*\*
43. \* 检查用户和商品id是否为空
44. \* 用户在数据库中必须存在👎商品id不为空
45. \*
46. \* @param user          用户
47. \* @param productIDList 商品id列表
48. \*/
49. **private** **void** checkParams(User user, List<String> productIDList) {
50. checkParams(user);
52. **if** (productIDList == **null**) {
53. **throw** **new** CartException(PRODUCT\_EMPTY\_MESSAGE, PRODUCT\_EMPTY);
54. }
55. }
57. /\*\*
58. \* 检查用户和商品id是否为空
59. \* 用户在数据库中必须存在👎商品id不为空
60. \* 若商品id数量不等于购买数量 则报错
61. \*
62. \* @param user          用户
63. \* @param productIDList 商品id列表
64. \* @param numberList    加入购物车数量
65. \*/
66. **private** **void** checkParams(User user, List<String> productIDList, List<Integer> numberList) {
67. checkParams(user, productIDList);
69. **if** (numberList == **null**) {
70. **throw** **new** CartException(NUMBMER\_EMPTY\_MESSAGE, NUMBER\_EMPTY);
71. }
72. **if** (productIDList.size() != numberList.size()) {
73. **throw** **new** CartException(PRODUCT\_NUMBER\_NOT\_MATCH\_MESSAGE, PRODUCT\_NUMBER\_NOT\_MATCH);
74. }
75. }
77. /\*\*
78. \* 向购物车中加入一定数量的商品
79. \*
80. \* @param user          用户
81. \* @param productIDList 商品id列表
82. \* @param numberList    购买数量列表
83. \*/
84. @Transactional
85. **public** **void** insertProducts(User user, List<String> productIDList, List<Integer> numberList) {
87. checkParams(user, productIDList, numberList);
89. String userID = user.getUserId();
90. SqlSession sqlSession = sqlSessionFactory.openSession();
91. ProductMapper productMapper = sqlSession.getMapper(ProductMapper.**class**);
92. CartMapper cartMapper = sqlSession.getMapper(CartMapper.**class**);
94. //  更新数据库操作
95. **for** (**int** i = 0; i < productIDList.size(); i++) {
96. String productID = productIDList.get(i);
97. Integer number = numberList.get(i);
99. // 如果商品id为空，则忽略
100. **if** (productID == **null** || number == **null** || productID.equals("")) {
101. **continue**;
102. }
103. Product product = productMapper.selectByPrimaryKey(productID);
104. **if** (product == **null**) **continue**;
105. **if** (number == 0) **continue**;
107. // 如果库存不够 则报错
108. **if** (product.getProductLeftTotals() < number) {
109. sqlSession.close();
110. // TODO: throw exception OR return error message?
111. **throw** **new** CartException(NO\_MORE\_PRODUCT\_MESSAGE, NO\_MORE\_PRODUCT);
112. }
114. // 检查商品是否存咋
115. CartExample cartExample = **new** CartExample();
116. Criteria criteria = cartExample.createCriteria();
117. criteria.andUserIdEqualTo(userID).andProductIdEqualTo(productID);
118. List<Cart> productInCart = cartMapper.selectByExample(cartExample);
120. // 如果商品存在购物车中
121. **if** (productInCart != **null** && !productInCart.isEmpty()) {
122. // 如果购物车中相同的商品出现一次以上，则报错
123. **if** (productInCart.size() != 1) {
124. sqlSession.close();
125. **throw** **new** CartException(DUPLICATE\_SAME\_PRODUCT\_MESSAGE, DUPLICATE\_SAME\_PRODUCT);
126. }
127. // 更新商品
128. Cart cartToUpdate = productInCart.get(0);
129. Integer originNumber = cartToUpdate.getTotals();
130. cartToUpdate.setTotals(originNumber + number);
131. cartMapper.updateByExample(cartToUpdate, cartExample);
132. }
133. // 如果商品不存在购物车
134. **else** {
135. // 购物车中插入商品
136. Cart cartToInsert = **new** Cart();
137. cartToInsert.setTotals(number);
138. cartToInsert.setUserId(userID);
139. cartToInsert.setProductId(productID);
140. cartMapper.insertSelective(cartToInsert);
141. }
142. }
144. sqlSession.close();
145. }
147. /\*\*
148. \* 更新购物车商品
149. \*
150. \* @param user          用户
151. \* @param productIDList 商品id
152. \* @param numberList    对应的购买数量
153. \*/
154. @Transactional
155. **public** **void** updateProducts(User user, List<String> productIDList, List<Integer> numberList) {
157. checkParams(user, productIDList, numberList);
159. String userID = user.getUserId();
160. SqlSession sqlSession = sqlSessionFactory.openSession();
161. ProductMapper productMapper = sqlSession.getMapper(ProductMapper.**class**);
162. CartMapper cartMapper = sqlSession.getMapper(CartMapper.**class**);
164. // 更新数据库
165. **for** (**int** i = 0; i < productIDList.size(); i++) {
166. String productID = productIDList.get(i);
167. Integer number = numberList.get(i);
169. // 如果商品为空 则忽略
170. **if** (productID == **null** || number == **null** || productID.equals("")) {
171. **continue**;
172. }
173. Product product = productMapper.selectByPrimaryKey(productID);
174. **if** (product == **null**) **continue**;
176. // 检查库存是够满足
177. **if** (product.getProductLeftTotals() < number) {
178. sqlSession.close();
179. **throw** **new** CartException(NO\_MORE\_PRODUCT\_MESSAGE, NO\_MORE\_PRODUCT);
180. }
182. // 检查购买数量是否合理
183. **if** (number <= 0) {
184. sqlSession.close();
185. **throw** **new** CartException(WRONG\_PRODUCT\_NUMBER\_MESSAGE, WRONG\_PRODUCT\_NUMBER);
186. }
188. // 更新数据
189. Cart cart = **new** Cart();
190. cart.setProductId(productID);
191. cart.setUserId(user.getUserId());
192. cart.setTotals(number);
193. CartExample cartExample = **new** CartExample();
194. Criteria criteria = cartExample.createCriteria();
195. criteria.andProductIdEqualTo(productID).andUserIdEqualTo(userID);
196. cartMapper.updateByExample(cart, cartExample);
197. }
199. sqlSession.close();
200. }
202. /\*\*
203. \* 删除购物车信息
204. \*
205. \* @param user          用户
206. \* @param productIDList 商品id列表
207. \*/
208. @Transactional
209. **public** **void** deleteProducts(User user, List<String> productIDList) {
210. // 检查合法性
211. checkParams(user, productIDList);
213. String userID = user.getUserId();
214. SqlSession sqlSession = sqlSessionFactory.openSession();
215. CartMapper cartMapper = sqlSession.getMapper(CartMapper.**class**);
217. **for** (String productID : productIDList) {
218. CartExample cartExample = **new** CartExample();
219. Criteria criteria = cartExample.createCriteria();
220. criteria.andUserIdEqualTo(userID).andProductIdEqualTo(productID);
221. cartMapper.deleteByExample(cartExample);
222. }
224. sqlSession.close();
225. }
227. /\*\*
228. \* 获取用户购物车信息
229. \*
230. \* @param user 用户
231. \* @return 购物车中所有的商品
232. \*/
233. @Transactional
234. **public** List<Product> getAllProducts(User user) {
235. checkParams(user);
237. // 查询购物车中所有商品的id信息
238. String userID = user.getUserId();
239. CartExample cartExample = **new** CartExample();
240. Criteria criteria = cartExample.createCriteria();
241. criteria.andUserIdEqualTo(userID);
243. SqlSession sqlSession = sqlSessionFactory.openSession();
244. CartMapper cartMapper = sqlSession.getMapper(CartMapper.**class**);
245. ProductMapper productMapper = sqlSession.getMapper(ProductMapper.**class**);
246. List<Cart> cartList = cartMapper.selectByExample(cartExample);
248. List<Product> productList = **new** ArrayList<>();
249. // 提取商品信息
250. **for** (Cart cart : cartList) {
251. String productID = cart.getProductId();
252. Product product = productMapper.selectByPrimaryKey(productID);
254. Integer productLeft = product.getProductLeftTotals();
255. Integer productInCart = cart.getTotals();
256. **if** (productLeft < productInCart) {
257. cart.setTotals(productLeft);
258. CartExample updateExample = **new** CartExample();
259. Criteria updateCriteria = updateExample.createCriteria();
260. updateCriteria.andUserIdEqualTo(userID).andProductIdEqualTo(productID);
262. cartMapper.updateByExample(cart, updateExample);
263. }
265. product.setProductLeftTotals(cart.getTotals());
266. productList.add(product);
267. }
269. sqlSession.close();
270. **return** productList;
271. }
273. }

（4）**路由权限处理**

1. @Configuration
2. @EnableWebSecurity
3. **public** **class** WebSecurityConfig **extends** WebSecurityConfigurerAdapter {
5. /\*\*
6. \* 自己编写的用户信息的服务类
7. \*/
8. @Autowired
9. **private** CustomUserDetailsService customUserDetailsService;
11. /\*\*
12. \* 登陆成功处理
13. \*/
14. @Autowired
15. **private** LoginSuccessHandler loginSuccessHandler;
17. /\*\*
18. \* 配置
19. \* @param http
20. \* @throws Exception
21. \*/
22. @Override
23. **protected** **void** configure(HttpSecurity http) **throws** Exception {
25. http.headers().frameOptions().sameOrigin();
27. // FIXME
28. http.csrf().disable();
30. http.authorizeRequests().antMatchers("/registerpage",
31. "/register",
32. "/loginpage",
33. "/js/\*\*",
34. "/fonts/\*\*",
35. "/css/\*\*",
36. "/images/\*\*",
37. "/webjars/\*\*",
38. "/fail").permitAll()//无需登录认证权限访问
39. .anyRequest().authenticated() //其他所有资源需要认证，登陆后访问
40. .antMatchers("/userInfoPage","/changeUserInfo","/getUserInfo","/searchProductByName","/search-results.html","/index",
41. "/serverMessage","/sendMessage","/addUserAddress","/addProductToCart","/getUserAddress").hasAnyRole()
42. .antMatchers("/admin").hasRole("ADMIN") //登陆之后拥有"ADMIN"权限才能够访问，否则会出现“403”权限不足的提示
43. .and()
44. .formLogin() //表单登陆
45. .loginPage("/loginpage")
46. .loginProcessingUrl("/login") //指定登陆页面为"/login"
47. .successHandler(loginSuccessHandler) //
48. .failureUrl("/fail")
49. .defaultSuccessUrl("/index")//登录成功跳转页面
50. .and()
51. .rememberMe() //登陆后记住用户，下次自动登录，数据库中必须存在名为persistent\_logins的表
52. .tokenValiditySeconds(1209600);
53. }
55. @Autowired
56. **public** **void** configureGlobal(AuthenticationManagerBuilder auth) **throws** Exception {
57. //指定密码加密所使用的加密器为passwordEncoder()
58. //将密码加密后写入数据库
59. auth.userDetailsService(customUserDetailsService).passwordEncoder(**new** BCryptPasswordEncoder(10));
60. //不删除凭据，以便记住用户
61. auth.eraseCredentials(**false**);
62. }
63. }

**（5）文件操作**

1. @Service
2. **public** **class** FileService {
4. @Value("${image.user-image-folder-path}")
5. **private** String userImageFolderPath;
7. @Value("${image.user-image-relate-path}")
8. **private** String userImageRelatePath;
10. @Value("${image.product-image-folder-path}")
11. **private** String productImageFolderPath;
13. @Value("${image.product-image-relate-path}")
14. **private** String productImageRelatePath;
16. @Autowired
17. **private** SqlSessionFactory sqlSessionFactory;
19. /\*\*
20. \* 上传用户头像并刷新数据库
21. \*
22. \* @param userID 用户id
23. \* @param file   文件
24. \*/
25. **public** **void** uploadUserImage(String userID, MultipartFile file) {
27. // 使用当点用户的id作为图片名
28. String fileName = userID;
30. // 文件重命名
31. String fileAbsolutePath = userImageFolderPath + "/" + fileName;
32. System.out.println(fileAbsolutePath);
34. // 文件写入磁盘
35. uploadFile(fileAbsolutePath, file);
37. // 更新数据库
38. SqlSession sqlSession = sqlSessionFactory.openSession();
39. UserMapper userMapper = sqlSession.getMapper(UserMapper.**class**);
40. User user = userMapper.selectByPrimaryKey(userID);
41. user.setHeadImg(userID);
42. userMapper.updateByPrimaryKey(user);
43. sqlSession.close();
44. }
46. /\*\*
47. \* 将文件保存到本地
48. \*
49. \* @param productID 商品id
50. \* @param files     文件
51. \*/
52. **public** **void** insertProductImages(String productID, List<MultipartFile> files) {
54. //  检查文件是否为空
55. **if** (files == **null** || files.isEmpty()) {
56. **throw** **new** FileException(EMPTY\_FILE\_MESSAGE, EMPTY\_FILE);
57. }
59. SqlSession sqlSession = sqlSessionFactory.openSession();
60. ProductMapper productMapper = sqlSession.getMapper(ProductMapper.**class**);
61. ImageMapper imageMapper = sqlSession.getMapper(ImageMapper.**class**);
63. **for** (MultipartFile file : files) {
64. // 文件重命名
65. String fileName = UUID.randomUUID().toString();
66. String fileAbsolutePath = productImageFolderPath + "/" + fileName;
67. uploadFile(fileAbsolutePath, file);
69. // 保存到数据库
70. Image image = **new** Image();
71. image.setImageId(fileName);
72. image.setImageUrl(fileName);
73. image.setProductId(productID);
74. imageMapper.insert(image);
75. }
77. sqlSession.close();
78. }
80. /\*\*
81. \* 重新替换图片
82. \*
83. \* @param imageIDs 原图片id
84. \* @param files    文件
85. \*/
86. **public** **void** updateProductImages(List<String> imageIDs, List<MultipartFile> files) {
88. //  验证数据是否合理
89. **if** (imageIDs == **null** || imageIDs.isEmpty()) {
90. **throw** **new** FileException(IMAGE\_ID\_EMPTY\_MESSAGE, IMAGE\_ID\_EMPTY);
91. }
92. **if** (files == **null** || files.isEmpty()) {
93. **throw** **new** FileException(EMPTY\_FILE\_MESSAGE, EMPTY\_FILE);
94. }
95. **if** (imageIDs.size() != files.size()) {
96. **throw** **new** FileException(IMAGE\_ID\_FILE\_NOT\_MATCH\_MESSAGE, IMAGE\_ID\_FILE\_NOT\_MATCH);
97. }
99. SqlSession sqlSession = sqlSessionFactory.openSession();
100. ImageMapper imageMapper = sqlSession.getMapper(ImageMapper.**class**);
102. **for** (**int** i = 0; i < imageIDs.size(); i++) {
103. Image image = imageMapper.selectByPrimaryKey(imageIDs.get(i));
104. MultipartFile file = files.get(i);
106. // 如果图片不存在则忽略
107. **if** (image == **null**) **continue**;
109. // 如果文件为空或不存在则忽略
110. **if** (file == **null** || file.isEmpty()) **continue**;
112. // 写入本地磁盘
113. String fileName = image.getImageUrl();
114. String fileAbsolutePath = productImageFolderPath + "/" + fileName;
115. uploadFile(fileAbsolutePath, file);
116. }
118. sqlSession.close();
119. }
121. /\*\*
122. \* 删除商品图片
123. \*
124. \* @param image 图片
125. \*/
126. **public** **void** deleteProductImage(Image image) {
127. String imageID = image.getImageId();
128. String imageURL = image.getImageUrl();
130. // 验证图片合法性
131. **if** (imageID == **null** || imageID.equals("")) {
132. **throw** **new** FileException(IMAGE\_ID\_EMPTY\_MESSAGE, IMAGE\_ID\_EMPTY);
133. }
134. **if** (imageURL == **null** || imageURL.equals("")) {
135. **throw** **new** FileException(IMAGE\_URL\_EMPTY\_MESSAGE, IMAGE\_URL\_EMPTY);
136. }
138. // 删除文件
139. String fileAbsolutePath = productImageFolderPath + "/" + imageURL;
140. deleteFile(fileAbsolutePath);
142. // 更新数据库
143. SqlSession sqlSession = sqlSessionFactory.openSession();
144. ImageMapper imageMapper = sqlSession.getMapper(ImageMapper.**class**);
145. imageMapper.deleteByPrimaryKey(imageID);
146. sqlSession.close();
147. }
149. /\*\*
150. \* 根据图片id自动删除图片
151. \*
152. \* @param imageID 图片id
153. \*/
154. **public** **void** deleteProductImage(String imageID) {
155. // validate data
156. **if** (imageID == **null** || imageID.equals("")) {
157. **throw** **new** FileException(IMAGE\_ID\_EMPTY\_MESSAGE, IMAGE\_ID\_EMPTY);
158. }
160. // 从数据库中查询image
161. SqlSession sqlSession = sqlSessionFactory.openSession();
162. ImageMapper imageMapper = sqlSession.getMapper(ImageMapper.**class**);
163. Image image = imageMapper.selectByPrimaryKey(imageID);
164. sqlSession.close();
166. // 删除文件并且自动更新数据库
167. deleteProductImage(image);
168. }
170. /\*\*
171. \* 删除图片
172. \*
173. \* @param imageList 图片列表
174. \*/
175. **public** **void** deleteProductImages(List<Image> imageList) {
176. // 验证数据
177. **if** (imageList == **null**) {
178. **return**;
179. }
181. **for** (Image image : imageList) {
182. deleteProductImage(image);
183. }
184. }
186. /\*\*
187. \* 删除所有产品的图片
188. \*
189. \* @param productID 产品编号
190. \*/
191. **public** **void** deleteAllProductImages(String productID) {
192. // 验证数据
193. **if** (productID == **null** || productID.equals("")) {
194. **throw** **new** FileException(QUERY\_PID\_ERROR\_MESSAGE, QUERY\_PID\_ERROR);
195. }
197. // 创建查询实例
198. ImageExample imageExample = **new** ImageExample();
199. Criteria criteria = imageExample.createCriteria();
200. criteria.andProductIdEqualTo(productID);
202. // 查询所有图片
203. SqlSession sqlSession = sqlSessionFactory.openSession();
204. ImageMapper imageMapper = sqlSession.getMapper(ImageMapper.**class**);
205. List<Image> imageList = imageMapper.selectByExample(imageExample);
206. sqlSession.close();
208. **for** (Image image : imageList) {
209. deleteProductImage(image);
210. }
211. }
213. /\*\*
214. \* 返回用户头像的相关url
215. \*
216. \* @param userID 用户id
217. \* @return 用户头像的相对url地址
218. \*/
219. **public** String getUserImageURL(String userID) {
220. **if** (userID == **null** || userID.equals("")) {
221. **throw** **new** FileException(EMPTY\_USER\_MESSAGE, EMPTY\_USER);
222. }
224. SqlSession sqlSession = sqlSessionFactory.openSession();
225. UserMapper userMapper = sqlSession.getMapper(UserMapper.**class**);
226. User user = userMapper.selectByPrimaryKey(userID);
227. **if** (user == **null**) {
228. **throw** **new** FileException(EMPTY\_USER\_MESSAGE, EMPTY\_USER);
229. }
231. String userImageURL = user.getHeadImg();
232. sqlSession.close();
234. **return** userImageRelatePath + "/" + userImageURL;
235. }
237. /\*\*
238. \* 获取产品的所有图片
239. \* It <b>WILL NOT</b> return null but an empty list.
240. \*
241. \* @param productID 产品id
242. \* @return 产品的图片地址
243. \*/
244. **public** List<String> getProductImagesURL(String productID) {
245. **if** (productID == **null** || productID.equals("")) {
246. **throw** **new** FileException(QUERY\_PID\_ERROR\_MESSAGE, QUERY\_PID\_ERROR);
247. }
249. // 获取url
250. List<Image> images = getProductImages(productID);
251. List<String> pathList = **new** ArrayList<>();
252. **for** (Image image : images) {
253. pathList.add(productImageRelatePath + "/" + image.getImageUrl());
254. }
256. **return** pathList;
257. }
259. /\*\*
260. \* 通过产品id查询图片
261. \*
262. \* @param productID 产品id
263. \* @return 产品的图片
264. \*/
265. **private** List<Image> getProductImages(String productID) {
266. **if** (productID == **null** || productID.equals("")) {
267. **throw** **new** FileException(QUERY\_PID\_ERROR\_MESSAGE, QUERY\_PID\_ERROR);
268. }
270. // 数据库查询
271. ImageExample imageExample = **new** ImageExample();
272. Criteria criteria = imageExample.createCriteria();
273. criteria.andProductIdEqualTo(productID);
274. SqlSession sqlSession = sqlSessionFactory.openSession();
275. ImageMapper imageMapper = sqlSession.getMapper(ImageMapper.**class**);
276. List<Image> images = imageMapper.selectByExample(imageExample);
277. sqlSession.close();
279. **return** images;
280. }
282. /\*\*
283. \* 文件写入本地.
284. \*
285. \* @param fileAbsolutePath: 文件绝对路径
286. \* @param file:             从浏览器获取的文件
287. \*/
288. **private** **void** uploadFile(String fileAbsolutePath, MultipartFile file) {
289. **if** (fileAbsolutePath == **null** || fileAbsolutePath.equals("")) {
290. **throw** **new** FileException(IO\_EXCEPTION\_MESSAGE, IO\_EXCEPTION);
291. }
292. **if** (file.isEmpty()) {
293. **throw** **new** FileException(EMPTY\_FILE\_MESSAGE, EMPTY\_FILE);
294. }
296. // 检查文件是否存在
297. File fileToSave = **new** File(fileAbsolutePath);
298. **if** (!fileToSave.getParentFile().exists()) {
299. fileToSave.getParentFile().mkdir();
300. }
302. // 写入文件
303. **try** {
304. file.transferTo(fileToSave);
305. } **catch** (IOException e) {
306. String error = "不能写入文件: " + e.getMessage();
307. **throw** **new** FileException(IO\_EXCEPTION\_MESSAGE + error, IO\_EXCEPTION);
308. }
309. }
311. /\*\*
312. \* 删除文件.
313. \*
314. \* @param fileAbsolutePath 文件绝对路径
315. \*/
316. **private** **void** deleteFile(String fileAbsolutePath) {
317. **if** (fileAbsolutePath == **null** || fileAbsolutePath.equals("")) {
318. **throw** **new** FileException(IO\_EXCEPTION\_MESSAGE, IO\_EXCEPTION);
319. }
320. File fileToDelete = **new** File(fileAbsolutePath);
321. // 如果文件不错在或者是文件夹
322. **if** (!fileToDelete.exists() || !fileToDelete.isFile()) {
323. **return**;
324. }
325. fileToDelete.delete();
326. }
327. }

**（6）用户账户相关JavaScript**

1. $(**function** () {
2. $.getJSON("/getUserInfo", **function** (data) {
3. **var** username = data.data.userName;
4. **var** userimg = data.data.headImg;
5. **var** useremail = data.data.userId;
6. **var** userdate = data.data.birthday;
7. console.log(username);
8. // 打印用户信息
9. $('#user-info-name').attr('placeholder', username);
10. $('#user-info-email').attr('placeholder', useremail);
11. **if** (**null** != userdate) {
12. userdate = userdate.substring(0, 10);
13. $('#user-info-date').attr('placeholder', userdate);
14. }
15. $('#user-info-image').attr('src', userimg);
16. })
17. });
19. **function** check() {
20. **var** username = $('#user-info-name').val();
21. **var** userdate = $('#user-info-date').val();
22. **var** userpsw1 = $('#user-info-psw1').val();
23. **var** userpsw2 = $('#user-info-psw2').val();
24. **var** DATE\_FORMAT = /^[0-9]{4}-[0-1]?[0-9]{1}-[0-3]?[0-9]{1}$/; //日期的正则表达式
25. //对输入内容的判断
26. //至少得输入一项
27. **if** (!username && !userdate && !userpsw1 && !userpsw2) {
28. swal("", "请至少修改一项内容", "warning");
29. **return** **false**;
30. }
31. //密码得一样
32. **else** **if** (userpsw2 !== userpsw1) {
33. swal("", "两次密码输入不一致，请重试", "error");
34. **return** **false**;
35. }
36. //输入的格式不能错
37. **else** **if** (userdate && !DATE\_FORMAT.test(userdate)) {
38. swal("", "输入的日期格式有误，请按照xxxx-xx-xx的格式输入", "error");
39. **return** **false**;
40. }
42. }
44. **function** checkAddress() {
45. **var** address = $('input[name = addressInfo]').val();
46. **var** phone = $('input[name = phoneNumber]').val();
47. **if** (!address || !phone) {
48. swal("", "请将信息输入完整", "warning");
49. **return** **false**;
50. }
52. }
54. //上传图片功能
55. $('#user-info-image').click(**function** () {
56. $('#choose-image').trigger('click');
57. });
58. $('#choose-image').change(**function** () {
59. $('#submit-image').submit();
60. });
62. //order
63. $(**function** () {
64. **var** status = "";
65. **var** order\_number = 1;
66. $.getJSON("/getUserOrders", **function** (data) {
67. **for** (**var** i = 0; i < data.length; i++) {
68. status = data[i].orderStatus;
69. **var** orderID = data[i].orderId;
70. $.post("/getOrderItems", {
71. orderID: data[i].orderId
72. }, **function** (DATA) {
73. **for** (**var** j = 0; j < DATA.length; j++) {
74. **var** tr = "<tr>";
75. **if** (0 == j) {
76. tr += "<td rowspan='" + DATA.length + "'>" + order\_number + "</td>";
77. order\_number++;
78. }
79. tr += "<td>";
80. tr += "<h4><a href='productdetails?productID=" + DATA[j].productId + "'>" + DATA[j].productName + "</a></h4>";
81. tr += "<span>" + DATA[j].productDescription + "</span>";
82. tr += "</td>";
83. tr += "<td>" + DATA[j].productPrice + "</td>";
84. tr += "<td>" + DATA[j].productLeftTotals + "</td>";
85. tr += "<td>" + DATA[j].productPrice \* DATA[j].productLeftTotals + "</td>";
86. **if** (0 === j) {
87. **if** (status === "已付款") {
88. tr += "<td rowspan='" + DATA.length + "'>" + "<button id='"+  orderID +  "' onclick='pay(this)' style=\"margin-right:20px;\"  class=\"btn mini btn-success\"><i class=\"fa fa-edit\"></i>" + status + "</button></td>";
89. }
90. **else** {
92. tr += "<td rowspan='" + DATA.length + "'>" + "<button id='"+  orderID +  "' onclick='pay(this)' style=\"margin-right:20px;\"  class=\"btn mini btn-warning\"><i class=\"fa fa-edit\"></i>" + status + "</button></td>";
93. }
94. }
95. **if** (0 === j) {
96. tr += "<td rowspan='" + DATA.length + "'>" + "<a class=\'glyphicon glyphicon-trash\'href=\'\'></a></td>";
97. }
99. tr += "</tr>";
100. $('#order-info').append(tr);
101. }
102. })
103. }
105. })
106. });
108. **function** pay(obj) {
109. **var** btn = $(obj);
110. **var** id = btn.attr("id");
111. swal({
112. title: "确认付款",
113. text: "请确认您的付款",
114. type: "info",
115. showCancelButton: **true**,
116. closeOnConfirm: **false**,
117. showLoaderOnConfirm: **true**,
118. }).then(**function** (value) {
119. $.post(
120. "payOrderByOrderId",
121. {orderId: id},
122. **function** (result) {
123. **if**(result == **null** || result === "") {
124. //TODO
125. btn.text("已付款");
126. btn.removeClass("btn-warning");
127. btn.addClass("btn-success");
128. btn.prop("disabled", **true**);
129. swal("", "付款成功", "success");
130. }
131. **else** {
132. swal("", result, "error");
133. }
134. }
135. );
136. });
137. }



142. //address
143. $(**function** () {
144. $.getJSON("/getUserAddress",**function** (data) {
145. console.log(data);
146. **for**(**var** i=0;i<data.length;i++){
147. **var** tr = "<tr>";
148. tr += "<td>" + data[i].userId + "</td>";
149. tr += "<td>" + data[i].addressInfo + "</td>";
150. tr += "<td>" + data[i].phoneNumber + "</td>";
151. tr += "<td>" + "<a href=\"javascript:void(0);\">修改</a>|<a href=\"javascript:void(0);\">删除</a>" + "</td>";
152. **if**(data[i].isDefaultAddress === **false**) {
153. tr += "<td>" + "</td>";
154. }
155. **else** {
156. tr += "<td>" + "默认地址" +"</td>";
157. }
158. tr += "</tr>";
159. $('#address-info').append(tr);
160. }
161. })
163. });
165. //message
167. $(**function** () {
168. $.getJSON("/getUserMessage", **function** (data) {
169. **var** unread = 0;
170. **for** (**var** i = 0; i < data.length; i++) {
171. **var** div = "<div class=\"notif\">";
172. **if**(data[i].isRead === **false**){
173. div += "<div class=\"notif-status col-sm-1 col-md-1\">未读</div>";
174. unread ++;
175. }
176. **else**{
177. div += "<div class=\"notif-status col-sm-1 col-md-1\">已读</div>";
178. }
179. div += "<div class=\"col-sm-11 col-md-11\">\n" +
180. "<div class=\"notif-head\">";
181. div += "<span>" + data[i].senderId + "</span>";
182. **if**(data[i].sendTime != **null**){
183. div += "<span>" + data[i].sendTime + "</span>";
184. }
185. div += "</div>";
186. div += "<div class=\"notif-body fa-border\">";
187. div += "<p>" + data[i].messageContent + "</p>";
188. div += "</div>";
189. div += "</div>";
190. div += "</div>";
191. $('#user-notification').append(div);
192. }
193. $('.badge').text(unread);
194. })
195. });
197. //js实现标签栏跳转至对应的窗口
198. $(**function** () {
199. **var** url = decodeURI(window.location.href);
200. **var** argsIndex = url.split("?kw=");
201. **var** arg = argsIndex[1];
202. console.log(arg);
203. // $('#shop-menu > li').click(function () {
204. //     $('#shop-menu2 > li .active').removeClass('active');
205. //     console.log($('li .active'));
206. $('#shop-menu2').find('> li').each(**function** () {
207. **if** ($(**this**).hasClass('active')) {
208. $(**this**).removeClass('active');
209. }
210. })
211. $('.tab-content > div').each(**function** () {
212. **if** ($(**this**).hasClass('active')) {
213. $(**this**).removeClass('active');
214. }
215. })
216. **var** id = "#" + arg;
217. $('#shop-menu2').find('> li > a').each(**function** () {
218. **if** ($(**this**).attr('href') === id) {
219. $(**this**).parent().addClass("active");
220. $(id).addClass("active");
221. }
222. });
223. **if**(!arg){
224. $('#shop-menu2').find('> li > a').each(**function** () {
225. **if** ($(**this**).attr('href') === "#user-info") {
226. $(**this**).parent().addClass("active");
227. $("#user-info").addClass("active");
228. }
229. })
230. }
231. });

（7）**购物车相关操作JavaScript**

1. // TODO: 删除商品后刷新
2. // TODO: 自动删除商品后刷新
3. // TODO: 商品数量区间提示
4. // TODO: 输入框

7. window.onload = **function** (ev) {
8. $(".product-tr-template").hide(); // hide template
9. $.get(
10. "/productInCart",
11. **function** (result) {
12. showProducts(result);
13. }
14. )
15. }
17. **function** showProducts(data) {
18. **var** productCount = data.length;
19. **if** (productCount == 0) {
20. swal("", "购物车为空", "warning");
21. **return**;
22. }
23. **for** (**var** i = 0; i < productCount; i++) {
24. **var** name
25. **var** productRow = $(".product-tr-template").clone(**true**).prependTo("#product-tbody");
26. productRow.children(".cart\_product").find("img").attr("alt", data[i].productId);
27. // TODO: 跳转至详情页 productRow.children(".cart\_product").find("a").attr("href", "详情页?productID=data[i].productId");
28. productRow.children(".cart\_description").find("a").text(data[i].productName);
29. // TODO: 跳转至详情页 productRow.children(".cart\_description").find("a").attr("href", "详情页?productID=data[i].productId");
30. productRow.children(".cart\_description").find("p").text(data[i].productDescription);
31. productRow.children(".cart\_price").find("p").text(data[i].productPrice);
32. productRow.children(".cart\_quantity").find("input").attr("value", data[i].productLeftTotals);
33. productRow.children(".cart\_total").find("p").text(data[i].productPrice \* data[i].productLeftTotals);
34. productRow.find(".cart\_quantity\_delete").attr("href", "/deleteProduct?productID="+data[i].productId);
35. productRow.attr("class", "product-tr"); // change id to divide from template
36. productRow.show();
37. showImage(productRow, data[i].productId);
38. }
39. $(".product-tr-template").remove();
40. }
42. **function** showImage(productRow, productID) {
43. $.get(
44. "/productImageInCart",
45. {productID: productID},
46. **function** (result) {
47. productRow.children(".cart\_product").find("img").attr("src", result);
48. }
49. );
50. }
52. // listen to item row
53. $(".check").change(**function** () {
54. reComputeAll();
55. });
57. **function** reComputeAll() {
58. **var** totalCount = 0;
59. **var** totalPrice = 0;
60. $(".product-tr").each(**function** () {
61. **var** isChecked = $(**this**).find(".check").prop("checked");
62. **if** (isChecked) {
63. **var** itemCount = parseInt($(**this**).find(".cart\_quantity\_input").val());
64. **var** itemTotalPrice = parseFloat($(**this**).find(".cart\_total\_price").text());
65. totalCount += itemCount;
66. totalPrice += itemTotalPrice;
67. }
68. **else** {
69. // unselect "SelectAll" box
70. $("#cart\_purchase\_bar").find("input").prop("checked", **false**);
71. }
72. });
73. $(".cart\_quantity").children("em").text(totalCount);
74. $("#total\_price").html(totalPrice);
75. }
77. // listen to select all
78. $("#cart\_purchase\_bar").find("input").change(**function** () {
79. **var** isChecked = $(**this**).prop("checked");
80. $(".product-tr").each(**function** () {
81. $(**this**).find(".check").prop("checked", isChecked);
82. });
83. reComputeAll();
84. });
86. // listen to increase product
87. $(".cart\_quantity\_up").click(**function** () {
88. **var** thisObj = $(**this**);
89. **var** rowObj = $(**this**).parents("tr");
90. $.post(
91. "/modifyProductNumber",
92. {
93. productID: $(**this**).parents(".product-tr").find("img").attr("alt"),
94. productNumber: parseInt($(**this**).parents(".product-tr").find(".cart\_quantity\_input").val()) + 1,
95. },
96. **function** (result) {
97. **if** (result == **null** || result == "") {
98. **var** itemCount = parseInt(thisObj.siblings("input").val());
99. thisObj.siblings(".cart\_quantity\_input").attr("value", itemCount + 1);
101. // update row
102. **var** producePrice = parseFloat(rowObj.children(".cart\_price").find("p").text());
103. **var** productCount = parseInt(rowObj.children(".cart\_quantity").find("input").val());
104. rowObj.children(".cart\_total").find("p").text(producePrice \* productCount);
106. reComputeAll();
107. }
108. **else** {
109. swal("",result, "error");
110. }
111. }
112. );
113. });
115. // listen to decrease product
116. $(".cart\_quantity\_down").click(**function** () {
117. **var** thisObj = $(**this**);
118. **var** rowObj = $(**this**).parents("tr");
119. $.post(
120. "/modifyProductNumber",
121. {
122. productID: $(**this**).parents(".product-tr").find("img").attr("alt"),
123. productNumber: parseInt($(**this**).parents(".product-tr").find(".cart\_quantity\_input").val()) - 1,
124. },
125. **function** (result) {
126. **if** (result == **null** || result == "") {
127. **var** itemCount = parseInt(thisObj.siblings("input").val());
128. thisObj.siblings(".cart\_quantity\_input").attr("value", itemCount - 1);
130. // update row
131. **var** producePrice = parseFloat(rowObj.children(".cart\_price").find("p").text());
132. **var** productCount = parseInt(rowObj.children(".cart\_quantity").find("input").val());
133. rowObj.children(".cart\_total").find("p").text(producePrice \* productCount);
135. reComputeAll();
136. }
137. **else** {
138. swal("",result, "error");
139. }
140. }
141. );
142. });
144. // listen to input label
145. // remember old value
146. $(".cart\_quantity\_input").focus(**function** () {
147. $(**this**).attr("oldValue", $(**this**).val());
148. });
149. // check new value
150. $(".cart\_quantity\_input").blur(**function** () {
151. **var** thisObj = $(**this**);
152. **var** rowObj = $(**this**).parents("tr");
153. $.post(
154. "/modifyProductNumber",
155. {
156. productID: $(**this**).parents(".product-tr").find("img").attr("alt"),
157. productNumber: parseInt($(**this**).parents(".product-tr").find(".cart\_quantity\_input").val()),
158. },
159. **function** (result) {
160. **if** (result == **null** || result == "") {
161. **var** itemCount = parseInt(thisObj.siblings("input").val());
162. thisObj.siblings(".cart\_quantity\_input").attr("value", itemCount);
164. // update row
165. **var** producePrice = parseFloat(rowObj.children(".cart\_price").find("p").text());
166. **var** productCount = parseInt(rowObj.children(".cart\_quantity").find("input").val());
167. rowObj.children(".cart\_total").find("p").text(producePrice \* productCount);
169. reComputeAll();
170. }
171. **else** {
172. swal("",result, "error");
173. thisObj.val(thisObj.attr("oldValue"));
174. }
175. }
176. );
177. });
179. // modify submit data
180. $('#settle-account').submit(**function** (e) {
181. **var** form = $(**this**);
182. $(".product-tr").each(**function** () {
183. **var** isChecked = $(**this**).find(".check").prop("checked");
184. **if** (isChecked) {
185. **var** productID = $(**this**).find("img").attr("alt");
186. $("<input />").attr("type", "hidden").attr("name", "productID").attr("value", productID).appendTo(form);
187. }
188. **else** {
189. $(**this**).find(".cart\_quantity\_input").attr("disabled", "disabled");
190. }
191. });
192. **return** **true**;
193. });