# **购物车分析**

未登录状态操作购物车

登录状态操作购物车

合并购物车

## **购物车功能**

1.添加购物车商品

2.展示购物车列表页面

3.修改购物车商品数量

4.删除购物车商品

## **购物车设计**

1、用户未登录状态下：在不登陆的情况下把购物车信息写入cookie

​ 优点：

​ 1、不占用服务端存储空间

​ 2、代码实现简单。

​ 3、用户体验好

​ 缺点：

​ 1、cookie中保存的容量有限。最大4k

​ 2、把购物车信息保存在cookie中，更换设备购物车信息不能同步。

2、用户已登录状态下：把购物车信息保存到服务端的 Redis 中

​ 优点： 1、更换设备购物车信息可以同步

​ 缺点：1、占用服务端存储空间

### **创建usian\_cart\_service**

**Pom.xml**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

<dependency>

<groupId>com.usian</groupId>

<artifactId>common\_redis</artifactId>

<version>1.0-SNAPSHOT</version>

</dependency>

<dependency>

<groupId>com.usian</groupId>

<artifactId>common\_pojo</artifactId>

<version>1.0-SNAPSHOT</version>

</dependency>

**Application.yml**

server:

port: 8100

eureka:

client:

service-url:

defaultZone: http://127.0.0.1:8761/eureka/

spring:

application:

name: usian-cart-service

datasource:

driver-class-name: com.mysql.jdbc.Driver

url: jdbc:mysql://127.0.0.1:3306/usian?characterEncoding=UTF-8

username: root

password: 1111

type: com.alibaba.druid.pool.DruidDataSource

redis:

cluster:

nodes:

- 192.168.233.131:7001

- 192.168.233.131:7002

- 192.168.233.131:7003

- 192.168.233.131:7004

- 192.168.233.131:7005

- 192.168.233.131:7006

jedis:

pool:

max-active: 20 #连接池最大连接数

max-idle: 10 #连接池中的最大空闲连接

min-idle: 5 # 连接池中的最小空闲连接

#### **logback.xml**

#### **启动类**

@SpringBootApplication

@EnableDiscoveryClient

public class CartServiceApp {

public static void main(String[] args) {

SpringApplication.run(CartServiceApp.class, args);

}

}

### **创建usian\_cart\_feign**

#### **pom.xml**

<!--Spring Cloud OpenFeign Starter -->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-openfeign</artifactId>

</dependency>

<dependency>

<groupId>com.usian</groupId>

<artifactId>common\_pojo</artifactId>

<version>1.0-SNAPSHOT</version>

</dependency>

### **创建usian\_cart\_web**

#### **pom.xml**

<dependency>

<groupId>com.usian</groupId>

<artifactId>usian\_cart\_feign</artifactId>

<version>1.0-SNAPSHOT</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

<dependency>

<groupId>com.usian</groupId>

<artifactId>common\_utils</artifactId>

<version>1.0-SNAPSHOT</version>

</dependency>

<dependency>

<groupId>com.usian</groupId>

<artifactId>usian\_item\_feign</artifactId>

<version>1.0-SNAPSHOT</version>

</dependency>

#### **application.yml**

spring:

application:

name: usian-cart-web

server:

port: 8101

eureka:

client:

service-url:

defaultZone: <http://127.0.0.1:8761/eureka/>

#### **logback.xml**

#### **启动类**

@SpringBootApplication

@EnableDiscoveryClient

@EnableFeignClients

public class CartWebApp {

public static void main(String[] args) {

SpringApplication.run(CartWebApp.class, args);

}

}

## **添加购物车**

1、从cookie中查询商品列表：Map<itemId,TbItem> 商品购买数量使用TbItem的num保存

​ 购物车已存在则直接返回

​ 购物车不存在则创建空的购物车并返回

2、添加商品到购物车：

​ 如果购物车存在该商品，商品数量相加。

​ 如果购物车不存在该商品，根据商品id查询商品信息并添加到购车列表

3、把购车商品列表写入cookie。

​ 读写cookie可以使用CookieUtils工具类实现

## **查看购物车**

### **usian\_cart\_web**

#### **controller**

/\*\*

\* 查看购物车

\*/

@RequestMapping("/showCart")

public Result showCart(String userId,HttpServletRequest

request,HttpServletResponse response){

try{

if(StringUtils.isBlank(userId)){

//在用户未登录的状态下

List<TbItem> list = new ArrayList<TbItem>();

Map<String,TbItem> cart = this.getCartFromCookie(request);

Set<String> keys = cart.keySet();

for(String key :keys){

list.add(cart.get(key));

}

}else{

// 在用户已登录的状态

}

return Result.ok(list);

}catch(Exception e){

e.printStackTrace();

}

return Result.error("error");

}

## **修改购物车**

/\*\*

\* 修改购物车

\*/

@RequestMapping("/updateItemNum")

public Result updateItemNum(String userId,Long itemId,Integer num,

HttpServletRequest request,HttpServletResponse response){

try {

if(StringUtils.isBlank(userId)){

//未登录

//1、获得cookie中的购物车

Map<String, TbItem> cart = getCartFromCookie(request);

//2、修改购物车中的商品

TbItem tbItem = cart.get(itemId.toString());

tbItem.setNum(num);

cart.put(itemId.toString(),tbItem);

//3、把购物车写到cookie

addClientCookie(cart,request,response);

}else{

//登录

}

return Result.ok();

}catch (Exception e){

e.printStackTrace();

return Result.error("修改失败");

}

}

## **删除购物车**

/\*\*

\* 删除购物车中的商品

\*/

@RequestMapping("/deleteItemFromCart")

public Result deleteItemFromCart(Long itemId, String userId, HttpServletRequest

request, HttpServletResponse response) {

try {

if (StringUtils.isBlank(userId)) {

//在用户未登录的状态下

Map<String,TbItem> cart = this.getCartFromCookie(request);

cart.remove(itemId.toString());

this.addClientCookie(request,response,cart);

} else {

// 在用户已登录的状态

}

return Result.ok();

} catch (Exception e) {

e.printStackTrace();

}

return Result.error("删除失败");

}

## 在用户登录情况下

### 使用redis存储商品列表

使用hash对购物车进行归类

​ HASH :HASH\_KEY

​ |--KEY:VALUE

​ |--KEY:VALUE

​ HASH : CART\_REDIS\_KEY

​ |--22:[{...}]

​ |--23:[{...}]

## **添加购物车**

### **usian\_cart\_web**

#### **controller**

/\*\*

\* 购物车 Controller

\*/

@RestController

@RequestMapping("/frontend/cart")

public class CartController {

//... ... ...

@Autowired

private CartServiceFeign cartServiceFeign;

@RequestMapping("/addItem")

public Result addItem(Long itemId, String userId,@RequestParam(defaultValue = "1")

Integernnum,HttpServletRequest request, HttpServletResponse response) {

if (StringUtils.isBlank(userId)) {

/\*\*\*\*\*\*\*\*\*\*\*在用户未登录的状态下\*\*\*\*\*\*\*\*\*\*/

..........

} else {

/\*\*\*\*\*\*\*\*\*\*\*在用户已登录的状态\*\*\*\*\*\*\*\*\*\*/

// 1、从redis中查询商品列表。

Map<String,TbItem> cart = getCartFromRedis(userId);

//2、将商品添加大购物车中

this.addItemToCart(cart,itemId,num);

//3、将购物车缓存到 redis 中

Boolean addCartToRedis = this.addCartToRedis(userId, cart);

if(addCartToRedis){

return Result.ok();

}

return Result.error("error");

}

}

/\*\*

\* 把购车商品列表写入redis

\* @param userId

\* @param cart

\*/

private Boolean addCartToRedis(String userId, Map<String, TbItem> cart) {

return cartServiceFeign.insertCart(userId, cart);

}

/\*\*

\* 从redis中查询购物车

\* @param userId

\*/

private Map<String,TbItem> getCartFromRedis(String userId) {

Map<String,TbItem> cart = cartServiceFeign.selectCartByUserId(userId);

if(cart!=null && cart.size()>0){

return cart;

}

return new HashMap<String,TbItem>();

}

//... ... ...

}

### **usian\_cart\_feign**

@FeignClient("usian-cart-service")

public interface CartServiceFeign {

@RequestMapping("/service/cart/selectCartByUserId")

Map<String, TbItem> selectCartByUserId(@RequestParam String userId);

@RequestMapping("/service/cart/insertCart?userId=234234")

Boolean insertCart(@RequestParam String userId, Map<String, TbItem> cart);

}

### **usian\_cart\_service**

#### **application.yml**

#购物车

CART\_REDIS\_KEY: CART\_REDIS\_KEY

#### **Service**

/\*\*

\* 购物车操作业务层

\*/

@Service

public class CartServiceImpl implements CartService {

@Autowired

private RedisClient redisClient;

@Value("${CART\_REDIS\_NAME}")

private String CART\_REDIS\_NAME;

/\*\*

\* 根据用户 ID 查询用户购物车

\*/

@Override

public Map<String, TbItem> selectCartByUserId(String userId) {

return (Map<String, TbItem>) redisClient.hget(CART\_REDIS\_NAME,userId);

}

/\*\*

\* 缓存购物车

\*

\* @param cart

\*/

@Override

public Boolean insertCart(String userId, Map<String, TbItem> cart) {

return redisClient.hset(CART\_REDIS\_NAME, userId, cart);

}

}

#### **controller**

/\*\*

\* 购物车操作

\*/

@RestController

@RequestMapping("/service/cart")

public class CartController {

@Autowired

private CartService cartService;

/\*\*

\* 根据用户 ID 查询用户购物车

\*/

@RequestMapping("/selectCartByUserId")

public Map<String, TbItem> selectCartByUserId(@RequestParam String userId){

return this.cartService.selectCartByUserId(userId);

}

/\*\*

\* 将购物车缓存到 redis 中

\*/

@RequestMapping("/insertCart")

public Boolean insertCart(String userId, @RequestBody Map<String, TbItem> map) {

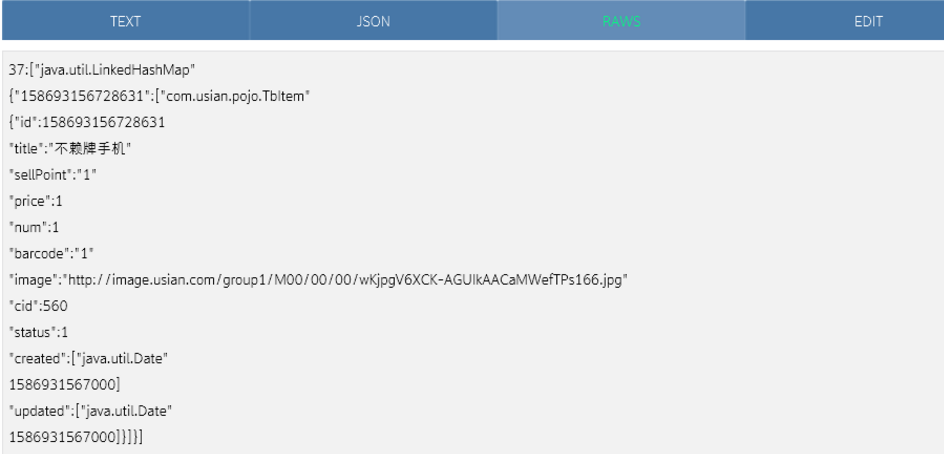
return this.cartService.insertCart(userId, map);

}

}

#### **测试**

查看redis缓存数据



## **查看购物车**

### **usian\_cart\_web**

#### **Controller**

/\*\*

\* 查看购物车

\*/

@RequestMapping("/showCart")

public Result showCart(String userId,HttpServletRequest

request,HttpServletResponse response){

try{

if(StringUtils.isBlank(userId)){

//在用户未登录的状态下

//............

}else{

// 在用户已登录的状态

List<TbItem> list = new ArrayList<TbItem>();

Map<String,TbItem> cart = this.getCart(userId);

Set<String> keys = cart.keySet();

for(String key :keys){

list.add(cart.get(key));

}

return Result.ok(list);

}

}catch(Exception e){

e.printStackTrace();

}

return Result.error("error");

}

## **修改购物车**

### **usian\_cart\_web**

#### **controller**

/\*\*

\* 修改购物车中商品的数量

\*/

@RequestMapping("/updateItemNum")

public Result updateItemNum(Long itemId, String userId, Integer

num, HttpServletRequest request, HttpServletResponse response) {

try{

if (StringUtils.isBlank(userId)) {

//在用户未登录的状态下

//.........

} else {

// 在用户已登录的状态

Map<String,TbItem> cart = this.getCart(userId);

TbItem item = cart.get(itemId.toString());

if(item != null){

item.setNum(num);

}

//将新的购物车缓存到 Redis 中

this.addCartToRedis(userId,cart);

return Result.ok();

}

}catch(Exception e){

e.printStackTrace();

}

return Result.error("修改失败");

}

## **删除购物车**

### **usian\_cart\_web**

#### **Controller**

/\*\*

\* 删除购物车中的商品

\*/

@RequestMapping("/deleteItemFromCart")

public Result deleteItemFromCart(Long itemId, String userId, HttpServletRequest

request, HttpServletResponse response) {

try {

if (StringUtils.isBlank(userId)) {

//在用户未登录的状态下

//.........

} else {

// 在用户已登录的状态

Map<String,TbItem> cart = this.getCart(userId);

cart.remove(itemId.toString());

//将新的购物车缓存到 Redis 中

this.addCartToRedis(userId,cart);

return Result.ok();

}

} catch (Exception e) {

e.printStackTrace();

}

return Result.error("删除失败");

}

# **购物车同步**

## **业务逻辑**

在用户未登录情况下往购物车添加商品，添加结束用户登录，此时需把cookie中的数据同步到redis：

​ a) 如果redis中有数据cookie中也有数据，需要做数据合并。相同商品数量相加，不同商品添加一个新商品。

​ b) 把cookie中的数据删除，避免下次重复同步

# **订单分析**

## **订单功能**

1、在购物车页面点击“去结算”按钮跳转到订单确认页面

​ a) 展示商品列表

​ b) 配送地址列表

​ c) 选择支付方式

2、订单确认页面需要根据用户查询配送地址，展示订单确认页面之前，应该确认用户身份

​ a) 如果用户未登录或登录过期跳转到登录页面

​ d) 登录成功后再跳转到订单确认页面

3、提交订单

​ a) 生成订单

4、扣减库存

5、关闭超时订单

​ a) 定时扫描超时2天未付款的订单，并关闭订单

​ b) 加回库存

## **创建工程**

### **usian\_order\_service**

### **usian\_order\_web**

### **usian\_order\_feign**

## 用户身份认证

在展示订单确认页面之前，需要对用户身份进行认证，要求用户必须登录。

## **功能分析**

1、使用springmvc的拦截器拦截所有订单的请求

2、业务逻辑

​ a) 从cookie中取token。

​ b) 根据token调用sso服务查询用户信息。

​ d) 如果查不到用户信息则跳转到登录页面。

​ e) 查询到用户信息放行。

## **usian\_order\_web**

### **interceptor**

/\*\*

\* 在结算之前判断用户是否登录

\*/

@Component

public class UserLoginInterceptor implements HandlerInterceptor {

@Autowired

private SSOServiceFeign ssoServiceFeign;

@Override

public boolean preHandle(HttpServletRequest request, HttpServletResponse

response, Object handler) throws Exception {

//对用户的 token 做判断

String token = request.getParameter("token");

if (StringUtils.isBlank(token)) {

return false;

}

//如果用户 token 不为空，则校验用户在 redis 中是否失效

TbUser tbUser = ssoServiceFeign.getUserByToken(token);

if (tbUser == null) {

return false;

}

return true;

}

}

### **config**

/\*\*

\* 拦截器配置类

\*/

@Component

public class WebConfig implements WebMvcConfigurer {

@Autowired

private UserLoginInterceptor userLoginInterceptor;

/\*\*

\* 注册拦截器

\* @param registry

\*/

@Override

public void addInterceptors(InterceptorRegistry registry) {

InterceptorRegistration registration =

registry.addInterceptor(this.userLoginInterceptor);

//拦截那个 URI

registration.addPathPatterns("/frontend/order/\*\*");

}

}

## 展示订单确认页面

## **功能分析**

1、在购物车页面点击“去结算”按钮跳转到订单确认页面。

2、请求的url：/frontend/order/goSettlement

3、参数：ids，userId，token

4、查询redis中的购物车数据并返回给前端

5、配送地址列表，需要用户登录。需要根据用户id查询收货地址列表。静态数据。

6、支付方式。静态数据。

## **usian\_order\_web**

### **controller**

/\*\*

\* 订单服务 Controller

\*/

@RestController

@RequestMapping("/frontend/order")

public class OrderController {

@Autowired

private CartServiceFeign cartServiceFeign;

@Autowired

private OrderServiceFeign orderServiceFeign;

@RequestMapping("/goSettlement")

public Result goSettlement(String[] ids, String userId) {

//获取购物车

Map<String, TbItem> cart = cartServiceFeign.selectCartByUserId(userId);

//从购物车中获取选中的商品

List<TbItem> list = new ArrayList<TbItem>();

for (String id : ids) {

list.add(cart.get(id));

}

if(list.size()>0) {

return Result.ok(list);

}

return Result.error("error");

}

}