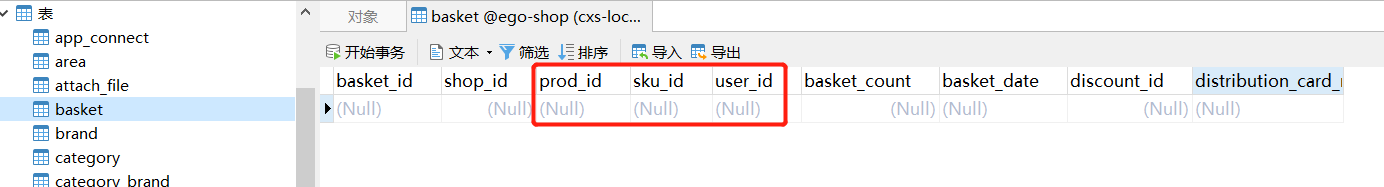
# EGO购物车

## 购物车表的设计



Prod\_id:商品的id

sku\_id：具体单个商品的id

User\_id:用户的openId

basket\_count：购物车商品的数量（一个商品买了几个）

basket\_date：添加购物车的时间

在表中 根据user\_id和sku\_id可以确定唯一一条数据

## 购物车数据结构的设计

**从用户出发：一个用户------>一个购物车**

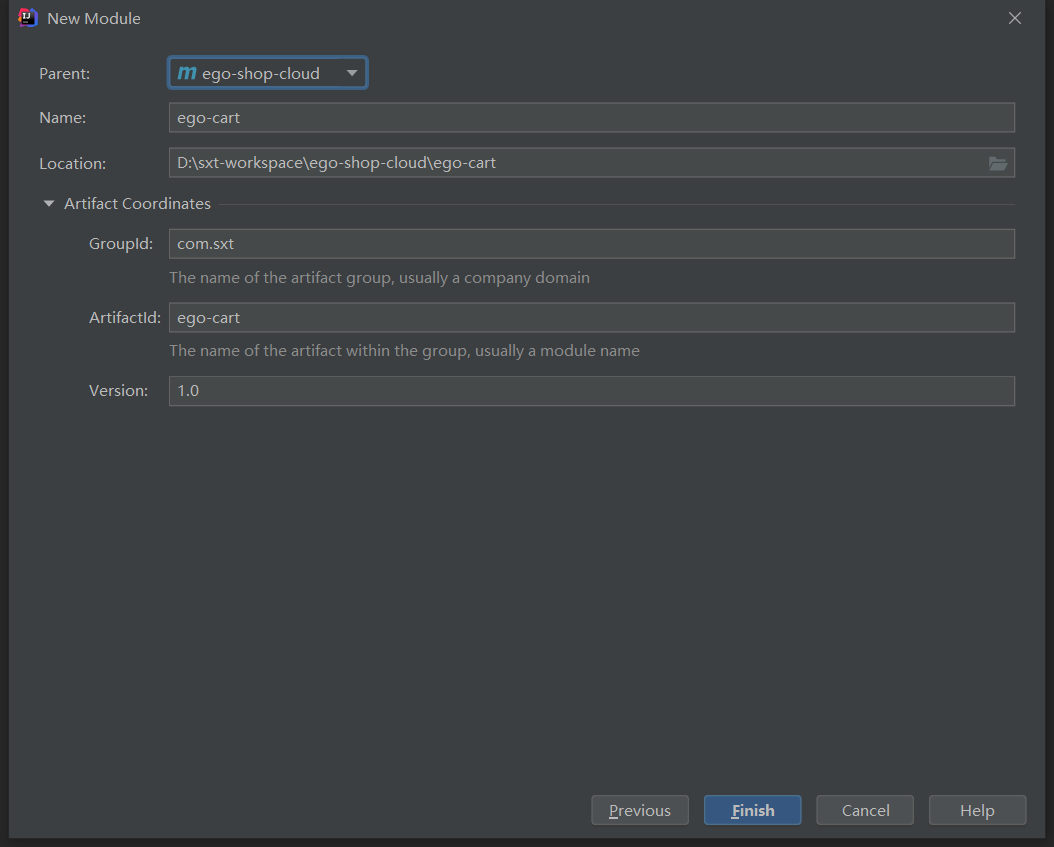
**userId -----> List<商品的sku>**

**所以购物车的数据结构是：**

**Map<String,List<sku>> carts**

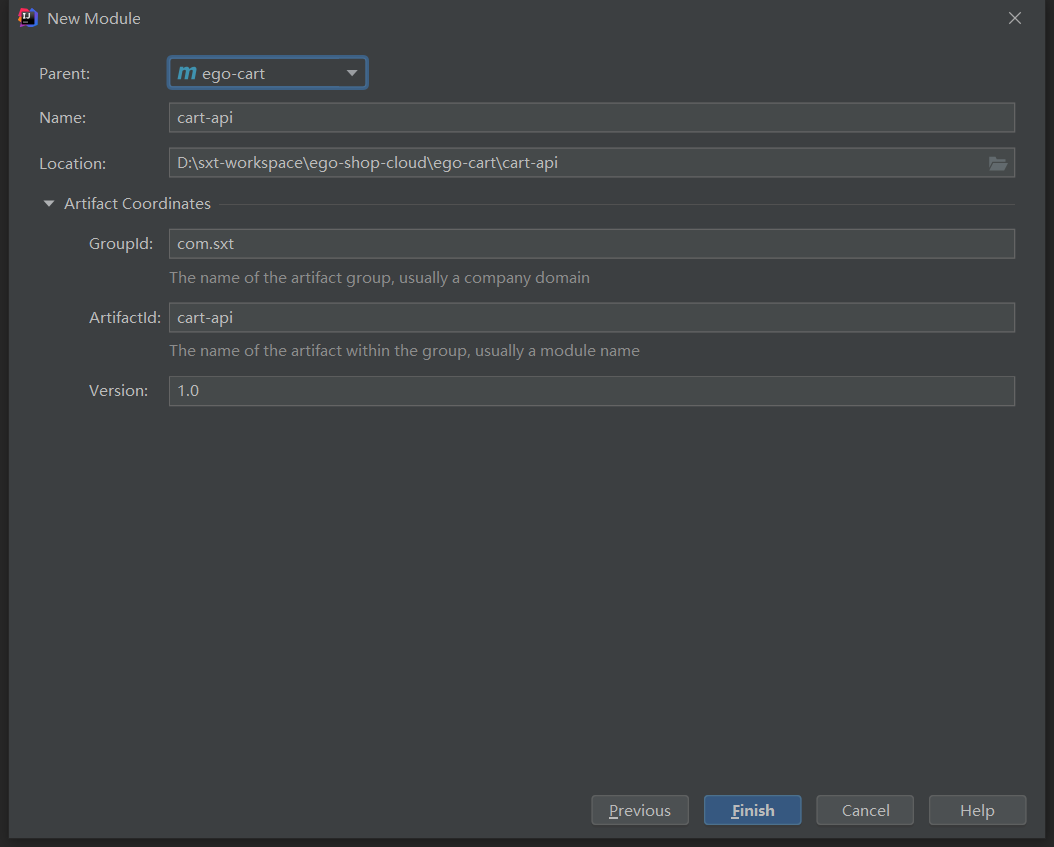
## 新建购物车模块

### 在ego-shop-cloud下面新建ego-cart模块



### 在ego-cart下面新建cart-api模块

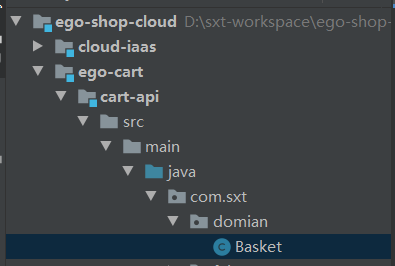
#### 新建maven模块



#### 修改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">  <parent>  <artifactId>ego-cart</artifactId>  <groupId>com.sxt</groupId>  <version>1.0</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>cart-api</artifactId>  <description>ego商城购物车api</description>  <dependencies>  <dependency>  <groupId>com.sxt</groupId>  <artifactId>ego-common</artifactId>  <version>1.0</version>  </dependency>  <dependency>  <groupId>com.sxt</groupId>  <artifactId>product-api</artifactId>  <version>1.0</version>  <exclusions>  <exclusion>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-elasticsearch</artifactId>  </exclusion>  </exclusions>  </dependency>  </dependencies>  </project> |

#### 逆向生成表basket



#### 创建远程调用的CartFeign

|  |
| --- |
| @FeignClient(value = "product-service", fallback = CartFeignHystrix.class)  public interface CartFeign {  /\*\*  \* 远程调用商品服务，计算购物车选中的价格  \*  \* @param skuId  \* @return  \*/  @GetMapping("/prod/prod/getCartTotalMoney")  BigDecimal getTotalMoney(@RequestParam("skuId") Long skuId, @RequestParam("count") Integer count);  /\*\*  \* 远程调用商品服务，拿到购物车中的信息  \*  \* @param skuId  \* @param prodId  \* @return  \*/  @GetMapping("cartGetProdInfo")  Map<String, String> cartGetProdInfo(@RequestParam("skuId") Long skuId, @RequestParam("prodId") Long prodId);  } |

#### 创建CartFeignHystrix熔断

|  |
| --- |
| @Component  @Slf4j  public class CartFeignHystrix implements CartFeign {  /\*\*  \* 远程调用商品服务，计算购物车选中的价格  \*  \* @param skuId  \* @param count  \* @return  \*/  @Override  public BigDecimal getTotalMoney(Long skuId, Integer count) {  log.error("远程调用商品服务计算价格失败♧");  return BigDecimal.ZERO;  }  @Override  public Map<String, String> cartGetProdInfo(Long skuId, Long prodId) {  log.error("远程调用获取购物车商品信息");  return new HashMap<String, String>();  }  } |

#### 创建购物车总金额的实体类

|  |
| --- |
| @Data  @Builder  public class CartTotalMoney {  @ApiModelProperty("最终金额")  private BigDecimal finalMoney;  @ApiModelProperty("总的金额")  private BigDecimal totalMoney;  @ApiModelProperty("满减金额")  private BigDecimal subtractMoney;  @ApiModelProperty("总的商品数量")  private Integer prodTotalCount;  public BigDecimal getFinalMoney() {  if (totalMoney == null) {  return new BigDecimal("0.00");  }  if (subtractMoney == null) {  return totalMoney;  }  return totalMoney.add(BigDecimal.valueOf(-1).multiply(subtractMoney)).setScale(2, BigDecimal.ROUND\_UP);  }  } |

#### 创建购物车展示类（每一行是一个对象）

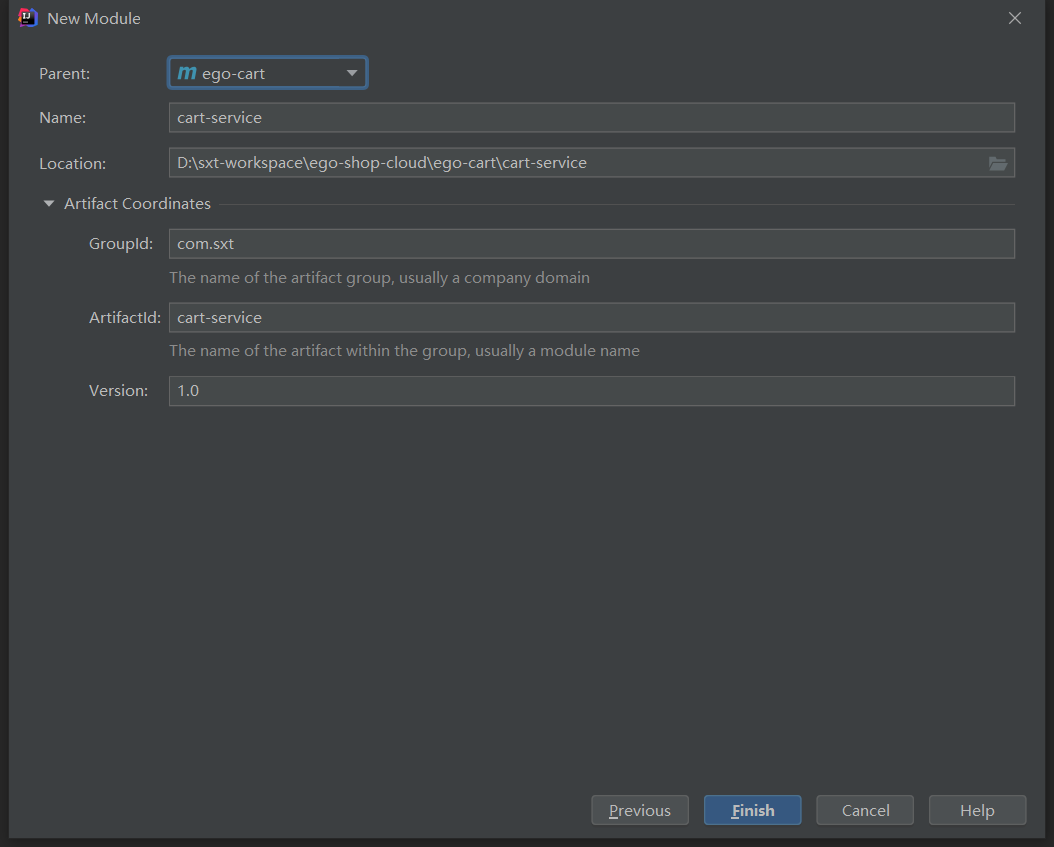
|  |
| --- |
| /\*\*  \* 购物车里面的一个条目  \*/  @Data  public class ShopCartItem implements Serializable {  @ApiModelProperty("购物车的id")  private Long basketId;  @ApiModelProperty("该条目是否被选择")  private Boolean checked;  @ApiModelProperty("该条目对应的商品id")  private Long prodId;  @ApiModelProperty("该条目对应的商品的名称")  private String prodName;  @ApiModelProperty("该条目SkuId")  private Long skuId;  @ApiModelProperty("该条目的名称")  private String skuName;  @ApiModelProperty("该条目对应的主图")  private String pic;  @ApiModelProperty("该条目对应的价格")  private String price;  @ApiModelProperty("该条目对应数量")  private Integer basketCount;  } |

#### 创建购物车查询返回对象（用于前端渲染）

|  |
| --- |
| @Data  @NoArgsConstructor  @AllArgsConstructor  public class ShopCartResult implements Serializable {  @ApiModelProperty  private List<ShopCartItem> shopCartItems = Collections.emptyList();  } |

### 在ego-cart下面新建cart-service模块

#### 新建maven模块



#### 修改pom.xml文件

|  |
| --- |
| <?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">  <parent>  <artifactId>ego-cart</artifactId>  <groupId>com.sxt</groupId>  <version>1.0</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>cart-service</artifactId>  <description>ego商城购物车服务service</description>  <dependencies>  <dependency>  <groupId>com.sxt</groupId>  <artifactId>cart-api</artifactId>  <version>1.0</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-actuator</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-bus-amqp</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-config-client</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-redis-reactive</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  </dependency>  <!-- 引入druid starter-->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid-spring-boot-starter</artifactId>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

#### 添加启动类

|  |
| --- |
| @SpringBootApplication  @EnableEurekaClient  @EnableFeignClients  public class CartServiceApplication {  public static void main(String[] args) {  SpringApplication.run(CartServiceApplication.class, args);  }  } |

#### 创建bootstrap.yml配置文件

|  |
| --- |
| spring:  application:  name: cart-service  cloud:  config:  discovery:  service-id: config-server  enabled: true  name: cart-service  label: master  profile: dev  eureka:  client:  service-url:  defaultZone: ${EUREKA\_SERVER:http://admin:admin@localhost:8761/eureka/}  instance:  hostname: ${APP\_HOST:localhost}  prefer-ip-address: true  instance-id: ${spring.application.name}:${server.port}  lease-renewal-interval-in-seconds: 10  lease-expiration-duration-in-seconds: 30 |

#### 创建远端cart-service/cart-service-dev.yml

|  |
| --- |
| server:  port: ${APP\_PORT:8087}  spring:  datasource:  druid:  url: jdbc:mysql://localhost:3306/ego-shop?useSSL=false&serverTimezone=GMT%2B8  username: root  password: root  driver-class-name: com.mysql.cj.jdbc.Driver  max-active: 30  min-idle: 3  initial-size: 3  validation-query: select 1  max-wait: 5000  stat-view-servlet:  enabled: true  login-username: admin  login-password: admin  allow:  deny:  url-pattern:  - /druid/\*  rabbitmq:  host: 192.168.226.129  password: admin  username: admin  port: 5672  redis:  host: 192.168.226.129  port: 6380  password: cxs1013??  database: 0  management:  endpoints:  web:  exposure:  include: '\*'  feign:  hystrix:  enabled: true  ribbon:  ReadTimeout: 18000  ConnectTimeout: 18000  hystrix: #hystrix的全局控制  command:  default: #default是全局控制，也可以换成单个方法控制，把default换成方法名即可  fallback:  isolation:  semaphore:  maxConcurrentRequests: 1000  circuitBreaker:  enabled: true #开启断路器  requestVolumeThreshold: 20 #失败次数（阀值）  sleepWindowInMilliseconds: 30000 #窗口时间  errorThresholdPercentage: 60 #失败率  execution:  isolation:  strategy: SEMAPHORE #使用信号量隔离方式  thread:  timeoutInMilliseconds: 5000  semaphore:  maxConcurrentRequests: 1000 #最大并发量  mybatis-plus:  mapper-locations: classpath:mapper/\*\*/\*.xml  configuration:  log-impl: org.apache.ibatis.logging.stdout.StdOutImpl  swagger2:  base-package: com.sxt.controller  name: sxt  url: https://gitee.com/smiledouble  email: 775610843@qq.com  title: 购物车管理系统接口  description: 实现购物车管理系统的所有接口  license: cxs  license-url: https://gitee.com/smiledouble  terms-of-service-url: https://gitee.com/smiledouble |

#### 创建BasketController

|  |
| --- |
| @RestController  public class BasketController {  @Autowired  private BasketService basketService;  /\*\*  \* 不分页  \* 根据用户id查询购物车里的信息  \*  \* @return  \*/  @PostMapping("/p/shopCart/info")  @ApiOperation("回显购物车的详情")  public ResponseEntity<List<ShopCartResult>> basketResultInfo() {  String userId = SecurityContextHolder.getContext().getAuthentication().getPrincipal().toString();  List<ShopCartResult> shopCartResult = this.basketService.getShopCartResult(userId);  return ResponseEntity.ok(shopCartResult);  }  /\*\*  \* 添加等  \*  \* @param basket  \* @return  \*/  @PostMapping({"/p/shopCart/changeItem"})  @ApiOperation("修改购物车的数量")  public ResponseEntity<Void> changeItem(@RequestBody @Validated Basket basket) {  String userId = SecurityContextHolder.getContext().getAuthentication().getPrincipal().toString();  basket.setUserId(userId);  this.basketService.changeCart(basket);  return ResponseEntity.ok().build();  }  @GetMapping({"p/shopCart/prodCount"})  @ApiOperation("获取当前用户购物车的总条数")  public ResponseEntity<Integer> getBasketTotalCount() {  String userId = SecurityContextHolder.getContext().getAuthentication().getPrincipal().toString();  Integer totalCount = this.basketService.getBasketTotalCount(userId);  return ResponseEntity.ok(totalCount);  }  @PostMapping("/p/shopCart/totalPay")  @ApiModelProperty("计算总的金额")  public ResponseEntity<CartTotalMoney> getTotalMoney(@RequestBody List<Long> basketIds) {  if (basketIds == null || basketIds.isEmpty()) {  CartTotalMoney cartTotalMoney = CartTotalMoney.builder().  subtractMoney(new BigDecimal("0.00")).  totalMoney(new BigDecimal("0.00")).build();  return ResponseEntity.ok(cartTotalMoney);  }  CartTotalMoney cartTotalMoney = basketService.getCartTotalMoney(basketIds);  return ResponseEntity.ok(cartTotalMoney);  }  @DeleteMapping({"/p/shopCart/deleteItem"})  @ApiOperation("清空购物车")  public ResponseEntity<Void> deleteItem(@RequestBody List<Long> basketIds) {  this.basketService.removeByIds(basketIds);  return ResponseEntity.ok().build();  }  } |

#### 修改BasketService

|  |
| --- |
| public interface BasketService extends IService<Basket> {  /\*\*  \* 获取购物车的总条数  \*  \* @param userId  \* @return  \*/  Integer getBasketTotalCount(String userId);  /\*\*  \* 计算购物车总金额  \*  \* @param basketIds  \* @return  \*/  CartTotalMoney getCartTotalMoney(List<Long> basketIds);  /\*\*  \* 修改购物车某商品数量  \*  \* @param basket  \*/  void changeCart(Basket basket);  /\*\*  \* 回显购物车的数据  \*  \* @param userId  \* @return  \*/  List<ShopCartResult> getShopCartResult(String userId);  } |

#### 修改BasketServiceImpl

|  |
| --- |
| @Service  @Slf4j  public class BasketServiceImpl extends ServiceImpl<BasketMapper, Basket> implements BasketService {  @Autowired  private BasketMapper basketMapper;  @Autowired  private CartFeign cartFeign;  /\*\*  \* 获取购物车的总条数  \*  \* @param userId  \* @return  \*/  @Override  public Integer getBasketTotalCount(String userId) {  //根据用户id查询所有的商品 累加数目  List<Object> count = basketMapper.selectObjs(new QueryWrapper<Basket>()  .select("SUM(basket\_count)")  .eq("user\_id", userId)  );  if (count != null && count.size() > 0) {  return Integer.valueOf(count.get(0).toString());  }  return 0;  }  /\*\*  \* 计算购物车总金额  \* 根据列表中的sku\_id 远程调用商品服务，查询价格后 累加计算  \*  \* @param basketIds  \* @return  \*/  @Override  @Transactional  public CartTotalMoney getCartTotalMoney(List<Long> basketIds) {  //查购物车的表  List<Basket> baskets = basketMapper.selectList(new LambdaQueryWrapper<Basket>()  .in(Basket::getBasketId, basketIds)  );  //总金额  BigDecimal totalMoney = BigDecimal.ZERO;  //满减金额  BigDecimal subtractMoney = BigDecimal.ZERO;  //总数量  Integer prodTotalCount = 0;  if (baskets != null && basketIds.size() > 0) {  //循环拿到sku\_id  for (Basket basket : baskets) {  //远程调用商品服务，计算价格  totalMoney = totalMoney.add(cartFeign.getTotalMoney(basket.getSkuId(), basket.getBasketCount()));  prodTotalCount += basket.getBasketCount();  }  }  //组装数据返回  CartTotalMoney cartTotalMoney = CartTotalMoney.builder()  .prodTotalCount(prodTotalCount)  .subtractMoney(subtractMoney)  .totalMoney(totalMoney.setScale(2, BigDecimal.ROUND\_HALF\_UP))  .build();  return cartTotalMoney;  }  /\*\*  \* 修改购物车数量 添加购物车  \*  \* @param basket  \*/  @Override  @Transactional  public void changeCart(Basket basket) {  //先看购物车里有没有这个商品 有的话就修改数目 没有的话就增加  Basket oldBasket = basketMapper.selectOne(new LambdaQueryWrapper<Basket>()  .eq(Basket::getSkuId, basket.getSkuId())  .eq(Basket::getUserId, basket.getUserId())  .eq(Basket::getShopId, 1)  );  if (oldBasket != null) {  //如果有,就修改数量  Integer basketCount = oldBasket.getBasketCount();  int count = basketCount + basket.getBasketCount();  if (count < 0) {  throw new IllegalArgumentException("参数有问题，商品数量不能为空");  }  if (count == 0) {  //删除  super.removeById(oldBasket.getBasketId());  return;  }  //修改  oldBasket.setBasketCount(count);  oldBasket.setBasketDate(new Date());  basketMapper.updateById(oldBasket);  }else {  //新增  basket.setBasketCount(1);  basket.setBasketDate(new Date());  basketMapper.insert(basket);  }  }  /\*\*  \* 回显购物车的数据  \* 根据用户id查询购物车  \* 组装数据  \*  \* @param userId  \* @return  \*/  @Override  @Transactional  public List<ShopCartResult> getShopCartResult(String userId) {  List<Basket> baskets = basketMapper.selectList(new LambdaQueryWrapper<Basket>()  .eq(Basket::getUserId, userId)  .eq(Basket::getShopId, 1)  .orderByDesc(Basket::getBasketDate)  );  if (ObjectUtils.isEmpty(baskets)) {  return Collections.emptyList();  }  //返回的结果集  List<ShopCartResult> shopCartResults = new ArrayList<>();  //购物车商品的结果集  List<ShopCartItem> shopCartItems = new ArrayList<>(baskets.size() \* 2);  //循环 组装数据  baskets.forEach(basket -> {  ShopCartItem shopCartItem = new ShopCartItem();  shopCartItem.setBasketId(basket.getBasketId());  shopCartItem.setBasketCount(basket.getBasketCount());  shopCartItem.setProdId(basket.getProdId());  shopCartItem.setSkuId(basket.getSkuId());  //远程调用把skuId和prodId 一起传过去 只调用一次  // 拿到skuName,skuPrice skuPic, 和 prodName我们用map封装接受一下  Map<String, String> map = cartFeign.cartGetProdInfo(basket.getSkuId(), basket.getProdId());  shopCartItem.setSkuName(map.get("skuName"));  shopCartItem.setPic(map.get("skuPic"));  shopCartItem.setPrice(map.get("skuPrice"));  shopCartItem.setProdName(map.get("prodName"));  //设置商品  shopCartItems.add(shopCartItem);  });  ShopCartResult shopCartResult = new ShopCartResult();  shopCartResult.setShopCartItems(shopCartItems);  shopCartResults.add(shopCartResult);  return shopCartResults;  }  } |