谷粒商城

版本：V 1.0

订单

# 业务介绍



**当前的位置**

订单业务在整个电商平台中处于核心位置，也是比较复杂的一块业务。是把“物”变为“钱”的一个中转站。

整个订单模块一共分四部分组成：

1. 结算
2. 下单
3. 对接支付服务
4. 对接库存管理系统：

# 结算页

入口：购物车点击计算按钮 ，结算必须要登录！



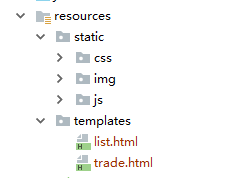
## 1 搭建模块

### 1.1 gmall-order-web（模块已添加）

application.properties

|  |
| --- |
| **spring.dubbo.application.name**=**order-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**=**10000 spring.dubbo.consumer.check**=**false  server.port**=**8081 spring.thymeleaf.cache**=**false spring.thymeleaf.mode**=**LEGACYHTML5**  **mapper.enum-as-simple-type**=**true** |

拷贝静态资源文件和html



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 order.gmall.com |

nginx.conf

|  |
| --- |
| upstream order.gmall.com{  server 192.168.67.1:8081;  }  server {  listen 80;  server\_name order.gmall.com;  location / {  proxy\_pass http://order.gmall.com;  proxy\_set\_header X-forwarded-for $proxy\_add\_x\_forwarded\_for;  }  } |

## 2 分析

分析页面需要的数据：

1. 得到用户信息
2. 购物车中选择的商品列表

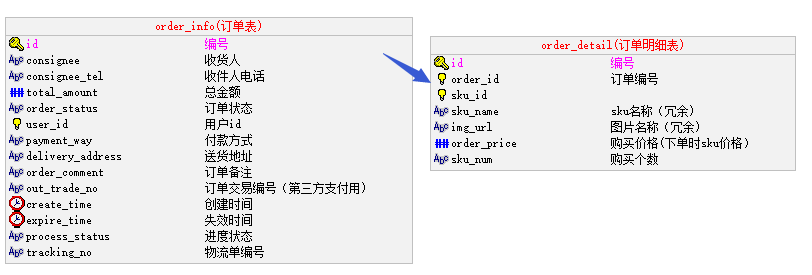
## 3 购物车中选择的商品列表

结算的时候，需要将信息添加到OrderDetail 表中！

### 3.1 订单的数据结构：

orderInfo ：订单表

orderDetail ：订单明细



|  |  |
| --- | --- |
| id | 主键。自动生成 |
| consignee | 收货人名称。页面获取 |
| consignee\_tel | 收货人电话。页面获取 |
| deliveryAddress | 收货地址。页面获取 |
| total\_amount | 总金额。计算 |
| order\_status | 订单状态，用于显示给用户查看。设定初始值。 |
| userId | 用户Id。从拦截器已放到请求属性中。 |
| payment\_way | 支付方式（网上支付、货到付款）。页面获取 |
| orderComment | 订单状态。页面获取 |
| out\_trade\_no | 第三方支付编号。按规则生成 |
| create\_time | 创建时间。设当前时间 |
| expire\_time | 默认当前时间+1天 |
| process\_status | 订单进度状态，程序控制、 后台管理查看。设定初始值， |
| tracking\_no | 物流编号,初始为空，发货后补充 |
| parent\_order\_id | 拆单时产生，默认为空 |

|  |  |
| --- | --- |
| id | 主键，自动生成 |
| order\_id | 订单编号，主表保存后给从表 |
| sku\_id | 商品id 页面传递 |
| sku\_name | 商品名称，后台添加 |
| img\_url | 图片路径，后台添加 |
| order\_price | 商品单价，从页面中获取，并验价。 |
| sku\_num | 商品个数，从页面中获取 |

增加实体Bean

|  |
| --- |
| **public class** OrderInfo **implements** Serializable { @Column @Id @GeneratedValue(strategy = GenerationType.***IDENTITY***) **private** String **id**;  @Column **private** String **consignee**;  @Column **private** String **consigneeTel**;   @Column **private** BigDecimal **totalAmount**;  @Column **private** OrderStatus **orderStatus**;  @Column **private** ProcessStatus **processStatus**;   @Column **private** String **userId**;  @Column **private** PaymentWay **paymentWay**;  @Column **private** Date **expireTime**;  @Column **private** String **deliveryAddress**;  @Column **private** String **orderComment**;  @Column **private** Date **createTime**;  @Column **private** String **parentOrderId**;  @Column **private** String **trackingNo**;   @Transient **private** List<OrderDetail> **orderDetailList**;   @Transient **private** String **wareId**;  @Column **private** String **outTradeNo**;  **public void** sumTotalAmount(){  BigDecimal totalAmount=**new** BigDecimal(**"0"**);  **for** (OrderDetail orderDetail : **orderDetailList**) {  totalAmount= totalAmount.add(orderDetail.getOrderPrice().multiply(**new** BigDecimal(orderDetail.getSkuNum())));  }  **this**.**totalAmount**= totalAmount; }  } |

增加枚举类

枚举类路径放到gmall-bean 模块中和bean同级目录。

在bean下建立order\_detail

|  |
| --- |
| **public class** OrderDetail **implements** Serializable {  @Id  @Column  **private** String **id**;  @Column  **private** String **orderId**;  @Column  **private** String **skuId**;  @Column  **private** String **skuName**;  @Column  **private** String **imgUrl**;  @Column  **private** BigDecimal **orderPrice**;  @Column  **private** Integer **skuNum**;   @Transient  **private** String **hasStock**;  } |

其中hasStock是一个非持久化属性，用户传递【是否还有库存】的标志。

如果商品在库存中有足够数据，suceess = “1”，fail=“0”

### 3.2 OrderController

加入trade方法

|  |
| --- |
| @Reference **private** CartService **cartService**;  @GetMapping(**"trade"**) @LoginRequire **public** String trade(HttpServletRequest request){  String userId =(String)request.getAttribute(**"userId"**);  *// 用户地址 列表* List<UserAddress> userAddressList = **userService**.getUserAddressList(userId);   request.setAttribute(**"userAddressList"**,userAddressList);   *// 用户需要结账的商品清单* List<CartInfo> checkedCartList = **cartService**.getCheckedCartList(userId);  BigDecimal totalAmount = **new** BigDecimal(**"0"**);  **for** (CartInfo cartInfo : checkedCartList) {  BigDecimal cartInfoAmount = cartInfo.getSkuPrice().multiply(**new** BigDecimal(cartInfo.getSkuNum()));  totalAmount= totalAmount.add(cartInfoAmount);  }  request.setAttribute(**"checkedCartList"**,checkedCartList);   request.setAttribute(**"totalAmount"**,totalAmount);   **return "trade"**; } |

### 3.1 在CartServiceImpl.java

|  |
| --- |
| *// 得到选中购物车列表* **public** List<CartInfo> getCartCheckedList(String userId){  *// 获得redis中的key* String userCheckedKey = CartConst.***USER\_KEY\_PREFIX*** + userId + CartConst.***USER\_CHECKED\_KEY\_SUFFIX***;  Jedis jedis = **redisUtil**.getJedis();  List<String> cartCheckedList = jedis.hvals(userCheckedKey);  List<CartInfo> newCartList = **new** ArrayList<>();  **for** (String cartJson : cartCheckedList) {  CartInfo cartInfo = JSON.*parseObject*(cartJson,CartInfo.**class**);  newCartList.add(cartInfo);  }  **return** newCartList; } |
| **接口**CartService  **public** List<CartInfo> getCartCheckedList(String userId) |

### 3.4 结算页面

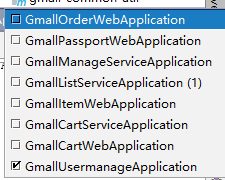
|  |
| --- |
| *<!--地址-->* <**div class="top-3"**>  <**ul** >  <**li class=".address default selected" th:each="userAddress :${userAddressList}"** >  <**input name="deliveryAddress" type="radio" value="" th:checked="${userAddress.isDefault}=='1'" checked="true"**>  <**span th:text="${userAddress.consignee}"** >收件人AAAA </**span**><**span th:text="${userAddress.userAddress}"** >具体地址111 </**span**>  </**li**>  </**ul**> </**div**> |

商品清单部分

|  |
| --- |
| <**div class="to\_right"**>  <**h5**>商家：自营</**h5**>  *<!--图片-->* <**div class="yun1" th:each="orderDetail: ${orderDetailList}"** >  <**img src="" th:src="${orderDetail.imgUrl}" class="yun"**/>  <**div class="mi"**>  <**div**><**p style="width**: 500**px**;**" th:text="${orderDetail.skuName}"**>商品名称111111111111 </**p**> <**span style="float**: **right"**> <**span align="center" style="color**: **red"** > </**span**> <**span th:text="'X'+${orderDetail.skuNum}"** > X 1 </**span**> <**span** >有货</**span**> <**span** >无货</**span**> </**span**> </**div**>  </**div**>  </**div**>  </**div**> |

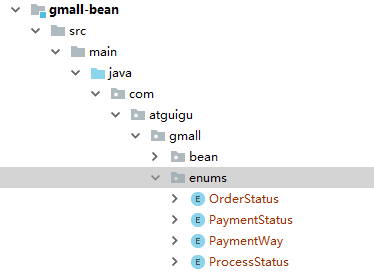
总价钱

|  |
| --- |
| <**p class="yfze\_a"**><**span class="z"**>应付总额：</**span**><**span class="hq" th:text="'￥'+${totalAmount}"** > ￥99999</**span**></**p**> |



# 三、下订单

## 1 数据结构



|  |
| --- |
| **public enum** OrderStatus {  ***UNPAID***(**"未支付"**),  ***PAID***(**"已支付"** ),  ***WAITING\_DELEVER***(**"待发货"**),  ***DELEVERED***(**"已发货"**),  ***CLOSED***(**"已关闭"**),  ***FINISHED***(**"已完结"**) ,  ***SPLIT***(**"订单已拆分"**);   **private** String **comment** ;    OrderStatus(String comment ){  **this**.**comment**=comment;  }   **public** String getComment() {  **return comment**;  }   **public void** setComment(String comment) {  **this**.**comment** = comment;  }   } |

|  |
| --- |
| **public enum** ProcessStatus {  ***UNPAID***(**"未支付"**,OrderStatus.***UNPAID***),  ***PAID***(**"已支付"**,OrderStatus.***PAID***),  ***NOTIFIED\_WARE***(**"已通知仓储"**,OrderStatus.***PAID***),  ***WAITING\_DELEVER***(**"待发货"**,OrderStatus.***WAITING\_DELEVER***),  ***STOCK\_EXCEPTION***(**"库存异常"**,OrderStatus.***PAID***),  ***DELEVERED***(**"已发货"**,OrderStatus.***DELEVERED***),  ***CLOSED***(**"已关闭"**,OrderStatus.***CLOSED***),  ***FINISHED***(**"已完结"**,OrderStatus.***FINISHED***) ,  ***PAY\_FAIL***(**"支付失败"**,OrderStatus.***UNPAID***),  ***SPLIT***(**"订单已拆分"**,OrderStatus.***SPLIT***);   **private** String **comment** ;  **private** OrderStatus **orderStatus**;   ProcessStatus(String comment, OrderStatus orderStatus){  **this**.**comment**=comment;  **this**.**orderStatus**=orderStatus;  }   **public** String getComment() {  **return comment**;  }   **public void** setComment(String comment) {  **this**.**comment** = comment;  }   **public** OrderStatus getOrderStatus() {  **return orderStatus**;  }   **public void** setOrderStatus(OrderStatus orderStatus) {  **this**.**orderStatus** = orderStatus;  } |

|  |
| --- |
| **public enum** PaymentWay {  ***ONLINE***(**"在线支付"**),  ***OUTLINE***(**"货到付款"** );   **private** String **comment** ;    PaymentWay(String comment ){  **this**.**comment**=comment;  }   **public** String getComment() {  **return comment**;  }   **public void** setComment(String comment) {  **this**.**comment** = comment;  }   } |

由于涉及枚举类所以 application.properties中要加入

|  |
| --- |
| **mapper.enum-as-simple-type**=**true** |

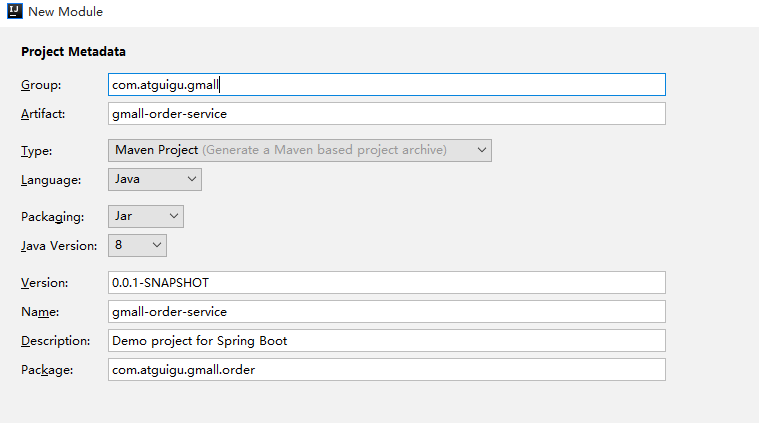
这个配置会把枚举类当成字符串处理。

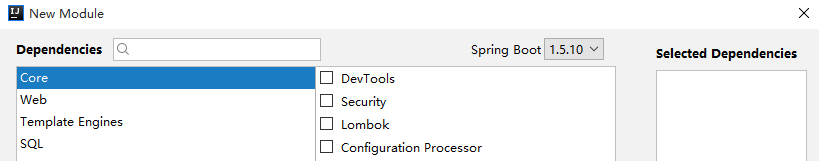
## 2 分析下单：

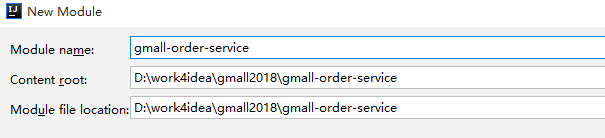
1. 保存单据前要做交易：验库存。
2. 保存单据: orderInfo orderDetail。
3. 保存以后把购物车中的商品删除。
4. 重定向到支付页面。

## 3 订单后台服务模块

### 3.1 搭建gmall-order-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-order-service</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>jar</packaging>   <name>gmall-order-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

|  |
| --- |
| server.port=8090  logging.level.root=debug  spring.dubbo.application.name=order-service 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  spring.datasource.url=jdbc:mysql://localhost:3306/gmall?characterEncoding=UTF-8 spring.datasource.username=root spring.datasource.password=root   mybatis.mapper-locations=classpath:mapper/\*Mapper.xml mybatis.configuration.mapUnderscoreToCamelCase=true  spring.redis.host=192.168.67.202 spring.redis.port=6379 spring.redis.database=0  **mapper.enum-as-simple-type**=**true** |

GmallOrderServiceApplication

|  |
| --- |
| @SpringBootApplication @MapperScan(basePackages = **"com.atguigu.gmall.order.mapper"**) @ComponentScan(basePackages = **"com.atguigu.gmall"**) **public class** GmallOrderServiceApplication {   **public static void** main(String[] args) {  SpringApplication.*run*(GmallOrderServiceApplication.**class**, args);  } } |

### 3.2 OrderService

|  |
| --- |
| **接口：返回orderId，保存完，应该调到支付，根据orderId。**  **public String** saveOrder(OrderInfo orderInfo); |
| **Mapper：**  **public interface** OrderInfoMapper **extends** Mapper<OrderInfo>{   } |

### 3.3 OrderServiceImpl

|  |  |
| --- | --- |
| @Service **public class** OrderServiceImpl **implements** OrderService{   @Autowired  **private** OrderInfoMapper **orderInfoMapper**;  @Autowired  **private** OrderDetailMapper **orderDetailMapper**;     @Override @Transactional **public void** saveOrder(OrderInfo orderInfo) {   **orderInfoMapper**.insertSelective(orderInfo);   List<OrderDetail> orderDetailList = orderInfo.getOrderDetailList();  **for** (OrderDetail orderDetail : orderDetailList) {  orderDetail.setOrderId(orderInfo.getId());  **orderDetailMapper**.insertSelective(orderDetail);  }  } |  |

### 3.4 OrderController

|  |
| --- |
| @RequestMapping(value = **"submitOrder"**,method = RequestMethod.***POST***) @LoginRequire **public** String submitOrder(OrderInfo orderInfo,HttpServletRequest request){  *// 检查tradeCode* String userId = (String) request.getAttribute(**"userId"**);  *// 初始化参数* orderInfo.setOrderStatus(OrderStatus.***UNPAID***);  orderInfo.setProcessStatus(ProcessStatus.***UNPAID***);  orderInfo.sumTotalAmount();  orderInfo.setUserId(userId);  *// 保存* String orderId = **orderService**.saveOrder(orderInfo);  *// 重定向* **return "redirect://payment.gmall.com/index?orderId="**+orderId; } |

### 3.5 页面trade.html

|  |
| --- |
| <**form action="./submitOrder" method="post" id="orderForm"**>  <**input name="consignee" id="consignee" type="hidden"**/>  <**input name="deliveryAddress" id="deliveryAddress" type="hidden"**/>  <**input name="paymentWay" id="paymentWay" type="hidden"**/>  <**input name="orderComment" id="orderComment" type="hidden"**/>  <**span th:each="orderDetail,stat: ${orderDetailList}"** >  <**input th:name="'orderDetailList['+${stat.index}+'].skuId'" type="hidden" th:value="${orderDetail.skuId}"** />  <**input th:name="'orderDetailList['+${stat.index}+'].skuNum'" type="hidden" th:value="${orderDetail.skuNum}"** />  <**input th:name="'orderDetailList['+${stat.index}+'].orderPrice'" type="hidden" th:value="${orderDetail.orderPrice}"** />  <**input th:name="'orderDetailList['+${stat.index}+'].skuName'" type="hidden" th:value="${orderDetail.skuName}"** />  </**span**>  <**input name="tradeNo" type="hidden" value="1"** /> </**form**> <**script**>  *$*(**function**() {   *$*(**"#submitButton"**).click(**function** () {  *// 收货人  $*(**"#consignee"**).val(*$*(**"input[type='radio']:checked"**).next().text()) ;  *// 收获地址  $*(**"#deliveryAddress"**).val( *$*(**"input[type='radio']:checked"**).next().next().text());  *$*(**"#paymentWay"**).val(**"ONLINE"**);  *// 订单备注  $*(**"#orderComment"**).val(*$*(**"#orderCommentPage"**).val());  **console**.log(*$*(**"#orderForm"**).html());  *$*(**"#orderForm"**).submit();   }); |

## 4 如何解决用户利用浏览器回退重复提交订单？

在进入结算页面时，生成一个结算流水号，然后保存到结算页面的隐藏元素中，每次用户提交都检查该流水号与页面提交的是否相符，订单保存以后把后台的流水号删除掉。那么第二次用户用同一个页面提交的话流水号就会匹配失败，无法重复保存订单。

### 4.1 修改结算页增加流水号的生成。

#### 4.1.1 OrderServiceImpl

|  |
| --- |
| @Override **public** String genToken(String userId) {  *//token type String key user:10201:trade\_code value token* String token = UUID.*randomUUID*().toString();  String tokenKey=**"user:"**+userId+**":trade\_code"**;  Jedis jedis = **redisUtil**.getJedis();  jedis.setex(tokenKey,10\*60,token);  jedis.close();   **return** token; }   @Override  **public boolean** verifyToken(String userId, String token) {  String tokenKey=**"user:"**+userId+**":trade\_code"**;  Jedis jedis = **redisUtil**.getJedis();  String tokenExists = jedis.get(tokenKey);  jedis.watch(tokenKey);  Transaction transaction = jedis.multi();  **if**(tokenExists!=**null**&&tokenExists.equals(token)){  transaction.del(tokenKey);  }  List<Object> list = transaction.exec();  **if**(list!=**null**&&list.size()>0&&(Long)list.get(0)==1L){  **return true**;  }**else**{  **return false**;  }   } |
| *将三个方法添加到接口中* |

4.1.2 在OrderController中添加tradeInit方法保存code编码

|  |
| --- |
| *// 获取TradeCode号* String token = **orderService**.genToken(userId);  request.setAttribute(**"tradeNo"**,token); |
| *在页面表单提交的时候，将TradeNo进行保存* |

### 4.2 OrderController中的submitOrder方法中

|  |
| --- |
| @PostMapping(**"submitOrder"**) @LoginRequire **public** String submitOrder(OrderInfo orderInfo,HttpServletRequest request){  String userId = (String) request.getAttribute(**"userId"**);   String tradeNo = request.getParameter(**"tradeNo"**);   **boolean** isEnableToken = **orderService**.verifyToken(userId, tradeNo);  **if**(!isEnableToken){  request.setAttribute(**"errMsg"**,**"页面已失效，请重新结算！"**);  **return "tradeFail"**;  }   orderInfo.setOrderStatus(OrderStatus.***UNPAID***);  orderInfo.setProcessStatus(ProcessStatus.***UNPAID***);  orderInfo.setCreateTime(**new** Date());  orderInfo.setExpireTime(DateUtils.*addMinutes*(**new** Date(),15));  orderInfo.sumTotalAmount();  orderInfo.setUserId(userId);   List<OrderDetail> orderDetailList = orderInfo.getOrderDetailList();  **for** (OrderDetail orderDetail : orderDetailList) {  SkuInfo skuInfo = **manageService**.getSkuInfo(orderDetail.getSkuId());  orderDetail.setImgUrl(skuInfo.getSkuDefaultImg());  orderDetail.setSkuName(skuInfo.getSkuName());  }    **orderService**.saveOrder(orderInfo);  **return "redirect://payment.gmall.com/index"**; } |

## 5 验库存

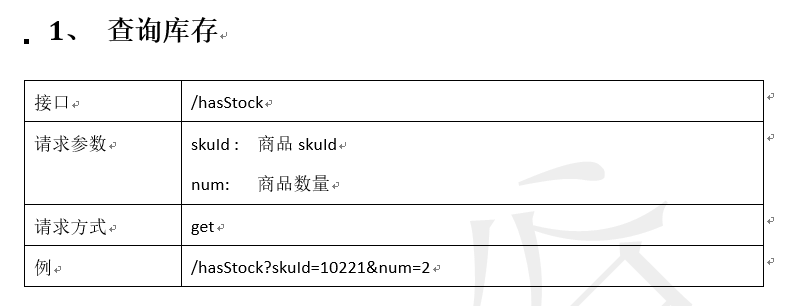
通过restful接口查询商品是否有库存

一般电商系统的商品库存，都不由电商系统本身来管理，由另外一套仓库管理系统，或者进销存系统来管理，电商系统通过第三方接口调用该系统。

由于库管系统可能是异构的系统，所以不在dubbo的分布式体系之内。只支持restful风格的webservice调用和消息队列的调用。

详见《库存管理系统手册》

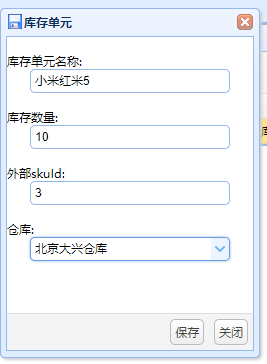
根据手册中的接口文档，编写调用代码。



### 导入项目gware-manage项目

详情查看库存管理系统文档

填写库存信息！



需要配置nginx

|  |
| --- |
| upstream www.gware.com {  server 192.168.67.1:9001;  }  server {  listen 80;  server\_name www.gware.com;  location /{  proxy\_pass http://www.gware.com;  proxy\_set\_header X-forwarded-for $proxy\_add\_x\_forwarded\_for;  }  } |

### 查询仓库数量，进行校验

**为什么要用多线程：**

由于库存系统假定为第三方系统， 有可能调用过程中延迟较大，如果订单中有多个商品，会串行操作执行时间为 订单个数\*单次访问时间。

由于商品之间检查库存不存在依赖关键，完全可以并行执行，那么执行时间等于 最长的那次单次访问时间。

**多线程操作Demo**

|  |
| --- |
| *// List 1,2,3,4,5,6,7,8,9 找出 所有能够被3整除的数 放到一个清单里* @Test **public void** test1(){  List<Integer> list= Arrays.*asList*(1,2,3,4,5,6,7,8,9);  *// List rsList=new CopyOnWriteArrayList(); 适合多读少写* List rsList=Collections.*synchronizedList*(**new** ArrayList<>()); *//适合多写少读* Stream<CompletableFuture<Boolean>> completableFutureStream = list.stream().map(num ->  CompletableFuture.*supplyAsync*(() -> checkNum(num)).whenComplete((ifPass, ex) -> {  *//supplyAsync中 添加异步执行的线程处理任务 //whenComplete 添加线程执行完毕后的造操作 //* **if** (ifPass) {  rsList.add(num);  }  })  ); *// 流式处理 相当于把list<integer>里的转化为一个 Future数组 ,Future可以理解为一个不知道什么时候执行完的异步结果* CompletableFuture[] completableFutures = completableFutureStream.toArray(CompletableFuture[]::**new**);   *// 归集操作allOf代表此处阻塞 直到线程全部执行完 anyOf代表阻塞到只要有一个执行完就可。* CompletableFuture.*allOf*(completableFutures).join();   System.***out***.println(rsList); }  **private** Boolean checkNum(Integer num){  **try** {  Thread.*sleep*(500);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  **if**(num%3==0){  **return true**;  }**else** {  **return false** ;  }  } |

### 5.3 submitOrder中增加 方法

|  |
| --- |
| @PostMapping(**"submitOrder"**)  @LoginRequire  **public** String submitOrder(OrderInfo orderInfo,HttpServletRequest request){  String userId = (String) request.getAttribute(**"userId"**);   String tradeNo = request.getParameter(**"tradeNo"**);   **boolean** isEnableToken = **orderService**.verifyToken(userId, tradeNo);  **if**(!isEnableToken){  request.setAttribute(**"errMsg"**,**"页面已失效，请重新结算！"**);  **return "tradeFail"**;  }   *// 验库存 验价* orderInfo.setOrderStatus(OrderStatus.***UNPAID***);  orderInfo.setProcessStatus(ProcessStatus.***UNPAID***);  orderInfo.setCreateTime(**new** Date());  orderInfo.setExpireTime(DateUtils.*addMinutes*(**new** Date(),15));  orderInfo.sumTotalAmount();  orderInfo.setUserId(userId);   List<OrderDetail> orderDetailList = orderInfo.getOrderDetailList();  **for** (OrderDetail orderDetail : orderDetailList) {  SkuInfo skuInfo = **manageService**.getSkuInfo(orderDetail.getSkuId());  orderDetail.setImgUrl(skuInfo.getSkuDefaultImg());  orderDetail.setSkuName(skuInfo.getSkuName());   **if**(!orderDetail.getOrderPrice().equals(skuInfo.getPrice())){  request.setAttribute(**"errMsg"**,**"商品价格已发送变动请重新下单！"**);  **return "tradeFail"**;   }  }   List<OrderDetail> errList=Collections.*synchronizedList*(**new** ArrayList<>());  Stream<CompletableFuture<String>> completableFutureStream = orderDetailList.stream().map(orderDetail ->  CompletableFuture.*supplyAsync*(() -> checkSkuNum(orderDetail)).whenComplete((hasStock, ex) -> {  **if** (hasStock.equals(**"0"**)) {  errList.add(orderDetail);  }  })  );  CompletableFuture[] completableFutures = completableFutureStream.toArray(CompletableFuture[]::**new**);   CompletableFuture.*allOf*(completableFutures).join();   **if**(errList.size()>0){  StringBuffer errStingbuffer=**new** StringBuffer();  **for** (OrderDetail orderDetail : errList) {  errStingbuffer.append(**"商品："**+orderDetail.getSkuName()+**"库存暂时不足！"**);  }  request.setAttribute(**"errMsg"**,errStingbuffer.toString());  **return "tradeFail"**;  }   String orderId = **orderService**.saveOrder(orderInfo);   *// 此处应该删除购物车信息  // xxxx* **return "redirect://payment.gmall.com/index?orderId="**+orderId;  } |
| **public** String checkSkuNum(OrderDetail orderDetail){  String hasStock = HttpClientUtil.*doGet*(**"http://www.gware.com/hasStock?skuId="** + orderDetail.getSkuId() + **"&num="** + orderDetail.getSkuNum());  **return** hasStock; } |