# **77**\_代码实战:实现通过延时消息取消订单、退回优惠券、退回积分

**儒猿架构官网上线**,内有石杉老师架构课最新大纲,儒猿云平台详细介绍,敬请浏览

官网: www.ruyuan2020.com (建议 PC 端访问)

# 1.开篇

上节课为了解决大量未支付超时订单扫表性能问题,基于 RocketMQ 的延时消息 进行优化。在创建订单的时候发送延迟消息(30 分钟),消费者在接受到延迟消息的时候再判断是否支付,未支付则取消订单。本节课会带大家通过代码的方式实现上述功能。今天课程的内容包括以下几个部分:

- 消息队列的配置
- 消息队列的服务
- 消息队列的应用

## 2.消息队列的配置

首先需要在 pom 文件中引入 RocketMQ 的依赖。如图 1 所示,在 pom.xml 文件中加入对应的依赖项。



图 1 pom 依赖项

再到 application.properties 文件中加入和 RocketMQ 相关配置项。如图 2 所示,添加 name server 的服务器地址; topic 名字、producer group 名字、consumer

group 名字; 延迟的 topic 名字、延迟的 consumer group 名字; 还有延迟等级信息。



图 2 application.properties 配置信息

这些信息会在下面的消息队列服务中被使用到。

## 3.消息队列的服务

有了 RocketMQ 的基本配置以后再来看看对应的生产者和消费者的服务是如何编写的。

首先需要对传递的消息进行定义,由于需要适应两种场景:主动取消订单和超时取消订单,所以这里我们分别建立两个消息的实体类。

如图 3 所示,在 spring/dto 下面建立 OrderMessageDTO 类,里面会定义消息类型和消息内容。

```
package com.ruyuan.little.project.spring.dto;
little
                                  3
4
5
6
7
8
9
10
11
12
13
14
15

▼ Image: spring

                                           import com.ruyuan.little.project.common.enums.MessageTypeEnum;
   ▶ annotion▶ aspect
    ▶ 🛅 cache
                                            * @author <a href="mailto:little@163.com">little</a>
    ▶ ☐ controller
                                              Description:订单消息
    ▶ 🛅 dao
                                          public class OrderMessageDTO {
         Comment
         c 😉 Consumer
                                               /**
* 消息内容
         © % ConsumerCoupon
         Coupon
         © 'a Order
© 'a OrderDelayMessageDTO
                                               private String content;
                                  16
17
18
19
20
21
22
23
26
27
30
31
34
35
38
39
40 ••↑
      © • OrderMessageDTO
           ► Teacher
                                                * 订单消息推送类型 {@link MessageTypeEnum}
    ▶ 🖿 enums
                                               private MessageTypeEnum messageType;
      exception
    ▶ 🖿 handler
                                               public String getContent() { return content; }
      interceptor
      ims jms
                                               public void setContent(String content) { this.content = content; }
    ► mapper
► service
                                               public MessageTypeEnum getMessageType() { return messageType; }
    ▶ □ utils
    ▶ □ validator
                                               public void setMessageType(MessageTypeEnum messageType) { this.messageType = messageType; }
onsumerMapper.xml
                                               ouponMapper.xml
rderMapper.xml
acherMapper.xml
                                                             ", messageType=" + messageType +
'}';
                                  43
44
cation.properties
                                  45
                                               }
ack.properties
ack.xml
                                          }
```

图 3 OrderMessageDTO

然后是 OrderDelayMessageDTO 的定义,内容和 OrderMessageDTO 保持一致。

```
* Description: 1) 早消息
dao
 🛅 dto
                                  public class OrderDelayMessageDTO {
   © 🖫 Comment
   Consume
   © 🖫 ConsumerCoupon
                          14
15
                                       * 消息内容
   Coupon
   © ₃ Order
                                      private String content;
© № OrderDelayMessageDTO
  © Teacher
                          18
19
                                      /**
* 订单消息推送类型 {@link DelayMessageTypeEnum}
 enums
                          20
 event
                                      private DelayMessageTypeEnum delayMessageType;
 exception
                          22
23
26
27
nandler 🖿
                                      public String getContent() { return content; }
 intercepto
ims ims
                                      public void setContent(String content) { this.content = content; }
 mapper mapper
                          30
31
34
35
service
                                      public DelayMessageTypeEnum getDelayMessageType() { return delayMessageType; }
 utils
ualidator 🔤
                                      public void setDelayMessageType(DelayMessageTypeEnum delayMessageType) { this.delayMessageType =
                          35
38
39
40 •1
41
42
43
44
                                     rMapper.xml
lapper.xml
oper.xml
                                                                + content + '\'' +
lapper.xml
                                                      delayMessageType=" + delayMessageType +
properties
                          45
46
perties
```

图 4 OrderDelayMessageDTO

如图 5 所示,在 spring/jms 目录下面创建 MqProducerConfiguration 类文件,它主要对消息生产者(Producer)的初始化和销毁做了定义。Init 方法主要为消息生产者创建生产者租,同时制定 name server 地址,并且启动这个生产者。而destroy 方法则是用来关闭生产者服务。

```
public MqProducerConfiguration(String producerGroupName, String nameServerAddr) {
                                     32
                                                      this.producerGroupName = producerGroupName;
this.nameServerAddr = nameServerAddr;
 spring
   annotion
                                     34
35
  aspect
                                             cache
                                     36
                                                     bean初始化执行方法
 constants
   controller
                                     38
                                                     @throws Exception
   dao
                                     39
40
   dto
                                                  public void init() throws Exception {
   enums
                                     41
42
                                                       LOGGER.info("开始启动消息生产者服务...
   event
   exception
                                                      //创建一个消息生产者,并设置一个消息生产者组
producer = new DefaultMQProducer(producerGroupName);
//指定 NameServer 地址
                                     43
44
 ▶ lim handler
   interceptor
                                     45
 ▼ 🛅 jms
                                                      producer、setNamesrvAddr(nameServerAddr);
//初始化 SpringProducer,整个应用生命周期内只需要初始化一次
                                     46
      © % MgConsumerConfiguration
      © 1 MqDelayConsumerConfiguration
                                     48
                                                       producer.start();
49
         OrderDelayMessageListener
                                     50
51
                                                      LOGGER.info("消息生产者服务启动成功.");
      © % OrderMessageListener
 ▶ mapper
                                     52
53
 ▶ Service
   utils 🖿
                                     54
55
                                                   * bean消费时执行方法
 validator
                                     56
                                                  public void destroy() {
                                                      LOGGER.info("开始关闭消息生产者服务...");
umerMapper.xml
                                     58
onMapper.xml
                                     59
                                                      if (producer != null) {
Mapper.xml
                                     60
                                                           producer.shutdown():
nerMapper.xml
on.properties
                                     62
perties
                                                      LOGGER.info("消息生产者服务已关闭.");
properties
                                     64
xml
```

图 5 MqProducerConfiguration

定义完了生产者再来看看消费者的定义,如图 6 所示,MqConsumerConfiguration 类中对消息消费者做了定义,init 方法中定义了消费者组、name server 地址、并且设置消费消息的方式(从队列头部开始消费)、订阅的 topic 信息。最重要的是注册消息的监听器,这里将监听器指定为 orderMessageListener,后面会介绍这个监听器的职能,最后就是启动消费者服务。与 init 方法对应的是 destroy 方法用

来停止消费者服务。

```
pring
                                          public void init() throws Exception {
annotion
                                              LOGGER.info("开始启动消息消费者服务...");
                              62
                              63
cache
                              64
                                               //创建一个消息消费者,并设置一个消息消费者组
constants
                              65
                                              consumer = new DefaultMQPushConsumer(consumerGroupName);
controlle
                              66
                                              //指定 NameServer 地址
dao dao
                              67
                                              consumer.setNamesrvAddr(nameServerAddr);
dto
                              68
                                               //设置Consumer第一次启动是从队列头部开始消费还是队列尾部开始消费
enums
                                              {\tt consumer.setConsumeFromWhere}. {\tt ConsumeFromWhere}. {\tt ConsumeFromWhere}. {\tt FROM\_FIRST\_OFFSET});
                              69
                              70
event
                              71
72
73
exception
                                              //订阅指定 Topic 下的所有消息
handler handler
                                              consumer.subscribe(topicName, subExpression: "*");
interceptor
ims ims
© 1 MqConsumerConfiguration
                               75
76
                                              consumer.registerMessageListener(orderMessageListener);
     MqDelayConsumerConfiguration
 © 1/2 MqProducerConfiguration
                                              // 消费者对象在使用之前必须要调用 start 初始化
  OrderDelayMessageListener
                               78
79
  OrderMessageListener
mapper mapper
                              80
                                              LOGGER.info("消息消费者服务启动成功.");
service
                              81
utils
                              82
validato
                              83
                              84
                                             bean消费方法
                              85
/lapper.xml
                                          public void destrov() {
                              86
                                              LOGGER.info("开始关闭消息消费者服务...");
pper.xml
                              87
er.xml
                              88
                                              if (consumer != null) {
                                                   consumer.shutdown():
                              89
operties
                              90
                                              LOGGER.info("消息消费者服务已关闭.");
                              91
                              92
                              93
g.xml
```

#### 图 6 MgConsumerConfiguration

这里需要说明一下由于有两个取消订单场景:主动取消订单和超时取消订单。上面介绍的消费者 MqConsumerConfiguration 属于主动取消订单的功能。我们将为超时取消订单建立单独的消息消费者和监听器。

说完了主动取消订单的消费者,再来看看对应的监听器。如图7所示,

OrderMessageListener 实现了 MessageListenerOrderly 并且 override 了 consumeMessage 方法。方法中对消息内容进行解析,在获取 orderMessage 之 后判断消息类型是否是取消订单,如果是的话对积分也存在扣减的情况,那么就退还积分:如果同时也适用了优惠券,对优惠券也实现回退的操作。

```
little

project
                                                                                           public class OrderMessageListener implements MessageListenerOrderly {
    ▼ Ispring
► Ispring
► Ispring
► Ispring
                                                                            28
29
30
31
32
33
34
35
36
37
38
39
40
41
                                                                                                         * 日志管理组件
             ache
                                                                                                     private static final Logger LOGGER = LoggerFactory.getLogger(OrderMessageListener.class);
             constants
                                                                                                        * 消费者service组件
             controlle
            dao
dto
enums
event
exception
handler
                                                                                                    @Autowired
private ConsumerService consumerService;
                                                                                                        * 优惠券service组件
             interceptor
          ▼ ■ jms
□ № MqConsumerConfiguration
□ № MqDelayConsumerConfiguration
□ № MqProducerConfiguration
                                                                                                     private CouponService couponService;
                                                                            42

43 e1

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

60

61

62

63

64
                                                                                                    @Uverride
public ConsumeOrderlyStatus consumeMessage(List<MessageExt> list, ConsumeOrderlyContext consumeOrderlyContext) {
    for (MessageExt ext : list) {
        String content = new String(ext.getBody(), StandardCharsets.UTF_8);
        LOGEFR.info("received order message:{}", content);
        // 订单消息
        OrderMessageDTO orderMessage = JSON.parseObject(content, OrderMessageDTO.class);
        // 订单均息
                   C 'a OrderDelayMe
 © № OrderMessageListener
         ► mapper
► service
► utils
► validator
                                                                                                                      기 기부위함
Order order = JSON.parseObject(orderMessage.getContent(), Order.class);
MessageTypeEnum messageType = orderMessage.getMessageType();
per
:onsumerMapper.xml
                                                                                                                      if (messageType.equals(MessageTypeEnum.WX_CANCEL_ORDER)) {
    // 订单取消 积分还原
    if (order.getDeductCredits() > 0) {
        LOGGER.info''订单取消, 积分还原");
        consumerService.orderCancelInformUnDeductCredits(order);
    }
 ouponMapper.xml
 orderMapper.xml
eacherMapper.xml
lication.properties
lication.properti
:.properties
>ack.properties
>ack.xml
| atis-config.xml
| ng-jdbc.xml
                                                                                                                                // 订单取消 优惠券还原
if (!StringUtils.isEmpty(order.getConsumerCouponId())) {
    LOGGER.info("订单取消, 优惠券还原");
    couponService.orderCancelUpdateCouponStatus(order);
ng-web.xml
                                                                                                              return ConsumeOrderlyStatus.SUCCESS;
```

图 7 OrderMessageListener

说完了主动取消订单的场景,再来看看超时订单的场景,它与前者的实现方式非常 相似。

如图 8 所示,MqDelayConsumerConfiguration 中也定义了 init 和 destroy 方法。 Init 中定义延迟消费者组、name server 地址、延迟消费的消费方式、订阅的 topic 信息。也注册了消息的监听器,这里将监听器指定为 orderDelayMessageListener。与 init 方法对应的是 destroy 方法用来停止消费者服

务。

```
project
🛮 🖿 spring
                                                       * bean加载是init的方法
  ▶ mannotion
                                        58
59
  ▶ ■ aspect
                                                         @throws Exception
  ▶ □ cache
                                        60
61
62
63
64
65
66
67
  constants
                                                     public void init() throws Exception {
  ▶ □ controller
                                                          LOGGER.info("开始启动延时消息消费者服务...");
  ▶ lim dao
  ▶ 🛅 dto
                                                           //创建一个消息消费者,并设置一个消息消费者组
  ▶ ■ enums
                                                          delayConsumer = new DefaultMQPushConsumer(delayConsumerGroup);
  ▶ ■ event
                                                          //指定 NameServer 地址
  ▶ m exception
                                                          delayConsumer.setNamesrvAddr(nameServerAddr);
                                                          //设置Consumer第一次启动是从队列头部开始消费还是队列尾部开始消费
delayConsumer.setConsumeFromWhere(ConsumeFromWhere.CONSUME_FROM_FIRST_OFFSET);
                                        68
69
  ▶ interceptor
  ▼ 🛅 jms
                                        70
71
72
73
74
75
76
77
78
80
81
82
83
84
85
       © 1/2 MgConsumerConfiguration
                                                          //订阅指定 Topic 下的所有消息
© 1 MqDelayConsumerConfiguration
                                                          delayConsumer.subscribe(delayTopicName, subExpression: "*");
       © 1/2 MgProducerConfiguration
       © № OrderDelayMessageListener
                                                           //注册消息监听器
       © 🖫 OrderMessageListener
                                                          delayConsumer.registerMessageListener(orderDelayMessageListener);
  ▶ ■ service
                                                              消费者对象在使用之前必须要调用 start 初始化
 ► utils
► validator
                                                          delayConsumer.start();
                                                          LOGGER.info("延时消息消费者服务启动成功.");
sumerMapper.xml
on Mapper.xml
rMapper.xml
                                                      * bean消费方法
herMapper.xml
ion.properties
                                                     public void destroy() {
    LOGGER.info("开始关闭延时消息消费者服务...");
                                        86
87
88
89
90
91
92
93
perties
properties
                                                          if (delayConsumer != null) {
    delayConsumer.shutdown();
.xml
-config.xml
dbc.xml
                                                          .
LOGGER.info("延时消息消费者服务已关闭.");
ms.xml
web.xml
                                                }
                                        95
```

图 8 MqDelayConsumerConfiguration

最后来到 OrderDelayMessageListener 的部分,如图 9 所示,他是延迟消息消息费者的监听器,实现了 MessageListenerOrderly 并且 override 了 consumeMessage 方法。方法中对消息内容进行解析,在获取 orderMessage 之后判断消息类型是否是支付超时订单,如果是的话将订单取消并且更新订单状态。后面处理积分和优惠券的方式与主动取消订单如出一辙。

```
private CouponService couponService;
@Override
  project

| Spring | Damotion | Da
                                                                                                                                                                  Т
                                                                                                                                                                                       public ConsumeOrderlyStatus consumeMessage(List<MessageExt> list, ConsumeOrderlyContext consumeOrderlyContext) {
                                                                                                                                                                                                                       .Onsumeruer ystatus communication (MessageExt ext : list) {
    LOGGER.info("延时消息队列消息总数{}",list.size());
    String content = new String(ext.getBody(), StandardCharsets.UTF_8);
    LOGGER.info("received order delay message:{}", content);
                                                                                                                                        52
53
54
55
56
57
58
59
60
61
62
63
64
                                                                                                                                                                                                                        OrderDelayMessageDTO orderMessage = JSON.parseObject(content, OrderDelayMessageDTO.class);
             enums
             event exception
                                                                                                                                                                                                                        Order order = JSON.parseObject(orderMessage.getContent(), Order.class);
DelayMessageTypeEnum delayMessageType = orderMessage.getDelayMessageType();
             nandler 🚞
             interceptor
                                                                                                                                                                                                                       try {
    if (DelayMessageTypeEnum.PAY_DELAY.equals(delayMessageType)) {
        LOGGER.info("订单支付超时,现在开始取消订单,订单信息{}",order);
        // 订单取消
        order = orderMapper.findById(order.getId());
        if (!OrderStatusEnum.MAITING_FOR_PAY.getStatus().equals(order.getStatus())){
            throw new BusinessException(EducationBusinessErrorCodeEnum.ORDER_CANT_CANCEL.getMsg());
    }
        ▼ 🛅 ims
                         C & MgConsumerConfiguration
© in MqDelayConsumerConfiguration
© in MqProducerConfiguration
© in OrderDelayMessageListener
© in OrderMessageListener
                                                                                                                                          68
69
70
71
72
73
74
75
76
77
78
80
81
82
83
84
85
86
87
                                                                                                                                                                                                                                                          order.setStatus(OrderStatusEnum.CANCELED.getStatus());
                                                                                                                                                                                                                                                         // 更新订单取消封间和订单状态
int updateRow = orderMapper.cancelOrder(order.getId(), OrderStatusEnum.CANCELED.getStatus());
if (updateRow > 0) {
    // 订单取消 积分还原
    if (order.getDeductCredits() > 0) {
        LOGGER.info("订单取消, 积分还原");
        consumerService.orderCancelInformUnDeductCredits(order);
    }
:umerMapper.xml
:onMapper.xml
 rMapper.xml
 nerMapper.xml
 ion.properties
                                                                                                                                                                                                                                                                          // 汀辛取消 优惠券还原
if (!StringUtils.isEmpty(order.getConsumerCouponId())) {
    LOGGER.info("汀辛取消, 忧惠券还原");
    couponService.orderCancelUpdateCouponStatus(order);
                                                                                                                                                                                                                                                                                       订单取消 优惠券还原
  properties.
   -config.xml
 dbc.xml
                                                                                                                                                                                                                        }catch (BusinessException e){
                                                                                                                                                                                                                                         // 订单已经取消,无需再取消
LOGGER.info("订单已经取消,无需再取消");
                                                                                                                                                                                                        return ConsumeOrderlyStatus.SUCCESS;
```

图 9 OrderDelayMessageListener

定义完消息体、消息生产者、消息消费者以及监听器以后,还需要对相关的 Bean 实例化。如图 10 所示,在 resources 目录下面建立 spring-jms.xml 文件对相关 Bean 进行实例化。我们将红框的部分从上往下介绍。

- 对订单消息 OrderMessageListener 实例化,它是主动取消订单的监听器。
- 对 MqConsumerConfiguration 订单消息消费者进行初始化配置,指明初始化方法 "init"和销毁方法"destroy"。以及初始化所需要的参数 name server address、 consumer group name 和 topic name。最重要的是指定监听器,也就是上面实例 化的 OrderMessageListener。
- 对 OrderDelayMessageListener 进行实例化,这个是延迟消息的监听器。
- 对 MqDelayConsumerConfiguration 延迟订单消息消费者进行初始化配置,指明初始化方法 "init"和销毁方法"destroy"。 以及初始化所需要的参数 name server address、consumer group name 和 topic name。指定监听器是OrderDelayMessageListener。
- 对 MqProducerConfiguration 订单消息生产者配置,也需要对"init"和"destroy"进行指定。同时制定 name server addres 和 producer group name 的信息。
   另外,本文件中读取的参数信息例如: "\${rocketmq.namesrv.address}"这样的信息都是来自于 application.properties 文件中。

```
▼ ■ spring

► ■ annotion

► ■ aspect

► ■ cache
                                                                                                        <beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemalocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
                                                                                    ▶ □ constants
                                                                                                                   <!-- 订单消息Listener-->
<br/>
<br/>
dean id="orderMessageListener" class="com.ruyuan.little.project.spring.jms.OrderMessageListener" />
                                     controller
                                b dao
b dto
b enums
                                                                                                                   event
                                      exception
                                     la handler
                                     interceptor

intercept
inte
                                                                                                                    <bean id="orderDelayMessageListener" class="com.ruyuan.little.project.spring.jms.OrderDelayMessageListener" />
   resources
       mapper
               a consumerMapper.xml
                                                                                                                                 订单延时消息消费者配置-
                                                                                                                   <!-- : 1 单处时消息消费者配置—->
<bean id="mqpDelayConsumerConfiguration" class="com.ruyuan.little.project.spring.jms.MqDelayConsumerConfiguration"
    init-method="init" destroy-method="destroy">
    <constructor-arg name="nameServerAddr" value="${rocketmq.namesrv.address}"/>
    <constructor-arg name="delayConsumerGroup" value="${rocketmq.order.delay.consumer.group}"/>
    <constructor-arg name="delayTopicName" value="${rocketmq.order.delay.topic}" />
    <constructor-arg name="orderDelayMessageListener" ref="orderDelayMessageListener" />

         consumer Mapper.xml
couponMapper.xml
teacherMapper.xml
          jdbc.properties
          logback.properties
          logback.xml
        mybatis-config.xml
spring-jdbc.xml
spring-jms.xml
                                                                                                                   spring-web.xml
■ WEB-INF
```

图 10 spring-jms.xml

# 4.消息队列的应用

消息队列应用的场景有主动取消和延迟取消,本节的重点在超时取消。下面先将主动取消的入口告诉大家,然后在转入超时取消的部分。

在 spring/controller 下面的 OrderController 中,加入 cancelOrder 方法,该方法

调用 OrderService 中的 cancelOrder 对订单进行取消操作。

```
if (total != 0) {
      ▼ 🖿 controller
                                                                        条数不为空查询订单列表
            © 1/2 ConsumerController
                                                                     tableData.setRows(orderService.getPage(order));
            HealthController
                                              62
            © № JdbcDemoController
                                                                commonResponse.setData(tableData);
         C & OrderController
                                              64
                                                                return commonResponse;
              TeacherController
                                              65
66
67
                                                           }
      ▶ 🖿 dao
        🛅 dto
      ▶ menums
                                                            * 根据订单id查询订单信息
                                              68
        event
                                              69
70
71
72
73
74
           C LogonEvent
                                                            * @param id 订单id
           © 1 LogonEventPublisher
                                                              @return 结果
           © № LogonListener
       exception
                                                           @GetMapping("/{id}")
public CommonResponse findById(@PathVariable Integer id)
      ▶ lim handler
        interceptor
                                              75
76
77
78
79
80
                                                                return CommonResponse.success(orderService.findById(id));
      ▶ Imsims
                                                           }
       ▶ mapper mapper
      ▶ ■ service
                                                            * 撤销订单
      validator
                                              81
82
                                                            * @param orderId 订单id
napper
                                                            * @return 结果
consumerMapper.xml
                                              83
couponMapper.xml
                                                           @RequestMapping("/cancel")
                                              84
orderMapper.xml
                                              85 @
86
                                                           public CommonResponse cancelOrder(Integer orderId) {
teacherMapper.xml
                                                                return CommonResponse.success(orderService.cancelOrder(orderId)):
pplication.properties
                                              87
dbc.properties
                                              88
gback.properties
                                              89
aback.xml
                                                      }
nybatis-config.xml
                                              90
                                             91
prina-idbc.xml
```

如图 11 OrderController 的 cancelOrder 方法

如图 12 所示,在 OrderServiceImpl 中的 cancelOrder 方法主要是更新订单状态,同时发送取消订单的消息。后面就是 OrderMessageListener 监听到该消息进行后续的取消订单处理。

```
CouponServiceImpl
                                 98 📭
                                             public int cancelOrder(Integer orderId) {
    OrderEvenManagerImpl
                                                 Order order = orderMapper.findById(orderId);
if (!OrderStatusEnum.WAITING_FOR_PAY.getStatus().equals(order.getStatus())){
    throw new BusinessException(EducationBusinessErrorCodeEnum.ORDER_CANT_CANCEL.getMsg());
© ¹a OrderServiceImpl

© ¹a TeacherServiceImpl
                                100
101
102

    ConsumerService

    □ CouponService

                                 103
104
105
                                                 order.setStatus(OrderStatusEnum.CANCELED.getStatus());
 OrderEventManage

    OrderService
    TeacherService

utils
validator
                                                      orderEventManager.cancelOrderEvent(order);
                                109
110
                                                                                                                发送取消订单的消息
                                                 return updateRow;
apper.xml
```

图 12 OrderServiceImpl 中的 cancelOrder 方法

说完了主动 取消订单再来看看超时取消订单的处理过程。如图 13 所示,在 OrderController 中有 add 方法用来创建订单。这里也是直接调用 OrderService 中

的 createOrder 方法, 传入 Order 对象作为输入参数。



图 13 OrderController 中的 add 创建订单

如图 **14** 所示,在 OrderServiceImpl 中的 createOrder 方法体中执行的内容和上周的基本一样,只是加入了一条发送消息的调用。这里调用了 OrderEventManager 中的 createOrderEvent 方法,传入的就是 Order 实体。

```
aspect
cache
                                                                                                      @Override
public Order findById(int id) {
    Order order = orderMapper.findById(id);
    order.setStatusName(OrderStatusEnum.getDescByStatus(order.getStatus()));
    order.setCourseName(CourseTypeEnum.getNameByCode(order.getCourse()));
    constants
 ▼ 🖿 controller
          © 'a ConsumerContro
© 'a HealthController
                                                                               68
69
70
71
72
73
•1
75
76
77
78
80
81
82
83
84
85
86
87
88
89
           😊 🆫 JdbcDemoControlle
© 🔓 OrderController
          © № TeacherControlle
► 🖿 dao
► 🖿 dto
                                                                                                       @Override
                                                                                                       goverride

public void createOrder(Order order) {

// 根据订单金额获取积分

BigDecimal payAmount = order.getPayAmount();

if (payAmount != null) {

order.setReceiveCredits(payAmount.intValue());
 ▶ menums
 ▶ ■ event
    exception
handler
interceptor
 order.setStatus(OrderStatusEnum.WAITING_FOR_PAY.getStatus());
order.setStatusName(OrderStatusEnum.getDescByStatus(order.getStatus()));
order.setCourseName(CourseTypeEnum.getNameByCode(order.getCourse()));
          © % MaConsumerConfiguration
          © ¹a MqDelayConsumerConfiguration
© ¹a MqProducerConfiguration
© ¹a OrderDelayMessageListener
© ¹a OrderMessageListener
                                                                                                                 // ドロギ
orderMapper.add(order);
// 使用优惠券
couponService.usedCoupon(order);
    mapper
                                                                                                                  consumerService.orderCreateInformDeductCredits(order);
               CouponSen
                                                                                                                 orderEventManager.createOrderEvent(order);
               © 1 OrderEvent
© 1 OrderService
```

#### 图 14 OrderServiceImpl 中的 createOrder

如图 15 所示,在 spring/service/impl 下面创建 OrderEventManagerImpl 类并且 定义 createOrderEvent 方法,该方法会调用 sendOrderMessage 的私有方法。该 方法定义了要发送 OrderMessageDTO,设置好 topic 和消息类型,通过 MQProducerConfiguration 生成的消息生产者发送这个延迟消息。发送延迟消息的 内容和主动取消订单都是一样的,唯一不同就是会设置延迟消息的等级

# OrderDelayLevel.



图 15 OrderEventManagerImpl 中的 createOrderEvent 方法

说明一下这里的 orderDelayLevel 是从 application.properties 中读取的,其值为 9. 这里的 9 是时间对应延迟时间的位移量,如果设置 8 就对应 4m,也就是 4 分钟。 我们将每个延迟时间对应的 level 值的对应关系放到如下表格中,给大家参考,在测试的时候可以调整具体时间。

时间	1s	5s	10s	30s	1m	2m	3m	4m	5m	6m	7m	8m
Level	1	2	3	4	5	6	7	8	9	10	11	12
9m	10m	20m	30m	1h	2h							
13	14		16	17	18							

15

# 5.总结

本节课介绍了主动取消订单和延迟取消订单的代码实施,其中包括消息队列的配置: pom 文件和 appliaction.properties 文件;消息队列的服务:消息生产者、消息消费者、消息监听器;消息队列的应用: OrderServiceImpl 和 OrderEventManagerImpl。下节课会带大家进行取消订单业务代码的测试。这里将代码给大家,下期见,拜拜。