第15章 秒杀

学习目标

- 防止秒杀重复排队

```
重复排队：一个人抢购商品，如果没有支付，不允许重复排队抢购
```

- 并发超卖问题解决

```
1个商品卖给多个人：1商品多订单
```

- 秒杀订单支付

```
秒杀支付：支付流程需要调整
```

- 超时支付订单库存回滚

```
1.RabbitMQ延时队列
2.利用延时队列实现支付订单的监听，根据订单支付状况进行订单数据库回滚
```

1 售罄处理

商品在秒杀过程中，会出现最后一个商品抢购现象，最后一个商品抢购商品的剩余库存数量会为0，此时需要将数据同步到数据库中，并且清理掉Redis中的缓存数据。

```java
@Autowired
private SeckillGoodsMapper seckillGoodsMapper;

/****
 * 下单操作
 */
@RabbitHandler
public void readOrderMessage(String message){
    //获取抢单信息   username,id,time
    SeckillStatus seckillStatus = JSON.parseObject(message,SeckillStatus.class);

    //有可能存在未支付的订单   key1=SeckillOrder    key2=username
    Object order = redisTemplate.boundHashOps( key: "SeckillOrder").get(seckillStatus.getUsername());

    if(order!=null){
        //更新抢单状态
        System.out.println("--------存在未支付订单，不允许排队抢单----");
        return;
    }
    //商品是否有库存(存在缺陷)->是否超卖
    SeckillGoods seckillGoods = (SeckillGoods) redisTemplate.boundHashOps( key: "SeckillGoods_"+seckillStatus.getTime()).get(seckillStatus

    if(seckillGoods!=null && seckillGoods.getStockCount()>0){
        //创建订单对象
        SeckillOrder seckillOrder = new SeckillOrder();
        seckillOrder.setId("No"+idWorker.nextId());
        seckillOrder.setSeckillId(seckillStatus.getGoodsId());
        seckillOrder.setMoney(seckillGoods.getCostPrice());
        seckillOrder.setUserId(seckillStatus.getUsername());
        seckillOrder.setCreateTime(seckillStatus.getCreateTime());
        seckillOrder.setStatus("0");     //未支付
        //将数据存入到Redis
        redisTemplate.boundHashOps( key: "SeckillOrder").put(seckillStatus.getUsername(),seckillOrder);

        //库存递减
        seckillGoods.setStockCount(seckillGoods.getStockCount()-1);
        if(seckillGoods.getStockCount()<=0){
            //数据同步到MySQL中
            seckillGoodsMapper.updateByPrimaryKeySelective(seckillGoods);

            // 同时删除Redis中的数据
            redisTemplate.boundHashOps( key: "SeckillGoods_"+seckillStatus.getTime()).delete(seckillGoods.getId());
        }else{
            //数据同步到Redis
            redisTemplate.boundHashOps( key: "SeckillGoods_"+seckillStatus.getTime()).put(seckillGoods.getId(),seckillGoods);
        }
        //更新抢单状态
        seckillStatus.setMoney(Float.valueOf(seckillGoods.getCostPrice()));
        seckillStatus.setOrderId(seckillOrder.getId());
        seckillStatus.setStatus(2); //抢单成功，  等待支付
        stringRedisTemplate.boundValueOps( key: "SeckillStatus_"+seckillStatus.getUsername()).set(JSON.toJSONString(seckillStatus));
    }
}
```

上图代码如下：

```java
@Autowired
private SeckillGoodsMapper seckillGoodsMapper;

/****
 * 下单操作
 */
@RabbitHandler
public void readOrderMessage(String message){
    //....略

    if(seckillGoods!=null && seckillGoods.getStockCount()>0){
        //.....略

        //库存递减
        seckillGoods.setStockCount(seckillGoods.getStockCount()-1);
        if(seckillGoods.getStockCount()<=0){
            //数据同步到MySQL中
```

```
            seckillGoodsMapper.updateByPrimaryKeySelective(seckillGoods);

            // 同时删除Redis中的数据

 redisTemplate.boundHashOps("SeckillGoods_"+seckillStatus.getTime()).delete(seckillGoods.getId());
        }else{
            //数据同步到Redis

 redisTemplate.boundHashOps("SeckillGoods_"+seckillStatus.getTime()).put(seckillGoods.getId(),seckillGoods);
        }
        //...略
    }
}
```

2 防止秒杀重复排队

用户每次抢单的时候，一旦排队，我们设置一个自增值，让该值的初始值为1，每次进入抢单的时候，对它进行递增，如果值>1，则表明已经排队,不允许重复排队,如果重复排队，则对外抛出异常，并抛出异常信息100表示已经正在排队。

1.1 后台排队记录

修改`seckillOrderServiceImpl`的add方法，新增递增值判断是否排队中，代码如下：

```java
/***
 * 秒杀抢单
 * @param username : 抢单用户
 * @param id : 商品ID
 * @param time : 时间 20200106
 */
@Override
public Boolean add(String username, String id, String time) {
    /***
     * boundValueOps("UserQueueCount_"+username).increment(N)
     * key的值递增N
     * 如果该key的值不存在，则表示初始值为9
     */
    //在Redis中定义一个变量 UserQueueCount_username,用户每次进入该方法，都让它的值递增1
    //第一次请求抢单UserQueueCount_username=1
    //返回值：递增之后的结果
    Long increment = redisTemplate.boundValueOps( key: "UserQueueCount_" + username).increment( delta: 1);

    if(increment>1){
        //正在排队  100表示重复抢单
        throw new RuntimeException("100");
        //return true;
    }

    //封装抢单信息
    SeckillStatus seckillStatus = new SeckillStatus();
    seckillStatus.setUsername(username);
    seckillStatus.setCreateTime(new Date());
    seckillStatus.setStatus(1);      //排队中
    seckillStatus.setGoodsId(id);    //商品ID
    seckillStatus.setTime(time);     //商品所在的key的时间后缀

    //状态信息
    String statusJson = JSON.toJSONString(seckillStatus);

    //队列削峰
    //将抢单信息发送到RabbitMQ    交换机、队列、队列与交换机绑定
    rabbitTemplate.convertAndSend( exchange: "seckillOrderExchange",  routingKey: "seckillOrderQueue",statusJson);

    //将抢单信息存入到Redis  key:value
    //                       key=SeckillStatus_username
    //                       value=seckillStatus
    String key = "SeckillStatus_"+username;
    stringRedisTemplate.boundValueOps(key).set(statusJson);
    return true;
}
```

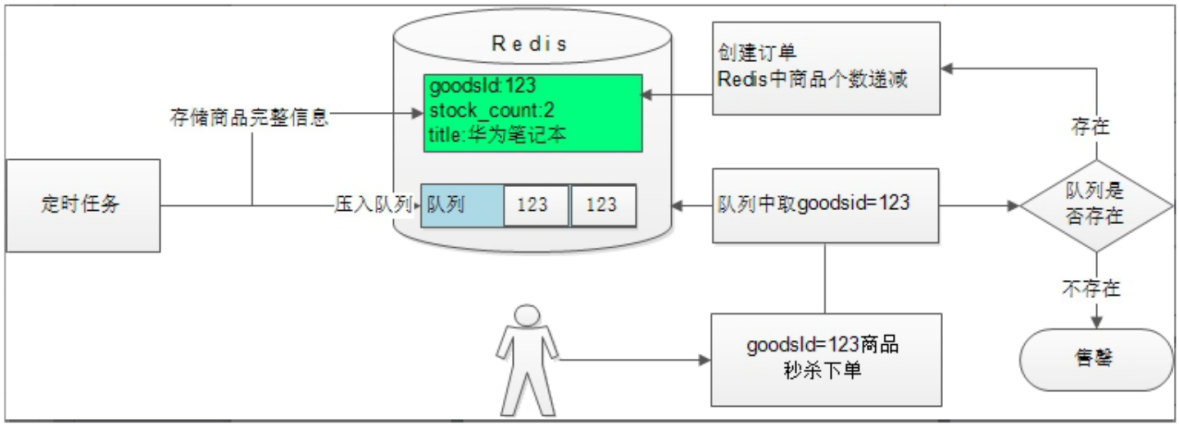
上图代码如下：

```java
//返回值：递增之后的结果
Long increment = redisTemplate.boundValueOps("UserQueueCount_" +
username).increment(1);

if(increment>1){
    //正在排队  100表示重复抢单
    throw new RuntimeException("100");
    //return true;
}
```

3 并发超卖问题解决

超卖问题，这里是指多人抢购同一商品的时候，多人同时判断是否有库存，如果只剩一个，则都会判断有库存，此时会导致超卖现象产生，也就是一个商品下了多个订单的现象。

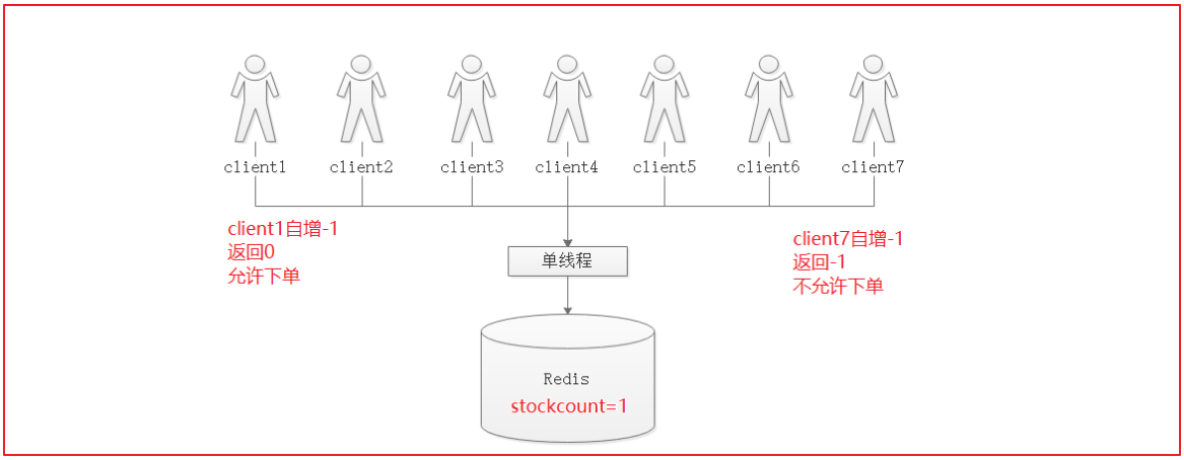
3.1 思路分析



解决超卖问题，可以利用Redis队列实现，给每件商品创建一个独立的商品个数队列，例如：A商品有2个，A商品的ID为1001，则可以创建一个队列,key=SeckillGoodsCountList_1001,往该队列中塞2次该商品ID。

每次给用户下单的时候，先从队列中取数据，如果能取到数据，则表明有库存，如果取不到，则表明没有库存，这样就可以防止超卖问题产生了。

在我们队Redis进行操作的时候，很多时候，都是先将数据查询出来，在内存中修改，然后存入到Redis，在并发场景，会出现数据错乱问题，为了控制数量准确，我们单独将商品数量整一个自增键，自增键是线程安全的，所以不担心并发场景的问题。



3.2 代码实现

每次将商品压入Redis缓存的时候，另外多创建一个商品的队列。

修改SeckillGoodsPushTask,添加一个pushIds方法，用于将指定商品ID放入到指定的数字中，代码如下：

```java
/***
 * 将商品ID存入到数组中
 * @param len:长度
 * @param id :值
 * @return
 */
public String[] pushIds(int len,String id){
    String[] ids = new String[len];
    for (int i = 0; i <ids.length ; i++) {
        ids[i]=id;
    }
    return ids;
}
```

修改SeckillGoodsPushTask的loadGoodsPushRedis方法，添加队列操作，代码如下：

```java
@Scheduled(cron = "0/30 * * * * ?")
public void loadGoodsPushRedis(){
    //获取时间段集合
    List<Date> dateMenus = DateUtil.getDateMenus();
    //循环时间段
    for (Date startTime : dateMenus) {
        //...略
        //将秒杀商品数据存入到Redis缓存
        for (SeckillGoods seckillGood : seckillGoods) {
            redisTemplate.boundHashOps( key: "SeckillGoods_"+extName).put(seckillGood.getId(),seckillGood);
            //商品数据队列存储,防止高并发超卖
            Long[] ids = pushIds(seckillGood.getStockCount(), seckillGood.getId());
            redisTemplate.boundListOps( key: "SeckillGoodsCountList_"+seckillGood.getId()).leftPushAll(ids);
            //自增计数器
            redisTemplate.boundHashOps( key: "SeckillGoodsCount").increment(seckillGood.getId(),seckillGood.getStockCount());
        }
    }
}
```

上图代码如下：

```java
//商品数据队列存储,防止高并发超卖
Long[] ids = pushIds(seckillGood.getStockCount(), seckillGood.getId());
redisTemplate.boundListOps("SeckillGoodsCountList_"+seckillGood.getId()).leftPushAll(ids);
//自增计数器
redisTemplate.boundHashOps("SeckillGoodsCount").increment(seckillGood.getId(),seckillGood.getStockCount());
```

3.3 超卖控制

修改RabbitMQ的消费者`com.changgou.seckill.mq.SeckillOrderConsumer`下单方法，分别修改数量控制，以及售罄后用户抢单排队信息的清理，修改代码如下图：

```java
@Component
@RabbitListener(queues = {"seckillOrderQueue"})
public class SeckillOrderConsumer {

    @Autowired
    private RedisTemplate redisTemplate;

    @Autowired
    private StringRedisTemplate stringRedisTemplate;

    @Autowired
    private IdWorker idWorker;

    @Autowired
    private SeckillGoodsMapper seckillGoodsMapper;


    /****
     * 下单操作
     */
    @RabbitHandler
    public void readOrderMessage(String message){
        //获取抢单信息   username,id,time
        SeckillStatus seckillStatus = JSON.parseObject(message,SeckillStatus.class);

        //用户抢单的时候已经实现了重复抢单的排除功能
        //有可能存在未支付的订单  key1=SeckillOrder    key2=username
        //Object order = redisTemplate.boundHashOps("SeckillOrder").get(seckillStatus.getUsername());

        //if(order!=null){
            //更新抢单状态
            //System.out.println("-------存在未支付订单，不允许排队抢单----");
            //return;
        //}

        //从队列中获取该商品的队列值，判断是否有库存
        Object goodsId = redisTemplate.boundListOps( key: "SeckillGoodsCountList_" + seckillStatus.getGoodsId()).rightPop();

        //如果此时商品的队列值为null，则表明该商品售罄
        if(goodsId==null){
            //清理用户排队抢单标识
            clearSeckillStatus(seckillStatus);
            return;
        }
        //商品是否有库存(存在缺陷)->是否超卖
        SeckillGoods seckillGoods = (SeckillGoods) redisTemplate.boundHashOps( key: "SeckillGoods_"+seckillStatus.getTime()).get(seckillSta

        if(seckillGoods!=null && seckillGoods.getStockCount()>0){
            //创建订单对象
            SeckillOrder seckillOrder = new SeckillOrder();
            seckillOrder.setId("No"+idWorker.nextId());
            seckillOrder.setSeckillId(seckillStatus.getGoodsId());
            seckillOrder.setMoney(seckillGoods.getCostPrice());
            seckillOrder.setUserId(seckillStatus.getUsername());
            seckillOrder.setCreateTime(seckillStatus.getCreateTime());
            seckillOrder.setStatus("0");    //未支付
            //将数据存入到Redis
            redisTemplate.boundHashOps( key: "SeckillOrder").put(seckillStatus.getUsername(),seckillOrder);

            //库存递减
            //seckillGoods.setStockCount(seckillGoods.getStockCount()-1);
            //计数器递减
            Long size = redisTemplate.boundHashOps( key: "SeckillGoodsCount").increment(seckillStatus.getGoodsId(),  delta: -1);
            seckillGoods.setStockCount(size.intValue());

            //如果此时StockCount==0,商品已经卖完，买完了后需要将数据同步到MySQL中
            //if(seckillGoods.getStockCount()==0){
            if(size==0){
                //同步到数据库中
                seckillGoodsMapper.updateByPrimaryKeySelective(seckillGoods);
                //Redis缓存清理掉
                redisTemplate.boundHashOps( key: "SeckillGoods_"+seckillStatus.getTime()).delete(seckillGoods.getId());
            }else{
                //否则直接同步到Redis即可
                redisTemplate.boundHashOps( key: "SeckillGoods_"+seckillStatus.getTime()).put(seckillGoods.getId(),seckillGoods);
            }

            //更新抢单状态
            seckillStatus.setMoney(Float.valueOf(seckillGoods.getCostPrice()));
            seckillStatus.setOrderId(seckillOrder.getId());
            seckillStatus.setStatus(2); //抢单成功，  等待支付
            stringRedisTemplate.boundValueOps( key: "SeckillStatus_"+seckillStatus.getUsername()).set(JSON.toJSONString(seckillStatus));
        }
    }
```

```
    }


    /***
     * 清理用户排队标示信息
     */
    public void clearSeckillStatus(SeckillStatus seckillStatus){
        //排队标示清理
        redisTemplate.delete( key: "UserQueueCount_" + seckillStatus.getUsername());
        //状态清理
        stringRedisTemplate.delete( key: "SeckillStatus_"+seckillStatus.getUsername());
    }
}
```

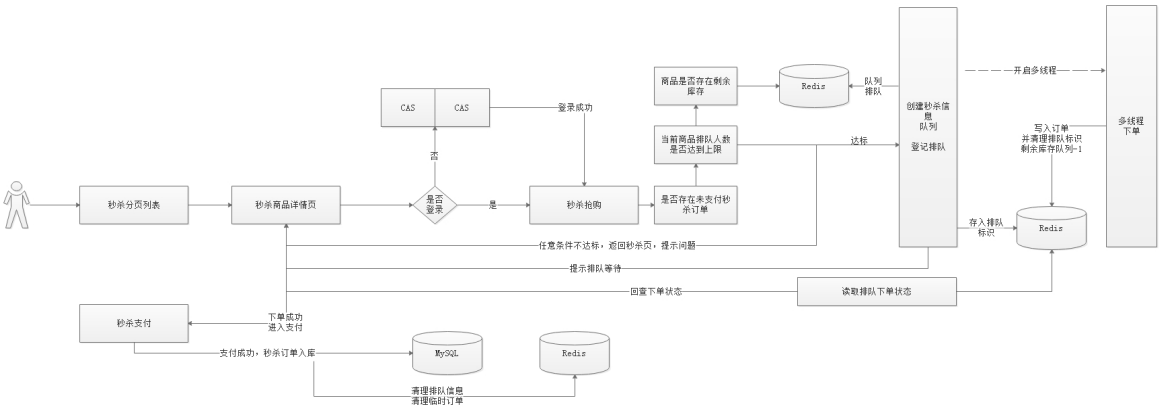
上图代码如下：

```
@Component
@RabbitListener(queues = {"seckillOrderQueue"})
public class SeckillOrderConsumer {

    @Autowired
    private RedisTemplate redisTemplate;

    @Autowired
    private StringRedisTemplate stringRedisTemplate;

    @Autowired
    private IdWorker idWorker;

    @Autowired
    private SeckillGoodsMapper seckillGoodsMapper;


    /****
     * 下单操作
     */
    @RabbitHandler
    public void readOrderMessage(String message){
        //获取抢单信息   username,id,time
        SeckillStatus seckillStatus =
JSON.parseObject(message,SeckillStatus.class);

        //用户抢单的时候已经实现了重复抢单的排除功能
        //有可能存在未支付的订单  key1=SeckillOrder    key2=username
        //Object order =
redisTemplate.boundHashOps("SeckillOrder").get(seckillStatus.getUsername());

        //if(order!=null){
            //更新抢单状态
            //System.out.println("-------存在未支付订单，不允许排队抢单----");
            //return;
        //}

        //从队列中获取该商品的队列值，判断是否有库存
        Object goodsId = redisTemplate.boundListOps("SeckillGoodsCountList_"
+ seckillStatus.getGoodsId()).rightPop();

        //如果此时商品的队列值为null，则表明该商品售罄
        if(goodsId==null){
            //清理用户排队抢单标识
            clearSeckillStatus(seckillStatus);
            return;
```

```
        }
        //商品是否有库存(存在缺陷)->是否超卖
        SeckillGoods seckillGoods = (SeckillGoods)
redisTemplate.boundHashOps("SeckillGoods_"+seckillStatus.getTime()).get(seck
illStatus.getGoodsId());

        if(seckillGoods!=null && seckillGoods.getStockCount()>0){
            //创建订单对象
            SeckillOrder seckillOrder = new SeckillOrder();
            seckillOrder.setId("No"+idWorker.nextId());
            seckillOrder.setSeckillId(seckillStatus.getGoodsId());
            seckillOrder.setMoney(seckillGoods.getCostPrice());
            seckillOrder.setUserId(seckillStatus.getUsername());
            seckillOrder.setCreateTime(seckillStatus.getCreateTime());
            seckillOrder.setStatus("0");    //未支付
            //将数据存入到Redis

 redisTemplate.boundHashOps("SeckillOrder").put(seckillStatus.getUsername(),
seckillOrder);

            //库存递减
            //seckillGoods.setStockCount(seckillGoods.getStockCount()-1);
            //计数器递减
            Long size =
redisTemplate.boundHashOps("SeckillGoodsCount").increment(seckillStatus.getG
oodsId(), -1);
            seckillGoods.setStockCount(size.intValue());

            //如果此时StockCount==0,商品已经卖完，买完了后需要将数据同步到MySQL中
            //if(seckillGoods.getStockCount()==0){
            if(size==0){
                //同步到数据库中

 seckillGoodsMapper.updateByPrimaryKeySelective(seckillGoods);
                //Redis缓存清理掉

 redisTemplate.boundHashOps("SeckillGoods_"+seckillStatus.getTime()).delete(
seckillGoods.getId());
            }else{
                //否则直接同步到Redis即可

 redisTemplate.boundHashOps("SeckillGoods_"+seckillStatus.getTime()).put(sec
killGoods.getId(),seckillGoods);
            }

            //更新抢单状态

 seckillStatus.setMoney(Float.valueOf(seckillGoods.getCostPrice()));
            seckillStatus.setOrderId(seckillOrder.getId());
            seckillStatus.setStatus(2); //抢单成功，  等待支付

 stringRedisTemplate.boundValueOps("SeckillStatus_"+seckillStatus.getUsernam
e()).set(JSON.toJSONString(seckillStatus));
        }
    }


    /***
```

```
88      * 清理用户排队标示信息
89      */
90     public void clearSeckillStatus(SeckillStatus seckillStatus){
91         //排队标示清理
92         redisTemplate.delete("UserQueueCount_" +
seckillStatus.getUsername());
93         //状态清理
94
 stringRedisTemplate.delete("SeckillStatus_"+seckillStatus.getUsername());
95     }
96 }
```

4 订单支付



完成秒杀下订单后，进入支付页面，此时前端会每3秒中向后台发送一次请求用于判断当前用户订单是否完成支付，如果完成了支付，则需要清理掉排队信息，并且需要修改订单状态信息。

4.2 创建支付二维码

下单成功后，会跳转到支付选择页面，在支付选择页面要显示订单编号和订单金额，所以我们需要在下单的时候，将订单金额以及订单编号信息存储到用户查询对象中。

选择微信支付后，会跳转到微信支付页面，微信支付页面会根据用户名查看用户秒杀订单，并根据用户秒杀订单的ID创建预支付信息并获取二维码信息，展示给用户看,此时页面每3秒查询一次支付状态，如果支付成功，需要修改订单状态信息。

4.2.1 回显订单号、金额

之前的秒杀订单状态查询已经包含了订单ID和支付金额。

访问：`<http://data-changgou-java.itheima.net/order/status?name=xiaobai>`效果如下:

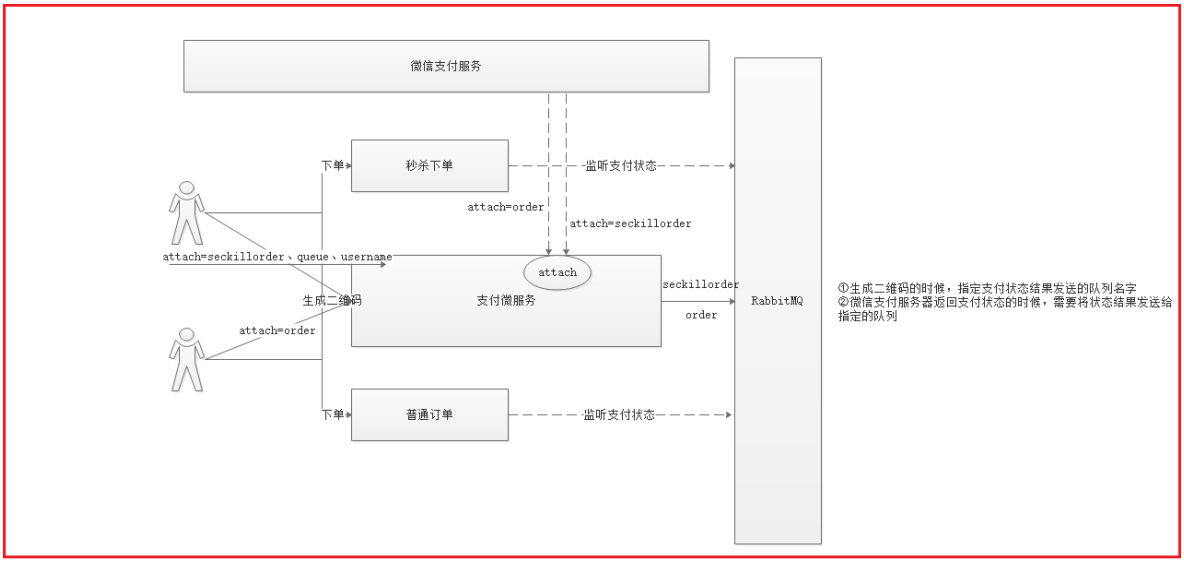
4.2.2 创建二维码

用户创建二维码，可以先查询用户的秒杀订单抢单信息，然后再发送请求到支付微服务中创建二维码，将订单编号以及订单对应的金额传递到支付微服务:`/weixin/pay/create/native`。

测试效果如下:

http://localhost:18090/weixin/pay/create/native?outtradeno=1132510782836314112&totalfee=1

4.3 支付流程分析



支付微服务目前需要为多个服务提供支付功能，支付完成后需要将微信服务器返回的支付状态发送给RabbitMQ，如果每次发送的队列名字一样，那么将无法区分是普通订单支付状态还是秒杀订单支付状态。

微信支付提供了附加参数功能，可以利用附加参数功能，在创建二维码的时候，携带附加参数为队列名字、交换机名字、用户名字，微信服务器将支付状态发送给支付微服务的时候，支付微服务可以获取原来的附加参数中的队列名字，用这种方法可以区分不同订单的队列名字。

4.4 支付回调更新

支付回调这一块代码已经实现了，但之前实现的是订单信息的回调数据发送给MQ，指定了对应的队列，不过现在需要实现的是秒杀信息发送给指定队列，所以之前的代码那块需要动态指定队列。

4.4.1 支付回调队列指定

关于指定队列如下：

```
1.创建支付二维码需要指定队列
2.回调地址回调的时候，获取支付二维码指定的队列，将支付信息发送到指定队列中
```

在微信支付统一下单API中，有一个附加参数,如下：

```
attach:附加数据,String(127)，在查询API和支付通知中原样返回，可作为自定义参数使用。
```

我们可以在创建二维码的时候，指定该参数，该参数用于指定回调支付信息的对应队列，每次回调的时候，会获取该参数，然后将回调信息发送到该参数对应的队列去。

3.4.1.1 改造支付方法

修改支付微服务的WeixinPayController的createNative方法，代码如下：

```
/***
 * 创建二维码
 * seckillorder(attach)
 */
@GetMapping(value = "/create/native")
//public Result createNative(String outtradeno,String totalfee) throws Exception{
public Result createNative(@RequestParam Map<String,String> dataMap) throws Exception{
    //创建二维码
    //Map<String,String> result = weixinPayService.createNative(outtradeno,totalfee);
    Map<String,String> result = weixinPayService.createNative(dataMap);
    return new Result( flag: true, StatusCode.OK, message: "创建二维码成功！",result);
}
```

修改支付微服务的WeixinPayService的createNative方法，代码如下：

```
/***
 * 创建预支付记录，获取支付二维码地址
 *  outtradeno:订单号
 *  totalfee:支付金额，单位分
 */
//Map createNative(String outtradeno,String totalfee) throws Exception;
Map createNative(Map<String,String> dataMap) throws Exception;
```

修改支付微服务的WeixinPayServiceImpl的createNative方法，代码如下：

```
/***
 * 创建预支付记录，获取支付二维码地址
 * outtradeno:订单号
 * totalfee:支付金额，单位分
 * @return
 */
@Override
//public Map createNative(String outtradeno, String totalfee) throws Exception {
public Map createNative(Map<String,String> parameterMap) throws Exception {
    //封装请求参数
    Map<String,String> dataMap = new HashMap<~>();
    dataMap.put("appid",appid); //应用ID
    dataMap.put("mch_id",partner);  //商户号
    dataMap.put("nonce_str", WXPayUtil.generateNonceStr()); //随机字符串
    dataMap.put("body","畅购商品");
    //============================
    dataMap.put("out_trade_no",parameterMap.get("outtradeno")); //订单编号
    dataMap.put("total_fee",parameterMap.get("totalfee"));      //交易金额，单位：分
    //============================
    //队列参数名字 queue
    Map<String,String> attachMap = new HashMap<~>();
    attachMap.put("queue",parameterMap.get("queue"));
    attachMap.put("exchange",parameterMap.get("exchange"));
    //用户名 username，适用于秒杀订单
    String username = parameterMap.get("username");
    if(!StringUtils.isEmpty(username)){
        attachMap.put("username",username);
    }

    String attachjson = JSON.toJSONString(attachMap);
    dataMap.put("attach",attachjson);

    dataMap.put("spbill_create_ip","127.0.0.1");
    //该地址用于接收微信服务器发送的微信支付状态
    dataMap.put("notify_url",notifyurl);
    dataMap.put("trade_type","NATIVE");
    //将Map转成XML字符串，并且生成签名,将请求参数转成XML字符串
    String xmlParameters = WXPayUtil.generateSignedXml(dataMap, partnerkey);

    //请求
    HttpClient httpClient = new HttpClient( url: "https://api.mch.weixin.qq.com/pay/unifiedorder");

    //参数
    httpClient.setXmlParam(xmlParameters);

    //属于Https协议
    httpClient.setHttps(true);

    //使用Post提交
    httpClient.post();

    //获取返回结果 xml格式
    String result = httpClient.getContent();

    //将返回结果转成Map
    return  WXPayUtil.xmlToMap(result);
}
```

我们创建二维码的时候，需要将下面几个参数传递过去

```
username:用户名,可以根据用户名查询用户排队信息
outtradeno：商户订单号，下单必须
totalfee：支付金额，支付必须
queue：队列名字，回调的时候，可以知道将支付信息发送到哪个队列
exchange：交换机名字
```

修改WeixinPayApplication，添加对应队列以及对应交换机绑定，代码如下：

```
@Configuration
public class RabbitMQConfig {

    /****
     * 可以获取配置文件中的指定属性的值
     */
    @Autowired
    private Environment env;

    /****
     * 创建交换机
     */
    @Bean
    public DirectExchange basicExchange(){
        return new DirectExchange(env.getProperty("mq.pay.exchange.order"), durable: true, autoDelete: false);
    }


    /***
     * 创建队列
     */
    @Bean("queueOrder")
    public Queue queueOrder(){
        return new Queue(env.getProperty("mq.pay.queue.order"));
    }

    /***
     * 秒杀支付状态队列创建队列
     */
    @Bean("queueSeckillOrder")
    public Queue queueSeckillOrder(){
        return new Queue( name: "queueSeckillOrder");
    }

    /***
     * 秒杀状态队列绑定交换机
     */
    @Bean
    public Binding basicBindingSeckill(DirectExchange basicExchange,Queue queueSeckillOrder){
        return BindingBuilder.bind(queueSeckillOrder).to(basicExchange).with( routingKey: "queueSeckillOrder");
    }

    /***
     * 队列绑定交换机
     */
    @Bean
    public Binding basicBinding(DirectExchange basicExchange,Queue queueOrder){
        return BindingBuilder.bind(queueOrder).to(basicExchange).with(env.getProperty("mq.pay.routing.key"));
    }
}
```

上图代码如下：

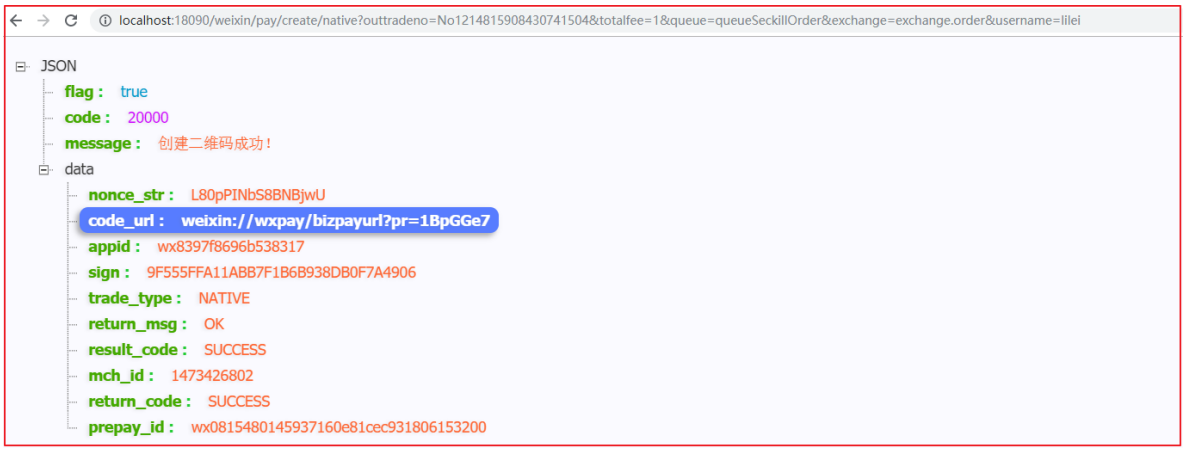
```
/***
 * 秒杀支付状态队列创建队列
 */
@Bean("queueSeckillOrder")
public Queue queueSeckillOrder(){
```

```
    return new Queue("queueSeckillOrder");
}

/***
 * 秒杀状态队列绑定交换机
 */
@Bean
public Binding basicBindingSeckill(DirectExchange basicExchange,Queue queueSeckillOrder){
    return BindingBuilder.bind(queueSeckillOrder).to(basicExchange).with("queueSeckillOrder");
}
```

4.4.1.2 测试

创建二维码测试

`<http://localhost:18090/weixin/pay/create/native?outtradeno=No121481590843074150 4&totalfee=1&queue=queueSeckillOrder&exchange=exchange.order&username=lilei>`



以后每次支付，都需要带上对应的参数，包括前面的订单支付。

4.4.1.3 改造支付回调方法

修改`com.changgou.pay.controller.WeixinPayController`的notifyUrl方法，获取自定义参数，并转成Map，获取queue地址，并将支付信息发送到绑定的queue中，代码如下：

```
/****
 * 接收微信服务器发送的支付状态数据
 */
@RequestMapping(value = "/notify/url")
public String notifyUrl(HttpServletRequest request) throws Exception{
    //获取支付结果
    ServletInputStream is = request.getInputStream();

    //接收存储网络输入流(微信服务器返回的支付状态数据)
    ByteArrayOutputStream os = new ByteArrayOutputStream();

    //缓冲区定义
    byte[] buffer = new byte[1024];
    int len = 0;

    //循环读取输入流，并写入到os中
    while ((len=is.read(buffer))!=-1){
        os.write(buffer, off: 0,len);
    }

    //关闭资源
    os.close();
    is.close();

    //将支付结果转成xml的字符串
    String xmlResult = new String(os.toByteArray(), charsetName: "utf-8");

    System.out.println(xmlResult);
    //将xmlResult转成Map
    Map<String, String> responseMap = WXPayUtil.xmlToMap(xmlResult);

    /**
     * 将支付结果发送给RabbitMQ
     * 从响应结果中获取attach附加参数 {"exchange":"exchange.order","queue":"queueSeckillOrderStatus"}
     */
    Map<String,String> queueMap = JSON.parseObject(responseMap.get("attach"),Map.class);
    rabbitTemplate.convertAndSend(queueMap.get("exchange"),queueMap.get("queue"), JSON.toJSONString(responseMap));
    //rabbitTemplate.convertAndSend(exchange,routing, JSON.toJSONString(responseMap));

    //返回结果
    Map<String,String> resultMap = new HashMap<~>();
    resultMap.put("return_code","SUCCESS");
    resultMap.put("return_msg","OK");
    return WXPayUtil.mapToXml(resultMap);
}
```

4.4.2 支付状态监听

支付状态通过回调地址发送给MQ之后，我们需要在秒杀系统中监听支付信息，如果用户已支付，则修改用户订单状态，如果支付失败，则直接删除订单，回滚库存。

在秒杀工程中创建`com.changgou.seckill.mq.SeckillOrderStatusListener`,实现监听消息，代码如下:

```
@Component
@RabbitListener(queues = {"queueSeckillOrder"})
public class SeckillOrderStatusListener {

    @Autowired
    private SeckillOrderService seckillOrderService;

    /****
```

```
     * 秒杀订单支付状态监听
     */
    @RabbitHandler
    public void getPayStatus(String message){
        System.out.println("message:"+message);
    }
}
```

4.4.3 修改订单状态

监听到支付信息后，根据支付信息判断，如果用户支付成功，则修改订单信息，并将订单入库，删除用户排队信息，如果用户支付失败，则删除订单信息，回滚库存，删除用户排队信息。

4.4.3.1 业务层

修改SeckillOrderService，添加修改订单方法，代码如下

```
/***
 * 修改订单状态
 */
void updatePayStatus(String username,String transactionid);
```

修改SeckillOrderServiceImpl，添加修改订单方法实现，代码如下：

```
/***
 * 修改订单状态
 */
@Override
public void updatePayStatus(String username, String transactionid) {
    //查询订单信息
    SeckillOrder order = (SeckillOrder)
redisTemplate.boundHashOps("SeckillOrder").get(username);

    //完善订单信息
    order.setPayTime(new Date());   //支付时间
    order.setStatus("1");   //已支付
    order.setTransactionId(transactionid); //微信交易流水号

    //同步到MySQL中
    seckillOrderMapper.insertSelective(order);

    //Redis中删除订单
    redisTemplate.boundHashOps("SeckillOrder").delete(username);

    //用户状态更新
    clearSeckillStatus(username);
}
```

4.4.3.2 修改订单对接

修改微信支付状态监听的代码，当用户支付成功后，修改订单状态，也就是调用上面的方法，代码如下：

```
/****
 * 秒杀订单支付状态监听
 */
@RabbitHandler
public void getPayStatus(String message){
    System.out.println("message:"+message);
    Map<String,String> resultMap = JSON.parseObject(message,Map.class);

    //通信标识return_code=SUCCESS
    String return_code = resultMap.get("return_code");

    if(return_code.equals("SUCCESS")){
        //业务结果result_code=SUCCESS
        String result_code = resultMap.get("result_code");

        //修改指定订单的状态，必须知道username "SeckillStatus_"+username
        //订单 必须知道username,redisTemplate.boundHashOps("SeckillOrder").put(seckillStatus.getUsername(),seckillOrder);
        //从attach附加参数中获取用户名
        Map<String,String> attachMap = JSON.parseObject(resultMap.get("attach"),Map.class);

        if(result_code.equals("SUCCESS")){
            //微信支付交易流水号transaction_id
            String transactionid =resultMap.get("transaction_id");
            //用户支付成功，修改订单状态
            seckillOrderService.updatePayStatus(attachMap.get("username"),transactionid);
        }
    }
}
```

4.4.4 删除订单回滚库存

如果用户支付失败，我们需要删除用户订单数据，并回滚库存。

4.4.4.1 业务层实现

修改SeckillOrderService，创建一个关闭订单方法，代码如下：

```
/***
 * 删除订单
 */
void closeOrder(String username);
```

修改SeckillOrderServiceImpl，创建一个关闭订单实现方法，代码如下：

```
/***
 * 删除订单
 * @param username
 */
@Override
public void closeOrder(String username) {
    //获取订单状态
"SeckillGoods_"+DateUtil.data2str(dateMenu,DateUtil.PATTERN_YYYYMMDDHH);
```

```
    SeckillStatus seckillStatus = JSON.parseObject(stringRedisTemplate.boundValueOps("SeckillStatus_"+username).get(),SeckillStatus.class);

    //回滚数据
    //查询对应的商品
    SeckillGoods seckillGoods = (SeckillGoods) redisTemplate.boundHashOps("SeckillGoods_" + seckillStatus.getTime()).get(seckillStatus.getGoodsId());

    // seckillGoods->redis=null->MySQL(同步)->MySQL少了1个->+1  AND 同步到Redis
    if(seckillGoods==null){
        seckillGoods = seckillGoodsMapper.selectByPrimaryKey(seckillStatus.getGoodsId());
        seckillGoods.setStockCount(seckillGoods.getStockCount()+1);
        //同步到MySQL
        seckillGoodsMapper.updateByPrimaryKeySelective(seckillGoods);
    }
    //同步到Redis,由于有定时器，这里不建议写库存回滚w
    //redisTemplate.boundHashOps("SeckillGoods_" + seckillStatus.getTime()).put(seckillGoods.getId(),seckillGoods);

    //计数器、队列

 //redisTemplate.boundListOps("SeckillGoodsCountList_"+seckillGoods.getId()).leftPush(seckillGoods.getId());

 //redisTemplate.boundHashOps("SeckillGoodsCount").increment(seckillGoods.getId(),1);

    //删除订单
    redisTemplate.boundHashOps("SeckillOrder").delete(username);
}
```

3.4.4.2 调用删除订单

修改 `com.changgou.seckill.mq.SeckillOrderStatusListener`，在用户支付失败后调用关闭订单方法，代码如下：

```
if(result_code.equals("SUCCESS")){
    //微信支付交易流水号transaction_id
    String transactionid =resultMap.get("transaction_id");
    //用户支付成功，修改订单状态
    seckillOrderService.updatePayStatus(attachMap.get("username"),transactionid);
}else{
    //支付失败，删除订单(真删了),回滚数据库(Redis|MySQL)数据
    seckillOrderService.closeOrder(attachMap.get("username"));
}
```

5 RabbitMQ延时消息队列

5.1 延时队列介绍

延时队列即放置在该队列里面的消息是不需要立即消费的，而是等待一段时间之后取出消费。 那么，为什么需要延迟消费呢？我们来看以下的场景

网上商城下订单后30分钟后没有完成支付，取消订单(如：淘宝、去哪儿网) 系统创建了预约之后，需要在预约时间到达前一小时提醒被预约的双方参会 系统中的业务失败之后，需要重试 这些场景都非常常见，我们可以思考，比如第二个需求，系统创建了预约之后，需要在预约时间到达前一小时提醒被预约的双方参会。那么一天之中肯定是会有很多个预约的，时间也是不一定的，假设现在有1点 2点 3点 三个预约，如何让系统知道在当前时间等于0点 1点 2点给用户发送信息呢，是不是需要一个轮询，一直去查看所有的预约，比对当前的系统时间和预约提前一小时的时间是否相等呢？这样做非常浪费资源而且轮询的时间间隔不好控制。如果我们使用延时消息队列呢，我们在创建时把需要通知的预约放入消息中间件中，并且设置该消息的过期时间，等过期时间到达时再取出消费即可。

Rabbitmq实现延时队列一般而言有两种形式： 第一种方式：利用两个特性： Time To Live(TTL)、Dead Letter Exchanges（DLX）[A队列过期->转发给B队列]

第二种方式：利用rabbitmq中的插件x-delay-message

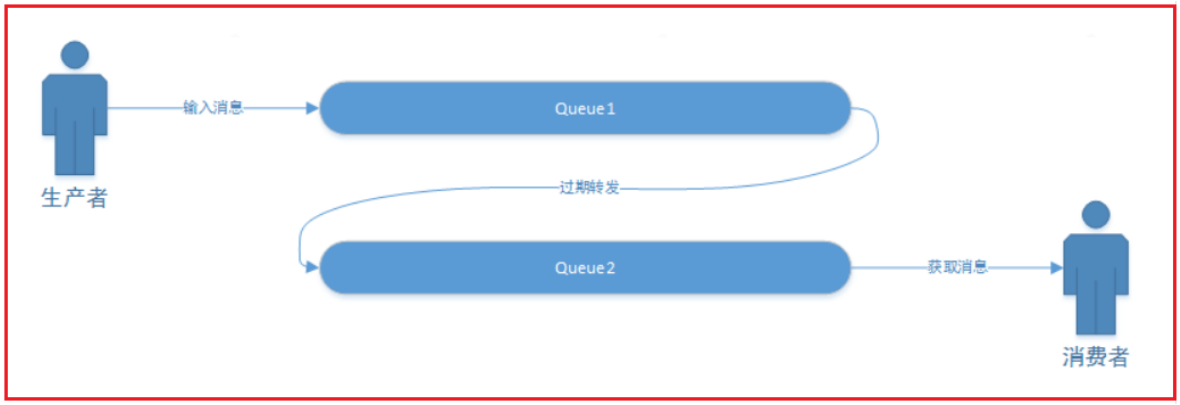
5.2 TTL DLX实现延时队列

5.2.1 TTL DLX介绍

TTL RabbitMQ可以针对队列设置x-expires(则队列中所有的消息都有相同的过期时间)或者针对Message设置x-message-ttl(对消息进行单独设置，每条消息TTL可以不同)，来控制消息的生存时间，如果超时(两者同时设置以最先到期的时间为准)，则消息变为dead letter(死信)

Dead Letter Exchanges（DLX） RabbitMQ的Queue可以配置x-dead-letter-exchange和x-dead-letter-routing-key（可选）两个参数，如果队列内出现了dead letter，则按照这两个参数重新路由转发到指定的队列。 x-dead-letter-exchange：出现dead letter之后将dead letter重新发送到指定exchange

x-dead-letter-routing-key：出现dead letter之后将dead letter重新按照指定的routing-key发送



5.2.3 DLX延时队列实现

5.2.3.1 创建工程

创建springboot-rabbitmq-delay工程，并引入相关依赖

```
1 <?xml version="1.0" encoding="UTF-8"?>
```

```xml
<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.itheima</groupId>
    <artifactId>springboot-rabbitmq-delay</artifactId>
    <version>1.0-SNAPSHOT</version>
    <description>
        RabbitMQ延时队列
    </description>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.4.RELEASE</version>
    </parent>

    <dependencies>
        <!--starter-web-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <!--加入ampq-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-amqp</artifactId>
        </dependency>

        <!--测试-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
        </dependency>
    </dependencies>
</project>
```

application.yml配置

```yaml
spring:
  application:
    name: springboot-demo
  rabbitmq:
    host: 192.168.211.132
    port: 5672
    password: guest
    username: guest
```

5.2.3.2 队列创建

创建2个队列，用于接收消息的叫延时队列queue.message.delay，用于转发消息的队列叫queue.message，同时创建一个交换机，代码如下：

```
@Configuration
public class RabbitMQConfig {

    /***
     * 创建Queue1,过期队列
     */
    @Bean
    public Queue queue1(){
        return QueueBuilder.durable("queue1")   //队列名字
                .withArgument("x-dead-letter-exchange","delayExchange")  //消息进入死信队列后，将消息发送给指定的交换机
                .withArgument("x-dead-letter-routing-key","queue2")
                .build();
    }

    /***
     * 创建Queue2,真正读取消息的队列
     */
    @Bean
    public Queue queue2(){
        return new Queue("queue2");
    }

    /***
     * 创建交换机
     */
    @Bean
    public DirectExchange delayExchange(){
        return new DirectExchange("delayExchange");
    }


    /**
     * 绑定Queue2
     */
    @Bean
    public Binding bindQueu2DelayExchange(Queue queue2,Exchange delayExchange){
        return BindingBuilder.bind(queue2).to(delayExchange).with("queue2").noargs();
    }
}
```

5.2.3.3 消息监听

创建MessageListener用于监听消息，代码如下：

```
@Component
@RabbitListener(queues = "queue2")
public class MessageListener {

    /***
```

```
     * 接收延时队列数据
     * @param message
     */
    @RabbitHandler
    public void getMessage(String message){
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        System.out.println("接到消息时间："+simpleDateFormat.format(new Date()));
        System.out.println(message);
    }
}
```

5.2.3.4 创建启动类

```
@SpringBootApplication
@EnableRabbit
public class SpringRabbitMQApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringRabbitMQApplication.class,args);
    }
}
```

5.2.3.5 测试

```
@SpringBootTest
@RunWith(SpringRunner.class)
public class RabbitMQTest {

    @Autowired
    private RabbitTemplate rabbitTemplate;

    /***
     * 延时消息发送
     */
    @Test
    public void testSendMessage(){
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        System.out.println("发送消息时间："+simpleDateFormat.format(new Date()));

        rabbitTemplate.convertAndSend("queue1", (Object) "哈哈哈过期数据！", new MessagePostProcessor() {
            @Override
            public Message postProcessMessage(Message message) throws AmqpException {
                //消息过期设置  10秒过期
                message.getMessageProperties().setExpiration("10000");
                return message;
            }
        });
```

```
        try {
            Thread.sleep(20000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

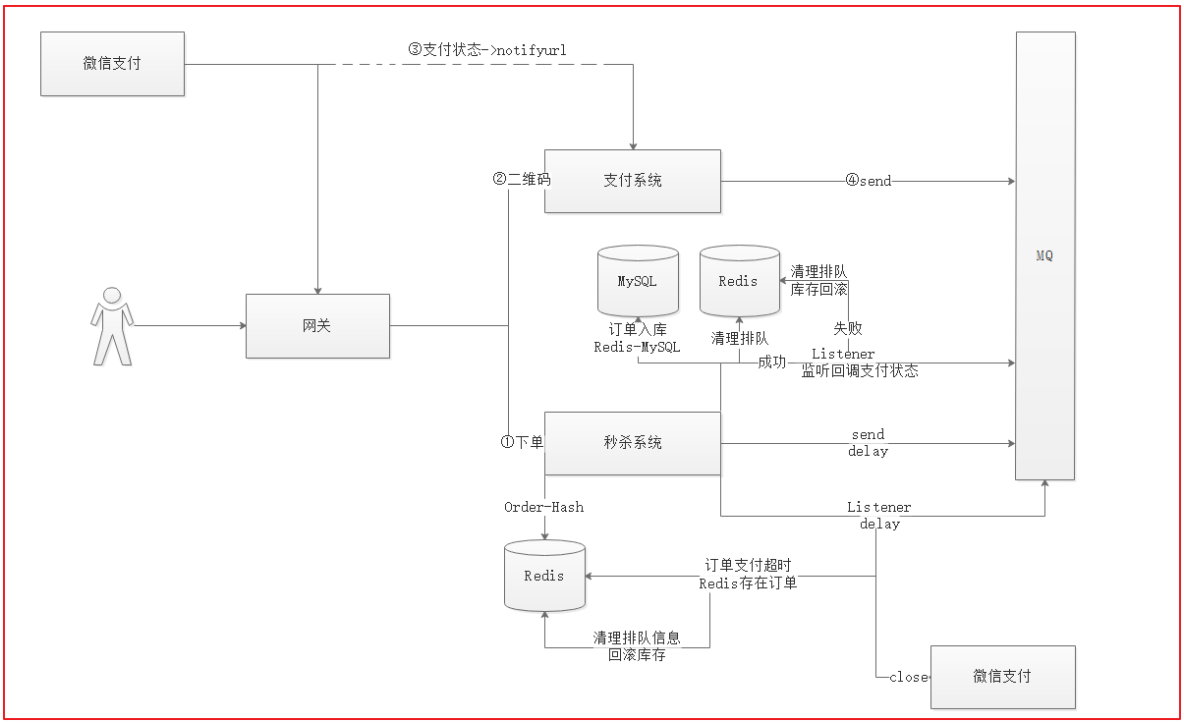
其中`message.getMessageProperties().setExpiration("10000");`设置消息超时时间,超时后，会将消息转入到另外一个队列。

测试效果如下：

```
发送当前时间:2019-05-26 14:49:30
当前时间:2019-05-26 14:49:40
收到信息:(Body:'{name=szitheima}' MessageProperties [headers={x-first-death-exchange=, x-death=[{reason=expired, original-expiration=10000, count=1, exchange=, time=Su
```

6 库存回滚(作业)

6.1 秒杀流程回顾



如上图，步骤分析如下：

1. 1.用户抢单，经过秒杀系统实现抢单，下单后会将向MQ发送一个延时队列消息，包含抢单信息，延时半小时后才能监听到
2. 2.秒杀系统同时启用延时消息监听，一旦监听到订单抢单信息，判断Redis缓存中是否存在订单信息，如果存在，则回滚
3. 3.秒杀系统还启动支付回调信息监听，如果支付完成，则将订单保存到MySQL，如果没完成，清理排队信息回滚库存
4. 4.每次秒杀下单后调用支付系统，创建二维码，如果用户支付成功了，微信系统会将支付信息发送给支付系统指定的回调地址，支付系统收到信息后，将信息发送给MQ，第3个步骤就可以监听到消息了。

延时队列实现订单关闭回滚库存：

```
1.创建一个过期队列   Queue1
2.接收消息的队列     Queue2
3.中转交换机
4.监听Queue2
    1)SeckillStatus->检查Redis中是否有订单信息
    2)如果有订单信息，调用删除订单回滚库存->[需要先关闭微信支付]
    3)如果关闭订单时，用于已支付，修改订单状态即可
    4)如果关闭订单时，发生了别的错误，记录日志，人工处理
```

6.2 关闭支付

用户如果半个小时没有支付，我们会关闭支付订单，但在关闭之前，需要先关闭微信支付，防止中途用户支付。

修改支付微服务的WeixinPayService，添加关闭支付方法，代码如下：

```
/***
 * 关闭支付
 * @param orderId
 * @return
 */
Map<String,String> closePay(Long orderId) throws Exception;
```

修改WeixinPayServiceImpl，实现关闭微信支付方法，代码如下：

```
/***
 * 关闭微信支付
 * @param orderId
 * @return
 * @throws Exception
 */
@Override
public Map<String, String> closePay(Long orderId) throws Exception {
    //参数设置
    Map<String,String> paramMap = new HashMap<String,String>();
    paramMap.put("appid",appid); //应用ID
    paramMap.put("mch_id",partner);    //商户编号
    paramMap.put("nonce_str",WXPayUtil.generateNonceStr());//随机字符
    paramMap.put("out_trade_no",String.valueOf(orderId));   //商家的唯一编号

    //将Map数据转成XML字符
    String xmlParam = WXPayUtil.generateSignedXml(paramMap,partnerkey);

    //确定url
    String url = "https://api.mch.weixin.qq.com/pay/closeorder";

    //发送请求
    HttpClient httpClient = new HttpClient(url);
    //https
    httpClient.setHttps(true);
    //提交参数
    httpClient.setXmlParam(xmlParam);
```

```
28
29      //提交
30      httpClient.post();
31
32      //获取返回数据
33      String content = httpClient.getContent();
34
35      //将返回数据解析成Map
36      return  WXPayUtil.xmlToMap(content);
37  }
```

6.3 关闭订单回滚库存

6.3.1 配置延时队列

在application.yml文件中引入队列信息配置，如下：

```
1  #位置支付交换机和队列
2  mq:
3    pay:
4      exchange:
5        order: exchange.order
6      queue:
7        order: queue.order
8        seckillorder: queue.seckillorder
9        seckillordertimer: queue.seckillordertimer
10       seckillordertimerdelay: queue.seckillordertimerdelay
11     routing:
12       orderkey: queue.order
13       seckillorderkey: queue.seckillorder
```

配置队列与交换机,在SeckillApplication中添加如下方法

```
1  /**
2   * 到期数据队列
3   * @return
4   */
5  @Bean
6  public Queue seckillOrderTimerQueue() {
7      return new Queue(env.getProperty("mq.pay.queue.seckillordertimer"),
   true);
8  }
9
10 /**
11  * 超时数据队列
12  * @return
13  */
14 @Bean
15 public Queue delaySeckillOrderTimerQueue() {
16     return
   QueueBuilder.durable(env.getProperty("mq.pay.queue.seckillordertimerdelay"))
```

```
            .withArgument("x-dead-letter-exchange", env.getProperty("mq.pay.exchange.order"))        // 消息超时进入死信队列，绑定死信队列交换机
            .withArgument("x-dead-letter-routing-key", env.getProperty("mq.pay.queue.seckillordertimer"))   // 绑定指定的routing-key
            .build();
}

/***
 * 交换机与队列绑定
 * @return
 */
@Bean
public Binding basicBinding() {
    return BindingBuilder.bind(seckillOrderTimerQueue())
            .to(basicExchange())
            .with(env.getProperty("mq.pay.queue.seckillordertimer"));
}
```

6.3.2 发送延时消息

修改MultiThreadingCreateOrder，添加如下方法：

```
/***
 * 发送延时消息到RabbitMQ中 队列1  队列2
 * @param seckillStatus
 */
@RequestMapping(value="/test")
public void sendTimerMessage(SeckillStatus seckillStatus){

 rabbitTemplate.convertAndSend(env.getProperty("mq.pay.queue.seckillordertimerdelay"), (Object) JSON.toJSONString(seckillStatus), new MessagePostProcessor() {
        @Override
        public Message postProcessMessage(Message message) throws AmqpException {
            message.getMessageProperties().setExpiration("10000");
            return message;
        }
    });
}
```

在createOrder方法中调用上面方法，如下代码：

```
//发送延时消息到MQ中
sendTimerMessage(seckillStatus);
```

6.3.3 库存回滚

创建SeckillOrderDelayMessageListener实现监听消息，并回滚库存，代码如下：

```
@Component
```

```java
@RabbitListener(queues = "${mq.pay.queue.seckillordertimer}")
public class SeckillOrderDelayMessageListener {

    @Autowired
    private RedisTemplate redisTemplate;

    @Autowired
    private SeckillOrderService seckillOrderService;

    @Autowired
    private WeixinPayFeign weixinPayFeign;

    /***
     * 读取消息
     * 判断Redis中是否存在对应的订单
     * 如果存在，则关闭支付，再关闭订单
     * @param message
     */
    @RabbitHandler
    public void consumeMessage(@Payload String message){
        //读取消息
        SeckillStatus seckillStatus =
JSON.parseObject(message,SeckillStatus.class);

        //获取Redis中订单信息
        String username = seckillStatus.getUsername();
        SeckillOrder seckillOrder = (SeckillOrder)
redisTemplate.boundHashOps("SeckillOrder").get(username);

        //如果Redis中有订单信息，说明用户未支付
        if(seckillOrder!=null){
            System.out.println("准备回滚---"+seckillStatus);
            //关闭支付
            Result closeResult =
weixinPayFeign.closePay(seckillStatus.getOrderId());
            Map<String,String> closeMap = (Map<String, String>)
closeResult.getData();

            if(closeMap!=null &&
closeMap.get("return_code").equalsIgnoreCase("success") &&
                    closeMap.get("result_code").equalsIgnoreCase("success")
){
                //关闭订单
                seckillOrderService.closeOrder(username);
            }
        }
    }
}
```