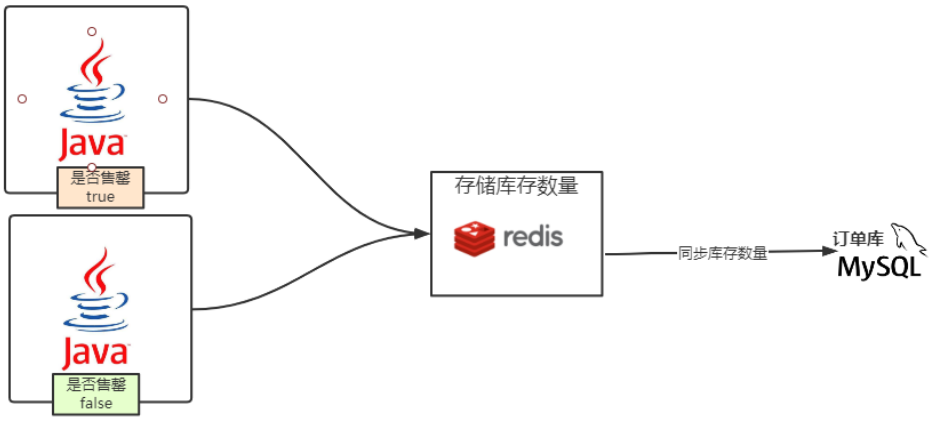
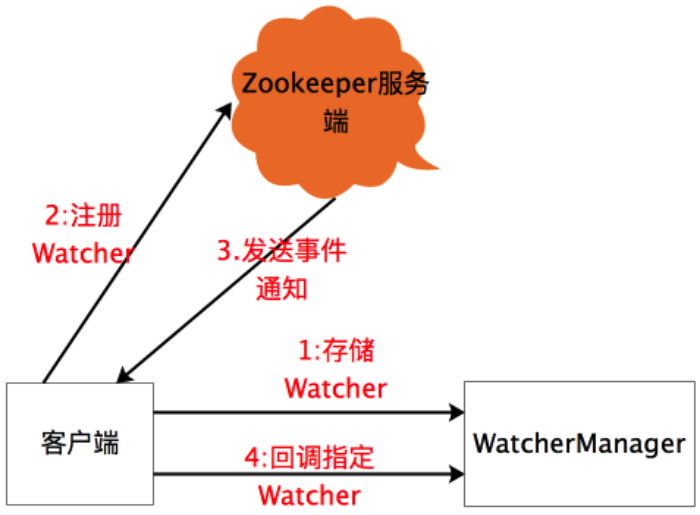
售罄状态同步方案：



1、zookeeper

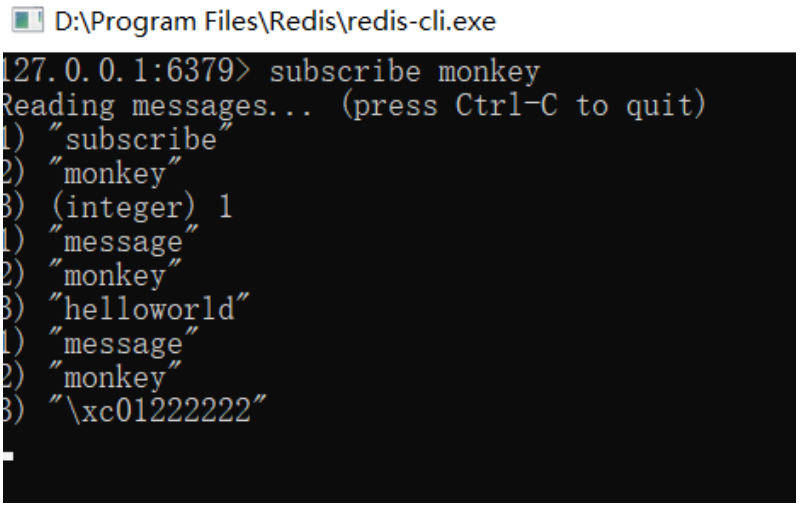
利用zk的watcher机制实现



2、redis

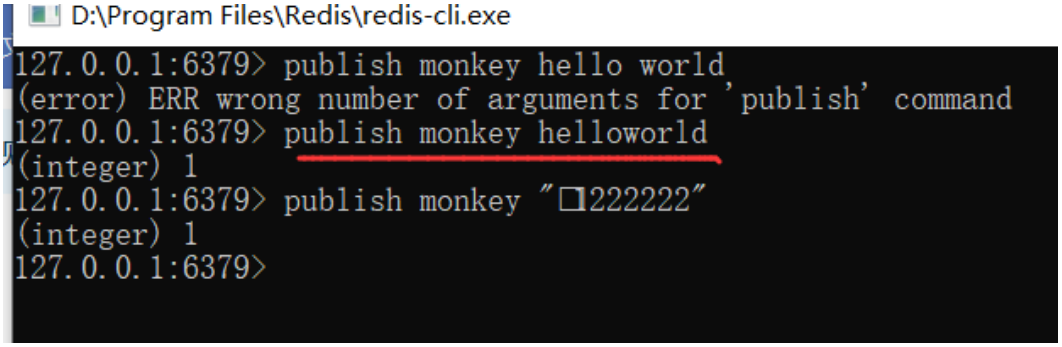
利用redis的channel机制实现。

一个客户端订阅主题：



命令：subscribe monkey

一个客户端向订阅的主题（channel）发送消息：



命令：publish monkey hello

```
//通知服务群,清除本地售罄标记缓存
if (shouldPublishCleanMsg(productId)) {
 redisOpsUtil.publish("cleanNoStockCache", productId);
}
```

监听类：

com.tuling.tulingmall.component.RedisChannelListener

3、mq...其他方式

这就不都说了

那zk和redis哪种方案更好了？同学们

redis这种发布与订阅是没有ack的，发出去了不会管有没有收到。这是它的不足，那优点是什么了

优点就是其缺点，吞吐量相当来说就会提高，因为减少了通讯，那处理数据的能力就就会上升

秒杀商品的预热【product】：

在启动的时候就把秒杀商品的库存放到redis中。

```
com.tuling.tulingmall.config.RedisConifg
```

全量刷秒杀库存：
com.tuling.tulingmall.config.RedisConifg#afterPropertiesSet

异步下单：

之前方案不足： 数据库insert 很多表 数据库优化
1、颠覆性 》mysql oracle
2、改进型》索引、分库分表、读写分离

处理未支付的订单（20分钟）：
Topic：order-status-check
生产端：
com.tuling.tulingmall.component.rocketmq.OrderMessageSender#sendTimeOutOrderMessage
消费端：
com.tuling.tulingmall.component.rocketmq.RocketMqCancelOrderReciever

Canal同步topic：
Topic：productDetailChange
生产端：canal
消费端：com.tuling.tulingmall.mq.RefreshCacheListener

async-order异步下单topic：
实际生产订单：SecKillOrderServiceImpl#asyncCreateOrder
处理订单：com.tuling.tulingmall.component.rocketmq.AscynCreateOrderReciever
com.tuling.tulingmall.service.impl.SecKillOrderServiceImpl#asyncCreateOrder //TODO 分布事务、分布式消息

部署不能这么玩：拆分

异步订单查询接口：

com.tuling.tulingmall.controller.OmsPortalOrderController#miaoShaResult

```
@ApiOperation("根据购物车信息生成订单")
@GetMapping("/miaosha/result")
@ResponseBody
```

```
public CommonResult miaoShaResult(@RequestParam("productId") Long productId,@RequestHeader("memberId") Long memberId){
  String status = redisOpsUtil.get(RedisKeyPrefixConst.MIAOSHA_ASYNC_WAITING_PREFIX + memberId
  + ":" + productId);

  if(ObjectUtils.isEmpty(status)){
  return CommonResult.success(null,"无正在秒杀中的订单！");
   }

   if(status.equals("-1")){
   return CommonResult.success(status,"秒杀失败！");
   }

   if(status.equals("1")){
   return CommonResult.success(status,"正在排队中,请耐心等待！");
   }
   //如果Status>1，则秒杀成功,返回订单编号
   return CommonResult.success(status);
}
```

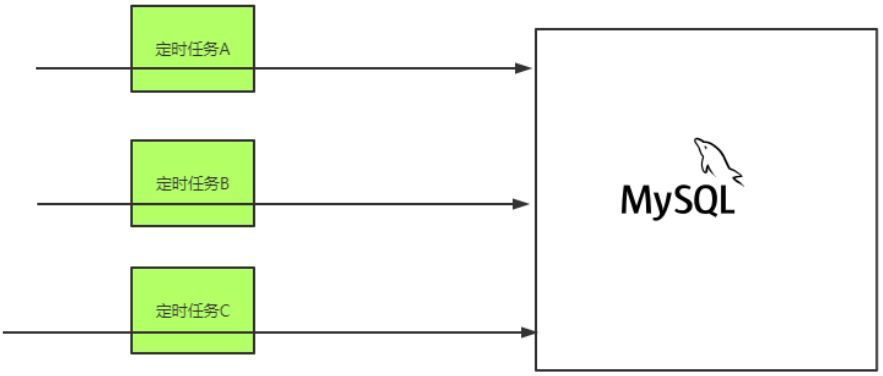
总结：

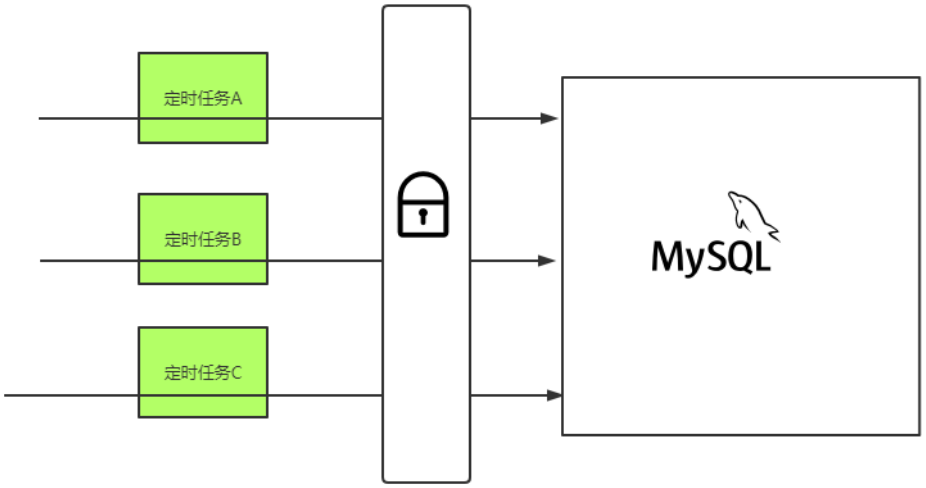
1、异步下单可以分流、让服务器处理的压力变小、数据库压力减少
2、解耦的话，业务更加清晰。
3、天然的排队处理能力。
4、消息中间件有很多特性可以利用，比如订单取消。

订单取消：

订单超时取消，回滚库存：
com.tuling.tulingmall.component.rocketmq.RocketMqCancelOrderReciever

定时任务的问题：
1、11点启动定时任务，每半个小时扫码数据库一次，我们发现在11:01分下的单并不能30分钟之后失效，而是要到12点也就是定时任务第三次扫码数据库才能让订单失效。
2、定时扫数据库的话消耗性能也很大，自然效率也会很低。对数据库压力太大。
3、定时任务的话，集群还需要保证处理的幂等性和分布式问题。这也给系统带来了很多的负担。





异步取消订单：

com.tuling.tulingmall.component.rocketmq.RocketMqCancelOrderReciever

创建订单、发送延迟20分钟消息：

```
com.tuling.tulingmall.service.impl.SecKillOrderServiceImpl#asyncCreateOrder
 >com.tuling.tulingmall.component.rocketmq.OrderMessageSender#sendTimeOutOrderMessage
/*
```

```
 * 如果订单创建成功,需要发送定时消息,20min后如果没有支付,则取消当前订单,释放库存
 */
try {
 boolean sendStatus = orderMessageSender.sendTimeOutOrderMessage(order.getId() + ":" + flashPromotionRelationId + ":" + orderItem.getProductId());
 if (!sendStatus) {
 throw new RuntimeException("订单超时取消消息发送失败！");
 }
} catch (Exception e) {
 throw new RuntimeException("订单超时取消消息发送失败！");
}

```

预减库存preDecrRedisStock方法：

通过redis的decr函数扣减库存。

如果没有库存了，stock小于0时 发消息给rocketmq同步库存 redis设置为0

redis与db同步订单：

com.tuling.tulingmall.component.rocketmq.OrderMessageSender#sendStockSyncMessage

利用rocketmq延迟消息的一个特性来解决“定时任务”来取消订单操作。

RocketMQ消息零丢失：

生产端：同步发送消息、重试机制、事务消息、状态

服务端：刷盘存储、主从同步、 状态返回

消费端：pull broker offset 消费端并且返回成功 偏移值，如果消费失败那条数据

RocketMQ消息不被重复消费：

重复问题、幂等性问题。redis incr 数据库唯一主键

orderid：20210630 》key 1

数据同步Canal：

场景模拟：在秒杀后台把价格修改之后，如何同步到缓存中，比如redis如何

canal闪亮登场

#参考另外一份文档 《canal安装与使用》

项目中对product和秒杀表修改做同步操作：

```
com.tuling.tulingmall.mq.RefreshCacheListener#onMessage
```

文档：06 秒杀系统-订单交易全链路优化实战?..
链接：http://note.youdao.com/noteshare?id=eae460dc68dee2f5767ab04223d65a3e&sub=7FA459137A104969B64095B90AD2A5BC