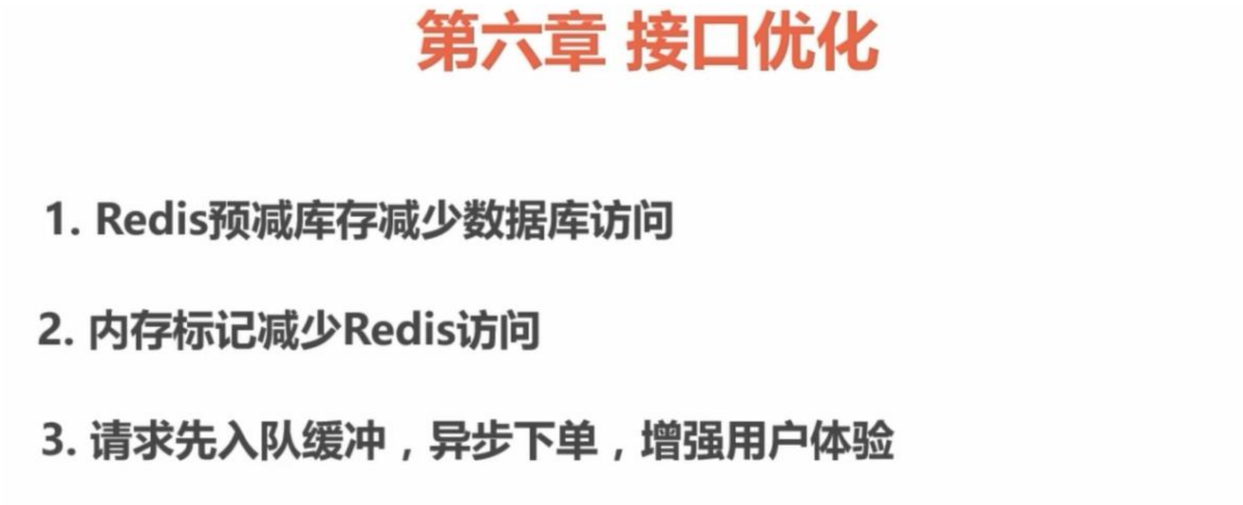
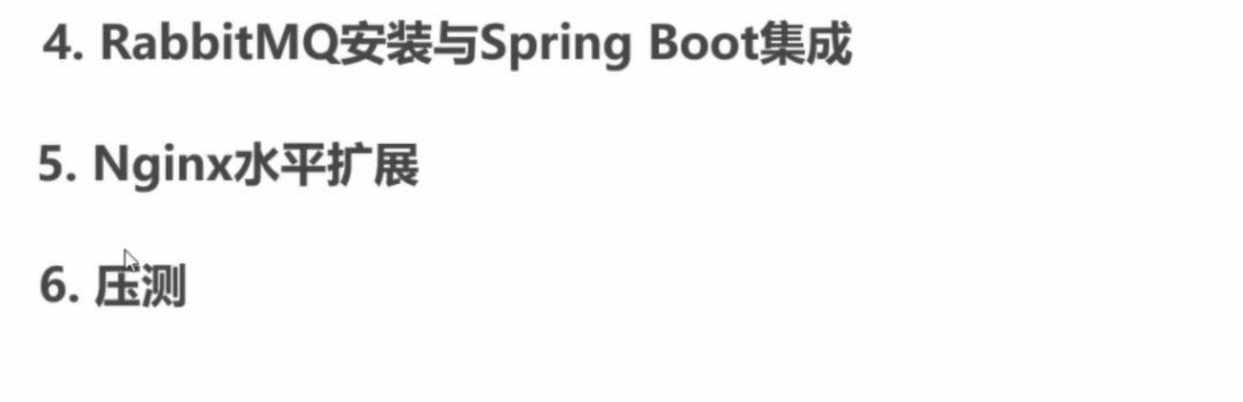


第六章 接口优化

1. Redis预减库存减少数据库访问

2. 内存标记减少Redis访问

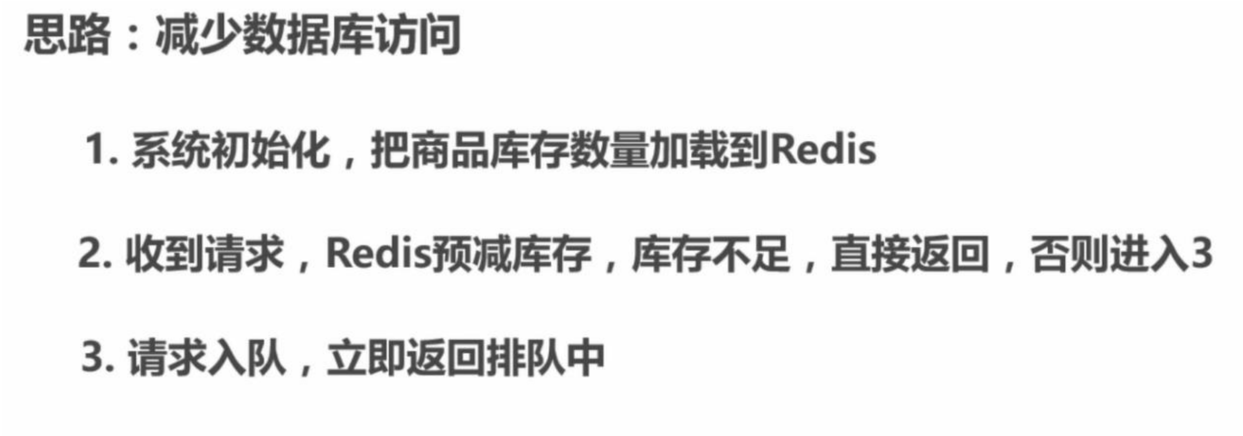
3. 请求先入队缓冲，异步下单，增强用户体验



4. RabbitMQ安装与Spring Boot集成

5. Nginx水平扩展

6. 压测



思路：减少数据库访问

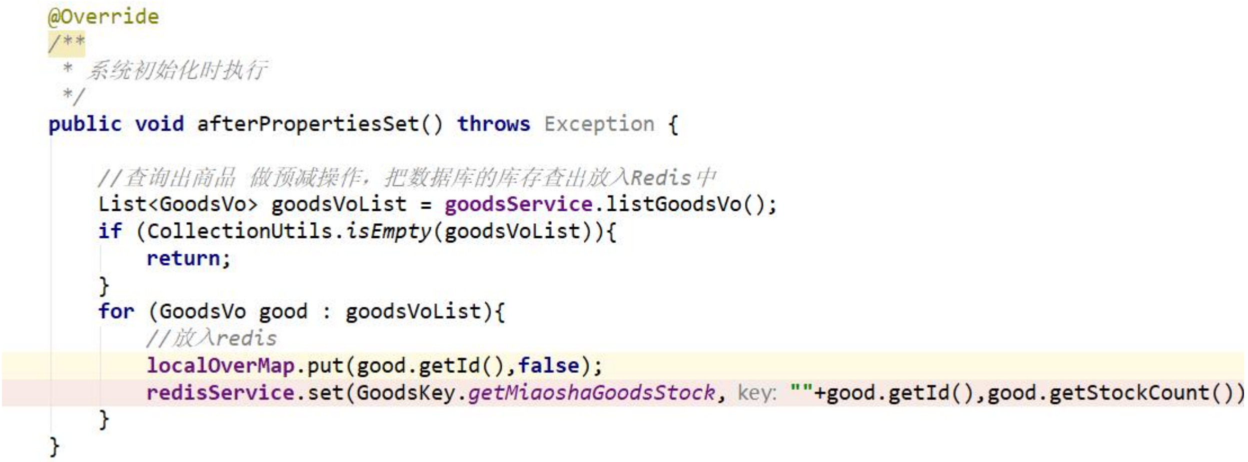
1. 系统初始化，把商品库存数量加载到Redis

2. 收到请求，Redis预减库存，库存不足，直接返回，否则进入3

3. 请求入队，立即返回排队中

1、redis预减库存

controller实现接口InitializingBean

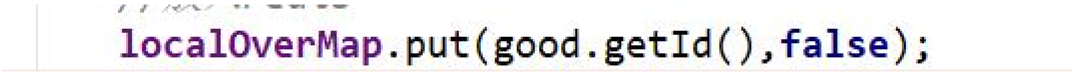


```
@Override
/**
 * 系统初始化时执行
 */
public void afterPropertiesSet() throws Exception {

    //查询出商品 做预减操作，把数据库的库存查出放入Redis中
    List<GoodsVo> goodsVoList = goodsService.listGoodsVo();
    if (CollectionUtils.isEmpty(goodsVoList)){
        return;
    }
    for (GoodsVo good : goodsVoList){
        //放入redis
        localOverMap.put(good.getId(),false);
        redisService.set(GoodsKey.getMiaoshaGoodsStock, key: ""+good.getId(),good.getStockCount())
    }
}
```

2、内存标记减少访问redis

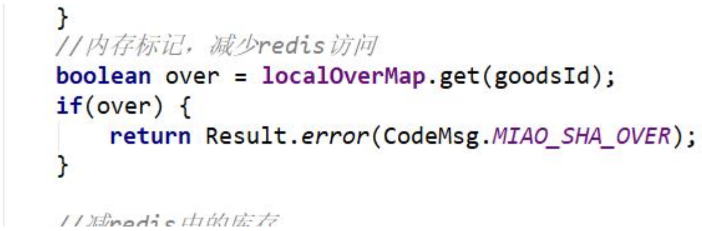
```
//内存标记
private HashMap<Long, Boolean> localOverMap =  new HashMap<Long, Boolean>();
```

在redis预减时打个标记



```
    localOverMap.put(good.getId(),false);
```

在秒杀时先判断是否over了



```
}
//内存标记，减少redis访问
boolean over = localOverMap.get(goodsId);
if(over) {
    return Result.error(CodeMsg.MIAO_SHA_OVER);
}

//减redis中的库存
```

如果redis已经减空了就设置为true

```
if (stock < 0){
  //减空了
    localOverMap.put(goodsId,true);
  return Result.error(CodeMsg.MIAO_SHA_OVER);
}
```

3、使用rabbitmq，使用队列来缓存 这样减库存下订单的操作是异步的，增强用户体验。

```
//发送消息给队列 入队操作
MiaoshaMessage mm  = new MiaoshaMessage();


mm.setGoodsId(goodsId);
mm.setUser(user);
sender.sendMiaoshaMessage(mm);
```

```
return Result.success(0);//排队中
```

MQConfig.java 用于创建队列bean

```
@Configuration
public class MQConfig {

  public static final String MIAOSHA_QUEUE = "miaosha.queue";
@Bean
public Queue miaosh_queue() {
  return new Queue(MIAOSHA_QUEUE, true);
}
}
```

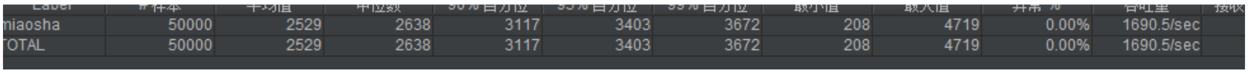
MQSender.java 用于发消息　（生产者）

```
@Service
public class MQSender {

  private static Logger log = LoggerFactory.getLogger(MQSender.class);

  @Autowired
  AmqpTemplate amqpTemplate ;

  public void sendMiaoshaMessage(MiaoshaMessage mm) {
    String msg = RedisService.beanToString(mm);
    log.info("send message:"+msg);
    amqpTemplate.convertAndSend(MQConfig.MIAOSHA_QUEUE, msg);
  }
}
```

MQReceiver.java 用于接收消息并作出相应的处理　（消费者）

```
@Service
public class MQReceiver {

    private static Logger log = LoggerFactory.getLogger(MQReceiver.class);

    @Autowired
    RedisService redisService;

    @Autowired
    GoodsService goodsService;

    @Autowired
    OrderService orderService;

    @Autowired
    MiaoshaService miaoshaService;

    @RabbitListener(queues=MQConfig.MIAOSHA_QUEUE)
    public void receive(String message) {
      log.info("receive message:"+message);
      MiaoshaMessage mm  = RedisService.stringToBean(message, MiaoshaMessage.class);
      MiaoshaUser user = mm.getUser();
      long goodsId = mm.getGoodsId();

      GoodsVo goods = goodsService.getGoodsVoByGoodsId(goodsId);
       int stock = goods.getStockCount();
       if(stock <= 0) {
         return;
       }
       //判断是否已经秒杀到了
           MiaoshaOrder order = orderService.getMiaoshaOrderByUserIdGoodsId(user.getId(),
goodsId);
       if(order != null) {
         return;
       }
       //减库存 下订单 写入秒杀订单
           miaoshaService.miaosha(user, goods);
    }
```

压测结果



Label	# 样本	平均值	中位数	90% 百分位	95% 百分位	99% 百分位	最小值	最大值	异常 %	吞吐量	接收
miaosha	50000	2529	2638	3117	3403	3672	208	4719	0.00%	1690.5/sec	
TOTAL	50000	2529	2638	3117	3403	3672	208	4719	0.00%	1690.5/sec	

之前是1295

测试时发现在第一次测试时没有异常，但是紧接着第二次压测的时候回出现异常，原因估计是window的tcp连接数的问题。