**项目目的 ：**

实现一个Java的高并发的抢购商品系统后台开发 。 利用SpringBoot熟悉Java后台开发的简单流程 ，常用工具，以及实际处理高并发业务常用的手段 。

**使用场景和问题 ：**

阿里巴巴的双11，京东618 ， 再到12306的抢票机制 ......但是，客户端在同一时刻查询的人次过多 ，每个人次都会查询，修改数据，对后端数据库产生极大压力

(MySQL的访问速度一般认为为 800 ~ 2000次 / 秒) 。

**解决思路 ：**

减少对单个数据库服务器的访问压力，主要思路 ：

尽量把数据的访问拦截在MySQL服务器之前 ，优化的方法 ：

1. 缓存优化 ： Redis 。

提前把商品数据放入内存数据库Redis中，每次优先访问内存中的Redis数据库 。 访问策略 ：

读取 ： 优先在 Redis中访问。 存在则使用之 ；不存在则再去访问MySQL，写回到Redis并返回

修改 ： 先访问Redis 。 如果命中则修改之 ，再异步地进行MySQL的修改 ；不存在则先修改MySQL，再写回到Redis ,附加一个生存时间 。

2. 异步 && 削峰 : RabbitMQ 。

消息队列进行异步处理并且削峰 。 使用方法 ：

在Redis中的商品一次抢购之后 ，立即返回抢购结果给用户 。对数据库的修改加入消息队列，Redis过滤过得消息 ， 让消息队列进行一次削峰，并且异步修改MySQL 。

3. 进一步考虑 （完善点）：

分布式方向 。增加数据库服务器 ，主从复制 ， 实现读写分离 ： 在主服务器上写，从服务上读 。 这个方向后期完善， 项目中并未实现之。

**架构分析 ：**

1. 前端模块 。

项目本身专注于Java后台部分的开发 ，前端部分仅仅是使用SpringBoot配套的Thymeleaf ，只做到显示数据变化，并未专注于此的 ， 后来涉及到前端的异步请求知识，自己未能成功解决，于是放弃了Thymeleaf ，直接返回一个数据给网页 。

如果要优化之（项目本身没有实现），可考虑方向 ：

1）页面静态化 ，把秒杀页面提前放到CDN中

2）客户端缓存 。可以考虑浏览器http的Cache-Control字段设置 ，让数据缓存一定的时间，提高用户体验

2. 限购模块（核心）。

使用Redis的集合数据结构，每个下单的客户端都要求先在Redis集合中判断一次ip存在与否 。 如果已经存在，则用户已经操作成功 ， 不能参与下一次抢购 ； 不存在 ，则加入ip到这个集合，最后再进入排队模块。

3. 排队模块（核心部分）。

提前把数据加入到Redis中，抢购开始后 ：

1） 客户端对Redis中的抢购商品访问并且预减库存 ，然后立即返回抢购结果 ，请用户等待相应 。 这里使用Redis把大部分请求都拦截住 ，因为Redis的操作都是原子性的 ，且是单线程实例，所以，不用考虑并发安全性问题，即可拦截大部分请求 。

2）每次的抢购预减库存要求写回MySQL数据库 ，这使用消息队列， 异步地写回数据库 ， 可以控制并发的峰值降低 。

4. 服务模块 （核心）。

消息队列的消费者，业务逻辑即是使用事务控制（锁）对数据库的下订单 ，减库存操作 。

这里为了避免无条件的表锁（批量操作的update操作时，MySQL会倾向于会锁住整个表，少量时候，如果没有索引，所以update会锁表，如果加了索引，就会锁行），采用的MySQL 语句是 select .....for update 显示加锁 。

5. 全局的异常处理 ：

在下订单的过程中 ，如果某些用户下单失败 ：

1） 存库不足 ，已为 <0 。提示用户 ，库存不足 ，下次努力 。

2） 限购模块 。 取消用户下一步的排队模块 ， 返回用户 ，已经抢购成功 。

**开发环境和技术选型 ：**

1. IDEA集成的SpringBoot框架 。

原因 ：简单，快速。

springboot并不是替代其他Spring开发框架的技术 ，我的理解是 ，它是spring框架的框架 ， 使用它可以减少繁琐的配置。 在整个开发过程中 ，全程只涉及pom.xml和application.properties的修改 ，简单太多太多。普通 其内部继 承了tomcat ，很多配置也都是开发的默认配置。在其他的Java项目中 ，大量的XML文件配置，而且第三方框架的配置出现冲突问题多。

但是也有不足。目前的帮助文档并不足够多，很多并非是正确的 。比如我在开发中配置数据库 ，就没想到springboot中默认使用的Jpa ，没有自己修改配置 ；

适用于全新的Java项目 。 把一个传统的Spring Framework项目转化成SpringBoot是很消耗精力的。

集成度过高 ，有时问题不好排查 。

2 . 内存数据库 ： Redis

优点太多 ：

1) 支持丰富的数据结构 ： 链表 ，集合 ，有序集合 ， 哈希 ，字符串 ，还有比较少使用的HyperLogLog ,位图 ， 地址位置信息GEO，数据的大小限制也松，可以到达521M ~ 1GB，还有时间过期功能

2) 支持持久化数据 。 两种方式持久化 ， 一种周期性生成快照 ，恢复快 ，但是可能损失数据 ； 一种记录所有的修改操作 ，无数据损失 ，但是恢复的话，要重写一次Redis所有的修改操作 ，较慢。

3) 基于内存，速度更快 。默认读写速度在 10,000次 / 秒

4) 线程安全 。基于异步队列的服务实例 ，而且使用单进程单线程去执行所有的原子操作 ，简单安全

5）支持事务 。

6） 适用于阻塞队列 ，发布订阅 ， 计数，缓存，好友推荐（集合交并补操作）

7）再高级的就是 主从 ， 哨兵 ，集群 ，分布式锁了

另外 ： Redis和Springboot都是pivotal赞助开发的

3.消息队列 ： RabbitMQ 。

使用消息队列，只要考虑三个好处 ：异步 ，削峰 ， 解耦

主流的消息队列 ，RabbitMQ ,Kafaka ,Rocket。

比较： RocketMQ是阿里巴巴在Kafaka的基础上修改的 ，采用Java编写 ，两者支持的并发量级都是10W级， 而RabbitMQ更早出现 ， 使用erlang语言开发的 ， 尽管并发量级在W级 ，但是对消息的实时性保障更高。

另外RabbitMQ的使用帮助也最多,比kafaka更成熟 ，且自身支持持久化数据，擅长处理业务方面 ，kafaka擅长支持处理日志。

没有最好 ，只能更合适。尽管后两个支持并发量级更高，但是在可用性稳定上考虑，确是前者要好 ， 所以我选择RabbitMQ.

另外 ，并非就是阿里巴巴一定使用Rocket. 阿里云计算的第一负责人余锋，就支持使用RabbitMQ . 旺旺平台也是 ， 他的理由 ：

1. 基于 AMQP协议

2. 支持高并发 ，Erlang语言 ，高性能 ，W级别，但是CPU消耗也挺大的 。

3.高性能 ，高可靠 ，处处维稳

4.支持多种语言 ： python,php

5.社区活跃 ， Spring AMQP还极大简化了RabbitMQ的使用。

**深入优化 ：**

**这部分项目中并未实现 ，但是考虑未来的优化的话 ，可以从此着手**

1.提前预加热数据

在秒杀开始前 ，就把抢购商品加入到Redis中 。

2. 限制用户数

其实在秒杀开始一段时间 ，我们可以设置一个阈值 ，限制参与秒杀抢购的用户数目

3. 主从复制， 读写分离

在对后端数据库访问时，可以考虑使用主从模式 ，读在Salve ,写在master ,但是可能主从复制还会有一定的缓冲时间来同步数据 ，所以，不好说