# 随手快递

随手快递的业务模式与滴滴很相似，以众包的方式利用社会闲散人力资源，解决O2O最后三公里即时性配送难题（目前达达已经与京东到家合并）。 达达业务主要包含两部分：商家发单，配送员接单配送，如下图所示。

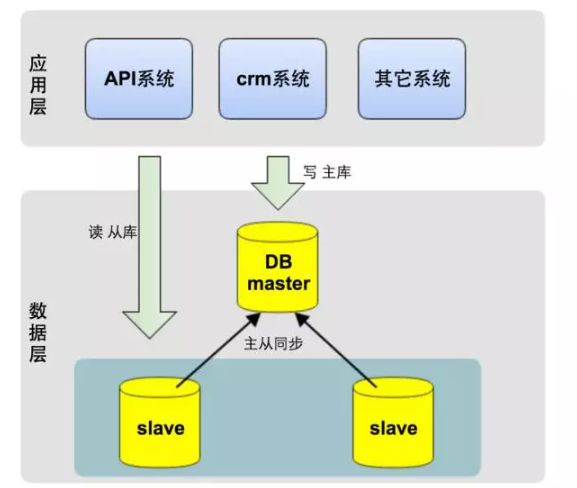


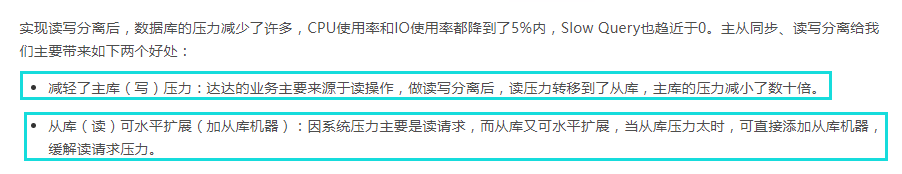
压力主要分为两类: 读压力、写压力。读压力来源于配送员在APP中抢单，高频刷新查询周围的订单，每天访问量几亿次，高峰期QPS高达数千次/秒。写压力来源于商家发单、达达接单、取货、完成等操作。达达业务读的压力远大于写压力，读请求量约是写请求量的30倍以上。

## 问题一

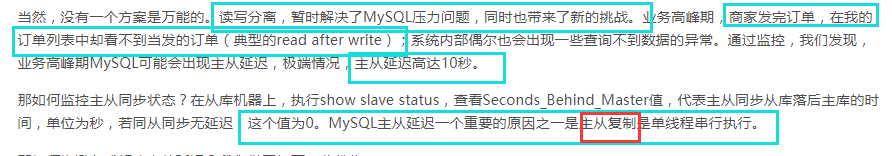
性能不能满足：每次请求的响应时间越来越长，比如配送员在app中刷新周围订单，响应时间从最初的500毫秒增加到了2秒以上。业务高峰期，系统甚至出现过宕机。通过监控，我们发现高期峰MySQL CPU使用率已接近80%，磁盘IO使用率接近90%，Slow Query从每天1百条上升到1万条，而且一天比一天严重。数据库俨然已成为瓶颈，我们必须得快速做架构升级。

## 升级1MySQL主从同步和应用服务端读写分离方案。

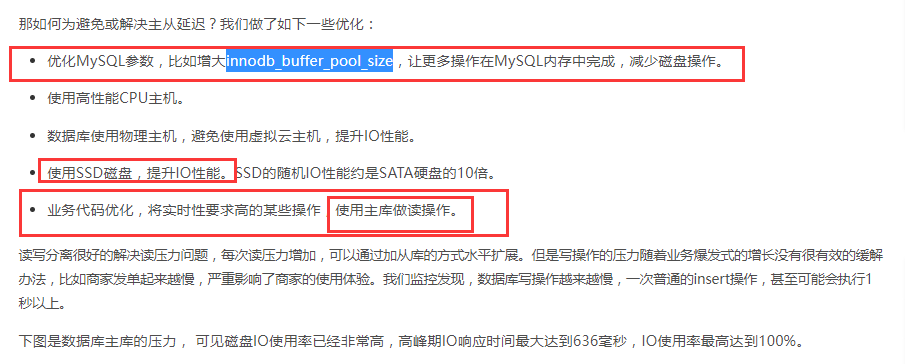




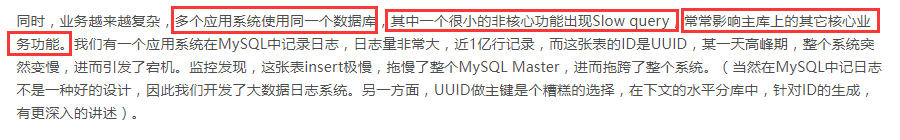
## 问题二



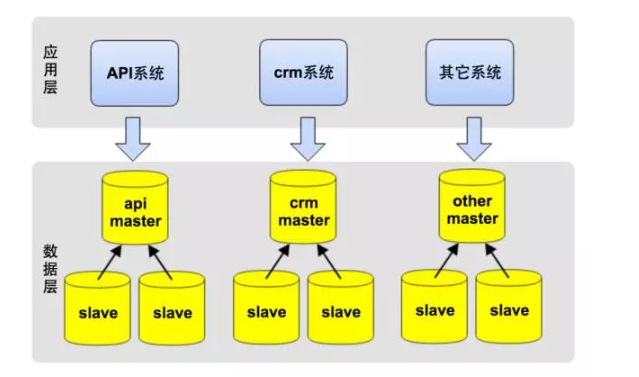
## 升级二



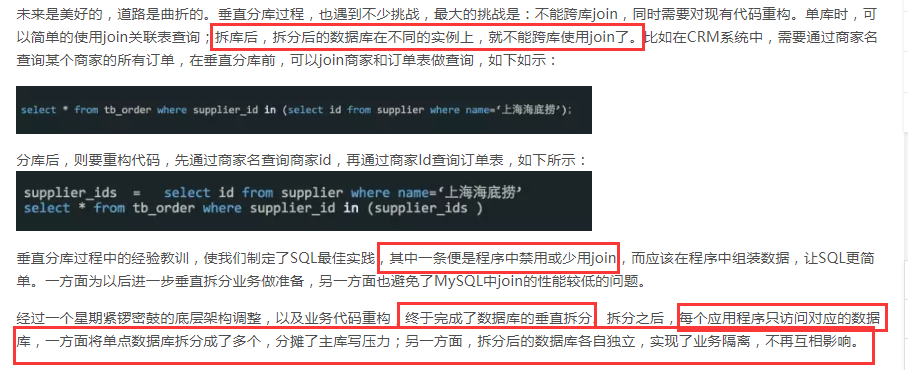
## 问题三



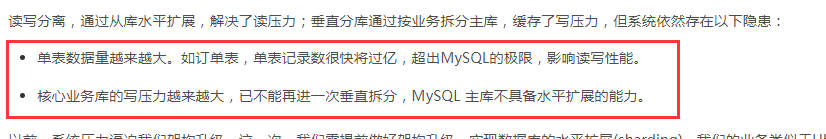
## 升级三

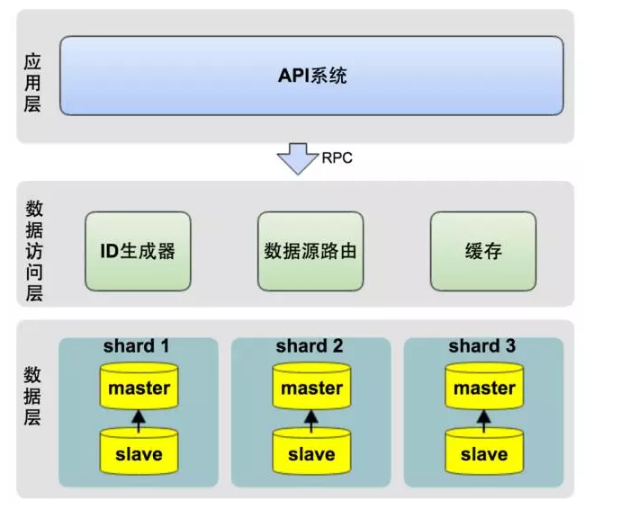


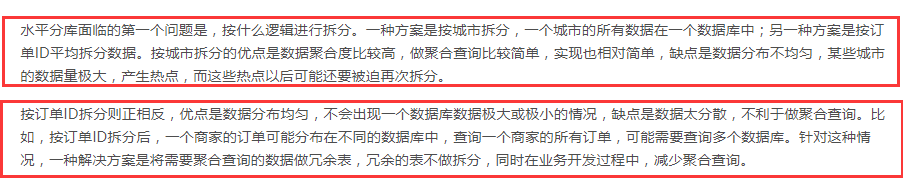
## 问题四

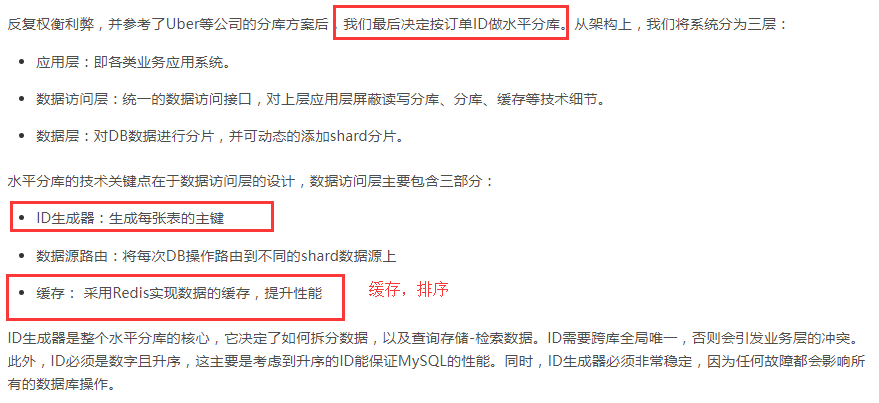


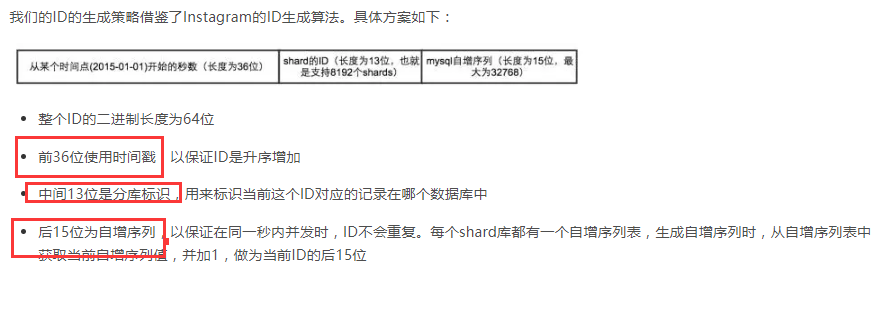
问题五











# 项目1（SpringBoot打造猫眼商城）（等于support-portal）

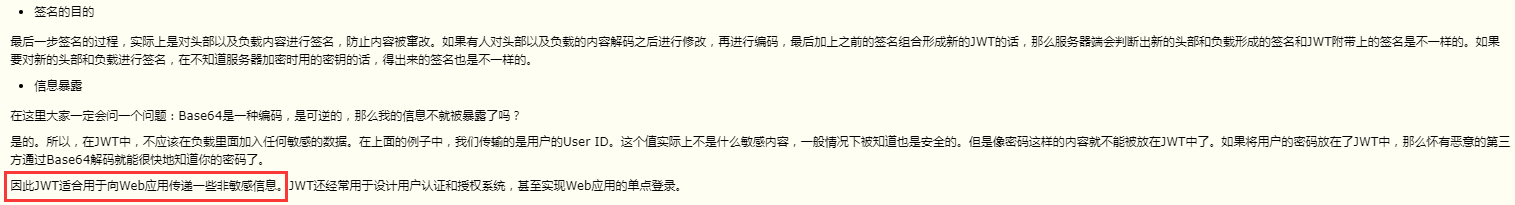
## JWT

SpringMvc+Springboot+mybatis整合开发，前后端分离，用户验证JWT,支付宝对接，复杂SQL翻页排序，引入限流措施，接口设计原则。

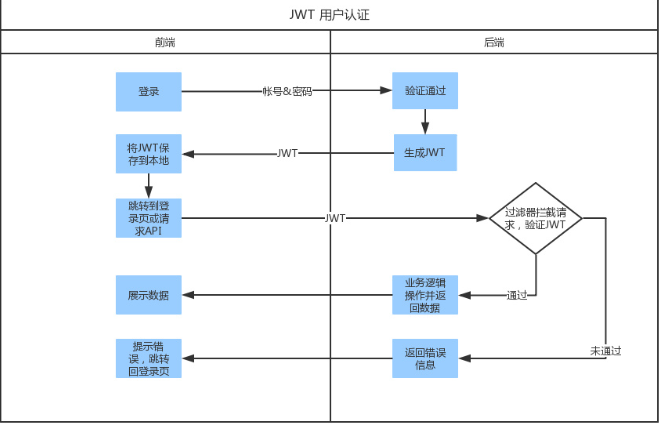
### 用户验证JWT(json web token)







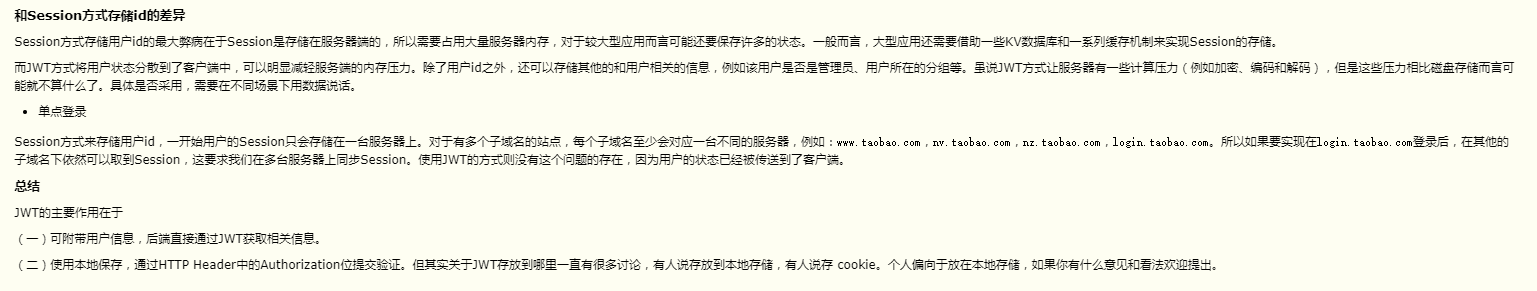
### JWT使用





### 总结

使用 token 加 session，token 和 session 都是临时且唯一的，redis 的 k，v 分别是 token，session



## 防刷和限流措施

技术栈：nginx+lua+redis

### 防刷

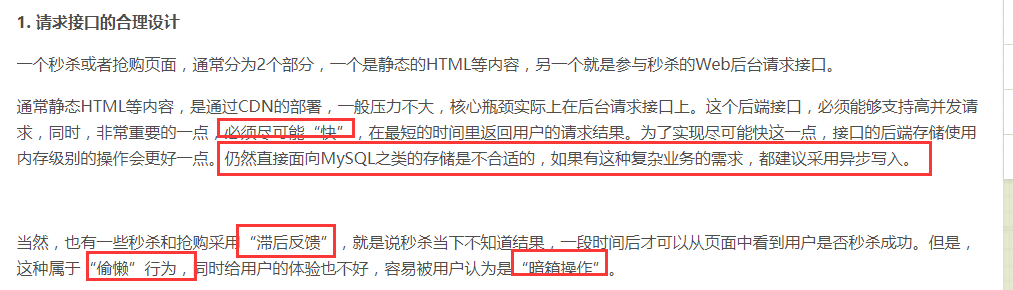
请求过来的时候获取当前访问者的ip，拼接出两个key，一个是用于统计该ip的请求次数的，另一个是用来标识是否被限制了，然后redis里去先去get一下这个ip是否被限制了，如果返回1标识被限制了，直接nginx返回403，否则的话对当前ip进行计数，第一次计数时，计数完毕后设置该计数器的失效时间1秒，后面判断计数器的大小是否查过了200，也就是说，如果在1秒的失效时间内请求了超过200次，那么设置当前ip为受限，并设置受限时间为10分钟。

### 限流

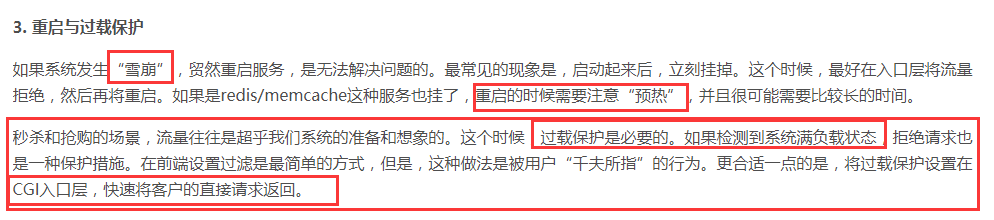
限流的逻辑是这样的请求过来获取你的url然后以这个url作为key去redis里计数，然后判断当前url被请求过多少次，如果是一次那么设置该key的失效时间是1秒，然后有一个while循环，循环的条件是当前url被请求的次数大于10，里面做的事情是让当前这个请求等待1秒，此时前面设置的失效时间应该到了，然后再去对当前url计一次数此时返回的数量理论上市1，如果数量等于1那么设置失效时间为1，然后继续while的循环，如果大于10就让他一直去循环做等待1秒然后设置请求次数加1，如果不大于1秒那么就放行允许请求去访问我们的tomcat，所以这里的逻辑其实限制了并发的请求数量是10，也就是同一时刻只允许10个请求通过校验走到tomcat，后面的请求都在排队等待。

说白了就是10个请求进来后，然后让后面的请求等待1秒然后再放进来10个一直这样循环处理。

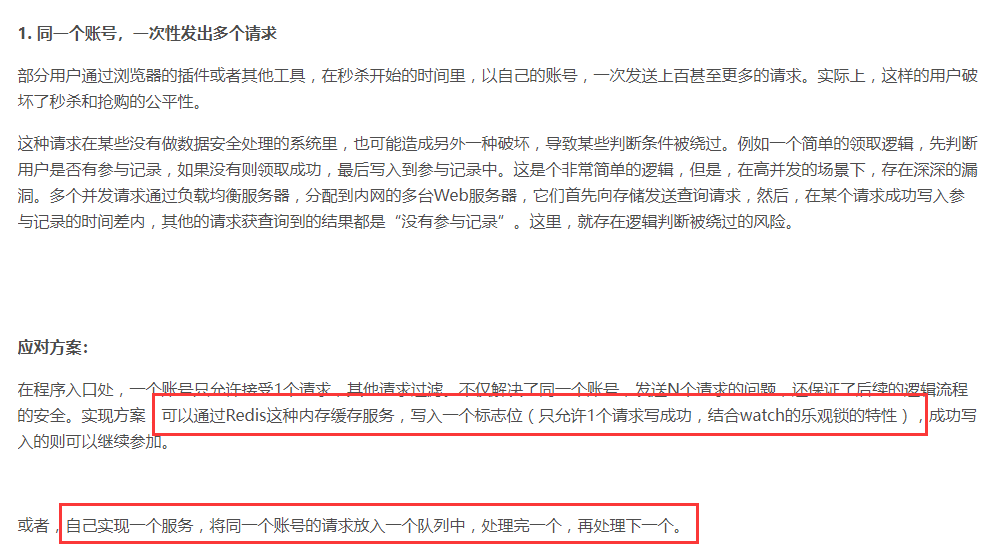
## 电商秒杀和抢购的限流方案

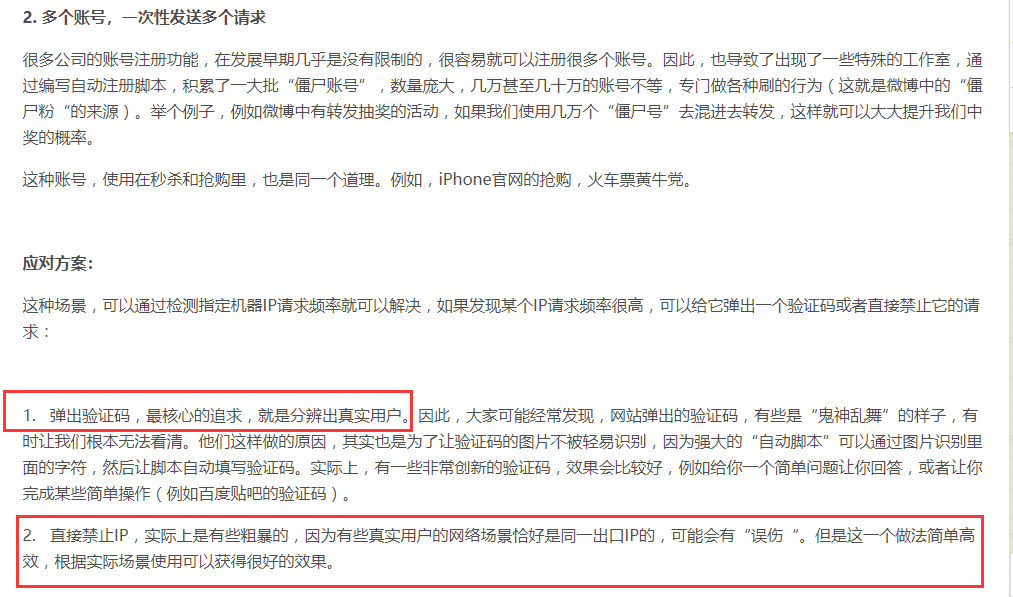


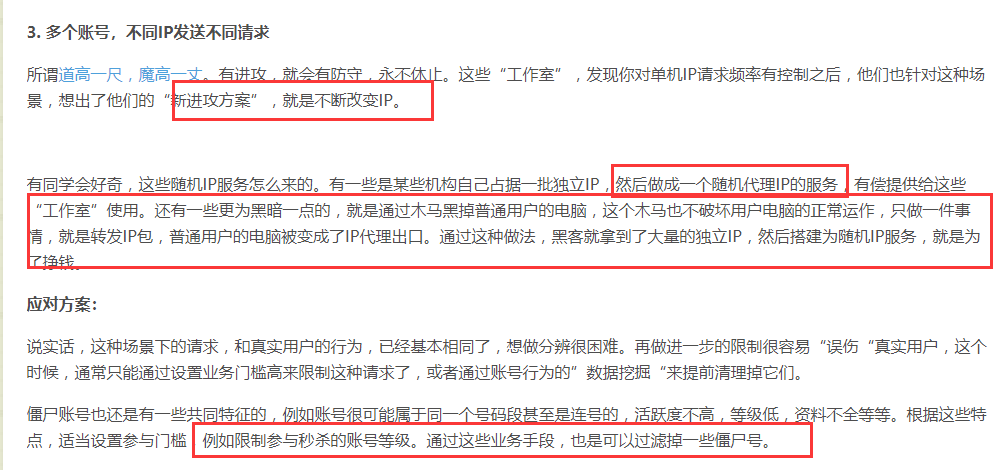


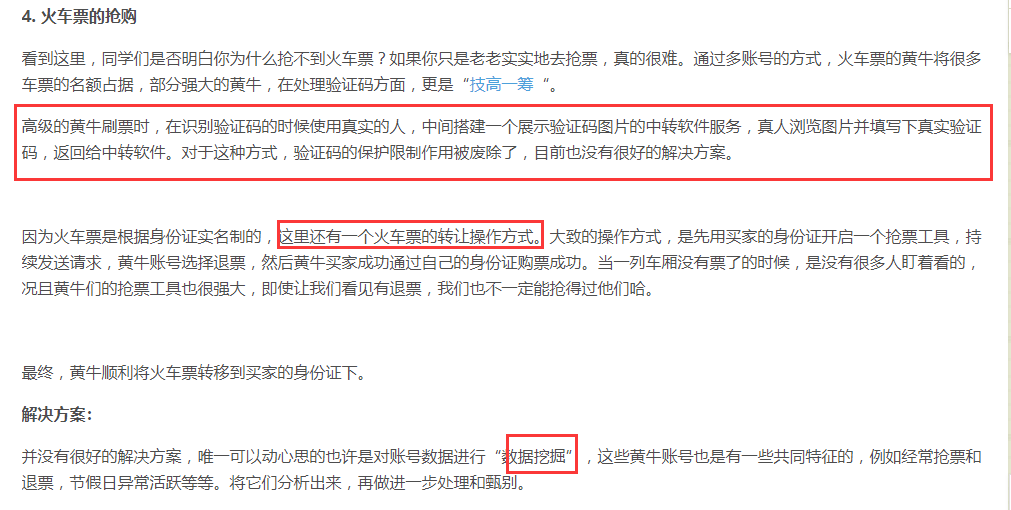


### 进攻与防守

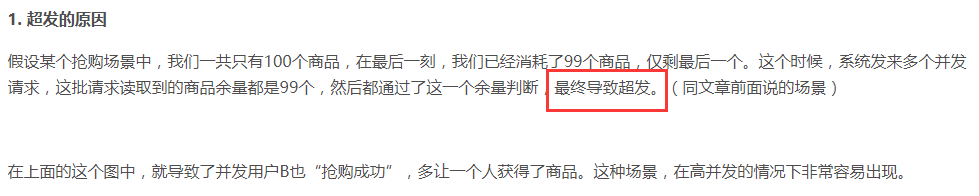


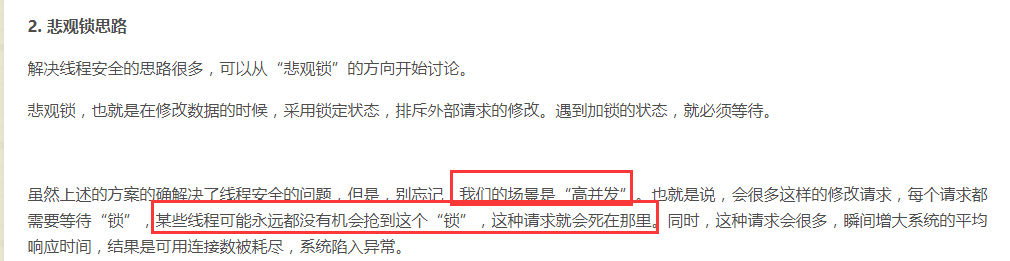


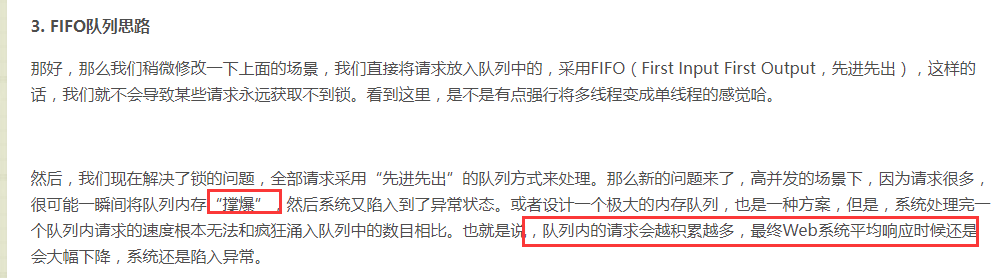


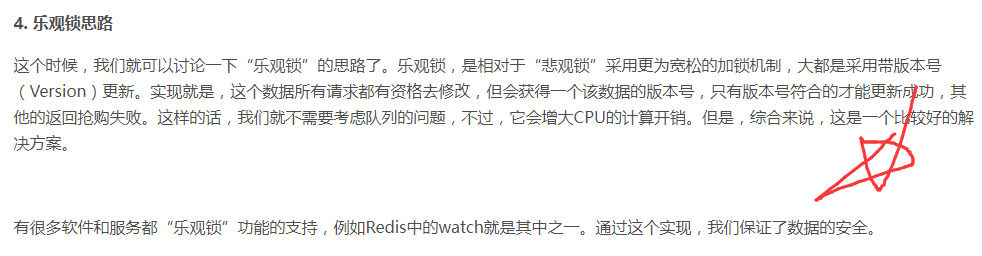


## 数据安全

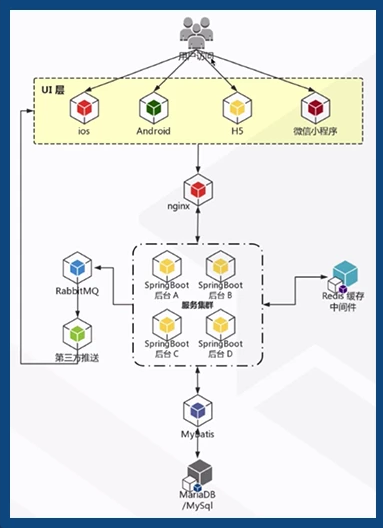








# 项目2（java分布式电影预告小程序后台）

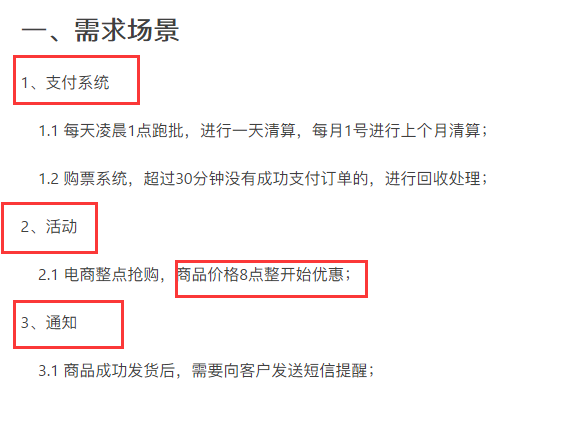


分层，分布式，集群，消息，缓存，高并发

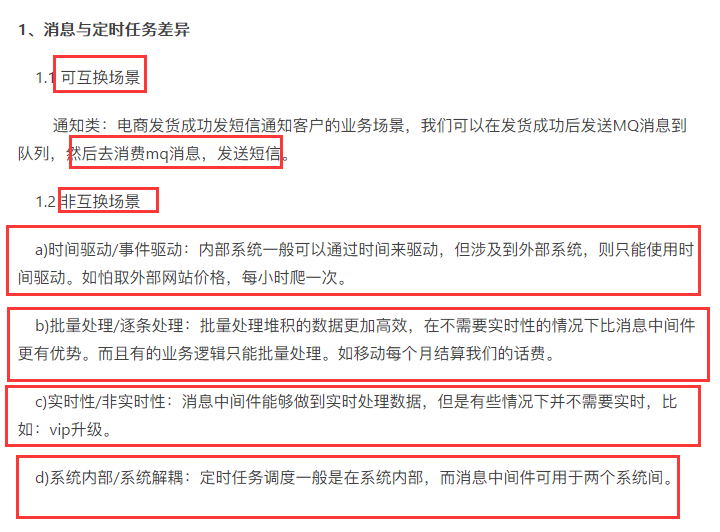
核心技术：Nginx,redis,rabbitmq,分布式session,唯一id主键，分布式任务调度,swagger文档技术，收集端消息推送，微信登录，掌握分布式系统的发布与上线流程

## 分布式任务调度

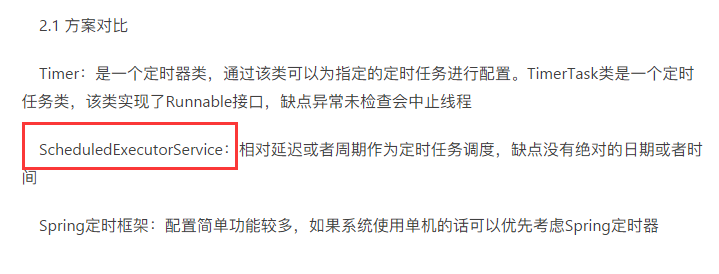
### 需求场景



### 解决方案



### 单机实现方案



### 分布式实现方案

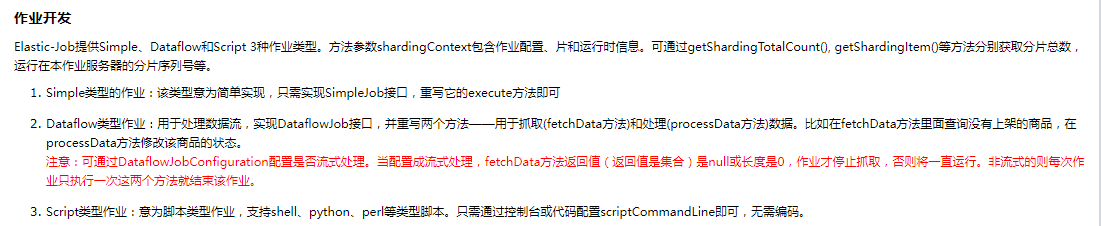
特点： 批量任务，多主机，多线程，动态扩展，实时或定时任务，分片，并发，不重复执行。



### 分布式定时任务调度平台Elastic-job技术详解

<https://blog.csdn.net/adi851270440/article/details/80493367>

Elastic-Job框架可以帮助解决定时任务在集群部署情况下的协调调度问题，保证任务不重复不遗漏的执行。



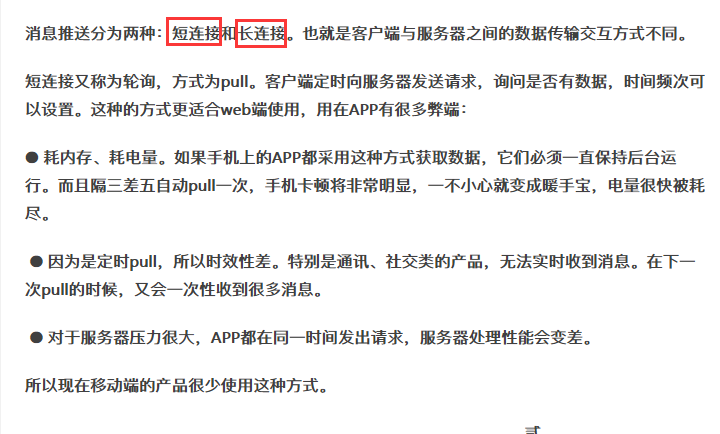
<https://www.cnblogs.com/yushangzuiyue/p/9655847.html>

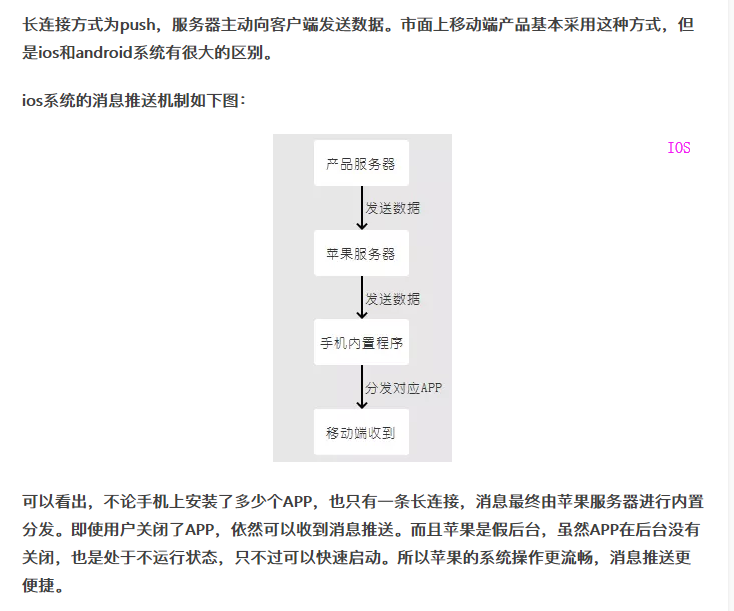
## swagger文档技术

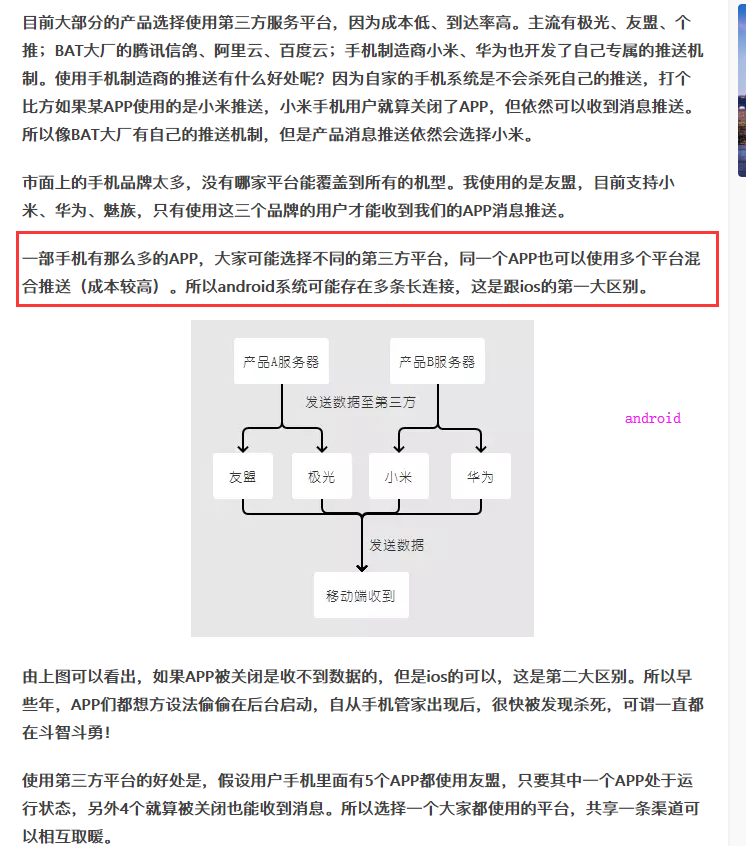
Swagger 是一个规范和完整的框架，用于生成、描述、调用和可视化 RESTful 风格的 Web 服务。总体目标是使客户端和文件系统作为服务器以同样的速度来更新。文件的方法，参数和模型紧密集成到服务器端的代码，允许API来始终保持同步。

作用：  
1. 接口的文档在线自动生成。  
2. 功能测试。

## 手机端消息推送







### 方案

**方案1、**使用GCM服务（Google Cloud Messaging）  
简介：Google推出的云消息服务，即第二代的G2DM。  
优点：Google提供的服务、原生、简单，无需实现和部署服务端。  
缺点：Android版本限制（必须大于2.2版本），该服务在国内不够稳定、需要用户绑定Google帐号，受限于Google。

**方案2、**使用XMPP协议（Openfire + Spark + Smack）  
简介：基于XML协议的通讯协议，前身是Jabber，目前已由IETF国际标准化组织完成了标准化工作。  
优点：协议成熟、强大、可扩展性强、目前主要应用于许多聊天系统中，且已有开源的Java版的开发实例androidpn。  
缺点：协议较复杂、冗余（基于XML）、费流量、费电，部署硬件成本高。

**方案3、**使用MQTT协议（更多信息见：[MQTT: MQ Telemetry Transport](https://links.jianshu.com/go?to=https%3A%2F%2Flink.zhihu.com%2F%3Ftarget%3Dhttp%253A%2F%2Fmqtt.org%2F)）  
简介：轻量级的、基于代理的“发布/订阅”模式的消息传输协议。  
优点：协议简洁、小巧、可扩展性强、省流量、省电，目前已经应用到企业领域（参考：[Software - runtimes, APIs and libraries for MQTT](https://links.jianshu.com/go?to=https%3A%2F%2Flink.zhihu.com%2F%3Ftarget%3Dhttp%253A%2F%2Fmqtt.org%2Fsoftware)），且已有C++版的服务端组件rsmb。  
缺点：不够成熟、实现较复杂、服务端组件rsmb不开源，部署硬件成本较高。

**方案4、**使用HTTP轮循方式  
简介：定时向HTTP服务端接口（Web Service API）获取最新消息。  
优点：实现简单、可控性强，部署硬件成本低。  
缺点：实时性差。

## 分布式系统的发布与上线流程

# 项目3（商城项目，高并发系统技术分析）

前后端分离，springboot+dubbo微服务，zookeeper分布式锁，基于Redis+Mongdb+Rabbitmq秒杀实战，Fastdfs文件系统,Nginx分流分压，Mysql(读写分离)，docker容器实战，微信支付

<https://www.bilibili.com/video/av32170669?from=search&seid=15422156380335135410>

# 调优经历1（线程池调优经历）

最近的一个项目需要用到招标，临时加了给我们的系统增加了一个性能需求，多少呢？

一秒钟300次NTP（不知道ntp的同学可以百度一下），平均3ms一次啊，没测试过，心里没有底。(⊙o⊙)…

## 情景介绍

系统是一个时间服务器系统，客户端就是window系统，或者其他的一些服务器，来向时间服务器同步时间。

## 第一次测试

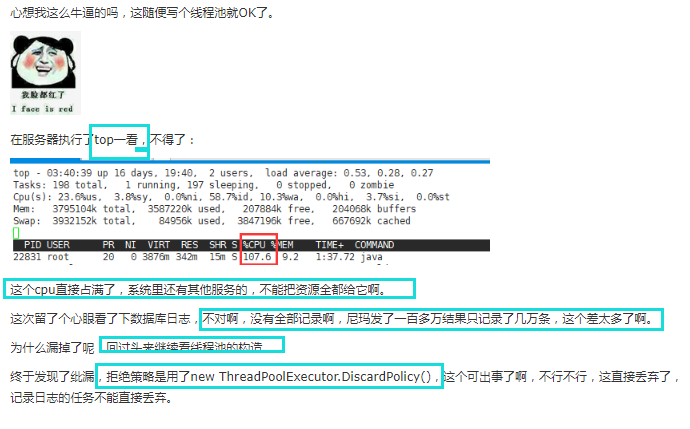


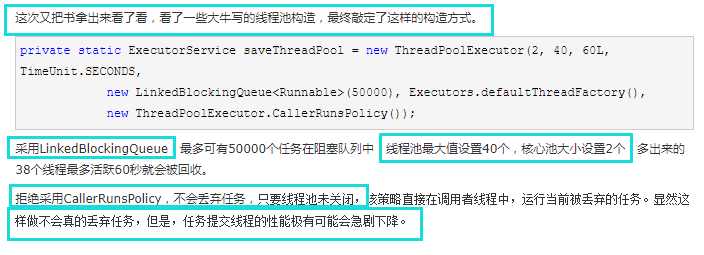
## 第二次测试



## 第三次测试









# jvm调优

## 经验概述1

1、初始化内存和最大内存尽量保持一致，避免内存不够用继续扩充内存。最大内存不要超过物理内存，例如内存8g，你可以设置最大内存4g/6g但是不能超过8g否则加载类的时候没有空间会报错。

2、gc/full gc频率不要太高、每次gc时间不要太长、根据系统应用来定。

# 经验概述2

### 1堆信息查看







### 2线程监控

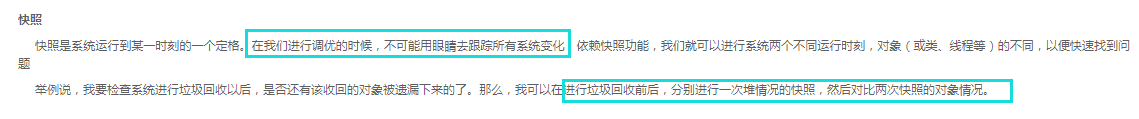




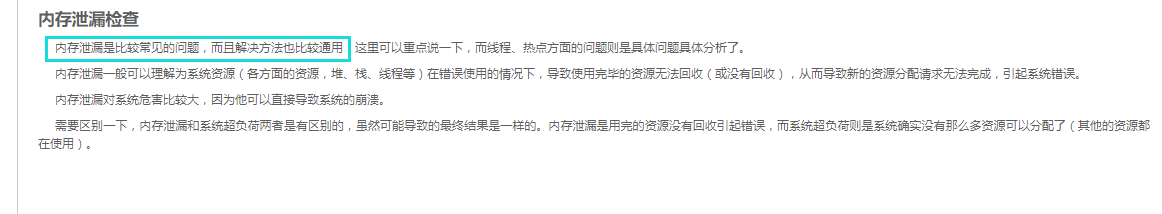
### 3．热点分析



### 4快照

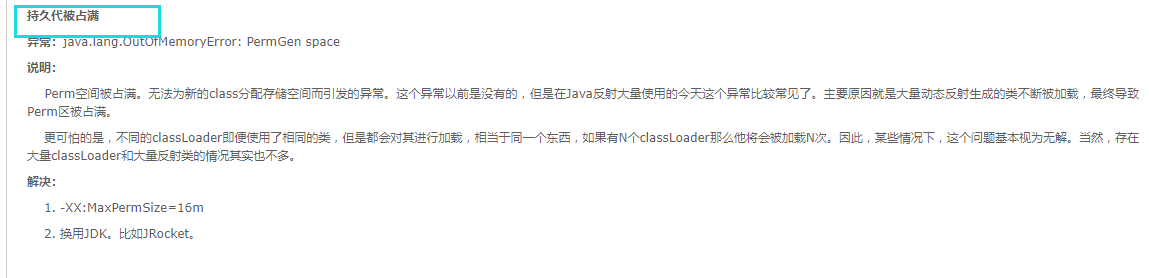


### 5内存泄漏

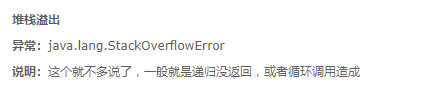




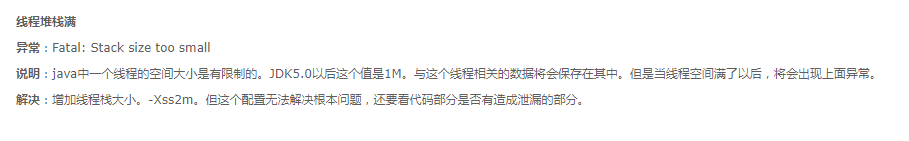
### 6持久代被占满



### 7堆栈溢出



### 8线程堆栈满



### 9系统内存被占满

