1.论秒杀难点，透过现象看本质

2.悲观锁与乐观锁的大比拼

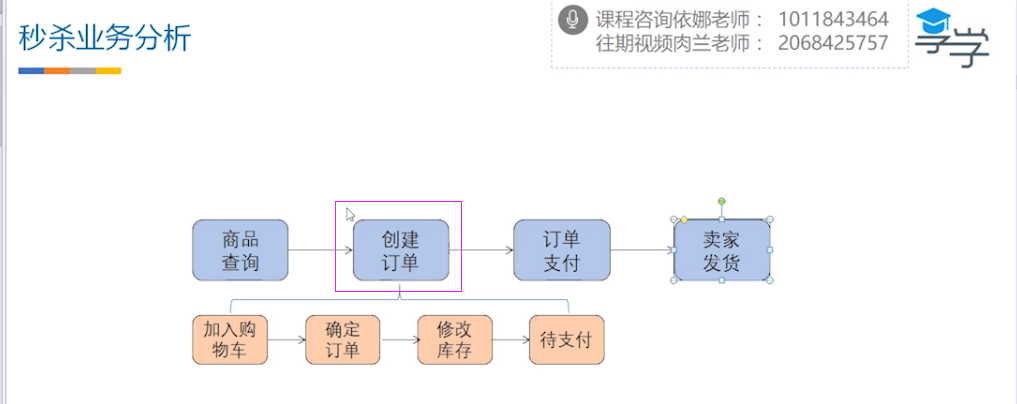
3.巧写代码衡量系统吞吐量

4.秒杀核心业务如何落地

5.架构师眼中的秒杀系统全貌

# 秒杀核心业务分析

## 流程



秒杀系统最核心的服务修改库存

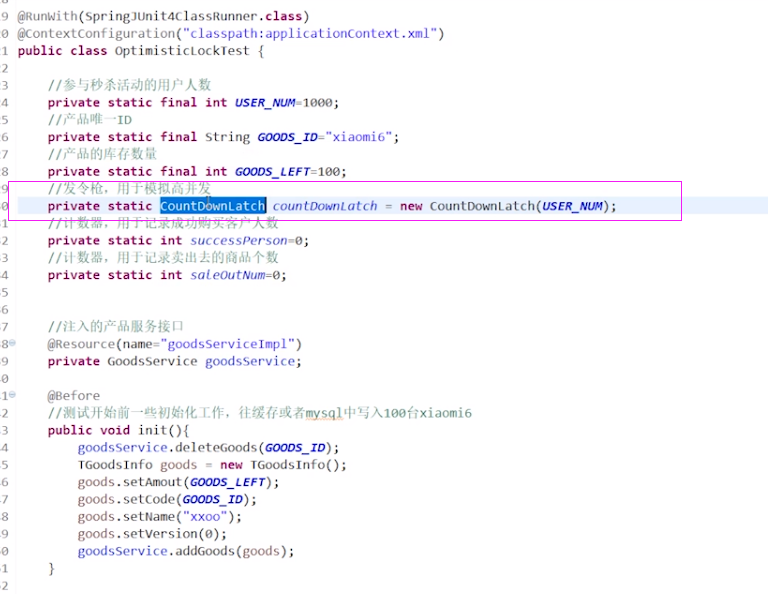
## 哪些场景属于秒杀业务





## 模拟高并发

用countDown来准备线程（模拟1000个人来抢购）（当countDown计数到0时所有停止的线程开始抢购）







上面高并发产生的问题（多卖了）

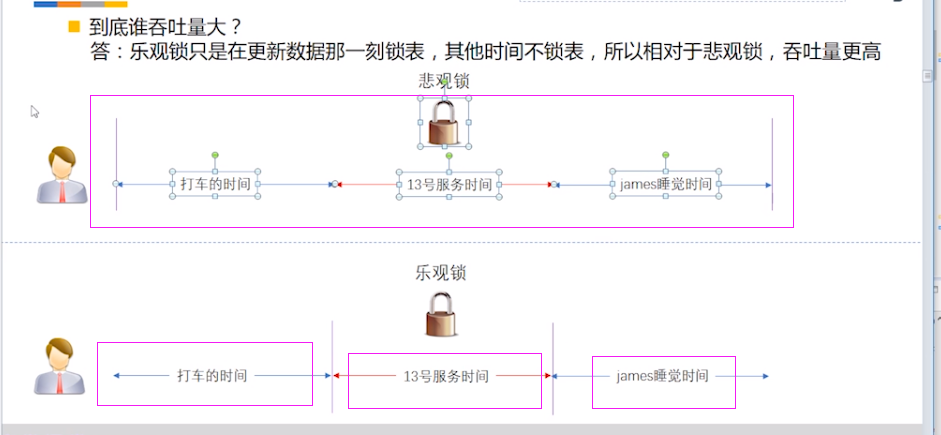
解决方案在方法上加了syn锁（锁的力度太大，变成单并发），但是牺牲了高并发

# 关于锁的那些事



当你使用事务的时候就有个锁的竞争机制（事物的隔离级别）。

## 乐观锁和悲观锁到底谁的吞吐量大（乐观锁）



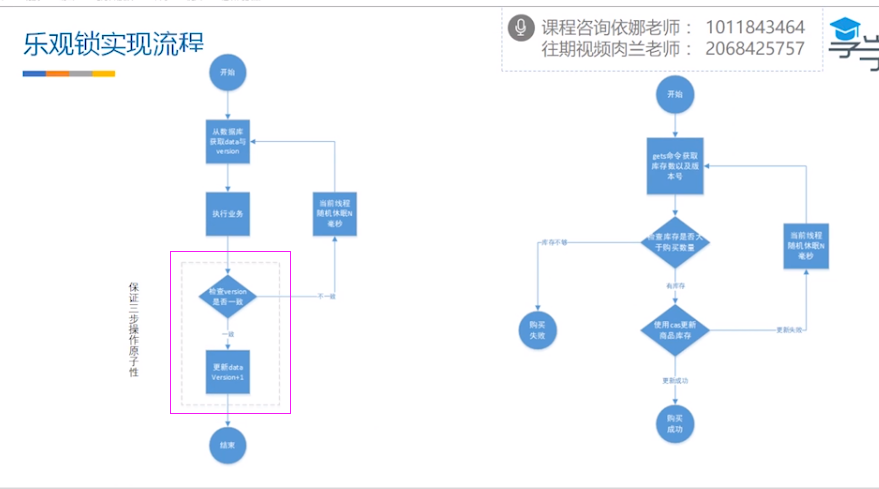
## 数据说明乐观锁的吞吐量比悲观锁大

synchronized是悲观锁

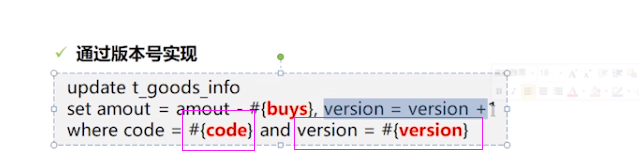


# 秒杀核心服务实战

## 一乐观锁实现（通过版本号实现,缺点是加了个字段）



基于数据库版本号实现乐观锁

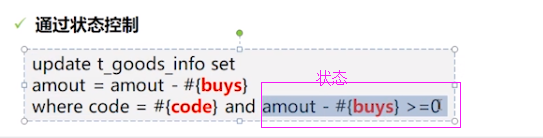


### 代码实现





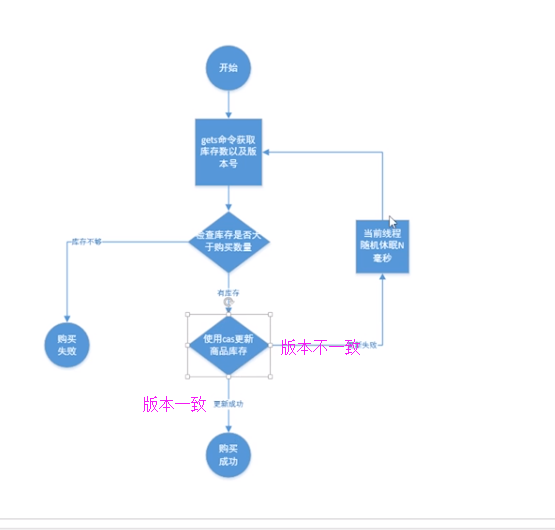
## 二乐观锁实现（通过状态控制，有0.3的提升）

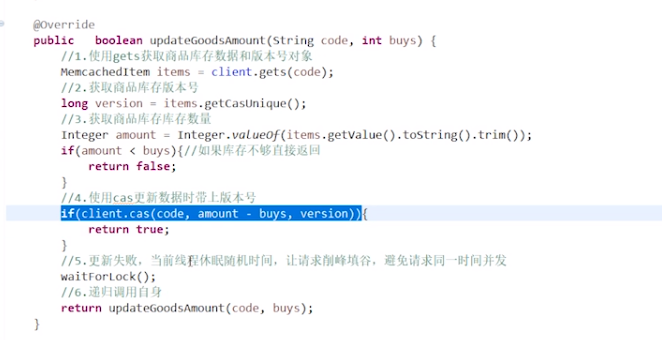


基于数据库乐观锁实现的优点在于简单高效，稳定可靠，缺点在于并发能力低（数据库的瓶颈值300（机器）-700（固态））

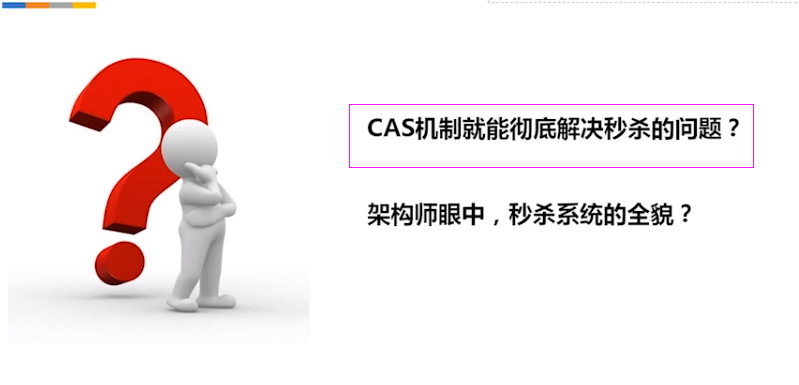
## 三基于memcached的cas机制实现（时间缩短了到1/3，支持10万差不多了）

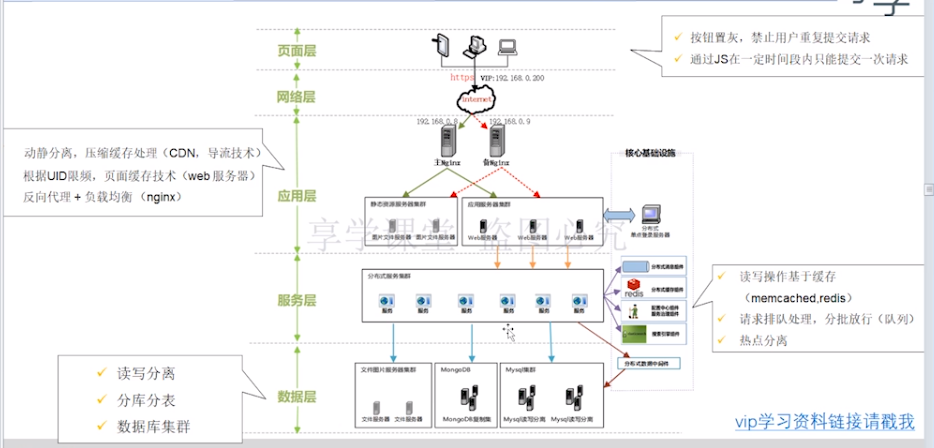






# 秒杀系统架构设计





分这么多层的好处，把压力拦截在上游（能走缓存坚决不进数据库），热点隔离（分时段售票）把热点业务分离出来。

页面层：按钮置灰，禁止用户重复提交请求，通过js在一定时间内只能提交一次请求。

nginx:动静分离，压缩缓存处理（CDN,倒流技术），根据UID限频（在10秒内只让你一个请求有效），页面缓存技术（web服务器），反向代理+负载均衡。

读写操作基于缓存，请求排队处理，分批放行，热点分离，削峰（使用消息中间间过一段时间告诉你结果）

数据库层，读写分离，分库分表，数据库集群。

互联网数据最宝贵

# 课程总结

# 产生的问题

栈溢出（错峰执行）