压力测试&性能基本分析方法&服务端优化实现

今日课程主题：

1、认识项目整体结构（后端系统，前端系统）

2、服务器环境（4 台服务器，一台阿里云测试服务器）---- 服务器中相关服务配套设施

3、服务上线部署（阿里云平台上进行部署）--- 传统的部署模式

4、压力测试（商品详情页接口测试： 查询） --- 认识高并发相关的性能参数，评定的指标

5、性能瓶颈问题分析

6、服务端性能调优 --- 压力测试验证调优的效果

7、jvm 调优相关原理

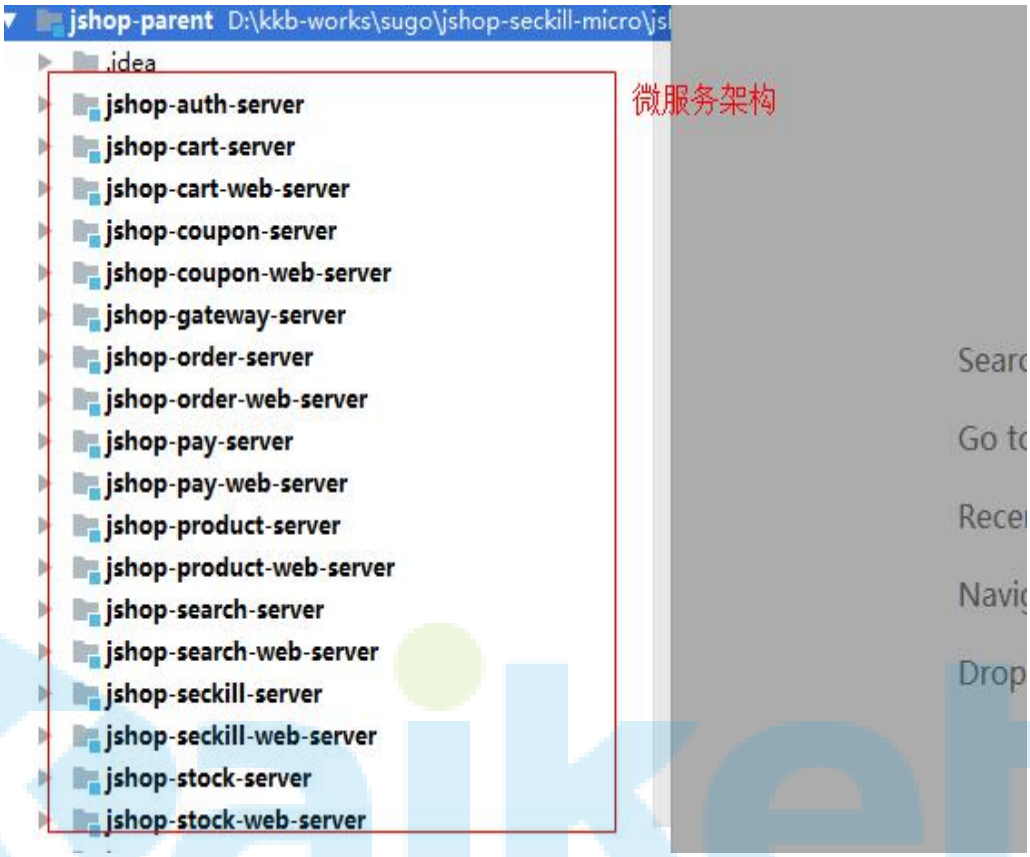
1 项目整体结构

1.1 前端系统

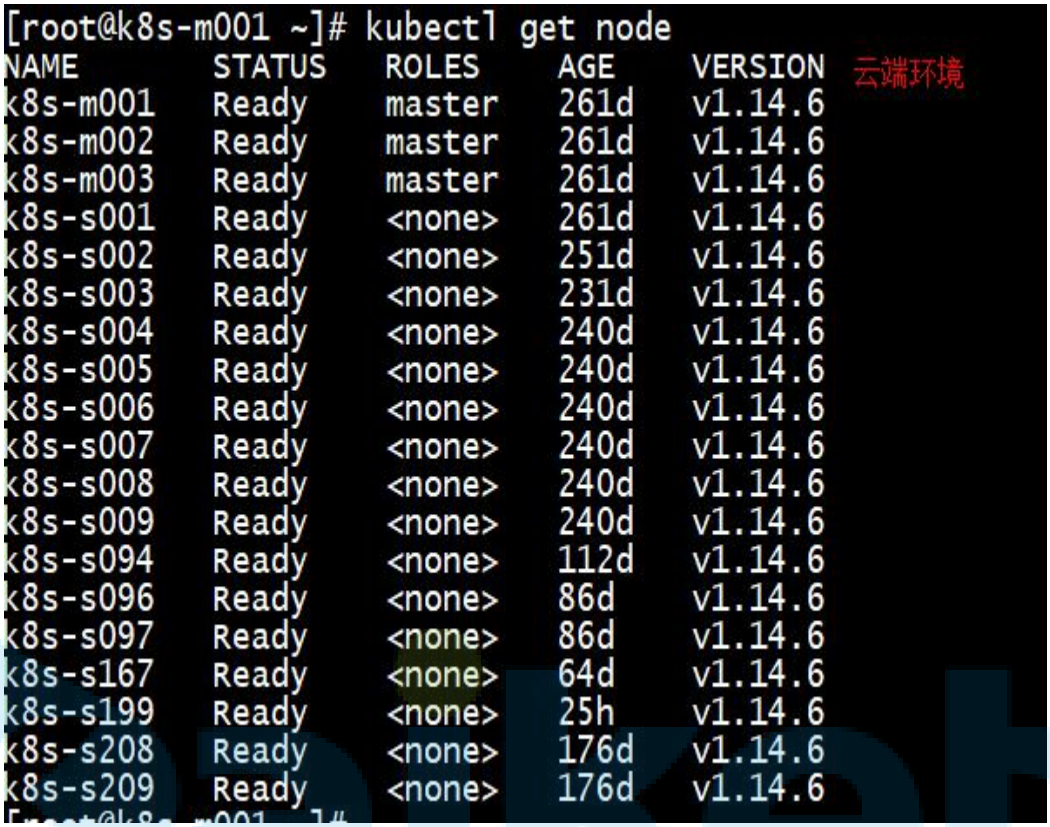
单体架构开始，再到微服务架构，然后还要上云（云架构）

1）单体架构

2）微服务架构


jshop-parent D:\kkb-works\sugo\jshop-seckill-micro\js
.idea
jshop-auth-server
jshop-cart-server
jshop-cart-web-server
jshop-coupon-server
jshop-coupon-web-server
jshop-gateway-server
jshop-order-server
jshop-order-web-server
jshop-pay-server
jshop-pay-web-server
jshop-product-server
jshop-product-web-server
jshop-search-server
jshop-search-web-server
jshop-seckill-server
jshop-seckill-web-server
jshop-stock-server
jshop-stock-web-server
微服务架构

3）云端迁移部署


[root@k8s-m001 ~]# kubectl get node
NAME STATUS ROLES AGE VERSION
k8s-m001 Ready master 261d v1.14.6
k8s-m002 Ready master 261d v1.14.6
k8s-m003 Ready master 261d v1.14.6
k8s-s001 Ready <none> 261d v1.14.6
k8s-s002 Ready <none> 251d v1.14.6
k8s-s003 Ready <none> 231d v1.14.6
k8s-s004 Ready <none> 240d v1.14.6
k8s-s005 Ready <none> 240d v1.14.6
k8s-s006 Ready <none> 240d v1.14.6
k8s-s007 Ready <none> 240d v1.14.6
k8s-s008 Ready <none> 240d v1.14.6
k8s-s009 Ready <none> 240d v1.14.6
k8s-s094 Ready <none> 112d v1.14.6
k8s-s096 Ready <none> 86d v1.14.6
k8s-s097 Ready <none> 86d v1.14.6
k8s-s167 Ready <none> 64d v1.14.6
k8s-s199 Ready <none> 25h v1.14.6
k8s-s208 Ready <none> 176d v1.14.6
k8s-s209 Ready <none> 176d v1.14.6
云端环境

4）前端系统效果

化歌儿Wacoal内衣品牌矩献

1.5 折起 | 部分商品折上4.8折起

回南天家电除湿除菌回南天家电除湿除菌

1.3 折起

耐克Nike男女运动专场耐克Nike男女运动专场

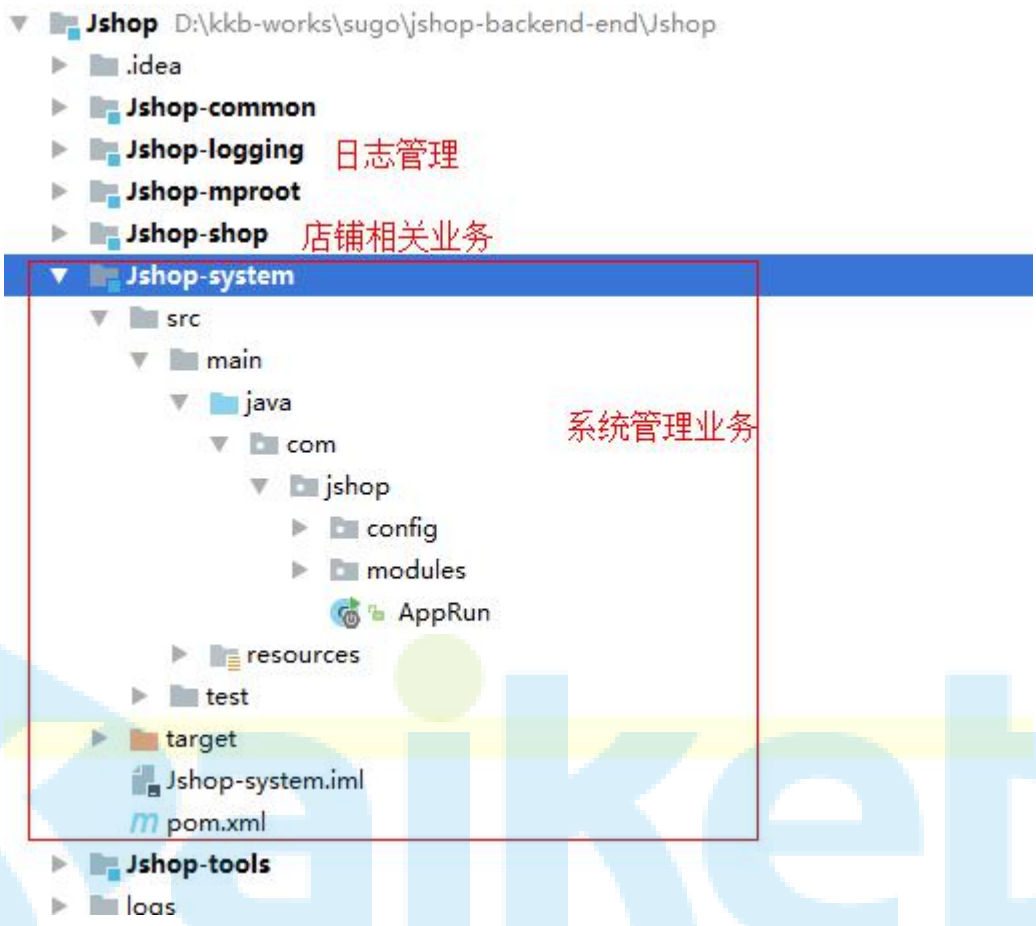
2.6 折起 | 部分商品折上5.8折起

雅莹集团女装新春特惠专场雅莹集团女装新春特惠专场

0.5 折起

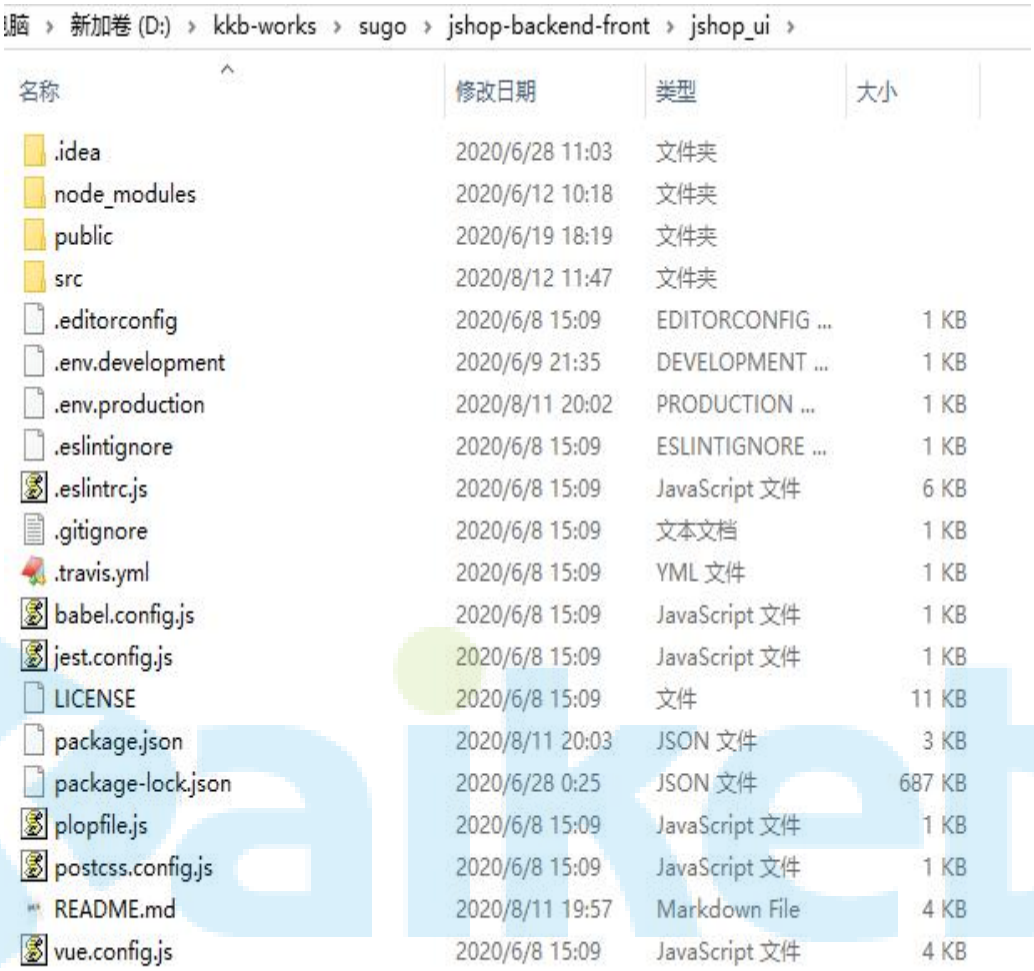
1.2 后端系统

1) 后端代码



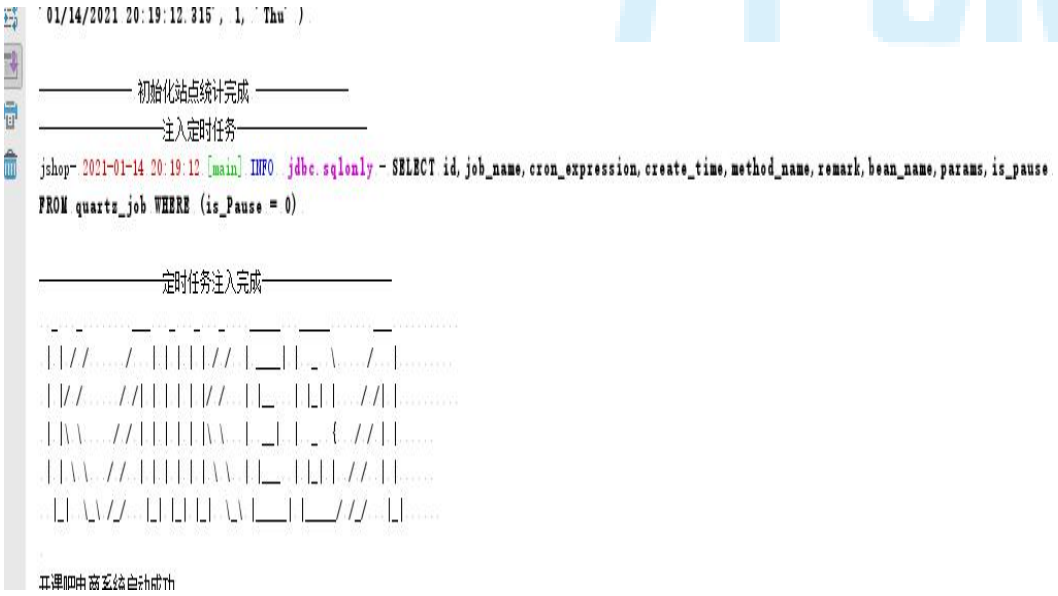
2）前端代码

前端系统： 使用 nodejs 作为服务器来运行，运行命令： npm start ,npm run dev



名称	修改日期	类型	大小
.idea	2020/6/28 11:03	文件夹	
node_modules	2020/6/12 10:18	文件夹	
public	2020/6/19 18:19	文件夹	
src	2020/8/12 11:47	文件夹	
.editorconfig	2020/6/8 15:09	EDITORCONFIG ...	1 KB
.env.development	2020/6/9 21:35	DEVELOPMENT ...	1 KB
.env.production	2020/8/11 20:02	PRODUCTION ...	1 KB
.eslintignore	2020/6/8 15:09	ESLINTIGNORE ...	1 KB
.eslintrc.js	2020/6/8 15:09	JavaScript 文件	6 KB
.gitignore	2020/6/8 15:09	文本文档	1 KB
.travis.yml	2020/6/8 15:09	YML 文件	1 KB
babel.config.js	2020/6/8 15:09	JavaScript 文件	1 KB
jest.config.js	2020/6/8 15:09	JavaScript 文件	1 KB
LICENSE	2020/6/8 15:09	文件	11 KB
package.json	2020/8/11 20:03	JSON 文件	3 KB
package-lock.json	2020/6/28 0:25	JSON 文件	687 KB
plopfile.js	2020/6/8 15:09	JavaScript 文件	1 KB
postcss.config.js	2020/6/8 15:09	JavaScript 文件	1 KB
README.md	2020/8/11 19:57	Markdown File	4 KB
vue.config.js	2020/6/8 15:09	JavaScript 文件	4 KB

查看运行效果：

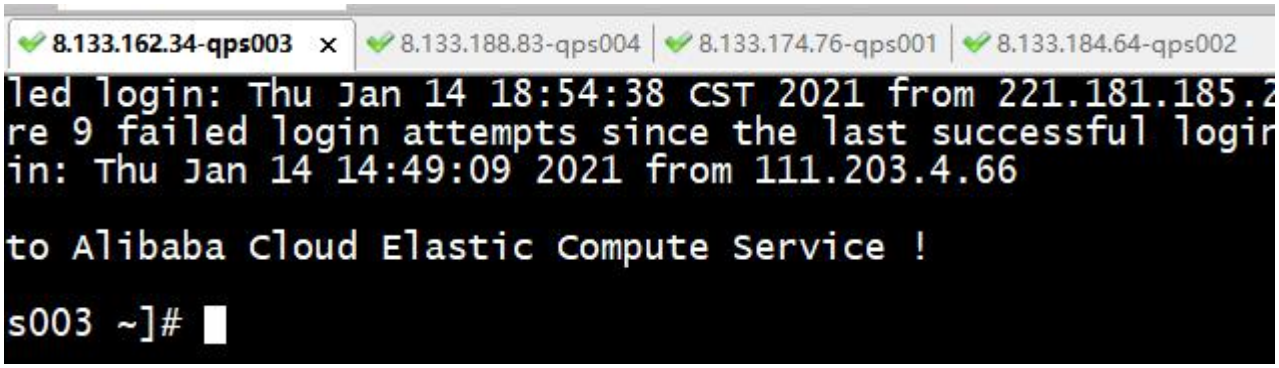


```
'01/14/2021 20:19:12.815', 1, 'Thu' )

————————— 初始化站点统计完成 —————————
————————————注入定时任务————————————
jshop- 2021-01-14 20:19:12 [main] INFO  jdbc.sqlonly - SELECT id,job_name,cron_expression,create_time,method_name,remark,bean_name,params,is_pause
FROM quartz_job WHERE (is_Pause = 0)

————————————定时任务注入完成————————————
```

2 服务器环境

2.1 阿里云环境

阿里云服务器： 4 台服务器



阿里云：1 台测试服务器 4cpu，8GB --- 阿里云 windows 镜像服务---构建内网测试环境

云端环境：

```
[root@k8s-m001 ~]# kubectl get node
NAME       STATUS   ROLES    AGE    VERSION
k8s-m001   Ready    master   261d   v1.14.6
k8s-m002   Ready    master   261d   v1.14.6
k8s-m003   Ready    master   261d   v1.14.6
k8s-s001   Ready    <none>   261d   v1.14.6
k8s-s002   Ready    <none>   251d   v1.14.6
k8s-s003   Ready    <none>   231d   v1.14.6
k8s-s004   Ready    <none>   240d   v1.14.6
k8s-s005   Ready    <none>   240d   v1.14.6
k8s-s006   Ready    <none>   240d   v1.14.6
k8s-s007   Ready    <none>   240d   v1.14.6
k8s-s008   Ready    <none>   240d   v1.14.6
k8s-s009   Ready    <none>   240d   v1.14.6
k8s-s094   Ready    <none>   112d   v1.14.6
k8s-s096   Ready    <none>   86d    v1.14.6
k8s-s097   Ready    <none>   86d    v1.14.6
k8s-s167   Ready    <none>   64d    v1.14.6
k8s-s199   Ready    <none>   25h    v1.14.6
k8s-s208   Ready    <none>   176d   v1.14.6
k8s-s209   Ready    <none>   176d   v1.14.6
```

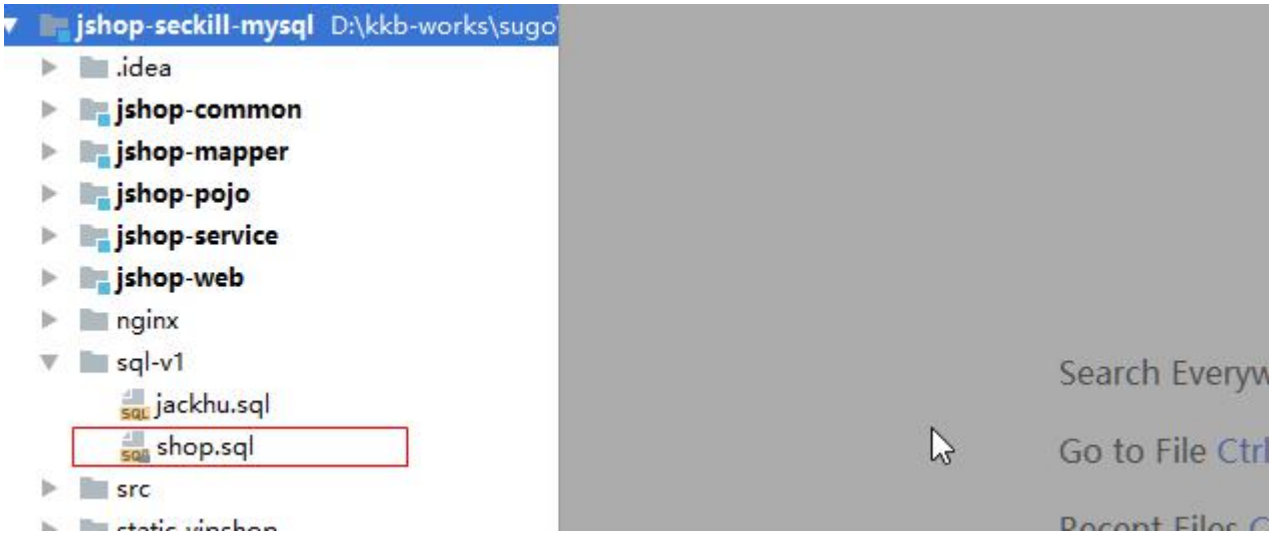
2.2 相关服务

服务器安装对应配套服务：JDK,MYSQL,Redis,RocketMQ,openresty,zookeeper

3 服务上线部署

3.1 数据库环境

数据库脚本已经上传到码云上面，直接下载即可；（在代码目录下）



导入 SQL 脚本：

导入指令： mysql -uroot -proot < shop.sql # 数据库登录后，执行 SQL 脚本导入工作

```
Database changed
mysql> show tables;
+---------------------------+
| Tables_in_shop            |
+---------------------------+
| alipay_config             |
| article                   |
| column_config             |
| dept                      |
| dict                      |
| dict_detail               |
| email_config              |
| express                   |
| front_user                |
| gen_config                |
| gen_test                  |
| job                       |
| local_storage             |
| log                       |
| material                  |
| material_group            |
| menu                      |
| monitor_server            |
| picture                   |
| qiniu_config              |
| qiniu_content             |
| quartz_job                |
| quartz_log                |
```

3.2 服务打包

注意：服务进行部署的时候，没有使用 jenkins 实现自动化的打包方式，使用手动的方式打包模式；
项目打包： 必须导入如下的插件，否则项目打包的时候，不会把项目依赖包打包到当前的项目中，这样的话此项目就无法运行；

```
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
```

3.3 服务部署

后端服务启动指令：

```
nohup java -jar jshop-web-1.0-SNAPSHOT.jar --spring.config.addition-location=application.yaml > jshop.log 2>&1 &
```

注意：为了部署方便，不用每次都修改配置文件，（本地开发，测试，服务器部署），使用外挂的配置文件，外挂配置文件加载模式：
--spring.config.addition-location=application.yaml

部署成功：访问商品详情的接口效果图

4 压力测试

压力测试： 及时发现系统问题，系统瓶颈（预期目标），并及时对系统进行改进，对系统进

行性能优化（BUG 修复，性能提升），因此压力测试在工程开发中是非常有必要；
架构师： 掌握一定的压力的方法，压力测试是保障软件高质量交付的必备条件之一；
压力测试—性能测试： 压力测试主要是模拟大量用户来测试系统在高负载情况下，系统的响应时间，系统问题，吞吐量等指标是否满足性能的需求；

4.1 压力测试维度

1) 负载测试

确定并确保系统在超出最大预期工作量的情况下任然能正常运行；评估系统的性能，RT,TPS, 采用梯形测试方法，逐渐加大测试量

评定系统最大容量，找出系统的拐点；

例如：200 斤坚持多少时间，是否能坚持 10 分钟

2）强度测试

使得服务器一直处于极限状态（满负荷），测试系统在极限状态下运行的是否稳定； 测试指标：RT,TPS CPU

3）容量测试

确定系统可以同时在最大用户数量

4.2 Jmeter 工具

测试工具：

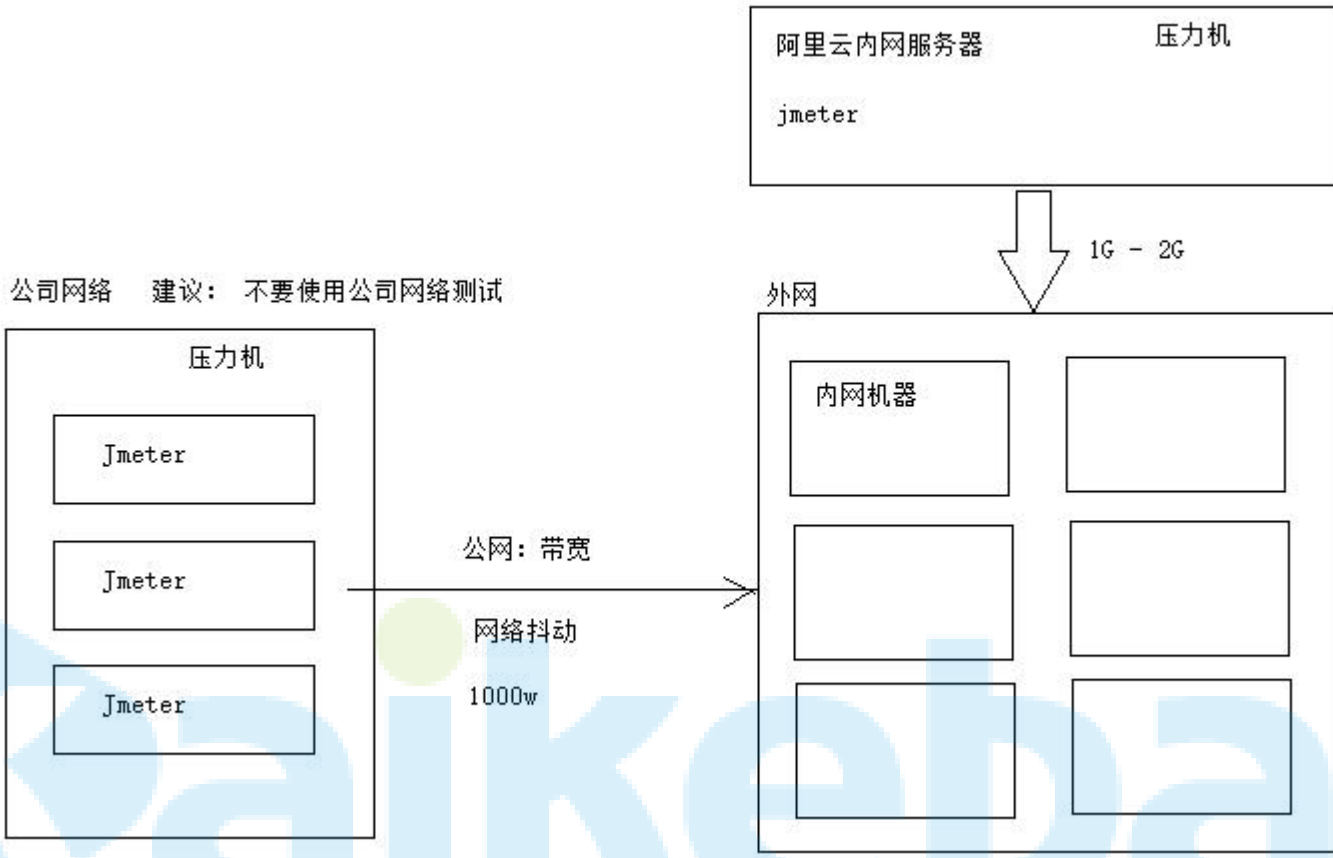
1、AB 测试工具

2、ngrinter 韩国研发测试工具

3、阿里云测试服务

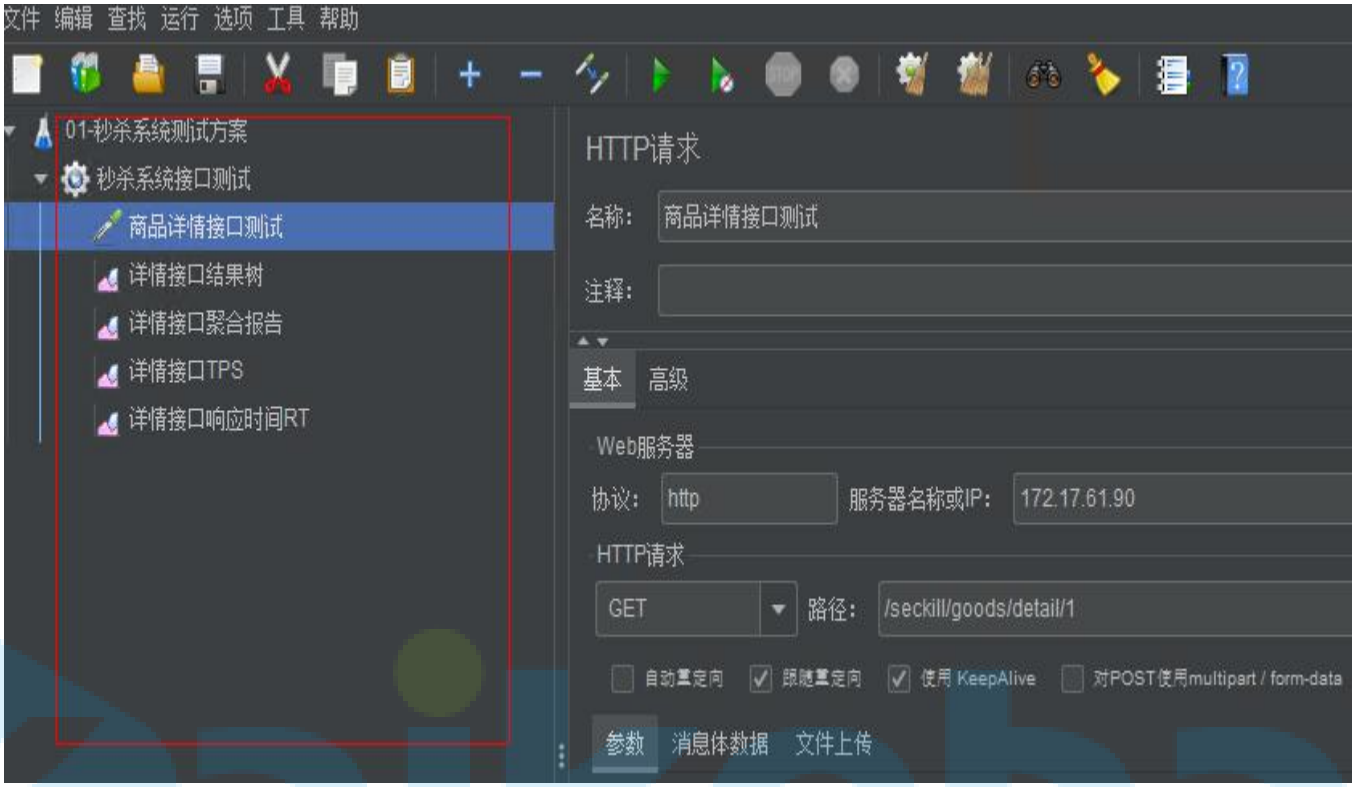
4、jmeter 测试简单，可视化界面

注意：测试容易受到网络抖动干扰，服务硬件配置的环境影响，因此在压力测试下，都应该在内网进行测试，防止网络抖动，带宽的限制；这些限制都会导致测试结果不准确；



4.3 开始测试

创建商品详情接口测试计划：



开始测试：查看 jmeter 测试报告一些性能指标图示

1）TPS 性能曲线图 （吞吐能力）

通过 10w 个样本测试，发现商品详情页接口性能稳定在 5400 TPS ,但是注意：此时此刻使用的是一个不耗时的操作，数据库主键查询；

2）Response Time

测试商品详情接口响应时间曲线图，可以观察接口在压力测试下，RT 时间，以便于对系统进行调优；

3）聚合报告

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
商品详情...	100000	385	357	573	642	672	40	731	0.00%	4310.7/
总体	100000	385	357	573	642	672	40	731	0.00%	4310.7/

测试的样本数： 10w
平均响应时间： 385ms
中位数： 50%请求的响应时间 ，表示 50%的请求在 357ms 之内响应结束
90%百分位： 90%的请求在 573ms 之内响应结束
最小值： 一个请求响应的最小时间 是 40ms

4.4 并发线程测试

5 性能分析常用方法

系统出现问题分类：

1、系统异常：

CPU 占用率过高，磁盘满了，磁盘 IO 频繁，网络流量异常,通过指令进行排查： top, free, dstat ,pstack,vmstat,strace 获取系统异常数据；

可视化工具： promethues , zabbix

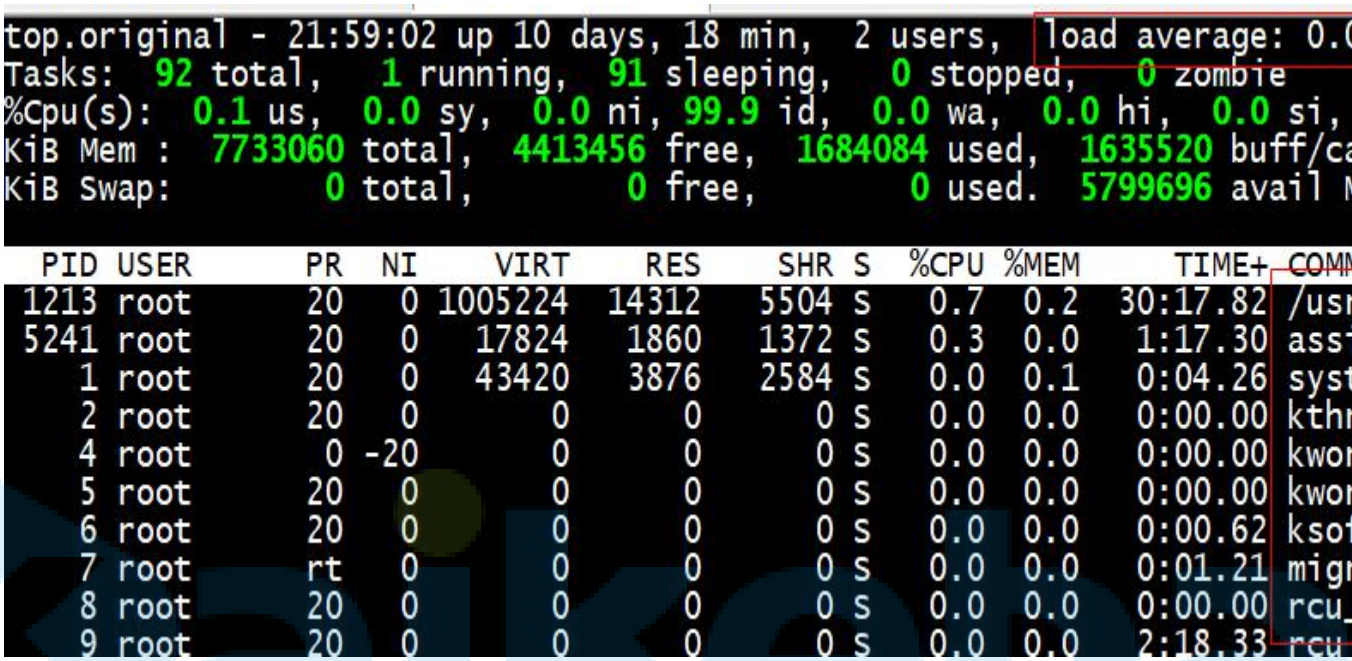
2、业务异常

流量太多系统扛不住，耗时长，线程死锁，多线程并发，频繁 full gc, oom ，排查方式：top,jstack,pstack,strace ,gc 日志，业务日志（通过日志发现问题，解决问题）

可视化工具： promethues , zabbix

5.1 TOP 指令

TOP 指令监控 cpu 使用情况，根据 cpu 使用情况分析系统整体运行情况；



Load average： 参数一：1 分钟内 cpu 平均使用率，参数二：5 分钟内 cpu 平均使用率，参数三：15 分钟之内 cpu 平均使用率

单核心 CPU:

Load average < 1 ,表示 cpu 毫无压力，比较空闲，运行通畅
Load average = 1，表示 cpu 刚刚被占满，没有可供提供的 cpu 资源了
Load average > 1，表示 cpu 满负荷运作，线程处于阻塞状态，等待 cpu 的资源
Load average > 5，表示 cpu 线程已经严重阻塞，必须进行处理了；

4 核心 CPU:

Load average < 4 ,表示 cpu 毫无压力，比较空闲，运行通畅
Load average = 4，表示 cpu 刚刚被占满，没有可供提供的 cpu 资源了
Load average > 4，表示 cpu 满负荷运作，线程处于阻塞状态，等待 cpu 的资源
Load average > 10，表示 cpu 线程已经严重阻塞，必须进行处理了；

生产环境中： cpu 持续占用率，长时间超过 70%，其实就必须对服务进行处理了；

5.2 Free

Free 是排查线上内存问题的重要指令，内存问题很多时候是引起 cpu 使用率较高的原因；

```
[root@qps004 java18]# free -m
              total        used        free      shared  buff/cache   avai
Mem:           7551        1648        4319           0        1583
Swap:             0           0           0
[root@qps004 java18]#
```

5.3 磁盘

df 指令查看磁盘使用情况，有时候服务出现了问题，有可能是磁盘不够；

```
[root@qps004 java18]# df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        3.7G     0  3.7G   0% /dev
tmpfs           3.7G     0  3.7G   0% /dev/shm
tmpfs           3.7G  544K  3.7G   1% /run
tmpfs           3.7G     0  3.7G   0% /sys/fs/cgroup
/dev/vda1       148G   13G  129G   9% /
tmpfs           756M     0  756M   0% /run/user/0
[root@qps004 java18]# df -m /root/
Filesystem     1M-blocks  Used Available Use% Mounted on
/dev/vda1         151061 12756    131941   9% /
```

5.4 网络

Dstat，集成了 vmstat , iostat ,netstat 工具功能完成查询任务；

```
[root@qps004 java18]# dstat -n
-net/total-
 recv  send
   0     0
 120B  106B
 102B  244B
  60B   42B
 126B  212B
 892B   10k
  60B   54B
  60B  106B
  60B  106B
```

业务问题： jmap ,jstack ,jinfo,jstat ,可视化工具排查

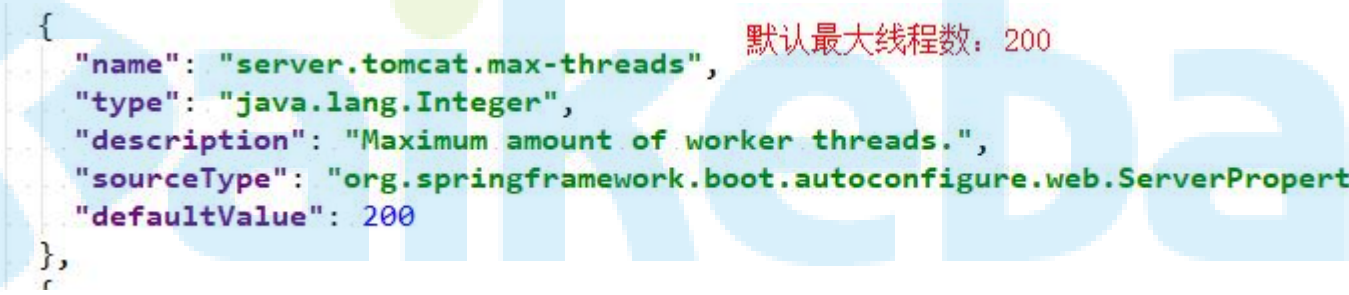
6 服务端调优

6.1 Tomcat 服务器调优(内置服务器)

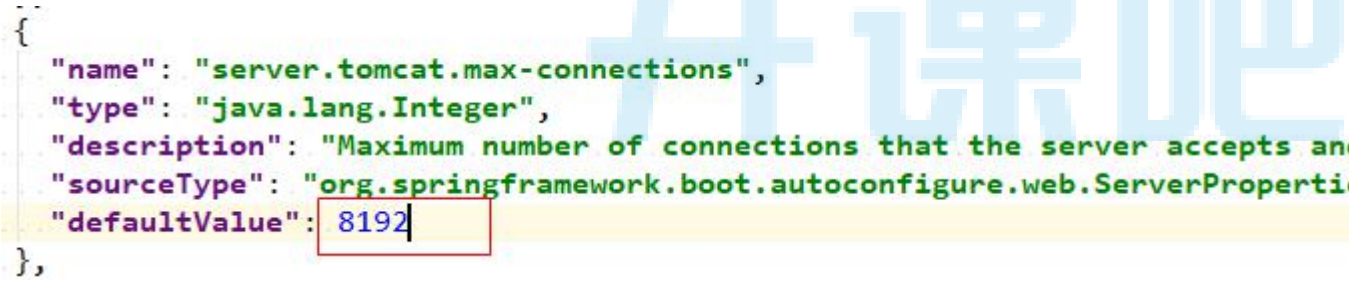
问一个问题：什么时间点介入调优？？？（开发的时候，服务上线运行的时候）
答案： 服务上线后进行介入调优，遇到问题，解决问题；持续不断对服务进行调优，问题（性能问题）排查

1）tomcat 服务器原始参数

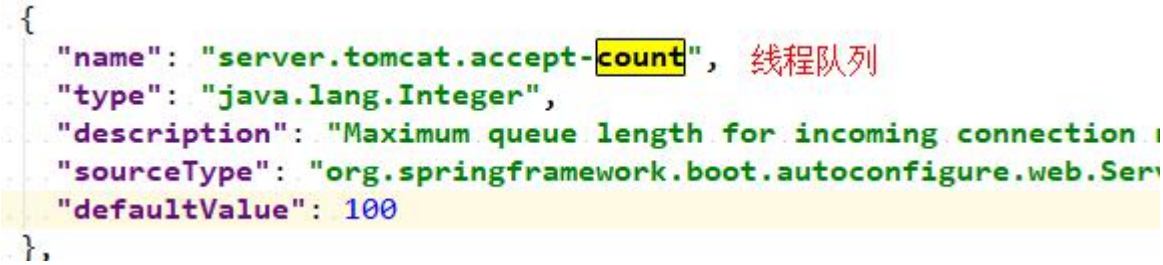
Tomcat 默认使用的线程数：



Tomcat 默认连接数：



Tomcat 等等队列：



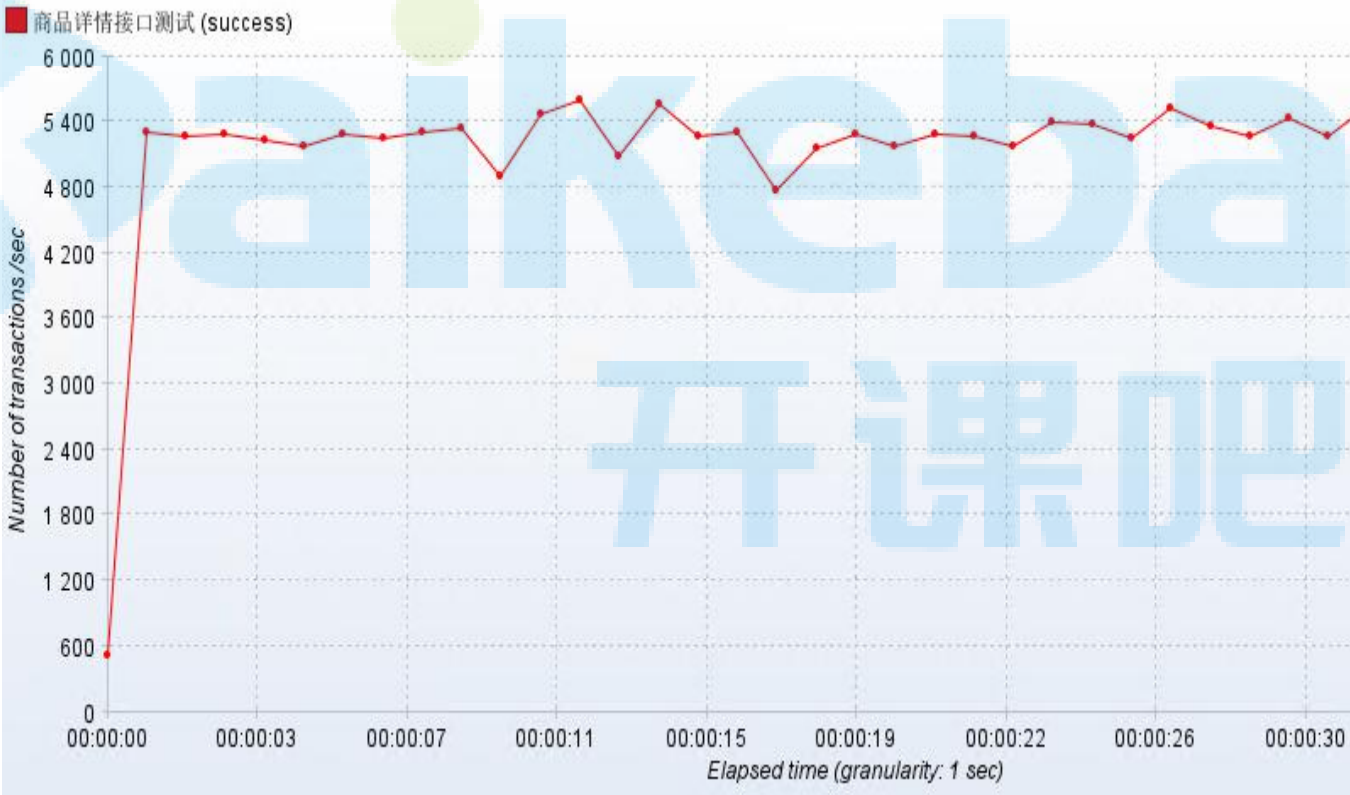
因此可以看见 tomcat 默认等等队列 100，最大线程 200，相当于做了限流，最多只能进入这

么多线程；超过这个线程数，就会被抛弃；

2）Tomcat 服务调优

```
# tomcat 并发线程数调整，产生更多的并发线程，处理业务
tomcat:
  max-threads: 800  # 最大线程数
  accept-count: 1000 # 等待对象，最多允许 1000 个线程在队列中等待
  max-connections: 20000 # 最大允许有 20000 个链接被建立
  min-spare-threads: 100 # 最大空闲数，防止流量洪峰
  uri-encoding: utf-8
```

没有优化之前性能：

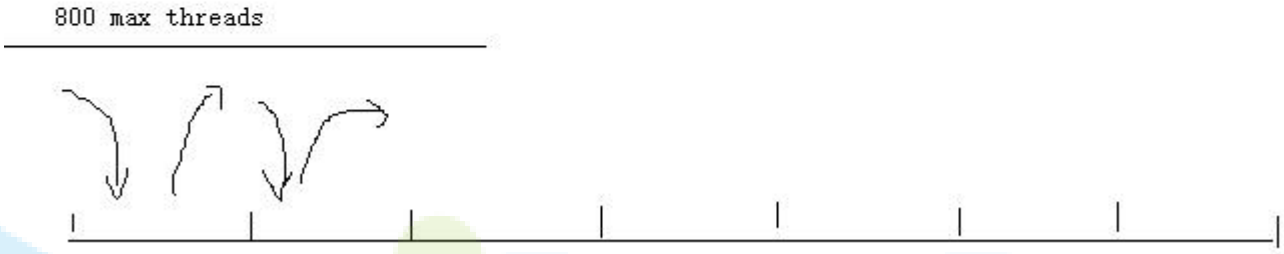


优化之后：查看 TPS 是否有提升：

问题：经过 tomcat 参数优化后，性能为什么没有获得提升呢？？ 按性能调优原理来说： 增大线程数，性能一定会提升的，但是....

```
@Override
public TbSeckillGoods findOne(Integer id){

    //直接从数据库查询
    //主键查询 ： cpu不耗时操作
    TbSeckillGoods seckillGoods = seckillGoodsMapper.selectByPrimaryKey
    //计算对象大小
```

主键查询： 没有执行任何的业务，只是一个主键查询，主键查询是数据库最快查询方式，查询耗时 0ms~10ms，此操作为一个不耗时操作；不耗时操作对调优无感；



6.2 模拟耗时操作

模拟业务耗时时间： 1s 钟时间

```
@Override
public TbSeckillGoods findOne(Integer id){

        //直接从数据库查询
        //主键查询 ： cpu不耗时操作
        TbSeckillGoods  seckillGoods = seckillGoodsMapper.selectByPrimaryKey
        //计算对象大小

        //模拟程序耗时操作，如果方法是一个笔记耗时的操作，性能优化非常有必要的！！
        try {
            Thread.sleep( millis: 1000);

            LOGGER.info("模拟耗时操作，睡眠1s时间！ ");
            LOGGER.info("对象占用jvm堆内存大小：{}", RamUsageEstimator.huma

        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        //返回结果
        return seckillGoods;

}
```

1）优化前压力测试结果

没有优化之前： 最大线程 200+

```
[root@qps004 java18]# pstree -p 20566 | wc -l
224
[root@qps004 java18]# pstree -p 20566 | wc -l
224
[root@qps004 java18]#
```

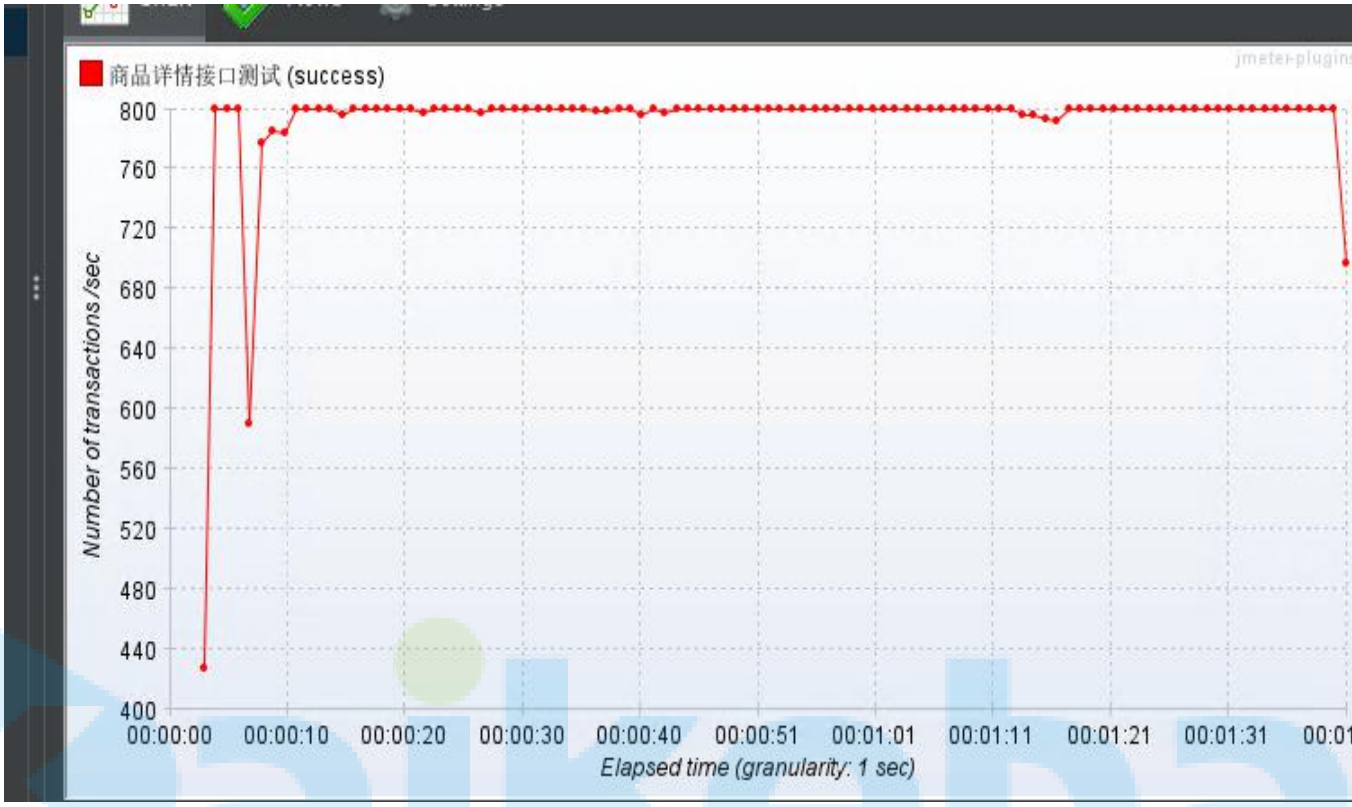
没有优化之前，性能测试： 吞吐能力就在 200 TPS

2）优化后结果

可以发现工作最大线程数提升了 4 倍，因此性能必然会提升：

```
[root@qps004 java18]# pstree -p 20906 | wc -l
824
[root@qps004 java18]# pstree -p 20906 | wc -l
824
[root@qps004 java18]# pstree -p 20906 | wc -l
824
[root@qps004 java18]# pstree -p 20906 | wc -l
824
[root@qps004 java18]#
```

服务吞吐能力获得 4 倍的提升，因此优化成功!!


商品详情接口测试 (success)
Number of transactions /sec
800
760
720
680
640
600
560
520
480
440
400
00:00:00
00:00:10
00:00:20
00:00:30
00:00:40
00:00:51
00:01:01
00:01:11
00:01:21
00:01:31
Elapsed time (granularity: 1 sec)