02-项目性能优化

刘亚雄

极客时间-Java 讲师

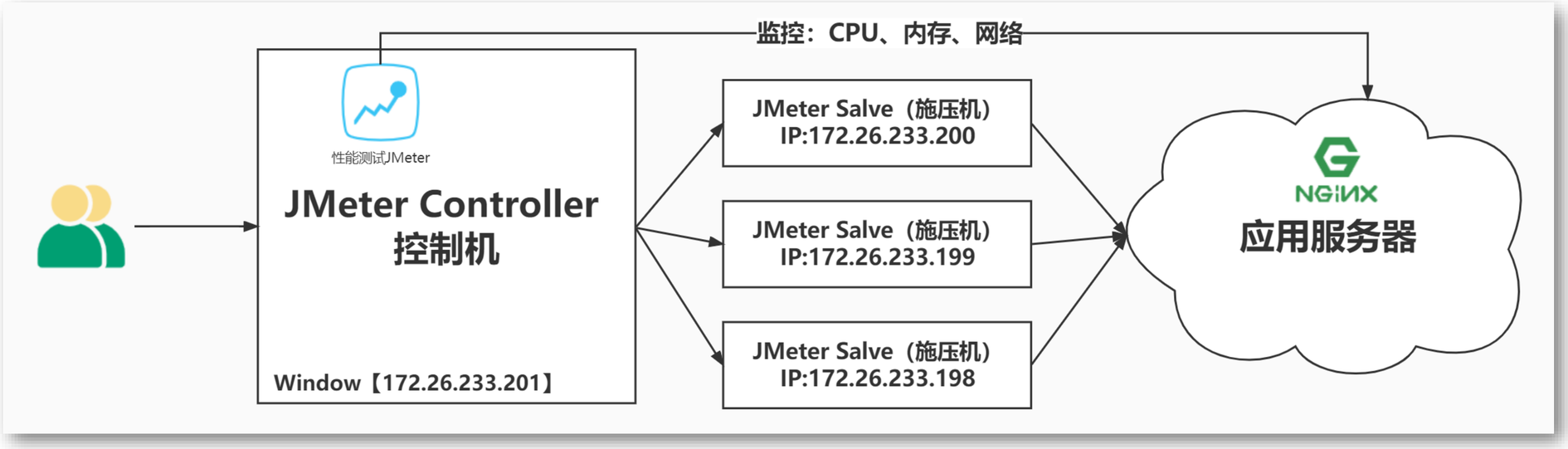
今日目标

1. 了解分布式压力测试
2. 掌握Web服务容器优化
3. 理解为什么进行数据库调优，及数据库调优的调什么
4. 掌握OpenResty调优案例
5. 理解缓存调优和JVM调优必要性和价值点

二、压力测试

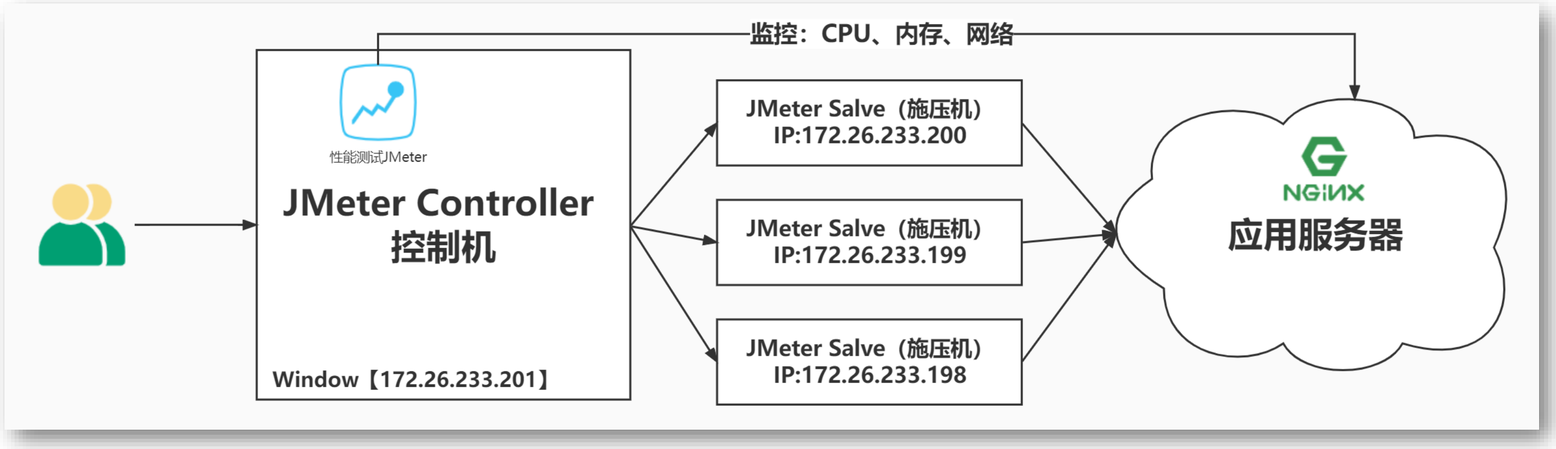
2.10 分布式压测

JMeter模拟高并发场景，**单机受限内存、CPU、网络带宽**，会出现服务压力还没有上去，但是施压机先到达瓶颈！为了让JMeter可以施加更强大的压力，JMeter提供分布式压测能力。



在JMeter Master节点配置**线程数10**，**循环100次**。如果有3台Salve 节点。那么每台Salve都会对被测服务发起10x100次请求。因此整个分布式压测集群产生的**总样本量**是：10 x 100 x 3 = **3000次**。

2.10 分布式压测



搭建分布式压测步骤：

- 第一步：JMeter Master搭建在Windows Server环境下
- 第二步：三台JMeter Salve搭建在Linux环境下
- 第三步：配置压测脚本，启动压测

搭建注意：

- 需保证Salve和server都在一个网络中，且在多网卡环境需要保证启动的网卡都在一个网段
- 需保证server和Salve之间的时间同步
- 需在内网配置JMeter主从通信端口【1个固定，1个随机】

三、服务容器优化

3.1 Tomcat调优

为什么对SpringBoot嵌入式的Web服务器Tomcat进行调优？

- 基于梯度压测分析瓶颈得出结论：**当接口响应时间比较长时，性能瓶颈主要卡在Web服务器中**
- Tomcat三大配置maxThreads、acceptCount、maxConnections

```
server.tomcat.uri-encoding: UTF-8
server.tomcat.accept-count: 1000 # 等待队列最多允许1000个请求在队列中等待
server.tomcat.max-connections: 20000 # 最大允许20000个链接被建立
## 最大工作线程数，默认200，4核8g内存，线程数经验值800
server.tomcat.threads.max: 800 # 并发处理创建的最大的线程数量
server.tomcat.threads.min-spare: 100 # 最大空闲连接数，防止突发流量


# 暴露所有的监控点
management.endpoints.web.exposure.include: '*'
# 定义Actuator访问路径
management.endpoints.web.base-path: /actuator
# 开启endpoint 关闭服务功能
management.endpoint.shutdown.enabled: true
```

1）最大线程数maxThreads

- 决定了Web服务器最大同时可以处理多少个请求任务数量
- 线程池最大线程数，默认值200

2）最大等待数accept-count

- 是指队列能够接受的最大等待数，如果等待队列超了请求会被拒绝
- 默认值100

最大线程数并不是越大越好?

3）最大连接数MaxConnections

- 是指在同一时间，Tomcat能够接受的最大连接数，如果设置为-1，则不限制连接数
- **最大连接数**和**最大等待数**关系：当连接数达到**最大连接数**后还会继续接请求，但不能超过**最大等待数**，否则拒绝连接

3.1 Tomcat调优

最大线程数并不是越大越好

- 最大线程数只是TPS的影响因素之一
- 增加线程是有成本的，不能无限制增大，创建线程需要消耗内存（Xss=1m）
- 线程过多会带来**频繁的线程上下文切换**，**什么是线程上下文切换呢？**

最大线程数的值应该设置多少合适呢？

- 需要基于业务系统的监控结果来定：RT均值很低不用设置，RT均值很高考虑加线程数
- 接口响应时间低于100毫秒，足以产生足够的TPS
- 如果没有证据表明系统瓶颈是线程数，则不建议设置最大线程数
- 个人经验值：**1C2G线程数200，4C8G线程数800**

Tomcat调优：优化Tomcat最大线程数

调优结论：在高负载场景下，TPS提升近1倍，RT大幅降低，异常占比降低

注意：配置修改需确认配置生效，否则再苦再累也白搭！

```
server.tomcat.uri-encoding: UTF-8
server.tomcat.accept-count: 1000 # 等待队列最多允许1000个请求在队列中等待
server.tomcat.max-connections: 20000 # 最大允许20000个链接被建立
## 最大工作线程数，默认200，4核8g内存，线程数经验值800
server.tomcat.threads.max: 800 # 并发处理创建的最大的线程数量
server.tomcat.threads.min-spare: 100 # 最大空闲连接数，防止突发流量


# 暴露所有的监控点
management.endpoints.web.exposure.include: '*'
# 定义Actuator访问路径
management.endpoints.web.base-path: /actuator
# 开启endpoint 关闭服务功能
management.endpoint.shutdown.enabled: true
```

3.1 Tomcat调优前后性能对比

调优前

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 KB/sec	发送 KB/sec
100Thread-HTT...	20000	525	520	524	529	566	515	1956	0.00%	188.8/sec	200.32	0.00
200Thread-HTT...	40000	565	522	530	539	1825	515	15578	0.37%	317.0/sec	336.49	0.00
300Thread-HTT...	60000	754	752	812	824	847	516	21050	0.15%	342.9/sec	364.05	0.00
400Thread-HTT...	80000	997	1006	1031	1051	1964	516	21061	0.68%	368.6/sec	391.47	0.00
500Thread-HTT...	100000	1203	1177	1264	1288	2143	516	21102	0.93%	346.1/sec	368.27	0.00
600Thread-HTT...	120000	1380	1149	1300	1329	21030	516	24631	2.48%	338.4/sec	362.62	0.00
700Thread-HTT...	140000	1502	1079	1294	1322	21038	515	21132	2.95%	351.1/sec	378.66	0.00
800Thread-HTT...	160000	1731	1057	1353	1543	21043	515	26484	4.69%	352.3/sec	382.90	0.00
总体	720000	1293	1012	1278	1327	21030	515	26484	2.27%	339.5/sec	364.15	0.00

调优后

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 KB/sec	发送 KB/sec
100Thread-HTT...	20000	530	520	528	535	617	515	2583	0.00%	187.1/sec	198.55	0.00
200Thread-HTT...	40000	525	522	530	534	548	515	7566	0.02%	356.0/sec	377.83	0.00
300Thread-HTT...	60000	534	525	534	539	562	515	21060	0.10%	458.8/sec	487.00	0.00
400Thread-HTT...	80000	572	529	562	588	687	514	21075	0.31%	550.4/sec	584.41	0.00
500Thread-HTT...	100000	742	535	674	779	5617	515	27762	1.62%	452.9/sec	482.06	0.00
600Thread-HTT...	120000	843	534	670	823	15604	515	21148	2.05%	474.2/sec	507.07	0.00
700Thread-HTT...	140000	884	536	661	768	21033	514	21236	1.97%	490.5/sec	525.92	0.00
800Thread-HTT...	160000	1028	532	601	738	21038	515	21144	3.17%	502.4/sec	541.38	0.00
总体	720000	796	530	614	714	15574	514	27762	1.70%	457.7/sec	489.20	0.00

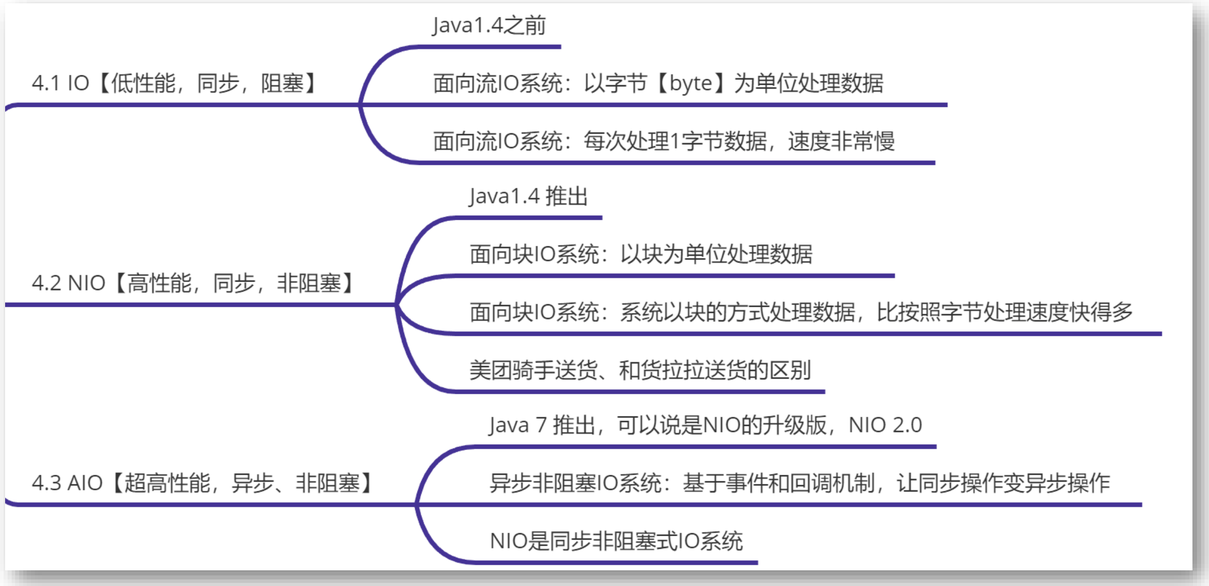
吞吐量增长34%

3.2 网络IO模型调优

众所周知**系统文件读写性能（IO）**是影响应用程序性能的关键因素之一。

Java的NIO【New IO】是从Java 1.4版本开始引入的一套新的IO API用于替代标准的Java IO API。NIO与原来的IO有同样的作用，但是使用的方式完全不同。

JDK1.7之后Java对NIO再次进行了极大的改进，增强了对文件处理和文件系统特性的支持。我们称之为AIO，也可以叫NIO2

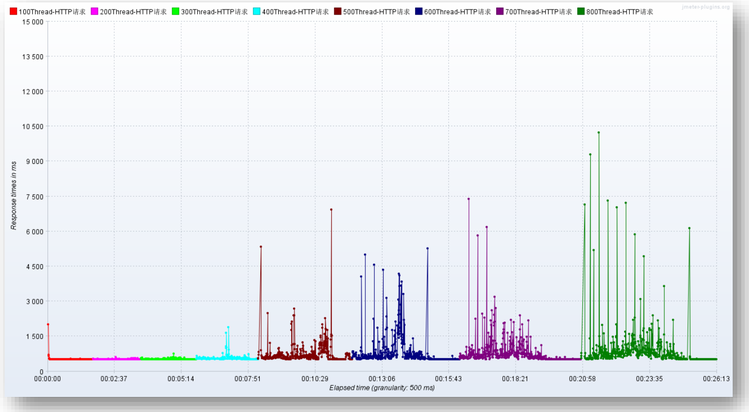


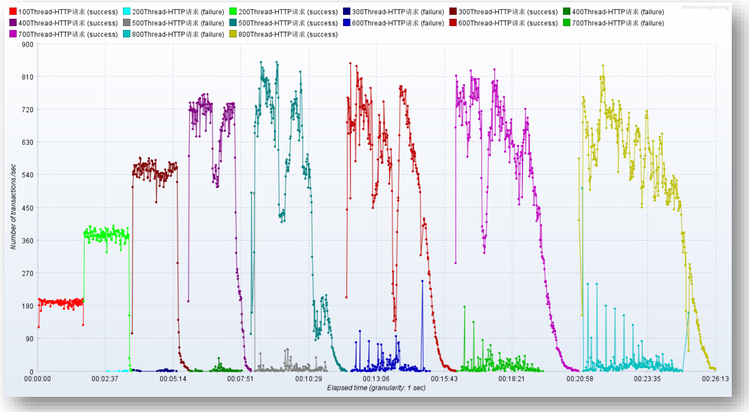
IO模型调优：使用NIO2的Http协议实现，对Web服务器进行优化

调优结论：可以发现服务响应时间大幅缩短，并且稳定

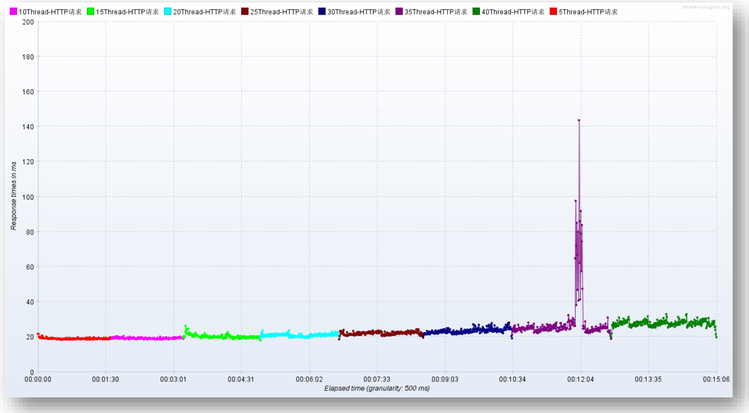
3.2 网络IO调优前后性能对比

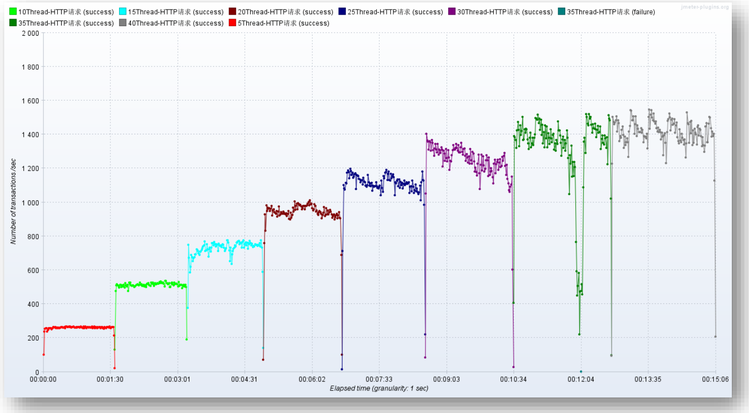
调优前





调优后





将NIO升级为AIO之后，RT的毛刺大幅降低，异常数（超时3s）几乎为0

3.3 Web容器优化

Undertow红帽公司开发的一款**基于NIO的高性能Web嵌入式服务器**，轻量级Servlet容器，比Tomcat更轻量级，没有可视化操作界面，没有其他诸如JSP模板引擎的功能，只专注于服务器部署，因此Undertow服务器性能略好于Tomcat服务器

特性：

- 支持Http协议
- 支持Http2协议
- 支持Web Socket
- 最高支持到Servlet4.0
- 支持嵌入式

SpringBoot切换Web容器

① Spring-boot-starter-tomcat排除tomcat依赖

② 导入undertow的starter依赖

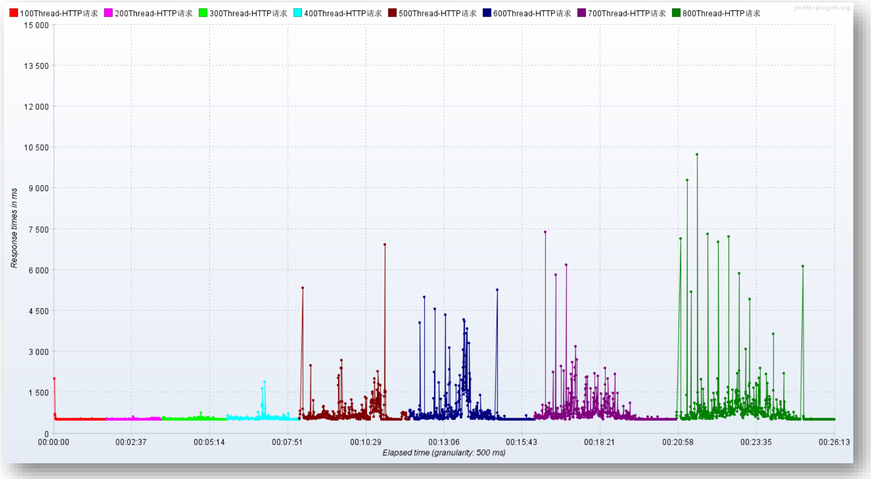
③ 配置undertow

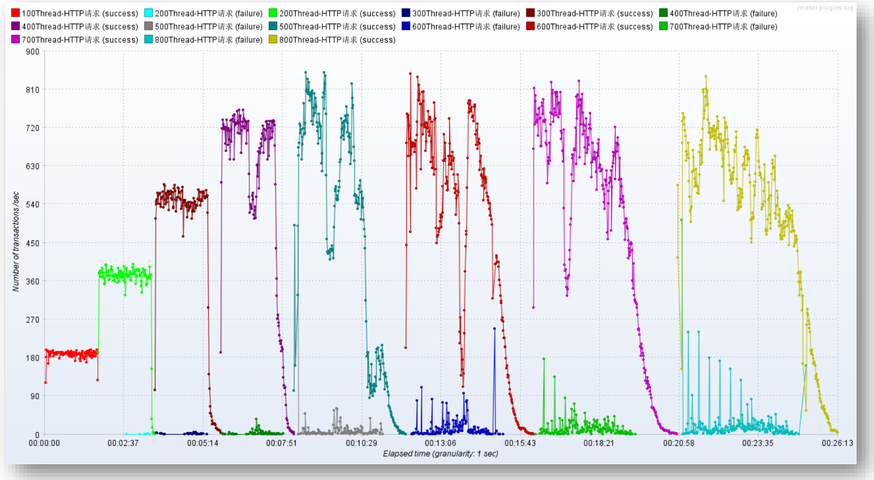
调优结论：

- **更换了服务容器之后，RT更平稳，TPS的增长趋势更平稳**
- **异常数几乎为0**
- **在低延时情况下，接口通吐量不及Tomcat**

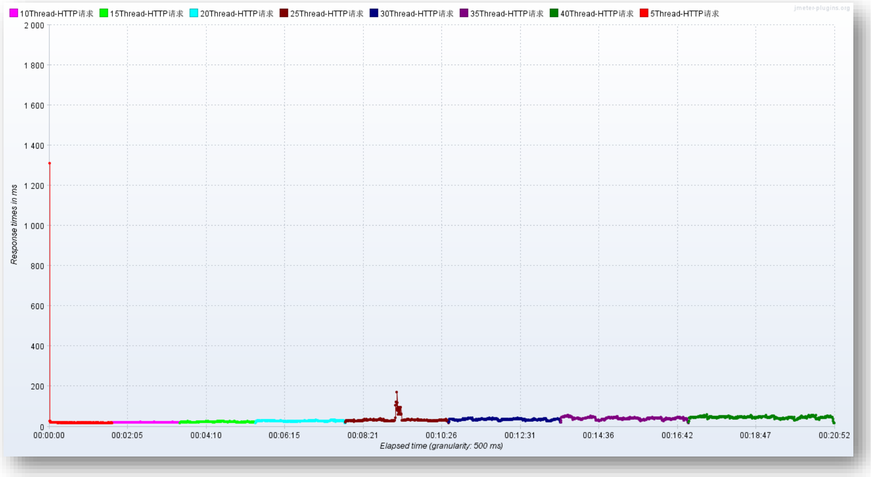
3.3 Tomcat与Undertow前后性能对比

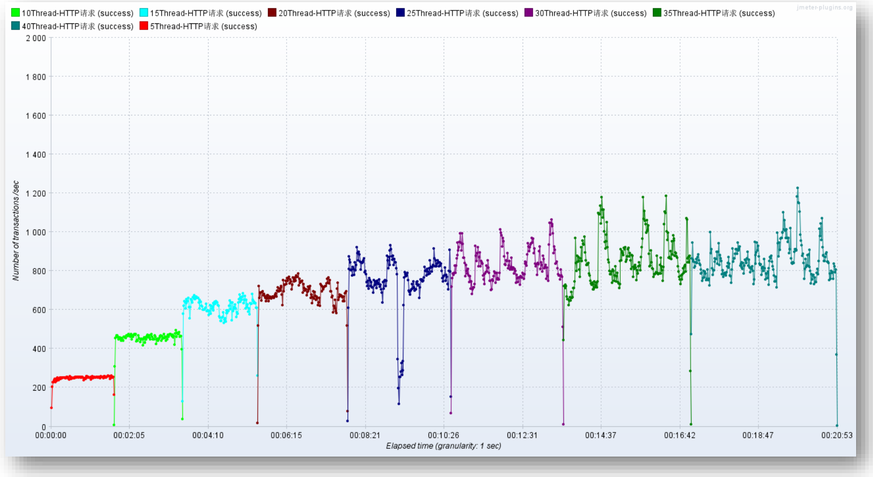
调优前





调优后





更换了服务容器之后，RT更加平稳，TPS的增长趋势更平稳，异常数（超时3s）几乎为0

四、其他：DB、OpenResty、Cache、JVM调优

4.1 数据库调优

请问这两条SQL语句有什么区别呢？你来猜一猜，那条SQL语句执行查询效果更好！

```sql
# 语句01
select id from tb_sku where name='华为 HUAWEI 麦芒7 6G+64G 魅海蓝 全网通  前置智慧双摄  移动联通电信4G手机 双卡双待';
# 语句02
select id from tb_sku where spu_id=10000018913700;
```

01-为什么要进行MySQL数据库调优？

- 提升网站整体通吐量，优化用户体验
- 后端性能优化性价比最高的一种方式

02-什么影响数据库性能？

- MySQL：**表结构设计，效率低下的SQL语句**，超大表，大事务，数据库相关配置，数据库架构
- 服务器： OS、CPU、memory、network

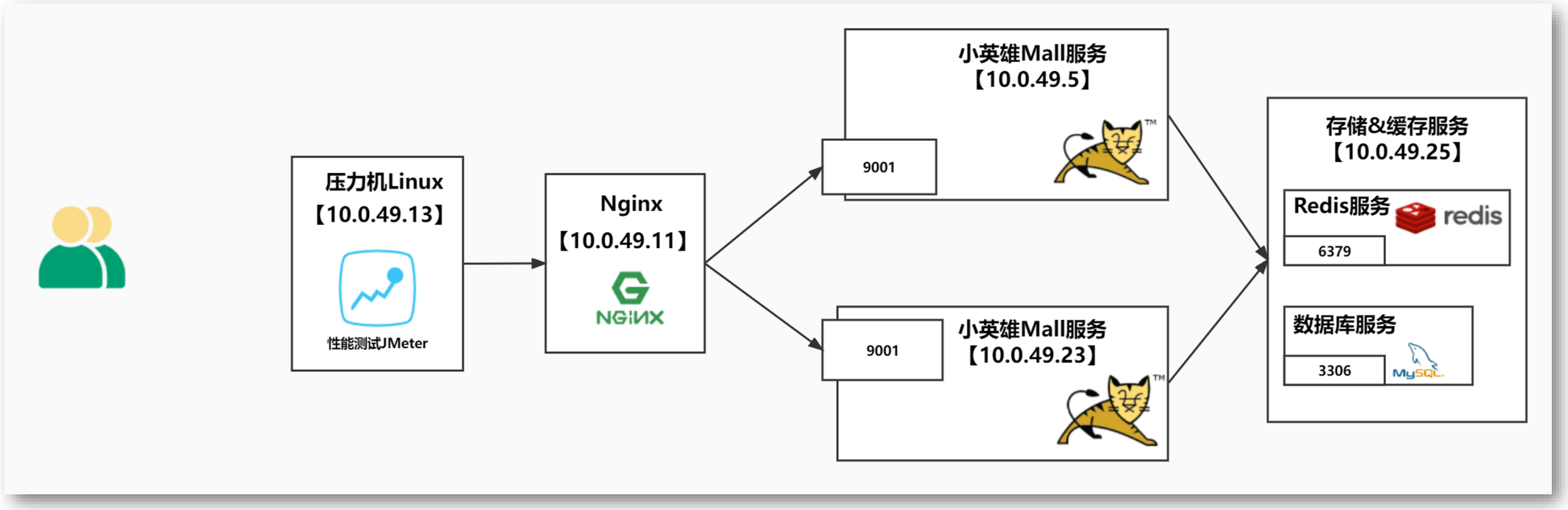
03-数据库调优到底调什么？

- 优化SQL语句、数据库表结构
- MySQL配置调优：最大连接数，连接超时，线程缓存，查询缓存，排序缓存，连接查询缓存等等
- DB服务器硬件优化：多核CPU、更大内存

上案例：压测两条SQL语句看一下谁效果更好

4.2 OpenResty调优

OpenResty**是一个基于 Nginx 与 Lua 的高性能 Web 平台**，其内部集成了大量精良的 Lua 库、第三方模块以及大多数的依赖项。用于方便地搭建能够处理超高并发、扩展性极高的动态 Web 应用、Web 服务和动态网关。



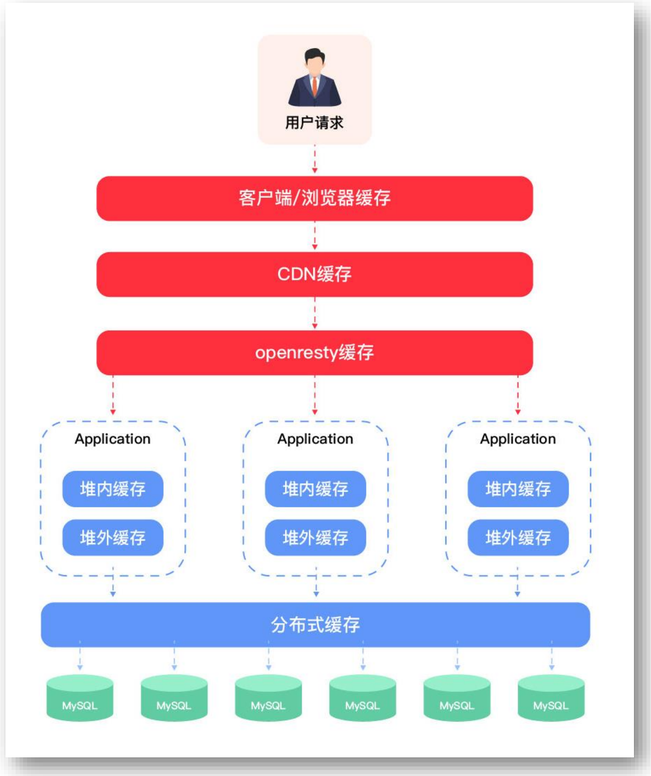
OpenResty调优：反向代理集群服务

调优结论：OpenResty调优之后，TPS会在原有的基础上再翻倍

4.3 多级缓存调优

需求：页面上显示广告轮播图，动态查询广告接口信息

- 实现方案01：查询数据库
- 优化方案02：多级缓存
- 优化方案03：进程内缓存



调优结论：TPS大幅提升，RT显著降低，距离用户越近，缓存价值越大

4.4 JVM调优

01-为什么JVM调优？

- **调优的最终目的都是为了应用程序使用最小的硬件消耗来承载更大的吞吐量。升职加薪利器**
- 更少的内存，更低的延迟，更大吞吐量，减少应用的GC暂停时间
- GC暂停会在高并发场景下，P99百分位的响应时间会产生影响

02-什么时候JVM调优？

- 1. 系统吞吐量与响应性能不高或下降
- 2. Heap内存（老年代）持续上涨达到设置的最大内存值
- 3. Full GC 次数频繁
- 4. GC 停顿时间过长（超过1秒）
- 5. 应用出现OutOfMemory 等内存异常
- 6. 应用中有使用本地缓存，且占用了大量的内存空间

那种配置JVM更好?

JVM配置01

```
JAVA_OPT="${JAVA_OPT} -Xms836m -Xmx836m -XX:MetaspaceSize=128m -Xss512k"

JAVA_OPT="${JAVA_OPT} -XX:+UseParallelGC -XX:+UseParallelOldGC "

JAVA_OPT="${JAVA_OPT} -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintGCDateStamps -XX:+PrintHeapAtGC -Xloggc:${BASE_DIR}/logs/gc-ps-po.log"
```

JVM配置02

```
JAVA_OPT="${JAVA_OPT} -Xms1024m -Xmx1024m -XX:MetaspaceSize=256m -Xss512k"

JAVA_OPT="${JAVA_OPT} -XX:+UseParNewGC -XX:+UseConcMarkSweepGC "

JAVA_OPT="${JAVA_OPT} -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintGCDateStamps -XX:+PrintHeapAtGC -Xloggc:${BASE_DIR}/logs/gc-parnew-cms.log"
```

03-调优调什么？

- 内存分配 + 垃圾回收器选配

THANKS

教育不是注满一桶水，而是点燃一把火