**[性能测试常见瓶颈分析及调优方法](https://www.cnblogs.com/imyalost/p/10850811.html)**

**一、注意事项**

**1、断言**

在压测时，为了判断发送的请求是否成功，一般会通过对请求添加断言来实现。使用断言时，建议遵循如下规范：

①、断言内容尽量以**status/code、msg/message**来判断（当然前提是接口设计遵循Restful规范）

Jmeter示例：



阿里云PTS：

如果使用的是PTS压测，则断言设置中，以code/status、msg/message**等于**对应的值为准；

②、尽可能**不要将所有的Response Body内容作为断言判断的内容**，这样很可能会导致大量的“断言”失败；

PS：然后很遗憾的是，见过很多做压测的童鞋，断言内容以整个响应参数内容做断言，导致大量的报错。

**2、成功率**

一般在性能测试中，我们都追求99.99%的成功率，但在实际的测试过程中，为了尽可能覆盖代码逻辑，在准备阶段会尽可能的准备较多的热点数据去做到覆盖。

这样的话，我们所关注的成功率指标，就要分为如下两种：

**①、事务成功率**

事务成功率在某些时候也可以视为请求成功率，在断言判断时以code/status等内容来作为请求是否成功的衡量依据；

**②、业务成功率**

实际的业务场景中，所谓的成功率，并不能仅根据返回的code/status来判断。比如：一个查询请求，无论是返回正确的查询结果还是由于对应数据返回空，这个请求都是成功的。

对应的响应参数可能是： {"status":"200","message":"success"} ；也可能是： {"status":"200","message":"暂无对应结果"} 。

**PS**：在性能测试过程中，考虑到业务成功率和请求成功率的不同指标，结合断言内容，需要灵活设置断言的方式（当然，我依然建议遵循如上的2点断言规范）！

**二、常见性能瓶颈解析及调优方案**

在性能测试中，导致性能出现瓶颈的原因很多，但通过直观的监控图表现出来的样子，根据出现的频次，大概有如下几种：

|  |  |
| --- | --- |
| 性能瓶颈出现频次 | 具体表现 |
| 高 | TPS波动较大 |
| 高 | 高并发下大量报错 |
| 中 | 集群类系统，各服务节点负载不均衡 |
| 中 | 并发数不断增加，TPS上不去，CPU耗用不高 |
| 低 | 压测过程中TPS不断下降，CPU使用率不断降低 |

下面对常见的几种性能瓶颈原因进行解析，并说说常见的一些调优方案：

**1、TPS波动较大**

**原因解析**：出现TPS波动较大问题的原因一般有**网络波动**、**其他服务资源竞争**以及**垃圾回收问题**这三种。

性能测试环境一般都是在内网或者压测机和服务在同一网段，可通过监控网络的出入流量来排查；

其他服务资源竞争也可能造成这一问题，可以通过**Top命令或服务梳理**方式来排查在压测时是否有其他服务运行导致资源竞争；

垃圾回收问题相对来说是最常见的导致TPS波动的一种原因，可以通过**GC监控命令**来排查，命令如下：

1 # 实时打印到屏幕

2 jstat -gc PID 300 10

3 jstat -gcutil PID 300 10

4 # GC信息输出到文件

5 jstat -gc PID 1000 120 >>/path/gc.txt

6 jstat -gcutil PID 1000 120 >>/path/gc.txt

**调优方案**：

网络波动问题，可以让运维同事协助解决（比如切换网段或选择内网压测），或者等到网络较为稳定时候进行压测验证；

资源竞争问题：通过命令监控和服务梳理，找出压测时正在运行的其他服务，通过沟通协调停止该服务（或者换个没资源竞争的服务节点重新压测也可以）；

垃圾回收问题：通过GC文件分析，如果发现有频繁的FGC，可以通过**修改JVM的堆内存参数Xmx**，然后再次压测验证（Xmx最大值不要超过服务节点内存的50%！）

**2、高并发下大量报错**

**原因解析**：出现该类问题，常见的原因有**短连接导致的端口被完全占用**以及**线程池最大线程数配置较小**及**超时时间较短**导致。

**调优方案**：

短连接问题：修改服务节点的tcp\_tw\_reuse参数为1，释放TIME\_WAIT scoket用于新的连接；

线程池问题：修改服务节点中容器的server.xml文件中的配置参数，主要修改如下几个参数：

[复制代码](javascript:void(0);)

# 最大线程数，即服务端可以同时响应处理的最大请求数

maxThreads="200"

# Tomcat的最大连接线程数，即超过设定的阈值，Tomcat会关闭不再需要的socket线程

maxSpareThreads="200"

# 所有可用线程耗尽时，可放在请求等待队列中的请求数，超过该阈值的请求将不予处理，返回Connection refused错误

acceptCount="200"

# 等待超时的阈值，单位为毫秒，设置为0时表示永不超时

connectionTimeout="20000"

[复制代码](javascript:void(0);)

**3、集群类系统，各服务节点负载不均衡**

**原因解析**：出现这类问题的原因一般是SLB服务设置了会话保持，会导致请求只分发到其中一个节点。

**调优方案**：如果确认是如上原因，可通过修改SLB服务（F5/HA/Nginx）的会话保持参数为None，然后再次压测验证；

**4、并发数不断增加，TPS上不去，CPU使用率较低**

**原因解析**：出现该类问题，常见的原因有：**SQL没有创建索引**/**SQL语句筛选条件不明确**、代码中设有**同步锁**，高并发时出现锁等待；

**调优方案**：

SQL问题：没有索引就创建索引，SQL语句筛选条件不明确就优化SQL和业务逻辑；

同步锁问题：是否去掉同步锁，有时候不仅仅是技术问题，还涉及到业务逻辑的各种判断，是否去掉同步锁，建议和开发产品同事沟通确认；

**5、压测过程中TPS不断下降，CPU使用率不断降低**

**原因解析**：一般来说，出现这种问题的原因是因为线程block导致，当然不排除其他可能；

**调优方案**：如果是线程阻塞问题，修改线程策略，然后重新验证即可；

**6、其他**

除了上述的五种常见性能瓶颈，还有其他，比如：**connection reset、服务重启、timeout**等，当然，分析定位后，你会发现，我们常见的性能瓶颈，

导致其的原因大多都是因为参数配置、服务策略、阻塞及各种锁导致。。。

**性能瓶颈分析参考准则：从上至下、从局部到整体！**