1、 测试环境准备

jmeter地址： https://jmeter.apache.org/download_jmeter.cgi

下载地址：http://mirrors.tuna.tsinghua.edu.cn/apache//jmeter/binaries/apache-jmeter-5.0.zip

教程：http://www.cnblogs.com/jessicaxu/p/7501770.html

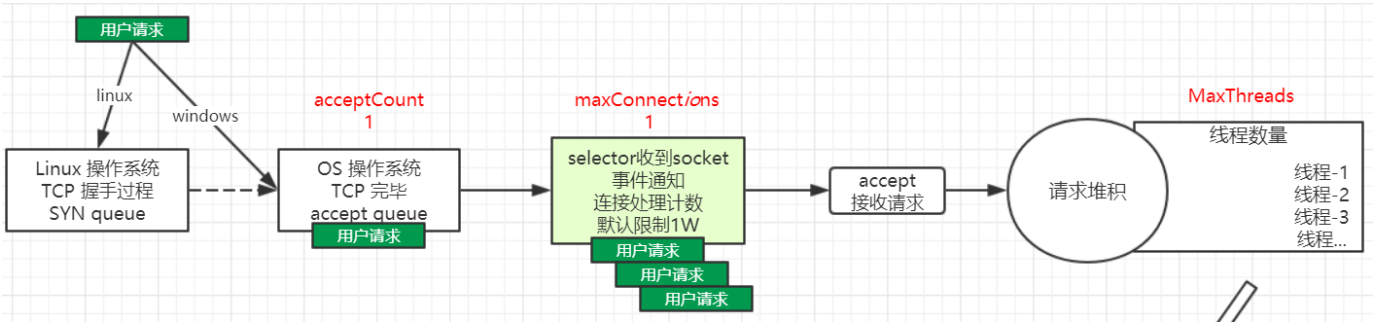
测试脚本：WebDemo测试用例.jmx

2、 连接数 调整

总共连接数 = acceptCount+ connections

connections： Tomcat能接收的请求限制；

acceptCount： 超过Tomcat能接收的请求数以后，堆积在操作系统的数量(windows 和 linux 略有不同)；




2.1 什么时候需要调整connections？如何调整？

connections小于maxThread的时候；需要调大；最好是比预期的最高并发数要大20%；反正是堆积到tomcat的work处理线程池中(堆积占内存)；

2.2 什么时候需要调整acceptCount?

想受理更多用户请求，却又不想堆积在tomcat中，利用操作系统来高效的堆积，可以调整为最高并发数 - connections；

实际上不需要调整，tomcat默认100，linux默认128；最好是把连接控制交给应用程序，这样方便管理。

启动方式： java -jar web-demo-1.1.0.jar --server.tomcat.max-connections=1 --server.tomcat.max-thread=1 --server.tomcat.acceptCount=1

3、 并发处理线程数 调整

线程太少，CPU利用率过低，程序的吞吐量变小，资源浪费，容易堆积。

线程太多，上下文频繁切换，性能反而变低。

3.1 线程数调为多少合适?

场景代入：服务器配置2核，不考虑内存问题。收到请求，java代码执行耗时50ms，等待数据返回50ms

理想的线程数量= (1 + 代码阻塞时间/代码执行时间) * cpu数量

实际情况是跑起代码，压测环境进行调试。不断调整线程数，将CPU打到80~90%的利用率。

java -jar web-demo-1.1.0.jar --server.tomcat.max-threads=500

4、 总结

请求多，CPU占用率高了，如果能接受很慢的响应，就加大。 否则就集群分流

认清现实，优化代码才是王道，配置只能是锦上添花!