### 1、引子

对于互联网公司，线上CPU飙升的问题很常见（例如某个活动开始，流量突然飙升时），按照本文的步骤排查，基本1分钟即可搞定！

### 2、问题复现

线上系统突然运行缓慢，CPU飙升，甚至到100%，以及Full GC次数过多，接着就是各种报警：例如接口超时报警等。此时急需快速线上排查问题。

### 3、问题排查

不管什么问题，既然是CPU飙升，肯定是查一下耗CPU的线程，然后看看GC。

#### 3.1 核心排查步骤

1. 执行top命令：查看所有进程占系统CPU的排序。极大可能排第一个的就是咱们的java进程（COMMAND列）。PID那一列就是进程号。
2. 执行top -pid 进程号命令：查看java进程下的所有线程占CPU的情况。
3. 执行printf "%x\n 10命令 ：后续查看线程堆栈信息展示的都是十六进制，为了找到咱们的线程堆栈信息，咱们需要把线程号转成16进制。例如,printf "%x\n 10-》打印：a，那么在jstack中线程号就是0xa.
4. 执行 jstack 进程号 | grep 线程ID 查找某进程下-》线程ID（jstack堆栈信息中的nid）=0xa的线程状态。如果"VM Thread" os\_prio=0 tid=0x00007f871806e000 nid=0xa runnable，第一个双引号圈起来的就是线程名，如果是“VM Thread”这就是虚拟机GC回收线程了
5. 执行jstat -gcutil 进程号 统计间隔毫秒 统计次数（缺省代表一致统计），查看某进程GC持续变化情况，如果发现返回中FGC很大且一直增大-》确认Full GC! 也可以使用jmap -heap 进程ID查看一下进程的堆内从是不是要溢出了，特别是老年代内从使用情况一般是达到阈值(具体看垃圾回收器和启动时配置的阈值)就会进程Full GC。
6. 执行jmap -dump:format=b,file=filename 进程ID，导出某进程下内存heap输出到文件中。可以通过eclipse的mat工具查看内存中有哪些对象比较多。

#### 3.2 原因分析

1.内存消耗过大，导致Full GC次数过多

执行步骤1-5：

* 多个线程的CPU都超过了100%，通过jstack命令可以看到这些线程主要是垃圾回收线程-》上一节步骤2
* 通过jstat命令监控GC情况，可以看到Full GC次数非常多，并且次数在不断增加。--》上一节步骤5

确定是Full GC,接下来找到具体原因：

* 生成大量的对象，导致内存溢出-》执行步骤6，查看具体内存对象占用情况。
* 内存占用不高，但是Full GC次数还是比较多，此时可能是代码中手动调用 System.gc()导致GC次数过多，这可以通过添加 -XX:+DisableExplicitGC来禁用JVM对显示GC的响应。

2.代码中有大量消耗CPU的操作，导致CPU过高，系统运行缓慢；

执行步骤1-4：在步骤4jstack，可直接定位到代码行。例如某些复杂算法，甚至算法BUG，无限循环递归等等。

3.由于锁使用不当，导致死锁。

执行步骤1-4：如果有死锁，会直接提示。关键字：deadlock.步骤四，会打印出业务死锁的位置。

造成死锁的原因：最典型的就是2个线程互相等待对方持有的锁。

4.随机出现大量线程访问接口缓慢。

代码某个位置有阻塞性的操作，导致该功能调用整体比较耗时，但出现是比较随机的；平时消耗的CPU不多，而且占用的内存也不高。

思路：

首先找到该接口，通过压测工具不断加大访问力度，大量线程将阻塞于该阻塞点。

执行步骤1-4：

"http-nio-8080-exec-4" #31 daemon prio=5 os\_prio=31 tid=0x00007fd08d0fa000 nid=0x6403 waiting on condition [0x00007000033db000]

java.lang.Thread.State: TIMED\_WAITING (sleeping)-》期限等待

at java.lang.Thread.sleep(Native Method)

at java.lang.Thread.sleep(Thread.java:340)

at java.util.concurrent.TimeUnit.sleep(TimeUnit.java:386)

at com.\*.user.controller.UserController.detail(UserController.java:18)-》业务代码阻塞点

如上，找到业务代码阻塞点，这里业务代码使用了TimeUnit.sleep()方法，使线程进入了TIMED\_WAITING(期限等待)状态。

5.某个线程由于某种原因而进入WAITING状态，此时该功能整体不可用，但是无法复现；

执行步骤1-4：jstack多查询几次，每次间隔30秒，对比一直停留在parking 导致的WAITING状态的线程。

例如CountDownLatch倒计时器，使得相关线程等待->AQS->LockSupport.park()。

"Thread-0" #11 prio=5 os\_prio=31 tid=0x00007f9de08c7000 nid=0x5603 waiting on condition [0x0000700001f89000]

java.lang.Thread.State: WAITING (parking) ->无期限等待

at sun.misc.Unsafe.park(Native Method)

at java.util.concurrent.locks.LockSupport.park(LockSupport.java:304)

at com.\*.SyncTask.lambda$main$0(SyncTask.java:8)-》业务代码阻塞点

at com.\*.SyncTask$$Lambda$1/1791741888.run(Unknown Source)

at java.lang.Thread.run(Thread.java:748)

### 4、总结

按照3.1节的6个步骤走下来，基本都能找到问题所在。