深入理解JVM（七）——性能监控工具

1. **前言**

工欲善其事必先利其器，性能优化和故障排查在我们大都数人眼里是件比较棘手的事情，一是需要具备一定的原理知识作为基础，二是需要掌握排查问题和解决问题的流程、方法。本文就将介绍利用性能监控工具，帮助开发者更快更准的找到问题产生的根源。本文分为三部分，第一部分将介绍在Linux环境下的常用监控工具，第二部分介绍Windows环境下的监控工具，第三部分将通过一个案例，介绍利用这些监控工具一步一步找出java应用程序的问题。

1. **Linux环境下的监控工具**

需要先声明的是，下面介绍的部分工具其实在Liunx环境、Windows环境下都可以使用，只不过在不同的环境下使用什么工具更合适。

下面我们想象这么一个场景：有一天运维人员看到生产环境下的服务器负载升高、CPU飙升，内存占用增大，请问接下来他该怎么办？可能有人会说，那就找到负载升高、CPU飙升。。的原因啊，如果是应用程序造成的就把它kill掉。如果采用这种方法，问题可能会短时间解决，但带来的问题是应用在一段时间不可用，并且我们没有找到问题发生的根源，下次重启后，问题依然会产生。那么如果出现类似的问题，严格的排查流程是怎样的呢？在回答这个问题之前，我们先了解一下Linux上常用的几个监控工具：

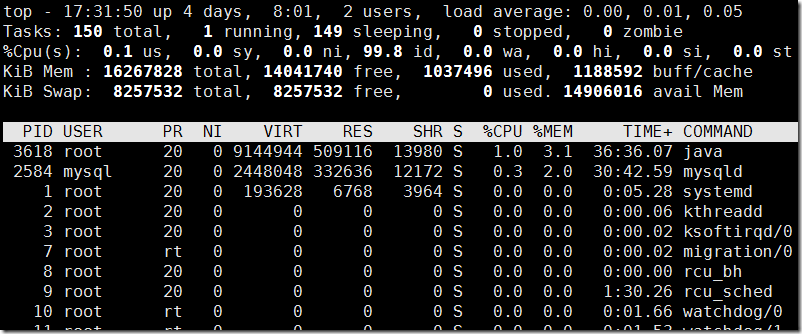
Uptime

|  |
| --- |
| //当前系统时间  [root@centos7\_template ~]#uptime  10:31:42 up 4 days, 1:01, 1 user,load average: 0.02, 0.02, 0.05 10:31:42  //持续运行时间,时间越大，说明你的机器越稳定。  up 4 days, 1:01  //用户连接数，是总连接数而不是用户数  1 user  //系统平均负载，统计最近1，5，15分钟的系统平均负载  load average: 0.02, 0.02, 0.05 |

该命令将显示目前服务器持续运行的时间，以及负载情况。

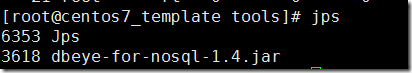
通过这个命令，可以最简便的看出系统当前基本状态信息，这里面最有用是负载指标，如果你还想查看当前系统的CPU/内存以及相关的进程状态，可以使用TOP命令。

TOP



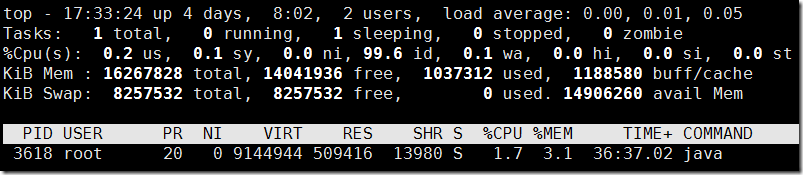
通过TOP命令可以详细看出当前系统的CPU、内存、负载以及各进程状态（PID、进程占用CPU、内存、用户）等。从上面的结果看出该系统上安装了MySQL、java，可以看到他们各自的进程ID，假如这时Java进程占用较高的CPU和内存，那么你就要留心了，如果程序中没有计算量特别大、占用内存特别多的代码，可能你的java程序出现了未知的问题，可以根据进程ID做进一步的跟踪。除了通过TOP命令找到进程信息以外，还可以通过jdk自带的工具JPS直接找到java程序的进程号。

JPS



可以看到jps命令直接罗列出了当前系统中存在的java进程，这里第一个是jps命令自己的java进程，而另外一个是我启动的nosql监控工具进程。通过这种方法查询到java程序的进程ID后，可以进一步通过：

top -p 3618 // 这里的3618就是上面查询到的java程序的进程ID



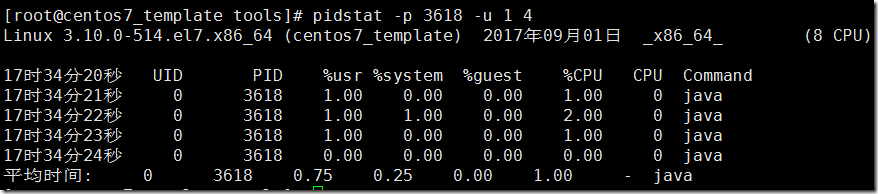
通过此方法可以准确的查看指定java进程的CPU/内存使用情况。

除此之外，vmstat命令也可以查看系统CPU/内存、swap、io等情况：

上面的命令每隔1秒采样一次，一共采样四次。CPU占用率很高，上下文切换频繁，说明系统有线程正在频繁切换，这可能是你的程序开启了大量的线程存在资源竞争的情况。另外swap也是值得关注的指标，如果swpd过高则可能系统能使用的物理内存不足，不得不使用交换区内存，还有一个例外就是某些程序优先使用swap，导致swap飙升，而物理内存还有很多空余，这些情况是需要注意的。

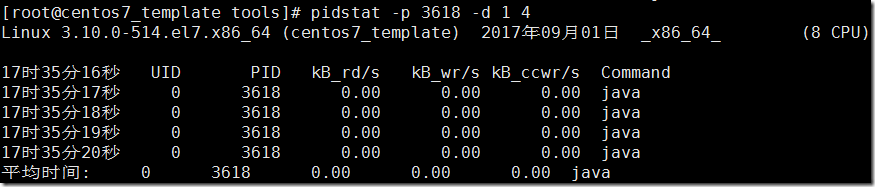
查看系统指标，还有一个第三方工具：pidstat，这个工具还是很好用的，需要先安装：

yum install sysstat



该命令监控进程id为3618的CPU状态，每隔1秒采样一次，一共采样四次。“%CPU”表示CPU使用情况，“CPU”则表示正在使用哪个核的CPU，这里为0表示正在使用第一个核。如果还要显示线程ID，则可以使用： pidstat -p 3618 -u -t 1 4

如果要监控磁盘读写情况，这可以使用：

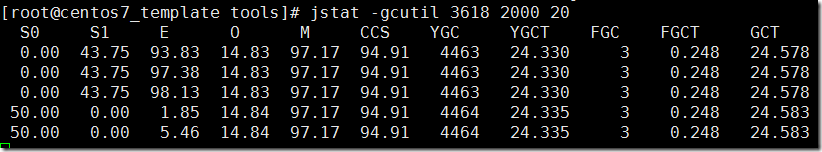


pidstat还有其他的参数，可以通过pidstat --help获取，再次不再赘述。

下面再介绍几个JDK自带有用的工具：jps、jstat、jmap、jstack。

jps：上面我们已经使用过了，他可以罗列出目前再服务器上运行的java程序及进程ID；

jstat：用于输出java程序内存使用情况，包括新生代、老年代、元数据区容量、垃圾回收情况。



上述命令输出进程ID为3618的内存使用情况（每2000毫秒输出一次，一共输出20次）

* S0：幸存1区当前使用比例
* S1：幸存2区当前使用比例
* E：伊甸园区使用比例
* O：老年代使用比例
* M：元数据区使用比例
* CCS：压缩使用比例
* YGC：年轻代垃圾回收次数
* FGC：老年代垃圾回收次数
* FGCT：老年代垃圾回收消耗时间
* GCT：垃圾回收消耗总时间

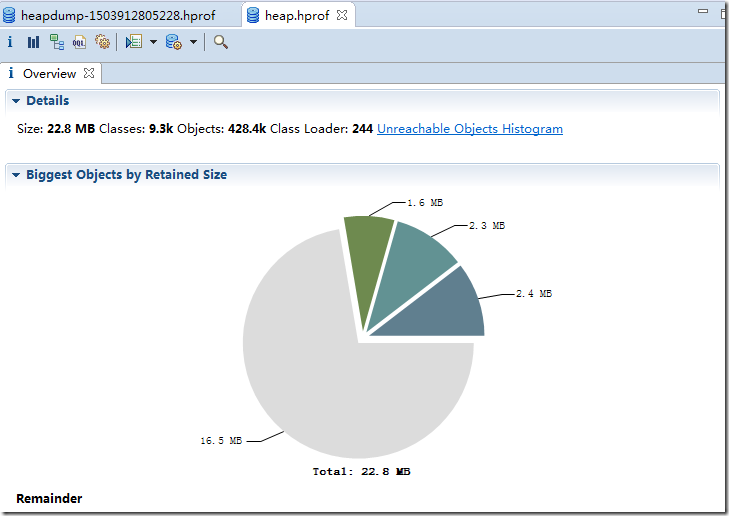
jmap：用于输出java程序中内存对象的情况，包括有哪些对象，对象的数量。

jmap -histo 3618

上述命令打印出进程ID为3618的内存情况。但我们常用的方式是将指定进程的内存heap输出到外部文件，再由专门的heap分析工具进行分析,例如mat（Memory Analysis Tool），所以我们常用的命令是：

jmap -dump:live,format=b,file=heap.hprof 3618

将heap.hprof传输出来到window电脑上使用mat工具分析：



jstack：用户输出java程序线程栈的情况，常用于定位因为某些线程问题造成的故障或性能问题。

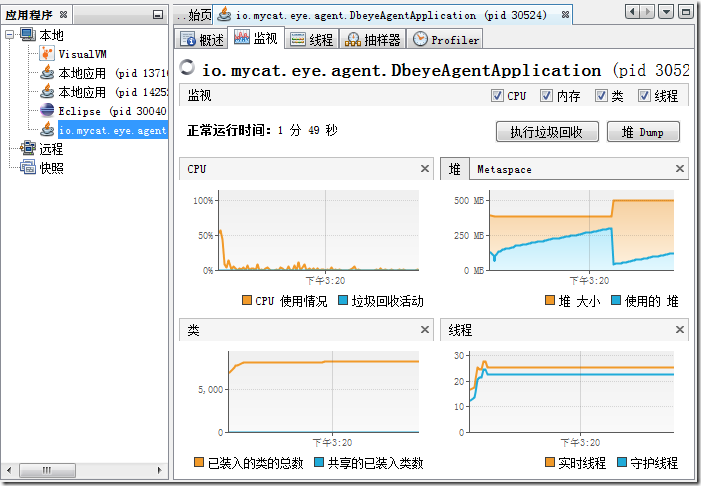
jstack 3618 > jstack.out

上述命令将进程ID为3618的栈信息输出到外部文件，便于传输到windows电脑上进行分析。

1. **Windows环境下的监控工具**

Windows环境下的监控工具也有很多，但是本文主要推荐jvisualvm.exe、MemoryAnalyzer.exe，有了他们其他工具几乎不需要了。

jvisualvm.exe在JDK安装目录下的bin目录下面，双击即可打开。



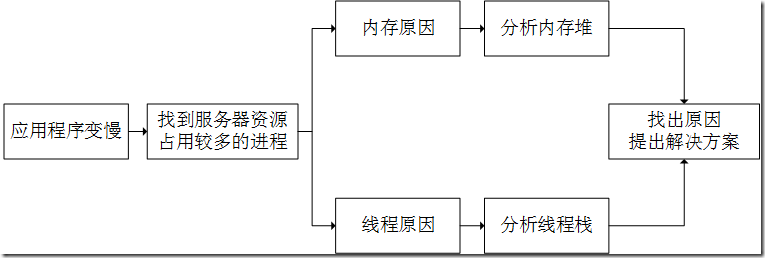
双击左侧你需要监控的java程序即可对它进行监控，这个工具包括对CPU、内存、线程、类都做了监控，功能非常强大，上文中介绍的所有功能，其他在这个工具上都已经有了。当然怎么用、如何分析它需要花时间去一点点积累。

MemoryAnalyzer.exe：上文我们已经提到，常用于分析内存堆使用情况，也是非常强大的工具。详细使用方法，这里就不再赘述，可以下载下来尝试一下。

上述介绍了基于Linux、Windows环境的监控工具，有了这些工具我们就要利用他们做对应的事情，下面将通过一个简单的案例，说明如何使用他们。

1. **案例分析**

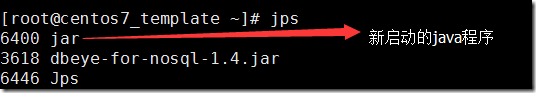
首先通过上述的介绍，我们对故障排查流程应该有了一个印象，这里先梳理出来：



**案例：**

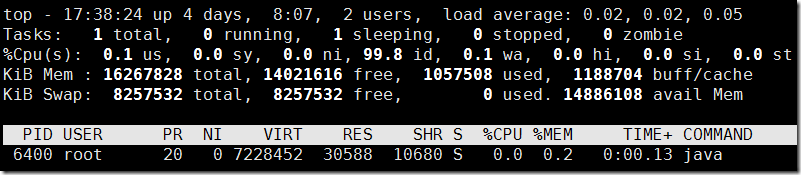
一个java应用启动以后，使用人员发现应用不可用，针对该现象我们做以下分析排查：

1. 首先查看服务器上的应用状态。使用jps命令查询当前在运行中的java进程：



这里进程ID为6400的java应用就是我们刚启用的，说明应用并没有挂掉，还在运行中。

1. 通过进程ID查询所占用的CPU、内存以及当前负载情况,top -p 6400。

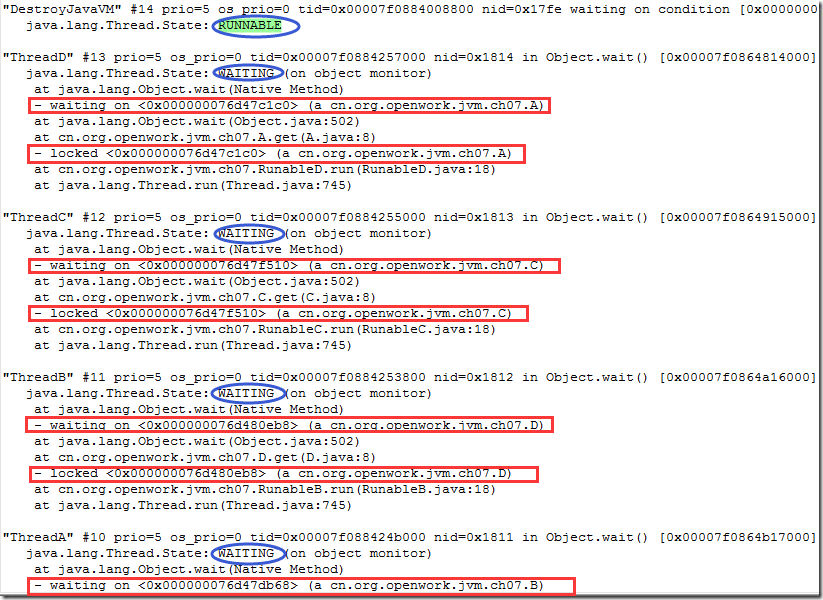


从以上结果看出该应用并没有引起系统负载过高，CPU、内存也没有出现异常情况。

1. 通过上述结果我们推测因为内存原因引起的故障可性能较小，所以我们优先排查线程栈，使用jstack命令，导出线程栈。

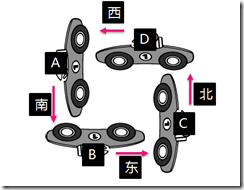
jstack 6400 > stack.out

我们将该文件传输出来便于查看。



查看线程栈可以看出，主线程处于运行状态，而子线程ThreadA、ThreadB、ThreadC、ThreadD一边在等待一个锁，同时又持有另外一个锁，其实看到这里我们基本推断该应用程序存在死锁，因此造成线程等待，应用不可用。通过以上栈的信息，我们就可以到程序代码中详细查看代码了，并且修改bug解决此问题。

死锁原理补充：



如图所示，造成死锁的原因是线程之间存在相互制约的情况，而任一线程都无法继续执行。

1. **来源**

http://www.cnblogs.com/leefreeman/p/7464179.html