最后修改：20140422

# Linux命令或工具

## uptime

系统运行时间，终端连接数，负载

[powerop@FATD00041763 100000558]$ uptime

20:17:53 up 22 days, 4:39, 0 users, load average: 14.16, 13.61, 11.80

[powerop@FATD00041763 100000558]$ uptime

20:18:43 up 22 days, 4:40, 0 users, load average: 14.52, 13.71, 11.93

解释参见TOP

## top

<https://linux.cn/article-2352-1.html>

|  |
| --- |
| op - 12:38:33 up 50 days, 23:15, 7 users, load average: 60.58, 61.14, 61.22  Tasks: 203 total, 60 running, 139 sleeping, 4 stopped, 0 zombie  Cpu(s) : 27.0%us, 73.0%sy, 0.0%ni, 0.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st  Mem: 1939780k total, 1375280k used, 564500k free, 109680k buffers  Swap: 4401800k total, 497456k used, 3904344k free, 848712k cached  PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND  1 root 20 0 18460 10104 8552 S 0.0 0.1 0:02.67 cexecutor  26 root 20 0 123080 18276 7328 S 0.0 0.2 1:00.10 supervisord  41 root 20 0 109536 14948 7348 S 0.0 0.2 0:00.05 python  下面是从其它地方拷贝的  PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND  4338 oracle 25 0 627m 209m 207m R 0 11.0 297:14.76 oracle  4267 oracle 25 0 626m 144m 143m R 6 7.6 89:16.62 oracle |

### 一. TOP前五行统计信息

统计信息区前五行是系统整体的统计信息。

#### 第一行是任务队列信息

op - 12:38:33 up 50 days, 23:15, 7 users, load average: 60.58, 61.14, 61.22

同 uptime 命令的执行结果:

12:38:33 — 当前系统时间

50 days, 23:15 — 系统已经运行了126天14小时29分钟（在这期间没有重启过）

7 users — 当前有2个用户登录系统

load average: 60.58, 61.14, 61.22 —分别是1分钟、5分钟、15分钟的负载情况。

其内容如下：

|  |  |
| --- | --- |
| 12:38:33 | 当前时间 |
| up 50days | 系统运行时间，格式为时:分 |
| 1 user | 当前登录用户数 |
| load average: 0.06, 0.60, 0.48 | 系统负载，即任务队列的平均长度。 分别为1，5,15分钟内的平均值。 |

#### 2. 第二，三行进程和CPU信息

当有多个CPU时，这些内容可能会超过两行。内容如下：

203 total, 60 running, 139 sleeping, 4 stopped, 0 zombie

Cpu(s) : 27.0%us, 73.0%sy, 0.0%ni, 0.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st

Tasks: 29 total 进程总数

1 running 正在运行的进程数

28 sleeping 睡眠的进程数

0 stopped 停止的进程数

0 zombie 僵尸进程数

6.7% us user，运行(未调整优先级的)用户进程占用CPU的百分比。

0.4% sy system，运行内核进程占用CPU的百分比。

0.0% ni niced，运行已调整优先级的用户进程的CPU百分比

92.9% id 空闲CPU百分比

0.0% wa IO wait，用于等待IO完成占用CPU的百分比

0.0% hi 硬中断（Hardware IRQ）占用CPU的百分比

0.0% si 软中断（Software Interrupts）占用CPU的百分比

0.0% st 这个虚拟机被hypervisor偷去的CPU时间

#### 3. 第四，五行为内存信息。

内容如下：

Mem: 1939780k total, 1375280k used, 564500k free, 109680k buffers

Swap: 4401800k total, 497456k used, 3904344k free, 848712k cached

第四行: 物理内存使用状态

8306544k total — 全部可用内存，物理内存总量（8GB）

7775876k used — 已使用内存，使用中的内存总量（7.7GB）

530668k free — 空闲，空闲内存总量（530M）

79236k buffers —缓冲内存， 缓存的内存量 （79M）

第五行：虚拟内存信息，swap交换分区

2031608k total — 全部，交换区总量（2GB）

2556k used — 已使用，使用的交换区总量（2.5M）

2029052k free — 空闲，空闲交换区总量（2GB）

4231276k cached — 缓冲，缓冲的交换区总量（4GB）

|  |  |
| --- | --- |
| Mem: 191272k total | 物理内存总量 |
| 173656k used | 使用的物理内存总量 |
| 17616k free | 空闲内存总量 |
| 22052k buffers | 用作内核缓存的内存量 |
| Swap: 192772k total | 交换区总量 |
| 0k used | 使用的交换区总量 |
| 192772k free | 空闲交换区总量 |
| 123988k cached | 缓冲的交换区总量。 内存中的内容被换出到交换区，而后又被换入到内存，但使用过的交换区尚未被覆盖， 该数值即为这些**内容已存在于内存中**的交换区的大小。相应的内存再次被换出时可不必再对交换区写入。 |

### 二.  进程信息

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

1 root 20 0 18460 10104 8552 S 0.0 0.1 0:02.67 cexecutor

26 root 20 0 123080 18276 7328 S 0.0 0.2 1:00.10 supervisord

41 root 20 0 109536 14948 7348 S 0.0 0.2 0:00.05 python

第七行以下：各进程（任务）的状态监控

PID 进程id

USER 进程所有者

PR 进程优先级，这个字段的一些值是'rt'。这意味这这些进程运行在实时态

NI nice值(优先级) 越小优先级越高，负值高优先级，正值低优先级

VIRT 进程使用的虚拟内存总量，单位kb。VIRT=SWAP+RES

RES 驻留内存大小，单位kb，驻留内存是任务使用的非交换物理内存；

进程使用的、未被换出的物理内存大小。RES=CODE+DATA

SHR 共享内存大小，单位kb；

S 进程状态。D不可中断的睡眠态;R运行态;S睡眠态;T跟踪/停止;Z僵尸态

%CPU 上次更新到现在所有任务使用的的CPU时间百分比

%MEM 进程使用的物理内存百分比

TIME+ 进程启动后到现在使用的全部CPU时间总计，精确到百分之一秒

COMMAND 运行进程使用的命令

可能的其它属性

列名 含义

PPID 父进程id

UID 进程所有者的用户id

USER 进程所有者的用户名

GROUP 进程所有者的组名

TTY 启动进程的终端名。不是从终端启动的进程则显示为 ?

P 最后使用的CPU，仅在多CPU环境下有意义

TIME 进程使用的CPU时间总计，单位秒

SWAP 进程使用的虚拟内存中，被换出的大小，单位kb。

CODE 可执行代码占用的物理内存大小，单位kb

DATA 可执行代码以外的部分(数据段+栈)占用的物理内存大小，单位kb

nFLT 页面错误次数

nDRT 最后一次写入到现在，被修改过的页面数。

WCHAN 若该进程在睡眠，则显示睡眠中的系统函数名

Flags 任务标志，参考 sched.h

### 其它交互：

h/? 帮助

l 控制第一行显示

t 控制cpu，task显示

m 切换内存显示

默认3秒刷新一次，空格或回车可以手动刷新

### 多U多核CPU监控

数字1：切换单个CPU状况：

|  |
| --- |
| %Cpu(s): 17.0 us, 3.3 sy, 0.0 ni, 79.3 id, 0.0 wa, 0.0 hi, 0.4 si, 0.0 st |
| %Cpu0 : 0.0 us, 0.0 sy, 0.0 ni,100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st  %Cpu1 : 0.0 us, 0.0 sy, 0.0 ni,100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st  %Cpu2 : 0.0 us, 0.0 sy, 0.0 ni,100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st  %Cpu3 : 0.0 us, 0.0 sy, 0.0 ni,100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st |

### 高亮，排序，字段设置

默认各进程是按照CPU的占用量来排序的

b：打开/关闭加亮效果

y：键关闭或打开运行态进程的加亮效果

x：打开/关闭排序列的加亮效果）

shift + >或shift + < 向右或左改变排序列

f：键，进入字段设置页,可是设置要显示的字段和默认排序字段

### 不足

top监控最新单位是进程，看不到Java线程数和客户连接数，可以通过ps和netstate来查看

## ps

pa -aux

## vmstat

统计系统CPU，内存，swap，io等情况

#vmstat s r

<http://www.51testing.com/html/48/202848-249833.html> good 实例分析

## kill -3 [pid]

查看线程的相关信息

<http://developer.51cto.com/art/201205/339006.htm>

JavaCore是关于CPU的

HeapDump是关于内存的

## pidstat

安装 sudo apt-get install sysstat

监控CPU,IO,Mem

pidstat -p pid -u[监控CPU] 每m秒一次 共n次

-t 显示线程

pidstat 2 5 //所有活动进程的CPU 使用情况

pidstat - p 3132 2 5 //进程1643的CPU使用情况显示

pidstat - p 3132 2 5 - r //进程1643的内存使用情况显示

|  |
| --- |
| [root@iZ23okbw3qkZ ~]# pidstat -p 15572 2 5 //显示进程信息  Linux 2.6.32-431.23.3.el6.i686 (iZ23okbw3qkZ) 09/13/2016 \_i686\_ (1 CPU)  08:21:25 PM PID %usr %system %guest %CPU CPU Command  08:21:27 PM 15572 99.00 0.00 0.00 99.00 0 java  08:21:29 PM 15572 99.50 0.00 0.00 99.50 0 java  Average: 15572 99.30 0.00 0.00 99.30 - java |

参考

<http://www.cnitblog.com/tjitty/archive/2010/07/12/67429.html> <http://www.cnblogs.com/bangerlee/articles/2555307.html>

Linux服务器性能数据收集【sar、pidstat、iostat、mpstat、sadf、sadc】

## pidstat -t 显示线程信息

|  |
| --- |
| [root@iZ23okbw3qkZ ~]# pidstat -p 15572 2 5 -t //显示线程信息  Linux 2.6.32-431.23.3.el6.i686 (iZ23okbw3qkZ) 09/13/2016 \_i686\_ (1 CPU)  08:37:48 PM TGID TID %usr %system %guest %CPU CPU Command  08:37:50 PM 15572 - 99.50 0.00 0.00 99.50 0 java  08:37:50 PM - 15572 0.00 0.00 0.00 0.00 0 |\_\_java  08:37:50 PM - 15573 99.00 0.00 0.00 99.00 0 |\_\_java  08:37:50 PM - 15574 0.00 0.00 0.00 0.00 0 |\_\_java  08:37:50 PM - 15575 0.00 0.00 0.00 0.00 0 |\_\_java  08:37:50 PM - 15576 0.00 0.00 0.00 0.00 0 |\_\_java  08:37:50 PM - 15577 0.00 0.00 0.00 0.00 0 |\_\_java  08:37:50 PM - 15578 0.00 0.00 0.00 0.00 0 |\_\_java  08:37:50 PM - 15579 0.00 0.00 0.00 0.00 0 |\_\_java  08:37:50 PM - 15580 0.00 0.00 0.00 0.00 0 |\_\_java |

15573 线程比较忙

## pidstat -d 显示磁盘信息

|  |
| --- |
| [root@iZ23okbw3qkZ ~]# pidstat -p 15572 2 5 -d  Linux 2.6.32-431.23.3.el6.i686 (iZ23okbw3qkZ) 09/13/2016 \_i686\_ (1 CPU)  08:42:56 PM PID kB\_rd/s kB\_wr/s kB\_ccwr/s Command  08:42:58 PM 15572 0.00 0.00 0.00 java  08:43:00 PM 15572 0.00 0.00 0.00 java  08:43:02 PM 15572 0.00 0.00 0.00 java |

## Sar

sar（System Activity Reporter系统活动情况报告）是目前 Linux 上最为全面的系统性能分析工具之一，可以从多方面对系统的活动进行报告，包括：文件的读写情况、系统调用的使用情况、磁盘I/O、CPU效率、内存使用状况、进程活动及IPC有关的活动等。

sar [options] [-A] [-o file] t [n]

其中：

t为采样间隔

n为采样次数，默认值是1

-o file表示将命令结果以二进制格式存放在文件中，file 是文件名

options 为命令行选项，sar命令常用选项如下：

|  |
| --- |
| [powerop@FATD00041462 bin]$ sar --help  Usage: sar [ options ] [ <interval> [ <count> ] ]  Options are:  [ -A ] [ -B ] [ -b ] [ -C ] [ -d ] [ -H ] [ -h ] [ -p ] [ -q ] [ -R ]  [ -r ] [ -S ] [ -t ] [ -u [ ALL ] ] [ -V ] [ -v ] [ -W ] [ -w ] [ -y ]  [ -I { <int> [,...] | SUM | ALL | XALL } ] [ -P { <cpu> [,...] | ALL } ]  [ -m { <keyword> [,...] | ALL } ] [ -n { <keyword> [,...] | ALL } ]  [ -j { ID | LABEL | PATH | UUID | ... } ]  [ -f [ <filename> ] | -o [ <filename> ] | -[0-9]+ ]  [ -i <interval> ] [ -s [ <hh:mm:ss> ] ] [ -e [ <hh:mm:ss> ] ] |

sar 2 5 //查看CPU使用情况

sar 2 5 - r //查看内存使用情况

sar 2 5 - n DEV //查看网络吞吐量

参考

<http://www.cnitblog.com/tjitty/archive/2010/07/12/67429.html>

Linux服务器性能数据收集【sar、pidstat、iostat、mpstat、sadf、sadc】

<http://blog.csdn.net/techsupporter/article/details/55194600>

Linux统计/监控工具SAR详细介绍

## nmon

http://os.51cto.com/art/201406/442795.htm

## cpulimit

用于限制进程

## 监控java线程数：

ps -eLf | grep java | wc -l

## 监控网络客户连接数：

netstat -n | grep tcp | grep 侦听端口 | wc -l

## xxx

# windows工具

任务管理器

perfmon

Windows自带性能监视器

Process Explorer

第三方工具

pslist

第三方命令行工具，可用于脚本自动化数据收集

显示java程序的运行情况

-d也可以看到线程信息；

# jdk自带工具

## jps虚拟机进程状况工具

\*与unix的ps类似，查看本地运行的java进程：

\*在root用户下显示所有进程，在user用户下显示user的进程。

\*运行该命令需要JAVA\_HOME

\*可以通过RMI协议查询远程虚拟机进程状态

参数：

-q 只输出进程ID，不输出类名称

-m 输出传递给main函数的参数

-l 输出主函数的完整路径，如果进程执行的是Jar，输出Jar路径

-v 输出JVM参数

|  |
| --- |
| [root@AY130305094734633c36 ~]# jps -help  usage: jps [-help]  jps [-q] [-mlvV] [<hostid>]  Definitions:  <hostid>: <hostname>[:<port>] |

|  |
| --- |
| [root@AY130305094734633c36 ~]# jps  10342 Jps  5177 QuorumPeerMain  12936 Bootstrap |

|  |
| --- |
| [root@AY130305094734633c36 ~]# jps -q  5177  12936  9812 |

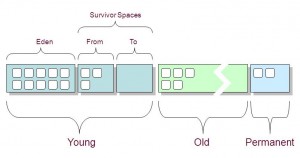
|  |
| --- |
| [root@AY130305094734633c36 ~]# jps -m  9862 Jps-m  5177 QuorumPeerMain/data/paas/zookeeper/zookeeper-3.4.5/bin/../conf/zoo.cfg  12936 Bootstrap start |

|  |
| --- |
| [root@AY130305094734633c36 ~]# jps -l  5177 org.apache.zookeeper.server.quorum.QuorumPeerMain  9888 sun.tools.jps.Jps  12936org.apache.catalina.startup.Bootstrap |

|  |
| --- |
| [root@AY130305094734633c36 ~]# jps -v  5177 QuorumPeerMain-Dzookeeper.log.dir=. -Dzookeeper.root.logger=INFO,CONSOLE -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.local.only=false  12936 Bootstrap -Djava.util.logging.config.file=/usr/local/tomcat7/conf/logging.properties -Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager -Xms2048m -Xmx2048m -XX:PermSize=1024m -XX:MaxPermSize=1024m -agentlib:jdwp=transport=dt\_socket,address=8000,server=y,suspend=n -Djava.endorsed.dirs=/usr/local/tomcat7/endorsed -Dcatalina.base=/usr/local/tomcat7 -Dcatalina.home=/usr/local/tomcat7 -Djava.io.tmpdir=/usr/local/tomcat7/temp |

|  |
| --- |
| [root@AY130305094734633c36 ~]# jps -mlv  10056 sun.tools.jps.Jps -mlv -Dapplication.home=/usr/local/jdk1.7.0\_45 -Xms8m  5177 org.apache.zookeeper.server.quorum.QuorumPeerMain /data/paas/zookeeper/zookeeper-3.4.5/bin/../conf/zoo.cfg -Dzookeeper.log.dir=. -Dzookeeper.root.logger=INFO,CONSOLE -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.local.only=false  12936 org.apache.catalina.startup.Bootstrap start -Djava.util.logging.config.file=/usr/local/tomcat7/conf/logging.properties -Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager -Xms2048m -Xmx2048m -XX:PermSize=1024m -XX:MaxPermSize=1024m -agentlib:jdwp=transport=dt\_socket,address=8000,server=y,suspend=n -Djava.endorsed.dirs=/usr/local/tomcat7/endorsed -Dcatalina.base=/usr/local/tomcat7 -Dcatalina.home=/usr/local/tomcat7 -Djava.io.tmpdir=/usr/local/tomcat7/temp |

## jstat虚拟机统计信息监视工具



\* jstat(Java Virtual Machine statistics monitoring tool)

\*监视虚拟机各种运行状态信息的命令行工具

各种堆和非堆的大小及其内存使用量;

详细查看堆内各个部分的使用量，以及类装载，垃圾收集，JIT等;

包括了对Heap size和垃圾回收状况的监控;

启动jvm监控服务，是一个基于rmi的应用，向远程机器提供本地jvm信息，默认端口1099

语法结构如下：jstat [Options] vmid [interval] [count]

Options 选项，我们一般使用 -gcutil查看gc情况

vmid VM的进程号，即当前运行的java进程号

interval 间隔时间，单位为秒或者毫秒

count   打印次数，如果缺省则打印无数次

Options:表示用户希望查询的虚拟机信息，主要分3类：类装载，垃圾收集，运行期编译情况。

-gccapacity:

### jstat -help

|  |
| --- |
| ums@C113649 logs]$ jstat -help  Usage: jstat -help|-options  jstat -<option> [-t] [-h<lines>] <vmid> [<interval> [<count>]]  Definitions:  <option> An option reported by the -options option  <vmid> Virtual Machine Identifier. A vmid takes the following form:  <lvmid>[@<hostname>[:<port>]]  Where <lvmid> is the local vm identifier for the target  Java virtual machine, typically a process id; <hostname> is  the name of the host running the target Java virtual machine;  and <port> is the port number for the rmiregistry on the  target host. See the jvmstat documentation for a more complete  description of the Virtual Machine Identifier.  <lines> Number of samples between header lines.  <interval> Sampling interval. The following forms are allowed:  <n>["ms"|"s"]  Where <n> is an integer and the suffix specifies the units as  milliseconds("ms") or seconds("s"). The default units are "ms".  <count> Number of samples to take before terminating.  -J<flag> Pass <flag> directly to the runtime system. |

### jstat -class

监视类装载、卸载数量，及所占空间，类装载所消耗的时间；

|  |
| --- |
| [guru@C114728 ~]$ jstat -class 10155  Loaded Bytes Unloaded Bytes Time  5206 10634.3 0 0.0 0.84 |

### jstat -compiler

显示VM实时编译的数量等信息。

|  |
| --- |
| [guru@C114728 ~]$ jstat -compiler 10155  Compiled Failed Invalid Time FailedTypeFailedMethod  903 0 0 12.49 0 |

### jstat -gc

\*监视java堆状况，包括Eden,survivor,年老代,永久代等的容量，已用空间，GC时间等信息

\*可以显示gc的信息，查看gc的次数，及时间。其中最后五项，分别是young gc的次数，young gc的时间，full gc的次数，full gc的时间，gc的总时间。

|  |
| --- |
|  |

jstat -gc 26365 2000 100

### jstat -gcutil

除了以上一个参数外，还可以同时加上两个数字，如：jstat -printcompilation 3024 250 6是每250毫秒打印一次，一共打印6次，还可以加上-h3每三行显示一下标题。

|  |
| --- |
| [root@C114728 ~]# jstat -gcutil 29640 1000  S0 S1 E O P YGC YGCT FGC FGCT GCT  88.58 0.00 55.81 32.43 60.53 1252 6.779 0 0.000 6.779  88.58 0.00 55.81 32.43 60.53 1252 6.779 0 0.000 6.779  88.58 0.00 55.86 32.43 60.53 1252 6.779 0 0.000 6.779 |

29640：进程号

1000： 每1000毫秒打印一条记录

S0 -Heap上的 Survivor space 0 区已使用空间的百分比

S1 -Heap上的 Survivor space 1 区已使用空间的百分比

E -Heap上的 Eden space 区已使用空间的百分比

O -Heap上的 Old space 区已使用空间的百分比

P -Perm space 区已使用空间的百分比

YGC -从应用程序启动到采样时发生 Young GC 的次数

YGCT-从应用程序启动到采样时 Young GC 所用的时间(单位秒)

FGC -从应用程序启动到采样时发生 Full GC 的次数

FGCT-从应用程序启动到采样时 Full GC 所用的时间(单位秒)

GCT -从应用程序启动到采样时用于垃圾回收的总时间(单位秒)

### jstat -gccapacity:

可以显示，VM内存中三代（young,old,perm）对象的使用和占用大小，如：PGCMN显示的是最小perm的内存使用量，PGCMX显示的是perm的内存最大使用量，PGC是当前新生成的perm内存占用量，PC是但前perm内存占用量。其他的可以根据这个类推， OC是old内纯的占用量.

|  |
| --- |
|  |

### jstat -gcnew pid:

new对象的信息。

|  |
| --- |
| [guru@C114728 ~]$ jstat -gcnew 10155  S0C S1C S0U S1U TTMTT DSS EC EU YGC YGCT  45312.0131072.00.040765.361545312.0 786432.0 391433.1 5 0.151 |

jstat -gcnew26365 2000100

### jstat -gcnewcapacity pid:

new对象的信息及其占用量。

|  |
| --- |
| [guru@C114728 ~]$ jstat -gcnewcapacity 10155  NGCMN NGCMX NGC S0CMX S0C S1CMX S1C ECMX EC YGC FGC  1048576.01048576.0 1048576.045312.0 349504.0 349504.0 131072.0 1048448.0786432.050 |

|  |
| --- |
|  |

### jstat -gcold pid:

old对象的信息。

|  |
| --- |
| [guru@C114728 ~]$ jstat -gcold 10155  PC PU OC OU YGC FGC FGCT GCT  65536.0 37631.6 2097152.0 0.0 5 0 0.000 0.151 |

### jstat -gcoldcapacity pid:

old对象的信息及其占用量。

|  |
| --- |
| [guru@C114728 ~]$ jstat -gcoldcapacity 10155  OGCMN OGCMX OGC OC YGC FGC FGCT GCT  2097152.0 2097152.0 2097152.0 2097152.0 5 0 0.000 0.151 |

### jstat -gcpermcapacity pid:

perm对象的信息及其占用量。

|  |
| --- |
| [guru@C114728 ~]$ jstat -gcpermcapacity 10155  PGCMN PGCMX PGC PC YGC FGC FGCT GCT  65536.0 86016.0 65536.0 65536.0 5 0 0.000 0.151 |

### jstat -gcutil pid:

统计gc信息统计。

jstat -gcutil26365 2000 100

|  |
| --- |
| [guru@C114728 ~]$ jstat -gcutil 10155  S0 S1 E O P YGC YGCT FGC FGCT GCT  0.00 31.10 50.90 0.00 57.42 5 0.151 0 0.000 0.151 |

### jstat -printcompilation pid:

当前VM执行的信息。

|  |
| --- |
| [guru@C114728 ~]$ jstat -printcompilation 10155  Compiled Size Type Method  903 248 1 com/sun/messaging/jmq/io/ReadOnlyPacketreadFully |

## jstatd

## jinfo

查看参数，可修改部分参数

|  |
| --- |
| [admin@localhost ~]$ jinfo 8095  Attaching to process ID 8095, please wait...  Debugger attached successfully.  Server compiler detected.  JVM version is 16.3-b01  Java System Properties:  java.runtime.name = Java(TM) SE Runtime Environment  sun.boot.library.path = /usr/local/jdk1.6.0\_20/jre/lib/amd64  java.vm.version = 16.3-b01  java.vm.vendor = Sun Microsystems Inc.  java.vendor.url = http://java.sun.com/  path.separator = :  java.vm.name = Java HotSpot(TM) 64-Bit Server VM  file.encoding.pkg = sun.io  sun.java.launcher = SUN\_STANDARD  user.country = US  sun.os.patch.level = unknown  java.vm.specification.name = Java Virtual Machine Specification  user.dir = /usr/local/jetty  java.runtime.version = 1.6.0\_20-b02  java.awt.graphicsenv = sun.awt.X11GraphicsEnvironment  java.endorsed.dirs = /usr/local/jdk1.6.0\_20/jre/lib/endorsed  os.arch = amd64  webapp.root = /home/admin/relation2/output/tmp/Jetty\_0\_0\_0\_0\_8080\_root.war\_\_\_.whto0i/webapp  java.io.tmpdir = /home/admin/relation2/deploy/bin/../../output/tmp  line.separator =  jetty.http.port = 8080  java.vm.specification.vendor = Sun Microsystems Inc.  os.name = Linux  sun.jnu.encoding = UTF-8  java.library.path = /home/admin/relation2/deploy/native/pt2sd/linux-x64  java.specification.name = Java Platform API Specification  java.class.version = 50.0  jetty.home = /usr/local/jetty  sun.management.compiler = HotSpot 64-Bit Server Compiler  os.version = 2.6.18-164.el5  project.output = /home/admin/relation2/deploy/bin/../../output  user.home = /home/admin  user.timezone = Asia/Shanghai  java.awt.printerjob = sun.print.PSPrinterJob  file.encoding = UTF-8  java.specification.version = 1.6  java.class.path = /usr/local/jetty/lib/jetty-xml-7.0.1.v20091125.jar:/usr/local/jetty/lib/servlet-api-2.5.jar:/usr/local/jetty/lib/jetty-http-7.0.1.v20091125.jar:/usr/local/jetty/lib/jetty-continuation-7.0.1.v20091125.jar:/usr/local/jetty/lib/jetty-server-7.0.1.v20091125.jar:/usr/local/jetty/lib/jetty-security-7.0.1.v20091125.jar:/usr/local/jetty/lib/jetty-servlet-7.0.1.v20091125.jar:/usr/local/jetty/lib/jetty-webapp-7.0.1.v20091125.jar:/usr/local/jetty/lib/jetty-deploy-7.0.1.v20091125.jar:/usr/local/jetty/lib/jetty-servlets-7.0.1.v20091125.jar:/usr/local/jetty/lib/jetty-annotations-7.0.1.v20091125.jar:/usr/local/jetty/lib/annotations/asm-3.1.jar:/usr/local/jetty/lib/annotations/asm-commons-3.1.jar:/usr/local/jetty/lib/annotations/asm-tree-3.1.jar:/usr/local/jetty/lib/annotations/geronimo-annotation\_1.0\_spec-1.1.1.jar:/usr/local/jetty/lib/jdbc/derby/derby-10.4.1.3.jar:/usr/local/jetty/lib/jdbc/derby/derbytools-10.4.1.3.jar:/usr/local/jetty/lib/jsp/ant-1.6.5.jar:/usr/local/jetty/lib/jsp/core-3.1.1.jar:/usr/local/jetty/lib/jsp/jetty-jsp-2.1-7.0.1.v20091125.jar:/usr/local/jetty/lib/jsp/jsp-2.1-glassfish-9.1.1.B60.25.p2.jar:/usr/local/jetty/lib/jsp/jsp-api-2.1-glassfish-9.1.1.B60.25.p2.jar:/usr/local/jetty/lib/jta/geronimo-jta\_1.1\_spec-1.1.1.jar:/usr/local/jetty/lib/jta/atomikos/atomikos-util-3.5.5.jar:/usr/local/jetty/lib/jta/atomikos/transactions-3.5.5.jar:/usr/local/jetty/lib/jta/atomikos/transactions-api-3.5.5.jar:/usr/local/jetty/lib/jta/atomikos/transactions-jdbc-3.5.5.jar:/usr/local/jetty/lib/jta/atomikos/transactions-jta-3.5.5.jar:/usr/local/jetty/lib/jetty-jndi-7.0.1.v20091125.jar:/usr/local/jetty/lib/jetty-plus-7.0.1.v20091125.jar:/usr/local/jetty/lib/jndi/activation-1.1.jar:/usr/local/jetty/lib/jndi/mail-1.4.jar:/usr/local/jetty/resources:/usr/local/jetty/lib/jetty-websocket-7.0.1.v20091125.jar:/usr/local/jetty/lib/jetty-util-7.0.1.v20091125.jar:/usr/local/jetty/lib/jetty-io-7.0.1.v20091125.jar  user.name = admin  java.vm.specification.version = 1.0  java.home = /usr/local/jdk1.6.0\_20/jre  sun.arch.data.model = 64  user.language = en  java.specification.vendor = Sun Microsystems Inc.  java.vm.info = mixed mode  java.version = 1.6.0\_20  java.ext.dirs = /usr/local/jdk1.6.0\_20/jre/lib/ext:/usr/java/packages/lib/ext  sun.boot.class.path = /usr/local/jdk1.6.0\_20/jre/lib/resources.jar:/usr/local/jdk1.6.0\_20/jre/lib/rt.jar:/usr/local/jdk1.6.0\_20/jre/lib/sunrsasign.jar:/usr/local/jdk1.6.0\_20/jre/lib/jsse.jar:/usr/local/jdk1.6.0\_20/jre/lib/jce.jar:/usr/local/jdk1.6.0\_20/jre/lib/charsets.jar:/usr/local/jdk1.6.0\_20/jre/classes  java.vendor = Sun Microsystems Inc.  file.separator = /  java.vendor.url.bug = http://java.sun.com/cgi-bin/bugreport.cgi  sun.io.unicode.encoding = UnicodeLittle  sun.cpu.endian = little  project.home = /home/admin/relation2/deploy  jetty.port = 8080  sun.cpu.isalist =  VM Flags:  -Xms3072m -Xmx3072m -Xss512k -XX:PermSize=64m -XX:HeapDumpPath=/home/admin/relation2/deploy/bin/../../output -XX:+HeapDumpOnOutOfMemoryError -Dproject.home=/home/admin/relation2/deploy -Dproject.output=/home/admin/relation2/deploy/bin/../../output -Djava.io.tmpdir=/home/admin/relation2/deploy/bin/../../output/tmp -Djava.library.path=/home/admin/relation2/deploy/native/pt2sd/linux-x64 -Djetty.home=/usr/local/jetty -Djetty.port=8080 -Djetty.http.port=8080 |

## jinfo -flag <name> pid 查看修改指定参数

jinfo -flag <name> 打印指定JVM的参数值

jinfo -flag [+|-]<name> 设置JVM参数布尔值

jinfo -flag <name>=<value> 设置JVM参数值

jinfo -flags 5056

|  |
| --- |
| D:\app\PowerCmd22>jinfo -h  Usage:  jinfo [option] <pid>  (to connect to running process)  jinfo [option] <executable <core>  (to connect to a core file)  jinfo [option] [server\_id@]<remote server IP or hostname>  (to connect to remote debug server)  where <option> is one of:  -flag <name> to print the value of the named VM flag  -flag [+|-]<name> to enable or disable the named VM flag  -flag <name>=<value> to set the named VM flag to the given value  -flags to print VM flags  -sysprops to print Java system properties  <no option> to print both of the above  -h | -help to print this help message  # jinfo -flag MaxTenuringThreshold 6431  -XX:MaxTenuringThreshold=15  # jinfo -flag PrintGCDetails 6431  -XX:-PrintGCDetails  # jinfo -flag +PrintGCDetails 6431 //运行时修改参数 |

## jmap 堆快照

统计对象信息, 生产java进程堆快照, 输出所有内存中对象，甚至可以将VM 中的heap，以二进制输出成文本。

dump出来的文件可以用MAT、VisualVM，jhat（Java Heap Analysis Tool）等工具查看

这里用jhat查看：

# jhat -port 9998 f1

### jmap -dump pid

jmap -dump:format=b,file=c:\\heap.bin 302

4可以将3024进程的内存heap输出到heap.bin文件里。

将内存信息dump到文件

|  |
| --- |
| [guru@C114728 ~]$ jps  9978 Jps  10155 start.jar  [guru@C114728 ~]$ jmap -dump:file=f10155.bin 10155  Dumping heap to /home/guru/f10155.bin ...  Heap dump file created |

参考

<https://my.oschina.net/feichexia/blog/196575>

### jmap -permstat pid

打印进程的类加载器和类加载器加载的持久代对象信息，输出：类加载器名称、对象是否存活（不可靠）、对象地址、父类加载器、已加载的类大小等信息，如下图：

### jmap -heap pid

查看进程堆内存使用情况，包括使用的GC算法、堆配置参数和各代中堆内存使用情况。

### jmap -histo pid

统计java进程内存中所有对象信息，如：产生那些对象，及其数量;

间隔用jmap -histo pid>ax.log使用文本对比工具,可以对比GC回收了哪些对象;

jmap -histo[:live] pid查看堆内存中的对象数目、大小统计直方图，如果带上live则只统计活对象，如下：

root@ubuntu:/# jmap -histo 21711 > c:/s.txt

root@ubuntu:/# jmap -histo:live 21711 | more

## jhat

Java Head Analyse Tool

是用来分析java堆的命令

# jhat heap\_file 默认端口7000

# jhat -port 9998 heap\_file 查看http://localhost:9998/

## jstack

jstack 线程栈信息，jmap堆信息

jstack <pid> >> t.txt

-l 打印锁信息

-m 打印java和native的帧信息

-F 强制dump，当jstack没有响应时使用

示例：

结果说明：中的tid是jvm中线程ID，nid是操作系统中的id，也就是nativeID，且都以16进制格式显示

参考命令：linux 10制转16进命令：printf “%x\n” pid

|  |
| --- |
| [root@iZ23okbw3qkZ ~]# ps -mp 15572 -o THREAD,tid,time  USER %CPU PRI SCNT WCHAN USER SYSTEM TID TIME  root 99.3 - - - - - - 01:13:24  root 99.3 19 - - - - 15573 01:13:22  ……  root 0.0 19 - - - - 15580 00:00:02  [root@iZ23okbw3qkZ ~]# jstack 15572  2016-09-13 21:35:03  Full thread dump Java HotSpot(TM) Client VM (25.91-b14 mixed mode):  "Attach Listener" #7 daemon prio=9 os\_prio=0 tid=0xb768dc00 nid=0x3dd8 waiting on condition [0x00000000]  java.lang.Thread.State: RUNNABLE  "Service Thread" #6 daemon prio=9 os\_prio=0 tid=0xb7683400 nid=0x3cdb runnable [0x00000000]  java.lang.Thread.State: RUNNABLE  "C1 CompilerThread0" #5 daemon prio=9 os\_prio=0 tid=0xb767fc00 nid=0x3cda waiting on condition [0x00000000]  java.lang.Thread.State: RUNNABLE  "Signal Dispatcher" #4 daemon prio=9 os\_prio=0 tid=0xb767e400 nid=0x3cd9 runnable [0x00000000]  java.lang.Thread.State: RUNNABLE  "Finalizer" #3 daemon prio=8 os\_prio=0 tid=0xb7664000 nid=0x3cd8 in Object.wait() [0xa4fad000]  java.lang.Thread.State: WAITING (on object monitor)  at java.lang.Object.wait(Native Method)  - waiting on <0xa5807ed8> (a java.lang.ref.ReferenceQueue$Lock)  at java.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:143)  - locked <0xa5807ed8> (a java.lang.ref.ReferenceQueue$Lock)  at java.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:164)  at java.lang.ref.Finalizer$FinalizerThread.run(Finalizer.java:209)  "Reference Handler" #2 daemon prio=10 os\_prio=0 tid=0xb7661400 nid=0x3cd7 in Object.wait() [0xa4ffe000]  java.lang.Thread.State: WAITING (on object monitor)  at java.lang.Object.wait(Native Method)  - waiting on <0xa5805f48> (a java.lang.ref.Reference$Lock)  at java.lang.Object.wait(Object.java:502)  at java.lang.ref.Reference.tryHandlePending(Reference.java:191)  - locked <0xa5805f48> (a java.lang.ref.Reference$Lock)  at java.lang.ref.Reference$ReferenceHandler.run(Reference.java:153)  "main" #1 prio=5 os\_prio=0 tid=0xb7606400 nid=0x3cd5 runnable [0xb7771000]  java.lang.Thread.State: RUNNABLE  at T.main(T.java:6)  "VM Thread" os\_prio=0 tid=0xb765cc00 nid=0x3cd6 runnable  "VM Periodic Task Thread" os\_prio=0 tid=0xb7685800 nid=0x3cdc waiting on condition  JNI global references: 7 |

## jstackd

jvm监控服务，基于rmi的应用，提供本机jvm应用程序的信息。默认端口1099。

实例：jstatd -J-Djava.security.policy=my.policy

my.policy文件需要自己建立，内如如下：

grant codebase "file:$JAVA\_HOME/lib/tools.jar" {

permissionjava.security.AllPermission;

};

这是安全策略文件，因为jdk对jvm做了jaas的安全检测，所以我们必须设置一些策略，使得jstatd被允许作网络操作

## jconsole xx

**jconsole** [ *options* ] [ *pid* | [ *host*:*port* ]]

参考：http://afunti.javaeye.com/blog/785523

在启动脚本中开启远程监控要添加启动参数：

-Djava.rmi.server.hostname=10.241.14.41

-Dcom.sun.management.jmxremote

-Dcom.sun.management.jmxremote.port=19005

-Dcom.sun.management.jmxremote.authenticate=false

-Dcom.sun.management.jmxremote.ssl=false

|  |
| --- |
| -Djava.rmi.server.hostname=10.241.14.41 -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.port=18080 -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false |

|  |
| --- |
| -Djava.rmi.server.hostname=192.168.39.1 -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.port=18080 -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false |

netstat -na|grep18080

SUNJMX="-Dcom.sun.management.jmxremote.port=9999 -   
Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false"   
fi

-Dcom.sun.management.jmxremote.password.file=/usr/local/jdk1.5.0/jconsole\_pwd/jmxremote.password

或修改localhost对应到真实ip

vi /etc/hosts

|  |
| --- |
| java -Dcom.sun.management.jmxremote.port=8999 -Dcom.sun.management.jmxremote.authenticate=**false** -Dcom.sun.management.jmxremote.ssl=**false** com.yourpackage.yourApp |

## jvisualvm(Java VisualVM)

java/bin/jvisualvm.exe

监控项总共分为Overview，Monitor，Threads和一个Sampler。

1.Overview(概述:工作环境，jvm启动参数，系统参数)

2.Monitor(监视：CPU,内存/GC，类，线程)

3.Threads(线程：名字，状态，)

线程状态(运行，休眠，等待，监视)，运行时间

可分析：CPU，内存，类，线程，死锁

参考:

<http://blog.csdn.net/a19881029/article/details/8432368>

Java jvisualvm简要说明

远程监控App或tomcat

<http://www.cnblogs.com/wade-xu/p/4369094.html>

性能分析神器VisualVM

<https://www.ibm.com/developerworks/cn/java/j-lo-visualvm/>

<http://www.cnblogs.com/amosli/p/3901794.html>

Java监控工具介绍，VisualVm ,JProfiler,Perfino,Yourkit,Perf4J,JProbe,Java微基准测试

## VisualVM？

https://visualvm.github.io/

APP

IDE plugin

## jdb

命令行调试工具

<http://blog.csdn.net/arkblue/article/details/39718947>

## tda

## javah

## hprof

hprof（Heap/CPU Profiling Tool）

hprof能够展现CPU使用率，统计堆内存使用情况。

语法格式如下：

java -agentlib:hprof[=options] ToBeProfiledClass

java -Xrunprof[:options] ToBeProfiledClass

javac -J-agentlib:hprof[=options] ToBeProfiledClass

## JMC

JRocket Mission Control

Jrocket自带的内存监控工具，据说不错

# 其它工具

## jvmtop

<https://code.google.com/p/jvmtop/wiki/ExampleOutput>

下载wget<https://jvmtop.googlecode.com/files/jvmtop-0.7.1.tar.gz>

## BTrace

## HuseMD

## TProfiler

## pslist

命令行，可自动化数据收集，显示java程序运行情况

## JMC

# OOM

操作系统可分配内存 = 对空间+线程栈空间+直接内存

## 堆溢出

public static void main(String args[]){

ArrayList<byte[]> list=new ArrayList<byte[]>();

for(int i=0;i<1024;i++){

list.add(new byte[1024\*1024]);

}

}

Exception in thread "main" java.lang.OutOfMemoryError: Java heap space

at geym.jvm.ch8.oom.SimpleHeapOOM.main(SimpleHeapOOM.java:14)

## 栈溢出

创建一万个线程

可以减少堆空间

for(int i=0;i<1000;i++){

new Thread(new SleepThread(),"Thread"+i).start();

System.out.println("Thread"+i+" created");

}

Exception in thread "main" java.lang.OutOfMemoryError:

unable to create new native thread

## 永久区溢出

for(int i=0;i<100000;i++){

CglibBean bean = new CglibBean("geym.jvm.ch3.perm.bean"+i,new HashMap());

}

Caused by: java.lang.OutOfMemoryError: **PermGen space**

[Full GC[Tenured: 2523K->2523K(10944K), 0.0125610 secs] 2523K->2523K(15936K),

[Perm : **4095K->4095K(4096K)**], 0.0125868 secs] [Times: user=0.02 sys=0.00, real=0.01 secs]

Heap

def new generation total 4992K, used 89K [0x28280000, 0x287e0000, 0x2d7d0000)

eden space 4480K, 2% used [0x28280000, 0x282966d0, 0x286e0000)

from space 512K, 0% used [0x286e0000, 0x286e0000, 0x28760000)

to space 512K, 0% used [0x28760000, 0x28760000, 0x287e0000)

tenured generation total 10944K, used 2523K [0x2d7d0000, 0x2e280000, 0x38280000)

the space 10944K, 23% used [0x2d7d0000, 0x2da46cf0, 0x2da46e00, 0x2e280000)

compacting perm gen total 4096K, used 4095K [0x38280000, 0x38680000, 0x38680000)

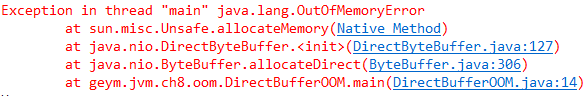
the space 4096K, **99%** used [0x38280000, 0x3867fff0, 0x38680000, 0x38680000)

ro space 10240K, 44% used [0x38680000, 0x38af73f0, 0x38af7400, 0x39080000)

rw space 12288K, 52% used [0x39080000, 0x396cdd28, 0x396cde00, 0x39c80000)

## 直接内存溢出

ByteBuffer.allocateDirect(1024\*1024);



解决办法：加大内存，或减少堆空间

# 分析dump文件-MAT

参考：

<http://blog.csdn.net/aaa2832/article/details/19419679/>

内存分析工具 MAT 的使用

<http://www.blogjava.net/rosen/archive/2010/05/21/321575.html>

使用Memory Analyzer tool(MAT)分析内存泄漏（一）

<http://www.blogjava.net/rosen/archive/2010/06/13/323522.html>

使用Memory Analyzer tool(MAT)分析内存泄漏（二）

Java堆分析

MAT：Memory Analyzer <http://www.eclipse.ogr/mat>

浅堆Shallow Heap，深堆Retained Heap

入应用incoming，出引用outgoing

对象应用图，支配树

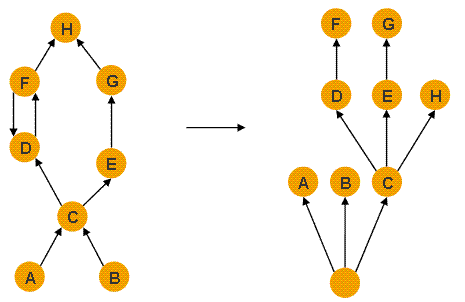
使用Visual VM分析堆

Tomcat OOM分析

## 对象引用图，支配树

要到B必须经过A，A就是B的支配对象；

A离B最近，A是B的的直接支配对象



引用图 支配树

支配者被回收，则支配的对象也被回收

## MAT

对象应用的对象

应用当前对象的对象

Threaddump

# 参数设置:

http://calvin1978.blogcn.com/articles/jvmoption-2.html

# 参考:

## CPU 100%调查1

pidstat(第三方linux工具) 找到最忙的线程id

pidstat -p pid 1 3 -u -t

后续如下：

## CPU 100%调查2

1 查找进程

根据top命令找到CPU高的进程ID，如： xxxx

2 首先显示线程列表,并按照CPU占用高的线程排序：

ps -mp xxxx -o THREAD,tid,time | sort –rn

显示结果如下：

USER %CPU PRI SCNT WCHAN USER SYSTEM TID TIME

root 10.5 19 - - - - 3626 00:12:48

root 10.1 19 - - - - 3593 00:12:16

找到了耗时最高的线程3626

或者

ps -Lfp xxx

top -p xxx再shift+h

top -Hp xxx

3 打印线程的堆栈信息

printf "%x\n" 3626

e18

jstack xxxx | grep e18 -A 30

结果

"main" #1 prio=5 os\_prio=0 tid=0xb7606400 nid=0x3ad1 runnable [0xb777e000]

java.lang.Thread.State: RUNNABLE

at T.main(T.java:6)

## 死锁检测

jstack

wait for…

Locked …

## 进程文件

创建：

java程序启动后，默认会在/tmp/hsperfdata\_userName目录下以该进程的id为文件名新建文件/tmp/hsperfdata\_userName/pid，并在该文件中存储jvm运行的相关信息，而jps、jconsole、jvisualvm等工具的数据来源就是这个文件；

位置：

设置该文件位置的参数为-Djava.io.tmpdir=/tmp

/tmp/hsperfdata\_userName目录会存放该用户所有已经启动的java进程信息：/tmp/hsperfdata\_userName/pid

所以当该文件不存在或是无法读取时就会出现jps无法查看该进程号，jconsole无法监控等问题

关于设置该文件位置的参数为-Djava.io.tmpdir=/tmp

清除：

/tmp/hsperfdata\_userName/pid文件会在对应java进程退出后被清除。

如果java进程非正常退出(如kill -9)，那么pid文件会被保留，直到执行一次java命令或是加载了jvm程序的命令(如jps、javac、jstat)，会将所有无用的pid文件都清除掉

## 关键字

**Introduction,**MochiWeb

## 词汇

Eden 

亦作 Garden of Eden，乐园，天堂；最大幸福，至乐。(圣经)伊甸园

Survivor

n幸存者;残存者;生还者

|  |
| --- |
| a survivor from the Titanic |

|  |  |
| --- | --- |
|  | 泰坦尼克号的幸存者 |

tenure

保有，保有期, 保有状况, 保有权, 保有条件；任期

garbage collection : GC,垃圾回收

## cmd

tail -n500 exception.log

tail -n500 root.log

w

df

killall -3 java

vi output/logs/webserver.log

sz output/logs/webserver.log

[admin@localhost logs]$ ulimit -a

core file size (blocks, -c) 0

dataseg size (kbytes, -d) unlimited

scheduling priority (-e) 0

file size (blocks, -f) unlimited

pending signals (-i) 81920

max locked memory (kbytes, -l) 32

max memory size (kbytes, -m) unlimited

open files (-n) 1024

pipe size (512 bytes, -p) 8

POSIX message queues (bytes, -q) 819200

real-time priority (-r) 0

stack size (kbytes, -s) 10240

cpu time (seconds, -t) unlimited

max user processes (-u) 81920

virtual memory (kbytes, -v) unlimited

file locks (-x) unlimited

==============================

top/w 看负载

jps/psux | grep java 看进程

jstat -gcutil进程号 1000（每秒） 垃圾回收

jmap -heap 进程号

jmap -dump:file=1.bin 进程号

vmstat 1（每秒） 当前任务数

cat /proc/cpuinfo 查看cpu信息

|  |
| --- |
| temp |

|  |
| --- |
|  |

## 参考add

Java内存泄露监控工具：JVM监控工具介绍及分析

http://developer.51cto.com/art/201203/321461.htm

Java几款性能分析工具的对比

http://developer.51cto.com/art/201204/327130.htm

http://dev.firnow.com/course/3\_program/java/javajs/2008318/104919.html

jps:http://java.sun.com/j2se/1.5.0/docs/tooldocs/share/jps.html

jstat:http://java.sun.com/j2se/1.5.0/docs/tooldocs/share/jstat.html

jmap:http://java.sun.com/j2se/1.5.0/docs/tooldocs/share/jmap.html

jconsole:http://java.sun.com/j2se/1.5.0/docs/guide/management/jconsole.html

JVM调优总结 -Xms -Xmx -Xmn–Xss

<http://dev.firnow.com/course/3_program/java/javajs/2008630/129292.html>

JVM内存管理：深入垃圾收集器与内存分配策略

http://www.javaeye.com/topic/802638

<http://www.javaeye.com/topic/212967>

记一次Java GC调整经历

http://arbow.javaeye.com/blog/362861

主题：CMS gc实践总结

<http://www.javaeye.com/topic/473874>

Java几款性能分析工具的对比

<http://developer.51cto.com/art/201203/326355.htm>

http://www.boobooke.com/v/bbk3399/ java背后的秘密，背后原理，原则和方法

http://www.iteye.com/topic/752755

http://zhaozhi.iteye.com/category/90261深入java虚拟机学习笔记

慢慢琢磨JVM

http://hi.baidu.com/zoouyong20/blog/item/db26d6127d3d3e15203f2e8e.html

## JVM参数

深入JVM

http://developer.51cto.com/col/1006/

JVM对象生命周期

http://developer.51cto.com/art/201009/227897.htm

详细介绍什么是Java虚拟机(1)

http://developer.51cto.com/art/200701/37403.htm

JVM启动参数大全zz

http://www.blogjava.net/midstr/archive/2008/09/21/230265.html

10 个非常重要的HotSpot JVM 参数

http://www.oschina.net/translate/hotspot-jvm-options-java-examples

java jvm参数 -Xms -Xmx -Xmn -Xss调优总结

http://hi.baidu.com/sdausea/item/f4671be89a7ec40d570f1d2d

Java虚拟机的统计监测工具:jstat

http://www.jvmer.com/java%E8%99%9A%E6%8B%9F%E6%9C%BA%E7%9A%84%E7%BB%9F%E8%AE%A1%E7%9B%91%E6%B5%8B%E5%B7%A5%E5%85%B7jstat/

dump java 虚拟机内存

http://www.jvmer.com/dump-java-%E8%99%9A%E6%8B%9F%E6%9C%BA%E5%86%85%E5%AD%98/

JVM参数收集

http://www.jvmer.com/jvm%E5%8F%82%E6%95%B0%E6%94%B6%E9%9B%86/

JVM -XX: 参数介绍

http://www.jvmer.com/jvm-xx-%E5%8F%82%E6%95%B0%E4%BB%8B%E7%BB%8D/

Java虚拟机的统计监测工具（二）:jstack

<http://www.jvmer.com/java%E8%99%9A%E6%8B%9F%E6%9C%BA%E7%9A%84%E7%BB%9F%E8%AE%A1%E7%9B%91%E6%B5%8B%E5%B7%A5%E5%85%B7%EF%BC%88%E4%BA%8C%EF%BC%89jstack/>

## 编写对GC友好的代码

Baidu