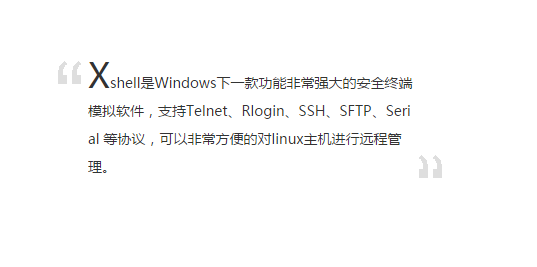
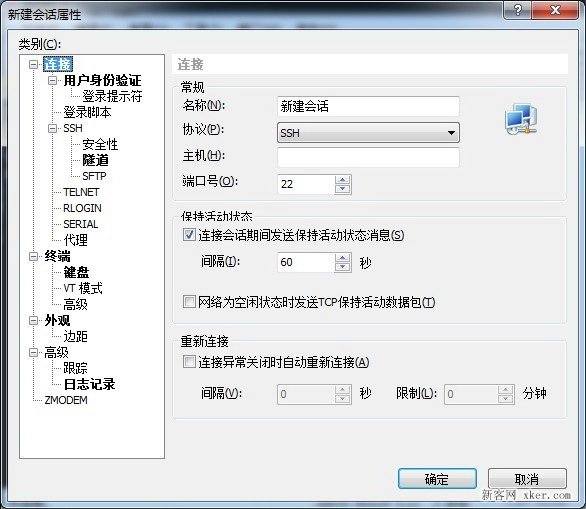
# 打开xshell

1. 



新建一个连接 填写ip和端口 22端口是SSH端口

# Xshell的操作记录

## ps命令 Process Status:

如果直接用ps命令，会显示所有进程的状态，通常结合grep命令查看某进程的状态。

grep （global search regular expression(RE) and print out the line,全面搜索正则表达式并把行打印出来）是一种强大的文本搜索工具，它能使用正则表达式搜索文本，并把匹配的行打印出来。

启动这个进程的用户和它启动的时间（u）

ps aux 列出目前所有在内存中的程序

ps a 流出同一终端下的所有程序,包括其他用户程序

ps u 以用户为主的格式来显示程序状况

px x 显示所有程序,不以终端机来区分

[root@localhost test6]# ps aux

USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND

root 1 0.0 0.0 10368 676 ? Ss Nov02 0:00 init [3]

root 2 0.0 0.0 0 0 ? S< Nov02 0:01 [migration/0]

root 3 0.0 0.0 0 0 ? SN Nov02 0:00 [ksoftirqd/0]

root 4 0.0 0.0 0 0 ? S< Nov02 0:01 [migration/1]

root 5 0.0 0.0 0 0 ? SN Nov02 0:00 [ksoftirqd/1]

root 6 0.0 0.0 0 0 ? S< Nov02 29:57 [events/0]

root 7 0.0 0.0 0 0 ? S< Nov02 0:00 [events/1]

root 8 0.0 0.0 0 0 ? S< Nov02 0:00 [khelper]

root 49 0.0 0.0 0 0 ? S< Nov02 0:00 [kthread]

root 54 0.0 0.0 0 0 ? S< Nov02 0:00 [kblockd/0]

root 55 0.0 0.0 0 0 ? S< Nov02 0:00 [kblockd/1]

root 56 0.0 0.0 0 0 ? S< Nov02 0:00 [kacpid]

……省略部分结果

说明:

USER：该 process 属于那个使用者账号的

PID ：该 process 的号码

%CPU：该 process 使用掉的 CPU 资源百分比

%MEM：该 process 所占用的物理内存百分比

VSZ ：该 process 使用掉的虚拟内存量 (Kbytes)

RSS ：该 process 占用的固定的内存量 (Kbytes)

TTY ：该 process 是在那个终端机上面运作，若与终端机无关，则显示 ?，另外， tty1-tty6 是本机上面的登入者程序，若为 pts/0 等等的，则表示为由网络连接进主机的程序。

STAT：该程序目前的状态，主要的状态有

R ：该程序目前正在运作，或者是可被运作

S ：该程序目前正在睡眠当中 (可说是 idle 状态)，但可被某些讯号 (signal) 唤醒。

T ：该程序目前正在侦测或者是停止了

Z ：该程序应该已经终止，但是其父程序却无法正常的终止他，造成 zombie (疆尸) 程序的状态

START：该 process 被触发启动的时间

TIME ：该 process 实际使用 CPU 运作的时间

COMMAND：该程序的实际指令

## Ctrl +c： 退出所有正在执行的操作

## source

source a.sh

在当前shell内去读取、执行a.sh，而a.sh不需要有"执行权限"

source命令可以简写为"."

. a.sh（注意：中间是有空格的。）

## sh/bash

sh a.sh

bash a.sh

都是打开一个subshell去读取、执行a.sh，而a.sh不需要有"执行权限"

通常在subshell里运行的脚本里设置变量，不会影响到父shell的。

## ./

./a.sh

#bash: ./a.sh: 权限不够

chmod +x a.sh

./a.sh

打开一个subshell去读取、执行a.sh，但a.sh需要有"执行权限"

可以用chmod +x添加执行权限

## fork、source、exec

使用fork方式运行script时， 就是让shell(parent process)产生一个child process去执行该script，当child process结束后，会返回parent process，但parent process的环境是不会因child process的改变而改变的。

使用source方式运行script时， 就是让script在当前process内执行， 而不是产生一个child process来执行。由于所有执行结果均于当前process内完成，若script的环境有所改变， 当然也会改变当前process环境了。

使用exec方式运行script时， 它和source一样，也是让script在当前process内执行，但是process内的原代码剩下部分将被终止。同样，process内的环境随script改变而改变。

通常如果我们执行时，都是默认为fork的。

## vim编辑文件

vi模式下编辑文件：

vi /etc/saikik表示显示/etc/saikik文件的内容。-- 此时不能编辑

### 翻页：

Page Up和Page Down键可以上下翻页;

### 编辑：

按下Insert键，可以见到窗口左下角有“Insert”字样，表示当前为插入编辑状态，这时从键盘输入的内容将插入到光标位置;

### 替换：

再按下Insert键，左下角将有“Replace”字样，表示当前为替换编辑状态，这时从键盘输入的内容将替换光标位置的内容。

### 保存

Esc键+:wq :编辑完内容后，按下Esc键，并输入“:wq”，然后回车就可以保存退出。

### 不不保存退出

如果不想保存而直接退出，则按下Esc键后，输入“:q!”，然后回车即可。“wq”表示Write和Quit，即保存退出;“q!”表示忽略修改强行退出。

### vim文件跳转到顶部：

:1

### vim文件跳转到底部：

:$

## 5. kill -n

-n 指的是 信号编号

## 6.$ su -c "kill -15 4982"

Password:

若是没有权限，会报错：Operation not permitted

## 7.chmod -R 777 \* ：

表示对当前目录及目录下所有的文件赋予可读可写可执行权限。

在linux终端先输入ll,可以看到如:

-rwx-r--r-- (一共10个参数) 表示文件所属组和用户的对应权限。

第一个跟参数属于管理员，跟chmod无关,先不管.

2-4参数:属于user

5-7参数:属于group

8-10参数:属于others

接下来就简单了:r==>可读 w==>可写 x==>可执行

r=4 w=2 x=1

所以755代表 rwxr-xr-x

777 代表 rwxr-rwx-rwx 所有用户都可读可写可执行。

## 8. sudo chmod 777 /ser/data/me.txt --- 用 sudo语句修改文件的访问权限

## 11. 查看服务的pid

sh taiyi\_game status

## 12.查看内存：df -h

df：列出文件系统的整体磁盘使用量；

du：评估文件系统的磁盘使用量（常用于评估目录所占容量）

//查看当前目录下各个文件及目录占用空间大小

du -sh \*

df参数：

-a：列出所有的文件系统，包括系统特有的/proc等文件系统

-k：以KB的容量显示各文件系统

-m：以MB的容量显示各文件系统

-h：以人们较易阅读的GB,MB,KB等格式自行显示

-H：以M=1000K替代M=1024K的进位方式

-T：连同该分区的文件系统名称（例如ext3）也列出

-i：不用硬盘容量，而以inode的数量来显示

## 13.grep 的使用

grep 同时满足多个关键字和满足任意关键字

① grep -E "word1|word2|word3"   file.txt

   满足任意条件（word1、word2和word3之一）将匹配。

② grep word1 file.txt | grep word2 |grep word3

   必须同时满足三个条件（word1、word2和word3）才匹配。

# 14. 查看CPU

## 1.查看每个CPU核的信息

cat /proc/cpuinfo

cat /proc/cpuinfo | grep "physical id" | uniq | wc -l

grep -c 'processor' /proc/cpuinfo

总逻辑CPU数 = 物理CPU个数 \* 每颗物理CPU的核数 \* 超线程数

查询CPU个数：

cat /proc/cpuinfo| grep "physical id"| sort| uniq| wc -l

查询核数：

cat /proc/cpuinfo| grep "cpu cores"| uniq

查询逻辑CPU总数：

cat /proc/cpuinfo| grep "processor"| wc -l

## 15. \*\*uniq命令：删除重复行;wc –l命令：统计行数\*\*

1.2 查看CPU核数

# cat /proc/cpuinfo | grep "cpu cores" | uniq

cpu cores : 4

1.3 查看CPU型号

# cat /proc/cpuinfo | grep 'model name' |uniq

model name : Intel(R) Xeon(R) CPU E5630 @ 2.53GHz

总结：该服务器有2个4核CPU，型号Intel(R) Xeon(R) CPU E5630 @ 2.53GHz

## 16. 查看总内存大小

#cat /proc/meminfo | grep MemTotal

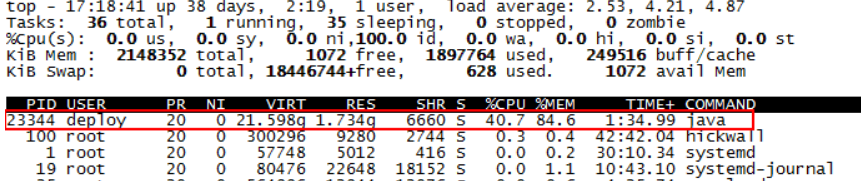
MemTotal: 32941268 kB //内存32G

# dmidecode |grep -A16 "Memory Device$"

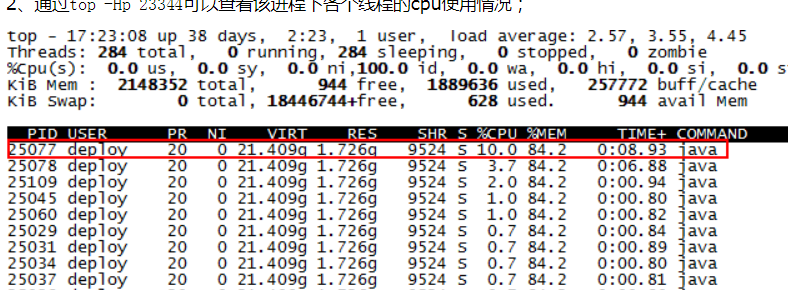
## /proc/meminfo

# Linux 中 问题的排查

## 1通过top命令查看各个进程的cpu使用情况，默认按cpu使用率排序

、

## 2、通过top -Hp 23344可以查看该进程下各个线程的cpu使用情况；



上图中可以看出pid为25077的线程占了较多的cpu资源，利用jstack命令可以继续查看该线程当前的堆栈状态。

## 3继续使用jstack pid命令查看当前java进程的堆栈状态

### 转换为16进制格式：printf "%x\n" tid

### 查询指定java进程的所有对象使用内存情况。

jmap -histo  pid

### ps -mp pid -o THREAD,tid,time  查询所有线程列表

printf "%x\n" 21742

得到21742的十六进制值为54ee

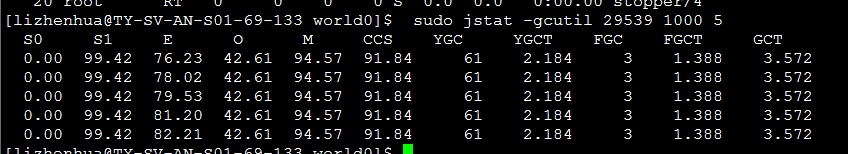
root@ubuntu:/# jstack 21711 | grep 54ee

"PollIntervalRetrySchedulerThread" prio=10 tid=0x00007f950043e000 nid=0x54ee in Object.wait()

# 查看linux gc 使用情况

没有权限的话，需要添加 速sudo命令

## 总垃圾回收统计 sudo jstat -gcutil 31939 1000 5



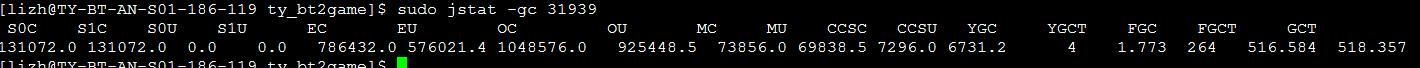
* **S0：**幸存1区当前使用比例
* **S1：**幸存2区当前使用比例
* **E：**伊甸园区使用比例
* **O：**老年代使用比例
* **M：**元数据区使用比例
* **CCS：**压缩使用比例
* **YGC：**年轻代垃圾回收次数
* **FGC：**老年代垃圾回收次数
* **FGCT：**老年代垃圾回收消耗时间
* **GCT：**垃圾回收消耗总时间

## 堆内存统计sudo jstat -gccapacity 31939



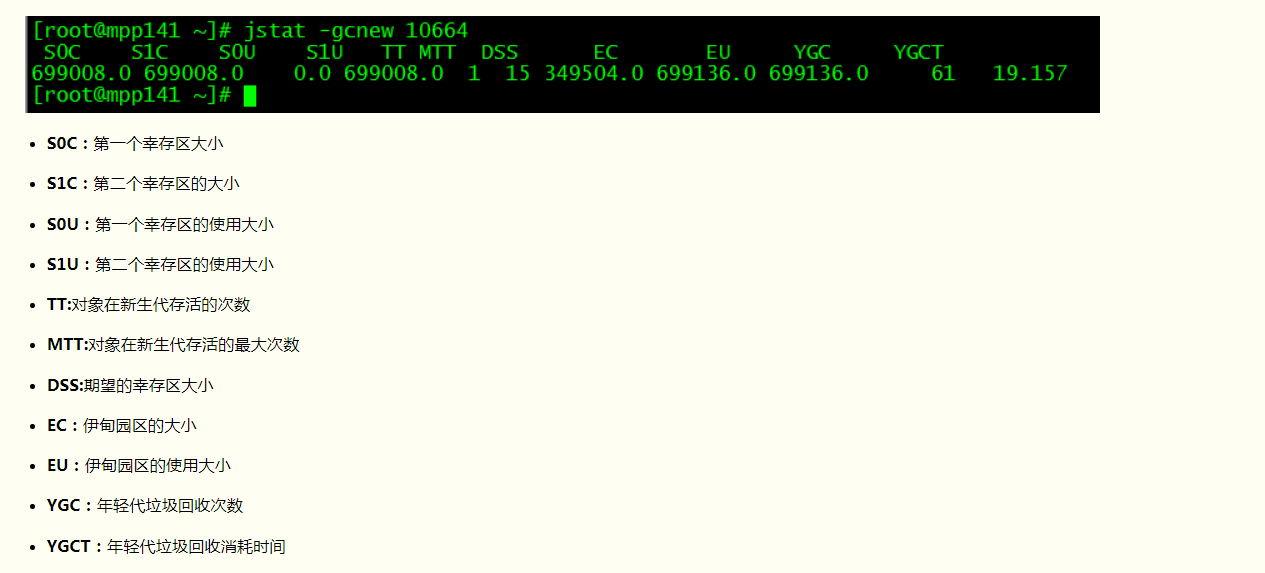
* **NGCMN：**新生代最小容量
* **NGCMX：**新生代最大容量
* **NGC：**当前新生代容量
* **S0C：**第一个幸存区大小
* **S1C：**第二个幸存区的大小
* **EC：**伊甸园区的大小
* **OGCMN：**老年代最小容量
* **OGCMX：**老年代最大容量
* **OGC：**当前老年代大小
* **OC:**当前老年代大小
* **MCMN:**最小元数据容量
* **MCMX：**最大元数据容量
* **MC：**当前元数据空间大小
* **CCSMN：**最小压缩类空间大小
* **CCSMX：**最大压缩类空间大小
* **CCSC：**当前压缩类空间大小
* **YGC：**年轻代gc次数
* **FGC：**老年代GC次数

## 垃圾回收统计 sudo jstat –gc 31939

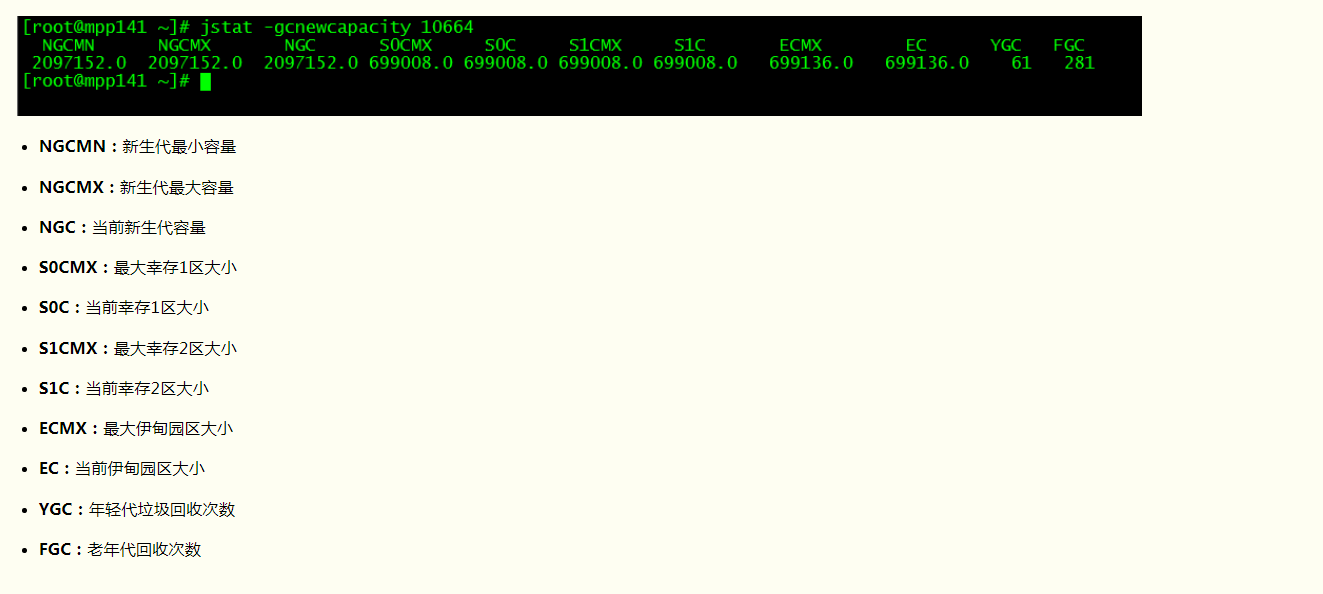


* **S0C：**第一个幸存区的大小
* **S1C：**第二个幸存区的大小
* **S0U：**第一个幸存区的使用大小
* **S1U：**第二个幸存区的使用大小
* **EC：**伊甸园区的大小
* **EU：**伊甸园区的使用大小
* **OC：**老年代大小
* **OU：**老年代使用大小
* **MC：**方法区大小
* **MU：**方法区使用大小
* **CCSC:**压缩类空间大小
* **CCSU:**压缩类空间使用大小
* **YGC：**年轻代垃圾回收次数
* **YGCT：**年轻代垃圾回收消耗时间
* **FGC：**老年代垃圾回收次数
* **FGCT：**老年代垃圾回收消耗时间
* **GCT：**垃圾回收消耗总时间

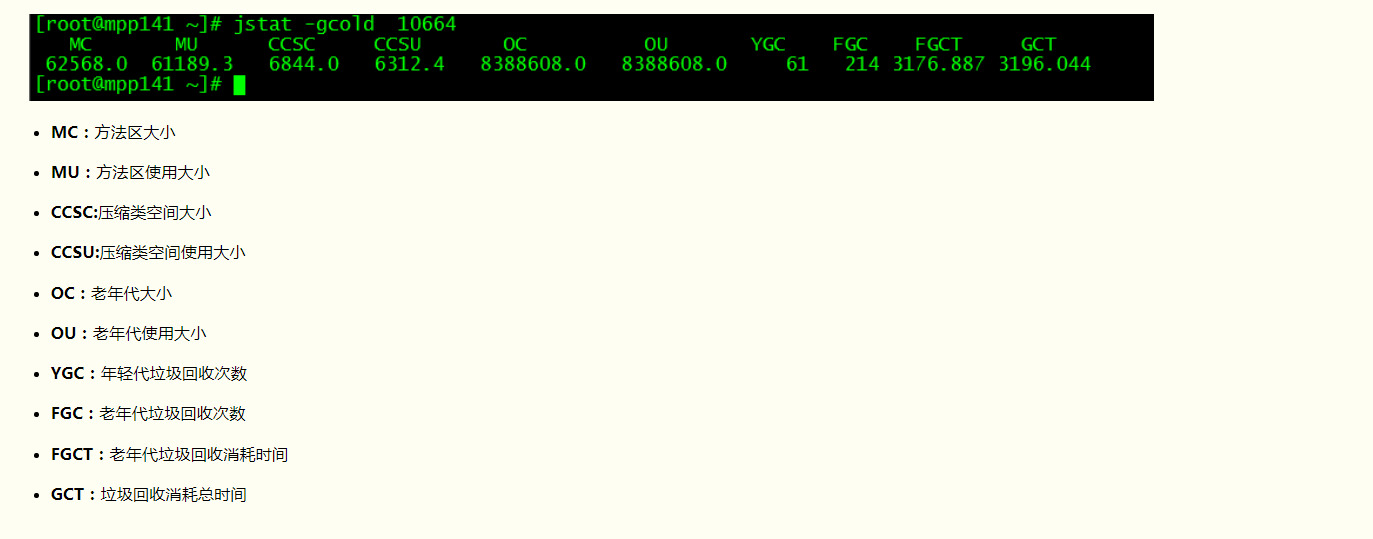
## 新生代垃圾回收统计 sudo jstat -gcnew 25998



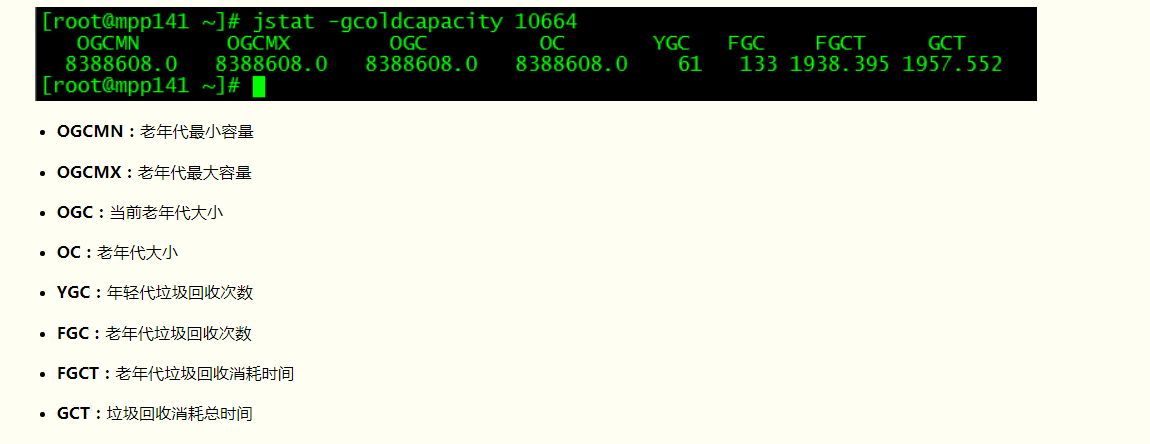
## 新生代内存统计sudo jstat -gcnewcapacity 25998



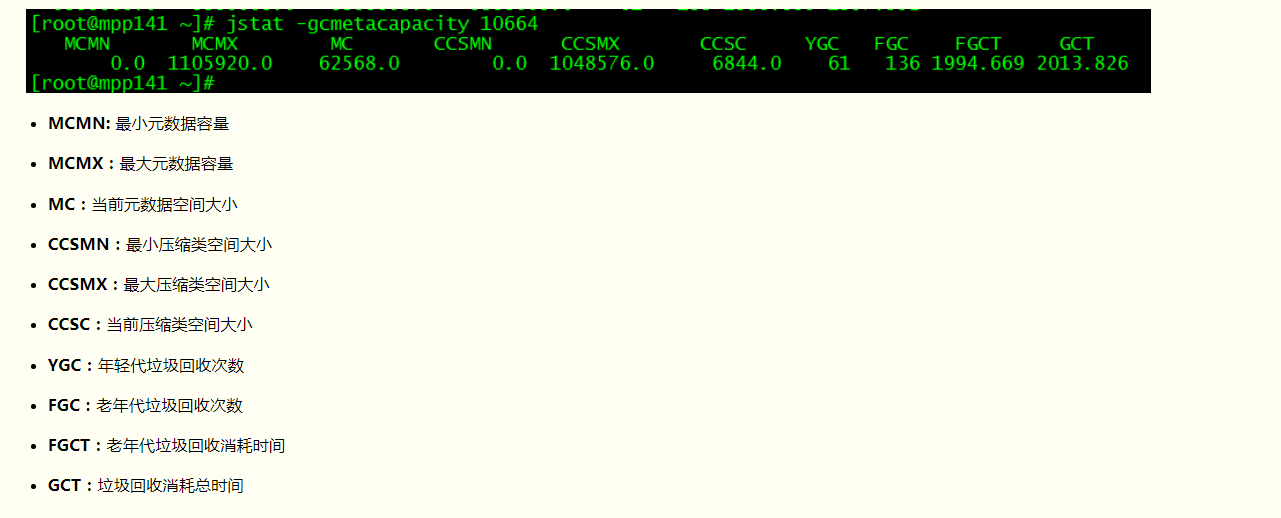
## 老年代垃圾回收统计 sudo jstat -gcold 25998



## 老年代内存统计



## 元数据空间统计



## JVM编译方法统计



1.兑换码获得的元宝，有记录，后台C:\Users\Administrator\AppData\Roaming\Tencent\Users\329731049\TIM\WinTemp\RichOle\E1@5AX`GZQPY7$2XMU280)Q.png  
2.充值赠送的元宝，

普通充值，第n次：元宝值+额外赠送元宝 

普通充值，首充：元宝值+首充赠送元宝

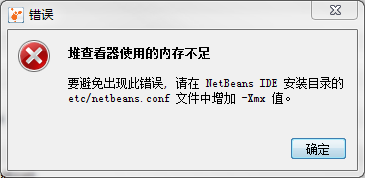
月卡：元宝值 

Ios月卡：元宝值+ 

# 内存对象日志：

## jmap -histo 31939 >jsmap\_histo\_31939.txt

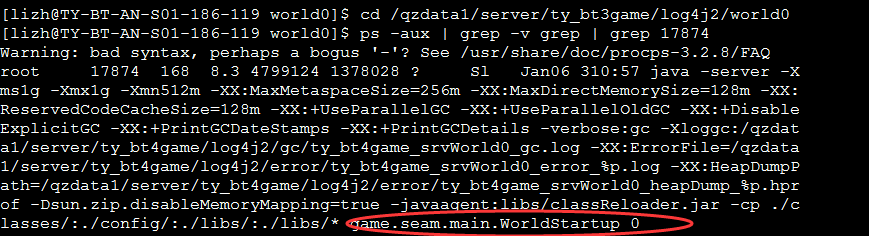
把生成的对象的日志放到指定的位置



# 通过 pid 查找进程的位置 等信息

## 1.top 命令获取 pid 信息

## 2. # ps -aux |grep -v grep|grep 28990



## 3. 得知PID之后可以直接进入/proc/28990

通过上面的信息我们也可以找出这是一个java进程，在/opt/sonatype/nexus/bin/nexus目录下，当然可以有别的更简单直接的方法.

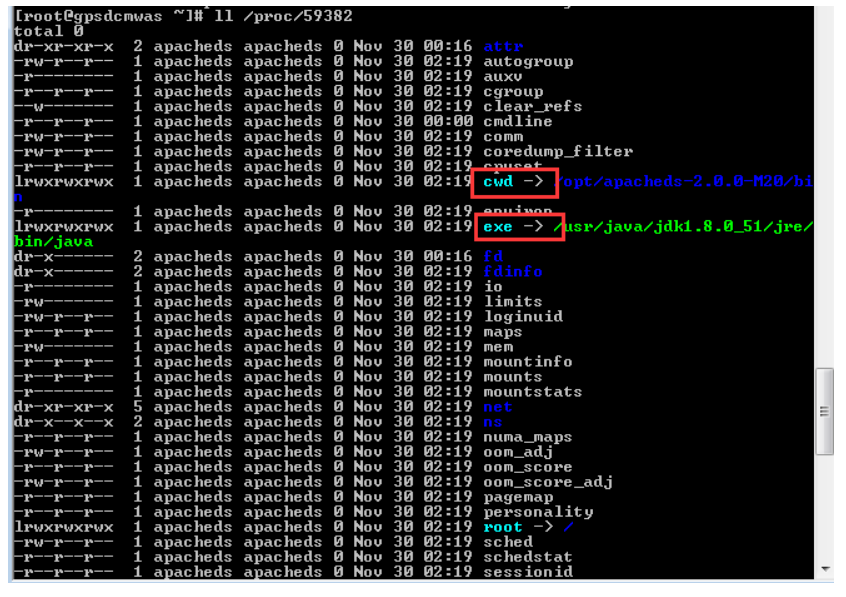
# 通过pid查找进程的绝对位置

通过ps及top命令查看进程信息时，只能查到相对路径，查不到的进程的详细信息，如绝对路径等

[**Linux**](http://lib.csdn.net/base/linux)在启动一个进程时，系统会在/proc下创建一个以PID命名的文件夹，在该文件夹下会有我们的进程的信息，其中包括一个名为exe的文件即记录了绝对路径，通过ll或ls –l命令即可查看。

## 1先通过top查看进程PID

## 2 ll /proc/PID



cwd符号链接的是进程运行目录；

exe符号连接就是执行程序的绝对路径；

cmdline就是程序运行时输入的命令行命令；

environ记录了进程运行时的环境变量；

fd目录下是进程打开或使用的文件的符号连接

# 对端口占用 的查询：

## 1、netstat -anp|grep 8282

查看此端口的占用情况；16923为pid

## 2、ps -ef|grep 16923

查看该pid的进程的详细信息；

# 查看所有的进程,端口和PID

sudo netstat -autup

# 根据端口查PID

#方法一

sudo netstat -nlp|grep :80

#方法二

sudo lsof -i :80

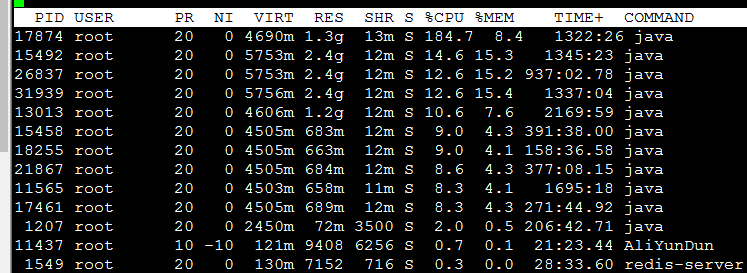
# 根据PID查端口

sudo netstat -antup|grep 2226

# 根据进程名来查PID

sudo ps -ef|grep nginx

# top 命令各个参数详解：



PID：进程的ID  
　　*USER：进程*所有者  
　　PR：进程的优先级别，越小越优先被执行  
　　NInice：进程Nice值，代表这个进程的优先值，进程可被执行的优先级的修正数值 取值范围[-20,19] 正值表示低优先级，负值表示高优先级

对nice值一个形象比喻，假设在一个CPU轮转中，有2个runnable的进程A和B，如果他们的nice值都为0，假设内核会给他们每人分配1k个cpu时间片。但是假设进程A的为0，但是B的值为-10，那么此时CPU可能分别给A和B分配1k和1.5k的时间片。故可以形象的理解为，nice的值影响了内核分配给进程的cpu时间片的多少，时间片越多的进程，其优先级越高，其优先级值（PRI）越低。%nice，就是改变过优先级的进程的占用CPU的百分比，如上例中就是0.5k/2.5k=1/5=20%。

VIRT：进程占用的虚拟内存  
　　RES：进程占用的物理内存  
　　SHR：进程使用的共享内存  
　　S：进程的状态。S表示休眠，R表示正在运行，Z表示僵死状态，N表示该进程优先值为负数  
　　%CPU：进程占用CPU的使用率  
　　%MEM：进程使用的物理内存和总内存的百分比  
　　TIME+：该进程启动后占用的总的CPU时间，即占用CPU使用时间的累加值。  
　　COMMAND：进程启动命令名称

# top 查看进程 ：

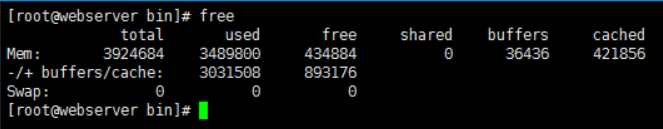
通过top命令查看各个进程的cpu使用情况，默认按cpu使用率排序

top 查看线程：通过top -Hp 23344可以查看该进程下各个线程的cpu使用情况；

退出：键入q即可或按ctrl c

查看线程代码： jstack 进程pid | grep 线程pid16进制 例如： jstack 46924 | grep e58f

# Free 命令各个参数详解：

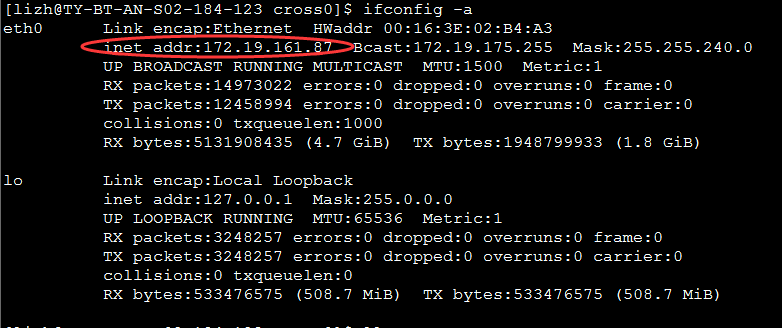


total:总计物理内存的大小。  
used:已使用多大。  
free:可用有多少。  
Shared:多个进程共享的内存总额。  
Buffers/cached:磁盘缓存的大小。  
第三行(-/+ buffers/cached):  
used:已使用多大。  
free:可用有多少。  
第四行就不多解释了。  
区别：第二行(mem)的used/free与第三行(-/+ buffers/cache) used/free的区别。 这两个的区别在于使用的角度来看，第一行是从OS的角度来看，因为对于OS，buffers/cached 都是属于被使用，所以他的可用内存是434884KB,已用内存是3489800KB,  
第三行所指的是从应用程序角度来看，对于应用程序来说，buffers/cached 是等于可用的，因为buffer/cached是为了提高文件读取的性能，当应用程序需在用到内存的时候，buffer/cached会很快地被回收。  
所以从应用程序的角度来说，可用内存=系统free memory+buffers+cached。

 如上例：  
893176=434884+36436+421856

# 查看内网ip:

ifconfig -a

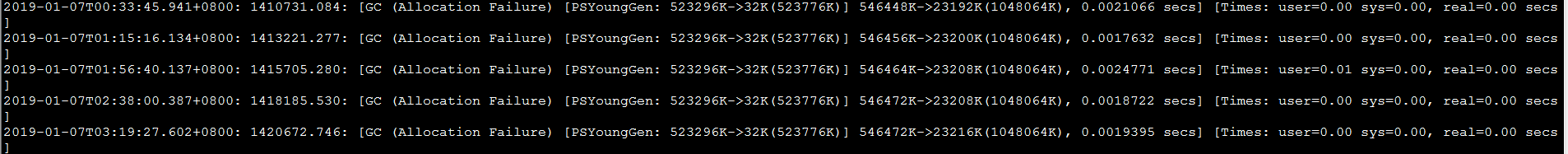


# 查看外网ip:

curl ifconfig.me



# GC日志分析1：



## **GC：**

表明进行了一次垃圾回收，前面没有Full修饰，表明这是一次Minor GC ,注意它不表示只GC新生代，并且现有的不管是新生代还是老年代都会STW。

## **Allocation Failure：**

表明本次引起GC的原因是因为在年轻代中没有足够的空间能够存储新的数据了。

## **P**SYoungGen：

GC日志中的PSYoungGen（PS是指Parallel Scavenge）为Eden+FromSpace，而整个YoungGeneration为Eden+FromSpace+ToSpace

例如：

新生代大小：10240k = PSYoungGen+ToSpace =9216K+1024K

=Eden + FromSpace + ToSpace = 8192K+1024K+1024K

## PSYoungGen: 523296K->32K(523776K)

回收前->回收后（总大小）

## 整个堆区的回收

**546480K->23216K(1048064K), 0.0015430 secs**

四个参数分别为：堆区垃圾回收前的大小，堆区垃圾回收后的大小，堆区总大小。该内存区域GC耗时，单位是秒

**[Times: user=0.04 sys=0.00, real=0.01 secs]：**

    分别表示用户态耗时，内核态耗时和总耗时

## 进入老年区的大小：

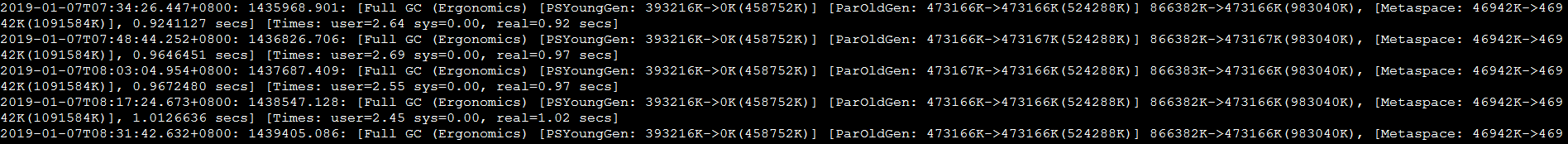
**523296K（**P**SYoungGen回收前大小） – 32k（**P**SYoungGen回收后大小）= 523264k**

**546480K（回收前总堆大小） -23216K （回收后总堆大小）= 523264k**

**523264k - 523264k = 0;**

**说明，没有对象回收进入老年区**

# GC日志分析2：



**[PSYoungGen: 393216K->0K(458752K)] [ParOldGen: 473166K->473166K(524288K)] 866382K->473166K(983040K), [Metaspace: 46942K->46942K(1091584K)], 1.0957651 secs] [Times: user=2.48 sys=0.00, real=1.10 secs]**

## PSYoungGen 回收：

**393216-0K = 393216K**

## ParOldGen 回收：

**473166K - 473166K =0k**

## 总堆内存 回收：

**866382K - 473166K = 393216**

## Metaspace 回收:0k

**即：老年区根本没有回收内存，新生区内存全部回收了！！, Metaspace 分配的空间超出需要，可以减小**