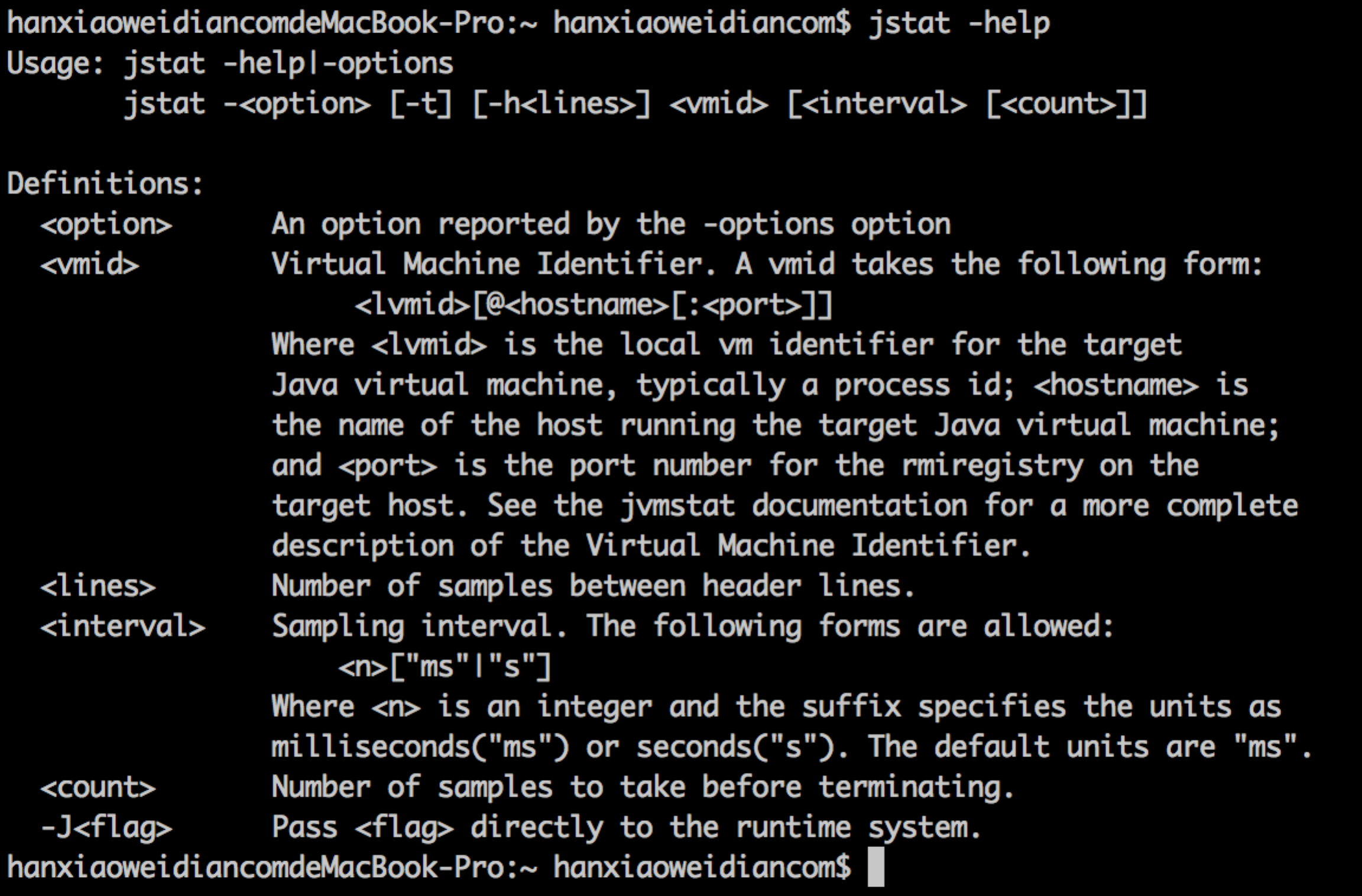
一、Jstat——虚拟机统计信息监视工具

1. 介绍：jstat是用于 监控虚拟机各种运行状态信息的命令行工具。它可以显示本地或者远程虚拟机进程中的类装载、内存、垃圾收集、JIT编译等运行数据。
2. 使用说明：

(1)输入命令行：jstat －help，结果如下图，内容为jstat的用法，包括语法和参数：



(2)输入jstat -options，结果如下图，内容为jstat的options：



其中，各个options对应的参数意义如下表所示：

表 jstat的options涵义

|  |  |
| --- | --- |
| class | 用于查看类加载情况的统计 |
| compiler | 用于查看hotspot中即时编译器编译情况的统计 |
| gc | 用于查看jvm中堆的垃圾回收情况的统计 |
| gccapacity | 用于查看新生代、老生代以及持久代的存储容量情况 |
| gccause | 用于查看垃圾回收的统计情况（这个和gcutil选项一样），如果发生垃圾回收，它还会显示最后一次及当前正在发生垃圾回收的原因 |
| gcnew | 用于查看新生代垃圾回收情况 |
| gcnewcapacity | 用于查看新生代的存储容量情况 |
| gcold | 用于查看老生代以及持久代垃圾回收情况 |
| gcoldcapacity | 用于查看老生代的存储容量情况 |
| gcpermcapacity | 用于查看持久代的存储容量情况 |
| gcutil | 用于查看新生代、老生代及持久代垃圾回收情况 |
| printcompilation | Hotspot编译方法的统计 |

3、使用实例

如前面所述，jstat语法格式为：

jstat -<option> [-t] [-h<lines>] <vmid> [<interval> [<count>]]

（1）输入命令：$ jstat -gc -t -h 2 592 2 4

其中：

-gc 用于查看jvm中堆的垃圾回收情况的统计

－t 用于在输出内容的第一列显示时间戳，这个时间戳代表的时JVM开始启动到现在的时间

－h 2 用于指定每隔2就输出列头，如果不指定，默认是只在第一行出现列头。

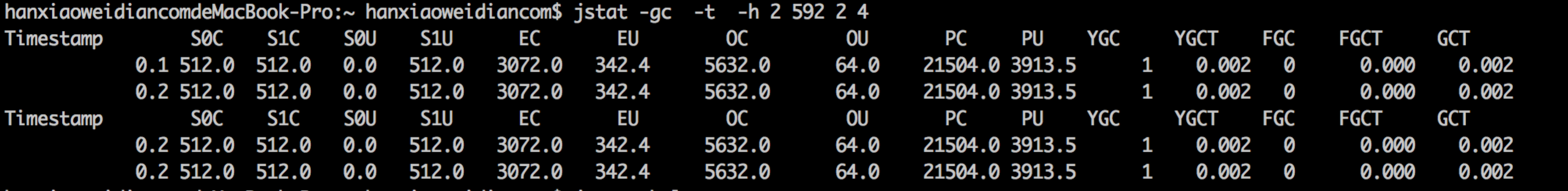
Vimd 用于指定虚拟机进程（此处指定的是594），需要特别说明的是：如果是本地虚拟机进程，VMID与LVMID是一致的，如果是远程虚拟机进程，那么vmid的格式应当是：

［protocol;］[//]lvmid[@hostname[:port]/servername]

interval 指定查询间隔，单位为毫秒（此处指定2ms查询一次）

Count 指定查询次数（此处指定查询4次），若省略interval和count参数，则默认只查询一次。

综上，该条指令表示：每2ms查询一次进程592的垃圾回收情况，一共查询4次，输出内容的第一列显示时间戳，每隔2就输出列头，查询结果如下图：



查询结果表明：

进程592的S0C 新生代中Survivor space中S0当前容量的大小为512KB

S1C 新生代中Survivor space中S1当前容量的大小为512KB

S0U 新生代中Survivor space中S0容量使用的大小为0KB

S1U新生代中Survivor space中S1容量使用的大小为512KB

EC 新生代重Eden space当前容量的大小为3072 KB

EU 新生代重Eden space当前容量的大小为342.4 KB

OC Old space当前容量的大小为5632 KB

OU Old space使用容量的大小为64 KB

PC Permanent space当前容量的大小为21504（B

PU Permanent space使用容量的大小为3913.5 KB

YGC 从应用程序启动到采样时发生 Young GC 的次数为1

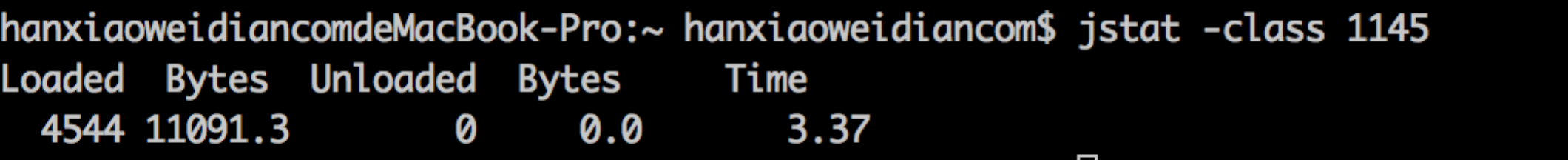
YGCT 从应用程序启动到采样时 Young GC 所用的时间为0.002秒

FGC 从应用程序启动到采样时发生 Full GC 的次数为0

FGCT 从应用程序启动到采样时 Full GC 所用的时间为0秒

GCTT从应用程序启动到采样时用于垃圾回收的总时间，它的值等于YGC+FGC为0.002秒。

（2）输入命令：$ jstat –class 1145,结果如下：



查询结果表明：

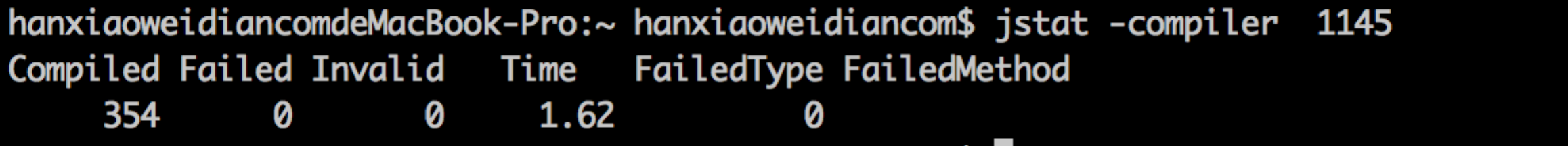
进程1145的Loaded 加载了的类的数量为4544

Bytes 加载了的类的大小为11091.3Kb

Unloaded 卸载了的类的数量为0

Time 花在类的加载及卸载的时间为3.37s

（3）输入命令:$ jstat –compilerc1145，结果如下：



查询结果表明：

进程1145的 Compiled 编译任务执行的次数为354

Failed 编译任务执行失败的次数为0

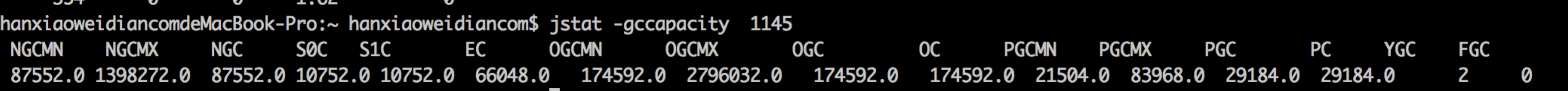
Invalid 编译任务非法执行的次数为0

Time 执行编译花费的时间为1.62s

FailedType 最后一次编译失败的编译类型为0

FailedMethod 最后一次编译失败的类名及方法名

（4）输入命令：$ stat –gccapacity，结果如下：



查询结果表明：

进程1145的 NGCMN新生代的最小容量大小为87552（KB）

NGCMX新生代的最大容量大小1398272（KB）

NGC当前新生代的容量大小87552（KB）

S0C当前新生代中survivor space 0的容量大小10752（KB）

S1C当前新生代中survivor space 1的容量大小10752（KB）

EC Eden space当前容量的大小66048（KB）

OGCMN老生代的最小容量大小174592（KB）

OGCMX老生代的最大容量大小2796032（KB）

OGC当前老生代的容量大小174592（KB）

OC当前老生代的空间容量大小174592（KB）

PGCMN持久代的最小容量大小21504（KB）

PGCMX持久代的最大容量大小83968（KB）

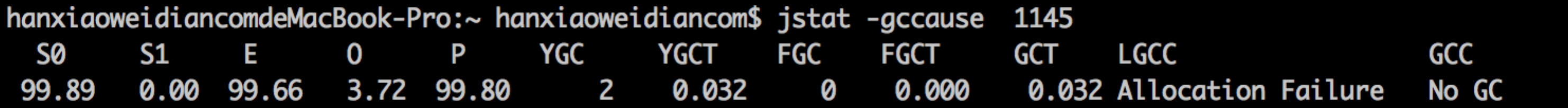
PGC当前持久代的容量大小29184（KB）

PC当前持久代的空间容量大小29184（KB）

YGC 从应用程序启动到采样时发生 Young GC 的次数为2

FGC 从应用程序启动到采样时发生 Full GC 的次数为0

（5）输入命令：$jstat –gccause 1145,结果如下：



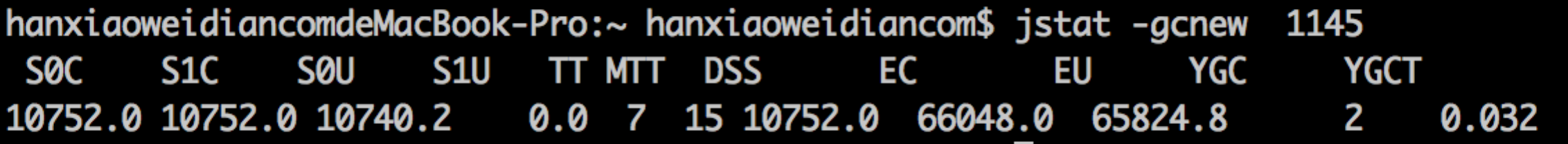
查询结果表明：

进程1145的gccause选项的前几列内容和－gcutil一样，此处不再赘述，仅解释后两个：

LGCC 最后一次垃圾收集的原因为： allocation failure

Gcc 当前垃圾回收的原因为： NO GC

(6)输入命令：$ jstat –gcnew 1145,结果如下：



查询结果各列意义为：

S0C当前新生代中survivor space 0的容量大小（KB）

S1C 当前新生代中survivor space 1的容量大小（KB）

S0U S0已经使用的大小（KB）

S1U S1已经使用的大小（KB）

TT (Tenuring threshold ) 这个阈值用于表示某个/些在执行批定次数youngGC后还活着的对象，即使此时新生的的Survior没有满，也同样被认为是长周期对象，将会被移到老生代中。

MTT (Maximum tenuring threshold)，用于表示TT的最大值

DSS (Desired survivor size) (KB), 参考网址

http://blog.csdn.net/yangjun2/article/details/6542357

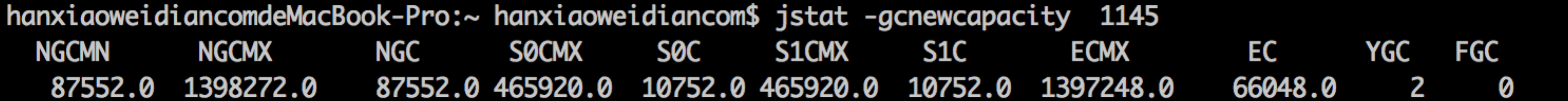
EC Eden space当前容量的大小（KB）

EU Eden space已经使用的大小（KB）

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(秒)

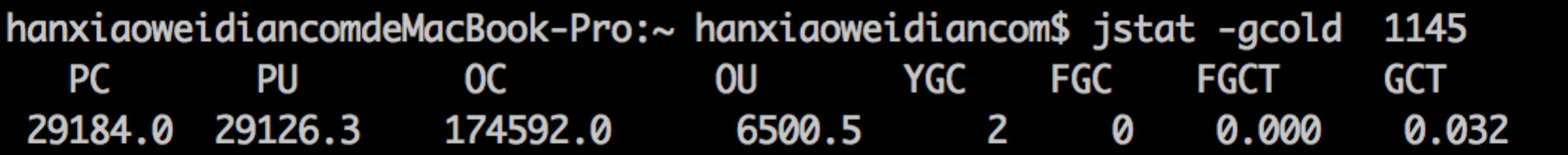
（7）输入命令：$ jstat －gcnewcapacity 1145,结果如下：



查询结果各列意义为：

|  |  |
| --- | --- |
| **列名** | **说明** |
| NGCMN | 新生代的最小容量大小（KB） |
| NGCMX | 新生代的最大容量大小（KB） |
| NGC | 当前新生代的容量大小（KB） |
| S0CMX | 新生代中SO的最大容量大小（KB） |
| S0C | 当前新生代中SO的容量大小（KB） |
| S1CMX | 新生代中S1的最大容量大小（KB） |
| S1C | 当前新生代中S1的容量大小（KB） |
| ECMX | 新生代中Eden的最大容量大小（KB） |
| EC | 当前新生代中Eden的容量大小（KB） |
| YGC | 从应用程序启动到采样时发生 Young GC 的次数 |
| FGC | 从应用程序启动到采样时发生 Full GC 的次数 |

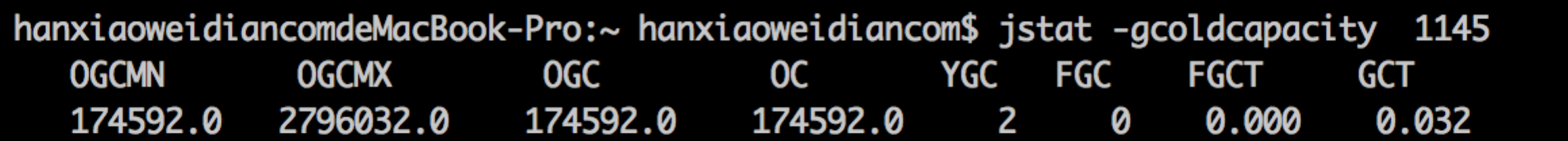
（8）输入命令：$jstat –gcold 1145,结果如下：



查询结果各列意义为：

|  |  |
| --- | --- |
| PC | 当前持久代容量的大小（KB） |
| PU | 持久代使用容量的大小（KB） |
| OC | 当前老年代容量的大小（KB） |
| OU | 老年代使用容量的大小（KB） |
| YGC | 从应用程序启动到采样时发生 Young GC 的次数 |
| 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)，它的值等于YGC+FGC |

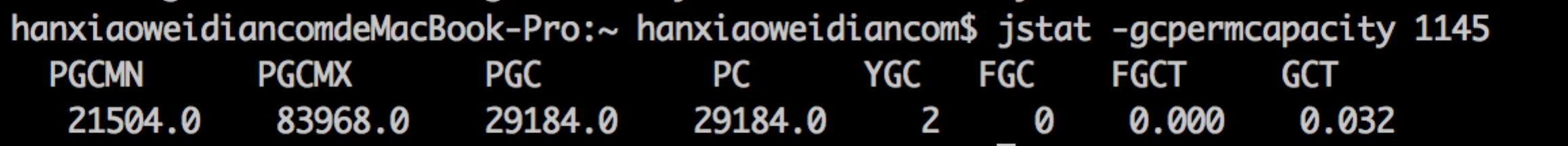
（9）输入命令：$ jstat –gcoldcapacity 1145



查询结果各列意义如下：

|  |  |
| --- | --- |
| OGCMN | 老生代的最小容量大小（KB） |
| OGCMX | 老生代的最大容量大小（KB） |
| OGC | 当前老生代的容量大小（KB） |
| OC | 当前新生代的空间容量大小（KB） |
| YGC | 从应用程序启动到采样时发生 Young GC 的次数 |
| 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)，它的值等于YGC+FGC |

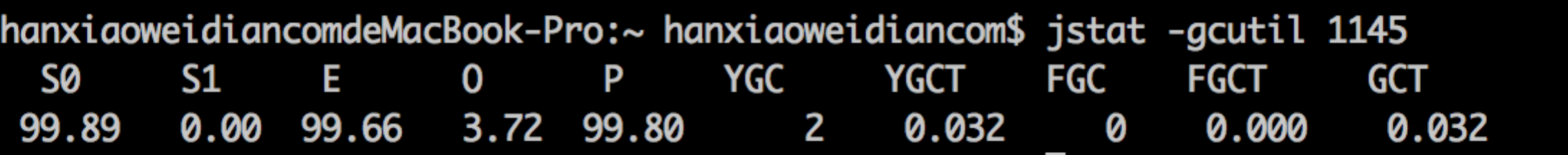
（10）输入命令：$ jstat －gcpermcapacity 1145,结果如下：



查询结果各列意义如下：

|  |  |
| --- | --- |
| PGCMN | 持久代的最小容量大小（KB） |
| PGCMX | 持久代的最大容量大小（KB） |
| PGC | 当前持久代的容量大小（KB） |
| PC | 当前持久代的空间容量大小（KB） |
| YGC | 从应用程序启动到采样时发生 Young GC 的次数 |
| FGC | 从应用程序启动到采样时 Full GC 所用的时间(单位秒) |
| 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)，它的值等于YGC+FGC |

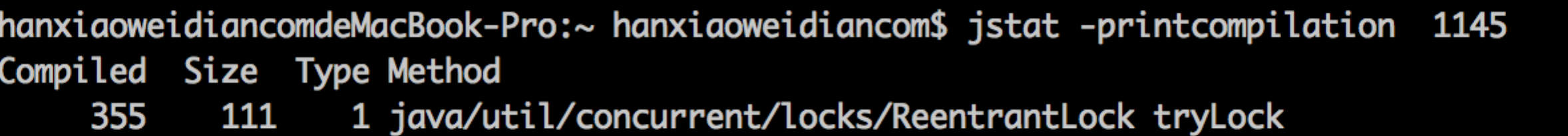
（11）输入命令：$ jstat –gcutil 1145 ,结果如下：



查询结果各列意义如下：

|  |  |
| --- | --- |
| 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) |

（13）输入命令：$ jstat –printcompilation 1145,结果如下：



|  |  |
| --- | --- |
| **列名** | **说明** |
| Compiled | 编译任务执行的次数 |
| Size | 方法的字节码所占的字节数 |
| Type | 编译类型 |
| Method | 指定确定被编译方法的类名及方法名，方法名是被指定的类中的方法，这两个字段的格式是由HotSpot中的“-**XX:+PrintComplation**”选项确定的。 |

（14）查看远程服务器上的GC情况。

这个需要先在远程服务器上面开启jstatd服务，可以选看这里如何开启jstatd服务，

[http://blog.csdn.net/fenglibing/article/details/17323515](http://blog.csdn.net/fenglibing/article/details/17323515" \t "_blank)

  下面是一个执行示例：

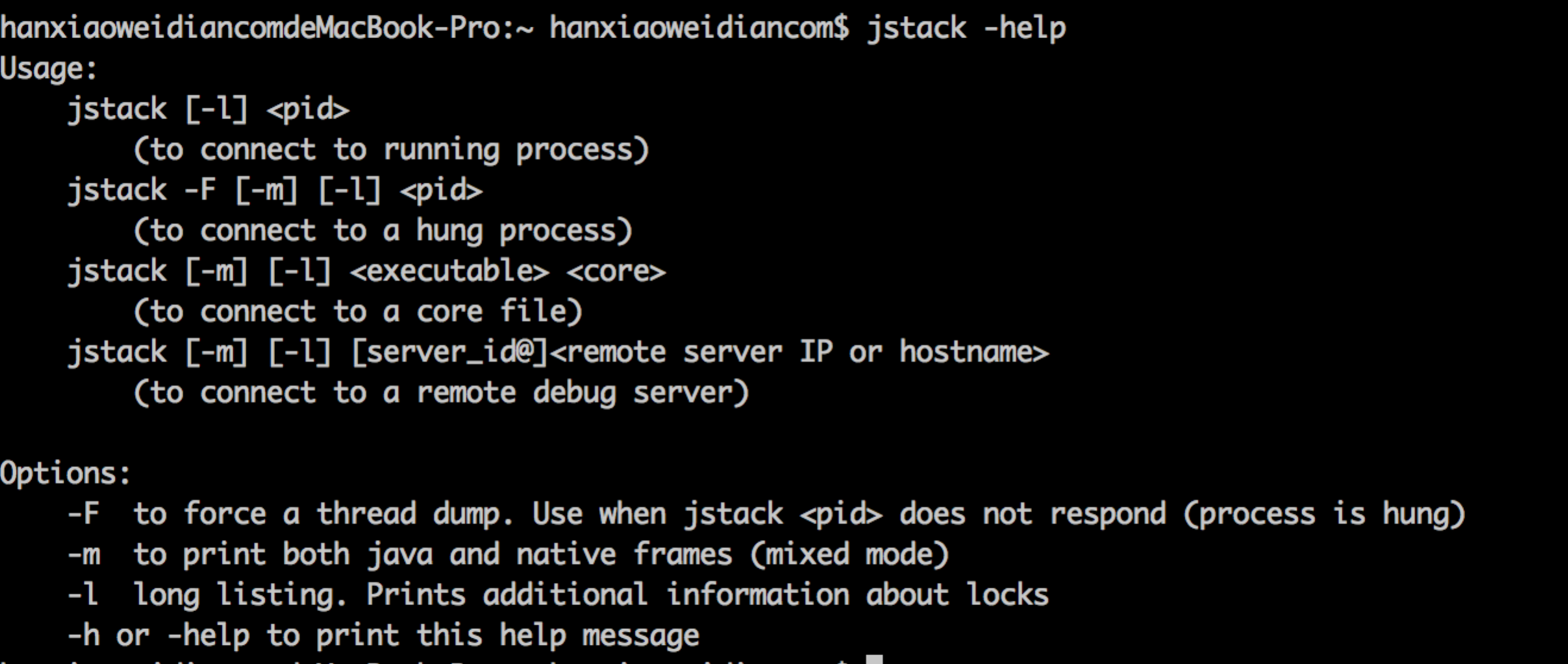
$ jstat -gcutil 18272@the\_ip:2021/jstatdName 1000

  S0     S1     E      O      P     YGC     YGCT    FGC    FGCT     GCT  
   0.00   0.00   8.00   0.00  24.36      0    0.000     0    0.000     0.000  
  0.00   0.00   8.00   0.00  24.36      0    0.000     0    0.000    0.000  
  0.00   0.00   8.00   0.00  24.36      0    0.000     0    0.000    0.000  
  0.00   0.00   8.00   0.00  24.36      0    0.000     0    0.000     0.000  
  0.00   0.00   8.00   0.00  24.36      0    0.000     0    0.000    0.000  
  0.00   0.00   8.00   0.00  24.36      0    0.000     0    0.000     0.000

其他示例以此类推，就不一一例举。

二 、Jstack——java堆栈跟踪工具

1. 介绍：jstack命令用于生成虚拟机当前时刻的线程快照（一般称为threaddump或者javacore文件）。线程快照就是当前虚拟机内每一条线程正在执行的方法堆栈集合，生成线程快照的主要目的是定位线程出现长时间停顿的原因，如线程间死锁、死循环、请求外部资源导致的长时间等待等都是导致线程长时间停顿的常见原因。线程出现停顿的时候通过jstack来查看各个线程的调用堆栈，就可以知道没有响应的线程到底在后台做什么事情或者等待什么资源。
2. 使用说明及参数说明：



如上所示，jstack命令格式为：

jstack ［option］<pid>

参数说明如下：

（1)options：

executable Java executable from which the core dump was produced.

(可能是产生core dump的java可执行程序)

core 将被打印信息的core dump文件

remote-hostname-or-IP 远程debug服务的主机名或ip

server-id 唯一id,假如一台主机上多个远程debug服务

（2）基本参数：

-F当’jstack [-l] pid’没有相应的时候强制打印栈信息

-l长列表. 打印关于锁的附加信息,例如属于java.util.concurrent的ownable synchronizers列表.

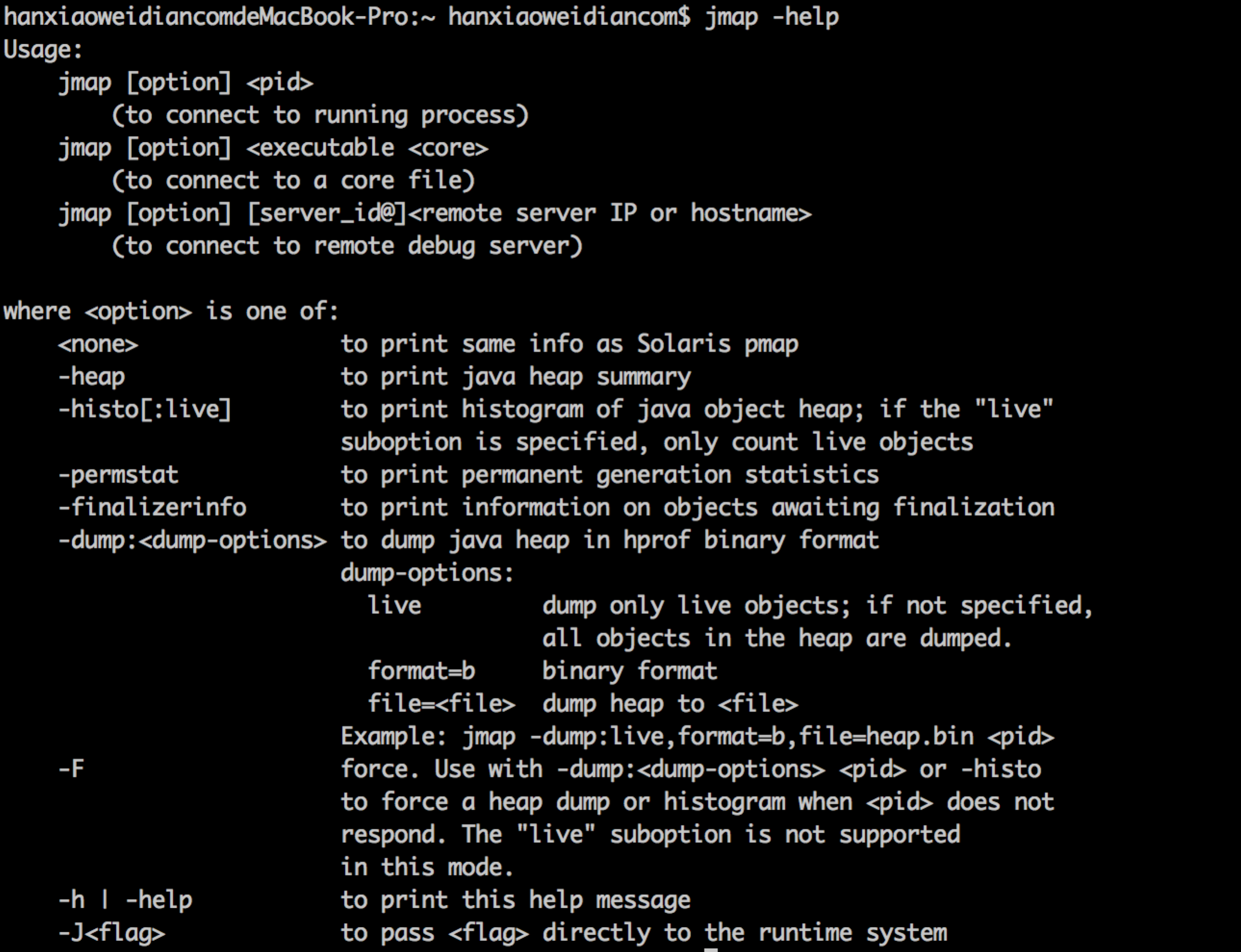
-m打印java和native c/c++框架的所有栈信息.

-h | -help打印帮助信息

pid 需要被打印配置信息的java进程id,可以用ps查询.

三 、jmap——java内存映像工具

1. 介绍：jmap命令用于生成堆转储快照（一般称为heapdump或dump文件），jmap的作用并不仅仅是为了获取dump文件，它还可以查询finalize执行队列、java堆和永久代的详细信息，如空间使用率，当前用的哪种收集器等。
2. 使用说明及参数说明：



（1）options：

executable Java executable from which the core dump was produced.

(可能是产生core dump的java可执行程序)

core 将被打印信息的core dump文件

remote-hostname-or-IP 远程debug服务的主机名或ip

server-id 唯一id,假如一台主机上多个远程debug服务

（2）基本参数：

-dump:[live,]format=b,file=<filename> 使用hprof二进制形式,输出jvm的heap内容到文 件. live子选项是可选的，假如指定live选项,那么只输出活的对象到文件.

-finalizerinfo 打印正等候回收的对象的信息.

-heap 打印heap的概要信息，GC使用的算法，heap的配置及wise heap的使用情况.

-histo[:live] 打印每个class的实例数目,内存占用,类全名信息. VM的内部类名字开头会加上前缀”\*”. 如果live子参数加上后,只统计活的对象数量.

-permstat 打印classload和jvm heap长久层的信息. 包含每个classloader的名字,活泼性,地址,父classloader和加载的class数量. 另外,内部String的数量和占用内存数也会打印出来.

-F 强迫.在pid没有相应的时候使用-dump或者-histo参数. 在这个模式下,live子参数无效.

-h | -help 打印辅助信息

-J 传递参数给jmap启动的jvm.

pid 需要被打印配相信息的java进程id,创业与打工的区别 - 博文预览,可以用ps查问.