**JDK命令:监控和故障处理工具**

|  |  |
| --- | --- |
| **名称** | **主要作用** |
| jps | 显示指定系统内的所有的Hotspot虚拟机进程 |
| jstat | 用于收集Hotspot虚拟机各方面的运行数据 |
| jinfo | 显示虚拟机配置信息 |
| jmap | 生成虚拟机的内存存储快照(heapdump文件) |
| jhat | 用于分析heapdump文件，可以在浏览器中查看相关信息 |
| jstack | 显示虚拟机的线程快照 |

# 1、JPS

|  |  |
| --- | --- |
| **选项** | **作用** |
| -q | 只输出LVMD，省略主要类的名称 |
| -m | 输出虚拟机启动时传递给主类main()函数的参数 |
| -l | 输出主类的全名，如果进程执行的是jar包，则输出Jar路径 |
| -v | 输出虚拟机进程启动时JVM的参数 |
| 常用：jps -l | |

# 2、Jstat

|  |  |
| --- | --- |
| **选项** | **作用** |
| -class | 监视类装载、卸载数量、总空间以及类装载所消耗的时间 |
| -gc | 监视java堆状况，包括Eden和两个survivor区、老年代、永久代等的容量、已用空间、GC时间合计等信息 |
| -gccapacity | 监视内容与-gc相同，但输出主要关注java堆各个区域使用到的最大最小空间 |
| **-gcutil** | **监视内容与-gc基本相同，但输出主要关注已使用空间占总空间的百分比** |
| -gccause | 与-gcutil功能一样，但是会额外输出导致上一次GC产生的原因 |
| -gcnew | 监视新生代GC状况 |
| -gcnewcapacity | 与gcnew基本相同，输出主要关注使用到的最大最小空间 |
| -gcold | 监视老年代GC状况 |
| -gcpermcapacity | 输出永久代使用到的最大最小空间 |
| -compiler | 输出JIT编译器编译过的方法、耗时等信息 |
| -printcompilation | 输出已经被JIT编译的方法 |

**Eg：jstat -gcutil 3876 总结垃圾回收统计**



由上图可看到两个survivor区（S0和S1）区分别占用了0和19%。E表示新生代区（Eden）占用了26.84%，O表示老年代（old）占用了58.95%，YGC表示（Yong GC）次数为37次，YGCT表示Yong GC Time总耗时为0.972秒，FGC表示（Full GC）次数为4次，FGCT表示Full GC Time总耗时1.207秒。GCT表示所有GC的总耗时为2.178秒。CCS表示压缩类空间

**Eg：jstat -gc 3876 垃圾回收统计**



S0C：第一个幸存区大小；S1C第二个幸存区大小；

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：伊甸园区大小；EU：伊甸园区使用大小；

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：Yong GC 年轻代GC次数；YGCT：年轻代GC总耗时；

FGC：老年代GC次数；FGCT：老年代GC总耗时；

GCT：GC总耗时 GCT=YGCT+FGCT

**Eg：jstat - gccapacity 3876 堆内存统计**



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：第二个幸存区大小；EC：伊甸园区大小；

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

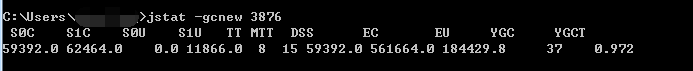
CCSMN：最小压缩类空间大小；CCSMX：最大压缩类空间大小；CCSC：当前压缩类空间大小

YGC：年轻代GC次数；FGC：老年代GC次数

**Eg：jstat - gccause 3876 上次GC产生原因**



**Eg：jstat -gcnew 3876 新生代垃圾回收统计**



S0C：第一个幸存区大小；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

TT：对象在新生代存活的次数；MTT：对象在新生代存活的最大次数

DSS：期望的幸存区大小

EC：伊甸园区的大小；EU：伊甸园区的使用大小

YGC：年轻代垃圾回收次数；YGCT：年轻代垃圾回收消耗时间

**Eg：jstat -gcnewcapacity 3876 新生代内存统计**



NGCMN：新生代最小容量；NGCMX：新生代最大容量；NGC：当前新生代容量

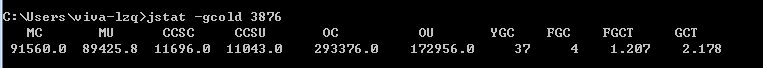
S0CMX：最大幸存1区大小；S0C：当前幸存1区大小

S1CMX：最大幸存2区大小；S1C：当前幸存2区大小

ECMX：最大伊甸园区大小；EC：当前伊甸园区大小

YGC：年轻代垃圾回收次数；FGC：老年代回收次数

**Eg：jstat -gcold 3876 老年代垃圾回收统计**



MC：方法区大小；MU：方法区使用大小

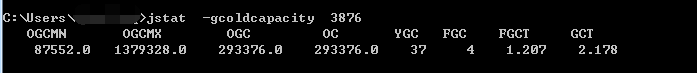
CCSC:压缩类空间大小；CCSU:压缩类空间使用大小

OC：老年代大小；OU：老年代使用大小

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

**Eg：jstat -gcoldcapacity 3876 老年代内存统计**



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：老年代大小

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

**Eg：jstat -compiler 3876 编译统计**



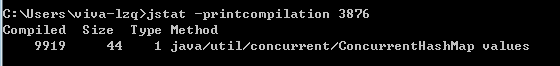
Compiled：编译数量。；Failed：失败数量

Invalid：不可用数量

Time：时间

FailedType：失败类型；FailedMethod：失败的方法

**Eg：jstat -printcompilation 3876 JVM编译方法统计**



# 3、Jinfo

------实时的查看和调整虚拟机的各项参数

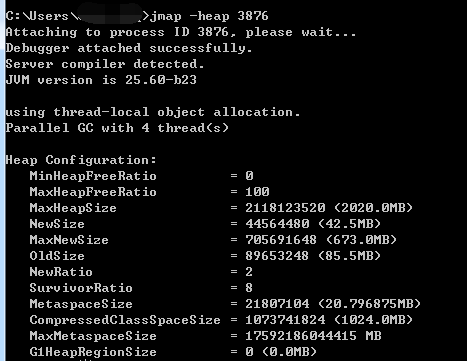


# 4、Jmap

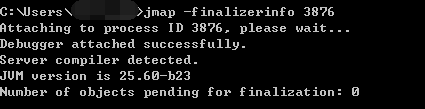
------内存映像工具

|  |  |
| --- | --- |
| **选项** | **作用** |
| -dump | 生成java对存储快照，其中live子参数说明是否只dump出存活对象 |
| -finalizerinfo | 显示在F-Queue中等待Finalizer线程执行Finalize方法的对象（Linux下） |
| -heap | 显示java的堆信息，如使用哪种回收器、参数配置、分代状况等（Linux下） |
| -histo | 显示堆中对象统计信息，包括类、实例数量、合计容量 |
| -permstat | 以ClassLoader为统计口径显示永久代内存状态（Linux下） |
| -F | 当虚拟机进程对dump选项没有回应时，使用F强制生成快照（Linux下） |

**Eg：jmap -heap 3876 查看3876的堆信息**



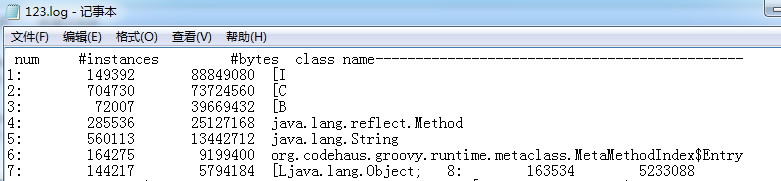
**Eg：jmap -finalizerinfo 3876 即将垃圾回收对象个数**



**Eg：jmap -dump:format=b,file=D:/dump.log 3876 生成堆dump文件到filename文件夹下**



**Eg：jmap -histo 3876 >d:/123.log 实例个数以及占用内存大小**

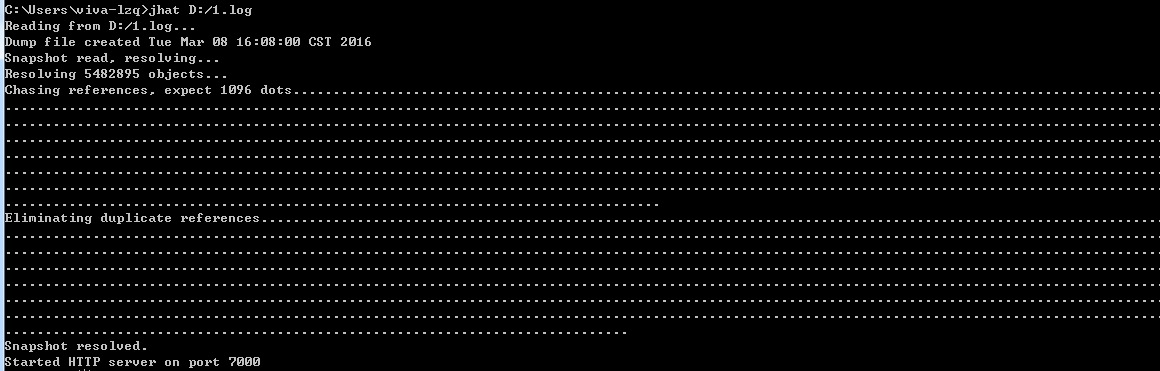


num：序号；instances：实例数量；bytes：占用空间大小；class name：类名称

# 5、Jhat

------虚拟机堆存储快照分析工具

**Eg：jhat D:/dump.log 然后在浏览器中输入：localhost:7000即可查看堆中的对象信息**



# 6、Jstack

------java堆栈跟踪工具

|  |  |
| --- | --- |
| **选项** | **作用** |
| -F | 当正常的请求不被响应时，强制输出线程堆栈 |
| -l | 除堆栈外，显示关于锁的附加信息 |
| -m | 如果调用本地方法的话，可以显示c/c++的堆栈 |

**Eg：jstack -l 3876**

线程的状态有RUNNABLE，TIMED\_WAITING，WAITING。

* WAITING状态下有两种情况，一种是获得了对象锁，但是某些条件不达成而调用了wait方法。另外一种是对象锁被其他线程占用，此线程等待该锁释放。
* TIMED\_WAITING状态表示调用了线程的sleep方法，此线程会睡一段时间。
* RUNNABLE状态表示线程正在执行，也有两种情况第一种是线程真正的在执行，另外一种是线程因为IO而阻塞。

# 7、Linux下性能分析

查看Linux下的进程ID：ps -ef|grep "进程名"

查看Linux下指定进程的线程信息：top -Hp '进程ID' | ps -mp "进程ID"

将十进制转化为十六进制：printf "%x\n" 线程ID

采用jstack分析该进程： jstack "进程ID "| grep "线程ID"

8、GC相关参数

8.1 Trace跟踪参数

-verbose:gc -XX:+printGC 可打印GC的简要信息

