第02章：JVM监控及诊断工具-命令行篇

第02章：JVM监控及诊断工具-命令行篇 1

1 01-概述 4

1.1 简单命令行工具 4

2 02-jps：查看正在运行的Java进程 6

2.1 基本情况 6

2.2 测试 6

2.3 基本语法 6

2.3.1 options参数 7

2.3.2 hostid参数 8

3 03-jstat：查看JVM统计信息 8

3.1 基本情况 8

3.2 基本语法 8

3.2.1 option参数 9

-gc 12

新生代相关 12

S0C是第一个幸存者区的大小（字节） 12

S1C是第二个幸存者区的大小（字节） 12

S0U是第一个幸存者区已使用的大小（字节） 12

S1U是第二个幸存者区已使用的大小（字节） 12

EC是Eden空间的大小（字节） 12

EU是Eden空间已使用大小（字节） 12

老年代相关 12

OC是老年代的大小（字节） 12

OU是老年代已使用的大小（字节） 12

方法区（元空间）相关 12

MC是方法区的大小 12

MU是方法区已使用的大小 12

CCSC是压缩类空间的大小 12

CCSU是压缩类空间已使用的大小 12

其他 13

YGC是从应用程序启动到采样时young gc的次数 13

YGCT是指从应用程序启动到采样时young gc消耗时间（秒） 13

FGC是从应用程序启动到采样时full gc的次数 13

FGCT是从应用程序启动到采样时的full gc的消耗时间（秒） 13

GCT是从应用程序启动到采样时gc的总时间 13

3.2.2 interval参数 13

用于指定输出统计数据的周期，单位为毫秒。即：查询间隔 13

3.2.3 count参数 13

用于指定查询的总次数 13

3.2.4 -t参数 13

可以在输出信息前加上一个Timestamp列，显示程序的运行时间。单位：秒 13

经验 13

3.2.5 -h参数 15

可以在周期性数据输出时，输出多少行数据后输出一个表头信息 15

3.3 补充 15

4 04-jinfo：实时查看和修改JVM配置参数 15

4.1 基本情况 15

4.2 基本语法 16

4.2.1 查看 17

jinfo -sysprops 进程id 17

可以查看由System.getProperties()取得的参数 17

jinfo -flags 进程id 17

查看曾经赋过值的一些参数 18

jinfo -flag 参数名称 进程id 18

查看某个java进程的具体参数信息 18

4.2.2 修改 18

针对boolean类型 19

jinfo -flag [+|-]参数名称 进程id 19

针对非boolean类型 19

jinfo -flag 参数名称=参数值 进程id 19

4.3 拓展 20

4.3.1 java -XX:+PrintFlagsInitial 20

查看所有JVM参数启动的初始值 20

4.3.2 java -XX:+PrintFlagsFinal 20

查看所有JVM参数的最终值 21

4.3.3 java -参数名称:+PrintCommandLineFlags 21

查看那些已经被用户或者JVM设置过的详细的XX参数的名称和值 21

5 05-jmap：导出内存映像文件&内存使用情况 21

5.1 基本情况 21

5.2 基本语法 22

5.2.1 -dump 23

生成Java堆转储快照：dump文件 23

特别的：-dump:live只保存堆中的存活对象 23

5.2.2 -heap 23

输出整个堆空间的详细信息，包括GC的使用、堆配置信息，以及内存的使用信息等 23

5.2.3 -histo 23

输出堆中对象的同级信息，包括类、实例数量和合计容量 23

特别的：-histo:live只统计堆中的存活对象 23

5.2.4 -permstat 23

以ClassLoader为统计口径输出永久代的内存状态信息 23

仅linux/solaris平台有效 23

5.2.5 -finalizerinfo 23

显示在F-Queue中等待Finalizer线程执行finalize方法的对象 23

仅linux/solaris平台有效 23

5.2.6 -F 23

当虚拟机进程对-dump选项没有任何响应时，可使用此选项强制执行生成dump文件 24

仅linux/solaris平台有效 24

5.2.7 -h | -help 24

jamp工具使用的帮助命令 24

5.2.8 -J <flag> 24

传递参数给jmap启动的jvm 24

5.3 使用1：导出内存映像文件 24

5.3.1 手动的方式 25

jmap -dump:format=b,file=<filename.hprof> <pid> 26

jmap -dump:live,format=b,file=<filename.hprof> <pid> 26

5.3.2 自动的方式 26

-XX:+HeapDumpOnOutOfMemoryError 27

-XX:HeapDumpPath=<filename.hprof> 27

5.4 使用2：显示堆内存相关信息 27

5.4.1 jmap -heap 进程id 27

5.4.2 jmap -histo 进程id 27

5.5 使用3：其他作用 28

5.5.1 jmap -permstat 进程id 28

查看系统的ClassLoader信息 28

5.5.2 jmap -finalizerinfo 28

查看堆积在finalizer队列中的对象 28

5.6 小结 28

6 06-jhat：JDK自带堆分析工具 28

6.1 基本情况 29

6.2 基本语法 29

6.2.1 options参数 30

7 07-jstack：打印JVM中线程快照 31

7.1 基本情况 31

7.2 基本语法 32

7.2.1 option参数：-F 33

当正常输出的请求不被响应时，强制输出线程堆栈 33

7.2.2 option参数：-l 33

除堆栈外，显示关于锁的附加信息 33

7.2.3 option参数：-m 33

如果调用本地方法的话，可以显示C/C++的堆栈 33

7.2.4 option参数：-h 33

帮助操作 33

8 08-jcmd：多功能命令行 33

8.1 基本情况 34

8.2 基本语法 34

8.2.1 jcmd -l 34

列出所有的JVM进程 34

8.2.2 jcmd 进程号 help 34

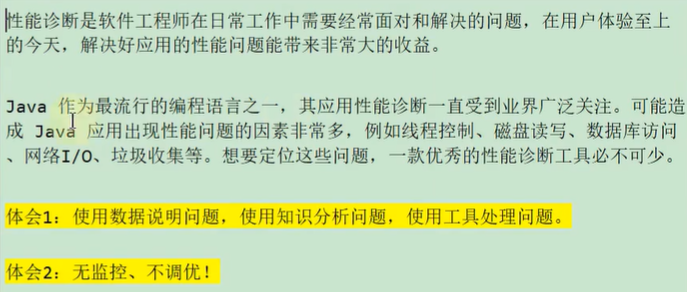
针对指定的进程，列出支持的所有具体命令 35

8.2.3 jcmd 进程号 具体命令 35

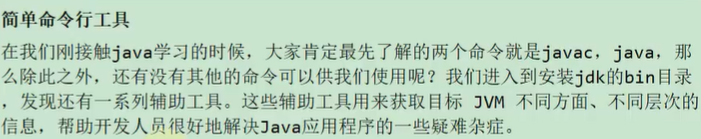
显示指定进程的指令命令的数据 37

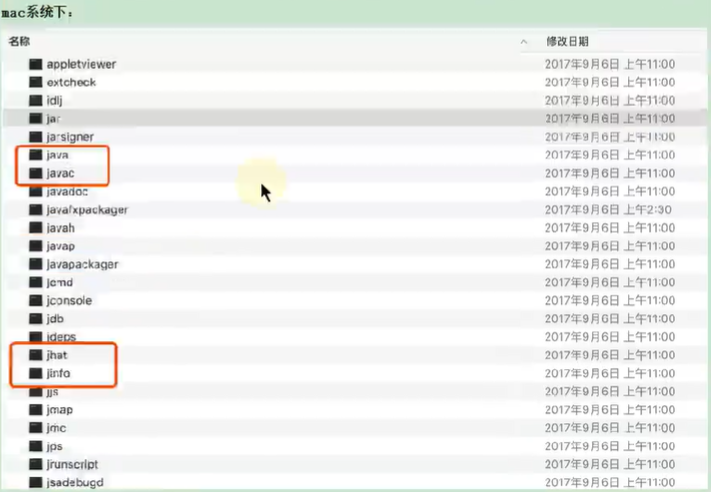
9 09-jstatd：远程主机信息收集 37

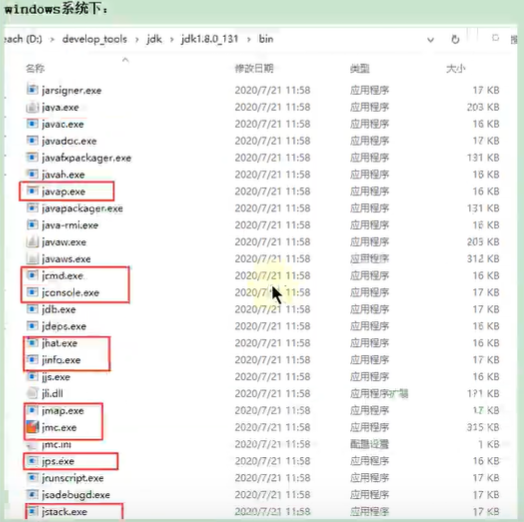
1. 01-概述



* 1. 简单命令行工具

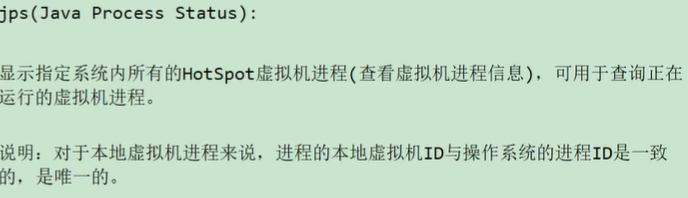








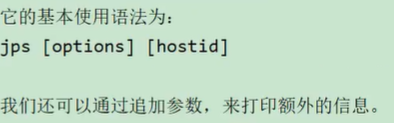
1. 02-jps：查看正在运行的Java进程
   1. 基本情况



* 1. 测试

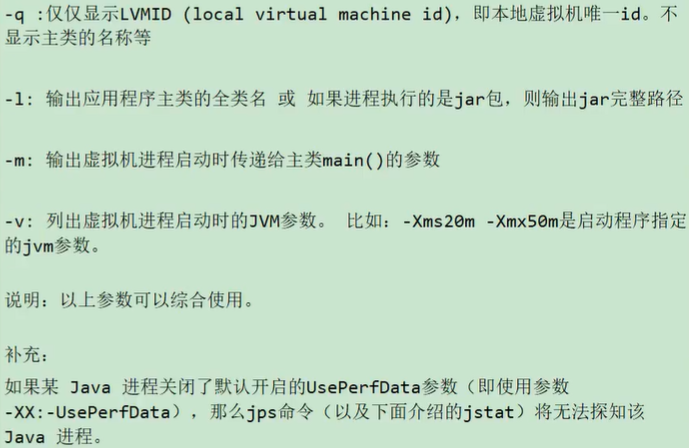
根据基本语法进行测试即可

* 1. 基本语法



可以通过 jps -help 来查看对应的参数信息

* + 1. options参数



**综合使用：**

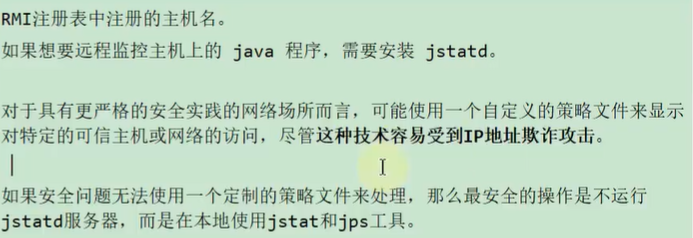
jps -l -m等价于jps -lm

**如何将信息输出到同级文件中：**

语法：命令 > 文件名称

例如：jps -l > a.txt

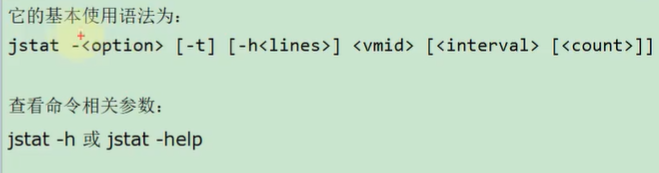
* + 1. hostid参数



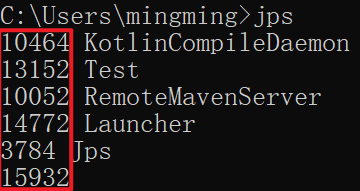
1. 03-jstat：查看JVM统计信息
   1. 基本情况



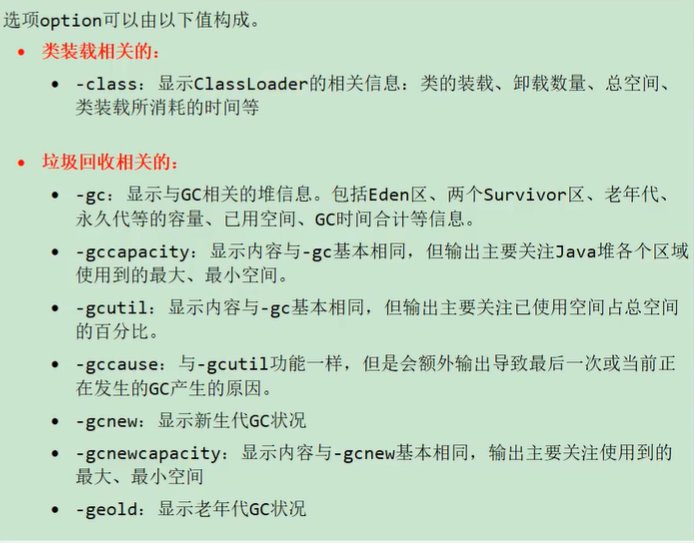
* 1. 基本语法

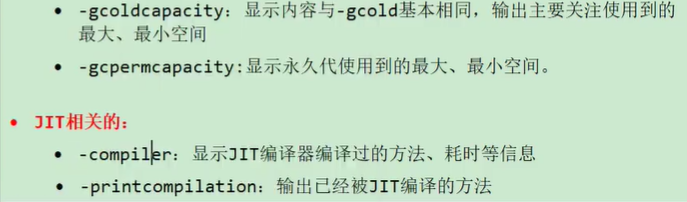


其中vmid是进程id号，也就是jps之后看到的前面的号码，如下：

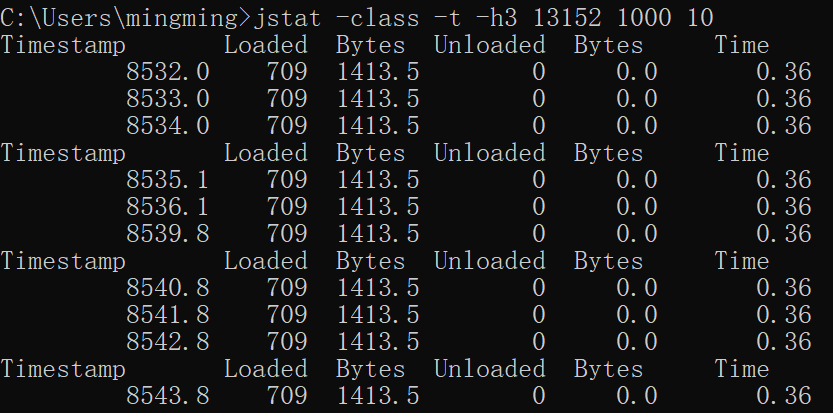


* + 1. option参数



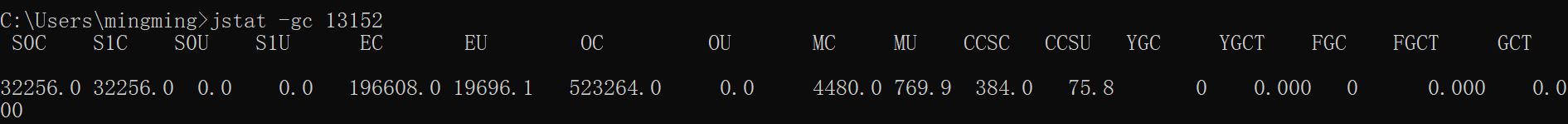


**1、-class举例：**jstat -class -t -h3 13152 1000 10，其中h3中的3代表每隔3个分隔一次，13152代表类的进程id，1000代表每隔1000毫秒打印一次，10代表一共打印10次，如下所示：



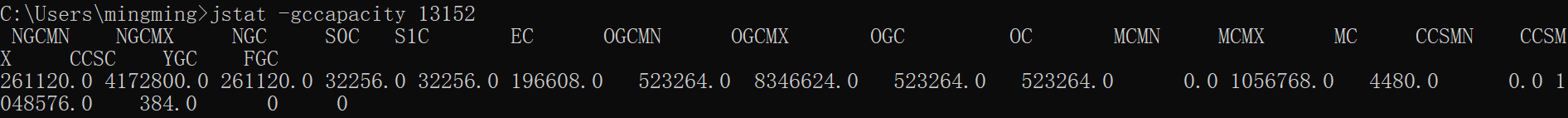
其中Timestamp代表程序至今的运行时间，单位为秒；Loaded代表加载的类的数目；Bytes代表加载的类的总字节数；Unloaded代表卸载的类的数目；Bytes代表卸载的类的总字节数；Time代表类装载所消耗的时间；

**2、-gc举例：**jstat -gc 13152，其中13152代表类的进程id，执行结果如下所示：



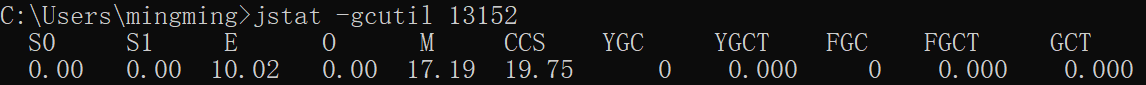
其中S0C代表幸存者0区的总容量，S1C代表幸存者1区的总容量，S0U代表幸存者0区使用的容量，S1U代表幸存者1区使用的容量，EC代表伊甸园区的总容量，EU代表伊甸园区使用的总容量，OC代表老年代的总容量，OU代表老年代已经使用的容量，MC代表方法区的总容量，MU代表方法区的总容量，CCSC代表压缩类的总容量，CCSU代表压缩类使用的容量，YGC代表年轻代垃圾回收的次数，YGCT年轻代进行垃圾回收需要的时间，FGC代表代表Full GC的次数，FGCT代表Full GC的时间，GCT代表垃圾回收的总时间

**3、-gccapacity举例：**jstat -gccapacity 13152，其中13152代表类的进程id，执行结果如下：



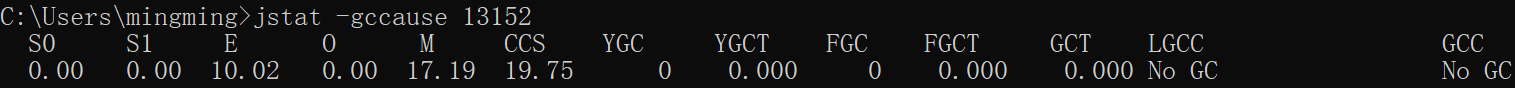
其中S0C代表幸存者0区的容量，S1C代表幸存者1区的容量，EC代表伊甸园区的容量，CCSC代表压缩类的容量，YGC代表年轻代垃圾回收的时间，FGC代表Full GC垃圾回收的时间

**4、-gcutil举例：**jstat -gcutil 13152，其中13152代表类的进程id，执行结果如下所示：



以上是各区域占比以及垃圾回收的情况，S0代表幸存者0区，S1代表幸存者1区，E代表伊甸园区，O代表老年代，M代表方法区，CCS代表压缩类，以上这些值都是占比情况，YGC代表年轻代垃圾回收的次数，YGCT年轻代进行垃圾回收需要的时间，FGC代表代表Full GC的次数，FGCT代表Full GC的时间，GCT代表垃圾回收的总时间

**5、-gccause举例：**jstat -gccause 13152，其中13152代表类的进程id，执行结果如下：



以上是各区域占比以及垃圾回收的情况，还有触发垃圾回收的原因解释，S0代表幸存者0区，S1代表幸存者1区，E代表伊甸园区，O代表老年代，M代表方法区，CCS代表压缩类，以上这些值都是占比情况，YGC代表年轻代垃圾回收的次数，YGCT年轻代进行垃圾回收需要的时间，FGC代表代表Full GC的次数，FGCT代表Full GC的时间，GCT代表垃圾回收的总时间，LGCC和GCC代表垃圾回收的原因

-gc

新生代相关

S0C是第一个幸存者区的大小（字节）

S1C是第二个幸存者区的大小（字节）

S0U是第一个幸存者区已使用的大小（字节）

S1U是第二个幸存者区已使用的大小（字节）

EC是Eden空间的大小（字节）

EU是Eden空间已使用大小（字节）

老年代相关

OC是老年代的大小（字节）

OU是老年代已使用的大小（字节）

方法区（元空间）相关

MC是方法区的大小

MU是方法区已使用的大小

CCSC是压缩类空间的大小

CCSU是压缩类空间已使用的大小

其他

YGC是从应用程序启动到采样时young gc的次数

YGCT是指从应用程序启动到采样时young gc消耗时间（秒）

FGC是从应用程序启动到采样时full gc的次数

FGCT是从应用程序启动到采样时的full gc的消耗时间（秒）

GCT是从应用程序启动到采样时gc的总时间

* + 1. interval参数

用于指定输出统计数据的周期，单位为毫秒。即：查询间隔

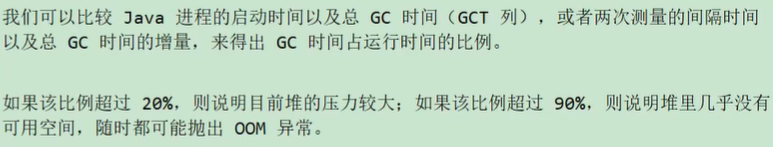
* + 1. count参数

用于指定查询的总次数

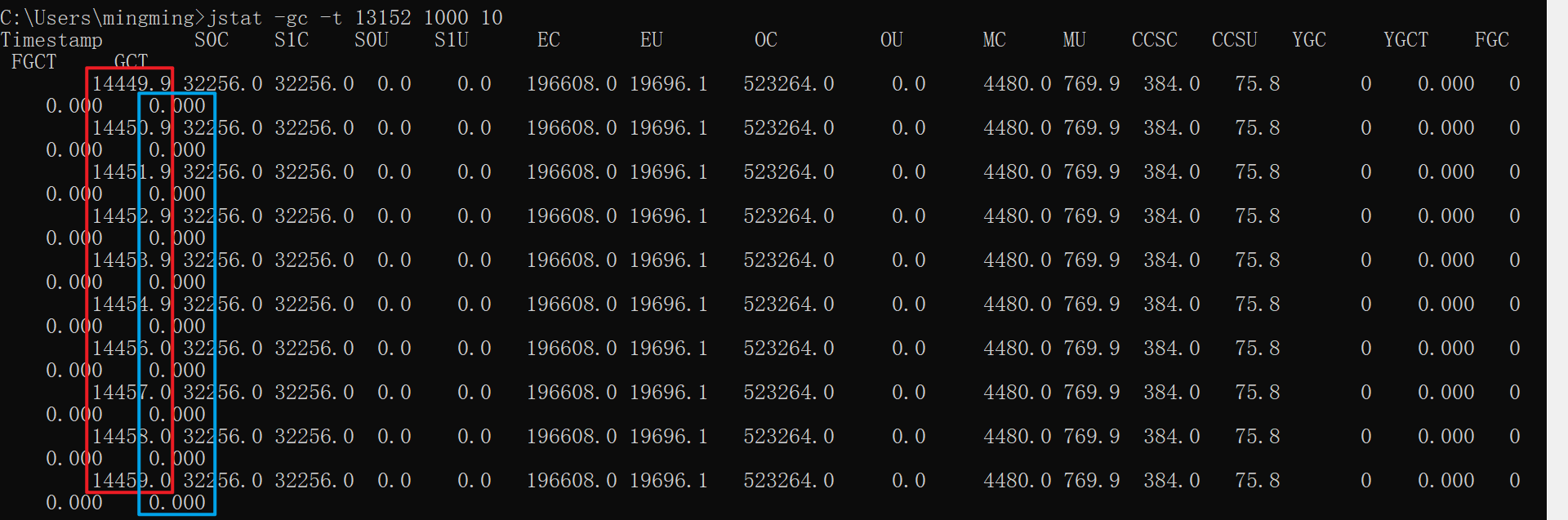
* + 1. -t参数

可以在输出信息前加上一个Timestamp列，显示程序的运行时间。单位：秒

经验



我们执行jstat -gc -t 13152 1000 10，这代表1秒打印出1行，一共10行，-t代表打印出Timestamp总运行时间，结果如下所示：



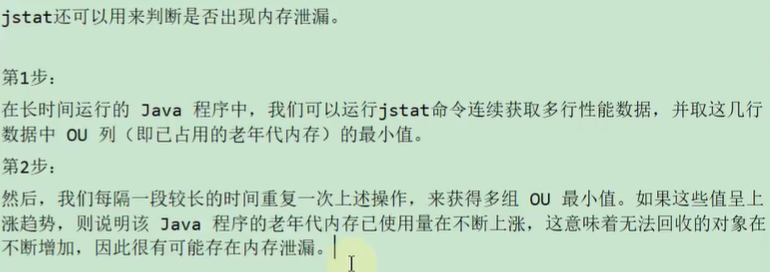
上方红色框框中代表Timestamp，而蓝色框框中代表垃圾回收时间，单位都是秒，如果让红色框框中的某两个值相减，假设这个值是num1，然后让对应行的蓝色框框中的另外两个值相减，假设这个值是num2，之后让num2/num1，得出的差值就是上述所说的GC时间占运行时间的比例

虽然这种方式比较繁琐，但是在项目部署之后就需要使用命令行去看了，就没有可视化界面了，所以这种方式也要会

* + 1. -h参数

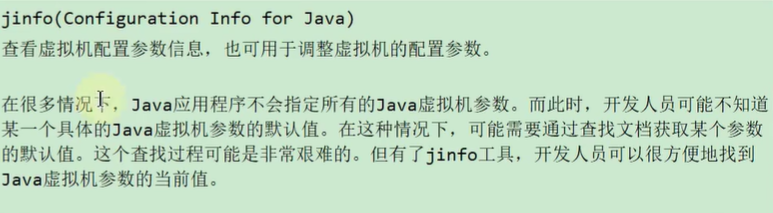
可以在周期性数据输出时，输出多少行数据后输出一个表头信息

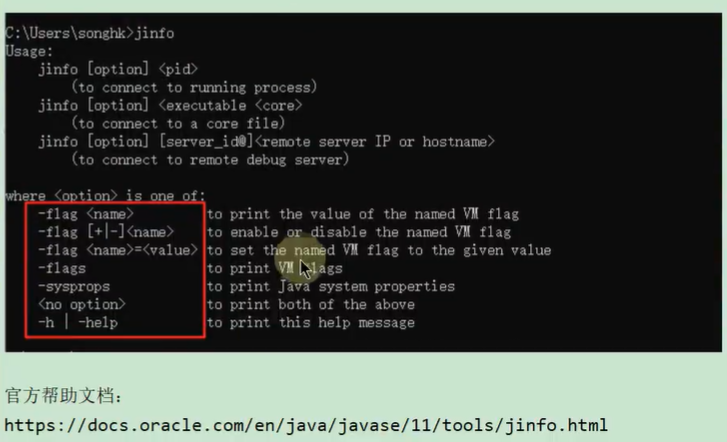
* 1. 补充



第1步可以执行命令：jstat -gc -t 13152 1000 10

1. 04-jinfo：实时查看和修改JVM配置参数
   1. 基本情况





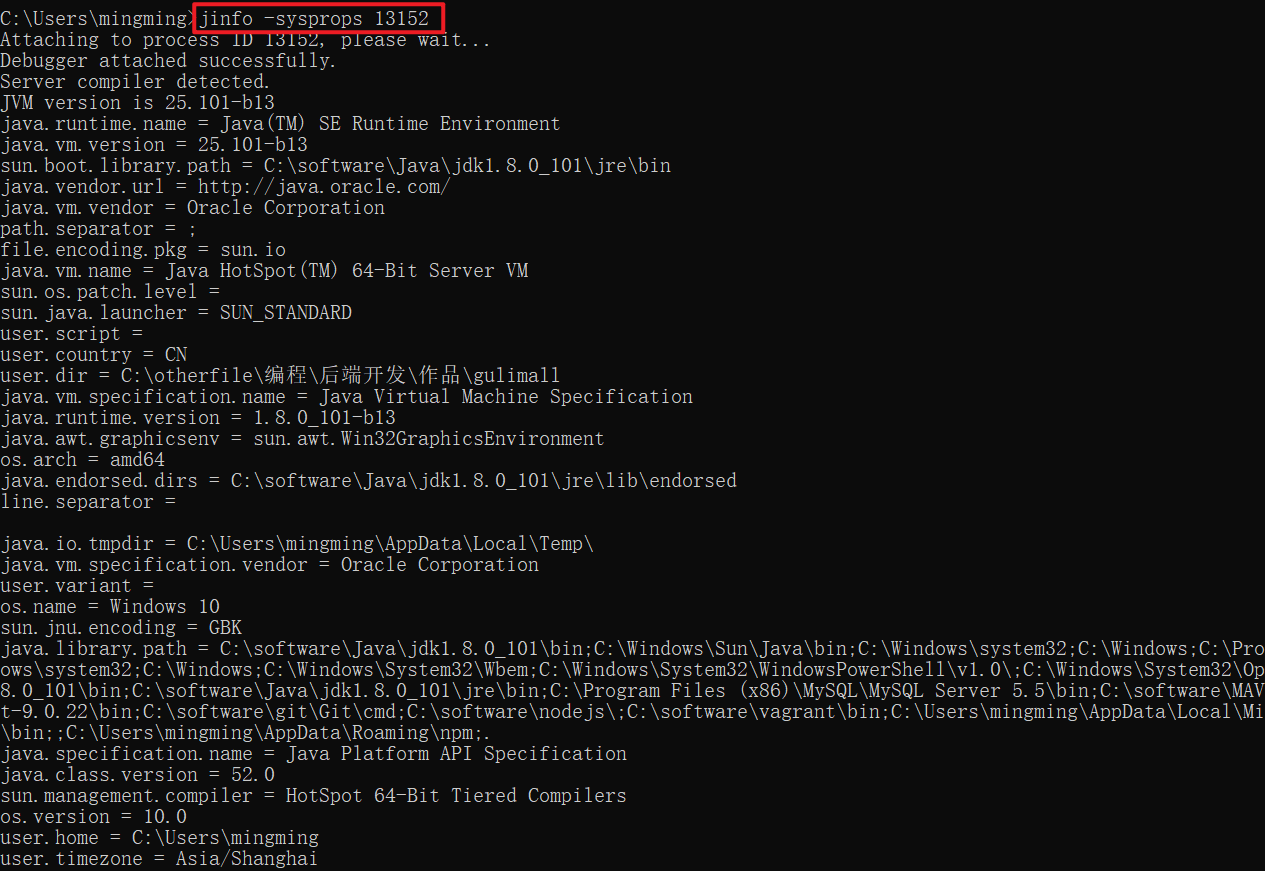
* 1. 基本语法



* + 1. 查看

jinfo -sysprops 进程id

进程id可以通过jps命令查看，操作结果如下：

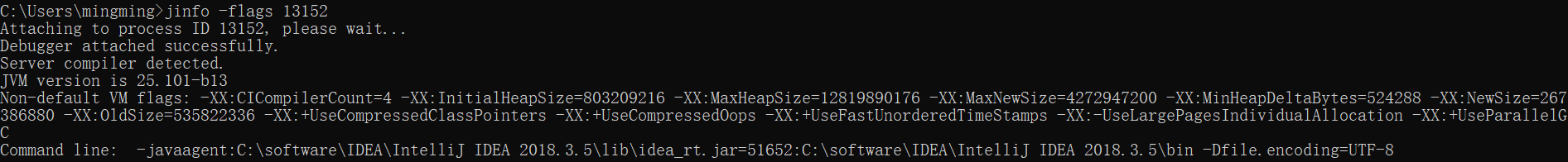


其中13152代表进程id

可以查看由System.getProperties()取得的参数

jinfo -flags 进程id

进程id可以通过jps命令查看，参数赋值的一部分是我们自己设置的，另外一部分是系统自动优化设置的参数信息，具体操作如下：



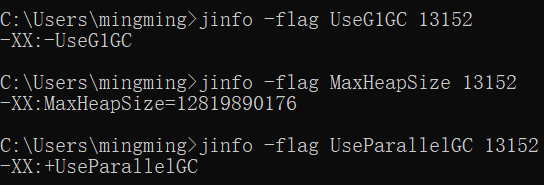
其中13152代表进程id

查看曾经赋过值的一些参数

jinfo -flag 参数名称 进程id

进程id可以通过jps命令查看

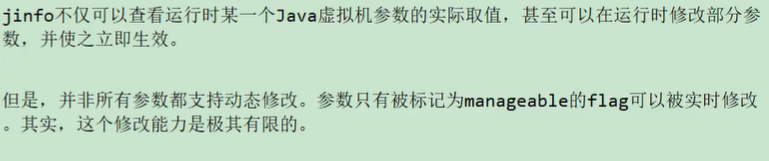
**举例：**



其中13152代表进程id

查看某个java进程的具体参数信息

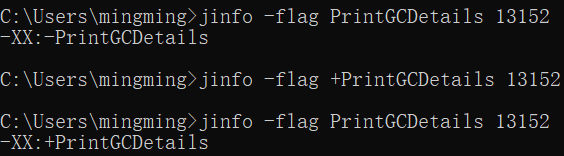
* + 1. 修改

针对boolean类型

jinfo -flag [+|-]参数名称 进程id

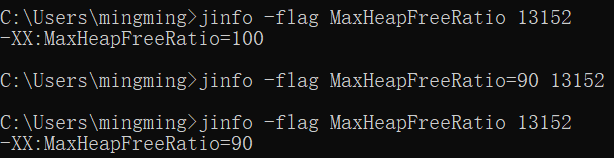
PID可以通过jps命令查看，如果使用+号，那就可以让该参数起作用，否则使用-号就让该参数不起作用，具体例子如下：



针对非boolean类型

jinfo -flag 参数名称=参数值 进程id

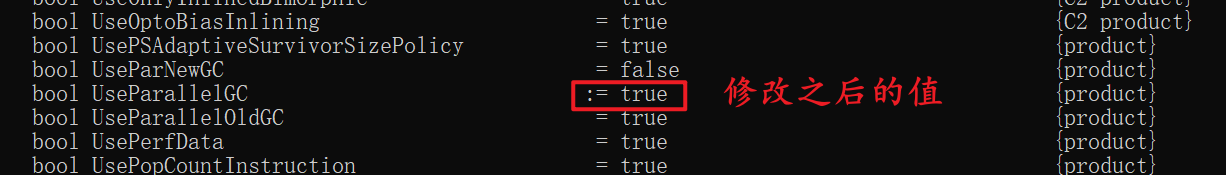
PID可以通过jps命令查看，如果使用+号，那就可以让该参数起作用，否则使用-号就让该参数不起作用，具体例子如下：



* 1. 拓展
     1. java -XX:+PrintFlagsInitial

查看所有JVM参数启动的初始值

* + 1. java -XX:+PrintFlagsFinal



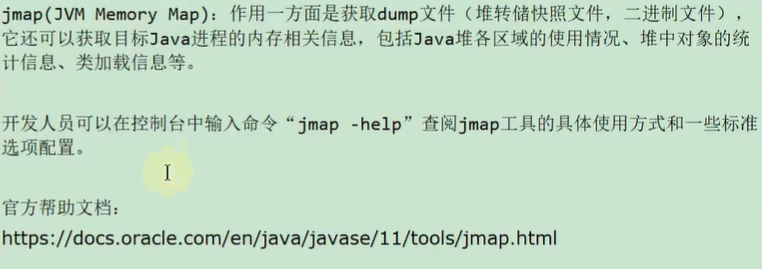
值前面添加冒号:的是修改之后的值，没有添加的都是没有发生改变的初始值

查看所有JVM参数的最终值

* + 1. java -参数名称:+PrintCommandLineFlags

查看那些已经被用户或者JVM设置过的详细的XX参数的名称和值

1. 05-jmap：导出内存映像文件&内存使用情况
   1. 基本情况



* 1. 基本语法



1. 使用语法可以通过在DOS窗口中使用jmap/jmap -h/jmap -help查看
2. <executable <core>代表可执行的代码，比如使用> 文件名称来指定生成的dump文件的生成位置
3. [server\_id@]<……>是为远程连接准备的
   * 1. -dump

生成Java堆转储快照：dump文件

特别的：-dump:live只保存堆中的存活对象

* + 1. -heap

输出整个堆空间的详细信息，包括GC的使用、堆配置信息，以及内存的使用信息等

* + 1. -histo

输出堆中对象的同级信息，包括类、实例数量和合计容量

特别的：-histo:live只统计堆中的存活对象

* + 1. -permstat

以ClassLoader为统计口径输出永久代的内存状态信息

仅linux/solaris平台有效

* + 1. -finalizerinfo

显示在F-Queue中等待Finalizer线程执行finalize方法的对象

仅linux/solaris平台有效

* + 1. -F

当虚拟机进程对-dump选项没有任何响应时，可使用此选项强制执行生成dump文件

仅linux/solaris平台有效

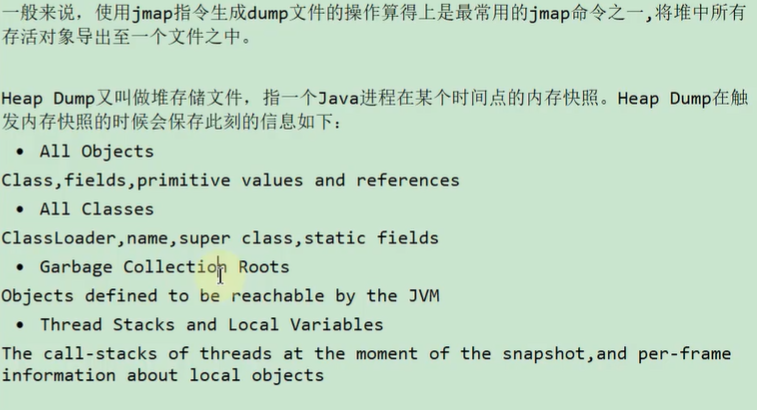
* + 1. -h | -help

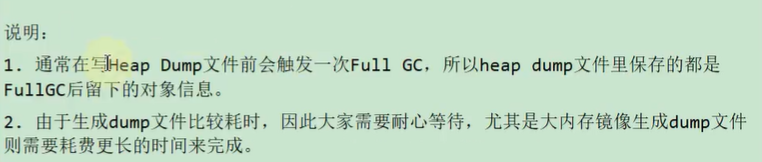
jamp工具使用的帮助命令

* + 1. -J <flag>

传递参数给jmap启动的jvm

* 1. 使用1：导出内存映像文件





**注意：**

1. 对于以上说明中的第1点是自动方式才会这样做，而手动不会在Full GC之后生成Dump
2. 使用手动方式生成dump文件，一般指令执行之后就会生成，不用等到快出现OOM的时候
3. 使用自动方式生成dump文件，当出现OOM之前先生成dump文件
4. 如果使用手动方式，一般使用第2种，毕竟生成堆中存活对象的dump文件是比较小的，便于传输和分析
   * 1. 手动的方式

**说明：**

1. <filename.hprof>中的filename是文件名称，而.hprof是后缀名，<\*\*\*>代表该值可以省略<>，当然后面的<pid>是进程id，需要通过jps查询出来
2. format=b表示生成的是标准的dump文件，用来进行格式限定

**具体例子如下：**

生成堆中所有对象的快照：

graphic

生成堆中存活对象的快照：

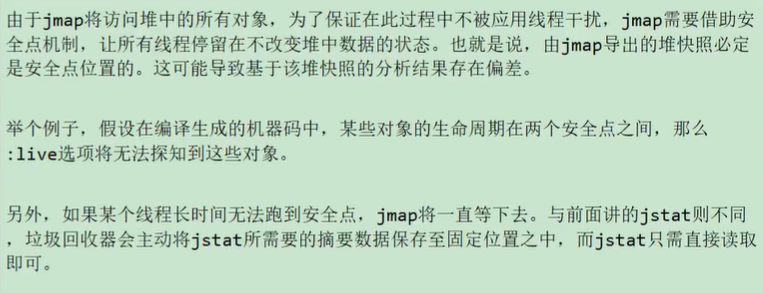
graphic

其中file=后面的是生成的dump文件地址，最后的11696是进程id，可以通过jps查看

一般使用的是第二种方式，也就是生成堆中存活对象的快照，毕竟这种方式生成的dump文件更小，我们传输处理都更方便

jmap -dump:format=b,file=<filename.hprof> <pid>

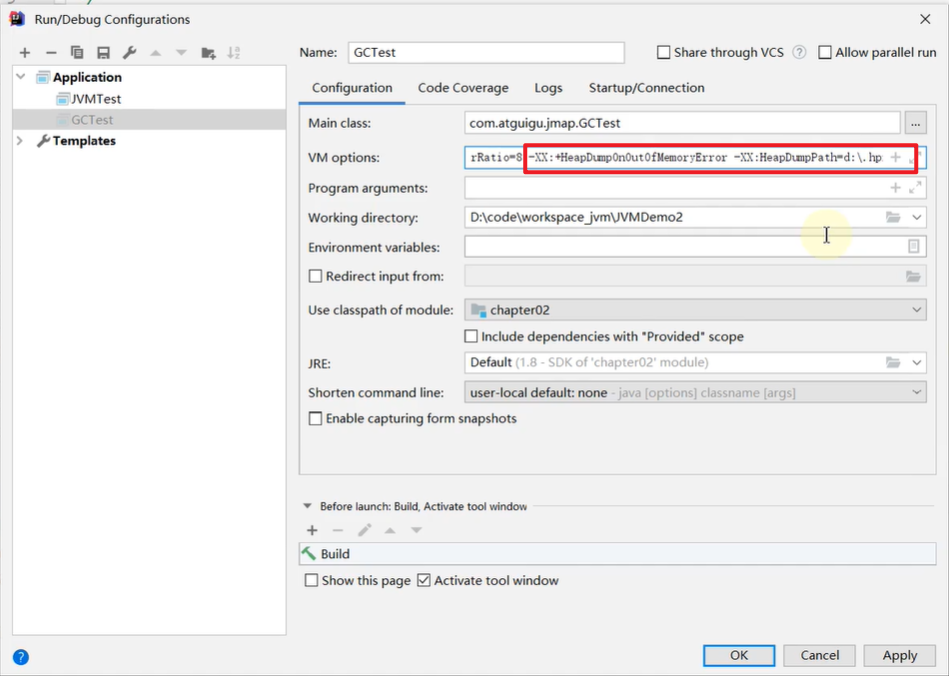
jmap -dump:live,format=b,file=<filename.hprof> <pid>



* + 1. 自动的方式



具体使用如下：



-XX:+HeapDumpOnOutOfMemoryError

-XX:HeapDumpPath=<filename.hprof>

* 1. 使用2：显示堆内存相关信息
     1. jmap -heap 进程id

1. jmap -heap 进程id只是时间点上的堆信息，而jstat后面可以添加参数，可以指定时间动态观察数据改变情况，而图形化界面工具，例如jvisualvm等，它们可以用图表的方式动态展示出相关信息，更加直观明了
2. 例子如下：

graphic

* + 1. jmap -histo 进程id

输出堆中对象的同级信息，包括类、实例数量和合计容量，也是这一时刻的内存中的对象信息

例子如下：

graphic

* 1. 使用3：其他作用

这两个指令仅linux/solaris平台有效，所以无法在windows操作平台上演示，并且使用比较小众，不在多说

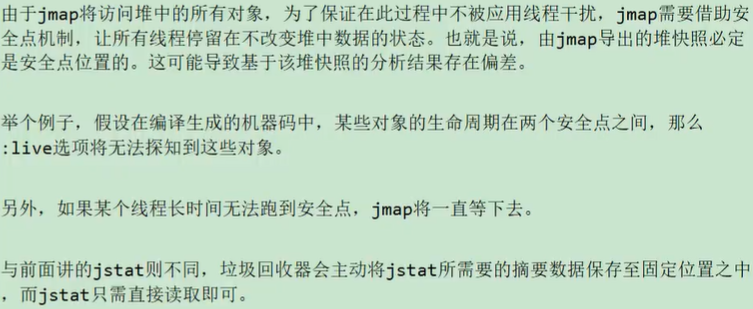
* + 1. jmap -permstat 进程id

查看系统的ClassLoader信息

* + 1. jmap -finalizerinfo

查看堆积在finalizer队列中的对象

* 1. 小结



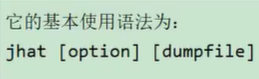
1. 06-jhat：JDK自带堆分析工具

jhat命令在jdk9及其之后就被移除了，官方建议使用jvisualvm代替jhat，所以该指令只需简单了解一下即可

* 1. 基本情况



* 1. 基本语法



其中dumpfile代表dump文件的地址以及名称，例如：

graphic

* + 1. options参数



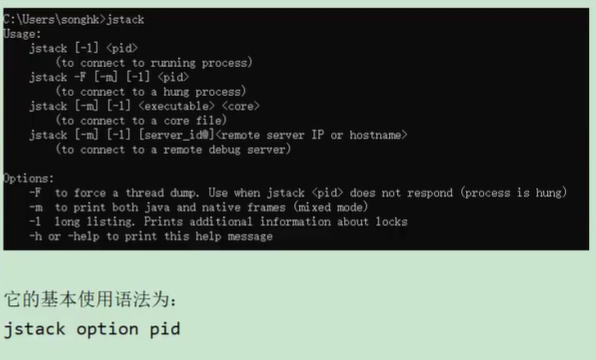
**举例如下：**

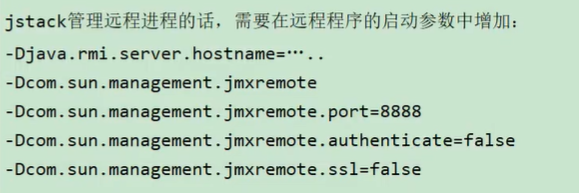
1. graphic
2. graphic
3. graphic
4. 07-jstack：打印JVM中线程快照
   1. 基本情况



graphic

* 1. 基本语法





**举例如下：**

1. graphic
2. 加-l参数：graphic

**总结：**

如果程序出现等待问题，可以使用该指令去查看问题所在，结果中也会提示你问题所在

* + 1. option参数：-F

当正常输出的请求不被响应时，强制输出线程堆栈

* + 1. option参数：-l

除堆栈外，显示关于锁的附加信息

* + 1. option参数：-m

如果调用本地方法的话，可以显示C/C++的堆栈

* + 1. option参数：-h

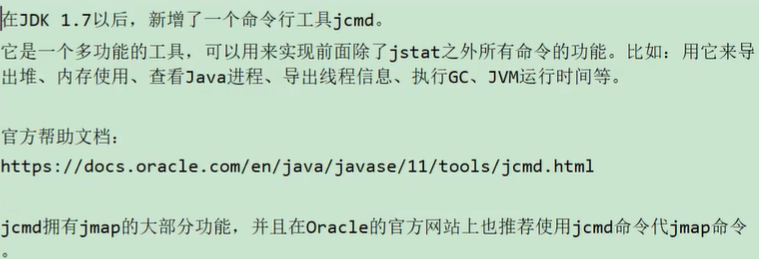
帮助操作

1. 08-jcmd：多功能命令行



一个顶多个，真的很强

* 1. 基本情况

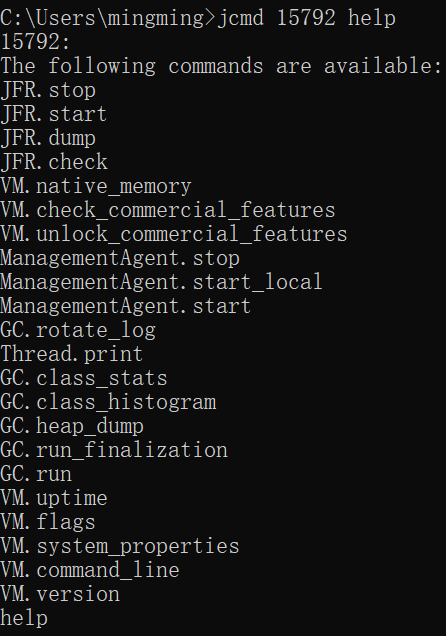


* 1. 基本语法
     1. jcmd -l

列出所有的JVM进程

* + 1. jcmd 进程号 help

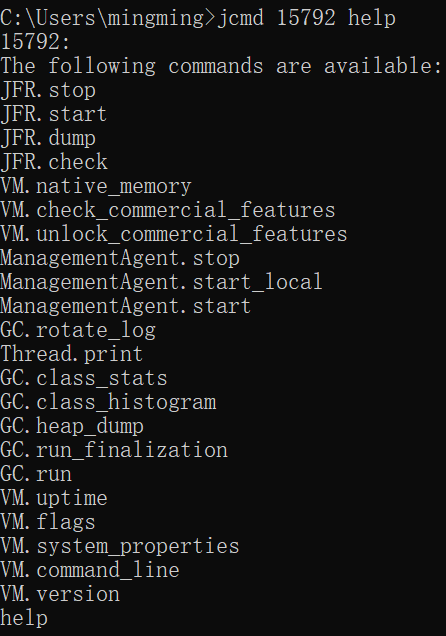
执行效果：



针对指定的进程，列出支持的所有具体命令

* + 1. jcmd 进程号 具体命令

首先通过jcmd 进程号 help得出以下命令列表：



根据以上命令来替换之前的那些操作：

1. Thread.print 可以替换 jstack指令
2. GC.class\_histogram 可以替换 jmap中的-histo操作
3. GC.heap\_dump 可以替换 jmap中的-dump操作
4. GC.run 可以查看GC的执行情况
5. VM.uptime 可以查看程序的总执行时间，可以替换jstat指令中的-t操作
6. VM.system\_properties 可以替换 jinfo -sysprops 进程id
7. VM.flags 可以获取JVM的配置参数信息

显示指定进程的指令命令的数据

1. 09-jstatd：远程主机信息收集

