**[jstat的使用方法 -- 分析JVM的使用情况](http://xianqi-h.iteye.com/blog/1346491)**

<http://xianqi-h.iteye.com/blog/1346491>

JDK自带VM分析工具jps，jstat，jmap，jconsole

关键字: jdk自带vm分析工具jps，jstat，jmap，jconsole

## 一、概述

SUN 的JDK中的几个工具，非常好用。秉承着有免费，不用商用的原则。以下简单介绍一下这几种工具。(注：本文章下的所有工具都存在JDK5.0以上版本的工具集里，同javac一样，不须特意安装) 。

我一共找到以下四个工具：重点看看jconsole和jmap。

jps

:与unix上的ps类似，用来显示本地的java进程，可以查看本地运行着几个java程序，并显示他们的进程号。

jstat

:一个极强的监视VM内存工具。可以用来监视VM内存内的各种堆和非堆的大小及其内存使用量。

jmap

:打印出某个java进程（使用pid）内存内的所有‘对象’的情况（如：产生那些对象，及其数量）。

jconsole

:一个java GUI监视工具，可以以图表化的形式显示各种数据。并可通过远程连接监视远程的服务器VM。

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

## 二、 使用介绍：       1、jps  ：我想很多人都是用过unix系统里的ps命令，这个命令主要是用来显示当前系统的进程情况，有哪些进程，及其 id。 jps 也是一样，它的作用是显示当前系统的java进程情况，及其id号。我们可以通过它来查看我们到底启动了几个java进程（因为每一个java程序都会独占一个java虚拟机实例），和他们的进程号（为下面几个程序做准备），并可通过opt来查看这些进程的详细启动参数。      使用方法：在当前命令行下打 jps(需要JAVA\_HOME，没有的话，到改程序的目录下打) 。  可惜没有linux下的ps好用，名称不好用。但是在第四个工具jconsole的界面里面会有具体JAR包的名称。

**2、jstat ：**对VM内存使用量进行监控。

jstat工具特别强大，有众多的可选项，详细查看堆内各个部分的使用量，以及加载类的数量。使用时，需加上查看进程的进程id，和所选参数。以下详细介绍各个参数的意义。   
    jstat -class pid:显示加载class的数量，及所占空间等信息。   
    jstat -compiler pid:显示VM实时编译的数量等信息。   
    jstat -gc pid: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 -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。   
    jstat -gcnewcapacity pid:new对象的信息及其占用量。   
    jstat -gcold pid:old对象的信息。   
    jstat -gcoldcapacity pid:old对象的信息及其占用量。   
    jstat -gcpermcapacity pid: perm对象的信息及其占用量。   
    jstat -util pid:统计gc信息统计。   
    jstat -printcompilation pid:当前VM执行的信息。   
    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。

**3、jmap** 是一个可以输出所有内存中对象的工具，甚至可以将VM 中的heap，以二进制输出成文本。使用方法 jmap -histo pid。如果连用 SHELL jmap -histo pid>a.log可以将其保存到文本中去（windows下也可以使用），在一段时间后，使用文本对比工具，可以对比出GC回收了哪些对象。 jmap -dump:format=b,file=f1 3024可以将3024进程的内存heap输出出来到f1文件里。   
       
    **4、jconsole** 是一个用java写的GUI程序，用来监控VM，并可监控远程的VM，非常易用，而且功能非常强。由于是GUI程序，这里就不详细介绍了，不会的地方可以参考ＳＵＮ的官方文档。   
    使用方法：命令行里打 jconsole，选则进程就可以了。   
       
    友好提示：windows查看进程号，由于任务管理器默认的情况下是不显示进程id号的，所以可以通过如下方法加上。ctrl+alt+del打开任务管理器，选择‘进程’选项卡，点‘查看’->''选择列''->加上''PID''，就可以了。当然还有其他很好的选项。

## 三、参考资料：

article:http://elf8848.javaeye.com/blog/442806   
  
  
    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

JDK之jstat的用法

jstat的用法   
  
  
用以判断JVM是否存在内存问题呢？如何判断JVM垃圾回收是否正常？一般的top指令基本上满足不了这样的需求，因为它主要监控的是总体的系统资源，很难定位到java应用程序。   
  
jstat 是JDK自带的一个轻量级小工具。全称“Java Virtual Machine statistics monitoring tool”，它位于java的bin目录下，主要利用JVM内建的指令对Java应用程序的资源和性能进行实时的命令行的监控，包括了对Heap size和垃圾回收状况的监控。可见，jstat是轻量级的、专门针对JVM的工具，非常适用。由于JVM内存设置较大，图中百分比变化不太明显   
  
一个极强的监视VM内存工具。可以用来监视VM内存内的各种堆和非堆的大小及其内存使用量。   
  
jstat工具特别强大，有众多的可选项，详细查看堆内各个部分的使用量，以及加载类的数量。使用时，需加上查看进程的进程id，和所选参数。   
  
执行：cd $JAVA\_HOME/bin中执行jstat，**注意jstat后一定要跟参数。**

语法结构：   
  
**Usage: jstat -help|-options   
  
       jstat -<option> [-t] [-h<lines>] <vmid> [<interval> [<count>]]**  
参数解释：   
  
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

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)

实例使用1：   
  
[root@localhost bin]# **jstat -gcutil 25444**  
  
  S0     S1     E      O      P     YGC     YGCT    FGC    FGCT     GCT   
  
11.63   0.00   56.46  66.92  98.49 162    0.248    6      0.331    0.579   
  
  
  
实例使用2：(25444是java的进程号，ps -ef | grep java)   
  
[root@localhost bin]# jstat -gcutil 25444 1000 5   
  
  S0     S1     E      O      P     YGC     YGCT    FGC    FGCT     GCT   
  
73.54   0.00  99.04  67.52  98.49    166    0.252     6    0.331    0.583   
  
73.54   0.00  99.04  67.52  98.49    166    0.252     6    0.331    0.583   
  
73.54   0.00  99.04  67.52  98.49    166    0.252     6    0.331    0.583   
  
73.54   0.00  99.04  67.52  98.49    166    0.252     6    0.331    0.583   
  
73.54   0.00  99.04  67.52  98.49    166    0.252     6    0.331    0.583   
  
我们可以看到，5次young gc之后，垃圾内存被从Eden space区(E)放入了Old space区(O)，并引起了百分比的变化，导致Survivor space使用的百分比从73.54%(S0)降到0%(S1)。有效释放了内存空间。绿框中，我们可以看到，一次full gc之后，Old space区(O)的内存被回收，从99.05%降到67.52%。   
  
图中同时打印了young gc和full gc的总次数、总耗时。而，每次young gc消耗的时间，可以用相间隔的两行YGCT相减得到。每次full gc消耗的时间，可以用相隔的两行FGCT相减得到。例如红框中表示的第一行、第二行之间发生了1次young gc，消耗的时间为0.252-0.252＝0.0秒。   
  
常驻内存区(P)的使用率，始终停留在98.49%左右，说明常驻内存没有突变，比较正常。   
  
如果young gc和full gc能够正常发生，而且都能有效回收内存，常驻内存区变化不明显，则说明java内存释放情况正常，垃圾回收及时，java内存泄露的几率就会大大降低。但也不能说明一定没有内存泄露。   
  
GCT 是YGCT 和FGCT的时间总和。   
  
以上，介绍了jstat按百分比查看gc情况的功能。其实，它还有功能，例如加载类信息统计功能、内存池信息统计功能等，那些是以绝对值的形式打印出来的，比较少用，在此就不做介绍。

[root@localhost bin]# ps -ef | grep java   
  
root     25917     1  2 23:23 pts/2    00:00:05 /usr/local/jdk1.5/bin/java -Djava.endorsed.dirs=/usr/local/jakarta-tomcat-5.0.30/common/endorsed -classpath /usr/local/jdk1.5/lib/tools.jar:/usr/local/jakarta-tomcat-5.0.30/bin/bootstrap.jar:/usr/local/jakarta-tomcat-5.0.30/bin/commons-logging-api.jar -Dcatalina.base=/usr/local/jakarta-tomcat-5.0.30 -Dcatalina.home=/usr/local/jakarta-tomcat-5.0.30 -Djava.io.tmpdir=/usr/local/jakarta-tomcat-5.0.30/temp org.apache.catalina.startup.Bootstrap start   
  
jstat -class pid:显示加载class的数量，及所占空间等信息。   
  
实例使用3：   
  
[root@localhost bin]# jstat -class 25917   
  
Loaded  Bytes  Unloaded  Bytes     Time   
  
2629    2916.8       29   24.6     0.90   
  
  
  
jstat -compiler pid:显示VM实时编译的数量等信息。   
  
实例使用4：   
  
[root@localhost bin]# jstat -compiler 25917   
  
Compiled Failed Invalid   Time   FailedType FailedMethod   
  
     768      0       0   0.70            0   
  
  
  
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。   
  
  
  
[root@localhost bin]# jstat -gccapacity 25917   
  
NGCMN       640.0   
  
NGCMX       4992.0   
  
NGC         832.0   
  
S0C         64.0   
  
S1C         64.0   
  
EC          704.0   
  
OGCMN       1408.0   
  
OGCMX       60544.0   
  
OGC         9504.0   
  
OC          9504.0                  OC是old内纯的占用量   
  
PGCMN       8192.0                  PGCMN显示的是最小perm的内存使用量   
  
PGCMX       65536.0                 PGCMX显示的是perm的内存最大使用量   
  
PGC         12800.0                 PGC是当前新生成的perm内存占用量   
  
PC          12800.0                 PC是但前perm内存占用量   
  
YGC         164   
  
FGC         6   
  
  
  
jstat -gcnew pid: new对象的信息   
  
[root@localhost bin]# jstat -gcnew 25917   
  
S0C    S1C    S0U    S1U   TT MTT  DSS      EC       EU     YGC     YGCT   
  
64.0   64.0   47.4   0.0   2  15   32.0    704.0    145.7    168    0.254   
  
  
  
jstat -gcnewcapacity pid: new对象的信息及其占用量   
  
[root@localhost bin]# jstat -gcnewcapacity 25917   
  
NGCMN  NGCMX   NGC   S0CMX  S0C   S1CMX  S1C   ECMX    EC      YGC   FGC   
  
640.0  4992.0  832.0 64.0   448.0 448.0  64.0   4096.0  704.0  168     6   
  
  
  
jstat -gcold pid: old对象的信息。   
  
[root@localhost bin]# jstat -gcold 25917   
  
   PC       PU        OC          OU       YGC    FGC    FGCT     GCT   
  
12800.0  12617.6     9504.0      6561.3   169     6    0.335    0.591   
  
  
  
jstat -gcoldcapacity pid:old对象的信息及其占用量。   
  
[root@localhost bin]# jstat -gcoldcapacity 25917   
  
OGCMN      OGCMX        OGC         OC       YGC   FGC    FGCT     GCT   
  
1408.0     60544.0      9504.0      9504.0   169     6    0.335    0.591   
  
  
  
jstat -gcpermcapacity pid: perm对象的信息及其占用量。   
  
[root@localhost bin]# jstat -gcpermcapacity 25917   
  
PGCMN      PGCMX       PGC         PC      YGC   FGC    FGCT     GCT   
  
8192.0    65536.0    12800.0    12800.0   169     6    0.335    0.591   
  
  
  
jstat -printcompilation pid:当前VM执行的信息。   
  
[root@localhost bin]# jstat -printcompilation -h3  25917 1000 5   
  
每1000毫秒打印一次，一共打印5次，还可以加上-h3每三行显示一下标题。   
  
Compiled  Size  Type Method   
  
     788     73    1 java/io/File <init>   
  
     788     73    1 java/io/File <init>   
  
     788     73    1 java/io/File <init>   
  
Compiled  Size  Type Method   
  
     788     73    1 java/io/File <init>   
  
     788     73    1 java/io/File <init>   
  
  
jstat监控gc情况   
文章分类:Java编程   
  
性能测试过程中，我们该如何监控java虚拟机内存的使用情况，用以判断JVM是否存在内存问题呢？如何判断JVM垃圾回收是否正常？一般的top指令基本上满足不了这样的需求，因为它主要监控的是总体的系统资源，很难定位到java应用程序。   
在项目实践过程中，我们探索和使用了一款新工具－－Jstat。   
    先秀一下。Jstat是JDK自带的一个轻量级小工具。全称“Java Virtual Machine statistics monitoring tool”，它位于java的bin目录下，主要利用JVM内建的指令对Java应用程序的资源和性能进行实时的命令行的监控，包括了对Heap size和垃圾回收状况的监控。可见，Jstat是轻量级的、专门针对JVM的工具，非常适用。   
那，该怎么用呢？   
    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   
    下面给出一个实际的例子：   
  
注：由于JVM内存设置较大，图中百分比变化不太明显 

 图中参数含义如下：   
  
    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)   
  
    上图的示例，红框中，我们可以看到，5次young gc之后，垃圾内存被从Eden space区(E)放入了Old space区(O)，并引起了百分比的变化，导致Survivor space使用的百分比从19.69%(S0)降到10.34%(S1)。有效释放了内存空间。绿框中，我们可以看到，一次full gc之后，Old space区(O)的内存被回收，从36.81%降到35.01%。   
  
    图中同时打印了young gc和full gc的总次数、总耗时。而，每次young gc消耗的时间，可以用相间隔的两行YGCT相减得到。每次full gc消耗的时间，可以用相隔的两行FGCT相减得到。例如红框中表示的第一行、第二行之间发生了1次young gc，消耗的时间为52.281-52.252＝0.029秒。   
  
    常驻内存区(P)的使用率，始终停留在37.6%左右，说明常驻内存没有突变，比较正常。   
  
如果young gc和full gc能够正常发生，而且都能有效回收内存，常驻内存区变化不明显，则说明java内存释放情况正常，垃圾回收及时，java内存泄露的几率就会大大降低。但也不能说明一定没有内存泄露。   
  
  
  
    以上，介绍了Jstat按百分比查看gc情况的功能。其实，它还有其它功能，例如加载类信息统计功能、内存池信息统计功能等，那些是以绝对值的形式打印出来的，比较少用，在此就不做介绍。   
  
    
  
    为了更全面的监控JVM内存使用情况，我们需要引入更强大的工具来进一步分析–JConsole。敬请关注。