**jstat命令(Java Virtual Machine Statistics Monitoring Tool)**

<https://blog.csdn.net/fenglibing/article/details/6411951>

[JDK内置工具使用](http://blog.csdn.net/fenglibing/archive/2011/05/11/6411999.aspx)

[一、javah命令(C Header and Stub File Generator)](http://blog.csdn.net/fenglibing/archive/2011/05/11/6411924.aspx)

[二、jps命令(Java Virtual Machine Process Status Tool)](http://blog.csdn.net/fenglibing/archive/2011/05/11/6411932.aspx)

[三、jstack命令(Java Stack Trace)](http://blog.csdn.net/fenglibing/archive/2011/05/11/6411940.aspx)

[四、jstat命令(Java Virtual Machine Statistics Monitoring Tool)](http://blog.csdn.net/fenglibing/archive/2011/05/11/6411951.aspx)

[五、jmap命令(Java Memory Map)](http://blog.csdn.net/fenglibing/archive/2011/05/11/6411953.aspx)

[六、jinfo命令(Java Configuration Info)](http://blog.csdn.net/fenglibing/archive/2011/05/11/6411958.aspx)

[七、jconsole命令(Java Monitoring and Management Console)](http://blog.csdn.net/fenglibing/archive/2011/05/11/6411965.aspx)

[八、jvisualvm命令(Java Virtual Machine Monitoring, Troubleshooting, and Profiling Tool)](http://blog.csdn.net/fenglibing/archive/2011/05/11/6411969.aspx)

[九、jhat命令(Java Heap Analyse Tool)](http://blog.csdn.net/fenglibing/archive/2011/05/11/6411971.aspx)

[十、Jdb命令(The Java Debugger)](http://blog.csdn.net/fenglibing/archive/2011/05/11/6411974.aspx)

[十一、Jstatd命令(Java Statistics Monitoring Daemon)](http://blog.csdn.net/fenglibing/article/details/17323515)

## 1、介绍

Jstat用于监控基于HotSpot的JVM，对其堆的使用情况进行实时的命令行的统计，使用jstat我们可以对指定的JVM做如下监控：

- 类的加载及卸载情况

- 查看新生代、老生代及持久代的容量及使用情况

- 查看新生代、老生代及持久代的垃圾收集情况，包括垃圾回收的次数及垃圾回收所占用的时间

- 查看新生代中Eden区及Survior区中容量及分配情况等

jstat工具特别强大，它有众多的可选项，通过提供多种不同的监控维度，使我们可以从不同的维度来了解到当前JVM堆的使用情况。详细查看堆内各个部分的使用量，使用的时候必须加上待统计的Java进程号，可选的不同维度参数以及可选的统计频率参数。

它主要是用来显示GC及PermGen相关的信息，如果对GC不怎么了解，先看这篇文章：<http://blog.csdn.net/fenglibing/archive/2011/04/13/6321453.aspx>，否则其中即使你会使用jstat这个命令，你也看不懂它的输出。

## 2、语法

     jstat [ generalOption | outputOptions vmid [interval[s|ms] [count]] ]

**generalOption** - 单个的常用的命令行选项，如-help, -options, 或 -version。

**outputOptions -**一个或多个输出选项，由单个的statOption选项组成，可以和-t, -h, and -J等选项配合使用。

**statOption**：

    根据jstat统计的维度不同，可以使用如下表中的选项进行不同维度的统计，不同的操作系统支持的选项可能会不一样，可以通过-options选项，查看不同操作系统所支持选项，如：

|  |  |
| --- | --- |
| **Option** | **Displays...** |
| [class](http://docs.oracle.com/javase/1.5.0/docs/tooldocs/share/jstat.html#class_option) | 用于查看类加载情况的统计 |
| [compiler](http://docs.oracle.com/javase/1.5.0/docs/tooldocs/share/jstat.html#compiler_option) | 用于查看HotSpot中即时编译器编译情况的统计 |
| [gc](http://docs.oracle.com/javase/1.5.0/docs/tooldocs/share/jstat.html#gc_option) | 用于查看JVM中堆的垃圾收集情况的统计 |
| [gccapacity](http://docs.oracle.com/javase/1.5.0/docs/tooldocs/share/jstat.html#gccapacity_option) | 用于查看新生代、老生代及持久代的存储容量情况 |
| [gccause](http://docs.oracle.com/javase/1.5.0/docs/tooldocs/share/jstat.html#gccause_option) | 用于查看垃圾收集的统计情况（这个和-gcutil选项一样），如果有发生垃圾收集，它还会显示最后一次及当前正在发生垃圾收集的原因。 |
| [gcnew](http://docs.oracle.com/javase/1.5.0/docs/tooldocs/share/jstat.html#gcnew_option) | 用于查看新生代垃圾收集的情况 |
| [gcnewcapacity](http://docs.oracle.com/javase/1.5.0/docs/tooldocs/share/jstat.html#gcnewcapacity_option) | 用于查看新生代的存储容量情况 |
| [gcold](http://docs.oracle.com/javase/1.5.0/docs/tooldocs/share/jstat.html#gcold_option) | 用于查看老生代及持久代发生GC的情况 |
| [gcoldcapacity](http://docs.oracle.com/javase/1.5.0/docs/tooldocs/share/jstat.html#gcoldcapacity_option) | 用于查看老生代的容量 |
| [gcpermcapacity](http://docs.oracle.com/javase/1.5.0/docs/tooldocs/share/jstat.html#gcpermcapacity_option) | 用于查看持久代的容量 |
| [gcutil](http://docs.oracle.com/javase/1.5.0/docs/tooldocs/share/jstat.html#gcutil_option) | 用于查看新生代、老生代及持代垃圾收集的情况 |
| [printcompilation](http://docs.oracle.com/javase/1.5.0/docs/tooldocs/share/jstat.html#printcompilation_option) | HotSpot编译方法的统计 |

**-h** n

        用于指定每隔几行就输出列头，如果不指定，默认是只在第一行出现列头。

**-J**javaOption

        用于将给定的javaOption传给java应用程序加载器，例如，“-J-Xms48m”将把启动内存设置为48M。如果想查看可以传递哪些选项到应用程序加载器中，可以相看如下的文档：

        Linux and Solaris：[http://docs.oracle.com/javase/1.5.0/docs/tooldocs/solaris/java.html](http://docs.oracle.com/javase/1.5.0/docs/tooldocs/solaris/java.html" \t "_blank)

        Windows： <http://docs.oracle.com/javase/1.5.0/docs/tooldocs/windows/java.html>

**-t**n

        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（注：在IBM JDK5中是没有这个选项的）。

**vmid**- VM的进程号，即当前运行的java进程号。

    还有两个关于显示频率的选项：

**interval**–间隔时间，单位可以是秒或者毫秒，通过指定s或ms确定，默认单位为毫秒。

**count**-打印次数，如果缺省则打印无数次。

**3、不同的统计维度（statOption）及输出说明**

### -class

|  |  |
| --- | --- |
| 类加载情况的统计 | |
| **列名** | 说明 |
| Loaded | 加载了的类的数量 |
| Bytes | 加载了的类的大小，单为Kb |
| Unloaded | 卸载了的类的数量 |
| Bytes | 卸载了的类的大小，单为Kb |
| Time | 花在类的加载及卸载的时间 |

### -compiler

|  |  |
| --- | --- |
| HotSpot中即时编译器编译情况的统计 | |
| 列名 | 说明 |
| Compiled | 编译任务执行的次数 |
| Failed | 编译任务执行失败的次数 |
| Invalid | 编译任务非法执行的次数 |
| Time | 执行编译花费的时间 |
| FailedType | 最后一次编译失败的编译类型 |
| FailedMethod | 最后一次编译失败的类名及方法名 |

### -gc

|  |  |
| --- | --- |
| JVM中堆的垃圾收集情况的统计 | |
| 列名 | 说明 |
| S0C | 新生代中Survivor space中S0当前容量的大小（KB） |
| S1C | 新生代中Survivor space中S1当前容量的大小（KB） |
| S0U | 新生代中Survivor space中S0容量使用的大小（KB） |
| S1U | 新生代中Survivor space中S1容量使用的大小（KB） |
| EC | Eden space当前容量的大小（KB） |
| EU | Eden space容量使用的大小（KB） |
| OC | Old space当前容量的大小（KB） |
| OU | Old space使用容量的大小（KB） |
| PC | Permanent space当前容量的大小（KB） |
| PU | Permanent space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 |
| YGCT | 从应用程序启动到采样时 Young GC 所用的时间(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(秒) |
| GCT | T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 |

### -gccapacity

|  |  |
| --- | --- |
| 新生代、老生代及持久代的存储容量情况 | |
| 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） |
| 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） |
| EC | Eden space当前容量的大小（KB） |
| 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 | 当前老生代的空间容量大小（KB） |
| 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发生 Full GC 的次数 |

### -gccause

     这个选项用于查看垃圾收集的统计情况（这个和-gcutil选项一样），如果有发生垃圾收集，它还会显示最后一次及当前正在发生垃圾收集的原因，它比**-gcutil**会多出最后一次垃圾收集原因以及当前正在发生的垃圾收集的原因。

|  |  |
| --- | --- |
| 用于查看垃圾收集的统计情况，包括最近发生垃圾的原因 | |
| 列名 | 说明 |
| LGCC | 最后一次垃圾收集的原因，可能为“unknown GCCause”、“System.gc()”等 |
| GCC | 当前垃圾收集的原因 |

### -gcnew

|  |  |
| --- | --- |
| 新生代垃圾收集的情况 | |
| 列名 | 说明 |
| 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，要了解这个参数，我们需要了解一点Java内存对象的结构，在Sun JVM中，（除了数组之外的）对象都有两个机器字（words）的头部。第一个字中包含这个对象的标示哈希码以及其他一些类似锁状态和等标识信息，第二个字中包含一个指向对象的类的引用，其中第二个字节就会被垃圾收集算法使用到。 在新生代中做垃圾收集的时候，每次复制一个对象后，将增加这个对象的收集计数，当一个对象在新生代中被复制了一定次数后，该算法即判定该对象是长周期的对象，把他移动到老生代，这个阈值叫着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).可以参与这里：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(单位秒) |

### -gcnewcapacity

|  |  |
| --- | --- |
| 新生代的存储容量情况 | |
| 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 |

### -gcold

|  |  |
| --- | --- |
| 老生代及持久代发生GC的情况 | |
| 列名 | 说明 |
| 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 |

### -gcoldcapacity

|  |  |
| --- | --- |
| 老生代的存储容量情况 | |
| 列名 | 说明 |
| 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 |

### -gcpermcapacity

    从应用程序启动到采样时发生 Full GC 的次数

|  |  |
| --- | --- |
| 持久代的存储容量情况 | |
| 列名 | 说明 |
| 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 |  |
| 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 |

### -gcutil

|  |  |
| --- | --- |
| 新生代、老生代及持代垃圾收集的情况 | |
| 列名 | 说明 |
| 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)，它的值等于YGC+FGC |

### -printcompilation

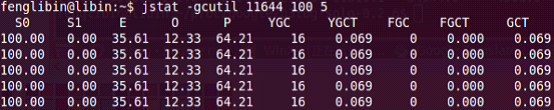
|  |  |
| --- | --- |
| HotSpot编译方法的统计 | |
| 列名 | **说明** |
| Compiled | 编译任务执行的次数 |
| Size | 方法的字节码所占的字节数 |
| Type | 编译类型 |
| Method | 指定确定被编译方法的类名及方法名，类名中使名“/”而不是“.”做为命名分隔符，方法名是被指定的类中的方法，这两个字段的格式是由HotSpot中的“-**XX:+PrintComplation**”选项确定的。 |

## 4、使用示例

示例1）、

http://hi.csdn.net/attachment/201105/11/0_13050966388fce.gif

示例2）：



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64.21%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。

示例3）、

http://hi.csdn.net/attachment/201105/11/0_13050966471wL6.gif

    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。

示例4）、jstat -class pid:显示加载class的数量，及所占空间等信息。

http://hi.csdn.net/attachment/201105/11/0_13050966533yVc.gif

 示例5）、jstat -compiler pid:显示VM实时编译的数量等信息。

http://hi.csdn.net/attachment/201105/11/0_1305096664XzFs.gif

示例6）、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>

    下面是一个执行示例：

C:\Documents and Settings\Administrator>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

    示例就不一一例举，有兴趣的可以自己尝试尝试。

[**Oracle上关于Jstat的使用说明：http://docs.oracle.com/javase/1.5.0/docs/tooldocs/share/jstat.html**](http://docs.oracle.com/javase/1.5.0/docs/tooldocs/share/jstat.html)