JVM性能监控工具

JDK本身提供了很多方便的JVM性能监控工具，除了集成式的VisualVM和jConsole外，还有jps、jstack、jmap、jhat、jstat等小巧的工具。

# 1.jps，Java Virtual Machine Process Status Tool

主要用来输出JVM中运行的进程状态信息，语法格式：

*jps [options] [hostid]*

如果不指定hostid，则默认为当前的主机或者服务器

命令行参数:

*-q 不输出类名、Jar名和传入main方法的参数*

*-m 输出传入main方法的参数*

*-l 输出main类或Jar的全限名*

*-v 输出传入JVM的参数*

如下所示：

*[root@sgserver002 ~]# jps -m -l*

*10700 org.apache.hadoop.hbase.thrift2.ThriftServer start*

*31490 org.apache.hadoop.hbase.master.HMaster start*

*4950 org.apache.hadoop.hdfs.server.datanode.DataNode*

*28211 org.apache.hadoop.yarn.server.nodemanager.NodeManager*

*31358 org.apache.hadoop.hdfs.tools.DFSZKFailoverController*

*7663 org.apache.hadoop.hdfs.server.namenode.NameNode*

*15095 org.apache.hadoop.hbase.regionserver.HRegionServer start*

# 2.jstack

jstack主要用来查看某个java进程内的线程堆栈信息，语法格式如下：

*jstack [option] pid*

*jstack [option] executable core*

*jstack [option] [server-id@]remote-hostname-or-ip*

命令行参数选项说明如下：

*-l long listings，会打印出额外的锁信息，在发生死锁时可以用jstack -l pid来观察锁持有情况*

*-m mixed mode，不仅会输出Java堆栈信息，还会输出C/C++堆栈信息（比如Native方法）*

Jstack可以定位到线程堆栈，根据堆栈信息可以定位到具体代码，所有在JVM性能调试中使用很多。

下面的实例找出java进程中最耗费cpu的java线程并定位堆栈信息：

1. 找出java进程ID，可以通过jps或者ps命令，例如找到NameNode进程

*[root@sgserver002 ~]# ps -ef | grep NameNode | grep -v grep*

*hdfs 7663 1 0 Feb15 ? 00:14:10 /usr/jdk64/jdk1.7.0\_67/bin/java*

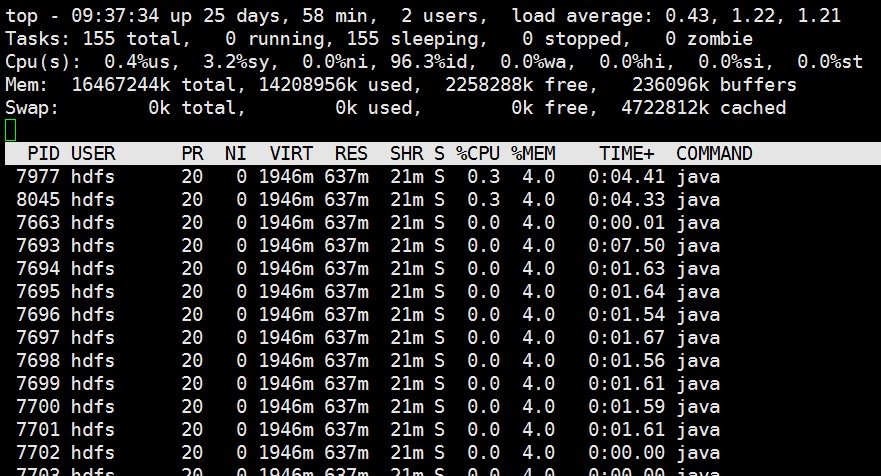
*……*

*org.apache.hadoop.hdfs.server.namenode.NameNode*

得到进程ID为7663

1. 找到该进程中最耗费CPU的线程，可以使用ps -Lfp pid或者ps -mp pid -o THREAD, tid, time或者top -Hp pid

*[root@sgserver002 ~]# top -Hp 7663*



TIME+列是各个Java进程耗费的CPU时间，CPU最长时间的线程ID为7950的线程，将7950转换为16进程

*print “%x\n” 7950 >= 1f0e*

7693的十六进制值为1f0e

1. 使用jstack输出7693的堆栈信息，根据线程ID的十六进制值grep，如下

*$ jstack 7663|grep 1f0e*

*"Socket Reader #1 for port 8020" prio=10 tid=0x00007f11a9119800 nid=0x1f0e runnable [0x00007f118e694000]*

看一看到CPU消耗在runnable这个类中，然后根据这个找到对应的代码

# 3.jmap和jhat

jmap用来查看对内存使用状况，一般使用jhat使用

*jmap [option] pid*

*jmap [option] executable core*

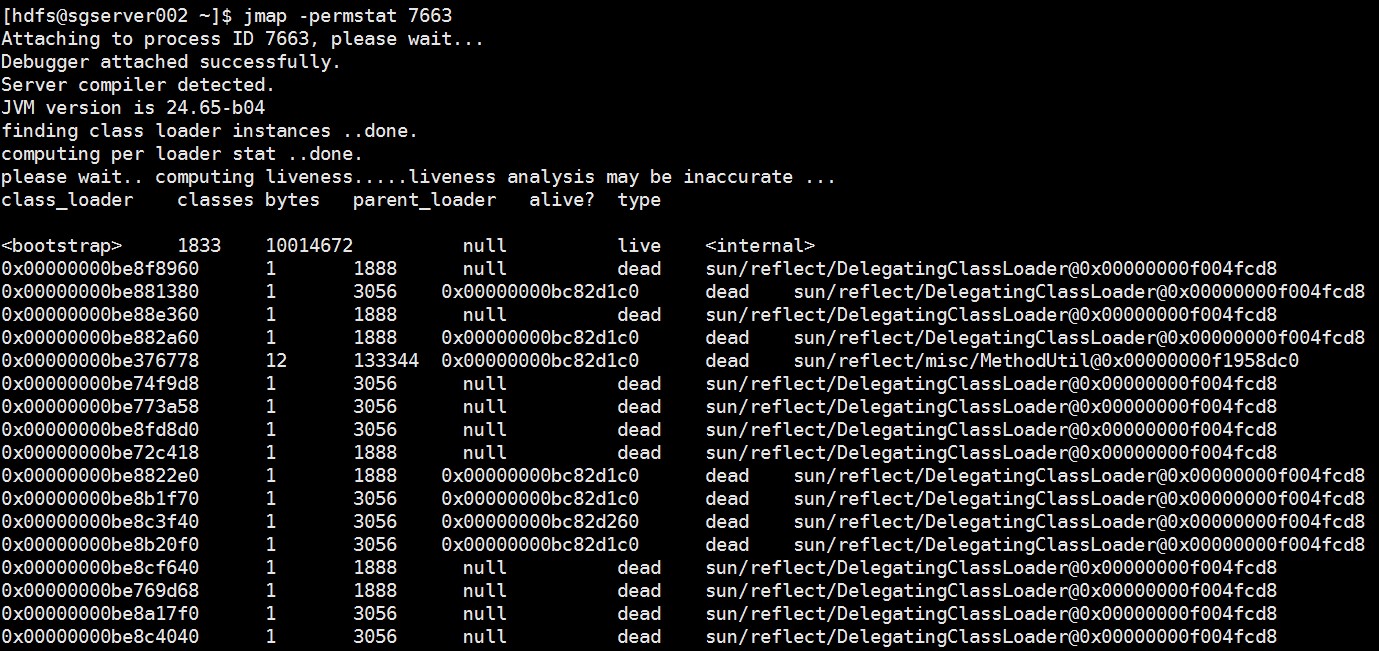
*jmap [option] [server-id@]remote-hostname-or-ip*

如果运行在64位JVM上，可能需要指定-J-d64命令项参数

介绍下面的使用方法：

*jmap –permstat pid*

打印进程的类加载器和类加载器的持久化对象信息，输出：类加载器名称、对象是否存活、对象地址、父类加载器、已加载的类大小等信息，如下图所示：



使用jmap –heap pid查看进程对内存使用情况，包括GC算法、堆配置参数和各代中堆内存使用情况，如下面例子所示：

*$ jmap -heap 7663*

*Attaching to process ID 7663, please wait...*

*Debugger attached successfully.*

*Server compiler detected.*

*JVM version is 24.65-b04*

*using parallel threads in the new generation.*

*using thread-local object allocation.*

*Concurrent Mark-Sweep GC*

*Heap Configuration:*

*MinHeapFreeRatio = 40*

*MaxHeapFreeRatio = 70*

*MaxHeapSize = 1073741824 (1024.0MB)*

*NewSize = 209715200 (200.0MB)*

*MaxNewSize = 209715200 (200.0MB)*

*OldSize = 5439488 (5.1875MB)*

*NewRatio = 2*

*SurvivorRatio = 8*

*PermSize = 134217728 (128.0MB)*

*MaxPermSize = 268435456 (256.0MB)*

*G1HeapRegionSize = 0 (0.0MB)*

*Heap Usage:*

*New Generation (Eden + 1 Survivor Space):*

*capacity = 188743680 (180.0MB)*

*used = 141928880 (135.3539276123047MB)*

*free = 46814800 (44.64607238769531MB)*

*75.19662645128038% used*

*Eden Space:*

*capacity = 167772160 (160.0MB)*

*used = 120957360 (115.35392761230469MB)*

*free = 46814800 (44.64607238769531MB)*

*72.09620475769043% used*

*From Space:*

*capacity = 20971520 (20.0MB)*

*used = 20971520 (20.0MB)*

*free = 0 (0.0MB)*

*100.0% used*

*To Space:*

*capacity = 20971520 (20.0MB)*

*used = 0 (0.0MB)*

*free = 20971520 (20.0MB)*

*0.0% used*

*concurrent mark-sweep generation:*

*capacity = 864026624 (824.0MB)*

*used = 280129104 (267.1519317626953MB)*

*free = 583897520 (556.8480682373047MB)*

*32.421350942074675% used*

*Perm Generation:*

*capacity = 134217728 (128.0MB)*

*used = 43388384 (41.378387451171875MB)*

*free = 90829344 (86.62161254882812MB)*

*32.32686519622803% used*

*14932 interned Strings occupying 1426072 bytes.*

使用jmap –histo[:live] pid查看对内存中的对象数目、大小统计直方图，如果带上live则只统计活对象，如下：

*[hdfs@sgserver002 ~]$ jmap -histo:live 7663|more*

*num #instances #bytes class name*

*----------------------------------------------*

*1: 4 13762624 [Lorg.apache.hadoop.util.LightWeightGSet$LinkedElement;*

*2: 83090 10778832 <constMethodKlass>*

*3: 83090 10646128 <methodKlass>*

*4: 6137 7871848 <constantPoolKlass>*

*5: 6137 4798312 <instanceKlassKlass>*

*6: 5177 4505600 <constantPoolCacheKlass>*

*7: 22628 4468128 [B*

*8: 31238 2868344 [C*

*9: 3360 1612456 <methodDataKlass>*

*10: 6634 804496 java.lang.Class*

*11: 29682 712368 java.lang.String*

*12: 10429 692728 [S*

*13: 10646 607656 [[I*

*14: 149 473696 [Ljava.nio.ByteBuffer;*

*15: 4949 467344 [Ljava.lang.Object;*

*16: 12070 386240 java.util.HashMap$Entry*

*17: 9800 313600 java.util.concurrent.ConcurrentHashMap$HashEntry*

*18: 480 261120 <objArrayKlassKlass>*

*19: 1406 214792 [Ljava.util.HashMap$Entry;*

*20: 6619 211808 java.util.Hashtable$Entry*

*21: 10639 170224 java.lang.Object*

*22: 2164 155808 org.apache.hadoop.hdfs.server.namenode.INodeFile*

*23: 441 144312 [Ljava.util.concurrent.ConcurrentHashMap$HashEntry;*

*24: 1663 133040 java.lang.reflect.Method*

*25: 3173 126920 java.util.LinkedHashMap$Entry*

*26: 617 121624 [I*

*27: 2099 117544 org.apache.hadoop.hdfs.server.blockmanagement.BlockInfo*

*28: 324 71520 [Ljava.util.Hashtable$Entry;*

*29: 1768 70720 java.util.TreeMap$Entry*

*30: 1268 60864 java.util.HashMap*

*31: 1822 58304 java.lang.ThreadLocal$ThreadLocalMap$Entry*

还有一种很常用的情况，用jmap把进程内存使用情况dump到文件中，再用jhat分析查看，命令格式如下：

*jmap -dump:format=b,file=dump.dat 7663*

dump出来的文件可以用MAT、VisualVM等工具查看，这里用jhat查看

*$ jhat -port 9998 dump.dat*

*Reading from dump.dat...*

*Dump file created Thu Feb 16 10:56:19 CST 2017*

*Snapshot read, resolving...*

*Resolving 846236 objects...*

*Chasing references, expect 169 dots.........................................................................................................................................................................*

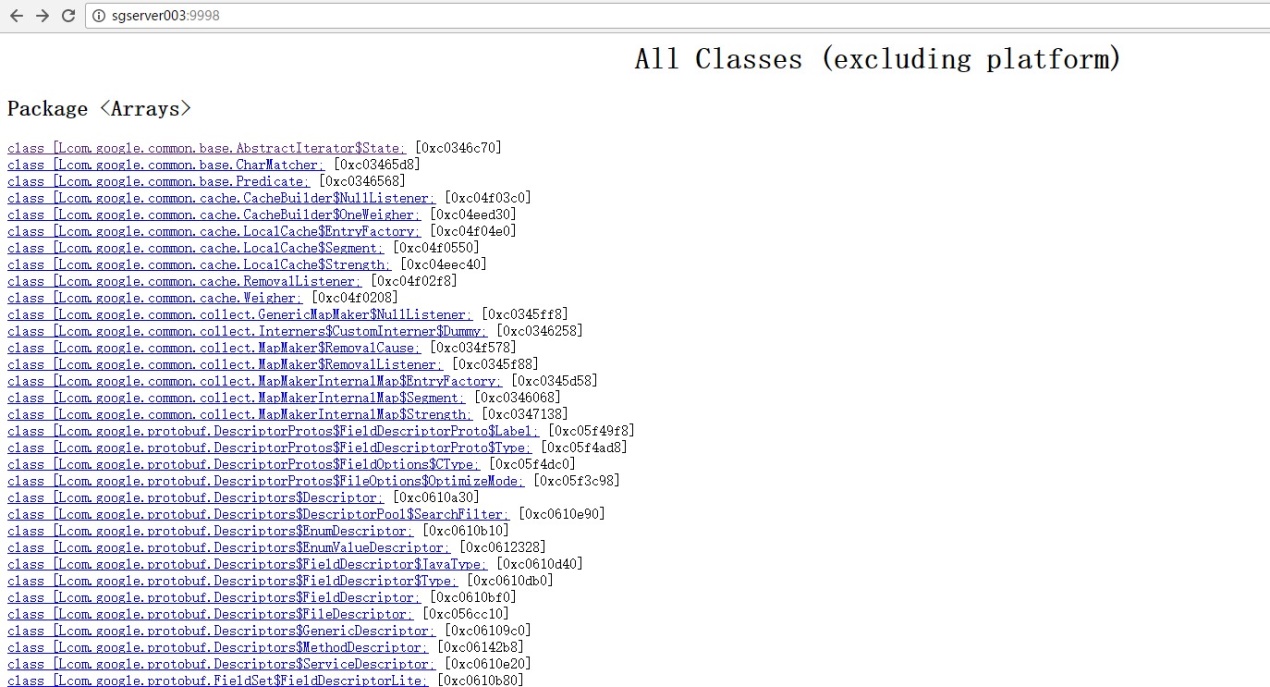
*Eliminating duplicate references.........................................................................................................................................................................*

*Snapshot resolved.*

*Started HTTP server on port 9998*

*Server is ready.*

在浏览器中输出主机地址:9998，如下所示：



# 4.jstat（JVM统计监测工具）

语法格式如下：

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

vmid是虚拟机ID，在Linux/Unix系统上一般就是进程ID。Interval是采样时间间隔。Count是采样数目。比如下面输出的GC信息，采用时间即那个为250ms，采用数为4：

*$ jstat -gc 9839 250 4*

*S0C S1C S0U S1U EC EU OC OU PC PU YGC YGCT FGC FGCT GCT*

*20480.0 20480.0 13398.3 0.0 163840.0 80482.7 843776.0 57903.2 131072.0 48069.8 136 1.033 0 0.000 1.033*

*20480.0 20480.0 13398.3 0.0 163840.0 80482.7 843776.0 57903.2 131072.0 48069.8 136 1.033 0 0.000 1.033*

*20480.0 20480.0 13398.3 0.0 163840.0 80491.8 843776.0 57903.2 131072.0 48069.8 136 1.033 0 0.000 1.033*

*20480.0 20480.0 13398.3 0.0 163840.0 80491.8 843776.0 57903.2 131072.0 48069.8 136 1.033 0 0.000 1.033*

各列含义如下：

*S0C、S1C、S0U、S1U：Survivor 0/1区容量（Capacity）和使用量（Used）*

*EC、EU：Eden区容量和使用量*

*OC、OU：年老代容量和使用量*

*PC、PU：永久代容量和使用量*

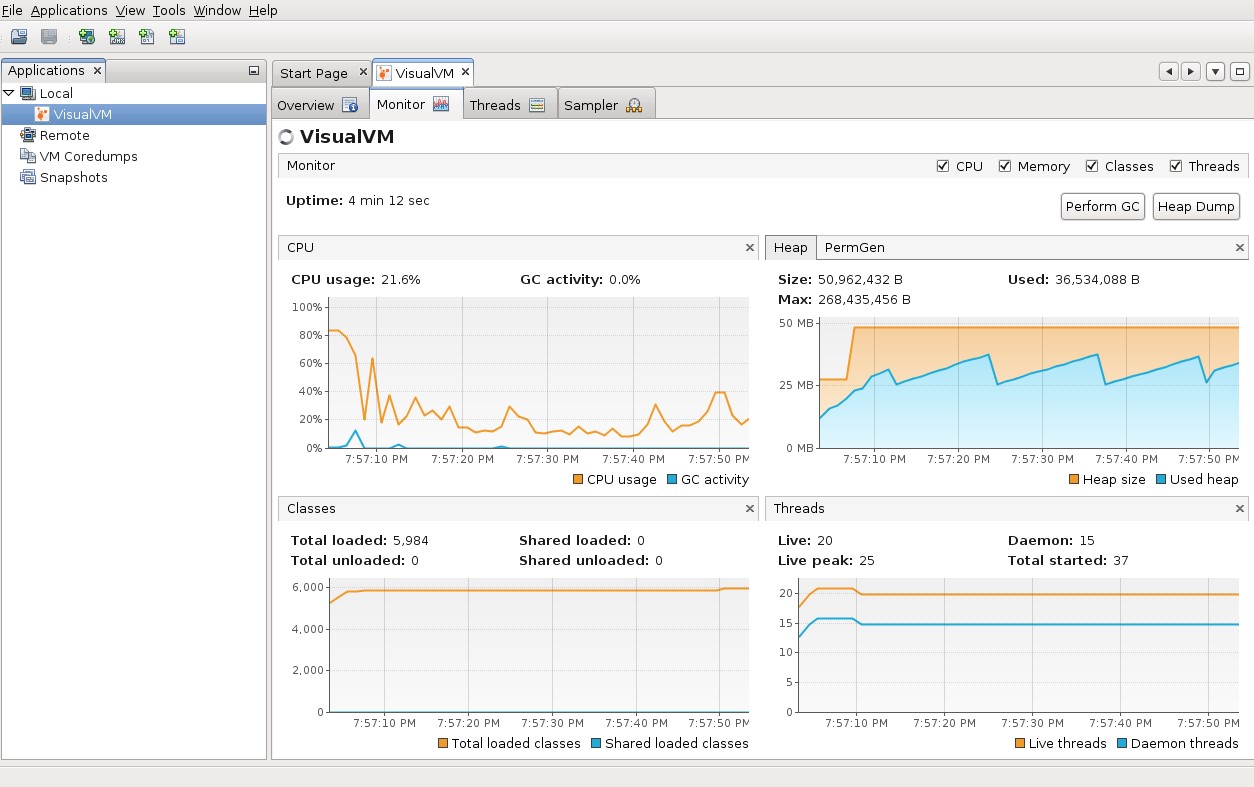
*YGC、YGT：年轻代GC次数和GC耗时*

*FGC、FGCT：Full GC次数和Full GC耗时*

*GCT：GC总耗时*

# 5.MAT及VirtualVM

MAT和VisualVM是Jvm调试的可视化工具，ORABLE JDK中自带VirtualVM，执行文件为bin/jvisualvm，打开页面如下所示：



参考链接：

https://www.ibm.com/developerworks/cn/java/j-lo-visualvm/

MAT是eclipse内存分析工具，不再介绍

参考链接：

http://chiyx.iteye.com/blog/1528782