简述

在jdk中提供了一些工具帮助开发人员解决一些问题。上一篇中提到的jps，jstack就是出自jdk。
jdk/bin目录下提供了很多exe文件其实都是jar文件的包装，真正的实现在jdk/lib/tools.jar中

jps–查看Java进程

jps命令可以查看所有的Java进程

jps存放在JAVA_HOME/bin/jps，使用时为了方便请将JAVA_HOME/bin/加入到Path

```
[no1@localhost ~]$ jps
9415 2.jar
9672 Jps
```

参数说明

```
-q 只显示pid
-m 输出传递给main 方法的参数
-l 输出应用程序main class的完整package名 或者 应用程序的jar文件完整路径名
-v 输出传递给JVM的参数
```

jstat–查看Java虚拟机运行时信息

可以查看Java虚拟机运行时信息，可以查看堆信息情况，GC情况

```
jstat -<option> [-t] [-h<lines>] <vmid> [<interval> [<count>]]

 参数解释:

Options – 选项
    -class              显示ClassLoad的相关信息;
    -compiler           显示JIT编译的相关信息;
    -gc                 显示和gc相关的堆信息;
    -gccapacity         显示各个代的容量以及使用情况;
     -gccause           显示垃圾回收的相关信息（通-gcutil）,同时显示最后一次或当前正在发生的垃圾回收的诱因;
     -gcnew             显示新生代信息;
     -gcnewcapacity     显示新生代大小和使用情况;
     -gcold             显示老年代和永久代的信息;
     -gcoldcapacity     显示老年代的大小;
     -gcpermcapacity    显示永久代的大小;
     -gcutil            显示垃圾收集信息;
     -printcompilation  输出JIT编译的方法信息;

```

```
-t      可以在打印的列加上Timestamp列，用于显示系统运行的时间

vmid    – VM的进程号，即当前运行的java进程号PID

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
```

class – 查看ClassLoad信息

查看pid为9415的ClassLoad相关信息

```
[no1@localhost ~]$ jstat -class 9415
Loaded  Bytes  Unloaded  Bytes     Time
   427   880.6        0     0.0       0.08
```

- Loaded 加载了类的总数量，后面的byte表示加载了类的总大小
- Unloaded 卸载了类的总数量，后面的byte表示卸载了类的总大小
- Time 加载和卸载类所花费的时间

compiler–显示JIT编译的相关信息

```
[no1@localhost ~]$ jstat -compiler 9415
Compiled Failed Invalid   Time   FailedType FailedMethod
     121      0       0     0.17          0
```

- Compiled 编译任务执行次数
- Failed 编译失败次数
- Invalid 编译不可用次数
- Time 编译总耗时
- FailedType 最后一次编译失败的类型
- FailedMethod 最后一次编译失败的方法和类名

gc – gc相关的堆信息

```
# 打印gc信息，每1000毫秒一次
[root@localhost ~]# jstat -gc 51720 1000
 S0C    S1C    S0U    S1U      EC       EU        OC         OU       MC     MU    CCSC   CCSU   YG
80384.0 79360.0  0.0   50373.1 162304.0 137885.8  432128.0   282726.5  39680.0 37602.9 5120.0 4708.
80384.0 79360.0  0.0   50373.1 162304.0 137885.8  432128.0   282726.5  39680.0 37602.9 5120.0 4708.
80384.0 79360.0  0.0   50373.1 162304.0 137885.8  432128.0   282726.5  39680.0 37602.9 5120.0 4708.
80384.0 79360.0  0.0   50373.1 162304.0 137885.8  432128.0   282726.5  39680.0 37602.9 5120.0 4708.
```

- S0C: Survivor0(幸存区0)大小（KB）
- S1C: Survivor1(幸存区1)1大小（KB）
- S0U: Survivor0(幸存区0)已使用大小（KB）
- S1U: Survivor1(幸存区1)已使用大小（KB）
- EC : Eden（伊甸区）大小（KB）
- EU : Eden（伊甸区）已使用大小（KB）
- OC ： 老年代大小（KB）
- OU : 老年代已使用大小（KB）
- PC : Perm永久代大小（KB）
- PU : Perm永久代已使用大小（KB）
- YGC：新生代GC个数
- YGCT：新生代GC的耗时（秒）
- FGC ： Full GC次数
- FGCT：Full GC耗时（秒）

- GCT： GC总耗时（秒）

gccapacity–显示各个代的容量以及使用情况

```
# 每1000毫秒一次
[root@localhost ~]# jstat -gccapacity 51720 1000
 NGCMN    NGCMX     NGC     S0C   S1C       EC      OGCMN      OGCMX       OGC         OC       MCMN
 20480.0 322048.0 322048.0 80384.0 79360.0 162304.0    40960.0   644608.0   432128.0   432128.0
 20480.0 322048.0 322048.0 80384.0 79360.0 162304.0    40960.0   644608.0   432128.0   432128.0
 20480.0 322048.0 322048.0 80384.0 79360.0 162304.0    40960.0   644608.0   432128.0   432128.0

```

- NGCMN：新生代最小值（KB）
- NGVMX：新生代最大值（KB）
- NGC：当前新生代大小（KB）
- S0C：当前新生代s0区大小
- S1C：当前新生代s1区大小
- EC：当前新生代Eden区大小
- OGCMN：老年代最小值（KB）
- OGCMX：老年代最大值（KB）
- OGC：当前老年代大小（KB）
- PGCMN：永久代最小值（KB）
- PGCMX：永久代最大值（KB）
- PGC：当前永久代大小（KB）

gccause – 最近一次GC的原因

```
[root@localhost ~]# jstat -gccause 51720
  S0     S1     E      O      M     CCS    YGC     YGCT    FGC    FGCT     GCT    LGCC
  0.00  63.47  84.96  65.43  94.77  91.97     11    0.184     4    0.369    0.552 Allocation Failure
```

- LGCC：上一次GC的原因，这里显示（Allocation Failure）的内存不够，需要回收在分配
- GCC： 当前GC的原因

gcnew – 显示新生代详细信息

```
[root@localhost ~]# jstat -gcnew 51720
 S0C    S1C    S0U    S1U   TT MTT  DSS      EC       EU     YGC     YGCT
80384.0 79360.0    0.0 50373.1  2  15 94208.0 162304.0 139509.1     11    0.184
```

- TT:新生代到老年代的年龄；
- MTT:新生代到老年代的最大年龄；
- DSS:所需的survivor的大小；

gcnewcapacity –输出新生代各个区的详细信息

```
[root@localhost ~]# jstat -gcnewcapacity 51720
  NGCMN       NGCMX        NGC       S0CMX     S0C     S1CMX     S1C       ECMX        EC      YGC   FG
   20480.0   322048.0   322048.0 107008.0  80384.0 107008.0  79360.0   321024.0   162304.0    11
```

- S0CMX:S0最大空间大小（KB）
- S1CMX:S1最大空间大小（KB）
- ECMX:Eden最大空间大小（KB）
- NGCMX:年轻代最大空间大小（KB）

gcutil–显示垃圾回收信息

```
[root@localhost ~]# jstat -gcutil 51720
  S0     S1     E      O      M     CCS    YGC     YGCT    FGC    FGCT     GCT
  0.00  63.47  85.96  65.43  94.77  91.97     11    0.184     4    0.369    0.552
```

- S0 — s0区已使用空间的百分比
- S1 — s1区已使用空间的百分比
- E — Eden区已使用空间的百分比
- O — 老年区已使用空间的百分比
- P — 永久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)

jinfo–查看虚拟机参数

jinfo可以用来查看正在运行的Java应用程序的扩展参数，甚至支持运行时修改部分参数

```
jinfo <option> pid
```

pid 即 Java进程ID

options参数可以是如下信息

- -flag : 打印指定Java虚拟机的参数值
- -flag [+|-] : 设置指定Java虚拟机参数的布尔值
- -flag =: 设置某一个参数的值value

例如查看是否打印GC信息，这里是输出结果 -XX:-PrintGC -表示不打印

```
[no1@localhost ~]$ jinfo -flag PrintGC 9415
-XX:-PrintGC
```

修改参数，开启打印GC
开启打印GC信息

```
[no1@localhost ~]$ jinfo -flag +PrintGC 10056
```

再来查看是否打印GC信息，发现已经开启

```
[no1@localhost ~]$ jinfo -flag PrintGC 10056
-XX:+PrintGC
```

jmap – java内存映像工具

jmap命令可以生成Java程序的堆dump文件，也可以查看堆对象实例的统计信息，查看ClassLoader的信息以及finalizer队列。

```
jmap [ option ] pid
```

- -dump:[live,]format=b,file= 使用hprof二进制形式,输出jvm的heap内容到文件=. live子选项是可选的，假如指定live选项,那么只输出活
- -finalizerinfo 打印正等候回收的对象的信息.
- -heap 打印heap的概要信息，GC使用的算法，heap的配置及wise heap的使用情况.
- -histo[:live] 打印每个class的实例数目,内存占用,类全名信息. VM的内部类名字开头会加上前缀”*”. 如果live子参数加上后,只统计活的
- -permstat 打印classload和jvm heap长久层的信息. 包含每个classloader的名字,活泼性,地址,父classloader和加载的class数量.另外,
  印出来

histo打印class类的实例

```
jmap -histo 10056 > c:/1.txt
```

输出：

```
num     #instances         #bytes  class name
----------------------------------------------
   1:          4983       6057848  [I
   2:         20929       2473080  <constMethodKlass>
.....
1932:             1             8  sun.java2d.pipe.AlphaColorPipe
1933:             1             8  sun.reflect.GeneratedMethodAccessor64
Total        230478      22043360
```


dump堆信息

```
[no1@localhost ~]$ jmap -dump:format=b,file=/home/no1/dump.hprof 10056
Dumping heap to /home/no1/dump.hprof ...
Heap dump file created
```

dump出的文件可以用 jhat（不推荐使用，功能有限），visual vm，MAT，IBM heapAnalyzer，Eclipse Memory Analyzer等打开查看

jstack–堆栈跟踪工具

jstack（stack Trace for Java） 命令用于生成虚拟机当前时刻的线程快照。线程快照即：当前虚拟机内每一个线程正在执行的方法的堆栈集都执行在哪个方法上了，状态是什么样子的（运行，等待等）

```
jstack -l pid
```

```
[no1@localhost ~]$ jstack -l 10056
2017-11-13 16:34:55
Full thread dump Java HotSpot(TM) 64-Bit Server VM (25.151-b12 mixed mode):

"Attach Listener" #12 daemon prio=9 os_prio=0 tid=0x00007f2414001000 nid=0x277b waiting on condition
   java.lang.Thread.State: RUNNABLE

   Locked ownable synchronizers:
    - None

"DestroyJavaVM" #11 prio=5 os_prio=0 tid=0x00007f2450008800 nid=0x2749 waiting on condition [0x0000
   java.lang.Thread.State: RUNNABLE

   Locked ownable synchronizers:
    - None

"Thread-2" #10 prio=5 os_prio=0 tid=0x00007f2450175000 nid=0x2756 waiting on condition [0x00007f243
   java.lang.Thread.State: TIMED_WAITING (sleeping)
    at java.lang.Thread.sleep(Native Method)
    at HoldIOMain$T2.run(HoldIOMain.java:34)
    at java.lang.Thread.run(Thread.java:748)

   Locked ownable synchronizers:
    - None

"Thread-1" #9 prio=5 os_prio=0 tid=0x00007f2450173800 nid=0x2755 waiting on condition [0x00007f243c
   java.lang.Thread.State: TIMED_WAITING (sleeping)
    at java.lang.Thread.sleep(Native Method)
    at HoldIOMain$T2.run(HoldIOMain.java:34)
    at java.lang.Thread.run(Thread.java:748)

```

```
   Locked ownable synchronizers:
    - None

"Thread-0" #8 prio=5 os_prio=0 tid=0x00007f2450171800 nid=0x2754 runnable [0x00007f243dadf000]
   java.lang.Thread.State: RUNNABLE
    at java.io.FileOutputStream.write(Native Method)
    at java.io.FileOutputStream.write(FileOutputStream.java:290)
    at HoldIOMain$T1.run(HoldIOMain.java:12)
    at java.lang.Thread.run(Thread.java:748)

   Locked ownable synchronizers:
    - None
```

jcmd

在JDK 1.7之后，新增了一个命令行工具jcmd。它是一个多功能工具，可以用来导出堆，查看java进程，导出线程信息，执行GC等。
jcmd具有jmap命令大部分功能，oracle官方网站推荐jcmd替代jmap

查看正在运行的虚拟机

```
# 参数说明: jcmd -l 可以查看所有的Java虚拟机，类似jps
[no1@localhost ~]$ jcmd -l
10056 /home/no1/soft/2.jar
10252 sun.tools.jcmd.JCmd -l
```

查看指定虚拟机

-l命令是列出所有Java虚拟机，针对每一个虚拟机可以通过help命令查询所支持的命令。

```
语法 jcmd pid help
```

```
[no1@localhost ~]$ jcmd 10056 help
10056:
The following commands are available:
JFR.stop
JFR.start
JFR.dump
JFR.check
VM.native_memory
VM.check_commercial_features
VM.unlock_commercial_features
ManagementAgent.stop
ManagementAgent.start_local
ManagementAgent.start
GC.rotate_log
Thread.print
GC.class_stats
GC.class_histogram
GC.heap_dump
GC.run_finalization
GC.run
VM.uptime
VM.flags
VM.system_properties
VM.command_line
```

```
VM.version
help
```

VM.uptime–查看虚拟机启动时间

```
[no1@localhost ~]$ jcmd 10056 VM.uptime
10056:
4140.026 s
```

Thread.print – 打印线程栈信息

```
[no1@localhost ~]$ jcmd 10056 Thread.print
10056:
2017-11-13 17:00:19
Full thread dump Java HotSpot(TM) 64-Bit Server VM (25.151-b12 mixed mode):

"Attach Listener" #12 daemon prio=9 os_prio=0 tid=0x00007f2414001000 nid=0x277b waiting on condition [0x0000000000000000]
   java.lang.Thread.State: RUNNABLE

"DestroyJavaVM" #11 prio=5 os_prio=0 tid=0x00007f2450008800 nid=0x2749 waiting on condition [0x0000000000000000]
   java.lang.Thread.State: RUNNABLE

"Thread-2" #10 prio=5 os_prio=0 tid=0x00007f2450175000 nid=0x2756 waiting on condition [0x00007f243d8dd000]
   java.lang.Thread.State: TIMED_WAITING (sleeping)
        at java.lang.Thread.sleep(Native Method)
        at HoldIOMain$T2.run(HoldIOMain.java:34)
        at java.lang.Thread.run(Thread.java:748)

"Thread-1" #9 prio=5 os_prio=0 tid=0x00007f2450173800 nid=0x2755 waiting on condition [0x00007f243d9de000]
   java.lang.Thread.State: TIMED_WAITING (sleeping)
        at java.lang.Thread.sleep(Native Method)
        at HoldIOMain$T2.run(HoldIOMain.java:34)
        at java.lang.Thread.run(Thread.java:748)

"Thread-0" #8 prio=5 os_prio=0 tid=0x00007f2450171800 nid=0x2754 runnable [0x00007f243dadf000]
   java.lang.Thread.State: RUNNABLE
        at java.io.FileInputStream.read0(Native Method)
        at java.io.FileInputStream.read(FileInputStream.java:207)
        at HoldIOMain$T1.run(HoldIOMain.java:18)
        at java.lang.Thread.run(Thread.java:748)
```


GC.heap_dump–导出堆信息

```
[no1@localhost ~]$ jcmd 10056 GC.heap_dump /home/no1/gcdump.dump
10056:
Heap dump file created
```

VM.system_properties – 获取系统properties

Jconsole–Java监视与管理控制台

Jconsole是jdk自带的图形化工具，在%JAVA_HOME%/bin 目录下jconsole.exe打开即可启动

可以连接本地和远程连接两种方式

VisualVM–多合一故障处理工具

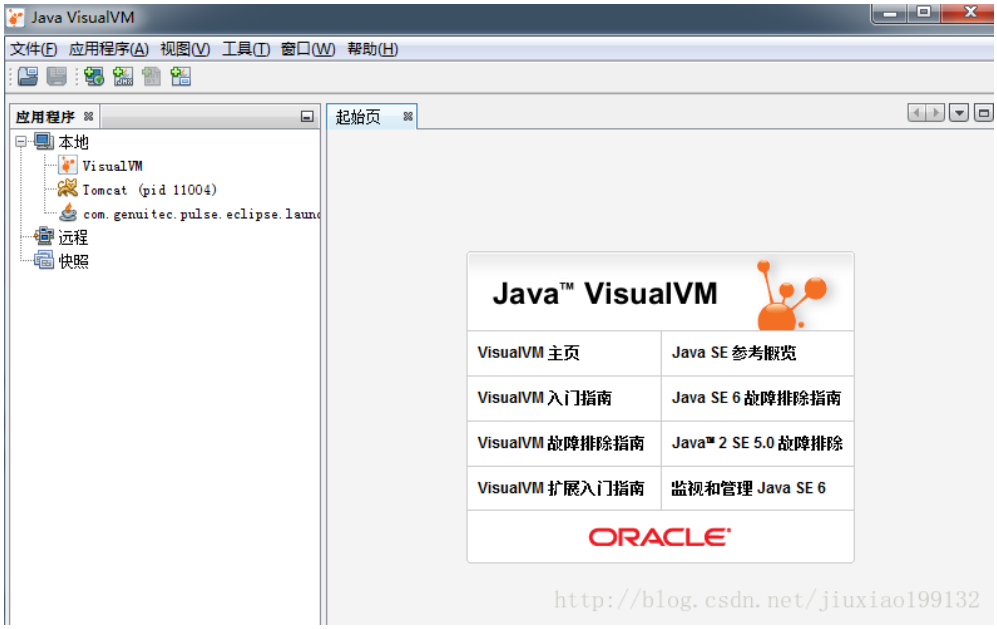
VisualVM是一个多功能合一的故障诊断，性能监控的可视化工具。集成了多种性能统计工具的功能，在jdk update7版本以后跟随jdk一起发
载http://visualvm.java.net/

VisualVm可以做到：

- 显示虚拟机进程的配置和环境信息（jps，jinfo）
- 监视应用程序的GC，CPU，堆，方法区，线程信息（jstack，jstat）
- dump及分析堆转储快照（jmap，jhat）
- 方法级的程序性能分析，找出被调用最多，运行时间最长的方法。
- 很多其他插件（plugins）

启动

在%JAVA_HOME%/bin目录下jvisualvm.exe 双击即可打开



左侧本地会列出机器上的所有Java程序：图中有tomcat，eclipse，visual vm。
远程 可以连接到远程机器的Java虚拟机进行检测。需要通过jstatd。
快照可以浏览导出的本地文件进行分析。

远程监控（jconsole 和 visual vm）

两个工具方式都是一样的

jar包(不需要密码的方式)

1. 启动jar包，参数中加上远程连接信息（ps：特别注意有一个host的参数，好多网站没提）

```
[root@localhost home]# java -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.port=8011
```

2. 通过jconsole连接就行了，输入对应的ip:port即可

tomcat（需要密码的方式）

- 下载远程监控需要的jar

```
https://mirror.bjtu.edu.cn/apache/tomcat/tomcat-8/v8.5.45/bin/extras/catalina-jmx-remote.jar
将jar包放在tomcat 的 lib目录下
```

- 修改tomcat7的$CATALINA_BASE/conf/server.xml,在<Server port="8005" shutdown="SHUTDOWN"> 下加入监听器：

```
<Listener className="org.apache.catalina.mbeans.JmxRemoteLifecycleListener"
          rmiRegistryPortPlatform="10001" rmiServerPortPlatform="10002" />
```

- 新建访问文件

```
vim jmxremote.access
```

```
# 设置账号权限
admin readwrite
```

- 新建密码文件

```
vim jmxremote.password

# 设置账号密码
admin jx199132
```

- 修改CATALINA.sh 文件

```
CATALINA_OPTS="$CATALINA_OPTS -Xms128m -Xmx200m -XX:PermSize=64M -XX:MaxPermSize=256m -XX:+UseConcM
-Djava.rmi.server.hostname=172.16.72.129 \
-Dcom.sun.management.jmxremote.password.file=/usr/local/apache-tomcat-8.5.45/conf/jmxremote.passwor
-Dcom.sun.management.jmxremote.access.file=/usr/local/apache-tomcat-8.5.45/conf/jmxremote.access \
-Dcom.sun.management.jmxremote.ssl=false"
```

- 连接

```
172.16.72.129:10001
```