## jmap

cd /tmp/hsperfdata\_root

ls

root@iZ94tvmh67fZ:/# jps

7108 Jps

12728 jar

root@iZ94tvmh67fZ:/# jmap -histo 12728 |head -n 30

jmap -dump:format=b,file=heap.hprof 12728

### jmap命令(Java Memory Map) - 上善若水，水善利万物而不争。 - 博客频道 - CSDN.NET

http://blog.csdn.net/xifeijian/article/details/17535375

[**一、jstat命令(Java Virtual Machine Statistics Monitoring Tool)**](http://blog.csdn.net/xifeijian/article/details/17534949)

[**二、JConsole命令(Java Monitoring and Management Console)**](http://blog.csdn.net/xifeijian/article/details/17535359)

[**三、jmap命令(Java Memory Map)**](http://blog.csdn.net/xifeijian/article/details/17535375)

[**四、Java VisualVM(Java Virtual Machine Monitoring, Troubleshooting, and Profiling Tool)**](http://blog.csdn.net/xifeijian/article/details/17535393)

[**五、MAT(Memory Analyzer Tool)工具入门介绍**](http://blog.csdn.net/xifeijian/article/details/17535661)

1、介绍

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）。

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，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=outfile 3024可以将3024进程的内存heap输出出来到outfile文件里，再配合MAT（内存分析工具(Memory Analysis Tool），使用参见：<http://blog.csdn.net/xifeijian/article/details/17535661>）或与jhat (Java Heap Analysis Tool)一起使用，能够以图像的形式直观的展示当前内存是否有问题。

64位机上使用需要使用如下方式：

jmap -J -d64 -heap pid

2、命令格式

SYNOPSIS

       jmap [ option ] pid

       jmap [ option ] executable core

       jmap [ option ] [server-id@]remote-hostname-or-IP

3、参数说明

1)、options：

executable Java executable from which the core dump was produced.

(可能是产生core dump的java可执行程序)

core 将被打印信息的core dump文件

remote-hostname-or-IP 远程debug服务的主机名或ip

server-id 唯一id,假如一台主机上多个远程debug服务

2）、基本参数：

**-dump:**[live,]format=b,file=<filename> 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,那么只输出活的对象到文件.

-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,只统计活的对象数量.

-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，内部String的数量和占用内存数也会打印出来。

-F 强迫.在pid没有相应的时候使用-dump或者-histo参数，在这个模式下，live子参数无效。

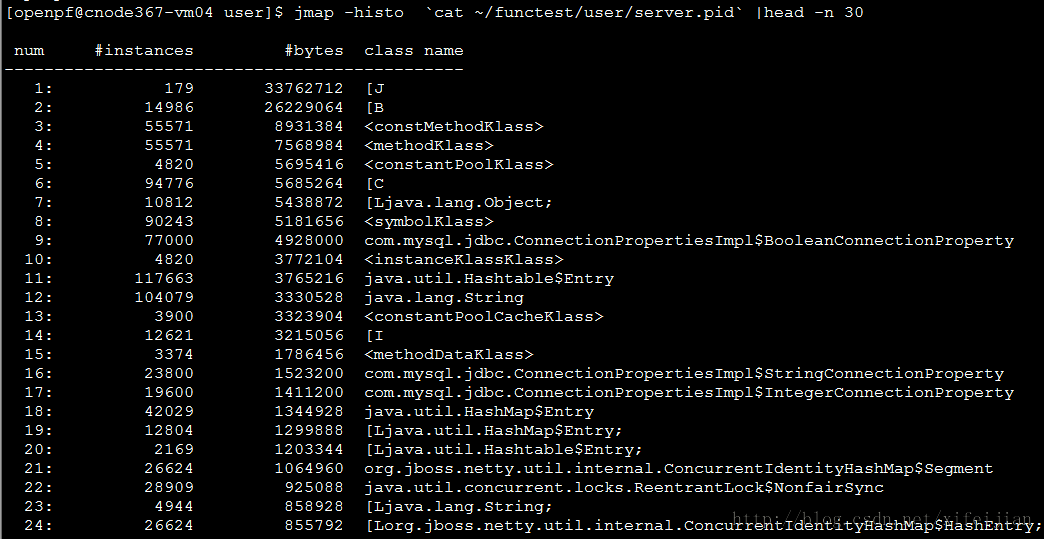
-h | -help 打印辅助信息

-J 传递参数给jmap启动的jvm.

pid 需要被打印配相信息的java进程id可以用jps查问。

4、使用示例

1)、[openpf@cnode367-vm04 user]$ **jmap -histo  `cat ~/functest/user/server.pid` |head -n 30**  （因为太多，只列出前30条）



2）、[openpf@cnode367-vm04 user]$ jmap -dump:format=b,file=./test.bin  `cat ~/functest/user/server.pid`

http://img.blog.csdn.net/20131224201550515?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQveGlmZWlqaWFu/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

版权声明：本文为博主原创文章，未经博主允许不得转载。

### jmap、jstack、jps无法连接jvm解决办法 - windows9834的日志 - 网易博客

http://windows9834.blog.163.com/blog/static/273450042012919101435608/

一、背景

在对线上服务器的java应用dump操作时发现，以下报错，不能dump。jps也获取不到java进程的pid。

# jmap -dump:file=/data/dump/jvm\_en.hprof 20176

20176: Unable to open socket file: target process not responding or HotSpot VM not loaded

The -F option can be used when the target process is not responding

重启后，jps可以获得该java进程的pid，jstack也可以dump线程。

而tomcat:

jdk1.6.24版本下的，jps、jstack都无法操作

jdk1.6.18版本可以执行jps、jstack。

二、原因分析

jvm运行时会生成一个目录hsperfdata\_$USER($USER是启动java进程的用户)，在linux中默认是/tmp。目录下会有些pid文件，存放jvm进程信息。

jps、jstack等工具读取/tmp/hsperfdata\_$USER下的pid文件获取连接信息。

**2.1jstack报错原因**

jstack报错：Unable to open socket file。是因为这个java进程的pid文件删除了。

为什么会被删除呢？这是因为linux操作系统为了防止/tmp目录文件过多，有个删除管理机制：tmpwatch。

查看关键配置/etc/cron.daily/tmpwatch：

flags=-umc /usr/sbin/tmpwatch "$flags"   
-x /tmp/.X11-unix -x /tmp/.XIM-unix \   
 -x /tmp/.font-unix -x /tmp/.ICE-unix   
-x /tmp/.Test-unix 240 /tmp /usr/sbin/tmpwatch "$flags" 720 /var/tmp   
for d in /var/{cache/man,catman}/{cat?,X11R6/cat?,local/cat?};   
do if [ -d "$d" ]; then /usr/sbin/tmpwatch "$flags" -f 720 "$d" fi done

系统每天会用tmpwatch命令检查并删除 /tmp 下超过240小时未访问过的文件和目录。

**2.2高版本jps、jstack不能工作原因**

这是一个从Java 6 update 21 引入的bug [sunbug 7009828](http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=7009828)，在Java 6 update 25修复。具体原因是：

jdk16\_21/24开始，jvm启动时产生进程号的临时文件目录优先使用-Djava.io.tmpdir指定的目录，没有指定-Djava.io.tmpdir参数才使用/tmp/hsperfdata\_$USER。

正好tomcat指定了-Djava.io.tmpdir=${tomcat\_home}/tmp/。而jps、jstack从/tmp/hsperfdata\_$USER目录读取不到pid信息，所以才报错。

三、解决办法

3.1 修改tmpwatch设置

排查对应的/tmp/hsperfdata\_\*的目录，让jvm自己来管理，保证jps,jstat等命令可用。

修改/etc/cron.daily/tmpwatch

/usr/sbin/tmpwatch "$flags" -x /tmp/hsperfdata\_\* -x /tmp/.X11-unix -x /tmp/.XIM-unix  
 -x /tmp/.font-unix -x /tmp/.ICE-unix -x /tmp/.Test-unix 240 /tmp

3.2 修改tomcat配置或者升级jdk

1）修改tomcat的Djava.io.tmpdir参数，统一使用/tmp目录。

修改catalina.sh添加

CATALINA\_TMPDIR=/tmp

重启tomcat

2）升级jdk到Java 6 update 25.

3.3 其他java程序重启

重启java进程，重新生成pid文件。

参考URl：

1. <http://pt.alibaba-inc.com/wp/experience_382/java-long-running-jps-tools-such-as-a-solution-can-not-connect-jvm-2.html>

2. <http://underlap.blogspot.com/2011/03/java-update-breaks-jps-jconsole-etc.html>

另外说明：

1、在JDK 64bit 1.7.0\_01版本也出现了这个问题。  
2、在CentOS6以后，/etc/cron.daily/tmpwatch有所改变  
#! /bin/sh   
flags=-umc   
/usr/sbin/tmpwatch "$flags" -x /tmp/.X11-unix -x /tmp/.XIM-unix \   
        -x /tmp/.font-unix -x /tmp/.ICE-unix -x /tmp/.Test-unix \   
        -X '/tmp/hsperfdata\_\*' 10d /tmp   
/usr/sbin/tmpwatch "$flags" 30d /var/tmp   
for d in /var/{cache/man,catman}/{cat?,X11R6/cat?,local/cat?}; do   
    if [ -d "$d" ]; then   
        /usr/sbin/tmpwatch "$flags" -f 30d "$d"   
    fi   
done  
上面红色字体就是新加入的。  
目录/etc/cron.daily/，这个目录是每天执行一次计划任务的目录，所以说，如果设置了比一天更短的清理时间，它是不起作用的。

### 使用VisualVM查看Java Heap Dump - \_\_kingzone\_\_的专栏 - 博客频道 - CSDN.NET

http://blog.csdn.net/kingzone\_2008/article/details/9077019

浏览Heap Dump

可以使用VisualVM浏览heap dump文件的内容，从而快速查看在堆中分配的对象。Heap dumps在主窗口的heap dump子标签页中显示。你可以打开保存在本地的heap dump文件（.hprof）或者使用VisualVM捕获正在运行的程序的heap dumps。

一个heap dump是Java虚拟机（JVM）在某一时刻所有对象的快照。JVM从堆中为所有的类实例和数组分配内存。当一个对象不再被使用并且没有对它的引用时，垃圾回收器回收其堆内存。通过查看堆，你可以找到对象创建的位置，发现对象的引用。如果JVM试图从堆中移除不再需要的对象时失败了，VisualVM可以定位到离该对象最近的垃圾回收根（garbage collecting root）。

**打开Heap Dump文件**

如果你有一个保存在本地的heap dump文件，在VisualVM中通过File > Load菜单项加载该文件。VisualVM能打开.hprof格式的heap dumps文件。打开后，heap dump作为主窗口的一个子标签页。

**捕获Heap Dump**

可以使用VisualVM捕获一个本地运行的应用程序的heap dump。捕获的heap dump文件是一个临时文件，关闭VisualVM后自动删除，若要保留，需要将其另存为文件。

可通过以下两种方法捕获heap dump：

* 在左侧“Application”（应用程序）子窗口中右击相应的应用程序，选择Heap Dump（堆Dump）。
* 在Monitor（监视）子标签页中点击Heap Dump（堆Dump）按钮。

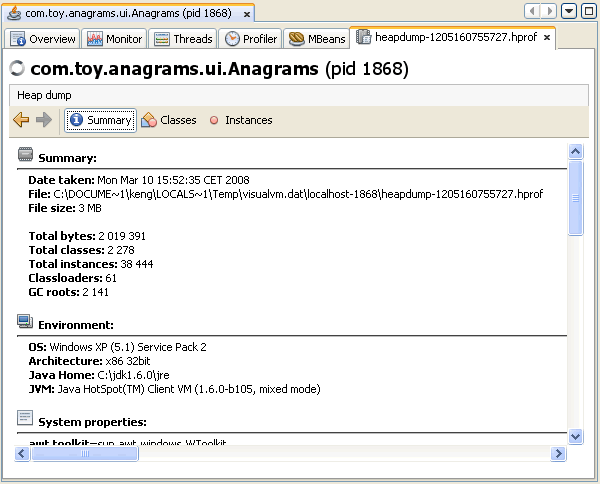
本地应用程序的Heap dumps作为应用程序标签页的一个子标签页打开。同时，heap dump在左侧的Application（应用程序）栏中对应一个含有时间戳的节点。右击这个节点选择save as（另存为）即可将heap dump保存到本地。

浏览Heap Dump

VisualVM提供了一下可视化视图来浏览heap dumps：

**Summary View（概述）**

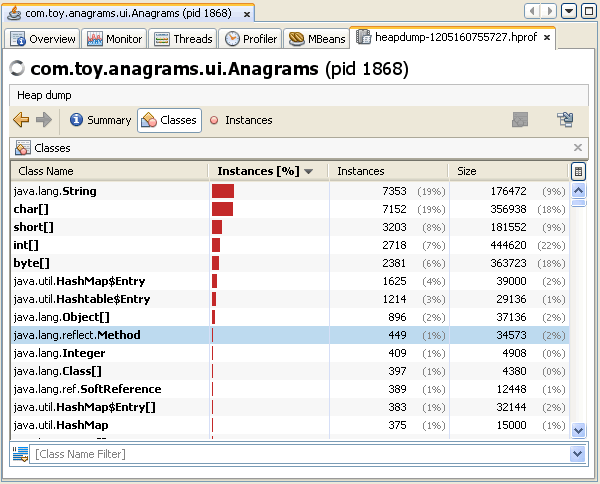
打开一个heap dump时，VisualVM默认显示“概述”标签页。概述视图显示了该heap dump的捕获环境和其他系统属性。



**Classes View（类）**

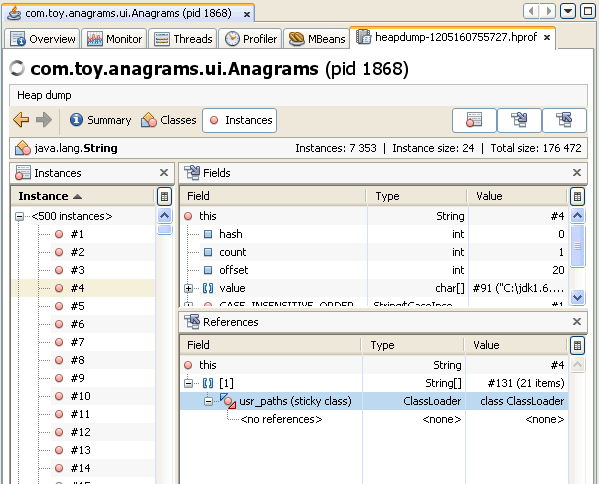
类视图显示了类列表和其对应的实例数量、所占比例。右击类名选择“Show in Instances View（在实例视图中显示）”即可查看指定类的实例列表。

点击列头可按指定列排序。可以使用类列表下方的“类名过滤器（Class Name Filter）”来过滤类，也可以通过右击一个类名选择“Show Only Subclasses（只显示子类）”来将显示结果限制为指定类的子类。



**Instances View（实例数）**

实例数视图显示了选中类的对象实例。在“实例（Instance）”栏中选中一个实例，右侧的“字段（Fields）”和“引用（References）”栏将显示该实例对应的字段和对它的引用。在引用栏中，右击一条并选择“显示最近的垃圾回收根节点（Show Nearest GC Root）”就会显示最近的垃圾回收根节点。



英文原文：[Browsing a Heap Dump](http://visualvm.java.net/heapdump.html)

### 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）。

jmap -histo pid |less

产生那些对象，及其数量

jmap -histo pid>a.log

将结果输出到文本当中

jmap -dump:format=b,file=outfile 3024

可以将3024进程的内存heap输出出来到outfile文件里，结合MAT（内存分析工具(Memory Analysis Tool）以图像的形式直观的展示当前内存是否有问题