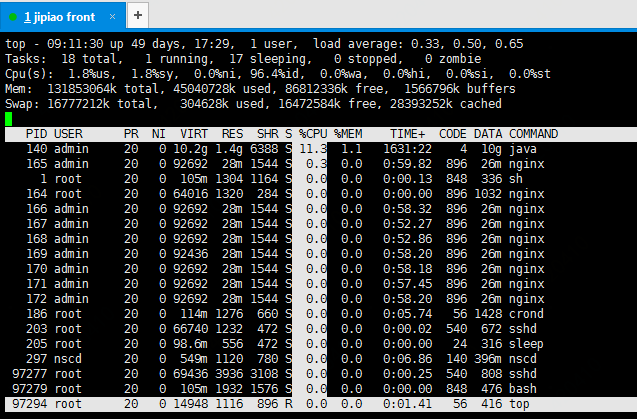
# Cpu占用率过高：

## 1.1．监控机器的系统状态top

使用linux命令取查看机器上的进程使用情况。Top命令。



b: 打开/关闭加亮效果

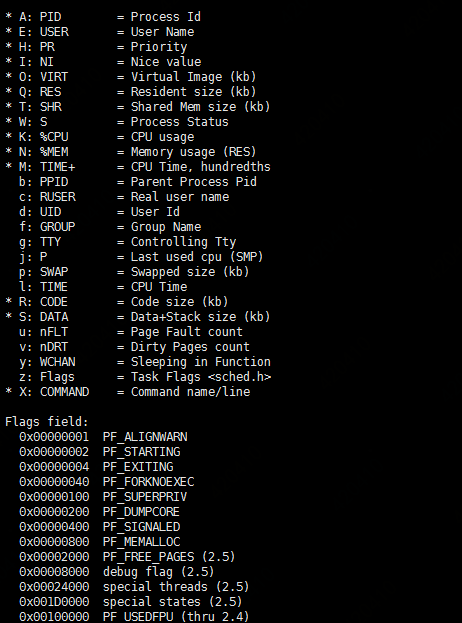
y：打开/关闭运行进程的加亮效果

x：打开/关闭排序列的加亮效果

shift + >或 shift + <: 可以向右或左改变排序列

f: 进入编辑显示字段视图

\*和大写字母的是可显示的字段，小写字母是不可显示的。可以通过输入：r和s是Code和Data字段显示。



## 1.2．查看进程下线程ps、top

使用linux命令取查看该进程下的线程使用情况，得到cpu使用率最高的线程号。

参考：[Linux系统查看CPU使用率的几个命令](https://blog.csdn.net/albenxie/article/details/72885951)

[linux的top命令参数详解](http://www.cnblogs.com/ggjucheng/archive/2012/01/08/2316399.html)

### 方法一：PS

在ps命令中，“-T”选项可以开启线程查看。下面的命令列出了由进程号为<pid>的进程创建的所有线程。

$ ps -T -p <pid>

“SID”栏表示线程ID，而“CMD”栏则显示了线程名称。

### 方法二： Top

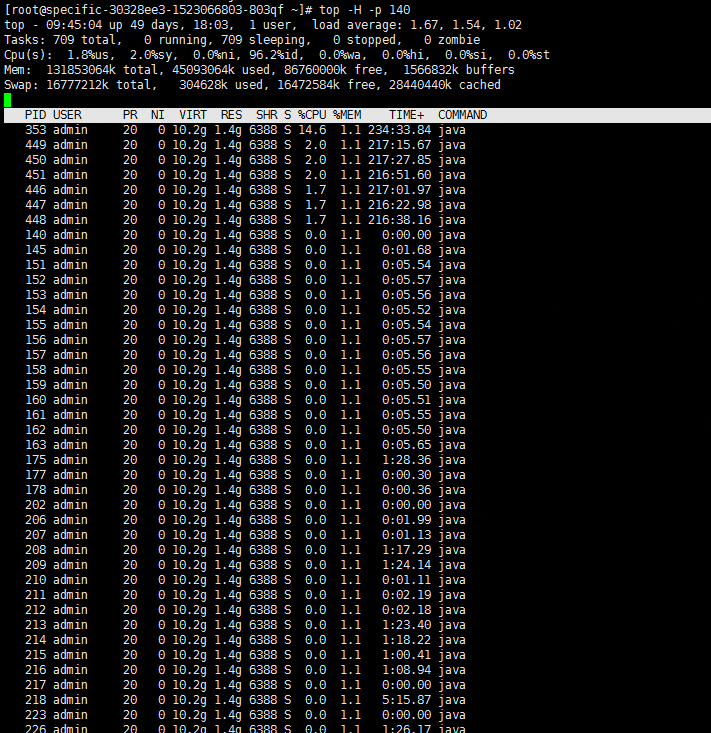
top命令可以实时显示各个线程情况。要在top输出中开启线程查看，请调用top命令的“-H”选项，该选项会列出所有Linux线程。在top运行时，你也可以通过按“H”键将线程查看模式切换为开或关。

$ top -H

要让top输出某个特定进程<pid>并检查该进程内运行的线程状况：

$ top -H -p <pid>

Shift+H可以开启关闭线程显示



1. 使用Jstack去查看该线程下的堆栈信息，看哪块cpu占用多

参考：[Linux有问必答：Linux上如何查看某个进程的线程](https://www.cnblogs.com/lidabo/p/5628058.html)

## 1.3．查看线程的堆栈信息pstack

## 1.4．[Linux查看物理CPU个数、核数、逻辑CPU个数](http://www.cnblogs.com/emanlee/p/3587571.html)

# 总核数 = 物理CPU个数 X 每颗物理CPU的核数

# 总逻辑CPU数 = 物理CPU个数 X 每颗物理CPU的核数 X 超线程数

超线程数一般为2

# 查看物理CPU个数

cat /proc/cpuinfo| grep "physical id"| sort| uniq| wc -l

# 查看每个物理CPU中core的个数(即核数)

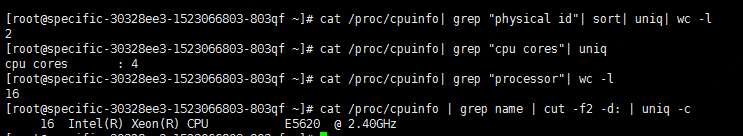
cat /proc/cpuinfo| grep "cpu cores"| uniq

# 查看逻辑CPU的个数

1.cat /proc/cpuinfo| grep "processor"| wc -l

 2.grep -c 'model name' /proc/cpuinfo

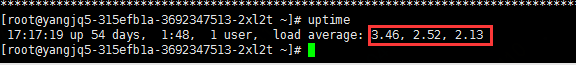
 查看CPU信息（型号）  
cat /proc/cpuinfo | grep name | cut -f2 -d: | uniq -c



# Cpu负荷

参考：[理解Linux系统负荷](http://www.ruanyifeng.com/blog/2011/07/linux_load_average_explained.html)

使用命令upTime，分别代表1，5，15分钟的系统平均负载



系统负荷除以总的核心数，只要每个核心的负荷不超过1.0，就表明电脑正常运行

# 项目中问题

## 机票数据库cpu使用率过高问题排查总结

SELECT

order\_code,user\_id, created,split\_type, parent\_order\_code,order\_cd, payment\_method,trade\_no,user\_level,user\_name,order\_goods,contact\_person,

order\_money, order\_type,place\_order,is\_order\_play,passenger\_number,is\_safe,ticket\_time,play\_time,order\_status,modified,vender\_id,discount\_total\_money,

isab,ab\_status,ab\_retries,pay\_cd,ip,cost\_money,profit\_money,CASE WHEN sourcetype IS NULL THEN 0 ELSE sourcetype END AS sourcetype,insure\_cost\_money,

insure\_profit\_money,internal,is\_payment,UUID,is\_payment\_insure,order\_remark,order\_paid,is\_vip\_lounge,has\_activity,is\_first\_order,sale\_order\_code,

sale\_order\_type,is\_changed,is\_fast\_ticket,big\_invoice,promotion\_beans,is\_vender\_member\_product,env\_type,cancel\_workflow,pay\_timeout,cancel\_reason

FROM `air\_airplane\_order` air

WHERE

((air.order\_status = 1 AND air.is\_order\_play = 0)

OR (air.order\_status = 6 AND air.cancel\_workflow < 7)) AND ABS(air.order\_code) % 5 = 2

ORDER BY created LIMIT 100\G

由于此sql数据的查询涉及到了elastic-job的DataFlow方式的任务执行，因此下一步就考虑是否由于worker的使用不当，导致数据库使用频率过高。

**Dataflow类型作业**

Dataflow类型用于处理数据流，需实现DataflowJob接口。该接口提供2个方法可供覆盖，分别用于抓取(fetchData)和处理(processData)数据。

**流式处理**

可通过DataflowJobConfiguration配置是否流式处理。

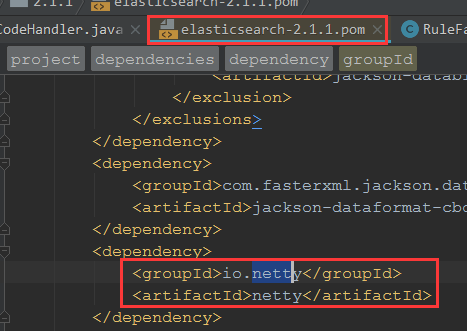
流式处理数据只有fetchData方法的返回值为null或集合长度为空时，作业才停止抓取，否则作业将一直运行下去； 非流式处理数据则只会在每次作业执行过程中执行一次fetchData方法和processData方法，随即完成本次作业。如果采用流式作业处理方式，建议processData处理数据后更新其状态，避免fetchData再次抓取到，从而使得作业永不停止。

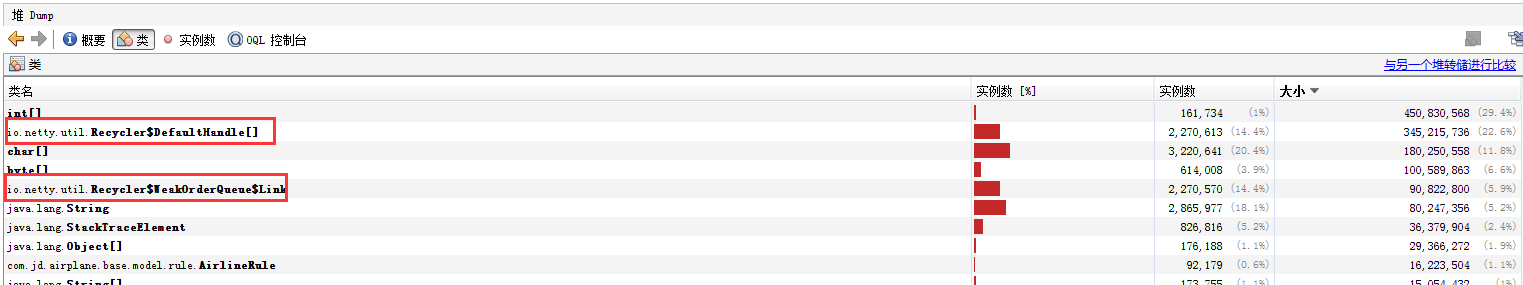
我们有5台服务器，每台服务器处理一部分数据（分片策略）。worker每一分钟获取一次数据，新的数据不断生产出来，流式处理的数据一直不为null，所以作业一直在抓取，导致数据库频繁进行IO，cpu使用率飙升。每

关闭流式处理后，mysql的cpu使用率有明显的下降。

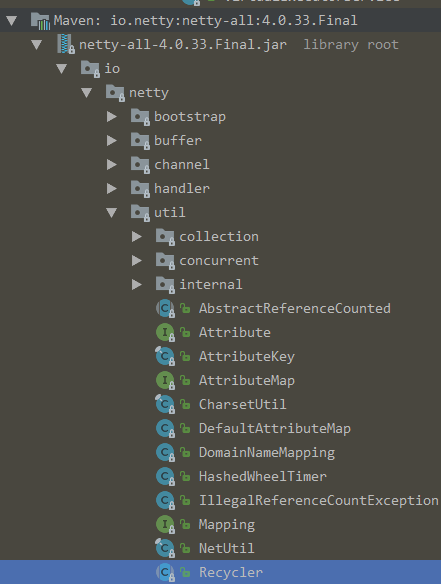
airplane数据库目前最大的连接数为500，虽然我们有数据库连接池缓解压力，但随着worker任务的增加，新的读写操作的增加，都对主库进行操作，压力只会越来越大。所以要考虑读写分离了。

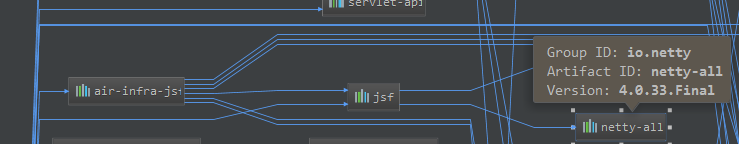
## Flight-rule-server系统内存使用率80%





查到这个类io.netty.util.Recycler在包netty-all中



引用关系如下：最终是air-base-service-impl包中引用

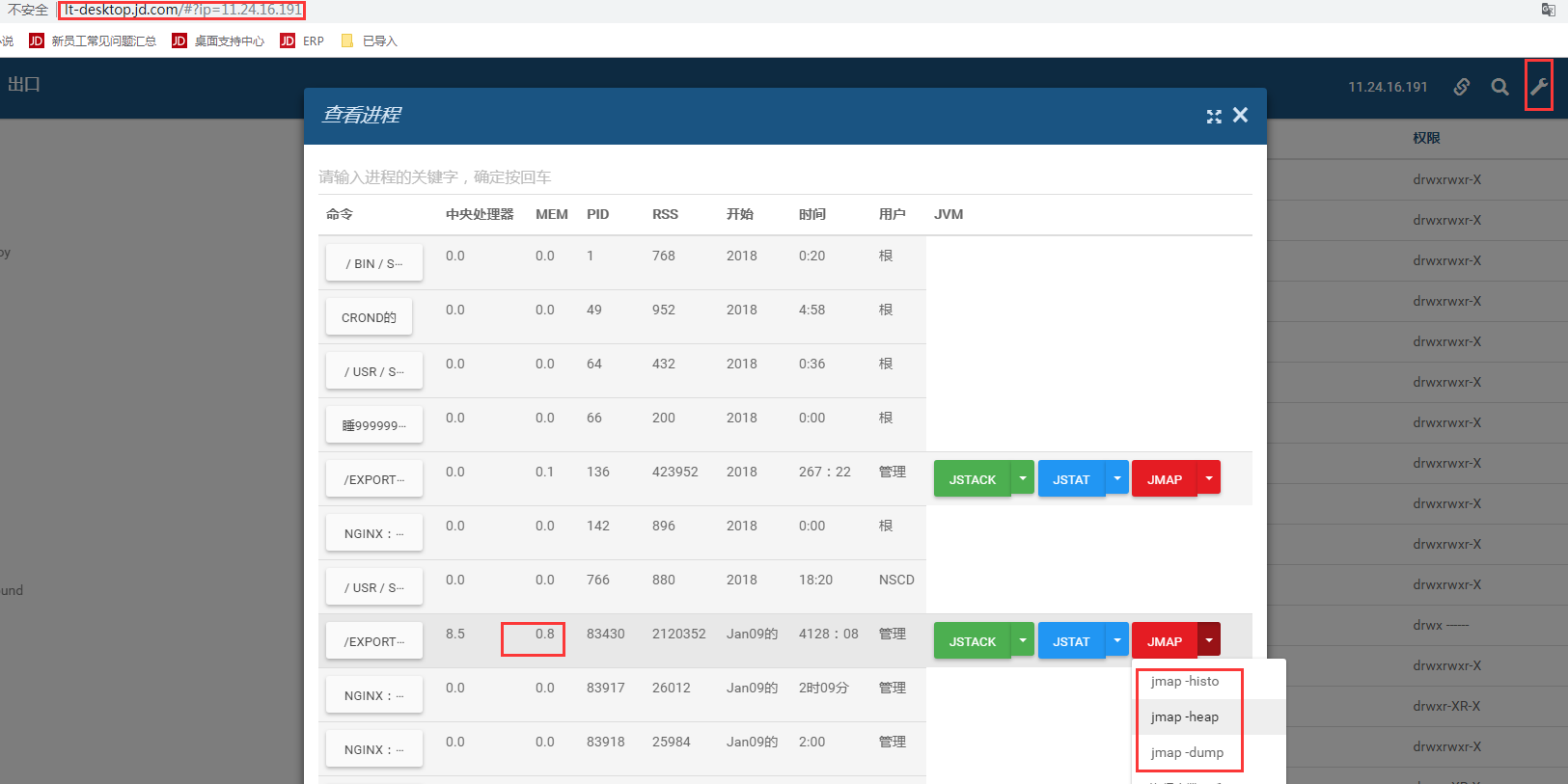
[Netty源码 Recycler 对象池全面解析](https://blog.csdn.net/levena/article/details/78144924)

# 内存使用率过高

[通过 jstack 与 jmap 分析一次线上故障](http://www.importnew.com/28916.html)

[**java命令--jmap命令使用**](https://www.cnblogs.com/kongzhongqijing/articles/3621163.html)

在j-one中通过文件夹链接到it-desktop.jd.com中，使用工具打开，使用jmap命令查看线程的堆内存使用情况。



**虚拟机配置参数：**

/export/servers/jdk1.8.0\_20/bin/java -Djava.util.logging.config.file = / export / Instances / flight-rule-server / server1 / conf / logging.properties -Djava.util.logging.manager = org.apache.juli.ClassLoaderLogManager -Djava.library.path = / usr / local / lib -server -Xms2048m -Xmx2048m -XX：MaxPermSize = 512m -XX：+ UnlockExperimentalVMOptions -Djava.awt.headless = true -Dsun.net .client.defaultConnectTimeout = 60000 -Dsun.net.client.defaultReadTimeout = 60000 -Djmagick.systemclassloader = no -Dnetworkaddress.cache.ttl = 300 -Dsun.net.inetaddr.ttl = 300 -XX：+ HeapDumpOnOutOfMemoryError -XX：HeapDumpPath = / export / Instances / flight-rule-server / server1 / logs -XX：ErrorFile = / export / Instances / flight-rule-server / server1 / logs / java\_error\_％p.log -Djava.endorsed.dirs = / export / servers /tomcat8.0.30/endorsed -classpath /export/servers/tomcat8.0.30/bin/bootstrap.jar:/export/servers/tomcat8.0。30 / bin / tomcat-juli.jar -Dcatalina.base = / export / Instances / flight-rule-server / server1 -Dcatalina.home = / export / servers / tomcat8.0.30 -Djava.io.tmpdir = / export / Instances / flight-rule-server / server1 / temp org.apache.catalina.startup.Bootstrap -config /export/Instances/flight-rule-server/server1/conf/server.xml start

## jmap –heap命令

**执行后的文件：**

Attaching to process ID 83430, please wait...

Debugger attached successfully.

Server compiler detected.

JVM version is 25.20-b23

using thread-local object allocation.

Parallel GC with 43 thread(s)

Heap Configuration:

MinHeapFreeRatio = 0

MaxHeapFreeRatio = 100

MaxHeapSize = 2147483648 (2048.0MB)

NewSize = 715653120 (682.5MB)

MaxNewSize = 715653120 (682.5MB)

OldSize = 1431830528 (1365.5MB)

NewRatio = 2

SurvivorRatio = 8

MetaspaceSize = 21807104 (20.796875MB)

CompressedClassSpaceSize = 1073741824 (1024.0MB)

MaxMetaspaceSize = 17592186044415 MB

G1HeapRegionSize = 0 (0.0MB)

Heap Usage:

PS Young Generation

Eden Space:

capacity = 708313088 (675.5MB)

used = 537061472 (512.1817321777344MB)

free = 171251616 (163.31826782226562MB)

75.82261024096734% used

From Space:

capacity = 3670016 (3.5MB)

used = 3244128 (3.093841552734375MB)

free = 425888 (0.406158447265625MB)

88.39547293526786% used

To Space:

capacity = 3670016 (3.5MB)

used = 0 (0.0MB)

free = 3670016 (3.5MB)

0.0% used

PS Old Generation

capacity = 1431830528 (1365.5MB)

used = 911122232 (868.9138717651367MB)

free = 520708296 (496.5861282348633MB)

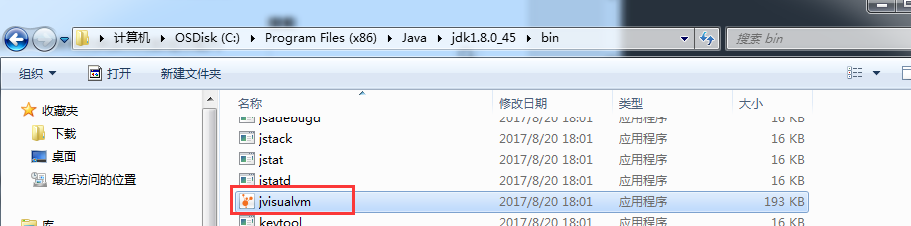
63.633384969984384% used

32547 interned Strings occupying 3557992 bytes.

## jmap –dump命令

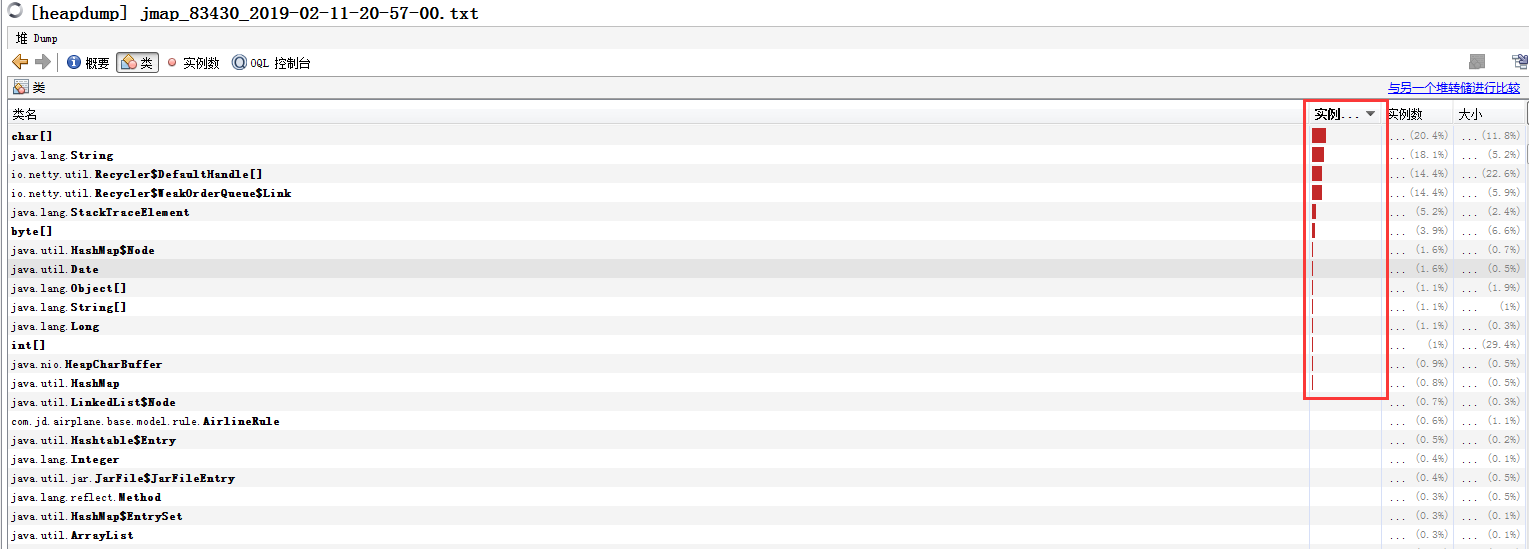
**执行后的文件，见文件**

生成的文件有1.5G，文件过大，不能使用记事本和notepad++打开，使用jvm自带的工具jvisualvm.exe打开



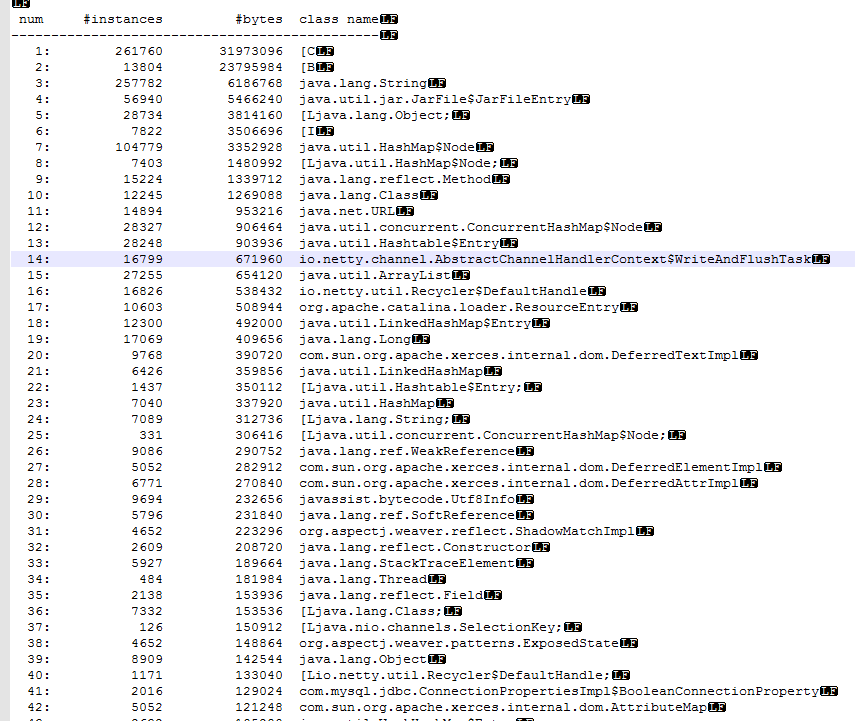
打开后可以看到堆dump情况，按照占用内存大小倒序



按照实例数倒序：  


## jmap –histo命令

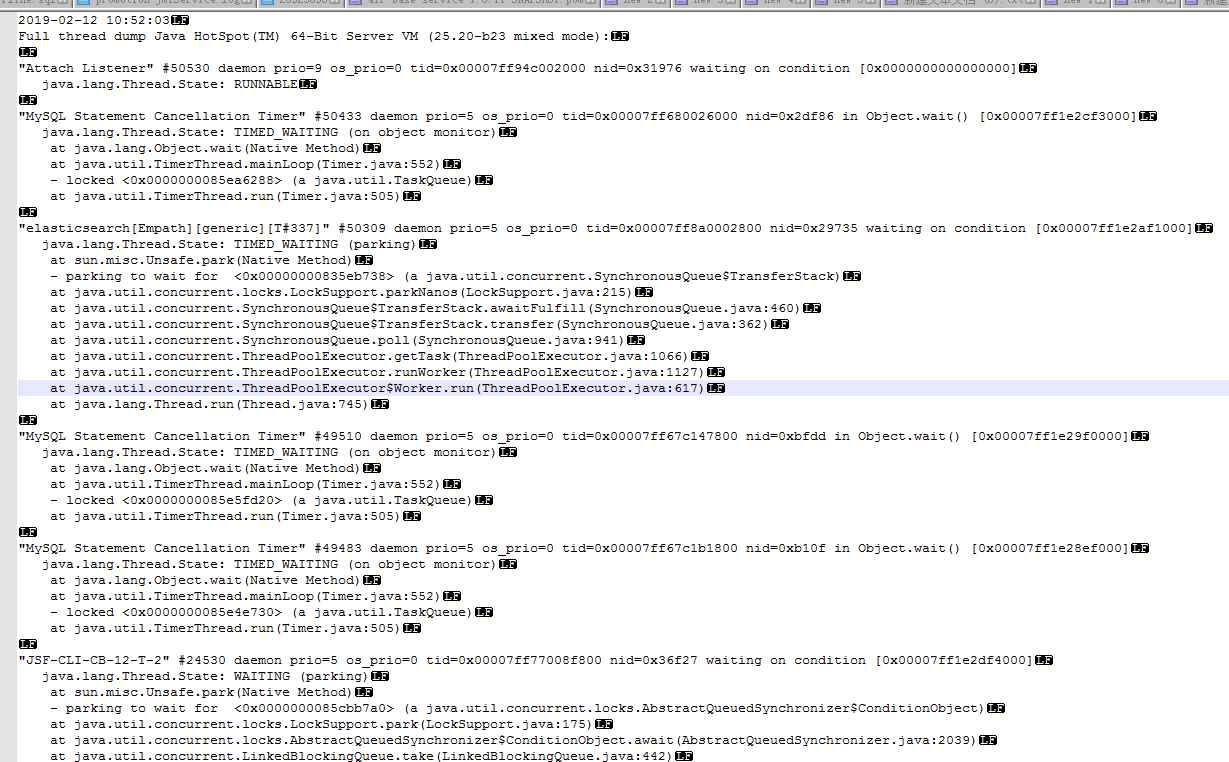
可以展示类的实例数，占用内存大小



## jstack命令

[java命令--jstack 工具](https://www.cnblogs.com/kongzhongqijing/articles/3630264.html)

jstack 查看线程具体在做什么，可看出哪些线程在长时间占用CPU，尽快定位问题和解决问题



## jstat命令

# linux命令

system函数：启动一个shell来执行新的进程程序

exec函数：原来进程将由新程序替换，新程序从其main函数开始执行。因为调用exec并不创建新进程，只是用另一个新进程替换当前进程的正文、数据、堆和栈段，调用exec前后进程ID不会改变

# 环回接口

