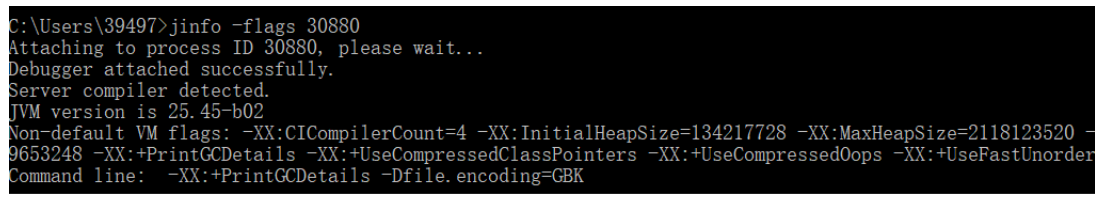
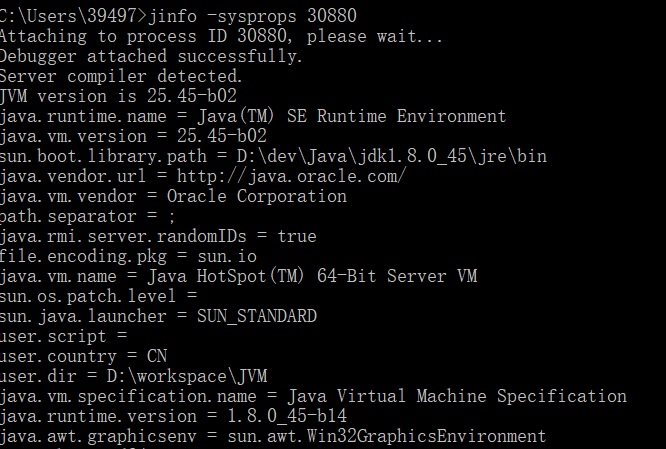
**Jinfo**

查看正在运行的Java应用程序的扩展参数

查看jvm的参数



查看java系统参数



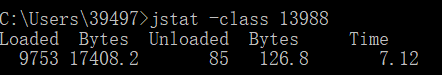
**Jstat**

jstat命令可以查看堆内存各部分的使用量，以及加载类的数量。命令的格式如下：

jstat [-命令选项] [vmid] [间隔时间/毫秒] [查询次数]

注意：使用的jdk版本是jdk8.

类加载统计：



Loaded：加载class的数量

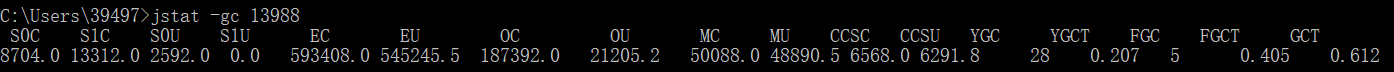
Bytes：所占用空间大小

Unloaded：未加载数量

Bytes:未加载占用空间

Time：时间

垃圾回收统计



S0C：第一个幸存区的大小

S1C：第二个幸存区的大小

S0U：第一个幸存区的使用大小

S1U：第二个幸存区的使用大小

EC：伊甸园区的大小

EU：伊甸园区的使用大小

OC：老年代大小

OU：老年代使用大小

MC：方法区大小(元空间)

MU：方法区使用大小

CCSC:压缩类空间大小

CCSU:压缩类空间使用大小

YGC：年轻代垃圾回收次数

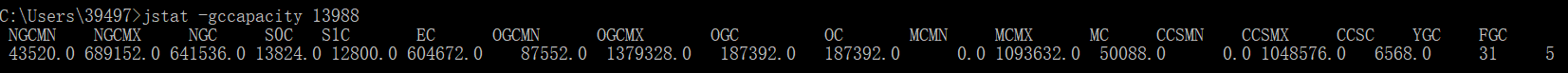
YGCT：年轻代垃圾回收消耗时间

FGC：老年代垃圾回收次数

FGCT：老年代垃圾回收消耗时间

GCT：垃圾回收消耗总时间

堆内存统计



NGCMN：新生代最小容量

NGCMX：新生代最大容量

NGC：当前新生代容量

S0C：第一个幸存区大小

S1C：第二个幸存区的大小

EC：伊甸园区的大小

OGCMN：老年代最小容量

OGCMX：老年代最大容量

OGC：当前老年代大小

OC:当前老年代大小

MCMN:最小元数据容量

MCMX：最大元数据容量

MC：当前元数据空间大小

CCSMN：最小压缩类空间大小

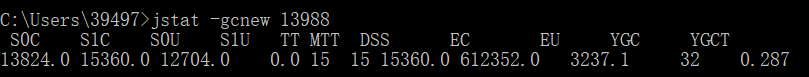
CCSMX：最大压缩类空间大小

CCSC：当前压缩类空间大小

YGC：年轻代gc次数

FGC：老年代GC次数

新生代垃圾回收统计



S0C：第一个幸存区的大小

S1C：第二个幸存区的大小

S0U：第一个幸存区的使用大小

S1U：第二个幸存区的使用大小

TT:对象在新生代存活的次数

MTT:对象在新生代存活的最大次数

DSS:期望的幸存区大小

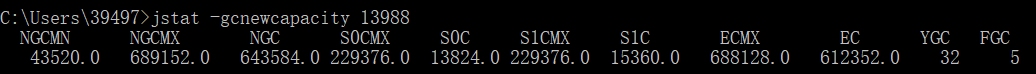
EC：伊甸园区的大小

EU：伊甸园区的使用大小

YGC：年轻代垃圾回收次数

YGCT：年轻代垃圾回收消耗时间

新生代内存统计



NGCMN：新生代最小容量

NGCMX：新生代最大容量

NGC：当前新生代容量

S0CMX：最大幸存1区大小

S0C：当前幸存1区大小

S1CMX：最大幸存2区大小

S1C：当前幸存2区大小

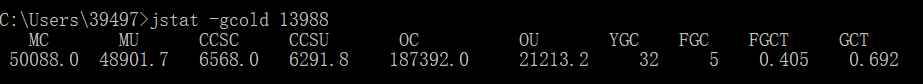
ECMX：最大伊甸园区大小

EC：当前伊甸园区大小

YGC：年轻代垃圾回收次数

FGC：老年代回收次数

老年代垃圾回收统计



MC：方法区大小

MU：方法区使用大小

CCSC:压缩类空间大小

CCSU:压缩类空间使用大小

OC：老年代大小

OU：老年代使用大小

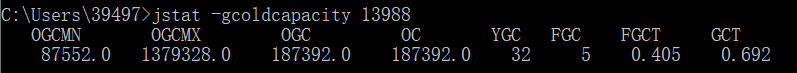
YGC：年轻代垃圾回收次数

FGC：老年代垃圾回收次数

FGCT：老年代垃圾回收消耗时间

GCT：垃圾回收消耗总时间

老年代内存统计



OGCMN：老年代最小容量

OGCMX：老年代最大容量

OGC：当前老年代大小

OC：老年代大小

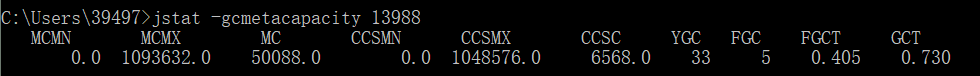
YGC：年轻代垃圾回收次数

FGC：老年代垃圾回收次数

FGCT：老年代垃圾回收消耗时间

GCT：垃圾回收消耗总时间

元数据空间统计



MCMN:最小元数据容量

MCMX：最大元数据容量

MC：当前元数据空间大小

CCSMN：最小压缩类空间大小

CCSMX：最大压缩类空间大小

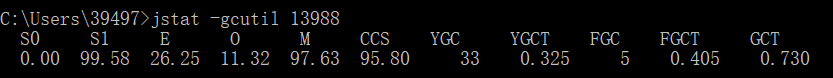
CCSC：当前压缩类空间大小

YGC：年轻代垃圾回收次数

FGC：老年代垃圾回收次数

FGCT：老年代垃圾回收消耗时间

GCT：垃圾回收消耗总时间



S0：幸存1区当前使用比例

S1：幸存2区当前使用比例

E：伊甸园区使用比例

O：老年代使用比例

M：元数据区使用比例

CCS：压缩使用比例

YGC：年轻代垃圾回收次数

FGC：老年代垃圾回收次数

FGCT：老年代垃圾回收消耗时间

GCT：垃圾回收消耗总时间

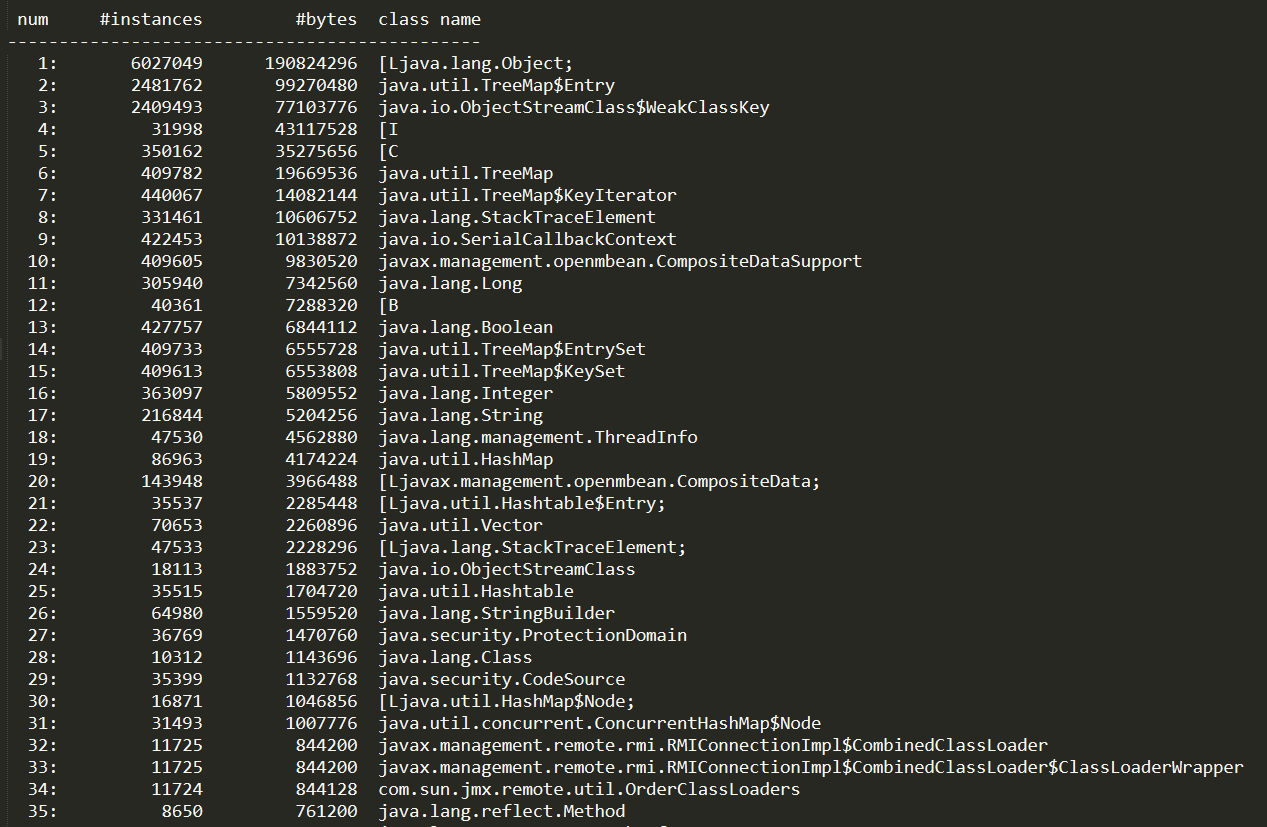
**Jmap**

此命令可以用来查看内存信息。

实例个数以及占用内存大小

clipboard.png

打开log.txt，文件内容如下：



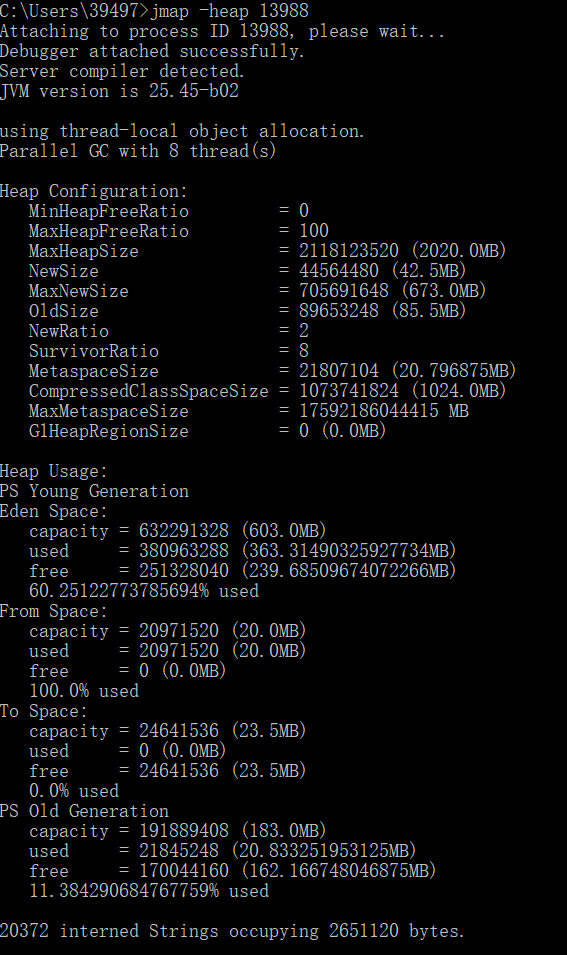
num：序号

instances：实例数量

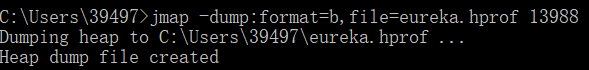
bytes：占用空间大小

class name：类名称

堆信息



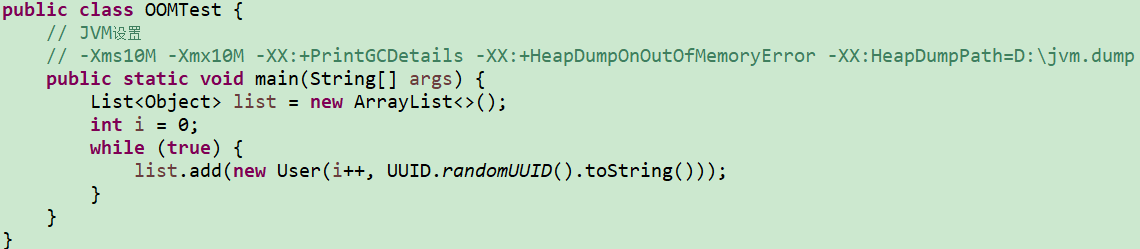
堆内存dump



也可以设置内存溢出自动导出dump文件(内存很大的时候，可能会导不出来)

-XX:+HeapDumpOnOutOfMemoryError

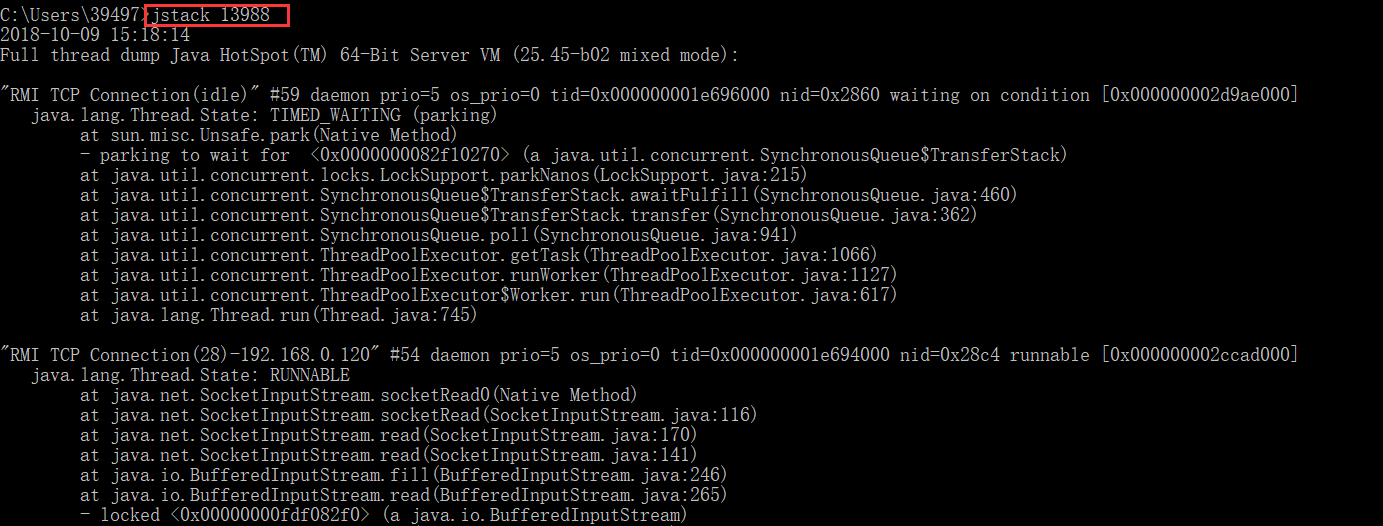
-XX:HeapDumpPath=./   （路径）



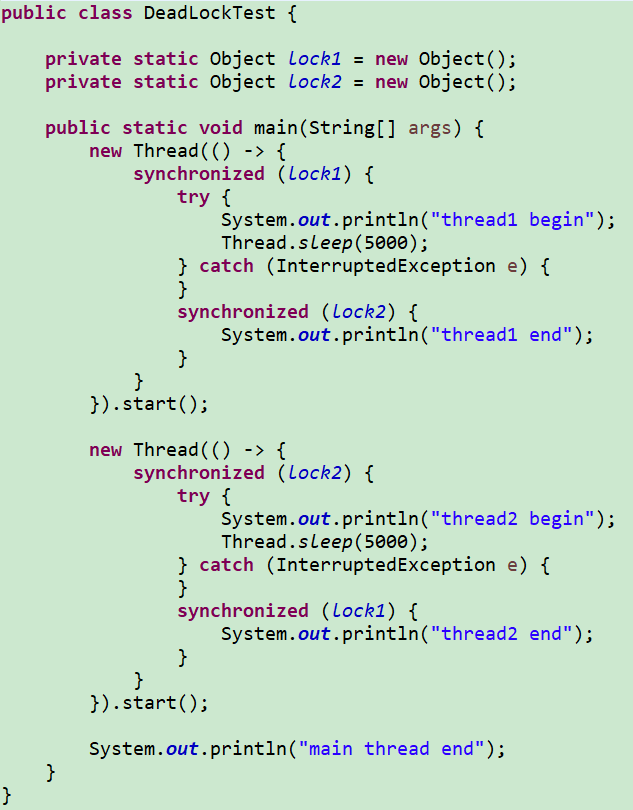
**可以用jvisualvm命令工具导入该dump文件分析**

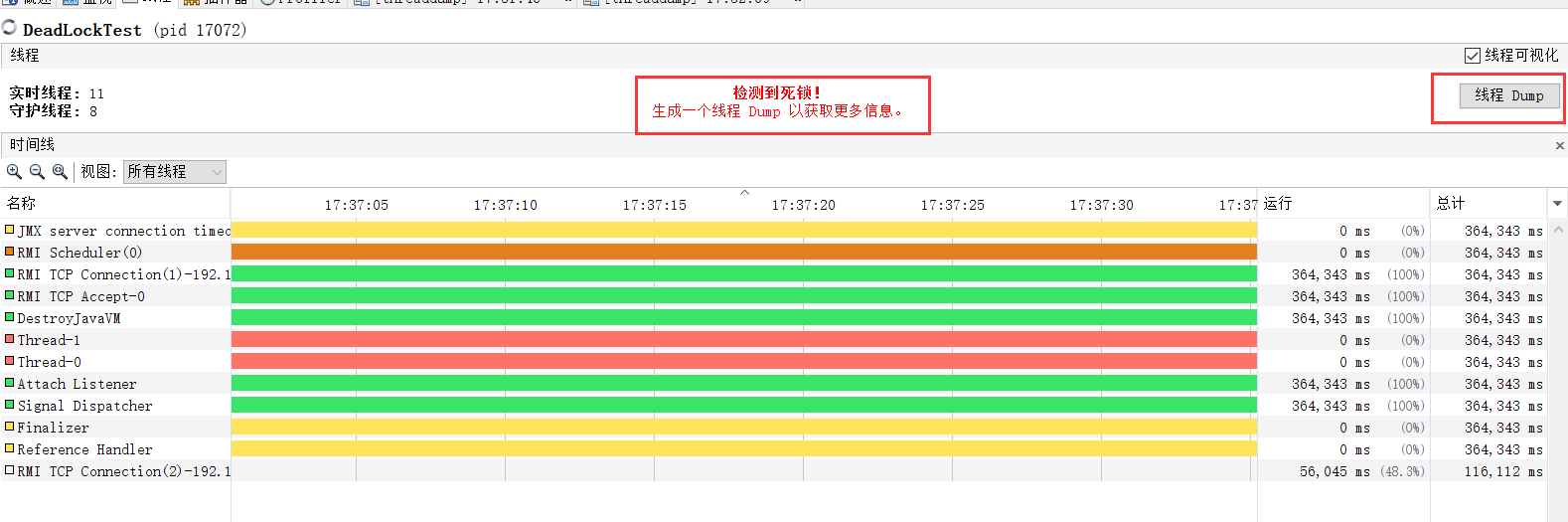


**Jstack**



用jstack查找死锁，见如下示例，也可以用jvisualvm查看死锁





**远程连接jvisualvm**

**启动普通的jar程序JMX端口配置：**

java -Dcom.sun.management.jmxremote.port=12345 -Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.jmxremote.authenticate=false -jar foo.jar

**tomcat的JMX配置**

JAVA\_OPTS=-Dcom.sun.management.jmxremote.port=8999 -Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.jmxremote.authenticate=false

jvisualvm远程连接服务需要在远程服务器上配置host(连接ip 主机名)，并且要关闭防火墙

**jstack找出占用cpu最高的堆栈信息**

1，使用命令top -p <pid> ，显示你的java进程的内存情况，pid是你的java进程号，比如4977

2，按H，获取每个线程的内存情况

3，找到内存和cpu占用最高的线程tid，比如4977

4，转为十六进制得到 0x1371 ,此为线程id的十六进制表示

5，执行 jstack 4977|grep -A 10 1371，得到线程堆栈信息中1371这个线程所在行的后面10行

6，查看对应的堆栈信息找出可能存在问题的代码