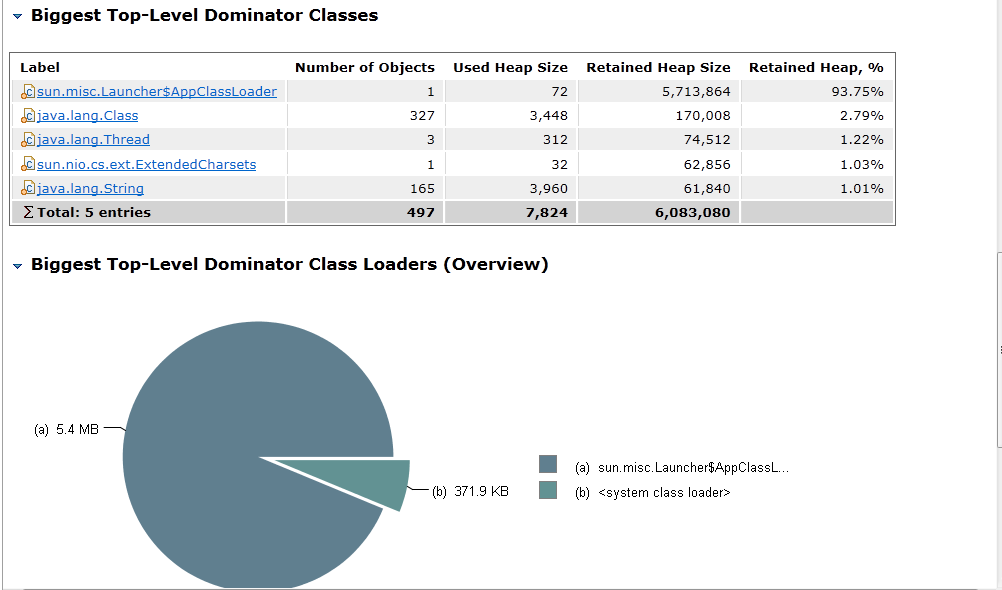
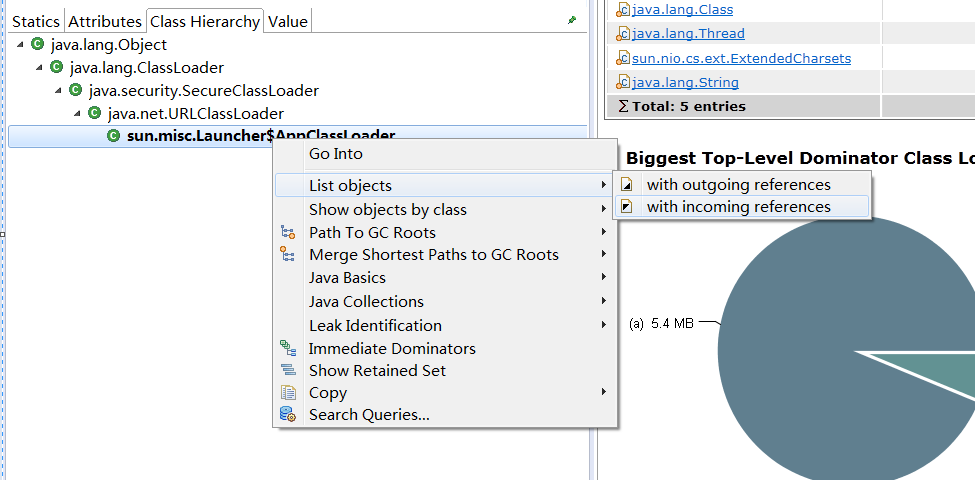
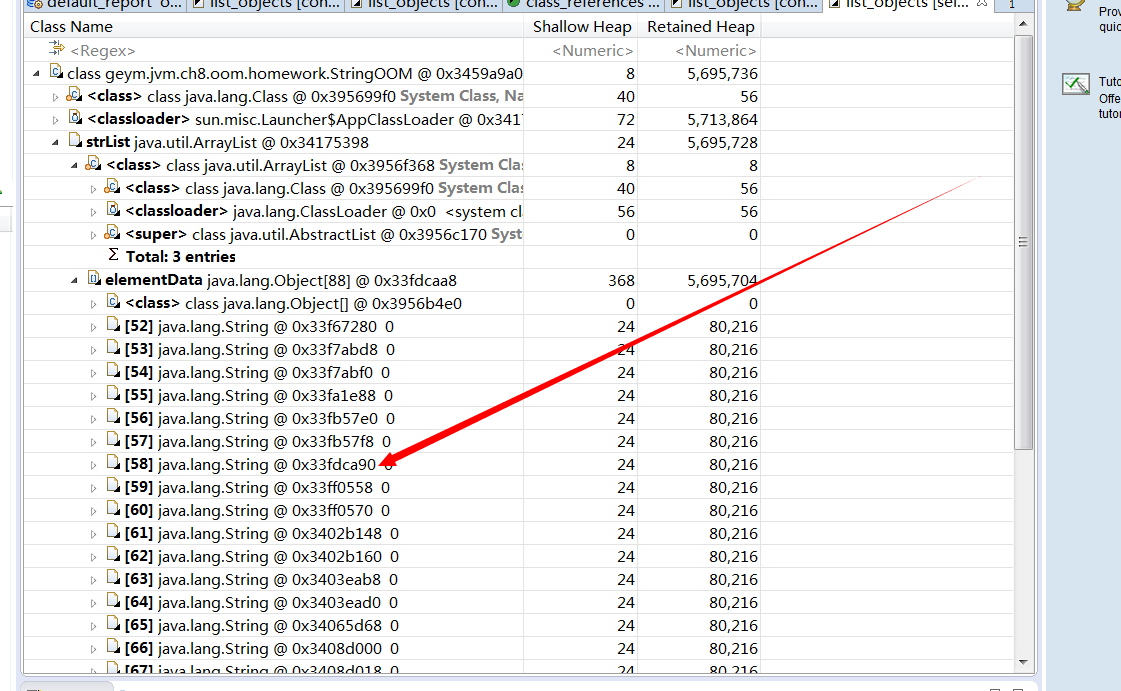
1.对于文件 oom\_第8周第1题.rar(http://pan.baidu.com/s/1kTl32Eb) ，给出最合理的堆溢出的原因

答：通过mat工具,打开堆快照。通过默认分析可以大致看到是appclassloader内存占用很多，并且string实例很多：

。

2通过对appclassloader进行with outcoming references

3、可以依次找到在StringOOM的类中加载了一个String集合， 里面存放了大量String类型的字符串,导致了内存泄露

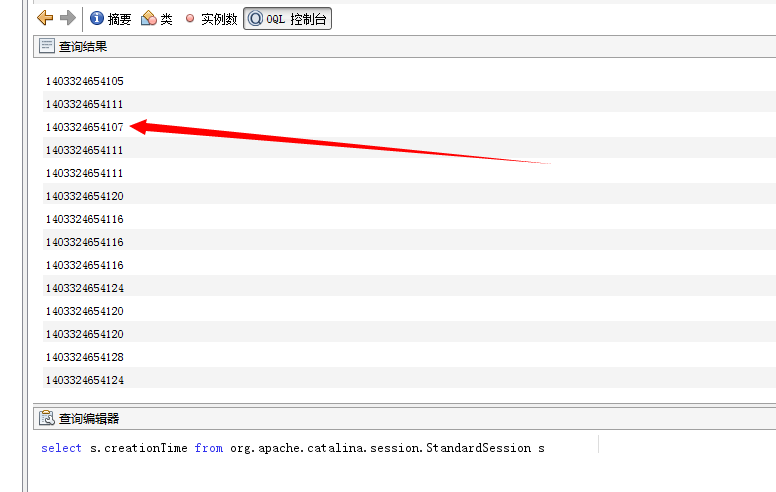


2. tomcat\_第8周第2题.rar（http://pan.baidu.com/s/1hqJz6hY）是tomcat被大量请求后oom的dump，使用jvisualvm的库，以秒为单位，计算得出系统承受负载最大的时刻，给出那个时刻的时间点（起始时间和终止时间合计一秒），以及那一秒钟内，系统的每秒接受的请求数量。比如，在系统 1234毫秒 到 2234毫秒这一秒内，请求量合计500次。 给出你的结论，并提供操作工作截图或相关代码

答：

首先一个请求会创建一个org.apache.catalina.session.StandardSession ,并且Session里面有一个创建时间，可以通过这个创建时间统计一秒内的承受负载，从而统计到最大负载时刻，

1. 通过jvisualvm打开tomcat.hprof的堆文件，



通过简单命令查询，可以得到s.creationTime,并且发现得到的session的creationTime并不是有序的。按照我统计高峰的思路，需要先对session的创建时间升序排序，然后用双重循环依次递增求出最大高峰。

命令如下：

var objArr =sort(heap.objects("org.apache.catalina.session.StandardSession"),'lhs.creationTime - rhs.creationTime')

var size = objArr.length;

var maxNum=0;

var stime = 0;

var etime = 0;

for(var i = 0;i<size;i++){

var tempStime=objArr[i].creationTime;

var num=0;

for(var j = i;j<size;j++){

if(objArr[j].creationTime-tempStime<=1000){

num++;

}else{

if(num>maxNum){

stime=tempStime;

etime=objArr[j-1].creationTime;

maxNum=num;

}

break;

}

}

}

"maxNum:"+maxNum+",stime:"+stime+",really etime:"+etime+",virtual time:"+(parseInt(stime)+1000);

程序运行结果截图：

