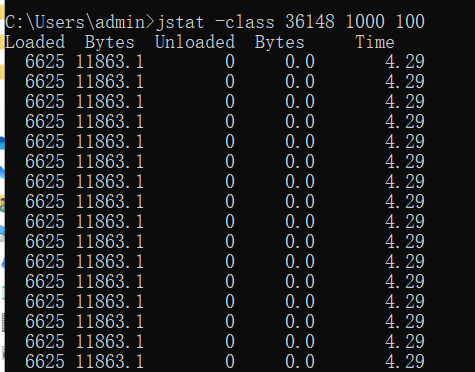
#### G1GC测试

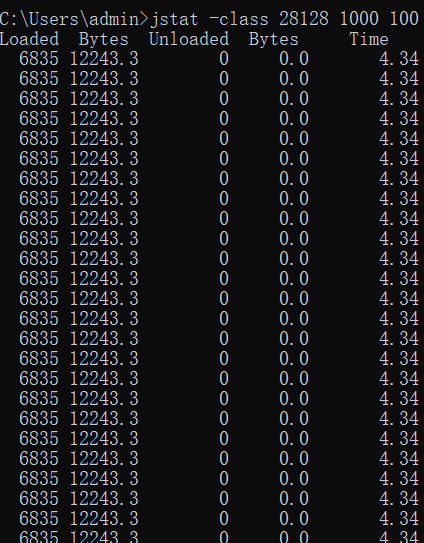
1. 采用G1垃圾回收器，启动jar: Java -Xmx1g -Xms1g -XX:-UseAdaptiveSizePolicy -XX:+UseG1GC -jar gateway-server-0.0.1-SNAPSHOT.jar
2. jstat操作
3. 查看class信息：jstat -class 36148 1000 100

PS:主要包含：类加载数量，类卸载数量，耗费时间

压测前：



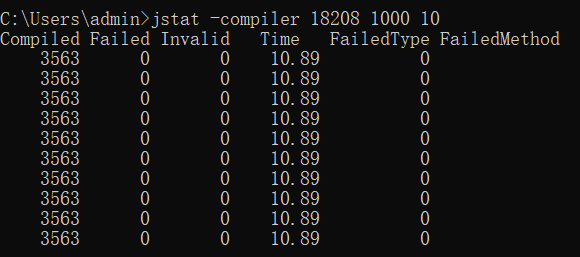
压测：ab -n1000 <http://localhost:8080/api/hello>



压测之后装载数量明显有增多

1. -compiler 编译的类

PS：主要包含编译器编译的方法(数量)和耗时



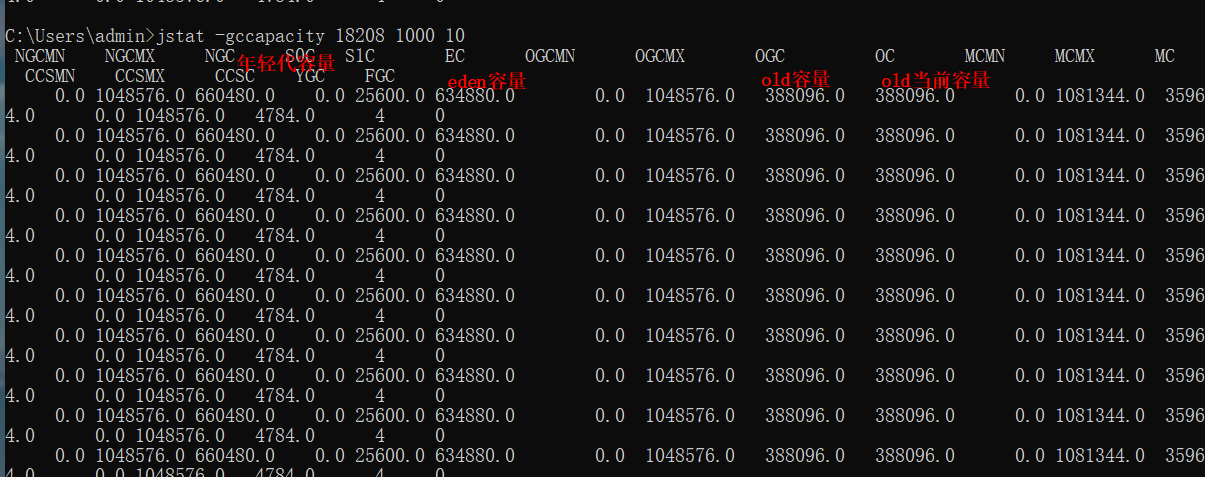
1. -gc: 监控java 堆状况：

压测前与压测后的比较：

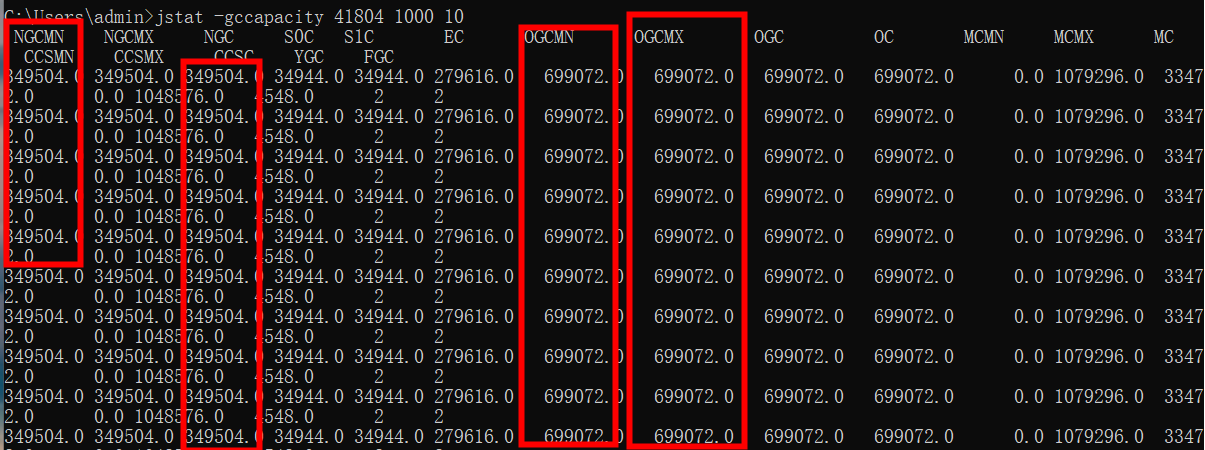


分析：压测前和压测后S1C,S1U明显增大，（扩容比例，触发扩容的阈值等看完复制算法再回来填写）EC变化不大， EU因为了垃圾回收，在压测前后变化比较大，元数据区、压缩类空间的容量和使用量都有明显的变化，压测后YGC时间明显增大，fullGC 为0，且老年代数量没有变化说明没有对象晋升到老年代

1. -gccapacity: 监控内存容量



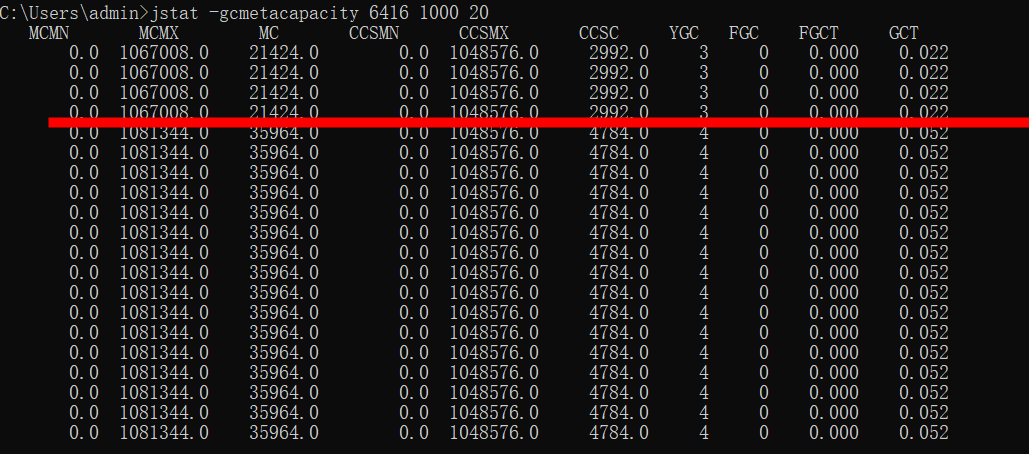
PS：贴一张CMS对比图：



用G1和CMS有几个同的点，G1的新生代，老年代的最大容量，存活区自容量和CMS中不太一样， 新生代和老年代的最大容量设置为1G，应该是和G1采用region方式有关

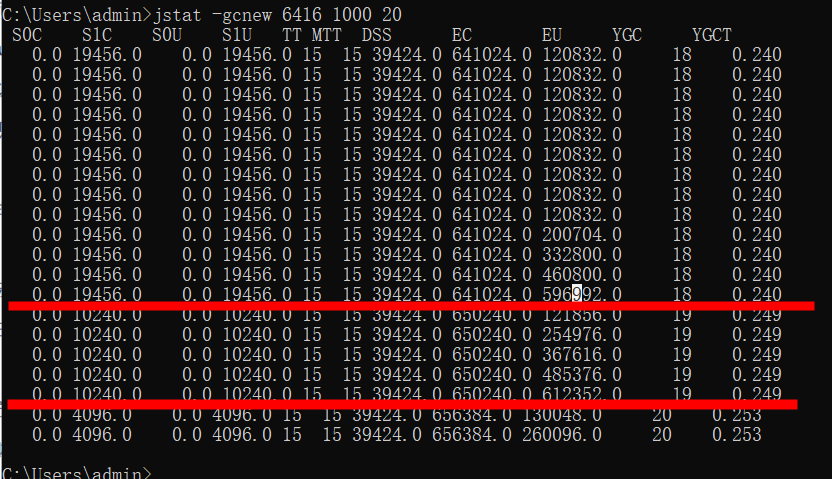
1. -gcmetacapacity:

压测前后，元数据区容量与压缩空间容量，GC时间都有明显的增大， 元数据区包括方法区，运行时常量池，类元信息等

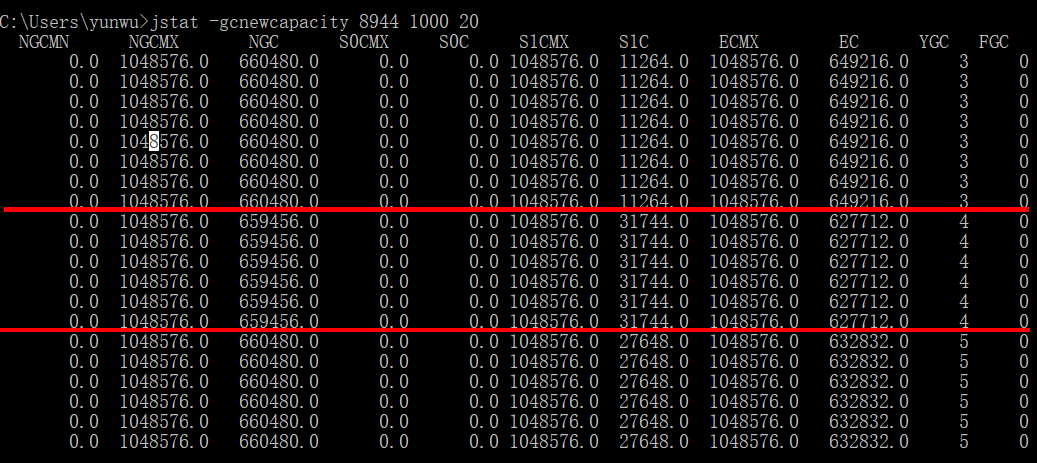


1. jstat -gcnew

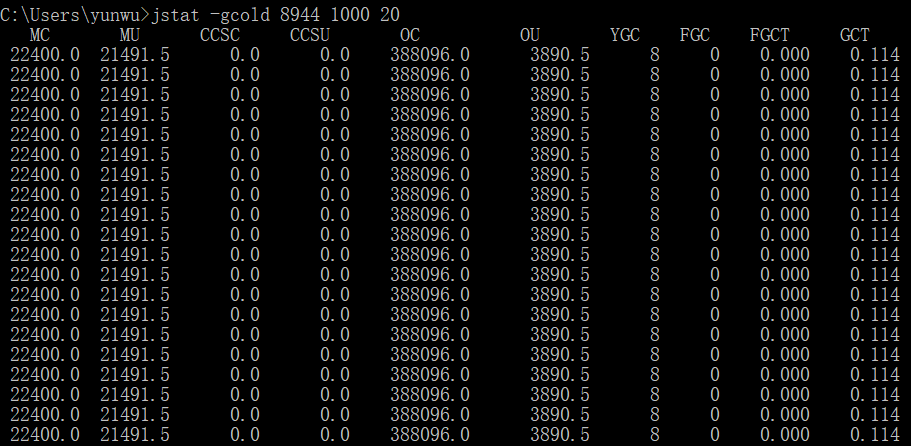
两条线是压测时发生YGC的点，Eden区变化可以体现



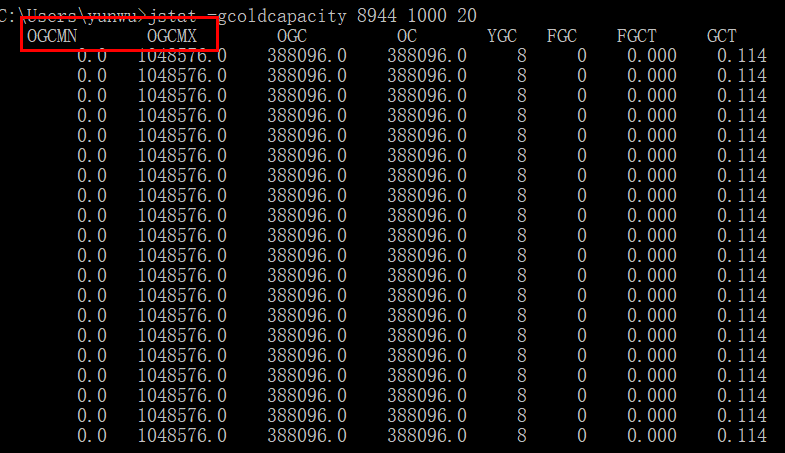
1. jstat -gcnewcapacity:



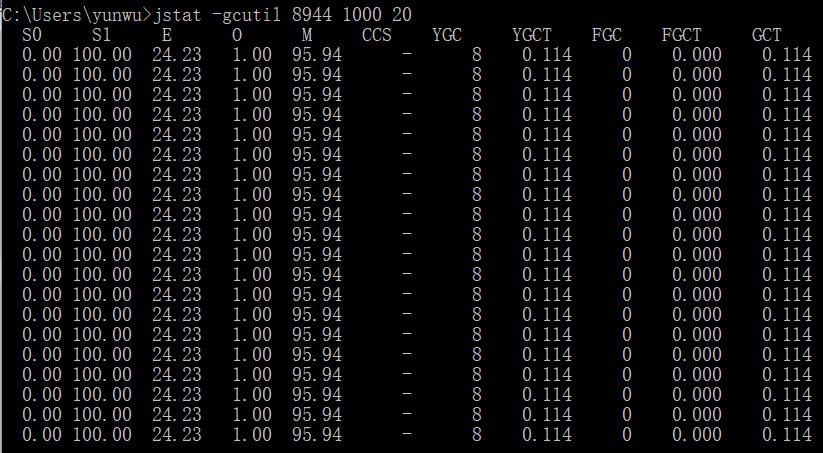
（8）jstat -gcold: 明显看出没有发生fullGC



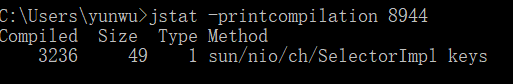
（9）jstat -gcoldcapacity:相对于上面的命令增加了old区容量的最大值最小值两个字段



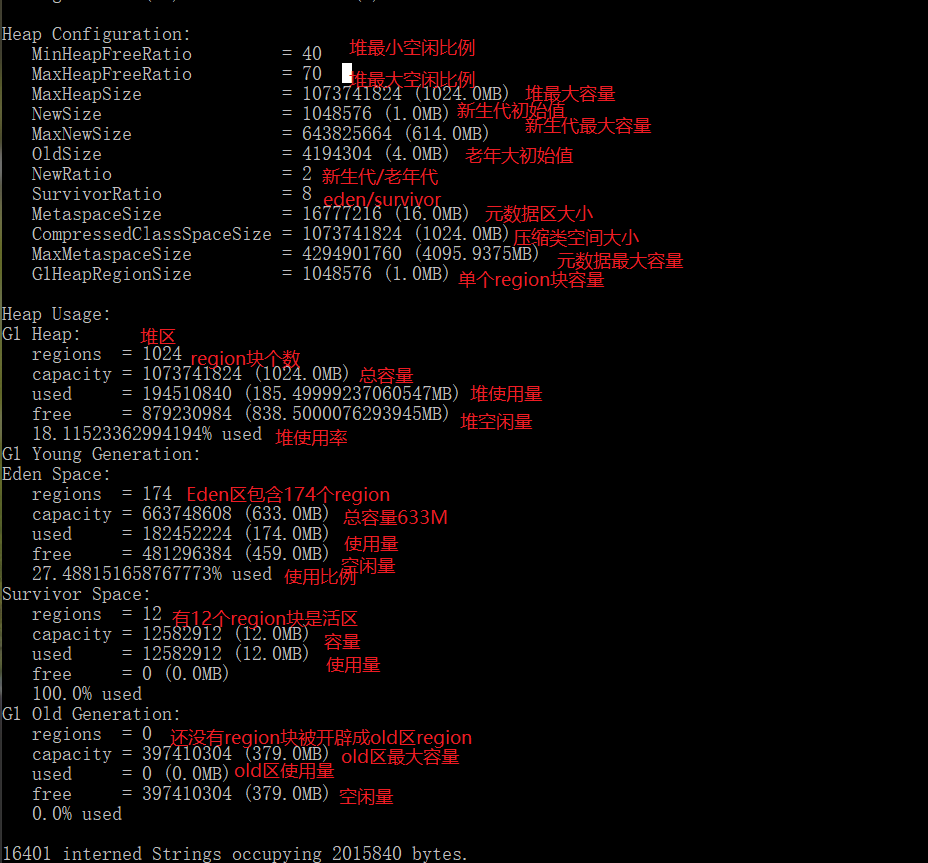
1. jstat -gcutil :当前各区的使用比例，区块使用空间/区块容量



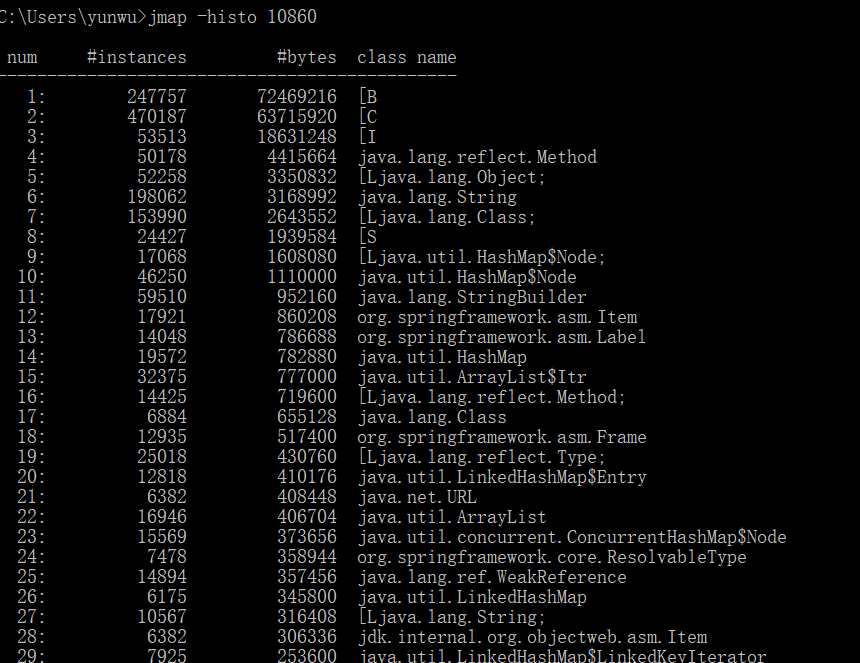
1. jstat -printcompilation pid: 打印最近编译的类信息



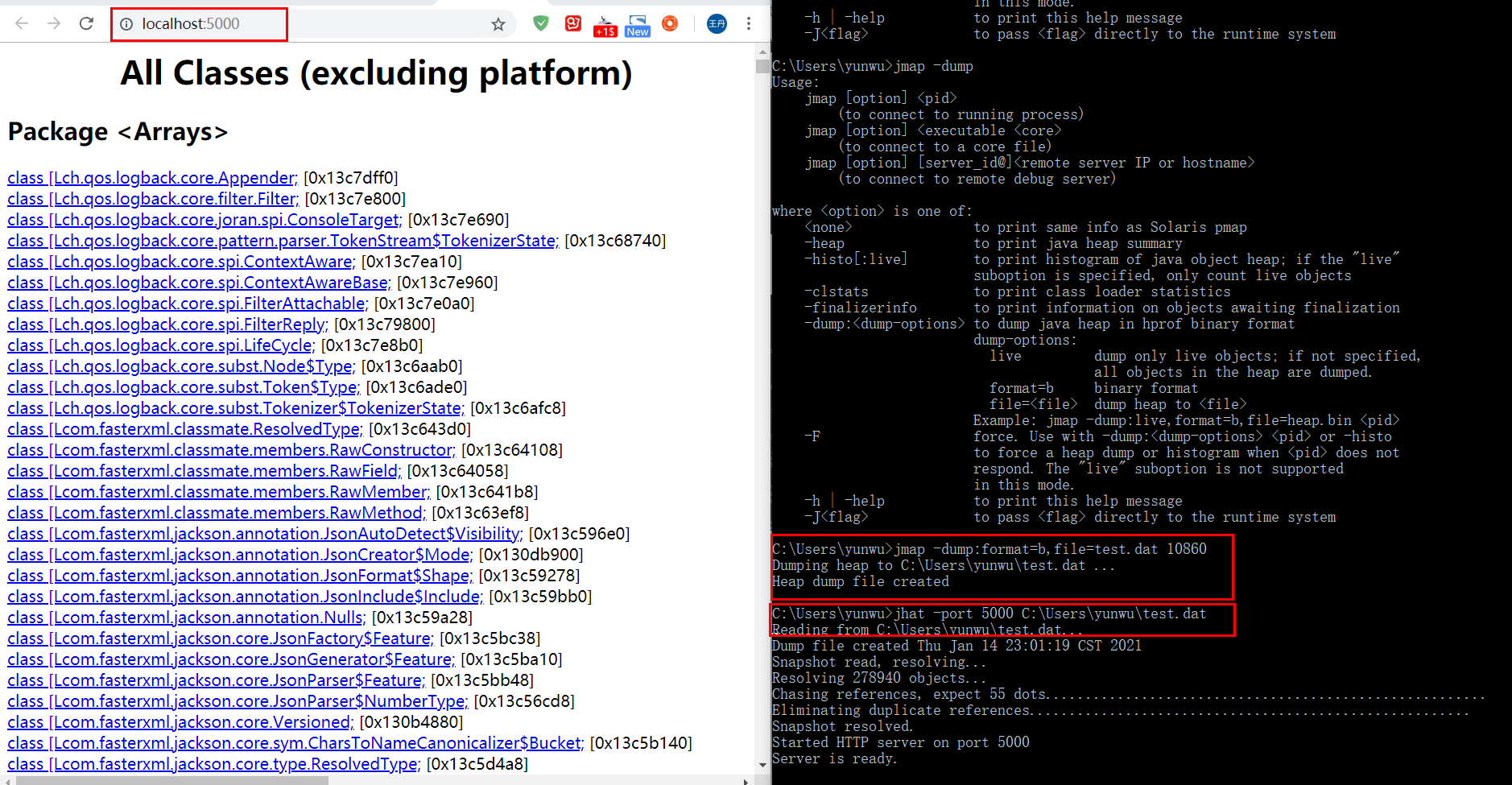
1. Jmap操作
2. jmap -heap 监控堆区状况



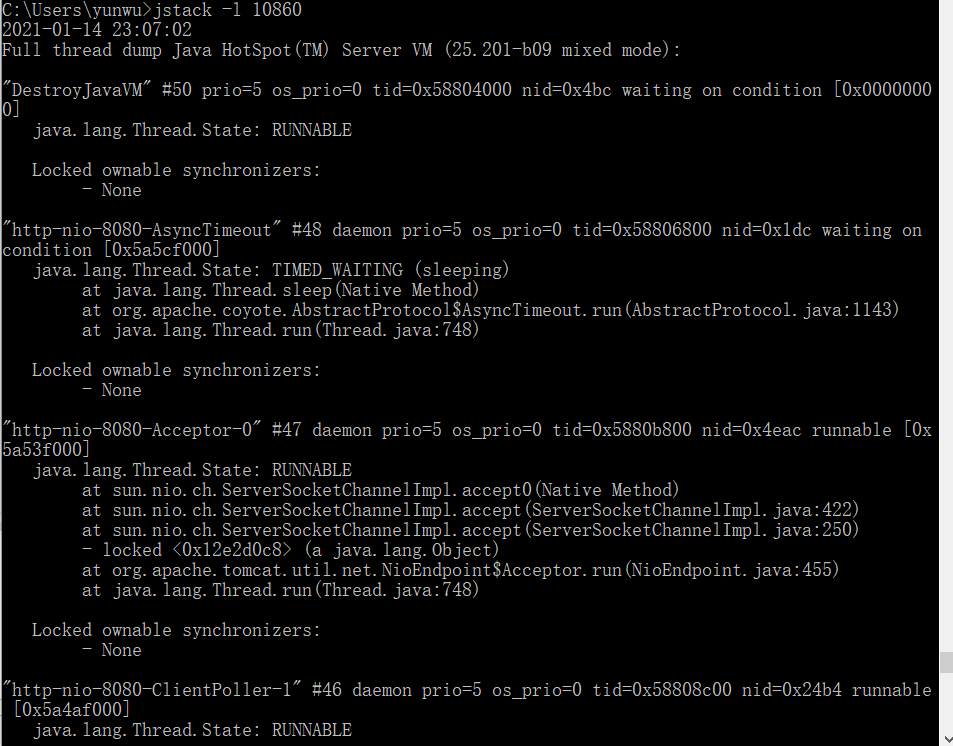
1. jmap -histo：根据对象数量排序，统计堆中对象



1. jmap -dump:format=b,file=test 将内存信息导出到文件中



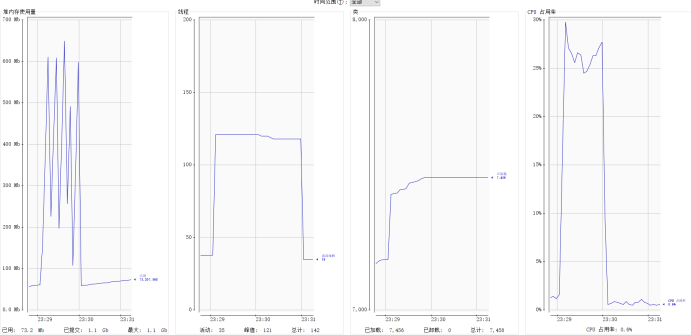
1. Jstack 使用
2. jstack -l pid



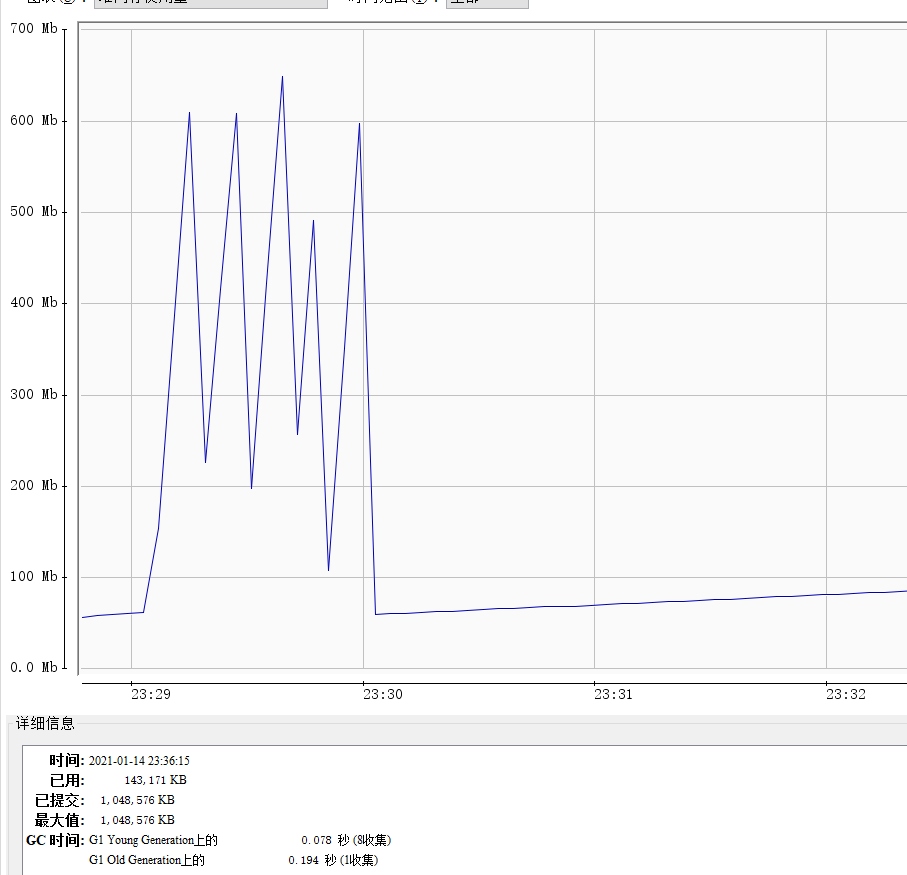
Fastthread生成的报告：



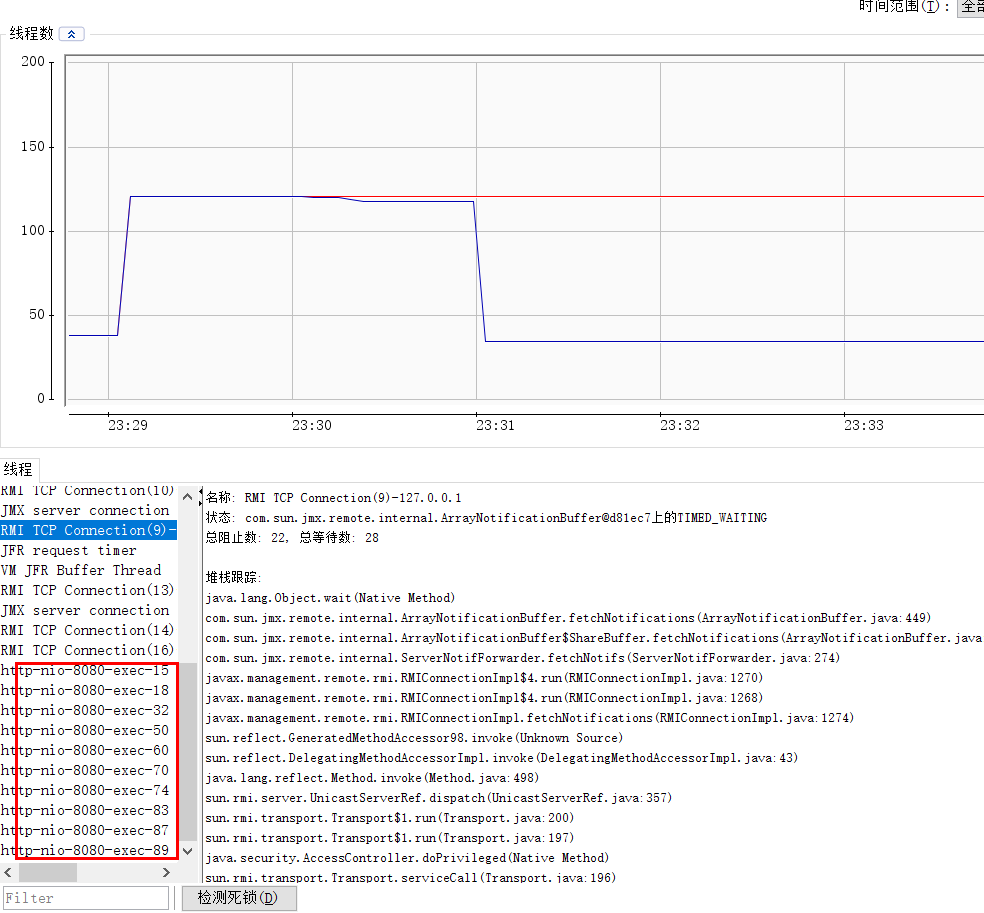
1. Jconsole, 100000请求， 100个线程并发：



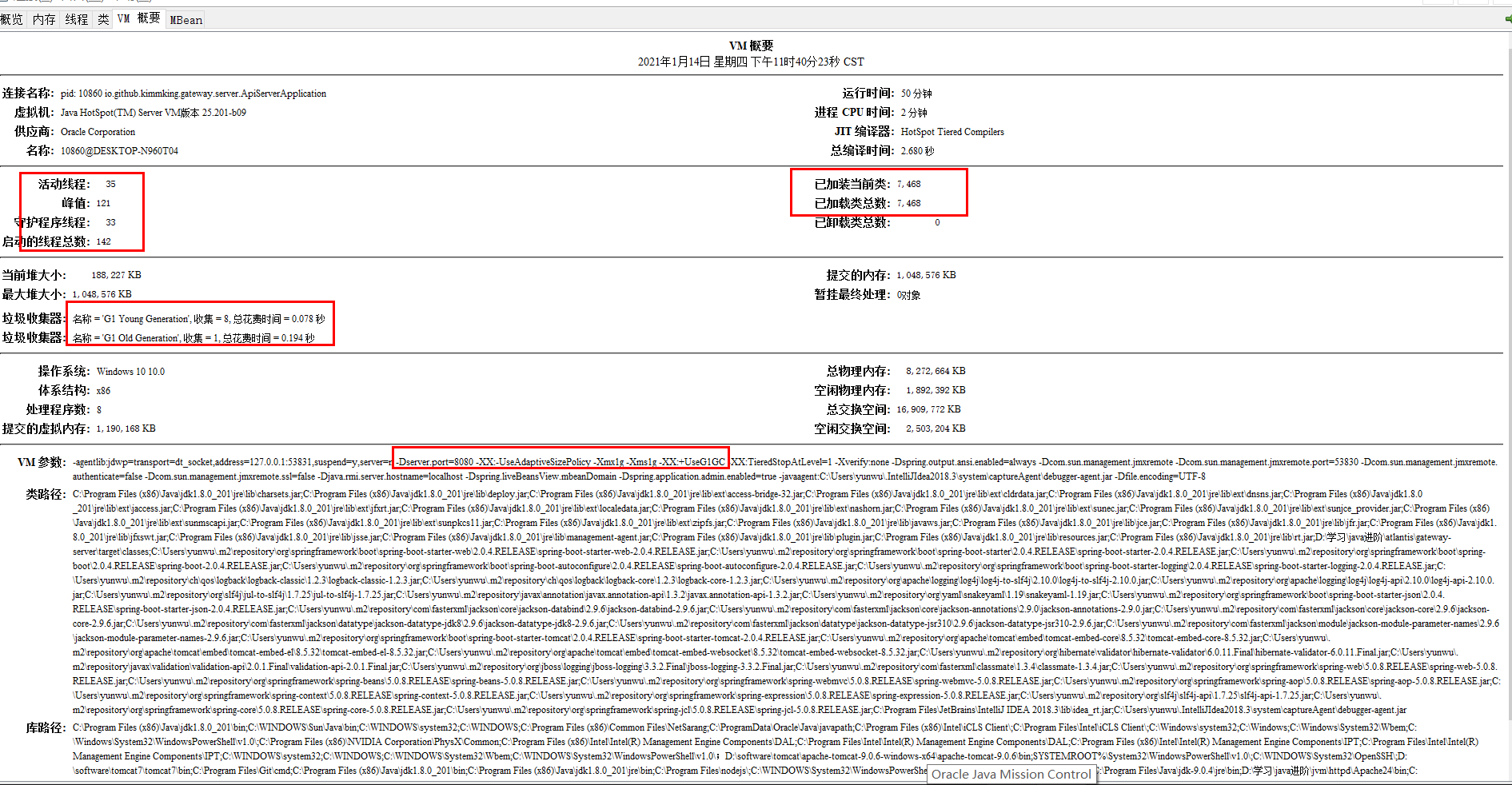
堆内存使用以及回收信息：



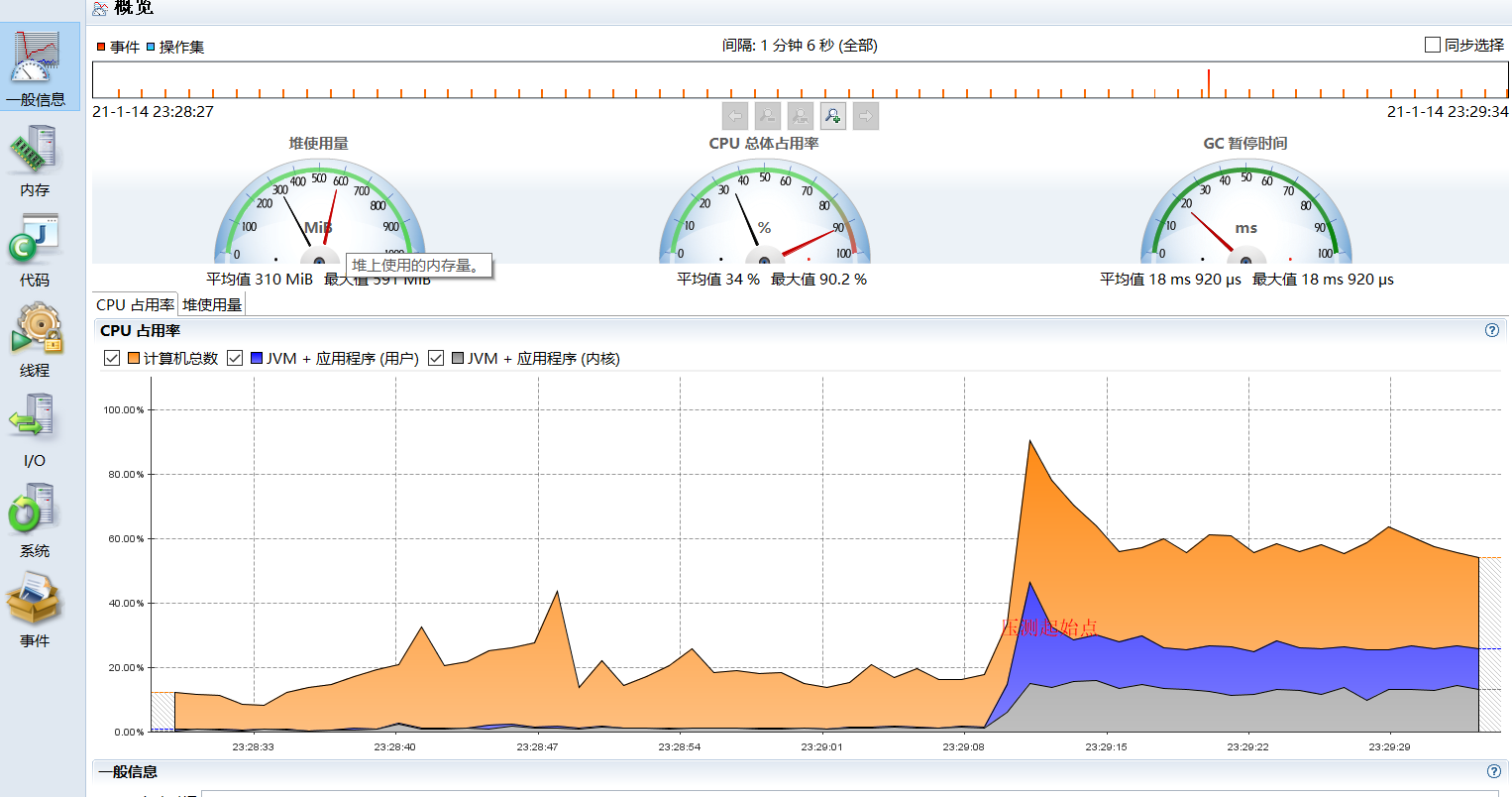
线程信息，因为有并发操作，会有很多线程



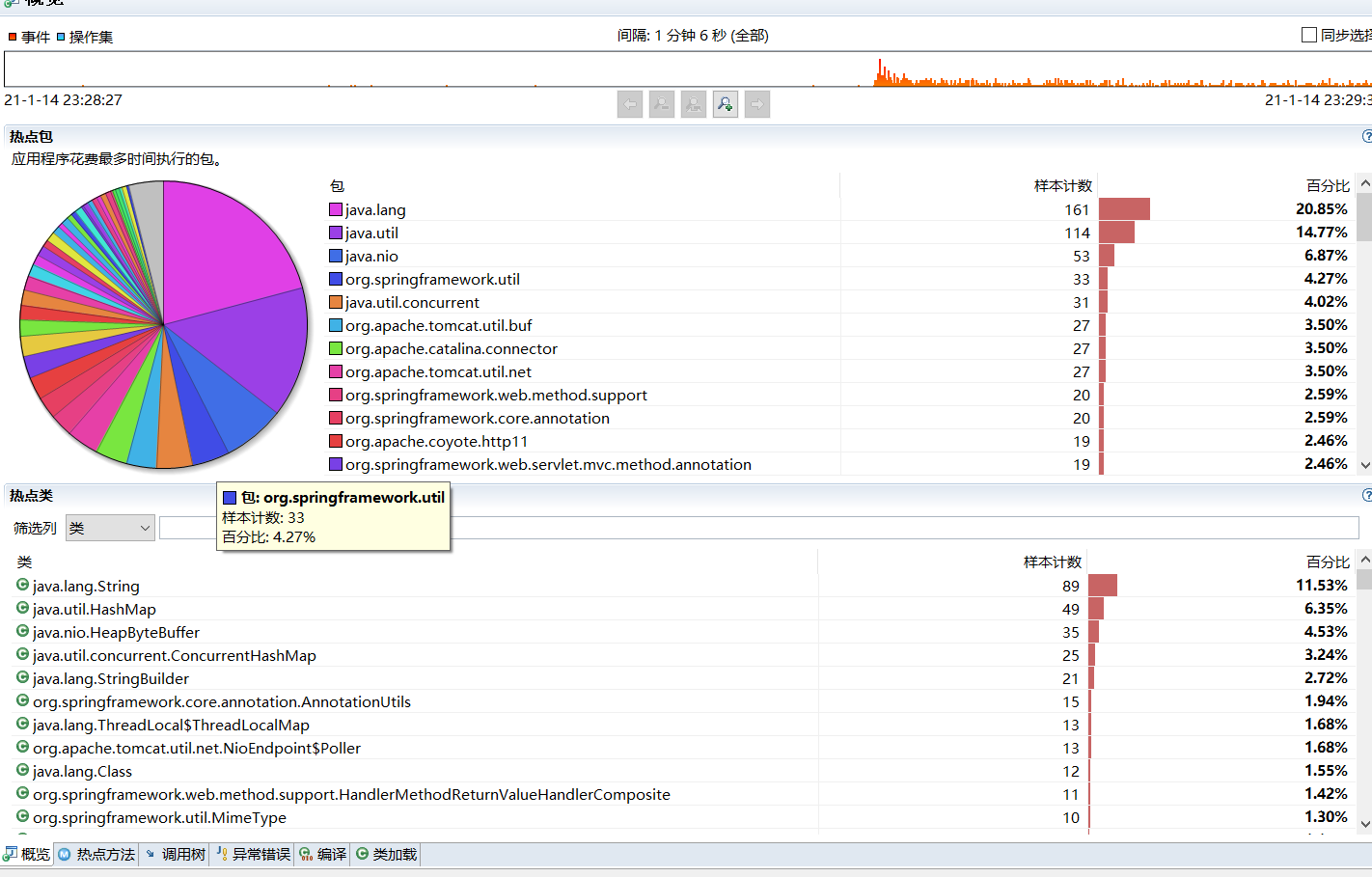
VM概要



1. JMC分析：



热点代码：



1. Idea VisualGC: 100000请求，100个线程并发

