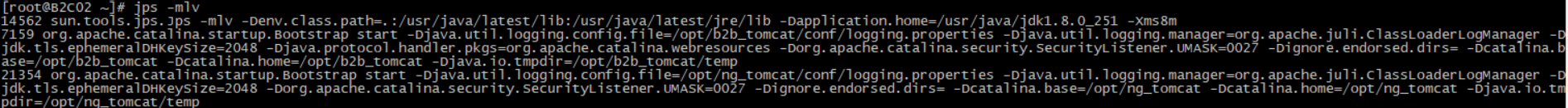
这个是我们测试环境下的用于测试的业务系统，因此在进行分析的时候系统重启了若干次，所以PID是有变化的，但并不影响后续的分析。

该测试环境是一台8核32G的阿里云云主机，系统是CentOS6.8，JDK是1.8，系统是部署在Tomcat下，Tomcat是7.x。

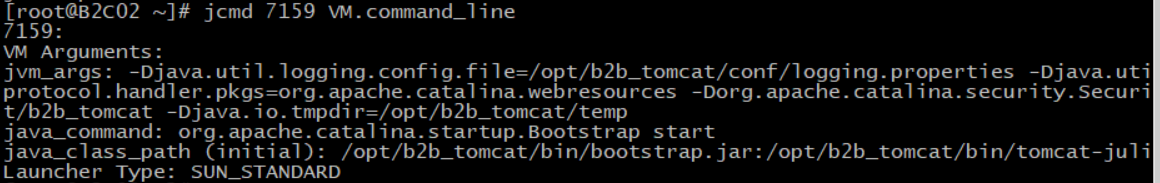
以下是操作和分析。

首先使用jps -mlv，查看各java进程



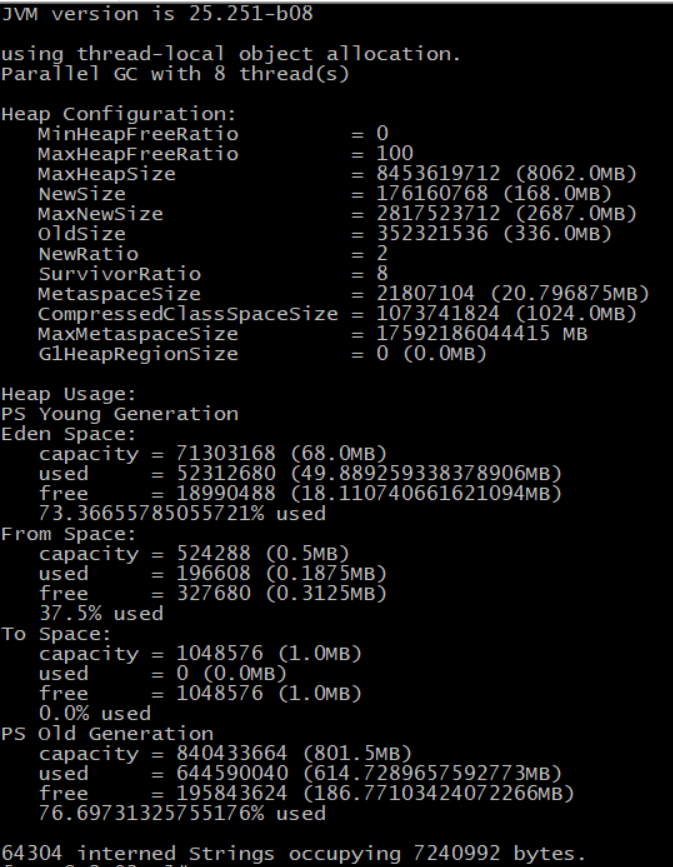
其中pid为7159的进程是目标系统的java进程，还有其他系统进程，与后续分析关联不大。

使用jcmd 7159 VM.command\_line，更仔细看一下命令行传入的参数



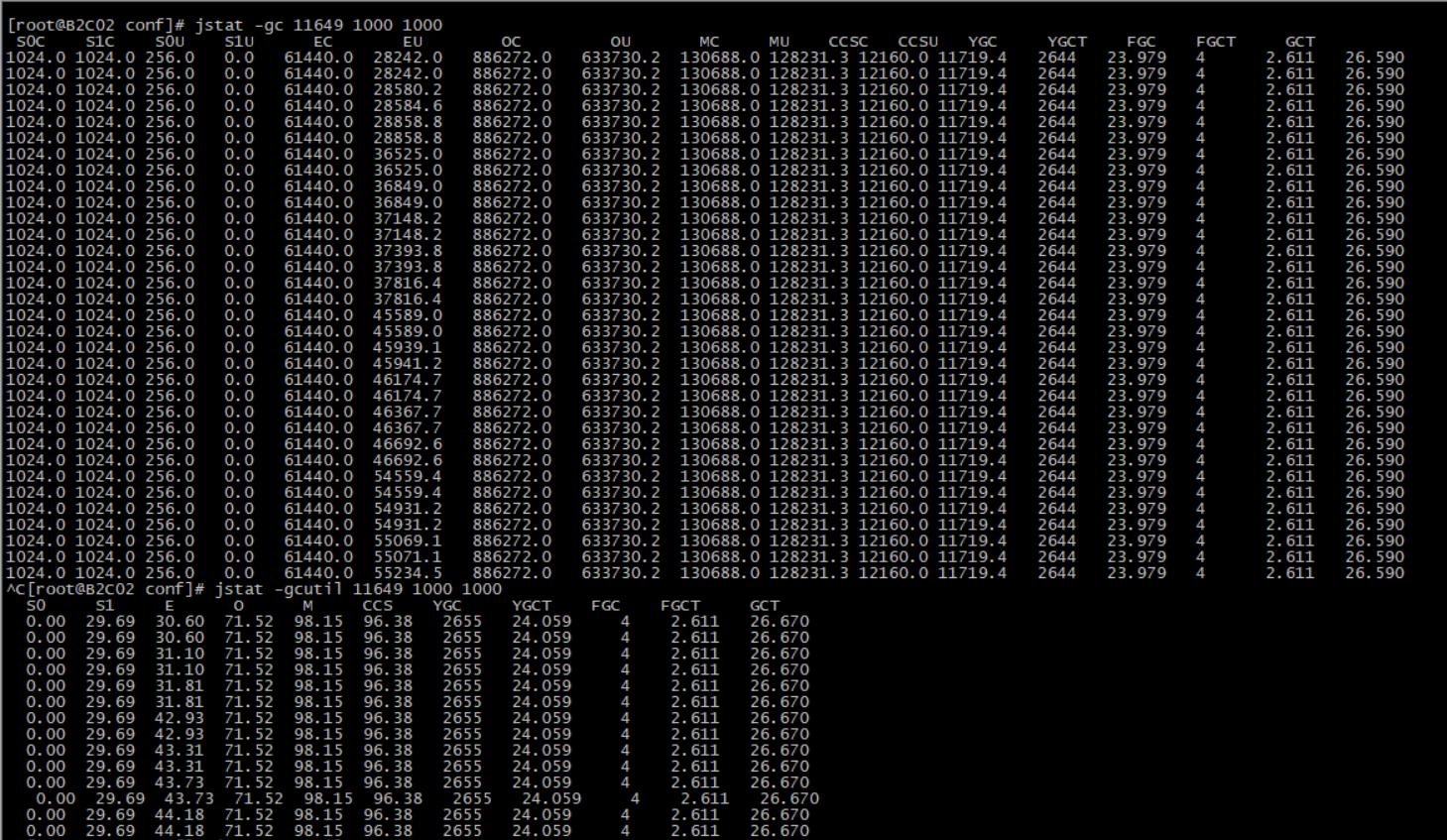
可以看出，完全使用tomcat缺省参数。

使用jmap -heap 7159，见下图



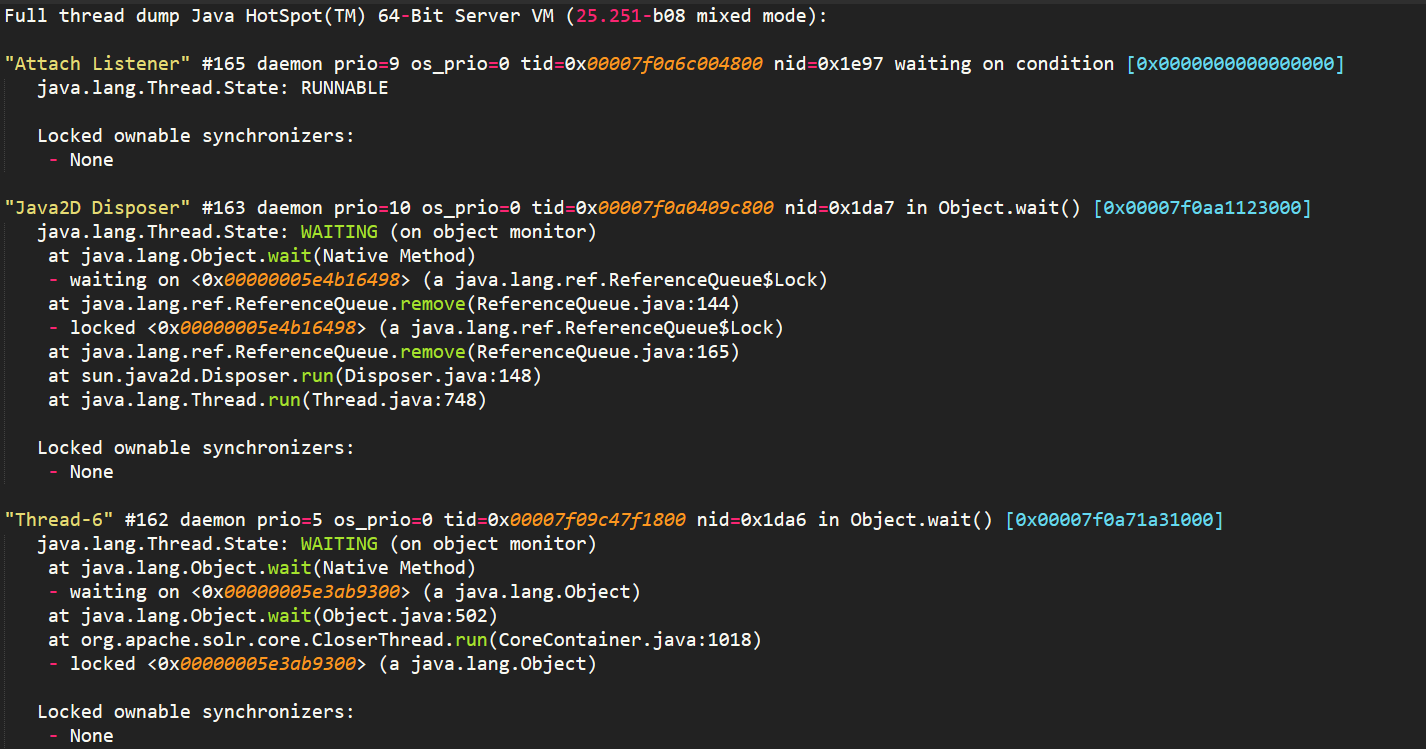
1. GC使用的是Parallel GC，这个是Java 8缺省的GC，使用8个线程是当前CPU核心数。
2. MaxHeapSize是8G，正好是系统32G内存的四分之一。
3. 可以看到NewSize 是168M，OldSize 是336M，而NewRatio为2，符合OldSzie:NewSize = 2。而且NewSize+ OldSize= 504M，是系统内存32G的64分之一。
4. SurvivorRadio是8，但是Eden区是68M，Survivor区一个是0.5M，另一个是1M。
5. 非堆的情况：MetaspaceSize（元输进区）是20.8M，CompressedClassSpaceSize是1G，因为没有设置MaxMetaspaceSize，所以MaxMetaspaceSize是无限制。

使用jstat查看gc情况。注：这是系统重启前获取的，所以PID不一样。

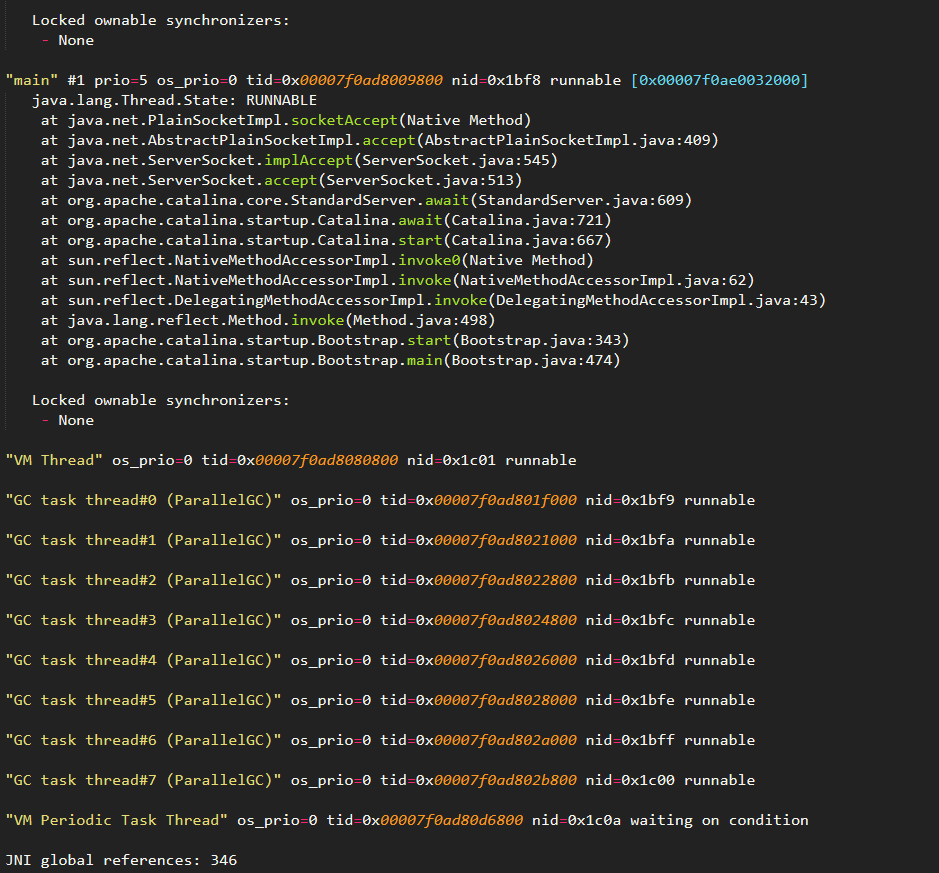


可以看见YGC进行了2655次，一共耗时24.059秒，平均一次YGC需要9ms。FGC进行了4次，一共耗时2.611秒，平均一次FGC需要653ms。

使用jstack -l查看线程情况。我把输出dump到文件1.txt中。因为中间内容比较重复，只列出开头和结尾部分。



以下是结尾部分



最后，说明一下可以进行改进的部分：

1. 启动参数（JAVA\_OPTS）中将Xms和Xmx设置成相同，比如8g。
2. 启动参数（JAVA\_OPTS）中设置-XX:MaxMetaspaceSize，比如512m。
3. 可以将GC改为G1GC。启动参数（JAVA\_OPTS）中设置-XX:+UseG1GC -XX:MaxGCPauseMillis=50
4. 增加错误日志输出，启动参数（JAVA\_OPTS）中设置-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/data/log/dump/b2b/heapdump.hprof -XX:ErrorFile=/data/log/jvmlog/b2b/hs\_err\_pid%p.log
5. 如果需要收集GC的信息，可以添加-XX:+PrintGCDetails -XX:+PrintGCDateStamps
6. 在tomcat中添加JMX配置，可以使用图形工具远程分析