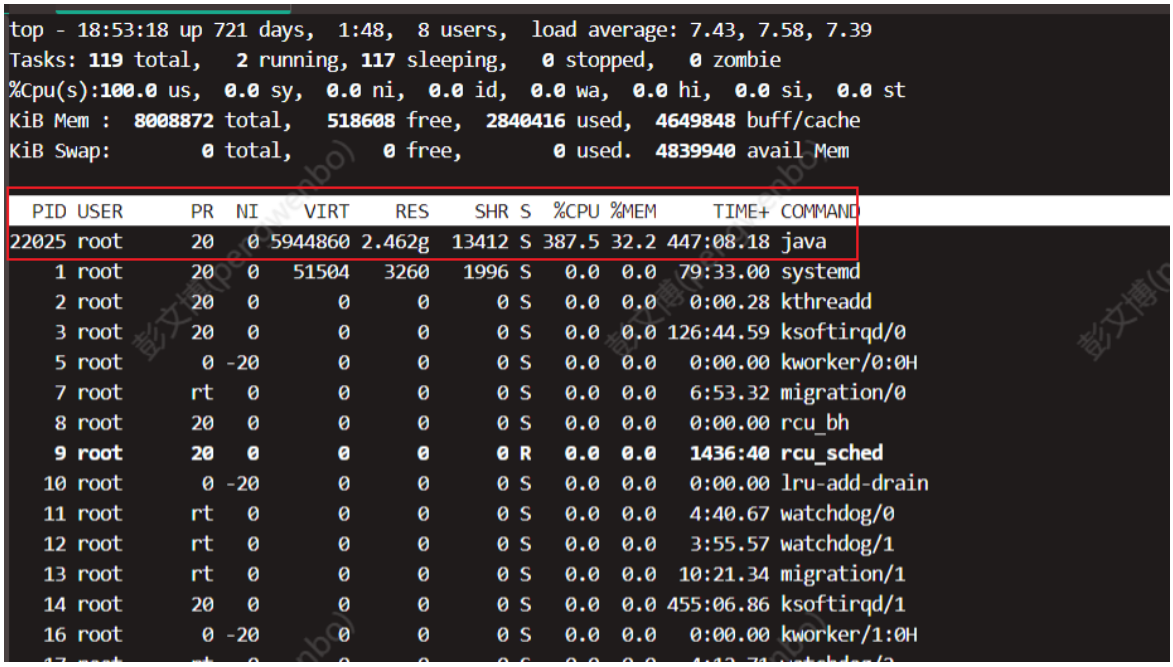
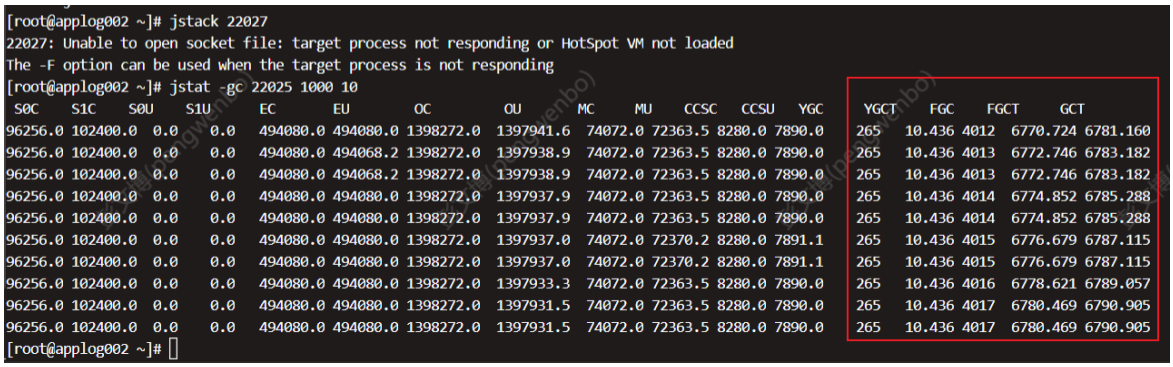
现象：收集用户操作日志的日志平台服务器CPU100%，服务宕机

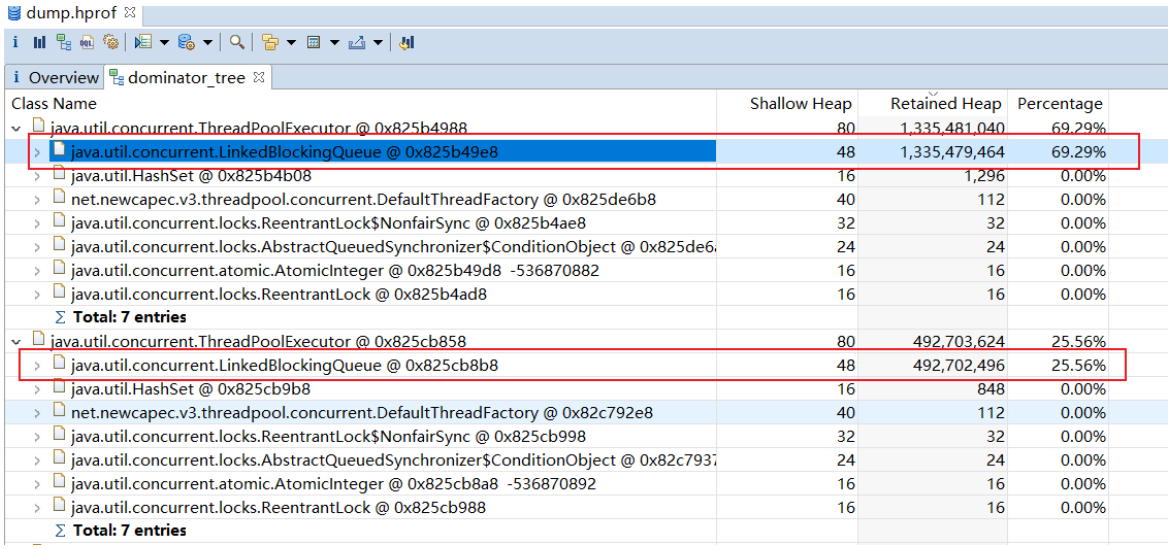


从上图可以看到pid为22025的java进程使用了387%的CPU使用率，内存占到了32%，显然是处于异常状态了。



通过jstack分析进程和jstat命令分析gc状态，结果发现上图的状态，java在一直不停的执行Full GC，执行4000多次了，很明显出现了内存泄露或大内存长时间占用内存的情况。

通过jmap命令导出堆内存快照hprof文件，并使用MAT工具分析内存占用状态，得到下面的结果：



Class Name	Shallow Heap	Retained Heap	Percentage
java.util.concurrent.ThreadPoolExecutor @ 0x825b4988	80	1,335,481,040	69.29%
java.util.concurrent.LinkedBlockingQueue @ 0x825b49e8	48	1,335,479,464	69.29%
java.util.HashSet @ 0x825b4b08	16	1,296	0.00%
net.newcapec.v3.threadpool.concurrent.DefaultThreadFactory @ 0x825de6b8	40	112	0.00%
java.util.concurrent.locks.ReentrantLock$NonfairSync @ 0x825b4ae8	32	32	0.00%
java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject @ 0x825de6	24	24	0.00%
java.util.concurrent.atomic.AtomicInteger @ 0x825b49d8 -536870882	16	16	0.00%
java.util.concurrent.locks.ReentrantLock @ 0x825b4ad8	16	16	0.00%
Σ Total: 7 entries			
java.util.concurrent.ThreadPoolExecutor @ 0x825cb858	80	492,703,624	25.56%
java.util.concurrent.LinkedBlockingQueue @ 0x825cb8b8	48	492,702,496	25.56%
java.util.HashSet @ 0x825cb9b8	16	848	0.00%
net.newcapec.v3.threadpool.concurrent.DefaultThreadFactory @ 0x82c792e8	40	112	0.00%
java.util.concurrent.locks.ReentrantLock$NonfairSync @ 0x825cb998	32	32	0.00%
java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject @ 0x82c793	24	24	0.00%
java.util.concurrent.atomic.AtomicInteger @ 0x825cb8a8 -536870892	16	16	0.00%
java.util.concurrent.locks.ReentrantLock @ 0x825cb988	16	16	0.00%
Σ Total: 7 entries			

通过类空间占用排列，发现两个线程池占用了95%的内存空间，进而分析对象结构，看到这两个线程池都使用了LinkedBlockingQueue无界队列作为缓冲空间，两个对象存储的都是发送MQ的用户行为日志。

Class Name	Shallow Heap	Retained Heap	Percentage
<Regex>	<Numeric>	<Numeric>	<Numeric>
java.util.concurrent.ThreadPoolExecutor @ 0x825b4988	80	1,335,481,040	69.29%
java.util.concurrent.LinkedBlockingQueue @ 0x825b49e8	48	1,335,479,464	69.29%
java.util.HashSet @ 0x825b4b08	16	1,296	0.00%
net.newcapec.v3.threadpool.concurrent.DefaultThreadFactory @ 0x825de6b8	40	112	0.00%
java.lang.String @ 0x825de6e0 \u53d1\u9001mq\u65e5\u5fd7 发送mq日志	24	56	0.00%
java.lang.Object @ 0x825de758	16	16	0.00%
Σ Total: 2 entries			
java.util.concurrent.locks.ReentrantLock$NonfairSync @ 0x825b4ae8	32	32	0.00%
java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject @ 0x825de6a0	24	24	0.00%
java.util.concurrent.atomic.AtomicInteger @ 0x825b49d8 -536870882	16	16	0.00%
java.util.concurrent.locks.ReentrantLock @ 0x825b4ad8	16	16	0.00%
Σ Total: 7 entries			
java.util.concurrent.ThreadPoolExecutor @ 0x825cb858	80	492,703,624	25.56%
java.util.concurrent.LinkedBlockingQueue @ 0x825cb8b8	48	492,702,496	25.56%
java.util.HashSet @ 0x825cb9b8	16	848	0.00%
net.newcapec.v3.threadpool.concurrent.DefaultThreadFactory @ 0x82c792e8	40	112	0.00%
java.lang.String @ 0x829578d0 \u6279\u91cf\u8bb0\u5f55\u70b9\u51fb\u65e5\u5fd7	24	56	0.00%
java.lang.Object @ 0x82c79310 批量点击日志	16	16	0.00%
Σ Total: 2 entries			
java.util.concurrent.locks.ReentrantLock$NonfairSync @ 0x825cb998	32	32	0.00%
java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject @ 0x82c79378	24	24	0.00%
java.util.concurrent.atomic.AtomicInteger @ 0x825cb8a8 -536870892	16	16	0.00%
java.util.concurrent.locks.ReentrantLock @ 0x825cb988	16	16	0.00%
Σ Total: 7 entries			
org.apache.catalina.loader.ParallelWebappClassLoader @ 0x80030ae0	136	11,949,704	0.62%
org.apache.tomcat.util.net.NioEndpoint @ 0x80233b28	192	8,528,008	0.44%

```
top - 18:53:18 up 721 days,  1:48,  8 users,  load average: 7.43, 7.58, 7.39
Tasks: 119 total,   2 running, 117 sleeping,   0 stopped,   0 zombie
%Cpu(s):100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  8008872 total,   518608 free,  2840416 used,  4649848 buff/cache
KiB Swap:        0 total,        0 free,        0 used.  4839940 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
22025 root      20   0 5944860 2.462g  13412 S 387.5 32.2 447:08.18 java
    1 root      20   0   51504   3260   1996 S   0.0  0.0  79:33.00 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.28 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0 126:44.59 ksoftirqd/0
    5 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:0H
    7 root      rt   0       0      0      0 S   0.0  0.0   6:53.32 migration/0
    8 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_bh
    9 root      20   0       0      0      0 R   0.0  0.0   1436:40 rcu_sched
   10 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 lru-add-drain
   11 root      rt   0       0      0      0 S   0.0  0.0   4:40.67 watchdog/0
   12 root      rt   0       0      0      0 S   0.0  0.0   3:55.57 watchdog/1
   13 root      rt   0       0      0      0 S   0.0  0.0  10:21.34 migration/1
   14 root      20   0       0      0      0 S   0.0  0.0 455:06.86 ksoftirqd/1
   16 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/1:0H
```

从上图可以看到pid为22025的java进程使用了387%的CPU使用率，内存占到了32%，显然是处于异常状态了。

```
[root@applog002 ~]# jstack 22027
22027: Unable to open socket file: target process not responding or HotSpot VM not loaded
The -F option can be used when the target process is not responding
[root@applog002 ~]# jstat -gc 22025 1000 10
 S0C    S1C    S0U    S1U      EC       EU        OC         OU       MC     MU    CCSC   CCSU   YGC     YGCT    FGC    FGCT     GCT
96256.0 102400.0  0.0    0.0   494080.0 494080.0 1398272.0  1397941.6  74072.0 72363.5 8280.0 7890.0   265   10.436 4012   6770.724 6781.160
96256.0 102400.0  0.0    0.0   494080.0 494068.2 1398272.0  1397938.9  74072.0 72363.5 8280.0 7890.0   265   10.436 4013   6772.746 6783.182
96256.0 102400.0  0.0    0.0   494080.0 494068.2 1398272.0  1397938.9  74072.0 72363.5 8280.0 7890.0   265   10.436 4013   6772.746 6783.182
96256.0 102400.0  0.0    0.0   494080.0 494080.0 1398272.0  1397937.9  74072.0 72363.5 8280.0 7890.0   265   10.436 4014   6774.852 6785.288
96256.0 102400.0  0.0    0.0   494080.0 494080.0 1398272.0  1397937.9  74072.0 72363.5 8280.0 7890.0   265   10.436 4014   6774.852 6785.288
96256.0 102400.0  0.0    0.0   494080.0 494080.0 1398272.0  1397937.0  74072.0 72370.2 8280.0 7891.1   265   10.436 4015   6776.679 6787.115
96256.0 102400.0  0.0    0.0   494080.0 494080.0 1398272.0  1397937.0  74072.0 72370.2 8280.0 7891.1   265   10.436 4015   6776.679 6787.115
96256.0 102400.0  0.0    0.0   494080.0 494080.0 1398272.0  1397933.3  74072.0 72363.5 8280.0 7890.0   265   10.436 4016   6778.621 6789.057
96256.0 102400.0  0.0    0.0   494080.0 494080.0 1398272.0  1397931.5  74072.0 72363.5 8280.0 7890.0   265   10.436 4017   6780.469 6790.905
96256.0 102400.0  0.0    0.0   494080.0 494080.0 1398272.0  1397931.5  74072.0 72363.5 8280.0 7890.0   265   10.436 4017   6780.469 6790.905
[root@applog002 ~]#
```

通过jstack分析进程和jstat命令分析gc状态，结果发现上图的状态，java在一直不停的执行Full GC，执行4000多次了，很明显出现了内存泄露或大内存长时间占用内存的情况。

通过jmap命令导出堆内存快照hprof文件，并使用MAT工具分析内存占用状态，得到下面的结果：

dump.hprof

Overview | dominator_tree

Class Name	Shallow Heap	Retained Heap	Percentage
java.util.concurrent.ThreadPoolExecutor @ 0x825b4988	80	1,335,481,040	69.29%
java.util.concurrent.LinkedBlockingQueue @ 0x825b49e8	48	1,335,479,464	69.29%
java.util.HashSet @ 0x825b4b08	16	1,296	0.00%
net.newcapec.v3.threadpool.concurrent.DefaultThreadFactory @ 0x825de6b8	40	112	0.00%
java.util.concurrent.locks.ReentrantLock$NonfairSync @ 0x825b4ae8	32	32	0.00%
java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject @ 0x825de6	24	24	0.00%
java.util.concurrent.atomic.AtomicInteger @ 0x825b49d8 -536870882	16	16	0.00%
java.util.concurrent.locks.ReentrantLock @ 0x825b4ad8	16	16	0.00%
Σ Total: 7 entries			
java.util.concurrent.ThreadPoolExecutor @ 0x825cb858	80	492,703,624	25.56%
java.util.concurrent.LinkedBlockingQueue @ 0x825cb8b8	48	492,702,496	25.56%
java.util.HashSet @ 0x825cb9b8	16	848	0.00%
net.newcapec.v3.threadpool.concurrent.DefaultThreadFactory @ 0x82c792e8	40	112	0.00%
java.util.concurrent.locks.ReentrantLock$NonfairSync @ 0x825cb998	32	32	0.00%
java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject @ 0x82c793	24	24	0.00%
java.util.concurrent.atomic.AtomicInteger @ 0x825cb8a8 -536870892	16	16	0.00%
java.util.concurrent.locks.ReentrantLock @ 0x825cb988	16	16	0.00%
Σ Total: 7 entries			

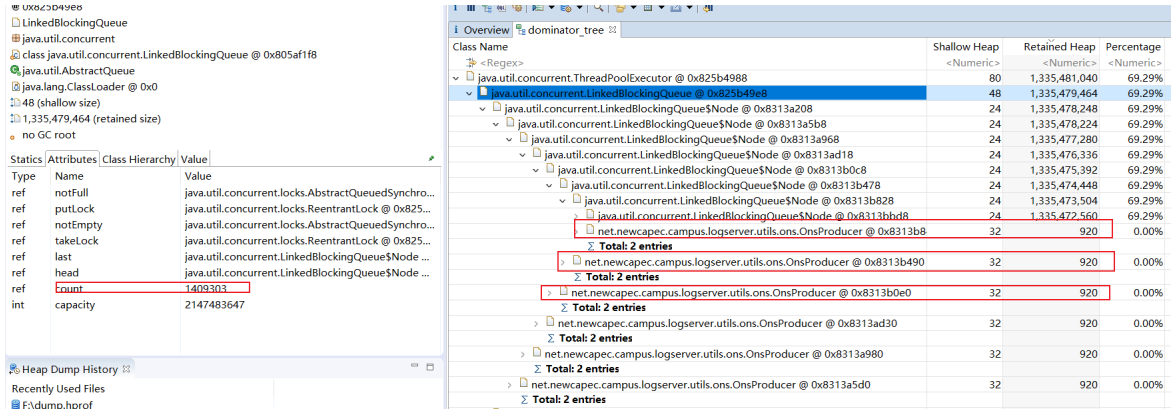
通过类空间占用排列，发现两个线程池占用了95%的内存空间，进而分析对象结构，看到这两个线程池都使用了LinkedBlockingQueue无界队列作为缓冲空间，两个对象存储的都是发送MQ的用户行为日志。

Class Name	Shallow Heap	Retained Heap	Percentage
<Regex>	<Numeric>	<Numeric>	<Numeric>
java.util.concurrent.ThreadPoolExecutor @ 0x825b4988	80	1,335,481,040	69.29%
java.util.concurrent.LinkedBlockingQueue @ 0x825b49e8	48	1,335,479,464	69.29%
java.util.HashSet @ 0x825b4b08	16	1,296	0.00%
net.newcapec.v3.threadpool.concurrent.DefaultThreadFactory @ 0x825de6b8	40	112	0.00%
java.lang.String @ 0x825de6e0 \u53d1\u9001mq\u65e5\u5fd7 发送mq日志	24	56	0.00%
java.lang.Object @ 0x825de758	16	16	0.00%
Σ Total: 2 entries			
java.util.concurrent.locks.ReentrantLock$NonfairSync @ 0x825b4ae8	32	32	0.00%
java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject @ 0x825de6a0	24	24	0.00%
java.util.concurrent.atomic.AtomicInteger @ 0x825b49d8 -536870882	16	16	0.00%
java.util.concurrent.locks.ReentrantLock @ 0x825b4ad8	16	16	0.00%
Σ Total: 7 entries			
java.util.concurrent.ThreadPoolExecutor @ 0x825cb858	80	492,703,624	25.56%
java.util.concurrent.LinkedBlockingQueue @ 0x825cb8b8	48	492,702,496	25.56%
java.util.HashSet @ 0x825cb9b8	16	848	0.00%
net.newcapec.v3.threadpool.concurrent.DefaultThreadFactory @ 0x82c792e8	40	112	0.00%
java.lang.String @ 0x829578d0 \u6279\u91cf\u8bb0\u5f55\u70b9\u51fb\u65e5\u5fd7	24	56	0.00%
java.lang.Object @ 0x82c79310 批量点击日志	16	16	0.00%
Σ Total: 2 entries			
java.util.concurrent.locks.ReentrantLock$NonfairSync @ 0x825cb998	32	32	0.00%
java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject @ 0x82c79378	24	24	0.00%
java.util.concurrent.atomic.AtomicInteger @ 0x825cb8a8 -536870892	16	16	0.00%
java.util.concurrent.locks.ReentrantLock @ 0x825cb988	16	16	0.00%
Σ Total: 7 entries			
org.apache.catalina.loader.ParallelWebappClassLoader @ 0x80030ae0	136	11,949,704	0.62%
org.apache.tomcat.util.net.NioEndpoint @ 0x80233b28	192	8,528,008	0.44%

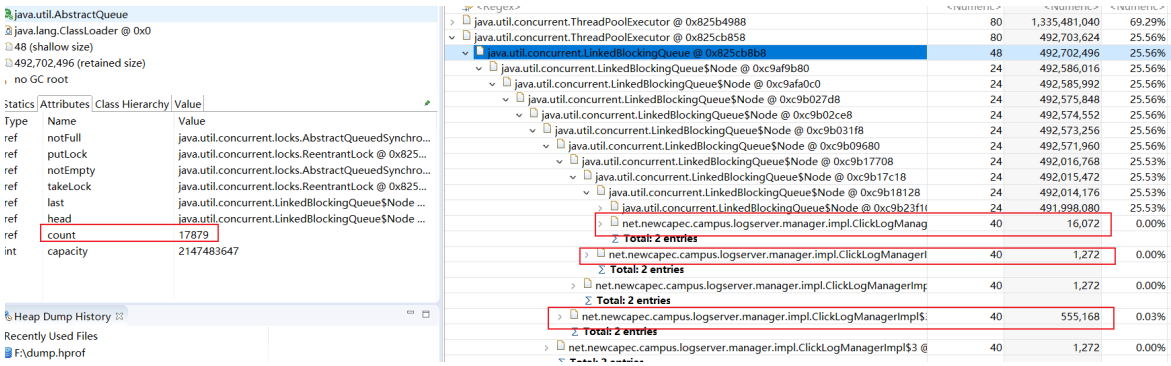
结合之前分析开放平台数据库性能不稳定问题，此次问题很有可能是近些时段内用户数量增多导致，产生了足够过多的用户日志，但是日志服务处理线程池有限，大量的日志信息堆积在等待队列当中，最终占完了全部应用的内存空间导致进程崩溃。

问题处理方式：

进一步分析发送MQ日志线程队列数据，每个发送的OnsProducer对象大小是920B，可以看成1K，队列中一共堆积了140万的待处理对象



而批量点击日志线程池的处理队列中一共堆积了1万7的ClickLogManagerImpl对象数据，每个对象大小从1.2K到10k~50多K不等



- 增加线程池处理线程数，使用有界队列作为等待队列(发送MQ日志线程池设置100万，批量点击日志线程池设置为1万)，使非核心线程发挥作用(最大线程数扩大一倍)，必要时任务过多可以抛弃溢出数据或进行持久化补偿操作

- 增加大应用内存空间到4g，或者增加机器提高负载能力

- 增加服务器内存监控，达到应用内存上限的80%时进行告警，手动调整服务器负载能力