先说一下最终方式【不知道会不会有效...】

修改JDK8的小版本:

https://blog.csdn.net/g412086027/article/details/106115777/

【曾猜测: org.hibernate.internal.SessionFactoryImpl 导致的内存泄漏】

须知:

top: https://segmentfault.com/a/1190000008125059

VIRT: 进程所使用的虚拟内存大小

RES: 系统为虚拟内存分配的物理内存大小【存在SHR】

SHR: 共享内存

【Linux查看内存: pmap -x \${pid} | sort -k 3 -n -r 】

sort的-r选项【sort默认升序,降序使用 -r】

sort的-n选项【避免10 < 2 的情况】

sort的-k选项【存在多列时指定排序列,通常与-t选项(指定分隔符)同用】

事情是这个样子的: 【Linux系统内存飙升, 但是就是<mark>找不到泄漏</mark>的地方】

提供已知情况 - 【GC】:

```
FGCT
0.00
      43.82
              83.05
97.33
                             77.12
                                                    168.660
168.660
                                                                     19.255
19.255
                      31.32
                                     74.67
                                             3990
                             77.12
0.00
       43.82
                      31.32
                                              3990
              3.23
7.90
19.42
        0.00
                             77.12
                                             3991
                             77.12
77.12
                      31.33
                                             3991
        0.00
                                                    168.711
                      31.33
                                             3991
        0.00
              23.06
        0.00
                     31.33
                             77.12
                                             3991
                                                    168.711
                                                                              187.966
              33.85
                      31.33
                             77.12
        0.00
                                              3991
        0.00
              39.04
                      31.33
                             77.12
                                              3991
        0.00
              41.14
                      31.33
                             77.12
                                              3991
        0.00
              55.80
                      31.33
                             77.12
                                              3991
              78.28
79.08
81.52
       0.00
                             77.12
                                             3991
                      31.33
                                                    168.711
                                                                              187.966
                             77.12
77.12
       0.00
                      31.33
                                             3991
                                                    168.711
                                                                                  966
       0.00
                      31.33
                                     74.67
                                             3991
                                                    168.711
                                                                     19.255
                                                                              187.966
                                                                              187.966
        0.00
              89.36
                      31.33
                                              3991
                                                                     19.255
```

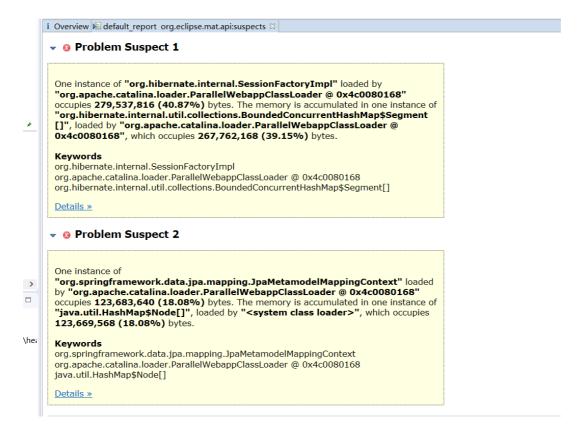
提供已知情况 - 【instances】:

```
0.00 53.77 32.27 31.34 77.12 74.67 0.00 53.77 37.48 31.34 77.12 74.67
                                                   3992 168.762
                                                                              19.255 188.016
`C[root@cnbj01vl00213 logs]# jmap -histo:live 32519 | head -10
          #instances
                                #bytes class name
                            367955192 [C
  1:
             1649812
                                          java.util.HashMap$Node
[Ljava.util.HashMap$Node;
             4380036
                             140161152
   2:
                             42258552
              133510
   3:
                              39490464 java.lang.String
38093664 org.hibernate.hql.internal.ast.tree.Node
             1645436
   4:
              793618
   5:
              500021
                              30944976 [Ljava.lang.Object;
   6:
             1653857
                              26461712
                                          java.lang.Integer
[root@cnbj01vl00213 logs]#
```

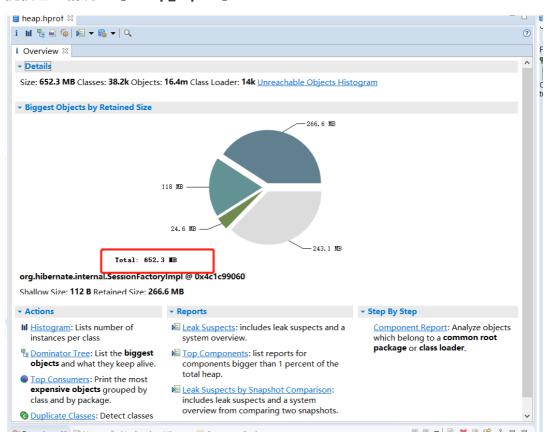
提供已知情况 - 【heap】:

```
1 root@zoms1 × 2 root@zoms2 × +
   NewRatio
                            = 2
   SurvivorRatio
                            = 8
                            = 21807104 (20.796875MB)
   MetaspaceSize
   CompressedClassSpaceSize = 1073741824 (1024.0MB)
                        = 17592186044415 MB
   MaxMetaspaceSize
   G1HeapRegionSize
                          = 0 (0.0MB)
Heap Usage:
PS Young Generation
Eden Space:
   capacity = 3637510144 (3469.0MB)
          = 1368281152 (1304.8945922851562MB)
   used
           = 2269228992 (2164.1054077148438MB)
   37.61587178683068% used
From Space:
   capacity = 297271296 (283.5MB)
  used
          = 297093640 (283.33057403564453MB)
   free
          = 177656 (0.16942596435546875MB)
   99.94023775507743% used
To Space:
   capacity = 360185856 (343.5MB)
          = 0 (0.0MB)
   used
   free
           = 360185856 (343.5MB)
   0.0% used
PS Old Generation
   capacity = 8589934592 (8192.0MB)
           = 992431944 (946.4568557739258MB)
           = 7597502648 (7245.543144226074MB)
   11.553428415209055% used
```

提供已知情况 - 【dump_report】:



提供已知情况 - 【dump_report】:



以上五幅图便是已知条件

Q: 这个dump是系统正常运行时抓下来的快照还是error时产生的?

A: 系统正常运行, 但是内存占用率达到了快90%;

【SQ】: 因为dump文件一共才1G左右, 所以问了一下,

后来咨询同事说:【oom时dump会默认筛掉一些无关的信息,可能会导致dump文件比jvm的内存小 - 本人颇为不信】

Q: 系统jvm给多少?

A: liunx是16G JVM给了12G; 其他没调都是默认的。

再怎么筛,也不会把12G*90%给筛成了1G吧...

```
java.specification.vendor = Oracle Corporation
awt.toolkit = sun.awt.X11.XToolkit
java.vm.info = mixed mode
java.version = 1.8.0 281
java.version = 1.8.0 281
java.version = 1.8.0 281
java.sext.dirs = /app/jdk/jdk1.8.0 281/jre/lib/ext:/usr/java/packages/lib/ext
sun.boot.class.path = /app/jdk/jdk1.8.0 281/jre/lib/resources.jar:/app/jdk/jdk1.8.0 281/jre/lib/rt.jar:/app/jdk/jdk1.8.0 281/jre/lib/sun.rsaign.jar:/app/jdk/jdk1.8.0 281/jre/lib/jsse.jar:/app/jdk/jdk1.8.0 281/jre/lib/jce.jar:/app/jdk/jdk1.8.0 281/jre/lib/jce.jar:/app/jdk/jdk1.8.0 281/jre/classes
server.loader =
java.vendor = Oracle Corporation
catalina.base = /app/tomcat/tomcat
file.separator = /
java.security.egd = file:/dev/./urandom:-Djava.library.path=/app/tomcat/tomcat/lib:-Djava.awt.headless=true
java.vendor.urt.bug = http://bugreport.sun.com/bugreport/
common.loader = "${catalina.base}/lib", "${catalina.base}/lib/*.jar", "${catalina.home}/lib", "${catalina.home}/lib/*.jar"
sun.font.fontmanager = sun.awt.X11FontManager
sun.cpu.endian = little
package.access = sun.,org.apache.catalina.,org.apache.coyote.,org.apache.jasper.,org.apache.tomcat.
sun.cpu.isalist =

VM Flags:
Non-default VM flags: -XX:CICompilerCount=12 -XX:InitialHeapSize=12884901888 -XX:MaxHeapSize=12884901888 -XX:MaxNewSize=429
4967296 -XX:MinHeapDeltaBytes=524288 -XX:NewSize=4294967296 -XX:OldSize=8589934592 -XX:+UseCompressedClassPointers -XX:+Use
CompressedOops -XX:+UseParallelGC
Command line: -Ocatalina.home=/app/tomcat/tomcat -Dcatalina.base=/app/tomcat/tomcat -Djava.io.tmpdir=/var/tmp -Djava.security.ged=file:/dev/./urandom:-Djava.library.path=/app/tomcat/tomcat/tomcat/tomcat/tomcat -Dcommons.daemon.process.parent=3251
8 -Dcommons.daemon.version=1.1.0 abort
```

虽然不信,但是还是按照.hprof文件分析了【若按照dump文件分析,问题应是: JPA使用in时的Cache】

| 🏮 org.hibernate.internal.SessionFactoryImpl (0x6869a) |
|--|
| ▼ ===== 267 MB (95.8%) queryPlanCache ② org.hibernate.engine.query.spi.QueryPlanCache (0x6d120) |
| 🔻 💳 267 MB (95.8%) queryPlanCache 🔷 org.hibernate.internal.util.collections.BoundedConcurrentHashMap (0x6d121) |
| 🔻 ===== 267 MB (95.8%) segments 🤤 org.hibernate.internal.util.collections.BoundedConcurrentHashMap\$Segment[] (0xd073 |
| 🔻 🛮 25,137 kB (9.0%) element 🛟 org.hibernate.internal.util.collections.BoundedConcurrentHashMap\$Segment (0x6d13c) |
| 🔻 7,468 kB (2.7%) [transitive reference] 🛟 org.hibernate.internal.util.collections.BoundedConcurrentHashMap\$LIRSHashE |
| ▶ 17,315 kB (2.6%) value (declared by org.hibernate.internal.util.collecti <u>ons.BoundedConcurrentHashMap\$Hash</u> Entry) 😂 org |
| 153 kB (0.1%) [transitive reference] (java.lang.String (0xab4980) ["from InvoiceDetail where invoiceId in 71, ?2, ?3, ?4, ? |
| → Another 1 instances with a total retained size of 32 bytes and a maximum single retained size of 32 bytes |
| 🔻 5,864 kB (2.1%) [transitive reference] 🛟 org.hibernate.internal.util.collections.BoundedConcurrentHashMap\$LIRSHashE |
| 🕨 I 5,746 kB (2.1%) value (declared by org.hibernate.internal.util.collections.BoundedConcurrentHashMap\$HashEntry) 🤤 org |
| |

https://stackoverflow.com/questions/31557076/spring-hibernate-query-plan-cache-memory-usage

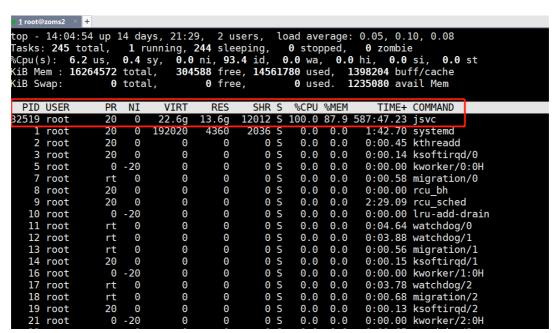
但是缓存一共才占用267M, 信这个我就是傻子!!!

------dump分析不出来,告一段落,换个思路------

https://giraffetree.me/2019/12/03/memory-leak/

Q:上午说的90%内存使用,指的是Linux?

A: JVM消耗掉Linux的内存:



辉哥说可能分配的ivm内存这么多久默认都使用了。【辉哥说的应该不对...】

其言外之意是【初始堆内存与最大堆内存设置完后,jvm就甭管用不用,都吃掉内存经测-并不是】

通过 arthas 定位【堆内内存、code区域 、 使用 unsafe.allocateMemory 和 **DirectByteBuffer **申请的堆外内存】

可见一共使用的内存也只有【2G左右】

| 1 root@zoms2 × + | | | | | | | () |
|---------------------------|------------|--------------|------------------|---------------|-----------|-----------|------------|
| ID NAME | GROUP | PRIORIT | / STATE %CPU | DELTA TIM | TIME | INTERRUPT | DAEMON |
| 200 http-nio-8080-exec-15 | main | 5 | TIMED WAI 0.0 | 0.000 | 49:53.300 | false | true |
| 170 http-nio-8080-exec-7 | main | 5 | TIMED WAI 0.0 | 0.000 | 21:20.978 | false | true |
| 169 http-nio-8080-exec-6 | main | 5 | TIMED_WAI 0.0 | 0.000 | 21:18.845 | false | true |
| 201 http-nio-8080-exec-16 | main | 5 | TIMED_WAI 0.0 | 0.000 | 21:16.158 | false | true |
| 198 http-nio-8080-exec-13 | main | 5 | TIMED WAI 0.0 | 0.000 | 21:0.563 | false | true |
| 167 http-nio-8080-exec-4 | main | 5 | TIMED WAI 0.0 | 0.000 | 20:57.264 | false | true |
| 197 http-nio-8080-exec-12 | main | 5 | RUNNABLE 0.0 | 0.000 | 20:41.980 | false | true |
| 171 http-nio-8080-exec-8 | main | 5 | TIMED_WAI 0.0 | 0.000 | 20:37.522 | false | true |
| 166 http-nio-8080-exec-3 | main | 5 | TIMED WAI 0.0 | 0.000 | 20:24.783 | false | true |
| 168 http-nio-8080-exec-5 | main | 5 | TIMED WAI 0.0 | 0.000 | 20:18.973 | false | true |
| 199 http-nio-8080-exec-14 | main | 5 | TIMED WAI 0.0 | 0.000 | 20:17.896 | false | true |
| 164 http-nio-8080-exec-1 | main | 5 | TIMED_WAI 0.0 | 0.000 | 20:1.884 | false | true |
| Memory us | ed total | max usage | GC | | | | |
| heap 24 | 00M 11870M | 11870M 20.22 | gc.ps_scavenge.c | ount | 4315 | | |
| ps_eden_space 62 | 6M 3247M | 3247M 19.29 | | ime(ms) | 181508 | | |
| ps_survivor_space 11 | 1M 430M | 430M 25.91 | gc.ps_marksweep. | count | 32 | | |
| ps_old_gen 16 | 62M 8192M | 8192M 20.29 | gc.ps_marksweep. | time(ms) | 22789 | | |
| nonheap 41 | | -1 76.82 | | | | | |
| code_cache 14 | 4M 191M | 240M 60.22 | 6 | | | | |
| | 1M 310M | -1 77.71 | ő | | | | |
| compressed_class_space 27 | | 1024M 2.73% | | | | | |
| direct 31 | | - 100.0 | 9% | | | | |
| mapped 0K | 0K | - 0.00% | | | | | |
| Runtime | | | | | | | |
| os.name | | | Linux | | | | |
| os.version | | | 3.10.0-957.el7.x | 86_64 | | | |
| java.version | | | 1.8.0_281 | | | | |
| java.home | | | /app/jdk/jdk1.8. | 0_281/jre | | | |
| systemload.average | | | 0.02 | | | | |
| processors | | | 16 | | | | |
| timestamp/uptime | | | Tue Feb 23 14:05 | :45 CST 2021/ | 399150s | | |

这就古怪了...

jsvc【即: jvm】占用了内存: 16G * 89% arthas分析java层面的占用内存只有2G 剩下的内存用在哪儿了? 只可能是JNI层、或者再向下C层