

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/q412086027/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，但是就是找不到泄漏的地方】

提供已知情况 - 【GC】：

```
root@cnbj01vl00213 logs]# jstat -gcutil 32519 2000 20
S0    S1    E      O      M      CCS      YGC      YGCT      FGC      FGCT      GCT
0.00  43.82  83.05  31.32  77.12  74.67    3990    168.660    27    19.255    187.915
0.00  43.82  97.33  31.32  77.12  74.67    3990    168.660    27    19.255    187.915
37.44  0.00    3.23  31.33  77.12  74.67    3991    168.711    27    19.255    187.966
37.44  0.00    7.90  31.33  77.12  74.67    3991    168.711    27    19.255    187.966
37.44  0.00   19.42  31.33  77.12  74.67    3991    168.711    27    19.255    187.966
37.44  0.00   23.06  31.33  77.12  74.67    3991    168.711    27    19.255    187.966
37.44  0.00   33.85  31.33  77.12  74.67    3991    168.711    27    19.255    187.966
37.44  0.00   39.04  31.33  77.12  74.67    3991    168.711    27    19.255    187.966
37.44  0.00   41.14  31.33  77.12  74.67    3991    168.711    27    19.255    187.966
37.44  0.00   55.80  31.33  77.12  74.67    3991    168.711    27    19.255    187.966
37.44  0.00   78.28  31.33  77.12  74.67    3991    168.711    27    19.255    187.966
37.44  0.00   79.08  31.33  77.12  74.67    3991    168.711    27    19.255    187.966
37.44  0.00   81.52  31.33  77.12  74.67    3991    168.711    27    19.255    187.966
37.44  0.00   89.36  31.33  77.12  74.67    3991    168.711    27    19.255    187.966
37.44  0.00   98.11  31.33  77.12  74.67    3991    168.711    27    19.255    187.966
```

提供已知情况 - 【instances】：

```

0.00 53.77 32.27 31.34 77.12 74.67 3992 168.762 27 19.255 188.016
0.00 53.77 37.48 31.34 77.12 74.67 3992 168.762 27 19.255 188.016
^C[root@cnbj01vl00213 logs]# jmap -histo:live 32519 | head -10

num      #instances      #bytes  class name
-----
 1:       1649812      367955192  [C
 2:       4380036      140161152  java.util.HashMap$Node
 3:        133510       42258552  [Ljava.util.HashMap$Node;
 4:       1645436       39490464  java.lang.String
 5:        793618       38093664  org.hibernate.hql.internal.ast.tree.Node
 6:        500021       30944976  [Ljava.lang.Object;
 7:       1653857       26461712  java.lang.Integer
[root@cnbj01vl00213 logs]# █

```

提供已知情况 - 【heap】：

```

1 root@zoms1 x 2 root@zoms2 x +
NewRatio                = 2
SurvivorRatio            = 8
MetaspaceSize            = 21807104 (20.796875MB)
CompressedClassSpaceSize = 1073741824 (1024.0MB)
MaxMetaspaceSize        = 17592186044415 MB
G1HeapRegionSize        = 0 (0.0MB)

Heap Usage:
PS Young Generation
Eden Space:
  capacity = 3637510144 (3469.0MB)
  used     = 1368281152 (1304.8945922851562MB)
  free     = 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)
  used     = 0 (0.0MB)
  free     = 360185856 (343.5MB)
  0.0% used
PS Old Generation
  capacity = 8589934592 (8192.0MB)
  used     = 992431944 (946.4568557739258MB)
  free     = 7597502648 (7245.543144226074MB)
  11.553428415209055% used
█

```

提供已知情况 - 【dump_report】：

Overview default_report org.eclipse.mat.api:suspects

Problem Suspect 1

One instance of **"org.hibernate.internal.SessionFactoryImpl"** loaded by **"org.apache.catalina.loader.ParallelWebappClassLoader @ 0x4c0080168"** occupies **279,537,816 (40.87%)** bytes. The memory is accumulated in one instance of **"org.hibernate.internal.util.collections.BoundedConcurrentHashMap\$Segment []"**, loaded by **"org.apache.catalina.loader.ParallelWebappClassLoader @ 0x4c0080168"**, which occupies **267,762,168 (39.15%)** bytes.

Keywords
 org.hibernate.internal.SessionFactoryImpl
 org.apache.catalina.loader.ParallelWebappClassLoader @ 0x4c0080168
 org.hibernate.internal.util.collections.BoundedConcurrentHashMap\$Segment []

[Details »](#)

Problem Suspect 2

One instance of **"org.springframework.data.jpa.mapping.JpaMetamodelMappingContext"** loaded by **"org.apache.catalina.loader.ParallelWebappClassLoader @ 0x4c0080168"** occupies **123,683,640 (18.08%)** bytes. The memory is accumulated in one instance of **"java.util.HashMap\$Node[]"**, loaded by **"<system class loader>"**, which occupies **123,669,568 (18.08%)** bytes.

Keywords
 org.springframework.data.jpa.mapping.JpaMetamodelMappingContext
 org.apache.catalina.loader.ParallelWebappClassLoader @ 0x4c0080168
 java.util.HashMap\$Node []

[Details »](#)

提供已知情况 - 【dump_report】：

heap.hprot

Overview

Size: **652.3 MB** Classes: **38.2k** Objects: **16.4m** Class Loader: **14k** [Unreachable Objects Histogram](#)

Biggest Objects by Retained Size

Category	Size (MB)
org.hibernate.internal.SessionFactoryImpl @ 0x4c1c99060	266.6
Other	243.1
Other	118
Other	24.6
Total	652.3

org.hibernate.internal.SessionFactoryImpl @ 0x4c1c99060
 Shallow Size: **112 B** Retained Size: **266.6 MB**

Actions

- Histogram:** Lists number of instances per class
- Dominator Tree:** List the **biggest objects** and what they keep alive.
- Top Consumers:** Print the most **expensive objects** grouped by class and by package.
- Duplicate Classes:** Detect classes

Reports

- Leak Suspects:** includes leak suspects and a system overview.
- Top Components:** list reports for components bigger than 1 percent of the total heap.
- Leak Suspects by Snapshot Comparison:** includes leak suspects and a system overview from comparing two snapshots.

Step By Step

- Component Report:** Analyze objects which belong to a **common root package** or **class loader**.

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：linux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.ext.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/sunrsasign.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/charsets.jar:/app/jdk/jdk1.8.0_281/jre/lib/jfr.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.url.bug = http://bugreport.sun.com/bugreport/
common.loader = "${catalina.base}/lib","${catalina.base}/lib/*.jar","${catalina.home}/lib","${catalina.home}/lib/*.jar"
sun.io.unicode.encoding = UnicodeLittle
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=4294967296 -XX:MinHeapDeltaBytes=524288 -XX:NewSize=4294967296 -XX:OldSize=8589934592 -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseParallelGC
Command line: -Dcatalina.home=/app/tomcat/tomcat -Dcatalina.base=/app/tomcat/tomcat -Djava.io.tmpdir=/var/tmp -Djava.security.egd=file:/dev/./urandom:-Djava.library.path=/app/tomcat/tomcat/lib:-Djava.awt.headless=true -Xms12288M -Xmx12288M -Dfile.encoding=UTF-8 -Dsun.zip.disableMemoryMapping=true -Dcommons.daemon.process.id=32519 -Dcommons.daemon.process.parent=32518 -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 → 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)
        └─ 17,468 kB (2.7%) [transitive reference] → org.hibernate.internal.util.collections.BoundedConcurrentHashMap$LIRSHashE
          └─ 17,315 kB (2.6%) value (declared by org.hibernate.internal.util.collections.BoundedConcurrentHashMap$HashEntry) → org
            └─ 153 kB (0.1%) [transitive reference] → java.lang.String (0xab4980) "from InvoiceDetail where invoiceId in (?, ?, ?, ?, ?
              └─ 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$LIRSHashE
          └─ 5,746 kB (2.1%) value (declared by org.hibernate.internal.util.collections.BoundedConcurrentHashMap$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:

```
1 root@zoms2 x +
top - 14:04:54 up 14 days, 21:29, 2 users, load average: 0.05, 0.10, 0.08
Tasks: 245 total, 1 running, 244 sleeping, 0 stopped, 0 zombie
%Cpu(s): 6.2 us, 0.4 sy, 0.0 ni, 93.4 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 16264572 total, 304588 free, 14561780 used, 1398204 buff/cache
KiB Swap: 0 total, 0 free, 0 used. 1235080 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
32519	root	20	0	22.6g	13.6g	12012	S	100.0	87.9	587:47.23	jsvc
1	root	20	0	192020	4360	2036	S	0.0	0.0	1:42.70	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.45	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:00.14	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	0:00.58	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	S	0.0	0.0	2:29.09	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	0:04.64	watchdog/0
12	root	rt	0	0	0	0	S	0.0	0.0	0:03.88	watchdog/1
13	root	rt	0	0	0	0	S	0.0	0.0	0:00.56	migration/1
14	root	20	0	0	0	0	S	0.0	0.0	0:00.15	ksoftirqd/1
16	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/1:0H
17	root	rt	0	0	0	0	S	0.0	0.0	0:03.78	watchdog/2
18	root	rt	0	0	0	0	S	0.0	0.0	0:00.68	migration/2
19	root	20	0	0	0	0	S	0.0	0.0	0:00.13	ksoftirqd/2
21	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/2:0H

辉哥说可能分配的jvm内存这么多默认都使用了。【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 x +
ID NAME GROUP PRIORITY 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	used	total	max	usage	GC
heap	2400M	11870M	11870M	20.22%	gc.ps_scavenge.count 4315
ps eden space	626M	3247M	3247M	19.29%	gc.ps_scavenge.time(ms) 181508
ps survivor space	111M	430M	430M	25.91%	gc.ps_marksweep.count 32
ps old gen	1662M	8192M	8192M	20.29%	gc.ps_marksweep.time(ms) 22789
nonheap	413M	538M	-1	76.82%	
code cache	144M	191M	240M	60.22%	
metaspace	241M	310M	-1	77.71%	
compressed_class_space	27M	37M	1024M	2.73%	
direct	310K	310K	-	100.00%	
mapped	0K	0K	-	0.00%	

```
Runtime
os.name Linux
os.version 3.10.0-957.el7.x86_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