

#记一次线上内存报警排查过程

作者：樊春帅（神帅）

创作日期：2019-08-14

专栏地址：[【稳定大于一切】](#)

PDF 格式：[记一次线上内存报警排查过程](#)

今天风和日丽，刚到公司，看看博客，微信&钉钉消息，，，突然发现报警群里有很多报警说16.28的内存不够，报警信息如下：

告警地址: x.x.16.28
监控取值: 869.46 MB
告警等级: Warning
告警信息: x.x.16.28 内存剩余小于900M
告警时间: 2019.10.31-09:50:23
持续时间: 1h 0m

开始时间大概是从昨天晚上11点多开始的，而且持续到今天上午10点多，事出有因必有妖，下面看一下排查思路和排查过程。

####1.查一下16.28的内存使用情况

```
Welcome to aliyun Elastic Compute Service!

[readonly@-16-28 ~]$ free -m
              total        used         free       shared    buffers     cached
Mem:           15951        15776           174           0           54          660
-/+ buffers/cache:        15061          889
Swap:              0              0              0
[readonly@-16-28 ~]$ top
```

####2.排查最近是否有新上线服务，导致内存紧张

rpcservice list, ps -ef | tomcat 两个命令发现业务服务有7个，进程存活时间较长，不太可能，同时根据另一台16.29机器的服务部署情况也验证了没有新上线服务。

####3.排查是否有java服务在持续FGC

使用top命令查一下，发现9个java服务，7个业务服务，2个日志进程服务。使用 jstat -gcutil pid 2000命令——排查，发现GC情况正常，没有服务有持续的YGC，FGC情况存在。

####4.排查异常占用内存的java服务

由于有7个业务服务，直觉告诉我dwf服务应该比rpc服务占用的内存少
这一步走错了两个方向

1. 以为web服务占用内存较大，比RPC服务还高，但是发现不是
2. 以为其中一个日志进程服务（flume）占用内存较大，发现另一台16.29的日志进程服务占用的内存跟出问题的这一台机器是一样的
两步走错，浪费了一些时间~~~~

####5.top命令对比16.28/16.29两台服务器

发现其中肯定有同一个java进程占用的内存比另一个java进程占用的内存高。

看下面的图：

```
[readonly@16-28 ~]$ top
top - 10:12:24 up 455 days, 15:33, 3 users, load average: 1.34, 0.92, 0.94
Tasks: 157 total, 1 running, 156 sleeping, 0 stopped, 0 zombie
Cpu(s): 2.4%us, 1.5%sy, 0.0%ni, 96.1%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 16334084k total, 16142592k used, 191492k free, 28300k buffers
Swap: 0k total, 0k used, 0k free, 671784k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+  COMMAND
 28591 work       20   0 6755m 3.3g 12m S  0.7  21.0 405:16.78 java
 35205 work       20   0 7311m 3.1g 13m S  2.7  19.7 730:38.92 java
 13021 work       20   0 5953m 2.3g 13m S  5.3  15.0 930:10.78 java
 28895 work       20   0 5714m 1.9g 12m S  1.0  12.3 434:53.88 java
 1145  work       20   0 5282m 1.6g 13m S  1.3  10.4 6882:11 java
 64312 work       20   0 4311m 706m 12m S  2.7   4.4 4169:14 java
 5723  work       20   0 4648m 663m 14m S  1.0   4.2 85:36.78 java
 65089 work       20   0 4219m 623m 12m S  2.0   3.9 3203:33 java
 26665 work       20   0 4209m 367m 12m S  0.7   2.3 3302:39 java
 42722 root        20   0 682m  34m 2448 S  0.3   0.2 33:39.85 salt-minion
   933 root        20   0 246m  9.9m 836 S  0.0   0.1 1:50.13 rsyslogd
 22806 root         0 -20 126m 8624 4880 S  0.3   0.1 471:57.78 AliYunDun
```

5个

```
[readonly@16-29 ~]$ top
top - 10:16:28 up 455 days, 15:35, 2 users, load average: 1.97, 1.09, 0.88
Tasks: 151 total, 1 running, 150 sleeping, 0 stopped, 0 zombie
Cpu(s): 2.0%us, 1.3%sy, 0.0%ni, 96.7%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st

  PID USER      PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+  COMMAND
 47926 work       20   0 6755m 3.3g 12m S  0.3  21.1 256:08.60 java
 42023 work       20   0 7054m 2.9g 13m S  2.0  18.3 560:41.71 java
 19065 work       20   0 5938m 2.3g 13m S  5.0  14.9 709:48.57 java
 1141  work       20   0 5280m 1.6g 14m S  1.0  10.5 6295:43 java
 48201 work       20   0 5755m 1.0g 12m S  0.3   6.3 271:38.56 java
 34263 work       20   0 4284m 747m 13m S  1.7   4.7 2490:49 java
 7417  work       20   0 4649m 662m 12m S  0.3   4.2 49:18.30 java
 35060 work       20   0 4219m 619m 13m S  1.3   4.9 2499:47 java
 15188 work       20   0 4208m 368m 12m S  0.7   2.3 3175:10 java
```

####6.排查内存占用

由于之前排查过程中跟踪过出问题的这一台的服务情况，但是肉眼没有看出来，通过内存占用对比（top命令，然后shift +M）对比占用内存最高的几个进程，现在很明显两台机器中有一个服务肯定有问题。

####7.通过对比可以发现有个服务是有问题的

```
[readonly@16-28 ~]$ ps -ef | grep 28895
work 28895 28893 0 Sep25 ? 07:14:57 /opt/soft/java/bin/java -Djava.util.logging.config.file=/opt/web/..._hr_job/conf/logging.properties -Djava.util
.logging.manager=org.apache.juli.ClassLoaderLogManager -server -Xms2g -Xmx2g -Xmn768m -Xss1024K -XX:PermSize=256m -XX:MaxPermSize=512m -XX:ParallelGCThreads=8 -XX:+UseC
onMarkSweepGC -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+UseCMSCompactAtFullCollection -XX:SurvivorRatio=4 -XX:MaxTenuringThreshold=10 -XX:CMSInitiatingOccupancyFrac
tion=90 -Duser.dir=/opt/web/..._hr_job -Duser.home=/opt/web/..._hr_job -Djava.endorsed.dirs=/opt/soft/tomcat/endor
sed -classpath /opt/soft/tomcat/bin/bootstrap.jar:/opt/soft/tomcat/bin/tomcat-juli.jar -Dcatalina.base=/opt/web/..._hr_job -Dcatalina.home=/opt/soft/tomcat -Dj
ava.io.tmpdir=/opt/web/..._hr_job/temp org.apache.catalina.startup.Bootstrap start

[readonly@16-29 ~]$ ps -ef | grep 48201
work 48201 48199 0 Sep25 ? 04:31:57 /opt/soft/java/bin/java -Djava.util.logging.config.file=/opt/web/..._corehr_job/conf/logging.properties -Djava.util
.logging.manager=org.apache.juli.ClassLoaderLogManager -server -Xms2g -Xmx2g -Xmn768m -Xss1024K -XX:PermSize=256m -XX:MaxPermSize=512m -XX:ParallelGCThreads=8 -XX:+UseC
onMarkSweepGC -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+UseCMSCompactAtFullCollection -XX:SurvivorRatio=4 -XX:MaxTenuringThreshold=10 -XX:CMSInitiatingOccupancyFrac
tion=90 -Duser.dir=/opt/web/..._corehr_job -Duser.home=/opt/web/..._corehr_job -Djava.endorsed.dirs=/opt/soft/tomcat/endor
sed -classpath /opt/soft/tomcat/bin/bootstrap.jar:/opt/soft/tomcat/bin/tomcat-juli.jar -Dcatalina.base=/opt/web/..._corehr_job -Dcatalina.home=/opt/soft/tomcat -Dj
ava.io.tmpdir=/opt/web/..._corehr_job/temp org.apache.catalina.startup.Bootstrap start
readonly@16-29 ~$ ps -ef | grep 48201
[readonly@16-29 ~]$
```

####8.结合之前已经截图的现场可以发现

16.28的corehr_job服务占用内存是12.3%，16.29的corehr_job服务占用内存是6.3%，很明显的，到这里我们已经揪出有问题的服务了。下面继续追查为啥不一样，先透个底，有预感觉得是由于corehr_job中的一些定时任务执行之后没有释放内存导致的。看一下这个服务的堆内存占用内存比例大小，如下图：

####9.现在要看看这两个机器的同一个服务堆内存到底有什么对象

```
readonly@-16-29 ~]$
readonly@-16-29 ~]$ sudo djava jmap -histo 48201

num      #instances      #bytes  class name
-----
 1:      1182660      216427280 [C
 2:       246965      189562208 [B
 3:      1323700      42358400  java.util.concurrent.locks.AbstractQueuedSynchronizer$Node
 4:       260850      13398864  [Ljava.lang.String;
 5:        12481      11314776 [I
 6:       239656      5751744  java.lang.String
 7:       77749      2761720  [Ljava.lang.Object;
 8:       35945      2588040  org.apache.log4j.spi.LoggingEvent
 9:       49923      2396304  java.nio.HeapCharBuffer
10:       36660      2346240  java.text.DateFormatSymbols
11:       43354      2080992  java.util.HashMap
12:       74831      1795944  java.lang.StringBuilder
13:       51869      1659808  java.util.HashMap$Node
14:       42390      1356480  java.io.ObjectStreamClass$WeakClassKey
15:       59896      958336  java.lang.Integer
16:        8439      942816  java.lang.Class
17:       28303      905696  java.util.concurrent.ConcurrentHashMap$Node
18:       37280      894720  java.lang.StringBuffer
19:       11146      802512  java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask
20:       18818      752720  java.util.TreeMap$Entry
21:        6384      720136  [Ljava.util.HashMap$Node;
22:       11142      713088  com.mysql.jdbc.protocol.v1.Protocol
23:       11137      712768  com.mysql.jdbc.protocol.v1.Protocol
24:       15739      629560  java.util.HashMap$KeyIterator
25:       11138      534624  [Lcom.mysql.jdbc.protocol.v1.Protocol;
26:       11138      534624  [Lcom.mysql.jdbc.protocol.v1.Protocol;
sfp.enumeration.SerializedTime;
```

很明显我们可以看到16.28中的这个有问题的服务堆内存占用的对象比另一个正常的多，由于很小心的保留了现场我们可以分析一下，为啥有占用呢？

由于老年代占用76%，没有达到FGC的阈值，导致大量对象在年轻代,老年代驻留，下面尝试一下触发FGC，

####10.使用命令触发FGC

sudo djava jmap -histo:live 28895 执行这个命令可能引发一次FGC，然后释放内存，执行完之后确实触发了一次FGC。

```
readonly@-16-28 ~]$
readonly@-16-28 ~]$ sudo djava jmap -histo:live 28895 2000

S0      S1      E      O      M      CCS      YGC      YGCT      FGC      FGCT      GCT
0.00    0.00    1.56    24.28    92.48    86.85    1002      22.379      3      0.551    22.930
0.00    0.00    1.56    24.28    92.48    86.85    1002      22.379      3      0.551    22.930
0.00    0.00    1.56    24.28    92.48    86.85    1002      22.379      3      0.551    22.930
0.00    0.00    1.56    24.28    92.48    86.85    1002      22.379      3      0.551    22.930
```

####11.再次进行top (shift+m)

发现内存占用依然没有解决，也就是说虽然触发了FGC，但是应用程序已经申请的内存是不会释放的，笑笑~~~~

```
Tasks: 157 total,   1 running, 156 sleeping,   0 stopped,   0 zombie
Cpu(s):  3.0%us,  1.6%sy,  0.0%ni, 95.2%id,  0.1%wa,  0.0%hi,  0.1%si,  0.0%st
Mem: 16334084k total, 16117020k used,  217064k free,  30588k buffers
Swap:   0k total,   0k used,   0k free,  644128k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+  COMMAND
 28591 work      20   0 6755m 3.3g 12m S   1.0 21.0 405:33.60 java
 35205 work      20   0 7311m 3.1g 13m S   3.3 19.7 732:01.04 java
 13021 work      20   0 5953m 2.3g 13m S   6.7 15.0 932:34.58 java
 28895 work      20   0 5714m 1.9g 12m S   1.3 12.3 435:10.99 java
  1145 work      20   0 5282m 1.6g 13m S   1.0 10.4 6882:38 java
 64312 work      20   0 4311m 706m 12m S   3.0  4.4 4170:05 java
  5723 work      20   0 4648m 663m 14m S   1.0  4.2 85:53.44 java
 65089 work      20   0 4219m 623m 12m S   2.3  3.9 3204:12 java
 26665 work      20   0 4212m 367m 12m S   1.0  2.3 3302:53 java
```

分析到此结束，根据现场保留，排查数据和线索可以有以下应对方案和措施：

1. 已知引起原因，目前已重启该问题服务，内存紧张报警解除
2. 提工单进行服务器升配（不止升级有问题的这一台，还有另一台），机智~~~~
3. 排查获取大数据量的job，增加对象回收的逻辑比如用完之后clear(),设置为null之类的

这里引申两个问题：

1. java 应用程序申请的内存触发FGC之后会返回给操作系统吗？
2. 使用CMS垃圾回收算法的情况下触发FGC的条件是什么？

3. 有什么方法可以让应用触发FGC之后将内存归还给操作系统?

此外，根据涯海的总结我们可以得出出现此类问题的一些原因

针对这个案例，一个长时间运行的 Java 程序，如果在没有变更的情况下出现系统内存不足。通常可以分为以下几种情况：

1. 如果是突然不足，一般是请求了一个超大的对象（数组）；
2. 预期外的持续流量脉冲；
3. 如果是内存余量缓慢减少，通过是内存泄漏（大量引用对象未释放），可以重点检查下数据库连接/文件资源/本地缓存等资源的释放情况。

参考：<https://www.cnblogs.com/seifon/p/11228224.html>