一份超详细的Java问题排查工具单

程序员大咖 1周前

来自：云栖社区，作者：红魔七号

链接：https://yq.aliyun.com/articles/69520

前言

平时的工作中经常碰到很多疑难问题的处理，在解决问题的同时，有一些工具起到了相当大的作用，在此书写下来，一是作为笔记，可以让自己后续忘记了可快速翻阅，二是分享，希望看到此文的同学们可以拿出自己日常觉得帮助很大的工具，大家一起进步。
闲话不多说，开搞。

Linux命令类

tail

最常用的tail -f

```
tail -300f shopbase.log #倒数300行并进入实时监听文件写入模式
```

grep

```
grep forest f.txt     #文件查找
grep forest f.txt cpf.txt #多文件查找
grep 'log' /home/admin -r -n #目录下查找所有符合关键字的文件
cat f.txt | grep -i shopbase
grep 'shopbase' /home/admin -r -n --include *.{vm,java} #指定文件后缀
grep 'shopbase' /home/admin -r -n --exclude *.{vm,java} #反匹配
seq 10 | grep 5 -A 3    #上匹配
seq 10 | grep 5 -B 3    #下匹配
seq 10 | grep 5 -C 3    #上下匹配，平时用这个就妥了
cat f.txt | grep -c 'SHOPBASE'
```

awk

1.基础命令

```
awk '{print $4,$6}' f.txt
awk '{print NR,$0}' f.txt cpf.txt
awk '{print FNR,$0}' f.txt cpf.txt
awk '{print FNR,FILENAME,$0}' f.txt cpf.txt
awk '{print FILENAME,"NR="NR,"FNR="FNR,"$"NF"="$NF}' f.txt cpf.txt
echo 1:2:3:4 | awk -F: '{print $1,$2,$3,$4}'
```

2.匹配

```
awk '/ldb/ {print}' f.txt   #匹配ldb
awk '!/ldb/ {print}' f.txt  #不匹配ldb
awk '/ldb/ && /LISTEN/ {print}' f.txt   #匹配ldb和LISTEN
awk '$5 ~ /ldb/ {print}' f.txt #第五列匹配ldb
```

3.内建变量

NR:NR表示从awk开始执行后，按照记录分隔符读取的数据次数，默认的记录分隔符为换行符，因此默认的就是读取的数据行数，NR可以理解为Number of Record的缩写。

FNR:在awk处理多个输入文件的时候，在处理完第一个文件后，NR并不会从1开始，而是继续累加，因此就出现了FNR，每当处理一个新文件的时候，FNR就从1开始计数，FNR可以理解为File Number of Record。

NF: NF表示目前的记录被分割的字段的数目，NF可以理解为Number of Field。

find

```
sudo -u admin find /home/admin /tmp /usr -name *.log(多个目录去找)
find . -iname *.txt(大小写都匹配)
find . -type d(当前目录下的所有子目录)
find /usr -type l(当前目录下所有的符号链接)
find /usr -type l -name "z*" -ls(符号链接的详细信息 eg:inode,目录)
find /home/admin -size +250000k(超过250000k的文件，当然+改成-就是小于了)
find /home/admin f -perm 777 -exec ls -l {} ; (按照权限查询文件)
find /home/admin -atime -1  1天内访问过的文件
find /home/admin -ctime -1  1天内状态改变过的文件
find /home/admin -mtime -1  1天内修改过的文件
find /home/admin -amin -1  1分钟内访问过的文件
find /home/admin -cmin -1  1分钟内状态改变过的文件
find /home/admin -mmin -1  1分钟内修改过的文件
```

pgm

批量查询vm-shopbase满足条件的日志

```
pgm -A -f vm-shopbase 'cat /home/admin/shopbase/logs/shopbase.log.2017-01-17|grep 2069861630'
```

tsar

tsar是咱公司自己的采集工具。很好用，将历史收集到的数据持久化在磁盘上，所以我们快速来查询历史的系统数据。当然实时的应用情况也是可以查询的啦。大部分机器上都有安装。

`tsar` ##可以查看最近一天的各项指标

```
[yingchao.zyc@v218130055 ~]$ tsar
Time             ---cpu-- ---mem-- ---tcp-- -----traffic---- --xvda-- -xvda1-- --xvdb--  ---load-
Time              util     util    retran    pktin  pktout     util     util     util     load1
01/02/17-15:00   10.93    72.94      0.18    12.00   13.00     2.59     2.59     0.00      0.24
01/02/17-15:05   11.01    72.84      0.17    11.00   13.00     3.39     3.39     0.00      0.46
01/02/17-15:10   10.82    72.85      0.17    11.00   13.00     2.56     2.56     0.00      0.41
01/02/17-15:15   10.90    72.87      0.22    11.00   13.00     2.59     2.59     0.00      0.26
01/02/17-15:20   10.80    72.74      0.19    11.00   13.00     2.50     2.50     0.00      0.59
01/02/17-15:25   10.87    72.73      0.14    12.00   13.00     3.01     3.01     0.00      0.04
01/02/17-15:30   10.77    72.75      0.19    12.00   13.00     2.71     2.71     0.00      0.26
01/02/17-15:35   10.94    72.74      0.20    11.00   13.00     3.93     3.93     0.00      0.23
01/02/17-15:40   10.68    72.75      0.14    12.00   13.00     3.09     3.09     0.00      0.69
01/02/17-15:45   10.87    72.77      0.19    11.00   13.00     2.74     2.74     0.00      0.40
01/02/17-15:50   10.78    72.79      0.19    11.00   13.00     2.83     2.83     0.00      0.04
01/02/17-15:55   11.04    72.71      0.14    11.00   13.00     3.27     3.27     0.00      0.57
```

`tsar --live` ##可以查看实时指标，默认五秒一刷

```
[yingchao.zyc@v218130055 ~]$ tsar --live
Time             ---cpu-- ---mem-- ---tcp-- -----traffic---- --xvda-- -xvda1-- --xvdb--  ---load
Time              util     util    retran    pktin  pktout     util     util     util     load1
02/02/17-15:03:21   9.83    73.21      0.00     8.00   12.00     2.00     2.00     0.00      0.58
02/02/17-15:03:26   8.62    73.21      0.00     3.00    4.00     1.60     1.60     0.00      0.53
02/02/17-15:03:31   9.65    73.22      0.00     6.00    9.00     2.14     2.14     0.00      0.65
02/02/17-15:03:36  10.05    73.21      0.00    16.00   15.00     2.34     2.34     0.00      0.60
02/02/17-15:03:41  10.48    73.22      0.00     7.00   11.00     1.82     1.82     0.00      0.71
02/02/17-15:03:46  10.27    73.22      0.00     9.00   13.00     2.10     2.10     0.00      0.65
```

`tsar -d 20161218` ##指定查看某天的数据，貌似最多只能看四个月的数据

```
[yingchao.zyc@v218130055 ~]$ tsar -d 20161218
Time          ---cpu-- ---mem-- ---tcp-- -----traffic---- --xvda-- -xvda1-- --xvdb--  ---load-
Time            util     util   retran    pktin  pktout     util     util     util     load1
18/12/16-00:05  8.51    63.18     0.22    11.00   12.00     2.69     2.69     0.00      0.17
18/12/16-00:10  8.32    63.04     0.26     9.00   11.00     2.98     2.98     0.00      0.23
18/12/16-00:15  8.53    63.03     0.24    11.00   12.00     2.62     2.62     0.00      0.13
18/12/16-00:20  8.58    63.06     0.20     9.00   11.00     2.53     2.53     0.00      0.18
18/12/16-00:25  8.53    63.05     0.24    11.00   12.00     2.69     2.69     0.00      0.78
18/12/16-00:30  8.37    63.05     0.23     9.00   11.00     2.81     2.81     0.00      0.40
18/12/16-00:35  8.72    63.03     0.23    12.00   12.00     2.58     2.58     0.00      0.42
18/12/16-00:40  8.52    63.08     0.20     9.00   11.00     2.43     2.43     0.00      0.42
```

tsar --mem
tsar --load
tsar --cpu
##当然这个也可以和-d参数配合来查询某天的单个指标的情况

```
[yingchao.zyc@v218130055 ~]$ tsar --mem
Time             ----------------------mem----------------------
Time               free    used    buff    cach   total    util
01/02/17-15:20   62.8M    2.8G  148.8M  867.5M    3.9G   72.74
01/02/17-15:25   61.7M    2.8G  148.8M  869.1M    3.9G   72.73
01/02/17-15:30   59.3M    2.8G  148.8M  870.8M    3.9G   72.75
01/02/17-15:35   58.1M    2.8G  148.8M  872.4M    3.9G   72.74
01/02/17-15:40   55.8M    2.8G  148.8M  874.1M    3.9G   72.75
```

```
[yingchao.zyc@v218130055 ~]$ tsar --load
Time             -------------------load-----------------
Time               load1   load5  load15    runq    plit
01/02/17-15:20      0.59    0.55    0.40    1.00    1.2K
01/02/17-15:25      0.04    0.28    0.33    2.00    1.2K
01/02/17-15:30      0.26    0.26    0.31    4.00    1.2K
01/02/17-15:35      0.23    0.27    0.31    4.00    1.2K
01/02/17-15:40      0.69    0.54    0.42    1.00    1.2K
01/02/17-15:45      0.40    0.45    0.41    2.00    1.2K
```

```
[yingchao.zyc@v218130055 ~]$ tsar --cpu
Time             ---------------------cpu----------------------
Time               user     sys    wait    hirq    sirq    util
01/02/17-15:25     6.29    4.47    1.18    0.02    0.04   10.83
01/02/17-15:30     6.15    4.55    1.13    0.02    0.04   10.77
01/02/17-15:35     6.36    4.50    1.66    0.03    0.04   10.94
01/02/17-15:40     6.14    4.48    1.33    0.02    0.03   10.68
01/02/17-15:45     6.28    4.52    1.17    0.02    0.04   10.87
01/02/17-15:50     6.25    4.46    1.15    0.02    0.04   10.78
01/02/17-15:55     6.37    4.60    1.40    0.02    0.04   11.04
```

top

top除了看一些基本信息之外，剩下的就是配合来查询vm的各种问题了

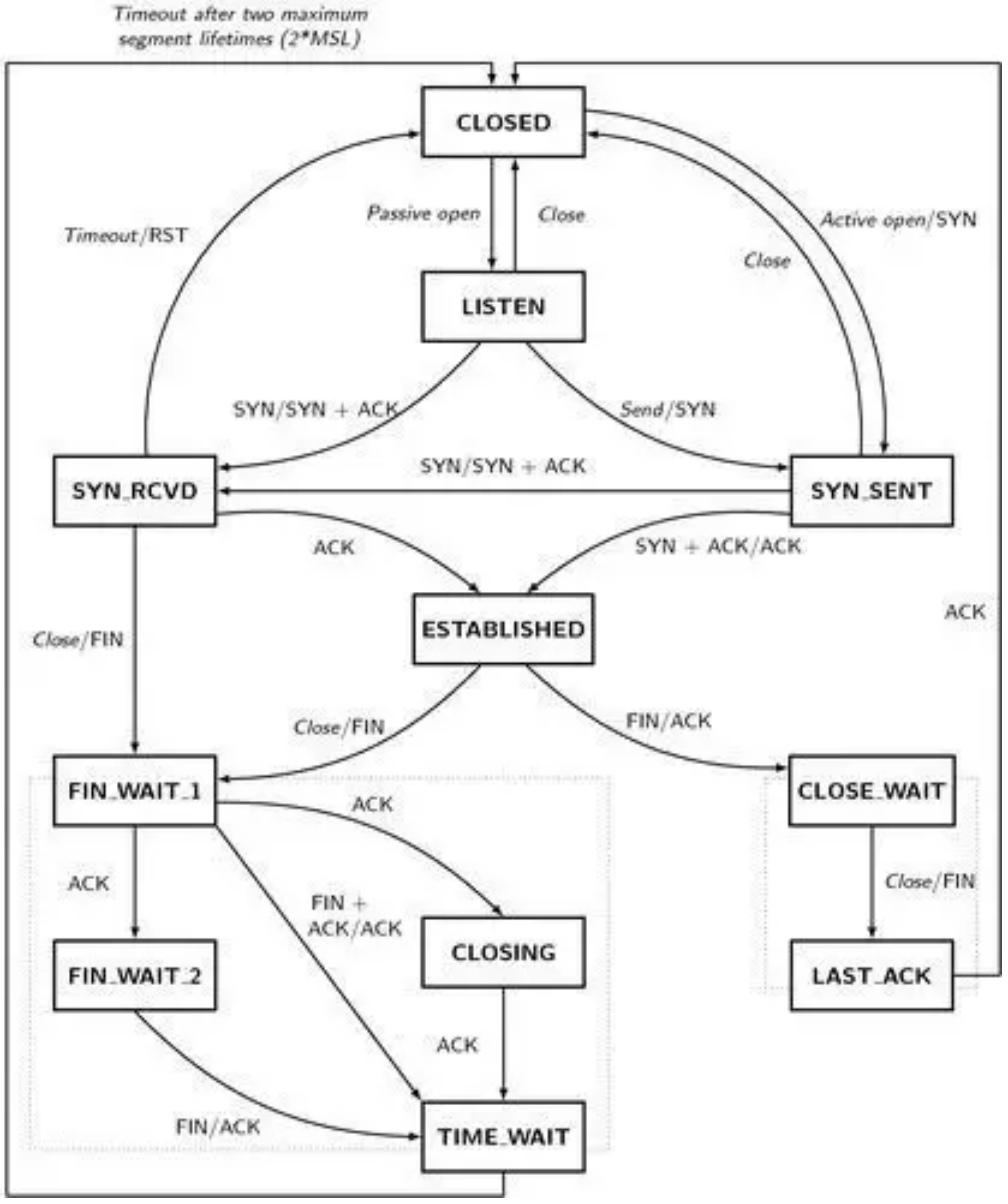
```
ps -ef | grep java
top -H -p pid
```

获得线程10进制转16进制后jstack去抓看这个线程到底在干啥

其他

```
netstat -nat|awk  '{print $6}'|sort|uniq -c|sort -rn
#查看当前连接，注意close_wait偏高的情况，比如如下
```

```
[yingchao.zyc@v218130055 ~]$ netstat -nat|awk  '{print $6}'|sort|uniq -c|sort -rn
    220 CLOSE_WAIT
     29 LISTEN
     21 ESTABLISHED
     15 SYN_RECV
     11 TIME_WAIT
```



排查利器

btrace

首当其冲的要说的是btrace。真是生产环境&预发的排查问题大杀器。 简介什么的就不说了。直接上代码干

1.查看当前谁调用了ArrayList的add方法，同时只打印当前ArrayList的size大于500的线程调用栈

```
@OnMethod(clazz = "java.util.ArrayList", method="add", location = @Location(value = Kind.CALL, clazz =
"/.*/", method = "/.*/"))
public static void m(@ProbeClassName String probeClass, @ProbeMethodName String probeMethod, @TargetInstance
Object instance, @TargetMethodOrField String method) {
    if(getInt(field("java.util.ArrayList", "size"), instance) > 479){
        println("check who ArrayList.add method:" + probeClass + "#" + probeMethod  + ", method:" + method +
", size:" + getInt(field("java.util.ArrayList", "size"), instance));
        jstack();
        println();
        println("==========================");
        println();
    }
}
```

2.监控当前服务方法被调用时返回的值以及请求的参数

```
@OnMethod(clazz = "com.taobao.sellerhome.transfer.biz.impl.C2CApplyerServiceImpl", method="nav", location = @
Location(value = Kind.RETURN))
public static void mt(long userId, int current, int relation, String check, String redirectUrl, @Return AnyTy
pe result) {
        println("parameter# userId:" + userId + ", current:" + current + ", relation:" + relation + ", chec
k:" + check + ", redirectUrl:" + redirectUrl + ", result:" + result);

}
```

其他功能集团的一些工具或多或少都有，就不说了。感兴趣的请移步。
https://github.com/btraceio/btrace

> 注意:
> 经过观察，1.3.9的release输出不稳定，要多触发几次才能看到正确的结果
> 正则表达式匹配trace类时范围一定要控制，否则极有可能出现跑满CPU导致应用卡死的情况
> 由于是字节码注入的原理，想要应用恢复到正常情况，需要重启应用。

Greys

Greys是@杜琨的大作吧。说几个挺棒的功能(部分功能和btrace重合):

sc -df xxx: 输出当前类的详情,包括源码位置和classloader结构

trace class method: 相当喜欢这个功能! 很早前可以早JProfiler看到这个功能。打印出当前方法调用的耗时情况，细分到每个方法。对排查方法性能时很有帮助，比如我之前这篇就是使用了trace命令来的:http://www.atatech.org/articles/52947。

其他功能部分和btrace重合，可以选用，感兴趣的请移步。
http://www.atatech.org/articles/26247

另外相关联的是arthas，他是基于Greys的，感兴趣的再移步 http://mw.alibaba-inc.com/products/arthas/docs/middleware-container/arthas.wiki/home.html?spm=a1z9z.8109794.header.32.1lsoMc

javOSize

就说一个功能
classes：通过修改了字节码，改变了类的内容，即时生效。 所以可以做到快速的在某个地方打个日志看看输出，缺点是对代码的侵入性太大。但是如果自己知道自己在干嘛，的确是不错的玩意儿。

其他功能Greys和btrace都能很轻易做的到，不说了。

可以看看我之前写的一篇javOSize的简介http://www.atatech.org/articles/38546
官网请移步http://www.javosize.com/

JProfiler

之前判断许多问题要通过JProfiler，但是现在Greys和btrace基本都能搞定了。再加上出问题的基本上都是生产环境(网络隔离)，所以基本不怎么使用了，但是还是要标记一下。
官网请移步https://www.ej-technologies.com/products/jprofiler/overview.html

大杀器

eclipseMAT

可作为eclipse的插件，也可作为单独的程序打开。
详情请移步http://www.eclipse.org/mat/

zprofiler

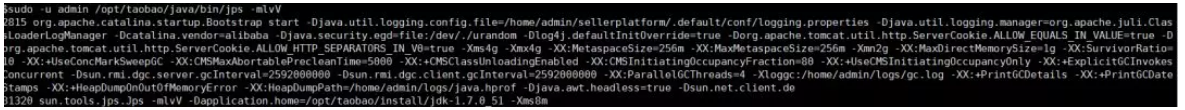
集团内的开发应该是无人不知无人不晓了。简而言之一句话:有了zprofiler还要mat干嘛
详情请移步zprofiler.alibaba-inc.com

Java三板斧，噢不对，是七把

jps

我只用一条命令：

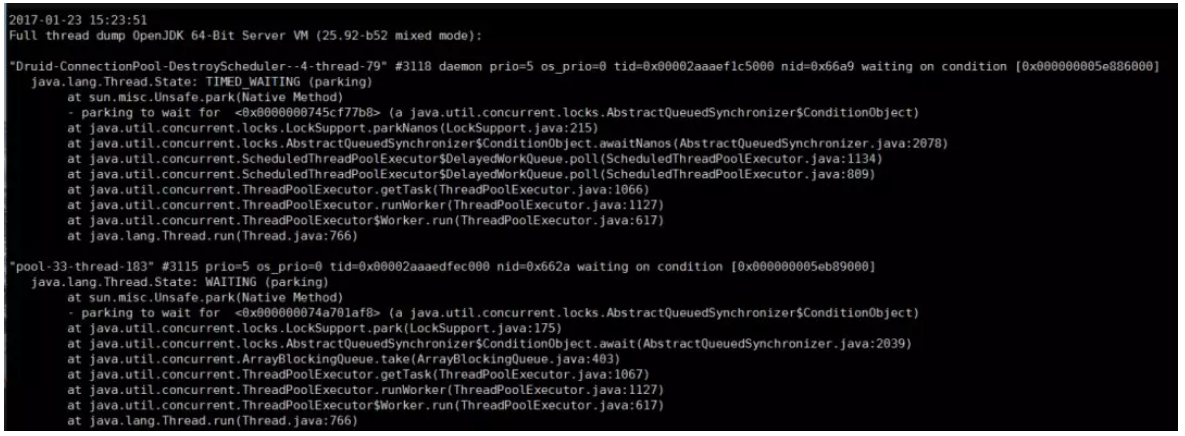
```
sudo -u admin /opt/taobao/java/bin/jps -mlvV
```



jstack

普通用法:

```
sudo -u admin /opt/taobao/install/ajdk-8_1_1_fp1-b52/bin/jstack 2815
```



native+java栈:

```
sudo -u admin /opt/taobao/install/ajdk-8_1_1_fp1-b52/bin/jstack -m 2815
```

```
Attaching to process ID 2815, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 25.92-b52
Deadlock Detection:

No deadlocks found.

---------------- 2819 ----------------
0x00002b7a3127eab6      __libc_poll + 0x66
---------------- 2822 ----------------
0x00002b7a30b88f59      __pthread_cond_wait + 0xb9
0x00002b7a3269b0bb      _ZN7Monitor5IWaitEP6Threadl + 0xcb
0x00002b7a3269b64e      _ZN7Monitor4waitEblb + 0x9e
0x00002b7a328a78c8      _ZN10GangWorker4loopEv + 0x58
0x00002b7a326dda42      _ZL10java_startP6Thread + 0x122
---------------- 2823 ----------------
0x00002b7a30b88f59      __pthread_cond_wait + 0xb9
0x00002b7a3269b0bb      _ZN7Monitor5IWaitEP6Threadl + 0xcb
0x00002b7a3269b64e      _ZN7Monitor4waitEblb + 0x9e
0x00002b7a328a78c8      _ZN10GangWorker4loopEv + 0x58
0x00002b7a326dda42      _ZL10java_startP6Thread + 0x122
---------------- 2824 ----------------
0x00002b7a30b88f59      __pthread_cond_wait + 0xb9
0x00002b7a3269b0bb      _ZN7Monitor5IWaitEP6Threadl + 0xcb
0x00002b7a3269b64e      _ZN7Monitor4waitEblb + 0x9e
0x00002b7a328a78c8      _ZN10GangWorker4loopEv + 0x58
0x00002b7a326dda42      _ZL10java_startP6Thread + 0x122
---------------- 2825 ----------------
```

jinfo

可看系统启动的参数，如下

```
sudo -u admin /opt/taobao/install/ajdk-8_1_1_fp1-b52/bin/jinfo -flags 2815
```

jmap

两个用途

1.查看堆的情况

```
sudo -u admin /opt/taobao/install/ajdk-8_1_1_fp1-b52/bin/jmap -heap 2815
```

```
$sudo -u admin /opt/taobao/install/ajdk-8_1_1_fp1-b52/bin/jmap -heap 2815
Attaching to process ID 2815, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 25.92-b52

using parallel threads in the new generation.
using thread-local object allocation.
Concurrent Mark-Sweep GC

Heap Configuration:
   MinHeapFreeRatio         = 40
   MaxHeapFreeRatio         = 70
   MaxHeapSize              = 4294967296 (4096.0MB)
   NewSize                  = 2147483648 (2048.0MB)
   MaxNewSize               = 2147483648 (2048.0MB)
   OldSize                  = 2147483648 (2048.0MB)
   NewRatio                 = 2
   SurvivorRatio            = 10
   MetaspaceSize            = 268435456 (256.0MB)
   CompressedClassSpaceSize = 1073741824 (1024.0MB)
   MaxMetaspaceSize         = 268435456 (256.0MB)
   G1HeapRegionSize         = 0 (0.0MB)

Heap Usage:
New Generation (Eden + 1 Survivor Space):
   capacity = 1968570368 (1877.375MB)
   used     = 1822766864 (1738.325942993164MB)
   free     = 145803504 (139.04905700683594MB)
   92.59343194583735% used
Eden Space:
   capacity = 1789657088 (1706.75MB)
   used     = 1774411392 (1692.2105712890625MB)
   free     = 15245696 (14.5394287109375MB)
   99.14812194457669% used
From Space:
   capacity = 178913280 (170.625MB)
   used     = 48355472 (46.11537170410156MB)
   free     = 130557808 (124.50962829589844MB)
   27.02732407566392% used
```

2.dump

```
sudo -u admin /opt/taobao/install/ajdk-8_1_1_fp1-b52/bin/jmap -dump:live,format=b,file=/tmp/heap2.bir
```

或者

```
sudo -u admin /opt/taobao/install/ajdk-8_1_1_fp1-b52/bin/jmap -dump:format=b,file=/tmp/heap3.bin 2815
```

3.看看堆都被谁占了？再配合zprofiler和btrace，排查问题简直是如虎添翼

```
sudo -u admin /opt/taobao/install/ajdk-8_1_1_fp1-b52/bin/jmap -histo 2815 | head -10
```

```
$sudo -u admin /opt/taobao/install/ajdk-8_1_1_fp1-b52/bin/jmap -histo 2815 | head -10

 num     #instances         #bytes  class name
----------------------------------------------
   1:        853529       86068120  [C
   2:        355583       68141472  [B
   3:         62522       25186360  [I
   4:        760046       18241104  java.lang.String
   5:         80380        5667176  [Ljava.lang.Object;
   6:        140015        4480480  com.taobao.csp.sentinel.util.LongAdder
   7:        133162        4261184  com.taobao.pandora.loader.util.AsciiBytes
```

jstat

jstat参数众多，但是使用一个就够了

```
sudo -u admin /opt/taobao/install/ajdk-8_1_1_fp1-b52/bin/jstat -gcutil 2815 1000
```

```
$sudo -u admin /opt/taobao/install/ajdk-8_1_1_fp1-b52/bin/jstat -gcutil 2815 1000
  S0     S1     E      O      M     CCS    YGC     YGCT    FGC    FGCT     GCT
  3.06   0.00  27.40   9.76  97.15  94.39     30    2.580     2    1.644    4.224
  3.06   0.00  27.52   9.76  97.15  94.39     30    2.580     2    1.644    4.224
  3.06   0.00  27.54   9.76  97.15  94.39     30    2.580     2    1.644    4.224
  3.06   0.00  27.54   9.76  97.15  94.39     30    2.580     2    1.644    4.224
  3.06   0.00  27.60   9.76  97.15  94.39     30    2.580     2    1.644    4.224
  3.06   0.00  28.04   9.76  97.15  94.39     30    2.580     2    1.644    4.224
  3.06   0.00  28.04   9.76  97.15  94.39     30    2.580     2    1.644    4.224
  3.06   0.00  28.05   9.76  97.15  94.39     30    2.580     2    1.644    4.224
  3.06   0.00  28.05   9.76  97.15  94.39     30    2.580     2    1.644    4.224
  3.06   0.00  28.05   9.76  97.15  94.39     30    2.580     2    1.644    4.224
  3.06   0.00  28.12   9.76  97.15  94.39     30    2.580     2    1.644    4.224
  3.06   0.00  28.12   9.76  97.15  94.39     30    2.580     2    1.644    4.224
  3.06   0.00  28.20   9.76  97.15  94.39     30    2.580     2    1.644    4.224
  3.06   0.00  28.20   9.76  97.15  94.39     30    2.580     2    1.644    4.224
  3.06   0.00  28.20   9.76  97.15  94.39     30    2.580     2    1.644    4.224
```

jdb

时至今日，jdb也是经常使用的。
jdb可以用来预发debug,假设你预发的java_home是/opt/taobao/java/，远程调试端口是8000.那么

```
sudo -u admin /opt/taobao/java/bin/jdb -attach 8000.
```

```
[yingchao.zyc@v218130055 ~]$ sudo -u admin /opt/taobao/java/bin/jdb -attach 8000
设置未捕获的java.lang.Throwable
设置延迟的未捕获的java.lang.Throwable
正在初始化jdb...
>
```

出现以上代表jdb启动成功。后续可以进行设置断点进行调试。
具体参数可见oracle官方说明http://docs.oracle.com/javase/7/docs/technotes/tools/windows/jdb.html

CHLSDB

CHLSDB感觉很多情况下可以看到更好玩的东西，不详细叙述了。 查询资料听说jstack和jmap等工具就是基于它的。

```
sudo -u admin /opt/taobao/java/bin/java -classpath /opt/taobao/java/lib/sa-jdi.jar sun.jvm.hotspot.Cl
```

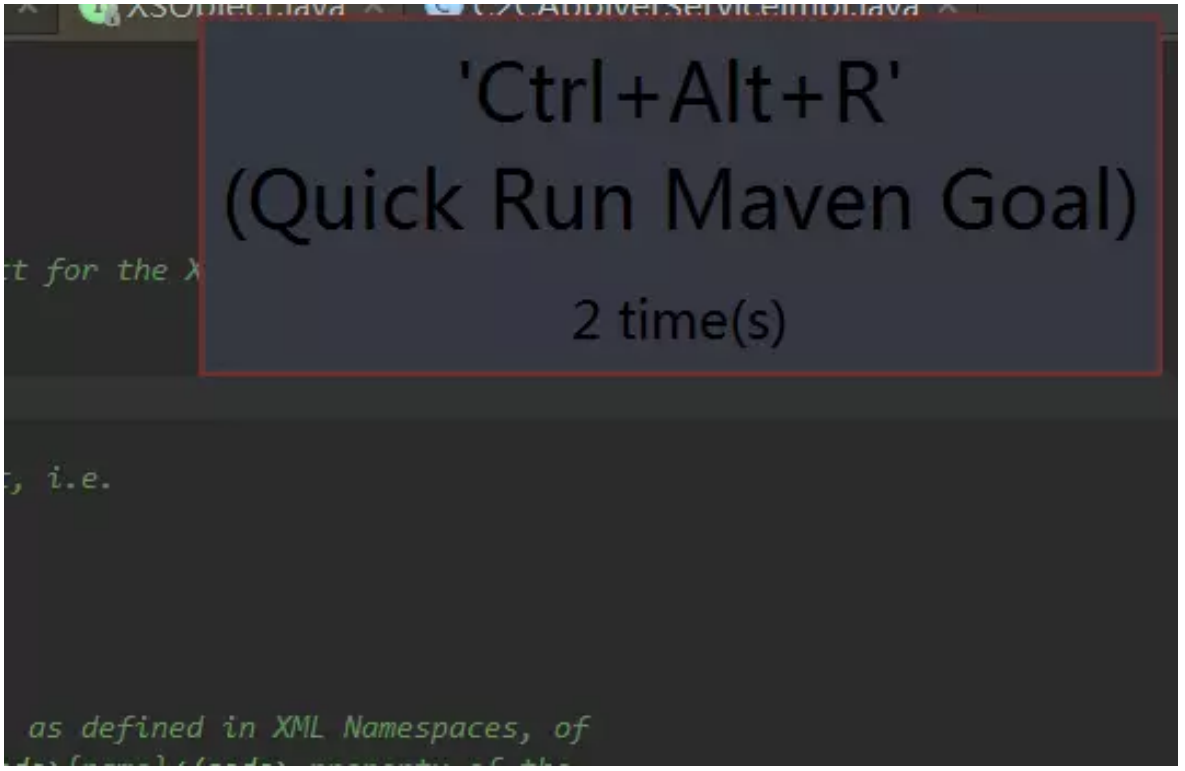
更详细的可见R大此贴

http://rednaxelafx.iteye.com/blog/1847971

> **plugin of intellij idea**

key promoter

快捷键一次你记不住，多来几次你总能记住了吧？



maven helper

分析maven依赖的好帮手。

> **VM options**

1. 你的类到底是从哪个文件加载进来的？

```
-XX:+TraceClassLoading
结果形如[Loaded java.lang.invoke.MethodHandleImpl$Lazy from D:programmejdkjdk8U74jrelibt.jar]
```

2. 应用挂了输出dump文件

```
-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/home/admin/logs/java.hprof
集团的vm参数里边基本都有这个选项
```

jar包冲突

把这个单独写个大标题不过分吧？每个人或多或少都处理过这种烦人的case。我特么下边这么多方案不信就搞不定你?

mvn dependency:tree > ~/dependency.txt
打出所有依赖

mvn dependency:tree -Dverbose -Dincludes=groupId:artifactId
只打出指定groupId和artifactId的依赖关系

-XX:+TraceClassLoading
vm启动脚本加入。在tomcat启动脚本中可见加载类的详细信息

-verbose
vm启动脚本加入。在tomcat启动脚本中可见加载类的详细信息

greys:sc
greys的sc命令也能清晰的看到当前类是从哪里加载过来的

tomcat-classloader-locate
通过以下url可以获知当前类是从哪里加载的
curl http://localhost:8006/classloader/locate?class=org.apache.xerces.xs.XSObject

ALI-TOMCAT带给我们的惊喜(感谢@务观)

1.列出容器加载的jar列表
curl http://localhost:8006/classloader/jars

2.列出当前当当前类加载的实际jar包位置，解决类冲突时有用
curl http://localhost:8006/classloader/locate?class=org.apache.xerces.xs.XSObject

```
$curl http://localhost:8006/classloader/locate?class=org.apache.xerces.xs.XSObject
Find loaded class in :
-----------------------------------------------------------------------------------------
|1   |/home/admin/sellerplatform/target/sellerplatform.war/WEB-INF/lib/xercesImpl-2.6.2.jar|
-----------------------------------------------------------------------------------------
```

其他

gpref

http://www.atatech.org/articles/33317

dmesg
如果发现自己的java进程悄无声息的消失了，几乎没有留下任何线索，那么dmesg一发，很有可能有你想要的。

```
sudo dmesg|grep -i kill|less
```

去找关键字oom_killer。找到的结果类似如下:

```
[6710782.021013] java invoked oom-killer: gfp_mask=0xd0, order=0, oom_adj=0, oom_scoe_adj=0
[6710782.070639] [<ffffffff81118898>] ? oom_kill_process+0x68/0x140
[6710782.257588] Task in /LXC011175068174 killed as a result of limit of /LXC011175068174
[6710784.698347] Memory cgroup out of memory: Kill process 215701 (java) score 854 or sacrifice chilc
[6710784.707978] Killed process 215701, UID 679, (java) total-vm:11017300kB, anon-rss:7152432kB, file
```

以上表明，对应的java进程被系统的OOM Killer给干掉了，得分为854.
解释一下OOM killer（Out-Of-Memory killer），该机制会监控机器的内存资源消耗。当机器内存耗尽

前，该机制会扫描所有的进程（按照一定规则计算，内存占用，时间等），挑选出得分最高的进程，然后杀死，从而保护机器。

dmesg日志时间转换公式:
log实际时间=格林威治1970-01-01+(当前时间秒数-系统启动至今的秒数+dmesg打印的log时间)秒数：

```
date -d "1970-01-01 UTC `echo "$(date +%s)-$(cat /proc/uptime|cut -f 1 -d' ')+12288812.926194"|bc ` s
```

剩下的，就是看看为什么内存这么大，触发了OOM-Killer了。

> **新技能get**

想要精细的控制QPS? 比如这样一个场景，你调用某个接口，对方明确需要你限制你的QPS在400之内你怎么控制？这个时候RateLimiter就有了用武之地。详情可移步http://ifeve.com/guava-ratelimiter/

【点击成为源码大神】

Read more