生产环境线上问题的诊断

# 1 CPU性能评估

1. upTime命令

uptime是监控系统性能最常用的一个命令，主要用来统计系统当前的运行状况，输出的信息依次为：系统现在的时间、系统从上次开机到现在运行了多长时间、系统目前有多少登陆用户、系统在一分钟内、五分钟内、十五分钟内的平均负载。看下面的一个输出：

[kafka@wangqg-app01 ~]$ uptime

15:13:14 up 527 days, 09:08, 2 users, load average: 1.12, 1.20, 0.89

这里需要注意的是load average这个输出值，这三个值的大小一般不能大于系统CPU的个数

[root@demo01 ~]# grep 'model name' /proc/cpuinfo |wc -l

**2**

例如，本输出中系统有2个CPU,如果load average的三个值长期大于2时，说明CPU很繁忙，负载很高，可能会影响系统性能，但是偶尔大于2时，倒不用担心，一般不会影响系统性能。相反，如果load average的输出值小于CPU的个数，则表示CPU还有空闲的时间片，比如本例中的输出，CPU是非常空闲的。

CPU三个负责时间段的隐藏含义

(1)如果1分钟，5分钟，15分钟三个值基本相同，或者相差不大，那说明系统负责很平稳

(2)如果1分钟的值远小于15分钟的值，那说明系统最近1分钟的负载在减少，而过去15分钟内相对有比较大的负载

(2)如果1分钟的值远大于15分钟的值，说明最近1分钟的负载在增加，需要持续观察，如果1分钟的平均负载接近了CPU个数，那系统已经发生过载问题，需要分析如何优化。

vmstat [-V] [-n] [delay[count]

各个选项及参数含义如下：

-V：表示打印出版本信息，是可选参数。

-n：表示在周期性循环输出时，输出的头部信息仅显示一次。

delay：表示两次输出之间的间隔时间。

count：表示按照“delay”指定的时间间隔统计的次数。默认为1

样例

vmstat 3  
表示每3秒钟更新一次输出信息，循环输出，按ctrl+c停止输出。  
vmstat 3 5  
表示每3秒更新一次输出信息，统计5次后停止输出。

[kafka@wanggq ~]$ vmstat 3 5

procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----

r b swpd free buff cache si so bi bo in cs us sy id wa st

1 0 11624 70877456 979720 119543936 0 0 0 2 0 0 0 1 99 0 0

1 0 11624 70877208 979720 119543952 0 0 0 112 8273 14056 1 1 98 0 0

1 0 11624 70877832 979720 119544008 0 0 0 68 7257 13249 0 1 98 0 0

1 0 11624 70873480 979720 119544240 0 0 0 52 8468 14988 2 2 97 0 0

1 0 11624 70872928 979720 119544264 0 0 0 181 7504 14071 0 1 98 0 0

Procs

r列表示运行和等待cpu时间片的进程数，这个值如果长期大于系统CPU的个数，说明CPU不足，需要增加CPU。

b列表示在等待资源的进程数，比如正在等待I/O、或者内存交换等。

memory

swpd列表示切换到内存交换区的内存数量（以k为单位）。如果swpd的值不为0，或者比较大，只要si、so的值长期为0，这种情况下一般不用担心，不会影响系统性能。

free列表示当前空闲的物理内存数量（以k为单位）

buff列表示buffers cache的内存数量，一般对块设备的读写才需要缓冲。

cache列表示page cached的内存数量，一般作为文件系统cached，频繁访问的文件都会被cached，如果cache值较大，说明cached的文件数较多，如果此时IO中bi比较小，说明文件系统效率比较好。

swap

si列表示由磁盘调入内存，也就是内存进入内存交换区的数量。

so列表示由内存调入磁盘，也就是内存交换区进入内存的数量。  
一般情况下，si、so的值都为0，如果si、so的值长期不为0，则表示系统内存不足。需要增加系统内存

IO

Bi列表示从块设备读入数据的总量（即读磁盘）（每秒kb）。

Bo列表示写入到块设备的数据总量（即写磁盘）（每秒kb）  
这里我们设置的bi+bo参考值为1000，如果超过1000，而且wa值较大，则表示系统磁盘IO有问题，应该考虑提高磁盘的读写性能。

system

in列表示在某一时间间隔中观测到的每秒设备中断数。

cs列表示每秒产生的上下文切换次数。  
上面这2个值越大，会看到由内核消耗的CPU时间会越多。

CPU项 ,此列是我们关注的重点。

us列显示了用户进程消耗的CPU 时间百分比。us的值比较高时，说明用户进程消耗的cpu时间多，但是如果长期大于50%，就需要考虑优化程序或算法。

sy列显示了内核进程消耗的CPU时间百分比。sy的值较高时，说明内核消耗的CPU资源很多。  
根据经验，us+sy的参考值为80%，如果us+sy大于 80%说明可能存在CPU资源不足。

id 列显示了CPU处在空闲状态的时间百分比。

wa列显示了IO等待所占用的CPU时间百分比。wa值越高，说明IO等待越严重，根据经验，wa的参考值为20%，如果wa超过20%，说明IO等待严重，引起IO等待的原因可能是磁盘大量随机读写造成的，也可能是磁盘或者磁盘控制器的带宽瓶颈造成的（主要是块操作）。

mpstat [-P {cpu|ALL}] [internal [count]

各个选项及参数含义如下：

-P {cpu l ALL}：表示监控哪个CPU， cpu在[0,cpu个数-1]中取值

internal： 相邻的两次采样的间隔时间

count：采样的次数，count只能和delay一起使用

分析：

[kafka@wanggq-app01 ~]$ mpstat 3 5

Linux 3.10.0-693.el7.x86\_64 (wanggq-app01) 08/07/2020 \_x86\_64\_ (96 CPU)

03:05:48 PM CPU %usr %nice %sys %iowait %irq %soft %steal %guest %gnice %idle

03:05:51 PM all 1.24 0.00 0.07 0.00 0.00 0.00 0.00 0.00 0.00 98.68

03:05:54 PM all 2.16 0.00 0.15 0.00 0.00 0.00 0.00 0.00 0.00 97.69

03:05:57 PM all 1.11 0.00 0.04 0.00 0.00 0.00 0.00 0.00 0.00 98.85

03:06:00 PM all 1.06 0.00 0.04 0.00 0.00 0.00 0.00 0.00 0.00 98.90

03:06:03 PM all 1.21 0.00 0.03 0.00 0.00 0.00 0.00 0.00 0.00 98.76

Average: all 1.35 0.00 0.07 0.00 0.00 0.00 0.00 0.00 0.00 98.57

这是直接执行mpstat，然后指定输出间隔和输出次数，这种情况下统计的是系统所有CPU核的状态数据，mpstat输出中每列的含义如下：

CPU： 处理器ID，多处理器时，会显示每个处理器ID号。

%usr： 显示了用户进程消耗的CPU 时间百分比。

%nice： 显示了运行正常进程所消耗的CPU 时间百分比。

%sys： 显示了系统进程消耗的CPU时间百分比。

%iowait：显示了IO等待所占用的CPU时间百分比。

%irq： 显示了硬中断时间占用的CPU时间百分比。

%soft： 显示了软中断时间占用的CPU时间百分比。

%steal：显示了在内存相对紧张的环境下page in强制对不同的页面进行的steal操作。

%guest：显示运行虚拟处理器时CPU花费时间的百分比。

%idle：显示了CPU处在空闲状态的时间百分比。

mpstat中统计都适合 vmstat中所有关于CPU的统计，具体调优过程中，可以通过两个命令的结合，来综合判断CPU是否有性能问题。例如，当通过两个命令都发现较低的%idle 数字时，那么可以判断应该是CPU不足的问题。而当看到较高的%iowait数字时，就应该马上知道在当前负载下I/O子系统出现了某些问题。

## 诊断案例

实验材料：

yum install -y epel-release

终端模拟一个CPU使用率100%的场景，仿照BCryptPasswordEncodercpu密集型进程。

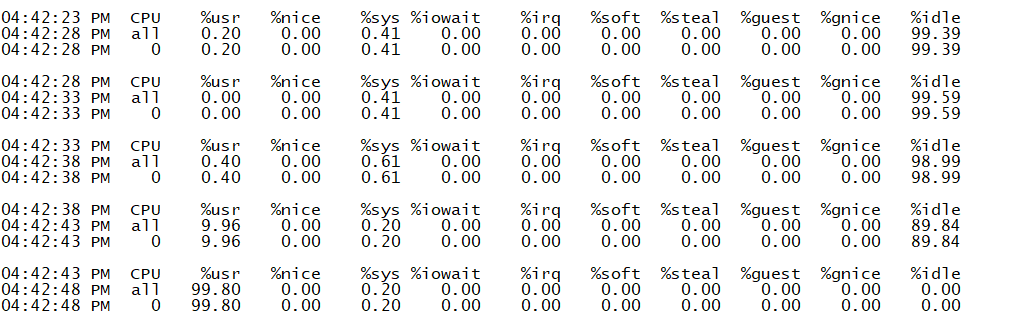
[kafka@demo1 ~]# stress --cpu 1 --timeout 600

观察CPU

[kafka@demo1 ~]# watch -d uptime

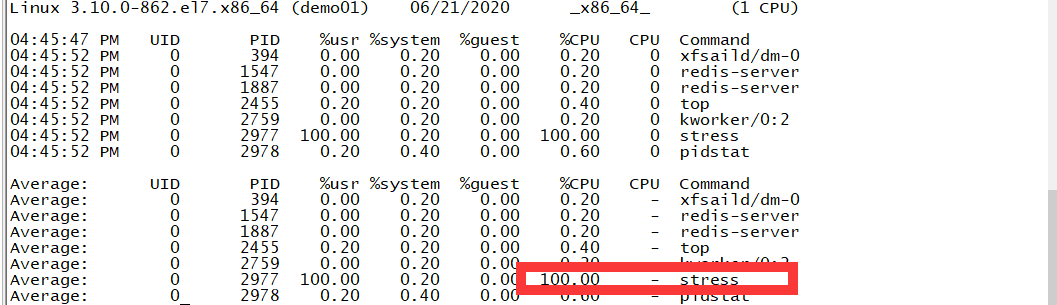
分析负载过高的原因

[kafka@demo1 ~]# mpstat -P ALL 5

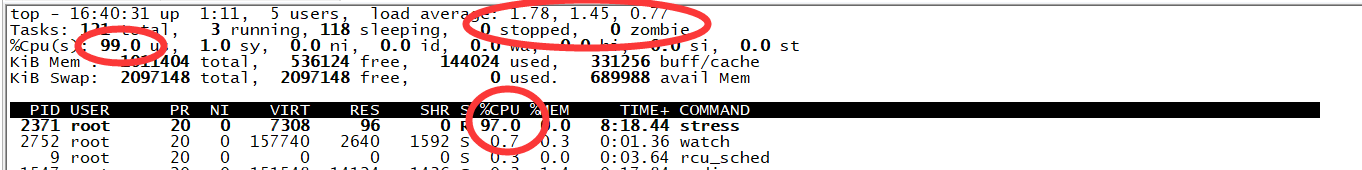


查看什么进程导致了CPU使用率过高

[kafka@demo1 ~]# pidstat -u 5 1

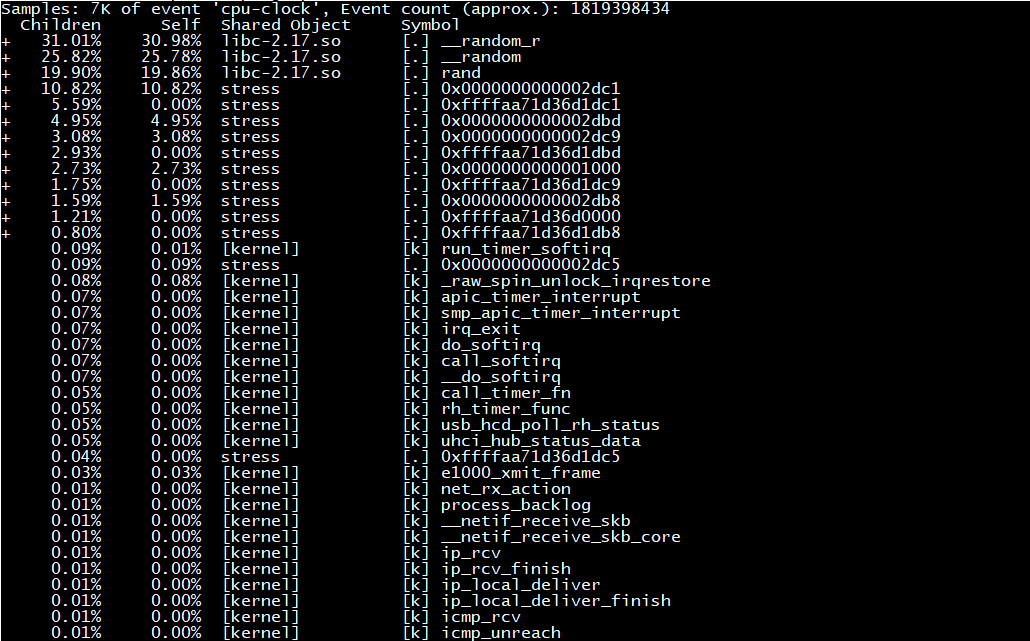


观察系统：



Perf分析性能问题

[kafka@demo01 ~]# perf top -g -p 2371



[root@demo01 ~]# perf record -g ## 记录性能事件，大约15s反ctrl+c退出

[root@demo01 ~]# perf report

# 内存性能评估

## 2.1内存监控命令free

[root@kafka01 ~]$ free

total used free shared buff/cache available

Mem: 263850960 45927492 47795596 117328 170127872 216526020

Swap: 67108860 1029300 66079560

free命令输出中显示了系统的各种内存状态，具体包括物理内存和swap，物理内存中又分为已使用内存（used）、空闲内存（free）、共享内存（shared）、系统缓存（buff/cache）和目前可用内存（available），我们可以直接通过查看available列的值，即可知道目前系统还有多少可用的物理内存。

## 2.2 top

按下m切换到内存排序

[root@kafka01 ~] top

KiB Mem : 8169348 total, 6871440 free, 267096 used, 1030812 buff/cache

KiB Swap: 0 total, 0 free, 0 used. 7607492 avail Mem

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

430 root 19 -1 122360 35588 23748 S 0.0 0.4 0:32.17 systemd-journal

1075 root 20 0 771860 22744 11368 S 0.0 0.3 0:38.89 snapd

1048 root 20 0 170904 17292 9488 S 0.0 0.2 0:00.24 networkd-dispat

1 root 20 0 78020 9156 6644 S 0.0 0.1 0:22.92 systemd

12376 azure 20 0 76632 7456 6420 S 0.0 0.1 0:00.01 systemd

12374 root 20 0 107984 7312 6304 S 0.0 0.1 0:00.00 sshd

top 输出界面的顶端，也显示了系统整体的内存使用情况比如 VIRT、RES、SHR 以及 %MEM 等。这些数据，包含了进程最重要的几个内存使用情况，我们挨个来看。

VIRT 是进程虚拟内存的大小，只要是进程申请过的内存，即便还没有真正分配物理内存，也会计算在内。

RES 是常驻内存的大小，也就是进程实际使用的物理内存大小，但不包括 Swap 和共享内存。

SHR 是共享内存的大小，比如与其他进程共同使用的共享内存、加载的动态链接库以及程序的代码段等。

%MEM 是进程使用物理内存占系统总内存的百分比。

模拟磁盘和文件写案例

生成500M的文件

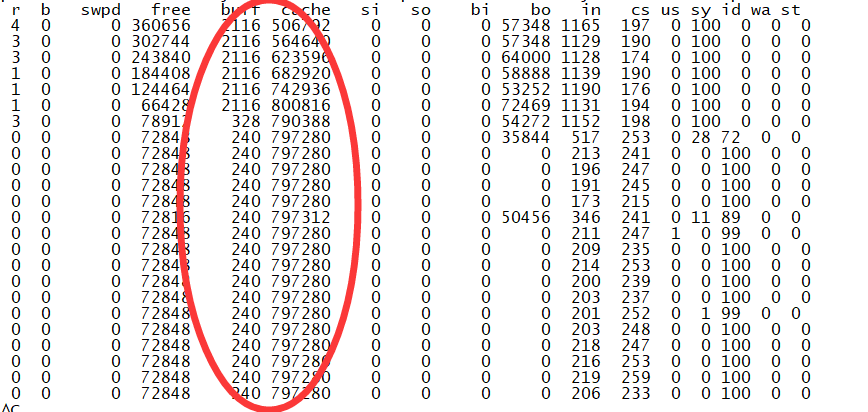
[root@kafka01 ~]# dd if=/dev/urandom of=/tmp/file bs=1M count=500

[root@kafka01 ~]# vmstat 1

内存部分的 buff 和 cache ，以及 io 部分的 bi 和 bo 就是我们要关注的重点。

buff 和 cache 就是我们前面看到的 Buffers 和 Cache，单位是 KB。

bi 和 bo 则分别表示块设备读取和写入的大小，单位为块/秒。因为 Linux 中块的大小是 1KB，所以这个单位也就等价于 KB/s。  
观察 vmstat 的输出，我们发现，在dd命令运行时， Cache在不停地增长，而Buffer基本保持不变。



再进一步观察I/O的情况，你会看到，

在 Cache 刚开始增长时，块设备 I/O 很少 bo，而过一段时间后，才会出现大量的块设备写 。

当 dd 命令结束后，Cache 不再增长，但块设备写还会持续一段时间，并且，多次 I/O 写的结果加起来，才是 dd 要写的 500M 的数据。

# IO性能评估

对磁盘性能评估，常用的工具有iotop、iostat、下面分别介绍如下：

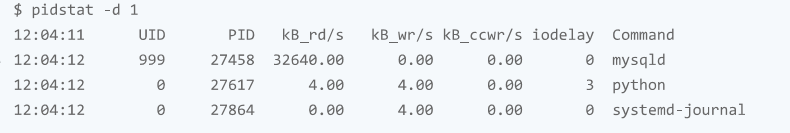
## 3.1磁盘监测工具Iotop命令

iotop是一个用来监视磁盘I/O使用状况的top类工具，可监测到哪一个程序使用的磁盘IO的实时信息。这对于线上业务系统来说非常有用。要使用这个工具，需要简单安装一下，在centos7.x版本下，直接执行yum在线安装即可：

[root@kafka02 ~]# yum -y install iotop

## 3.2 Mysql IO 案例

[root@kafka02 ~]# pidstat -d 1



从pidstat,可以分析PID=27458的mysqld进程正在进行大量的读操作。

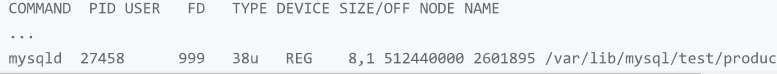
[root@kafka02 ~]# strace -f -p 27258

[root@kafka02 ~]# strace -f -T -tt -p 27258

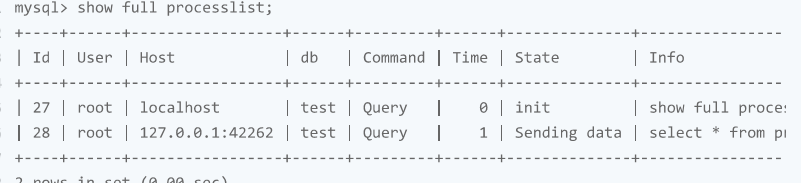


通过strace命令，可以分析28014正在读取文件标识符38的文件。

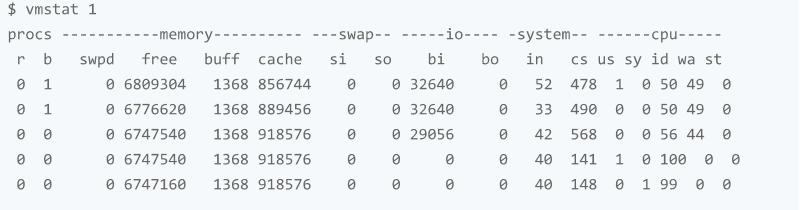
[root@kafka02 ~]# lsof -p 27458



找到读取的文件，输入show full processlist



建立索引

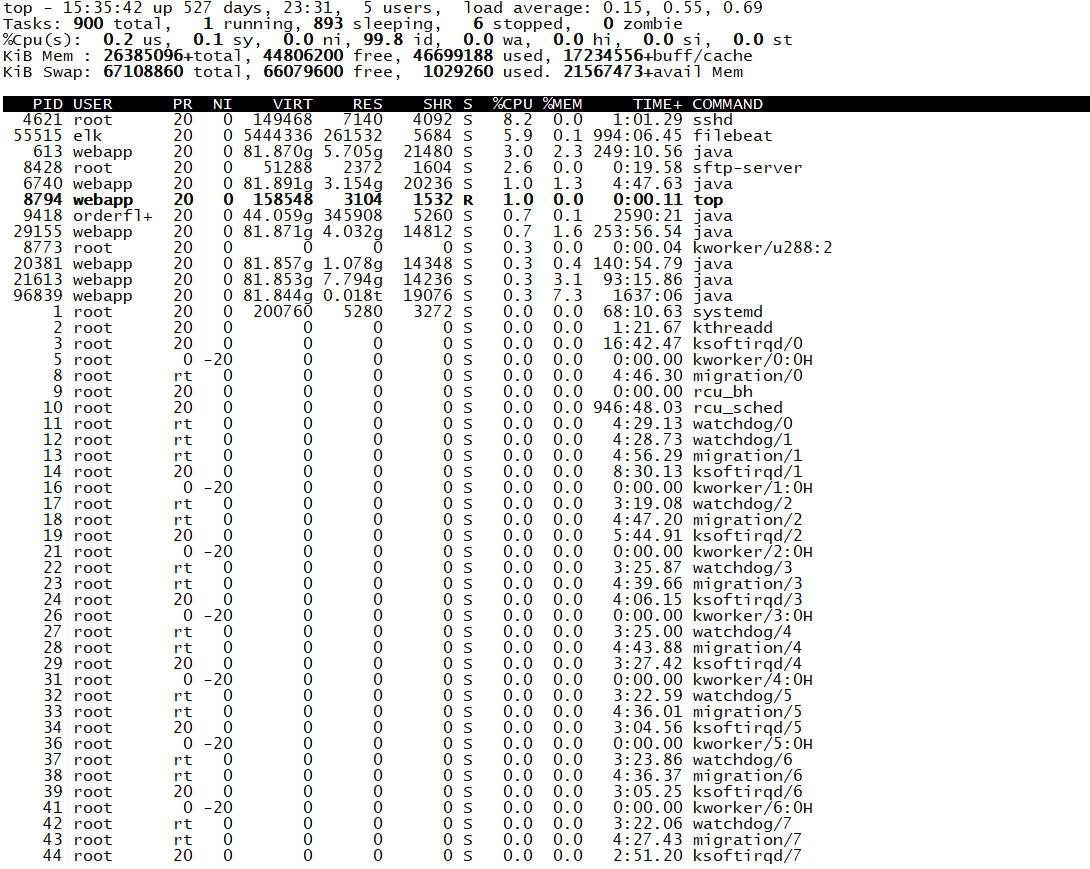


磁盘读（bi）和 iowait（wa）刚开始还是挺大的，但没过多久，就都变成了 0 。换句话说，I/O 瓶颈消失了

# 综合诊断评估工具

## 实时状态监控

top命令提供了实时的对系统处理器状态的监控，它能够实时显示系统中各个进程的资源占用状况。该命令可以按照对CPU的使用、内存使用和执行时间对系统任务进程进行排序显示，同时top命令还可以通过交互式命令进行设定显示。  
下面通过一个具体的例子，来解读top的输出内容，如下图所示：



从图中可以看到，top的输出可以分为统计信息区和进程信息区两个部分，即前5行显示为统计信息区，下面为进程信息区，我们分别进行介绍。

统计信息区

第1行为任务队列信息，含义如下：

15:38:33：表示当前系统时间。

up 527 days, 23:34：表示系统已经启动527天零23小时34分钟了。

5 users 当前登录系统的用户数。

lload average: 0.13, 0.33, 0.57：表示系统平均负载，3个数值分别表示1分钟、5分钟、15分钟前到现在的系统平均负载值。

第2行和第3行分别为进程和CPU信息，具体含义如下：

Tasks: 898 total：进程的总数。

1 running：正在运行的进程数。

891 sleeping：处于休眠的进程数。

6 stopped：停止的进程数。

0 zombie：僵死的进程数。

Cpu(s): 0.3% us：表示用户进程占用CPU的百分比。

0.1% sy：系统进程占用CPU的百分比。

0.0% ni：用户进程空间内改变过优先级的进程占用CPU百分比。

99.3% id：空闲CPU占用的百分比。

0.2% wa：等待输入输出的进程占用CPU的百分比。

最后两行输出的是内存信息，具体含义如下：

Mem: 26385096+ total：系统的物理内存大小。

46779456 used：已经使用的物理内存大小。

44615432 free：目前空余内存大小。

17245606 buffers：用作内核缓冲区的内存大小。

Swap: 67108860 total：交换分区内存大小。

1029232 used：已经使用的交换分区大小。

66079628 free：空闲的交换分区大小。

1029232 cached：高速缓存。

进程信息区

进程信息区显示了每个进程的运行状态，我们先来看一下每列输出的含义。

PID：进程的id。

USER：进程所有者的用户名。

PR：进程优先级。

NI：nice值。负值表示高优先级，正值表示低优先级。

VIRT：进程使用的虚拟内存总量，单位kB。VIRT=SWAP+RES。

RES：进程使用的、未被换出的物理内存大小，单位kB。RES=CODE+DATA。

SHR：共享内存大小，单位kB。

S：进程状态，D表示不可中断的睡眠状态，R表示运行状态，S表示睡眠状态，T表示跟踪/停止，Z表示僵死进程。

%CPU：上次更新到现在的CPU时间占用百分比。

%MEM：进程占用的物理内存百分比。

TIME+：进程使用的CPU时间总计，单位为1/100秒。

COMMAND：正在运行进程的命令名或者命令路径。

通过使用top命令，可以对系统整体运行状态有一个清晰的了解，结合上面介绍的CPU、磁盘、内存等工具，对于判断系统性能问题，完全可以给出一个正确的结论。

## top案例

1. 使用 top 命令，查找到使用 CPU 最多的某个进程，记录它的 pid。使用 Shift + P 快捷键可以按 CPU 的使用率进行排序。

Top

1. 再次使用 top 命令，加 -H 参数，查看某个进程中使用 CPU 最多的某个线程，记录线程的 ID。

top -Hp $pid

1. 定位线程问题（通过命令查看$pid进程的线程情况）

ps p pid -L -o pcpu,pmem,pid,tid,time,tname,cmd

1. 使用 printf 函数，将十进制的 tid 转化成十六进制。

printf %x $tid

1. 使用 jstack 命令，查看 Java 进程的线程栈。

jstack $pid >$pid.log

1. 使用 less 命令查看生成的文件，并查找刚才转化的十六进制 tid，找到发生问题的线程上下文。

less $pid.log

# 案例诊断01

package com.neusoft.study;

public class NewDemo01 {

public static void main(String[] args) {

byte[] bytes = null ;

for(int i = 0 ; i <=10; i ++){

bytes = new byte[6\*1020\*1024] ;

}

}

}

-Xms12m -Xmx12m -XX:+PrintGCDetails -XX:+PrintTenuringDistribution -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=D:\logs\

运行结果:

[GC (Allocation Failure)

Desired survivor size 524288 bytes, new threshold 7 (max 15)

[PSYoungGen: 1079K->512K(3584K)] 7199K->6840K(11776K), 0.0035654 secs] [Times: user=0.02 sys=0.01, real=0.00 secs]

[GC (Allocation Failure)

Desired survivor size 524288 bytes, new threshold 7 (max 15)

[PSYoungGen: 512K->512K(3584K)] 6840K->6856K(11776K), 0.0006653 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]

[Full GC (Allocation Failure) [PSYoungGen: 512K->0K(3584K)] [ParOldGen: 6344K->6784K(8192K)] 6856K->6784K(11776K), [Metaspace: 2666K->2666K(1056768K)], 0.0046388 secs] [Times: user=0.00 sys=0.00, real=0.01 secs]

[GC (Allocation Failure)

Desired survivor size 524288 bytes, new threshold 7 (max 15)

[PSYoungGen: 0K->0K(3584K)] 6784K->6784K(11776K), 0.0002564 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]

[Full GC (Allocation Failure) [PSYoungGen: 0K->0K(3584K)] [ParOldGen: 6784K->6772K(8192K)] 6784K->6772K(11776K), [Metaspace: 2666K->2666K(1056768K)], 0.0060093 secs] [Times: user=0.06 sys=0.00, real=0.01 secs]

java.lang.OutOfMemoryError: Java heap space

Dumping heap to D:\logs\java\_pid15752.hprof ...

Heap dump file created [7661373 bytes in 0.013 secs]

Exception in thread "main" java.lang.OutOfMemoryError: Java heap space

at com.neusoft.study.NewDemo01.main(NewDemo01.java:8)

Heap

PSYoungGen total 3584K, used 92K [0x00000000ffc00000, 0x0000000100000000, 0x0000000100000000)

eden space 3072K, 3% used [0x00000000ffc00000,0x00000000ffc17228,0x00000000fff00000)

from space 512K, 0% used [0x00000000fff00000,0x00000000fff00000,0x00000000fff80000)

to space 512K, 0% used [0x00000000fff80000,0x00000000fff80000,0x0000000100000000)

ParOldGen total 8192K, used 6772K [0x00000000ff400000, 0x00000000ffc00000, 0x00000000ffc00000)

object space 8192K, 82% used [0x00000000ff400000,0x00000000ffa9d298,0x00000000ffc00000)

Metaspace used 2697K, capacity 4486K, committed 4864K, reserved 1056768K

class space used 289K, capacity 386K, committed 512K, reserved 1048576K

分析，程序循环分配6M内存，gc full gc 后，无法继续分配6M内存，程序发生OOM挂掉，挂掉后生成dump信息到d:\logs目录

使用阿里的Arthas进行分析。或者jdk 自动的工具jmap or jconsole 进行内存泄漏分析，具体哪个对象占用了内存无法释放。

# 案例诊断02

问题现象  
传统ssm项目运行半个月突然无故挂了，重启后cpu，内存，io表现正常，再过1个月左右又挂掉，反复出现。

32g 服务器tomcat设置

server.xml文件的默认参数需要根据应用的特性进行适当的修改，例如可以修改“connectionTimeout“、“maxKeepAliveRequests”、“maxProcessors”等几个Tomcat配置文件的参数，适当加大这几个参数值。同时，对tomcat的JVM在内存方面进行优化，Tomcat内存优化主要是对tomcat启动参数优化，我们可以在tomcat的启动脚本TOMCAT\_HOME/bin/catalina.sh中增加如下内容：

–Xms8192m –Xmx8192m -Xmn1g -XX:PermSize=256M -XX:MaxPermSize=512m -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/jdk/dump/

-Xms：设置JVM初始堆内存为8192M

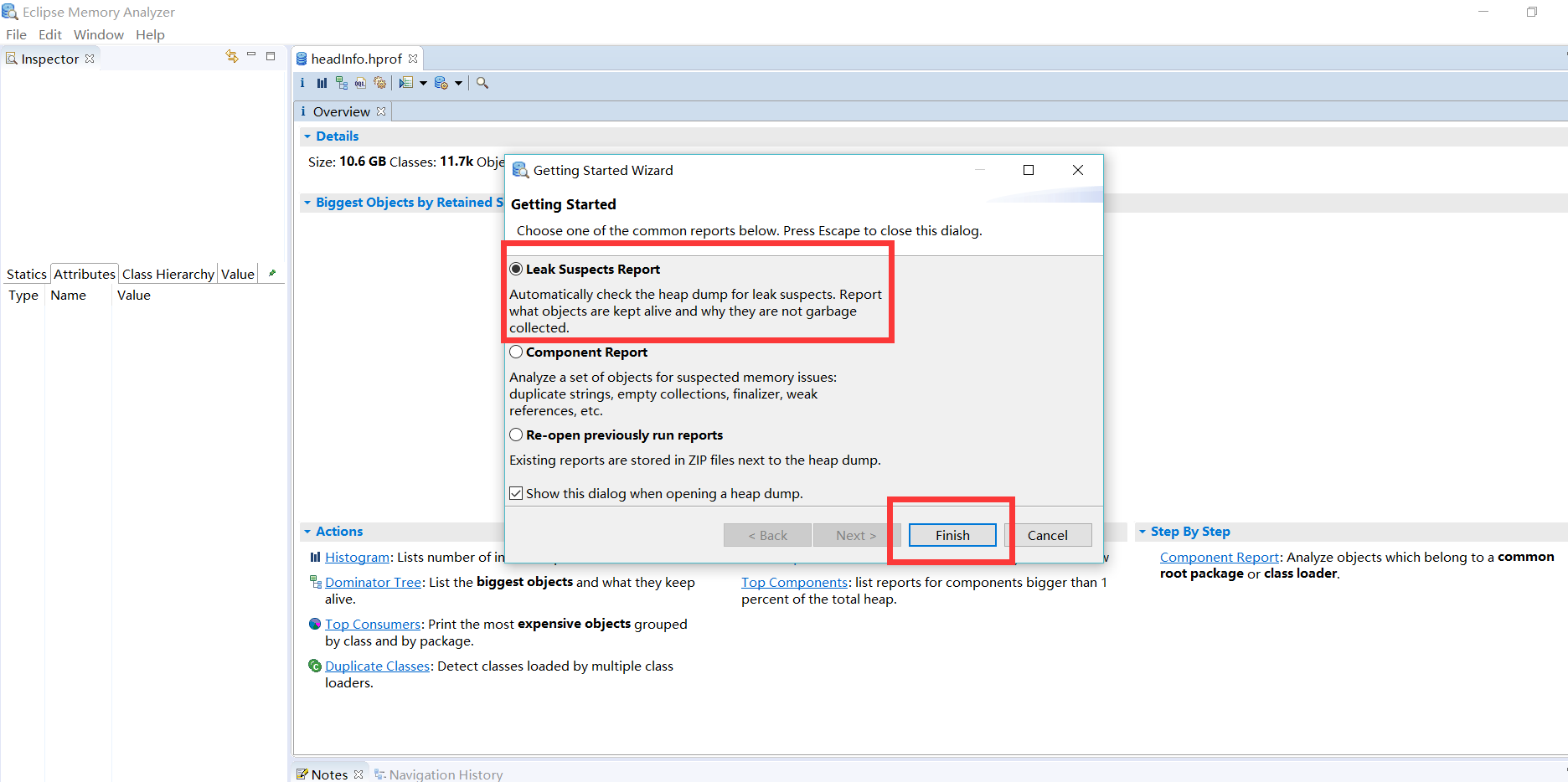
-Xmx：设置JVM最大堆内存为8192M

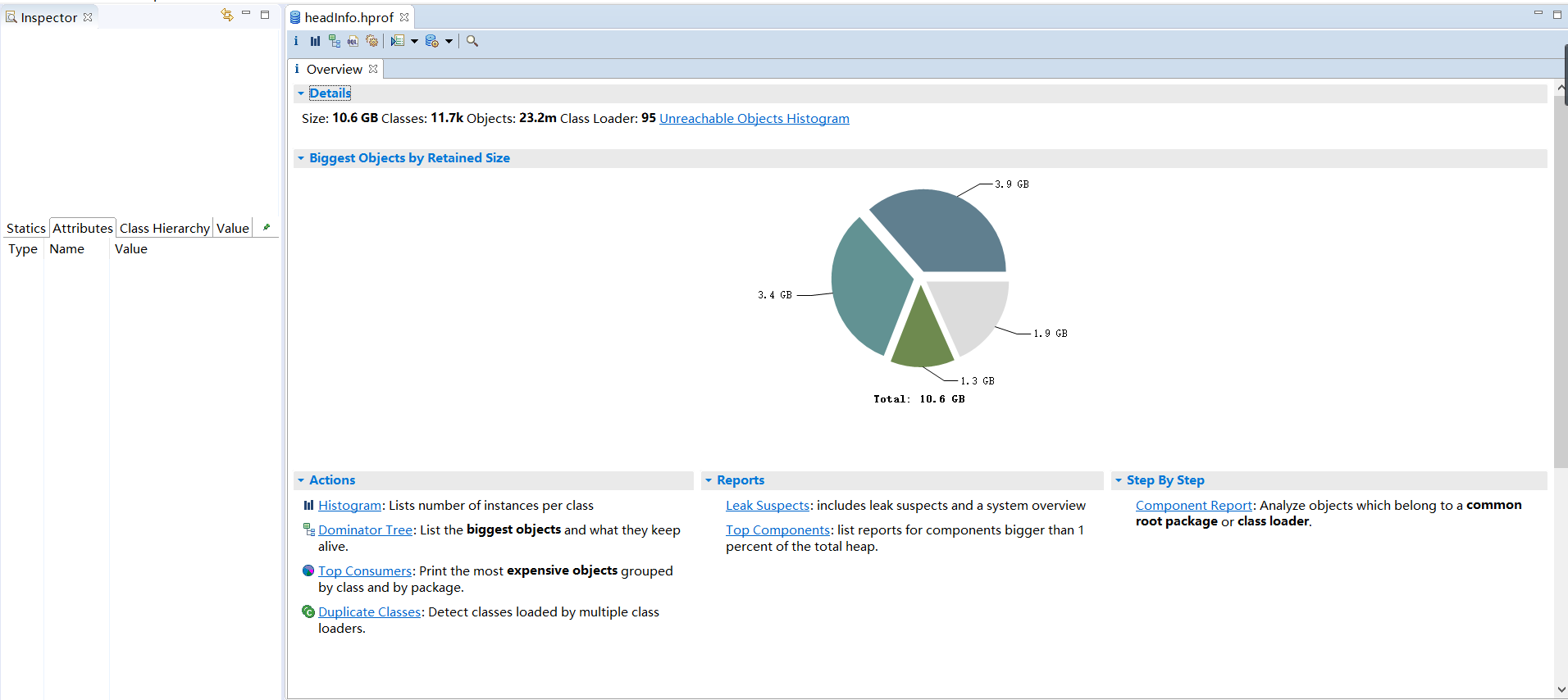
-XX:PermSize：设置堆内存持久代初始值为256M

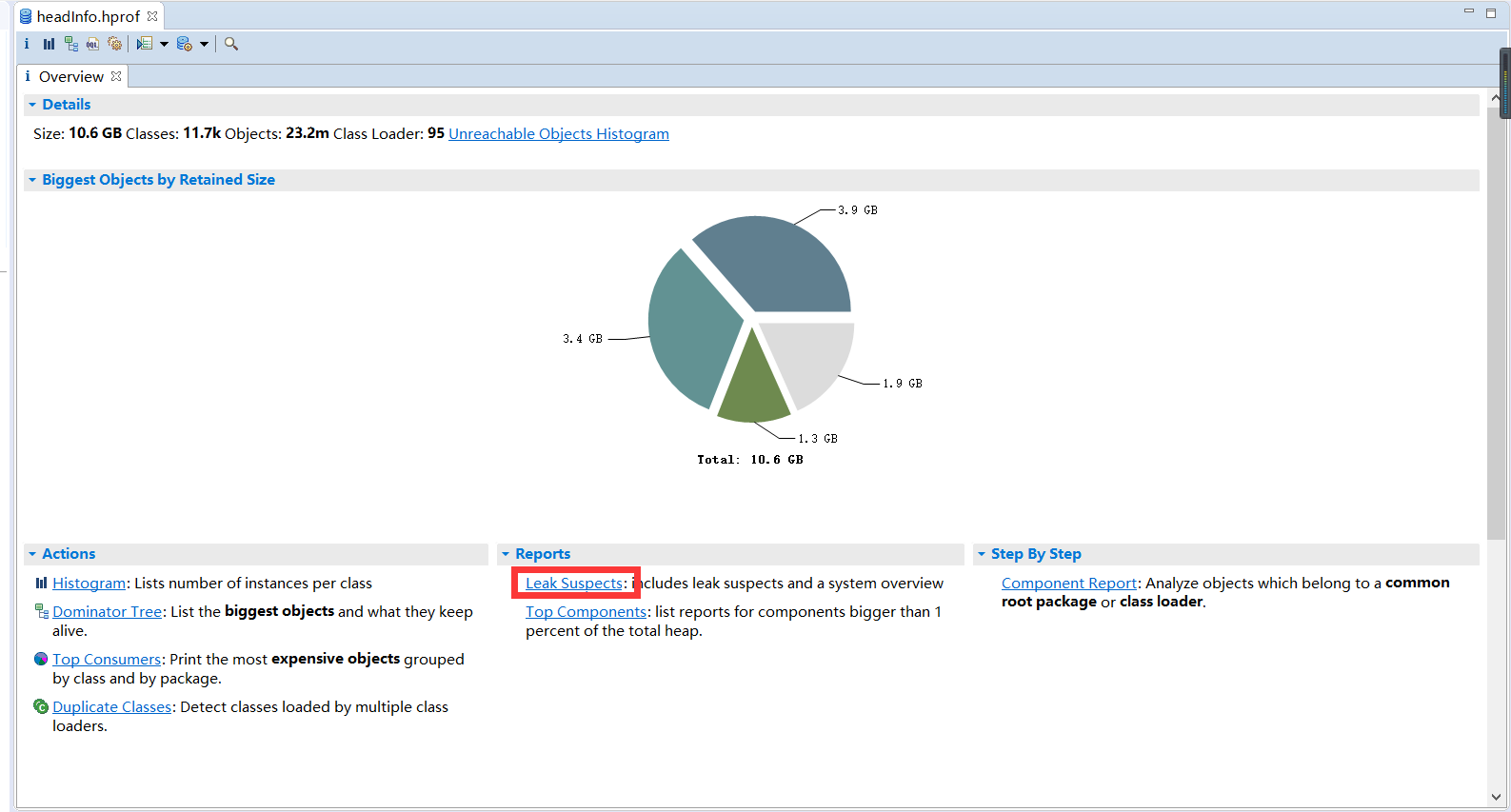
-XX:MaxPermSize：设置持久代最大值为512M  
-Xmn1g：设置堆内存年轻代大小为1G。整个堆内存大小 = 年轻代大小 + 年老代大小 + 持久代大小，持久代一般固定大小为64m，所以增大年轻代后，将会减小年老代大小。此值对系统性能影响较大。

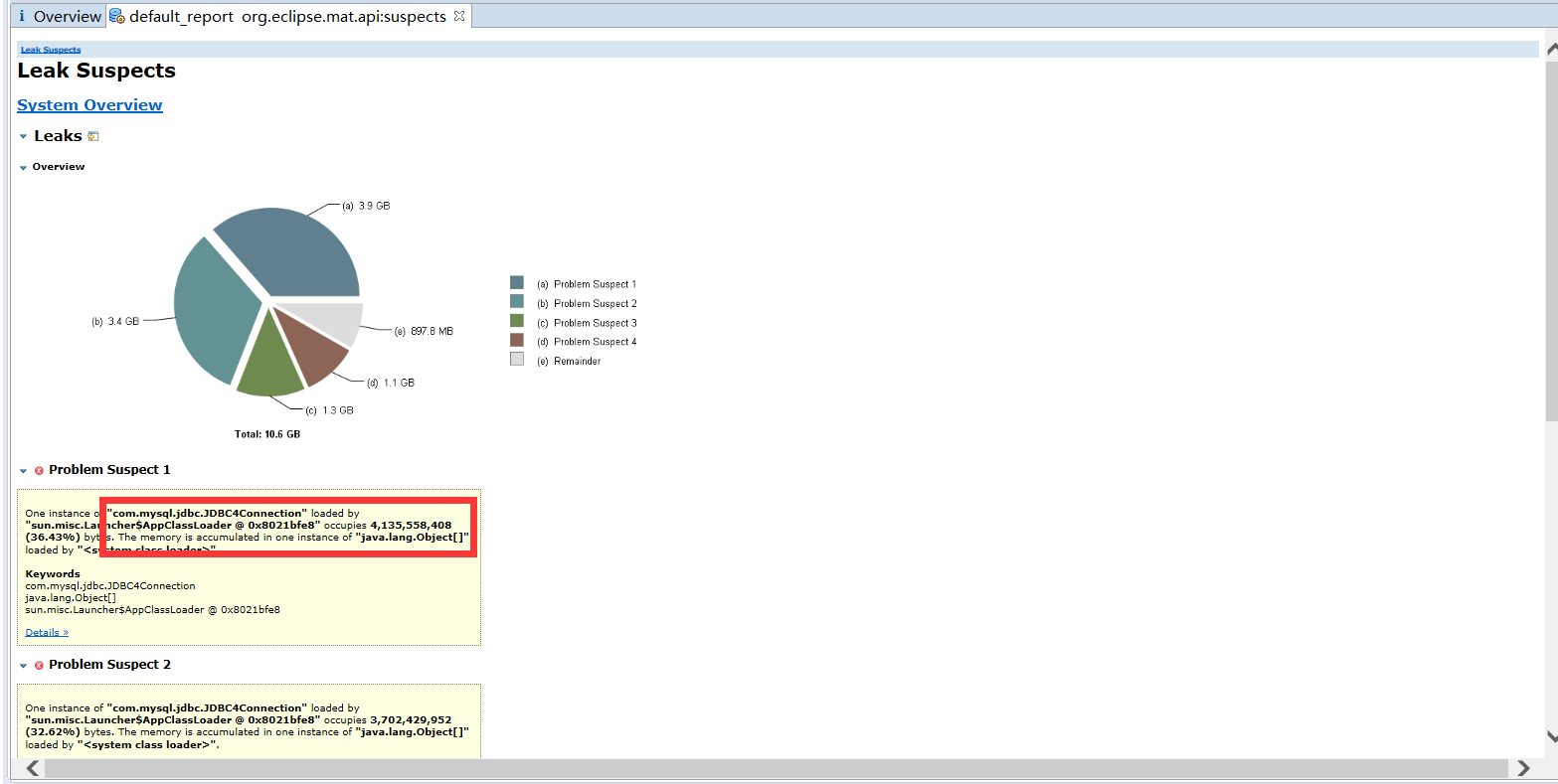
当出现oom时，dump到/jdk/dump/目录

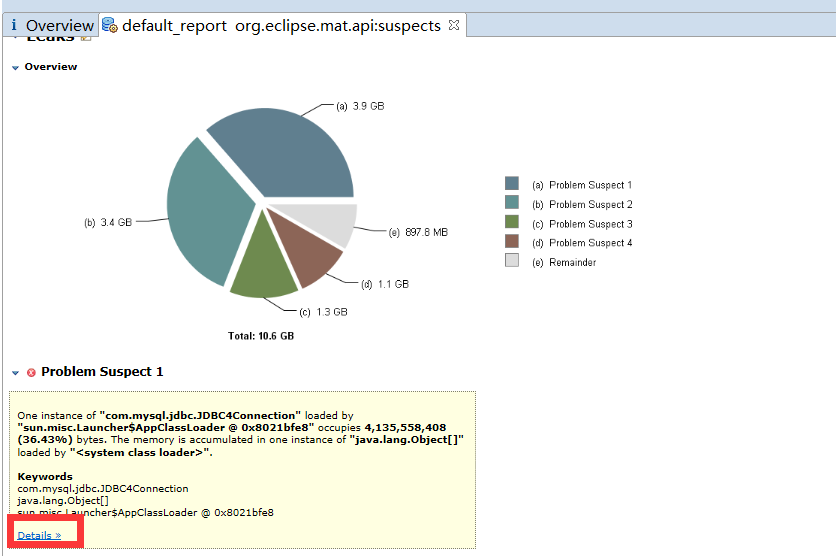
使用软件分析：

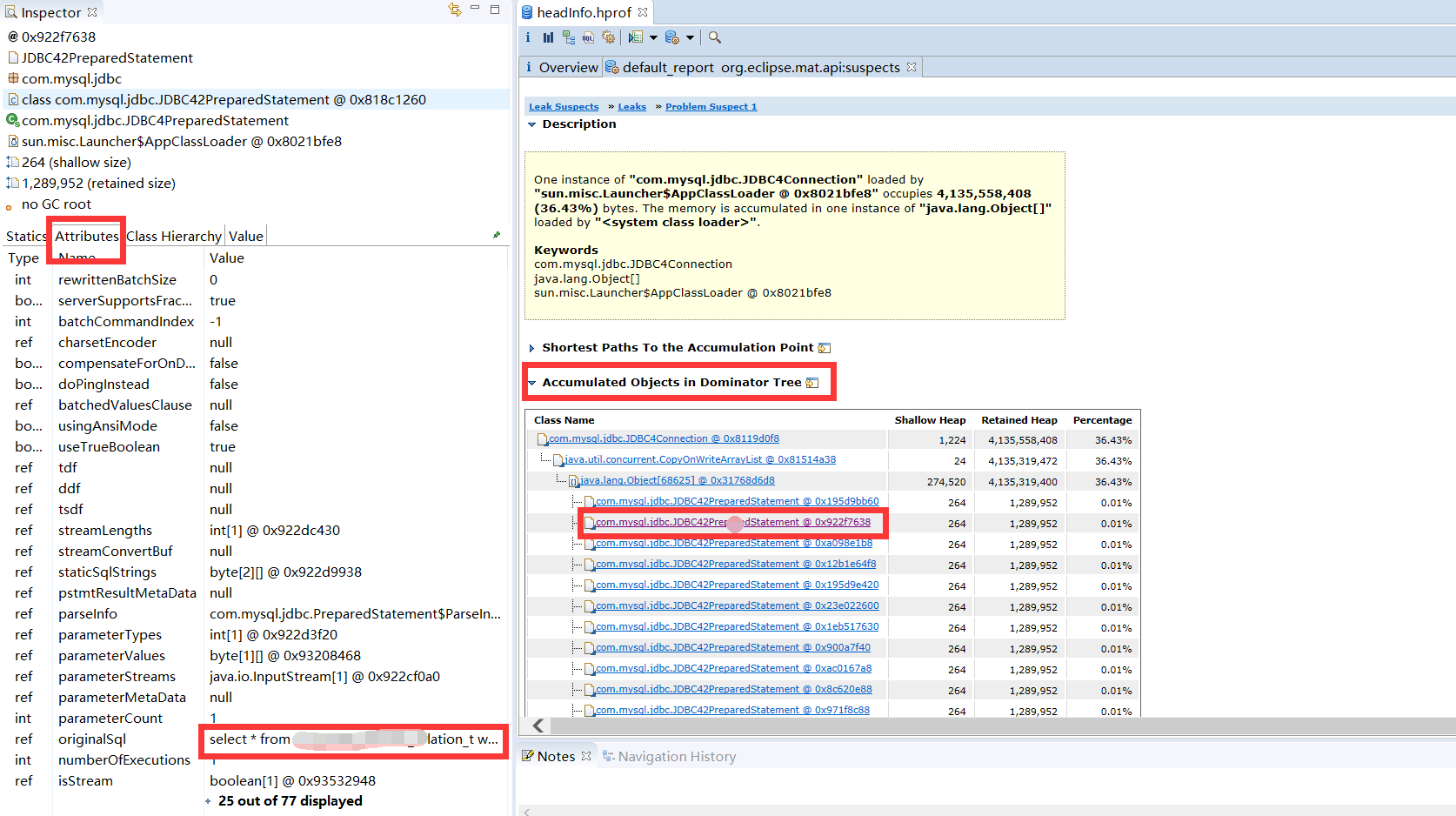




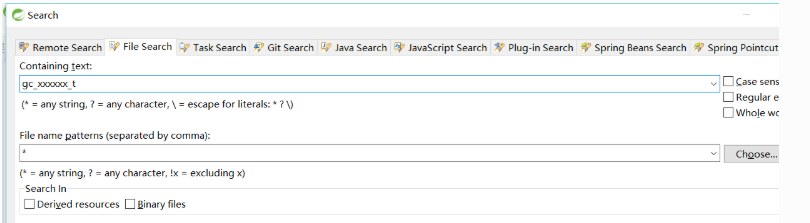








查到大量的PreparedStatement 停留在内存中，original Sql 是正在执行的SQL，在project中搜索此代码的dao，ctrl + s + h 打开代码搜索功能



查询到代码使用PreparedStatement预编译 不同的sql语句未释放，同时查询的这个表有大文本，导致PreparedStatement占用内存未及时回收。



使用jdbc，PreparedStatement每次使用需要及时close，不能一个方法预编译不同的sql语句。

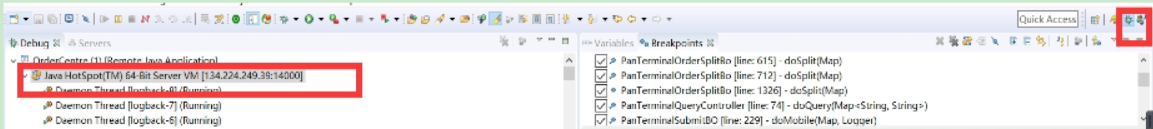
# 7远程debug

当你的代码本地开发环境与测试环境或者生产环境不一致时候，如何debug判断线上的问题？

1. 在服务器代码中以下配置，开启远程dbug端口14000

-Xdebug -Xrunjdwp:server=y,transport=dt\_socket,address=14000,suspend=n

1. .切换debug模式



3、发报文进入debug

