linux CPU 过高，怎么排查问题

ImportNew 2022-05-13 19:39

以下文章来源于潜行前行 ，作者cscw

潜行前行

分享技术总结，游戏心得，与大家在前行的路上登高望远

今天跟大家就CPU、磁盘、网络及内存方面的问题，聊聊如何排查和调优。

- CPU过高，怎么排查问题
- linux内存
- 磁盘IO
- 网络IO
- java 应用内存泄漏和频繁 GC
- java 线程问题排查
- 常用 jvm 启动参数调优

linux CPU 过高，怎么排查问题

CPU 指标解析

- 平均负载
  - 平均负载等于逻辑 CPU 个数，表示每个 CPU 都恰好被充分利用。如果平均负载大于逻辑 CPU 个数，则负载比较重
- 进程上下文切换
  - 无法获取资源而导致的自愿上下文切换
  - 被系统强制调度导致的非自愿上下文切换
- CPU 使用率
  - 用户 CPU 使用率，包括用户态 CPU 使用率（user）和低优先级用户态 CPU 使用率（nice），表示 CPU 在用户态运行的时间百分比。用户 CPU 使用率高，通常说明有应用程序比较繁忙
  - 系统 CPU 使用率，表示 CPU 在内核态运行的时间百分比（不包括中断），系统 CPU 使用率高，说明内核比较繁忙

- 等待 I/O 的 CPU 使用率，通常也称为 iowait，表示等待 I/O 的时间百分比。iowait 高，说明系统与硬件设备的 I/O 交互时间比较长
- 软中断和硬中断的 CPU 使用率，分别表示内核调用软中断处理程序、硬中断处理程序的时间百分比。它们的使用率高，表明系统发生了大量的中断

查看系统的平均负载

```
$ uptime
 10:54:52 up 1124 days, 16:31,  6 users,  load average: 3.67, 2.13, 1.79
```

- 10:54:52 是当前时间；up 1124 days, 16:31 是系统运行时间；6 users 则是正在登录用户数。而最后三个数字依次是过去 1 分钟、5 分钟、15 分钟的平均负载（Load Average）。平均负载是指单位时间内，系统处于可运行状态和不可中断状态的平均进程数
- 当平均负载高于 CPU 数量 70% 的时候，就应该分析排查负载高的问题。一旦负载过高，就可能导致进程响应变慢，进而影响服务的正常功能
- 平均负载与 CPU 使用率关系
  - CPU 密集型进程，使用大量 CPU 会导致平均负载升高，此时这两者是一致的
  - I/O 密集型进程，等待 I/O 也会导致平均负载升高，但 CPU 使用率不一定很高
  - 大量等待 CPU 的进程调度也会导致平均负载升高，此时的 CPU 使用率也会比较高

CPU 上下文切换

- 进程上下文切换：
  - 进程的运行空间可以分为内核空间和用户空间，当代码发生系统调用时（访问受限制的资源），CPU 会发生上下文切换，系统调用结束时，CPU 则再从内核空间换回用户空间。一次系统调用，两次 CPU 上下文切换
  - 系统平时会按一定的策略调用进程，会导致进程上下文切换
  - 进程在阻塞等到访问资源时，也会发生上下文切换
  - 进程通过睡眠函数挂起，会发生上下文切换
  - 当有优先级更高的进程运行时，为了保证高优先级进程的运行，当前进程会被挂起
- 线程上下文切换：
  - 同一进程里的线程，它们共享相同的虚拟内存和全局变量资源，线程上下文切换时，这些资源不变
  - 线程自己的私有数据，比如栈和寄存器等，需要在上下文切换时保存切换

- 中断上下文切换：
  - 为了快速响应硬件的事件，中断处理会打断进程的正常调度和执行，转而调用中断处理程序，响应设备事件

查看系统的上下文切换情况：

vmstat 和 pidstat。vmvmstat 可查看系统总体的指标，pidstat则详细到每一个进程服务的指标

```
$ vmstat 2 1
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    0 3498472 315836 3819540 0 0   0  1  2  0  3  1  96 0  0

--------
cs（context switch）是每秒上下文切换的次数
in（interrupt）则是每秒中断的次数
r（Running or Runnable）是就绪队列的长度，也就是正在运行和等待 CPU 的进程数.当这个值超过了CPU数目
b（Blocked）则是处于不可中断睡眠状态的进程数
```

```
# pidstat -w
Linux 3.10.0-862.el7.x86_64 (8f57ec39327b)      07/11/2021      _x86_64_        (6 CPU)

06:43:23 PM   UID       PID   cswch/s nvcswch/s  Command
06:43:23 PM     0         1      0.00      0.00  java
06:43:23 PM     0       102      0.00      0.00  bash
06:43:23 PM     0       150      0.00      0.00  pidstat

------各项指标解析----------------------------
PID         进程id
Cswch/s     每秒主动任务上下文切换数量
Nvcswch/s   每秒被动任务上下文切换数量。大量进程都在争抢 CPU 时，就容易发生非自愿上下文切换
Command     进程执行命令
```

怎么排查 CPU 过高问题

- 先使用 top 命令，查看系统相关指标。如需要按某指标排序则 使用 `top -o 字段名` 如：`top -o %CPU` 。 `-o` 可以指定排序字段，顺序从大到小

```
# top -o %MEM
```

```
top - 18:20:27 up 26 days,  8:30,  2 users,  load average: 0.04, 0.09, 0.13
Tasks: 168 total,   1 running, 167 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.3 us,  0.5 sy,  0.0 ni, 99.1 id,  0.0 wa,  0.0 hi,  0.1 si,  0.0 st
KiB Mem:  32762356 total, 14675196 used, 18087160 free,      884 buffers
KiB Swap:  2103292 total,        0 used,  2103292 free.  6580028 cached Mem

PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
2323 mysql     20   0 19.918g 4.538g   9404 S 0.333 14.52 352:51.44 mysqld
1260 root      20   0 7933492 1.173g  14004 S 0.333 3.753  58:20.74 java
1520 daemon    20   0  358140   3980    776 S 0.333 0.012   6:19.55 httpd
1503 root      20   0   69172   2240   1412 S 0.333 0.007   0:48.05 httpd

---------各项指标解析-----------------------------------------------------
第一行统计信息区
    18:20:27                        当前时间
    up 25 days, 17:29               系统运行时间，格式为时:分
    1 user                          当前登录用户数
    load average: 0.04, 0.09, 0.13  系统负载，三个数值分别为 1分钟、5分钟、15分钟前到现在的平均值

Tasks：进程相关信息
    running   正在运行的进程数
    sleeping  睡眠的进程数
    stopped   停止的进程数
    zombie    僵尸进程数
Cpu(s)：CPU相关信息
    %us：表示用户空间程序的cpu使用率（没有通过nice调度）
    %sy：表示系统空间的cpu使用率，主要是内核程序
    %ni：表示用户空间且通过nice调度过的程序的cpu使用率
    %id：空闲cpu
    %wa：cpu运行时在等待io的时间
    %hi：cpu处理硬中断的数量
    %si：cpu处理软中断的数量
Mem  内存信息
    total 物理内存总量
    used 使用的物理内存总量
    free 空闲内存总量
    buffers 用作内核缓存的内存量
Swap 内存信息
    total 交换区总量
    used 使用的交换区总量
    free 空闲交换区总量
    cached 缓冲的交换区总量
```

- 找到相关进程后，我们则可以使用 `top -Hp pid` 或 `pidstat -t -p pid` 命令查看进程具体线程使用 CPU 情况，从而找到具体的导致 CPU 高的线程
  - %us 过高，则可以在对应 java 服务根据线程ID查看具体详情，是否存在死循环，或者长时间的阻塞调用。java 服务可以使用 jstack
  - 如果是 %sy 过高，则先使用 strace 定位具体的系统调用，再定位是哪里的应用代码导致的

- 如果是 %si 过高，则可能是网络问题导致软中断频率飙高
- %wa 过高，则是频繁读写磁盘导致的。

linux 内存

查看内存使用情况

- 使用 top 或者 free、vmstat 命令

```
# top
top - 18:20:27 up 26 days,  8:30,  2 users,  load average: 0.04, 0.09, 0.13
Tasks: 168 total,   1 running, 167 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.3 us,  0.5 sy,  0.0 ni, 99.1 id,  0.0 wa,  0.0 hi,  0.1 si,  0.0 st
KiB Mem:  32762356 total, 14675196 used, 18087160 free,      884 buffers
KiB Swap:  2103292 total,        0 used,  2103292 free.  6580028 cached Mem

PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
2323 mysql     20   0 19.918g 4.538g   9404 S 0.333 14.52 352:51.44 mysqld
1260 root      20   0 7933492 1.173g  14004 S 0.333 3.753  58:20.74 java
....
```

- bcc-tools 软件包里的 cachestat 和 cachetop、memleak
  - achestat 可查看整个系统缓存的读写命中情况
  - cachetop 可查看每个进程的缓存命中情况
  - memleak 可以用检查 C、C++ 程序的内存泄漏问题

free 命令内存指标

```
# free -m
              total used   free   shared  buffers  cached
Mem:          32107 30414  1692   0       1962     8489
-/+ buffers/cache:    19962  12144
Swap:             0     0     0
```

- shared 是共享内存的大小，一般系统不会用到，总是0
- buffers/cache 是缓存和缓冲区的大小，buffers 是对原始磁盘块的缓存，cache 是从磁盘读取文件系统里文件的页缓存
- available 是新进程可用内存的大小

内存 swap 过高

Swap 其实就是把一块磁盘空间或者一个本地文件，当成内存来使用。swap 换出，把进程暂时不用的内存数据存储到磁盘中，并释放这些数据占用的内存。swap 换入，在进程再次访问这些内存的时候，把它们从磁盘读到内存中来

- swap 和内存回收的机制
  - 内存的回收既包括了文件页（内存映射获取磁盘文件的页）又包括了匿名页（进程动态分配的内存）
  - 对文件页的回收，可以直接回收缓存，或者把脏页写回磁盘后再回收
  - 而对匿名页的回收，其实就是通过 Swap 机制，把它们写入磁盘后再释放内存
- swap 过高会造成严重的性能问题，页失效会导致频繁的页面在内存和磁盘之间交换
  - 一般线上的服务器的内存都很大，可以禁用 swap
  - 可以设置 /proc/sys/vm/min_free_kbytes，来调整系统定期回收内存的阈值，也可以设置 /proc/sys/vm/swappiness，来调整文件页和匿名页的回收倾向

linux 磁盘I/O 问题

文件系统和磁盘

- 磁盘是一个存储设备（确切地说是块设备），可以被划分为不同的磁盘分区。而在磁盘或者磁盘分区上，还可以再创建文件系统，并挂载到系统的某个目录中。系统就可以通过这个挂载目录来读写文件
- 磁盘是存储数据的块设备，也是文件系统的载体。所以，文件系统确实还是要通过磁盘，来保证数据的持久化存储
- 系统在读写普通文件时，I/O 请求会首先经过文件系统，然后由文件系统负责，来与磁盘进行交互。而在读写块设备文件时，会跳过文件系统，直接与磁盘交互
- linux 内存里的 Buffers 是对原始磁盘块的临时存储，也就是用来缓存磁盘的数据，通常不会特别大（20MB 左右）。内核就可以把分散的写集中起来（优化磁盘的写入）
- linux 内存里的 Cached 是从磁盘读取文件的页缓存，也就是用来缓存从文件读写的数据。下次访问这些文件数据时，则直接从内存中快速获取，而不再次访问磁盘

磁盘性能指标

- 使用率，是指磁盘处理 I/O 的时间百分比。过高的使用率（比如超过 80%），通常意味着磁盘 I/O 存在性能瓶颈。

- 饱和度，是指磁盘处理 I/O 的繁忙程度。过高的饱和度，意味着磁盘存在严重的性能瓶颈。当饱和度为 100% 时，磁盘无法接受新的 I/O 请求。
- IOPS（Input/Output Per Second），是指每秒的 I/O 请求数
- 吞吐量，是指每秒的 I/O 请求大小
- 响应时间，是指 I/O 请求从发出到收到响应的间隔时间

IO 过高怎么找问题，怎么调优

- 查看系统磁盘整体 I/O

```
# iostat -x -k -d 1 1
Linux 4.4.73-5-default (ceshi44)        2021年07月08日  _x86_64_        (40 CPU)

Device:  rrqm/s   wrqm/s  r/s    w/s    rkB/s   wkB/s  avgrq-sz avgqu-sz await r_await w_aw
sda      0.08     2.48    0.37   11.71  27.80   507.24  88.53   0.02     1.34   14.96    0.
sdb      0.00     1.20    1.28   16.67  30.91   647.83  75.61   0.17     9.51    9.40    9.
------
rrqm/s:   每秒对该设备的读请求被合并次数，文件系统会对读取同块(block)的请求进行合并
wrqm/s:   每秒对该设备的写请求被合并次数
r/s:      每秒完成的读次数
w/s:      每秒完成的写次数
rkB/s:    每秒读数据量(kB为单位)
wkB/s:    每秒写数据量(kB为单位)
avgrq-sz: 平均每次IO操作的数据量(扇区数为单位)
avgqu-sz: 平均等待处理的IO请求队列长度
await:    平均每次IO请求等待时间(包括等待时间和处理时间，毫秒为单位)
svctm:    平均每次IO请求的处理时间(毫秒为单位)
%util:    采用周期内用于IO操作的时间比率，即IO队列非空的时间比率
```

- 查看进程级别 I/O

```
# pidstat -d
Linux 3.10.0-862.el7.x86_64 (8f57ec39327b)      07/11/2021      _x86_64_        (6 CPU)

06:42:35 PM   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s  Command
06:42:35 PM     0         1      1.05      0.00      0.00  java
06:42:35 PM     0       102      0.04      0.05      0.00  bash
------
kB_rd/s   每秒从磁盘读取的KB
kB_wr/s   每秒写入磁盘KB
kB_ccwr/s 任务取消的写入磁盘的KB。当任务截断脏的pagecache的时候会发生
Command   进程执行命令
```

- 当使用 pidstat -d 定位到哪个应用服务时，接下来则需要使用 strace 和 lsof 定位是哪些代码在读写磁盘里的哪些文件，导致IO高的原因

```
$ strace -p 18940
strace: Process 18940 attached
...
mmap(NULL, 314576896, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f0f7aee9(
mmap(NULL, 314576896, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f0f682e8(
write(3, "2018-12-05 15:23:01,709 - __main"..., 314572844
) = 314572844
munmap(0x7f0f682e8000, 314576896)       = 0
write(3, "\n", 1)                       = 1
munmap(0x7f0f7aee9000, 314576896)       = 0
close(3)                                = 0
stat("/tmp/logtest.txt.1", {st_mode=S_IFREG|0644, st_size=943718535, ...}) = 0
```

- strace 命令输出可以看到进程18940 正在往文件 /tmp/logtest.txt.1 写入300m

```
$ lsof -p 18940
COMMAND   PID USER   FD   TYPE DEVICE  SIZE/OFF    NODE NAME
java  18940 root  cwd    DIR   0,50      4096 1549389 /
…
java  18940 root    2u   CHR  136,0       0t0       3 /dev/pts/0
java  18940 root    3w   REG    8,1 117944320     303 /tmp/logtest.txt
----
FD 表示文件描述符号，TYPE 表示文件类型，NODE NAME 表示文件路径
```

- lsof 也可以看出进程18940 以每次 300MB 的速度往 /tmp/logtest.txt 写入

linux 网络I/O 问题

当一个网络帧到达网卡后，网卡会通过 DMA 方式，把这个网络包放到收包队列中；然后通过硬中断，告诉中断处理程序已经收到了网络包。接着，网卡中断处理程序会为网络帧分配内核数据结构（sk_buff），并将其拷贝到 sk_buff 缓冲区中；然后再通过软中断，通知内核收到了新的网络帧。内核协议栈从缓冲区中取出网络帧，并通过网络协议栈，从下到上逐层处理这个网络帧

- 硬中断：与系统相连的外设(比如网卡、硬盘)自动产生的。主要是用来通知操作系统系统外设状态的变化。比如当网卡收到数据包的时候，就会发出一个硬中断
- 软中断：为了满足实时系统的要求，中断处理应该是越快越好。linux为了实现这个特点，当中断发生的时候，硬中断处理那些短时间就可以完成的工作，而将那些处理事件比较长

的工作，交给软中断来完成

网络I/O指标

- 带宽，表示链路的最大传输速率，单位通常为 b/s（比特 / 秒）
- 吞吐量，表示单位时间内成功传输的数据量，单位通常为 b/s（比特 / 秒）或者 B/s（字节 / 秒）吞吐量受带宽限制，而吞吐量 / 带宽，也就是该网络的使用率
- 延时，表示从网络请求发出后，一直到收到远端响应，所需要的时间延迟。在不同场景中，这一指标可能会有不同含义。比如，它可以表示，建立连接需要的时间（比如 TCP 握手延时），或一个数据包往返所需的时间（比如 RTT）
- PPS，是 Packet Per Second（包 / 秒）的缩写，表示以网络包为单位的传输速率。PPS 通常用来评估网络的转发能力，比如硬件交换机，通常可以达到线性转发（即 PPS 可以达到或者接近理论最大值）。而基于 Linux 服务器的转发，则容易受网络包大小的影响
- 网络的连通性
- 并发连接数（TCP 连接数量）
- 丢包率（丢包百分比）

查看网络I/O指标

- 查看网络配置

```
# ifconfig em1
em1       Link encap:Ethernet  HWaddr 80:18:44:EB:18:98
          inet addr:192.168.0.44  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::8218:44ff:feeb:1898/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3098067963 errors:0 dropped:5379363 overruns:0 frame:0
          TX packets:2804983784 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1661766458875 (1584783.9 Mb)  TX bytes:1356093926505 (1293271.9 Mb)
          Interrupt:83
-----
TX 和 RX 部分的 errors、dropped、overruns、carrier 以及 collisions 等指标不为 0 时，
通常表示出现了网络 I/O 问题。
errors 表示发生错误的数据包数，比如校验错误、帧同步错误等
dropped 表示丢弃的数据包数，即数据包已经收到了 Ring Buffer，但因为内存不足等原因丢包
overruns 表示超限数据包数，即网络 I/O 速度过快，导致 Ring Buffer 中的数据包来不及处理（队列满）而
carrier 表示发生 carrirer 错误的数据包数，比如双工模式不匹配、物理电缆出现问题等
collisions 表示碰撞数据包数
```

- 网络吞吐和 PPS

```
# sar -n DEV 1
Linux 4.4.73-5-default (ceshi44)        2022年03月31日  _x86_64_        (40 CPU)

15时39分40秒     IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
15时39分41秒       em1   1241.00   1022.00    600.48    590.39      0.00      0.00    165.00
15时39分41秒        lo    636.00    636.00   7734.06   7734.06      0.00      0.00      0.00
15时39分41秒       em4      0.00      0.00      0.00      0.00      0.00      0.00      0.00
15时39分41秒       em3      0.00      0.00      0.00      0.00      0.00      0.00      0.00
15时39分41秒       em2     26.00     20.00      6.63      8.80      0.00      0.00      0.00
----
rxpck/s 和 txpck/s 分别是接收和发送的 PPS，单位为包 / 秒
rxkB/s 和 txkB/s 分别是接收和发送的吞吐量，单位是 KB/ 秒
rxcmp/s 和 txcmp/s 分别是接收和发送的压缩数据包数，单位是包 / 秒
```

- 宽带

```
# ethtool em1 | grep Speed
Speed: 1000Mb/s
```

- 连通性和延迟

```
# ping www.baidu.com
PING www.a.shifen.com (14.215.177.38) 56(84) bytes of data.
64 bytes from 14.215.177.38: icmp_seq=1 ttl=56 time=53.9 ms
64 bytes from 14.215.177.38: icmp_seq=2 ttl=56 time=52.3 ms
64 bytes from 14.215.177.38: icmp_seq=3 ttl=56 time=53.8 ms
64 bytes from 14.215.177.38: icmp_seq=4 ttl=56 time=56.0 ms
```

- 统计 TCP 连接状态工具 ss 和 netstat

```
[root@root ~]$>#ss -ant | awk '{++S[$1]} END {for(a in S) print a, S[a]}'
LISTEN 96
CLOSE-WAIT 527
ESTAB 8520
State 1
SYN-SENT 2
TIME-WAIT 660

[root@root ~]$>#netstat -n | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a]}'
CLOSE_WAIT 530
ESTABLISHED 8511
FIN_WAIT2 3
TIME_WAIT 809
```

网络请求变慢，怎么调优

- 高并发下 TCP 请求变多，会有大量处于 TIME_WAIT 状态的连接，它们会占用大量内存和端口资源。此时可以优化与 TIME_WAIT 状态相关的内核选项
  - 增大处于 TIME_WAIT 状态的连接数量 net.ipv4.tcp_max_tw_buckets ，并增大连接跟踪表的大小 net.netfilter.nf_conntrack_max
  - 减小 net.ipv4.tcp_fin_timeout 和 net.netfilter.nf_conntrack_tcp_timeout_time_wait ，让系统尽快释放它们所占用的资源
  - 开启端口复用 net.ipv4.tcp_tw_reuse。这样，被 TIME_WAIT 状态占用的端口，还能用到新建的连接中
  - 增大本地端口的范围 net.ipv4.ip_local_port_range 。这样就可以支持更多连接，提高整体的并发能力
  - 增加最大文件描述符的数量。可以使用 fs.nr_open 和 fs.file-max ，分别增大进程和系统的最大文件描述符数
- SYN FLOOD 攻击，利用 TCP 协议特点进行攻击而引发的性能问题，可以考虑优化与 SYN 状态相关的内核选项
  - 增大 TCP 半连接的最大数量 net.ipv4.tcp_max_syn_backlog ，或者开启 TCP SYN Cookies net.ipv4.tcp_syncookies ，来绕开半连接数量限制的问题
  - 减少 SYN_RECV 状态的连接重传 SYN+ACK 包的次数 net.ipv4.tcp_synack_retries
- 加快 TCP 长连接的回收，优化与 Keepalive 相关的内核选项
  - 缩短最后一次数据包到 Keepalive 探测包的间隔时间 net.ipv4.tcp_keepalive_time
  - 缩短发送 Keepalive 探测包的间隔时间 net.ipv4.tcp_keepalive_intvl
  - 减少 Keepalive 探测失败后，一直到通知应用程序前的重试次数 net.ipv4.tcp_keepalive_probes

java 应用内存泄漏和频繁 GC

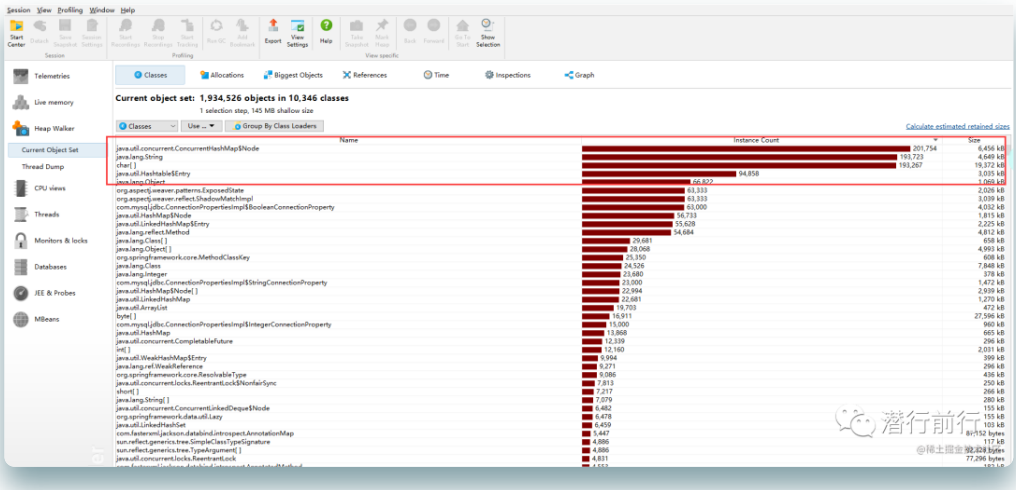
区分内存溢出、内存泄漏、内存逃逸

- 内存泄漏：内存被申请后始终无法释放，导致内存无法被回收使用，造成内存空间浪费
- 内存溢出：指内存申请时，内存空间不足
  - 1-内存上限过小
  - 2-内存加载数据太多
  - 3-分配太多内存没有回收，出现内存泄漏
- 内存逃逸：是指程序运行时的数据，本应在栈上分配，但需要在堆上分配，称为内存逃逸

- java中的对象都是在堆上分配的，而垃圾回收机制会回收堆中不再使用的对象，但是筛选可回收对象，回收对象还有整理内存都需要消耗时间。如果能够通过逃逸分析确定对象不会逃出方法之外，那就可以让这个对象在栈上分配内存，对象所占用的内存就可以随栈帧出栈而销毁，就减轻了垃圾回收的压力
- 线程同步本身比较耗时，如果确定一个变量不会逃逸出线程，无法被其它线程访问到，那这个变量的读写就不会存在竞争，对这个变量的同步措施可以清除
- java 虚拟机中的原始数据类型(int，long及reference类型等) 都不能再进一步分解，它们称为标量。如果一个数据可以继续分解，那它称为聚合量，java 中最典型的聚合量是对象。如果逃逸分析证明一个对象不会被外部访问，并且这个对象是可分解的，那程序运行时可能不创建该对象，而改为直接创建它的若干个被方法使用到的成员变量来代替。拆散后的变量便可以被单独分析与优化，可以各自分别在栈帧或寄存器上分配空间，原本的对象就无需整体分配空间

内存泄漏，该如何定位和处理

- 使用 `jmap -histo:live [pid]` 和 `jmap -dump:format=b,file=filename [pid]` 前者可以统计堆内存对象大小和数量，后者可以把堆内存 dump 下来
- 启动参数中指定 `-XX:+HeapDumpOnOutOfMemoryError` 来保存OOM时的dump文件
- 使用 JProfiler 或者 MAT 软件查看 heap 内存对象，可以更直观地发现泄露的对象



java线程问题排查

java 线程状态

- NEW：对应没有 Started 的线程，对应新生态
- RUNNABLE：对于就绪态和运行态的合称
- BLOCKED，WAITING，TIMED_WAITING：三个都是阻塞态
  - sleep 和 join 称为WAITING

  - sleep 和 join 方法设置了超时时间的，则是 TIMED_WAITING
  - wait 和 IO 流阻塞称为BLOCKED
- TERMINATED：死亡态

线程出现死锁或者被阻塞

- `jstack -l pid | grep -i -E 'BLOCKED | deadlock'` ， 加上参数 -l，jstack 命令可以快速打印出造成死锁的代码

```
# jstack -l 28764
Full thread dump Java HotSpot(TM) 64-Bit Server VM (13.0.2+8 mixed mode, sharing):
.....
"Thread-0" #14 prio=5 os_prio=0 cpu=0.00ms elapsed=598.37s tid=0x000001b3c25f7000 nid=0x4ab
   java.lang.Thread.State: BLOCKED (on object monitor)
        at com.Test$DieLock.run(Test.java:52)
        - waiting to lock <0x0000000712d7c230> (a java.lang.Object)
        - locked <0x0000000712d7c220> (a java.lang.Object)
        at java.lang.Thread.run(java.base@13.0.2/Thread.java:830)

   Locked ownable synchronizers:
        - None

"Thread-1" #15 prio=5 os_prio=0 cpu=0.00ms elapsed=598.37s tid=0x000001b3c25f8000 nid=0x198
   java.lang.Thread.State: BLOCKED (on object monitor)
        at com.Test$DieLock.run(Test.java:63)
        - waiting to lock <0x0000000712d7c220> (a java.lang.Object)
        - locked <0x0000000712d7c230> (a java.lang.Object)
        at java.lang.Thread.run(java.base@13.0.2/Thread.java:830)
.....
Found one Java-level deadlock:
=============================
"Thread-0":
  waiting to lock monitor 0x000001b3c1e4c480 (object 0x0000000712d7c230, a java.lang.Object
  which is held by "Thread-1"
"Thread-1":
  waiting to lock monitor 0x000001b3c1e4c080 (object 0x0000000712d7c220, a java.lang.Object
  which is held by "Thread-0"

Java stack information for the threads listed above:
===================================================
"Thread-0":
        at com.Test$DieLock.run(Test.java:52)
        - waiting to lock <0x0000000712d7c230> (a java.lang.Object)
        - locked <0x0000000712d7c220> (a java.lang.Object)
        at java.lang.Thread.run(java.base@13.0.2/Thread.java:830)
"Thread-1":
        at com.Test$DieLock.run(Test.java:63)
        - waiting to lock <0x0000000712d7c220> (a java.lang.Object)
        - locked <0x0000000712d7c230> (a java.lang.Object)
```

```
        at java.lang.Thread.run(java.base@13.0.2/Thread.java:830)
Found 1 deadlock.
```

- 从 jstack 输出的日志可以看出线程阻塞在 Test.java:52 行，发生了死锁

常用 jvm 调优启动参数

- -verbose:gc 输出每次GC的相关情况
- -verbose:jni 输出native方法调用的相关情况，一般用于诊断jni调用错误信息
- -Xms n 指定jvm堆的初始大小，默认为物理内存的1/64，最小为1M；可以指定单位，比如k、m，若不指定，则默认为字节
- -Xmx n 指定jvm堆的最大值，默认为物理内存的1/4或者1G，最小为2M；单位与-Xms一致
- -Xss n 设置单个线程栈的大小，一般默认为512k
- -XX:NewRatio=4 设置年轻代（包括Eden和两个Survivor区）与年老代的比值（除去持久代）。设置为4，则年轻代与年老代所占比值为1：4，年轻代占整个堆栈的1/5
- -Xmn 设置新生代内存大小。整个堆大小=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，Sun官方推荐配置为整个堆的3/8
- -XX:SurvivorRatio=4 设置年轻代中Eden区与Survivor区的大小比值。设置为4，则两个Survivor区与一个Eden区的比值为2:4，一个Survivor区占整个年轻代的1/6
- -XX:MaxTenuringThreshold=0 设置垃圾最大年龄。如果设置为0的话，则年轻代对象不经过Survivor区，直接进入年老代。对于年老代比较多的应用，可以提高效率。如果将此值设置为一个较大值，则年轻代对象会在Survivor区进行多次复制，这样可以增加对象再年轻代的存活时间，增加在年轻代即被回收的概率

- EOF -

推荐阅读 — 点击标题可跳转

太极限了，JDK的这个BUG都能被我踩到

船新 IDEA 2022.1 正式发布，新特性真香！

Java中9种常见的CMS GC问题分析与解决

发表于广东

喜欢此内容的人还喜欢

号称最强深度学习笔记本电脑，雷蛇与Lambda公司推出，售价超2万

机器之心

Python 3.11 性能测评超 3.10 近 64%

21CTO

发布日志记录、公开所有代码，Meta开放1750亿参数大模型，媲美GPT-3

机器之心