排查方案

> 现象：应用无法访问不，后台服务器有两种状况导致该现象的发生

前置条件：网络通信处于正常状态

可用过以下方式进行验证：

- ping ip ： 通过ping命令 进行网络的判断，如果有数据包进行返回，说明服务器直接网络通信正常
- tenlet ip 端口 ： 通过telnet 命令，测试端口状态是否异常

如果以上情况数据返回正常，可以判定网络通信属于正常状态

排查过程：

- 首先判断查看进程是否存在

查看金蝶进程是否存在的命令 Java

```
ps aux | grep apusic
```

1. 金蝶中间件的进程不存在,出现宕机

大致原因：

JVM发生OOM而产生的宕机，内存持续飙高，导致被操作系统kill掉

如果出现宕机，中间件停止运行，此时cpu、内存、资源都已经释放，只能通过日志文件信息来寻找原因

1、正常情况下**JVM会生成了hs_err_pid.log**这样的文件，其中包含了虚拟机崩溃原因的重要信息

查看**mydomain目录**下是否有生成了宕机过后的**hs_err.log文件**，如果有，收集相关的日志信息

2、查看**中间件日志文件**，里面将会记录宕机前的相关信息，以及应用系统的异常信息，可以收集相关的日志信息，

日志文件路径：**/aas/domains/mydomain/logs/apusic.log.0**

3、可以查看操作系统日志/var/log/messages ，可以收集相关的日志信息

> 用grep “Out of memory” /var/log/messages 查看Linux系统触发oom-killer机制杀死占用内存过大的进程

4、无任何性能文件产生：

宕机的时候必定会产JavaCore及HeapDump文件，默认不会保留，如需保留并输出到指定路径，再结合工具对dump文件进行分析

设置两个JVM参数：

–XX:+HeapDumpOnOutOfMemoryError –XX:HeapDumpPath=D:\study\log_hprof\gc.hprof

注：

–XX:+HeapDumpOnOutOfMemoryError 参数表示当JVM发生OOM时，自动生成DUMP文件，记录了jvm即将崩溃时的内存快照

–XX:HeapDumpPath=${目录}参数表示生成dump文件的路径，也可以指定文件名称

2. 金蝶中间件进程存在,出现"假死"状态

此时，可判断中间件运行正常，应用程序出现了异常，具体原因需要进行排查

> 可以通过浏览器对中间件管理界面进行访问，对应用进行访问，如果后台进程存在，但是无响应，大概率是出现了jvm虚拟机已经处于"假死"状态，jvm正在做自我修复，自我垃圾清理，处理http请求的相关线程处于阻塞状态，无法响应到客户端

排查过程:

1、查看进程占用的cpu、内存状态：

获取进程的pid：ps aux | grep apusic

查看cpu 、内存状态：top –p pid(进程号)

示例：

```
root      90750  0.0  0.0 112824   984 pts/0    S+   14:47   0:00 grep --color=auto apusic
[root@k8s-nodes2 ~]# top -p 3755
top - 14:50:40 up 11 days, 20:49,  2 users,  load average: 0.89, 0.83, 0.84
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.7 us,  2.2 sy,  0.0 ni, 97.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  1862816 total,    90444 free,   465044 used,  1307328 buff/cache
KiB Swap:        0 total,        0 free,        0 used.  1125056 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 3755 root      20   0  149484  28760  13580 S   3.0  1.5 655:36.49 calico-node
```

2、收集中间件日志文件：/aas/domains/mydomain/logs/apusic.log.0 ，可以分析里面是否有异常信息，应用程序的异常信息也会在此打印

3、**捕抓堆栈信息**：将堆栈信息进行收集

采集日志相关命令：**jstack 进程号>/home/jstack01.txt** （捕抓堆栈日志时机为正在发生第一时间，确保信息准确）尽可能捕抓多个堆栈日志快照日志（3–10个）

4、查看当前java进程jvm**虚拟机垃圾回收状态是否异常**：

查看进程gc情况相关命令 ： jstat –gcutil pid 1000

说明：当前jvm虚拟机在频繁进行fullgc操作，导致程序一段时间无响应，可以推断出应用程序出现内存泄漏，需要抓取内存dump，分析heap.hprof 文件

5、看当前活动线程数是否到达最大线程数,如果达到最大线程数，新的请求将会被阻塞，无法响应客户端请求

统计当前线程数：

```shell
[@localhost ~]$ jstack -l 22529(pid) | grep 'java.lang.Thread.State' | wc -l
70
```

统计 RUNNABLE 的线程数

```shell
[@localhost ~]$ jstack -l 22529 | grep 'java.lang.Thread.State: RUNNABLE' | wc -l
15
```

6、对**内存信息**进行分析：

下载案发现场的**heap.hprof** （容量比价大，大概5g左右），下载耗时

下载命令： **jmap –dump:live,format=b,file=/usr/local/heap.hprof pid** 生成dump文件了

使用工具对内存进行分析（mat工具或者jproflier）进行分析，

此处使用jproflier工具进行对heap.hpro 分析