Linux命令及问题排查

CPU占用100%

1. top -c，显示进程运行信息列表。按下P按照cpu使用率排序，找到pid
2. top -Hp #{pid}，根据pid查出cpu消耗最高的线程号。按P按照cpu使用率排序
3. printf "%x\n" tid，此时线程号为十进制，转为十六进制
4. jstack -l #{十进制pid} > ./#{十进制pid}.stack，导出进程快照
5. cat #{十进制pid} | grep '#{十六进制pid}' -C 8，根据第三步获得的十六进制线程PID使用grep命令进行查找
6. 从查询得到的堆栈信息中即可定位有问题的代码

内存问题

1. **堆溢出**
    - 查找关键报错信息，比如"java.lang.OutOfMemoryError: Java heap space"
    - 使用内存映像分析工具（MAT、Jprofiler）对Dump出来的堆存储快照进行分析，分析清除是**内存泄漏**还是**内存溢出**
    - 如果是**内存泄漏**，可进一步通过工具查看泄露对象到GC Roots的引用链，修复应用程序中的内存泄漏
    - 如果不存在泄漏，先检查代码是否有死循环，递归等，在考虑增加堆大小
2. **栈溢出**
    - 查找关键报错信息，确定是StackOverFlowError还是OutOfMemoryError
    - 如果是StackOverflowError，检查代码是否递归调用方法等
    - 如果是OutOfMemoryError，检查是否有死循环创建线程等，可以通过-Xss降低每个线程栈大小的容量
3. **方法区溢出**

    又称永久代，JDK8之后元空间替换了永久代。用于存放Class的相关信息，运行时产生大量的类，会填满方法区，造成溢出。

溢出原因：

- 使用CGLib生成了大量的代理类，导致方法区被撑爆
- 大量jsp和动态产生jsp
- 应用长时间运行，没有重启

排查解决思路：

- 检查是否永久代空间设置得过小
- 检查是否跟jsp有关
- 检查是否使用CGLib生成了大量的代理类
- 重启JVM

可用命令

jstat -gcutil pid interval(ms)

jmap -dump:live,format=b,file=myjmapfile.txt pid