1、面试题

jvm栈在哪些情况下会溢出？java堆在哪些情况下会溢出？做过哪些jvm优化？用了哪些方法？达到了什么样的效果？jvm问题（内存泄露、线程卡死、jvm崩溃、内存溢出、频繁gc），该如何定位和排查（jmap和jstack）？

2、面试官心里分析

这块其实是jvm面试必问，因为其实基础的jvm知识大家都知道，什么jvm区域划分，什么时候回收了，gc算法了，分代gc，几种垃圾回收器之类的，包括类加载之类的。但是其实在线上，我们经常遇到的jvm问题就两种，一种是oom，就是内存溢出了；另外一种是频繁gc导致系统一直卡顿。

所以如果是我，也肯定会问问你平时对于oom问题，如果发生之后，你怎么排查、定位和解决的；如果出现频繁gc，你怎么排查、定位和解决的，这个是高级工程师很重要的线上问题解决能力。

3、面试题剖析

**3.1 线上oom问题**

常见的问题是突然之间进程就没了，就是线上的java进程跑着跑着就没了。

有个一个命令：dmesg |grep -E ‘kill|oom|out of memory’，可以查看操作系统启动后的系统日志，这里就是查看跟内存溢出相关联的系统日志。如果你确实是oom导致进程被杀死了，那么系统日志里会出现这样的字眼：Out of memory，kill process 13987（java）。。。。。意思就是说，OOM了，然后就杀了你的那个java进程。那么至少用这个命令查看系统日志，就可以确认是OOM问题发生了。

如果一旦oom，就会导致你的jvm死掉，然后你肯定得重启吧。可能你重启之后，先用ps -aux|grep java命令查看一下你的java进程，就可以找到你的java进程的进程id。然后你可以用top命令看一下，top命令显示的结果列表中，会看到%MEM这一列，这里可以看到你的进程可能对内存的使用率特别高。

接着，可以用jstat -gcutil 20886 1000 10命令，就是用jstat工具，对指定java进程（20886就是进程id，通过ps -aux | grep java命令就能找到），按照指定间隔，看一下统计信息，这里会每隔一段时间显示一下，包括新生代的两个Survivor区、Eden区，以及老年代的内存使用率，还有young gc以及full gc的次数。

看到的东西类似下面那样：

S0 S1 E O YGC FGC

26.80 0.00 10.50 89.90 86 954

其实如果大家了解原理，应该知道，一般来说大量的对象涌入内存，结果始终不能回收，会出现的情况就是，快速撑满年轻代，然后young gc几次，根本回收不了什么对象，导致survivor区根本放不下，然后大量对象涌入老年代。老年代很快也满了，然后就频繁full gc，但是也回收不掉。

然后对象持续增加不就oom了，内存放不下了，爆了呗。

所以jstat先看一下基本情况，马上就能看出来，其实就是大量对象没法回收，一直在内存里占据着，然后就差不多内存快爆了。

接着就是用jmap来一把，jmap -histo:live 20886，看看现在现在java进程里的对象分布情况，就是根据每个对象占用内存从大到小来排列的，你可能会看到下面的东西：

1: 485009 489005612 [B

2: 2609794 98034943 [C

。。。。N多行

其实你一下就可以发现是哪些对象占据内存过多了

接着就是要获取一个堆内存快照了，jmap -dump:format=b,file=文件名 [pid]，就可以把指定java进程的堆内存快照搞到一个指定的文件里去，但是jmap -dump:format其实一般会比较慢一些，也可以用gcore工具来导出内存快照。

先在linux上安装gdb，这个自己百度一下就好，很多的

gdb -q --20886　//启动gdb命令

(gdb) generate-core-file //这里调用命令生成gcore的dump文件

(gdb) gcore /tmp/dump.core　//dump出core文件

(gdb) detach //detach是用来断开与jvm的连接的

(gdb) quit //退出

就用上面的命令即可

jmap -dump:format=b,file=heap.hprof /opt/zhss/java/bin /tmp/dump.core

接着用上面这行命令，将dump.core文件转换成hprof文件即可

接着就是可以用MAT工具，或者是Eclipse MAT的内存分析插件，来对hprof文件进行分析，看看到底是哪个王八蛋对象太多了，导致内存溢出了

用MAT分析一下hprof文件，马上就是出来一个图

byte[] 489005612 489005612 489005612

。。。。。下面一大堆

这个其实也是给你显示每个对象占用的内存大小，大概就是这个

其实到此为止就差不多了，我们没时间带着大家来实战，以后架构班都会带着大家深入学习和实战，如果面试，你能把上面的步骤说出来就不错了，很标准的步骤，一些工具的使用都ok了，你就说看到具体哪个对象之后，就去看代码，解决问题就行了。

一般常见的OOM，要么是短时间内涌入大量的对象，导致你的系统根本支持不住，此时你可以考虑优化代码，或者是加机器；要么是长时间来看，你的很多对象不用了但是还被引用，就是内存泄露了，你也是优化代码就好了；这就会导致大量的对象不断进入老年代，然后频繁full gc之后始终没法回收，就撑爆了

要么是加载的类过多，导致class在永久代理保存的过多，始终无法释放，就会撑爆

我这里可以给大家最后提一点，人家肯定会问你有没有处理过线上的问题，你就说有，最简单的，你说有个小伙子用了本地缓存，就放map里，结果没控制map大小，可以无限扩容，最终导致内存爆了，后来解决方案就是用了一个ehcache框架，自动LRU清理掉旧数据，控制内存占用就好了。

另外，务必提到，线上jvm必须配置-XX:+HeapDumpOnOutOfMemoryError，-XX:HeapDumpPath=/path/heap/dump。因为这样就是说OOM的时候自动导出一份内存快照，你就可以分析发生OOM时的内存快照了，到底是哪里出现的问题。

**3.2 系统频繁full gc**

比OOM稍微好点的是频繁full gc，如果OOM就是系统自动就挂了，很惨，你绝对是超级大case，但是频繁full gc会好多，其实就是表现为经常请求系统的时候，很卡，一个请求卡半天没响应，就是会觉得系统性能很差。

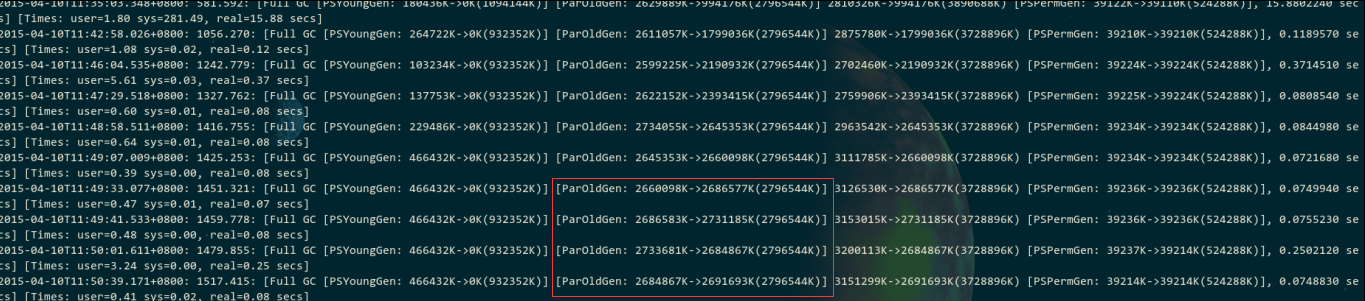
首先，你必须先加上一些jvm的参数，让线上系统定期打出来gc的日志：

-XX:+PrintGCTimeStamps

-XX:+PrintGCDeatils

-Xloggc:<filename>

这样如果发现线上系统经常卡顿，可以立即去查看gc日志，大概长成这样：



如果要是发现每次Full GC过后，ParOldGen就是老年代老是下不去，那就是大量的内存一直占据着老年代，啥事儿不干，回收不掉，所以频繁的full gc，每次full gc肯定会导致一定的stop the world卡顿，这是不可能完全避免的

接着采用跟之前一样的方法，就是dump出来一份内存快照，然后用Eclipse MAT插件分析一下好了，看看哪个对象量太大了

接着其实就是跟具体的业务场景相关了，要看具体是怎么回事，常见的其实要么是内存泄露，要么就是类加载过多导致永久代快满了，此时一般就是针对代码逻辑来优化一下。

给大家还是举个例子吧，我们线上系统的一个真实例子，大家可以用这个例子在面试里来说，比如说当时我们有个系统，在后台运行，每次都会一下子从mysql里加载几十万行数据进来各种处理，类似于定时批量处理，这个时候，如果对几十万数据的处理比较慢，就会导致比如几分钟里面，大量数据囤积在老年代，然后没法回收，就会频繁full gc。

当时我们其实就是根据这个发现了当时两台机器已经不够了，因为我们当时线上用了两台4核8G的虚拟机在跑，明显不够了，就要加机器了，所以增加了机器，每台机器处理更少的数据量，那不就ok了，马上就缓解了频繁full gc的问题了。

面试就先到这里，以后我们架构师课程会走实战派，大量线上系统jvm问题在机器上给大家模拟出来，然后带着大家实战一步一步演练和处理。

853769620