**一个面试官对JVM面试问题的分析**

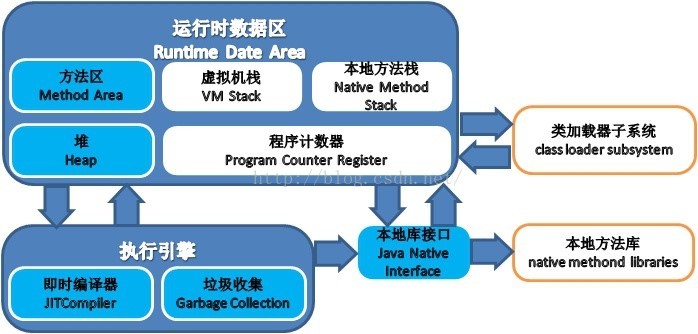
这个帖子的背景是今晚看到je上这张贴：http://www.iteye.com/topic/715256，心血来潮写下的文字，如果能抛砖引玉，能有其他面试官分析一下自己面试时问的问题，那或许是件很有意义的事情。   
  
    在公司当技术面试官几年间，从应届生到工作十几年的应聘者都遇到过。先表达一下我自己对面试的观点：   
  
    1.笔试、面试去评价一个人肯定是不够准确的，了解一个人最准确的方式就是“路遥知马力，日久见人心”。通过一、二个小时内的做题、交流，只是没有其他办法下进行的无奈之举，所以通过了面试不代表有多成功，没通过也不代表有多失败。   
    2.好的面试官本身交谈的时候就不应当把自己一个居高临下的角色上，应当把自己和应聘者当做两个做技术的人平等的交流，把自己当作权威往往就会受到观点的角度、语言表达、工作领域的惯性的制约。   
    3.好的考察题目则是大家能经常接触，不同层次的人能有不同层次的答案，能从问题引申出后面继续讨论的话题。   
  
    举个例子抛砖引玉，下面这个问题是我以前常问的，从应届生到工作十几年的人都问过：

**引用**

“地球人都知道，Java有个东西叫垃圾收集器，它让创建的对象不需要像c/cpp那样delete、free掉，你能不能谈谈，GC是在什么时候，对什么东西，做了什么事情？”

我自己分析一下这个问题，首先是“什么时候”，不同层次的回答从低到高排列：   
  
    1.系统空闲的时候。   
    分析：这种回答大约占30%，遇到的话一般我就会准备转向别的话题，譬如算法、譬如SSH看看能否发掘一些他擅长的其他方面。   
  
    2.系统自身决定，不可预测的时间/调用System.gc()的时候。   
    分析：这种回答大约占55%，大部分应届生都能回答到这个答案，起码不能算错误是吧，后续应当细分一下到底是语言表述导致答案太笼统，还是本身就只有这样一个模糊的认识。   
  
    3.能说出新生代、老年代结构，能提出minor gc/full gc   
    分析：到了这个层次，基本上能说对GC运作有概念上的了解，譬如看过《深入JVM虚拟机》之类的。这部分不足10%。   
  
    4.能说明minor gc/full gc的触发条件、OOM的触发条件，降低GC的调优的策略。   
    分析：列举一些我期望的回答：eden满了minor gc，升到老年代的对象大于老年代剩余空间full gc，或者小于时被HandlePromotionFailure参数强制full gc；gc与非gc时间耗时超过了GCTimeRatio的限制引发OOM，调优诸如通过NewRatio控制新生代老年代比例，通过MaxTenuringThreshold控制进入老年前生存次数等……能回答道这个阶段就会给我带来比较高的期望了，当然面试的时候正常人都不会记得每个参数的拼写，我自己写这段话的时候也是翻过手册的。回答道这部分的小于2%。   
  
PS：加起来不到100%，是因为有确实少数直接说不知道，或者直接拒绝回答的= =#   
  
    分析第二个问题：“对什么东西”：   
  
    1.不使用的对象。   
    分析：相当于没有回答，问题就是在问什么对象才是“不使用的对象”。大约占30%。   
  
    2.超出作用域的对象/引用计数为空的对象。   
    分析：这2个回答站了60%，相当高的比例，估计学校教java的时候老师就是这样教的。第一个回答没有解决我的疑问，gc到底怎么判断哪些对象在不在作用域的？至于引用计数来判断对象是否可收集的，我可以会补充一个下面这个例子让面试者分析一下obj1、obj2是否会被GC掉？   
    class C{   
         public Object x;   
    }   
    C obj1、obj2 = new C();   
    obj1.x = obj2;   
    obj2.x = obj1;   
    obj1、obj2 = null;   
  
    3.从gc root开始搜索，搜索不到的对象。   
    分析：根对象查找、标记已经算是不错了，小于5%的人可以回答道这步，估计是引用计数的方式太“深入民心”了。基本可以得到这个问题全部分数。   
    PS：有面试者在这个问补充强引用、弱引用、软引用、幻影引用区别等，不是我想问的答案，但可以加分。   
  
    4.从root搜索不到，而且经过第一次标记、清理后，仍然没有复活的对象。   
    分析：我期待的答案。但是的确很少面试者会回答到这一点，所以在我心中回答道第3点我就给全部分数。   
  
    最后由一个问题：“做什么事情”，这个问发挥的空间就太大了，不同年代、不同收集器的动作非常多。   
  
    1.删除不使用的对象，腾出内存空间。   
    分析：同问题2第一点。40%。   
  
    2.补充一些诸如停止其他线程执行、运行finalize等的说明。   
    分析：起码把问题具体化了一些，如果像答案1那样我很难在回答中找到话题继续展开，大约占40%的人。   
    补充一点题外话，面试时我最怕遇到的回答就是“这个问题我说不上来，但是遇到的时候我上网搜一下能做出来”。做程序开发确实不是去锻炼茴香豆的“茴”有几种写法，不死记硬背我同意，我不会纠语法、单词，但是多少你说个思路呀，要直接回答一个上网搜，我完全没办法从中获取可以评价应聘者的信息，也很难从回答中继续发掘话题展开讨论。建议大家尽量回答引向自己熟悉的，可讨论的领域，展现给面试官最擅长的一面。   
  
    3.能说出诸如新生代做的是复制清理、from survivor、to survivor是干啥用的、老年代做的是标记清理、标记清理后碎片要不要整理、复制清理和标记清理有有什么优劣势等。   
    分析：也是看过《深入JVM虚拟机》的基本都能回答道这个程度，其实到这个程度我已经比较期待了。同样小于10%。   
  
    4.除了3外，还能讲清楚串行、并行（整理/不整理碎片）、CMS等搜集器可作用的年代、特点、优劣势，并且能说明控制/调整收集器选择的方式。   
    分析：同上面2个问题的第四点。 

    最后介绍一下自己的背景，在一间不大不小的上市软件公司担任平台架构师，有3年左右的面试官经验，工作主要方向是大规模企业级应用，参与过若干个亿元级的项目的底层架构工作。

1. Sun HotSpot VM，是JDK和Open JDK中自带的虚拟机，也是目前使用范围最广的Java虚拟机。  
   JVM内存分布  
     
   程序计数器：是一块较小的内存空间，可以看作是当前线程所执行的字节码的行号指示器。程序中的分支、循环、跳转、异常处理、线程恢复等基础功能都需要依赖这个计数器完成。由于多线程是通过线程轮流切换并分配处理器执行时间的方式来实现的，故该区域为线程私有的内存。  
   虚拟机栈：描述的是Java方法执行的内存模型，用于存储局部变量表、操作数栈、动态链接、方法出口等  
   堆：是Java虚拟机所管理的内存中最大的一块，Java堆是被所有线程共享的一块内存区域，在虚拟机启动时创建，存放所实例，也是垃圾收集器管理的主要  
   方法区：用于存放已被虚拟机加载的类信息、常量、静态变量、即时编译后的代码等数据。HotSVM针对该区域也进行GC，主要是常量回收以及类
2. JVM内存分配策略  
   对象的内存分配，在大方向上，是在Java堆上进行分配。  
   大多数情况下，对象在新生代Eden区中分配，当Eden区没有足够空间进行分配时，虚拟机将发起一次Minor GC。  
   大多数情况下，大对象直接进入老年代，虚拟机提供了参数来定义大对象的阀值，超过阀值的对象都会直接进入老年代。  
   经过多次Minor GC后仍然存活的对象（长期存活的对象），将进入老年代。虚拟机提供了参数，可以设置阀值。
3. JVM垃圾回收算法  
   标记-清除算法：首先标记出所有需要回收的对象，在标记完成后统一回收所有被标记的对象。  
   复制算法：将可用内存按容量划分为大小相等的两块，每次只使用其中的一块。当一块内存用完了，将还存另外一块上面，然后在把已使用过的内存空间一次清理掉。  
   标记-整理算法：标记过程与“标记-清除”算法一样，但后续步骤不是直接对可回收对象进行清理，而是让所一端移动，然后直接清理掉端边界以外的内存。  
   分代收集算法：一般是把Java堆分为新生代和老年代，根据各个年代的特点采用最适当的收集算法。新生代都发现有大批对象死去，选用复制算法。老年代中因为对象存活率高，必须使用“标记-清理”或“标记-整理”算法来进行回收。
4. 垃圾收集器  
   Serial收集器：是一个单线程的收集器，只会使用一个CPU或一条收集线程去完成垃圾收集工作，在进行垃圾收集时，必须暂停其他所有的工作线程，直到它收集结束。  
   ParNew收集器：是Serial收集器的多线程版本，除了使用多条线程进行垃圾收集之外，其余行为与Serial收集器完全一样。  
   CMS收集器：是一种以获取最短回收停顿时间为目标的收集器。过程分为以下四个步骤：  
       初始标记  
       并发标记  
       重新标记  
       并发清除
5. JVM常见启动参数  
   -Xms / -Xmx — 堆的初始大小 / 堆的最大大小  
   -Xmn — 堆中年轻代的大小  
   -XX:-DisableExplicitGC — 让System.gc()不产生任何作用  
   -XX:+PrintGCDetails — 打印GC的细节  
   -XX:+PrintGCDateStamps — 打印GC操作的时间戳  
   -XX:NewSize / XX:MaxNewSize — 设置新生代大小/新生代最大大小  
   -XX:NewRatio — 可以设置老生代和新生代的比例  
   -XX:PrintTenuringDistribution — 设置每次新生代GC后输出幸存者乐园中对象年龄的分布  
   -XX:InitialTenuringThreshold / -XX:MaxTenuringThreshold：设置老年代阀值的初始值和最大值  
   -XX:TargetSurvivorRatio：设置幸存区的目标使用率
6. JAVA类生命周期  
   Java类从被加载到虚拟机内存中开始，到卸载出内存为止，它的整个生命周期包括：加载、验证、准备、解析、初始化、使用、卸载七个阶段。
7. JVM类加载  
   启动（Bootstrap）类加载器：是用本地代码实现的类装入器，它负责将 <Java\_Runtime\_Home>/lib下面的类库加载到内存中（比如rt.jar）。由于引导类加载器涉及到虚拟机本地实现细节，开发者无法直接获取到启动类加载器的引用，所以不允许直接通过引用进行操作。  
   标准扩展（Extension）类加载器：是由 Sun 的 ExtClassLoader（sun.misc.Launcher$ExtClassLoader）实现Java\_Runtime\_Home >/lib/extjava.ext.dir指定位置中的类库加载到内存中。开发者可以直接使用标准扩展类加载器。  
   系统（System）类加载器：是由 Sun 的 AppClassLoader（sun.misc.Launcher$AppClassLoader）实现的。径（CLASSPATH）中指定的类库加载到内存中。开发者可以直接使用系统类加  
   双亲委派机制描述 ：某个特定的类加载器在接到加载类的请求时，首先将加载任务委托给父类加载器，依次递归，如果父类加载器可以完成类加载任务，就成功返回；只有父类加载器无法完成此加载任务时，才自己去加载。
8. JVM调优  
   查看堆空间大小分配（年轻代、年老代、持久代分配）  
   垃圾回收监控（长时间监控回收情况）  
   线程信息监控：系统线程数量  
   线程状态监控：各个线程都处在什么样的状态下  
   线程详细信息：查看线程内部运行情况，死锁检查  
   CPU热点：检查系统哪些方法占用了大量CPU时间  
   内存热点：检查哪些对象在系统中数量最大