JVM虚拟机

1 Jvm数据区域

- 程序计数器：当前线程所执行的字节码的行号指示器，如果线程执行的是一个java方法，这个计数器记录的是正在执行的虚拟机字节码指令的地址；
- Java虚拟机栈：描述的是java方法执行的内存模型，每个方法在执行的时候都会创建一个栈帧用于存放局部变量表、操作数栈、动态链接、方法出口等；
- 本地方法栈：和虚拟机栈类似，不过是为本地Native方法服务；
- Java堆：存放对象实例，是垃圾回收的主要区域；
- 方法区：用于存储已经被虚拟机加载的类信息、常量、静态常量等数据

方法区和堆是所有线程共享的内存区域。 程序计数器、虚拟机栈和本地方法栈是线程私有，隔离的内存区域。

2 垃圾回收方法

线程私有的程序计数器、虚拟机栈和本地方法栈三个区域随着线程生，随着线程而灭，因此不需要多考虑回收问题。 而java对和方法区内存的分配是动态的，垃圾回收主要关注的是这部分。

2.1 判断对象已经可以被回收

2.1.1 引用计数法

给对象中添加一引用计数器，每当一个地方引用它时，计数器+1，当引用失效时计数器-1，任何计数器为0的对象不可能再被使用；但存在的问题是很难解决对象互相循环引用的问题。

2.1.2 可达性分析

基本思想是通过一系列被称为“GC Roots”的对象作为起始点，从这些节点向下搜索，搜索走过的路径称为引用链，当一个对象到GC Roots没有任何引用链相连时，证明此对象不可达

2.1.3 回收方法区

方法区（永久代）一般不要求回收垃圾，因为方法与的垃圾回收效率低。 永久代的垃圾回收主要包括：废弃常量和无用的类（类信息回收）。 回收废弃常量和回收Java堆中的类似，当没有地方引用这个常量是回发生回收； 而回收类要满足的条件要相对复杂： 1、该类的所有实例对象已经被回收； 2、该类的ClassLoader已经被回收； 3、该类对应的java.lang.class没有地方引用，即不能通过反射来访问类。

2.2 垃圾回收算法

新生代占堆的1/3，老年代占堆的2/3；其中Eden占新生代的8/10，from占1/10，to占1/10；

2.2.1 标记-清除算法

算法分为两步：首先标记处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； 主要不足所有两点：1效率不高；2标记清除后会产生大量不连续的内存碎片。

2.2.2 复制算法

复制算法用于新生代，因为适用于对象的存活率较低的区域； 将新生代内存划分为Eden和两块较小的Survivor区域，每次只使用Eden和一块Survivor。当回收时将这两块区域中活着的对象一次性复制到另一块Survivor中。如果Survivor不够用时，需要老年代进行分配担保。

2.2.3 标记-整理算法

同样先需要进行标记回收对象，但后续不是直接对可回收对象进行处理，而是让存活的对象都向一端移动，然后清除掉边界外的内存。

2.2.4 分代回收

新生代：每次回收都有大量对象死去，所以使用复制算法； 老年代：对象存活率较高，没有额外空间进行分配担保，必须使用标记-清除和标记-整理算法。

3 对象内存分配

对象主要分配在新生代的Eden区上，如果启动了本地线程分配缓冲，会按线程优先在TLAB（Thread Local Allocation Buffer）分配。少数情况下会直接分配在老年代。

3.1 对象优先分配在Eden

大部分情况下，对象新生代Eden去中分配。当Eden区中没有足够的空间进行分配时，虚拟机将触发一次Minor GC；

3.2 大对象将直接进入老年代

大对象指需要大量连续内存空间的java对象，典型的大对象就是那种很长的字符串以及数组；

3.3 长期存活的对象将进入老年代

虚拟机给每个对象定义了一个对象年龄计数器，如果对象在Eden出生并经过一次Minor GC后被移动到Survivor中，设置年龄为1.然后每在Survivor中经过一次Minor GC存活年龄就+1，当它的年龄增长的一定程度（默认为15）将被晋升为老年代。

4 类加载机制

4.1 类的声明周期

加载、验证、准备、解析、初始化、使用和卸载

4.2 双亲委派模型

双亲委派模型要求除了顶层的启动类加载器外，其余的类加载器都应当有自己的父类加载器。 双亲委派模型的工作过程是：如果一个类加载器收到类加载的请求，它首先不会自己去尝试加载这个类，而是把这个请求委派给父类加载器去完成，每个层析的类加载器都如此，这样所有的加载请求都会传送到顶层的启动类加载器，只有父加载器反馈自己无法完成加载时，子加载器才会尝试自己加载。

5 Java内存模型（JMM）

JMM本身是一种抽象的概念，并不真实存在，它描述的是一组规则或规范，通过这组规范定义了程序中各个变量的访问方式。由于JVM运行程序的实体是线程，而每个线程创建时JVM都会为其创建一个工作内存(有些地方称为栈空间)，用于存储线程私有的数据。而Java内存模型中规定所有变量都存储在主内存，主内存是共享内存区域，所有线程都可以访问，但线程对变量的操作(读取赋值等)必须在工作内存中进行，首先要将变量从主内存拷贝的自己的工作内存空间，然后对变量进行操作，操作完成后再将变量写回主内存，不能直接操作主内存中的变量，线程间的通信(传值)必须通过主内存来完成。