Java程序中内存的分配和回收是java虚拟机控制的，一起了解下java中的垃圾回收机制GC吧。在jvm内存中，程序计数器、java栈和本地方法栈的内存随着栈帧的出栈会被销毁掉，所以GC主要针对的是java堆和方法区。

1、怎么判定对象的存活

2、垃圾的收集算法有哪些

3、GC有的分类和触发条件

一、对象存活的判定

判断对象是否存活的算法有两种：计数器算法和可达性算法，其中在java虚拟机使用的是可达性算法，让我们分别看下这两种算法吧

1、计数器算法

每个存活的对象都有一个计数器，有来引用它的时候计数器加1，引用取消以后计数器减1，当计数器为0的时候，就没有地方引用这个对象了。这时这个对象可以被列入回收的队伍中了。

计数器算法很简单效率也很高，那为什么java虚拟机不使用它呢？计数器算法在出现相互引用的情况下，会出现计数器无法归零的情况。两个对象obj1和obj2分别是对方的一个属性，当obj1=null; obj2=null;时引用应该为0，由于相互引用，计数器不能为0会一直判定为存活。

2、可达性算法

可达性算法中通过GC Roots对象作为起点，向下通过引用链来检索，没有被引用链相连的就是不可达的对象。这些不可达的对象就可以作为GC回收的对象

注意：GC Roots对象不是某个特定的对象，是一系列的对象，哪些对象可以作为GC Roots呢

1、栈帧中局部变量表中的引用对象

2、方法区中静态的引用对象

3、方法区中产量的引用对象

4、Native方法中的引用对象

要理解可达性算法，可以参考下图



Obj4和Obj5就是不可达的对象，可以被回收掉。

二、垃圾收集算法

堆内存分为年轻代和年老代两部分，年轻代使用的是复制算法，老年代用的标记整理算法

1、复制算法

复制算法，当一块内存用完了，标记还存活的对象并复制到另一块内存上，然后把这块内存清空。

复制算法不需要内存碎片的整理，既简单又高效，缺点是内存的利用率只有原来的一半。

在年轻代中使用的是复制算法，年轻代用来存放新创建的对象，通常认为年轻代中对象的死亡率很高，在年轻代的三个区域使用复制算法通常回收率能达到90%。

2、标记整理算法

这个算法分为两个过程

标记清除，使用可达算法标记存活的对象，清理没有被标记的对象

内存整理，清理后的内存存在大量的内存碎片，为了接下来内存的使用，需要将内存整理成连续的内存空间。

这种算法保证了内存的连续性，并充分利用了所有内存，缺点是效率相对其他算法偏低。

年老代使用的标记整理算法，老年代存储的是存活了很久的对象，对象被回收的概率比较低。

三、GC详述

GC有三种： Minor GC、 Major GC和Full GC

Minor GC :清理年轻代内存，在内存空间满了以后，就会触发

Major GC：清理方法区（永久代）内存（需要Minor GC来触发）

Full GC：清理所有堆内存（包括年轻代和年老代）

Full GC的触发条件

1、System.gc()会建议系统触发，不是绝对的

2、年老代存储空间不足

3、空间分配担保失败（每次Minor GC之前会检查老年代的最大连续内存空间是否超过年轻代存活对象的总空间，如果小于会查询HandlePromotionFailure设置的值是否允许冒险，如果不允许，就会触发一次Full GC）

JVM采用的垃圾收集算法叫分代收集算法，分代收集算法和上文中的收集算法有什么关系呢？

JVM把堆内存分为年轻代和年老代，年轻代垃圾收集算法使用复制算法，年老代使用标记整理算法。存活对象在达到一定条件会从年轻代移到年老代中。这就是分代收集算法。

对象什么情况下会从年轻代移动到老年代呢？

Jvm给每一个对象都定义了一个年龄计数器，每次Minor GC以后，存活的对象年龄加1，经过多次Minor GC达到一定年龄（默认为15 可通过参数-XX:MaxTenuringThreshold设置）的存活对象会被放到老年代。

还有以下情况在对象没有达到预定年龄时，也被放到老年代

1、存活对象太多，Eden存储空间不足

2、大对象会直接进入老年代（-XX：PretenureSizeThreshold设置大小）

3、相同年龄的对象超过survivor空间的一半，大于等于这个年龄的会进入老年代

GC不是在随便位置都能进行的，GC时中需要中断所有线程（Stop The World）。为什么要这么做呢，如果在GC的时引用关系还在变化（例如需要被回收的引用，又重新被引用），回收结果会有问题。所以需要中断所有线程。但中断所有线程，需要记录所有线程的上下文切换状态这需要很大代价，所以提出了安全点（SafePoint）的概念，在程序中设置了一些特殊位置作为安全点，当线程运行到最近的安全点会根据需要判断是否停下来。

那线程到达安全点怎么判断是否要停下来呢？

有两种方法：

抢先试中断，GC发生时线中断所有线程，对于不在安全点的线程，让它恢复运行，运行到最近的安全点上。

主动式中断，GC发生时，设置一个标志，线程执行时会轮询这个标志，发现需要中断时就把停止运行。轮询的位置和安全点重合。

HotSpot采用的主动式中断。安全点的选择是以能否让程序长时间执行为标准来选择的。

线程在运行状态，GC时进入安全点停顿，如果线程已经进入阻塞或者等待状态呢，不可能等到线程重新获取到了时间片再GC吧，所以这里有另外一个概念，安全区域（Safe Region）。

安全区域，是安全点的扩展，当代码运行到这个区域引用关系不会发生变化，在这个区域中进行GC是安全的。当线程运行到安全区域时会标识自己的线程状态为Safe Region，当GC发生时不会管这些标识过的线程。当线程要离开安全区域时，会检查是否GC结束，如果没有结束就会一直等待收到可以安全离开安全区域的信号，才会继续执行。

到这里GC的知识点已经基本结束，有问题可以留言一起讨论。