**垃圾回收**

JVM中会在以下情况触发回收：对象没有被引用，作用域发生未捕捉异常，程序正常执行完毕，程序执行了System.exit()，程序发生意外终止。

JVM中标记垃圾使用的算法是一种根搜索算法。简单的说，就是从一个叫GC Roots的对象开始，向下搜索，如果一个对象不能达到GC Roots对象的时候，说明它可以被回收了。这种算法比一种叫做引用计数法的垃圾标记算法要好，因为它避免了当两个对象啊互相引用时无法被回收的现象。

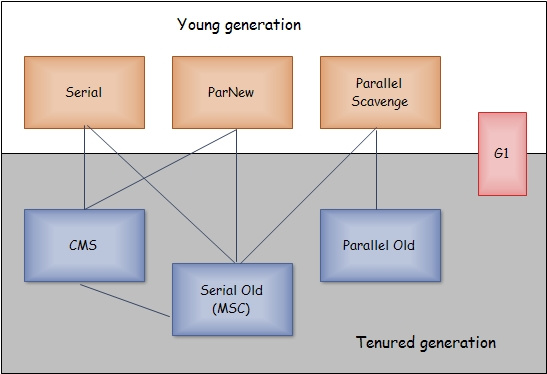
JVM中对于被标记为垃圾的对象进行回收时又分为了一下3种算法：

1.标记清除算法，该算法是从根集合扫描整个空间，标记存活的对象，然后在扫描整个空间对没有被标记的对象进行回收，这种算法在存活对象较多时比较高效，但会产生内存碎片。

2.复制算法，该算法是从根集合扫描，并将存活的对象复制到新的空间，这种算法在存活对象少时比较高效。

3.标记整理算法，标记整理算法和标记清除算法一样都会扫描并标记存活对象，在回收未标记对象的同时会整理被标记的对象，解决了内存碎片的问题。

JVM中，不同的 内存区域作用和性质不一样，使用的垃圾回收算法也不一样，所以JVM中又定义了几种不同的垃圾回收器（图中连线代表两个回收器可以同时使用）：



1.Serial GC。从名字上看，串行GC意味着是一种单线程的，所以它要求收集的时候所有的线程暂停。这对于高性能的应用是不合理的，所以串行GC一般用于Client模式的JVM中。

2.ParNew GC。是在SerialGC的基础上，增加了多线程机制。但是如果机器是单CPU的，这种收集器是比SerialGC效率低的。

3.Parrallel Scavenge GC。这种收集器又叫吞吐量优先收集器，而吞吐量=程序运行时间/(JVM执行回收的时间+程序运行时间),假设程序运行了100分钟，JVM的垃圾回收占用1分钟，那么吞吐量就是99%。Parallel Scavenge GC由于可以提供比较不错的吞吐量，所以被作为了server模式JVM的默认配置。

4.ParallelOld是老生代并行收集器的一种，使用了标记整理算法，是JDK1.6中引进的，在之前老生代只能使用串行回收收集器。

5.Serial Old是老生代client模式下的默认收集器，单线程执行，同时也作为CMS收集器失败后的备用收集器。

6.CMS又称响应时间优先回收器，使用标记清除算法。他的回收线程数为(CPU核心数+3)/4，所以当CPU核心数为2时比较高效些。CMS分为4个过程：初始标记、并发标记、重新标记、并发清除。

7.GarbageFirst（G1）。比较特殊的是G1回收器既可以回收Young Generation，也可以回收Tenured Generation。它是在JDK6的某个版本中才引入的，性能比较高，同时注意了吞吐量和响应时间。

对于垃圾收集器的组合使用可以通过下表中的参数指定：



默认的GC种类可以通过jvm.cfg或者通过jmap dump出heap来查看，一般我们通过jstat -gcutil [pid] 1000可以查看每秒gc的大体情况，或者可以在启动参数中加入：-verbose:gc -XX:+PrintGCTimeStamps -XX:+PrintGCDetails -Xloggc:./gc.log来记录GC日志。

GC中有一种情况叫做Full GC，以下几种情况会触发Full GC：

1.Tenured Space空间不足以创建打的对象或者数组，会执行FullGC，并且当FullGC之后空间如果还不够，那么会OOM:java heap space。

2.Permanet Generation的大小不足，存放了太多的类信息，在非CMS情况下回触发FullGC。如果之后空间还不够，会OOM:PermGen space。

3.CMS GC时出现promotion failed和concurrent mode failure时，也会触发FullGC。promotion failed是在进行Minor GC时，survivor space放不下、对象只能放入旧生代，而此时旧生代也放不下造成的；concurrent mode failure是在执行CMS GC的过程中同时有对象要放入旧生代，而此时旧生代空间不足造成的。

4.判断MinorGC后，要晋升到TenuredSpace的对象大小大于TenuredSpace的大小，也会触发FullGC。

可以看出，当FullGC频繁发生时，一定是内存出问题了。