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。
CMS是基于“标记--清理”算法实现的，整个过程分为四个步骤：1.初始标记，2.并发标记，3.重新标记，4.并发清理。

初始标记：仅仅是标记一下GC root能直接关联的对象。速度很快。
并发标记：就是进行GC roots tracing的过程。
重新标记：重新标记阶段就是为了修正并发标记期间因为用户程序继续运行而导致标记产生变动的那一部分对象的标记记录，这个阶段的停顿时间一般会比初始标记阶段的时间稍长，远远比并发标记阶段时间短。

优点：并发收集，低停顿。
理由：由于整个过程中最耗时的并发标记和并发清除过程收集器程序都可以和用户线程一起工作，所以总体来说，CMS收集器内存回收过程是与用户线程一起并发执行的。

缺点：
1.CMS收集器对CPU资源非常敏感。
在并发阶段，虽然不会导致用户线程停顿，但是会因为占用了一部分线程使应用程序变慢，总吞吐量会降低，为了解决这种情况，虚拟机提供了一种“增量式并发收集器”。就是在并发标记和并发清除的时候让GC线程和用户线程交替运行，尽量减少GC线程独占资源的时间，这样整个垃圾收集的过程会变成，但是对用户程序的影响会减少。

2.CMS在并发清理阶段线程还在进行，伴随着程序的运行自然也会产生新的垃圾，这一部分垃圾产生在标记过程之后，CMS无法在当次过程中处理，所以只有等到下次GC时候再清理掉，这一部分垃圾就叫做“浮动垃圾”。

3.CMS是基于“标记-清除”算法实现的，所以在收集结束的时候会有大量的空间碎片产生。空间碎片太多的时候，将会给大对象的分配带来很大的麻烦，往往会出现老年代还有很大的空间剩余，但是无法找到足够大的连续空间来分配当前对象，只能提前出发full gc。

G1是面向服务端应用的垃圾收集器。具有如下特点。
1.并行于并发：G1能充分利用CPU、多核环境下的硬件优势，使用多个cpu来缩短stop-the-world停顿时间。部分其他收集器原本需要停顿java线程执行的GC动作，G1收集器仍然可以通过并发的方式让java程序继续执行。

2.分代收集：虽然G1可以不需要其他收集器配合就能独立管理整个GC堆，但是还是保留了分代的概念。它能够采用不同的方式去处理新创建的对象和已经存活了一段时间的对象，熬过了多次GC的旧对象以获取更好的收集效果。

3.空间整合：与CMS的“标记-清除”算法不同，G1从整体来看是基于“标记-整理”算法实现的收集器，从局部上来看是基于“复制”算法实现的。

4.可预测的停顿：这是G1相对于CMS的另一个大优势，降低停顿时间是G1和CMS共同的关注点，但G1除了追求低停顿外，还能建立可预测的停顿时间模型，能让使用者明确指定在一个长度为M毫秒的时间片段内。

5.G1收集器的运作步骤：

1.初始标记 2.并发标记 3.最终标记 4.筛选回收