垃圾收集算法

- **标记-清除算法：**标记所有回收对象，标记完成后统一回收所有被标记对象（不足：效率低，产生大量内存碎片）
- **复制算法：**将可用内存容量划分为大小相等两块，每次使用一块，当一块用完，就将还存活着的对象复制到另外一块上面，清理已使用过的内存空间（代价是将内存缩小为原来的一半），多采用这种方法收集新生代，将内存划分为一块较大的Eden空间和两块较小的Survivor空间，比例（8：1:1），每次使用Eden和一块较小的Survivor，当回收时，将区域中活着的对象复制到另一块Survivor空间中，最后清理掉Eden和刚才用过的Survivor空间。当Survivor空间不够时，依赖老年代进行分配担保，这些对象通过分配担保机制进入老年代
- **标记-整理算法：**与标记-清除算法一样，但后续需要让所有存活对象都向一端移动，然后直接清理掉端边界以外的内存。
- **分代收集算法：**将java堆划分为新生代和老年代，根据各年代特点采取最适当的收集算法，新生代-复制算法，老年代-标记-清理，标记-整理算法

垃圾收集器

GC停顿：GC进行时必须停止所有java执行线程，不可以出现分析过程中对象引用关系还在不断变化的情况，该点不满足的话分析结果准确性无法得到保证
安全点：程序并非在所有地方都可以暂停开始GC,只有到达安全点时才暂停:主动式中断：GC需要中断线程时，设置一个标志，各个线程执行时主动去轮询这个标志，为真时自己中断挂起，轮询标志地点和安全点重合
安全区域：对于程序“不执行”时，即引用关系不发生变化时，在任意地方开启GC都安全.
Serial收集器：新生代，单线程（收集时暂停其他所有工作线程），它依然是虚拟机运行在Client模式下的默认新生代收集器，
优点：简单高效，对于限定单个CPU，没有线程交互的开销，适用于用户桌面应用场景

ParNew收集器：Serial收集器的多线程版本，运行在Server模式下首选的新生代收集器，除Serial收集器外，只有它能与CMS收集器配合工作

Parallel Scanvenge收集器：新生代的多线程收集器，目标是达到一个可控制的吞吐量，自适应调节策略

Serial Old收集器：Serial收集器的老年代版本，单线程收集器，基于标记整理算法。

Parallel Old收集器：Parallel Scanvenge收集器的老年代版本，使用多线程和标记整理算法。

CMS收集器：获取最短回收停顿时间为目标的标记器，基于标记-清除算法，

分为：初始标记、并发标记、重新标记、并发清除4个步骤

初始标记和重新标记仍然需要GC停顿

内存回收过程与用户线程一起并发执行的

优点：并发收集、低停顿

缺点：对CPU资源敏感，无法处理浮动垃圾（并发清理阶段用户线程还在运行，产生新的垃圾，CMS无法在当次收集中处理他们），采用标记-清除算法产生大量内存碎片，

G1收集器：

特点：

- 并行与并发
- 分代收集：不需要其他收集器配合就可以管理整个GC堆
- 空间整合：整体基于标记整理算法，不会产生内存空间碎片
- 可预测的停顿：可以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，消耗在垃圾收集上的时间不得超过N毫秒

G1将整个java堆分成多个大小相等的独立区域(Region)

G1收集器之所以能建立可预测的停顿时间模型，是因为它可以有计划的避免在整个Java堆中进行全区域的垃圾收集。G1跟踪各个Regin里面的垃圾堆积的价值大小（回收所获得的价值大小以及回收所需时间的经验值），在后台维护一个优先列表，优先回收最大的Regin（Garbage-first名称的由来）并且在G1收集

器中，Region之间的对象引用都是通过Remembered Set来避免全堆扫描的，每个Regin都有一个与之对应的Remembered Set
不计算维护Remembered Set的操作，G1收集器的运作划分以下几个步骤：初始标记、并发标记、最终标记、筛选回收