**垃圾收集器**

垃圾收集器是垃圾回收算法的具体实现。Java虚拟机规范中没有明确规定如何实现垃圾收集器，因此不同产商、不同版本的JVM提供的垃圾收集器也各不相同，在java 7/8以后，HotSpot中有7种垃圾回收器：Serial、ParNew、Parallel Scavenge、Serial Old、Parallel Old、CMS、G1。



处于不同区域的垃圾收集器的收集区域也是不同的，它们之间的连线代表可以搭配使用。

新生代收集器：Serial、ParNew、Parallel Scavenge

老年代收集器：Serial Old、Parallel Old、CMS

整堆收集器：G1

1. Serial收集器

Serial收集器是最基本的收集器，它工作在新生代，是一个单线程的收集器，主要特点是

* 仅仅使用一个线程完成垃圾收集工作；
* 在垃圾收集时必须暂停其他所有的工作线程，直到垃圾收集结束；
* Stop the World是在用户不可见的情况下执行的，会造成某些应用响应变慢；
* 使用复制算法；

Serial/Serial Old组合收集的工作过程如下：



特点：

* Serial收集器是HotSpot运行在Client模式下的默认新生代收集器。它的优点是简单而高效（与其它收集器的单线程相比）；
* 对于单个CPU的环境来说，Serial收集器没有线程交互的开销，可以获得最高的单线程收集效率；
* 在一般情况下，垃圾收集造成的停顿时间可以控制在几十毫秒甚至一百多毫秒以内。

1. ParNew收集器

ParNew收集器是Serial收集器的多线程版本，除了在垃圾收集过程中使用多个线程，其它行为包括控制参数、收集算法、Stop the World、对象分配规则和回收策略等与Serial都一样，也使用复制算法。两个收集器共用了不少代码。

ParNew/Serial Old组合收集的工作过程如下：



特点：

* 在Server模式下，ParNew收集器是一个非常重要的收集器，因为除Serial外，目前只有它能与CMS收集器配合工作，但在单个CPU环境中，不会比Serail收集器有更好的效果，因为存在线程交互开销。

设置参数

 -XX:+UseConcMarkSweepGC：指定使用CMS后，会默认使用ParNew作为新生代收集器；

-XX:+UseParNewGC：强制指定使用ParNew；

 -XX:ParallelGCThreads：指定垃圾收集的线程数量，ParNew默认开启的收集线程与CPU的数量相同；

3、Parallel Scavenge收集器

Parallel Scavenge收集器是一个新生代收集器，因为它关注的是吞吐量，又叫“吞吐量优先收集器”。

使用复制算法，又是并行的多线程收集器。

吞吐量：

是CPU用于运行用户代码的时间与CPU总消耗时间的比值，即吞吐量=运行用户代码时间/（运行用户代码时间+运行垃圾收集时间）。如果虚拟机一共运行100分钟，垃圾收集运行了1分钟，那么吞吐量就是99%。

特点：

CMS等收集器的关注点是尽可能缩短垃圾收集时用户线程的停顿时间，但是Parallel Scavenge收集器的目的是达到一个可控制的吞吐量。

应用场景：

**停顿时间越短就越适合与用户交互的程序，良好的响应速度能提升用户体验，而高吞吐量则可以高效的利用CPU时间，尽快完成程序的运算任务，主要适合在后台运算而不需要太多交互的任务。**

参数设置：

Parallel Scavenge收集器提供了两个参数用于精确控制吞吐量：

最大垃圾收集停顿时间(-XX:MaxGCPauseMillis)：大于0的毫秒数，收集器将尽可能在给定时间内完成垃圾收集。不过垃圾收集时间的缩短是以牺牲吞吐量和新生代空间为代价的，短的垃圾收集时间会导致更加频繁的垃圾收集行为，从而导致吞吐量的降低。

吞吐量大小(-XX:GCTimeRatio):大于0且小于100的整数，也就是垃圾收集时间占总时间的比率，相当于吞吐量的倒数。如果设置为19，允许的最大GC时间就是总时间的5% (1/(1+19))。默认是99，也就是允许最大1%的垃圾收集时间

GC 自适应调节开关(-XX:UseAdaptiveSizePolicy):这是一个开关参数，当这个参数打开后，无需手工指定新生代的大小(-Xmn)、Eden和Survivor的比例(-XX:SurvivorRatio)、晋升老年代对象年龄(-XX:PretenureSizeThreshold)等细节了，JVM会根据当前系统运行情况收集性能监控信息，动态调整这些参数，以提供最合适的停顿时间或最大的吞吐量，这种调节方式称为GC自适应的调节策略(GC Ergonomiscs)

4、Serial Old收集器

Serial Old是 Serial收集器的老年代版本；是一个单线程收集器，使用“标记-整理”算法。

应用场景

      主要用于Client模式；

      而在Server模式有两大用途：

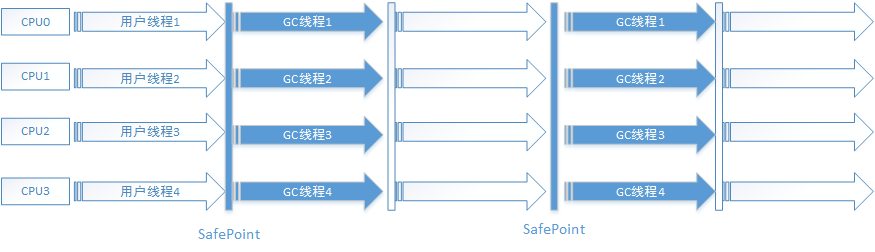
      在JDK1.5及之前，与Parallel Scavenge收集器搭配使用（JDK1.6有Parallel Old收集器可搭配）；

      作为CMS收集器的后备预案，在并发收集发生Concurrent Mode Failure时使用（后面详解）；

5、Parallel Old收集器

Parallel Old垃圾收集器是Parallel Scavenge收集器的老年代版本，使用“标记-整理”算法和多线程，JDK1.6中才开始提供。

Parallel Scavenge/Parallel Old组合收集的工作流过程如下：



应用场景

      JDK1.6及之后用来代替老年代的Serial Old收集器；

      在Server模式，多CPU的情况下，在注重吞吐量以及CPU资源敏感的场景，就有了Parallel Scavenge加Parallel Old收集器的"给力"应用组合；

设置参数

      "-XX:+UseParallelOldGC"：指定使用Parallel Old收集器；

6、CMS收集器

并发标记清理(Concurrent Mark Sweep，CMS）：是一种以获取最短回收停顿时间为目标的服务器，又被称为“并发低停顿收集器”(Concurrent Low Pause Collector)。是老年代的收集器，基于"标记-清除"算法(不进行压缩操作，产生内存碎片)。



工作过程：

初识标记(CMS initial mark)：仅标记一下GC Roots能直接关联到的对象，速度很快；但需要"Stop The World"。

并发标记(CMS concurrent mark)：进行GC Roots Tracing的过程，刚才产生的集合中标记出存活对象，应用程序也在运行；并不能保证可以标记出所有的存活对象；

重新标记(CMS remark)：修正并发标记期间因用户程序继续运作而导致标记变动的那一部分对象的标记，需要"Stop The World"，且停顿时间比初始标记稍长，但远比并发标记短，采用多线程并行执行来提升效率；

并发清除(CMS concurrent sweep)：回收所有的垃圾对象；

优点：

并发收集、低停顿。由于整个过程中耗时最长的并发标记和并发清除都可以与用户线程一起工作，所以总体上说，CMS收集器的内存回收过程与用户线程一起并发执行。

缺点：

* + CMS收集器对CPU资源非常敏感，它默认启动的回收线程数是（CPU数量+3）/4，当CPU个数大于4时，垃圾收集线程占用不少于25%的CPU资源，当CPU个数不足时，CMS对用户程序的影响很大；
  + CMS收集器无法处理浮动垃圾，可能出现“Concurrent Mode Failure”失败而导致另一次Full GC。在并发清理过程用户线程仍然有新的垃圾产生，这部分垃圾出现在标记之后，CMS无法在这次收集中回收它们，只能留在下一次。
  + CMS使用标记-清除算法，会产生内存碎片；

参数设置：

-XX:CMSInitiatingOccupancyFraction(设置老年代预留空间)：老年代使用比例达到这个百分值时触发一次FullGC。

-XX:+UseCMSCompactAtFullCollection(开启碎片整理):使得CMS出现上面这种情况时不进行Full GC，而开启内存碎片的合并整理过程，但合并整理过程无法并发，停顿时间会变长， 默认开启（但不会进行，结合下面的CMSFullGCsBeforeCompaction）。

-XX:+CMSFullGCsBeforeCompaction()：设置执行多少次不压缩的Full GC后，来一次压缩整理，为减少合并整理过程的停顿时间，默认为0，也就是说每次执行Full GC都进行压缩整理。

G1收集器

G1(Garbage First) 收集器是最先进的收集器之一，是面向服务端的垃圾收集器。

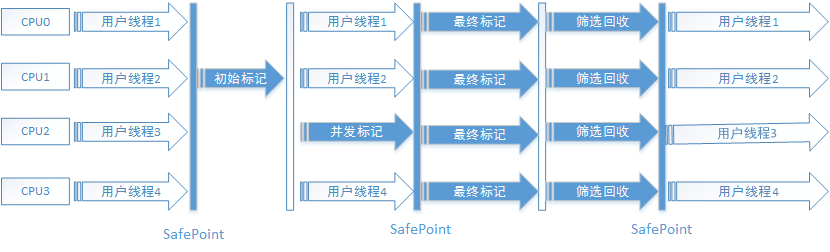
特点：

并行与并发：可以并行来缩短“STW”停顿时间，也可以并发让用户线程和垃圾回收线程执行。

分代收集：G1可以独立管理整个Java堆，可以采用不同的方式收集不同存活时间长度的对象。

空间整合：从局部看使用的是复制算法，总体上看使用的是标记-整理算法，不会产生内存碎片。

可预测的停顿：G1除了降低停顿外，还能建立可预测的停顿时间模型。可以明确指定M毫秒时间片内，垃圾收集消耗的时间不超过N毫秒



工作过程：

初始标记(Initial mark)：仅标记一下GC Roots能直接关联到的对象，需要"Stop The World"，但速度很快。

并发标记(Concurrent mark)：GC Roots Tracing的过程，耗时较长，但应用程序也在运行。

最终标记(Final mark)：为了修正并发标记期间因用户程序继续运作而导致标记变动的那一部分对象的标记，需要"Stop The World"，且停顿时间比初始标记稍长，但远比并发标记短，采用多线程并行执行来提升效率。

筛选回收(Live Data Counting and Evacuation)：首先排序各个Region的回收价值和成本，然后根据用户期望的GC停顿时间来制定回收计划，最后按计划回收一些价值高的Region中垃圾对象。回收时采用"复制"算法，从一个或多个Region复制存活对象到堆上的另一个空的Region，并且在此过程中压缩和释放内存，可以并发进行，降低停顿时间，并增加吞吐量。