jvm垃圾回收器的分类

在学习垃圾回收器之前，先了解一下并行和并发

并行：多条垃圾回收线程并行工作，但是此时用户线程出于等待状态

并发：用户线程与垃圾收集线程同事执行（不一定，可能交替执行），用户程序在继续运行，而垃圾回收程序运行在另一个cpu上

1.Serial收集器(串行垃圾收集器)

（1）串行垃圾回收器在进行垃圾回收的时候，是一个单线程的收集器，只会使用一个cpu或者一条手机线程去完成垃圾回收工作，更重要的是，在他进行垃圾回收的时候，必须暂停其他所有的工作线程，知道结束

（2）串行垃圾回收器是最古老、最稳定以及效率最高的垃圾回收器，但是会导致较长的时间停顿。新生代、老年代度采用串行回收，新生代使用复制算法，老年代使用标记-整理，垃圾回收过程会产生Stop the world

使用方法： -XX:+UseSerialGC 串联收集，在client模式来说是一个不错的选择，不设置vm参数，默认是使用串行垃圾回收器

2.ParNew收集器（串行多线程垃圾收集器）

parNew是对Serial收集器的多线程版本，新生代并行、老年代串行；新生代复制算法，老年代标记-整理算法。除了使用多线程外，其他的行为一个参数、回收算法、回收策略都与Serial收集器一样。parnew作为Server模式下虚拟机首选的新生代收集器的原因是，其中一个与性能无关的原因是，除了Serial收集器、目前只要它能与CMS收集器配合使用

使用方法：ParNew收集器是使用-XX:+UseConcMarkSweepGC选项后默认新生代的收集器，也可以使用-XX:+UseParNewGC去指定，在单个cpu或者两个Cpu的情况下，不一定性能比Serial好，因为有线程交互的开销、但是当cp的数量增加的时候，它对GC时资源的有效利用还是有好处的，默认的线程与CPU数量相同，可以使用-XX:ParalleGCThreads指定垃圾回收线程的条数

3.Parallel Scavenge收集器

Parallel Scavenge收集器是一个新生代收集器，它是使用复制算法的收集器，又是一个多线程收集器，和ParNew类似，但是Parallel Scavenge的关注点和其他的不同，CMS等垃圾回收器的关注点是尽可能的缩短垃圾回收时用户线程时停顿时间，而Parallel Scavenge的目的是达到一个可控的吞吐量，吞吐量的计算：运

行用户代码时间/(运用用户代码时间 + 垃圾回收时间)
使用方法：-XX:MaxGCPauseMillis 控制最大的垃圾收集停顿时间 -XX:GCTimeRatio直接设置吞吐量 -XX:+UseAdaptiveSizePolicy这个参数打开以后，就不需要手工指定新生代大小，eden与Survior去的比例了

4.Serial old收集器
详见上面Serial收集器，使用标记-整理算法，用于老年代

5. Parallel old收集器
是Parallel Scavenge收集器的老年代版本，使用标记-整理算法，和Parallel scavenge收集器一起使用，名副其实的应用组合，在注重吞吐量以及CPU资源敏感的场合，都有限使用Parallel Scavenge加Parallel old收集器

6. CMS收集器(Concurrent Mark Swap)
是一种以获取最短回收停顿时间为目标的收集器。目前很大一部分java应用集中在B/S服务上，这类应用尤其重视服务器的响应速度，希望系统停顿时间最短，给用户较好的体验，CMS就比较符合应用的需求
（1）初始标记 (stop the world)：标记一下GC Roots能直接关联的对象，速度很快
（2）并发标记：GC roots Tracing过程
（3）重新标记 (stop the world)：修正并发标记期因用户继续运作而导致标记产生该表的那一部分对象的标记记录
（4）并发清除
整个过程中，耗时最长的是并发表及和并发清除过程都可以和用户线程一起进行
（老年代收集器，与ParNew一起使用）
优点：并发收集、低停顿
缺点：
（1）cms对CPU资源比较敏感，线程数是：（cpu数量 + 3） / 4。因此当cpu数量不多的时候，会占用较大的cpu资源
（2）cms收集器无法处理浮动垃圾，可能出现Concurrent Model Failure失败而导致另一次Full gc的发生。因为并发清理阶段，用户线程还在运行，伴随着用户线程的运行，还会有新的垃圾产生，这一部分垃圾出现在标记过程后，CMS无法清除，因此需要下一次GC清理，这部分垃圾就是浮动垃圾。因为垃圾回收阶段，不能像其他的垃圾回收一样，等到沾满后在进行回收，必须要预留出内存空

间供用户使用，jdk1.5设置当老年代使用了68%后，就会被激活，这是一个保守设置，如果老年代增长不是很快，可以适当调增-XX:CMSInitiatingOccupancyFraction来提高触发百分比。jdk1.6后，启动阈值已经提升到了92%。要是CMS预留的空间不足的时候，就会出现ConcurrentModeFailure失败，这个时候将会临时启用Serial old进行回收老年代，这样的话，停顿时间就会很长了， 而且性能反而更低了

（3）cms是一款基于“标记-清除”算法实现的，因此会有空间碎片的问题。如果分配大对象无法找到足够大的连续空间来分配当前对象，就不得不触发Full gc。为了解决这个问题，CMS收集器提供了-XX:UseCMSCompactAtFullCollection开关（默认是开启的），用于顶不住要进行FullGc是开启内存碎片的合并整理过程，内存整理过程是无法并发的，空间碎片的问题是没有了，但是停顿时间不得不变长了，因此还提供另一个参数-XX:CMSFullGCsBeforeCompaction：这个参数用于设置执行多少此不压缩的fullgc后，跟着来一次代压缩的（默认值是0，表示每次进入Full gc都进行碎片整理）

使用方法：-XX:+UseConcMarkSweepGC 使用CMS回收器

-XX:+UseCMSCompactAtFullCollection full gc后，进行一次碎片整理，过程是独占的。会引起停顿时间变长

-XX:+CMSFullGCsBeforeCompaction: 设置进行几次Fullgc后，进行一次碎片整理

-XX:+ParallelCMSThreads 这是CMS的线程数，默认值(cpu数量 + 3)/4

7.G1收集器

G1收集器是面向服务端的垃圾收集器，赋予的使命是在未来替换掉CMS，与其他的收集器相比，G1具有如下特征

（1）并行和并发：G1能充分的利用多CPU,多核环境下的硬件优势，使用多个CPU来缩短Stop-The-World的停顿时间，部分其他垃圾收集器原本需要停顿java线程去执行GC动作，G1收集器仍然能通过并发的方式让JAVA程序继续执行

（2）分代收集：与其他收集器一样，分代概念在G1中仍然保留，虽然G1可以不需要其他的回收器配合就能独立管理整个GC堆，但是他能够采用不同的方式去处理新建的对象和已经存货了一段时间，熬过了多次GC的旧对象以获取更好的收集效果

（3）空间整理：和CMS算法不同，G1整体基于“标记-整理”算法，从局部（两个Regin之间）上看来是基于“复制”算法实现的，这两种算法都意味着G1运作期间不会产生空间碎片，收集后能提供规整的可用内存

（4）可预测停顿：降低停顿是G1和CMS的共同关注点，但G1出了追求停顿外、还可以建立可预测的停顿时间模型，能让使用者明确在一个长度为M毫秒的时间段内，消耗在垃圾回收商的时间不得超过M秒，机会是已经是实时java的垃圾回收期的特征了

G1管理的是整个范围的新生代和老年代，与之前的收集器不一样，G1收集器将java堆划分成多少个大小相等的独立区域（Region），虽然还保留有新生代和老年代的概念，但新生代和老年代不再是物理隔离，他们是一部分Region（不需要连续）的集合

G1跟踪各个Regin里面的垃圾堆积的价值大小（回收大小和所需时间），在后台维护一个有限列表，每次根据允许时间，有限回收价值最大的Region（这也是Garbage-First又来）。这种使用Region划分内存空间以及有优先级的区域回收方式，保证了G1收集器在有限的时间内可以获取尽可能高的收集效率

因为划分了多个Region，可能各个Region的对象相互引用，这样的话，不可能扫描整个java堆进行，因此Region之间的对象引用，虚拟机都是使用Remembered Set来避免全堆扫描的。G1中每个Region都有一个与之对应的Remembered Set，虚拟机发现程序在对Reference类型进行写操作的时候，会产生一个Write Barrier（隔离）暂时中止写操作，检查Reference引用是否处于不同的Region，如果处于不同的Region，便通过CardTable把相关信息记录到被引用对象所属Region的Remembered Set之中，如果进行内存回收是，在GC跟节点的枚举范围中加入Remebered Set即可保证部队全堆扫描也不会有遗漏

如果不计算维护Remebered Set的操作，G1收集器的运作大致分为以下几个步骤

（1）初始标记：标记一下GC roots能直接关联到的对象，并且修改Next Top at Mark start的值，让下一阶段的用户程序并发运行时，能在正确可用的Region中创建新对象，这个需要停顿线程，耗时短

（2）并发标记：从Gc roots开始对堆中的对象进行可达性分析，找到活动对象，这个阶段耗时长，但可以与用户线程并发执行

（3）最终标记：修改并发阶段因用户线程继续运行导致的标记产生变动的那一部分标记，虚拟机讲这段时间对象变化记录到在线程Remembered Set logs里面，最终阶段需要把logs的数据合并到Remebred Set中，需要停顿，但是可以并行执行

（4）筛选回收：筛选各个Region中回收成本和成本进行排序，根据用户期待的Gc停顿时间进行定制回收计划

使用方法：-XX:+UseG1GC -XX:MaxGCPauseMillis=50 （期待回收时间ms）