chenpt

随笔- 74 文章- 0 评论- 35 阅读- 13万

Jvm垃圾回收器（终结篇）

Jvm垃圾回收目前就准备了这三篇博文进行整理，在写博文的过程中我也是边看边记载的，我觉得这种学习方式更容易让人记住，不会轻易忘记。以前的学习模式都是看PDF文档、看书等，但是有个缺点就是当时记住了过段时间就会忘记，因此想把学习过程中重要的部分做个笔记总结，以便于后期复习回顾（学习技巧仅个人观点）同时也希望lz的博客能帮助到广大园友一丢丢。在此立个Flag！以后我会坚持写博客的。哈哈--好了 接下来言归正传。

知识回顾：

第一篇《Jvm垃圾回收器（基础篇）》主要讲述了判断对象的生死？两种基础判断对象生死的算法、引用计数法、可达性分析算法，方法区的回收。在第二篇《Jvm垃圾回收器（算法篇）》中主要介绍了垃圾回收的几种常用算法：标记-清除、复制算法、标记-整理算法、分代收集算法。那么接下来我们重点研究Jvm的垃圾收集器（serial收集器、parnew收集器、parallel scavenge收集器、serial old 收集器、parallel old收集器、cms收集器、g1收集器）。前面说了那么多就是为它做铺垫的。

正式进入前先看下图解HotSpot虚拟机所包含的收集器：

我的参与
最新评论
我的标签
更多链接

随笔分类

Java工具类(5)
Java基础(10)
Java进阶-设计模式(23)
Java虚拟机(6)
Spring(2)
spring-boot之路(8)
并发编程(3)
面试题(3)
数据库(4)
算法(2)
微服务(8)

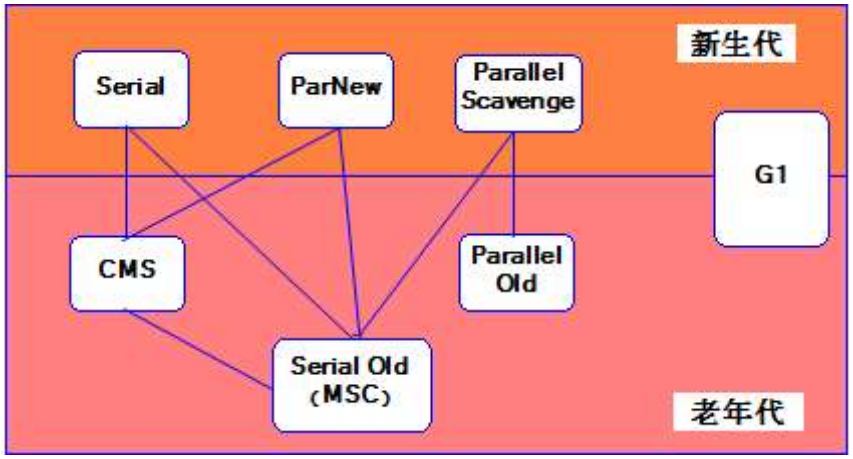
随笔档案

2021年4月(1)
2020年9月(7)
2020年8月(8)
2019年12月(1)
2019年6月(2)
2018年11月(3)
2018年10月(6)
2018年9月(8)
2018年8月(19)
2018年7月(13)
2018年6月(2)
2018年5月(1)
2018年4月(3)

最新评论

1. Re:SpringCloudAlibaba-入门学习之环境搭建
靠谱

--理葵



图中展示了7种作用于不同分代的收集器，如果两个收集器之间存在连线，则说明它们可以搭配使用。虚拟机所处的区域则表示它是属于新生代还是老年代收集器。

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

老年代收集器：CMS、Serial Old、Parallel Old

整堆收集器： G1

几个相关概念：

并行收集：指多条垃圾收集线程并行工作，但此时用户线程仍处于等待状态。

并发收集：指用户线程与垃圾收集线程同时工作（不一定是并行的可能会交替执行）。用户程序在继续运行，而垃圾收集程序运行在另一个CPU上。

吞吐量：即CPU用于运行用户代码的时间与CPU总消耗时间的比值（吞吐量 = 运行用户代码时间 / (运行用户代码时间 + 垃圾收集时间)）。例如：虚拟机共运行100分钟，垃圾收集器花掉1分钟，那么吞吐量就是99%

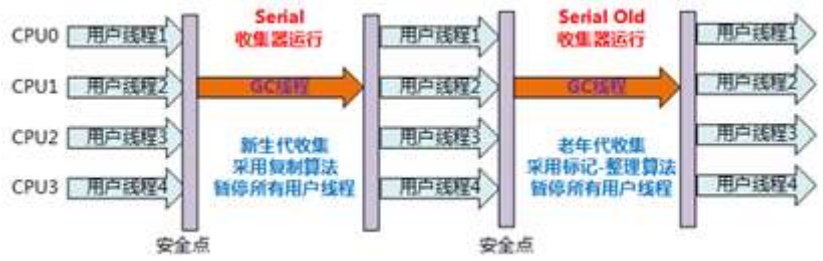
一：Serial 收集器

Serial收集器是最基本的、发展历史最悠久的收集器。

特点：单线程、简单高效（与其他收集器的单线程相比），对于限定单个CPU的环境来说，Serial收集器由于没有线程交互的开销，专心做垃圾收集自然可以获得最高的单线程手机效率。收集器进行垃圾回收时，必须暂停其他所有的工作线程，直到它结束（Stop The World）。

应用场景：适用于Client模式下的虚拟机。

Serial / Serial Old收集器运行示意图



二：ParNew收集器

ParNew收集器其实就是Serial收集器的多线程版本。

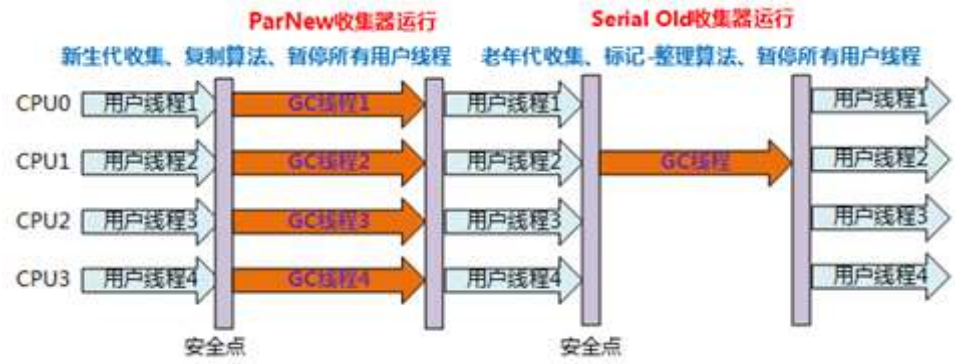
除了使用多线程外其余行为均和Serial收集器一模一样（参数控制、收集算法、Stop The World、对象分配规则、回收策略等）。

特点：多线程、ParNew收集器默认开启的收集线程数与CPU的数量相同，在CPU非常多的环境中，可以使用-XX:ParallelGCThreads参数来限制垃圾收集的线程数。

和Serial收集器一样存在Stop The World问题

应用场景：ParNew收集器是许多运行在Server模式下的虚拟机中首选的新生代收集器，因为它是除了Serial收集器外，唯一一个能与CMS收集器配合工作的。

ParNew/Serial Old组合收集器运行示意图如下：



三：Parallel Scavenge 收集器

与吞吐量关系密切，故也称为吞吐量优先收集器。

特点：属于新生代收集器也是采用复制算法的收集器，又是并行的多线程收集器（与ParNew收集器类似）。

该收集器的目标是达到一个可控制的吞吐量。还有一个值得关注的点是：GC自适应调节策略（与ParNew收集器最重要的一个区别）

GC自适应调节策略：Parallel Scavenge收集器可设置-XX:+UseAdptiveSizePolicy参数。当开关打开时不需要手动指定新生代的大小（-Xmn）、Eden与Survivor区的比例（-XX:SurvivorRation）、晋升老年代的对象年龄（-XX:PretenureSizeThreshold）等，虚拟机会根据系统的运行状况收集性能监控信息，动态设置这些参数以提供最优的停顿时间和最高的吞吐量，这种调节方式称为GC的自适应调节策略。

Parallel Scavenge收集器使用两个参数控制吞吐量：

- XX:MaxGCPauseMillis 控制最大的垃圾收集停顿时间
- XX:GCRatio 直接设置吞吐量的大小。

四：Serial Old 收集器

Serial Old是Serial收集器的老年代版本。

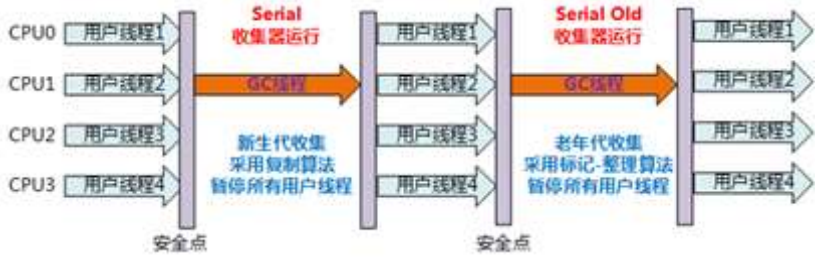
特点：同样是单线程收集器，采用标记-整理算法。

应用场景：主要也是使用在Client模式下的虚拟机中。也可在Server模式下使用。

Server模式下主要的两大用途（在后续中详细讲解…）：

1. 在JDK1.5以及以前的版本中与Parallel Scavenge收集器搭配使用。
2. 作为CMS收集器的后备方案，在并发收集Concurent Mode Failure时使用。

Serial / Serial Old收集器工作过程图（Serial收集器图示相同）：



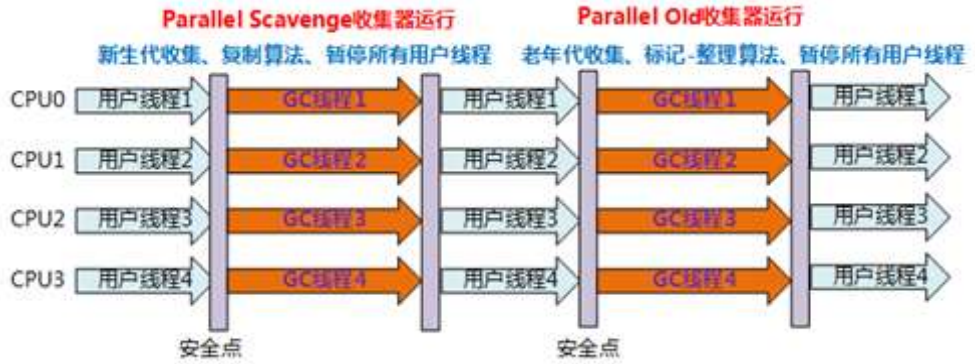
五：Parallel Old 收集器

是Parallel Scavenge收集器的老年代版本。

特点：多线程，采用标记-整理算法。

应用场景：注重高吞吐量以及CPU资源敏感的场合，都可以优先考虑Parallel Scavenge+Parallel Old 收集器。

Parallel Scavenge/Parallel Old收集器工作过程图：



六：CMS收集器

一种以获取最短回收停顿时间为目标的收集器。

特点：基于标记-清除算法实现。并发收集、低停顿。

应用场景：适用于注重服务的响应速度，希望系统停顿时间最短，给用户带来更好的体验等场景下。如web程序、b/s服务。

CMS收集器的运行过程分为下列4步：

初始标记：标记GC Roots能直接到的对象。速度很快但是仍存在Stop The World问题。

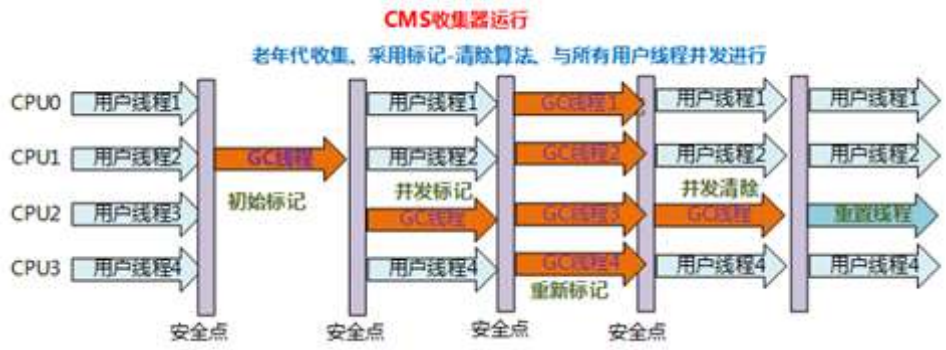
并发标记：进行GC Roots Tracing 的过程，找出存活对象且用户线程可并发执行。

重新标记：为了修正并发标记期间因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。仍然存在Stop The World问题。

并发清除：对标记的对象进行清除回收。

CMS收集器的内存回收过程是与用户线程一起并发执行的。

CMS收集器的工作过程图：



CMS收集器的缺点：

- 对CPU资源非常敏感。
- 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的产生。
- 因为采用标记-清除算法所以会存在空间碎片的问题，导致大对象无法分配空间，不得不提前触发一次Full GC。

七：G1收集器

一款面向服务端应用的垃圾收集器。

特点如下：

并行与并发：G1能充分利用多CPU、多核环境下的硬件优势，使用多个CPU来缩短Stop-The-World停顿时间。部分收集器原本需要停顿Java线程来执行GC动作，G1收集器仍然可以通过并发的方式让Java程序继续运行。

分代收集：G1能够独自管理整个Java堆，并且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、熬过多次GC的旧对象以获取更好的收集效果。

空间整合：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。

可预测的停顿：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毫秒的时间段内，消耗在垃圾收集上的时间不得超过N毫秒。

G1为什么能建立可预测的停顿时间模型?

因为它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Region里面的垃圾堆积的大小，在后台维护一个优先列表，每次根据允许的收集时间，优先回收价值最大的Region。这样就保证了在有限的时间内可以获取尽可能高的收集效率。

G1与其他收集器的区别：

其他收集器的工作范围是整个新生代或者老年代、G1收集器的工作范围是整个Java堆。在使用G1收集器时，它将整个Java堆划分为多个大小相等的独立区域（Region）。虽然也保留了新生代、老年代的概念，但新生代和老年代不再是相互隔离的，他们都是一部分Region（不需要连续）的集合。

G1收集器存在的问题：

Region不可能是孤立的，分配在Region中的对象可以与Java堆中的任意对象发生引用关系。在采用可达性分析算法来判断对象是否存活时，得扫描整个Java堆才能保证准确性。其他收集器也存在这种问题（G1更加突出而已）。会导致Minor GC效率下降。

G1收集器是如何解决上述问题的?

采用Remembered Set来避免整堆扫描。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类型进行写操作时，会产生一个Write Barrier暂时中断写操作，检查Reference引用对

象是否处于多个Region中（即检查老年代中是否引用了新生代中的对象），如果是，便通过CardTable把相关引用信息记录到被引用对象所属的Region的Remembered Set中。当进行内存回收时，在GC根节点的枚举范围中加入Remembered Set即可保证不对全堆进行扫描也不会有遗漏。

如果不计算维护 Remembered Set 的操作，G1收集器大致可分为如下步骤：

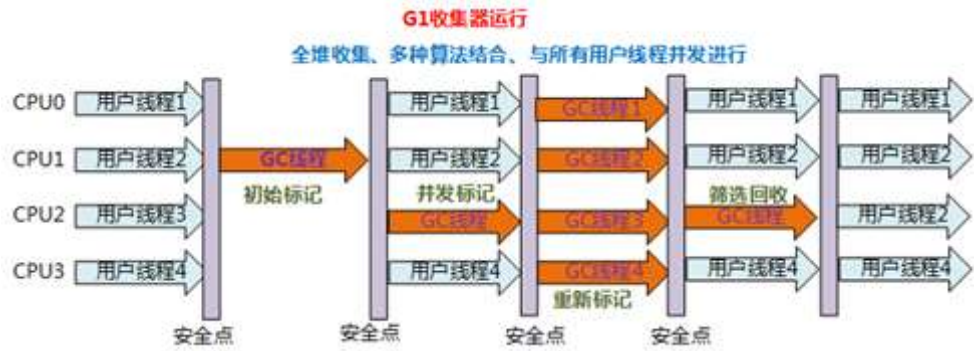
初始标记：仅标记GC Roots能直接到的对象，并且修改TAMS（Next Top at Mark Start）的值，让下一阶段用户程序并发运行时，能在正确可用的Region中创建新对象。（需要线程停顿，但耗时很短。）

并发标记：从GC Roots开始对堆中对象进行可达性分析，找出存活对象。（耗时较长，但可与用户程序并发执行）

最终标记：为了修正在并发标记期间因用户程序执行而导致标记产生变化的那一部分标记记录。且对象的变化记录在线程Remembered Set Logs里面，把Remembered Set Logs里面的数据合并到Remembered Set中。（需要线程停顿，但可并行执行。）

筛选回收：对各个Region的回收价值和成本进行排序，根据用户所期望的GC停顿时间来制定回收计划。（可并发执行）

G1收集器运行示意图：



-- 结束-- JVM垃圾收集暂告一段落。

作者： 不二尘
出处： https://home.cnblogs.com/u/chenpt/

分类: Java虚拟机

« 上一篇： Jvm垃圾回收器（算法篇）
» 下一篇： 记一次电话面试的题目

posted @ 2018-10-18 11:16 不二尘 阅读(36391) 评论(4) 编辑 收藏 举报

编辑推荐：
· CSS 世界中的方位与顺序
· 在 .NET 中创建对象的几种方式的对比
· 10倍程序员的思考模型
· 学习 CLR 源码：连续内存块数据操作的性能优化
· 记一次大数据量后台服务的性能优化

最新新闻:
· B站焊武帝爆火出圈：纯手工拼晶体管自制CPU 可跑程序
· 恐龙会不会是灭绝于一场核战争？
· 华为昇腾AI正在壮大！开发者超35万
· 比亚迪回应汉EV碰撞后静置起火：媒体测试存在诸多疑点、将安排现场排查
· 这个用屁股开车的猛男 让无数网友当起了侦探
» 更多新闻...