面试常考

> CMS收集器

- 初始标记(Stop the world)
- 并发标记
- 预清理
- 可被终止的预清理
- 重新标记(Stop the world)
- 并发清理
- 并发重置

> G1收集器

- 初始标记(Stop the world)
- 并发标记
- 最终标记(Stop the world)
- 筛选回收(Stop the world)

在G1中，还有一种特殊的区域，叫Humongous区域。 如果一个对象占用的空间超过了分区容量50%以上，G1收集器就认为这是一个巨型对象。这些巨型对象，默认直接会被分配在年老代，但是如果它是一个短期存在的巨型对象，就会对垃圾收集器造成负面影响。为了解决这个问题，G1划分了一个Humongous区，它用来专门存放巨型对象。如果一个H区装不下一个巨型对象，那么G1会寻找连续的H分区来存储。为了能找到连续的H区，有时候不得不启动Full GC。

G1收集器

Garbage-First Collector 适合服务端，多处理器，大内存的场景。可以很大概率的满足预期的停顿时间，同时实现高吞吐量。在JDK7之后才支持的。

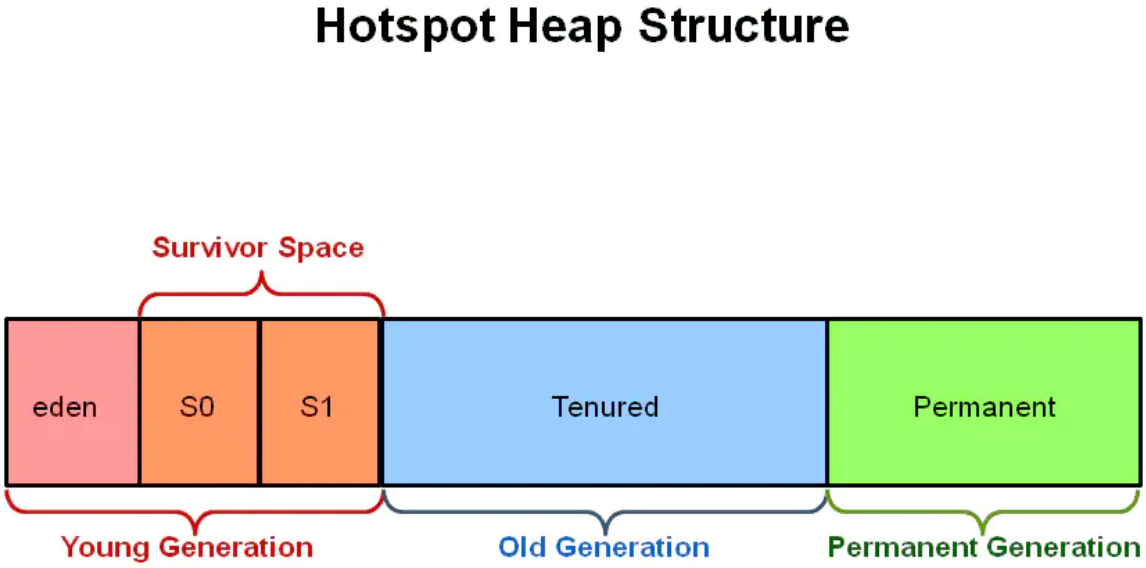
- 可以和CMS收集器等应用程序同时运行
- 在较短的停顿时间内完成内存空闲碎片的整理
- 不需要更大的Java堆
- 不想牺牲过多的吞吐量
- 有更可预测的GC暂停持续时间

> G1 计划作为并发标记扫描CMS的长期替代品：

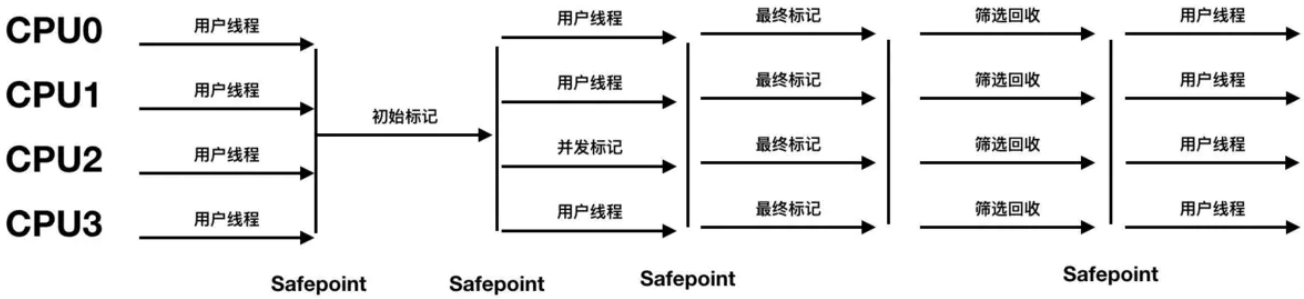
- G1 是一个压缩算法的实现，充分的压缩空间避免使用细粒度的自由列表进行分配
- 大部分消除了潜在的碎片问题
- 比CMS有更多的可预测的垃圾收集暂停

G1概述

CMS会把堆分为： 新生代，老年代，永久代



G1 把堆分割成了一组大小相等的堆区域，每个区域都是连续的虚拟内存范围，分为Eden, Survivor, old



初始标记（STW）

- 仅仅标记一下GC Roots能直接关联到的对象，这阶段需要线程停顿，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，这阶段耗时较长，但可与用户线程并发执行

最终标记 （STW）

- 修正在并发标记期间因用户线程继续运行导致标记发生变化的那一部分标记记录
- 并发期间，虚拟机会把这段时间发生变化的对象记录在线程Remember Set log 里面

最后回收阶段（STW）

- 首先对各个regin的回收价值和成本进行排序，根据用户期望的停顿时间来制定回收计划，
- 只回收一部分的Regin
- STW将大幅度提高收集效率，

G1 特点

- 划分Region
- 有优先级的垃圾回收
- 保证在有限的时间内获取尽可能高的收集效率

并行与并发

- 从分利用多CPU, 多核环境硬件优势，利用多个CPU来缩短stop the word停顿时间，部分其它收集器需要停顿java线程执行的GC操作，G1 收集器任然可以通过并发的方式让java程序继续执行

分代收集

- 仍然分了三代（区域，Region），Eden, Survivor, old(不需要连续)，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，熬过多次GC的旧对象以获得更好的收集效果。

空间整合

- CMS 是标记-清理，而G1是标记-整理
- 从局部的两个区域看是基于复制算法实现的，意味着不会产生内存碎片，能提供规整的可用内存

可预测的停顿

建立了一个可预测的停顿时间模型，避免在整个java堆中进行全区域的垃圾收集，G1跟踪各个Region里面的垃圾堆积的价值大小，优先回收价值最大的Regin（这也是Garbage First名称的由来）

推荐使用G1的场景

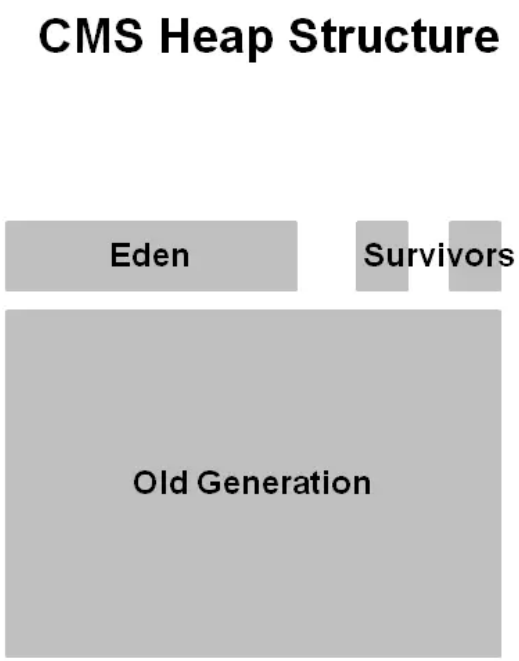
- Full GC 持续时间太长或太频繁
- 对象分配率或提升率明显不同
- 不想要长时间GC停顿

> 如果CMS或者 Parllerl Old GC 没有经历过长时间的GC停顿，完全不用切换到G1

复习垃圾收集步骤

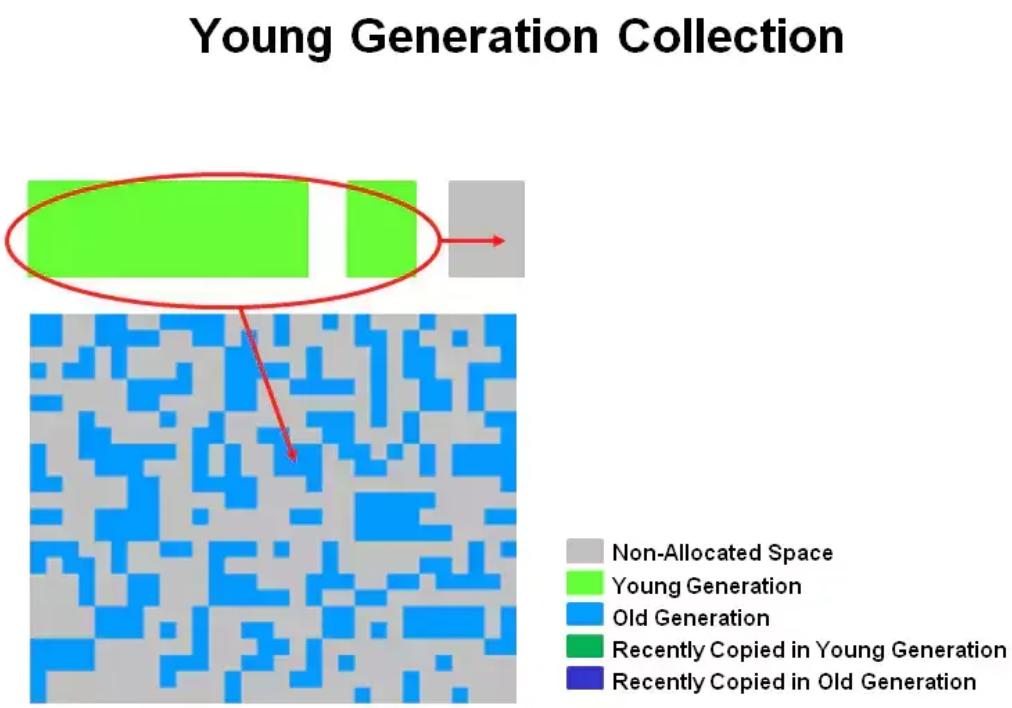
CMS 堆结构

新生代分为了Eden和两个suvivor区域， 老年代是一个连续的空间，除非进行碎片整理（就是一次full GC）



新生代的收集

活着的对象会从Eden和suvivor 被复制到另一个suvivor区域，年龄太大就会复制晋升到老年代



参考

- 《深入理解Java虚拟机——JVM高级特性与最佳实践》 - 周志明
- 英文原文
- 简书
- 掘金

下一篇我们会带领大家一起看看如何优化Java GC，敬请期待！