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)

CMS的初衷和目的：为了消除Throught收集器和Serial收集器在Full GC周期中的长时间停顿。 CMS的适用场景：如果你的应用需要更快的响应，不希望有长时间的停顿，同时你的CPU资源也比较丰富，就适合适用CMS收集器。

CMS提供了两个参数来解决这个问题： （1）UseCMSCompactAtFullCollection，在要进行Full GC的时候进行内存碎片整理； （2）CMSFullGCsBeforeCompaction，每隔多少次不压缩的Full GC后，执行一次带压缩的Full GC。

下面分别介绍两个收集器：

CMS 简介

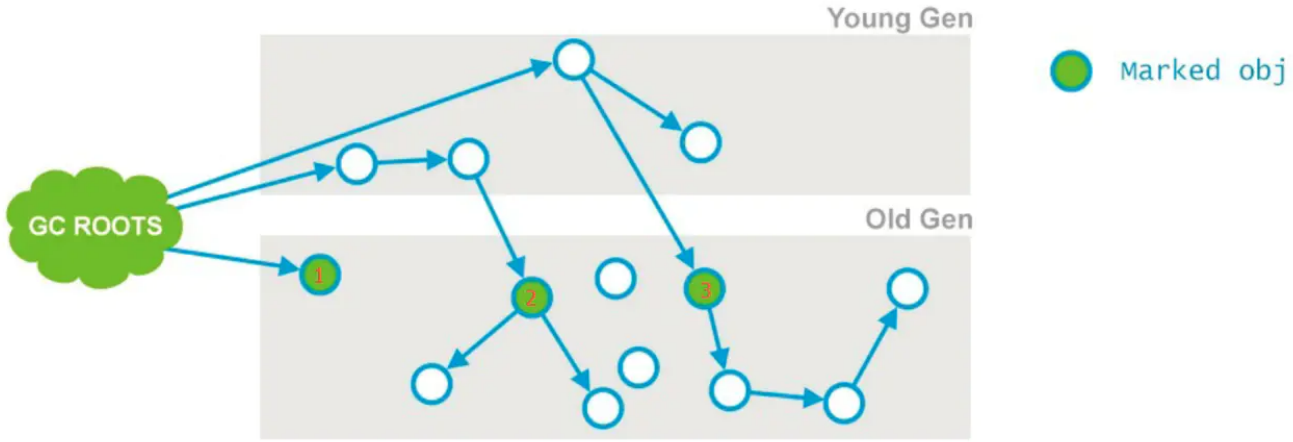
CMS全称Concurrent Mark Sweep，从名字可知是基于“标记——清除”算法实现。以获取最短回收停顿时间为目标；CSM为`老年代垃圾收集器`，在收集的过程中`可以与用户线程并发操作`。

- CMS牺牲了系统的吞吐量来追求收集速度，适合追求垃圾收集速度的服务器。
- 可以通过JVM启动参数: -XX：+UseConcMarkSweepGC来开启CMS
- `收集的是没有被标记的对象`

CMS 清理的详细过程

初始标记 --- 标记老年代符合回收条件的存活对象

- 标记老年代中所有的GC Roots对象,如节点1
- 标记老年代中被年轻代中活着的对象引用的对象，如节点2，3



GC Roots 对象包括如下几种:

1. 虚拟机栈中的引用的对象（栈帧中的本地变量表）
2. 方法区中的类静态属性引用的对象
3. 方法区中的常量引用的对象
4. 本地方法栈中JNI的引用的对象

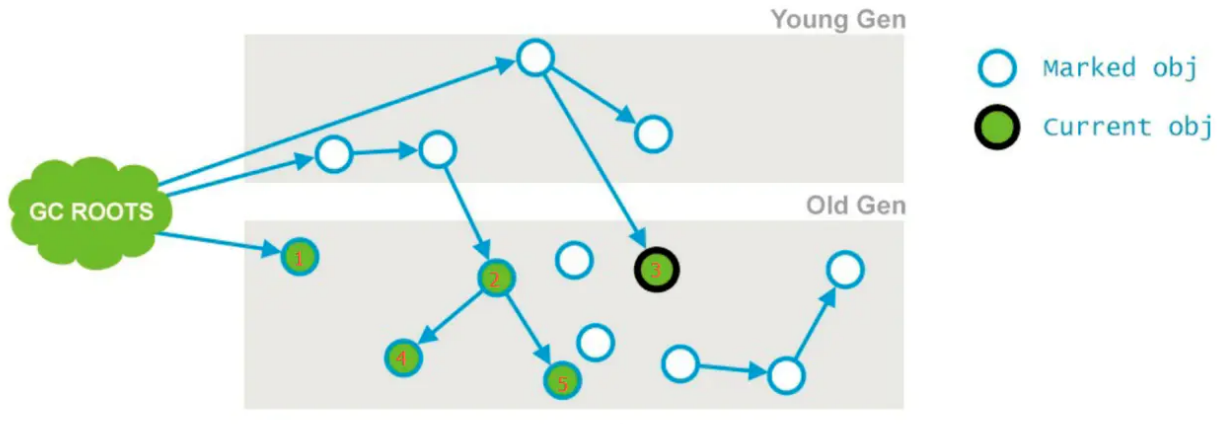
> ps : 为了加快此阶段的处理速度，减少停顿时间，可以开启初始标记并行化，同时调大并行标记的线程数，线程数不要超过cpu的核数

并发标记

> 特点：和应用程序并发运行，会导致对象的引用发生改变 （如新生代的对象晋升到了老年代，或者更新了引用关系）

- 重新标记，避免漏标，遗漏的情况
- 只负责把上述发生变化的对象所在的Card标识为Dirty, 后续只扫描Dirty Card, 避免扫描整个老年代，不负责处理
- 由于和用户线程并发运行，可能会导致concurrent mode failure

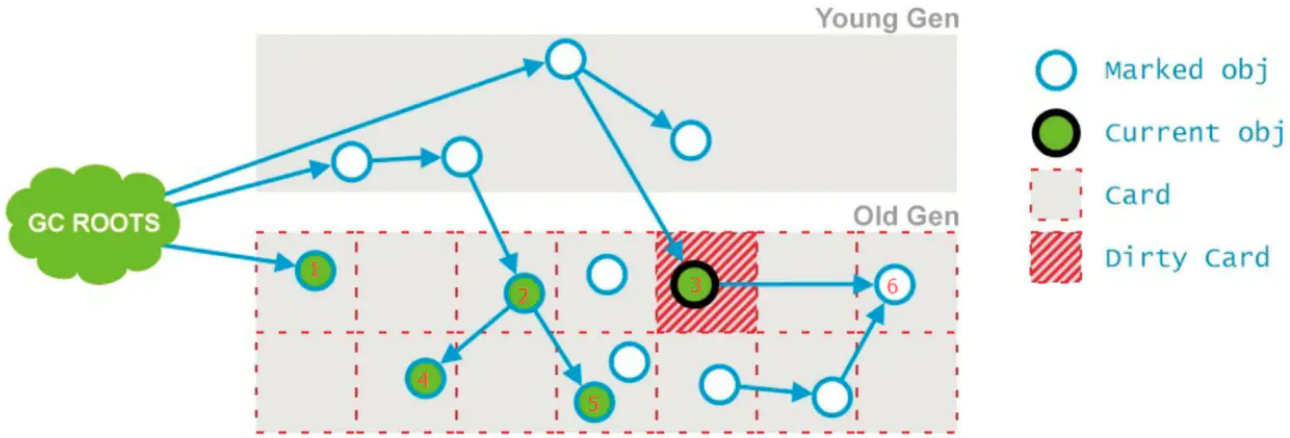
> ps：仍然可能有对象引用关系发生了改变导致没清理到



预清理阶段

- 用来处理前一个阶段因为引用关系改变导致没有标记到的存活对象（前一阶段仍可能遗漏）

如下图3改变了，被标记为Dirty



可终止的预处理

- 此阶段重复的做标记工作，直到发生abort条件，如持续时间超过5s等
- 之所以是5s，是期待5s内能有ygc，清理年轻代的引用

重新标记

会导致很长的stop the word 来整理发生的改变

> 目标是完成标记整个老年代的所有的存活对象，范围是整个堆（包括young 和 old）

- 重新标记前一般会执行一次ygc, 回收到年轻代无用的对象，或放入幸存区，或晋升到老年代。
- 扫描年轻代时，只需要扫描幸存区对象即可，一般很小，大大节省了时间

并发清理

- 老年代所有符合回收条件的对象(没被标记)，通过Garbage Collector 采用清扫的方式回收那些不能用的对象 并回收空间

> 由于CMS清理时用户线程还在运行，自然还有新的垃圾产生，这些垃圾出现在标记过程之后，CMS无法当次清理掉，只好在下一次GC时在清理。

CMS 需要注意的点

> 减少remark阶段停顿

- 一般CMS的GC 耗时80% 都在remark, 如果觉得时间长，可以通过参数： -XX:+CMSScavengeBeforeRemark，在remark操作之前做一次Young GC，目的是减少年轻代对老年代的无效引用，降低remark时的开销

> 内存碎片问题

- CMS基于标记-清楚算法，只会删除无用的对象，不会对内存做压缩，会造成内存碎片，默认是0，既等到CMS GC顶不住了才会压缩。可以通过参数-XX：CMSFullGCsBeforeCompaction=n进行配置

> concurrent mode failure

- 由于新生代空间放不下了需要放入老年代，CMS还没有机会回收老年代产生的。 可以通过参数设置CMS达到了内存的百分比70%后开始GC

> promotion failed （过早提升或提升失败）

早提升的原因：

1. 新生代幸存区太小，放不下，老年代也放不下导致的，由于碎片化，大的对象到老年代找不到一段连续区域存放新生代的对象导致的，将导致遍历整个堆
2. 对象太大，没有足够的空间来放了

提升失败的原因：

1. 老年代空闲空间不够
2. 老年代空闲多，但是碎片太多，不连续

解决方法

- 提升失败，若是内存碎片导致，需要空间整理压缩
- 提升过快，可以尝试调大Survivor
- 如果是老年代空间不够，尝试将CMS触发的阈值调低

总结

1. CMS 只收集老年代
2. 收集过程分为6个阶段，重新标记阶段最耗时，可以先进行一次yGC
3. 为了减少失败或异常，可以让CMS尽早GC（达到内存阈值百分比），一定次数后做一次内存压缩，减少内存碎片。
4. `对象提升失败，进而触发Full GC`

参考资料

- 《深入理解Java虚拟机——JVM高级特性与最佳实践》 - 周志明
- 掘金博文

下一篇我们将介绍G1 收集器，敬请期待！