# CMS 的 STW 问题

**GC 的触发：**

新生代回收发生在新生代内存已经满了，或者说剩余内存小于即将 new 出来的对象的体积的时候。此时会发生新生代 GC。

老年代回收发生在剩余内存无法装载新生代存活的对象的时候和无法装载大对象的时候（大对象直接进入老年代）。

**CMS 可以进行并发清理的原因：**

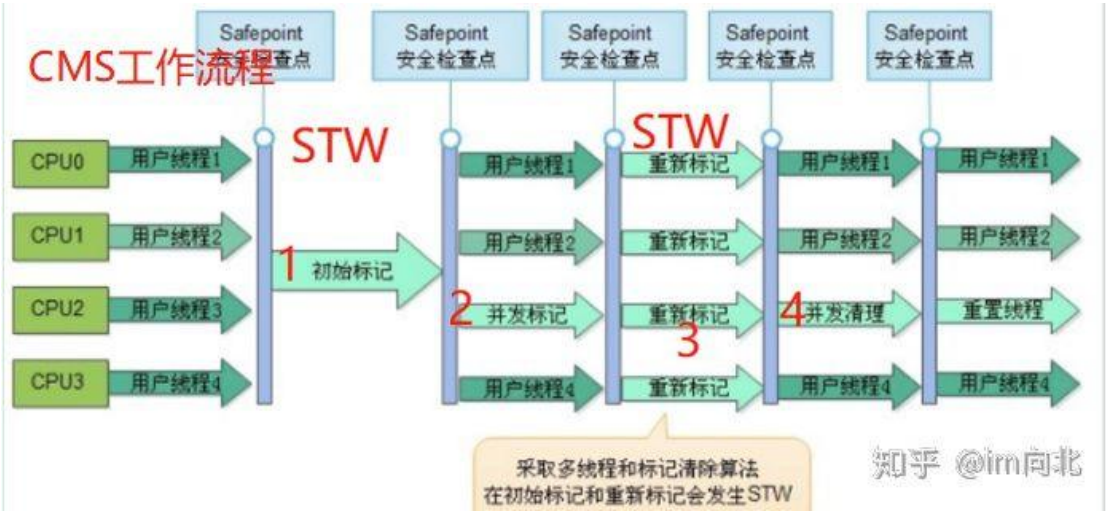
基于 Mark & Sweep 的算法并不会挪动存活引用的位置，因此不会影响程序的正常运行，因此并发清理过程可以用户线程同时进行。

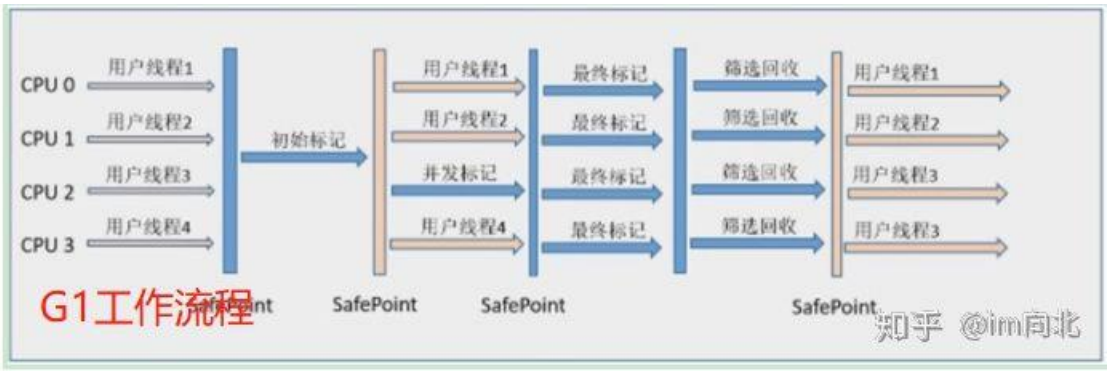
·如果它要选用 mark-compact 为基本算法的话，就只有 mark 阶段可以并发执行（其中 root scanning 阶段仍然需要暂停 mutator，这是 initial marking；后面的 concurrent marking 才可以跟 mutator 并发执行），然后整个 compact 阶段都要暂停 mutator。回想最初提到的：compact 阶段的时间开销与活对象的大小成正比，这对年老代来说就不划算了。

于是选用 mark-sweep 为基本算法就是很合理的选择：mark 与 sweep 阶段都可以与 mutator 并发执行。Sweep 阶段由于不移动对象所以不用修正指针，所以不用暂停 mutator。

**初始标记的 STW：**

枚举根节点是一定需要停顿所有的执行线程的，因为要在一个能确保一致性的快照中进行分析，初始标记就是用来枚举根节点以及根节点能直接关联的对象的




**问题 1：CMS 和 G1 的重新标记，最终标记的目的是什么？**

工作流程上来看，CMS 的重新标记，和 G1 的最终标记之前，都是并发标记。

并发标记就是，标记程序和用户程序同时执行。既然是同时运行，用户程序就可能修改对象的引用关系。

修改对象引用关系就可能影响 GC 回收。所以，CMS 重新标记，G1 最终标记都是为了解决一件事，那就是 并发标记过程中用户程序修改了对象引用关系后，如何让 GC 收集器仍旧能正确回收垃圾对象的问题。

**问题 2：CMS 和 G1 是如何解决并发标记过程中 对象引用被更改的问题？**

论 CMS 和 G1，并发标记阶段的结果都是 获得 一个可到达的对象关系；然后 GC 程序按照这个关系进行 GC。但是，有可能并发标记的同时，用户程序也修改了对象-引用关系

这就出现一个信息合并的问题：

1）并发标记产生的可到达对象关系 Rcon; R 是 Relationship 关系；

2）用户程序修改的对象引用关系，Ruser；

如果把 Ruser 和 Rcon 合并一下，形成一个新的，完整的可到达对象关系 Rfinal，交给 GC 程序，是不是就完美解决问题了。

那如何处理这个信息合并的问题呢？

CMS 和 G1 都采取一种方式 Write barrier+log，3 个步骤：

1）Write barrier 监听用户修改对象引用关系

2）监听同时写日志

用户程序修改引用关系时候，监听并记录 Ruser 关系到日志 Rslog（关键词是 Remember Set log）

3）然后合并

后期处理 Rslog 的过程中，把 Ruser 这部分信息与 Rcon 合并成 Rfinal，然后用 Rfinal 进行 GC

这里的后期处理，就是 CMS 的 remark 阶段和 G1 的 final mark 阶段。

**问题 3：CMS 和 G1 的 Rslog 中记录的是什么？**

Rslog 的作用就是记录用户程序对对象关系的修改；

用户程序的修改只能有 2 种：

1）增加了引用-对象关系；Object o=new Object();或者 o1=new Object()；

2）删除了引用-对象关系； o=null;

而这两种刚好是 CMS 和 G1 实现的方式，

对于 CMS，Rslog 记录的新增关系；删除关系会被忽略，变成浮动垃圾，下一次 GC 回收；

writer barrier 是 insert barrier，会监听新增的引用关系，并记录日志

所以 CMS 会处理新增关系，忽略删除关系

对于 G1，Rslog 记录的删除关系；新增关系会被忽略，变成浮动垃圾，以后 GC 处理；

writer barrier 是 delete barrier，会监听引用关系删除，并记录日志，所以 G1 会处理删除关系，忽略新增关系。

G1 说，我是以 region 为目标回收的，新增关系的情况，我记录到其他 region A 就可以了；至于 A region 啥时候被回收，要看具体情况了！