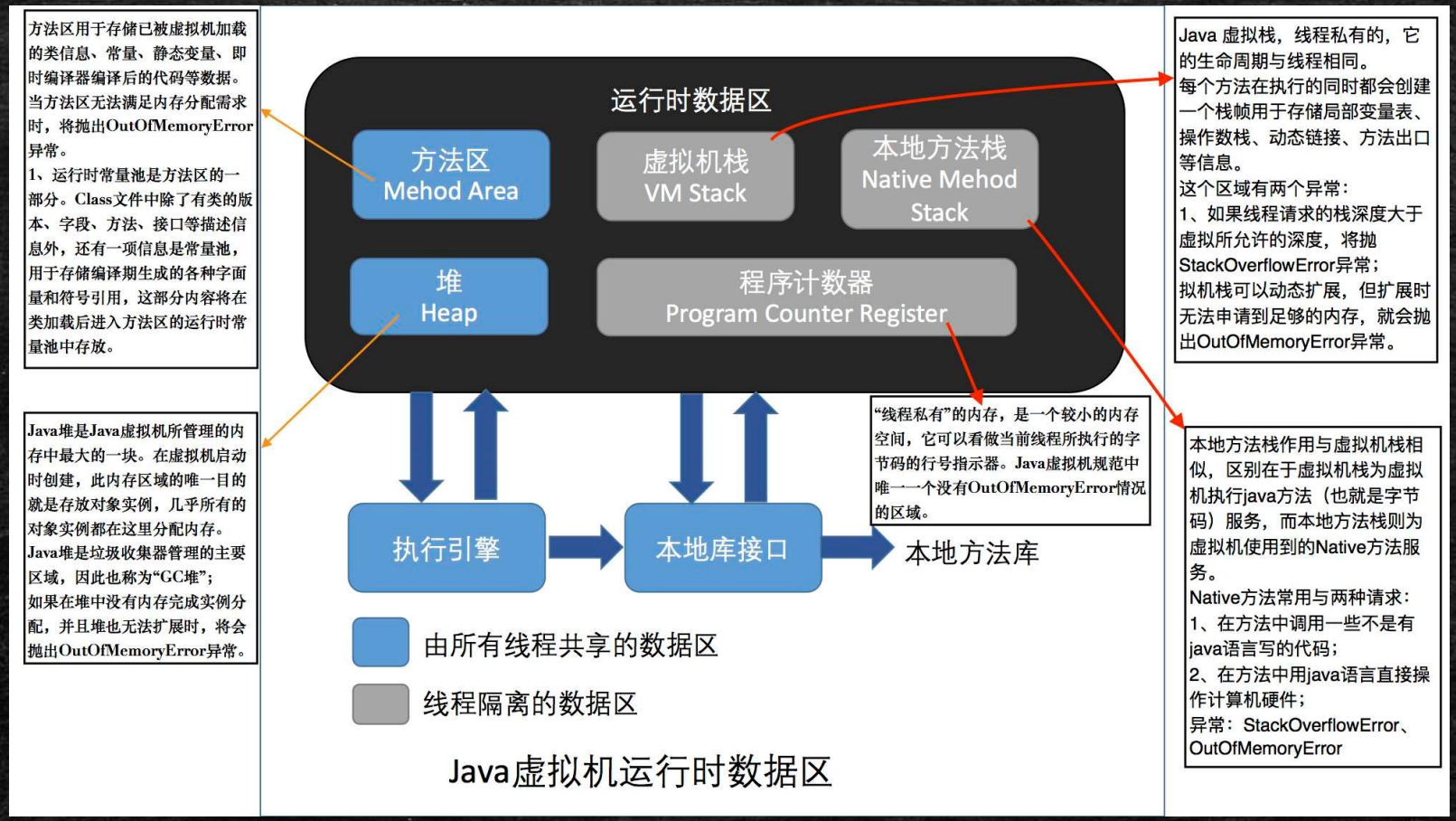
CMS垃圾收集器

- 连老师

Java虚拟机内存模型



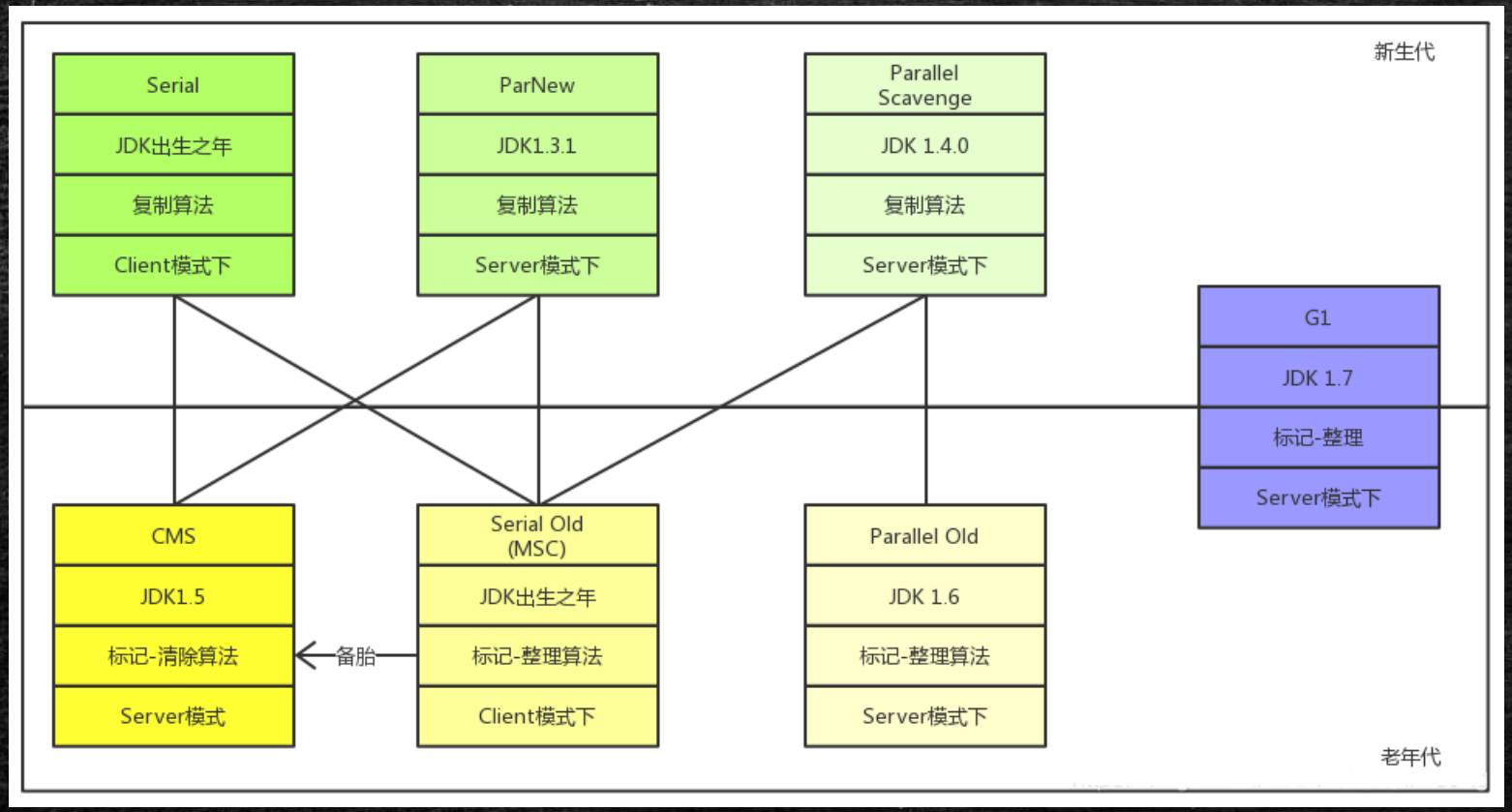
垃圾回收算法

- 引用计数法
- 根搜索算法
- 标记-清除算法（Mark-Sweep）
- 复制算法（Copying）
- 标记-压缩算法（Mark-Compact）
- 增量算法（Incremental Collecting）
- 分代收集算法（Generational Collecting）

Garbage Collection

- Serial收集器
- ParNew收集器
- Parallel收集器
- CMS收集器
- Garbage First GC

垃圾收集器的实现机制


新生代
Serial
JDK出生之年
复制算法
Client模式下
ParNew
JDK1.3.1
复制算法
Server模式下
Parallel Scavenge
JDK 1.4.0
复制算法
Server模式下
G1
JDK 1.7
标记-整理
Server模式下
CMS
JDK1.5
标记-清除算法
Server模式
备胎
Serial Old (MSC)
JDK出生之年
标记-整理算法
Client模式下
Parallel Old
JDK 1.6
标记-整理算法
Server模式下
老年代

新生代垃圾回收器的比较

名称	串行/并行	回收算法	使用场景	是否可以与cms配合
SerialGC	串行	复制	单cpu	是
ParNewGC	并行	复制	多cpu	是
ParallelScavengeGC	并行	复制	多cpu且关注吞吐量	否

老年代回收器

名称	串行/并行/并发	回收算法	适用场景
SerialOldGC	串行	标记整理	单cpu
ParNewOldGC	并行	标记整理	多cpu
CMS	并发，几乎不会暂停用户线程	标记清除	多cpu且与用户线程共存

CMS

CMS(Concurrent Mark Sweep)收集器是一种以获取最短回收停顿时间为目标的收集器，适用于集中在互联网站或者B/S系统的服务端的Java应用。CMS收集器是基于“标记-清除”算法实现的，可以跟新生代的Parallel New、Serial搭配使用。

CMS的特性

- 1、cms指挥回收老年代和永久代的垃圾，不会收集年轻代
- 2、cms是一种预处理垃圾回收器，它不能等到old内存用尽时回收，需要在内存用尽之前，完成回收操作，否则会导致并发回收失败，所以cms垃圾回收器开始执行回收操作，有一个触发阈值，JDK6之前的默认阈值时68%，而JDK6及以上版本是92%。
- 3、cms不会移动对象以保证空闲空间的连续性，相反，cms保存所有空闲内存片段的列表，通过这样的方式，cms可以避免为存活对象重新分配位置引起的开销，但是相应的会引起内存的碎片化。

CMS的执行步骤

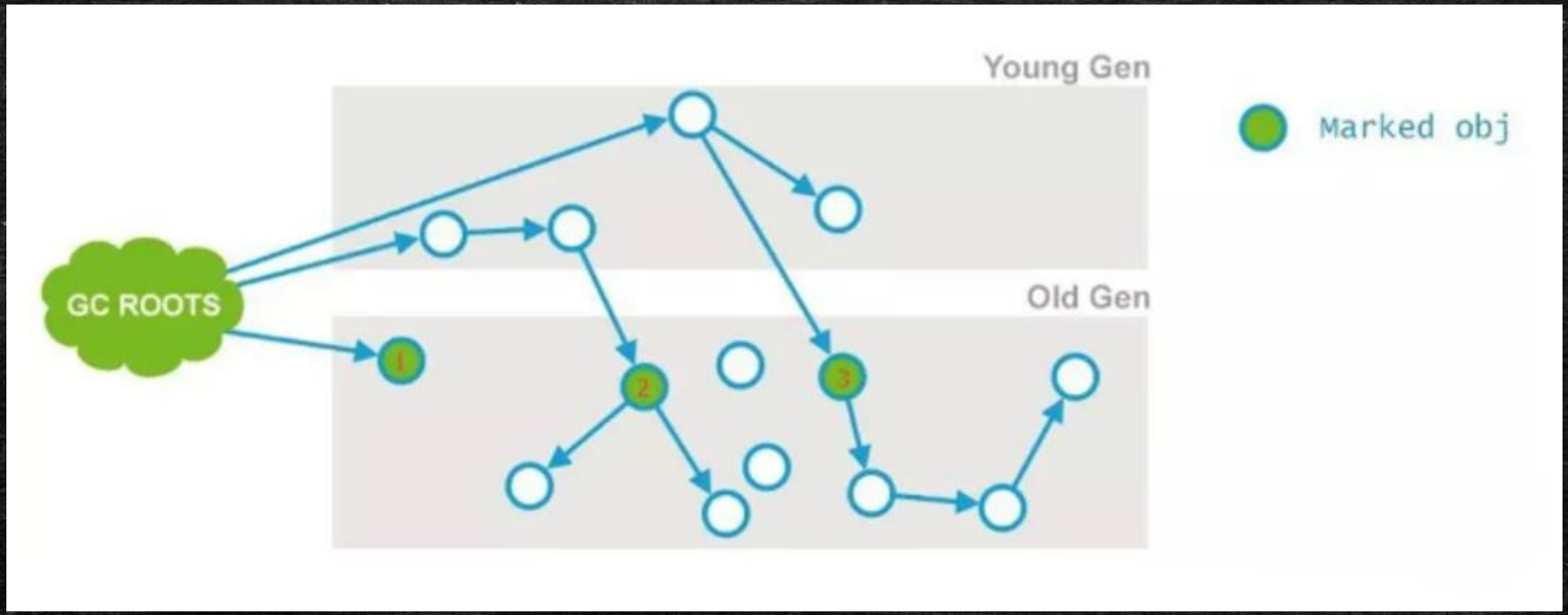
步骤	阶段	名称	功能	是否STW
1	CMS-initial-mark	初始标记	标记GC ROOT能直接关联到的对象	是
2	CMS-concurrent-mark	并发标记	由前阶段标记过的对象触发，所有可达的对象都在本阶段标记	否
3	CMS-concurrent-preclean	并发预处理	此阶段标记从新生代晋升的对象，新分配到老年代的对象以继在并发阶段被修改了的对象	否
4	CMS-remark	重标记	重新扫描堆中的对象，进行可达性分析，标记活着的对象	是
5	CMS-concurrent-sweep	并发清理	回收在对象图中不可达对象	否
6	CMS-concurrent-reset	并发重置	做一些收尾，以便下一次GC	否

运行程序检测

- 添加参数运行：
- -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+PrintGC -XX:+PrintGJava注解又称Java标注，是Java语言5.0版本开始支持加入源代码的特殊语法元数据。为我们在代码中添加信息提供了一种形式化的方法，使我们可以在稍后某个时刻非常方便的使用这些数据。
  Java语言中的类、方法、变量、参数和包等都可以被标注。和Javadoc不同，Java标注可以通过反射获取注解内容。在编译器生成类文件时，注解可以被嵌入到字节码中。Java虚拟机可以保留注解内容，在运行时可以获取到注解内容。
  CDetails -XX:+PrintGCDateStamps -XX:+PrintHeapAtGC -Xloggc:tmp/gctest.log

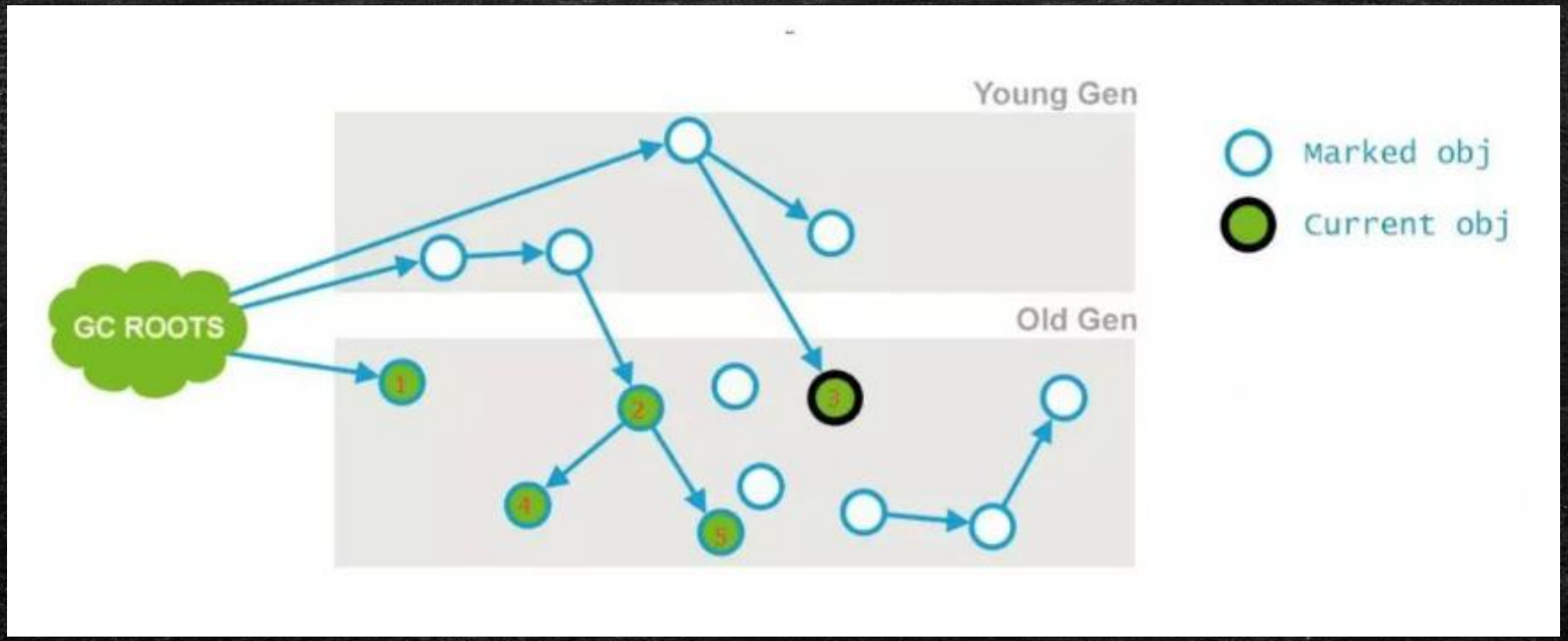
初始标记

- 标记老年代中国所有的GC Roots对象
- 标记年轻代中活着的对象引用到的老年代的对象（指的是年轻代中还存活的引用类型对象，引用指向老年代中的对象）
- GC Roots对象包括：
  - 1、虚拟机栈中的引用的对象
  - 2、类静态属性引用的对象
  - 3、常量池引用的对象
  - 4、本地方法栈中JNI的引用的对象


Young Gen
Marked obj
GC ROOTS
Old Gen
1
2
3

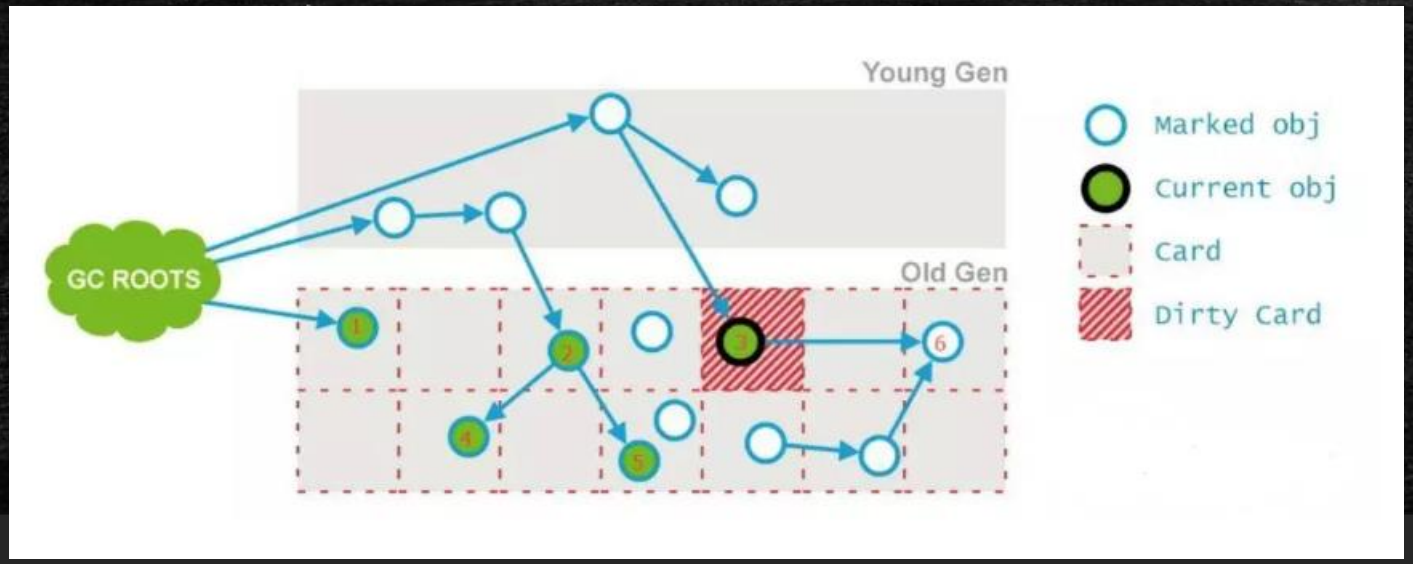
并发标记

因为是并发运行的，在运行期间会发生新生代的对象晋升到老年代、或者是直接在老年代分配对象、或者更新老年代对象的引用关系等等，对于这些对象，都是需要进行重新标记的，否则有些对象就会被遗漏，发生漏标的情况。为了提高重新标记的效率，该阶段会把上述对象所在的Card标识为Dirty，后续只需扫描这些Dirty Card的对象，避免扫描整个老年代；

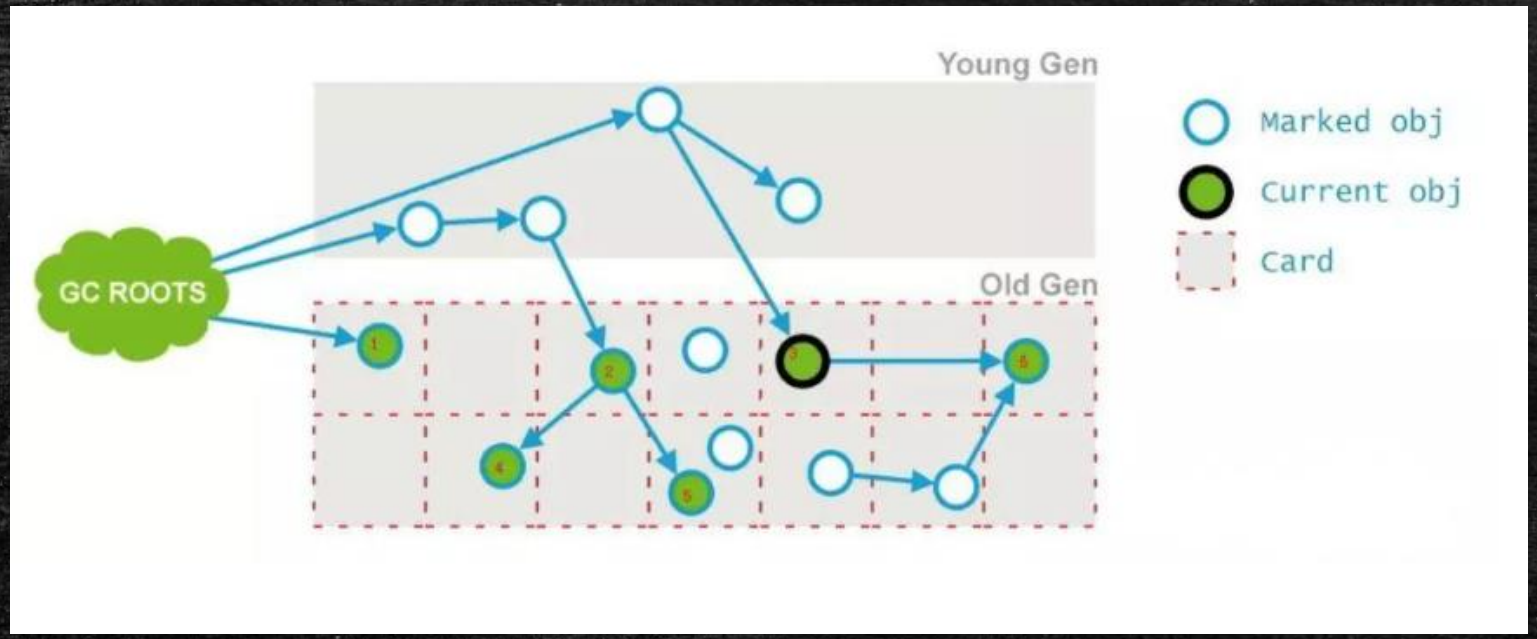

Young Gen
Old Gen
GC ROOTS
1
2
3
4
5
Marked obj
Current obj

预清理阶段

- 前一个阶段已经说明，不能标记出老年代全部的存活对象，是因为标记的同时应用程序会改变一些对象引用，这个阶段就是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的，它会扫描所有标记为Direty的Card
- 如下图所示，在并发清理阶段，节点3的引用指向了6；则会把节点3的card标记为Dirty；



- 最后将6标记为存活,如下图所示:



重新标记

- 这个阶段会导致第二次stop the word，该阶段的任务是完成标记整个年老代的所有的存活对象。
- 这个阶段，重新标记的内存范围是整个堆，包含_young_gen和_old_gen。为什么要扫描新生代呢，因为对于老年代中的对象，如果被新生代中的对象引用，那么就会被视为存活对象，即使新生代的对象已经不可达了，也会使用这些不可达的对象当做cms的“gc root”，来扫描老年代； 因此对于老年代来说，引用了老年代中对象的新生代的对象，也会被老年代视作“GC ROOTS”:当此阶段耗时较长的时候，可以加入参数-XX:+CMSScavengeBeforeRemark，在重新标记之前，先执行一次ygc，回收掉年轻带的对象无用的对象，并将对象放入幸存带或晋升到老年代，这样再进行年轻带扫描时，只需要扫描幸存区的对象即可，一般幸存带非常小，这大大减少了扫描时间

并发清理

- 通过以上5个阶段的标记，老年代所有存活的对象已经被标记并且现在要通过Garbage Collector采用清扫的方式回收那些不能用的对象了。
- 这个阶段主要是清除那些没有标记的对象并且回收空间；
- 由于CMS并发清理阶段用户线程还在运行着，伴随程序运行自然就还会有新的垃圾不断产生，这一部分垃圾出现在标记过程之后，CMS无法在当次收集中处理掉它们，只好留待下一次GC时再清理掉。这一部分垃圾就称为“浮动垃圾”。

并发重置

- 这个阶段并发执行，重新设置CMS算法内部的数据结构，准备下一个CMS生命周期的使用.

优化

1、一般CMS的GC耗时 80%都在remark阶段，如果发现remark阶段停顿时间很长，可以尝试添加该参数：-XX:+CMSScavengeBeforeRemark

2、.CMS是基于标记-清除算法的，只会将标记为为存活的对象删除，并不会移动对象整理内存空间，会造成内存碎片，这时候我们需要用到这个参数;-XX:CMSFullGCsBeforeCompaction=n

- CMS GC要决定是否在full GC时做压缩，会依赖几个条件。其中，
- 1. UseCMSCompactAtFullCollection 与 CMSFullGCsBeforeCompaction 是搭配使用的；前者目前默认就是true了，也就是关键在后者上。
- 2. 用户调用了System.gc()，而且DisableExplicitGC没有开启。
- 3. young gen报告接下来如果做增量收集会失败；简单来说也就是young gen预计old gen没有足够空间来容纳下次young GC晋升的对象。
- 上述三种条件的任意一种成立都会让CMS决定这次做full GC时要做压缩。

优化

3、执行CMS GC的过程中，同时业务线程也在运行，当年轻带空间满了，执行ygc时，需要将存活的对象放入到老年代，而此时老年代空间不足，这时CMS还没有机会回收老年带产生的，或者在做Minor GC的时候，新生代救助空间放不下，需要放入老年代，而老年代也放不下而产生concurrent mode failure。

要确定发生concurrent mode failure的原因是因为碎片造成的，还是Eden区有大对象直接晋升老年代造成的，一般有大量的对象晋升老年代容易导致这个错，这种是存在优化空间的，要保证大部分对象尽肯能的再新生代gc掉

优化

4、CMS默认启动的回收线程数目是 (ParallelGCThreads + 3)/4)，这里的ParallelGCThreads是年轻代的并行收集线程数，感觉有点怪怪的；

年轻代的并行收集线程数默认是(ncpus <= 8) ? ncpus : 3 + ((ncpus * 5) / 8)，可以通过-XX:ParallelGCThreads= N 来调整；如果要直接设定CMS回收线程数，可以通过-XX:ParallelCMSThreads=n，注意这个n不能超过cpu线程数，需要注意的是增加gc线程数，就会和应用争抢资源