1.什么是垃圾?

垃圾是指在运行程序中没有任何指针指向的对象，这个对象就是需要被回收的对象。

如果不及时对内存中的垃圾进行清理，那么，这些垃圾对象所占的内存空间会一直保留到应用程序结束，被保留的空间无法被其他对象使用。甚至导致内存溢出。

2.为什么需要垃圾回收?

①如果不进行垃圾回收，内存会被消耗尽

②除了释放没用的对象，垃圾回收也可以清楚内存碎片，已便将内存分配给新的对象

3.垃圾回收算法

3.1 标记阶段

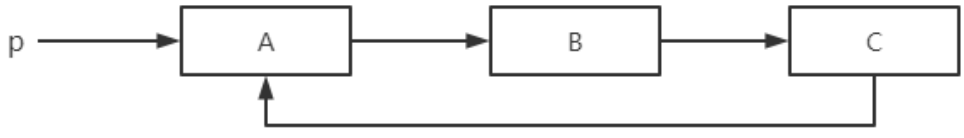
在堆里存放着几乎所有的Java实例，在GC执行垃圾回收之前，首先需要区分出内存中哪些是存活对象，哪些是死亡对象，只有被标记为死亡的对象，GC才会在执行垃圾回收时，释放掉其所占用的内存空间，因此这个过程我们称为垃圾的标记阶段。

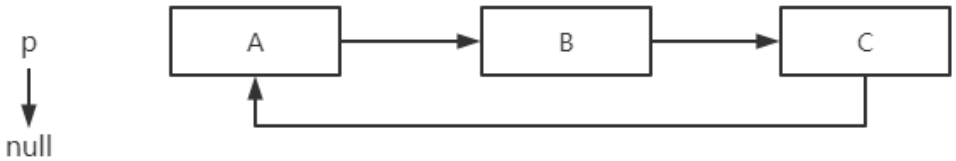
①引用计数算法

对每个对象保存一个整型的引用计数器属性，用于记录对象被引用的情况。对于对象A，只要有任何一个对象引用了A，则A的引用计数器就加1；当引用失效时，引用计数器就减1。只要对象A的引用计数器的值不为0，即表示对象A不可能再被使用，就可以进行回收。

优点：实现简单，判定效率高

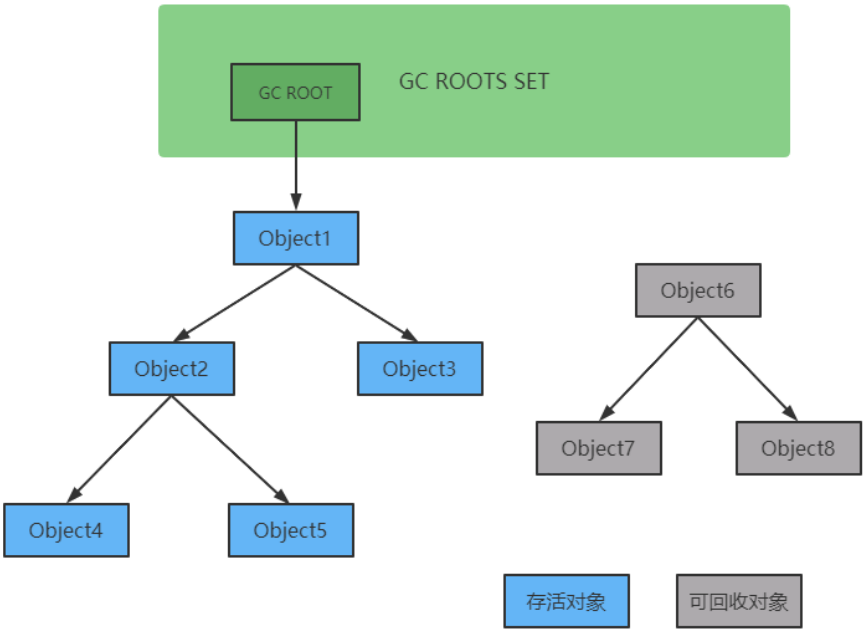
缺点：需要存储额外的字段，增加了存储开销空间；每次都需要更新计数器，增加时间开销；无法处理循环引用





②可达性分析算法

可达性分析算法是以根对象集合（GC roots）为起始点，按照从上至下的方式搜索被根对象集合所连接的对象的目标对象是否可达。内存中的存活对象都会被根对象集合直接或间接连接着，搜索走过的路径成为引用链。



GC ROOTS包括以下几类元素：

①虚拟机栈（栈帧中的局部变量表）中引用的对象：各个线程被调用的方法堆栈中使用的参数、局部变量、临时变量等

②方法区中类静态属性引用的对象：java类的引用类型静态变量

③方法区中常量引用的对象：字符串常量池里的引用

④本地方法栈内JNI（本地方法）引用的对象

⑤Java虚拟机内部的引用：基本数据类型对应的Class对象，一些常驻的异常对象（如NullPointException、OutofMemoryError），还有系统类加载器

⑥所有被同步锁（synchronized关键字）持有的对象

⑦反映java虚拟机内部情况的JMXBean、JVMTI中注册的回调、本地代码缓存

除了以上固定的GC ROOTS集合之外，根据用户所选用的垃圾收集器以及当前回收的内存区域不同，还可以有其他对象临时性加入，共同构成完整GC ROOTS集合。比如：跨代引用的关联区域的对象。

注：若使用可达性分析算法来判断内存是否可回收，那么分析工作必须在一个保障一致性的快照中进行。所以导致GC进行时必须STW（Stop the world）。

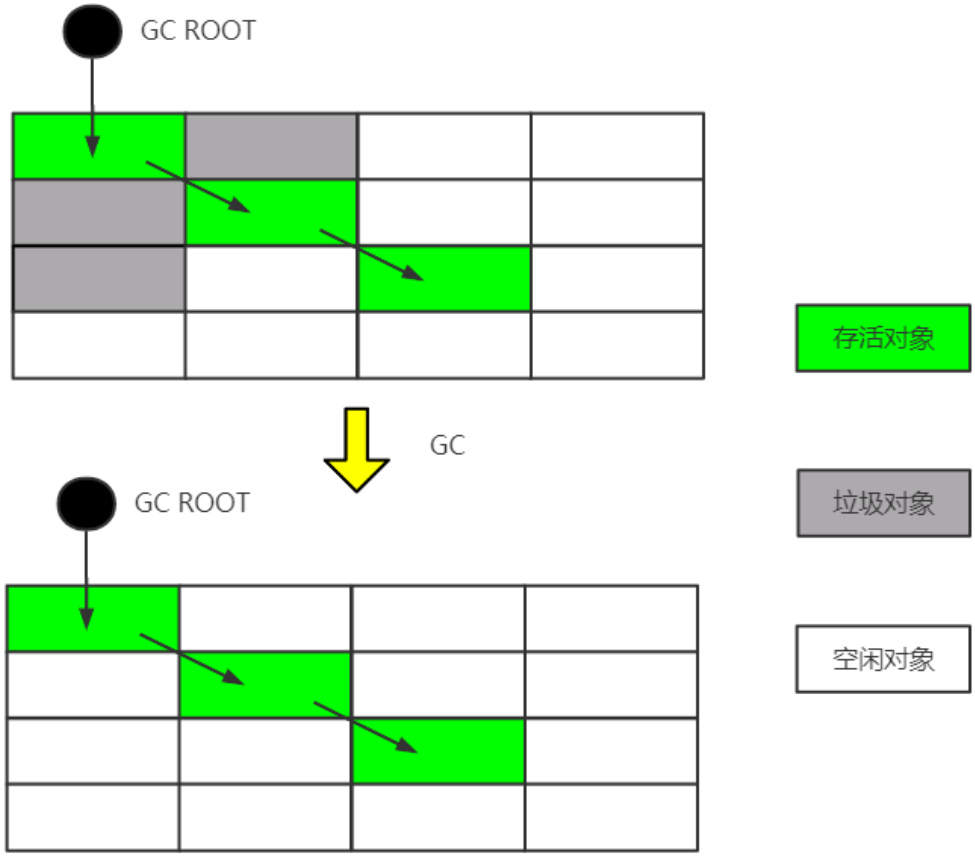
3.2 清除阶段

当成功分出内存中存活对象和死亡对象后，GC接下来的任务就是执行垃圾回收，释放掉无用对象所占用的内存空间，以便有足够的可用内存空间为新对象分配内存。

（1）标记-清除算法（Mark-Sweep）

①标记：从引用根节点开始遍历，标记所有被引用的对象。在对象头中记录为可达对象。

②清除：从堆内存从头到尾进行线性的遍历，如果发现某个对象没有标记为可达对象，则将其回收。



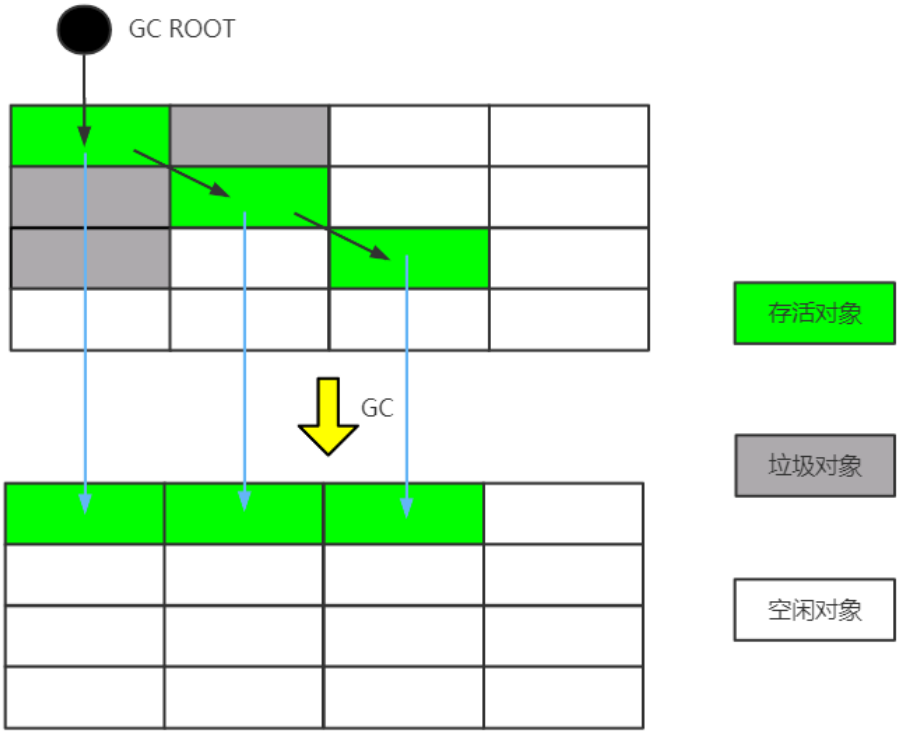
优点：比较容易实现

缺点：效率不高；在GC时需要STW，用户体验不好；会产生内存碎片

注：①如果内存规整，则采用指针碰撞分配内存；如果不规整则虚拟机需要维护一个空闲列表来分配内存 ②清除其实并不是真正的清除，而是把清除的对象地址保存到空闲的地址列表。下次有新对象需要加载时，判断垃圾的位置空间是否够用，如果够就存放

（2）复制算法

将内存空间分为两块，每次只使用其中一块，在垃圾回收时将正在使用的内存中的存活对象复制到未使用的内存块中，之后清除正在使用的内存块中的所有对象，交换两个内存的角色，最后完成垃圾回收。



优点：没有标记和清除过程，实现简单，运行高效；不会存在内存碎片化问题

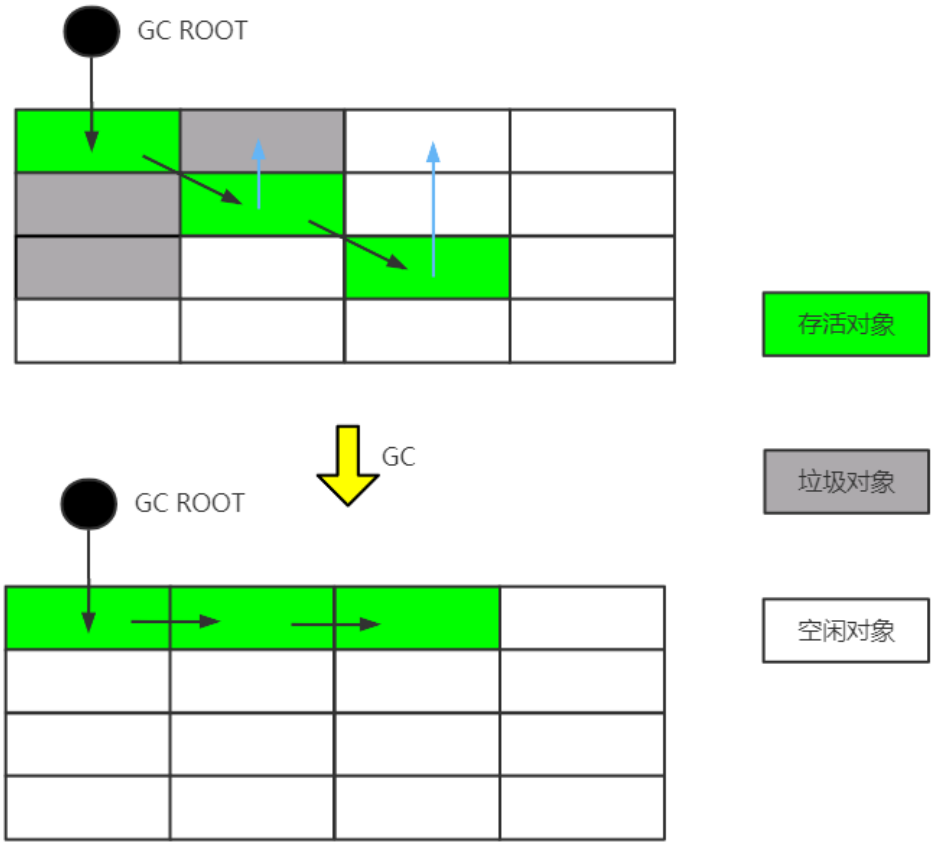
缺点：浪费了一半空间

注：假如存活对象过多，则复制算法效率会很差

(3) 标记-整理算法

①标记：从引用根节点开始遍历，标记所有被引用的对象。在对象头中记录为可达对象。

②整理：将所有的存活对象整理到内存的一端 按顺序排放。



优点：没有内存碎片；不浪费空间

缺点：效率上略低于复制算法；需要修改引用的地址；移动过程需要STW

（4）分代收集算法

由于不同对象的生命周期是不一样的。因此，因此不同生命周期的对象可以采取不同的收集方式，以便提高回收效率。**一般是把java堆分为新生代和老年代，这样根据各个年代的特点使用不同的回收算法，以提高回收的效率。目前几乎所有的GC都是采用分代收集算法执行垃圾回收的。**

在HotSpot中，基于分代的理念，GC所使用的内存回收算法必须结合年轻代和老年代各自的特点

①年轻代：内存较小；对象的生命周期短、存活率低，回收比较频繁。**这种情况复制算法回收整理，速度最快。**

②老年代：区域较大；对象的生命周期长、存活率高，回收不太频繁。**一般由标记-清除或标记清除与标记整理混合使用**

（5）增量收集算法

上述的算法中，在垃圾回收过程中，应用软件将处于一种STW的状态，在STW状态下，应用程序所有的线程都会挂起，暂停一切正常的工作，等待垃圾回收的完成。如果垃圾回收时间过长，将严重用户的体验。为了解决这种问题，对实时垃圾回收算法的研究产生了增量收集算法（Incremental Collecting）

基本思想，增量收集算法的基础仍然是传统的标记清除和复制算法。增量收集算法通过对线程间冲突的妥善处理，允许垃圾收集线程以分阶段的方式完成标记、清理或复制工作。

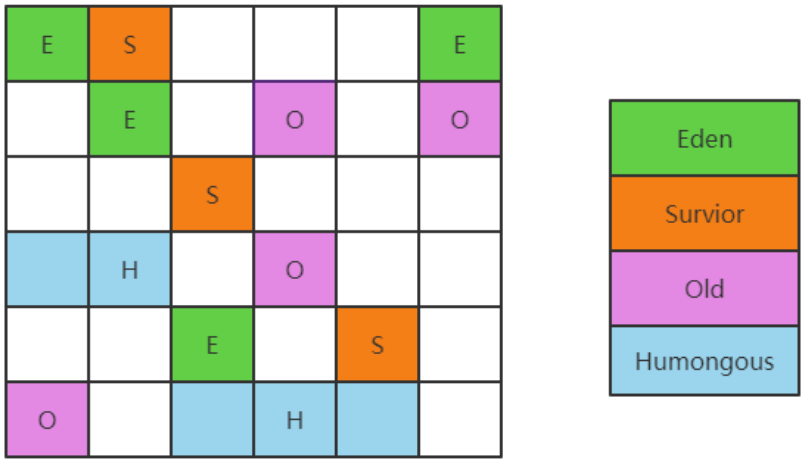
优点：提高用户体验

缺点：由于线程切换和上下文消耗，会使得垃圾回收的总体成本上升，造成系统的**吞吐量**下降。

（6）分区算法

一般来说，在相同条件下，堆空间越大，一次GC时所需要的时间就越长，有关GC产生的停顿也越长。为了更好的控制GC产生的停顿时间，将一块打的内存区域分割成多个小块，根据目标的停顿时间，每次合理地回收若干个小区间，而不是整个堆空间，从而减少一次GC的停顿时间。

每个小区间都独立使用，独立回收。这种算法的好处是可以控制一次回收多少个小区间。



4.垃圾回收器

4.1 垃圾回收器的发展史

①1999年，JDK1.3，发布Serial GC，它是第一款垃圾回收器。ParNew是垃圾收集器的多线程版本。

②2002年，JDK1.4，发布Parallel GC和Concurrent Mark Sweep GC。JDK6后，HotSpot默认Parallel GC为默认垃圾回收器。

③2012年，JDK1.7，发布G1。JDK9后，默认垃圾收集器为G1。

④2018年3月，JDK11，引入Epsilon和ZGC（低延迟）

⑤2019年3月，JDK12，引入Shenandoah GC（低延迟）

⑥2020年3月，JDK14，删除CMS垃圾回收器；同年9月，JDK15，ZGC和Shenandoah由实验变为生产

⑦2021年3月，JDK16，ZGC支持并发栈处理，解决了最后一个重大瓶颈，把 ZGC 中的线程栈处理从安全点移到了并发阶段

⑧即将发布，JDK17，JEP 404，Shenandoah支持分代

4.2 经典垃圾回收器

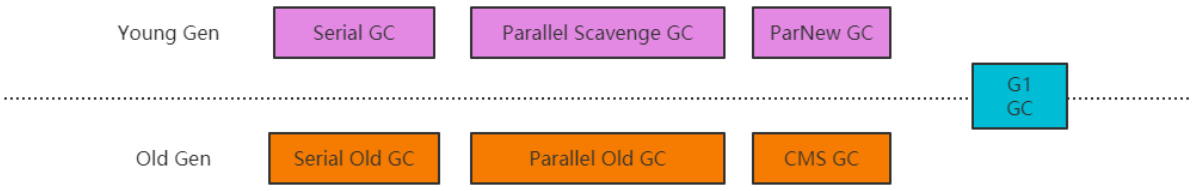
（1）按回收方式划分

串行回收：Serial GC、Serial Old GC

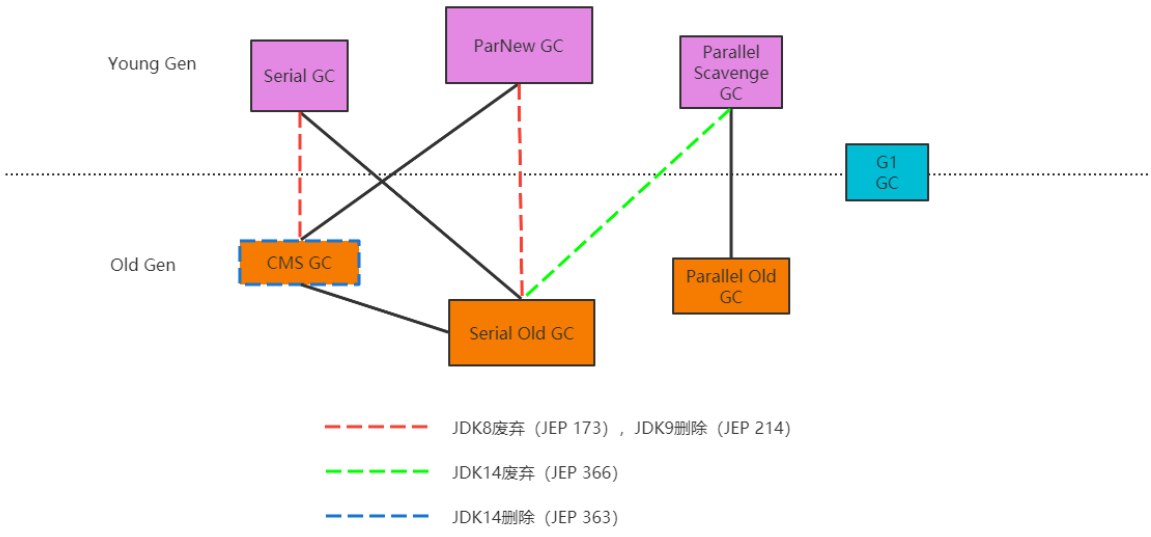
并行回收：ParNew、Parallel Scavenge GC、Parallel Old GC

并发回收：CMS、G1

（2）按分代划分



（3）组合关系



（4）查看默认的垃圾回收器：-XX：+PrintCommandLineFlags 查看命令行相关参数（包括了垃圾回收器）

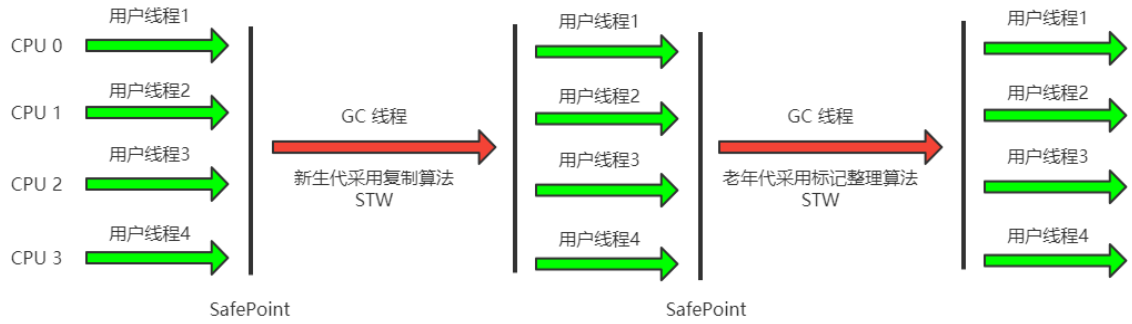
jinfo -flag 相关参数 进程ID

（5）Serial GC

Serial回收器是最早、历史最悠久的垃圾回收器，是JDK1.3之前唯一的选择。

Serial回收器是HotSpot中Client模式下的默认新生代垃圾收集器。

Serial回收器采用的算法为**复制算法，串行回收。**配合使用的是Serial Old回收器，使用**标记整理算法**

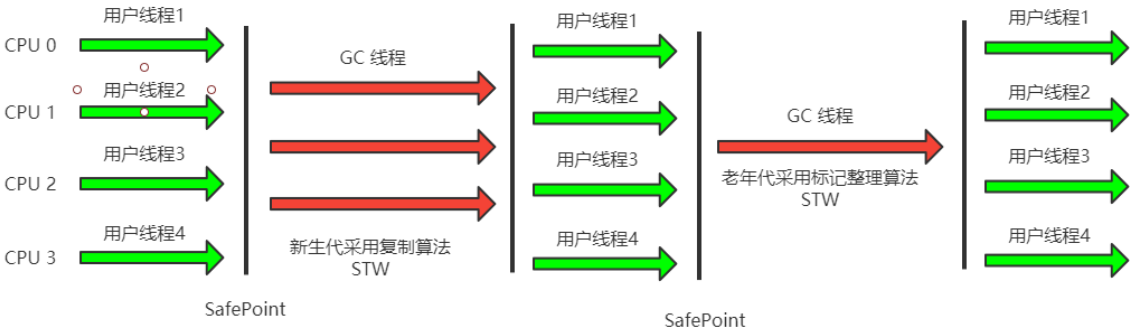


（6）ParNew GC

ParNew除了采用并行回收的方式外，几乎和Serial没任何区别

ParNew是很多JVM运行在Server模式下新生代的默认垃圾回收器

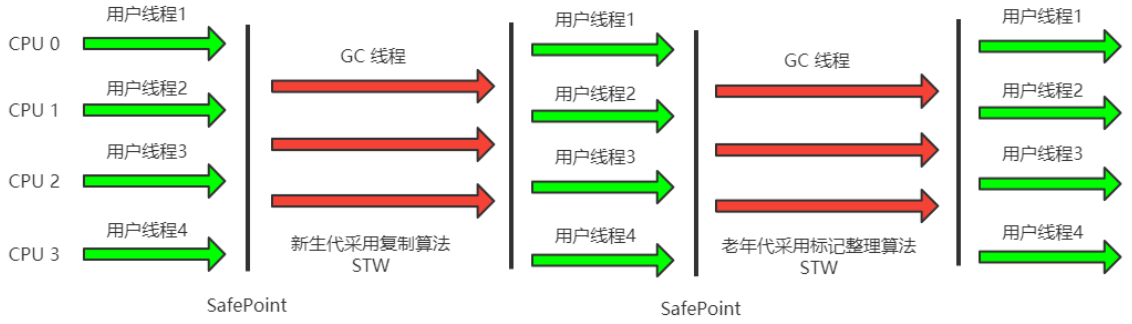
ParNew回收器采用的算法为**复制算法**



（7）Parallel GC

吞吐量优先，高吞吐量适合不需要太多交互的任务

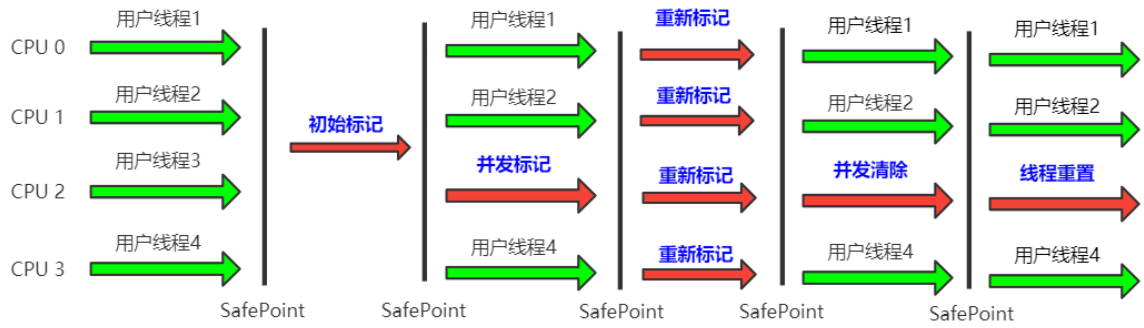
Parallel Scavenge回收器采用的算法为**复制算法**，配合使用的是Parallel Old回收器，使用**标记整理算法，并行回收**



（8）CMS

这款回收器是HotSpot虚拟机中第一款真正意义上的并发回收器，它是第一次实现了让垃圾回收线程与用户线程同时工作

CMS垃圾回收算法采用**标记清除算法**，会STW，但是CMS不可以和Parallel Scavenge回收器配合使用，只能选择Serial和ParNew结合使用



工作流程：

①初始标记：仅仅标出GC Roots能直接关联的对象，需要STW，但速度非常快。

②并发标记：从GC Roots的直接关联对象开始遍历整个对象图的过程，这个过程耗时较长但不需要停顿用户线程。

③重新标记：由于在并发标记阶段中，程序的工作线程会和垃圾收集线程同时运行，因此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。

④并发清除：此阶段清理掉标记阶段判断已经死亡的对象，直接释放内存空间。

由于最耗时的并发标记和并发清除阶段并不需要STW，所以整体回收是低延迟的。因此，CMS不像其他回收器那样等到老年代几乎完全填满才进行回收，而是当堆内存使用率达到某一阈值（JDK5 68%；JDK6 92%）时，便开始回收，以确保应用程序在CMS工作过程中依然有足够的空间支持应用程序运行。如果无法满足，则会出现Concurrent Mode Failure失败，这时虚拟机就会启动后备方案，临时启用Serial Old回收器来重新进行老年代的垃圾回收，这样停顿时间将会很长。

优点：并发回收，低延迟

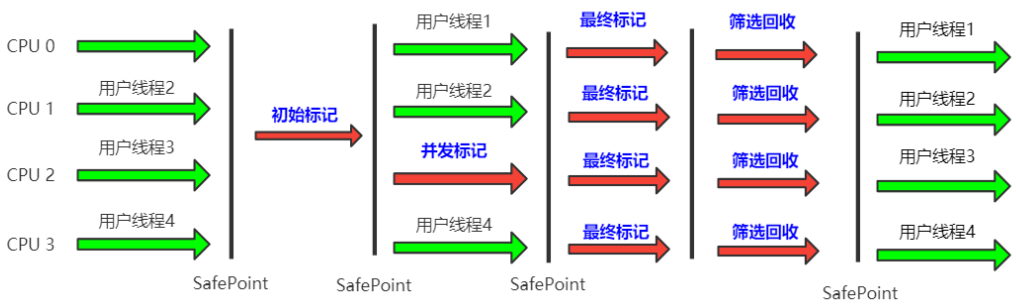
缺点：产生内存碎片；对CPU资源非常敏感；无法处理浮动垃圾，浮动垃圾只能下次回收

（8）G1

G1回收器是垃圾回收器技术发展历史上的里程碑式的成果，他开创了回收器面向局部收集的设计思路和基于Region的内存布局形式。

G1将内存划分为一个个的Region。内存回收是以Region作为基本单位。Region之间是**复制算法**，但整体上可看作是**标记压缩算法**，两种都可以避免内存碎片。

它将Java堆分为2048个大小相同的独立region，每个大小根据实际堆内存决定，整体被控制在1-32MB之间，且为2的N次幂，可以通过-XX:G1HeapRegionSize设定。



工作流程：

①初始标记：标记一下GC ROOTS能直接关联到的对象，修改TAMS指针（G1为每个Region设计了两个TAMS指针）。

②并发标记：遍历整个对象图，找出要回收的对象。对象图扫描完后，还会重新处理STAB（原始快照）记录下的在并发时有引用变动的对象。

③最终标记：用于处理并发阶段后仍然留下的少量STAB记录。

④筛选回收：对Region回收价值排序，根据用户期望的停顿时间制定回收计划，将任意个Region组成回收集并行回收。

优点：

与CMS相比，当Java堆非常大时，G1的优势会非常明显。

由于分区的原因，G1可以只选择部分区域进行回收，这样缩小了回收的范围，因此对于全局停顿情况的发生也能得到较好的控制。

G1会跟踪每个Region里面的垃圾堆积的价值大小，在后台维护一个优先列表，每次根据允许回收时间，优先回收价值最大的Region，保证了G1回收器在有限的时间内可以尽可能高的回收效率。

缺点：

在用户程序中，G1为了垃圾回收产生的内存占用和程序运行时的额外执行负载都比CMS高。经验上来讲，小内存上CMS优于G1，大内存G1优于CMS，平衡点6-8G。

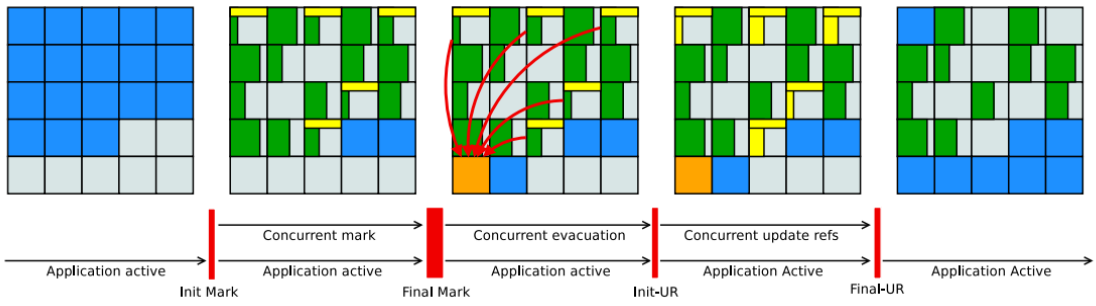
4.3 低延迟垃圾回收器

（1）Shenandoah GC

目标是实现一种能在任何堆大小都可以把垃圾回收停顿时间降低在10ms以下，除了并发标记外还要并发清理和并发整理。

Shenandoah摒弃了在G1中耗费大量内存和计算资源去维护的记忆集，改为连接矩阵的全局数据结构来处理跨Region的引用关系，降低了处理跨代指针时的记忆集维护消耗，也降低了伪共享问题发生的概率。

图片来源：https://shipilev.net/talks/devoxx-Nov2017-shenandoah.pdf



黄色：被选入回收集的Region。绿色：还存活的对象。蓝色：可以分配对象的Region

工作流程：

①初始标记：标记与GC ROOTS直接关联的对象

②并发标记：遍历对象图，标记全部可达对象

③最终标记：处理STAB，计算回收价值最高的Region，组成回收集

④并发清理：清理整个Region中一个存活对象都没有的对象

⑤并发回收：先把回收集里边的存活对象复制一个到没有被使用的Region

⑥初始引用更新：把堆中所有指向旧对象的地址引用修正到复制后的新地址

⑦并发引用更新：按照内存物理地址的顺序，线性的搜索引用类型，把旧值改为新值

⑧最终引用更新：修正与GC ROOTS中的引用

⑨并发清理：整个回收集中已无存活对象，直接回收

（2）ZGC

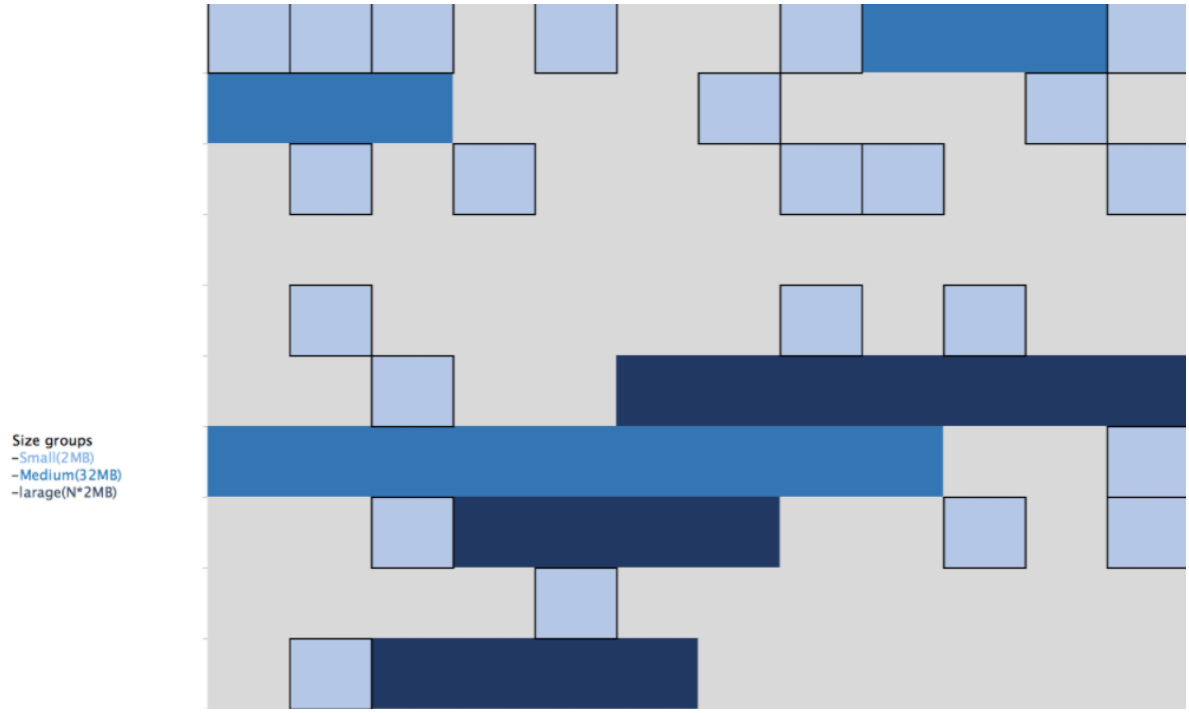
ZGC是基于Region内存布局的，暂时不设分代的，使用了读屏障、染色指针和内存多重映射等技术来实现可并发的标记整理算法的，以低延迟为首要目标的一款垃圾回收器。

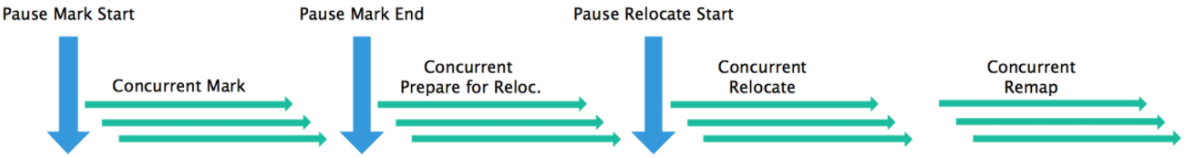
ZGC的Region不再是固定大小，具有动态创建和销毁的特性。在x64平台下，ZGC的Region分为小中大三个容量：

小型Region：固定2MB，存放小于256KB的小对象

中型Region：固定32MB，存放大于等于256KB但小于4MB的对象

大型Region：容量不固定，但必须为2M的整数倍，用于存放4MB或以上的大对象





工作流程：

①并发标记：遍历对象图做可达性分析，标记阶段不是在对象上进行而是在指针上进行的。

②并发预备重分配：根据特定的条件统计出本次收集过程需要清理哪些Region，将这些Region组成重分配集。

③并发重分配：把重分配集中的存活对象复制到新Region上，并为重分配集中的每个Region维护一个转发表，记录从旧对象到新对象的转向关系。

④**并发重映射**：修正整个堆中指向重分配集中旧对象的所有引用，合并到了下一次垃圾回收的并发标记阶段。