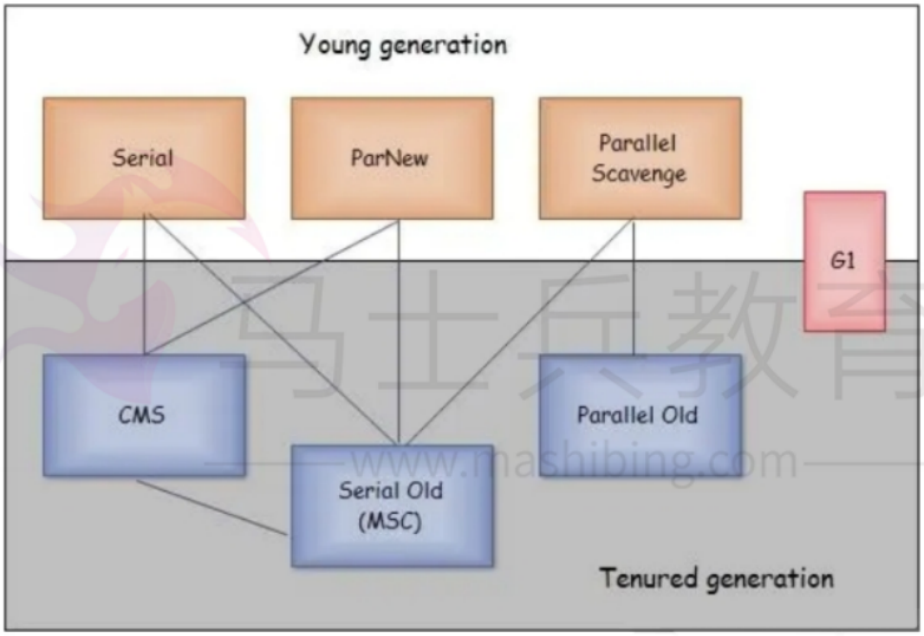
如果说垃圾收集算法是内存回收的方法论，那么垃圾收集器就是内存回收的具体 实现。下图展示了7种作用于不同分代的收集器，其中

回收新生代的收集器包括 Serial、PraNew、Parallel Scavenge；

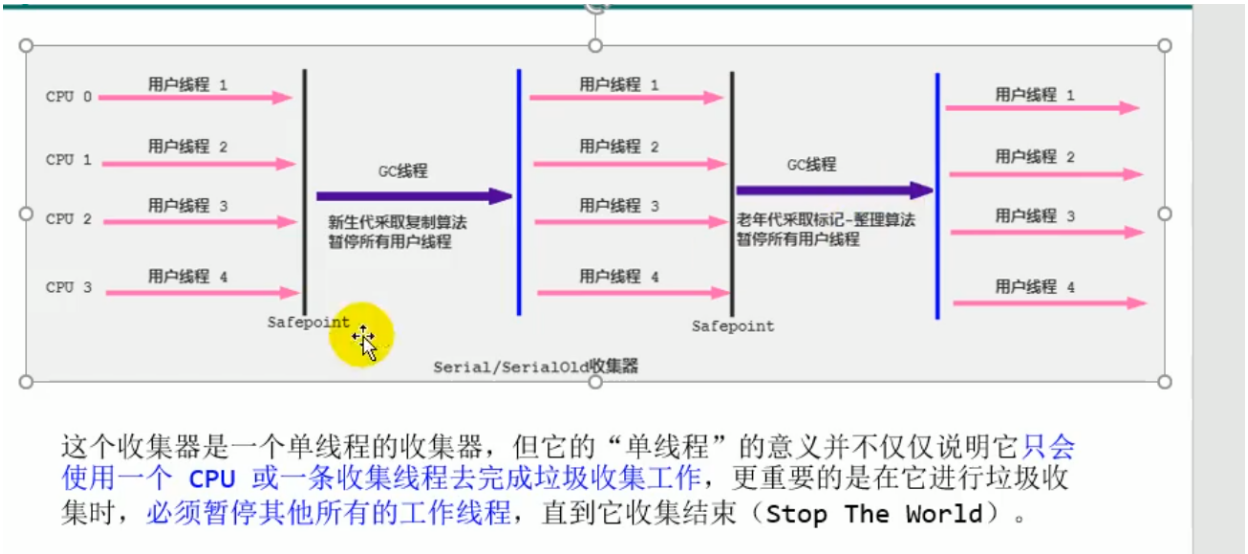
回收老年代的收集器包括 Serial Old、Parallel Old、CMS，

还有用于**回收整个Java堆的G1收集器。**

不同收集器之间的连线表示它们可以搭配使用。

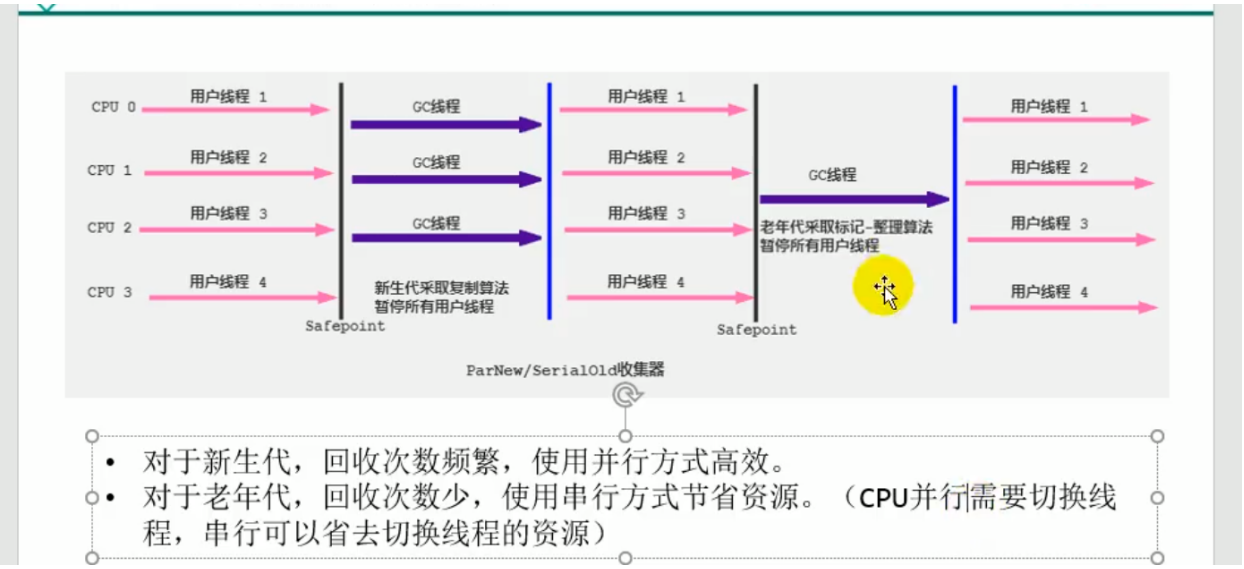


- Serial收集器（**复制算法**): 新生代**单线程**收集器，串行回收、STW机制，标记和清理都是单线程，优点 是简单高效；
- Serial Old收集器 (**标记-压缩算法**): 老年代单线程收集器，Serial收集器的老年代版本，串行回收、STW机制；



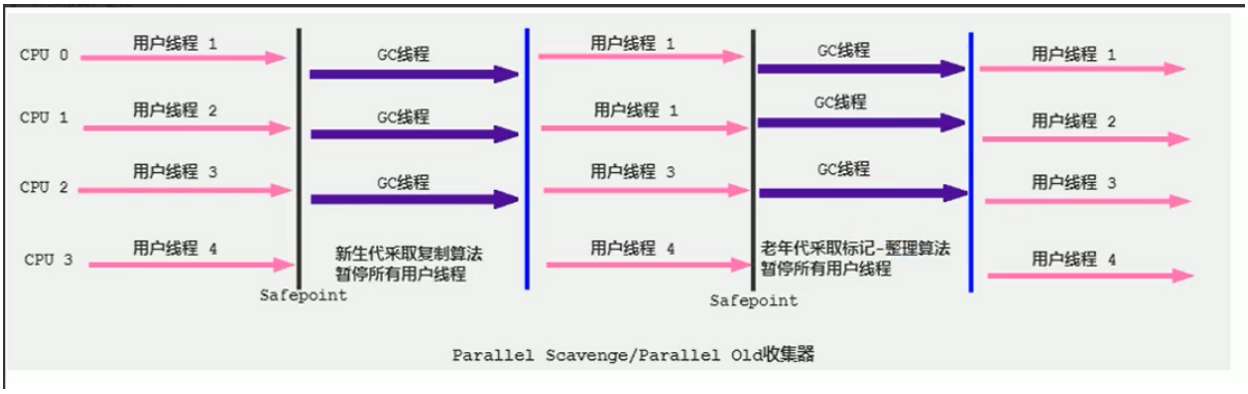
这种垃圾收集器大家了解，现在已经不用串行的了。而且在限定单核cpu才可以用。现在都不是单核的了。

- ParNew收集器 (**复制算法**): 新生代**并行**收集器，STW机制，实际上是Serial收集器的**多线程**版本，在多核CPU环境下有着比Serial更好的表现；



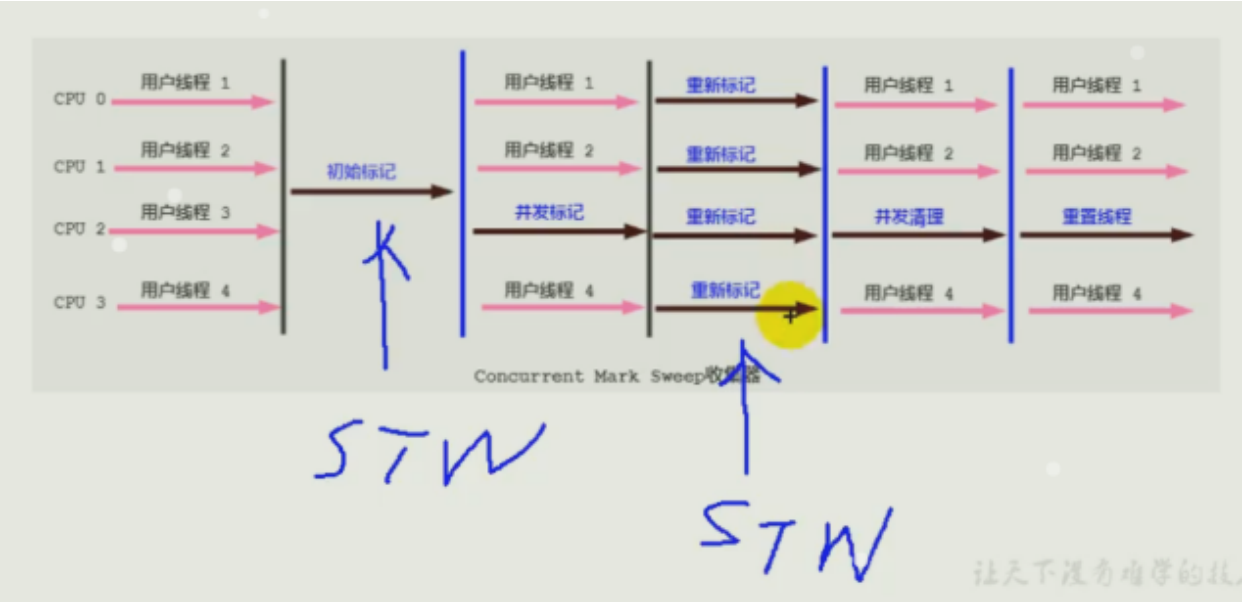
- Parallel Scavenge收集器 (**复制算法**): 新生代**并行收集**器，STW机制**追求高吞吐量**，高效利用 CPU。吞吐量= 用户线程时间/(用户线程时间+GC线程时间)，高吞吐量可以高 效率的利用CPU时间，尽快完成程序的运算任务，适合后台应用等对交互相应要求不 高的场景；

- Parallel Old收集器 (**标记-压缩算法**)： 老年代**并行收集**器，**吞吐量优先**，STW机制， Parallel Scavenge收集器的老年代版本；



- CMS(Concurrent Mark Sweep)收集器（**标记-清除算法**）： 老年代**并行收集**器，以获取最短回收停顿时间为目标的收集器，具有**高并发**、**低停顿**的特点，追求最短GC回收停顿时间。并且也会STW，由于使用该算法，避免不了产生内存碎片，无法使用指针碰撞，只能使用空闲列表分配内存

CMS工作原理：

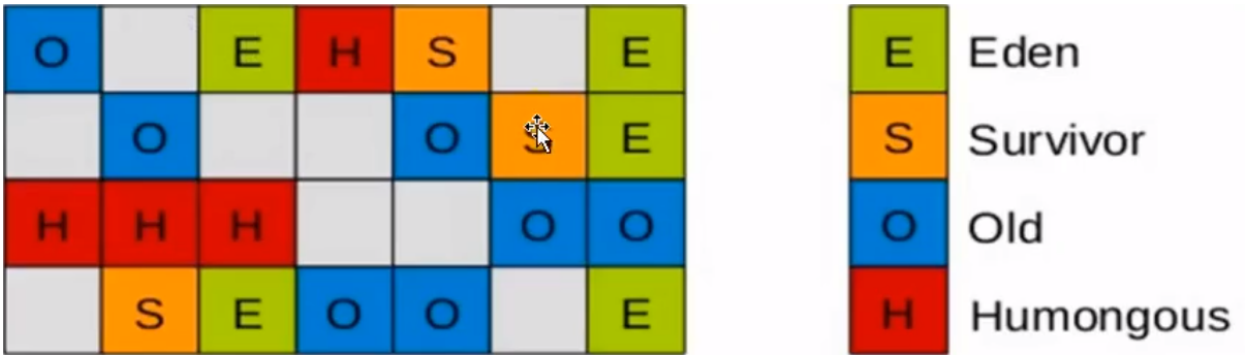


CMS 使用的是标记-清除的算法实现的，所以在 gc的时候回产生大量的内存碎 片，当剩余内存不能满足程序运行要求时，系统将会出现Concurrent Mode Failure，临时 CMS 会采用 Serial Old 回收器进行垃圾清除，此时的性能将会降低。

- G1(Garbage First)收集器 (**标记-整理(压缩)算法**)： Java堆**并行收集**器，G1收集器是 JDK1.7提供的一个新收集器，G1收集器基于“标记-整理”算法实现，也就是说不会产生内存碎片。此外，G1收集器不同于之前的收集器的一个重要特点是：G1**回收的范围是整个Java堆**(包括新生代，老年代)，而前六种收集器回收的范围仅限于新生代 或 老年代。G1把堆内存划分成一个个不相关的区域Region，这些区域包含了逻辑上的年轻代和老年代，内存回收以Region为单位的。Region是复制算法，但整体算是“标记-压缩算法”，这两种算法都可以避免碎片化

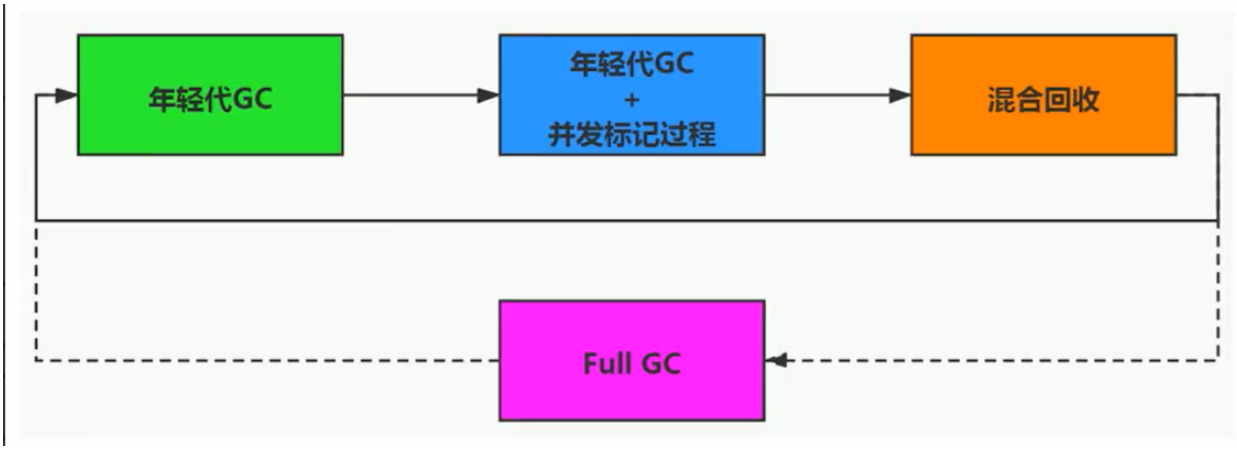
为什么名字叫做Garbage First（G1）呢？

- 因为G1是一个并行回收器，它把堆内存分割为很多不相关的区域（Region）（物理上不连续的）。使用不同的Region来表示Eden、幸存者0区，幸存者1区，老年代等。
- G1 GC有计划地避免在整个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（回收所获得的空间大小以及回收所需时间的经验值），在后台维护一个优先列表，每次根据允许的收集时间，优先回收价值最大的Region。
- 由于这种方式的侧重点在于回收垃圾最大量的区间（Region），所以我们给G1一个名字：垃圾优先（Garbage First）。



G1 GC垃圾回收过程主要包括如下三个环节：

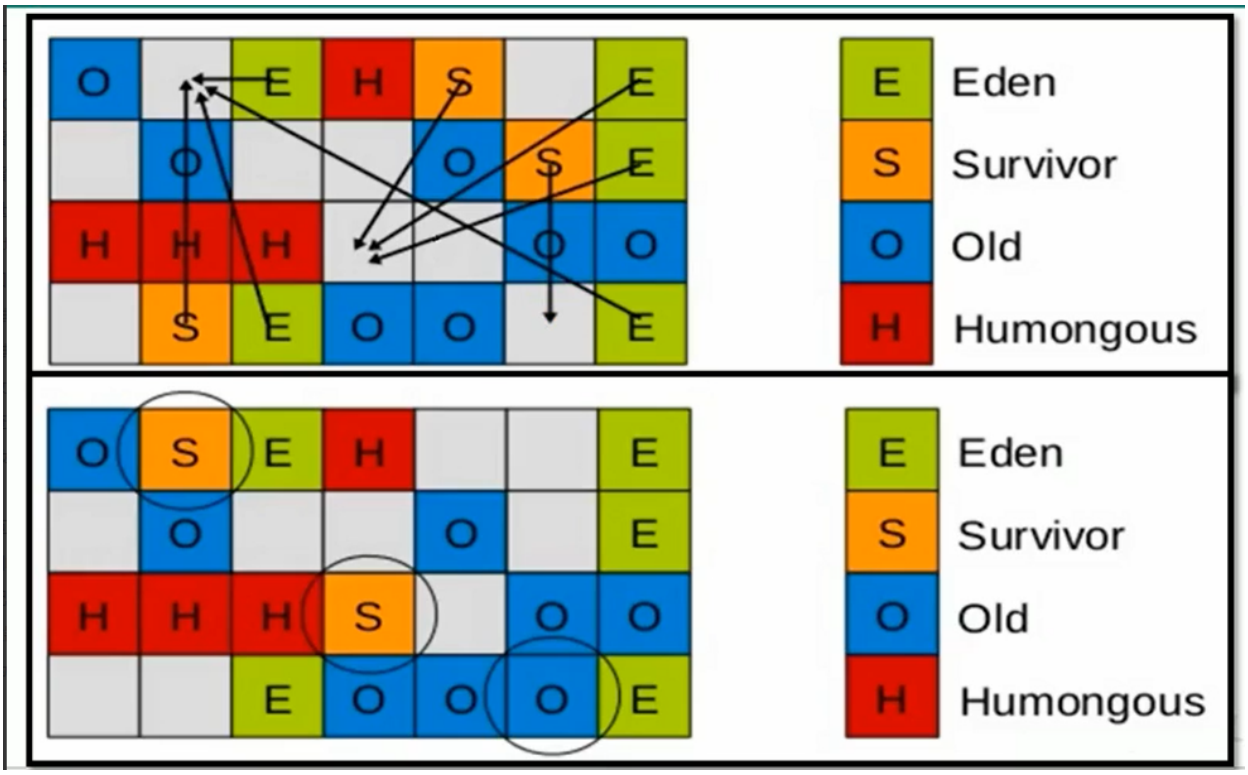
- 年轻代GC（Young GC）
- 老年代并发标记过程（Concurrent Marking）
- 混合回收（Mixed GC）



顺时针，young gc->young gc+concurrent mark->Mixed GC顺序，进行垃圾回收。

应用程序分配内存，当年轻代的Eden区用尽时开始年轻代回收过程；G1的年轻代收集阶段是一个并行的独占式收集器。在年轻代回收期，G1 GC暂停所有应用程序线程，启动多线程执行年轻代回收。然后从年轻代区间移动存活对象到Survivor区间或者老年区间，也有可能是两个区间都会涉及。

当堆内存使用达到一定值（默认45%）时，开始老年代并发标记过程。
标记完成马上开始混合回收过程。对于一个混合回收期，G1 GC从老年区间移动存活对象到空闲区间，这些空闲区间也就成为了老年代的一部分。和年轻代不同，老年代的G1回收器和其他GC不同，G1的老年代回收器不需要整个老年代被回收，一次只需要扫描/回收一小部分老年代的Region就可以了。同时，这个老年代Region是和年轻代一起被回收的。



hotspot有这么多的GC，Serial GC 、Parallel GC 、CMS有什么不同呢？

- 如果想要最小化的内存和并行开销，请选择：Serial GC + Serial Old GC
- 如果你想要最大吞吐量，请选择：Parallel GC +Parallel Old GC
- 如果你想要低延迟延迟，请选择：ParNew GC + CMS GC

分代垃圾回收器是怎么工作的？

新生代使用的是复制算法，新生代里有 3 个分区：Eden、To Survivor、From Survivor，它们的默认占比是 8:1:1，它的执行流程如下：

- 把 Eden + From Survivor 存活的对象放入 To Survivor 区；
- 清空 Eden 和 From Survivor 分区；
- From Survivor 和 To Survivor 分区交换，From Survivor 变 To Survivor，To Survivor 变 From Survivor。

每次在 From Survivor 到 To Survivor 移动时都存活的对象，年龄就 +1，当年 龄到达 15（默认配置是15）时，升级为老生代。大对象也会直接进入老生代。

老生代当空间占用到达某个值之后就会触发全局垃圾收回，一般使用标记整理的 执行算法。以上这些循环往复就构成了整个分代垃圾回收的整体执行流程。

垃圾收集器	分类	作用位置	使用算法	特点	适用场景
Serial	串行运行	作用于新生代	复制算法	响应速度优先	适用于单CPU环境下的client模式
ParNew	并行运行	作用于新生代	复制算法	响应速度优先	多CPU环境Server模式下与CMS配合使用
Parallel	并行运行	作用于新生代	复制算法	吞吐量优先	适用于后台运算而不需要太多交互的场景
Serial Old	串行运行	作用于老年代	标记-压缩算法	响应速度优先	适用于单CPU环境下的Client模式
Parallel Old	并行运行	作用于老年代	标记-压缩算法	吞吐量优先	适用于后台运算而不需要太多交互的场景
CMS	并发运行	作用于老年代	标记-清除算法	响应速度优先	适用于互联网或B/S业务
G1	并发、并行运行	作用于新生代、老年代	标记-压缩算法、复制算法	响应速度优先	面向服务端应用