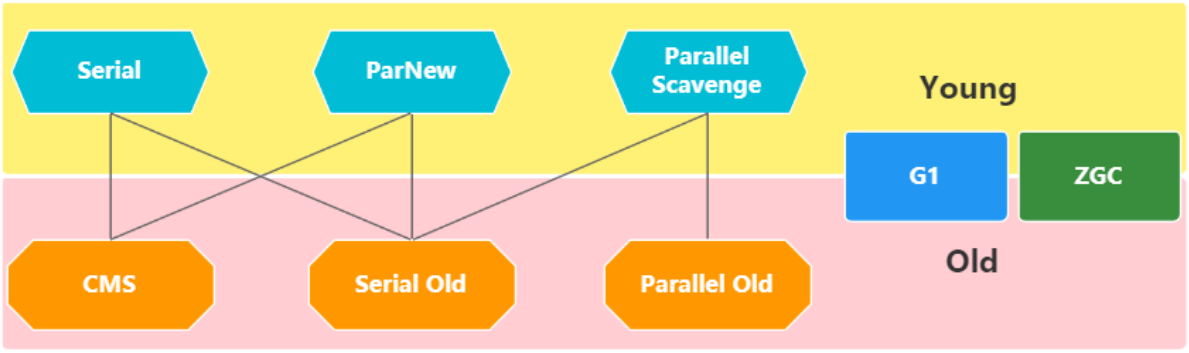
常见GC

常用的垃圾回收器



前面几个不光逻辑上区分年轻代和老年的，物理上也区分年轻的和老年的

G1 只是在逻辑上 区分年轻的和老年的 jdk8 已经比较成熟了， jdk 11后默认是G1

逻辑上管理的一块一块的区域可以是 eden、可以是 survivor、也可以是old

但在物理上这些一块一块的区域可以角色转换，如：回收前世eden区，回收后下一次可能会变成old区

ZGC 只有一代

ZGC 和 Shenandoah 逻辑上物理上都不分（Single Generation ） jdk 11 才引入

只有整体内存，以后GC就越来越容易

Epsilon debug 用的 不去管它 jdk 11 才有

这儿的Serial 指的是单线程，Parallel指的是多线程

常见垃圾回收期的组合

- Serial + Serial Old
- Parallel Scavenge + Parallel Old
- PerNew + CMS

注：上图中有连线关系的GC都可以组合

垃圾回收期的历史：

JDK诞生的时候，Serial追随，就是说JDK诞生之后，第一个垃圾回收器就是Serial，提高效率，诞生了PS（多线程），为了配合CMS，诞生了PerNew 。但是CMS是1.4版本的后期引入，CMS是里程碑式的，开启了并发回收的机制。但是CMS毛病较多，因此目前没有任何一个JDK版本默认是CMS。

并发垃圾回收是因为无法忍受STW

常见组合详解

▼ **Serial组合（Serial + Serial Old）=======**

Serial（现在用的极少）

工作过程：当Serial线程干活的时候，所有的工作线程全部停止

STW：

Stop the world，这个过程发生时间点就是GC开始干活的那个时间点，在安全点上所有线程停止

和面试官聊天就说STW，不要说stop the world

Safe point：

安全点，STW的时候，不能立即停止，如果有执行到一半的线程，等待执行完后再GC

说明：

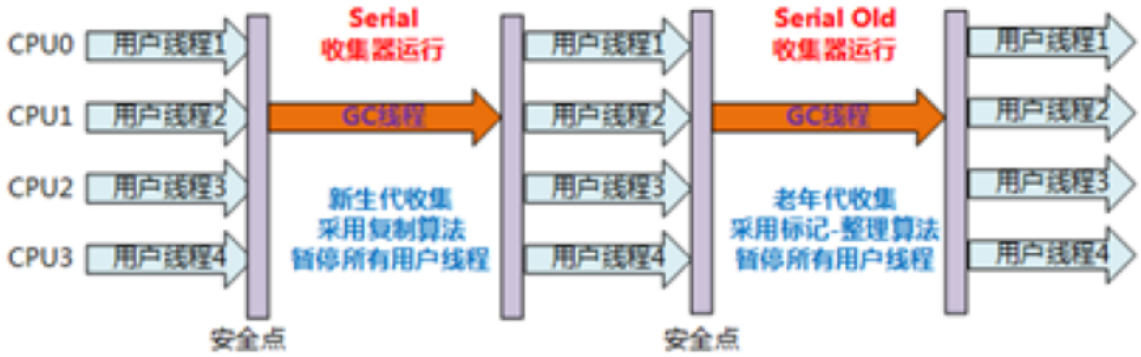
好多个线程在运行过程中一直产生垃圾，当产生的垃圾堆满的时候，所有的工作线程都停止运行（即阻塞），GC开始干活，等清理完成之后工作线程继续执行，循环往复进行该操作

使用场景：停顿时间（STW的时间）较短的情况，内存小的情况

这种情况是内存很小的情况，jdk 刚开始的时候内存都不大，那个时候一个线程在这个里面做清理，工作现场可以接受，因为清理的内存本身也不大，时间上可以等待。现在这种机器，内存超级大，如果单线程要清理这么大的内存，STW停顿时间太长，工作现场要急死，等不了。

Serial Old (现在基本不用)

单线程在Old 区的情况，用的是 mark-sweep-compact 算法，也是单线程



▼ **Parallel组合（Parallel Scavenge+Parallel Old）=======（1.8默认）**

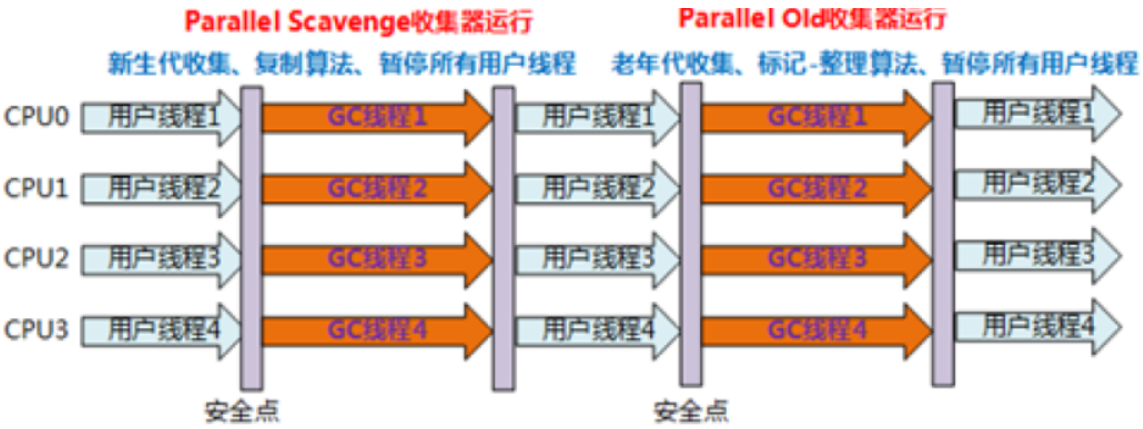
Paralle Scavenge

假如在线之前没有对JVM做任何调优，默认就是 PS + PO

和Serial 不同的是，Parallel Scavenge 是多线程清理垃圾，其它都不变

Paralle Old

和 perial old 一样，多线程清理Old区



▼ **PerNew组合（PerNew + CMS）======**

ParNew

PerNew：Parallel New的意思，GC的演化过程本身就不连贯，所以命名也不连贯

还是STW：

现在有没有不会产生STW的GC—目前没有，只是时间问题

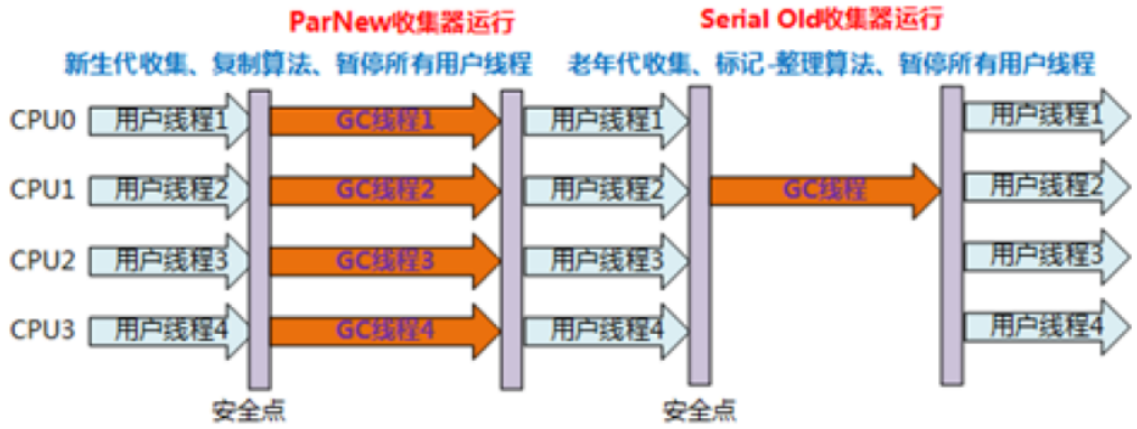
不过ZGC号称STW目标停顿时间是10ms以内，可是实际测试时间是2ms—很牛叉

是Parallel Scavenge的变种，和Parallel Scavenge 的区别就是做了一些增强，以便和CMS配合使用

增强是：CMS在某一个特定阶段的时候和PerNew可以同时运行

ParNew 是和CMS配合使用的，是Parallel Scavenge的一个变种

延伸阅读：
https://docs.oracle.com/en/java/javase/13/gctuning/ergonomics.html#GUID-3D0BB91E-9BFF-4EBB-B523-14493A860E73

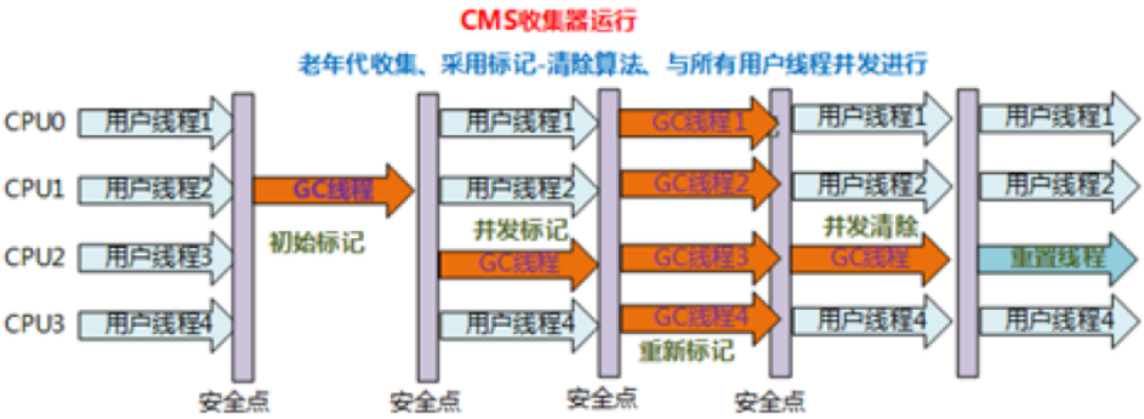


CMS（非常重要）—-Concurrent mark sweep

高响应，低停顿

CMS是一个里程碑式的GC，其它的GC是在工作的时候，工作线程必须停止，可以和工作现场并发执行。

也就是说工作线程在工作产生垃圾的时候（工作线程继续工作），GC就开始工作了



并发垃圾回收是因为无法忍受STW

在内存比较小的时候，Parallel Scavenge 这个是可以清理的，因为内存比较小，清理起来也比较慢，现在的服务器内存都是非常大的，原来内存可以比作一个房间，现在服务器内存就可以比作一个天安门广场。所以这么大的一个区域，那怕是好多个线程同时清理都要好长时间。长的时候不可想象几天或者几个小时

CMS 论文发表到第一版产生经理了很长世间

下面描述的是

CMS 触发条件————老年代分配不下的情况

CMS是一个老年的的垃圾回收期，和其它老年的的垃圾回收期的触发条件没差别

Old 区内存不够了后，会出发FGC执行，默认执行器是 PS + PO

CMS 的几个常见阶段——标记的不是垃圾，没被标记的最后被清理

1、初始标记（STW）：直接找到最根上的对象，并标记

由于只标记了一些开始的对象，所以时间并不长

2、并发标记：一边产生垃圾，一遍标记。

这块是最浪费时间的（CMS早版本），所以后期将这块最浪费时间的和应用程序同时运行，这样应用程序就不会停止，只是客户会觉得访问慢了一点而已。

3、重新标记（STW）：并发标记过程中工作现场产生的新垃圾。需要重新标记。

这种情况是多数的对象垃圾在并发标记的过程中已经标记完了，在并发标记的过程中会产生的新的垃圾。也有原来是垃圾的对象被别的线程又捡起来变

成不是垃圾了，需要重新标记。这种情况下垃圾量也很少，所以用的也是STW（因为不停的话就会一直产生新垃圾，为了标记清楚所以要停掉工作线程）

4、并发清理：这个过程中也会有问题，并发清理的时候也会其它工作线程也会产生新的垃圾，这种垃圾叫浮动垃圾，这种浮动垃圾就要等待下一次CMS做清理的时候再次清理了。

CMS 缺点

上图中由于CMS 还和 Serial Old 是有关联的，这个的意思是CMS一旦不行了之后，SerialOld就要上场了。单线程扫天安门。。。。。

1. **Memory Fragmentation　内存碎片化**

   因为CMS叫 mark sweep，天然问题就是碎片化，CMS在起初设计面向的内存不是很大，现在很多人用CMS处理的内存都相当大（32G内存），应付大于32G内存的时候一般会出问题，假如说新的对象已经不能向Old区装了（年轻代对象到Old 区找不到空间了），这个时候CMS就把Serial Old请出来，用一个线程在Old区做标记压缩（工作线程停止运行）。

   > -XX:+UseCMSCompactAtFullCollection
   > -XX:CMSFullGCsBeforeCompaction 默认为0 指的是经过多少次FGC才进行压缩

2. **Floating Garbage 浮动垃圾过大**

   > Concurrent Mode Failure　问题和 PromotionFailed 问题
   > 产生：if the concurrent collector is unable to finish reclaiming the unreachable objects before the tenured generation fills up, or if an allocation cannot be satisfiedwith the available free space blocks in the tenured generation, then theapplication is paused and the collection is completed with all the applicationthreads stopped

CMS 设计的初衷是让停顿时间变短（重新标记），但是当发生浮动垃圾太多的时候，新对象在Old区找不到空间的时候会出现（Concurrent Mode Failure或PromotionFailed 的问题），由于浮动垃圾太多，内存太大，所以停顿时间会超级长。

怎么解决：

- 降低触发CMS的阈值

  调整参数 –XX:CMSInitiatingOccupancyFraction 92%

  意思是92%的时候才会触发FGC，可以把这个值降低一点 ，老年的内存被占用到92%的时候CMS开始工作。让CMS保持老年代足够的空间

▼ **G1** **点击跳转**

常见垃圾回收器的参数设定（1.8）

- -XX:+UseSerialGC = Serial New (DefNew) + Serial Old

  通过这条命令可以指定垃圾回收器

  小型程序。默认情况下不会是这种选项，HotSpot会根据计算及配置和JDK版本自动选择收集器

- -XX:+UseParNewGC = ParNew + SerialOld **（这种组合已经很少用了）**
  - 这个组合已经很少用（在某些版本中已经废弃）
  - https://stackoverflow.com/questions/34962257/why-remove-support-for-parnewserialold-anddefnewcms-in-the-future
- -XX:+UseConcMarkSweepGC = ParNew + CMS + Serial Old
- -XX:+UseParallelGC = Parallel Scavenge + Parallel Old (1.8默认) 【PS + SerialOld】
- -XX:+UseParallelOldGC = Parallel Scavenge + Parallel Old
- -XX:+UseG1GC = G1
- Linux中没找到默认GC的查看方法，而windows中会打印UseParallelGC
  - java +XX:+PrintCommandLineFlags -version

    - 通过GC的日志来分辨
- Linux下1.8版本默认的垃圾回收器到底是什么？
    - 1.8.0_181 默认（看不出来）Copy MarkCompact
    - 1.8.0_222 默认 PS + PO

各种GC

1. JDK诞生 Serial追随 提高效率，诞生了PS，为了配合CMS，诞生了PN，CMS是1.4版本后期引入，CMS是里程碑式的GC，它开启了并发回收的过程，但是CMS毛病较多，因此目前没有任何一个JDK版本默认是CMS
   并发垃圾回收是因为无法忍受STW ---Stop the word
2. Serial 年轻代 串行回收
3. PS 年轻代 并行回收 并行，多个GC线程同时回收
4. ParNew 年轻代 配合CMS的并行回收
5. SerialOld
6. ParallelOld
7. ConcurrentMarkSweep 老年代 并发的，GC线程和应用程序线程同时执行， 垃圾回收和应用程序同时运行，降低STW的时间(200ms)
   CMS问题比较多，所以现在没有一个版本默认是CMS，只能手工指定
   CMS既然是MarkSweep，就一定会有碎片化的问题，碎片到达一定程度，CMS的老年代分配对象分配不下的时候，使用SerialOld 进行老年代回收
   想象一下：
   PS + PO → 加内存 换垃圾回收器 → PN + CMS + SerialOld（几个小时 - 几天的STW）
   几十个G的内存，单线程回收 → G1 + FGC 几十个G → 上T内存的服务器 ZGC
   算法：三色标记 + Incremental Update
8. G1(10ms)
   算法：三色标记 + SATB
9. ZGC (1ms) PK C++
   算法：ColoredPointers + LoadBarrier

10. Shenandoah
    算法：ColoredPointers + WriteBarrier
11. Eplison
12. PS 和 PN区别的延伸阅读：
    ▪https://docs.oracle.com/en/java/javase/13/gctuning/ergonomics.html#GUID-3D0BB91E-9BFF-4EBB-B523-14493A860E73
13. 垃圾收集器跟内存大小的关系
    1. Serial　几十兆
    2. PS　上百兆 - 几个G
    3. CMS - 20G
    4. G1 - 上百G
    5. ZGC - 4T - 16T（JDK13）

1.8默认的垃圾回收：PS + ParallelOld