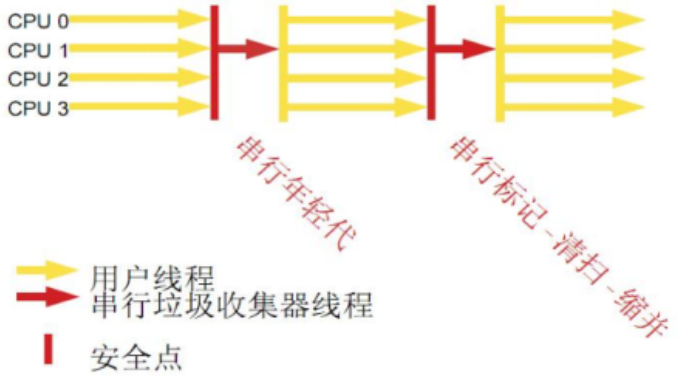
一、串行的垃圾收集器

1、Serial收集器(新生代收集器,串行GC)（复制算法)



特性：

Serial收集器是最基本的新生代收集器，这个收集器是一个单线程的收集器，但它的“单线程”的意义并不仅仅说明它只会使用一个CPU或一条收集线程去完成垃圾收集工作，更重要的是在它进行垃圾收集时，必须暂停其他所有的工作线程，直到它收集结束(Stop The World).

垃圾回收的过程：

先是所有的用户线程运行，然后在进行垃圾回收时，其他的用户线程都会在安全点停留，只有一个线程进行垃圾回收，然后其他的线程都会进行阻塞，直到垃圾回收线程收集结束之后，其他所有线程恢复运行。

应用场景：

Serial收集器是虚拟机运行在Client模式下的默认新生代收集器。用单线程处理所有垃圾回收工作，因为无需多线程交互，所以效率比较高。但是，也无法使用多处理器的优势，所以此收集器适单处理器机器。当然，此收集器也可以用在小数据量（100M左右）情况下的多处理器机器上。可以使用-XX:+UseSerialGC打开。

优势:

简单而高效（与其他收集器的单线程比），对于限定单个CPU的环境来说，Serial收集器器由于没有线程交互的开销，专心做垃圾收集自然可以获得最高的单线程收集效率。 实际上到现在为止 ： **它依然是虚拟机运行在Client模式下的默认新生代收集器。**

2、Serial Old收集器(老年代收集器，串行GC)（标记+整理)

与之对应的老年代收集器就是Serial Old收集器，它同样是一个单线程收集器，用的是标记加整理算法。

应用场景：

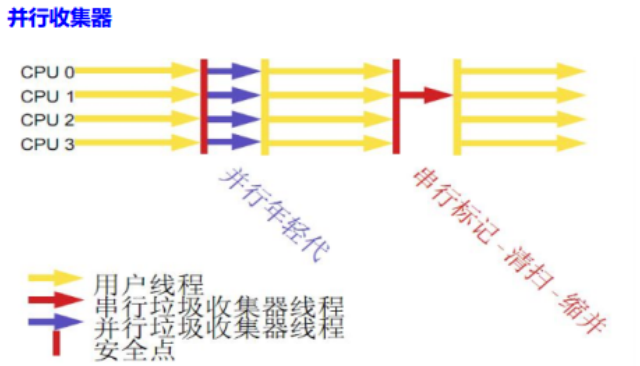
Client模式 ：Serial Old收集器的主要意义也是在于给Client模式下的虚拟机使用。

Server模式： 如果在Server模式下，那么它主要还有两大用途：一种用用途是在JDK 1.5以及之前的版本中与Parallel Scavenge收集器搭配使用，另一种用途就是作为CMS收集器器的后备预案，在并发收集发生Concurrent Mode Failure时使用。

二、吞吐量优先的垃圾收集器

1，Parallel Scavenge收集器(新生代收集器,并行GC)（复制算法）

Parallel Scavenge收集器是一个新生代并行收集器，它也是使用复制算法的收集器，又是并行的多线程收集器。



Parallel Scavenge收集器使用两个参数控制吞吐量：

XX:MaxGCPauseMillis 控制最大的垃圾收集停顿时间

XX:GCRatio 直接设置吞吐量量的大小

直观上，只要最大的垃圾收集停顿时间越小，吞吐量是越高的，但是GC停顿时间的缩短是以牺牲吞吐量和新生代空间作为代价的。比如原来10秒收集一次，每次停顿100毫秒，现在变成5秒收集一次，每次停顿70毫秒。停顿时间下降的同时，吞吐量也下降了。

应用场景：

停顿时间越短就越适合需要与用户交互的程序，良好的响应速度能提升用户体验，而高吞吐量则可以高效率地利用用CPU时间，尽快完成程序的运算任务，主要适合在后台运算而不需要太多交互的任务。

GC自适应的调节策略：

Parallel Scavenge收集器器有一个参数**- XX:+UseAdaptiveSizePolicy** 。当这个参数打开之后，就不需要手工指定新生代的大小、Eden与Survivor区的比例、晋升老年代对象年龄等细节参数了了，虚拟机会根据当前系统的运行情况收集性能监控信息，动态调整这些参数以提供最合适的停顿时间或者最大的吞吐量，这种调节方式称为GC自自适应的调节策略（GC Ergonomics）

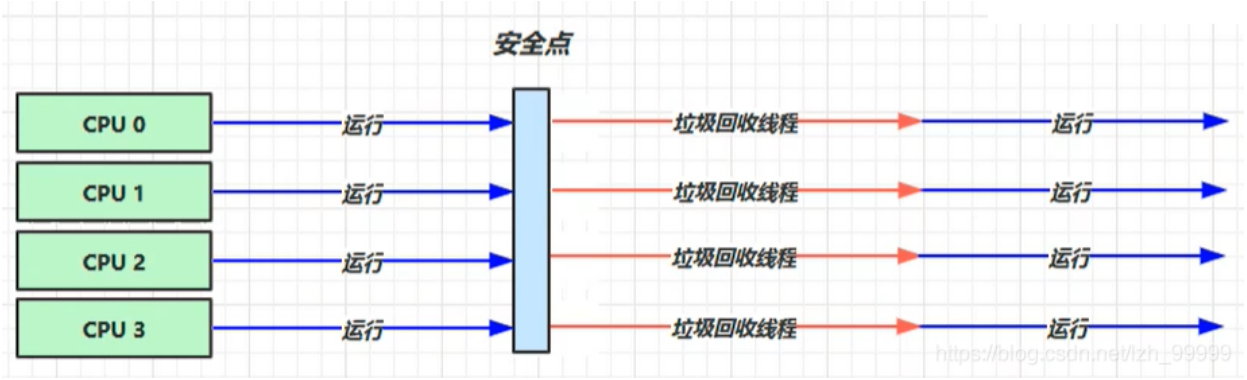
2，Parallel Old收集器(老年代收集器，并行GC)（标记+整理)

使用-XX:+UseParallelOldGC打开。
使用-XX:ParallelGCThreads=<N>设置并行垃圾回收的线程数。此值可以设置与机器处理器数量相等。
此收集器可以进行如下配置：
最大垃圾回收暂停:指定垃圾回收时的最长暂停时间，通过-XX:MaxGCPauseMillis=<N>指定。<N>
为毫秒.如果指定了此值的话，堆大小和垃圾回收相关参数会进行调整以达到指定值。设定此值可能会
减少应用的吞吐量。
吞吐量:吞吐量为垃圾回收时间与非垃圾回收时间的比值，通过-XX:GCTimeRatio=<N>来设定，公式
为1/（1+N）。例如，-XX:GCTimeRatio=19时，表示5%的时间用于垃圾回收。默认情况为99，即
1%的时间用于垃圾回收。

特性： Parallel Old是Parallel Scavenge收集器的老年代版本，使用多线程和“标记－整理”算法。

应用场景：
在注重吞吐量以及CPU资源敏感的场合，都可以优先考虑Parallel Scavenge加Parallel Old收集器。

垃圾回收的过程

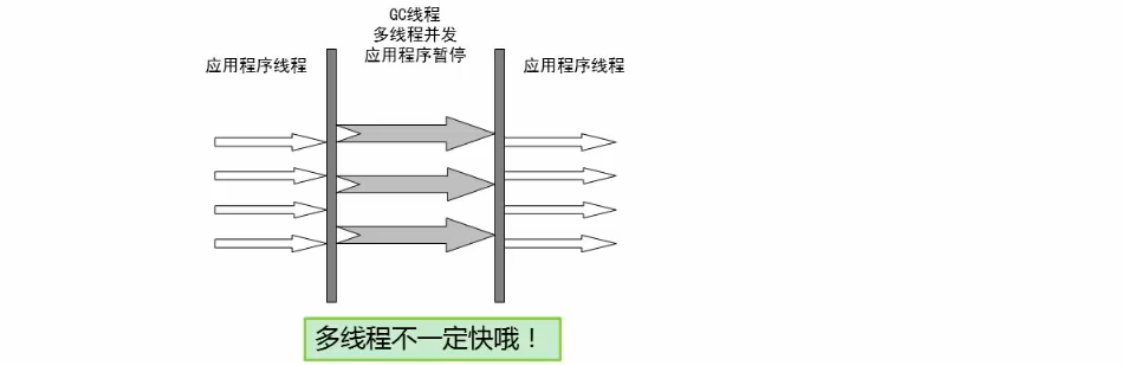


在多线程的情况下运行，有效的利用率cpu的资源

在多个线程进行运行时，要进行垃圾回收时，多个线程会停止到安全点，然后所有的线程都进行垃圾回收，然后回收完毕后都重新进行运行。

特点：提高了cpu的使用效率，并且加大了吞吐量，**就是停顿时间加长了，主要适用于不太进行交互的任务，并且吞吐量大的时候**，还有一个GC 自适应调节策略，可以自己控制吞吐量以及最大停顿时间。

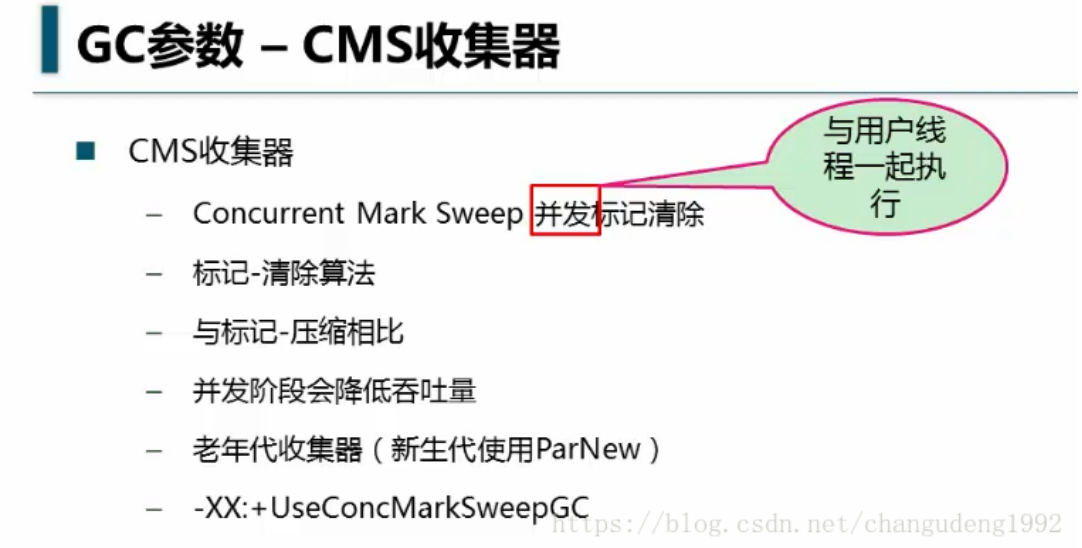
3. ParNew收集器（新生代并行收集器）（复制算法）




```
0.834: [GC 0.834: [ParNew: 13184K->1600K(14784K), 0.0092203 secs] 13184K->1921K(63936K),
0.0093401 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
```


4. CMS 并发收集器（老年代收集器，并发GC）（标记+清除）

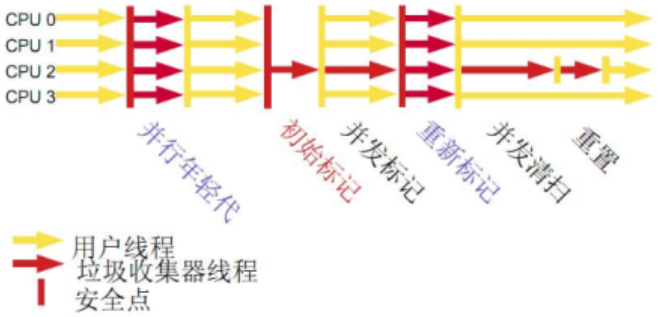
并发收集器主要减少年老代的暂停时间，他在应用不停止的情况下使用独立的垃圾回收线程，跟踪可达对象。在每个年老代垃圾回收周期中，在收集初期并发收集器 会对整个应用进行简短的暂停，在收集中还会再暂停一次。第二次暂停会比第一次稍长，在此过程中多个线程同时进行垃圾回收工作。



并发concurrent的含义：**垃圾回收器与应用程序线程一起执行，交替执行，减少应用程序的停顿时间，**但是**并发运行垃圾回收器会消耗cpu的资源导致应用的整个吞吐量下降。**CMS收集器是个单纯的老年代的收集器，通过如上命令开启。

并发收集器

可以保证大部分工作都并发进行（应用不停止），垃圾回收只暂停很少的时间，此收集器适合对响应时间要求比较高的中、大规模应用。使用-XX:+UseConcMarkSweepGC打开。



GC参数 – CMS收集器

- CMS运行过程比较复杂，着重实现了标记的过程，可分为
  - 初始标记
    - 根可以直接关联到的对象
    - 速度快
  - 并发标记（和用户线程一起）
    - 主要标记过程，标记全部对象
  - 重新标记
    - 由于并发标记时，用户线程依然运行，因此在正式清理前，再做修正
  - 并发清除（和用户线程一起）
    - 基于标记结果，直接清理对象

并发标记和并发清除这两步是和用户线程一起执行的， 但是初始标记和重新标记是会有停顿的。

- 特点
  - 尽可能降低停顿
  - 会影响系统整体吞吐量和性能
    - 比如，在用户线程运行过程中，分一半CPU去做GC，系统性能在GC阶段，反应速度就下降一半
  - 清理不彻底
    - 因为在清理阶段，用户线程还在运行，会产生新的垃圾，无法清理
  - 因为和用户线程一起运行，不能在空间快满时再清理
    - -XX:CMSInitiatingOccupancyFraction设置触发GC的阈值
    - 如果不幸内存预留空间不够，就会引起concurrent mode failure

CMS回收器失败产生，启用备用的串行回收器。

由于CMS要与用户线程同步，不能对碎片进行整理， 所以采用标记清除算法。与串行回收和并行回收的区别是这二者使用的是标记压缩算法可以有效提高内存使用效率。

CMS线程数量一般设置为可用CPU的数量， 不宜设置过大。

5. G1并发收集器 （全堆收集器，并发GC） （标记-复制算法 + 标记-整理算法）

The Garbage-First (G1) 是服务器类型的垃圾收集器，它适用于大内存的垃圾收集处理并且可以控制垃圾收集 STW 暂停时间长度（当然这个时间的设置需要是合理的），在 JDK 1.7 版本 G1 得到了完全的支持。

G1概览

G1 GC 全称是Garbage First Garbage Collector，垃圾优先垃圾回收器，以下简称G1。
G1是HotSpot JVM的短停顿垃圾回收器。
其实关于G1的论文早在2004年就有了，但是G1是在2012年4月发布的JDK 7u4中才实现。
从长期来说，G1旨在取代CMS（Concurrent Mark Sweep）垃圾回收器。
G1从JDK9开始已经作为默认的垃圾回收器。
如果对于应用程序来说停顿时间比吞吐量更重要，G1是非常合适的选择。

总体来说G1具有如下特点：

G1仍旧是分代（年轻代，老年代）的垃圾回收器
G1实现了两种垃圾回收算法 -> 标记-复制算法 + 标记-整理算法
G1不用一次性将堆中的垃圾都回收,G1会动态选择垃圾多的一块内存优先回收

G1收集器（或者垃圾优先收集器）的设计初衷是为了尽量缩短处理超大堆（大于4GB）时产生的停顿。相对于CMS的优势而言是内存碎片的产生率大大降低。

G1 通过将活动对象从将一组或者多组区域（该区域称为 集合集 CSet） 增量并行复制到一个或者多个不同的新的区域来实现空间压缩来减少堆内存的碎片。目标是回收尽可能多的内存空间，所以在垃圾回收时 G1 会在满足暂停时间期望的前提下优先挑选垃圾对象最多的区域进行回收。

开启G1收集器的方式-XX:+UseG1GC

小结

串行处理器：（Serial New, Serial Old）

--适用情况：数据量比较小（100M左右）；单处理器下并且对响应时间无要求的应用。

--缺点：只能用于小型应用

并行处理器： （Parallel Scavenge，ParNew，Parallel Old）

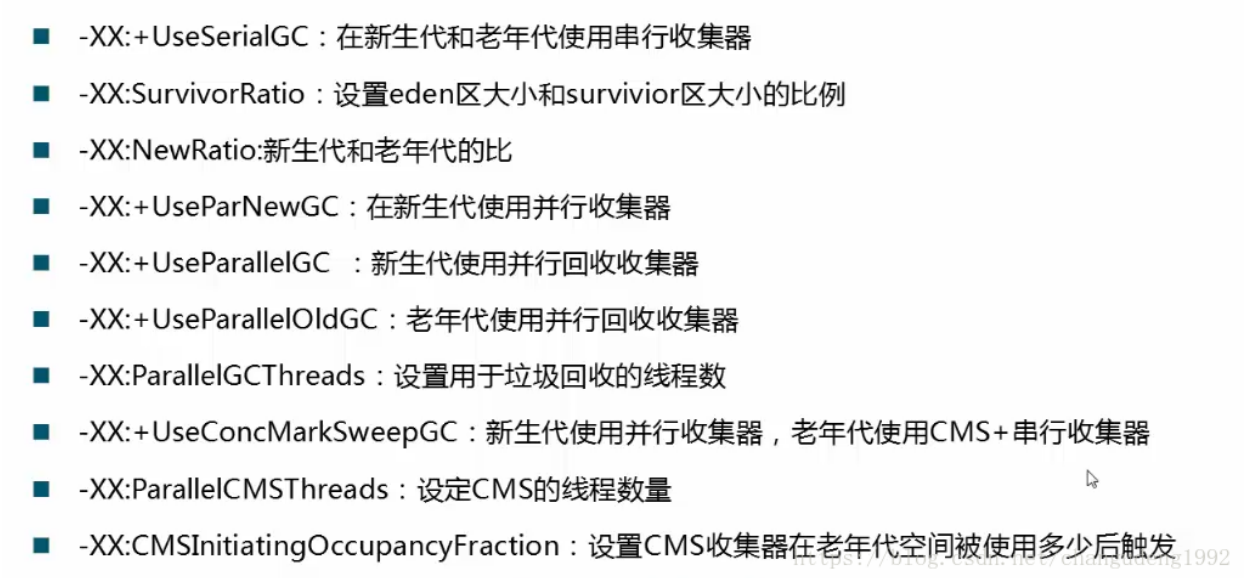
--适用情况：“对吞吐量有高要求”，多CPU、对应用响应时间无要求的中、大型应用。举例：后台处理、科学计算。

--缺点：垃圾收集过程中应用响应时间可能加长

并发处理器： （CMS）

适用情况：“对响应时间有高要求”，多CPU、对应用响应时间有较高要求的中、大型应用。举例：Web服务器/应用服务器、电信交换、集成开发环境

GC参数整理：



java垃圾收集器的历史

第一阶段，Serial（串行）收集器

在jdk1.3.1之前，java虚拟机仅仅能使用Serial收集器。 Serial收集器是一个单线程的收集器，但它的“单线程”的意义并不仅仅是说明它只会使用一个CPU或一条收集线程去完成垃圾收集工作，更重要的是在它进行垃圾收集时，必须暂停其他所有的工作线程，直到它收集结束。

PS：开启Serial收集器的方式

-XX:+UseSerialGC

第二阶段，Parallel（并行）收集器

Parallel收集器也称吞吐量收集器，相比Serial收集器，Parallel最主要的优势在于使用多线程去完成垃圾清理工作，这样可以充分利用多核的特性，大幅降低gc时间。

PS:开启Parallel收集器的方式

-XX:+UseParallelGC -XX:+UseParallelOldGC

第三阶段，CMS（并发）收集器

CMS收集器在Minor GC时会暂停所有的应用线程，并以多线程的方式进行垃圾回收。在Full GC时不再暂停应用线程，而是使用若干个后台线程定期的对老年代空间进行扫描，及时回收其中不再使用的对象。

PS:开启CMS收集器的方式

-XX:+UseParNewGC -XX:+UseConcMarkSweepGC

第四阶段，G1（并发）收集器

G1收集器（或者垃圾优先收集器）的设计初衷是为了尽量缩短处理超大堆（大于4GB）时产生的停顿。相对于CMS的优势而言是内存碎片的产生率大大降低。

PS:开启G1收集器的方式

-XX:+UseG1GC