JVM

1：

new的时候触发隐式加载 和显式加载， 双亲委派机制；

加载：class 解析成方法区的数据--生成Class类

验证：检查是否会对jvm造成破坏，符号\字节码的合法性 都做了验证

准备: 把静态数据置为初始值

解析: 解析动态引用

初始化:执行静态代码块

加载=》链接

检查

准备

解析

初始化

2：java的内存结构

本地方法栈，方法区 程序计数器 堆 栈

3：永久代就是方法区， 存放类的原始信息的地方

后来改成元空间 由操作系统管理

4：

垃圾回收的算法

5：垃圾回收期

cms：

首先标机根 stw 然后 并发标记出来， 在修复上一个阶段的修改 stw并行， 最后并发的执行清理工作， 这样增大了程序的吞吐量；

G1：

分区算法 把堆分成了一个小块 一个小块的，

每个小块的角色是可以转换的

对一个小块执行回收算法 回收到survirvor里面；

6：辣鸡回收算法

标机清理

标机复制

标机整理

分区 分代

在新生代 eden区域的辣鸡 会和from赋值到to 然后to和from的角色呼唤

如果对象很大 分配到老年代

到达15的阈值也会到老年代

新生代是复制

老年代通常就是整理算法了；

7：辣鸡泄露的情景

主要是静态变量(可以炫一波threadlocal的弱引用和启发式扫描原理(介于纯lazy和强制所有扫描之间的一种方式)) 数据库连接对象（需要经常关闭） 会长期占有内存，造成内存泄漏的问题

提一嘴ThreadLocal 使用的是弱引用， 所以会主动清理；

启发式扫描；

8：

jni java本地方法的时候 gc的时候 会使用finalize（）

9：jmm内存模型

原子性：

10 volatile 和内存屏障的问题：

几个区域的理解：

1：虚拟机栈为什么为OOM

stackoverflow就是这个原理

创建的线程数目过多 会发生OOM

2：

eden s1 s2的具体描述：

们 new 一个对象后，会先放到 Eden 划分出来的一块作为存储空间的内存，但是我们知道对堆内存是线程共享的，所以有可能会出现两个对象共用一个内存的情况。这里 JVM 的处理是为每个线程都预先申请好一块连续的内存空间并规定了对象存放的位置，而如果空间不足会再申请多块内存空间。这个操作我们会称作 TLAB，有兴趣可以了解一下。

tlab是线程的自己的空间；

Eden 空间满了之后，会触发一个叫做 Minor GC（就是一个发生在年轻代的 GC）的操作，存活下来的对象移动到 Survivor0 区。from区域 from满了 minor gc到 to区域 此时from为空； 交换from和to的指针，使得 to再次指向空区域；

大的对象 直接到老年代；

每次minorgc后存活 生命+1 达到15就可以移动到老年代；

Survivor0 区 的对象什么时候垃圾回收呢？

假设 Survivor0 区现在是满的，此时又触发了 Minor GC ，发现 Survivor0 区依旧是满的，存不下，此时会将 S0 区与 Eden 区的对象一起进行可达性分析，找出活跃的对象，将它复制到 S1 区并且将 S0 区域和 Eden 区的对象给清空，这样那些不可达的对象进行清除，并且将 S0 区 和 S1 区交换。

老年代fullgc的时候 如果没有办法分配内存 就会oom，虚拟机崩了，首先检测自己的代码逻辑有没有问题， 出现了内存泄漏没有， 然后在考虑扩大堆内存；

3:吞吐量和暂停时间

如果选择以吞吐量优先，那么必然需要降低内存回收的执行频率，但是这样会导致 GC 需要更长的暂停时间来执行内存回收

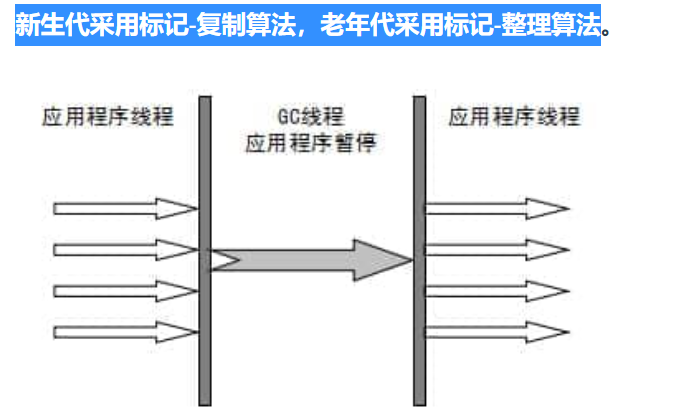
相反的，如果选择以低延迟优先为原则，那么为了降低每次执行内存回收时的暂停时间，也只能频繁地执行内存回收，但这又引起了年轻代内存的缩减和导致程序吞吐量的下降；

拆分成小的任务，有附加的拆分代价，导致整体的吞吐量下降；

标记清除

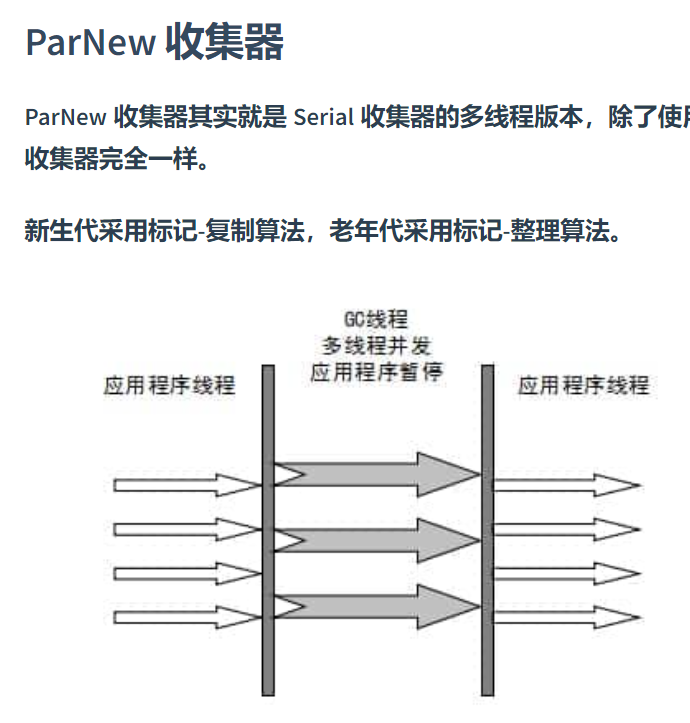
4：不同辣鸡回收的原理：

seial：

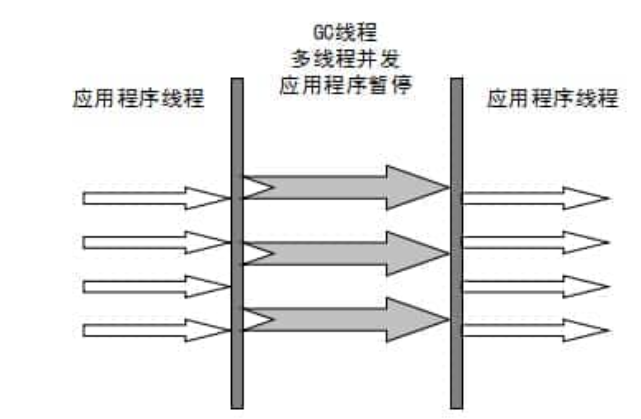


标记复制

标记整理（压缩）



parallel scavenge算法



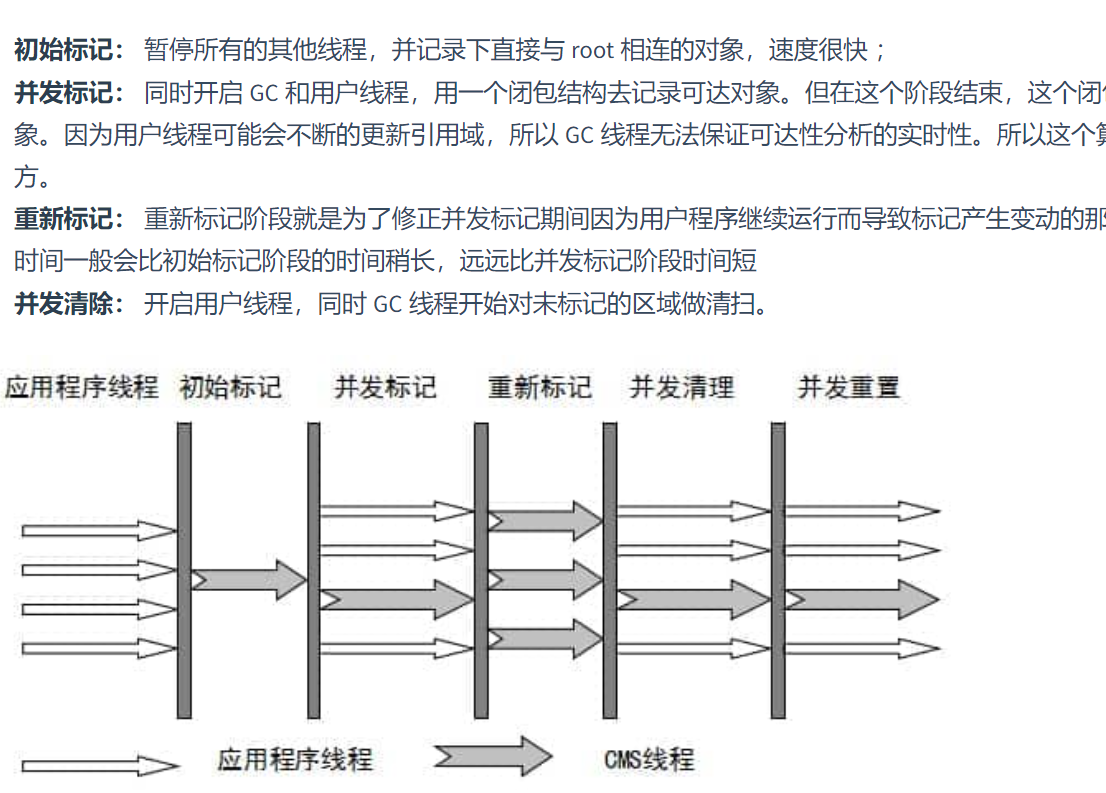
Serial Old 收集器

Serial 收集器的老年代版本，它同样是一个单线程收集器。它主要有两大用途：一种用途是在 JDK1.5 以及以前的版本中与 Parallel Scavenge 收集器搭配使用，另一种用途是作为 CMS 收集器的后备方案。

Parallel Old 收集器

Parallel Scavenge 收集器的老年代版本。使用多线程和“标记-整理”算法。在注重吞吐量以及 CPU 资源的场合，都可以优先考虑 Parallel Scavenge 收集器和 Parallel Old 收集器。

CMS算法：



初始标记：所有gc直连的

并发标记：用户线程工作 并发的标记处来可达分析

重新标记：STW做变动

并发清除：对未标记的清除；