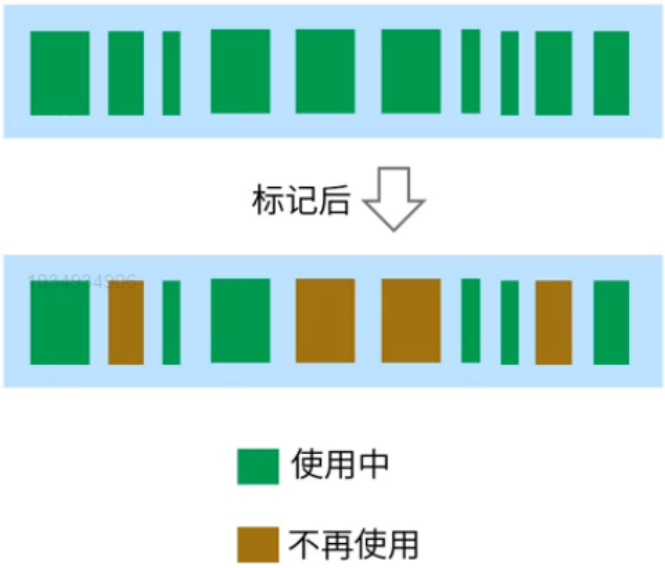
垃圾回收机制

自动垃圾收集

1. 自动垃圾收集是查看堆内存，识别正在使用哪些对象以及哪些对象未被删除以及删除未使用对象的过程。
2. 使用中的对象或引用的对象意味着程序的某些部分仍然维护着指向该对象的指针。
3. 程序的任何部分都不再引用未使用的对象或者未引用的对象，因此可以回收未引用对象使用的内存。
4. C语言中，分配和释放内存是一个手动的过程，在Java中解除分配内存的过程由垃圾收集器自动处理。

如何确定内存需要被回收

1. 该过程的第一步称为标记。这是垃圾收集器识别哪些内存正在使用而哪些不再使用的地方。方法区内存回收对应于类的卸载过程。

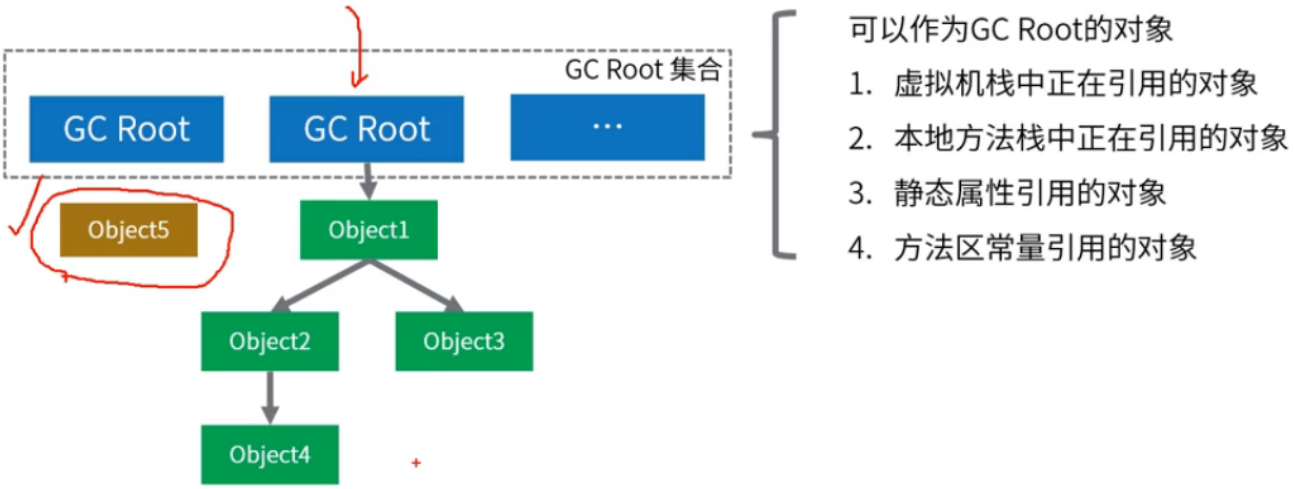


不同类型内存的判断方式

- ❑ 对象回收 – 引用计数
- ❑ 对象回收 - 可达性分析
- ❑ 方法区回收

可达性分析算法

1. 简单来说，将对象及其引用关系看作一个图，选定活动对象作为GC Roots。然后跟踪调用链，如果一个对象和GC Roots之间不可达，也就是不存在引用，那么即可认为是可回收对象。



引用类型和可达性分析

引用类型

1. 强引用：最常见的普通对象引用，只要还有强引用指向一个对象，就不会回收。
2. 软引用：JVM认为内存不足，才会去试图回收软引用指向的对象(缓存场景)。
3. 弱引用：虽然是引用，但随时可能被回收掉。
4. 虚引用：不能通过它访问对象。供对象被finalize以后，执行指定逻辑的机制(Cleaner)。比如清理堆外内存，当虚拟机要回收堆外内存时，会去检查虚引用队列中是否有值，如果有值说明该对象要被虚拟机回收了，就需要清理堆外的相应内存。

可达性级别：

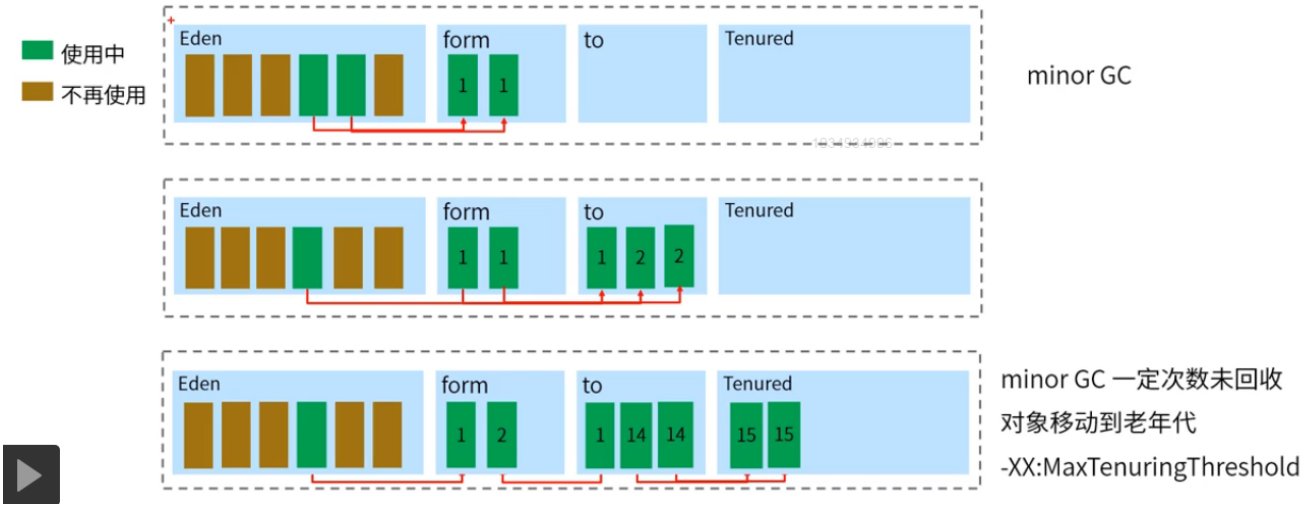
1. 强可达：一个对象可以有一个或多个线程可以通过各种引用访问到的情况。
2. 软可达：就是当我们只能通过软引用才能访问到对象的状态。
3. 弱可达：只能通过弱引用访问时的状态。当弱引用被清除的时候，就符合销毁条件。
4. 幻象可达：不存在其他引用，并且finalize过了(下一次垃圾回收会回收该对象所占用的内存)，只有幻象引用指向这个对象。
5. 不可达：意味着对象可以被清除了。

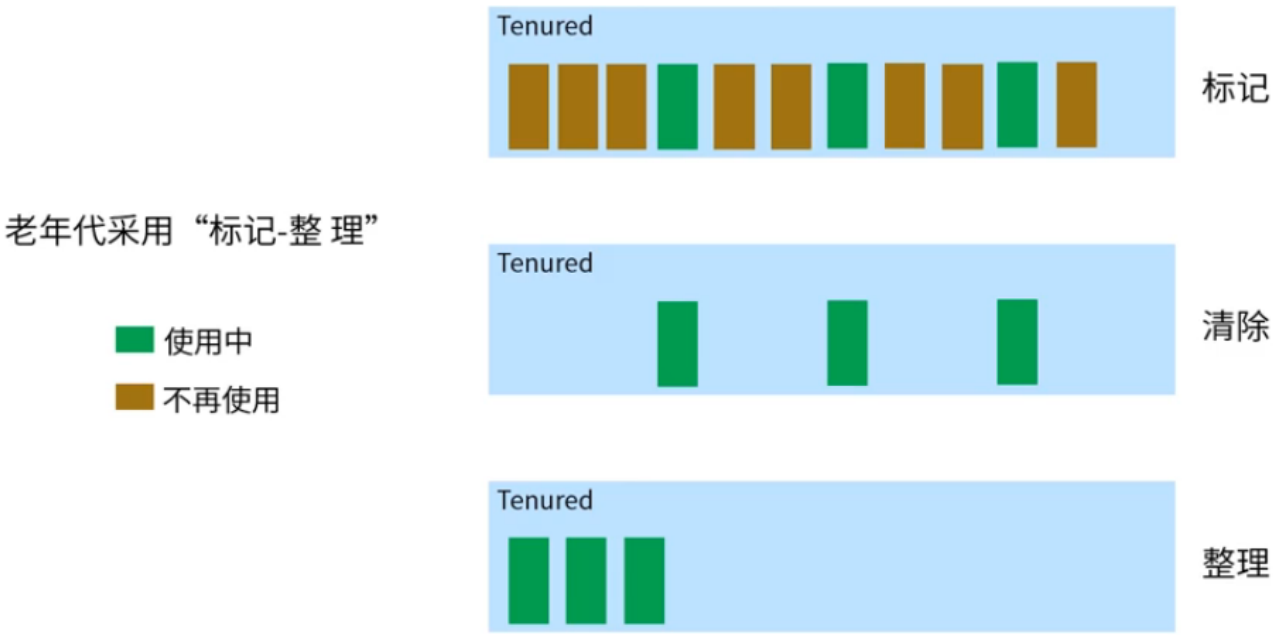
垃圾收集算法

1. 标记-清除：首先标识出所有要回收的对象，然后进行清除。标记、清除过程效率有限，有内存碎片化问题，不适合特别大的堆。收集算法基本基于标记-清除的思路进行改进。
2. 复制：划分两块同等大小的区域，收集时将活着的对象复制到另一块区域。拷贝过程中将对象顺序放置，就可以避免内存碎片化。复制+预留内存，有一定的浪费。
3. 标记-整理：类似于标记-清除，但为避免内存碎片化，它会在清理过程中将对象移动，以确保移动后的对象占用连续的内存空间。

分代收集

1. 根据对象的存活周期，将内存划分为几个区域，不同区域采用合适的垃圾收集算法。新对象会分配到Eden，如果超过-XX:+PretenureSizeThreshold：设置大对象直接进入老年代的阈值。新生代S0:S1:Eden->1:1:8，新生代:老年代->1:2。





垃圾收集器

串行垃圾收集器

1. 串行垃圾收集器-Serial GC -XX:+UseSerialGC，单个线程来执行所有垃圾收集工作，适合单处理器机器。Client模式下JVM默认选项。
2. 串行垃圾收集器-Serial Old -XX:+UseSerialGC，可以在老年代使用，它采用标记-整理算法，区别于新生代的复制算法。

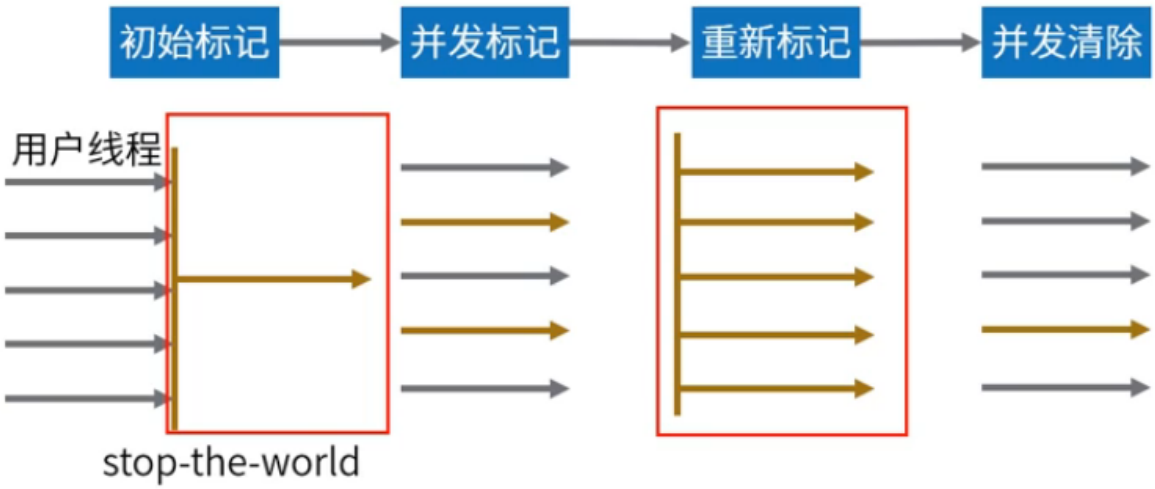
并行垃圾收集器

1. 并行垃圾收集器-Parallel GC -XX:+UserParallelGC，-Parallel GC -XX:UseParallelOldGC。
2. Server模式JVM的默认GC选择，整理算法和Serial比较相似，区别是新生代和老年代GC都是并行进行的。可以设置GC时间或吞吐量等值，可以自动进行适应性调整Eden，Survivor大小和MaxTenuringThreshold的值。也称为吞吐量优先的GC:吞吐量=用户代码运行时间/(用户代码时间 + GC时间)。
3. -XX:ParallelGCThreads：设置用于垃圾回收的线程数。通常情况下可以和CPU数量相等。
4. -XX:MaxGCPauseMills：设置最大垃圾收集停顿时间。它的值是一个大于0的整数。
5. -XX:GCTimeRatio：设置吞吐量打下，它的值是一个0-100之间的值。
6. -XX:+UseAdaptiveSizePolicy：打开自使用GC策略。以达到在堆大小、吞吐量和停顿时间之间的平衡点。

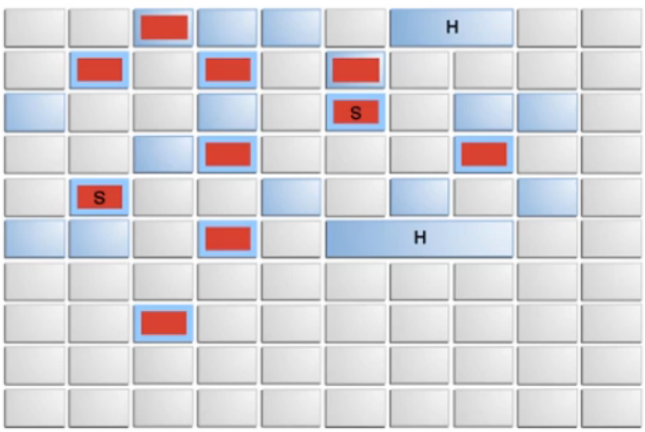
并发收集器

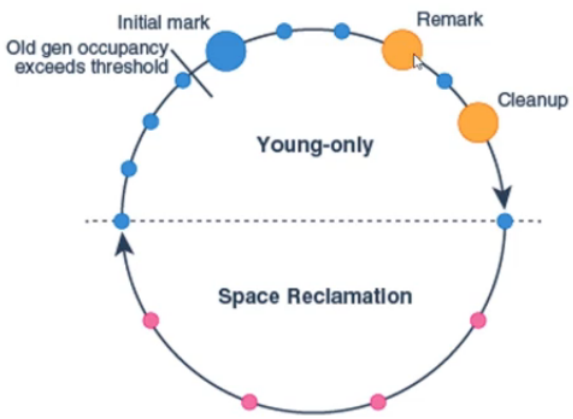
1. CMS GC -XX:+UseConcMarkSweepGC。
2. 专用老年代，基于标记-清除算法，设计目标是尽量减少停顿时间。采用的标记-清除算法，存在着内存碎片化问题，长时间运行等情况会发生full GC，导致恶略的停顿。CMS会占用过多的CPU资源，并和用户线程争抢。

   减少了停顿时间，这一点对于互联网web等对时间敏感的系统非常重要，一直到今天，仍然有很多系统使用CMSGC。



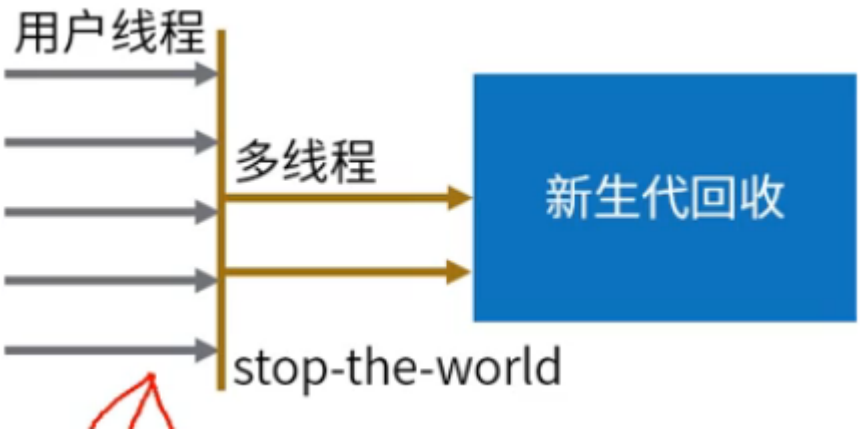
3. -G1 -XX:+UseG1GC。
4. 针对大堆内存设计的收集器。兼顾吞吐量和停顿时间，JDK9后为默认选型，目标是替代CMS。G1将堆分为固定大小的区域，Region之间是复制算法，但整体上实际上可以看作是标记-整理算法，可以有效地避免内存碎片。红色新生代(Eden和Survivor)，淡蓝色老年代。找不到大内存时执行FullGC。





并行垃圾收集器

1. -ParNew GC -XX:+UseParNewGC。
2. 新生代GC实现，它实际是Serial GC的多线程版本。可以控制线程数量，参数：-XX:ParallelGCThreads。最常见的应用场景是配合老年代的CMS GC工作。参数-XX:+UseConcMarkSweepGC。



垃圾回收器组合