JVM 的垃圾回收主要指 JVM 中的堆内存回收。以下从这些方面讲述垃圾回收原理：

目录

1.哪些对象可以回收？

JVM 通过一种可达性分析算法进行垃圾对象的识别： GC Roots 是所有对象的根对象，在 JVM 加载时，会创建一些普通对象引用正常对象。这些对象作为正常对象的起始点，在垃圾回收时，会从这些 GC Roots 开始向下搜索，当一个对象到 GC Roots 没有任何引用链相连时，就证明此对象是不可用的。目前 HotSpot 虚拟机采用的就是这种算法。

JVM 规范中并没有明确 GC 的运作方式，各个厂商可以采用不同的方式实现垃圾收集器。

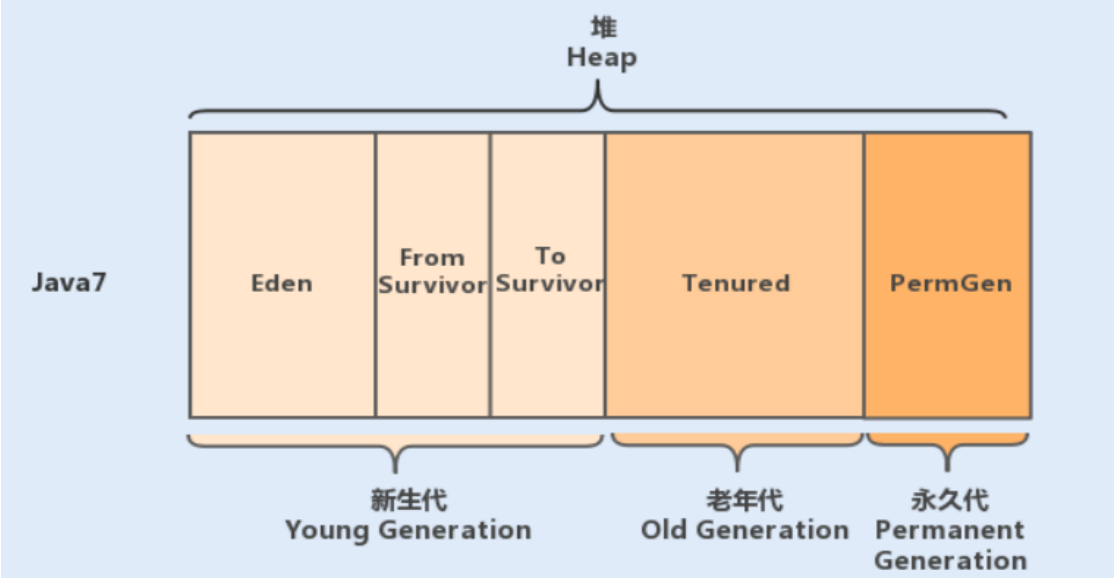
2.使用什么回收算法：

JVM 提供了不同的回收算法来实现这一套回收机制，通常垃圾收集器的回收算法可以分为以下几种：

回收算法类型	优点	缺点
标记-清除算法（Mark-Sweep）	不需要移动对象，简单高效	标记-清除过程效率低，GC产生内存碎片
复制算法（Copying）	简单高效，不会产生内存碎片	内存使用率低，且有可能产生频繁复制问题
标记-整理算法（Mark-Compact）	综合了前两种算法的优点	仍需要移动局部对象
分代收集算法（Generational Collection）	分区回收	对于长时间存活对象的场景回收效果不明显，甚至起到反作用

3. 堆内存结构:

堆被划分为新生代和老年代，新生代又被进一步划分为 Eden 和 Survivor 区，最后 Survivor 由 From Survivor 和 To Survivor 组成。在 Java6 版本中，永久代在非堆内存区；到了 Java7 版本，永久代的静态变量和运行时常量池被合并到了堆中；而到了 Java8，永久代被元空间取代了。



4. 内存的分配与回收过程：

1）当我们新建一个对象时，对象会被优先分配到新生代的 Eden 区中，这时虚拟机会给对象定义一个对象年龄计数器（通过参数 -XX:MaxTenuringThreshold 设置）。

2）当 Eden 空间不足时，虚拟机将会执行一个新生代的垃圾回收（Minor GC）。这时 JVM 会把存活的对象转移到 Survivor 中，并给对象的年龄 +1。对象在 Survivor 中同样也会经历 MinorGC，每经过一次 MinorGC，对象的年龄将会 +1

3）当一个对象经过几次 MinorGC 后依然存活，那么这个对象就会被复制到老年代区域。

4）当老年代空间已满，就会对新生代和老年代的内存空间进行一次全量垃圾回收，即 Full GC。

5.垃圾回收器

如果说垃圾回收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。

回收器类型	回收算法	特点	设置参数
Serial New / Serial Old回收器	复制算法/标记-整理算法	单线程复制回收，简单高效，但会暂停程序导致停顿	-XX:+UseSerialGC（年轻代、老年代回收器为：Serial New、Serial Old）
ParNew New / ParNew Old回收器	复制算法/标记-整理算法	多线程复制回收，降低了停顿时间，但容易增加上下文切换	-XX:+UseParNewGC（年轻代、老年代回收器为：ParNew New、Serial Old，JDK1.8中无效） -XX:+UseParallelOldGC（年轻代、老年代回收器为：Parallel Scavenge、Parallel Old）
Parallel Scavenge回收器	复制算法	并行回收器，追求高吞吐量，高效利用CPU	-XX:+UseParallelGC（年轻代、老年代回收器为：Parallel Scavenge、Serial Old） -XX:ParallelGCThreads=4（设置并发线程）
CMS回收器	标记-清理算法	老年代回收器，高并发、低停顿，追求最短GC回收停顿时间，CPU占用比较高，响应时间快，停顿时间短	-XX:+UseConcMarkSweepGC（年轻代、老年代回收器为：ParNew New、CMS（Serial Old作为备用））
G1回收器	标记-整理+复制算法	高并发、低停顿，可预测停顿时间	-XX:+UseG1GC（年轻代、老年代回收器为：G1、G1） -XX:MaxGCPauseMillis=200（设置最大暂停时间）

其中重点关注以下的 4 种垃圾回收器。

1） Serial 串行垃圾回收器，这是 JVM 早期的垃圾回收器，只有一个线程执行垃圾回收。

2） Parallel 并行垃圾回收器，它启动多线程执行垃圾回收。如果 JVM 运行在多核 CPU 上，那么显然并行垃圾回收要比串行垃圾回收效率高。在串行和并行垃圾回收过程中，当垃圾回收线程工作的时候，必须要停止用户线程的工作，否则可能会导致对象的引用标记错乱，因此垃圾回收过程也被称为 stop the world，在用户视角看来，所有的程序都不再执行，整个世界都停止了。

3） CMS 并发垃圾回收器，在垃圾回收的某些阶段，垃圾回收线程和用户线程可以并发运行，因此对用户线程的影响较小。Web 应用这类对用户响应时间比较敏感的场景，适用 CMS 垃圾回收器。

4） G1 垃圾回收器，它将整个堆空间分成多个子区域，然后在这些子区域上各自独立进行垃圾回收，在回收过程中垃圾回收线程和用户线程也是并发运行。G1 综合了以前几种垃圾回收器的优势，适用于各种场景，是未来主要的垃圾回收器。