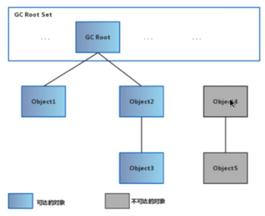
GC

判断一个对象是否是垃圾

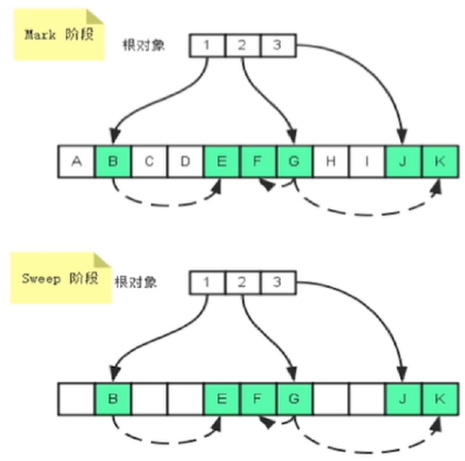
- 标准 — 该对象没有被其他对象引用
- 算法
  - 引用计数算法
    - 引用计数算法
      - 判断对象的引用数量
      - ➢ 通过判断对象的引用数量来决定对象是否可以被回收
      - ➢ 每个对象实例都有一个引用计数器，被引用则+1，完成引用则-1
      - ➢ 任何引用计数为0的对象实例可以被当作垃圾收集
    - 优点：执行效率高，程序执行受影响较小
    - 缺点：无法检测出循环引用的情况，导致内存泄露
  - 可达性分析算法
    - 可达性分析算法
      - 通过判断对象的引用链是否可达来决定对象是否可以被回收



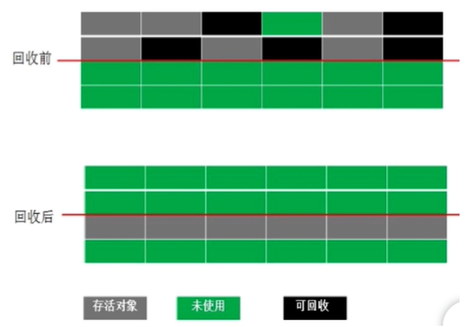
- 可以作为GC Root的对象
  - ➢ 虚拟机栈中引用的对象（栈帧中的本地变量表）
  - ➢ 方法区中的常量引用的对象
  - ➢ 方法区中的类静态属性引用的对象
  - ➢ 本地方法栈中JNI（Native方法）的引用对象
  - ➢ 活跃线程的引用对象

4种垃圾回收算法

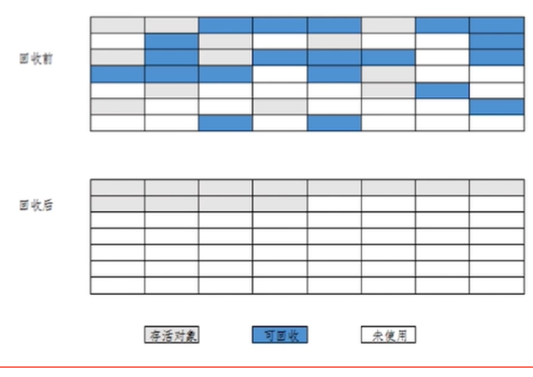
- 标记-清除
  - 标记-清除算法(Mark and Sweep)
    - ➢ 标记：从根集合进行扫描，对存活的对象进行标记
    - ➢ 清除：对堆内存从头到尾进行线性遍历，回收不可达对象内存
  - ➢ 碎片化



- 复制算法
  - 复制算法(Copying)
    - ➢ 分为对象面和空闲面
    - ➢ 对象在对象面上创建
    - ➢ 存活的对象被从对象面复制到空闲面
    - ➢ 将对象面所有对象内存清除
  - ➢ 解决碎片化问题
  - ➢ 顺序分配内存，简单高效
  - ➢ 适用于对象存活率低的场景



- 标记整理
  - ➢ 标记：从根集合进行扫描，对存活的对象进行标记
  - ➢ 清除：移动所有存活的对象，且按照内存地址次序依次排列，然后将末端内存地址以后的内存全部回收。
  - 标记-整理算法(Compacting)
    - ➢ 避免内存的不连续行
    - ➢ 不用设置两块内存互换
    - ➢ 适用于存活率高的场景



- 分代收集
  - ➢ 垃圾回收算法的组合拳
  - ➢ 按照对象生命周期的不同划分区域以采用不同的垃圾回收算法

jvm内存大小的取舍

内存大小的取舍

1. 扩大内存可以更少的触发gc
2. 内存太大触发gc时候的停顿时间会长

因此要根据你实际的业务场景设置成一个“合适”的值，并配合压测和线上环境的实际情况做不断的调优

吞吐量=花费在非GC停顿上的工作时间/总时间 至少需要优化到95%

-Xms 启动JVM时堆内存的大小

-Xmx 堆内存最大限制

两者需要设置的一样防止扩缩容

-XX:NewSize 年轻代大小

-XX:MaxNewSize 最大年轻代大小

两者需要设置的一样防止扩缩容

-XX:SurvivorRatio Eden Survivor 占比，默认为8

Eden需要比Survivor尽可能的大，防止多次触发young gc导致年龄快速增长到可以进入老年代的case

-XX:MetaspaceSize 元空间初始空间大小

-XX:MaxMetaspaceSize=512m 元空间最大空间，默认是没有限制的