JVM-GC

参考文献，深入理解Java虚拟机：JVM高级特性与最佳实践（第3版）

判断对象是否存活

- 计数算法
  - 给每一个对象设置一个引用计数器，当有一个地方引用该对象的时候，引用计数器加1，引用失效时，引用计数器减1。当引用计数器为0的时候，就说明这个对象没有被引用，也就是说明对象，等待回收。
  - 缺点：无法解决循环引用的问题，当A引用B，而B引用A的时候，此时A和B彼此引用数不为0，此时就无法回收回收。所以一般主流虚拟机都不采用这个方法。
- 可达性分析法
  - JGC Roots(可作为GC根源，每个一个线程JGC Roots是任何可用线程时，说明此对象不可用
  - GC Roots可能分类
    - 虚拟机栈中引用的对象
    - 方法区类静态属性引用的变量
    - 方法区常量池引用的对象
    - 本地方法栈内JNI引用的对象

- 三色标记法
  - 白色：未可复达在被标记阶段的对象。
  - 灰色：已被标记，但是对象引用的属性对象还没有被全部访问。
  - 黑色：本对象已经被访问过，并且本对象引用的属性对象也被全部访问了。
  - 1. 初始状态，所有对象都是白色。
  - 2. 将 GC Roots 直接引用的对象标记为灰色。
  - 3. 从灰色对象出发，遍历其引用的对象，将这些对象也变成灰色。之后将当前灰色对象变成黑色。
  - 4. 重复步骤 3，直到不再有灰色对象为止。
  - 三色标记结束后，白色对象表示没有被引用的对象（比如上图中的 H 和 I），可以被回收了。

  满足上述条件时，不会当上GC，因要进行两次标记
  1. 判断当前对象是否不finalize()过并且没有实现finalize()，若不存在该方法，则标记为回收对象
  2. 将当前对象放入F-Queue队列，并生成一个finalize线程执行finalize()方法（虽然执行不保证该方法一定被执行，因为标准线程执行过程太重大了耗时，导致GC效率降低）
  3. 如果执行了finalize()的情况 GC Roots引用，则可重启GC(对象回收)

引用规则

- 强引用：普通的对象引用关系，如 String s = new String("Constalong");
- 软引用：内存不足时，GC才会回收，通过SoftReference实现
- 弱引用：GC时直接回收，通过WeakReference实现
- 虚引用：一种形同虚设的引用，基本用不到，通过PhantomReference实现

垃圾回收算法

- 标记清除法
  - 标记需要回收的对象，然后统一回收

  - 特点：效率低，容易产生内存碎片，影响分配大对象空间的效率
  - 特点：内存空间变短或断续，一段回收效用，一段正常使用
- 标记复制法
  - 特点：速度一个空间，不会产生内存碎片
- 标记整理法
  - 标记需要回收的对象，把所有存活的对象往一端移动，最后清除掉另一端

  - 特点：对于移动大对象，开销很大，不会产生内存碎片
- 分代收集算法
  - 根据不同的生命周期区域，使用不同的算法
  - 新生代使用复制算法
  - 老年代使用标记整理法
- 总结

算法	优点	缺点	适用场景
标记-清除	简单，不需要移动对象	产生内存碎片，分配大对象可能失败	内存碎片不敏感的场景
标记-复制	无内存碎片，分配效率高	内存利用率低（只用一半），频繁复制开销大	新生代（对象存活率低）
标记-整理	无内存碎片，内存利用率高	移动对象开销大，需要更多时间	老年代（对象存活率高）
分代收集	结合各算法优势，针对不同区域优化	实现复杂，需要维护多个区域	大多数现代JVM实现

概述及基础概念引入

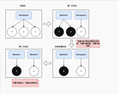
- 相关域等标示意图

  图 1.6 Hotspot 虚拟机垃圾收集器[27]

- 垃圾收集器相关版本的可以分为
  - 单行：并行收集的是多条垃圾收集器线程之间的关系，说明同一时间有多条这样的线程在协同工作。
  - 虚线等距式表时间产生线程是有子等待状态
  - 并发等单位
  - 并发：并发收集的是垃圾收集器线与用户线程之间的关系，说明同一时间垃圾收集器线程与用户线程都在运行。由于用户线程多未被冻结，所以程序的思维时间基本还是，但由于垃圾收集器线程占用了一部分系统资源，此时应用程序的吞吐量将会降低。
  - 此时应用程序的吞吐量将会降低，新增某部一定影响。

- JDK11引入，JDK15不在是实验功能正式投入
  - 三个重要特性特性
  - 内存非易失型
  - 染色指针
  - 内存布局
  - 读屏障

  - ZGC 初始化后，整个内存空间的地址视图被设置为 Remapped
- GC过程
  - 初始标记
    - 从 GC Roots 出发，找出 GC Roots 直接引用的对象，进入活跃对象集合。这个过程需要 STW，不过 STW 的时间跟 GC Roots 数量成正比，相对比较短
  - 并发标记
    - 并发标记过程中，GC 线程和 Java 应用线程会并行运行
    - GC 标记线程会同时操作，如果对象地址视图是 Remapped，就把对象地址视图切换到 Marked0，如果对象地址视图已是 Marked0，说明已经被其他标记线程标过了，跳过不处理。
    - 标记过程中Java 应用线程新创建的对象会直接进入 Marked0 视图。
    - 标记过程中Java 应用线程新创建的对象时，如果对象的地址视图是 Remapped，就把对象地址视图切换到 Marked0，可以参考前面讲的读屏障。
    - 标记结束后，原来对象地址视图是 Marked0，那就是活跃的，如果对象地址视图是 Remapped，那就是不活跃的。


Heap
Marked0
Marked1
Remapped
Object
Object
Object