# 垃圾收集算法

**1.引用计数器算法：**

引用计数器算法采用的是分散式管理方式，通过给每个对象设置一个计数器，当有地方引用这个对象的时候，计数器+1，当引用失效的时候，计数器-1，当计数器为0的时候，JVM就认为对象不再被使用，是“垃圾”了。

引用计数器实现简单，效率高；但是不能解决循环引用的问题（A对象引用B对象，B对象又引用A对象，但是A,B对象已不被任何其他对象引用），同时每次计数器的增加和减少都带来了很多额外的开销，所以在JDK1.1之后，这个算法已经不再使用了。

**2.根搜索方法：**

根搜索方法采用的是集中式的管理方式，全局记录对象之间的引用状态。执行时用一些“GC Roots”对象作为起点，从这些节点开始往下搜索，搜索通过的路径称为引用链（Reference Chain），当一个对象没有被GC Roots的引用链连接的时候，说明这个对象是不可用的。

GC Roots对象包括：

a) 虚拟机栈（栈帧中的本地变量表）中引用的对象。

b) 方法区中的类静态属性引用的对象。

c) 方法区中常量引用的对象。

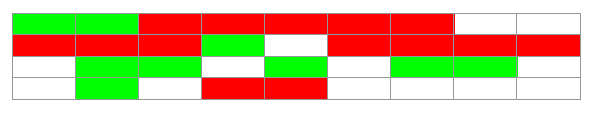
d) 本地方法栈中JNI（Native方法）引用的对象。

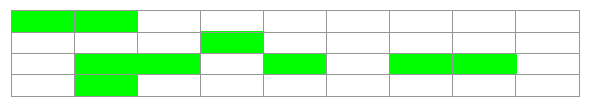
了解了JVM是怎么确定对象是“垃圾”之后，进入正题，让我们来看看垃圾回收的算法。

**2.1 标记—清除算法（Mark-Sweep）**

标记—清除算法包括两个阶段：“标记”和“清除”。在标记阶段，确定所有要回收的对象，并做标记。清除阶段紧随标记阶段，将标记阶段确定不可用的对象清除。

标记—清除算法是最基础的收集算法，标记和清除阶段的效率不高，而且清除后会产生大量的不连续内存，这样当程序需要分配大内存对象时，可能无法找到足够的连续空间。

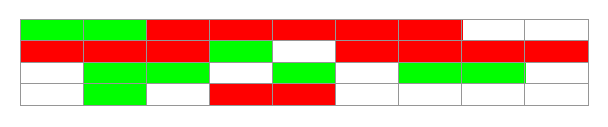
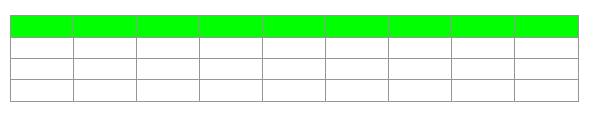
垃圾回收前：

垃圾回收后：绿色：存活对象 红色：可回收对象 白色：未使用空间

**2.2 复制算法（Copying）**

复制算法是把内存分成大小相等的两块，每次使用其中一块，当垃圾回收的时候，把存活的对象复制到另一块上，然后把这块内存整个清理掉。这样使得每次都是对其中的一块进行内存回收，不会产生碎片情况，只要移动堆顶的指针，按顺序分配内存即可。实现简单，运行高效。

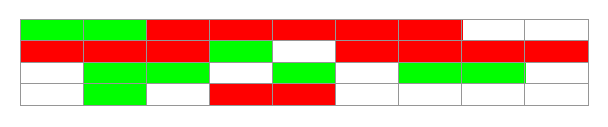
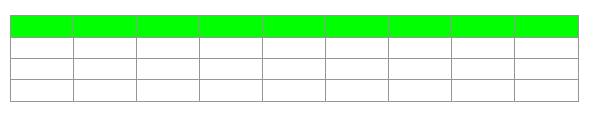
复制算法实现简单，运行效率高，但是由于每次只能使用其中的一半，造成内存的利用率不高。现在的JVM用复制方法收集新生代，由于新生代中大部分对象（98%）都是朝生夕死的，所以两块内存的比例不是1:1(大概是8:1)。

垃圾回收前：垃圾回收后：绿色：存活对象 红色：可回收对象 白色：未使用空间

**2.3 标记—整理算法（Mark-Compact）**

标记—整理算法和标记—清除算法一样，后续操作不只是直接清理对象，而是在清理无用对象完成后，让所有存活的对象都向一端移动，并更新引用其对象的指针。

标记—整理算法在标记—清除的基础上还需进行对象的移动，成本相对较高，好处则是不会产生内存碎片。它适合收集对象存活时间较长的老年代。

垃圾回收前：垃圾回收后：

绿色：存活对象 红色：可回收对象 白色：未使用空间

**2.4 分代收集（Generational Collection）**

分代收集是根据对象的存活时间把内存分为新生代和老年代，根据个代对象的存活特点，每个代采用不同的垃圾回收算法。新生代采用复制算法，老年代采用标记—整理算法。

垃圾算法的实现涉及大量的程序细节，而且不同的虚拟机平台实现的方法也各不相同。