## 一. 标记-清除算法：

**1. 概念：**

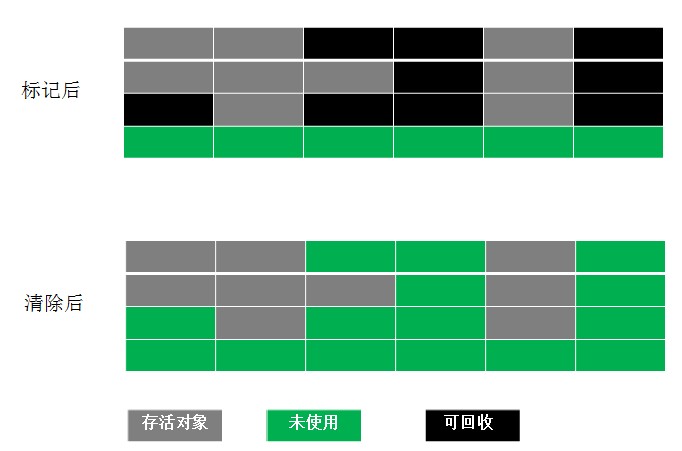
标记阶段：**先通过根节点，标记所有从根节点开始的可达对象**。因此，未被标记的对象就是未被引用的垃圾对象；

清除阶段：清除所有未被标记的对象。

**2. 缺点：**

标记和清除的过程**效率不高**（标记和清除都需要从头遍历到尾）

标记清除后**会产生大量不连续的碎片**。



## 二. 复制算法：（新生代的GC）

**1. 概念：**

　　将原有的**内存空间分为两块**，每次只使用其中一块，在垃圾回收时，将正在使用的内存中的**存活对象**复制到未使用的内存块中，然后清除正在使用的内存块中的所有对象。

**2. 优点：**

这样使得每次都是对整个半区进行回收，内存分配时也就**不用考虑内存碎片**等情况

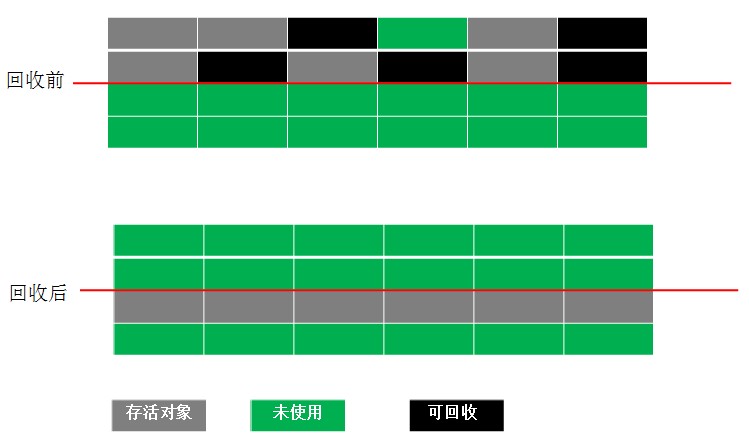
只要移动堆顶指针，按顺序分配内存即可，实现简单，**运行效率高**

**3. 缺点：空间的浪费**

　　从以上描述不难看出，复制算法要想使用，最起码对象的存活率要非常低才行。

　　现在的商业虚拟机都采用这种收集算法来回收新生代，新生代中的对象98%都是“朝生夕死”的，所以并不需要按照1:1的比例来划分内存空间，而是将**内存分为一块比较大的Eden空间和两块较小的Survivor空间**，每次使用Eden和其中一块Survivor。当回收时，将Eden和Survivor中还存活着的对象一次性地复制到另外一块Survivor空间上，最后清理掉Eden和刚才用过的Survivor空间。HotSpot虚拟机默认Eden和Survivor的大小比例是8:1，也就是说，每次新生代中可用内存空间为整个新生代容量的90%（80%+10%），只有10%的空间会被浪费。

当然，98%的对象可回收只是一般场景下的数据，我们没有办法保证每次回收都只有不多于10%的对象存活，**当Survivor空间不够用时，需要依赖于老年代进行分配担保，所以大对象直接进入老年代**。整个过程如下图所示：



## 三. 标记-整理算法：（老年代的GC）

    复制算法在对象**存活率高**的时候要进行较多的复制操作，效率将会降低，所以在老年代中一般不能直接选用这种算法。

**1. 概念：**

标记阶段：先通过根节点，标记所有从根节点开始的可达对象。因此，未被标记的对象就是未被引用的垃圾对象

整理阶段：**将将所有的存活对象压缩到内存的一端**；之后，清理边界外所有的空间

**2. 优点：**

　　不会产生内存碎片。

**3. 缺点：**

　　在标记的基础之上还需要进行对象的移动，成本相对较高，效率也不高。

它们的区别如下：（>表示前者要优于后者，=表示两者效果一样）

（1）效率：复制算法 > 标记/整理算法 > 标记/清除算法（此处的效率只是简单的对比时间复杂度，实际情况不一定如此）。

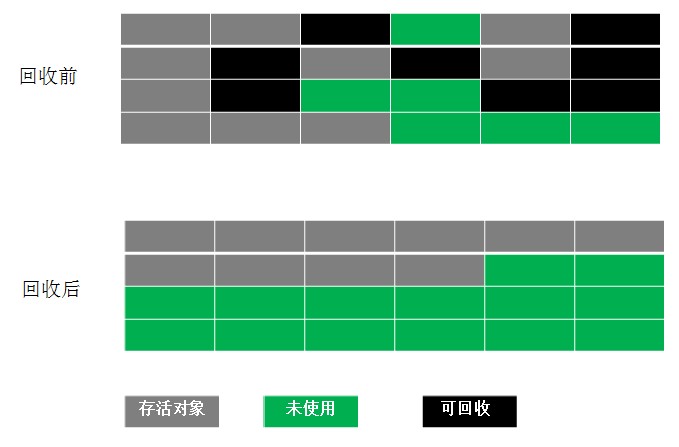
（2）内存整齐度：复制算法=标记/整理算法>标记/清除算法。

（3）内存利用率：标记/整理算法=标记/清除算法>复制算法。

注1：标记-整理算法不仅可以弥补标记-清除算法当中，内存区域分散的缺点，也消除了复制算法当中，内存减半的高额代价。

注2：可以看到标记/清除算法是比较落后的算法了，但是后两种算法却是在此基础上建立的。

注3：时间与空间不可兼得。



## 四. 分代收集算法：

　　当前商业虚拟机的GC都是采用的“分代收集算法”，这并不是什么新的思想，**只是根据对象的存活周期的不同将内存划分为几块儿**。一般是把Java堆分为新生代和老年代：**短命对象归为新生代，长命对象归为老年代**。

* **存活率低：少量对象存活，适合复制算法**：在新生代中，每次GC时都发现有大批对象死去，只有少量存活（新生代中98%的对象都是“朝生夕死”），那就选用复制算法，只需要付出少量存活对象的复制成本就可以完成GC。
* 存活率高：大量对象存活，适合用标记-清理/标记-整理：在老年代中，因为对象存活率高、没有额外空间对他进行分配担保，就必须使用“标记-清理”/“标记-整理”算法进行GC。

注：**老年代的对象中，有一小部分是因为在新生代回收时，老年代做担保，进来的对象；绝大部分对象是因为很多次GC都没有被回收掉而进入老年代**。