**JVM垃圾回收算法**

## 判断Java中对象存活的算法

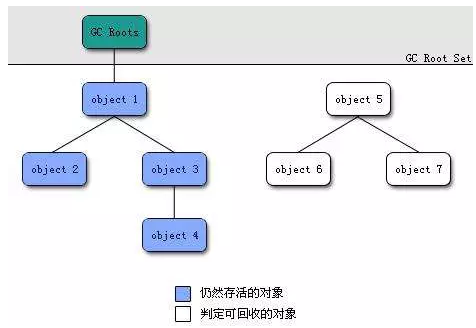
##### 1.引用计数器算法：

引用计数器算法是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问问题（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：

根搜索方法是通过一些“GCRoots”对象作为起点，从这些节点开始往下搜索，搜索通过的路径成为引用链（ReferenceChain），当一个对象没有被GCRoots的引用链连接的时候，说明这个对象是不可用的。  
GCRoots对象包括：

1. 虚拟机栈（栈帧中的本地变量表）中的引用的对象。
2. 方法区域中的类静态属性引用的对象。
3. 方法区域中常量引用的对象。
4. 本地方法栈中JNI（Native方法）的引用的对象。

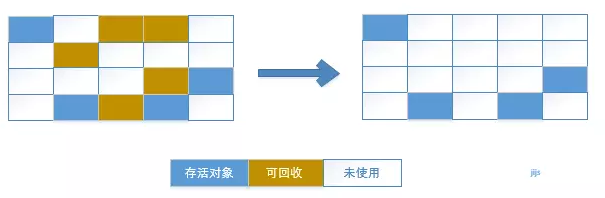


从图中可以看出，object5、object6、object7 对象是没有GC-Roots 引用（间接引用），因此，这三个对象可以被垃圾收集器回收。

## JVM垃圾回收算法

#### 1. 标记-清除算法

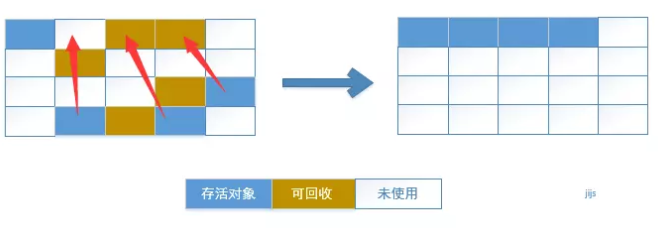
标记-清除（Mark-Sweep）算法是现代垃圾回收算法的思想基础。标记-清除算法将垃圾回收分为两个阶段：标记阶段和清除阶段。一种可行的实现是，在标记阶段，首先通过根节点，标记所有从根节点开始的可达对象。因此，未被标记的对象就是未被引用的垃圾对象（好多资料说标记出要回收的对象，其实明白大概意思就可以了）。然后，在清除阶段，清除所有未被标记的对象。  
如图：



缺点：  
1、效率问题，标记和清除两个过程的效率都不高；  
2、空间问题，标记清除之后会产生大量不连续的内存碎片，空间碎片太多可能会导致以后在程序运行过程中需要分配较大的对象时，无法找到足够的连续内存而不得不提前触发另一次垃圾收集动作。

#### 2. 标记整理算法

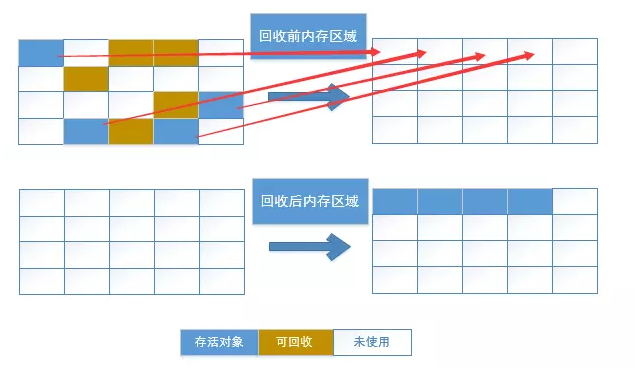
标记整理算法类似与标记清除算法，不过它标记完对象后，不是直接对可回收对象进行清理，而是让所有存活的对象都向一端移动，然后直接清理掉边界以外的内存。



缺点：  
1、效率问题，（同标记清除算法）标记和整理两个过程的效率都不高；  
优点：  
1、相对标记清除算法，解决了内存碎片问题。  
2、没有内存碎片后，对象创建内存分配也更快速了（可以使用TLAB进行分配）。

#### 3. 复制算法

复制算法可以解决效率问题，它将可用内存按容量划分为大小相等的两块，每次只使用其中的一块，当这一块内存用完了，就将还存活着的对象复制到另一块上面，然后再把已经使用过的内存空间一次清理掉，这样使得每次都是对整个半区进行内存回收，内存分配时也就不用考虑内存碎片等复杂情况，只要移动堆顶指针，按顺序分配内存即可（还可使用TLAB进行高效分配内存）。

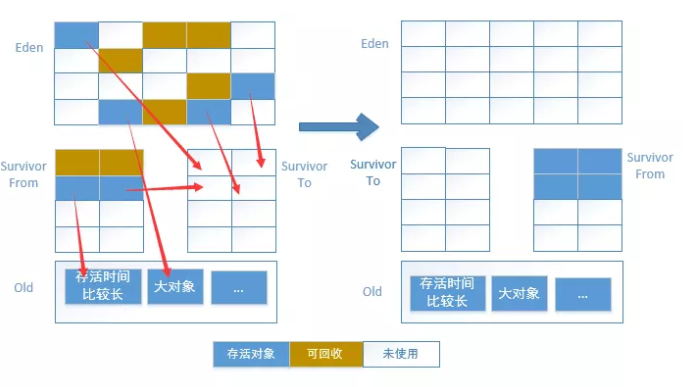


图的上半部分是未回收前的内存区域，图的下半部分是回收后的内存区域。通过图，我们发现不管回收前还是回收后都有一半的空间未被利用。

优点  
效率高，没有内存碎片  
缺点：  
1、浪费一半的内存空间  
2、复制收集算法在对象存活率较高时就要进行较多的复制操作，效率将会变低。.

#### 4. 分代收集算法

当前商业虚拟机都是采用分代收集算法，它根据对象存活周期的不同将内存划分为几块，一般是把Java堆分为新生代和老年代，然后根据各个年代的特点采用最适当的收集算法，在新生代中，每次垃圾收集都发现有大批对象死去，只有少量存活，就选用复制算法，而老年代因为对象存活率高，没有额外空间对它进行分配担保，就必须使用“标记清理”或者“标记整理”算法来进行回收。



图的左半部分是未回收前的内存区域，右半部分是回收后的内存区域。

**对象分配策略：**

1. 对象优先在Eden区域分配，如果对象过大直接分配到Old区域。
2. 长时间存活的对象进入到Old区域。

**改进复制算法：**  
现在的商业虚拟机都采用这种收集算法来回收新生代，IBM公司的专门研究表明，新生代中的对象98%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%的内存会被“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，需要依赖其他内存（这里指老年代）进行分配担保（Handle Promotion）。