关于jvm垃圾回收那些事（一）.md

# 关于jvm垃圾回收那些事（一）

## 对象真的死了吗？

垃圾回收顾名思义就是回收垃圾，对于我们java对象来说话，如果一个对象不存在任何引用，那么我们就可以说这个对象死了，也就是说这个对象是垃圾，可是这个对象是否死亡是怎么评定的呢？

### 引用计数器算法

在很多地方判断对象是否存活的算法是这样的，给对象中添加一个引用计数器，每当有一个地方引用它时便加1；当这个引用失效时就减1。如果这个计数器为0的时候，那就是代表这个对象已经没有任何引用了，也就是可以称之为垃圾，此对象已死。

很多人被问到jvm如何判定对象是否存活的时候都会这样回答，但其实主流的虚拟机并没有使用引用计数器算法来管理内存，原因主要是因为很难解决对象之间存在循环引用的问题。

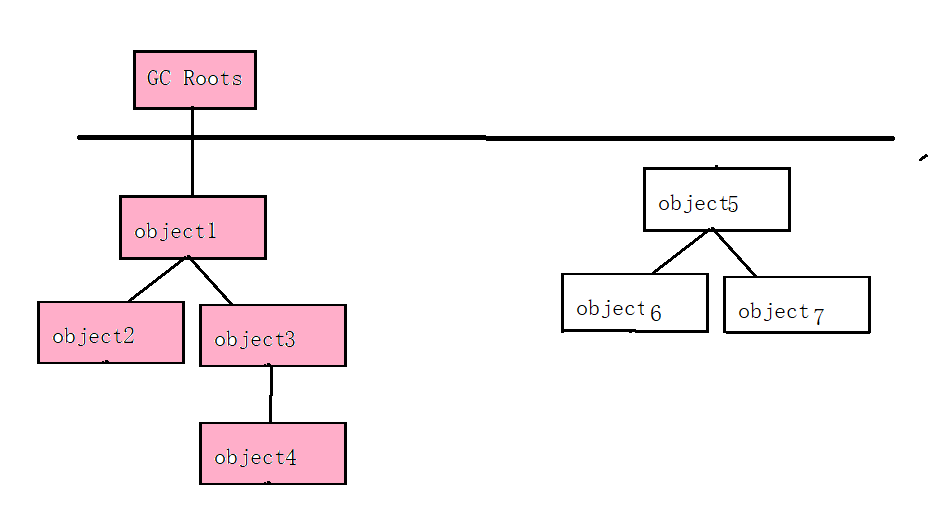
public class Test{  
 public Object obj ;   
   
 public void testGc(){  
 Test a = new Test();  
 Test b = new Test();  
   
 a.obj = B;  
 b.obj = A;  
   
 a = null;  
 b = null;  
   
 System.gc();  
 }  
}

以上代码发生GC，a和b是否被垃圾回收呢？

显然是不会的，因为这两个对象已经没有了任何引用，但因为他们互相引用对方，他们的计数器都不为0，所以无法通知垃圾收集器回收它们。通过以上代码我们推测出，虚拟机并不是通过引用计数器这种算法来判定对象是否依然存活的。

### 可达性分析算法

在主流虚拟机中都是使用这种算法来判定对象是否存活，这种算法的基本思想：



* 粉色：仍然存活的对象
* 白色：判定可回收的对象

通过称为GC Roots的对象作为起始点，开始向下搜索，搜索的路径称为引用链，从GC Roots到这个对象不可达的时候，证明此对象是不可用的。

在java当中，可以作为GC Roots的对象如下几种：

* 虚拟机栈中引用的对象。
* 方法区中类静态属性引用的对象。
* 方法区常量引用的对象。
* Native方法引用的对象。

### 再聊聊引用

无论通过引用计数器判断对象的引用数量还是可达性分析算法判断对象的引用链，都离不开引用。

自从JDK1.2之后，java将引用分为：

* 强引用：比如Object obj = new Object();这种引用，只要强引用还在，垃圾收集器永远不会回收掉被引用的对象。
* 软引用：用来描述一些有用但非必须的对象，在发生内存溢出之前会把此类对象进行二次回收，如果此时还没足够的内存，才会抛出内存溢出。java提供了SoftReference类来实现软引用。
* 弱引用：同样也是用来描述非必需的对象，但是它的强度比软引用弱一些，在发生垃圾回收的时候无论内存是否够用，都会回收调弱引用的对象。java中提供了WeakReference类来实现弱引用。
* 虚引用：是最弱的一种引用关系。一个对象是否有虚引用的存在，完全不会对其生存时间构成影响，同样也无法通过虚引用来获取一个实例。当一个对象设置为虚引用时，唯一的目的就是为了在这个对象被垃圾回收的时候收到一条通知。java中提供了PhantomReference类来实现虚引用。

## 垃圾回收算法

### 分代收集

首先要声明的是当前虚拟机大都采用了分代收集的理论。

1. 弱分代假说：绝大多数对象都是朝生夕灭的。
2. 强分代假说：熬过越多次垃圾收集过程的对象就越难以消亡。

所谓的分代收集也就是我们常说的新生代和老年代。

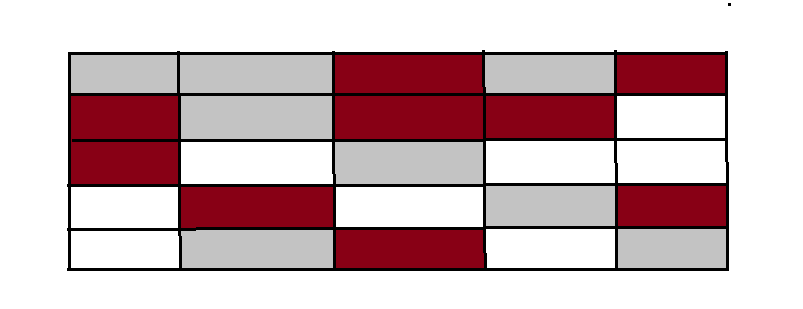
在新生代中每次垃圾回收时都会有大量的对象死去，每次回收后存活的对象经过多次新生代的垃圾回收依然还存活的话晋升到老年代。

### 标记-清除

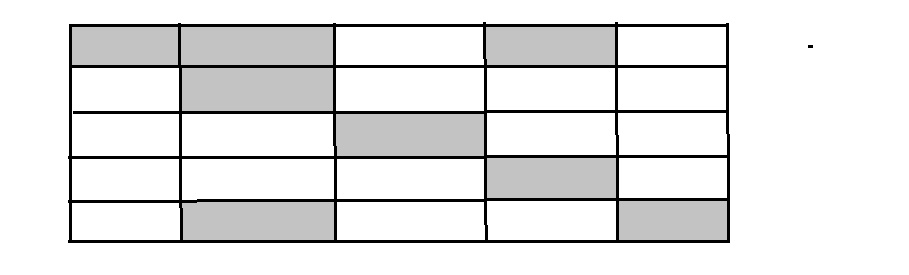
标记清除是最早的也是最基础的垃圾收集算法，标记清除顾名思义就是先将内存中的垃圾依次标记出来，然后对已标记的对象进行统一的垃圾回收，从而达到释放内存的效果。

缺点：

* 执行效率不稳定，如果内存中有大量的对象，其中大部分对象都是需要回收的，那此时就要对很多对象进行标记，就会导致标记和清除的执行时间随着对象的增加而增加。
* 会产生内存碎片，标记清除之后会产生大量的不连续的内存碎片，由于空间碎片过多，可能会导致在之后大对象需要分配空间时，发现内存不够，触发full GC。导致性能下降。



垃圾回收之前



垃圾回收之后

* 灰色：存活对象
* 红色：可回收内存
* 白色：空闲内存

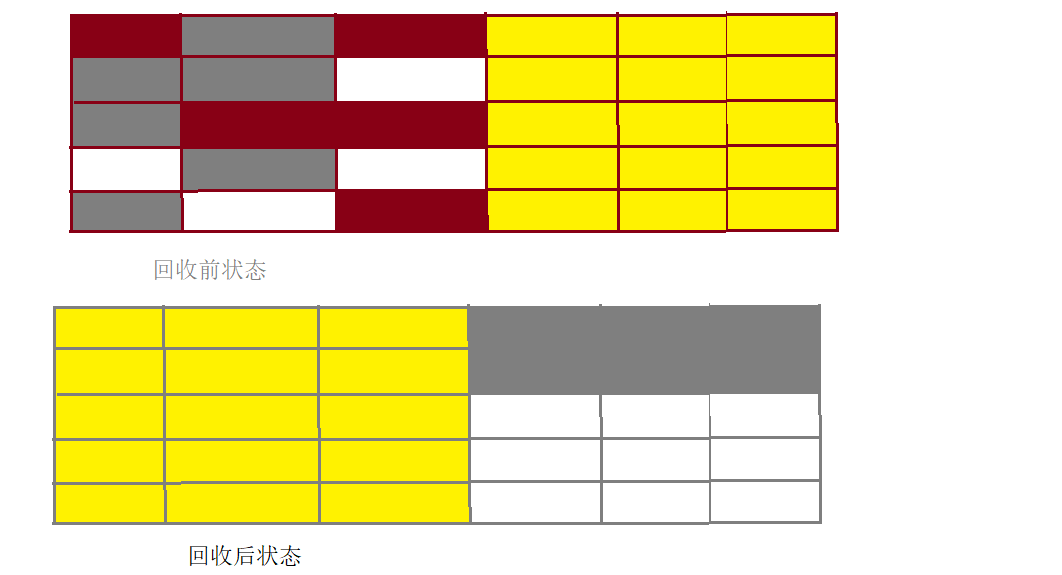
### 标记-复制

标记复制的出现是为了解决标记清除面对大量可回收对象执行效率低的问题。

它将可用的内存分为大小相等的两块，每次使用其中一块。当一块用完了，就将存活的对象复制到另一块上，然后把之前使用过的另一块清理掉。如果内存中大量对象都是可回收的，那需要复制的就是少量的对象。每次回收都是针对整个半区，也就不用考虑内存碎片的问题了。

缺点：

* 如果内存中大量对象都是存活的，这种算法就会产生大量的内存复制。
* 这种算法相当于牺牲了一般的内存空间，对于内存这种宝贵的资源来说，这种代价有点太大了。



* 红色：可回收内存
* 白色：空闲内存
* 灰色：存活对象
* 黄色：预留内存

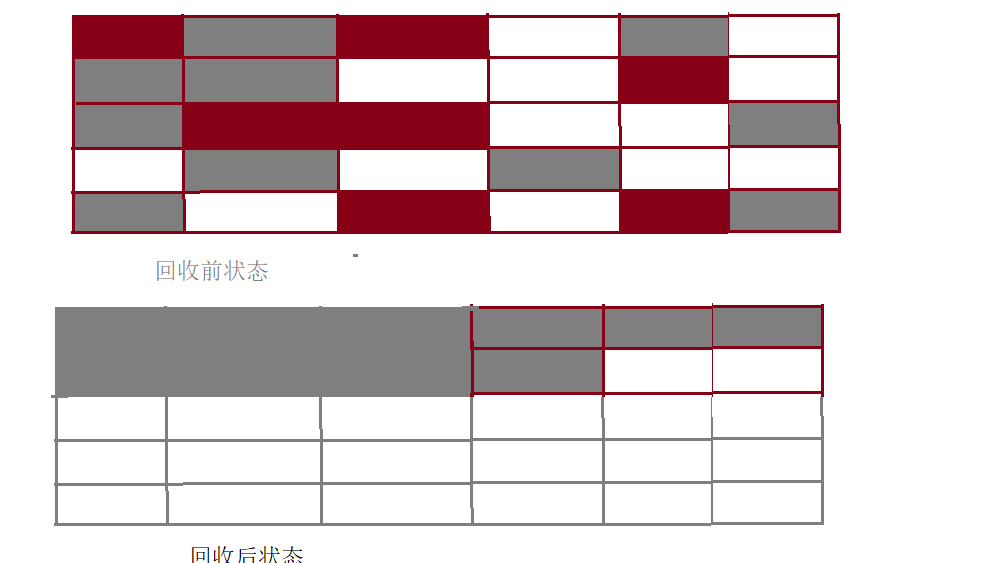
值得一提的是现在大多数虚拟机新生代都采用标记复制的算法，因为新生代的对象都是朝生夕死的，只需要复制少量的对象就可以了。

### 标记- 整理

针对老年代的存亡特征，标记过程与标记清除一样，接下让所有的对象都向内存空间的一端移动，然后直接清理掉边界以外的内存。

缺点：

* 在老年代大量对象存活的地方移动对象的时候，必须要暂停所有的用户线程，也就是我们所说的full GC



* 灰色：存活对象
* 红色：可回收对象
* 白色：空闲内存

下次会继续分享基于垃圾回收算法实现的垃圾收集器。