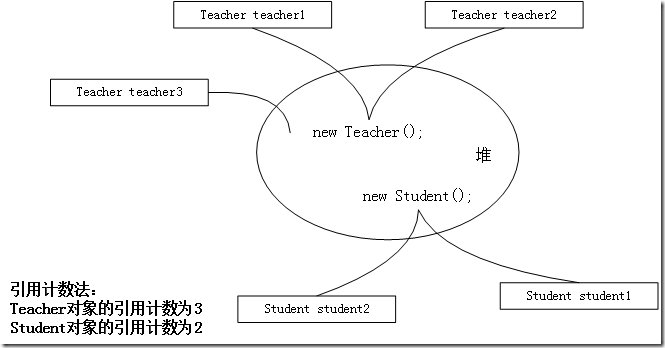
深入理解JVM（四）—垃圾回收算法

1. **stop the world**

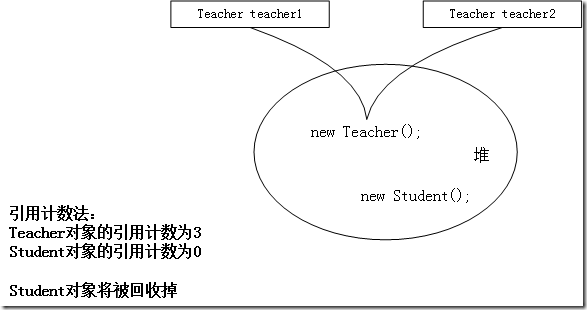
在介绍垃圾回收算法之前，我们需要先了解一个词“stop the world”，stop the world会在执行某一个垃圾回收算法的时候产生，JVM为了执行垃圾回收，会暂时java应用程序的执行，等垃圾回收完成后，再继续运行。如果你使用JMeter测试过java程序，你可能会发现在测试过程中，java程序有不规则的停顿现象，其实这就是“stop the world”，停顿的时候JVM是在做垃圾回收。所以尽可能减少stop the world的时间，就是我们优化JVM的主要目标。接下来我们看一下目前有哪些常见垃圾回收的算法。

1. **引用计数法**

引用计数法顾名思义，就是对一个对象被引用的次数进行计数，当增加一个引用计数就加1，减少一个引用计数就减1。



上图表示3个Teacher的引用指向堆中的Teacher对象，那么Teacher对象的引用计数就是3，以此类推Student对象的引用计数就是2。



上图表示Teacher对象的引用减少为2，Student对象的引用减少为0（减少的原因是该引用指向了null，例如teacher3=null）,按照引用计数算法，Student对象的内存空间将被回收掉。

引用计数算法原理非常简单，是最原始的回收算法，但是java中没有使用这种算法，原因有2。1是频繁的计数影响性能，2是它无法处理循环引用的问题。

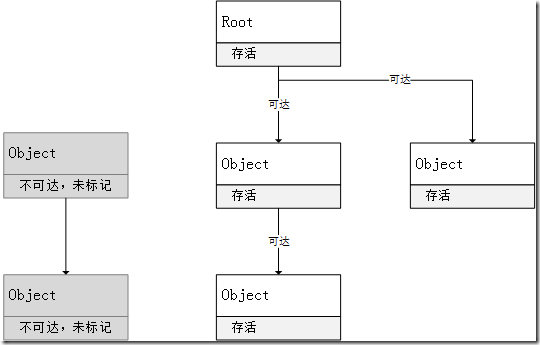
例如Teacher对象中引用了Student对象，Student对象中又引用了Teacher对象，这种情况下，对象将永远无法被回收。

1. **标记清除**

标记清除算法，它是很多垃圾回收算法的基础，简单来说有两个步骤：标记、清除。

标记：遍历所有的GC Roots，并将从GC Roots可达的对象设置为存活对象；

清除：遍历堆中的所有对象，将没有被标记可达的对象清除；



注意上图灰色的对象，因为从GC Root遍历不到它们（尽管它们本身有引用关系，但从GC Root无法遍历到它们），因此它们没有被标记为存活对象，在清除过程中将会被回收。

这里需要注意的是标记清除算法执行过程中，会产生“stop the world”，让java程序暂停等待以保证在标记清除的过程中，不会有新的对象产生。为什么必须暂停java程序呢？举个例子，如果在标记过程完成后，又新产生了一个对象，而该对象已经错过了标记期，那么在接下来的清除流程中，这个新产生的对象因为未被标记，所以将被视为不可达对象而被清除，这样程序就会出错，因此标记清除算法在执行时，java程序将被暂停，产生“stop the world”。

接下来我们总结一下标记清除算法：

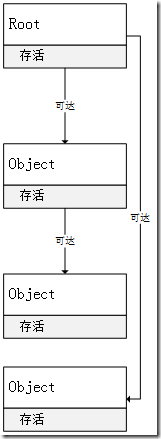
1、因为涉及大量的内存遍历工作，所以执行性能较低，这也会导致“stop the world”时间较长，java程序吞吐量降低；

2、我们注意到对象被清除之后，被清除的对象留下内存的空缺位置，造成内存不连续，空间浪费。

接下来我们看一下其他算法能不能改善这些问题？

1. **标记压缩**

标记压缩算法你可能已经想到了，它就是在标记清除算法的基础上，增加了压缩过程。

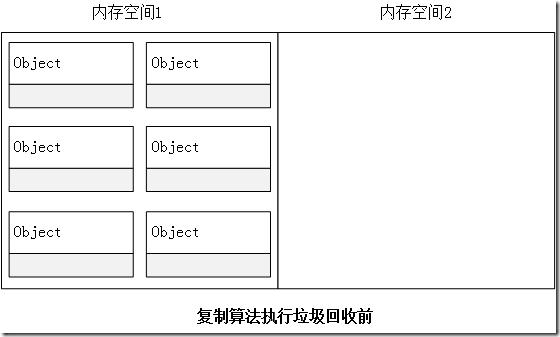


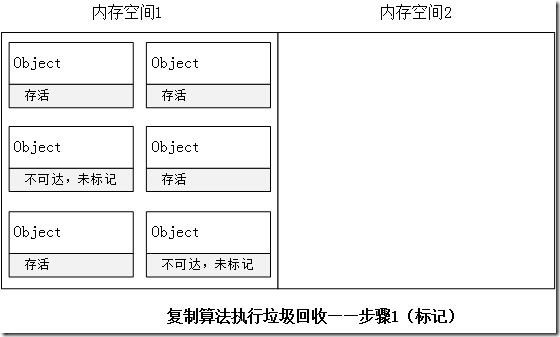
在进行完标记清除之后，对内存空间进行压缩，节省内存空间，解决了标记清除算法内存不连续的问题。

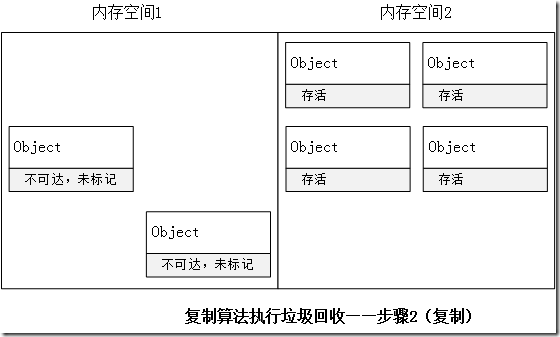
注意标记压缩算法也会产生“stop the world”，不能和java程序并发执行。在压缩过程中一些对象内存地址会发生改变，java程序只能等待压缩完成后才能继续。

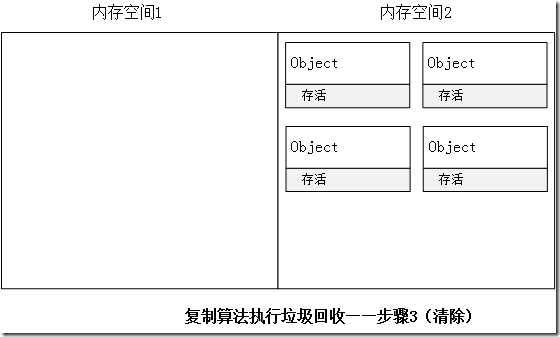
1. **复制算法**

复制算法简单来说就是把内存一分为二，但只使用其中一份，在垃圾回收时，将正在使用的那份内存中存活的对象复制到另一份空白的内存中，最后将正在使用的内存空间的对象清除，完成垃圾回收。









复制算法相对标记压缩算法来说更简洁高效，但它的缺点也显而易见，它不适合用于存活对象多的情况，因为那样需要复制的对象很多，复制性能较差，所以复制算法往往用于内存空间中新生代的垃圾回收，因为新生代中存活对象较少，复制成本较低。它另外一个缺点是内存空间占用成本高，因为它基于两份内存空间做对象复制，在非垃圾回收的周期内只用到了一份内存空间，内存利用率较低

1. **小结**

以上我们介绍了常见的垃圾回收算法，这些算法各有各的优缺点，但在JVM中并不是单纯的使用特定的算法，而是使用的一种叫垃圾回收器的东西，垃圾回收器可以看做一系列算法的不同组合，在不同的场景使用合适的垃圾回收器，才能起到事半功倍的效果。我们下一篇将介绍垃圾回收器。

1. **来源**

http://www.cnblogs.com/leefreeman/p/7389919.html