

JAVA基础：Java堆的管理?垃圾回收 PDF转换可能丢失图片或格式，建议阅读原文

https://www.100test.com/kao_ti2020/145/2021_2022_JAVA_E5_9F_BA_E7_A1_80_c104_145259.htm

1 引言 Java的堆是一个运行时数据区,类的实例(对象)从中分配空间。Java虚拟机(JVM)的堆中储存着正在运行的应用程序所建立的所有对象，这些对象通过new、newarray、anewarray和multianewarray等指令建立，但是它们不需要程序代码来显式地释放。一般来说，堆的是由垃圾回收来负责的，尽管JVM规范并不要求特殊的垃圾回收技术，甚至根本就不需要垃圾回收，但是由于内存的有限性，JVM在实现的时候都有一个由垃圾回收所管理的堆。垃圾回收是一种动态存储管理技术，它自动地释放不再被程序引用的对象，按照特定的垃圾收集算法来实现资源自动回收的功能。

2 垃圾收集的意义 在C中，对象所占的内存在程序结束运行之前一直被占用，在明确释放之前不能分配给其它对象；而在Java中，当没有对象引用指向原先分配给某个对象的内存时，该内存便成为垃圾。JVM的一个系统级线程会自动释放该内存块。垃圾收集意味着程序不再需要的对象是"无用信息"，这些信息将被丢弃。当一个对象不再被引用的时候，内存回收它占领的空间，以便空间被后来的新对象使用。事实上，除了释放没用的对象，垃圾收集也可以清除内存记录碎片。由于创建对象和垃圾收集器释放丢弃对象所占的内存空间，内存会出现碎片。碎片是分配给对象的内存块之间的空闲内存洞。碎片整理将所占用的堆内存移到堆的一端，JVM将整理出的内存分配给新的对象。垃圾收集能自动释放内存空间，减轻编程的负担。这使Java虚拟机具有一些优

点。首先，它能使编程效率提高。在没有垃圾收集机制的时候，可能要花许多时间来解决一个难懂的存储器问题。在用Java语言编程的时候，靠垃圾收集机制可大大缩短时间。其次是它保护程序的完整性，垃圾收集是Java语言安全性策略的一个重要部份。垃圾收集的一个潜在的缺点是它的开销影响程序性能。Java虚拟机必须追踪运行程序中有用的对象，而且最终释放没用的对象。这一个过程需要花费处理器的时间。其次垃圾收集算法的不完备性，早先采用的某些垃圾收集算法就不能保证100%收集到所有的废弃内存。当然随着垃圾收集算法的不断改进以及软硬件运行效率的不断提升，这些问题都可以迎刃而解。

3 垃圾收集的算法分析

Java语言规范没有明确地说明JVM使用哪种垃圾回收算法，但是任何一种垃圾收集算法一般要做2件基本的事情：（1）发现无用信息对象；（2）回收被无用对象占用的内存空间，使该空间可被程序再次使用。大多数垃圾回收算法使用了根集(root set)这个概念；所谓根集就是正在执行的Java程序可以访问的引用变量的集合(包括局部变量、参数、类变量)，程序可以使用引用变量访问对象的属性和调用对象的方法。垃圾收集首选需要确定从根开始哪些是可达的和哪些是不可达的，从根集可达的对象都是活动对象，它们不能作为垃圾被回收，这也包括从根集间接可达的对象。而根集通过任意路径不可达的对象符合垃圾收集的条件，应该被回收。下面介绍几个常用的算法。

3.1 引用计数法(Reference Counting Collector)

引用计数法是唯一没有使用根集的垃圾回收得法，该算法使用引用计数器来区分存活对象和不再使用的对象。一般来说，堆中的每个对象对应一个引用计数器。当每一次创建一个对象并赋

给一个变量时，引用计数器置为1。当对象被赋给任意变量时，引用计数器每次加1。当对象出了作用域后(该对象丢弃不再使用)，引用计数器减1，一旦引用计数器为0，对象就满足了垃圾收集的条件。基于引用计数器的垃圾收集器运行较快，不会长时间中断程序执行，适宜地必须实时运行的程序。但引用计数器增加了程序执行的开销，因为每次对象赋给新的变量，计数器加1，而每次现有对象出了作用域生，计数器减1。

3.2 tracing算法(Tracing Collector)

tracing算法是为了解决引用计数法的问题而提出，它使用了根集的概念。基于tracing算法的垃圾收集器从根集开始扫描，识别出哪些对象可达，哪些对象不可达，并用某种方式标记可达对象，例如对每个可达对象设置一个或多个位。在扫描识别过程中，基于tracing算法的垃圾收集也称为标记和清除(mark-and-sweep)垃圾收集器。

3.3 compacting算法(Compacting Collector)

为了解决堆碎片问题，基于tracing的垃圾回收吸收了Compacting算法的思想，在清除的过程中，算法将所有的对象移到堆的一端，堆的另一端就变成了一个相邻的空闲内存区，收集器会对它移动的所有对象的所有引用进行更新，使得这些引用在新的位置能识别原来的对象。在基于Compacting算法的收集器的实现中，一般增加句柄和句柄表。

3.4 coping算法(Coping Collector)

该算法的提出是为了克服句柄的开销和解决堆碎片的垃圾回收。它开始时把堆分成一个对象面和多个空闲面，程序从对象面为对象分配空间，当对象满了，基于coping算法的垃圾收集就从根集中扫描活动对象，并将每个活动对象复制到空闲面(使得活动对象所占的内存之间没有空闲洞)，这样空闲面变成了对象面，原

来的对象面变成了空闲面，程序会在新的对象面中分配内存。一种典型的基于coping算法的垃圾回收是stop-and-copy算法，它将堆分成对象面和空闲区域面，在对象面与空闲区域面的切换过程中，程序暂停执行。

3.5 generation算法(Generational Collector) stop-and-copy垃圾收集器的一个缺陷是收集器必须复制所有的活动对象，这增加了程序等待时间，这是coping算法低效的原因。在程序设计中有这样的规律：多数对象存在的时间比较短，少数的存在时间比较长。

因此，generation算法将堆分成两个或多个，每个子堆作为对象的一代(generation)。由于多数对象存在的时间比较短，随着程序丢弃不使用的对象，垃圾收集器将从最年轻的子堆中收集这些对象。在分代式的垃圾收集器运行后，上次运行存活下来的对象移到下一最高代的子堆中，由于老一代的子堆不会经常被回收，因而节省了时间。

3.6 adaptive算法(Adaptive Collector) 在特定的情况下，一些垃圾收集算法会优于其它算法。基于Adaptive算法的垃圾收集器就是监控当前堆的使用情况，并将选择适当算法的垃圾收集器。

4 透视Java垃圾回收 4.1 命令行参数透视垃圾收集器的运行使

用System.gc()可以不管JVM使用的是哪一种垃圾回收的算法，都可以请求Java的垃圾回收。在命令行中有一个参

数-verbosegc可以查看Java使用的堆内存的情况，它的格式如

下：`java -verbosegc classfile` 可以看个例子：

```
class TestGC {  
public static void main(String[] args) { new TestGC(). System.gc().  
System.runFinalization(). } }
```

在这个例子中，一个新的对象被创建，由于它没有使用，所以该对象迅速地变为可达，程序编译后，执行命令：`java -verbosegc TestGC` 后结果为：`[Full GC`

168K->97K(1984K), 0.0253873 secs] 机器的环境为 , Windows 2000 JDK1.3.1,箭头前后的数据168K和97K分别表示垃圾收集GC前后所有存活对象使用的内存容量 , 说明有 $168K-97K=71K$ 的对象容量被回收 , 括号内的数据1984K为堆内存的总容量 , 收集所需要的时间是0.0253873秒 (这个时间在每次执行的时候会有所不同)。 100Test 下载频道开通 , 各类考试题目直接下载。详细请访问 www.100test.com