Java 垃圾回收新算法刍探

张鹏飞[1]，钱 敏[2]

（1. 中国矿业大学 计算机科学与技术学院，江苏 徐州 221116；
2. 中国矿业大学 管理学院，江苏 徐州 221116）

摘 要：由 Java 语言与 C/C++对象在内存管理方式的不同，引出了 Java 语言的优势技术——垃圾处理技术。通过对 GC 工作原理的阐述及对一些传统的垃圾收集器的分析，提出了一种新的垃圾处理算法，一定程度上改善和提高了 Java 垃圾处理的性能。

关键词：Java；GC；垃圾回收；收集器

中图分类号：TP312 **文献标识码**：A **文章编号**：1674-7720(2011)02-0015-03

Java preliminary probe into the new garbage collection algorithm

Zhang Pengfei[1], Qian Min[2]

(1. School of Computer Science and Technologe, China University of Mining and Technology, Xuzhou 221116, China;
2. School of Management, China University of Mining and Technology, Xuzhou 221116, China)

Abstract: This article from the Java language and C/C++ memory management of objects in different ways, leads to the advantage of Java language technology-Waste Technology. GC works on the elaboration and the traditional analysis of the refuse collector, a new waste treatment method, to some extent to improve and enhance the performance of Java garbage disposal.

Key words: Java; GC; garbage collection; collector

Java 从诞生以来以其在网络应用开发上独特的魅力以及“一次开发，随处运行”的可移植性引起了人们极大的兴趣。Java 与以往的高级语言如 C/C++相比，在开发方面具有很大的优势，其中以对象内存管理机制中的垃圾处理机制(GC)最为突出。

1 C/C++与 Java 对象内存管理差别

C/C++将内存划分成四部分：数据区、代码区、栈区、堆区。Java 则把内存划分成三部分即代码区、栈区、堆区，代码区主要用于存放程序的代码，栈区主要用于存放局部变量、内部变量等中间性变量，堆区主要用来存放对象。

C/C++中的对象内存管理是通过语句 new()/delete()或 malloc()/free()进行申请和释放的。用 new()或 malloc()申请内存后，若不使用 delete()或 free()进行释放，则所申请的内存一直被占用，即使不使用也不能自动释放，必须人为释放，导致编程工作很繁琐。

Java 中的对象内存管理则改进了对内存的释放过程，使用 new()或其他方法申请的内存在不使用时，可以自动进行垃圾处理，释放内存，从而节省内存，使内存的使用更加高效、合理。

Java 中可以通过三种方法来销毁对象实现内存释放，这三种方法被称为 Java 销毁对象的三把利剑：垃圾回收器；finalize 方法；利用 System.gc 方法强制启动垃圾回收器。

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实现资源自动回收的功能系统，会自动进行 GC 策略。

2 GC 概述

Java 垃圾处理主要是针对堆的管理，对堆中不使用的空间进行回收处理。判断一个对象的内存空间是否无用的标准是：如果该对象不能再被程序中任何一个“活动的部分”所引用，此时该对象的内存空间已经无用。所谓“活动的部分”，是指程序中某部分参与程序的调用，正在执行过程中，尚未执行完毕。

具体以下两例予以说明：

实例 1：

```
int [][]matrix=new int [2][3];
matrix=null;
```

此例中，第一句是用 new 语句在堆中为数组申请了一个空间，然后用 matrix 来引用此空间的对象（这里数

组可以理解为对象),此时这个内存空间就是有用的。第二句是给 matrix 赋空值,matrix 则不再引用此数组。此时,这个空间就是无用的。

实例 2:

```
int [][] m1=new int[2][3];
int [][] m2=new int[2][3];
m1=m2;
```

此例中,第一句是用 new 语句在堆中为数组申请了一个空间,用 m1 引用这个数组。第二句是用 new 语句在堆中为数组申请了一个空间,用 m2 引用这个数组。第三句是改变引用对象,把 m2 的引用赋给 m1,则此时 m1 也引用 m2 引用的对象数组,因此,m1 原来引用的数组无人引用,成为垃圾。

3 传统收集器简介

Java 依托于垃圾收集 GC 机制,可以自动回收垃圾即释放堆空间,让其他对象可以使用此部分空间。而采用了某种 GC 算法的收集器(Collector)称之为某某垃圾收集器(Garbage Collector)。目前 Java 中采用的垃圾收集器一般包括:引用计数法(Reference Counting Collector)、Tracing 算法(Tracing Collector)、Compacting 算法(Compacting Collector)、Coping 算法(Coping Collector)、Generation 算法(Generational Collector)、Adaptive 算法(Adaptive Collector)。

3.1 引用计数法

引用计数法是唯一没有使用根集的垃圾回收的方法,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,必须适宜地实时运行的程序。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,计数器减 1。

3.2 Tracing 算法

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

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 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

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

在特定的情况下,一些垃圾收集算法会优于其他算法。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 GC 新算法概述

基于上述对几种收集器算法优缺点的对比分析,提出一种既可以满足程序对实时性的要求,同时也能避免内存泄漏的较完全的垃圾处理算法。

4.1 算法描述

4.1.1 内存划分

(1)把一个堆内存划分成两大块,一块是活跃区,占堆大小的 2/3,活跃区划分成大小相同的 8 个块,并且为每个块设置一个计数器 int cnt[x](x 取 1~8),用来记录每块内存中动态分配的被引用对象总数。另一块是保留区,占堆大小的 1/3,设置两个常数 min 和 max,分别用来表示保留区的初始大小和最大可增加到的大小。(min 的值小于堆大小的 1/3,max 的值可以自行设置但最大值不超过堆大小的 1/3)。

(2)为堆中的每个对象设置一个标记位(标记位放在一个专用数组 cnt [x](x 取 1~8) 中)以表示其是否被引用,在对象被引用时计数器就开始动态地统计计数,记录本块中被引用的对象个数。

(3) 通过比较计数器值的大小来判断应该扫描哪个块，而不是利用搜索所有对象的方法。具体结构如图 1。

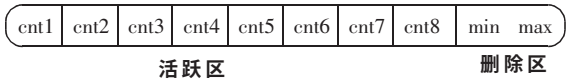

图 1 堆内存结构图

4.1.2 具体算法

(1)程序开始运行，对象动态地申请堆区，这时，每个块的计数器启动，根据引用对象的标记位情况来改变计数器的值，为 1 则计数器加 1，8 个计数器值放在数组里，并比较 8 个计数器值的大小，选取最大的计数器所在块，进行扫描。

(2)扫描出的垃圾转移到删除区，等待被删除。

(3)继续比较计数器值，但已经进行扫描的块不参加此后的比较，待删除的垃圾占的空间达到 min 值时，垃圾器开始对活跃区压缩内存碎片，并且在删除区同时开始进行垃圾删除申请。

(4)当删除区的空间达到了 max 值时，删除区的垃圾还没有被删除，这时停止活跃区的扫描，等待删除区进行垃圾删除。

4.2 实例分析

看下面一段程序：

```
int [][] m1=new int[2][3];
int [][] m2=new int[2][3];
m1=m2;
```

此例中，第一句是用 new 语句在堆中为数组申请了一个空间，然后用 matrix 引用此空间的对象（这里数组可以理解为对象），此时这个内存空间就是有用的。第二句是给 matrix 赋空值，matrix 则不再引用此数组。此时，这个空间就是无用的。

对于原来的算法，m1 引用的数组在堆中是随机存放的，若要查找垃圾，则会遍历整个堆内存，先标记，然后再清理垃圾。设耗费时间为 T_a。

m1 引用的数组在堆中是随机存放的，所以假设其放在活跃区中的 cnt[x]区（x 取值为 0~7 之一），下面分两种情况来考虑：

(1)最好的情况，m1 原来引用的数组放在 cnt1 中为数组 cnt[x]中最大的数，则查找到这个垃圾的时间为：T_8+cnt[x]。

(2)最差情况，m1 原来引用的数组放在 cnt[x]中为数组 cnt[x]中最小的数，则查找到这个垃圾的时间为：T_8+T_a。

则，平均查找时间 T_p 为：(最好情况+最坏情况)/2

$$T_p=(T_8+\mathrm{cnt}[x])+(T_8+T_a)\ /\ 2$$
$$=T_8+(T_a+\mathrm{cnt}[x])/2$$

因为 $T_{cnt}[x]=T_a/8$，$T_8<<T_a$，所以 $T_a-T_p=T_a-(T_a/8+T_a)/2-T_8$

$$T_{cnt}[x]=7T_a/16-T_8$$
$$\approx 7T_a/16$$

由此可以节省 $7T_a/16$ 的时间。此新算法可以大大减少垃圾处理所需的时间。

Java 语言对垃圾的处理是利用 Java 的垃圾处理器自动进行的，JVM 虽然没有明确程序员必须了解垃圾处理器的过程和实质，但是，一个优秀的 Java 程序员应该掌握和熟悉垃圾处理器的工作机制，充分利用好内存空间，减少不必要的空间浪费，从而使程序更好地运行。

参考文献

[1] 邓洋春，梁昔明. Java 无用单元回收方式与性能分析[J]. 现代计算机(专业版)，2009(01)：114-116.

[2] JONES R，LINES R. 垃圾收集[M]. 谢之易，译. 北京：人民邮电出版社，2004.

[3] 贾晓霞，吴际，金茂忠，等. Java 程序内存泄漏综述[J]. 计算机应用研究，2006，23(09)：1-3.

[4] SIERRA K. Java 2 学习指南[ISBN]978-7-115-11803-5.2004(1).

[5] 欧阳辰，周欣. 垃圾收集器与 Java 编程. http://www.ibm.com/developerworks/cn/java/l-JavaMemoryLeak2/2002.11.16.

(收稿日期：2010-09-27)

作者简介：

张鹏飞，男，1988 年生，本科，主要研究方向：计算机应用。

钱敏，女，1988 年生，本科，主要研究方向：会计理论与实务。