一种基于半空间的不完全拷贝垃圾回收机制

吴 昊，季振洲

（哈尔滨工业大学 计算机科学与技术学院，150001 哈尔滨，wuhaoster@gmail.com）

摘 要：为了满足半空间拷贝垃圾回收的实时性需求，克服在半空间拷贝回收过程中因大量的内存拷贝操作带来的时间开销方面的不足，提出了一种将对象分类并进行不完全拷贝的回收机制. 针对大对象的生命周期较长及空间分布的连续性特点，在半空间拷贝回收的遍历阶段，对活动的大对象采取只标记不拷贝的策略，被标记的对象仍留在起始空间，通过增加一个整理阶段维护了较低水平的内存碎片程度. SPECjvm2008和Dacapo两项基准测试验证了提出的方法的有效性，对比实验结果表明相比半空间拷贝回收，有效地降低了回收的平均时间开销，提高了半空间垃圾回收的实时性.

关键词：垃圾回收；半空间拷贝；实时系统；Java 虚拟机

中图分类号：TP314 **文献标志码**：A **文章编号**：0367－6234(2011)11－0060－05

Partial copying garbage collection mechanism based on semispace

WU Hao, JI Zhen-zhou

(School of Computer Science and Technology, Harbin Institute of Technology, 150001 Harbin, China, wuhaoster@gmail.com)

Abstract: To meet the needs of real-time characteristics as well as to overcome the time overhead caused by memory copying operations during the process of semispace collection, an improved method was proposed which was based on partial copying mechanism. Based on the fact that the large objects are always long-lived objects as well as distributed continuously in memory space, the lived large objects will only be marked without being moved during the collection phase of semispcae GC. The marked objects remain in the original space, and a compaction phase is added to maintain a low degree of memory fragmentation. SPECjvm2008 and Dacapo test suites are used to demonstrate the effectiveness of the proposed algorithm. Comparison experiments shows that proposed method can efficiently reduce the average pause time of the collection and ensure the real-timeness for semispace garbage collection.

Key words: garbage collection; semispace copying; real-time system; Java JVM

在一些面向对象程序设计语言中，如Java，支持自动内存管理的机制已经被广泛地接受，其重要原因就是自动内存管理可以有效地克服手动内存管理的缺陷. 传统的手动内存管理的缺陷主要体现在2个方面：1）程序员使用内存操作函数进行分配和回收内存，当由于疏忽或错误造成程序没有及时释放已不再使用的内存，就会造成内存泄露，降低程序性能；2）如果程序频繁地申请和释放不同大小的内存，会产生大量细小的无法分配的空闲内存，也就是内存碎片，内存碎片同样会导致系统性能的下降. 随着程序规模和复杂程度越来越大，自动内存管理显得越来越重要.

自动内存管理是由垃圾回收器来完成，大部分的垃圾回收器都是由一组根指针开始遍历所有被这些根指针直接或间接引用的对象，最后把没有被遍历到的对象作为垃圾回收. 经典的半空间垃圾回收算法属于追踪类垃圾回收算法的一种，其优点是回收过程简单且不产生内存碎片，但缺点也很明显，主要体现在：1）回收过程中对象拷贝的时间开销大；2）内存始终仅有1/2被利用，浪费了1/2的内存. 目前，在垃圾回收算法中，针

收稿日期：2010－10－23.
基金项目：国家自然科学基金资助项目（60475012）.
作者简介：吴 昊（1984—），男，博士研究生；
季振洲（1965—），男，教授，博士生导师.

对不同特征的对象或者在不同的回收阶段使用不同的回收策略来提高垃圾回收性能的改进方式越来越被关注. 这其中一个成功的应用就是基于年代的垃圾回收机制，它将对象按生命周期进行分类，通过减少对具有较长生命周期的对象的拷贝提高回收的性能. 针对半空间垃圾回收的不足，本文尝试将对象按大小分类，通过减少对大对象的拷贝的方式提高半空间垃圾回收的性能，减少时间开销.

1　相关工作

目前的垃圾收集算法中，依据区分活动对象和垃圾方式的不同，可以把回收算法分为：引用计数式、追踪式和分代式垃圾回收算法. 引用计数方式圾圾回收算法最适合用于实时应用，但由于其不能解决循环引用而带来的内存泄露问题，因此常常作为辅助的方式配合其他方式的回收；追踪式垃圾回收算法能有效的克服引用计数的不足，它是由一组根节点遍历整个对象引用图，遍历到的对象标记为活动对象，没有标记的对象被认为是垃圾将被回收；分代式垃圾回收算法将堆内存划分为不同的区域，用来存放不同“年龄”的对象，根据不同生命的对象采用不同的回收策略，分代式垃圾回收算法往往能获得较好的性能.

在实时垃圾回收方面，Bacon[1]通过设计低开销的读拦截方式，有效地降低回收过程的内存碎片并在回收过程中降低拷贝发生的次数；Auerbach[2]设计了一个支持实时垃圾回收的 Java 虚拟机，作为核心模块之一的回收器 Metronome 是一个基于时间调度的支持异步回收机制的回收器，可以很好地满足实时性要求；在此基础上，设计了 Tax-and-Spend[3]的调度机制，融合了多个现有的实时垃圾回收调度机制，在不同的应用需求下来决策使用不同的调度策略；Kalibera[4]通过优化对象在堆上的读写方式，设计了一个低内存碎片且满足实时需求的回收机制.

为了满足硬实时系统任务对垃圾回收的时限要求，Henriksso 等[5-7]提出将垃圾回收作为单独的实时任务参与调度来保证实时性，而不是由任务操作触发. 此外，Garner[8]认为在追踪遍历阶段的消耗，影响了系统的性能，并通过预取的方式进行了改进；Chen 等[9]针对嵌入式，内存受限条件下提出如何进行回收性能调优，并详细阐述了在内存受限的结构中垃圾回收的重要性；Siebert[10]分析了多核环境下的标记清除方式的垃圾回收，并给出了最坏的情况下性能提升的上界.

2　不完全拷贝回收

2.1　追踪类算法比较

在追踪类的垃圾回收算法中，垃圾回收器从一组根结点开始，依次遍历所有对象，并识别出哪些是活动的对象，哪些是垃圾对象，非活动的对象所占的内存由回收器收回. 此类算法中比较成熟的是标记清除算法(mark-sweep)和半空间拷贝算法(semispace). 标记清除算法将垃圾回收工作分为 2 个阶段：1)在遍历过程中定位并标记所有的活动对象；2)清除回收所有未被标记的对象，为了避免因此带来内存碎片，回收过程也会采取对内存进行压缩的策略. 半空间拷贝回收与标记清除回收不同的是：半空间算法将内存划分为2 个大小相等的空间，分别称为起始空间(from-space)和到达空间(to-space). 回收开始前，所有对象都存放在起始空间，一旦回收工作启动，在遍历过程中将发现的活动对象依次移动到到达空间，当回收工作完成时，所有的活动对象都被移动到到达空间，起始空间的内存整体回收，下次回收启动时再将 2 个空间的角色互换. 半空间回收是一种高效的回收机制，但是需要浪费 1/2 的内存空间，表 1比较了 2 种追踪类回收算法.

表 1　2 种追踪类回收算法比较

比较项目	标记 - 清除	半空间拷贝
内存碎片	有	无
内存利用率	无浪费	浪费 1/2
回收时机	分为标记和清除 2 个阶段	遍历过程中移动对象
是否压缩	需要	不需要
主要开销	压缩，碎片整理	内存移动

2.2　算法思想

相比标记清除回收，半空间算法不需要考虑内存碎片的压缩问题，在执行效率上优势明显. 但半空间算法需要进行大量的内存拷贝，对象的个数及大小决定了该回收算法的执行速度，因此减少拷贝是算法性能提升的关键. 半空间的算法的另一优势是在内存回收过程中不会产生内存碎片，这也是在改进过程中需要考虑的因素. 在如何减少拷贝上，本文的出发点是减少大对象发生拷贝的次数. 大对象、大数组在回收拷贝过程中耗费较多的时间，被认为是影响系统实时性的主要因素，在文献[1,3]中对大对象的生命周期及空间分布特点总结为：1)大对象往往是生命周期较长的对象；2)大对象通常在内存空间分布上有一定的连续性. 本文认为其合理性主要体现在：1)大

对象的加载工作集中在系统的启动阶段,符合实时系统的一般特征;2)频繁的加载 - 弃用大对象的行为只存在少数的应用之中. 在此基础上,本文提出了一种改进的不完全拷贝的半空间回收算法,算法的主要机制为:1)在回收的遍历阶段,只对活动的大对象采取只标记不拷贝的策略,被标记的对象仍留在起始空间;2)增加一个整理阶段,仅当起始空间的碎片程度较大时才执行整理. 按上述 2 点改进的算法的合理性主要体现在:根据条件 1)算法体现了按对象生命长度进行分代回收的思想;在满足条件 1)、2)的前提下,算法仍可以维护一个较低水平的内存碎片程度;相比标记 - 清除回收,算法增加的整理过程代价要小,与半空间拷贝相比,算法拷贝的代价要小.

2.3 算法描述

图 1 给出了不完全拷贝回收的流程. 算法使用半空间策略,在初始阶段将内存划分为 2 个等大小的区域:1)到达空间(to-space);2)起始空间(from-space),在回收前所有对象都存放在起始空间. 当垃圾回收工作时,初始化 2 个变量 N_{live},N_{gar} 分别用来记录活动的大对象和垃圾的大对象,在 LargeObj 中判断该对象是否满足预先设定的大对象条件. 在第 1 个阶段,算法分别遍历所有的活动对象和垃圾对象,对于活动的大对象需要进行标记并更新计数 N_{live},其他的活动对象则直接拷贝到到达空间;对于被识别为垃圾的大对象,仅需要更新计数 N_{gar}. 在第 2 个阶段利用 N_{live},N_{gar} 评估当前起始空间的碎片程度,当碎片度满足预先给定的上限时则需要在起始空间进行一次内存整理. 在整理的过程中仅需要移动在第 1 阶段被标记的大对象即可. 最后起始空间和到达空间角色互换,下次回收在重复上述过程.

```
Procedure partial_copying_GC
begin
  1. N_live := 0
  2. N_gar := 0
  3. for each Object obj in live set do
  4.    if LargedObj (obj. Size) then
  5.       obj. Marked := true //MARK LARGE OBJECT
  6.       N_live = N_live + 1
  7.    else
  8.          Move obj to To_Space     //MOVE OBJECT
  9. for each Object obj in garbage set do
  10.   if LargeOjb (obj. Size) then//CHECK LARGE OBJ
  11.         N_gar := N_gar + 1
     //CHECK FRAGMENTATION
  12.    if FragmentUpperBound (N_gar, N_live) then
  13.       Compact(From_Space) //COMPACT
  14.    Flip(From_Space, To_Space) //FLIP
end.
```

图 1 不完全拷贝回收过程

在回收过程中内存被划分为 2 个等半区,每次回收并不对所有的活动对象都进行拷贝,随着回收工作的进行,活动的大对象在 2 个半区都有分布,但是在每一个回收周期中,垃圾对象只产生于 FROM-SPACE 角色的半区中.

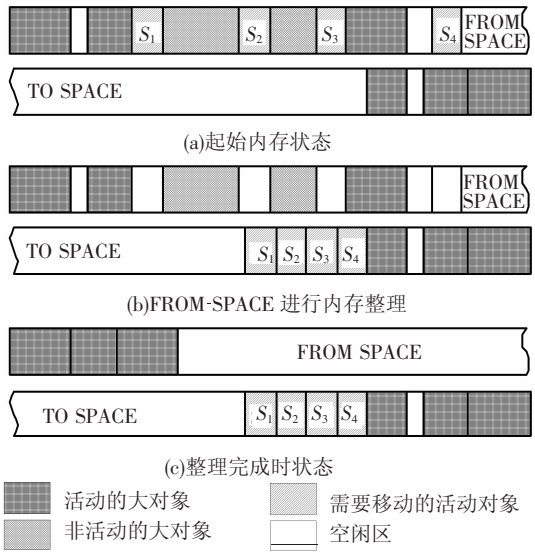

图 2 内存回收及整理的过程

由上述算法可知,垃圾回收的工作分为 2 个阶段来完成:1)对象拷贝的过程;2)当内存碎片达到一定程度时执行半区内的内存整理过程. 图 2演示了执行 2 个阶段内存回收工作的过程. 假设以图 2(a)为起始的内存状态,在 FROM-SPACE 半区遍历的过程中发现 S_1、S_2、S_3、S_4 这 4 个活动对象需要移动,然后回收器把它们拷贝到 TO-SPACE 半区中,如图 2(b)所示. 在遍历过程的同时记录下被识别为垃圾的大对象,以确定是否有必要进行内存整理. 在第 2 阶段,对如图 2(b)所示的 FROM-SPACE 进行内存整理,由左到右依次将活动的大对象进行移动,整理完成时状态如图 2(c)所示,FROM-SPACE 垃圾全部回收. 在下一次执行回收前将 FROM-SPACE 和 TO-SPACE 的角色互换,然后再重复上述回收过程. 在内存碎片程度不高的情况下,算法仅需要对一部分活动的对象进行拷贝,且拷贝的对象都不是大对象;在最坏的情况下,算法需要对 FROM-SPACE 半区进行整理,但并不需要对全部的大对象进行拷贝,由于是不完全的半空间回收,随着程序的执行大对象在内存的 2 个半空间都有分布,而回收和整理只在 FROM-SPACE 半区进行.

3 实验结果

3.1 实验平台

为了对不完全拷贝垃圾回收的性能进行评

估，本文采用 2 种基准测试来检验算法的有效性. 本文使用虚拟机平台为 Apache-Harmony 5.0[11] 提供的 Java 虚拟机 DRLVM，在该平台上设计了本文提出的不完全拷贝垃圾回收回收算法. 在实验中，使用的是比较公认的 Java 虚拟机性能基准测试程序 SPECjvm2008[12] 和 Dacapo[13]. 具体的实验环境及测试项目如表 2 所示.

表 2　实验平台及测试环境

项目	描述
实验平台	Intel Core2 Quad CPU Q8300，2.50 GHz 2 GB 内存 Windows XP Professional 2002 SP3
基准测试	SPECjvm2008 1.01 测试项目： mpegaudio sunflow compiler crypto scimark Dacapo 9.12 测试项目： avrora fop jython luindex lusearch pmd sunflow
JVM 环境	Apache-Harmony 5.0

3.2　实验 1：SPECjvm2008 基准测试

SPECjvm2008 是一个观测 JRE 运行性能的基准测试套件. 它的测试用例涵盖了大部分 Java 基础应用场景，是架构选型和 VM 性能评测的重要依据. 本实验中选中的 6 项测试涵盖了多线程图片渲染、矩阵运算、音频解码、不对称加密、压缩算法和普通 Java 编译这些应用. 在实验中，使用 SPECjvm2008 分别对使用不完全拷贝回收的虚拟机和使用半空间回收的虚拟机进行基准评测，并给出比较的结果. 图 3 是 2 种回收算法在 SPECjvm2008 下的基准评测得分结果，包括 6 个项目的评测得分和最后的平均得分(composite). 测试得分反映了虚拟机执行基准测试的吞吐性能，得分越高表示在单位时间内执行的操作数越多. 可以看出相比半空间回收，使用不完全拷贝回收的虚拟机的基准评测得分要高一些.

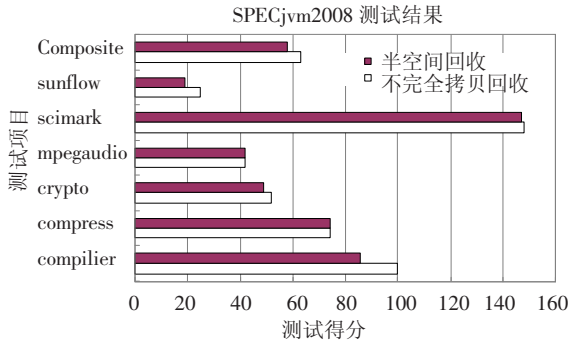

图 3　SPECjvm2008 基准测试实验结果

3.3　实验 2：垃圾回收性能测试

在 SPECjvm2008 的 6 项基准测试中，主要是对使用不同的垃圾回收算法的虚拟机的综合性能进行比较. 而 Dacapo 的测试程序集更侧重于对 Java 虚拟机的内存管理方面的评估. 在测试中，本文对 Dacapo 的 7 项测试的每一项重复执行 10 次，然后统计运行每一项测试的平均耗时并统计垃圾回收的情况，对于垃圾回收分别记录下了平均的回收次数和平均耗时以及最大耗时，测试结果如表 3 所示.

表 3　Dacapo 基准测试结果

测试项目	运行时间/ms	回收次数	回收耗时/ms	
			最大	平均
avrora	25 542	3.5	30	8
fop	1 437	3.0	91	26
jython	9 528	24.0	76	23
luindex	3 708	2.5	30	10
lusearch	4 103	91.0	30	5
pmd	3 058	6.0	76	23
sunflow	6 450	63.0	30	8

回收耗时反应了在执行垃圾回收过程造成的应用程序停顿时间，平均的回收耗时是衡量一个回收算法性能的重要指标，实验 2 比较了 2 种垃圾回收算法的平均回收时间的情况. 由图 4 可以看出相比半空间垃圾回收不完全拷贝回收的平均时间开销较小，在平均的垃圾回收耗时上本文的算法相比经典的半空间回收降低了 1/4 左右.

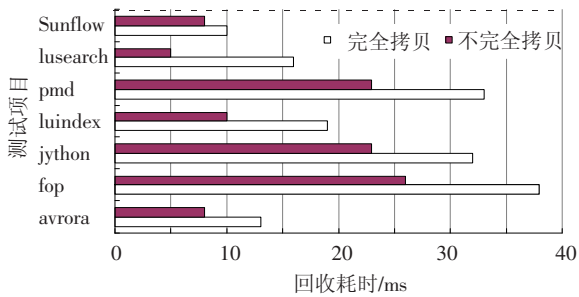

图 4　Dacapo 测试下的平均回收耗时

4　结　论

1)深入研究了追踪类垃圾回收的原理，针对传统的半空间垃圾回收的不足，提出了将对象分类并进行不完全拷贝回收的改进方法.

2)基于半空间的不完全拷贝回收机制，考虑了大对象的生命周期及空间分布特点，通过减少对大对象的拷贝次数、降低内存碎片以及内存压缩开销等方式来提高半空间垃圾回收的性能.

3)基准测试及对比实验验证了算法的有效性，表明该方法可以有效地降低回收的平均时间开销，提高了半空间垃圾回收的实时性.

参考文献:

[1] BACON D F, CHENG P, RAJAN V T. A real-time garbage collector with low overhead and consistent utilization[C]//Proceedings of the 30th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. New York, NY: ACM, 2003: 285 – 298.

[2] AUERBACH J, BACON D F, BLAINEY B, *et al.* Design and implementation of a comprehensive real-time Java virtual machine[C]//Proceedings of The 7th ACM/IEEE International Conference on Embedded Software. New York, NY: ACM, 2007: 249 – 258.

[3] AUERBACH J, BACON D F, CHENG P, *et al.* Tax-and-spend: Democratic scheduling for real-time garbage collection[C]//Proceedings of the 7th ACM International Conference on Embedded software. New York, NY: ACM, 2008: 245 – 254.

[4] KALIBERA T. Replicating real-time garbage collector for Java[C]//Proceedings of the 7th International Workshop on Java Technologies for Real-time and Embedded Systems. New York, NY: ACM, 2009: 100 – 109.

[5] HENRIKSSON R. Scheduling Garbage Collection in Embedded Systems[D]. Sweden: PhD thesis, Lund University, 1998.

[6] ROBERTZ S G, HENRIKSSON R. Time-triggered garbage collection: Robust and adaptive real-time GC scheduling for embedded systems[C]//ACM SIGPLAN 2003 Conference on Languages, Compilers, and Tools for Embedded Systems. New York, NY: ACM, 2003: 93 – 102.

[7] BACON D F, CHENG P, RAJAN V T. Controlling fragmentation and space consumption in the Metronome, a real-time garbage collector for Java[C]//Proceedings of the Conference on Languages, Compilers, and Tools for Embedded Systems. New York, NY: ACM, 2003: 81 – 92.

[8] GARNER R, BLACKBURN S M, FRAMPTON D. Effective prefetch for mark-sweep garbage collection[C]//Proceedings of the 6th International Symposium on Memory Management. New York, NY: ACM, 2007: 43 – 54.

[9] CHEN G, SHETTY R, KANDEMIR M, *et al.* Tuning garbage collection for reducing memory system energy in an embedded java environment[J]. ACM Transactions on Embedded Computing Systems, 2002, 1(1): 27 – 55.

[10] SIEBERT F. Limits of parallel marking garbage collection[C]//Proceedings of the 7th International Symposium on Memory Management. New York, NY: ACM, 2008: 21 – 29.

[11] The Apache Software Foundation. Dynamic Runtime Layer Virtual Machine Developer′s Guide[EB/OL]. [2009-11-18]. http://harmony.apache.org/subcomponents/drlvm/developers_guide.html.

[12] Standard Performance Evaluation Corporation. SPECjvm2008 Run and Reporting Rules[EB/OL]. [2008-04-14]. http://www.spec.org/jvm2008/docs/Runrules.html.

[13] BLACKBURN S M, GARNER R. The DaCapo Benchmarks: Java benchmarking development and analysis [C]//Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-oriented Programing, Systems, Languages, and Applications. New York, NY: ACM, 2006: 169 – 190.

(编辑 张 红)