基于实时性的 Java 虚拟机垃圾收集算法 *

白江涛, 钟 勇, 朱颢东
(中国科学院 成都计算机应用研究所, 成都 610041)

摘 要: 提出了一种适用于实时性环境的 Java 虚拟机垃圾收集算法。该算法对增量式收集器中堆空间的划分方式、引用跟踪等方面进行了改进,以减少垃圾收集带来的不确定性暂停,并可以使用户指定一个时间段内垃圾收集导致应用程序暂停的最长时间,从而使其适用于实时性环境。实验结果表明,该算法有效减少了暂停的频率和时长。

关键词: 垃圾收集; 实时性; 增量式收集器; 堆空间划分; 引用追踪
中图分类号: TP18 **文献标志码**: A **文章编号**: 1001-3695(2010)09-3431-03
doi:10.3969/j.issn.1001-3695.2010.09.061

Real-time garbage collection algorithm in Java virtual machine

BAI Jiang-tao, ZHONG Yong, ZHU Hao-dong
(*Chengdu Institute of Computer Application, Chinese Academy of Sciences, Chengdu* 610041, *China*)

Abstract: This paper described a garbage collection algorithm for real-time environment. The algorithm improved the division of heap and reference tracking of the incremental garbage collection algorithm for reducing the uncertain pauses caused by garbage collection, furthermore, and provided user to give a biggest time value that the application paused in a certain period. All of these made it suitable for real-time environment. Experiment results show that this algorithm does reduce the frequency and duration of pauses.

Key words: garbage collection(GC); real-time; incremental garbage collection; division of heap; reference tracking

0 引言

垃圾收集机制(GC)是 Java 语言的一大优势,它实现了对内存的自动收集与重用,使程序员可以集中精力于算法和程序的逻辑设计,提高软件的质量和生产效率,最重要的是极大减少内存泄露情况的发生。但由于 GC 的运行,通常会导致应用程序中的不确定性暂停,暂停的频率和时长都不可预测,使得 Java 语言在传统上并不适合开发实时应用程序,限制了其在工业控制、新兴的实时网络游戏等领域的应用。

提高 Java 实时性的一个重要方面就是提高 GC 的实时性,其基本思想是将垃圾收集导致的暂停限制在一个可控的范围内,满足实时应用程序低暂停时间[1]的要求。目前已经有了针对实时性要求而优化的垃圾收集器,如增量式收集器。本文首先介绍这些收集器,在其基础上进行改进,使其更好地适用于实时性环境,并通过实验验证了改进后的效果。

1 实时性 GC 分析

1.1 GC 对性能影响的来源

1)造成当前运行线程的停顿 传统 GC 采用 Stop-the-World[1]的方式运行,运行时,所有其他的线程都被暂停,直到 GC 工作完成。

2)遍历对象引用的开销 如果 JVM 中的对象很多,那遍历完所有存活对象必将是很大的开销。

3)清理和回收垃圾的开销 遍历完对象引用之后,对垃圾的清理和回收也有较大的开销。这部分开销可能包括复制内存块、更新对象引用等。

1.2 分代收集器(generational GC)

分代垃圾收集基于一个事实,即绝大多数对象存活的时间很短。研究表明,在大多数程序中,绝大多数对象(超过 90%)都是临时对象,存活的时间很短[2]。从而将堆空间按照对象的生命周期划分为不同的区域,每个区域根据对象的生命周期不同采用不同的垃圾收集算法[2]。如图 1,是 Sun 公司的 JVM 堆的分代图,灰色部分表示年轻代区域,年轻代可以再分,将存活的对象逐代提升。

图1 JVM堆空间分代图

由于分代收集算法在大多数时候都只需对年轻代进行垃圾回收[2],有效减少了遍历标记阶段的计算量和停顿时间,有较高的垃圾回收率。

1.3 增量式收集器(incremental GC)

增量式收集器仍采用分代的方式,并对实时性作了进一步优化。简单地说,增量式收集器就是通过一定的算法,把一个长时间的中断,划分为很多个小的中断,通过这种方式减少垃圾收集对用户程序的影响。虽然增量式收集在整体性能上可能不如传统垃圾收集器的效率高[3],但是它能够减少程序的

收稿日期: 2010-03-13; **修回日期**: 2010-04-28 **基金项目**: 四川省科技计划项目(2008GZ0003);四川省科技攻关项目(07GG006-019)
作者简介:白江涛(1983-),男,硕士,主要研究方向为软件过程技术与方法; 钟勇(1966-),男,研究员,博导,主要研究方向为软件过程技术与方法;朱颢东(1980-),男,博士,CCF 会员,主要研究方向为软件过程技术与方法、文本挖掘、智能信息处理(damianfang001@163.com).

最长停顿时间。

图 2 就表示了增量式收集器和传统收集器的比较。其中灰色部分表示线程占用 CPU 的时间。

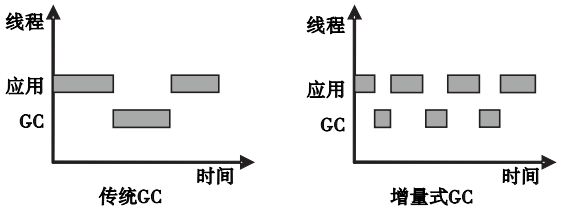

图2　增量式GC与传统GC的比较

Sun JDK 提供的 HotSpot JVM 支持增量式垃圾收集,它采用 Train GC 算法,基本思想是:在堆中老代与年轻代之间创建一个新区域。这些堆区域划分为火车(train),每个火车又分为一系列的车厢(car),每个车厢可以分别收集[3],这样每次收集时仅对若干车厢进行,因此可以得到很短的收集暂停。

不过这种方法也有很大的不足:a)对象之间引用的维护会很复杂,增加了应用程序和垃圾收集的工作量;b)尽管垃圾收集引发的暂停缩短,但是垃圾收集的周期,以及每次收集的时间还是不确定。

2　实时性 GC 的改进

1)堆空间的划分方式

新算法中将堆空间划分为多个固定大小的区域(region),如图 3 所示的 $R_1 \sim R_4$,同时在每个区域上保留代的概念,以利用分代收集算法的高效率。

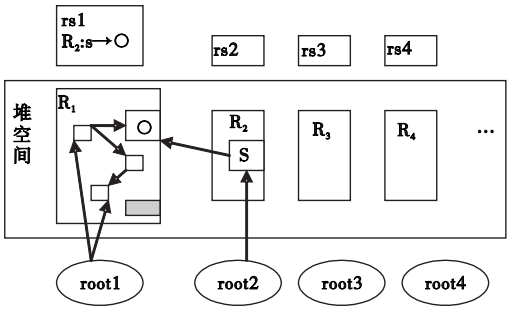

图3　新算法中堆空间的划分

这样划分的好处有:

a)对堆中的对象进行遍历标记时,可以更好地并行执行;

b)垃圾收集以这些区域为单位进行,回收时可以仅回收一部分区域,增加了灵活性,同时较之增量收集器也减少了维护区域内对象之间引用的复杂性;

c)更大的好处在于,因为区域大小固定,所以允许 JVM 指定在某个时间段内 GC 所导致的应用暂停的最大时间为多少,通过预测在这个时间内可以完整回收若干区域,这对实时性应用来说非常重要。

假设允许暂停的最大时间为 T,预测收集每个区域的平均时间为 t(包括标记和清理的时间),规定一个阈值 L,表示每次 GC 所能收集的区域数的最大值。这样就得到在暂停的时间段内可以进行的垃圾回收次数为($T/tL+1$)的整数值。因此垃圾收集的频率和每次收集的时间就可以确定了,适当调整参数的值,完全可以使系统达到一种准实时响应的效果。

2)引用追踪

堆空间划分多个区域后,区域之间的对象引用通过记录集来维护,如图 3 所示的 $r_s1 \sim r_s4$。每个区域都有一个对应的记录集。其中包含了引用当前区域中对象的区域的对象的指针,如在图 3 的 r_s1 中就记录了一条从 R_2 中的 S 对象到 R_1 中的 O 对象的引用。由于除了 GC 外应用也会造成区域中对象的引用关系不断地发生改变,每个应用的线程都有一个关联的记录集 log,用于缓存和序列化线程运行时对区域间引用关系的修改。如果造成的改变仅为同一区域中的对象之间的关联,则不记录 log;如造成的改变为跨区域中的对象的关联,则记录到线程的记录集 log 中,待垃圾收集进行时,log 中的记录被处理,相关的区域的记录集表更新。

3)收集过程

(1)首先从每个区域的根集[4]出发,对本区域的对象进行标记,由于区域之间相对独立,可以利用多 CPU 或多线程对所有区域并行处理,这一步要暂停应用的执行。

(2)按照(1)步中扫描到的对象进行遍历,以识别这些对象的下层对象的活跃状态,这一步中会涉及对象之间跨区域引用的情况,并对区域的记录集及时更新。在此期间应用线程可以并发执行,修改的对象的依赖关系则记录到线程的记录集 log 中。

(3)把应用线程中存在的记录集 log 的内容进行处理,并相应地修改区域对应的记录集,这一步需要暂停应用,且并行运行。

(4)根据区域中活跃对象所占的空间大小(Byte)进行排序,首先回收活跃对象空间小以及回收耗时短(预估出来的时间)的区域;回收的方法为将此区域中的活跃对象复制到另外的区域中,并根据指定的 GC 所能占用的时间来估算能回收多少区域。

整个收集过程如图 4 所示,GC 线程的双箭头表示并行运行。

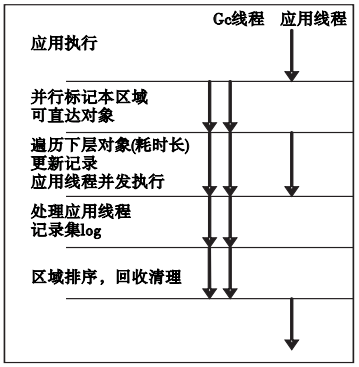

图4　垃圾收集过程

3　实验结果与分析

实验采用几种典型的应用来对新算法和增量式垃圾回收算法,在回收效率、平均暂停时间、暂停次数等方面进行比较。其中新算法用 C 语言实现,增量式垃圾回收算法直接使用 Sun JDK1.4.1(更高版本已不支持[5])中所带的,使用参数-Xincgc 启用[6]。

在参数设置上,增量式收集器将最大堆空间设为 64 MB(使用参数-Xmx[6]),其他采用默认设置;新算法中最大堆空间也设为 64 MB,堆内域的数量为 10。

应用包括一款游戏软件(game)、一个实时温度监控系统(temperature)、一个股票应用软件(stock)和一段产生较多垃圾

对象的程序(program)。从表 1 中可以看出,新算法在有效压缩 GC 引起的暂停时间和暂停次数的同时,均大幅提高了垃圾回收的效率,最大的增幅有两倍多,这主要得益于堆空间划分方式对并行收集的良好支持。

表 1　算法的执行结果

应用	效率/千字节/ms		平均暂停时间/ms		暂停次数	
	新算法	增量式 GC	新算法	增量式 GC	新算法	增量式 GC
game	7.8	4.0	5.9	8.8	25	39
temperature	5.7	3.1	6.2	9.4	17	36
stock	5.2	2.4	5.0	8.2	23	44
program	11.8	9.6	5.5	7.9	9	25

通过对堆空间域数量参数的调节发现,垃圾收集的效率随着域数量的增加呈抛物线性变化,开始性能的提升得益于分代收集算法的思想,当域数量增大到一定程度时,维护引用、遍历标记对象以及垃圾清理的开销都会变大,从而降低了收集的效率。而暂停时间和频率因为最大暂停时间的限制,保持在一个相对稳定的状态。

4　结束语

本文提出了一种对 Java 堆空间的新的划分方法,将因垃圾收集而导致的程序暂停时间和次数限制在一个可控的范围内,并可以通过相关参数的调节,达到一个最优的状态,满足了实时性应用所要求的短暂停,同时保持了一个较高的收集效率。

参考文献:

[1] BAKER H G. List processing in real-time on a serial computer[J]. Communications of the ACM,1978,21(4):280-294.

[2] 谌宁,覃征. 基于嵌入式 Java 虚拟机的垃圾回收算法.[J]. 计算机应用,2005,25(1):218-223.

[3] GOETZ B. Java theory and practice:Garbage collection in the HotSpot JVM[EB/OL]. [2003-11-25]. http://www.ibm.com/developerworks/library/j-jtp11253/index.html? S_TACT = 105AGX52&S_CMP = cn-a-j.

[4] YUASA T. Real-time garbage collection on general-purpose machines [M]. NewYork:Elsevier Science Inc,1990.

[5] Tuning garbage collection with the 1.4.2 Java virtual machine[EB/OL]. [2003]. http://java.sun.com/docs/hotspot/gc1.4.2/.

[6] A collection of JVM options[EB/OL]. [2007]. http://blogs.sun.com/watt/resource/jvm-options-list.html.

[7] SACHINDRAN N, ELIOT J, MOSS B. Mark-copy: fast copying GC with less space overhead[C]//Proc of the 18th ACM SIGPLAN OOPSLA. New York:ACM,2003:326-343.

[8] BIRON B,SCIAMPACONE R. Real-time Java[M]. Beijing:O'Reilly,2007.

[9] 张宁,熊光泽. 基于关键引用验证的分布式实时垃圾搜集器[J]. 计算机应用研究,2009,26(11):4036-4038.

[10] 赵苑苑,王万诚,黄广君,等. 无用内存单元自动回收过程的实时性问题研究[J]. 计算机应用与软件,2006,23(2):137-139.

[11] DETLEFS D,FLOOD C, HELLER S, *et al*. Garbage-first garbage collection[EB/OL]. [2009]. http://research.sun.com/jtech/pubs/04-g1-paper-ismm.pdf.

(上接第 3408 页)不仅包含了基于浏览速度的计算方法,而且进一步引入了基于有效信息的算法,更客观地考虑了影响兴趣度权值的因素。

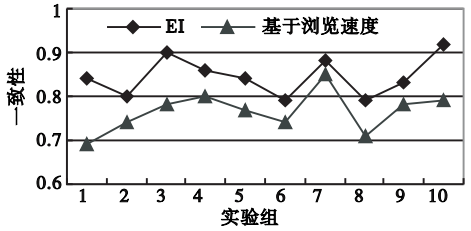

图3　不同权值算法下的一致性比较

5　结束语

为了提高用户兴趣分类及其兴趣度计算的准确性,在原有方法基础上,考虑特征词语义以及在文本层面的重要性,引入加权语义网对文本相似度进行计算;在计算用户兴趣度时,提出有效信息概念及其计算方法,更准确描述了浏览时间对用户兴趣度的影响。实验证明,该方法提高了用户分类的准确性,提高了兴趣度计算的准确性。个性化用户兴趣建模是一个比较复杂的过程,本文方法仍需进一步改进:a)由于 WordNet 是一个英文词汇网,本文相似度计算只适合英文站点,未来的研究中,进行中文语义方面的研究。b)更全面地考虑用户其他浏览行为,进一步提高兴趣度权值的计算精度。

参考文献:

[1] WU Yi-hung, CHEN Yong-chuan, CHEN A L P. Enabling personalized recommendation on the Web based on user interests and behaviors[C]//Proc of the 11th International Workshop on Research Issues in Data Engineering. Washington DC: IEEE Computer Society, 2001:17-24.

[2] 陈冬玲, 王大玲, 于戈. 支持个性化检索的 User Profile 研究概述[J]. 小型微型计算机系统,2008,29(10):1903-1907.

[3] GAUCH S, CHAFEE J, PRETSCHNER A. Ontology-based personalized search and browsing[J]. Web Intelligence and Agent Systems,2003,1(3):219-234.

[4] KIM H. Learning implicit user interest hierarchy for Web personalization [D]. Florida: Institute of Technology,2005.

[5] TAKAMA Y, ISHIBASHI T. M2VSM: extension of vector space model by introducing meta keyword[C]//Proc of World Automation Congress. Hawaii:World Automation Congress Proceedings,2008:179-184.

[6] GANESAN P, GARCIA-MOLINA H, WIDOM J. Exploiting hierarchical domain structure to compute similarity[J]. ACM Trans on Information Systems,2003,21(1):64-93.

[7] YANG Zhen, LEI Jian-jun, Wang Jian, *et al*. Improved VSM for incremental text classification [C]//Proc of International Evectrohic Conference on Computer Science. [S. l.]: AIP Conference Proceedings,2008:369-373.

[8] 郭庆琳, 李艳梅, 唐琦. 基于 VSM 的文本相似度计算的研究[J]. 计算机应用研究,2008,25(11):3256-3258.

[9] LARSEN B, AONE C. Fast and effective text mining using linear-time document clustering[C]//Proc of International Conference on Knowledge Discovery and Data Mining. California: ACM Press,2006: 16-22.