第 34 卷 第 23 期 计 算 机 工 程 2008 年 12 月
Vol.34 No.23 Computer Engineering December 2008
·软件技术与数据库· 文章编号：1000—3428(2008)23—0084—02 文献标识码：A 中图分类号：TP311.5

Java 程序内存低效使用问题的分析

柳永坡[1,2]，贾晓霞[1]，吴 际[1]，金茂忠[1]，孙惠丽[3]
(1. 北京航空航天大学计算机学院，北京 100083；2. 大庆石油学院计算机与信息技术学院，大庆 163318；3. 高等教育出版社，北京 100029)

摘 要：Java 程序内存的低效使用是导致其性能问题的主要因素。该文分析了泄漏对象、蚍蜉对象和空闲对象 3 类导致内存低效使用的情况，探讨解决上述问题的方法，并提出构造对象行为模式。实验结果表明，该方案是有效的。
关键词：内存低效使用；内存泄漏；垃圾回收

Analysis of Memory Usage with Low Efficiency in Java Program

LIU Yong-po[1,2], JIA Xiao-xia[1], WU Ji[1], JIN Mao-zhong[1], SUN Hui-li[3]
(1. School of Computer, Beijing University of Aeronautics & Astronautics, Beijing 100083; 2. School of Computer and Information Technology, Daqing Petroleum Institute, Daqing 163318; 3. Higher Education Press, Beijing 100029)

【Abstract】The main memory usage with low efficiency is key factor to downgrade Java program performance. Three cases including leaked objects, instantaneous objects and idle objects are analyzed, which downgrade Java programs performance. How to solve these problems is also discussed. Moreover, the constructing the object lifecycle behavior model is proposed. Experimental results show this scheme is effective.
【Key words】memory usage with low efficiency; memory leak; garbage collection

1 概述

Java 语言由于其平台独立性及面向对象等特点成为当前使用最为广泛的编程语言之一。Java 源文件在编译成字节码(byte code)后在 Java 虚拟机(Java Virtual Machine, JVM)上运行，其内存回收工作由 JVM 提供的垃圾回收器(Garbage Collector, GC)自动完成，程序员无需关心对象的回收工作。因此，很多人认为 Java 的内存回收由 JVM 负责而不会出现任何问题。其实，Java 程序也存在内存泄漏，即不再有用的对象无法被回收，其占用的内存空间无法被循环使用；Java 程序中的某些对象在被创建后的大部分时间内处于空闲状态，这些对象占用的内存也无法被循环利用。当 JVM 无法再申请到所需内存时，系统便会崩溃。对那些大规模应用程序，特别是运行时间很长、提供关键服务的 Server 端程序，这种崩溃是致命的。

本文在分析 Java 内存低效使用问题及其对程序运行效率危害的基础上，提出解决该问题的方案。

2 Java 程序的内存低效使用

2.1 泄露对象、蚍蜉对象和空闲对象

在Java程序中，很多变量所需的内存空间在编译时无法确定，需在程序执行过程中动态分配，并在不需要时被释放。Java语言内存机制的特点是其内存回收工作由JVM提供的垃圾收集器GC自动完成。GC监控对象的运行状态，采用可达测试(reachability test)[1]检测对象是否从根可达。GC回收失去引用、从根不可达的对象。但仍被引用(直接或间接被根引用)的对象不一定仍然有用。无用对象被其他有用对象引用导致其无法被回收，从而成为泄漏对象。

Java 程序中可能存在大量生存期较短的临时对象及一次性对象(对象循环，object recycling)。这些对象占用较大规模内存，导致 GC 频繁启动，消耗大量程序时间。这里形象地将其称为蚍蜉对象。

Java 程序还存在这样的状况——对象被分配后长时间处于空闲状态。这些对象生存时间往往较长，但其活动通常只集中于某几个短时间段内。该类对象最终被 GC 回收，但从被分配到被回收的较长时间内，对象并无活动(事实上，泄漏对象是种特殊的长时间无活动对象，称之为“空闲对象”)。

为了下文叙述便利，本文将以上 3 种情况称为 Java 内存的低效使用。

2.2 Java 内存低效使用的危害

Java 内存的低效使用从空间和时间 2 个角度危害程序性能：(1)程序在运行的某时刻消耗完 JVM 所能申请的所有内存而导致 JVM 崩溃；(2)导致 GC 频繁回收降低程序的运行效率。表 1 对此进行了简要分析。

表 1 Java 程序内存低效使用分析

类别	能否 GC	空间效率影响	时间效率影响	检测难度	修正难度
泄漏对象	否	显著	显著	不大	大
蚍蜉对象	能	不显著	显著	不大	不大
空闲对象	能	显著	显著	大	大

从表 1 可以看出，蚍蜉对象在 GC 启动后即被回收，对空间效率无显著影响；泄漏对象和空闲对象导致其占用的内存无法被循环利用，降低了空间效率。蚍蜉对象使 GC 频繁

基金项目：国家自然科学基金资助项目“基于数据挖掘的服务器端软件性能分析与诊断方法研究”(60603039)；国家“863”计划基金资助项目“基于 Web Services 的系统软件核心技术及运行平台研究”(2004AA112030)
作者简介：柳永坡(1971－)，男，副教授、博士研究生，主研方向：构件技术，软件测试，故障诊断分析；贾晓霞，博士研究生；吴 际，讲师、博士；金茂忠，教授、博士生导师；孙惠丽，硕士
收稿日期：2008-04-20 **E-mail：**liuypo@sei.buaa.edu.cn

启动，降低了时间效率；泄漏对象和空闲对象增加了 GC 进行可达检测的时间，也降低了程序的时间效率。

从检测和修正的角度看，在 JVM 关闭前转储堆中对象信息可检测到泄漏对象，且检测难度较低；但泄漏对象往往是由对象自身的逻辑缺陷引起的，修正这类问题必须分析对象的行为逻辑，修正难度较大。对于蚍蜉对象，只需从某段时间内 GC 的活动频度及其所回收的类实例种类和数目即可判断，由于大多是临时性对象，可以简单地根据对象的创建位置进行修正，因此检测难度和修正难度都不大。对于空闲对象，必须收集对象整个生存期内的行为信息才能判断对象的活动期和空闲期，发现难度较大，修正这类问题需要分析对象的行为和使用逻辑，修正难度较大。

3 当前的研究工作

很多泄漏对象是在操作中临时创建的，但在操作结束后未被回收。文献[2]设定了临时对象的期望生命周期，存活期限超出期望生命周期的对象便是泄漏对象。文献[3]针对 Java 中的数组提出一个结合前向数据流分析和后向控制流分析的算法，GC 利用数组间的约束关系回收不再有用的数组变量。文献[4]基于堆中的对象引用图分析了堆中对象的结构(增长迅速的结构可能被泄漏)。相对于上述内存泄漏检测的研究，内存泄漏的修正则困难许多。对象泄漏是由于对象自身的逻辑及被使用的逻辑不当造成的，要修正这类问题必须分析对象的行为逻辑，因此修正难度较大。

蚍蜉对象生存期很短，由于其大量存在导致 GC 频繁启动。当观测 JProbe Memory Debugger 的堆使用图发现尖刺(spikes)时，便是可疑的蚍蜉对象。对该段时间内 GC 的启动频度及回收的实例对象进行分析可检测并修正蚍蜉对象。所以，蚍蜉对象的检测和修正相对较容易一些。

空闲对象生存时间较长，对其检测需要很多信息，如对象的整个生存期行为、对象的活动及对象的空闲，因此，检测并修正空闲对象比较困难。

4 现有工具对比分析

目前已有很多工具致力于检测和修正内存低效使用，以提高Java程序性能。应用较广泛的软件有：JProbe Memory Debugger, BEA JRockit以及IBM Purify。这些工具大多基于JVMPI(Java Virtual Machine Profiling Interface)获取程序运行时的内存分配/回收信息[5](堆信息及对象的分配、引用关系及垃圾回收信息)，并对这些信息进行分析以发现内存低效使用。下面对这些工具进行简要对比。

JProbe 和 Purify 基于转储的堆进行离线分析，JRockit 集成 JVM 进行交互式在线分析。JProbe 提供内存泄漏医生，帮助用户定位可能引发内存泄漏的引用；Purify 提供方法的内存使用信息，用户结合该信息及对象的内存使用信息可在方法层寻找引发内存泄漏的原因。表 2 对这 3 种工具进行了综合对比分析。

表 2　3 种常用工具的对比分析

类别	支持的 JVM 种类	分析方式	对象分配轨迹	引用关系	特点
JProbe	无特殊限制	离线	完整	有	利用内存泄漏医生快速找到导致泄漏的引用，利用 GC 信息修正蚍蜉对象
JRockit	BEA JVM	在线	灵活定制	有	利用引用关系倒推引发泄漏的对象
Purify	无特殊限制	离线	无	有	可显示泄漏内存最多的方法

表 2 显示了 3 种工具的共性和差异。JRockit 只能运行于 BEA JVM；JProbe 和 Purify 没太多限制。JRcokit 内嵌于 JVM 进行在线分析，效率较高。对象间的引用关系是 3 个工具都用到的。表 3 总结了 3 种常用工具对解决内存低效使用问题的支持。

表 3　3 种常用工具对内存低效使用解决的支持

类别	Jprobe		JRockit		Purify	
	检测	修正	检测	修正	检测	修正
泄漏对象	支持	部分支持	支持	部分支持	支持	部分支持
蚍蜉对象	支持	支持	支持	不支持	不支持	不支持
空闲对象	不支持					

从表 3 可以看到，3 种工具都支持泄露对象的检测，但笔者认为其并未完全支持泄露对象的修正。因为对象的泄露是由其行为逻辑的缺陷造成的。3 种工具都可得到导致对象泄露的引用，但未就导致泄漏的缺陷代码范围给出建议。对于蚍蜉对象，观测 JProbe 和 JRockit 的堆使用图可发现蚍蜉对象的踪迹。利用 JProbe 的垃圾回收监视器中的回收对象种类及数目可修正蚍蜉对象。JRockit 和 Purify 则无明显的支持。对于空闲对象，3 种工具都不支持。

5 当前研究工作的不足及原因分析

当前研究存在如下不足：对于泄漏对象的研究大多集中于如何检测，在如何修正方面都未给出有效的解决方案；未能针对空闲对象给出相应的检测及修正方法；多数工具都要求用户对程序行为和实现有一定理解，否则难以有效利用工具提供大量信息。表 4 列出了当前的研究以及所运用的工具在检测和修正方面用到的信息。

表 4　检测和修正内存低效时所需的信息

类别	检测利用的信息	修正利用的信息
泄漏对象	对象间引用关系、对象生命周期，对象是否可用	对象间引用关系
蚍蜉对象	对象生命周期、堆使用图	GC 信息，对象分配位置
空闲对象	——	——

在程序执行过程中，对象被使用，其行为及逻辑产生多种可观测到的信息，如图 1 所示。

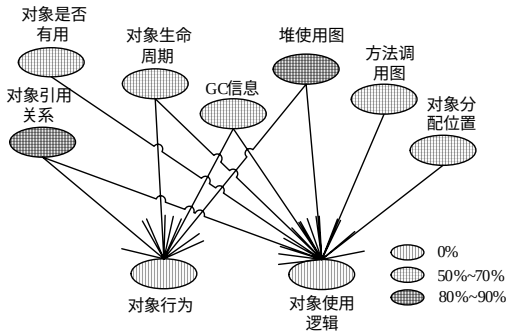

图 1　检测及修正内存低效使用所用到的信息

从图 1 中可以看到，所有信息都是 JVMPI 可观察的信息，它们依赖于对象行为及其使用逻辑。图中还列出了这些可观察信息在内存低效使用的检测和修正中被利用的程度。80%~90%的研究和工具都利用对象间的引用关系和堆使用图；50%~70%的研究和工具利用对象生命周期和 GC 信息检测并修正内存低效使用。

虽然这些可观察信息在一定程度上反映了对象行为和对象使用逻辑，但还不够全面。如果要对内存低效使用进行深入检测和修正，就必须全面利用对象行为和对象使用逻辑信息，否则不完整的输入信息不可能支持获得精确的检测和修

（下转第 91 页）