计算机科学2000Vol.27№.3

Java芯片系统存储管理的设计和实现*)

Java Chip System: the Design and Implemetation of Memory Management

秦莹 陈虎 戴葵 杨晓东

(国防科技大学计算机学院 长沙410073)

Abstract Java chip system is a platform of Java applications, which consists chip and its operation system. Java chip implements JVM(Java virtual machine) instruction set, while os manages hardware such as memory, I/O etc, and classloader, thread manager are also included. Memory management is one part of chip system, it not only supports the execution of Java instruction, but supplies the basic support to the Java chip system. In this paper, we introduce the design and implementation of memory management in Java chip system.

Keywords Java chip system, Memory management

Java是一种面向对象、通用、支持并发的程序设计语言，自1995年推出以来，其应用领域已经从主要面向Internet转向大型分布式软件和嵌入式系统。Java语言源程序(*.java)并不基于任何一个平台，而是基于一台抽象的机器——Java虚拟机(Java virtual machine—JVM)。JVM有自己的一套指令集并且使用独立的存储区域。编译后的代码称为类文件(*.class)。有相应的Java虚拟机规范、定义虚拟机指令集、类文件的格式、Java虚拟机的实现等标准，确保了类文件的可移植性。

现有的JVM多在宿主机操作系统上用软件实现，解释执行Java虚拟机指令。与软件实现JVM不同，Java芯片系统用芯片直接实现Java虚拟机指令集中的大部分指令，并在系统软件的配合下支持Java应用程序运行。它主要面向嵌入式系统，在当今网络计算机上也有较好的应用前景。

存储管理是Java芯片系统一个重要组成部分，其主要任务是分配和回收。分配是指为Java程序运行分配存储空间，包括分配运行前可确定的静态的程序空间和分配程序运行时动态申请的存储空间。回收是指将程序不再使用的存储空间变为可以重新分配的空间。由于Java语言没有提供显式释放在程序运行时动态申请的内存手段，需要由回收程序确定哪些程序空间不再使用，垃圾的自动回收也就成为存储管理的重要任务。

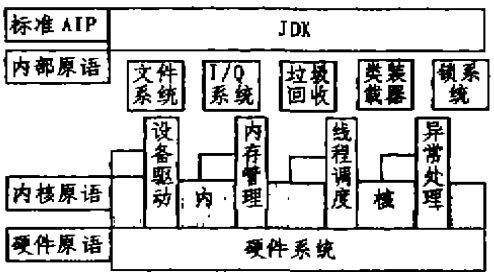

图1 Java芯片系统结构

1 Java芯片系统的特点

Java芯片系统由硬件和软件两部分组成：硬件是一块Java芯片，实现Java虚拟机指令集中大部分指令；系统软件，由管理硬件(如内存、I/O)的内存管理、I/O控制和支持类文件运行的异常处理、线程控制等内核程序组成，还包括类装载器，垃圾回收等高层程序。软硬件共同支持Java应用程序的运行。Java芯片系统的基本结构如图1所示。

*)本课题得到国家自然科学基金资助。秦莹 硕士生，主要研究方向：计算机体系结构、Java存储管理实现技术；陈虎 博士生，主要研究方向：计算机体系结构、Java实现技术等；戴葵 博士，副教授，主要研究方向：计算机体系结构、神经网络等；杨晓东 教授，博导，主要研究方向：并行计算机体系结构、故障检测等。

Java 芯片系统面向嵌入式系统，主要任务是支持类文件的运行，它遵循 JVM 规范，具有以下特点：

1 面向对象 JVM 指令集中有一类重要指令：对象指令，包括一系列创建对象指令(new 类指令)、访问对象域(putfield，getfield)指令，对象指令的执行需要内存布局上有相应的支持，如确定对象内部结构、系统类表等。不仅如此，Java 应用程序运行机制都是面向对象的。Java 类文件就是类的字节码表示，其中包含了域和方法。类文件由相应的类装载器装载，并链接到系统运行时的类表中，方法是类文件运行的基本单位，由 Java 虚拟机中 invoke 指令激活并执行。

与此相适应，Java 芯片系统中的各种数据结构都被看作是对象，芯片系统各部分都是面向对象的，也就是为对象提供服务。其中的存储管理也就是为对象提供空间服务，包括为对象分配存储空间以及回收无用对象的空间。

2 垃圾的自动回收 Java 程序在运行过程中会动态创建对象，由于 Java 中没有相应的释放对象指令，即使对象不再被使用，程序也不能显式释放。因此，系统必须能够识别无用对象，并回收它占用的空间，这个过程称为垃圾回收。芯片系统实现垃圾的自动回收，这是存储管理的重要任务。

3 实时性 Java 芯片系统主要面向嵌入式系统，这类系统都有一定的实时性要求，因此 Java 芯片系统能满足一定的实时性要求。

4 支持线程 虚拟机规范中指出方法是由线程执行的，并且虚拟机支持多线程同时运行。Java 芯片系统用硬件支持线程切换，并由相应的软件完成线程的调度。

2 Java 芯片系统中的存储管理

存储管理是芯片操作系统重要组成部分。不仅要负责 Java 芯片的内存管理，还要支持 JVM 指令中对象指令的实现，同时存储管理还要实现垃圾的自动回收。Java 芯片系统存储管理的结构如图2，阴影部分为存储管理的组成，包括三个部分：

1 内存管理。直接面向内存，在由芯片提供的一些硬件原语的支持下，完成内存分配和回收。这一层没有对象的概念，内存以块为单位进行管理。

2 对象分配。进行对象创建指令(new 类指令)的陷入处理，实现 new 类指令的功能，是存储管理的核心内容。它负责将对象创建指令的对象分配请求转化为底层的内存分配，并且完成对象结构的设置。

3 垃圾回收。进行面向整个内存的对象回收，其工作是扫描整个存储空间，标记出不再为程序使用的对象，并释放对象占用的空间。

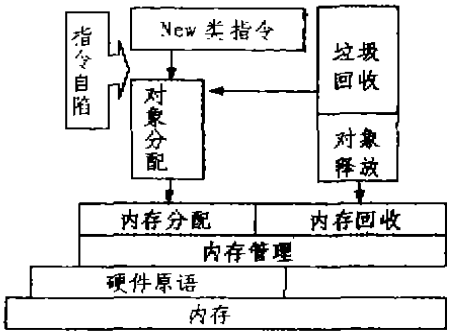

图2 芯片系统存储管理结构图

由图2可以看出，芯片系统的存储管理具有层次结构。存储管理一方面要支持创建对象指令的运行，是面向对象的；另一方面，存储管理要管理底层的内存。内存的组织分配没有面向对象的特点。存储管理底层的非面向对象性和高层的面向对象决定了存储管理的层次化。这也为存储管理的实现提供了灵活性，在保证界面不变的前提下，底层的内存分配、回收的算法可以灵活地改变；上层对象分配模块的设计和实现可以尽可能考虑适合类文件运行。

虚拟机规范并没有对垃圾回收的实现作具体规定，因此垃圾回收的算法和进行方式也可以根据芯片系统的特点进行设计。下面我们将对存储管理的设计和实现作具体介绍。

3 Java 芯片系统存储管理的设计和实现

3.1 内存管理

内存管理包括内存分配和内存回收。对 benchmark 程序的统计显示，程序运行时动态创建的对象多是一些小对象，占用空间多为几个字节；而在装载和链接类文件，建立运行的内存布局时，类装载器会创建类对象、线程上下文等大对象。这一现象表明，类文件的运行需要的各种对象大小不均，需要的内存块大小多种多样。

与此相适应，我们维护了多条空闲链，存放不同大小的空闲块，其中一条被称为公共空闲链，其链中的块以页为单位，初始时，系统可用内存的首地址放在公共空闲链中，相当于在公共空闲链中链入了 pagesize * pagenum 大小的一个内存块，其中 pagesize 为页大小而 pagenum 为系统内存包含的页数。其余的链存放小于一页的块，并且一条链中的块大小固定。空闲链的链头放在称为空闲链表(freelist)的数组项中。初始时，除了公共空闲链外，其余都为空链。

分配算法：

```
入口参数：要求分配的块大小(blocksize)
出口参数：分配的块的地址
1  如果 blocksize 大于一页，转到7
2  求出 blocksize 大小的块所属的空闲链号 i
3  如果第 i 号空闲链不为空，从链头取出一个空闲
   块，返回该空闲块地址；正常结束。
4  如果公共空闲链不为空
   ①从公共空闲链中分配一页；
   ②将页分为空闲链 i 中链接的块大小的块，链到空
     闲队列 i 中；
   ③从空闲队列 i 头中取出一块，返回该块的地址，
     正常结束。
5  如果 gcruned=1，抛出内存溢出错，终止。
6  进行垃圾回收，gcruned=1，转到4；
7  求出 blocksize 需要的页数 num；
8  如果公共空闲链不为空，从公共空闲链中分配连续
   的 num 页；返回起始地址，正常结束。
9  如果 gcruned=1，抛出内存溢出错，终止。
10 进行垃圾回收，gcruned=1，转到8；
```

内存管理引入了预分配的思想：对于小于一页的块，一旦请求，相应的空闲链就不为空，下次分配时，就可以直接从空闲链中取出一块；而且，一页中块的大小和块数是成反比的，这也符合 Java 程序运行时对象请求的特点：较小的对象请求的频度要高于大对象的请求。

空闲块释放算法

```
入口参数：要释放的块的地址(address)和大小(block-
          size)
出口参数：无
1  如果 blocksize 大于一页，转到5；
2  求出 blocksize 大小的块所属的空闲块链号 i；
3  将块按照地址递增的顺序链到空闲块链 i 中；
4  如果空闲链 i 中的块可以连成一页，将连成的页放
   入公共空闲链中；结束。
5  求出 blocksize 占的页数 num；
6  将 num 页按地址递增序链入公共空闲链，如果前
   后相连，将其合并为大块，结束。
```

释放算法虽然简单，但对于动态调整块大小的比率卓有成效。当有块释放回系统时，如果是页内分配的块，一旦发现块可以合并为一页，将其放入公共空闲队列，而后就可以将其分配为其他大小的块。这种动态的调整可以降低外部碎片的危害，进一步的措施可以保证相连的块合并为一个大块，满足分配请求，从而可以进一步降低外部碎片的存在。

3.2 对象创建指令的实现

对象是芯片系统的核心。对象的内部结构直接影响一些对象指令（如 putfield，getfield）的执行，也直接关系到垃圾回收的效率，是存储管理的基础。

芯片系统中对象由对象头和对象体两部分构成。对象头用于存储管理，也为对象指令的执行提供支持，对应用程序透明；对象体是对象的数据区，包括了对象所有的域，应用程序可以通过访问对象域（putfield，getefield）指令进行操作。对象的结构如图3所示。

在每个对象前面都有四个字的对象头，其数据结构如下所示：

```
struct ObjectHead
{
 int next   // 指向颜色队列中下一个对象；
 int pointerlength&mark; // 包含了对象标志和对象体内的结
                            构信息
 Classclass *class; // 指向该对象所属的类
 int length; // 对象的大小
 ;
```

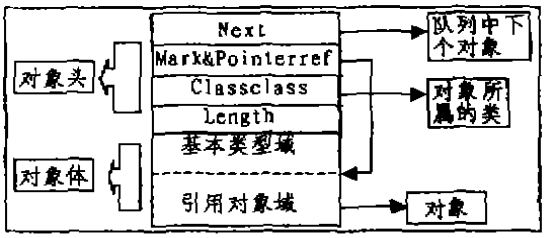

图3 对象内部结构

在 Java 虚拟机指令集中有前面所提到的 new 类四条创建对象指令：①new：创建一个对象；②newarray：创建一个基本类型数组对象；③newrefarray：创建一维引用类型数组对象；④*newmultiarray*：创建多维数组对象。这四条指令创建的是不同类型的对象，其实质是为应用程序对象分配存储空间。芯片系统中四条对象创建指令由软件完成。在代码执行过程中，当译码器遇到对象创建指令时，将引起一次自陷，主动陷入到存储管理中相应的对象分配方法中。每条对象分配指令有一个对应的对象分配方法。

对象分配方法的任务是分配对象空间，并且填写相应的对象头。例如对于 new 指令，对象分配方法首先为对象分配一块内存，然后将对象头的信息填写到内存相应的位置。对象头的信息通过指令参数获得。

数组是一种特殊的对象，在芯片系统中我们预先定义了基本类型数组类，这样一来，数组也就可以象一般对象一样创建。

芯片系统中，对象类型（也就是类）可在应用程序中定义，所以系统不可能预先定义引用类型数组类，因此我们在分配引用类型数组对象的方法中还必须创建相应的引用类型数组类对象。

创建对象指令用硬件实现起来非常复杂，而软件实现可大大降低硬件的复杂度，而且也可以达到系统要求。

3.3 垃圾回收的设计和实现

垃圾的自动回收是 JVM 的一个重要特点，既可以减轻程序员的编程负担，还可以减少错误的发生。

Java 芯片中的垃圾回收用一个独立的 GC 线程实现。GC 线程由系统在初始时创建。GC 线程在两个时机被激活：①系统中没有其他线程正在运行，此时被激

活的GC称为异步GC，它可以被其他线程中断；②运行中的线程要创建对象，但空间不够，此时激活的GC为同步GC，它是不可中断的。

垃圾回收可分为两个阶段。一是标记过程：GC线程遍历存储空间，标记所有可被系统中运行线程访问到的对象；二是对象回收过程，将未标记的对象（垃圾对象）占用的内存释放到内存空闲链中。

标记过程要遍历整个对象存储空间，这是一项耗时费力的工作。遍历从根集开始，根集的选取应保证正确性和高效性。正确性要求对每一个活对象（应用程序可访问到的对象），都存在一个根集到该对象的引用序列。Java芯片系统中的根集包括系统类表入口和线程队列。为了提高遍历的效率，我们在对象的组织上考虑了对检查的支持，在对象域的组织上，我们将引用域连续存放，这样就很容易识别出引用域。由于线程栈中也会出现引用，我们在对象类中对栈结构作了描述，遍历时，通过线程对象找到当前线程栈上的对象，而后就很容易识别引用域。这样比较容易标记所有线程可引用到的对象。

一般情况，垃圾回收标记过程应不可中断，因为中断期间运行的应用程序可能改变对象之间的引用关系。芯片系统中允许中断异步垃圾的回收，为保证正确性，芯片系统提供硬件的同步支持。在程序执行对象域写指令时，硬件自动检查对象头中的标志位。当标志位满足一定条件时，引起一次陷入，执行相应的处理方法，完成同步操作。

芯片系统的垃圾回收通过异步垃圾回收，有效地降低了同步垃圾回收进行的频度，从而提高了系统的实时性；而且通过由分配引起的同步回收，解决了异步垃圾回收浮动垃圾的问题，从整体上提高了系统性能。

总结 Java芯片系统面向嵌入式系统，其设计和实现具有较强的针对性，存储管理也不例外。我们在了解Java存储管理要求之后，结合Java芯片系统和应用特点，设计了存储管理方案，并进行了实现。由于针对性强，设计时可以比较灵活，且效果良好。

参考文献

1 Lindolm T, Yellin F. The Java Virtual Machine Specification. Addison-Wesley, 1996

2 Wilkinson T J. Kaffe 0.8.4 Source Code. Available at: http://www.kaffe.org

3 Agesen O, Detlets D. Finding References in Java Stacks. Available at: http://www.sun.com, also in OOPSLA'97 Workshop on Garbage Collectionand Memory Menagement, 1997

4 朱海滨. 面向对象. 长沙：国防科技大学出版社

5 Wilson P. Uniprocessor Garbage Collection Techniques

（上接第50页）

● 可扩展的定理基础，支持Z工具包并允许修改以适应新的应用领域；

● 灵活的用户界面，支持构造与程序推导的重用。

本文还对现有的精化工具进行了比较。现有工具大多不支持数据精化，自动化程度尚低，而且不支持大型复杂的软件精化，因此在精化工具的研究方面，还有许多工作要做。

参考文献

1 Carrington D, et al. Structured presentation of refinements and proofs: [TR-95-46]. Software Verification Research Centre, The University of Queensland

2 Hsiang J, et al. The term rewriting approach to automated theorem proving. Journal of Logic Programming, 1992

3 Utting M, Whitwell K. Ergo user manual: [TR-93-19]. Software Verification Research Centre, The University of Queensland, 1994

4 Robinson P, et al. Formalizing a hierarchical structure of practical mathematical reasoning. Journal of Logic and Computation, 1993

5 Vickers T. An overview of a refinement editor. In: Proc. of the 5th Australian Soft. Eng. Conf., 1990

6 von Wright J. Program refinement by theorem prover. In Till [Ti194]

7 Back R J R, et al. A program refinement environment. Ser A 139, Abo Akademi, 1992

8 Bloesch A, et al. The Cogito methodology and system. IEEE Computer Society Press, 1994