Java 虚拟机中异常机制实时性的研究及实现

王新雨,须文波,柴志雷
WANG Xin-yu,XU Wen-bo,CHAI Zhi-lei

江南大学 信息学院,江苏 无锡 214122
College of Information,Jiangnan University,Wuxi,Jiangsu 214122,China
E-mail:james.w1982@163.com

WANG Xin-yu,XU Wen-bo,CHAI Zhi-lei.Real-time extension for exception handling mechanism in JVM.Computer Engineering and Applications,2008,44(34):84-86.

Abstract: Exception handling is an important feature of modern programming languages.This paper introduces a method to enhance the real-time performance of the exception-handling in the JVM.This method is implemented through modifying the famous open-source JVM,SableVM under the Linux OS.The result shows that after enhancement,the Java Virtual Machine not only improves the efficiency of catching exception object when exception happened,but also makes the seeking time linear.
Key words: Java Virtual Machine(JVM);real-time;exception handling

摘 要:异常处理机制是程序设计语言的重要特征之一。讨论了对 Java 异常处理进行实时性改造的可行性和具体方法,并且在 Linux 平台上,实现了对开源 Java 虚拟机 SableVM 中异常处理机制的实时性改造。实验结果表明改进后的虚拟机在异常捕获时间趋于线性的同时,异常表查询效率也得到了提高。
关键词:Java 虚拟机;实时性;异常处理
DOI:10.3778/j.issn.1002-8331.2008.34.025 **文章编号**:1002-8331(2008)34-0084-03 **文献标识码**:A **中图分类号**:TP316.2

1 引言

尽管计算机系统的可靠性不断提高,但系统的失效现象仍然时有发生。如果程序设计人员不对各种异常现象进行有效地处理,当异常发生时,轻则可能使程序终止,重则可能丢失关键数据、破坏系统,甚至于造成灾难性事故。因此,人们希望程序设计语言具备方便、直接的异常处理机制。

传统的异常处理方法常常导致处理代码同程序的正常执行流混杂在一起。Java 语言则直接在语言中引进异常处理设施,提供更加结构化的异常处理机制。然而,传统的 Java 虚拟机中的异常处理不具有实时性。一个异常被捕获的时间在传统的 Java 虚拟机中是不确定的, 这将导致整个 Java 虚拟机处理发生异常程序的时间也是不确定的。所以说解决异常处理的实时性问题是 Java 虚拟机实时性改造过程中不能缺少的一个重要组成部分。

异常处理机制可以由硬件或者软件实现。虽然虚拟 Java 机异常处理机制已经在硬件(国产平台 COSIX)上被实现[1],但是却没有针对异常的实时性进行改造。很多文章都提出了对 Java 虚拟机的异常处理机制进行改进和扩展[2-3]的方法,以及在实时 Java[4]中对异常处理的要求[5],但是,对异常处理实时性的具体扩展方法以及具体实现仍缺乏详细论述。为此,针对异常的实时处理技术展开相关讨论, 并且在开源的 Java 虚拟机 SableVM 中实现了对异常实时性的改造。

2 异常以及异常处理

2.1 异常及异常处理概念

异常(Exception),是可被硬件或软件检测到的要求进行特殊处理的事件[6],对应着 Java 语言特定的运行错误处理机制,这种固定的机制用于识别和处理错误。例如,在程序执行中,由于应用程序使用系统资源而引发的运行错误就是异常的一种,而这种错误是应用程序本身无法预料的。

异常处理就是在异常引发后,应用程序能够自动转移到相应的异常处理模块中去进行一些尝试性修复处理,然后再决定程序走向,使应用程序能够正常运行、或降级运行或安全地终止应用程序的执行,以提高应用系统的可靠性。分析数据表明,对异常的不当处理会引起系统崩溃[7]。

2.2 Java 异常处理机制

当异常发生时,一个代表该异常的对象被创建并且在导致该异常的方法中被抛出。Java 运行时系统可以自动抛出一个异常。如果需要手动抛出一个异常,用关键字 throw。

异常可以由用户代码捕获并处理, 或由 Java 运行时系统捕获并处理。将有可能产生异常的代码段包含在一个 try 块中。如果在 try 块中发生异常,异常被抛出。程序员的代码可以捕捉

基金项目:国家自然科学基金(the National Natural Science Foundation of China under Grant No.60703106)。
作者简介:王新雨(1982-),硕士生,主要研究方向为嵌入式系统、实时系统;须文波(1946-),男,博士生导师,主要研究方向为人工智能、计算机控制技术、嵌入式操作系统;柴志雷(1975-),男,副教授,主要研究方向为嵌入式及实时系统。
收稿日期:2007-12-18 **修回日期**:2008-03-07

这个异常(catch)并且用某种合理的方法处理该异常。

如果当前程序计数器在异常表入口所制定的范围内,而且所抛出的异常类是该入口所指向的类(或为指定类的子类),那么该入口即为所搜寻的入口(异常被捕捉)。Java 虚拟机按照每个入口在表中出现的顺序进行检索。当遇到第一个匹配项时,虚拟机将程序计数器设为新的 PC 指针偏移量位置,然后从该位置继续执行。如果没有找到匹配项,虚拟机将当前栈桢从 Java 栈中弹出后,将现在栈顶桢设置为当前栈桢,返回搜索。这就使得虚拟机在该方法中再次执行同样的搜寻异常表的操作。

Java 中的异常分为已被检测的异常和未被检测的异常。两者之间的区别在于:对于未检验的异常来说,即使可能抛出此异常,方法中也可以不在 throws 子句中声明对此异常的处理。但是对于已被检测的异常来说,如果要抛出此异常,方法中一定要在 throws 子句中声明对此异常的处理。

3　Java 虚拟机异常机制实时性改造的设计

本文通过对 Java 栈进行结构化优化来提高 Java 异常处理的实时性。以下将结合 SableVM 对改进过程进行详述。SableVM 是一个健壮,简洁,易于维护与扩展,非常轻便并符合规范的 Java 虚拟机。

3.1　SableVM 中的异常处理

SableVM 可以处理的系统异常有:

ArithmeticException

ArrayIndexOutOfBoundsException

ArrayStoreException

ClassCastException

IllegalMonitorStateException

NegativeArraySizeException

NoSuchFieldException

NoSuchMethodException

NullPointerException

InterruptedException

InliningException

如果某方法执行时产生了异常,则异常就会被抛出(异常抛出),同时产生并释放一个标志异常抛出的信号(signal)。此信号的 signo 对应着被抛出异常的异常类型代号。通过 Linux 内核以及本地接口将 signal 捕捉,进入查找异常表程序(解释器),查找过程如 2.2 节所述。

3.2　SableVM 结构上和实时性上的不足

在 SableVM 所提供的模式中,并不能保证异常查找的实时性。其原因在于:

(1)结构上的不足

虽然虚拟机在扫描 class 文件时已经为会抛出异常的方法创建了异常表,但是方法与方法之间异常表相互并不可见,如图 1。可知,当所抛出的异常在方法栈的当前栈桢的异常表中找不到其匹配项时,当前桢会被弹出,进入下一个栈桢的异常表查找匹配项。这个过程本身不可预测,原因在于虚拟机本身也不知道何时才能找到异常的匹配项,造成了查找异常表的时间不确定。

(2)出错跳转

在 SableVM 中,捕捉到抛出异常之前,还要对异常本身进行检查。包括:

_svmm_new_native_local

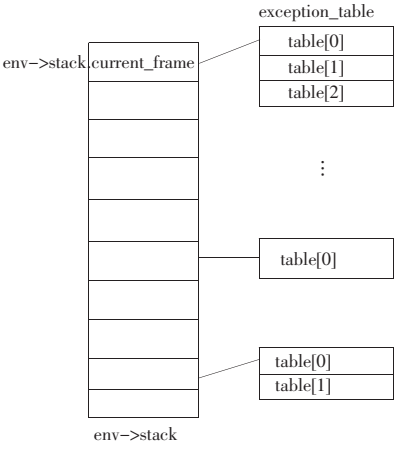

图 1　SableVM **中方法栈的结构**

//为中间量本身分配空间(被抛出异常不能直接与异常表中的项相比较,需要要通过变量)

_svmf_resolve_CONSTANT_Class

//将被抛出异常类型与栈顶方法异常表中的异常类型进行比较

_svmf_link_type

//连接栈顶方法异常表中的异常和抛出的异常

在进入异常表的搜索查找后,虚拟机会在每一次对异常表的查找之前进行以上三种检查操作,以防找到异常匹配项以后进行处理的过程中出现错误(内存出错,连接错误等错误)。如果以上三种检查过程中任一种不能返回正常值,虚拟机就会弹出当前桢。如果被抛出异常的匹配项恰好在当前桢中,虚拟机也会找不到正确的匹配项。

3.3　针对 SableVM 不足做出的改进

Java 的异常机制的实时性改造是通过对 Java 栈的结构性优化实现的。

在 Java 程序的运行过程中,每一个线程都有自己的 Java 栈。Java 栈以桢为单位保存线程的运行状态。栈桢中包含许多虚拟机运行时所需要的信息,并且可以利用桢来存储参数、局部变量、中间运算结果等数据。最重要的是栈桢中的方法信息中,包括本方法的常量池起始地址、异常表和 pc 指针等信息。透过这些信息虚拟机可以进行异常的捕捉还有异常的处理。

对 Java 栈的改进着重于结构。SableVM 的 Java 栈是一个传统意义上的顺序栈,内存为其分配了连续的内存空间,基本操作为对栈桢的压栈及出栈。这种结构的缺点在于,无法对压入栈的栈桢所属方法的异常表进行即刻检验,从而达到估计顺序扫描其内容的时间的目的,也就无法估计在对抛出异常捕捉的整个过程中搜索异常所用的时间。

由于 SableVM 中规定的栈桢内容非常的丰富,不但包括了此桢所携带的方法,还有桢的偏移量、方法是否存在同步锁以及 PC 等信息。可以利用栈桢中的信息对方法的异常表进行优化。

(1)解决结构的问题

修改原有的线程共用区域(env)的结构,添加如下结构以及标志:

jboolean frame_finish_flag ;//新异常表重置的条件,如果此值为真,就要将新异常表重置。

struct _svmt_Huge_Exception_Table *exception_table_mine ;//记录新异常表的头地址

```
struct _svmt_Huge_Exception_Table *table_top ;//记录新异常表的操作点
struct TIME
{
jint time; //记录进入查询的次数
jint table_length;//记录当前新异常表的长度,用于估计时间
};
```

(2)构造新的数据结构

一个双向链表用以代替原来的异常表。放在进程共用的区域中,所以每个属于这个进程的线程都可以查找这个新异常表。

```
struct _svmt_Huge_Exception_Table
{
struct _svmt_exception_table *exception_table ;//用于记录当前需要查询的异常表单位
struct _svmt_stack_frame_struct *frame;//记录当前异常表属于的栈桢地址
struct _svmt_Huge_Exception_Table *previous ;//用于链表的连接
struct _svmt_Huge_Exception_Table *next ;//用于链表的连接
};
```

新数据结构叫它"新异常表"。它产生的过程是:在进入查询之前,如果 env 的 frame_finish_flag 为真,说明有新的栈桢加入,并且有可能产生新的异常表,异常表需要重置。从栈顶开始顺序扫描栈中所有的栈桢, 如果扫描到的栈桢附带异常表,就把它的异常表设置成 struct _svmt_Huge_Exception_Table 的数据形式在 env->exception_table_mine 处加到新异常链表中,如果异常表的数目大于 1,顺序将 table[0],table[1],…加进新异常链表(图 2),并保证扫描新表的时候,扫描的顺序依旧。如果,当前扫描的栈桢没有异常表,也要将当前栈桢的异常表存储为 struct _svmt_Huge_Exception_Table 的形式,只不过 exception_table 设置为 0x0,表示这个栈桢没有异常表。

改造以后的查询过程如图 2 所示。

查询过程: 从 env->table_top 处取出一个新异常表单位(左端),如果这个 table[0]和抛出的异常匹配,就把 pc 设置成 table[0].pc 进行异常的处理。如果搜索 frame0 的 table[0]和所要查找的异常表不能匹配,就将整个单位删除,进入下一个单位查找 frame1 的 table[0]。若还是没有匹配,删除这个单元,继续查询,直到找到所要找的异常表。

结构优点:由于异常表已经被重新储存在新的数据结构中,而这个新的结构是线性排列的。这样搜索异常表就变成搜索线性表,而此操作的时间复杂度是 O(N),N 是此线性表的长度。查询的时间复杂度(当前所查异常表的长度)是一个确定的值,并且已经被保存在 TIME 结构体的 table_length 中。

当遇到检查异常出错的情况时,新的数据结构只去掉那个出错的异常表所属的 struct_svmt_Huge_Exception_Table 结构,而对同栈桢的其他异常表没有影响。

4 实验结果及分析

对用户自定义异常的处理过程和系统异常处理过程的区别在于虚拟机对它们所做的处理过程不同。但是对于捕捉异常的过程,对这两种不同的异常来说,都是通过查找异常表来实现的。图 3~图 5 显示的效果是修改虚拟机前后查找异常表的时间差异(对异常处理的时间不包括在显示的范围内)。

本实验意在证明, 在相同的运行条件下, 改进过后的 SableVM 不但在实时性上有所提高而且还提高了程序的运行效率。异常从抛出到捕捉所用的时间要在 3 种情况下进行比较,分别是:

(1)在搜索一次异常表就可以找到所要查找的异常并且进入此异常处理,比较如图 3。

(2)在搜索二次异常表就可以找到所要查找的异常并且进入此异常处理,比较如图 4。

(3)在搜索三次异常表就可以找到所要查找的异常并且进入此异常处理,比较如图 5。

从图中可以看出,改进后的 SableVM 在运行时间上比为改进之前有较大提高。从图 3~图 5 中可以看到, 修改前的 SableVM 抛出异常的时间没有规律性, 并且在随机进行的 25 次测试中,时间跨度很大。修改后的 SableVM 抛出异常的运行时间有了很大的改进, 不但在随机 25 次实验中基本都把时间保持在一定范围内,而且抛出的时间基本可以看成定值(虽然对不同的异常类型抛出的时间不尽相同,但是对于每种异常的处理时间可以准确估计)。

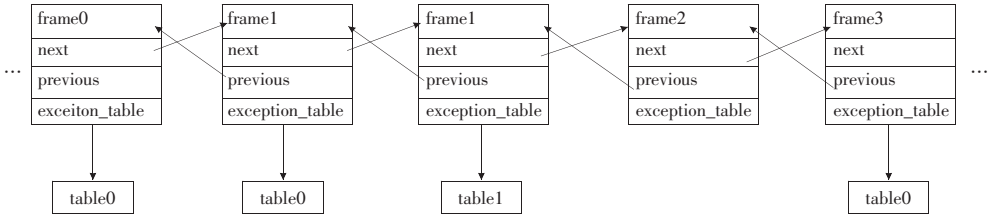

图 2 huge_exception_table 的查询过程

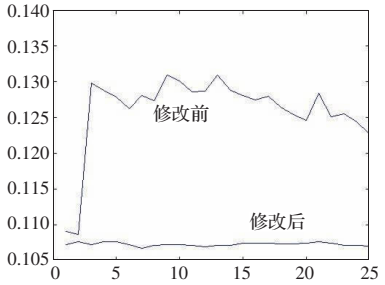

图 3 搜索一次异常表的时间差异

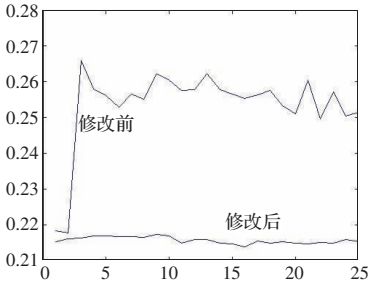

图 4 搜索二次异常表的时间差异

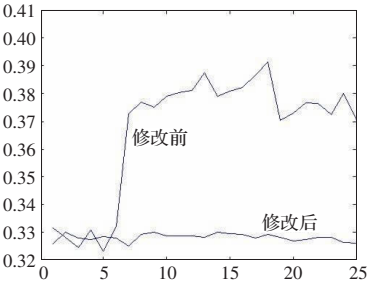

图 5 搜索三次异常表的时间差异

(下转 131 页)