[19] 中华人民共和国国家知识产权局

[51] Int. Cl[7]
G06F 12/02

[12] 发明专利申请公开说明书

[21] 申请号 00816416.9

[43] 公开日 2003 年 3 月 12 日

[11] 公开号 CN 1402847A

[22] 申请日 2000.10.27 [21] 申请号 00816416.9
[30] 优先权
[32] 1999.10.28 [33] SE [31] 9903890 - 3
[86] 国际申请 PCT/SE00/02096 2000.10.27
[87] 国际公布 WO01/31455 英 2001.5.3
[85] 进入国家阶段日期 2002.6.5
[71] 申请人 阿佩尔虚拟机公司
地址 瑞典斯德哥尔摩
[72] 发明人 乔基姆·达尔斯泰特
[74] 专利代理机构 北京市柳沈律师事务所
代理人 马 莹 邵亚丽

权利要求书 1 页 说明书 4 页 附图 1 页

[54] 发明名称 把无用方法收集到垃圾箱中的方法

[57] 摘要

一种在使用虚拟机时提高数据处理应用程序的有效性的方法，其中该程序包括许多个方法，即许多个程序段，它们都存储在所用计算机的存储器中，和其中该程序利用了垃圾收集过程。该发明的特征在于，在第 1 步骤中，根据由此所需的方法分析所有所谓的线程堆栈；在第 2 步骤中，使所述所需方法的每一个都得到更新，其中正在调用更新以前的方法被调用更新后的方法所取代；在第 3 步骤中，删除所有未更新的方法，和由该程序清理相应的存储空间。

ISSN1008-4274

1. 一种在使用虚拟机时提高数据处理应用程序的有效性的方法，其中该程序包括许多个方法，即许多个程序段，它们都存储在所用计算机的存储器中，和其中该程序利用了垃圾收集过程，其特征在于，在第 1 步骤中，根据由此所需的方法分析所有所谓的线程堆栈；在第 2 步骤中，使所述所需方法的每一个都得到更新，其中，正在调用更新以前的方法被调用更新后的方法所取代；和在第 3 步骤中，删除所有未更新的方法，和由该程序清理相应的存储空间。

2. 根据权利要求 1 所述的方法，其特征在于，当程序正在运行时，一次让一个线程停止；把用于停止线程的方法转移到一个列表中，和此后，重新启动线程；更新和存储所述列表中的方法；在按照所述方式处理完所有线程之后，使所有线程同时停止；删除没有被更新的所有方法，和用更新的方法重新启动所有线程。

3. 根据权利要求 1 或 2 所述的方法，其特征在于，当利用锁定机制时，在所述第 1 步骤之前的步骤中识别最有效的锁定机制；和更新利用如此确定的锁定机制的那些方法。

4. 根据权利要求 1、2 或 3 所述的方法，其特征在于，当利用不同的垃圾收集算法时，在所述第 1 步骤之前的步骤中确定各个对象的分配和寿命，然后，识别最有效的垃圾收集算法；更新构成所需垃圾收集算法的方法；和删除其余的垃圾收集算法。

把无用方法收集到垃圾箱中的方法

本发明涉及一种提高数据处理应用程序的有效性的方法。

更准确地说，本发明涉及提高虚拟机中的数据处理速率，尤其与 Java 程序语言有关。

虽然下面主要针对 Java 程序语言描述本发明，但是，本发明不限于 Java 程序语言，而是可以与许多程序语言一起应用。

本方法打算用在程序的自适应优化上。在自适应优化中，当运行程序时，重构程序和优化程序的各个不同部分。运行的程序越长，所需的存储空间就越大，因此，提高数据处理能力的一般性问题在于迅速创建新的存储场地。

Java 和其它动态程序语言包括自动存储器管理。这意味着程序员无需关心存储器已经使用的那些部分。虚拟机有时进行所谓的垃圾收集（garbage collection），大体上指的是虚拟机扫描整个存储器，找出哪些对象还存储在存储器中和哪些对象程序不再访问。归还存储器的这些部分供以后使用。

Java 还包括有关所谓线程管理方法的方法。因此，Java 包含了支持或模拟两个或更多个程序的同时处理的系统。线程管理可以分成两个部分。一个部分涉及到以控制方式构造不同线程的方式。另一个部分涉及到哪些线程将被运行和哪些线程将是被动的和等待着被运行。

为了进一步提高有效性和让程序清理占用的存储空间，仅仅根据各个对象来优化存储器是不够的。

本发明解决了这个问题。

因此，本发明涉及当使用虚拟机时，提高数据处理应用程序的有效性的方法，其中所述程序包括许多个方法，即许多个程序段，它们都存储在所用计算机的存储器中，和其中所述程序利用了垃圾收集过程，其中这种新发明的方法的特征在于，在第 1 步骤中，根据由此所需的方法分析所有所谓的线程堆栈（thread stack）；在第 2 步骤中，使所需方法的每一个都得到更新，其中，正在调用更新以前的方法被调用更新后的方法所取代；和在第 3 步骤中，删除所有未更新的方法，其中，由所述程序清理相应的存储空间。

现在参照如附图所示的本发明示范性实施例，在一定程度上比较详细地

描述本发明，在附图中：

图 1 是方块图；

图 2 显示了旧方法；和

图 3 显示了根据本发明更新的方法。

图 1 显示了 Java 虚拟机 JVM，无论操作系统是 WinNT、LINUX、Solaris、还是一些其它系统，这种虚拟机 JVM 都可以用于运行不同的数据程序 1、2、3。如上所述，尽管 Java 程序语言是非常流行的程序语言，但是，本发明不限于这种语言，而是可以应用于所有面向对象的和与平台无关的相应程序语言。

因此，本发明涉及当使用虚拟机时，提高数据处理应用程序的有效性的方法，其中，该程序包括许多个方法，即许多个程序段，它们都存储在所用计算机的存储器中，和其中，该程序利用了垃圾收集进程。

大家知道，垃圾箱收集各个对象，随之删除当前不再使用的对象，从而使相应的存储空间得到清理。

在大多数系统中，许多方法，即许多程序段，被使用一次或数次，或者这些方法在很短的时间内得到应用，然后就不再被使用了。

在 Java 和相应程序的情况中，装入新方法之后，旧方法就再也没有用了。

此外，自适应优化导致存储在存储器中的方法的优化或再优化，其中留下的旧方法就再也没有用了。

当优化锁定机制选择和垃圾收集选择时，有必要用新机制取代使用旧机制的所有已使用过的方法。

根据本发明，在新发明的方法的第 1 步骤中，根据所需的各个方法分析所有所谓的线程堆栈。在第 2 步骤中，更新所需方法的每一个，其中，正在对所述更新以前的方法的调用被对更新方法的调用所取代。在第 3 步骤中，删除所有未更新的方法，和由所述程序清理相应的存储空间。

这个过程不仅的确清除了无用方法，而且致使已经更新的那些方法得到重排，以便把方法的调用直接指向更新的方法，而不是通过不再使用的旧方法来进行。

这显示在图 2 和 3 中，其中图 2 显示了旧方法，和图 3 显示了所用的更新方法。在图 2 中显示了三个方法 foo、apa 和 bar。foo 从存储器地址 4711 开始，apa 从地址 4714 开始，和 bar 从地址 4720 开始。

线程堆栈的分析表明，只有方法 foo 和 bar 被使用，因此，foo 和 bar 不再引用方法 apa。

把方法 foo 和 bar 更新成图 3 所示的那些方法。在这种情况中，方法 foo 和 bar 精确地重建起来，尽管不同之处在于，各个方法获得了新地址，和此时，foo 对 bar 的调用指向新的 bar 地址 4903。

删除了所有旧方法，即图 2 中的方法，以前被这些方法占用的存储空间也空出来供未来使用。

当进行对象的垃圾收集时，一般来说，在进行垃圾收集的时候，程序停止运行。在垃圾收集和删除了不再使用的对象之后，程序重新启动运行。

当应用本发明时可以使用这样的方法。

但是，最好用如下的方法来取代。

当实施新发明的方法时，在程序正在运行的时候停止一个线程，与此同时，把用于停止线程的方法转移到一个列表中，然后，重新启动该线程。接着，更新和存储该列表中的方法。在按照这种方式处理完所有线程之后，使所有线程同时停止，即，以便更新与所关心的线程相关的所有已用方法。删除没有被更新的所有方法，和用更新方法重新启动所有线程。

由于更新是断断续续进行的，这种方法避免了停止运行程序的需要。

如上所述，在 Java 和相应的语言中利用了锁定机制。可以选择不同的锁定机制。重要的是，选择在避免与另一个线程同时访问给定对象的多于一个的线程的出现方面最为有效的锁定机制。

同步问题出现在几个线程想要访问同一个对象或资源的时候。为了在 Java 程序语言中解决这个问题，每个线程尽力达到资源锁定。可以以各种各样的方式利用资源锁定机制。不同锁定机制的有效性取决于线程如何致力于获得同步资源。

根据优选实施例，当利用锁定机制时，在所述第 1 步骤之前的步骤中识别最有效的锁定机制，和更新利用如此确定的锁定机制的方法。

至于垃圾收集算法，也需要加以选择。许多面向对象的语言都利用垃圾收集。这意味着，程序员无需明确地告诉系统不再需要某个对象了。系统负责这种检测，和收回被对象占用的存储器部分。为了有效实现这种检测和收回，已经提出了许多不同的算法。已经证明，不同的算法最好适合于不同的应用程序。选择最适合于正在运行的应用程序的垃圾收集算法对于取得与所

关心的程序有关的最大执行速率是非常有意义的。

根据本发明的另一个优选实施例，当利用不同的垃圾收集算法时，在所述第 1 步骤之前的步骤中确定各个对象的分配和寿命，此后，使最有效的垃圾收集算法得到识别，和更新构成所需垃圾收集算法的方法，然后删除其余的垃圾收集算法。

优选实施例的应用提供了优化代码、线程和存储器管理的高效方法，其中，一般特征在于方法的识别和更新，以便不把无用方法装入系统中。

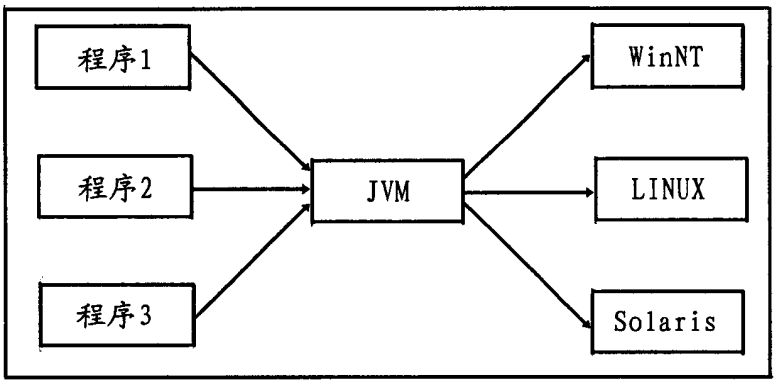

图 1

```
foo( )
{
   x;              4711:x
   bar( );         4712:call 4720
   y;              4713:y
}

apa( )
{
                   4714:..
                   4715:..
                   4716:..
                   4717:..
                   4718:..
}                  4719:..

bar( )
{
    .
}                  4720:..
```

图 2

```
foo_ny( )
{
    x;             4900:x
    bar( );        4901:call 4903
    y;             4902:y
}

bar( )
{
                   4903:..
}
```

图 3