

[19] 中华人民共和国国家知识产权局

[51] Int. Cl.  
G06F 12/02 (2006.01)



# [12] 发明专利说明书

专利号 ZL 00816416.9

[45] 授权公告日 2008 年 4 月 16 日

[11] 授权公告号 CN 100382047C

[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/SE2000/002096 2000.10.27

[87] 国际公布 WO2001/031455 英 2001.5.3

[85] 进入国家阶段日期 2002.6.5

[73] 专利权人 BEA 系统公司

地址 美国加利福尼亚州

[72] 发明人 乔基姆·达尔斯泰特

[56] 参考文献

US5873105 1999.2.16

US4797810 1989.1.10

US5293614 1994.3.8

审查员 郑 红

[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 4 页 说明书 5 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和系统

[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，分析所有线程堆栈，以确定哪些方法是所述线程堆栈需要的方法；在第 2 步骤中，更新每一个需要的方法，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并释放由所述未更新的方法所占用的存储空间。

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，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，分析所有线程堆栈，以确定哪些方法被所述线程使用；

在第2步骤中，通过创建每个所述使用的方法的拷贝，更新每个所述使用的方法，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，删除所有旧的方法并释放由所述旧的方法所占用的存储空间；

其中在所述第1步骤中，当程序正在运行时，让线程一次一个地被暂停；传输由所述暂停的线程使用的方法到一列表；重新开始被暂停的线程；

在所述第2步骤中，更新和存储所述列表中的方法；

在所述第3步骤中，当所有线程的第1和第2步骤的处理完成时，让所有线程同时停止；以及

删除所有旧的方法，并用更新的方法重新开始这些线程。

2. 根据权利要求1所述的方法，其中，所述虚拟机是Java虚拟机。

3. 根据权利要求1或2所述的方法，在使用锁定机构时，包括步骤：

在所述第1分析步骤之前，识别最优锁定机构；以及

在所述更新步骤中，对使用所识别的锁定机构的那些方法进行更新。

4. 根据权利要求1或2所述的方法，在使用不同的垃圾收集算法时，包括步骤：

在所述分析步骤之前，确定各个程序对象的分配和寿命；以及识别最优垃圾收集算法；

在所述更新步骤中，对构成所识别的垃圾收集算法的方法进行更新；以及

在所述删除步骤中，删除其余的垃圾收集算法。

5. 根据权利要求3所述的方法，在使用不同的垃圾收集算法时，包括步骤：

在所述分析步骤之前，确定各个程序对象的分配和寿命；以及识别最优垃圾收集算法；

在所述更新步骤中，更新构成所识别的垃圾收集算法的方法；以及在所述删除步骤中，删除其余的垃圾收集算法。

6.一种用于对计算机或虚拟机的存储空间内的方法的分布进行优化的方法，包括步骤：

分析所有线程堆栈，以便确定哪个方法被线程使用；

通过创建每个所述使用的方法的拷贝，对每个所述使用的方法进行更新，并以对所更新的方法的调用来替换对每一个使用的方法的调用；以及

删除所有旧的方法，并释放由这些旧的方法占用的存储空间；

其中，所述分析步骤包括如下子步骤：对所有线程

当程序正在运行时，一次一个地暂停每一个线程，

传输被暂停的线程使用的方法到一更新列表，和

重新开始被暂停的线程；

并且

其中所述更新每个所述使用的方法的步骤包括步骤：

更新并存储所述更新列表中的方法；以及

其中所述删除步骤包括如下子步骤：

当所有线程的处理完成时同时停止所有线程，

删除所有旧的方法，和

用更新的方法重新开始所述线程。

7. 根据权利要求 6 所述的用于优化计算机或虚拟机的存储空间中的方法的分布的方法，在使用锁定机构时，包括步骤：

在所述第 1 分析步骤之前，识别最优锁定机构；以及

在所述更新步骤中，对使用所识别的锁定机构的那些方法进行更新。

8. 根据权利要求 6 所述的用于优化计算机或虚拟机的存储空间中的方法的分布的方法，在使用不同的垃圾收集算法时，包括步骤：

在所述分析步骤之前，确定使用的对象的分配和寿命，以及识别最优垃圾收集算法；

在所述更新步骤中，还对构成所识别的垃圾收集算法的方法进行更新；以及

在所述删除步骤中，删除其余的垃圾收集算法。

9. 根据权利要求 7 所述的用于优化计算机或虚拟机的存储空间中的方法

的分布的方法，在使用不同的垃圾收集算法时，包括步骤：

在所述分析步骤之前，确定使用的对象的分配和寿命，以及识别最优垃圾收集算法；

在所述更新步骤中，还对构成所识别的垃圾收集算法的方法进行更新；以及

在所述删除步骤中，删除其余垃圾收集算法。

10. 一种用于对计算机或虚拟机的存储空间中的方法进行重排的方法，其中所述虚拟机包括一个或多个线程，包括步骤：

在所述虚拟机的存储空间中原始分布的方法中，确定对要被更新的所述方法的选择，其中所述确定步骤包括分析线程堆栈以确定当前所述线程需要的那些方法；

创建更新的方法，并以对所述更新的方法的调用替换对选择的每一个方法的调用，其中所述创建更新的方法的步骤包括一次停止所述线程一个并在备用存储器地址再创建所述停止的线程所需方法的拷贝；以及

删除原始的方法，并释放由删除的方法占用的存储空间的部分。

11. 一种对计算机或虚拟机中的存储空间中的方法进行重排的方法，包括步骤：

在所述存储空间中原始存在的方法中，确定要保留的所述方法的子集，所述确定步骤包括分析线程以确定当前所述线程需要的那些方法；

创建要保留的方法子集的拷贝，并将对子集中的每个原始方法的调用修改为对那个方法的新的拷贝的调用；以及

删除所有其他方法并释放由所述删除的方法占用的存储空间。

12. 一种优化计算机或虚拟机的存储空间中的方法分布的方法，包括步骤：

从多个原始方法选择一组将要更新的方法；

更新所述组的方法，包括创建从所述要更新的组拷贝的新的方法组，并以对所述新的方法组的相应调用代替对原始方法组的调用；和

删除在多个原始方法中的所有方法，并释放由删除的方法占用的存储空间的部分。

13. 根据权利要求 12 所述的方法，还包括步骤：

在更新所述方法之前停止所述线程，和

在所述方法被更新后重新开始所述线程。

## 把无用方法收集到垃圾箱中的方法和系统

### 技术领域

本发明涉及一种提高数据处理应用程序的有效性的方法和系统。

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。

### 背景技术

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。

### 发明内容

本发明解决了这个问题。

因此，本发明涉及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，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，分析所有

线程堆栈，以确定哪些方法被所述线程使用；在第2步骤中，通过创建每个所述使用的方法的拷贝，更新每个所述使用的方法，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，删除所有旧的方法并释放由所述旧的方法所占用的存储空间；其中在所述第1步骤中，当程序正在运行时，让线程一次一个地被暂停；传输由所述暂停的线程使用的方法到一列表；重新开始被暂停的线程；在所述第2步骤中，更新和存储所述列表中的方法；在所述第3步骤中，当所有线程的第1和第2步骤的处理完成时，让所有线程同时停止；以及删除所有旧的方法，并用更新的方法重新开始这些线程。

本发明还提供一种用于对计算机或虚拟机的存储空间内的方法的分布进行优化的方法，包括步骤：分析所有线程堆栈，以便确定哪个方法被线程使用；通过创建每个所述使用的方法的拷贝，对每个所述使用的方法进行更新，并以对所更新的方法的调用来替换对每一个使用的方法的调用；以及删除所有旧的方法，并释放由这些旧的方法占用的存储空间；其中，所述分析步骤包括如下子步骤：对所有线程当程序正在运行时，一次一个地暂停每一个线程，传输被暂停的线程使用的方法到一更新列表，和重新开始被暂停的线程；并且其中所述更新每个所述使用的方法的步骤包括步骤：更新并存储所述更新列表中的方法；以及其中所述删除步骤包括如下子步骤：当所有线程的处理完成时同时停止所有线程，删除所有旧的方法，和用更新的方法重新开始所述线程。

#### 附图说明

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。

#### 具体实施方式

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, 该程序包括许多方法, 即许多程序段, 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。

当优化锁定机构选择和垃圾收集选择时, 有必要用新机构取代使用旧机构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不使用的方法, 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。在这种情况下, 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构。可以选择不同的锁定机构。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构。

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构。不同锁定机构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，当利用锁定机构时，在所述第 1 步骤之前的步骤中识别最有效的锁定机构，和更新利用如此确定的锁定机构的方法。

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。

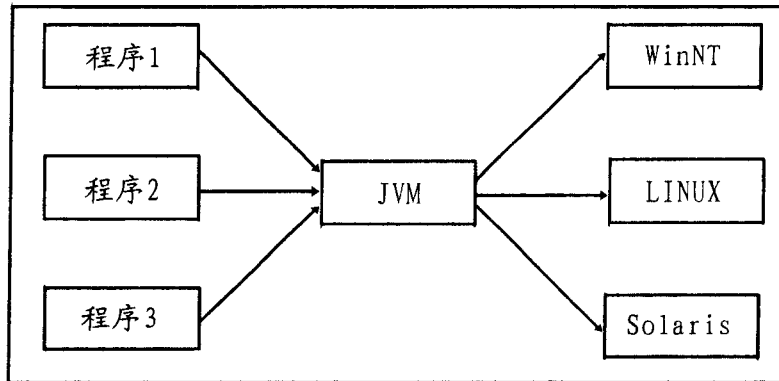


图 1

```
foo ( )
{
  x;          4711: x
  bar ( );   4712: call 4720
  y;         4713: y
}
apa ( )
{
          4714: ...
          4715: ...
          4716: ...
          4717: ...
          4718: ...
          4719: ...
}
bar ( )
{
}
          4720: ...
```

图 2

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

图 3