

[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 9/42 (2006.01)



[12] 发明专利说明书

专利号 ZL 200410030276.0

[45] 授权公告日 2007 年 10 月 10 日

[11] 授权公告号 CN 100342329C

[22] 申请日 2000.2.2

[21] 申请号 200410030276.0

分案原申请号 00805920.9

[30] 优先权

[32] 1999.2.2 [33] US [31] 09/243107

[73] 专利权人 太阳微系统有限公司

地址 美国加利福尼亚州

[72] 发明人 J·E·施瓦贝 J·B·苏塞尔

[56] 参考文献

EP0588447A2 1994.3.23

EP0810522A2 1997.12.3

CN1152254A 1997.7.2

审查员 谢阿涟

[74] 专利代理机构 中国专利代理(香港)有限公司

代理人 吴立明 王忠忠

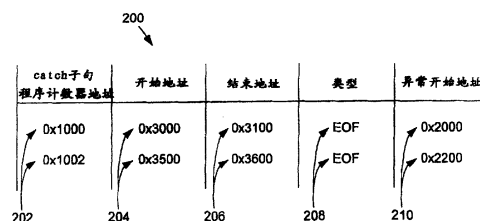
权利要求书 5 页 说明书 16 页 附图 7 页

[54] 发明名称

零开销异常处理

[57] 摘要

用于在由机器执行一个或多个类的方法的过程中处理可丢出异常的设备与过程，包括计算机实现的过程。每个方法包括一个定义与方法相关的异常处理器的异常处理器数组。该方法包括把用于所有方法的异常处理器数组结合成一个单独的异常处理器表。



1. 一种计算机实现的管理过程，用于在由机器执行一个或多个类中的一个或多个方法(530, 535, 540)的过程中管理可丢出的一个或多个异常，一个或多个方法(530, 535, 540)中的每一个方法包括可执行代码(531, 536, 541)和异常处理器数组(532, 537, 542)，所述异常处理器数组(532, 537, 542)定义与该方法(530, 535, 540)相关的异常处理器，该管理过程包含：

把用于两个或多个方法(530, 535, 540)的异常处理器数组(532, 537, 542)结合成一个单独的异常处理器表(556)。

2. 权利要求1的管理过程，还包括把用于一个类中所有方法的所有异常处理器数组结合成一个单独的异常处理器表。

3. 权利要求1的管理过程，还包括把用于所有类中所有方法的所有异常处理器数组结合成一个单独的异常处理器表。

4. 权利要求1的管理过程，还包括把用于一个Java包中的所有方法的所有异常处理器数组结合成一个单独的异常处理器表。

5. 权利要求1的管理过程，其中

类文件包括方法；和

所述结合所有异常处理器数组还包括：把用于一个类文件中的所有方法的异常处理器数组结合成一个单独的异常处理器表。

6. 权利要求1的管理过程，进一步包括：

在执行两个或多个方法之一的过程中，当丢出一个异常时，搜索异常处理器表，所述搜索包括在单独的异常处理器表中定位第一个匹配异常。

7. 权利要求6的管理过程，其中所述搜索还包括：

从异常处理器表中按顺序检索异常处理器入口；以及

针对第一个匹配异常处理器，检查每一个异常处理器的类型和范围。

8. 权利要求7的管理过程，进一步包括：

如果当前异常处理器不匹配并且是一个相关方法中被保护代码的最顶层的最后一个处理器，那么就停止该搜索。

9. 权利要求1的管理过程，其中类文件是Java类文件。

10. 权利要求 1 的管理过程, 其中一个或多个类中的一个或多个方法被分组在一个包中, 其中该包包括包含第一和第二部分的包数据结构, 该管理过程包括把异常处理器表存储在该包的第一部分中并且把所有方法存储在该包的第二部分中。

11. 权利要求 10 的管理过程, 其中所述结合还包括: 链接异常处理器数组, 包括按照预定义的顺序把每个异常处理器数组装载到包数据结构的第一部分中。

12. 权利要求 11 的管理过程, 其中预定义的顺序是根据存储在包数据结构的第二部分中的方法的顺序来确定的。

13. 权利要求 1 的管理过程, 其中该机器是实现在资源受限设备上的虚拟机。

14. 权利要求 13 的管理过程, 其中资源受限设备是智能卡。

15. 权利要求 14 的管理过程, 其中一个或多个类中的方法被分组在一个包中, 并且该包被安装到智能卡上。

16. 权利要求 15 的管理过程, 进一步包括:

创建一个包, 其中该包包括包含第一和第二部分的包数据结构, 该管理过程包括把用于两个或多个方法中的每一个方法的异常处理器数组链接到一个异常处理器表中, 把异常处理器表存储在包的第一部分中, 并把所有方法存储在包的第二部分中。

17. 一种用于最小化在执行程序时运行时栈所需要的存储量的方法, 该运行时栈在由机器执行程序的过程中为了存储一个或多个帧而在运行时间被保持, 其中每个帧包括指向调用该程序中的当前执行方法 (530, 535, 540) 的调用方法 (530, 535, 540) 的返回指针, 该最小化的方法包括:

把用于该程序中的一个或多个方法 (530, 535, 540) 的异常处理器信息结合到一个复合异常处理器表 (556) 中; 和

在一个或多个方法 (530, 535, 540) 之一执行的过程中, 当一个异常被丢出时, 搜索复合异常处理器表 (556), 以便在不需要在运行时栈上存储指向异常处理器信息的指针的情况下定位异常处理器信息。

18. 权利要求 17 的方法, 其中该指针是指向异常处理器信息的直接指针。

19. 权利要求 17 的方法, 其中该程序是 Java 程序。

20. 权利要求 19 的方法，其中该机器是实现 Java 虚拟机的虚拟机。

21. 权利要求 17 的方法，其中该程序包括方法包，并且其中虚拟机被实现在该包在其上被安装并被执行的资源受限设备上。

22. 权利要求 21 的方法，其中资源受限设备是智能卡。

23. 权利要求 21 的方法，进一步包括：

在安装时把该包注册在一个注册服务中，该注册服务维护指针和范围，该指针指示与给定包相关的复合异常处理器表在资源受限设备中的位置，该范围定义资源受限设备中与该包相关的方法被定位在的地址的范围。

24. 权利要求 23 的方法，其中

所述定位还包括：定位与当前执行方法相关的包，包括比较异常被丢出处的地址与登记在注册服务中的每一个包的范围；和

所述搜索还包括搜索与已定位的包相关的复合异常处理器表。

25. 一种把类文件转换为已转换的小应用程序以便在资源受限设备上执行的方法，该方法包括：

接收一个或多个类文件，该一个或多个类文件包括一个或多个方法 (530, 535, 540)，该一个或多个方法 (530, 535, 540) 中的每一个方法包括定义可被该方法 (530, 535, 540) 捕获的异常处理器的异常处理器数组 (532, 537, 542)；

为该已转换的小应用程序定义一个数据结构 (550)，用于存储该一个或多个方法 (530, 535, 540) 的可执行代码 (531, 536, 541) 和异常处理器，该数据结构 (550) 包括第一和第二部分 (552, 554)；

为该一个或多个方法 (530, 535, 540) 定义一个顺序；

按照这个顺序，把该一个或多个方法 (530, 535, 540) 的可执行代码 (531, 536, 541) 装载到该数据结构 (550) 的第二部分 (554) 中；

把用于所有方法 (530, 535, 540) 的异常处理器数组 (532, 537, 542) 结合成一个单独的异常处理器表 (556)，包括按照为该一个或多个方法 (530, 535, 540) 定义的顺序给异常处理器数组 (532, 537, 542) 排序；和

把单独的异常处理器表 (556) 存储在该数据结构 (550) 的第一部分 (552) 中。

26. 一种计算机实现的管理过程，用于在由虚拟机执行一个或多个类中的两个或多个方法(530, 535, 540)的过程中管理可丢出的一个或多个异常，一个或多个类中的每个方法(530, 535, 540)包括可执行代码(531, 536, 541)和异常处理器数组(532, 537, 542)，所述异常处理器数组(532, 537, 542)定义与该方法相关的异常处理器，单个的异常处理器数组(532, 537, 542)被结合并形成用于两个或多个方法的单独的异常处理器表(556)，该管理过程包括：

在执行两个或多个方法(530, 535, 540)之一的过程中，当一个异常被丢出时，搜索异常处理器表(556)，包括在单独的异常处理器表(556)中定位第一个匹配异常。

27. 一种计算机系统，用于在由机器执行一个或多个类中的方法(530, 535, 540)的过程中管理可丢出的一个或者多个异常，每个方法(530, 535, 540)包括定义与该方法相关的异常处理器的异常处理器数组(532, 537, 542)，该系统包括：

处理器，被配置成将用于所有方法(530, 535, 540)的异常处理器数组(532, 537, 542)结合成一个单独的异常处理器表(556)。

28. 一种计算机系统，用于在执行程序时最小化运行时栈所需要的存储量，该运行时栈在由机器执行程序的过程中为了存储一个或多个帧而在运行时间被保持，所述一个或多个帧包括指向调用该程序中的当前执行方法(530, 535, 540)的调用方法(530, 535, 540)的返回指针，该系统包括：

第一处理器，被配置成将程序中的两个或多个方法(530, 535, 540)的异常处理器信息结合成一个复合异常处理器表(556)；和

第二处理器，被配置成在两个或多个方法(530, 535, 540)之一的执行过程中，当一个异常被丢出时，搜索复合异常处理器表(556)，以便在不要求在运行时栈上存储指向异常处理器信息的指针的情况下定位异常处理器信息。

29. 一种计算机系统，用于把类文件转换为已转换的小应用程序以便在资源受限设备上执行，该系统包括一个处理器，被配置成：

接收一个或多个类文件，该一个或多个类文件包括一个或多个方法，所述一个或多个方法(530, 535, 540)中的每一个方法包括定义可被该方法捕获的异常处理器的异常处理器数组(532, 537, 542)；

为该已转换的小应用程序定义一个数据结构(550)，用于存储该一个或多个方法(530, 535, 540)的可执行代码(531, 536, 541)和异常处理器，该数据结构(550)包括第一和第二部分(552, 554)；

为该一个或多个方法(530, 535, 540)定义一个顺序；

按照这个顺序，把该一个或多个方法(530, 535, 540)的可执行代码(531, 536, 541)装载到该数据结构(550)的第二部分(554)中；

把用于所有方法(530, 535, 540)的异常处理器数组(532, 537, 542)结合成一个单独的异常处理器表(556)，包括按照为该一个或多个方法(530, 535, 540)定义的顺序给异常处理器数组(532, 537, 542)排序；和

把单独的异常处理器表(556)存储在该数据结构(550)的第一部分(552)中。

30. 一种计算机实现的系统，用于在由虚拟机执行一个或多个类中的方法(530, 535, 540)的过程中管理可丢出的一个或多个异常，在类中的每个方法(530, 535, 540)由类文件来描述并包括定义与该方法相关的异常处理器的异常处理器数组(532, 537, 542)，单个的异常处理器数组(532, 537, 542)结合并形成用于两个或多个方法的单独的异常处理器表(556)，该系统包括：

处理器，被配置成在执行两个或多个方法(530, 535, 540)之一的过程中，当一个异常被丢出时，搜索异常处理器表(556)，包括在单独的异常处理器表(556)中定位第一个匹配异常。

零开销异常处理

公开申明

本专利文档公开的一部分包含受版权保护的材料。版权所有者不反对复制已经出现在专利与商标管理处专利文件或记录中的专利公开的任一部分，但保留除此之外的所有版权。

技术领域

本发明一般涉及面向对象的计算机软件，特别涉及为了在一个实现在资源受限的设备如智能卡中的 Java 虚拟机中支持异常处理时使栈存储要求最小的数据结构和方法。

背景技术

一个虚拟机是由在处理器上执行的应用软件或指令序列产生的一个抽象的计算机器。在虚拟机上执行的程序可以是“体系结构中立 (architecture-neutral)”。术语“体系结构中立”指的是程序，像那些用 Java™ 语言编写的程序，它们可以在有多种不同计算机体系结构的不同计算机平台上由虚拟机执行。这样，举例来说，一个在基于 Windows™ 的个人计算机系统上实现的虚拟机可以使用与在基于 UNIX™ 的计算机系统上实现的虚拟机相同的指令系统。一个虚拟机的指令序列的平台无关编码的结果是一个含一个或多个字节码的流，例如它们中的每一个是一个单字节长的数字代码。

Java 编程语言是一个面向对象的编程语言。在一个面向对象的系统中，“类”描述数据以及对这些数据进行操作的方法的集合。合在一起，这些数据和方法描述一个对象的状态和行为。

Java 编程语言也是可检验的，这意味着，在一个用 Java 编程语言编写的应用程序执行之前，就可以判断是否有程序中的指令序列试图为字节码处理不正确类型的数据或者程序中字节码指令的执行是否会引起操作数栈的下溢或上溢。

Java 虚拟机执行用 Java 编程语言写的虚机器代码并满足下面提到的“Java™ 虚拟机规范”。Java 虚拟机被设计为使用 32 位的体系结构。然而，许多资源受限的设备，像智能卡，有 8 位和 16 位体系结

构。

智能卡，也被认为是智能移动数据传输卡，通常用塑料或金属制成并有一个包含一个用来执行程序的嵌入式微处理器和一个用来存储程序 and 数据的存储器的电子芯片。这些设备可以像一张信用卡大小，通常只有有限的存储容量。有限的体系结构和存储器使得在这些设备上实现一个 Java 虚拟机不切实际或者是不太可能的。例如，一些智能卡只有不到 1K 的随机访问存储器 (RAM) 和 16K 的只读存储器 (ROM)。当在一个资源受限的设备上实现一个 Java 虚拟机时一个困难的例子出现在对异常的处理中。

参照图 1，在用 Java 编程语言编写的计算机程序的上下文中，异常处理器 100 是一个过程 (或者是过程中的一个指令集合)，用来保护程序代码的一个特定集合，称作被保护代码块 102。当一个 Java 程序违反了 Java 编程语言的语义约束时，Java 虚拟机把这个错误作为一个异常通知给该程序。在相应被保护代码的执行过程中，无论何时只要可适用的异常被“丢出”异常处理器就被执行。Java 编程语言规范规定在语义约束被违反时一个异常将被丢出并引起一个从异常发生点到一个可以由程序员指定的点的非本地传输控制。一个异常被说成是从它发生的点被丢出并在控制被传向的点被捕获到。例如，一个特定的异常处理器，像一个处理“文件终点” I/O 错误的过程，可以被定义为适用于第一个方法 104 的一个特定部分。如果相应异常 (也就是本例中的文件终点异常) 在被保护代码的执行过程中产生，异常处理器 100 的执行就被初始化。异常可以被隐含地或明确地丢出。隐含异常被 Java 虚拟机作为程序指令执行的结果丢出，像空指针异常。作为选择，明确异常可以被包括在使用 Java “throw” 语句的方法体中。

如果有一个针对被丢出异常 (thrown exception) 的可适用的封装的异常处理器被丢出的异常就被说成是被异常处理器捕获到。封装的异常处理器的可适用指令范围包括丢出相应异常的指令。从一个方法中特定指令的观点来看，封装的异常处理器集就是那些可适用指令范围 (被保护代码的集合) 包括特定指令的异常处理器的集合。

当提到异常处理器时 Java 编程语言通常指的是“try 语句”，“try 块”，“catch 子句”以及“finally 子句”。一个 try 语句包括一个 try 块，零个或多个 catch 子句以及可选择的 finally 子句。异常

被 try 块中的封装代码捕获到。try 块是特定异常处理器适用代码(也就是被保护的代码块)的一部分。catch 子句定义了一个异常处理器。try 语句中的 finally 子句为执行一节代码提供了一个机制,无论是否有异常被丢出。在一个 Java 程序中,如果语句或表达式出现在 try 语句的 try 块中且 catch 子句是 try 语句的一部分,或者语句或表达式的调用者被 catch 子句动态封装,则语句或表达式被 catch 子句动态封装。

一个特定的 catch 子句是否处理一个异常是由对被丢出异常对象的类与 catch 子句中参数的声明类型进行比较来决定的。如果 catch 子句的参数的类型是异常的类或是异常类的超类 catch 子句就处理异常。等价地,一个 catch 子句将捕获是所声明参数类型的实例的任何异常。

如果第一个方法的被保护部分包括对其他方法 106 的调用(在 Java 虚拟机规范中被称为“invoke”指令,见下),它可能依次包括对更多方法 108, 110 的嵌套调用,这时由被被保护代码 102 直接或间接调用的方法 106, 108, 110 中的任何一个所产生的“文件终点”错误也将会引起异常处理器 100 的执行被调用。然而,嵌套的方法 112 可以包括它自己的文件终点异常处理器 114. 如果一个异常在执行方法 112 时被丢出,这时异常处理器 114 将被用来处理由被包括在该嵌套方法中的指令的执行所引起的文件终点异常,以及由被嵌套方法 112 调用的任一方法 116 的执行所产生的异常。

在传统 Java 程序中,与一个对象类相关的所有方法一起被存储在一个称为类文件的数据结构中,它被定义在 Java 虚拟机规范中。每一个方法有它自己的异常表而且每个方法的代码包括用于被它的异常表所引用的异常处理器的代码。当一个 Java 类文件被创建时,与一个方法相关的所有异常被排列在一个列表中,作为异常处理器表被查阅。参见图 2,一个传统的异常处理器表 200 包括一个或多个 catch 子句(异常处理器)202. 每个 catch 子句 202 包括一个起始程序计数器地址 204 和一个结束程序计数器地址 206,它们描述了对其来说异常处理器是活动的 Java 虚拟机指令范围,类型指示器 208 描述 catch 子句将要处理的异常的类型以及一个异常处理器代码的异常将从其开始的地址 210。

异常处理器表中 catch 子句的顺序非常重要。一个异常的丢出会

导致 Java 虚拟机在异常处理器表中进行搜索。Java 虚拟机的执行在第一个匹配的 catch 子句处继续。因为 Java 代码是结构化的，把用于一个方法的所有异常处理器排列在一个列表中通常是不可能的。对任意可能的程序计数器值这个列表都可以被搜索以找到合适的异常处理器，也就是说，最里层的异常处理器不仅封装了程序计数器(pc)值(在那里异常被丢出)并且可以处理正在被丢出的异常。

如果没有匹配的 catch 子句，当前方法就被说成是有一个无法捕获的异常。当一个异常无法捕获时，调用者，也就是调用当前方法的方法(如果有的话)的异常状态被恢复。即使异常已经在调用者中调用实际产生异常的方法的指令处发生异常的传播仍然继续。

Java 虚拟机保持许多不同的运行期数据结构以追踪方法的执行和调用。一个 Java 虚拟机可以同时支持许多执行线程。每个 Java 虚拟机线程有自己的 pc(程序计数器)寄存器。该 pc 寄存器包括当前被执行的 Java 虚拟机指令地址。每个 Java 虚拟机线程有一个私有的与线程同时被创建的 Java 栈。Java 栈存储 Java 虚拟机帧。Java 栈等价于传统语言，如 C，中的栈：它存储局部变量和中间结果并支持方法调用和返回。

现在看图 3，显示了一个包括很多 Java 虚拟机帧的传统 Java 栈 300。每个帧 302 包括一个对一个方法的引用 304 和一个返回指针 306。

引用 304 是对当前正在执行的方法的引用，指的是当前方法。引用 304 被用来指示当在当前方法的执行过程中一个异常被丢出时哪个异常处理器表将被搜索。引用 304 可以是一个地址的形式，当前方法被存储在其处。用于每个方法的代码一般包括该方法所需要的最大栈空间，被该方法所用的最大寄存器数，用于执行方法的实际字节码以及一个异常处理器表。引用 304 可以指向当前方法的起点，异常处理器表通常被存储在那里。

返回指针 306 指向调用当前方法的方法，更明确一点是指向在当前方法执行完毕后调用方法中执行将在其处被恢复的位置。

像上面所描述的那样，智能卡和其它资源受限设备一般只有有限的存储容量。相应地，包括许多需要大量字节以被存储在 Java 栈中的嵌套方法的程序在运行时可能引起栈溢出。在允许对 Java 虚拟机中传统异常处理进行全支持的同时，希望限制在运行时被要求存储在存储

器中的信息。

发明内容

一方面，本发明提供了一种计算机实现的过程，用于在由机器执行在一个或多个类中的方法的过程中管理可丢出的异常。每个方法包括一个定义与该方法相关联的异常处理器的异常处理器数组。该方法包括将用于所有方法的异常处理器数组结合到一个单独的异常处理器表中。

本发明的各个方面包括下述特征中的一个或多个。用一个类中所有方法或所有类中所有方法的所有异常处理器数组可以被存储在一个单独的异常处理器表中。在用于一个 Java 包中所有方法的所有异常处理器数组可以被结合成单独的异常处理器表。一个方法可以被包含在一个类文件中。结合所有异常处理器数组的步骤可以包括把一个类文件中所有方法的异常处理器结合到单独的异常处理器表中。

该过程可以包括在执行其中一个方法的过程中当一个异常被丢出时搜索异常处理器表，包括在单独的异常处理器表中定位第一匹配的异常。搜索步骤可以包括从异常处理器表中按顺序检索异常处理器入口，为第一匹配的异常处理器检查每一个异常处理器的类型和范围并在当前异常处理器不匹配且是相关方法中被保护代码的最顶层的最后一个异常处理器时停止搜索。

一个类中所有的方法被描述在一个单独的类文件中。类文件可以是 Java 类文件。一个或多个类中的方法可以被分组在一个包括一个有第一和第二两个部分的包数据结构的包中。该过程可以包括把异常处理器表存储在包的第一部分中和把所有方法存储在包的第二部分中。结合步骤可以包括链接异常处理器数组，包括把每个异常处理器数组依照预定义的顺序装入到包数据结构的第二部分。预定义的顺序可以依据存储在包数据结构第二部分中的方法的顺序来确定。

机器可以是在一个资源受限设备上实现的虚拟机。资源受限设备可以是一个智能卡。一个或多个类中的方法可以被分组在一个包中且这个包可以被安装在智能卡上。该过程可以包括创建一个包括一个有第一和第二两个部分的包数据结构的包。该过程可以包括把用于每一个类文件的异常处理器数组链接成一个异常处理器表，把异常处理器表存储到包的第一部分，把所有方法存储到包的第二部分。

另一方面，本发明提供一个最小化在执行程序时运行时栈所需要的存储量的方法。在机器执行程序的过程中在运行时保持一个运行时栈，用于存储一个或多个帧，其中每一帧包括一个指向调用程序中当前执行方法的调用方法的返回指针。本方法包括把用于包含在程序中的所有方法的异常处理器信息结合成一个复合异常处理器表并在方法之一的执行过程中当有异常被丢出时定位和搜索复合异常处理器表，以便在不需要在运行时栈上存储指向异常处理器的指针的情况下定位异常处理器信息。

本发明的各个方面可以包括一个或多个特征。程序可以是一个 Java 程序。机器可以是实现一个 Java 虚拟机的虚拟机。程序可以包括方法包，其中方法包括在一个或多个类中。虚拟机可以被实现在一个资源受限设备中，在该资源受限设备上包被安装并在其上执行。

该方法可以包括在安装时把包登记在一个注册服务中。该注册服务保存一个指针和一个范围。指针指示与一个给定包相关的复合异常处理器表在资源受限设备中的一个位置。范围定义了资源受限设备中与包相关联的方法被定位在的地址的范围。

定位步骤可以包括定位一个与当前执行方法相关联的包，包括把异常被丢出处的地址与登记在注册服务中每个包的范围进行比较。搜索步骤可以包括搜索与一个已定位的包相关的复合异常处理器表。

另一方面，本发明提供一个方法把类文件转换成一个已转换的 Java 小应用程序以在资源受限设备上执行并且包括接收一个或多个类文件，每个类文件都包含一个或多个方法。每个方法包括一个定义可被方法丢出的异常处理器的异常处理器数组。该方法包括定义一个包含第一和第二两个部分的数据结构，用于存储已转换的 Java 小应用程序的方法和异常处理器，并且为方法定义顺序，并且按照顺序把方法装入到数据结构的第二部分中。用于所有方法的异常处理器数组被结合成一个单独的异常处理器表。异常处理器数组被按照为方法定义的顺序在表中排序。本方法包括把单独的异常处理器数组存储在数据结构的第一部分。

另一方面，本发明提供了一种计算机实现的过程，用于由虚拟机管理在执行一个或多个类中的方法过程中可被丢出的异常。每个方法包括定义与该方法相关的异常处理器的异常处理器数组。单个异常处

理器数组被结合并形成用于两个或多个方法的单独的异常处理器表。该过程包括在执行其中一个方法的过程中当一个异常被丢出时搜索异常处理器表，包括在单独的异常处理器表中定位第一匹配异常。

另一方面，本发明提供一种计算机系统，它包含被配置成把用于方法的异常处理器数组结合成单独的异常处理器表的处理器。

另一方面，本发明提供一种计算机系统，它包含：第一处理器，其被配置成把用于程序中方法的异常处理器信息结合成一个复合异常处理器表并定位；以及第二处理器，其被配置成在其中一个方法的执行过程中当有异常被丢出时搜索该组合异常处理器表，以定位异常处理器信息而不需要在运行时栈上存储指向异常处理器的指针。

另一方面，本发明提供一种包含使计算机系统接收一个或多个类文件的指令的计算机系统。每个类文件包含一个或多个方法，且每个方法包含一个定义可被该方法丢出的异常处理器的异常处理器数组。指令被包含用来使计算机系统定义包括第一和第二两个部分的数据结构，用于存储用于已转换的小应用程序的方法和异常处理器，为方法定义一个顺序，并按照该顺序把方法装载到数据结构的第二部分，把用于两个或多个方法的异常处理器数组结合成一个单独的异常处理器表，包括按照给方法定义的顺序给异常处理器数组排序，并把单独的异常处理器数组存储在数据结构的第一部分。

另一方面，本发明提供一种计算机系统，它包含被配置成在执行方法的过程中当有异常被丢出时搜索异常处理器表的处理器。指令被包含用来在单独的异常处理器表中定位第一匹配异常。

本发明的实施方案可以包括一个或多个优点。需要存储在运行时栈中的信息量可以在仍然支持传统异常处理方法的同时被减少。可以产生链接的异常处理器表，它包括用于与在资源受限设备中所用方法包相关的所有异常的入口。不要求栈开销来支持异常处理。一旦异常被丢出，Java 虚拟机可以有效地搜索链接的异常处理器表，以找到正确的异常处理器。利用优化，对链接的异常处理器表的搜索可以在不要求为嵌套 catch 子句的特定类比较链接表中的所有入口的情况下实现。表入口可以被排序，以确保匹配的搜索终点在顶层处理器上。

其他的特征和优点可以从下面的详细描述以及所附的插图和权利要求中清楚地知道。

附图说明

图 1 是在运行时通过过程调用彼此互相连接的一组方法的示意图。

图 2 显示了包含了用于被封装在方法中的每一个异常的入口的传统 Java 异常处理器表。

图 3 显示了一个传统的 Java 栈运行时数据区。

图 4 是一个示意结构图，描述一个包含了依照本发明驻留在智能卡上的虚拟机的示范系统。

图 5a 是针对一个创建链接的异常处理器表的过程的流程图。

图 5b 是包含了用于将要在 JavaCard™ 虚拟机上执行的包的所有方法的一个方法组件数据结构的示意结构图。

图 6 是一个用于一个 Java 包的链接的异常处理器表的示意结构图。

图 7 一个由 JavaCard 虚拟机保存的异常记录的示意结构图。

图 8 显示了一个由 JavaCard 虚拟机保存的运行时栈数据区。

图 9 是一个用于搜索链接的异常处理器表的运行时方法的流程图。

图 10a 是一个用于搜索链接的异常处理器表的优化的运行时方法的流程图。

图 10b 是对图 10a 中所描述的优化的运行时方法的实现的详细流程。

具体实施方式

下面描述在一个 Java 包在资源受限设备上执行期间用于处理被丢出异常的数据结构和方法。资源受限设备通常被认为是那些在存储器和/或计算能力或速度上受限制的设备。虽然下述特定的实现是引用智能卡来描述的，本发明也可被用于其他资源受限设备，包括，但不限于，蜂窝电话，边界扫描设备，域可编程设备，个人数据助理(PDAs)和寻呼机，以及其他微型或小面积设备。这样的设备一般只有有限的存储容量。例如，一些智能卡只有不到 1K 的随机访问存储器(RAM)和有限有只读存储器(ROM)和/或不可变存储器，像电子可擦可编程只读存储器(EEPROM)。同样地，一些资源受限设备是基于为少于 32 位而设计的体系结构。例如，一些可以使用本发明的资源受限设备是基于 8

位或 16 位体系结构，而不是 32 位体系结构。

参见图 4，用于一个受限设备，像智能卡 40 的 Java 小应用程序的开发以类似于 Java 程序的开发模式开始。也就是说，一个开发者编写一个或多个 Java 类并用 Java 编译器编译源代码以产生一个或多个类文件 10。Java 小应用程序可以被运行，测试和调试，例如，在一个使用仿真工具以仿效智能卡 40 上环境的工作站上。当 Java 小应用程序准备被下载到智能卡 40 上时，类文件 10 被转换器 14 转换成一个已转换的 Java 小应用程序 (CAP) 文件。转换器可以被实现为一个由桌面计算机执行的 Java 应用程序。除了类文件 10 之外转换器 14 还接收一个或多个输出文件 12 作为它的输入以被转换。输出文件 12 包括用于被正在被转换的类引入的其他包的内容的命名和链接信息。

通常，CAP 文件 16 包括在一个单独的 Java 包中定义的所有类和接口并被表示成一个 8 位字节流。所有 16 位和 32 位量通过读 2 个或 4 个连续的 8 位字节被分别构造。在其他情况中，CAP 文件 16 包括一个独立于方法组件 20 被打包的常量池组件 18。常量池组件 18 包括常量的几个字节，范围从在编译时知道的数字文字到在程序被下载到智能卡 40 或在由智能卡 40 执行时都被解析的方法和域引用。方法组件 20 规定了将要被下载到智能卡 40 上并随之由其执行的指令集。一个示范性的已转换 Java 小应用程序文件 16 结构进一步的细节在国际应用期间题为“用于其他资源受限设备的一个面向对象指令集”（由 Joshua B. Susser et al. 所作）中有所论述，同时与本应用归档，它基于美国应用序列号 No. 09/243,101。

通常，为资源受限平台像智能卡 40 所写的实现和 Java 小应用程序遵循用于 Java 平台包的标准规则。Java 虚拟机和 Java 编程语言在 T. Lindholm et al. 的“Java™ 虚拟机规范”（1997）和 K. Arnold et al. 的“Java™ 编程语言第二版”（1998）中有所描述，在这里通过对它们的全面引用而把它们组合在了一起。用于智能卡平台的应用卡接口 (API) 被写成包括包名称的 Java 源文件，这里一个包包括一批编辑单元和唯一的名字。包机制用于对类，域以及方法的访问的识别和控制。

像上面所描述的，转换器 14 在类文件 10 之外还接收一个或多个输出文件 12 作为它的输入以被转换并创建一个已转换 Java 小应用程序文件 16。每个类文件包括一个与给定类中每个方法相关的异常处理

器表。转换器 14 把用于与一个包相关的所有类的所有方法的所有异常处理器表链接成一个单独的异常处理器表。链接异常处理器表和用于包的基础方法一起被保存并像下面所详细描述地那样在运行时当一个异常被丢出时被搜索。

参见图 5a 和 5b, 一个由转换器 14(图 4)执行的用来链接用于包中所有方法的异常处理器表的过程。为包中的方法定义一个序列 504。该序列为相关方法组件 550 中的基础方法定义代码的位置。例如, 三个方法 530, 535 和 540(方法 A, B, C)被包括在包 545 中。每个方法包括代码(分别是 531, 536 和 541)以及一个相关的异常处理器表(分别是 532, 537 和 542)。

方法组件描述包中所声明的每一个方法。与每个方法相关的异常处理器表也被描述。在一个实现中, 方法组件被表示成下面的结构:

```
method-component {  
    u1 tag  
    u2 size  
    u1 handlers-count  
    exception-handler-info exception-handlers[handlers-count]  
method-info methods[]  
}
```

表 1: 方法组件

tag 项有一个标识该数据结构为方法组件的值。Size 项指出方法组件结构中的字节数, 不包括 tag 项与 size 项。handlers-count 项表示在链接异常处理器表中的入口数。exception-handlers 项表示排列在表中的 8 位异常处理器结构的一个数组, 称为链接的异常处理器表。每个异常处理器结构代表定义在包的一个方法中的一个 catch 子句或 finally 子句。链接的异常处理器表中的入口被按照 methods 项中在方法组件的起点与每个活动异常处理器范围的终点之间的距离升序排列。下面详细描述了与包中特定方法相关的异常处理器块的顺序。

methods 项代表一个可变长方法信息结构的数组。每个入口代表在类中声明的一个方法或给定包的一个接口。

为了当前应用的目的, 方法组件 550 可以被表示成一个包括第一

部分 552 和第二部分 554 的一个数据结构。第一部分 552 是一个用于针对包中所有基础方法的异常处理器的占位器 (place holder)。异常处理器被存储在链接的异常处理器表 556 中。单个的方法代码存储在第二部分 554 中。

单个的异常处理器表被装入链接的异常处理器表中并按照步骤 504 (506) 中定义的顺序被排序。也就是说, 用于每个方法的每个异常处理器表以块的形式被装入链接的异常处理器表 556 中。在用于每个方法的每一单个异常处理器表中定义的顺序也被保持。在图 5b 所示的例子中, 在排序步骤 504 执行之后方法在方法组件的第二部分 554 中被排成 A, C 和 B。相关的异常处理器表 A, B 和 C 被按照相同的顺序 (A, C 然后是 B) 装入到链接的异常处理器表 556 中。单个异常处理器表中单个异常处理器的顺序满足传统 Java 编程语言构造以确保由 Java 虚拟机在运行时找到的第一匹配是可以出现的最精确的匹配。

在一个实现中, 异常被组织在类层次中。异常类层次有一个称为“可丢出”的最高级异常类和两个主要分支: 一组超类是“错误”类的格外严重的异常和一组超类是“异常”类的不那么悲惨的异常。

当一个异常被丢出时, Java 虚拟机执行对适用于被丢出异常的封装的异常处理器进行树状搜索找到的第一异常处理器。为了确保适用于被丢出异常的最低类的封装异常处理器被使用, Java 程序 (也就是方法) 的作者通常要给方法中的异常处理器排序以使较低类的封装异常处理器位于较高级类的异常处理器之前 (类级别由异常层次的位置决定)。更进一步, 当有两个或更多适用于引起异常被丢出指令并用于几乎相同的异常状况的封装异常处理器时, Java 虚拟机执行由调用引起异常方法的方法链中最近的方法所建立的适用的封装异常处理器。

当单个异常处理器表在步骤 506 中被排序并被装载之后, 链接的异常处理器表中的每个异常处理器入口被转换成一个用来在资源受限设备像智能卡 (图 4) 中使用的优化的数据结构。在一个实现中, 用于链接的异常处理器表 556 中优化入口的数据结构被列在图 6 中。链接的异常处理器表 600 包括多个入口 602, 它们中的每一个都包括与异常处理器信息数据结构一致的数据。异常处理器信息数据结构包括起始偏移 604, 活动长度 606, 处理器偏移 608 以及捕获类型索引 610。

起始偏移 604 和结束偏移 (未显示) 是进入到特定方法组件之中的

字节偏移。起始和结束偏移用一个字节码数组来指示异常处理器的活动范围。起始偏移 604 的值必须是到相对于一条指令的操作码的字节码数组之中的一个有效偏移。

结束偏移被定义为起始偏移 604 加上活动长度 606。结束偏移的值必须是到相对于一条指令的操作码的字节码数组之中的一个有效偏移或者必须等于方法的字节码数量(代码数组的长度)。起始偏移 604 的值必须小于结束偏移的值。起始偏移 604 是内包含的而结束偏移是外包含的;也就是说,当执行地址在区间[起始偏移,结束偏移)内时,异常处理器必须是活动的。

活动长度 606 用字节码定义由给定异常处理器封装的指令范围。在一个实现中,活动长度 606 被编码以指出特定异常处理器的活动范围是否被嵌套在另一异常处理器之中,更明确一点,当前异常处理器是否是特定被保护代码块相关的异常处理器列表中的最后一个处理器。对用 Java 编程语言编写的程序来说,位被编码以指出当前异常处理器是否用于被保护代码块的异常处理器列表中的最后一个处理器(catch 或 finally 子句)。如果被保护代码块不被包容在另一被保护代码块之中并且当前处理器是与被保护代码块相关的最后一个处理器则活动长度 606 的高位被设成 1。如果被保护代码块被包容在另一被保护代码块之中或者当前处理器不是与被保护代码块相关的最后一个处理器则活动长度 606 的高位被设为 0。在使用编码的地方,结束偏移被定义为起始偏移 604 加上活动长度 606 以及 0x7FFF。

处理器偏移 608 表示一个到方法组件的 info 项之中的字节偏移。更明确一点,处理器偏移 608 指出当一个特定异常被捕获时异常处理器的起始和执行将被 Java 虚拟机恢复的位置。处理器偏移 608 的值必须是到相对于一条指令操作码的方法的字节码数组之中的一个有效偏移,并且必须小于方法的字节码数量的值。

捕获类型索引 612 指示异常处理器类型。为了把控制传输到异常处理器,与被丢出异常相关的程序计数器必须落在为异常处理器定义的范围之中并且必须是同一类型。如果捕获类型索引 610 的值是非零,它必须是到代表由异常处理器捕获到的异常的类的常量池表之中的一个有效索引。如果异常处理器代表一个 finally 子句,捕获类型索引 610 的值被设为 0。一个 finally 子句异常处理器被称为是用于在起始

偏移与结束偏移的范围中被丢出的所有异常，不考虑类型。

再看图 4，在转换之后，CAP 文件可以被存储在一个计算机可读介质 17 如硬盘驱动器，软盘驱动器，光学存储介质，闪存装置或其他合适的介质上。

这时 CAP 文件 16 被拷贝或传输到一个带有外围卡接收设备 (CAD) 24 像桌面计算机一样的终端 22 中。卡接收设备 24 可以被写信息并从智能卡 40 上接收信息。卡接收设备 24 包括一个可把智能卡 40 插入到其中的卡端口 (未显示)。一旦插入进去，同对压在智能卡 40 的连接区表面上的一个连接器接触以提供电源并允许与智能卡 40 通信。终端 22 还包括一个用来装入 CAP 文件并把它传送给智能卡 40 的安装工具或程序 26。

智能卡 40 有一个输入/输出 (I/O) 端口 42，包括一套通过程序进行的联系，可以提供数据和通信。智能卡 40 还包括一个用于接收 CAP 文件的内容并为在智能卡 40 上的执行准备 Java 小应用程序的安装工具 46。安装工具 46 可以被实现为一个 Java 程序并可在智能卡 40 上被执行。智能卡 40 还有存储器，包括可变存储器像随机访问存储器 50 和不可变存储器像电子可擦可编程只读存储器 (EEPROM)。智能卡 40 还有只读存储器像只读存储器 52。由转换器 14 准备的 Java 小应用程序 (CAP 文件 16) 可以被存储在电子可擦可编程只读存储器中。

作为安装过程的一部分，安装工具 46 创建一个或多个数据区以易于运行时操作。数据区的一个例子是异常记录。异常记录是在解释一个丢出的异常时由 JavaCard™ 使用的一个列表。

参见图 4 和图 7，异常记录 700 可以是一个可延长的链接好的列表数据结构。新入口可以加在列表的头部。异常记录 700 中的每个入口 702 代表一个包含异常处理器的方法组件 550 (见图 5b)。没有异常处理器的方法组件不被包含在内。

在一个已转换 Java 小应用程序文件的安装过程中，安装工具 46 调用一个 JavaCard 虚拟机以登记与包相关联的方法组件。每个入口包括相关方法组件的地址 704 和它的大小 706。地址 704 和大小 706 通过对程序计数器 (pc) 的引用为包括在包中的指令定义了一个范围。异常记录的使用在下面有详细描述。

像在上面的背景中所描述的那样，Java 虚拟机保持不同的运行时

数据结构以跟踪方法的执行和引用。JavaCard 虚拟机也是这样做的。在运行时，JavaCard 虚拟机保持一个程序计数器和一个运行时栈。运行时栈存储 JavaCard 虚拟机帧。现在看图 8，显示了一个包括大量 Java 虚拟机帧 802 的运行时栈 800。每一帧包括一个返回指针 806。

返回指针 806 指向调用当前方法的方法，更明确一点是指向在当前方法执行完毕时执行在调用方法中恢复的地方。

像下面所进一步解释的那样，没有对当前正在执行方法的引用需要被存储在运行时栈上。用于与一个包相关联的所有方法的所有异常处理器被存储在一个单独的链接的异常处理器表中，它可以在运行时被搜索以找到与当前程序计数器有关的异常处理器。

现在看图 9，一个用于运行时异常处理的过程 900 在 902 步开始。JavaCard 虚拟机明确地或隐含地丢出一个异常 904。异常的类 906 与异常被丢出处的程序计数器位置 908 一起被确定。

在异常记录中进行搜索以定位封装了异常被丢出处的程序计数器的方法组件 910。这取决于当前帧的方法被实现在哪个包中。如果没有找到相匹配的封装的方法组件（并且因此没有链接的异常处理器表）912，那么就检查确定是否有栈帧可以被弹出运行时栈 914。如果没有可被弹出运行时栈的栈帧，JavaCard 虚拟机将带着一个未捕获的异常被停机 916。如果有其他帧可以从 Java 栈中被弹出，则该帧被弹出 918。帧的弹出包括恢复先前的帧并给程序计数器设一个新值，也就是由存储在运行时栈中的返回指针所指示并与先前的帧相关的返回位置。此后，过程继续在 910 步执行。入口从运行时栈中弹出允许包之间的方法嵌套。也就是说，在第一个包中的一个方法可能被第二个不同的包中的调用方法调用。像这样，调用方法被称为是封装了被调用方法，并相应地可以包括适用于被调用方法没有捕获到给定异常的事件的异常处理器。

如果一个封装的方法组件在搜索步骤 910 中被定位，那么相关的链接异常处理器表（当前的链接异常处理器表）中与方法组件相关的一个入口就被获得（在这种情况下是第一个入口）930。在一个实现中，存储在异常记录中的起始地址指向用于给定方法组件的链接异常处理器表中第一个入口的地址。像这样，下列步骤的执行可以在从异常记录中得到起始地址后直接进行。

检查与正在被测试的当前入口相关的范围以确定是否封装了异常被丢出处的程序计数器 932。如果没有匹配发生,那么当前链接异常处理器表中的下一个入口在步骤 930 中被得到。如果没有更多入口可用 934,则过程继续步骤 914 来检查确定是否有更多的栈帧可以像上面所描述地那样被弹出栈。

如果在步骤 932 中范围封装了程序计数器,则进行检查来确定由异常处理器指出的异常类型是否匹配丢出的异常 936。如果丢出异常的类型与给定类相同或者是给定类的子类则类型匹配。匹配还发生在异常处理器与一个 finally 子句相一致时。finally 子句是一个 Java 语言构造,它允许程序员为所有类类型定义将要被执行的异常处理器并因此对丢出异常的所有类型来说都是一个匹配。如果类型匹配,与当前入口相关的异常处理器就被执行 938 且过程结束 940。更明确地,程序计数器被按照为异常处理器(从图 6 的异常处理器偏移 610)而存储在当前入口中的地址来设置且执行从被捕获异常处理器的第一个语句继续执行。

在一个实现中,并不是链接异常处理器表中的所有入口都被要求在是否一个未捕获异常已被丢出的判断之前进行测试。现在看图 10a,在步骤 936 中的类型测试之后,执行一个额外的测试以确定是否有其它封装的 try 语句被包括当前方法中 937。像上面所描述的,一个封装的 try 语句反映其范围也包括当前程序计数器的其他异常处理器。如果当前没有封装的 try 语句,那么过程可以立刻转到运行时栈弹出步骤 914 而不管当前链接的异常处理器表中是否有其他入口仍在被处理。

在一个实现中,可以用为当前入口而存储在链接异常处理器表中的优化位(活动长度 606 项的已编码的高位)来调用 937 步。更明确点,用于当前入口的活动长度的高位被获得 937a。进行检查以确定该位是否被设置 937b。如果该位没有被设置(指示当前有一封装的 try 语句或有助于同一 try 语句的更进一步的处理器),则过程继续 934 步。如果该位被设置,则过程继续 914 步。

在本发明已经通过对一些特定实施方案的引用被描述过时,这些描述是说明本发明而不应该被解释为限制本发明。对那些本领域的专家来说在不偏离由附录的权利要求所定义的本发明的范围的情况下各

种不同的更改也可以出现。

本发明适用于以除 Java 以外的编程语言所编写的方法和程序，包括被编译成平台无关代码的编程语言。所描述的技术可用于其他环境中，例如用于以 C 或 Pascal 编写的程序，那里也要求类似的栈操作。

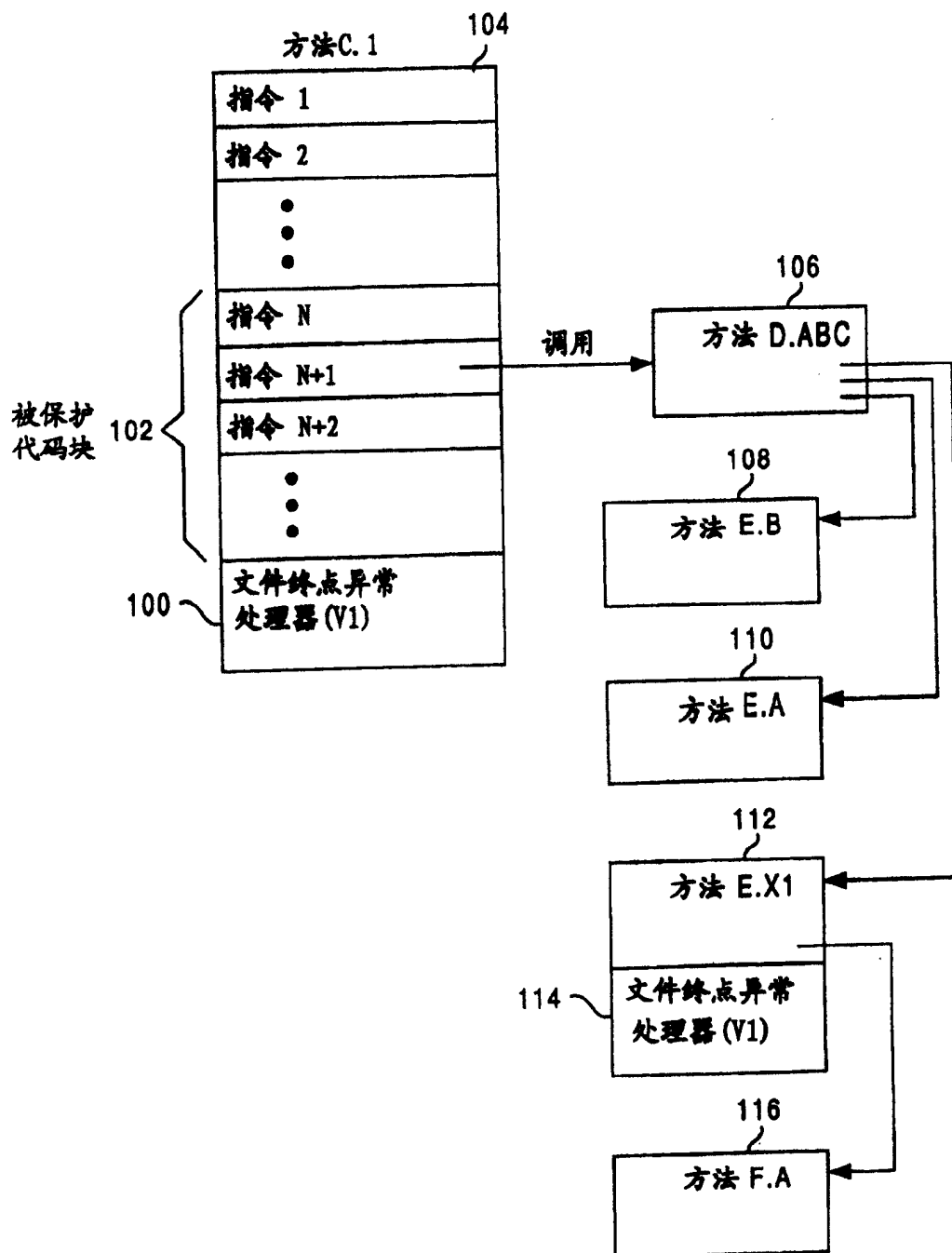


图 1

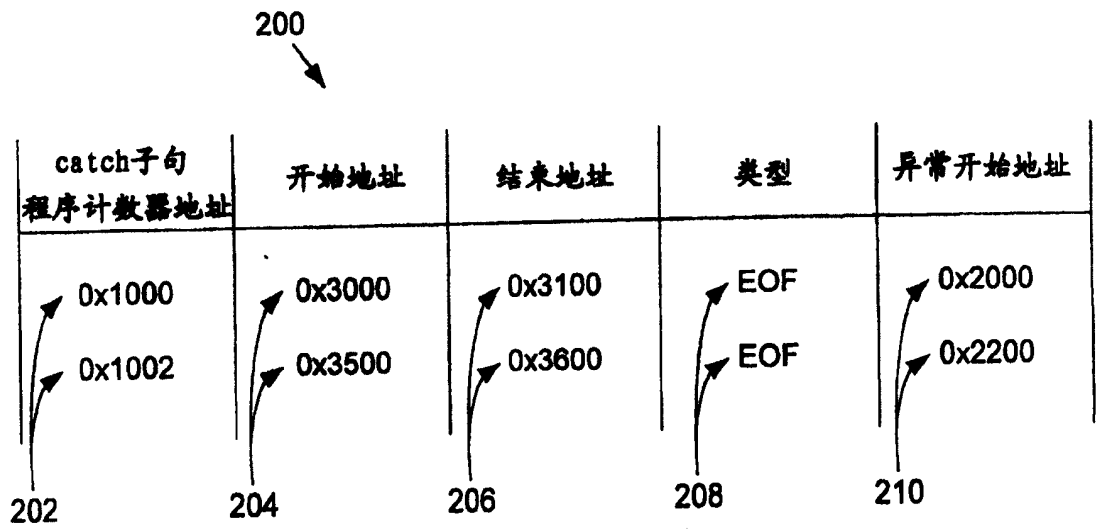


图 2

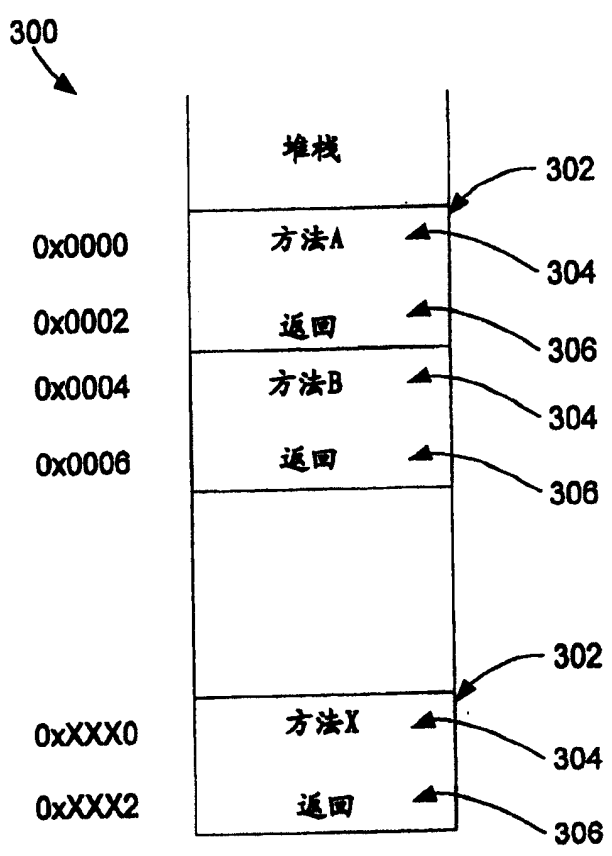


图 3

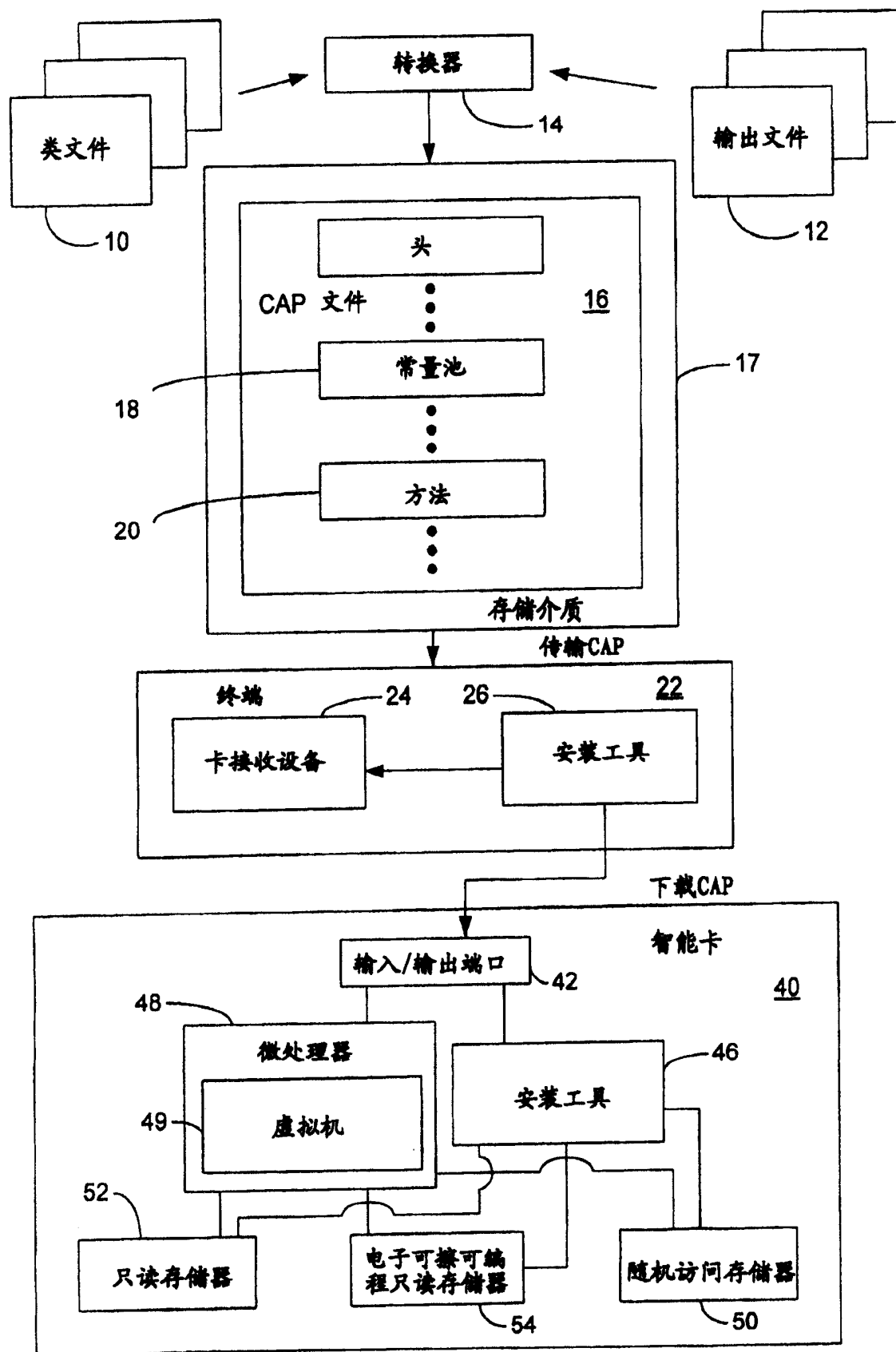


图 4

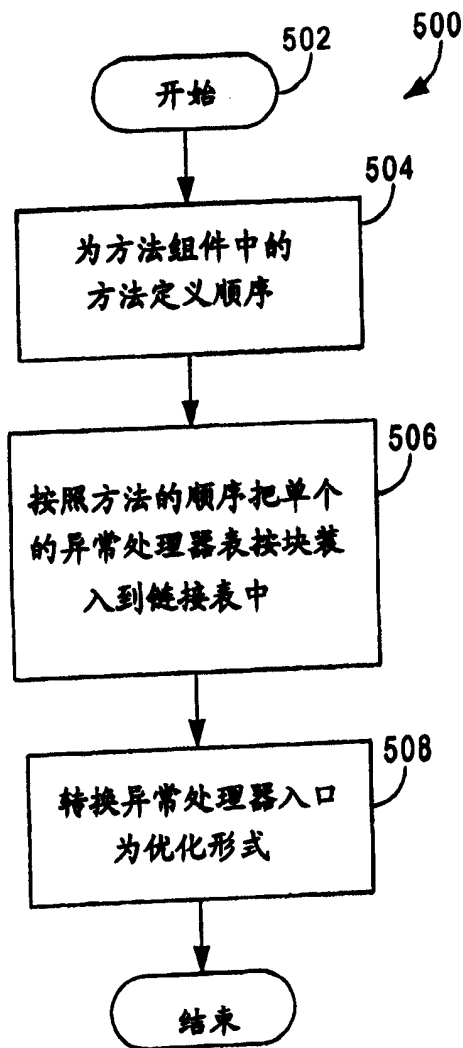


图 5a

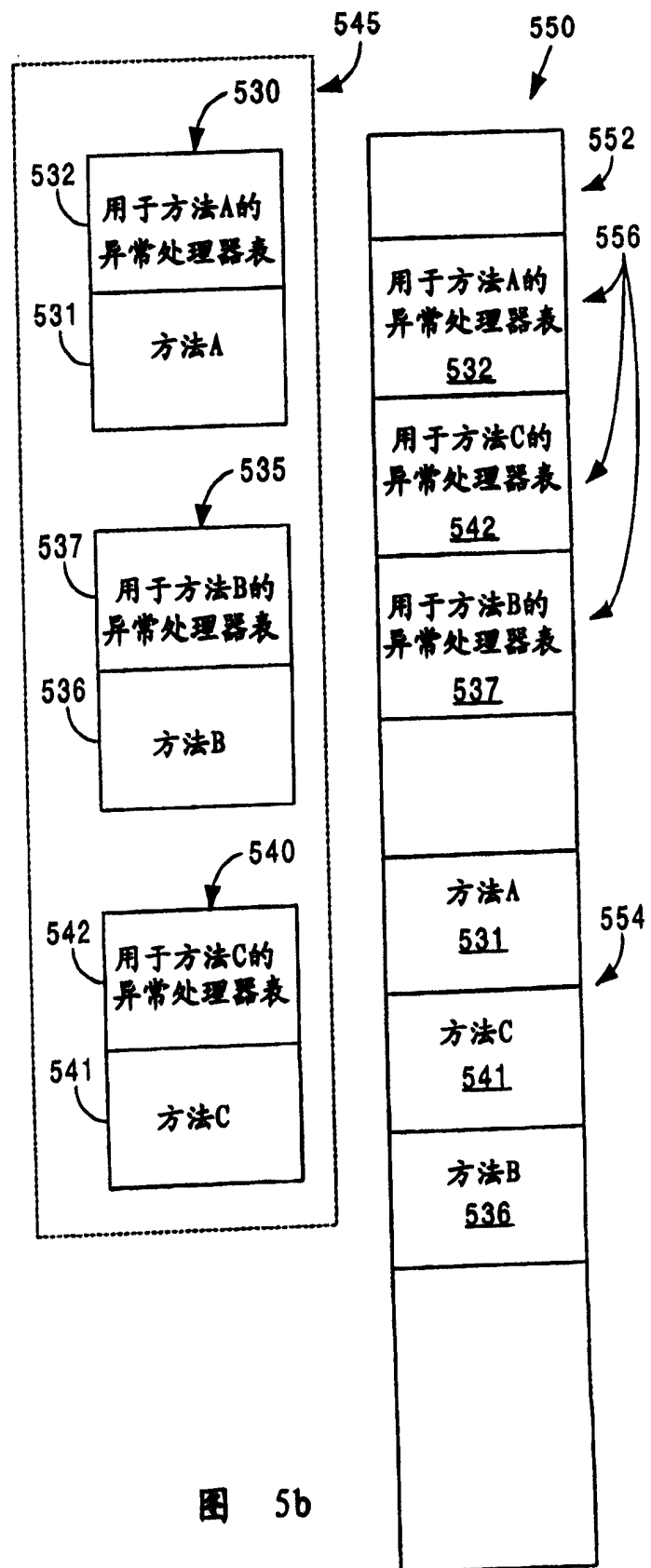


图 5b

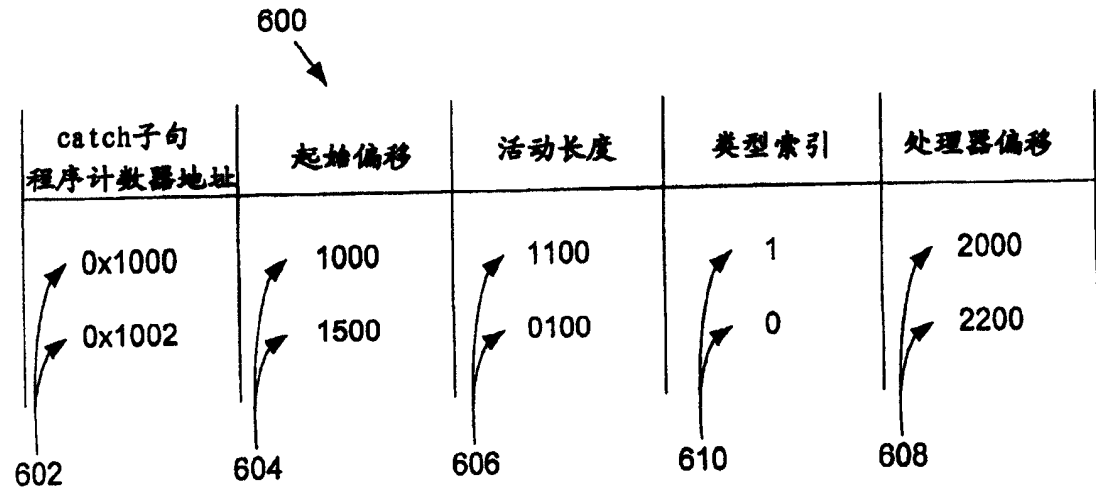


图 6

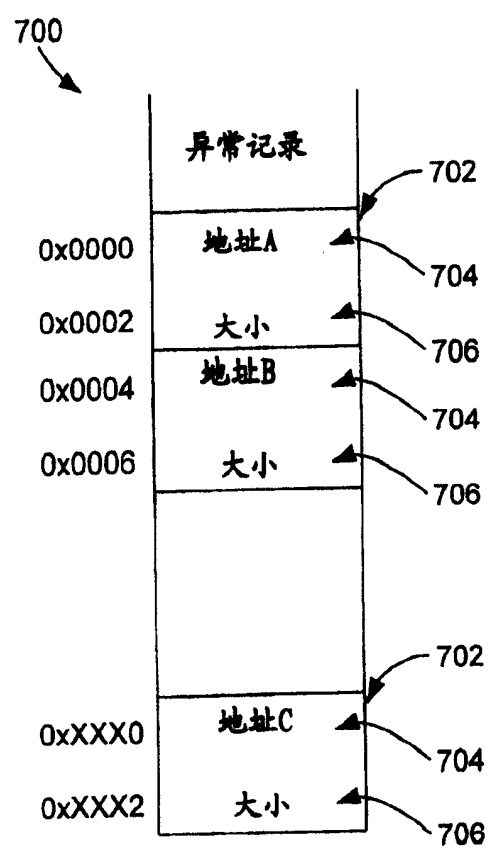


图 7

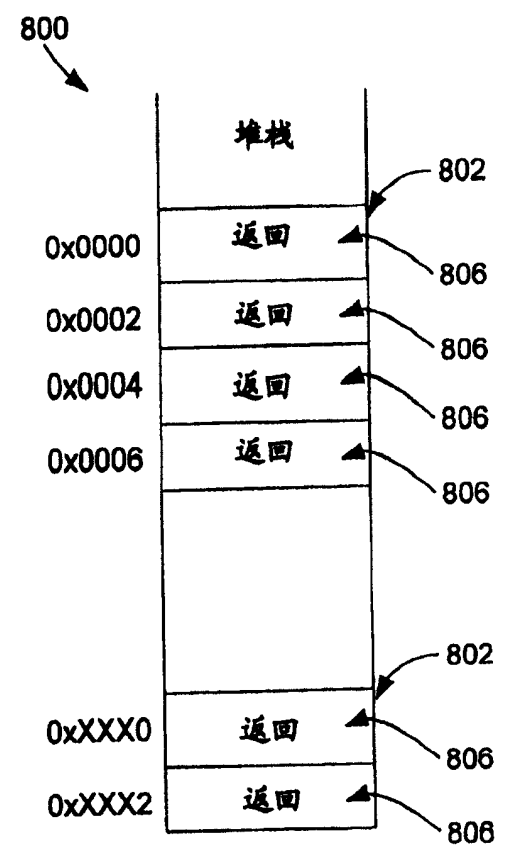


图 8

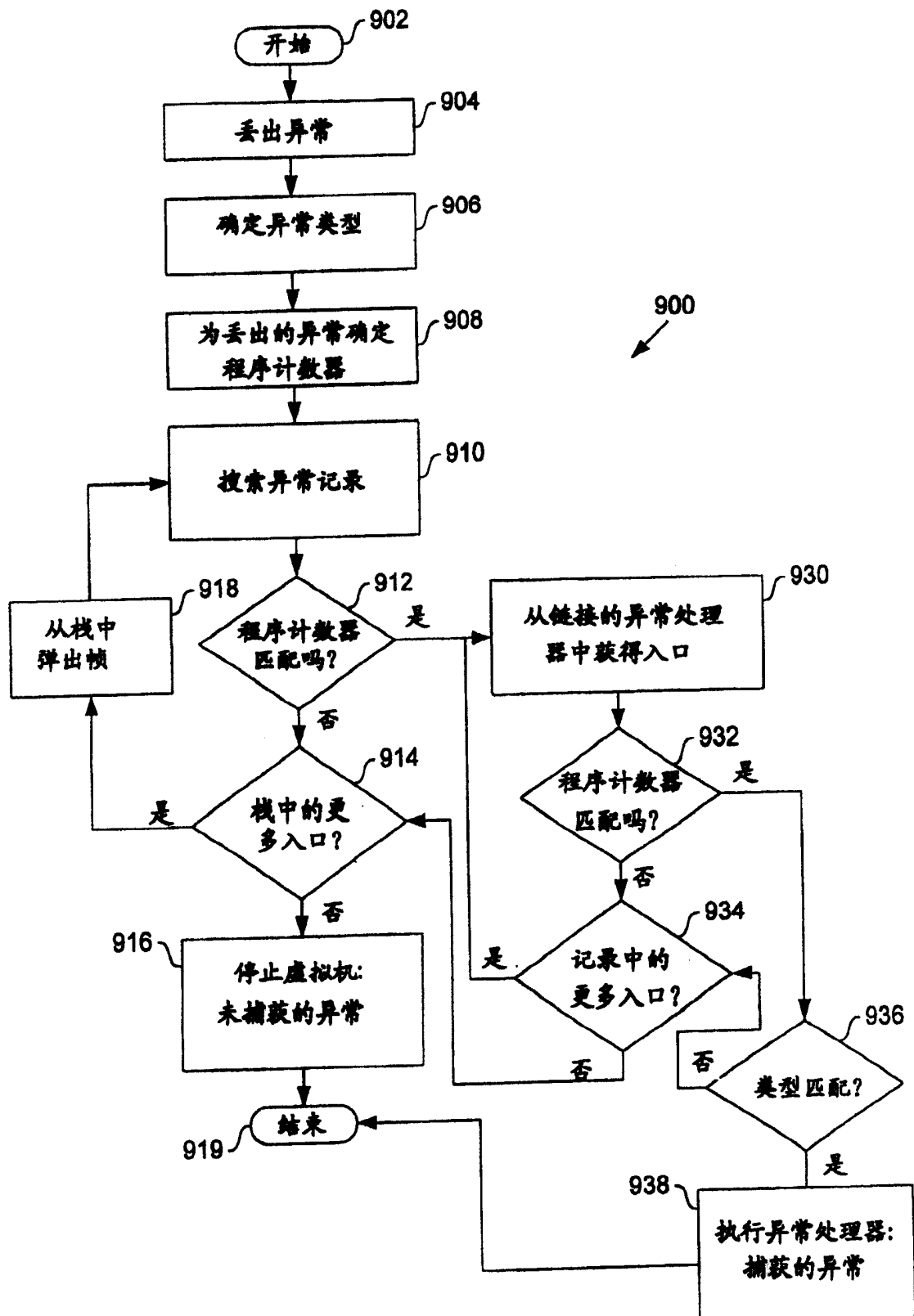


图 9

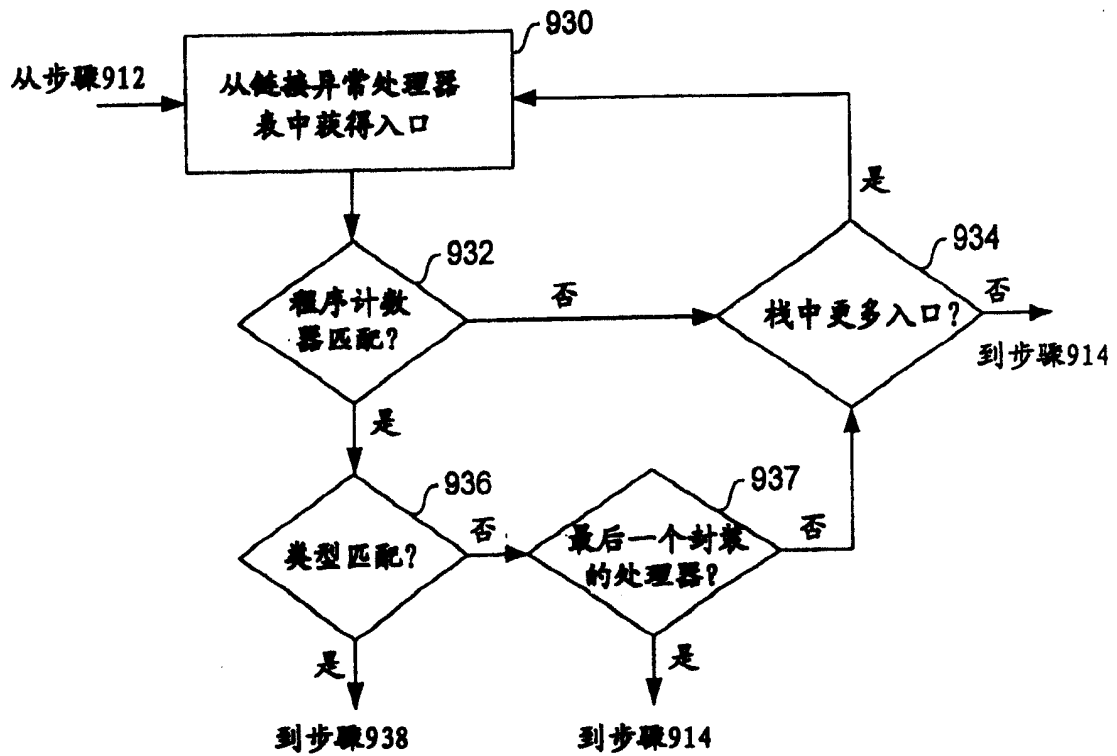


图 10a

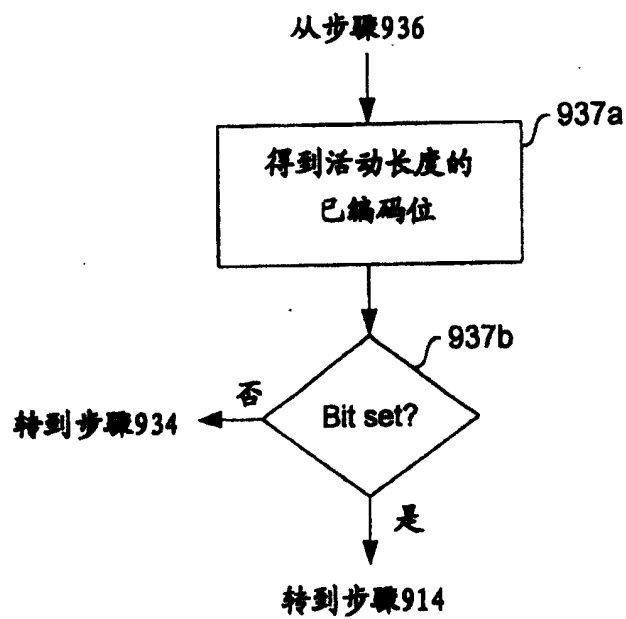


图 10b