



## [12] 发明专利说明书

[21] ZL 专利号 99104080.5

[45] 授权公告日 2004 年 3 月 24 日

[11] 授权公告号 CN 1143212C

[22] 申请日 1999.3.23 [21] 申请号 99104080.5

[30] 优先权

[32] 1998.3.23 [33] US [31] 60/079110

[32] 1998.6.30 [33] US [31] 09/107224

[71] 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 L·巴克 S·米特洛维

U·霍尔兹勒

审查员 韩 燕

[74] 专利代理机构 中国专利代理(香港)有限公司

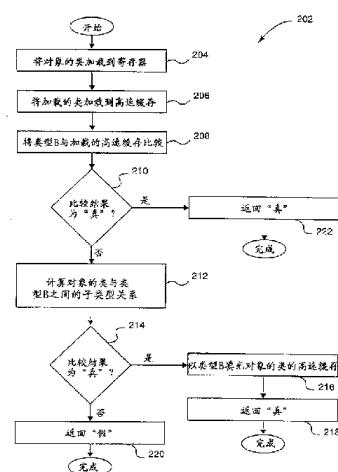
代理人 王 勇 王 岳

权利要求书 5 页 说明书 10 页 附图 5 页

[54] 发明名称 实现快速子类和子类型检查的方法与设备

## [57] 摘要

本文披露了在程序执行期间进行快速子类型检查的方法和装置。根据本发明的一个方面，判定与作为基于对象的计算系统一部分的对象关联的类是否是另一个类型的子类型的方法，包括从与该对象关联的类所关联的动态存储位置获取一个候选类型，将该候选类型与潜在地与该候选类型相同的第一类作比较。然后判断该候选类型实际上是否等于第一个类型。如果确定该候选类型实际上等于第一个类型，就提供一个表示该候选类型实际上等于第一个类型的子类型的指示。



5

1. 一种计算机实现的方法，用于确定与作为一个基于对象的计算系统一部分的对象相关联的类是否是另一个类型的子类型，该方法包括：

获取一个候选类型，该候选类型是从与该对象关联的类相关的一个动态存储位置获取的；

将该候选类型与第一个类型作比较；

判断该候选类型是否等于第一个类型；

10

如果判定该候选类型等于该第一个类型，就提供一个表示该候选类型是该第一个类型的子类型的指示。

2. 权利要求 1 所述的计算机实现的方法，其中，从与该对象关联的类相关的动态存储位置获取的该候选类型，是从该对象关联的类的高速缓存单元获取的。

15

3. 权利要求 2 所述的计算机实现的方法，其中，将该候选类型与第一个类型作比较包括：

将该候选类型从高速缓存单元装入寄存器；

将高速缓存单元的内容与第一个类型比较。

20

4. 上述任一个权利要求所述的计算机实现的方法，其中，如果判定该候选类型不等于第一个类型，则该方法进一步包括：

计算该对象的类与第一个类型之间的类型关系。

25

5. 权利要求 4 所述的计算机实现的方法，进一步包括：判断该对象的类与第一个类型之间是否存在一个类型关系，其中，如果判定该对象的类与第一个类型之间存在着类型关系，则将第一个类型的指示存储到该对象的类的高速缓存单元中。

6. 权利要求 5 所述的计算机实现的方法，进一步包括将该对象的类装入寄存器。

7. 一个计算机系统，配置用于确定与驻留在该计算机系统上的第一个对象关联的子类型，该计算机系统包括：

30

一个处理器；

一个将一个候选类型装入与计算机系统关联的内存的加载机构，该候选类型是从与第一个对象关联的类对象所关联的存储位置获取的；

5 一个配置成用于将该候选类型与第一个类型比较的比较机构；

一个配置成用于确定该候选类型是否等于第一个类型的判断机构；

10 一个配置成用于在确定该候选类型等于该第一个类型时，提供一个表示该候选类型等于第一个类型的子类型的指示的指示器。

8. 根据权利要求 7 的计算机系统，其中，从该类对象获取的该候选类型是从该类对象的高速缓存单元获取的。

9. 根据权利要求 8 的计算机系统，其中，配置成用于将该候选类型与该第一个类型比较的比较机构包括：

15 配置成用于将该候选类型从该高速缓存单元装入寄存器的加载机构；

配置成用于将该高速缓存单元的内容与第一个类型比较的比较机构。

20 10. 根据权利要求 7~9 中任一项权利要求的计算机系统，进一步包括，配置成用于在确定该候选类型不等于第一个类型时，计算该类对象与第一个类型之间的类型关系的计算机构。

11. 根据权利要求 10 的计算机系统，进一步包括：

配置成用于判断该类对象与第一个类型之间是否存在类型关系的判断机构；

25 配置成用于在确定该类对象与第一个类型之间存在着类型关系时，将第一个类型的标志存储到该类对象的高速缓存单元的存储机构。

12. 根据权利要求 7 的计算机系统，其中，可操作存储单元以包括子类型检查结果。

30 13. 根据权利要求 7 的计算机系统，其中，可操作存储单元

以包括在先的子类型检查结果。

14. 一种计算机实现的方法，用于对属于一个特定类的成员的对象执行子类型检查，该方法包含：

从与该特定类相关联的一个位置获取一个存储单元，该存储单元包括潜在地与该对象关联的第一个子类型的有关信息；

判断该存储单元所含信息是否与该对象相关联的一个实际子类型有关；

在确定该存储单元的信息与该实际类型有关时，提供一个表示该存储单元所含信息与该实际子类型有关的指示。

15. 权利要求 14 所述的计算机实现的方法，进一步包括：

在确定该存储单元所含信息与该实际子类型无关时，判断与该对象相关的该实际子类型；

并将与该实际子类型有关的信息存储到与该特定类关联的位置。

16. 一种计算机实现的方法，用于确定与作为基于对象的计算系统一部分的对象关联的类是否是另一个类型的子类型，其中该对象属于第一个类型，该方法包括：

获取第二个类型，该第二个类型是该对象的候选类型；

将第一个类型与第二个类型作比较；

判断第二个类型是否等于第一个类型；

如果判定第二个类型等于第一个类型，就将与第二个类型关联的信息存储到该对象关联的类所关联的动态存储位置，其中，在该动态存储位置存储信息允许该信息被访问，用于随后的类型检查。

25 17. 权利要求 16 所述的计算机实现的方法，进一步包括：

在确定第二个类型等于第一个类型时，提供第二个类型是第一个类型的子类型的指示。

18. 权利要求 16 或 17 所述的计算机实现的方法，其中，获取第二个类型包括从包含多个类型的数据结构中获取第二个类型。

19. 权利要求 18 所述的计算机实现的方法，进一步包括创建包含多个类型的数据结构。

20. 权利要求 16 所述的计算机实现的方法，进一步包括：  
5 在确定第二个类型不等于第一个类型时，获取第三个类型，  
该第三个类型是该对象的候选类型，其中该第三个类型是从包含  
多个类型的数据结构中获取的；

将第一个类型与第三个类型作比较；  
判断第三个类型是否等于第一个类型；  
如果判定第三个类型等于第一个类型，就将与第三个类型关  
10 联的信息存储到该动态存储位置。

21. 一种计算机实现的方法，用于确定与作为基于对象的计算系统一部分的对象关联的类是否是另一个类型的子类型，该方法包括：

将第一个候选类型存储到与该对象关联的类所关联的一个动  
15 态存储位置；

从该动态存储位置获取第一个候选类型；  
将第一个候选类型与某对象类型作比较，该对象类型是与该  
对象关联的对象类型；

判断第一个候选类型是否与该对象类型相同；  
如果判定第一个候选类型与该对象类型相同，就提供一个表  
示第一个候选类型是该对象类型的子类型的指示。

22. 权利要求 21 所述的计算机实现的方法，其中，当判定  
第一个候选类型与该对象类型不相同时，该方法进一步包括：

25 获取一个第二个候选类型，其中该第二个候选类型不是从该  
动态存储位置获取的；

将第二个候选类型与该对象类型作比较；  
判断第二个候选类型是否与该对象类型相同；  
如果判定第二个候选类型与该对象类型相同，就提供一个表  
示第二个候选类型是对象类型的子类型的标志；

30 将第二个候选类型存储到该动态存储位置。

- 
23. 一个计算机系统，配置成用于确定驻留在该计算机系统上的第一个类型的对象所关联的子类型，该计算机系统包括：
- 5      一个处理器；
- 一个配置成用于获取第二个类型的第一机构，其中该第二个类型是该对象的一个候选类型；
- 用于将第一个类型与第二个类型比较的比较器；
- 用于确定第二个类型是否等于第一个类型的判断机构；
- 10     用于在确定第二个类型等于第一个类型时，将与第二个类型关联的信息存储到与该对象关联的类所关联的一个动态存储位置的动态存储机构，该动态存储位置存储的该信息用于在随后的类型检查中被访问。

## 实现快速子类和子类型检查的方法与设备

本发明总体涉及基于对象的系统的对象之间关系的确定，更具体  
5 地说，本发明涉及对基于对象的系统中的对象有效地执行子类型检  
查。

许多基于对象的计算系统是这样构造的，其中的对象是定义了可  
10 用于对象的功能的特定类和子类的成员。在程序执行期间，虚拟机一  
般涉及检查对象之间的关系，以利程序的执行。举例来说，虚拟机可  
以检查对象之间的子类或子类型关系。在有些程序设计语言、例如美  
国加州 Palo Alto 的 Sun Microsystems 公司开发的 Java™ 程序设计  
语言中，程序设计语言内的构造涉及子类检查。这种子类检查一般涉  
及判断某特定对象是否是给定类型的。就是说，检查与程序关联的类  
结构以确定特定对象的类型。

15 图 1 是一种常规类结构的图示。类结构 102 是一个类分层结构，  
包括类 106 和子类 110。一般来说，类 106 是一个抽象类，可以包含  
任意数量的子类 110。如图所示，子类 “1” 110a、子类 “2” 110b 和  
子类 “N” 110c 是类 106 的“直接” 子类，而子类 “A1” 110d 是子类  
“1” 110a 的直接子类。子类 “A1” 110d 可看作是类 106 的间接子类，  
20 因为子类 “A1” 110d 是子类 “1” 110a 的子类，后者是类 106 的子类。

类 106 一般包括各种不同的函数 (function) 或过程 (method)。  
每个子类 110 通常包括一个不同的函数集合。举例来说，子类 “1” 110a  
一般包括专门用于作为子类 “1” 110a 的一部分的对象的函数。作为  
类 106 的成员的对象实际上可以执行与类 106 相关的所有函数。是任  
何子类 110 的成员的一个对象也是类 106 的成员。因此，是任何子类  
110 的成员的一个对象也可以执行与类 106 相关联的所有函数。然而，  
25 是某特定子类例如子类 “1” 110a 的成员的对象，却不能执行与不同  
的子类例如子类 “2” 110b 相关联的特定函数。因此，判断某对象属  
于哪个子类 110，就能有效地确定该对象所能执行的函数。

30 运行时可用一种窄投射 (narrowing cast) 来有效地将由类 106  
定义的对象作为由子类 “1” 110a 定义的对象来看待。然而，由于由  
类 106 定义的对象，可以由子类 “2” 110b 定义而不是由子类 “1” 110a

5 定义，一般要进行检查以确定将该对象与子类“1”110a 关联是否准确。本领域的熟练人员会明白，关于某对象是否与子类“1”110a 关联的检查，实际上就是判断该对象是否至少与子类“1”110a 关联的检查。换言之，与子类“A1”110d 关联的某对象一般会被确定为也与子类“1”110a 关联。

10 在 Java™ 环境中，确定对象的子类型的函数，例如 `is_subtype` 函数，可以是静态编码的。尽管用来静态编码函数的方法会有所不同，一种常用的方法是采用一个二维位矩阵，其中位于  $(i, j)$  位置的一个位编码 `is_subtype(ti, tj)` 的结果。采用这种矩阵，子类型检查实际上涉及变址到矩阵去确定对象的子类型。然而，由于矩阵规模可能很大，并且由于一般要求的指令的位操作，所以子类型检查的速度经常很慢。

15 一般来说，为了确定某特定对象的子类型而进行子类型检查时，通常必须检查一个类型的几乎所有子类型，例如一个类的几乎所有子类。在有些层次化类结构中，例如图 1 的类结构 102 中，必须检查的子类的数量会相对较多。例如，某些类可能具有数百个相关的子类。因此，当有多个子类型时，例如在有用 Java™ 程序设计语言定义的接口的情况下，子类型检查的执行效率通常不高。就是说，当有多个子类型时，检查每个子类型一般都很费时，正如上文所述。此外，在使用 20 多个继承层（*inheritance layer*）的系统例如 C++ 程序设计语言定义的系统中执行子类型检查经常也效率不高。对于有多个继承层的系统来说，有效地执行子类型检查一般有困难，因为必须检查每个继承层。

25 有效地执行子类型检查是重要的，因为检查可能频繁地发生。在程序执行期间频繁发生检查时，与检查相关联的开销就会相对较高。在有些情况下，一个运行时子类型检查或测试，可能要求相当于大约 8 条指令量级的开销，本领域的熟练人员知道，这对于整体程序来说是相当大的开销，在重复进行运行时子类型检查时尤其如此。因此，程序执行的速度由于频繁的子类型检查而受抵销。

30 一般来说，当在程序执行期间进行子类型检查时，实际上所有与程序关联的类和过程都必须是已知的。通常要构造数据结构来列举所有与程序关联的类和过程，以便这些类和过程处于准备好被存取的状

态。换言之，子类型检查中所用的数据结构通常必须在程序执行之前计算出来。这种数据结构通常相当大，耗费相当多的系统资源。所有与程序关联的类和过程都必须是已知的这个要求，是与采用动态连接或动态类加载的系统不兼容的，因为动态连接允许与程序关联的类和过程有效地改变。如果不能采用动态连接，程序的功能就会受到影响。在采用动态连接的环境中，每次进行涉及类加载的操作后，数据结构一般都要重新计算，这就耗费时间，因而效率不高。

因此，需要一种提高子类型检查效率的方法和装置。更特别地，需要一种提高子类型检查效率、并且每次类加载操作后不要求重新计算数据结构的方法和装置。

本发明披露在程序执行期间进行快速子类型检查的方法和装置。根据本发明的一个方面，快速有效地确定作为与基于对象的计算系统一部分的对象相关联的类型的方法，包括从与一个类（该类与该对象相关联）相关联的动态存储位置获取一个候选类型，将候选类型与潜在地与该候选类型相同的第一个类型作比较。然后判断候选类型是否等于第一个类型。如果确定候选类型等于第一个类型，就提供一个表示候选类型等于第一个类型的指示。

根据本发明的另一个方面，安排一个计算机系统来确定驻留在计算机系统上的第一个对象相关联的类型。计算机系统包括处理器、内存和将候选类型装入内存的加载机构。候选类型是从与第一个对象关联的类对象获取的。计算机系统还包括一个将候选类型与第一个类型比较的比较机构，一个能确定候选类型是否等于第一个类型的判断机构。计算机系统中有一个指示器，用于在确定候选类型等于第一个类型时，提供一个表示候选类型等于第一个类型的指示。

根据本发明再另一个方面，对属于一个特定类的成员的对象执行子类型检查的方法，包括从与该特定类相关联的某位置获取一个存储单元。该存储单元包括潜在地与该对象相关联的第一个子类型的有关信息。该方法还包括判断存储单元所含信息是否和与该对象相关联的实际子类型有关，并且在存储单元与实际类型有关时提供一个表示存储单元所含信息与实际子类型有关的指示。在一个实施例中，该方法还在存储单元所含信息与实际子类型无关时，判断对象的实际子类

型，并将与实际子类型相关的信息存储到该特定类的关联位置。

通过阅读以下的详细说明，研究各种附图，能更好理解本发明。

参考以下结合附图的说明，就能理解用具体实施例阐述的本发明。各附图简介如下：

5 图 1 是一种常规类分层结构的图示。

图 2a 是表示按照本发明的实施例，判断某对象是否是特定类型的子类型的相关步骤的流程图。

图 2b 是表示按照本发明的实施例，将一个类型与一个所加载的高速缓存比较—即图 2a 的步骤 208 的相关步骤的流程图。

10 图 2c 是表示按照本发明的一个实施例，具有一个高速缓存单元的类的图示。

图 3 是适于实现本发明的计算机系统的图示。

图 4 是按照本发明一个实施例的虚拟机的图示。

15 一般来说，判定对象之间子类型关系所执行的子类型检查要占用相当多的计算机内存空间。要求占用这个空间的，是与执行子类型检查相关的指令，以及预先计算的、通常用于存储所有与计算机程序的执行相关的所有类型和过程的数据结构。子类型检查要求占用的空间，以及执行反复的子类型检查所需开销，经常影响程序执行的速度。

20 通过保存子类型检查的预期结果，可以有效地减少执行子类型检查相关的开销量。具体来说，运行时，在首次要将与某对象关联的子类型针对某特定类型进行检查时，实际上可以采用任何方法来进行实际的检查，本领域的熟练人员是清楚这一点的。因为随后对相同类的对象进行的检查可能得出相同的子类型结果，所以保存实际检查结果，例如首次检查的计算结果，会显著降低与进行子类型检查相关的开销。换言之，这是一种动态子类型检查机制，它采用高速缓存技术，  
25 允许将以前的子类型检查结果高速缓存起来，随后在执行同样的检查时，如果判定所高速缓存的值正确即适当的话就允许使用高速缓存的值，从而加快同样检查的速度。在一个实施例中，高速缓存的数据可以被保存在正被测试或检查的值—例如对象“S”的类型描述符中。

30 在执行了确定与某个类关联的某对象“S”的子类型的初始检查之后，初始检查的结果被存储起来。由于存储了初始子类型检查的结果，下一次要求检查确定对象“S”的子类型时，存储的结果就能使相同的

子类型检查的进行速度加快。举例来说，当要将对象“S”的子类型对照类型“T”作检查时，即要判断对象“S”是否属于类型“T”时，就能相对快速地将与对象“S”关联的存储的类型描述符与子类型“T”进行比较。如果对象“S”的类型等于子类型“T”，则认为比较结果是成功的，一般就不再要求进行常规的子类型检查。因此可避免与一个常规子类型检查相关的开销。另一方面，如果对象“S”的类型不等于子类型“T”，则要用常规的子类型检查来确定对象“S”的类型。在某些情况下，这种检查要遍历一个类层次结构以定位适当的子类型，如果找不到适当的子类型，就产生一个异常。

参见图 2a，现在来说明按照本发明的实施例，判断某对象是否是特定类型的子类型或子类的相关步骤。下面具体说明判断某对象是不是类型 B 的子类型的流程。判断某对象是不是类型 B 的子类型的流程 202 始于步骤 204，该步骤将对象的类型，例如类，装入与计算机系统相关的寄存器。本领域的熟练人员明白，对象的总类即抽象类是已知的。一旦对象的类被装入寄存器，然后在步骤 206，加载的类的高速缓存单元就被加载，例如加载到计算机内存。下文将结合图 2c 来说明高速缓存单元或高速缓存。

高速缓存被加载后，在步骤 208，将类型 B 对照该加载的高速缓存进行比较。将类型 B 与加载的高速缓存比较的相关步骤将在下文结合图 2b 来说明。在步骤 210，对类型 B 与加载的高速缓存的比较结果进行判断。在所述实施例中，如果类型 B 与加载的高速缓存的比较产生一个真结果，即类型 B 与加载的高速缓存的比较确定它们匹配，则在步骤 222 将真值返回给请求判定对象是否是类型 B 的子类型的函数。换言之，如果加载的类与类型 B 是相同的类，或者如果加载的类是类型 B 的子类型，则返回真值。一旦返回真值后，判断对象是否是类型 B 的子类型的流程即告完成。

另一方面，如果步骤 210 判定，类型 B 与加载的高速缓存的比较得不出真结果，则在步骤 212 计算对象的类与类型 B 之间的子类型关系。就是说，当判定对象的类与类型 B 不相同时，就要通过计算来确定对象的类与类型 B 之间的子类型关系，如果存在的话。计算子类型关系之后，过程流继续到步骤 214，判定对子类型的计算是否产生真结果，即是否对象的类与类型 B 之间存在一个有效的子类型关系。

如果步骤 214 判定，对象的类与类型 B 之间不存在子类型关系，则在步骤 220，向请求子类型检查的函数返回一个假值。一旦返回假值后，判断对象是否是类型 B 的子类型的流程即告完成。然而，如果步骤 214 判定，对象的类与类型 B 之间存在子类型关系，则过程流继续到步骤 216，在该步骤中，用类 B “填充” 对象的类的高速缓存。换言之，类 B 或者说对类型 B 的引用，被存储到该对象的类的高速缓存单元。最后，在用类型 B “填充” 该对象的类的高速缓存之后，在步骤 218 向系统返回一个真值，以指示对象的类与类型 B 之间存在子类型关系。然后，判断一个对象是否是类型 B 的子类型的流程即告完成。

下面结合图 2b，说明按照本发明的实施例，将类型 B 与加载的高速缓存—例如某特定对象的类进行比较的相关步骤。就是说，要讨论的是图 2a 的步骤 208 的一个实施例。类型 B 与加载的高速缓存的比较始于步骤 232，将高速缓存单元装入一个寄存器。特别地，将该对象的类的高速缓存单元存入一个寄存器（下文将结合图 2c 说明高速缓存单元）。高速缓存单元被存入寄存器后，在步骤 234 将寄存器的内容与类型 B 作比较。如上所述，一种采用高速缓存技术的检查机制允许将以前的子类型检查结果高速缓存起来，目的是在随后执行相同检查时通过使用该高速缓存的值来加快相同检查的速度。由于检查时采用高速缓存的单元，可以随时访问该单元。几乎可采用任何适当方法来比较高速缓存单元与寄存器的内容。本领域的熟练人员一般都很了解这类方法。一旦寄存器内容与类型 B 作比较后，比较类型 B 与加载的高速缓存的过程即告完成。

如上所述，类例如类对象包括一个可在其中存储以前的子类型检查结果的高速缓存单元。图 2c 是表示按照本发明的一个实施例，具有高速缓存单元的类的图示。对象 “S” 252 包括首部 256，并有一个指向类 260 的类指针。类 260 包括首部 264 和高速缓存单元 268。前文说过，高速缓存单元 268 被用来存储以前—例如第一个—与类 260 关联的子类型检查的结果。应当明白，当类 260 被初始化时，高速缓存单元 268 一般可初始化为任意值。举例来说，可将高速缓存单元 268 初始化来标识类 260。一般来说，高速缓存单元 268 可看作是个“动态” 存储单元，这是因为，随着程序的执行，或更确切地说，随着子类型检查的执行，高速缓存单元 268 中存储的结果可能会改变。就是

说，高速缓存单元 268 的内容可以被更新。

如果已知对象 “S” 252 是类 260 的成员时，则在涉及对象 “S” 252 的子类型检查期间，可以访问高速缓存单元 268，以获取涉及类 260 的最新子类型检查结果。一般来说，可以更新高速缓存单元 268，以存储涉及类 260 的最新子类型检查结果。这样，如果高速缓存单元 268 存储的结果不是与对象 “S” 252 关联的子类型时，则一旦在确定了与对象 “S” 252 关联的实际子类型时，就可以将其存储到高速缓存单元 268。应当明白，在有些实施例中，向高速缓存单元 268 存储以及从中检索信息可能涉及到同步，以解决可能出现的高速缓存相干性问题。

图 3 表示一个典型的适于实现本发明的通用计算机系统。计算机系统 330 包括任意多个与存储器连接的处理器 332 即中央处理单元 (CPU)。存储器一般包括诸如随机存取存储器 (RAM) 的主存储器 334 和诸如只读存储器 (ROM) 的主存储器 336.

可以安排计算机系统 330—或更确切地说—各 CPU，去支持一个虚拟机。本领域的熟练人员明白这一点。后文将结合图 4 描述计算机系统 330 上支持的一例虚拟机。本领域众所周知，ROM 334 的作用是单向地向各 CPU 332 传输数据和指令，而 RAM 336 的作用则是双向地向各 CPU 332 传输数据和指令和从其接收数据和指令。两种主存储器 334、336 都可以包括几乎任何适当的计算机可读介质。二级存储介质 338 一般来说是海量存储器，也能双向地连接到各 CPU 332。一般来说，二级存储介质 338 可以配置来提供额外的存储容量，可以是一种用于存储计算机代码、计算机程序代码设备、数据等等的计算机可读介质。一般来说，二级存储介质 338 是诸如硬盘或磁带的存储介质，速度比主存储器 334、336 慢。二级存储介质 338 的形式可以是众所周知的设备，包括（但不限于）磁带机和纸带机。本领域的熟练人员明白，二级存储介质 338 中保存的信息在适当情况下，可像 RAM 336 的一部分那样以标准方式被引用，例如作为虚拟内存。一种诸如 CD-ROM 的专用主存储器 334 也可以单向地向各 CPU 传送数据。

各 CPU 也连接到一个或多个输入 - 输出设备 340，输入 - 输入设备包括（但不限于）视频监视器、轨迹球、鼠标器、键盘、麦克风、触摸敏感显示屏、传感器读卡机、磁带或纸带阅读机、图形输入板、

输入笔、语音或手写识别器、以及其它已知的输入设备诸如其它计算机。最后，CPU 332 也能采用 312 所示的网络连接，可选地连接到计算机或电信网络，例如因特网或内部网。采用了这种网络连接 312，就可认为各 CPU 332 能从网络接收数据。CPU 332 在执行上述各方法 5 步骤的过程中也能向网络输出数据。这种数据通常表示为一个待由各 CPU 332 执行的指令序列，可以例如实现在载波中的计算机数据信号从网络接收和向网络输出。计算机硬件和软件领域的熟练人员都熟悉上述的设备和材料。

上文说过，可以在计算机系统 330 上执行一个虚拟机。图 4 是图 3 10 的计算机系统 330 支持的、适于实现本发明的虚拟机的图示。如果要执行计算机程序，例如用 Java™ 程序设计语言编写的计算机程序，要将源代码 410 提供给编译时环境 405 内的编译器 420。编译器 420 将源代码 410 翻译成字节代码 430。一般来说，源代码 410 在软件开发者创建了源代码 410 时就被翻译成字节代码 430。

15 字节代码 430 一般来说可以复制、下载或以其它方式通过例如图 3 中网络 312 的网络传播，或者在诸如图 3 中主存储器 334 的存储器上存储。在所述实施例中，字节代码 430 是独立于平台的。就是说，字节代码 430 可以在运行在适当虚拟机 440 上的几乎任何计算机系统上执行。

20 字节代码 430 被提供给包含虚拟机 440 的运行环境 435。在一个实施例中，虚拟机可以是 Java™ 虚拟机。运行时环境 435 一般可采用诸如图 3 中 CPU 的一个或多个处理器来执行。虚拟机 440 包括编译器 440、解释器 444 和运行时系统 446。字节代码 430 可提供给编译器 442 或解释器 444。

25 如果将字节代码 430 提供给编译器 442，则字节代码 430 中包含的过程就被编译成机器指令。在一个实施例中，编译器 442 是一个适时的编译器，它能将对字节代码 430 所含过程的编译，推迟到该过程马上要被执行时才进行。如果将字节代码 430 提供给解释器 444，则字节代码 430 以每次一个字节代码的方式被读入解释器 444。每当一个字节代码被读入解释器 444，解释器 444 就执行该字节代码所定义的操作。就是说，解释器 444 “解释” 字节代码 430，这一点本领域的熟练人员都明白。一般来说，解释器 444 几乎是连续地处理字节代码 30

和执行与字节代码 430 相关联的操作。

当一个过程被另一个过程调用，或从运行时环境 435 调用时，如果该过程是个解释过程，则运行时系统 446 可以从运行环境 435 以一个字节代码 430 序列的形式获取该过程，这种形式的过程可被解释器 444 直接执行。另一方面，如果被调用的过程是个尚未编译的编译过程，则运行时系统 446 也是从运行时环境 435 以一个字节代码 430 序列的形式获取该过程，然后在下一步去启动编译器 442。编译器 442 于是从字节代码生成机器指令，所生成的机器语言指令可由图 3 的 CPU 直接执行。一般来说，当虚拟机 440 中止时，机器语言指令即被抛弃。

尽管本文仅描述了本发明的一些实施例，应当明白，本发明可以以许多其它形式来实现而不偏离本发明的精神和范围。例如，在有些实施例中，类对象可以包含一个以上的高速缓存单元。如果一个类对象包含一个以上的高速缓存单元，则可以存储先前的一个以上的子类型检查结果。就是说，对于某个对象，可以存储一个以上的可能子类型，这样，如果确定一个高速缓存单元中存储的子类型不是该对象的子类型时，就检查在另一个高速缓存单元中存储的子类型。

尽管本发明是就在类的高速缓存单元中存储子类型检查结果而进行说明的，应当明白，以前的结果并不是非要在高速缓存单元中存储。相反，以前的结果可以被存储在几乎任何在子类型检查期间能访问到的动态存储单元中。例如，可以将以前的子类型检查结果存储到一段与该类并无直接关联的计算机代码中。另一种方法是，可以将以前的子类型检查结果存储到一个动态的、可全局访问的表中，每当执行一个涉及某特定类的子类型检查时就对其进行访问。这样一个表可以直接与该特定类相关。

应当明白，在一个实施例中，如果不采用为判定某特定对象是否是某个子类型的对象的检查，可转而采用为判定某特定对象是否不是某个子类型的对象的检查。这种检查的结果一般可存储在一个类的高速缓存单元中、计算机代码段中或作为全局表的一部分。换言之，可以用高速缓存单元来保存对于一个特定子类型检查一个可能不会匹配的子类型名。

一般来说，使用子类型检查的指令或操作，会因为特定系统的要求而差别很大。例如在 Java™ 环境中，“astore” 指令、“checkcast”

指令和“instanceof”指令一般采用子类型检查。

此外，根据本发明进行子类型检查的相关步骤也可以变化。在不偏离本发明的精神和范围的情况下，这些步骤一般可以修改、变化次序、增加或者去除。例如，对比较和计算是否产生一个“真”标志的  
5 判断，可以改为对比较和计算是否产生一个“假”标志的判断。另外，如果类对象包括一个以上的高速缓存单元，则进行子类型检查的相关步骤就可以包括循环步骤，循环地测试每个高速缓存单元，直到找出子类型匹配或者测试完所有高速缓存单元时才结束循环。因此，本文  
10 中的例子应视为是示例性的而不是限制性的，本发明并不局限于本文所述的细节，而是由后附的权利要求及其等同的全部范围所规定的。

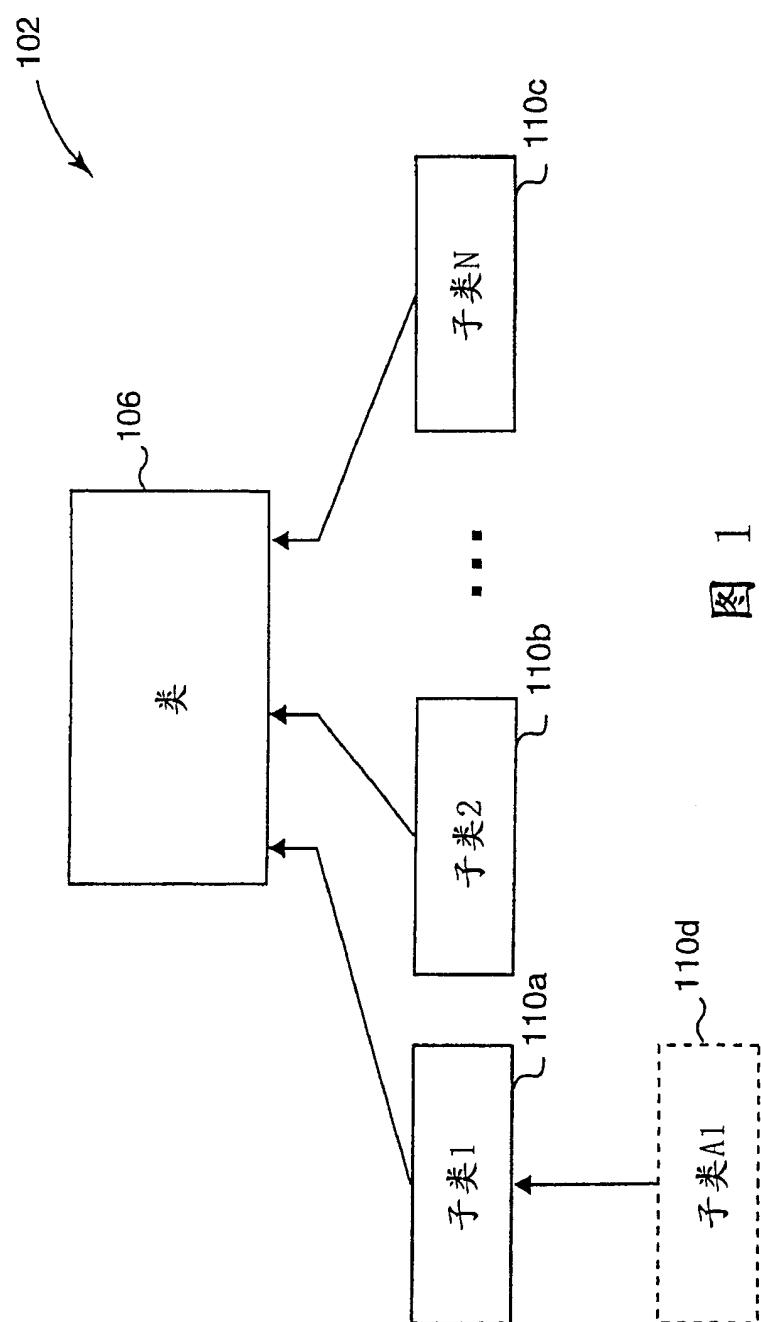


图 1

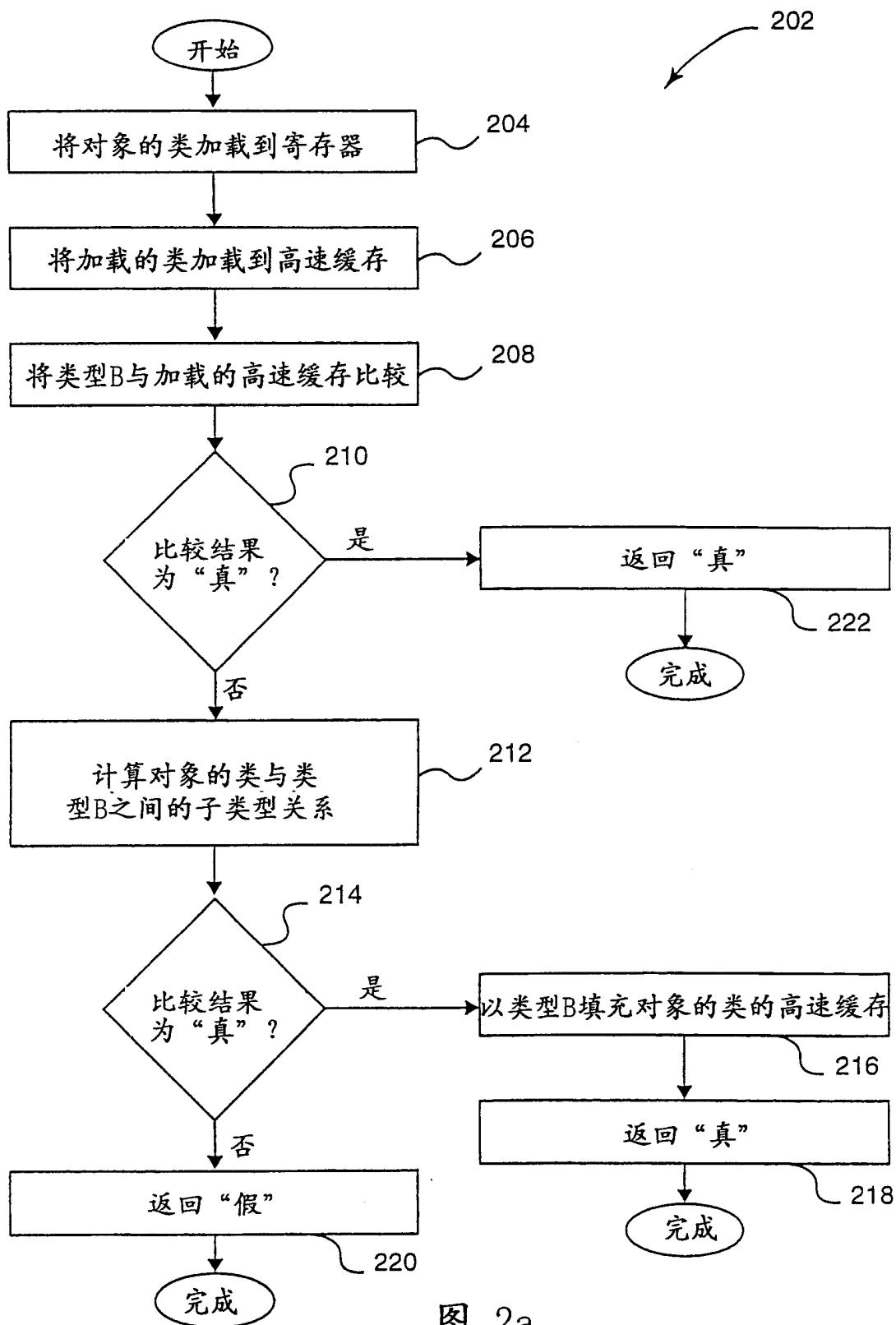


图 2a

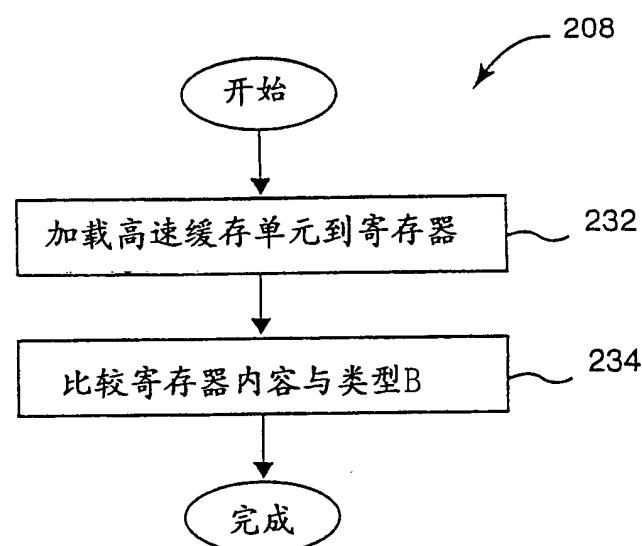


图 2b

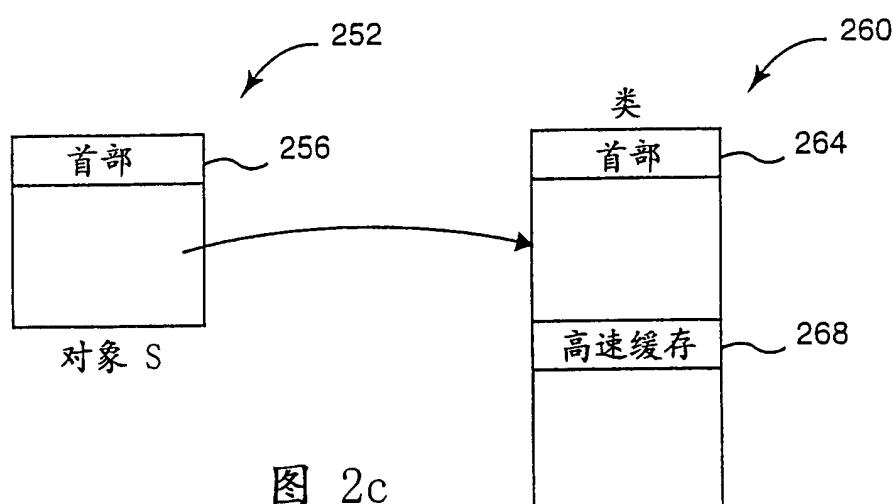


图 2c

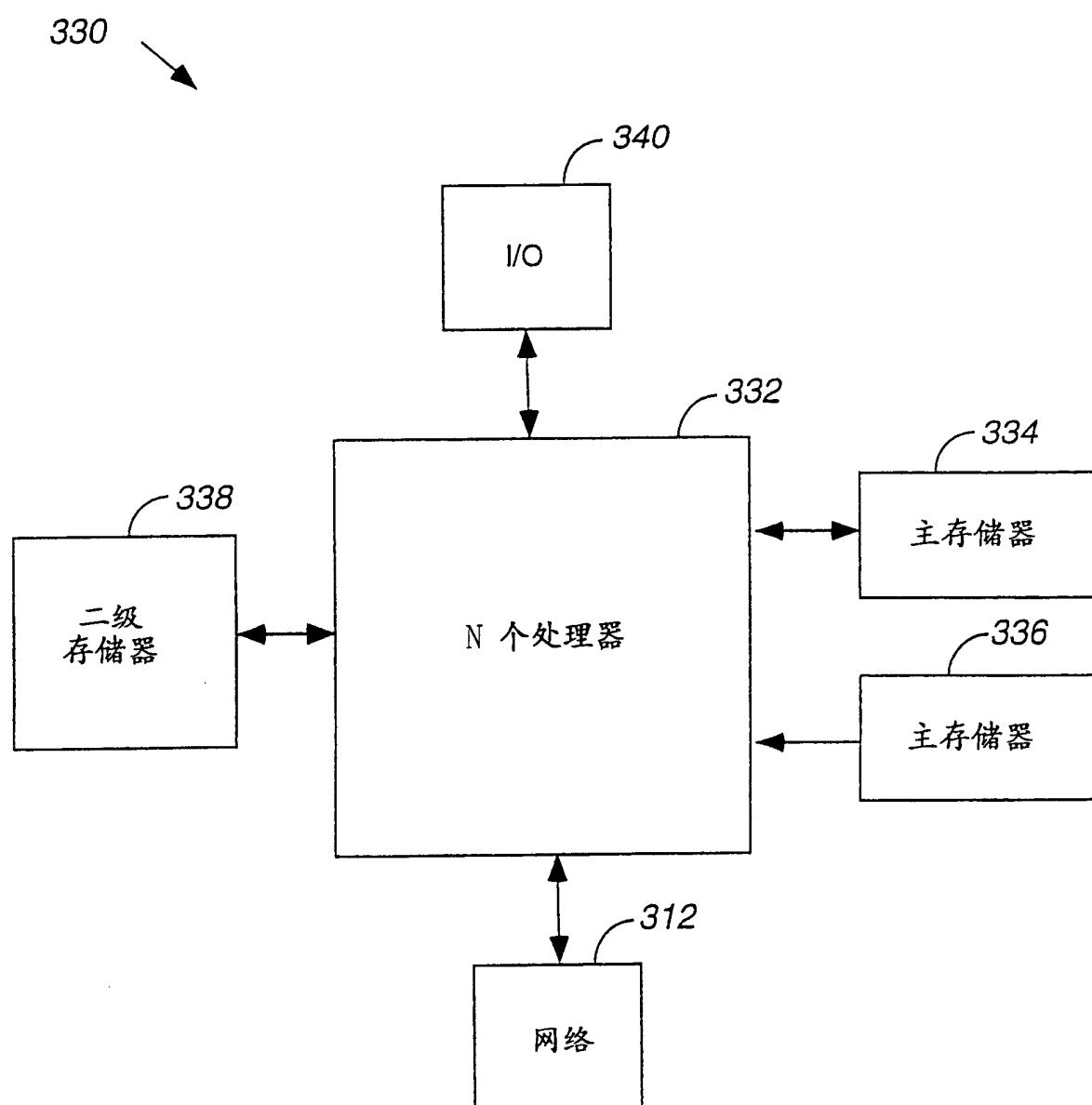


图 3

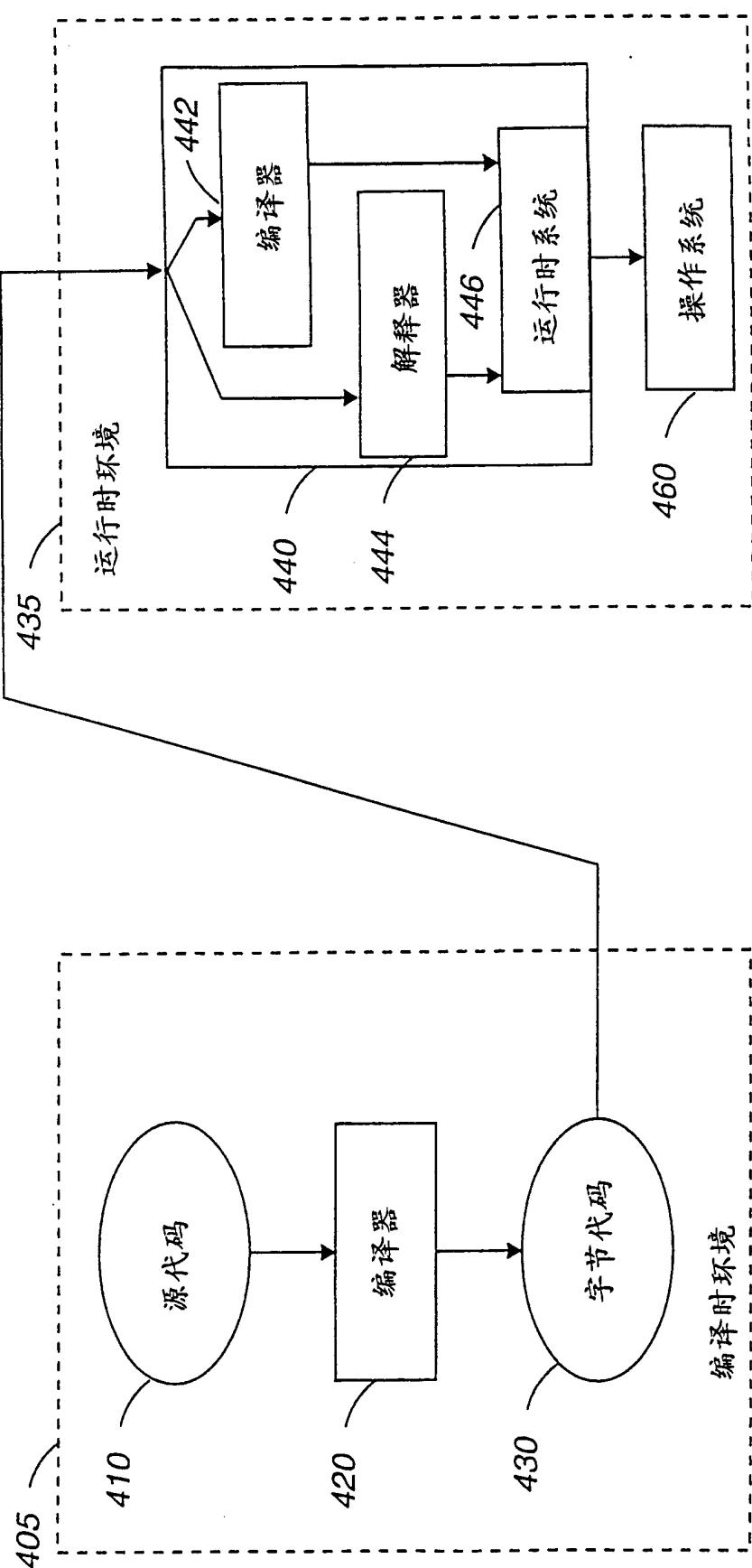


图 4