[19] 中华人民共和国国家知识产权局

[51] Int. Cl[7]
G06F 9/44

[12] 发明专利说明书

[21] ZL 专利号 00118813.5

[45] 授权公告日 2005 年 1 月 5 日

[11] 授权公告号 CN 1183447C

[22] 申请日 2000.4.26 [21] 申请号 00118813.5

[30] 优先权

[32] 1999. 4.26 [33] US [31] 09/299946

[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 G·布拉查 D·维斯瓦纳塔恩

审查员 解 欣

[74] 专利代理机构 中国专利代理(香港)有限公司

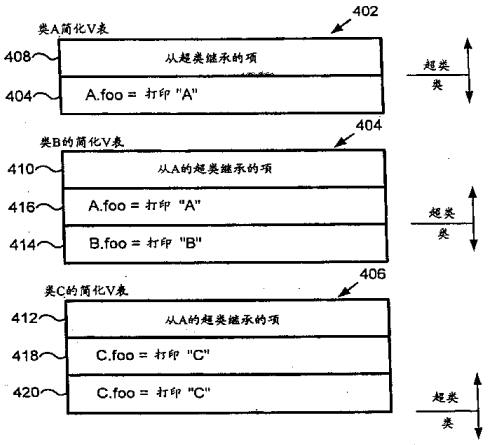
代理人 张志醒

权利要求书 2 页 说明书 21 页 附图 17 页

[54] 发明名称 用于发送表构造的方法

[57] 摘要

公开了用于构造发送表的装置及计算机程序产品。 在本发明的一个实施例中，配置一个新的发送表项的确定受一个类的可访问类型影响。 描述了发送表和发送表的构造方法，其中用于 V 表的项是这样确定的，即能避免可访问类型和类层次结构之间的冲突。 特别地，描述了在确定适当的覆盖语言的表构造技术时考虑到方法的可访问类型和包状态的发送表和发送表构造方法。 发送表可具有用于一个方法的多于一个的不同的项。



ISSN1008-4274

1. 一种为在从一个直接超类继承的一个第一类中的方法构造一个发送表的方法，所述方法包括:

从所述直接超类复制一个发送表;

判断在所述第一类中的一个所选方法是否也存在于所述第一类的一个祖先超类中;

当判断所选方法也存在于所述祖先超类中时，判断所选方法的所述祖先超类是否是可访问的;

当判断所选方法的所述祖先超类是不可访问类型时，在用于所述第一类中的所选方法的所述发送表中生成一个新的项; 和

当判断所述第一类中的所选方法是可访问类型时，将一个项重写。

2. 根据权利要求 1 所述的方法，其特征在于，它还包括从至少一个超接口继承一种方法。

3. 根据权利要求 2 所述的方法，其特征在于，所述生成的项是一种从一个超接口继承的、在其类或其任意的超类的定义中未完成的方法。

4. 根据权利要求 1 所述的方法，其特征在于，所述构造发送表的方法是由一种面向对象的语言实现的。

5. 根据权利要求 4 所述的方法，其特征在于，所述面向对象的语言是 JAVA。

6. 根据权利要求 1 所述的方法，其特征在于，所述项或所述新的项还包括一个索引。

7. 根据权利要求 1 所述的方法，其特征在于，所述类属于一个包。

8. 根据权利要求 1 所述的方法，其特征在于，所述方法的类型还包括公用可访问类型、专用可访问类型、被保护可访问类型和包专用可访问类型的其中之一。

9. 一种为在从一个上行层次结构中的一个直接超类继承的一个第一类中的方法构造一个发送表的方法，所述方法包括:

从所述直接超类复制一个发送表;

判断在所述第一类中的一个所选方法和一种可访问类型是否也存在于所述上行层次结构的一个超类中；

对于每一种可访问类型，当判断所述所选方法和可访问类型也存在于所述上行层次结构的所述超类中时，为所述方法和可访问类型加上所述发送表的一个索引；和

对于加到所述所选方法和可访问类型的每一个索引，在所述发送表中配置一个项。

10. 根据权利要求 9 所述的方法，其特征在于，它还包括在所述发送表中为加到所述所选方法和可访问类型的每一个项设置一个优先级。

11. 根据权利要求 10 所述的方法，其特征在于，将一个基本优先级加到与相应于所述第一类的可访问类型的与所述所选方法和可访问类型相对应的项上。

12. 根据权利要求 9 所述的方法，其特征在于，它还包括当将所述方法的所有可访问类型都加上一个索引之后，停止判断所述第一类中的所述所选方法和可访问类型是否也存在于所述上行层次结构中的一个超类中。

13. 根据权利要求 9 所述的方法，其特征在于，一种方法的可访问类型为公用可访问类型、专用可访问类型、被保护可访问类型或包专用可访问类型。

14. 根据权利要求 9 所述的方法，其特征在于，利用一个索引来对所述第一类中的所选方法和可访问类型是否存在于所述上行层次结构的一个超类中进行判断，所述索引从当未找到所选方法和可访问类型时的一个第一状态改变为当找到所选方法和可访问类型时的一个第二状态。

15. 根据权利要求 9 所述的方法，其特征在于，它还包括确定所述发送表的尺寸。

16. 根据权利要求 15 所述的方法，其特征在于，它还包括配置至少一个与每一个另加的项相对应的发送表的项。

17. 根据权利要求 16 所述的方法，其特征在于，它还包括填写所述发送表。

用于发送表构造的方法

技术领域

本发明一般地涉及计算机软件和可移植软件领域，特别地涉及构造发送表。

背景技术

为便于以下的讨论，首先简略回顾传统的面向对象的计算环境。在面向对象的计算环境当中，对象指的是变量和相关方法的软件包。就变量而论，对象保持其状态即对象知道什么；就方法而论，对象实现其性能即对象能做什么。信息是软件对象相互联系和互相通讯的工具。总起来说，可将变量和方法称作为成员。一个对象的变量和方法也可以称作为实例变量和实例方法。

一般地，类变量和类方法是指被定义属于一个类的那些变量和方法。一个类可以被看作是定义某一个种类的所有对象所共有的变量和方法的可复用方案。一个实例指的是属于一个类的一个对象，在所述类中存储单元被分配给类中的实例变量。

类可排列成类层次结构形式或继承树形式，其中一个类在层次结构中出现的位置越向下，该类便越特殊。图 1 是表示一个特定的类层次结构 100 的框图。类层次结构 100 包括含有变量 104 和方法 106 的一个类 A102。在说明性的实施例中，类 A 具有包括方法 M 在内的多个相关的方法。类层次结构还包括继承类 A 的类 B110，以及继承类 B110 的类 C，D 和 E(分别为 112，114 和 116)。每一子类也都包括变量 104 和方法 106。

一个超类指的是类的直接祖先及类的所有上行类。“直接”超类将类层次结构中最近的双亲超类从类层次结构中的另外的向上超类中区分出来。一个子类提供除了超类所提供的共有部分的基础之外的专门性能。为了做到这一点，每一子类都继承其超类的变量和方法。此外，子类还可以将变量和方法加入到其从超类继承的变量和方法中。例如，类 B110 继承了在其超类(类 A102)中定义的可访问方法，包括方法 M108。在所示实施例中，类 B 也定义了其自身的

新方法，包括方法 N109。重要的是应注意，子类也可覆盖一个或多个其继承的方法并为这些方法提供专门的工具。

当一种语言允许类继承一个单超类时，它具有单继承性。相反，当一种语言允许类继承多于一个超类时，则被认为具有多继承性。在图 1 所示的类层次结构中，由于每一子类仅只继承一个直接超类，所以具有单继承性。

在一些语言中，在继承实行方法中可重复使用一个超类。一般地，一个子类从其超类继承所有该子类可访问的、而该子类又无法简单覆盖的方法。可访问性由超类成员的可访问声明和子类成员的可访问声明联合确定。

将可访问性区别开来便可使整个范围从不受约束的源代码共享变成为特定的专用形式。两种传统的可访问级类型为公用类型和专用类型。公用成员可被其它类看到或访问。专用成员只能由其本身的类访问。特殊的语言还可以使用另外的可访问类型。例如，C++使用一种被保护的可访问类型，其中被保护成员一般地只能由其子类进行访问。此外，Java 使用一种包专用可访问类型，其中包专用成员只能由特定包内的类访问。一个包是指为该包所有组成部分提供访问保护和名字空间管理的相关的类和接口的集合。类可集合成包以便使类易于查找和使用，以避免命名冲突并可对访问进行控制。

通常，信息的可访问类型根据成员的声明来确定。如果最初未提供可访问类型，则可以使用缺省可访问类型。例如，在一个 Java 包中，最初未指定可访问类型的成员的缺省可访问类型为包专用。

图 2 是描述一个类层次结构的框图，其中层次结构中的每一类都在本地定义了一个方法“foo”。具体地，类 A202 属于一个包 P1 并且包含一个具有未确定可访问类型的方法“A.foo”204。在这种情形下，方法 A.foo204 相对于包 P1 缺省为一个包专用可访问类型。类似地，类 B206 属于一个包 P2 且包含一个具有公用可访问类型的方法“B.foo”208。类 C210 属于包 P1 且包含一个具有公用可访问类型的方法“C.foo”212。方法 204，206 和 208 都能实现方法“foo”，但三者之间的差别在于其相应的类、包、源代码和可访问类型。这种差别可影响方法的性能，例如，如果该方法调用打印语句的源代码来表明该方法的类，则如图所示三种方法将具有不同的输出。

按常规，每一类和接口都具有一个方法表，该方法表含有其在本地定义的所有方法。此外，每一类都具有一个发送表或 V 表，该表具有可被类调用的所

有外部可访问方法、包括继承方法在内的项。一般地，专用方法不包括在 V 表中。在常规的 V 表构造中，每一外部可访问方法与一个单一 V 表项相关联，该 V 表项指向与一个方法相对应的一组代码。因此，不管是本地方法还是继承方法，每一 V 表项都指向其所对应的方法。

图 3 的框图示出与图 2 的类 B206 相对应的 V 表 300 的常规格式。V 表 300 包括一个超类部分 302 和一个类部分 304，其中超类部分 302 含有与从直接超类(类 A202)中所继承的方法相对应的所有项，类部分 304 含有与在类 B206 本地所定义的方法相对应的项。类部分 304 中的本地定义的方法对于类 B206 来说是新方法。例如，在类部分 304 当中加入一个未对与类 A202 方法相对应的项进行重写的新项 306，结果增大了 V 表尺寸。或者，覆盖一个相应的超类方法的本地定义方法对超类部分 302 中已有的 V 表项进行重写。例如，指向 B.foo 代码的 V 表 300 的项 308 对指向本地定义方法 C.foo 代码的 V 表 300 的超类部分相应项进行了重写。

Java 编程语言试图采用单继承性。在 Java 中，一个子类被定义为继承其超类和祖先的所有可访问成员，并通过隐藏或覆盖这些可访问成员来利用这些成员作为其自身的成员。子类继承了那些声明为公用或被保护类型的超类成员。此外，只要子类与超类在同一包中，子类就可继承无可访问标识的超类成员(即它们都缺省为包专用)。因此，Java 可灵活地提供自不受限制的源代码共享到可控专用级别的可访问范围。基于相反的继承计划的常规 V 表构造不允许这种可访问范围。

鉴于上述，很明显需要改进的发送表构造技术。

发明内容

根据本发明，公开了用于构造发送表的方法、装置及计算机程序产品。在本发明的一个实施例中，判断新发送表项位置易感受到可访问类型的不同。在另一个实施例中，提供了一种 V 表，该 V 表中的方法具有多于一个的项。

根据本发明的一个实施例，本发明涉及一种用于为在继承一个单超类的一个类中的方法建立一个发送表的方法。该方法包括复制用于一个超类的一个发送表。该方法还包括判断该类中的所选的一个方法是否也存在于该类的一个祖先超类中。当判断所选方法也存在于第一个类所继承的一个祖先超类中时，该

方法还包括判断所选方法的一个所选超类形式是否是可访问的。当判断所选方法的所选超类形式是不可访问的时，该方法还包括在用于第一个类的所选方法的发送表中生成一个新的项。当判断第一个类中的所选方法是可访问的时，该方法还包括对一个项重写。当不存在该方法的超类形式时，则生成该方法的一个新的项。

在另一个实施例中，本发明涉及一种为在继承一个上行层次结构中的一个直接超类的类中的方法建立一个发送表的方法。该方法包括从一个超类复制一个发送表。该方法也包括判断该类中的所选方法和可访问类型是否也存在于该上行层次结构的一个超类中。该方法还包括指定该发送表的一个项的一个索引。

而在另一个实施例中，本发明涉及一种用于继承至少一个超类的一个类的发送表，该发送表包括用于特定方法的多个项。

附图说明

通过参考结合相关附图作出的下述描述，将更好地理解本发明，其中：

图 1 是描述一个类层次结构的典型格式的框图。

图 2 是描述包含有在各自的类中都分别定义了的一个特定的方法 foo 的类层次结构的框图。

图 3 示出与图 2 的类 B 相对应的传统的 V 表格式。

图 4 是与根据本发明的一个实施例的图 2 的类相对应的一组简化 V 表。

图 5 是与根据本发明的另一个实施例的图 2 的类 C 相对应的简化 V 表。

图 6 是与根据本发明的另一实施例的继承图 4 的类 C 的类 D 的简化 V 表。

图 7a 是一个框图/流程图，示出从包含有 Java 源代码的 Java™程序向在特定平台或计算机上运行的本地代码的转换。

图 7b 是由下述的图 18 的计算机系统所支持的虚拟机的图形表示。

图 8 示出根据本发明的一个实施例的发送表构造方法。

图 9 示出的框图描述在图 8 的步骤 804 所述的根据公用/被保护协议来设置方法 M 的方法。

图 10 示出的框图描述将适当的方法源代码指针插入到一项中的方法。

图 11 示出的框图描述在图 8 的步骤 808 所述的根据包专用协议来设置方

法M的方法。

图12示出的框图描述在图8的步骤810所述的根据专用协议来设置方法M的方法。

图13示出的框图描述在图8的步骤814所述的根据miranda方法协议在发送表中设置项的方法。

图14示出的框图描述一种优选的发送表构造方法，根据本发明的一个优选实施例，该方法能够减少搜索量。

图15示出的框图描述在图14的步骤1404所述的在整个上行层次结构中进行搜索时用来指定可访问索引并使表增大的方法。

图16示出的框图描述完成上行层次结构搜索之后在V表中生成索引的方法。

图17示出根据本发明的一个实施例的V表。

图18是适于实施本发明的一个实施例的典型的计算机系统的框图。

具体实施方式

下面将对本发明的一个特定实施例进行详细的说明。本实施例的一个例子示于附图中。尽管本发明是结合一个特定实施例而描述的，但可以理解，它并不试图将本发明限制在一个具体的实施例。相反，它可覆盖由后附的权利要求书所限定的包括在本发明的精神和范围之内的所有替换、修改和等效方案。

对发送表和发送表的构造方法进行描述，其中这样地确定用于V表的项，即避免可访问类型和类层次结构之间的冲突。特别地，对于在确定适当的覆盖语义和表结构中考虑到方法的可访问类型的发送表和发送表构造方法进行了描述。对于一种方法，发送表可以具有多于一个的不同的项。为了避免方法的不当处理，必须确定将被覆盖的项。在一些情形下，即使一种方法覆盖了一个超类方法，也仍必须生成一个新的V表项。

在所述的Java实施例中，提供了四种不同的可访问类型标识。相应地，对于每一中标识至少沿超类层次结构的一部分向上进行多次搜索。只有找到一个超类中的所有的可访问标识时，才停止沿上行层次结构向上进行的搜索。由于一个类也可继承一个超接口，因此类层次结构和可访问类型之间遇到的语义干扰也出现在接口层次结构中。

此外，一个V表中的用于一个特定方法的多个项中的每一项都可被赋予一个用于一个类中的该特定方法的优先级。一般地，包括与方法的可访问类型具有相同的可访问标识的项被指定为基本项。当声明特定的方法时，一般可调用该基本项。此外，在V表中也存在用于与不同的可访问标识相对应的特定方法的一个第二和第三项。

通常，一个新V表项位置的确定易受方法可访问类型的影响。它也可取决于一个被覆盖方法的可访问类型。例如，一个公用或被保护方法只有在无公用或被保护方法被覆盖时才需要一个新的V表项。否则，会采用被覆盖的公用/被保护方法的V表索引。类似地，一个包专用方法只有在没有同一包的包专用

决于一个被覆盖方法的可访问类型．例如，一个公用或被保护方法只有在无公用或被保护方法被覆盖时才需要一个新的V表项．否则，会采用被覆盖的公用/被保护方法的V表索引．类似地，一个包专用方法只有在没有同一包的包专用方法被覆盖时才需要一个新的V表项．一个专用项总是需要一个新的V表项。因此，为新的和被覆盖的方法更新V表项的方法与最多使用三个必须正确设置的V表项的方法相关．

参照图4对根据本发明的一个实施例的V表构造规则进行描述，图4所示的分别是图2的类A202、类B206和类C210的简化V表402、404和406．一般地，每一V表包括在每一V表中分别由部分408、410和412表示的多个继承项．如图2所示，为了说明的目的，假设类A、B和C本身都定义了一个方法“foo”．类A属于包P1，方法A.foo定义成包专用．类B属于包P2，方法B.foo为公用．类C属于包P1(如类A)，方法C.foo为公用。

在与类A、B和C相对应的V表构造中，必须判断本地定义的方法“foo”是否需要一个新的V表项或者是否需要对V表的超类部分的一个或多个相应的项重写．在方法“foo”的超类类型是否可为该类进行访问的基础上可简单地进行上述判断．与可访问方法类型相对应的任一V表项都被重写．如果没有与方法“foo”相对应的超类项与可与该类访问的方法相对应，则需生成一个新的V表项。

借助例子来考虑用于类B的V表构造．首先，对其超类(此例中为类A)的V表进行复制．由于类B在本地定义了一个方法B.foo，所以必须判断是重写用于A.foo的V表项还是形成一个新的V表项．如前所述，类A202的方法204对于包P1来说为包专用，因此无法对类B206的方法208进行访问，这是由于类B在包P2中(与包P1相对)．因此，在用于类B404的V表中提供一个新的与类B206的方法208相对应的项414．因而，用于类B的V表现在有两个与“foo”方法相对应的项．第一个项是与方法A.foo相对应的原来的项416.第二个项是与B.foo相对应的新的项414.也就是，将与B.foo相对应的新的项414加入到了V表404中．以这种安排，当类B的实例调用方法“foo”时，类B404实例的V表返回到与用于类B206的适当方法208相对应的B.foo项414.

接下来考虑类C的V表构造．如前所述，类C210继承了其直接超类即

类 B206 的所有方法. 因此, 类 C 的 V 表含有来自其直接超类(此例中为类 B404)V 表的、包括前面判断的项 414 和 416 在内的所有项. 在所示实例中, 类 C 为包 P1 的一部分(同类 A, 但非类 B), 且也在本地定义了一个“foo”方法(在此指的是 C. foo). 如前所述, 任何可为类 C 访问的已有方法“foo”类型都将被覆盖而任何不可访问的已有方法“foo”类型都不被覆盖. 在这种情形下, 类 B206 的方法 208 是公用的且可为类 C210 的方法 212 所访问. 因此, 在用于类 C 的 V 表 406 中, 与类 B206 的方法 208 相对应的项 420 将被重写为标记 C. foo. 此外, 类 A202 的方法 204 对于包 P1 来说为包专用且由于类 A 和类 C 在同一包(包 P1)中所以也能为类 C210 所访问. 因此, 在用于类 C 的 V 表 406 中, 与类 A202 的方法 204 相对应的项 418 也将被重写为标记 C. foo. 结果, 在同一 V 表中, C. foo 被标记两次. 以这种安排, 当方法“foo”被类 C 的实例调用时, 实例的 V 表将返回 C. foo 项.

下面参见图 5, 对第二个例子进行描述. 在此实例中, 除将类 C 定义为第三个包、包 P3(不是图 4 实例中的包 P1)的一部分这一特例以外, 全部保留图 2 的类层次结构和方法可访问类型. 在与类 C 相对应的 V 表构造中, 类 B210 的方法 208 仍可为类 C 所访问. 因此, 与方法 B. foo 相对应的 V 表项(同前一情形)将由与 C. foo 相对应的项 504 来重写. 而方法 A. foo 对于包 P1 来说为包专用, 因此不能为作为包 P3 的一部分的类 C 所访问. 因此, 在用于类 C 的 V 表 500 中不能对项 502 重写.

在面向对象的语言当中经常发现的另一个值得注意的规则是, 一旦建立了公用性以后便不能再降低方法的公用性, 即可访问类型无法削弱. 在此方式下, 子类可如其超类所期望地总可以被使用, 即所继承的公用方法不能转化成专用方法.

图 6 示出一个构造用于类 C210 的一个子类 D 的 V 表 600 的例子, 它是专用的且属于包 1. 用于子类 D 的 V 表 600 继承了其直接超类即图 2 的类 C210 的所有方法. 因此, 它含有其直接超类的所有项(在此情形下它是图 4 的用于类 C210 的 V 表 406). 可与前面所述的 V 表的构造类似地来完成用于类 D 的 V 表 600 的构造. 类 C 的方法 212 是公用的且可为类 D 所访问. 于是, 对项 606 重写使其成为方法 D. foo. 进而, 类 A202 的方法 204 对于包 P1 来说为包专用并且可为属于包 P1 的类 D 方法来进行访问. 此方法驻留在项 604

中并且类似地被重写。最后，在实例D本地所定义的方法“foo”(即D.foo)是专用的且生成一个新的项608来反映该项。因此对于同一方法“foo”目前在V表600当中有三项。下面将参考图8-13详细地对生成改进的V表600的方法进行描述。

与类和超类层次结构相类似，也存在一个独立的接口层次结构。通常，一个接口指的是一个用来互相联系互不相关的对象的机构。为了在一种面向对象的语言中做到这一点，一个接口包括一组方法标签。一般地，没有与方法标签相关的代码。可以实现一个接口的一个类可以实现在该接口中定义的所有方法并且从而实现特定的性能。

一个抽象类定义了同属性的性能且可以部分地实现这些方法，但是该类的大部分并未定义且不能实现。此未填写的标识允许单个的工具来完成用于特定的子类的细节(即代码)。一个抽象类可以包含一个用来规定其可被编译器识别的状态的声明。当实现一个接口而且类是抽象的时，编译器允许从接口继承未经规定的方法。然而，如果类不是一个抽象类的话，编译器将坚持要填写完未经规定的方法。这些从一个超接口继承的、在其类或其任意的超类的定义中未完成的方法被称作“miranda方法”。

这些miranda方法如在类中已被声明，则也需要一个项。由于一个类可以类似地从一个超接口继承而来，因此在类层次结构和可访问类型之间遇到的语义矛盾，也会出现在接口层次结构中。接口层次结构中的一个显著的不同点在于接口方法一般总是公用的。但当在单继承层次结构中的一个类只有一个超类时，它可以有多个超接口，这将大大地增加系统的复杂性。

为了在抽象方法被调用时能够对其进行处理，应当发出一个用于调用无相应的代码的方法的错误信息，用一个短指针来代替方法代码指针。此短指针指向表示特例的代码。由于抽象方法是通用的而且允许互不相关的设计者根据抽象类来设计代码，所以此短指针存在，以便控制相应的子类中的通用性能。

在一个特定的实施例中，miranda方法在类的V表中被定义，但却不出现在任何类的方法表中。为了建立一个V表，由于miranda方法不由类进行声明，所以必须对其分开处理。在本发明的一个实施例中，为一个miranda方法提供了一个单独的V表项。

采用可访问结构和源代码继承性的任何软件语言都可实现本发明。在本发明的一个特定的实施例中，可以用发送表构造中的 Java 语言来实现本发明。

图 7a 是一个框图，示出根据本发明的一个实施例的、由 Java 源代码生成本地指令所涉及的输入/输出和执行软件/系统。在其它实施例中，可以用于另一种语言的虚拟机或利用非 Java 类文件的类文件来实现本发明。自图左侧开始，第一输入是将由 Mountain View, Californiar 的 Sun Microsystem 开发的 Java™ 编程语言所编写的 Java 源代码 701 输入到一个字节代码编译器 703。字节代码编译器 703 实质上是一个将源代码 701 编译成字节代码的程序。字节代码包含在一个或多个 Java 类文件 705 中。Java 类文件是可移植的，可以在任何具有 Java 虚拟机(JVM)的计算机上执行。虚拟机的组成部分由图 7B 更详细地示出。Java 类文件 705 输入到 JVM707。JVM707 可以在任何计算机上使用，而不必限于具有字节代码编译器 703 的同一计算机。JVM707 具有多种用途，如翻译器或编译器。如果用作一个编译器，则还可起“实时”(JIT)编译器或起适配型编译器的作用。当用作一个翻译器时，JVM707 可以对 Java 类文件 705 中所包含的每一字节代码指令进行翻译。

图 7B 是可由下述图 18 的计算机系统 1800 支持的虚拟机 711 如 JVM707 的示意图。如上所述，当将一个计算机程序如用 Java™ 编程语言编写的程序由源代码转化成字节代码时，在编译时间环境 709 中将源代码 701 输入到字节代码编译器 703 中。字节代码编译器 703 将源代码 701 转化成字节代码 705。通常，是在软件开发器生成时间源代码 701 之后，再将源代码 701 转化成字节代码 705。

一般地，字节代码 705 能够被复制、下载或例如通过图 18 的网络接口 1824 分布到网络上、或者保存在如图 18 的基本存储器 1804 的一个存储装置中。在所述实施例中，字节代码 703 属独立平台。即，字节代码 703 基本上可以在能够运行适当虚拟机 711 的任何计算机系统上执行。通过编译字节代码而形成的本地指令可留存为 JVM 以后所用。由此可见，多次执行可使转换成本减少，使本地代码较之翻译代码具有速度优势。通过实例可以看出，在 Java™ 环境中，字节代码 705 可以在运行 JVM 的一个计算机系统上执行。

字节代码 705 被输送到包括虚拟机 711 在内的运行时间环境 713 中。运

行时间环境713通常采用如图18的CPU1002的处理器来执行。虚拟机711包括一个编译器715、一个翻译器717和一个运行时间系统719。字节代码705通常可输入到编译器715或翻译器717中。

当字节代码705输送到编译器715时，包含在字节代码705中的方法被编译成本地机器指令(未示出)。另一方面，当字节代码705输送到翻译器717中时，字节代码705被一次一个字节代码地读取到翻译器717中。接着当每一字节代码都读入到翻译器717中时，翻译器717进行由每一字节代码定义的操作。一般地，翻译器717对字节代码705进行处理，几乎连续地进行与字节代码705相关的运算。

当从操作系统721调用一个方法时，如果判断被调用方法为一个翻译方法，则运行时间系统719可从翻译器717中得到该方法。另一方面，如果判断被调用方法为一个编译方法，则运行时间系统719启动编译器715。编译器715由字节代码70生成本地机器指令，并执行该机器语言指令。通常，当虚拟机711停止时，机器语言指令也被放弃。在Tim Lindholm和Frank yellin的Java™虚拟机说明第二版(发行号0-201-43294-3)中对虚拟机特别是Java™虚拟机的工作情况进行了较为详细的描述，在此全面地将其引入作为参考。

图8-13示出根据本发明的一个实施例的发送表构造方法800。该发送表建立来用于具有一个直接超类S的一个包P中的类C。该发送表构造方法800同时计算出V表的大小，对V表进行配置，并将其填写到非易失性存储器的一个大小合适的部分中。

发送表构造方法800(图8)始于类C中的所有本地定义方法的反复进行。相应地，当前重复的方法称作方法M。为了区分建立发送表的可访问类型起见，将要明确的三组可访问类型分别为公用/被保护类型、专用类型和包专用类型。在图8所示情形下，公用和被保护类型由于在建立发送表时他们的性能基本上相同而被划分在一起。

发送表构造方法800始于判断方法M是否是公用或被保护的(802)。如果是公用或被保护的，则方法继续进行，根据下文据图9更为详述的公用/被保护协议(804)来配置方法M。如果不是公用或被保护的，则该方法判断方法M是否是包专用(806)。如果是包专用，则方法继续进行，根据下文据

图 11 更为详述的包专用协议(808)来配置方法M. 如果不是包专用，则方法M缺省为专用可访问类型，并根据下文据图 12 更为详述的专用协议(810)来进行配置. 在上述任一情形下，当根据适当的方式对方法M进行配置时，方法都要判断是否有未经处理的方法(812)并返回到下一方法的开始处. 当所有的方法都处理完后，必要的话，再进行一个额外步骤来处理下文当中据图 13 更为详述的任何miranda方法.

图 9 示出一种用于根据公用/被保护协议在发送表中配置方法 M 的方法(804). 该方法始于判断直接超类 S 或直接超类 S 的任何一个超类或超接口中是否有公用或被保护类型的方法 M(902). 换言之，该方法需判断上行层次结构中是否存在一种类型的方法M. 如果上行层次结构中不存在公用或被保护类型的方法 M，则将一个新的项加入到发送表中(904). 相应地，该方法将方法M的适当指针(1000)插入下文据图10更为详述的新形成的项(906)中. 如果超类中存在公用或被保护类型的方法M，则通过在类C发送表被识别出的超类项上插入适当指针(1000)来对识别出的超类项进行覆盖.

方法 804 还要判断上行层次结构中是否存在可访问的包专用(方法 M 所对应的包 P)型方法 M(910). 如果存在，则在识别出的超类项中为可访问的包专用方法插入适当指针，从而对识别出的超类项进行覆盖.

图 10 所示的框图描述一种将适当的指针插入用于在一个项中的一个方法的源代码上的方法(1000). 该方法始于判断方法 M 是否是一个抽象方法(1002). 如果该方法不是一个抽象方法，则将方法代码指针插入到发送表的适当项上(1004). 或者，如果该方法是一个抽象方法，则将一个短指针插入到发送表的适当项上(1006). 重要的是应注意，在此所插入的指针不论是普通指针还是短指针，都与已有的项指针无关，即一个短指针能够重写一个方法代码指针.

图 11 示出一种根据上文中据图 8 所述的包专用协议来配置方法的方法(808). 该方法始于判断在直接超类 S 或超类 S 的任何超类或超接口中是否存在可访问的包专用型方法M(1102). 如果上行层次结构中不存在具有适当包专用状态(此处为包 P)的方法 M，则将一个新的项加入到发送表中(1104). 相应地，该方法将包专用型方法M的适当指针(1000)插入到新形成的项中(1106). 如果上行层次结构中存在包专用型方法M，则在识别出的超

类项中插入适当的指针以对识别出的超类项进行覆盖.

方法808还要判断在直接超类S或超类S的任何超类或超接口中是否有公用或被保护型的方法M(1110). 如果有, 则在识别出的超类项中的识别出的公用或被保护型方法 M 上插上适当指针, 以对识别出的超类项进行覆盖(1112).

图 12 示出一种据上述图 8 所述的专用协议(810)来配置方法 M 的方法. 由于一个专用项一般需要一个新的 V 表项, 所以该方法始于向发送表中加入一个新的项. 相应地, 该方法将专用型方法M的适当指针插入到新形成的项中(1204). 在这种情形下, 不必检查是否是一个抽象类, 因为专用方法M不可能是抽象的.

方法810也要判断在直接超类S或超类S的任何超类或超接口中是否存在公用或被保护型方法M(1206). 如果存在, 则在识别出的超类项中插入适当的指针(1000)以便对识别出的超类项进行覆盖(1208).

方法810还要判断在直接超类S或超类S的任何超类或超接口中是否存在包专用型方法M(1210). 如果存在, 则在识别出的超类项中插入适当的指针(1000)以便对识别出的超类项进行覆盖.

当在发送表构造方法 800 中配置完成了类 C 中所有本地定义方法之后, 必须对任何 miranda 方法进行处理. 图 13 示出一种根据上述图 8 所述的 miranda 方法协议(814)来配置发送表中的项的方法.

Mirand 方法协议(814)始于判断是否存在未处理的类C超接口(1302). 如果不存在未处理的类C超接口, 则方法814结束. 如果存在未处理的超接口, 则选择下一个接口 I(1304). 对于此接口 I, 方法 814 反复进行接口 I 的每一方法(1306). 如果接口 I 无未处理的方法, 则方法返回判断是否存在未处理的类C超接口(1302). 否则, 回到接口 I 的下一个方法并进行检查, 判断在类C的发送表中是否已存在接口 I 的同类方法(1310). 如果发送表中已存在接口 I 的同类方法, 则方法继续反复进行接口 I 的其余方法(1306). 如果不存在类C的同类方法, 则为类C的发送表生成一新项(1312). 将指针插入到发送表的新项上(1314), 方法继续反复进行接口 I 的其它方法(1306).

为了构造发送表, 需要实施各种程序. 例如, 对传统发送表构造方法的

每一步稍作修改之后，可以利用传统的发送表构造方法。在传统方法当中，首先应判断发送表超类的尺寸。由此继承的尺寸，为类C中的每一本地定义方法提供一个新的项。累计新的项并加到到继承的超类发送表尺寸上。最后，再处理所有miranda方法从而完成发送表的构造。

由于本发明允许有多个可访问类型标识，对于每一种可访问类型都要沿至少超类或超接口层次结构的一部分向上进行单独检索。最好，当在一超类中找到特定可访问类型时，对此特定可访问类型的沿上行层次结构向上进行的检索便可停止。

此外，可对特定方法多项中的每一个项赋予一个优先级。一般地，与方法可访问类型相同的项为基本项。由于与此基本优先级相对应的可访问类型根据方法和包的不同而变化，所以优先级索引允许建立与包无关的基本方法。如果在上行层次结构中发现还有与其它可访问类型相对应的第二和第三项，则也可赋予到相应的项上。

实际上，当调用方法时只利用基本索引。因此，与第二和第三项相对应的索引不必配置或永久保留在发送表中，可根据实现方法中的需要来进行配置或长期保留。

为一个方法的多个项建立相对优先级的合理性在于，当链接基本方法时，由于被调用方法的可访问类型是基于项结构建立的，所以被调用方法的可访问状态将不模糊。

优先级使可访问类型相同的项可为一个类所用，同时对保持其它可访问类型项的需求很敏感。因此，类所调用的非基本索引本地新定义的项在类中相对于其它的层次结构进行适当刷新。本地定义的专用方法通常总是要求在V表中有一个新的项，可将该项设置成该方法的基本项。而自超类中继承的其它可访问方法(即公用方法)可被覆盖并正确索引以避免同一方法中两个项发生混淆。

上述方法适用于据不同的可访问级别为一个方法配置多个项。然而，发送表构造方法 800 要求对每一可访问类型都在相对上行层次结构中进行冗余查询。

图14至16示出根据本发明的另一个实施例的、减少查寻量的另一种发送表构造方法 1400。该方法同时也要判断发送表的尺寸和与 V 表中每一方

法相关联的索引，完成这一步骤之后，该方法还要对V表进行配置并填入适当的项．同样，该发送表建立来用于具有一个直接超类S的一个包P中的类C．发送表构造方法1400不同于前一例子，这是因为它是在类和接口层次结构交叉处为每一可访问类型方法M判断一个索引并且同时判断V表的尺寸．

发送表构造方法1400(图14)始于估计V表的初始尺寸，最初的近似值为超类的V表的尺寸(1402)．发送表构造方法1400还对用于判断在上行层次结构中是否找到一个特定的可访问类型的至少一个索引进行初始化．

与前面利用一个索引来判断在上行层次结构中是否找到方法M的情形相对，发送表构造方法1400利用多个索引．例如，当扫描上行层次结构时，特别地与公用可访问类型方法相关的可访问类型索引得以保持，而和特别地与包专用可访问类型相关的可访问索引无关．此外，任何可访问类型索引可始于一个缺省值(例如一个负数)，该缺省值表明在上行层次结构中并未找到一个特定的方法．该索引可改变为一个第二值(一个正数)，以表示相应方法的位置．

图14中论及的在对上行层次结构进行查寻时配置可访问索引并增大表尺寸(1404)如图15所示．该方法对类C中的每一个本地定义方法M反复进行处理(1502)．如果所有本地定义方法都已处理完毕，则方法返回到先前的发送表构造方法1400(1408)．对下一个未处理方法M，沿上行层次结构向上进行横向查找以寻找相应的方法M．这一方法始于判断是否存在直接超类S(1504)．

如果存在直接超类S，则发送表构造方法1400检查可访问类型索引的状态以判断是否已找到方法M所有可能的相关可访问类型．例如，如果与公用/被保护及包专用型方法相对应的所有可访问索引表明在上行层次结构中存在该方法(1512)，则方法继续进行为检索到的信息赋值．如果与公用、专用及包专用型方法相对应的所有可访问索引都未找到，则方法1400继续上溯到层次结构中(1504)．

接下来，发送表构造方法1400判断当前超类S是否包含被方法M覆盖的方法(1514-1520)．如果不存在这样一种方法，则结束当前超类S的方法1400并返回判断层次结构中是否有另一个超类(1504)．如果包含被方法M覆盖的方法，则需判断该方法的可访问类型．如果被覆盖方法是公用或被保

护类型(1516)，则用于该方法的基本 V 表索引将被配置为公用/被保护索引(1518)。相反，如果被覆盖方法是专用类型(1520)，则用于该方法的基本 V 表索引将被配置成包专用索引(1522)。在任何情形下，方法 1400 都继续上溯到层次结构中(1504)。

当找到了方法 M 的所有可访问类型或整个上行层次结构都查找完毕时，将该信息通知 V 表(1506)。上行层次结构查寻完之后生成 V 表 1700 中的索引的方法如图 16 和 17 所示。用于 V 表 1700 的索引的生成受方法 M 的可访问类型影响。相应地，方法 M 加索引方法应首先判断出方法 M 的可访问类型。

在此实施例中，首先判断方法 M 是否是公用或被保护型(1602)。如果方法 M 是公用或被保护型，则检查可访问索引以判断在上行层次结构中是否找到方法公用型 M(1604)。如果在上行层次结构中找到公用或被保护型方法 M，则加到方法 M 上的基本 V 表索引将是在超类中找到的公用/被保护可访问索引 1702(1606)。如果在上行层次结构中未找到公用或被保护型方法 M，则在 V 表 1700 中生成一个新的项 1704，用于方法 M 的基本索引加到新的项 1704 上(1608)，且利用 V 表指针 1706 使得当前 V 表的尺寸增大(1610)。结果不管在上行层次结构中是否找到方法 M，都在 V 表 1700 中加入了基本 V 表索引。

在 V 表 1700 中，V 表指针 1706 指向下一个项 1710。V 表指针 1706 始于与继承的超类 S 方法相对应的估算尺寸和位置处。由于类 C 本地定义的方法都加入一个新的项，因此 V 表指针 1706 上移并保持 V 表 1700 的尺寸不变。

接下来，如果已经记录有包专用可访问索引，则必须配置第二索引。所配置的索引之后被用于对 V 表 1700 超类部分的适当的项进行重写。

如果方法 M 属于包专用(1616)，则检查可访问索引以判断在上行层次结构中是否已找到方法 M(1618)。如果在上行层次结构中已找到包专用方法 M，则加到方法 M 上的基本 V 表索引将是在超类中找到的包专用可访问索引 1708(1620)。如果在上行层将结构中未找到包专用方法 M，则在 V 表 1700 中仍生成用于方法 M 的新的项 1704，并将方法 M 的基本索引加到新的项 1704 上(1622)，利用 V 表指针 1706 使当前 V 表尺寸增大。

再者，如果已记录有公用/被保护索引，则必须将其加到第二索引上。所加索引之后用于重写 V 表 1700 超类部分中的适当的项。

最后，对于前一方法1400，如果方法M不是公用、被保护或包专用类型，则它必是专用类型(1616)。由于本地定义的专用方法总是要求一个新项，所以在V表1700中生成用于方法M1704的新项，将方法M的基本索引加到该新项1704上(1628)，且利用V表指针1706使当前V表的尺寸增大。

接下来，如果已经记录了公用/被保护可访问索引，则必须将其加到第二索引上。接着用所加索引对V表1700超类部分的适当的项重写。类似地，如果已经记录了包专用索引，则必须将其加到第三索引上，所加索引用于对V表1700超类部分中的适当的项重写。

当将所有本地定义方法都加上索引并且判断出表的尺寸之后(1404)，如果必要的话，前述方法1400将对所有miranda方法进行处理(1406)。将V表1700分配到存储器的适当部位处(1408)。V表1700首先填入超类S的内容(1410)，然后填入每一本地定义方法(1412)。

对于每一本地定义方法M，前述方法1400将指针分配给位于基本索引处的相应方法源代码(1414)。如果已经加有第二索引(1416)，则在第二索引处设置源代码指针。同样地，如果已经加有第三索引(1420)，则在第三索引处设置源代码指针(1422)。

当一个子类实施一个miranda方法时，必须更新相应的V表项。因此，必须单独地判断是否有任何已被重写的miranda方法。这可通过查找超类的V表来完成。更具体地，从底部开始对V表进行扫描，寻找与当前方法名字和描述符相一致的miranda方法。与一个miranda方法相对应的一个项1712示于V表1700中。

为新的miranda方法配置项的方法与图13类似。在此实例中，不是将一个新的项加到发送表并将一个短指针插入到该新的项(1314)，而是将一个新的方法加入到本地定义的miranda组上。为此方法加上一个基本索引，该V表的尺寸增大。

在一个特定的实施例中，上述V表构造方法可以在SUN微系统的Hotspot内部类结构的类结构当中实现。在此情形下，每一个类和接口都有一个包含其所有本地定义方法的方法表。此外，每一个类都有一个发送表，该发送表包含有可在类实例上进行调用的所有方法(包括继承的方法)的项。

本发明可采用各种应用保存在计算机系统中的信息的计算机操作方

法。这些操作方法包括但并不限于那些要求对物理量的物理控制。通常，尽管不是必须的，但这些量应尽可能采取可被存储、传送、组合、比较以及其它被操作控形式的电或磁信号形式。本文所述的构成本发明的一部分的上述操作方法是非常有效的机器操作。所执行的控制在术语上通常是指例如生成、识别、运行、判断、比较、执行、下载或检测。主要是为了通用的原因，有时将这些电或磁信号方便地看作位、数值、元素、变量、字符或者类似等。然而，应当记住，所有这些以及类似的术语都与适当的物理量相关，并且仅仅是用于这些物理量的方便的符号。

本发明还涉及用于执行上述操作的一种器件、系统或装置。该系统为所需目的特别构造而成，或者可以是一个由存储在计算机中的计算机程序有选择地启动或配置的通用型计算机。上面给出的方法本身不与任何特定的计算机或其它计算装置相关。特别地，各种通用计算机都可用于根据本文的教导所写出的程序，或者另外可以更方便地构造一个更为具体的计算机系统来完成所要求的操作方法。

图 18 是根据本发明的一个实施例的适于实现上述处理方法的通用计算机系统 1800 的框图。例如，JVM707、虚拟机 711 或字节代码编译器 703 可在通用计算机系统 1800 上运行。图 18 示出一种通用计算机系统的一个实施例。其它的计算机系统结构和配置也可用于执行本发明的处理方法。由下面所述的各种子系统构成的计算机系统 1800 包括至少一个微处理器子系统(也称为一个中央处理单元或 CPU)1802。也就是，CPU1802 可由一个单片处理器或多个处理器来实现。CPU1802 是一个用来控制计算机系统 1800 运行的通用型数字处理器。利用从存储器中取出的指令，CPU1802 可以控制输入信息的接收和操作以及输出设备上信息的输出和显示。

CPU1802 通过存储总线 1808 与一般地是一个随机存取存储器(RAM)的第一基本存储器 1804 双向连接，与一般地是一个只读存储器(ROM)的第二基本存储器 1806 单向连接。在本领域内众所周知，基本存储器 1804 可用作一个通用存储区并可用作高速暂存存储器，而且也可以用来存储输入数据和已处理数据。它除了用于保存 CPU1802 上方法操作的其它数据和指令之外，还能保存编程指令和数据，并且一般地可用于通过存储总线 1808 进行数据和指令的双向快速传输。而且，在本领域内也是众所周知的，基本存储器 1806

一般地包括CPU1802所用的基本操作指令、程序代码、数据和对象以实现其功能。基本存储器1804和1806如下所述根据数据存取需双向还是单向进行而可以包括任何适用的计算机可读存储介质。CPU1802也能够频繁地直接并且迅速地从一个超高速缓冲存储器1810中取出和保存所需数据。

一个可移动的大容量存储装置1812为计算机系统1800提供了额外的数据存储容量，并且是通过一个外围总线1814双向或单向地连接到CPU1802上的。例如，通常被称作CD-ROM的一个特殊的可移动大容量存储装置将数据单向地传输到CPU1802上，而一个软盘可与CPU1802进行双向数据传输。存储装置1812还可以包括其它的计算机可读介质如磁带、瞬时存储器，载波形式信号、智能卡、便携式大容量存储装置以及其它存储装置。一个固定的大容量存储器1816也能提供额外的数据存储容量，它通过外围总线1814与CPU1802双向地连接。总体而言，这些介质的存取速度慢于基本存储器1804和1806的存取速度。大容量存储器1812和1816通常保存额外的编程指令、数据、以及CPU1802所不常用的数据。可以知道，如果需要的话，大容量存储器1812和1816中保存的信息也可以标准形式加到用作虚拟内存的基本存储器1804一部分上。

外围总线1814除了使CPU1802对存储器子系统进行存取操作之外，还可通过其对其它子系统和设备进行存取操作。在所述实施例中，这些子系统和设备包括一个显示监视器1818和适配器1820、一个打印机装置1822、一个网络接口1824、一个辅助输入/输出设备接口1826、一个声卡1828和扬声器1830，以及其它所需的子系统。

网络接口1824利用一种所称的网络连接将CPU1802连接到另外的计算机、计算机网络、或电信网络。通过网络接口1824，CPU1802可以从另一个网络上的计算机中接收信息，例如对象、程序指令、或字节代码指令，或者在进行上述方法的步骤中将信息输出到另一个网络上的计算机。可在CPU上执行的、通常表示成指令序列的信息，例如可以载波的计算机数据信号的形式从和向另一个网络接收和输出。可用接口卡或类似的设备以及由CPU1802完成的适当的软件以将计算机系统1800连接到一个外部网络并根据标准协议传送数据。也就是，本发明的方法实施例可以单独地在CPU1802上执行，或者可通过网络如因特网、内部网或局域网来与共享部分处理程序的远程

CPU 相结合进行. 附加的大容量存储器(未示出)也可通过网络接口 1824 连接到 CPU1802 上.

辅助 I/O 设备接口 1826 表示通用和专用接口, 它可使 CPU1802 发送数据、更一般地是从其它设备接收数据. 同时通过一个局域总线 1834 连接到 CPU1802 上的是键盘控制器 1832, 用于从一个键盘 1836 或一个指针设备 1838 接收输入, 并且将来自键盘 1836 或指针设备 1838 的译码符发送给 CPU1802. 指针设备可以是一个鼠标, 触针, 跟踪球, 或小平板, 对于与图形用户接口相连接是有用的.

此外, 本发明的实施例还涉及具有计算机可读介质的计算机存储产品, 该介质包含用于执行各种计算机实施的操作的程序代码. 计算机可读介质是指能够存储计算机系统读取的数据的任何的数据存储装置. 计算机可读介质的例子包括但不限于所有上述的介质, 包括硬盘、软盘、和特殊配置的硬件装置如专用集成电路(ASICs)或可编程逻辑器(PLDs). 计算机可读介质也可以是以载波形式作为数据信号分布在连接有计算机系统的网上, 使得计算机可读代码以分布形式保存和执行.

本领域的技术人员可以知道, 上述硬件和软件部件都是标准的设计和结构. 适于本发明使用的其它计算机系统可以包括另外的或更少的子系统. 此外, 存储总线 1008、外围总线 1814、以及局部总线 1834 都是用于连接子系统的任何互连方式的例子. 例如, 局部总线可用于将 CPU 连接到固定的大容量存储器 1816 和显示适配器 1820. 图 18 中的计算机系统只是适于采用本发明的计算机系统的一个实例. 也可采用具有不同的子系统配置的计算机结构.

虽然为了理解清楚起见详细地描述了本发明, 但应清楚在后附的权利要求的范围之内可以有一定的修改和变形. 例如, 尽管本发明是对于构造一个类的发送表来讨论的, 但类似地可根据所公开的步骤来构造一个接口发送表. 在另一实例中, 对 V 表进行区分的静态方法和方法未得以扩展, 但本发明适用于对其进行区分. 此外, 作为一个优化实例, 只要 V 表项能够覆盖超类或超接口中的一个方法, 便能够避免为最后一个方法定义 V 表项. 而且, 应当注意, 尽管本发明只给出了四种可访问类型, 但很明显更多的可访问类型也是可能的并可适用于本发明. 再有, 还应注意, 还有本发明能够灵活适

应的、未指明地址的其它发送表构造矛盾和细节．因此，本实施例仅用作示意性的而非限制性的，且本发明并不限于本说明书中所给出的细节，可在后附的权利要求的范围和等效结构中进行修改。

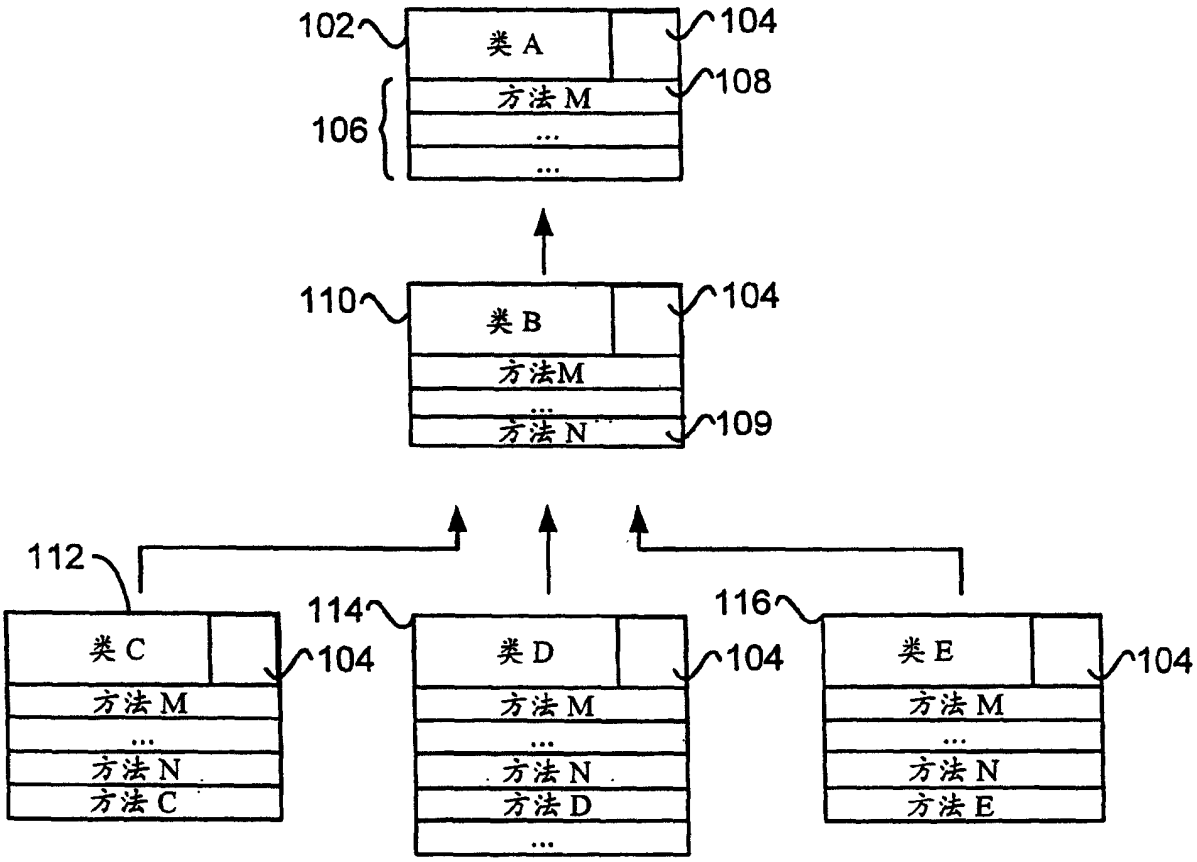

102
类 A
104
108
106
方法 M
...
...
110
类 B
104
方法M
...
方法 N
109
112
类 C
104
方法 M
...
方法 N
方法 C
114
类 D
104
方法 M
...
方法 N
方法 D
...
116
类 E
104
方法 M
...
方法 N
方法 E

图 1

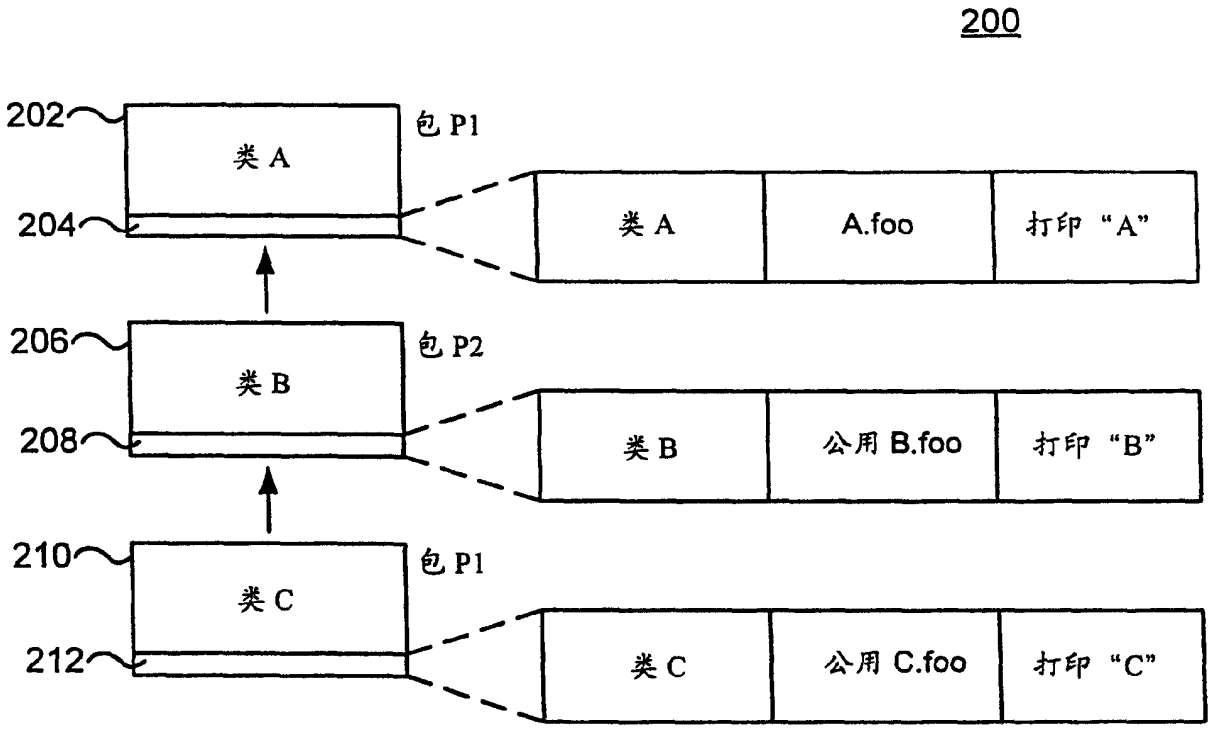


图 2
(现有技术)

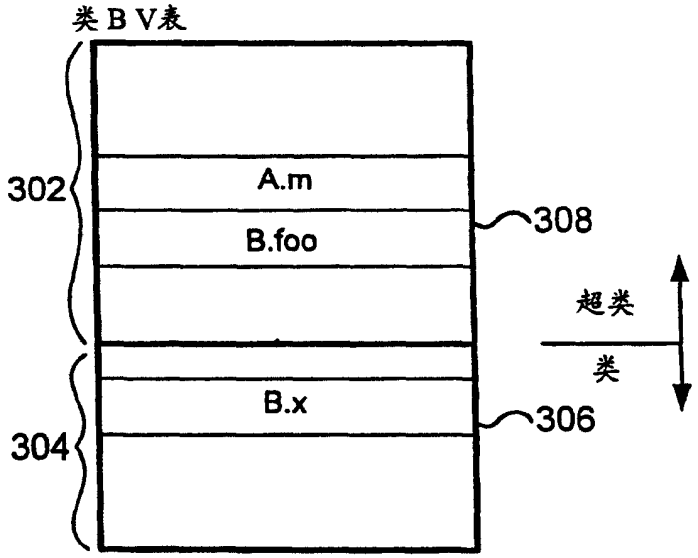


图 3
(现有技术)

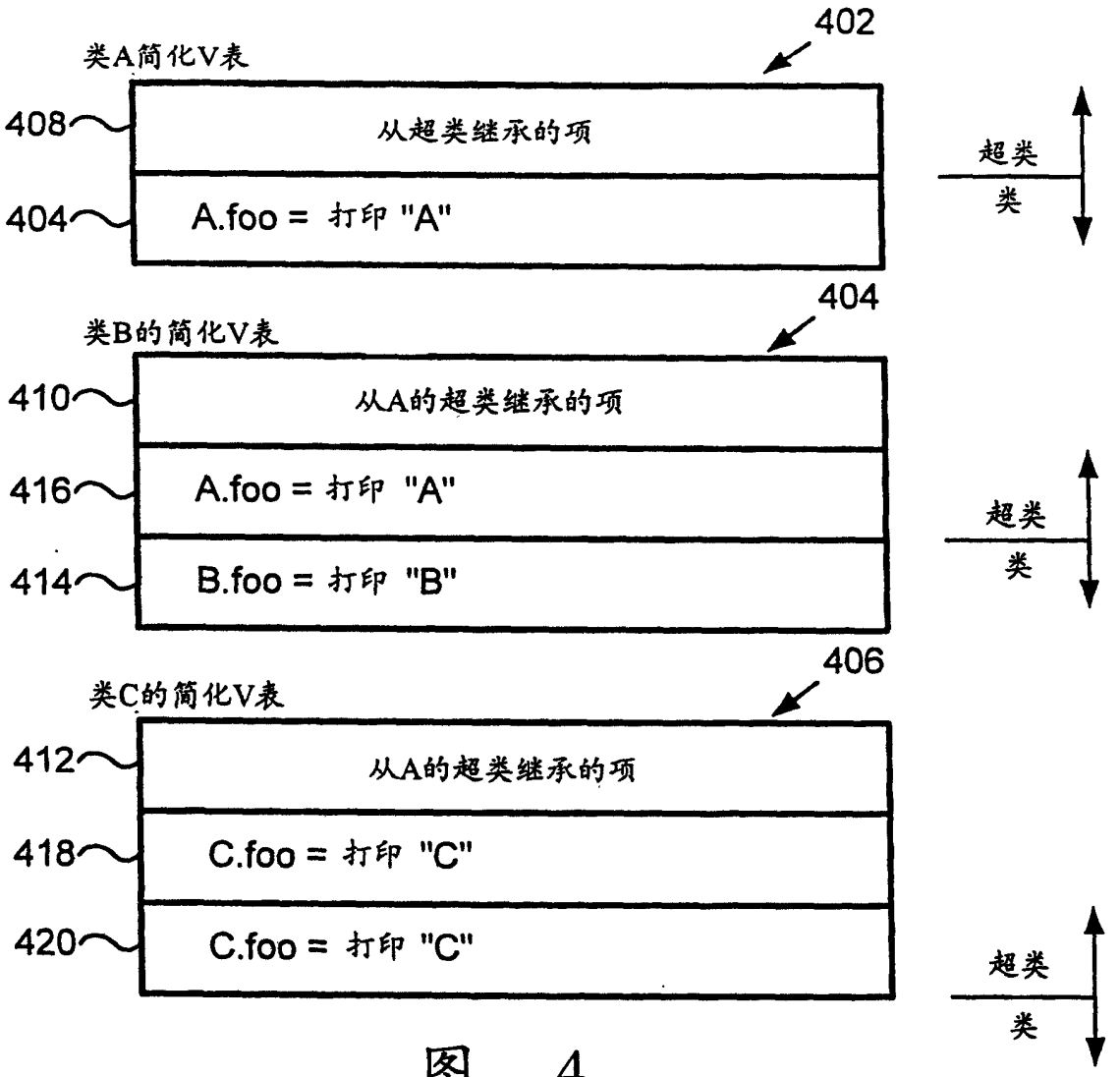

402
类A简化V表
408
从超类继承的项
404
A.foo = 打印 "A"
超类
类
404
类B的简化V表
410
从A的超类继承的项
416
A.foo = 打印 "A"
414
B.foo = 打印 "B"
超类
类
406
类C的简化V表
412
从A的超类继承的项
418
C.foo = 打印 "C"
420
C.foo = 打印 "C"
超类
类

图 4

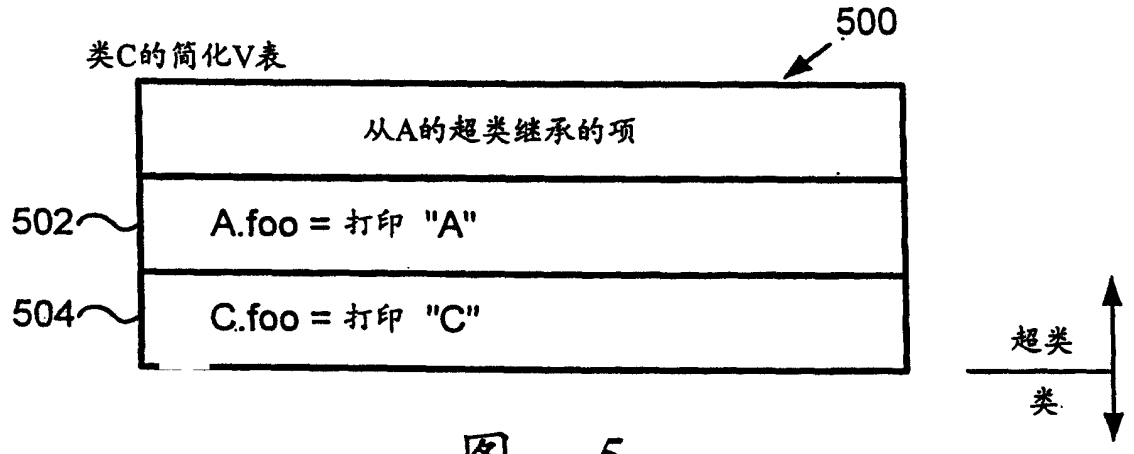

500
类C的简化V表
从A的超类继承的项
502
A.foo = 打印 "A"
504
C.foo = 打印 "C"
超类
类

图 5

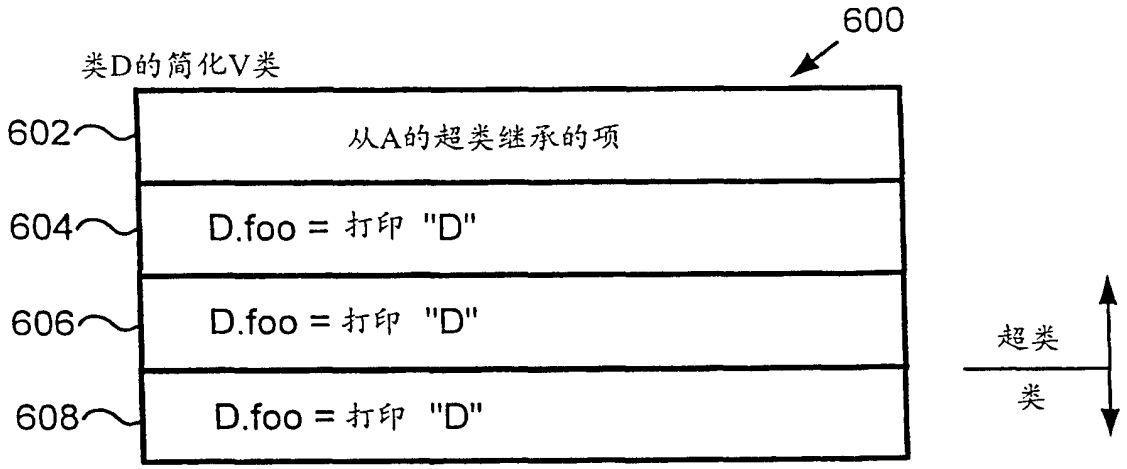

600
类D的简化V类
602
从A的超类继承的项
604
D.foo = 打印 "D"
606
D.foo = 打印 "D"
608
D.foo = 打印 "D"
超类
类

图 6

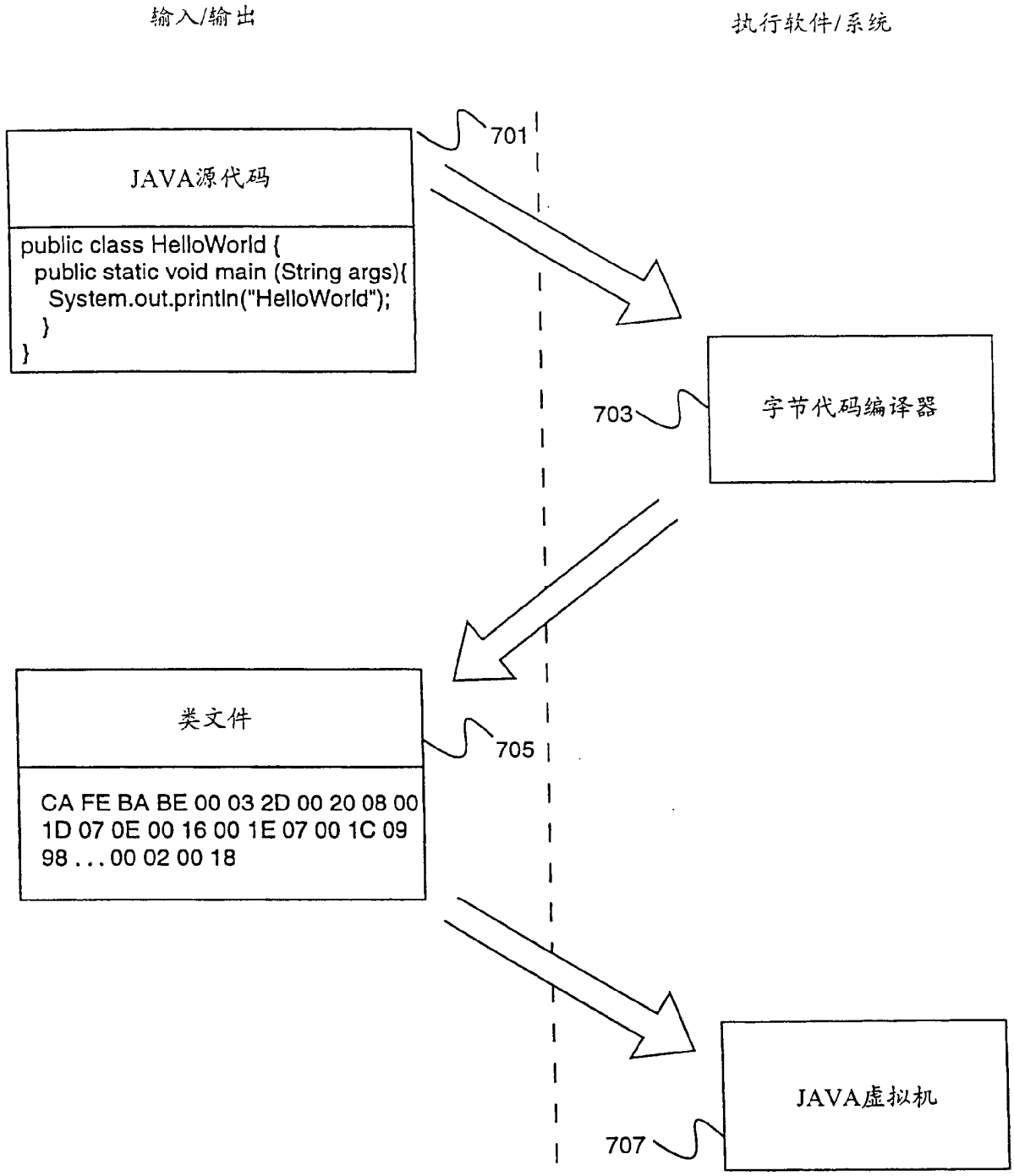

输入/输出
执行软件/系统
JAVA源代码
public class HelloWorld {
public static void main (String args){
System.out.println("HelloWorld");
}
}
701
703
字节代码编译器
类文件
CA FE BA BE 00 03 2D 00 20 08 00
1D 07 0E 00 16 00 1E 07 00 1C 09
98 . . . 00 02 00 18
705
JAVA虚拟机
707

图 7a

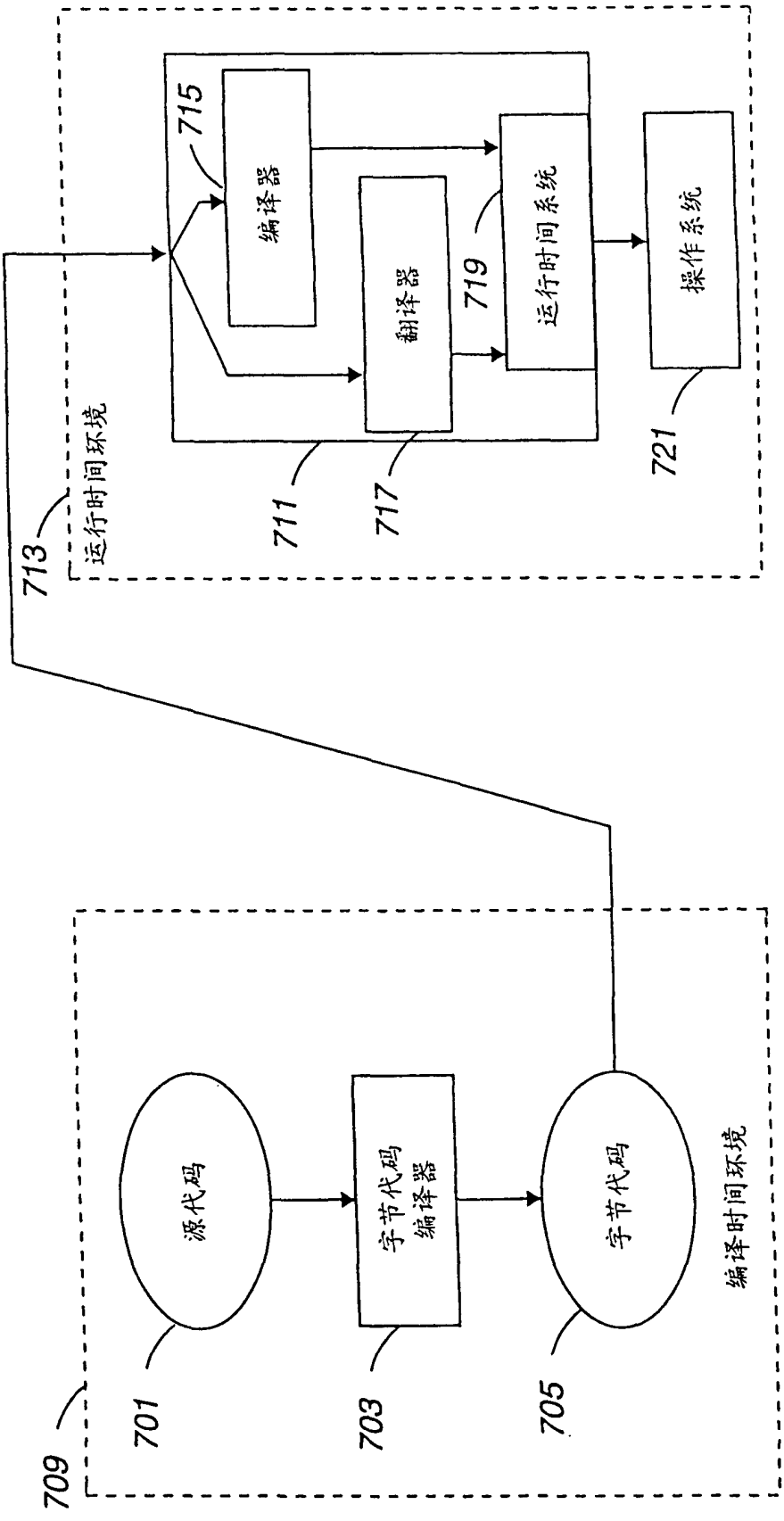

709
编译时间环境
源代码
701
字节代码编译器
703
字节代码
705
713
运行时间环境
711
编译器
715
翻译器
717
运行时间系统
719
操作系统
721

图 7b

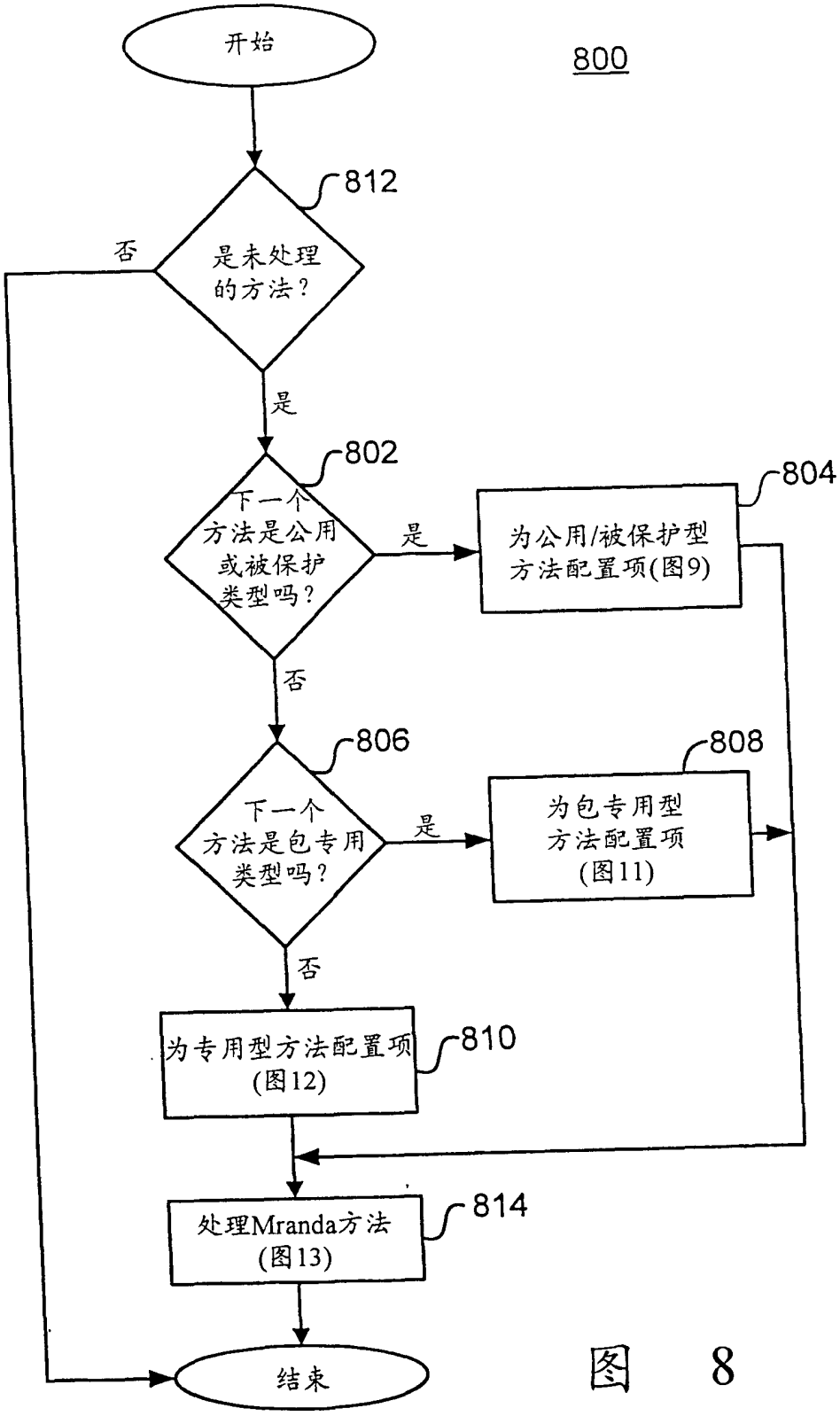

开始
800
812
是未处理的方法？
否
是
802
下一个方法是公用或被保护类型吗？
是
804
为公用/被保护型方法配置项(图9)
否
806
下一个方法是包专用类型吗？
是
808
为包专用型方法配置项(图11)
否
为专用型方法配置项(图12)
810
处理Mranda方法(图13)
814
结束

图 8

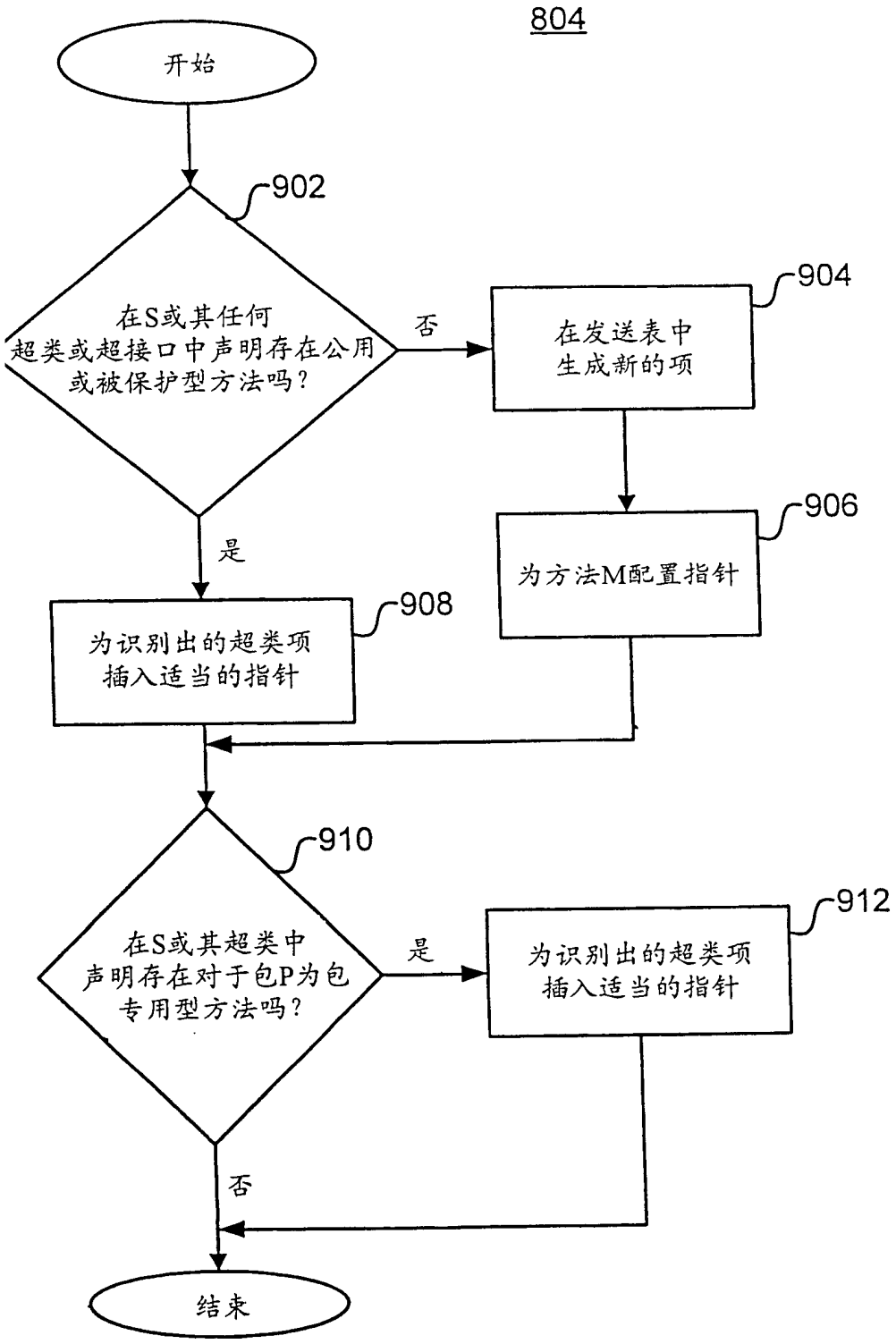

804
开始
902
在S或其任何
超类或超接口中声明存在公用
或被保护型方法吗?
否
904
在发送表中
生成新的项
906
为方法M配置指针
是
908
为识别出的超类项
插入适当的指针
910
在S或其超类中
声明存在对于包P为包
专用型方法吗?
是
912
为识别出的超类项
插入适当的指针
否
结束

图 9

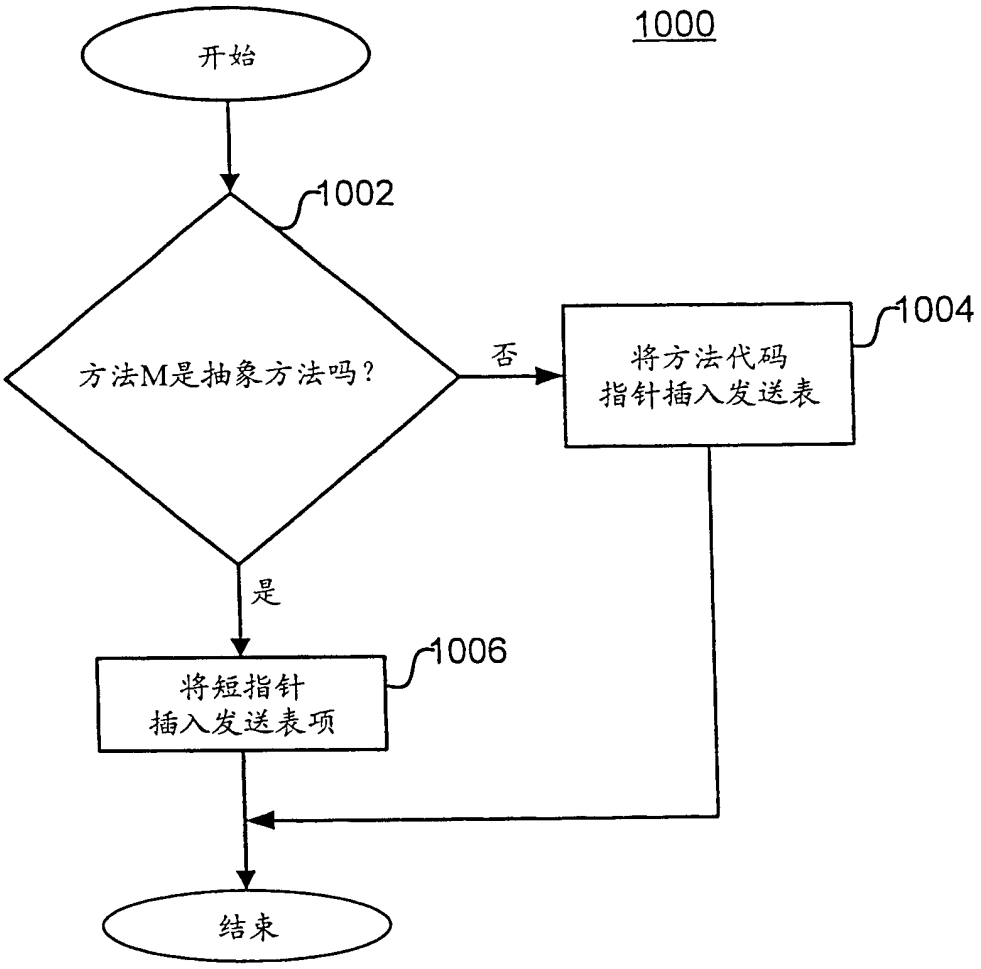

1000
开始
1002
方法M是抽象方法吗？
否
1004
将方法代码
指针插入发送表
是
1006
将短指针
插入发送表项
结束

图 10

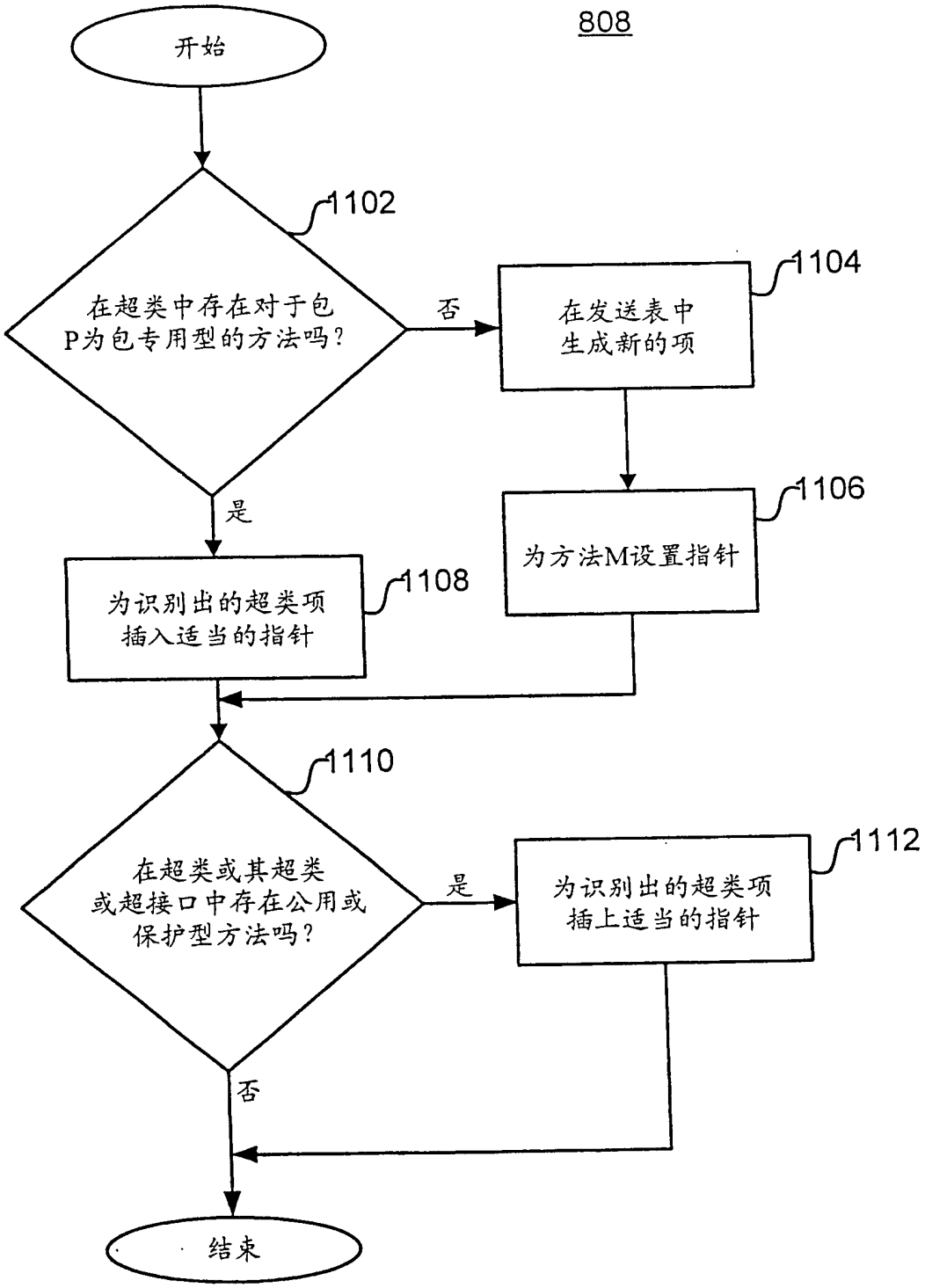

808
开始
1102
在超类中存在对于包P为包专用型的方法吗?
否
1104
在发送表中生成新的项
1106
为方法M设置指针
是
1108
为识别出的超类项插入适当的指针
1110
在超类或其超类或超接口中存在公用或保护型方法吗?
是
1112
为识别出的超类项插上适当的指针
否
结束

图 11

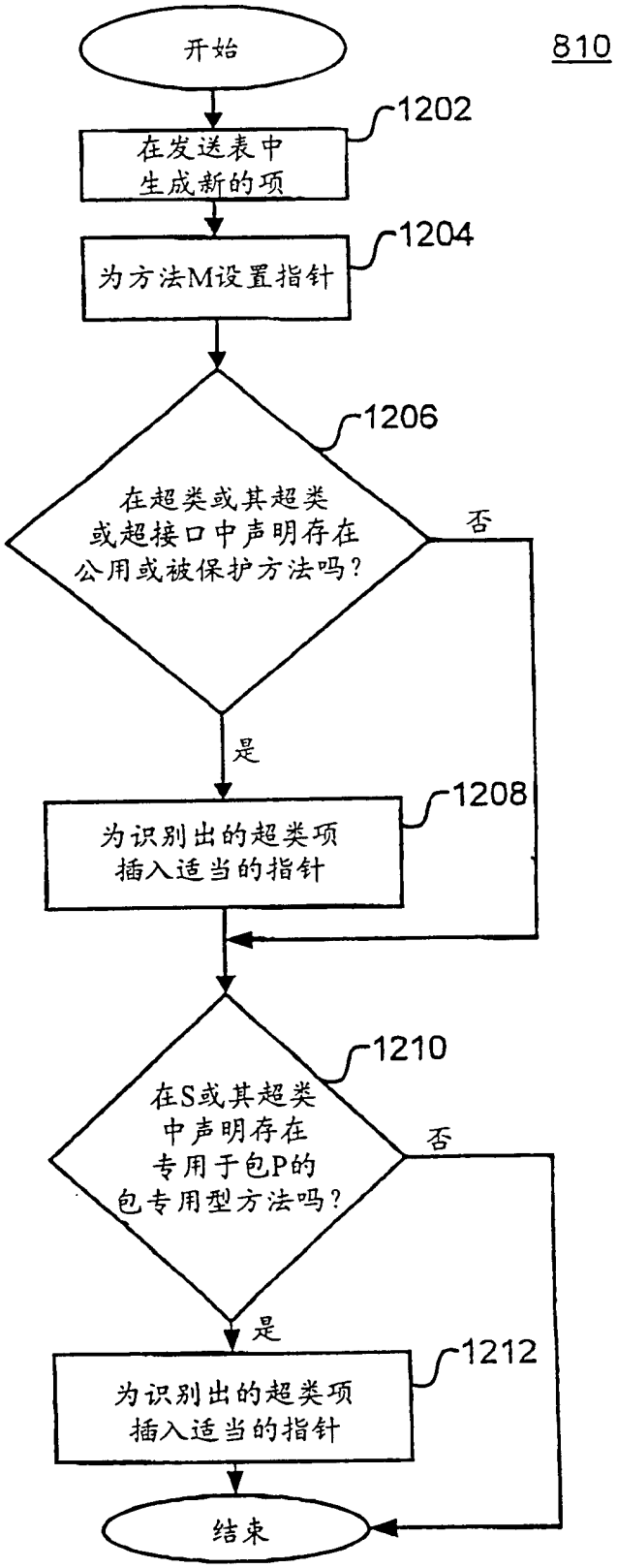

开始
810
1202
在发送表中
生成新的项
1204
为方法M设置指针
1206
在超类或其超类
或超接口中声明存在
公用或被保护方法吗?
否
是
1208
为识别出的超类项
插入适当的指针
1210
在S或其超类
中声明存在
专用于包P的
包专用型方法吗?
否
是
1212
为识别出的超类项
插入适当的指针
结束

图 12

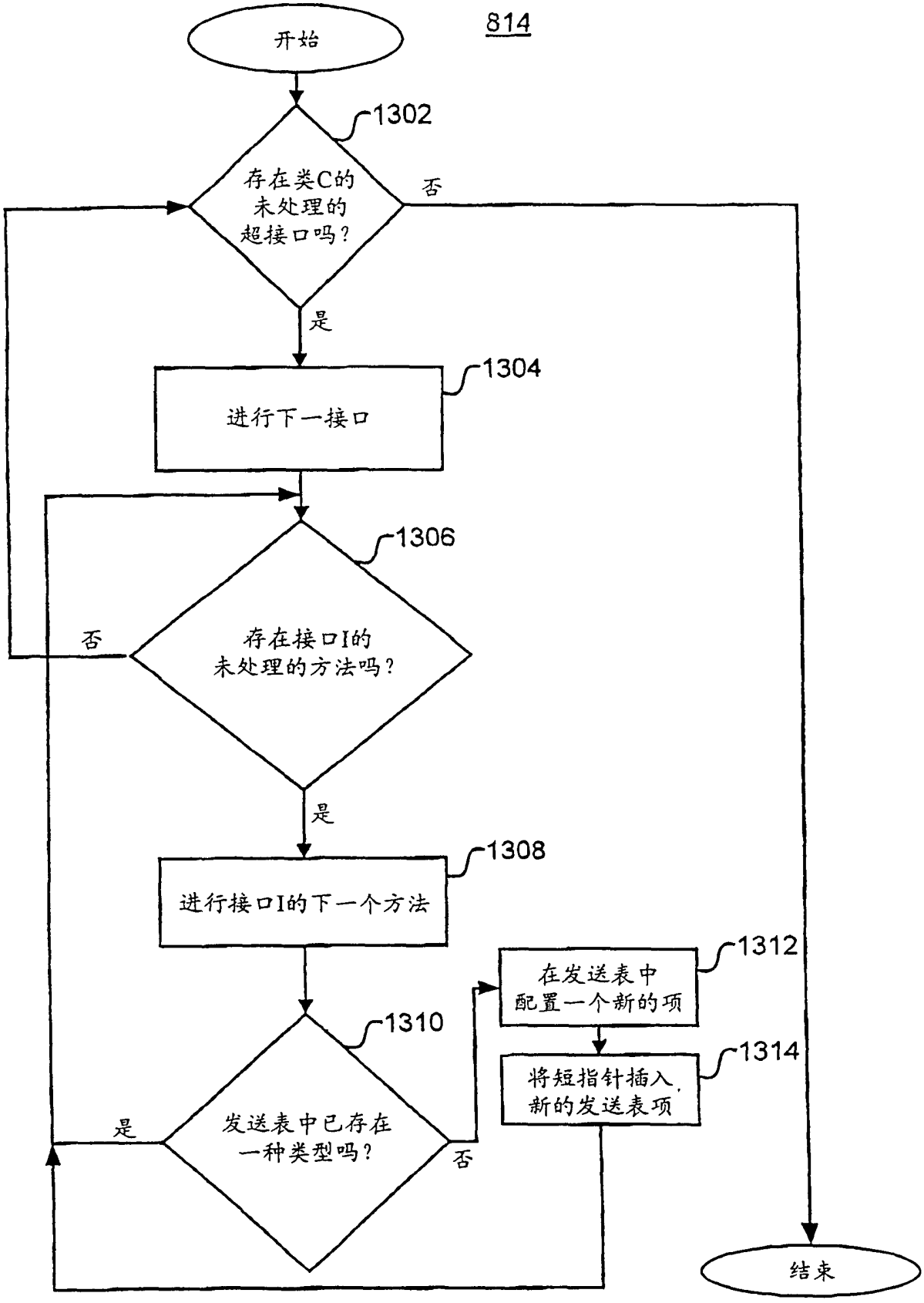

814
开始
1302
存在类C的
未处理的
超接口吗？
否
是
1304
进行下一接口
1306
存在接口I的
未处理的方法吗？
否
是
1308
进行接口I的下一个方法
1312
在发送表中
配置一个新的项
1310
1314
将短指针插入
新的发送表项
发送表中已存在
一种类型吗？
是
否
结束

图 13

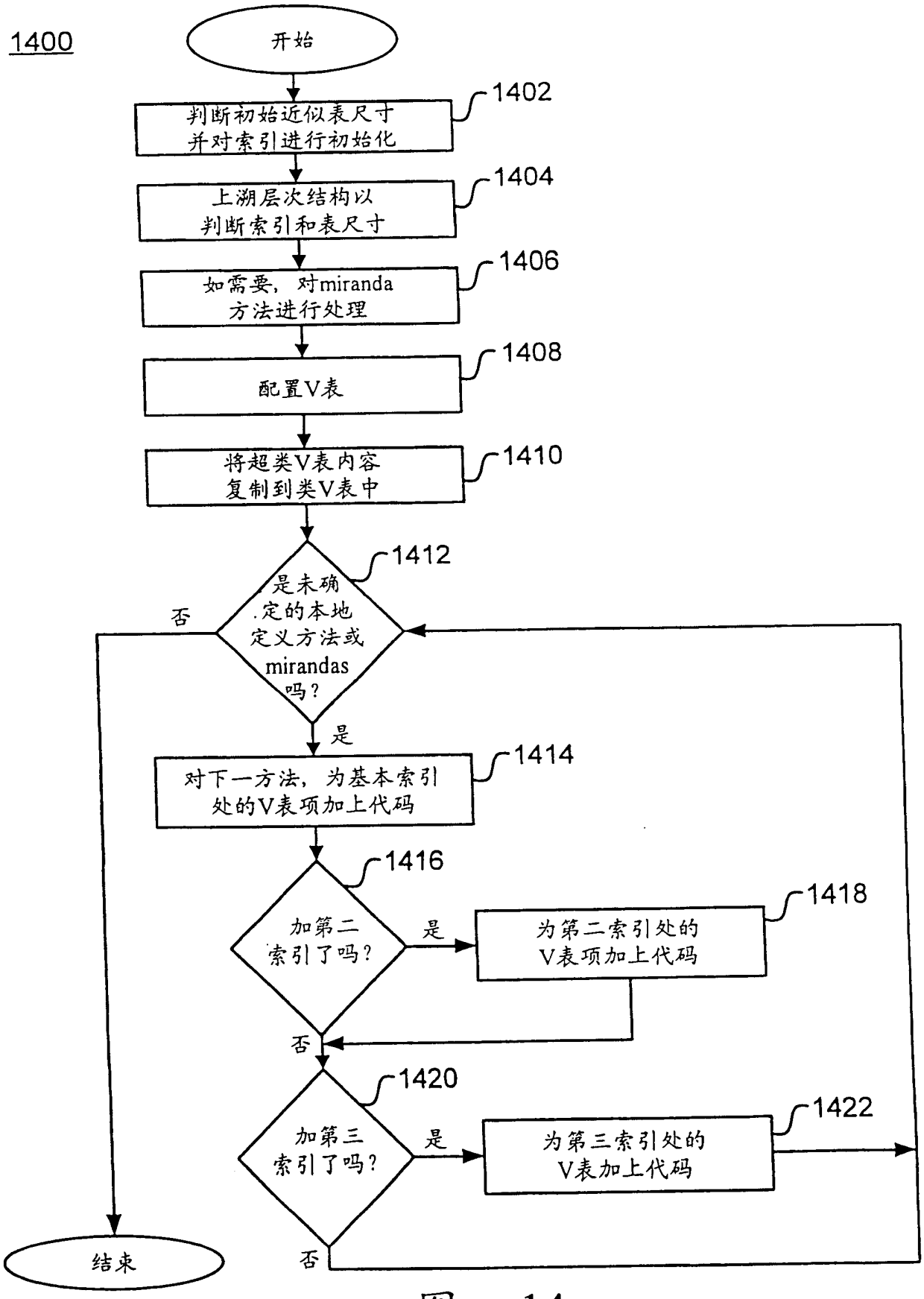

1400
开始
判断初始近似表尺寸
并对索引进行初始化
1402
上溯层次结构以
判断索引和表尺寸
1404
如需要，对miranda
方法进行处理
1406
配置V表
1408
将超类V表内容
复制到类V表中
1410
是未确
定的本地
定义方法或
mirandas
吗？
1412
否
是
对下一方法，为基本索引
处的V表项加上代码
1414
加第二
索引了吗？
1416
是
为第二索引处的
V表项加上代码
1418
否
加第三
索引了吗？
1420
是
为第三索引处的
V表加上代码
1422
否
结束

图 14

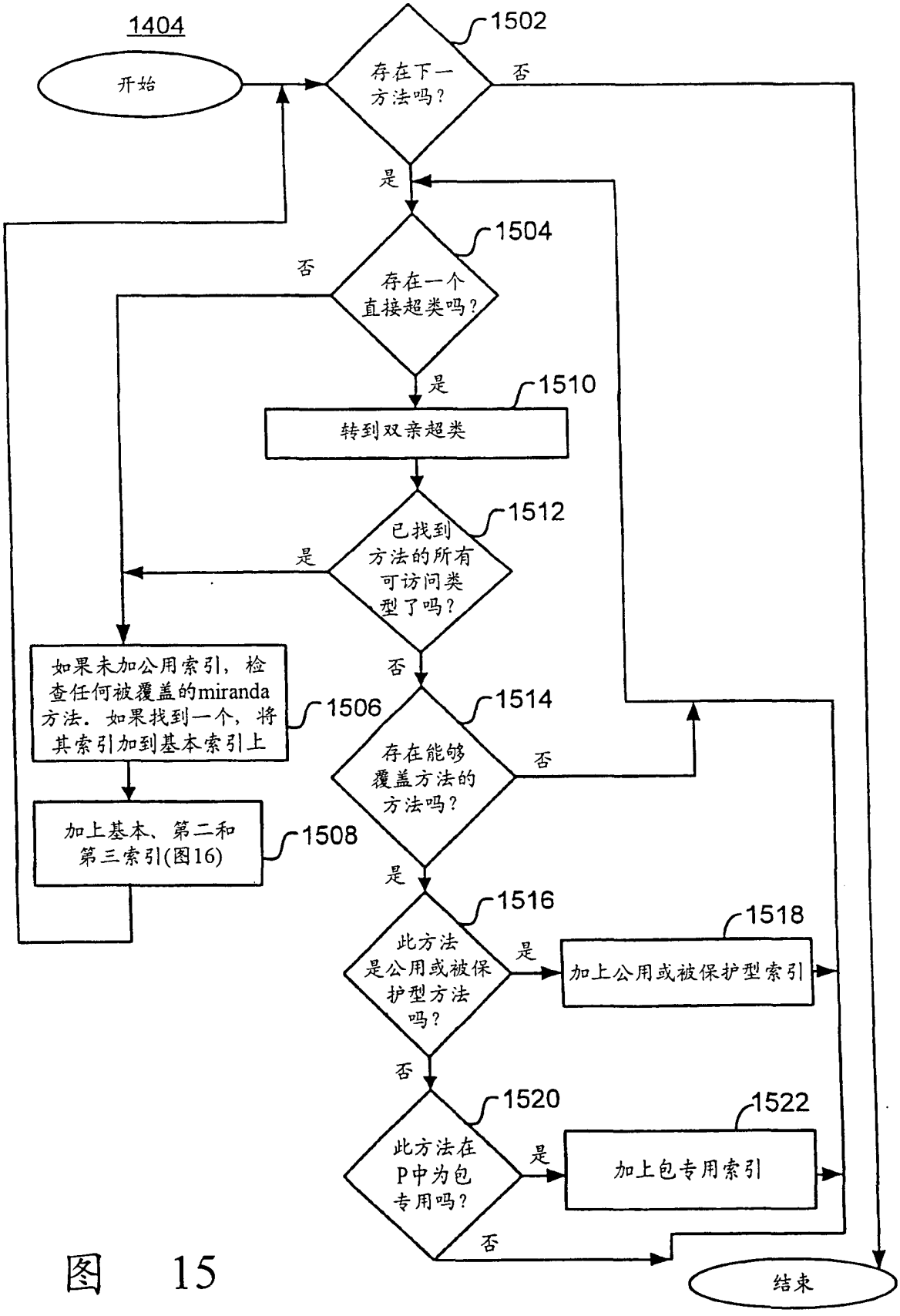

1404
开始
1502
存在下一方法吗？
否
是
1504
存在一个直接超类吗？
否
是
1510
转到双亲超类
1512
已找到方法的所有可访问类型了吗？
是
否
如果未加公用索引，检查任何被覆盖的miranda方法．如果找到一个，将其索引加到基本索引上
1506
加上基本、第二和第三索引(图16)
1508
1514
存在能够覆盖方法的方法吗？
否
是
1516
此方法是公用或被保护型方法吗？
是
1518
加上公用或被保护型索引
否
1520
此方法在P中为包专用吗？
是
1522
加上包专用索引
否
结束

图 15

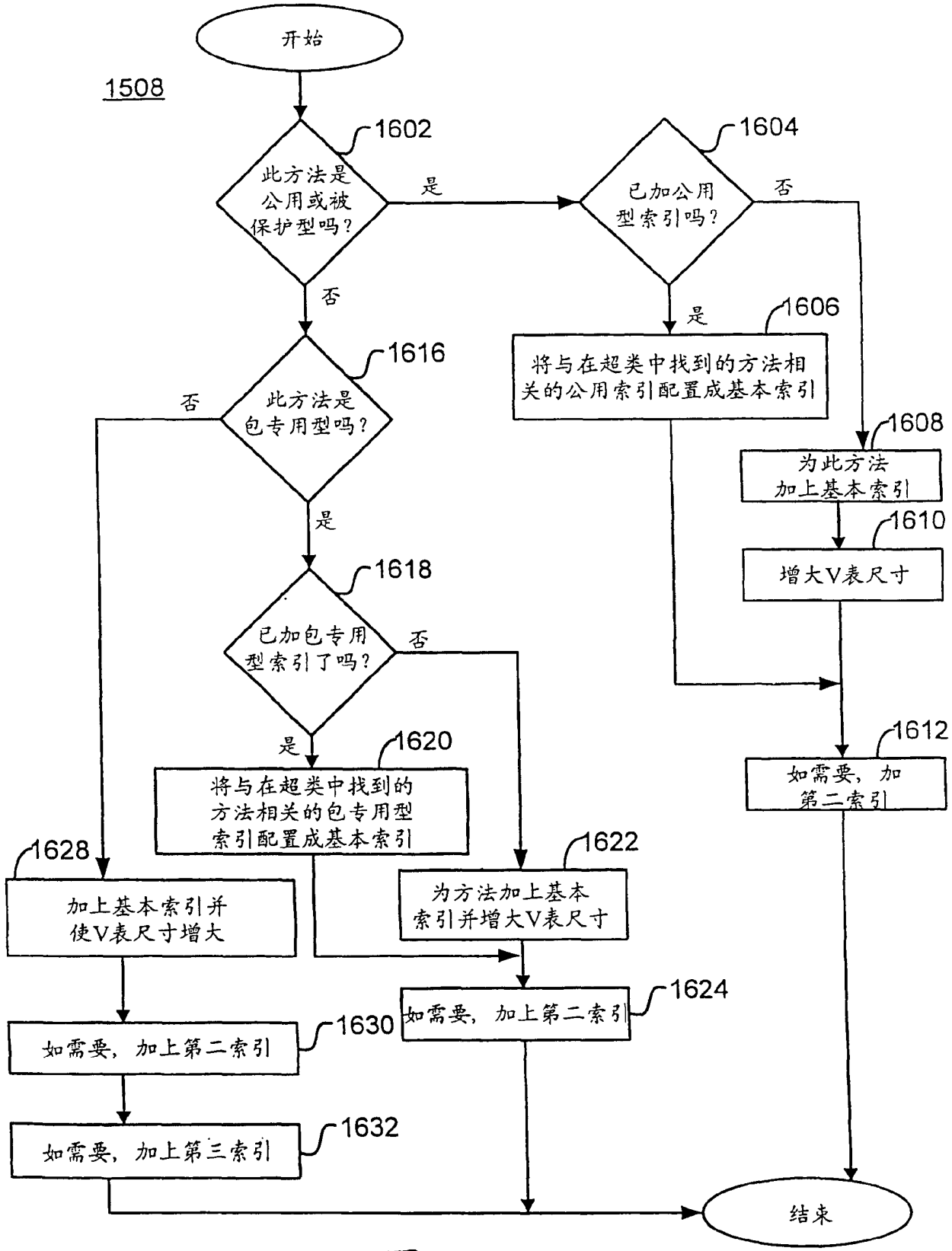

开始
1508
1602
此方法是公用或被保护型吗？
是
1604
已加公用型索引吗？
否
是
1606
将与在超类中找到的方法相关的公用索引配置成基本索引
否
1616
此方法是包专用型吗？
否
1608
为此方法加上基本索引
是
1610
增大V表尺寸
1618
已加包专用型索引了吗？
否
是
1620
将与在超类中找到的方法相关的包专用型索引配置成基本索引
1612
如需要，加第二索引
1628
加上基本索引并使V表尺寸增大
1622
为方法加上基本索引并增大V表尺寸
1624
如需要，加上第二索引
1630
如需要，加上第二索引
1632
如需要，加上第三索引
结束

图 16

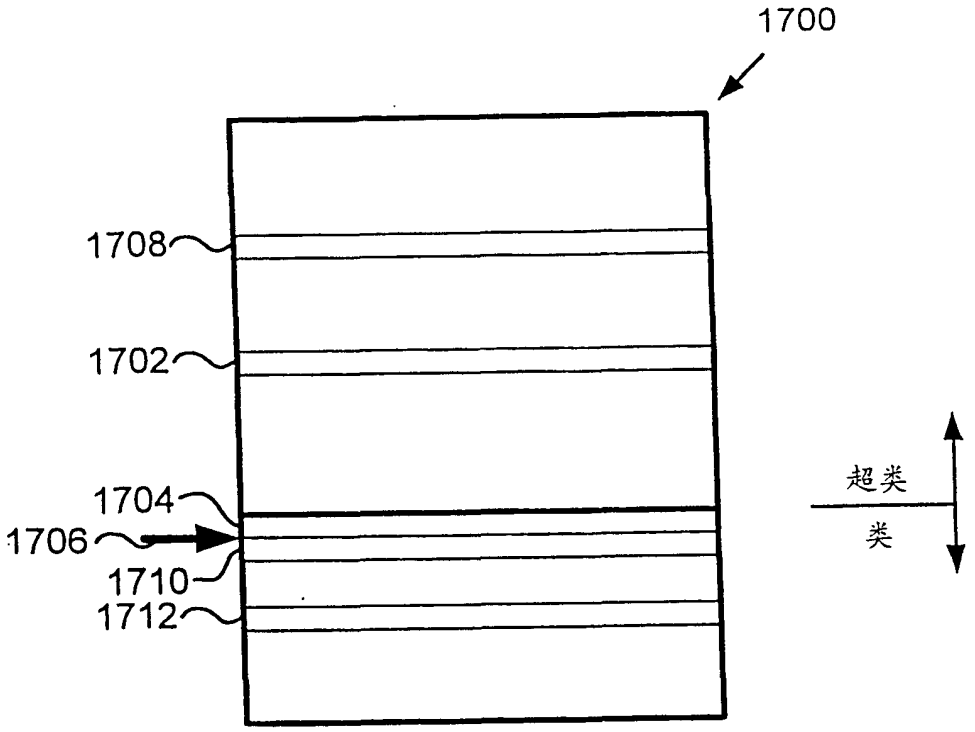


图 17

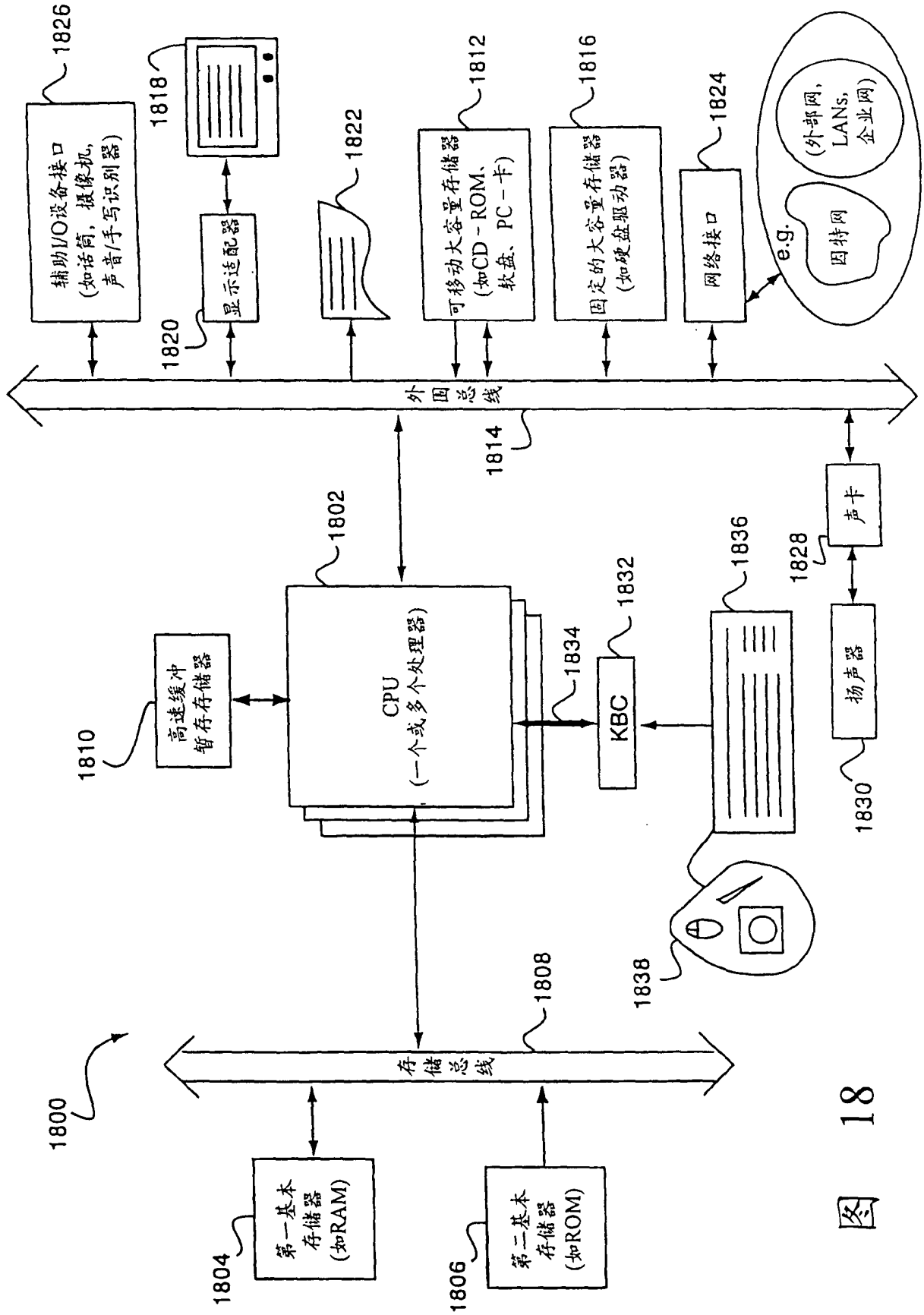

1826
辅助I/O设备接口
(如话筒，摄像机，
声音/手写识别器)
1818
显示适配器
1820
1822
1812
可移动大容量存储器
(如CD-ROM、
软盘、PC-卡)
1816
固定的大容量存储器
(如硬盘驱动器)
1824
网络接口
e.g.
因特网
(外部网，
LANs，
企业网)
外围总线
1814
1802
CPU
(一个或多个处理器)
高速缓冲
暂存存储器
1810
1834
KBC
1832
1836
1828
声卡
扬声器
1830
1838
1808
存储总线
1800
第一基本
存储器
(如RAM)
1804
第二基本
存储器
(如ROM)
1806

图 18