[19]中华人民共和国国家知识产权局

[51]Int. Cl[6]
G06F 9/445

[12]发明专利申请公开说明书

[21]申请号 99104169.0

[43]公开日 1999年9月29日

[11]公开号 CN 1229944A

[22]申请日 99.3.22 [21]申请号 99104169.0
[30]优先权
[32]98.3.20 [33]US [31]045508
[71]申请人 太阳微系统有限公司
地址 美国加利福尼亚州
[72]发明人 川原秀也
内蒂姆·弗雷斯科

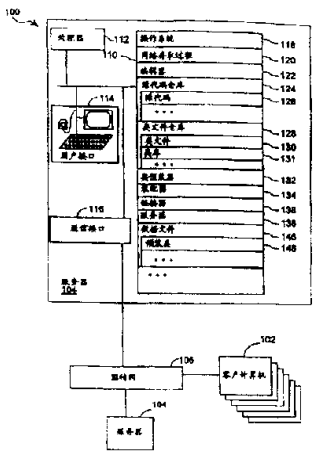
[74]专利代理机构 柳沈知识产权律师事务所
代理人 吕晓章

权利要求书3页 说明书14页 附图页数12页

[54]发明名称 用于减少预装类的脚印的系统和方法

[57]摘要

一种减少由运行时引擎为内部数据结构分配的ROM空间的方法和系统。内部数据结构存储应用程序使用的预装类的成员信息。该系统确定类中表示的内部数据结构的类型,识别每类成员的可能值。该系统还确定在值表中存储每个类型的值所需的空间量和索引该表的每个进入所需的位数。该系统确定成员的出现是用一组值表索引和值表表示,还是用传统方式表示。该确定取决于通用变量大小、成员出现次数、每个索引需要的存储器和值表的大小。



ISSN1008-4274

权　利　要　求　书

1. 一种用于减少要加入到运行时的环境中的预装类的存储器脚印(memory footprint)的方法，包括以下步骤:

确定在一个或多个用于定义多个预装类的类文件中表示的数据结构的类型，每个数据结构类型包括一个或多个成员;

确定可以由每个成员采用的各异的值;

为至少一个子组成员存储所述的每个值，以减少包括所述的预装类的相应的内部数据结构的大小;

产生一组值索引，用于寻址在存储步骤中存储的值;

产生存取器函数和成员声明，能够使运行时的环境使用用所存储的值和所述组的值索引表示的所选择的成员。

2. 一种用于减少要加入到运行时的环境中的预装类的存储器脚印的系统，包括:

一组类文件；和

一个类预装器，构造成从所述的类文件产生一组预装类和多个内部数据结构声明，该结构声明使所述的预装类在存储器中分配时其大小最小化;

所述类预装器构造成产生内部数据结构声明，以使每一组所选择的数据结构成员都用存储结构的索引来表示，所述存储结构保持了所选择的数据结构成员的各异值。

3. 一种计算机程序产品，用于在计算机系统中减少要加入到运行时的环境中的预装类的存储器脚印，在执行引擎上执行，该计算机程序产品包括计算机可读存储器媒体和嵌入其中的计算机程序机构，所述的计算机程序机构包括:

一个类预装器，构造成从一组类文件中产生所述预装类和多个内部数据结构声明，所述结构声明使所述的预装类在执行引擎的存储器中分配时其大小最小化;

所述类预装器构造成产生内部数据结构声明，以使每一组所选择的数据结构成员都用存储结构的索引来表示，所述存储结构保持了所选择的数据结构成员的各异值。

4. 一种运行时的环境，从一预装类的集合中建立，所述预装类由一个或

多个内部数据结构类型来定义，每个内部数据结构类型包括一个或多个成员，所述运行时的环境包括:

一存储结构，保持了可以被至少一个子组的成员采用的各异值；和

一存储结构入口的索引，该存储结构入口保持了所述的子组的成员的各个成员的各异值，用所述的各异值作为所述的各个成员的每次出现;

这样，运行时的环境在必要时通过在由所述索引定义的位置提取所述存储结构的内容，来确定所述的各个成员的各异值。

5. 如权利要求 4 所述的运行时的环境，其中所有的所述的各异值是已知的，并且使用能够索引所有与所述各个成员相关的各异值的最少位，来实现所述存储器结构入口的索引。

6. 一种方法，用于将要在执行引擎上执行的要加入到运行时的环境中的预装类来装载所述的执行引擎，包括：下载所述预装类到所述的执行引擎，所述预装类包括构成所述预装类的多个内部数据结构声明，以使每一组所选择的数据结构成员都用存储的所选择的数据结构成员的各异值的索引来表示，所述选择的组使所述预装类在所述执行引擎的存储器中分配时其大小最小化。

7. 如权利要求 6 所述的方法，其中所述预装类在因特网上下载。

8. 如权利要求 6 所述的方法，还包括：将所述预装类分配在所述的执行引擎的存储器中。

9. 一种方法，用于产生和装载要由客户执行的要加入到运行时的环境中的预装类到客户，以便在所述客户上执行，包括:

在服务器上产生所述预装类，该预装类包括构成所述预装类的多个内部数据结构声明，以使每一组所选择的数据结构成员都用存储结构的索引来表示，所述存储结构保持了所选择的数据结构成员的各异值，所述选择的组使所述预装类在所述客户的存储器中分配时其大小最小化；和

下载所述预装类到所述的客户。

10. 一种用于减少要加入到运行时的环境中的预装类的存储器脚印的方法，包括以下步骤:

确定可以由构成所述预装类的内部数据结构的成员采用的各异的值;

为至少一个子组的成员存储所述的每个值，以减少所述的内部数据结构的大小；和

用子组成员的每次出现的值的索引取代该每次出现，使该每次出现的值能够经所述索引被提取。

11. 一种系统，用于减少要加入到运行时的环境中的预装类的存储器脚印，包括:

一个类预装器，构造成产生一组内部数据结构声明，所述声明使所述的预装类在存储器中分配时其大小最小化，所述内部数据结构声明声明构成所述的预装类的内部数据结构;

所述类预装器构造成产生所述内部数据结构声明，以使每一组所选择的数据结构成员都用存储结构的索引来表示，所述存储结构保持了所选择的数据结构成员的各异值。

说　明　书

用于减少预装类的脚印的系统和方法

本发明主要涉及一种类预装器，尤其涉及一种用于减少预装的 Java 类在只读存储器中的大小的系统和方法。

一个 Java 程序包括几个小的软件成分，称为类。每个类包括代码和数据，由各个类文件中的信息来定义。每个类文件根据相同的独立于平台的“类文件格式”来组织。参看图 1，示出了类文件格式的方框图，根据该方框图，每个类文件 400 包括首部信息 402、常数存储池 404、方法表 406 和字段表 408。首部信息 402 确认类文件格式、常数存储池的大小、在方法表 406 中的方法的数目和在字段表 408 中的字段的数目。常数存储池 404 是结构表，代表各种串常数、类名、字段名和在类文件结构和其子结构内所参看的其它常数。方法表 406 包括一个或多个方法结构，其中每个给出了由类显式声明的方法的全部描述和 Java 代码。字段表 408 包括一个或多个字段结构，其中每个给出了由类声明的字段的全部描述。字段表 408 的一个例子现在参看图 1B 描述。

Java 程序是在一台包括称为虚拟机（ VM ）的程序的计算机上执行的，该虚拟机负责执行 Java 类的代码。习惯上在程序执行中尽可能迟地装载 Java 程序的类：在程序执行中，当第一次引用时，它们根据从网络服务器或从本地文件系统来的命令被装载。VM 为不同的成分定位和装载每个类，分析类文件格式，分配内部数据结构，和将其与其它的已装载的类链接。该进程使类中的方法代码容易被 VM 执行。

对于小的和嵌入的系统，类装载需要的设施，诸如网络连接、本地文件系统或其它永久存储器是没有的，希望能够将类预装入只读存储器（ ROM ）。在美国专利申请序列号 No. 08/655474（“一种用于在只读存储器中预装类的方法和系统”）中描述了一种预装方案。这里完全地录入，以供参考。在该方法和系统中，在存储器中的代表类、字段和方法 VM 数据结构是由类预装器离线产生的。然后，预装器输出被链接在包括 VM 的系统中和放在只读存储器中。这样就不必存储类文件和进行动态类装载。

参看图 2A，示出了由类预装器产生的 VM 数据结构 1200 的更详细的方

框图。数据结构 1200 包括类块 1202、多个方法块 1204，多个字段块 1214 和常数存储池 1224。

类块 1202 是固定大小的数据结构，可以包括以下信息:

●类名 1230;

●指针 1232，指向当前类的类块的即时超类;

●指针 1234，指向方法块 1204;

●指针 1236，指向字段块 1214；和

●指针 1238，指向类的常数存储池。

类块数据结构的元素这里被称为类块成员。

方法块 1204 是固定大小的数据结构，代表一种类方法。

方法块数据结构的元素这里被称为方法块成员。

字段块 1214 是固定大小的数据结构，代表一种类的实例变量。

字段块数据结构的元素这里被称为字段块成员。

VM 数据结构的每个类型，包括类块 1202、方法块 1204、字段块 1214 和常数存储池 1224，都具有由对应的数据结构声明定义的格式。例如，单个方法块声明定义了所有方法块 1204 的格式。数据结构声明也定义了存取器函数（或宏），由 VM 使用以存取数据结构成员。这些数据结构声明对 VM 是内部的，不能由类成分使用。现在参看图 2B 描述现有技术的数据结构声明。

参看图 2B，示出了数据结构声明 1230 的描述，其定义了由特定的 VM 采用的所有数据结构类型的格式。每个声明 1230 包括一组成员声明 1232 和用于存取各个成员的存取器函数 1234。成员声明 1232 和存取器函数 1234 是根据在 VM 的实施所用的语言的语法，传统地定义的。例如，假定在数据结构声明 1230 中使用 C 语言，类属的(generic)字段数据结构 1230.N（图 2B 中所示）可以被定义为有五个下面的类型的成员和相应的存取器函数的结构 T:

成员名	成员类型	存取器函数	
member1	mtype1	(T)的 mem1	T->member1
member2	mtype2	(T)的 mem2	T->member2
member3	mtype3	(T)的 mem3	T->member3
member4	mtype4	(T)的 mem4	T->member4
member5	mtype5	(T)的 mem5	T->member5

在该例子中，成员类型可以是由相关的计算机语言定义的任何类型，包括用户定义的类型或语言类型，诸如整型、浮点型、字符型或双精度型。存取器函数是由 VM 使用的宏，以存取字段，而无需直接存取包含该字段的结构。例如，替代采用表达式“ T->member1 ”，以存取结构类型 T 中的字段 1，VM 仅需要采用表达式“(T)的 mem1 ”。存取器函数在诸如 C 的程序语言中是众所周知的，它能够提供复杂的数据结构。

用于存储“类元数据”（即，类、方法和字段块 1202 、 1204 、 1214 ）的内部数据结构，需要只读存储器中大的固定量的空间。实际上，测量显示这种类元数据常常比类方法本身的字节代码占据更多的空间。这些内部数据结构因而常常不适用于小的资源受限的装置，在这些装置中类预装是想要的和/或必须的。

另外，如果内部数据结构被单个修正，以节约存储空间， VM 代码将需要被大范围地修改，以便使 VM 能够正确地存取修正的数据结构。对 VM 做这样的变化是艰巨的和低效的。

因而，需要修正内部数据结构，它比现有技术的数据结构要小，且包括 VM 需要的所有信息，不需要 VM 代码的大范围的或艰巨的修正。

总的来说，本发明是一种减少预装 Java 类所需的 ROM 空间的方法和系统。

特别地，本发明的方法和系统的根据是，在 Java VM 类被预装的环境中，很可能 VM 应是一个封闭的系统，具有一组类和类成分，诸如字段和方法。这样的封闭的 VM 应包括固定数目的内部数据结构，诸如类块、方法块和字段块。另外，这些数据结构的每个成员（例如，方法块或字段块成员）应有已知组的各异值之一。

在这一假设及其推论下，本发明减少了需要的存储空间，用下面的方法表示内部数据结构：

1) 确定每个数据结构成员的类型的各异值；
2) 确定每个数据结构成员类型的出现（例如，在方法块中字段块成员类型的每次出现）和每次出现的值；
3) 确定如果每次出现用数据结构成员类型的值表的索引表示而不是用传统的方式（在通用变量中存储每次出现的值）表示所节省的存储空间；

4) 如果结果是节省了足够的空间，那么分配包含各异数据结构成员类型值的值表，用合适的值表入口的索引构造该字段块成员类型的每次出现；和

5) 产生新的源给 VM，这样其对修正的结构的存取被自动适应。

在一个优选实施例中，如果下面的比较为真，就决定用值表索引和值表来表示数据结构成员类型:

(#类型的出现次数)×（索引的大小）+（值表的大小）<(#类型的出现次数)×（通用变量的大小）

一旦本方法已经确定，对于每个数据结构成员类型，该类型的出现是用值表的索引来表示，还是用存储值的通用变量来表示，那么本方法发出该类型的合适的信息，包括存取器函数，语言声明和初始化值表的源代码。存取器函数是宏，通过它，所有的运行时的对数据结构成员的存取由 VM 完成。最好是，在发出上述信息之前，本方法确定值表索引、传统的成员表示法和值表的最紧密的安排，相应地产生值表、值表索引，存取器函数和类。

本方法在确定是否修正数据结构成员的传统表示法之后，发出存取器函数，声明和其它数据结构信息。结果是，所有发出的数据结构信息与在内部类表示法中的变化一致。这种一致的数据结构信息的自动产生使 VM 的改变最小化，这一点在新类加入 VM 和类表示法改变的任何时候都是需要的。这一点是对现有技术的重大改进。

本发明的系统包括类文件、实现了上述方法的 Java 类预装器和由预装器产生的输出文件的集合，该文件包括预装类、首部文件和源代码文件。

类文件定义了要预装的类的完整的组。预装器在类文件上执行第一通路(pass)，以确定:

内部数据结构的成员的不同类型，

每个类型成员的各异值，

存储这些值需要的空间量，

值索引的大小，和

每个成员类型的出现的次数。

预装器然后在类文件和内部数据结构上执行第二通路，以确定每个成员要如何表示，是用传统的方式还是用值表入口的索引，然后发出合适的输出文件。

输出文件和由传统的 Java 系统采用的类似文件是兼容的。就是，预装类可以被装配或编辑到类对象数据中，首部文件和源文件可以用 VM 被编辑到 VM 对象数据中。 VM 和类对象数据然后可以被以传统的方式链接到特定 Java 环境的可执行的 VM 中。

通过参照附图的以下详细描述和所附权利要求，本发明的其它目的和特点将更加清楚，附图中:

图 1 示出了现有技术和本发明的公共的类文件格式;

图 2A 是在现有技术中所用的 VM 内部数据结构的方框图，该数据结构是为了编码类、方法和字段信息;

图 2B 是示出定义图 2A 所示的 VM 内部数据结构的格式的数据结构声明;

图 3 是分布计算机系统的方框图，其中本发明的类预装器方法和系统可以被实现;

图 4 是在图 1 的分布计算机系统中的执行引擎的方框图，其中，由图 3 的类预装器产生的预装的类被装载入 ROM;

图 5 是示出用于产生预装的可执行模块的进程成分的流程图;

图 6 是示出用于减少预装的可执行模块的存储脚印(footprint)的进程成分的流程图;

图 7A 示出更新的图 6 的首部文件 614 的组织;

图 7B 示出图 6 的值表 616 的组织;

图 8A 示出根据本发明在执行引擎 ROM208 中分配后同一成员的出现和值的组织;

图 8B 示出根据现有技术在执行引擎 ROM208 中分配后同一成员的出现的组织;

图 9A 示出了具有由本发明产生的五个成员的数据结构实例的紧密组织;

图 9B 示出了由现有技术产生的图 9A 的数据结构实例的组织;

图 10 是流程图，示出了由类预装器使用的方法，以建立在预装类中使用的内部数据结构; 和

图 11 是示出预装的应用程序映射到只读存储器和随机访问存储器中和指示由静态类启动程序映射到随机访问存储器的方法和数据的部分的装载的

方框图。

这里描述的方法和系统是针对 Java 类预装器的。该 Java 类预装器被构造成输出预装 Java 类，该 Java 类被优化以存储在目标计算机（这里被称为执行引擎）的 ROM 中。假若执行引擎可能是有很少或没有辅助存储器的计算机，最好 Java 类预装器在独立的计算机中实现，该计算机这里被称为服务器。假定这种结构，预装的类可以从服务器以各种方式（例如，网络连接、直接通信链接或可读媒体如软盘或 CD 的“寄生者网络”传递）传递到执行引擎。相应地，这里描述的本发明的优选实施例是针对有服务器和执行引擎的计算机系统的，其中预装的类由服务器产生，随后被传递到执行引擎以用于 VM 中。现在参看图 3 和 4 描述优选实施例。

参看图 3，示出了分布计算机系统 100，本发明在其中实现。计算机系统 100 有一个或多个执行引擎 102 和一个或多个服务器计算机 104。在优选实施例中，每个执行引擎 102 经因特网 106 连接到服务器 104，尽管也可以使用计算机 102、104 之间的其它类型的通信连接（例如，网络连接、直接通信链接或可读媒体如软盘或 CD 的“寄生者网络”传递）。最好是，服务器和执行引擎是桌面计算机，诸如 Sun 工作站，IBM 兼容机和/或苹果 Macintosh 计算机；然而，实际上任何类型的计算机都可以是服务器或执行引擎。另外，该系统不限于分布计算机系统。它可以在各种计算机系统中和在紧密连接的处理器的各种结构、或者模型中或者在松散连接的微处理器系统的各种结构中实现。

服务器计算机 104 一般包括一个或多个处理器 112、通信接口 116、用户接口 114 和存储器 110。存储器 110 存储：

- 操作系统 118；
- 因特网通信管理器程序或其它类型的网络存取过程 120；
- 编辑器 122，用于将用 Java 程序语言写的源代码翻译为字节代码流；
- 源代码仓库 124，包括一个或多个包含 Java 源代码的源代码文件 126；
- 类文件仓库 128，包括一个或多个独立于平台的类文件 130 和一个或多个包含类文件的类库 131，每个类文件包含代表特定类的数据；
- 类预装器 132，产生一组预装类 148，用于执行引擎的特定结构（类预装器有时称为静态类装载器）；

● 装配器 134，产生链接器可识别的格式的代表类成员的对象文件，类数据结构和存储器指示器；

● 链接器 136，用于确定一组预装类的存储布局和解决所有的符号引用；和

● 一个或多个数据文件 146（包括预装类 148），由服务器使用。

注意类文件仓库 128、类预装器 132、装配器 134、链接器 136 不需要驻留在服务器 104 上，但是可以在任何其输出（例如，代表预装类 148 的文件和消息）可以被拷贝到执行引擎 102 的计算机上。

参看图 4，执行引擎 102 可以包括一个或多个处理器 202、通信接口 206、用户接口 204、只读存储器 208 和随机访问存储器 210。只读存储器 208 存储没有未解决的引用的程序方法和在程序运行中保持常量的程序数据。在优选实施例中，在 ROM208 中存储的方法和数据包括 Java 应用程序 212 的部分和执行引擎的支持过程。执行支持过程包括操作系统 213、网络存取过程 214、预装类 232 和由预装类 232 使用的内部数据结构 1200（图 2）。

随机访问存储器 210 存储：

● Java 应用程序 215 的第二部分和支持过程 216、217，包括有未解决的引用的方法和在应用程序的执行中改变的数据；和

● 一个或多个数据文件 228，执行引擎在其进程中可能利用。

参看图 5，示出了一个流程图，表示用于产生预装的可执行模块的步骤的顺序。应该注意这里所述的方法和系统属于预装一 Java 应用程序和其它支持过程。任何 Java 应用程序或者任何其它系列的在运行时间中正常链接的方法可以用这里所述的方法和系统预装。

包括 Java 应用程序的每个类的源代码 126，由编辑器 122 编辑进入类文件 130，类文件 130 是独立于平台的类的表示法。如图 1 中所述，类文件包含字段和方法表、每个方法的字节代码、常数数据和其它信息。或者，对应于应用程序的类文件可以已经驻留在一个或多个类库 131 中。整个组的类文件 128 构成预装的应用程序，被传送到类预装器 132。

类预装器的工作是为执行引擎 102（图 4）产生预装类 148。预装类 148 包括参考图 2 描述的类块 1202、方法块 1204、字段块 1214 和常数存储池 1224。在其它物件中，类预装器 132 确定哪个与每个类 130 相关的方法和字段可以被存储在只读存储器 208 中，哪个必须存储在随机访问存储器装置 210

中。例如，调用 Java 接口或利用非静态实例变量的方法需要驻留在随机访问存储器中。这是因为实现接口的字节代码在运行时间确定，对于每个相关的类的实例化，非静态实例变量是变化的。

在代码被装载入执行引擎 ROM208 时，类预装器 132 也执行许多优化，以便产生更紧密可执行代码的内部表示法。例如，类预装器 132 结合与每个类相关的常数存储池，以便去除在类常数存储池 310 的内部表示法中的冗余。根据本发明，类预装器 132 也修正内部数据结构 1200（图 2A），以便在执行引擎 102 的 ROM 中占据较少的空间。本发明的一个优点是该数据结构优化使内部表示法大大摆脱了现有技术中所用的的低效标准数据结构格式 1200。

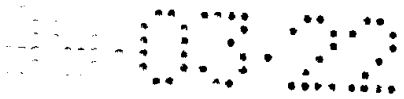
预装类 148 被传送到装配器或编辑器 134，编辑器 134 产生具有链接器 136 需要的格式的对象模块 304，以便将数据映射到合适的地址空间。最好是有两个地址空间，一个用于随机访问存储器装置，另一个用于只读存储器装置。然后对象模块 304 被传送到链接器 136，链接器 136 为应用程序中的类产生存储布局。一旦确定了存储布局，链接器 136 就解决所有的符号引用，并用直接地址取代它们。存储布局被分区为两个地址空间。只读存储器标志（flag）的方法和字段包括在第一地址空间中，随机访问存储器中标志为需要的存储器的方法和数据包括在第二地址空间中。链接器 136 的输出是预装的可执行模块 306，可执行模块 306 包含这两个地址空间的方法和数据。现在参考图 6 描述本发明的进程流。

参考图 6，示出了本发明采用的进程的数据流图，目的是减少由 VM 使用的内部数据结构的存储脚印。如已参考图 3 所描述的，类预装器 132 从类文件 128 产生一组特定平台的预装类 148。预装类 148 是数据结构声明，可以在装配器源中声明或由高级语言声明。装配器 134 或编辑器 122 然后将这些数据声明转化成对象数据 622。类预装器 132 也确定组成预装类 232 的内部数据结构 1200 的最有效的表示法。

在优选实施例中，内部数据结构的一个成员可以用两种方法之一来表示：

1. 作为一种类属的记忆字（generic memory word）（例如，32 位），该记忆字是成员的值；或

2. 作为各异值的表的索引，每次成员出现时该值可以被成员采用。



第一种表示法仅是在现有技术的类预装器的数据结构中所用的表示法。当特定成员仅有几个不同值而存在成百上千次的出现时，该表示法是很低效的。在该情况下，即使仅存储几个不同的值，全宽的记忆字（例如，32位宽）也被分配给每次出现，在ROM208中占据了成千个字的难得的存储空间(scarce storage)。本发明采用的第二种表示法解决了该问题。它产生一个值表616，以保持这样成员的有限的值，它为成员的每次出现产生一个索引，该索引只有寻址所有的值表入口所需要的那么多位。第二种表示法的优点是，当应分配给特定成员类型的索引和值表的存储器小于类属的表示法需要的被分配的存储器时，第二种表示法是有利的。参看图10，下面描述本发明确定的方法如何编码成员数据结构。

一旦确定如何表示这些成员，类预装器132为要以索引+表的格式表示的每个成员输出更新的首部信息614（包括修正的成员声明和能够使VM246存取修正的成员信息的存取器函数）和相应的值表616。作为源代码产生的首部信息614和值表616，与虚拟机源618一起由编辑器122编辑。虚拟机源618定义了要在执行引擎102中执行的虚拟机。链接器136链接导出的对象数据620和对象数据622，以产生预装的可执行模块306，预装的可执行模块306可以被装载入执行引擎102。本发明的一个副产品是，无论何时新类或成员要被加入预装类148中时，必须产生新的VM246。这是因为对应的首部信息614和值表616必须和VM源618一起被编辑。然而，因为本发明自动产生任何组的类的首部信息614和值表616，所以产生新VM不需要或需要最少的VM代码的改变。这是因为VM246总是利用存取器函数，存取器函数是首部信息614的一部分。这样，本发明能够产生有效的数据结构成员的表示法，同时使VM的产生容易。

类预装器132能够产生有效的索引/表成员表示法，因为成员的所有可能值是已知的。作为结果，每个索引需要的位的数目也是已知的。每个成员的出现数目也是已知的。另外，优选实施例假定，类文件128代表要被预装入目标执行引擎102的全部组的类。这一假定特别适用于执行引擎102，该执行引擎102是小的手持计算机，不可能有计算功率和/或通信带宽，以用传统方式在空中下载类。假定索引的数目和值是已知的，且没有可能性加入附加的成员或类，那么当由执行引擎102分配时，对于类预装器132，有可能安排索引具有最佳的紧密的或近似最佳的安排。类预装器132通过在更新的首

部信息 614 中选择索引的次序，实现这一层次的紧密性。

参看图 7A 和 7B，示出了更新的首部信息 614 和值表 616 的组织，以及每个数据结构的特定例子。这些例子表示了对应于图 2B 的数据结构声明 1230.N 的由类预装器 132 产生的输出。

图 7A 中所示的更新的首部信息 614 包括一组数据结构声明 702，其每个可以以任何结合的方式包括更新的成员声明 704 和未更新的成员声明 706。每个数据结构声明 702 对应于由 VM246 使用的数据结构之一。更新的成员声明 704 是用于已经由类预装器 132 修正的数据结构成员作为索引/表成员，未更新的成员声明 706 是用于类预装器 132 确定的数据结构成员，最好用类属的方式表示。每个数据结构声明 702 与更新的成员表声明 708、更新的成员存取器函数 710 和未更新的成员存取器函数 712 相联系。每个更新的成员表声明 708 与对应的值表 616 相联系，以合适的程序语言声明该表。更新的成员存取器函数 710 用在相应的更新的成员表声明 708 中定义的表名，为更新的（即，索引/表）成员定义存取器函数。未更新的成员存取器函数 712 和由传统的类预装器 132 产生的是一样的。

例如，图 7A 示出更新的首部文件信息 614，用于图 2B 的数据结构 1230.N（Struct T）。该例子假定类预装器 132 确定：

1) member1 有 400 个值，最好以索引/表成员来表示，

2) member2 最好传统地表示，

3) member3 有 200 个值，最好以索引/表成员来表示，

4) member4 有 1500 个值，最好以索引/表成员来表示，和

5) member5 最好传统地表示。

结果是，类预装器 132 已经产生一修正的“Struct T”声明 704，其中 member1 以 9 位整数索引 m1_idx(9 位足够存取 400 个值)表示，member3 以 8 位整数索引 m3_idx(足够存取 200 个值)表示，member4 以 11 位整数索引 m4_idx(足够存取 1500 个值)表示。其它成员，member2 和 member5 分别留下不修正，作为类型 mtype2 和 mtype5 的类属成员。

类预装器 132 也已经产生 member1 的更新的成员表声明 708，表明 member1 的值存储在类型 member1 的值表(member1_value[])中。member1_value 表被声明为外部变量（extern），告诉编辑器 122 表的实际值定义在另一文件中，在此例中，是在值表文件 616 中。为 member3 和 member4

产生相似的更新的成员表声明 708。

更新的 member1 的存取器函数 710 相应地修正，这样每次相应的存取器函数(T)的 member1 被调用时，存取预装的方法的 VM246 使用 member1 值（即， 9 位 m1_idx ）作为 member1_value 表的索引。更新的 member3 和 member4 的存取器函数 710 以相似的方式被修正。

参看图 7B，示出了值表 616 的表示法，包括表 722.1，定义了可以被在首部文件 614 中声明的 member1_value 表采用的有限值。在该情况下，member1_value 表被定义为常数数列（“ const mtype memeber1_value[]”），包括 400 个值： val1,...,val400 。也提供了类似的 member3 和 member4 的值表的表示法（例如，在 member3 和 4 表 722.3， 722.4 中）。

参看图 8A，示出了本发明的内部数据结构（具体说，单个成员类型的成员出现(occurrence) 802 和值表 806 ）在执行引擎 ROM208 中的组织方式。每次发生表示在预装类中同一成员 802 和拥有其的数据结构类型 805 的一次出现（数据结构类型 805 可能包括多个成员，见图 2B ）。假定一个特定成员有 N 个各异的值 808，存储在值表 806 中，该成员的 M 次出现 802 的每一次被分配为到值表 806 的入口的宽（ $\lfloor \log_2(N) \rfloor + 1$ ）位的索引 804，该值表 806 保持了成员的值。例如，出现 802.1 和 802.6 的每个都是到值表 806.N 的索引。该入口 806.N 存储了与那些成员出现相关的有限值 808.N 。这样，该模型的整个存储使用量是每成员 M × （ $\lfloor \log_2(N) \rfloor + 1$ ） + value_table_size 位。

参看图 8B，示出了现有技术在特定成员的执行引擎 ROM 中组织出现 852 的方式。出现 852 的每一个表示在预装类中特定成员的一次出现。该成员的 M 次出现 852 的每一次被分配为全宽记忆字，该全宽记忆字存储了该成员这次出现的值 854（即，这些出现的每个都以上述第一格式来表示）。这样，该模型的整个存储使用量是 M × 32 位（假定 32 位记忆字）。结果是，当 M × （ $\lfloor \log_2(N) \rfloor + 1$ ） + value_table_size 小于 M × memory_word_size（例如， M × 32 ）时，本发明节省了在数据结构中分配给特定成员的存储器。如图 8A 的例子中，字段 802 可能正是数据结构声明中的一个元素。

参看图 9A，示出了本发明的类预装器 132 如何在执行引擎 ROM 中有效存储特定数据结构 902 的所有成员 802（例如，图 2B 的 Struct T1230.N 的结构的成员）的例子。通常，本发明封装存储的值（即，索引 804 ），这样它们占据了尽可能多的固定长度记忆字。在所述的情况下，记忆字是 32 位宽，

但是本发明适用于任何长度的记忆字。在图 9A 中所示的例子中， Struct T902 的 9 位、 8 位和 11 位成员 m1 、 m3 和 m4 被封装入单个的 32 位记忆字。成员 m2 和 m5 的值，用传统的方式表示（例如，作为 32 位值），被存储在跟着第一个字的相应的 32 位通用变量中。在优选实施例中，这些传统地表示的成员必须被统一为字的范围（例如，每个成员 32 位）。对于修正的成员没有这一要求。因而，对于每个数据结构实例，在第四个成员 m4 和第一个通用变量 m2 之间仅有 4 位的未用空间 904 。类预装器 132 目的在于，对给定的任何成员表示法和成员大小的结合，将内部数据结构的成员尽可能有效地封装在记忆字中。

参看图 9B，示出了图解由现有技术类预装器存储的同一数据结构 Struct T 的格式。注意，在该系统中，数据结构需要 5 个字以存储 5 个成员。这样，现有技术远比本发明（仅需 3 个字以存储同样的数据结构信息）低效。现在参看图 10 描述本发明的方法。

这一安排不会给存取器函数的预装类的使用带来问题，由于索引 804 的不同的存储位置由编辑器 122 解决且索引本身存储了它们的相关值 808 的索引。

参看图 10 ， 示出了在类预装器 132 中实现的本发明的方法的流程图。本方法的实现有两条通路，包括一条记帐通路（由 1104 所标的方框表示）和数据结构声明产生通路（由其余的步骤表示）。作为第一记帐步骤（对所有内部数据结构执行）， 类预装器 132 识别内部数据结构的所有成员类型（ 1106 ）。例如， 参看图 2B， Struct T 的五个成员是数据结构的成员类型。类预装器 132 然后对于每个成员类型执行以下进程:

识别成员类型的 M 个出现（ 1108 ）;

识别 M 次出现的 N 个值（ 1110 ）;

确定存储每个值所需要的存储空间（ 1112 ）;

确定存储可以寻址 N 个值的一个索引所需要的存储空间（索引必须至少是 $\lfloor \log_2(N) \rfloor + 1$ 位）（ 1114 ）;

确定成员出现的传统表示法的大小（ 1116 ）。

该进程在进行以方框 1118 开始的步骤之前，对所有内部数据结构的所有成员执行。该进程次序较好，由于在方框 1104 中由过程产生的记帐统计数字被随后的第二通路步骤使用。通常，暂时存储记帐统计数字，以用于第二

通路。

一旦所有的统计数字已经产生，类预装器 132 为每个成员类型计算:

传统的成员出现的表示法所需的存储空间（ LHS ）（ 1120 ）; 和

新型的每个成员出现作为值表的索引的表示法所需的存储空间（ RHS ）（ 1122 ）.

类预装器 132 按下式计算步骤 1120 中的 LHS 值:

LHS = （传统表示法的大小）× 出现的次数

= （传统表示法的大小）× M 位。

类预装器 132 按下式计算步骤 1122 中的 RHS 值:

RHS = （成员值的大小）× 出现的次数 + 值表的大小

= （成员值的大小）× M + M × （ $\lfloor \log_2(N) \rfloor + 1$ ）位。

如果 RHS 小于 LHS （ 1124-Y ），类预装器 132 表示成员类型为值表和索引（ 1126 ）。如果 RHS 不小于 LHS （ 1124-N ），类预装器 132 将传统地表示成员类型（ 1128 ）。

类预装器重复步骤 1120、 1122、 1124、 1126、 1128，同时，其它成员保留以待处理(1124-N)。

一旦每个成员已经被处理（ 1124-Y ），类预装器 132 执行数据结构声明产生过程 1130 。在该过程中，对于每个数据结构，类预装器 132 确定数据结构成员的最佳次序（ 1132 ）。进程次序及其考虑因素已经参看图 9A 描述了。类预装器 132 然后根据最佳次序产生成员首部信息 614 和值表 616 （ 1134 ）。首部信息 614 和值表 616 的产生已经参看图 7A 和 7B 描述。

参看图 11，预装的可执行模块和引导时间启动程序 1320 永久地存储在执行引擎计算机的只读存储器中。每次执行引擎计算机被通电或重新引导时，引导时间启动程序 1320 自动执行。在其它的任务中，引导时间启动程序在执行中，拷贝必须驻留在随机访问存储器中的所有方法和数据，到由链接器指定给它们的随机访问存储器位置。

尽管这里描述的方法和系统是参照 Java 程序语言描述的，本发明适用于使用利用类的动态运行时间装载的其它面向对象类的计算机系统。

另外，本发明适于在除了诸如随机访问存储器的存储装置之外的各种类型的可执行媒体上执行。其它类型的可执行媒体可以被使用，诸如，但不限于，可读计算机存储媒体，可以是任何存储装置、光盘或软盘。

前述方法和系统已经就执行类属的 Java 应用程序进行了描述，它适用于任何 Java 应用程序。例如，本发明可被用于趋向运行在手持计算机上的 Java 编码的个人信息管理器使用的预装类。另外， Java 应用程序可不必运行在分布的环境中，它可以运行于在执行引擎或服务器计算机中执行的单机模式下，而无需从外部系统输入新类。

尽管本发明已经参考几个特定实施例进行了描述，但该说明是本发明的示例，不是想要解释为限制本发明。在不脱离本发明的精神和范围的情况下，本领域的技术人员可以进行各种修正，本发明以所附权利要求及其等效物的全部范围来限定。

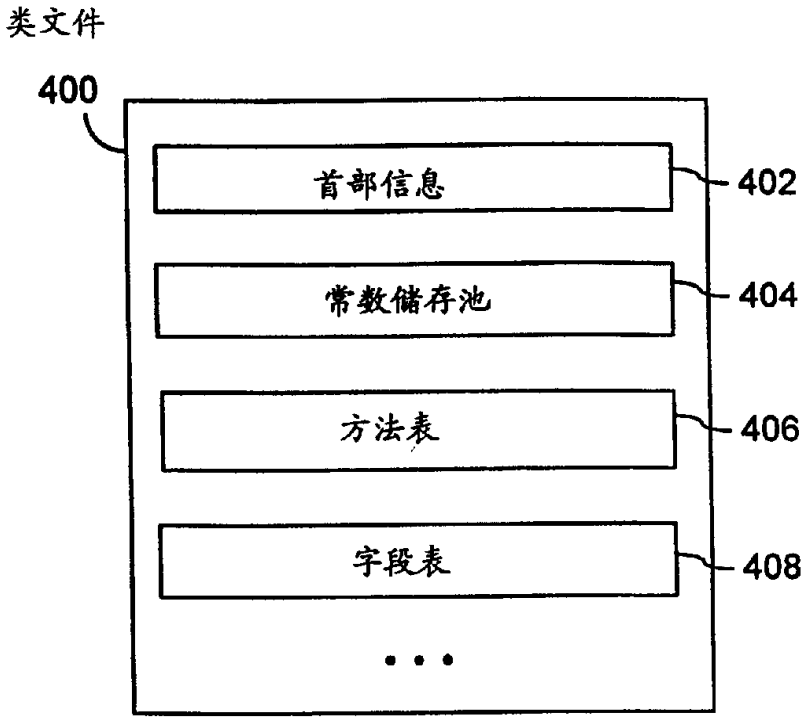


图 1

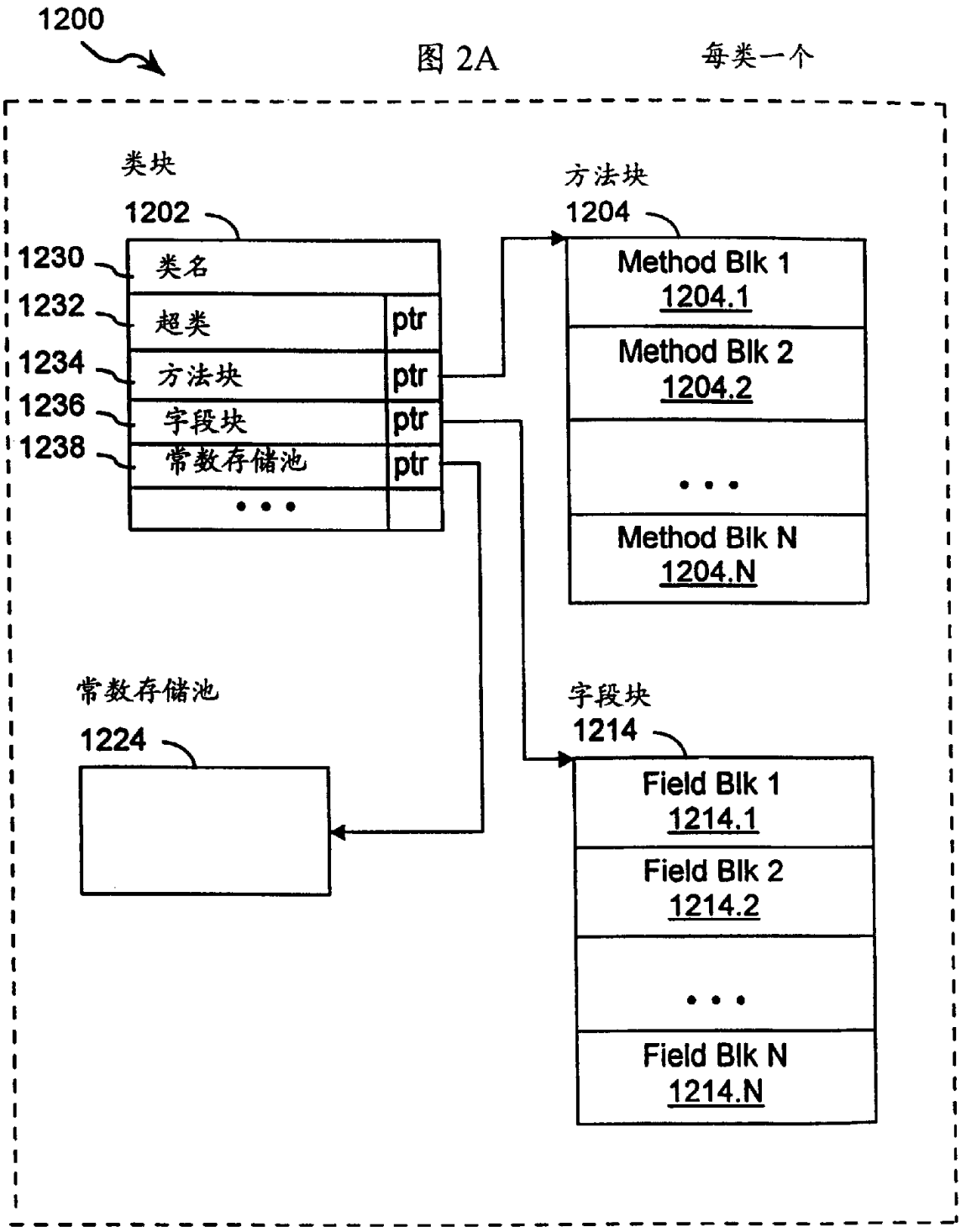
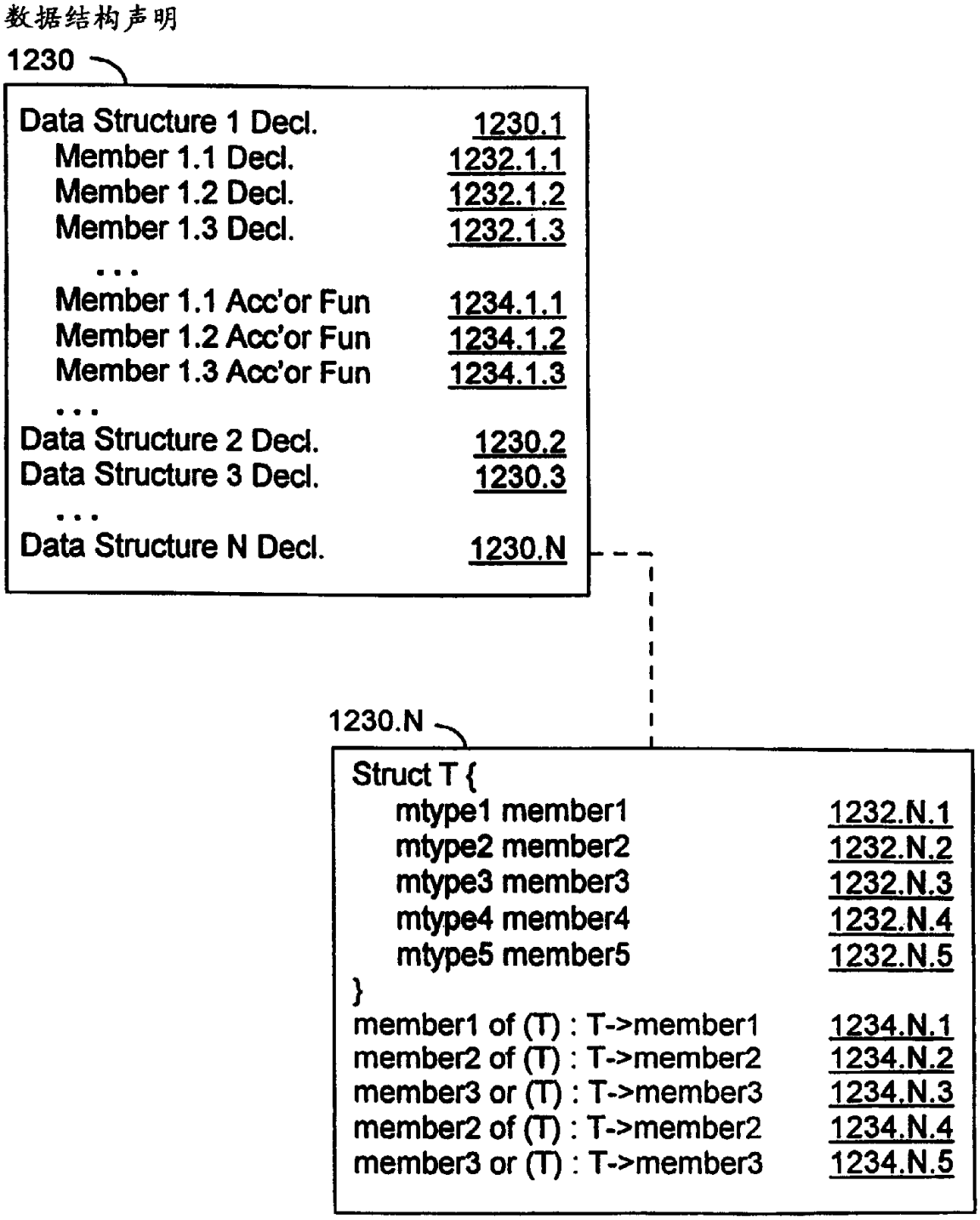

1200
图 2A
每类一个
类块
1202
1230
类名
1232
超类
ptr
1234
方法块
ptr
1236
字段块
ptr
1238
常数存储池
ptr
• • •
方法块
1204
Method Blk 1
1204.1
Method Blk 2
1204.2
• • •
Method Blk N
1204.N
常数存储池
1224
字段块
1214
Field Blk 1
1214.1
Field Blk 2
1214.2
• • •
Field Blk N
1214.N

图 2B


数据结构声明
1230
Data Structure 1 Decl. 1230.1
Member 1.1 Decl. 1232.1.1
Member 1.2 Decl. 1232.1.2
Member 1.3 Decl. 1232.1.3
...
Member 1.1 Acc'or Fun 1234.1.1
Member 1.2 Acc'or Fun 1234.1.2
Member 1.3 Acc'or Fun 1234.1.3
...
Data Structure 2 Decl. 1230.2
Data Structure 3 Decl. 1230.3
...
Data Structure N Decl. 1230.N
1230.N
Struct T {
mtype1 member1 1232.N.1
mtype2 member2 1232.N.2
mtype3 member3 1232.N.3
mtype4 member4 1232.N.4
mtype5 member5 1232.N.5
}
member1 of (T) : T->member1 1234.N.1
member2 of (T) : T->member2 1234.N.2
member3 or (T) : T->member3 1234.N.3
member2 of (T) : T->member2 1234.N.4
member3 or (T) : T->member3 1234.N.5

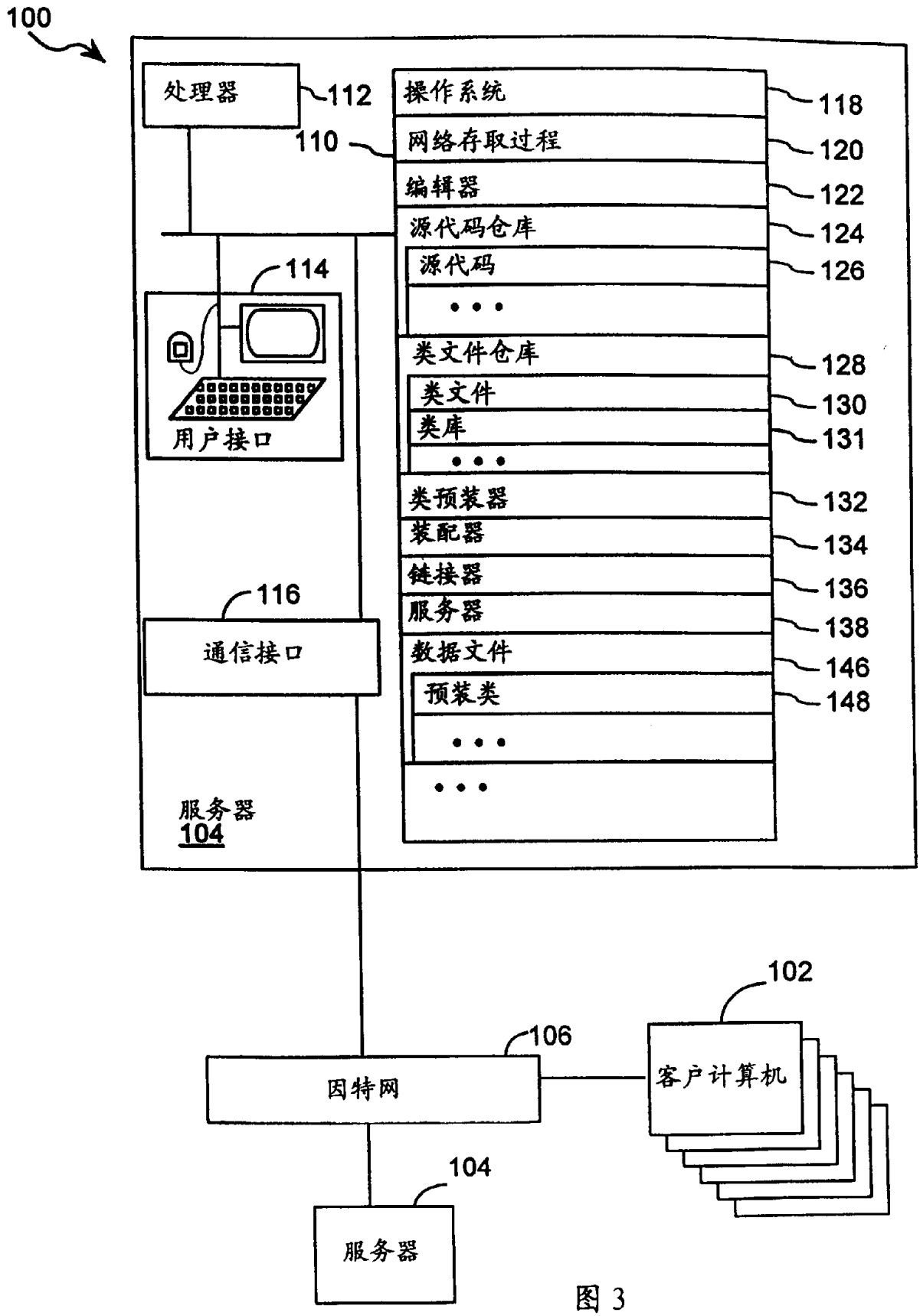

100
处理器
112
操作系统
118
网络存取过程
110
120
编辑器
122
源代码仓库
124
源代码
126
114
类文件仓库
128
类文件
130
类库
131
用户接口
类预装器
132
装配器
134
链接器
136
服务器
138
116
通信接口
数据文件
146
预装类
148
服务器
104
102
106
因特网
客户计算机
104
服务器

图 3

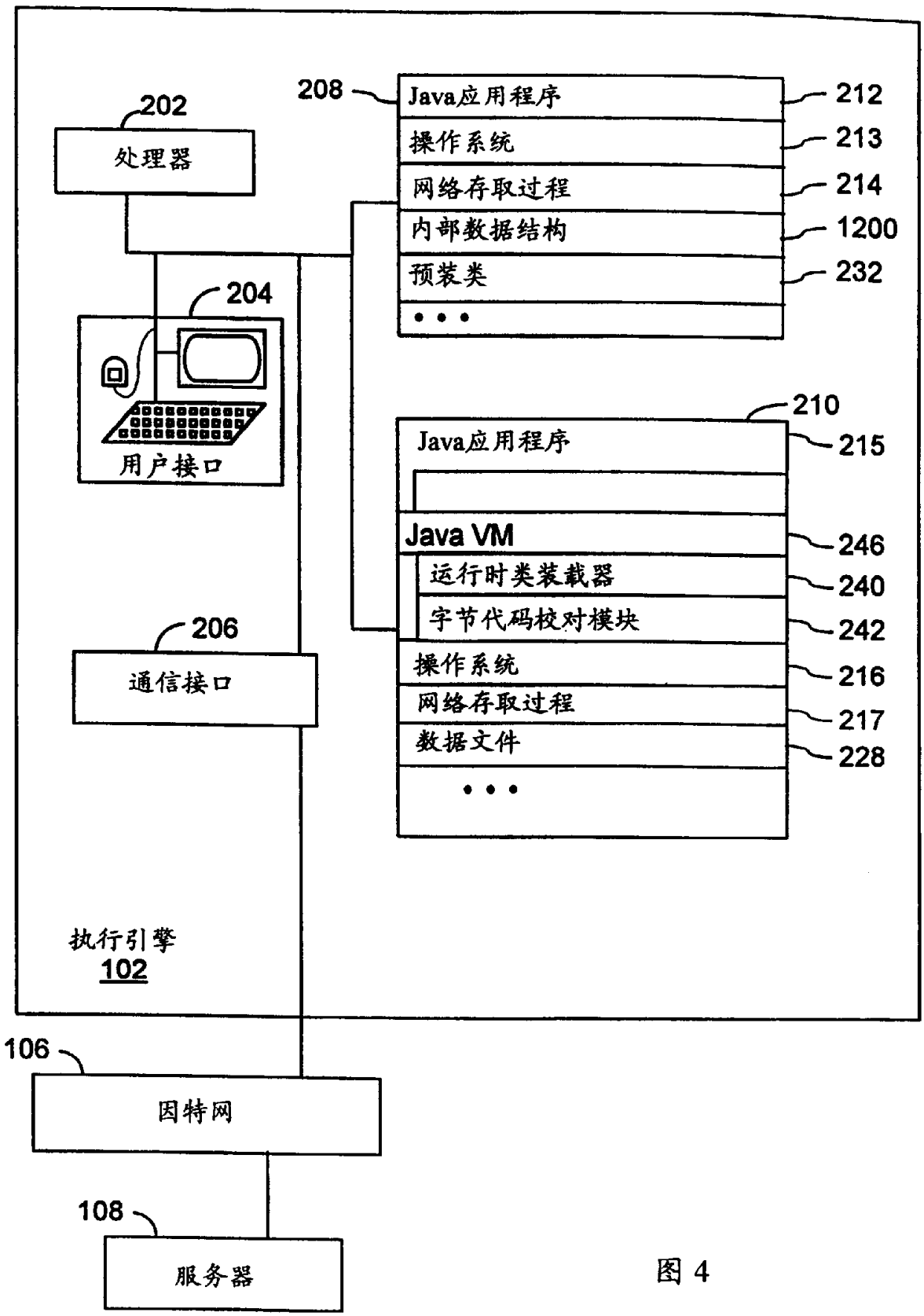

208
Java应用程序
212
操作系统
213
网络存取过程
214
内部数据结构
1200
预装类
232
202
处理器
204
用户接口
210
Java应用程序
215
Java VM
246
运行时类装载器
240
字节代码校对模块
242
206
通信接口
操作系统
216
网络存取过程
217
数据文件
228
执行引擎
102
106
因特网
108
服务器

图 4

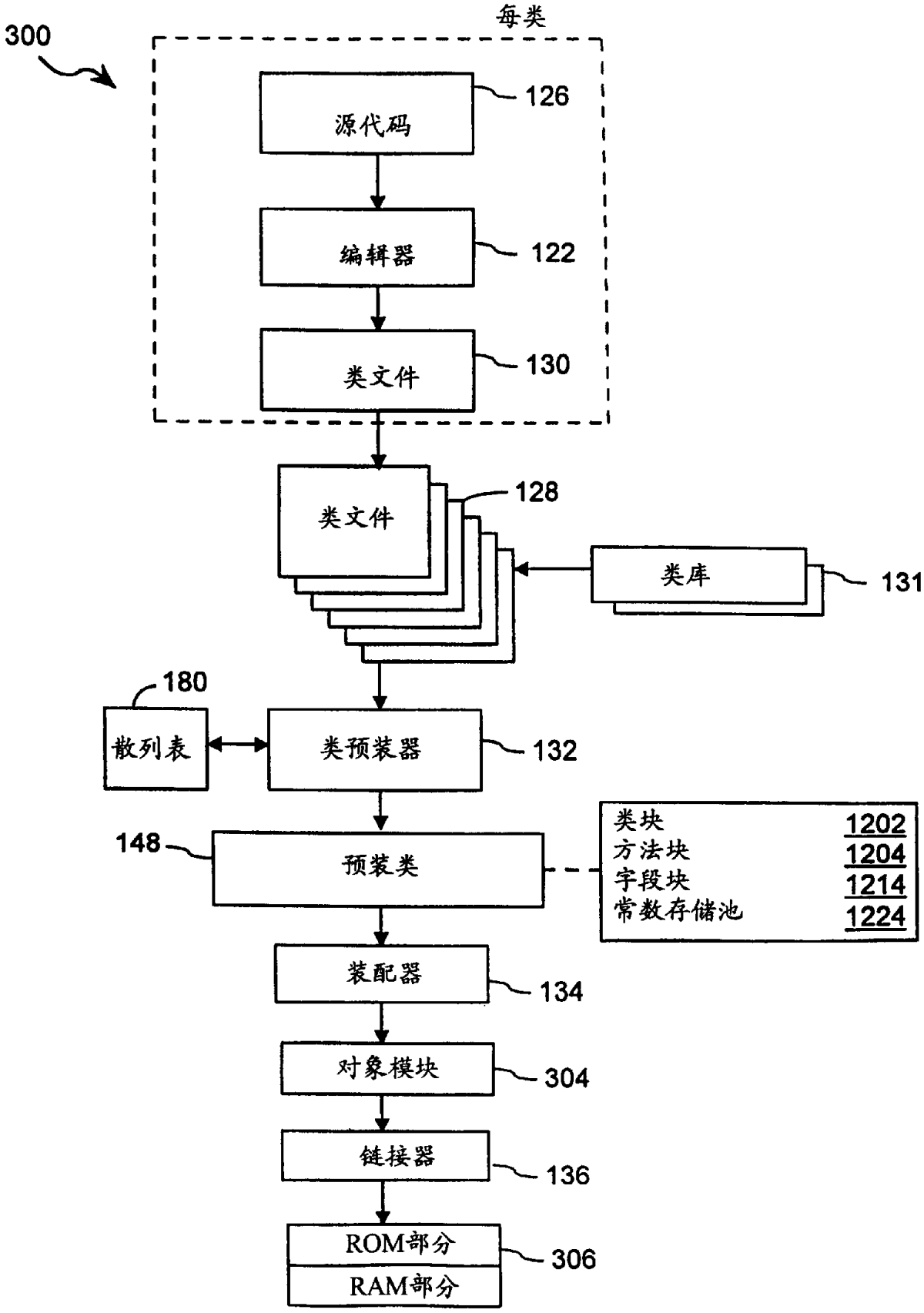

300
每类
源代码
126
编辑器
122
类文件
130
类文件
128
类库
131
180
散列表
类预装器
132
148
预装类
类块 1202
方法块 1204
字段块 1214
常数存储池 1224
装配器
134
对象模块
304
链接器
136
ROM部分
RAM部分
306
预装的可执行模块

图 5

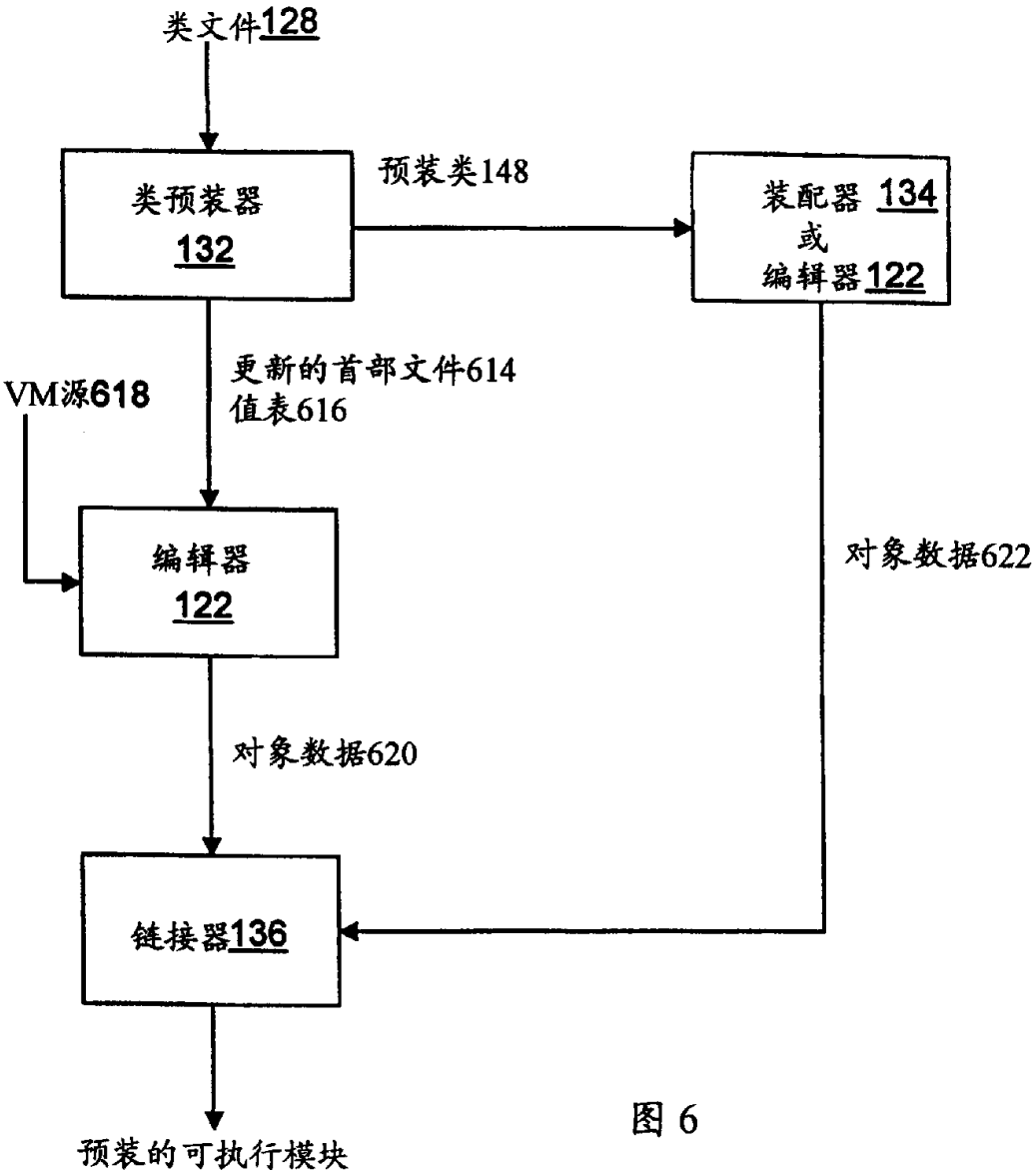

类文件128
类预装器
132
预装类148
装配器 134
或
编辑器122
更新的首部文件614
值表616
VM源618
编辑器
122
对象数据622
对象数据620
链接器136
预装的可执行模块

图 6

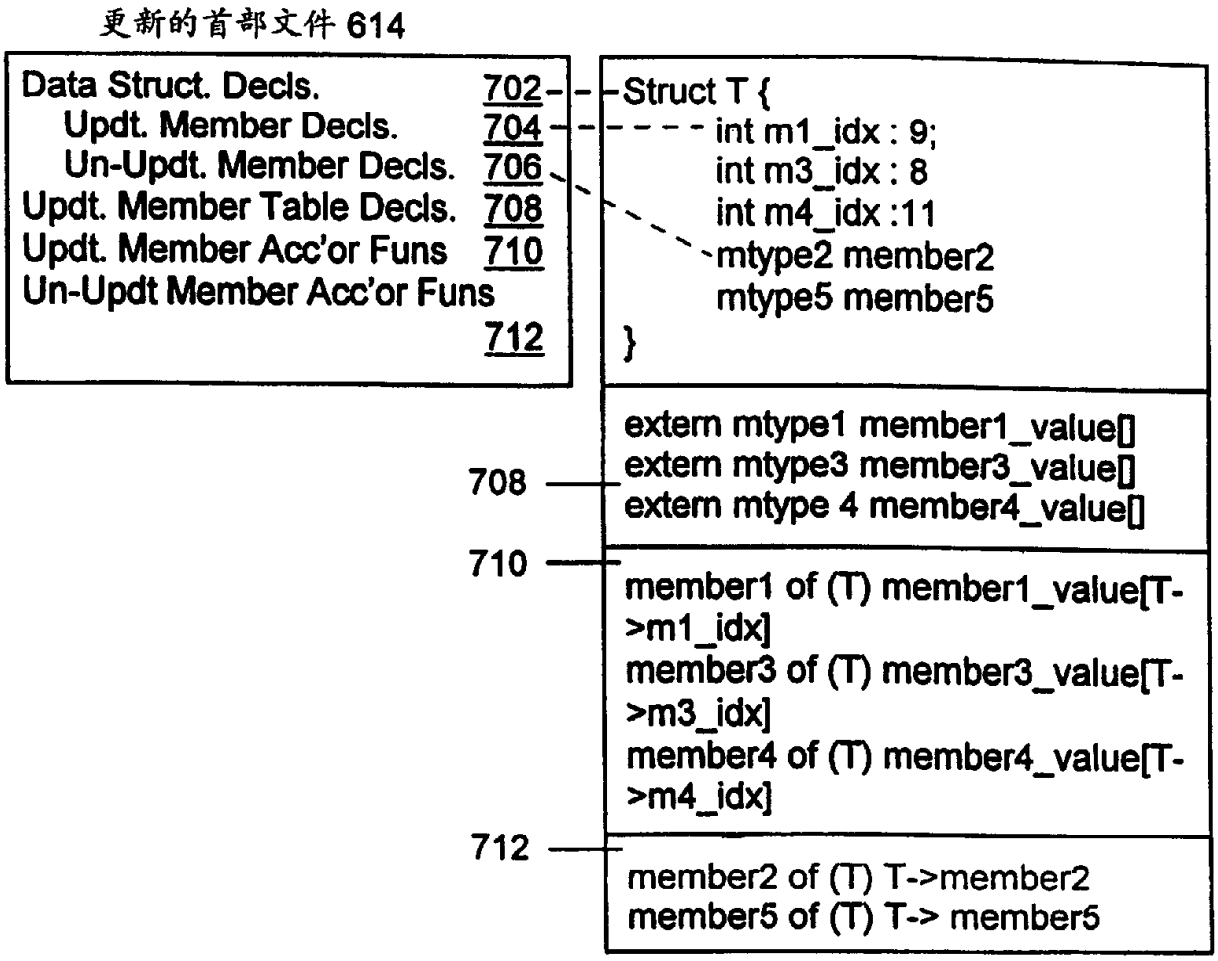
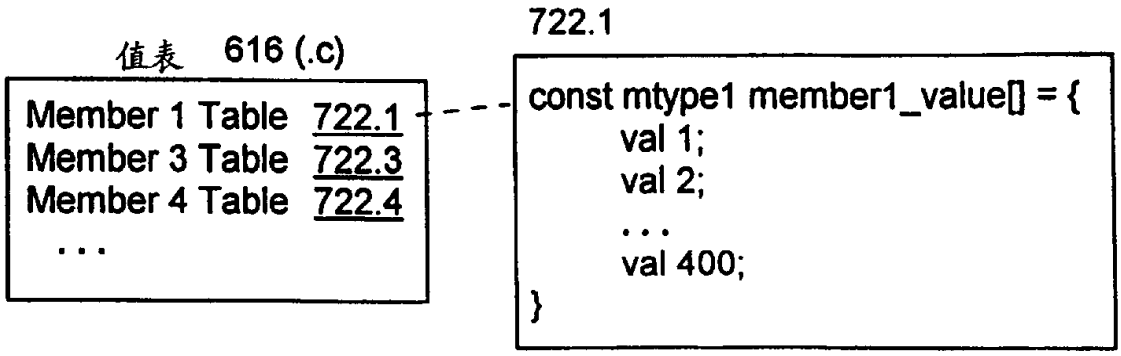

更新的首部文件 614
Data Struct. Decls. 702
Updt. Member Decls. 704
Un-Updt. Member Decls. 706
Updt. Member Table Decls. 708
Updt. Member Acc'or Funs 710
Un-Updt Member Acc'or Funs 712
Struct T {
int m1_idx : 9;
int m3_idx : 8
int m4_idx :11
mtype2 member2
mtype5 member5
}
708
extern mtype1 member1_value[]
extern mtype3 member3_value[]
extern mtype 4 member4_value[]
710
member1 of (T) member1_value[T->m1_idx]
member3 of (T) member3_value[T->m3_idx]
member4 of (T) member4_value[T->m4_idx]
712
member2 of (T) T->member2
member5 of (T) T-> member5

图 7A

图 7B


值表 616 (.c)
Member 1 Table 722.1
Member 3 Table 722.3
Member 4 Table 722.4
...
722.1
const mtype1 member1_value[] = {
val 1;
val 2;
...
val 400;
}

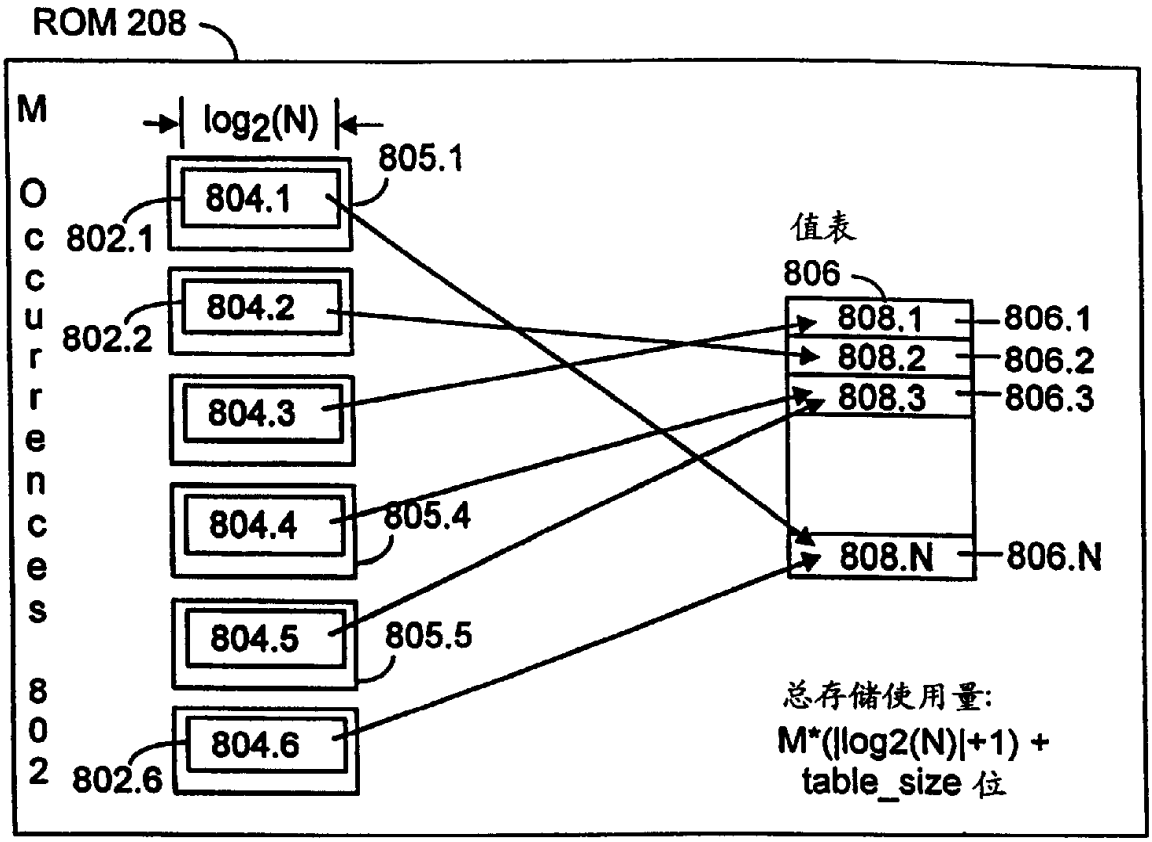

ROM 208
M
Occurrences 802
log2(N)
804.1
804.2
804.3
804.4
804.5
804.6
802.1
802.2
802.6
805.1
805.4
805.5
值表
806
808.1
808.2
808.3
808.N
806.1
806.2
806.3
806.N
总存储使用量:
M*(|log2(N)|+1) +
table_size 位

图 8A

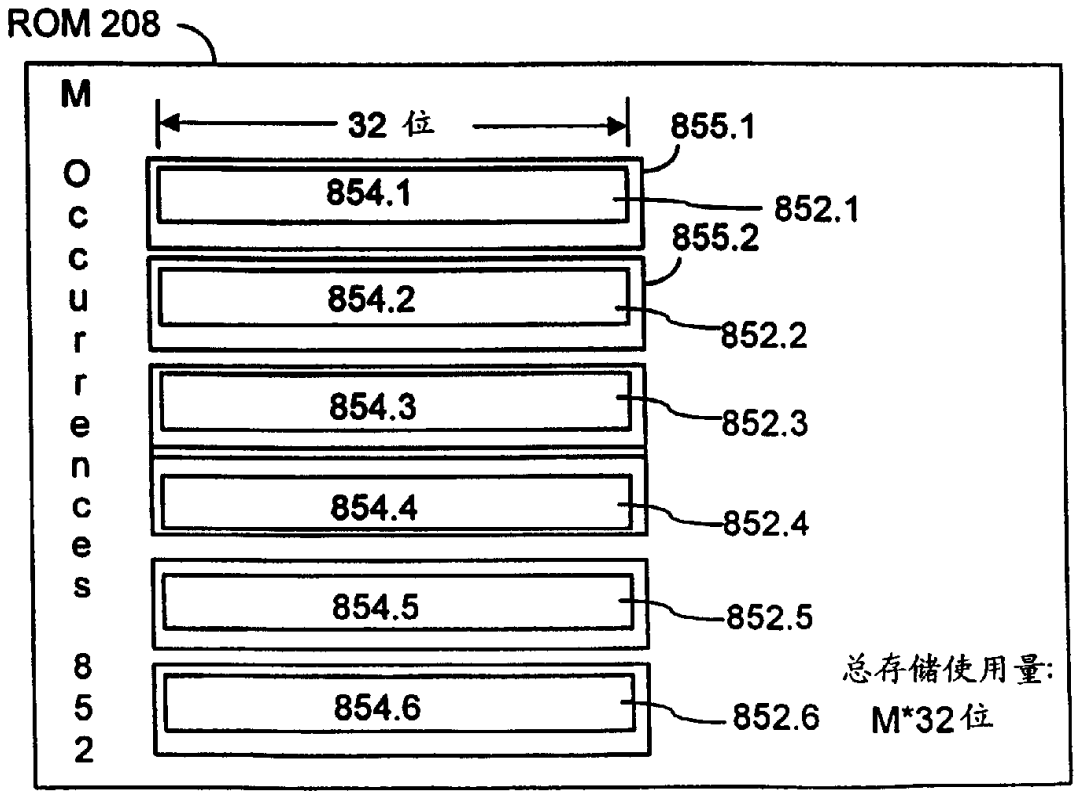

ROM 208
M
Occurrences 852
32 位
854.1
854.2
854.3
854.4
854.5
854.6
855.1
855.2
852.1
852.2
852.3
852.4
852.5
852.6
总存储使用量:
M*32位

图 8B

本发明中数据结构T902的一次
出现的数据结构存储使用量

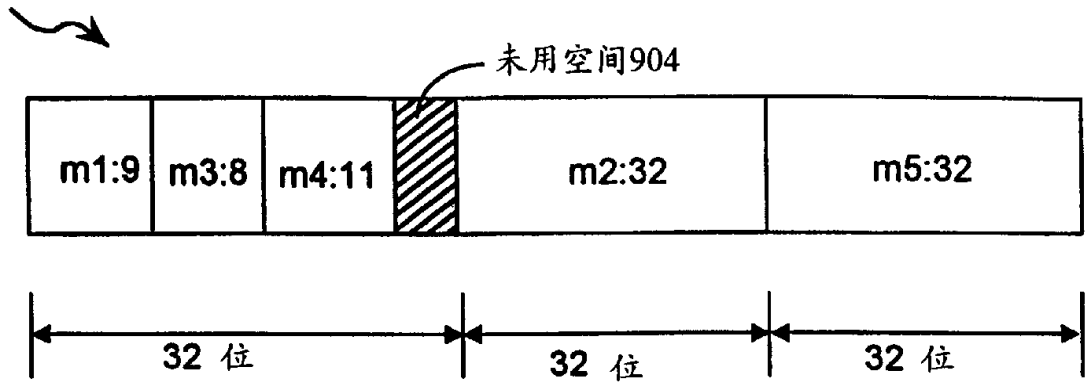


图 9B

现有技术中数据结构T906的一次
出现的数据结构存储使用量

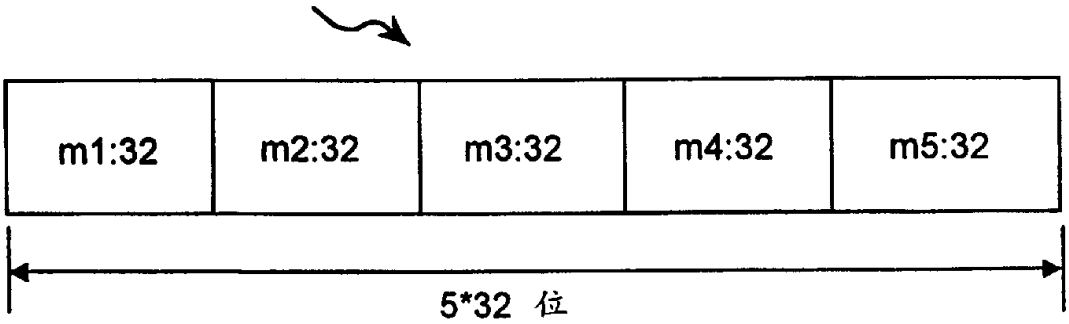


图 9B

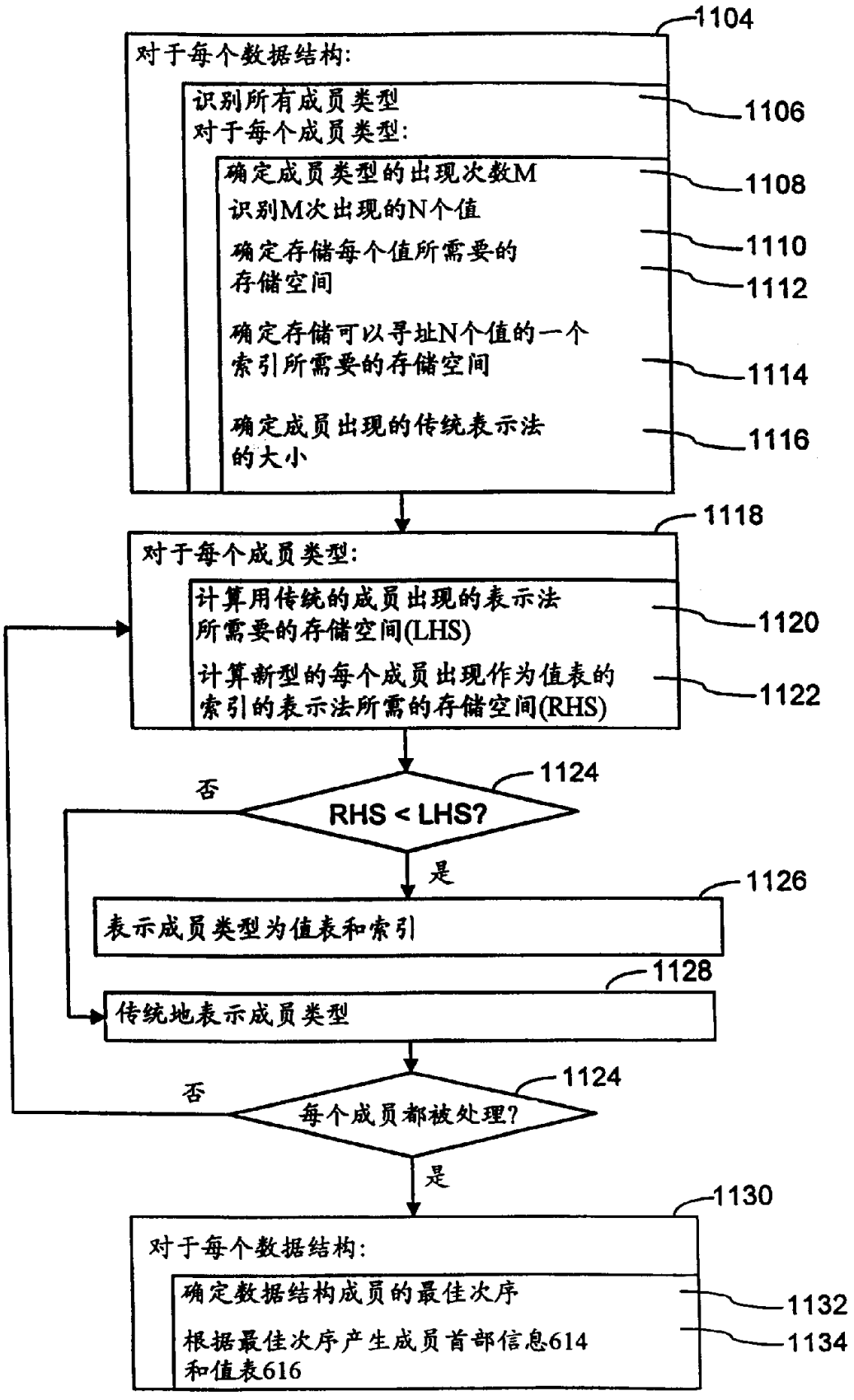

1104
对于每个数据结构:
识别所有成员类型
对于每个成员类型:
1106
确定成员类型的出现次数M
识别M次出现的N个值
1108
确定存储每个值所需要的
存储空间
1110
1112
确定存储可以寻址N个值的一个
索引所需要的存储空间
1114
确定成员出现的传统表示法
的大小
1116
1118
对于每个成员类型:
计算用传统的成员出现的表示法
所需要的存储空间(LHS)
1120
计算新型的每个成员出现作为值表的
索引的表示法所需的存储空间(RHS)
1122
否
1124
RHS < LHS?
是
1126
表示成员类型为值表和索引
1128
传统地表示成员类型
否
1124
每个成员都被处理?
是
1130
对于每个数据结构:
确定数据结构成员的最佳次序
1132
根据最佳次序产生成员首部信息614
和值表616
1134

图 10

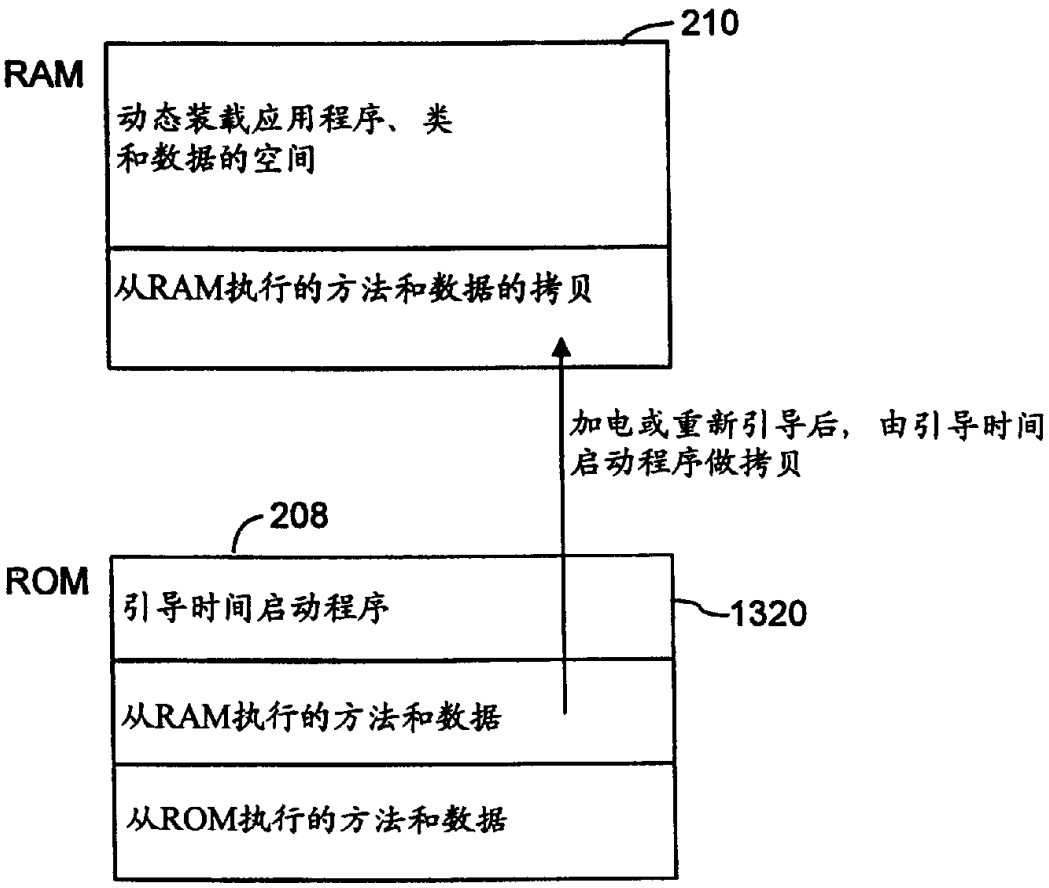

210
RAM
动态装载应用程序、类
和数据的空间
从RAM执行的方法和数据的拷贝
加电或重新引导后，由引导时间
启动程序做拷贝
208
ROM
引导时间启动程序
1320
从RAM执行的方法和数据
从ROM执行的方法和数据

图 11