[19]中华人民共和国专利局

[51]Int.Cl[6]
G06F 12/00

[12] 发明专利申请公开说明书

[21] 申请号 97112905.3

[43]公开日 1998 年 2 月 4 日

[11] 公开号 CN 1172303A

[22]申请日 97.5.29

[30]优先权

[32]96.5.30 [33]US[31]655,474

[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 西伦·D·托克

[74]专利代理机构 柳沈知识产权律师事务所

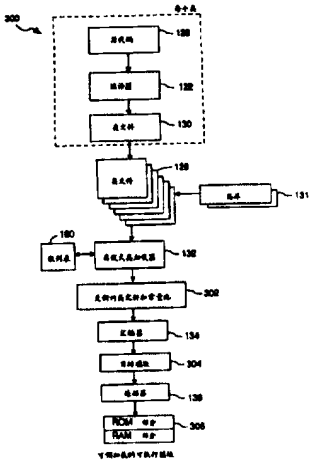
代理人 马 莹

权利要求书 2 页 说明书 12 页 附图页数 12 页

[54]发明名称 一种在只读存储器中加载类的方法及系统

[57]摘要

提供可执行模块的方法和系统，该模块有两个地址空间，分别驻留在 ROM 和 RAM 中。该可执行模块表示为动态类加载而构造的 Java 类。静态类加载器修改类结构使其适合静态加载，它还鉴别包含未解决的符号引用的方法和在模块执行过程中变化的数据，以便将其置于 RAM 中的地址空间。静态类加载器对于分布式计算环境中很少或没有辅助存储器的客户机很有利，因为通过使用 ROM 存储静态可加载类，可腾出 RAM 用于其它用途。



权　利　要　求　书

1. 一种操作一台分布式计算机系统中的计算机的方法，所述方法包括如下步骤:

在存储器中存储多个类，每个类包括数据和至少一个方法，每个方法包括多个字节码，所述字节码的一个子集包括对所述计算机可存取的方法的符号引用;

如果所述类数据可被所述类方法之一修改，则对于每个所述类标记所述类数据;

将每个对存储在所述存储器中的一指定方法的所述符号引用，用所述指定方法的存储位置来替换;

如果所述方法的字节码之一包括对一个不存储在所述存储器中的方法的符号引用，则对于每个所述类的每个所述方法，标记所述方法；以及

提供一个包括每个所述类的可执行模块，在执行所述可执行模块时，每个未标记的方法和每个未标记的数据将被存入只读存储介质中，并且在执行所述可执行模块时，每个所述标记的方法和每个所述标记的数据将被存入可读可写的存储介质中。

2. 一种操作一个计算机系统的方法，所述方法包括如下步骤:

在只读存储器设备中存储被分成第一子模块和第二子模块的浏览器模块，每个子模块包括多个指令和数据，所述第二子模块有一个所述指令的子集和一个所述数据的子集，所述指令子集和数据子集包括在所述多个指令执行过程中可修改的指令和数据;

在计算机系统启动时，将第二子模块存储到随机存取存储器设备中；以及

执行浏览器模块以输入信息内容数据和计算机可执行模块，所述输入数据和所述计算机可执行模块存储在随机存取存储器设备中。

3. 在一个具有用通信链路连接起来的多个计算机的分布式数据处理系统中，所述计算机之一包括:

存储多个类的存储器，每个所述类包括多个数据和至少一个方法，每个所述方法包括多个字节码，第一组所述字节码引用存储在存储器中方法，第二组所述字节码引用可通过所述通信链路访问的方法;

离线式类加载器，当类数据可以被所述类方法之一修改时，对每个类标记所述类数据，在所述方法的字节码之一包括对不存储在所述存储器中的方法的符号引用时，对每个所述类的每个所述方法，标记所述方法；

产生可执行模块的连接器，具有第一部分和第二部分，第一部分包括每个被所述离线式类加载器标记的所述类方法和所述数据，而第二部分包括每个不在所述第一部分的所述类方法和所述数据。

4. 如权利要求 3 所述的计算机，其中

当可执行模块被执行时，所述可执行模块的所述第一部分被设置为存入可读可写的介质，当所述可执行模块被执行时，所述第二部分被设置为存入只读存储介质。

说　明　书

一种在只读存储器中
加载类的方法及系统

本发明涉及具有运行时间动态加载的类的面向对象的计算机系统，具体地说，涉及一种在只读存储器中预加载类的一个子集的系统及方法。

当前面向对象的编程语言的一个趋势是扩展语言的功能，以适应在分布式计算环境中动态内容的分布。在一种这样的语言中，它是通过运行时间动态加载类来实现的。类是一些变量与方法的集合，它们是一个对象行为的模型。通过运行时间动态加载类，已有的应用程序可以通过连接入新类来增加功能，这些新类驻留在分布式计算环境中的任何计算机系统中。

在这种语言中，符号引用(symbolic references)用于指代类成员(如类的方法与类变量)。当一个类被调用时，动态加载器确定类的存储方案并解决符号引用。当访问不断被更新的类时，这样的加载方案是有益的。然而，这种加载方案的一个局限性是它依赖于读/写存储器设备，如随机存取存储器(RAM)。在一种有很少或没有辅助存储器(如非易失性磁盘存储器)的计算环境中，如此动态加载类会很快用尽 RAM 的存储容量。由于 RAM 的容量是有限的，尽量减少应用程序使用 RAM 的量是需要的。因此，在执行含有可动态加载的类的面向对象的程序代码时，需要限制其所用的 RAM 的量。

提供一种克服现有技术此缺陷的方法及系统将是有益的。

简言之，本公开属于一种离线式(offline)类加载器，它用于产生一个可执行模块，该模块的类被预加载入存储器中而不需要运行时间动态加载。尽管如此，该可执行模块仍然包括一个专用于运行时间动态加载的类结构。因此，这种离线式类加载器修改现有类结构以适应静态加载。然而，这种类结构允许改变包含未解决的引用的数据和方法。该离线式类加载器给这些方法和数据打上标记，指明它们将要被存入随机存取存储器中。其他数据都存储在只读存储器中。当静态加载过程完成后，就产生一个包含两个地址空间的可预加载的可执行模块。第一地址空间包括含有未解决的引用的方法，和在模块执行过程中变化的数据　该第一地址空间被加载到随机存取存储器。第二地址空间包括含有被静态加载的类的方法和被加载到只读存储器的常数。

在客户机有很少或没有辅助存储器的分布式计算机系统中，这种可预加载的可执行模块是有益的。这样的客户机需要完全在随机存取存储器中运行应用程序，随机存储器很快就成为有限的资源。通过采用这种离线式类加载器将应用程序分成两个地址空间，可预加载模块使用的 RAM 数量就减至最少。在一个实施例中，一台具有最低限度辅助存储器的客户机采用离线式类加载器在客户机的只读存储器中预加载了一个浏览器。该浏览器被分入前述的两个地址空间。当系统初始化或上电时，该浏览器的随机存取存储器部分就从只读存储器加载到随机存取存储器。通过在只读存储器执行浏览器的一大部分，浏览器就拥有了更多的 RAM 用来存储信息内容和可执行模块，它们是从与客户机通讯的其它服务器计算机上获得的。

从以下的详细描述和所附权利要求并结合附图，本发明的其他目的和特征会更加明白，附图为:

图 1 是分布式计算机系统的方框图。

图 2 是图 1 所示分布式计算机系统中客户机的方框图。

图 3 是说明处理部件的流程图，该处理部件用于产生可预加载的可执行模块。

图 4 是说明类文件的文件构造。

图 5 是说明常量池(constant pool)的文件构造。

图 6 说明类块数据结构。

图 7 说明一个指令字节码流。

图 8A 和 8B 是离线式类加载器所用方法的流程图。

图 9 是建立类块数据结构的方法的流程图。

图 10 是删除重复常量的方法的流程图。

图 11 是将非快速(non － quick)指令格式转换为快速指令格式的方法的流程图。

图 12 为一方框图，说明将应用程序预加载到只读存储器和随机存取存储器的映射关系，并说明被静态类初始化器映射到随机存取存储器中的方法及数据部分的加载过程。

这里描述的方法与系统采用分布式计算环境，该分布式计算环境具有连接至少一台服务器计算机和一些客户机的通信链路。有些客户机具有很少或没有辅助存储器(如非易失性磁盘存储器)，因此要求应用程序完全在随机存

取存储器中运行。一种用 Java 编程语言开发的应用程序在这样一个客户机上执行。最好，该应用程序是这样一个浏览器，要从一个或多个服务器计算机上输入 Java 内容，如 Java 小应用程序(applet)。通常该浏览器是一个解释程序模块，它采用超文本传输协议(HTTP)检索网络文档，以便从作为网址的服务器存取一个或多个网页，这些网页的格式为超文本标记语言(HTML)。HTML 文档经解释后显示给客户机端的用户。通常，HTML 文档嵌入小应用程序中。小应用程序是一个可执行模块，表示为一个 Java 类。浏览器将该小应用程序及相关类载入，以执行此小应用程序。

浏览器、HTML 文档及小应用程序都驻留在计算机的 RAM 中。有时，载入 RAM 中的数据是会超过其容量。由于客户机可能没有辅助存储器，将浏览器和其它基本支持类的一部分置入只读存储器是有益的。这样，RAM 存储器特地为输入的小应用程序而保留。最好，浏览器和其它基本支持类(如 I/O 和实用类)被预加载到只读存储器。

离线式类加载器将 Java 应用程序，如浏览器及基本支持类，分成至少两个分离的地址空间。第一地址空间驻留在只读存储器中，包括不需要动态加载的方法和保持不变的数据。第二地址空间驻留在读/写存储器(如随机存取存储器)中，包括需要动态加载的方法和在执行过程中变化的数据。

这样划分的浏览器可初始存储在客户机的只读存储器中。系统上电时，第二地址空间被预加载到 RAM 。这样会留出大量 RAM 存储器以供浏览器输入 HTML 文档、小应用程序、其它信息前后文(information - context)以及可执行模块，它们都可通过通信链路存取。

应当指出，本公开是参考 Java 编程语言进行描述的。因此，本描述将采用 Java 中的术语。下述 Java 术语在全篇描述中将会频繁使用，在此作简要说明。类是用于描述对象行为的实例(instance)变量和方法的组合。对象是类的一个实例。实例变量是对象的数据，它由类实例化而来。静态实例变量对于类的所有实例都是相同的。非静态实例变量对类的每个实例都不同。常量数据指那些在程序执行过程中不改变的数据。

方法是执行一系列含义明确的操作的程序段。在 Java 中，方法是用字节码流表示的指令实现的。字节码是一个 8 位的代码，它可以是指令的一部分，如一个 8 位的操作数或操作码。接口是一个抽象类，其中实现方法的字节码在运行时间定义。Java 应用程序是包含字节码的可执行模块，它可以采用

Java 解释器或 Java 及时(just - in - time)编译器执行。 Java 编程语言的特色在 Tim Ritchey 所著的 Programming with Java Beta 2.0, New Riders Publishing (1995)一书中有详细描述。

参照图 1，所示分布式计算机系统 100，有多个客户机 102 和多个服务器计算机 104。在一个实施例中，每台客户机 102 通过 Internet 106 与服务器 104 连接，当然也可采用其它形式的通信连接。最好，服务器和客户计算机可以是桌面计算机，如 Sun 工作站、 IBM 兼容机及 Macintosh 机，然而，实际上任何类型的计算机均可作为服务器或客户机。而且，该系统并不限于分布式计算机系统。它可以不按这里说明的细节实现，可以实现为不同的计算机系统、不同结构的紧耦合处理器或不同结构的松耦合微处理器系统。

在一个实施例中，一个或多个服务器计算机作为网址包括一个 HTML 文档库，其中包含 Java 内容和小应用程序。客户机执行浏览器，该浏览器为客户机端的用户提供从服务器计算机存取 HTML 文档的机会。参照图 1，服务器计算机通常包括一个或多个处理器 112、通讯接口 116、用户接口 114 及存储器 110。存储器 110 存储:

· 操作系统 118;

· Internet 通信管理器程序或其它类型的网络访问程序 120;

· 编译器 122，用于将 Java 编程语言写的源代码翻译成字节码流;

· 源代码库 124，包括一个或多个包含 Java 源代码的源代码文件;

· 类文件库 128，包括一个或多个类文件 130、及包含类文件的一个或多个类库 131，每个类文件包含代表特定类的数据;

· 离线式类加载器 132，用于预加载特定的一组类；离线式类加载器也可称为静态类加载器;

· 汇编器 134，产生目标文件，以连接器可识别的格式表示类成员、类数据结构及存储器存储指示器;

· 连接器 136，为一系列预加载的类确定存储器布局，以及解决所有符号引用;

· 浏览器 138，用于存取 HTML 文档;

· 一个或多个数据文件，供服务器使用。

浏览器可以包括:

· 运行时间类加载器模块 140，将类加载入用户地址空间，并使用字节

码程序校验器校验与每个被加载类相关的方法的完整性;

· 字节码程序校验器模块 142，校验一个指定程序是否满足某些预定的完整性准则；和

· HTML 加载器 144，用于加载 HTML 文档;

以及其它模块。

图 2 所示为客户机，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 和随机存取存储器 210。只读存储器 208 存储浏览器的一部分 212 和支持程序(包括操作系统 213 和网络访问程序 214)，该支持程序中的方法不含未解决的引用，且该支持程序中的数据保持不变。

随机存取存储器 210 存储:

· 浏览器的第二部分 215 和支持程序 216、217，包括含有未解决的引用的应用程序和执行过程中变化的数据;

· HTML 文档库 220，包括一个或多个 HTML 文档 222，这些文档是应用户请求通过用户接口 204 由浏览器获取的;

· 类文件库 224，包括一个或多个类文件或小应用程序 226；以及

· 一个或多个数据文件 228，客户机在其处理中可能会用到这些数据文件。

图 3 是产生可预加载的可执行模块所用步骤的顺序的概况图。应该指出，这里描述的方法和系统适合预加载浏览器和其它支持程序。然而，这里描述的方法和系统并不限于此处特定的 Java 应用程序。任何 Java 应用程序或其它运行时间连接的方法集都可用这里描述的方法和系统预加载。

每个包含 Java 应用程序的类的源代码 126 被编译器 122 编译成类文件 130。类文件包含代表类的类数据结构、每个方法的字节码、常量数据及其它信息。对类文件更详细的说明见后。另一种方法是，与应用程序对应的类文件可以已经驻留在一个或多个类库 131 中。一旦类文件可以使用，组成将被预加载的应用程序的整套类文件 128 就被传送给离线式类加载器 132。

离线式类加载器 132 的作用是确定与每个类相关的哪些方法和变量能存入只读存储器，哪些必须存入随机存取存储器。调用 Java 接口或使用非静态实例变量的方法需驻留在随机存取存储器中。这是因为实现接口的字节码是运行时间确定的，且非静态实例变量对于每个相关类的实例化是变化的。离线式类加载器 132 找到这些方法和变量，并插入一个特殊的指示器标记出它

们将被加载到随机存取存储器。离线式类加载器还进行一系列的优化，以产生更紧凑的可执行代码。例如，与每个类相关的常量池被结合起来供所有驻留在应用程序中的类使用。另外，离线式类加载器还进行附加的处理，以便将原来为动态加载构造的类文件改为供预加载的类环境使用。

离线式类加载器的输出 302 由两个文件组成：一个是包含整个应用程序的常量数据的常量池文件；另一个是更新的类文件，包含类数据结构和类成员。这两个文件中的数据的格式均为数据定义，其中每个定义指定一个字节码和一个指示存储器位置的偏移量。更新的类文件包含存储器指示器，它指明特定的一组定节码存入哪种类型的存储器。然而，此处描述的方法和系统并不限于产生这两个文件。也可采用其它文件结构，包括但不限于一个包含所有相关类数据的文件。

接着，该文件传送给汇编器 134，它产生一个目标模块，该模块具有连接器将数据映射到相应地址空间所需的格式。最好有两个地址空间，一个给随机存取存储器，另一个给只读存储器。接着目标模块传送给连接器 136，它给应用程序中的类生成存储器布局。一旦确定了存储器布局，连接器 136 将解决所有的符号引用，并以直接地址取代它们。该存储器布局被分成两个地址空间。被标记为只读存储器的方法和数据包含在第一地址空间，被标记为需存入随机存取存储器的方法和数据包含在第二地址空间。连接器 136 的输出是一个可预加载的可执行模块 306，包含这两个地址空间的方法和数据。

如上所述，离线式类加载器的主要功能是确定哪些方法和数据要被存入只读存储器及哪些要被存入随机存取存储器。另外，所有预加载的类的常量池最好结合起来，以将使用的只读存储器的量降至最少。为结合常量池，进行了某些优化以降低存储器的使用量。更具体地说，删除了重复的表达式，作为长字符串一部分的字符串被长字符串中相应子字符串位置的指针所取代。

包含所有类的通用常量池，被分成两段，第一段横跨 256 字节，第二段横跨 64K - 256 字节。第一段可以包括最多 256 个常量，第二段包括其余的常量。常量池的排序使得最常引用的常量存入池的第一段，最不常引用的常量存入第二段。一旦常量池被结合了，引用常量的字节码可能需要加以修改。第一段中的常量用 8 位操作数引用，而第二段中的常量用两个 8 位操作数引

用(见图 7，其中操作数 702 是一个 8 位操作数，操作数 704 和 706 共同组成了一个 16 位操作数)。从 8 位操作数扩展成 16 位操作数，需调整引用通用常量池第二段中的常量的字节码，同时考虑到这些方法中字节码改变的相对位置，调整方法中的字节码偏移量(如在分支指令中)。另外，字节码的改变要求更新存储在异常表中的偏移量，以反映方法中字节码起始和结束位置的改变，据此指定不同的异常处理程序。

进而，离线式类加载器执行另两个变换，以使类结构适合于预加载类。产生一个静态类初始化器，它对预加载的类进行类初始化。采用非快速指令格式用符号引用方法的字节码，重新编码为快速指令格式以直接引用该方法。

图 8 更为具体地描述了离线式类加载器 132 所采用的步骤。起始时离线式类加载器接收一个类文件，其中的每一个类都是应用程序的一部分，该应用程序的类将被预加载。图 4 说明该类文件的格式。该类文件包含一个或多个首标记录(header record)402 、一个常量池 404 、一个或多个方法 406 以及一个异常表 408 。首标记录 402 可以指明常量池的大小、方法的个数和异常表的大小。常量池 404 包括应用程序执行过程中保持不变的数据。这种数据的例子可包括字符串常量、静态最终整数(final integers)、对方法的引用、对类的引用以及对接口的引用。方法数据 406 由实现每个方法的字节码流组成，异常表中的每一项给出字节码起始和结束偏移量、异常类型、以及异常处理程序的偏移量。该项指明在起始和结束偏移量指明的代码中出现指明类型的异常时，该异常的处理程序可在给定处理程序的偏移量处找到。

每个类文件都被离线式类加载器读入(步骤 802)，并给每个类建立合适的类数据结构(804)，该类数据结构存入用于预处理应用程序的计算机的存储器中。图 6 说明类数据结构 600 。每个类都有一个类块 602 、一个或多个方法块 604 、每个方法的字节码 608 、一个或多个字段块 614 、用于各字段的各分离数据区 618 、常量池 624 、映射表 626 和异常表 628 。

类块是一个定长的数据结构，可包括下列数据:

· 类名 630 ;

· 指针 632 ，指向当前类的最近超类(immediate superclass)的类块;

· 一个或多个指针的数组 634 ，每个指针引用一个方法块;

· 一个或多个指针的数组 636 ，每个指针引用一个字段块;

· 指向类的常量池的指针 638；

· 指向类的异常表的指针 640 。

方法块 604 是一个定长的数据结构，包括确定数目的方法。分配一个或多个方法块以包含类的所有方法。方法块 604 包括方法名 612 和指向相应字节码 608 的指针 610 。

字段块 614 是一个定长的数据结构，包括实例变量或字段。 Java 为两种不同类型的变量提供不同的字段块格式。第一种格式 616 用于整型或浮点型实例变量。该格式 616 包含实例变量名、类型(如整型或浮点型)及其值。第二种格式 620 用于双精度型或长型实例变量。该格式 620 包含实例变量名、类型(如双精度型或长型)及指向实例变量值 618 位置的指针。

图 9 说明用于建立类数据结构的步骤。首标记录中的信息用于给每个类数据结构分配空间(步骤 1002)。一旦类数据结构被分配好，指向这些结构位置中的每一个的指针就被包括在类块中(步骤 1004)。

接着，离线式类加载器读入常量池。在讨论这些步骤之前，首先描述常量池的内容。图 5 说明存储在每个类文件中的常量池的结构。第一项包含类名和超类名(502)。这些名字存储为字符串常量，第一项包含指向常量池中的这些字符串位置的指针。第二项属于字段和实例变量。首标 504 用来指示常量池中字段的个数。首标后面是各个字段 506 。

类似地，字符串常量前面是首标 508，该首标指示常量池中字符串常量的个数。跟着的是各个字符串常量 510 。字符串常量用于表示方法名、类名及接口名。后面，存储在常量池中的方法引用之前是首标，该首标指示方法的数量 512 。常量池中对每个方法都有方法指针 511 ，该方法指针包含指向方法名的指针 514 和指向方法类名的指针 516 。这些名字以字符串常量存储在常量池中。方法指针 511 用来用符号引用一个方法。这用在调用方法指令的非快速格式中。

一个方法可以包含一条调用一个方法的指令。该指令用非快速格式可表示如下:

调用方法“类”.“方法” (1)

这里“类”指包含类名的字符串常量，“方法”指包含方法名的字符串常量。调用方法指令包含一个指向方法指针 511 的指针。离线式类加载器通过向方法名加入指向方法块的指针来解决符号引用。一旦连接器确定了各个类的存

储器布局，连接器以直接引用方法(即通过存储方法的地址)的快速格式取代调用方法指令的非快速格式。通过解决符号引用，该方法可被预加载。

回过头来参照图 9，为映射表 626 分配了存储器，该映射表将跟踪常量在类文件常量池中从最初位置到不同的临时位置的移动，直到常量被存入通用常量池。给每个类建立映射表 626，将常量池中每一个常量的最初地址映射到它的当前地址(步骤 1006)。常量池中除字段外的所有数据都从类文件读取并存入常量池(步骤 1008)。随着常量从类文件中转移到常量池中，映射表 626 要进行更新，以反映常量池中的最初地址与当前地址(步骤 1008)。然而，从常量池中读取的字段被加载入一个或多个字段块，在映射表中没有为它们创建项(步骤 1008)。

方法数据从类文件中读取，并存入一个或多个方法块。附加存储器分配给与每个方法相关的字节码。方法名和指向字节码位置的指针一起放入方法块(步骤 1010)。类似地，异常表也从类文件中加载入相应的异常表数据结构中(步骤 1012)。

回过头来参照图 8，类数据结构建立之后，按照所有常量池的总长度分配了一个散列表(hash table)180(见图 3)，后面将用它删除重复常量(步骤 805)。

接着，离线式类加载器开始删除重复常量。这样做的目的是将所有类的常量池结合成一种有效利用空间的形式。对于每个类文件(步骤 806)，扫描类常量池中的每一项来查找重复常量(步骤 812)。参照图 10，采用散列表检测重复常量。常量的散列值由一个合适的散列函数(hashing function)确定(步骤 1102)。进行检查以确定该散列值是否包含在散列表中(步骤 1104)。如果散列表中存在该散列值，则该常量是重复的，于是通过改变映射表中的常量项以反映已有常量的存储位置，来从常量池中删除该项(步骤 1106)。否则，该常量的散列值和存储位置存入散列表(步骤 1108)。

回过头来参照图 8，离线式类加载器开始确定公用子字符串(common substrings)。公用子字符串是作为已存储在常量池中的一个长字符串的一部分包含在其中的字符串。对于每个类文件(步骤 814)，扫描每个字符串以确定它是否是常量池中包含的某个长字符串的一部分(步骤 816)。这可以采用几种有名的字符串匹配算法之一来完成。当找到这样一个子字符串，它就被指向较大字符串中子字符串位置的指针所取代(步骤 818)。

接着离线式类加载器开始扫描包含在每个类文件中所有方法的字节码(步骤 820 - 824)。参照图 11，在字节码中查找调用接口指令(步骤 1202)。具有调用接口指令的方法都标记为 RAM 存储器，因为被这种指令调用的方法只有运行时间才能完成(步骤 1204)。否则，在字节代码中查找调用方法指令(步骤 1206)。在这种情况下，向存储在常量池中的方法名加入一个指向方法块的指针 518(见图 5)，该方法块包含要被调用的方法(步骤 1208)。后面当连接器确定了所有类的存储器布局，连接器以方法的直接地址取代方法的符号引用。这是通过跟踪指向方法块的方法名指针确定的，该方法块包含指向该方法的指针。

由于每个字节码都被扫描，对每个被字节码存取的变量产生一个引用计数(步骤 828)。引用计数是包含在映射表中的附加字段。这在后面将用于确定最常用常量，从而对常量池重新排序。另外，扫描每个字节码以查找被字节码改变的字段。这可以通过检查该字段是否曾使用在一个赋值语句的左端来确定。在这种情况下，字段接受一个新值。如果被改变的字段将其值存入字段块，那么整个字段块都标记为 RAM 存储器(步骤 830)。

一旦为通用常量池分配了空间，来自不同类常量池的项都合并入通用常量池中(步骤 902)。如前所述，常量池分成两段。第一段最多可容纳 256 个最常引用的常量。第二段容纳其余的常量。不曾被引用的常量被删除，且不存储在通用常量池中。由于映射表具有引用计数，具有被删除的重复项，且具有指向公用子字符串的指针，所以常量池项从每个类的映射表读取。

一旦形成了通用常量池，引用存储在通用常量池第二段中的常量的字节码需要一个双长度偏移量值，它占据了两个字节而不是一个来引用该常量。这要求扫描每个字节码以查找每个类的方法(步骤 906 - 908)，并且对于每个引用通用常量池第二段中一个常量的字节码，以一个双字节偏移量取代单字节偏移量以寻址该常量(步骤 910)。这可以通过为存储该字节码而分配另一个数据区来实现。随着字节码被扫描，它们被读入新数据区并且任何要求用双字节偏移量取代的单字节偏移量在此拷贝过程中被取代。然后方法块被更新以反映字节码新的存储位置。表示分支指令的字节码的偏移量操作数按照在分支字节码和分支目标字节码之间位置加入的字节数(如果有)进行调整。

类似地，如果类文件中的任何方法引用常量池第二段中的常量，每个类文件的异常表都需要调整。常量池第二段中被调整的引用常量的指令将会比

以前占用更多的空间，因此影响到异常表中的偏移量。被双长度常量池偏移量的插入影响的修正过的起始与结束偏移置以及处理程序的偏移量都将被计算并存入异常表中(步骤 916 - 918)。

接着，产生一种处理静态类初始化的新方法。通常情况下，当一个类被动态加载后，同时运行一个类初始化器来初始化与该类相关的某些变量等。由于这些类被预加载到在此描述的方法和系统中，所以将不进行类初始化。因此，离线式类加载器需要创建一个方法，该方法将对预加载的类进行类初始化。然而，为了确定每个类初始化器的执行顺序，先执行数据流分析(步骤 920 - 924)。由于一个类可能用到在另一个类中初始化的数据，因此这个顺序是重要的。如果不保持正确的顺序，静态类初始化器将产生错误的结果。一旦确定了顺序，将产生一个新的方法，该方法将按照类被预加载的顺序对每个类进行初始化(步骤 926)。

然后在类块结构中插入连接指示器(linkage indicator)以标志哪些方法和字段块将被存入随机存取存储器。连接器利用这些信息为将被存入随机存取存储器的方法和字段块生成一个分离的地址空间。连接器可以认为没有指示器意味着方法和字段将被存入只读存储器。另一种可能的选择是，一个附加的指示器可用来显式地指示那些将被存入只读存储器的方法、类数据结构和字段(步骤 928)。

最后，离线式类加载器输出通用常量池、一个包含类数据结构和指明存储器要求的指示器的更新的类文件，以及一个特殊的引导时间启动器(boot time initiator)(步骤 930)。

参照图 12，可预加载的可执行模块和引导时间启动器 1220 永久性地存储在客户机的只读存储器中。每一次客户机上电或重新启动时，都将自动执行引导时间启动器。在其它任务中，引导时间启动器将那些在执行过程中必须驻留在随机存取存储器中的方法和数据，拷贝到随机存取存储器中由连接器程序指定的位置。

虽然前述系统和方法是按照执行 Java 浏览器和支持程序用于在 Internet 上访问 HTML 文档来描述的，但此处描述的方法和系统可应用于任何 Java 应用程序。而且， Java 应用程序不必运行于分布式环境中，它可以以独立模式(stand - alone mode)运行于客户机或服务器计算机上，而不从外部系统输入新的类。在独立模式中，应用程序为满足特殊计算环境的存储器限制被分

成两个地址空间。

虽然此处描述的方法和系统是参考 Java 编程语言描述的，但它也可用于其它采用面向对象的类的计算机系统，这些类使用动态运行时间加载。

进而，上述方法和系统可以在多种类型的可执行媒体上执行，而不仅限于如随机存取存储器这样的存储设备。也可采用其它类型的可执行媒介，例如计算机可读存储介质，它可以是任何存储器设备、光盘和软磁盘，但也不局限于此。

本发明由所附的权利要求的全部等价的保护范围明确定义。

说 明 书 附 图

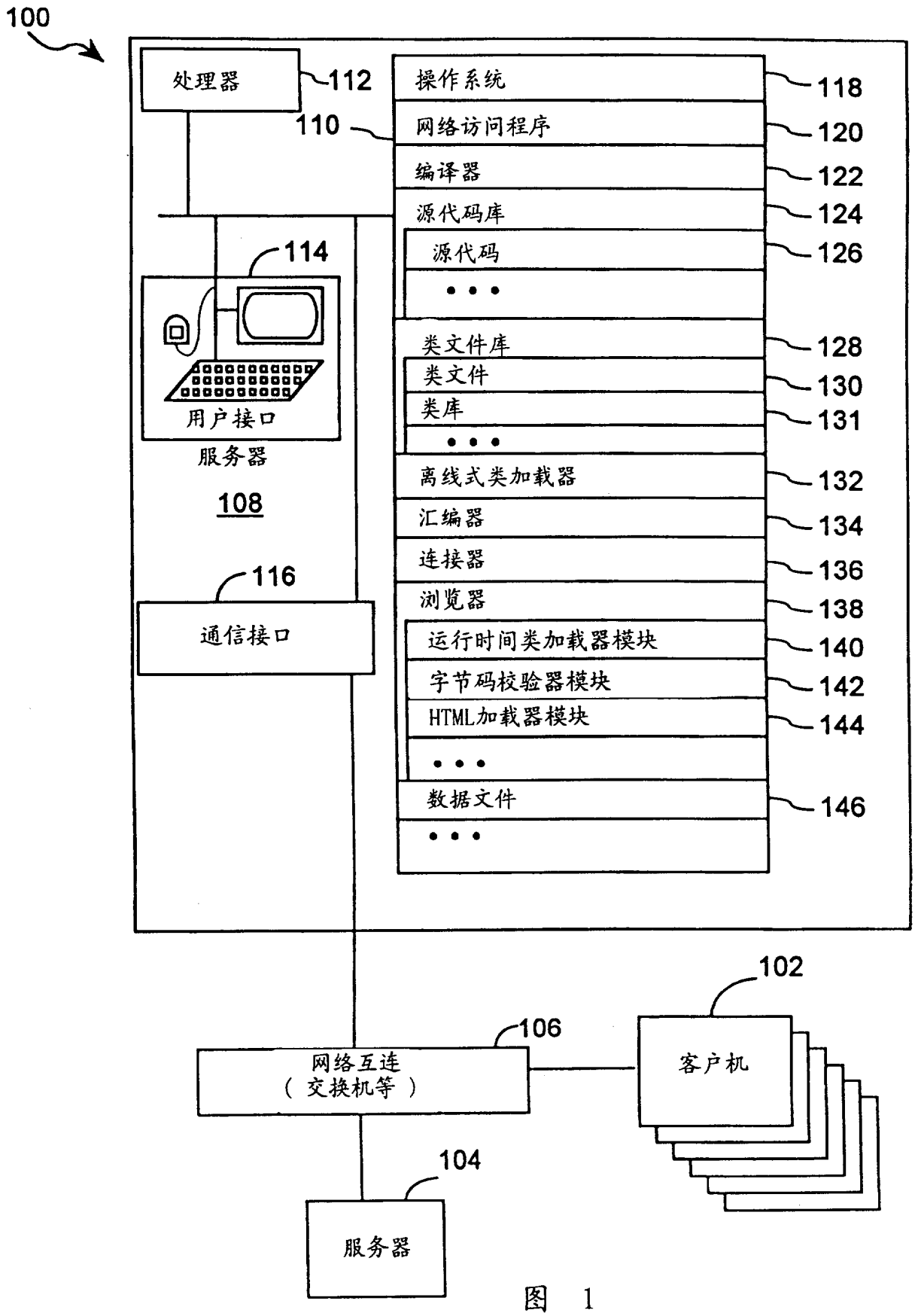

100
处理器
112
110
操作系统
118
网络访问程序
120
编译器
122
源代码库
124
源代码
126
114
用户接口
服务器
108
类文件库
128
类文件
130
类库
131
离线式类加载器
132
汇编器
134
连接器
136
浏览器
138
运行时间类加载器模块
140
字节码校验器模块
142
HTML加载器模块
144
数据文件
146
116
通信接口
102
106
网络互连
（ 交换机等 ）
客户机
104
服务器

图 1

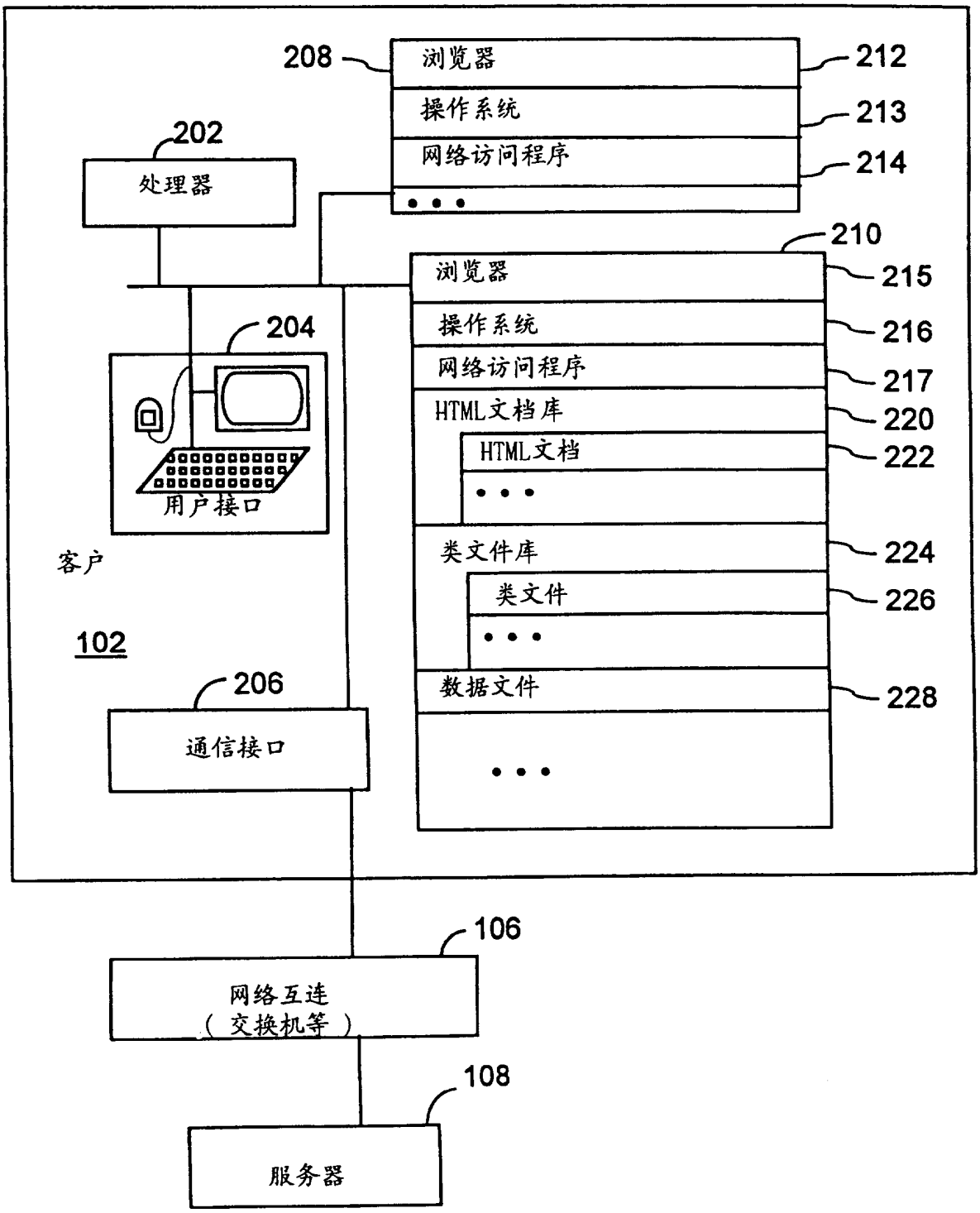

208
浏览器
212
操作系统
213
网络访问程序
214
202
处理器
210
浏览器
215
操作系统
216
网络访问程序
217
HTML文档库
220
HTML文档
222
204
用户接口
客户
类文件库
224
类文件
226
102
206
数据文件
228
通信接口
106
网络互连
（交换机等）
108
服务器

图 2

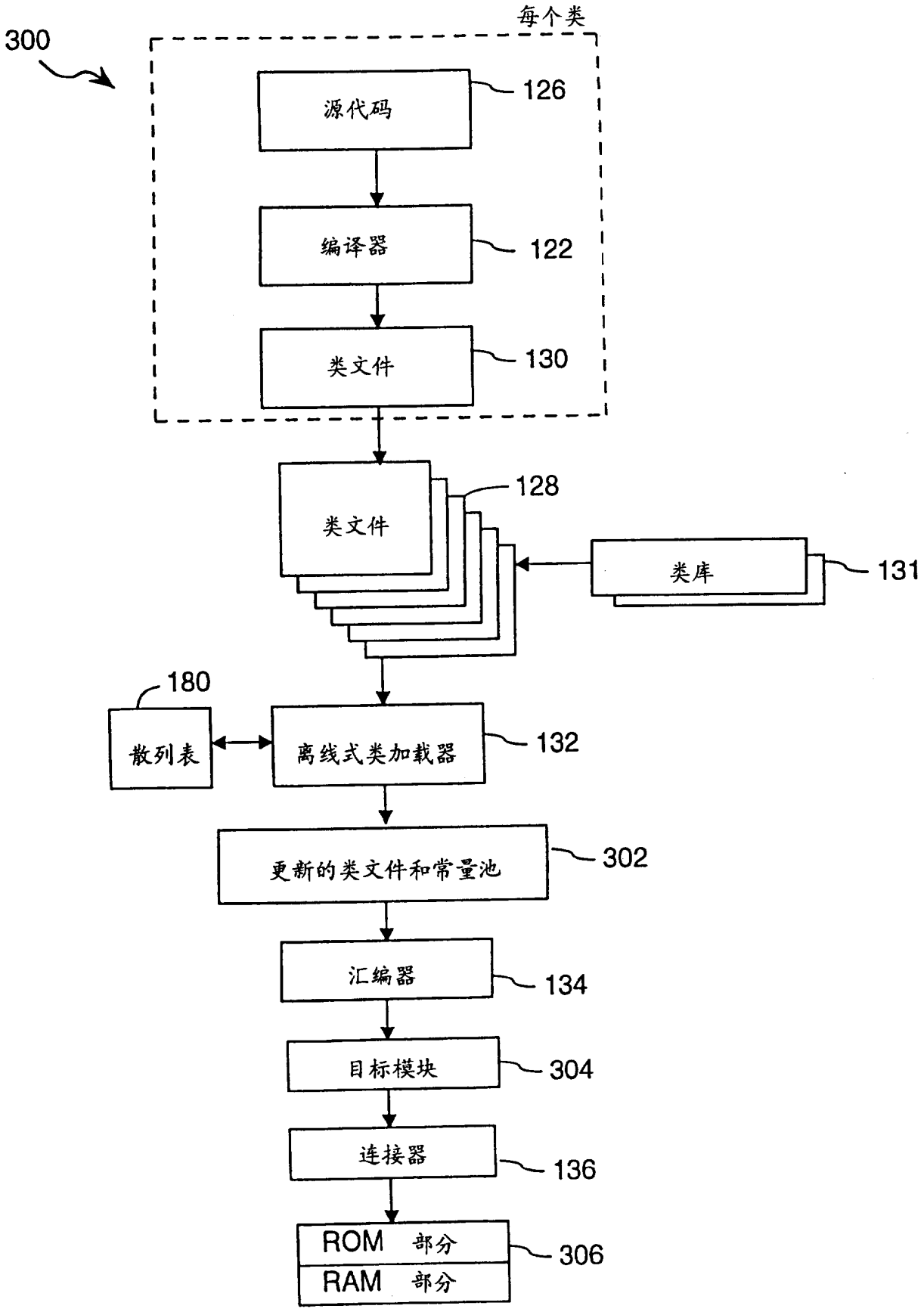

每个类
300
源代码
126
编译器
122
类文件
130
类文件
128
类库
131
180
散列表
离线式类加载器
132
更新的类文件和常量池
302
汇编器
134
目标模块
304
连接器
136
ROM 部分
RAM 部分
306
可预加载的可执行模块

图 3

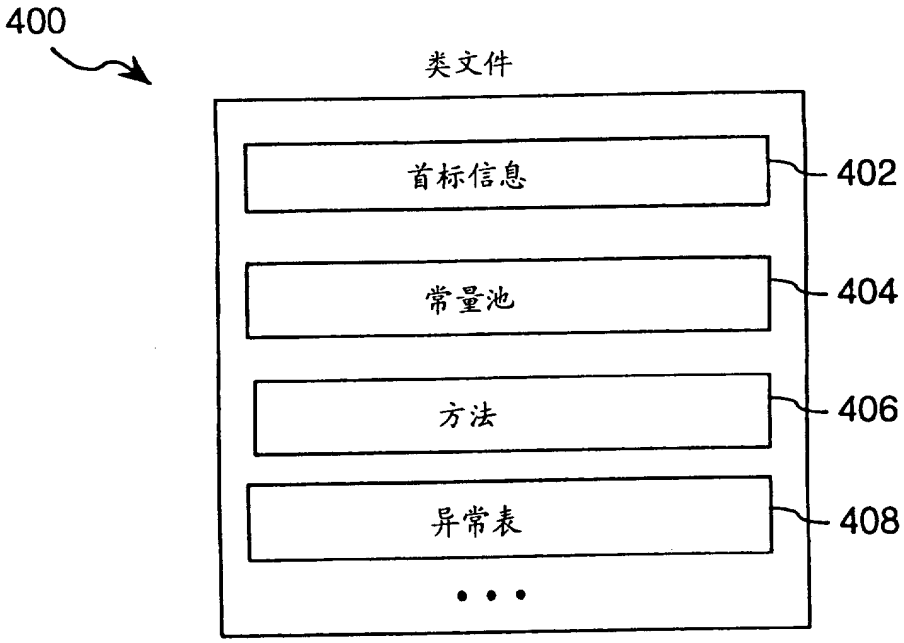

400
类文件
首标信息
402
常量池
404
方法
406
异常表
408

图 4

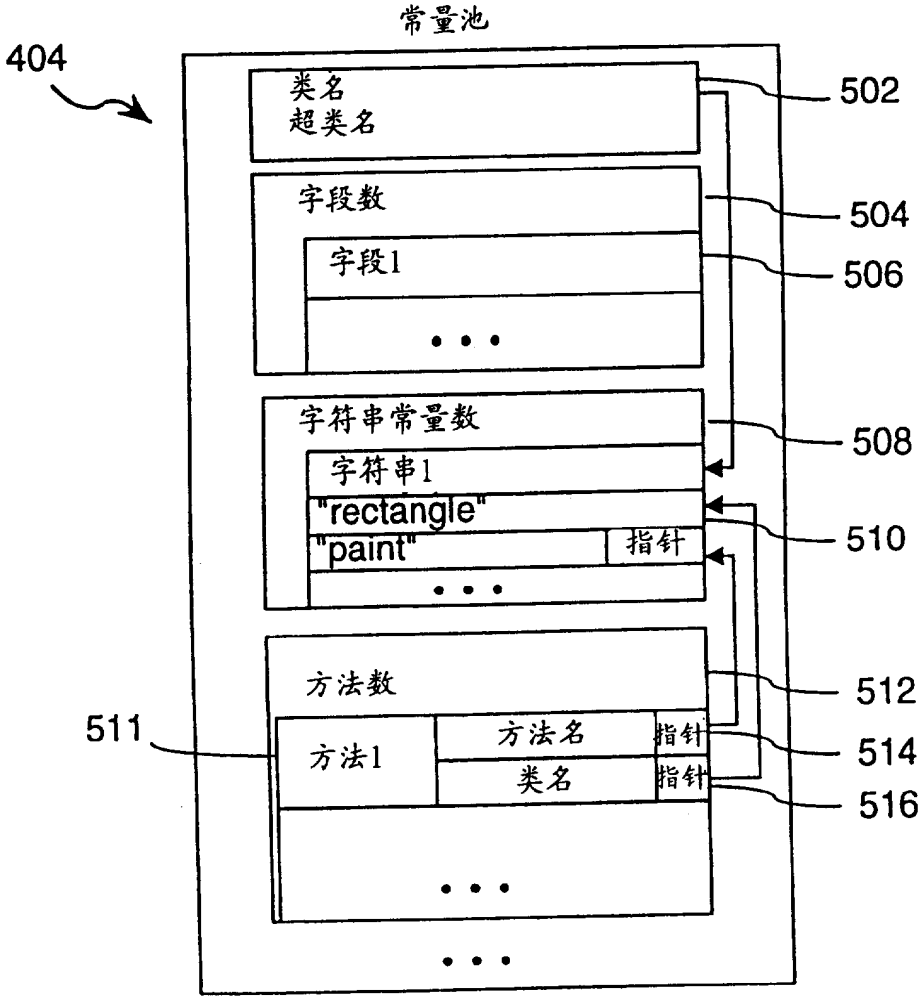

常量池
404
类名
超类名
502
字段数
504
字段1
506
字符串常量数
508
字符串1
"rectangle"
"paint"
指针
510
方法数
512
511
方法1
方法名
指针
514
类名
指针
516

图 5

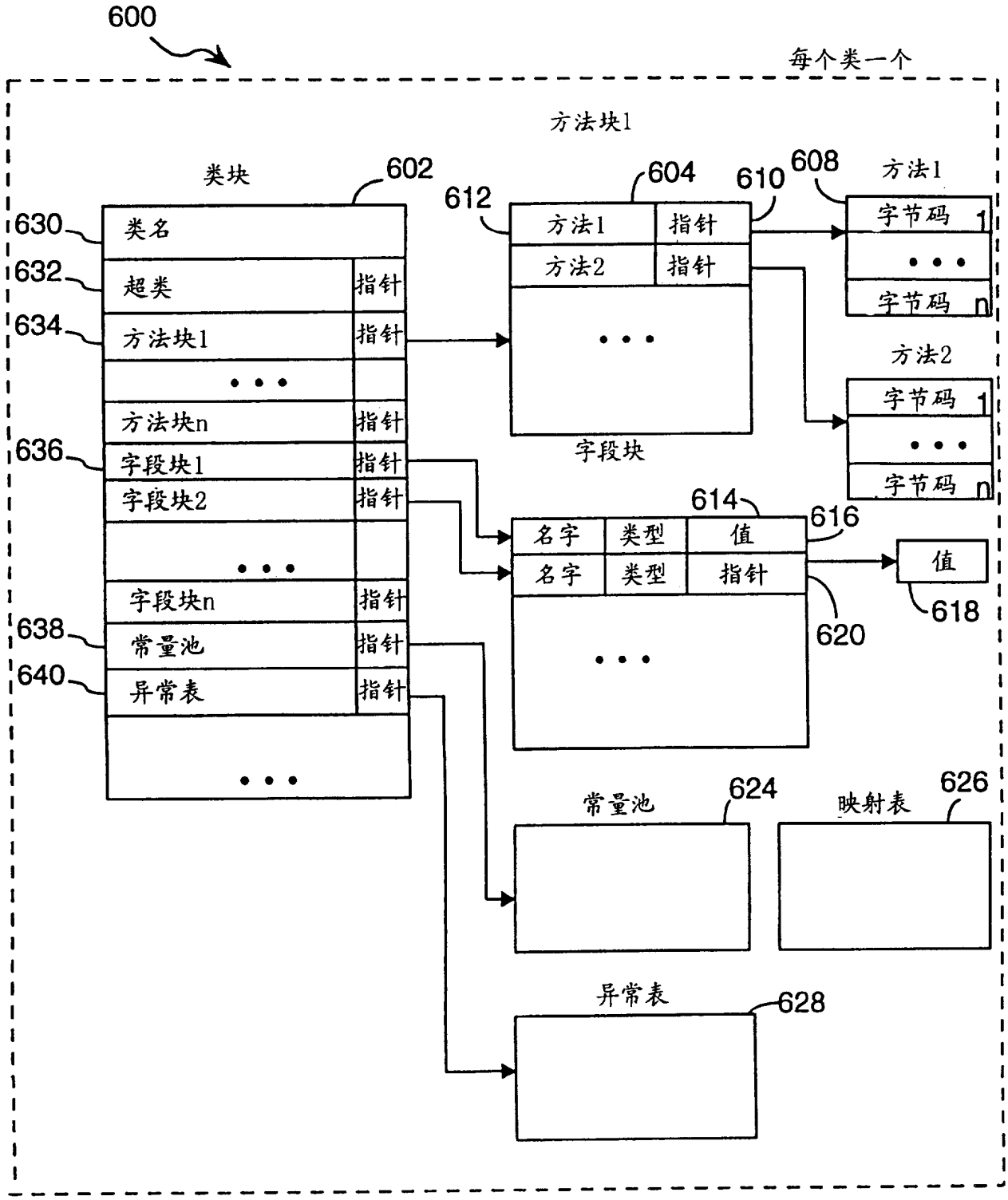

600
每个类一个
方法块1
类块
602
604
610
608
方法1
612
630
类名
方法1
指针
字节码 1
632
超类
指针
方法2
指针
字节码 n
634
方法块1
指针
方法2
方法块n
指针
字节码 1
636
字段块1
指针
字节码 n
字段块2
指针
字段块
614
名字
类型
值
616
字段块n
指针
名字
类型
指针
值
638
常量池
指针
618
640
异常表
指针
620
常量池
624
映射表
626
异常表
628

图 6

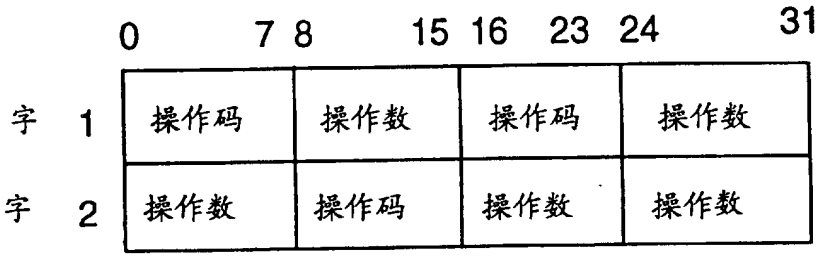


图 7

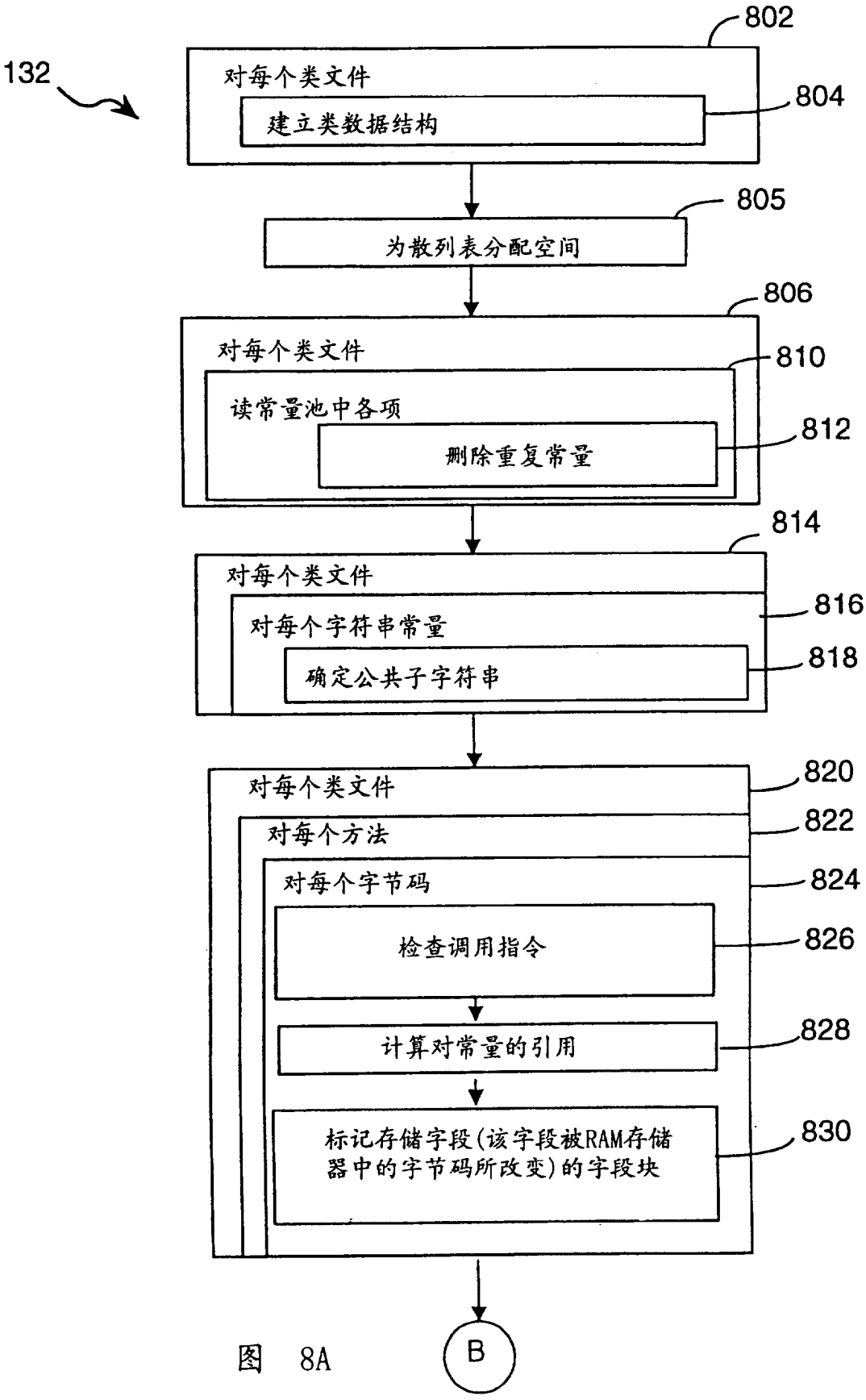

132
802
对每个类文件
804
建立类数据结构
805
为散列表分配空间
806
对每个类文件
810
读常量池中各项
812
删除重复常量
814
对每个类文件
816
对每个字符串常量
818
确定公共子字符串
820
对每个类文件
822
对每个方法
824
对每个字节码
826
检查调用指令
828
计算对常量的引用
830
标记存储字段(该字段被RAM存储器中的字节码所改变)的字段块
B

图 8A

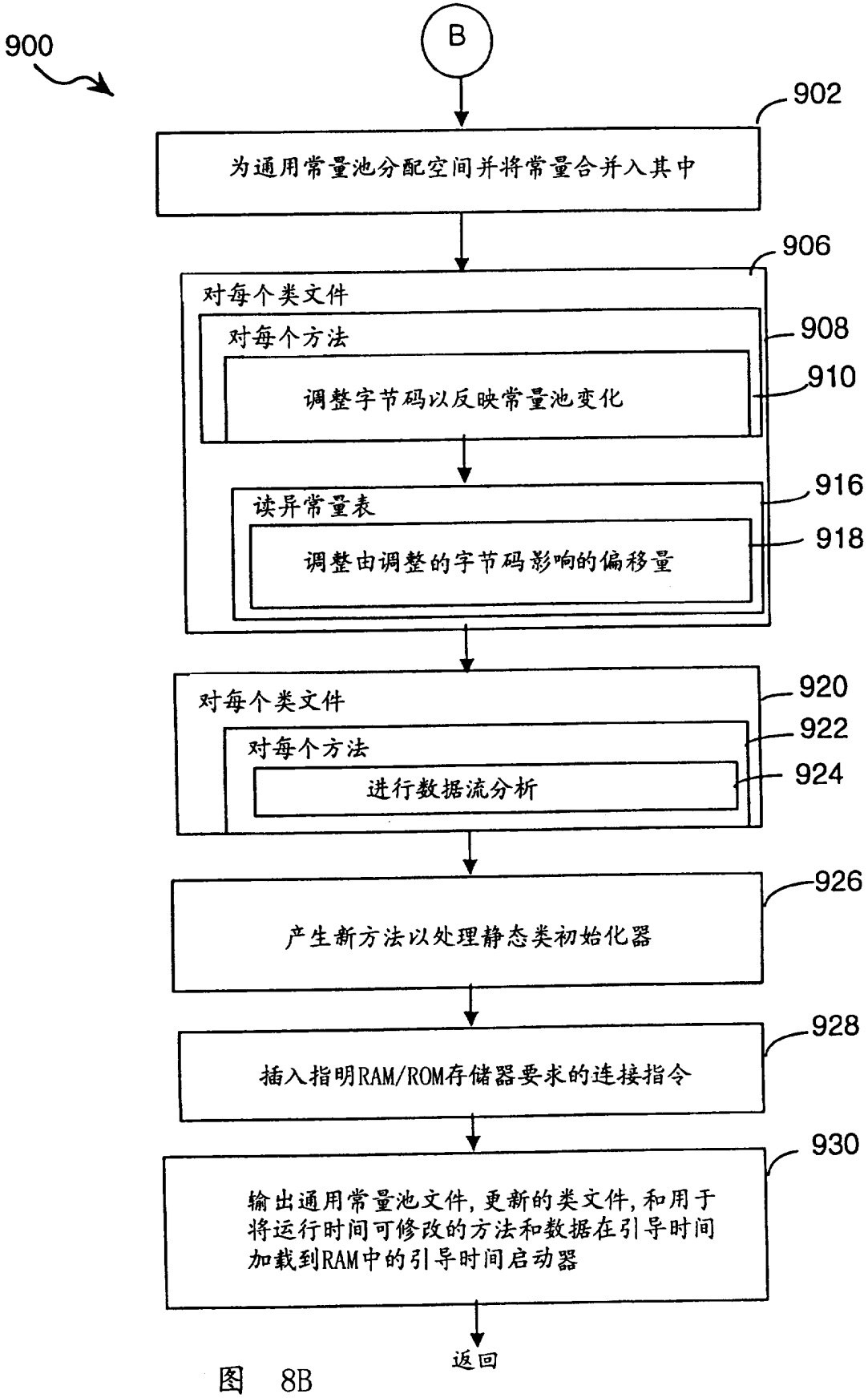

900
B
902
为通用常量池分配空间并将常量合并入其中
906
对每个类文件
908
对每个方法
910
调整字节码以反映常量池变化
916
读异常量表
918
调整由调整的字节码影响的偏移量
920
对每个类文件
922
对每个方法
924
进行数据流分析
926
产生新方法以处理静态类初始化器
928
插入指明RAM/ROM存储器要求的连接指令
930
输出通用常量池文件，更新的类文件，和用于将运行时间可修改的方法和数据在引导时间加载到RAM中的引导时间启动器
返回

图 8B

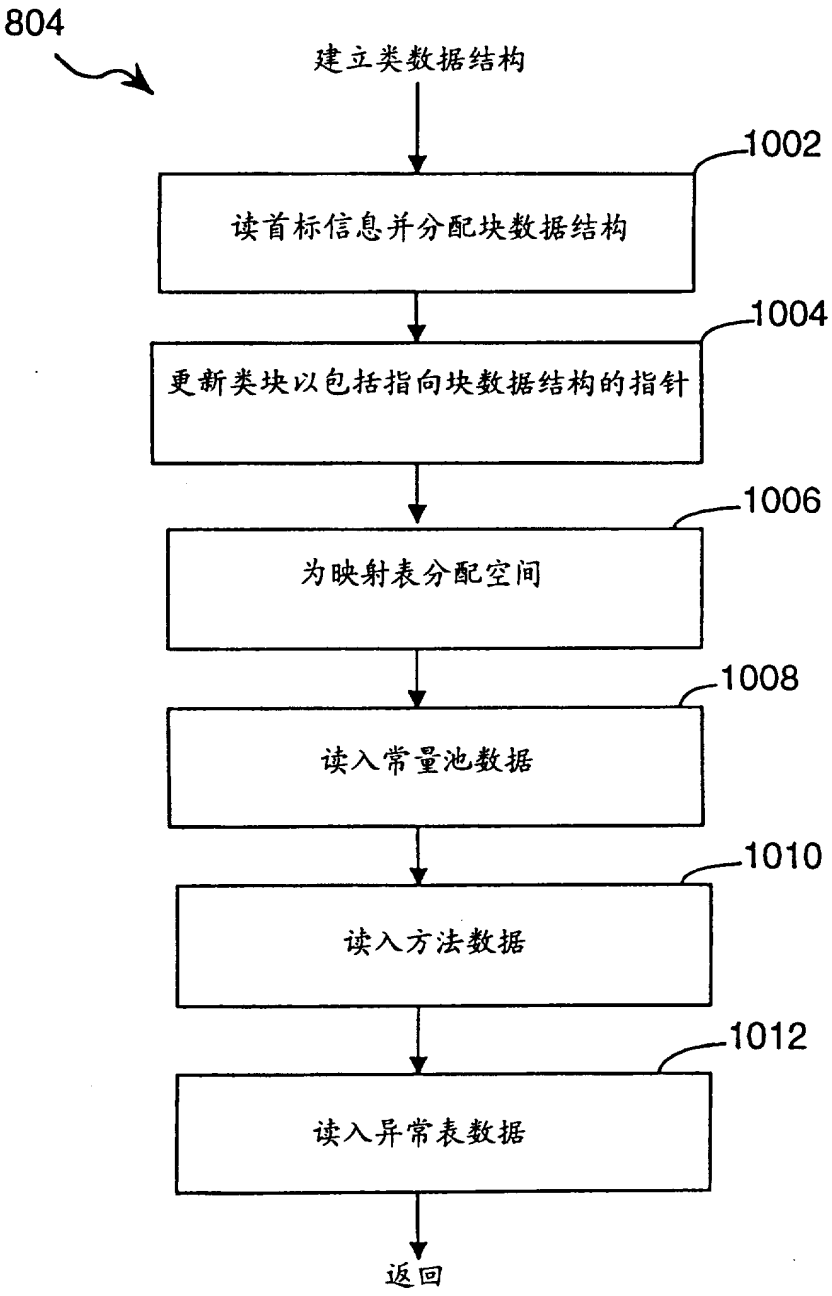

804
建立类数据结构
1002
读首标信息并分配块数据结构
1004
更新类块以包括指向块数据结构的指针
1006
为映射表分配空间
1008
读入常量池数据
1010
读入方法数据
1012
读入异常表数据
返回

图 9

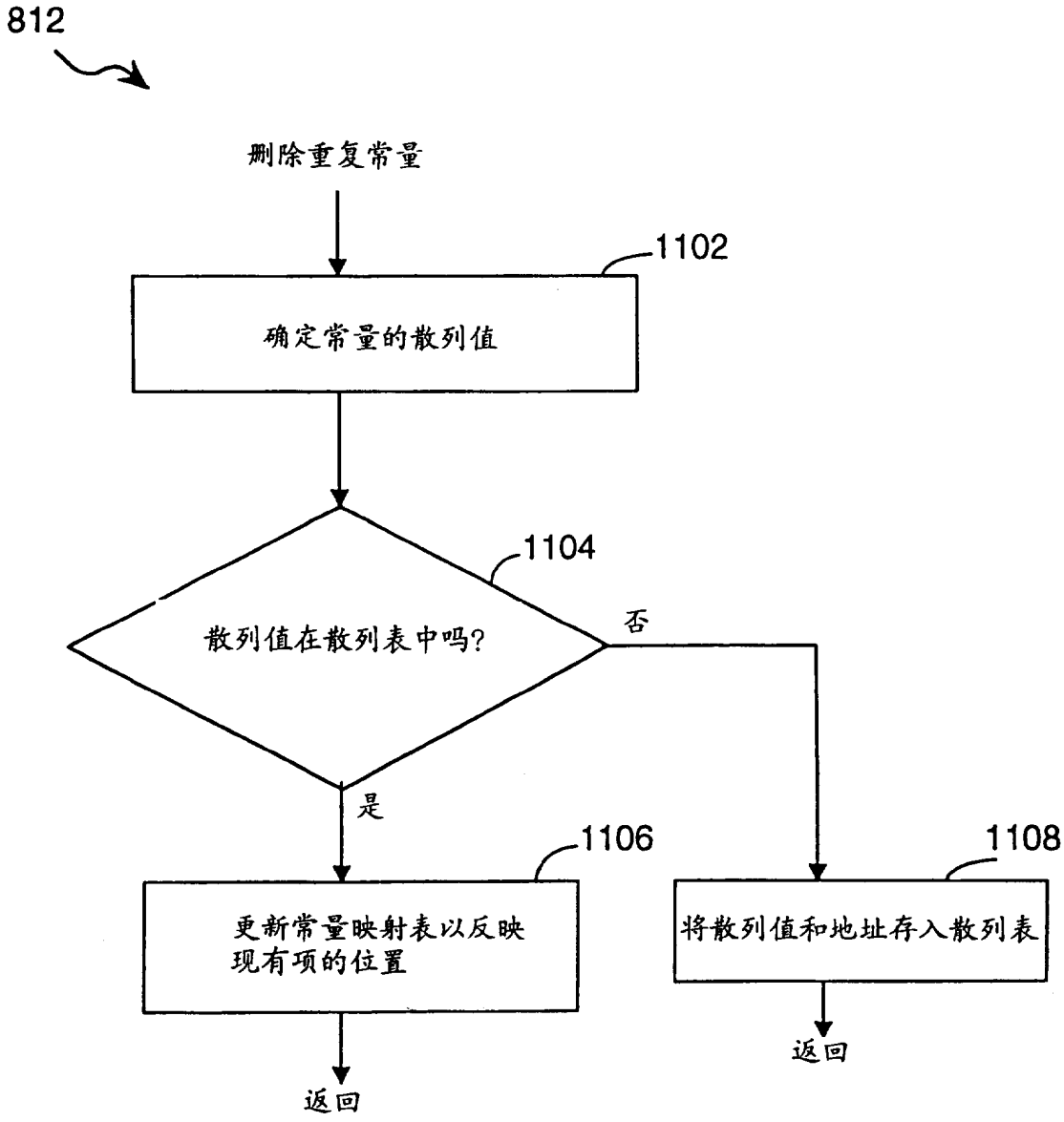

812
删除重复常量
1102
确定常量的散列值
1104
散列值在散列表中吗?
否
是
1106
更新常量映射表以反映现有项的位置
返回
1108
将散列值和地址存入散列表
返回

图 10

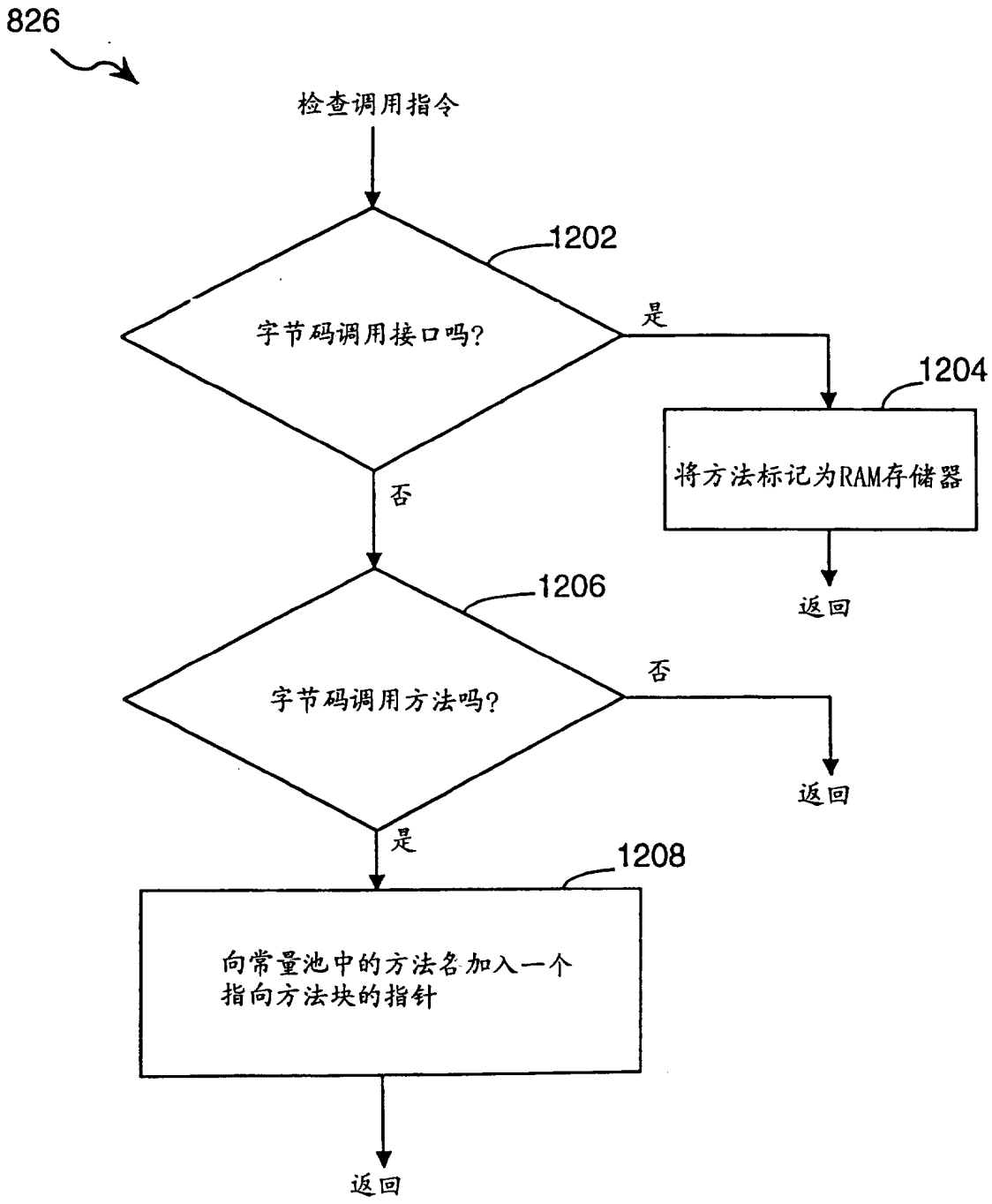

826
检查调用指令
1202
字节码调用接口吗?
是
1204
将方法标记为RAM存储器
返回
否
1206
字节码调用方法吗?
否
返回
是
1208
向常量池中的方法名加入一个
指向方法块的指针
返回

图 11

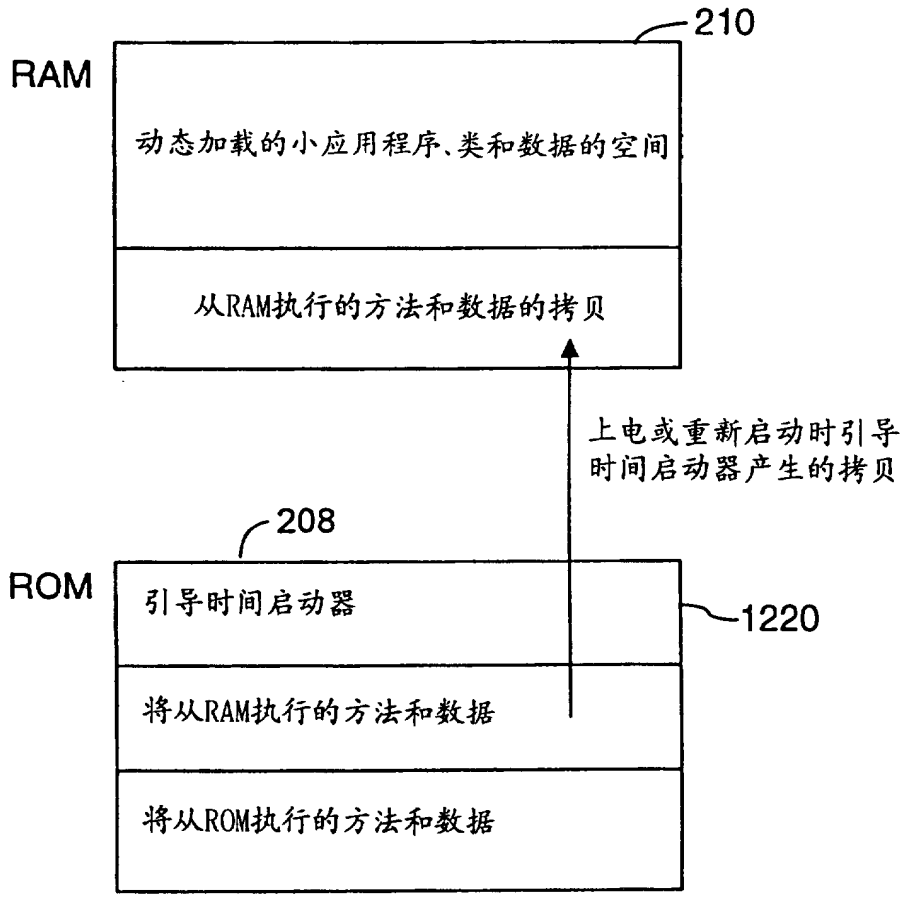

210
RAM
动态加载的小应用程序、类和数据的空间
从RAM执行的方法和数据的拷贝
上电或重新启动时引导
时间启动器产生的拷贝
208
ROM
引导时间启动器
1220
将从RAM执行的方法和数据
将从ROM执行的方法和数据

图 12