[19] 中华人民共和国国家知识产权局

[51] Int. Cl[7]
G06F 7/22
G06F 13/00

[12] 发明专利说明书

[21] ZL 专利号 99107382.7

[45] 授权公告日 2005 年 6 月 8 日

[11] 授权公告号 CN 1205537C

[22] 申请日 1999.3.19 [21] 申请号 99107382.7

[30] 优先权

[32] 1998. 3.20 [33] US [31] 044904

[71] 专利权人 太阳微系统公司

地址 美国加利福尼亚

[72] 发明人 皮特·W·马德尼 里查德·塔克 尼迪姆·福勒斯克

审查员 白雪涛

[74] 专利代理机构 中国国际贸易促进委员会专利商标事务所

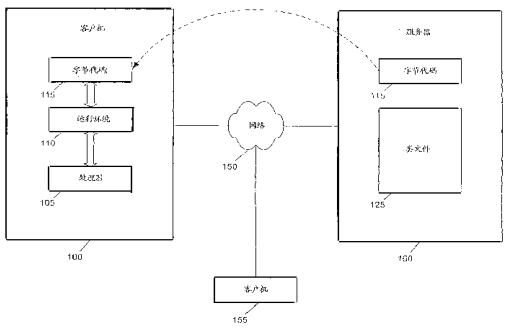
代理人 李德山

权利要求书 6 页 说明书 11 页 附图 8 页

[54] 发明名称 用于为远程执行链接程序的方法

[57] 摘要

一个在服务器上执行的链接编辑器接收用于封装在一个客户机上的程序执行所需的软件成份的指令。 链接编辑器通过循环地分析程序对其他软件成份的引用并将这些成份从其父类中提取出来，产生一个输出文件。 链接编辑器将完成的输出文件发送给一个接口任务，接口任务然后将其发送给客户机。



ISSN1008-4274

1.一种用于封装在分布式系统中执行的程序成份的方法，包括如下步骤：

确定由一个程序所引用的成份；

从一个存储的类中提取出所述成份；

产生一个包括所述程序和提取出的成份的程序包；

分析所述成份对另一个成份的任何引用；

从一存储的类中提取所引用的另一个成份；以及

把提取的所引用的另一个成份增加到程序包中。

2.如权利要求 1 所述的方法，还包括提取出与存储的类相联系的辅助信息的步骤。

3.如权利要求 1 所述的方法，还包括下列步骤：

为一个替代方法分析一个子类；

从一个存储的类中提取出所述替代方法；以及

将提取出的所述替代方法增加到程序包中。

4.如权利要求 1 所述的方法，还包括下列步骤：

为一个替代以前加载的方法的方法分析一个存储的类；

从存储的类中提取出所述替代方法；以及

将提取出的所述替代方法增加到程序包中。

5.如权利要求 1 所述的方法，其特征在于，提取步骤还包括根据一个以前提取的成份确定是否应该提取出一个成份的步骤。

6.如权利要求 1 所述的方法，其特征在于，提取步骤还包括下列步骤：

接收一个关于不应被增加到所述程序包中的被排除的成份的指令；以及

根据所述指令确定是否应该提取出一个成份。

7.如权利要求 2 所述的方法，还包括下列步骤：

接收一个关于不应被增加到所述程序包中的被排除的成份的指

令；以及

根据所述指令确定是否应该提取出一个成份。

8.如权利要求 **1** 所述的方法，还包括下列步骤：

接收一个关于应该被增加到所述程序包中的附加成份的指令；

从一个存储的类中提取出所述附加成份；以及

将所述附加成份增加到程序包中。

9.如权利要求 **2** 所述的方法，还包括下列步骤：

接收一个关于应该被增加到所述程序包中的附加成份的指令；

从一个存储的类中提取出所述附加成份；以及

将所述附加成份增加到程序包中。

10.如权利要求 **1** 所述的方法，还包括下列步骤：

接收程序执行的开始点；以及

接收一个类文件的位置。

11.如权利要求 **1** 所述的方法，其中所引用的另一个成份是以最后一个被引用成份结尾的一个引用成份序列中的第一成份，所述方法还包括下列步骤：

为对该序列中一较低成份的任何引用分析所述所引用成份序列中的每一个所引用成份；

从一个存储的类中提取所述序列中的每个所引用的较低的成份；以及

把所述序列中的每个提取出的较低的引用成份增加到程序包中，其中循环分析所述序列中的每个引用成份、提取出每个较低的引用成份和增加每个提取出的引用成份的各步骤，直至从一个存储的类中提取出所述引用成份序列中的最后一个引用成份并且把它增加到程序包中。

12.如权利要求 **1** 所述的方法，其中所述程序引用多个成份，并且其中所述程序引用的每个成份是以最后一个引用成份结尾的相应引用成份序列中的第一成份，所述方法还包括下列步骤：

确定由所述程序引用的多个成份中的每个成份；

从一个相应的存储的类中提取出所述多个成份中的每个成份；

把提取出的多个成份增加到程序包中；

为对各个序列中一较低成份的任何引用分析每个引用成份序列中的每个引用成份；

从一存储的类中提取每个序列中的每个引用的较低的成份；以及

把每个序列中的每个提取出的较低的引用成份增加到程序包中，其中循环分析每个序列中的每个引用成份、提取出每个较低的引用成份和增加每个提取出的引用成份的各步骤，直至从一个存储的类中提取出每个引用成份序列中的最后一个引用成份并且把它增加到程序包中。

13.一种用于封装在分布式系统中执行的程序成份的方法，所述分布式系统包括通过网络互连的第一计算机节点和第二计算机节点，该方法包括如下步骤：

确定由一个程序所引用的成份；

在第一计算机节点处从一个存储的类中提取出所述成份；

在第一计算机节点处产生一个包括所述程序和提取出的成份的程序包；

把所述程序包发送到第二计算机节点；以及

在第二计算机节点处执行所述程序包中包括的程序和提取出的成份。

14.如权利要求 13 所述的方法，还包括下列步骤：

在第一计算机节点处接收一个来自第二计算机节点的关于不应被增加到所述程序包中的被排除的成份的指令；以及

在第一计算机节点处根据所述指令确定是否应该提取出一个成份。

15.如权利要求 13 所述的方法，还包括下列步骤：

在第一计算机节点处接收一个来自第二计算机节点的关于应该被增加到所述程序包中的附加成份的指令；

在第一计算机节点处从一个存储的类中提取出所述附加成份；以

及

将所述附加成份增加到程序包中。

16.如权利要求 13 所述的方法，其中所引用的成份是以最后一个被引用成份结尾的一个引用成份序列中的第一成份，所述方法还包括下列步骤：

在第一计算机节点处为对该序列中一较低成份的任何引用分析所述所引用成份序列中的每一个所引用成份；

在第一计算机节点处从一个存储的类中提取所述序列中的每个所引用的较低的成份；以及

在第一计算机节点处把所述序列中的每个提取出的较低的引用成份增加到程序包中，其中循环分析所述序列中的每个引用成份、提取出每个较低的引用成份和增加每个提取出的引用成份的各步骤，直至从一个存储的类中提取出所述引用成份序列中的最后一个引用成份并且把它增加到程序包中。

17.如权利要求 13 所述的方法，其中所述程序引用多个成份，并且其中所述程序引用的每个成份是以最后一个引用成份结尾的相应引用成份序列中的第一成份，所述方法还包括下列步骤：

在第一计算机节点处确定由所述程序引用的多个成份中的每个成份；

在第一计算机节点处从一个相应的存储的类中提取出所述多个成份中的每个成份；

在第一计算机节点处把提取出的多个成份增加到程序包中；

在第一计算机节点处为对各个序列中一较低成份的任何引用分析每个引用成份序列中的每个引用成份；

在第一计算机节点处从一存储的类中提取每个序列中的每个引用的较低的成份；以及

在第一计算机节点处把每个序列中的每个提取出的较低的引用成份增加到程序包中，其中循环分析每个序列中的每个引用成份、提取出每个较低的引用成份和增加每个提取出的引用成份的各步骤，直至

从一个存储的类中提取出每个引用成份序列中的最后一个引用成份并且把它增加到程序包中。

18.一种用于封装在分布式系统中执行的程序成份的方法，由服务器执行的该方法包括下列步骤:

确定由客户计算机请求的程序所引用的成份;

从一个存储的类中提取所述成份;

产生一个包括所述程序和提取出的成份的程序包;

分析所述成份对另一个成份的任何引用;

从一个存储的类中提取出所引用的另一个成份;

将提取出的所引用的另一个成份增加到程序包中; 以及

把所述程序包发送到所述客户计算机。

19.如权利要求 **18** 所述的方法，其中所引用的另一个成份是以最后一个被引用成份结尾的一个引用成份序列中的第一成份，所述方法还包括在向客户计算机发送所述程序包前执行下列步骤:

为对该序列中一较低成份的任何引用分析所述所引用成份序列中的每一个所引用成份;

从一个存储的类中提取所述序列中的每个所引用的较低的成份; 以及

把所述序列中的每个提取出的较低的引用成份增加到程序包中，其中循环分析所述序列中的每个引用成份、提取出每个较低的引用成份和增加每个提取出的引用成份的各步骤，直至从一个存储的类中提取出所述引用成份序列中的最后一个引用成份并且把它增加到程序包中。

20.如权利要求 **18** 所述的方法，其中所述程序引用多个成份，并且其中所述程序引用的每个成份是以最后一个引用成份结尾的相应引用成份序列中的第一成份，所述方法还包括在向客户计算机发送所述程序包前执行下列步骤:

确定由所述程序引用的多个成份中的每个成份;

从一个相应的存储的类中提取出所述多个成份中的每个成份;

把提取出的多个成份增加到程序包中；

为对各个序列中一较低成份的任何引用分析每个引用成份序列中的每个引用成份；

从一存储的类中提取每个序列中的每个引用的较低的成份；以及

把每个序列中的每个提取出的较低的引用成份增加到程序包中，其中循环分析每个序列中的每个引用成份、提取出每个较低的引用成份和增加每个提取出的引用成份的各步骤，直至从一个存储的类中提取出每个引用成份序列中的最后一个引用成份并且把它增加到程序包中。

用于为远程执行链接程序的方法

本申请与下面的美国专利申请相关，并将其作为参考：美国专利申请No.09/044,900，题目为“用于为远程执行封装程序的方法和装置”，其代理文档(attorney docket)为no.06502.0073-00000，并且与本申请的申请日相同。

技术领域

本发明的实施例一般地涉及分布式计算机系统，特别涉及用于为远程执行封装程序的方法和装置。

背景技术

在当今社会中，因特网已经成为信息交换的重要媒介。虽然因特网现在在一般公众中非常普及，其最初是开始于一个由政府和学术研究者使用的互连计算机的系统（或网络）。这个网络的早期问题是由互连计算机不相同这个事实所产生的；他们采用不同的硬件和不同的操作系统。在这样一个多机种网络上的信息交换提出了一个通信问题。这个问题是通过在共同标准上的协议来解决的，包括诸如传输控制协议/互连网络协议（TCP/IP）和超文本传输协议（HTTP）等的协议。这些协议使各种互连机器能以静态文本或图形文件的形式共享信息。

然而，这些协议在因特网的发展中只代表了两个步骤。虽然用户可以在与因特网相连的各种计算机中交换信息文件，但他们不能交换以常规语言、例如C或C++写成的可执行应用程序，这些可执行应用程序是设计为与一特定处理器（例如，Intel奔腾处理器）和/或一特定操作系统（例如，Windows 95或DOS）进行连系的。这个问题是以Java™程序设计语言及其相关的运行系统的出现而解决的。

Java程序设计语言是一个面向对象的程序设计语言，在例如由Addison-Wesley的Mary Campione和Kathy Walrath在1996年所著的题为“Java™指导”的教科书中进行了描述[1]。重要地是，Java程序设计语言是一个与平台无关的解释语言—也就是说，其实用性并不限于一个特定的计算机系统。使用Java程序设计语言，软件开发者以通常被称为Java源代码的形式写程序。当开发者完成程序的编写时，他则以一个Java编译器将其编译成被称为字节代码的中间形式。Java源代码和字节代码都是与平台无关的。

然后，可以将编译过的字节代码在任何采用了一个包括虚拟机（VM）的可

兼容运行系统的计算机系统上执行，例如包括 Java 虚拟机（JVM）和 Java 类库的 Java 运行环境（JRE）。JVM 在由 Addison Wesley 的 Tim Lindholm 和 Frank Yellin 在 1996 年所著的题为“Java 虚拟机说明书”的教科书中进行了描述。Java VM 用作为字节代码和所使用的特定计算机系统之间的解释器。通过使用与平台无关的字节代码和 Java VM，可以在任何计算机系统上执行以 Java 程序设计语言写出的程序。这在将不同的计算机系统进行互连的网络、例如因特网中是特别有用的。

在能够执行一个 Java 程序之前，必须将某些必不可少的类（class）装载进执行该程序的计算机的存储器中。这些类可以从计算机的磁盘装载，但更常用的是通过网络从一个服务器传输。习惯上，将这些类在程序执行期间尽可能晚地装载；换句话说，它们是根据需要在程序执行期间首次被引用时才装载的。当这种装载出现时，一旦需要一个类的任何一部分，通常装载整个类。

根据一种常规方法，当机器上的用户发出执行驻留在一个远程服务器上的程序的请求时，将包含 main 方法的类文件通过网络从该服务器装载到该客户机。这个类文件包含程序字节代码。虚拟机然后通过调用程序的 main 方法开始执行。

执行一直持续进行到程序引用一个成份，例如，被称为“F”的成份。响应于该引用，将包含成份 F 的整个类从服务器上的类文件通过网络传输给客户机。所引用的成份 F 被使用，执行然后继续进行，直到引用另一个成份，例如被称为“G”的成份。响应于该引用，将包含成份 G 的整个类从服务器上的类文件通过网络传输给客户机。执行则继续进行到结束。一旦执行已经完成，在客户机和服务器之间通过网络的一系列连接最终可以被终止。

这个说明证明常规的方法有两个显著的缺陷。首先，在程序执行期间，它需要在客户机和服务器之间的重复连接。这样一种连接所必须的长周期对于诸如移动计算的情况是成问题的。其次，一旦引用一个成份，例如成份 F，该方法需要从服务器加载包含所引用成份的整个类。然而，如果在类内只有很少的成份最终被使用，则从服务器向客户机传输未用成份所需的带宽就被浪费了。这在涉及客户机和服务器之间的有限带宽、即慢的连接速度或高等待时间的连接的情况中是成问题的。

因此，需要一种通过只将一整个程序的所需成份封装在一起并在执行开始之前将其传送给客户机来减轻这些问题的系统。已经采用了用于远程执行的预封装软件，该软件使用其他需要机器兼容性的编译计算机语言，例如 Cobol、C 和 Fortran。然而，还没有采用使用一种面向对象的语言、例如 Java 程序设计语言

的软件，其中 Java 程序设计语言提供了很多另外的好处，例如从其父类提取所需的成份。

发明内容

本发明的一个目的是提供一种实际只封装程序执行所必需的成份的方法、设备和系统。

根据本发明的第一方面，提供了一种用于封装在分布式系统中执行的程序成份的方法，包括如下步骤：确定由一个程序所引用的成份；从一个存储的类中提取出所述成份；产生一个包括所述程序和提取出的成份的程序包；分析所述成份对另一个成份的任何引用；从一存储的类中提取所引用的另一个成份；以及把提取的所引用的另一个成份增加到程序包中。

根据本发明的另一方面，提供了一种用于封装在分布式系统中执行的程序成份的方法，所述分布式系统包括通过网络互连的第一计算机节点和第二计算机节点，该方法包括如下步骤：确定由一个程序所引用的成份；在第一计算机节点处从一个存储的类中提取出所述成份；在第一计算机节点处产生一个包括所述程序和提取出的成份的程序包；把所述程序包发送到第二计算机节点；以及在第二计算机节点处执行所述程序包中包括的程序和提取出的成份。

根据本发明的另一方面，提供了一种用于封装在分布式系统中执行的程序成份的方法，由服务器执行的该方法包括下列步骤：确定由客户计算机请求的程序所引用的成份；从一个存储的类中提取所述成份；产生一个包括所述程序和提取出的成份的程序包；分析所述成份对另一个成份的任何引用；从一个存储的类中提取出所引用的另一个

成份；将提取出的所引用的另一个成份增加到程序包中；以及把所述程序包发送到所述客户计算机。

附图说明

在本说明书中所包含的并构成说明书的一部分的附图显示了本发明的一个实施例，并与说明书一起用于解释本发明的优点和原理。在附图中，

图 1 是一个典型的客户机-服务器结构的方框图，用于说明使用包括一个虚拟机的运行环境的远程程序执行;

图 2 是显示一个典型的可执行面向对象程序及其引用的成份和它们的依赖性的方框图;

图 3 是说明与本发明一致的链接和执行的定时的时线;

图 4 是显示典型的与服务器和客户机有关的接口和链接编辑器任务的方框图;

图 5 是由与本发明一致的封装处理中的接口任务执行的步骤的流程图;

图 6 是在本发明的一个实施例中由封装输出文件中的链接编辑器执行的步骤的流程图；以及

图 7 是在本发明的一个实施例中为了增加必要的方法而执行的步骤的流程图；以及

图 8 是在本发明的一个实施例中为了执行程序而由客户机执行的步骤的流程图。

具体实施方式

下面将参考附图详细说明本发明的实施形式。其中，在整个附图和下面的说明中，相同的标号代表相同或相似的部分。

A. 概述

依据本发明的系统和方法工作在一个分布式计算机系统中，该系统典型地具有多个客户机和一个或多个服务器。例如，试图执行一个程序的客户机向服务器请求将运行程序所需的软件封装在一起。在形成该请求时，客户机可以指定应该包括在该程序包内的特定成份。

在服务器执行的任务（“接口任务”）接收到这个请求，并确定已经驻留在客户机内的软件成份。接口任务应用该信息形成一组给被称为链接编辑器的另一任务的指令。这些指令可以包括：要执行的程序名，因为已经驻留在客户机内而不必被封装的成份，以及客户机可能已经指定要包括在该程序包内的任何软件成份的名称。

链接编辑器接收此信息，并产生一个包含驻留在服务器内并且程序执行所必须的所有软件成份的输出文件。这个输出文件是通过迭代地分析引用其他软件成份的程序并从这些成份的父类提取出这些成份而产生的。链接编辑器将完成的输出文件发送给接口任务，接口任务将其传送给需要的客户机，程序开始执行。

B. 术语

为了说明的缘故，下面的详细说明是基于 Java 程序设计语言的。由于这个原因，下面是在下文所用到的术语的简要定义部分。然而，本领域的普通技术人员应该认识到，下面说明的原理同样适用于其他程序设计语言。

Java 应用程序包括一个或多个类定义，每个类定义已经被编译成它自己的.class 文件，包含字节代码和其他信息。一个“类”本身又是一个数据（“字段”）、在数据上操作的“方法”、以及辅助信息的集合。例如，辅助信息可以包括共享的数据结构、超类（superclass）的名字、以及实现的接口。在这里所用的术语“成份”指的是方法或字段或二者皆包括。“对象”是采用由一个类提供的蓝图所生成的东西，即它是类的一个“实例”。Java 应用程序必须包含一个定义 main 方法的类，该 main 方法代表 Java 解释器开始执行程序的一点。这样一种应用程序可以由 Java 解释器、即 Java VM 来执行。

与一个单独的应用程序相比，Java 小应用程序(applet)不包含 main 方法，因此不能直接由 Java 解释器执行。相反，Java 小应用程序是由一个已经执行的

Java 应用程序、例如网浏览器所加载的类。Java 应用程序在合适的时间调用小应用程序的各种方法。

在这里所用的术语“程序”，在单独使用时，可以代表应用程序、Java 小应用程序、过程或其他软件代码。术语“任务”可以代表一个在计算机处理器上执行的程序。术语“程序包”可以包括成份、辅助信息或程序执行所需的其他数据。

为了简化的缘故，在这里所包含的例子假设一个应用程序正在执行。然而，本领域普通技术人员应该认识到，本发明的权利要求可以包括一个小应用程序或其他软件代码的执行。

C. 结构

图 1 显示了 Java 程序设计语言在一个分布式计算系统中的使用。该系统包括通过网络 150 互连的一个或多个服务器、例如服务器 160 和一个或多个客户机、例如客户机 100 和 155。软件开发者应用 Java 程序设计语言生成一个程序，并将其编译成字节代码 115，存储在服务器 160 上。服务器 160 一般还包括很多由 Java 程序使用的类文件 125。当客户 100 希望执行一个 Java 程序时，它向服务器 160 发出一个请求。作为响应，服务器 160 向客户机 100 发送程序 115 的字节代码版本。在客户机 100，字节代码 115 在一个运行环境 110 上执行，运行环境 110 在字节代码 115 与驻留在客户机 100 上的处理器之间进行解释。

图 2 是显示在一个典型的可执行程序内的成份引用的例示性方框图。在这个例子中，程序 200 引用四个成份，分别显示为成份 A 210、成份 B 220、成份 C 230 和成份 D 240。这些引用成份自己又引用其他成份。例如，成份 B 220 引用成份 B1 250。同样，成份 C 230 引用成份 A 210 和成份 C1 270。成份 C1 270 自己又引用成份 C1A 280。这样一种由一个成份对另一个成份的引用通常被称作为依赖性（dependency）。因为在程序执行期间会被用到，每个被引用的成份必须被加载。

D. 时线

图 3 是用于说明依据本发明的链接和执行的定时的时线。当用户发出一个在客户机 100 上执行程序的请求时（点 310），程序开始。作为响应，服务器 160 确定程序执行需要哪些成份和依赖性（点 320）。然后，将这些所需成份的每一个与辅助类信息一起通过网络 150 从服务器 160 传送给客户机 100（点 330）。此时，已经将程序执行所需的所有成份和类都从服务器 150 传送到客户机 100，

因此，在两者之间的连接可以中断。

然后程序执行开始（点 340），并引用成份 A 210（点 350）。然而，因为该成份已经（在点 330）被加载在客户机 100 上，不需要将该成份从服务器 160 向客户机 100 单独传送。同样，在引用成份 B 220 时（点 360），因为它已经被从服务器 160 传送到客户机 100，所以不需要再传送它。因此，执行不中断地进行到结束（点 370）。

图 3 证明了依据本发明的处理仅仅需要客户机 100 与服务器 160 保持连接到程序加载结束（从点 300 至点 330）。一旦所需的成份和辅助类信息已经被加载（在点 330），客户机 100 可以断开与服务器 160 的连接。则对于整个程序执行期间（点 340 至点 370），客户机 100 与服务器 160 无须相连。此外，服务器 160 只向客户机 100 传送对于程序执行所必须的那些成份和辅助类信息（点 330）。通过去除了对无用成份的传输，则更有效地利用了带宽。

E. 接口任务和链接编辑器的结构

图 4 是显示与服务器和客户机相关的接口任务和链接编辑器的方框图。客户机 100 通过网络 150 与服务器 160 互连。服务器 160 包含一个可以包含 Java 类文件的存储器 482。存储器 482 的例子包括随机存取存储器（RAM）、只读存储器（ROM）、硬盘驱动器或基于光盘的 ROM（CD-ROM）。有两个任务在服务器 160 上执行：接口任务 484 和链接编辑器 487。向服务器 160 发出一个封装所需 Java 代码的请求，以便在客户机 100 上执行程序。在服务器 160，这个请求由接口任务 484 接收。接口任务 484 然后形成一组封装指令，并将其发送给链接编辑器 487。链接编辑器 487 生成一个包含驻留在服务器 160 内的执行所需的任何成份的程序包，并将该程序包发送给接口任务 484。接口任务 484 接收到该程序包，将其发送给客户机 100，客户机 100 应用该程序包来执行程序。

F. 接口任务

图 5 是由依据本发明的封装过程中的接口任务所执行的步骤的流程图。在过程开始时，接口任务 484（在服务器 160 上执行）接收为一指定客户机、例如客户机 100 上的执行封装 Java 程序的请求（步骤 510）。Java 程序一般被指定为是 Java 类和成份的集合。一些类文件的位置是预先定义的。其他所必需的类的位置被包括在封装 Java 程序的请求中。

响应于这个请求，接口任务 484 为链接编辑器形成一组指令。作为这些指令的一部分，接口任务通知链接编辑器 487 关于程序的开始点（步骤 515）。这

给了链接编辑器 487 一个用于确定需要哪些成份的开始点。

接口任务 484 还通知链接编辑器 487 任何已经存在于客户机 100 上的类(步骤 520)。通过提供这个信息，接口任务避免了不必要的封装：链接编辑器 487 不需要封装已经存在于客户机 100 上的成份。这通过使输出文件的大小最小而节省了使用的带宽。该信息可以作为封装成份请求的一部分提供给接口任务(步骤 510)。或者，该信息可以被预先提供，存储在服务器 160 的存储器 482 内。为了使这一步骤能正常工作，已经存在于客户机 100 上的类与在服务器 160 上的那些必须是一致的。

接口任务 484 还通知链接编辑器 487 关于任何应该增加到程序包中的附加成份(步骤 530)。这对于有些情况是必须的，例如，如果由于包含一个成份的类已经存在于客户机 100 上、则该成份被排除在链接步骤之外的话；然而，这个被排除的成份可能会引用在客户机 100 上没有的其他成份，因此可能会需要由链接编辑器 487 增加到程序包内。除了由于其类已经存在于客户机 100 上而被排除在外的成份之外，还可能有一些依赖性不能被有计划地发现。接口任务 484 一般(在步骤 510)被通知将任何附加成份作为封装成份请求的一部分。

例如，方法 quicksort 可以被排除在链接步骤之外，因为包含该方法的类 sun.misc.Sort 已经被加载在客户机 100 上了。当方法 quicksort 被调用时，它的一个参数是具有方法 doCompare 的对象。Quicksort 将调用 doCompare，而 doCompare 可能还未被加载在客户机 100 上。因此，接口任务 484 必须通知链接编辑器 487 方法 doCompare 必须作为一个附加成份被加载。

接口任务 484 还通知链接编辑器 487 用于查找所需 Java 类文件的位置列表(步骤 540)。接口任务 484 将这些指令发送给链接编辑器 487(步骤 550)，链接编辑器 487 产生一个输出文件，这将在下面参考图 5 进行说明。接口任务 484 接收该输出文件，并将其发送给客户机 100(步骤 560)，该过程结束。

G. 链接编辑器

图 6 是依据本发明的由链接编辑器 487 在封装输出文件中执行的步骤的详细流程图。出于例示的目的，下面对流程的说明是基于图 2 的例子的。

在过程开始时，链接编辑器 487(一般在服务器 160 上执行)从接口任务 484 接收一组指令。链接编辑器 487 然后生成并以要执行的程序的开始点初始化一个列表(步骤 605)。该列表被称为“成份列表”，包含对每个必须由链接编辑器 487 加载的所需成份的引用。

链接编辑器 487 然后选择在成份列表中的下一项，最初将为 main 方法（步骤 610）。链接编辑器 487 查看所选成份是否在要从链接步骤中排除的项目列表上（步骤 615）。成份可以在要排除的项目列表上，例如，因为该成份更容易从其他来源、例如客户机 100 或其他服务器获得。因为 main 方法不会在要排除的项目列表上，所以链接编辑器 487 查看以确定所选择成份是否以前已经加载进链接过程（步骤 620）。因为 main 方法不会已经在以前加载过，则链接编辑器 487 采用由服务器在其给链接编辑器 487 的指令中提供的文件位置列表对包含要加载的成份的类文件进行定位（步骤 625）。链接编辑器 487 读取类文件、提取与该类文件相联系的任何辅助信息、从该类文件提取出所选择成份、并将提取出的辅助信息和成份增加到一个输出文件（步骤 630）。应该注意，通过这样做，链接编辑器 487 只提取出所需成份和辅助信息，而不是加载整个类。

在加载了提取出的成份之后，链接编辑器 487 检查被替代的方法（步骤 632）。该步骤将在下面参考图 7 进一步说明。接着，链接编辑器 487 分析提取出的成份，以确定其是否引用了其他成份（步骤 635）。换句话说，链接编辑器 487 分析提取出的成份的依赖性。在这个例子中，main 方法引用了四个成份：A、B、C 和 D。因为提取出的成份（main）包含依赖性（步骤 640），所以将这些依赖性增加到成份列表中（步骤 645），然后（在步骤 610）重复该过程。

然后对于成份 A、B、C 和 D 进行与上述的关于 main 方法相同的过程。假设成份 D 被包含在一个接口任务 484 指示链接编辑器 487 从封装过程中排除的类中（例如，因为该类已经存在于客户机 100 上），则成份 D 将被排除在输出文件之外（步骤 615）。另一方面，成份 A、B 和 C 将被从它们的各个类中（与这些类相联系的辅助信息一起）提取出来，并增加到输出文件中（步骤 625-630）。当这些成份被提取和分析时，链接编辑器 487 将发现，成份 B 引用了成份 B1，成份 C 引用了成份 A 和 C1（步骤 635）。因此成份 B1、A 和 C1 将被增加到成份列表中（步骤 645），该过程再一次进行重复（步骤 610）。

在这下一次循环中，将不提取成份 A，因为它已经被增加到输出文件中了（步骤 620）。但成份 B1 和 C1 将被从它们各自的类中提取出来，与它们的类相联系的辅助信息一起，增加到输出文件中（步骤 625-630）。链接编辑器 487 将发现，成份 C1 引用了成份 C1A（步骤 635），于是将其增加到成份列表（步骤 645）。过程再重复一次，在此期间，成份 C1A 被从其相应的类中（与该类的辅助信息一起）提取出来，增加到输出文件中（步骤 625-630）。

通过应用这个迭代过程，链接编辑器 487 生成一个包含驻留在服务器 160 中的执行程序所必需的所有成份（及辅助类信息）的输出文件。链接编辑器 487 将这个输出文件发送到接口任务 484（步骤 655），该过程结束。

图 7 是依据本发明为了检查被替代的方法、由链接编辑器 487 执行的步骤的流程图。替代是一种用于将一个方法的超类的实施以子类的实施来代替的技术。为了检查被替代的方法，链接编辑器 487 首先确定提取出的成份是一个方法还是数据（步骤 705）。因为替代问题仅仅与是方法的成份有关，如果提取出的成份是数据的话，则不必进行任何操作。另一方面，如果提取出的成份是方法，链接编辑器 487 确定该方法是否是一个构造程序（即，它是否生成一个对象）（步骤 710）。

如果提取出的方法不是一个构造程序，链接编辑器 487 接着确定该方法是否是一个静态方法（步骤 715）。如果它是静态的，则不必进行任何操作，因为静态方法不能被替代。如果该方法不是静态的，则链接编辑器 487 查看该提取出的方法是否已被替代。特别地，链接编辑器 487 搜索（提取出的方法的类的）已经具有一个被加载的构造程序的所有子类。如果这些子类中的任何一个具有替代了所提取方法的方法，则必须将这个替代的方法加到成份列表中（步骤 720）。

如果提取出的方法是一个构造程序，链接编辑器 487 搜索所提取方法的类，以确定其是否包含替代了以前加载的方法的任何方法。特别地，链接编辑器 487 为非静态方法（静态方法不能被替代）搜索包含所提取成份的类（步骤 725）。如果这些非静态方法中的任何一个替代了以前从所提取成份的类的超类加载的一个方法（步骤 730），则替代的非静态方法必须被增加到成份列表中（步骤 735）。上述方法不仅用于被替代的方法，还用于作为 Java 接口说明的一部分的方法。

H. 客户机过程

图 8 是依据本发明由执行一个程序的客户机执行的步骤的流程图。首先，客户机 100 发出一个开始执行指定程序的请求（步骤 810）。响应于该请求，客户机 100 从服务器 160 接收一个包含程序执行所需的所有成份的封装文件（即，上面提到的输出文件）。客户机 100 将所提供的成份增加到那些已经存在于客户机 100 上的成份中（步骤 830）。特别地，客户机 100 从封装文件中读取信息，建立内部数据结构，并将这些数据结构链接成它自己的类名列表。客户机 100 然后开始执行程序（步骤 840），根据需要分辨（resolve）类名。该过程然后结束。

I. 结论

上述系统只将程序执行所必须的成份封装在一起，并在执行开始之前将它们传送给该客户机。上面对本发明的一个实施形式的说明只是为了例示和说明的目的。例如，所述的实施例包括软件，但本发明也可以被实现为硬件与软件的结合或单独以硬件实现。在上述说明的提示下，可以作出各种修改和变化，或者可以从本发明的实践中得出。

虽然依据本发明的系统和方法是在例示性的分布式系统和 Java 程序设计环境下操作的，本领域普通技术人员应该理解，本发明可以在其他系统和程序设计环境中实现。另外，虽然本发明的各个方面被描述为存储在存储器中，本领域普通技术人员应该理解，这些方面也可以存储在其他类型的计算机可读介质中，例如二级存储设备，象硬盘、软盘或 CD-ROM；来自因特网的载体；或其他形式的 RAM 或 ROM。本发明的范围由附带的权利要求及其等效物限定。

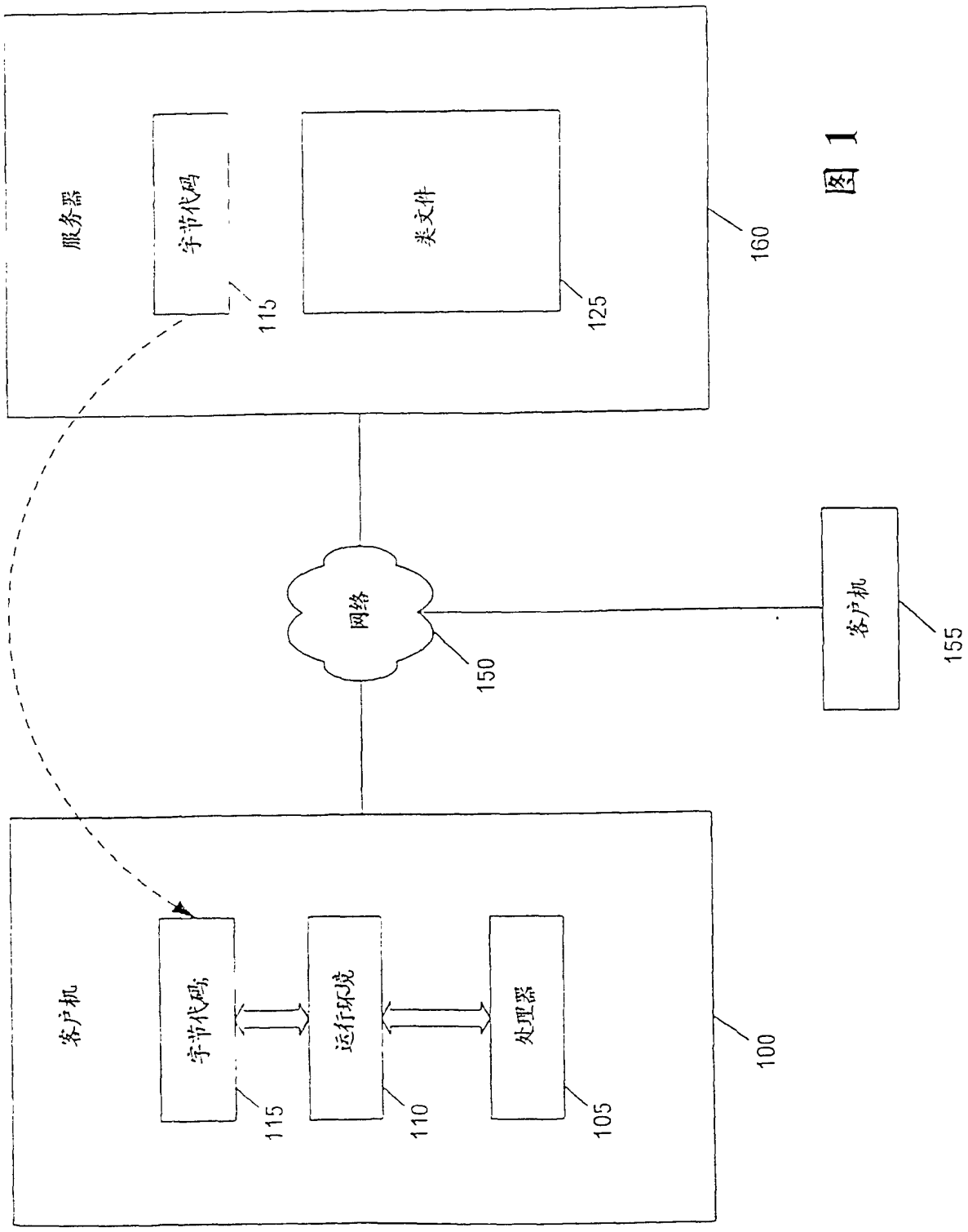

服务器
字节代码
115
类文件
125
160
网络
150
客户机
155
客户机
字节代码
115
运行环境
110
处理器
105
100

图 1

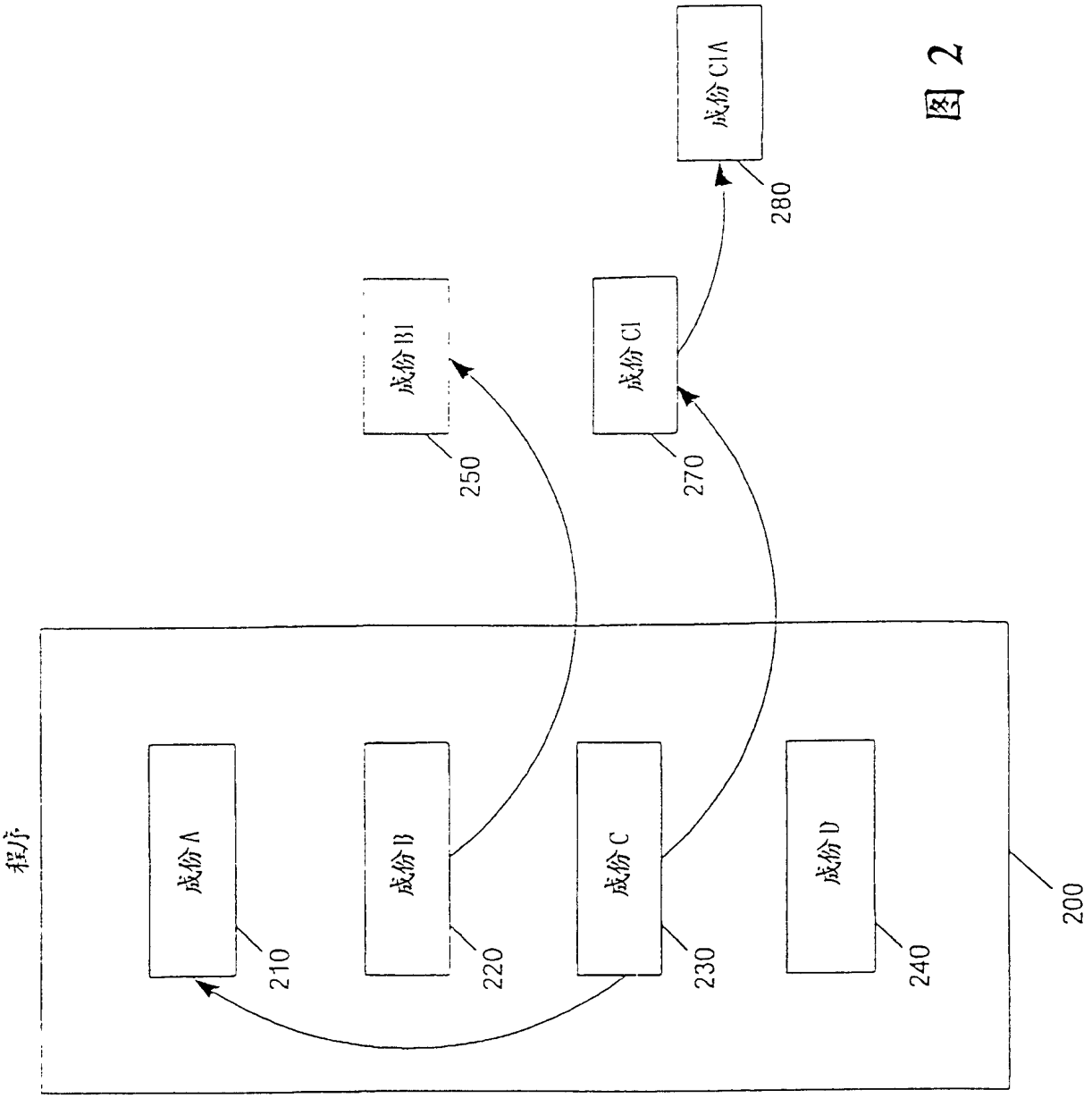

程序
成份A
210
成份B
220
成份C
230
成份D
240
200
成份B1
250
成份C1
270
成份C1A
280

图 2

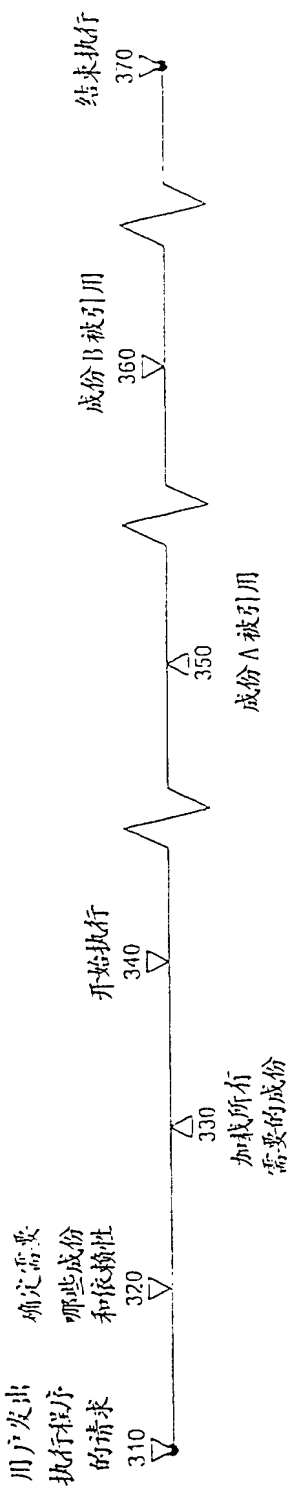

用户发出执行程序的请求
310
确定需要哪些成份和依赖性
320
加载所有需要的成份
330
开始执行
340
成份A被引用
350
成份B被引用
360
结束执行
370

图 3

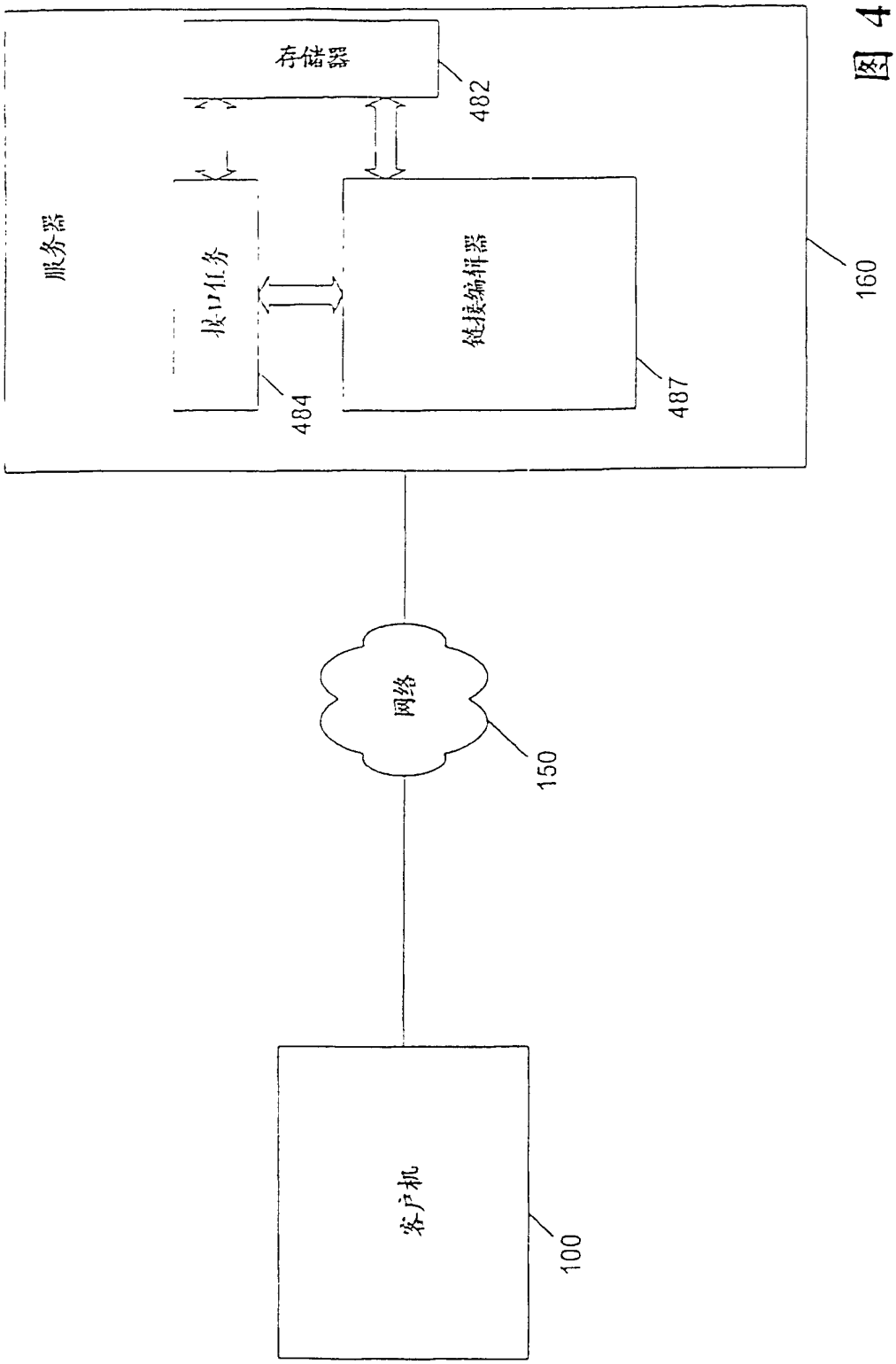


图 4

图 5

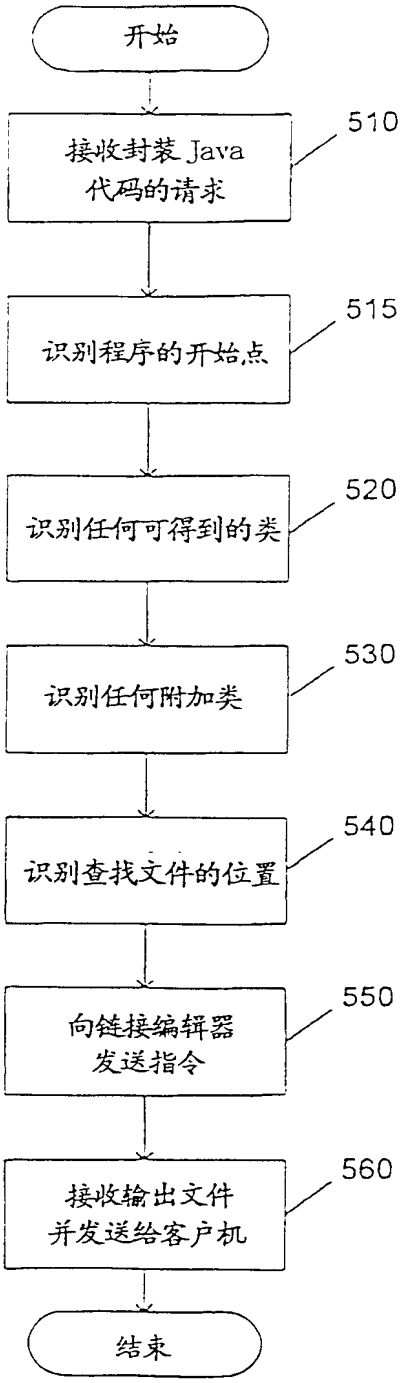

开始
接收封装 Java 代码的请求
510
识别程序的开始点
515
识别任何可得到的类
520
识别任何附加类
530
识别查找文件的位置
540
向链接编辑器发送指令
550
接收输出文件并发送给客户机
560
结束

图 6

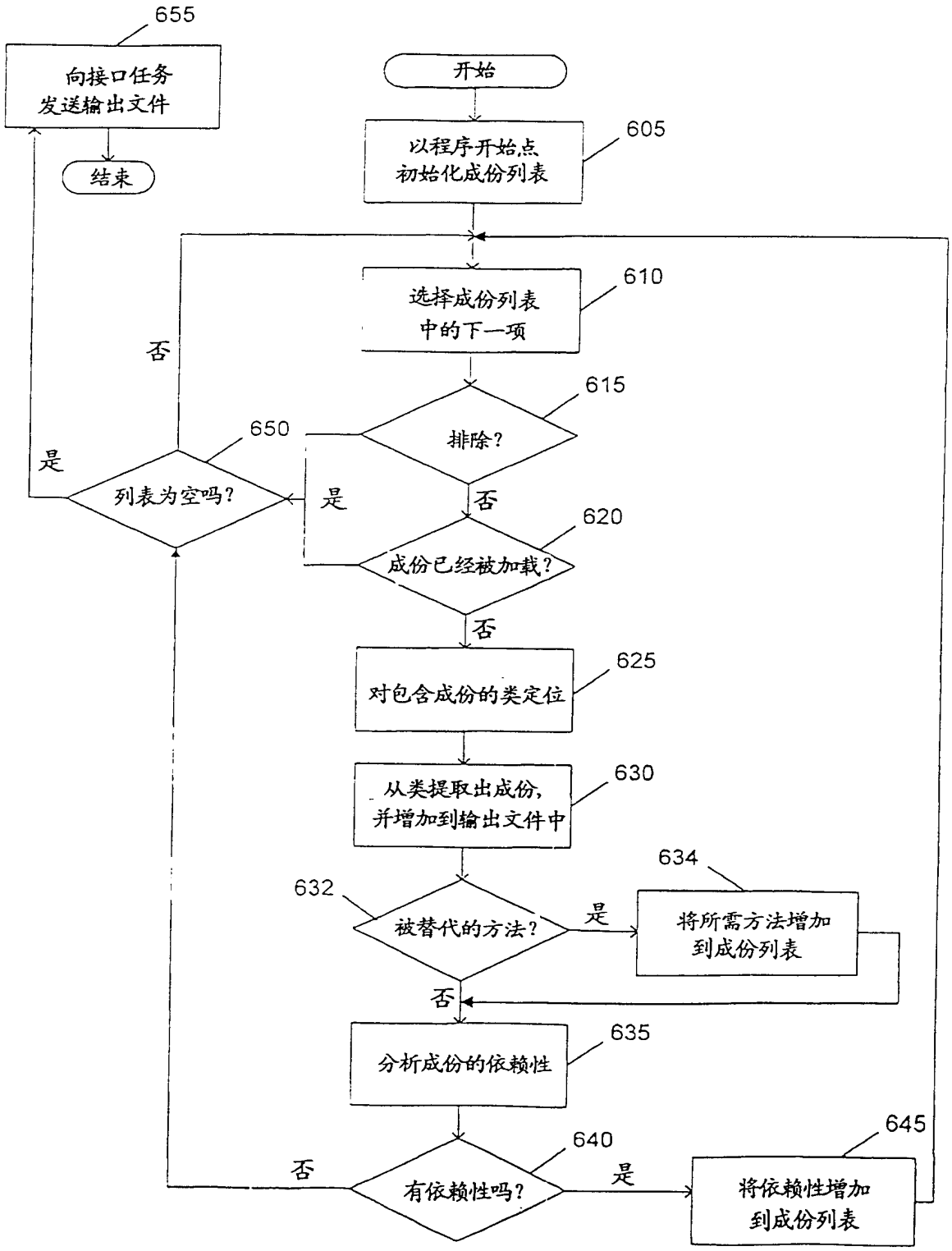

开始
以程序开始点初始化成份列表
605
选择成份列表中的下一项
610
排除?
615
否
是
成份已经被加载?
620
否
是
列表为空吗?
650
是
否
对包含成份的类定位
625
从类提取出成份,并增加到输出文件中
630
被替代的方法?
632
是
将所需方法增加到成份列表
634
否
分析成份的依赖性
635
有依赖性吗?
640
否
是
将依赖性增加到成份列表
645
向接口任务发送输出文件
655
结束

图 7

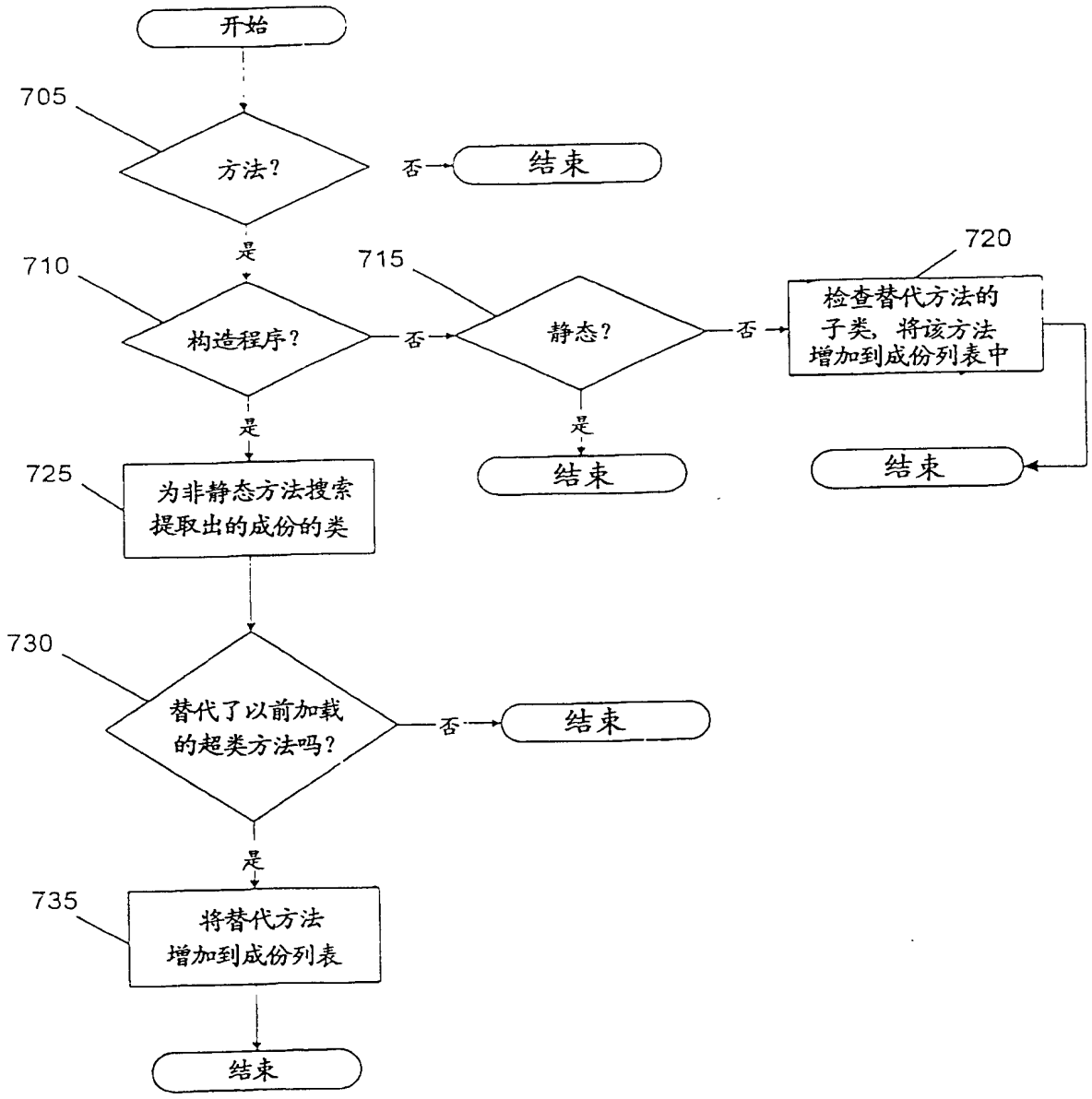
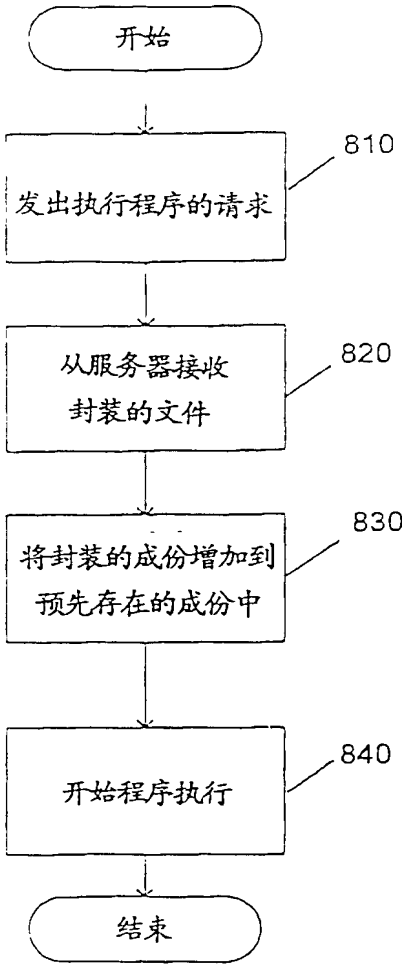

开始
705
方法?
否
结束
是
710
构造程序?
否
715
静态?
否
720
检查替代方法的
子类，将该方法
增加到成份列表中
是
是
结束
结束
725
为非静态方法搜索
提取出的成份的类
730
替代了以前加载
的超类方法吗?
否
结束
是
735
将替代方法
增加到成份列表
结束

图 8


开始
发出执行程序的请求
810
从服务器接收
封装的文件
820
将封装的成份增加到
预先存在的成份中
830
开始程序执行
840
结束