[19]中华人民共和国国家知识产权局

[51]Int. Cl[7]
G06F 9/445

[12]发明专利申请公开说明书

[21]申请号 00120239.1

[43]公开日 2001年1月24日

[11]公开号 CN 1281180A

[22]申请日 2000.7.14 [21]申请号 00120239.1

[30]优先权

[32]1999.7.15 [33]US [31]09/353880

[71]申请人 国际商业机器公司

地址 美国纽约州

[72]发明人 P·J·米勒 E·C·史密斯

T·J·沃尔夫

[74]专利代理机构 中国专利代理(香港)有限公司

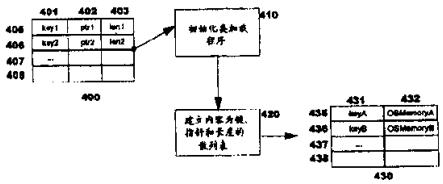
代理人 吴立明 王忠忠

权利要求书3页 说明书13页 附图页数9页

[54]发明名称 为应用程序透明地加载只读存储器中的资源

[57]摘要

一种为Java虚拟机上执行的应用程序透明地从只读存储器加载资源的方法、系 统和计算机程序产品。本发明一般将在没有磁盘存储器、没有文件系统软件的 嵌入式计算设备上使用。本文所述的新颖的类加载程序,从ROM加载资源并 按应用程序的预期返回输入流对象。所以,在使用本发明时,不需要改动现有 应用程序代码就能使用ROM中存储的资源。本发明可以用于以Java程序设计 语言编写的程序,或者用于在JVM上执行的以其它语言编写的程序。



ISSN1008-4274

00·07·14

权 利 要 求 书

1. 一种在计算机可读的媒体上体现的、由某计算环境中的计算机可读的、用于透明地从计算机的只读存储器（“ROM”）加载资源的计算机程序产品，包含:

存储在 ROM 中的若干资源文件;

用于在计算机上执行一个 Java 虚拟机（“JVM”）的计算机可读的程序代码装置;

用于在 JVM 上执行一个应用程序的计算机可读的程序代码装置;

用于在 JVM 上执行一个 ROM 类加载程序的计算机可读的程序代码装置—ROM 类加载程序适合从 ROM 加载资源文件;

用于创建包含每个资源文件的对应项的表的计算机可读的程序代码装置;

用于由 ROM 类加载程序接收应用程序对选定一个资源文件上的输入流的请求的计算机可读的程序代码装置;

用于利用表在 ROM 中定位选定资源文件的计算机可读的程序代码装置; 和

用于利用找到的资源返回输入流的计算机可读的程序代码装置;

2. 按照权利要求 1 的透明地加载资源的计算机程序产品，其中，表中的每项包含:

与特定一个资源文件的标识符对应的键值;

特定资源文件在 ROM 中的位置;

在特定资源文件所处位置存储的数据的长度; 并且，利用表的计算机可读的程序代码装置进一步包含:

用于用特定资源文件的标识符作为键值来访问表的计算机可读的程序代码装置;

用于在由用于访问的计算机可读的程序代码装置在表中找到匹配的项时返回特定资源文件的位置和长度的计算机可读的程序代码装置。

3. 按照权利要求 1 的透明地加载资源的计算机程序产品，进一步包含用于将 JVM 配置得能使用 ROM 类加载程序的计算机可读的程序代码装置。

4. 按照权利要求 1 的透明地加载资源的计算机程序产品，其中，请求保持应用程序中所规定的已有的请求语义不变。

5. 一个在计算环境中透明地从计算环境中的计算机的只读存储器（“ROM”）加载资源的系统，包含：

存储在 ROM 中的若干资源文件；

用于在计算机上执行一个 Java 虚拟机（“JVM”）的装置；

用于在 JVM 上执行一个应用程序的装置；

用于在 JVM 上执行一个 ROM 类加载程序的装置—ROM 类加载程序适合从 ROM 加载资源文件；

用于创建包含每个资源文件的对应项的表的装置；

用于由 ROM 类加载程序接收应用程序对选定一个资源文件上的输入流的请求的装置；

用于利用表在 ROM 中定位选定资源文件的装置；和

用于利用找到的资源返回输入流的装置；

6. 按照权利要求 5 的透明地加载资源的系统，其中，表中的每项包含：

与特定一个资源文件的标识符对应的键值；

特定资源文件在 ROM 中的位置；

在特定资源文件所处位置存储的数据的长度；并且，利用表的装置进一步包含：

用于用特定资源文件的标识符作为键值来访问表的装置；

用于在由用于访问的装置在表中找到匹配的项时返回特定资源文件的位置和长度的装置。

7. 按照权利要求 5 的透明地加载资源的系统，进一步包含用于将 JVM 配置得能使用 ROM 类加载程序的装置。

8. 按照权利要求 5 的透明地加载资源的系统，其中，请求保持应用程序中所规定的已有的请求语义不变。

9. 一种透明地从计算环境中的计算机的只读存储器（“ROM”）加载资源的方法，包含的步骤为：

在 ROM 中存储若干资源文件；

在计算机上执行一个 Java 虚拟机（“JVM”）；

在 JVM 上执行一个应用程序；

在 JVM 上执行一个 ROM 类加载程序—ROM 类加载程序适合从 ROM 加载资源文件;

创建包含每个资源文件的对应项的表;

由 ROM 类加载程序接收应用程序对选定一个资源文件上的输入流的请求;

利用表在 ROM 中定位选定资源文件;

利用找到的资源返回输入流;

10. 按照权利要求 9 的透明地加载资源的方法，其中，表中的每项包含:

与特定一个资源文件的标识符对应的键值;

特定资源文件在 ROM 中的位置;

在特定资源文件所处位置存储的数据的长度; 并且，利用表的步骤进一步包含的步骤为:

用特定资源文件的标识符作为键值来访问表;

在由访问步骤在表中找到匹配的项时返回特定资源文件的位置和长度。

11. 按照权利要求 9 的透明地加载资源的方法，进一步包含将 JVM 配置得能使用 ROM 类加载程序的步骤。

12. 按照权利要求 9 的透明地加载资源的方法，其中，请求保持应用程序中所规定的已有的请求语义不变。

00·07·14

说　明　书

为应用程序透明地
加载只读存储器中的资源

本发明涉及计算机系统，更具体来说涉及为 Java 虚拟机上执行的应用程序透明地从只读存储器（“ROM”）加载资源（诸如所存储的位图、图象、字体和声音文件）的方法、系统和计算机程序产品。

Java 是由太阳微系统公司开发的一种强健、可移植、面向对象的程序设计语言，在编写因特网和万维网（以下简称 WEB）程序的方面正在获得广泛的接受。大多数程序设计语言的编译程序生成适用于特定操作环境的代码，而 Java 则使得用“一次写成，处处运行”（Write Once, Run Anywhere）的范例编写程序成为可能。（“Java”和“Write Once, Run Anywhere”是太阳微系统公司的商标。）

Java 是通过使用专门设计的虚拟机（“VM”）而得到其可移植性的。这个虚拟机也称作“Java 虚拟机”或“JVM”。虚拟机使潜在硬件的细节与用于编译 Java 程序设计指令的编译程序得以分离。这些细节是通过虚拟机的实现而提供的，包括诸如在运行 Java 程序的机器上使用的是小 Endian 还是大 Endian 格式等等。因为这些从属于机器的细节不在编译代码中反映，所以能将代码移植到不同的环境（不同的硬件机器、不同的操作系统，等等），并在该环境中执行代码，而不要求改编或重新编译代码—所以就有了“一次写成，处处运行”这个说法。被称为 Java“字节码”的编译代码然后在 JVM 的顶上运行—其中 JVM 是按具体的操作环境改编的。这种具体改编 JVM 的例子是，由于字节码是按规范格式（大 Endian）生成的，如果 JVM 在是小 Endian 的机器上运行，JVM 就要负责先将指令从字节码转换，再将它们传送给微处理器。

Java 运行环境包括 JVM 以及运行 Java 应用程序或小应用程序所需要的若干文件和类。自此，除非另外注明，术语“JVM”和“运行环境”将互可交换地在本文中使用。

Java 应用程序通常是从诸如太阳微系统公司的“JDK”（Java 开发工具箱）产品的开发工具包、或用也是出自太阳微系统公司的

"JRE"（Java 运行环境）产品被执行的。JRE 是 JDK 的子集，提供应用程序执行所需的功能。当采用 JRE 时，程序是从命令行执行的。

被称为"类加载程序"的 Java 类在 Java 环境中用来动态地加载运行程序中的类和资源。图 1 表示使用 JVM 和类加载程序的类加载的现有水平技术。JVM 的类加载程序功能使 Java 应用程序能随着程序的执行而递增地加载。本领域中众所周知，程序员编写 Java 程序后，将其编译成 Java 字节码。含有 Java 字节码的文件被称为"类文件"。程序员 100 然后将类文件装入类文件的储存库 110 或 111。在以后的某个时刻，应用程序 150 在客户计算机 160 上被 JVM 140 执行。当应用程序 150 试图使用某个在客户计算机 160 上尚未调入的类时，JVM 140 的类加载程序部件 130 会向类服务器 120 发出请求 102a。（类服务器功能 120 通常包含在标准 Web 服务器内。）这个请求 102a 通知类服务器 120 从储存库 110 提取(103a、104a)该类文件，并将其返回给（105a）JVM 140。或者，类加载程序 130 也可以在本机文件系统 111 的目录中查找所需要的类。在这种情况下，要从文件系统 111 请求(102b)所需的类并将其返回给(105b)给 JVM 140。不管文件是从位置 110 还是 111 中检索出来的，应用程序 150 然后都要用检索出的类文件继续执行。这种对类文件的动态加载，操作上对应用程序 150 的用户是透明的。

Java 类加载程序在查找文件时采用预定的检索策略，对特定位置予以优先考虑。按照 Java 1.2 平台规范，最高优先级的检索位置是自引导的运行和国际化类，分别称为"rt.jar"和"i18n.jar"。如果在这些位置找不到所需的类，次高优先级要在安装的扩展类中查找—扩展类是在 JRE 的"lib/ext"目录中存储的 JAR 中的类。（"JAR"指"Java 档案库"，是用于对应用程序所用文件进行分布和存档的一种文件格式。）最后，如果还没有找到该类，就要考虑类路径系统的属性设置。由类路径规定的任何路径，以及在用这些路径找到的 JAR 文件的清单的类路径属性中标识的任何路径，都可以用来检索所需的类。（欲知详细信息，可在网上参看"认识扩展类加载"(Understanding Extentsion Class Loading)一文，网址为。）此外，可以通过（用目录和文件名标识）指定类文件在文件系统中的位置从已知位置来加载类文件，或者通过指定类文件的统

一资源定位器（URL）从网络服务器上的位置来加载类文件。

现有技术的动态加载方法也用于对 Java 应用程序所用资源文件的检索和加载。资源可能已经单独地存储在存储可执行类文件的同一个目录中；它们可能被包装在包装类文件的同一个 JAR 文件中；或者，它们可以与类文件一起存储在公用的目录结构中，但被收集到单独的子目录（诸如“图象”或“资源”子目录）中。

现有的类加载程序功能假设存在文件系统，并且 CLASSPATH 环境变量或“java.class.path”系统属性将确定在该文件系统中类加载程序能动态检索所需类文件和资源的某个位置。然而，人们正在开发大量为方便移动或便携使用的新型计算设备。这些设备要设计得重量轻、体积小、效率高（从费用角度和操作角度而言），所以许多设备都被设计得没有磁盘驱动器的系统开销，没有用来存取磁盘文件的文件系统软件。（此外，这些设备经常被配置以相对较小的存储器，容量数量级在几兆字节。）本文自此将这些设备称为“嵌入设备”，它们的例子包括个人数字助理（PDA）；蜂窝电话和可视电话（screen phones）、寻呼机以及便携式（wearable）计算设备。

这些嵌入设备如果没有磁盘驱动存储器，可以将资源文件存储在只读存储器中。然而如前文所述，现有的 Java 类加载程序机构优先检索在文件系统中存储的文件。由于嵌入设备没有文件系统，所以要为嵌入设备提供一种可替代的技术。这个技术必须能高效地查找出在嵌入设备上执行的应用程序所需要的资源。在名称为“访问 ROM 中存储的资源文件”（Access to Resource Files Stored in ROM）的待审定美国专利申请（序列号 09/___，受让给本发明的同一个受让人，本文引用作为参考）中，描述了一种技术。然而，该技术要求修改现有应用程序代码来专门检索 ROM 存储器中的资源。尽管这种方法在特定情况中是有益的，但如果换一种方法能避免修改应用程序代码则会更好。修改应用程序代码是人们不希望做的事情，这是因为它有导入错误的可能，需要进行额外的测试来验证改编后的代码，也因为成百上千的应用程序可能已经广泛地散布到千千万万的用户手中，使应用程序修改成为难以控制的任务。

因此，需要一种技术，现有应用程序用它能透明地（不加改动地）访问已经在 ROM 中存储的资源。这种技术应当是后向兼容的，

使得在具有文件系统的现有计算环境中执行的程序不受影响。

本发明的一个目的是提供一种现有应用程序代码能通过其透明地加载 ROM 中存储的资源的技术。

本发明的另一个目的是通过一种可替代的类加载机制来提供这种技术。

本发明的另一个目的是以与资源不在 ROM 中存储的现有计算环境后向兼容的方式来提供这种技术。

本发明的另一个目的是以不要求用户行动或不要求用户意识到该可替代的加载机制或策略的方式来提供这种技术。

本发明的其它目的和优点，部分将在本说明和以下的各附图中陈述，部分将通过本说明阐明或者通过对本发明的实践得到了解。

为了实现以上目的，按照本文广义地说明的本发明的目的，本发明提供一种用于在计算环境中从 ROM 存储器透明地加载资源的方法、系统和计算机程序产品。这个技术包含：在 ROM 中存储多个资源文件；在计算机上执行 Java 虚拟机（“JVM”）；在 JVM 上执行应用程序；在 JVM 上执行 ROM 类加载程序—ROM 类加载程序要适合从 ROM 加载资源文件；创建包含对应每个资源文件的项的表；由 ROM 类加载程序接收应用程序对选定的一个资源文件上的输入流的请求；利用表定位 ROM 中所选择的资源文件；利用查找到的资源返回该输入流。

表中的每项最好包含：与特定一个资源文件的标识符对应的键值；该特定资源文件在 ROM 中的位置；在该特定资源文件的位置存储的数据的长度。对表的使用最好还包含：用该特定资源文件的标识符作为键值来访问表；访问表时如果发现匹配的项就返回该特定资源文件的位置和长度。

这个技术最好进一步包含将 JVM 配置得能使用 ROM 类加载程序。此外，该请求最好原封不动地保留最好是在应用程序中规定的预先存在的语义。

现在将结合以下附图来说明本发明。各附图中相同的标注号自始至终表示同一个部件。

图 1 表示现有技术水平用 Java 虚拟机进行类加载的技术；

图 2 是可以在其中实践本发明的计算机工作站环境的框图；

图 3 是可以在其中实践本发明的连网的计算环境的框图；

图 4 至 6 表示的流程图叙述的是实现本发明最佳实施例时使用的逻辑。

图 7 至 9 表示的是可用来实现本发明最佳实施例的 Java 程序设计语言样本。

图 2 表示一个可以在其中实践本发明的代表性的工作站硬件环境。图 2 的环境包含一个代表性的、包括有关外围设备在内的用户计算机工作站 210—例如个人电脑。工作站 210 包括微处理器 212 和总线 214，后者用于按已知技术连接微处理器 212 与工作站 210 的部件并使它们之间能进行通信。工作站 210 通常包括用户接口适配器 216，它将微处理器 212 通过总线连接到一个或多个接口设备，诸如键盘 218、鼠标器 220 和/或其它接口设备 222—后者可以是任何用户接口设备，诸如触摸屏、数字输入键盘、等等。总线 214 也将显示器 224（诸如 LCD 屏幕或监视器）通过显示适配器 226 与微处理器 212 相连。总线 214 还将微处理器 212 连接到存储器 228 和长期存储器 230，后者可包括硬盘驱动器、软盘驱动器、磁带驱动器，等等。

工作站 210 例如可以通过某个通信频道或调制解调器 232 与其它计算机或计算机网络进行通信。工作站 210 或者也可以用 232 表示的无线接口—诸如 CDPD（蜂窝数字包数据）—进行通信。工作站 210 可以与局域网（LAN）或宽域网（WAN）中的其它计算机联系在一起，工作站 210 或者也可以是在与另一个计算机的客户机/服务器组合中的客户机，等等。所有这些配置以及适当的通信硬件和软件，都是本领域中已知的。

图 3 表示可以在其中实践本发明的数据处理网络 240。数据处理网络 240 可以包括若干个别的网络，诸如无线网络 242 和网络 244，它们每个都包含若干个别的工作站 210。此外，本领域的熟练人员知道，可以包含一个或多个 LAN（未予示出），其中，LAN 可以包含若干与宿主处理机相连的智能工作站。

继续参看图 3，网络 242 和网络 244 也可以包括主计算机或服务器，诸如网关计算机（gateway computer）246 或应用服务器 247（它可以访问数据储存库 248）。网关计算机 246 起着进入每个网络 244 的入口点的作用。网关 246 最好通过通信链路 250a 与另一个网络 242

相连。网关也可以用通信链路 250b、250c 直接连接到一个或多个工作站 210。网关计算机 246 可以用 IBM 公司的企业系统体系结构/370（Enterprise Systems Architecture/370）、企业系统体系结构/390（Enterprise Systems Architecture/390）计算机等来实现。视具体应用而定，可以采用中等的计算机，诸如应用系统/400（Application System/400）（也叫 AS/400）。（Enterprise Systems Architecture/370 是 IBM 的商标，Enterprise Systems Architecture/390、Application System/400 和 AS/400 是 IBM 的注册商标。）

网关计算机 246 也可以连接 249 到存储器（诸如数据储存库 248）。此外，网关 246 可以直接或间接地连接到一个或多个工作站 210。

本领域的熟练人员将知道，网关计算机 246 可以位于地理位置远离网络 242 的地方，类似地，工作站 210 可以位于与网络 242 和 244 有相当距离的位置。例如，网络 242 可以位于美国加州，而网关 246 可以位于德克萨斯，一个或多个工作站可以位于纽约。工作站 210 可以用诸如传输控制协议/网际协议（TCP/IP）的连网协议，经过若干可选择的连接媒介（诸如蜂窝电话、无线电频率网络、卫星网络等等），与无线网络 242 连接。无线网络 242 最好用诸如 TCP 或 UDP（用户数据报协议）的网络连接 250a，经过 IP、X.25、帧中继、ISDN（综合业务数字网）、PSTN（公共交换电话网）等，与网关 246 连接。工作站 210 也可以选择用拨号连接 250b 或 250c 直接连接到网关 246。此外，无线网络 242 和网络 244 可以以图 3 中所示的类似的方式连接到一个或多个网络（未予示出）。

体现本发明的软件程序代码，一般是由工作站 210 的微处理器 212 从某种类型的长期存储介质（诸如 CD-ROM 或硬盘驱动器）中访问的。软件应用程序可以收录在已知各种用于数据处理系统的媒体的任何一种上（诸如软盘，硬盘驱动器或 CD-ROM）。代码可以在这类媒体上传播，或者可以经过某种类型的网络从一个计算机系统的内存或外存传播到其它计算机系统，供这些其它系统的用户使用。或者，程序代码也可以在包含在存储器 228 上，由微处理器 212 用总线 214 来访问。在存储器中、物理媒体上收录软件程序代码和/或

通过网络传播软件代码的技术和方法是众所周知的，本文将不作更深的讨论。

本发明可以在不与网络连接的、以独立方式运行的用户工作站上使用。或者，本发明可以在与网络相连的用户工作站上使用。当本发明用在客户机-服务器连网环境中时，本发明运行所在的客户计算机可以用导线连接或无线连接与服务器相连。有线连接是采用诸如电缆和电话线的物理媒体的连接；而无线连接采用的媒体诸如是无线链路、无线电频率波和红外波。许多连接技术都可以用于这些各种媒体，例如：用计算机的调制解调器经过电话线建立连接；用 LAN 卡，诸如令牌环或以太网；用蜂窝式调制解调器建立无线连接；等等。客户计算机可以是具有处理功能的（也可以有通信功能的）任何类型的计算机处理机，包括膝上型、手提式或移动式计算机；车载设备；桌面计算机；大型机；等等。类似地，远程服务器可以是具有处理和通信功能的众多不同类型的计算机中的任何一种。这些技术在本领域是众所周知的，硬件设备和能使硬件有用的软件很容易获得。自此，将在相同的意义上把客户计算机称为“工作站”、“机器”、“设备”或计算机，任何这些术语或术语“服务器”都是指上述的任何类型的计算设备。

在最佳实施例中，本发明是以计算机软件程序（或程序）的一个或多个模块（也称为代码子例程，或者，在面向对象的程序设计中称为“对象”）实现的。这个实现通常将驻留在嵌入设备上并在嵌入设备上执行。或者，它也可以在诸如没有磁盘存储器和文件系统的桌面计算机的客户工作站上运行，而不会对程序在这个环境中的运行产生不利的影响。本发明可以用于因特网环境中的工作站。或者，环境也可以是公司内部网、外部网（extranet）或任何其它网络环境。或者，本发明可以在独立的环境中使用。在连网的环境中使用时，本发明的代码在客户机设备上运行。最佳实施例的程序代码最好以 Java 面向对象程序设计语言中的对象来实现。或者，本发明也可以与在 Java 虚拟机上执行的、以任何程序设计语言（诸如 Javascript 或 NetRexx）编写的程序代码一起使用。（Javascript 是太阳微系统 s 公司的商标，NetRexx 是 IBM 公司的商标。）本发明也可以用于具有语义与本文所描述的 Java 类加载程序的语义类似的

动态部件加载机制的其它环境。

最佳实施例提供的透明地从 ROM 加载资源的方法，要使用本文定义的一下文将结合图 4-9 作更详细讨论的一一组部件，以后将会说明，它们是一起工作的。这些部件是：(1)在本文中称为“OSMemory”的对象类，它使应用程序代码能访问 ROM 存储器；（2）在本文中称为“OSMemoryInputStream”的对象类，它是 java.io.InputStream 类的子类，为 OSMemory 对象提供流接口；（3）在本文中称为“RomLoadingClassLoader”的对象类，它是一个新颖的 Java 类加载程序，知道如何从 ROM 向外加载资源。

如前文结合图 1 所述，类加载程序在 Java 应用中，用来在执行的应用程序需要类或资源时，动态地加载类和资源；当程序请求资源或调用类时，正是 ClassLoader 去寻找并返回该资源或类。

在现有技术的应用程序代码中，访问资源通常要针对某个类调用 Class.getResouceAsStream()过程。这个过程将资源的名称作为参数。Class.getResouceAsStream()过程以调用 ClassLoader.getResouceAsStream()过程而结束。每个类都有一个与其相关联的 ClassLoader 对象。各种 getResouceAsStream()过程返回一个 InputStream 类型的对象。这个 InputStream 对象是应用程序代码用来读资源数据的标准接口。

在这种现有技术的从应用程序代码加载资源方法（如前文结合图 1 的系统体系结构所述）中使用的过程调用一般如下所列一以名为“picture.gif”的资源被请求时为例：

（1）应用程序调用 getClass.getResourceAsStream(“picture.gif”)。

（2）Class.getResourceAsStream()确定定义该类所在的包名，将该包名置于所请求资源名称的前面，并（假设包名是“package_name”）调用 getClassLoader.getResourceAsStream(“package_name/picture.gif”)。

（3）ClassLoader.getResourceAsStream(“package_name/picture.gif”)寻找文件系统中以文件形式存在的、或是 JAR 文件中以文件形式存

在的、或是在网络服务器上存在的该“picture.gif”资源，并返回一个应用程序能用来读取该资源文件中数据的 InputStream 对象。

如上所述，本发明定义一个新颖的类加载程序—本文称之为对象“RomLoadingClassLoader”。RomLoadingClassLoader 是从 ROM 加载的所有类的类加载程序。这个类加载程序的这个实现，用一个资源名表来确定所请求的资源位于存储器中的什么位置。按照最佳实施例，资源将与类一起被封装（packaged），作为映象（image）装入 ROM 中。进行封装所用的技术不是本发明的组成部分。这样来同时定位（co-locating）类和资源，使特定类和其所需资源的安装和拆卸变得更加容易。一旦资源被封装在一起后，就对照该封装映象运行一个实用程序（它不是本发明组成部分）。这个实用程序确定在该映象中含有的资源的名称或标识符（它们将是一个串值）、所存储资源在映象内的起点、资源的存储数据的长度。根据这个信息，实用程序构造一个用于访问该被存储资源的表，其中，表（或者其它存储机构，诸如阵列）中的每项包含一个设置成该串标识符的键值，以及一个最好包含一（1）指向资源位置的指针和（2）资源长度一的值。可以用来以所述方式封装资源、创建标识资源的表的商业产品，是与 IBM VisualAge® for Embeded Systems 一起提供的 jxeLink 工具—这是出自 IBM 公司的 Java™ Technology Edition 产品。对本领域的熟练人员来说，显然，这种访问表技术在资源尚未被封装在一起时也是可以使用的。在这种情况下，要求知道被存储的资源的标识符、位置和长度值。然后就能用该信息来构造表。

当 getResourceAsStream() 的调用请求某资源时，romLoadingClassLoader 在该表中检查 ROM 中当前可用的资源。如果它发现对应该被请求资源的项，就创建一个 OSMemoryInputStream 类型的对象。这个对象是 InputStream 的子类，所以可以安全地被返回给应用程序代码。这个类的实现能在 ROM 存储器之上使用 InputStream 的性能。所以，期望从输入流读取资源的现有程序，无论是在资源位于文件系统中的桌面计算机上执行的，还是在资源位于 ROM 中的嵌入设备上执行的，其运行都是透明的。

本发明的最佳实施例用以下过程调用（假设被请求资源是与前面例子中所用的相同的“picture.gif”）从应用程序代码加载资源：

（1）应用程序调用getClass.getResourceAsStream("picture.gif")。这个调用原封不动地出自现有技术，所以现有应用程序将继续工作，而不要求修改它们现有的调用语义。

（2）Class.getResourceAsStream()确定定义该类所在的包名，将该包名置于所请求资源名称的前面，然后调用RomLoadingClassLoader.getResourceAsStream("package_name/picture.gif")。此时，被调用的是本发明的新颖的类加载程序，而不是现有技术的类加载程序。

（3）RomLoadingClassLoader.getResourceAsStream()利用资源表寻找该对象在ROM中的位置。它构造一个OSMemory对象，该对象最好明确（1）指向该对象位置的指针，（2）在该位置存储的资源数据的长度。过程然后为这个OSMemory对象创建一个OSMemoryInputStream对象。由于这个对象符合应用程序期望的InputStream协议，所以应用程序就像是在文件系统中存储的文件中读取该数据。

现在将结合图4至9更详细地讨论本发明的最佳实施例。图4至6表示是本发明最佳实施例使用的逻辑的流程图。图7至9表示的是可用来实现本发明最佳实施例的Java程序设计语言的类和过程的样本。

按照本发明，需要建立计算机系统来从ROM加载类和资源，但这种工作无需改动应用程序代码。所需要的是对JVM确定新的类加载程序romLoadingClassLoader，供其启动时使用。所以，并不是需要改动现有应用程序代码才能运行本发明。规定作为JVM一个选项的类加载程序的技术是现有技术水平已知的，是特定于系统的；所以，本文将不详细说明这些技术。许多嵌入设备都用Java代码来支持它们的用户界面。这种设备因而将在设备通电时启动JVM。否则，JVM就将在在JVM上运行的应用程序被执行时启动。JVM的启动方式并不是本发明达组成部分。

图4表示在romLoadingClassLoader执行时发生的初始化过程。对应这个过程的程序设计语言语句，在图7中表示，表现形式是包700中类705的过程710、715。向过程RomLoadingClassLoader710发送

一个消息，将指针 711 传送给资源表 400。过程 710 的调用对应于图 4 的方框 410。这个过程 710 调用过程 initializeResTable 715—如 712 处所示，也传送资源表指针。过程 715 的调用对应于方框 420。如前文所述，资源表 400 含有一些数量的项 405、406、407、408 等等，它们每个有一个键值 401、一个指针 402 和一个长度值 403。过程 715 根据资源表 400 创建散列表 430。按照最佳实施例，项 435、436、437、438 等每个包含(1)来自对应资源表项的相同键值 431 和(2)OSMemory 对象 432。每个 OSMemory 对象 432 包含来自对应资源表项的的指针和长度值(如图 8 中 OSMemory 类 805 的元素 806、807 处所定义的那样)。过程 715 所创建的 hashTable 对象在图 7 中被称为“resTable”—如元素 706 处所指示的那样。

创建散列表的技术在本领域中是已知的，可以使用标准的 Java 类“Hashtable”。所以在图 7 中没有显示过程 715 的详细内容。此外，用散列表是一种可选的优化，在最佳实施例中用于提高检索资源信息的速度。创建和使用散列表要求有额外的存储空间来存储该表，所以，特定的实现可以选择优化空间而不是速度，因而不使用散列表实现。

既然 romLoadingClassLoader 被作为类加载程序安装并且建立了散列表 430，应用程序代码的执行就开始。图 5 表示应用程序请求资源的逻辑流程，以及最佳实施例所用的在 ROM 中查找资源并向应用程序返回输入流的技术。假设应用程序含有的代码要创建一个关于按“.gif”文件存储的并具有资源名“picture.gif”的图形的输入流—如前一个例子中一样，并且程序员选择了名称“pictureStream”作为将被用于这个资源的流。过程调用于是就可以如下:

```
pictureStream=getClass().getResourceAsStream("picture.gif")
```

这个程序设计语句对应于图 5 的方框 505。这个语句导致关于该对象类的 getClass().getResourceAsStream()过程被调用，传递资源名“picture.gif”—如方框 510 所示。该类将按照现有技术自动确定其包名，然后将自动将该名置于资源名参数前面（方框 515）。如果包名例如是“package_name/...”，方框 515 就创建串

“package_name/.../picture.gif”。该类然后将调用一个关于其类加载程序的过程，传递这个串作为参数（方框 520）。按照现有技术水平，这个调用是自动发生的，调用格式为:

getClassLoader().getResourceAsStream(“package_name/.../picture.gif”)

因为 romLoadingClassLoader 已经被作为类加载程序安装，并被定义为 ClassLoader 的子类（见图 7 的元素 707），所以它将截获这个请求。RomLoadingClassLoader 类 705 中的过程 getResourceAsStream 702 然后将执行。方框 525 至 545 对应于过程 720。就是否在散列表 430 存在这个资源的对应项进行判定（方框 525）。根据本例，用于访问散列表的键值是“package_name/.../picture.gif”。如果在表中找不到该键值，方框 530 将向应用程序返回空值，图 5 的对当前资源请求的处理就结束。特定于应用程序的处理然后就会处理这个情况，方法例如是向用户生成一个例外条件或出错消息—这种处理超出了本发明的范围。另一方面，如果找到了该键值，方框 535 就从散列表中检出 OSMemory 对象 432。（如果如上所述的那样尚未构建散列表，方框 525 就检查资源表 400。如果在表中找到该键值，方框 535 就根据资源表 400 中存储的指针 402 和长度值创建 OSMemory 对象。）

然后为这个 OSMemory 对象构建一个 OSMemoryInputStream 对象。这在 721 处的过程调用被执行时发生，导致 OSMemory 类 805 的过程 getInputStream825 被调用。过程 825 随后调用 OSMemoryInputStream 类 905 的过程 OSMemoryInputStream 910。过程 910 向 721 处的调用过程返回一个类型 OSMemoryInputStream 的输入流对象。这个输入流然后被返回（方框 545）到应用程序，后者获得 OSMemoryInputStream 对象（方框 550），然后开始用这个对象来访问资源。在较早时使用的程序设计语言语句的例子中，应用程序要将这个流对象赋予对象“pictureStream”。

既然应用程序已经有了用于访问该资源的、作为 OSMemoryInputStream 对象一个实例的输入流，它就可以开始使用该实例—如图 6 中所示。在方框 605，应用程序对 OSMemoryInputStream 调用 read()过程。图 9 中 925 处表示了读过程的一个例子。方框 610

至 635 代表与这个过程 925 中的代码对应的逻辑。如方框 610 所示，read()过程进行查验以保证偏移（offset）实例变量的值小于资源的总体大小。这是通过调用 getSize()过程而确定的，该过程返回（821）“大小”（size）实例变量（811）的值。如果这个比较（方框 615）结果是否定的，则最好将-1 返回（方框 620）给应用程序，表示已经到达该流的结尾。图 6 的处理于是就结束。否则，如果方框 615 的结果是肯定的，处理就在方框 625 继续，此处，getByte()过程 815 被调用。这个过程 815 返回在通过偏移值与指向资源起始位置的指针的值相加而计算出的位置存储的字节。（实现这个本机过程的细节不是本发明的组成部分，而对本领域的一般熟练人员来说则是显而易见的）。调用过程 925 随后向应用程序返回（方框 635）这个字节，图 6 的对这个调用的处理结束。（注意，图 6 和 9 表示的是应用程序一次获得一个字节，可以用按照现有技术水平定义的过程来用一次调用获得多个字节，其中现有技术的过程的潜在代码保证反复地调用单一字节的返回）。

因此可见，本发明提供了以对应用程序透明的方式从 ROM 存储器加载资源的有益技术。这就免得需要在在 ROM 中存储资源时修改现有应用程序代码，使得应用程序能无视资源是在不是诸如文件系统和网络服务器的“常规”存储器的存储器中而继续执行。

对本领域的一般熟练人员来说，显然，本文中所使用的类和对象的名称只是示意性的，可以等价地使用其它名称。此外，可以向本文中说明的类添加过程，向已经说明过的那些过程添加额外的功能，可以使用能替代图 7 至 9 中所示内容的类结构，这些都不偏离本文中所揭示的发明思想。

尽管说明了本发明的最佳实施例，对于本领域的熟练人员来说，一旦了解了基本的发明思想，就可以在实施例中作出另外的改动和修改。因此，希望后附的权利要求将被解释为将最佳实施例和所有这种改动和修改这两方面都包含在本发明的精神和范围内。

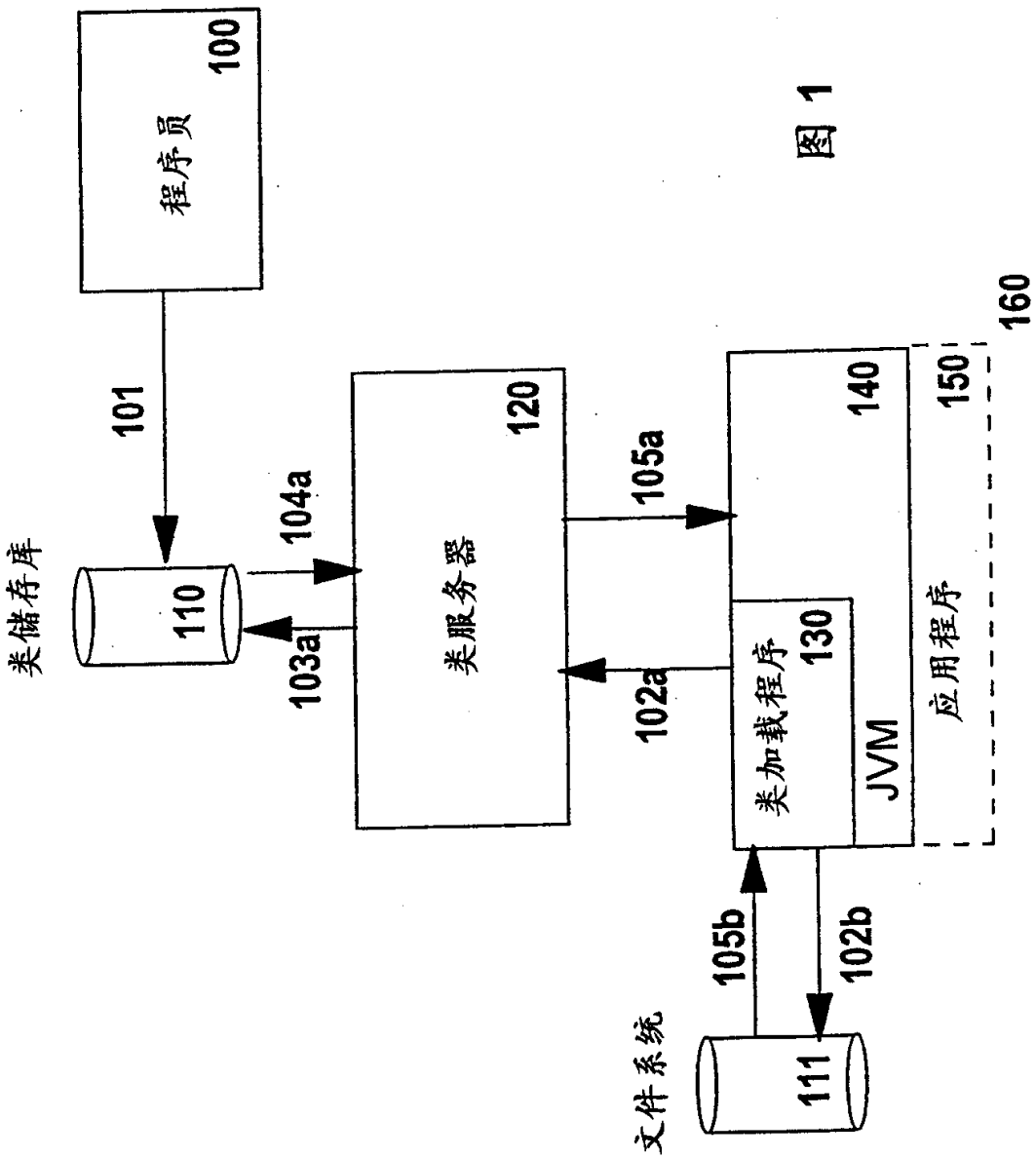

程序员
100
101
类储存库
110
104a
103a
类服务器
120
105a
102a
140
150
160
类加载程序
130
JVM
应用程序
105b
102b
文件系统
111

图 1

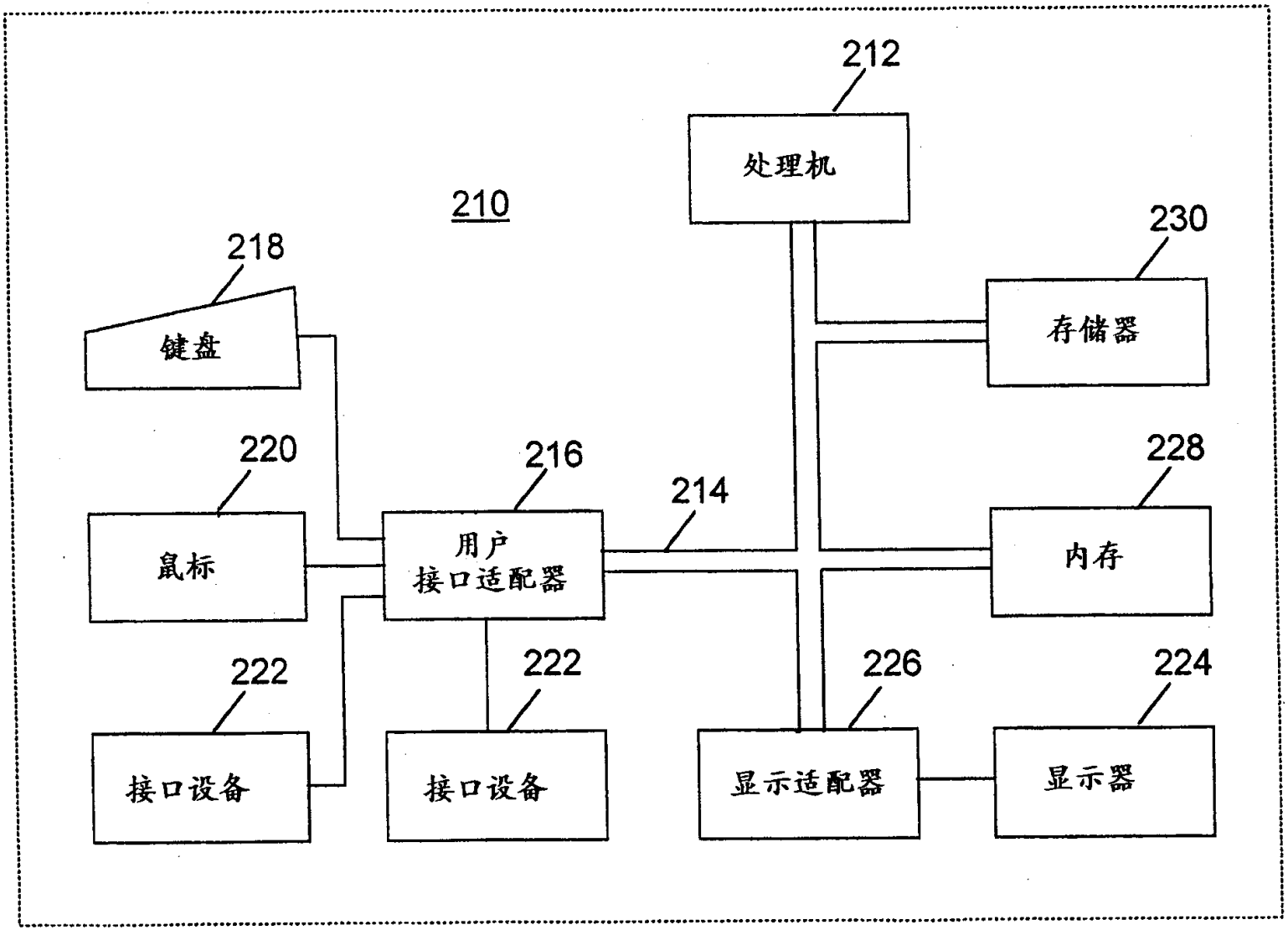

210
212
处理机
230
存储器
218
键盘
220
鼠标
216
用户
接口适配器
214
228
内存
232
222
接口设备
222
接口设备
226
显示适配器
224
显示器

图 2

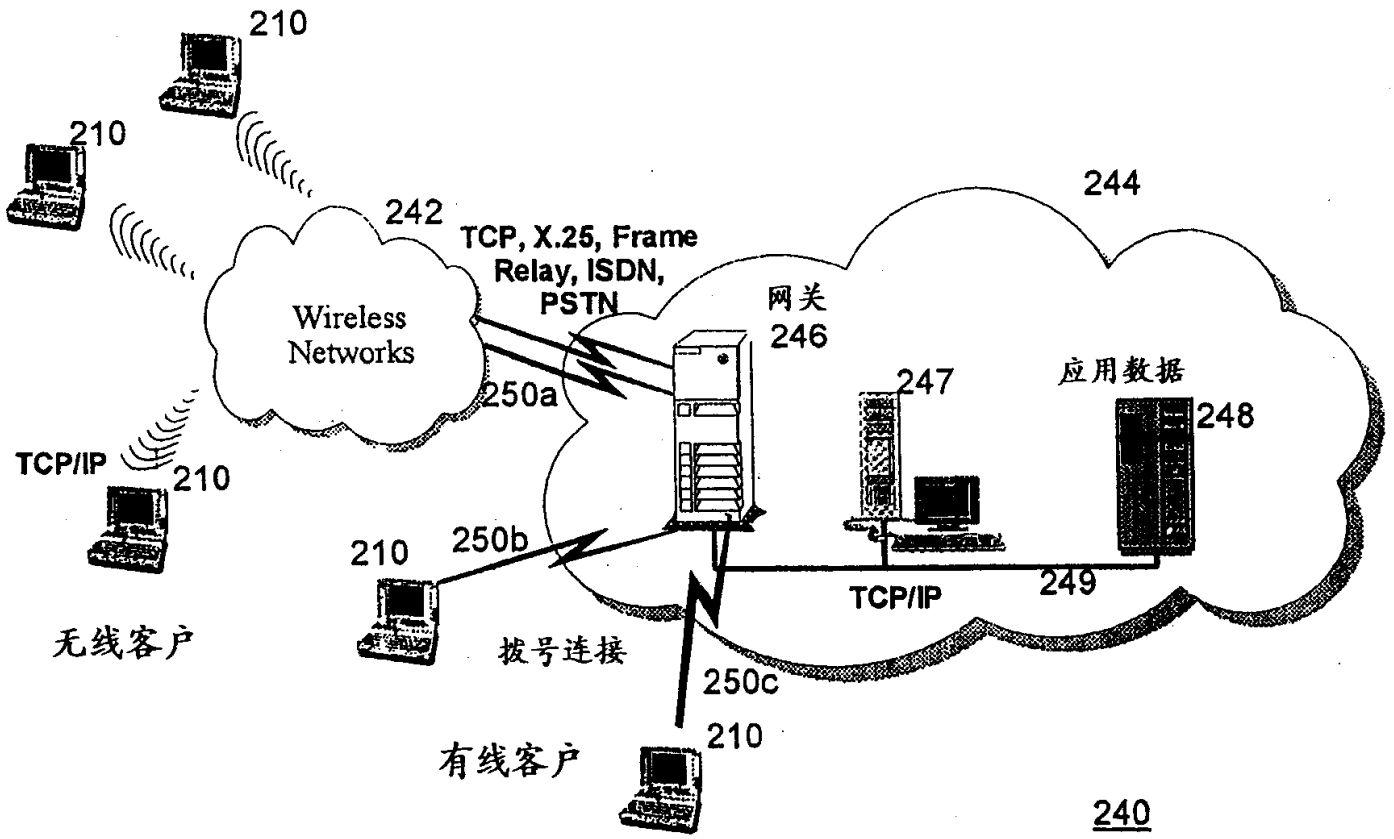

210
210
242
TCP, X.25, Frame Relay, ISDN, PSTN
Wireless Networks
244
网关
246
247
应用数据
248
250a
TCP/IP
210
210
250b
249
TCP/IP
无线客户
拨号连接
250c
210
有线客户
240

图 3

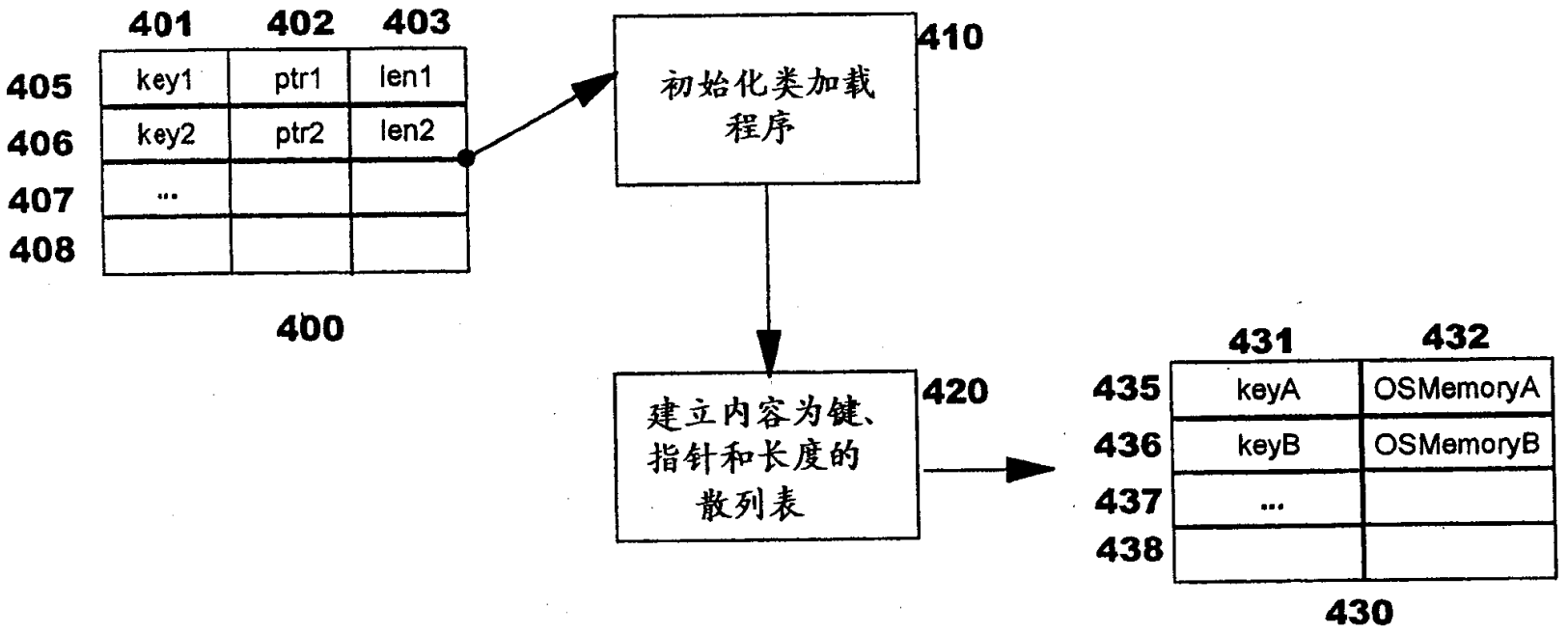

401
402
403
405
406
407
408
key1
ptr1
len1
key2
ptr2
len2
...
400
初始化类加载程序
410
建立内容为键、指针和长度的散列表
420
431
432
435
436
437
438
keyA
OSMemoryA
keyB
OSMemoryB
...
430

图 4

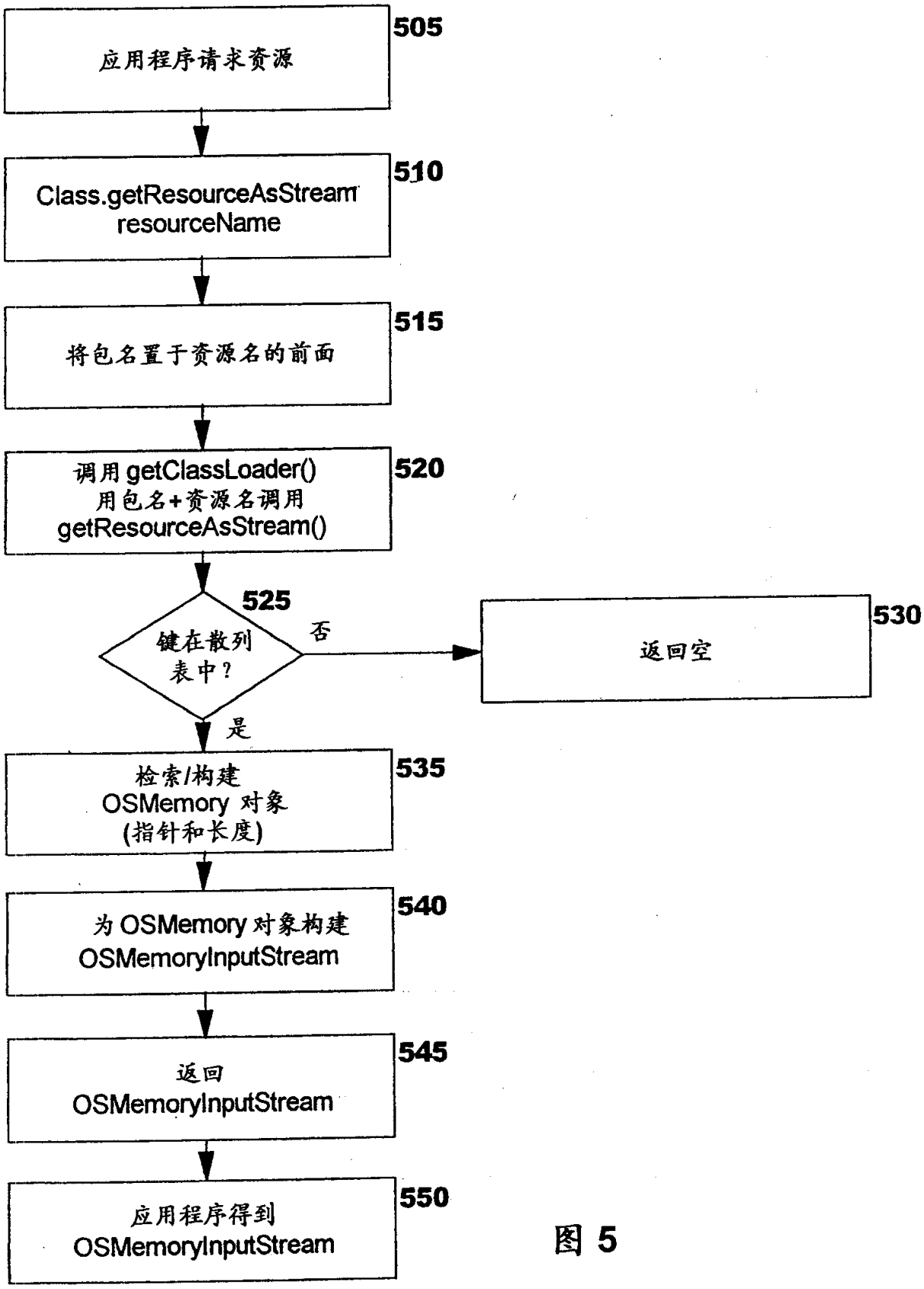

应用程序请求资源
505
Class.getResourceAsStream
resourceName
510
将包名置于资源名的前面
515
调用 getClassLoader()
用包名+资源名调用
getResourceAsStream()
520
525
键在散列
表中？
否
返回空
530
是
检索/构建
OSMemory 对象
(指针和长度)
535
为 OSMemory 对象构建
OSMemoryInputStream
540
返回
OSMemoryInputStream
545
应用程序得到
OSMemoryInputStream
550

图 5

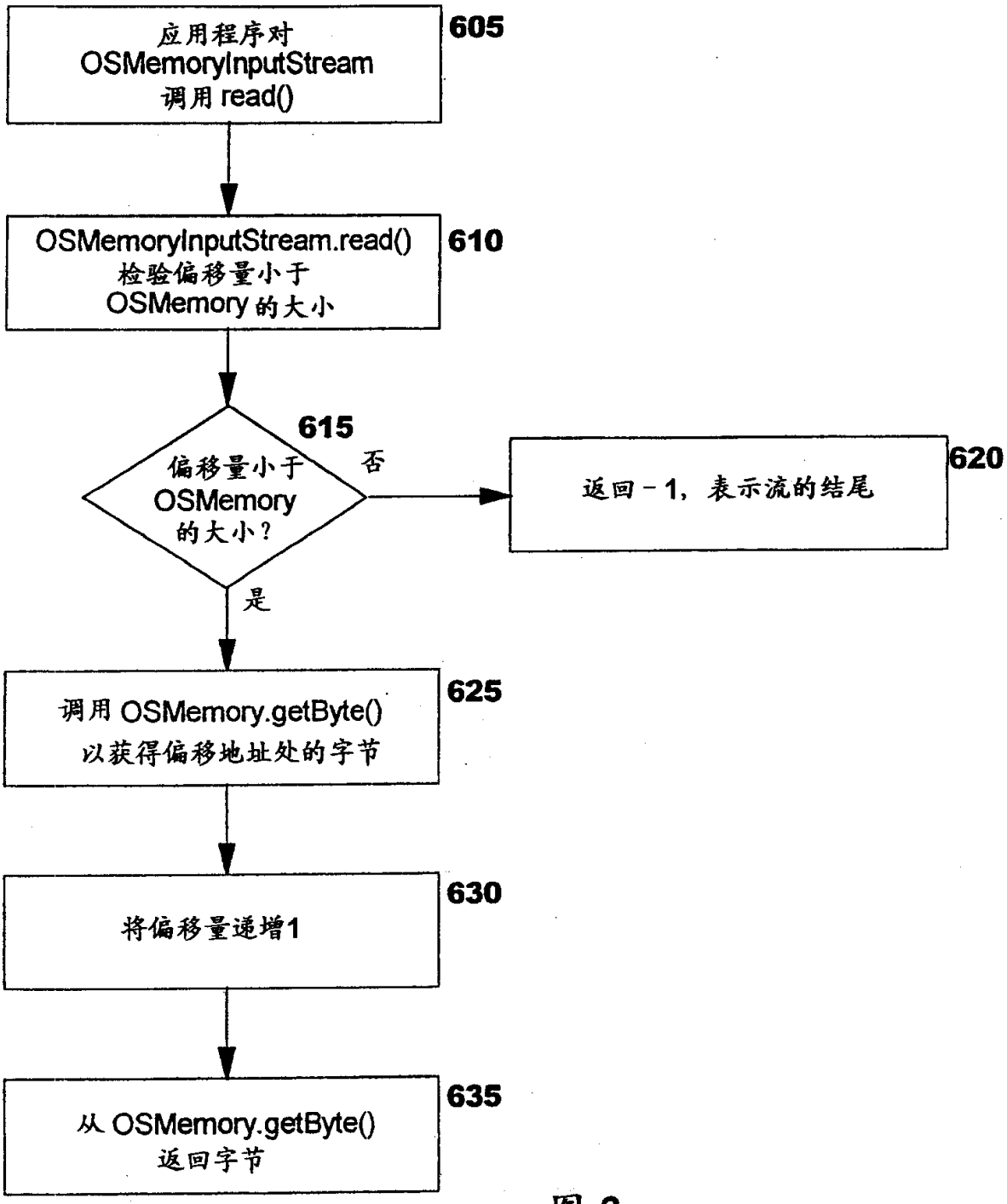

应用程序对
OSMemoryInputStream
调用 read()
605
OSMemoryInputStream.read()
检验偏移量小于
OSMemory 的大小
610
615
偏移量小于
OSMemory
的大小？
否
返回－1，表示流的结尾
620
是
调用 OSMemory.getByte()
以获得偏移地址处的字节
625
将偏移量递增1
630
从 OSMemory.getByte()
返回字节
635

图 6

```
700  package com.ibm.ive.romLoadingClassLoader;

     import java.io.*;
     import java.util.*;
     import com.ibm.ive.osMemory.*;
                                                    707
705  public class RomLoadingClassLoader extends ClassLoader implements
     OSMemoryResourceLoader {
       private Hashtable resTable();
                                  706
710  public RomLoadingClassLoader(ClassLoader parent, OSMemory pointer) {
       super(parent);                                                  711
       initializeResTable(pointer);
     }      712

715  private void initializeResTable(OSMemory pointer) {
       ... Given the pointer to the resource table
       ... build a Hashtable with keys and pointer to
       ... the resources as well as resource length.
     }

720  public InputStream getResourceAsStream(String name) {
       OSMemory osMemory;

       osMemory =  (OSMemory) resTable.get(name);
       if (osMemory == null) return null;

       return osMemory.getInputStream();
     }          721

     }
```

图 7

```
800     package com.ibm.ive.osMemory;

        import java.io.*;

805     public class OSMemory {
          private int   pointer;  806
          private int   size;     807

810     public OSMemory(int pointer, int size) {
          super();

          this.pointer   = (int) pointer;
          this.size      = size;     811
          this.allocated = false;
        }

815     native public byte getByte(int offset);

820     public int getSize() {
          return size;
        }   821

825     public InputStream getInputStream() {
          return new OSMemoryInputStream(this);
        }

        }
```

图 8

```
900  package com.ibm.ive.osMemory;

     import java.io.*;

905  class OSMemoryInputStream extends InputStream {
       private OSMemory osMemory;
       private int      offset;

910  public OSMemoryInputStream(OSMemory osMemory) {
       super();

       this.osMemory = osMemory;
       this.offset   = 0;
     }

920  public int available() throws IOException {
       return osMemory.getSize() - offset;
     }

925  public int read() throws IOException {
       if (offset >= osMemory.getSize()) return -1;

       return osMemory.getByte(offset++) & 0xFF;
     }

     }
```

图 9