



(12) 发明专利

(10) 授权公告号 CN 101002174 B

(45) 授权公告日 2011. 12. 07

(21) 申请号 200580027006. 7

(51) Int. Cl.

(22) 申请日 2005. 06. 07

G06F 9/445 (2006. 01)

(30) 优先权数据

(56) 对比文件

04/06493 2004. 06. 15 FR

CN 1321281 A, 2001. 11. 07, 说明书第 1 页第 6-21 行.

(85) PCT 申请进入国家阶段日

审查员 唐楹琰

2007. 02. 09

(86) PCT 申请的申请数据

PCT/EP2005/052621 2005. 06. 07

(87) PCT 申请的公布数据

W02005/124544 FR 2005. 12. 29

(73) 专利权人 格马尔托股份有限公司

地址 法国默东

(72) 发明人 G·格里莫德 J·-J·范德瓦勒

(74) 专利代理机构 中国专利代理(香港)有限公

司 72001

代理人 曾祥尧 王忠忠

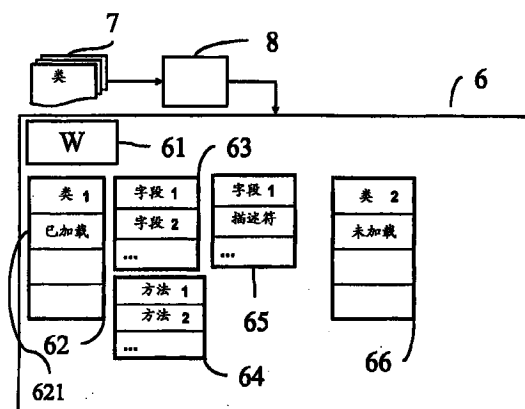
权利要求书 2 页 说明书 6 页 附图 2 页

(54) 发明名称

在便携式装置中加载具有面向对象的中间语言的软件的方法

(57) 摘要

本发明涉及用于在便携式数字装置 (5) 的非易失性存储器中加载包括可加载的多个模块 (7) 的软件的方法, 其中加载至少一个模块包括测试非易失性存储器 (6) 中是否存在与所述模块关联的数据结构, 以及当需要时, 创建指示它为空的数据结构 (62), 解析内部元素的总体的链接, 此后, 将结构标记为已加载 (621), 测试是否存在可加载模块中引用的每个其他模块的关联的数据结构, 以及当需要时, 创建所述结构并指示它为空。所述发明使得减少加载模块期间板上软件所占用的空间。



1. 一种用于加载软件的方法,所述软件被编译成面向对象的中间语言并包括用于加载到便携式数字装置(5)的非易失性存储器中的多个模块(7),所述便携式数字装置配备有虚拟执行机,以能够由所述虚拟执行机的解释器执行的形式来加载这些模块(7),其特征在于加载所述模块的至少其中之一包括如下步骤:

- 检查所述非易失性存储器(6)中是否存在与待加载的模块关联的数据结构;以及
- 如果此数据结构不存在,则在所述非易失性存储器中创建与此模块关联的数据结构(62),并将此数据结构标记为未加载;
- 在此数据结构中解析此模块的所有内部元素的链接,然后将此数据结构标记为已加载(621);
- 对于正在加载的模块中引用的每个其他模块,检查所述非易失性存储器(6)中是否存在与此其他模块关联的数据结构;
- 如果与此其他模块关联的数据结构不存在,则在所述非易失性存储器中创建与此其他模块关联的数据结构(66),并将此数据结构标记为未加载。

2. 如权利要求1所述的方法,其特征在于,创建与要加载的模块关联的数据结构的步骤包括在此数据结构中存储此要加载的模块内所含的要加载的模块的描述符。

3. 如权利要求1或2所述的方法,其特征在于:
- 所述链接解析步骤包括:
 - 验证是否存在与每个内部元素关联的数据结构;
 - 如果此数据结构不存在,则在非易失性存储器中创建与此内部元素关联的数据结构(64、65);
 - 对于正在加载的模块中提及的每个外部元素:
 - 验证是否存在与此外部元素关联的数据结构;
 - 如果此数据结构不存在,则在所述非易失性存储器中创建与此外部元素关联的数据结构。

4. 如权利要求1或2所述的方法,其特征在于,加载所述模块的步骤还包括,在链接解析之后,如果此模块提及的所有外部模块均被标记为已加载,则将此模块标记为已链接。

5. 如权利要求4所述的方法,其特征在于:
- 检查是否存在数据结构包括,如果此数据结构存在,则在此数据结构与正在加载的模块中此数据结构的引用之间进行相符性测试,并存储相符性检查信息;
 - 当所述模块被标记为已链接时,删除所引用的相符性检查信息。

6. 如权利要求4所述的方法,其特征在于,所述方法还包括如下步骤:

- 初始化标记为已链接的模块的静态数据;
- 将此模块标记为可供使用。

7. 如权利要求6所述的方法,其特征在于,一旦所述模块被标记为可供使用,则删除所述便携式数字装置中存储的且用于初始化静态数据的代码。

8. 如权利要求1或2所述的方法,其特征在于,要加载的模块是编译的Java.class文件。

9. 如权利要求8所述的方法,其特征在于,所述数据结构是可以由所述虚拟执行机处理的数据结构。

10. 如权利要求 8 所述的方法,其特征在于,所述数字装置 (5) 是具有嵌入式 Java 或 DotNet 虚拟机的芯片卡。

11. 如权利要求 1 或 2 所述的方法,其特征在于,对于所有程序模块执行所述模块加载。

12. 如权利要求 1 或 2 所述的方法,其特征在于,在模块不可用之后,检查此模块的可用性,并由所述便携式数字装置自动加载此模块。

在便携式装置中加载具有面向对象的中间语言的软件的方法

技术领域

[0001] 本发明涉及嵌入在便携式装置中的软件和执行环境,更具体来说涉及在便携式装置中加载此类软件的方法。

背景技术

[0002] 已经开发了许多面向对象的中间编程语言。这些语言的主要目的是使它们的程序独立于执行它们所必需的硬件。由此,通常使程序员免于与特定硬件相联系的约束。为不同硬件发布相同版本的程序也是可能的。因此已经见到面向对象的中间语言、例如通过编译 Java 源语言获得的 Java 伪代码(命名为字节码)随之而来的普及度方面的提高。

[0003] 桌上型计算机的 Java 程序常常以 .class 文件构成的一组模块的形式来发布。这些文件对应于软件的已编译形式。每个已编译文件对应于类数据结构,因此包括在此类中的信息:即,类元素(其内容、字段、方法)的描述、类使用并在其他类中定义的元素(字段和方法)的描述、可以由 Java 虚拟机解释器解释的指令(伪代码)形式的类方法的代码。在 .class 文件中,在引用表(常数池)中将类元素的描述和类使用的元素分组在一起,它们主要将数值符号(常数池中的项编号)与元素描述关联。因此,在可解释的指令(伪代码)中,对类元素和该类使用的元素的引用通过它们在引用表(常数池)中的项编号来“引用”。.class 文件可以具有例如如下结构:

```
[0004]  ClassFile{
[0005]  Id_info;           // 版本号
[0006]  Constant_pool;     // 存储对类元素和该类使用的元素的所有
[0007]  引用
[0008]      Base_info;      // 访问标志,类名和超类
[0009]      Interface_list;  // 类实现的接口
[0010]      Field_list;      // 类字段的描述
[0011]      Method_list;     // 此类中的方法(包括它们的伪代码,即
[0012]  可以由虚拟机的解释器解释的指令)
[0013]      Attribute_list   // 属性(例如:调试信息等)
```

[0014] 编译的程序文件被存储在计算机的海量存储装置上、例如存储在硬盘上。Java 软件由安装在计算机上的 Java 虚拟机执行。该虚拟机加载 ClassFile 信息并将其转换成工作存储器中的数据结构,其中这些数据结构专用于虚拟机并允许此虚拟机执行该程序。

[0015] Java 软件的这种加载(和转换)阶段可以在大多数具有相当多随机存取存储器(RAM)和处理器资源的桌上型计算机上执行。这些资源允许它们通过即时且在 RAM 中创建虚拟机的内部数据结构来有效地加载程序,然后允许程序在 RAM 中执行。执行前的这种加载阶段主要包括链接编辑阶段,其中将对创建的数据结构中类的内部和外部元素的引用替换为由此创建的数据结构的存储地址。

[0016] 但是,必须运行 Java 软件的某些系统有非常多的受限能力,这阻碍了这种类型的执行。因此,由于小型化的约束,芯片卡通常包括 1 与 4 千字节之间的 RAM(用作工作存储器),32 与 64 千字节之间的可重写非易失性存储器(用作海量存储器和工作存储器)以及约 256 千字节的只读存储器(用于存储执行环境代码)。因此此类芯片是不适于这种类型的功能,包括在非易失性存储器中存储 .class 文件和此阶段为每次执行在易失存储器中的加载。而且,卡上的类数据因此而是重复的:非易失性存储器中的 .class 文件中的数据和易失存储器中数据结构形式的数据。

[0017] 第一个解决方案包括将芯片卡上的虚拟机的规范所定义的数据结构直接存储在芯片卡的非易失性存储器中。在此方式中,该卡上的虚拟机不再需要执行加载操作,因为它接收数据结构格式的类数据,因此已经可以执行。

[0018] 文献 US2003/028686 描述了此类加载技术。因此将 .class 文件转换成加载机上的 Java Card(注册商标)格式的 .cap 文件,然后在制造过程中或稍后其使用期间将此文件加载到芯片卡上。具体来说 .cap 文件包括头、引用表、方法描述符和外部元素描述符。该加载机还生成称为 ExportFile 的文件,它与生成的 .cap 文件关联。此 ExportFile 文件使得能够在 .cap 文件中声明公用元素,例如以使其他分开的应用程序可以使用它的元素。

[0019] 但是,此解决方案的使用有它的缺点。Java Card 具有相对于 Java 的缩减的指令集。具体来说,Java Card 不支持动态类加载。在将 .cap 文件加载到芯片卡之后,限制动态链接编辑。从使用 Java Card 不支持的指令的 .class 文件生成加载的 .cap 文件是不容易的。而且,该转换阶段使程序员的任务复杂化,因为必须对每种芯片卡类型提供特定的转换(与已经加载到卡上且转换期间必须为其提供的对应 ExportFile 的程序有关)。还必须一次性地将程序加载到芯片卡上。传输中的任何中断都将涉及从头重启加载过程。最后,此解决方案的两个其他缺点是使用专用于卡(.cap)的类发布文件格式以及虚拟机的内部数据结构的必要标准化,这些在由此定义的新 .cap 文件中描述。

[0020] 第二种解决方案包括在安装 .class 文件时将 .class 文件编译成芯片卡的本地语言。芯片卡的工作存储器不再被虚拟机占用,并显著缩短执行时间。随后执行加载和链接编辑过程。以此方式生成的本地代码有许多缺点,因为它占用远比解释的代码大的存储量。而且,编译是复杂的且成本高,因为它必须专用于芯片卡。芯片卡的安全性也被弱化,因为编译成本地机器代码的程序可以在卡的工作时提供数据。

发明内容

[0021] 本发明的目的在于解决一个或多个这些缺点。因此本发明的目的在于提供一种用于加载软件的方法,所述软件被编译成面向对象的中间语言和并包括用于加载到配备有虚拟执行机的便携式数字装置的非易失性存储器中的多个模块,需要以可由所述虚拟机的解释器执行的形式来加载这些模块,这些模块的至少其中之一的加载包括如下步骤:

[0022] - 检查非易失性存储器中是否存在与此模块关联的数据结构;以及

[0023] - 如果此数据结构不存在,则在非易失性存储器中创建与此模块关联的数据结构,并将此数据结构标记为未加载;

[0024] - 在此数据结构中解析此模块的所有内部元素的链接,然后将此数据结构标记为已加载;

[0025] - 对于正在加载的模块中引用的每个其他模块,检查是否存在与此其他模块关联的数据结构;

[0026] - 如果与此其他模块关联的数据结构不存在,则在非易失性存储器中创建与此其他模块关联的数据结构,并将此数据结构标记为未加载。

[0027] 根据本发明的备选实施例,创建与要加载的模块关联的数据结构的步骤包括在此数据结构中存储要加载的此模块内所含的要加载的模块的描述符。

[0028] 根据本发明的另一个备选实施例,链接解析步骤包括:

[0029] - 验证是否存在与每个内部元素关联的数据结构;

[0030] - 如果此数据结构不存在,则在非易失性存储器中创建与此内部元素关联的数据结构;

[0031] - 对于正在加载的模块中提及的每个外部元素:

[0032] - 验证是否存在与此外部元素关联的数据结构;

[0033] - 如果此数据结构不存在,则在非易失性存储器中创建与此外部元素关联的数据结构;

[0034] 根据本发明的另一个备选实施例,加载所述模块的步骤还包括在链接解析之后,如果此模块提及的所有外部模块均被标记为已加载,则将此模块标记为已链接。

[0035] 检查是否存在数据结构的步骤可以包括,在如果存在的此数据结构与正在加载的模块中的其引用之间的相符性测试,并存储相符性检查信息;当该模块被标记为已链接时删除这些引用的相符性检查信息。

[0036] 此类方法可以包括如下步骤:

[0037] - 初始化标记为已链接的模块的静态数据;

[0038] - 将此模块标记为可供使用。

[0039] 一旦该模块被标记为可供使用,就可以删除数字装置中存储的且用于初始化静态数据的代码。

[0040] 根据本发明的另一个备选实施例,要加载的模块是编译的 Java. class 文件。这些数据结构是可以由虚拟机处理的数据结构。数字装置是具有嵌入式 Java 或 DotNet 虚拟机的芯片卡。

[0041] 可以支持对所有程序模块执行此类模块加载。

[0042] 在模块不可用之后,可以检查此模块的可用性,以便可以由便携式装置自动加载该模块。

[0043] 本发明还涉及用于一种用于程序的加载器,该程序被编译成面向对象的中间语言并包括要在便携式数字装置中加载的多个模块,该加载器能够存储在装置的非易失性存储器中,能够访问源装置上存储的模块以及能够实现前面描述的方法。本发明还涉及根据此类方法在便携式装置中加载的程序。

[0044] 本发明还涉及包括存储前述加载器的非易失性存储器的便携式数字装置。根据本发明的备选实施例,非易失性存储器存储前述程序。

[0045] 根据本发明的另一个备选实施例,该装置是芯片卡。

附图说明

[0046] 通过阅读以非限定方式作为示例提供且与所附权利要求相关的描述,本发明的其他特征和优点将变得显而易见,其中:

[0047] 图 1 示出用于向芯片卡中加载的实现的计算机系统的示例;

[0048] 图 2 示出加载期间存储在芯片卡的非易失性存储器中的数据;

[0049] 图 3 示出与加载期间执行的步骤相关的数据结构标记符的状态演化。

具体实施方式

[0050] 便携式电子装置下文将用于指代如便携式计算机的便携式装置,但是也指代配备有包括处理器的微处理器和存储装置的任何其他电子装置。

[0051] 本发明提出通过为加载期间第一次提及的元素创建数据结构来将程序模块加载到便携式装置上。因此,加载期间通过模块的外部元素直接创建要由便携式装置的虚拟机执行的数据结构,而无需存储其临时表示。当加载、链接编辑和优化步骤由程序模块加载器执行时,从非便携式装置的非易失性存储器中删除不需要执行的元素。

[0052] 图 1 示出为将编译的程序加载到芯片卡 5 上而实施的部件的示例,芯片卡 5 在此示例中构成便携式装置。计算机 1 配备有芯片卡读卡器 2,芯片卡 5 插入其中。例如,计算机 1 通过计算机网络 4 连接到服务器 3。编译的程序模块驻留在服务器 3 上(例如对于 Java 编译的程序采用一组 .class 文件的形式)。芯片卡 5 配备有用于执行加载的程序的虚拟机。

[0053] 下文是在以 Java 编译的程序的特定上下文中详细描述本发明,当然本发明还可以应用于编译成其他面向对象的中间语言(例如,.NET(注册商标))的程序。

[0054] 图 2 示出在模块 7 由加载器 8 加载时存储在芯片卡 5 的存储器 6 中的数据。加载器 8 是例如存储在芯片卡 5 的只读存储器中的应用程序。加载器 8 访问要加载的名为 Class1 的第一模块。加载器将 Class1 全部或一部分置于工作存储器 61 中。

[0055] 加载器执行初步检查,以查看 .class 文件是否可用(例如可以将要动态加载的类的 .class 文件与 URL 地址关联)。加载器还检查非易失性存储器中是否存在与 Class1 关联的数据结构 61。实际上,根据本发明的加载方法,可以在加载另一个模块时预先创建该数据结构。如果与 Class1 关联的数据结构 62 不存在,则在卡 5 的非易失性存储器中创建此数据结构。结构 62 包括加载状态标记符 621。当创建数据结构时,将标记符 621 置于“未加载”状态。图 3 中示出了不同标记符状态和将它们分开的步骤。

[0056] 当加载器 8 确认数据结构 62 存在且处于“未加载”状态时,加载器解析 Class1 中发现的引用的所有内部元素的链接。加载器然后删除执行程序或 Class1 所不需要的参数(例如线路编号表、源文件、调试信息等)。由此大大减少存储的数据在卡的非易失性存储器中占用的空间。

[0057] 可以设想每个模块仅创建一个数据结构,并将此模块的内部元素的所有数据存储在关联的数据结构中。在非易失性存储器中有利地为要加载的模块中引用的每个元素创建数据结构。因此可以更快且更容易地处理这些数据结构,具体来说例如允许删除链接编辑或优化之后不再需要的引用的元素。

[0058] 在该示例中,内部链接解析意味着创建可以由虚拟机处理的数据结构 63、64 和 65。这些数据结构是在先前加载另一个模块期间(下文详细解释的外部链接解析期间)创

建的或由加载器 8 在此内部链接解析期间已确定它们不存在而创建的。数据结构 63 和 64 分别是列出内部字段和其引用出现在 Class1 中的方法的表。数据结构 65 对应于数据结构 63 中列出的字段的描述符。由此,可以将数据结构与每个内部元素或内部元素的每个列表关联。

[0059] 这些数据结构在有关它们的元素实际被加载时才完成。因此这种加载使得在这些数据结构中仅存储执行所需的元素成为可能,从而自动删除从多个其他元素引用的元素的冗余。

[0060] 一旦执行了该链接解析,则标记符 621 被置于“已加载”状态。此链接解析步骤有利地用于执行模块的调试。因此,此链接编辑步骤将使容易地确定可能原来是故障的内部链接成为可能。

[0061] 可以在加载之后删除某个量的数据,以便在装置的非易失性存储器中占用最小的可能空间。具体来说,注意.class 文件引用表包含许多仅在加载模块过程中而非执行期间使用的项。因此,可以删除编辑链接时使用的元信息。一旦对.class 文件执行了链接编辑,则还可以删除某些方法描述符。例如,一旦执行了内部链接编辑,则可以删除模块的私有内部模块的文本描述:此描述仅用于编辑此链接,并且将不再使用。

[0062] 此外,可以在预期没有其他模块将能够引用它们的情况下,删除某些公有内部元素(要被其他模块读取)的描述。这种情况主要是在模块的所有后续加载被禁止以便冻结便携式装置时。可以通过使加载器 8 失效或加载器 8 的代码存储在非易失性存储器中的情况下删除该代码来冻结装置。

[0063] 在内部链接解析步骤之后,加载器 8 执行对正在加载的 Class1 模块中引用的外部元素的链接的解析。因此,对于 Class1 中引用的每个外部模块,检查关联的数据结构是否存在。如果不存在关联的数据结构,则创建关联的数据结构,并将其标记符置于“未加载”状态。在该示例中,Class1 引用外部模块 Class2。因为在加载 Class1 之前没有 Class2 的数据结构,所以 Class1 的外部链接解析的步骤暗示数据结构 66 的创建。

[0064] 如果必须为模块中的每个元素创建数据结构,则必须解析对正在加载的模块中的外部元素的每个引用。相似地,如果测试确定关联的数据结构不再存在,则为每个外部元素创建数据结构。

[0065] 在解析外部元素的链接期间,存在性测试有利地包括已发现的数据结构与其正在加载的模块中的引用之间的相符性的测试。如果此相符性测试是否定的,则生成加载失败消息。此相符性测试提供附加的调试方式。

[0066] 因此,模块的加载使得在便携式装置的非易失性存储器中尽可能地为其其他要加载的模块创建数据结构成为可能。此外,标记符使得在此加载被中断的点处恢复加载成为可能。此类加载使得在关联的模块可用之前仅加载某些模块成为可能。因此本发明使得在加载模块时动态地修改它们的数据结构成为可能。

[0067] 当解析了要加载的模块的所有外部引用的链接时,其数据结构的标记符被置于“已链接”状态。

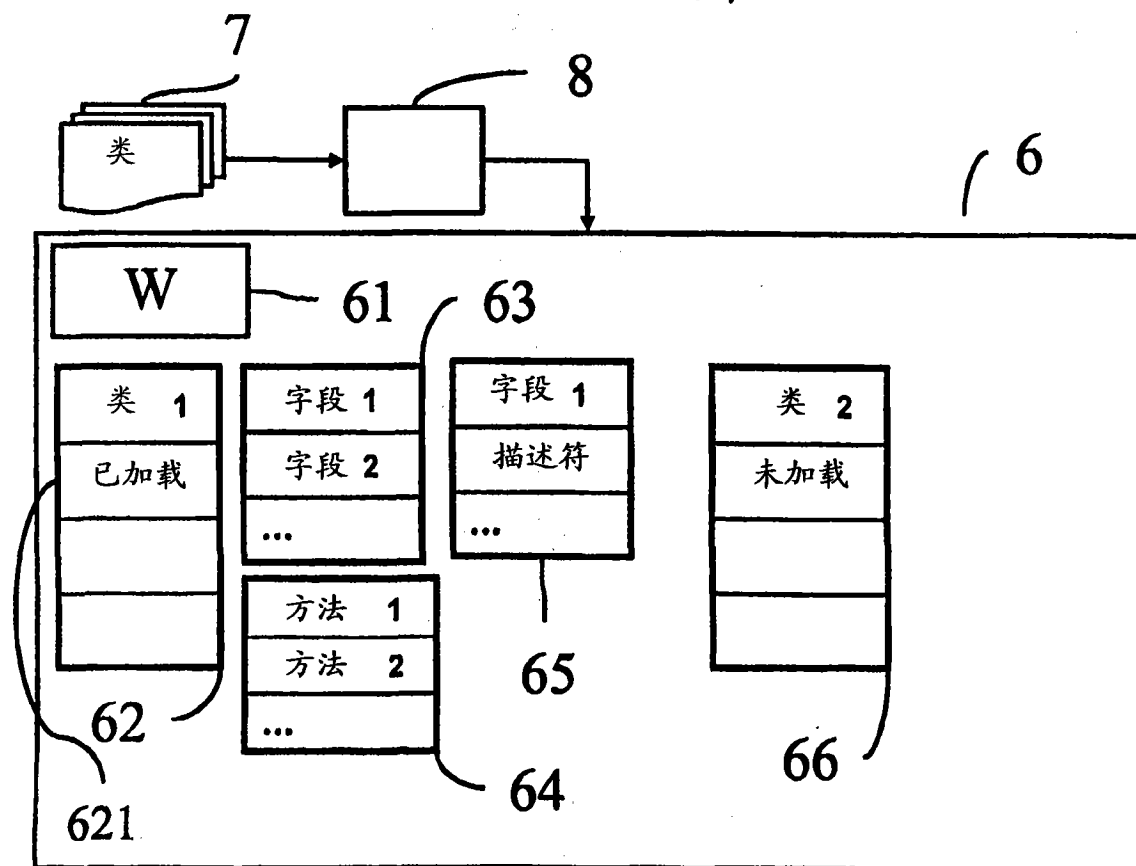
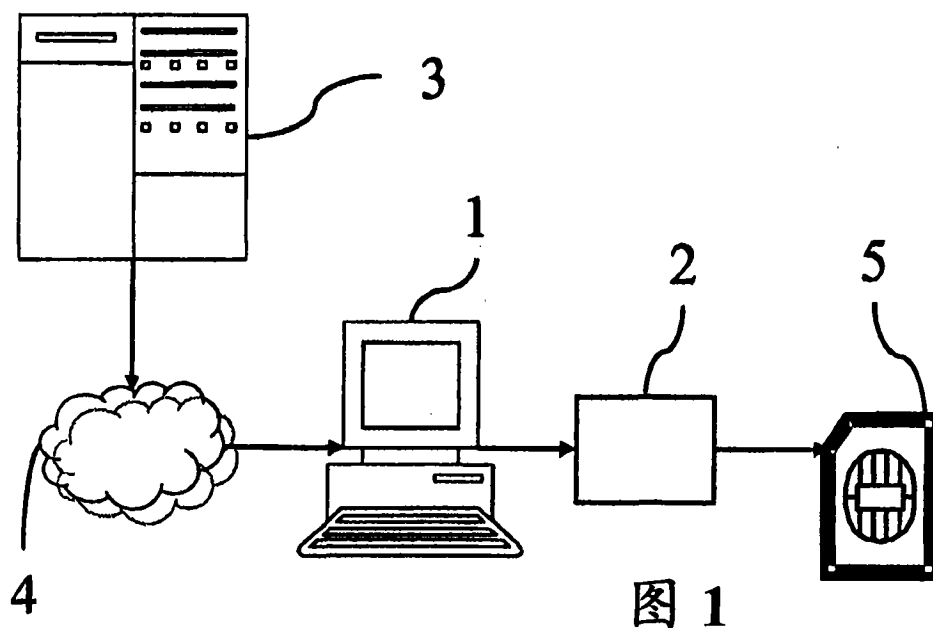
[0068] 然后,初始化此已链接的模块的静态数据。然后将其数据结构的标记符置于“就绪”状态。该模块则位于便携式装置的非易失性存储器中并已可以使用。当数据结构处于“就绪”状态时删除存储在便携式装置中且要初始化静态数据的代码,以便减少占用的存储

空间。当数据结构的标记符被置于“就绪”状态时,可以删除位于工作存储器中的模块的元素以便释放数字装置中的空间。

[0069] 本领域技术人员将知道如何定义编译程序的连续模块的合适加载次序。具体来说,加载器 8 可以存储其数据结构已经创建的未加载的模块,然后连续加载它们。加载器 8 还能够仅加载要提交给它的模块。

[0070] 如果要加载的模块不可用,则加载器还可以重复检查可用性,并尽可能地自动加载此模块。

[0071] 由于非易失性存储器空间被释放,所以可以将能够加载 .class 文件的标准 Java 虚拟机存储在芯片卡中。开发人员就可以对便携式装置和桌上型计算机发布相同的程序。开发人员还将拥有对他 / 她选择的库元素的访问权。不同于在 Java Card 编程中,开发人员无需先恢复 ExportFile 文件。



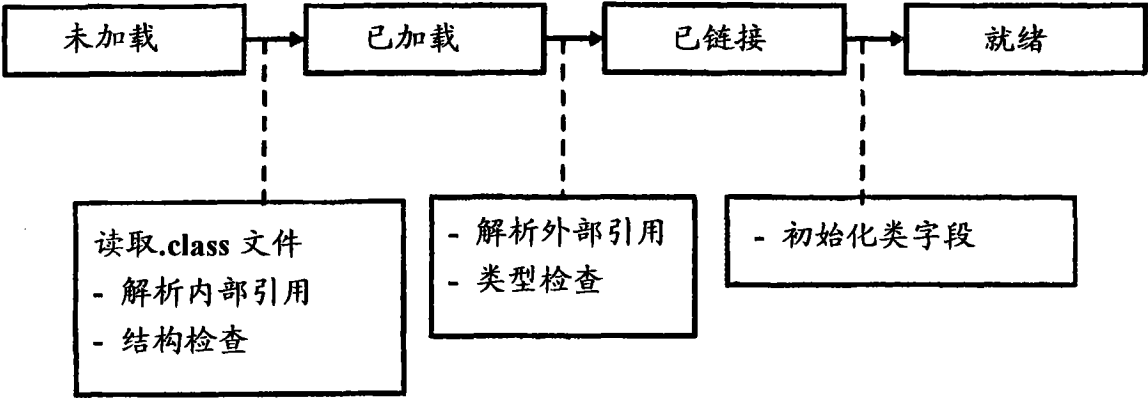


图 3