(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 102081546 B
(45)授权公告日 2013.12.18

(21)申请号 201010543228.7
(22)申请日 2010.11.15
(30)优先权数据
12/627,206 2009.11.30 US
(73)专利权人 国际商业机器公司
地址 美国纽约
(72)发明人 H·W·亚当斯三世 S·C·弗利
C·E·赫利斯丘克 A·R·劳
P·D·希普顿
(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 11038
代理人 李玲
(51)Int.Cl.
G06F 9/455(2006.01)
(56)对比文件
EP 1387265 A1,2004.02.04, 全文.
CN 1531680 A,2004.09.22, 全文.
EP 1491999 A2,2004.12.29, 全文.
CN 101042671 A,2007.09.26, 全文.
CN 1499363 A,2004.05.26, 全文.
CN 101162439 A,2008.04.16, 全文.
审查员 张晓芳

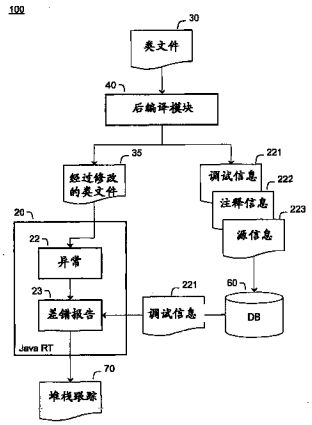
权利要求书2页 说明书8页 附图7页

(54)发明名称

通过分隔额外信息来内存优化虚拟机代码的方法和系统

(57)摘要

本发明涉及一种通过从可执行的虚拟机代码或解释代码中划分出额外信息来优化内存的方法、计算机程序产品和系统。该额外信息可以单独保存,或者可以在需要时从初始代码中访问,以便调试或服务于字段中的代码。在保持用于调试以及为字段中的代码提供服务所必需的处理的额外信息的可访问性的同时,通过减少内存足迹,该方法优化了内存使用率。



CN 102081546 B

1. 一种用于优化虚拟机或解释代码的方法，包括：

接收包含多个程序指令以及第一额外信息的代码，其中所述代码用供虚拟机运行的语言编写，并且其中所述第一额外信息是第一类型的不可执行信息；

通过移除所述第一额外信息以及将其替换成引用所述第一额外信息的位置的键值来修改所述代码，其中所述位置是以下位置之一：保存移除的第一额外信息的存储位置、保存第一额外信息的代码的存储位置、能够取回所述第一额外信息的代码的位置，所述键值用于标识所述位置；以及

执行经过修改的代码，并且如果在运行过程中发生请求所述第一额外信息的事件，则响应于所述事件使用所述键值来定位处于所述位置的第一额外信息，并且从所述位置加载所述第一额外信息。

2. 根据权利要求 1 所述的方法，其中所述事件是故障，并且所述第一类型的不可执行信息是调试信息。

3. 根据权利要求 2 所述的方法，还包括：

修改所述多个程序指令中的一个或多个，以便拦截异常处理进程，并且使用键值将所述异常处理进程重定向到所述第一额外信息的位置。

4. 根据权利要求 1 所述的方法，其中所述代码包括第二额外信息，并且其中所述第二额外信息是第二类型的不可执行信息，以及第一和第二类型的不可执行信息是不同的。

5. 根据权利要求 4 所述的方法，其中第二类型的不可执行信息是源信息或注释信息。

6. 根据权利要求 1 所述的方法，其中所述代码是 Java 类文件或 Java 归档 (JAR) 文件，以及所述多个程序指令是 Java 字节码指令。

7. 根据权利要求 1 所述的方法，其中所述代码是 .NET 代码文件，并且所述多个程序指令是公用中间语言 (CIL) 字节码指令。

8. 根据权利要求 1 所述的方法，其中所述语言是解释语言。

9. 根据权利要求 1 所述的方法，其中所述位置是保存移除的第一额外信息的存储位置，并且所述方法还包括：

将移除的第一额外信息保存在所述保存移除的第一额外信息的存储位置中。

10. 根据权利要求 1 所述的方法，其中所述位置是保存第一额外信息的代码的存储位置，并且其中所述加载包括从所述保存第一额外信息的代码的存储位置加载代码。

11. 根据权利要求 1 所述的方法，其中所述位置是能够取回所述第一额外信息的代码的位置，并且其中所述加载包括从所述能够取回所述第一额外信息的代码的位置取回代码。

12. 一种用于优化虚拟机或解释代码的系统，包括：

用于接收包含多个程序指令以及第一额外信息的代码，其中所述代码用供虚拟机运行的语言编写，并且其中所述第一额外信息是第一类型的不可执行信息的装置；

用于通过移除所述第一额外信息以及将其替换成引用所述第一额外信息的位置的键值来修改所述代码，其中所述位置是以下位置之一：保存移除的第一额外信息的存储位置、保存第一额外信息的代码的存储位置、能够取回所述第一额外信息的代码的位置，所述键值用于标识所述位置的装置；以及

用于执行经过修改的代码，并且如果在运行过程中发生请求所述第一额外信息的事

件，则响应于所述事件使用所述键值来定位处于所述位置的第一额外信息，并且从所述位置加载所述第一额外信息的装置。

13. 根据权利要求 12 所述的系统，其中所述事件是故障，并且所述第一类型的不可执行信息是调试信息。

14. 根据权利要求 13 所述的系统，还包括：

用于修改所述多个程序指令中的一个或多个，以便拦截异常处理进程，并且使用键值将所述异常处理进程重定向到所述第一额外信息的位置的装置。

15. 根据权利要求 12 所述的系统，其中所述代码包括第二额外信息，并且其中所述第二额外信息是第二类型的不可执行信息，以及第一和第二类型的不可执行信息是不同的。

16. 根据权利要求 15 所述的系统，其中第二类型的不可执行信息是源信息或注释信息。

17. 根据权利要求 12 所述的系统，其中所述代码是 Java 类文件或 Java 归档 (JAR) 文件，以及所述多个程序指令是 Java 字节码指令。

18. 根据权利要求 12 所述的系统，其中所述代码是 .NET 代码文件，并且所述多个程序指令是公用中间语言 (CIL) 字节码指令。

19. 根据权利要求 12 所述的系统，其中所述语言是解释语言。

20. 根据权利要求 12 所述的系统，其中所述位置是保存移除的第一额外信息的存储位置，并且所述系统还包括：

用于将移除的第一额外信息保存在所述保存移除的第一额外信息的存储位置中的装置。

21. 根据权利要求 12 所述的系统，其中所述位置是保存第一额外信息的代码的存储位置，并且其中所述加载包括从所述保存第一额外信息的代码的存储位置加载代码。

22. 根据权利要求 12 所述的系统，其中所述位置是能够取回所述第一额外信息的代码的位置，并且其中所述加载包括从所述能够取回所述第一额外信息的代码的位置取回代码。

23. 根据权利要求 12 所述的系统，其中所述虚拟机是公用语言运行时 (CLR) 虚拟机或 Java 虚拟机 (JVM)。

通过分隔额外信息来内存优化虚拟机代码的方法和系统

技术领域

[0001] 本发明主要涉及虚拟机代码，尤其涉及通过从可执行代码中划分额外信息来内存优化虚拟机代码。

背景技术

[0002] 虚拟机技术的优点已得到广泛认可。在这些优点中，有一个优点是其能在单个主机平台上运行多个虚拟机，这样做可以更好地使用硬件能力，同时确保每个用户都享有“完整的”计算机特征。随着计算机软件复杂度的增加以及在日常生活和商业活动中对于软件系统的依赖度的增长，用户期望具有很高的软件运行性能。由于虚拟机软件通常是在内存受限的系统例如在 PDA 和智能电话这类内存往往少于传统计算机系统的无线设备上运行的，因此这种虚拟机软件的性能尤其重要。由此，人们努力减少虚拟机代码在运行时使用的内存，例如将本地函数调用减至最少、限制本地代码中提供的功能的类型、减小应用的大小、以及要求有效的编码。对于优化软件性能而言，减小内存使用率仍旧是一个关键目标。

发明内容

[0003] 因此，本发明的实施例包括一种用于优化虚拟机或解释代码的方法、计算机程序产品和系统，包括：接收包含多个程序指令以及第一额外信息的代码，其中该代码用供虚拟机运行的语言编写，并且其中第一额外信息是第一类型的不可执行信息；通过移除第一额外信息以及将其替换成引用第一额外信息的位置的键值 (key) 来修改该代码；以及执行经过修改的代码，并且如果在运行过程中发生请求第一额外信息的事件，则响应于所述事件使用该键值来定位处于所述位置的额外信息，并且从所述位置加载第一额外信息。

[0004] 特别地，在通过结合附图来考虑下文中的详细描述时，可以清楚了解本发明实施例的上述及其他特征和优点，其中不同附图中的相同附图标记被用于表示相同的组件。

附图说明

[0005] 图 1 是示出了根据本发明实施例的包含虚拟机的例示计算机系统的框图。

[0006] 图 2 是示出了根据本发明第一实施例的例示的划分及内存优化处理的框图。

[0007] 图 3 是示出了根据本发明第一实施例的图 2 划分处理作用于例示 Java 类文件的效果的框图。

[0008] 图 4A 和 4B 是示出了根据本发明第一实施例的划分及内存优化处理的流程图。

[0009] 图 5 是示出了根据本发明第二实施例的例示划分及内存优化处理的框图。

[0010] 图 6 是示出了根据本发明第二实施例的图 5 划分处理作用于例示 Java 类文件的效果的框图。

[0011] 图 7A 和 7B 是描述根据本发明第二实施例的划分及内存优化处理的流程图。

具体实施方式

[0012] 现在参考附图，图 1 示出的是一个根据本发明实施例的例示计算机系统。图 1 显示的计算机系统 10 包括处理器 12、内存 14、网络适配器 16 以及 Java 虚拟机 20，所有这些组件全都通过系统总线 18 可通信地耦接在一起。内存 14 可以用任何常规或其他的存储器或存储设备（例如 RAM、缓存、闪存等等）来实现，并且可以包括任何适当的存储容量。网络适配器 16 可以被实现为使得计算机系统 10 能够借助任何数量的任何适当通信介质（例如 WAN、LAN、因特网、内部网、有线网络、无线网络等等）通过共享内存或是采用其他任何适当的方式来与一个或多个其他系统通信。本实施例的计算机系统可以包括借助于任何常规或其他协议而在网络或其他通信信道上通信、并且可以使用任何类型的连接（例如有线、无线等等）来进行访问的任何常规或其他通信设备。

[0013] Java 虚拟机 20 包含各种组件，例如类加载器 21、异常处理机 (exception handler) 模块 22、差错报告模块 23、包含解释器 25 和即时 (JIT) 编译器 26 的运行引擎 24、类库 27 以及内存 28。应该理解的是，虽然这里论述的虚拟机 20 是 Java 虚拟机，但是本发明并不局限于 Java 技术，而是可以结合其他技术和虚拟机来实施，例如 CPython 虚拟机、.NET 以及公用语言运行时、Parrot 虚拟机、Rubinius 虚拟机或是其他任何适当的虚拟机。

[0014] 类加载器 21 加载代码，例如 Java 类 (.class) 或 Java 归档 (.jar) 文件之类的字节码文件，或是 CIL 字节码文件。该代码包含了从任何适合与虚拟机一起使用的语言中编译的程序指令，举例来说，适合与 CPython 虚拟机结合使用的语言包括 Python，适合与 Java 虚拟机 (JVM) 结合使用的语言包括 Java、Clojure、Fan、Groovy、IBMNetRexx、JavaFX Script、JavaScript、JRuby、Jython、JSchemeMacromedia ColdFusion、Rhino、Scala 等等，适合与 .NET 公用语言运行时 (CLR) 一起使用的语言包括 C#、C++/CLI、Cobra、F#、Fan、IronPython、IronRuby、J#、JScript.NET、受管 JScript、Mondrian、Nemerle、VB.NET 等等，适合与 Parrot 虚拟机一起使用的语言包括 Perl 6，以及适合与 Rubinius 虚拟机一起使用的语言包括 Ruby。该语言可以是解释语言或编译语言，举例来说，与 JVM 一起使用的语言可以是 JavaScript 或 Groovy，其中 JavaScript 通常是解释性的，而 Groovy 则通常是编译性的。

[0015] 计算机系统 10 既可以采用处理系统的形式来实现，也可以采用软件的形式来实现。计算机系统 10 可以由任何数量的常规或其他计算机系统或设备（例如计算机终端、个人计算机（如 IBM 兼容机、AppleMacIntosh、平板电脑、膝上型计算机等等）等等）、蜂窝电话、个人数字助理（例如 Palm Pre、Treo、iPhone 等等）等等来实现，并且可以包括任何商用操作系统（例如 AIX、Linux、OSX、Sun Solaris、Unix、Windows 等等）以及任何商用或定制软件（例如浏览器软件、通信软件、字处理软件等等）。这些系统可以包括多种类型的显示和输入设备（例如键盘、鼠标、语音识别等等），以便输入和 / 或查看信息。如果是在软件中实现的（例如作为虚拟映像），那么计算机系统 10 既可以在可记录介质（例如磁性、光学、软盘、DVD、CD 等等）上得到，也可以采用从信源经由通信介质（例如公告板、网络、LAN、WAN、内部网、因特网等等）下载的载波或信号的形式。

[0016] 图 2-4 描述的是本发明第一实施例的不同方面。在本实施例中，代码是通过从可执行代码中分离出额外信息以及通过单独存储额外信息而被划分的，由此可执行代码可以正常加载，而额外信息则可以只在需要的时候延迟加载。在保持用于调试以及为字段中的代码提供服务所必需的其他处理的额外信息的可访问性的同时，通过减少内存足迹，该方

案优化了内存使用率。

[0017] 图2示出的是根据本发明第一实施例可由图1系统实施的例示划分和内存优化处理。在本处理中，诸如Java类文件或CIL代码文件之类的代码30由后编译模块40接收，并且在后编译模块40，代码30中的额外信息被识别。该额外信息是不可执行信息，例如调试信息221、注释信息222或源信息223。通过移除额外信息产生经过修改的代码35，以及将额外信息保存在某个数据存储位置，例如数据库60、共享类缓存、嵌套内部类、或是独立的二进制对象，可以修改代码30。在经过修改的代码35中插入用于标识所述存储位置的唯一标识键值。

[0018] 在运行时，经过修改的代码35被正常加载到虚拟机20中。如果在运行时需要该额外信息，则系统可以定位并仅加载特定任务所需要的恰当类型的额外信息。举个例子，如果发生故障，那么异常处理进程22和差错报告机制23会请求调试信息221，以便抛出异常并报告差错，但其不需要注释信息222和源信息223。同样，如果代码使用了反射，那么反射进程可以请求源信息223，但是不会请求调试信息221。在图示的示例中，差错报告机制23是Java堆栈跟踪报告方法，该方法请求四个信息来产生堆栈跟踪70：类、方法、文件和行号。通常，该信息是由虚拟机在构造异常的时候填入的，但在这些实施例中，文件和行号会作为调试信息221的一部分而被移除，而所述调试信息则是保存在数据库60中的。相应地，异常处理进程22和差错报告机制23使用键值来找出保存了调试信息221的存储位置60，然后则通过加载调试信息221来填充堆栈追踪70。

[0019] 现在参考图3，该图显示了一个例示的Java类文件30，其中所述Java类文件30具有10个主要分量：幻数(magic number)201、版本202、常量池203、访问标记204、本类205、超类206、接口207、字段208、方法209以及属性210。属性可以包括调试信息221、注释信息222以及源信息223。例示的Java类文件30是通过从程序指令中划分出额外信息以及将额外信息保存在存储位置60而被修改的，在本范例中，所述额外信息是调试信息221、注释信息222以及源信息223。在经过修改的Java类文件35中，额外信息被键值224所取代，其中该键值标识的是存储该额外信息的存储位置。

[0020] 在Jvava技术的上下文中，调试信息221可以包括如下属性：

[0021] SourceFile：(依照类)没有路径的源文件名，例如ClassName.java；

[0022] SourceDebugExtension：通常是未使用的；

[0023] LineNumberTable：(依照方法)供调试器单步调试以及供异常堆栈跟踪以打印行号的行号(将字节码索引映射成源文件中的行号)；

[0024] LocalVariableTable：(依照方法)在单步调试时用于调试器的本地变量的名称；以及

[0025] LocalVariableTypeTable：(依照方法)对于一般类型(并且仅仅对于一般类型)来说，源文件中的本地变量的类型不同于VM中的类型。例如，当源文件类型可以是一般散列表<整数，字串>时，虚拟机将会看到散列表。

[0026] 对于代码中的程序指令的正常运行来说，该信息并不是必需的，这是因为它仅仅用于调试和堆栈跟踪，并且在运行程序的过程中是不起作用的。

[0027] 在Java技术的上下文中，注释信息222可以包括如下属性：

[0028] AnnotationDefault；

[0029] RuntimeVisibleAnnotations；

[0030] RuntimeInvisibleAnnotations；

[0031] RuntimeVisibleParameterAnnotations；以及

[0032] RuntimeInvisibleParameterAnnotations。

[0033] 此外，在 Java 技术的上下文中，源信息 223 可以包括如下属性：

[0034] EnclosingMethod：标识本地类的围绕方法；

[0035] Signature：标识用于类的一般签名、字段类型以及方法签名；

[0036] Deprecated：标识弃用的项目；

[0037] Synthetic：标识编译器产生的项目；以及

[0038] InnerClasses：标识内部和外部类关系（包括嵌套和匿名）

[0039] 对于代码中的程序指令的正常运行来说，该信息并不是必需的，这是因为运行虚拟机是不需要该信息的。该信息通常只用于反射。相应地，虽然移除该信息有可能会影响到大量使用反射的方法，但是一般来说，大多数的程序不会受到这种移除处理的影响。

[0040] 现在转到图 4A 和 4B，该图概括性地显示了第一实施例的用于划分和内存优化处理的处理 300 和 400。处理 300 可以在运行时之外的时间执行（例如在加载类的时候），而处理 400 则通常是在运行时执行的。以图 1 的系统为例，在处理 300 中，该系统在步骤 310 中接收代码，例如经过后处理器解析的代码，并且在步骤 320 标识所述代码中的额外信息。在步骤 330，系统从代码中移除额外信息，并且在步骤 340 中将移除的额外信息存入一个数据存储位置。在步骤 350，系统在代码中插入一个键值，以便引用所述额外信息的位置。在一个系统、例如 IBM J9 系统中，该处理可以由某种工具来执行，例如 Java 优化器 (JAPT) 类操作工具。

[0041] 在可选步骤 360，系统改写代码，以使其间接而不是直接地访问额外信息，例如，通过修改多个程序指令中的一个或多个，可以拦截异常处理进程，并且可以使用键值将其重定向到存储位置。该步骤是可选的，这是因为所述进程也可以采用其他方式来执行，例如修改类库来变更本地函数的操作，以便透明地加载所存储的额外信息，例 如 Throwable.getStackTrace()、Class.getDeclaredAnnotations() 以 及 Class.getGenericInterfaces()。举个例子，对于堆栈跟踪来说，通过修改类库，可以在 StackTraceElement 中存储程序计数器。该程序计数器是标引发生了 StackTraceElement 所代表的方法调用的字节码指令的索引，并且其被用于找到行号调试信息结构中的行号。在可选步骤 370 中，经过修改的代码可以保存在缓冲存储器中，例如共享类缓存，由此以后可以从共享类缓存中加载类，以便减少任何加载时间损失。

[0042] 如图 4B 所示，在运行时，系统在步骤 410 中加载经过修改的代码，然后在步骤 420 中确定是否发生了需要额外信息的事件，例如故障或反射处理。如果没有的话，该处理 400 结束。如果有的话，那么在步骤 430，系统使用键值来定位存储位置中的期望额外信息，并且在步骤 440 中加载该信息。然后，内存优化处理结束，并且系统可以使用取回的额外信息来执行任何期望的操作。所定位和加载的额外信息可以是一种或多种额外信息，例如先前描述的调试信息、注释信息以及源信息。系统可以只加载特定目的所需要的额外信息，举例来说，如果系统正在创建堆栈跟踪，那么它可以定位并只加载填充堆栈跟踪所需要的调试信息，或者如果系统使用了反射，那么它可以定位并只加载执行反射所需要的源信息。

[0043] 图 5-7 描述的是本发明第二实施例的不同方面。在该实施例中，代码是通过从可执行代码中分离出额外信息以及丢弃额外信息而被划分的。如果需要的话，可以从初始代码加载额外信息。在保持用于调试以及为字段中的代码提供服务所必需的其他处理的额外信息的可访问性的同时，通过减少内存足迹，该方案优化了内存使用率。

[0044] 图 5 示出的是根据本发明第二实施例可以由图 1 系统执行的例示划分和内存优化处理。在该处理中，诸如 Java 类文件或 CIL 代码文件之类的代码 30 由虚拟机 20 中的定制类加载器 21 接收，在虚拟机 20 中，所述代码 30 中的额外信息被标识。该额外信息是不可执行的信息，例如调试信息、注释信息或源信息。代码 30 是通过移除额外信息来产生经过修改的代码 35 而被修改的，并且在经过修改的代码 35 中插入用于标识初始代码 30 的位置的唯一标识键值。

[0045] 在运行时，经过修改的代码 35 被正常加载到虚拟机 20 中。如果在运行时需要额外信息，则系统可以定位并且仅加载特定任务所需要的恰当类型的额外信息。举例来说，如果发生故障，那么异常处理进程 22 和差错报告机制 23 会请求调试信息，以便抛出异常并报告差错，但其不需要注释信息和源信息。异常处理进程 22 和差错报告机制 23 使用键值来找出存储初始代码 30 的存储位置，并且从文件中加载调试信息，由此可以填充堆栈跟踪。同样，如果代码使用了反射，那么反射进程可以请求源信息，但是不会请求调试信息。如果初始代码是从服务器产生或下载的，而不是在本地存在的，那么可以再次产生或下载所述初始代码。

[0046] 现在参考图 6，该图显示的是具有 10 个基本分量的例示 Java 类文件 30 ：幻数 201、版本 202、常量池 203、访问标记 204、本类 205、超类 206、接口 207、字段 208、方法 209 以及属性 210。属性可以包括调试信息 221、注释信息 222 以及源信息 223。例示的 Java 类文件 30 是通过从程序指令中划分出额外信息以及将其丢弃在回收站 80 或是所收集的无用信息中而被修改的，在本范例中，所述额外信息是调试信息 221、注释信息 222 以及源信息 223。在经过修改的 Java 类文件 35 中，额外信息被键值 224 取代，该键值标识的是存储原始代码 30 或者可从例如通过下载其取回原始代码 30 的位置。

[0047] 现在转到图 7A 和 7B，该图概括性显示了第二实施例的划分和内存优化处理的处理 500 和 600。处理 500 可以在运行时之外的时间执行，或者可以由系统在运行时（例如在加载类的时候）执行。处理 600 通常是在运行时执行的。以图 1 的系统为例，在处理 500 中，该系统在步骤 510 加载初始代码，例如类加载器加载的代码，并且在步骤 520 中标识代码中的额外信息。在步骤 530，系统从代码中移除额外信息，并且在步骤 540 中将一个键值插入所述代码，以便引用存储了包含额外信息的初始代码或者可供取回所述代码的位置。在例如 IBM J9 系统的系统中，该处理可以由某种工具来执行，例如 Java 优化器 (JAPT) 类操作工具，或者在运行时，它可以由运行时 J9 系统作为扩展加载处理的一部分来执行。在由运行时系统执行时，经过修改的代码代表的是保存在虚拟机内部并且可选地保存在缓存内部的代码。

[0048] 在可选步骤 550，系统改写所述代码，以使其间接而不是直接地访问额外信息，举例来说，通过修改多个程序指令中的一个或多个，可以拦截异常处理进程，并且可以使用键值将其重定向到存储的代码。该步骤是可选的，这是因为所述进程也可以采用其他方式来执行，例如修改类库来变更本地函数的操作，以便透明地从所存储的代码加载

额外信息，例如 Throwable.getStackTrace()、Class.getDeclaredAnnotations() 以及 Class.getGenericInterfaces()。举个例子，对于堆栈跟踪来说，通过修改类库，可以在 StackTraceElement 中存储程序计数器。该程序计数器是标引发生了 StackTraceElement 所代表的方法调用的字节码指令的索引，并且其被用于找到行号调试信息结构中的行号。在可选步骤 560 中，经过修改的代码可以被存入缓冲存储器，例如共享类缓存，由此以后可以从共享类缓存中加载所述类，以便减小加载时间损失。

[0049] 如图 7B 所示，在运行时，系统在步骤 610 中加载经过修改的代码，然后在步骤 620 中确定是否发生了需要额外信息的事件，例如故障或反射处理。如果没有的话，该处理 600 结束。如果有的话，那么在步骤 630，系统使用键值来定位存储初始代码或从其可以取回初始代码的位置中的期望额外信息，并且在步骤 640 中加载该信息。然后，内存优化处理结束，并且系统可以使用取回的额外信息来执行任何期望的操作。所定位和加载的额外信息可以是一种或多种额外信息，例如先前描述的调试信息、注释信息以及源信息。系统可以只加载特定目的所需要的额外信息，举例来说，如果系统正在创建堆栈跟踪，那么它可以定位并只加载填充堆栈跟踪所需要的调试信息，或者如果系统使用了反射，那么它可以定位并只加载执行反射所需要的源信息。

[0050] 本领域技术人员应该理解，本发明的这些方面可以作为系统、方法或计算机程序产品来实现。相应地，本发明的这些方面可以采用全硬件实施例的形式、全软件实施例的形式（包括固件、驻留软件、微代码等等）或是组合了软件和硬件方面的实施例来实现，所有这些在这里可以统称为“电路”、“模块”或“系统”。此外，本发明的这些方面可以采用在包含了计算机可读程序代码的一个或多个计算机可读介质上实现的计算机程序产品的形式。

[0051] 可以使用一种或多种计算机可读介质的任何组合。计算机可读介质可以是计算机可读信号介质或计算机可读存储介质。举例来说，计算机可读介质可以是电子、磁性、光学、电磁、红外或半导体系统、装置或设备，亦或是前述各项的任何组合，但其并不局限于此。计算机可读存储介质的更具体的示例（非穷举列表）包括下列各项：具有一条或多条线路的电连接，便携式计算机碟片，硬盘，随机存取存储器 (RAM)，只读存储器 (ROM)，可擦写可编程只读存储器 (EPROM 或闪存)，光纤，便携式紧凑型只读存储器 (CD-ROM)，光学存储设备，磁性存储设备，或是前述各项的任何适当组合。在本文档的语境中，计算机可读存储介质可以是任何包含或存储了可供指令执行系统、装置或设备使用或与之结合使用的程序的有形介质。

[0052] 计算机可读信号介质可以包括那些包含了计算机可读程序代码的传播数据信号，其中举例来说，所述信号可以处于基带或者作为载波的一部分。这种传播信号可以采用多种形式中的任何一种，包括但不局限于电磁信号、光信号或是其任何适当的组合。计算机可读信号介质可以是并非计算机可读存储介质但却可以传递、传播或传送供指令执行系统、装置或设备使用或与之结合的程序的任何计算机可读介质。包含在计算机可读介质上的程序代码可以使用任何恰当的介质来传送，包括但不局限于无线、有线、光缆、RF 等等，或是前述各项的任何组合。

[0053] 用于执行本发明的各个方面的操作的计算机程序代码可以用一种或多种编程语言的任何组合来编写，包括：面向对象的编程语言，例如 Java、Smalltalk、C++ 等等，以及常规编程语言，例如“C”编程语言或类似的编程语言。程序代码既可以完全在用户计算机上

运行，也可以部分在用户计算机上运行，还可以作为独立软件包而部分在用户计算机以及部分在远程计算机上运行，或者完全在远程计算机或服务器上运行。在后一种情况中，远程计算机可以通过包括局域网 (LAN) 或广域网 (WAN) 在内的任何类型的网络与用户计算机相连，或者可以（例如通过使用因特网服务供应商的因特网）与外部计算机相连。

[0054] 应该理解，用于本发明实施例的计算机系统的软件可以用任何期望的计算机语言实施，并且可以由计算机领域的普通技术人员根据说明书中包含的功能性描述以及附图中示出的流程图来开发。作为示例，该软件可以用 C#、C++、Python、Java 或 PHP 编程语言来实现。更进一步，这里关于执行各种功能的软件的任何引用主要是指在软件控制下执行这些功能的计算机系统或处理器。

[0055] 作为替换，本发明实施例的计算机系统可以用任何类型的硬件和 / 或其他处理电路来实现。计算机系统的不同功能可以采用任何方式分布在任何数量的软件模块或单元、处理或计算机系统和 / 或电路中，其中计算机或处理系统既可以采用处于彼此本地的方式部署，或者采用彼此远离的方式部署，并且是可以借助任何适当的通信介质通信（例如 LAN、WAN、内部网、因特网、硬件、调制解调器连接、无线等等）的。

[0056] 本发明的这些方面是参考根据本发明实施例的流程图例证和 / 或方法、装置（系统）以及计算机程序产品的框图来描述的。应该理解流程图例证和 / 或框图中的每一个方框以及流程图例证和 / 或框图中的方框的任何组合都可以由计算机程序指令来实施。这些计算机程序指令可以被提供给通用计算机、专用计算机或其他可编程数据处理设备的处理器，以产生一机器，使得借助计算机或其他可编程数据处理设备的处理器运行的指令创建用于实施流程图和 / 或框图的一个或多个方框中规定的功能 / 操作的手段。

[0057] 这些计算机程序产品还可以保存在计算机可读介质中，其中所述介质可以指导计算机、其他可编程数据处理装置或其他设备以特定的方式运作，以使保存在计算机可读介质中的指令产生包含指令的制品，其中该指令实施的是流程图和 / 或框图的一个或多个方框中规定的功能 / 操作。计算机程序指令还可以加载到计算机、其他可编程数据处理装置或其他设备中，以便在计算机、其他可编程装置或其他设备上执行一系列操作步骤，从而产生计算机实施的处理，由此，在计算机或其他可编程装置上运行的指令提供用于实施流程图和 / 或框图的一个或多个方框中规定的功能 / 操作。

[0058] 适合存储和 / 或执行程序代码的处理系统可以由任何常规或其他计算机或处理系统来实施，其中所述系统优选地配备了显示器或监视器，基础（例如包括处理器、内存和 / 或内部或外部通信设备（例如调制解调器、网卡等等）以及可选输入设备（例如键盘、鼠标或其他输入设备））。该系统可以包括通过系统总线直接或间接耦合到内存部件的至少一个处理器。内存部件可以包括在实际执行程序代码的过程中使用的本地存储器，大容量存储器，以及通过为至少某个程序代码提供临时存储来减少运行过程中必须从大容量存储器中取回代码的次数的缓存存储器。输入 / 输出或 I/O 设备（包括但不局限于键盘、显示器、指示设备等等）既可以直接耦合到系统，也可以通过居间的 I/O 控制器而被间接地耦合到系统。网络适配器还可以耦合到系统，以使系统能够通过居间的私有或公共网络而被耦合到其他处理系统、或远端打印机或存储设备。调制解调器、电缆调制解调器和以太网卡仅仅是当前可用的网络适配器中的少量适配器。

[0059] 附图中的流程图和框图示出了根据本发明不同实施例的系统、方法和计算机程序

产品的可行实施方式的架构、功能和操作。就此而论，流程图或框图中的每一个方框都可以代表一个模块、分段或代码部分，包含用于实施一个或多个指定的逻辑功能的一个或多个可执行指令。此外还应该指出，在一些替换的实施方式中，方框中标注的功能有可能不是按照附图给出的顺序发生的。例如，连续显示的两个方框实际有可能是以基本同时的方式运行的，或者这些方框有时可以按照相反的顺序运行，这一点取决于所涉及的功能。应该指出的是，框图和 / 或流程图例证中的每一个方框以及方框和 / 或流程图例证中的方框的组合可以由执行规定的功能或操作的基于硬件的专用系统实施，或者由专用硬件和计算机指令的组合来实施。

[0060] 这里使用的术语旨在描述特定的实施例，而不是对本发明加以限制。除非在上下文中以别的方式清楚规定，否则这里使用的单数形式“一”、“一个”、“该”也包含了复数的形式。此外还应该理解，本说明书中使用的术语“包括”和 / 或“包含”规定的是存在所陈述的特征、整体、步骤、操作、部件和 / 或组件，但是并不排除一个或多个特征、整体、步骤、操作、部件、组件和 / 或其群组的存在或添加。

[0061] 下列权利要求中的相应结构、材料、操作以及所有装置或步骤加功能部件的等同物旨在包含与特别要求保护的其他要求保护的部件相结合来执行功能的任何结构、材料或操作。本发明的描述是出于例证和描述目的给出的，但是并不仅限于所公开的发明形式。对本领域技术人员来说，在没有脱离本发明的范围和精神的情况下，很多的修改和变更都是显而易见的。选择和描述这些实施例是为了最好地说明本发明的原理以及实际应用，并使本领域的其他普通技术人员能够理解具有与所设想的特定用途相适合的不同修改方式的本发明的不同实施例。

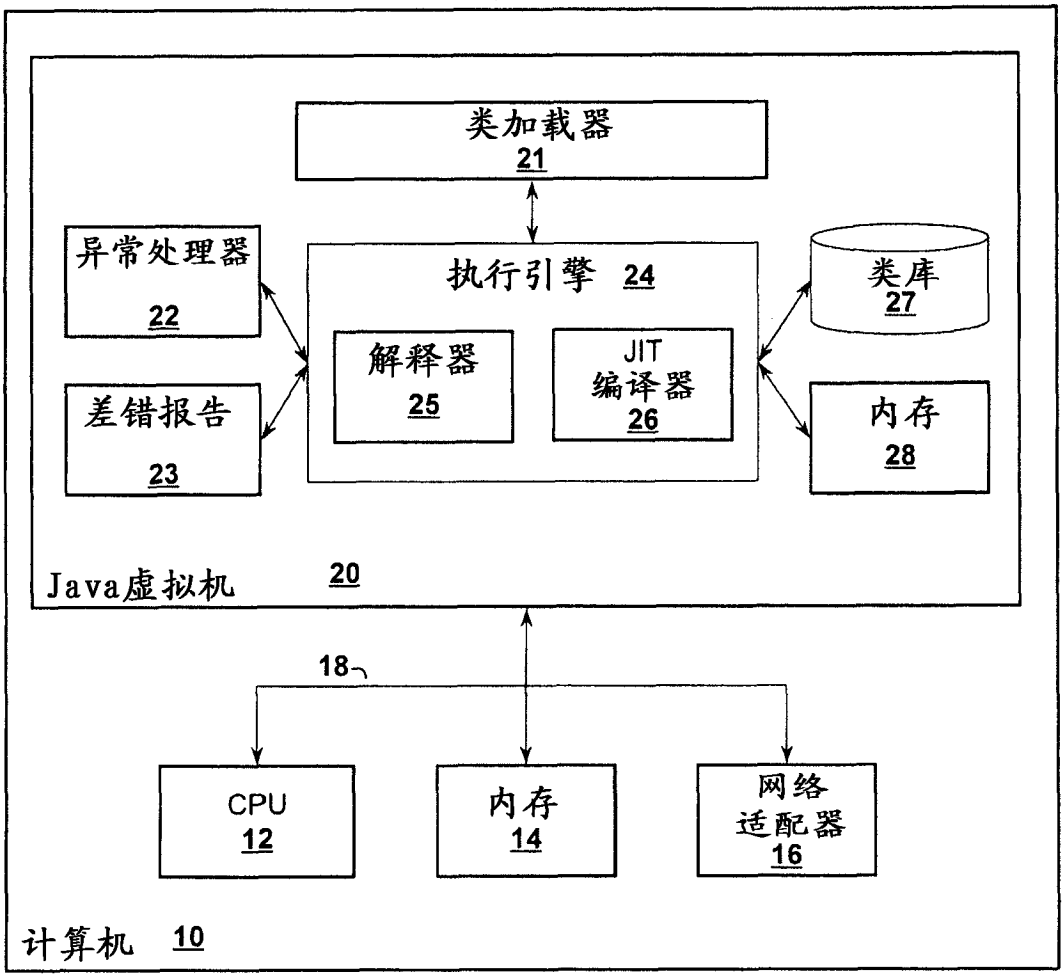

类加载器
21
异常处理器
22
执行引擎 24
类库
27
解释器
25
JIT
编译器
26
差错报告
23
内存
28
Java虚拟机 20
18
CPU
12
内存
14
网络
适配器
16
计算机 10

图 1

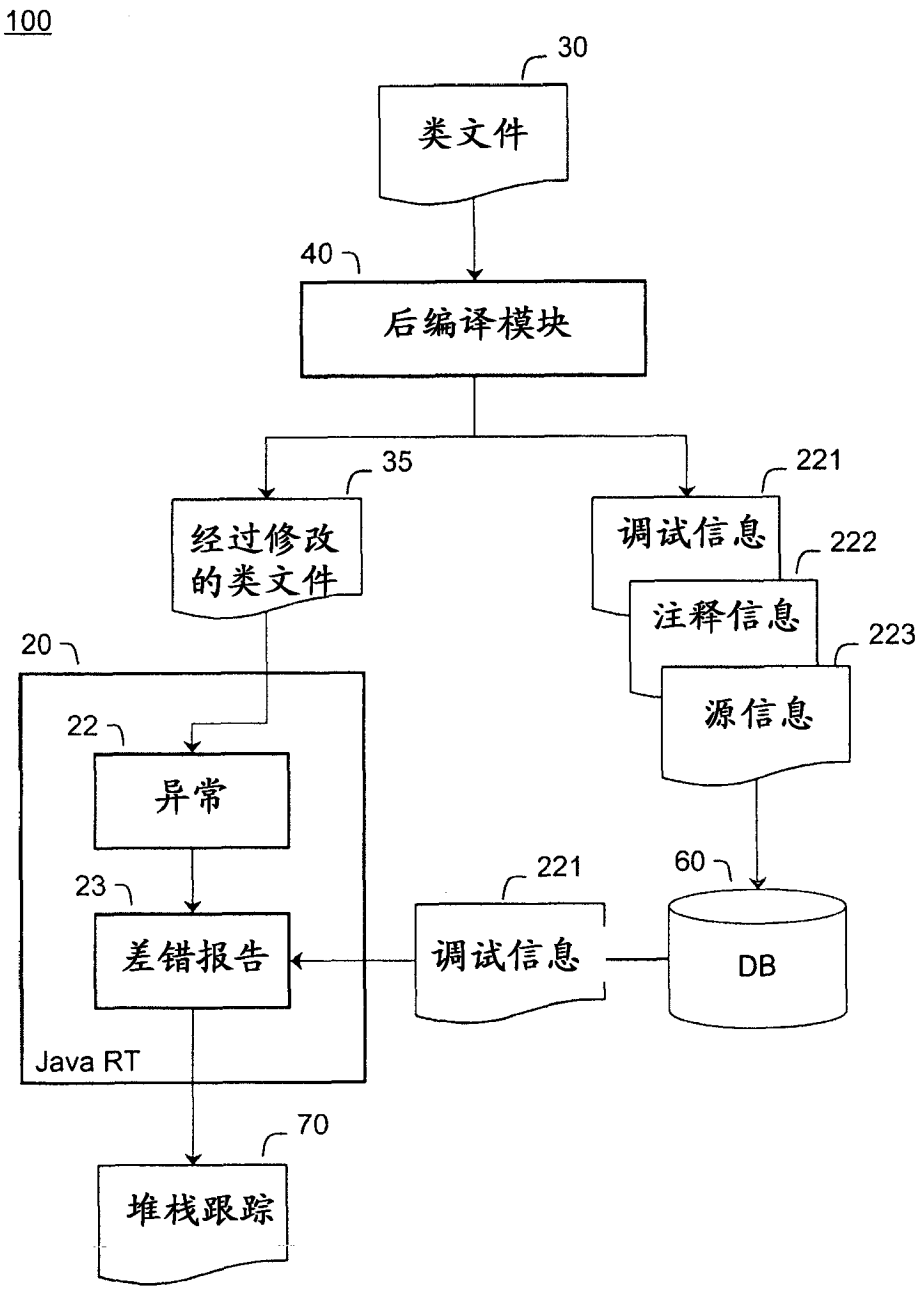

100
30
类文件
40
后编译模块
35
经过修改的类文件
221
调试信息
222
注释信息
223
源信息
20
22
异常
23
差错报告
Java RT
221
调试信息
60
DB
70
堆栈跟踪

图 2

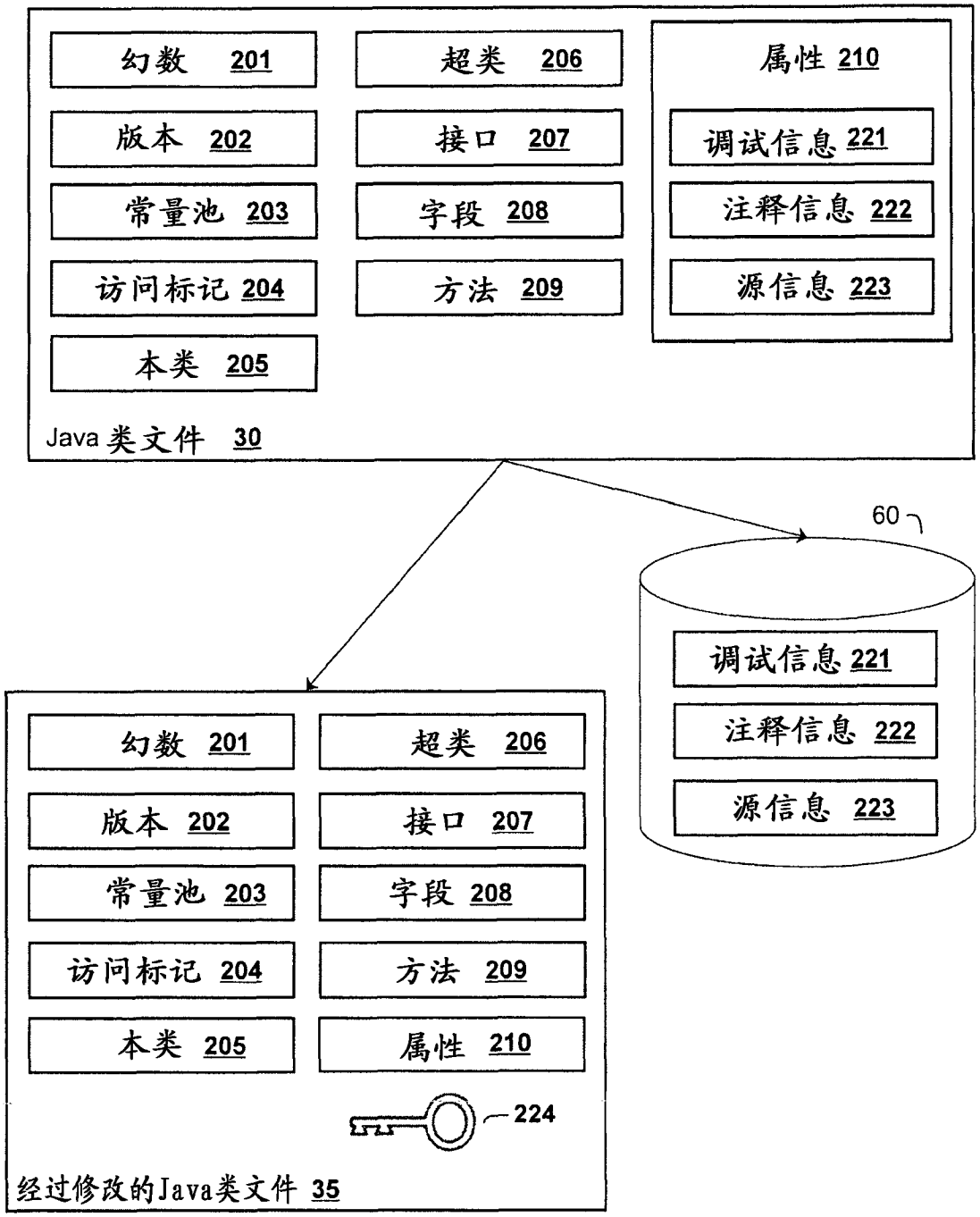

幻数 201
超类 206
属性 210
版本 202
接口 207
调试信息 221
常量池 203
字段 208
注释信息 222
访问标记 204
方法 209
源信息 223
本类 205
Java 类文件 30
60
调试信息 221
注释信息 222
源信息 223
幻数 201
超类 206
版本 202
接口 207
常量池 203
字段 208
访问标记 204
方法 209
本类 205
属性 210
224
经过修改的Java类文件 35

图 3

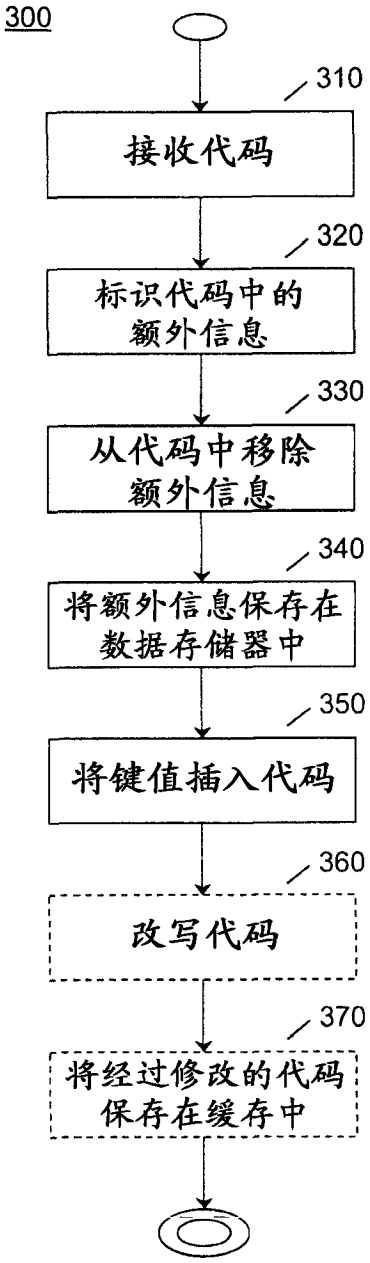

300
310
接收代码
320
标识代码中的额外信息
330
从代码中移除额外信息
340
将额外信息保存在数据存储器中
350
将键值插入代码
360
改写代码
370
将经过修改的代码保存在缓存中

图 4A

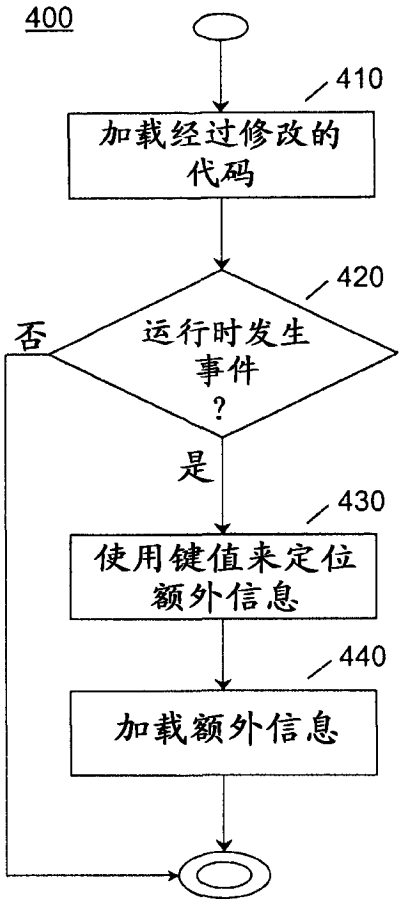

400
410
加载经过修改的代码
420
运行时发生事件？
否
是
430
使用键值来定位额外信息
440
加载额外信息

图 4B

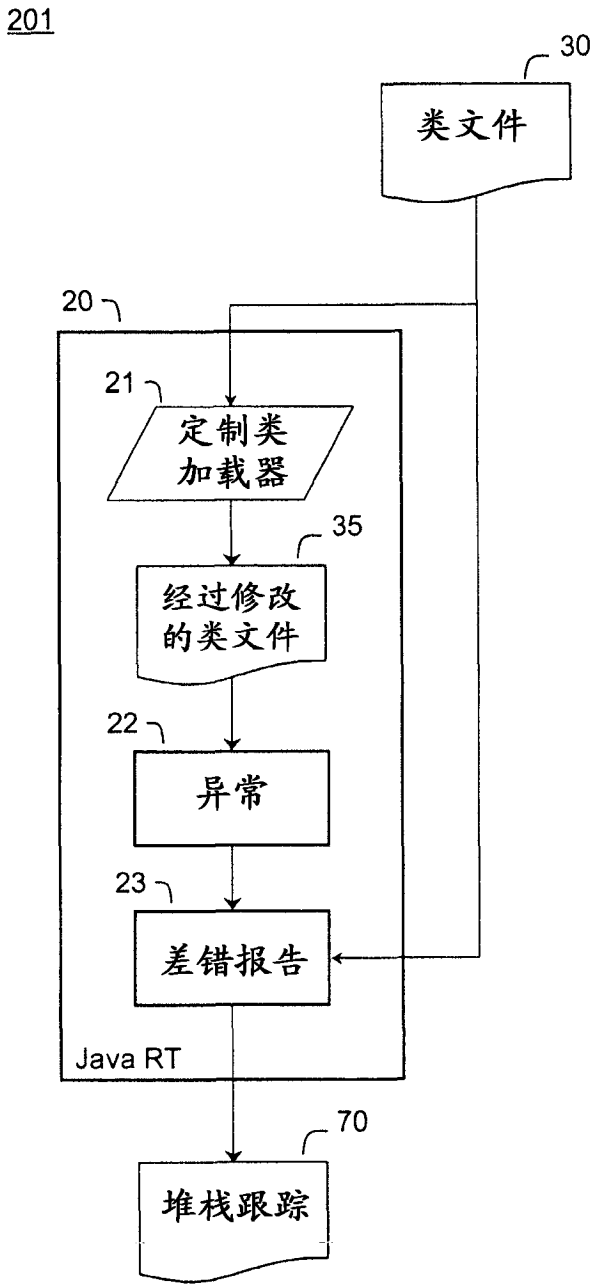

201
30
类文件
20
21
定制类
加载器
35
经过修改
的类文件
22
异常
23
差错报告
Java RT
70
堆栈跟踪

图 5

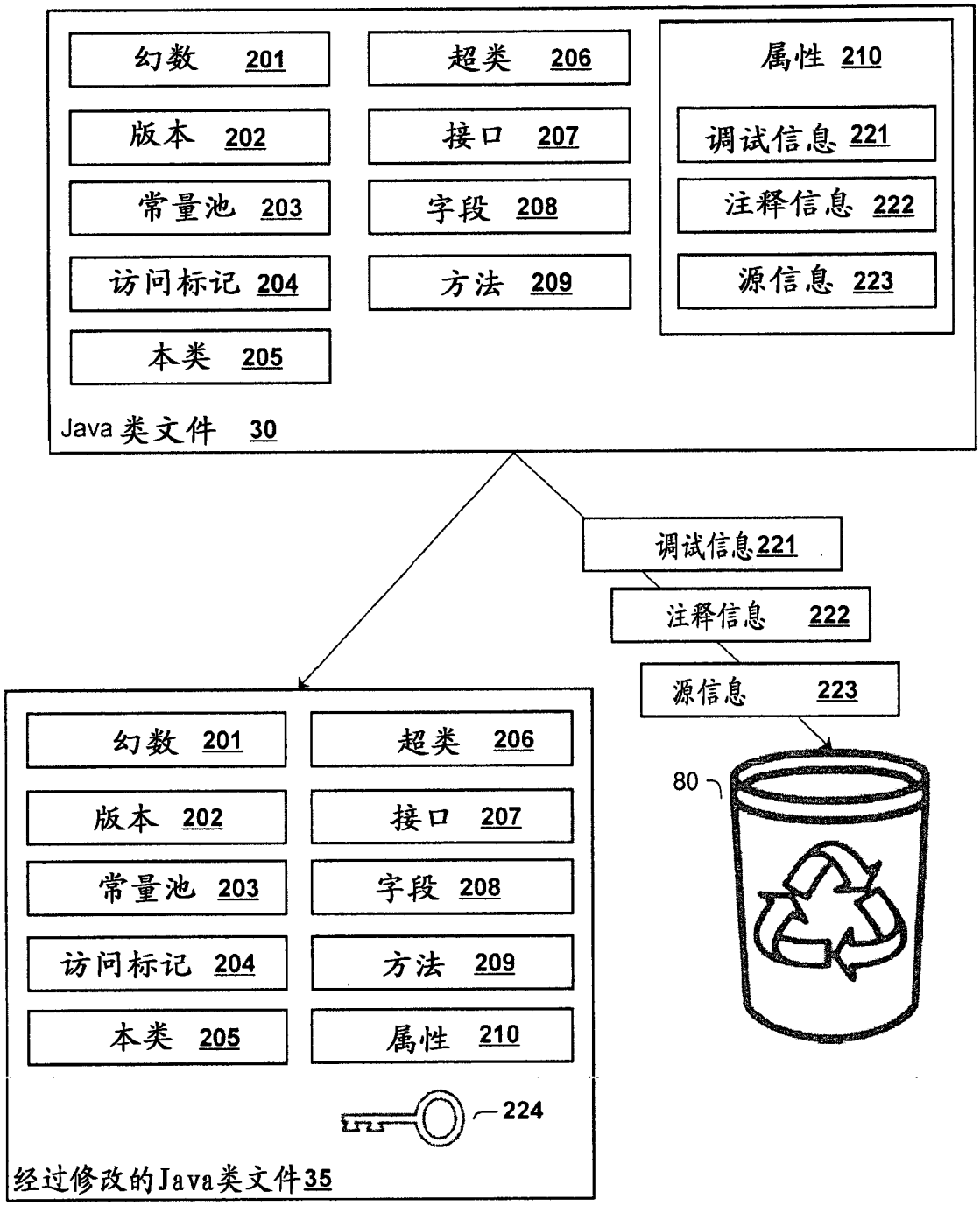

幻数 201
超类 206
属性 210
版本 202
接口 207
调试信息 221
常量池 203
字段 208
注释信息 222
访问标记 204
方法 209
源信息 223
本类 205
Java 类文件 30
调试信息221
注释信息 222
源信息 223
80
幻数 201
超类 206
版本 202
接口 207
常量池 203
字段 208
访问标记 204
方法 209
本类 205
属性 210
224
经过修改的Java类文件35

图 6

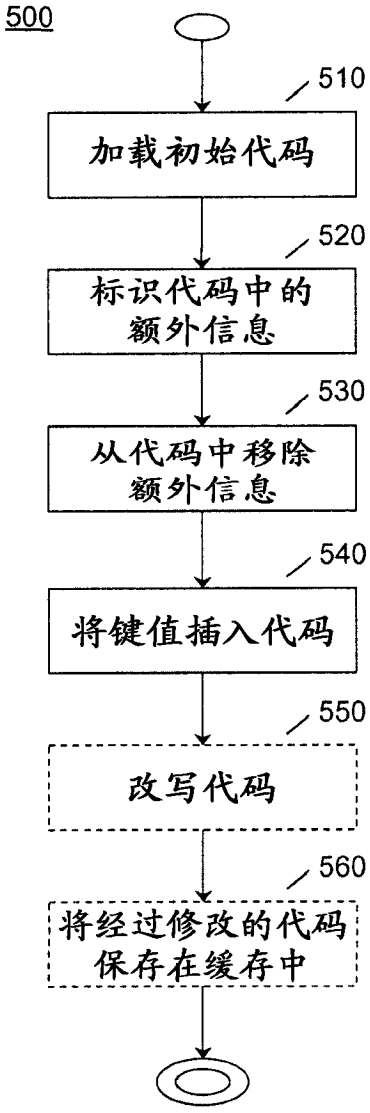

500
510
加载初始代码
520
标识代码中的额外信息
530
从代码中移除额外信息
540
将键值插入代码
550
改写代码
560
将经过修改的代码保存在缓存中

图 7A

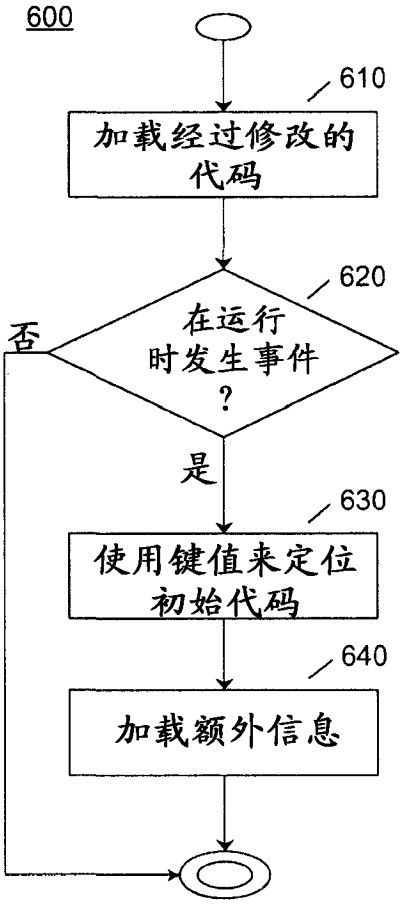

600
610
加载经过修改的代码
620
在运行时发生事件？
否
是
630
使用键值来定位初始代码
640
加载额外信息

图 7B