[19] 中华人民共和国国家知识产权局

[51] Int. Cl[7]
G06F 9/44

[12] 发明专利申请公开说明书

[21] 申请号 200510007833.1

[43] 公开日 2005 年 8 月 31 日

[11] 公开号 CN 1661551A

[22] 申请日 2005.1.27
[21] 申请号 200510007833.1
[30] 优先权
[32] 2004. 2.27 [33] US [31] 10/789,201
[71] 申请人 微软公司
地址 美国华盛顿州
[72] 发明人 A·C·石 C·W·布鲁姆
J·S·格雷 J·C·霍金斯
S·E·特罗布里吉

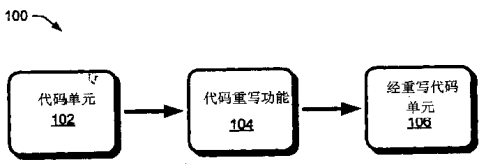
[74] 专利代理机构 上海专利商标事务所有限公司
代理人 谢喜堂

权利要求书 7 页 说明书 17 页 附图 10 页

[54] 发明名称 代码重写

[57] 摘要

系统和方法通过使用可扩展、可组合的一组代码重写器提供代码单元的重写和变换，这些重写器可在贯穿代码单元开发、调度和执行的各个阶段实现。 所述系统和方法向程序开发者和系统管理员提供功能强大的方法，以在贯穿程序的开发、调度和执行的不同阶段上实现相当独立于这种程序的代码变换，且并未大大增加源程序、编译器、或执行环境的复杂度。



ISSN1008-4274

1. 包括多个数据结构的一种或多种计算机可读介质，其特征在于，所述多个数据结构包括：

一代码单元，其具有可执行指令；

一重写器列表，其标识至少一个重写器；以及

一个或多个重写器，其包括所述至少一个重写器，其中每个重写器能够重写所述代码单元。

2. 如权利要求 1 所述的一种或多种计算机可读介质，其特征在于，还包括一重写管理器，所述重写管理器具有可执行指令，所述可执行指令被配置为从所述一个或多个重写器访问所述至少一个重写器，并根据所述代码单元执行所述至少一个重写器，从而产生一经重写代码单元。

3. 如权利要求 1 所述的一种或多种计算机可读介质，其特征在于，还包括一高速缓存，所述重写管理器还被配置为把所述经重写代码单元存储到所述高速缓存中。

4. 如权利要求 1 所述的一种或多种计算机可读介质，其特征在于，所述重写器列表从包括以下内容的组中选取：

所述代码单元中一个或多个定制属性的列表；

一安全政策中的列表；

一安装工具中的列表；

一配置文件中的列表；以及

一 XML（可扩展标记语言）文件中的列表。

5. 如权利要求 2 所述的一种或多种计算机可读介质，其特征在于，所述重写管理器是从包括以下内容的组中选取的一模块：

一单机模块；

一操作系统模块；

一执行环境模块；

一 JIT 编译器模块；

一源代码编译器模块；以及

一安装工具，其被配置以安装所述代码单元。

6. 如权利要求 1 所述的一种或多种计算机可读介质，其特征在于，所述计算机从包括以下内容的组中选取：

一开发计算机，其被配置以创建所述代码单元；

一中间计算机，其被配置以调度所述代码单元；以及

一调度计算机，其被配置以执行所述代码单元。

7. 包括一代码单元的一种或多种计算机可读介质，其特征在于，所述代码单元具有可执行指令，所述指令被配置为用来：

启动所述代码单元的变换；以及

标识一个或多个重写器以实现所述变换。

8. 如权利要求 7 所述的一种或多种计算机可读介质，其特征在于，所述一个或多个重写器是多个重写器，每个重写器被配置以实现所述代码单元的一唯一变换，所述代码单元还包括可执行指令，所述指令被配置用来对所述多个重写器排序以特定顺序实现所述唯一变换。

9. 如权利要求 7 所述的一种或多种计算机可读介质，其特征在于，所述代码单元还包括可执行指令，所述指令被配置用来引导所述一个或多个重写器以根据所述代码单元中的被标识元素来实现所述变换。

10. 如权利要求 7 所述的一种或多种计算机可读介质，其特征在于，所述启动包括：

标识在其中应实现所述变换的一环境；以及

仅当所述代码单元在所述被标识环境中时，启动所述变换。

11. 如权利要求 10 所述的一种或多种计算机可读介质，其特征在于，所述标识一环境包括：

标识一源代码编译环境；

标识一执行前编译环境；

标识一安装时编译环境；

标识一执行环境；以及

标识一安装环境。

12. 如权利要求 7 所述的一种或多种计算机可读介质，其特征在于，所述标识一个或多个重写器包括列出所述代码单元中的所述一个或多个重写器。

13. 如权利要求 7 所述的一种或多种计算机可读介质，其特征在于，所述标识一个或多个重写器包括标识与具有所述一个或多个重写器列表的所述代码单元分

开的一文件。

14. 包括计算机可执行指令的一种或多种计算机可读介质，其特征在于，所述指令被配置用来：

接收具有可执行指令的一代码单元；

确定用来重写所述代码单元的至少一个重写器；

调用所述至少一个重写器；以及

根据所述代码单元执行所述至少一个重写器，以产生一经重写代码单元。

15. 如权利要求 14 所述的一种或多种计算机可读介质，其特征在于，还包括可执行指令，其被配置用来在所述执行之前证实所述代码单元和所述证实一个重写器的可信度。

16. 如权利要求 15 所述的一种或多种计算机可读介质，其特征在于，所述证实包括鉴别一数字签名。

17. 如权利要求 14 所述的一种或多种计算机可读介质，其特征在于，还包括可执行指令，其被配置用来把所述经重写代码单元存储在一高速缓存中。

18. 如权利要求 17 所述的一种或多种计算机可读介质，其特征在于，还包括可执行指令，其被配置用来：

接收执行所述代码单元的一指令；

识别已重写的所述代码单元；

从所述高速缓存中装入所述经重写代码单元；以及

执行所述经重写代码单元。

19. 如权利要求 17 所述的一种或多种计算机可读介质，其特征在于，还包括可执行指令，其被配置用来：

为所述经重写代码单元产生一数字签名；以及

把所述数字签名关联到所述经重写代码单元。

20. 如权利要求 14 所述的一种或多种计算机可读介质，其特征在于，所述至少一个重写器是多个重写器，所述一种或多种计算机可读介质还包括可执行指令，其被配置用来对所述多个重写器排序以特定重写顺序重写所述代码单元。

21. 如权利要求 20 所述的一种或多种计算机可读介质，其特征在于，所述排序包括：

访问一重写器列表；以及

根据所述重写器列表设定所述重写顺序。

22. 如权利要求 21 所述的一种或多种计算机可读介质，其特征在于，所述访问包括访问在从包括以下内容的组中选取的位置上访问所述重写器列表：

所述代码单元；

关联于所述代码单元的一单独文件；以及

一系统政策。

23. 如权利要求 14 所述的一种或多种计算机可读介质，其特征在于，对所述至少一个重写器的所述确定包括读取一重写器列表。

24. 如权利要求 14 所述的一种或多种计算机可读介质，其特征在于，其包含在从包括以下内容的组中选取的一工具中：

一源代码编译器；

一安装工具；

一受控执行环境；

一单机重写管理工具；以及

一 JIT（运行时编译执行）编译器。

25. 如权利要求 14 所述的一种或多种计算机可读介质，其特征在于，所述计算机从包括以下内容的组中选取：

一开发计算机，其被配置以开发所述代码单元；

一中间计算机，其被配置以安装所述代码单元；以及

一调度计算机，其被配置以执行所述代码单元。

26. 一种计算机，其特征在于，包括：

一代码单元；

一组可组合的重写器，每个重写器被配置以唯一方式重写所述代码单元；

一重写管理器，其被配置以从所述可组合重写器组中标识一个或多个重写器，并根据所述代码单元执行所述被标识的一个或多个重写器。

27. 如权利要求 26 所述的计算机，其特征在于，还包括一重写高速缓存，所述重写管理器还被配置为把所述经重写代码单元存储到所述重写高速缓存中。

28. 如权利要求 26 所述的计算机，其特征在于，还包括一重写器列表，所述重写管理器从所述重写器列表中标识所述一个或多个重写器以根据所述代码单元执行之。

29. 如权利要求 28 所述的计算机，其特征在于，所述重写列表上从包括以下内容的组中选取的一组件：

所述代码单元中的一重写器列表；

一独立文件中的一重写器列表；

一安全政策中的一重写器列表；以及

一安装工具中的重写器列表。

30. 如权利要求 26 所述的计算机，其特征在于，还包括：

一第一数字签名，其关联于所述代码单元；以及

一组第二数字签名，每个第二数字签名关联于来自所述可组合重写器组的一特定重写器；

其中所述重写管理器还被配置以确定是否基于所述第一数字签名信任所述代码单元，确定是否基于来自所述第二数字签名组的对应第二数字签名信任来自所述被标识的一个或多个重写器的每个重写器，并仅当所述代码单元和来自所述被标识的一个或多个重写器的每个重写器都被信任时，根据所述代码单元来执行所述被标识的一个或多个重写器。

31. 如权利要求 30 所述的计算机，其特征在于，还包括一第三数字签名，其由所述重写管理器关联到所述经重写代码单元，并被配置以证实所述经重写代码单元被信任。

32. 如权利要求 26 所述的计算机，其特征在于，所述重写管理器是从包括以下内容的组中选取的一组件：

一单机重写模块；

一重写模块，其被配置为一操作系统的一部分；

一重写模块，其被配置为一安装工具的一部分；以及

一重写模块，其被配置为一安全政策的一部分。

33. 如权利要求 26 所述的计算机，其特征在于，从包括以下内容的组中选取：

一开发计算机，其被配置以开发所述代码单元；

一中间计算机，其被配置以安装所述代码单元；以及

一调度计算机，其被配置以执行所述代码单元。

34. 一种方法，其特征在于，包括：

接收一可执行代码单元；

确定需要重写的所述代码单元；

确定用于重写所述代码单元的一个或多个重写器；以及

根据所述代码单元运行所述一个或多个重写器，以产生一经重写代码单元。

35. 如权利要求 34 所述的方法，其特征在于，还包括在所述运行之前证实所述代码单元和所述一个或多个重写器的可信度。

36. 如权利要求 35 所述的方法，其特征在于，所述证实包括鉴别一数字签名。

37. 如权利要求 34 所述的方法，其特征在于，还包括把所述经重写代码单元存储在一高速缓存中。

38. 如权利要求 37 所述的方法，其特征在于，还包括：

接收执行所述代码单元的一指令；

识别存储在所述高速缓存中的经重写代码单元；

从所述高速缓存中装入所述经重写代码单元；以及

执行所述经重写代码单元。

39. 如权利要求 37 所述的方法，其特征在于，还包括：

为所述经重写代码单元产生一数字签名；以及

把所述数字签名关联到所述经重写代码单元。

40. 如权利要求 34 所述的方法，其特征在于，还包括对所述一个或多个重写器排序，以特定重写顺序重写所述代码单元。

41. 如权利要求 40 所述的方法，其特征在于，所述排序包括：

访问一重写器列表；以及

根据所述重写器列表设定所述重写顺序。

42. 如权利要求 34 所述的方法，其特征在于，所述确定一个或多个重写器包括读取一重写器列表。

43. 如权利要求 42 所述的方法，其特征在于，所述读取包括在从包括以下内容的组中选取的位置上读取所述重写器列表：

所述可执行代码单元；以及

关联于所述可执行代码单元的一独立文件。

44. 一种方法，其特征在于，包括：

接收一代码单元；

确定要由一重写器重写所述代码单元；

确定是否信任所述代码单元和所述重写器；

如果信任所述代码单元和所述重写器，则根据所述代码单元运行所述重写器以产生一经重写代码单元；

把所述经重写代码单元存储在一高速缓存中。

45. 如权利要求 44 所述的方法，其特征在于，还包括：

为所述经重写代码单元产生一数字签名；以及

把所述数字签名附加到所述经重写代码单元。

46. 如权利要求 45 所述的方法，其特征在于，还包括：

接收执行所述代码单元的一调用；

识别所述代码单元已被重写；

从所述高速缓存装入所述经重写代码；

证实附加在所述经重写代码单元的所述数字签名；以及

如果所述证实表明所述经重写代码单元是安全的，则执行所述经重写代码单元。

47. 如权利要求 44 所述的方法，其特征在于，所述重写器是一应用兼容性重写器，且所述对要重写所述代码单元的确定包括：

把所述代码单元标识为一应用；以及

查阅一应用兼容性重写数据库，以确定为了与当前执行环境兼容是否需要重写所述应用的任一部分。

代码重写

技术领域

本发明涉及重写代码，尤其涉及贯穿从开发直至执行的代码的整个生命周期能够重写代码的一组可组合重写器。

背景技术

程序开发一般遵循写源代码、编辑源代码、编译源代码、以及执行结果二元码的周期。开发周期可在各种计算环境范围内出现，例如，从单个开发者的计算机到包括开发计算机、中间计算机、和经一个或多个网络互相连接的企业调度计算机的计算装置系统。因而，周期也可包括中间步骤，它涉及例如在最终调度计算机上程序的安装和执行之前在中间计算机上安装和执行程序。

程序可能需要在其整个开发周期的某些点上进行变换，有各种原因。例如，可能需要把代码引入程序，以监视其执行并确定它是否在正确运行。这种“看门狗”代码可执行任务，诸如计算调用特定功能的次数或把特定功能的输出发送给文件。如果在程序的执行中发现了问题，需要进一步变换程序以包括被设计用来发现问题来源的附加代码。

一些编程语言支持允许进行诸如“看门狗”代码引入的某些变换的特征。然而，这些变换特征通常很有限。另外，这种特征通过语言专用编译器来实现。因而，可能一种编程语言支持“看门狗”代码变换特征而另一种却不支持。

在开发期间变换程序的其它原因可能包括将要调度程序的变化执行环境。例如，程序可作为操作系统、数据库、或应用程序的一部分运行。从一个环境到另一个环境，语义学或已定义的系统行为可变化，或者当环境改变时（诸如操作系统升级）它们在该环境中也会改变。因而，在不同的和变化的环境中程序的表现会不同。因此，保证程序在不同环境中以持续方式执行可能需要程序变换。这种变换可通过开发者把环境专用代码引入来源，或通过由编译器引入这种代码来完成。另一选项是变换执行环境。每一个这种选项都有大大增加程序、编译器、或执行环境复杂度的缺点。

因此，需要有一种方法，它在贯穿程序开发和调度周期的不同阶段上实现但并未大大增加源程序、编译器、或执行环境的复杂度代码变换。

发明内容

系统和方法通过使用可扩展、可组合的一组代码重写器提供代码单元的重写和变换。可应用在贯穿代码单元开发和调度的各个阶段上实现的重写器。重写管理器可因各种原因开始代码单元的重写，包括由开发者嵌入代码单元本身的信息、诸如安全或安装政策的系统政策等等。重写管理器标识一个或单个重写器并根据代码单元执行重写器，从而产生经重写代码单元或一系列经重写代码单元的连续版本。重写管理器也可排列重写器，以在代码单元上以特定顺序执行重写。在根据代码单元执行重写器之前，重写管理器确认代码单元和重写器两者的可信度。经重写代码被存储在高速缓存中，从而随后的执行代码单元的调用不会导致重写过程的重复。可直接从高速缓冲存储器访问经重写代码单元并执行之。

附图说明

相同参考标记贯穿所有附图指向相同组件和特征。

图 1 阐述了用来重写代码的示例性过程，该过程可能出现在各种开发和调度环境中。

图 2 阐述了适于实现代码重写的示例性计算环境。

图 3 阐述了适于实现代码重写的另一示例性计算环境。

图 4 示出了详述便于代码重写的各种组件的开发计算机、中间计算机、以及调度计算机的示例性框图表示。

图 5 示出了详述用来实现图 1 重写代码的示例性过程的示例性框图。

图 6 阐述用来实现代码重写过程的组件能作为操作系统组件的一部分被嵌入。

图 7 把用来实现代码重写过程的组件图示为单机模块和各种系统组件的一部分。

图 8 把用来实现代码重写过程的组件图示为单机模块和各种系统组件的一部分。

图 9 阐述用来实现代码重写过程的组件被包括在代码单元本身内。

图 10 阐述了与图 2 所示环境一致的示例性代码调度环境。

图 11 阐述了与图 2 所示环境一致的示例性代码调度环境。

图 12 阐述了称为应用兼容性重写器的示例性系统代码重写器。

图 13 示出了用来实现代码重写的示例性方法的框图。

图 14 阐述了适于实现诸如图 2 和图 3 示例性环境中所示的开发计算机、中间计算机、以及调度计算机的示例性计算环境。

具体实施方式

纵览

以下讨论指向通过贯穿代码单元的开发和调度的各个阶段上实现的可扩展、可组合的一组代码重写器，提供代码单元的重写和变换的系统和方法。所述系统和方法的优点包括向程序开发者和系统管理员提供功能强大的并未大大增加源程序、编译器、或执行环境复杂度的方法，以在贯穿代码单元的开发和调度的不同阶段上实现代码变换。

示例性环境

图 1 阐述了用来重写代码的基本过程 100，所述过程可出现在各种开发和调度环境中。用来重写代码单元 102 的过程 100 包括遭遇把一个或多个代码重写器应用到代码单元 102 的重写功能 104 的代码单元 102，以生成经重写代码单元 106。代码单元 102 可包括例如，由语言专用编译器编译成可在 PC 上执行的本机码的对象模块、用中间语言（例如抽象和/或机器无关语言）为受控代码环境调度的 DLL 或可执行文件集、以解释性语言写成的脚本（script）程序等等。

代码单元 102 可能在开发和调度的各个阶段遭遇重写功能 104。因此，重写功能 104 在代码单元 102 的开发、调度和执行中涉及的不同场景和不同计算机上可操作。一般而言，重写功能 104 接收代码单元 102、确定代码单元是否要重写、标识并排列要用来重写代码单元的重写器、并根据代码单元便于重写器的执行，导致经重写的代码单元 106。重写功能 104 一般包含和/或包括重写管理器、被配置重写代码单元 102 的一个或多个代码重写器、以及标识要应用到代码单元的重写器及其顺序的信息（例如重写器列表）。

图 2 和 3 阐述适于实现代码重写的示例性环境。仅作为示例而非限制地提供图 2 和 3 的示例性环境。因此，可以理解其中在代码开发和调度各个阶段可实现代码重写的许多其它计算环境是可能的，如此所述。

图 2 的示例性环境 200 显示了适于实现代码重写的开发和调度场景，其中代码在相对直接的分布信道上开发和分配。通常，代码单元 106 在开发计算机 202

上开发，并被分配到可在其中安装和执行的调度计算机 204 上。然而，可以理解，在计算机 202 上开发的代码单元 106 也能在计算机 202 上开发和执行，且代码重写功能 104 可根据代码单元 102 在计算机 202 和 204 上（其中一台或两台）实现以生成经重写代码单元 106。

代码单元 102 可通过若干方式从开发计算机 202 传送到调度计算机 204，包括例如经网络 206 将其从开发计算机 202 下载到调度计算机 204，或者在各种便携式介质 208 上（例如盘、压缩闪卡）将其从开发计算机 202 传送到调度计算机 204。根据特定系统配置，网络 206 可包括本地和远程连接。因而，网络 206 可包括例如调制解调器、电缆调制解调器、LAN（局域网）、WAN（广域网）、企业内部互联网、因特网、或其它适当的通信连接。

图 3 显示了适于实现代码重写的另一示例性环境 300，其中代码在更复杂的分布系统上开发和调度。环境 300 包括通常如上所述的开发计算机 202、调度计算机 204、加上中间计算机 302。调度计算机 204 在诸如公司防火墙内保护的公司内部互联网的企业计算系统 304 内。在图 3 的环境 300 中，代码单元 102 的开发和调度可包括在调度计算机 204 上调度代码单元 102 之前，经网络 206 把代码单元 102 从开发计算机 202 传送到中间计算机 302。通常，这种中间计算机 302 用来把代码单元分阶段，并将它们分配到调度计算机 204。发生在中间计算机 302 上的代码重写可与发生在开发或调度计算机上的代码重写不同。例如，中间计算机上的代码重写可把代码重写成与企业 304 的调度计算机 204 上某些系统和/或安全要求相适合。另外，代码单元 102 可作为其开发的一部分暂时在中间计算机 302 上安装和执行，以便在将其移到调度计算机 204 之前证实其性能。如下所述参照一示例性实施例，代码单元 102 可由一个或多个重写器在环境 300 各个计算机 202、204 和 302 上贯穿其开发、安装和执行的不同阶段进行重写。

图 2 和 3 的计算机 202、204 和 302 可实现为各种常规计算装置的任一种，包括例如台式 PC、笔记本或其它便携式计算机、工作站、服务器、典型计算机等等。计算机 202、204 和 302 通常能执行普通的计算功能，例如电子邮件、日程安排、任务组织、字处理、网络浏览等等。计算机 202、204 和 302 运行一操作系统，诸如来自华盛顿州 Redmond Microsoft®公司的 Windows®操作系统。开发计算机 202、调度计算机 204 和中间计算机 302 的一种示例性实现将参照图 14 在后面进行更详细的描述。

示例性实施例

图 4 是计算机 202、204 和 302 的框图表示，它阐述有助于描述图 3 示例性环境 300 中的代码重写的各种组件。代码重写可在计算机 202、204 和 302 的全部或任一台上出现，且一台计算机上的代码重写可影响另一台计算机随机的代码重写。例如，开发计算机 202 上重写功能 104 的实现可导致传送给中间计算机 302 的经重写代码单元。来自开发计算机 202 的经重写代码单元可能会被中间计算机 302 再次重写，然后传送到调度计算机 204，而在其上该代码单元可能会被又一次地重写，

图 4 中的开发计算机 202 包括由开发者用诸如 C、C++、C#、Java、Microsoft Visual Basic®等写成的源程序 400。每一台图 4 上的计算机 202、302 和 204 可包括各种编译器 402。从一台计算机到另一台计算机编译器 402 会不同，因此在开发计算机 202、中间计算机 302、和调度计算机 204 上它们被分别分配以参考标记 402(1)、402(2)和 402(3)。编译器 402 可包括语言专用编译器，其中每一个被配置为把专用语言的源程序编译成一个或多个代码单元 102。代码单元 102 可包括例如，编译成可在 PC 上执行的本机码的对象模块、适于在受控代码环境中执行的编译为二元码或中间语言（例如抽象和/或机器无关语言）的 DLL 或可执行文件集、以解释性语言写成的脚本程序等等。因此，编译器 402 可包括把源语言编译成运行于特定平台上的本机码的本机码编译器，以及把源语言编译成诸如中间语言或二元码的受控代码的受控代码编译器。

另外，编译器 402 可包括把诸如中间语言或二元码的平台无关代码（即受控代码）变换成在特定处理器上执行的本机码的执行前编译器。因而，编译器 402 还可包括在执行之前把受控代码（例如二元码、中间语言）编译成本机码的 JIT（运行时编译执行）编译器，以及把受控代码预先编译成本机码从而使其在安装后准备好执行的安装时编译（compile-on-install）编译器。

每台计算机 202、302 和 204 可包括代码重写功能 104，该功能被配置为接收代码单元 102、确定代码单元是否要重写、标识并排列要被应用以重写代码单元的重写器、并根据代码单元便于重写器的执行（例如，把代码单元 102 和适当的重写器装入 RAM 用来在处理器上执行）。如图 5 中所示，代码重写功能 104 包含和/或包括重写管理器 500、一个或多个被配置为重写代码单元 102 的代码重写器 502、以及标识要应用到代码单元的重写器及其顺序/序列的信息（例如重写器列表）。

每台计算机 202、302 和 204 还包括各种系统组件 404，诸如操作系统（OS）、系统政策（诸如安全政策）、应用、工具、执行环境等等。代码重写功能 104 的一种或多种组件（即重写管理器 500、重写器 502、重写器清单 504）可嵌入到任一

个或多个各种系统组件 404 中。例如，图 6 示出了代码重写功能 104 的所有组件，包括重写管理器 500、重写器 502 以及重写器列表 504 都作为操作系统的一部分被嵌入。该场景在例如某些应用变得不适合操作系统的升级版时会有用。把重写功能 104（重写管理器 500、重写器 502 以及重写器列表 504）包含在升级版操作系统中可被用来更改这种应用以确保其与升级版操作系统的兼容性。

图 7 和 8 显示了示例性实现，其阐述了代码重写功能 104 的组件可以是单机模块和/或可被分散给各种系统组件 404。图 7 中的重写管理器 500 是在调度计算机 204 上执行的系统安全政策 404(3)的组件，而重写器 502 和重写器列表 504 则是单机模块。因而，作为在调度计算机 204 上实现的安全政策 404(3)的一部分，重写管理器 500 将执行以确定是否需要重写代码单元以遵循安全政策。如果需要，重写管理器将继续，以通过使用重写列表 504 来标识并排列适当的重写器 502，然后根据代码单元执行被标识的重写器。在图 8 中，重写管理器 500 被示为中间计算机 302 上安装工具 404(2)的组件，而重写器 502 和重写列表 504 是单机模块。因而，当安装代码单元 102 时，安装工具 404(2)中的重写管理器 500 将执行以确定是否需要重写代码单元。如果需要，重写管理器将继续，以通过使用重写列表 504 来标识并排列适当的重写器 502，然后根据代码单元执行被标识的重写器。

图 9 示出了开发者如何将部分代码重写器组件包含到代码单元 102 本身中。如图 9 中所示，开发者已经把重写器列表 504 包含在代码单元 102 本身中。重写管理器 500 被示为是 JIT（运行时编译执行）编译器的一部分，而重写器 502 是单机模块。因而，开发者能使用代码单元 102 本身作为组合重写器 502 集的一种方法，以重写代码单元 102。例如，开发者可将重写指令（例如标记、属性、字符串等等）嵌入源程序 400 中，这些指令会被传递到新编译的代码单元 102 并指示重写管理器 500 使用一个或多个重写器来重写代码单元 102。这样，开发者有能力组合在贯穿代码单元开发和调度的各个阶段上实现的一组代码重写器。在由 JIT 编译器 402 对代码单元 102 进行随后的执行前编译之后，JIT 编译器 402 中的重写管理器 500 从嵌入在代码单元中的重写器列表 504 中，确定哪个重写器 502 要根据代码单元执行。代码单元 102 因而在其由 JIT 编译器 402 编译成本机码和随后的执行之前进行重写。

该过程使开发者能把一组重写器串起来或“组合”起来，其中每个重写器将轮流以特定方式更改代码单元。在一优选实施例中，当代码单元由重写器“组合”组中的每一个重写器持续重写时，每个经重写代码单元的格式保持一致。因此，作

为重写器输入的代码单元与作为重写器输出的代码单元有着一样的格式。这使开发者能组合任意组重写器，每个重写器以相同格式或表示摄入或释出代码单元。在一可选实施例中，每个重写器可以与输入代码格式不同的格式输出代码单元。然而，该可选实施例有点限制了开发者在组合重写器组时可达到的灵活性和随意性。

再参看图 5，示出了可对于图 4 中所示的任何或所有计算机 202、304 和 204 描述的一般代码重写过程。代码重写过程包括信任模型的实现，该模型特别适于说明在例如计算机 202、304 和 204 上贯穿其开发和调度发生在代码单元上的各种变换。因而，重写管理器 404 支持在重写代码单元前后提供证实其身份和可信度的信任模型。

在典型信任模型中，开发者把数字签名附在代码单元上。然后数字签名能被用来鉴别代码单元的身份，并确保未篡改或更改代码单元的原始分配版。一般提供有使用数字签名的技术和软件。在一种这样的技术中，可使用适当软件获取代码单元的散列或数学归纳。然后可使用从公私钥权限中获取的私钥对散列加密。加密钥是能与代码单元一起发送的数字签名。然后该组合被送往的用户（即计算机）对代码单元取散列（数学归纳），并使用公钥对散列解密。当散列相符，代码单元被鉴别。因而，一般信任原则是如果代码单元已从其原始“签名”版作了更改，那么其签名也将被更改且代码单元将不再受信任。

因为当前描述的代码重写系统包括对代码单元的重写，以上所述的典型信任模型并不适合。经一次或多次代码重写而更改的数字化“签名”代码单元基于在重写期间所做的变化将不能被证实为可靠。因此，图 5 所示的一般代码重写过程包括信任模型的实现，该模型特别适于说明在例如计算机 202、304 和 204 上贯穿其开发和调度发生在代码单元上的各种变换。

在图 5 所示的一般重写过程中，代码单元 102 由代码重写功能 104 的重写管理器 500 接收。该代码单元包括数字签名“签名#1”。重写管理器 500 开始时确定是否需要重写代码单元 102。参照图 6-9 如上所述，可用各种方法作出这种确定。例如，重写管理器 500 可确定当代码单元 102 从开发计算机 202 上的源程序 400 编译（编译成例如本机码、二元码、中间语言等）时，需要重写代码单元 102。

然后重写管理器 500 标识来自一组重写器 502 的适当重写器并对其进行排序。这时通过查阅一重写列表 504 而完成的。重写列表 504 向重写管理器 500 标识重写器，并提供根据代码单元 102 要执行的已标识重写器的顺序。然后重写管理器 500 从一个或多个重写器 502 的组合中访问已标识重写器，并将它们连同代码单元 102

装入存储器（RAM）。在根据代码单元 102 执行已标识重写器之前，重写管理器 500 鉴别重写器和代码单元 102 的数字签名（通过使用公钥解密散列或数学归纳的代码单元 102 的“签名 #1”和重写器 502 的“签名 #2”）以证实其身份和可信度。如果数字签名是可信的，重写管理器根据代码单元 102 执行已标识重写器以生成经重写代码单元 106。重写管理器还可为经重写代码单元 106 产生数字签名（例如“签名 #3”，是使用私钥加密的经重写代码单元 106 的散列或数学归纳），并把经重写代码 106 连同该签名一起存储在重写高速缓存 406 中。

一般而言，根据代码单元 102 执行的代码重写器 502 能打开磁盘上（如果已经不在存储器上）的代码单元文件，装入本机码、二元码、中间语言、元数据等，并把模块的逻辑机构、类型、方法等恢复成“存储器上”的数据结构。代码重写器可遍历这些数据结构，并通过例如改变方法（也称为“功能”）主体来任意插入或移除或更改代码来对类型和方法表示作任意的更改。代码重写器可插入或删除整个的模块、类型、方法、字段等等。当从代码单元自身指向重写时（例如由标记、属性、字符串等等；如上图 9；如下图 10 和 11），所作的更改可由附加的指示、标记、属性、字符串等引导，它们可以在代码单元外部，或嵌在代码单元中，有时则附属于代码单元中的特定元素或位置来更改或控制特定代码重写器对代码单元所做的改变。

再参看图 4，所述信任模型说明在计算机 202、304 和 204 上贯穿其开发和调度发生在代码单元 102 上的各种变换。例如，可以把开发计算机 202 的经重写代码单元 106(1)传送到中间计算机 302 作为代码单元 102(2)。尽管代码单元 102(2)已经从开发计算机 202 上的代码单元原始版 102(1)作了更改（即更改成经重写代码单元 106(1)），但代码单元 102(2)通常在代码更改发生后才会作数字签名。因此，中间计算机 302 将仍能基于伴随代码单元 102(2)的数字签名来鉴别该代码单元的身份和可信度。

重写高速缓存 406 用来存储经重写代码单元 106，从而用来对代码单元 102 编译、安装、执行等等的随后调用不会导致重写过程的重复。因而，当代码单元 102 已由特定重写器或重写器组 502 重写时，重写管理器 500 把结果经重写代码单元 106 存储在重写高速缓存 406 中，并在对于特定重写器或重写器组 502 有必要避免重复重写过程时，从高速缓存 406 访问经重写代码单元 106。一旦重写了代码单元 102 并将其作为经重写代码单元 106 存储在重写高速缓存 406 中，就无需又经过重写过程而可从高速缓存 406 直接访问并执行它。

图 10 和 11 示出了相对于在受控代码环境中代码而实现的代码重写功能的更具体实例。受控代码环境管理对以若干受支持语言的任一种所写成程序的执行，允许这些程序共享以任一语言写成的普通面向对象类。受控代码环境的实例包括 Sun Microsystems 为运行从 Java 语言编译的程序提供的 Java 虚拟机，以及是美国华盛顿州 Redmond 微软公司创建的.NET™ 平台一部分的公共语言运行时（CLR）。图 10 和 11 所示实例参照微软 CLR 在此描述。其它有关.NET™ 框架基础信息可在众多介绍性文本中得到，诸如 2003 年微软出版社 Pratt 的“介绍微软.NET”第三版。

CLR 是微软.NET™ 框架的核心，并为所有.NET 代码提供执行环境。因而，被创建以使用 CLR 并运行于 CLR 中的代码被称为“受控代码”。CLR 提供程序执行所需的各种功能和服务，包括运行时编译执行（JIT）编译、分配和管理存储器、实施类型安全、故障处理、线程管理和安全。CLR 在第一次调用.NET™ 例程之后装入。为了得到最佳性能，通常在执行之前把受控代码编译成本机码。

当编写受控代码时，调度单元被称为组装件（assembly），它是作为一个单元进行版本化及调度的一个或多个文件的组装件。组装件是.NET™ 框架应用的创建基块。所有受控类型和资源都包含于组装件中，并被标记为仅在组装件内可访问，或从其它组装件中的代码可访问。组装件被包装为 DLL 或可执行（EXE）文件。当可执行文件可自己运行时，在现有应用中必须提供 DLL。

为了简化讨论，图 10 和 11 示出了与以上所述参照图 2 的环境 200 相一致的示例性调度环境。因此，源程序 400 在开发计算机 202 上开发，并以相对直接的方式将其分配（例如经网络 206 下载或用磁盘传送）给调度计算机 204。另外，为了一致性，在受控代码环境中可被另外称为“组装件”的代码调度单元在图 10 和 11 的代码重写示例中被称为代码单元。

在图 10 中，名为“Hello.cs”的示例性源程序 400，用 C#编程语言写成，并包括请求特定代码重写器的“重写定制属性”，以及不是代码重写所特有的其它定制属性。属性都用方括弧“[]”指出。这两种显示的定制属性被嵌入程序源代码 400 中，并应用于类中以便描述这些类的特征性。因而，在图 10 的源程序 400 中，属性“[Attr]”是被应用于类“T”以描述类“T”部分特征的定制属性。

定制属性是开发者包括在源程序 400 中的可定制标注，它们可通过源程序 400 的编译 402 带到代码单元 102(1)。一般而言，.NET 框架实施例允许定制属性集的随意扩展。该定制属性不是固定的。相反，任何代码可引入新的用户定义定制属性或成为其目标。定制属性可应用于代码单元 102(1)中的类型、方法等等。应用于代

码单元 102(1)的其类型源自重写定制属性类型的定制属性被称为“重写定制属性”。重写定制属性将触发代码重写器(例如来自一组代码重写器 502)在代码单元 102(1)上的应用。因此重写定制属性能构成指定或确定要应用于代码单元 102(1)的重写列表 504。在源程序 400 中，重写定制属性“RCA1”和“RCA2”被分别标识在括弧“[RCA1]”和“[RCA2]”中。当要根据代码单元 102(1)应用已标识代码重写器时，开发者可在重写定制属性中指定，从而指示了用来根据代码单元 102(1)执行已标识重写器的顺序。

图 10 的编译器 402 是 C#编程语言专用的 C#编译器(CSC)。CSC 编译器 402 是可驻留在开发计算机 402 上的各种语言专用.NET 编译器之一，用来把各种语言专用源程序编译成包括公共的处理器无关的中间语言（IL）和元数据的代码单元。第一遍时，编译器 402 编译源程序 400、执行错误检查、并建立名为“hello.exe”的代码单元 102(1)（代码单元 102 的版本 1)。源程序 400 中包括重写定制属性((“[RCA1]”和“[RCA2]”)的定制属性表示被传递到代码单元 102(1)的元数据中。

图 10 和 11 的受控代码示例中的代码单元 102 包括 IL 和元数据。IL 以抽象方式描述发生在方法主体中的操作。元数据包括定制属性和表格，它们描述代码单元的结构：代码单元包括许多模块、每个模块包括许多类型定义、每个类型定义包括许多字段和方法（代码、功能)、每种方法有一返回类型和一列正式变量类型等等。在此情形中，元数据指示已从源程序 400 传递了两个重写器定制属性（RCA1 和 RCA2)，且有描述 T 的方法的称为“T”的类型和表格。T 的方法之一称为 Main。诸如属性“Attr”的其它定制属性也可存在，但是这些表示重写器定制属性且不触发任何代码重写。

第二遍时，编译器 402 的重写管理器组件 500 被配置为检查在第一遍时建立的代码单元 102(1)。检查之后，如果编译器 402 的重写管理器 500 发现任何请求代码重写器 502 的任何重写定制属性（例如 RCA1、RCA2)，重写管理器 500 装入存储器（未示出）并根据代码单元 102(1)运行被请求的代码重写器 502。重写管理器 500 又鉴别关联于代码单元 102(1)和代码重写器 502 两者的数字签名，以根据代码单元 102(1)在运行被请求代码重写器 502 之前证实其身份和可信度，如上一般所述。

在此例中，两个称为 RCA1 和 RCA2 的重写定制属性分别请求应用组装件重写器 CR1 和 CR2 以更改代码单元 102(1)。然而，定制属性 RCA2 被用指示（例如下标“T”）作了标记 ，它指示开发者不希望 CR2 代码重写器根据代码单元 102(1)在开发计算机 202 上运行。源程序 400 包括第二个重写定制属性 RCA2 上的

“DeploymentTime”标记，该标记由编译器 402 在第一遍时带到代码单元 102(1)。因此，第二遍时，编译器 402 的重写管理器 500 识别出：由已标记属性 RCA2 请求的 CR2 代码重写器不会根据代码单元 102(1)在开发计算机 202 上运行。因此，在此例中，仅有代码重写器 CR1 根据代码单元 102(1)在开发计算机 202 上运行。这导致已更改代码单元 102(2)（代码单元 102 的版本 2）。另外，编译器 402 用以上所述的方式任选地产生数字签名，并将其附加在已更改代码单元 102(2)上。代码单元 102(2)的最后版本被分配到调度计算机 204，用来安装和执行。注意图 10 中类 T 的方法“$Main_1$”的图示旨在指示代码单元 102(1)中类 T 的原始方法“$Main_0$”已由 CR1 对其原始版本作了更改。

已被标记用于后来实现的重写定制属性，诸如在调度计算机 204 上的属性（例如 $RCA2_T$）被传递给已更改代码单元 102(2)。已将被请求代码重写器（例如 CR1）应用到开发计算机202上代码单元102(1)的其它重写定制属性也可被传送到已更改代码单元 102(2)，如图 10 所示。然而，这并非是必须的，因为这些属性的目标都已达到。

参照图 11，调度计算机 204 可包括任意数量的应用程序代码单元，诸如从开发计算机 202 分配的代码单元 102(2)。如上所述，代码单元 102 包括在.NET 编译器上编译的，并被配置为在 CLR（公共语言运行时）1100 受控执行环境中执行的 IL 代码和元数据。另外，代码单元 102 还可包括诸如 RCA1 和 RCA2 的各种重写器定制属性，这些属性被配置为请求用来在不同时间更改代码单元 102 的代码重写器，诸如当代码单元 102 安装在调度计算机 204 上时。此外，代码单元 102 可包括用来证实代码单元 102 本身真实性的一个或多个数字签名，以及由重写器定制属性在代码单元 102 中请求的任何代码重写器 502。

当代码单元 102(2)装入调度计算机 204 时，体现为 CLR 1100 中装入程序 1102 的重写管理器 500 把代码单元 102(2)装入存储器并对其进行检查。检查之后，CLR 1100 的重写管理器 500 确定代码单元 102(2)的真实性和可信度。真实性基于代码单元 102(2)中发现的代码单元数字签名得到证实。以上简述了使用数学散列和公-私钥鉴别的这种证实的一个示例。代码单元 102(2)的可信度也可部分地基于 CLR 1100 的一些其它附加安全政策得到确定。例如，CLR 1100 安全政策可信任来自某些发行商或网站的代码单元，并也可拒绝来自某些其它发行商或网站的代码单元。一般而言，安全政策将信任先前已在调度计算机 204 上安装的系统代码单元。

在检查代码单元 202(2)之后，CLR 1100 的重写管理器 200 确定是否有在调度

计算机 204 上执行之前请求代码重写器 502(2)更改代码单元 102(2)的诸如 RCA2T 的任何潜在（未决的）重写定制属性（即被标记用来于执行前在调度计算机 204 上实现的重写定制属性）。除了请求重写器去更改代码单元 102(2)的潜在重写定制属性，CLR 1100 自身可启动它直到需要根据代码单元 102(2)运行的一个或多个系统代码重写器。系统重写器的一个示例是以下要更详细讨论的“应用兼容性重写器”。

当请求重写器时，CLR 1100 的重写管理器 500 装入来自一组驻留重写器 502(2)的第一个任意重写器，并启动在代码单元 102(2)上执行的相同类型的真实性和可信度检查。因而，在图 11 示例中，CLR 1100 的重写管理器 500 从重写器 502(2)装入 CR2，并证实 CR2 的真实性和可信度。

在成功证实代码单元 102(2)和重写器（例如 CR2）之后，CLR 210 的重写管理器 200 根据代码单元 102(2)运行重写器 CR2，导致已更改代码单元 102(2)。如果代码单元 102(2)中的其它重写定制属性也请求使用其它重写器，则 CLR 1100 的重写管理器 500 以相同方式根据代码单元 102(2)装入、证实、并运行其它重写器。在把所有被请求重写器应用到代码单元 102(2)之后，最终的已更改代码单元 102(3)被存储在调度计算机 204 上的重写高速缓存 406 上。（重写管理器 500 也可选择高速缓存一个或多个中间被重写组装件而不是最终的已重写组装件。）然后 CLR 1100 管理使用 JIT（运行时编译执行）编译器把已更改代码单元 102(3)编译成本机码 1104。本机码 1104 然后在处理器 1106 上运行。

在一可选实施例中，用代码重写器把代码单元 102(2)更改成 102(3)的过程，以及随后的把代码单元 102(3)译成本机码 1104 的翻译，可使用把受控代码预先编译成本机码（从而使其可以在安装后执行）的安装时编译（compile-on-install）编译器在把代码单元 102(2) 安装在调度计算机 204 上之后发生。

把最终的更改后代码单元 102(3)存储在调度计算机 204 上的重写高速缓存 406，使 CLR 1100 的重写管理器 500 为所有对执行代码单元 102(3)的随后调用领先进行代码重写过程。因而，当代码单元 102(3)将来出现时，CLR 210 的重写管理器 500 从重写高速缓存 406 访问代码单元 102(3)的被缓存最终重写版，并管理将代码单元 102(3)译成本机码 1104 的翻译，用来在处理器 1106 上执行。

高速缓存无效算法 1108 被配置为用来确定存储在重写高速缓存 406 上的代码单元是否保持有效。算法 1108 通常制定用来使重写缓存器 406 的部分或所有内容无效并将之丢弃的预定政策。例如，这种政策可包括当代码单元改变或给定代码重

写器的参数改变时使高速缓存无效。

图 12 示出了参照图 11 如上所述的相同代码重写器的一个示例。图 12 显示了调度计算机 204 上的“应用兼容性重写器”1200 以及便于应用兼容性重写器 1200 的实现和讨论的其它组件，包括 ACR 数据库 1202、操作系统 1204、重写高速缓存 406 以及代码单元 102。尽管对应用兼容性重写器 1200 的当前讨论并非以受控代码环境和 CLR 的上下文进行，可以连接这种讨论一般都适用于受控和非受控代码环境。

一般而言，应用兼容性重写器 1200 是说明与特定现有应用程序相一致的在平台/操作系统上所作变化的代码重写器。例如，图 12 的代码单元 102 指示其为自处理应用而操作系统 1204 指示其为升级版。常常，当操作系统升级时，某些应用程序需要打补丁或者进行修理以便继续在新操作系统环境中正常运行。

应用兼容性重写器 1200 维护各种应用程序与平台/操作系统升级版的兼容性。例如，当尝试装入用来执行的代码单元 102 时，应用兼容性重写器 1200 被配置为用来检查代码单元 102 并确定其是否与操作系统 1204 的升级版相兼容。应用兼容性重写器 1200 标识代码单元 102（例如某自处理应用）并查阅 ACR 数据库 1202 以查看为了与升级后的操作系统 1204 相兼容，是否需要重写代码单元 102 的任意部分。如果需要，应用兼容性重写器 1200 对代码单元 102 应用适当的修理。

示例性方法

用来实现代码重写的示例方法将主要参照图 13 的流程图进行描述。方法应用于以上参照图 1-12 所述的示例性实施例。尽管一种或多种方法通过流程图以及关联于流程图方框的文字来揭示，可以理解所述方法的元素不必按其显示的顺序执行，且其它可选顺序会导致类似的优势。此外，方法并非是排他的，可单独或结合另外的方法执行。所述方法的元素可用任何适当方式执行，包括通过例如 ASIC 上的硬件逻辑块或者定义在计算机可读介质上的计算机可读指令的执行。

在此所用的“计算机可读介质”可以是能包含、存储、通信、传播或传输由处理器适用或执行的任何装置。计算机可读介质可以是，而非限制，电子、磁性、光学、电磁、红外线、或半导体的系统、仪器、装置或传播介质。计算机可读介质的更多具体示例包括，在其它介质中的，有一根或多根接线的电子（电气）连接、便携式计算机盘（磁性）、随机存取存储器(RAM)（磁性）、只读存储器(ROM)（磁性）、可擦除可编程只读存储器（EPROM 或闪存）、光纤（光学）、可重写压缩盘（CD-RW）（光学）、以及便携式压缩盘只读存储器(CD-ROM)（光学）。

在方法 1300 的方框 1302，接收到代码单元。由作为单机模块或在计算机系统上执行的组件中模块（例如操作系统、安全政策、应用工具等等）的重写管理器接收代码单元。在方框 1304，重写管理器确定需要重写代码单元。可用各种方式作出该确定，包括例如读取代码单元中的告诉重写管理器需要重写代码单元的重写指令（例如标记、属性、字符串等等）。在方框 1306，重写管理器确定要重写代码单元的一个或多个重写器。通过访问标识重写器的重写列表，重写管理器确定要重写代码单元的重写器。重写列表可以是代码单元自身中的指令、或是计算机系统上的单机模块、或是计算机系统另一组件的一部分。可选地，重写列表可以是经诸如网络 206 的网络可访问的位于远程计算机装置的文件。

在确定要重写代码单元的重写器之后，重写管理器访问从一个或多个重写器组中标识的重写器，并把它们以及代码单元装入存储器（RAM）中。尽管一般说来重写器驻留于执行它们的计算机装置中，它们也可位于经诸如网络 206 的网络可访问的远程计算机装置上。在此情形中，重写管理器将访问远程装置上的重写器，并把它们以及代码单元装入 RAM 中。然后重写管理器证实代码单元和重写器是可靠的，如方框 1308 所示。这可通过鉴别关联于每个代码单元和重写器的数字签名来完成。在方框 1310，重写管理器将重写器按特定顺序排列，用来根据代码单元执行。也可能只有一个重写器要应用到代码单元中，这样就不会有重写器的排序了。根据重写列表确定一个以上重写器的顺序。在方框 1312，重写管理器根据代码单元以适当顺序一次一个地执行一个或多个重写器。每个重写器根据代码单元的执行，以唯一方式更改了代码单元。由一个或多个重写器进行重写的经过是经重写代码单元。在方框 1314，产生经重写代码单元的数字签名。如方框 1316 所示，数字签名关联到（例如：附加到）经重写代码单元。在方框 1318，经重写代码单元（且任选地，相关联数字签名）被存储在重写高速缓存中。

在方框 1320，接收一执行代码单元的指令。在方框 1322，重写管理器识别经重写代码单元存储在重写高速缓存中。因此，不必再次实现重写过程。反之，在方框 1326 直接从重写高速缓存装入经重写代码单元，并在方框 1328 执行之。

尽管一种或多种方法通过流程图以及关联于流程图方框的文字来揭示，可以理解所述方法的元素不必按其显示的顺序执行，且其它可选顺序会导致类似的优势。此外，方法并非是排他的，可单独或结合另外的方法执行。

示例性计算机

图 14 图示了适于实现所有参照图 1～13 如上所述的开发计算机 202、中间计

算机 302、和调度计算机 204 等的示例性计算环境。尽管图 14 中示有一具体配置，但开发计算机 202、中间计算机 302、和调度计算机 204 可在其它计算配置中实现。

计算环境 1400 具有计算机 1402 形式的通用计算系统。计算机 1402 的组件可包括，但不限于，一个或多个处理器或处理单元 1404、系统存储器 1406、和耦合各种系统组件，包括将处理器 1404 耦合到系统存储器 1406 的系统总线 1408。

系统总线 1408 代表若干类型的总线结构中任意的一种或多种，包括存储器总线或存储器控制器、外围总线、加速图形端口、和使用各种总线架构的任一种的处理器或本地总线。系统总线 1408 的一种示例将是外围部件互连（PCI）总线，也称为 Mezzanine 总线。

计算机 1402 具有各种计算机可读介质。这样的介质可以是计算机 1402 可访问的任何可用介质，且包括易失和非易失介质、可移动和不可移动介质。系统存储器 1406 包括诸如随机存取存储器（RAM） 1410 的易失存储器形式的、和/或诸如只读存储器（ROM）1412 的非易失存储器的计算机可读介质。包含有助于计算机 1402 如起动时在元件间传送信息的基本例程的基本输入/输出系统（BIOS）1414 存储在 ROM 1412 中。RAM 1410 包含可被处理单元 1404 立即访问和/或现时操作的数据和/或程序模块。

计算机 1402 还可包括其它可移动/不可移动、易失/非易失计算机存储介质。作为示例，图 14 图示了读取和写入不可移动、非易失磁性介质（未示出）的硬盘驱动器 1416，读取和写入可移动、非易失磁盘 1420（如“软盘”）的磁盘驱动器 1418，读取和写入可移动、非易失光盘 1424，如 CD-ROM、DVD-ROM、或其它光学介质的光盘驱动器 1422。硬盘驱动器 1416、磁盘驱动器 1418、和光盘驱动器 1422 分别由一个或多个数据介质接口 1426 连接到系统总线 1408。可选地，硬盘驱动器 1416、磁盘驱动器 1418、和光盘驱动器 1422 可经由 SCSI 接口（未示出）连接到系统总线 1408。

盘驱动器及其相关联的计算机可读介质为计算机 1402 提供计算机可读指令、数据结构、程序模块、和其它数据的非易失存储。尽管该示例中图示了硬盘 1416、可移动磁盘 1420、和可移动光盘 1424，但应明白其它类型的计算机可访问的可存储数据的计算机可读介质，如磁带或其它磁性存储设备、闪存卡、CD-ROM、数字化视频光盘（DVD）或其它光学存储设备、随机存取存储器（RAM）、只读存储器（ROM）、电子可擦可编程只读存储器（EEPROM）等等，也可被用来实现示例性计算系统和环境。

任何数量的程序模块，包括作为示例的操作系统 1426、一个或多个应用程序 1428、其它程序模块 1430、和程序数据 1432，可存储于硬盘 1416、磁盘 1420、光盘 1424、ROM 1412、和/或 RAM 1410。每一个这样的操作系统 1426、一个或多个应用程序 1428、其它程序模块 1430、和程序数据 1432（或其中某些组合）都可具有用户网络访问信息高速缓存方案的一个实施例。

计算机 1402 可具有各种标识为通讯介质的计算机/处理器可读介质。通讯介质包含调制数据信号形式的计算机可读指令、数据结构、程序模块、或其它数据，诸如载波或其它传送机制，且包含任何信息传递介质。术语“已调数据信号”意指用将信息编进信号的方法设置或改变其一个或多个特征的信号。作为示例，而非限制，通讯介质包括诸如有线网络或直线连接的有线介质，和诸如声学、射频、红外线和其它无线介质的无线介质。所有以上内容的组合也应包含在“计算机可读介质”范围之内。

用户可通过输入装置如键盘 1434 和定位装置 1436（如“鼠标”）向个人计算机系统 1402 输入指令和信息。其它输入装置 1438（未具体示出）可包括话筒、游戏杆、游戏垫、卫星接收器、扫描仪等等。这些和其它输入装置通常通过与系统总线 1408 耦合的输入/输出接口 1440 连接到处理单元 1404，但也可通过其它接口相连，如并行端口、游戏端口或通用串行总线（USB）。

监视器 1442 或其它类型显示装置也通过接口，如视频适配器 1444 和系统总线 1408 相连。除了显示器 1442，其它外围输出装置可包括可通过输入/输出接口 1440 与计算机 1402 连接的组件，如扬声器（未示出）和打印机 1446。

计算机 1402 可以在使用与一台或多台远程计算机，诸如远程计算设备 1448 的逻辑连接的网络化环境中运行。作为示例，远程计算设备 1448 可以是个人计算机、便携式计算机、服务器、路由器、网络计算机、同等装置或其它普通网络节点等等。远程计算设备 1448 图示为可包括在此所述与计算机系统 1402 相关的许多或全部部件的便携式计算机。

计算机 1402 和远程计算机 1448 间的逻辑连接包括局域网(LAN) 1450 和广域网(WAN) 1452。这样的网络化环境在办公室、企业范围计算机网络、企业内联网和因特网上是常见的。当在 LAN 网络化环境中实现时，计算机 1402 通过网络接口或适配器 1454 与本地网 1450 连接。当在 WAN 网络化环境中实现时，计算机 1402 包括调制解调器 1456 或其它用于在广域网 1452 中建立通讯的装置。可以内置或外置于计算机 1402 的调制解调器 1456，通过输入/输出接口 1440 或其它适当机制连

接到系统总线 1408。可以理解的是，所示网络连接是示例性的，且其它用于在计算机 1402 和 1448 间建立通讯连接的技术也可以使用。

在诸如计算环境 1400 图示的网络化环境中，所描述的计算机 1402 相关程序模块或其中部分模块，可存储在远程存储器存储设备中。作为示例，远程应用程序 1458 位于远程计算机 1448 的存储设备上。为阐明本发明，诸如操作系统的应用程序和其它可执行程序组件在此图示为离散块，尽管可以理解这样的程序和组件在各种时刻会驻留于计算机系统 1402 的不同存储组件上，并由该计算机的数据处理器执行。

结论

尽管本发明已用结构化特征和/或方法论行动的专用语言作了说明，但可以理解的是在所附权利要求书中定义的本发明无须受限于所述特定特征或行动。相反，具体特征和行动是以实现本发明的示例性形式被揭示的。

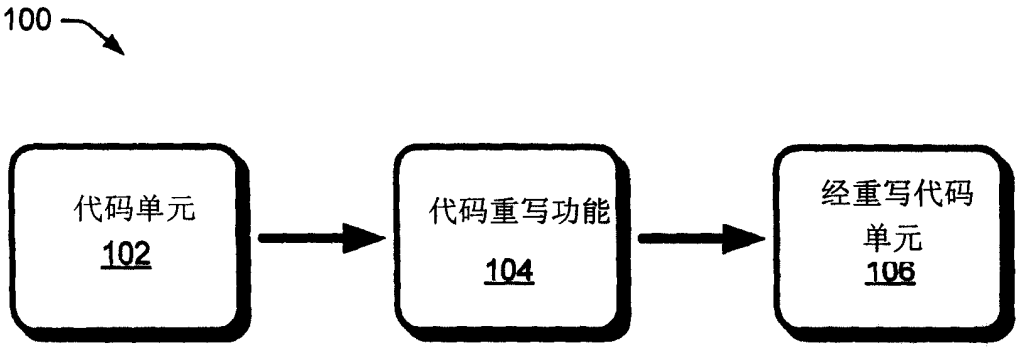

100
代码单元
102
代码重写功能
104
经重写代码
单元
106

图 1

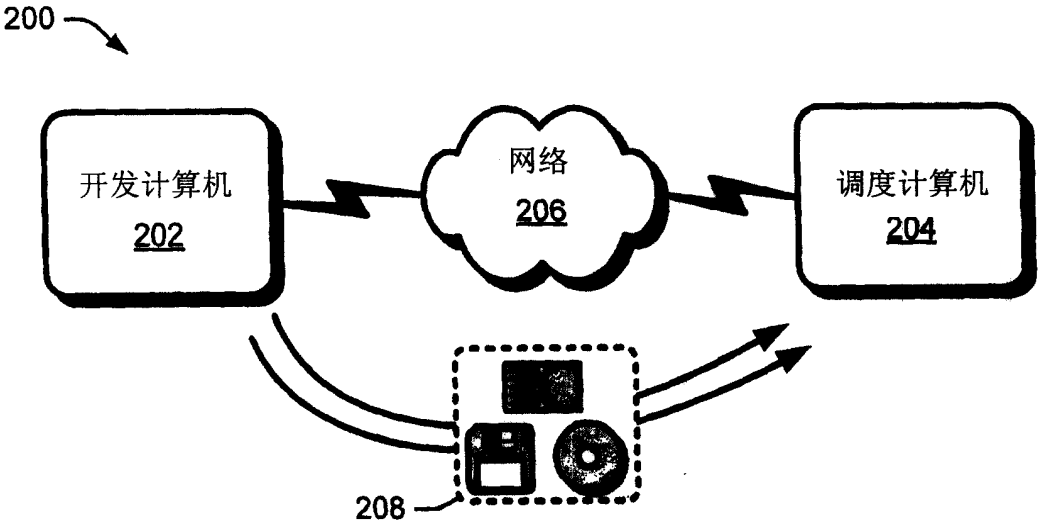

200
开发计算机
202
网络
206
调度计算机
204
208

图 2

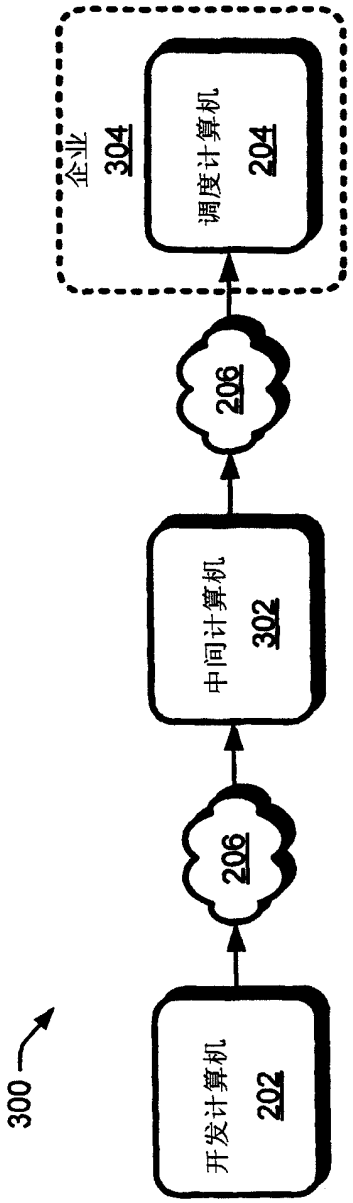


图 3

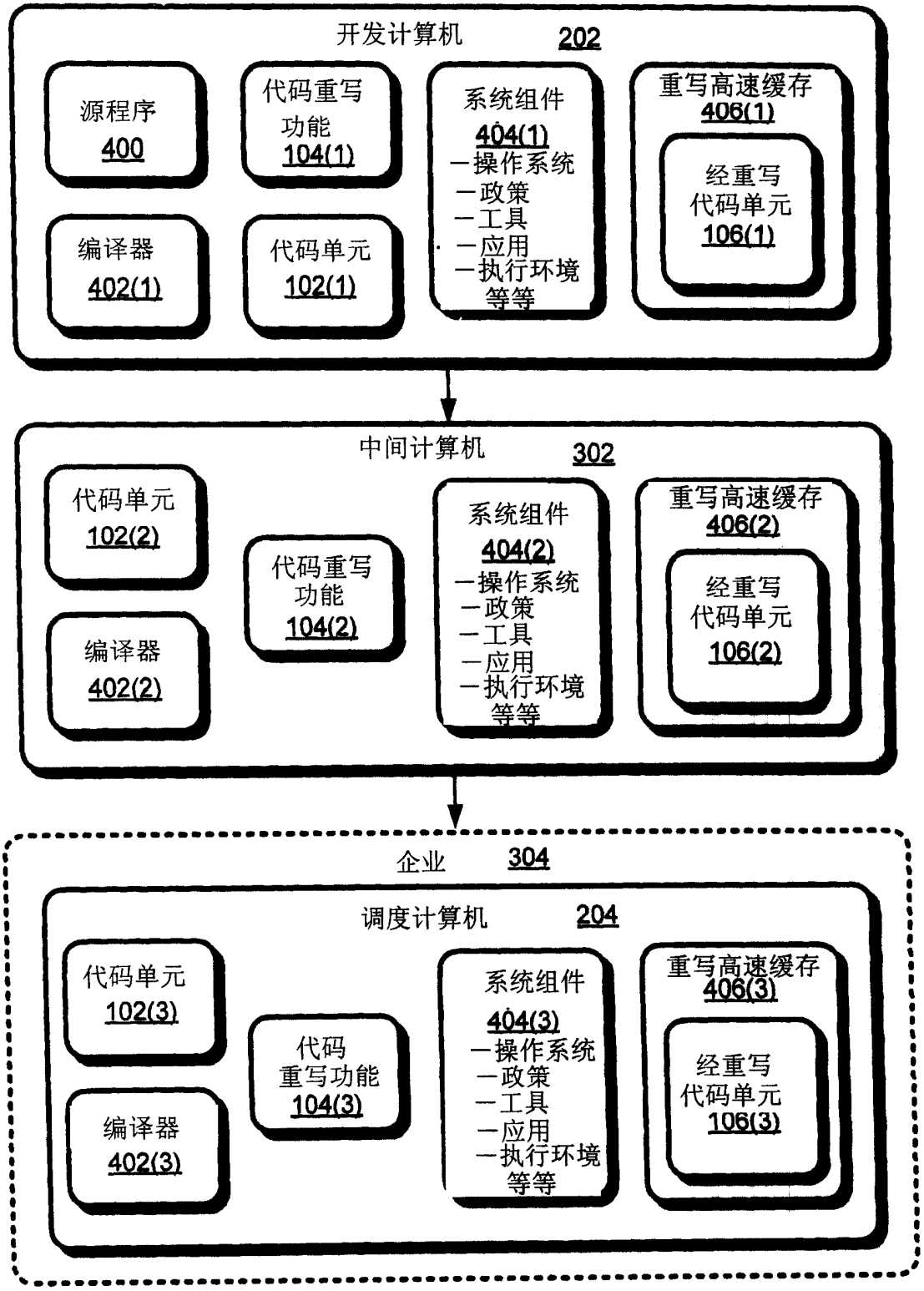

开发计算机
202
源程序
400
代码重写
功能
104(1)
系统组件
404(1)
—操作系统
—政策
—工具
—应用
—执行环境
等等
重写高速缓存
406(1)
经重写
代码单元
106(1)
编译器
402(1)
代码单元
102(1)
中间计算机
302
代码单元
102(2)
代码重写
功能
104(2)
系统组件
404(2)
—操作系统
—政策
—工具
—应用
—执行环境
等等
重写高速缓存
406(2)
经重写
代码单元
106(2)
编译器
402(2)
企业
304
调度计算机
204
代码单元
102(3)
代码
重写功能
104(3)
系统组件
404(3)
—操作系统
—政策
—工具
—应用
—执行环境
等等
重写高速缓存
406(3)
经重写
代码单元
106(3)
编译器
402(3)

图 4

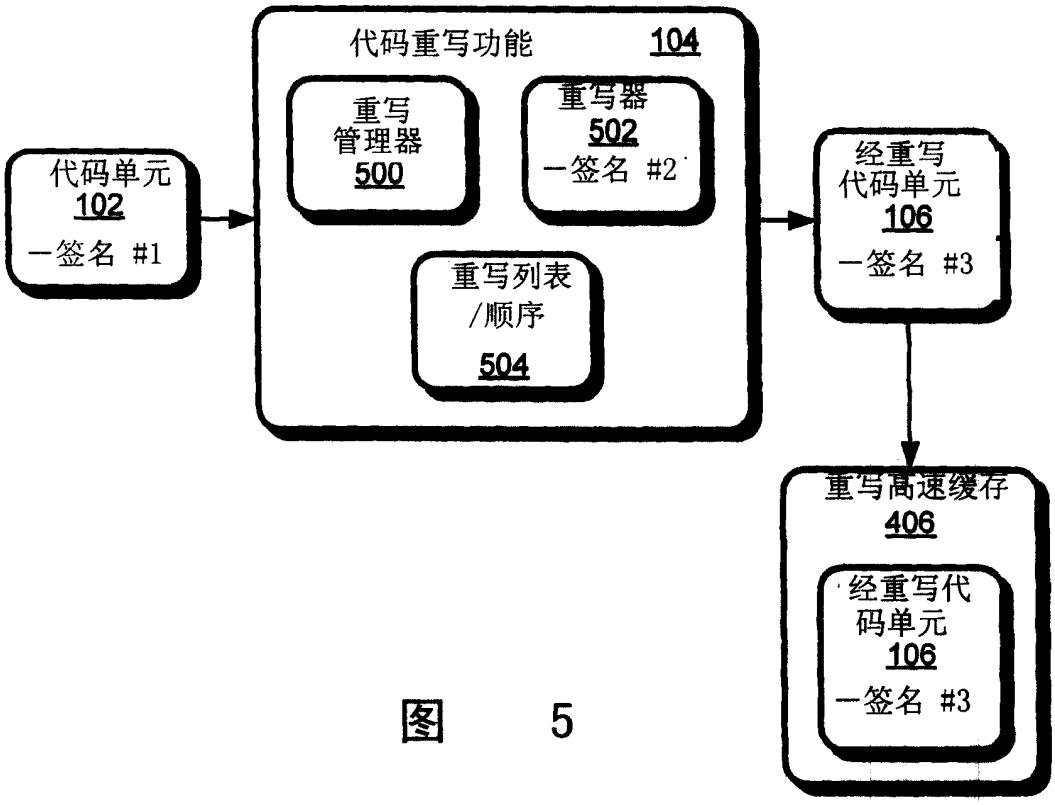

代码重写功能
104
重写
管理器
500
重写器
502
一签名 #2
代码单元
102
一签名 #1
重写列表
/顺序
504
经重写
代码单元
106
一签名 #3
重写高速缓存
406
经重写代
码单元
106
一签名 #3

图 5

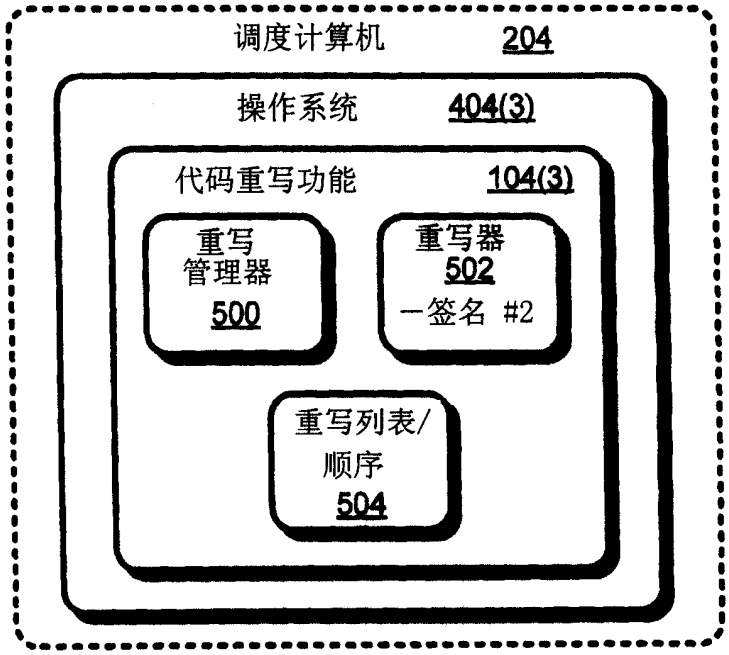

调度计算机
204
操作系统
404(3)
代码重写功能
104(3)
重写
管理器
500
重写器
502
一签名 #2
重写列表/
顺序
504

图 6

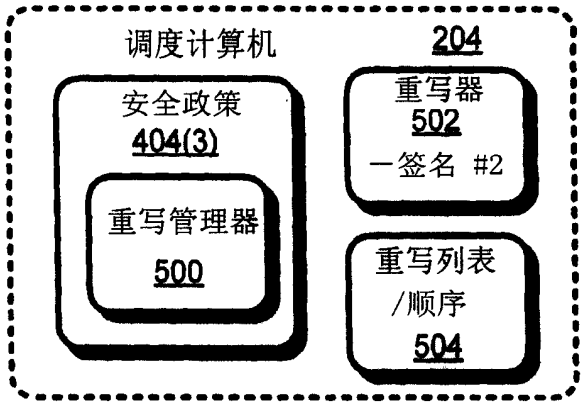

调度计算机
204
安全政策
404(3)
重写管理器
500
重写器
502
一签名 #2
重写列表
/顺序
504

图 7

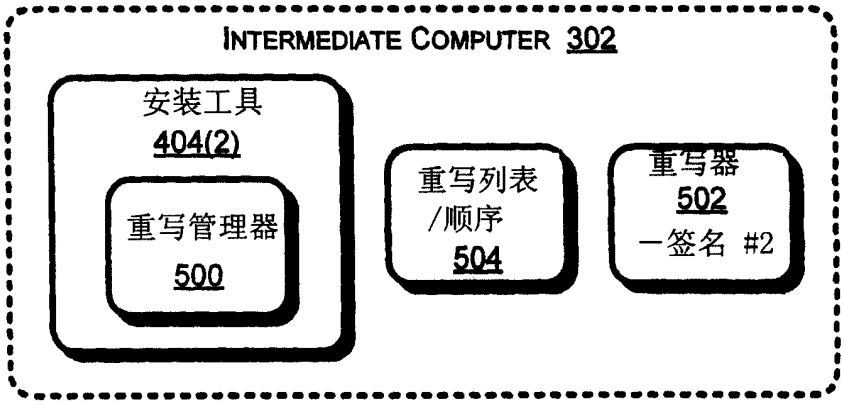

INTERMEDIATE COMPUTER 302
安装工具
404(2)
重写管理器
500
重写列表
/顺序
504
重写器
502
一签名 #2

图 8

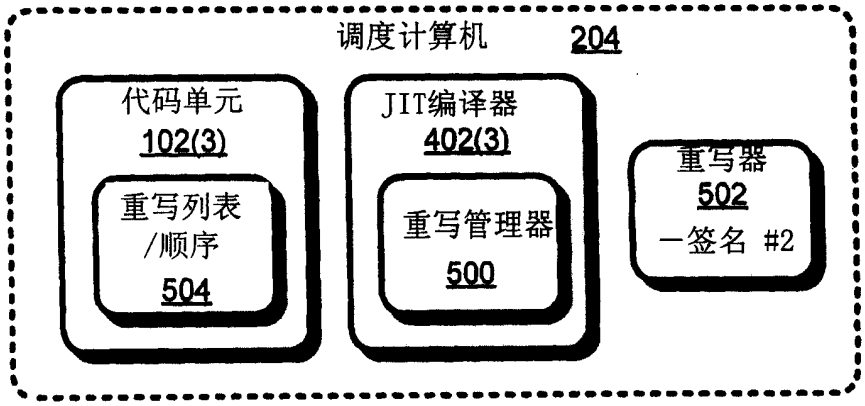

调度计算机
204
代码单元
102(3)
重写列表
/顺序
504
JIT编译器
402(3)
重写管理器
500
重写器
502
一签名 #2

图 9

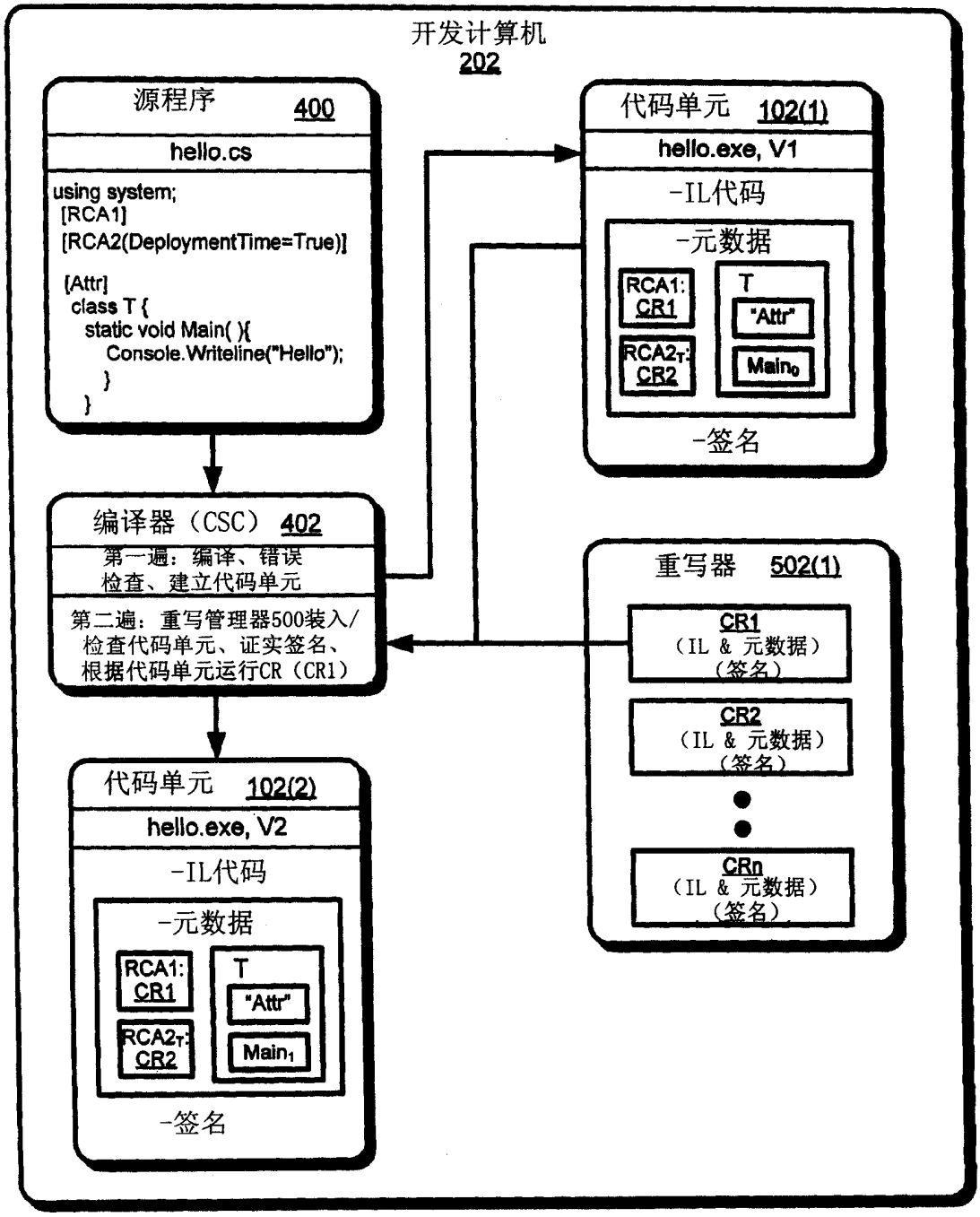

开发计算机
202
源程序 400
hello.cs
using system;
[RCA1]
[RCA2(DeploymentTime=True)]
[Attr]
class T {
static void Main( ){
Console.Writeline("Hello");
}
}
编译器（CSC） 402
第一遍：编译、错误检查、建立代码单元
第二遍：重写管理器500装入/检查代码单元、证实签名、根据代码单元运行CR（CR1）
代码单元 102(1)
hello.exe, V1
-IL代码
-元数据
RCA1: CR1
RCA2T: CR2
T
"Attr"
Main0
-签名
重写器 502(1)
CR1
（IL & 元数据）
（签名）
CR2
（IL & 元数据）
（签名）
CRn
（IL & 元数据）
（签名）
代码单元 102(2)
hello.exe, V2
-IL代码
-元数据
RCA1: CR1
RCA2T: CR2
T
"Attr"
Main1
-签名

图 10

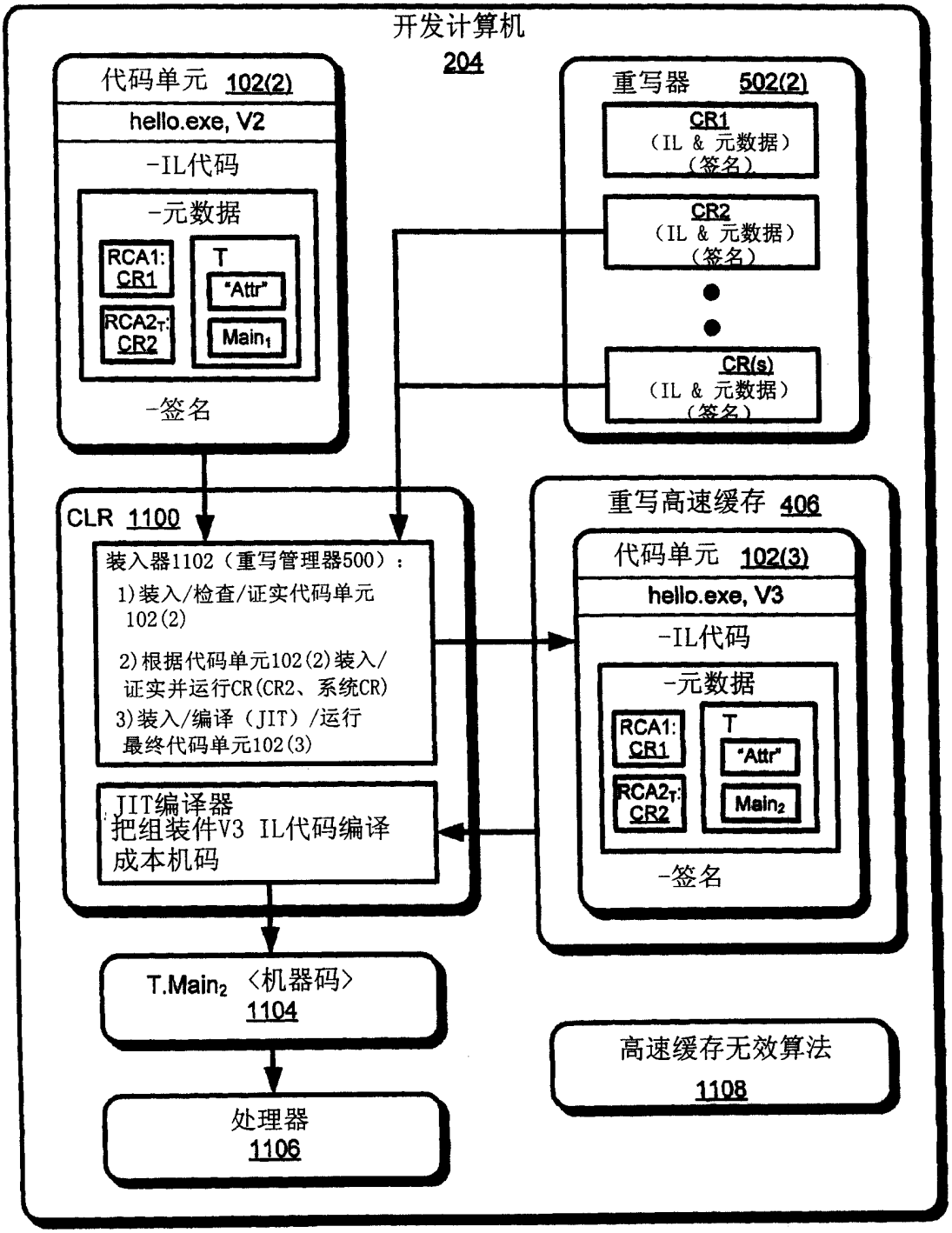

开发计算机
204
代码单元 102(2)
hello.exe, V2
-IL代码
-元数据
RCA1: CR1
T
"Attr"
RCA2T: CR2
Main1
-签名
重写器 502(2)
CR1
(IL & 元数据)
(签名)
CR2
(IL & 元数据)
(签名)
CR(s)
(IL & 元数据)
(签名)
CLR 1100
装入器1102(重写管理器500):
1)装入/检查/证实代码单元102(2)
2)根据代码单元102(2)装入/证实并运行CR(CR2、系统CR)
3)装入/编译(JIT)/运行最终代码单元102(3)
JIT编译器
把组装件V3 IL代码编译成本机码
重写高速缓存 406
代码单元 102(3)
hello.exe, V3
-IL代码
-元数据
RCA1: CR1
T
"Attr"
RCA2T: CR2
Main2
-签名
T.Main2 〈机器码〉
1104
处理器
1106
高速缓存无效算法
1108

图 11

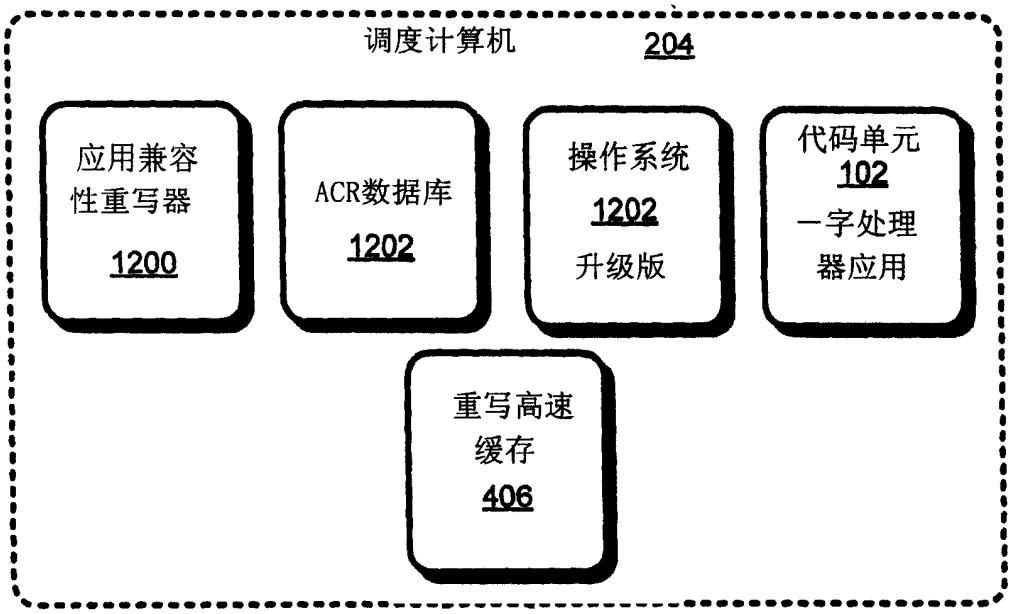

调度计算机
204
应用兼容性重写器
1200
ACR数据库
1202
操作系统
1202
升级版
代码单元
102
一字处理器应用
重写高速缓存
406

图 12

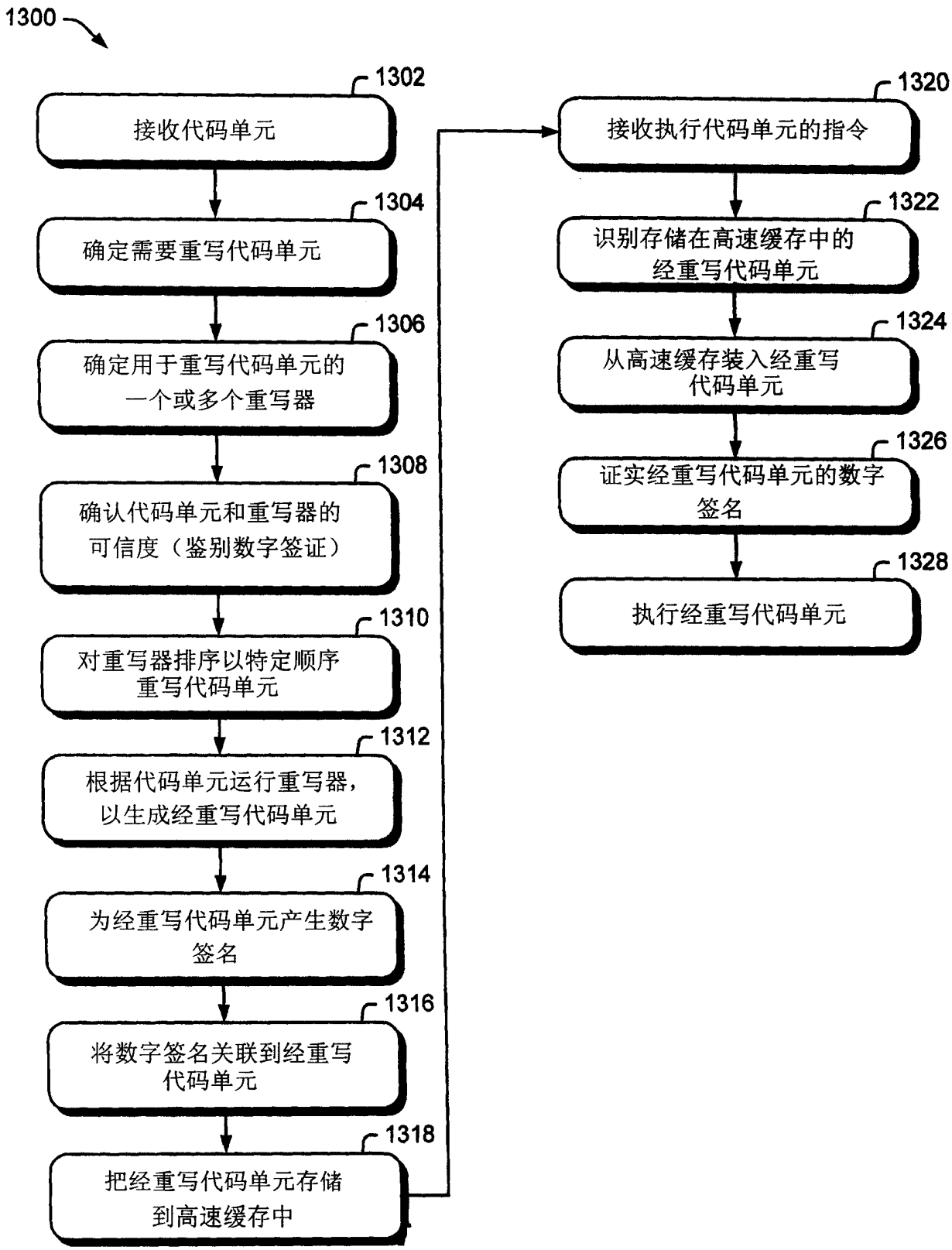

1300
1302
接收代码单元
1304
确定需要重写代码单元
1306
确定用于重写代码单元的一个或多个重写器
1308
确认代码单元和重写器的可信度（鉴别数字签证）
1310
对重写器排序以特定顺序重写代码单元
1312
根据代码单元运行重写器，以生成经重写代码单元
1314
为经重写代码单元产生数字签名
1316
将数字签名关联到经重写代码单元
1318
把经重写代码单元存储到高速缓存中
1320
接收执行代码单元的指令
1322
识别存储在高速缓存中的经重写代码单元
1324
从高速缓存装入经重写代码单元
1326
证实经重写代码单元的数字签名
1328
执行经重写代码单元

图 13

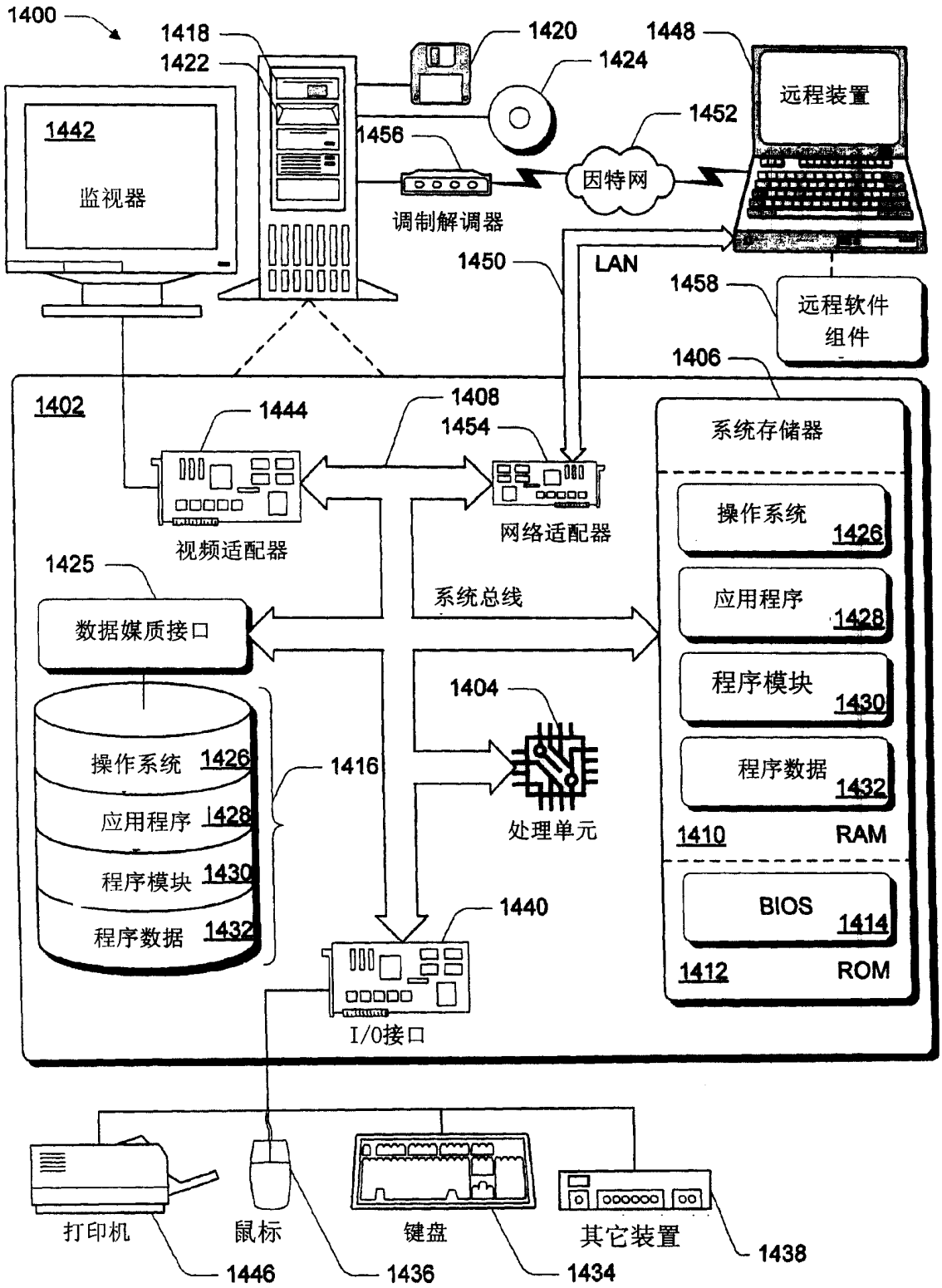

1400
1418
1422
1420
1424
1448
远程装置
1452
1442
监视器
1456
因特网
调制解调器
1450
LAN
1458
远程软件组件
1406
1402
1444
1408
1454
系统存储器
操作系统 1426
应用程序 1428
程序模块 1430
程序数据 1432
1410 RAM
BIOS 1414
1412 ROM
网络适配器
视频适配器
1425
数据媒质接口
系统总线
1404
处理单元
1416
操作系统 1426
应用程序 1428
程序模块 1430
程序数据 1432
1440
I/O接口
打印机
鼠标
键盘
其它装置
1438
1446
1436
1434

图 14