(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 101329636 B
(45)授权公告日 2014.01.29

(21)申请号 200810131669.9
(22)申请日 2005.09.23
(30)优先权数据
10/711,737 2004.09.30 US
10/711,736 2004.09.30 US
10/711,735 2004.09.30 US
10/711,734 2004.09.30 US
10/711,733 2004.09.30 US
10/711,732 2004.09.30 US
10/956,723 2004.10.01 US
11/231,284 2005.09.19 US
11/231,316 2005.09.19 US
11/231,317 2005.09.19 US
11/231,315 2005.09.19 US
11/231,370 2005.09.19 US
(62)分案原申请数据
200580040872.X 2005.09.23
(73)专利权人 茨特里克斯系统公司
地址 美国佛罗里达州
(72)发明人 L·G·拉波尔茨法尔维
A·罗伊乔德里 A·G·博尔茨基
H·C·钦 R·J·马扎费里
(74)专利代理机构 中国专利代理(香港)有限公司 72001
代理人 蒋骏 王小衡
(51)Int.Cl.
G06F 9/445(2006.01)
G06F 17/30(2006.01)
(56)对比文件
US 2003074393 A1,2003.04.17,
US 6047312 A,2000.04.04,
CN 1421773 A,2003.06.04,
审查员 冯婷霆

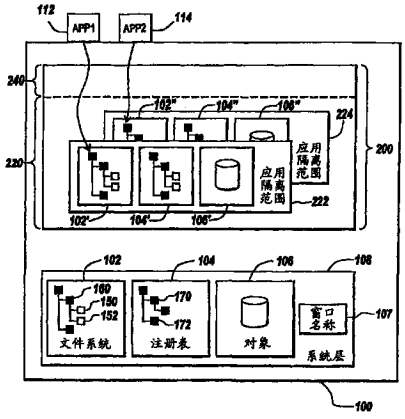
权利要求书2页 说明书51页 附图23页

(54)发明名称

虚拟化窗口信息的方法和设备

(57)摘要

一种用于对窗口访问虚拟化的方法和设备包括挂钩机制、窗口名称虚拟化引擎、和操作系统接口。从在用户帐号的上下文中执行的进程接收与窗口有关的请求,该请求包括虚拟窗口名称。利用范围特定标识符来确定窗口的真实名称。向操作系统发出包括所确定真实窗口名称的请求。窗口句柄与所确定虚拟窗口名称关联。



CN 101329636 B

1. 一种用于在单一的计算机上将文件的文件类型与一个或多个程序关联的方法，所述方法包括步骤：

(a) 接收在配置存储器中存储文件类型关联信息的请求；

(b) 根据该请求来确定要与配置存储器中的文件类型相关联的应用程序；和

(c) 文件类型与选择器工具的关联关系被写入到配置存储器，

其中所述选择器工具提供用户接口，该用户接口列出一个或多个为访问与文件类型关联的文件而调用的应用程序。

2. 权利要求 1 的方法，其中步骤 (a) 还包括通过用户模式挂钩机制、内核模式挂钩机制、文件系统过滤器驱动程序、和微型驱动程序之一来截取请求。

3. 权利要求 1 的方法，其中请求包括更新在配置存储器中的文件类型关联关系的条目。

4. 权利要求 1 的方法，其中请求包括创建在配置存储器中的文件类型关联关系的新条目。

5. 权利要求 1 的方法，还包括步骤：

提供应用程序与在选择器工具配置存储器中的文件类型的关联关系。

6. 权利要求 5 的方法，其中配置存储器包括选择器工具配置存储器。

7. 权利要求 1 的方法，其中配置存储器包括注册表数据库。

8. 权利要求 1 的方法，还包括步骤：

选择与文件类型关联的文件以调用应用程序，并响应于选择该文件来调用选择器工具。

9. 权利要求 1 的方法，还包括步骤：

通过选择器工具显示与文件类型关联的一个或多个应用程序的列表。

10. 权利要求 1 的方法，还包括步骤：

通过选择器工具在系统范围、应用隔离范围、和用户隔离范围之一中调用所选择的应用程序。

11. 权利要求 1 的方法，其中应用程序、文件、配置存储器和选择器工具之一在隔离环境内与系统范围、应用隔离范围、和用户隔离范围之一相关联。

12. 权利要求 1 的方法，其中选择器工具是与文件类型关联的缺省应用程序。

13. 权利要求 1 的方法，还包括在配置存储器中将第二文件类型与选择器工具关联。

14. 一种用于在单一的计算机上调用与文件类型关联的应用程序的方法，该方法包括步骤：

(a) 选择文件以便调用应用程序，该文件与文件类型关联；

(b) 响应于选择文件，从配置存储器获得与文件类型关联的对选择器工具的引用，配置存储器包括文件类型关联信息；以及

(c) 响应于选择文件来调用选择器工具，选择器工具显示一个或多个访问所选文件的应用程序的列表。

15. 权利要求 14 的方法，其中步骤 (a) 还包括通过点击文件一次或多次来选择文件，以调用应用程序。

16. 权利要求 14 的方法，其中步骤 (b) 还包括从选择器工具配置存储器获得与所选文

件的文件类型关联的一个或多个应用程序的列表。

17. 权利要求 16 的方法，其中步骤 (c) 还包括选择器工具显示与从选择器工具配置存储器获得的文件类型关联的一个或多个应用程序中的至少一个。

18. 权利要求 16 的方法，其中配置存储器包括选择器工具配置存储器。

19. 权利要求 14 的方法，其中配置存储器包括注册表数据库。

20. 权利要求 14 的方法，还包括步骤：

调用从由选择器工具显示的列表选择的应用程序。

21. 权利要求 20 的方法，还包括：在系统范围、应用隔离范围、和用户隔离范围之一中调用所选择的应用程序。

22. 权利要求 14 的方法，其中应用程序、文件、配置存储器和选择器工具之一在隔离环境内与系统范围、应用隔离范围、和用户隔离范围之一相关联。

23. 权利要求 14 的方法，其中选择器工具是与配置存储器中的文件类型相关联的缺省应用程序。

24. 一种用于在单一的计算机上调用与文件类型关联的程序的系统，该系统包括：

选择器工具，被配置成提供为访问与文件类型相关联的文件而调用的一个或多个应用程序的选择，其中所述选择器工具包括用户接口，用于显示所述一个或多个应用程序的列表；以及

重定向器，获得在配置存储器中存储文件类型关联信息的请求，根据该请求来确定要与配置存储器中的应用程序关联的文件类型；以及向配置存储器写入以便将文件类型与选择器工具相关联。

25. 权利要求 24 的系统，其中重定向器包括截取请求的下列内容之一：用户模式挂钩机制、内核模式挂钩机制、文件系统过滤器驱动程序、和微型驱动程序。

26. 权利要求 24 的系统，包括选择器工具配置存储器，用于存储将文件类型与请求的应用程序关联的条目。

27. 权利要求 26 的系统，其中重定向器启动将条目写到选择器工具配置存储器的动作。

28. 权利要求 26 的系统，其中配置存储器包括选择器工具配置存储器。

29. 权利要求 24 的系统，其中配置存储器包括注册表数据库。

30. 权利要求 26 的系统，其中选择器工具显示与从选择器工具配置存储器获得的文件类型关联的至少一个应用程序。

31. 权利要求 24 的系统，其中选择器工具调用从该列表选择的应用程序。

虚拟化窗口信息的方法和设备

技术领域

[0001] 本发明涉及管理计算机的软件应用的执行，并尤其涉及用于降低不同应用程序之间以及由同一计算机系统执行的同一应用的各个用户之间的兼容性和集群度问题的方法和设备。

背景技术

[0002] 计算机软件应用程序，在执行和安装期间，利用各种由计算机的操作系统提供的本地资源。在图 1A 中描述传统的单用户计算机。如图 1A 所示，由操作系统 100 提供的本地资源可包括文件系统 102、注册表数据库 104 和对象 106。文件系统 102 为应用程序提供打开、创建、读取、复制、修改和删除数据文件 150，152 的机制。数据文件 150、152 可在目录 160、162 的逻辑层次中被分组在一起。注册表数据库 104 存储与物理附着到计算机的硬件、已经选择了哪些系统选项、如何安装计算机存储器、应用特定数据的各种项、和什么应用程序在操作系统 100 启动时应当出现有关的信息。如图 1A 所示的，注册表数据库 104 通常组织在“键”170、172 的逻辑层次中，所述键是注册表值的容器。操作系统 100 还可提供多个通信和同步对象 106，包括信号灯、段、互斥体、定时器、变异体和管道。通过操作系统 100 而可用的文件系统 102、注册表数据库 104、对象 106 和任何其它本地资源一起在本文献中被称为“系统层”108。由系统层 108 提供的资源可对于任何应用或系统程序 112、114 可用。

[0003] 但是，当试图执行或安装两个不兼容的应用程序 112、114 时出现问题。如图 1A 所示，两个应用程序 APP1 112 和 APP2 114 在操作系统 100 “之上”执行，即是，应用程序利用由操作系统提供的函数来访问本地资源。当应用程序在执行期间或在安装过程期间以互斥的方式利用本地资源 102、104、106 时被说成是彼此不兼容的。APP1 112 可要求或试图安装由路径名称 c:\windows\system32\msvcrt.dll 定位的文件，而 APP2 114 可要求或试图安装由相同路径名称定位的第二个不同的文件。在该情况下，APP1 112 和 APP2 114 不能在同一计算机上执行并且被说成是彼此不兼容的。对于其它本地资源可遇到类似问题。这最多给要求在相同操作系统 100 环境中一起安装或执行 APP1 112 和 APP2 114 的计算机用户带来不便。

[0004] 图 1B 描述的多用户计算机系统，其支持代表若干用户的应用程序 112、114、112’、114’并发执行。如图 1B 所示，APP1 112 的第一实例和 APP2 114 的第一实例在第一用户会话 110 的上下文中执行，APP1112’的第二实例在第二用户会话 120 的上下文中执行，并且 APP2 114’的第二实例在第三用户会话 130 的上下文中执行。在该环境下，如果 APP1 112、112’两者的实例和 APP2 114、114’两者的实例利用本地资源 102、104、106 就如同只有单个用户执行应用，则出现问题。例如 APP1 112 可在注册表键 170 存储应用特定数据。当在第一用户上下文 110 中执行的 APP1 112 的第一实例和在第二用户上下文 120 中执行的 APP1 112’的第二实例都试图存储配置数据到相同的注册表键 170 时，将为用户之一存储错误的配置数据。类似的问题可能对于其它本地资源出现，比如窗口名称和窗口类信息。

[0005] 本发明解决了这些应用程序兼容性和集群度问题并且还提供将多个应用与文件

类型关联的能力。

发明内容

[0006] 本发明允许在单个计算机上安装和执行彼此不兼容的应用程序以及同一应用程序的不兼容版本。此外，其允许在多用户计算机上安装和执行为单用户计算机创建的或在没有考虑当在多用户计算机上执行时会出现问题的情况下创建的程序。所描述的方法和系统可应用于单用户计算环境，其包括多用户一个接一个地使用单个计算机的环境，和多用户计算环境，其中多个用户并发使用单个计算机。本发明虚拟化用户和应用对本地资源的访问，比如文件系统、注册表数据库、系统对象、窗口类和窗口名称，而不用修改应用程序或底层的操作系统。此外，虚拟化的本地资源可用本地格式来存储（即，虚拟化的文件存储在文件系统中，虚拟化的注册表条目存储在注册表数据库中，等），从而允许利用标准工具和技术来完成虚拟化资源的查看和管理。

[0007] 在一个方面，本发明涉及用于虚拟化对窗口访问的方法。从在用户帐号的上下文中执行的进程接收与窗口有关的请求，该请求包括虚拟窗口名称。利用范围特定标识符来确定窗口的真实名称。向操作系统发出包括所确定真实窗口名称的请求。窗口句柄与所确定虚拟窗口名称关联。

[0008] 在一个实施例中，从在用户帐号的上下文中执行的进程截取与窗口有关的请求。在另一个实施例中，与包括在请求中的虚拟窗口名称关联的规则被确定并用于确定该窗口的真实名称。在一些实施例中，虚拟窗口名称存储在与窗口句柄关联的映射表中。

[0009] 在另一方面，本发明涉及用于虚拟化对窗口访问的方法。接收识别虚拟窗口名和虚拟窗口类标识符之一的请求，该请求从在用户帐号的上下文中执行的进程接收且包括窗口句柄。确定窗口句柄与所请求的虚拟窗口名和虚拟窗口类标识符之一相关联。所确定的窗口信息被返回到请求进程。在一个实施例中，根据映射表来确定与所请求的虚拟窗口名和虚拟窗口类标识符之一相关联的窗口句柄。

[0010] 在另一方面，本发明涉及用于虚拟化对窗口访问的设备。挂钩机制从在用户帐号的上下文中执行的进程接收与窗口有关的请求，该请求包括虚拟窗口名和虚拟窗口类标识符之一。窗口名称虚拟化引擎利用在请求中接收的虚拟窗口名和虚拟窗口类标识符之一和范围特定标识符来形成窗口的真实名称和真实窗口类标识符之一。操作系统接口发出与窗口有关的请求，该请求包括窗口的所形成的真实名称和所形成的真实窗口类标识符之一。

[0011] 在一个实施例中，挂钩机制截取从由下列项组成的组中选择的请求：寻找窗口、创建窗口、枚举窗口、销毁窗口、设置窗口名称、获取窗口名称、获取与窗口关联的窗口类标识符、注册窗口类、获取与窗口类有关的信息和注销窗口类。在另一个实施例中，映射表存储窗口句柄与虚拟窗口名和虚拟窗口类标识符之一之间的关联关系。

[0012] 在另一方面，本发明涉及用于虚拟化对窗口访问的方法。从请求者截取涂色窗口标题栏的请求，标题栏包括窗口名称，该请求包括窗口句柄。确定与窗口句柄关联的虚拟窗口名称。利用虚拟窗口名称涂色窗口的标题栏。向请求者指明标题栏已经被涂色。

[0013] 在又一个方面，本发明涉及用于虚拟化对窗口访问的方法。从在用户帐号的上下文中执行的进程接收与窗口类有关的请求，该请求包括虚拟窗口类标识符。利用范围特定的标识符来确定真实窗口类标识符。请求被发给操作系统，该请求包括所确定的真实窗口

类标识符。

[0014] 在一个实施例中，与包括在请求中的虚拟窗口类标识符关联的规则被识别并用于响应于所确定的规则来确定真实窗口类标识符。在另一个实施例中，虚拟窗口类标识符存储在与窗口句柄关联的映射表中。在另一个实施例中，虚拟窗口类标识符替换响应中所确定的真实窗口类标识符。

[0015] 在一个方面，本发明涉及用于将文件的文件类型与一个或多个程序关联的方法。接收在配置存储器中存储文件类型关联信息的请求。根据该请求来确定要与配置存储器中的文件类型相关联的应用程序。文件类型与选择器工具的关联关系被写入到配置存储器。

[0016] 在另一方面，本发明涉及用于调用与文件类型关联的应用程序的方法。选择文件以便调用应用程序。该文件与文件类型关联。响应于选择文件，从配置存储器获得与文件类型关联的对选择器工具的引用。配置存储器包括文件类型关联信息。响应于选择文件来调用选择器工具，并且选择器工具显示一个或多个访问所选文件的应用程序的列表。

[0017] 在另一个方面，本发明涉及用于调用与文件类型关联的程序的系统。重定向器获得在配置存储器中存储文件类型关联信息的请求并根据该请求来确定要与配置存储器中的应用程序关联的文件类型。选择器工具配置为提供一个或多个为访问与文件类型相关联的文件而调用的应用程序的选择。重定向器向配置存储器写入以便将文件类型与选择器工具相关联。

附图说明

[0018] 在所附权利要求中特别的指出了本发明。参考下面的描述并结合附图，上述本发明的优点以及本发明的进一步优点可更好地被理解，在附图中：

[0019] 图 1A 是现有操作系统环境的框图，支持执行代表一个用户的两个应用程序；

[0020] 图 1B 是现有操作系统环境的框图，支持并发执行代表若干用户的多个应用程序；

[0021] 图 2A 是已经降低了应用程序兼容性和集群度问题的计算机系统的一个实施例的框图；

[0022] 图 2B 是已经降低了应用程序兼容性和集群度问题的计算机系统的一个实施例的示意图；

[0023] 图 2C 是示出为将进程与隔离范围关联所采取的步骤的一个实施例的流程图；

[0024] 图 3A 是示出为虚拟化对计算机系统中本地资源的访问所采取的步骤的一个实施例的流程图，

[0025] 图 3B 是示出为识别执行模式中替换实例所采取的步骤的一个实施例的流程图；

[0026] 图 3C 是描述在安装模式中当接收打开本地资源的请求时识别真实资源所采取的步骤的一个实施例的流程图，该请求指明资源在有修改它的意图时被打开；

[0027] 图 3D 是描述在安装模式中当接收创建虚拟资源的请求时识别真实资源所采取的步骤的一个实施例的流程图；

[0028] 图 4 是描述为在所描述的虚拟化环境中打开文件系统中的条目所采取的步骤的一个实施例的流程图；

[0029] 图 5 是描述为在所描述的虚拟化环境中从文件系统删除条目所采取的步骤的一个实施例的流程图；

[0030] 图 6 是描述为在所描述的虚拟化环境中枚举文件系统中的条目所采取的步骤的一个实施例的流程图；

[0031] 图 7 是描述为在所描述的虚拟化环境中创建在文件系统中的条目所采取的步骤的一个实施例的流程图；

[0032] 图 7A 是描述为在创建新文件之后分配唯一的短文件名所采取的步骤的一个实施例的流程图；

[0033] 图 8 是描述为在所描述的虚拟化环境中打开注册表键所采取的步骤的一个实施例的流程图；

[0034] 图 9 是描述为在所描述的虚拟化环境中删除注册表键所采取的步骤的一个实施例的流程图；

[0035] 图 10 是描述为在所描述的虚拟化环境中枚举注册表数据库中键的子键所采取的步骤的一个实施例的流程图；

[0036] 图 11 是描述为在所描述的虚拟化环境中创建注册表键所采取的步骤的一个实施例的流程图；

[0037] 图 12 是描述为虚拟化对命名对象进行访问所采取的步骤的一个实施例的流程图；

[0038] 图 13 是描述为在所描述的环境中虚拟化窗口名称和窗口类所采取的步骤的一个实施例的流程图；

[0039] 图 13A 是描述为确定真实窗口名称和窗口类名称所采取的步骤的一个实施例的流程图；

[0040] 图 14 是描述为在所描述的虚拟化环境中调用进程外 COM 服务器所采取的步骤的一个实施例的流程图；

[0041] 图 15 是描述为利用文件类型关联关系来虚拟化应用调用所采取的步骤的一个实施例的流程图；并且

[0042] 图 16 是描述为将进程从源隔离范围移动到目标隔离范围所采取的步骤的一个实施例的流程图。

[0043] 索引

[0056] 4.1.3 文件系统枚举操作

[0057] 4.1.4 文件系统创建操作

[0058] 4.1.5 短文件名管理

[0059] 4.2 注册表虚拟化

[0060] 4.2.1 注册表打开操作

[0061] 4.2.2 注册表删除操作

[0062] 4.2.3 注册表枚举操作

[0063] 4.2.4 注册表创建操作

[0064] 4.3 命名对象虚拟化

[0065] 4.4 窗口名称虚拟化

[0066] 4.5 进程外 COM 服务器虚拟化

[0067] 4.6 虚拟化的文件类型关联关系

[0068] 4.7 隔离环境之间进程的动态移动

具体实施方式

[0069] 1.0 隔离环境的总体概述

[0070] 1.1 应用隔离

[0071] 现在参考图 2A，示出了在操作系统 100 的控制下运行的计算机的一个实施例，该实施例已经降低了应用程序兼容性和集群度的问题。操作系统 100 经由其系统层 108 使各种本地资源对于应用程序 112、114 可用。由系统层 108 体现的资源视图将被称为“系统范围”。为了避免应用程序 112、114 对本地资源 102、104、106、107 的访问冲突，提供了隔离环境 200。如图 2A 所示，隔离环境 200 包括应用隔离层 220 和用户隔离层 240。概念上，隔离环境 200 经由应用隔离层 220 为应用程序 112、114 提供本地资源比如文件系统 102、注册表 104、对象 106 和窗口名 107 的唯一视图。每个隔离层修改提供给应用程序的本地资源的视图。由层提供的本地资源的修改视图被称为层的“隔离范围”。如图 2A 所示，应用隔离层包括两个应用隔离范围 222、224。范围 222 表示提供给应用 112 的本地资源的视图，并且范围 224 表示提供给应用 114 的本地资源的视图。因此，图 2A 所示的实施例中，APP1 112 具有文件系统 102’的特定视图，而 APP2 114 具有文件系统 102’特定于它的另一视图。在一些实施例中，应用隔离层 220 提供本地资源 102、104、106、107 的特定视图给每个在操作系统 100 之上执行的单个应用程序。在其它实施例中，应用程序 112、114 可分组为集合，并且在这些实施例中，应用隔离层 220 为应用程序的每个集合提供本地资源的特定视图。冲突的应用程序可被放入到分离的组中以增强应用程序的兼容性和集群度。在又进一步的实施例中，属于一个集合的应用可由管理员来配置。在一些实施例中，可定义“穿过”隔离范围，其精确对应于系统范围。换句话说，在穿过隔离范围内执行的应用直接在系统范围上工作。

[0072] 在一些实施例中，应用隔离范围还被划分为分层的子范围。主要的子范围包含基础应用隔离范围，并且附加的子范围包含对此范围的各种修改，该范围对于应用的多个执行实例是可见的。例如，子范围可包含对体现应用补丁等级 (patch level) 变化的范围的修改，或者附加特征的安装或移除。在一些实施例中，对于执行应用的实例可看见的附加子范围的集合是可配置的。在一些实施例中，可见子范围的该集合对于执行应用的所有实例

是相同的，而与应用执行所代表的用户无关。在其它实施例中，可见子范围的集合对于执行应用的不同用户可变化。在另外其它实施例中，子范围的各种集合被定义且用户可具有与使用哪个集合有关的选择。在一些实施例中，在不再需要时可丢弃子范围。在一些实施例中，包含在子范围的集合中的修改可合并在一起以形成单个子范围。

[0073] 1.2 用户隔离

[0074] 现在参考图 2B，描述已经降低了应用程序兼容性和集群度问题的多用户计算机。多用户计算机包括系统层 108 中的本地资源 102、104、106、107，以及上面刚讨论的隔离环境 200。应用隔离层 220 如上讨论的那样工作，为应用或一组应用提供本地资源的修改视图。用户隔离层 240 在概念上为应用程序 112、114 提供进一步基于用户的用户身份而改变的本地资源的视图，执行该用户所代表的应用。如图 2B 所示，用户隔离层 240 可被认为是包括多个用户隔离范围 242’、242”、242”’、242””、242””’、242””” (一般地为 242)。用户隔离范围 242 提供本地资源的应用特定视图的用户特定视图。例如，在代表用户“a”的用户会话 110 中执行的 APP1 112 具有文件系统视图 102’(a)，其可由用户隔离范围 242’和应用隔离范围 222 改变或修改。

[0075] 以另一种方式，用户隔离层 240 通过将由应用特定视图修改“之上”的用户隔离范围 242’提供的用户特定视图修改分层来改变每个单独用户的本地资源视图，其中应用特定视图修改由应用隔离范围 222 提供，其依次位于由系统层提供的本地资源的系统宽视图“之上的层”。例如，当 APP1 112 的第一实例访问注册表数据库 104 中的条目时，特定于第一用户会话和应用 104’(a) 的注册表数据库的视图被咨询。如果在注册表 104’(a) 的用户特定视图中找到所请求的注册表键，则将该注册表键返回给 APP1 112。如果没有，则特定于应用 104’的注册表数据库的视图被咨询。如果在注册表 104’的应用特定视图中找到所请求的注册表键，则将该注册表键返回给 APP1 112。如果没有，则将在系统层 108 中的注册表数据库 104 中存储的注册表键（即本地资源注册表键）返回给 APP1 112。

[0076] 在一些实施例中，用户隔离层 240 为每个单独用户提供隔离范围。在其它实施例中，用户隔离层 240 为一组用户提供隔离范围，这可由组织内的职能来定义或可由管理员预定。在另外其它的实施例中，不提供用户隔离层 240。在这些实施例中，由应用程序看到的本地资源视图由应用隔离层 220 提供。隔离环境 200，尽管相对于支持应用程序由不同用户并发执行的多用户计算机作了描述，但还可以用于单用户计算机以解决由于由不同用户在同一计算机系统上顺序执行应用程序所引起的应用程序兼容性和集群度问题，以及由于由同一用户安装和执行不兼容程序所引起的那些问题。

[0077] 在一些实施例中，用户隔离范围进一步被划分为子范围。用户隔离范围对呈现到该范围中执行的应用的视图进行的修改是在该范围中的每个子范围内所包含修改的聚集。这些子范围分层于彼此之上，并且在聚集视图中，对较高子范围中的资源的修改覆盖了对较低层中的相同资源的修改。

[0078] 在这些实施例的一些中，这些子范围的一个或多个可包含对特定于用户的视图的修改。在这些实施例的一些中，一个或多个子范围可包含对特定于用户集合的视图的修改，用户集合可由系统管理员定义或在操作系统中被定义为用户组。在这些实施例的一些中，这些子范围之一可包含对特定于特殊登录会话的视图的修改，且因此这些修改在会话结束时被丢弃。在这些实施例的一些中，与用户隔离范围关联的应用实例对本地资源的改变总

是影响这些子范围之一，并且在其它实施例中，那些变化可根据改变的特殊资源来影响不同的子范围。

[0079] 1.3 本地资源的聚集视图

[0080] 上述的概念体系结构允许代表用户执行的应用以本地资源的聚集、或统一、虚拟化视图来呈现，其特定于应用和用户的组合。该聚集的视图可称为“虚拟范围”。代表用户执行的应用实例以本地资源的单个视图呈现，该视图反应了本地资源的所有可操作虚拟化实例。概念上，该聚集的视图首先包括由系统范围中的操作系统提供的、与可应用于执行应用的应用隔离范围中体现的修改相覆盖的、进一步与可应用于代表用户执行的应用的用户隔离范围中体现的修改相覆盖的本地资源集合。除了在操作系统许可拒绝了对特定用户或应用访问的情况之外，系统范围中的本地资源通过对系统上的所有用户和应用是公共的来表征。对应用隔离范围中体现的资源视图的修改被表征为对与该应用隔离范围相关联的应用的所有实例是公共的。对用户隔离范围中体现的资源视图的修改被表征为对与可应用的应用隔离范围相关联的所有应用是公共的，所述所有应用代表与用户隔离范围关联的用户执行。

[0081] 这个概念被扩展到子范围；对用户子范围中体现的资源视图的修改对于与代表用户或用户组执行的可应用隔离子范围关联的所有应用是公共的，所述用户或用户组与用户隔离子范围关联。在整个说明书中，应当理解，每当一般引用“范围”时，在子范围存在的情况下还旨在引用子范围。

[0082] 当应用请求枚举本地资源，比如文件系统或注册表数据库的一部分时，虚拟化枚举通过首先枚举本地资源的“系统范围”实例，即在系统层找到的实例（如果有的话）来构造。接下来，枚举请求资源的“应用范围”实例，即在适当应用隔离范围中找到的实例（如果有的话）。在应用隔离范围中遇到的任何枚举的资源被添加到视图。如果枚举的资源已经存在于视图（因为其也在系统范围中存在），则用应用隔离范围中遇到的资源的实例来替换它。类似地，枚举本地资源的“用户范围”实例，即在适当用户隔离范围中找到的实例（如果有的话）。并且，在用户隔离范围中遇到的任何枚举的资源被添加到视图。如果本地资源已经存在于视图（因为其也在系统范围或适当用户隔离范围中存在），则以用户隔离范围中遇到的资源的实例来替换它。以如此方式，本地资源的任何枚举将正确地反应所枚举的本地资源的虚拟化。概念上，相同的方法应用于枚举包括多个子范围的隔离范围。用来自更高子范围的资源来枚举各个子范围，更高子范围替换了来自聚集视图中的较低子范围的匹配实例。

[0083] 在其它实施例中，可从用户隔离范围向下到系统层而不是以相反方向来进行枚举。在这些实施例中，用户隔离范围被枚举。接着，应用隔离范围被枚举，并且在应用隔离范围中出现的未在用户隔离范围中被枚举的任何资源实例被添加到在构造中的聚集视图。对于只在系统范围中出现的资源，可重复类似过程。

[0084] 在此外的其它实施例中，所有隔离范围可同时被枚举且相应的枚举组合。

[0085] 如果应用试图打开本地资源的现存实例而不是要修改该资源，则返回到应用的特定实例是在虚拟范围找到的实例，或等效地为在所请求资源的双亲的虚拟化枚举中出现的实例。从隔离环境的角度看来，应用可说成是请求打开“虚拟资源”，并且用于满足该请求的本地资源的特殊实例被说成是对应于所请求资源的“真实资源”。

[0086] 如果代表用户执行的应用试图打开资源并指明其正在如此做且要修改该资源，则该应用实例通常被给予要修改的该资源的专用拷贝，因为应用隔离范围和系统范围中的资源对于代表其它用户执行的应用是公共的。通常进行对资源的用户范围拷贝，除非用户范围实例已经存在。由虚拟范围提供的聚集视图的定义是指将应用范围或系统范围资源复制到用户隔离范围的动作对于所涉及的用户和应用不改变虚拟范围提供的聚集视图，对于任何其它用户、以及任何其它应用实例也不改变。代表用户执行的应用实例对所复制资源的后续修改不影响任何其它不共享相同用户隔离范围的应用实例的聚集视图。换句话说，那些修改对于其它用户或其它不与相同应用隔离范围相关联的应用实例不改变本地资源的聚集视图。

[0087] 1.4 进程与隔离范围的关联关系

[0088] 应用可安装在特殊隔离范围中（如下详细描述）。安装在隔离范围中的应用总是与该范围关联。可替换地，应用可起动进入特殊隔离范围，或者多个隔离范围。实际上，应用被起动且与一个或多个隔离范围关联。关联的一个或多个隔离范围为进程提供本地资源的特殊视图。应用还可起动进入系统范围，即是它们可与隔离范围不关联。这允许在隔离环境内选择性地执行操作系统应用，比如 InternetExplore，以及第三方应用。

[0089] 在隔离范围内起动应用而与应用安装的位置无关的能力缓和了应用程序兼容性和集群度问题，而不要求在隔离范围内单独安装应用。选择性地起动在不同隔离范围内安装的应用的能力提供使需要帮助者应用（比如 Word、Notepad 等）的应用用相同的规则集合来起动那些帮助者应用的能力。

[0090] 此外，在多个隔离环境内起动应用的能力允许隔离的应用和公共的应用之间更好的集成。

[0091] 现在参考图 2C，并且概观中，用于将进程与隔离范围关联的方法包括步骤：以挂起状态起动进程（步骤 282）。获取与所期望隔离范围关联的规则（步骤 284），并且在存储器元件中存储进程标识符和所获取的规则（步骤 286），以及恢复挂起的进程（步骤 288）。截取或钩住由进程作出的用于访问本地资源的随后调用（步骤 290），并且如果有的话，用与进程标识符关联的规则来虚拟化对所请求资源的访问（步骤 292）。

[0092] 仍然参考图 2C 且更详细地，在挂起状态下起动进程（步骤 282）。在一些实施例中，客户起动程序用于完成该任务。在这些实施例的一些中，起动程序专门设计用于让进程起动进入所选的隔离范围中。在其它实施例中，例如通过命令行选项，起动程序将所期望的隔离范围的规范作为输入接受。

[0093] 获取与所期望的隔离范围关联的规则（步骤 284）。在一些实施例中，从永久存储元件中获取规则，比如硬盘驱动器或其它固态存储器元件。该规则可作为关系数据库、平面文件数据库、树结构数据库、二叉树结构、或其它永久数据结构来存储。在其它实施例中，规则可存储在专门配置为存储它们的数据结构中。

[0094] 进程的标识符，比如进程 id(PID) 以及所获取的规则存储在存储器元件中（步骤 286）。在一些实施例中，提供内核模式驱动程序来接收与新进程创建有关的操作系统消息。在这些实施例中，PID 和所获取规则可存储在驱动程序的上下文中。在其它实施例中，提供文件系统过滤器驱动程序或微型过滤器来截取本地资源请求。在这些实施例中，PID 和所获取的规则可存储在过滤器中。在另外其它的实施例中，由用户模式挂钩来执行所有截取

且根本不存储PID。由用户模式挂钩装置在进程初始化期间加载规则，并且不需要其它组件知道应用 PID 的规则，因为规则关联全部在进程中执行。

[0095] 恢复挂起的进程（步骤 288）且截取或钩住由进程做出的用于访问本地资源的随后调用（步骤 290），并且如果有的话，用与进程标识符关联的规则来虚拟化对所请求资源的访问（步骤 292）。在一些实施例中，文件系统过滤器驱动程序或微型过滤器截取访问本地资源的请求并确定与所截取请求关联的进程标识符是否已经与规则集合关联。如果是，与所存储的进程标识符关联的规则可用于虚拟化访问本地资源的请求。如果不是，则访问本地资源的请求不经过修改而被传递。在其它实施例中，动态链接库被加载到新创建的进程中且所述库加载隔离规则。在另外其它实施例中，内核模式技术（挂钩、过滤器驱动程序、微型过滤器）以及用户模式技术都用于截取用于访问本地资源的调用。对于文件系统过滤器驱动程序存储规则的实施例，库可从文件系统过滤器驱动程序加载规则。

[0096] 作为与隔离范围关联的进程的“孩子”的进程与它们的“双亲”进程的隔离范围关联。在一些实施例中，这是通过内核模式驱动程序通知文件系统过滤器驱动程序何时创建孩子进程来完成的。在这些实施例中，文件系统过滤器驱动程序确定双亲进程的进程标识符是否与隔离范围关联。如果是，文件系统过滤器驱动程序存储新创建的孩子进程的进程标识符与双亲进程的隔离范围之间的关联关系。在其它实施例中，可直接从系统调用文件系统过滤器驱动程序而无需使用内核模式驱动程序。在其它实施例中，在与隔离范围关联的进程中，创建新进程的操作系统函数被钩住或截取。当从这样的进程接收创建新进程的请求时，新的孩子进程和双亲的隔离范围之间的关联关系被存储。

[0097] 在一些实施例中，范围或子范围可与各个线程而不是与整个进程关联，以允许隔离在每个线程的基础上执行。在一些实施例中，每线程隔离可用于 Services 与 COM+ 服务器。

[0098] 1.4.1 范围外进程与隔离范围关联

[0099] 本发明的另一方面是将任何应用实例与任何应用隔离范围关联而不管应用是安装在该应用隔离范围、还是在另一应用隔离范围中、或没有安装在应用隔离范围中的能力。没有安装在特殊应用范围中的应用可仍旧代表用户在应用隔离范围以及对应用户隔离范围的上下文中执行，因为其本地资源经由聚集的虚拟范围对于它们是可用的，聚集的虚拟范围是由用户隔离范围、应用隔离范围和系统范围形成的。在期望在隔离范围中运行应用的情况下，这为直接安装在系统范围中的应用提供在隔离范围内运行而不要求在隔离范围内单独安装应用的能力。这还为直接安装在系统范围中的应用提供在任何隔离范围的上下文中用作为帮助者应用的能力。

[0100] 包括组成执行应用的所有进程的每个应用实例与零个或一个应用隔离范围关联，并且通过扩展只与零个或一个对应的用户隔离范围关联。该关联由规则引擎在确定如果有的话将哪个规则应用于资源请求时使用。关联关系不一定是应用安装的应用隔离范围，如果有的话。安装在隔离范围中的许多应用当在不同隔离范围或不在隔离范围中运行时将不正确工作，因为它们没有找到必须的本地资源。但是，因为隔离范围是包括系统范围的资源视图的聚集，因此在系统范围中安装的应用可通常正确地在任何应用隔离范围内工作。这意味着帮助者程序，以及进程外 COM 服务器可由在特殊隔离范围中代表用户执行的应用来调用和执行。

[0101] 在一些实施例中，安装在系统范围中的应用在隔离范围中执行，以为了识别对计算机的文件和配置设置因为该执行而做出了哪些变化。因为所有受影响的文件和配置设置隔离在用户隔离范围中，因此这些文件和配置设置是容易识别的。在这些实施例的一些中，使用此以便报告应用对文件和配置设置所做的改变。在一些实施例中，在应用执行结束时删除文件和配置设置，这有效地确保了对计算机的文件和配置设置的改变不会作为应用执行的结果存储。在另外其它实施例中，在应用执行结束时，文件和配置设置可选择性地被删除或不被删除，这有效地确保了只有一些对计算机文件和配置设置的改变作为应用执行的结果存储。

[0102] 2.0 虚拟化机制概述

[0103] 现在参考图 3A，示出在执行模式中虚拟化对本地资源访问所采用的步骤的一个实施例，执行模式将与下面的安装模式相区别。概观上，访问本地资源的请求被截取和接收（步骤 302）。该请求识别访问所搜寻的本地资源。确定与如何对待所接收的访问请求有关的可应用规则（步骤 304）。如果该规则指明请求应当被忽略，则访问请求不经过修改被传递到系统层（步骤 306）并且结果返回到请求者（步骤 310）。如果规则指明访问请求应当被重定向或被隔离，则资源满足请求的真实实例被识别（步骤 308），真实资源的修改或替换请求被传递到系统层（步骤 306）并且结果返回到请求者（步骤 310）。

[0104] 仍然参考图 3 且更详细地，识别本地资源的请求被截取或接收（步骤 302）。在一些实施例中，通过由操作系统为应用作出本地资源请求而提供的“挂钩”函数来截取本地资源的请求。在特定实施例中，这作为动态链接库来实现，该动态链接库被加载到由操作系统创建的每个新进程的地址空间中，并且在其初始化例程期间执行挂钩。加载 DLL 到每个进程可通过由操作系统提供的工具来实现，或可替换地通过修改 DLL 的可执行图像列表以便在磁盘文件或在存储器中输入来实现，因为进程的可执行图像是从磁盘加载的。在其它实施例中，功能挂钩由服务、驱动程序或后台程序执行。在其它实施例中，由操作系统提供的可执行图像，包括共享库和可执行文件可被修改或打补丁，以便提供功能挂钩或直接体现本发明的逻辑。对于操作系统是微软 WINDOWS 操作系统家族的一员的实施例，截取可由内核模式驱动程序钩住系统服务分派表来执行。在另外其它实施例中，操作系统可提供工具来允许第三方钩住请求访问本地资源的函数。在这些实施例的一些中，操作系统可经由应用编程接口 (API) 或调试工具来提供该工具。

[0105] 在其它实施例中，由与本地资源关联的驱动程序栈或句柄栈中的过滤器截取本地资源请求。例如，微软 WINDOWS 操作系统家族的一些成员提供将第三方过滤器驱动程序或微型过滤器插入到文件系统驱动程序栈的能力，并且文件系统滤器驱动程序或微型过滤器可用于提供下述的隔离功能。在另外其它实施例中，本发明包括直接合并本发明逻辑的文件系统实现方式。可替换地，操作系统可被改写以便直接提供下述的功能。在一些实施例中，可同时使用上面列出的用于截取或接收对资源的请求的一些或所有方法的组合。

[0106] 在许多实施例中，只钩住或截取打开现有本地资源或创建新的本地资源的请求。在这些实施例中，对本地资源的初始访问是引起资源被虚拟化的访问。在初始访问后，做出请求的应用程序能够利用句柄或指针或由操作系统提供且直接识别真实资源的其它标识符来与涉及虚拟化资源的操作系统通信。在其它实施例中，对虚拟化本地资源进行操作的其它类型的请求也被钩住或截取。在这些实施例的一些中，由应用打开或创建虚拟资源的

请求返回不直接识别真实资源的虚拟句柄，并且隔离环境负责将针对虚拟句柄的后续请求转换为对应的真实资源。在那些实施例的一些中，附加的虚拟操作可延迟，直到证实为需要。例如，将资源的专有可修改拷贝提供给隔离范围的操作可以延迟，直到做出了改变资源的请求，而不是当资源以允许后续修改的模式被打开的时候。

[0107] 一旦本地资源请求被截取或接收，确定如何对待特殊请求的可应用规则被确定（步骤 304）。最可应用的规则可通过参考规则引擎、规则数据库、或平面文件来确定，平面文件包含利用适当数据结构，比如列表或树结构组织的规则。在一些实施例中，规则被给与一优先权，用来确定哪个规则被认为是在两个或多个规则应用时最可应用的。在这些实施例的一些中，规则优先权包括在规则本身中，或可替换地，规则优先权可嵌入在用于存储规则的数据结构中，例如，规则优先权可由规则在树结构中的位置来指示。所确定的规则可包括与如何处理虚拟化资源请求有关的附加信息，例如将请求重定向到哪个真实资源。在特定实施例中，规则是包括过滤器字段、动作字段和数据字段的三元组。在该实施例中，过滤器字段包括用于匹配所接收的本地资源请求以确定规则对于所请求的资源名称是否有效的过滤器。动作字段可以是“忽略”、“重定向”或“隔离”。数据字段可以是任何与在规则有效时采取的动作有关的附加信息，包括在规则有效时使用的函数。

[0108] “忽略”的规则动作意思是请求直接对系统范围中所请求的本地资源进行操作。即是，将请求不经修改地传递到系统层 108（步骤 306），并且履行请求就如同没有隔离环境 200 存在一样。在该情况下，隔离环境被说成是具有一个“孔”，或者请求可称为“穿过”请求。

[0109] 如果规则动作指明本地资源请求应当被重定向或隔离，那么满足请求的真实资源被识别（步骤 308）。

[0110] “重定向”的规则动作意思是请求直接对系统范围本地资源进行操作，即使是与请求中所规定的不同的资源。通过将由所确定规则的数据字段规定或指示的映射函数应用于所请求本地资源的名称来识别真实资源。在最一般的情况下，真实本地资源可位于系统范围中的任何位置。作为一个简单的例子，规则 {prefix_match（“c:\temp\”，资源名称），重定向，replace_prefix（“c:\temp\”，“d:\wutemp\”，资源名称）} 将把对文件 c:\temp\examples\d1.txt 的请求访问重定向到真实文件 d:\wutemp\\examples\d1.txt。包括在规则的数据字段中的映射函数以及匹配函数进一步被概括为例如利用正则表达式来支持复杂的行为。一些实施例可提供规定映射函数的能力，即在用户隔离范围或可应用于代表用户执行的应用的子范围，或者应用隔离范围或可应用于该应用的子范围内定位真实资源。另外的实施例可提供规定映射函数的能力，即在可应用于不同应用的应用隔离范围内定位真实资源以便提供隔离应用之间交互的控制形式。在一些特殊实施例中，“重定向”动作可被配置为提供与“忽略”规则动作等效的行为。在这些实施例中，真实资源就是所请求的本地资源。当该条件被配置时，隔离环境可说成是具有一个“孔”，或者请求可称为“穿过”请求。

[0111] “隔离”的规则动作意思是请求对真实资源进行操作，真实资源是利用适当的用户隔离范围和应用隔离范围而被识别的。即是，真实资源的标识符通过修改所请求的本地资源的标识符来确定，所述修改利用了用户隔离范围、应用隔离范围、或两个范围或不利用这两个范围。所识别的特殊真实资源取决于所请求访问的类型以及所请求的本地资源的实例

是否已经存在于可应用的用户隔离范围、可应用的应用隔离范围和系统范围。

[0112] 图 3B 描述当接收打开本地资源的请求时，为识别真实资源所采取的步骤的一个实施例（图 3A 中的步骤 306），该请求指明资源正在被打开且要修改它。简要地，确定所请求的本地资源的用户范围实例，即在可应用的用户范围或用户子范围中存在的实例是否存在（步骤 354）。如果是，用户范围实例被识别为请求的真实资源（步骤 372），且实例被打开且返回到请求者。如果用户范围实例不存在，则确定所请求本地资源的应用范围实例是否存在（步骤 356）。如果应用范围实例存在，则将其识别为“候选”资源实例（步骤 359），并且与候选实例关联的许可数据被检查以便确定是否允许对该实例的修改（步骤 362）。如果没有应用范围实例存在，则确定所请求本地资源的系统范围实例是否存在（步骤 358）。如果不是，将错误条件返回给请求者，指明所请求虚拟资源不存在于虚拟范围（步骤 360）。但是，如果系统范围资源存在，将其识别为候选资源实例（步骤 361），并且与候选实例关联的许可数据可被检查以便确定该实例的修改是否被允许（步骤 362）。如果不是，将错误条件返回给请求者（步骤 364），指明不允许对虚拟资源的修改。如果许可数据指明候选资源可被修改，则进行对本地资源的候选实例的用户范围复制（步骤 370），用户范围实例被识别为请求的真实实例（步骤 372），并且被打开且返回给请求者。

[0113] 仍然参考图 3B 且更详细地，确定用户范围资源是否存在，或者换句话说，所请求的资源是否存在于可应用的用户范围或子范围（步骤 354）。可应用的用户范围或子范围是与用户关联的范围，其位于与做出请求的应用关联的应用隔离范围的上层。用户隔离范围或子范围在文件系统情况中可以是目录，在用户隔离范围中存在的所有文件都存储在该目录之下。在这些实施例的一些中，在用户隔离目录之下的目录树结构反应了所请求资源的路径。例如，如果所请求的文件是 c\temp\test.txt 且用户隔离范围目录是 d:\user1\app1\，则到用户范围真实文件的路径可以是 d:\user1\app1\c\temp\test.txt。在其它实施例中，到用户范围真实的路径可用本地命名约定来定义。例如，到用户范围真实文件的路径可以是 d:\user1\app1\device\hardisk1\temp\test.txt。在另外其它实施例中，用户范围文件可以都存储在单个目录下，该单个目录的名称被选择为是唯一的，且数据库可用于存储所请求的文件名与在目录中存储的对应真实文件的名称之间的映射。在另外其它实施例中，真实文件的内容可以存储在数据库中。在另外其它实施例中，本地文件系统为单个文件提供用于包含多个独立命名的“流”的工具，且用户范围文件的内容作为与文件关联的附加流存储在系统范围中。可替换地，真实文件可存储在客户文件系统中，客户文件系统被设计用于优化磁盘使用或其它感兴趣的标准。

[0114] 如果用户范围资源实例不存在，则确定应用范围资源是否存在，或者换句话说所请求的资源是否存在于应用隔离范围（步骤 356）。上述的方法用于做出该确定。例如，如果所请求的文件是 c\temp\test.txt 且应用隔离范围目录是 e:\app1\，那么到应用范围文件的路径可以是 e:\app1\c\temp\test.txt。如上，到应用范围文件的路径可用本地命名约定来存储。上述的实施例也可应用于应用隔离范围。

[0115] 如果应用范围资源不存在，则确定系统范围资源是否存在，换句话说，所请求的资源是否存在于系统范围（步骤 358）。例如如果所请求的文件是 c\temp\test.txt，则到系统范围文件的路径是 c\temp\test.txt。如果所请求的资源不存在于系统范围，则将所请求的资源不存在于虚拟范围的指示返回给请求者（步骤 360）。

[0116] 所请求资源的候选资源实例是位于应用隔离范围还是在系统范围中，确定是否允许对候选资源实例的修改（步骤 362）。例如，候选本地资源实例可具有关联的本地许可数据，指明用户不允许修改候选实例。此外，规则引擎可包括配置设置，其指令隔离环境服从或覆盖用于资源虚拟化拷贝的本地许可数据。在一些实施例中，规则可为一些虚拟资源规定修改可发生的范围，例如系统范围或应用隔离范围或子范围，或者用户隔离范围或子范围。在一些实施例中，规则引擎可根据层次或所访问资源的类型来规定应用于所虚拟化本地资源的子集的配置设置。在这些实施例的一些中，配置设置可特定于每个原子本地资源。在另一个例子中，规则引擎可包括配置数据，其禁止或允许对文件特定类的修改，比如由操作系统所定义的可执行代码或 MIME 类型或文件类型。

[0117] 如果在步骤 362 中确定不允许对候选资源实例的修改，则错误条件返回给请求者，以指明不允许对虚拟资源的写访问（步骤 364）。如果在 362 中确定允许对候选资源实例的修改，则将候选实例复制到适当的用户隔离范围或子范围（步骤 370）。对于所请求本地资源的逻辑层次结构不在隔离范围中维护的实施例中，将资源的候选实例复制到用户隔离范围可要求在用户隔离范围内创建层次位置标志符。层次位置标志符是放置在层次中的节点，用于在隔离范围中正确定位复制的资源。层次位置标志符不存储数据，被识别为位置标志符节点，并“不存在”即是其不是返回给请求者的真实资源。在一些实施例中，将节点识别为位置标志符节点是通过将事实记录在元数据中来进行的，所述元数据附着到该节点、或该节点的双亲或在系统层中的一些其它有关实体。在其它实施例中，维护单独的位置标志符节点名的存储库。

[0118] 在一些实施例中，规则可规定对特殊资源的修改可在特殊范围，比如应用隔离范围上进行。在那些情况下，步骤 370 中的复制操作可扩展以便确定对候选资源实例的修改是否在找到其的范围或子范围上被允许。如果不是，则候选资源实例被复制到允许修改的范围或子范围，其不总是用户隔离范围，并且新的拷贝被识别为真实资源实例（步骤 372）。如果是，则候选资源实例被识别为真实实例（步骤 372），并且被打开且将结果返回给请求者（步骤 306）。

[0119] 参考回图 3A，真实资源实例无论在步骤 354 中定位或在步骤 370 中创建，被打开（步骤 306）且返回给请求者（步骤 310）。在一些实施例中，这是通过将“打开”命令发给操作系统并向请求者返回来自操作系统对“打开”命令的响应来完成的。

[0120] 如果代表用户执行的应用删除本地资源，则呈现给该应用作为虚拟范围的本地资源的聚集视图必须反应该删除。删除资源的请求是专门类型修改的请求，尽管是通过将资源的存在性完全移除的修改资源的类型。概念上，删除资源的请求以类似于图 3A 中描绘的方式进行，包括按图 3B 中描绘的来确定真实资源。但是步骤 306 对于隔离资源和对于重定向或忽略资源有不同的操作。对于重定向和忽略，从系统范围删除真实资源。对于隔离，真实资源被“虚拟地”删除，或换句话说，它被删除的事实被记录在用户隔离范围中。删除的节点不包含数据，被识别为被删除的，且它及其所有后代“不存在”。换句话说，如果它是满足资源请求的资源或资源的祖先，则“资源未找到错误”被返回到请求者。进一步细节将在部分 4 中描绘。在一些实施例中，节点被识别为删除的节点是通过记录事实到元数据中来完成的，元数据附着到节点、或该节点的双亲或在系统层中的一些其它有关实体。在其它实施例中，例如在单独的子范围中维护单独的删除节点名的存储库。

[0121] 3.0 安装到隔离环境中

[0122] 上述的应用隔离范围可被认为是关联的应用实例独立于任何用户或等效地代表所有可能用户共享资源的范围，所述资源包括那些应用实例创建的资源。这种资源的主类是在应用安装到操作系统上时创建的集合。如图 1A 所示，两个不兼容应用都不能安装在同一系统范围中，但是该问题可通过安装这些应用的至少一个到隔离环境中来解决。

[0123] 隔离范围、或与隔离范围关联的应用实例可以“安装模式”操作以支持应用的安装。这相对于结合图 4-16 在下面描述的“执行模式”。在安装模式下，应用安装程序与应用隔离范围关联且假定为代表所有用户执行。应用隔离范围对于应用实例动作就如同它是“所有用户”的用户隔离范围一样，并且用户隔离范围对于应用实例不是活动的。

[0124] 图 3C 描述在安装模式中当接收打开本地资源的请求时识别真实资源所采取的步骤的一个实施例，该请求指明资源在有修改它的意图时被打开。简要地，由于没有用户隔离范围是活动的，则首先确定所请求的本地资源的应用范围实例是否存在（步骤 374）。如果应用范围实例存在，则将其识别为真实资源实例（步骤 384）。如果没有应用范围实例存在，则确定所请求的本地资源的系统范围实例是否存在（步骤 376）。如果不存在，则将错误条件返回给请求者，以指明所请求的虚拟化资源在虚拟范围中不存在（步骤 377）。但是，如果系统范围资源存在，则将其识别为候选资源实例（步骤 378），并且与候选实例关联的许可数据被检测以确定该实例的修改是否被允许（步骤 380）。如果不，则将错误条件返回给请求者（步骤 381），以指明不允许对虚拟化资源的修改。如果许可数据指明候选资源可被修改，则因为没有用户隔离范围是活动的，所以进行对本地资源的候选实例的应用范围复制（步骤 382），且将应用范围实例识别为请求的真实实例（步骤 384）。在一些实施例中，候选文件可复制到由规则引擎定义的位置。例如，规则可规定文件被复制到应用隔离范围。在其它实施例中，规则可规定文件应当被复制到的特殊的应用隔离子范围或用户隔离子范围。在文件复制到的隔离范围中没有出现的所请求文件的祖先作为隔离范围中的位置标志符被创建以便在层次中正确地定位复制的实例。

[0125] 图 3D 示出在安装模式中当接收创建本地资源的请求时识别真实资源所采取的步骤的一个实施例。简要地，由于没有隔离范围是活动的，因此首先确定所请求的本地资源的应用范围实例是否存在（步骤 390）。如果应用范围实例存在，则将错误条件返回给请求者，以指明资源由于已经存在而不能被创建（步骤 392）。如果没有应用范围实例存在，则确定所请求的本地资源的系统范围实例是否存在（步骤 394）。如果系统范围实例存在，则将错误条件返回给请求者，以指明资源由于已经存在而不能被创建（步骤 392）。在一些实施例中，用于打开资源的请求可规定资源的任何现存的系统范围实例可以被改写。如果系统范围资源实例不存在，则可将应用范围资源实例识别为将为履行请求而创建的真实实例（步骤 396）。

[0126] 通过比较图 3B 与图 3C 和 3D，可以看出，安装模式以与执行模式类似的方式操作，并且应用隔离范围代替了用户隔离范围。换句话说，对永久资源的修改，包括新资源的创建，发生在适当的应用隔离范围而不是适当的用户隔离范围。此外，对现有隔离资源访问的虚拟化还忽略了适当的用户隔离范围并开始在应用隔离范围中搜索候选真实资源。

[0127] 存在两个其它情况，其中应用隔离范围以如此方式操作以包含对现有资源的修改和新资源的创建。首先，可存在配置为在没有用户隔离层的情况下操作的隔离环境，或配置

为在没有用户隔离范围的情况下操作的虚拟范围。在该情况下，应用隔离范围只是隔离范围，其可隔离修改的和新创建的资源。其次，管理虚拟资源特殊集合的规则可规定它们要隔离在适当的应用隔离范围中而不是在适当的用户隔离范围中。并且，这意味着受该规则影响的对资源的修改和资源的创建将隔离到适当的应用隔离范围中，其中它们对于共享该范围的所有应用实例是可见的，而不是隔离在用户隔离范围中，其中它们仅仅对于执行那些应用实例的用户是可见的。

[0128] 在另外其它实施例中，隔离环境可被配置为允许某些资源在系统范围中被共享，即是，隔离环境可对于第一个或多个系统资源动作，就如同没有用户隔离范围也没有应用隔离范围存在一样。在系统范围中共享的系统资源在有修改意图来访问时绝不会被复制，因为它们由所有应用和所有用户共享，即它们是全局对象。

[0129] 4.0 详细的虚拟化例子

[0130] 上述的方法和设备可用于虚拟化各种各样的本地资源 108。下面详细描述多个这些方法和设备。

[0131] 4.1 文件系统虚拟化

[0132] 上述的方法和设备可用于虚拟化对文件系统的访问。如上所述，文件系统通常以目录的逻辑层次来组织，这些目录本身是文件且可包含其它的目录和数据文件。

[0133] 4.1.1 文件系统打开操作

[0134] 在概观中，图 4 描述为在上面所描述的虚拟化环境中打开文件所采取的步骤的一个实施例。接收或截取打开文件的请求（步骤 402）。请求包含文件名，其由隔离环境当作为虚拟文件名。确定应用于文件系统打开请求的目标的处理规则（步骤 404）。如果规则动作是“重定向”（步骤 406），则根据可应用的规则将在请求中提供的虚拟文件名映射到真实文件名（步骤 408）。利用真实文件名打开真实文件的请求被传递给操作系统且来自操作系统的结果被返回给请求者（步骤 410）。而如果规则动作是“忽略”（步骤 406），则确定真实文件名就是虚拟文件名（步骤 412），并且将打开真实文件的请求传递给操作系统且来自操作系统的结果被返回给请求者（步骤 410）。如果在步骤 406 中，规则动作是“隔离”，则在用户隔离范围中对应于虚拟文件名的文件名被识别为候选文件名（步骤 414）。换句话说，候选文件名是通过将虚拟文件名映射到特定于可应用的用户隔离范围的对应的本地文件名来形成的。候选文件存在的类别是通过检查用户隔离范围和任何与候选文件关联的元数据来确定的（步骤 416）。如果候选文件被确定为具有“否定存在”，因为候选文件或者其在用户隔离范围中的祖先目录之一被标记为删除，这意味着所请求的虚拟文件被认为不存在。在该情况下，指明所请求文件没有找到的错误条件被返回给请求者（步骤 422）。而如果在步骤 416 中候选文件被确定为“肯定存在”，因为候选文件存在于用户隔离范围且没有被标记为位置标志符节点，则所请求的虚拟文件被认为存在。候选文件被识别为请求的真实文件（步骤 418），且发出打开真实文件的请求且将结果返回给请求者（步骤 420）。但是，如果在步骤 416，候选文件具有“中性存在”，因为候选文件不存在，或者候选文件存在但被标记为位置标志符节点，则还不知道虚拟文件是否存在。在该情况下，对应于虚拟文件名的应用范围文件名被识别为候选文件名（步骤 424）。换句话说，候选文件名是通过将虚拟文件名映射到特定于可应用的应用程序隔离区域的对应本地文件名而形成的。候选文件存在的类别是通过检查应用隔离范围和任何与候选文件关联的元数据来确定的（步骤 426）。

如果候选文件被确定为具有“否定存在”，因为候选文件或者其在应用隔离范围中的祖先目录之一被标记为删除，这意味着所请求的虚拟文件被认为不存在。在该情况下，指明所请求文件没有找到的错误条件被返回给请求者（步骤 422）。而如果在步骤 426 中候选文件被确定为“肯定存在”，因为候选文件存在于应用隔离范围且没有被标记为位置标志符节点，则所请求的虚拟文件被认为存在。检查请求以确定打开请求是否指明修改文件的意图（步骤 428）。如果不是，则候选文件被识别为请求的真实文件（步骤 418），并且发出打开真实文件的请求且返回结果给请求者（步骤 420）。但是，如果在步骤 428，确定打开请求指明修改文件的意图，则与文件关联的许可数据被检查以确定文件的修改是否被允许（步骤 436）。如果没有，将错误条件返回给请求者（步骤 438）以指明文件的修改不被允许。如果许可数据指明文件可被修改，候选文件被复制到用户隔离范围（步骤 440）。在一些实施例中，候选文件被复制到由规则引擎定义的位置。例如，规则可规定文件被复制到应用隔离范围。在其它实施例中，规则可规定文件应当被复制到的特殊应用隔离子范围或用户隔离子范围。没有在文件复制到的隔离范围中出现的所请求文件的任何祖先作为隔离范围中的位置标志符被创建，以便在层次中正确地定位复制的实例。范围实例被识别为真实文件（步骤 442），并且发出打开真实文件的请求且返回结果给请求者（步骤 420）。返回步骤 426，如果候选文件具有中性存在，因为候选文件不存在，或者候选文件找到但被标记为位置标志符节点，则还不知道虚拟文件是否存在。在该情况下，对应于虚拟文件名的系统范围文件名被识别为候选文件名（步骤 430）。换句话说，候选文件名就是虚拟文件名。如果候选文件不存在（步骤 432），指明虚拟文件没有找到的错误条件被返回给请求者（步骤 434）。另一方面，如果候选文件存在（步骤 432），则检查请求以确定打开请求是否指明修改文件的意图（步骤 428）。如果不是，则候选文件被识别为请求的真实文件（步骤 418），并且发出打开真实文件的请求且返回结果给请求者（步骤 420）。但是，如果在步骤 428，确定打开请求指明修改文件的意图，则与文件关联的许可数据被检查以确定文件的修改是否被允许（步骤 436）。如果没有，将错误条件返回给请求者（步骤 438）以指明文件的修改不被允许。如果许可数据指明文件可被修改，则候选文件被复制到用户隔离范围（步骤 440）。在一些实施例中，候选文件被复制到由用户引擎定义的位置。例如，规则可规定文件被复制到应用隔离范围。在其它实施例中，规则可规定文件应当被复制到的特殊应用隔离子范围或用户隔离子范围。没有在隔离范围中出现的所请求文件的任何祖先作为隔离范围中的位置标志符被创建，以便在层次中正确地定位复制的实例。范围实例被识别为真实文件（步骤 442），并且发出打开真实文件的请求且返回结果给请求者（步骤 420）。

[0135] 这个实施例可以被稍微修改以执行对文件存在而不是打开文件的检查。在步骤 420 中打开真实文件的意图用检查真实文件以及返回给请求者的状态的存在来替换。

[0136] 仍然参考图 4 且现在更详细地，打开虚拟文件的请求被接收或截取（步骤 402）。对应的真实文件属于用户隔离范围、应用隔离范围或系统范围，或者其范围可以是应用隔离子范围或用户隔离子范围。在一些实施例中，通过替换操作系统函数或用于打开文件的函数的函数来钩住请求。在另一个实施例中，挂钩动态链接库被用来截取请求。挂钩函数可在用户模式或内核模式中执行。对于挂钩函数以用户模式执行的实施例，当创建进程时，挂钩函数可被加载到该进程的地址空间。对于挂钩函数以内核模式执行的实施例，挂钩函数可与操作系统资源相关联，这些资源用于为本地文件分派请求。对于为每种类型的文件

操作提供单独的操作系统函数的实施例，每个函数可被分别钩住。可替换地，可提供为若干类型的文件操作截取创建或打开调用的单个挂钩函数。

[0137] 请求包含文件名，隔离环境将其当作虚拟文件名。可应用于文件系统打开请求的处理规则通过咨询规则引擎来确定（步骤 404）。在一些实施例中，可应用于打开请求的处理规则是利用包括在打开请求中的虚拟名称来确定的。在一些实施例中，规则引擎可作为关系数据库提供。在其它实施例中，规则引擎可以是树结构数据库、散列表或平面文件数据库。在一些实施例中，为所请求文件提供的虚拟文件名被用作为索引以在规则引擎中定位应用于请求的一个或多个规则。在这些实施例的特殊的一些中，多个规则可存在于用于特殊文件的规则引擎中，并且在这些实施例中，具有与虚拟文件名匹配的最长前缀的规则是应用于请求的规则。在其它实施例中，进程标识符用于在规则引擎中定位应用于请求的规则，如果其存在的话。与请求关联的规则可以是忽略请求、重定向请求、或隔离请求。尽管在图 4 中示出了单个数据库事务或文件中的单次查找，但是规则查找可作为一系列规则查找来执行。

[0138] 如果规则动作是“重定向”（步骤 406），根据可应用的规则，将在请求中提供的虚拟文件名映射到真实文件名（步骤 408）。打开由真实文件名识别的真实文件的请求被传递给操作系统且来自操作系统的结果被返回给请求者（步骤 410）。例如，打开名为“file_1”文件的请求可导致打开名为“Different_file_1”的真实文件。在一个实施例中，这是通过调用挂钩函数的原始版本且将形成的真实文件名作为参数传递给该函数来完成的。对于利用文件系统过滤器驱动程序的实施例，利用虚拟文件名打开文件的第一请求导致从文件系统过滤器驱动程序返回指明所确定真实名称的 STATUS_REPARSE 响应。I/O 管理器接着重新发出文件打开请求，同时所确定真实名称包括在 STATUS_REPARSE 响应中。

[0139] 而如果规则动作是“忽略”（步骤 406），则真实文件名被确定为就是虚拟文件名（步骤 412），并且将打开真实文件的请求传递给操作系统且来自操作系统的结果被返回给请求者（步骤 410）。例如，打开名为“file_1”文件的请求可导致打开名为“file_1”的实际文件。在一个实施例中，这是通过调用挂钩函数的原始版本且将形成的真实文件名作为参数传递给该函数来完成的。

[0140] 如果在步骤 406，规则动作是“隔离”，则对应于虚拟文件名的用户范围文件名被识别为候选文件名（步骤 414）。换句话说，候选文件名是通过将虚拟文件名映射到特定于可应用的用户隔离范围的对应的本地文件名来形成的。例如，打开名为“file_1”文件的请求可导致打开名为“Isolated_file_1”的真实文件。在一个实施例中，这是通过调用挂钩函数的原始版本且将形成的真实文件名作为参数传递给该函数来完成的。对于利用文件系统过滤器驱动程序的实施例，利用虚拟文件名打开文件的第一请求导致从文件系统过滤器驱动程序返回指明所确定真实名称的 STATUS_REPARSE 响应。I/O 管理器接着重新发出文件打开请求，同时所确定真实名称包括在 REPARSE 响应中。

[0141] 在一些实施例中，为了隔离所请求的系统文件所形成的真实名称是基于所接收的虚拟文件名和范围特定标识符的。范围特定标识符可以是与应用隔离范围、用户隔离范围、会话隔离范围、应用隔离子范围、用户隔离子范围或上面范围的组合关联的标识符。范围特定标识符用于“重整”在请求中接收的虚拟名称。

[0142] 在其它实施例中，用户隔离范围或子范围可以是目录，在该目录之下存储了用户

隔离范围中存在的所有文件。在这些实施例的一些中，在用户隔离目录下的目录树结构反应了所请求资源的路径。换句话说，通过将虚拟文件路径映射到用户隔离范围来形成真实文件路径。例如，如果所请求的文件是c\temp\test.txt且用户隔离范围目录是d:\user1\app1\，则到用户范围真实文件的路径可以是d:\user1\app1\c\temp\test.txt。在其它实施例中，到用户范围真实的路径可用本地命名约定来定义。例如，到用户真实文件的路径可以是d:\user1\app1\device\hardisk1\temp\test.txt。在另外其它实施例中，用户范围文件可以都存储在单个目录下，该单个目录的名称被选择为是唯一的，且数据库可用于存储所请求的文件名与在目录中存储的对应真实文件的名称之间的映射。在另外其它实施例中，真实文件的内容可以存储在数据库中。在另外其它实施例中，本地文件系统为单个文件提供用于包含多个独立命名的“流”的工具，且用户范围文件的内容作为与文件关联的附加流存储在系统范围中。可替换地，真实文件可存储在客户文件系统中，客户文件系统被设计用于优化磁盘使用或其它感兴趣的标准。

[0143] 候选文件存在的类别是通过检查用户隔离范围和任何与候选文件关联的元数据来确定的（步骤416）。如果候选文件被确定为具有“否定存在”，因为候选文件或者其在用户隔离范围中的祖先目录之一被标记为删除，这意味着所请求的虚拟文件被认为不存在。在该情况下，指明所请求文件没有找到的错误条件被返回给请求者（步骤422）。

[0144] 在一些实施例中，少量有关文件的元数据被直接存储在真实文件名中，比如将元数据指示符作为虚拟名称的后缀，其中元数据指示符是与特殊元数据状态唯一关联的串。元数据指示符可指示或编码元数据的一个或多个位。通过虚拟文件名访问文件的请求检查由于元数据指示符的存在而引起的真实文件名的可能变化，并且获取文件本身名称的请求被钩住或截取以便以真实名称做出响应。在其它实施例中，文件的一个或多个选用名称可由虚拟文件名和元数据指示符来形成，且可利用由文件系统提供的硬链接或软链接工具来创建。如果给出的请求是利用链接的名称来访问文件，这些链接的存在可由隔离环境通过指明文件未找到而对于应用隐藏。特殊链接的存在或不存在可为每个元数据指示符指明元数据的一个位，或者可存在具有元数据指示符的链接，其可呈现多个状态来指明元数据的若干位。在另外其它实施例中，其中在文件系统支持变化的文件流的情况，变化的文件流可被创建以体现元数据，以及指明元数据若干位的流尺寸。在另外其它实施例中，文件系统可直接提供将每个文件的一些第三方元数据存储在文件系统中的能力。

[0145] 在这些实施例的特定一些中，删除的文件或文件系统元素的列表可被维护和咨询以优化对删除文件的检查。在这些实施例中，如果删除的文件被重建，则从删除文件列表中移除该文件名。在其它这些实施例中，如果列表增长超过某个大小，则从列表移除文件名。

[0146] 而如果在步骤416中，候选文件被确定为“肯定存在”，因为候选文件存在于用户隔离范围且没有被标记为位置标志符节点，则所请求的虚拟文件被认为存在。候选文件被识别为请求的真实文件（步骤418），且发出打开真实文件的请求且将结果返回给请求者（步骤420）。

[0147] 但是，如果在步骤416，候选文件具有“中性存在”，因为候选文件不存在，或者候选文件存在但被标记为位置标志符节点，则还不知道虚拟文件是否存在。在该情况下，对应于虚拟文件名的应用范围文件名被识别为候选文件名（步骤424）。换句话说，候选文件名是通过将虚拟文件名映射到特定于可应用的应用程序隔离区域的对应本地文件名而形成的。

候选文件存在的类别是通过检查应用隔离范围和任何与候选文件关联的元数据来确定的（步骤 426）。

[0148] 如果应用范围候选文件被确定为具有“否定存在”，因为候选文件或者其在应用隔离范围中的祖先目录之一被标记为删除，这意味着所请求的虚拟文件被认为不存在。在该情况下，指明所请求文件没有找到的错误条件被返回给请求者（步骤 422）。

[0149] 如果在步骤 426 中候选文件被确定为“肯定存在”，因为候选文件存在于应用隔离范围且没有被标记为位置标志符节点，则所请求的虚拟文件被认为存在。检查请求以确定打开请求是否指明修改文件的意图（步骤 428）。如果不是，则候选文件被识别为请求的真实文件（步骤 418），并且发出打开真实文件的请求且返回结果给请求者（步骤 420）。

[0150] 但是，如果在步骤 428，确定打开请求指明修改文件的意图，则与文件关联的许可数据被检查以确定文件的修改是否被允许（步骤 436）。在一些实施例中，许可数据与应用范围候选文件关联。在这些实施例的一些中，许可数据存储在规则引擎或与候选文件关联的元数据中。在其它实施例中，由操作系统来提供与候选文件关联的许可数据。此外，规则引擎可包括配置设置，其指令隔离环境服从或覆盖资源虚拟化拷贝的本地许可数据。在一些实施例中，规则可为一些虚拟资源规定修改可发生的范围，例如系统范围或应用隔离范围或子范围，或者用户隔离范围或子范围。在一些实施例中，规则引擎可根据层次或所访问资源的类型来规定应用于所虚拟化本地资源的子集的配置设置。在这些实施例的一些中，配置设置可特定于每个原子本地资源。在另一个例子中，规则引擎可包括配置数据，其禁止或允许对文件特定类的修改，比如由操作系统所定义的可执行代码或 MIME 类型或文件类型。

[0151] 如果与候选文件关联的许可数据指明其不可修改，则将错误条件返回给请求者（步骤 438）以指明文件修改不被允许。如果许可数据指明文件可被修改，则候选文件被复制到用户隔离范围（步骤 440）。在一些实施例中，候选文件被复制到由规则引擎定义的位置。例如，规则可规定文件被复制到另一个应用隔离范围。在其它实施例中，规则可规定文件应当被复制到的特殊应用隔离子范围或用户隔离子范围。没有在文件被复制到的隔离范围中出现的所请求文件的任何祖先作为隔离范围中的位置标志符被创建，以便在层次中正确地定位复制的实例。

[0152] 在一些实施例中，元数据与复制到隔离范围的文件关联，隔离范围识别数据和复制文件的时间。该信息可用于比较与文件的复制实例关联的时间戳和文件的原始实例最后修改的时间戳，或者位于较低隔离范围中文件的另一实例最后修改的时间戳。在这些实施例中，如果文件的原始实例或位于较低隔离范围中文件的实例与比所复制文件的时间戳晚的时间戳相关联，则该文件可复制到隔离范围以更新候选文件。在其它实施例中，隔离范围中文件的复制可与元数据关联，该元数据识别包含所复制的原始文件的范围。

[0153] 在另外的实施例中，复制到隔离范围的文件由于它们已经以试图修改它们的方式打开，所以监视这些文件以确定它们实际上是否被修改。在一个实施例中，复制的文件与一个标志关联，该标志在文件实际被修改时被设置。在这些实施例中，如果复制的文件没有被实际修改，则将其关闭之后从其所复制到的范围移除它，以及与所复制文件关联的任何位置标志符节点。

[0154] 范围实例被识别为真实文件（步骤 442），并且发出打开真实文件的请求且返回结

果给请求者（步骤420）。

[0155] 返回步骤426，如果候选文件具有中性存在，因为候选文件不存在，或者如果候选文件找到但被标记为位置标志符节点，则还不知道虚拟文件是否存在。在该情况下，对应于虚拟文件名的系统范围文件名被识别为候选文件名（步骤430）。换句话说，候选文件名就是虚拟文件名。

[0156] 如果候选文件不存在（步骤432），指明虚拟文件没有找到的错误条件被返回给请求者（步骤434）。另一方面，如果候选文件存在（步骤432），则检查请求以确定打开请求是否指明修改文件的意图（步骤428）。

[0157] 如上所述，如果打开候选文件而没有修改它的意图，则系统范围候选文件被识别为请求的真实文件（步骤418），并且发出打开真实文件的请求且返回结果给请求者（步骤420）。但是，如果在步骤428，确定打开请求指明修改文件的意图，则与文件关联的许可数据被检查以确定文件的修改是否被允许（步骤436）。在一些实施例中，许可数据与系统范围候选文件关联。在这些实施例的一些中，许可数据存储在规则引擎或与候选文件关联的元数据中。在其它实施例中，由操作系统来提供与候选文件关联的许可数据。

[0158] 如果与系统范围候选文件关联的许可数据指明文件不可修改，将错误条件返回给请求者（步骤438）以指明文件修改不允许。但是，如果许可数据指明文件可被修改，则候选文件被复制到用户隔离范围（步骤440）。在一些实施例中，候选文件被复制到由规则引擎定义的位置。例如，规则可规定文件被复制到应用隔离范围，或其可留在系统范围中。在其它实施例中，规则可规定文件应当被复制到的特殊应用隔离子范围或用户隔离子范围。没有在隔离范围中出现的所请求文件的任何祖先作为隔离范围中的位置标志符被创建，以便在层次中正确地定位复制的实例。

[0159] 在一些实施例中，元数据与复制到隔离范围的文件关联，隔离范围识别数据和复制文件的时间。该信息可用于比较与文件的复制实例关联的时间戳和文件的原始实例最后修改的时间戳。在这些实施例中，如果文件的原始实例与比所复制文件的时间戳晚的时间戳相关联，则原始文件可复制到隔离范围以更新候选文件。在其它实施例中，复制到隔离范围的候选文件可与元数据关联，该元数据识别所复制的原始文件来自的范围。

[0160] 在另外的实施例中，复制到隔离范围的文件由于它们已经以试图修改它们的方式打开，所以监视这些文件以确定它们实际上是否被修改。在一个实施例中，复制的文件与一个标志关联，该标志在文件实际被修改时被设置。在这些实施例中，如果复制的文件没有被实际修改，则将其关闭时从其所复制到的范围移除它，以及与所复制文件关联的任何位置标志符节点。在又另外的实施例中，当文件被实际修改时只将文件复制到适当的隔离范围。

[0161] 范围实例被识别为真实文件（步骤442），并且发出打开真实文件的请求且返回结果给请求者（步骤420）。

[0162] 4.1.2 文件系统删除操作

[0163] 现在参考图5，并且在概观上，描述了删除文件所采取的步骤的一个实施例。接收或截取删除文件的请求（步骤502）。请求包含文件名，隔离环境将该文件名当作虚拟文件名。规则确定如何处理文件操作（步骤504）。如果规则动作是“重定向”（步骤506），则根据规则将虚拟文件名直接映射到真实文件名（步骤508）。删除真实文件的请求被传递给操作系统，且来自操作系统的结果被返回给请求者（步骤510）。如果规则动作是“忽略”（步

骤 506)，则就将真实文件名识别为虚拟文件名（步骤 513)，并且删除真实文件的请求被传递给操作系统，且来自操作系统的结果被返回给请求者（步骤 510)。如果规则动作是“隔离”(步骤 506)，则确定虚拟文件存在（步骤 514)。如果虚拟文件不存在，则将指明虚拟文件不存在的错误条件返回给请求者（步骤 516)。如果虚拟文件存在，并且如果虚拟化的文件规定了目录而不是普通的文件，则虚拟目录被咨询以便确定其是否包含任何虚拟文件或虚拟子目录（步骤 518)。如果所请求的虚拟化文件是包含任何虚拟文件或虚拟子目录的虚拟目录，则虚拟目录不能被删除，且错误消息被返回（步骤 520)。如果所请求的虚拟化文件是普通的文件或没有包含虚拟文件或虚拟子目录的虚拟目录，则识别对应于虚拟文件的真实文件（步骤 522)。检查与文件关联的许可数据以确定是否允许删除（步骤 524)。如果不，返回许可错误消息（步骤 526)。但是，如果允许删除文件，且真实文件在适当的用户隔离范围中（步骤 528)，则删除真实文件（步骤 534)，且在适当的用户隔离范围中创建表示删除的虚拟文件的“删除”节点（步骤 536)。但是，如果在步骤 528 中确定真实文件不在用户隔离范围中而在适当的应用隔离范围或系统范围中，则还未存在的所请求文件的用户范围实例的每个用户范围祖先的实例被创建且标记为位置标志符（步骤 532)。这样做是为了维护用户隔离范围中目录结构的逻辑层次。接着在适当的用户隔离范围中创建表示所删除虚拟文件的用户范围的“删除”节点（步骤 536)。

[0164] 仍然参考图 5 且更详细地，删除文件的请求被接收或截取（步骤 502)。该文件属于用户隔离范围、应用隔离范围或系统范围，或者某些可应用的隔离子范围。在一些实施例中，通过替换操作系统函数或用于删除文件的函数的函数来钩住请求。在另一个实施例中，挂钩动态链接库被用来截取请求。挂钩函数可在用户模式或内核模式中执行。对于挂钩函数以用户模式执行的实施例，当创建进程时，挂钩函数可被加载到该进程的地址空间。对于挂钩函数以内核模式执行的实施例，挂钩函数可与操作系统资源相关联，这些资源用于为本地文件分派请求。对于为每种类型的文件提供单独的操作系统函数的实施例，每个函数可被分别钩住。可替换地，可提供为若干类型的文件截取创建或打开调用的单个挂钩函数。

[0165] 请求包含文件名，隔离环境将该文件名当作虚拟文件名。通过咨询规则引擎来确定可应用于删除操作的处理规则（步骤 504)。在一些实施例中，为所请求文件提供的虚拟文件名用于在规则引擎中定位应用于请求的规则。在这些实施例的特殊的一些中，多个规则可存在于用于特殊文件的规则引擎中，并且在这些实施例的一些中，具有与虚拟文件名匹配的最长前缀的规则是应用于请求的规则。在一些实施例中，规则引擎可作为关系数据库提供。在其它实施例中，规则引擎可以是树结构数据库、散列表或平面文件数据库。在一些实施例中，在请求中提供的虚拟文件名被用作为索引以在规则引擎中定位应用于请求的一个或多个规则。在其它实施例中，进程标识符用于在规则引擎中定位应用于请求的规则，如果其存在的话。与请求关联的规则可以是忽略请求、重定向请求、或隔离请求。尽管在图 5 中示出了一系列判定，但是规则查找可作为单个数据库事务出现。

[0166] 如果规则动作是“重定向”(步骤 506)，则根据可应用的规则将虚拟文件名直接映射到真实文件名（步骤 508)。删除真实文件的请求被传递给操作系统，且来自操作系统的结果被返回给请求者（步骤 510)。例如，删除名为“file_1”文件的请求可导致删除名为“Different_file_1”真实文件。在一个实施例中，这是通过调用挂钩函数的原始版本且将形成的真实文件名作为参数传递给该函数来完成的。对于利用文件系统过滤器驱动程序的

实施例，利用虚拟文件名删除文件的第一请求导致从文件系统过滤器驱动程序返回指明所确定真实名称的 STATUS_REPARSE 响应。I/O 管理器接着重新发出文件删除请求，同时所确定真实名称包括在 STATUS_REPARSE 响应中。

[0167] 在一些实施例中，与真实文件“Different_file_1”关联的操作系统许可可防止真实文件的删除。在这些实施例中，返回文件不能被删除的错误消息。

[0168] 如果规则动作是“忽略”（步骤 506），则就将真实文件名识别为虚拟文件名（步骤 513），并且删除真实文件的请求被传递给操作系统，且来自操作系统的结果被返回给请求者（步骤 510）。例如，删除名为“file_1”文件的请求可导致删除名为“file_1”的实际文件。在一个实施例中，这是通过调用挂钩函数的原始版本且将形成的真实文件名作为参数传递给该函数来完成的。对于利用文件系统过滤器驱动程序的实施例，利用虚拟文件名删除文件的第一请求导致从文件系统过滤器驱动程序返回指明真实名称的 STATUS_REPARSE 响应。I/O 管理器接着重新发出文件删除请求，同时所确定真实名称包括在 STATUS_REPARSE 响应中。

[0169] 在一些实施例中，与真实文件“file_1”关联的操作系统许可可防止真实文件的删除。在这些实施例中，返回文件不能被删除的错误消息。

[0170] 如果规则动作是“隔离”（步骤 506），则确定虚拟文件存在（步骤 514）。如果该文件不存在，则返回指明文件没有找到的错误（步骤 516）。

[0171] 但是，如果在步骤 518 确定文件存在但它不是普通的文件且不是空的虚拟目录，即它包含虚拟文件或虚拟子目录，则返回指明文件不可删除的错误消息（步骤 520）。

[0172] 但是，如果确定文件存在，并且如果所请求的虚拟化文件是普通的文件或是空的虚拟目录，即它不包含虚拟文件且不包含虚拟子目录（步骤 518），则对应于虚拟文件的真实文件被识别（步骤 522）。根据由隔离规则规定的虚拟文件名来确定真实文件名。例如，删除名为“file_1”文件的请求可导致删除名为“Isolated_file_1”的真实文件。在一个实施例中，这是通过调用挂钩函数的原始版本且将形成的真实文件名作为参数传递给该函数来完成的。对于利用文件系统过滤器驱动程序的实施例，利用虚拟文件名删除文件的第一请求导致从文件系统过滤器驱动程序返回指明真实名称的 STATUS_REPARSE 响应。I/O 管理器接着重新发出文件删除请求，同时所确定真实名称包括在 STATUS_REPARSE 响应中。

[0173] 一旦对应于虚拟文件的真实文件被识别，就确定真实文件是否可删除（步骤 524）。如果文件不可删除，则指明文件不能被删除的错误被返回（步骤 524）。在一些实施例中，许可数据和系统范围候选文件相关联。在这些实施例的一些中，许可数据存储在规则引擎或与候选文件关联的元数据中。在其它实施例中，由操作系统来提供与候选文件关联的许可数据。

[0174] 但是，如果允许文件的删除，且如果真实文件在适当的用户隔离范围中（步骤 528），则删除真实文件（步骤 534），且在适当的用户隔离范围中创建表示删除的虚拟文件的“删除”节点（步骤 536）。

[0175] 但是，如果在步骤 528 中确定真实文件不在用户隔离范围中而在适当的应用隔离范围或系统范围中，则还未存在的所请求文件的用户范围实例的每个用户范围祖先的实例被创建且标记为位置标志符（步骤 532）。这样做是为了维护用户隔离范围中目录结构的逻辑层次。接着在适当的用户隔离范围中创建表示所删除虚拟文件的用户范围的“删除”节

点（步骤536）。在一些实施例中，所删除文件的身份存储在文件或其它缓冲存储器中以优化对删除文件的检查。

[0176] 在一些实施例中，定位的虚拟化文件可与指明已经删除该虚拟化文件的元数据相关联。在其它实施例中，虚拟化文件的祖先（例如包含该文件的更高目录）与指明其可被删除的元数据相关联。在这些实施例中，指明虚拟化文件不存在的错误消息被返回。在这些实施例的特定一些中，删除文件或文件系统元素的列表被维护和咨询以优化对删除文件的检查。

[0177] 4.1.3 文件系统枚举操作

[0178] 现在参考图6，并且在概观上，示出了在所描述的虚拟化环境中枚举目录所采取的步骤的一个实施例。接收或截取枚举文件的请求（步骤602）。请求包含目录名，隔离环境将该目录名当作虚拟目录名。概念上，虚拟目录的存在是如部分4.1.1所描述的来确定的（步骤603）。如果虚拟目录不存在，则指明虚拟目录未找到的结果被返回给请求者（步骤620）。而如果虚拟目录存在，咨询规则引擎以确定为目录在枚举请求中规定的规则（步骤604）。如果规则规定了“重定向”的动作（步骤606），则如规则所规定的来确定对应于虚拟目录名的真实目录名（步骤608），且枚举由真实名称识别的真实目录，以及将枚举结果存储在工作数据存储器中（步骤612），之后跟着在后面描述的步骤630。如果所规定的规则动作不是“重定向”而是“忽略”（步骤610），则真实目录名就是虚拟目录名（步骤613）且真实目录被枚举，以及将枚举结果存储在工作数据存储器中（步骤612），之后跟着在后面描述的步骤630。但是，如果规则动作规定“隔离”，则首先枚举系统范围；即是，候选目录名就是虚拟目录名，并且如果候选目录存在，则枚举它。将枚举结果存储在工作数据存储器中。如果候选目录不存在，工作数据存储器在该阶段保持为空（步骤614）。接着，候选目录被识别为虚拟目录的应用范围实例，并且候选目录存在的类别被确定（步骤615）。如果候选目录具有“否定存在”，即它或其在范围中的祖先之一被标记为删除，则在该范围内认为它被删除，并且这是通过刷新工作数据存储器来指明的（步骤642）。而如果候选目录不具有否定存在，候选目录被枚举且将所获得的任何枚举结果合并到工作数据存储器中。尤其为枚举中的每个文件系统元素，确定其存在的类别。从工作数据存储器中移除具有否定存在的元素，并且具有肯定存在的元素，即那些存在且没有被标记为位置标志符和没有被标记为删除的元素被添加到工作数据存储器，如果在工作数据存储器中已经存在对应元素，则替换它（步骤616）。

[0179] 在任何一种情况下，候选目录被识别为虚拟目录的用户范围实例，且候选目录存在的类别被确定（步骤617）。如果候选目录具有“否定存在”，即它或其在范围中的祖先之一被标记为删除，则在该范围内认为它被删除，并且这是通过刷新工作数据存储器来指明的（步骤644）。而如果候选目录不具有否定存在，候选目录被枚举且将所获得的任何枚举结果合并到工作数据存储器中。尤其为枚举中的每个文件系统元素，确定其存在的类别。从工作数据存储器中移除具有否定存在的元素，并且具有肯定存在的元素，即那些存在且没有被标记为位置标志符和没有被标记为删除的元素被添加到工作数据存储器，如果在工作数据存储器中已经存在对应元素，则替换它（步骤618），之后跟着在后面描述的步骤630。

[0180] 接着，对于所有三类规则，来执行步骤630。询问规则引擎以寻找这样的规则集合，该规则集合的过滤器匹配所请求目录的中间孩子，但不匹配所请求的目录本身（步骤

630)。对于集合中的每个规则,利用部分 4.1.1 中描绘的逻辑来询问名称与规则中的名称匹配的虚拟孩子的存在。如果孩子具有肯定存在,其被添加到工作数据存储器,以代替那里已经有相同名称的任何孩子。如果孩子具有否定存在,移除工作数据存储器中对应于孩子的条目,如果有的话(步骤632)。最后,所构造的枚举接着从工作数据存储器返回到请求者(步骤620)。

[0181] 仍然参考图 6 且更详细地,枚举目录的请求被接收或截取(步骤602)。在一些实施例中,通过替换操作系统函数或用于枚举目录的函数的函数来钩住请求。在另一个实施例中,挂钩动态链接库被用来截取请求。挂钩函数可在用户模式或内核模式下执行。对于挂钩函数以用户模式执行的实施例,当创建进程时,挂钩函数可被加载到该进程的地址空间。对于挂钩函数以内核模式执行的实施例,挂钩函数可与操作系统资源相关联,这些资源用于为文件操作分派请求。对于为每种类型的文件操作提供单独的操作系统函数的实施例,每个函数可被分别钩住。可替换地,可提供为若干类型的文件操作截取创建或打开调用的单个挂钩函数。

[0182] 确定虚拟目录的存在(步骤603)。这是如部分 4.1.1 所描述的来实现的。如果虚拟目录不存在,则不枚举它且指明虚拟目录不存在的结果被返回给请求者(步骤620)。

[0183] 请求包含目录名,隔离环境将该目录名当作虚拟目录名。如果虚拟目录存在,则通过咨询规则引擎以定位确定枚举操作如何被处理的规则(步骤604)。在一些实施例中,规则引擎可作为关系数据库提供。在其它实施例中,规则引擎可以是树结构数据库、散列表或平面文件数据库。在一些实施例中,为所请求目录提供的虚拟目录名被用于在规则引擎中定位应用于请求的规则。在这些实施例的特殊的一些中,多个规则可存在于用于特殊目录的规则引擎中,并且在这些实施例中,具有与虚拟目录名匹配的最长前缀的规则是应用于请求的规则。在其它实施例中,进程标识符用于在规则引擎中定位应用于请求的规则,如果其存在的话。与请求关联的规则可以是忽略请求、重定向请求、或隔离请求。尽管在图 6 中示出了单个数据库事务或文件中的单次查找,但是规则查找可作为一系列规则查找来执行。

[0184] 如果规则动作是“重定向”(步骤606),则根据规则来将虚拟目录名直接映射到真实目录名(步骤608)。将枚举真实目录的请求传递到操作系统(步骤612),并且如后面描述的来执行步骤630。例如,枚举名为“directory_1”目录的请求可导致枚举名为“Different_directory_1”的真实目录。在一个实施例中,这是通过调用挂钩函数的原始版本且将形成的真实文件名作为参数传递给该函数来完成的。对于利用文件系统过滤器驱动程序的实施例,利用虚拟名称打开枚举目录的第一请求导致指明所确定真实名称的“STATUS_REPARSE”请求响应。I/O 管理器接着重新发出用于枚举的目录打开请求,同时所确定真实名称包括在 STATUS_REPARSE 响应中。

[0185] 如果规定动作不是“重定向”(步骤606)而是“忽略”(步骤610),则真实目录名就被识别为虚拟目录名(步骤613),并且将枚举真实目录的请求传递到操作系统(步骤612),并且如后面描述的来执行步骤630。例如,枚举名为“directory_1”目录的请求将导致枚举名为“directory_1”的真实目录。在一个实施例中,这是通过调用挂钩函数的原始版本且将形成的真实文件名作为参数传递给该函数来完成的。对于利用文件系统过滤器驱动程序的实施例,利用虚拟名称枚举目录的第一请求由过滤器驱动程序不经修改地传递。

[0186] 如果在步骤610中确定的规则动作不是“忽略”而是“隔离”，则枚举系统范围，即是，在请求中提供的虚拟名称被用来识别枚举的目录（步骤614）。将枚举结果存储在工作数据存储器中。在一些实施例中，工作数据存储器由存储器元件组成。在其它实施例中，工作数据存储器包括数据库或文件或固态存储器元件或永久数据存储器。

[0187] 接着，候选目录被识别为虚拟目录的应用范围实例，并且候选目录存在的类别被确定（步骤615）。如果候选目录具有“否定存在”，即它或其在范围中的祖先之一被标记为删除，则在该范围内认为它被删除，并且这是通过刷新工作数据存储器来指明的（步骤642）。

[0188] 在一些实施例中，少量有关文件的元数据被直接存储在实际文件名中，比如将元数据指示符作为虚拟名称的后缀，其中元数据指示符是与特殊元数据状态唯一关联的串。元数据指示符可指示或编码元数据的一个或多个位。通过虚拟文件名访问文件的请求检查由于元数据指示符的存在而引起的真实文件名的可能变化，并且获取文件本身名称的请求被钩住或截取以便以真实名称做出响应。在其它实施例中，文件的一个或多个选用名称可由虚拟文件名和元数据指示符来形成，且可利用由文件系统提供的硬链接或软链接工具来创建。如果给出的请求是利用链接的名称来访问文件，这些链接的存在可由隔离环境通过指明文件未找到而对于应用隐藏。特殊链接的存在或不存在可为每个元数据指示符指明元数据的一个位，或者可存在具有元数据指示符的链接，其可呈现多个状态来指明元数据的若干位。在另外其它实施例中，在文件系统支持变化的文件流的情况下，变化的文件流可被创建以体现元数据，以及指明元数据若干位的流尺寸。在另外其它实施例中，文件系统可直接提供将每个文件的一些第三方元数据存储在文件系统中的能力。在又其它实施例中，单独的子范围可用于记录删除的文件，并且在该子范围中文件的存在（没有被标记为位置标志符）是为了指出文件被删除。

[0189] 而如果候选目录不具有否定存在，候选目录被枚举且将所获得的任何枚举结果合并到工作数据存储器中。尤其为枚举中的每个文件系统元素，确定其存在的类别。从工作数据存储器中移除具有否定存在的元素，并且具有肯定存在的元素，即那些存在且没有被标记为位置标志符和没有被标记为删除的元素被添加到工作数据存储器，如果在工作数据存储器中已经存在对应元素，则替换它（步骤616）。

[0190] 在任何一种情况下，候选目录被识别为虚拟目录的用户范围实例，且候选目录存在的类别被确定（步骤617）。如果候选目录具有“否定存在”，即它或其在范围中的祖先之一被标记为删除，则在该范围内认为它被删除，并且这是通过刷新工作数据存储器来指明的（步骤644）。而如果候选目录不具有否定存在，候选目录被枚举且将所获得的任何枚举结果合并到工作数据存储器中。尤其为枚举中的每个文件系统元素，确定其存在的类别。从工作数据存储器中移除具有否定存在的元素，并且具有肯定存在的元素，即那些存在且没有被标记为位置标志符和没有被标记为删除的元素被添加到工作数据存储器，如果在工作数据存储器中已经存在对应元素，则替换它（步骤618），之后跟着在后面描述的步骤630。

[0191] 接着，对于所有三类规则，来执行步骤630。询问规则引擎以寻找这样的规则集合，该规则集合的过滤器匹配所请求目录的中间孩子，但不匹配所请求的目录本身（步骤630）。对于集合中的每个规则，利用部分4.1.1中描绘的逻辑来询问名称与规则中的名称匹配的虚拟孩子的存在。如果孩子具有肯定存在，其被添加到工作数据存储器，以代替那里

已经有相同名称的任何孩子。如果孩子具有否定存在，移除工作数据存储器中对应于孩子的条目，如果有的话（步骤632）。最后，所构造的枚举接着从工作数据存储器返回到请求者（步骤620）。

[0192] 本领域技术人员将认识到，上述的分层枚举过程可与较少修改一起应用于枚举单个隔离范围的操作，该单个隔离范围包括多个隔离子范围。工作数据存储器被创建，相继的子范围被枚举且结果合并到工作数据存储器以形成隔离范围的聚集枚举。

[0193] 4.1.4 文件系统创建操作

[0194] 现在参考图7，并且在概观上，示出了在隔离环境中创建文件所采取的步骤的一个实施例。接收或截取创建文件的请求（步骤702）。请求包含文件名，隔离环境将该文件名当作虚拟文件名。利用完全虚拟化打开所请求的文件的意图利用了可应用的规则，即利用了适当的用户和应用隔离范围，如部分4.1.1所描述的（步骤704）。如果访问被拒绝（步骤706），访问拒绝错误被返回给请求者（步骤709）。如果访问被准许（步骤706），并且所访问的文件被成功打开（步骤710），所请求的文件被返回到请求者（步骤712）。但是，如果访问被准许（步骤706），但是所请求的文件没有被成功打开（步骤710），那么如果所请求的文件的双亲也不存在（步骤714），则适于请求语义的错误被发给请求者（步骤716）。另一方面，如果利用适当的用户和应用范围在完全虚拟化视图中找到所请求的文件的双亲（步骤714），则规则接着确定文件操作如何被处理（步骤718）。如果规则动作是“重定向”或“忽略”（步骤720），则根据规则将虚拟文件名直接映射到真实文件名。具体地，如果规则动作是“忽略”，则真实文件名就被识别为虚拟文件名。而如果规则动作是“重定向”，则根据规则所规定的虚拟文件名来确定真实文件名。接着创建真实文件的请求被传递给操作系统，并且结果被返回到请求者（步骤724）。另一方面，如果在步骤720中确定的规则动作是“隔离”，则真实文件名被识别为虚拟文件名在用户隔离范围中的实例。如果真实文件已经存在，但与指明其是位置标志符或其被删除的元数据相关联，则关联的元数据被修改以移除那些指示，并且确保文件为空。在另一情况下，打开真实文件的请求被传递给操作系统（步骤726）。如果真实文件被成功打开（步骤728），则真实文件被返回到请求者(730)。另一方面，如果在步骤728中，所请求的文件未能打开，则当前不存在于用户隔离范围中的真实文件的每个祖先的位置标志符（步骤732），并且利用真实名称创建真实文件的请求被传递给操作系统且返回结果到请求者（步骤734）。

[0195] 仍然参考图7且更详细地，创建文件的请求被接收或截取（步骤702）。在一些实施例中，通过替换操作系统函数或用于创建文件的函数的函数来钩住请求。在另一个实施例中，挂钩动态链接库被用来截取请求。挂钩函数可在用户模式或内核模式中执行。对于挂钩函数以用户模式执行的实施例，当创建进程时，挂钩函数可被加载到该进程的地址空间。对于挂钩函数以内核模式执行的实施例，挂钩函数可与操作系统资源相关联，该资源用于为文件分派请求。对于为每种类型的文件操作提供单独的操作系统函数的实施例，每个函数可被分别钩住。可替换地，可提供为若干类型的文件操作截取创建或打开调用的单个挂钩函数。

[0196] 请求包含文件名，隔离环境将该文件名当作虚拟文件名。请求者试图利用完全虚拟化来打开所请求的文件，这利用了可应用的规则，即利用了适当的用户和应用隔离范围，如部分4.1.1所描述的（步骤704）。如果在完全虚拟化打开操作期间访问被拒绝（步骤

706)，访问拒绝错误被返回给请求者（步骤709)。如果访问被准许（步骤706)，并且所请求的虚拟文件被成功打开（步骤710)，对应的真实文件被返回到请求者（步骤712)。但是，如果访问被准许（步骤706)，但是所请求的文件没有被成功打开（步骤710)，那么虚拟文件被确定为不存在。如果所请求的虚拟文件的虚拟双亲也不存在，如部分4.1.1中的过程所确定的（步骤714)，则适于请求语义的错误被发给请求者（步骤716)。另一方面，如果利用适当的用户和应用隔离范围在完全虚拟化视图中找到所请求的虚拟文件的虚拟双亲（步骤714)，则通过咨询规则引擎来定位确定如何处理创建操作的规则（步骤718)。在一些实施例中，规则引擎可作为关系数据库提供。在其它实施例中，规则引擎可以是树结构数据库、散列表或平面文件数据库。在一些实施例中，为所请求文件提供的虚拟文件名被用作在规则引擎中定位应用于请求的规则。在这些实施例的特殊的一些中，多个规则可存在于用于特殊文件的规则引擎中，并且在这些实施例的一些中，具有与虚拟文件名匹配的最长前缀的规则是应用于请求的规则。在一些实施例中，进程标识符用于在规则引擎中定位应用于请求的规则，如果其存在的话。与请求关联的规则可以是忽略请求、重定向请求、或隔离请求。尽管在图7中示出了单个数据库事务或文件中的单次查找，但是规则查找可作为一系列规则查找来执行。

[0197] 如果规则动作是“重定向”或“忽略”（步骤720)，则根据规则将虚拟文件名直接映射到真实文件名（步骤724)。如果规则动作是“重定向”（步骤720)，则根据规则所规定的虚拟文件名来确定真实文件名（步骤724)。如果规则动作是“忽略”（步骤720)，则真实文件名就被识别为虚拟文件名（步骤724)。如果规则动作是“忽略”或者规则动作是“重定向”，则将利用所确定的真实文件名创建真实文件的请求传递给操作系统，并且来自操作系统的结果被返回到请求者（步骤724)。例如，创建名为“file_1”虚拟文件的请求可导致创建名为“Different_file_1”的真实文件。在一个实施例中，这是通过调用挂钩函数的原始版本且将形成的真实文件名作为参数传递给该函数来完成的（步骤724)。对于利用文件系统过滤器驱动程序的实施例，利用虚拟文件名打开文件的第一请求导致指明所确定真实名称的“STATUS_REPARSE”请求响应。I/O管理器接着重新发出文件打开请求，同时所确定真实名称包括在STATUS_REPARSE响应中。

[0198] 如果在步骤720中所确定的规则动作不是“忽略”或“重定向”，而是“隔离”，则真实文件名被识别为虚拟文件名在用户隔离范围中的实例。如果真实文件已经存在，但与指明其是位置标志符或其被删除的元数据相关联，则关联的元数据被修改以移除那些指示，并且确保文件为空。

[0199] 在一些实施例中，少量有关文件的元数据被直接存储在实际文件名中，比如将元数据指示符作为虚拟名称的后缀，其中元数据指示符是与特殊元数据状态唯一关联的串。元数据指示符可指示或编码元数据的一个或多个位。通过虚拟文件名访问文件的请求检查由于元数据指示符的存在而引起的真实文件名的可能变化，并且获取文件本身名称的请求被钩住或截取以便以真实名称做出响应。在其它实施例中，文件的一个或多个选用名称可由虚拟文件名和元数据指示符来形成，且可利用由文件系统提供的硬链接或软链接工具来创建。如果给出的请求是利用链接的名称来访问文件，这些链接的存在可由隔离环境通过指明文件未找到而对于应用隐藏。特殊链接的存在或不存在可为每个元数据指示符指明元数据的一个位，或者可存在具有元数据指示符的链接，其可呈现多个状态来指明元数据的

若干位。在另外其它实施例中，在文件系统支持变化的文件流的情况下，变化的文件流可被创建以体现元数据，以及指明元数据若干位的流尺寸。在另外其它实施例中，文件系统可直接提供将每个文件的一些第三方元数据存储在文件系统中的能力。

[0200] 在这些实施例的特定一些中，删除的文件或文件系统元素的列表可被维护和咨询以优化对删除文件的检查。在这些实施例中，如果删除的文件被重建，则从删除文件列表中移除该文件名。在其它这些实施例中，如果列表增长超过某个大小，则从列表移除文件名。

[0201] 在另一情况下，打开用户范围真实文件的请求被传递给操作系统（步骤 726）。在一些实施例中，规则可规定对应于虚拟文件的真实文件应当在用户隔离范围以外的范围中被创建，比如应用隔离范围、系统范围、用户隔离子范围或应用隔离子范围。

[0202] 如果真实文件被成功打开（步骤 728），则真实文件被返回到请求者 (730)。另一方面，如果在步骤 728 中，所请求的文件未能打开，则为当前不存在于用户隔离范围中的真实文件的每个祖先创建位置标志符（步骤 732），并且利用真实名称创建真实文件的请求被传递给操作系统且返回结果到请求者（步骤 734）。

[0203] 该实施例用于这样的操作系统，该操作系统具有只支持创建每调入 / 调用一级的 API 或工具。到每调入 / 调用多级的扩展对于本领域技术人员是显而易见的。

[0204] 4.1.5 短文件名管理

[0205] 在一些文件系统中，给予每个文件短的和长的文件名。每种名称可用于以上述任何文件操作来访问文件。对于拥有短和长文件名的每个文件，这隐含地创建了分配给该文件的短的和长的文件名之间的关联关系。在这些文件系统的一些中，由文件系统自动地将短名称分配给利用长文件名创建的文件。如果短的和长的文件名之间的关联关系不是由隔离环境维护的，则在同一目录中但在不同范围级中具有不同长名称的文件可能具有相同的短文件名，导致如果用短文件名来访问虚拟文件时出现混淆。可替换地，当文件被复制到用户隔离范围以便修改时，短文件名可能改变，意味着虚拟文件不再能够利用原始短文件名来访问。

[0206] 为了防止这些问题，首先，以修改“更高”范围的意图复制所打开的文件实例的文件系统操作保留与所复制的实例关联的短的和长的文件名之间的关联关系。其次，为新创建的隔离文件创建唯一短名称，以替代由操作系统分配的文件名。所生成的短文件名应当满足所生成的文件名不匹配相同隔离范围中同一目录中或“较低”隔离范围中同一目录中的任何已有短文件名。例如，为位于用户隔离中的文件实例所生成的短文件名不应当匹配在目录的应用范围实例中或目录的系统范围实例中的已有短文件名。

[0207] 现在参考图 7A，示出在创建新文件之后分配唯一的短文件名所采取的步骤的一个实施例。在概观上，进行检查以确定是否应当生成短文件名（步骤 752）。如果不是，返回指明将不生成短文件名的状态（步骤 754）。否则，检查文件名以根据文件系统确定其是否已经是合法的短文件名（步骤 756）。如果其已经是合法的短文件名，则返回指明将不生成短文件名的状态（步骤 754）。否则，构建合适的短文件名（步骤 758）。

[0208] 仍然参考图 7A 且更详细地，进行检查以确定是否应当生成短文件名（步骤 752）。在一些实施例中，根据存储文件名所指的文件的设备来做出判断。在其它实施例中，对于特定范围或子范围，或对于整个隔离环境能够生成短文件名。在这些实施例的一些中，注册表设置可规定是否将为特殊文件名生成短文件名。如果不应当生成短文件名，则返回将不生

成短文件名的状态（步骤 754）。

[0209] 否则，检查文件名以确定其是否已经是合法的短文件名（步骤 756）。在一些实施例中，合法的短文件名包含在文件名中的八个字符以及在可选扩展名中的三个字符。在一些实施例中，合法的短文件名只包含合法字符，比如 A-Z、a-z、0-9、`、-、！、@、#、$、%、^、&、*、(、)、-、_、'、{、和 }。在一些实施例中，前导空格或“.”或一个以上的“.”是非法的。如果所提供的文件名已经是合法的短文件名，则返回将不生成短文件名的状态（步骤 754）。

[0210] 否则，如果在步骤 756 中确定文件名是非法的短文件名，则构造适合的短文件名（步骤 758）。在一些实施例中，这是通过利用在短文件名中使用合法的长文件名的某些部分来实现的，并组合编码的重复计数以形成候选短文件名。重复计数被增加直到关联的候选短文件名合适，即其是合法的短文件名，该合法的短文件名没有被相同范围中的同一目录中，或较低范围中的同一目录中的任何其它文件使用。在其它实施例中，长文件名被重整或散列并被编码，并且与编码的重复计数组合以形成候选短文件名。重复计数被增加直到关联的候选短文件名合适，即其是合法的短文件名，该合法的短文件名没有被相同范围中的同一目录中，或较低范围中的同一目录中的任何其它文件使用。在所有这些实施例中，范围特定的串可合并到候选短文件名中以增加以低重复计数找到合适的候选短文件名的可能性。

[0211] 4.2 注册表虚拟化

[0212] 上述的方法和设备可用于虚拟化对注册表数据库的访问。如上所述，注册表数据库存储与物理附着到计算机的硬件、已经选择了哪些系统选项、如何安装计算机存储器、应用特定数据的各种项、和什么应用程序在操作系统启动时应当出现有关的信息。寄存器数据库通常以“键”170、172 的逻辑层次来组织，这些键是用于注册表值的容器。

[0213] 4.2.1 注册表键打开操作

[0214] 在概观中，图 8 描述为在上述隔离环境中打开注册表键所采取的步骤的一个实施例。接收或截取打开注册表键的请求，请求包含注册表键名，其由隔离环境当作为虚拟键名（步骤 802）。请求中可应用于虚拟名称的处理规则确定如何处理注册表键操作（步骤 804）。如果动作是“重定向”（步骤 806），则将在请求中提供的虚拟键名映射到如可应用规则规定的真实键名（步骤 808）。利用真实键名打开真实注册表键的请求被传递给操作系统且来自操作系统的结果被返回给请求者（步骤 810）。如果规则动作不是“重定向”而是“忽略”（步骤 806），则虚拟键名被识别为真实键名（步骤 812），并且将打开真实注册表键的请求传递给操作系统且来自操作系统的结果被返回给请求者（步骤 810）。如果在步骤 806 中确定的规则动作不是“重定向”且不是“忽略”而是“隔离”，则将请求中提供的虚拟键名映射到用户范围候选键名，其是与特定于可应用用户隔离范围的虚拟键名对应的键名（步骤 814）。用户范围候选键存在的类型通过检查用户隔离范围和与候选键关联的任何元数据来确定（步骤 816）。如果候选键被确定为具有“否定存在”，因为候选键或者其在用户隔离范围中的祖先键之一被标记为删除，这意味着所请求的虚拟注册表键被认为不存在。在该情况下，指明所请求文件没有找到的错误条件被返回给请求者（步骤 822）。而如果在步骤 816 中候选键被确定为具有“肯定存在”，因为候选键存在于用户隔离范围且没有被标记为位置标志符节点，则所请求的虚拟键被认为存在。候选键被识别为请求的真实键（步骤

818)，且发出打开真实键的请求且将结果返回给请求者（步骤820)。但是，如果在步骤816，候选键具有“中性存在”，因为候选键不存在，或者候选键存在但被标记为位置标志符节点，则还不知道虚拟键是否存在。在该情况下，对应于虚拟键名的应用范围键名被识别为候选键名（步骤824)。换句话说，候选键名是通过将虚拟键名映射到特定于可应用的应用隔离范围的对应本地键名而形成的。候选键存在的类别是通过检查应用隔离范围和任何与候选键关联的元数据来确定的（步骤826)。如果候选键被确定为具有“否定存在”，因为候选键或者其在应用隔离范围中的祖先键之一被标记为删除，这意味着所请求的虚拟键被认为不存在。在该情况下，指明所请求键没有找到的错误条件被返回给请求者（步骤822)。而如果在步骤826中候选键被确定为具有“肯定存在”，因为候选键存在于应用隔离范围且没有被标记为位置标志符节点，则所请求的虚拟键被认为存在。检查请求以确定打开请求是否指明修改键的意图（步骤828)。如果不是，则候选键被识别为请求的真实键（步骤818)，并且发出打开真实键的请求且返回结果给请求者（步骤820)。但是，如果在步骤828，确定打开请求指明修改键的意图，则与键关联的许可数据被检查以确定键的修改是否被允许（步骤836)。如果没有，将错误条件返回给请求者（步骤838)以指明键修改不被允许。如果许可数据指明键可被修改，候选键被复制到用户隔离范围（步骤840)。在一些实施例中，候选键被复制到由规则引擎定义的位置。例如，规则可规定键被复制到应用隔离范围。在其它实施例中，规则可规定键应当被复制到的特殊应用隔离子范围或用户隔离子范围。没有在键复制到的隔离范围中出现的所请求键的任何祖先作为隔离范围中的位置标志符被创建，以便在层次中正确地定位复制的实例。新复制的范围实例被识别为真实键（步骤842)，并且发出打开真实键的请求且返回结果给请求者（步骤820)。返回步骤826，如果候选键具有中性存在，因为候选键不存在，或者因为候选键找到但被标记为位置标志符节点，则还不知道虚拟键是否存在。在该情况下，对应于虚拟键名的系统范围键名被识别为候选键名（步骤830)。换句话说，候选键名就是虚拟键名。如果候选键不存在（步骤832)，指明虚拟键没有找到的错误条件被返回给请求者（步骤834)。另一方面，如果候选键存在（步骤832)，则检查请求以确定打开请求是否指明修改键的意图（步骤828)。如果不是，则候选键被识别为请求的真实键（步骤818)，并且发出打开真实键的请求且返回结果给请求者（步骤820)。但是，如果在步骤828，确定打开请求指明修改键的意图，则与键关联的许可数据被检查以确定键的修改是否被允许（步骤836)。如果没有，将错误条件返回给请求者（步骤838)以指明键的修改不被允许。如果许可数据指明键可被修改，则候选键被复制到用户隔离范围（步骤840)。在一些实施例中，候选数据被复制到由规则引擎定义的位置。例如，规则可规定键被复制到应用隔离范围。在其它实施例中，规则可规定键应当被复制到的特殊应用隔离子范围或用户隔离子范围。没有在隔离范围中出现的所请求键的任何祖先作为隔离范围中的位置标志符被创建，以便在层次中正确地定位复制的实例。新复制的范围实例被识别为真实键（步骤842)，并且发出打开真实键的请求且返回结果给请求者（步骤820)。

[0215] 仍然参考图8且现在更详细地，打开虚拟注册表键的请求被接收或截取（步骤802)。对应的真实注册表键属于用户隔离范围、应用隔离范围或系统范围，或者其范围可以是应用隔离子范围或用户隔离子范围。在一些实施例中，通过替换操作系统函数或用于打开注册表键的函数的函数来钩住请求。在另一个实施例中，挂钩动态链接库被用来截取请

求。挂钩函数可在用户模式或内核模式中执行。对于挂钩函数以用户模式执行的实施例，当创建进程时，挂钩函数可被加载到该进程的地址空间。对于挂钩函数以内核模式执行的实施例，挂钩函数可与操作系统资源相关联，这些资源用于为本地注册表键分派请求。对于为每种类型的键操作提供单独的操作系统函数的实施例，每个函数可被分别钩住。可替换地，可提供为若干类型的注册表键操作截取创建或打开调用的单个挂钩函数。

[0216] 请求包含注册表键名，隔离环境将其当作虚拟注册表键名。可应用于注册表键打开请求的处理规则通过咨询规则引擎来确定（步骤804）。在一些实施例中，规则引擎可作为关系数据库提供。在其它实施例中，规则引擎可以是树结构数据库、散列表或平面文件数据库。在一些实施例中，为所请求注册表键提供的虚拟注册表键名被用于在规则引擎中定位应用于请求的规则。在这些实施例的特殊的一些中，多个规则可存在于用于特殊注册表键的规则引擎中，并且在这些实施例中，具有与虚拟注册表键名匹配的最长前缀的规则是应用于请求的规则。在其它实施例中，进程标识符用于在规则引擎中定位应用于请求的规则，如果其存在的话。与请求关联的规则可以是忽略请求、重定向请求、或隔离请求。尽管在图8中示出了单个数据库事务或文件中的单次查找，但是规则查找可作为一系列规则查找来执行。

[0217] 如果规则动作是“重定向”（步骤806），根据可应用的规则，将在请求中提供的虚拟注册表键名映射到真实注册表键名（步骤808）。使用真实注册表键名打开真实注册表键的请求被传递给操作系统且来自操作系统的结果被返回给请求者（步骤810）。例如，打开名为“registry_key_1”注册表键的请求可导致打开名为“Different_registry_key_1”的真实注册表键。在一个实施例中，这是通过调用挂钩函数的原始版本且将形成的真实名称作为参数传递给该函数来完成的。在其它实施例中，概念上类似于文件系统过滤器驱动程序工具的注册表过滤器驱动程序工具可由操作系统提供。在这些实施例中，打开真实注册表键可通过应答打开虚拟键的原始请求来实现，所述应答是通过向注册表过滤器管理器发信号以利用所确定的真实键名重新解析该请求来进行的。而如果规则动作是“忽略”（步骤806），则真实注册表键名被确定为就是虚拟注册表键名（步骤812），并且将打开真实注册表键的请求传递给操作系统且来自操作系统的结果被返回给请求者（步骤810）。例如，打开名为“registry_key_1”注册表键的请求可导致打开名为“registry_key_1”的真实注册表键。在一个实施例中，这是通过调用挂钩函数的原始版本且将形成的真实名作为参数传递给该函数来完成的。在另一个实施例中，这是通过向注册表过滤器管理器发信号以便继续以正常方式处理原始的未修改请求来完成的。

[0218] 如果在步骤806，规则动作是“隔离”，则对应于虚拟注册表键名的用户范围注册表键名被识别为候选注册表键名（步骤814）。换句话说，候选注册表键名是通过将虚拟注册表键名映射到特定于可应用的用户隔离范围的对应的本地注册表键名来形成的。例如，打开名为“registry_key_1”注册表键的请求可导致打开名为“Isolated_UserScope_UserA_registry_key_1”的真实注册表键。在一个实施例中，这是通过调用挂钩函数的原始版本且将形成的真实名作为参数传递给该函数来完成的。在其它实施例中，打开真实注册表键可通过应答打开虚拟键的原始请求来实现，所述应答是通过向注册表过滤器管理器发信号以利用所确定的真实键名重新解析该请求来进行的。

[0219] 在一些实施例中，为了隔离所请求的虚拟注册表键所形成的真实名称可基于所接

收的虚拟注册表键名和范围特定标识符。范围特定标识符可以是与应用隔离范围、用户隔离范围、会话隔离范围、应用隔离子范围、用户隔离子范围或上面范围的某些组合关联的标识符。范围特定标识符用于“重整”在请求中接收的虚拟名称。

[0220] 在其它实施例中，用户隔离范围或子范围可以是注册表键，在该注册表键之下存储了用户隔离范围中存在的所有键。在这些实施例的一些中，在用户隔离键下的键层次反应了所请求资源的路径。换句话说，通过将虚拟键路径映射到用户隔离范围来形成真实键路径。例如，如果所请求的键是 HKLM\Software\Citrix\Mykey 且用户隔离范围键是 HKCU\Software\UserScope\，则到用户范围真实键的路径可以是 HKCU\Software\UserScope\HKLM\Software\Citrix\Mykey。在其它实施例中，到用户范围真实的路径可用本地命名约定来定义。例如，到用户真实键的路径可以是 HKCU\Software\UserScope\Registry\Machine\Software\Citrix\Mykey。在另外其它实施例中，用户范围键可以都存储在单个键下，该单个键的名称被选择为是唯一的，且数据库可用于存储所请求的键名与在用户隔离键中存储的对应实际键的名称之间的映射。在另外其它的实施例中，真实键的内容可以存储在数据库或文件存储器中。

[0221] 候选键存在的类别是通过检查用户隔离范围和任何与候选键关联的元数据来确定的（步骤 816）。如果候选键被确定为具有“否定存在”，因为候选键或者其在用户隔离范围中的祖先键之一被标记为删除，这意味着所请求的虚拟键被认为不存在。在该情况下，指明所请求键没有找到的错误条件被返回给请求者（步骤 822）。

[0222] 在一些实施例中，真实注册表键可与元数据关联，该元数据指明虚拟化注册表键已经被删除。在一些实施例中，与注册表键有关的元数据可作为由该键保存的区别值来存储，并且该值的存在对于注册表 API 的普通应用使用是隐藏的。在一些实施例中，少量有关注册表键的元数据被直接存储在真实键名中，比如将元数据指示符做为虚拟名称的后缀，其中元数据指示符是与特殊元数据状态唯一关联的串。元数据指示符可指示或编码元数据的一个或多个位。通过虚拟名称访问键的请求检查由于元数据指示符的存在而引起的真实键名的可能变化，并且获取键本身名称的请求被钩住或截取以便以真实名称做出响应。在其它实施例中，元数据指示符可用子键名或注册表值名称而不是用键名本身来编码。在又其它实施例中，注册表键系统可直接提供为每个键存储一些第三方元数据的能力。在一些实施例中，元数据存储在数据库或与注册表数据库分离的其它知识库中。在一些实施例中，分离的子范围可用于存储标记为删除的键。键在子范围中的存在指明该键被标记为删除。

[0223] 在这些实施例的特定一些中，删除的键或键系统元素的列表可被维护和咨询以优化对删除键的检查。在这些实施例中，如果删除的键被重建，则从删除键列表中移除该键名。在其它这些实施例中，如果列表增长超过某个大小，则从列表移除键名。

[0224] 而如果在步骤 816 中，候选键被确定为具有“肯定存在”，因为候选键存在于用户隔离范围且没有被标记为位置标志符节点，则所请求的虚拟键被认为存在。候选键被识别为请求的真实键（步骤 818），且发出打开真实键的请求且将结果返回给请求者（步骤 820）。

[0225] 但是，如果在步骤 816，候选键具有“中性存在”，因为候选键不存在，或者候选键存在但被标记为位置标志符节点，则还不知道虚拟键是否存在。在该情况下，对应于虚拟键名的应用范围键名被识别为候选键名（步骤 824）。换句话说，候选键名是通过将虚拟键名映

射到特定于可应用的应用隔离范围的对应本地键名而形成的。候选键存在的类别是通过检查应用隔离范围和任何与候选键关联的元数据来确定的（步骤826）。

[0226] 如果应用范围候选键被确定为具有“否定存在”，因为候选键或者其在应用隔离范围中的祖先键之一被标记为删除，这意味着所请求的虚拟键被认为不存在。在该情况下，指明所请求键没有找到的错误条件被返回给请求者（步骤822）。

[0227] 但是，如果在步骤826中候选键被确定为具有“肯定存在”，因为候选键存在于应用隔离范围且没有被标记为位置标志符节点，则所请求的虚拟键被认为存在。检查请求以确定打开请求是否指明修改键的意图（步骤828）。如果不是，则候选键被识别为请求的真实键（步骤818），并且发出打开真实键的请求且返回结果给请求者（步骤820）。

[0228] 但是，如果在步骤828，确定打开请求指明修改键的意图，则与键关联的许可数据被检查以确定键的修改是否被允许（步骤836）。在一些实施例中，许可数据与应用范围候选键关联。在这些实施例的一些中，许可数据存储在规则引擎或与候选键关联的元数据中。在其它实施例中，由操作系统来提供与候选键关联的许可数据。此外，规则引擎可包括配置设置，其指令隔离环境服从或覆盖资源虚拟化拷贝的本地许可数据。在一些实施例中，规则可为一些虚拟资源规定修改可发生的范围，例如系统范围或应用隔离范围或子范围，或者用户隔离范围或子范围。在一些实施例中，规则引擎可根据层次来规定应用于所虚拟化本地资源的子集的配置设置。在这些实施例的一些中，配置设置可特定于每个原子本地资源。

[0229] 如果与候选键关联的许可数据指明其不可修改，则将错误条件返回给请求者（步骤838）以指明键修改不被允许。如果许可数据指明键可被修改，则候选键被复制到用户隔离范围（步骤840）。在一些实施例中，候选数据被复制到由规则引擎定义的位置。例如，规则可规定键被复制到另一个应用隔离范围。在其它实施例中，规则可规定键应当被复制到的特殊应用隔离子范围或用户隔离子范围。没有在键复制到的隔离范围中出现的所请求键的任何祖先作为隔离范围中的位置标志符被创建，以便在层次中正确地定位复制的实例。

[0230] 在一些实施例中，元数据与复制到隔离范围的键关联，元数据识别复制键的日期和时间。该信息可用于比较与键的复制实例关联的时间戳和键的原始实例最后修改的时间戳，或者位于较低隔离范围中键的另一实例最后修改的时间戳。在这些实施例中，如果键的原始实例或位于较低隔离范围中键的实例与比所复制键的时间戳晚的时间戳相关联，则该键可复制到隔离范围以更新候选键。在其它实施例中，隔离范围中键的复制可与元数据关联，该元数据识别包含所复制的原始键的范围。

[0231] 在另外的实施例中，复制到隔离范围的键由于它们已经以试图修改它们的方式打开，所以监视这些键以确定它们实际上是否被修改。在一个实施例中，复制的键与一个标志关联，该标志在键实际被修改时被设置。在这些实施例中，如果复制的键没有被实际修改，则将其关闭之后从其所复制到的范围移除它，以及与所复制键关联的任何位置标志符节点。

[0232] 范围实例被识别为真实键（步骤842），并且发出打开真实键的请求且返回结果给请求者（步骤820）。

[0233] 返回步骤826，如果候选键具有中性存在，因为候选键不存在，或者如果候选键找到但被标记为位置标志符节点，则还不知道虚拟键是否存在。在该情况下，对应于虚拟键名的系统范围键名被识别为候选键名（步骤830）。换句话说，候选键名就是虚拟键名。

[0234] 如果候选键不存在（步骤832），指明虚拟键没有找到的错误条件被返回给请求者（步骤834）。另一方面，如果候选键存在（步骤832），则检查请求以确定打开请求是否指明修改键的意图（步骤828）。

[0235] 如上所述，如果打开候选键而没有修改它的意图，则系统范围候选键被识别为请求的真实键（步骤818），并且发出打开真实键的请求且返回结果给请求者（步骤820）。但是，如果在步骤828，确定打开请求指明修改键的意图，则与键关联的许可数据被检查以确定键的修改是否被允许（步骤836）。在一些实施例中，许可数据与应用范围候选键关联。在这些实施例的一些中，许可数据存储在规则引擎或与候选键关联的元数据中。在其它实施例中，由操作系统来提供与候选键关联的许可数据。此外，规则引擎可包括配置设置，其指令隔离环境服从或覆盖资源虚拟化拷贝的本地许可数据。在一些实施例中，规则可为一些虚拟资源规定修改可发生的范围，例如系统范围或应用隔离范围或子范围，或者用户隔离范围或子范围。在一些实施例中，规则引擎可根据层次来规定应用于所虚拟化本地资源的子集的配置设置。在这些实施例的一些中，配置设置可特定于每个原子本地资源。

[0236] 如果与系统范围候选键关联的许可数据指明键不可修改，将错误条件返回给请求者（步骤838）以指明键修改不允许。但是，如果许可数据指明键可被修改，则候选键被复制到用户隔离范围（步骤840）。在一些实施例中，候选数据被复制到由规则引擎定义的位置。例如，规则可规定键被复制到应用隔离范围，或其可留在系统范围中。在其它实施例中，规则可规定键应当被复制到的特殊应用隔离子范围或用户隔离子范围。没有在隔离范围中出现的所请求键的任何祖先作为隔离范围中的位置标志符被创建，以便在层次中正确地定位复制的实例。

[0237] 在一些实施例中，元数据与复制到隔离范围的键关联，元数据识别复制键的日期和时间。该信息可用于比较与键的复制实例关联的时间戳和键的原始实例最后修改的时间戳。在这些实施例中，如果键的原始实例与比所复制键的时间戳晚的时间戳相关联，则原始键可复制到隔离范围以更新候选键。在其它实施例中，复制到隔离范围的候选键可与元数据关联，该元数据识别所复制的原始键来自的范围。

[0238] 在另外的实施例中，复制到隔离范围的键由于它们已经以试图修改它们的方式打开，所以监视这些键以确定它们实际上是否被修改。在一个实施例中，复制的键与一个标志关联，该标志在键实际被修改时被设置。在这些实施例中，如果复制的键没有被实际修改，则当其关闭时从其所复制到的范围移除它，以及与所复制键关联的任何位置标志符节点。在又另外的实施例中，当键被实际修改时只将键复制到适当的隔离范围。

[0239] 范围实例被识别为真实键（步骤842），并且发出打开真实键的请求且返回结果给请求者（步骤820）。

[0240] 4.2.2 注册表键删除操作

[0241] 现在参考图9，并且在概观上，描述了删除注册表键所采取的步骤的一个实施例。在键可被删除之前，必须以删除访问来首先成功地打开键（步骤901）。如果键没有被成功打开，则返回错误（步骤916）。如果成功打开虚拟键，则接收或截取删除虚拟化注册表键的请求，请求包括对应于虚拟键的真实键的句柄（步骤902）。规则确定如何处理注册表键操作（步骤904）。除了可应用于要删除的键的规则之外，检查任何可应用于中间子键的其它规则（步骤905）。对于所找到的可应用于中间子键的每个规则，试图打开虚拟子键，虚拟子

键的名称由在步骤905中找到的规则所给定的名称规定。如果具有对应于在步骤905中找到的规则之一的名称的子键被成功打开（步骤906），则虚拟键被认为是具有子键，这意味着其不能被删除，并且返回错误（步骤907）。

[0242] 如果在已经试图打开步骤905中提取的所有虚拟键名（步骤906）之后，没有发现存在虚拟键，则要求进一步检查。如果规则动作不是“隔离”而是“重定向”，或者是“忽略”（步骤908），则删除实际注册表键的请求被传递给操作系统，且来自操作系统的结果被返回给请求者（步骤911）。但是，如果在步骤908中确定的规则动作是“隔离”，则聚集的虚拟化注册表键被咨询以确定其是否包含任何虚拟子键（步骤914）。如果虚拟化键具有虚拟子键，则删除不能继续，并且返回指明键还不能被删除的错误（步骤920）。如果虚拟化键不具有虚拟子键，则对应于虚拟键的真实键被检查以确定其是否在另一范围级中用相同虚拟名称掩盖了范围键（922）。如果对应于虚拟键的真实键不用相同虚拟名称掩盖不同范围的键，则删除对应于虚拟键的真实键，且返回结果（步骤926）。如果对应于虚拟键的真实键用相同虚拟名称掩盖不同范围的键，则对应于虚拟键的真实键被指明其可被删除的值来标记，并且成功的结果返回给调用者（步骤924）。

[0243] 仍然参考图9且更详细地，为了删除键，就必须首先以删除访问来打开它（步骤901）。以删除访问来打开键的请求包括隔离环境作为虚拟名称对待的键名。如部分4.2.1中描述的来执行完全虚拟化键的打开。如果虚拟化打开操作失败，错误被返回给请求者（步骤916）。如果虚拟化打开操作成功，则对应于虚拟键的真实键的句柄被返回给请求者。随后，在步骤901中删除所打开的注册表键的请求被接收或截取（步骤902）。所打开的真实注册表键属于用户隔离范围、应用隔离范围或系统范围，或者某些可应用的隔离子范围。在一些实施例中，通过替换操作系统函数或用于删除注册表键的函数的函数来钩住删除请求。在另一个实施例中，挂钩动态链接库被用来截取删除请求。挂钩函数可在用户模式或内核模式中执行。对于挂钩函数以用户模式执行的实施例，当创建进程时，挂钩函数可被加载到该进程的地址空间。对于挂钩函数以内核模式执行的实施例，挂钩函数可与操作系统资源相关联，这些资源用于为本地注册表键分派请求。在其它实施例中，概念上类似于文件系统过滤器驱动程序工具的注册表过滤器驱动程序工具可由操作系统提供。本领域技术人员可创建操作系统传递请求所到达的注册表过滤器驱动程序，以执行注册表操作，从而提供截取注册表操作请求的机制。对于为每种类型的注册表键函数提供单独的操作系统函数的实施例，每个函数可被分别钩住。可替换地，可提供为若干类型的注册表键操作截取创建或打开调用的单个挂钩函数。

[0244] 删除请求包含真实键句柄。通过向操作系统询问与该句柄关联的真实名称来确定与该句柄关联的虚拟键名。通过咨询规则引擎来确定与真实名称关联的虚拟名称，如果有的话。通过咨询规则引擎来获得确定如何处理注册表键操作的规则（步骤904）。在一些实施例中，要删除的注册表键的虚拟键名用于在规则引擎中定位应用于请求的规则。在这些实施例的特殊的一些中，多个规则可存在于用于特殊虚拟注册表键的规则引擎中，并且在这些实施例的一些中，具有与虚拟键名匹配的最长前缀的规则是应用于请求的规则。在一些实施例中，规则引擎可作为关系数据库提供。在其它实施例中，规则引擎可以是树结构数据库、散列表或平面注册表键数据库。在一些实施例中，对应于请求中虚拟键句柄的虚拟键名被用作为索引以在规则引擎中定位应用于请求的一个或多个规则。在一些实施例中，进

程标识符用于在规则引擎中定位应用于请求的规则，如果其存在的话。与请求关联的规则可以是忽略请求、重定向请求、或隔离请求。规则查找可作为一系列判定出现，或规则查找可作为单个数据库事务出现。

[0245] 要删除的键的虚拟名称用于咨询规则引擎以定位可应用于要删除的虚拟键的任何中间孩子键而不可应用于要删除的虚拟键的规则集合。该规则集合被定位出那些孩子键是否存在（步骤 905）。如果可应用于中间孩子键的规则集合不为空，则这些规则的每一个的虚拟名称都被提取。依次试图完全虚拟化打开所提取的每个虚拟孩子键名（步骤 906）。如果对应于这些虚拟名称的任何虚拟键可被成功地打开，则这意味着虚拟子键存在。这意味着虚拟键不能被删除，因为其具有存在的虚拟孩子，且返回错误（步骤 907）。如果在检查可应用于虚拟键的中间孩子的所有规则集合（步骤 905）之后，没有找到存在的虚拟子键，则能够继续删除。例如，具有虚拟名称“key_1”的键可具有可应用于“key1\subkey_1”和“key1\subkey_2”的孩子规则。在该步骤中，试图虚拟化打开“key1\subkey_1”和“key1\subkey_2”。如果这些虚拟子键中的任意一个可被成功地打开，则删除将失败，并且返回错误（步骤 907）。仅当这些虚拟子键都不存在时，删除才可继续。

[0246] 如果规则动作不是“隔离”而是“重定向”，或者是“忽略”（步骤 908），则利用真实键句柄删除实际注册表键的请求被传递给操作系统，且来自操作系统的结果被返回给请求者（步骤 911）。如果真实键包含真实子键，则请求将失败。在一个实施例中，删除真实注册表键的请求是通过调用挂钩函数的原始版本且将真实键句柄作为参数传递给该函数来完成的。在利用注册表过滤器驱动程序的实施例中，这是通过应答具有完成状态的请求来实现的，该完成状态向操作系统发信号以执行对请求的正常处理。在一些实施例中，与真实注册表键关联的操作系统许可可防止其的删除。在这些实施例中，返回虚拟注册表键不能被删除的错误消息。

[0247] 如果在步骤 908 中确定的规则动作是“隔离”，则聚集的虚拟化注册表键被咨询以确定其是否包含任何虚拟子键（步骤 914）。如果所请求的虚拟注册表键包含虚拟子键，则不能删除该虚拟键，并且返回错误给调用者（步骤 920）。

[0248] 如果所请求的虚拟注册表键不包含虚拟子键，则可删除虚拟键。接下来采取的动作取决于包含要删除的真实键的范围。例如，删除虚拟注册表键的请求可导致应用范围真实键的删除。包含真实键的范围可通过向规则引擎咨询到真实键的完整路径来确定。

[0249] 如果在特殊范围中找到要删除的真实键，且该真实键掩盖在另一范围中相同虚拟名称的另一个键，则要删除的真实键被标记为删除，且返回结果给请求者（步骤 924）。例如，如果具有相同虚拟名称的对应应用范围键或者具有相同虚拟名称的对应系统范围键具有“肯定存在”，即在范围中存在，且没有被标记为位置标志符，且没有被认为是被删除，则对应于用户范围真实键的虚拟键被认为掩盖不同范围的键。类似地，如果系统范围键存在且不认为是被删除，则应用范围键被认为掩盖了对应于相同虚拟名称的系统范围键。

[0250] 如果要删除的真实键被发现不掩盖在另一范围中相同虚拟名称的另一个键，则要删除的真实键被实际删除且结果被返回（步骤 926）。

[0251] 在一些实施例中，与真实注册表键关联的操作系统许可可防止真实注册表键的删除。在这些实施例中，返回虚拟注册表键不能被删除的错误消息。

[0252] 在一些实施例中，真实注册表键可与元数据关联，该元数据指明虚拟化注册表键

已经被删除。在一些实施例中，与注册表键有关的元数据可作为由该键保存的区别值来存储，并且该值的存在对于注册表 API 的普通应用使用是隐藏的。在一些实施例中，少量有关注册表键的元数据被直接存储在实际键名中，比如将元数据指示符作为虚拟名称的后缀，其中元数据指示符是与特殊元数据状态唯一关联的串。元数据指示符可指示或编码元数据的一个或多个位。通过虚拟名称访问键的请求检查由于元数据指示符的存在而引起的真实键名的可能变化，并且获取键本身名称的请求被钩住或截取以便以真实名称做出响应。在其它实施例中，元数据指示符可用子键名或注册表值名称而不是用键名本身来编码。在又其它实施例中，注册表键系统可直接提供为每个键存储一些第三方元数据的能力。在一些实施例中，元数据存储在数据库或与注册表数据库分离的其它知识库中。在一些实施例中，分离的子范围可用于存储标记为删除的键。键在子范围中的存在指明该键被标记为删除。

[0253] 在这些实施例的特定一些中，删除的键或键系统元素的列表可被维护和咨询以优化对删除键的检查。在这些实施例中，如果删除的键被重建，则从删除键列表中移除该键名。在其它这些实施例中，如果列表增长超过某个大小，则从列表移除键名。

[0254] 在一些实施例中，真实注册表键在相同范围中的祖先与指明其被删除的元数据关联，或其否则被指明为被删除。在这些实施例中，指明虚拟化注册表键不存在的错误消息可被返回。在这些实施例的特定一些中，所删除的注册表键或注册表键系统元素的列表可被维护和咨询以优化对所删除注册表键的检查。

[0255] 4.2.3 注册表键枚举操作

[0256] 现在参考图 10，并且在概观上，示出了在所描述的虚拟化环境中枚举键所采取的步骤的一个实施例。在键可被枚举之前，必须以枚举访问来首先成功地打开键（步骤 1001）。如果键没有被成功打开，则返回错误（步骤 1040）。如果成功打开虚拟键，则接收或截取枚举的请求，请求包括对应于虚拟键的真实键的句柄（步骤 1002）。

[0257] 对应于句柄的虚拟键名被确定，并且规则引擎被咨询以确定在枚举请求中为键规定的规则（步骤 1004）。如果规则没有规定“隔离”动作，而是规定“忽略”或规定“重定向”（步骤 1006），则枚举由真实键句柄识别的真实键，以及将枚举结果存储在工作数据存储器中（步骤 1012），之后跟着在后面描述的步骤 1030。

[0258] 但是，如果规则动作规定“隔离”，则首先枚举系统范围；即是，候选键名就是虚拟键名，并且如果候选键存在，则枚举它。将枚举结果存储在工作数据存储器中。如果候选键不存在，工作数据存储器在该阶段保持为空（步骤 1014）。接着，候选键被识别为虚拟键的应用范围实例，并且候选键存在的类别被确定（步骤 1015）。如果候选键具有“否定存在”，即它或其在范围中的祖先之一被标记为删除，则在该范围内认为它被删除，并且这是通过刷新工作数据存储器来指明的（步骤 1042）。而如果候选键不具有否定存在，候选键被枚举且将所获得的任何枚举结果合并到工作数据存储器中。尤其为枚举中的每个子键，确定其存在的类别。从工作数据存储器中移除具有否定存在的子键，并且具有肯定存在的子键，即那些存在且没有被标记为位置标志符和没有被标记为删除的子键被添加到工作数据存储器，如果在工作数据存储器中已经存在对应子键，则替换它（步骤 1016）。

[0259] 在任何一种情况下，候选键被识别为虚拟键的用户范围实例，且候选键存在的类别被确定（步骤 1017）。如果候选键具有“否定存在”，即它或其在范围中的祖先之一被标记为删除，则在该范围内认为它被删除，并且这是通过刷新工作数据存储器来指明的（步

骤1044)。而如果候选键不具有否定存在，候选键被枚举且将所获得的任何枚举结果合并到工作数据存储器中。尤其为枚举中的每个子键，确定其存在的类别。从工作数据存储器中移除具有否定存在的子键，并且具有肯定存在的子键，即那些存在且没有被标记为位置标志符和没有被标记为删除的子键被添加到工作数据存储器，如果在工作数据存储器中已经存在对应子键，则替换它（步骤1018)，之后跟着在后面描述的步骤1030。

[0260] 接着，对于所有三类规则，来执行步骤1030。询问规则引擎以寻找这样的规则集合，该规则集合的过滤器匹配所请求虚拟键名的中间孩子，但不匹配所请求的虚拟键名本身（步骤1030)。对于集合中的每个规则，确定名称与规则中的名称匹配的虚拟孩子的存在。如果孩子具有肯定存在，其被添加到工作数据存储器，以代替那里已经有相同名称的任何孩子。如果孩子具有否定存在，移除工作数据存储器中对应于孩子的条目，如果有的话（步骤1032)。最后，所构造的枚举接着从工作数据存储器返回到请求者（步骤1020)。

[0261] 仍然参考图10且更详细地，为了枚举键，就必须首先以枚举访问来打开它（步骤1001)。以枚举访问来打开键的请求包括隔离环境作为虚拟名称对待的键名。如部分4.2.1中描述的来执行完全虚拟化键的打开。如果虚拟化打开操作失败，错误被返回给请求者（步骤1040)。如果虚拟化打开操作成功，则对应于虚拟键的真实键的句柄被返回给请求者。随后，在步骤1001中枚举所打开的注册表键的请求被接收或截取（步骤1002)。所打开的真实注册表键属于用户隔离范围、应用隔离范围或系统范围，或者某些可应用的隔离子范围。在一些实施例中，通过替换操作系统函数或用于枚举注册表键的函数的函数来钩住枚举请求。在另一个实施例中，挂钩动态链接库被用来截取枚举请求。挂钩函数可在用户模式或内核模式中执行。对于挂钩函数以用户模式执行的实施例，当创建进程时，挂钩函数可被加载到该进程的地址空间。对于挂钩函数以内核模式执行的实施例，挂钩函数可与操作系统资源相关联，该资源用于为本地注册表键分派请求。在其它实施例中，概念上类似于文件系统过滤器驱动程序工具的注册表过滤器驱动程序工具可由操作系统提供。本领域技术人员可创建操作系统传递请求所到达的注册表过滤器驱动程序，以执行注册表操作，从而提供截取注册表操作请求的机制。对于为每种类型的注册表键函数提供单独的操作系统函数的实施例，每个函数可被分别钩住。可替换地，可提供为若干类型的注册表键函数截取创建或打开调用的单个挂钩函数。

[0262] 枚举请求包含真实键句柄。通过向操作系统询问与该句柄关联的真实名称来确定与该句柄关联的虚拟键名。通过咨询规则引擎来确定与真实名称关联的虚拟名称，如果有的话。

[0263] 通过咨询规则引擎来获得确定如何处理注册表键操作的规则（步骤1004)。在一些实施例中，要枚举的虚拟注册表键的虚拟键名用于在规则引擎中定位应用于请求的规则。在这些实施例的特殊的一些中，多个规则可存在于用于特殊虚拟注册表键的规则引擎中，并且在这些实施例的一些中，具有与虚拟注册表键名匹配的最长前缀的规则是应用于请求的规则。在一些实施例中，规则引擎可作为关系数据库提供。在其它实施例中，规则引擎可以是树结构数据库、散列表或平面注册表键数据库。在一些实施例中，对应于请求中虚拟键句柄的虚拟键名被用作为索引以在规则引擎中定位应用于请求的一个或多个规则。在一些实施例中，进程标识符用于在规则引擎中定位应用于请求的规则，如果其存在的话。与请求关联的规则可以是忽略请求、重定向请求、或隔离请求。规则查找可作为一系列判定出

现，或规则查找可作为单个数据库事务出现。

[0264] 如果规则动作不是“隔离”（步骤 1006），而是“忽略”或是“重定向”，则利用真实键句柄将枚举真实键的请求传递到操作系统，以及将枚举结果存储在工作数据存储器中（步骤 1012），如果有的话，并执行在后面描述的步骤 1030。

[0265] 在一个实施例中，这是通过调用挂钩函数的原始版本且将形成的真实名称作为参数传递给该函数来完成的。在其它实施例中，概念上类似于文件系统过滤器驱动程序工具的注册表过滤器驱动程序工具可由操作系统提供。在这些实施例中，枚举真实注册表键可通过应答枚举键的原始请求来实现的，所述应答是通过向注册表过滤器管理器发信号以用正常方式来处理未修改的请求来进行的。

[0266] 如果在步骤 1010 中确定的规则动作是“隔离”，则枚举系统范围。为了实现此，候选键被识别为对应于要枚举的虚拟键的系统范围键。枚举候选键，且将枚举结果存储在工作数据存储器中（步骤 1014）。在一些实施例中，工作数据存储器由存储器元件组成。在其它实施例中，工作数据存储器包括数据库或键或固态存储器元件或永久数据存储器。

[0267] 接着，候选键被识别为虚拟键的应用范围实例，并且候选键存在的类别被确定（步骤 1015）。如果候选键具有“否定存在”，即它或其在范围中的祖先之一被标记为删除，则在该范围内认为它被删除，并且这是通过刷新工作数据存储器来指明的（步骤 1042）。

[0268] 在一些实施例中，候选注册表键可与元数据关联，该元数据指明候选注册表键已经被删除。在一些实施例中，与注册表键有关的元数据可作为由该键保存的区别值来存储，并且该值的存在对于注册表 API 的普通应用使用是隐藏的。在一些实施例中，少量有关注册表键的元数据被直接存储在真实键名中，比如将元数据指示符作为虚拟名称的后缀，其中元数据指示符是与特殊元数据状态唯一关联的串。元数据指示符可指示或编码元数据的一个或多个位。通过虚拟名称访问键的请求检查由于元数据指示符的存在而引起的真实键名的可能变化，并且获取键本身名称的请求被钩住或截取以便以真实名称做出响应。在其它实施例中，元数据指示符可用子键名或注册表值名称而不是用键名本身来编码。在又其它实施例中，注册表键系统可直接提供为每个键存储一些第三方元数据的能力。在一些实施例中，元数据存储在数据库或与注册表数据库分离的其它知识库中。在一些实施例中，分离的子范围可用于存储标记为删除的键。键在子范围中的存在指明该键被标记为删除。

[0269] 而如果在步骤 1015 中，候选键不具有否定存在，候选键被枚举且将所获得的任何枚举结果合并到工作数据存储器中。尤其为枚举中的每个子键，确定其存在的类别。从工作数据存储器中移除具有否定存在的子键，并且具有肯定存在的子键，即那些存在且没有被标记为位置标志符和没有被标记为删除的子键被添加到工作数据存储器，如果在工作数据存储器中已经存在对应子键，则替换它（步骤 1016）。

[0270] 在任何一种情况下，候选键被识别为虚拟键的用户范围实例，且候选键存在的类别被确定（步骤 1017）。如果候选键具有“否定存在”，即它或其在范围中的祖先之一被标记为删除，则在该范围内认为它被删除，并且这是通过刷新工作数据存储器来指明的（步骤 1044）。而如果候选键不具有否定存在，候选键被枚举且将所获得的任何枚举结果合并到工作数据存储器中。尤其为枚举中的每个子键，确定其存在的类别。从工作数据存储器中移除具有否定存在的子键，并且具有肯定存在的子键，即那些存在且没有被标记为位置标志符和没有被标记为删除的子键被添加到工作数据存储器，如果在工作数据存储器中已经

存在对应子键，则替换它（步骤 1018），之后跟着在后面描述的步骤 1030。

[0271] 接着，对于所有三类规则，来执行步骤 1030。询问规则引擎以寻找这样的规则集合，该规则集合的过滤器匹配所请求键的中间孩子，但不匹配所请求的键本身（步骤 1030）。对于集合中的每个规则，确定名称与规则中的名称相匹配的虚拟孩子的存在。在一些实施例中，这是通过检查与虚拟孩子关联的适当隔离范围和元数据来确定的。在其它实施例中，这是通过试图打开键来确定的。如果打开请求成功，则虚拟孩子具有肯定存在。如果打开请求失败并指明虚拟孩子不存在，则虚拟孩子具有否定存在。

[0272] 如果孩子具有肯定存在，其被添加到工作数据存储器，以代替那里已经有相同名称的任何孩子。如果孩子具有否定存在，移除工作数据存储器中对应于虚拟孩子的孩子，如果有的话（步骤 1032）。最后，所构造的枚举接着从工作数据存储器返回到请求者（步骤 1020）。

[0273] 本领域技术人员将认识到，上述的分层枚举过程可与较小修改一起应用于枚举单个隔离范围的操作，该单个隔离范围包括多个隔离子范围。工作数据存储器被创建，相继的子范围被枚举且结果合并到工作数据存储器以形成隔离范围的聚集枚举。

[0274] 4.2.4 注册表创建操作

[0275] 现在参考图 11，并且在概观上，示出了在隔离环境中创建键所采取的步骤的一个实施例。接收或截取创建键的请求（步骤 1102）。请求包含键名，隔离环境将该键名当作虚拟键名。利用完全虚拟化打开所请求的键的意图利用了可应用的规则，即利用了适当的用户和应用隔离范围，如部分 4.2.1 所描述的（步骤 1104）。如果访问被拒绝（步骤 1106），访问拒绝错误被返回给请求者（步骤 1109）。如果访问被准许（步骤 1106），并且所访问的键被成功打开（步骤 1110），所请求的键被返回到请求者（步骤 1112）。但是，如果访问被准许（步骤 1106），但是所请求的键没有被成功打开（步骤 1110），那么如果所请求的键的双亲也不存在（步骤 1114），则适于请求语义的错误被发给请求者（步骤 1116）。另一方面，如果利用适当的用户和应用范围在完全虚拟化视图中找到所请求的键的双亲（步骤 1114），则规则接着确定键操作如何被处理（步骤 1118）。如果规则动作是“重定向”或“忽略”（步骤 1120），则根据规则将虚拟键名直接映射到真实键名。具体地，如果规则动作是“忽略”，则真实键名就被识别为虚拟键名。而如果规则动作是“重定向”，则根据规则所规定的虚拟键名来确定真实键名。接着创建真实键的请求被传递给操作系统，并且结果被返回到请求者（步骤 1124）。另一方面，如果在步骤 1120 中确定的规则动作是“隔离”，则真实键名被识别为虚拟键名在用户隔离范围中的实例。如果真实键已经存在，但与指明其是位置标志符或其被删除的元数据相关联，则关联的元数据被修改以移除那些指示，并且确保键为空。在另一情况下，打开真实键的请求被传递给操作系统（步骤 1126）。如果真实键被成功打开（步骤 1128），则真实键被返回到请求者 (1130)。另一方面，如果在步骤 1128 中，所请求的键未能打开，则当前不存在于用户隔离范围中的真实键的每个祖先的位置标志符（步骤 1132），并且利用真实名称创建真实键的请求被传递给操作系统且返回结果到请求者（步骤 1134）。

[0276] 仍然参考图 11 且更详细地，创建键的请求被接收或截取（步骤 1102）。在一些实施例中，通过替换操作系统函数或用于创建键的函数的函数来钩住请求。在另一个实施例中，挂钩动态链接库被用来截取请求。挂钩函数可在用户模式或内核模式中执行。对于挂

钩函数以用户模式执行的实施例，当创建进程时，挂钩函数可被加载到该进程的地址空间。对于挂钩函数以内核模式执行的实施例，挂钩函数可与操作系统资源相关联，该资源用于为键操作分派请求。对于为每种类型的键操作提供单独的操作系统函数的实施例，每个函数可被分别钩住。可替换地，可提供为若干类型的键操作截取创建或打开调用的单个挂钩函数。

[0277] 请求包含键名，隔离环境将该键名当作虚拟键名。在一些实施例中，虚拟键名可被表达为到双亲键的句柄以及到后代键的相对路径名称的组合。双亲键句柄与真实键名关联，真实键名本身与虚拟键名关联。请求者试图利用完全虚拟化来打开虚拟键，这利用了可应用的规则，即利用了适当的用户和应用隔离范围，如部分 4.2.1 所描述的（步骤 1104）。如果在完全虚拟化打开操作期间访问被拒绝（步骤 1106），访问拒绝错误被返回给请求者（步骤 1109）。如果访问被准许（步骤 1106），并且所请求的虚拟键被成功打开（步骤 1110），对应的真实键被返回到请求者（步骤 1112）。但是，如果访问被准许（步骤 1106），但是虚拟键没有被成功打开（步骤 1110），那么虚拟键被确定为不存在。如果所请求的虚拟键的虚拟双亲也不存在，如部分 4.2.1 中的过程所确定的（步骤 1114），则适于请求语义的错误被发给请求者（步骤 1116）。另一方面，如果利用适当的用户和应用范围在完全虚拟化视图中找到所请求的虚拟键的虚拟双亲（步骤 1114），则通过咨询规则引擎来定位确定如何处理创建操作的规则（步骤 1118）。在一些实施例中，规则引擎可作为关系数据库提供。在其它实施例中，规则引擎可以是树结构数据库、散列表或平面键数据库。在一些实施例中，为所请求键提供的虚拟键名被用作在规则引擎中定位应用于请求的规则。在这些实施例的特殊的一些中，多个规则可存在于用于特殊键的规则引擎中，并且在这些实施例的一些中，具有与虚拟键名匹配的最长前缀的规则是应用于请求的规则。在一些实施例中，进程标识符用于在规则引擎中定位应用于请求的规则，如果其存在的话。与请求关联的规则可以是忽略请求、重定向请求、或隔离请求。尽管在图 11 中示出了单个数据库事务或键中的单次查找，但是规则查找可作为一系列规则查找来执行。

[0278] 如果规则动作是“重定向”或“忽略”（步骤 1120），则根据规则将虚拟键名直接映射到真实键名（步骤 1124）。如果规则动作是“重定向”（步骤 1120），则根据规则所规定的虚拟键名来确定真实键名（步骤 1124）。如果规则动作是“忽略”（步骤 1120），则真实键名就被确定为虚拟键名（步骤 1124）。如果规则动作是“忽略”或者规则动作是“重定向”，则将利用所确定的真实键名创建真实键的请求传递给操作系统，并且从操作系统的结果被返回到请求者（步骤 1124）。例如，创建名为“key_1”虚拟键的请求可导致创建名为“Different_key_1”的真实键。在一个实施例中，这是通过调用挂钩函数的原始版本且将形成的真实键名作为参数传递给该函数来完成的（步骤 1124）。在其它实施例中，概念上类似于文件系统过滤器驱动程序工具的注册表过滤器驱动程序工具可由操作系统提供。在这些实施例中，创建真实注册表键可通过应答创建虚拟键的原始请求来实现，所述应答是通过向注册表过滤器管理器发信号以利用所确定的真实键名重新解析该请求来进行的。

[0279] 如果在步骤 1120 中所确定的规则动作不是“忽略”或“重定向”，而是“隔离”，则真实键名被识别为虚拟键名在用户隔离范围中的实例。如果真实键已经存在，但与指明其是位置标志符或其被删除的元数据相关联，则关联的元数据被修改以移除那些指示，并且确保键为空。

[0280] 在一些实施例中，与注册表键有关的元数据可作为由该键保存的区别值来存储，并且该值的存在对于注册表 API 的普通应用使用是隐藏的。在一些实施例中，少量有关注册表键的元数据被直接存储在真实键名中，比如将元数据指示符作为虚拟名称的后缀，其中元数据指示符是与特殊元数据状态唯一关联的串。元数据指示符可指示或编码元数据的一个或多个位。通过虚拟名称访问键的请求检查由于元数据指示符的存在而引起的真实键名的可能变化，并且获取键本身名称的请求被钩住或截取以便以真实名称做出响应。在其它实施例中，元数据指示符可用子键名或注册表值名称而不是用键名本身来编码。在又其它实施例中，注册表键系统可直接提供为每个键存储一些第三方元数据的能力。在一些实施例中，元数据存储在数据库或与注册表数据库分离的其它知识库中。在一些实施例中，分离的子范围可用于存储标记为删除的键。键在子范围中的存在指明该键被标记为删除。

[0281] 在这些实施例的特定一些中，删除的键或键系统元素的列表可被维护和咨询以优化对删除键的检查。在这些实施例中，如果删除的键被重建，则从删除键列表中移除该键名。在其它这些实施例中，如果列表增长超过某个大小，则从列表移除键名。

[0282] 在另一情况下，打开用户范围真实键的请求被传递给操作系统（步骤 1126）。在一些实施例中，规则可规定对应于虚拟键的真实键应当在用户隔离范围以外的范围中被创建，比如应用隔离范围、系统范围、用户隔离子范围或应用隔离子范围。

[0283] 如果真实键被成功打开（步骤 1128），则真实键被返回到请求者 (1130)。另一方面，如果在步骤 1128 中，所请求的键未能打开，则为当前不存在于用户隔离范围中的真实键的每个祖先创建位置标志符（步骤 1132），并且利用真实名称创建真实键的请求被传递给操作系统且返回结果到请求者（步骤 1134）。

[0284] 该实施例用于这样的操作系统，该操作系统具有只支持创建每调入 / 调用一级的 API 或工具。到每调入 / 调用多级的扩展对于本领域技术人员是显而易见的。

[0285] 4.3 命名对象虚拟化

[0286] 利用上述技术可虚拟化的系统范围资源的另一类是命名对象，其包括信号灯、互斥体、变异体、可等待定时器、事件、作业对象、段、命名管道和 mailslot。这些对象的特征在于，它们通常只存在于创建它们的进程的持续期间。这些对象的命名空间可在整个计算机上是有效的（全局范围）或者只在单个用户会话中有效（会话范围）。

[0287] 现在参考图 12，且在概观上，接收或截取创建或打开命名对象的请求（步骤 1202）。请求包含对象名，隔离环境将该对象名称当作虚拟名称。确定确定如何对待请求的规则（步骤 1204）。如果规则指明请求应当被忽略（步骤 1206），则真实对象名被确定为虚拟名称 (1207)，并且将创建或打开真实对象的请求发给操作系统 (1214)。如果所确定的规则不是忽略请求，而是指明请求应当被重定向（步骤 1208），则根据如重定向规则规定的虚拟名称来确定真实对象名（步骤 1210），并且真实对象的创建或打开请求被发给操作系统（步骤 1214）。如果规则不指明请求应当被重定向（步骤 1208），而是指明请求应当被隔离，则根据如隔离规则规定的虚拟名称来确定真实对象名（步骤 1212），并且真实对象的创建或打开命令被发给操作系统（步骤 1214）。响应于所发出的创建或打开命令而由操作系统返回的真实对象的句柄被返回给请求创建或打开虚拟对象的程序（步骤 1216）。

[0288] 仍然参考图 12 且更详细地，截取创建或打开命名对象的进程的请求（步骤 1202）。命名对象可属于会话范围或其可属于全局范围。在一些实施例中，通过替换操作系统函数

或用于创建或打开命名对象的函数的函数来钩住请求。在另一个实施例中，挂钩动态链接库被用来截取请求。挂钩函数可在用户模式或内核模式中执行。对于挂钩函数以用户模式执行的实施例，当创建进程时，挂钩函数可被加载到该进程的地址空间。对于挂钩函数以内核模式执行的实施例，挂钩函数可与操作系统资源相关联，该资源用于为系统对象分派请求。创建或打开命名对象的请求可指各种各样系统范围资源中的任意一个，系统范围资源用于进程间通信和同步并且由唯一的标识符来识别，包括信号灯、互斥体、变异体、可等待定时器、文件映射对象、事件、作业对象、段、命名管道和 mailslot。对于为每种类型的对象提供单独的操作系统函数的实施例，每个函数可被分别钩住。可替换地，可提供为若干类型的对象截取创建或打开调用的单个挂钩函数。

[0289] 截取的请求包含对象名，隔离环境将该对象名当作虚拟名称。通过咨询规则引擎来确定如何对待用于对象的请求的规则（步骤 1204）。在一些实施例中，规则引擎可作为关系数据库提供。在其它实施例中，规则引擎可以是树结构数据库、散列表或平面文件数据库。在一些实施例中，为所请求对象提供的虚拟名称被用作在规则引擎中定位应用于请求的规则。在这些实施例的特殊的一些中，多个规则可存在于用于特殊对象的规则引擎中，并且在这些实施例中，具有与虚拟名称匹配的最长前缀的规则是应用于请求的规则。在一些实施例中，进程标识符用于在规则引擎中定位应用于请求的规则，如果其存在的话。与请求关联的规则可以是忽略请求、重定向请求、或隔离请求。尽管在图 12 中作为一系列判定示出，但是规则查找可作为单个数据库事务出现。

[0290] 如果规则指明请求应当被忽略（步骤 1206），则真实对象名被确定为虚拟名称，并且将创建或打开真实对象的请求发给操作系统 (1214)。例如，创建或打开名为“Object_1”命名对象的请求可导致创建名为“Object_1”的实际对象。在一个实施例中，这是通过调用挂钩函数的原始版本且将形成的真实名称作为参数传递给该函数来完成的。

[0291] 如果通过访问规则引擎所确定的规则不是忽略请求，而是指明请求应当被重定向（步骤 1208），则根据如重定向规则规定的虚拟名称来确定真实对象名（步骤 1210），并且真实对象的创建或打开请求被发给操作系统（步骤 1214）。例如，创建或打开名为“Object_1”命名对象的请求可导致创建名为“Different_Object_1”的实际对象。在一个实施例中，这是通过调用挂钩函数的原始版本且将形成的真实名称作为参数传递给该函数来完成的。

[0292] 如果规则不指明请求应当被重定向（步骤 1208），而是指明请求应当被隔离，则根据如隔离规则规定的虚拟名称来确定真实对象名（步骤 1212），并且真实对象的创建或打开命令被发给操作系统（步骤 1214）。例如，创建或打开名为“Object_1”命名对象的请求可导致创建名为“Isolated_Object_1”的实际对象。在一个实施例中，这是通过调用挂钩函数的原始版本且将形成的真实名称作为参数传递给该函数来完成的。

[0293] 为了隔离所请求的系统对象所形成的真实名称可基于所接收的虚拟名称和范围特定标识符。范围特定标识符可以是与应用隔离范围、用户隔离范围、会话隔离范围、或这三者的组合相关联的标识符。范围特定标识符用于“重整”在请求中接收的虚拟名称。例如，如果对命名对象“Object_1”的请求对于关联标识符是“SA1”的应用隔离范围是隔离的，则真实名称可以是“Isolated_AppScope_SA1_Object_1”。下面的表格识别重整具有会话隔离范围或用户隔离范围，以及应用隔离范围的对象名称的效果。对范围组合的重整合并了在表中列出的限制。

[0294]

	会话特定标识符	用户特定标识符	应用特定标识符
全局对象	对于在用户会话的上下文中执行的所有隔离应用可用的对象	对于代表用户执行的所有隔离应用可用的对象	对于在应用隔离范围中执行的所有隔离应用可用的对象
会话对象	对于在用户会话的上下文中执行的所有隔离应用可用的对象	对于在代表用户的会话中执行的所有隔离应用可用的对象	对于在会话内的应用隔离范围中执行的所有隔离应用可用的对象

[0295] 对于操作系统是 WINDOWS 家族操作系统之一的实施例，通过切换与对象关联的全局 / 局部名称前缀来修改对象范围，对于隔离的应用，该前缀具有与重整具有对话特定标识符的对象名称相同的效果。但是，切换全局 / 局部名称前缀还影响非隔离应用的对象范围。

[0296] 响应于在步骤 1214 所发出的创建或打开命名对象的命令而由操作系统返回的真实对象的句柄被返回给请求创建或打开虚拟对象的程序（步骤 1216）。

[0297] 4.4 窗口名称虚拟化

[0298] 利用上述技术可虚拟化的系统范围资源的其它类是窗口名称和窗口类名称。图形软件应用使用窗口的名称或其窗口类作为识别应用程序是否已经正在运行的方式，或用于其它的同步形式。现在参考图 13，并且在概观上，与窗口名称或窗口类有关的请求被接收或截取（步骤 1302）。请求可以是 Win32 API 调用的形式或窗口消息的形式。这两种类型的请求被处理。那些请求包含，或请求获取由隔离环境作为虚拟名称对待的窗口名称和 / 或窗口类名称。如果请求是要获取由句柄所识别的窗口的窗口名称或窗口类（步骤 1304），则窗口映射表被咨询以确定句柄和所请求的与窗口有关的信息是否已知（步骤 1306）。如果是，来自窗口映射表的所请求的信息被返回给请求者（步骤 1308）。如果不是，请求被传递到操作系统（步骤 1310），且结果返回给请求者（步骤 1314）。如果在步骤 1304，请求提供窗口名称或窗口类，则检查请求以确定其是否规定了由操作系统定义的窗口类之一（步骤 1320）。如果是，则请求被发给操作系统，且从操作系统返回的结果被返回给请求者（步骤 1322）。如果请求没有规定由操作系统定义的窗口类之一，则基于虚拟类名和规则来确定真实类名（步骤 1324）且基于虚拟窗口名称和规则来确定真实窗口名称（步骤 1326）。接着利用真实窗口和真实类名将请求传递到操作系统（步骤 1328）。如果在步骤 1324 和 1326 确定的真实窗口名称或真实窗口类名不同于对应的虚拟名称，则窗口句柄的窗口映射表条目被更新以记录在请求中提供的虚拟窗口名称或虚拟类名（步骤 1330）。如果来自操作系统的响应包括本地窗口名称或类的本地标识，则它们被在请求中提供的虚拟窗口名称或虚拟类名替换（步骤 1312）且将结果返回给请求者（步骤 1314）。

[0299] 仍然参考图 13 且更详细地，与窗口名称或窗口类有关的请求被接收或截取（步骤 1302）。那些请求包含，或请求获取由隔离环境作为虚拟名称对待的窗口名称和 / 或窗口类名称。

[0300] 如果请求是要获取由句柄所识别的窗口的窗口名称或窗口类（步骤 1304），则窗口映射表被咨询以确定句柄和所请求的与窗口有关的信息是否已知（步骤 1306）。在一些实施例中，替代映射表，利用由操作系统提供的工具为每个窗口和窗口类存储附加的数据。

[0301] 如果是，来自窗口映射表的所请求的信息被返回给请求者（步骤 1308）。如果不是，请求被传递到操作系统（步骤 1310），且请求返回给请求者（步骤 1314）。

[0302] 如果在步骤 1304，请求提供窗口名称或窗口类，则检查请求以确定其是否规定了由操作系统定义的窗口类之一（步骤 1320）。如果是，则请求被传递到操作系统，且从操作系统返回的结果被返回给请求者（步骤 1322）。

[0303] 如果请求没有规定由操作系统定义的窗口类之一，则基于虚拟类名和规则来确定真实类名（步骤 1324）且基于虚拟窗口名称和规则来确定真实窗口名称（步骤 1326）。接着利用真实窗口和真实类名将请求传递到操作系统（步骤 1328）。在一些实施例中，窗口名称和窗口类名可是原子 (atom)，而不是字符串文字。通常，应用将串放置在原子表中且接收 16 位整数，调用可用于访问串的原子。

[0304] 如果在步骤 1324 和 1326 确定的真实窗口名称或真实窗口类名不同于对应的虚拟名称，则窗口句柄的窗口映射表条目被更新以记录在请求中提供的虚拟窗口名称或虚拟类名（步骤 1330）。

[0305] 如果来自操作系统的响应包括本地窗口名称或类的本地标识，则它们被在请求中提供的虚拟窗口名称或虚拟类名替换（步骤 1312）且将结果返回给请求者（步骤 1314）。

[0306] 现在参考图 13A，如这里所示的来确定真实窗口名称或窗口类名。咨询规则引擎以确定应用于请求的规则（步骤 1352）。如果规则动作是“忽略”（步骤 1354），则真实名称等于虚拟名称（步骤 1356）。但是，如果规则动作不是“忽略”而是“重定向”（步骤 1358），则如由重定向规则规定的根据虚拟名称来确定真实名称（步骤 1360）。但是，如果规则动作不是“重定向”而是“隔离”，则利用范围特定标识符根据虚拟名称确定真实名称（步骤 1362）。

[0307] 在一些实施例中，特殊的范围特定标识符在规则中规定。在其它实施例中，所使用的范围特定标识符与应用隔离范围关联，该应用隔离范围与请求进程关联。这允许窗口或窗口类由任何其它与相同应用隔离范围关联的应用来使用。在诸如许多微软 WINDOWS 操作系统家族的操作系统中，窗口名称和窗口类已经在会话中被隔离，这意味着只有在与相同应用隔离范围关联的相同会话中执行的应用才能使用窗口名称或窗口类。

[0308] 在微软 WINDOWS 操作系统家族的一些中，窗口名称用作为标题栏中窗口的标题。所期望的是，处理非客户区画图窗口消息以确保在窗口标题栏中显示的窗口标题反应虚拟名称而不是特殊窗口的真实名称。当非客户区画图消息被截取时，从映射表获取与窗口关联的虚拟名称，如果有的话。如果获取虚拟名称，则利用虚拟名称作为窗口标题来画图非客户区，且指明已经处理了请求消息。如果没有获取虚拟名称，则将请求指明为未被处理，利用窗口的真实名称将请求传递到画图标题栏的原始函数。

[0309] 4.5 进程外 COM 服务器虚拟化

[0310] 软件组件技术，比如 COM、CORBA、.NET 以及其它允许软件组件作为离散单元被开发、部署、寄存、发现、激活或实例化及利用。在多数组件模型中，组件可以在调用者的进程中或在相同计算机或分离计算机的分离进程中整体地执行，尽管一些组件只支持这些情况的子集。

[0311] 一个或多个唯一标识符识别这些组件。通常，组件基础结构提供代理激活请求的服务或守护程序。希望利用组件启动的软件进程将请求传递给代理以便激活由组件标识符规定的组件。代理激活所请求的组件，如果可能的话，且返回对所激活实例的引用。在这些组件基础结构的一些中，相同组件的多个版本可能不共存，因为组件标识符在版本之间仍然相同。

[0312] 微软 WINDOWS 操作系统家族的一些成员提供称为 COM 的组件基础结构。COM 组件（“COM 服务器”）由称为类标识符 (CLSID) 的 GUID 来识别，并且每个组件提供一个或多个接口，每个接口具有其自己的唯一接口标识符 (UIID)。COM 服务器控制管理器 (CSCM) 是用于进程外激活请求的代理，并且它提供允许调用者请求经由 CLSID 激活 COM 服务器的接口。尽管下面的描述将在 COM 服务器和 COM 客户端方面来叙述，但是本领域技术人员将理解其可应用于 CORBA、.NET 和其它为软件组件提供动态激活的软件架构。

[0313] 当 COM 组件被安装到计算机时，它们在注册表数据库的已知部分中注册它们的 CLSID，以及 CSCM 起动 COM 服务器新实例所需要的信息。对于进程外 COM 服务器，这可包括使可执行程序运行的路径和命令行参数。相同 COM 服务器的多个版本共享相同的 CLSID，因此一次只有一个版本可以安装到计算机上。

[0314] 在某些实施例中，应用（作为 COM 客户端）通过调用 COM API（例如 CoCreateInstance() 或 CoCreateInstanceEx()）来实例化 COM 服务器。对该调用的参数规定了所期望的激活上下文：进程中；在相同计算机上的进程外；在远程计算机上的进程外；或允许 COM 子系统确定使用这三种情况的哪种。如果确定要求进程外激活，则包括 CLSID 的请求被传递到 CSCM。CSCM 使用注册表数据库来定位起动在 COM 服务器中驻留的可执行程序所需要的路径和参数。当起动该可执行程序时，其利用 COM API CoRegisterClassObject() 向 CSCM 注册其支持的所有 COM 服务器的所有 CLSID。如果所请求的 CLSID 被注册，CSCM 返回对该 COM 服务器的引用到调用者。COM 客户端和 COM 服务器之间的所有后续交互独立于 CSCM 发生。

[0315] 先前描述的隔离环境允许具有相同 CLSID 的 COM 服务器的多个实例安装在计算机上，每一个在不同的隔离范围中（其中是系统范围的不超过一个）。但是，仅此将不会使那些 COM 服务器对于 COM 客户端可用。

[0316] 图 14 描述为虚拟化对 COM 服务器访问所采取步骤的一个实施例。在概观上，为每个在隔离范围中起动的进程外 COM 服务器创建新的 CLSID，之后称作隔离 CLSID（或 ICLSID）（步骤 1402）。通过定义，这就是 CLSID，且因此必须在所有其它 CLSID 中是唯一的，换句话说，其必须具有 GUID 的属性。创建将所述对 (CLSID，应用隔离范围）映射到 ICLSID 的映射表。为 ICLSID 创建 COM 服务器注册表条目，其描述如何用起动参数起动 COM 服务器，起动参数启动在适当应用隔离范围中可执行的 COM 服务器（步骤 1404）。由 COM 客户端对诸如 CoCreateInstance() 或 CoCreateInstanceEx() 的 COM API 的调用被钩住或截取（步骤 1406）。如果确定 (a) 请求通过进程中 COM 服务器满足或 (b)COM 客户端和 COM 服务器都不与任何隔离范围关联，则将请求不经修改地传递到原始 COM API 且将结果返回给调用者（步骤 1408）。要使用的 COM 服务器的适当实例被识别（步骤 1410）。如果所选择的 COM 服务器实例在应用隔离环境中，则利用上面描绘的数据结构来确定其 ICLSID。否则，使用请求中的 CLSID（步骤 1412）。如果在步骤 1412 中识别 ICLSID，则利用 ICLSID 调用原始

的 CoCreateInstance() 或 CoCreateInstanceEx() 函数。这将把请求传递给 CSCM(步骤 1414)。CSCM 通过为确定起动参数在注册表中查找所请求的 CLSID 来寻找并且起动可以正常方式执行的 COM 服务器。如果 ICLSID 被请求,则在步骤 1404 中描述的 ICLSID 系统范围注册表条目被找到且在适当的应用隔离范围中起动 COM 服务器(步骤 1416)。所起动的 COM 可执行程序用其所支持的 COM 服务器的 CLSID 来调用所钩住的 CoRegisterClassObject() API,并且将其转换为适当的 ICLSID,将该 ICLSID 传递到原始 CoRegisterClassObject() API(步骤 1418)。当 CSCM 从具有所期望的 ICLSID 的 CoRegisterClassObject() 调用接收响应时,其将对 COM 服务器实例的引用返回给调用者(步骤 1420)。

[0317] 仍然参考图 14 且更详细地,为每个在隔离范围中起动的进程外 COM 服务器创建 ICLSID(步骤 1402)。在一些实施例中,在安装 COM 服务器期间创建 ICLSID。在其它实施例中,在安装之后立即创建 ICLSID。在又其它实施例中,在将 COM 服务器起动到隔离范围中之前创建 ICLSID。在所有这些实施例中,可通过钩住或截取创建或询问注册表数据库中的 CLSID 条目的系统请求来创建 ICLSID。可替换地,可通过钩住或截取创建 COM 服务器实例的 COM API 调用,比如 CoCreateInstance() 或 CoCreateInstanceEx() 来创建 ICLSID。可替换地,可在已经进行安装之后观察注册表数据库的 CLSID 特定部分的变化。

[0318] 创建将所述对(CLSID,应用隔离范围)映射到 ICLSID 的映射表,以及具有该 ICLSID 的 COM 服务器的适当注册表条目,ICLSID 描述如何用起动参数来起动 COM 服务器,起动参数启动在适当应用隔离范围中可执行的 COM 服务器(步骤 1404)。在许多实施例中,该表存储在永久存储器元件中,比如硬盘驱动器或固态存储器元件。在其它实施例中,该表可存储在注册表、平面文件、数据库或易失性存储器元件中。在又其它实施例中,例如通过将特定于该目的的新子键添加到由 CLSID 识别的每个适当 COM 服务器条目,来在整个注册表数据库的 COM 特定部分分布所述表。可在安装期间或在安装之后立即通过钩住或截取创建注册表数据库中 CLSID 条目的调用、或者通过观察在安装进行之后注册表数据库的 CLSID 特定部分的变化、或者通过钩住或截取创建 COM 服务器实例的 COM API 调用,比如 CoCreateInstance() 或 CoCreateInstanceEx() 来创建该表中的条目。COM 服务器到特定隔离范围中的安装可以被永久地记录。可替换地,特殊 COM 服务器和隔离范围到 ICLSID 的映射可被动态地创建且作为条目存储在非永久数据库中,或注册表数据库中。

[0319] COM 客户端对 COM API 的调用,诸如 CoCreateInstance() 或 CoCreateInstanceEx() 被钩住或截取(步骤 1406)。如果确定 (a) 请求通过进程中 COM 服务器满足或 (b)COM 客户端和 COM 服务器驻留在系统范围中(步骤 1407),则将请求不经修改地传递到原始 COM API 且将结果返回给调用者(步骤 1408)。

[0320] 如果请求不能通过进程中 COM 服务器来满足且 COM 客户端或者 COM 服务器不驻留在系统范围中(步骤 1407),则识别要使用的 COM 服务器的适当实例(步骤 1410)。对于 COM 客户端在特殊隔离范围中执行的实施例,优选将 COM 服务器安装在相同的应用隔离范围中,随后将它们安装在系统范围中(可能在客户端的应用隔离范围中执行),随后将 COM 服务器安装到其它应用隔离范围中。在这些实施例的一些中,安装在系统范围中的 COM 服务器可在相同的应用隔离范围中作为 COM 客户端执行。这由规则引擎和管理设置控制以允许这对于以该模式正确执行的 COM 服务器发生,并对于不如此执行的 COM 服务器不发生。对于 COM 客户端在系统范围中执行的实施例,优选系统范围 COM 服务器,随后是隔离范围中的

COM 服务器。COM 客户端可规定 COM 服务器在创建 COM 服务器的实例的调用中使用。可替换地，配置存储器可存储识别要实例化的 COM 服务器的信息。在一些实施例中，规定的 COM 服务器在另一计算机上驻留，该计算机是分离的、物理机器或虚拟机。在上面结合步骤 1404 所述的映射表可用于寻找可应用的 COM 服务器集合且（如果需要）基于规则来计算优选。

[0321] 对于可应用的 COM 服务器存在于另一计算机上的实施例，可向在相同远程计算机上执行的服务或守护程序询问要使用的 ICLSID。如果 COM 客户端挂钩确定远程 COM 服务器被要求，则 COM 客户端挂钩首先询问服务或守护程序以确定要使用的 CLSID/ICLSID。服务或守护程序确定与在请求中给定的 CLSID 相对应的 ICLSID。在一些实施例中，可基于管理员定义的配置数据、包含在规则引擎中的规则、或内置的硬编码逻辑来选择或创建由服务或守护程序返回的 ICLSID。在其它实施例中，请求可规定服务器上要使用的隔离范围。在又其它实施例中，所请求的 COM 服务器可与服务器的系统范围相关联，在该情况下，与 COM 服务器关联的 CLSID 被返回。在又其它实施例中，所请求的 COM 服务器可与服务器的隔离范围之一相关联，在该情况下，其返回与 COM 服务器的实例和隔离范围关联的 ICLSID。在一些实施例中，上述的服务或守护程序可用于支持起动本地的进程外 COM 服务器。

[0322] 如果所选择的 COM 服务器实例在本地计算机的应用隔离环境中，则利用上面结合步骤 1404 描述的数据结构来确定其 ICLSID。相反，如果所选择的 COM 服务器实例在本地计算机上的系统范围中，则使用请求中的 CLSID（步骤 1412）。在这些实施例的一些中，可动态创建利用 ICLSID 的 COM 服务器的条目。

[0323] 如果返回 ICLSID，则将其传递到原始 COM API 以替换原始的 CLSID。例如，所确定的 ICLSID 可被传递到原始的 CoCreateInstance() 或 CoCreateInstanceEx() 函数，该函数将请求传递给 CSCM（步骤 1414）。对于 COM 服务器在另一计算机上驻留的实施例，CSCM 将 ICLSID 传递给承载 COM 服务器的计算机，其中该计算机的 CSCM 处理 COM 服务器的起动。

[0324] CSCM 通过为确定起动参数在注册表中查找所请求的 CLSID 或 ICLSID 来寻找并且起动可以正常方式执行的 COM 服务器。如果 ICLSID 被请求，则在步骤 1404 中描述的 ICLSID 系统范围注册表条目被找到且在适当的应用隔离范围中起动 COM 服务器（步骤 1416）。

[0325] 如果所起动的 COM 服务器实例在应用隔离范围中执行（无论是安装在该范围中还是安装在系统范围中），钩住或截取 COM 服务器实例的 CoRegisterClassObject() 的 COM API 函数。利用如在步骤 1404 中定义的映射表，将传递到 CoRegisterClassObject() 的每个 CLSID 映射到对应的 ICLSID，用该 ICLSID 来调用原始的 CoRegisterClassObject()API（步骤 1418）。

[0326] 当 CSCM 从具有所期望的 ICLSID 的 CoRegisterClassObject() 调用接收响应时，其将对 COM 服务器实例的引用返回给调用者（步骤 1420）。

[0327] 当 COM 客户端和 COM 服务器在应用隔离范围（包括不同范围）和系统范围的任何组合中执行时，该技术支持 COM 服务器的执行。ICLSID 特定于服务器（由 CLSID 识别）和所期望的适当隔离范围的组合。客户端只需要确定正确的 ICLSID（或者，如果服务器安装在系统范围中且在其中执行，则只确定原始 CLSID）。

[0328] 4.6 虚拟化的文件类型关联关系 (FTA)

[0329] 文件类型关联关系是已知的图形用户接口技术，用于调用应用程序的执行。向用户呈现表示数据文件的图形图标。用户利用键盘命令或利用指针设备，诸如鼠标来选择数

据文件，且在图标上点击或双击以指明用户想要打开文件。可替换地，在一些计算环境中，用户在命令行提示而不是以命令来输入到文件的路径。文件通常具有关联的文件类型指示，该指示用于确定在打开文件时要使用的应用程序。这通常是利用将文件类型指示映射到特定应用的表来完成的。在微软 WINDOWS 操作系统家族的许多成员中，该映射通常以元组存储在注册表数据库中，元组包括识别要执行应用的文件类型指示符和完整路径名，并且只有一个应用程序可与任何特殊文件类型关联。

[0330] 在所述的隔离环境中，可在单个计算机上安装和执行应用的多个版本。因此，在这些环境中，文件类型和关联的应用程序之间的关系不再是一对一的关系，而是一对多的关系。类似问题对于 MIME 附件类型也存在。在这些环境中，该问题是通过在选择给定文件类型时替换识别要起动的应用程序的路径名来解决的。路径名用选择器工具的路径名来替换，选择器工具给予用户对要起动的应用程序的选择。

[0331] 现在参考图 15，且在概观上，截取将文件类型关联数据写到配置存储器的请求（步骤 1502）。确定该请求是否正在更新配置存储器中的文件类型关联信息（步骤 1504）。如果不是，即如果条目已经存在，则没有更新发生（步骤 1506）。否则，利用上面在部分 4.1.4 或 4.2.4 中描述的虚拟化技术来创建新条目，或更新已有的条目（步骤 1508）。为适当的隔离范围虚拟化的新的或更新的条目将文件类型映射到选择器工具，选择器工具允许用户选择在观看或编辑文件时使用多个应用程序中的哪个。

[0332] 仍然参考图 15 且更详细地，截取将文件类型关联数据写到配置存储器的请求（步骤 1502）。在一些实施例中，配置存储器是 WINDOWS 注册表数据库。将数据写到配置存储器的请求可由用户模式挂钩函数、内核模式挂钩函数、文件系统过滤器驱动程序或微型驱动程序来截取。

[0333] 确定该请求是否试图更新配置存储器中的文件类型关联信息（步骤 1504）。在一个实施例中，这是通过检测所截取的请求是否指明其试图修改配置存储器来完成的。在另一个实施例中，请求的目标与包括在请求中的信息相比较，以确定请求是否正在试图修改配置存储器。对于配置存储器是注册表数据库的实施例，修改注册表的请求被截取，如上面部分 4.2 所述。

[0334] 确定请求不正在试图更新配置存储器，则没有更新发生（步骤 1506）。在一些实施例中，确定不试图更新配置存储器，因为所截取的请求是读取请求。在其它实施例中，当配置存储器中的目标条目和包括在所窃取请求中的信息相同或基本相同时，可做出该确定。

[0335] 但是，如果在步骤 1504 确定请求试图更新配置存储器，则在配置存储器中创建新条目，或者更新已有的条目（步骤 1508）。在一些实施例中，规则确定在哪个隔离范围中创建或更新条目。在一些实施例中，在系统范围或应用隔离范围中创建新条目或更新已有条目。在许多实施例中，在适当的用户隔离范围中创建新条目或更新已有条目。如果创建新条目，则其不是去识别在所截取请求中识别的应用程序，而是将选择器应用作为在特殊类型的文件被访问时要使用的应用列出。在一些实施例中，当安装应用程序的新版本时，或者在处理相同文件类型的另一应用被安装时，或者在应用为处理特殊类型的文件而注册或注销其本身时，选择器工具被自动更新。在一些实施例中，如果选择器工具在用户范围中执行，则选择器工具可合并其任何应用所注册的适当应用的列表，以处理在其它范围，比如在系统范围和应用范围中维护的部分配置存储器中的相同文件类型。如果更新已有条目，且

已有条目已经将选择器应用作为要在使用该特殊文件类型的文件时使用的应用列出，则由选择器对该文件类型呈现的应用列表可被更新以包括更新的应用。如果现有条目被更新，但其没有列出选择器应用，则更新的条目被用来将选择器应用作为要在使用该特殊文件类型的文件时使用的应用列出。在这些实施例中，与关联的应用有关的信息可存储在关联的配置文件中，或在一些实施例中，作为注册表数据库中的条目存储。

[0336] 选择器应用可向用户呈现与所选文件类型关联的应用的列表。应用还可允许用户选择用户想要用来处理文件的应用程序。选择器接着在适当范围：系统范围、应用隔离范围、或用户隔离范围中起动该应用程序。在一些实施例中，选择器工具维护与文件类型关联的缺省应用程序的标识。在这些实施例中，缺省应用可由不访问桌面或配置为使用缺省句柄的进程使用，而不用向用户提供选择。

[0337] 4.7 隔离环境之间进程的动态移动

[0338] 本发明的附加方面是在不同虚拟范围之间移动运行进程的工具。换句话说，在应用执行的同时，将由隔离环境 200 呈现给应用实例的本地资源的聚集视图改变为不同的聚集视图。这允许在特殊隔离范围内已经隔离的进程在进程运行同时被“移动”到另一隔离范围。这对于一次只有一个实例被执行的系统服务或进程来说是尤其有用的，比如 WINDOWS 操作系统中的 MSI 服务。本发明的这个方面还可用于允许用户按顺序在若干隔离范围中工作。

[0339] 参考图 16，且在概观上，示出用于在一个隔离范围和第二个隔离范围之间，或者在系统范围和隔离范围之间移动进程的过程的一个实施例。如本说明书中所使用的，术语“目标隔离范围”将用于表示进程被移动到的隔离范围，包括系统范围，并且术语“源隔离范围”将用于表示所移动进程来自的隔离范围，包括系统范围。如图 16 所示，且在概观上，用于将进程移动到目标隔离范围的方法包括步骤：确保进程处于安全状态（步骤 1602）；在规则引擎中将进程的关联关系从其源隔离范围改变到目标隔离范围（步骤 1604）；对于任何过滤器驱动程序或挂钩将进程的关联关系从源隔离范围改变到目标隔离范围（步骤 1606）；且允许进程恢复执行（步骤 1608）。

[0340] 仍然参考图 16，且更详细地，进程在向不同的隔离范围移动时可处于“安全”状态（步骤 1602）。在一些实施例中，监视进程以确定其何时不处理请求。在这些实施例中，认为进程处于“安全”状态以便在进程没有处理请求时移动。在这些实施例的一些中，一旦认为进程处于“安全”状态，到进程的新请求被延迟，直到进程被移动。在其它实施例中，诸如在诊断应用方面，用户接口可被提供以触发隔离范围中的变化。在这些实施例中，用户接口可运行将要移动的进程放置到“安全”状态下的代码。在另外其它实施例中，管理程序可通过延迟到进程的所有进入的请求且等待进程完成任何活动请求的执行来强迫进程进入“安全”状态。

[0341] 如果与目标隔离范围关联的规则在规则引擎中已经不存在，则将它们加载到规则引擎中（步骤 1603）。

[0342] 在规则引擎中改变进程与源隔离范围的关联关系（步骤 1604）。如上所述，进程可与任何隔离范围关联。该关联关系可由规则引擎用在对虚拟本地资源的每个请求上，以确定要应用到该请求的规则。通过改变规则引擎中的适当数据结构可将应用实例与目标隔离范围关联。在一些实施例中，写入将进程与新隔离范围相关联的新数据库条目。在其它实

施例中，存储与进程关联的隔离范围的标识符的树节点被改写，以识别新的隔离范围。在另外其它实施例中，可做出操作系统请求以为进程分配附加的存储，从而存储与目标隔离范围关联的规则，或在一些实施例中，存储规则的标识符。

[0343] 无论关联关系或规则存储在规则引擎的外侧，比如过滤器驱动程序、内核模式挂钩或用户模式挂钩的任何之处，都可改变进程与源隔离范围的关联关系（步骤1606）。对于进程和隔离范围规则之间的关联关系是基于PID维护的实施例，进程PID和规则集合之间的关联关系被改变。对于PID不用于维护进程和可应用的隔离规则集合之间关联关系的实施例，用户模式挂钩函数可被改变以访问与目标隔离范围关联的规则集合。对于与隔离范围的规则集合关联的进程在规则引擎中维护的实施例，在步骤1604中改变存储在规则引擎中的关联关系就足够了。

[0344] 允许进程恢复在新隔离范围中的执行（步骤1610）。对于新请求被延迟或禁止作出新请求的实施例，那些请求被发给进程且新的请求被允许。

[0345] 在一个特别有用的方面中，上述的方法可用于虚拟化由微软提供且在微软WINDOWS操作系统家族的一些中可用的MSI、安装打包和安装技术。由用于安装的该技术打包的应用称为MSI插件。支持该技术的操作系统具有称为MSI服务的WINDOWS服务，该服务帮助安装MSI插件。在系统上存在该服务的单个实例。希望安装MSI插件的进程在它们的会话中运行MSI进程，它们的会话对MSI服务作出COM调用。

[0346] MSI安装可被虚拟化以将MSI插件安装到应用隔离环境中。概念上，这是通过钩住或截取在MSI服务的安装会话中对MSI API做出的调用来实现的。可使用互斥体以确保一次只进行一个安装。当接收或截取请求启动新安装的对MSI API的调用，且调用进程与特殊的应用隔离范围相关联时，在允许该调用进行之前，将MSI服务放置在该隔离范围的上下文中。在MSI服务执行其正常安装动作时安装进行，尽管根据可应用的隔离范围来虚拟化MSI服务作出的本地资源请求。当检测到安装进程结束时，MSI服务与隔离范围之间的关联关系被移除。尽管上述参考MSI来描述，所描述的技术可应用于其它安装技术。

[0347] 等效性

[0348] 本发明可作为在一个或多个制造物品中体现的一个或多个计算机可读程序来提供。制造物品可以是软盘、硬盘、CD-ROM、闪速存储器卡、PROM、RAM、ROM或磁带。一般地，计算机可读程序可用任何程序设计语言，LISP、PERL、C、C++、PROLOG或任何字节码语言，比如JAVA来实现。软件程序可作为目标代码存储在一个或多个制造物品之上或之中。

[0349] 已经描述了本发明的某些实施例，现在对于本领域技术人员清楚的是，还可使用合并了本发明概念的其它实施例。因此，本发明不应当限于某些实施例，而只应当由随后权利要求的精神和范围来限定。

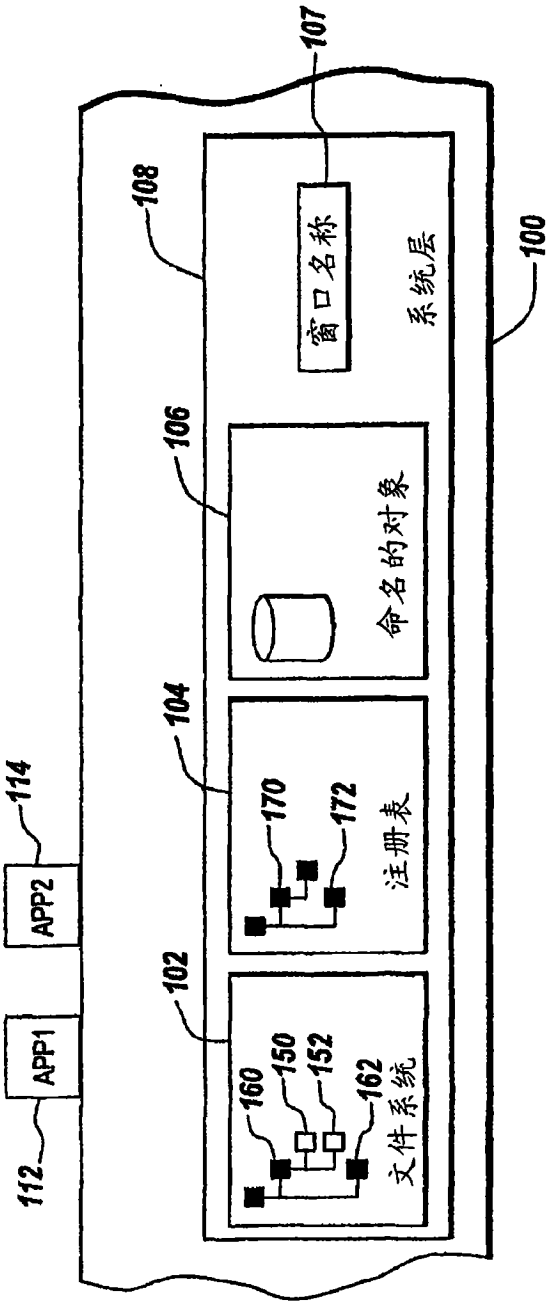

107
108
100
窗口名称
系统层
106
命名的对象
104
170
172
注册表
102
160
150
152
162
文件系统
APP2
114
APP1
112

图 1A
(现有技术)

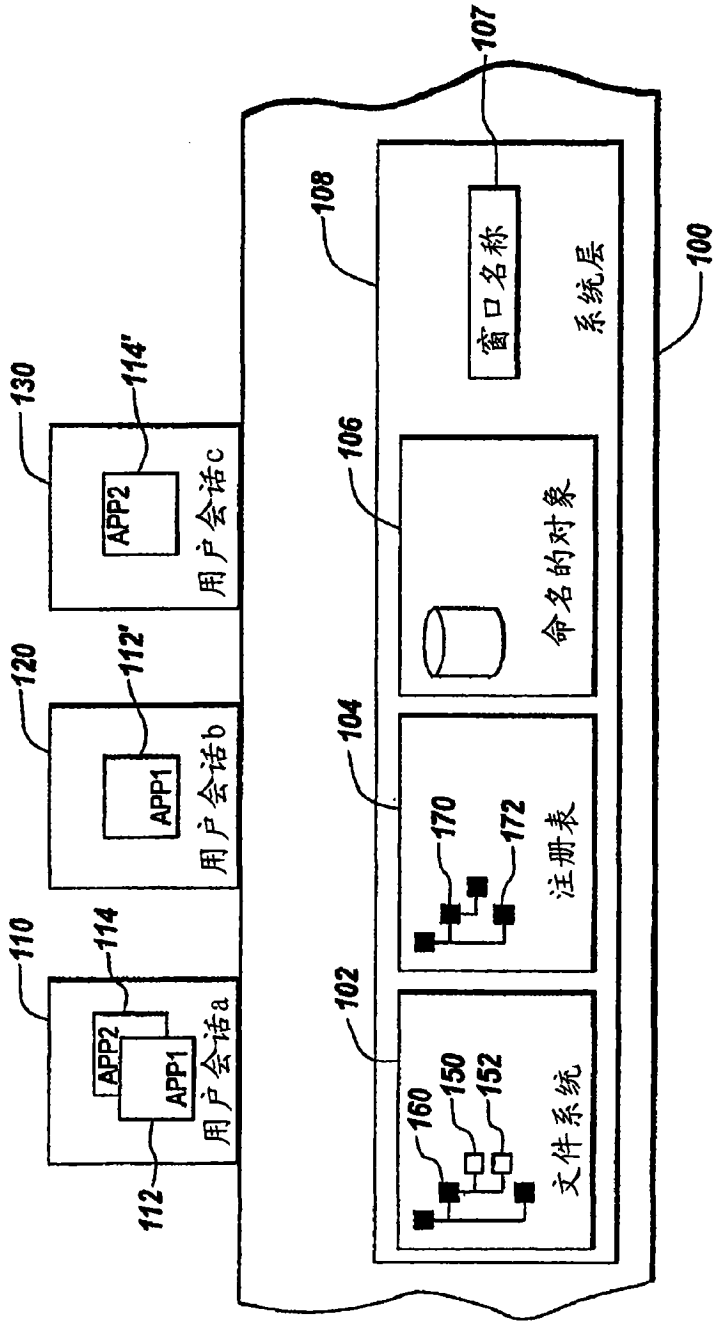

100
107
108
窗口名称
系统层
106
命名的对象
104
170
172
注册表
102
160
150
152
文件系统
130
114'
APP2
用户会话c
120
112'
APP1
用户会话b
110
114
APP2
APP1
用户会话a
112

图 1B
(现有技术)

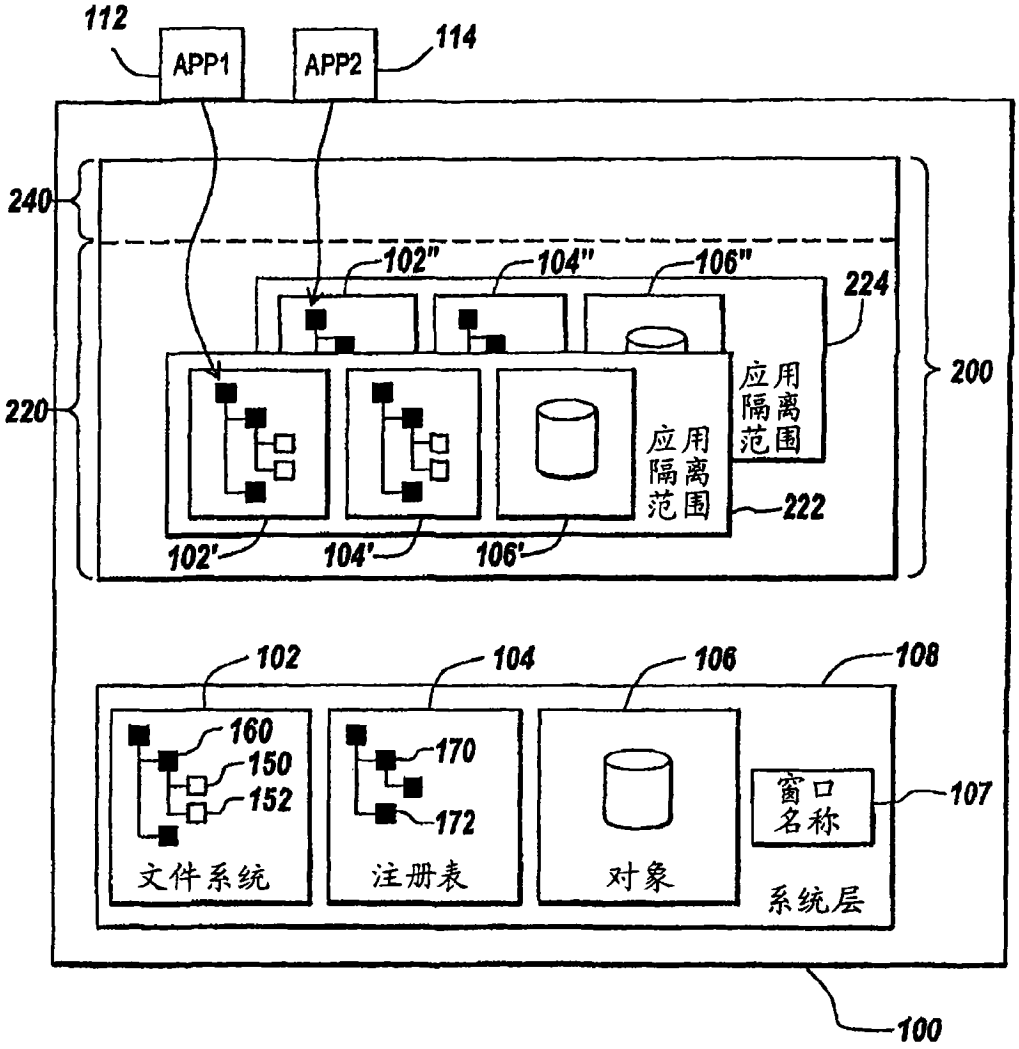

112
APP1
APP2
114
240
102"
104"
106"
224
220
200
应用隔离范围
应用隔离范围
222
102'
104'
106'
102
104
106
108
160
150
152
170
172
窗口名称
107
文件系统
注册表
对象
系统层
100

图 2A

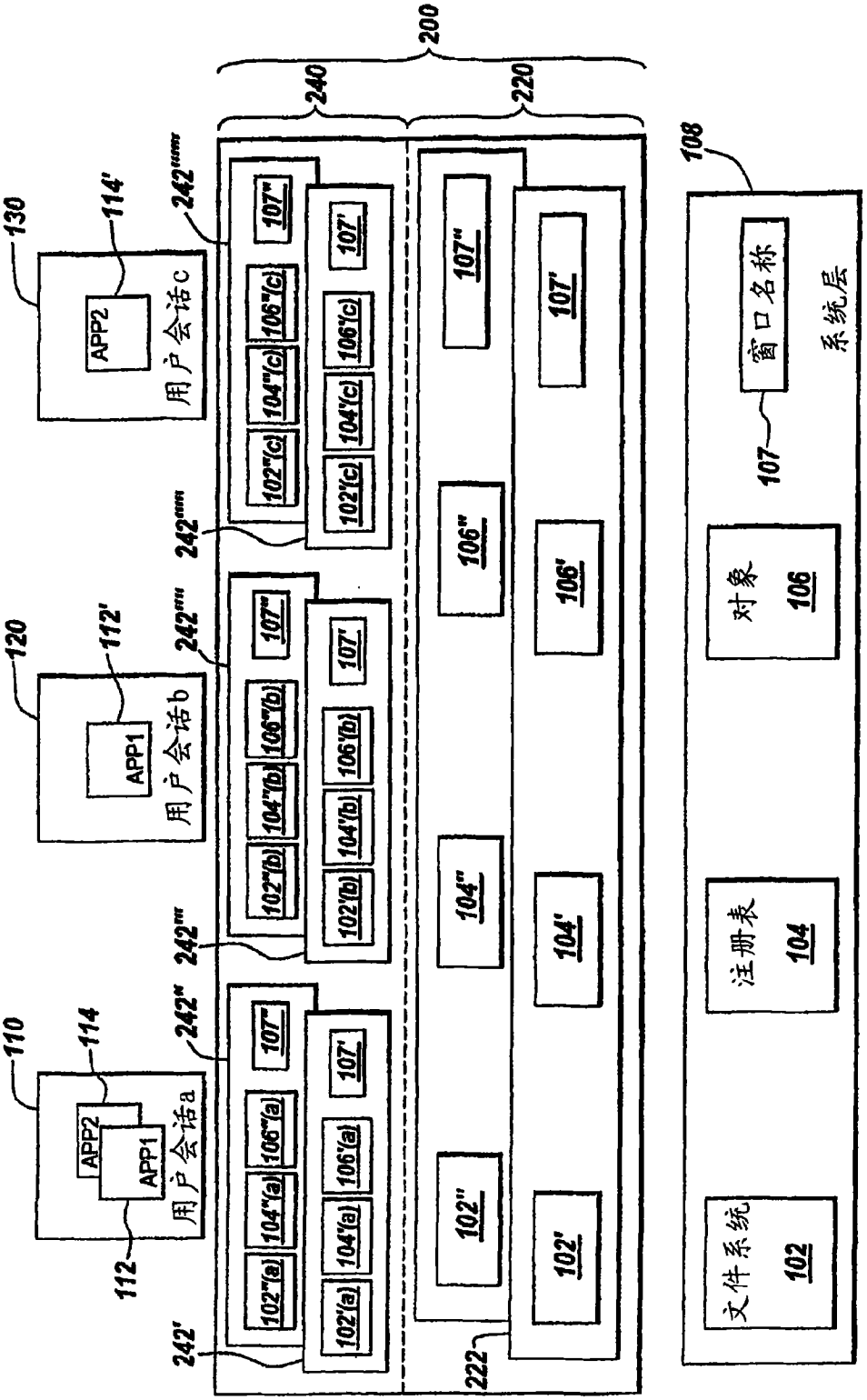

200
240
220
108
130
114'
APP2
用户会话c
242''''
120
112'
APP1
用户会话b
242'''
110
114
APP2
APP1
用户会话a
242''
112
242'
222
102''(a)
104''(a)
106''(a)
107''
102'(a)
104'(a)
106'(a)
107'
102''(b)
104''(b)
106''(b)
107''
102'(b)
104'(b)
106'(b)
107'
102''(c)
104''(c)
106''(c)
107''
102'(c)
104'(c)
106'(c)
107'
102''
104''
106''
107''
102'
104'
106'
107'
文件系统
102
注册表
104
对象
106
窗口名称
107
系统层

图 2B

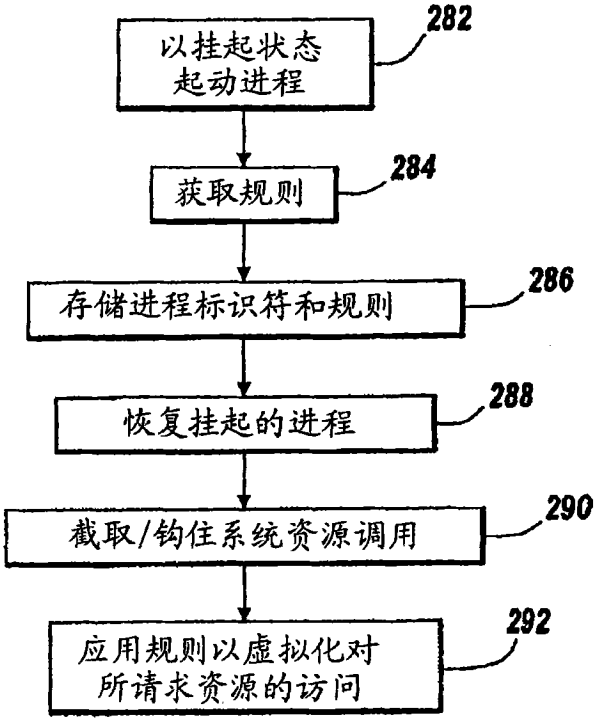

以挂起状态起动进程
282
获取规则
284
存储进程标识符和规则
286
恢复挂起的进程
288
截取/钩住系统资源调用
290
应用规则以虚拟化对所请求资源的访问
292

图 2C

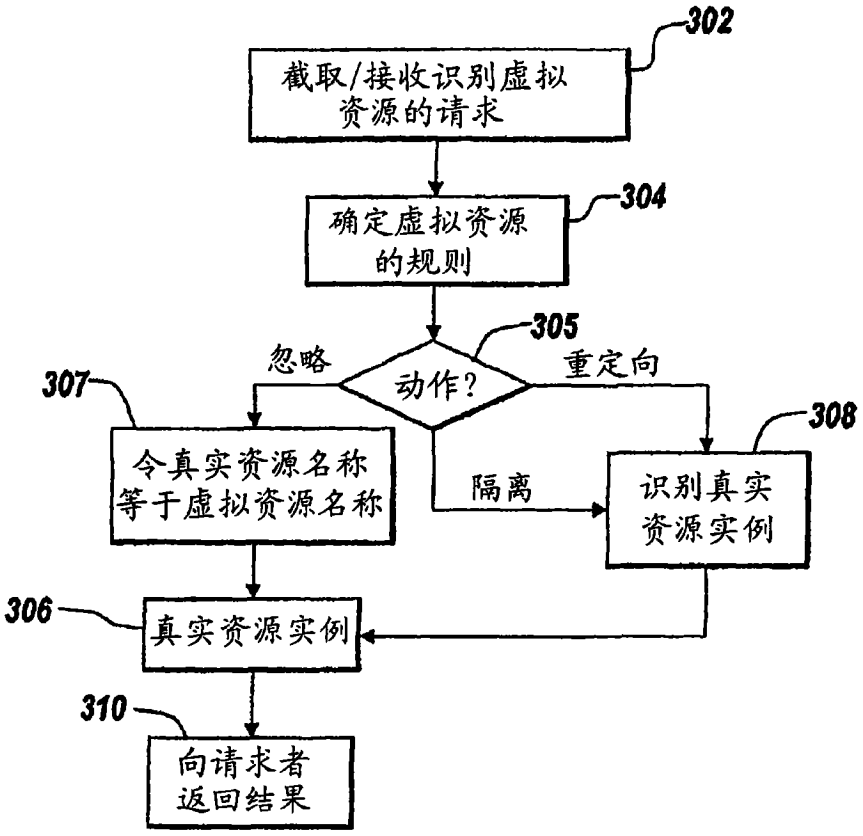

302
截取/接收识别虚拟资源的请求
304
确定虚拟资源的规则
305
动作?
忽略
重定向
隔离
307
令真实资源名称等于虚拟资源名称
308
识别真实资源实例
306
真实资源实例
310
向请求者返回结果

图 3A

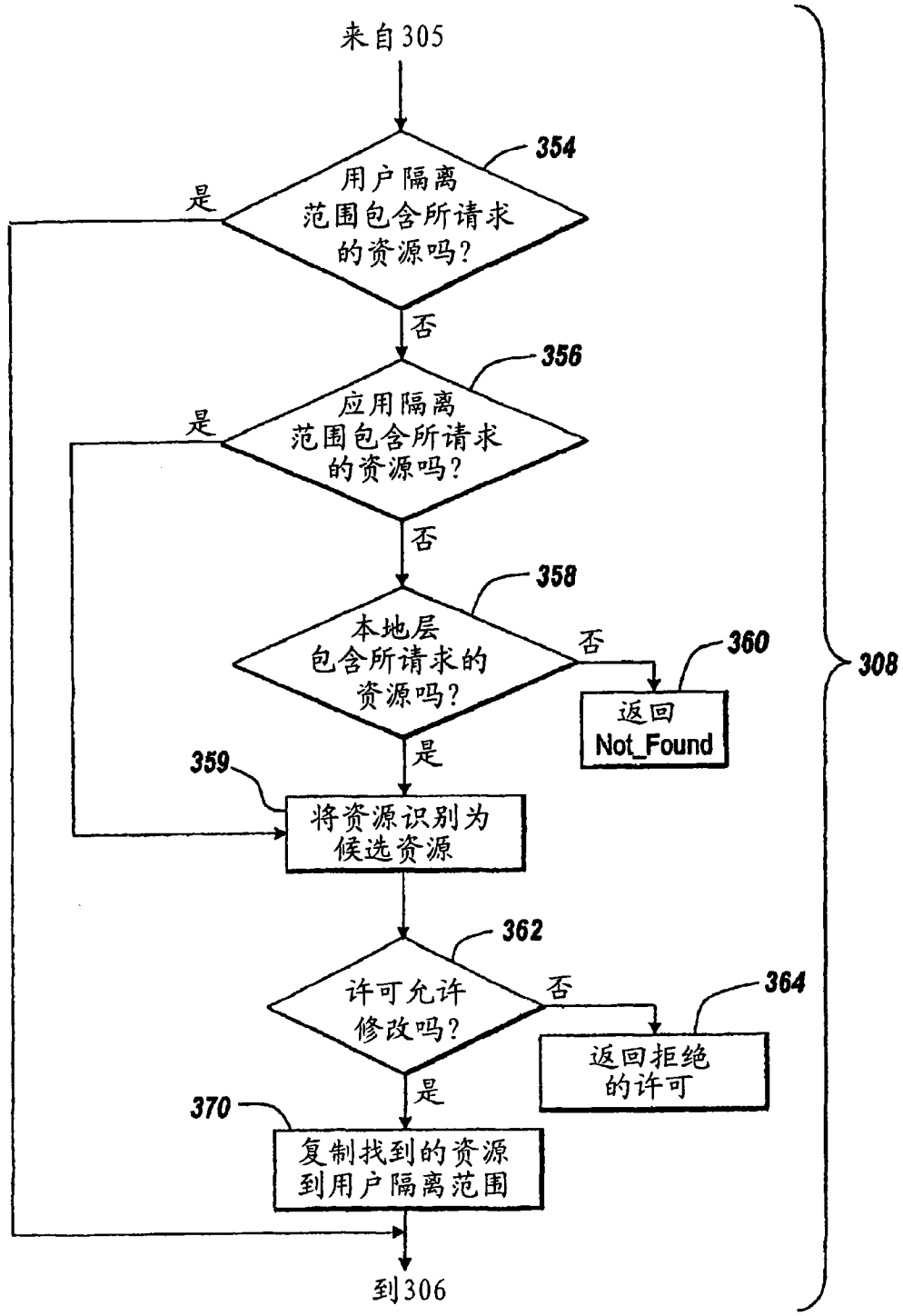

来自305
354
用户隔离范围包含所请求的资源吗?
是
否
356
应用隔离范围包含所请求的资源吗?
是
否
358
本地层包含所请求的资源吗?
否
360
返回Not_Found
是
359
将资源识别为候选资源
362
许可允许修改吗?
否
364
返回拒绝的许可
是
370
复制找到的资源到用户隔离范围
到306
308

图 3B

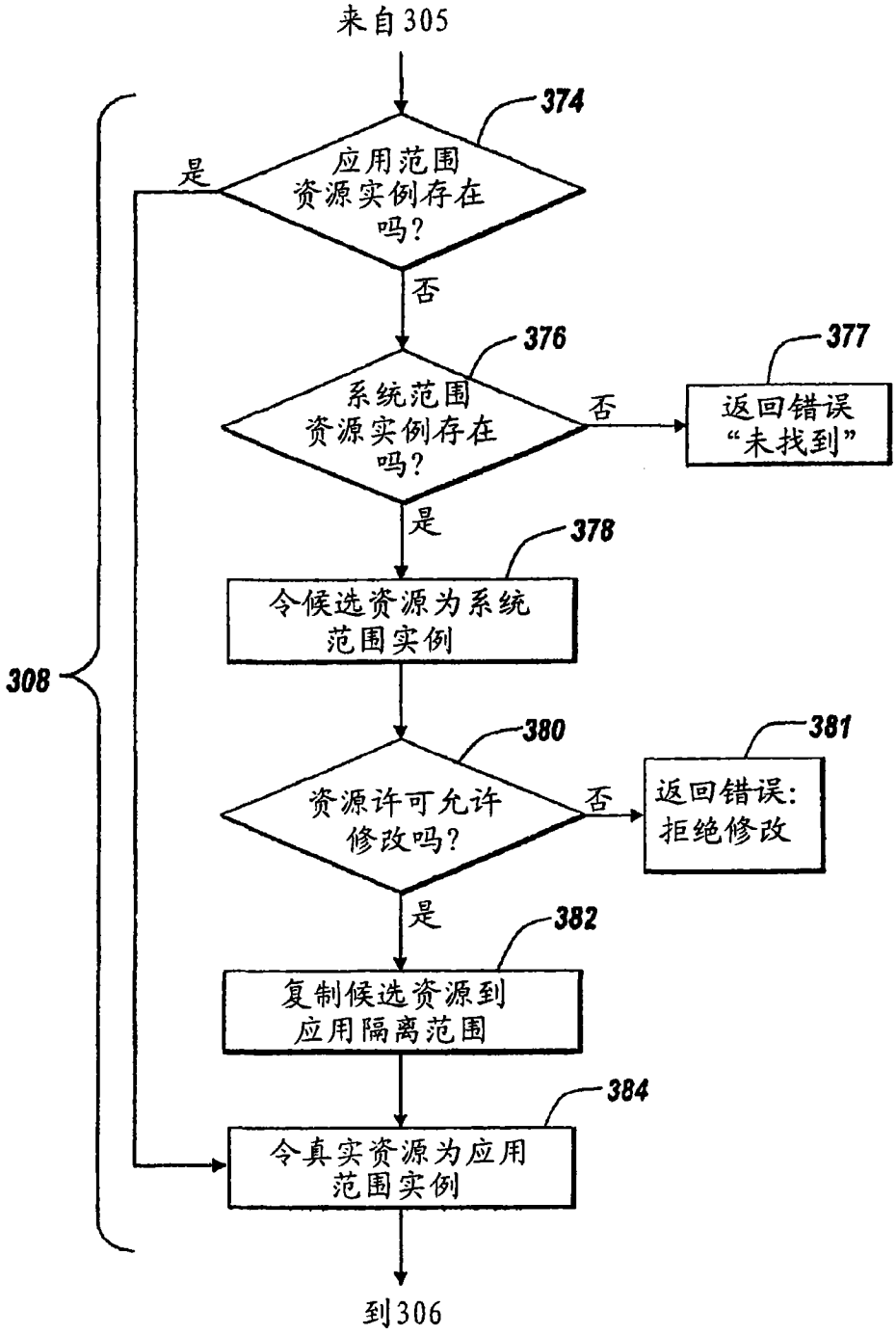

来自305
374
应用范围
资源实例存在
吗?
是
否
376
系统范围
资源实例存在
吗?
否
377
返回错误
“未找到”
是
378
令候选资源为系统
范围实例
308
380
资源许可允许
修改吗?
否
381
返回错误:
拒绝修改
是
382
复制候选资源到
应用隔离范围
384
令真实资源为应用
范围实例
到306

图 3C

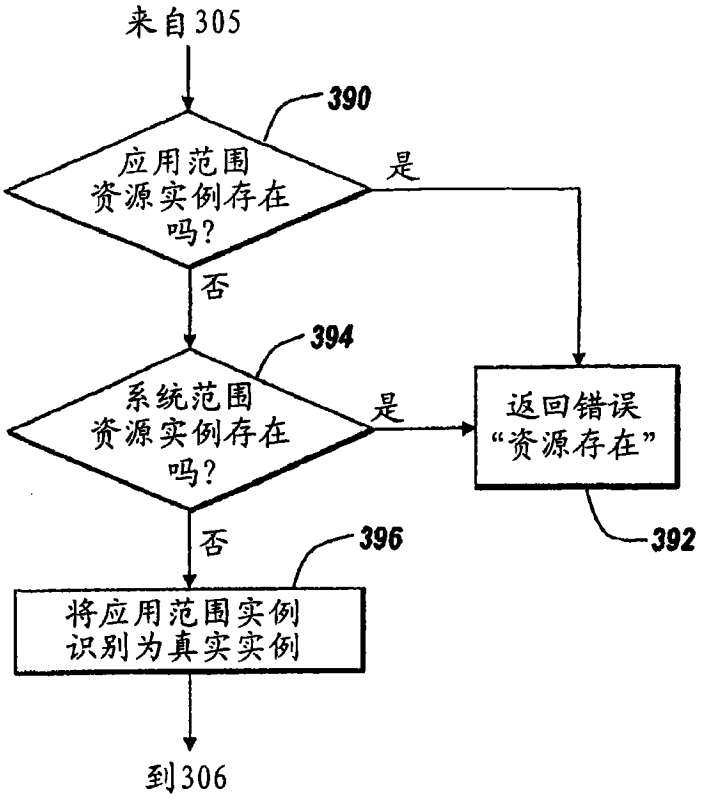

来自305
390
应用范围
资源实例存在
吗?
是
否
394
系统范围
资源实例存在
吗?
是
返回错误
“资源存在”
392
否
396
将应用范围实例
识别为真实实例
到306

图 3D

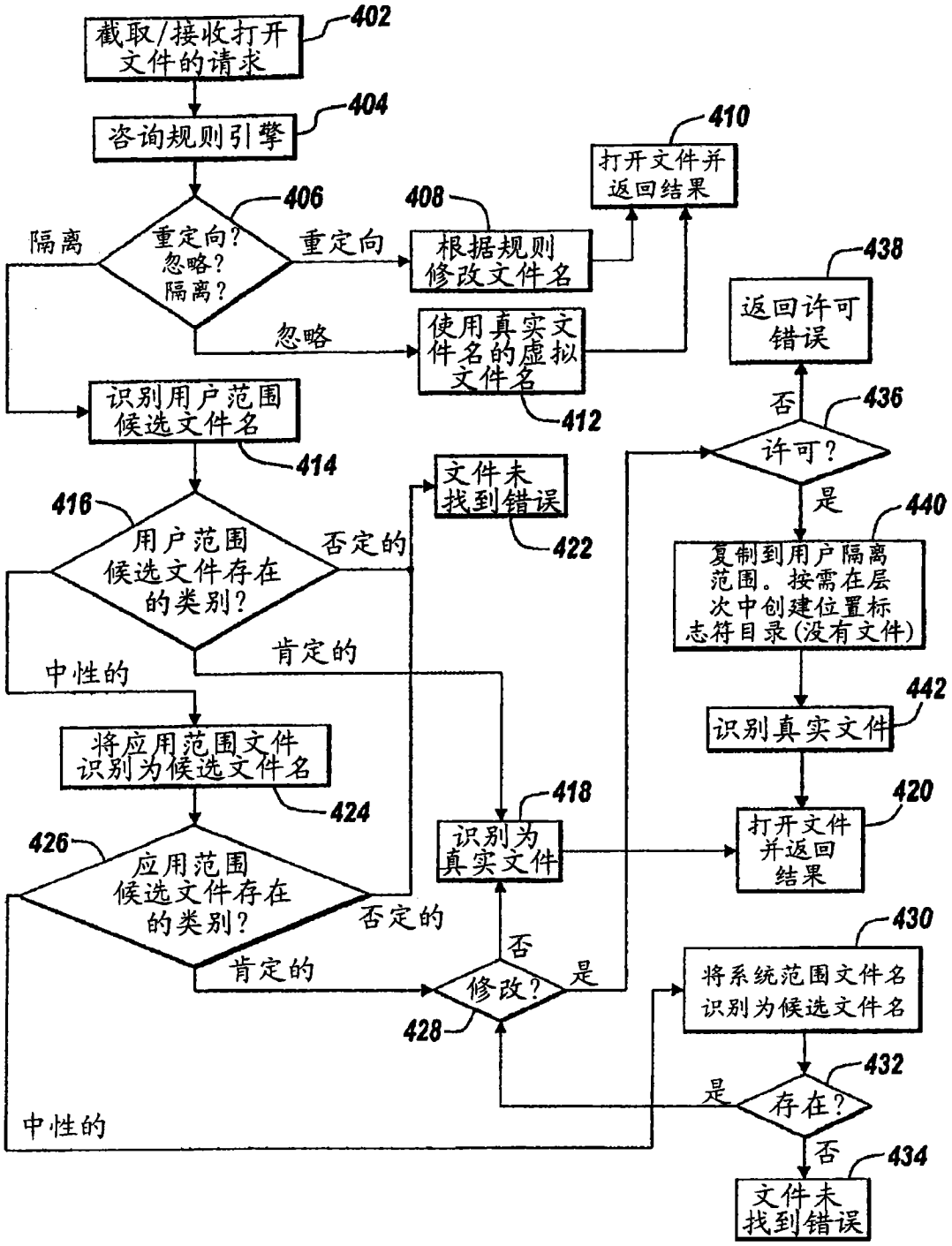

截取/接收打开文件的请求
402
咨询规则引擎
404
重定向?
忽略?
隔离?
406
隔离
重定向
根据规则修改文件名
408
打开文件并返回结果
410
忽略
使用真实文件名的虚拟文件名
412
识别用户范围候选文件名
414
文件未找到错误
422
用户范围候选文件存在的类别?
416
否定的
肯定的
中性的
将应用范围文件识别为候选文件名
424
应用范围候选文件存在的类别?
426
否定的
肯定的
识别为真实文件
418
修改?
428
否
是
返回许可错误
438
许可?
436
否
是
复制到用户隔离范围。按需在层次中创建位置标志符目录(没有文件)
440
识别真实文件
442
打开文件并返回结果
420
将系统范围文件名识别为候选文件名
430
存在?
432
是
否
文件未找到错误
434
中性的

图 4

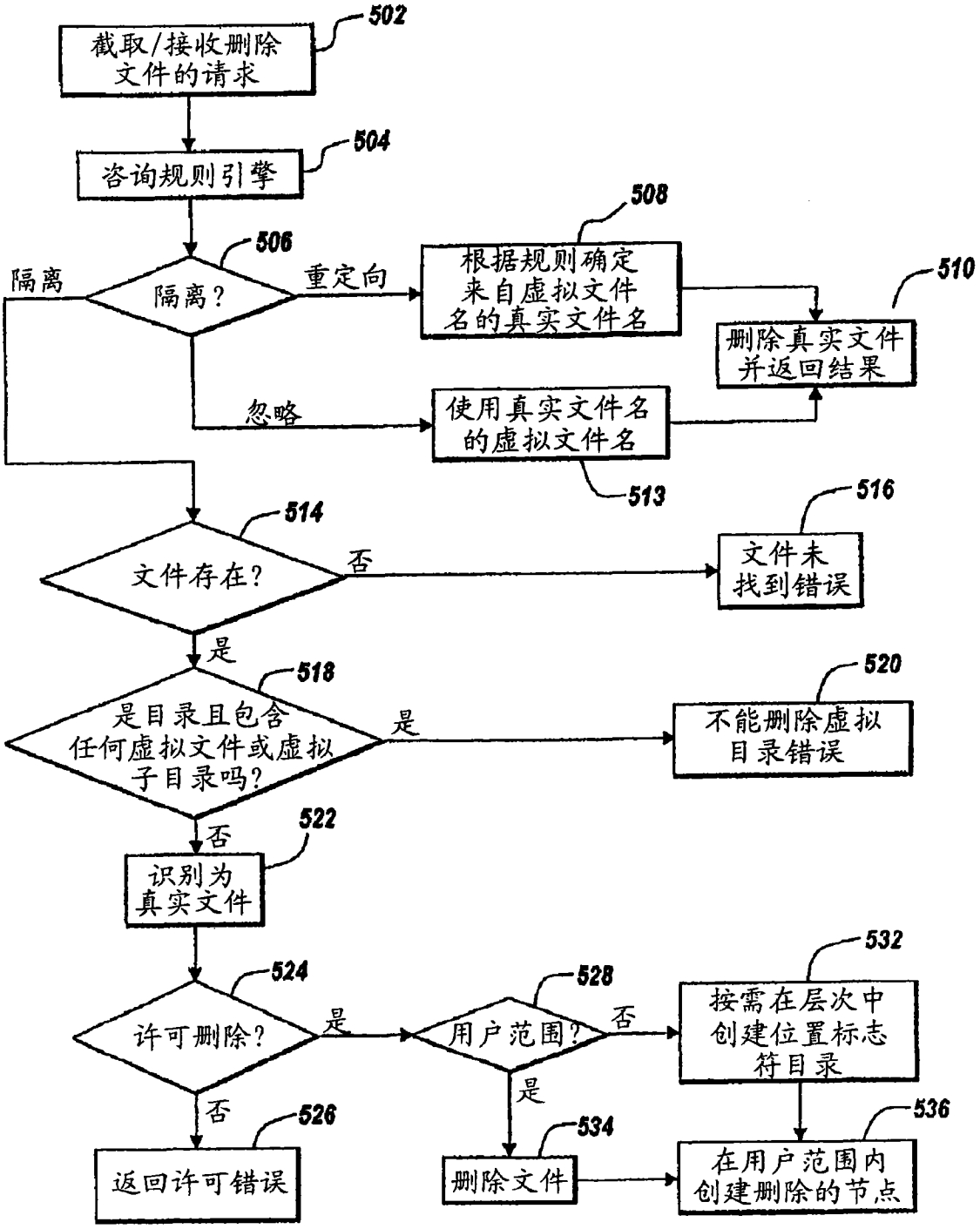

截取/接收删除文件的请求
502
咨询规则引擎
504
隔离？
506
隔离
重定向
根据规则确定来自虚拟文件名的真实文件名
508
删除真实文件并返回结果
510
忽略
使用真实文件名的虚拟文件名
513
文件存在？
514
否
文件未找到错误
516
是
是目录且包含任何虚拟文件或虚拟子目录吗？
518
是
不能删除虚拟目录错误
520
否
识别为真实文件
522
许可删除？
524
是
用户范围？
528
否
按需在层次中创建位置标志符目录
532
否
返回许可错误
526
是
删除文件
534
在用户范围内创建删除的节点
536

图 5

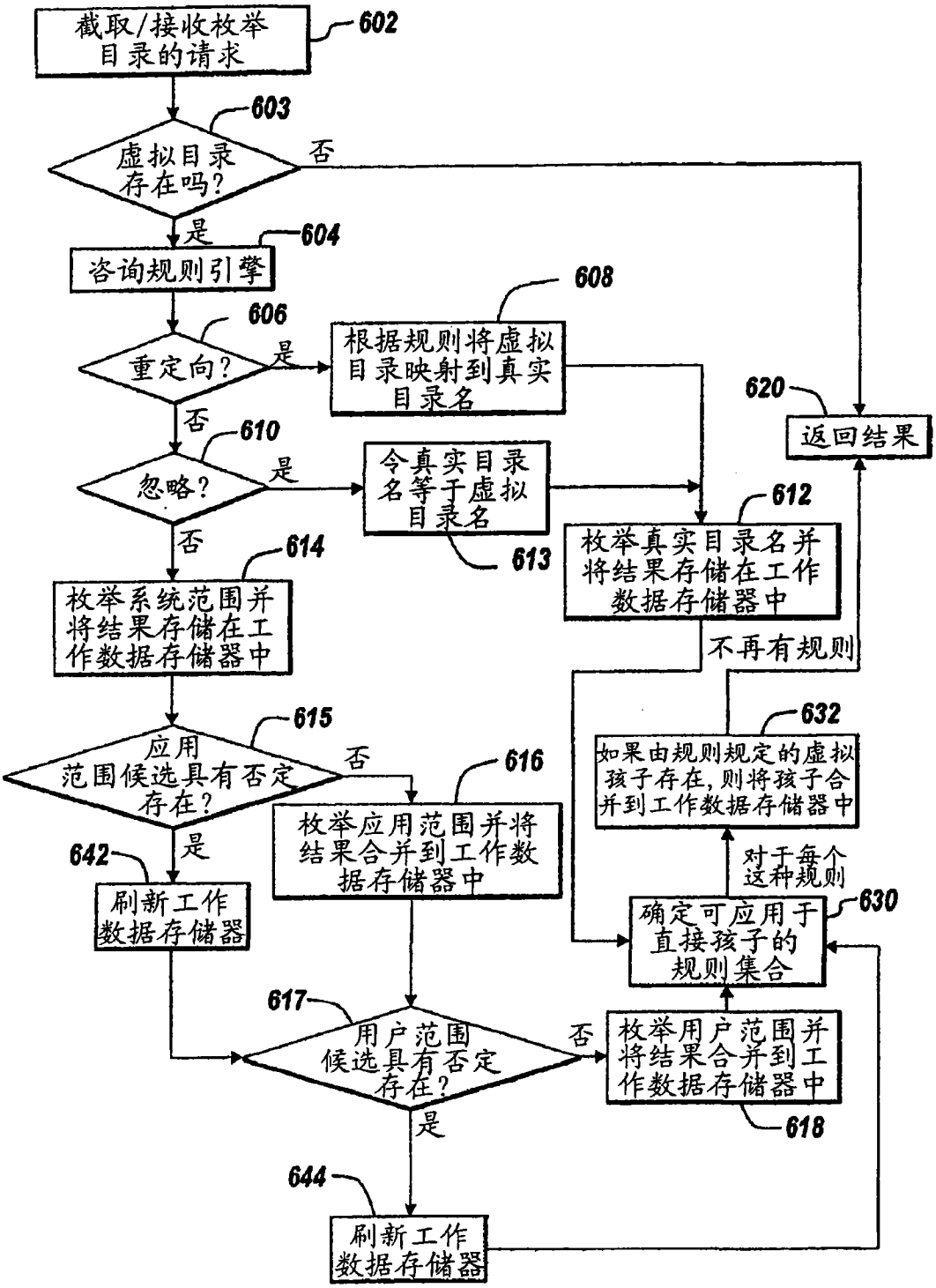

截取/接收枚举目录的请求
602
603
虚拟目录存在吗?
否
是
604
咨询规则引擎
606
重定向?
是
608
根据规则将虚拟目录映射到真实目录名
否
610
忽略?
是
令真实目录名等于虚拟目录名
613
620
返回结果
612
枚举真实目录名并将结果存储在工作数据存储器中
否
614
枚举系统范围并将结果存储在工作数据存储器中
不再有规则
615
应用范围候选具有否定存在?
否
616
枚举应用范围并将结果合并到工作数据存储器中
632
如果由规则规定的虚拟孩子存在,则将孩子合并到工作数据存储器中
是
642
刷新工作数据存储器
对于每个这种规则
630
确定可应用于直接孩子的规则集合
617
用户范围候选具有否定存在?
否
枚举用户范围并将结果合并到工作数据存储器中
618
是
644
刷新工作数据存储器

图 6

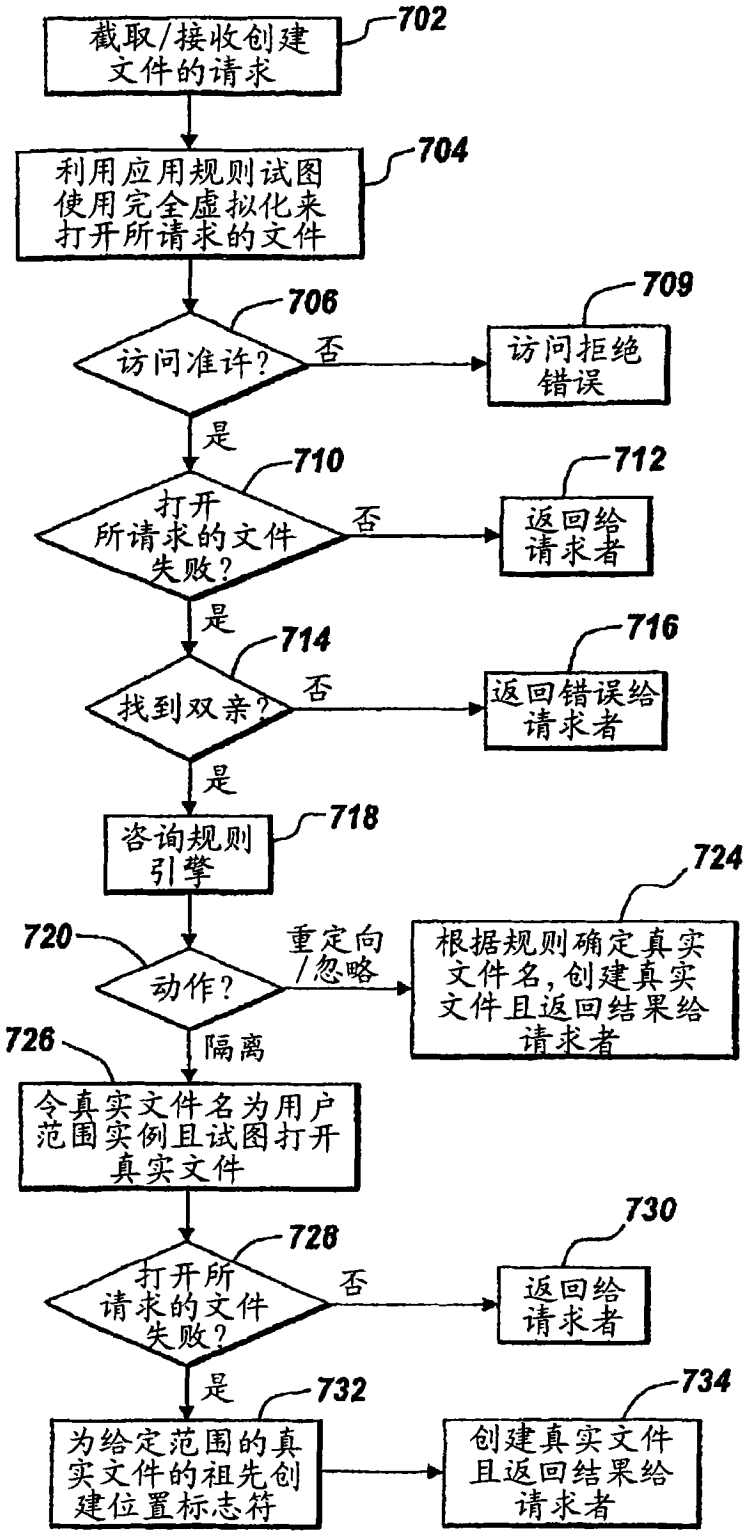

截取/接收创建文件的请求
702
利用应用规则试图使用完全虚拟化来打开所请求的文件
704
访问准许?
706
否
访问拒绝错误
709
是
打开所请求的文件失败?
710
否
返回给请求者
712
是
找到双亲?
714
否
返回错误给请求者
716
是
咨询规则引擎
718
动作?
720
重定向/忽略
根据规则确定真实文件名,创建真实文件且返回结果给请求者
724
隔离
令真实文件名为用户范围实例且试图打开真实文件
726
打开所请求的文件失败?
728
否
返回给请求者
730
是
为给定范围的真实文件的祖先创建位置标志符
732
创建真实文件且返回结果给请求者
734

图 7

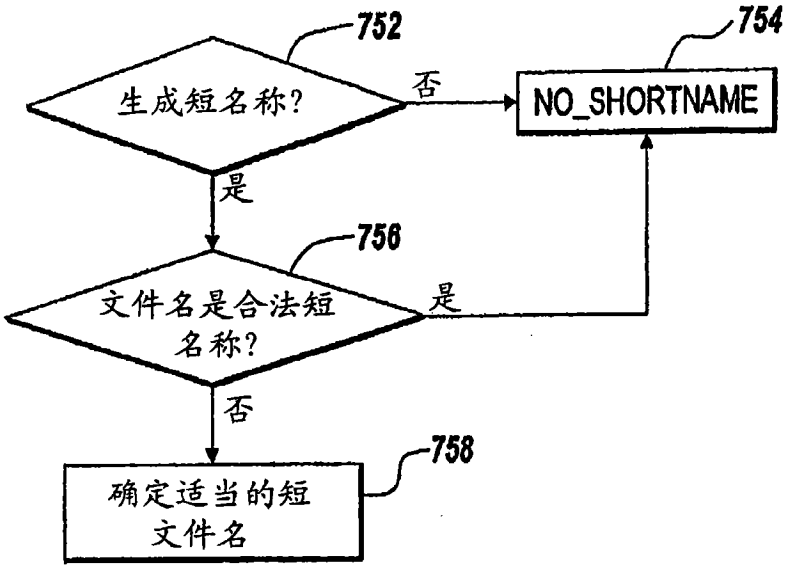

752
生成短名称?
否
754
NO_SHORTNAME
是
756
文件名是合法短名称?
是
否
758
确定适当的短文件名

图 7A

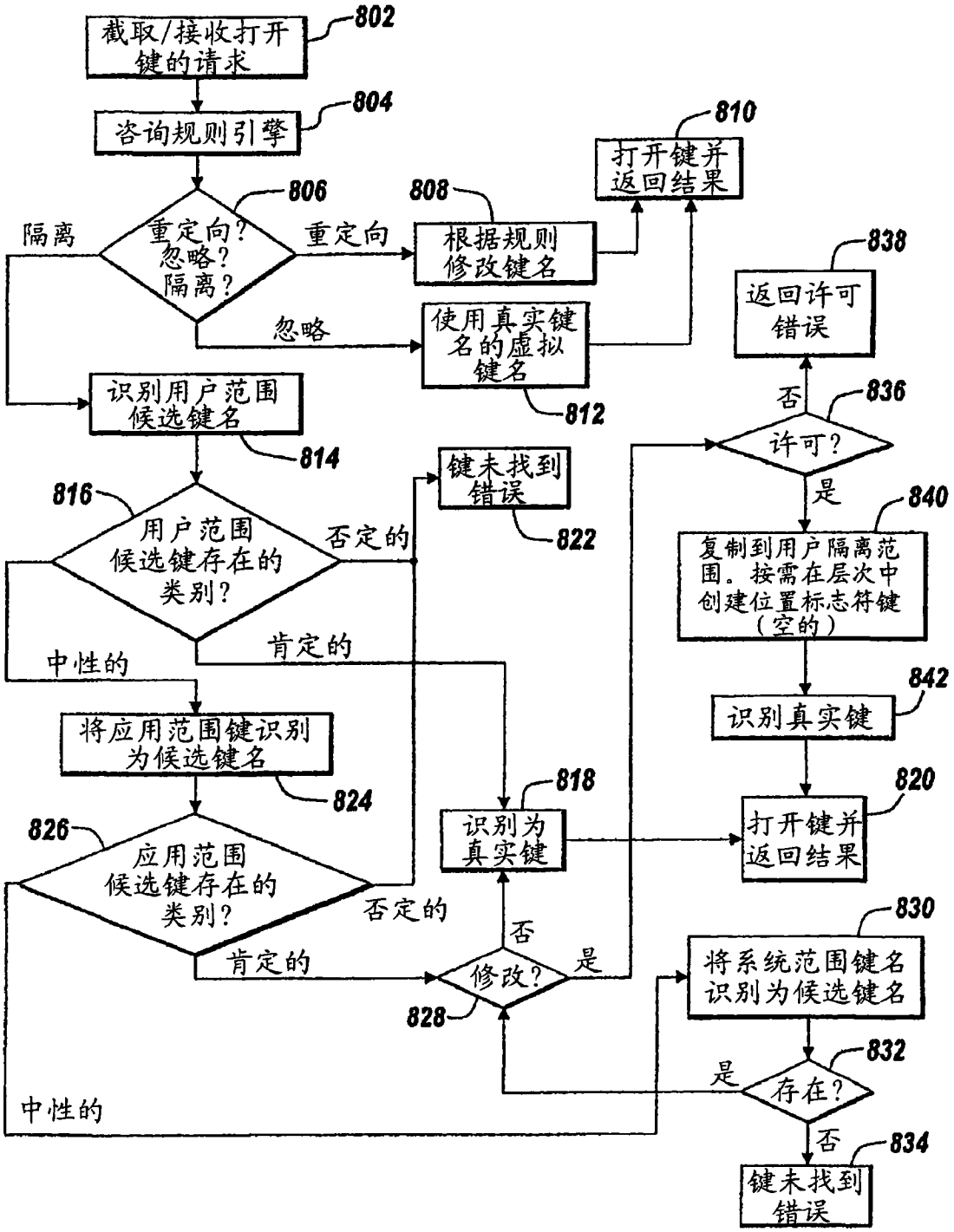

截取/接收打开键的请求
802
咨询规则引擎
804
重定向？忽略？隔离？
806
隔离
重定向
忽略
根据规则修改键名
808
打开键并返回结果
810
使用真实键名的虚拟键名
812
识别用户范围候选键名
814
用户范围候选键存在的类别？
816
否定的
肯定的
中性的
键未找到错误
822
将应用范围键识别为候选键名
824
应用范围候选键存在的类别？
826
否定的
肯定的
识别为真实键
818
修改？
828
否
是
许可？
836
否
是
返回许可错误
838
复制到用户隔离范围。按需在层次中创建位置标志符键（空的）
840
识别真实键
842
打开键并返回结果
820
将系统范围键名识别为候选键名
830
存在？
832
是
否
中性的
键未找到错误
834

图 8

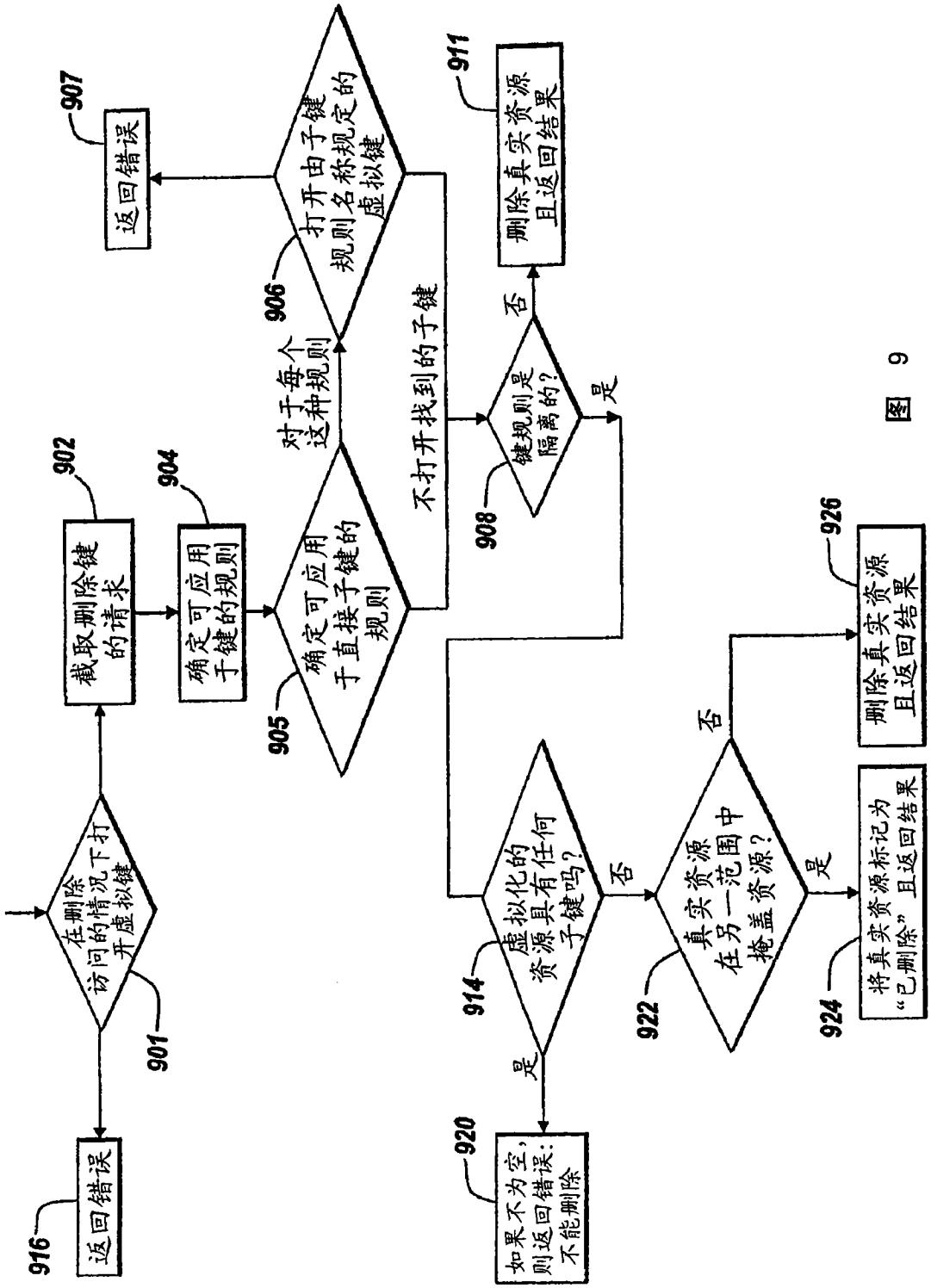

在删除访问的情况下打开虚拟键
901
截取删除键的请求
902
确定可应用于键的规则
904
确定可应用于直接子键的规则
905
对于每个这种规则
打开由子键规则名称规定的虚拟键
906
返回错误
907
不打开找到的子键
键规则是隔离的?
908
否
是
删除真实资源且返回结果
911
虚拟化的资源具有任何子键吗?
914
否
是
真实资源在另一范围中掩盖资源?
922
否
是
删除真实资源且返回结果
926
将真实资源标记为"已删除"且返回结果
924
如果不为空,则返回错误:不能删除
920
返回错误
916

图 9

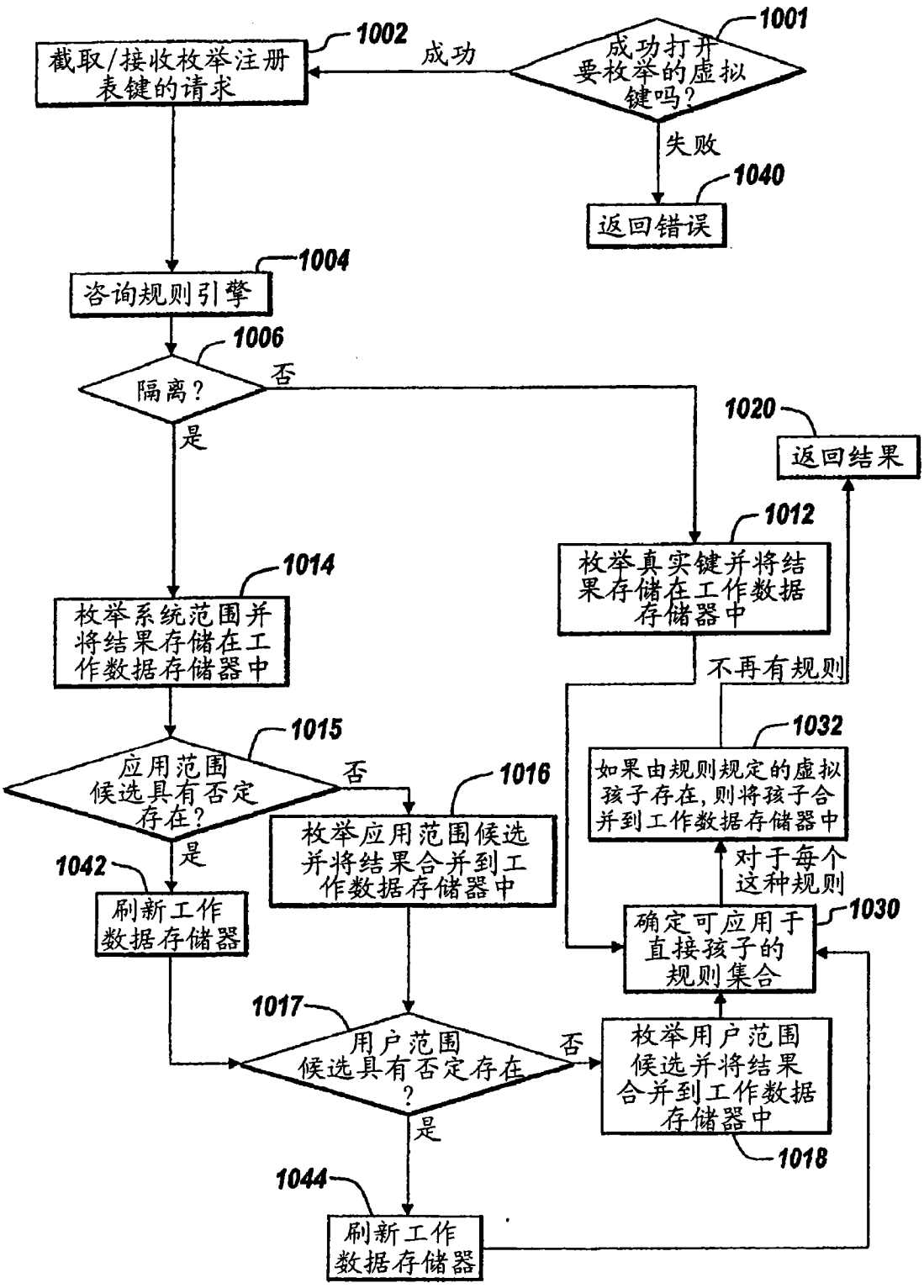

1001
成功打开要枚举的虚拟键吗?
成功
1002
截取/接收枚举注册表键的请求
失败
1040
返回错误
1004
咨询规则引擎
1006
隔离?
否
是
1020
返回结果
1012
枚举真实键并将结果存储在工作数据存储器中
1014
枚举系统范围并将结果存储在工作数据存储器中
不再有规则
1032
如果由规则规定的虚拟孩子存在,则将孩子合并到工作数据存储器中
1015
应用范围候选具有否定存在?
否
1016
枚举应用范围候选并将结果合并到工作数据存储器中
对于每个这种规则
1030
确定可应用于直接孩子的规则集合
是
1042
刷新工作数据存储器
1017
用户范围候选具有否定存在?
否
枚举用户范围候选并将结果合并到工作数据存储器中
1018
是
1044
刷新工作数据存储器

图 10

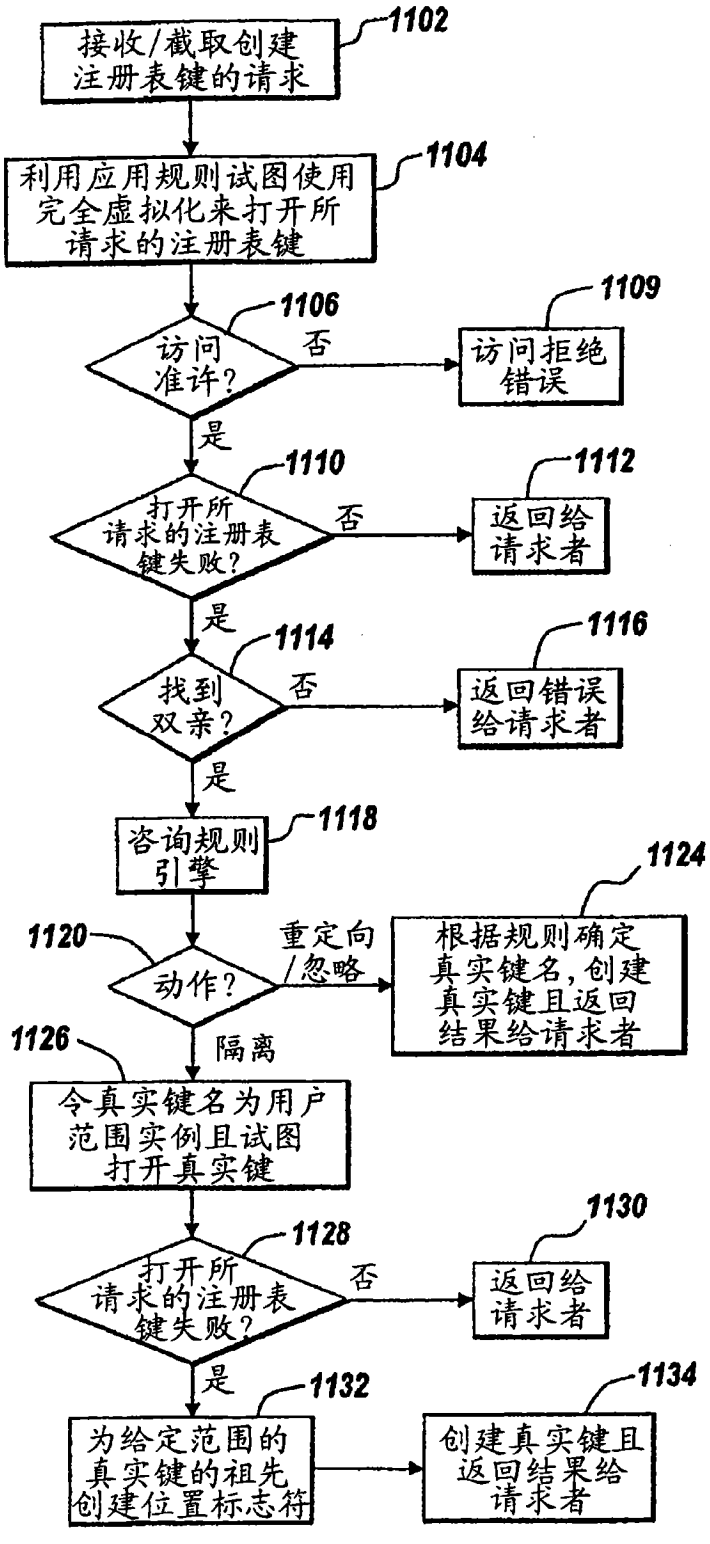

接收/截取创建注册表键的请求
1102
利用应用规则试图使用完全虚拟化来打开所请求的注册表键
1104
访问准许?
1106
否
访问拒绝错误
1109
是
打开所请求的注册表键失败?
1110
否
返回给请求者
1112
是
找到双亲?
1114
否
返回错误给请求者
1116
是
咨询规则引擎
1118
动作?
1120
重定向/忽略
根据规则确定真实键名,创建真实键且返回结果给请求者
1124
隔离
令真实键名为用户范围实例且试图打开真实键
1126
打开所请求的注册表键失败?
1128
否
返回给请求者
1130
是
为给定范围的真实键的祖先创建位置标志符
1132
创建真实键且返回结果给请求者
1134

图 11

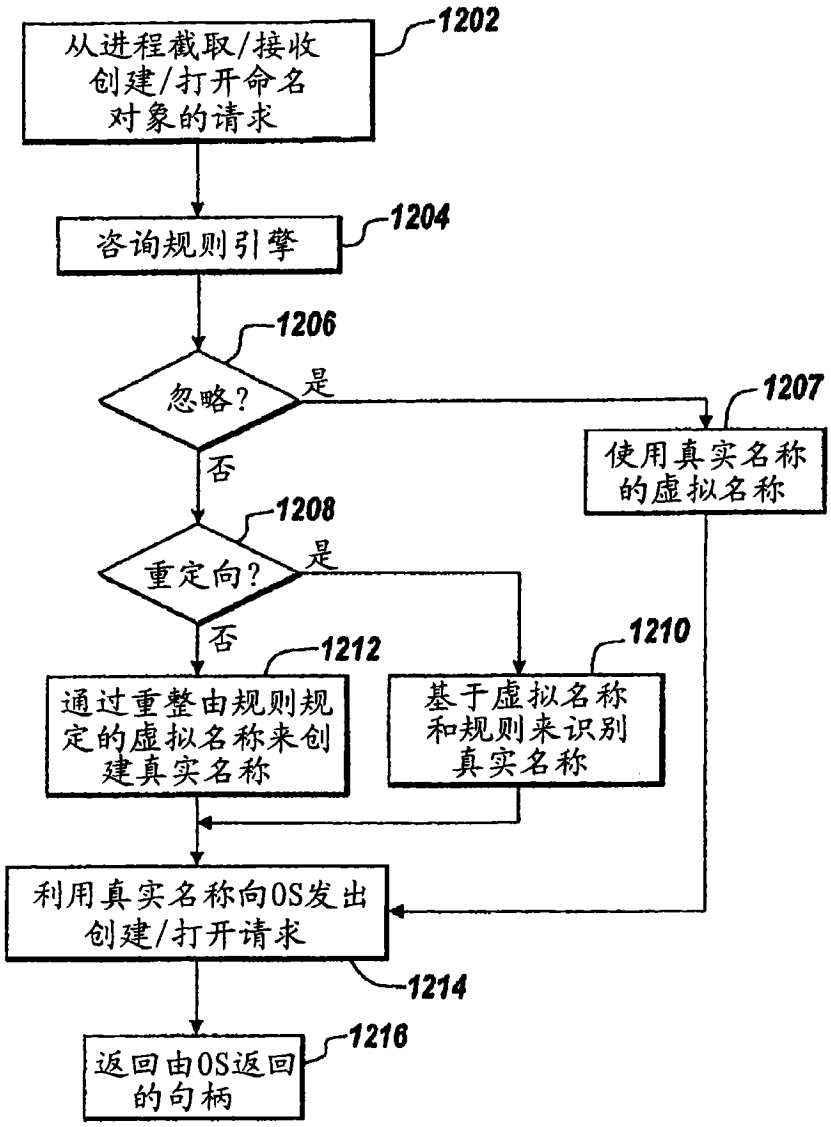

从进程截取/接收创建/打开命名对象的请求
1202
咨询规则引擎
1204
1206
忽略?
是
1207
使用真实名称的虚拟名称
否
1208
重定向?
是
否
1212
通过重整由规则规定的虚拟名称来创建真实名称
1210
基于虚拟名称和规则来识别真实名称
利用真实名称向OS发出创建/打开请求
1214
返回由OS返回的句柄
1216

图 12

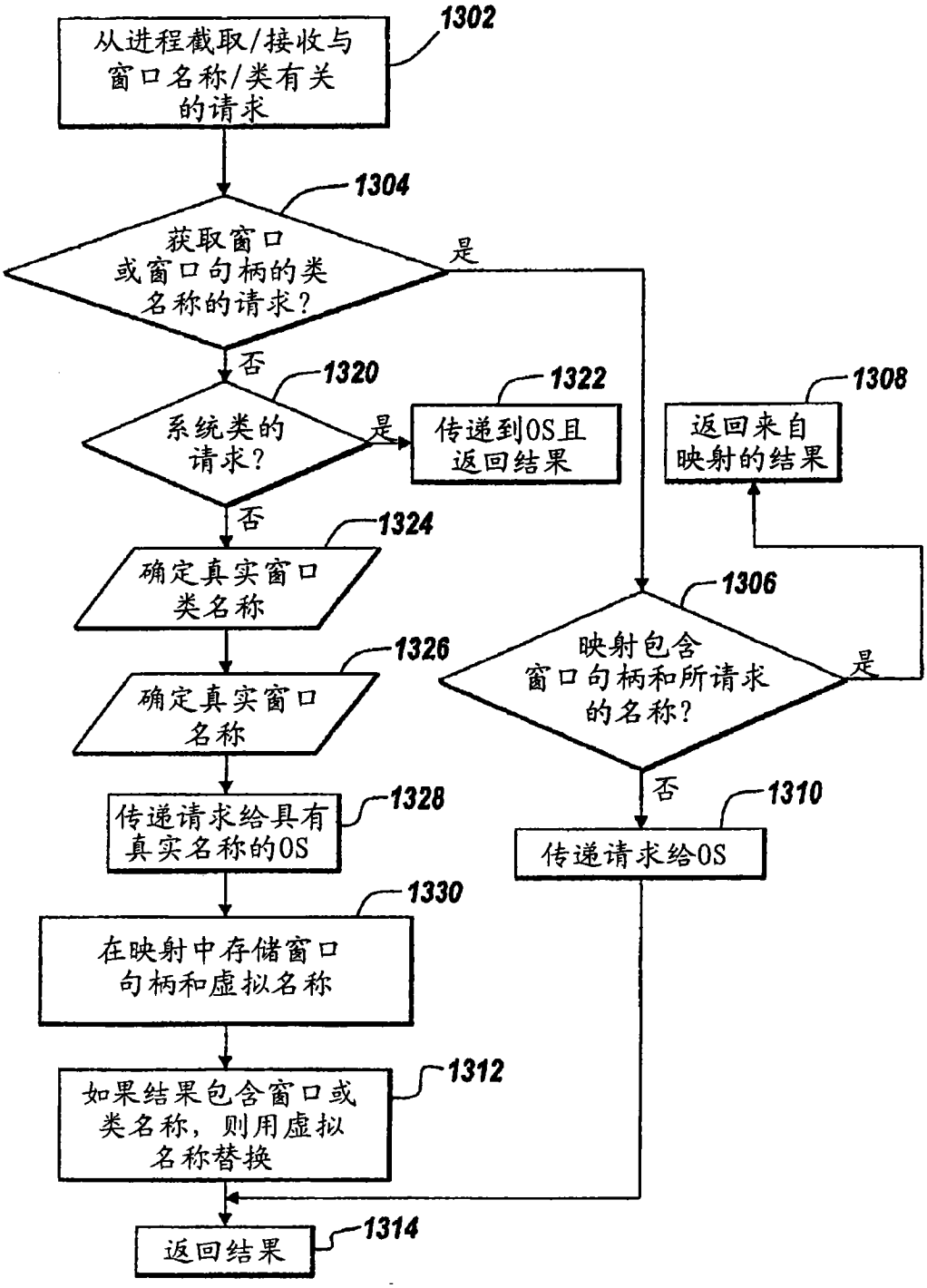

从进程截取/接收与窗口名称/类有关的请求
1302
获取窗口或窗口句柄的类名称的请求?
1304
是
否
系统类的请求?
1320
是
传递到OS且返回结果
1322
返回来自映射的结果
1308
否
确定真实窗口类名称
1324
映射包含窗口句柄和所请求的名称?
1306
是
确定真实窗口名称
1326
否
传递请求给具有真实名称的OS
1328
传递请求给OS
1310
在映射中存储窗口句柄和虚拟名称
1330
如果结果包含窗口或类名称，则用虚拟名称替换
1312
返回结果
1314

图 13

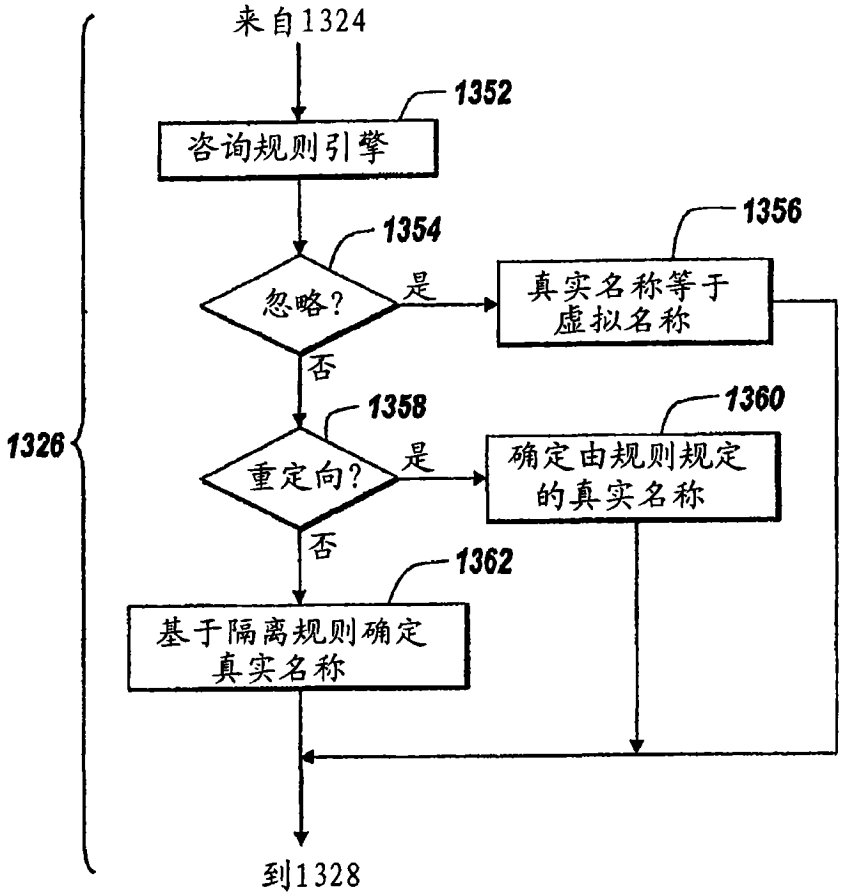

来自1324
1352
咨询规则引擎
1354
忽略?
是
1356
真实名称等于
虚拟名称
否
1358
重定向?
是
1360
确定由规则规定
的真实名称
否
1362
基于隔离规则确定
真实名称
1326
到1328

图 13A

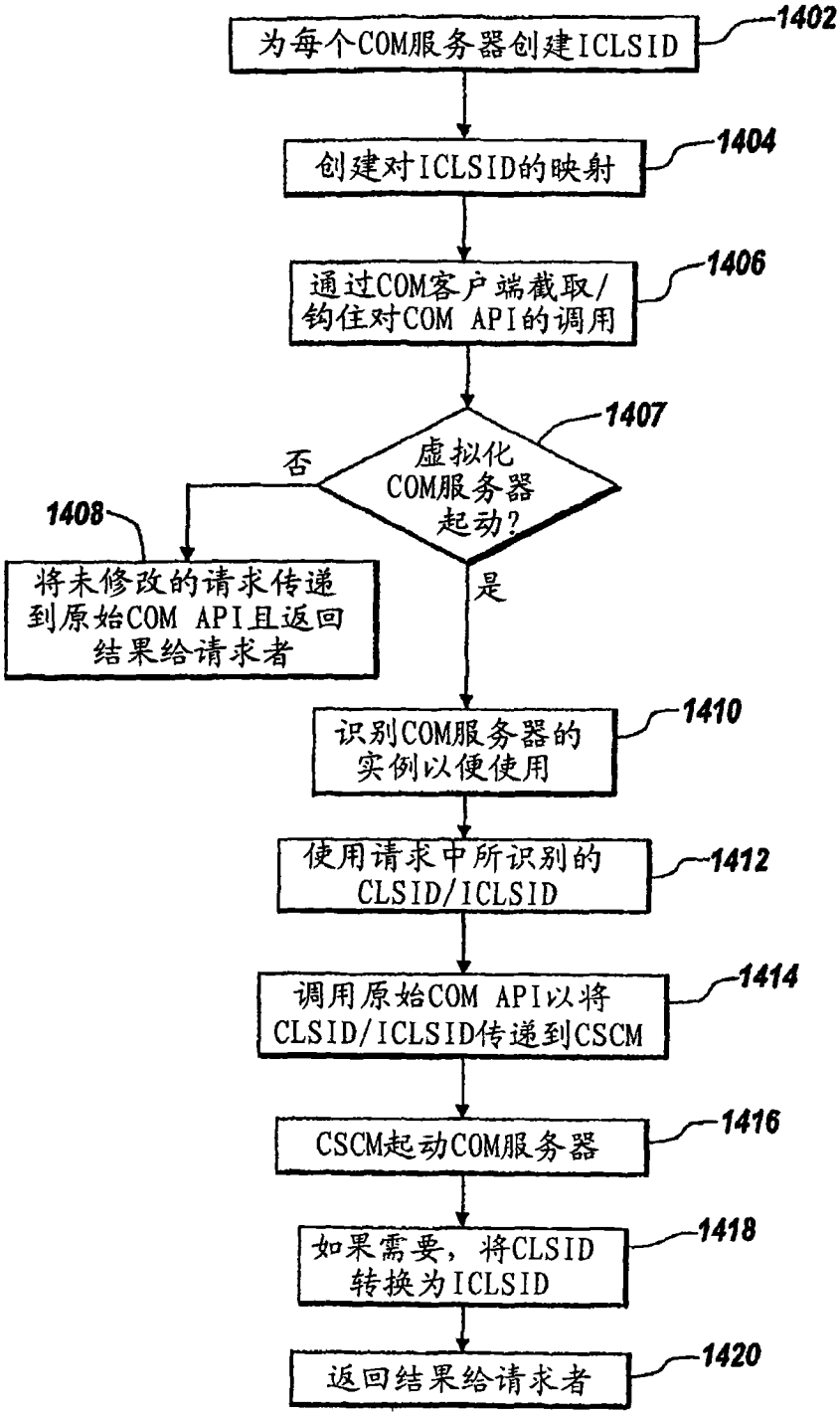

为每个COM服务器创建ICLSID
1402
创建对ICLSID的映射
1404
通过COM客户端截取/钩住对COM API的调用
1406
虚拟化COM服务器起动?
1407
否
是
1408
将未修改的请求传递到原始COM API且返回结果给请求者
识别COM服务器的实例以便使用
1410
使用请求中所识别的CLSID/ICLSID
1412
调用原始COM API以将CLSID/ICLSID传递到CSCM
1414
CSCM起动COM服务器
1416
如果需要，将CLSID转换为ICLSID
1418
返回结果给请求者
1420

图 14

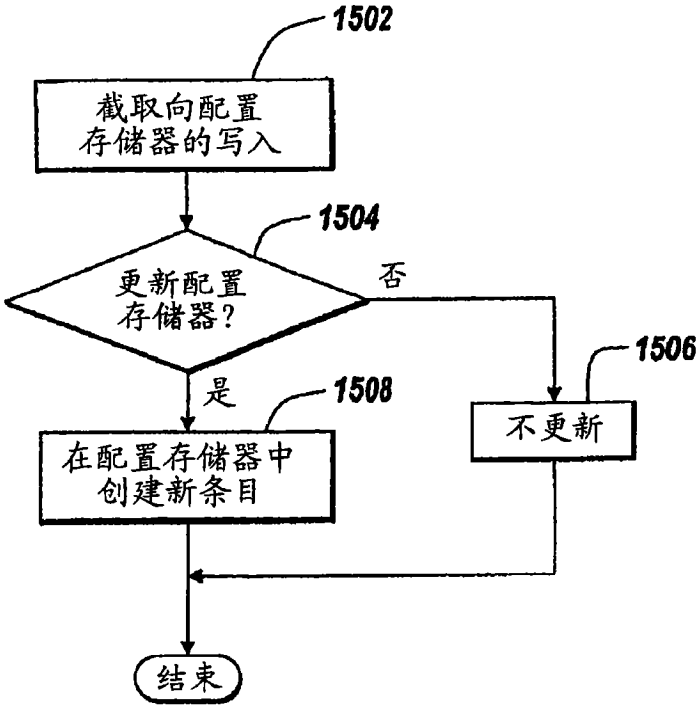

1502
截取向配置存储器的写入
1504
更新配置存储器?
否
1506
不更新
是
1508
在配置存储器中创建新条目
结束

图 15

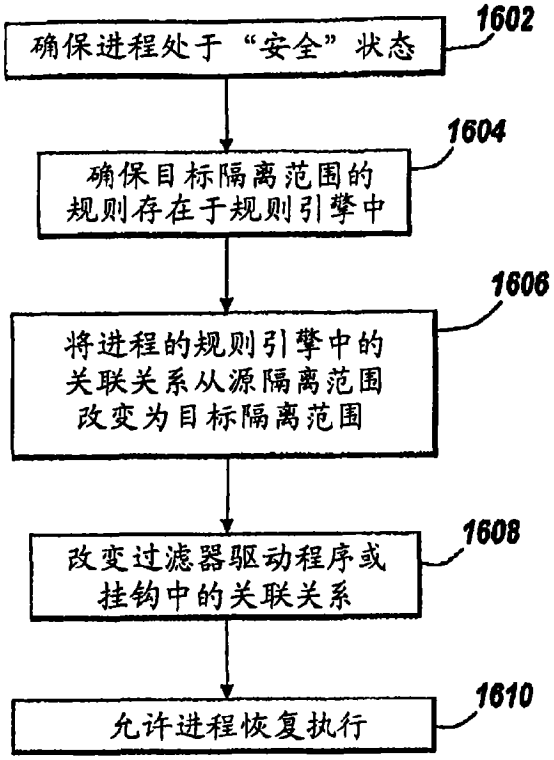

确保进程处于“安全”状态
1602
确保目标隔离范围的规则存在于规则引擎中
1604
将进程的规则引擎中的关联关系从源隔离范围改变为目标隔离范围
1606
改变过滤器驱动程序或挂钩中的关联关系
1608
允许进程恢复执行
1610

图 16