(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 1828527 B
(45)授权公告日 2010.09.29

(21)申请号 200610004388.8

(22)申请日 2006.01.28

(30)优先权数据
60/657,556 2005.02.28 US
11/171,905 2005.06.30 US

(73)专利权人 微软公司
地址 美国华盛顿州

(72)发明人 A·K·诺瑞 A·T·怀特恩
D·伍德夫特 J·A·布雷克雷
P·赛利斯 P·赛莎德锐
S·H·阿加瓦尔 S·特雷克

(74)专利代理机构 上海专利商标事务所有限公司 31100
代理人 张政权

(51)Int.Cl.
G06F 9/44(2006.01)
G06F 17/30(2006.01)

(56)对比文件
US 6128624 A,2000.10.03, 全文.
CN 1524217 A,2004.08.25, 全文.
P.A.Mittal,M.Kumar,M.K.Mohania,M.Nair,N.Batra,P.Roy,A.Saronwala,L.Yagnik. A framework for eGovernance solutions. IBM J.RES. &DEV.48 5/6.2004,48(5/6),717-733.

审查员 李锋

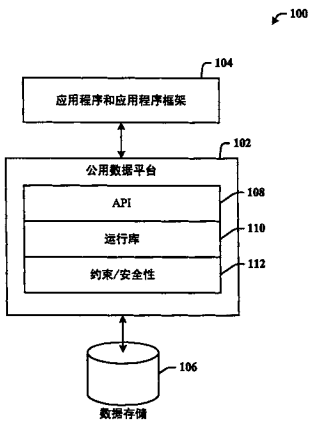
权利要求书 2 页 说明书 34 页 附图 23 页

(54)发明名称

用于跨不同应用程序框架的数据服务的平台

(57)摘要

公用数据存储和多个完全不同应用程序框架的多个应用程序之间的数据管理。提供了数据存储组件以便于数据的存储,其中数据包括结构化的、半结构化的和非结构化的数据。该公用数据平台接口到数据存储组件,以便提供多个完全不同的应用程序框架可访问的数据服务,该数据服务允许不同框架的相应应用程序访问数据。



CN 1828527 B

1. 一种通过提供数据服务来便于数据管理的数据平台，所述数据服务可由多个完全不同的应用程序框架访问，所述多个完全不同的应用程序框架允许对数据的统一访问，包括：

应用程序编程接口 API，它便于以公共类、接口和静态助手函数中的至少一种的形式与应用程序通信；

运行库组件，它接口到所述 API，并提供对象 - 关系映射、查询映射和约束强制实施中的至少一个；以及

公用数据模型，所述公用数据模型通过创建域专用类型、约束和关系中的至少一个，以析出对所述多个完全不同的应用程序框架公用的数据接口。

2. 如权利要求 1 所述的平台，其特征在于，所述域专用类型是实体类型，所述实体类型是用于对属性和方法中的至少一个进行分组的规范，其中，域专用类型采用实体、表、表集和关系中的至少一个。

3. 如权利要求 2 所述的平台，其特征在于，所述公用数据模型的模式定义了所述域专用类型所采用的所述实体、关系和表集中的至少一个，使得用于对在所述模式中定义的元素的名字定范围的名字空间与其相关联。

4. 如权利要求 1 所述的平台，其特征在于，所述公用数据模型定义了查询语言，其中，所述查询语言允许对于对象结构的丰富查询，所述对象结构提供了对存储在启用了所述数据平台的存储中的数据的类型化的、基于对象的抽象。

5. 如权利要求 4 所述的平台，其特征在于，所述查询语言是 Opath 和面向对象的结构化查询语言 OSQL 中的至少一个。

6. 如权利要求 1 所述的平台，其特征在于，还包括约束 / 安全性引擎，它便于声明性地创作约束，并控制对所述数据平台的至少一个实体的访问。

7. 如权利要求 1 所述的平台，其特征在于，还包括持久化引擎，它通过调用命令性对象 - 关系映射和非命令性对象 - 关系映射中的至少一种来调用将语言类映射到底层表式表示的对象 - 关系映射。

8. 如权利要求 7 所述的平台，其特征在于，所述对象 - 关系映射是从应用程序空间到所述公用数据模型的映射，且独立于从所述公用数据模型到存储的映射。

9. 如权利要求 1 所述的平台，其特征在于，所述多个完全不同的应用程序框架包括业务线框架、最终用户框架、系统管理框架、用户应用程序框架、协作框架、业务框架和个人信息框架中的至少一个。

10. 如权利要求 1 所述的平台，其特征在于，所述应用程序是最终用户应用程序、知识工作者应用程序、业务线应用程序、web 应用程序、联系人管理应用程序、文档管理应用程序、协作应用程序、电子邮件应用程序、顾客关系管理应用程序、企业资源规划应用程序和系统管理应用程序中的至少一个。

11. 如权利要求 1 所述的平台，其特征在于，所述运行库组件提供对实体状态的管理，所述实体状态包括标识映射、对象改变追踪和原始值的至少之一。

12. 如权利要求 1 所述的平台，其特征在于，所述数据平台和各自的组件是层不可知的，且存在于客户机层、中间层、服务器层和 web 服务层中的至少一个中。

13. 如权利要求 1 所述的平台，其特征在于，所述公用数据模型提供数据存储组件内的

共享数据，使得与对应的框架相关联的完全不同的应用程序可以访问所述共享数据。

14. 如权利要求 1 所述的平台，其特征在于，所述公用数据模型提供数据存储组件内的专用数据，使得所述专用数据仅由与特定框架相关联的特定应用程序访问。

15. 如权利要求 1 所述的平台，其特征在于，还包括规则服务、改变跟踪服务、冲突检测服务、事件处理服务和通知服务中的至少一个。

16. 一种管理数据的计算机实现的方法，包括：

将数据平台覆盖在对结构化的、半结构化的和非结构化的数据类型进行建模和存储的数据存储上，以便提供支持完全不同的应用程序框架的丰富数据模型、映射、查询和数据访问机制中的至少一个的服务；

将一个或多个应用程序框架覆盖到所述数据平台上，以便允许每个框架中的至少一个应用程序访问所述数据存储；

以公共类、接口和静态助手函数中的至少一种的形式与应用程序通信；

提供对象 - 关系映射、查询映射和约束强制实施中至少一种；以及

通过创建域专用类型、约束和关系中的至少一个，析出对多种完全不同的应用程序框架公用的建模概念。

17. 如权利要求 16 所述的方法，其特征在于，还包括：

创建对象；

使用会话打开到所述存储的连接并建立安全上下文；

将存储上下文的实例返回给应用程序；

展示检索对象的接口；

在应用安全性的同时，将查询映射到结构化查询语言 SQL；

将结果返回给所述数据平台的运行库和所述应用程序；以及

保存经封装的存储上下文对象上的改变。

18. 如权利要求 16 所述的方法，其特征在于，还包括：

将共享数据提供给所述完全不同的应用程序框架上完全不同的应用程序；以及

使用专用于特定的完全不同的应用程序框架上的应用程序的专用数据。

19. 一种管理数据的计算机实现的系统，包括：

用于将数据平台覆盖在对结构化的、半结构化的和非结构化的数据类型进行建模和存储的数据存储上，以便提供支持完全不同的应用程序框架的丰富数据模型、映射、查询和数据访问机制中的至少一个的服务的装置；

用于将一个或多个应用程序框架覆盖到所述数据平台上，以便允许每个框架中的至少一个应用程序访问所述数据存储的装置；

用于以公共类、接口和静态助手函数中至少一种的形式与应用程序通信的装置；

用于提供对象 - 关系映射、查询映射和约束强制实施中的至少一种的装置；以及

用于通过创建域专用类型、约束和关系中的至少一个，析出对多种完全不同的应用程序框架公用的建模概念的装置。

用于跨不同应用程序框架的数据服务的平台

[0001] 相关申请的交叉引用

[0002] 本申请要求于2005年2月28日提交的，题为“PLATFORM FOR DATASERVICES ACROSS DISPARATE APPLICATION FRAMEWORKS(用于跨不同应用程序框架的数据服务的平台)”的美国专利申请第60/657,556号的优先权。本申请也涉及以下美国专利申请：于2005年2月28日提交的，题为“DATA MODELFOR OBJECT-RELATIONAL DATA(用于对象－关系数据的数据模型)”的临时专利申请第60/657,295号；于_____提交，题为“DATA MODEL FOROBJECT-RELATIONAL DATA(用于对象－关系数据的数据模型)”的专利申请第____号；于2005年2月28提交的，题为“STORAGE API FOR A COMMONDATA PLATFORM(用于共用数据平台的存储API)”的临时专利申请第60/657,522号；以及于_____提交的，题为“DATA MODEL FOR OBJECT-RELATIONALDATA(用于对象型关系数据的数据模型)”的专利申请第_____号。这些专利申请全部引用在此作为参考。

背景技术

[0003] 数据成为了几乎所有应用程序中的重要资产，无论该应用程序是用于浏览产品和生成订单的业务线(LOB)应用程序还是用于在人员之间调度会议的个人信息管理(PIM)应用程序。应用程序对应用程序数据执行数据访问／操纵和数据管理操作。典型的应用程序操作查询数据集合、取得结果集、执行改变数据状态的某些应用逻辑，最后将数据保存到存储介质中。

[0004] 传统上，客户机／服务器应用程序将查询和持久动作提交给部署在数据层中的数据库管理系统(DBMS)。如果有以数据为中心的逻辑，就将其编码为数据库系统中的存储过程。数据库系统按照表格和行对数据进行操作，而应用层内的应用程序按照编程语言对象（例如类和结构）对数据进行操作。应用程序和数据层中的数据操作服务（以及机制）的不匹配在客户机／服务器系统中是可以容忍的。然而，随着web技术（以及面向服务的体系结构）的出现以及应用程序服务器在较广的范围内得到接受，应用程序变成多层的，更重要的是现在数据在每一层中存在。

[0005] 在这种分层应用程序体系结构中，在多个层中操纵数据。此外，随着在可寻址性和大容量存储器方面的硬件进步，更多数据变为存储器常驻的。应用程序也处理不同类型的数据，诸如对象、文件和XML(可扩展标记语言）数据。

[0006] 在硬件和软件环境中，越来越多地需要能很好地与编程环境集成的丰富数据访问和操纵服务。所引入解决上述问题的一个常规实现是数据平台。数据平台提供了服务（机制）的集合，以供应用程序访问、操纵和管理与应用程序编程环境很好地集成的数据。然而，这种常规体系结构有很多不足之处。对这种数据平台的一些关键要求包括复杂对象建模、丰富关系、逻辑和物理数据抽象的分离、查询丰富数据模型概念、主动通知、更好地与中层架构集成。

发明内容

[0007] 以下示出了本发明的简单概述，以便提供对该体系结构的某些方面的基本理解。该概述不是对体系结构的宽泛综述。它不是旨在标识本发明的关键 / 决定性元素或描绘本发明的范围。它唯一的目的是以简单的形式示出本发明的某些概念，作为以后示出的更为详细描述的序言。

[0008] 这里所解释和要求保护的本发明的一方面包括便于在多个不同的应用程序框架的公用数据存储和多个应用程序之间进行数据管理的体系结构。它形成了不同于将表格映射到对象的应用程序的映射层。该体系结构将桌面应用程序和业务线 (LOB) 应用程序框架之间的空白连接起来，以允许应用程序在应用程序对象级而非表格级处理数据。此外，该体系结构允许在所有的框架之间共享该数据，使得由最终用户应用程序定义的数据实体可以被 LOB 应用程序所使用，反之亦然。

[0009] 该体系结构包括便于数据存储的数据存储组件，其中数据包括结构化的、半结构化的和非结构化的数据。公用数据平台与数据存储组件接口，以便提供多个不同的应用程序框架能够访问的数据服务，其中数据服务允许不同框架的相应应用程序访问数据。该数据平台还包括 :API(应用程序编程接口)，以便利于与公共类、接口和静态助手函数形式的应用程序通信 ;运行时间组件，它接口到 API 并提供对象 - 关系映射、查询映射以及强制实施约束 ;以及约束 / 安全引擎，以便于声明性地创作约束，并控制对数据平台实体的访问。

[0010] 在本发明的另一方面，公用数据平台 (CDP) 提供在各种最终用户应用程序框架 (例如 PIM(个人信息管理器) 框架到 LOB(业务线) 应用程序框架) 间公用的数据服务。应用程序的范围包括诸如浏览器、邮件和媒体应用程序等最终用户应用程序 ;诸如文档管理和协作应用程序等知识工作者应用程序 ;诸如 ERP(企业资源计划) 和 CRM(客户关系管理) 等 LOB 应用程序 ;Web 应用程序和系统管理应用程序。

[0011] 在本发明的另一方面，CDP 为应用程序提供的好处包括在多层环境中提供建模和存储结构化、半结构化和非结构化数据的能力的丰富存储、灵活组织、丰富查询 / 搜索、丰富行为、灵活管理、数据同步、共享、方案和灵活部署。

[0012] 为了达到上述和相关目的，这里结合以下描述和附图描述该体系结构的某些说明性方面。然而，这些方面仅示出各种可采取本体系结构原理的方式中的几种，且本发明旨在包括所有这些方面以及它们的等价方面。结合附图，该体系结构的其他优点和新颖性将通过以下对本发明的详细描述而变得显而易见。

附图说明

[0013] 图 1 示出了依照本体系结构采用公用数据平台 (CDP) 的系统。

[0014] 图 2 示出了依照所揭示的体系结构的更为详细的 CDP 系统。

[0015] 图 3 示出了实现便于管理数据的公用数据平台的方法。

[0016] 图 4 示出了关于本发明的体系结构的 CDP 组件的示意图。

[0017] 图 5 示出了 CDP 各个组件内的数据流。

[0018] 图 6 示出了可以使用 CDP 实现的各种框架。

[0019] 图 7 示出了允许多个应用程序共享数据的公用的基于数据库的文件存储系统情况。

[0020] 图 8 示出了依照 CDP 和相关联的体系结构使用多个框架的单个应用程序。

[0021] 图 9 示出了与关联于多个不同的框架的多个应用程序共享数据的 CDP。

[0022] 图 10 示出了便于数据管理的 CDP 的双层部署。

[0023] 图 11 示出了便于数据管理的共享数据的双层部署。

[0024] 图 12 示出了第二配置，使得应用程序具有不希望其他应用程序看到和 / 或使用的私有数据。

[0025] 图 13 示出了所感兴趣的第三配置，使得另一应用程序直接地访问存储。

[0026] 图 14 示出了 CDP 组件的三层部署配置。

[0027] 图 15 示出了 CDP 组件的三层部署配置。

[0028] 图 16 示出了同时运行在客户机层和中间层上的应用程序逻辑图。

[0029] 图 17 示出了同时运行在客户机层和中间层上的应用程序逻辑图。

[0030] 图 18 示出了使用至少一个实体对项进行建模。

[0031] 图 19 示出了通过将 UAF 结合在 CDP 的之上来实现各种功能的可扩展机制。

[0032] 图 20 示出了在 CDP 上实现的 LOB 应用程序的例子。

[0033] 图 21 示出了便于在 CDP 的各个组件内管理数据流的方法。

[0034] 图 22 示出了便于跨多个不同的框架部署 CDP 的方法，其中不同应用程序可与每个框架有关。

[0035] 图 23 示出了可用于执行所揭示的框架的计算机的框图。

[0036] 图 24 示出了依照本发明的示例性计算机环境的示意性框图。

具体实施方式

[0037] 现在参考附图描述该体系结构，在所有附图中，相同的参考标号用于标识相同的组件。在以下附图中，为了说明起见，提供了大量的具体细节，以便提供对本发明的全面理解。然而，显然的是，可以不采用这些具体细节来实现本发明。在其他例子中，以框图形式示出了公知的结构和设备，以便于描述该体系结构。

[0038] 如本申请中所使用的，术语“组件”和“系统”指的是计算机相关实体，它们可以是硬件、硬件和软件的组合、软件或执行中的软件。例如，组件可以是，但不限于，在处理器上运行的进程、处理器、对象、可执行代码、执行的线程、程序和 / 或计算机。作为示例，运行在服务器上的应用程序和服务器都可以是组件。一个或多个组件可以驻留在执行的进程和 / 或线程内，组件可以位于一个计算机上和 / 或在两个或多个计算机上分布。

[0039] 数据平台是为应用程序提供服务（或机制）集合的平台，以便于应用程序访问、操纵和管理与应用程序编程环境很好地集成的数据。本发明是对常规数据平台的改进。该体系结构是提供跨各种应用程序框架（例如 PIM（个人信息管理器）框架和 LOB（业务线）框架）公用的数据服务的公用数据平台 (CDP)。应用程序的范围包括，诸如浏览器、邮件和媒体应用程序等最终用户应用程序；诸如文件管理和协作应用程序等知识工作者应用程序；诸如 ERP（企业资源计划）和 CRM（客户关系管理）等 LOB 应用程序；Web 应用程序和系统管理应用程序。

[0040] CDP 至少向应用程序提供了以下好处：

[0041] 1. 丰富存储—建模和存储所有类型的数据（结构化的、半结构化的和非结构化的）的能力。

[0042] a. 关系性数据建模和访问。

[0043] b. 丰富对象抽象和编程环境。

[0044] c. 经由 XML 存储和查询的半结构化数据建模。

[0045] d. 作为文件的非结构化数据。

[0046] 2. 灵活组织—非静态地组织任意对象集合为表格的能力。

[0047] a. 支持文件系统名字空间和组织。

[0048] 3. 丰富查询 / 搜索—查询所有数据的能力。

[0049] a. 支持丰富查询（例如 SQL、OSQL（面向对象 SQL）、XML 查询、C# 序列）。OSQL 是作为 SQL 的超集的功能语言。

[0050] 4. 丰富行为—支持丰富数据行为。这不是对应用程序 / 业务过程逻辑的替代。

[0051] 5. 灵活管理—不同粒度上的管理（例如项级操作，诸如复制、移动和串行化）。

[0052] 6. 数据同步—任意数据集合的对等和主从管理同步。

[0053] 7. 共享—跨多个应用程序和多个应用程序框架共享数据的能力。例如跨 Outlook 和 CRM 应用程序共享联系人。

[0054] 8. 方案—用于用户和 ISV（独立支持销售商）应用程序的丰富、取出即可用 (out-of-the-box) 方案，以便于彼此协作。

[0055] 9. 灵活部署—在二和三层环境中可部署。

[0056] CDP 和相关联的体系结构允许上述的所有好处。关键的创新包括分层体系结构、析出在多个应用程序框架之间公用的建模概念的公用数据模型以及 CDP 组件（功能）体系结构。

[0057] 首先参考附图，图 1 示出了采用 CDP102 的系统 100。CDP102 用于提供数据应用程序和应用程序框架 104 以及数据存储器 106 上的数据之间的数据管理。数据存储器 106 可以存储，例如结构化的、半结构化的和非结构化的数据类型。如上所述，CDP102 提供跨应用程序框架和与之相关联的最终用户应用程序公用的数据服务。CDP102 还包括便于与应用程序和应用程序框架 104 接口的 API108、运行库组件 110 以及约束 / 安全引擎组件 112。API108 为使用 CDP 的应用程序提供了公共类、接口和静态助手功能形式的编程接口。例子包括 StorageContext（存储环境）、StorageSearcher（存储搜索器）、Entity（实体）、TableSet（表集）、Table（表）、EntityReference（实体引用）以及 TableReference（表引用）。应该理解，数据库编程语言集成（例如 C# 序列操作符）可以是 API108 的一部分。

[0058] CDP 运行库组件 110 是实现在公共 API 层 108 中所展示的各种特征的层。它通过提供对象 - 关系映射和查询映射、强制实施数据模型约束等来实现公用数据模型。更具体地，CDP 运行库 110 包括 ：公用数据模型组件实现 ；查询处理器组件 ；会话和事务组件 ；对象高速缓存，它可以包括会话高速缓存和显式高速缓存 ；服务组件，它包括改变跟踪、冲突检测以及事件处理 ；指针和规则组件 ；业务逻辑主机组件 ；以及持久化和查询引擎，它提供了核心的持久化和查询服务。持久化和查询服务内部是对象 - 关系映射，包括查询 / 更新映射。CDP102 也包括约束 / 安全引擎 112，用于将约束应用到数据存储器 106 上和应用安全策略，例如基于角色的安全。

[0059] 图 2 示出了可以包括 CDP102 的更为详细的 CDP 系统 200，它接口到分离的数据存储（未示出）的存储管理组件 202。或者，存储管理组件 202 可以包括诸如可以与 SQL 服务

器实现相关联的数据存储。应该理解，数据存储可以存储结构化的、半结构化的和非结构化的数据类型。

[0060] CDP 的目标是通过允许支持各种应用程序框架 204（被示为 AF_1，AF_2，...，AF_z）来支持快速的应用程序发展。框架 204 可以包括例如 LOB、最终用户和系统管理应用程序框架。与应用程序框架 204（分别示为 AF_1，AF_2 和 ... AF_Z）相关联的应用程序 206（示为 APP_1，...，APP_S；APP_1，...，APP_T）可以充分利用各自的应用程序框架 204、CDP102 和底层存储 202 来开发丰富的应用程序。分层方法的好处在下文中描述。

[0061] 存储管理层 202 提供了对核心数据管理能力（例如可伸缩性、容量、可用性和安全性）的支持；CDP 层 102 支持对应用程序框架 204 的丰富数据模型、映射、查询和数据访问机制。CDP 机制是可扩展的，使得多个应用程序框架 204 可以被构建在数据平台上。应用程序框架 204 是专用于应用程序域（例如最终用户应用程序和 LOB 应用程序）的附加模型和机制。分层体系结构方法具有若干优点。它允许每一层独立和快速地更新和部署。CDP 层 102 与存储层 202 相比，可以更为敏捷、更新更为自由且更为频繁地更新。分层方法将 CDP 层 102 与公司策略相结合。最终，存储层 202 可以集中在核心数据管理能力上，从而与策略相一致。

[0062] 现在参考图 3，示出了实现公用数据平台的方法。尽管为了简化说明起见，这里以例如流程图形式示出的一个和多个方法被示出和描述为一系列动作，但是应该理解，本体系结构不受动作顺序的限制，因为依照本发明，有些动作会以不同顺序和 / 或与这里所示和描述的其他动作同时发生。例如，本领域的技术人员会理解，方法可以选择性地被表示为诸如状态图中一系列互相关连的状态或事件。此外，不是所有示出的动作都是实现依照该体系结构的方法所必需的。

[0063] 在 300 处，提供了核心数据管理层，用于对结构化的、半结构化的和非结构化的数据类型建模并将它们存储在数据存储器中。在 302 处，CDP110 层被应用到核心数据管理层上，以便提供对应用程序框架支持丰富数据模型、映射、查询和数据访问机制的数据服务。在 304 处，一个或多个应用程序框架覆盖 CDP。在 306 处，在每个应用程序框架内提供一个或多个应用程序，这些应用程序框架现在可以经由 CDP 提供的数据服务来访问数据存储的数据。

[0064] 图 4 示出了该体系结构的 CDP 组件的示意框图。应该理解，该示意图中任何组件和 / 或框的定位并不意味着（或必须防止）跨进程 / 机器边界的任何特定的部署。CDP 使用优化并发性模型，这样使得如果要保存变化，且已经对底层数据作了其他改变，那么冲突检测以应用程序专用的方式来解决它。为了成为有效的数据平台，CDP 包括诸如编程语言集成、丰富数据建模、持久化框架、服务等特征。API108 便于语言集成和应用程序 400 经由 CDP 运行库 110 对存储器 202 进行数据访问。域不可知意味着 CDP 对数据的特性和形态以及对其所需的语义约束作出最少的假设。为此，CDP 提供了以下特征（在下文中更为详细地描述）：

[0065] 公用数据模型 (CDM)：在 CDP 运行库 110 的中心是 CDM402。CDM402 的目标是析出跨多个应用程序域（从主要对用户数据工作的应用程序 (PIM，文档等）到 LOB 和企业数据）公用的建模概念。除了提供丰富对象和关系抽象之外，CDM402 提供了对结构化的、非结构化的和半结构化的数据的支持。

[0066] 行 / 实体数据—CDM402 支持丰富实体关系模型，以便捕捉结构化数据（例如业务数据）的结构和行为。CDM402 是核心关系型模型的超集，其中具有对丰富对象抽象和关系建模的扩展（例如文档和联系人之间的作者关系；采购订单和订单线之间的线关系）。

[0067] 文件数据—CDM402 支持“文件流”数据类型，以便存储和操纵非结构化（文件）数据。文件流数据类型可以将数据存储为文件并支持文件访问 API。SQL 服务器中本身就支持映射到 NTFS 文件流的文件流数据类型，并支持所有基于文件句柄 / 流的操作。除将非结构化内容建模为 CDM402 中的文件流之外，使用实体类型，有用的内容可以被升级为结构化属性。基于数据库的文件存储系统定义了文件后备项的概念，文件后备项是对结构化属性以及非结构化内容的文件流建模的实体。文件后备项提供了相关联的文件流上的丰富查询以及基于流的操作。

[0068] XML 数据—XML 文档可以被建模为 CDM402 中两种主要的方式：(1) 将其存储为 XML 数据类型；(2) 将 XML 文档映射到一个或多个实体（例如类似于数据契约）。CDM402 支持 SQL 服务器中所支持的 XML 数据类型。XML 数据类型可以是任意实体属性的类型；XML 数据类型允许存储非类型化或类型化的 XML 文件。通过将一个或多个 XML 模式与 XML 文件属性相关联，提供了强类型化。

[0069] API108 中的编程语言集成，包括查询：核心 CDP 特征组件一会话和事务 404、查询 406、持久化 408、指针 410、服务 412、对象高速缓存 414 和业务逻辑主机 416 被封装在 CDP API108 中可用的几个“运行库”类中（例如 StorageContext（存储环境））。基于在 CDM402 中创作的类型，CDP 设计时工具生成强类型化的 CLR（公共语言运行库）类。CDP 要求对由 CDM402 定义的类型化系统的查询语言。CDP 可支持 C# 序列操作符和 OPATH 作为其查询语言。为了本申请的目的，CDP 所支持的查询语言一般被称为公共查询语言 (CQL)。CQL 被设想成包括核心关系型代数（例如，选择、结合和项操作符）。虽然其句法可能与 SQL 不同，但 CQL 可以用直接的方式被映射到 SQL。CQL 允许对程序设计员所处理的对象结构进行丰富查询。目标是使得 CQL 与 C# 组所作的序列操作符工作结合。这些特征有效地提供了对存储在关系型数据库管理系统 (RDBMS) 中（或者任何其他启用了 CDP 的存储）的数据的强类型化的、基于对象的抽象。此外，CDP 的持久化框架可以用于持久地坚持和查询无格式普通 CLR 对象 (POCO)。

[0070] 持久化引擎—持久化引擎提供了声明性映射定义，它描述了对象是如何从来自关系型存储的组件片段组装的。该引擎包括查询生成组件（未示出），它按照对象查询表达式采用由查询处理器定义的表达式，并接着将其与声明性映射相组合。这转换成访问数据库中的底层表的等价查询表达式。更新生成组件（未示出）查找改变跟踪服务，并在映射元数据的协助下描述如何将对象领域中的那些改变转换成表领域中的改变。

[0071] 查询 / 搜索文件和 XML 数据一如上所解释的，CDM402 分别使用文件流和 XML 数据类型来存储非结构化的和半结构化的数据。CQL 能够查询这些数据类型。对于被升级为结构化实体（例如 WinFS 文件后备项）的文件内容，CQL 的关系型操作符可以查询这些实体。可以使用全文本搜索来查询被存储为文件流的非结构化数据。可以使用 XPath 或 XQuery 来查询 XML 内容。

[0072] 对象 - 关系映射：由于 CDP 在关系型（表式）存储之上提供了基于对象的抽象，因此需要提供 O-R 映射组件。CDP 支持命令性映射 (CDP 描述映射如何发生）和非命令性映射

（类型设计者在指定映射方面具有某些灵活性）。注意，当今的基于数据库的文件存储系统实现使用命令性的映射，而更一般的 O-R 持久化框架需要非命令性映射。

[0073] 高速缓存 :CDP 运行库维护查询结果（例如指针）和未被提交的更新的高速缓存。这被称为会话高速缓存。CDP 也提供显式高速缓存，它允许应用程序以断开模式工作。CDP 为显式高速缓存中的数据提供各种一致性保证。高速缓存通过将数据的盘上身份与存储器中的对象相关来执行身份管理。

[0074] 查询处理器 :当查询存储时，CQL 查询被映射到 SQL ;然而，当查询显式高速缓存时，CQL 查询由 QP 组件处理。数据库访问是经由查询处理器来完成的。查询处理器允许多个前端来处理要表达并接着被映射到内部规范格式的多个查询语言。这是按照它所处理的应用程序的域模型和对象来完成的。查询接着被传递到处理器（这是管道），被接着被转换成后端专用查询。

[0075] 指针 :CDP 提供仅前向和可滚动指针。指针支持通告、使用展开 / 折叠状态的多级分组、动态排序以及过滤。

[0076] 业务逻辑主机 :CDP 提供运行时环境以便主存类型 / 实例和操作的数据中心逻辑。这种数据中心业务逻辑不用于应用程序 / 业务过程逻辑，后者可以主存在应用程序服务器中。对象不仅是数据库中的行。当对象在存储器中物化后，它们实际上是具有应用程序可调用的行为的对象。在系统中有扩展点，它们主要是所有在运行时操作以扩展 CDP 的事件和回叫。这些对象不仅是对象，而是 CLR 对象、.NET 对象等。CDP 允许截取这些对象中的属性剩余方法调用。应用程序可以定制这些对象的行为。

[0077] 服务 :CDP 提供对所有 CDP 客户机可用的一组核心服务。这些服务包括 :规则、改变跟踪、冲突检测、事件处理以及通告。事件处理扩展了来自框架级服务或用于应用程序的 CDP 运行库 110 以添加附加行为，且还用于用户界面处的数据绑定。

[0078] 约束 :CDP 提供约束 / 安全组件 112 以便允许类型化设计者至少能够声明性地创作一个约束。这些约束在存储中执行。一般地，CDP 约束的范围包括诸如长度、精度、比例、默认值、校验等概念。这些约束在运行时由 CDP 约束引擎来强制实施。

[0079] 安全 :CDP 提供了基于角色的安全模型—用户的凭证确认其“角色”（诸如管理员、权限高的用户、批准者等）。每个角色被分配一组访问权限。CDP 的安全引擎强制实施这些安全策略。此外，CDP 提供安全模型以便控制对 CDP 中实体的访问。安全模型可以支持对操作系统用户的认证、实体的授权级别（例如，对读和更新有分开的权限）等。

[0080] 注意，约束 / 安全组件 112 被示为与 CDP 运行库组件 110 分离，因为它可以作为与其分离的实体来运作。或者，可能更为有效的是，约束 / 安全组件 112 与存储器组件 202 相结合，存储器组件可以是数据库系统。

[0081] 一起考虑，这些特征提供了强大的平台，用于开发可以跨不同的层来灵活部署的数据中心应用程序和逻辑。注意，该图中运行库组件（或框）的定位并不意味着（或者必须防止）跨进程 / 机器边界的任何特定部署。它是用于示出功能组件的示意图。

[0082] CDP 体系结构的关键优点之一是它提供了实现的灵活性。这意味着以下两点 :

[0083] 1) 图 4 所示的某些组件是“移动的”，这意味着它们可以存活于不同的进程 / 层中。特别地，约束 / 安全引擎 112 一般存活于图 2 的存储进程 202 中。

[0084] 2) 不是图 4 中示出的所有组件都需要被实现以便具有全功能的数据平台。特别

地，对象高速缓存 414 可以仅由一个会话高速缓存组成。在另一个实现中，缓存 414 可以包括将与存储同步的显式高速缓存。查询处理器 406 对对象高速缓存 414 中的对象进行操作。

[0085] 在下文中更为详细地描述了 CDP 的若干个特征和 / 或组件。如上所述，CDP 的中心是公用数据模型 (CDM) 402，其中 CDM402 的意图是析出跨多个应用程序域公用的建模概念，这些应用程序从主要处理用户数据（例如 PIM、文件等）的应用程序到 LOB 和企业数据。一般而言，有两种可能的技术可用于实现这样的功能 :1) 专用于每个可想象的（或可想到为重要的）域的模型概念。例如，精确地定义“顾客”的含义（从 LOB 域）和“人员”的含义（从用户域）等等 ;以及 2) 提供应用程序设计者可以在其上创建它们自己的域专用类型、约束、关系的灵活基础。CDM402 使用第二种方法，这样使得它提供一组基本类型，并定义了灵活的框架用于创作新类型。在这种意义上，CDM402 可以是数据模型（例如它实际上定义了某些类型的它们的语义），且也可以是数据元模型（例如，它允许其他模型的规范）。

[0086] 以下讨论 CDM402 的某些特征，但是它们不应作为对本申请的限制。数据模型可以包括关系型数据模型。换言之，CDM402 展示了表、表上的行、查询和更新的概念。CDM402 可为数据定义比表和行更丰富的对象抽象。特别地，它允许使用诸如实体、实体间的关系、继承、包含及其集合等概念对真实世界的人工制品进行建模。此外，CDM402 可以通过将编程语言类型系统与其中建模的应用程序抽象紧密结合，将应用程序结构和存储结构之间的阻抗不匹配最小化。此外，可以提供对应用程序行为（例如方法、功能）和对行为的灵活部署的支持以便启用双层和多层应用程序。CDM402 也可以捕捉独立于底层物理存储的持久化语义，从而允许在各种各样存储上实现对 CDM402 的启用。

[0087] CDM402 可以调用多个概念。以下概念可由要实现的元模型使用来设计域专用数据模型。特别地，以下概念可以看作是 CDM402 的核心 :1) 实体类型可以是应用程序设计者对属性和方法的分组的规范，其中实体是实体类型的实例。可以理解，可以通过继承分层结构来组织实体类型 ;2) 表是实体的集合，它可以是其他实体的属性。使用实体、继承和表，应用程序可以递归地定义数据分层结构。表可以被强类型化，在此意义上，给定的表仅包括给定类型或其子类型的实体 ;3) 表集可以是其属性为表的实体。这是使用表和实体定义的递归数据分层结构的基本范例。它可以实质上类似于数据库的概念 ;以及 4) 关系可以表达实体之间的语义连接。应该理解，关系可以被扩展来定义关联、包含等。

[0088] 实体、关系和 / 或表集定义可以在例如模式的上下文中发生。为了本申请的目的，模式的主要目的是定义用于对在模式中定义的元素的名字定范围的名字空间。表集可以形成 CDM402 的“顶层”。可以通过创建表集来直接和 / 或间接地分配存储。例如，以下伪代码示出了表集的一个例子 :

[0089] <Schema Namespace =" MySchemas.MyLOB" >

[0090] <TableSetType Name =" LOBData" >

[0091] <Property Name =" Orders" Type =" Table(Order)" />

[0092] <Property Name =" Customers" Type =" Table(Customer)" />

[0093] <Property Name =" Products" Type =" Table(Product)" />

[0094] <Property Name =" Suppliers" Type =" Table(Supplier)" />

[0095] <Property Name =" PSLinks" Type =" Table(ProductSupplierLink)" />

[0096] </TableSetType>

[0097] <TableSet Name =" LOB" Type =" TableSetType" />

[0098] </Schema>

[0099] Entity(实体)类型可以具有与其相关联的属性和方法。为了指定属性类型、方法参数和方法返回值,CDM402 提供了若干内置类型 :1) 简单类型 :Int32、串、其他 CLR 值类型 ;2) 枚举类型 :等价于 CLR 枚举 ;3) 引用类型(在下文中讨论);以及 4) 数组类型 :经排序的内联类型集合(在下文中讨论)。这些内建类型的属性可以被分组在一起以形成内联类型,其中内联类型可以具有其他内联类型的成员。以下是上述的一个示例 :

[0100] <InlineType Name =" Address" >

[0101] <Property Name =" Linel" Type =" String" Nullable =" false" >

[0102] <Length Maxiumum =" 100" />

[0103] </Property>

[0104] <Property Name =" Line2" Type =" String" Nullable =" true" >

[0105] <Length Maxiumum =" 100" />

[0106] </Property>

[0107] <Property Name =" City" Type =" String" Nullable =" false" >

[0108] <Length Maxiumum =" 50" />

[0109] </Property>

[0110] <Property Name =" State" Type =" String" Nullable =" false" >

[0111] <Length Minimum =" 2" Maximum =" 2" />

[0112] </Property>

[0113] <Property Name =" ZipCode" Type =" String" Nullable =" false" >

[0114] <Length Minimum =" 5" Maximum =" 5" />

[0115] </Property>

[0116] </InlineType>

[0117] 可以通过利用内建类型和 / 或内联类型来构建实体。例如,以下伪代码示出了实体 :

[0118] <EntityType Name =" Customer" Key =" CustomerId" >

[0119] <Property Name =" CustomerId" Type =" String" Nullable =" false" >

[0120] <Length Minimum =" 10" Maximum =" 10" />

[0121] </Property>

[0122] <Property Name =" Name" Type =" String" Nullable =" false" >

[0123] <Length Maximum =" 200" />

[0124] </Property>

[0125] <Property Name =" Addresses" Type =" Array(Address)" >

[0126] <Occurs Minumum =" 1" Maximum =" 3" />

[0127] </Property>

[0128] <NavigationProperty Name =" Orders" Association =" OrderCustomer"

[0129] FromRole =" Customer" ToRole =" Orders" />

[0130] </EntityType>

[0131] 至少部分地基于由于表集是顶层组织单元，且表集是由表格组成的，因此实体（除表集之外）可以包含在表中。在表范围内，每个实体可以具有唯一的关键字值。在存储器级范围处，每个实体可以具有唯一的身份一其关键字值与其表的身份递归地串接。实体可以是 CDM402 中可以由关键字和 / 或身份引用的最小单位。存储操作可以以实体为目标，其中操作可以是，但不限于，持久化、存储、移动、复制、删除、重命名、备份、恢复等等。内联类型实例可以用在包含实体的上下文中。CDM402 可以定义抽象实体类型的概念，它们实质上类似于 CLR 中的抽象类。换言之，它们不能被直接例示；它们仅可被导出以创建其他可例示类型。

[0132] 实体引用可以被定义为对实体的持久的、持续的引用。引用值的范围包括实体身份。间接实体得出其引用，解除引用得出实体实例。引用的主要目的是启用实体共享：例如，对于 Ref(Customer) 属性而言，同一顾客的所有订单将实质上具有类似的值，这样使得定单 (order) 实体被认为是共享顾客 (customer) 实体（例如以下代码示例中代码的第六行是一个示例）。

[0133] 与 CDM402 相关联的数据在其组成部分之间有关系。关系模型并不明确地支持关系；PK/FK/ 引用完整性提供了以有限的方式实现关系的工具。然而，CDM402 支持使用关联和合成对关系的显式概念建模。示出以下示例以便理解关联和合成的性能：

[0134] 1.<EntityType Name =" Order" Key =" OrderId" >

[0135] 2. <Property Name = " OrderId " Type = " String " Nullable =" false" >

[0136] 3. <Length Minimum =" 10" Maximum =" 10" />

[0137] 4. </Property>

[0138] 5. <Property Name = " Date " Type = " DateTime " Nullable =" false" />

[0139] 6. <Property Name =" Customer" Type =" Ref(Customer)"

[0140] 7. Association =" OrderCustomer" />

[0141] 8. <Property Name =" Lines" Type =" Table(OrderLine)"

[0142] 9. Composition =" OrderOrderLine" />

[0143] 10. <Property Name =" ShippingAddress" Type =" Address"

[0144] Nullable =" false" />

[0145] 11.</EntityType>

[0146] 12.<Association Name =”OrderCustomer”>

[0147] 13. <End Role =”OrderRole”Type =”Order”Multiplicity =”*”

[0148] 14. Table =”SalesData.Customers”/>

[0149] 15. <End Role =”CustomerRole”Type =”Customer”Multiplicity =”1”/>

[0150] 16. <Reference FromRole =”OrderRole”ToRole =”CustomerRole”

[0151] Property =”Customer”/>

[0152] 17.</Association>

[0153] 18.<Composition Name =" OrderOrderLine" >

[0154] 19. <ParentEnd Role = " Order " Type = " Order " Property

＝" Lines" />

[0155] 20. <ChildEnd Role ＝" OrderLine" Type ＝" OrderLine"

[0156] Multiplicity ＝" 100" />

[0157] 21.</Composition>

[0158] 关联可以表示实体之间的对等关系。在以上示例中，订单经由关联与顾客相关。在上述代码示例中，第 6 行示出了订单具有相关联的顾客（它是由引用属性 Customer 来指定的）。关联的特性在 12-15 行定义：它表示 OrderCustomer 关联是从 Order 到 Customer(15 行）；它也表示对于每个 Customer（在 14 行，Multiplicity ＝“1”），可以有多个 Order（在 13 行，Multiplicity ＝“*”）。上述的关联类型可以被称为引用关联。

[0159] CDM402 定义了两种其他类型的关联：值关联和关联实体的关联。值关联允许通过任何属性表达关系，而非仅通过身份引用（例如 Document.Author 属性经由相等条件与 Contact.Name 相关）。关联实体允许关系的建模，其中关系本身带有一些数据（例如在公司和个人之间的雇佣关系带有如雇佣期限或在公司中人员等级和职位等属性）。

[0160] 合成可以表示父子关系和 / 或包含关系。考虑 Order 和 OrderLine（订单线）（例如 Order 是在网站上放入购物车的货物总数；OrderLine 是车中的每个个别项一书、DVD 等）。每个 OrderLine 只有在 Order 的环境下才有意义。OrderLine 不可能在包含 Order 之外独立存在。换言之，OrderLine 包含在 Order 中，其使用期是由 Order 的使用期确定的。

[0161] 上述关系可以使用合成来建模。行 8 示出了合成的一个示例。Line 属性和 OrderOrderLine 合成（18-22 行）表示订单控制其线，且线依赖于包含它们的订单。应该理解订单是父，线是子。合成和内联类型的主要区别是合成涉及实体。换言之，OrderLine 可能是引用的目标，而内联类型不能在上述示例中。

[0162] CDM402 及其关系的显式建模的一个好处是它提供了对查询的元数据支持。也可以使用上行查询。例如，给定一顾客，可以通过在 CDM402 中实现 NavigationProperty（导航属性）来找到所有的订单（而无需存储显式反向指针）。这在上述代码段的 28 行示出，且为方便起见在以下复制：

[0163] 28.<EntityType Name ＝" Customer" Key ＝" CustomerId" >

[0164] 29. <NavigationProperty Name ＝ " Orders " Association ＝" OrderCustomer"

[0165] FromRole ＝" Customer" ToRole ＝" Orders" />

[0166] 30.</EntityType

[0167] 持久化引擎 408 可以包括对象 - 关系映射。换言之，CDP 提供的建模、访问和查询抽象是基于对象的。CDP 所使用的主要存储技术是基于关系的（例如 SQL2000）。持久化引擎 408 使用对象 - 关系映射（也称为“O-R 映射”），其中持久化引擎可以将语言类映射到底层表式表示。

[0168] 当考虑 O-R 映射，持久化引擎 408 可以提供两种情况：1) 命令性 O-R 映射；以及 2) 非命令性 O-R 映射。命令性 O-R 映射是 CDP 类型之间的映射，其中它们的关系表示可以被硬编码到 CDP 中。类型设计者在选择底层表的格式时具有很小和 / 或没有灵活性。其一个示例可以是基于数据库的文件存储系统。非命令性 O-R 映射是开发人员在选择 CLR 类如何映射到底层存储结构时具有不同程度的灵活性的映射。可以考虑两种子情况。1) 将现

有关系模式展示为对象。类型设计者使用高级规范语言来设计 CDM 类型、使用工具来生成基于它们的类、使用灵活性来指定类型如何映射到表。当 CDP 应用程序与现有关系型应用程序并排开发时（在使用实质上类似数据的意义上），发生这种情况。例如，汽车公司的 IT 部门可以具有 LOB 应用程序，其中它希望编写针对同一数据的 CDP 应用程序（可能是逐步迁移策略的一部分）。但是要求是 LOB 应用程序和新的 CDP 应用程序一起针对同一数据运行。2) 将类集合持久保存到关系模式中。开发人员不使用生成的类；而是使用自己设计的类。开发人员希望将这些类映射到关系型模式。应该理解，有很多产生这种要求的情况。

[0169] CDP 还可以包括可以在设计期间使用的编程接口（未示出）。可以使得编程接口对 CDP 应用程序设计者和 / 或程序员可用。编程接口可以被类型化为三种通用方面：1) 设计时编程工具（例如，用于使得类型设计者能够创作 CDM 类型及其约束，从这些类型生成 CLP 类，以及将行为添加到类型中的工具）；2)API（例如，用于编写 CDP 应用程序的类和方法）；以及 3) 查询（例如，用于查询诸如实体实例等 CDM 对象的语言）。这些编程表面的组件协同工作，以便提供对底层存储数据的强类型化的、基于对象的抽象。

[0170] CDP 提供了声明性的公共模式定义语言 (CSDL)，它类似于 SQL 的数据定义语言或 C# 的类定义，用于定义实体类型、实体表、实体类型之间的关系以及约束。有三种主要的设计时组件：

[0171] 1.API 生成器。应用程序设计者使用 CSDL 设计 CDM 类型和关系，并使用称为 APIG（发音是 ay-pig) 的设计时 CDP 工具，该工具生成对应于这些类型和关系的部分 CLR 类。由 APIG 生成的类作为程序集对应用程序编程者可用，并且可以由它们的应用程序用 C# 的 using 子句来引用。在某种意义上，由 APIG 生成的类是规范类；它们可以是应用程序内的 CDM 类型的直接表示。在一个示例中，应用程序类可以在它们的定义中有约束—诸如当应用程序使用预先编写的类库（图包、数学包等）的类时。应用程序可以使用 CDP 的对象持久化框架在存储中持久地存储并查询这些类的实例。这些对象可以被称为无格式普通 CLR 对象，即 POCO。CDP 也支持 POCO 的情况。

[0172] 2. 关系 - 对象映射。CSDL 的该组件帮助应用程序设计者声明存储概念之间具体的、非命令性的映射，诸如表和视图以及 CLR 类。它也指定了如何能够将按照 CDM402 定义的约束映射到 SOL 声明性约束、触发器或存储过程。

[0173] 3. 行为。CSDL 使得应用程序设计者能够确定业务逻辑的哪个部分被实现为实例方法、静态函数、存储过程。它也确定了其上可以运行逻辑的层（例如 CDP 运行库与存储）。

[0174] 编程接口还可以包括 CDP、API，可以对照该 API 来编写编程表面应用程序。CDP API 可以具有三个子部分：

[0175] 1. 一般 CDP 数据访问。这是 API 中展示存储、会话、事务（例如 StorageContext）、查询服务（例如 StorageSearcher）和 CRUD 服务（例如 SaveChanges）的部分。

[0176] 2.CDM 数据类。这是规范的、应用程序无关的类，它展示了诸如实体、关系、扩展等 CDM 概念。

[0177] 3. 域数据类。这是应用程序 / 框架专用类，诸如 Contact（联系人）、Message（消息）、PurchaseOrders（采购订单）等，它们符合 CDM402，但是具有域专用的属性和行为。

[0178] CDM402 也可以定义查询语言，即 CQL。CQL 被设计成允许对程序员处理的对象结构进行丰富查询。以下是用作 CQL 形式基础的三个标识的技术：

[0179] 1.OPath :OPath 语言的根源在 SQL 和 XPath 中，且被设计成 XPath 的 CLR 对象版本。设计构建在路径表达式的 XPath 概念之上，以便揭示出依次解除对象属性的引用的方法。设计是基于一个简单的原理 :开发人员希望看到对象集合在面向对象的 API 中作为主要“结构”构造。OPath 可以是基于数据库文件存储系统的 POR 查询形式。

[0180] 2. 对象 SQL :该方法扩展了 SQL 查询语言，以操作 CDM 对象的图和集合。Windows 查询语言 (WinQL) 是被设计成 SQL 查询和操作 CRL 对象的图的 SQL 的变体，它是 SQL 中所需的扩展的候选设计。

[0181] 3.C# 序列操作符 :这是用于强类型化的、编译时检查的查询和可以应用于 CLR 对象的暂时或持久集合的宽泛类的设置操作（例如，经由关系 - 对象映射）的一组 C# 扩展。

[0182] 从策略上而言，C# 序列操作符方法对于成为 CQL 的框架是最有意义的。CQL 是一种查询语言。创建、更新、删除是作为对象操作（新的、属性设置器等）来执行的。持久化引擎 408 中的 O-R 映射组件可以将这些操作映射到 SQL 的底层 DML 操作中。

[0183] 以下描述 CDM 类型和编程表面之间的关系。在 CDM402 中“类型”的概念可以从三个不同的级别来看 :

[0184] 1. 模式空间 :在 CDM 模式中对类型的描述。这些是抽象类型，意为它们在运行时栈的任何组件中可能不会被显式地物化（例如从应用程序一直到存储）。

[0185] 2. 应用程序空间 :在 CDP API 中将类型表示为 CLR 类。在模式空间中的实体 / 内联类型和应用程序空间中的数据类之间可以是 1-1 的对应关系。换言之，CDM 方案中的每一实体和内联类型可导致一个 CLR 类。通常，这些类由 APIG 自动生成 ;然而，在 POCO 的情况下，开发人员可以明确地指定在 CLR 类和模式空间中的类型之间的映射。除了用于实体和内联类型的类之外，应用程序空间也可以包括关系类。

[0186] 3. 存储空间 :底层存储中类型的持久格式。如果存储是关系型存储，那么这些类型是表 /UDT/ 核心 SQL 类型。CDP 的 O-R 映射组件支持允许将模式空间中的类型映射到存储空间中的类型的映射模式（例如采购订单实体类型可以被映射到 SQL 服务器的 PurchaseOrder 表中）。CDP 查询语言以应用程序空间为目标。这是有意义的，因为开发人员期望使用实质上类似于他们用于其他操作（例如对象和集合）的抽象来查询。然而，使用 CDM 抽象（模式空间）来描述 CQL 的语义。

[0187] CDP 也可以包括约束 / 安全性 112。当在数据的较大语义上下文中检查时，几乎所有的数据会在其类型域上有某一或另一形式的约束。对于 CDP 而言，为类型和应用程序设计者提供一种表达这些约束的方式是很重要的。CSDL 可以用于在类型设计时声明性地创作约束。约束的示例包括但不限于 :1) 简单类型约束，诸如长度、精度、比例、默认值以及校验 ;2) 数组类型约束，诸如元素约束、出现、唯一以及校验 ;以及 3) 属性约束等。

[0188] 这些约束可以在运行时由 CDP 约束引擎来强制实施。注意，真正符合 CDM402 的动作意味着一组可以从底层关系型存储级得出的约束。例如，CDM402 要求“每个实体在其约束表范围内具有唯一的关键字”。在存储级，这被转换成唯一关键字约束。这样的约束有若干个其他示例。这里的要点是 CDP 约束引擎强制实施两种类型的约束 :由 CDM402 所隐含（和符合 CDM402 所需）的约束，以及由类型设计者创作的约束。除了在 CSDL 中创作的声明性约束之外，可以使用 SQLServer 的存储过程来编写约束。相比声明性语言所可能表达的，这种方法允许表达更为复杂的约束。

[0189] 此外，约束 / 安全性 112 可以提供安全模型，以便控制对 CDP 中的实体的访问。用于 CDP 的安全模型必须至少满足以下情况 ：

[0190] 认证 ：安全模型可以支持认证操作系统用户。这包括域、工作组或断开的客户机中的用户。它也可以包括对基于 NTLM 和 Kerbero 的认证的支持。

[0191] 授权 ：CDP 安全模型可以至少在实体级支持安全授权。它也必须允许管理对实体的读取和更新的单独的权限。至少，约束 / 安全 112 规定实体的“属性”和 / 或属性集被视为实体的安全标识符。实体的访问权限是由与将安全标识符作为参数的表相关联的函数来确定的。CDP 应该也允许分别从能够改变剩余实体的用户中规定能够改变安全标识符的用户。应该理解，CDP 可以支持基于更为通用的角色的模型，这也允许与仅读和写不同的权限。

[0192] CDP 运行库 110 维护查询结果（例如在下文中详细讨论的指针）和未被提交的更新的高速缓存 414（例如对象高速缓存 414），其中这样的高速缓存可以被称为会话高速缓存，因为它绑定到会话、事务 404。此外，它在会话开始创建时开始存在，在会话终止时消失。CDP 会话被封装在 StorageContext（存储上下文）对象中。应用程序可以例示 StorageContext 的多个实例，由此启动多个会话并因此启动多个会话高速缓存。CDP 也可以展示另一种类型的高速缓存，称为显式高速缓存。显式高速缓存为来自一个或多个查询的数据提供了高速缓存。一旦数据被物化到显式高速缓存中，就可以提供以下数据一致性保证 ：1) 只读、非授权性 ；2) 直写、授权性 ；以及 3) 经由外部通知自动刷新。对于显式高速缓存的编程和查询模型可以实质上类似于存储数据的模型。

[0193] 指针、规则 410 是允许一次处理一个从 CQL 返回的数据实体集的机制。应用程序可以通过简单地将整个结果集复制到存储器中并在存储器结构中其顶部覆盖滚动模式在结果集上创建指针。但是该要求的普遍性以及在实现指针时有时涉及的复杂性（尤其当考虑更新、分页等时）意味着任何数据平台都应该提供指针模型。

[0194] CDP 同时提供仅前向和可滚动指针。处浏览和滚动的基本功能之外，CDP 指针提供以下特征 ：1) 外部通知和维护 ；2) 对展开 / 折叠状态进行多级分组 ；以及 3) 动态排序和过滤（例如“后处理”）。应该理解，指针可以不是指定结果集的不同机制 ；结果集是由查询指定的，且指针是在这些查询之上。

[0195] CDP 也可以包括业务逻辑主机 416。当多个应用程序操纵实质上类似的数据时，关键的要求是确保数据保持可信性一即，确保数据符合各种确认规则、业务规则和由类型设计者和 / 或数据持有人创立的校验和平衡的任何其他系统。应用程序一般是不可信的是很好的假设。由于无法预料的使用模式的愚蠢的、恶意的和 / 或简单的紧急需要，应用程序保存和 / 或试图保存无效值。例如，用户可以输入 292 作为区域代码，即使 292 是无效的区域代码且由此电话号码字段中的值不再表示电话号码，应用程序也保存该数字。换言之，它不能被“信任”为电话号码。防止这一现象的通常做法是创建信任边界 ：声明性规则 / 确认代码 / 等的某一主体（通常被称为业务逻辑），它在单独的进程中运行，且检查由应用程序作出的数据改变以便“批准”这样的改变。接着它可以将这些改变保存到存储中。许多时候，业务逻辑不限于检查和批准 ；它也强制实施业务规则，使得工作流发生等（例如当插入新顾客时，应该发送电子邮件给信用检查部分以便确保信用价值）。

[0196] CDP 提供用于创作业务逻辑 (BL) 的若干机制。这些机制可以被分成以下 5 类 ：约束、事件处理程序、静态 / 实例方法、可绑定行为以及境外服务方法，它们中的每一个会在

下文中更为详细地讨论。如上文所述的约束 / 安全性 112 可以是声明性的和程序上的。这些约束可以在存储上接近于数据而执行。因此，约束 112 被认为在信任边界之内。此外，这些约束也可以由类型设计者来创作。

[0197] 业务逻辑主机 416 可以采用事件处理程序。CDP API 对数据改变操作引发若干事件。BL 作者可以经由处理程序代码挂钩到这些事件。例如，考虑订单管理应用程序。当有新订单时，应用程序需要确保订单值小于对该客户授权的信用限制。这个逻辑可以是在将订单插入到存储中之前运行的事件处理程序代码的一部分。

[0198] 一般而言，可以有以下类型的事件 :1) 确认（例如，这些事件为感兴趣的一方提供检查所建议的值并确认它的机会）;2) 预保存（例如，正好在将改变保存到存储之前引发该事件，且该事件可以在意向和行为上实质上类似于 SQLServer 中的“BEFORE”触发器）;以及 3) 后保存（例如，该事件正好在将改变保存到存储之后引发，且可以在意向和行为上实质上类似于 SQLServer 中的 AFTER 触发器）。这种类型的 BL 在 CDP 中运行，由此可以在其上部署 CDP 的任一层上运行。因此，当它在客户机层上运行时，它可以被其他应用程序绕过（例如，它不在信任边界内运行）。

[0199] 此外，业务逻辑主机 416 可以调用静态 / 实例方法。为 CDM 类型自动生成的类是局部类。类型设计者可以通过对它们添加附加的方法来完成这些局部类，一般用于实现对一个或一组特定类型有意义的逻辑。考虑以下例子 :person.GetOnlineStatus()，其中 person（个人）是 Person 类型实例 ;emailAddr.IsValidAddress()，其中 emailAddr（电子邮件地址）是 SMTPEmailAddress(SMTP 电子邮件地址）类型的实例 ;等。由于其真正的特性，这种类型的 BL 是不可强制实施的 ;例如，要靠应用程序调用 IsValidAddress() 来确保有效性。它运行在其上部署 CDP 的任一层上。因此，当 CDP 在客户机层上时，它不在信任边界内运行。

[0200] 可绑定行为是允许类型设计者为第三方扩展创建插入点的代码编写模式。经典示例是用于电子邮件消息的类型。不同的电子邮件程序可以在给定的机器上运行。每个程序希望使用公用消息类型，但是每个程序也需要定制 SendMessage（发送消息）方法的行为。类型设计者通过定义 SendMessage 方法的基本行为，并允许第三方提供指向实现的指针来完成这一过程。可绑定行为也可以在其上部署 CDP 的任一层上运行。因此，当 CDP 在客户机层上时，它不在信任边界内运行。

[0201] 静态服务方法是在中间层上编写和部署的 BL，且对客户机层而言是远程的。换言之，BL 作为中间层上的 web 服务运行。例如，考虑提供诸如 CreateAppointment()、GetFreeBusy() 等的日历管理服务。这些服务（“静态服务方法”）是使用 CDP 来实现的，且 web 服务部署在中间层上。客户机层具有应用程序用于使用通道（以下讨论）来调用这些服务的 web 服务代理。这种类型的 BL 可以在中间层上运行且在信任边界之内。

[0202] 应该理解，组件化的体系结构使得 CDP 可能保持存储逻辑在一定程度上不可知。诸如对象高速缓存、指针、会话、事务等 CDP 特征使用 CDP 级抽象。到底层存储抽象的映射在 O-R 映射和持久层中发生。通过重写映射逻辑，CDP 可以在不同的存储中实现。

[0203] 图 5 示出了 CDP 的各种组件中的数据流。使用以下示例检查各种组件之间响应于应用程序 500（类似于应用程序 206 和应用程序 400）的方法调用的交互是启发性的。

[0204] 1.void AddToCart(String customerId,String productId)

[0205] 2.{

[0206] 3. using(OrderData od = new OrderData())

[0207] 4. {

[0208] 5. ShoppingCart cart = od.ShoppingCarts.Searcher.Filter(

[0209] 6. " CustomerId = {0}" ,customerId).GetFirst() ;

[0210] 7. if(cart == null)

[0211] 8. throw new Exception(“No shopping cart”) ;

[0212] 9. Product product = od.Products.Searcher.Filter(

[0213] 10. " ProductId = {0}" ,productId).GetFirst() ;

[0214] 11. if(product == null)throw new Exception(“Missing product) ;

[0215] 12. cart.Products.Add(product) ;

[0216] 13. od.SaveChanges() ;

[0217] 14.}

[0218] 15. }

[0219] 该例子对持久 ShoppingCart(购物车)添加了一个项。举例而言,设想该方法是作为处理 ASP.NET 网页的一部分来调用的。

[0220] 第 3 行:创建存储上下文。StorageContext 是由应用程序 500 创建的 OrderData(订单数据)对象来封装的。OrderData 类可以表示在 CDM 模式中描述的表集类型。OrderData 对象创建被配置为与存储 202 交互所必需的 StorageContext 对象。StorageContext 的初始化代码可以是打开到存储 202 的连接的运行时会话和事务组件 404 的一部分,并且完成发起会话和创建事务上下文所必需的工作。安全上下文在约束 / 安全组件 112 中建立。最后,StorageContext 的实例由 API108 返回给应用程序 500。在 2 层情况下,获取 StorageContext 导致到存储 202 的连接。应该理解,在 3 层部署中的连接可以稍有不同。

[0221] 第 5 行:查询。第 5 行中表达式的右侧是 OPath 查询。持久化和查询引擎 408 展示对用于基于 CDM 查询检索对象的方法的基本接口。CDM402 中的 CDM 实现调用具有特定 OPath 的方法。持久化和查询引擎 408 将查询映射到 SQL,并通过连线将其作为 TDS 有效负载发送。约束 / 安全组件 112 确保适当地应用了安全性且应用程序 / 用户只看到它们被允许看到的数据。存储执行查询并将结果返回到 CDP 运行库 110。CDM402 和持久化 / 查询引擎 408 协同工作,以混合来自 TDS 结果的对象,且这些对象被置于对象高速缓存 414(例如会话高速缓存)中。结果是 API108 将 ShoppingCart 对象返回给应用程序 500。

[0222] 第 9 行:查询。既不是这个查询也不是先前导致创建任何指针的那个查询 (GetFirst() 方法本质上应用于查询的“第 1 个”子句)。然而,如果查询要求创建指针,那么指针 / 规则组件 410 执行这个操作。

[0223] 第 12 行:更新:对象高速缓存 414 中的 ShoppingCart 对象在指定的 Product(产品)中更新。

[0224] 第 13 行:刷新改变。在 OrderDate 对象上的 SaveChanges() 的实现调用经封装的 StorageContext 对象上的 SaveChanges()。StorageContext.SaveChanges() 是业务逻辑主机组件 416 的部分。这涉及以下步骤。首先,运行预保存逻辑。接着,运行确认代码,之后是

后保存过程。确认代码被挂钩到由 CDPAPI108 定义的事件。注意，在另一实现中，确认代码可以被挂钩到对象的设置器。接着，运行预保存代码。该代码被挂钩到由 CDPAPI108 定义的事件。将改变写入到存储中。首先，主机组件 416 与对象高速缓存 414 协同工作以便获取改变向量，该改变向量含有在该存储上下文中作出的所有改变。持久化引擎 408 展示被称为 IPersist（持久化）的接口，它是诸如 Write(< 改变向量 >_) 等方法的基本接口。主机组件 416 从持久化引擎 408 获取 IPersist 并用改变向量来调用 Ipersist.Write()。持久化引擎 408 将写请求映射到适当的 SQL 更新（实际的 UPDATE 语句或存储过程调用）中，并使用该更新将改变写入到存储 202 中。在这个过程中，约束 / 安全性组件 112 确保完成了适当的安全强制实施。它也运行任何约束逻辑。最后，运行后保存代码。该代码被挂钩到由 CDPAPI108 定义的事件。

[0225] 注意，业务逻辑的运行会导致对高速缓存 414 中对象的改变。这些改变通过对 myStorageContext.SaveChange() 的调用来持久保存在存储 202 中，以确保不会绕过业务逻辑 416。多个 ISV（独立支持销售商）可能希望对数据改变运行逻辑，在这种情况下他们将其处理程序挂钩到事件和由 CLR 以 FIFO（先进先出）顺序调用的处理程序。在这个示例中，业务逻辑 416 主存 ISV 验证、预保存和后保存逻辑。

[0226] 图 6 示出了可以用 CDP 实现的各种框架。CDP 是被设计成可跨各中指定的垂直域（诸如用户数据、LOB 数据等）使用的数据平台。CDM 提供了域不可知数据模型，它足够丰富来表达域专用结构和语义，但同时足够普通以便跨不同的域使用。各种 CDP 特征是基于 CDM 的，且跨所有域的应用程序可用。

[0227] 对照 CDP 编写的所有应用程序的总体可以被分成以下三类：

[0228] 1. 框架：框架使用由 CDP 提供的可扩展机制以便为特定的域定制 CDP。框架将值以及类型规范和附加服务添加到 CDP。然而，展示给应用程序的编程模型是 CDP 编程模型；特别地，应用程序仍然使用数据类、StroageContext、StorageSearcher 以及 CQL。基于数据库的文件存储系统可以是 CDP 之上的框架的一个示例，它是为用户数据域定制的。

[0229] 2. 垂直平台：CDP 之上的单独的层，具有其自己的 API、抽象和数据模型。它隐藏了 CDP，并将整个不同的编程模型展示给应用程序。例如，与电子邮件结合使用的应用程序可以使用 CDP，但是为其用户展示电子邮件对象模型。

[0230] 3. “常规”应用程序：仅仅是旨在完成一组特定任务的 CDP 应用程序。

[0231] 它不专用于任何 CDP 类型，或展示编程模型，或使用任何框架或垂直平台。

[0232] 垂直平台和“常规”应用程序只是代码；它们可以用任何方式使用 CDP，而不带有喜好或偏见。框架有一些不同；由于它们将值添加到 CDP 而不将它向应用程序隐藏，它们可以遵守以下规则：

[0233] 1. 框架数据模型等同于 CDM，或者是简单的、很好地证明的 CDM 规范。它可以定义新的类型，但是这些类型是以实体作为最终的超类。

[0234] 2. 框架会对现有 CDM 类型定义附加约束和 / 或使用 CSDL 来创作新的约束。换言之，对于约束定义必须使用 CDM 方法来表达约束。

[0235] 3. 框架通常不展示它们自己的查询语言；即使它们这么做，也是作为对 CQL 的附加，而不是代替 CQL。

[0236] 4. 框架通常不展示它们自己的编程模型；即使它们这么做，也是作为 CDP API 的

附加，而不是代替 CDP API。

[0237] 5. 框架在 CDP 之上提供附加的专用服务。这些服务可以被实现为 CDP 业务逻辑或附加的助手类和方法。

[0238] 应该理解，所有上述规则旨在确保由特定框架保存到 CDP 中的数据对所有应用程序可访问，而不管应用程序是否使用该框架。

[0239] 图 6 示出了在 CDP 层 602 之上的三个 (3) 框架：用户应用程序框架 (UAF)604（例如，基于数据库的文件存储系统、WinFS 等）、协作框架 608（诸如 WSS) 以及业务框架 610(BF)（例如 LOB 框架）。属于每种框架的数据以相同的模式示出为框架块。例如 UAF604 具有数据联系人 618 和项 620；协作框架 608 具有数据文档库 622；而 BF610 具有数据订单 624。注意，所有这些类型最终是以实体 626 作为超类的。

[0240] 图 6 也示出了应用层中的三个 (3) 应用程序：联系人管理应用程序 612、协作应用程序 614（诸如电子邮件应用程序）以及顾客关系管理 (CRM) 应用程序 616。联系人管理应用程序 612 完全用来自 UAF604 的数据工作；CRM 应用程序 616 用来自 UAF614 和 BF610 两者的数据工作；而协作应用程序 614 用来自所有三个框架（如，UAF604、协作框架 608 和 BF610) 的数据工作。

[0241] 图 7 示出了允许多个应用程序共享数据的公用的、基于数据库文件存储系统环境。换言之，图 7 示出了多个应用程序使用单个框架。CDP 组件和存储组件 702（在图 7 中示为 CDP+ 存储）可以用作操作系统的单个数据平台，它可以由任何和所有应用程序充分利用。优点（如上所述）是丰富建模、数据透明度以及数据共享。这些优点将在下文中作出更详细的描述。

[0242] CDM 提供了可用于描述一组完全不同的应用程序和情形所需类型的灵活的建模环境。例如，用户数据（例如文档、文件、照片、音乐 ...）、LOB 数据（例如顾客、订单、订单细节 ...）、PIM 数据（例如联系人、电子邮件、日历、任务 ...）都可以使用 CDM 来建模。跨越结构化、半结构化和非结构化数据且也跨越垂直域的这种类型的丰富建模使得单个应用程序可能使用公用抽象和查询语言用不同类型的数据来工作。换言之，CDP 可以用作一个存储。

[0243] CDP 可用作所有应用程序都能充分利用的单个数据平台。此外，使用 CDP 存储的数据可被所有的应用程序用于操作（例如，服从安全策略）。考虑以下情况：每个应用程序以对除了其本身（例如，存储了数据的应用程序）以外的其他应用程序都不透明的格式来存储数据。给出两个示例：电子邮件收件箱的内容对除了电子邮件应用程序外的所有其他应用程序都是不透明的；CRM 应用程序有一组详细的模式，它覆盖在创建诸如顾客、案例等的抽象的表之上——因此使“顾客”的概念对所有其他应用程序是不透明的（例如，除非应用程序知道 CRM 应用程序使用的模式）。

[0244] 显然，在电子邮件应用程序中有概念上与 CRM 应用程序存储的数据相似的数据——一个示例是联系人信息。据用户所关注的，联系人是 Contact；从这个观点出发，很难理解为何相同的联系人信息被存储两次，一次在 CRM 中，一次在电子邮件收件箱中。这里的问题不只是多余的存储，而是所有暗示的不规则——使得更新发生在两个地点，协调删除和确保插入发生在两个地点等等。考虑当电子邮件应用程序和 CRM 应用程序都被构建在 CDP 存储 702 上时将发生什么情况。使用 CDM，联系人类型可从实体类型导出，并且其结构

对电子邮件应用程序和 CRM 应用程序变为透明的。因此，只要两个应用程序在类型的模式上达成一致，则完全不同的应用程序可使用彼此的数据而不需要知道彼此的存在。由于 CDP 提供一个公用查询模型，因此（例如）CRM 应用程序可查询联系人数据，而忽略联系人的一个特定实例是否“属于”它。

[0245] 丰富建模的组合，数据透明度和平台框架体系结构允许涉及多个应用程序和框架的组合的许多共享 / 互用的情形。应该理解，术语共享可以指只要数据存储在 CDP 中，就能够不考虑数据存储在哪个应用程序中和 / 或使用哪种框架来存储数据而使用数据的应用程序。

[0246] 特别地，图 7 示出了一个常见的 UAF 情形，其中多个应用程序共享数据，在这种情况下，数据是从项 706 中导出的一组 UAF 类型。CDP 和存储 792 可包括与 UAF 框架 704 有关的一组 UAF 类型。该组 UAF 类型可从项 706 导出，其中该组可包括电子邮件 708、文档 710 以及联系人 712。还应该理解，项 706 可从实体 714 导出。多个应用程序可与 CDP 和 UAF 框架 704 结合使用，诸如但不限于，电子邮件应用程序 716、丰富回避 (evite) 客户机以及项目 M720。应该理解，对应用程序、CDP、UAF 所驻留的层没有约束。例如，图 7 的应用程序中的一个可在中间层执行和 / 或运行（例如，协作应用程序）。

[0247] 图 8 示出了根据 CDP 和相关联体系结构使用多个框架的单个应用程序。CDP 和存储 702 可向操作系统提供所有应用程序充分利用的数据平台。主要在 LOB 框架 804 之上编写的 CRM 应用程序 802 可使用与 UAF 框架 808 相关联的联系人数据 806。应该理解，CRM 应用程序 802 通常使用与其相关联的数据，诸如但不限于，订单细节 814 和购买订单 816。CRM 应用程序 802 在使用 UAF 数据（例如，联系人数据 806、项 810、实体 812 等）时可使用 CDP 级抽象。换言之，CRM 应用程序 802 不需要使用 UAF 框架 808 方法。此外，应该理解和明白，CRM 应用程序 802 可以驻留在一层上。

[0248] 图 9 示出了 CDP 与关联于多个完全不同的框架的多个应用程序共享数据。图 9 描述了三个框架，UAF 框架 904、协作框架 908 和在 CDP902 之上的 BF 框架 910。多个应用程序可使用框架级和 CDP 级编程的组合。特别地，联系人管理应用程序 912、协作应用程序 914 和 CRM 应用程序 916 可使用框架级和 CDP 级编程的组合。CDP902 提供与多个完全不同的框架相关联的应用程序以共享存储 928 中的数据。

[0249] 特别地，存在多个应用程序与数据交互的各种方式。联系人管理应用程序 912 可使用 CQL 来查询联系人 918；它可以使用 UAF904 方法，诸如项级移动、复制、contact.GetBestEAddress() 等。联系人管理应用程序 912 还可使用核心 CDP 运行库类，诸如但不限于，StorageContext、StorageSearcher，以及 CDP 数据类（例如，联系人类和相关联的获取器和设置器）。

[0250] 协作应用程序 914 可使用 CQL 来查询联系人 918、文档库 922 中的任何文档、甚至可能查询订单 924。协作应用程序 914 不需要知道 UAF904 和 / 或 BF910 的存在来完成这样的查询——它可以完全在 CDP 级完成，而不需要使用由其他框架编写的任何特殊代码。它使用协作框架 908 的操作细节来操纵文档库 922，诸如 AddDocumentToDocLib(< 文档 >,< 文档库 >) 等。协作应用程序 914 还可使用 CDP 级类，诸如 StorageContext、StorageSearcher（存储搜索器）、Contact、Order、DocLibrary（文档库）和相关联设置器和获取器。

[0251] CRM 应用程序 916 使用 CQL 来按照给定的联系人查询所有的订单。应该理解，CRM

应用程序 916 可以在不知道实际使用 UAF904 创建的联系人的情况下进行这个查询。它使用 BF910 提供的方法和服务（例如，FindShipStatus(< 订单 >)）来操纵订单。它还可使用 CDP 级类，诸如 StorageContext、StorageSearcher、Contact、Order 和相关联的设置器和获取器。

[0252] 当共享非 CDP 存储时，重要的是注意，CDP 不使用供应者模型，由此任意数据源可表现为 CDP 存储。当 CDP/ 框架应用程序想要用非 CDP 存储中的数据来工作时，它可以使用两个选项 :1) 使用同步适配器体系结构 (UAF 的一部分）以将该数据同步到 CDP 存储中 ;2) 构建自定义逻辑以与非 CDP 存储集成。

[0253] 图 10 示出了 CDP 的一种双层部署。构成 CDP 的各种组件在某种意义上是移动的。采用某个限制，它们可跨不同进程和机器边界部署，结果是 2 层、3 层和 N 层（这里 N 是一个大于或等于 1 的整数）配置。应该理解和明白，虽然示出了 2 层部署，但是本发明并不如此局限，并且可采用任意层数的配置。

[0254] 特别地，CDP API1002 和 CDP 运行库 1004 可以都在与应用程序 1006 相关的应用程序进程之中。因此，CDP 组件（例如，CDP 运行库 1004、API1002 和约束 / 安全性 1008）可存在于各个层中。例如，API1002、CDP 运行库 1004 和应用程序 1006 可存在于客户机层 1010 中，其中的组件可存在于其自己的进程 / 机器边界中。另外，存储 1012 和约束 / 安全性 1008 可存在于服务器层 1014 中，其中的组件可存在于其各自的进程 / 机器边界中。应该理解，约束 / 安全性 1008 可被主存在存储进程中，而剩下的 CDP 组件可在客户机进程中。这是为何 CDP 组件被认为是可移动的一个主要示例。

[0255] 图 11 示出了根据本发明的一方面具有共享数据的一种双层部署。下文讨论的第一配置是当多个应用程序共享同一数据的时候。这不是说应用程序必须共享这些数据 ;而是说任何应用程序的数据对其他应用程序都是可用的。同时注意，数据的可用性是在应用程序的上下文中而不是用户上下文中的，因此这和用户凭证的概念是截然不同的。CDP 运行库的约束 / 安全性模块不需要借助应用程序就可以处理。

[0256] 应用程序与 API 和 CDP 运行库可以交互，其中各种应用程序可分别与各自的组件一起存在，因此每一应用程序、API 和 CDP 运行库可以有它们自己的机器 / 进程边界，被示为边界 1102、边界 1104 和边界 1006。简化起见，示出了三个应用程序（例如，应用程序 1、应用程序 2 和应用程序 3），然而应该理解，可以采用任意数目的应用程序。应用程序可在其自己进程 / 机器边界 1112 中访问存储 1110 中的共享数据 1108。应该理解，在完全不同的应用程序之间共享数据时，强制实施了约束 / 安全性 1114。

[0257] 该配置在很多用户情形中是很重要的 ;例如，在基于数据库的文件存储构想中这是模式化数据的基础，这些数据可被 ISV 充分利用来构建智能的、知晓数据的应用程序。项目 M 可依赖于该配置来实现其成为所有用户数据通用画布的构想。这是 CDP 支持的主要配置。

[0258] 图 12 图示了第二配置，使得应用程序具有不希望被其他应用程序看到和 / 或使用的专用数据。换言之，存在涉及专用数据的双层部署。存在很多需要应用程序专用数据的用户和 ISV 情形。例如，如果应用程序决定在基于数据库的文件存储系统中存储其配置数据（例如，ini 文件等效物），那么这个过程对应用程序最好是专用的。很多时候需要局部专用性——可读但不可写。例如，在电子邮件应用程序中，最好只显示收件箱但自身保留修

改收件箱的权限。

[0259] 在双层部署中，CDP 对此配置有受限的支持。在 SQL 服务器存储中没有对应用程序级安全性的合理支持；因此，在防止数据访问的严格意义上，一个数据片段可能对于给定的应用程序不被标记为专用。然而，这种状况可通过以下方式来部分地支持：

[0260] • 应用程序使用其自己的类型，然后将其类型放入一个独立的名字空间中，并为从这些类型中得到的数据类创建专用程序集。由于对属于该模式的所有 CDP 级访问是通过这些程序集来进行的，所以其他应用程序将不具有对相应类的访问。

[0261] • 应用程序创建其自己的专用 CDP 存储（例如，可在其上创建 StorageContext 的 CDP 中的一组实体），其名字不对其他应用程序公开。

[0262] • 通过使用文档。

[0263] 应该理解，应用程序可选择一些或者全部上述方法来获得专用数据。

[0264] 应该注意，CDP 体系结构本身对于实现专用数据的真实概念不形成阻碍。因此，当应用程序级安全性在底层平台中变为可用时，CDP 可容易地展示它是可能的。同时注意，在很多情况下，“专用数据需求”的出现不是由于限制可见性的真实需求，而是由于在数据上强制实施应用程序专用业务逻辑的需求。例如，由电子邮件应用程序创建的本地收件箱有日历文件夹；规则是只有日历项可放置在这个文件夹中。只要强制实施这个规则，电子邮件应用程序就不关心另一个应用程序（诸如一个完全不同品牌的电子邮件应用程序）是否可以查看 / 修改其收件箱。只要所有应用程序都经历 CDP 层，CDP 体系结构就提供对所有业务逻辑的强制实施。应该理解，由于中间层可以强制实施这一规则，专用应用程序数据在三层部署中得到支持。

[0265] 继续图 12，示出了与 API 和 CDP 运行库交互的应用程序的机器 / 进程边界 1202，以及与 API 和 CDP 运行库交互的应用程序的机器 / 进程边界 1204。简化起见，只示出了两个应用程序，但应该理解，任意数目的应用程序在其自己的机器 / 进程边界 1208 内可访问共享数据 1210 和 / 或访问存储 1206 中各自的专用数据（例如，应用程序 1 访问专用数据 1210；应用程序 2 访问专用数据 1212）。

[0266] 图 13 示出了使得另一个应用程序可直接访问存储的所感兴趣的第三配置。换言之，存在具有直接存储访问的双层部署。在机器 / 进程边界 1320 内的应用程序 2 例如可能通过 ADO.NET 或其他数据访问 API 可直接访问 SQL 存储 1306。例如，具有现有 SQL 应用程序的大型 IT（信息技术）商店不可能消除它，并一起移动到基于 CDP 的应用程序。相反，可实现在分段基础上向 CDP 的迁移。由于零停机时间和稳定性在生产环境中是关键问题，所以很可能 CDP 应用程序可继续与 SQL 应用程序并行运行一段时间。由于 CDP 提供灵活的、非命令性 O-R（对象 - 关系）映射，因此 CDP 应用程序可在现有模式上部署。自然地，CDP 体系结构允许直接 SQL 访问。这是由于‘应用程序 1 数据’就是一组表，并且只要应用程序 2 有适当的许可，就不能阻止它的直接访问。

[0267] 注意以下有关应用程序 2 的结论：

[0268] 1) 它可能无法访问 CDP 服务（或者由 CDP 之上的框架构建的任何服务）。

[0269] 2) 特别地，它不具有 CDM 的好处——所以它必须算出表式表示，并在该级直接发布查询 / 更新。

[0270] 注意以下有关应用程序 1 的结论：

[0271] 1)BL 服务中的业务逻辑被应用程序 2 有效地绕过。

[0272] 2) 某些约束——例如那些没有被实现为触发器 /DRI(声明性引用完整性)的约束也被应用程序 2 绕过。

[0273] 在这个特定的部署中,应用程序设计员和 / 或部署管理员有责任确保应用程序 2 使其本身逻辑强制实施约束等,从而使得正确的事情发生。

[0274] 图 14 和图 15 示出了 CDP 组件的三层部署配置。各种 CDP 组件可被部署在三层配置中。在此配置中,CDP 运行库 1402 同时存在于客户机层和中间层(如图 15 所示)。应用程序位于客户机层,而存储 1404 位于服务器层(如图 15 所示)。应用程序逻辑与图 14 和 15 中的两个要求者有关:第一个是客户机 1406。第二个是 web 服务代理 1408、web 服务 1410(见图 15) 和业务逻辑主机 1412(见图 15)(例如,确认、预保存逻辑和后保存逻辑)。尽管客户机 1406 是应用程序,并且因此其中包含的逻辑可被合法地称为“应用程序逻辑”,但这不是所要参考的。而是参考包含在 web 服务代理 1408、web 服务 1410 和业务逻辑主机 1412 中的逻辑。这是由 ISV 编写的代码并且将被部署在中间层;因此,事实上这是中间层应用程序。应用程序逻辑可同时驻留在客户机层和中间层。取决于应用程序逻辑运行的地点,下面考虑了多个可能的情形。

[0275] 在开始考虑各种情形之前,应该理解多层部署的主题与跨层遥控应用程序动作的方式密切相关。术语遥控可包含以下三个通用的方法来跨层遥控应用程序级或 CDP 服务级操作。

[0276] 1) 通过 web 服务的应用程序级遥控:在这个情况中,应用程序逻辑驻留在中间层,且作为遥控的静态方法展示给客户机。这将在下文中详细讨论。

[0277] 2) 隐式 CDP 服务调用遥控:诸如 FindAll()、FindOne()、SaveChanges() 等 CDPAPI 调用通过遥控代理和遥控服务组件被隐式地发送给中间层。该体系结构在下文中描述。而且,随后的章节中有描述其如何工作的示例。

[0278] 3) 显式的、断开的遥控:CDP API 定义一种编程模式,由此应用程序显式地定义何时应该发生跨层操作。如果这个操作导致数据检索,那么检索的数据被高速缓存在客户机层。这个模式经常被称为“断开模式”(下文讨论)。

[0279] 特别地,图 14 和图 15 示出了运行在中间层(例如,web 服务)的应用程序逻辑。中间层部署的主要情形是当应用程序逻辑在中间层显式地运行的情况;客户机 1406 通过 web 服务机制(例如,web 服务代理 1408 和 web 服务 1410) 调用此逻辑。应该理解,服务器层的安全引擎可被主存在中间层 CDP 进程中。在一 2 层部署中,CDP 调用由客户机进程中的 CDP 运行库 1402 处理;运行库在必要的时候联系服务器。在 3 层部署中,某些 CDP 调用是在本地(通过客户机层)处理的,某些则远程(通过中间层)处理。此外,还有一些在两个地方都可以处理。3 层部署定义了用于遥控适当的调用的方法。

[0280] 在客户机层上的遥控代理 1414 是可以使用通道(例如,Indigo) 来打包和发送请求给中间层的 CDP 的组件(这是远程过程调用的实际动作)。在中间层存在一基本足够服务这些请求的遥控服务 1416(见图 15)。此模式是通常所称的面向服务的体系结构 (SOA) 的一部分。SOA 的一个特征是不同的层通过交换消息来相互通信。为此 CDP 可使用 Indigo 基础结构。

[0281] 遥控服务提供一组面向数据的服务一诸如“执行查询”、“插入”、“删除”、“更新”、

“创建”、“保存”、“给出关键字获取对象”、“获取根关键字”。与 SOA 范例一致，这些操作可以是 SOAP 消息内的动词。客户机层希望在中间层执行的任何动作都按照这些简单动词来表达。这些基本消息通信动词被抽象为使用 Indigo 所提供工具的 2 个接口上的方法；事实上，这是上文讨论过的 Ipersist 和 IQuery 接口。因此，遥控代理 1414 和遥控服务 1416 一起用作 Indigo 通道的断点来遥控 IPersist 和 IQuery 接口的方法。应该理解，IPersist 和 IQuery 接口中的方法在以下意义上是“粗粒度的”：它们可用于查询或操作一大组对象。例如：响应于 SaveChanges() 方法，遥控代理 1414 用整个一组改动对象向遥控服务 1416 发布一次 IPersist.Write()。因此，在客户机和中间层之间的接口是面向批量的而非繁复的。

[0282] 响应于方法调用，可描述以下伪代码示例以检查图 14 和图 15 所示的跨各种模块的数据 / 控制流。应该理解和明白，以下内容是一示例，本发明的体系结构不限于此。

[0283] 1.WinFSData wd =

[0284] new WinFSData(@" \\CorpSvr01\SharedSchedule\AnilNori"))

[0285] 2.ScheduleEntry s = wd.Items.FilterByType<ScheduleEntry>().Filter(

[0286] " StartTime>@0" ,new DateTime(xxxx,10,29,9,0,0)).GetFirst() ;

[0287] 3.s.DisplayName = s.DisplayName+" [important,please come !]" ;

[0288] 4.ScheduleService ss = new ScheduleService(wd) ;

[0289] /*public bool CreateAppointment(ScheduleEntry appointment,

[0290] * string path)*/

[0291] 5.if(ss.CreateAppointment(s,@" \\CorpSvr01\SharedSchedule\PCelis"))

[0292] 6.{

[0293] 7. Console.WriteLine(" Appointment created ! ") ;

[0294] 8.}

[0295] 在此示例中，应用程序查询在公司内联网上 Anil Nori 的共享日历以获得他的 10 月 29 日的日历条目—xxxx 于上午 9 点。这部分地是由 ScheduleEntry（日程表条目）对象来表示的，它是从实体导出的类型（例如，ScheduleEntry 是 PIM 模式的一部分，并且描述了用户日程表中的项目）。它修改 ScheduleEntry——将文本“[important,please come !]”追加到约会的标题。它然后调用 web 服务（称为 ScheduleService（日程表服务））上的 CreateAppointment（创建约会）方法，以将此修改的 ScheduleEntry 放到 Pedro Celis 的共享日历中。此代码片断示出了 3 层部署中的几个要点：

[0296] 1)1. 客户机使用本地 CDP 运行库来查询存储实体。查询在中间层执行。

[0297] 2) 查询结果在客户机层 CDP 的会话高速缓存中。

[0298] 整个“应用程序逻辑”——包括业务逻辑、确认等——在中间层由 web 服务和 CDP 的 BL 主机引擎运行。此处理过程由对 CreateAppointment() 方式的调用触发。以下内容是对各种模块之间 / 跨各种模块的数据流的详细检查。

[0299] 第 1 行：创建存储上下文。StorageContext 对象（客户机层上的 API 1418）被应用程序和 / 或客户机 1406 创建的 Data（数据）对象封装。Data 类表示在 CDM 模式中描述的表集类型。Data 对象创建如与存储 1404 交互所需地配置的 StorageContext 对象。。StorageContext 的初始化代码是客户机层上 CDP 运行库的一部分。

[0300] 第 2 行：查询。第 2 行中的表达式的 RHS 是 OPath 查询。此查询返回最多一个

ScheduleEntry 对象——在 10/29/04 上午 9 点的第一个条目（例如，假设存在“第一个条目”的精确定义）。客户机层上的 CDP 运行库 1402 获得遥控代理 1414 上的 IQuery 接口并在其上调用 ExecuteQuery(<路径>)。遥控代理 1414 可使用 Indigo 通道，并向中间层上的遥控服务 1416 发送此查询。此查询如同在双层情况下那样映射和执行，并且结果被返回给客户机层。这里有两种可能性：

[0301] 1) 原始 TDS 结果从中间层返回给客户机层，而不混合对象。客户机层上的 CDP 运行库 1402 然后混合对象。

[0302] 2) 如果这些对象已经存在于对象高速缓存 414 中，被混合的对象被返回给客户机层。

[0303] 应该理解，整个 OPath 查询穿过 Indigo 通道被发送。例如，如果查询是“寻找类型为 ScheduleEntry 的所有对象对象”（即，FindAll() 方法调用），那么此整个查询可在一个 SOAP 消息中被发送给（中间层上的遥控服务 1416），而不是每个对象一个信息。

[0304] 第 3 行：操纵客户机层对象高速缓存。一旦 ScheduleEntry 对象被返回给客户机层，它就可在客户机层上的 CDP 运行库的会话高速缓存内用于进一步的操纵。当客户机 1406 改变 ScheduleEntry 对象的 DisplayName（显示名称）属性时，这将完全由客户机层上的 CDP 运行库来处理。

[0305] 第 4 行：更新客户机层上的 web 服务代理。假定客户机 1406 已经在设计时添加了对适当 asmx（或者 Indigo 等价物）的引用。第 4 行可创建客户机上的 web 服务代理对象的实例。此调用完全由 web 服务代理 1408 来服务。

[0306] 第 5 行：调用 web 服务方法。CreateAppointment() 是由中间层上的 web 服务 1410 遥控的方法中的一个。此方法取 ScheduleEntry 对象和 CDP 连接串；它使用此信息在由连接串定义的 StorageContext 内创建 ScheduleEntry 对象。此写操作的内部是适当业务逻辑和确认逻辑的运行。此方法由 web 服务代理 1408 打包并通过 Indigo 通道经由 SOAP 消息发送给中间层上的 web 服务 1410。web 服务 1410 通过调用在中间层上的 CDP API1420 来实现该方法，如同它是任何其他应用程序一样。关键是注意 CreateAppointment() 的全部逻辑是在中间层上运行的。

[0307] 图 16 和图 17 示出了同时运行在客户机层和中间层上的应用程序逻辑的图表。响应于方法调用通过不同组件和层的数据 / 控制流可使用一个示例来更详细描述。以下的示例与上述示例相似。

[0308]
```
1.void AddToCart(String customerId,String productId)
```

[0309]
```
2.{
```

[0310]
```
3.    using(OrderData od = new OrderData()
```

[0311]
```
4.    {
```

[0312]
```
5.      ShoppingCart cart = od.ShoppingCarts.Searcher.Filter(
```

[0313]
```
6.        " CustomerId = {0}" ,customerId).GetFirst() ;
```

[0314]
```
7.      if(cart == null)
```

[0315]
```
8.        throw new Exception(“No shopping cart”) ;
```

[0316]
```
9.      Product product = od.Products.Searcher.Filter(
```

[0317]
```
10.       " ProductId = {0}" ,productId).GetFirst() ;
```

[0318] 11. if(product == null)throw new Exception("Missing product");

[0319] 12. cart.Products.Add(product);

[0320] 13. od.SaveChanges();

[0321] 14. }

[0322] 15.}

[0323] 16.

[0324] 在上面的示例中可以看到,第 3 行创建存储上下文,第 5 行和第 9 行与查询有关,第 12 行与更新有关。

[0325] 第 13 行:刷新改变。考虑以下两个可能性:

[0326] 1)BL 同时在客户机层和中间层上运行:在这种情况下,客户机层上的业务逻辑主机 416 运行确认和预保存逻辑,并且使用 IPersist.Write(<改变向量>)来调用客户机层上的遥控代理 1414。遥控代理 1414 向中间层上的遥控服务 1416(见图 17)发送调用。遥控服务 1416 改动中间层上的对象高速缓存 414 并且调用 SaveChanges()。这运行 BL 和前述的持续化步骤,并返回给遥控服务 1416,遥控服务 1416 然后返回给客户机层上的遥控代理 1414,遥控代理 1414 进而又返回给业务逻辑主机 416。客户机端后保存逻辑可以不被业务逻辑主机 416 运行。

[0327] 2)BL 只运行在中间层上。在这种情况下,业务逻辑主机 416 立即向遥控代理 1414 发送调用,遥控代理 1414 进而将其发送给遥控服务 1416。处理如上所述地在中间层发生。

[0328] 同时在两层上运行 BL 的一个优点是当预保存逻辑确认出错时,他们可在客户机层被俘获,而不需要付出连接到中间层的代价。

[0329] 无缝离线体验是基于数据库的文件存储系统的目标之一。这要求预远程存储同步数据的本地存储 1602。本地存储 1602 还可包括约束 / 安全性 1604。在这种情况下,本地存储 1602 在充分相似的机器上,但在不同进程中(在我们的定义中仍然是 2 层部署)。由于 3 层和 2 层部署的编程模块是对称的,因此对于诸如同步等服务而言,在本地存储 1602 和中间层之间操作以及保持数据同步就变得容易了。

[0330] 考虑上文示出的代码示例的第 2 行。查询导致跨层操作。在这个特定的示例中,有一个返回的对象(ScheduleEntry 对象)。然而,通常这可以潜在地返回非常大的结果集。类似的注释应用于前述代码示例的第 5 行。有两个可以考虑的且在三层部署中相关的问题:

[0331] • 穿过层可能是昂贵的,因此可能不会隐式地发生:在第 2 行中没有明确地指示这将导致一个穿层操作——换言之,涉及"魔法"。"魔法"在这里使用意味着发生了一些应用程序不知道或者没有能力控制其发生的事情。很多时候,魔法是好的;事实上,这是很多软件隐藏底层复杂性以及使事件"好像有魔法"一样发生的目标。然而,在这个特定的情况中,长期的经验已经示出应用程序编写者不管愿意不愿意都发送大量的查询,假设底层的代码以某种方式返回许多数据,而不会堵塞网络或向服务器增加压力。它是一个已被证明的设计范例,可以使任何穿层魔法对应用程序是显式的,因此奖励明智的编码实践(是"select* 所需要的 from< 上百万行的表 >"或者可能使用 WHERE 子句)。

[0332] • 客户机端高速缓存和无边界操作:虽然尝试明智的编码,还是有需要应用程序与(可能大量的)数据集合作的时候;经常地,它知道此数据集是什么。为了在这样的情况下

优化数据访问，应用程序应有能力运行查询、取出（可能大量的）数据并在高速缓存中本地存储数据。对本地的数据副本进行进一步的查询 / 排序 / 过虑 / 改变。最后刷新操作将那些改变写回存储。在本地高速缓存上工作意味着中间层维护及其最小（或没有）的状态，因此使其变得更容易缩放。

[0333] 该解决方案是提供一种显式断开模式。这是由以下模式表征的：

[0334] 1. 应用程序通过以下方式例示本地高速高速缓存：

[0335]
```
LocalContext lc = new LocalContext();
```

[0336] 2. 本地高速缓存将包含一个或多个查询的结果，指定如下：

[0337]
```
lc.QueryCollection.Add("<query1>");
```

[0338]
```
lc.QueryCollection.Add("<query2>");
```

[0339]
```
//等等
```

[0340] 3. 应用程序“填充”本地上下文

[0341]
```
lc.Fill();
```

[0342] 4. 它在本地上下文上工作，正如它对任何存储上下文一样。例如：

[0343]
```
ScheduleEntry s =
```

[0344]
```
lc.Entities.FilterByType<ScheduleEntry>().Filter(
```

[0345]
```
" StartTime>@0",new DateTime(
```

[0346]
```
2004,10,29,9,0,0)).GetFirst();
```

[0347]
```
s.DisplayName = s.DisplayName+ " [important,please come ! ]";
```

[0348] 5. 最后，它向存储发送所有改变，指定如下：

[0349]
```
//sc 是 StorageContext
```

[0350]
```
lc.SaveChanges(sc);
```

[0351] 注意，应用程序是如何对它希望穿层操作发生变为显式的一步骤 3 中的 lc.Fill()—所以没有无知的代码触发的魔法。同时注意，所有后续操作可发生在本地高速缓存中，并且因此穿层被最小化（连同在中间层上的状态的伴随维护）。应该理解，由上述代码片断隐含的模型与在 ADO.NET. 中的数据集模型是相当类似的。CDP 也可提供一种断开模型。

[0352] 2 层应用程序不应该部署在 3 层环境中，除非以下条件中的一个为真：(a) 它只使用断开编程模型，或 (b) 它被重写以使用断开的编程模型。

[0353] CDP 使用在 3 层部署中同时允许连接和断开的编程模型的方法。应用程序将被给予一个方针—如果“它们期望被部署在三层环境中，则它们应该使用断开的高速缓存。”

[0354] 为对以下讨论 CDP 存储的章节建立上下文，应该注意，所有 SQL Server 数据的整体在以下的 4 级分层结构中分区：实例、数据库、模式和表。可连接单元是一个实例；数据库是容器，在此容器上定义了备份、恢复和复制。数据库和模式的组合为查询提供了上下文。CDP 使用一个 3 级分层结构：存储、模式和类型。CDP 存储是可连接单元；模式为查询提供了上下文。给定模式可由多个 CDP 存储主存（例如，一组类型 (CRM 模式）可部署在两个不同的 CDP 实例上）。如果“同一性”是需要的，那么应该使用 CDP 的外部机制（复制、批量复制）。给定 CDP 存储可以有多个部署在其上的模式——诸如 HR 模式、会计模式等等。

[0355] 命名和发现在此讨论。考虑以下前述代码的第 3 行。

[0356] Using(StorageContext sc =

[0357] new StorageContext(@\\corp001\defaultstore))

[0358] 以下着眼于 CDP 存储的命名和可用存储器的发现。CDP 存储被更清楚地定义。有两种可能性：

[0359] 1. 它是实际的、物理存储——实际服务器上的数据库。

[0360] 2. 它是逻辑存储——对于 ctor 的形参标识了实体实例的逻辑容器。事实上，这可以被部署为复制的物理存储的场 (farm)，且前端服务器与负载平衡器合作以挑选形成此特定会话的上下文的实际物理存储。

[0361] 在 CDP 模型中，存储上下文标识逻辑存储而不是物理存储。CDP 不指定如何复制、备份 / 恢复工作于物理存储级的机制。

[0362] 关于 ctor 形参的格式，连接串是统一资源标识符，即 URL。个别框架定义替换命名格式以供其应用程序使用。例如，UAF 可能选择让其应用程序通过指定 UNC 名（例如，\\server\share）来建立存储上下文。然而，应当总是可能通过 URI 连接到 CDP 存储；换言之，框架使用的任何替换名称必须有到对应的 CDP 级 URI 的良好定义的映射。

[0363] CDP 不指定存储如何被发现。期望的是应用程序可为此可使用现有的机制和存储库（例如，UDDI）。另外，框架可指定其自己的发现方法。

[0364] 在本节中描述了应用程序能充分利用的附加 CDP 服务。这些服务包括：

[0365] ●监视 / 通知服务

[0366] ●同步服务

[0367] ●显式高速缓存服务

[0368] ●实用程序操作

[0369] 这部分应该被认为是描述性的，而非体系结构的。

[0370] 监视 / 通知服务。通知（也称为监控程序）提供引发对持久保存在底层存储中的实体（数据）的改变的异步通知的能力。应用程序（或者任何其他组件）可使用此项服务来监控持久保存的实体的改变。应用程序将拥有对其监控的内容和它们被通知的频率的完全控制。例如，多应用程序视图 (RAV) 通知是使用监控程序来构建的；客户机端浏览应用程序使用这些通知，利用 RAV 来积极地对数据改变作出反应。

[0371] CDP 编程模型支持能够监控实体中的改变的 Watcher（监控程序）类。实体监控程序机制足够使框架和应用程序构建更高级的监控程序抽象。例如，基于数据库的文件存储系统可在实体监控程序抽象上构建项、项扩展和链接监控程序抽象（例如，注意，实体是可被监控的最细粒度的数据片段）。

[0372] 同步服务。写入 CDP 的应用程序及其之上的框架都将从下面的同步相关服务中获益：

[0373] 1) 用于改变跟踪的模式注解。模式设计者可为其实体类型指定改变单元边界。改变单元规范控制改变跟踪服务的运作。

[0374] 2) 改变跟踪。很大程度上对应用程序是不可见的，它在所有 CDP 操作期间维护改变单元的版本，以及诸如实体删除等关键操作的日志。改变跟踪即使在传统应用程序持续作出绕过 CDP 运行库的变化时也能正确地运作。

[0375] 3) 改变列举。允许 CDP 应用程序检索实体集及其在特定水印前已经被修改的改变单元。改变作为 CDP 实体和行集返回。面对失败、备份和恢复以及复杂的同步拓扑结构，提供了一组服务用于水印维护。

[0376] 4) 冲突检测。允许 CDP 应用程序确定 CDP 操作（诸如更新）是否与已经执行的操作（同样基于水印）冲突。

[0377] 使用该核心功能，框架可以构建附加的、更高级的同步服务。

[0378] 显式高速缓存服务。CDP 中的显式高速缓存服务提供了应用程序的改进的性能 / 可缩放性、对断开的编程模型的支持（例如，注意，断开的编程模型可以无需全特征显式高速缓存的益处来实现）以及对瞬态数据的支持。以下可以作为显式高速缓存的特征：

[0379] ●高速缓存不同类型的数据（例如实体、非结构化的和 XML 数据）

[0380] ●不同的高速缓存访问模式（例如只读、读写、共享等）

[0381] ●与所存储数据的高速缓存一致性（例如对于存储在 SQL Server 中的数据）

[0382] ●跨多个 CDP 高速缓存对应用程序故障转移的高速缓存（某些类型的数据，例如会话上下文数据）一致性。

[0383] 用于显式高速缓存的编程表面可展示：

[0384] ●高速缓存的创建；

[0385] ●高速缓存的填充

[0386] ●将（部分数据的）高速缓存持久保存到底层存储

[0387] ●查询和更新经高速缓存的数据

[0388] 实用程序操作。CDP 提供对实体和实体集合上的各种管理和实用程序操作的支持。这种操作的示例包括：复制、移动、串行化 / 反串行化以及备份 / 恢复。

[0389] 现在回到图 18，示出了使用实体对项目进行建模。基于数据库的文件存储系统（例如 WINFS）实现包括 CDP 和用户应用程序框架 (UAF) 两者的各方面。注意，CDP 体系结构并不意味着重写基于数据库的文件存储系统，而仅是对其中的组件的再分割。

[0390] 在本节中，定义了 UAF，并接着检查基于数据库的文件存储系统的各个组件是如何被分割成 UAF 和 CDP 的。

[0391] UAF 是有关对“用户”数据建模的 CDP 框架。用户数据是指与典型最终用户有关的公用的、日常的数据，诸如文档、照片、音乐和联系人等。

[0392] UAF 向基本 CDP 基础结构添加了：

[0393] ●基本项目类型（以及相关类型）

[0394] ●用于对用户数据建模的实际类型

[0395] ●诸如生存期管理、包含等约束

[0396] ●用户可以对项进行的动作：移动、复制、重命名、串行化 ...

[0397] ●用于项的组织构造：容器、列表、自动列表、注解、类别

[0398] ●对项的最终用户编程抽象（诸如规则创作）

[0399] 应该理解和明白，对于应用程序设计人员，CDP 是 UAF 编程模型。

[0400] 特别地，图 18 描述了 UAF 中项的概念以及它实际上是如何从若干个实体导出的。文档项 1802 可以从若干个实体中导出，这些实体诸如但不限于文档 1804、多个链接 1806 以及文档扩展 1808。作者项 1810 可以从若干个实体中导出，这些实体诸如但不限于，作者

1812 和作者扩展 1814。

[0401] 转向图 19，示出了可扩展机制，用于通过在 CDP 之上实现 UAF 来实现各种功能。由于 UAF 是构建在 CDP 之上的，因此它可以使用 CDP 可扩展性机制来实现附加功能。UAF 在 CDP 上的构建可以包括各种层和 / 或组件。存储 1902 可以包括 CDP 约束引擎 1904，其中 CDP 约束引擎 1904 包括至少一个 UAF 约束 1906。CDP 运行库 1908 可以包括 BL 主机 1912，BL 主机可以包括 UAF 项行为 1910。UAF 项行为 1910 还可包括 UAF 可绑定行为 1914。在 CDP 运行库 1908 之上可以存在任何其他的 UAF 服务 1916。

[0402] UAF 使用 CDP 的约束引擎来强制实施项语义（以及其他类型语义）。这些是使用 CSDL 来创作的，且模式生成器为它们创建存储级约束。诸如移动、串行化等项行为是使用 CDP 的 BL 机制来实现的。UAF 类型可以具有与其相关联的可绑定行为。这些行为是在设计和开发了类型之后由 UAF 应用程序开发人员创作的。诸如同步、元数据处理等其他 UAF 服务被实现为常规 CDP 代码。一并考虑，这些运行在 CDP 的各个层上的单独的逻辑形成了 UAF。

[0403] 以下描述适用于在 CDP 和 UAF 之间划分基于数据库的文件存储系统。基于数据库的文件存储系统中的以下能力属于 CDP 层：

[0404] 1. 1. O-R 映射—实体到表格的映射。CDP 支持非命令性映射以便处理 POCO 情况以及基于数据库的文件存储系统服务器情况。这也包括更新映射，从而提供对实体（以及导出的）类型的基本 CUD 操作。

[0405] 2. OPath 查询映射

[0406] 3. 实体和其他 CDM 核心类型的实现

[0407] 4. StorageContext 和 StorageSearcher 以及会话和事务管理

[0408] 5. 会话高速缓存，高速缓存刷新逻辑 (SaveChanges)

[0409] 6. 改变跟踪

[0410] 7. 对实体类型的监控程序

[0411] 8. 指针服务，包括 RAV

[0412] 9. 项级安全性强制实施机制（行级安全性，安全性判定包括在类型视图中）基于数据库的文件存储系统中的以下能力属于 UAF 层：

[0413] 1. 可绑定的、每个实例的行为

[0414] 2. 基于数据库文件存储系统 API 元数据（客户机类和行为被表达为 CLR 元数据）

[0415] 3. 项级方法（复制、移动、串行化、重命名）

[0416] 4. 同步、同步范围、改变枚举

[0417] 5. 对容器的监控程序

[0418] 6. 有效路径名计算和项域的路径表

[0419] 7. 元数据处理程序

[0420] 8. 基于数据库的文件存储系统名字空间

[0421] 9. 用于强制实施项一致性的代码（容器、项部分、链接、文件流、生存期管理等）

[0422] 图 20 示出了在 CDP 上实现的 LOB 应用程序的示例。以下描述 LOB 框架要求以及它们如何能够被 CDP 支持。业务框架应用程序可以被看作 LOB 应用程序。用于业务应用程序的核心特征集是作为共享业务组件的包。这些组件的分组管理不同的业务功能，诸如财务中的分类总帐到 CRM 中的销售人员自动服务。关键的特征是这些组件是不知名的、可扩

展的，且可用于根据所使用功能和复杂级别来满足多种市场的需求。

[0423] 业务框架 (BF) 可以由业务解决方案框架和业务应用程序框架组成。业务解决方案框架提供了可用于构建大多数业务应用程序的功能。这包括基本业务数据类型，诸如货币和数量；应用程序家族范围的业务实体，诸如顾客、业务单元、多货币信息和付款术语；用于实现公用业务模式的构建块，诸如业务交易和帐户；以及公用业务处理模式，诸如用于发送业务交易。

[0424] 解决方案框架是使用业务应用程序框架来编写的，后者通过提供用于数据访问、安全性、用户界面、工作流、组件编程模型等的丰富服务来支持编写组件。如果由解决方案框架定义的业务模型和规则不适用于一个应用程序，那么它会被绕过，且应用程序的开放人员会直接使用应用程序框架。

[0425] 业务应用程序框架提供一种命令性编程模型，该模型使用 .NET 框架，且将其能力集中到业务应用程序上。在相当可扩展的同时，它为应用程序开发人员作出了很多较为通用的解决方案不会作出的决定，从而提高了使用它的生态系统中所有应用程序的实现和结构的生产力和一致性。业务应用程序框架提供了用于编写基于 web 的分布式 OLTP 应用程序的编程模型和服务。它可能不包含专用于任何产品的业务逻辑，因此不仅适合创作业务应用程序，也适合符合其基本概况的任何其他应用程序。它提供了一组服务，该组服务提供了对数据访问、消息通信（诸如使用 SOAP 和其他协议）、工作流、事件代理、实例激活、诊断、配置、元数据管理（反射）、应用程序组件安全性、全球化、业务桌面外壳等的支持。对 CDP 的要求主要来自于 BF 的业务应用程序框架部分，特别是在数据访问和数据逻辑遥控的范围内。

[0426] 实体持久性 (EP)，业务框架中的数据访问子系统支持基于实用对象关系型映射框架的丰富数据模型。它是关系型对象，因为开发人员处理被映射到关系行的 (C#) 对象。核心数据建模概念是实体和实体之间的关系。公共数据模型 (CDM) 本质上支持 BF 数据访问的数据建模需求。MBF EP 要求对以下数据访问动作的支持：

[0427] ●实体创建、读取、更新和删除

[0428] ●返回数据集的特别查询

[0429] ●在数据库中执行的基于集合的操作

[0430] BF 规定了用于支持分布式面向服务的配置的代理 / 服务框架。给定某些业务功能，代理尽可能地接近功能的用户而运行，服务尽可能地接近数据而运行。“尽可能接近”随着每个部署环境和用户类型而不同。代理 / 服务模式提供从 2 层（客户机一服务器）到多层部署的部署灵活性。在这种部署中，服务提供可以跨服务边界来调用的接口；代理一般取得接近于客户机（用户）的数据，对其进行操作并将改变传播到服务。

[0431] 特别地，图 20 示出了 LOB 框架和 / 或应用程序如何能使用 CDP。框架和使用框架构建的应用程序可以被主存在中间层的超应用程序服务器 2002 中。它可以用到客户机应用程序 2004 的 web 服务接口的形式提供标准的 LOB 服务，诸如但不限于工作流、消息通信、业务处理等。超应用程序服务器 2002 可以使用 CDP 来创作存储约束 2006（经由 CDP 约束引擎 2014）和以数据为中心的业务逻辑 2008。客户机应用程序 2004 可以在 Indigo 通道上调用 web 服务方法（例如，使用 web 服务代理 2010 和 web 服务接口 2012）。此外，它可以为其目标持久性 / 数据访问需求使用客户机层上的 CDP。

[0432] CDP 可以满足以下 :1) 会话管理 ;2)CRUD ;3) 公共数据模型 (CDM) 支持 (例如实体抽象、实体扩展);4) 查询 (例如特别、实体);5) 运行对象高速缓存 (隐式);6) 并发性管理 (例如优化、隔离级别、冲突检测等);7) 业务逻辑 (例如在方法、确认 / 默认值、属性模式、事件中);8) 安全性扩展 ;9) 与提供者 (例如,关系型、基于数据库的文件存储系统) 的映射 (查询、模式);10) 扩展元数据的能力 (支持实体的其他使用);11) 设置操作 ;12) 调用存储过程的能力 ;以及 13)N 层部署。

[0433] BF 实体持久性自然符合 CDP。CDP 完全支持大多数 BF 的持久性要求。CDP 也解决一些 SOA 要求。然而,对代理 / 服务模型的完全支持,BF 业务操作和过程可以构建在 CDP 上作为 LOB 框架。MBF 的业务解决方案框架也位于 CDP 之上的层。

[0434] 图 21 和 22 示出了依照本发明的方法。为了简化解释,方法被示出和描述为一系列动作。应该理解,本发明不限于所示的动作和 / 或动作的顺序,例如动作可以按不同的顺序和 / 或与这里没有呈现和描述的动作同时发生。此外,并非所示的所有动作都是实现依照本发明的方法所必需的。此外,本领域的技术人员会理解和明白,方法可以经由状态图或事件可选地被表示为一系列相关的状态。

[0435] 图 21 示出了便于在 CDP 的各种组件内管理数据流的方法。在参考标号 2102 处,应用程序创建订单数据对象。订单数据类可以表示在 CDM 模式中描述的表集类型。订单数据对象创建可以如与存储器交互所需地配置的存储上下文对象。在参考标号 2104 处,通过发起会话并创建事务上下文来打开到存储器的连接,在该上下文中建立了安全上下文。在参考标号 2106 处,存储上下文的实例被返回给应用程序。

[0436] 在参考标号 2108 处,展示接口以便基于 CDM 查询检索对象。在参考标号 2110 处,在适当地应用安全性的同时,查询被映射到 SQL 中。此外,应用程序 / 用户只能够看到被允许看到的数据。在参考标号 2112 处,来自查询的结果被返回给 CDP 运行库并返回给应用程序。在参考标号 2114 处,可以对经封装的存储上下文对象调用保存改变功能以便刷新改变。

[0437] 图 22 示出了便于跨多个不同的框架部署 CDP 的方法,其中完全不同的应用程序可以与每个框架相关。在参考标号 2202 处,创建能够存储结构化的、半结构化的和非结构化的数据的数据存储。在参考标号 2204 处,创建 CDP 并将其覆盖到数据存储上。继续在参考标号 2206 处,具有相关联的完全不同的应用程序的多个框架可以访问数据存储。在参考标号 2208 处,共享数据被提供给完全不同框架上的完全不同的应用程序。换言之,数据存储内的数据可以在多个完全不同的应用程序之间共享,而不管各自的框架是什么。在参考标号 2210 处,可以使用专用数据,这样使得专用数据可以对特定框架上的特定应用程序是专用的。

[0438] 现在参考图 23,示出了可用于执行 CDP 的所揭示体系结构和相关联的组件和 / 或进程的计算机的框图。为向本发明的各个方面提供额外的环境,图 23 和以下论述旨在提供可在其中实现本发明的各方面的合适的操作环境 910 的简要概括描述。虽然本发明是在诸如由一台或多台计算机或其它设备执行的程序模块等计算机可执行指令的一般上下文环境中描述的,但本领域的技术人员会认识到,本发明也可以结合其它程序模块实现和 / 或被实现为硬件和软件的组合。

[0439] 程序模块一般包括可执行特定任务或实现特定抽象数据类型的例程、程序、对象、

组件、数据结构等。此外，本领域的技术人员会理解，本发明的方法也可以在其他计算机系统配置中实现，包括单处理器或多处理器计算机系统、小型机、大型机以及个人计算机、手持式计算设备、基于微处理器的系统、可编程消费者电子设备等，它们中的每一个可以在操作上耦合到一个或多个相关联的设备。

[0440] 所示方面也可以在分布式计算环境中实现，其中可以使用经由通信网络链接的远程处理设备来执行某些任务。在分布式计算环境中，程序模块可以位于本地和远程存储器存储设备中。

[0441] 计算机一般包括各种计算机可读介质。计算机可读介质可以是计算机可访问的任何可用介质，包括易失性和非易失性介质、可移动和不可移动介质。作为示例而非限制，计算机可读介质可以包括计算机存储介质和通信介质。计算机存储介质包括以任何方法或技术实现用于存储诸如计算机可读指令、数据结构、程序模块或其他数据等信息的易失性和非易失性、可移动和不可移动介质。计算机存储介质包括但不限于，RAM、ROM、EEPROM、闪存或其他存储器技术、CD-ROM、数字多功能盘 (DVD) 或其他光盘存储、磁盒、磁带、磁盘存储或其他磁存储设备、或者任何其他可以用于存储所希望信息且可以由计算机访问的介质。

[0442] 通信介质通常具体化为诸如载波或其它传输机制等已调制数据信号中的计算机可读指令、数据结构、程序模块或其它数据，并包括任一信息传送介质。术语“已调制数据信号”指以对信号中的信息进行编码的方式设置或改变其一个或多个特征的信号。作为示例而非局限，通信介质包括有线介质，如有线网络或直接连线连接，以及无线介质，如声学、RF、红外和其它无线介质。上述任一的组合也应当包括在计算机可读介质的范围之内。

[0443] 再参考图 23，用于实现各个方面的示例性环境 2300 包括计算机 2302，计算机 2302 包括处理单元 2304、系统存储器 2306 和系统总线 2308。系统总线 2308 将包括但不限于系统存储器 2306 的系统组件耦合到处理单元 2304。处理单元 2304 可以是可以各种商业上可用的处理器中的任意一种。也可以采用双微处理器和其他多处理器体系结构作为处理单元 2304。

[0444] 系统总线 2308 可以是若干种总线结构类型的任一种，它可以进一步互连到存储器总线（有或没有存储器控制器）、外围总线以及使用多种市场上可购买的总线体系结构中任意一种的局部总线。系统存储器 2306 包括只读存储器 (ROM)2310 和随机存取存储器 (RAM)2312。基本输入 / 输出系统 (BIOS) 存储在诸如 ROM、EPROM、EEPROM 等非易失性存储器 2310 中，其中 BIOS 包括如在启动时帮助在计算机 2302 内的元件之间传输信息的基本例程。RAM2312 通常包含诸如用于高速缓存数据的静态 RAM 等高速 RAM。

[0445] 计算机 2302 还包括内部硬盘驱动器 (HDD)2314（例如 EIDE、SATA），它也可以被配置成在适当的外壳（未示出）中供外部使用；磁软盘驱动器 (FDD)2316（例如从可移动磁盘 2318 中读取或写入到其中）和光盘驱动器 2320（例如读取 CD-ROM 盘 2322 或从诸如 DVD 等其他高容量光媒体读取或写入到其中）。硬盘驱动器 2314、磁盘驱动器 2316 和光盘驱动器 2320 分别通过硬盘驱动器接口 2324、磁盘驱动器 2326 和光盘驱动器 2328 被连接到系统总线 2308。用于外部驱动器实现的接口 2324 包括通用串行总线 (USB) 和 IEEE1394 接口技术的至少之一或两者兼有。

[0446] 所述驱动器和其他相关联的计算机可读介质提供了对数据、数据结构、计算机可执行指令等的非易失性存储。对于计算机 2302，驱动器和介质以适当的数字格式容纳任何

数据的存储。虽然上述对计算机可读介质的描述是指 HDD、可移动磁盘和诸如 CD 或 DVD 等可移动光盘，但本领域的技术人员应该理解，诸如 Zip 驱动器、磁带盒、闪存卡、盒式磁带等计算机可读的其他类型的介质也可以用在示例性操作环境中，而且，任何这样的介质可以包含用于执行本发明的方法的计算机可执行指令。

[0447] 多个程序模块可以被存储在驱动器和 RAM2312 中，包括操作系统 2330、一个或多个应用程序 2332、其他程序模块 2334 和程序数据 2336。操作系统、应用程序、模块和 / 或数据的全部或部分也可以高速缓存在 RAM2314 中。应该理解，本体系结构可以用各种市场上可购买的操作系统或操作系统的组合来实现。

[0448] 用户可以通过一个或多个有线 / 无线输入设备输入命令和信息到计算机 2302 中，输入设备诸如键盘 2338 和诸如鼠标 2340 等指点设备。其它输入设备（未示出）可以包括麦克风、IR 遥控器、操纵杆、游戏垫、指示笔、触摸屏等等。这些和其它输入设备一般通过耦合到系统总线 2308 的输入设备接口 2342 连接到处理单元 2304，但也可能由其它接口连接，诸如并行端口、IEEE1394 串行端口、游戏端口、USB 端口、IR 接口等等。

[0449] 监视器 2344 或其他类型的显示设备也可以经由诸如视频适配器 2346 等接口连接到系统总线 2308。除了监视器 2344 之外，计算机通常包括其他外围输出设备（未示出），诸如扬声器、打印机等。

[0450] 计算机 2302 可使用到一个或多个诸如远程计算机 2348 这样的远程计算机的逻辑连接在网络环境内工作。远程计算机 2348 可以是工作站、服务器、路由器、个人计算机、便携式计算机、基于微处理器的娱乐设备、对等设备或其它常见的网络节点，并且一般包括上文相对于计算机 2302 所描述的许多或所有元件，然而为了简明起见，仅示出了存储器存储设备 2350。所示的逻辑连接包括到局域网 (LAN)2352 和或诸如广域网 (WAN)2354 等较大网络的有线 / 无线连接。这种 LAN 和 WAN 网络环境常见于办公室和公司，且便于诸如企业内部互联网等企业范围的计算机网络，其中它们都可连接到例如因特网等全球通信网络。

[0451] 当用于 LAN 网络环境时，计算机 2302 通过有线和 / 或无线通信网络接口或适配器 2356 连接到局域网 2352。适配器 2356 便于到 LAN2352 的有线或无线通信，LAN2352 也包括置于其上的无线接入点，用于与无线适配器 2356 的通信。

[0452] 当用在 WAN 网络环境中时，计算机 2302 可以包括调制解调器 2358 或连接到 WAN2354 上的通信服务器上，或者具有其他用于通过 WAN2354，诸如通过 Internet 建立通信的装置。调制解调器 2358，可以是内置的或外置的，且可以是有线或无线设备，它经由串行端口接口 2342 连接到系统总线 2308。在网络环境中，相对于计算机 2302 或其部分所描述的程序模块可以存储在远程存储器 / 存储设备 2350 中。应该理解，所示的网络连接是示例性的，且可以使用在计算机之间建立通信链路的其他手段。

[0453] 计算机 2302 可以用于与任何在操作上置于无线通信中的无线设备或实体通信，例如打印机、扫描仪、台式和 / 或便携式计算机、便携式数据助理、通信卫星、与无线可监测标记（例如公用电话亭、报亭、休息室）相关联的任何设备或位置、以及电话。这至少包括无线保真 (Wi-Fi) 和 Bluetooth™（蓝牙）无线技术。因此，通信可以是诸如传统网络等的预定义结构或简单地是至少两台设备之间的特别通信。

[0454] Wi-Fi 或无线保真允许从家里的沙发、宾馆房间的床上、工作地点的会议室到因特网的无线连接。Wi-Fi 是一种类似于用在蜂窝电话中的无线技术，它允许诸如计算机

等设备在基站范围内的任何地方在户内或户外发送和接收数据。Wi-Fi 网络使用称为 IEEE802.11(a、b、g 等等）的无线技术来提供安全的、可靠的、快速的无线连接。Wi-Fi 网络可以用于将计算机互相连接、将计算机连接到因特网和到有线网络（使用 IEEE802.3 或以太网）。Wi-Fi 网络工作在未经许可的 2.4 和 5GHz 无线波段，以 11Mbps(802.11a) 或 54Mbps(802.11b) 的数据速率工作，或使用同时包含两个波段（双波段）的产品，使得网络可以提供类似于在许多办公室中使用的基本 10BaseT 有线以太网网络的真实性能。

[0455] 现在参见图 24，示出了可由 CDP 使用的示例性计算环境 2400 以及各自的组件和 / 或用于提供数据管理的进程的示意性框图。系统 2400 包括一个或多个客户机 2402。客户机 2401 可以是硬件和 / 或软件（例如线程、进程、计算设备）。客户机 2402 可以例如通过采用本发明的体系结构容纳 cookie 和 / 或相关联的上下文信息。

[0456] 系统 2400 也包括一个或多个服务器 2404。服务器 2404 也可以是硬件和 / 或软件（例如线程、进程、计算设备）。服务器 2404 可以例如通过采用本发明的体系结构容纳线程来执行转换。客户机 1202 和服务器 2404 之间的一个可能的通信可以是适用于在两个或多个计算机进程之间发送的数据分组的形式。数据分组可以包括例如 cookie 和 / 或相关联的上下文信息。系统 2400 包括通信框架 2406（诸如因特网等全球通信网络），它可用于便于客户机 2402 和服务器 2404 之间的通信。

[0457] 可以经由有线（包括光纤）和 / 或无线技术来促进通信。客户机 2402 在操作上可以连接到一个或多个客户机数据存储 2408，该客户机数据存储 1208 可以用于存储对客户机 2402 而言本地的信息（例如 cookie 和 / 或相关联的上下文信息）。类似地，服务器 2404 在操作上可以连接到一个或多个服务器数据存储 2410，该服务器数据存储 2410 可用于存储对服务器 2404 而言本地的信息。

[0458] 以上描述的包括示例。当然，不可能为了描述本发明而描述组件或方法的每种想象得到的组合，但是本领域的技术人员会认识到，可能有许多其他的组合和变换。因此，本发明旨在包含落入所附权利要求书的精神和范围内的所有这样的改变、修改和变化。此外，在详细描述或权利要求中使用术语“包括”的意义上，这样的术语旨在以类似于术语“包含”的方式是包含性的，如同术语“包含”在用作权利要求中的过渡词时的解释。

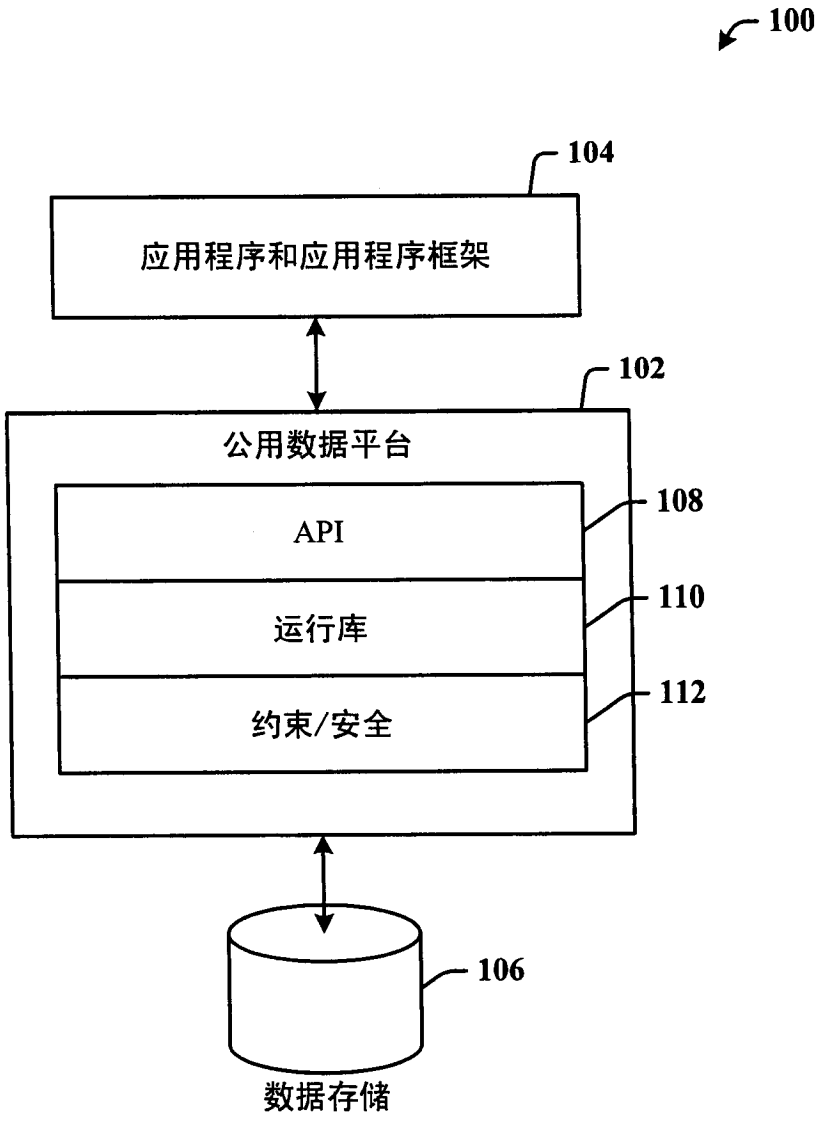

100
104
应用程序和应用程序框架
102
公用数据平台
API
108
运行库
110
约束/安全
112
106
数据存储

图 1

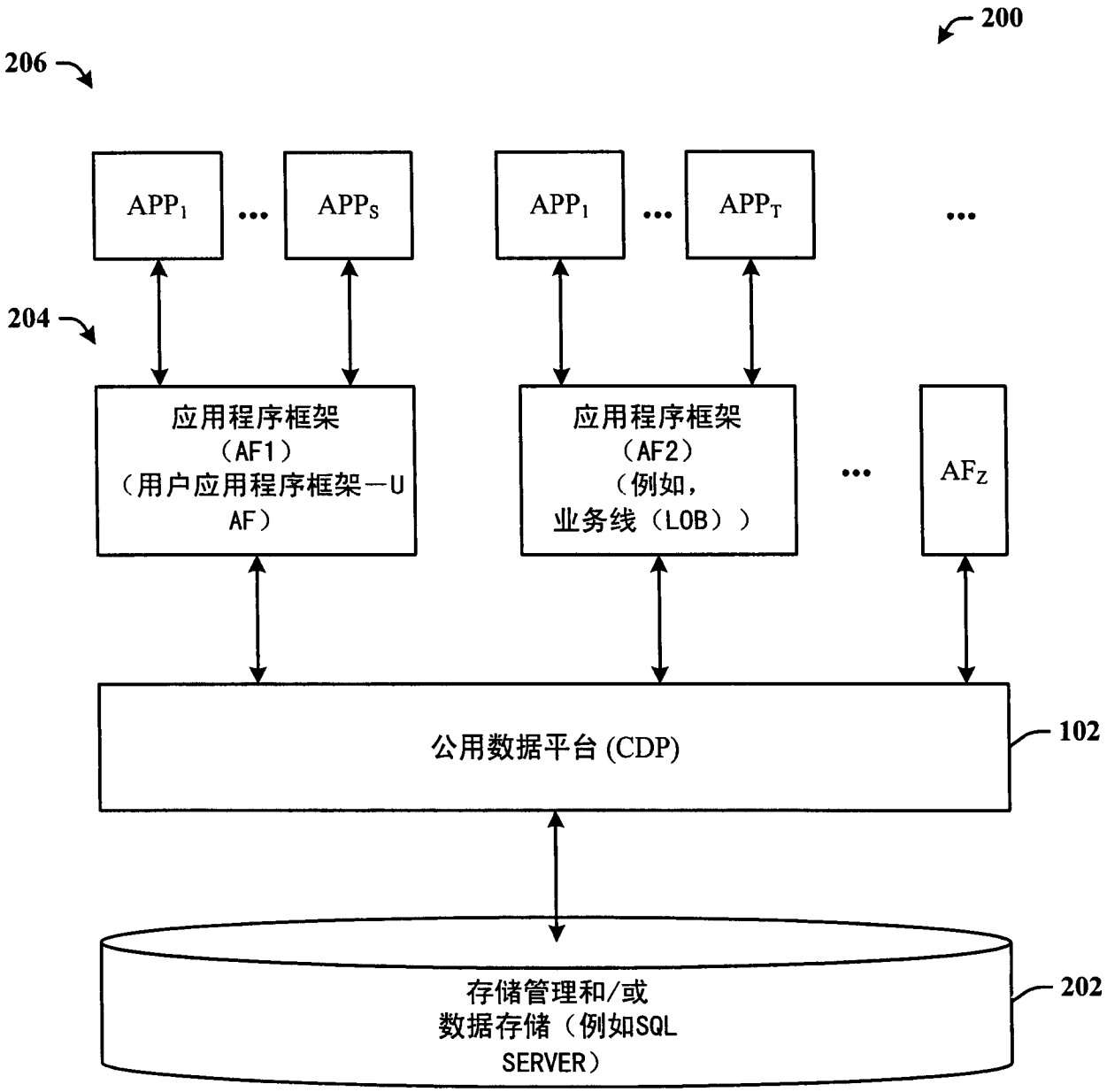

200
206
APP1
APPS
APP1
APPT
204
应用程序框架
（AF1）
（用户应用程序框架—UAF）
应用程序框架
（AF2）
（例如，
业务线（LOB））
AFZ
公用数据平台 (CDP)
102
存储管理和/或
数据存储（例如SQL SERVER）
202

图 2

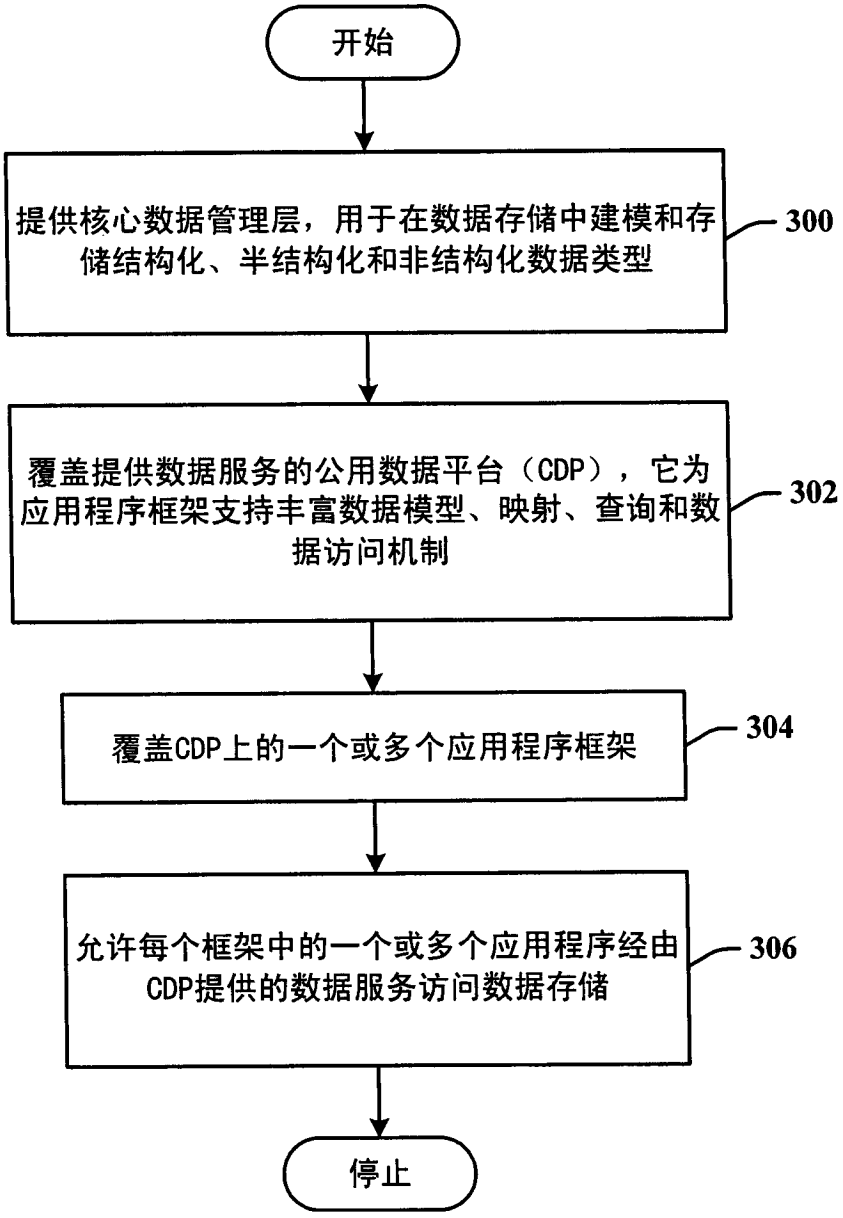

开始
提供核心数据管理层，用于在数据存储中建模和存储结构化、半结构化和非结构化数据类型
300
覆盖提供数据服务的公用数据平台（CDP），它为应用程序框架支持丰富数据模型、映射、查询和数据访问机制
302
覆盖CDP上的一个或多个应用程序框架
304
允许每个框架中的一个或多个应用程序经由CDP提供的数据服务访问数据存储
306
停止

图 3

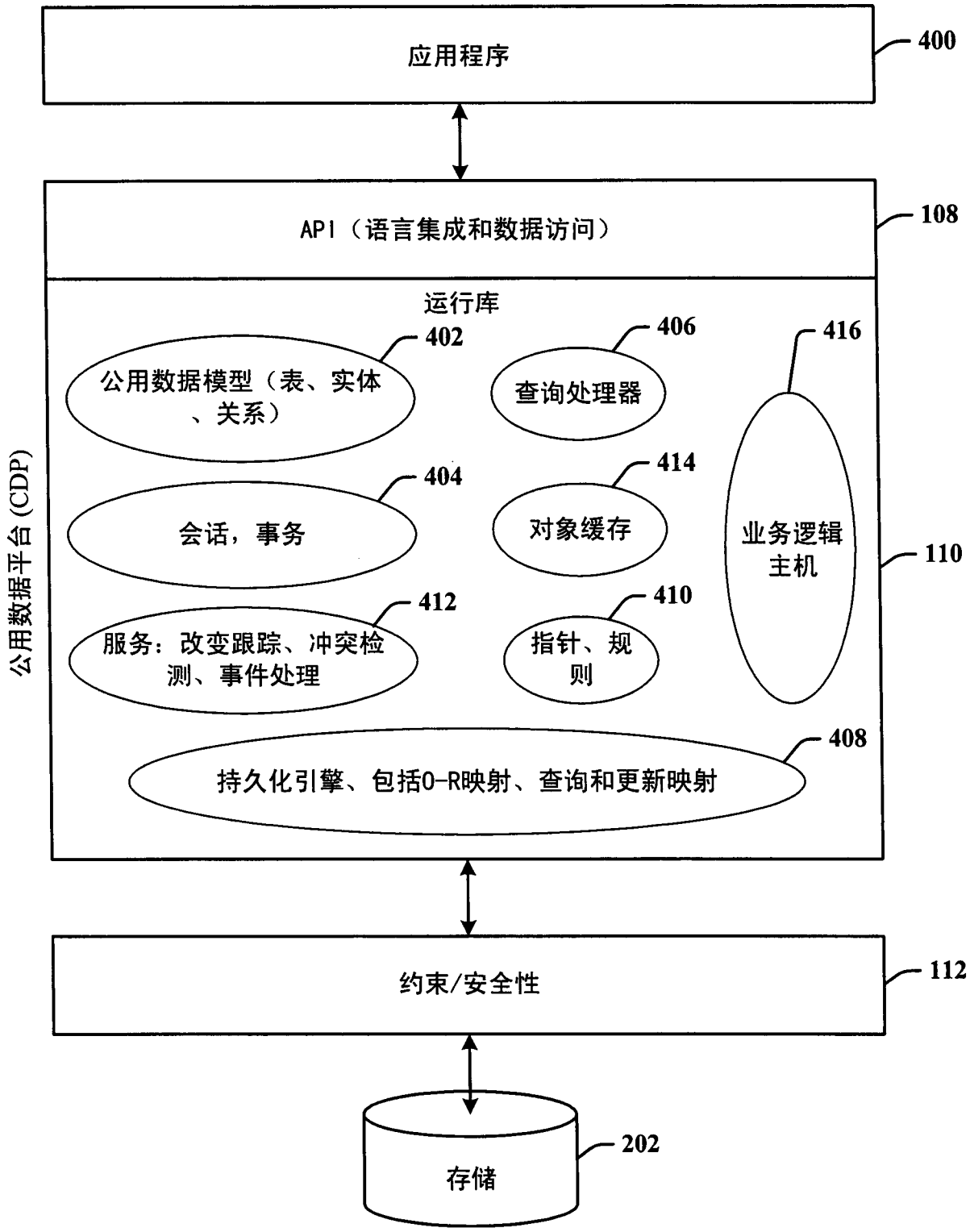

应用程序
400
API（语言集成和数据访问）
108
运行库
公用数据平台 (CDP)
402
公用数据模型（表、实体、关系）
406
查询处理器
416
404
会话，事务
414
对象缓存
业务逻辑主机
110
412
服务：改变跟踪、冲突检测、事件处理
410
指针、规则
408
持久化引擎、包括O-R映射、查询和更新映射
约束/安全性
112
存储
202

图 4

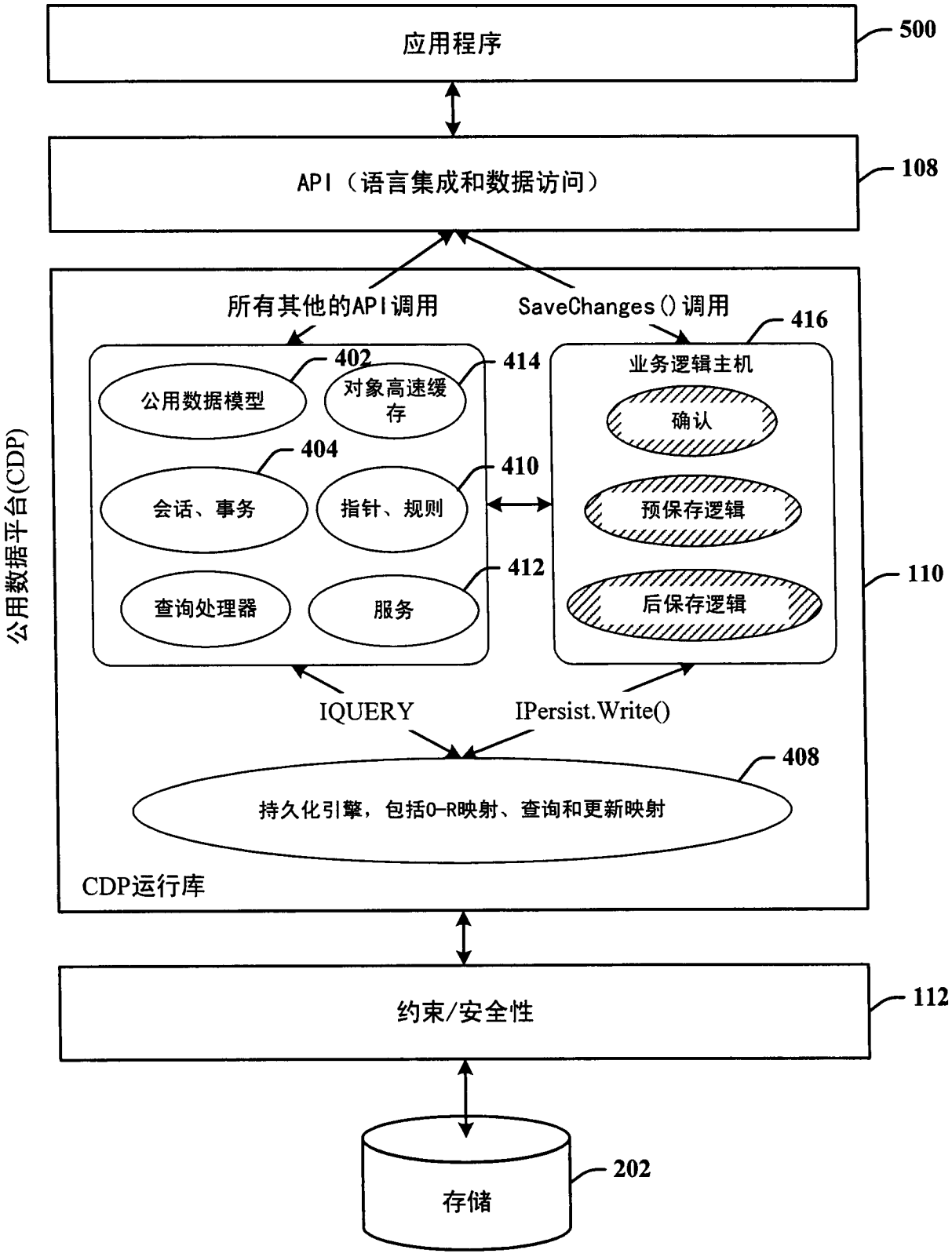

应用程序
500
API（语言集成和数据访问）
108
所有其他的API调用
SaveChanges()调用
416
402
414
公用数据模型
对象高速缓存
业务逻辑主机
确认
404
410
会话、事务
指针、规则
预保存逻辑
412
查询处理器
服务
后保存逻辑
110
公用数据平台(CDP)
IQUERY
IPersist.Write()
408
持久化引擎，包括O-R映射、查询和更新映射
CDP运行库
约束/安全性
112
存储
202

图 5

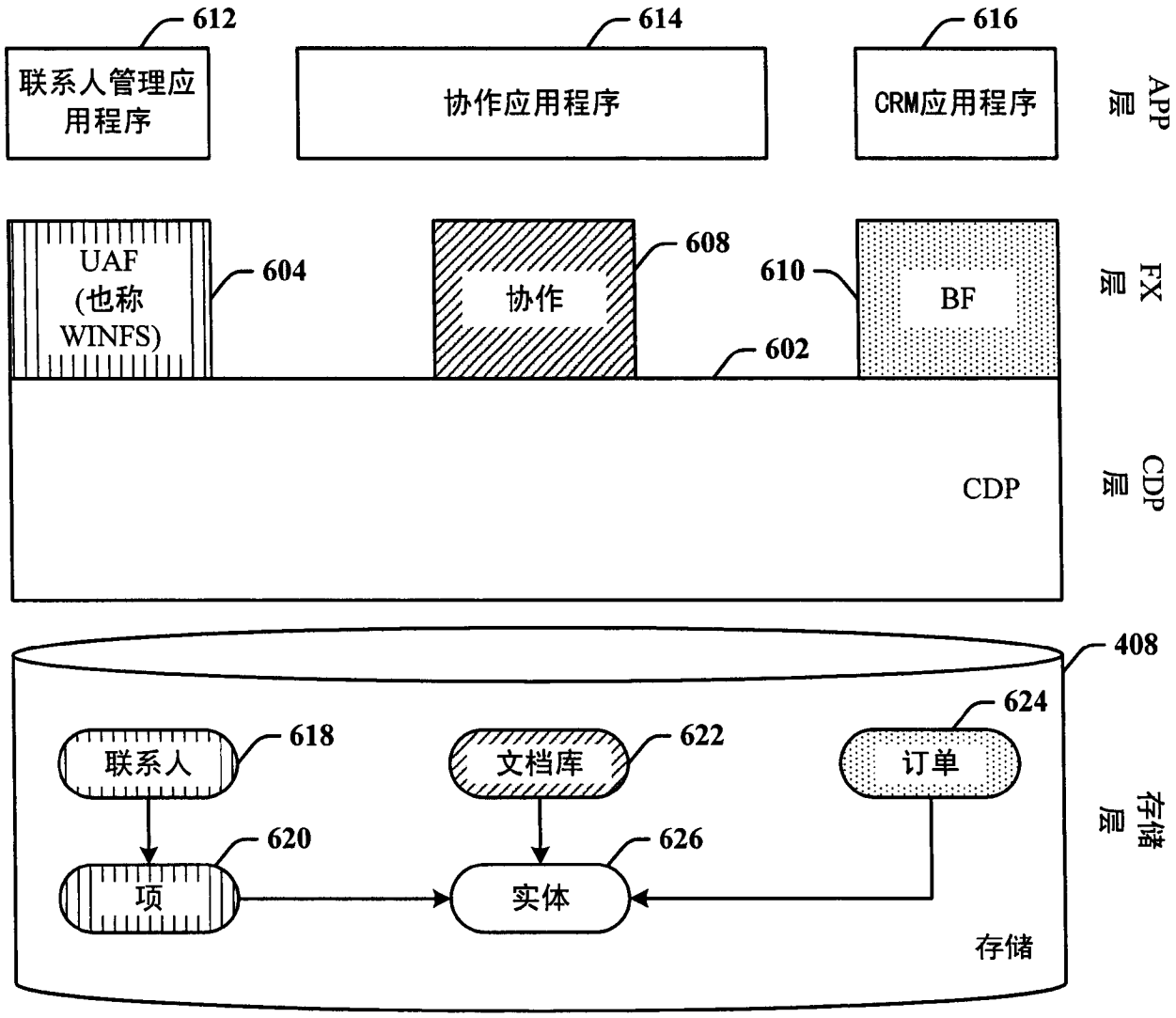

612
614
616
联系人管理应用程序
协作应用程序
CRM应用程序
APP层
UAF (也称 WINFS)
604
608
协作
610
BF
602
FX层
CDP
CDP层
408
624
618
622
联系人
文档库
订单
620
626
项
实体
存储
存储层

图 6

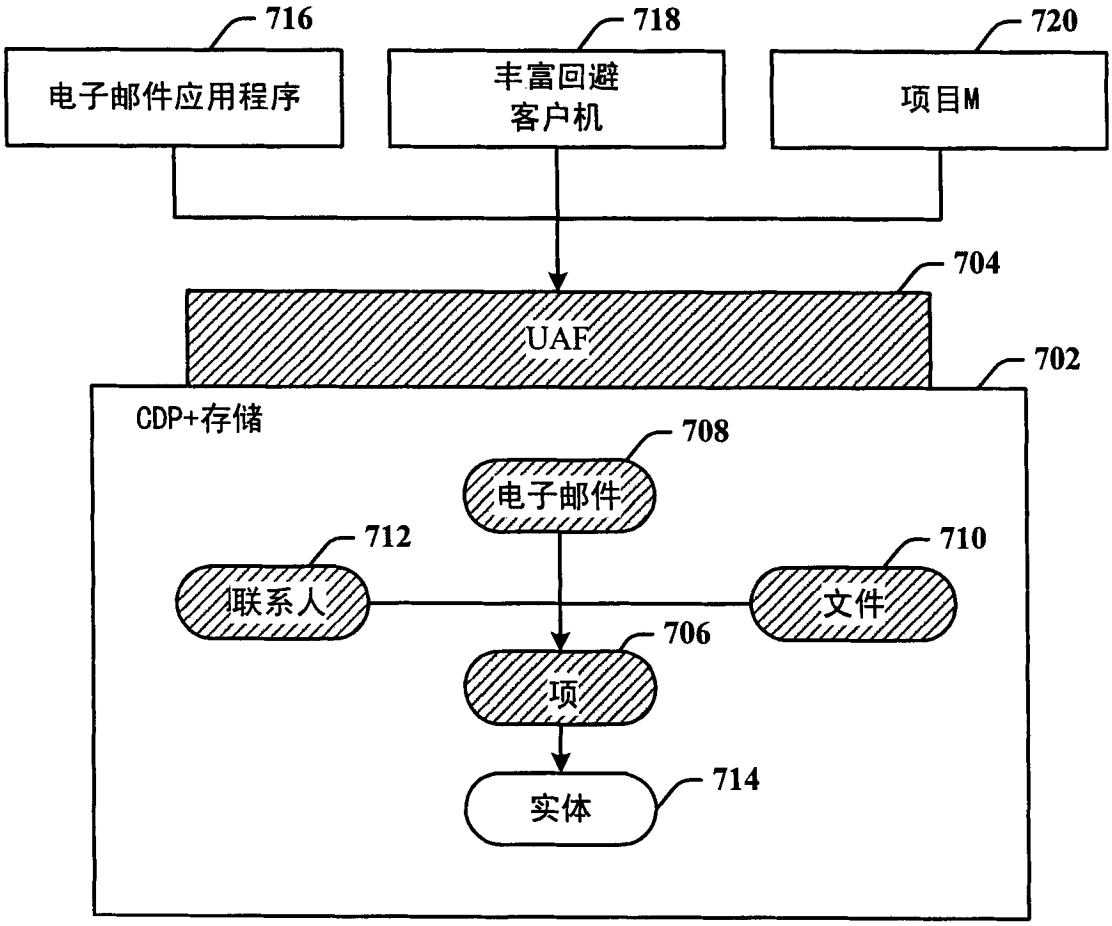

716
718
720
电子邮件应用程序
丰富回避
客户机
项目M
704
UAF
702
CDP+存储
708
电子邮件
712
710
联系人
文件
706
项
714
实体

图 7

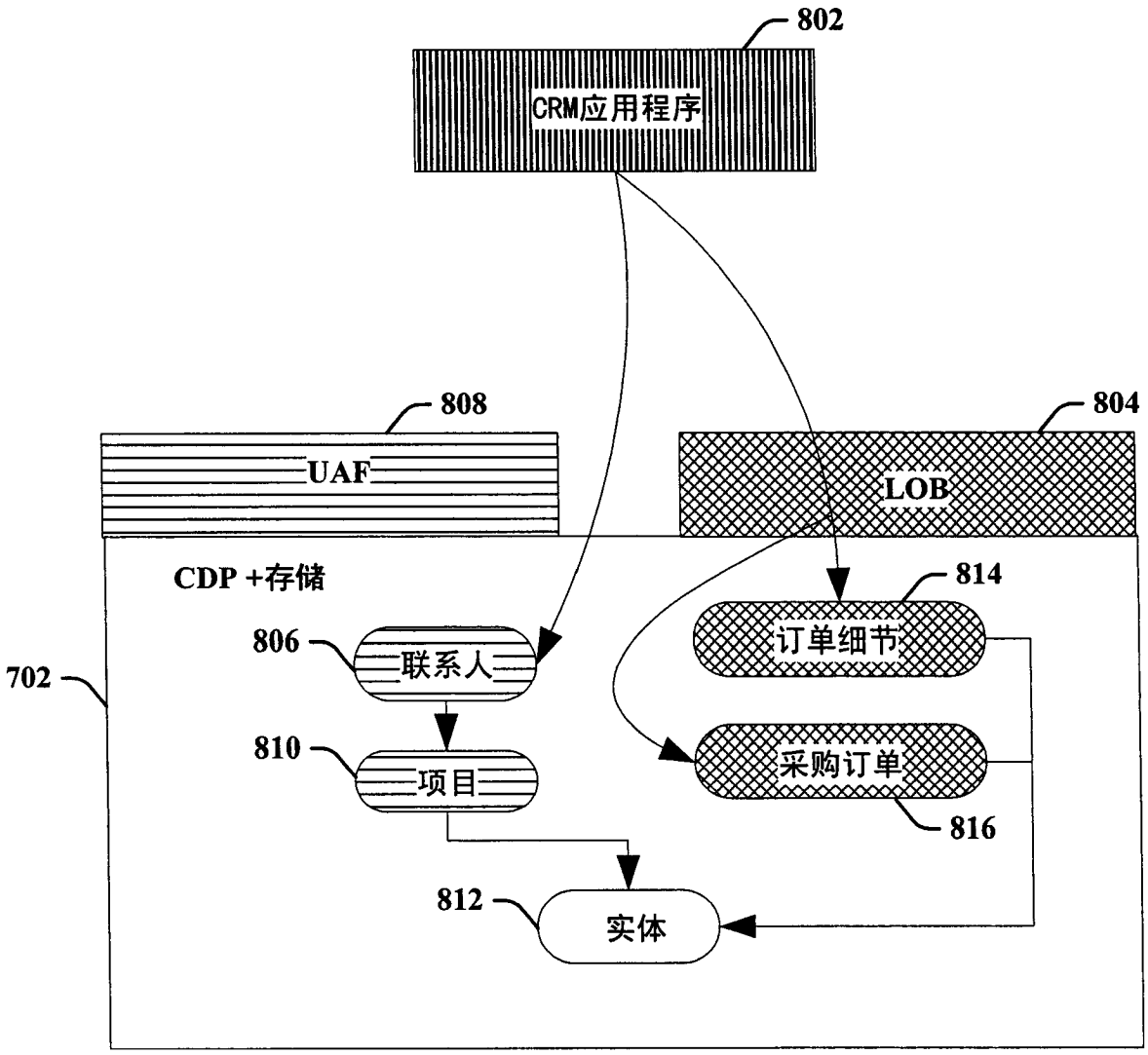

802
CRM应用程序
808
UAF
804
LOB
CDP +存储
814
806
联系人
订单细节
702
810
项目
采购订单
816
812
实体

图 8

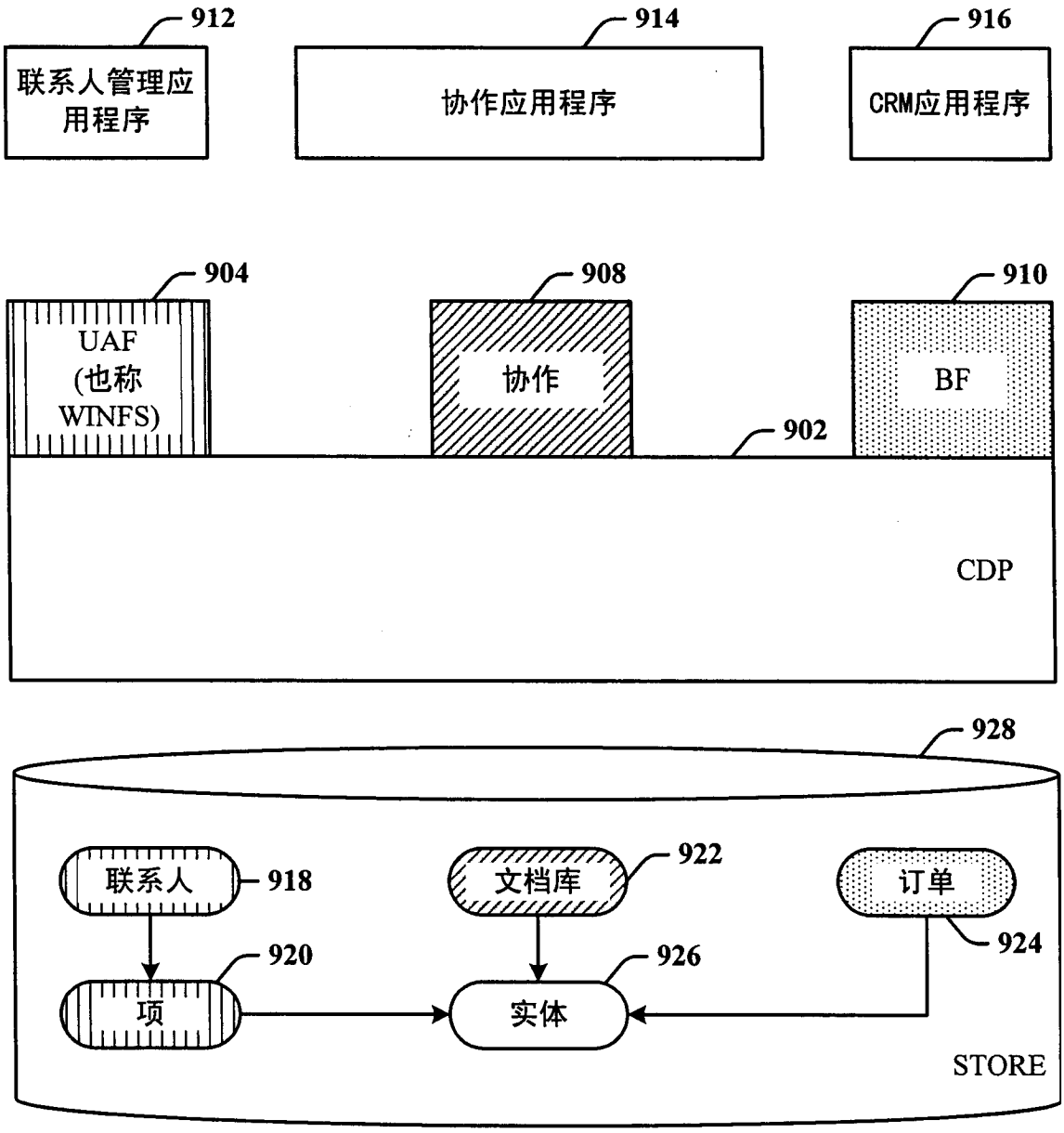

912
联系人管理应用程序
914
协作应用程序
916
CRM应用程序
904
UAF (也称 WINFS)
908
协作
910
BF
902
CDP
928
联系人
918
文档库
922
订单
924
920
项
926
实体
STORE

图 9

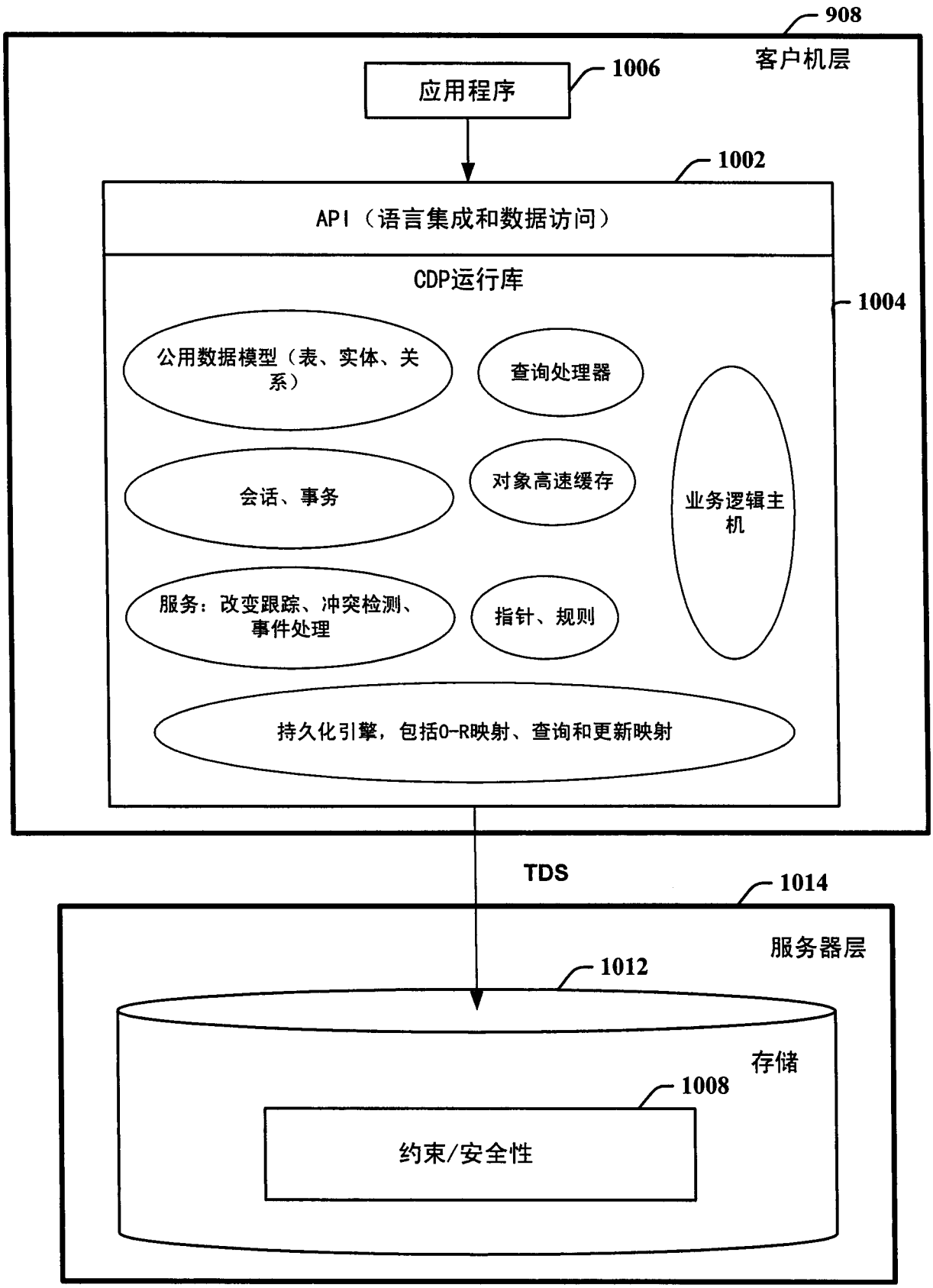

908
客户机层
应用程序
1006
1002
API（语言集成和数据访问）
CDP运行库
1004
公用数据模型（表、实体、关系）
查询处理器
会话、事务
对象高速缓存
业务逻辑主机
服务：改变跟踪、冲突检测、事件处理
指针、规则
持久化引擎，包括O-R映射、查询和更新映射
TDS
1014
服务器层
1012
存储
1008
约束/安全性

图 10

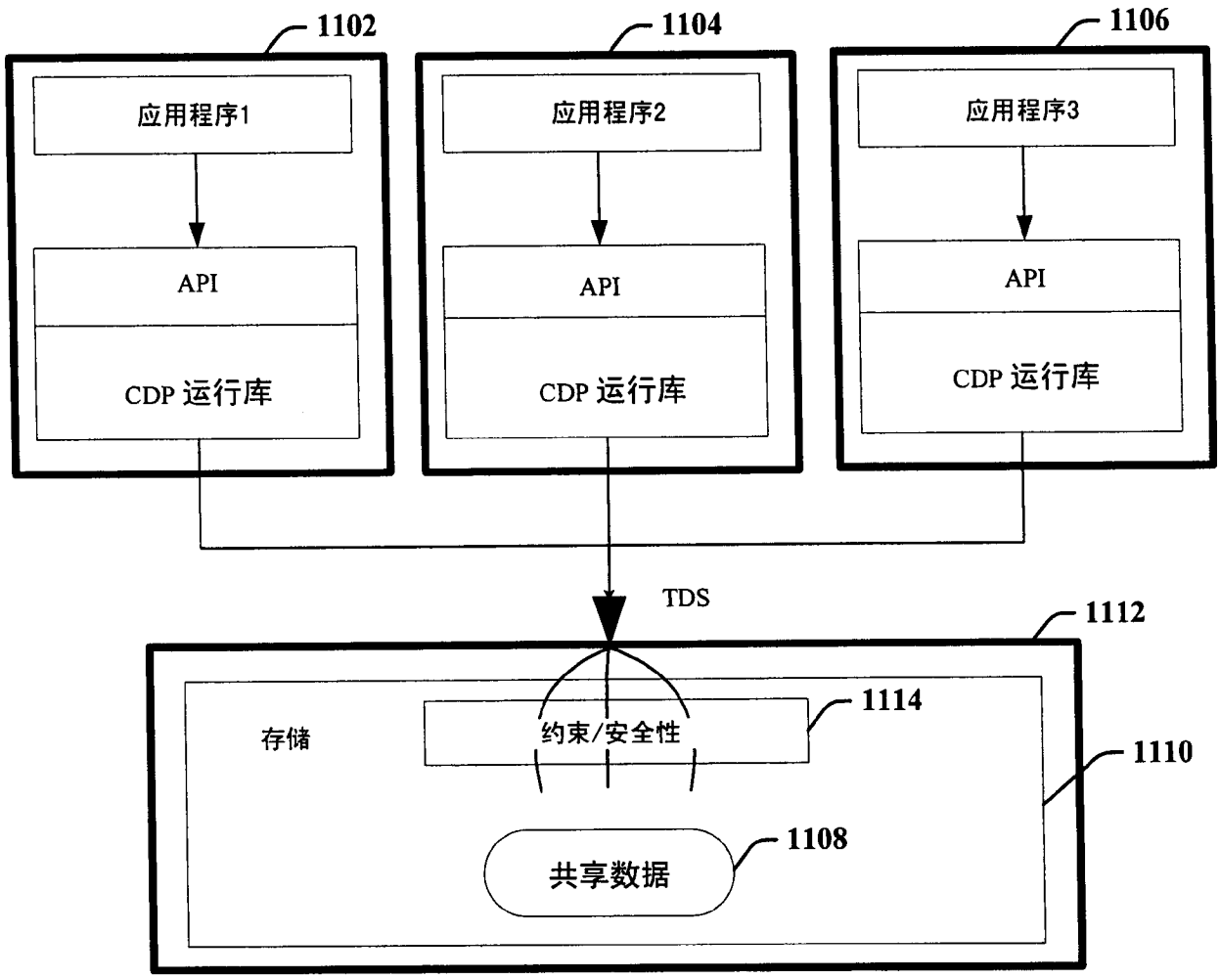

1102
1104
1106
应用程序1
应用程序2
应用程序3
API
API
API
CDP 运行库
CDP 运行库
CDP 运行库
TDS
1112
存储
约束/安全性
1114
1110
共享数据
1108

图 11

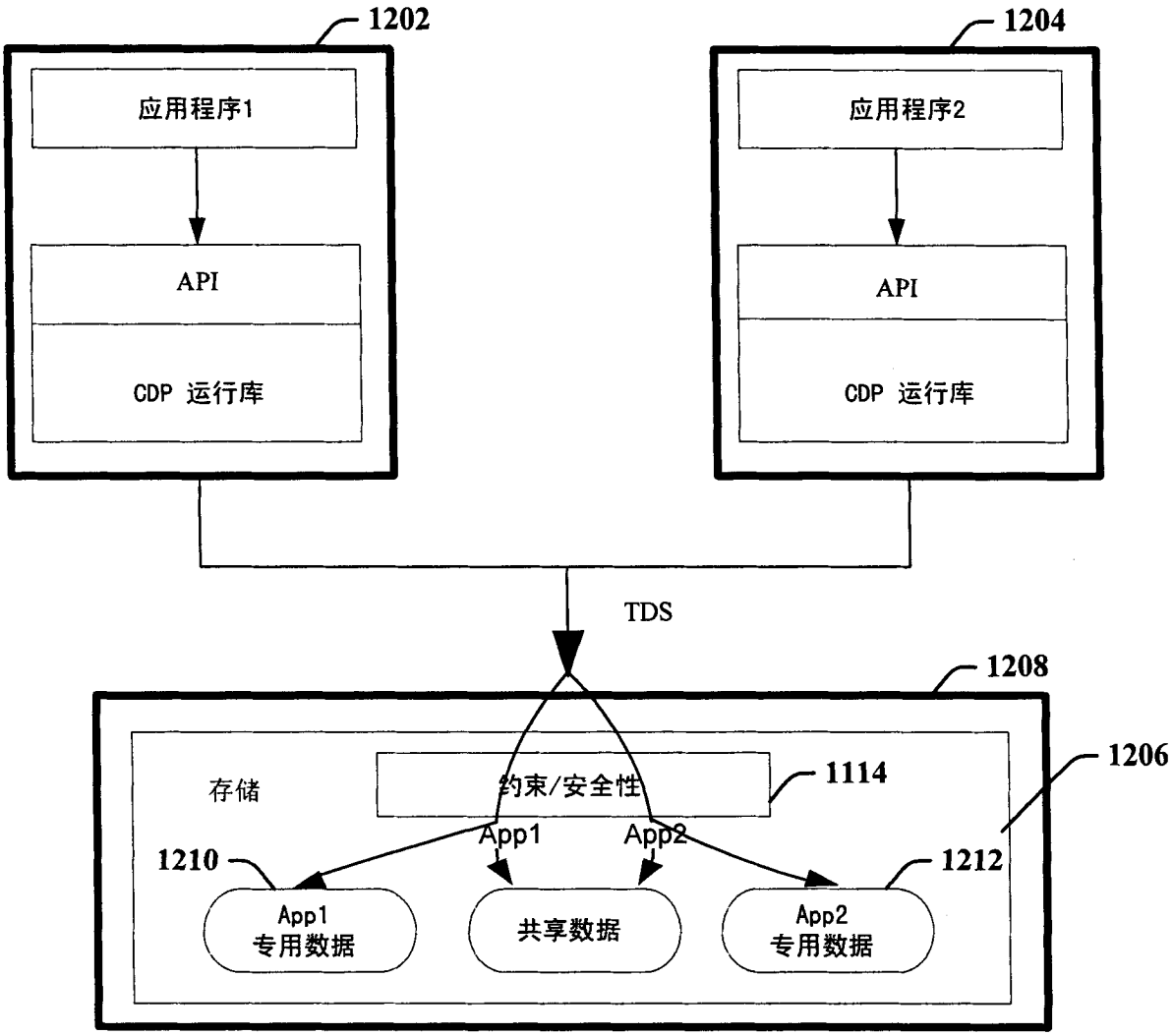

1202
应用程序1
API
CDP 运行库
1204
应用程序2
API
CDP 运行库
TDS
1208
1206
存储
约束/安全性
1114
App1
App2
1210
1212
App1
专用数据
共享数据
App2
专用数据

图 12

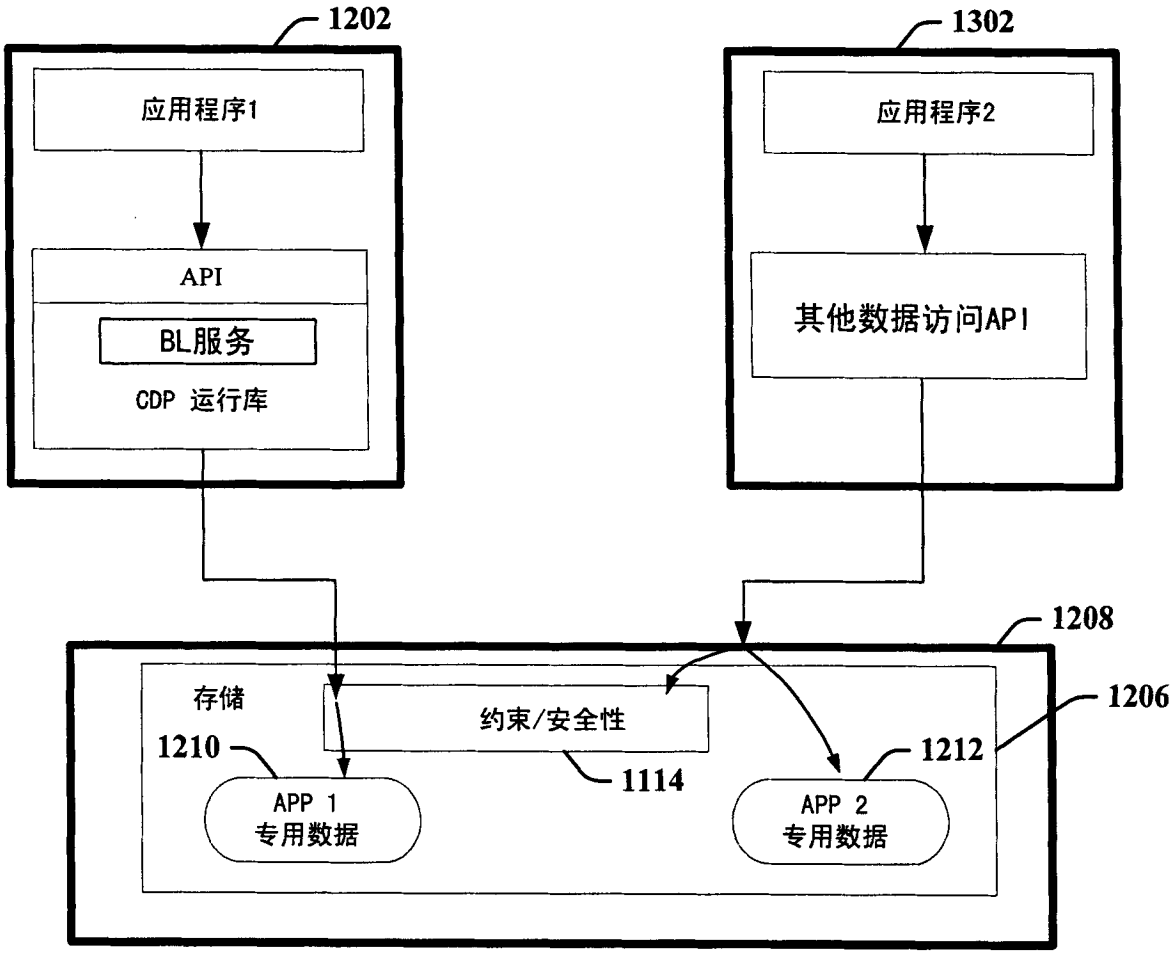

1202
应用程序1
API
BL服务
CDP 运行库
1302
应用程序2
其他数据访问API
1208
1206
存储
1210
约束/安全性
1114
1212
APP 1
专用数据
APP 2
专用数据

图 13

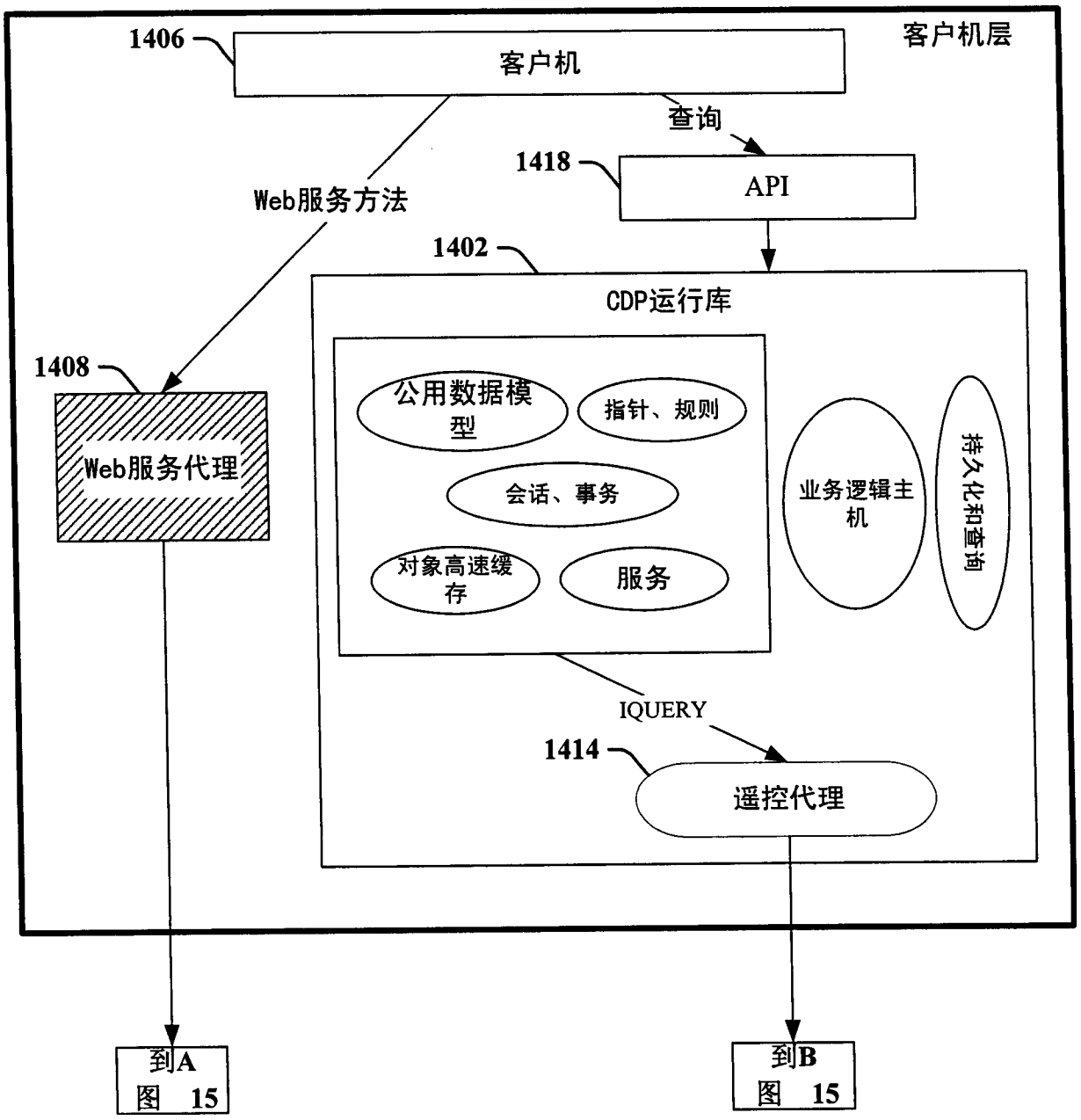

1406
客户机
客户机层
查询
1418
API
Web服务方法
1402
CDP运行库
1408
公用数据模型
指针、规则
Web服务代理
会话、事务
业务逻辑主机
持久化和查询
对象高速缓存
服务
IQUERY
1414
遥控代理
到A
图 15
到B
图 15

图 14

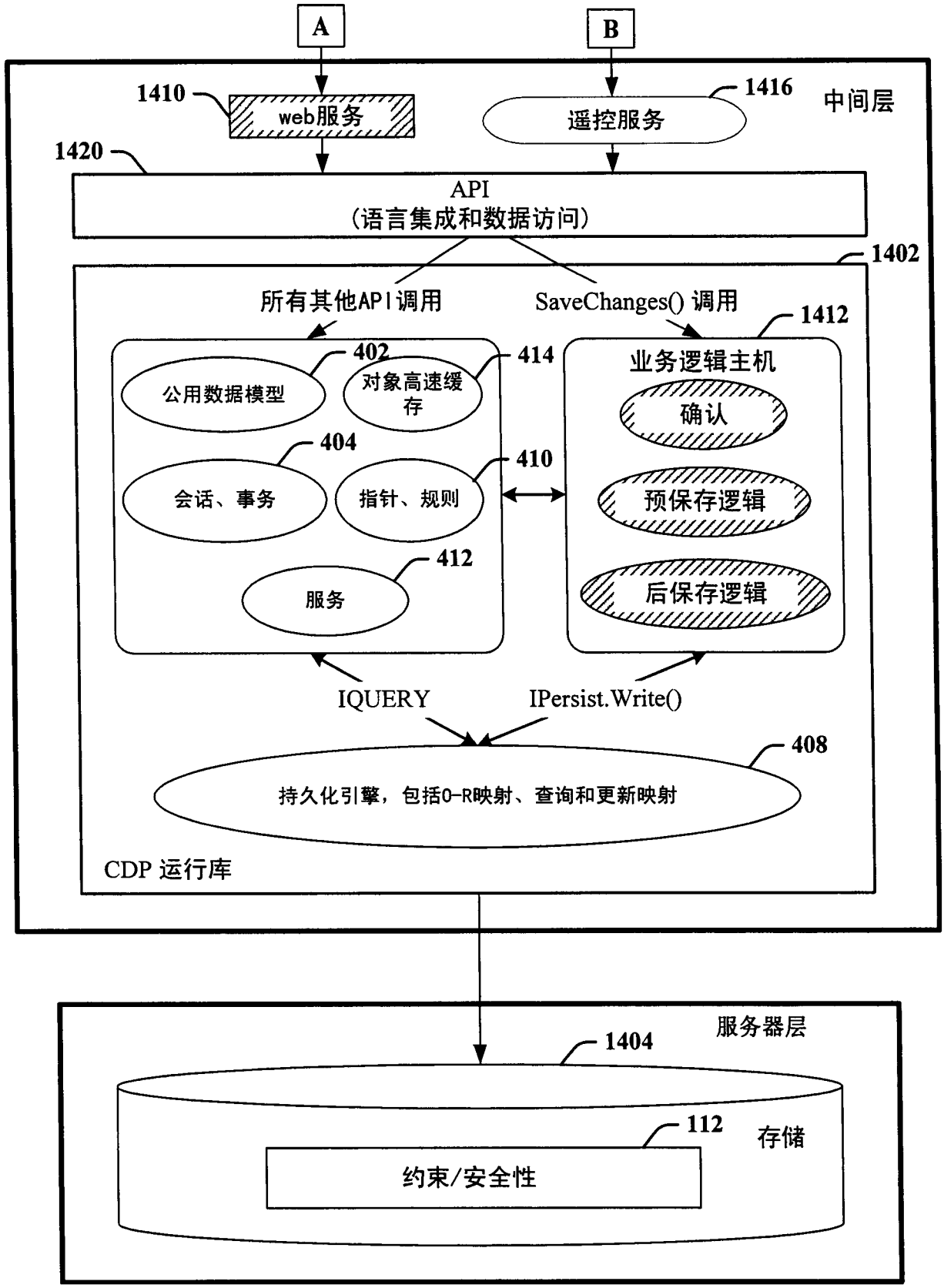

A
B
1410
web服务
1416
遥控服务
中间层
1420
API
(语言集成和数据访问)
1402
所有其他API调用
SaveChanges() 调用
1412
402
公用数据模型
414
对象高速缓存
业务逻辑主机
确认
404
会话、事务
410
指针、规则
预保存逻辑
412
服务
后保存逻辑
IQUERY
IPersist.Write()
408
持久化引擎，包括O-R映射、查询和更新映射
CDP 运行库
服务器层
1404
112
存储
约束/安全性

图 15

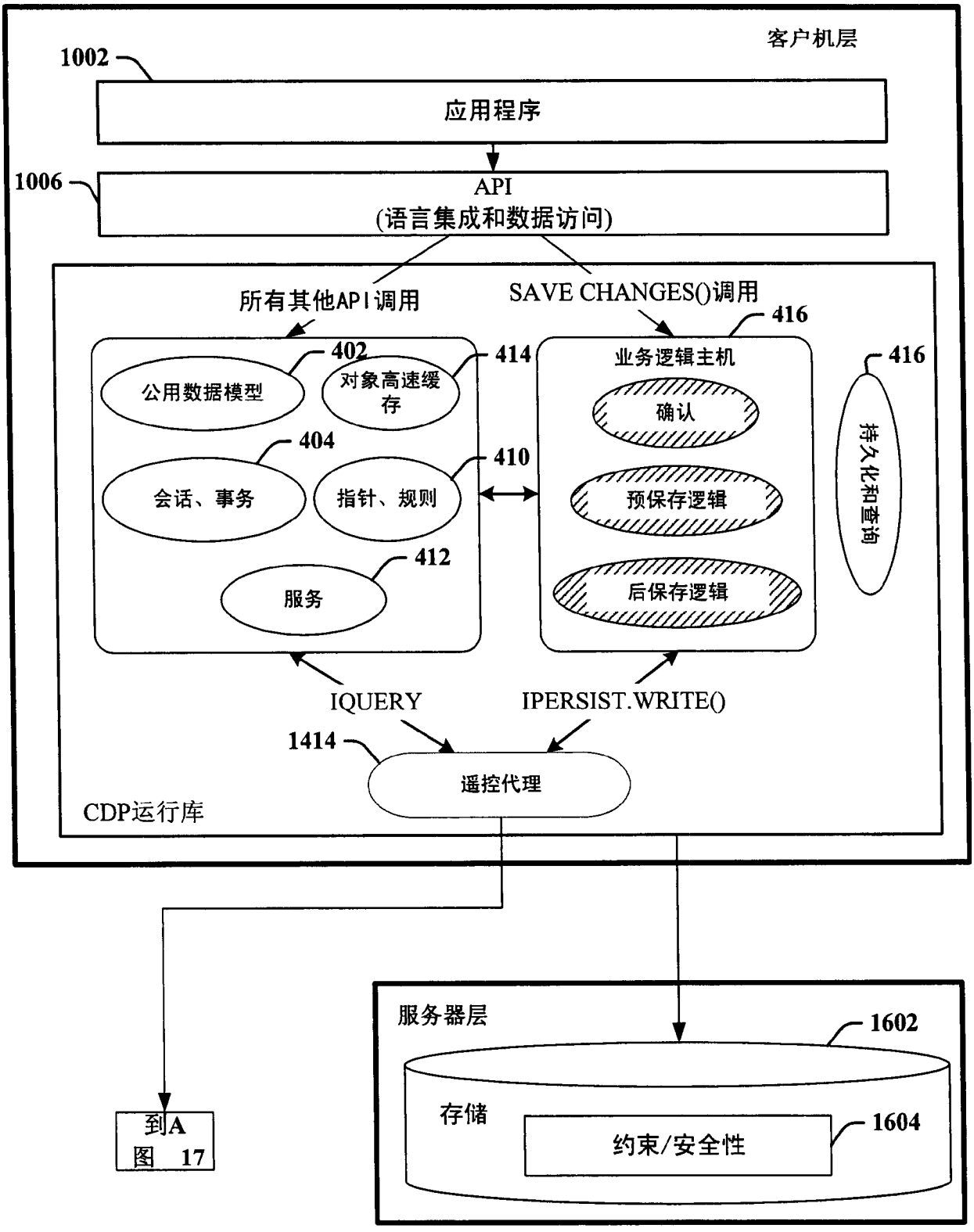

客户机层
1002
应用程序
1006
API
(语言集成和数据访问)
所有其他API调用
SAVE CHANGES()调用
416
402
414
公用数据模型
对象高速缓存
业务逻辑主机
416
确认
404
410
会话、事务
指针、规则
预保存逻辑
持久化和查询
412
服务
后保存逻辑
IQUERY
IPERSIST.WRITE()
1414
遥控代理
CDP运行库
服务器层
1602
存储
1604
约束/安全性
到A
图 17

图 16

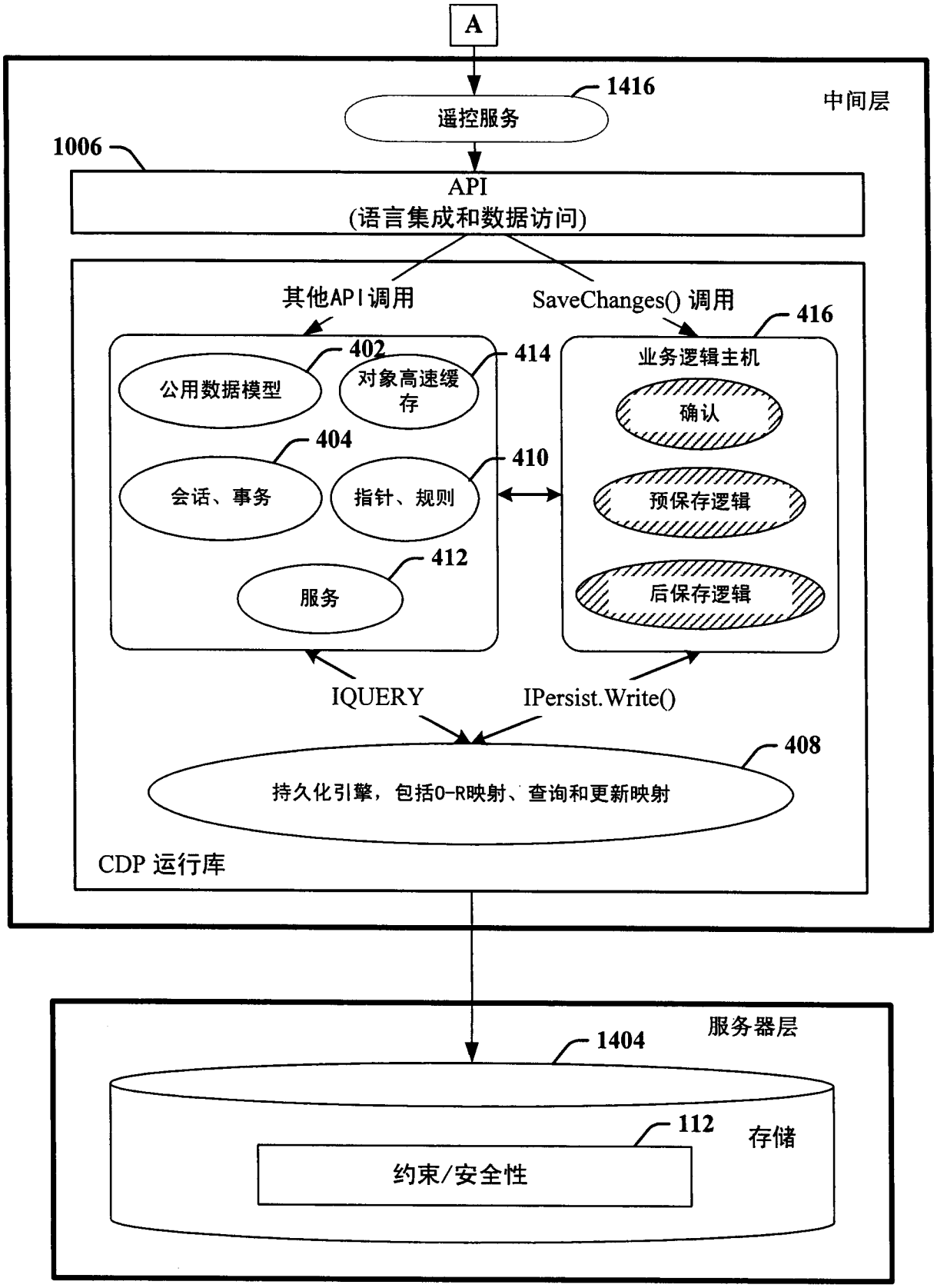

A
1416
遥控服务
中间层
1006
API
(语言集成和数据访问)
其他API调用
SaveChanges() 调用
416
402
414
公用数据模型
对象高速缓存
业务逻辑主机
确认
404
410
会话、事务
指针、规则
预保存逻辑
412
服务
后保存逻辑
IQUERY
IPersist.Write()
408
持久化引擎，包括O-R映射、查询和更新映射
CDP 运行库
服务器层
1404
112
存储
约束/安全性

图 17

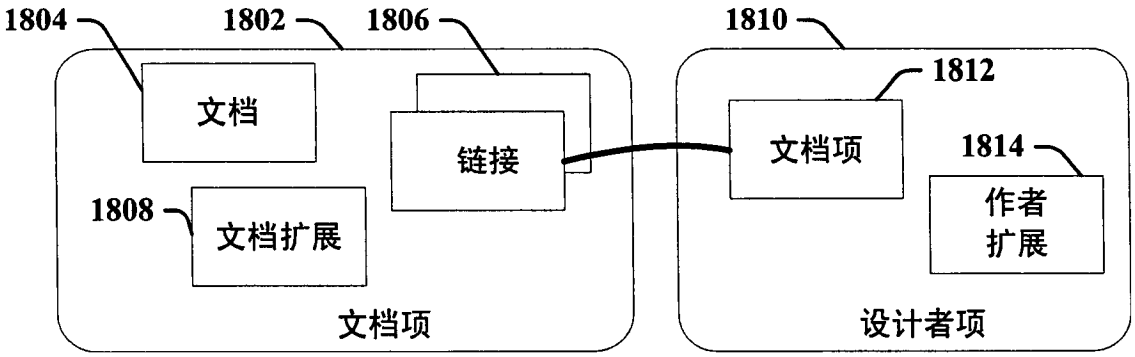

1804
1802
1806
1810
1812
文档
链接
文档项
1814
作者
扩展
1808
文档扩展
文档项
设计者项

图 18

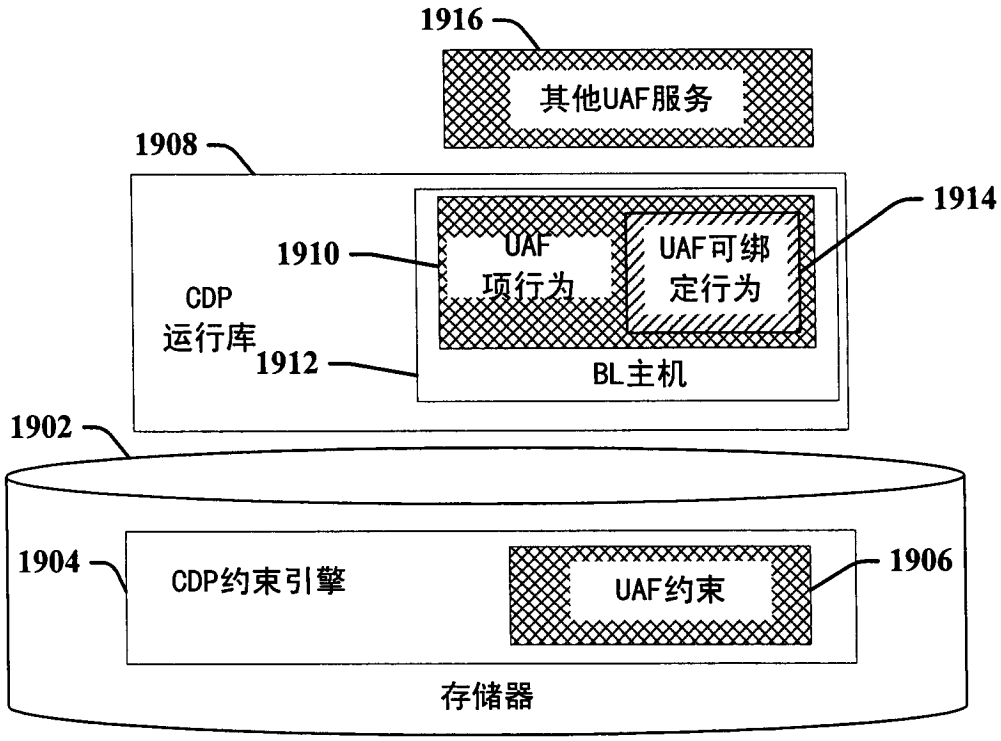

1916
其他UAF服务
1908
1914
1910
UAF
项行为
UAF可绑
定行为
CDP
运行库
1912
BL主机
1902
1904
CDP约束引擎
UAF约束
1906
存储器

图 19

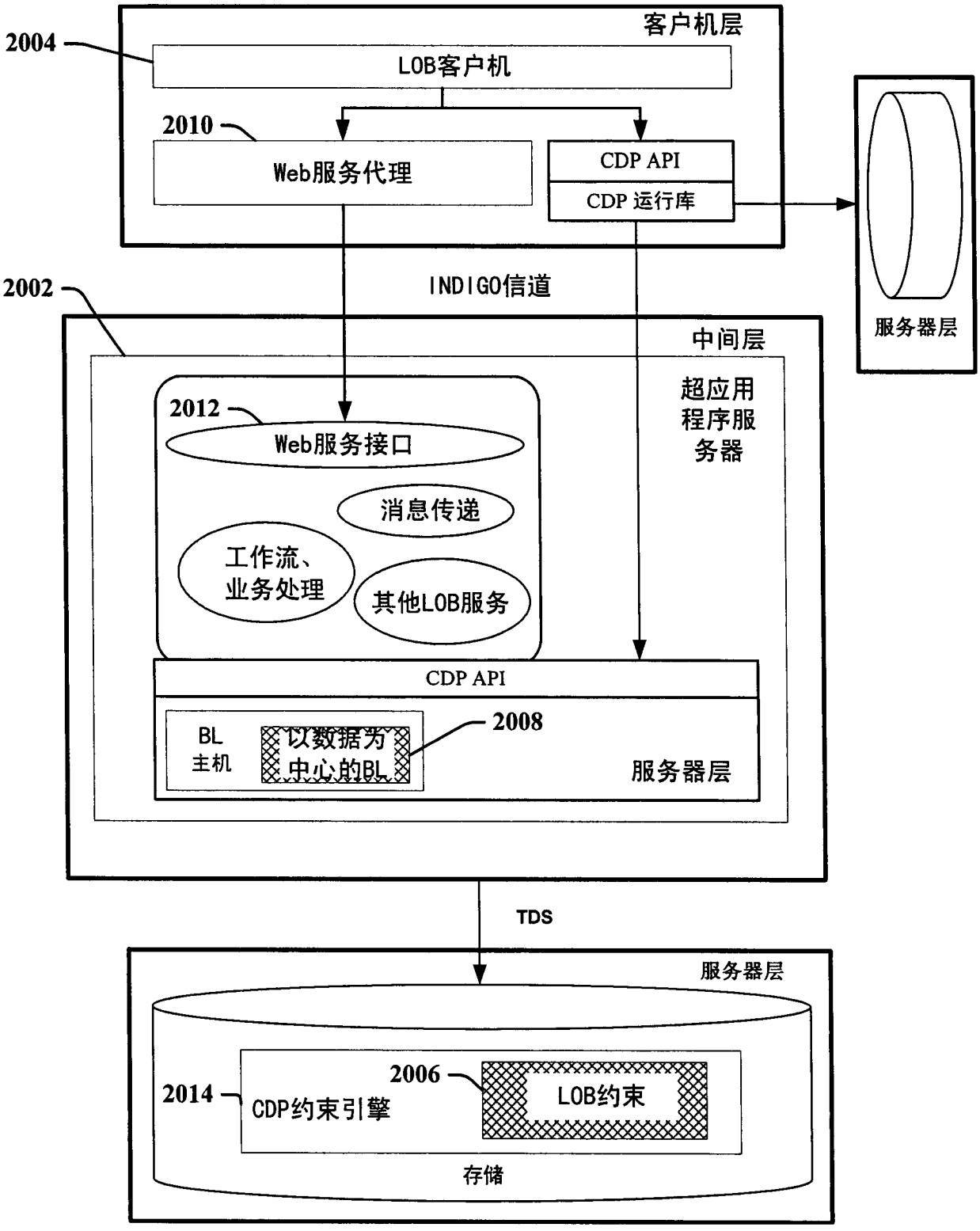

客户机层
2004
LOB客户机
2010
Web服务代理
CDP API
CDP 运行库
服务器层
INDIGO信道
2002
中间层
超应用程序服务器
2012
Web服务接口
消息传递
工作流、业务处理
其他LOB服务
CDP API
BL主机
以数据为中心的BL
2008
服务器层
TDS
服务器层
2014
CDP约束引擎
2006
LOB约束
存储

图 20

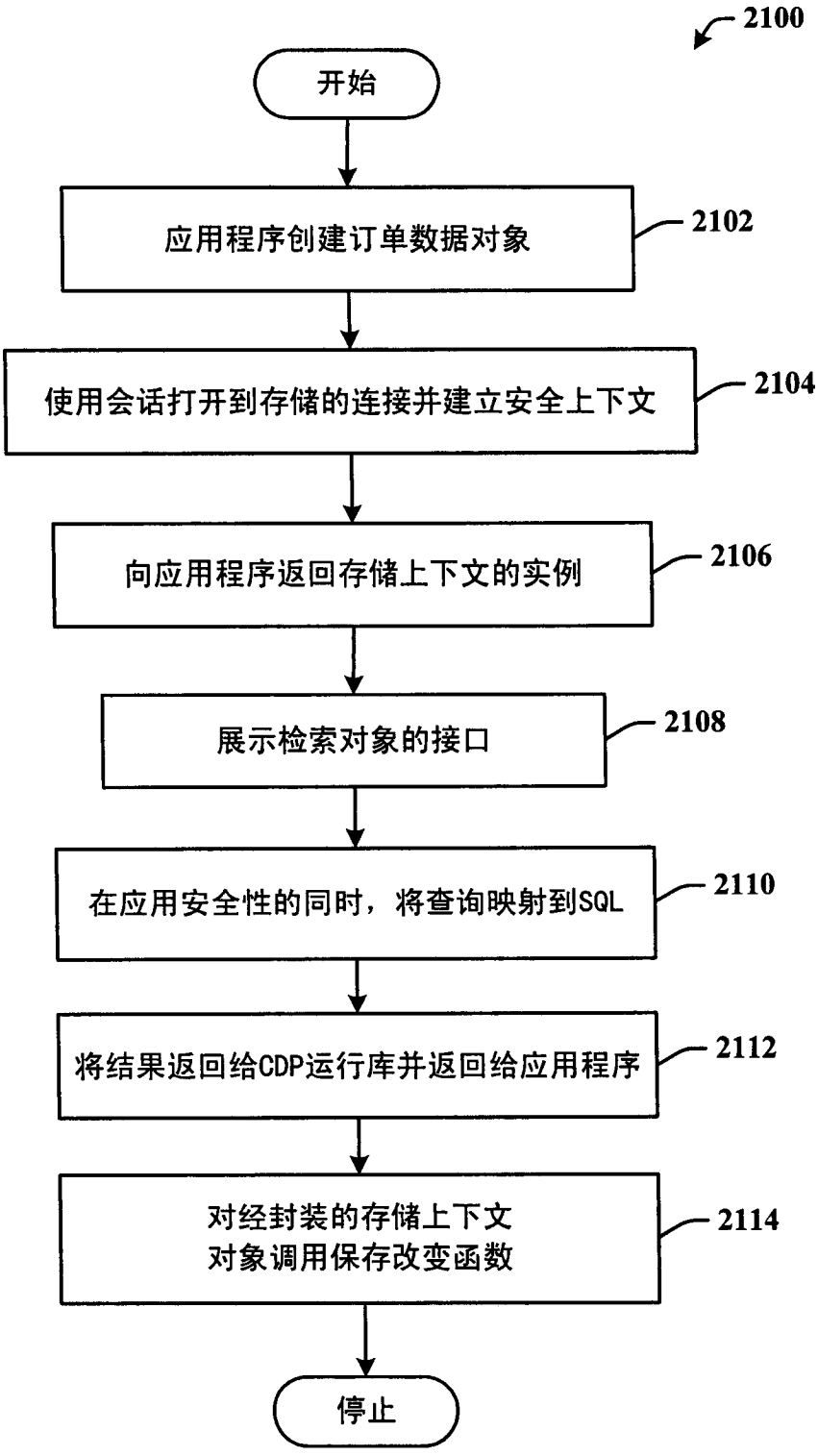

2100
开始
应用程序创建订单数据对象
2102
使用会话打开到存储的连接并建立安全上下文
2104
向应用程序返回存储上下文的实例
2106
展示检索对象的接口
2108
在应用安全性的同时，将查询映射到SQL
2110
将结果返回给CDP运行库并返回给应用程序
2112
对经封装的存储上下文
对象调用保存改变函数
2114
停止

图 21

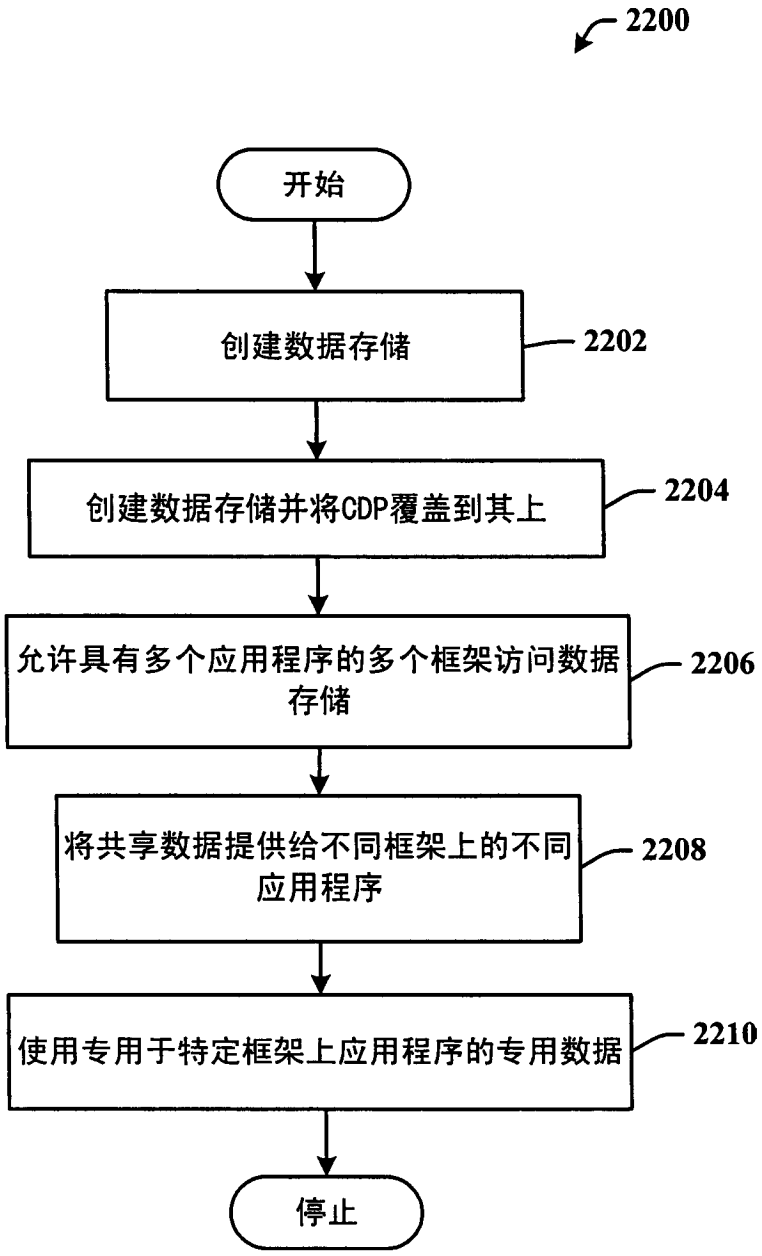

2200
开始
创建数据存储
2202
创建数据存储并将CDP覆盖到其上
2204
允许具有多个应用程序的多个框架访问数据存储
2206
将共享数据提供给不同框架上的不同应用程序
2208
使用专用于特定框架上应用程序的专用数据
2210
停止

图 22

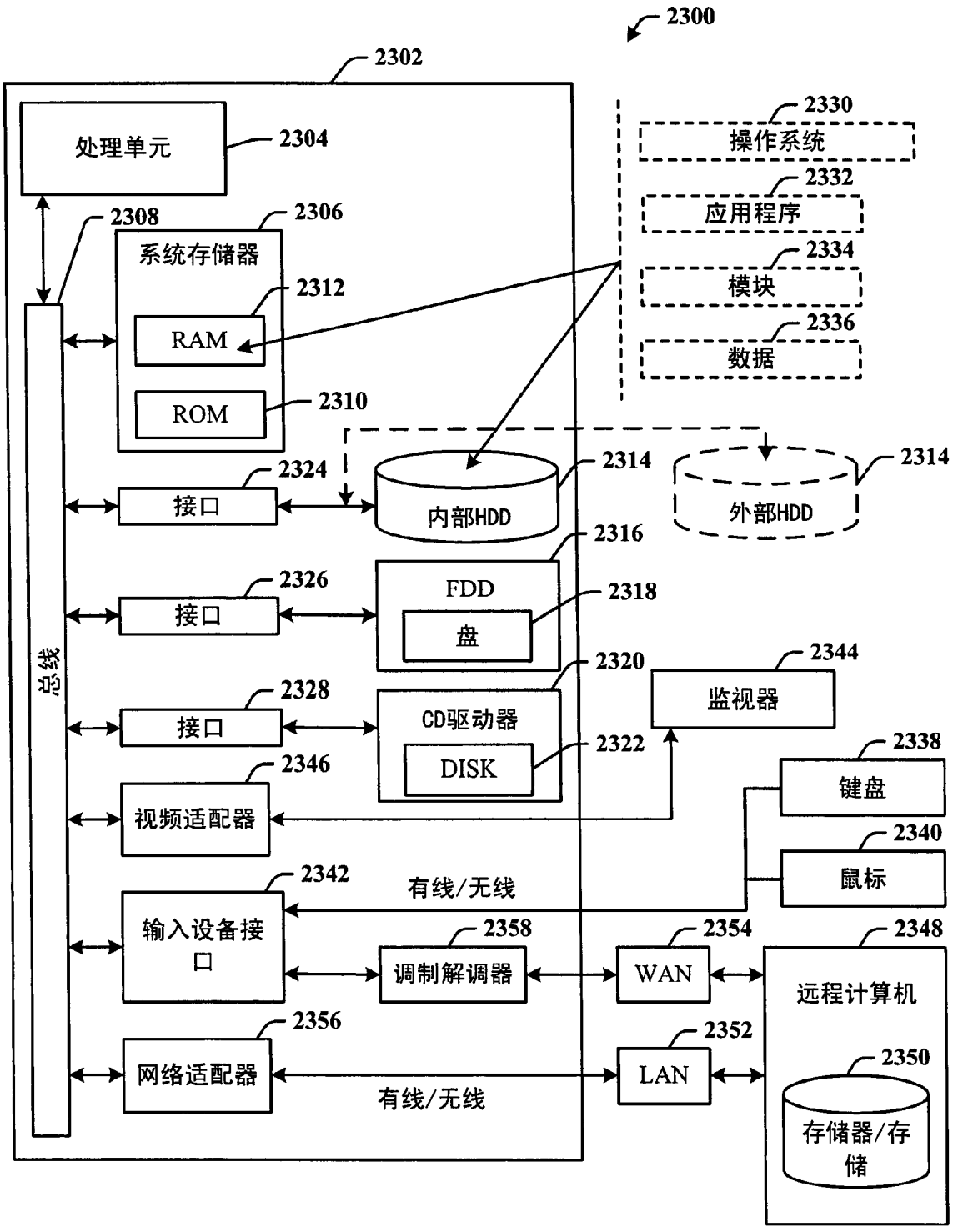

2300
2302
处理单元
2304
2308
2306
系统存储器
2312
RAM
ROM
2310
2330
操作系统
2332
应用程序
2334
模块
2336
数据
2324
接口
内部HDD
2314
外部HDD
2314
2316
2326
接口
FDD
盘
2318
2320
2344
监视器
2328
接口
CD驱动器
DISK
2322
2338
键盘
2346
视频适配器
2340
鼠标
2342
有线/无线
输入设备接口
2358
2354
2348
调制解调器
WAN
远程计算机
2356
2352
网络适配器
LAN
2350
有线/无线
存储器/存储
总线

图 23

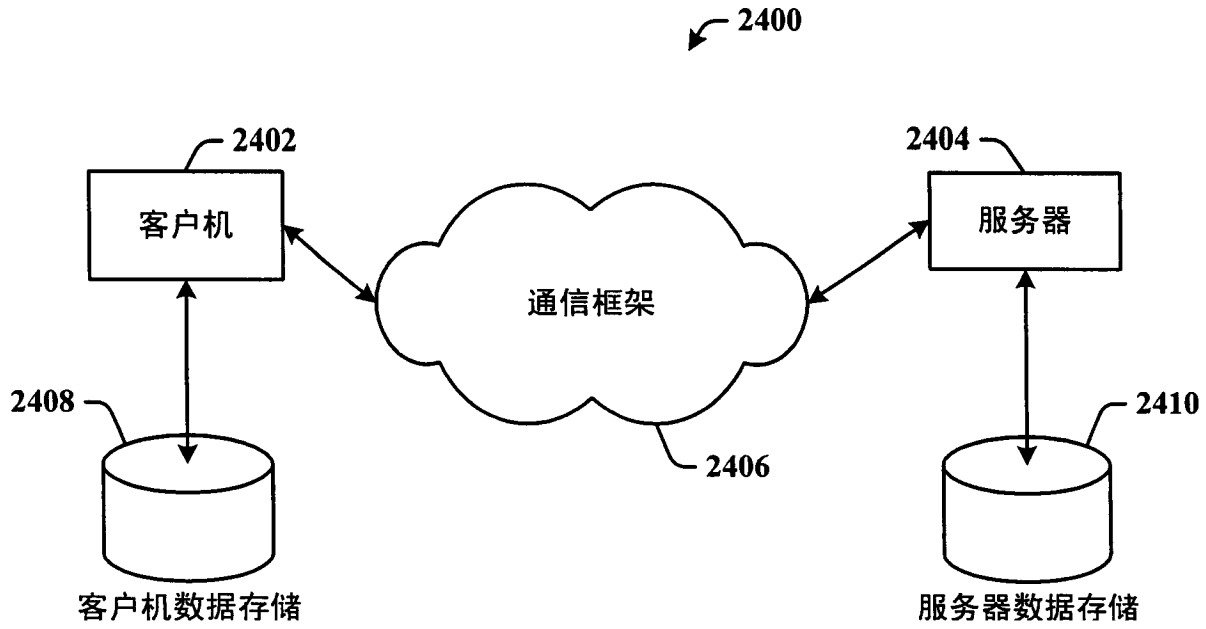


图 24