[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 17/30 (2006.01)
G06Q 10/00 (2006.01)

[12] 发明专利说明书

专利号 ZL 200610006100.0

[45] 授权公告日 2009 年 8 月 19 日

[11] 授权公告号 CN 100530181C

[22] 申请日 2002.4.25
[21] 申请号 200610006100.0
分案原申请号 02812143.0
[30] 优先权
[32] 2001. 4.25 [33] US [31] 60/286,466
[32] 2001.12.13 [33] US [31] 10/021,855
[73] 专利权人 BEA 系统公司
地址 美国加利福尼亚州
[72] 发明人 米歇尔·比森 蒂莫西·布里登
查尔斯·帕克拉特 汤姆·斯塔姆
史蒂文·威尔科克斯
[56] 参考文献
US5812865A 1998.9.22
US5754939A 1998.5.19
CN1201949A 1998.12.16
US6052685A 2000.4.18
审查员 庄锦军
[74] 专利代理机构 北京市柳沈律师事务所
代理人 郭定辉 黄小临

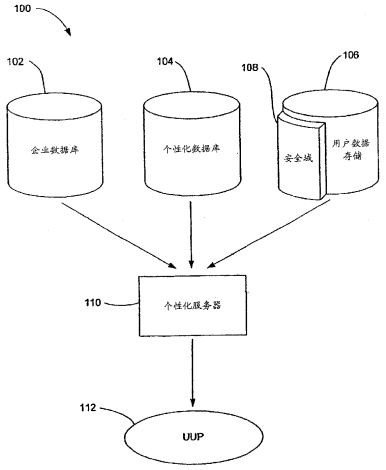
权利要求书 2 页 说明书 12 页 附图 6 页

[54] 发明名称

用于生成统一用户特征集的系统和方法

[57] 摘要

本发明包括生成统一用户特征集(112)的系统及方法，该系统包括第一数据源和第二数据源，以及允许访问所述第一与第二数据源的服务器，其中所述服务器将来自所述第一与第二数据源的数据聚合到统一用户特征集内。 统一用户特征集用来提供对多个数据源的透明接口。 取得基本用户 Java Bean 以通过个性化服务器(110)工作，并访问个性化数据库(104)。 该基本用户 Java Bean 提供了透明接口，通过该接口可以检索并更新隐式与显式的属性。 然后创建企业 Java Bean，以扩展该基本用户 Java Bean，使隐式与显式的属性也可以通过该透明接口从外部用户数据库进行检索与更新。



1. 一种用于生成统一用户特征集的系统，以允许对多个数据源的透明访问，该系统包括:

（a）第一数据源;

（b）第二数据源；以及

（c）允许访问所述第一与第二数据源的服务器，其中所述服务器将来自所述第一数据源与所述第二数据源的数据聚合到统一用户特征集内，

其中所述服务器提供透明接口以便从所述第一数据源与所述第二数据源存储和检索数据。

2. 根据权利要求 1 的系统，其中所述第一数据源选自包括遗留数据库、企业数据库以及用户数据存储的组。

3. 根据权利要求 1 的系统，其中所述第一数据源包含选自包括验证信息、用户列表、组列表与组成员的组的数据。

4. 根据权利要求 1 的系统，还包括安全域，该安全域允许验证在所述第一数据源与所述第二数据源中至少一个中的数据。

5. 根据权利要求 1 的系统，其中所述服务器为个性化服务器。

6. 根据权利要求 1 的系统，其中所述服务器包含企业 Java Bean。

7. 根据权利要求 6 的系统,其中所述企业 Java Bean 使用选自包括 getProperty()与 setProperty()的组的方法，检索并更新在所述第一数据源与所述第二数据源中至少一个中的数据。

8. 根据权利要求 1 的系统，其中所述服务器包含扩展 Java Bean。

9. 一种用于生成允许对多个数据源的透明访问的统一用户特征集的方法，该方法包括步骤:

（a）提供对第一数据源的访问;

（b）提供对第二数据源的访问；以及

（c）提供服务器来访问所述第一与第二数据源，其中所述服务器将来自所述第一数据源与所述第二数据源的数据聚合到统一用户特征集中，

其中所述服务器提供透明接口以便从所述第一数据源与所述第二数据源存储和检索数据。

10. 根据权利要求 9 的方法，其中所述第一数据源选自包括遗留数据库、

企业数据库以及用户数据存储的组。

11. 根据权利要求 9 的方法，其中所述第一数据源包含选自包括验证信息、用户列表、组列表与组成员的组的数据。

12. 根据权利要求 9 的方法，还包括:

（d）提供安全域，该安全域允许验证在所述第一数据源与所述第二数据源中至少一个中的数据。

13. 根据权利要求 9 的方法，其中所述服务器为个性化服务器。

14. 根据权利要求 9 的方法，其中所述服务器包含企业 Java Bean。

15. 根据权利要求 14 的方法,其中所述企业 Java Bean 使用选自包括 getProperty()与 setProperty()的组的方法，检索并更新在所述第一数据源与所述第二数据源中至少一个中的数据。

16. 根据权利要求 9 的方法，其中所述服务器包含扩展 Java Bean。

用于生成统一用户特征集的系统和方法

本申请是申请日为2002年4月25日、名为“个性化服务器统一用户特征集”、进入中国国家阶段的专利申请号为02812143.0的专利申请的分案申请。

本申请要求于2001年4月25日提交的、名为“个性化服务器统一用户特征集”、专利申请号为60/286,466的美国临时专利申请的优先权，此处包括其内容，以作参考。

技术领域

一般地，本发明涉及来自多个资源的数据的集成。

背景技术

企业一直在寻找更好的方法，以将新信息与其现存数据集成，如可以从第三方资源获取的有关当前或可能客户的信息。为使这种集成有效，公司必须能够简化该集成过程，去除不必要的费用或采购，并去除中断时间，一般需要该中断时间以实现新的数据系统或修改现存数据结构。

例如，一个企业可能希望将额外的用户特征集数据整合到其已确立的用户数据中。该额外的特征集数据可以被分离的资源（如个性化服务器）所配置与维护。经常该企业具有原来就存在的企业客户或用户数据，这些数据在该个性化服务器范围之外。该数据一般位于现存企业数据库中，可能包括每个客户的信息，如名称、社会保险号码、和/或公司特有的信息，例如特定航空公司的常客的飞行里程。该企业会希望将该数据无缝地集成到其个性化解决方案之中，从而尽可能地避免数据迁移的困难。

因此本发明的目的在于：为来自现存数据源的数据与来自外部来源的数

据之间的集成，开发一种无缝的方法。

发明内容

本发明包括一种用来生成统一用户特征集(unified user profile)的系统。该系统一种用于生成统一用户特征集的系统，以允许对多个数据源的透明访问，该系统包括：第一数据源和第二数据源，以及允许访问所述第一与第二数据源的服务器，其中所述服务器将来自所述第一与第二数据源的数据聚合到统一用户特征集内。

本发明还包括用来生成统一用户特征集的方法，该方法包括步骤：提供对第一数据源的访问；提供对第二数据源的访问；以及提供服务器来访问所述第一与第二数据源，其中所述服务器将来自所述第一与第二数据源的数据聚合到统一用户特征集中。

本发明还包括用来生成统一用户特征集的结构。该结构可以建立在基本用户企业 Java Bean 之上,该基本用户企业 Java Bean 可以被扩展以整合来自用户数据存储的现存用户数据。然后可以生成用户特有企业 Java Bean,其允许对现存用户数据的透明读和写访问。

本发明还包括一种用来生成统一用户特征集的方法。在一个实施例中，取得了适用于通过个性化服务器工作以访问个性化数据库的基本用户 Java Bean。该基本用户 Java Bean 提供了一种透明接口，通过该接口可以检索并更新隐式与显式的属性。然后创建企业 Java Bean，以扩展该基本用户 Java Bean，以便也可以从外部用户数据库检索并更新隐式与显式的属性。

本发明还包括一种用来透明地访问多个数据源的方法。在该方法中，取得了适用于通过服务器工作以访问内部数据源的基本用户 Java Bean。该基本用户 Java Bean 提供了一种透明接口，通过该接口可以从该内部数据源检索并更新隐式与显式的属性。然后扩展该基本用户 Java Bean，以便可以进一步提供一种透明接口，通过该接口可以从至少一个外部数据源中检索并更新隐式与显式的属性。

本发明还包括一种用来透明地访问多个数据源的系统。该系统使用服务器与多个数据源通信。在该系统中包括了扩展的用户 Java Bean，该 Java Bean 适用于提供通过该服务器的对数据源的透明访问。

附图说明

图 1 示出根据本发明的一个实施例的 UUP 配置;

图 2 示出根据本发明的一个实施例的 UUP 配置;

图 3 示出根据本发明的一个实施例的 UUP 配置;

图 4 示出根据本发明的一个实施例的 UUP 配置;

图 5(a)与 5(b)的流程图示出根据本发明的一个实施例的为隐式与显式的情况调用 setUserPoints()的步骤;

图 6 的流程图示出根据本发明的一个实施例的运行 ejbFind 例程的步骤。

具体实施方式

根据前面对本发明的概述，以下给出本发明实施例的详细描述，该实施例目前被认为是最好的模式。

本发明的结构定义了现存用户数据可以与更易动态变化的个性化数据进行整合的途径。在服务器中，例如用来为特定用户或用户组个性化内容或服务的个性化服务器，系统用户一般由用户特征集表示。用户特征集提供了用户 ID（标识）以及对用户属性的访问，如年龄或电子邮件地址。属性值可以是单值的或多值的，并可以通过将属性名称作为键值的 getProperty()函数或类似方法请求。

本发明的用户特征集的优点在于：其可以被扩展并定制，以从现存数据源检索用户信息。例如，与服务器（如个性化服务器）或解决方案一起装上的用户特征集，可以将用户属性（如来自个性化服务器数据库的属性与来自 LDAP 服务器或本领域已知的遗留数据库的用户属性）组合到单一用户特征集之中，以备在应用中使用。然后，开发人员与系统用户就不必担心不同的底层数据源。为取得用户信息，该用户特征集是唯一需要访问的地方。

本发明的统一用户特征集（Unified User Profile,UUP）包括这种属性聚合，其将来自现存数据源与个性化服务器数据库表的属性聚合到单一的定制的用户特征集之中。更具体地，UUP 通过扩展用户组件，结合了现存用户/客户数据。通过将该个性化服务器数据库表装入现存数据库实例之中，并且扩展用户实现，开发人员就能够迅速创建定制的 UUP，其能够从现存数据库中检索属性，并将属性存储/更新到现存数据库中。需要这种灵活性是因为其允许对现存数据的访问，而不对使用该数据的现存应用进行数据迁移或中断。然而

应该理解，如果需要，现存数据可以被迁移到分离的个性化服务器数据库实例中。

与其他服务器解决方案相比，该 UUP 的一个主要优点在于：该 UUP 不需要在数据库管理系统（如客户的关系型数据库管理系统）中进行数据库模式更新或数据迁移。该 UUP 最好通过编写扩展 EJB 来创建，而不是通过更新数据库表，或运行数据迁移脚本。现有技术的服务器经常要求为额外的用户属性而更新用户数据库表模式。

图 1-4 显示本发明的 UUP 系统的可能配置。在图 1 的第一配置 100 中，企业、遗留（legacy）、或其他外部数据库 102 与个性化服务器数据库 104 向个性化服务器 110 提供属性数据。个性化服务器 110 还从用户数据存储 106 接收信息，如验证信息、用户列表、组列表、以及组成员。该用户数据存储可以是任何适当的系统，如本领域已知的 LDAP、Unix 或 NT 系统。用户数据存储 106 还包括用于验证的安全区 108。个性化服务器数据库 104 与安全区 108 在本配置中保持分离，因为可能需要这种对验证与检索的分离，然而这对本发明的实现不是必须的。当用户或组已经存在于某类数据存储时，如 LDAP 目录，可以使用该配置。然后该现存的用户属性数据被个性化服务器 110 所取得，并被与个性化数据合并以生成 UUP 112。

当用户或组已经存在于用户数据存储 204（如 LDAP 目录）中，并且没有现存用户数据必须被结合到 UUP 210 中时，如图 2 所示的第二配置 200 可能有用。此时，所有的用户与组属性数据最好都存储在个性化服务器数据库 202 中。在本配置中，个性化服务器 208 最好仍利用用户数据存储 204 的安全区 206。

在没有现存用户与组的存储的情形，如图 3 所示的第三配置 300 可能有用。个性化服务器数据库 302 的表包含所有的用户与组数据，并且最好还包含独立的安全区 304。此时个性化服务器 306 在生成 UUP 308 时只需要查看个性化服务器数据库 302。

当用户、组、以及属性数据位于企业、遗留、或其他外部数据库 402 中，并且必须由个性化服务器 408 聚合到 UUP 410 中时，如图 4 所示的第四配置 400 可能有用。此时必须创建定制安全区 404 以通过个性化服务器使用现存用户与组。该定制安全区不必一定与外部数据库 402 存储在一起，但可以被聚合到个性化数据库 406 之中。同样，检索与验证区最好保持分离。

本发明的结构的一个实施例依赖于三个主要来源以将数据聚合到 UUP 中：（1）基本用户企业 Java Bean（EJB），（2）用户数据存储，以及（3）用户特用企业 Java Bean（EJB）。

基本用户 EJB 为最好由个性化客户所扩展的 JAVA 类，以将现存用户数据聚合到个性化解决方案之中。

基本用户 EJB 最好提供单一的透明接口，通过该接口可以检索或更新隐式的属性与显式的属性。该基本用户 EJB 利用属性集合，可以使用该集合以给出属性的命名空间限定，以及定义属性类型、允许值等等。属性集合的行为类似用于用户属性的数据模式。此处所指的透明性一般指用户或应用可以进行调用或请求，而不用关心数据存储在哪里或者该数据使用哪种命名约定(naming convention)。如果该数据在遗留数据库中而不是个性化数据库中，则 UUP 将自动处理该请求，而该用户或应用永远不用知道其位置或名称。

在本发明的一个实施例中，基本用户 EJB 的子类使用两种方法以检索或更新：getProperty 与 setProperty。这些方法对隐式的或显式的属性都可以检索和/或更新。这些方法可以设置如下：

```
public Object getProperty(String propertySetName, String propertyName);
public void setProperty(String propertySetName, String propertyName,
                        Object propertyValue);
```

这些方法最后使用两个主要的属性：propertySetName 与 propertyName。propertySetName 属性指明此调用所采用的数据模式。这样，propertySetName 作为待检索或更新的属性的命名空间。propertyName 属性指明待更新或检索的属性的名称。使用 propertySetName-propertyName 对的一个优点是：可以在多个应用或子应用范围内使用单一的 propertyName。属性名称的多个实例也可以对应不同的定义。从基本用户 Bean 检索的属性将被称为隐式属性。

用户数据存储可能在聚合之前就已经存在，一般是其中存储有关当前用户或客户数据的数据库或表。该表可以与个性化服务器的数据库实例位于一处。该现存用户数据存储可以包含独立于个性化服务器数据库表而存在的用户数据。该个性化服务器可以要求现存用户数据存储与个性化服务器表存在于同一 RDBMS 实例中。

表 1 显示了示例的“AcmeCustomer” RDBMS 表。该表为每个客户定义三个值：（1）Customer_Name（客户名称），（2）Acme_Point（顶点），以及（3）

Acme_DisCount（顶点折扣）。Customer_Name 被用作每个客户的唯一标识符。一旦集成完成后，该唯一标识符最好被用来在整个个性化服务器内唯一地辨别用户。Acme_Points 为样品客户价值。顶点客户（Acme customer）可能在他或她购买顶点产品(Acme products)时收集点数。Acme_Discount 为客户在每次购买顶点产品时所获得的折扣。

表 1—AcmeCustmer(顶点客户)RDBMS 表

Customer_Name	Acme_Points	Acme_Discount
bsmith	50000	0.2
jpatadia	100000	0.1
ughandi	85000	0.15
mbisson	65000	0.3
kdickson	32000	0.05
tsamm	200000	0.2
tcook	100000	0.1

一旦完全理解了现存数据存储，则可以编写扩展基本用户 EJB 的 EJB，其利用透明的值检索与更新服务。计算机领域的技术人员应该熟知扩展 Java Bean 的方法。

为集成现存用户数据与由个性化服务器所提供的个性化表，可以编写 EJB 以扩展用户 Bean，该用户 Bean 可以与个性化服务器一起提供。一旦完成了该 Bean，个性化服务器的客户就具有对以前存储在用户特有数据库表（显式的属性）中的属性、以及存储在个性化服务器的属性表集合中的属性（隐式的属性）的透明读和写访问。

继续“顶点客户”数据存储的例子，可以编写 AcmeUser EJB(顶点用户 EJB)，其提供对现存表的数据更新与检索机制。为在本发明的限定内运行，该 AcmeUser EJB 可以定义如下方法：

public Long getAcmePoints()—返回客户到目前为止所收集的点数。

public void setAcmePoints(Long newAcmePointsValue)—更新客户的顶点点数。

public Double getAcmeDiscount()—返回客户当前的折扣。

public void setAcmeDiscount()—更新客户的顶点折扣。

从扩展的用户 Bean 所检索的属性被称为显式的属性。

一旦实现了扩展 Bean 的方法，则顶点点数与顶点折扣都可以用由基本用户 EJB 所实现的 getProperty()与 setProperty()的继承方法进行检索。

因此，例如对用户 bsmith，下面的两个方法都返回一包含值 50000 的长整形：

public Long getAcmePoints();

public Object getProperty(anyPropertySetName, “acmePoints”);

类似地，下面的每个方法都将 bsmith 的顶点点数更新为 60,000:

public void setAcmePoints(new Double(60000));

public void setProperty(anyPropertySetName, “acmetPoints”, new Double(60000));

在其对透明属性更新与检索的实现中，UUP 最好使用 Java 自反性（reflection）的概念，以在将属性当作隐式的处理并使用属性集合的概念之前确定该属性是否为显式的。自反性是 Java 编程语言的一项特点，其使运行的 Java 程序能够检查或反省自身，以操纵该程序的内部属性。如果 propterName 对应显式的属性，则最后在隐式的属性之前进行显式的属性的更新与检索。由于此搜索顺序，如果正在更新或检索显式的属性，则实际属性集合的名称无关紧要。

下面的例子展示了假想公司使用 UUP 以利用现存客户数据。该例子扩展了用户 Bean 并且从先前存在的数据库中检索数据。该例子显示了如何能够相对较容易地创建满足应用的保存(persistence)需求的定制的 UUP。下面的表 2 解释了可以扩展什么以创建定制 UUP。

表 2—示例扩展以创建定制 UUP

对象	必须扩展
UUP 主键	UserPk—没有附加键字段
UUP EJB 接口	User
UUP EJB 实现	UserImpl

UUP 为 ConfigurableEntity 这一事实意味着用户特征集具有显式地或隐式地设置与获取属性的概念。此处所称的显式地设置属性指直接调用属性的设置方法。隐式地设置属性指在没有显式的设置方法时，通过 setProperty()方法来设置属性。例如，如果 UUP 包含属性“userPoints”，直接调用 setUserPoints()

将显式地设置 userPoints 属性。用键值“userPoints”调用 setProperty()将隐式地设置 userPoints 属性。在被调用时，setProperty()先在用户特征集中查找 setUserPoints()设置方法来调用。如果存在这种设置方法，则该方法被调用，以设置该属性，并进行任何与值中修改有关的必要处理。最终，由 UUP 实现来负责保存被显式设置的属性值，即使其是通过 setProperty()被隐式调用的。最好只有在不存在显式的设置方法时，ConfigurableEntity 才处理被隐式设置的属性的保存。

图 5(a)与图 5(b)分别画出了对 setUserPoints()的显式 550 与隐式 500 调用。在这两种情况中，由 UUP Bean 负责存储 userPoints 值。如果在 UUP Bean 中不存在 setUserPoints()方法，则 ConfiruableEntity 实现将处理 userPoints 值的存储。在图 5(a)的隐式情况中，调用 setProperty()502。系统检查是否存在 serUserPoints()方法。如果存在，则调用 setUserPoints()506。如果不存在，则系统继续执行 setProperty 508。对于图 5(b)的显式情况，对 setUserPoints() 552 的调用将只调用 setUserPoints() 554。

这种隐式与显式地设置属性的概念允许在 UUP 实现中更高的灵活性。如果在获取或设置属性时需要发生任何特殊的逻辑，例如计算另一个值，则可以使用该属性的设置或获取方法来实现之。UUP 之外的功能可以确信具有用于属性访问的 setProperty()方法与 getProperty()方法，从而不再需要知道属性是否有自己的设置或获取方法。例如，即使 UUP 只提供了 getUserPoints()方法来检索 userPoints，<um:getproperty> JSP 标记也可以检索 userPoints 属性值。这是因为 UUP 的 getProperty()方法可以在检查其他地方之前先检查其是否具有 getUserPoints()。具有显式的设置与获取方法的属性被称为“显式属性”，而只能通过调用 setProperty()来进行设置的属性被称为“隐式属性”。

在实现定制 UUP EJB 时，可能只需要为 UUP 的显式属性实现显式的获取与设置方法。然后，这些设置与获取的实现将在现存数据存储中设置并检索这些属性值。

在一个实施例中，对 UUP 中所有的显式属性设置与获取都采用了获取 PropertyName()/设置 PropertyName()的方式。如果 UUP 具有显式 userPoints 属性，则提供显式 getUserPoints()方法，因为 retrieveUserPoints()将不工作。类似地，用 setUserPoints()方法设置 userPoints。在该实施例中，当通过隐式调用获取和设置属性时，getProperty()与 setProperty()方法查找符合此约定的

获取与设置方法。不允许重设 setProperty()或 getProperty()。对显式属性的获取与设置通过获取与设置方法完成。显式获取与设置方法使用并返回对象。基本数据类型如长整形与浮点数被包裹起来，如在 java.lang.Long 与 java.lang.Float 对象中，以与 ConfigurableEntity 的 getProperty()与 setProperty()方法相容。

如果提供了获取方法，则最好也提供设置方法，反之亦然，这是因为不能预测用户或应用何时会想设置或获取属性。例如，如果提供了获取方法用来从数据库表中检索属性值，但没有相应的设置方法，则调用 setProperty()将把该属性存储到个性化服务器表中。这不是我们所需要的，因为该值是从一个地方检索的，但是在另一个地方设置。下次检索该属性时，其将具有原来的值—而不是已设置的值。如果要提供只读属性，则可以实现空设置方法。

ConfigurableEntity 的 getProperty()方法的优选定义如下：

```
Public Object getProperty(   String propertySet,
                             String propertyName,
                             ConfigurableEntity explicitSuccessor,
                             Object defaultValue);
```

getProperty()方法最好按特定顺序搜索属性。例如，如果对用户没有发现属性，则可查询组来找该值。在这种情况中，该用户将从组继承该属性值。用 ConfigurableEntity 的术语来说， 组是用户的“后继（successor）”。如果在 ConfigurableEntity 中未发现属性，则查询 ConfigurableEntity 的后继。这样，ConfigurableEntity 可以从父辈实体继承并重设值。

后继可以是隐式或显式的。隐式后继是 ConfigurableEntity 的缺省后继，或者为特定属性集所设定的后继。显式后继最好是可以作为参数传递给 getProperty()方法的 ConfigurableEntity。下面是在优选 ConfigurableEntity 中存在的 getProperty()属性搜索的顺序：

为所指明的属性集合的该属性查看该实体。

为缺省（空）属性集合中的该属性查看该实体。

为保留的属性集合中的该属性（如果使用 LDAP 域，则为来自 LDAP 的属性）查看该实体。

在该实体的显式后继（如果已指明）中寻找该属性。

在所指明的属性集合的该实体的后继中寻找该属性。

在该实体的缺省后继中寻找该属性。

如果指明了（非空）属性集合，则寻找在该属性集合中所定义的缺省值。

返回传入 getProperty()方法的 defaultValue(缺省值)。

ConfigurableEntity 的 setProperty()方法的优选定义如下所示:

```
Public Object setProperty(    String propertySet,
                              String propertyName, Object value);
```

如果在该优选方法中，setProperty()被用来为属性集合设置不符合属性集合定义的属性，则抛出异常。例如，假设“UnifiedUserExample”属性集合被定义为具有整型的 userPoints 属性。如果有人试图为将“UnifiedUserExample”属性集合的 userPoints 属性设置为“abc”，则抛出异常，这是因为 userPoints 被定义为整型，而“abc”为文本。类似地，将布尔型属性值设置为“bar”也将导致异常，这是因为布尔型值限定于布尔型对象。

如果调用 setProperty()并且传递空作为属性集合，则可以在空属性集合中设置该属性值，该空属性集合被称为缺省属性集合。如上所述的 getProperty()搜索循序，最好在“保留”属性集合与后继中查找该属性值之前搜索该缺省属性集合。

“保留”属性集合最好为可以用来承载来自外部数据存储的属性值的只读属性集合。“保留”属性集合可以在个性化服务器中使用，如当从 LDAP 目录检索属性值时。试图设置“保留”属性集合中的属性可以导致抛出异常。“保留”属性集合中的属性及保留属性集合本身不能通过用户管理工具编辑。优选的用户管理工具使用户与组的属性规范可以从 LDAP 或其他服务器中检索。然后在运行时，只检索这些属性。

可以通过 setProperty()用不存在的特定属性集合设置属性。这可能不是我们所希望的。当这样做时，并没有为所指明的属性集合名称临时创建属性集合。而是所指明的属性集合名称只作为该属性的命名空间。类似地，我们可能不希望通过 setProperty()来为现存属性集合设置在该属性集合中不存在的属性。通过这些途径设置的属性不能通过用户管理工具编辑，而在“空”或“缺省”属性集合中的属性可以用户管理工具编辑。

如果调用 setProperty()时传递 java.lang.Integer 对象值，则调用 getProperty()时最好返回 java.lang.Long 对象。检索此类属性的代码可能如下所示:

```
Object value =myUser.getProperty( “my_property_set”,
```

"my_integer_property", null, null);

Number tempNumber=(Number)value;

int realValue = tempNumber.intValue();

如果以 java.lang.Float 对象调用 setProperty()，则调用 getProperty()时最好返回 java.lang.Double 对象。检索此类属性的代码可能如下所示:

Object value =myUser.getProperty("my_property_set",

"my_float_property", null, null);

Number tempNumber=(Number)value;

float realValue = tempNumber.intValue();

用户对象最好提供用于 EJB 查找操作的功能，该功能将简化 UUP 与个性化服务器的集成。图 6 示出 ejbFind()操作的流程图。扩展的 UUP ejbFind()在现存数据存储中搜索记录 602。如果成功，则调用 super.ejbFind() 604，即用户对象 ejbFind()。如果成功，则该用户对象 ejbFind()将为该 UUP 在个性化服务器数据库表中创建所需的记录（如果这些记录不存在的话 610），并返回适当的主键。如果用户对象 ejbFind()失败，则检查的底层安全域 608，以确定该用户名是否与有效用户对应。如果是，则用户对象 ejbFind()创建所需记录 610，由此消除了查找方法错误以及原来将用户数据迁移到个性化服务器用户数据库表中所需的时间。如果 ebjFind()失败或者该域中不存在该用户，则会遇到查找方法错误 606。如果没遇到查找方法错误，则返回适当的主键 612。

如果此配置中该域无法确认用户的存在，但又必须创建该用户，则可由 EJB 负责来创建原来未找到的超类记录。

在生成定制 UUP 的一个实施例的最后一步，要求将 UUP 在个性化或其他服务器上注册，例如通过用户管理工具。为了注册该 UUP，优选的用户管理工具使用下表 3:

表 3—注册 UUP（例子）

特征集类型名称	任意名称，以后通过用户管理系统的<um:getprofile>JSP 扩展标记来使用该名称以引用该特征集类型
Profile Home Class	该新特征集类型的主类
Profile Remote Interface	该新特征集类型的远程接口
Profile Primary Key Class	该新特征集类型的主键类

Profile JNDI Name	该新特征集类型的 JNDI 查找名称

通过将 UUP 在个性化服务器上注册，便可以用<um:getprofile> JSP 标记请求该新特征集类型：

<um:getprofile profileType="UnifiedUserExample"

profileKey="<%=username%>"/>

这样便可能通过<um:getproperty>与<um:setproperty> JSP 标记使用 UUP。

LDAP 属性检索支持

除对隐式与显式属性的透明检索/更新外，统一用户特征集机制的一个实施例还支持从 LDAP 服务器检索用户数据，而不需要来自个性化服务器客户的 Java 代码。借助一组管理工具，可以在应用运行时用属性请求从 LDAP 服务器检索用户与组属性的规范。优选的方案是：对个性化服务器客户检索 LDAP 属性的唯一要求就是该客户部署 LDAP 安全域。在运行时，UUP 查询特定配置信息，以检测当前是否在使用 LDAP 安全域。如果是，则待检索的用户与组属性的名称可以从 LDAPConfiguration(LDAP 配置)会话 Bean 得到，并且 f 检索当前用户的适当属性。可以不要求客户使用 LDAP 安全域，也能获得其他 UUP 功能的好处。

本发明的其他特征、方面与目标可以从附图与权利要求中发现。应该理解可以开发本发明的其他实施例，而其都落入本发明与权利要求的精神与范围内。

以上对本发明优选实施例的描述用于显示与说明目的。其并非穷尽，也非用来将本发明限定于所公开的具体形式中。显然，对本领域的普通技术人员来讲，明显可有许多改进与变化。选择并描述实施例是为了最清楚地解释本发明的原理及其实际的应用，以使本领域其他技术人员能够理解本发明，其可应用于与所构想的特定用途相适应的各种实施例与各种修改。本发明的范围由所附权利要求界定。

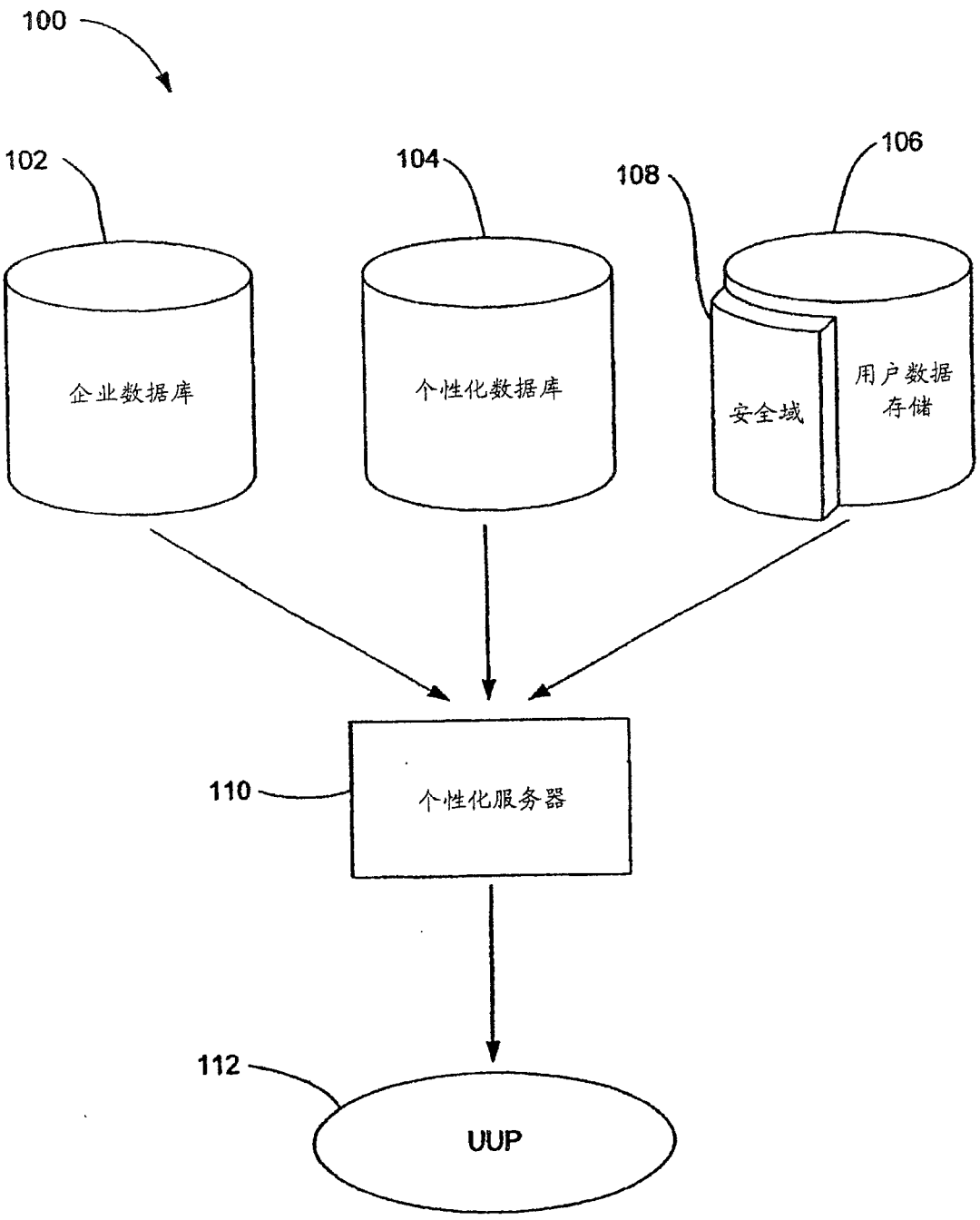

100
102
104
106
108
企业数据库
个性化数据库
安全域
用户数据存储
110
个性化服务器
112
UUP

图 1

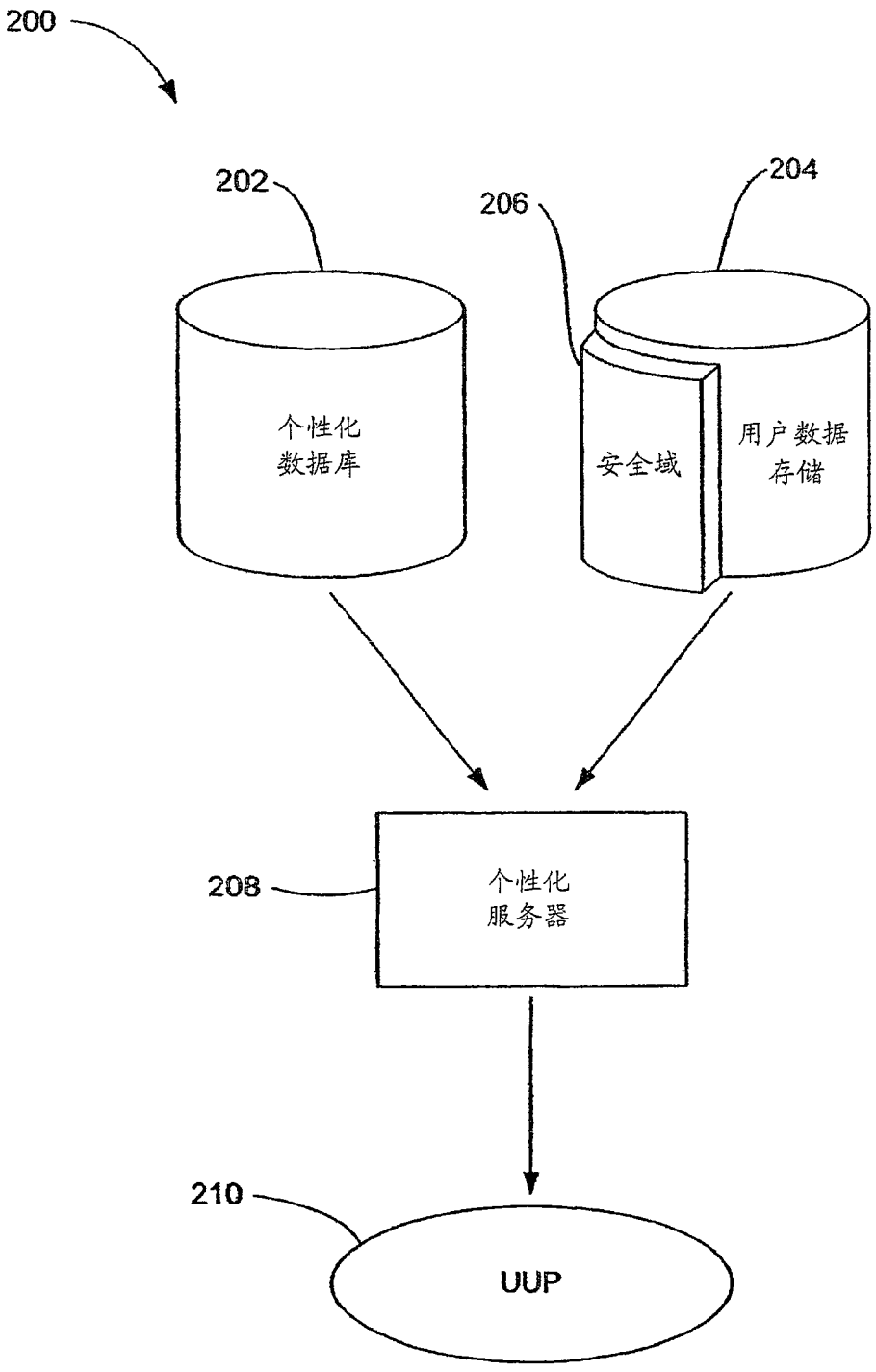


图 2

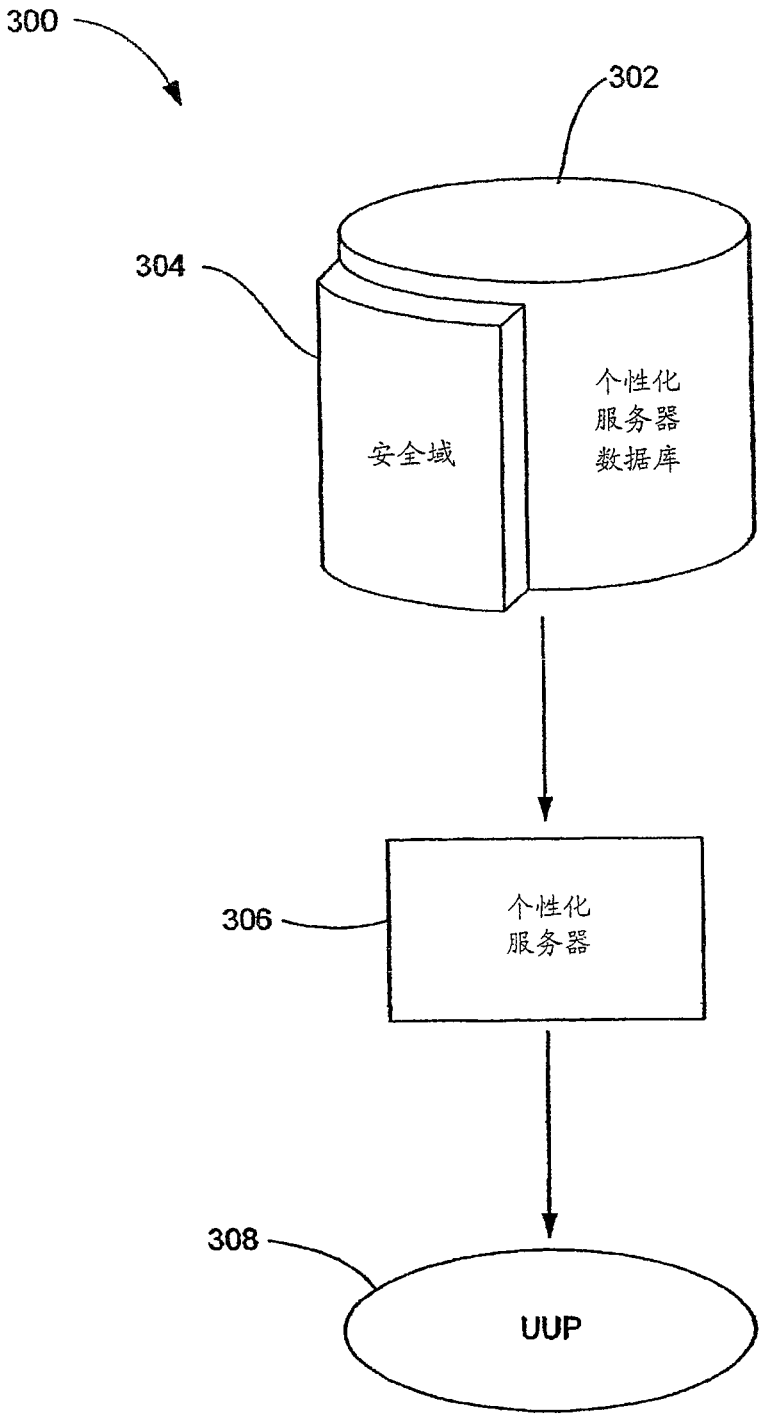


图 3

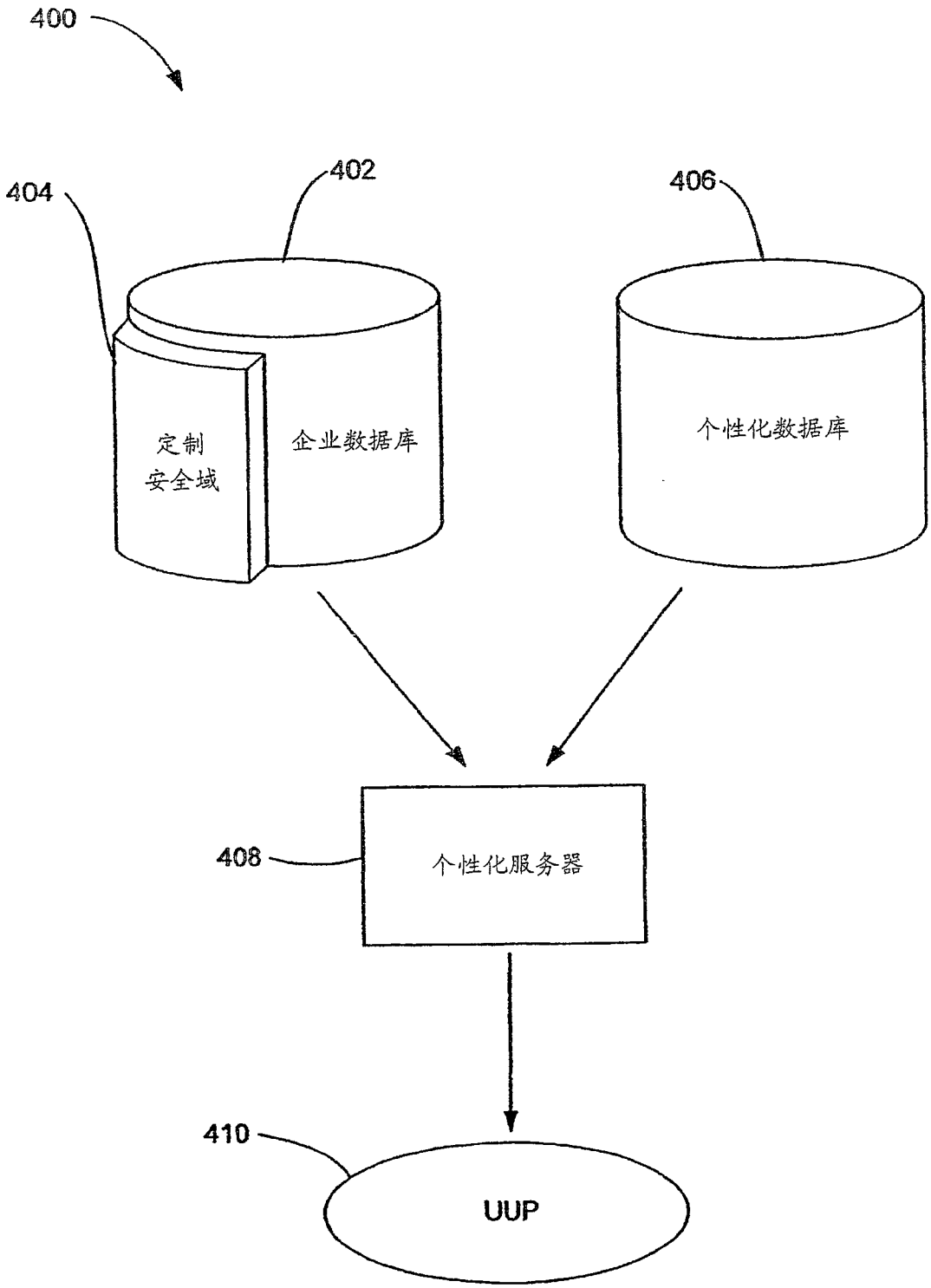


图 4

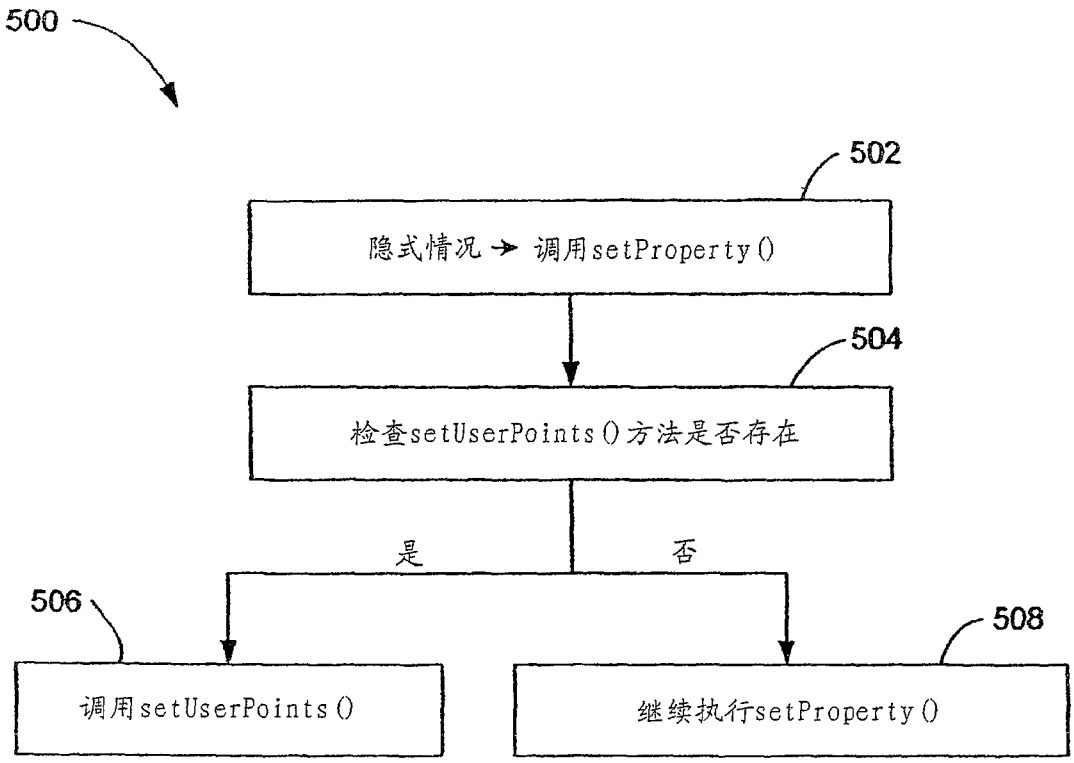

500
502
隐式情况 → 调用setProperty()
504
检查setUserPoints()方法是否存在
是
否
506
调用setUserPoints()
508
继续执行setProperty()

图 5a

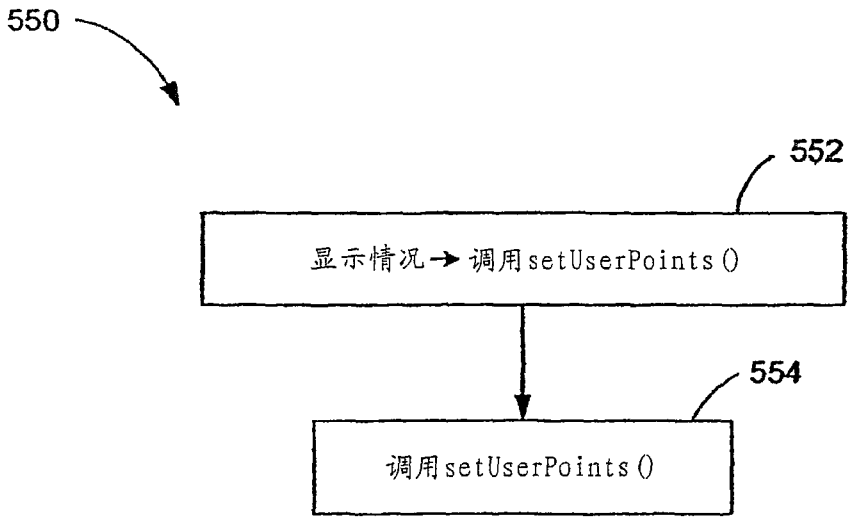

550
552
显示情况→调用setUserPoints()
554
调用setUserPoints()

图 5b

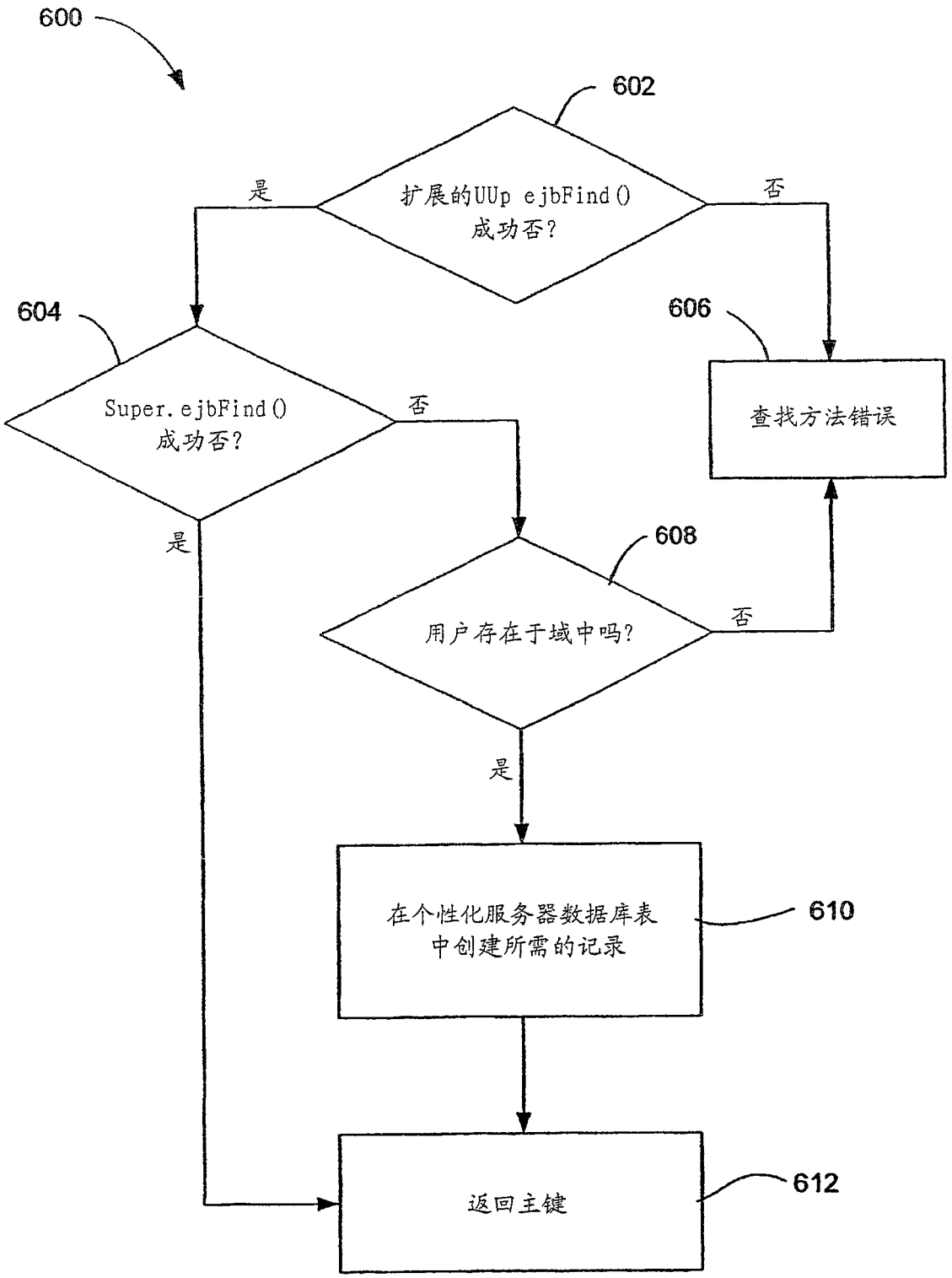

600
602
扩展的UUp ejbFind()
成功否?
是
否
604
Super.ejbFind()
成功否?
606
查找方法错误
否
是
608
用户存在于域中吗?
否
是
在个性化服务器数据库表
中创建所需的记录
610
返回主键
612

图 6