

第 52 章

目录服务

本章内容:

- Active Directory 的体系结构和概念
- 访问 Active Directory 的工具
- 如何读取和修改 Active Directory 中的数据
- 搜索 Active Directory 中的对象
- 以编程方式管理用户和组
- 使用 DSML(Directory Service Markup Language, 目录服务标记语言)访问 Active Directory

Microsoft 的 Active Directory 是一种目录服务, 它提供用户信息、网络资源和服务等中心分层存储器。还可以扩展这个目录服务中的信息, 同时存储企业感兴趣的自定义数据。例如, Microsoft Exchange Server 和 Microsoft Dynamics 广泛使用 Active Directory 来存储公共文件夹和其他项。

在 Active Directory 发布之前, Exchange Server 使用它自己的私有存储器来存储对象。系统管理员必须为一个人配置两个用户 ID: Windows NT 域中的用户账户(启用登录), 和 Exchange Directory 中的用户账户。这是必需的, 因为需要用户的其他信息(如电子邮件地址, 电话号码等), NT 域的用户信息不能扩展, 以添加需要的信息。

目前系统管理员只需要在 Active Directory 上为一个人配置一个用户账户; user 对象的信息可以扩展, 以便它满足 Exchange Server 的要求。我们也可以扩展这些信息。例如, 可以在 Active Directory 上用技能列表扩展用户信息。然后, 就可以搜索需要的 C#技能, 跟踪 C#开发人员。

本章介绍如何在 .NET Framework 中使用 System.DirectoryServices、System.DirectoryServices.AccountManagement 和 System.DirectoryServices.Protocols 名称空间中的类访问和处理目录服务中的数据。



本章使用兼具 Active Directory 配置的 Windows Server 2008 R2, 也可以使用 Windows 2003 Server 或其他目录服务。

在讨论了体系结构和如何对 Active Directory 编程之后, 就要创建一个 Windows 应用程序, 在该应用程序中指定一些属性和一个筛选器, 搜索 user 对象。与其他章节一样, 本章的示例代码也可以从 Wrox 网站 www.wrox.com 或随书附赠光盘中找到。

52.1 Active Directory 的体系结构

在开始对 Active Directory 编程前, 必须知道 Active Directory 的工作方式、用途, 以及什么数据可以存储在 Active Directory 中。

52.1.1 Active Directory 的功能

Active Directory 的功能可以总结为:

- Active Directory 中的数据以分层的方式组合。对象可以存储在其他容器对象中。用户并不是放在一个大型用户列表中, 而是组合到组织单元中。因为组织单元可以包含其他组织单元, 所以以这种方式可以构建一个树型视图。
- Active Directory 使用多主机复制方式(multimaster replication)。在 Active Directory 中, 每个域控制器(DC)都是主机。在多主机模型中, 更新可以应用于所有 DC。与单主机模型相比, 这个模型的伸缩性比较高, 因为可以同时在不同的服务器上更新。该模型的缺点是复制起来比较复杂。本章后面会讨论复制问题。
- 灵活的复制拓扑(replication topology), 这通过 WAN 中的慢速链接支持复制。数据复制的频率由域管理员配置。
- Active Directory 支持开放标准。LDAP(Lightweight Directory Access Protocol, 轻型目录访问协议)是一个 Internet 标准, 该标准用于访问许多不同目录服务, 包括 Active Directory 中的数据。在 LDAP 中, 也定义一个编程接口 LDAP API。LDAP API 可以使用 C 语言来访问 Active Directory。在 Active Directory 中使用的另一个标准是 Kerberos, 它用于身份验证。Windows Server Kerberos 服务也可用于验证 UNIX 客户端的身份。
- 活动目录服务接口(Active Directory Service Interface, ADSI)定义访问目录服务的 COM 接口。ADSI 可以访问 Active Directory 的所有功能。System.DirectoryServices 名称空间中的类包装 ADSI COM 对象, 用于从 .NET 应用程序中访问目录服务。
- 目录服务标记语言(Directory Service Markup Language, DSML)是另一个访问目录服务的标准, 它是独立于平台的方法, 得到 OASIS 组的支持。
- 细粒度安全性。对于 Active Directory, 细粒度安全性可用。存储在 Active Directory 中的每个对象都可以有一个关联的访问控制列表, 该列表确定谁可以对对象进行哪些处理。

目录中的对象都是强类型化的, 这说明, 对象的类型是精确定义的, 没有给对象添加未指定的属性。在架构中, 定义了对象类型和对象的各部分(特性)。特性可以是必选的, 也可以是可选的。

52.1.2 Active Directory 的概念

在编程之前, 必须了解 Active Directory 的一些基本术语和定义。

1. 对象

可以在 Active Directory 中存储对象。对象可以是用户、打印机或网络共享。对象包含描述它们的必选特性和可选特性。例如, user 对象的特性包含姓、名、电子邮件地址和电话号码等。

图 52-1 显示了一个容器对象，即 Wrox Press，它包含一些其他对象：两个用户对象、一个联系人对象、一台打印机对象和一个用户组对象。

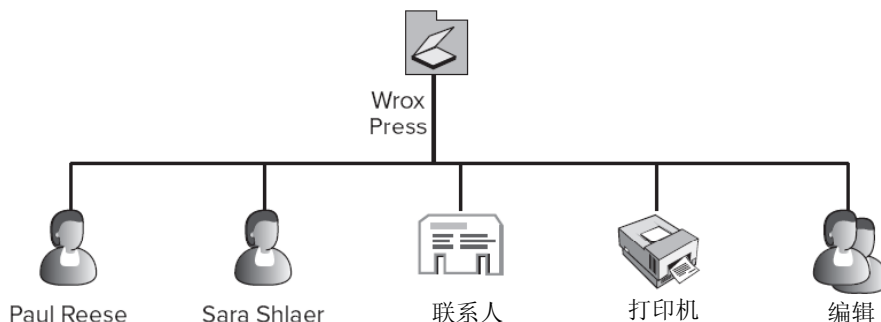


图 52-1

2. 架构

每个对象是类的一个实例，这个类在架构中定义。架构可以定义类型，架构本身存储在 Active Directory 的对象中。必须区分 classSchema 和 attributeSchema。对象的类型在 classSchema 中定义，其必选特性和可选特性也在 classSchema 中定义。attributeSchema 则定义特性的外观，以及特定特性的语法。

我们可以定义自定义类型和特性，并把它们添加到架构中。但要注意，不能从 Active Directory 中删除新架构类型。可以把它标记为未激活，这样就不能再创建新对象，但因为已有的对象可以是该类型，所以不能删除在架构中定义的类型或特性。

Administrator 用户组没有足够的权限创建新架构项，此时需要 Enterprise Admins 用户组。

3. 配置

除了对象和存储为对象的类定义之外，Active Directory 本身的配置也存储在 Active Directory 中。Active Directory 的配置存储所有站点的信息，如复制间隔等，这些都由系统管理员设置。因为配置本身存储在 Active Directory 中，所以可以像访问 Active Directory 中的所有其他对象那样访问配置信息。

4. Active Directory 域

域是 Windows 网络的安全边界。在 Active Directory 域中，对象以分层顺序进行存储。Active Directory 本身由一个或多个域组成。图 52-2 显示了域中对象的分层顺序，其中域用一个三角形表示。容器对象如 Users、Computers 和 Books 都可以存储其他对象。图 52-2 中的每个椭圆表示一个对象，对象之间的连线表示父-子关系。例如，Books 是 .NET 和 Java 的父对象，Pro C#、Beg C# 和 ASP.NET 是 .NET 对象的子对象。

5. 域控制器

一个域可以有多个域控制器，每个控制器存储域中的所有对象。没有主服务器，所有 DC 都一视同仁。这是一个多主机模型。可以在域中跨服务器复制对象。

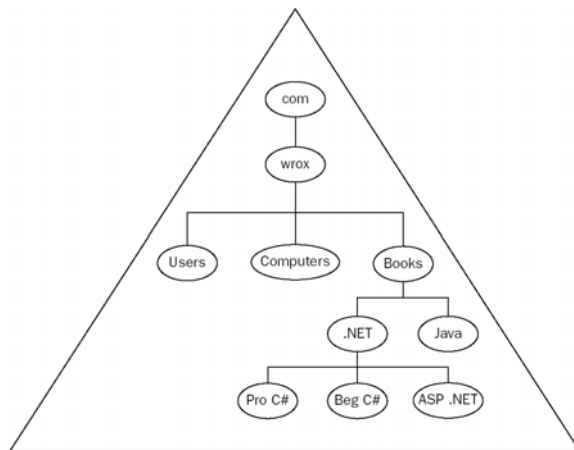


图 52-2

6. 站点

站点是网络中的一个位置，它至少拥有一个 DC。如果企业有多个办事处，办事处用慢速网络链接来连接，就可以在一个域中使用多个站点。由于备份或可伸缩性原因，每个站点都有一个或多个 DC 在运行。在一个站点的服务器之间复制数据，其间隔比较短，因为网络连接比较快。在跨站点的服务器之间进行复制的时间间隔可以配置得较长，这取决于网络的速度。当然，复制间隔由域管理员配置。

7. 域树

通过信任关系可以连接多个域。这些域共享公共架构、公共配置和全局目录(后面将介绍全局目录)。公共架构和公共配置表示可以跨域复制该数据。域树共享相同的类和属性架构。不能跨域复制对象自身。

以这种方式连接的域就构成了域树。域树中的域有一个连续和分层的名称空间。这表示，子域的域名是把子域的名称追加到父域名后。在这些域之间，建立使用 Kerberos 协议的信任关系。

例如，有一个根域 `wrox.com`，它是子域 `india.wrox.com` 和 `uk.wrox.com` 的父域。在父域和子域之间建立信任关系，这样域中的账户可以由其他域验证身份。

8. 森林

使用公共架构、公共配置和全局目录来连接多个域树，但没有使用连续的名称空间，该名称空间称为森林(forest)。森林是一组域树。如果公司有使用不同域名的子公司，就可以使用森林。例如，`wrox.com` 域应相对独立于 `wiley.com` 域，但应有一个公共管理，`wrox.com` 域的用户可以访问 `wiley.com` 域中的资源，反之亦然。使用森林，可以在多个域树之间建立信任关系。

9. 全局目录

对象的搜索可以跨越多个域。如果使用一些特性查找某个特定 `user` 对象，就必须搜索每个域。首先从 `wrox.com` 开始，接着搜索 `uk.wrox.com` 和 `india.wrox.com`，通过慢速链接，这样的搜索花费的时间会比较长。

要加快搜索速度，可以把所有对象都复制到全局目录 GC 中。GC 将被复制到森林的每个域中。

在每个域中至少有一个服务器包含 GC。出于性能和可伸缩性原因，在一个域中可以有多个 GC 服务器。使用 GC，可以在一个服务器上搜索所有对象。

GC 是所有对象的只读缓存，GC 只能用于搜索；必须使用域控制器更新。

并不是对象的所有特性都存储在 GC 中。可以定义特性是否和对象一起保存。特性是否存储在 GC 中，主要取决于该特性在搜索中使用的频率。如果特性在搜索中使用得很频繁，把它放在 GC 中，搜索就会比较快。用户的一幅图片在 GC 中不是很有用，因为我们从来不会搜索该图片。相反，在存储器中添加电话号码就比较有用。还可以定义该特性是否应被索引，如果进行索引，对该特性的查询就比较快。

10. 复制

程序员不喜欢配置复制，但因为会影响存储在 Active Directory 中的数据，所以必须了解它如何工作。Active Directory 使用了多主机服务器体系结构。域中的每个域控制器都可以进行更新。复制延迟时间定义了更新开始之前等待的时间。

- 如果某些特性发生变化，默认情况下，站点中每隔 5 分钟就发布一次变更通知，通知的内容是可配置的。因为发生改变的 DC 每隔 30 秒就通知另一个服务器，所以第 4 个 DC 可以在 7 分钟后得到变更通知。在默认情况下，把跨站点的变更通知设置为 180 分钟。站点之间和内部的复制可以配置为其他值。
- 如果没有发生改变，在站点内，每隔 60 分钟就进行一次预定复制。这将确保不遗漏一个变更通知。
- 对于敏感的安全信息，如账户被锁，会立即发出通知。

进行复制后，就只把改动的内容复制到 DC 中。在每次修改特性后，就会记录版本号(USN，更新序列号)和时间戳。如果更新不同服务器上的同一个特性，这就可以用于解决冲突。

下面看一个示例。用户 John Doe 的移动电话特性的 USN 号为 47。这个值已经复制到所有 DC 中。一个系统管理员改变电话号码。在服务器 DC1 上发生改变后；在服务器 DC1 上这个特性的新 USN 现在是 48，而其他域控制器的 USN 仍是 47。如果有人读取该特性，就会读取旧值，直到所有域控制器都进行复制为止。

下面的情况很少发生：另一个管理员改变电话号码特性，选择另一个域控制器，因为这个管理员从服务器 DC2 中接收到一个比较快的响应。在服务器 DC2 上，这个特性的 USN 也改为 48。

在通知的间隔期间，发出通知的原因是特性的 USN 改变，上次进行复制时，USN 的值是 47。使用复制机制可以检测到服务器 DC1 和 DC2 的电话号码特性的 USN 都是 48。虽然使用哪个服务器上的特性值并不重要，但必须使用其中一个服务器上的特性值。要解决这个冲突，就要使用改变的时间戳。因为 DC2 上的改变比较迟，所以会复制存储在 DC2 域控制器中的值。



在读取对象时，必须知道数据不一定是最新的。数据是否最新取决于复制延迟时间。在更新对象时，另一个用户可以在更新后仍获取旧值，同时还可能进行另一个更新操作。

52.1.3 Active Directory 数据的特征

假如 Active Directory 没有替代关系数据库或注册表，那么什么数据可以存储在 Active Directory 中？

- Active Directory 可以存储分层数据，容器也可以存储其他容器和对象。容器本身也是对象。
- 数据应主要用于读取。因为在一定的时间间隔中会进行复制，所以不能确定可以读取到最新的数据。在应用程序中，必须注意读取的信息有可能不是最新的信息。

- 数据应是企业普遍感兴趣的数据。因为给架构添加一个新数据类型，会把该数据类型复制到企业的所有服务器上。如果只有一小部分用户对该数据类型感兴趣，企业的域管理员就不会安装新的架构类型。
- 存储的数据量应合适，因为这些数据是要被复制的。如果存储数据量是 100KB，而且每星期仅修改一次数据，把它存储在目录中就不会出问题。但如果每小时修改一次数据，这个数据量就太大了。总是要考虑到数据复制到不同的服务器上、数据要传输到什么地方、复制的时间间隔等。如果数据量比较大，就要链接到 Active Directory 中，并把数据存储到另一个地方。总之，存储在 Active Directory 中的数据应分层组织，且数据量应合理，这对企业非常重要。

52.1.4 指定架构

Active Directory 对象是强类型化的。架构定义对象的类型、必选特性和可选特性，特性的语法和约束。如前所述，在架构中，必须区分架构对象和特性架构的对象。

类是特性的集合。通过类，就可以支持单一继承。从图 52-3 中可以看出，user 类派生自 organizationalPerson 类，而 organizationalPerson 类是 person 类的一个子类，它们的基类都是 top。classSchema 定义一个类，它用 systemMayContain 特性描述该特性。

图 52-3 只列出了几个 systemMayContain 值。使用 ADSI Edit 工具可以查看所有值。下一节将介绍这个工具。在 top 根类中，每个对象都有公共名称(cn)、displayName、objectGUID、whenChanged 和 whenCreated 特性。person 类派生自 top 类。person 对象也有 userPassword 和 telephoneNumber。organizationalPerson 派生自 person。除了 person 类的特性外，它还有 manager、department、company 特性，以及 user 登录系统所必需的其他特性。

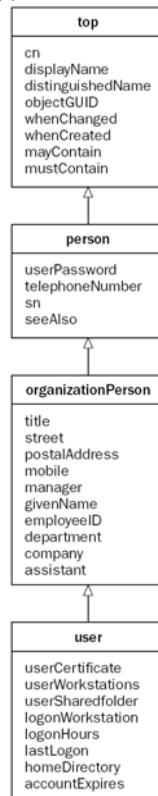


图 52-3

52.2 Active Directory 的管理工具

学习 Active Directory 的一些管理工具有助于理解 Active Directory、其中包含的数据，以及以编程方式可以完成的任务。

系统管理员可以用许多工具输入新数据，更新数据和配置 Active Directory。

- Active Directory Users and Computers MMC 插件用于更新用户数据、输入新用户。
- Active Directory Sites and Services MMC 插件用于配置域中的站点，在这些站点之间复制数据。
- Active Directory Domains and Trusts MMC 插件可以在树的域之间建立信任关系。
- ADSI Edit 是 Active Directory 的编辑器，可以在其中查看和编辑所有对象。



要在 Windows 7 上运行这些工具，需要安装 Windows 7 Remote Administration Tools。对于其他 Windows 版本，也需要单独下载这些系统管理工具。

下面几节介绍 Active Directory Users and Computers 和 ADSI Edit 工具的功能，因为这些工具对使用 Active Directory 创建应用程序非常重要。

52.2.1 Active Directory Users and Computers 工具

Active Directory Users and Computers 插件系统管理员用来管理用户的工具。选择 Start | Programs | Administrative | Tools | Active Directory Users and Computers 命令，就会启动这个程序，如图 52-4 所示。

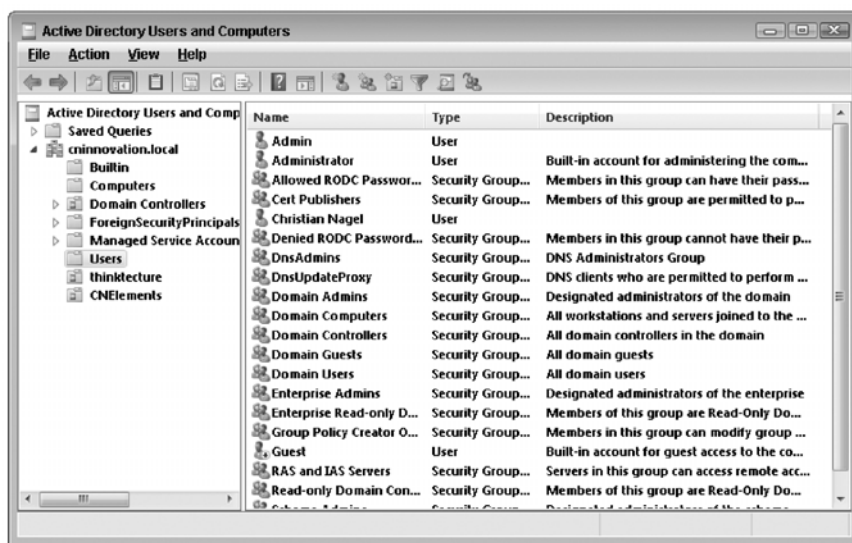


图 52-4

使用这个工具可以添加新用户、组、联系人、组织单元、打印机、共享文件夹或计算机，修改已有的项。在图 52-5 中，可以为 user 对象输入属性：办公室、电话号码、电子邮件地址、网页、公司信息、地址和组等。

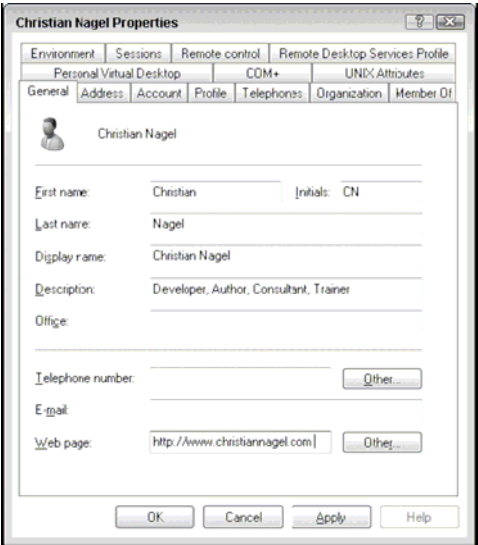


图 52-5

Active Directory Users and Computers 还可以用于有上百万对象的大公司。我们不必在有 1000 个对象的列表中查找，因为可以使用一个自定义筛选器，只显示某些对象。也可以执行 LDAP 查询，搜索公司中的对象。本章后面将讨论这些问题。

52.2.2 ADSI Edit 工具

ADSI Edit 是 Active Directory 的编辑器。如图 52-6 所示，ADSI Edit 提供的控制比 Active Directory Users and Computers 工具更多。使用 ADSI Edit，可以配置所有内容，也可以查看架构和对应配置。但使用这个工具并不是很容易，而且很容易输入错误的数据。

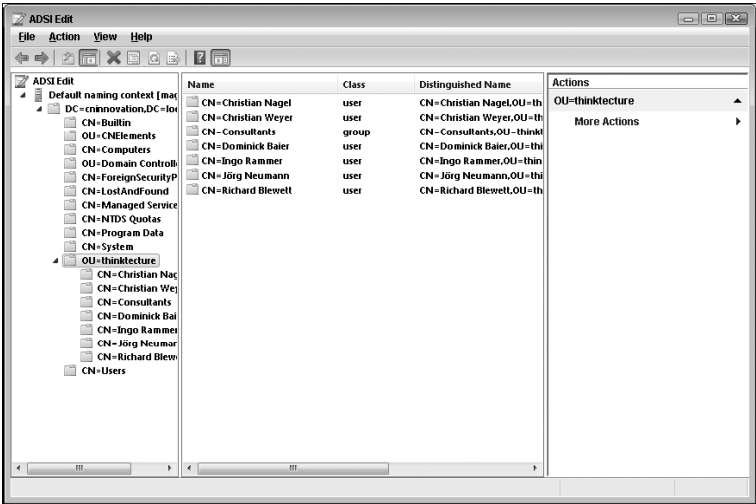


图 52-6

打开对象的 Properties 窗口，可以查看和修改 Active Directory 中对象的每个特性。利用该工具，可以查看必选特性、可选特性、特性的类型和值等，如图 52-7 所示。

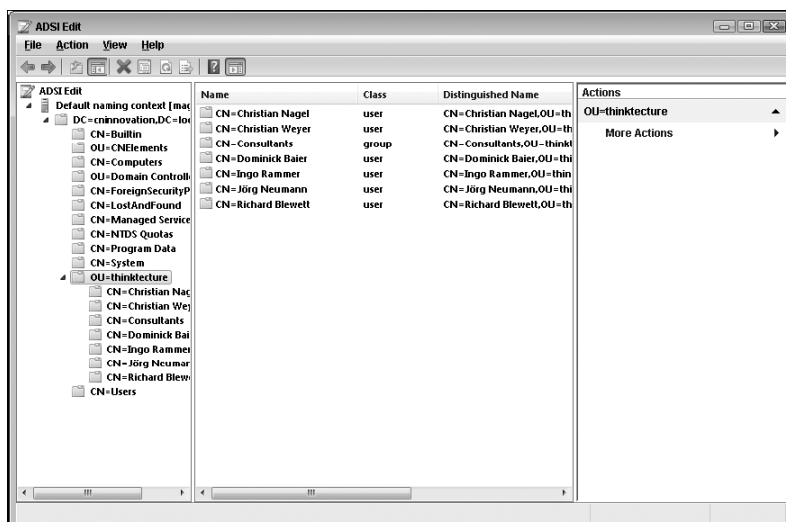


图 52-7

52.3 Active Directory 编程

要为 Active Director 开发程序, 可以使用 `System.DirectoryServices` 或 `System.Directory Services.Protocols` 名称空间中的类。在 `System.DirectoryServices` 名称空间中, 可以找到包装活动目录服务接口(Active Directory Service Interfaces, ADSI) COM 对象的类, 它们可以访问 Active Directory。

ADSI 是目录服务的一个编程接口。ADSI 定义一些由 ADSI 提供程序实现的 COM 接口。客户端可以在相同编程接口中使用不同目录服务。`System.DirectoryServices` 名称空间中的 .NET Framework 类可以使用 ADSI。

在图 52-8 中, 有一些实现 COM 接口(如 IADs 和 IUnknown)的 ADSI 提供程序(LDAP、IIS 和 NDS)。`System.DirectoryServices` 程序集使用 ADSI 提供程序。

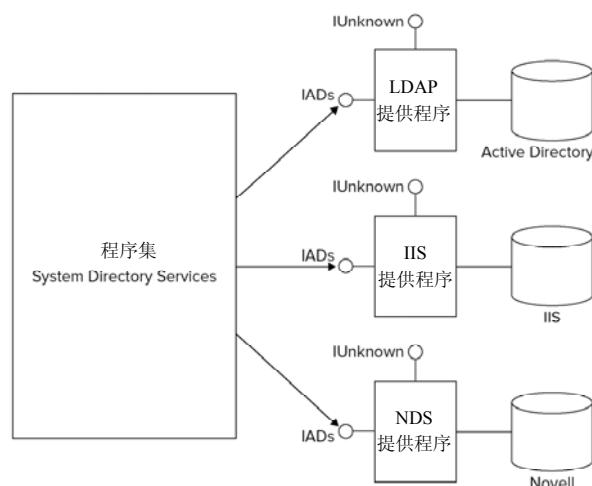


图 52-8

System.DirectoryServices.Protocols 名称空间中的类使用 Windows 的 DSML(Directory Services Markup Language, 目录服务标记语言)服务。OASIS 组(<http://www.oasis-open.org/committees/dsml>) 利用 DSML 定义了标准化的 Web 服务接口。

要使用 System.DirectoryServices 名称空间中的类, 必须引用 System.DirectoryServices 程序集。使用这个程序集中的类可以查询对象、查看和更新属性、搜索对象, 以及把对象移动到其他容器对象中。在本节后面的代码段中, 将使用一个简单的 C#控制台应用程序, 来说明 System.DirectoryServices 名称空间中的类的功能。

本节将介绍:

- System.DirectoryServices 名称空间中的类
- 连接 Active Directory 的处理方式: 绑定
- 获取目录项, 创建新对象, 并更新已有项
- 搜索 Active Directory

52.3.1 System.DirectoryServices 名称空间中的类

表 52-1 列出了 System.DirectoryServices 名称空间中的主要类。

表 52-1	
类	说 明
DirectoryEntry	这个类是 System.DirectoryServices 名称空间中的主类。这个类的对象表示 Active Directory 存储器中的一个对象。使用这个类可以绑定对象, 查看和更新属性。对象的属性都显示在 PropertyCollection 中。PropertyCollection 中的每一项都有一个 PropertyValueCollection
DirectoryEntries	DirectoryEntries 是 DirectoryEntry 对象的一个集合。DirectoryEntry 对象的 Children 属性返回 DirectoryEntries 集合中的一个对象列表
DirectorySearcher	这个类主要用于用指定的属性搜索对象。要定义该搜索, 可以使用 SortOption 类和 SearchScope、SortDirection 和 ReferralChasingOption 枚举。搜索的结果是一个 SearchResult 或 SearchResultCollection。也可以得到 ResultProperty Collection 和 ResultPropertyValueCollection 对象

52.3.2 绑定到 Directory Services

要获得 Active Directory 中一个对象的值, 必须连接到 Active Directory 服务。这个连接进程称为绑定。绑定路径如下:

LDAP://dc01.thinktecture.com/OU=Development, DC=Thinktecture, DC=Com

在绑定进程中, 可以指定如下项:

- 协议(protocol)指定要使用的提供程序
- 域控制器的服务器名(server name)
- 服务器进程的端口号(port number)
- 对象的可分辨名称(distinguished name), 这标识要访问的对象
- 如果允许访问 Active Directory 的用户不是当前登录的账户, 则提供用户名和密码

- 如果需要加密，应指定 authentication 类型。

下面详细介绍这些内容。

1. 协议

绑定路径(即协议)的第一部分指定 ADSI 提供程序。该提供程序作为一个 COM 服务器实现；对于标记，progID 可在注册表的 HKEY_CLASSES_ROOT 下。Windows 7 附带的提供程序如表 52-2 所示。

表 52-2

提 供 程 序	说 明
LDAP	LDAP 服务器，如 Exchange 目录和 Windows 2000 及以后版本的 Active Directory 服务器
GC	GC 用于访问 Active Directory 中的全局目录。它也可以用于快速查询
IIS	使用 IIS 的 ADSI 提供程序，可以在 IIS 目录中创建和管理新网站
NDS	这个 progID 用于和 Novell Directory Services 通信

2. 服务器名

在绑定路径中，服务器名在协议的后面。如果用户登录到 Active Directory 域上，服务器名就是可选的。如果不提供服务器名，就会发生无服务器绑定操作，此时 Windows Server 2008 会在域中查找与用户绑定过程相关的“最好的”域控制器。如果站点中没有服务器，就使用查找到的第一个域控制器。

无服务器绑定如下所示：

```
LDAP://OU=Sales, DC=Thinktecture, DC=Local
```

3. 端口号

在服务器名的后面，可以指定服务器进程的端口号，其语法是 xxx。LDAP 服务器的默认端口号是 389：

```
LDAP://dc01.sentinel.net:389
```

Exchange 服务器使用的端口号同 LDAP 服务器的端口号。如果在同一个系统上安装 Exchange 服务器，例如，用作 Active Directory 的域控制器，就可以配置另一个端口。

4. 可分辨名称

在路径中指定的第 4 部分是可分辨名称(Distinguished Name, DN)。DN 是一个唯一的名称，它标识要访问的对象。在 Active Directory 中，可以使用基于 X.500 的 LDAP 语法，指定对象的名称。

这是有一个可分辨名称：

```
CN=Christian Nagel, OU=Consultants, DC= Thinktecture, DC=local
```

这个 DN 指定 thinktecture.local 域中 thinktecture 域组件(Domain Component, DC)的 Consultants 组织单元(Organizational Unit, OU)的公共名称(Common Name, CN) Christian Nagel。最右边指定的

部分是域的根对象。该名称必须符合对象树中的层次结构。

可分辨名称的字符串表示的 LDAP 规范在 RFC 2253(<http://www.ietf.org/rfc/rfc2253.txt>)上。

(1) 相对可分辨名称

相对可分辨名称(RDN)用于引用容器对象中的对象。使用 RDN 时,不需要指定 OU 和 DC,因为一个公共名称就足够了。CN=Christian Nagel 就是组织单元中的一个相对可分辨名称。如果已经引用一个容器对象,并且要访问其子对象,就可以使用相对可分辨名称。

(2) 默认的命名上下文

如果在路径中没有指定可分辨名称,绑定进程就会使用默认的命名上下文。使用 rootDSE 可以读取默认命名上下文。LDAP 3.0 把 rootDSE 定义为目录服务器中目录树的根。例如

```
LDAP://rootDSE
```

或

```
LDAP://servername/rootDSE
```

通过枚举 rootDSE 的所有属性,将获得没有指定名称时使用的 defaultNamingContext 信息。schemaNamingContext 和 configurationNamingContext 指定用于访问 Active Directory 存储器中架构和配置所需要的名称。

通过下面的代码可获得 rootDSE 的所有属性:

```
try
{
    using (var de = new DirectoryEntry())
    {
        de.Path = "LDAP://magellan/rootDSE";
        de.Username = @"cninnovation\christian";
        de.Password = "Pa$$w0rd";

        PropertyCollection props = de.Properties;
        foreach (string prop in props.PropertyNames)
        {
            PropertyValueCollection values = props[prop];
            foreach (string val in values)
            {
                Console.WriteLine("{0}: ", prop);
                Console.WriteLine(val);
            }
        }
    }
}
catch (COMException ex)
{
    Console.WriteLine(ex.Message);
}
```

代码段 DirectoryServicesSamples/Program.cs



要在自己的计算机上运行这段代码,必须更改要访问的对象的`Path`属性,使之包含服务器名。

这个程序显示默认的命名上下文(defaultNamingContextDC=explorer、DC=local), 用于访问架构的上下文(CN=Schema、CN=Configuration、DC=cninnovation、DC=local)和上下文的命名环境(CN=Configuration、DC=cninnovation、DC=local), 如下所示。

```
currentTime: 20090925131508.0Z
subschemaSubentry: CN=Aggregate,CN=Schema,CN=Configuration,DC=cninnovation,
DC=local
dsServiceName: CN=NTDS Settings,CN=MAGELLAN,CN=Servers,
CN=Default-First-Site-Name,CN= Sites,CN=Configuration,DC=cninnovation,DC=local
namingContexts: DC=cninnovation,DC=local
namingContexts: CN=Configuration,DC=cninnovation,DC=local
namingContexts: CN=Schema,CN=Configuration,DC=cninnovation,DC=local
namingContexts: DC=DomainDnsZones,DC=cninnovation,DC=local
namingContexts: DC=ForestDnsZones,DC=cninnovation,DC=local
defaultNamingContext: DC=cninnovation,DC=local
schemaNamingContext: CN=Schema,CN=Configuration,DC=cninnovation,DC=local
configurationNamingContext: CN=Configuration,DC=cninnovation,DC=local
rootDomainNamingContext: DC=cninnovation,DC=local
supportedControl: 1.2.840.113556.1.4.319
supportedControl: 1.2.840.113556.1.4.801
```

(3) 对象标识符

每个对象都有一个全局唯一的标识符 GUID。GUID 是一个唯一的 128 位数字, 您可能已经从 COM 开发中了解了它。使用 GUID 可以绑定一个对象。这样, 不管对象是否移动到另一个容器中, 也可以得到同一个对象。GUID 在创建对象时生成, 且总是保持不变。

使用 DirectoryEntry.NativeGuid 可以得到 GUID 的字符串表示。这个字符串表示就可以用于绑定对象。

下面的示例显示一个无服务器绑定的路径名, 它绑定到 GUID 表示的一个特定对象上:

```
LDAP://<GUID=14abbd652aae1a47abc60782dcfc78ea>
```

5. 用户名

如果必须使用另一个用户名访问目录(也许这个用户没有访问 Active Directory 所必需的权限), 此时必须为绑定进程指定显式用户证书(user credential)。Active Directory 提供许多方式来指定用户名。

(1) Downlevel 登录

使用 downlevel 登录, 用户名可以用 Windows 2000 以前的域名指定:

```
domain\username
```

(2) 可分辨名称

也可以用 user 对象的可分辨名称来指定用户, 例如:

```
CN=Administrator, CN=Users, DC=thinktexture, DC=local
```

(3) 用户主体名 (UPN)

对象的 UPN 用 userPrincipalName 属性定义。系统管理员可以在 Active Directory Users and Computers 工具中 User 属性的 Account 选项卡上, 用登录信息指定 UPN, 注意, 这不是用户的电子邮件地址。

这些信息也唯一地标识了用户，可以用于登录：

```
Nagel@ thinktecture.local
```

6. 身份验证

对于安全加密身份验证，也可以指定身份验证(authentication)类型。身份验证可以用 `DirectoryEntry` 类的 `AuthenticationType` 属性设置。可以指定其值为 `AuthenticationTypes` 枚举中的一个值。因为枚举用 [Flags] 特性标记，所以可以指定多个值。发送的数据进行加密的可能值有 `ReadonlyServer` 和 `Secure`，`ReadonlyServer` 指定只需要读取访问；`Secure` 表示安全身份验证。

7. 用 DirectoryEntry 类绑定

`System.DirectoryServices.DirectoryEntry` 类可以用于指定所有绑定信息。可以使用默认的构造函数，并用 `Path`、`Username`、`Password` 和 `AuthenticationType` 属性定义绑定信息，或者把这些信息传递给构造函数：

```
DirectoryEntry de = new DirectoryEntry();
var de = new DirectoryEntry();
de.Path = "LDAP://platinum/DC=thinktecture, DC=local";
de.Username = "Christian.Nagel@thinktecture.local";
de.Password = "password";

// use the current user credentials
var de2 = new DirectoryEntry("LDAP://DC=thinktecture, DC=local");
```

即使构造 `DirectoryEntry` 对象成功，也并不意味着绑定成功。在第一次读取属性时进行绑定，可以避免不必要的网络流量。对象是否存在，或者指定的用户证书是否正确，都可以在第一次访问该对象时确定。

52.3.3 获取目录项

既然知道了如何指定 Active Directory 中对象的绑定特性，就可以转向读取对象的特性。在下面的示例中，读取用户对象的属性。

`DirectoryEntry` 类的一些属性可以提供对象的信息，即 `Name`、`Guid` 和 `SchemaClassName` 属性。第一次访问 `DirectoryEntry` 对象的属性时，会执行绑定操作，并填充基础 ADSI 对象的缓存。后面将详细讨论这些。其他属性可以从缓存中读取，同一对象的数据不需要通过与服务器的通信来获得。

在本例中，用 `thinktecture` 组织单元中的公共名称 `Christian Nagel` 访问 `user` 对象：



可从
wrox.com
下载源代码

```
using (var de = new DirectoryEntry())
{
    de.Path = "LDAP://magellan/CN=Christian Nagel, " +
              "OU=thinktecture, DC=cninnovation, DC=local";
    Console.WriteLine("Name: {0}", de.Name);
    Console.WriteLine("GUID: {0}", de.Guid);
    Console.WriteLine("Type: {0}", de.SchemaClassName);
    Console.WriteLine();

    //...
}
```

代码段 `DirectoryServicesSamples/Program.cs`

Active Directory 对象包含许多信息, 可用信息取决于对象的类型。Properties 属性返回一个 PropertyCollection。每个属性本身就是一个集合, 因为一个属性可以有多个值; 例如, user 对象可以有多个电话号码。在本例中, 用一个内层 foreach 循环查看这些值。从 properties[name]返回的集合是一个 object 数组。特性值可以是字符串、数字或其他类型。使用 ToString()方法就可以显示这些值:

```
Console.WriteLine("Properties: ");
PropertyCollection properties = de.Properties;
foreach (string name in properties.PropertyNames)
{
    foreach (object o in properties[name])
    {
        Console.WriteLine("{0}: {1}", name, o.ToString());
    }
}
```

在最终输出中, 可以查看 user 对象指定的所有特性。一些属性(如 otherTelephone)有多个值, 通过该属性, 可以定义许多电话号码。一些属性值只显示 System.__ComObject 对象的类型, 例如, lastLogoff、lastLogon 和 nTSecurityDescriptor。要得到这些特性的值, 必须直接使用 System.DirectoryServices 名称空间的类中的 ADSI COM 接口。

```
Name: CN=Christian Nagel
GUID: 0238fd5c-7e67-48bc-985f-c2f1ccf0f86c
Type: user

Properties:
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
cn: Christian Nagel
sn: Nagel
givenName: Christian
distinguishedName: CN=Christian Nagel,OU=thinktecture,DC=cninnovation,DC=local
instanceType: 4
whenCreated: 9/25/2009 12:42:05 PM
whenChanged: 9/25/2009 12:42:05 PM
displayName: Christian Nagel
uSNCreated: System.__ComObject
uSNChanged: System.__ComObject
name: Christian Nagel
objectGUID: System.Byte[]
userAccountControl: 66048
badPwdCount: 0
codePage: 0
countryCode: 0
badPasswordTime: System.__ComObject
lastLogoff: System.__ComObject
lastLogon: System.__ComObject
pwdLastSet: System.__ComObject
primaryGroupID: 513
objectSid: System.Byte[]
accountExpires: System.__ComObject
logonCount: 0
```

```
sAMAccountName: christian.nagel
sAMAccountType: 805306368
userPrincipalName: christian.nagel@cninnovation.local
objectCategory: CN=Person,CN=Schema,CN=Configuration,DC=cninnovation,DC=local
dSCorePropagationData: 1/1/1601 12:00:00 AM
nTSecurityDescriptor: System.__ComObject
```

使用 `DirectoryEntry.Properties` 可以访问所有属性。如果已知属性名，就可以直接访问其值：

```
foreach (string homePage in de.Properties["wWWHomePage"])
    Console.WriteLine("Home page: " + homePage);
```

52.3.4 对象集合

对象分层存储在 Active Directory 中。容器对象包含子对象。使用 `DirectoryEntry` 类的 `Children` 属性可以枚举容器中的子对象。另一方面，使用 `Parent` 属性可以得到对象的容器。

因为 `user` 对象没有子对象，所以下面的示例使用一个组织单元。非容器对象用 `Children` 属性返回一个空集合。下面从 `explore.local` 域 `thinktecture` 组织单元中获得所有 `user` 对象。`Children` 属性返回一个 `DirectoryEntries` 集合，其中包含 `DirectoryEntry` 对象。遍历所有 `DirectoryEntry` 对象，显示子对象的名称。



```
using (var de = new DirectoryEntry())
{
    de.Path = "LDAP://magellan/OU=thinktecture, DC=cninnovation, DC=local";

    Console.WriteLine("Children of {0}", de.Name);
    foreach (DirectoryEntry obj in de.Children)
    {
        Console.WriteLine(obj.Name);
    }
}
```

代码段 `DirectoryServicesSamples/Program.cs`

运行程序，会显示对象的公共名称：

```
Children of OU=thinktecture
OU=Admin
CN=Buddhike de Silva
CN=Christian Nagel
CN=Christian Weyer
CN=Consultants
CN=demos
CN=Dominick Baier
CN=Ingo Rammer
CN=Neno Loye
```

本例显示了组织单元中的所有对象：`users`、`contacts`、`printers`、`shares` 和其他对象。如果只需要查看某些对象类型，就可以使用 `DirectoryEntries` 类的 `SchemaFilter` 属性。`SchemaFilter` 属性返回一个 `SchemaNameCollection`。通过这个 `SchemaNameCollection`，就可以使用 `Add()` 方法定义要查看的对象类型。在本例中，因为只需要查看 `user` 对象，所以把 `user` 添加到这个集合中：


```

using (var de = new DirectoryEntry())
{
    de.Path = "LDAP://magellan/OU=thinktecture, DC=cninnovation, DC=local";

    Console.WriteLine("Children of {0}", de.Name);
    de.Children.SchemaFilter.Add("user");
    foreach (DirectoryEntry obj in de.Children)
    {
        Console.WriteLine(obj.Name);
    }
}

```

结果，只显示组织单元中的 **user** 对象，如下所示。

```

Children of OU=thinktecture
CN=Christian Nagel
CN=Christian Weyer
CN=Dominick Baier
CN=Ingo Rammer
CN=Jörg Neumann
CN=Richard Blewett

```

52.3.5 缓存

为了减少网络流量，ADSI 使用缓存来存储对象属性。如前所述，在创建 **DirectoryEntry** 对象时不访问服务器。只要从目录存储器中读取第一个属性，所有属性都会写入缓存中，这样，在访问下一个属性时，就不需要往返服务器。

给对象写入任何更改内容时，只会改变已缓存的对象。设置属性不会产生网络流量。必须使用 **DirectoryEntry.CommitChanges()** 方法刷新缓存，把任何已更改的数据传输到服务器。要从目录存储器中获取新写入的数据，可以使用 **DirectoryEntry.RefreshCache()** 读取属性。当然，如果没有调用 **CommitChanges()** 和 **RefreshCache()** 方法更改一些属性，那么所有更改都会丢失，因为我们将再次使用 **RefreshCache()** 方法读取目录服务中的值。

把 **DirectoryEntry.UsePropertyCache** 属性设置为 **false**，就可以关闭这个属性缓存。但除非正在调试代码，否则最好不要关闭缓存，因为这会与服务器间产生额外的往返。

52.3.6 创建新对象

创建新 Active Directory 对象时，如 **users**、**computers**、**printers** 和 **contacts** 等，可以使用 **DirectoryEntries** 类以编程方式完成创建工作。

要给目录添加新对象，首先必须绑定一个容器对象，如组织单元，可以在其中插入新对象。不能使用不包含其他对象的对象。下面的示例使用一个容器对象，其可分辨名称为 **CN=Users**，**DC=thinktecture**，**DC=local**：



可从
wrox.com
下载源代码

```

var de = new DirectoryEntry();
de.Path = "LDAP://magellan/CN=Users, DC=cninnovation, DC=local";

```

代码段 [DirectoryServicesSamples/Program.cs](#)

使用 `DirectoryEntry` 类的 `Children` 属性，可以得到 `DirectoryEntries` 对象：

```
DirectoryEntries users = de.Children;
```

`DirectoryEntries` 类提供的方法可以添加、删除和查找集合中的对象。下面创建一个新的 `user` 对象。使用 `Add()` 方法时，需要一个对象名和一个类型名。使用 `ADSI Edit` 可以直接获得类型名：

```
DirectoryEntry user = users.Add("CN=John Doe", "user");
```

对象现在有默认属性值。要指定特定属性值，可以使用 `Properties` 属性的 `Add()` 方法添加属性。当然，所有属性都必须存在于 `user` 对象的架构中。如果指定的属性不存在，就得到一个 `COMException` 异常：“指定的目录服务属性或值不存在”：

```
user.Properties["company"].Add("Some Company");
user.Properties["department"].Add("Sales");
user.Properties["employeeID"].Add("4711");
user.Properties["samAccountName"].Add("JDoe");
user.Properties["userPrincipalName"].Add("JDoe@explorer.local");
user.Properties["givenName"].Add("John");
user.Properties["sn"].Add("Doe");
user.Properties["userPassword"].Add("someSecret");
```

最后，为了把数据写入 `Active Directory` 中，必须刷新缓存：

```
user.CommitChanges();
```

52.3.7 更新目录项

`Active Directory` 服务中对象的更新和读取一样简单。可以在读取对象后更改它们的值。要删除一个属性的所有值，可以调用 `PropertyValueCollection.Clear()` 方法。使用 `Add()`，可以把新值添加到属性中。`Remove()` 和 `RemoveAt()` 方法可以从属性集合中删除指定的值。

要更改一个值，只需把这个值设置为指定的值。下面的示例通过 `PropertyValueCollection` 类的一个索引器把移动电话号码设置为一个新值。使用该索引器只能改变已存在的值。因此，应总是使用 `DirectoryEntry.Properties.Contains()` 方法来确定该特性是否可用。

```
using (var de = new DirectoryEntry())
{
    de.Path = "LDAP://magellan/CN=Christian Nagel, " +
              "OU=thinktexture, DC=cninnovation, DC=local";

    if (de.Properties.Contains("mobile"))
    {
        de.Properties["mobile"][0] = "+43(664)3434343434";
    }
    else
    {
        de.Properties["mobile"].Add("+43(664)3434343434");
    }

    de.CommitChanges();
}
```

在本例的 `else` 部分，如果移动电话号码不存在新属性，就用 `PropertyValue Collection.Add()` 方法添加它。如果对已有的属性使用 `Add()` 方法，那么最终结果将取决于属性的类型(单值属性或多值属性)。对已有的单值属性使用 `Add()` 方法，会得到一个 `COMException` 异常：“违反了一个约束。”对已有的多值属性使用 `Add()` 方法，则会成功地把另一个值添加到属性中。

因为把 `user` 对象的属性 `mobile` 定义为单值属性，所以不能添加其他移动电话号码。但是用户可以有多个移动电话号码。对于多个移动电话号码，`otherMobile` 属性可用。因为它是一个多值属性，允许设置多个移动电话号码，所以可以多次调用 `Add()` 方法。注意，对多值属性要检查其唯一性。如果把第二个电话号码添加到同一个 `user` 对象中，那么也会得到一个 `COMException` 异常：“指定的目录服务属性或值已经存在。”



在创建或更新目录对象后，应调用 `DirectoryEntry.CommitChanges()` 方法。否则，只能更新缓存中的信息，更改的信息不会发送到目录服务中。

52.3.8 访问本地 ADSI 对象

调用预定义的 ADSI 接口的方法常常比搜索对象的属性名容易得多。一些 ADSI 对象还支持不能从 `DirectoryEntry` 类中直接使用的方法。例如，`IADsServiceOperations` 接口有启动和停止 Windows 服务的方法。Windows 服务的详细内容请参见第 25 章。

如前所述，`System.DirectoryServices` 名称空间的类使用基础 ADSI COM 对象。`DirectoryEntry` 类支持直接使用 `Invoke()` 方法调用基础对象的主调方法。

`Invoke()` 方法的第一个参数是应在 ADSI 对象中调用的方法名，第二个参数的 `params` 关键字允许把数量可变的其他参数传递给 ADSI 方法：

```
public object Invoke(string methodName, params object[] args);
```

ADSI 文档介绍了可以用 `Invoke()` 方法调用的方法。域中的每个对象都支持 `IADs` 接口的方法。前面创建的 `user` 对象也支持 `IADsUser` 接口的方法。

在下面的代码示例中，使用 `IADsUser.SetPassword()` 方法改变前面创建的 `user` 对象的密码：

```
using (var de = new DirectoryEntry())
{
    de.Path = "LDAP://magellan/CN=John Doe, CN=Users, DC=cninnovation, DC=local";

    de.Invoke("SetPassword", "anotherSecret");
    de.CommitChanges();
}
```

如果不使用 `Invoke()` 方法，那么还可以直接使用基础 ADSI 对象。要使用这些对象，必须选择 `Project | Add Reference` 命令添加对 `Active DS Type Library` 的引用，如图 52-9 所示。这会创建一个包装器类，在该类中可以访问 `ActiveDs` 名称空间中的这些对象。

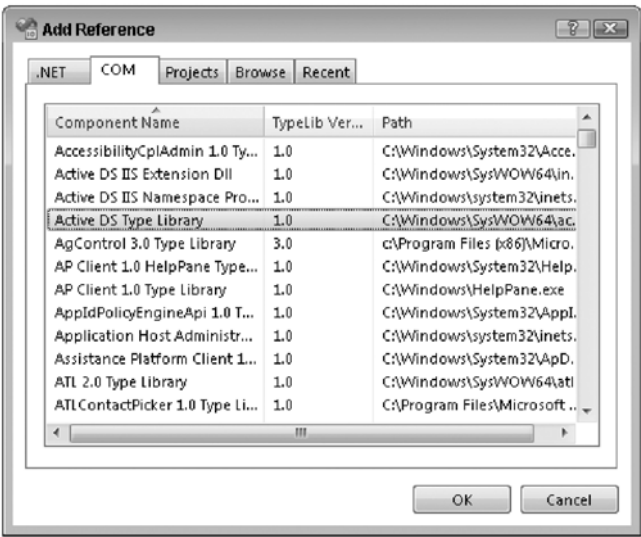


图 52-9

本地对象可以使用 DirectoryEntry 类的 NativeObject 属性访问。在下面的示例中，因为 de 对象是一个 user 对象，所以可以把它强制转换为 ActiveDs.IADsUser 类型。SetPassword()方法是在 IADsUser 接口中归档的方法，所以可以直接调用它，而不是使用 Invoke()方法调用它。把 IADsUser 接口的 AccountDisabled 属性设置为 false，就可以启用账户。与前面的示例一样，调用 CommitChanges() 方法和 DirectoryEntry 对象，把更改的内容写入目录服务中：

```
ActiveDs.IADsUser user = (ActiveDs.IADsUser)de.NativeObject;  
user.SetPassword("someSecret");  
user.AccountDisabled = false;  
de.CommitChanges();
```



从.NET 3.5 开始，不需要在.NET 类 DirectoryEntry 的后面调用本地对象。可以使用 System.DirectoryServices.AccountManagement 名称空间中的类管理用户。这个名称空间中的类参见本章后面的内容。

52.3.9 在 Active Directory 中搜索

因为 Active Directory 是一个数据存储器，它为大多数读取访问进行了优化，所以一般用来搜索一些值。要在 Active Directory 中搜索，可以使用.NET Framework 中的 DirectorySearcher 类。



DirectorySearcher 类只能和 LDAP 提供程序一起使用；它不能和 NDS 或 IIS 等提供程序一起使用。

在 DirectorySearcher 类的构造函数中，可以定义搜索的 4 个重要部分：搜索开始的根目录、筛选、应加载的属性和搜索的范围。也可以使用默认构造函数，并用属性定义搜索选项。

1. SearchRoot

搜索根目录指定搜索应从什么地方开始。SearchRoot 的默认值是当前使用的域的根。SearchRoot 用 DirectoryEntry 对象的 Path 指定。

2. 筛选器

筛选器定义希望获得的值。筛选器是一个必须放在圆括号中的字符串。

表达式中可以使用关系运算符，如 <=、=、>=。(objectClass=contact)搜索所有为 contact 类型的对象；(lastName>=Nagel)按字母顺序搜索 lastName 属性等于或大于 Nagel 的所有对象。

表达式可以和前缀运算符“&”和“|”组合使用。例如，(&(objectClass=user)(description=Auth*))搜索类型为 user，其 description 属性以字符串 Auth 开头的所有对象。因为“&”和“|”运算符在表达式的开头，所以可以把两个以上的表达式用一个前缀运算符组合在一起。

因为默认筛选器是(objectClass=*)，所以所有对象有效。



筛选器语法在 RFC 2254 “LDAP 搜索筛选器的字符串表示”中定义。这个 RFC 在 <http://www.ietf.org/rfc/rfc2254.txt> 中。

3. PropertiesToLoad

使用 PropertiesToLoad，可以定义需要的所有属性的一个 StringCollection。对象可以有許多属性，其中大多数对于搜索请求都不太重要。我们定义了应加载到缓存中的属性。如果没有指定属性返回的，默认属性就是对象的 Path 和 Name 属性。

4. SearchScope

SearchScope 是一个枚举，它定义搜索应扩展的深度：

- 因为 SearchScope.Base 只搜索开始搜索的对象中的特性，所以至多可以得到一个对象。
- SearchScope.OneLevel 表示在基对象的子集合中继续搜索。不搜索基对象本身。
- SearchScope.Subtree 定义在整个树中向下搜索。

SearchScope 属性的默认值是 SearchScope.Subtree。

5. 搜索的限制

在目录服务中搜索特定的对象可以跨几个域进行。要限制搜索对象的个数或搜索时间，可以定义其他几个属性(如表 52-3 所示)。

表 52-3


属 性	说 明
ClientTimeout	客户端等待服务器返回结果的最长时间。如果服务器没有响应，就不返回记录
PageSize	使用 paged search，服务器会返回用 PageSize 定义的许多对象，而不是所有对象。这会减少客户端获得第一个应答的时间和需要的内存。服务器把一个 cookie 发送给客户端，客户端再把下一个搜索请求发送回服务器，这样搜索就可以在上一次结束的地方继续进行

(续表)

属 性	说 明
ServerPageTimeLimit	对于 paged search, 这个值定义了一个搜索时间范围, 在该时间范围内应继续返回用 PageSize 值定义的许多对象。如果这段时间在 PageSize 值之前到达, 就会把此时查找到的对象返回给客户端。默认值为 - 1, 表示时间为无限长
SizeLimit	定义搜索返回的最大对象个数。如果把它设置为比服务器定义的值(1000)还大, 就使用服务器定义的限制
ServerTimeLimit	定义服务器搜索对象的最长时间。过了这段时间后, 就会把此时查找到的所有对象返回给客户端。默认值是 120 秒, 不能把搜索时间设置为较高的值
ReferralChasing	搜索可以跨几个域进行。如果用 SearchRoot 指定的根目录是一个父域或没有指定根目录, 就会继续在子域中搜索。使用这个属性可以指定是否应在不同的服务器上继续搜索。 ReferralChasingOption.None 表示不在不同的服务器上继续搜索。 ReferralChasingOption.Subordinate 指定继续在子域中搜索。当搜索从 DC=Wrox、DC=COM 开始时, 服务器可以返回一个结果集, 并引用 DC=France、DC=Wrox、DC=COM。客户端可以继续在于域中搜索。 ReferralChasingOption.External 表示服务器可以使客户机搜索不在子域中的另一个独立服务器, 这是默认选项。 ReferralChasingOption.All 表示返回外部引用和从属引用
Tombstone	如果将属性 Tombstone 设置为 true, 就返回所有匹配搜索的已删除对象
VirtualListView	如果搜索结果比较多, 就可以使用 VirtualListView 属性定义一个应从搜索中返回的子集。这个子集用 DirectoryVirtualListView 类定义

在搜索示例中, 要搜索 thinktecture 组织单元中 description 属性值为 Author 的所有 user 对象。

首先, 绑定 thinktecture 组织单元, 从该组织单元中开始搜索。创建一个 DirectorySearcher 对象, 在其中设置 SearchRoot。把筛选器定义为(&(objectClass=user)(description=Auth*)), 这样就可以搜索所有类型为 user、description 属性以 Auth 开头的对象。搜索的范围应在一个子树中, 这样在 thinktecture 的子组织单元中搜索:



可从
wrox.com
下载源代码

```
using (var de = new DirectoryEntry("LDAP://OU=thinktecture, DC=cninnovation, DC=local"))
using (var searcher = new DirectorySearcher())
{
    searcher.SearchRoot = de;
    searcher.Filter = "( &(objectClass=user)(description=Auth*))";
    searcher.SearchScope = SearchScope.Subtree;
```

代码段 DirectoryServicesSamples/Program.cs

要在搜索结果中包含的属性有 name、description、givenName 和 wWWHomePage。

```
searcher.PropertiesToLoad.Add("name");
searcher.PropertiesToLoad.Add("description");
searcher.PropertiesToLoad.Add("givenName");
searcher.PropertiesToLoad.Add("wWWHomePage");
```

下面准备开始搜索。还应对结果进行排序。DirectorySearcher 类有一个 Sort 属性，它可以设置一个 SortOption。SortOption 类的构造函数的第一个参数定义用于排序的属性，第二个参数定义排序的方向。SortDirection 枚举的值有 Ascending 和 Descending。

要开始搜索，可以使用 FindOne()方法查找第一个对象，或者使用 FindAll()方法。FindOne()方法返回一个简单的 SearchResult，而 FindAll()方法返回一个 SearchResultCollection。因为要返回所有作者，所以应使用 FindAll()方法：

```
searcher.Sort = new SortOption("givenName", SortDirection.Ascending);
SearchResultCollection results = searcher.FindAll();
```

使用一个 foreach 循环，可以访问 SearchResultCollection 中的每个 SearchResult。SearchResult 表示搜索缓存中的一个对象。Properties 属性返回一个 ResultPropertyCollection，其中使用属性名和索引器可以访问该集合中的所有属性：

```
SearchResultCollection results = searcher.FindAll();

foreach (SearchResult result in results)
{
    ResultPropertyCollection props = result.Properties;
    foreach (string propName in props.PropertyNames)
    {
        Console.WriteLine("{0}: ", propName);
        Console.WriteLine(props[propName][0]);
    }
    Console.WriteLine();
}
```

也可以在搜索之后获得完整的对象：SearchResult 有一个 GetDirectoryEntry()方法，它返回查找到的对象的相应 DirectoryEntry。

最终的输出应显示所有 thinktecture 对应的列表的开始，以及已选择的属性。

```
givenname: Christian
adspath: LDAP://magellan/CN=Christian Weyer,OU=thinktecture,DC=cninnovation,
DC=local
description: Author
name: Christian Weyer

givenname: Christian
adspath: LDAP://magellan/CN=Christian Nagel,OU=thinktecture,DC=cninnovation,
DC=local
description: Author
name: Christian Nagel

givenname: Dominick
adspath: LDAP://magellan/CN=Dominick Baier,OU=thinktecture,DC=cninnovation,
DC=local
description: Author
name: Dominick Baier

givenname: Ingo
adspath: LDAP://magellan/CN=Ingo Rammer,OU=thinktecture,DC=cninnovation,
DC=local
```

```
description: Author
name: Ingo Rammer

givenname: Jörg
adspath: LDAP://magellan/CN=Jörg Neumann,OU=thinktecture,DC=cninnovation,
DC=local
description: Author
name: Jörg Neumann
```

52.4 搜索用户对象

本节要创建一个 WPF 应用程序 UserSearch。这个应用程序非常灵活，因为可以输入特定域控制器、用户名和密码，访问 Active Directory 或者使用正在运行的进程的用户。在这个应用程序中，我们将访问 Active Directory 服务的架构，获得 user 对象的属性。该用户可以输入一个筛选器字符串，搜索域中的所有 user 对象。还可以设置应显示的 user 对象的属性。

52.4.1 用户界面

用户界面显示一些已编号的步骤，说明如何使用该应用程序(如图 52-10 所示)。

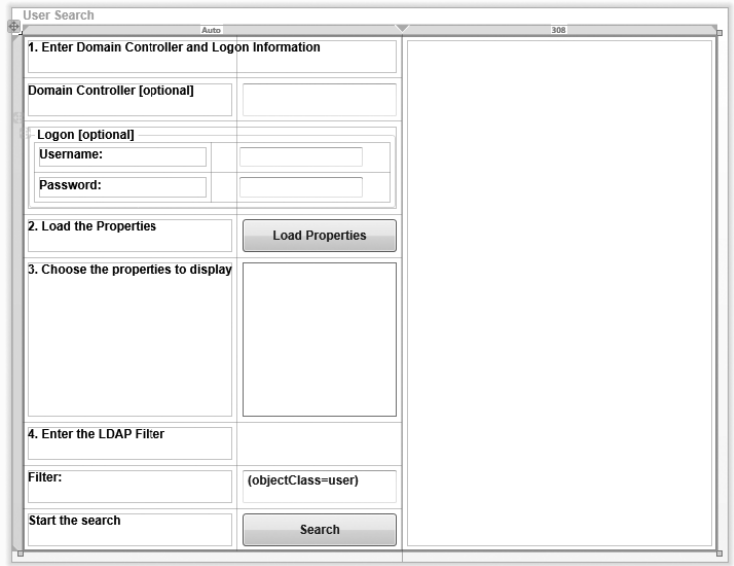


图 52-10

1. 输入用户名、密码和域控制器。所有这些信息都是可选的。如果没有输入域控制器，就使用无服务器绑定进行连接。如果没有输入用户名，就使用当前用户的安全上下文。
2. 使用一个按钮，就可以把 user 对象的所有属性名动态地加载到 listBoxProperties 列表框中。
3. 选择要显示的属性。把列表框的 SelectionMode 设置为 MultiSimple。
4. 可以输入限制搜索的筛选器，在该对话框中设置的默认值搜索所有 user 对象：(objectClass=user)。
5. 现在开始搜索。

52.4.2 获取架构命名上下文

这个应用程序只有两个处理程序方法：按钮的第一个方法加载属性，第二个方法则在域中开始搜索操作。首先，从架构中动态地读取 `user` 类的属性，在用户界面中显示它。

在 `buttonLoadProperties_Click()` 方法中，使用 `SetLogonInformation()` 方法从对话框中读取用户名、密码和主机名，并把它们存储在类的成员中。接着，`SetNamingContext()` 方法设置架构的 LDAP 名称和默认上下文的 LDAP 名称。这个架构的 LDAP 名称用于 `SetUserProperties()` 调用中，设置列表框中的属性。



```
private void OnLoadProperties(object sender, RoutedEventArgs e)
{
    try
    {
        SetLogonInformation();

        SetNamingContext();

        SetUserProperties(schemaNamingContext);
    }
    catch (Exception ex)
    {
        MessageBox.Show(String.Format("check your input! {0}", ex.Message));
    }
}
```

代码段 UserSearch/MainWindow.xaml.cs

在辅助方法 `SetNamingContext()` 中，使用目录树的根目录来获得服务器的属性。这里只考虑两个属性的值：`schemaNamingContext` 和 `defaultNamingContext`。

```
private void SetNamingContext()
{
    using (var de = new DirectoryEntry())
    {
        string path = "LDAP://" + hostname + "rootDSE";
        de.Username = username;
        de.Password = password;
        de.Path = path;
        schemaNamingContext = de.Properties["schemaNamingContext"][0].ToString();
        defaultNamingContext = de.Properties["defaultNamingContext"][0].ToString();
    }
}
```

52.4.3 获取 User 类的属性名

使用 LDAP 名称可以访问架构。使用它可以访问目录，并读取属性。我们不仅要介绍 `user` 类的属性，还将介绍 `user` 类的基类：`Organizational-Person`、`Person` 和 `Top`。在这个程序中，基类的名称是硬编码的。还可以使用 `subClassOf` 属性动态地读取基类。

`GetSchemaProperties()` 方法返回 `IEnumerable<string>`，其中包含指定对象类型的所有属性名。把所有属性名都添加到列表框中：



可从
wrox.com
下载源代码

```
private void SetUserProperties(string schemaNamingContext)
{
    var properties = from p in GetSchemaProperties(schemaNamingContext,
                                                    "User").Concat(
                                                    GetSchemaProperties(schemaNamingContext,
                                                    "Organizational-Person")).Concat(
                                                    GetSchemaProperties(schemaNamingContext, "Top"))
                    orderby p
                    select p;
    listBoxProperties.DataContext = properties;
}
```

代码段 UserSearch/MainWindow.xaml.cs

在 `GetSchemaProperties()` 方法中, 再次访问 Active Directory 服务。这次不使用 `rootDSE`, 而使用前面介绍的架构的 LDAP 名称。`systemMayContain` 属性包含 `objectType` 类中的所有属性的一个集合:

```
private IEnumerable<string> GetSchemaProperties(string schemaNamingContext,
                                              string objectType)
{
    IEnumerable<string> data;
    using (var de = new DirectoryEntry())
    {
        de.Username = username;
        de.Password = password;

        de.Path = String.Format("LDAP://{0}CN={1},{2}", hostname, objectType,
                                schemaNamingContext);

        PropertyValueCollection values = de.Properties["systemMayContain"];
        data = from s in values.Cast<string>()
              orderby s
              select s;
    }
    return data;
}
```

这样就完成了应用程序的第二步。Listbox 控件包含 `user` 对象的所有属性名。

52.4.4 搜索用户对象

搜索按钮的处理程序只调用辅助方法 `FillResult()`:



可从
wrox.com
下载源代码

```
private void OnSearch(object sender, RoutedEventArgs e)
{
    try
    {
        FillResult();
    }
    catch (Exception ex)
    {
        MessageBox.Show(String.Format("check your input! {0}", ex.Message));
    }
}
```

代码段 UserSearch/MainWindow.xaml.cs

在 FillResult()方法中,在整个 Active Directory 域中进行与前面一样的正常搜索。把 SearchScope 设置为 Subtree,把 Filter 设置为从 TextBox 获取中的字符串,应加载到缓存中的属性由用户在列表框中选择的值设置。DirectorySearcher 类的 PropertiesToLoad 属性是 StringCollection 类型,其中应加载的属性可以使用 AddRange()方法添加,该方法的参数是一个字符串数组。应加载的属性用 SelectedItems 属性从 listBoxProperties 列表框中读取。在设置 DirectorySearcher 对象的属性后,就调用 SearchAll()方法搜索属性。SearchResultCollection 中的搜索结果用于生成写入 textBoxResults 文本框中的汇总信息。

```
private void FillResult()
{
    using (var root = new DirectoryEntry())
    {
        root.Username = username;
        root.Password = password;
        root.Path = String.Format("LDAP://{0}/{1}",
                                hostname, defaultNamingContext);
        using (var searcher = new DirectorySearcher())
        {
            searcher.SearchRoot = root;
            searcher.SearchScope = SearchScope.Subtree;
            searcher.Filter = textFilter.Text;
            searcher.PropertiesToLoad.AddRange(
                listBoxProperties.SelectedItems.Cast < string > ().ToArray());

            SearchResultCollection results = searcher.FindAll();
            var summary = new StringBuilder();
            foreach (SearchResult result in results)
            {
                foreach (string propName in result.Properties.PropertyNames)
                {
                    foreach (object p in result.Properties[propName])
                    {
                        summary.AppendFormat(" {0}: {1}", propName, p);
                        summary.AppendLine();
                    }
                }
                summary.AppendLine();
            }
            textResult.Text = summary.ToString();
        }
    }
}
```

启动该应用程序,给出其中筛选器有效的所有对象对应的一个列表,如图 52-11 所示。

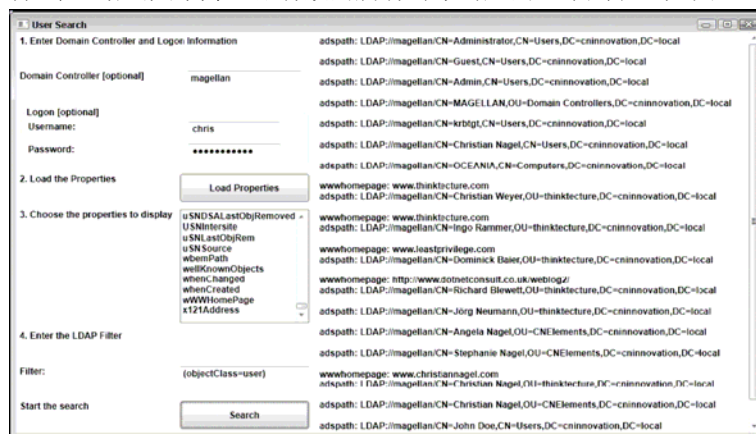


图 52-11

52.5 账户管理

在.NET 3.5 推出之前，很难创建和修改用户账户和组账户。以前使用的一种方式是利用 System.DirectoryServices 名称空间中的类或强类型化的本地 COM 接口。从.NET 3.5 开始新增了 System.DirectoryServices.AccountManagement 程序集，它提供 System.DirectoryServices 类的抽象，方法是使用特定的方法和属性搜索、修改、创建、更新用户和组。

这些类及其功能如表 52-4 所示。

表 52-4

类	说 明
PrincipalContext	使用 PrincipalContext 类可以配置账户管理的上下文。这里可以确定是否使用 Active Directory 域、本地系统中的账户或应用程序目录。为此，应把 ContextType 枚举设置为 Domain、Machine 或 ApplicationDirectory。根据上下文类型，还可以定义域名，指定用于访问的用户名和密码
Principal	Principal 类是所有主体的基类。使用静态方法 FindByIdentity()，可以获得一个 Principal 标识对象。通过主体对象可以访问各种属性，如架构中的名称、描述、可分辨的名称，以及对象类型。如果除了这个类中可用的方法和属性之外，还需要更多地控制主体，就可以使用 GetUnderlyingType()方法，它返回基础 DirectoryEntry 对象
AuthenticablePrincipal	AuthenticablePrincipal 类派生自 Principal 类，是所有可以验证身份的主体的基类。有几个查找主体的静态方法，如按登录时间或锁定时间、按不正确的密码尝试次数或按密码设置时间来查找。使用实例方法可以修改密码，释放账户
UserPrincipal ComputerPrincipal	UserPrincipal 类和 ComputerPrincipal 类派生自 AuthenticablePrincipal 基类，因此拥有这个基类的所有属性和方法。UserPrincipal 对象映射到用户账户，ComputerPrincipal 对象映射到计算机账户。UserPrincipal 类有许多属性可以获取和设置用户信息，如 EmployeeId、EmailAddress、GivenName 和 VoiceTelephoneNumber
GroupPrincipal	组不能进行身份验证，因此 GroupPrincipal 类直接派生自 Principal 类。在 GroupPrincipal 类中，可以使用 Members 属性和 GetMembers()方法获得组的成员
PrincipalCollection	PrincipalCollection 类包含一组 Principal 对象；例如，GroupPrincipal 类的 Members 属性返回一个 PrincipalCollection 对象
PrincipalSearcher	PrincipalSearcher 类是 DirectorySearcher 类的一个抽象，专门用于账户管理。使用 PrincipalSearcher 类不需要了解 LDAP 查询语法，因为这是自动创建的
PrincipalSearchResult<T>	PrincipalSearcher 类和 Principal 类的搜索方法返回一个 PrincipalSearchResult<T>

下面几节介绍可以使用 System.DirectoryServices.AccountManagement 名称空间中的类的场合。

52.5.1 显示用户信息

UserPrincipal 类的 Current 静态属性返回一个 UserPrincipal 对象和当前登录用户的信息：



```
using (var user = UserPrincipal.Current)
{
    Console.WriteLine("Context Server: {0}", user.Context.ConnectedServer);
    Console.WriteLine(user.Description);
    Console.WriteLine(user.DisplayName);
    Console.WriteLine(user.EmailAddress);
    Console.WriteLine(user.GivenName);
    Console.WriteLine("{0:d}", user.LastLogon);
}
```

代码段 AccountManagementSamples/Program.cs

运行这个应用程序，会显示用户的信息：

```
Context Server: Magellan.cninnovation.local
Developer, Author, Trainer, Consultant
Christian Nagel
Christian@ChristianNagel.com
Christian
2009/09/25
```

52.5.2 创建用户

使用 `UserPrincipal` 类可以创建新用户。首先需要有一个 `PrincipalContext` 类定义创建用户的位置。在 `PrincipalContext` 类中，根据应使用目录服务、计算机的本地账户还是应用程序目录，把 `ContextType` 设置为 `Domain`、`Machine` 或 `ApplicationDirectory` 的一个枚举值。如果当前用户不能在 Active Directory 中添加账户，那么还可以使用 `PrincipalContext` 类设置用户和密码，用于访问服务器。

接着，传递该 `PrincipalContext`，设置需要的属性，创建 `UserPrincipal` 类的一个实例。这里设置 `GivenName` 和 `EmailAddress` 属性。最后，必须调用 `UserPrincipal` 类的 `Save()` 方法，把新用户写入存储器中：



```
using (var context = new PrincipalContext(ContextType.Domain, "cninnovation"))
using (var user = new UserPrincipal(context, "Tom", "P@ssw0rd", true)
{
    GivenName = "Tom",
    EmailAddress = "test@test.com"
})
{
    user.Save();
}
```

代码段 AccountManagementSamples/Program.cs

52.5.3 重置密码

要给已有用户重置密码，可以使用 `UserPrincipal` 对象的 `SetPassword()` 方法：

```
using (var context = new PrincipalContext(ContextType.Domain, "cninnovation"))
using (var user = UserPrincipal.FindByIdentity(context, IdentityType.Name,
    "Tom"))
{
    user.SetPassword("Pa$$w0rd");
    user.Save();
}
```

代码段 AccountManagementSamples/Program.cs

运行上述代码的用户需要有重置密码的权限。要把旧密码改为新密码，可以使用 `ChangePassword()` 方法。

52.5.4 创建组

新组的创建方式与新用户的创建相同。这里仅使用 `GroupPrincipal` 类替代 `UserPrincipal` 类。与创建新用户一样，也要设置属性，并调用 `Save()` 方法：



```
using (var ctx = new PrincipalContext(ContextType.Domain, "cninnovation"))
using (var group = new GroupPrincipal(ctx)
{
    Description = "Sample group",
    DisplayName = "Wrox Authors",
    Name = "WroxAuthors"
})
{
    group.Save();
}
```

代码段 AccountManagementSamples/Program.cs

52.5.5 在组中添加用户

要在组中添加用户，可以使用 `GroupPrincipal`，并把 `UserPrincipal` 添加到组的 `Members` 属性中。要获得已有用户和组，可以使用静态方法 `FindByIdentity()`：



```
using (var context = new PrincipalContext(ContextType.Domain))
using (var group = GroupPrincipal.FindByIdentity(
    context, IdentityType.Name, "WroxAuthors"))
using (var user = UserPrincipal.FindByIdentity(
    context, IdentityType.Name, "Stephanie Nagel"))
{
    group.Members.Add(user);
    group.Save();
}
```

代码段 AccountManagementSamples/Program.cs

52.5.6 查找用户

`UserPrincipal` 对象的静态方法可以根据某些预定义的条件查找用户。这里的示例说明了如何使用 `FindPasswordSetTime()` 方法查找在最近 30 天内未修变密码的用户。这个方法返回一个 `PrincipalSearchResult<UserPrincipal>` 集合，迭代它就可以显示用户名、最后一次登录时间和重置密码的时间：



```
using (var context = new PrincipalContext(ContextType.Domain, "explorer"))
using (var users = UserPrincipal.FindByPasswordSetTime(context,
    DateTime.Today - TimeSpan.FromDays(30), MatchType.LessThan))
{
    foreach (var user in users)
    {
        Console.WriteLine("{0}, last logon: {1}, " +
            "last password change: {2}", user.Name, user.LastLogon,
```

```

        user.LastPasswordSet);
    }
}

```

代码段 AccountManagementSamples/Program.cs

UserPrincipal 类中用于查找用户的其他方法有 FindByBadPasswordAttempt()、FindByExpirationTime()、FindByLockoutTime() 和 FindByLogonTime()。

使用 PrincipalSearcher 类可以更灵活地查找用户。这个类是 DirectorySearcher 类的一个抽象，且在后台使用 DirectorySearcher 类。在 PrincipalSearcher 类中，可以把任意 Principal 对象赋予 QueryFilter 属性。

在下面的例子中，把带有 Surname 和 Enabled 属性的 UserPrincipal 对象赋予 QueryFilter。这样，就用 PrincipalSearchResult 集合返回所有姓氏为 Nag 且启用的用户对象。PrincipalSearcher 类创建一个 LDAP 查询字符串，进行搜索。



可从
wrox.com
下载源代码

```

var context = new PrincipalContext(ContextType.Domain);

var userFilter = new UserPrincipal(context);
userFilter.Surname = "Nag*";
userFilter.Enabled = true;

using (var searcher = new PrincipalSearcher())
{
    searcher.QueryFilter = userFilter;
    var searchResult = searcher.FindAll();
    foreach (var user in searchResult)
    {
        Console.WriteLine(user.Name);
    }
}

```

代码段 AccountManagementSamples/Program.cs

52.6 DSML

在 System.DirectoryServices.Protocols 名称空间中，可以通过 DSML(目录服务标记语言)访问 Active Directory。DSML 是 OASIS 组(<http://www.oasis-open.org>)定义的一个标准，它允许通过 Web 服务访问目录服务。要通过 DSML 访问 Active Directory，至少要安装 Windows Server 2003 R2。

图 52-12 显示了 DSML 的一个配置。提供 DSML 服务的系统通过 LDAP 访问 Active Directory。在客户端系统上，使用 System.DirectoryServices.Protocols 名称空间中的 DSML 类向 DSML 服务发出 SOAP 请求。

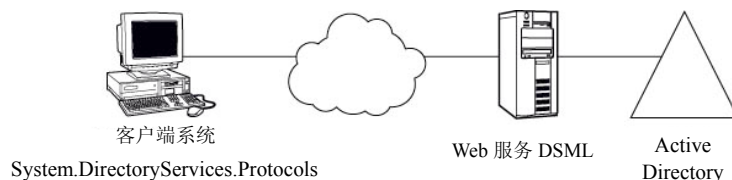


图 52-12


52.6.1 System.DirectoryServices.Protocols 名称空间中的类

表 52-5 列出了 System.DirectoryServices.Protocols 名称空间中的主要类。

表 52-5	
类	说 明
DirectoryConnection	DirectoryConnection 类是可以定义到目录服务的连接的所有连接类的基类。派生自 DirectoryConnection 类的类有 LdapConnection(为使用 LDAP 协议)、DsmSoapConnection 和 DsmSoapHttpConnection。使用 SendRequest()方法可以把消息发送给目录服务
DirectoryRequest	派生自 DirectoryRequest 基类的类可以把请求发送给目录服务。根据请求的类型, 使用 SearchRequest、AddRequest、DeleteRequest 和 ModifyRequest 等类发送请求
DirectoryResponse	用 SendRequest 返回的结果的类型派生自 DirectoryResponse 基类。这种派生类的示例有 SearchResponse、AddResponse、DeleteResponse 和 ModifyResponse

52.6.2 用 DSML 搜索 Active Directory 对象

本节用一个例子说明如何搜索目录服务对象。在下面的代码中, 首先实例化一个 DsmSoapHttpConnection 对象, 该对象定义与 DSML 服务的连接。该连接用包含 Uri 对象的 DsmDirectoryIdentifier 类定义, 还可以使用该连接设置用户证书:



可从
wrox.com
下载源代码

```
Uri uri = new Uri("http://dsmlserver/dsml");
var identifier = new DsmDirectoryIdentifier(uri);

var credentials = new NetworkCredential();
credentials.UserName = "cnagel";
credentials.Password = "password";
credentials.Domain = "explorer";

var dsmlConnection = new DsmSoapHttpConnection(identifier, credentials);
```

代码段 DsmSample/Program.cs

定义连接后, 就可以配置搜索请求。搜索请求包含开始搜索的目录项、LDAP 搜索筛选器和应从搜索中返回的属性值的定义。这里把筛选器设置为(objectClass=user), 以便从搜索中返回所有用户对象。把 attributesToReturn 属性设置为 null, 并且可以读取所有带值的属性。SearchScope 是 System.DirectoryServices.Protocols 名称空间中的一个枚举, 它类似于 System.DirectoryServices 名称空间中的 SearchScope 枚举, 它定义搜索的深度。这里把 SearchScope 属性设置为 Subtree, 搜索整个 Active Directory 树。

搜索筛选器可以用 LDAP 字符串定义, 或使用包含在 XmlDocument 类中的一个 XML 文档定义:

```
string distinguishedName = null;
string ldapFilter = "(objectClass=user)";
string[] attributesToReturn = null;// return all attributes

var searchRequest = new SearchRequest(distinguishedName,
    ldapFilter, SearchScope.Subtree, attributesToReturn);
```


用 `SearchRequest` 对象定义该搜索后,就调用 `SendRequest()`方法,把该搜索发送给 Web 服务。`SendRequest()`是 `DsmlSoapHttpConnection` 类的一个方法,它返回一个 `SearchResponse` 对象,从该对象中可以读取返回的对象。

除了调用同步的 `SendRequest()` 方法, `DsmlSoapHttpConnection` 类还提供了异步的 `BeginSendRequest()`和 `EndSendRequest()`方法,这些异步方法遵循异步.NET 架构。



异步架构可参见第 20 章。

```
SearchResponse searchResponse =
    (SearchResponse)dsmlConnection.SendRequest(searchRequest);
```

返回的 `Active Directory` 对象可以在 `SearchResponse` 中读取。`SearchResponse.Entries` 包含用 `SearchResultEntry` 类型包装的所有项的一个集合。`SearchResultEntry` 类的 `Attributes` 属性包含所有特性。每个属性都可以使用 `DirectoryAttribute` 类读取。

在代码示例中,把每个对象的可分辨的名称都写入控制台中。接着,访问组织单元的属性值,把组织单元的名称写入控制台中。之后把 `DirectoryAttribute` 对象的所有值写入控制台中。

```
Console.WriteLine("\r\nSearch matched {0} entries:",
    searchResponse.Entries.Count);
foreach (SearchResultEntry entry in searchResponse.Entries)
{
    Console.WriteLine(entry.DistinguishedName);

    // retrieve a specific attribute
    DirectoryAttribute attribute = entry.Attributes["ou"];
    Console.WriteLine("{0} = {1}", attribute.Name, attribute[0]);
    // retrieve all attributes
    foreach (DirectoryAttribute attr in entry.Attributes.Values)
    {
        Console.WriteLine("{0}=", attr.Name);

        // retrieve all values for the attribute
        // the type of the value can be one of string, byte[] or Uri
        foreach (object value in attr)
        {
            Console.WriteLine("{0} ", value);
        }
        Console.WriteLine();
    }
}
```

添加、修改和删除对象的操作与搜索对象类似。根据要执行的操作,使用相应的方法。

52.7 小结

本章介绍了 `Active Directory` 的体系结构,域、树和森林的重要概念。利用它,可以访问整个企业的信息。在编写访问 `Active Directory` 服务的应用程序时,必须注意读取的数据可能不是最新的,

因为有复制延迟。

使用 `System.DirectoryServices` 名称空间中的类，可以很容易地访问包装到 ADSI 提供程序中的 Active Directory 服务。`DirectoryEntry` 类可以直接读写数据存储器中的对象。

使用 `DirectorySearcher` 类可以进行复杂的搜索，定义筛选器、超时、加载的属性和范围等。使用全局目录，可以加快对整个企业中的对象的搜索，因为它在森林中存储了所有对象的只读版本。

DSML 是另一个允许通过 Web 服务接口访问 Active Directory 的 API。

`System.DirectoryServices.AccountManagement` 名称空间中的类提供了一个抽象，很容易创建和修改用户、组和计算机账户。