

第12章 通用数据访问

通用数据访问(Universal Data Access, UDA)是对Microsoft访问多种数据源数据的策略的称呼。不必担心,它并不是那些新的将改变我们所做的每项工作的开发技术。事实上,此时也许我们正在使用着UDA的一部分。前面的几章已经向读者介绍了怎样用ADO来访问数据库,尤其是关系数据库,而这就是UDA的一部分。数据以不同的形式出现,而UDA的思想就是通过使用OLE DB和ADO实现对数据的访问。

本章将介绍什么是UDA,以及为什么要集中介绍UDA。我们将介绍以下内容:

- UDA如何与全球的Web开发人员相适应。
- 可访问通用数据的OLE DB提供者。
- 半结构化的数据的定义,如何使用这些数据。
- Internet发布怎样改变人们的生活。
- 企业界中的数据存储和ASP。

实际上,本章所涉及的内容相当多且相当广泛,我们只能用较少的篇幅进行介绍。

12.1 UDA的构想

UDA实际上就是使用相同技术来访问不同类型数据的能力。这并不是一种新思想,而是对现有的ODBC技术的扩展。ODBC提供了一种访问关系数据库的方法,并成为跨数据库访问的标准。

虽然ODBC很成功,但也有其自身的缺点,那就是无法访问非关系数据库中的数据。当使用ODBC访问文本文件或电子表格时,确实受到较大范围的限制。当前越来越多的数据正以

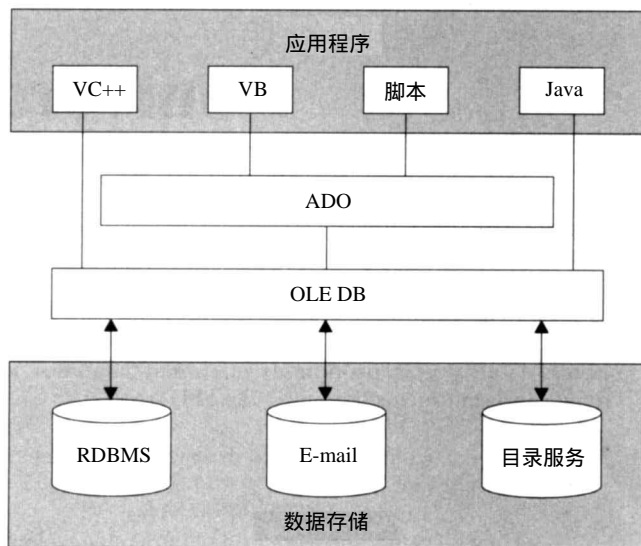


图12-1 UDA的构想示意图

文档的形式存储,如金融系统、邮件系统等,而 ODBC恰恰不是为访问这些数据而设计的。同时,还有大量的传统数据不是精确符合关系数据库的格式。

为了解决这个问题,Microsoft推出了UDA。实际上,UDA只是对Microsoft多年努力的成果的一个称呼。如果没有记错的话,我是在 1994年初识OLE DB的,即访问ODBC数据的一个测试版本(不是很完善)。从那时起,Microsoft就开始注意扩展OLE DB所访问的数据范围,不仅是关系数据库,还有其他数据提供者。

使用ADO 2.5可以访问不同形式的数据,但所有的数据访问都是通过相同的简单对象来实现的。这使得数据访问变得通用化了,即同样的技术可用不同的语言实现,只要稍做修改就可以了。

UDA的核心是OLE DB。而OLE DB基于ODBC,但比它更进一步。在前面第一次讨论ADO时,我们就是利用OLE DB提供者来访问数据存储的。为了进一步理解,看一下图 12-1。

这里,我们可以看到OLE DB不仅可用于访问数据库,而且可用于访问所有不同类型的数据。虽然现在OLE DB大多用于访问关系数据库,但访问其他类型的数据也毫无问题。

12.2 OLE DB提供者

是否有OLE DB提供者是唯一限制用户访问数据的因素。如果有,那么通过一般的OLE DB或ADO命令就可以访问数据。如果产品的供应商没有编写数据提供者,那么将来设计OLE DB提供者的可能性就会很大。这是一个很简单的事情。如果想让别人使用自己的产品,必须提供一种简易的访问数据的方法。

OLE DB和ADO是为了满足多种数据类型的需要,以及为在Web方案中使用数据而开发设计的。Web仍在不断扩展,ASP正成为Web应用程序中间层的一个重要组成部分,所以我们必须考虑Web服务器中数据的不同存储方式。

使用OLE DB提供者可以扩展Web应用程序而不必学习新的技术,我们只需为不同类型的数据使用不同的提供者。提供者是很多的,但这里需要研究的是那些可用来访问非关系数据库数据的提供者。特别是,将研究集成了多种Microsoft技术的提供者,并介绍如何在Web应用程序中使用它们。

12.2.1 Indexing Service

如果Web站点为用户提供信息访问功能,那么很有可能需要为用户提供某种形式的搜索功能。Indexing Service(索引服务)与IIS集成在一起,并且有一个OLE DB提供者可对Indexing Service目录进行只读访问。

本书对Indexing Service的安装、使用及其高级功能不做介绍。

1. Indexing Service分析

Indexing Service可以说与数据库管理服务器相似。它们都拥有信息,并允许通过查询访问信息。两者的本质区别在于数据库管理服务器,比如SQL Server必须需要某些人往表中填写数据,而Indexing Service自动地做了这项工作。对大多数开发人员来说,很难完全理解这个工作过程,但至少应理解以下几点:

- Indexing Service的最基本的组成部分是Index条目。如果这是数据库表中的一条记录,那么它仅是一个关键字或一个对已建立的关键字的文档的引用。这些记录可能会在数据

库中的一个表中，而另一个表可能包含第一个表所引用的所有文档。这两个表的概念在 Indexing Service中表示为 Word List和Saved Index。

- 有了 Indexing Service，离散数据库就是我们所知的目录 (catalog)。正如 SQL Server能处理多个数据库一样，Indexing Service也可以处理多个目录。目录一般被捆绑在主机所提供的基于Internet的服务上。因此，假如你有一个 Web站点并用 NNTP服务提供新闻组，可以为它们提供一个目录。如果使用虚拟 Web站点，它们中的每一个也可以有一个目录。

当服务启动时，他知道从哪里开始搜索索引的条目，因此他建立了一个被索引的文件列表。我们把这项工作称为扫描。被索引的条目可以是存储于文件夹中的任何文件。文件夹通常被设置为服务环境中的虚拟路径。这就是我们所知的作用范围。

一旦扫描过程完成，过滤过程就开始了。在过滤期间，Indexing Service将试图打开一个候选文件。一旦打开该文件，Indexing Service就会对此文件进行解析，并为每个关键字构建一个索引条目。同时，也为文件计算其他的键参数。解析一个文件需要了解文件的结构。这个信息收集在被称为过滤器的库文件中。Microsoft已经开发了大多数在 Internet和Internet 服务中常见的内容类型的过滤器。

使用非标准或非 Microsoft文件格式的厂商也提供自己的过滤器。例如，Adobe公司就为PDF文档标准提供了过滤器。因此，如果你创建了自己的文件格式，那么应该为它定义一个过滤器。

“特征”和“文档属性”是两个需要理解的重要概念。

在某种意义上，特征 (characterization)指的是文档的摘要 (或执行概要)。特征通常是文档中可表示部分的前 n 个字符。可表示部分一般是显示的 (或表现的)文档的部分内容。比如一个 HTML页面的特征一般是指位于 <BODY>标记之间的内容 (自然，也有例外)。Indexing Service允许用户决定文档的特征有多大。一般来说，250~500个词足够了。当从 Indexing Service返回查询结果时，特征是经常显示的信息。文本的开头部分通常是显示的最佳选择，因为他可使用户很快就能决定这个文档是否需要进行深入研究。

然而，文档的大量附加信息也可以从 Indexing Service的文档属性中获得。这些属性是正式的ActiveX文档属性。Indexing Service说明了40多种这样的属性。文件大小、数据的创建时间、最近访问时间、最近修改时间以及文件的位置是，ASP应用程序中对 Indexing Service查询起决定作用的最重要的属性。

Indexing Service 3.0与Index Server的对比

如果你在 IIS 3.0或4.0环境下开发 ASP程序，也许想知道为什么要使用 Indexing Service而不是Index Server，他们不是一回事吗？

可以说是，也可以说不是。Indexing Service与Index Server完成了相同的工作，即创建了引用文档的查询数据库。但 Indexing Service带来了一些新的以及改进过的功能。

- 那些抱怨Microsoft在IIS 4.0环境下的NT Workstation中删除了Index Server的人，现在可以放心了，因为在 Windows 2000 Professional中包含Indexing Service。事实上在 Windows 2000 家族中，当查找文件夹或文件时，Start菜单中的查找选项就使用了 Indexing Service。
- 现在结果集可以由 Query对象获得。利用这种方法，可以通过使用某个属性向 Indexing

Service请求返回结果集。

- 多种语言用于建立查询。
- 随同Web内容，Indexing Service也维护本地文件系统中的文件的一个目录。这有点类似于Microsoft Office中的快速查找程序。
- 在Index Server的早期版本中，文件系统在内容索引服务开始时就被彻底扫描一次。现在被索引的内容放在新的NTFS5文件系统中，只有改变过的文件才会被再次扫描。
- 协助维护Indexing Service的三个新对象与Query和Utility对象协同工作，它们分别是AdminIndexServer、CatAdm和ScopeAdm对象。

2. 搜索索引目录

使用ADO查找目录最为不方便之处在于需要经常不断地分析用户的输入，以使查询正常工作。分析一下查找“ASP ADO”的工作情况。它可能有以下三种含义：

- 找出所有既包含ASP又包含ADO的文档。
- 找出所有包含ASP，或者包含ADO，或者包含两者的文档。
- 找出所有包含ASP，并且ASP后面跟有字符ADO的页面。

通常，用户会假定某一种行为，除非你明确表明程序将做某些不同的事。而用户的假定可能各不相同。更糟的是仅仅传递一个“ASP ADO”，在ADO查询技术中会产生下面的错误：

Microsoft OLE DB Provider for Microsoft Indexing Service error '80040e14'

Incorrect Syntax near 'ADO'. Expected ''. SQLSTATE = 42000

这就是问题的所在。多数情况下，在把查找项目包括进SQL查询之前，要把它们分解为单个的条目和操作符。这要付出时间，如果不能达到目的，可能还需要另外的时间。在介绍Indexing Server中的SQL语言时，我们会详细分析怎样分解输入的词组。

由于Indexing Service不是数据库，因此有两点需加以注意：

- Indexing Service目录是连接的数据源。
- Indexing Service提供者使用SQL语言的扩展。

为了显示查询Indexing Service是多么容易，下面来创建一个Web页面的例子。首先，以搜索窗体SearchForm.html开始，如图12-2所示。

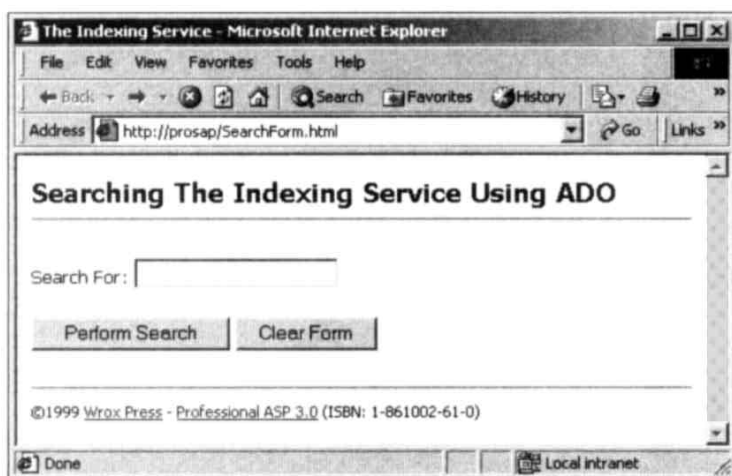


图12-2 SearchForm.html搜索窗体

相应的HTML脚本相当简单：

```
<FORM Name="frmSearch" ACTION="QueryIndexServer.asp" METHOD="POST">
Search For: <INPUT TYPE="TEXT" NAME="txtSearchFor">
<P>
<INPUT TYPE="SUBMIT" VALUE="Perform Search">
<INPUT TYPE="RESET" VALUE="Clear Form">
</FORM>
```

以上代码提供一个文本框，可以在其中输入搜索条件。按下 Perform Search 按钮将调用 ASP 页面 QueryIndexServer.asp。现在分析一下该 ASP 页面的代码。

首先，包含文件 Connection.asp：

```
<%@ LANGUAGE=VBSCRIPT %>
<!-- #INCLUDE FILE="../../Include/Connection.asp" -->
```

这包含了对 ADO 类型库的引用，这样就可以使用 ADO 常数了。

```
<!-- METADATA TYPE="typelib"
FILE="C:\Program Files\Common Files\System\ado\msado15.dll" -->
```

接下来，定义一些 CSS 样式。这些不是必须的，但这样可以比较轻松地格式化输出结果。

```
<HTML>
<HEAD>
<TITLE>The Indexing Service</TITLE>
<STYLE TYPE="text/css">
BODY {font-family:Tahoma,Arial,sans-serif; font-size:10pt}
.heading {font-family:Tahoma,Arial,sans-serif; font-size:14pt;
font-weight:bold}
.cite {font-family:Tahoma,Arial,sans-serif; font-size:8pt}
.document {font-size:10pt; font-weight:bold;
background-color:lightgrey; width:100%}
</STYLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<SPAN CLASS="heading">Results of search for
<I><%=Request.Form("txtSearchFor")%></I>
</SPAN><HR>
<!------->
```

现在是真正的 ASP 代码。我们创建 Recordset 对象，并设置连接字符串。稍后，我们将解释确切的连接字符串。

```
<%
Dim strSearch
Dim rsSearch

Set rsSearch = Server.CreateObject("ADODB.Recordset")

' Create the connection string
strConn = "Provider=MSIDXS; Data Source=Web"
```

接下来构造查询字符串，它是 SQL 语言的扩展集。在后面我们也会详细介绍。

```
' Construct the search string
strSearch = "SELECT DocTitle, Path, FileName, Characterization, Size" & _
" FROM SCOPE()" & _
" WHERE CONTAINS ('" & Request.Form("txtSearchFor") & " ')"
```

下一步打开 Recordset 对象，遍历记录，并显示结果。

```
' Open the recordset on the search
rsSearch.Open strSearch, strConn
```



```
' Show what's been searched for
While Not rsSearch.EOF
    Response.Write "<SPAN CLASS='document'>" & _
        rsSearch("DocTitle") & "</SPAN><BR>" & _
        rsSearch("Characterization") & "<BR>" & _
        "<A HREF='" & rsSearch("Path") & "'">" & _
        rsSearch("FileName") & "</A>" & _
        " (" & rsSearch("Size") & " bytes)<P>"

    rsSearch.MoveNext
Wend

' Tidy up
rsSearch.Close
Set rsSearch = Nothing
%>

</BODY>
</HTML>
```

在我的计算机里，查找“ado”给出了如图12-3所示的结果(读者可能会得到与此不同的结果，这取决于被索引的目录，以及在那些目录中所包含的文档)。

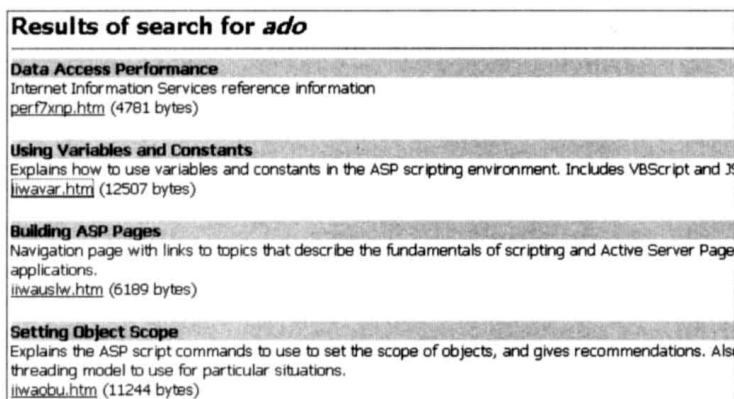


图12-3 查找“ado”的结果

让我们来分析一下两段重要的代码：连接和查询。

3. 用ADO连接到Indexing Service

与我们所见的其他ADO连接字符串一样，用于Indexing Service的连接字符串指定了OLE DB提供者的名称和数据的来源。

```
strConn = "Provider=MSIDXSS; Data Source=Web"
```

Data Source是要搜索的Indexing Service目录的名称。可以在Computer Management Console(在资源管理器中用鼠标右键单击My Computer，选择Management菜单项)中找到目录名，如图12-4所示。

4. 用于Indexing Service的搜索字符串

搜索字符串类似于一个标准的SQL查询，但有特殊的关键字。这个概念与数据库中的搜索概念一致，首先选择需要看到的项目，然后指定数据从哪里来，接着加入一些条件来限制输出结果。

我们先快速看一下这个查询，然后再详细讲解它的语法。

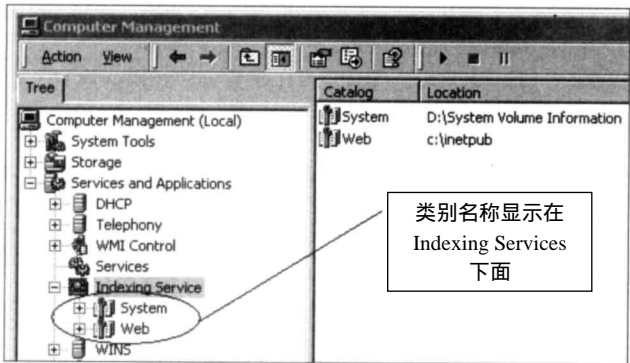


图12-4 Computer Management Console窗口

在SELECT语句中出现的字段确定选取目录的哪些属性：

```
strSearch = "SELECT DocTitle, Path, FileName, Characterization, Size" & _
```

接下来是FROM子句，它表明数据来自哪里。在本例中 SCOPE没有任何参数，所以搜索的范围没有限制，因此会检查所有的目录条目（在下面的“FROM子句”小节对此做了详细的说明）：

```
" FROM SCOPE()" & _
```

最后需要说明查找的内容，用CONTAINS来查找包含特定字符串的文档：

```
" WHERE CONTAINS ('' & Request.Form("txtSearchFor") & '')"
```

5. 字段名与属性

如果你对Indexing Service不熟悉，那么可能不知道哪些字段是可用的，实际上这个问题很容易解决，因为它们都列在Indexing Service管理工具中，如图12-5所示。

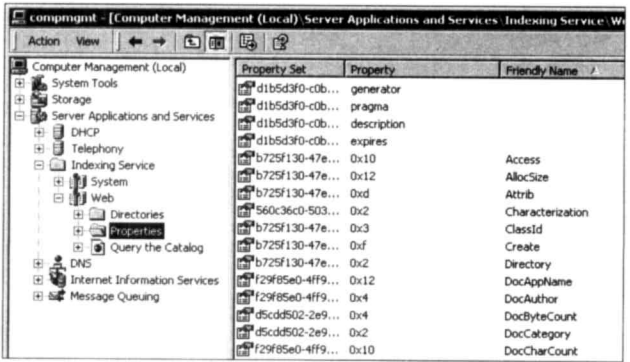


图12-5 Indexing Service的管理工具

可以使用友好名或属性名作为字段名。

注意，此项技术仅应用于Indexing Service提供者，并不适用于其他OLE DB提供者。

下面我们来看一下Indexing Service SQL与标准SQL的区别。

6. SELECT子句

SELECT子句只用来从可用的属性名列表中选取字段名。

可能你会倾向于使用SELECT * 而不是分别指定单个列名。然而，如果在查询目录时这样做的话，OLE DB提供者就会提示出错。*号选取仅对Indexing Service目录的视图有效。

7. FROM子句

From子句允许选择目录中的信息来源，这与一些标准 SQL的FROM子句有一点区别。有三种不同的使用FROM子句的方法：

- 使用预定义视图。
- 使用定制视图。
- 使用一个范围。

(1) 预定义视图

选择列的第一种方法是使用一个预定义视图名，这个视图只是一些字段的别名。例如：

```
SELECT *
FROM FILEINFO_ABSTRACT
```

表12-1为预定义视图的列表。

表12-1 预定义视图的列表

预定义视图	字段/属性
FILEINFO	path,FileName, size, write, attrib
FILEINFO_ABSTRACT	path,FileName, size, write, attrib,Characterization
EXTENDED_FILEINFO	path, FileName, size, write, attrib, DocTitle, DocAuthor, DocSubject, DocKeywords,Characterization
WEBINFO	Vpath, path, FileName, size, write, attrib,Characterization, DocTitle
EXTENDED_WEBINFO	Vpath, path, FileName, size, Characterization, write, DocAthor, DocSubject, DocKeywords, DocTitle
SSWEBINFO	URL, DocTitle, Rank, size, write
SSEXTENDED_WEBINFO	URL, DocTitle, Rank, Hitcount, DocAuthor, Characterization, size, write

预定义视图确实是一个捷径，它允许从预定义的列集中选择所有的列。同样，也可以选择列名而不使用整个视图。

(2) 定制视图

第二种方法是使用定制视图，它与预定义视图采用相同的格式：

```
SELECT *
FROM MyViewName
```

在这里我们不打算讨论怎样创建定制视图，详细内容请参考 Indexing Service 文档。

(3) 使用一个范围

第三种方法是在搜索中包含搜索的范围。范围真正指出了要搜索的目录，以及搜索的目录深度。范围最简单的一种形式是空范围，即：

```
FROM SCOPE()
```

这表明搜索首先从根目录开始，它下面的所有目录都要被搜索。如果要限制搜索的目录，那么可以加上相应的路径。


```
FROM SCOPE ('D:\users\davids')
```

这只返回选择的目录以及其下面的子目录中的条目。如果只需要搜索选择的目录而不搜索子目录，那么可以加入移动类型。

```
FROM SCOPE('SHALLOW TRAVERSAL OF "D:\users\davids"')
```

移动类型包括：

- SHALLOW TRAVERSAL OF：表明只搜索指定的路径。
- DEEP TRAVERSAL OF：表明搜索指定的路径及所有的子目录。

也可指定多个搜索范围：

```
FROM SCOPE('SHALLOW TRAVERSAL OF "D:\users\davids",  
            'DEEP TRAVERSAL OF "D:\users\janine"')
```

如果没有指定搜索的深度，那么 DEEP TRAVERSAL 是缺省值。同时，可不受物理路径的限制，即，也可以使用虚拟路径。比如：

```
FROM SCOPE('DEEP TRAVERSAL OF "/ProASP"')
```

以上代码使用虚拟目录 ProASP 作为搜索的根目录。

8. WHERE 子句

与标准的 WHERE 子句比较，这里的 WHERE 子句有一个配合搜索的谓词列表。

他们是：

- ARRAY：使用逻辑操作符比较两个数组。这对那些基于数组的字段十分有用，比如一个文件的 attribute 属性。
- CONTAINS：允许匹配单词和短语。例如，下面这行代码能够匹配包括 “ADO” 或 “RDS” 的文档：

```
WHERE CONTAINS("ADO" OR "RDS")
```

下面的例子匹配 “ADO” 在 “RDS” 附近 50 个单词内的文档：

```
WHERE CONTAINS("ADO" NEAR "RDS")
```

下面的例子匹配含有 “drink”、“drinking”、“drunk” 等的文档：

```
WHERE CONTAINS('FORMSOF(INFLECTIONAL, "drink")')
```

- FREETEXT：指定单词或短语的最佳匹配搜索，例如：

```
WHERE FREETEXT('programming ADO')
```

- LIKE：使用通配符执行匹配，例如：

```
WHERE DocAuthor LIKE 'A%'
```

- MATCHES：使用正规的表达式执行匹配，例如：

```
WHERE MATCHES (DocAuthor, "[a-e]*")
```

搜索 DocAuthor 的第一个字符为 a 到 e 之间的任一字符的所有条目。

- NULL：允许匹配空值。例如：

```
WHERE DocAuthor IS NULL
```

```
WHERE DocAuthor IS Not NULL
```

所有这些选择方法使得搜索目录变得更为容易。

详细的信息请参考 Indexing Service 文档。

12.2.2 活动目录

对 Windows 2000 来说，活动目录 (Active Directory, AD) 的 OLE DB 提供者并不是什么新

东西，但只有在 Windows 2000 中 AD 才被完全实现。你可能对活动目录服务接口 (Active Directory services Interface, ADSI) 这个术语更加熟悉。ADSI 是 AD 的编程接口。

对 ADSI 我们不加详述，因为在第 21 章已经覆盖了这部分内容。在这里我们只想简单介绍一下 ADSI 与 ADO 是如何相互作用的。

AD 是 Windows 2000 的目录服务，它管理一个或多个机器的资源。这实际上意味着对于所有的资源信息只有一个源，并且 AD 的 OLE DB 提供者让你能够运用现有的知识来访问这个资源信息。

使用这个 OLE DB 提供者相当简单，只要有一点 AD 的知识。大多数例子显示了一种相当怪异的访问目录信息的语法，就像下面的例子一样：

```
<LDAP://localhost>; ((objectClass=user));adspath;subtree
```

上面这行代码列出了所有的本地用户，但很不直观。幸运的是，我们还可以使用 SQL 样式的语法：

```
SELECT ADsPath, cn FROM 'LDAP://localhost' WHERE objectCategory='user'
```

这行代码输出的结果与上面的相同，但对那些不太了解目录查询语言的人来说更易于理解。

目录查询语言在 AD 文档中有介绍。

1. 路径

需要用路径来唯一地确定一个 ADSI 对象。路径的第一部分确定了目录服务提供者。例如：

- LDAP：代表轻量目录访问协议 (Lightweight Directory Access Protocol)。
- WINNT：代表 Windows NT。
- NDS：代表 NetWare 目录服务 (NetWare Directory Services)。
- NWCOMPAT：代表 Netware 兼容

路径的第二部分确定了名称空间，它因目录服务提供者的不同而不同。下面是一些完整的例子。

```
LDAP://localhost
```

```
LDAP://webdev.wrox.co.uk/CN=WebDev,DC=Wrox,DC=co,DC=uk
```

```
WinNT://HUNDREDACRE/Tigger,Tigger
```

```
WinNT://HUNDREDACRE/Davids
```

```
NDS://WebDev/O=Wrox/OU=Editorial/CN=Davids
```

```
NWCOMPAT://Netware1/HPLaserJet
```

关于目录服务的特殊语法的详细内容请参考相应的文档。

2. 使用针对 ADS 的 OLE DB 提供者

ADS 的 OLE DB 提供者由 Windows 2000 自动完成安装，所以使用它时不需做什么其他工作。

尽管路径与查询字符串很复杂，但从目录中访问各项目却很容易。接下来我们来看一个允许用户选择一个目录中的用户、组或所有对象的例子。

首先创建 ActiveDirectory.html 文件，它提供选择功能：

```
<FORM NAME="frmCategory" ACTION="ActiveDirectory.asp" METHOD="POST">  
<SELECT NAME="lstCategory">
```

```

<OPTION VALUE="user">Users</OPTION>
<OPTION VALUE="group">Groups</OPTION>
<OPTION VALUE="*">All Categories</OPTION>
</SELECT>
<INPUT TYPE="SUBMIT" VALUE="Run"></INPUT>
</FORM>

```

现在创建产生结果的ASP页面，即ActiveDirectory.asp文件。首先创建Recordset对象和连接字符串：

```

<!-- #INCLUDE FILE="../../../Include/RecordsetToTable.asp" -->

<%
    Dim rsUsers
    Dim strQuery
    Dim fldF

    Set rsUsers = Server.CreateObject("ADODB.Recordset")

    ' Set the connection string
    strConn = "Provider=ADSDSOObject"

```

现在我们可以建立查询。从本地机器上选择一般的名字和目录路径，但只选择那些与从选择窗体中挑选的目录相匹配的目录。

```

' Build the query
strQuery = "SELECT cn, ADsPath FROM 'LDAP://localhost' & _
           " WHERE objectCategory=' ' & _
           CStr(Request.Form("lstCategory")) & ' ' & _
           " ORDER BY cn"

```

然后打开Recordset对象并显示结果。RecordsetToTable函数在包含文件RecordsetToTable.asp中，它把一个ADO Recordset对象转换为一个HTML表格。

```

' Open the recordset
rsUsers.Open strQuery, strConn

' Build a table of the details
Response.Write RecordsetToTable(rsUsers, True)

rsUsers.Close
Set rsUsers = Nothing
Set fldF = Nothing

```

```

%>

```

以上代码只是简单地打开了与ADS提供者的一个连接，然后用选择的目录作为过滤器，构建一个简单的查询。得到的结果如图12-6所示。

cn	ADsPath
Administrator	LDAP://localhost/CN=Administrator,CN=Users,DC=ipona,DC=demon,DC=co,DC=uk
David Sussman	LDAP://localhost/CN=David Sussman,CN=Users,DC=ipona,DC=demon,DC=co,DC=uk
Guest	LDAP://localhost/CN=Guest,CN=Users,DC=ipona,DC=demon,DC=co,DC=uk
HUNDREDACRE\$	LDAP://localhost/CN=HUNDREDACRE\$,CN=Users,DC=ipona,DC=demon,DC=co,DC=uk
IUSR_W2000	LDAP://localhost/CN=IUSR_W2000,CN=Users,DC=ipona,DC=demon,DC=co,DC=uk
IWAM_W2000	LDAP://localhost/CN=IWAM_W2000,CN=Users,DC=ipona,DC=demon,DC=co,DC=uk
krbtgt	LDAP://localhost/CN=krbtgt,CN=Users,DC=ipona,DC=demon,DC=co,DC=uk
SQLAgentCmExec	LDAP://localhost/CN=SQLAgentCmExec,CN=Users,DC=ipona,DC=demon,DC=co,DC=uk
TsInternetUser	LDAP://localhost/CN=TsInternetUser,CN=Users,DC=ipona,DC=demon,DC=co,DC=uk
VUSR_W2000	LDAP://localhost/CN=VUSR_W2000,CN=Users,DC=ipona,DC=demon,DC=co,DC=uk

图12-6 查询结果

图12-7中显示了用户的一般名(Common Name, cn), 以及目录中每个用户对应的唯一目录路径。如果看一下Directory Service管理器, 就能知道它们是怎样匹配的。

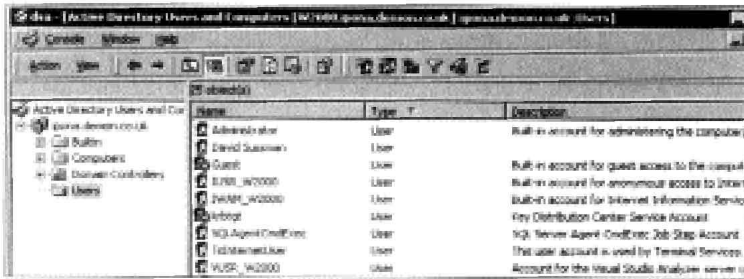


图12-7 Directory Service管理器窗口

12.2.3 Exchange Server

许多人(包括我在内)一直都期望能用ADO连接到Exchange Server。但遗憾的是, 在写本书的时候, 还没有办法通过OLE DB和ADO来访问Exchange Server。Microsoft的Exchange Server 6(代码称为Platinum)目前还在测试中, 它将会有一个可用的OLE DB提供者。虽然目前还没有对它的详细介绍, 但它很有可能允许我们通过使用ADO记录集来访问全部的Exchange存储(包括邮箱、公用文件夹、新闻组等等)。

用ADO访问Exchange目录的优点有以下两点:

- 已熟悉ADO的程序员不必为访问Exchange数据再学习另外的技术, 例如CDO。
- 使用ADO对于某些情况会更方便, 例如它可以快速访问数据存储。

确实有点遗憾, 我们还不得不为此而等待。现在许多Web站点已经提供了简单的邮件功能, 但Exchange提供者应该提供这种可能性, 允许我们更容易地访问公用文件夹、联系信息等等。

12.2.4 定制提供者

如果有一些不能通过现有的OLE DB提供者来访问的数据, 那么有两种方法可以解决这个问题:

- 把数据转换成提供者支持的数据格式。
- 编写自己的OLE DB提供者。

读者可能会认为第一种方案是最简单的解决方法, 也许只要把数据转换成一种数据库格式就可以了。但如果数据来自于另一个实时应用程序, 那么就必须要频繁地进行这种转换, 这也就意味着用户将看不到实时的数据。

第二种解决方案似乎有点不太好, 但情况并非如此。如果数据相对简单, 就能在很短的时间内编写出自己的提供者。例如, 在查看XML数据时, 只要XML文件符合Microsoft的模式, 那么就能以一个记录集的形式打开XML文件。但如果XML数据带有元素格式怎么办? 例如下面的代码:

```
<Authors>
  <Author>
    <au_id>172-32-1176</au_id>
```

```
<au_lname>White</au_lname>
<au_fname>Johnson</au_fname>
</Author>
<Author>
  <au_id>213-46-8915</au_id>
  <au_lname>Green</au_lname>
  <au_fname>Marjorie</au_fname>
  <phone>415 986-7020</phone>
</Author>
</Authors>
```

也许使用XSL或一些DOM代码，就可以很容易地把它们转换成模式版本的XML数据。为什么不编写一个读这些数据的OLE DB提供者呢？这比想象得要简单。

简易提供者

这种类型的提供者被称为简易提供者 (Simple Provider)，因为其数据只是一个简单的文本文件，并且它容易编写。实际编写一个这种类型的XML提供者要用去大约10分钟的时间。这里不详细地介绍，但给出在Visual Basic中怎样实现的简单指导。

要获得更多的细节请参考OLE DB文档，可以从MSDN得到最新的文档，其网址是<http://msdn.microsoft.com>。

用VB创建一个OLE DB简易提供者需要一个带有两个类和以下引用集的ActiveX DLL工程：

- Microsoft ADO 2.5库。
- Microsoft数据源接口。
- Microsoft OLE DB Simple Provider1.5库。
- Microsoft OLE DB错误库。

在本例中，我们的提供者还需要2.0版本的Microsoft XML。

所需的第一个类是XMLConnection，需要把它的Data Source Behavior属性设为1(vbDataSource)，以告知Visual Basic这是一个数据源，它是一个OLE DB提供者的一个组成部分。然后，用这个类的GetDataMember方法创建第二个类：XMLRecordset，它读取XML文件数据。

这听起来很复杂，但实际上很简单。图12-8显示了其工作方式。

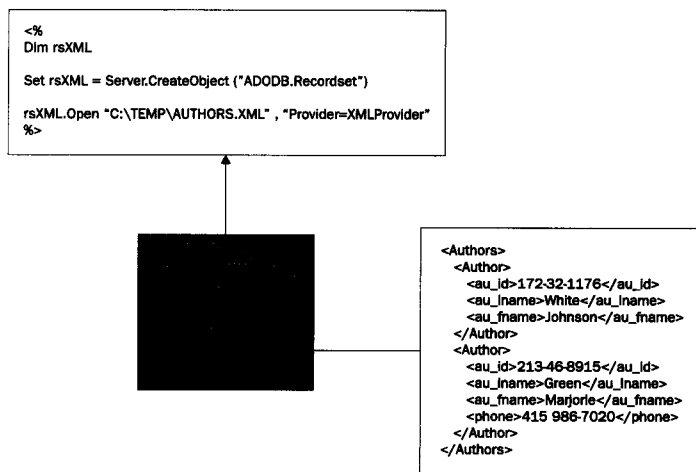


图12-8 创建一个OLE DB简易提供者

让我们来看一下怎样实现这两个类。

(1) XMLConnection类

XMLConnection类提供了应用程序和实际获取数据的类之间的接口。主要任务是创建真正的数据提供者的实例：

```
Private Sub Class_GetDataMember(DataMember As String, Data As Object)

' XMLRecords is the name of the class that actually
' reads in the XML data from the XML file
Dim objCustom As XMLRecords

Set objCustom = New XMLRecords

' Call the method of the data class, passing into it the
' DataMember (field name) requested

objCustom.processFile DataMember

Set Data = objCustom

End Sub
```

(2) XMLRecordset类

第二个类XMLRecordset是真正用来从文本文件中获取数据的类，它是提供者的真正内核。编写自己的简易提供者意味着必须实现 OLE DB 简易提供者的功能，因此应确保我们的类也能做到这一点。首先，声明几个全局变量：

```
Implements OLEDBSimpleProvider

Private mdomColumns As MSXML.IXMLDOMNodeList ' List of columns
Private mdomRows As MSXML.IXMLDOMNodeList ' List of rows
Private mdomXML As MSXML.DOMDocument ' The xml to parse
```

读取XML数据的方法只是创建一个 XML DOM对象并装载数据文件。当然它也设置了几个对象的引用以备将来使用：

```
Public Sub processFile(strCommand As String)

Set mdomXML = New MSXML.DOMDocument

' Load the xml document
mdomXML.Load strCommand

' Set the references to the rows and columns
Set mdomRows = mdomXML.childNodes(0).childNodes
Set mdomColumns = mdomXML.childNodes(0).childNodes

End Sub
```

设置对象的引用也许看起来有些奇怪，但回想一下 XML 数据的结构及在上一章介绍的 DOM，也就没什么奇怪了。DOM 文档有一个根节点，在它的下面是 XML，而且 XML 也有其自己的根节点(在本例中为 Authors)。因此我们用 childNodes 属性来索引树状结构的两层，并指向第一行。每一行都由列组成，所以我们再次用 childNodes 属性索引下一层。

XMLRecordset也实现了简易提供者的几个方法。有三个方法用于获取行数和列数：

```
Private Function OLEDBSimpleProvider_getColumnCount() As Long
```



```

OLEDBSimpleProvider_getColumnCount = mdomColumns.length
End Function

Private Function OLEDBSimpleProvider_getEstimatedRows() As Long

    OLEDBSimpleProvider_getEstimatedRows = mdomRows.length

End Function

Private Function OLEDBSimpleProvider_getRowCount() As Long

    OLEDBSimpleProvider_getRowCount = mdomRows.length

End Function

```

接下来是把数据返回给用户的方法：

```

Private Function OLEDBSimpleProvider_getVariant(ByVal intRow As Long, _
    ByVal intColumn As Long, ByVal format As OSPFORMAT) As Variant

    Dim varValue As Variant

    If intRow = 0 Then
        varValue = mdomColumns(intColumn - 1).nodeName
    Else
        varValue = mdomRows(intRow - 1).childNodes(intColumn - 1). _
            nodeTypedValue
    End If

    OLEDBSimpleProvider_getVariant = varValue

End Function

```

当访问一个特定字段时这个函数将被自动调用，所以这里需要做的是从 XML DOM对象中返回适当的成员。

这就是简易提供者的全部内容。

当然除此之外，还要做一些其他事情，但这里不详细介绍了。全部代码，加上这个提供者和一个测试程序的现成版本都可以在 Wrox的Web站点上找到，其网址是 <http://www.wrox.com>。

在使用一个定制的OLE DB提供者之前必须注册。对于本例中的提供者，我们已经提供了一个.reg文件来帮助完成注册工作。如果创建了自己的提供者，那么就需要自己创建.reg文件(或者手工编辑注册表)。简易提供者的OLE DB文档详细叙述了怎样创建注册表的条目。当然，如果提供者是用Visual Basic编写的，那么还需要将Visual Basic运行期库安装在要使用定制的提供者的系统上。使用Package and Deployment Wizard可以创建所需要的安装文件。

12.3 半结构化的数据

既然UDA可以访问来自多个地方的数据，因此就需要能够访问那些非结构化的数据。例如，在一个记录集中，所有的行都包含相同数目的字段。即使在嵌套多层的记录集中，子记录集也呈现同样的特征。然而，现实中却确实存在着大量的非结构化形式的数据。

半结构化的数据介于结构化数据和二进制数据(例如BLOB类型的数据)两者之间，也就是说它有一定的结构，但并不是每一行都有相同的结构。一个邮件系统就是一个很好的例子。

考虑图 12-9 所示的结构。

这里存在着某种形式的结构。Personal Folders 包含其他的文件夹，因此可以把它看作是一个记录集。但是每一个子文件夹却存储着不同类型的信息，所以它们也就有不同的属性。如果我们想用记录集来表示它，实现起来的确很复杂。如果记录集中一行的字段与上一行不同怎么办？

解决的方法就是使用半结构化数据，因为它就是被特别设计用来处理这种情况的。可以创建一个包含所有通用属性的记录集，然后再创建一些其他对象来表示那些特殊的属性。

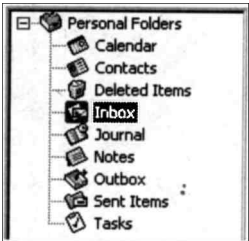


图12-9 邮件系统示意图

12.3.1 Record对象

Record对象用来处理半结构化数据的存储。当考虑这种类型的数据存储时，可以考虑一下带有节点和节点集合的树状结构，因为这很容易映射到 ADO对象上：

- 节点作为 Record建模。因此每个节点或文件夹就是一条记录。记录的属性是节点的那些独特的属性，这些属性构成了记录的字段。
- 节点集合作为 Recordset建模。因此像文件夹集合这样的项目就成为记录集。尽管每个单独的文件夹也许不会和其他文件夹一样，但它们共享一系列通用的属性，比如名字、最近访问的时间等等。这些通用的属性就成为记录集的字段。

因此，在图 12-9 所示的邮件系统中，Personal Folders是一条记录，因为它在树结构中是一个独特的节点。然而，它确实包含子节点，所以它有子代，那些子代构成了一个集合，所以可以把它们看作是一个记录集。在这个记录集中的每一行都可能会指向另一个节点，因此每一行可以是一条记录。这是需要的，因为 Contacts文件夹相对于Calendar和Inbox文件夹而言有一套完全不同的属性。任何不是节点的项都被作为记录集的行来处理。

在邮箱的例子中，可能会有一个 10 行的记录集，很有可能如表 12-2 所示。

表12-2 记录集内容

Name	Description	Post Item As
Calendar	Calendar Comment	Appointment
Contacts	Contacts Comment	Contact
Deleted Items	Deleted Items folder	Post
Inbox	Inbox folder	Post
Journal	Journal Comment	Journal entry
Notes	Notes Comment	Note
Outbox	Outbox folder	Post
Sent Items	Sent Items folder	Post
Tasks	Tasks Comment	Task

每一行都有一个相关联的记录，它代表这一行的独特属性。

12.3.2 Internet 发布

半结构化数据的另一个新特性是 Internet发布。ADO 2.1 版本通过装载 OLE DB提供者提供 Internet发布，但其功能有限。现在已经出现了具有完全功能的 ADO 2.5 版本，这令人相当兴奋。

Internet发布允许通过 ADO来访问网络资源，这意味着可以建立定制的应用程序来管理 Web 站点。这有什么优点呢？这就是如果你已懂得 ADO，那么就不需要再学习什么新东西了（当然除了新的对象），因此可以运用已有的技巧。

为了搞得更明白一些，我们假设有如图 12-10所示的虚拟 Web 目录。



图12-10 虚拟Web目录

这只是一个普通的虚拟目录，没有为使用 Internet 发布提供者而建立什么特别的东西。

1. Record和Recordset对象

许多人一直都分辨不清 Record对象和Recordset对象的区别。让我们来看一下图 12-11所示的示例Web 站点。

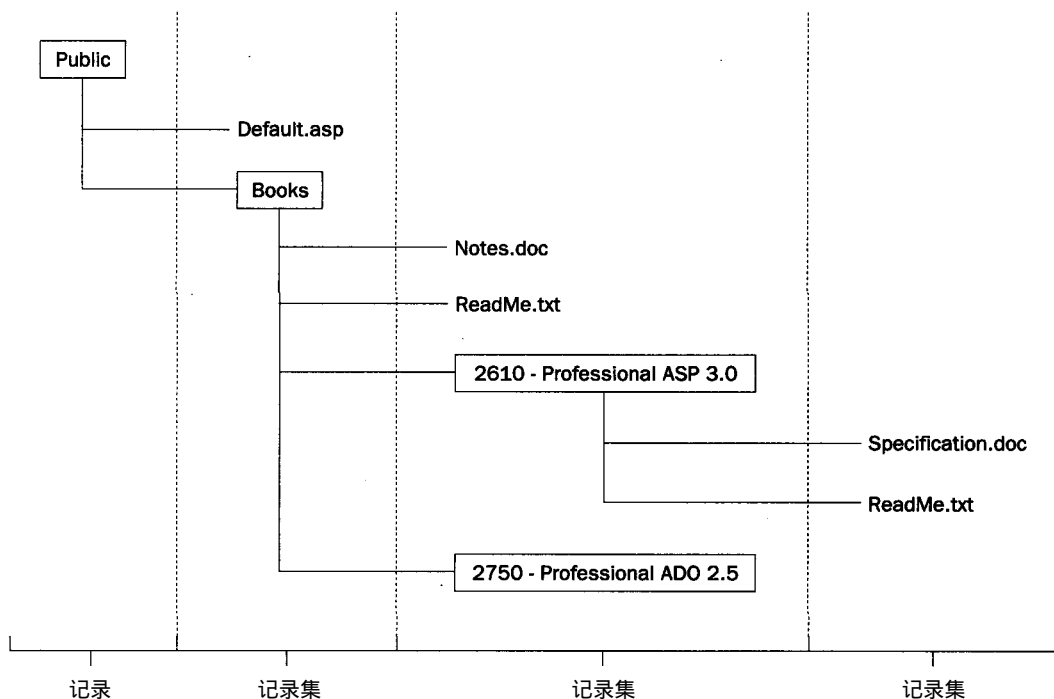


图12-11 Web站点示意图

在结构的顶端是 Record 对象，表示树的一个节点。这个节点有子节点（文件夹和文件），他们被建模为一个记录集，记录集的每一行要么是一个文件，要么是一个文件夹。如果它是一个文件夹，那么就表示树中还有另外一个节点，所以也就有一个记录。一开始就要理解这种概念有一点困难，但很容易记忆：

- 文件和文件夹的集合是一个记录集。
- 单独的文件或文件夹是一条记录。

(1) 打开一个 URL

用 Record 对象打开一个 URL 是很容易的，只需要使用 Record 对象的 Open 方法就可以实现：

```
Dim recNode

Set recNode = Server.CreateObject("ADODB.Record")
```

```
recNode.Open "public", "URL=http://localhost/"
```

以上代码打开了位于 URL //localhost 下的 Public 文件夹。

需要注意的是：这条语句的第一个参数确定了文件或文件夹，第二个参数则确定了连接字符串，即 URL。对我们的操作而言，这实质上代表了范围或根。因此进一步的操作就可以在这个根目录下的任何地方执行了。让我们再来看一下下面这行代码：

```
recNode.Open "", "URL=http://localhost/public"
```

这里第一个参数为空，我们想要打开的文件夹已被移到了连接字符串中。既然第一个参数为空，而打开了第二个参数中的文件夹，那么应该说它的效果和前面的例子是一样的。但是，随后的操作却受到了限制，此时所有的操作只能在 Public 目录下进行。

所以，虽然这两种方法可以产生同样的初始结果，但它们对随后的操作有不同的影响。

Open 方法还有其他几个参数，但这里就不详细叙述了。要获得更详细的说明，可参阅 ADO 文档。

(2) 一条记录的字段

为了看得更清楚一些，我们介绍一个简单的例子。用一个 Record 对象打开一个 URL，然后查看记录的字段：

```
<!-- #INCLUDE FILE="../../Include/Connection.asp" -->
```

```
<%
```

```
Dim recNode
```

```
Dim fldNode
```

```
Set recNode = Server.CreateObject ("ADODB.Record")
```

```
' Open the URL
```

```
recNode.Open "", "URL=http://localhost/public/"
```

```
' Loop through the fields, showing them in a table
```

```
Response.Write "<TABLE BORDER='1'>"
```

```
For Each fldNode In recNode.Fields
```

```
    Response.Write "<TR><TD>" & fldNode.Name & "</TD><TD>" & _  
                    fldNode.Value & "</TD></TR>"
```

```
Next
```

```
Response.Write "</TABLE>"
```

```
recNode.Close
```

```
Set recNode = Nothing
```

```
%>
```

以上代码给出图 12-12所示的结果。

Address http://kanga/proasp/RecordFields.asp	
The Fields of a Record for a Directory	
RESOURCE_PARSENAME	public
RESOURCE_PARENTNAME	http://localhost
RESOURCE_ABSOLUTEPARSENAME	http://localhost/public
RESOURCE_ISHIDDEN	False
RESOURCE_ISREADONLY	
RESOURCE_CONTENTTYPE	
RESOURCE_CONTENTCLASS	application/octet-stream

图12-12 目录记录的字段

在这里，一个目录的属性映射到字段。那么，如果把以上代码中的 Open命令行改成下面的语句会发生什么情况呢？

```
recNode.Open "readme.txt", "URL=http://localhost/public/"
```

它打开的是一个文件而不是一个目录。结果如图 12-13所示。

Address http://kanga/proasp/RecordFieldsFile.asp	
The Fields of a Record for a File	
RESOURCE_PARSENAME	readme.txt
RESOURCE_PARENTNAME	http://localhost/public
RESOURCE_ABSOLUTEPARSENAME	http://localhost/public/readme.txt
RESOURCE_ISHIDDEN	False
RESOURCE_ISREADONLY	
RESOURCE_CONTENTTYPE	
RESOURCE_CONTENTCLASS	text/plain

图12-13 文件记录的字段

显示的是相同的字段集，但它们的内容却不同。

表12-3列出了一条记录的字段集。数据类型的描述参见附录 F，它们是数据类型常量，可在DataTypeEnum下找到。

表12-3 记录的字段列表

字 段	类 型	说 明
RESOURCE_PARSENAME	adVarChar	资源的URL
RESOURCE_PARENTNAME	adVarChar	父资源的URL
RESOURCE_ABSOLUTEPARSENAME	adVarChar	绝对URL地址，包含路径
RESOURCE_ISHIDDEN	adBoolean	指出资源是否隐藏
RESOURCE_ISREADONLY	adBoolean	指出资源是否只读
RESOURCE_CONTENTTYPE	adVarChar	资源的MIME类型
RESOURCE_CONTENTCLASS	adVarChar	可能的资源用途
RESOURCE_CONTENTLANGUAGE	adVarChar	资源语言
RESOURCE_CREATIONTIME	adFileTime	创建资源的时间
RESOURCE_LASTACcesSTIME	adFileTime	资源最近被访问的时间
RESOURCE_LASTWRITETIME	adFileTime	资源更新的时间
RESOURCE_STREAMSIZE	adUnsignedBigInt	缺省的流的大小

(续)

字 段	类 型	说 明
RESOURCE_ISCOLLECTION	adBoolean	指出资源是否是一个集合，即有没有子代
RESOURCE_ISSTRUCTUREDDOCUMNET	adBoolean	指出资源是否为一结构化文档，比如一个Word文档
RESOURCE_DOCUMENT	adVarChar	文件夹缺省文档的URL
RESOURCE_DISPLAYNAME	adVarChar	资源的显示名称
RESOURCE_ISROOT	adBoolean	指出资源是否为一个集合的根
RESOURCE_ISMARKEDFOROFFLINE	adBoolean	指出资源是否被标记为离线使用

应该注意RESOURCE_CONTENTTYPE和RESOURCE_CONTENTCLASS的值可能相反。在本书编写时仍是这种情况，也许会在后面的版本中得到纠正。

(3) DAV字段

还有一些以DAV开始的字段，它们用于DAV中(Distributed Authoring and Versioning)，它们以表12-4所示的方式映射到现有的字段上。

表12-4 现有的字段与DAV字段的映射关系

现有的字段	DAV字段
RESOURCE_PARSENAME	DAV: lastpathsegment
RESOURCE_PARENTNAME	DAV: parentname
RESOURCE_ABSOLUTEPARSENAME	DAV: href
RESOURCE_ISHIDDEN	DAV: ishidden
RESOURCE_ISREADONLY	DAV: isreadonly
RESOURCE_CONTENTTYPE	DAV: getcontenttype
RESOURCE_CONTENTCLASS	DAV: getcontentclass
RESOURCE_CONTENTLANGUAGE	DAV: getcontentlanguage
RESOURCE_CREATIONTIME	DAV: creationtime
RESOURCE_LASTACCESSTIME	DAV: lastaccessed
RESOURCE_LASTWRITETIME	DAV: getlastmodified
RESOURCE_STREAMSIZE	DAV: getcontentlength
RESOURCE_ISCOLLECTION	DAV: iscollection
RESOURCE_ISSTRUCTUREDDOCUMNET	DAV: isstructureddocument
DEFAULT_DOCUMENT	DAV: defaultdocument
RESOURCE_DISPLAYNAME	DAV: displayname
RESOURCE_ISROOT	DAV: isroot
RESOURCE_ISMARKEDFOROFFLINE	DAV: getetag

有两套属性的原因是因为 Internet发布建立在名为 WebDAV的协议上，并且 IIS 5.0是一个 WebDAV服务器。因此可以用任何 WebDAV客户访问 IIS 5.0，所以它必须支持一套正规的 DAV 属性。

我们将在本章后面详细讨论基于 DAV的Web。

(4) 遍历目录

如果想遍历目录，需要获得一份 Web站点上的所有文件和文件夹的列表。例如，你可能正在建立一个允许自己管理自己的网站的基于前端的 Web。对大多数站点而言，可能仅有一

个或两个目录(例如广告、电子商务等等),但是现在可以把一个Web站点作为一个文档库来使用。在这种情况下,它们就可能有一个相当深的树状结构以映射到文档的组织。我们的示例站点就像这样。

因为这个网站有好几个层次,所以可以使用一个递归过程来定位每个文件和文件夹。

下面我们来看一些具体实现它的代码:

```
<%
    Dim intLevel
    Dim recRoot
    Dim conIPP

    Set recRoot = Server.CreateObject ("ADODB.Record")

    recRoot.Open "", "URL=http://localhost/public"

    TraverseTree recRoot

    recRoot.Close
    Set recRoot = Nothing
%>

Sub TraverseTree(recNode)

    Dim rsChildren
    Dim recChildNode

    ' Display the node name
    Response.Write FormatItem(recNode("RESOURCE_PARSENAME")) & "<BR>"

    ' Get any child nodes
    Set rsChildren = recNode.GetChildren

    ' Increase the indenting level
    intLevel = intLevel + 1

    ' Loop through the children
    While Not rsChildren.EOF

        ' Does this child contain other children
        If rsChildren("RESOURCE_ISCOLLECTION") Then
            ' Create a new child node
            Set recChildNode = Server.CreateObject("ADODB.Record")

            ' Open the new child node
            recChildNode.Open rsChildren

            ' Traverse this child
            TraverseTree recChildNode

            ' Close up
            recChildNode.Close
            Set recChildNode = Nothing
        Else
            Response.Write FormatItem(rsChildren("RESOURCE_PARSENAME")) & "<BR>"
        End If

        rsChildren.MoveNext
    Wend

    ' Decrease the indenting level
    intLevel = intLevel - 1
```

```

' Close the child recordset
rsChildren.Close
Set rsChildren = Nothing

End Sub

Function FormatItem(strItem)

    Dim strSpaces

    strSpaces = Space(intLevel * 2)
    strSpaces = Replace (strSpaces, " ", "&nbsp;&nbsp;&nbsp;")

    FormatItem = strSpaces & Replace(strItem, "%20", " ")

End Function

```

上面的代码得到如图 12-14 所示的输出。

下面来详细分析以上代码。该代码首先创建了一个 Record 对象，它是树结构的根：

```
Set recRoot = Server.CreateObject ("ADODB.Record")
```

现在打开这个 Record 对象。我们使用 Localhost 作为连接，连接的是 Localhost 上的文件夹 Public：

```

' Open the record based on the root node
recRoot.Open "", "URL=http://localhost/public"

```

然后，调用一个程序来遍历这个树结构：

```

' Call the traversal routine
TraverseTree recRoot

recRoot.Close
Set recRoot = Nothing

```

我们使用了一个独立的过程来遍历这个树结构，这是因为树结构在本质上是递归的（即，文件夹可以包含其他的文件夹）。因此，这个遍历函数接受一个记录作为它的起始点：

```
Sub TraverseTree(recNode)
```

接下来，声明两个变量来存放这个记录的子代。记录集将为每个子代存放一行，记录将容纳一个文件夹：

```

Dim rsChildren
Dim recChildNode

```

现在就可以显示记录的名称了。别忘了记录的属性是怎样被映射到 Fields 集合的，RESOURCE_PARSENAME 是条目的名字。

```

' Display the node name
Response.Write FormatItem(recNode("RESOURCE_PARSENAME")) & "<BR>"

```

接下来需要检查该节点是否有子节点（文件夹或文件），因此调用记录的 GetChildren 方法，返回一个包含所有子节点的记录集：

```

' Get any child nodes
Set rsChildren = recNode.GetChildren

```

增加缩进的层次，这纯粹是为了输出的格式化：

Using URLs in ADO

```

public
Books
  2750 - Professional ADO 2.5
  2610 - Professional ASP 3.0
    Specification.doc
    ReadMe.txt
    ReadMe.txt
    Notes.doc
    default.asp

```

图12-14 示例的输出结果

```
' Increase the indenting level
intLevel = intLevel + 1
```

现在就可以遍历任何子节点了：

```
' Loop through the children
While Not rsChildren.EOF
```

接下来需要检查该子节点是否还包含其他节点，也就是说，它自身是否为文件夹。为此，检测RESOURCE_ISCOLLECTION字段，因为一个包含其他节点的节点是一个集合：

```
' Does this child contain other children
If rsChildren("RESOURCE_ISCOLLECTION") Then
```

如果该节点是一个集合，那么使用当前记录指针作为数据源创建一个新的 Record对象，并打开它。这样就有了一个新节点，因此又可以把该新节点作为当前的根节点，并再次调用 TraverseTree例程。这是一种递归过程，因为对每一个节点(文件夹)都做一次遍历：

```
' Create a new child node
Set recChildNode = Server.CreateObject("ADODB.Record")

' Open the new child node
recChildNode.Open rsChildren

' Traverse this child
TraverseTree recChildNode

' Close up
recChildNode.Close
Set recChildNode = Nothing
```

如果记录不是一个集合，则只需显示它的名字：

```
Else
Response.Write FormatItem(rsChildren("RESOURCE_PARSENAME")) & "<BR>"
End If
```

接下来就可移向这个记录集的下一条记录：

```
rsChildren.MoveNext
Wend
```

然后递减树的层次，并关闭这个记录集：

```
' Decrease the indenting level
intLevel = intLevel - 1

' Close the child recordset
rsChildren.Close
Set rsChildren = Nothing
```

```
End Sub
```

FormatItems函数仅是在得到的字符串的起始处加入一些非空空格，然后把编码空格(%20)转化为空格字符。这只是使条目更加易读：

```
Function FormatItem(strItem)

Dim strSpaces

strSpaces = Space(intLevel * 2)
```

```
strSpaces = Replace (strSpaces, " ", "&nbsp;&nbsp;&nbsp;")

FormatItem = strSpaces & Replace(strItem, "%20", " ")

End Function
```

这就是全部过程。如果记住了以下这些要点，理解起来就会变得相当容易：

- 一个记录可能包含子记录，因此可用 GetChildren方法来返回一个包含那些子记录的记录集。
- 记录集中的一行可能是一个集合项，所以它也可能是一个记录的源。

以上内容似乎有些繁琐，但只要能理解它，总体上看还是比较简单的。

2. 管理资源

记录不仅可以用来访问 Web资源，而且可以用来管理这些资源，使用的方法如下：

- CopyRecord：从一个位置复制一条记录或一个 URL到另一个位置。
- MoveRecord：从一个位置移动一条记录或一个 URL到另一个位置。
- DeleteRecord：删除一条记录或一个 URL。

你会发现，仅用这三个方法就可以轻松地建立一个简单的远程管理 Web站点的系统。下面的程序显示了它们是怎样工作的：

```
Dim recFile

Set recFile = Server.CreateObject("ADODB.Record")

recFile.Open "ReadMe.txt", "URL="http://localhost/public/Books"

recFile.CopyRecord "", "http://webdev.wrox.co.uk/public/NewReadMe.txt"
```

尽管上面的例子只用了一个源和一个目标参数来调用 CopyRecord方法，但它也可以使用表12-5所示的几个参数。

表12-5 CopyRecord方法的参数

参 数	说 明
Source	源文件的URL。如果不提供，或者为空字符串，那么 Record对象被作为源
Destination	目标文件的URL
UserName	用于连接到目标资源的用户名
Password	用于连接到目标资源的口令
Options	下面的值之一： adCopyOverWrite，覆盖现有的文件或目录 adCopyNoneRecursive，只拷贝一个目录，不包括子目录。 adCopyAllowDataLoss，允许提供者远程操作使用下载/上载类型的拷贝
Async	True表明是异步操作，False则表明是同步操作

由于提供了用户名和密码，你可以连接到安全的远程的 Web站点。

在使用adCopyOverWrite选项时，必须非常小心，因为该操作不同于一般的文件拷贝，它将覆盖目标。如果拷贝一个，并指定了一个目录作为目标，那么这个目录将会被该文件覆盖。

MoveRecord方法的使用情况类似：

```
Dim recFile
```

```
Set recFile = Server.CreateObject("ADODB.Record")

recFile.Open "ReadMe.txt", "URL="http://localhost/public/Books"

recFile.MoveRecord "", "http://webdev.wrox.co.uk/public/NewReadMe.txt"
```

它的参数除了Options参数以外，与CopyRecord方法的参数是相同的。它的Options参数可以是以下值：

- adMoveOverWrite，覆盖已有的文件或目录。
- adMoveDontUpdateLinks，确保文件上的超级文本链接不被更新。缺省的状态是只要提供者具有此功能则更新链接。
- adMoveAllowEmulation，如果MoveRecord方法失败，使用下载/上传和删除模拟这次移动。

资源的安全性

对于那些看上去存在着很大安全漏洞的地方，不要感到惊慌。要认识到以下两点：

- Internet发布是IIS 5.0的一个组成部分。IIS 4的网上发布功能是非常有限的，它由FrontPage Server Extensions来提供。IIS 5.0仍然支持这些功能，但同时也支持新的DAV扩展。
- 通过OLE DB提供者访问的用于Internet发布的资源是标准的Web资源，控制Web资源访问的方式与其他Web资源是相同的。

如果确定了自己的站点没有写或删除的权限，那么其他人就不会把你的文件搞得一团糟。如果确实需要这些操作，那么可以授权给特定的用户，MoveRecord等方法需要得到访问权限，这些都可作为方法调用的一部分来提供。

因此，网站仍然是安全的。如果它还不具有安全性，那么现在我们应该使之具有安全性。

3. Stream对象

迄今为止，关于Web资源我们一直没有提及怎样使用Stream对象。在XML的章节中，读者已经看到了Stream对象的一些使用情况，但也可从Web资源获取这些数据。我们可以使用一个A标记，并放进文件的URL，这样一旦URL被单击，该文件就会被加载。

但是，如果要控制Web上的资源，而不是直接将其加载到浏览器中，那又该怎么办呢？也许你想把它显示在页面内，看上去也许可能像图12-15所示。

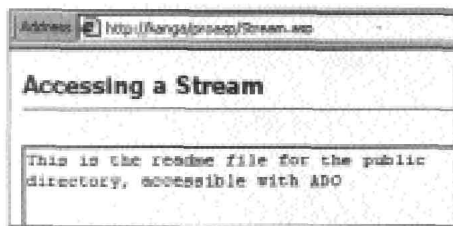


图12-15 访问Stream对象示意窗口

同样，这也很容易做到。与以前一样，我们先包含连接的细节，然后建立一个文本域以供显示文件的内容：

```
<!--#INCLUDE FILE="../../Include/Connection.asp" -->

<TEXTAREA NAME="txtStreamContents" STYLE="width:100%; height:50%">
下面是读取文件的ASP代码。首先创建一个Stream对象：

<%
Dim stmData

Set stmData = Server.CreateObject ("ADODB.Stream")
```

然后打开这个Stream对象，同时指定一个文件作为Stream对象的源文件：

```
stmData.Open "URL=http://localhost/public/readme.txt", _
             adModeRead, adOpenStreamFromURL
stmData.Charset = "ascii"
```

Stream对象的Open方法中的参数指定了要打开的文件的URL、文件的访问方式以及从哪里打开Stream对象。如果Stream的源(第一个参数)是一个Record对象，那么后一个参数就可以是adOpenStreamFromRecord。

一旦打开了Stream对象，就可以用ReadText方法读取其内容，并简单地把它显示出来：

```
Response.Write stmData.ReadText
```

最后，清理变量并关闭文本域：

```
stmData.Close
Set stmData = Nothing
%>
</TEXTAREA>
```

必须承认这并不是一个特别令人激动的例子，也可以通过别的方法来实现它。但是，考虑一下围绕着文本域增加一个FORM对象，以及增加一对命令按钮，如图12-16所示。

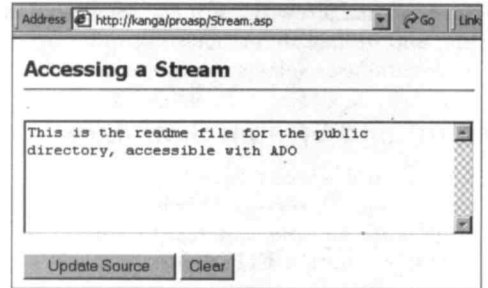


图12-16 带有命令按钮的访问Stream对象示意窗口

现在就可以更新这个文件了，这确实很令人兴奋。这个Update Source按钮只是调用了ASP文件(StreamUpdate.asp)，其代码也非常简单：

```
<!-- #INCLUDE FILE="../../Include/Connection.asp" -->
<%
    Dim stmData

    Set stmData = Server.CreateObject ("ADODB.Stream")
```

创建了Stream对象后，打开文件(此时使用读/写权限)：

```
With stmData
    .Open "URL=http://localhost/public/readme.txt", _
        adModeReadWrite, adOpenStreamFromURL
```

接下来，把Stream对象的当前位置移到Stream对象的开始，接着设置Stream对象的结尾为当前位置。这样能截短Stream对象，清除已存在的内容：

```
.Position = 0
.SetEOS
```

然后以调用ASP页面的形式从文本域中取得文本，同时使用WriteText方法向Stream对象中写入内容：

```
.WriteText Request.Form("txtStreamContents")
```

下面告诉用户发生了什么事：

```
Response.Write "File updated. Current size = " & .Size
```

最后关闭这个Stream对象：

```
.Close
End With
```



```
Set stmData = Nothing  
%>
```

这样仅用这几行代码就创建了两个可以读取和更新文本文件内容的 ASP 页面。虽然这些例子使用的是固定的文件名，但建立一种允许用户选择文件名的窗体也是很容易的。

4. Record和Stream对象小结

我们在前面几节看到的例子表明了管理 Web 服务器上的资源是很容易的。但这也只是冰山的一角，因为它扩展的可能性是很大的。举个例子，作为测试，作者只用 ADO 写了一个 Visual Basic 程序，它执行起来如同普通的 FTP 客户一样。为什么这么做？主要有如下两条原因：

- 要了解能从仅有的几个 ADO 方法中能获得哪些功能。
- 要证实 Internet 发布提供者的 ADO 功能。它建立在 DAV 上，即 HTTP。这意味着可使用 ADO 编写远程管理程序而不必担心打开防火墙。因为它使用 HTTP，它可以轻松地穿过防火墙。

当然，这也意味着必须确保你的 Web 站点是安全的，至少也要降低其潜在的安全风险。

这种技术的真正的核心仅由几个方法处理。可以使用 Record 对象的 CopyRecord 和 MoveRecord 方法来移动文件，同时 Stream 对象提供了访问文件内容的方法。这样就可以轻松地写出简单的远程管理工具了

12.3.3 WebDAV

在前述章节中我们已经数次提到了 DAV 和 WebDAV，但一直没有真正解释过它到底是什么。基于分布式设计和版本 (Distributed Authoring and Versions, DAV) 的 Web 是 Internet 发布提供者涉及的全部内容。WebDAV 是专门为分布式创作 Web 文档而设计的。

目前我们用 FrontPage 和 FrontPage Server Extensions 编辑和管理远程 Web 服务器上的文档，但这是一个专用的解决方案。多年来，W3C 和 IETF 的工作组已经整理出 WebDAV 的具体规范，现已成为大家认可的标准。

WebDAV 是一种基于 HTTP 1.1 扩展的协议规范，因此它与平台是无关的。目前 FrontPage 是一个很好的解决方案，但它也仅是 Microsoft 的解决方案。随着 IIS 5.0 的出现，Microsoft 已完全支持 WebDAV，并成为该工作组中的主体部分。

那么，WebDAV 究竟能为我们做什么呢？如果我们告诉你 Internet 发布提供者是建立在 DAV 之上的，那么你就会发现它的方便之处。你能远程管理文件，允许编辑、拷贝等等。

现在已有好几种包含 WebDAV 功能的 Web 服务器，并且各种客户也是层出不穷。这意味着用户可以在不损失功能的前提下，使用自己最为熟悉的工具。

1. WebDAV 的未来

WebDAV 仍旧是一个发展中的产品，并且目前版本仍在继续更新。这将带来真正意义上的合作开发，允许多次编辑且使用版本来控制资源。最终它可能替代现有的版本控制系统。

正在进行的其他方面的工作也影响了 WebDAV 的发展，更多的细节问题请查看 Web 站点：www.w3c.org。虽然 Microsoft 没有正式声明，但好像一旦 WebDAV 的版本被认可，他们将在未来的某个时候更新 IIS。

12.4 企业数据

在当今的企业中，选择 Microsoft 并不是唯一的策略。除非这家企业决定只使用 Microsoft

工具和产品来实现标准化应用，否则你会看到一个应用程序必须与其他系统交互。许多这样的系统甚至不能在一个通用协议上相互通信。

数据是一个企业最有价值的资产，绝大部分数据被存储在大型数据库中，比如 Oracle和 IBM的DB2，以及非关系数据库中，或者是直接作为一组数据文件存储在文件系统中。小型计算机系统也大量使用于企业，大量的数据一般基于 Digital(现为Compaq)的VAX/VMS或 UNIX系统，并且主要是围绕着四大关系数据库管理系统 (RDBMS)之一来建立的，它们是IBM、Informix、Oracle和Sybase。

12.4.1 Access和SQL Server二者取一

当Microsoft决定进军数据库市场时，它把SQL Server定位于服务于少数用户，即工作组，主要是为了满足小规模到中等规模的数据存储需要。Microsoft把SQL Server作为一种必备的工具来帮助推销操作系统 OS/2。因此Microsoft的工程师同 Sybase的工程师合作将 Sybase的UNIX数据库产品移到了Microsoft的OS/2版本上。

当Microsoft最终将SQL Server定位于Windows NT，并且在SQL Sever 4.2版本之后从Sybase手中接管其发展方向时，SQL Server仍然是一个工作组级的数据库产品。从我的观点看，其原因与其说是因为产品性能，还不如说是因为市场策略的缘故。无论什么原因，自始至终SQL Server一直被认为是一种工作组级的数据库，只有在现在才开始被认为是一种企业级的数据库产品。而今，多处理器支持和较好的可扩展性已使 Microsoft的SQL Server 7.0挤入了企业级数据库市场。

Microsoft进入桌面数据库市场稍晚了一些，但他们同样也使桌面市场的竞争者黯然失色。虽然这个领域将涉及 Access和SQL Server的取舍问题，但也不要完全抛弃 Access。我的意思是说对于ASP数据访问应有可供选择的数据库，而不是在 Windows环境下选择最好的前端数据库工具。我在Informix软件公司工作，这是一家主要的数据库供应商，但是我仍然选择 Microsoft的Access作为客户端工具。

今天的企业一般是围绕着 IBM、Oracle、Informix或Sybase四家公司的的大型数据库系统来建立数据存储。事实上它们都是关系数据库产品。一家企业的大多数数据资产其实并不完全位于关系数据库中，另外在较老的层次或 CODASYL数据库或在文件系统中也存在着。当然，大部分早期系统里的数据已经移到关系数据库管理系统 (Relational Database Management Systems, RDBMS)中了，因此我们集中介绍主要的数据库供应商的产品。

1. Oracle

当Microsoft公司宣布其SQL Server 7.0时，不久Oracle就宣布出\$1 000 000.00对Microsoft SQL Sever 7.0提出挑战。从那以后在这两大软件业巨头之间就开始了不停的竞争。Oracle想挫败Microsoft的COM/DCOM及其系列产品，他们选择了使用 Java和CORBA作为主要战略来实现其分布式和Web结构。

然而，他们实际上却不得不相互妥协，Oracle的确支持Microsoft的DNA、COM/DCOM、OLE DB、ADO和ASP。事实上，Microsoft自己也觉得与Oracle的协同工作能力是如此重要以致于还为Oracle提供了一个强大的Oracle OLE DB提供者，它使Oracle运行在与SQL Server 7.0相同的功能层上。Oracle也为OLE提供了具有ActiveX功能的Oracle对象(OO4O)。OO4O可在任何程序或脚本语言中(包括ASP)使用。

使用Microsoft的OLE DB提供者来连接Oracle数据库是比较容易的，它的功能与使用 SQL Server 7.0处于相同的级别上。然而，实际上，大多数 Oracle工作间仍在使用老版本的 Oracle引擎，比如7.x或更早的版本。这就提出了一个问题，也就是说需要使用一种支持特定的 Oracle引擎的OLE DB ODBC提供者和一个ODBC驱动程序。OLE DB ODBC提供者交付的功能要比本地的Oracle OLE DB提供者少得多，这主要是受所用的特定 ODBC驱动程序的限制。

然而，Oracle提供了最为广泛的多平台支持，但每一种操作平台可能会对 Oracle提出不同的要求。OLE DB ODBC网关是最容易的访问方法，但也可能出现种种不希望有的限制。开发者必须仔细评估每一个折衷的办法。

2. IBM

IBM针对他们的DB 2产品提供了五种不同的代码。IBM基于不同的操作系统和平台，在公司内的不同地方开发了各种不同的 DB 2版本。有传言说IBM正试图为其DB 2关系数据库提供一种通用代码。但有一点可以确定，一致的功能会贯穿 DB 2产品线，这样就能为所有DB 2产品带来功能上的统一。

IBM的战略重点放在开发 Java，以及在他们的数据库产品中集成 XML。通过与其他已经将这些技术转移到IBM平台的供应商的合作，他们对 COM、OLE DB 和 ASP提供了支持。有时这些解决方案是以很高的代价换来的，而且并不是所有提供的东西都能跨越全部 IBM平台。

在后面，我们将看到 Chili!Soft的ASP技术，它已被移植到 IBM的OS/390和 AS/400操作系统以及其他平台上。

据估计世界500家最富有的公司中就有470家使用IBM的大型机环境之一来存储企业数据。由Microsoft的SNA Server提供的SNA OLE DB提供者提供了一个从外部进入DB 2环境的网关。把IBM DB 2与ASP应用程序结合起来的关键在于对 IBM的SQL/CLI标准的有力支持。

ODBC是最通用的SQL/CLI实现。这也常常引起用户的误解，因为通常用户并没把SQL/CLI的API与ODBC联系起来。IBM最终放弃了CLI这个称呼，而用ODBC代替。

自从Microsoft似乎在ODBC领域的开发已到了尽头，IBM就一直就扛着ODBC的大旗。

3. Informix

Informix的战略是成为一个“标准公司”。他们支持所有主要的对象技术：COM/DCOM、CORBA和Java，同时他们也致力于将XML集成进其产品线的各个领域。Informix是唯一一家采用所有标准的公司。

Informix甚至有自己的Visual Basic工具，Visual Basic Data Director。它能帮助开发人员开发一个成功的ASP策略。Informix也支持本地的OLE DB提供者。

Informix的Web站点，www.informix.com，有好几个涉及使用ADO来访问他的产品的Internet白页。

随着Informix收购了Illustra，Informix已将对象相关技术集成到9.x版本的服务器产品中。Illustra是UC Berkeley 对象相关成果的商业化产品，增加了丰富的在其他相关产品中没有的功能。

Informix也实现了最新的SQL-99标准中某些先进的特性，同时由于收购了在数据存储系统中处于领导地位的Red Brick公司，Informix也相应加强了对数据存储的支持。

Cloudscape是一家百分之百的Java SQL数据库公司,已被Informix收购。他主要是通过提供桌面的和易于嵌入的数据库产品来推出其数据库系列产品。Informix的Cloudscape数据库将会提供许多Microsoft的Access所拥有的优点。Cloudscape将被集成到Informix的系列产品中,也将支持其他Informix产品(CORRBA、COM和Java)所支持的全部接口。也有可能包括ASP脚本程序。

Informix为ASP开发提供了一个较好的平台,同时 Informix.数据库产品又是一种工业强度系列的数据库,这是实现某些工业大容量数据库的基础。Informix的数据库分别提供了UNIX和Windows NT环境下的两个版本。

4. Sybase

Sybase对Microsoft早期开发SQL Server是有帮助的,因此难怪Microsoft的SQL Server与Sybase的产品很相似。然而,当Microsoft与Sybase最终决定分别以自己独立的方式开发产品时,他们的产品外观就显得不同了。

Sybase郑重承诺将把Java集成到他们的产品中。Sybase是SQLJ标准制作的最初参与者之一。此外,Sybase的PowerTools工具能直接生成ASP代码,因此Sybase的Web数据库集成是其公司内的一个重要成果。

使用Sybase的工具可以直接支持ASP开发,也可以通过他们的ODBC和OLE DB提供者来支持ASP开发。Sybase对ASP的支持以及PowerTools产品生成ASP代码的能力,使我们看到Sybase在未来将会把ASP的成果捆绑到其产品。

5. 结论

总之,对于ASP应用程序开发而言,来自主要数据库供应商的集成信息不应该是一个问题。所有主要数据库供应商都支持SQL/CLI标准和ODBC规范,同时也可通过第三方支持,直接或间接地提供ODBC和OLE DB 提供者。许多厂商也提供本地的OLE DB提供者,它们具有与Microsoft SQL Server 7.0的OLE DB提供者同样丰富的特性。

12.4.2 SNA Server和传统数据访问

正如前面提及的那样,许多企业数据集中存储在公司的大型机数据中心上,在大型机技术市场上IBM占据了主导地位。据估计目前全世界企业80%的关键数据都存放在IBM大型机和AS/400系统中。

因此,在任何新应用程序中进行信息集成的策略是非常重要的,这些新应用程序需要开发企业的数据资源并将其完美地集成到IBM的大型机技术中,比如SNA(IBM的网络体系结构)。Microsoft较早地认识到了这些问题,提供了一种系统网络体系结构(SNA)Server。信息系统专家依靠IBM的SNA网络将终端连接到大型机,但现在他们必须把这些信息集成到基于Web的应用程序中。

同时,大多数企业也配置了自己独立的PC网络,它们也都是关键的,比如像企业的电子邮件、文件创建、工作组数据库和论坛支持系统。当前,把两个或更多的系统紧密结合起来已成为一个很重要的问题,这样终端用户才能将存储在主机系统上的数据结合到他们的通信、分析和汇报中。因此终端用户使用比较简单的数据集成也就不足为奇了,例如:

- 从一个终端的仿真会话中拷贝和粘贴。
- 从打印报表中再次输入数据。

- 从文本文件转储中导入数据，这种转储必须涉及 EBCDIC到ASCII文件的转换。

EBCDIC和ASCII都是字符编码的标准。IBM的大型机使用EBCDIC(扩展二进制编码的十进制转换码)作为字符编码字符集。与此相反，PC和大部分其他系统使用ASCII(美国标准信息转换码)字符集。所以，主要的问题在于如何实现不同字符编码之间的转换。

IT专业人士已要求提供一种直接将PC系统、主机数据与运行数据库的大型机集成起来的方法。这是多年来一直存在的一个问题，现在由于Internet、Intranet和Extranet应用程序的兴起，这个问题又有了新的内容，这些应用程序需要利用这些关键数据。数据存储的在线分析处理或论坛支持(Online Analytical Processing or discussion support, OLAP)问题也进一步强化了对数据集成的要求。

Microsoft提供了一种在IBM大型机上开发企业数据的简便方法：通过使用Microsoft的SNA Server以及AS400和OS/390的OLE DB本地驱动程序。这就允许用ASP脚本来开发大型机的数据库。

12.5 企业中的ASP

从现在到2003年之间，多数开发和使用的应用程序将主要基于Intranet、Extranet和Internet技术。所以，选择一种将当前企业资产和资源集成到新的基于Web的应用程序中的中间件技术是有意义的。将当前的数据资产迁移到新的操作平台是不实际的，并且在经济上可行性也不高。增加一个额外的能访问资源或对当前企业的技术投资起决定作用的层次将会更有意义。

但不是使用AIX上的Perl、NT上的FrontPage以及IBM OS/390和AS/400上的某些其他东西，可以完全只用ASP：通过使用相同的跨平台开发工具来“粘合”所有操作平台。这确实大大节省和缩短了开发时间和工作量。现在只需编写一次应用程序，而不用考虑使用什么平台，因为ASP开发不仅仅用于Windows NT和IIS。ASP开发所提供的规模比NT解决方案的规模要大得多。

Chili!Soft ASP

Chili!Soft研制了UNIX和其他平台上使用的ASP，它也同样广泛支持各种Web服务器。这使ASP技术看上去不仅仅是一个Microsoft的中心中间件策略。例如，IBM和Chili!Soft已经将ASP引擎移植到AS/400和OS/390操作系统上。这也意味着ASP应用程序同样可以运行在占据企业数据存储80%份额的IBM平台上。ASP对UNIX的支持也将扩充驻留在基于RDBMS系统的UNIX上的企业数据资产，例如Oracle、Informix、DB2、Sybase及其他供应商的产品。

事实上，由于Chili!Soft研制出Chili!Beans，他们已经比Microsoft领先了一步，Chili!Beans技术允许off-the-shelf Java类和Java Beans成为COM ASP组件。如果Chili!Soft在长时间内是成功的，那么他们将能为我们改善ASP应用程序开发的环境。他们的Web站点的URL是：www.chilisoft.com。

1. Chili!Soft的目标是ISP

Chili!Soft产品的市场目标是ISP，以提供更强大的Web托管功能。他们相信这将为他们的技术开辟一个强大的市场，因为ASP已逐渐被大众所接受，同时大多数ISP运行在UNIX或

UNIX的变种上,比如FreeBSD或Linux。然而,我认为企业对同样的系统集成有一定的需要,因此企业中的用户会对Chili!Soft的ASP技术持有坚定的信心。

2. 在非Microsoft平台上使用ASP

正如前面所说的那样,大多数企业是由许多不同的平台和数据存储组成的。要知道把所有的鸡蛋都放进同一个篮子显然也不是一件好事。IT部门会选择不同的供应商分担不同的任务。这样可以在单个供应商的解决方案不能解决问题的情况下有所选择。一名有头脑的IT管理者不会想成为单个供应商的永久顾客。

除了上述观点以外,大多数企业都支持ANSI/ISO SQL数据库语言标准、TCP/IP网络协议标准等工业标准,因此单个供应商特有的方案并没有考虑到一个开放的标准。随着把ASP解决方案移植到其他的平台,ASP有可能成为一个实际的标准。如果不把ASP移植到其他平台或Web服务器上,ASP永远只能算是一个一流的Microsoft解决方案。

3. Chili!Soft ASP和分布式对象技术

分布式应用程序技术已经出现一段时间了,它在操作系统、网络操作系统和数据库引擎的开发中已经使用了一段时间。但只有在现在分布式应用程序才进入企业开发者的“武器库”中。我们需要理解它,并使用它来开发今天和未来的企业应用程序。

分布式技术的发展主要有三个方向:COM/DCOM、Java和CORBA。我们将对每一个技术方向做探讨,以便进一步了解Chili!Soft在各个技术方向的发展状况。

(1) Chili!Soft ASP和COM/DCOM

Microsoft的COM是Windows NT和UNIX上的Chili!Soft ASP的本地对象模型。Chili!Soft ASP 3.0版现在支持DCOM,同时也允许COM对象作为本地COM对象出现,甚至COM对象也可以分布在另一台或好几台机器上。这就要考虑分布与负载平衡问题。

Chili!Soft的UNIX支持也将DCOM模型扩展成支持UNIX平台,因此可以将分布式环境拓宽到具有沉重负荷的机器上。这也表明ASP组件并不一定要驻留在ASP服务器上。它们可以驻留在另外的数据资产和数据收集设备附近。数据收集设备可能是医疗仪器或任何与之同等复杂的设备。这种对分布式策略的广泛支持为我们带来了很大的希望。

(2) Chili!Soft ASP和企业JavaBeans(EJB)

Sun的新的EJB规范被认为是一个功能强大的可替代COM/DCOM的技术。EJB号称是一个功能强大,与平台无关的,并能提供创建基于组件的分布式应用程序的体系结构。

Chili!Soft ASP 3.0版本能够让开发者立即使用EJB Server去处理事务性的ASP组件。EJB Server提供了许多与Microsoft MTS和COM+在COM领域中提供的相同的服务。EJB Server提供了事务管理、生存期管理、对象转储、线程管理以及其他自动管理服务等服务。

这种组合为开发工业强度的、可扩展的、安全的和面向事务的Web应用程序提供了一个卓越的环境。这是对目前仅在Microsoft ASP环境下可获得的功能的扩展。

(3) Chili!Soft ASP和CORBA

通用对象请求代理体系结构(Common Object Request Broker Architecture, CORBA)是承诺通过使用对象来实现真正的分布式应用程序开发的最重要的技术之一,它是由对象管理组(OMG)开发的。

OMG正努力想通过其较为普遍的标准规范(CORBA/IIOP、对象服务、Internet工具和域接口规范),把CORBA建立成为到处都有的中间件。随着Chili!Beans技术与Java IDL的结合,

ASP开发人员现在可以使用 CORBA作为分布式应用程序开发的选择手段之一。CORBA对象可以通过 Chili!Beans和Java 2.0来访问。

有了 Chili!Soft的ASP方法，我们相信未来的 ASP还要有显著的扩展。服务于更多基于标准和规范的非 Microsoft平台的供应商，会使 ASP成为实际的分布式 Web应用程序开发脚本标准。

12.6 小结

本章探讨了 ADO大多数人倾向于支持关系数据库。但数据正以不同的方式被广泛利用，因此，在讨论有关数据的问题时，我们需要拓宽视野，这就是 UDA策略所涉及的主要内容。

本章主要讨论了以下几个问题：

- 如何从ADO来访问众多不同的数据存储。
- 半结构化数据的定义，如何使用新的 ADO对象来访问它。
- Internet发布的定义，如何访问远程资源。
- 简要地概述了非 Microsoft企业。

这一章所讲述的内容仅是某些更复杂的内容的开始。WebDAV等开发将会对我们管理 Web数据的方式产生重大的影响，同时也证明它是协作开发的一个主要的转折点。目前众多的网页编辑工具似乎都着眼于单个作者，当我们进入工作组进行开发或多人编辑时，会出现大量需要解决的问题。

随着在 Web上发布的数据量逐渐增加，研究一种便于编辑、查看和发布这些信息的方法是明智的(或者说是势在必行)。也就是说 WebDAV领域正变得越来越大。

同样，随着 Chili!Soft这样的公司扩展了对非 Microsoft平台的 ASP支持，ASP将迅速成为企业中所有形式的基于 Web应用程序开发的一个标准。

现在结束了对数据访问内容的讨论，从下一章起将介绍怎样在 ASP应用程序中使用功能强大的组件。