

## 第8章 ADO 基础

在本书前7章中，已经讲述了ASP的有关内容，以及ASP如何为Web站点带来动态的内容。已经见到其脚本程序允许自定义Web页面，使我们能够构建功能更为强大的ASP页面。

现在，将研究ASP和数据的集成。虽然对于网页中的脚本数量并无任何限制，但如果没有某种形式的数据，很快就会进入一个死胡同。数据构成了Web站点的实际内容，或者指出了如何设置Web站点，因此总的说来数据是非常重要的。如果围绕数据存储建立Web站点，改变Web站点时只需要改变相应的数据即可。

ActiveX数据对象(ADO)是允许用户与数据存储进行交互的组件。这意味着只要基于某些数据就可建立一个网页，或一种完全交互的电子商务系统。不论那种方式，都是ADO使我们能与数据进行通信。我们将讨论从数据存储获取和传送数据的主要内容，以及得到数据后的数据处理方法。

首先研究什么是ADO及其所包括的组件，然后讨论如何访问数据存储。在下一章，将进一步学习ADO更先进的一些特性，如命令、存储过程和优化应用程序的一些操作技巧。下一步研究Web服务器和浏览器之间的交互过程，以及数据处理过程。然后研究数据存取领域中极具潜力的XML。XML是什么？如何使用？因为XML代表着未来发展的一种趋势，我们将介绍微软关于通用数据存取的构想。在这个构想中，数据不只是从数据库中获得的。最后，看一下标准的微软数据库(如Access与SQL Server)以及在其中如何使用ADO。

本章从ADO开始，主要内容有：

- 研究ADO如何与数据进行交互。
- 了解ADO的组件。
- 如何与数据存储连接和创建数据集。
- 如何处理和修改数据。
- 如何处理ADO错误。

### 8.1 ADO的定义

ADO是一个相当简单的思想，一种让你仅用一种方式去访问数据的思想。ADO不算一个新思想，仅是采用现有的数据库访问技术，并将其融合而形成的一种适应现在和未来需要的新东西。适应未来的需求是一件十分重要的事。许多其他的技术，比如DAO和ODBC，在一些应用程序的开发过程中是可以接受的，然而随着Internet的兴起也出现了其自身的一些问题。

在许多情况下，传统的数据存取方法看上去能解决一些关于两层客户/服务器系统的问题，但要求与数据之间要保持一种永久性的连接，并要提供强大的功能，比如快速响应的查询、数据容易修改等。在Internet领域，现在必须考虑到Web无状态性本质，和潜在的众多可以访问Web站点的用户。要与数据建立永久的连接是不现实的，因此必须在设计应用程序时考虑这些因素。

那么，OLE DB和ADO确切地讲到底是什么？让我们与一些已有的数据存取技术做比较

后再来回答这个问题。如果读者曾经接触过数据库编程,或许比较熟悉 ODBC和RDO。开放数据库连接(ODBC)是允许访问关系数据库(比如Access和SQL Server)的应用程序编程接口(API)。正因为是一个API,许多程序员,特别是 Visual Basic领域的程序员,发现它使用起来很复杂。远程数据对象(RDO)是位于ODBC上层的ActiveX对象,可以提供ODBC的所有功能,并且使用起来比较简单。

可以将OLE DB等同于ODBC,ADO等同于RDO。

OLE DB是应用程序与数据源交互的一种基本技术。

这相当复杂,确实也只有C和C++程序员能够使用。正如ADO的名字所暗示的,它是易于访问OLE DB功能的ActiveX对象。

你或许发现术语 ActiveX与COM对象经常混用。对于ASP程序员来说它们并没有本质上的区别,因为两者都基于COM系统结构,只不过ActiveX是组件的一个跨平台标准,而COM是Windows专有的。

虽然微软已经引入了一种新的存取数据的技术,但并没有立即取消旧的技术。ODBC工作起来仍然很有效,并同OLE DB和ADO紧密地一起工作着。事实上,ODBC并不只是微软的产品,也受到国际组织的控制。并且由于广泛的使用,ODBC也不会突然消亡。隐藏在OLE DB背后的思想不是摒弃现有的技术,而是不断地改进它们。

### 8.1.1 OLE DB和ADO的体系结构

前面已经给出了OLE DB与ADO在一些主要方面的简要解释。图8-1显示了这两项技术与应用程序和数据存储相互关系。

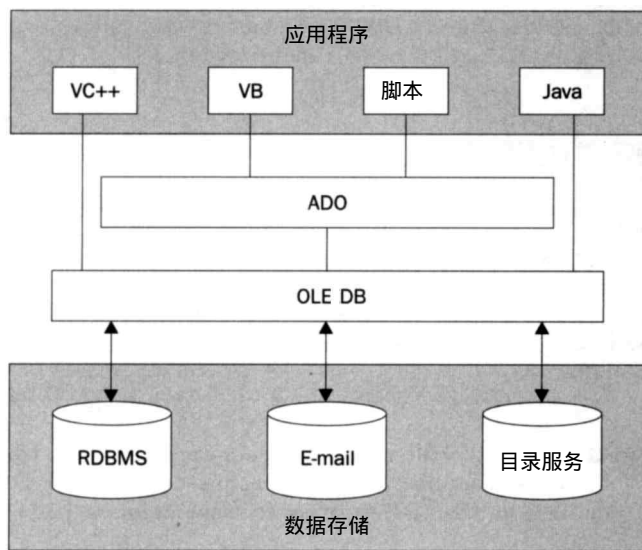


图8-1 OLE DB、ADO与应用程序和数据存储的关系

从图8-1中可以看出整体思路。图的顶端是应用程序(Web或常规的应用程序,这是无关紧要的),下面是提供对数据的访问的ADO和/或OLE DB。ADO和OLE DB两者兼有是因为OLE

DB是一项基本技术。然而，OLE DB并不适用于所有语言，所以ADO位于OLE DB的上层，为那些不能直接访问OLE DB的语言(如Visual Basic和脚本语言)提供编程接口。ADO提供了比OLE DB更容易的编程接口，因此即使那些可以直接使用OLE DB的编程语言，如C++或Java，也可使用ADO以简化对数据的访问。

图8-1显示的是微软的编程语言，而ADO是一个COM组件，因此可用于任何与COM兼容的编程语言，比如Delphi或支持Active Scripting接口的脚本语言。所以，虽然ADO与平台有关，但与开发的语言是无关的。当然，对于ASP主要使用VBScript和JScript，在组件中使用ADO时，有一些Visual Basic代码。

现在知道了OLE DB和ADO允许访问数据，可是为什么需要它们？老方法出问题了吗？这里有两个主要原因：

首先，OLE DB和ADO是用来访问数据存储的。注意这里指“数据存储”而不是“数据库”。尽管数据库仍旧是数据存储最为广泛的形式，但并不一定含有全部的数据。一些消息系统，如Microsoft Exchange Server，也普遍地用于存储数据。目录服务(Directory Service)正开始在初露端倪，它们包含着有关用户、机器等的信息；Web服务器中存有大量的信息。可以继续罗列下去，很明显需要一种能访问所有这些不同类型数据的方法。

其次，源于Internet应用程序的兴起与Web的状态性本质。过去的访问数据的方法主要考虑与数据存储保持永久连接的情况下处理数据。而OLE DB和ADO正是为解决这个问题而设计的，提供断开连接的记录集，我们将会在后面看到有关这方面的内容。

### 8.1.2 消费者与提供者

ADO系统结构图展示了ADO是如何在应用程序和真实数据存储之间发挥作用的。在微软的文献中，会看到两个易懂的术语：消费者(Consumer)和提供者(Provider)，但搞清它们的确切定义至关重要。

提供者是提供数据的物体，消费者是使用(消耗)这些数据的物体。

在编程中，经常会发现应用程序是数据的消费者。但提供者呢？一般是数据存储，并且由于OLE DB被设计成用于与不同的数据存储对话，因此对于每一个独特类型的数据存储都有一个OLE DB提供者。

这种单独提供者的思想并不新，但使编程变得容易了。编写程序与ADO或OLE DB对话，OLE DB再与提供者对话。这意味着只需学会一套访问数据的方法，无论数据如何存储，在某些情况下确实可以完全不改变任何代码而只更换提供者。这就是ADO和OLE DB真正优越的地方，为数据存储提供了一套常用的编程接口。

要连接到数据存储，必须使用OLE DB提供者。提供给ADO 2.5的初始设置为：

- Jet OLE DB 4.0：用于微软Access数据库。
- DTS Packages：用于SQL Server的数据转换服务(Data Transformation Services)。
- Internet Publishing：用于访问Web服务器。
- Indexing Services：用于索引目录(Index Catalogs)。
- Site Server Search：用于站点服务器查找目录。
- ODBC Drivers：用于ODBC数据源。

- OLAP Services：用于微软 OLAP 服务器。
- Oracle：用于 Oracle 数据库。
- SQL Server：用于微软 SQL Server 数据库。
- Simple Provider：用于简单的文本文件。
- MSDataShape：用于层次数据。
- Microsoft Directory Services：用于 Windows 2000 的目录服务。
- DTS Flat File：用于 SQL Server 的数据转换服务的平面文件管理。

这只是微软提供的初始列表，并取决于安装在服务器上的服务及软件。以 Oracle 数据提供者为例，要求在客户机上安装 Oracle 的客户端软件。

可以从别的制造商那里获得 OLE DB 提供者，用于其他数据存储。甚至还可以编写自己的提供者。在第 11 章，将演示如何编写简单的 OLE DB 提供者。

要想知道系统中安装了哪些提供者，可以使用 Data Link properties 对话框。在本章后面，将介绍如何使用它。

### 8.1.3 提供者与驱动程序

值得注意的是，OLE DB 对 ODBC 的兼容性，允许 OLE DB 访问现有的 ODBC 数据源。其优点很明显，由于 ODBC 相对 OLE DB 来说使用得更普遍，因此可以获得的 ODBC 驱动程序相应地要比 OLE DB 的要多。这样不一定要得到 OLE DB 的驱动程序，就可以立即访问原有的数据系统。

避免混淆提供者与驱动程序是重要的，图 8-2 明确了它们之间的区别。

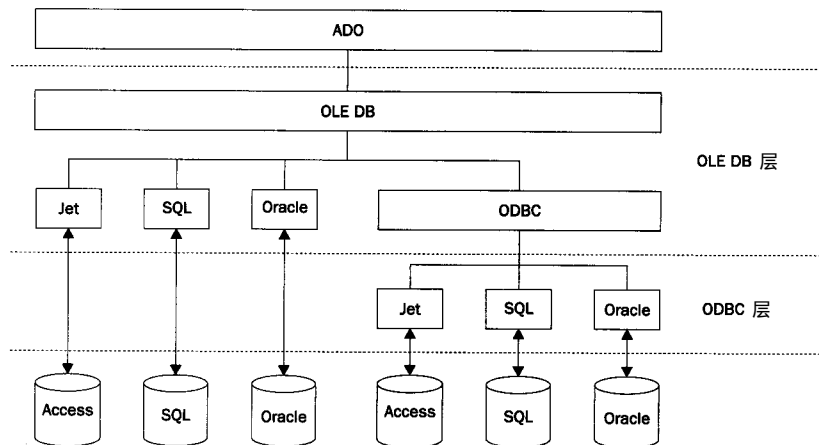


图8-2 提供者与驱动程序之间的区别

提供者位于 OLE DB 层，而驱动程序位于 ODBC 层。如果想使用一个 ODBC 数据源，需要使用针对 ODBC 的 OLE DB 提供者，它会接着使用相应的 ODBC 驱动程序。如果不需要使用 ODBC 数据源，那么可以使用相应的 OLE DB 提供者，这些通常称为本地提供者（native provider）。

可以清楚地看出使用 ODBC 提供者意味着需要一个额外的层。因此，当访问相同的数据时，

针对ODBC的OLE DB提供者可能会比本地的OLE DB提供者的速度慢一些。

## 8.2 ADO 2.5对象模型

虽然在ADO 2.5对象模型中出现了两个新对象，但与以前的版本基本上是类似的。图 8-3显示了这些对象以及每个对象之间的关系。

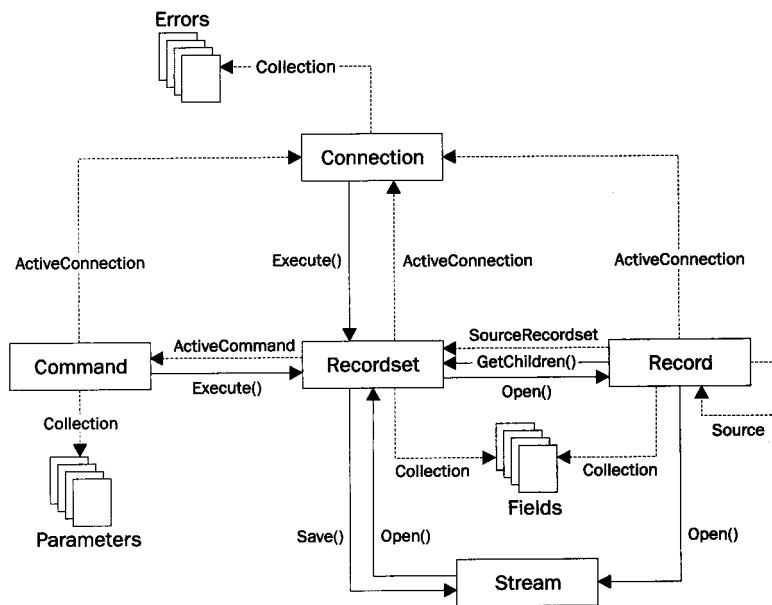


图8-3 对象之间的相互关系

如果以前曾使用过 ADO，你会发现这个新版本中出现了两个新对象：Stream 和 Record 对象。在第 11、12 章将详细地介绍它们。

Properties集合已经被有意地排除在图 8-3外，这样你对几个主要对象之间的交互关系就一目了然了。在本章的后面，有显示 Properties集合的简化对象模型。

让我们更详细地考察这几个对象。

### 8.2.1 Connection对象

Connection对象是使我们能与数据存储相连的对象。只有 Connection对象才能指定希望使用的OLE DB提供者、连接到数据存储的安全细节以及其他任何连接到数据存储特有的细节。

应该注意的是，不必显式创建一个 Connection对象以连接到数据存储。尽管确实需要指定连接细节，但没有 Connection对象，一样可以创建 Command、Recordset和Record对象。如果不创建自己的 Connection对象，ADO将会隐含地为你创建一个 Connection对象。如果要对提供者运行多条命令，应该显式地创建一个 Connection对象，这比每运行一条命令就创建一个连接更有效。

除了为数据存储提供连接以外，Connection对象允许针对数据存储执行命令操作。这些命令可以是结构化的或存储的命令(例如，SQL命令或一个存储过程)，并且可以有选择地从数据存储中返回一些数据。

### 8.2.2 Command对象

Command对象是对数据存储执行命令的对象。看到这里读者可能会产生疑问, Connection对象不也能这样做吗?是的,但是 Connection对象在处理命令的功能上受到一定的限制,而Command对象是特别为处理命令的各方面问题而创建的。实际上,当从 Connection对象中运行一条命令时,已经隐含地创建了一个 Command对象。

有时其他对象允许向命令传入参数,但在 Connection对象中不能指定参数的任何细节。使用Command对象允许指定参数(以及输出参数和命令执行后的返回值)的精确细节(比如,数据类型和长度)。

因此,除了执行命令和得到一系列返回记录,也可能得到一些由命令提供的附加信息。

对于那些不返回任何记录的命令,如插入新数据或更新数据的 SQL查询, Command对象也是有用的。

### 8.2.3 Recordset对象

Recordset对象是ADO中使用最为普遍的对象,因为它含有从数据存储中提取的数据集。我们经常运行不返回数据的命令,比如那些增加或更新数据的命令,但在大多数情况下很有可能会取得一系列记录。

Recordset对象是拥有这些记录的对象。可以更改(增加、更新和删除)记录集中的记录,上下移动记录,过滤记录并只显示部分内容等等。Recordset对象也包含Fields集合,Fields集合中有记录集中每一个字段(列)的Field对象。

无论是在ASP页面中处理数据,还是利用远程数据服务(RDS)远程使用数据,Recordset对象是必须处理的对象。

### 8.2.4 Record对象

ADO 2.5以前的版本在处理结构化数据上是很有效的,比如从数据库中取出记录集,但无法处理每一行的列(也就是列数和数据类型)可能不同的数据。对于SQL数据这不是一个问题,但对于文件和邮件系统,Web服务器和别的诸如此类的数据存储会如何呢?我们把这些数据看作是半结构化的数据,与记录集相比结构性较差,但与那些常用来代表文本或图像的二进制数据相比更具有结构性。

通常,半结构化数据的存储采用树状结构来组织,有节点、子节点和文件。例如,设想一个有文件夹、子文件夹和文件的Web站点。图8-4所示的屏幕图显示了一台机器的Web站点,特别是还有一个名为public的虚拟目录。

如果一定要在ADO中进行建模,会觉得这非常适合记录集,可能是嵌套的记录集。然而注意高亮显示的目录,该目录含有不同类型的文件,里面有几个目录、一个ASP文件、一个文本文件和一个WORD文档。你会很容易地将其映射到一个拥有名称、类型、上次存取时间等字段的记录集,遗憾的是并不是这样简单。对于访问权限而言,在文件和目录之间就有区别。对于目录,需要的是能访问目录下的文件;而对于文件,却可能是需要访问其内容。

由于其复杂性,引入了Record对象。在上面的情况下,存在有一些相同属性的条目的一个集合,但是每个条目也有独特的属性,因此需要使用别的方法去处理这些数据。把一个集合映射到一个记录集,一个单独的文件映射成一条记录,相应的文件属性就映射成Fields集合。



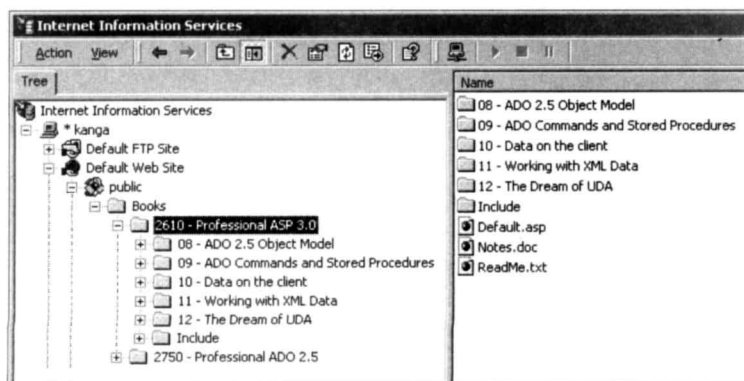


图8-4 显示了一台机器的Web站点的界面

这意味着有了一个含有九行记录的记录集。访问记录集中单独的一行就会得到该文件的属性(字段)，但是提供给我们的仅仅是属性。为了访问文件或目录的内容，需要使用 Record对象，该对象包含文件或目录的独特属性。习惯于这个概念有一定的困难，但不必担心，在第11、12章你会看到更多的有关Record对象的例子。

Windows 2000初始版本发布以来，只有用于 Internet发布的OLE DB提供者使用Record对象。一旦微软Exchange 6.0发布，将成为以类似方法利用记录集和记录提供对Exchange信息库访问的OLE DB提供者。

### 8.2.5 Stream对象

Stream对象用于访问节点的内容，比如一个 E-Mail消息，或一个Web页面。利用Stream对象可以访问文件或资源的真实内容。因此，结合 Record和Recordset对象，不仅可以访问 Web服务器上的文件或email消息，还可以访问相应的内容。这样，就可以创建一个只使用 ADO去访问邮件系统的邮件客户。这也许不会有太多的优点，但意味着可不必了解邮件系统的 API或对象模型，减少了学习上的弯路。

Stream的另一个用途是XML，可以访问一系列作为XML流的数据(结构化或半结构化)。

Stream对象用来处理二进制数据，所以，可以用来处理 BLOB类型的数据，比如数据库中的图像或大文本数据。

同样，在本书第11、12章中你会看到更多的有关Stream对象的例子。

### 8.2.6 集合

ADO对象库中有一些集合，每个集合都有零个或更多个与其关联的对象的拷贝。可以使用相同的代码结构去遍历这些集合：

在VBScript中的语法是：

```
For Each Object In Collection
    'Do something with Object
Next
```

例如，遍历一个Recordset对象的Fields集：

```
For Each objField In rs.Fields
    Response.Write objField.Name & "<BR>"
Next
```

如果选择JScript，那么可以使用Enumerator对象：

```
for (objField = new Enumerator(rs.Fields);
    !objField.atEnd(); objField.moveNext())
    Response.Write(objField.item().Name + '<BR>');
```

### 1. Fields集合

Fields集合拥有与记录集或记录关联的Field对象。对基于结构化数据的记录集，比如SQL数据，字段相应于数据中的列，并含有列的详细内容，比如名称、数据类型、长度等等。在以后几章看到大量的关于Fields集合的例子。

对于半结构化的数据，对象的属性相应于字段。在第12章会看到更多的相关的介绍。

### 2. Parameters集合

Parameters集合仅被Command对象使用，确定在存储命令中的参数。SQL数据库中的存储过程频繁地使用参数，并允许数据传入和传出预先定义的SQL语句。如果拥有向ADO返回信息的参数，则会十分有用，因为这样从存储过程中返回的就不只是一个记录集。例如，考虑一个更新多个表然后返回一个记录集的复杂存储过程，可以用一个输出参数显示更新了多少条记录。

另一个使用参数的理由是性能问题，特别是在仅仅需要从存储过程返回单个值的时候。在这种情况下，没有创建记录集的必要，只需要保存一个值即可，因而不需要返回记录集，而返回输出参数的值是更为有效的方法。

在第9章会看到有关Parameters集合的详细介绍。

### 3. Errors集合

Errors集合包含因运行命令而引起的上一次ADO或OLE DB提供者错误的详细内容，只能被Connection对象访问。可能会觉得这是个限制，因为不必显式定义Connection对象，但可以通过Command、Recordset和Record对象的ActiveConnection属性访问隐含的Connection对象。例如：

```
For Each objError In rs.ActiveConnection.Errors
    Response.Write objError.Name & "<BR>"
Next
```

在本章后面，将详细讨论Errors集合。

### 4. Properties集合

为了避免混乱，Properties集合没有在前面的对象模型图上加以显示。它与对象模型的关系显示在图8-5中。

Properties集合存在的原因是因为ADO是用来处理许多不同的数据存储，都有不同的特征。将属性(Property)组成一个集合，可以使之能够动态地根据不同的数据提供者而随时改变。例如，Jet的OLE DB提供者允许访问Jet特

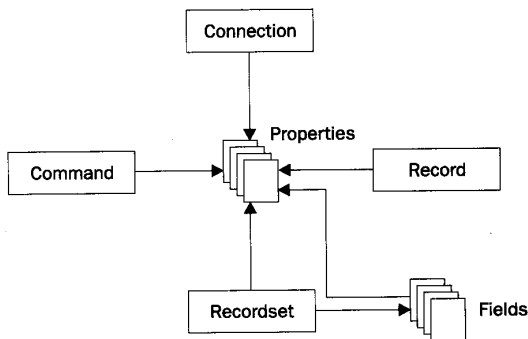


图8-5 Property与对象之间的关系



殊的安全属性：

```
Set conDB = Server.CreateObject("ADODB.Connection")
conDB.Open "DSN=Nwind"
```

```
conDB.Properties("Jet OLEDB:Database Password") = "LetMeIn"
```

其他的提供者没有这个属性，因此把它加到 Connection对象中作为一个静态属性是不明智的。ADO根据所使用的OLE DB提供者，会用提供者缺省值填充属性集合。

虽然这里有关于 Properties集合的使用说明，但在本书中不准备详细叙述 Properties集合。要获得更多的关于该集合的介绍，请参阅《Professional ADO 2.5 Programming》或《ADO 2.5 Programming's Reference》，两者都由 Wrox出版。

### 8.2.7 ADO 常数

当使用ADO时，会发现有许多预先定义的用于众多选项的常数，比如定义光标类型和锁类型的常数。使用像 Visual Basic或Visual C++这样的语言，一旦引用了ADO类型库，自然会用到这些常数。在ASP中情况不同，有两种选择。

引用常数的第一种方法是把它们包含进ASP文件：

```
<!-- #INCLUDE FILE="adovbs.inc" -->
```

可以将包含文件拷入本地目录，或者从安装目录引用它，其缺省路径为 C:\Program Files\Common Files\System\ado(以上文件包含用于VBScript的ADO常数——对于JScript，应该使用adojavas.Inc)。使用这个方法的一点不足是会使ASP页面变得过大，因为包含了全部的常数，而其中许多是不需要使用的。

可以创建自己的只含有所需要的常数的包含文件，但越来越多地使用ADO的功能时，很可能会发现需要不断地编辑、维护这个文件。

一个比较好的解决的方法是创建一个对类型库的引用，这种方法不需要将常数包含进ASP文件而直接可以引用常数：

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

不要怀疑这个DLL的名字msado15.dll，这是一个正确的名字，包含最新版本的ADO。

可以在需要的地方把这个METADATA语句包含进每一个ASP文件，或者放入global.asa文件，这样应用程序中的每个Web页面都可以引用这些常数。

## 8.3 连接到数据存储

如果需要访问一个数据存储，应该创建一个到数据存储的连接。前面已经提到过：可以显式地创建一个Connection对象，或者让ADO隐含地创建一个连接。对于任何一种方式，都必须知道数据存储的详细内容。

虽然用于连接的实际细节不尽相同，但对于所有类型的数据存储，其连接的实际方法是相同的。这并不令人惊奇，因为不同的提供者需要不同类型的信息。在允许用户访问数据存

储之前，一些提供者需要用户的证书，而别的提供者却接受默认的安全证书。

连接到数据源有好几种方法：

- 连接字符串。在字符串中放入连接的细节，或在打开数据存储时，直接将连接细节加入到命令中。这种方法的优点是连接细节将保留在 ASP 页面中。不足之处，如果你有较多的页面，在改变了连接细节时，将陷于繁重的维护工作当中。解决的方法是创建一个包含连接细节的字符串变量，并放进一个 ASP 包含文件，这样的话仅仅有一个连接字符串的实例，但能保持与其他的 ASP 页面相符。另一个常用的技术就是将应用程序中的连接字符串存储到状态变量中，这样可以被应用程序中的所有页面使用。
- 数据链接文件。这是一个含有连接细节的文件（扩展名为 .udl）。优点是对于任何数量的 ASP 页面只需要一个数据链接文件。要创建一个数据链接文件，只需创建一个新的文本文件，并重新命名（要确保 Windows 资源管理器显示文件扩展名）。一旦重新命名了该文件，就可以打开它（双击）以显示 Data Link Properties 对话框。以前版本的 ADO 允许从 Windows 资源管理器的 New 菜单建立数据链接文件。我们将在本章稍后看到有关数据链接文件的内容。
- ODBC 数据源，或 DSN。有点类似于数据链接文件，但只适用于 ODBC 数据源。它们集中起来用于 ASP 页面，数据源必须是系统数据源。ODBC 数据源从 ODBC 数据源管理器（ODBC Data Source Administrator）中创建，这个工具可在 Administrative Tools 文件夹中找到。

这三种方法无论那一种都可以使用，使用那一种只是一种偏爱而已。直接的连接字符串可能速度快一些，因为提供所有的连接细节。数据链接文件需要从文件中读出连接细节，ODBC 数据源需要从注册表中读取连接细节。当然，速度的差异是很小的，每种方法各有优缺点。

### 8.3.1 连接字符串

连接字符串依赖于提供者，因为每个数据提供者可能需要不同的细节。

值得注意的重要一点是，ODBC 的 OLE DB 提供者是缺省的，所以，如果不使用 Provide = 部分，系统将自动地使用 ODBC。

下面为不同的提供者列举了连接字符串的例子，在本书的后面将会看到更多的例子。

#### 1. 微软 Access

如果使用 ODBC，而没有 DSN：

```
Driver={Microsoft Access Driver (*.mdb)}; DBQ=C:\wrox\database_name.mdb
```

对于本地的 OLE DB 提供者：

```
Provider=Microsoft.Jet.OLEDB.4.0; Data Source= C:\wrox\database_name.mdb
```

上面的例子说明了 Access 数据库存放于 C:\wrox 目录下。虽然读者可能会尝试将数据库作为 Web 文件存放于相同的目录下，但不要这样做，否则任何人都可以下载整个数据库文件。将数据库存放于 Web 目录外永远是明智的，没有人可以从外面访问该文件。

#### 2. 微软 SQL Server

对于微软 SQL Server，使用针对 ODBC 的提供者：

```
Driver={SQL Server}; Server=server_name; Database=database_name; UID=user_name;  
PWD=user_password
```

例如：

```
Driver={SQL Server}; Server=WATCHER; Database=pubs; UID=davids; PWD=whisky
```

对于本地 OLE DB 提供者，语法类似：

```
Provider=SQLOLEDB; Data Source=server_name; Initial Catalog=database_name; User  
Id=user_name; Password=user_password
```

例如：

```
Provider=SQLOLEDB; Data Source=WATCHER; Initial Catalog=pubs; User Id=davids;  
Password=whisky
```

### 3. 微软索引服务

索引服务只能通过本地的 OLE DB 提供者使用。其语法：

```
Provider=MSIDXS; Data Source=catalog_name
```

例如，使用 Web 目录：

```
Provider=MSIDXS; Data Source=Web
```

### 4. ODBC 驱动程序

在使用针对 ODBC 的 OLE DB 提供者的例子中，Driver 显得较为冗长。例如：

```
Driver={Microsoft Access Driver (*.mdb)}; DBQ=C:\wrox\database_name.mdb
```

当创建一个新数据源时，使用的驱动程序的准确名字应该是从驱动程序列表中得到的，如图 8-6 所示。

### 5. 数据链接文件

以前版本的 ADO 允许在资源管理器中的目录上右击鼠标来创建一个数据链接文件。创建了新文件后，打开该文件从而得到 Data Link Properties 对话框。在写本书的时候，微软已经从鼠标右键菜单中删除了该选项，因为他们觉得这会让用户感到混乱。但微软声称会提供一个注册表文件以便再次引入这项功能。

Name	Version	Con
Microsoft Access Driver (*.mdb)	4.00.3719.09	Mic
Microsoft dBase Driver (*.dbf)	4.00.3719.09	Mic
Microsoft Excel Driver (*.xls)	4.00.3719.09	Mic
Microsoft ODBC for Oracle	2.573.3719.09	Mic
Microsoft Paradox Driver (*.db)	4.00.3719.09	Mic
Microsoft Text Driver (*.txt; *.csv)	4.00.3719.09	Mic
Microsoft Visual FoxPro Driver	6.01.8440.01	Mic
SQL Server	3.70.06.90	Mic

图 8-6 驱动程序列表

不要忘了，也可以简单地通过创建一个空文本文件，并将其扩展名改为 .udl 来创建一个数据链接文件。

一旦有了物理上的数据链接文件，就可以通过鼠标双击或者右击鼠标选择 Open 打开文件。接下来，读者会看到图 8-7 所示的对话框。

图中的详细内容因选择的提供者的不同而不同。上面的例子显示了 SQL Server 提供者，连接到一个称为 WATCHER 的 SQL Server 上，以 davids 的身份登录（口令被屏蔽了），使用 pubs 数据库。注意，如果选择 Allow saving password 选项，输入的口令将会在 UDL 文件中以明文保存下来。

如果要改变提供者，可以选择“Provider”选项卡，如图 8-8 所示。

通过这个选择，可以选出所需要的提供者，然后按下 Next 按钮填入适当的连接细节。

也可以在文本编辑器中编辑文件，如图 8-9 所示。

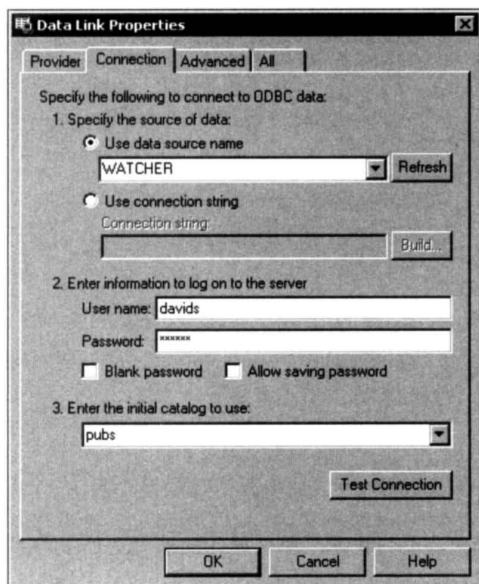


图8-7 Data Link Properties对话框

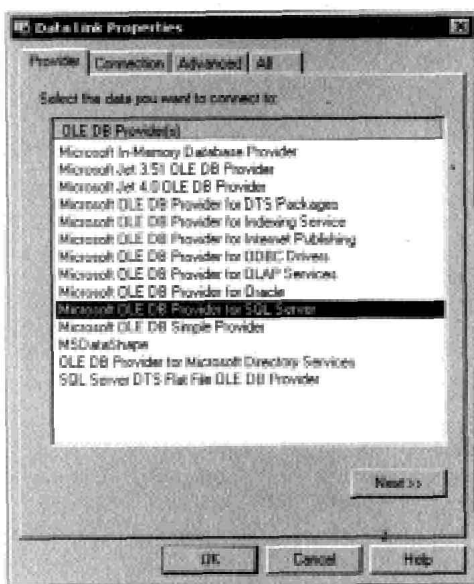


图8-8 改变提供者的选项卡

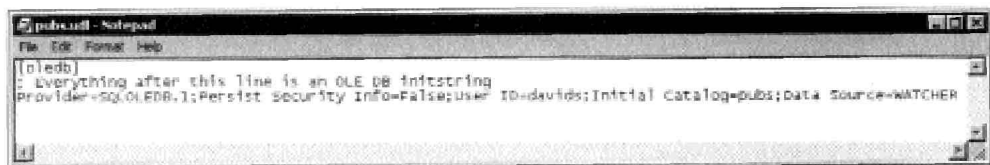


图8-9 编辑提供者的界面

可以看到在UDL文件中确实存有一个连接字符串。

要使用数据链接文件，仅需要在打开连接时指定这个数据链接文件：

```
conPubs.Open "File Name=C:\wrox\pubs.udl"
```

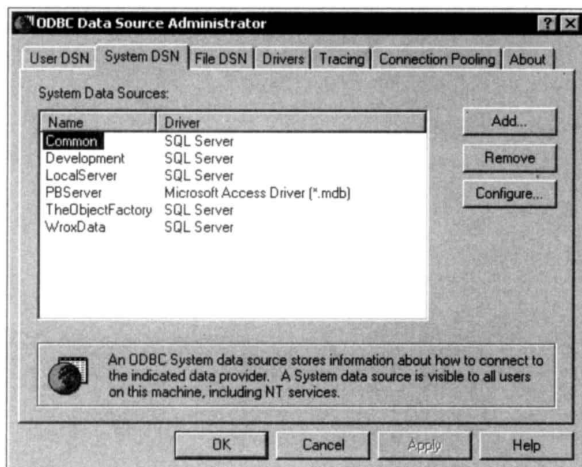


图8-10 选择数据源的界面

## 6. ODBC数据源

ODBC数据源(通常称为数据源名称,即DSN)可以通过Administrative菜单的Data Source选项进行设置。在以前版本的Windows中把它作为控制面板中的一个小程序。为了在ASP页面中访问DSN,必须确定该DSN已被设置为系统DSN。这只需在Data Source Administrator中选择System DSN选项卡,然后选择Add按钮,如图8-10所示。

然后,就可以选择希望使用的ODBC驱动程序,并填入适当的ODBC参数。

一旦建立了DSN,可以使用连接字符串的“DSN=”属性。例如:

```
conPubs.Open "DSN=pubs"
```

## 8.3.2 使用包含文件

使用包含连接字符串的包含文件提供了一个中心区域来存储许多ASP页面需要的连接细节。要这样做,仅仅需要创建一个新的ASP文件,不妨称为Connection.asp,并在其中加入下面的代码:

```
<%  
    strConn = "Provider=SQLOLEDB; Data Source=WATCHER; " & _  
              "Initial Catalog=pubs; User Id=davids; Password=whisky"  
%>
```

在ASP页面中,现在可以在该页的顶端加入这一行:

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

这样不必再为每个ASP页面都输入连接细节,同时方便于更改整个站点都使用的连接。

包含文件也是放置METADATA标签的好地方。

## 8.3.3 使用连接状态

将连接字符串存入应用程序变量是一个常用的技巧,同使用一个包含文件一样有效。例如,可以在global.asa文件中加入下面的代码:

```
Sub Application_OnStart()  
  
    strConn = "Provider=SQLOLEDB; Data Source=WATCHER; " & _  
              "Initial Catalog=pubs; User Id=davids; Password=whisky"  
  
    Set Application("ConnectionString") = strConn  
  
End Sub
```

在ASP页面中,可以使用下面的代码:

```
Set conPubs = Server.CreateObject("ADODB.Connection")  
  
conPubs.Open Application("ConnectionString")
```

从个人的角度,我更喜欢使用包含文件的方法,因为我写了许多不同的连接到各种服务和数据库的例子。使用应用程序方法将意味着每次都必须关闭浏览器重新启动应用程序。读者可以使用自己喜欢的任一种方法,在速度上它们并没有差别。

对于在本书的这节内的例子,将使用一个含有连接字符串的connection.asp文件作为一个包含文件。



8.3.4 连接语法

上面所叙述的是相关理论，当确实要与数据存储连接时，应该怎么办？如果使用显式定义的Connection对象，可以使用Open方法，它的语法如下：

```
connection.Open [ConnectionString], [UserID], [Password], [Options]
```

参数如表8-1所示。

表8-1 Open方法的参数及说明

参 数	说 明
ConnectionString	包含连接细节的字符串。可以是ODBC DSN的名称、数据链接文件的名称或真实的连接细节
UserID	连接期间，用户使用的名字。覆盖连接字符串中提供的任何用户名
Password	用户的口令。覆盖连接字符串中提供的任何口令
Options	可以是adAsyncConnect，指定异步地建立连接。忽略这个参数，则建立一个同步连接

异步连接不用于ASP环境，因为脚本语言不能接收来自ADO的事件。

8.3.5 连接的例子

下面是几个示例，这里假定strConn包含一个有效的连接字符串。

为了打开一个连接，使用Connection对象的Open方法。例如：

```
Set conPubs = Server.CreateObject("ADODB.Connection")

conPubs.Open strConn

' Some processing

conPubs.Close
```

也可以使用ConnectionString属性：

```
Set conPubs = Server.CreateObject("ADODB.Connection")

conPubs.ConnectionString = strConn
conPubs.Open

' Some processing

conPubs.Close
```

这两种实现方法之间没有什么区别，如果使用前一种方法来实现连接，ConnetionString属性同时也被赋值。

值得注意的是，一旦与数据存储建立了连接，ADO可能会改变ConnectionString属性值。不必担心，ADO只填写一些额外的属性值。

8.3.6 连接缓冲池

连接缓冲池(connection pool)总使许多人感到困惑，其实原理非常简单。当关闭一个连接，就用户(和ADO)而言，这个连接已经关闭。但实际上OLE DB并没有关闭这个连接，只是将其



放入了非活动的连接缓冲池中。任何时候用户 (或其他人) 打开一个连接, OLE DB 首先检测连接缓冲池中是否有相同连接细节的连接存在。如果有, 将直接从缓冲池中取得此连接。如果没有, 则为用户创建一个新的连接。为了避免浪费资源, 经过一段缺省的时间段后, 就从缓冲池中清除该连接。

那么, 它的优点在哪里? 打开一个连接可能是所进行的操作中最慢的操作之一, 连接缓冲池使用户能与数据存储再次连接而无须重新创建连接。这对于那些连续打开和关闭大量连接的Web站点显得特别重要。

对于ODBC连接, 连接缓冲池由 ODBC Data Source Administrator 控制。对于 OLE DB, 不能改变连接缓冲池 (或叫会话缓冲池)。

必须注意的是, 连接缓冲池不是连接共享。一个连接只有在被客户关闭后才能再次使用。正在使用的连接 (也就是没有被关闭的) 是不能重新打开的。

内务处理  
为了使连接缓冲池生效, 必须确保内务处理 (Housekeeping) 处于有序状态。这包括及时关闭Connection对象, 这样它们才能回到缓冲池重新使用。你可能认为不断地打开、关闭连接对系统的开销很大, 但必须衡量一下可扩展性——你的应用程序可能有许多人在使用, OLE DB 又非常善于管理连接资源。

一般的原则是: 尽可能晚地建立连接, 同时又要尽可能早地关闭连接, 这样保证连接打开的时间段最短。

8.4 记录集

前面已经提到, 记录集是 ADO 中最常用的对象, 这并不值得奇怪。毕竟, 他们包含着数据。但是, 对于记录集还有比想象的更多的内容, 知道数据如何保存和处理很重要, 因为这为选择使用哪种记录集提供了更多的参考。

记录集有不同的类型, 在一些细小的地方存在着差异, 很容易造成失误。首先需要认真谈论的是光标的概念。

8.4.1 光标

光标(cursor)是让许多人感到困惑的概念, 但实际上非常的简单。

光标用来管理记录集和记录集的当前位置, 后者是由当前记录指针来处理的。

这不是Recordset对象所做的吗? 是的, 但是记录集也是依靠它的光标。这仍然没有回答光标是什么这个问题, 那么先来看一个记录集, 如表 8-2 所示。

表8-2 一个记录集的例子

AU_ID	AU_LNAME	AU_FNAME	PHONE
172-32-1176	White	Bob	408 496-7223
213-46-8915	Green	Marjorie	415 986-7020
238-95-7766	Carson	Cheryl	415 548-7723
267-41-2394	O'Leary	Michael	408 286-2428
274-80-9391	Straight	Dean	415 834-2919
341-22-1782	Smith	Meander	913 843-0462

这里有六行四列。打开一个记录集，当前记录就是第一个记录，即为 Bob White 的那条记录。用什么来标识当前记录？用当前记录指针。那么又该如何处理这个指针呢？当需要移到下一条记录或者是其他记录时，是通过光标来实现的。在访问当前行的字段时，光标知道目前位于哪一行，所以能返回正确的值。如果试图移出记录集的最后一行，光标也会处理。

理解光标的一种好方法是将光标想象成为一个可以在记录集内移动的窗口。这一窗口与记录集的单个行同样高，同样长，因此一次只能看到一行数据值。当你移到另一条记录时，这个窗口也跟着移动。

也许认为这相当简单，但它确实很重要，因为能用光标做什么是由光标的类型决定的。

### 1. 光标类型

光标的类型标识了光标所能够提供的功能。这里有四种类型的光标：

- 静态(adOpenStatic)。静态光标含有对记录的静态拷贝。这意味着在记录集建立之后，记录集的内容就固定了。其他用户对记录的更改、添加和删除都是不可见的。允许在记录集中向前、向后移动。
- 只许前移(adOpenForwardOnly)。缺省的光标类型，除了只允许向前移动外，其余的与静态光标相同。
- 动态(adOpenDynamic)。动态的光标没有固定的记录集。其他用户的更改、添加或删除操作在记录集中是可见的。允许在记录集中向前、向后移动。
- 键集(adOpenKeyset)。键集类型的光标除了记录集是固定的，其余的与动态光标相似。可以看到其他用户的修改，但新记录却不可见。如果别的用户删除了记录，那么这些记录在记录集中将会变得不可访问。这项功能是通过标识记录集的键来实现的，所以键一直保留着，即使改变或删除记录。

为了理解这些概念，再想相光标窗口。对于只许前移的光标，可以看作是一个位于单向齿轮上的窗口，只能向前移动。这一特点的有利之处在于一旦通过了一条记录，光标就会完全忘记该记录，因为永远不会回到该记录上。静态光标则移去了单向齿轮，允许向后移动；因为也能向后移动，光标需要跟踪这些记录。由于这个原因，静态光标比只许前移的光标慢。

对于键集和动态类型的光标，窗口可以前后移动，但所看到的内容可能会改变。键集光标可以看到别人对数据的更改，但看不到新的或已删除的记录。因此，记录集是固定的，但不是内容不固定。动态光标将它扩展了，不仅可以改变记录的内容，而且可以改变记录集。所以在动态光标中能够看到有新的记录出现，同时删除的记录从记录集中消失。

使用的光标类型取决于想达到的目的。如果只想浏览记录，也许是为了创建一个表格或一个选择列表，那么用只许前移的光标是最好不过了。虽然使用其他类型的光标速度可能会慢一些，但也可以正常工作。

光标的类型会影响性能，特别是服务器光标。例如，在微软的 SQL Server 6.5 中，键集和静态类型的光标都需要在临时数据库 (tempdb) 中放入一个完整的数据拷贝。其中，键集类型的光标相比较而言稍微高效一些，因为它只将键拷入临时数据库。对于 SQL Server 7.0 情况不是这样，不同类型的光标的运行效率差别不是很大。

### 2. 光标位置

既然已经解释了什么是光标，以及光标如何管理数据，但是光标在哪里呢？答案不是固定的，因为光标依赖于数据存储。某些数据存储，比如微软的 SQL Server，有自身的光标服

务；而别的如微软的 Access 却没有光标服务。

当打开一个记录集时，必须选择是否希望数据存储管理光标，或是否希望 OLE DB 和 ADO 在本地为你管理光标。后者可以实现是因为 OLE DB 有其自己的光标服务。通过使用 Connection 对象或 Recordset 对象的 CursorLocation 属性可以设置这两个选项。可以设定该属性的值为：

- adUseServer：让数据存储管理光标。
- adUseClient：让 ADO 管理光标。

可以在打开连接或记录集之前设置这个属性：

```
conPubs.CursorLocation = adUseServer  
conPubs.Open strConn
```

或者：

```
rsAuthors.CursorLocation = adUseClient  
rsAuthors.Open "authors", conPubs
```

缺省的光标是基于服务器的，理解这两种类型的区别非常重要。对于一个服务器光标来说，数据存储的任务是管理记录，所以，当使用服务器光标建立一个记录集时，数据存储管理着记录的移动、记录的更新等等。

对于一个客户光标，记录集的全部内容复制给客户，受本地客户光标服务管理。这意味着对于一个客户光标，打开一个具有大量记录的记录集要比使用基于服务器的光标打开相同记录集所花费的时间长得多。也有需要使用基于客户的光标的时候，在本书后面，研究组件时，会看到更多的相关的例子。

### 3. “消防带”光标

你可能知道“消防带” (Firehose) 光标，由于能给应用程序带来高的运行效率，所以对其进行解释显得非常重要。因为“消防带”光标是一种特殊类型的光标，只有在与微软的 SQL Server 连接时才出现。SQL Server 创建用户请求的数据集，然后把数据直接传给客户以使其尽可能快地得到数据。SQL Server 自身几乎没有光标管理，这意味着它可以更快地处理数据。也就是说数据可以在非常短的时间内迅速返回到客户端。从客户方看，类似于只许前移的光标。

那么，在前面讨论光标类型时，为什么没有涉及到“消防带”光标呢？因为这类光标专用于 SQL Server，并仅用于使用基于服务器的光标时。这不是一种真正的光标类型，获得一个“消防带”类型光标的方法就是不指定光标的类型。

## 8.4.2 锁定

我们已经解释了光标和如何管理数据。现在可以创建记录集了吗？恐怕还不行，因为还有一个问题没有讨论，那就是锁定。

锁定就是如何确保数据的完整性，确保更改不会被覆盖。我们需要避免的典型情况是多次更新，比如一个用户改动了一些数据，接着另一个用户立即又将其做了修改。为了对这种情况加以保护，要锁定记录，有许多不同的方法可以保证记录得到保护。可通过锁定类型来设置这些方法。

### 1. 锁定类型

锁定类型决定更新记录时记录是否或如何被锁定。有四种类型的锁定：

- 只读(adLockReadOnly)：缺省锁定类型，记录集是只读的，不能修改记录。
- 悲观的(adLockPessimistic)：当修改记录时，数据提供者将尝试锁定记录以确保成功地编辑记录。只要编辑一开始，则立即锁住记录。
- 乐观的(adLockOptimistic)：直到用Update方法提交更新记录时才锁定记录。
- 批量乐观的(adLockBatchOptimistic)：允许修改多个记录，只有调用 UpdateBatch方法后才锁定记录。

当不需要改动任何记录时，应该使用只读的记录集，这样提供者不用做任何检测。对于一般的使用，乐观的锁定可能是最好的选择，因为记录只被锁定一小段时间，数据在这段时间被更新。这减少了资源的使用。

悲观的锁定提高了数据的完整性，但却是以牺牲并发性为代价的。并发性是许多用户在同一时间查阅数据的能力。锁定的记录对其他用户是不可见的，因而数据的并发性降低了。乐观的锁定只在一小段时间内锁定记录，所以增强了数据的并发性，但同时其他用户修改数据的几率也增加了。

关于并发性和锁定的问题在微软出版的《 Inside SQL Server 7.0 》(作者 Ron Soukup和Kalen Delaney)中做了较好的论述。这是一本很权威的专著，所以无论如何应购买这本专著，该书有大量的有价值的资料。

### 8.4.3 创建记录集

创建一个记录集十分容易，通过调用 Recordset对象的Open方法来实现：

```
Recordset.Open [Source], [ActiveConnection], [CursorType],  
               [LockType], [Options]
```

其参数及说明如表 8-3 所示。

表8-3 Open方法的参数及说明	
参 数	说 明
Source	数据源。可以是数据库中的表名、存储的查询或过程、SQL字符串、Command对象或适用于提供者的其他命令对象
ActiveConnection	记录集使用的连接。可以是一个连接字符串或者一个打开的 Connection对象
CursorType	使用的光标类型。必须是定义的光标类型中的一种，缺省值为 adForwardOnly
LockType	使用的锁定类型。必须是定义的锁定类型中的一种，缺省值为 adLockReadOnly
Options	告诉提供者 Source参数的内容是什么，如表、文本字符串等等

例如，要打开数据库 pubs中authors表上的记录集：

```
Dim rsAuthors  
  
Set rsAuthors = Server.CreateObject("ADODB.Recordset")  
  
rsAuthors.Open "authors", strConn  
  
' Do something here  
  
rsAuthors.Close  
Set rsAuthors = Nothing
```

注意，有几个参数没有指定。实际上，所有的参数都是可选的，可以在打开记录集之前

为它们设置相应的属性值：

```
Dim rsAuthors

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

With rsAuthors
    .Source = "authors"
    .ActiveConnection = strConn
    .CursorType = adOpenForwardOnly
    .LockType = adLockReadOnly
    .Open
End With

' Do something here

rsAuthors.Close
Set rsAuthors = Nothing
```

一旦打开记录集，当前指针自动地位于第一条记录上。如果在记录集中没有记录，那么 EOF 和 BOF 属性都是 True：

```
rsAuthors.Open "authors", strConn

If rsAuthors.BOF and rsAuthors.EOF Then
    ' Recordset is empty
End If
```

### 1. Options 参数

Open 方法的 Options 参数允许指定命令文本内容。它可以是以下 CommandTypeEnum 常数之一：

- adCmdText：文本命令，比如 SQL 字符串。
- adCmdTable：表名。
- adCmdStoredProc：存储过程名。
- adCmdFile：保存的记录集的文件名。
- adCmdTableDirect：表名。
- adCmdURLBind：URL 地址。

adCmdTable 与 adCmdTableDirect 的区别很小，如果想使用表中的全部列，使用 adCmdTableDirect 将由于 ADO 执行了某些内部优化而使运行速度变得稍快一些。

如果没有指定命令的类型，ADO 必须推算出执行的命令的类型，这将导致额外的开销。

这里还有两个选项：adCmdUnspecified 表示没有指定类型；adCmdUnknown 表示命令的类型未知。一般地可能不会使用它们。

### 额外的选项

Options 参数可以是以上常数中的任一个，但也可以加入下列 ExecuteOptionEnum 常数：

- adAsyncExecute：异步地执行命令。
- adAsyncFetch：取得初始的行集后，异步地获取剩下的行。
- adAsyncFetchNonBlocking：除了获取记录不阻止命令运行以外，其他与 adAsyncFetch 相似。
- adExecuteNoRecords：命令不返回任何记录。

异步处理意味着在后台执行操作，可以运行命令，然后继续其他工作，而不需等待其执行完毕(同步操作)。当创建用户界面时，这显得特别方便，因为可以从命令执行中返回，向用户显示一些内容，而同时数据的获取仍然在后台进行。当返回记录集时，这对 ASP程序员来说不是很有用，因为脚本语言不支持 ADO事件，所以记录集何时已完成填充移无法得知。当处理更新、删除或插入数据命令以及不返回记录集的时候，可以使用异步操作，即仅在不关心结果的情况下才能使用。

在另一方面，adExecuteNoRecords选项十分有用。它告诉 ADO执行的命令不返回任何数据。所以，就没有必要创建记录集(总之，可能为空)。这会加速正在运行的更新或添加数据的查询操作。

为了加入这些选项之一，可以使用 Or符号(等同于加号“+”)

```
adCmdStoredProc Or adExecuteNoRecords
```

```
adCmdStoredProc + adExecuteNoRecords
```

在下一章，将看到对相关内容更详细的介绍，因为这在处理命令(而不是记录集)时会更有用。

## 2. 在记录集中移动

一旦打开了一个记录集，经常需要遍历每一条记录。这需要使用 EOF属性。当到达记录集的末尾时，EOF就变为True，因此可以这样创建一个循环：

```
rsAuthors.Open "authors", strConn
```

```
While Not rsAuthors.EOF
    Response.Write rsAuthors("au_lname") & ", " & _
        rsAuthors("au_fname") & "<BR>"
    rsAuthors.MoveNext
Wend
```

上面的例子一直循环到EOF属性值为True时才退出。MoveNext方法用于移到下一条记录。

如果记录集允许向后移动，则可以使用 MovePrevious方法。在这种情况下，循环中需要检测BOF属性值。另外分别还有移动到第一条和最后一条记录的 MoveFirst和MoveLast方法：

```
rsAuthors.Open "authors", strConn, adOpenDynamic
' Now on first record
```

```
rsAuthors.MoveLast
' Now on last record
rsAuthors.MovePrevious
rsAuthors.MovePrevious
' Now three rows from the end of the recordset
```

```
rsAuthors.MoveFirst
' Back at the beginning again
```

## 3. 使用Fields集合

Fields集合包含记录集中每一字段(列)的Fields对象。Fields集合是记录集的缺省集合，因此在访问字段时可以省略，就如同上面的 While...Wend例子中的情况。因此，有多种访问字段的方法：

```
rsAuthors.Fields("au_lname").Value
rsAuthors("au_lname").Value
rsAuthors(1).Value
```



```
rsAuthors.Fields(1).Value
```

可以使用字段名，或使用它在集合中的位置。使用名字是最好的，因为这样将使代码更易于维护。

Value属性是字段的缺省属性，因此也可以省略，比如：

```
rsAuthors("au_lname")
```

如果想遍历所有字段，可以使用 For Each 结构：

```
For Each fldAuthor In rsAuthors.Fields
    Response.Write fldAuthor.Name & " : " & _
        fldAuthor.Value & "<BR>"
Next
```

这个例子将打印每一个字段的名称和值。

#### 4. 书签

当在记录集中移动时，可能需要保留记录的位置，以后再移回来。同真实的书签相似，一个记录集书签是一个指向单个记录的唯一的指针。

为了使用书签，只需将 Bookmark 属性值赋予一个变量：

```
varBkmk = rsAuthors.Bookmark
```

然后，可以在记录集中移动，以后可以通过相反的命令将记录移到做过书签标记的相应记录上：

```
rsAuthors.Bookmark = varBkmk
```

在记录集中查找记录时，书签是非常有用的。在本章稍后的 8.4.5 节中有一个相关的例子。

注意，并非所有记录集都支持书签，Supports 方法(在下面讨论)将能识别其是否支持书签。

值得注意的重要一点是，不能跨越不同的记录集使用书签，即使这些记录集是用相同的命令创建的。考虑一下以下的代码：

```
rsAuthors.Open "authors", strConn
rsAuthorsOther.Open "authors", strConn

varBkmk = rsAuthors.Bookmark
varBkmkOther = rsAuthorsOther.Bookmark
```

尽管两个记录集是用相同的命令创建的，但记录集的书签是不一样的。

可以使用 Clone 方法获得可交换的书签，但在这里我们不讨论它。

#### 5. 支持的功能

如上所述，并非所有的记录集都支持书签。还有许多其他的记录集选项也不是被所有的提供者或记录集类型支持的，因此可以用 Supports 方法验证一下。

Supports 方法使用一个或多个 CursorOptionEnum 值作为参数，返回 True 或 False 表明是否支持该选项。这些值的列表相当庞大，所以将其列于附录 F 中。

例如：

```
If rsAuthors.Supports(adBookmark) Then
    ' The recordset supports bookmarks
    varBkMk = rsAuthors.Bookmark
End If
```

可以使用Or或加号“+”组合多个常数：

```
If rsAuthors.Supports(adBookmark Or adFind) Then
' The recordset supports bookmarks and use of Find
End If
```

#### 8.4.4 过滤记录集

过滤是一种暂时地限定记录集中可见记录的一种方法。如果仅显示记录集中的某些记录，但又不需要每次都重新查询数据库，这种方法非常有用。

##### 1. 使用条件过滤

Filter属性拥有多个参数，其中一个就是条件表达式，它非常像 SQL中Where字句：

```
rsAuthors.Filter = "state = 'ca'"
```

这个语句限定记录集只显示州名为 ca的记录。使用这个过滤条件将使当前指针回到第一条匹配记录上。可以遍历记录集中的全部记录，并且只有匹配条件的记录才可见。

不仅仅限于单一条件，还可以使用 And或Or把多个条件连接在一起：

```
rsAuthors.Filter = "au_lname = 'Homer' Or au_lname = 'Francis'"
```

这将过滤出姓为Francis或Homer的记录。

上面的例子显示了一个列匹配一个值的过滤方法，也可以使用下面操作符中的任何一种：

<：小于。

>：大于。

<=：小于等于。

>=：大于等于。

<>：不等于。

LIKE：通配符。

当使用通配符操作时，可以使用“\*”或“%”符号。例如：

```
rsAuthors.Filter = "au_lname LIKE 'Ho%'"
```

“\*”或“%”作为一个通配符，匹配任何字符，因此上面的例子会匹配 au\_lname字段中以“Ho”字符开始的所有记录。

可以使用空字符串清空过滤条件，这样将显示全部记录：

```
rsAuthors.Filter = ""
```

##### 2. 使用常数过滤

Filter属性也能用FilterGroupEnum常数作为其参数：

- adFilterNone：清空当前过滤条件，与使用一个空字符串的效果相同。
- adFilterPendingRecords：只显示那些已改变的，但还没有送到服务器的记录，只在成批更新模式下可用。
- adFilterAffectedRecords：只显示那些受上一次调用 Delete、Resync、UpdateBatch和 CancelBatch方法影响的记录。
- adFilterFetchedRecords：显示高速缓存中的记录，即上一次调用读取记录的命令时的结果。
- adFilterConflictingRecords：显示在上一次成批更新中更新失败的记录。

稍后会看到关于成批更新的介绍。

### 3. 使用书签过滤

最后一种过滤记录集的方法是使用一个书签数组。可以使用这个技术创建一个记录列表，然后再应用一个过滤条件对其过滤。例如：

```
rsAuthors.Open "authors", strConn, adOpenKeyset, _
    adLockReadOnly, adCmdTableDirect

' Save bookmark for the first record
avarBkmk(0) = rsAuthors.Bookmark

' Move forward two records
rsAuthors.MoveNext
rsAuthors.MoveNext

' Save bookmark for the third record
avarBkmk(1) = rsAuthors.Bookmark

' Move to the end and save the bookmark
rsAuthors.MoveLast
avarBkmk(2) = rsAuthors.Bookmark

' Now apply the filter
rsAuthors.Filter = Array(avarBkmk(0), avarBkmk(1), avarBkmk(2))

' Now loop through the recordset
While Not rsAuthors.EOF
    Response.Write rsAuthors("au_lname") & "<BR>"
    rsAuthors.MoveNext
Wend
```

当循环至记录集末尾位置时，会发现只有三条记录，因为只有三个书签应用于过滤条件。注意，不能直接使用数组 avarBkmk，必须使用 Array 函数将各个书签转换成不同的数组。

### 8.4.5 查找记录

查找单个的记录由 Find 方法来完成。它类似于使用条件的过滤方法：

```
rsAuthors.Find "au_lname = 'Lloyd'"
```

它们之间最主要的区别在于这种方法只能有一个条件，不允许使用 And 或 Or。

可以使用可选的参数指定一些额外的选项，其完整的语法如下：

```
Recordset.Find Criteria, [SkipRows], [SearchDirection], [Start]
```

SkipRows 是一个数字，表示在开始查找记录前跳过的行数。缺省为 0，查询从当前行开始。

SearchDirection 可以是 adSearchForward，表示向前搜索记录；或者 adSearchBackward，表示向后搜索记录。

Start 是一个书签，指出开始查找记录的位置。

如果找到相应的记录，当前指针将位于匹配的记录上，如果没有找到记录，那么将位于下面两个位置中的一个：

- 如果是向前搜索，则位于记录集末尾位置的后面，EOF 被设置为 True。
- 如果是向后搜索，则位于记录集开始位置的前面，BOF 被设置为 True。

使用书签保存位置

如果没有找到相应的记录，记录的重新定位可以由书签轻松解决，因为可以为当前位置

制作书签，如果在查找记录过程中没有找到所需的记录，那么再移回到上次保存的位置。

例如：

```
' Save the current position
varBkmk = rsAuthors.Bookmark

' Find the record
rsAuthors.Find "au_lname = 'Sussman'"

' Was it found
If Not rsAuthors.EOF Then
    Response.Write "Found: " & rsAuthors("au_lname") & ", " & _
        rsAuthors("au_fname") & " <BR>"
Else
    Response.Write "Not found. Moving <BR>"
    rsAuthors.Bookmark = varBkmk
End If
```

使用Filter属性强于Find方法的一个原因是Find语句只能指定一个查询条件，而Filter属性允许指定多个条件。也就是说，当想要查找的字段条件不止一个时，不能使用Find方法。然而，可以先过滤记录，如果找到记录可以再删除过滤条件。

#### 8.4.6 修改记录

大部分的Web只用来显示信息，而Web应用程序正变得越来越普通。在这种情形下，如果只拥有只读数据确实没有什么用处。创建一个应用程序，几乎总是需要修改现存的数据或是添加新的数据，其方法有许多。在本节，将学习如何使用Recordset对象的方法来更改数据。在下一章，将会看到如何使用查询完成相同的任务。

可以设置除了adLockReadOnly之外的锁定类型，配合使用Recordset对象的方法去修改数据(假定有相应的权限)。记住，缺省的锁定类型是只读的。

##### 1. 添加记录

要在记录集中添加记录，使用AddNew方法。有两种使用AddNew的方法。第一种没有任何参数，仅仅调用AddNew，在记录集的最后添加一个空记录。在调用Update方法保存所做的更改之前，可以随意地修改字段中的数据：

```
With rsAuthors
    .Open "authors", strConn, adOpenDynamic, _
        adLockOptimistic, adCmdTableDirect

    .AddNew
    .Fields("au_id") = "123-12-1234"
    .Fields("au_lname") = "Lloyd"
    .Fields("au_fname") = "Janine"
    .Fields("contract") = 0
    .Update
End With
```

这只是添加了一条新纪录，设置四个强制型的字段值。

另一种方法是使用AddNew方法的可选参数，这是两个数组，一个是字段名，另一个是字段的值。

```
With rsAuthors
    .Open "authors", strConn, adOpenDynamic, _
```

```
adLockOptimistic, adCmdTableDirect
```

```
.AddNew Array("au_id", "au_lname", "au_fname", "contract"), _  
Array("123-12-1234", "Lloyd", "Janine", 0)
```

```
End With
```

这个方法不需要调用 Update 方法。

## 2. 编辑记录

编辑记录与添加记录的方法相似，不同之处在于不需要调用 AddNew 方法：

```
strSQL = "SELECT * FROM authors" & _  
" WHERE au_lname='Lloyd'"
```

```
With rsAuthors
```

```
.Open strSQL, strConn, adOpenDynamic, _  
adLockOptimistic, adCmdText
```

```
.Fields("contract") = 1  
.Update
```

```
End With
```

这仅仅是将当前记录 (在这种情况下是第一条记录，因为刚刚打开记录集) 的 contract 字段的值赋为 1。

## 3. 删除记录

删除记录需调用 Delete 方法。删除哪一条记录取决于可选的参数，可以是下面 AffectEnum 常数中的一个：

- adAffectCurrent：删除当前记录，缺省操作。
- adAffectGroup：删除匹配当前过滤条件的所有记录。
- adAffectAll：删除记录集中的全部记录。
- adAffectAllChapters：删除所有段(chapter)中的记录。

最简单的调用形式是：

```
rsAuthors.Delete
```

这将删除当前记录。如果有一个过滤条件，并想删除所有匹配该条件的记录，那么仅需加上适当的常数：

```
rsAuthors.Delete adAffectGroup
```

## 4. 自动递增的字段

当添加一条新记录时，一般会碰到这样一个问题：如何处理那些自动递增的或标识字段 (Identity Field)。这些字段是由服务器自动更新的数字字段，一般用于为每一行提供一个唯一的字段值。当数据库含有多个表时，那么这个唯一的字段经常被当作关联表的外键。所以，添加一条新记录时，经常需要找出它们的值。

例如，考虑一下有两个字段的表，一个自动递增的 ID 字段 (SQL Server 中的 IDENTITY 字段或 Access 中的 AutoNumber 字段)，一个字段名为 Name 的文本字段。现在考虑一下下面的向表中添加记录的代码：

```
With rsData
```

```
.Open "tblTest", adOpenDynamic, adLockOptimistic, adCmdTableDirect  
.AddNew  
.Fields("Name") = "Janine"
```

```
.Update  
  
intID = .Fields("ID")  
End With
```

看上去很平常，但添加记录后是否能够取到这个值依赖于光标的类型、锁定的类型以及ID字段是否被索引。表 8-4列出了哪些组合允许获取新插入的 ID字段的值。其他没有列在表中的组合不能返回ID字段的值。

表8-4 获取ID字段的值与光标、锁定的类型及 ID字段是否被索引的关系

提 供 者	对 象	索 引	光 标 位 置	光 标 类 型	锁 定 类 型
ODBC	Access 97	是	服务器	键集	悲观型
					乐观型
					悲观型
	Access 2000	是	服务器	键集	乐观型
					悲观型
		否	客户	所有	乐观型
	SQL Server 6.5	是	服务器	键集	悲观型
					乐观型
					悲观型
Jet 4.0	Access 97	是	服务器	所有	所有
					所有
					悲观型
	Access 2000	是	服务器	所有	所有
			客户	所有	乐观型
		否	客户	所有	悲观型
	SQL Server 6.5	是	服务器	键集	乐观型
					悲观型
					乐观型
SQL OLE DB	SQL Server 6.5	是	服务器	键集	悲观型
					乐观型
			客户	所有	悲观型
	SQL Server 7.0				乐观型
		否	客户	所有	悲观型
					乐观型

这清楚地说明必须使用正确的组合，才能保证能取得 ID字段的正确值。否则，可能会得到0、空值或NULL，这取决于组合的方式。

在下一章中处理存储过程时，将见到另一种从 SQL Server中获取IDENTITY字段值的方法。

8.5 管理错误

处理数据存储时，发生错误的可能性总是存在的：安全性问题，试图更新已被其他用户删除的记录，诸如此类的问题很多。不能保证一切都运行良好，因此必须构建某种形式的错误控制。

在第7章，研究了ASP页面中一般的错误处理，但现在涉及的是数据存储，所以必须考虑使用额外的代码进行错误处理。先看一下Errors集合，再讨论其如何满足ASP 3.0的错误处理机制。

8.5.1 Errors集合

Errors集合包含由单个ADO命令的执行而引起的每一个错误的 Error对象。使用Errors集合的原因是由于在一个命令的执行过程中，可能会引起多个错误，OLE DB提供者需要提供一种方式通知客户方已有多个错误发生。

关于Errors集合有两个重要的地方需要注意：



- 每次执行 ADO 命令，如果发生错误，就清空错误集，同时代之以新的错误内容。当然，如果没有错误发生，Errors 集合不会受到影响。所以，即使 ADO 命令成功执行，这个集合中也可能含有错误信息。
- OLE DB 提供者可能会将包含信息的信息或警告装入 Errors 集合，错误号为 0。所以不能只检查集合中的错误号而假定错误已经发生。比如，使用 ODBC 提供者与 SQL Server 连接，可能会得到一个“错误”，告知缺省的数据库已经被改变了。

如果回头看一下本章开始讲到的对象模型，可能会发现 Errors 集合只能由 Connection 对象访问。读者可能会觉得奇怪，如果不显式地创建一个 Connection 对象，如何访问 Errors 集合？Recordset 对象有一个 ActiveConnection 属性，含有当前记录集的 Connection 对象，这意味着可以这样得到 Errors 集合：

```
rsAuthors.ActiveConnection.Errors
```

如果想看发生的全部错误，则需要遍历整个 Errors 集合：

```
For Each errAuthors In rsAuthors.ActiveConnection.Errors
    'Display error
Next
```

为了显示一些合理的错误信息，需要确切地知道在 Errors 集合中到底有些什么。Error 对象包含表 8-5 所示的属性。

表8-5 Error对象的属性及说明

属 性	说 明
Number	ADO 错误号
NativeError	从数据提供者获得的错误号
SQLState	连接到 SQL 数据库时，5 位的 SQL 状态代码
Source	引起错误的对象
Description	错误说明文本

这意味着循环过程现在可以变成这样：

```
For Each errAuthors In rsAuthors.ActiveConnection.Errors

    Response.Write "Number: " & errAuthors.Number & _
        "<BR>NativeError: " & errAuthors.NativeError & _
        "<BR>SQLState: " & errAuthors.SQLState & _
        "<BR>Source: " & errAuthors.Source & _
        "<BR>Description: " & errAuthors.Description & _
        "<P>"

Next
```

8.5.2 ASP 页面中的 ADO 错误

在第 4、7 章，我们研究了 ASP 的错误，以及如何简洁并彻底地处理这些错误。ASP 3.0 的一个新特征就是自定义错误页面，但这对于 ADO 确实用处不大，因为脚本中的变量无法传入自定义的错误页面。这就意味着我们无法检测 Errors 集合。

面对这样的情况，必须提供自己的错误处理。如果你使用 JScript 作为服务器端编程语言，那么你将拥有新的 try/catch 特性，但是 VBScript 对于错误的处理仍然有许多不足。目前，最好的检测错误的方法是使用 On Error Resume Next 语句，然后在可能会引起错误的每一行 ADO 代

码后检查Errors集合。就像这样：

```
<%
    On Error Resume Next

    Dim rsAuthors
    Dim strSQL

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

    strSQL = "SELECT MissingColumn1, MissingColumn2, au_lname, au_fname " & _
        " FROM authors"

    rsAuthors.Open strSQL, strConn, adOpenDynamic, adLockOptimistic, adCmdText

    If CheckForErrors (rsAuthors.ActiveConnection) = False Then
        While Not rsAuthors.EOF
            Response.Write rsAuthors("au_lname") & ", " & _
                rsAuthors("au_fname") & "<BR>"
            rsAuthors.MoveNext
        Wend
    End If

    rsAuthors.Close
    Set rsAuthors = Nothing
%>
```

这里可使用CheckForErrors子程序来检测是否有错误发生：

```
Function CheckForErrors(objConn)

    Dim objError      ' Error object

    ' Errors means the count will be greater than 0
    If objConn.Errors.Count > 0 Then

        ' Loop through the errors
        For Each objError in objConn.Errors

            ' Errors with number 0 are informational
            If objError.Number <> 0 Then
                Response.Write "<TABLE BORDER=1>" & _
                    "<TR><TD>Error Property</TD><TD>Contents</TD>" & _
                    "</TR><TR><TD>Number</TD><TD>" & objError.Number & _
                    "</TD></TR><TR><TD>NativeError</TD><TD>" & _
                    objError.NativeError & "</TD></TR>" & _
                    "<TR><TD>SQLState</TD><TD>" & objError.SQLState & _
                    "</TD></TR><TR><TD>Source</TD><TD>" & _
                    objError.Source & "</TD></TR>" & _
                    "<TR><TD>Description</TD><TD>" & _
                    objError.Description & "</TD></TR></TABLE><P>"

                CheckForErrors = True
            End If

        Next

    Else

        CheckForErrors = False
    End If

End Function
```

这个程序检测是否有错误，如果有，则为每一个错误创建一个表格，并给出了如图 8-11 所示的结果。

Error Property	Contents
Number	-2147217900
NativeError	207
SQLState	42S22
Source	Microsoft OLE DB Provider for SQL Server
Description	Invalid column name 'MissingColumn1'.

Error Property	Contents
Number	-2147217900
NativeError	207
SQLState	42S22
Source	Microsoft OLE DB Provider for SQL Server
Description	Invalid column name 'MissingColumn2'.

图8-11 显示的错误结果

这并不是一个技术含量较高的解决方案，但确实是用 VBScript所能做到的最好的解决方案。真正的不足是必须自己检测错误。

## 8.6 小结

本章讨论了许多内容，包括下面几个重要的方面：

- ADO如何配合ASP。
- 记录集结构基础。
- 如何访问一些基本的数据存储，如何处理从数据存储中取得的数据。
- 如何管理数据存取时的错误。

现在，应该扩展基础知识，研究 Command对象，理解如何使用一些高级特性去改进性能和维护ASP页面。