

## 第9章 连接、命令和过程

上一章讨论了ADO的基础知识，内容主要涉及 Recordset对象以及对数据的处理。在大多数例子中，只是通过指定数据库的表名来获取数据，但正如从对象模型中看到的，ADO还有其他允许访问数据的对象。

本章将要更详细地介绍这些对象，特别将研究以下内容：

- Connection对象，以及如何用它来获取数据和运行命令。
- Command对象，为什么需要该对象及其所具有的功能。
- 如何运行存储过程，特别是那些需要参数的存储过程。
- 一些简单的改善ADO性能的优化技巧。
- 数据整形的概念及如何使用。

如同介绍Recordset对象那样，我们不打算覆盖所涉及对象的全部方法和属性。在这里只探讨其中最重要的主题，以及那些适用于ASP开发人员的方法与属性。

### 9.1 Connection对象

前一章中已经提及，Connection对象是为我们与数据存储提供连接的对象，但这并非Connection对象的全部功能。除了存储连接的细节外（比如数据存储的类型，以及其支持的特性），也可以利用Connection对象运行命令。

这些命令可能是查询动作，比如更新、插入或删除操作，也可以是返回一个记录集的命令。读者可能会觉得奇怪：既然有了Recordset对象，这又有什么用？这正是ADO的灵活性所在，可以根据当前的情况，以及对当前任务的适用性选择使用任一种对象。

从Connection对象运行的命令一般是查询动作，但了解能够得到返回的记录集也是非常有用的。

#### 9.1.1 返回一个记录集

为了从Connection对象返回一个记录集，要使用Execute方法。语法是：

Connection.Execute CommandText, [RecordsAffected], [Options]

参数及说明如表9-1所示。

表9-1 Connection对象的Execute方法的参数及说明

参 数	说 明
CommandsText	执行的命令文本。与Recordset的Open方法中的Source参数相同，也能代表一个现有的Command对象
RecordsAffected	受命令执行影响的记录数
Options	命令选项，可以是一个或多个 CommandTypeEnum或ExecuteOptionEnum常数，详细的值请见上一章

Execute方法可选择地返回一个记录集，在这种情况下只要将返回值赋给记录集变量。例如：

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

```
conPubs.Open strConn
```

```
Set rsAuthors = conPubs.Execute ("Authors")
```

读者可能会奇怪使用 Connection对象的Execute方法与使用Recordset对象的Open方法之间到底有什么区别？看上去区别不是很大，使用 Recordset对象的Open方法可以改变光标类型和锁定类型。这些选项对于 Connection对象的Execute方法是不可用的，因此永远只能得到一个只能前移的、只读的记录集。

### 9.1.2 操作命令

如果正在运行操作命令，比如一个 SQL UPDATE语句，那么可以使用 RecordsAffected参数找出有多少条记录受到该命令的影响。例如：

```
Dim strSQL As String
```

```
Dim lngRecs As Long
```

```
strSQL = "UPDATE Titles SET Price = Price * 1.10" & _  
        " WHERE Type='Business'"
```

```
conPubs.Execute strSQL, lngRecs, adCmdText
```

```
Response.Write lngRecs & " records were updated."
```

上述代码将所有类型为 Business的书的单价增加了 10%。一旦Execute命令执行完毕，受Update命令影响的记录数就返回到变量 lngRecs中，这就是RecordsAffected参数。

注意，已经为命令指定了 adCmdText选项，告诉ADOCommandText参数是一个文本命令。一般这不是严格必须的，其目的只是让 ADO预先知道执行的命令属于何种类型，这样能够提高性能。记住，这个值可以是一个或多个 CommandTypeEnum值。

无记录集返回

如果上面的例子不返回记录集，那么在 Execute语句中加入另一个选项也是较好的方法：

```
conPubs.Execute strSQL, lngRecs, adCmdText + adExecuteNoRecords
```

使用adExecuteNoRecords告诉ADO执行的命令不返回任何记录。所以，ADO不必费心去创建一个记录集。如果省略了该选项，那么 ADO将会创建一个空的记录集，很明显这浪费了时间，因此加上这个选项将会加快语句的执行。

## 9.2 Command对象

Command对象特定地为处理各种类型的命令而设计，特别是那些需要参数的命令。与 Connection对象相似，Command对象可以运行返回记录集和不返回记录集两种类型的命令。实际上，如果命令不含有参数，那么它并不关心是使用 Connection对象，还是Command对象，还是Recordset对象。

### 9.2.1 返回记录集

对于一个返回记录集的命令，可使用 Execute方法。然而，与 Connection对象不同，必须

使用CommandText属性，而不能在Execute方法中使用命令文本。

```
Set cmdAuthors = Server.CreateObject("ADODB.Command")

cmdAuthors.CommandText = "Authors"

Set rsAuthors = cmdAuthors.Execute
```

这是告诉Command对象去执行一个简单的、返回一个只读记录集的命令的最简便方法。Execute方法也有一些可选参数，如表9-2所示。

表9-2 Command对象的Execute方法的参数及说明

参 数	说 明
RecordsAffected	受命令影响的记录数
Parameters	参数值数组
Options	命令选项，与Recordset的Open方法中的Options选项相似

参数RecordsAffected与Options同前面解释的一样，另外也可以使用 CommandType属性设置命令类型：

```
Set cmdAuthors = Server.CreateObject("ADODB.Command")

cmdAuthors.CommandText = "Authors"
cmdAuthors.CommandType = adCmdTable

Set rsAuthors = cmdAuthors.Execute
```

如果不设置其他参数，也可以在Execute行上设置，必须为它们使用逗号：

```
Set rsAuthors = cmdAuthors.Execute(, , adCmdTable)
```

在本章后面处理存储过程和参数时，将会看到参数Parameters的用途。

#### 改变光标类型

值得注意的是，使用Execute方法返回的记录集具有缺省的光标和锁定类型。这意味着这是只能前移的、只读的记录集。虽然使用Execute方法不能改变这种情况，但对这个问题有一个解决的方法。

如果需要使用一个命令，并且要求不同的光标和锁定类型，那么应该使用Recordset的Open方法，此时Command对象作为Recordset的数据源。例如：

```
cmdAuthors.ActiveConnection = strConn
cmdAuthors.CommandText = "Authors"
cmdAuthors.CommandType = adCmdTable

rsAuthors.Open cmdAuthors, , adOpenDynamic, adLockOptimistic
```

注意，在Open命令行中忽略了连接细节，因为连接设置在Command对象中。连接细节在命令打开前已经设置在Command对象的ActiveConnetion属性中。

### 9.2.2 操作命令

对于操作命令，比如那些无记录返回的更新命令，整个过程相似，只需移去设置记录集的代码：

```
Set cmdUpdate = Server.CreateObject("ADODB.Command")
```

```
strSQL = "UPDATE Titles SET Price = Price * 1.10" & " WHERE Type='Business'"
```

```
cmdUpdate.ActiveConnection = strConn
```

```
cmdUpdate.CommandText = strSQL
```

```
cmdUpdate.CommandType = adCmdText
```

```
cmdUpdate.Execute , , adExecuteNoRecords
```

注意，我们在此设置了命令类型，然后在 Execute 行中增加了额外的设置选项。这段代码运行 UPDATE 命令，并且保证不会创建新的记录集。

### 9.2.3 存储过程

存储过程的使用是 Command 对象得到应用的一个领域。存储过程（有时也称存储查询）是存储在数据库中预先定义的 SQL 查询语句。

为什么应该创建和使用存储过程而不是在代码中直接使用 SQL 字符串呢？主要有以下几个理由：

- 存储过程被数据库编译过。这样可以产生一个“执行计划”，因此数据库确切地知道它将做什么，从而加快了过程的执行速度。
- 存储过程通常被数据库高速缓存，这样使它们运行得更快，因为此时不需从磁盘中读取它们。并非所有的数据库都支持这种缓存机制，比如微软的 Access 就不支持，而 SQL Server 却支持。
- 通过指定数据库中的表只能被存储过程修改，可以确保数据更安全。这意味着具有潜在危险的 SQL 操作不会执行。
- 可以避免将 ASP 代码和冗长的 SQL 语句混在一起，从而使 ASP 代码更易于维护。
- 可以将所有 SQL 代码集中存放于服务器。
- 可以在存储过程中使用输出参数，允许返回记录集或其他的值。

一般说来，存储过程几乎总是比相当的 SQL 语句执行速度快。

为了使用存储过程，只要将存储过程的名字作为命令文本，并设置相应的类型。例如，考虑前面更新书价的例子。如果在 SQL Server 上创建一个存储过程，可以编写代码：

```
CREATE PROCEDURE usp_UpdatePrices
```

```
AS
```

```
    UPDATE Titles
```

```
    SET Price = Price * 1.10
```

```
    WHERE Type='Business'
```

对于微软的 Access 数据库，可以使用一个简单的更新查询语句完成相同的任务，如图 9-1 所示。

要在 ASP 网页中运行该存储过程，只需使用以下代码：

```
Set cmdUpdate = Server.CreateObject("ADODB.Command")
```

```
cmdUpdate.ActiveConnection = strConn
```

```
cmdUpdate.CommandText = "usp_UpdatePrices"
```

```
cmdUpdate.CommandType = adCmdStoredProc
```

```
cmdUpdate.Execute , , adExecuteNoRecords
```

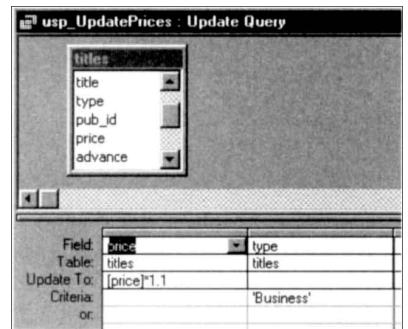


图9-1 使用微软的 Access 数据库完成更新查询

这只是运行存储过程。没有记录集返回，因为只是在更新数据。需要记住的是，除非确

实需要，不要创建记录集。

虽然这样做也可以，但并不是很灵活，因为仅仅处理一种类型的书。更好的做法是创建一个允许我们选择书类型的过程，这样就不必为每类书创建一个过程。同样也可去掉固定的10%的更新，这样使得灵活性更好。那么，如何才能做到这一点呢，很简单，使用参数。

### 1. 参数

存储过程的参数(或变量)与一般的过程和函数的参数一样，可以传到函数内部，然后函数可以使用它的值。SQL Server(其他数据库也一样，包括 Access)中的存储过程都具有这样的功能。

为了使存储过程能处理多种类型的书，甚至允许用户指定价格的增加(或减少)，需要增加一些参数：

```
CREATE PROCEDURE usp_UpdatePrices
    @Type      Char(12),
    @Percent    Money
AS
    UPDATE Titles
    SET   Price = Price * (1 + @Percent / 100)
    WHERE Type = @Type
```

现在，存储过程usp\_UpdatePrices带有两个参数：

- 一个是书的类型(@Type)。
- 一个是书价变化的百分比(@Percent)。

与VBScript的函数一样，这些参数都是变量。然而，与VBScript和其他脚本语言不同是：在这些脚本语言中的变量都是variant类型，而SQL变量具有确定的类型(char、Money等等)。必须遵守SQL变量的命名规范，即变量必须以符号@开始。

注意，我们让百分数作为一个整数(如10代表10%)，而不是作为一个分数值传入此过程。这只是让存储过程变得更直观一些。

### 2. Parameters集合

那么，现在有了带参数的存储过程，但如何通过ADO来调用它呢？我们已经见到了如何用Command对象调用不带参数的存储过程，实际上，它们之间并没有什么不同。不同之处在于Parameters集合的使用。

Parameters集合包含存储过程中每个参数的Parameter对象。然而，ADO并不会自动地知道这些参数是什么，因此，必须用CreateParameter方法创建它们，采用下面的形式：

```
Set Parameter = Command.CreateParameter (Name, [Type], [Direction],
                                           [Size], [Value])
```

参数及说明如表9-3所示。

表9-3 CreateParameter方法的参数及说明

参 数	说 明
Name	参数名。这是Parameters集合中的参数名，不是存储过程中的参数名。然而，使用相同的名字是一个好的做法
Type	参数的数据类型。可以是一个adDataType常数，详见附录
Direction	参数的方向，指明是参数向存储过程提供信息，还是存储过程向ADO返回信息。可以是下面的值之一：

(续)

参 数	说 明
	adParamInput, 参数是传给存储过程的输入参数
	adParamOutput, 参数是从存储过程检索出的输出参数
	adParamInputOutput, 参数可同时作为输入和输出参数
	adParamReturnValue, 该参数包含存储过程返回的状态
Size	参数长度。对于固定长度的类型, 比如整型, 该值可以忽略
Value	参数的值

一旦创建了参数就可以将其追加到 Parameters集合中, 例如:

```
Set parValue = cmdUpdate.CreateParameter("@Type", adVarChar, adParamInput, _
12, "Business")
cmdUpdate.Parameters.Append parValue

Set parValue = cmdUpdate.CreateParameter("@Percent", adCurrency, _
adParamInput, , 10)
cmdUpdate.Parameters.Append parValue
```

没有必要显式地创建一个对象去保存参数, 缺省的 Variant类型已经可以工作得相当好。如果不想创建一个变量, 也可以走捷径, 例如下面的代码:

```
cmdUpdate.Parameters.Append = _
cmdUpdate.CreateParameter("@Percent", adCurrency, adParamInput, , 10)
```

这使用 CreateParameter方法返回一个 Parameter对象, 并用 Append方法接收它。这种方法比使用变量运行得快, 却加长了代码行, 可读性比较差。可以根据自己的爱好选择其中一种方法。

参数加到 Parameters集合后, 就保留在其中, 因此, 不一定在创建参数时就为每个参数赋值。可以在命令运行前的任何时候设置参数的值。例如:

```
cmdUpdate.Parameters.Append = _
cmdUpdate.CreateParameter ("@Percent", adCurrency, adParamInput)

cmdUpdate.Parameters("@Percent") = 10
```

前一章提到了访问集合中的值有好几种方法, Parameters集合并没有什么不同。上面的例子使用参数的名字在集合中检索参数, 也可以使用索引号进行检索:

```
cmdUpdate.Parameters(0) = 10
```

以上代码对参数集合中第一个 (Parameters集合从0开始编号)参数进行了赋值。使用索引号比使用名字索引速度快, 但很显然使用名字使代码更易读。

重点注意 Parameters集合中参数的顺序必须与存储过程中参数的顺序相一致。

运行带参数的命令

一旦加入参数, 就可立即运行命令, 同时这些参数的值传入存储过程。现在可用一个友好的网页去更新用户选择的类型的书价。例如, 假设有一个名为 UpdatePrices.asp的网页, 其运行时的界面如图9-2所示

通过从数据库中获取书类型的列表, 可以很轻松地动态创建该页面。首先要做的是包含文件 Connection.asp, 该文件包含了连接字符串 (保存在变量 strConn中) 以及对 ADO常数的引用,



这在前面的章节已经讨论过。

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

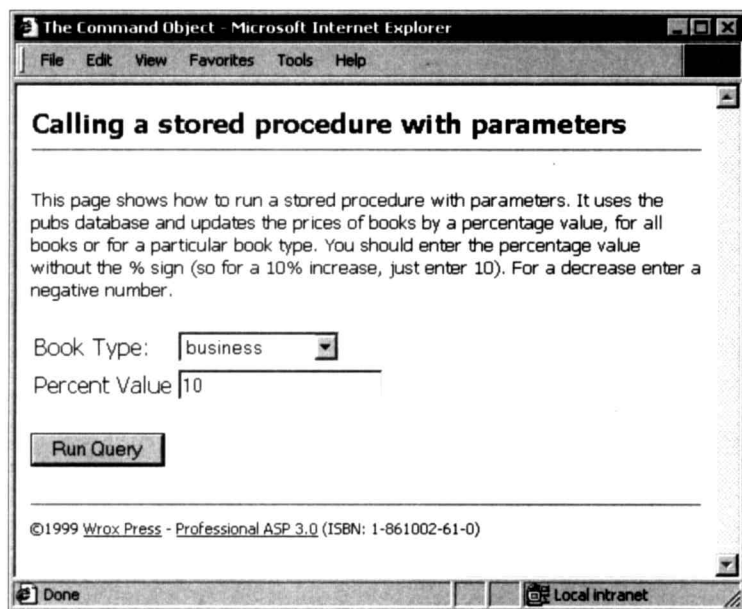


图9-2 UpdatePrices.asp网页运行时的界面

接下来，可以创建一个窗体(在这儿不显示大量文本，仅仅用一个样本文件)。该窗体调用一个名为StoredProcedure.asp的文件。

```
<FORM NAME="UpdatePrices" Method="Post" ACTION="StoredProcedure.asp">
<TABLE>
  <TR>
    <TD>Book Type:</TD>
    <TD>
      <SELECT NAME="lstTypes"></TD>

```

现在开始编写ASP脚本从titles表中读取书的类型。使用一个SQL字符串只返回唯一的书类型，然后将返回值放到HTML的OPTION标记中：

```
<%
  Dim rsTypes
  Dim strSQL
  Dim strQuote

  ' Predefine the quote character
  strQuote = Chr(34)

  ' Create a recordset of the types
  Set rsTypes = Server.CreateObject("ADODB.Recordset")

  strSQL = "SELECT DISTINCT type FROM titles"
  rsTypes.Open strSQL, strConn

  ' Create the book types
  While Not rsTypes.EOF
    Response.Write "<OPTION VALUE=" & strQuote & _

```

```

        rsTypes("Type") & strQuote & ">" & rsTypes("Type")
    rsTypes.MoveNext
Wend

rsTypes.Close
Set rsTypes = Nothing
%>

```

显示书的类型后，接着可以构建窗体的其他部分，包括一个允许用户输入书价变化百分数的文本框。

```

        </SELECT>
    </TD>
</TR>
<TR>
    <TD>Percent Value</TD>
    <TD><INPUT NAME="txtPercent" TYPE="TEXT"></TD>
</TR>
</TABLE>
<P>
<INPUT TYPE="Submit" VALUE="Run Query">
</FORM>

```

现在看一下 Run Query 按钮调用的 ASP 文件 StoredProcedure.asp。首先，声明变量并从调用窗体取出书的类型和百分数。

```

<%
Dim cmdUpdate
Dim lngRecs
Dim strType
Dim curPercent

' Get the form values
strType = Request.Form("lstTypes")
curPercent = Request.Form("txtPercent")

```

现在可以向用户显示一些确认信息，告诉他们将发生什么。

```

' Tell the user what's being done
Response.Write "Updating all books" & _
    " of type <B>" & strType & "</B>" & _
    " by " & curPercent & "%<P>"

```

现在重新回到代码内部，在此创建 Command 对象和参数。

```

Set cmdUpdate = Server.CreateObject("ADODB.Command")

' Set the properties of the command
With cmdUpdate
    .ActiveConnection = strConn
    .CommandText = "usp_UpdatePrices"
    .CommandType = adCmdStoredProc

```

利用从前面网页的窗体中提取的数据值，使用快捷方法创建和增加参数。

```

' Add the parameters
.Parameters.Append .CreateParameter("@Type", adVarChar, adParamInput, _
    12, strType)
.Parameters.Append .CreateParameter("@Percent", adCurrency, _
    adParamInput, , curPercent)

```

现在，运行存储过程。

```

' Execute the command
.Execute lngRecs, , adExecuteNoRecords

```



End With

为了确认，可以告诉用户已经更新多少条记录。

```
' And finally tell the user what's happened
Response.Write "Procedure complete. " & lngRecs & " records were updated."
```

```
Set cmdUpdate = Nothing
```

```
%>
```

这样就有了两个简单页面。前者创建了一个供选择的项目列表，后者使用其中某个项目值更新数据。这是许多需要显示和更新数据的 ASP 页面的基础。

### 3. 传递数组参数

Parameters 参数集合一般来说比较好用，但有时稍有麻烦（尤其对于新手）。好在有一种快捷方法，使用 Execute 方法的 Parameters 参数。例如，调用存储过程 usp\_UpdatePrices，但不使用 Parameters 集合。

创建一个 Command 对象，并同前面一样设置其属性。

```
' Set cmdUpdate = Server.CreateObject("ADODB.Command")
```

```
' Set the properties of the command
```

```
With cmdUpdate
```

```
.ActiveConnection = strConn
```

```
.CommandText = "usp_UpdatePrices"
```

```
.CommandType = adCmdStoredProc
```

但这里正是差异所在。我们仅是通过 Execute 方法传递参数给存储过程，而不是创建参数并添加到集合中。

```
' Execute the command
```

```
.Execute lngRecs, Array(strType, curPercent), adExecuteNoRecords
```

```
End With
```

这里使用了 Array 函数，将单个变量转换为数组，以适于方法调用。这种方法当然也有缺点：

- 只能使用输入参数。因为不能指定参数的类型和传递方向，而缺省为输入参数。
- 如果要多次调用存储过程，这种方法速度就比较慢，因为 ADO 将向数据库存储询问参数的内容及数据类型。

集合方法和数组方法之间在速度上的差异非常之小，几乎可以忽略。所以，如果只有输入参数，可随便使用哪一种。实际上，人们更喜欢使用 Parameters 集合的方法，尽管它稍为繁琐，但是使参数的属性更加明确。

### 4. 输出参数

我们已经知道如何获得受命令影响的记录数，如果需要更多信息，却又不想返回一个记录集，怎么办？也许想从存储过程中返回两个或三个值，但又不想费心创建一个记录集。在这时，可以定义一个输出参数，其值由存储过程提供。

例如，对于更新书价的程序，如果想在更新之后找出最高价格，可将存储过程改成：

```
CREATE PROCEDURE usp_UpdatePricesMax
```

```
@Type Char(12),
```

```
@Percent Money,
```

```
@Max Money OUTPUT
```

```
AS
```

```

BEGIN
    UPDATE Titles
    SET     Price = Price * (1 + @Percent / 100)
    WHERE  Type=@Type

    SELECT @Max = MAX(Price)
    FROM    Titles
END

```

这只是在执行更新后运行了一个简单的 SELECT 语句，并将值赋给输出参数。

现在可以改写 StoredProcedure.asp 的代码从而获取变量 @MAX 的值。

```

<%
    Dim cmdUpdate
    Dim lngRecs
    Dim strType
    Dim curPercent
    Dim curMax

    ' Get the form values
    strType = Request.Form("lstTypes")
    curPercent = Request.Form("txtPercent")

    ' Tell the user what's being done
    Response.Write "Updating all books" & _
        " of type <B>" & strType & "</B>" & _
        " by " & curPercent & "%<P>"

    Set cmdUpdate = Server.CreateObject("ADODB.Command")

    ' Set the properties of the command
    With cmdUpdate
        .ActiveConnection = strConn
        .CommandText = "usp_UpdatePricesMax"
        .CommandType = adCmdStoredProc

```

我们只是在集合中加入了另一个参数，但这次指定为输出参数。注意它并没有赋值，因为其值将由存储过程提供，记住这是一个输出参数。

```

    ' Add the parameters
    .Parameters.Append .CreateParameter ("@Type", adVarChar, adParamInput, _
        12, strType)
    .Parameters.Append .CreateParameter ("%Percent", adCurrency, _
        adParamInput, , curPercent)
    .Parameters.Append .CreateParameter ("%Max", adCurrency, adParamOutput)

    ' Execute the command
    .Execute lngRecs, , adExecuteNoRecords

```

一旦执行了这个过程，就可从集合中取得该值。

```

    ' Extract the output parameter, which the stored
    ' procedure has supplied to the parameters collection
    curMax = .Parameters("@Max")
End With

' And finally tell the user what's happened
Response.Write "Procedure complete. " & lngRecs & _
    " records were updated.<P>"
Response.Write "The highest price book is now " & _
    FormatCurrency(curMax)

```

```
Set cmdUpdate = Nothing
%>
```

如果有不止一个输出参数，可用相同的方法访问。可以使用参数名或索引号取出集合中的值。

### 5. 返回值

对函数返回值的处理不同于存储过程返回值的处理，这常常导致混淆。在函数中，经常是返回一个布尔值来表明函数运行的成功与否。

```
If SomeFunctionName() = True Then
    ' Function succeeded
```

但在调用一个存储过程时，却不能使用同样的方法，因为存储过程是用 Execute 方法运行的，同时返回一个记录集。

```
Set rsAuthors = cmdAuthors.Execute
```

如果得不到一个返回值，如何确定是否已正确执行存储过程？当发生错误时，会报告错误，这样就可使用前一章提供的错误处理代码来处理错误。但对于一些非致命的逻辑错误怎么办？

例如，考虑向 employee 表添加一个新职员的情形。你可能不想防止两个职员同名的情况，但想注明这个情况。那么，可以使用一个返回值以表明是否已有同名的职员存在。存储过程如下：

```
CREATE PROCEDURE usp_AddEmployee
    @Emp_ID      Char(9),
    @FName       Varchar(20),
    @MInit       Char(1),
    @LName       Varchar(30),
    @Job_Id      SmallInt,
    @Job_Lvl     TinyInt,
    @Pub_ID      Char(4),
    @Hire_Date   Datetime
AS
BEGIN
    DECLARE @Exists      Int          -- Return value

    -- See if an employee with the same name exists
    IF EXISTS(SELECT *
              FROM Employee
              WHERE FName = @FName
              AND   MInit = @MInit
              AND   LName = @LName)
        SELECT @Exists = 1
    ELSE
        SELECT @Exists = 0

    INSERT INTO Employee (emp_id, fname, minit, lname,
                        job_id, job_lvl, pub_id, hire_date)
    VALUES (@Emp_Id, @FName, @MInit, @LName, @Job_ID,
            @Job_Lvl, @Pub_ID, @Hire_Date)

    RETURN @Exists
END
```

该过程首先检查是否有同名的职员存在，并据此设定相应的变量 Exists，若存在同名，就设为1，否则为0。然后将该职员加到表中，同时把 Exists 的值作为返回值返回。

注意尽管返回了一个值，但并未将其声明为存储过程的参数。

调用该过程的ASP代码如下：

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

    Set cmdEmployee = Server.CreateObject("ADODB.Command")

    ' Set the properties of the command
    With cmdEmployee
        .ActiveConnection = strConn
        .CommandText = "usp_AddEmployee"
        .CommandType = adCmdStoredProc

    ' Create the parameters
    ' Notice that the return value is the first parameter
    .Parameters.Append .CreateParameter ("RETURN_VALUE", adInteger, _
        adParamReturnValue)
    .Parameters.Append .CreateParameter ("@Emp_Id", adChar, adParamInput, 9)
    .Parameters.Append .CreateParameter ("@FName", adVarChar, _
        adParamInput, 20)
    .Parameters.Append .CreateParameter ("@MInit", adChar, adParamInput, 1)
    .Parameters.Append .CreateParameter ("@LName", adVarChar, _
        adParamInput, 30)
    .Parameters.Append .CreateParameter ("@Job_Id", adSmallInt, adParamInput)
    .Parameters.Append .CreateParameter ("@Job_Lvl", adUnsignedTinyInt, _
        adParamInput)
    .Parameters.Append .CreateParameter ("@Pub_ID", adChar, adParamInput, 4)
    .Parameters.Append .CreateParameter ("@Hire_Date", adDBTimeStamp, _
        adParamInput, 8)

    ' Set the parameter values
    .Parameters("@Emp_Id") = Request.Form("txtEmpID")
    .Parameters("@FName") = Request.Form("txtFirstName")
    .Parameters("@MInit") = Request.Form("txtInitial")
    .Parameters("@LName") = Request.Form("txtLastName")
    .Parameters("@Job_ID") = Request.Form("lstJobs")
    .Parameters("@Job_Lvl") = Request.Form("txtJobLevel")
    .Parameters("@Pub_ID") = Request.Form("lstPublisher")
    .Parameters("@Hire_Date") = Request.Form("txtHireDate")

    ' Run the stored procedure
    .Execute lngRecs, , adExecuteNoRecords

    ' Extract the return value
    lngAdded = .Parameters("RETURN_VALUE")
End With

Response.Write "New employee added.<P>"
If lngAdded = 1 Then
    Response.Write "An employee with the same name already exists."
End If

Set cmdEmployee = Nothing
%>
```

需要重点注意，返回值应当作为集合中第一个参数被创建。即使返回值并不作为一个参数出现在存储过程中，总是Parameters集合中的第一个Parameters。

因此，特别强调一点：

存储过程的返回值必须声明为 Parameters 集合中第一个参数，同时参数的 Direction 值必须为 adParamReturnValue。

使用返回值

现在定义一个初始窗体，如图 9-3 所示。

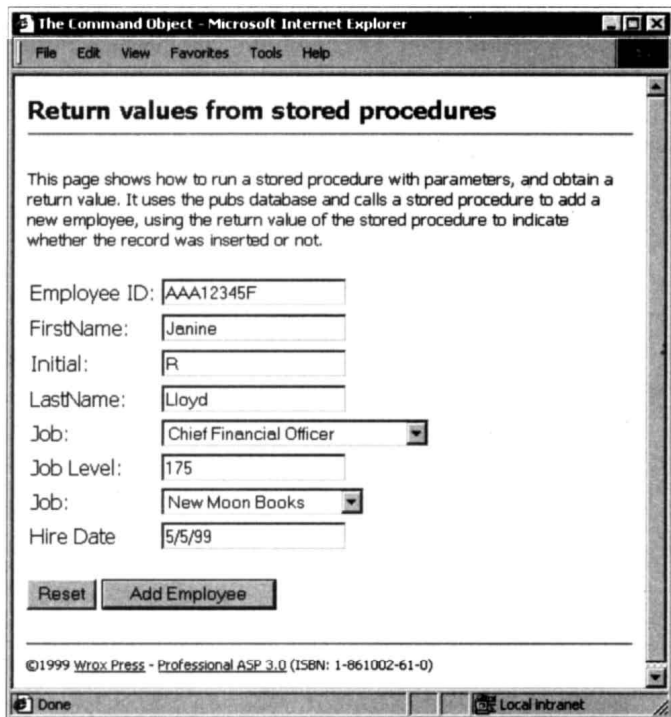


图9-3 初始窗体界面

按下 Add Employee 按钮会产生如图 9-4 所示的显示。

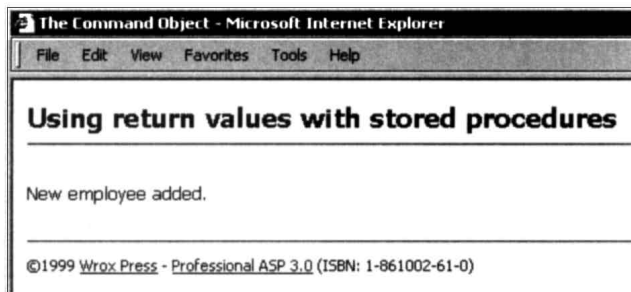


图9-4 按下 Add Employee 按钮后显示的界面

再添加同样的细节 (ID 号不同) 会得到如图 9-5 所示的显示。

#### 6. 更新参数

无需输入所有的参数细节，只需调用 Refresh 方法，就能让 ADO 完成更新。例如，假设已经创建了一个带有与前面例子相同的参数的过程 usp\_AddEmployee，并且没有改变运行的页面。

```
With cmdEmployee
    .ActiveConnection = strConn
    .CommandText = "usp_AddEmployee"
    .CommandType = adCmdStoredProc
```

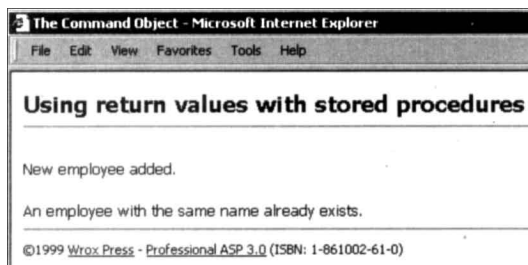


图9-5 添加细节后显示的界面

然后调用Refresh方法。

```
.Parameters.Refresh
```

这告诉ADO向数据存储请求每个参数的细节，并创建Parameters集合。然后可以为其赋值。

```
.Parameters("@Emp_Id") = Request.Form("txtEmpID")
.Parameters("@FName") = Request.Form("txtFirstName")
.Parameters("@MInit") = Request.Form("txtInitial")
.Parameters("@LName") = Request.Form("txtLastName")
.Parameters("@Job_ID") = Request.Form("lstJobs")
.Parameters("@Job_Lvl") = Request.Form("txtJobLevel")
.Parameters("@Pub_ID") = Request.Form("lstPublisher")
.Parameters("@Hire_Date") = Request.Form("txtHireDate")
```

注意并不需要创建任何参数，包括返回值。

这似乎真是一条捷径，但应意识到这种方法也造成了性能上的损失，因为 ADO必须向提供者查询以获得存储过程的参数细节。尽管如此，这种方法还是很有用的，尤其是在从参数中取出正确的值有困难的时候。

实际上，可以编写一个小实用程序作为开发工具使用，用来完成更新并建立Append语句，可以将其粘贴到自己的代码中。它看上去应该与图9-6所示的GenerateParameters.asp ASP页面类似。

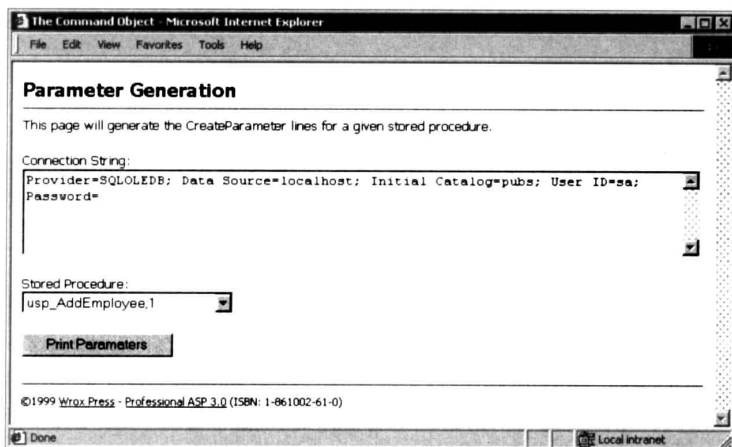


图9-6 GenerateParameters.asp ASP页面

其代码相当简单。首先是包含连接字符串和另一个 ADOX 常数文件。

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

接下来创建一个窗体，指定目标为 PrintParameters.asp ASP 页面。

```
<FORM NAME="Procedures" METHOD="Post" ACTION="PrintParameters.asp">
  Connection String:<BR>
  <TEXTAREA NAME="txtConnection" COLS="80" ROWS="5">
    <% = strConn %>
  </TEXTAREA>
  <P>
  Stored Procedure:<BR>
  <SELECT NAME="lstProcedures">
```

然后，使用 ADOX 从 SQL Server 中得到存储过程的列表，同时创建一个含有这些存储过程名字的列表框。

```
<%
  Dim catPubs
  Dim procProcedure
  Dim strQuote

  ' Predefine the quote character
  strQuote = Chr(34)

  Set catPubs = Server.CreateObject("ADOX.Catalog")

  catPubs.ActiveConnection = strConn
  For Each procProcedure In catPubs.Procedures
    Response.Write "<OPTION VALUE=" & _
      strQuote & procProcedure.Name & _
      strQuote & ">" & procProcedure.Name
  Next

  Set procProcedure = Nothing
  Set catPubs = Nothing
%>
</SELECT>
<P>
<INPUT TYPE="Submit" VALUE="Print Parameters">
</FORM>
```

这是一个简单的窗体，包括一个用于显示连接字符串的 TEXTAREA 控件和用于显示存储过程名称的 SELECT 控件。以前没有见过的是 ADOX，ADOX 是数据定义与安全的 ADO 扩展，可以用来访问数据存储的目录（或是元数据）。

本书不打算 ADOX 的内容，但其十分简单。进一步的细节可参见《ADO Programmer's Reference》，Wrox 出版社出版，2.1 版或 2.5 版都行。

上面的例子使用了 Procedures 集合，这个集合包含数据存储中的所有存储过程的列表。按下 Print Parameters 按钮时，将得到图 9-7 所示的显示。

可以简单地从这里拷贝参数行到代码中。在前面使用了一个以前从未见过的包含文件。该文件包含了几个将 ADO 常数（例如数据类型、参数方向等）转换为字符串值的函数：

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



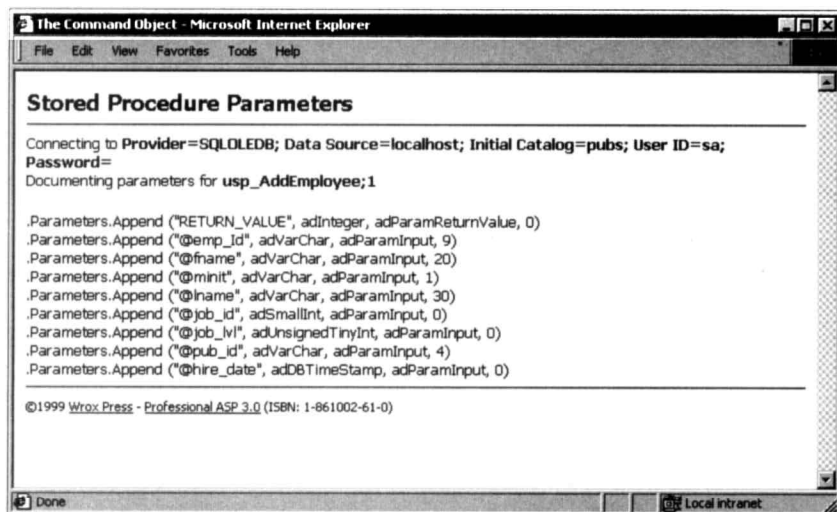


图9-7 按下Print Parameters按钮时显示的界面

接下来，定义一些变量，提取用户请求并创建 Command对象。

```
<%
Dim cmdProc
Dim parP
Dim strConnection
Dim strProcedure
Dim strQuote

' Get the connection and procedure name from the user
strQuote = Chr(34)
strConnection = Request.Form("txtConnection")
strProcedure = Request.Form("lstProcedures")

' Update the user
Response.Write "Connecting to <B>" & strConnection & "</B><BR>"
Response.Write "Documenting parameters for <B>" & _
    strProcedure & "</B><P><P>"

Set cmdProc = Server.CreateObject("ADODB.Command")

' Set the properties of the command, using the name
' of the procedure that the user selected
With cmdProc
    .ActiveConnection = strConnection
    .CommandType = adCmdStoredProc
    .CommandText = strProcedure
```

然后使用Refresh方法自动填写Parameters集合。

```
.Parameters.Refresh
```

现在可以遍历整个集合，写出包含创建参数所需的细节内容的字符串。

```
For Each parP In .Parameters
    Response.Write ".Parameters.Append " & _
        "(" & strQuote & parP.Name & _
        strQuote & ", " & _
        DataTypeDesc(parP.Type) & ", " & _
        ParamDirectionDesc(parP.Direction) & _
```

```
        ", " & _  
        parP.Size & ")<BR>"  
  
    Next  
End With
```

```
Set cmdProc = Nothing  
%>
```

在Descriptions.asp包含文件中可以找到函数 DataTypeDesc和ParamDirectionDesc。

Descriptions.asp包含文件以及其他的例子文件可以在 Web 站点 <http://www.wrox.com> 中找到。

这是一个非常简单的技术，它较好地使用了 Refresh 方法。

### 9.3 优化

优化是每个开发人员应该关心的问题。对于数据库访问，优化是一个关键问题。和其他任务相比，数据的访问显得相对慢些。

因为数据访问的变化是如此之多，以致于几乎不可能提出一套固定的数据库操作的优化规则。通常碰到这类问题，经常得到这样的回答：“这取决于...”，因为这类优化问题取决于准备做什么。

#### 9.3.1 常用的ADO技巧

尽管优化取决于所执行的任务，但是仍然有一些常用的技巧：

- 仅选择所需的列。当打开记录集时，不要自动地使用表名 (即SELECT \*)，除非需要获得所有的列。使用单独的列意味着将减少发送到服务器或从服务器取出的数据的数量。即使需要使用全部列，单独地命名每个列也会获得最佳的性能，因为服务器不必再解释这些列是什么名字。
- 尽可能使用存储过程。存储过程是预先编译的程序，含有一个已经准备好的执行计划，所以比SQL语句执行得更快。
- 使用存储过程更改数据。这总是比在记录集上使用 ADO 方法执行速度快。
- 除非必需否则不要创建记录集。运行操作查询时，要确定加入了 adExecuteNoRecords 选项，这样记录集就不会创建。当仅仅返回一个或两个字段的单行记录时 (比如ID值)，也可以在查询状态下使用这种方法。在这种情况下，存储过程和输出参数将会更快。
- 使用适当的光标和锁定模式。如果所做的全部工作是从记录集中读取数据，并将其显示在屏幕上 (比如，创建一个表)，那么使用缺省的只能前移的、只读的记录集。ADO 用来维护记录和锁定细节的工作越少，执行的性能就越高。

#### 9.3.2 对象变量

当遍历记录集时一个保证能提高性能的方法是使用对象变量指向集合中的成员。例如，考虑下面的遍历含有 Authors 表的记录集的例子。

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

可以用下面的方法加速代码执行，同时使其更易于理解。

```
Set FirstName = rsAuthors("au_fname")
Set LastName = rsAuthors("au_lname")

While Not rsAuthors.EOF
    Response.Write FirstName & " " & LastName & "<BR>"
    rsAuthors.MoveNext
Wend
```

这里使用了两个变量，并指向记录集的 Fidds 集合中的特定字段 (记住，Fidds 集合是缺省的集合)。因为这里建立了一个对象的引用，所以可以使用对象变量而不是实际的变量，这意味着脚本引擎的工作减少了，因为在集合中进行索引的次数变少了。

### 9.3.3 高速缓存大小

高速缓存的大小是指 ADO 每次从数据存储中读取的记录的数量，缺省为 1。这意味着当使用基于服务器的光标时，每当移动到另一条记录时，必须从数据存储中提取记录。举一个例子，如果增大高速缓存的大小为 10，那么每次读入 ADO 缓冲区的记录数将变为 10。如果访问位于高速缓存内的记录，那么 ADO 不需要从数据存储中取记录。当访问位于高速缓存外的记录时则下一批记录将读入到高速缓存中。

通过使用记录集的 CacheSize 属性，可以设置高速缓存的大小。

```
rsAuthors.CacheSize = 10
```

可以在记录集生命期的任何时候改变高速缓存的大小，但新的数量只在提取下一批记录后才有效。

与许多改进性能的技巧类似，高速缓存没有通用的最佳大小，因为它随任务、数据和提供者的不同而改变。但是，从 1 开始增加高速缓存的大小总是能提高性能。

如果你想看到这一点，可以使用 SQL Server Profiler 并查看使用缺省的高速缓存打开一个记录集发生的情况，并比较增大高速缓存后发生的情况。增大高速缓存的大小不仅减低了 ADO 的工作量，同时也降低了 SQL Server 的工作量。

### 9.3.4 数据库设计

不要希望只通过编程来提高对数据的访问效率，应该同时考虑一下数据库的设计。这里并不打算对数据库设计进行更多的讨论，但在使用 Web 站点数据库时应考虑以下几点：

- 实时数据：向用户显示数据时，确保数据内容总是最新是十分重要的。以一份产品目录为例，目录内容改变的频率有多快？如果该目录并非经常改变，那么不必每次都从数据库中提取数据。每周一次，或在数据改变时从数据库产生一个静态的 HTML 页面应是一个更好的办法。
- 索引：如果需要对比进行大量的查询，而不执行太多的添加数据操作，那么可以考虑为表建立索引。
- 不规范化：如果站点有两个不同的目的 (数据维护与数据分析)，那么可以考虑采用一些不规范化的表以便有助于数据的分析。可以提供独立的、完全不规范化的但能正常更新的分析用表，为了改善性能甚至可以将这些分析表移到另一台机器上。
- 数据库统计：如果使用的是 SQL Server 6.x，如果数据被添加或删除，那么应定期更新统计结果。这些统计结果用于产生一个查询计划，会影响查询的运行。请阅读 SQL

Books Online中的UPDATE STATISTIC以便了解更详细的内容。在SQL Server 7.0中这一任务自动完成。

这些都是十分基本的数据库设计技巧，但若只埋头于ASP代码可能不会考虑到这些。

### 9.3.5 数据高速缓存

首先需要注意的是，数据高速缓存与记录集高速缓存虽然都用于改善性能，但两者是无关的。数据高速缓存是临时的数据存储区，允许使用高速缓存中的数据，而不是重新生成新的数据。这只适用于那些不经常改动但多次被访问的数据。

在ASP中一个最简单的缓存数据的方法是使用 Application和Session范围的变量。例如，假设有一些需要选择书类型的网页。正常情况下，可能会创建一个含有以下函数的包含文件。

```
<%
Function BookTypes()

    Dim rsBookTypes
    Dim strQuote

    strQuote = Chr(34)

    Set rsBookTypes = Server.CreateObject("ADODB.Recordset")
    ' Get the book types
    rsBookTypes.Open "usp_BookTypes", strConn

    Response.Write "<SELECT NAME=" & strQuote & lstBookType & strQuote & ">"
    While Not rsBookTypes.EOF
        Response.Write "<OPTION>" & rsBookTypes("Type") & "</OPTION>"
        rsBookTypes.MoveNext
    Wend

    Response.Write "</SELECT>"

    rsBookTypes.Close
    Set rsBookTypes = Nothing

End Function
%>
```

这仅仅是调用一个存储过程，从而得到书的类型，同时创建一个 SELECT列表。上述代码的缺点在于每次调用该函数都必须访问数据库。因此，重新修改这个函数。

```
<%
Function BookTypes()

    Dim rsBookTypes
    Dim strQuote
    Dim strList

    ' See if the list is in the cache
    strList = Application("BookTypes")
    If strList = "" Then
        ' Not cached, so build up list and cache it
        strQuote = Chr(34)

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

        ' Get the book types
        rsBookTypes.Open "usp_BookTypes", strConn

        strList = "<SELECT NAME=" & strQuote & lstBookType & strQuote & ">"
```

```
While Not rsBookTypes.EOF
    strList = strList & "<OPTION>" & rsBookTypes("Type") & "</OPTION>"
    rsBookTypes.MoveNext
Wend
strList = strList & "</SELECT>"

rsBookTypes.Close
Set rsBookTypes = Nothing

' Cache the list
Application("BookTypes") = strList
End If

BookTypes = strList

End Function
%>
```

这段代码不只是打开记录集，它检查 Application 变量 BookType 的值是否为空。如果不为空，则使用该变量的内容。如果为空，则像以前一样打开记录集。显然，一旦第一个人运行了这一例程，便缓存了数据，因此这只对那些不常变化的数据是有用的。

如果想在用户基础上缓存数据，可以使用 Session 范围的变量，但这里必须注意 Session 存在有效期。过期后会话层变量将和会话一起取消，代码便有可能终止运行。

利用 Web Application Stress(WAS)工具，得到了表 9-4 的分析结果。

很明显性能有所改善。但不要采用上述方法缓存一切内容。毕竟，这种方法只适用于那些已经格式化后用于显示的数据。除此之外，还要考虑到如果 Web 服务器只为特定的一个人服务，那几乎不是一个典型的 Web 服务器的用法。使用 WAS 可以在一个服务器上模拟多个用户，这样可以更实际地测试应用程序。

表9-4 利用WAS工具得到的分析结果

方 法	页面点击次数
没有高速缓存	190
有高速缓存	11000

通过模拟一定数量的用户，Web Application Stress 工具可以对 Web 页面进行承受力测试。该工具有一个简单的图形界面，使用起来非常容易。可以从 [http : //homer.rte.microsoft.com/](http://homer.rte.microsoft.com/) 获得更多的信息，也可以下载该工具。

高速缓存对象

若要缓存未格式化的数据该怎么办？可以在不同地方以不同的方式使用吗？当然，也可以用 Application 或 Session 变量这样做。考虑一下书标题的情况。你或许希望在多个页面中使用这个标题，也许在一个表格中显示所有的标题，或在一个列表框中显示供用户选择等等。你可能会想到可以缓存记录集本身而无需缓存含有标签的 HTML 文本。

可以在 Application 或 Session 变量中缓存对象，但有两个主要的问题需要注意：

- 存放在 Application 变量中的对象必须支持自由线程，因此必须是自由线程对象或多线程对象。这意味着无法在 Application 变量中缓存由 VB 创建的组件。
- 在 Session 状态中存放单元线程对象意味着创建该对象的线程是唯一允许访问它的线程。因此 IIS 无法较好地完成线程管理，因为任何试图访问这个对象的页面都必须等待原有线程服务于该页面。这将扼杀扩展应用程序的任何机会。

对于线程问题的讨论参见第 15 章。

默认情况下，ADO 作为单元线程对象装载，这主要是因为部分 OLE DB 提供者并非是线程安全的。在 ADO 安装目录中有一个注册表文件，可将 ADO 转换成多线程模型，由此使 ADO

对象可以安全地存放在 Application 和 Session 对象中。

你也许会认为所有的问题都解决了，可以通过使用各种类型的对象获得显著的速度提升，但这并不一定。许多人已经认识到既然连接到数据库是一个相对昂贵的操作，那么缓存 Connection 对象可在再次连接时节省大量的时间。的确如此，但缓存 Connection 对象意味着该连接永远不会关闭，因此连接缓存池的工作效率比较低。连接缓存池隐含的一个思想实际上是减少服务器上使用的资源，而缓存 ASP 状态中的对象显然不能减少资源的使用。事实上还增加了对它们的占用，因为每缓存一个对象便要占用服务器的资源。对于一个繁忙的站点而言，这将极大地降低 Web 服务器的效率。

所以不应存储 Connection 对象，但对于 Recordset 对象，特别是断开连接的记录集呢？假定 ADO 已从单元线程变成了多线程，就没有什么理由不这么做了，只要确切知道自己在做什么。不要认为这会自地改善 ASP 页的性能。每一个缓存的记录集都在内存和 ASP 管理方面占用服务器的资源，因此不要缓存大的记录集。

另一个技巧是使用记录集的 GetRows 方法，将记录集转换成一个数组。因为数组并不像 Recordset 对象那样受线程问题的影响，因此非常适合用于会话层的变量。然而它同样也占用服务器资源，还必须考虑处理数组的时间。

构建自己的应用程序，缓存技巧并非是必要的。

## 9.4 数据整形

数据整形或分层的记录集能显示一个树状结构或相关记录集。这通过在记录集的字段中包含一个记录集来实现，可以展现数据库的关系，而且多个记录集能在一次调用中返回。有两个理由可以解释它为什么是有用的：

- 性能：当正确使用时，数据整形可以改善性能。
- 便利：在数据整形中非常容易映射父子关系。

要知道数据整形涉及到哪些内容，最简单的方法是看图 9-8 所示的内容。

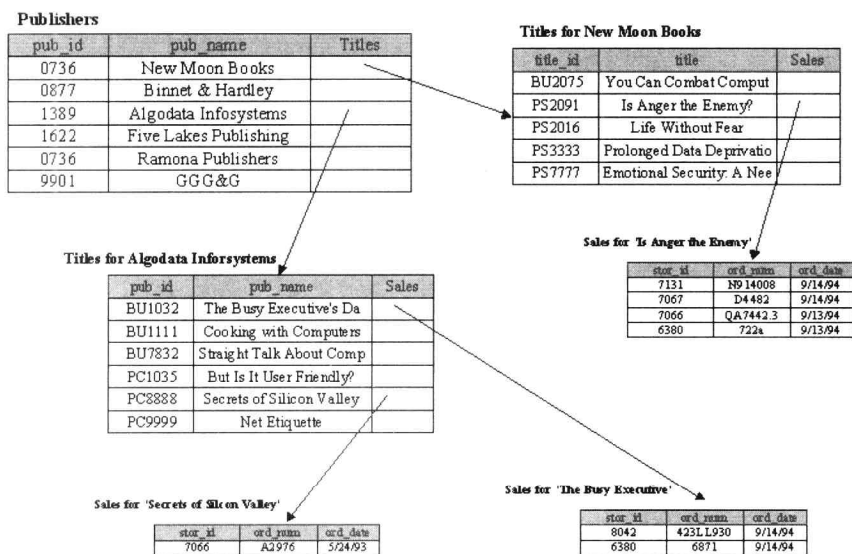


图 9-8 数据整形涉及的内容



图9-8显示了pubs数据库中表Publishers、Titles及Sales的层次关系。

值得注意的一点是每个子记录集都不是独立的记录集。因此，图 9-8中只有三个记录集，而不是六个。这是怎么来的呢？在层次关系中每一层都有一个记录集，分别是 Publishers、Title和Sales。在Publishers表中引用标题时，实际上是引用了 Titles记录集，但ADO过滤了Titles，所以只显示那些与被选择的出版社对应的记录。这就容易使人误以为每个子元素有一个独立的记录集。

#### 9.4.1 使用数据整形

应用数据整形必须：

- 使用MSDataShape OLEDB提供者。
- 使用一种特殊的整形语言，它是 SQL的一种扩充，允许构造层次。

尽管使用新的提供者，连接字符串的实际改变不会太大。这是因为仍然需要从某处获取数据。因此，可以这么做：

```
Provider=MSDataShape; Data Provider=SQLOLEDB; Data Source = ...
```

这里用MSDatashape作为提供者，而正常的 Provider变为Data Provider，而连接字符串的剩余部分保持不变。

为数据整形创建连接字符串的简便方法是从创建正规的连接字符串开始，然后附加到数据整形块的最后。例如，考虑以下正规的连接字符串。

```
strConn = "Provider=SQLOLEDB; Data Source=Kanga; " & _  
" Initial Catalog=pubs; User Id=sa; Password="
```

可以像下面这样为数据整形提供者创建连接字符串。

```
strConn = "Provider=MSDataShape; Data=" & strConn
```

这将提供者设置为MSDataShape，而Data Provider成为实际的数据源。初始的连接字符串已经包含"Provider = "，所以为了获得正确的连接细节，只须在前面加上 Data。

##### 1. 整型语言

整型语言有其自己的语法，但这里我们不打算涉及其构造，它已包含在 ADO文献中。大多数情况下会采用以下命令。

```
SHAPE {parent command} [AS parent alias]  
APPEND ({child command} [AS child alias]  
RELATE parent_column TO child_column) [AS parent_column_name]
```

要理解这一点，最简单的方法是看一个实例，以Publishers和Titles为例。

```
SHAPE {SELECT * FROM Publishers}  
APPEND ({SELECT * FROM Titles}  
RELATE Pub_ID TO Pub_ID) AS rsTitles
```

第一行是父记录集，第二行则是子记录集。第三行指明了关联父、子记录集的两个字段。这个例子中两个表都有一个名为 Pub\_ID的字段（出版社ID字段）。这个命令返回一个包含出版社的记录集，在记录集的最后又附加了一个含



图9-9 执行SHAPE命令后的结果



有子记录集的新列(类型为adChapter)。该列名由AS子句给出,在本例中是rsTitles。

AdChapter类型只是说明了该字段含有一个子记录集。我个人认为, adChild或adRecordset更合适。

通过遍历Fields集合,可以很容易看到父记录集的字段的情况。对于上面的SHAPE命令,得到图9-9所示的结果。

### 访问子记录集

现在,我们有了一个子记录集,它是另一个记录集的一个字段,那么如何访问这个子记录集呢?很简单,可以使用字段的Value属性来建立另一个记录集。

```
Set rsTitles = rsPublishers("rsTitles").Value
```

可以遍历父记录集,对应于每个父记录可以得到一个子记录集。下面的代码能实现这一点。通常以包含文件、变量声明开始。

```
<!-- #INCLUDE FILE="../../Include/Connection.asp" -->
<%
    Dim rsPublishers
    Dim rsTitles
    Dim strShapeConn
    Dim strShape

    Set rsPublishers = Server.CreateObject("ADODB.Recordset")
```

现在创建连接字符串。

```
' Create the provider command
strShapeConn = "Provider=MSDataShape; Data " & strConn
```

接下来,输入实际的整形命令。这将创建一个包含出版社的父记录集和一个含有书名的子记录集。

```
' Now the SHAPE command
strShape = "SHAPE {SELECT * FROM Publishers}" & _
    " APPEND ({SELECT * FROM Titles}" & _
    " RELATE Pub_ID TO Pub_ID) AS rsTitles"
```

然后正常打开记录集。

```
' Open the shaped recordset
rsPublishers.Open strShape, strShapeConn
```

像正常的记录集一样,能够遍历记录。

```
' Loop through the publishers
Response.Write "<UL>"
While Not rsPublishers.EOF
    Response.Write "<LI>" & rsPublishers("Pub_Name")
```

为了访问子记录集,设置一个变量来指向那个包含子记录集的字段的Value值。本例中该变量为rsTitles。

```
' Now the titles
Response.Write "<UL>"
Set rsTitles = rsPublishers("rsTitles").Value
```

变量rsTitles在这里是个记录集,其作用同普通的记录集相同。因此,可以遍历该记录集的值,该值只包含与父出版者匹配的书名。

```

' Loop through the titles
While Not rsTitles.EOF
    Response.Write "<LI>" & rsTitles("title")
    rsTitles.MoveNext
Wend
Response.Write "</UL>"

' Move to the next publisher
rsPublishers.MoveNext
Wend
Response.Write "</UL>"

rsPublishers.Close
Set rsPublishers = Nothing
Set rsTitles = Nothing
%>

```

于是得到一份令人满意的出版社与书名的列表，如图 9-10 所示。



图9-10 整形后的书名列表

用一些DHTML代码和一些额外的标记，就能轻松地隐藏起书名，并且只有选择出版社时才显示相应的书名。

## 2. 多个子记录集

如果对于每个记录集仅能有一个子记录集，那么数据整形就不够完善。但数据整形是极其灵活的。例如可以使用下面的程序来为出版社引用标题和雇员。

```

SHAPE {SELECT * FROM Publishers}
APPEND ({SELECT * FROM Titles})

```

```
RELATE Pub_ID TO Pub_ID) AS rsTitles,
    ({SELECT * FROM Employee}
RELATE Pub_ID TO Pub_ID) AS rsEmployees
```

只需在APPEND子句后加上其他子记录集，就能得到图 9-11的结果。



图9-11 多个子记录集的结果

访问子记录集的方法并没有改变，仍旧可以用列的 Value属性访问子记录集。只是此时有两个子记录集，因此需要使用两个变量。

```
Set rsTitles = rsPublishers("rsTitles").Value
Set rsEmployees = rsPublishers("rsEmployees").Value
```

### 3. 孙代记录集

在子记录集自身还包含子记录集的情况下，可能会出现孙代记录集。例如：

```
SHAPE {SELECT * FROM Publishers}
APPEND (( SHAPE {SELECT * FROM Titles}
    APPEND ({SELECT * FROM Sales}
        RELATE Title_ID TO Title_ID) AS rsSales)
RELATE Pub_ID TO Pub_ID) AS rsTitles
```

在第一个APPEND子句内是另一个SHAPE命令，而不是一个SELECT语句。与多个子记录集的例子相似，访问孙代记录集的方法是相同的。

```
Set rsTitles = rsPublishers("rsTitles").Value
Set rsSales = rsTitles("rsSales").Value
```

对子和孙记录集的深度没有理论上的限制，但也不太可能建立多于三级或四级的子记录集。

#### 9.4.2 性能

数据整形不会自动改善性能，但正确使用时可以改善性能。重要的是记住其工作方式：

- 对于SHAPE命令中的SELECT语句，将完全取出表中的数据。SQL语句并没有得到任何优化。这样，如果在父表中加入 WHERE子句来限制父记录集的记录数，仍能得到所有的子记录集。例如：

```
SHAPE {SELECT * FROM Publishers WHERE State='CA'}  
APPEND ((SELECT * FROM Titles)  
RELATE Pub_ID TO Pub_ID) AS rsTitles
```

APPEND语句返回所有的标题，并不仅限于加州(CA)的出版社。记住，这不是SQL JOIN语句。在加州的出版社以及所有的标题都被提取了，这样就完成了数据整型。

- 可以使用存储过程，这会提高一点性能。然而，如果使用一个参数化的存储过程产生子记录集，那么每次访问子记录集时，这个存储过程都会执行。这意味着，子记录集不在前端代码中编程产生，而是只包含存储过程产生的那些记录。不足之处是增加了服务器的工作，但这样却能保证数据是最新的，因为每次需要时就从数据库中提取数据。

在下一章当我们着眼于客户端数据时，会看到更多有关经过整形的记录集的介绍。

## 9.5 小结

本章主要涉及以下三个内容：

- Command对象，以及使用存储过程和参数。对于ADO程序员来说，这是一个极为重要的主题，因为可以轻松提高应用程序的性能。通过存储过程将SQL代码置于服务器上，得到了最大程度的优化，同时可以控制存储过程的数据输入和输出。
- 性能。这个话题是必不可少的，设计ASP应用程序时考虑其性能是很重要的。通过改写代码来改善性能通常是很困难的，最好一开始就考虑性能而进行相应的编码。关于性能最重要的是测试，测试，再测试。测试网站性能的一个好的简单方法就是使用Web Application Stress工具。
- 数据整形。它虽然没有前两点重要，但仍很有价值，能为相关数据的使用提供方便。利用数据整形可以为嵌套的数据集生成通用的例程。示例站点上就有这样的一个例程。

在下面两章以及本书的其余部分将看到相关的其他知识。在后续章节将详细探讨Web服务器与浏览器之间的交互。毕竟，仅有数据是不行的，还要让人们看见这些数据。