

## 第10章 ASP与客户端数据

在一本ASP专著中讨论客户端数据，这与服务器端的ASP编程是否矛盾？情况并非如此，因为我们至今尚未碰到只从事服务器端编程的ASP程序员。虽然ASP是一项服务器端技术，但可以想像，编程人员不可能仅仅使用ASP进行编程。从事ASP编程的Web开发人员，仍然需要与客户端数据进行交互。

因此，围绕着ASP构建一个应用程序时必须考虑整个应用程序的情况，这也意味着必须考虑客户端。为了获得一个运行良好、快速响应的应用程序，需要很好地使用客户端数据。

本章将讨论如何在客户端使用数据。特别将着重研究：

- 远程数据服务(Remote Data Services, RDS)，如何向客户端传送数据以及从客户端接收数据。
- 如何将ADO记录集绑定到HTML控件。
- 如何利用用户自定义组件提供数据。
- 如何更新客户端数据，并将其反馈到服务器。
- 如何从数据库中获取图像并将其显示在Web页面中。
- 如何创建基于表格的Web页面。

以上覆盖的范围相当广泛，同时有很多不同的方法可以取得相同的结果，但实际上实现起来并不是特别困难。

### 10.1 断开连接的记录集

首先需要掌握的是“断开连接的数据”的概念。迄今为止，在研究ADO的过程中，已经学习了获取记录集的方法，以及如何修改这些记录集中的数据。回顾一下，我们打开一个记录集，对数据做一些修改，然后再关闭这个记录集，在操作记录集的过程中，始终与服务器保持着连接。这是相当明显的，但别忘了Web在本质上是无状态的。如果想使用客户端数据，如何始终保持与服务器的连接？很简单，这是不可能实现的，这也是定义断开连接的记录集概念的缘由。

一个断开连接的记录集只是一个普通的记录集，但解除了与服务器的连接，成为孤立的对象，可以像普通的记录集那样对其执行更新、增加和删除操作。但这些变化只发生在记录集内部，并不反馈到服务器，因为记录集与服务器已不再保持着连接。这并不是缺点，因为可以与服务器重新建立连接，同时服务器可以对任何修改进行更新。即使服务器端的数据已经改变了，ADO仍然有方法让用户及时发现这些变化，这样用户就能决定哪些数据是正确的。这称为冲突处理(conflict resolution)。

断开连接的记录集使我们能在组件之间，包括服务器与客户之间，传送具有全部功能的记录集。本章后面将探讨如何在组件内创建断开连接的记录集。但这里不准备对此做过于详细的研究，因为在本书第13章至第18章已经覆盖了这部分内容，这里仅做简单的介绍，以便于了解组件是如何与远程数据服务交互的。

## 10.2 远程数据服务

远程数据服务(Remote Data Services, RDS)是允许我们处理客户端数据的一系列服务的统称。现在不用担心这方面的问题,因为 RDS本身就是ADO的一部分,只有在需要传送和使用客户端数据时,才会使用。实际上 RDS是由几个组件构成的。图 10-1说明了这些组件以及它们之间是如何协同工作的。

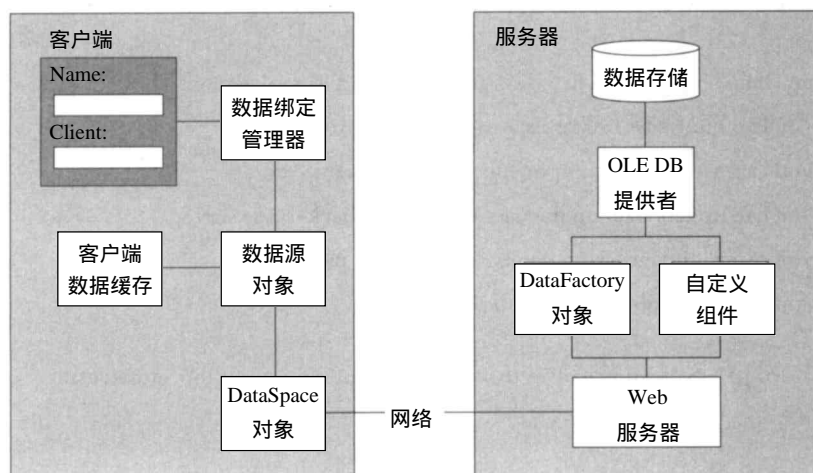


图10-1 RDS的组件构成

组件似乎很多,但并不是所有的组件在每种情形下都被使用,实际上有一些不是 RDS的一部分。然而这里还是把所有可能出现的组件都放在了图上,以备需要时查看。图 10-1分成了两部分,因为使用客户端数据需要一些向客户端传送数据的方法,同时数据一旦到达客户端,也需要一些管理数据的方法。我们先从服务器端开始。

### 10.2.1 RDS服务器组件

虽然RDS用于传送和访问客户端数据,但其确实有一些基于服务器的组件。这是必需的,因为肯定需要某种方式将数据传送到客户端。因此有了一系列能访问数据并允许发送数据到客户端的服务器组件。我们把实际的数据传送称为调度(marshal)。

服务器端组件图的最上端是数据存储,由 OLE DB提供者访问。它并不是 RDS的一部分,但这表示只要有相应的 OLE DB提供者,就可以通过 RDS在客户端使用任何数据。至于如何处理服务器上的数据,可以有两种选择:

- 数据工厂(DataFactory)是缺省的用于访问数据存储的服务器端组件。它作为服务器端 RDS组件的一部分安装在计算机上,除了能从数据存储中获取数据外,还为服务器处理发送到客户端以及从客户端发送来的数据。
- 自定义组件只是一个普通的提供了数据传送方法的 COM组件。当数据工厂不能提供所需的功能时,可以使用自定义组件。本章将介绍一个简单的组件例子,在本书的后面还有一个更复杂的例子。

Web服务器使用这两种组件作为客户和服务端数据的接口。

### 10.2.2 RDS客户组件

在客户端先从底端的 DataSpace 对象开始, 该对象作为客户端的一部分与数据工厂或自定义对象协同工作。DataSpace 对象是一个代理对象, 负责与服务器进行通信, 同时也是数据传输的通道 (或通常所说的调度)。DataSpace 对象是用客户端脚本语言或用 HTML 语言中的 < OBJECT > 标记创建的 COM 对象。在本章后面会看到关于这方面的例子。

DataSpace 对象上面是数据源对象 (Data Source Object, DSO), 负责存储客户端数据。一个数据源对象包含一个 ADO 数据记录集, 与客户数据缓存共同管理数据。客户数据缓存只是一种管理客户端数据的客户光标服务。同时数据源对象又是一个 COM 对象, 与 DataSpace 对象类似, 也可以通过客户端脚本或使用 HTML 语言中的 < OBJECT > 标记来创建。同样, 在本章稍后也会介绍关于这方面的一些例子。

数据源对象的上面是数据绑定管理器, 任务是建立 HTML 控件与数据源对象的连接。这就是我们所知道的绑定, 可以通过设置某些 HTML 控件的 DATASRC 和 DATAFLD 属性来实现。下面将对这些内容进行讨论, 并示范如何在浏览器中方便地使用数据。

### 10.2.3 支持RDS的浏览器

要知道 RDS 是微软的技术, 因此只能在微软的浏览器上工作。实际上, 只有在 IE 4.0 或更高版本的浏览器中才完全支持 RDS。

当编写依赖于 RDS 的应用程序时, 需要注意访问应用程序的客户的 RDS 版本可能会与服务器端有所不同。举例来说, IE 4 中的是 RDS 1.5 版本, 而 IE 5、Office 2000 和 Visual Studio 6 中的则是 RDS 2.0 版本。有两种方法可以处理这种兼容性问题:

- 确保所有用户已经升级到 RDS 的最新版本。如果客户运行的是 Windows 2000, 那么已经在运行最新版本的 RDS 了。否则, 可以从网址 [www.microsoft.com/data](http://www.microsoft.com/data) 处下载。RDS 2.5 版本是目前最新的随同 Windows 2000 一起发布的版本, 同时也是一个可单独下载的软件包。
- 当连接到数据源时, 指定数据工厂的模式。这可以指定使用的是哪一个版本的 RDS 组件, 后面将介绍这方面的一个例子。

### 10.2.4 数据源对象

数据源对象是一个存储和管理客户端数据的客户端对象。因为这是使用 RDS 最简单的一种方式, 首先研究一下这些对象。

这里有几个不同的数据源对象, 每一个都针对不同类型的数据:

- 表格数据控件 (Tabular Data Control, TDC), 用于处理表格形式或分隔形式的文本文件。
- RDS 数据控件, 用于连接 OLE DB 数据存储, 能够指定连接到哪个数据存储, 以及返回哪些数据。
- Java 数据库连接器, 这是一个通过 Java 数据库控件 (Java DataBase Control, JDBC) 连接到数据存储的 Java 小程序。这里我们不想讨论 JDBC, 因为它并不提供其他控件无法实现的功能。
- 微软的 HTML (MSHTML) 数据源对象用 HTML 标记数据, 并把它作为数据源。

• XML 数据源对象使用 XML 数据，用于结构化的或任意结构的 XML。

选用哪一种数据源对象取决于你想做什么，以及数据从哪里来。如果需要向客户提供少量的数据，并且不允许用户修改数据，那么表格数据控件（TDC）可能会比较适合。这种数据源是一个文本文件，不需要任何数据库，因此编辑起来比较简单。对于从数据库中取出数据并且可能需要更新的情况，RDS 数据控件是最合适的。而对于许多新数据源，会发现此时需要使用 XML 数据控件。这实际依赖于所使用的 Web 应用程序的类型，以及用户所需的功能。

我们将依次介绍这些数据控件，一旦了解了如何用它们把数据传送到客户端，将会介绍如何使用这些数据。

1. 表格数据控件

表格数据控件(Tabular Data Control，TDC)是最简单的数据源对象，主要用于少量的只读数据，特别是那些从不改变或很少修改的，不需要从客户端进行更新的静态数据。例如，表格数据控件能提供一个网页内的菜单项或链接的列表。

通过在 HTML 代码中使用 < OBJECT > 标记可以创建一个表格数据控件。参数 DataURL 可以指定包含文本数据的文件名。

```
<OBJECT CLASSID="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"
    ID="dsoAuthors" WIDTH="0" HEIGHT="0">
    <PARAM NAME="DataURL" VALUE="Authors.csv">
</OBJECT>
```

TDC 只读取表格中的数据或标记为表格形式的数据，例如，可以处理逗号分隔形式的数  
据(Comma Separated Value，CSV)，类似于下面的数据：

```
"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"
"409-56-7008","Bennet","Abraham","415 658-9932"
```

TDC 也可以由自定义。除了 DataURL 外，TDC 还有 16 个参数，可以通过设置 OBJECT 标记  
的参数项或编写脚本代码来配置这些参数。参数的说明如表 10-1 所示。

表10-1 TDC的参数及说明

属 性	数据类型	说 明	缺 省 值
AppendData	布尔型	确定新数据是替换还是追加到数据源对象中的现 有数据	False
CaseSensitive	布尔型	指出字符串比较时是否大小写敏感	True
CharSet	字符型	指出数据的字符集类型。附录中有字符集的列表	Windows-1252
DataURL	字符型	指出数据源文件的 URL	
EscapeChar	字符型	指定源文件中使用的转义字符。这些字符位于其他 字符的前面以避免与FieldDelim、RowDelim或TextQuali- fier混淆	
FieldDelim	字符型	指定字段之间的分隔字符	，(逗号)
Filter	字符型	设置过滤条件	
Filtercolumn	字符型	设置过滤的列	

(续)

属 性	数据类型	说 明	缺 省 值
FilterValue	字符型	设置过滤的列的值	
Language	字符型	指定数据文件使用的语言	en-us(美国英语)
ReadyState	长整型	指出控件的当前状态。可以是以下值： adcReadyStateComplete(4) 表明所有的数据都传 送完毕，或发生了一个错误 adcReadyStateInteractive(3) 表明数据正在传送中 adcReadyStateLoaded(2) 表明控件已被加载并等 待数据传输 这个属性是只读的	
RowDelim	字符型	指定文本文件中的行分隔符，缺省为回车符	一个新行字符
Sort	字符型	指出要排序的列的列表。列名前有加号表明是按 降序排序，否则按升序排序	
SortDirection	布尔型	如果排序是升序则为 True，降序则为 False	
SortColumn	字符型	指定排序的列	
TextQualifier	字符型	指定封闭文本字段的字符，缺省为双引号	" (双引号)
UseHeader	布尔型	表明文本文件中首行是否包含列名	False

下面是使用参数创建 TDC 的一个例子。

```
<OBJECT CLASSID="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"
  ID="dsoAuthors" WIDTH="0" HEIGHT="0">
  <PARAM NAME="DataURL" VALUE="Authors.csv">
</OBJECT>
```

也可以在客户端脚本中获取数据，下面的例子显示了给 TDC加载数据的 JScript 脚本。

```
function fillTDC()
{
    dsoAuthors.dataURL = 'authors.csv';
    dsoAuthors.Reset();
}
```

如果改变了 TDC 的 DataURL 参数，必须使用 Reset 方法，这样才能使新的 URL 起作用。当介绍数据绑定时，会更详细地讨论如何使用它。Reset 方法是 TDC 唯一的一个方法。

2. RDS 数据控件

RDS 数据控件能够访问一般的数据存储，而不是平面文件。它通常用于连接 SQL 数据库以从表、查询或存储过程获取数据。与 TDC 不同，RDS 数据控件允许更新数据。在本章稍后通过示例说明如何进行数据更新。

类似于 TDC，可以用 HTML 脚本中的 OBJECT 标记来创建一个 RDS 数据控件，并以类似的方式设置其属性。

```
<OBJECT CLASSID="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"
  ID="dsoAuthors" WIDTH="0" HEIGHT="0">
  <PARAM NAME="Connect" VALUE="Connection String">
  <PARAM NAME="Server" VALUE="Server Name ">
  <PARAM NAME="SQL" VALUE="Query Text">
</OBJECT>
```

同样，注意定制数据控件时参数的使用方法。上面的例子显示了最常见的设置方法。和 TDC 一样，还有很多其他可以设置的参数，如表 10-2 所示。

表10-2 RDS数据控件的参数及说明

属 性	数 据 类 型	说 明	缺 省 值
Connect	字符型	一个标识数据存储的ADO连接字符串	
ExecuteOptions	长整型	指出命令是异步执行还是同步执行。可以是以下值之一： adcExecSync(1) 同步执行(缺省值) adcExecAsync(2)异步执行	adcExecAsync
FetchOptions	长整型	获取数据的方式，可以是以下值： adcFetchUpFront(1) 先取数据，然后将控制交给应用程序 adcFetchBackground(2)先立即取得第一批数据，剩余的数据在后台获取 adcFetchAsync(3)在后台获取所有的数据	adcFetchAsync
FilterColumn	字符型	指定被过滤的列	
FilterCriterion	字符型	指定过滤的条件。可以是以下运算符： <(小于) <=(小于等于) =(等于) >=(大于等于) >(大于) <>(不等于)	
FilterValue	字符型	过滤的值	
Handler	字符型	自定义的数据处理器的名称和参数	MSDFMAP.Handler
InternetTimeout	长整型	在错误发生前等待的时间(毫秒为单位)	300000
ReadyState	长整型	控件的当前状态，可以是以下值： adcReadyStateComplete (4) 表明所有的数据都传送完毕，或发生了一个错误 adcReadyStateInteractive(3) 表明数据仍然在传送中 adcReadyStateLoaded(2) 表明控件已被加载并等待数据传输	
Recordset	记录集型	数据控件访问的ADO数据记录集。该参数只读	
Server	字符型	数据所在的服务器的名称。为了安全，必须与提供 Web页面的服务器同名。可以是一个标准URL，也可以是机器名称(如果使用DCOM)	
SortColumn	字符型	排序的列名	
SortDirection	布尔型	表明排序是否为升序	
SourceRecordset	字符型	将控件的基础记录集设置为一个现有的记录集。该属性只写	
SQL	字符型	获取数据的SQL字符串	
URL	字符型	数据源的URL	

异步执行是指在后台检索数据，可以在全部数据返回之前在 Web页面上使用已经



得到的数据。虽然可能需要的是全部的数据，但异步工作至少可提前开始处理数据。也可让用户先看到某些内容，这使得 Web 站点看上去响应能力更强。

与TDC类似，RDS数据控件可以通过设置 OBJECT标记的参数或编写代码来设置其属性。下面举一个例子：

```
<OBJECT CLASSID="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"
  ID="dsoAuthors" WIDTH="0" HEIGHT="0">
  <PARAM NAME="Connect" VALUE="DSN=pubs">
  <PARAM NAME="Server" VALUE="W2000">
  <PARAM NAME="SQL" VALUE="SELECT * FROM Authors">
</OBJECT>
```

等效于：

```
<OBJECT CLASSID="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"
  ID="dsoAuthors" WIDTH="0" HEIGHT="0">
</OBJECT>

<SCRIPT LANGUAGE=JScript>

function window.onload()
{
  dsoAuthors.Connect = "DSN=pubs";
  dsoAuthors.Server = "W2000";
  dsoAuthors.SQL = "SELECT * FROM Authors";
  dsoAuthors.Refresh();
}
</SCRIPT>
```

这里为Connect参数使用了一个DSN，因为这非常适合该页面，但也可以是任何有效的ADO连接字符串。

URL是ADO 2.5版提供的新特性，允许使用一个文件作为数据源。该文件可以有两种格式：一种是用Recordset.Save方法保存的记录集；另一种是一个ASP页面，它创建一个记录集，然后将其保存在一个流中。代码如下：

```
<OBJECT CLASSID="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"
  ID="dsoAuthors" WIDTH="0" HEIGHT="0">
  <PARAM NAME="URL" VALUE="DataPage.asp">
</OBJECT>
```

文件DataPage.asp包含以下VBScript代码：

```
<%
  Dim rsData
  Set rsData = Server.CreateObject("ADODB.Recordset")
  rsData.Open "SELECT * FROM Authors", strConn
  rsData.Save Response, adPersistXML
  rsData.Close
  Set rsData = Nothing
%>
```

这只是创建了一个记录集，然后用Save方法将记录集以XML格式保存到Response对象中。在ADO的早期版本中，只能将记录集存为物理文件，而ADO 2.5版本能够直接将其存为流。这个ASP页面的结果就是XML格式的记录集。下一章将研究关于流和XML数据的所有主题。

使用URL属性优于使用Connect和SQL属性，其最大优点是：在用户可以看到的网页中不会出现连接的细节。考虑下面的对象定义：

```
<OBJECT CLASSID="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"
  ID="dsoAuthors" WIDTH="0" HEIGHT="0">
  <PARAM NAME="Connect" VALUE="DSN=pubs">
  <PARAM NAME="Server" VALUE="W2000">
  <PARAM NAME="SQL" VALUE="SELECT * FROM Authors">
</OBJECT>
```

第一行显示了连接的细节。此时能够看到 DSN为pubs，并且我们选择了 authors表的全部列。这无疑为电脑黑客进入 Web 站点提供了潜在的路径，因为他们知道了服务器的名称以及数据库的一些细节。现在，考虑一下使用 URL属性的情况：

```
<OBJECT CLASSID="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"
  ID="dsoAuthors" WIDTH="0" HEIGHT="0">
  <PARAM NAME="URL" VALUE="DataPage.asp">
</OBJECT>
```

现在，用户所见到的的是一个 ASP网页的 URL地址，没有任何有关服务器和数据库的详细信息。

使用CONNECT/SQL属性的方法，用户可以清楚地见到连接的细节，而使用URL所见的却是数据。从这一点上来说，消除了一个安全问题。

在脚本中设置RDS数据控件的属性时，必须使用 Refresh方法，如下所示：

```
<SCRIPT LANGUAGE=JScript>

function window.onload()
{
    dsoAuthors.URL='DataPage.asp'
    dsoAuthors.Refresh();
}
</SCRIPT>
```

这将强迫数据控件使用新的属性值，并重新从数据提供者那里检索数据。除了 Refresh方法外，RDS数据控件还有许多其他方法，如表 10-3所示。

表10-3 RDS数据控制的方法及说明

方 法	说 明
Cancel	取消任何异步操作
CancelUpdate	取消对数据的任何修改
CreateRecordset	创建一个空的记录集，这允许在本地创建新的数据集
MoveFirst	移到第一条记录
MoveLast	移到最后一条记录
MoveNext	移到下一条记录
MovePrevious	移到上一条记录
Refresh	更新来自数据存储的数据
Reset	应用过滤或排序条件
SubmitChanges	将所有未解决的修改送回数据存储

在本章后面，会见到大多数方法的使用情况。

3. MSHTML 数据控件

微软HTML(MSHTML)数据控件比较特别的地方在于：MSHTML是IE的一个组成部分，



并能提供一个基于HTML文档的数据源。虽然本质上MSHTML并不是为数据存储使用的一种格式，但如果确实有许多包含某些数据格式的HTML网页，MSHTML可能会变得较为有用。

使用这个控件的代码如下所示：

```
<OBJECT ID="dsoAuthors" DATA="Authors.html" HEIGHT="0" WIDTH="0">
</OBJECT>
```

为了能使用这个控件，必须为HTML标记指定ID属性，因为正是ID属性确定了字段的名称。例如：

```
<DIV ID="au_id">172-32-1176</DIV>
<SPAN ID="au_lname">White</SPAN>
<H1 ID="au_fname">Bob</H1>
<PRE ID="au_id">213-46-8915</PRE>
<H2 ID="au_lname">Green</H2>
<H1 ID="au_fname">Cheryl</H1>
```

读者会注意到这看起来不太漂亮。是的，但正是这样才可以说明HTML标记的名称是无关紧要的，ID才是重要的。使用MSHTML DSO解析时，上面的HTML脚本将会产生两行数据，每一行有三个字段。最后会得到类似于表10-4的数据。

字段是由ID属性确定的。如果一个标记的ID与现有的ID相同，那么这个标记的数据将成为新的一行，否则在相同的行中创建一个新的字段。

表10-4 HTML脚本产生的数据

au_id	au_lname	Au_fname
172-32-1176	White	Bob
213-46-8915	Green	Cheryl

与已经讨论过的数据控件类似，MSHTML数据控件有一个Recordset属性，这也是该控件唯一的一个属性。MSHTML数据控件没有方法。

#### 4. XML数据控件

我们已经知道了一种将XML数据放入RDS控件的方法，即使用RDS数据控件和URL属性从ASP文件中获取XML数据。另外一种方法是使用XML Data Island(XML数据岛)，这需要使用时XML标记。在这里简要地提一下这个问题，因为在下一章会讨论处理XML数据的细节。

<XML>标记是一个与数据控件功能相似的浏览器HTML标记。在许多情况下与使用一个RDS数据控件类似，但该标记是为处理XML数据而特别设计的。使用的方式有二种。

第一种是使用SRC属性来指定数据的位置。

```
<XML ID="dsoAuthors" SRC="Authors.xml"></XML>
```

这表示使用文件Authors.xml作为数据源。

另外，也可在标记中嵌入XML。

```
<XML ID="dsoAuthors">
<Authors>
  <Author>
    <au_id>172-32-1176</au_id>
    <au_lname>White</au_lname>
    <au_fname>Johnson</au_fname>
    <phone>408 496-7223</phone>
    <contract>True</contract>
  </Author>
  <Author>
    <au_id>213-46-8915</au_id>
    <au_lname>Green</au_lname>
```

```
<au_fname>Marjorie</au_fname>
<phone>415 986-7020</phone>
<contract>True</contract>
</Author>
</Authors>
</XML>
```

我们将在下一章详细讨论 XML 数据控件。

### 10.2.5 数据绑定

迄今为止，已经可以用几个不同的 RDS 数据控件将数据送到客户端，但还没有介绍当数据到达客户端后，如何处理数据。实际上，这些数据控件负责的是数据的存储及管理，并不真正地显示数据。因此，问题在于如何将数据从数据控件中取出，并将其提供给 HTML 元素，使用户能够见到数据。

使用客户端数据最简单的方法是将数据与 HTML 标记绑定。绑定就是在 HTML 元素和数据控件之间建立一种联系。数据控件主要负责管理数据，并为 HTML 元素提供数据，而 HTML 元素则将数据显示在屏幕上。

为了将数据源与 HTML 元素绑定，需要设置两个属性：

- DATASRC，确定包含数据的数据控件。在数据源名称前总是要加上一个“#”。
- DATAFLD，确定绑定数据控件中的哪个字段。这些字段是数据控件管理的数据中的列名。因此对于一个数据库，就是表中的列名。

例如：

```
<OBJECT CLASSID="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"
  ID="dsoAuthors" WIDTH="0" HEIGHT="0">
  <PARAM NAME="URL" VALUE="DataPage.asp">
</OBJECT>

<DIV DATASRC="#dsoAuthors" DATAFLD="au_fname"></DIV>
<DIV DATASRC="#dsoAuthors" DATAFLD="au_lname"></DIV>
```

这里，dsoAuthors 是一个标准的 RDS 数据控件。我们创建了两个 DIV 元素，并通过设置属性 DATASRC 指向先前定义的数据控件。然后设置属性 DATAFLD 为字段名，在本例中为 au\_fname 和 au\_lname。这就是数据绑定的全部过程。其结果如图 10-2 所示。

需要记住一个重要事情是文本输出 (Johnson 和 White) 根本没有在 HTML 脚本中出现。实际上 HTML 只由以上显示的内容组成。

因此，数据绑定就是数据控件管理远程数据源的数据，同时 HTML 元素使用该数据并将其显示在屏幕上的能力。

除此之外，HTML 元素还有第三个属性：

DATAFORMATAS，可以是 HTML 或 TEXT，表示如何格式化字段中的数据。缺省为 TEXT，但如果需要 HTML 格式的数据，可以通过数据绑定将其格式化，例如下面的文本文件。

```
Description, Image
The main Wrox logo, <IMG SRC="logos/WroxLogo.gif">
The Wrox Conferences logo, <IMG SRC="logos/WroxConferencesLogo.gif">
```

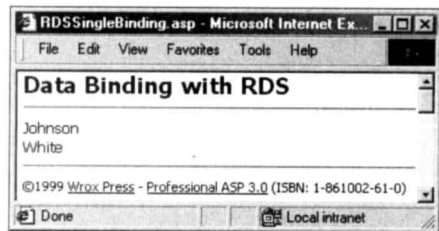


图10-2 数据绑定结果

The ASPToday logo, <IMG SRC="logos/ASPTodayLogo.gif">

这是TDC的源文件，并且包含两个字段，第一个是说明，第二个是显示某些图像的HTML文本。现在把它和一个HTML表格进行绑定。

```
<TABLE ID="tblData" DATASRC="#dsoLogos">
  <THEAD>
    <TR>
      <TD>Description</TD>
      <TD>Image</TD>
    </TR>
  </THEAD>
  <TBODY>
    <TR>
      <TD><SPAN DATAFLD="Description"></SPAN></TD>
      <TD><SPAN DATAFLD="Image"></SPAN></TD>
    </TR>
  </TBODY>
</TABLE>
```

这里不要为数据绑定担心，本例介绍的是表格数据绑定，下面将详细讨论这方面的内容。在这里使用是因为能较好地说明格式化数据。

在浏览器中打开它，将会看到图 10-3 所示的结果。

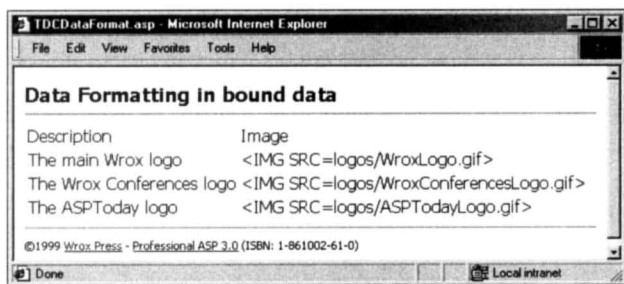


图10-3 绑定数据的格式化

可以看到数据文件中的三行数据全部显示在浏览器中，但是 HTML 作为文本显示。为了见到HTML格式的数据，需要使用属性 DATAFORMATAS。

```
<TD><SPAN DATAFLD="Description"></SPAN></TD>
<TD><SPAN DATAFLD="Image" DATAFORMATAS="HTML"></SPAN></TD>
```

此时在浏览器中打开它，会得到一个效果较好的HTML页面，如图 10-4 所示。

将第二个字段格式化为 HTML 文档，这一事实说明了字段中任何 HTML 标记都可以解释成为 HTML 文档。因此，IMG 标记变为真正的图像，而源文件中并不包含任何 IMG 标记，只有绑定的数据。

可以为任一数据源中的字段使用这种格式，任何 HTML 标记都会被解释。这对于那些允许用户输入格式化文本的情形是比较有利的。

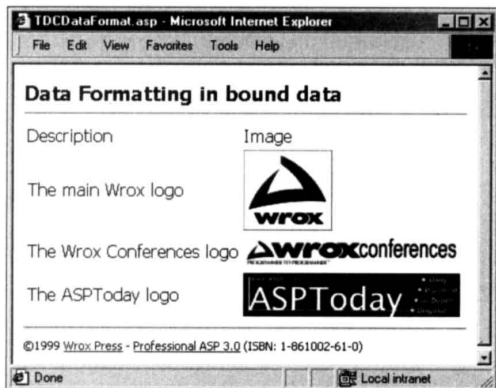


图10-4 绑定数据HTML格式化

1. 支持数据绑定的元素

在上面的例子，使用了 < SPAN >、< DIV > 和 < TABLE > 标记来绑定数据，但并不是所有的HTML元素都支持数据绑定。表 10-5详细列举了这些元素，表中列是：

- HTML元素定义支持数据绑定的元素。
- 绑定属性定义绑定到数据的 HTML元素属性。例如上面的 DIV 标记，绑定的数据就放在 innerText属性中。如果使用 A 标记，并将其绑定到一个字段，那么数据将会放在 href 属性中。
- 可否更新数据指出元素中的绑定数据是否能被更新。
- 可否表格绑定指出该元素是否允许绑定子元素。稍后会讨论这个内容。
- 可否作为HTML显示指出数据是否能格式化为 HTML文档。

表10-5 支持数据绑定的HTML元素

HTML 元素	绑定属性	可否更新数据	可否表格绑定	可否作为HTML显示
A	href	不可	不可	不可
APPLET	PARAM	可以	不可	不可
BUTTON	innerText和innerHTML	不可	不可	可以
DIV	innerText和innerHTML	不可	不可	可以
FRAME	src	不可	不可	不可
IFRAME	src	不可	不可	不可
IMG	src	不可	不可	不可
INPUT	checked	可以	不可	不可
TYPE=CHCKBOX INPUT	value	可以	不可	不可
TYPE=HIDDEN INPUT	value	可以	不可	不可
TYPE=LABEL INPUT	value	可以	不可	不可
TYPE=PASSWORD INPUT	checked	可以	不可	不可
TYPE=RADIO INPUT	value	可以	不可	不可
TYPE=TEXT LABEL	innerText和innerHTML	不可	不可	可以
LEGEND	innerText和innerHTML	不可	不可	不可
MARQUEE	innerText和innerHTML	不可	不可	可以
OBJECT	param	可以	不可	不可
SELECT	选择的<OPTION>元素的文本	可以	不可	不可
SPAN	innerText和innerHTML	不可	不可	可以
TABLE	无	不可	可以	不可
TEXTAREA	value	可以	不可	不可

2. 单个记录绑定

单个记录绑定用于只显示单行数据的情况。例如，考虑下面的代码：

```
ID:      <SPAN DATASRC="#dsoData" DATAFLD="au_id"></SPAN><BR>
First Name: <SPAN DATASRC="#dsoData" DATAFLD="au_fname"></SPAN><BR>
Last Name: <SPAN DATASRC="#dsoData" DATAFLD="au_lname"></SPAN><BR>
Phone:    <SPAN DATASRC="#dsoData" DATAFLD="phone"></SPAN><BR>
Address:  <SPAN DATASRC="#dsoData" DATAFLD="address"></SPAN><BR>
City:     <SPAN DATASRC="#dsoData" DATAFLD="city"></SPAN><BR>
State:    <SPAN DATASRC="#dsoData" DATAFLD="state"></SPAN><BR>
Zip:      <SPAN DATASRC="#dsoData" DATAFLD="zip"></SPAN><BR>
Contact:  <SPAN DATASRC="#dsoData" DATAFLD="contract"></SPAN><BR>
```

使用单个记录绑定时，每一个将被绑定的HTML元素都要确定数据源 (DATASRC)和绑定的字段(DATAFLD)。

以上数据绑定的结果如图 10-5所示。

作为一个结果来说，这已经满足要求了，但由于在HTML文档中忽略了空格，所以数据排列得不整齐。数据绑定使我们易于得到数据，但看上去不太美观。一个好方法是使用表格来对齐数据。

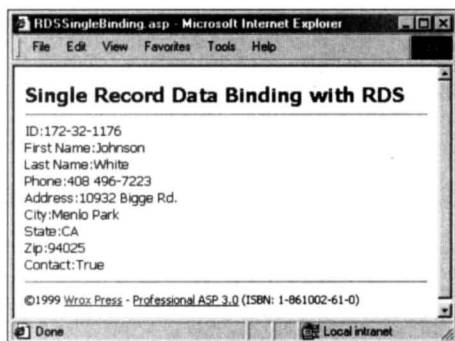


图10-5 单个记录绑定的结果

```
<TABLE ID="tblData">
  <TR><TD>ID:</TD>
    <TD><SPAN DATASRC="#dsoData" DATAFLD="au_id"></SPAN></TD></TR>
  <TR><TD>First Name:</TD>
    <TD><SPAN DATASRC="#dsoData" DATAFLD="au_fname"></SPAN></TD></TR>
  <TR><TD>Last Name:</TD>
    <TD><SPAN DATASRC="#dsoData" DATAFLD="au_lname"></SPAN></TD></TR>
  <TR><TD>Phone:</TD>
    <TD><SPAN DATASRC="#dsoData" DATAFLD="phone"></SPAN></TD></TR>
  <TR><TD>Address:</TD>
    <TD><SPAN DATASRC="#dsoData" DATAFLD="address"></SPAN></TD></TR>
  <TR><TD>City:</TD>
    <TD><SPAN DATASRC="#dsoData" DATAFLD="city"></SPAN></TD></TR>
  <TR><TD>State:</TD>
    <TD><SPAN DATASRC="#dsoData" DATAFLD="state"></SPAN></TD></TR>
  <TR><TD>Zip:</TD>
    <TD><SPAN DATASRC="#dsoData" DATAFLD="zip"></SPAN></TD></TR>
  <TR><TD>Contact:</TD>
    <TD><SPAN DATASRC="#dsoData" DATAFLD="contract"></SPAN></TD></TR>
</TABLE>
```

这个HTML文档虽然不容易阅读，但却提供了一个较好的显示结果，如图 10-6所示。



图10-6 单个记录绑定的表格显示结果

注意，这个例子只显示了使用 SPAN元素来存放数据。如果想编辑数据，那么可以使用 INPUT元素来实现。例如：

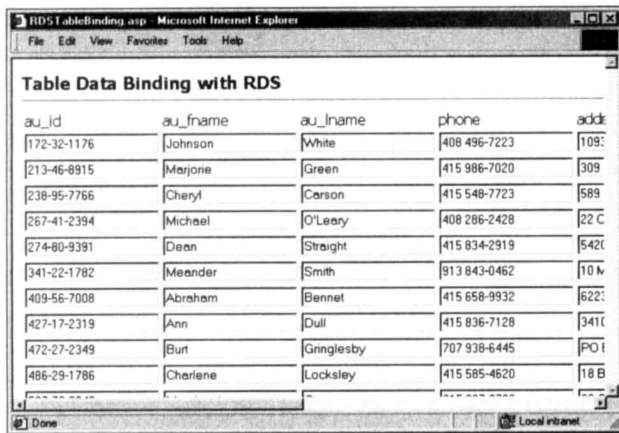
```
<TABLE ID="tblData">
  <TR><TD>ID:</TD>
    <TD>
      <INPUT TYPE="TEXT" DATASRC="#dsoData" DATAFLD="au_id"></INPUT>
    </TD>
  </TR>
</TABLE>
```





### 3. 表格绑定

表格绑定不同于单个记录绑定，因为不只是为对齐数据而使用表格。在表格绑定中把数据绑定到了TABLE元素，能够一次看到多条记录，如图 10-9所示。



au_id	au_fname	au_lname	phone	address
172-32-1176	Johnson	White	408 496-7223	1091
213-46-8915	Marjorie	Green	415 986-7020	309
238-95-7766	Cheryl	Carson	415 548-7723	589
267-41-2394	Michael	O'Leary	408 286-2428	22 C
274-80-9391	Dean	Straight	415 834-2919	542 C
341-22-1782	Meander	Smith	913 843-0462	10 W
409-56-7008	Abraham	Bennet	415 858-9932	622 C
427-17-2319	Ann	Dull	415 836-7128	341 C
472-27-2349	Burt	Gringlesby	707 938-6445	PO I
486-29-1786	Charlene	Locksley	415 585-4620	18 B

图10-9 表格绑定的界面

这甚至比单个记录绑定更容易，实现表格绑定需要使用表格的 DATASRC属性，然后使用 DATAFLD属性绑定表格元素。这样就将表格与数据控件绑定起来，每一个表格单元绑定到单独的字段。

然而，看一下能够被绑定的 HTML元素的列表，会发现表格单元元素 (TD)并不在其中。因为这个原因，一般为只读的表格使用 SPAN或DIV标记，而对于可编辑的表格则使用 INPUT标记。例如，图 10-9中的表格是用下列代码创建的：

```
<TABLE ID="tblData" DATASRC="#dsaData">
  <THEAD>
    <TR>
      <TD>au_id</TD>
      <TD>au_fname</TD>
      <TD>au_lname</TD>
      <TD>phone</TD>
      <TD>address</TD>
      <TD>city</TD>
      <TD>state</TD>
      <TD>zip</TD>
      <TD>contract</TD>
    </TR>
  </THEAD>
  <TBODY>
    <TR>
      <TD><INPUT TYPE="TEXT" DATAFLD="au_id"></INPUT></TD>
      <TD><INPUT TYPE="TEXT" DATAFLD="au_fname"></INPUT></TD>
      <TD><INPUT TYPE="TEXT" DATAFLD="au_lname"></INPUT></TD>
      <TD><INPUT TYPE="TEXT" DATAFLD="phone"></INPUT></TD>
      <TD><INPUT TYPE="TEXT" DATAFLD="address"></INPUT></TD>
      <TD><INPUT TYPE="TEXT" DATAFLD="city"></INPUT></TD>
      <TD><INPUT TYPE="TEXT" DATAFLD="state"></INPUT></TD>
      <TD><INPUT TYPE="TEXT" DATAFLD="zip"></INPUT></TD>
      <TD><INPUT TYPE="TEXT" DATAFLD="contract"></INPUT></TD>
    </TR>
  </TBODY>
</TABLE>
```

TABLE元素还有另外一个用于数据绑定的属性：DATAPAGESIZE，决定了在表格中可以显示的记录数。

```
<TABLE ID="tblData" DATASRC="#dsoData" DATAPAGESIZE="10">
```

在上面的例子中，表格一次只能含有 10 条记录。记录集的移动方法在这里不起作用，因为表格限制了可见的记录，所以必须使用表格的两个方法，如下所示：

```
<button id="cmdPreviousPage" title="Previous Page"
    onclick="tblData.PreviousPage()">Previous Page</button>
```

```
<button id="cmdNextPage" title="Next Page"
    onclick="tblData.NextPage()">Next Page</button>
```

#### 4. 动态绑定

到目前为止，所有的例子都只显示了一个固定的记录集，绑定的字段在设计期间已经创建。但看起来大量的代码不能重用，特别是在 Web 应用程序正给用户带来越来越强的功能的情况下，这种方式缺乏开发程序的灵活性，

解决这个难题的方法是根据数据控件中的数据动态创建表中的字段。实际上这也比较容易，依赖于客户端的脚本程序。那么，假定让用户在表 authors 和 publishers 中进行选择，如图 10-10 所示。

现在我们并不真想绑定两个表的所有字段，因为这会变得难以维护。如果源数据的结构改变了，或者想增加另一个表，情况将会怎样？处理这种情况的方法就是创建一个虚表，在运行期间动态地创建和绑定字段。

首先，创建数据控件。

```
<OBJECT CLASSID="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"
    ID="dsoData" HEIGHT="0" WIDTH="0"
    ONDATASETCOMPLETE="createCells()">
```

这是 RDS 数据控件，与前面例子唯一不同的是这里没有设置参数，代码中也是如此。唯一增加的是设置了一个在数据控件读完数据后运行的函数。

接下来，需要创建两个按钮来确定数据。

```
<BUTTON ID="cmdAuthors"
    onclick="resetData('authors')">authors</BUTTON>
<BUTTON ID="cmdPublishers"
    onclick="resetData('publishers')">publishers</BUTTON>
```

下面创建虚表。

```
<TABLE ID="tblData">
    <THEAD><TR></TR></THEAD>
    <TBODY><TR></TR></TBODY>
</TABLE>
```

这充当了模板的作用。注意，表格中还没有单元格。这是因为并不知道数据有多少个字段，所以也将在运行期间创建它们。

现在编写 JScript 代码。首先看一下 resetData 函数，该函数设置数据控件的属性并加载数据。

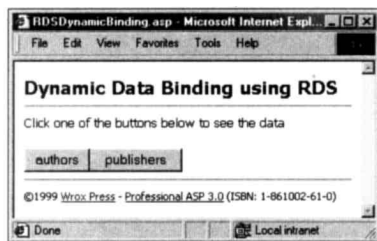


图10-10 使用RDS动态数据绑定的界面

```
function resetData(strTable)
{
    // reset the data
    dsoData.Connect = 'Provider=SQLOLEDB; Data Source=' +
        '<%= Request.ServerVariables("SERVER_NAME") %>' +
        '; Initial Catalog=pubs; User ID=sa; Password=';
    dsoData.Server = 'http://<%= Request.ServerVariables("SERVER_NAME") %>';
    dsoData.SQL = 'SELECT * FROM ' + strTable;
    dsoData.Refresh();
}
```

虽然这看起来比使用参数更复杂一些，但是仍然比较简单。别忘了参数名是如何映射到属性的？这里所做的就是设置那些属性，然后调用 Refresh方法更新数据控件。看上去，这可能比以前的例子更糟糕，因为在代码中只有不多的 ASP，也只是简单地在属性中填入 Web 服务器的名字。但使用该方法可以在不修改代码的情况下将此 ASP 页面从一个服务器移到另一个服务器。作为数据源的表名可以通过选择适当的按钮而传给函数。

一旦加载了数据，将触发数据控件的 ondatasetcomplete 事件，运行 createCells 函数。

```
function createCells()
{
    var fldF;
    var tblCell;

    // Delete what's there already
    deleteCells();

    // Now create the new cells
    for (fldF = new Enumerator(dsoData.recordset.Fields);
        !fldF.atEnd(); fldF.moveNext())
    {
        // Create a new cell for the heading
        tblCell = tblData.rows[0].insertCell();
        tblCell.innerHTML = '<B>' + fldF.item().name + '</B>';

        // Create a new cell for the body
        tblCell = tblData.rows[1].insertCell();
        tblCell.innerHTML = '<INPUT DATAFLD="' +
            fldF.item().name + '"></INPUT>';
    }

    // now bind to the data source
    tblData.dataSrc = '#dsoData';
}
```

这同样也很简单。首先删除了现有的表格单元格（马上会介绍这个函数），然后遍历记录集的字段。在行头为每个字段创建一个新单元格（这个表格只有两行：第一行，即第 0 行，是表头；第二行，即第 1 行，是表体）。表格单元创建完后，将 innerHTML 属性设为对应的字段名。在表体中创建新单元格的过程类似，但此时使用 innerHTML 元件保存绑定到数据字段的 INPUT 标记。当所有的字段都完成这样的操作后，这个表就与数据控件绑定了。

因为这个页面允许在两个不同的数据集之间进行切换，所以需要先删除现有的数据。

```
function deleteCells()
{
    var intCell;
    var intCells;

    // Unbind the table
```

```
tblData.dataSrc = '';

// Delete existing cells
intCells = tblData.rows[0].cells.length
for (intCell = 0; intCell < intCells; ++intCell)
{
    tblData.rows[0].deleteCell();
    tblData.rows[1].deleteCell();
}

</SCRIPT>
```

这个子程序只是对表解除绑定，然后在表格中遍历所有的单元格并删除它们。等到上述程序执行完毕，表格就只剩下空的表头和表体行。

这是一个用RDS和一些DHTML实现的简单例子。可以容易地将其加到一个 ASP包含文件中，并把该文件放到任何应用程序中，即使数据源不改变也可使用这种方法。

这个例子的全部源代码——文件RDSDynamicBinding.asp以及类似的其他类型的数据控件例子，可以在 Wrox的Web站点上找到。

### 10.2.6 更新数据

迄今为止，仅学习了在客户端如何取到数据，但还没有涉及如何更新客户端数据，和将其送回服务器。别忘了，记录集是断开连接的，那么如何更新数据呢？对数据所做的任何修改只是数据控件中本地记录的一部分，因此为了更新服务器必须发一条特殊的指令。然而这并不需做什么复杂的工作，因为 RDS数据控件有两个方法，允许我们要么取消最近对数据所做的任何修改，要么将所有修改送到服务器。

为了方便用户，可以为此创建一些按钮。

```
<BUTTON ID="cmdCancelAll" TITLE="Abandon All Changes"
ONCLICK="dsoData.CancelUpdate()">Cancel</BUTTON>&nbsp;

<BUTTON ID="cmdUpdateAll" TITLE="Save All Changes"
ONCLICK="dsoData.SubmitChanges()">Save</BUTTON>
```

SubmitChanges方法只将那些改动过的记录送回服务器，而 CancelUpdate方法则取消在本地记录集上所做的任何修改。

更新和取消更新操作并不是唯一所需的。如果想增加新的记录或删除一条现有的记录，怎么办？可以使用记录集的 AddNew和Delete方法。这将增加或删除记录集中的记录，然后在发送SubmitChanges命令后，服务器上的数据就可以被更新。

```
<BUTTON ID="cmdDelete" TITLE="Delete This Record"
ONCLICK="dsoData.recordset.Delete()">Delete</BUTTON>&nbsp;

<BUTTON ID="cmdAddNew" TITLE="Add New Record"
ONCLICK="dsoData.recordset.AddNew()">Add</BUTTON>&nbsp;
```

#### 1. 解决冲突的方法

由于与数据源断开连接，可能会碰到有关冲突的问题。例如在更新一条记录并将其保存到数据存储的时候，有人也修改了这条记录时，会发生什么情况？ SubmitChanges方法已经提

供了相应的处理冲突的方法，如果发生冲突，那么该方法将产生一个错误。

在调用SubmitChanges方法期间，只要其中一条记录更新失败，那么所有的记录更新都会失败。这保证了原始数据不会被部分更新。可以遍历记录集，并检测记录的 Status属性来告诉用户哪一条记录更新失败了。例如，最好调用自己的 updateData函数，而不只是在命令按钮中调用SubmitChanges方法。

```
function updateData()
{
```

该函数首先在一个try语句块中使用SubmitChanges方法，这样就可以捕获错误。

```
// Try and update the data
try
{
    // Submit the changes back to the server
    dsoData.SubmitChanges();
    dsoData.Refresh();
}
```

如果发生错误，可以在catch块中进行处理。

```
// RDS throws an error if the update fails
catch (e)
{
    var rsConflicts;
    var adRecUnmodified = 0x0000008;    // Record was not modified
    var adAffectAll = 3;                // resync all records
    var adResyncUnderlyingValues = 1;    // Only resync underlyingValue
```

此时，我们知道已经发生了一个错误，但并不知道是哪一个错误，因此必须重新同步当前数据与数据存储中的数据。使用 adResyncUnderlyingValues确保只有字段的 UnderlyingValue 属性被数据存储中的值覆盖，也就是说所做的修改是安全的（记住，修改的内容保存在 Value 属性中）。可以在后面的代码中比较当前的值与数据库中的值。

```
// There's been an error, so get the underlyingValues back
// from the server to see what's changed
dsoData.recordset.resync(adAffectAll, adResyncUnderlyingValues);
```

下一步是复制记录集，为的是能在不影响数据控件当前位置的情况下移动记录。

```
// Clone the recordset so we don't alter the position of the original
rsConflicts = dsoData.recordset.clone();
```

然后开始遍历记录，检查每个记录的 Status属性，只处理那些经过修改的记录。

```
// Loop through the records in the recordset
while (!rsConflicts.EOF)
{
    // We're only interested in records that are modified in some way
    if (rsConflicts.status != adRecUnmodified)
    {
        // Do something with the record
    }
    rsConflicts.moveNext();
}
}
```

Status可以是不同值的组合，详见附录。例子代码(RDSConflicts.asp)中有一个将这

些值转换为描述性字符串的函数。

我们知道记录有某些形式的冲突，但无法确切地知道为什么或哪一个字段引起了冲突。因此需要遍历字段检测它们的值。

```
for (fldF = new Enumerator(rsConflicts.Fields);
    !fldF.atEnd(); fldF.MoveNext())
{
    if (fldF.item().originalValue != fldF.item().underlyingValue)
    {
        // Do something
    }
}
```

这就是UnderlyingVaule属性发挥作用的地方。

字段有三种值：

- Value代表新值，即经过修改的字段值。
- UnderlyingValue代表数据存储中存储的字段值。
- OriginalValue代表从数据存储读取后，但还没有修改之前的字段值。

这意味着UnderlyingValue会保存其他用户修改过的值，而OriginalValue是字段原有的值。因此比较两者之值，如果不同，则说明字段已经被另外的用户修改了。

可以利用所有这些错误信息来创建一个表格以显示是否确实发生错误。例子(RDSConflicts.asp)产生的输出结果如图10-11所示。

The screenshot shows a web browser window titled "RDSConflicts.asp - Microsoft Internet Explorer". The page content is titled "Single Record Data Binding with RDS". It features a form with fields for ID, First Name, Last Name, Phone, Address, City, State, Zip, and Contact. Below the form are buttons for "Delete This Record", "Add New Record", "Cancel All Changes", and "Save All Changes". A message states: "An error occurred whilst updating the data. The following table details the errors:". Below this is a table with 5 columns: Status, Field, Your Value, Value in data store, and Value before you changed it. The table contains two rows of error data.

Status	Field	Your Value	Value in data store	Value before you changed it
The record was modified by another user.	lau_name	Whiter than white	Whiter	White
The record was modified by another user.	lau_name	Andy	John	Johnson

图10-11 RDSConflicts.asp产生的输出结果

这里可以见到三种不同的值。原始值是 Johnson。然后，在另一个窗口（如SQL Server Query Analyzer)中将值改为Johns。在浏览器窗口，利用RDS将这个值改为Andy，并按下Save All Changes按钮。Resync命令将数据库中的值取出并写入UnderlyingValue属性。我也对Last name列做了相似的修改。

使用这种方法，可以看到每一个发生变化的字段的值。由于SubmitChanges方法可以处理



多个字段，读者可能希望为这个表增加额外的列以显示 ID 字段，这样就可以看到是哪一个字段更新失败了。

### 10.3 在服务器和客户之间传输数据

在本章的开始，已经向大家提供了一张 RDS 的组件框图，确定了哪些组件是客户端的，以及哪些组件是服务器端的。在上一节，集中介绍了客户端组件，即数据控件，因为它们是使用 RDS 最简单的方法。它们允许从服务器向客户机无缝地传输数据，同时也提供了一种在 Web 页面上处理数据的简易方法。为了获得服务器上的数据，只需要设置几个属性，调用一、两个方法就足够了。

使用数据控件存在的问题在于它们获取数据的能力相对有限，即缺乏灵活性。对于 RDS 数据控件，必须通过参数，比如 Connection 和 SQL，指定连接和查询的细节，这把我们限制在数据库和简单的查询上。此外，用户还能看到这些参数，因此在使用 RDS 数据控件时存在一个潜在的安全漏洞。虽然属性 URL 通过指定实际的数据源，而不是获取数据的场所，部分解决了这个问题，但仍然存在缺陷。用户依旧可以获取 URL，直接读取数据，而这并不是我们所希望的。

IE 5.0 的引入使访问 XML 数据变得更为容易，因此以后 XML 数据肯定会广泛使用。与 RDS 数据控件的 URL 属性一样，用户也能直接读取数据，因为数据源的名称对他们可见的。

因此，下面一些问题需要考虑。

- 需要一种安全的方法检索服务器端数据，并在客户端使用它们。
- 需要一种更新数据的方法。
- 不损失灵活性。

这些问题的解决方法是使用组件。组件允许封装对所有数据的访问，包括数据源，确保只在 Web 页面上显示数据。可以隐藏连接的细节，因为它们在组件内部，不会暴露给用户，所以安全性不会受到损害。也可以创建自己的组件来更新数据，这样保留了可更新数据的灵活性。

此外又增加了安全性，因为在客户端访问服务器组件之前，需要修改注册表。这确保只创建所需的组件，阻止用户上传组件，然后从服务器访问他们。稍后将会看到如何修改注册表。

#### 10.3.1 基于服务器的组件

在这里并不打算大篇幅地讨论如何创建组件，因为本书后面的部分章节已经覆盖了这些内容，这里将介绍一些十分简单的用于服务器组件的代码。这有助于说明将要使用的 RDS 例子。

首先组件由一个定义连接细节的常数开始，在本例中针对 SQL Server 和 pubs 数据库。

```
Private Const wroxConnection As String = "Provider=SQLOLEDB; " & _  
    "Data Source=Kanga; Initial Catalog=pubs; User Id=sa; Password="
```

然后创建第一个方法，向客户端 Web 页面提供一个记录集。这是一个标准的 Visual Basic 函数，返回一个标准的 ADORRecordset 对象。

```
Public Function GetAuthors() As ADODB.Recordset
```

```

Dim objConn As New ADODB.Connection
Dim objRecAuthors As New ADODB.Recordset

objConn.Open wroxConnection

objRecAuthors.CursorLocation = adUseClient
objRecAuthors.Open "authors", objConn, adOpenKeyset, _
    adLockBatchOptimistic, adCmdTable

Set objRecAuthors.ActiveConnection = Nothing
Set GetAuthors = objRecAuthors

Set objRecAuthors = Nothing
objConn.Close
Set objConn = Nothing

End Function

```

这里有两个重要的地方需要注意。第一是把 CursorLocation 设为 adUseClient，确保使用客户光标引擎。这对于断开连接的记录集是必要的。第二点需要注意的是下面这一行代码：

```
Set objRecAuthors.ActiveConnection = Nothing
```

把记录集从服务器端“断开”，安全地将记录集返回给客户端。

组件的第二个方法用我们对断开的记录集的修改来更新服务器。它接受一个记录集参数，重新连接到数据存储，然后发出 UpdateBatch 命令。这将所有批处理修改送到服务器。

```

Public Function UpdateAuthors(ByRef recA As ADODB.Recordset) As Boolean

    On Error GoTo UpdateAuthors_Err

    recA.ActiveConnection = wroxConnection
    recA.UpdateBatch

    UpdateAuthors = True

UpdateAuthors_Exit:
    Exit Function

UpdateAuthors_Err:
    UpdateAuthors = False
    Resume UpdateAuthors_Exit

End Function

```

这就是这个组件的全部内容。很明显，我们把它简单化了，这样才能很快地显示其包含的内容，但实际上对于能在组件里放什么并没有任何限制。在本书第 13 章到第 18 章会看到更多的有关如何做以及为什么要这样做的例子。

### 10.3.2 DataSpace 对象

DataSpace 对象是负责与服务器进行通讯的客户端对象。它的任务是提供一种方法，从客户端创建服务器端组件，同时允许在客户机和服务器之间传输数据。

虽然听起来有些复杂，但实际上却是非常简单。在 Web 页面上创建一个 DataSpace 对象，再使用 DataSpace 对象创建服务器端组件。

创建一个 DataSpace 对象有两种方法。第一种方法使用 < OBJECT > 标记：

```
<OBJECT CLASSID="clsid:BD96C556-65A3-11D0-983A-00C04FC29E36"  
  ID="dspDataSpace" HEIGHT="0" WIDTH="0">  
</OBJECT>
```

这与过去创建其他客户端RDS组件的方法几乎相同，唯一不同的就是类ID。

第二种方法是使用脚本代码创建DataSpace对象。下面的代码显示了如何用JScript创建一个DataSpace对象。

```
var dspDataSpace = new ActiveXObject('RDS.DataSpace');
```

两种方法的主要区别在于：使用<OBJECT>标记时，DataSpace对象只在页面加载时才创建。使用脚本技术意味着当脚本运行时，就创建了DataSpace对象。如果想延迟对象创建以使页面加载得快一些，那么可以使用脚本技术。

利用DataSpace对象创建服务器组件

一旦创建了DataSpace对象，可以使用CreateObject方法创建服务器端对象。

```
var = dataspace.CreateObject(ProgID, Connection)
```

ProgID是希望创建的对象的ID，Connection是Web服务器的URL地址。例如，假设有一个提供pubs数据库数据的Visual Basic组件，如图10-12所示。

在这里，ProgID应该是WroxPubs.pubs。下面以一个完整的例子详细说明。

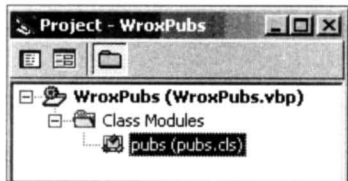


图10-12 使用pubs数据库的VB组件

#### (1) 创建组件和获取数据

假定pubs类有两个方法，getAuthors和setAuthors，用于读取和更新authors表。在创建组件之前，先创建一个RDS数据控件来存放和绑定数据。注意，这里没有连接的细节。

```
<OBJECT CLASSID="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"  
  ID="dsoData" HEIGHT="0" WIDTH="0">  
</OBJECT>
```

然后可以使用以下客户端脚本代码创建服务器组件：

```
// Create the data space  
var dspDataSpace = new ActiveXObject('RDS.DataSpace');  
  
// Create the custom object  
var objPubs = dspDataSpace.CreateObject('WroxPubs.pubs', 'http://localhost');
```

这里创建了DataSpace对象的一个实例，然后使用CreateObject方法创建服务器端对象。这里，指定了对象和Web服务器的名称。在本例中将Web服务器的名字设为localhost，它应该是服务器的实际名称。这不是一个安全问题，因为用户正在与Web站点连接。

现在，变量objPubs是WroxPubs.pubs对象的一个实例，可以像使用其他对象一样调用其方法。

```
// Call the method of the object to return a disconnected recordset  
var rsAuthors = objPubs.getAuthors();
```

调用getAuthors方法返回一个断开连接的记录集。现在需要做的就是使用这个记录集，因此可以使用一个RDS数据控件(用<OBJECT>标记创建)来存放记录集并绑定数据。使用数据控件的SourceRecordset属性来接收从组件中获取的记录集。

```
// Set the recordset to the data control  
dsoData.SourceRecordset = rsAuthors;
```

在这种情形下，数据控件表现得就像自己在获取数据一样，唯一不同的是我们已经从组件中提供了数据。

## (2) 使用组件更新数据

读者可能想知道使用服务器端组件时，如何更新数据？不能使用数据控件的 SubmitChanges 方法，因为数据控件没有连接的细节，因此不知道数据从哪里来。既然记录集是一个标准的断开连接的记录集，其数据又是由服务器组件提供的，那么组件似乎应该知道数据的来源？是的。因此需要把这个断开连接的记录集传给服务器组件，它会执行批处理更新。

为了实现这一点，首先从数据控件中取出记录集，然后将其作为参数传给服务器对象的方法。

```
rsAuthors = dsoData.recordset;  
  
objAuthors.setRecordset(rsAuthors);
```

实际上这相当简单。

### 10.3.3 使用服务器端组件的优点

那么服务器端组件最大的特色是什么？看一下它的优点：

- 连接细节对用户不可见，为我们提供了一个更安全的系统。同时，组件也必须被注册，这又是一道附加的安全限制。
- 可以用任何兼容 COM 的语言来创建组件。在例子中使用 Visual Basic 的原因是因为 VB 容易理解，但如果需要也可以使用 VC++ 或 Delphi。这允许开发人员使用自己最熟悉的语言。
- 创建的组件并不只局限于返回和更新记录集。可构建大量封装的业务逻辑作为数据处理的一部分。
- 可以在 setRecordset 方法中封装全部冲突处理的代码，而只返回错误的详细说明。这表明客户端脚本是规范的。同时，也意味着用户无法窃取你精心编写的代码。

本书的下一节会详细介绍组件的优点。

### 10.3.4 注册服务器端组件

在前面曾经说过，在 Web 浏览器中创建一个服务器组件之前必须修改注册表。为了让 RDS 能使用组件，必须将组件的 ProgID 添加到下面的注册表键中：

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\W3SVC\Parameters\  
ADCLaunch
```

**警告：**在编辑注册表时始终要很仔细。在修改之前备份注册表永远是明智的。

运行 regedit 后，选择 ADCLaunch 键，然后从 Edit 菜单中选择 New，再选择 Key。现在为组件输入 ProgID，例如图 10-13 所示。

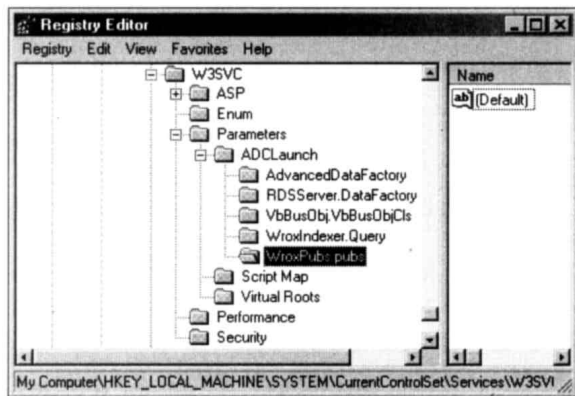


图10-13 编辑注册表的界面

还应该考虑对注册表做一点修改以标明组件为“safe for scripting”，这样在运行的时候，才会完全满足IE的要求。此时需要增加两个键，两个都需要组件的 GUID。

```
·HKEY_CLASSES_ROOT\CLSID\{your_component_class_id}\  
  Implemented Categories\{7DD95801-9882-11CF-9FA9-00AA006C42C4}
```

```
HKEY_CLASSES_ROOT\CLSID\{your_component_class_id}\  
  Implemented Categories\{7DD95802-9882-11CF-9FA9-00AA006C42C4}
```

实现的最好方法是使用 Visual studio 的 Package and Deployment Wizard 创建一个 Internet 下载软件包 (Internet Download Package)。如果选择了 Safe for Initialization 和 Safe for Scripting 这两个选项，那么当软件包安装到服务器时，将会更新注册表。如果确实需要得到组件的类 ID，那么这个向导也能创建一个 HTML 文件，里面是一个带有组件类 ID 的 < OBJECT > 标记。

### 10.3.5 自定义处理器

RDS 数据控件是一个优秀的客户端工具，可在客户端处理大量的数据。大多数例子已经表明数据控件使用的是固定的数据集，比如 authors 表。然而，动态数据绑定的例子表明了也可以让用户决定自己想要检索的数据。这可以通过创建按钮来实现，可为用户提供一个文本框来输入表名，甚至输入 SQL 查询字符串。

当然，这增加了 RDS 数据控件的灵活性，但却是以牺牲安全性为代价的。如果用户对数据的使用范围不受限制，他们要么会觉得混乱，要么就想偷看那些保密的财务数据。因而我们需要的是一种灵活的，但又受一定限制的检索数据的方法。

为了做到这一点可以使用一个处理器，可以在连接字符串或命令执行之前修改它们。

处理器是一个包含四部分内容，分别由回车符分隔的文本文件：

- Connect 节：确定连接字符串。
- SQL 节：确定 SQL 语句和命令。
- UserList 节：确定用户。
- Logs 节：确定日志记录能力。

该文本文件名缺省为 Msdfmap.ini，安装在 Windows 缺省目录中。

#### 1. Connect 节

Connect节允许覆盖连接字符串和访问细节，由下面两个条目之一标识：

- [connect default] 指出缺省的连接细节。
- [connect *ConnectionName*] 指出名为 *ConnectionName* 的连接细节。

在该节有两对名称/值，如表 10-6 所示。

表10-6 Connect节的名称/值对及说明

名 称	说 明
Connect	使用的连接字符串
Access	针对此连接的访问权限。可以是下列值之一： NoAccess 表明对数据源没有访问权 ReadOnly 表明只有读取数据的权力 ReadWrite 表明有读和写数据的权力

例如：

```
[connect PubsDatabase]
Connect="DSN=Otherpubs; UID=GuestUser; PWD=JustAGuest"
Access=ReadOnly
```

这表明，如果数据控件的连接字符串为 PubsDatabase，用户将以 GuestUser 的身份登录到数据库，并且只有读的权限。因此，如果用户执行以下操作：

```
conDB.Open "PubsDatabase"
```

那么实际上他们会连接到 Otherpubs 数据源。

对于缺省的连接，输入的内容可能是：

```
[connect default]
Access=NoAccess
```

这确保了对于在处理器文本文件中没有给予详细说明的连接不能进行访问。

2. SQL 节

SQL 节允许覆盖 SQL 字符串和命令，由下面两个条目之一标识：

- [sql default] 指出缺省的 SQL 命令。
- [sql *SQLCommandName*] 确定一个特殊的 SQL 命令，例如存储过程或存储查询。

在此节只有一个条目，它是一条用于取代提供的语句的 SQL 语句。例如：

```
[sql GetAuthors]
Sql="SELECT * FROM Authors"
```

上面的处理器条目允许在一个 RDS 数据控件中使用下面的代码：

```
<PARAM NAME="SQL" VALUE="GetAuthors">
```

也可以在 SQL 中使用参数化查询，比如：

```
[sql AuthorsByState]
sql="SELECT * FROM authors WHERE state=?" :
```

在这种情况下，执行的命令将会是：

```
<PARAM NAME="SQL" VALUE="AuthorsByState('CA')">
```

因此，虽然客户能看到 SQL 字符串，但实际上并不知道真正执行的 SQL 语句。

为了使任何 SQL 值无效，可使用下面的代码：



```
[sql default]
Sql=" "
```

这禁止任何依赖于指定连接的命令。

### 3. UserList节

UserList节允许为各个用户指定访问权限，与 Connect节相同，由 ConnectionName标识。

- [userlist ConnectionName] 确定使用指定连接的用户。

在此节只有一个值，是用户的访问权限。访问权限的值与 Connect节中显示的值相同。

例如：

```
[userlist PubsDatabase]
Administrator=ReadWrite
DavidS=ReadWrite
Guest=NoAccess
```

这里的值覆盖了 Connect节中的访问权限值。

为了正确使用 UserList节，服务器必须能够验证用户，因此这取决于服务器上的验证规则。

例如，如果使用匿名访问，那么所有用户都会以 Web站点中 IIS管理属性所指定的缺省用户身份连接到服务器。

### 4. Logs节

Logs节允许指定一个写入错误的文件名由 [logs]标识。日志记录应用于所有连接。如果此日志文件不存在则会创建一个新的日志文件。

在此节只有一个值 err，它指定日志文件名，例如：

```
[logs]
err=C:\Temp\DFErrors.txt
```

日志文件将包含用户名、错误代码以及每个错误发生的日期和时间。

### 5. 自定义处理器文件示例

在一个文本文件中组合了所有上述的节。例如：

```
[connect PubsDatabase]
Connect="DSN=Otherpubs; UID=GuestUser; PWD=JustAGuest"
Access=ReadOnly
```

```
[userlist PubsDatabase]
Administrator=ReadWrite
DavidS=ReadWrite
Guest=NoAccess
```

```
[connect default]
Access=NoAccess
```

```
[sql GetAuthors]
Sql="SELECT * FROM Authors"
```

```
[sql default]
Sql=" "
```

```
[logs]
err=C:\Temp\DFErrors.txt
```

### 6. 使用自定义处理器

可以使用 RDS数据控件的 Handler属性来指定处理器，并给指定处理器的名字。缺省的处

理器是MSDFMAP.Handler，它是RDS服务器端部分使用的一个组件。也可以编写自己的自定义处理器(标准的COM组件)，如果这样做，应该参考RDS的文档。

为了使用缺省的处理器，可以使用如下代码：

```
<OBJECT CLASSID="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"
    ID="dsoAuthors" WIDTH="0" HEIGHT="0">
    <PARAM NAME="Connect" VALUE="connection string">
    <PARAM NAME="Server" VALUE="server name ">
    <PARAM NAME="SQL" VALUE="query text">
    <PARAM NAME="Handler" VALUE="MSDFMAP.Handler">
</OBJECT>
```

以上代码将调用处理器，并使用缺省的文件。可以在 Handler属性的最后指定自己的文件。

```
<PARAM NAME="Handler" VALUE="MSDFMAP.Handler, MyHandler.ini">
```

处理器文件始终位于 Windows 目录。

10.4 记录集分页

研究RDS数据控件时，学习了如何用 DATAPAGESIZE来限制绑定表中显示的记录行数。虽然这是一个既好又简单的解决方法，但其缺点是把所有数据都取到了客户端。例子中只用到了一个规模较小的数据集，但如果需要用到一个很大的数据集，结果会如何？真的想把所有数据都取到客户端吗？毕竟，用户每次只能看到一屏数据。

这个问题的解决方法是引入分页的概念，每一次只处理一页数据，并且只返回这一页数据，而不是全部数据。为了实现分页处理数据，需要处理与 Recordset对象相关的三个属性，如表10-7所示。

表10-7 与Recordset对象相关的三个属性及说明

属 性	数据类型	说 明	缺 省 值
PageSize	长整型	一个页面中的记录数	10
PageCount	长整型	记录集的总页数	
AbsolutePage	长整型	当前页码	

可使用AbsolutePage属性将记录指针指向某页面中的第一条记录，这样可以直接在页面间移动。例如：

```
rsAuthors.AbsolutePage = 2
```

如果使用缺省的页面大小 10，上面的代码行则将记录指针指向记录集的第 11 条记录。

需要着重注意的是，页面中的记录不是固定的，记录的数量是固定的，但并不是记录本身。这听起来可能不太好理解，但如果回想一下讨论光标类型的第 8 章内容，可能会记得某些类型的光标是基于键的。实际的数据并不会被读取到客户端，直到请求这些数据行。这样，如果正在使用分页，而其他用户增加了新的行或删除了现有的记录，那么在页面中的实际记录就可能会改变。

同样需要记住的是，页面中记录的次序与记录集中的次序是一致的，由产生记录集的命令来定义。记录集中的记录没有记录号，因而无法确定它们在记录集中的位置。

将分页结合到应用程序有好几种方法，其中的两种使用这些属性，第三种方法使用类似的方法，但只适用于 SQL Server。

### 10.4.1 利用ASP页面分页记录集

第一个要讨论的方法是在本章前面介绍过的 RDS方法的一个扩展。我们将使用 RDS数据控件和 URL 参数，先有一个返回一组记录的 ASP 页面，然后这些记录由 RDS 数据控件使用。如果想引入分页，必须确定这个 ASP 页面只返回一个充满数据的页面。

先研究提供数据的 ASP 页面。为使其更具有灵活性，允许一个包含两条信息的查询字符串传到页面中：

- TABLE，确定要从中获取数据的表。
- PAGE，确定请求的页号。

假定 ASP 页面的名称为 RDSURLPagedData.asp，那么查询语句类似于：

```
RDSURLPagedData.asp?TABLE=authors&PAGE=2
```

在 ASP 页面中，首先要做的是创建一个 Recordset 对象，这与前面没有什么两样。

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

接下来，取出请求的细节。除了表名，同时也得到了页码。这就是请求的数据页。

```
' Get the requested data
strTable = CStr(Request.QueryString("TABLE"))
intPage = CInt(Request.QueryString("PAGE"))
```

现在设置页面大小。采用每页 10 条记录的分页法，当然可以很容易地进行改变。甚至可以让用户来设置页面的大小，并与表名和页码一起作为参数传入 ASP 页面。

```
' Set the page size
rsData.PageSize = 10
rsData.CursorLocation = adUseClient
```

现在基于请求的表打开记录集。

```
' Open the data
rsData.Open strTable, strConn, _
    adOpenForwardOnly, adLockReadOnly, adCmdTable
```

此时，得到了一个充满数据的记录集，但仅需要一个页面。因此，必须使用 AbsolutePage 属性来设置想要的页面。在设置 AbsolutePage 属性之前，需要检测请求的页码不是 0，或不大于总页数。在这两种情况下，将页面设为最后一页，这由 PageCount 属性确定。

```
' Set the page
If intPage = 0 Or intPage > rsData.PageCount Then
    rsData.AbsolutePage = rsData.PageCount
Else
    rsData.AbsolutePage = intPage
End If
```

现在，已经位于正确的页上，所以接下来的问题就是如何将这一页传送给客户端。为了实现这一点，只需用一个“转储”记录集把该页中的记录提取出来。创建一个新的，与含有数据的记录集相同结构的记录集，同时将数据拷贝到新的记录集中。

转储一个记录集比较简单，包括创建一个 Recordset 对象，设置光标位置为基于客户端。然后将字段添加到 Fields 集合，使用数据记录集为这些字段提供细节。一旦创建完字段，只需简单地打开记录集，不必指定任何数据源或连接的细节。

```
' Create the new recordset
Set rsNew = Server.CreateObject('ADODB.Recordset')
rsNew.CursorLocation = adUseClient
For Each fldF In rsData.Fields
    rsNew.Fields.Append fldF.Name, fldF.Type, fldF.DefinedSize, fldF.Attributes
Next
rsNew.Open , , adOpenKeyset, adLockOptimistic
```

此时，得到了一个空的新记录集，因此需把一页记录拷贝到此记录集中。这需要遍历页面中的记录，把每一个记录都添加到新的记录集中。

```
' Now append the data, but only the number of records in a page
For intRec = 1 To rsData.PageSize
    If Not rsData.EOF Then
        rsNew.AddNew
        For Each fldF In rsData.Fields
            rsNew.Fields(fldF.Name) = fldF.Value
        Next
        rsNew.Update
        rsData.MoveNext
    End If
Next
```

现在得到了一个只含有一页数据的记录集，这样就可以发送给客户端。

```
' Send the new data back to the client
rsNew.Save Response, adPersistXML
```

最后，关闭记录集。

```
' And clear up
rsData.Close
Set rsData = Nothing

rsNew.Close
Set rsNew = Nothing
```

```
%>
```

因此，现在有了一个只返回单一数据页的 ASP 页面，余下的工作是修改客户端 RDS 页面。RDS 数据控件由 < OBJECT > 标记创建，与前面介绍的 URL 的例子一样。

```
<OBJECT CLASSID="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"
    ID="dsoData" HEIGHT="0" WIDTH="0"
    ondatasetcomplete="createCells()">
</OBJECT>
```

选择表的按钮现在都调用名为 setTable 的子程序。

```
<BR>
Click one of the buttons below to see the data
<P>
<BUTTON ID="cmdAuthors" onclick="setTable('authors')">authors</BUTTON>
<BUTTON ID="cmdPublishers" onclick="setTable('publishers')">publishers</BUTTON>

<TABLE ID="tblData" BORDER="1">
    <THEAD><TR></TR></THEAD>
    <TBODY><TR></TR></TBODY>
</TABLE>
```

我们用新创建的按钮来控制分页，这些按钮调用 setPage方法，其参数确定要移到哪一页。

```
<BR>
<BUTTON ID="cmdFirstPage" ONCLICK="setPage('First')">First Page</BUTTON>
<BUTTON ID="cmdPreviousPage" ONCLICK="setPage('Previous')">
    Previous Page</BUTTON>
<BUTTON ID="cmdNextPage" ONCLICK="setPage('Next')">Next Page</BUTTON>
<BUTTON ID="cmdLastPage" ONCLICK="setPage('Last')">Last Page</BUTTON>
```

接下来是脚本程序。我们声明了两个全局变量，用来确定要查看的表和当前的页码。

```
<SCRIPT LANGUAGE=JScript>
```

```
var m_strFile;
var m_intPage = 1;
```

设置表名的程序只是在重置数据之前把页码设为第一页，同时将表名设为已定义的表。

```
function setTable(strTable)
{
    m_intPage = 1;
    m_strFile = strTable;
    resetData();
}
```

重置数据与以前使用过的方法大致相似，只是在 URL后增加了一部分内容，现在除了提供表，也提供页码。

```
function resetData()
{
    // Reset the data
    dsoData.URL = 'RDSURLPagedData.asp?TABLE=' + m_sFile +
        '&PAGE=' + m_iPage;
    dsoData.Refresh();
}
```

最后，改变页的程序如下，它仅仅是设置页码并重置数据。

```
function setPage(strPage)
{
    switch (strPage)
    {
        case 'first':
            m_intPage = 1;
            break;

        case 'Previous':
            m_intPage = m_intPage - 1;
            if (m_intPage == 0)
            {
                m_intPage = 1;
            }
            break;

        case 'Next':
            m_intPage = m_intPage + 1;
            break;

        case 'Last':
            m_intPage = 0;
            break;
    }

    resetData();
}
```

```
}

```

```
</SCRIPT>

```

以上代码将会产生图 10-14 所示的页面。



图10-14 记录集分页的结果页面

尽管如此，这段代码存在着一个主要的问题需要解决，因为客户并不知道有多少页数据。可以很容易地直接请求到最后一页数据，但是当一页页下移时，却不知道什么时候超出最后一页。ASP页面能够知道，但在客户端却不行。这意味着即使 ASP页面只显示最后一页，客户端的页号仍然在持续增加。

#### 使用自定义组件

解决前面例子中的页码问题的一种方法是使用组件来提供数据，而不是 ASP页面。这里不想进行仔细的研究，但这个概念非常类似。首先要清除 URL的细节，然后在 resetData函数中使用 DataSpace对象和 DataFactory对象来创建一个自定义组件。

这个组件所含的代码可能与 ASP页面的代码非常相似，也返回一个含有数据页的记录集。但由于组件可以不止返回一个记录集，所以实际上可以跟踪页的最大数量，这使分页变得更智能化。

#### 10.4.2 利用ADO分页

另一个解决页码问题的方法是使用一种著名的分页技术。不采用 RDS数据绑定，而是在 ASP页面内创建一个 HTML表格。使用相同的分页方法，但并不转储记录集。

下面分析 ASPPaging.asp的代码，然后将看到会产生什么样的结果。

首先声明变量并创建 Recordset对象。

```
<%
Dim rsData
Dim intPage
Dim intTotalPages
Dim fldF
Dim intRec
Dim strQuote

```



```
Dim strScriptName
```

```
strQuote = Chr(34) ' The double quote character
```

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

然后，设置页的大小和光标的位置。

```
' Set the page size
```

```
rsData.PageSize = 10
```

```
rsData.CursorLocation = adUseClient
```

接下来，打开记录集。

```
' Open the data
```

```
rsData.Open "Authors", strConn, _
```

```
adOpenForwardOnly, adLockReadOnly, adCmdTable
```

现在，必须设置页码。第一次显示时 QueryString 的值可能为空，因此缺省设为第一页。如果指定了页，那么检测是否超出可接受的页范围。如果小于第一页就设为第一页，如果大于最后一页就设为最后一页。

```
' Get the requested data
```

```
If Request.QueryString("PAGE") = "" Then
```

```
intPage = 1
```

```
Else
```

```
' Protect against out of range pages, in case
```

```
' of a user specified page number
```

```
If intPage < 1 Then
```

```
intPage = 1
```

```
Else
```

```
If intPage > rsData.PageCount Then
```

```
intPage = rsData.PageCount
```

```
Else
```

```
intPage = CInt(Request.QueryString("PAGE"))
```

```
End If
```

```
End If
```

```
End If
```

然后，为请求的页面设置绝对页码。

```
' Set the page
```

```
rsData.AbsolutePage = intPage
```

当位于正确的记录位置时就可以开始创建 HTML 表格。首先是创建表头：

```
' Start building the table
```

```
Response.Write "<TABLE BORDER=1><THEAD><TR>"
```

```
For Each fldF In rsData.Fields
```

```
Response.Write "<TD>" & fldF.Name & "</TD>"
```

```
Next
```

```
Response.Write "</TR></THEAD><TBODY>"
```

现在可以遍历页内的记录，并创建 HTML 表格的表体。

```
' Now loop through the page
```

```
For intRec = 1 To rsData.PageSize
```

```
If Not rsData.EOF Then
```

```
Response.Write "<TR>"
```

```
For Each fldF In rsData.Fields
```

```
Response.Write "<TD>" & fldF.Value & "</TD>"
```

```
Next
```

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

```

        rsData.MoveNext
    End If
Next
Response.Write "</TBODY></THEAD></TABLE><P>"

```

接下来讨论允许分页的控件。它们只是些 <A> 标记，指向同一个页但页码却不同。第一页(First Page)实现起来比较简单，因为页码始终是 1。

```

' Now some paging controls
strScriptName = Request.ServerVariables("SCRIPT_NAME")
Response.Write "&nbsp;<A HREF=" & strQuote & strScriptName & _
                "?PAGE=1" & strQuote & ">First Page</A>"

```

我们不想总是给出一个向前翻页的控件，因为可能已经位于第一页数据。如果是在第一页，只需使用一个 SPAN 标记让 “ Previous Page ” 显示着，这样用户点击它时没有反应。

```

' Only give an active previous page if there are previous pages
If intPage = 1 Then
    Response.Write "&nbsp;<SPAN>Previous Page</SPAN>"
Else
    Response.Write "&nbsp;<A HREF=" & strQuote & sMe & _
                    "?PAGE=" & intPage - 1 & strQuote & ">Previous Page</A>"
End If

```

对下一页控件采用相同的技术，因为有可能已位于最后一页。

```

' Only give an active next page if there are more pages
If intPage = rsData.PageCount Then
    Response.Write "&nbsp;<SPAN>Next Page</SPAN>"
Else
    Response.Write "&nbsp;<A HREF=" & strQuote & strScriptName & _
                    "?PAGE=" & intPage + 1 & strQuote & ">Next Page</A>"
End If

```

最后一页控件实现起来也比较简单。

```

Response.Write "&nbsp;<A HREF=" & strQuote & strScriptName & _
                "?PAGE=" & rsData.PageCount & strQuote & ">Last Page</A>"

```

最后，关闭记录集。

```

' And clear up
rsData.Close
Set rsData = Nothing
%>

```

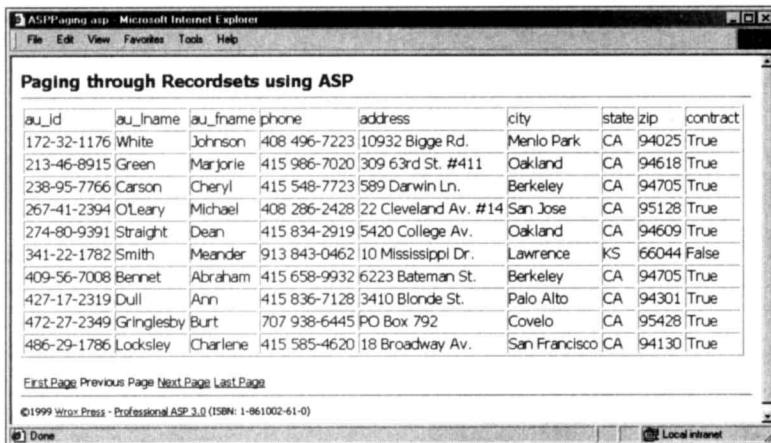


图10-15 一个简单的分页程序给出的结果

这就是ADO分页。一个相当简单的分页程序给出了图 10-15所示的结果。

这里显示的是第一页数据，因此可以看到上一页控件是不可用的。

这种解决方案给用户提供了一个较好的界面，因为控件以一种很实际的工作方式运转着。然而，仍然存在一个问题，向前翻页的实现使得每次请求页面时就要打开整个记录集。实际上所做的全部工作是把记录集从客户端移到了服务器。现在必须承认带宽方面是改进了，但很多请求的记录超出了我们所需要的。

### 10.4.3 利用SQL Server完成记录集分页

最后一种分页方法的实现也使用了相似的技术，但这次是在 SQL Sever中处理页面。这种方法是过去创建两层客户 / 服务器体系时常采用的一种方法，在这里似乎值得重新使用。利用SQL Server的存储过程创建一个临时表，然后在存储过程中只返回所需的记录。在某种意义上，这有点像转储记录集的解决方法，只是这个时候将转储移到 SQL Server中。

需要解决的主要是要选出请求的页中的记录。大多数关系数据库并没有记录号的概念，因此想挑出所需的记录比较困难。最简单的解决方法是复制一个表，增加一个唯一的有序的记录号，这样就可以从复制的表中取出所需的记录。在 SQL Server中用IDENTITY列很容易做到这一点，但对于基表我们不能依赖 IDENTITY列，因为记录可能会被删除，这样在编号中就会留下空缺。然而，可以创建一个临时表，加入 IDENDITY列，这样就能确保顺序编号。

看一下实现这一功能的存储过程。将请求的页以及页的大小作为参数传入存储过程：

```
CREATE PROCEDURE usp_PagedAuthors
    @iPage        int,
    @iPageSize    int
AS
BEGIN
```

首先，关闭自动行计数功能，以避免额外的信息返回到客户端。

```
-- disable row counts
SET NOCOUNT ON
```

接下来声明变量。

```
-- declare variables
DECLARE @iStart        int        -- start record
DECLARE @iEnd          int        -- end record
DECLARE @iPageCount    int        -- total number of pages
```

现在，创建与基表结构相同的临时表，但是增加了额外的 IDENTITY列。这里，明确地指定单个表，但也能动态地指定，允许表名由存储过程来提供。当然，动态指定表会降低存储过程的执行速度。

```
-- create the temporary table
CREATE TABLE #PagedAuthors
(
    ID            int            IDENTITY,
    au_id         varchar(11)    NOT NULL ,
    au_lname      varchar(40)    NOT NULL ,
    au_fname      varchar(20)    NOT NULL ,
    phone         char(12)       NOT NULL ,
```

```

address    varchar(40)    NULL ,
city       varchar(20)    NULL ,
state      char(2)        NULL ,
zip        char(5)        NULL ,
contract   bit            NOT NULL
)

```

临时表创建完后，在临时表中加入基表的全部记录。IDENTITY列的使用确保了记录拥有唯一的顺序编号的ID字段。

```

-- populate the temporary table
INSERT INTO #PagedAuthors (au_id, au_lname, au_fname,
                           phone, address, city, state, zip, contract)
SELECT    au_id, au_lname, au_fname,
          phone, address, city, state, zip, contract
FROM      authors

```

这样可以计算出总页数。

```

-- work out how many pages there are in total
SELECT    @iPageCount = COUNT(*)
FROM      authors

SELECT    @iPageCount = CEILING(@iPageCount / @iPageSize) + 1

```

接下来，可以检测请求的页码有没有超出页面的最大范围。

```

-- check the page number
IF @iPage < 1
    SELECT @iPage = 1

IF @iPage > @iPageCount
    SELECT @iPage = @iPageCount

```

为了只选择所需页的记录，必须计算该页开始和结束的记录号。

```

-- calculate the start and end records
SELECT @iStart = (@iPage - 1) * @iPageSize
SELECT @iEnd = @iStart + @iPageSize + 1

```

现在，可以只选择记录号位于该页中的记录。

```

-- select only those records that fall within our page
SELECT    au_id, au_lname, au_fname,
          phone, address, city, state, zip, contract
FROM      #PagedAuthors
WHERE     ID > @iStart
AND       ID < @iEnd

```

然后，删除临时表，并将记录计数功能打开。

```

DROP TABLE #PagedAuthors

-- turn back on record counts
SET NOCOUNT OFF

```

最后，返回页的总数。

```

-- Return the number of records left
RETURN @iPageCount
END

```

因此，最后就有了一个只返回一页记录的存储过程。现在需要用一些 ASP 代码来利用它。

从声明变量开始。

```
<%
Dim cmdAuthors
Dim rsData
Dim intPage
Dim intLastPage
Dim strQuote

strQuote = Chr(34)
```

与其他ASP页面类似，取出页码。

```
' Get the requested data
If Request.QueryString("PAGE") = "" Then
    intPage = 1
Else
    intPage = CInt(Request.QueryString("PAGE"))

    If intPage < 1 Then
        intPage = 1
    End If
End If
```

因为将要使用一个带参数的存储过程，所以此时需要创建两个对象。

```
' Create the objects
Set cmdAuthors = Server.CreateObject("ADODB.Command")
Set rsAuthors = Server.CreateObject("ADODB.Recordset")
```

不需要创建连接对象，因为ADO会创建一个隐含的连接。

接下来，为存储过程创建参数。第一个是返回值，保存着从存储过程中返回的页面总数；第二个和第三个参数被传入存储过程，分别指出了请求的页码以及页面的大小。

```
With cmdAuthors
    .ActiveConnection = strConn
    .CommandText = "usp_PagedAuthors"
    .CommandType = adCmdStoredProc

    .Parameters.Append .CreateParameter("RETURN_VALUE", adInteger, _
                                         adParamReturnValue)
    .Parameters.Append .CreateParameter("@iPage", adInteger, _
                                         adParamInput, 8, iPage)
    .Parameters.Append .CreateParameter("@iPageSize", adInteger, _
                                         adParamInput, 8, 10)
```

一旦设置了参数，就可以运行存储过程来返回记录。

```
Set rsData = .Execute
End With
```

现在开始创建表格。这与其他创建表格的程序是相似的，只不过此时记录集中只含有所需数量的记录。

```
' Create the table
' Start building the table
Response.Write "<TABLE BORDER=1><THEAD><TR>"
For Each fldF In rsData.Fields
    Response.Write "<TD>" & fldF.Name & "</TD>"
Next
Response.Write "</TR></THEAD><TBODY>"

' Now loop through the records
```

```

While Not rsData.EOF
    Response.Write "<TR>"
    For Each fldF In rsData.Fields
        Response.Write "<TD>" & fldF.Value & "</TD>"
    Next
    Response.Write "</TR>"
    rsData.MoveNext
Wend
Response.Write "</TBODY></THEAD></TABLE><P>"

```

接下来创建分页控件。创建第一页和上一页控件的代码与前面所见的一样。

```

' Now some paging controls
strScriptName = Request.ServerVariables("SCRIPT_NAME")
Response.Write "&nbsp;<A HREF=" & strQuote & strScriptName & _
    "?PAGE=1" & strQuote & ">First Page</A>"

' Only give an active previous page if there are previous pages
If intPage <= 1 Then
    Response.Write "&nbsp;<SPAN>Previous Page</SPAN>"
Else
    Response.Write "&nbsp;<A HREF=" & strQuote & strScriptName & _
        "?PAGE=" & intPage - 1 & strQuote & ">Previous Page</A>"
End If

```

创建下一页和最后一页控件需要总页数，它是存储过程的返回值，因此关闭记录集，然后从参数中取出相应的值。

```

' Close the recordset and extract the number of records left
rsData.Close
intLastPage = cmdAuthors.Parameters("RETURN_VALUE")

' Only give an active next page if there are more pages
If intLastPage = intPage Then
    Response.Write "&nbsp;<SPAN>Next Page</SPAN>"
Else
    Response.Write "&nbsp;<A HREF=" & strQuote & strScriptName & _
        "?PAGE=" & intPage + 1 & strQuote & ">Next Page</A>"
End If

Response.Write "&nbsp;<A HREF=" & strQuote & strScriptName & _
    "?PAGE=" & intLastPage & strQuote & ">Last Page</A>"

' Clean up
Set rsData = Nothing
Set cmdAuthors = Nothing
%>

```

这种分页方法与前面介绍的使用 ASP 页面分页的方法有些相似，但仍然有一个问题。必须创建一个含有全部基表记录的临时表。如果基表非常大，即使是一个快速的存储过程，执行起来也会很慢。顺便说一下，如果允许用户对一个含有大量记录的记录集进行分页，那么也必须接受性能损失。

#### 10.4.4 数据分页小结

到目前为止，已经讨论了三种不同的数据分页方法，但哪一种最好取决于设计的约束条件。

第一种方法使用 RDS，为用户提供了响应速度快的页面。因为所有的数据都在本地机器



上,不需要向服务器获取更多的数据。然而,由于所有数据必须立即传到客户端机器,页面加载的速度比其他页面要慢。这也是 IE解决方案的一个缺陷。另外两种方法能很快地将页面加载到浏览器,但分页则需要额外的时间。然而,这两种方法不需要任何特殊的客户端组件,比如RDS,因为表格是纯粹的HTML文档。

后两个方法的差别不易察觉。用 Microsoft Web Application Stress工具进行的严格测试表明,转储记录集的方法速度上稍快一点,但是区别很小。但如果安装 SQL Server的机器与Web服务器不同,那么这种差别还是很大的。因此,如果使用这两种方法中的一种,建议最好自己实际测试一下。Microsoft Web Application Stress 工具可以从网址 <http://homer.rte.microsoft.com>获得。

另一个使用SQL Server进行分页的方法是使用SQL Server光标。它类似于ADO中的光标与记录集,允许访问从查询中获得的数据行。一个光标创建并管理 SQL Server 中的一个行集。使用 SQL光标能从行集的一个绝对位置开始检索数据行,因此,理论上可以创建一个只返回一页从绝对位置开始的数据行的存储过程。不幸的是,SQL光标只允许一次访问一行数据,所以页面中的每一行数据都被作为一个单独的记录集返回。在客户端,我们就必须使用 NextRecordset方法对服务器执行额外的操作。因此,这种方法虽然看上去似乎不错,但实际上与这里介绍的使用存储过程的方法相比效率是比较低的。

## 10.5 使用数据库中的图像

图像处理仍然是令许多人感到困惑和麻烦的一个领域。至今为止,在 ASP页面中处理图像的最好方法是将图像保存到文件系统中而不是数据库中。

然而,需要从数据库中访问图像的情况是存在的,因此这里有一个访问图像的简单方法,假定图像数据是SQL Server的一个BLOB字段。使用微软的Access存储图像有一个额外的问题,图像的头信息存放于OLE对象字段中。这里不打算研究Access的解决方案。

首先,创建一个名为GetLogo.asp的ASP文件。

```
<!-- #INCLUDE FILE='../Include/Connection.asp' -->
<%
    ' Turn on buffering and set the mime type
    Response.Buffer = True
    Response.ContentType = "image/bmp"

    Dim rsLogo
    Dim strSQL
    Dim bytChunk

    strSQL = "SELECT logo FROM pub_info WHERE pub_id='" & _
        Request.QueryString("pub_id") & "'"

    ' Run the command and extract the logo
    rsLogo.Open strSQL, strConn
    bytChunk = rsLogo("logo")
    rsLogo.Close
    Set rsLogo = Nothing

    ' Write image to the browser
    Response.BinaryWrite bytChunk
```

```
Response.End  
%>
```

以上代码接收一个出版商的 ID，并用来查询数据库。通过使用 BinaryWrite 方法将图像写入 Response 流。可以在别的 ASP 页面中用其作为 image 标记的数据源。例如：

```
<IMG SRC="GetLogo.asp?pub_id=0736">
```

当然，也可以使用存储过程来返回图像以加快执行速度。

这个例子只返回单个图像，当然也可以获取一个包含不止一幅图像的记录集。

另一个使用图像的方法是使用 IMG 标记和数据绑定，IMG 标记绑定的字段是 SRC 属性 SRC。当然，这个属性使用一个 URL，因此不能直接绑定存放于记录集中的图像。实际上，也没有直接使用记录集中的图像并将其放到 Web 页面中的方法。

## 10.6 小结

正如在本章开始提到的，世界上不存在纯粹的 ASP 程序员。这里集中讨论了将数据传到客户端的不同技术。其中的一些技术使用了客户端脚本和 RDS，然而其他的使用纯粹的 ASP 代码来产生数据。

本章着重研究了以下几方面的内容：

- RDS 数据控件的不同类型，以及所存在的某些使用上的缺陷。
- 如何将 HTML 元素与 RDS 数据控件绑定以减轻编程负担和改善用户与数据的交互。
- 如何使用 DataSpace 和 DataFactory 创建和访问服务器上的自定义组件。
- 当使用 RDS 时，如何增加 Web 服务器的安全性。
- 几个不同的客户端数据分页的方法。

所有这些内容将能为 Web 应用程序的用户提供更好的感受。