

## 第16章 ASP脚本组件

在研究ASP时，一个首要问题是如何编写能够为ASP页面提供运行期文本的组件。三年来，我们使用了很多方法去解决这个问题，使用设计期ActiveX控件(Design-time ActiveX Control, DTC)和ASP组件是解决这一问题的两种不同方法。

不要将DTC与分布事件协同(Distributed Transaction Coordinator)相混淆。

首先，DTC是可视化的，在调用ASP页面之前，它关联于用于生成ASP页面运行期文本的创作工具。可以将其理解为类似于先进的Word和Excel宏。一旦它们完成运行并你保存了页面，ASP脚本仅包含代替控件的文本。不必担心可能造成的额外的下载，因为程序已经使用了DTC，而DTC是开发期控件，因此运行时不进行任何多余的下载。通常，不必为自己的项目编写DTC，而是使用开发工具提供的DTC。例如Visual InterDev就提供大量DTC供使用。

正如在前面章节看到的，ASP组件更为直接和特定，并且只在运行期内发生作用。它们能够添加基于运行期环境的代码。这些组件用C++或Visual Basic编写，并由IIS控制在服务器上加载和执行。在使用ASP时，将直接影响站点的整体性能。

本章将介绍用脚本语言编写的ASP组件。这种ASP组件是名为Windows脚本组件(Windows Script Component, WSC)的常用技术的一个特例。它使程序员能够仅仅使用一种脚本语言如VBScript或JScript编写几种类型的COM对象。如果正在使用Windows 2000或IE 5.0，那么计算机上已经安装了这些组件。但是，也可以在Windows NT环境下的IIS 4.0中或Windows 9x环境下的PWS中运行。

所以，这里将简单介绍WSC技术的主要特征，主要内容有：

- 使用脚本语言编写COM对象的技术基础。
- 怎样使用脚本语言编写COM对象。
- 怎样编写ASP脚本组件。
- ASP脚本组件同其他ASP组件的区别。

在讲述理论之后，举个例子：用一种有力而易于维护的方法封装HTML的表以格式化数据。

### 16.1 使用WSC的环境

如果要重现在这章中所要讲述的所有内容，你的系统需要安装相应的一些程序。首先，把浏览器指向微软脚本网站：<http://msdn.microsoft.com/scripting>并下载WSC包，这是一个自安装文件，包含以下内容：

- WSC的二进制驱动程序。
- 文档。
- 帮助编写组件的向导。

如果你已经安装了Windows 2000或IE 5.0，则系统已经具有其驱动程序。安装Windows

98或IE 4.01的系统也是这样的，但其提供的是语法上稍有不同的较老版本。只要从微软的网站下载其软件包则一切可迎刃而解。

WSC向导并不包含在任何版本的浏览器和操作系统中，需要从微软站点下载。

## 16.2 一个Windows脚本组件

下面从Windows脚本组件的定义开始：

一个Windows脚本组件就是一个使用脚本语言编写的 COM对象。

初看这个定义会感觉到很奇怪：真的可以用脚本语言编写 COM对象吗？下面的这些情况也说明了这种疑惑的合理性：

- COM对象是二进制的文件，而VBScript和JScript不生成二进制文件。
- COM对象具有一个众所周知的二进制布局，脚本语言如何实现这种布局？
- COM对象提供接口，如何使脚本语言也能这样做。
- COM对象需要注册和一个CLSID，脚本语言如何处理这些问题。

上面的这些疑问将在下面给出相应的详尽解答。

### 16.2.1 发展历史

最初，组件化Web的尝试是由1997年发布的IE4实现的。到目前为止，IE仍是唯一的支持动态超文本标记语言(DHTML)Scriptlet的浏览器(它是今天Web组件的原形)。DHTMLScriptlet主要指嵌入另一HTML的HTML。通过使用一个特定的命名约定，子页面可以向容器页面提供一个可编程的接口，并允许容器页面调用其方法、属性或激活事件。

使用DHTML Scriptlet同使用ActiveX控件并没有什么不同，但是作为组件，二者主要分别用于IE和Visual Basic。从效率上来讲，它们并不是真正的由脚本语言编写的可重复使用组件。此外，DHTML Scriptlet组件是为显示用户界面而设计的，而不是作为被请求页面的辅助函数。

从DHTML Scriptlet发展到Scriptlet

不久便产生了简单地称为Scriptlet的组件。它们便是今天WSC的祖先。Scriptlet是一种常规的COM对象，可以从任何感知COM的开发工具调用，如Delphi、PowerBuilder、MFC和Visual Basic。

DHTML Scriptlet同Scriptlet相比较，其本身并没有什么不同，而编写方式却大不相同。组件源代码的结构由HTML和脚本代码的混合体变为XML。

COM对象并不仅是代码。它提供标准的二进制布局。这意味着当编写COM对象时，至少需要考虑两方面信息，分别是实现的接口和注册数据。正如我们前面注意到的那样，使用脚本语言无法传递这种特殊的信息。

微软有两种选择，分别是：

- 重新选择一个特殊的命名约定以插入注册和接口信息。

或：

- 使用完全不同的语法编写组件。

微软选择了后者并使用XML定义组件的语法。

IE 4.01 Scriptlet同目前的WSC的不同点在于：后者使用XML模式存储对象的附加信息。

### 16.2.2 HelloWorld WSC

WSC文件是扩展名为.wsc的XML文件，以前是.sct。正如其他的 HelloWorld例子一样，WSC版的例子也是很简单：

```
<component>

<registration progid="HelloWorld.Component"/>
<public>
  <method name="Welcome"/>
</public>

<script language="VBScript">
Function Welcome
  MsgBox "Hello, World"
End Function
</script>

</component>
```

脚本组件中所有的信息都必须置入 `<component>`、`</component>` 标记对中。也可以使用 `<registration>`、`<public>` 和 `<script>` 标记。它们分别用 CLSID 或 ProgID、实现的接口及其实际功能来描述 COM 对象。

现在你应当对 WSC 文件有了一个初步的了解，下面将更为详尽地讲述如何构造 WSC。

## 16.3 WSC 的结构

首先，WSC 的整个结构可以分解为如下三层：

- 描述层：提供组件的元数据信息。
- 脚本层：包含所需的脚本函数。
- 运行期层：以二进制提供给客户一个熟悉的 COM 布局。

图16-1显示了WSC源代码文件的不同层。

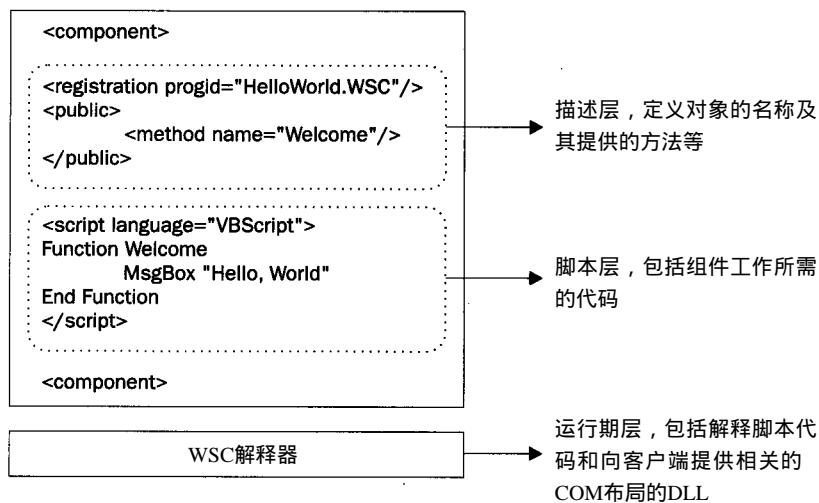


图16-1 WSC的源代码结构

### 16.3.1 描述层

在这一层中共有四种基本的标记可供使用，它们是：

- <registration>
- <public>
- <implements>
- <resource>

#### 1. <registration>

<registration>标记包含了向Windows注册表中添加组件时需要使用的信息。在这里，可以对组件进行较少的描述，如 ProgID和ClassID，也可以是任何基于注册或取消注册的可执行代码。

举个例子，一个新创建的组件会有一个类似下面的注册标记：

```
<registration
  description="The first HelloWorld WSC"           <!-- optional -->
  version="1.00"                                   <!-- optional -->
  progid="HelloWorld.Component"
  classid="{6432490f-0d48-48e3-bbe7-e2e773fc843e}"
>
</registration>
```

description和version属性是可选的，但是必须至少指定一个 progid和classid属性。否则，当试图注册组件时将产生错误。

当为单一类型的WSC创建COM对象时，必须有一个CLSID。没有CLSIDs的WSC无法工作，所以在注册过程中自动生成 CLSID。如果不指定 CLSID，特定的注册 COM对象的系统程序 (regsvr32.exe)会为其创建一个。但是要记住，只有最新的用于 IE 5.0、Windows 2000和WSC包的regsvr32.exe才会这样做。否则，注册过程将崩溃。

Progid用于在如VBScript的CreateObject或JScript的ActiveXObject这样的后绑定函数中创建对象实例。实际上，在脚本开发环境中，一个没有 ProgID的组件几乎没有任何作用。不定义ProgID，将无法用CreateObject或ActiveXObject来创建对象实例。只有Windows Script Host 2.0才允许仅使用CLSID来创建对象实例。

在注册和取消注册时执行代码

如果需要，组件具有在注册或取消注册时运行预定义代码的功能。通常，从 regsvr32.exe的视角来看，COM对象的注册是从COM对象DLL中调用一个函数。但是，每个WSC都通过服务器模块(scrobj.dll)注册。一旦调用，它就知道了正在工作的 WSC文件的名称，并且创建COM对象的所有注册项。

这个DLL还在<registration>标记中查找两个函数中的一个。根据组件是在注册还是在取消注册，它们分别是Register()和Unregister()：

```
<registration
  description="This is my component"           <!-- optional -->
  version="1.00"                               <!-- optional -->
  progid="MyComp.WSC"
  classid="{e8c35060-1879-11d3-b17c-00c0dfe39736}"
<script language="VBScript">
  Function Register
    MsgBox "Registering..." ' (e.g. generate a type-library on-the-fly)
```

```
End Function

Function Unregister
    MsgBox "Unregistering..."
End Function
</script>
</registration>
```

定义这些函数取决于是否需要。在注册组件前没有必要声明其中任何一个。当然，可以使用这两个函数去做更多的事情，而不仅仅是显示一个对话框。举个例子，可以动态生成组件类型库。

## 2. <public>

<public>标记定义了组件的公共编程接口。换句话说，定义了组件提供的并且在客户端可以调用的方法、属性和事件。其语法相当直接明了：

```
<public>
  <property name="MyProperty">
    <get/>
    <put/>
  </property>

  <method name="MyMethod">
    <parameter name="param1"/>
  </method>

  <event name="OnEvent">
    <parameter name="param1"/>
  </event>
</public>
```

其中，name属性定义了每个条目的公共名称。在上面的代码中，属性是通过变量MyProperty进行描述的，而方法体全部在一个名为MyMethod的函数中。二者都定义在WSC的<script>标记中。下面是更多的关于这方面的内容。

### (1) 内部名称

对于方法和属性同样也可以使用一个内部名称。这通过internalname属性来赋值。内部名称的基本用途是在WSC文件中引用方法或属性。

```
<public>
  <property name="MyProperty" internalname="m_MyProperty">
    <get/>
    <put/>
  </property>
  <method name="MyMethod" internalname="DoMyMethod">
    <parameter name="param1"/>
  </method>
</public>
```

在这种情况下，MyProperty的属性通过变量对m\_MyProperty来实现。同时，当客户调用MyMethod方法时，实际调用的是DoMyMethod函数。

### (2) 指定参数

一个方法根据需要可以有任意多个参数。这里，name属性指示了形参名称。同样，事件也可以有任意个参数，尽管一些老的文件没有提到这一点。

### (3) 只读和只写属性

客户端不能通过浏览器打开 WSC中使用的属性，这些属性也不能读写。在大多情况下，会发现组件同客户的交互需要多一点控制。假设组件显示当前会话个数，这个值应是只读的。如果客户能修改他，则很快就会出现错误。

每个<property>标记有两个子标记：<get/>和<put/>。两者分别定义了读和写属性值的方式。当需要读写属性内容时，执行二个过程中相应的一个。它们都有一个默认的名字，分别是get\_和put\_，加上属性的公共名称，但是二者都可通过使用前面讲到的 internalname属性进行更改。如果省略<get>标记，可自动使此属性只写。同理，省略<put>可以使属性只读。

#### (4) 激发事件

在WSC中激发一个事件必须调用一个 WSC的全局函数fireEvent。这可以触发一个由其第一个参数指定的事件，另外，还可以增加任意个参数。

```
fireEvent "OnEvent", dataToPass
```

#### 3. <implements>

如果仅需要WSC提供方法、属性和事件，则不需考虑 <implements>标记。这个标记主要用于定义你的WSC支持的COM接口。缺省情况下，WSC自动支持自动化(Automation)和事件处理接口。更准确地说，当插入<public>标记时，将自动要求支持这些接口。

一个用脚本语言编写的 COM对象并不能实现所有可能的 COM接口。尽管这在 C++和Delphi中可以非常轻易地实现，在 Visual Basic中将会稍微复杂一些，而用脚本代码几乎是无法实现的。为了围绕这个方面展开工作，WSC依赖于名为“接口处理器”(interface handlers, IH)的专用二进制模块。接口处理器由前面提及的 scrobj.dll管理。

#### 接口处理器

接口处理器(IH)是一种对任何附加的非自动化 COM接口提供支持的二进制模块。其结构如图16-2所示。

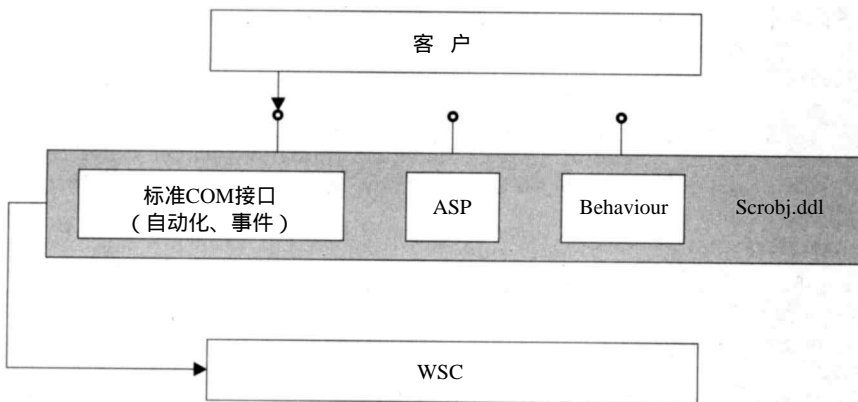


图16-2 接口处理器结构图

IH主要是作为WSC脚本代码和客户应用程序之间的一种中介，向客户提供给定接口的标准的COM布局。当客户调用接口的方法时，IH检索并执行脚本代码以实现其功能。

一旦声明<public>标记，scrobj.dll将自动为自动化和事件加载IH。如果需要支持附加的接口，可以在<implements>中显式请求。也就是说，<implements>标记是一种WSC与特定的处理器之间的联系方式。其语法如下：

```
<implements type="handler" id="name"/>
```

type属性定义IH的名称,而id是处理器自身实际使用的ID。在大多情况下可以省略id属性。在本章稍后将讨论WSC结构的内部细节。

#### 4. <resource>

使用这个标记可以隔离 WSC文件代码中所有是常量并且不想在脚本中进行硬编码的元素(字符串和数字)。每种资源通过ID定义:

```
<resource ID="hello">Hello, world!</resource>
<resource ID="version">1.02</resource>
```

可以使用ID在WSC文件代码中检索信息。ID必须传递到全局函数 getResource()以获得存储的数据:

```
<component>
  <registration progid="HelloWorld.WSC"/>
  <comment>
    Place here as many comments as you like.
  </comment>

  <resource id="hello">Hello, world!</resource>
  <public>
    <method name="Welcome"/>
  </public>
  <script language="VBScript">
    Function Welcome
      MsgBox getResource("hello")
    End Function
  </script>
</component>
```

上面的代码显示了另外一个标记: <comment>。顾名思义,置于这个标记中的所有文本将被分析器忽略,可以用做源代码的注释。

<resource>标记的典型用途是定义脚本中的常量。尽管 <resource>更为标准且语言上更为中性一些,但从功能上讲,使用 <resource>标记同在脚本中声明一组常量并没有区别。

### 16.3.2 脚本层

所有与WSC方法、属性以及辅助函数相关的函数都必须包含于 <script>标记中。其语法与HTML的<script>标记十分相似,但不同的是前者仅需要 language属性来进行修饰。

只要安装合适的解释器,就可以在脚本组件中使用任何语言。微软仅仅提供了基于VBScript和JScript的两个解释器。但是第三方厂商已经开发了用于 Perl和其他语言的 Windows 脚本兼容解释器。

#### XML适应性

正如前面提到的,一个WSC文件基本上就是一个XML文件,遵循XML 1.0规范。到此为止,实现它仍有两件工作要做。第一件是在文件顶部加入下面一行代码以声明它为XML。

```
<?xml version="1.0"?>
```

但是,当这样做了以后,必须将<script>标签中的所有内容包含进来。为了防止XML解释器对特殊字符的混淆,例如“<”或“&”,可使用CDATA分界符。

```
<script language="VBScript">
<![CDATA[
```



```
Function Welcome  
    MsgBox getResource("hello")  
End Function  
]]>  
</script>
```

### 16.3.3 运行期层

为了创建一个 WSC实例，通常需要使用一个类似 VBScript的CreateObject的函数并以 ProgID作为其第一个参数(另外组件所安装的服务器的名称是可选的第二个参数)。一旦调用，VBScript引擎试图定位实现组件的DLL的名称，进行下列步骤：

- 在注册表中搜寻 ProgID。
- 从ProgID获得组件的CLSID。
- 在注册表中搜寻 CLSID。
- 读取可执行文件的名称。

对于WSC而言，其服务器模块总是位于 Windows目录下的scrobj.dll文件。VBScript创建对象实例，scrobj.dll将接受代表此对象的 CLSID。现在，scrobj.dll的核心代码将使用这个 CLSID在注册表中搜寻名为 ScriptletURL的WSC键。这个键指向提供公共方法的运行期代码的 WSC文件。

WSC的运行期 DLL为一个虚拟对象创建一个实例，该实例实现 WSC定义的接口并将其返回给调用者。之后，当 VBScript代码执行一个方法时，将调用这个虚拟对象。由这个虚拟对象从WSC源代码中提取正确的代码段，并交由 VBScript或JScript解释器执行。

改变后不用再注册

当向外界提供COM的接口后，不会因为添加或删除其属性和方法而发生改变，也不必在脚本代码中对发生改变的组件进行重新注册。同理，也不必在服务器脚本组件发生改变后停止并重新启动服务器。对于代码执行，所有需要处理的事情都发生在运行期。

## 16.4 可用的接口

前面曾提到使用脚本语言编写 COM对象存在着种种问题。下面，将通过了解 WSC的底层运行进行期机制找到以上问题的答案。客户端程序将始终需要获得标准的 COM接口，但在COM的标准接口背后，运行期引擎事实上是运行从分离的 WSC文件中读取的脚本代码。

很清楚，这样做的结果就是必须为通过脚本代码提供的接口提供一个二进制的中介。这个中介就是前面所说的接口处理器。换句话说，可以通过脚本代码实现一些 COM接口，同时必须相应的处理器。

微软给出的有关这方面的技术指导，指出了仅有的几个可用的接口处理器：

- 自动化处理器。
- 事件处理器。
- ASP处理器。
- IE 5.0行为处理器。

所有这些处理器都在scrobj.dll内实现，不需其他文件支持。如果使用了 <public>标记，前面两个处理器会被自动激活。后两个则需要定义一个 <implements>标记。



```
<implements type="ASP" />
<implements type="Behavior" />
```

在同一个WSC文件中同时使用<public>和<implements>标记不会出现问题。自动化处理器和事件处理器可以同ASP处理器或行为处理器很好地共存。但是，应当避免将ASP组件同行为处理器置于同一个WSC组件中，因为二者将会相互冲突。可以编写两个分离的WSC文件，或者创建一个包(Package)。

#### 包

可以在WSC XML模式的根上找到<package>标记。<component>标记事实上是<package>标记的一个子标记。<package>标记作为单个文件中的多组件的“容器”。唯一的附加条件是，每个包中的组件必须指定一个唯一的ID号。例如：

```
<?xml version="1.0"?>
<package>

  <component id="One">
    <registration progid="MyComponent.One" />
    <implements type="ASP" />
    <public>
      <method name="DoThis" />
    </public>
    <script language="VBScript">
      <![CDATA[
function DoThis()
  n = 1
end function
]]>
    </script>
  </component>

  <component id="Two">
    <registration progid="MyComponent.Two" />
    <implements type="Behavior" />
    <public>
      <method name="DoSomething"/>
    </public>

    <script language="VBScript">
      <![CDATA[
function DoSomething()
  DoSomething = "Temporary Value"
end function
]]>
    </script>
  </component>

</package>
```

## 16.5 编写一个ASP脚本组件

实际上，使用脚本语言编写的ASP组件只是Windows脚本组件(WSC)的一个特定类型。唯一的不同是它们还具有与ASP内建对象的接口，如Response、Request、Session等等。

当然，只有在IIS或PWS地址空间中创建的对象才是可用的。为了在VBScript代码中安全地调用这些对象，需要对执行WSC<script>代码的环境添加对它们的引用。ASP接口处理器正

是用来完成这项工作的。

只有在特定的WSC文件中通过<implements>标记请求ASP支持时，处理器才被激活。在最后一节中将给出其示例。

16.5.1 AspTable组件

现在，看看如何编写一个不同寻常的组件 AspTable。之所以把它叫作 AspTable组件是因为它只是对HTML<table>标签的一个包装。

通常，使用表格显示已经或尚未进行附加处理的数据库记录。例如，姓和名常被合并，并且放置在数据库的同一栏中。

AspTable组件有一些有用的特征：

- 为表添加列，从一个ADO记录集中获取信息。
- 定义一个动态表达式来合并几个字段，并通过一个链接或动作将它们关联起来。
- 为某些栏指定风格和装饰性文本(如标题、脚注等)。

在此组件中定义的公共方法如表 16-1所示。

表16-1 AspTable组件定义的公共方法及说明

方 法	说 明
SetRs( <i>adoRS</i> )	用一个ADO记录集填充表
AddColumn( <i>heading, expr, style</i> )	在表中增加一列，指定栏标题、CSS风格和每行的计算表达式
GetText()	在页面中产生并插入用于表的HTML代码

按照组件的工作方式，这三个方法必须按一定的顺序进行调用才能正常工作。所以，在从记录集中生成HTML表格之前，需要先完成以下工作：使用 SetRs方法使ADO记录集作为数据源工作；使用 AddColumn方法添加任意多个所需要的列；最后调用组件的 Response.Write方法，将由GetText方法生成的HTML文本写到输出流。

AspTable组件还定义了如表 16-2所示的一些公共属性。

表16-2 AspTable组件定义的公共属性及说明

属 性	说 明
TableStyle	用于整个表的CCS风格的类名
HeaderStyle	用于表中所有列标题的CCS风格的类名
Title	显示在表顶部的HTML文本
Footnote	显示在表底部的HTML文本
ColumnCount	返回当前的栏数，只读

这些组件使你能够在表中显示信息，如图 16-3所示。

比起编写HTML来生成表格，使用这个组件有不可比拟的便捷性，而且使用组件更利于持久性和易读性。产生图 16-3的程序代码如下：

```
Set objTBL = CreateObject("AspTable.WSC")
objTBL.SetRS rs)

objTBL.Title = "<H2>List of contacts:</H2>"
objTBL.Footnote = "<HR>%&_RECCOUNT_% contact(s).<BR>"
objTBL.TableStyle = "globTab"
objTBL.HeaderStyle = "header"
```

```
objTBL.AddColumn "#", "N. %&_RECNO%", "num"
objTBL.AddColumn "Name", "%&FIRSTNAME% <b>%&LASTNAME%</B>", ""
objTBL.AddColumn "Company", "COMPANY", "company"
objTBL.AddColumn "Email", "EMAILADDRESS", ""
```

```
objTBL.GetText()
```



图16-3 运行AspTable组件时的界面1

上面的代码比原始的HTML代码更好。下面来分析组件的代码。

下面代码显示了AspTable对象的元数据。

```
<?XML version="1.0"?>
<component>
  <registration
    description="Formats a recordset into a TABLE"
    progid="AspTable.WSC"
    version="1.00"
    classid="{33351B20-D7B0-11d2-B7C8-00C0DFE39736}" />

  <public>
    <method name="GetText" />
    <method name="SetRS">
      <parameter name="rs" />
    </method>
    <method name="AddColumn">
      <parameter name="displayName" />
      <parameter name="colFormat" />
      <parameter name="className" />
    </method>
    <property name="Title" internalname="m_Title" />
    <property name="Footnote" internalname="m_Footnote" />
    <property name="TableStyle" internalname="m_TableStyle" />
    <property name="HeaderStyle" internalname="m_HeaderStyle" />
    <property name="ColumnCount">
      <get internalname="DoGetNumOfCols" />
    </property>
  </public>

  <implements type="ASP" />
```

请注意，几乎所有的属性都是使用内部名称（事实上是个人喜好问题），并没有使用由get和put方法组成的过滤器。仅有ColumnCount属性具有的一个<get>标记，因为需要它是只读的。

### 1. AspTable 的脚本代码

AspTable 对象的核心是 GetText 方法，这个方法可以从其他方法 (如列信息、记录集、风格等) 中采集数据，并利用 <TABLE> 标记将其组织起来。在深入研究这个方法之前，先来看看一些初级的方法，其中包括前面提到的 SetRs 和 AddColumn 方法。

```
<script language="VBScript">
<![CDATA[

'/*-----
'// Globals
'-----*/
Const FLDNAME_MUST_HAVE_DELIM = 1
Const FLDNAME_CAN_HAVE_DELIM = 0

Dim m_RS
Dim m_Dispcols()
Dim m_NameCols()
Dim m_ClsCols()
Dim m_colCount
m_colCount=0
m_RecNo=0

'/*-----
'// ColumnCount property
'-----*/
Function DoGetNumOfCols()
    DoGetNumOfCols = m_colCount
End Function

'/*-----
'// AddColumn method
'-----*/
Function AddColumn (displayName, colFormat, className)
    m_colCount = m_colCount + 1
    ReDim Preserve m_Dispcols(m_colCount)
    ReDim Preserve m_NameCols(m_colCount)
    ReDim Preserve m_ClsCols(m_colCount)

    ' Store the info
    m_Dispcols(m_colCount) = displayName
    m_NameCols(m_colCount) = colFormat
    m_ClsCols(m_colCount) = className
End Function

'/*-----
'// SetRS method
'-----*/
Function SetRS (rs)
    Set m_RS = rs
End Function
```

请注意任何 ADO 记录集都是可用的。需要记住的是，由 DBMS (如 SQL Server) 创建的记录集和由定制的 OLE DB 数据提供者生成的记录集以及其他任何不依赖于数据源而创建的记录集之间没有任何区别。这意味着可以通过使用组件来修改任何数据。

### 2. 创建表

GetText功能执行一个非常简单的算法，通过链接记录集中的行创建一个长字符串。这种方法和组件的真正强大之处是，可计算表达式而不是字段的内容。换句话说，不仅可以记录集字段值赋给列，还可以赋给列一个字符串表达式，例如链接两个或更多的字段。

这种组件的版本不接受表达式中的代码，但可接受一些宏，它与 C/C++中的sprintf相似。在VBScript中，它可被看成特殊形式的Replace。

```

'/*-----
'// GetText method
'-----*/
Function GetText()
    Dim strText

    ' Set the Title
    strText = ""
    If Len(m_Title) Then
        strText = strText + ExpandString(m_Title, FLDNAME_MUST_HAVE_DELIM) + vbCrLf
    End If

    ' Add the <TABLE> header
    strText = strText + "<TABLE class=" + m_TableStyle + ">" + vbCrLf
    strText = strText + "<TR>" + vbCrLf
    For i = 1 to m_colCount
        Dim strName
        strText = strText + " <TH class=" + m_HeaderStyle + ">"
        strText = strText + m_DisCols(i) + "</TH>" + vbCrLf
    Next
    strText = strText + "</TR>" + vbCrLf

    ' Walk the recordset and add the <TABLE> items
    While Not m_RS.EOF
        m_RecNo = m_RecNo + 1
        strRowText = ""

        For i = 1 to m_colCount
            Dim strClass

            ' Set the class string
            If VarType(m_ClsCols(i)) <> vbEmpty Then
                strClass = "class=" + m_ClsCols(i)
            End If

            ' Retrieve the column content
            strColumn = ExpandString(m_NameCols(i), FLDNAME_CAN_HAVE_DELIM)

            ' Format the row string
            strRowText = strRowText + "<TD " + strClass + ">" + vbCrLf
            strRowText = strRowText + strColumn + vbCrLf
            strRowText = strRowText + "</TD>" + vbCrLf
        Next

        ' Add the row to the final string to return
        strText = strText + "<TR>" + strRowText + "</TR>"

        ' Next record
        m_RS.MoveNext
    Wend

    ' Close the table
    strText = strText + "</TABLE>"

```

```

' Set the Footnote
If Len(m_Footnote) Then
    strText=strText + ExpandString(m_Footnote, FLDNAME_MUST_HAVE_DELIM) + vbCrLf
End If

' Closes and writes the page
GetText = strText
Response.Write strText
End Function

```

辅助函数ExpandString给所用的列字符串表达式赋值。它考虑字段名并且扩展开那些封闭在“%”内的前缀为“&”的字符串：

```
%&FIELDNAME%
```

除字段名外，另外两个宏可以使用，即 \_RECON\_和\_RECCOUNT\_。前者返回当前的记录号。后者返回显示的记录的总数。

```

'/*-----
'// ExpandString helper function
'// A field name is enclosed in %&fieldname%
'// A single string 'fieldname' is also seen as field name if
'// mustHaveDelim is False/0
'-----*/
Function ExpandString(text, mustHaveDelim)
    SEP_ID = "%"
    FLD_ID = "&"
    FLDNAME_ID = "%&"

    newText = text

    ' Splits the string into components by %
    aTokens = split(text, SEP_ID)

    ' Single strings default to a field name. Due to this code
    ' passing 'EmployeeID' and "%&EmployeeID% is the same
    If mustHaveDelim <> 1 Then
        If UBound(aTokens)=0 Then
            newText = FLDNAME_ID & newText & SEP_ID
            aTokens(0) = FLD_ID & aTokens(0)
        End If
    End If

    ' Extracts the field names from the token array
    aFields = Filter(aTokens, FLD_ID)

    ' Process each field and replaces it in the string using
    ' the value of the current record in the recordset
    For Each field In aFields

        ' Get the real name of the field
        realName = Right(field, Len(field)-1)

        ' Get the name to replace in the string
        findName = SEP_ID & field & SEP_ID

        ' Is it a special field name such as _RECCOUNT_ or _RECNO_?
        ' If yes handle them properly, otherwise get the recordset value
        Select Case realName
            Case "_RECCOUNT_"
                rsFieldValue = m_RS.RecordCount
            Case "_RECNO_"
                rsFieldValue = m_RecNo

```

```

Case else
    rsFieldValue = m_RS(realName)
    If VarType(rsFieldValue) = vbNull Then
        rsFieldValue = " "
    End If
End Select

' Replace the strings
newText = Replace(newText, findName, rsFieldValue)
Next

ExpandString = newText
End Function

```

asptable.WSC组件和代码中的其余支持资料可以从 Wrox 的Web站点下载。

现在, 还需做的是对这个组件进行注册。可右击 Windows Explorer中WSC文件, 选择快捷菜单中的Register。然后就可以从ASP页中调用这个组件了。如果注册没有问题, 就可以用这个组件来开发页面。

### 16.5.2 使用AspTable对象

下列页面用AspTable对象显示contacts数据库中的一些名称。可从本书的下载代码中找到这个样式表。

```

<HTML>
<HEAD>
<LINK REL="stylesheet" TYPE="text/css" HREF="styles.css"/>
</HEAD>
<BODY>

<H1>Testing the AspTable WSC component</H1>
<HR>

<%
    Dim rs
    Dim sql
    Dim objTBL

```

这里所用的数据库为本书下载代码的一部分, 还应创建一个称为“Contacts”的系统DSN来指向它。

```

' Get the recordset
sql = "select * from Contacts order by lastname"
Set rs = CreateObject("ADODB.Recordset")
rs.CursorLocation = 3 ' adUseClient
rs.Open sql, "DSN=Contacts"
rs.ActiveConnection = Nothing

Set objTBL = CreateObject("AspTable.WSC")
objTBL.SetRS (rs)

objTBL.Title = "<H2>List of contacts:</H2>"
objTBL.Footnote = "<HR>%&_RECCOUNT_% contact(s).<BR>"
objTBL.TableStyle = "globTab"
objTBL.HeaderStyle = "header"

objTBL.AddColumn "#", "N. %&_RECNO%", "num"
objTBL.AddColumn "Name", "%&FIRSTNAME% <B>%&LASTNAME%</B>", ""

```



```

objTBL.AddColumn "Company", "COMPANY", "company"
objTBL.AddColumn "Email", "EMAILADDRESS", ""

objTBL.GetText()
%>

</BODY>
</HTML>

```

通过典型的ADO调用得到记录集，然后在表中增加四列。如果AddColumn表达式参数包含一个单词，它被认为是字段名。这意味，email字段可通过EMAILADDRESS或%&EMAILADDRESS%显示。结果如图16-4所示。

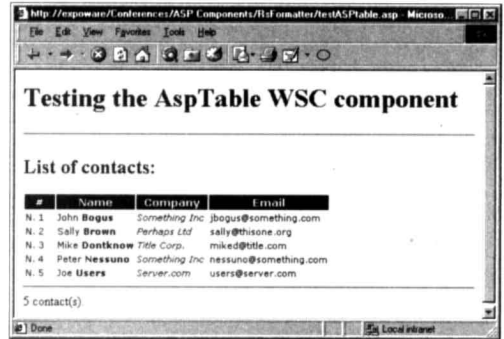


图16-4 运行AspTable组件时的界面2

第二列链接了每个联系人的姓和名，使用了不同的格式化风格，字符表达式如下所示。

```
%&FIRSTNAME% <b>%&LASTNAME%</b>
```

要改变列的表示方式或列的数目，可做如下改变：

```

Set objTBL = CreateObject("AspTable.WSC")
objTBL.SetRS (rs)

objTBL.Title = "<H2>List of contacts:</H2>"
objTBL.Footnote = "<HR>%&_RECCOUNT_% contact(s).<BR>"
objTBL.TableStyle = "globTab"
objTBL.HeaderStyle = "header"

objTBL.AddColumn "#", "N. %&_RECNO_%", "num"
objTBL.AddColumn "Name", "%&FIRSTNAME% <B>%&LASTNAME%</B>", ""
objTBL.AddColumn "Keyword", "LASTNAME", ""
objTBL.AddColumn "Company", "COMPANY", "company"
str = "<A href=mailto:%&EMAILADDRESS% >%&EMAILADDRESS%</A>"
objTBL.AddColumn "Email", str, ""

objTBL.GetText()

```

代码运行的结果如图16-5所示。

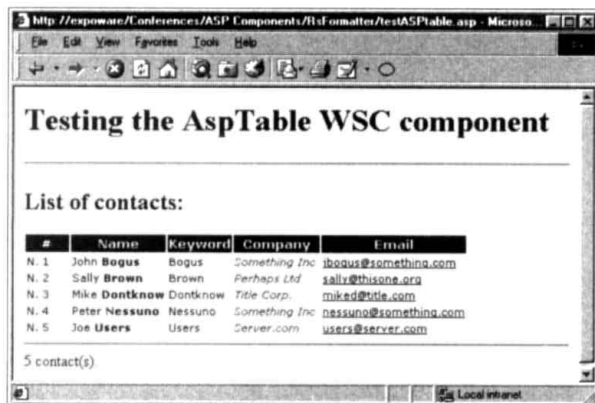


图16-5 运行AspTable组件时的界面3

在上面的表中增加了称为Keyword的一列，并且已将e-mail地址转换为超级链接。

## 16.6 ASP脚本组件的特点

我们应该从正反两方面考虑在工作中使用 ASP脚本组件的必要性。与编译的组件比较，ASP脚本组件有一个明显的特点：WSC脚本文件可在任何时候进行编辑，在下次执行时，组件中的变化可自动反映出来。不需要重新编译，不需重新启动服务器释放现有组件的引用，也不需要重新注册。并且，如果组件工作状态良好，可选用合适的语言，用已证明可靠的代码和/或算法创建一个编译版本，这同样适用于任何可转换组件。

ASP脚本组件可适应COM+。事实上，与其他ASP内置对象一样，这种组件可访问ObjectContext对象。因此，如果需要，可在任何时候使用 SetAbort和SetComplete。

脚本组件可用标准的COM接口对现有的业务对象(组件)进行包装，即使它们是用Perl编写的也可以进行包装。在包装过程中，不必知道如何用Perl编写，也不必了解相关知识。COM的语言中性特征使得互操作性更为容易。

代码的组件化不仅改善了Web网站的性能，而且还可通过重新使用通用代码块帮助设计和维护网站。

启用WSC的页面优于利用服务器端包含(Server-Side Include, SSI)文件的页面。虽然它们做的事情相同，但ASP脚本组件可根据运行期条件或输入参数动态地创建所需要的对象。SSI不允许动态选择，仅允许导入名称在页面的源代码中硬编码的文件。

当然，ASP脚本组件也存在一些不足：

- 解释组件不具备编译组件的性能，因此ASP脚本组件的性能不一定是最优的。但性能不是使用它们的理由。
- ASP脚本组件不能访问完整的Win32 API，完成一个系统级的任务需要特定的COM自动化对象。而且，脚本语言与C++相比，甚至与VBA相比还不是一种非常成熟的语言。

### 16.6.1 选择正确的工具

下面介绍创建ASP组件时选择正确工具的快捷方法。

要开发关键的和真正需要服务器性能最高的实时组件，应使用 Visual C++。

因为服务器端组件大多执行一些数据访问类的任务，所以考虑使用 C++和原始的OLE DB调用以获得最大的性能。Visual C++、ALT模板和OLE DB接口的组合或许对性能的提高是最有效的。本书的下一章将对用C++开发组件做更多的描述。

Visual Basic也是开发组件的强大工具。如果开发的组件不是关键性的，可以用 Visual Basic。除了同样具有编译代码、完全 Win32访问和组件化外，Visual Basic比Visual C++容易。

WSC 可使页面的原始代码组件化，通过COM创建脚本对象将增加一些额外工作量，但灵活性和可重用性较好，并且可减少维护工作量。

### 16.6.2 脚本组件与VBScript类的比较

如果打算在ASP组件中使用VBScript 5.0，可以考虑使用VBScript 类来替代脚本组件。

创建VBScript类比创建Windows脚本组件用的时间少一些，因为与 CreateObject不同，New不需要通过COM引擎和注册表对装载的文件定位。两者编程能力几乎相同，但使用 WSC

可利用不同脚本语言的优点。

## 16.7 小结

本章分析了 Windows 脚本组件 (WSC) 技术，ASP 脚本组件是它的特例。用脚本语言编写 COM 对象并不是难以实现的，事实上，这具有许多优点。但是本章内容并不是对每种情况都适合，所以不能解决实际碰到的所有问题。

本章内容重新归纳如下：

- WSC 的结构和接口处理器的作用。
- 页面脚本代码中 ASP 内置对象的映射。
- 客户和 ASP 脚本组件的交互。
- 用于 ASP 页面的一个完整的 WSC 例子。
- ASP 脚本对象的优缺点。

下一章将介绍如何使用 Visual C++ 编写功能强大的组件。