

## 第18章 对C++组件的进一步讨论

在前一章中，我们学习了用 C++ 创建 COM 组件的基本方法，用这些概念创建的组件对于在内存中处理数据是非常有用的。但是，要制作一个功能强大的服务器组件，需要使用微软平台提供的其他服务。特别是，创建的组件应能提供对数据的访问、与 COM+ 进行接口以及 ASP 交互的功能。本章主要内容有以下几个方面：

- 使用 ASP 内置接口。
- 与 COM+ 进行接口。
- 通过 C++ 使用 ADO。
- 使用 OLE DB 消费者模板。

### 18.1 与 ASP 进行接口

前一章创建的组件没有考虑到在 ASP 中的使用，如果想创建在任何环境中使用的一般组件这是有效的。然而，如果 ASP 页能访问信息，组件也能访问的话，这个组件在 Web 应用程序中将会十分有用。如果能够完成像 ASP 页一样的基于 Web 的交互功能，则这个组件将更有用。

COM 的世界十分精彩，可用 C++ 组件做任何事情。就像在 ASP 中可以有各种各样可用的对象一样，可从服务器对象访问这些组件。在 ASP 中所做的工作大部分可通过下列内置对象实现：Request、Response、Session、Application 和 Server。指向任一对象的指针可通过 ScriptingContext 获得。因此，关键是得到指向 ScriptingContext 的指针，然后寻找相应的内置对象。

与多数编程工作不同，程序的代码是由 Visual C++ AppWizard 产生的。我们要使第 17 章中创建的组件“启用 ASP”，这时将有两个选择：一种是在本章中创建组件，拷贝并粘贴合适的代码到前面的组件中；另一种从一个新的组件开始，并加入在前一章中编写的代码。我们选择后者。

使用不同的向导选项创建组件的框架，然后通过 Windiff 观察附加代码的生成，这将有助于理解。Windiff 是 Visual C++ 附带的程序。

使用在上一章创建 ASPComponents Visual C++ 项目，插入一个新的 ATL 对象。创建一个 ActiveX Server 组件而不是创建 Simple Object，命名为 CTableStorage2，如图 18-1 所示。

Simple Object 向导与 ActiveX Server 向导的不同在于后者包含 ASP 属性页。在 ASP 属性页中，仅选择 Response 复选框。在我们的例子中，仅使用 Response 对象，但是如果需要再添加其他内置对象也非常容易，如图 18-2 所示。

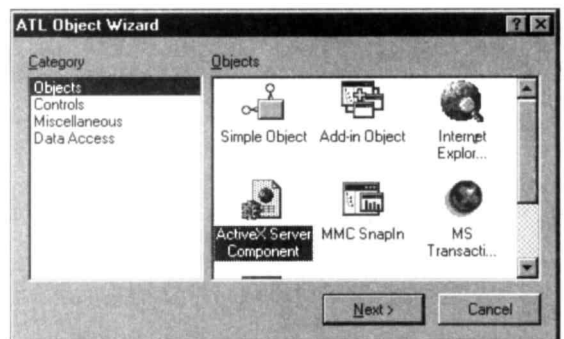


图18-1 创建ActiveX Server组件

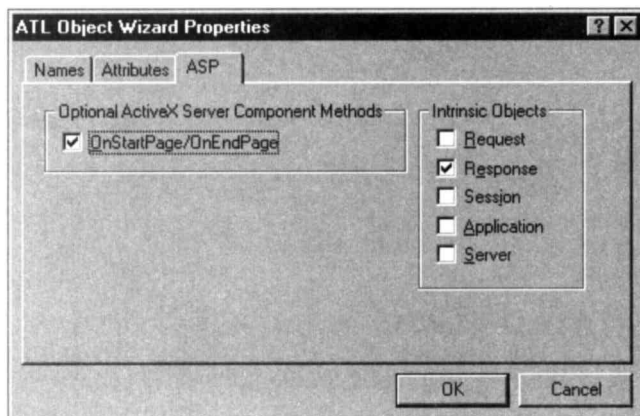


图18-2 选择ASP内置对象

这表现了Visual C++的灵活性和高效性，仅选择所需要的特性，可减少组件的代码。当然，与在ASP中不同，特性不仅仅是编程环境的一部分，增加特性就要增加代码。

不选择OnStartPage/OnEndPage时，所有内置对象均无效。必须调用OnStartPage才能得到内置对象。如果选择内置对象，将会自动产生OnStartPage和OnEndPage。这种相关性必须引起开发人员的注意。

我们来看一下选用ActiveX Server组件后产生的代码，如果已有一个启用ASP的组件，必须人工插入这段代码。

首先，文件TableStorage2.h中有一个新的头文件，它包含使用ScriptingContext和ASP内置对象所需的常数和定义。

```
#include <asptlb.h>           // Active Server Pages Definitions
```

注意，这里使用ScriptingContext而不是使用较新的ObjectContext，ATL向导产生使用ScriptingContext的代码，一般情况下不必进行修改。

你将看到增加了OnStartPage和OnEndPage方法。有了这两个方法，就可以在ASP页面创建这个组件时，或在ASP页面用过它之后做一些工作。这些方法将在以后讨论。

```
public:
    //Active Server Pages Methods
    STDMETHOD(OnStartPage)(IUnknown* IUnk);
    STDMETHOD(OnEndPage)();
```

在类里面增加了一些成员变量。其中一个是Response对象的指针。如果选择其他的ASP内置对象，也要为它们声明指针。指向对象的指针声明后，这个指针实际上还没有指向这个对象。

```
private:
    CComPtr<IResponse> m_piResponse;           //Response Object
    BOOL m_bOnStartPageCalled;                 //OnStartPage successful?
```

另一个成员变量是一个布尔变量，表示调用OnStartPage是否成功，后面你将看到其重要性。在构造器中这个变量被初始化为FALSE(在对象首次被创建时，会调用构造器)。在C++中构造器和析构器起的作用分别与VB中的Class\_Initialize和Class\_Terminate方法相同。

```
CTableStorage2()
{
    m_bOnStartPageCalled = FALSE;
}
```

这是头文件的全部变化。我们再来看一下 AppWizard 为 TableStorage2.cpp 生成的代码。这个代码实现了 OnStartPage 和 OnEndPage 方法。如果转换上一章中的组件，在 ASPComponents.cpp 中必须有一行是 #include TableStorage2.h。

OnStartPage 的主要任务是得到 ScriptingContext 和指向 ASP 内置对象的指针。在这个例子中只有得到 Response 对象的代码。再强调一次，如果选择了其他的 ASP 内置对象，就得到指向它们的指针。一旦成功获取了指向 ASP 内置对象的指针，m\_bOnStartPageCalled 将置为 TRUE。可用这个变量对使用 ASP 内置对象是否安全进行检查：

```
STDMETHODIMP CTableStorage2::OnStartPage(IUnknown* pUnk)
{
    if(!pUnk)
        return E_POINTER;

    CComPtr<IScriptingContext> spContext;
    HRESULT hr;

    // Get the IScriptingContext Interface
    hr = pUnk->QueryInterface(IID_IScriptingContext, (void **)&spContext);
    if(FAILED(hr))
        return hr;

    // Get Response Object Pointer
    hr = spContext->get_Response(&m_piResponse);
    if(FAILED(hr))
    {
        return hr;
    }

    m_bOnStartPageCalled = TRUE;
    return S_OK;
}
```

ASP 页处理完后，调用 OnEndPage 方法，这时 ASP 内置对象的指针没有意义，因此释放接口，并且 m\_bOnStartPageCalled 置为 FALSE：

```
STDMETHODIMP CTableStorage2::OnEndPage()
{
    m_bOnStartPageCalled = FALSE;
    // Release all interfaces
    m_piResponse.Release();

    return S_OK;
}
```

如果在 ASP 页中对象有效期的开头和结尾处要进行一些处理的话，可在 OnStartPage 和 OnEndPage 内编写程序。

简单的组件与 ActiveX Server 组件不同。OnStartPage 和 OnEndPage 方法通过 ActiveX 自动化提供，所以在组件的外部可以调用。要这样做必须在接口定义语言 (IDL) 文件中对此进行说明。在 ASPComponents.idl 中有这两条语句：

```
interface ITableStorage2 : IDispatch
{
    //Standard Server Side Component Methods
    HRESULT OnStartPage([in] IUnknown* piUnk);
    HRESULT OnEndPage();
};
```

编译组件时，MIDL编译器使用IDL文件创建一个类型库和调度代码。

现在已经可以在组件中使用 Response对象了。先添加一个方法来使用它。在前一章中我们编写了一个ASP程序，它用存储在组件中的数据生成HTML。如果由组件本身产生HTML表，组件会更强大。这一修改使得性能更为优越，并封装了绘制逻辑。

封装具有许多好处。对于生成表，所有绘制代码集中到一个地方，任何 ASP页可调用包含这段代码的组件，并可以运行它。很显然这比在页和页之间拷贝同样的代码或编写新的代码简单。开发人员可主要关心将像数据存取这样的高层次问题，而不是去寻找遍历数据并在表中显示的方式。调用一个组件和运行预编译代码比分析一系列复杂的 HTML命令快得多。

最后需要说明的是即使绘制编码需要有微小的修改，组件也要重新编译。开发组件时应仔细考虑程序员在生成表时需要设置的参数。用户可修改的特征，如边界尺寸、表格单元的背景色，可作为参数传给接口方法，而不必进行硬编码。但是如果组件接口改变，使用这个接口的ASP页必须进行修改。

还有一点需要说明，必须在 ITableStorage2中增加与ITableStorage相同的方法，例如 ParseCSV、GetField和 GetColumnName等。详细内容可参阅第17章。

在ITableStorage2中增加一个新方法 OutputTable，按照图 18-3所示的对话框进行配置。

虽然增加其他表格属性如颜色、单元宽度等也很简单，但为了说明方便，仅允许改变边界尺寸。OutputTable的代码如下：

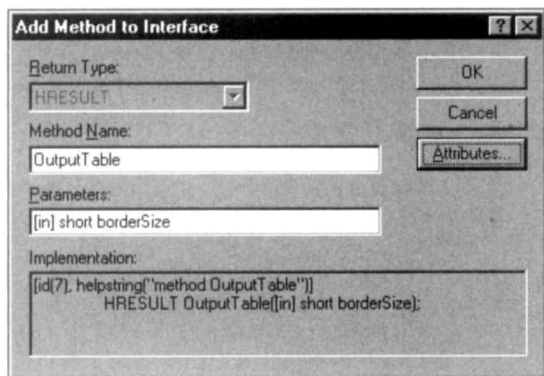


图18-3 增加新方法

```
STDMETHODIMP CTableStorage2::OutputTable( short borderSize )
{
    HRESULT      hResult = E_FAIL;
    wstring      tempString;
    char         tableString[32];

    if ( (m_bOnStartPageCalled == TRUE) && (m_piResponse != NULL) )
    {
        sprintf( tableString, "<TABLE border=%d><TR>", borderSize );
        m_piResponse->Write( CComVariant( tableString ) );
    }
}
```

首先确认m\_bOnStartPageCalled为TRUE并且，m\_piResponse确实包含一个指针。如果检查通过，就可以使用Response对象。

Write方法希望字符串为variant，在C++中使用variant必须指定一种variant类型。最简单的方法是将字符串作为字符类型，并用 CComVariant类构造一个variant。下一步输出表列：

```
// Output the tables headers.
COLUMN_INDEX_MAP::iterator  mapIter = m_columnIndexMap.begin();

while ( mapIter != m_columnIndexMap.end() )
{
    tempString = L"<TH>";
    tempString += (*mapIter).first.c_str();
    tempString += L"</TH>";
    m_piResponse->Write( CComVariant( tempString.c_str() ) );
}
```

```

        mapIter++;
    }
    m_piResponse->Write( CComVariant( "</TR>" ) );

```

列的名称作为键存储在列映射中，因此，可以遍历列映射得到列的名称，并显示在表格的表头中。像前面一样，将字符串转换为 variant，与 Response 对象一起送到浏览器中，一旦列输出完成，接下来处理行：

```

// Output the rows.
ROW_VECTOR::iterator      rowIter = m_rows.begin();
INDEX_FIELD_MAP::iterator fieldIter;

while ( rowIter != m_rows.end() )
{
    m_piResponse->Write( CComVariant( "<TR>" ) );

    mapIter = m_columnIndexMap.begin();

    while ( mapIter != m_columnIndexMap.end() )
    {
        tempString = L"<TD>";

        fieldIter = (*rowIter).find( (*mapIter).second );

        if ( fieldIter != (*rowIter).end() )
        {
            tempString += (*fieldIter).second.c_str();
        }
    }
}

```

每一行是 vector 数据结构中的一个元素，因此可遍历 m\_row。列的数据与列的标题对应。上一章讲过行数据使用映射可使存储空间最小。因此，可遍历列名称以判定特定行是否有值，如果其值存在，则输出，否则，输出一个空的表格单元。

最后关闭 HTML 标记并递增遍历器：

```

        tempString += L"</TD>";
        m_piResponse->Write( CComVariant( tempString.c_str() ) );

        mapIter++;
    }

    m_piResponse->Write( CComVariant( "</TR>" ) );
    rowIter++;
}

m_piResponse->Write( CComVariant( "</TABLE>" ) );

hResult = S_OK;
}
return hResult;
}

```

使用此组件的 ASP 代码类似于：

```
<H2>TableStorage C++ Component Test Driver</H2>
```

```

<%
Dim    objTableStorage2
Dim    csvString

csvString = "Name, Group, Instrument" & vbNewLine
csvString = csvString & "Robert Plant,Led Zeppelin,Vocals" & vbNewLine

```

```
csvString = csvString & "Jim Morrison,The Doors,Vocals" & vbNewLine
csvString = csvString & "Lenny Kravitz,,Everything" & vbNewLine
csvString = csvString & "Keith Moon,The Who,Drums" & vbNewLine
csvString = csvString & "Jimi Hendrix,,Guitar & Vocals" & vbNewLine

Set objTableStorage2 = Server.CreateObject("ASPComponents.TableStorage2")

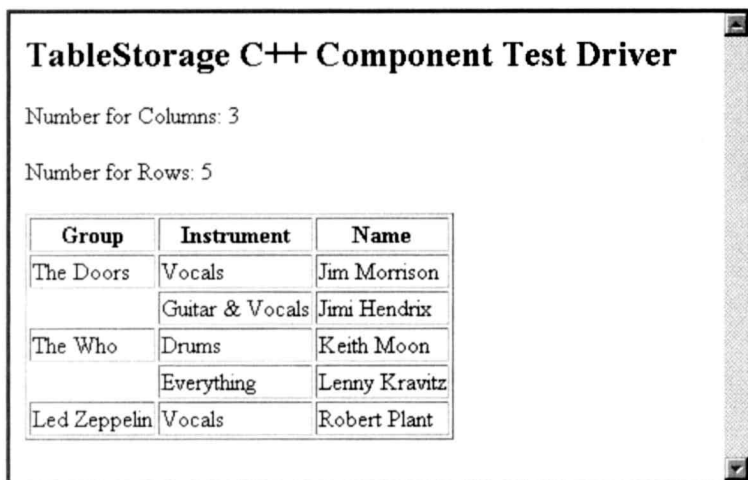
objTableStorage2.ParseCSV csvString
objTableStorage2.Sort "Name", 1

%>

<P>Number for Columns: <%=objTableStorage2.numColumns %></P>
<P>Number for Rows: <%=objTableStorage2.numRows%></P>

<%
    objTableStorage2.OutputTable 1
%>
```

这与上一章中的 ASP 代码非常相似，区别只是没有遍历字段以显示数据，仅须调用 OutputTable。这段代码比较简洁并可得到相似的结果，如图 18-4 所示。



Number for Columns: 3		
Number for Rows: 5		
Group	Instrument	Name
The Doors	Vocals	Jim Morrison
	Guitar & Vocals	Jimi Hendrix
The Who	Drums	Keith Moon
	Everything	Lenny Kravitz
Led Zeppelin	Vocals	Robert Plant

图18-4 浏览器中显示的表

此表仅是相似，列的顺序不同。这是因为 C++ 程序与 ASP 代码用不同的方式进行列遍历，C++ 中列按字母顺序存储在 map 数据结构中。另一方面，ASP 代码以列存储的顺序进行遍历，其存储的顺序是“Name、Group、Instrument”，是由上一章的例子生成的输出。可以扩展 OutputTable 方法，使用户可指定列的顺序，而不需要改变代码。

综上所述，如果可以与其环境交互，C++ 服务器组件会更有用。在这种情况下，C++ 组件可通过 COM 得到各种 ASP 内置对象的指针。这样可以让组件利用 ASP 的固有功能。我们仅分析了一个 ASP 内置对象，其余的都以同样的方式工作。下面介绍 C++ 服务器组件如何通过 COM+ 与其环境交互。

## 18.2 与 COM+ 进行接口

特别应当注意：Visual C++ 向导仍采用术语“MTS”，Visual C++ 头文件和静态库名称中仍含有“mtx”。因此，本节中我们使用术语“MTS/COM+”，而不只是“COM+”。



Visual C++在两个方面提供MTS/COM+支持：一是在创建项目时，二是在项目中使用指定的组件时。

首先来看一下MTS/COM+对于项目的支持。当编译和链接项目时，便转换为DLL或EXE。在项目中使用MTS/COM+服务时，必须链接指定的库。Visual C++ AppWizard可帮助完成此项任务。

在你第一次创建项目时，可能已注意到了Support MTS复选框，如图18-5所示。

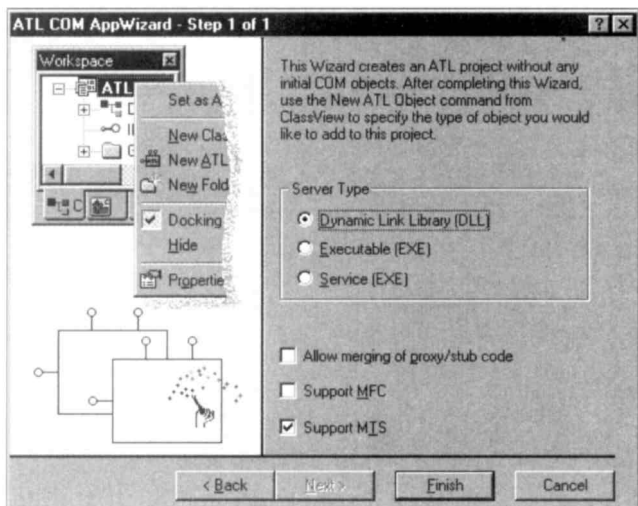


图18-5 Visual C++ AppWizard

如果选定此复选框，不会产生附加代码，但向导改变项目设置使其“启用COM+”。特别是，向链接线增加了mtx.lib、mtxguid.lib和delayimp.lib库。因此，如果没有对上述复选框进行选定，仍可通过如图18-6所示的方式在链接线中添加这些文件名，使现有的组件“启用COM+”。

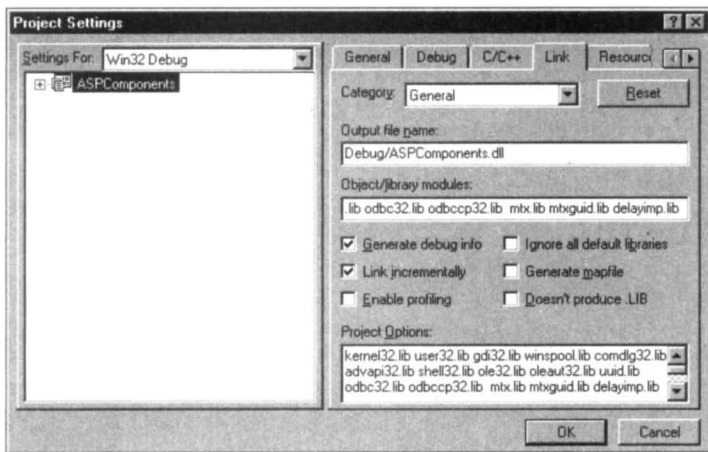


图18-6 添加库

另一个选择是创建启用COM+的组件。当在项目上添加一个新的ATL组件时，选择MS Transaction Server Component，如图18-7所示。

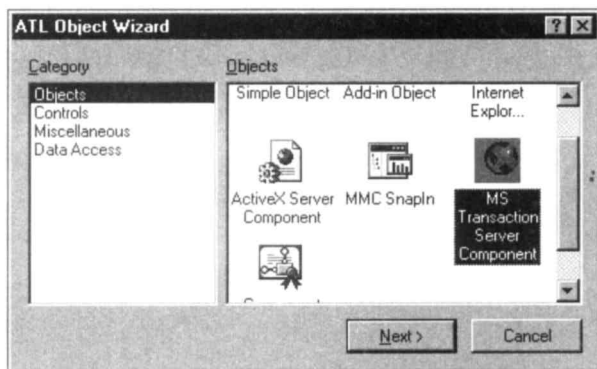


图18-7 在项目中增加新的 ATL 组件

单击属性页将出现图 18-8所示的内容。

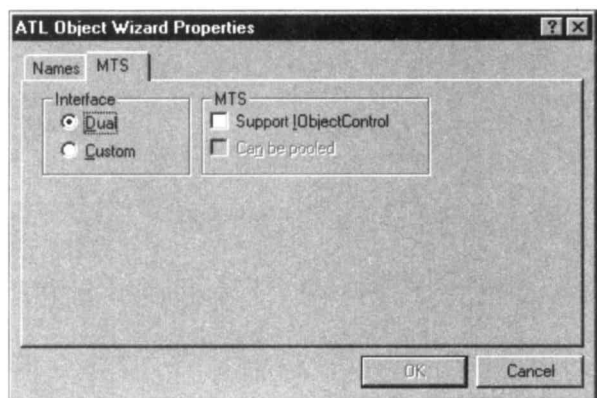


图18-8 ATL Object Wizard Properties对话框

创建MTS/COM+对象时，向导将在项目中添加代码，下面看一下这种配置产生的附加代码。

在新组件的头文件中，你可看到增加了一个包含文件：

```
#include <mtx.h>
```

和用于ASP内置对象的包含文件类似，此文件包含在 C++ 组件中使用 MTS/COM+ 组件所需的常数和定义。组件中头文件还有一处改变：

```
DECLARE_NOT_AGGREGATABLE(CTableStorage2)
```

MTS/COM+对象不能作为其他对象集合体的一部分使用。因此，必须插入代码防止这种情况发生。

如果不使用 AppWizard 创建组件，使组件“启用 COM+”是非常容易的。如果选择 IObjectControl 支持，应该在项目中插入下列附加代码。组件的头文件中也有些改变。

首先，组件继承 IObjectControl 接口，因此它必须实现该接口定义的方法：

```
class ATL_NO_VTABLE CTableStorage2 :  
    public CComObjectRootEx<CComSingleThreadModel>,  
    public CComCoClass<CTableStorage2, &CLSID_TableStorage2>,  
    public ISupportErrorInfo,  
    public IObjectControl,  
    ;
```



```
public IDispatchImpl<ITableStorage2, &IID_ITableStorage2,
&LIBID_ASPCOMPONENTSLib>
```

确保向导声明了 IObjectControl 的方法和ObjectContext的指针。

```
// IObjectControl
public:
    STDMETHOD(Activate)();
    STDMETHOD_(BOOL, CanBePooled)();
    STDMETHOD_(void, Deactivate)();

    CComPtr<IObjContext> m_spObjectContext;
```

向导在源文件中实现这些方法。当第一次使用对象时调用 Activate 方法，并自动生成ObjectContext的指针。

```
HRESULT CTableStorage2::Activate()
{
    HRESULT hr = GetObjectContext(&m_spObjectContext);
    if (SUCCEEDED(hr))
        return S_OK;
    return hr;
}
```

如果选择 IObjectControl 支持，而没有缓冲对象，CanBePooled 方法将返回 FALSE：

```
BOOL CTableStorage2::CanBePooled()
{
    return FALSE;
}
```

后面将详细讨论对象缓冲问题。最后，当对象不再使用时，将释放对 IObjContext 的引用。

```
void CTableStorage2::Deactivate()
{
    m_spObjectContext.Release();
}
```

对于服务器组件，有许多 MTS/COM+ 接口可用。下面以 IObjContext 接口为例进行讨论。

### 18.2.1 IObjContext 的事务处理

每个 MTS/COM+ 对象均有相应的环境。环境隐含着与对象相关的状态，包含对象的执行环境信息和参与的事务的信息。

IObjContext 可完成下列功能：

- 声明工作完成。
- 不允许提交暂时的或永久的事务。
- 在当前事务范围内开始新的 MTS/COM+ 对象。
- 检查调用者的角色。
- 检查安全性。
- 检查事务状态。

在前一节中，组件已经有一个对 IObjContext 接口的引用。我们将实现 Clear 方法，从组件中删除所有数据。对 ITableStorage2 接口添加一个不带任何参数的 Clear() 方法，用下列代码填写此方法的主体中：

```
STDMETHODIMP CTableStorage2::Clear()
{
    m_rows.clear();
    m_columnIndexMap.clear();
    if ( m_spObjectContext != NULL )
    {
        m_spObjectContext->SetComplete();
    }

    return S_OK;
}
```

这个方法的目的是删除组件中的所有数据。必须检查 `ObjectContext` 的引用是否确实存在。如果在 COM+ 以外使用组件，这个指针的值不存在，进行引用会引起组件崩溃。在这个例子中，调用 `SetComplete` 方法告诉 COM+ “不再保持状态，可以释放。”

### 18.2.2 IObjectControl的对象缓冲

`IObjectControl` 接口用于对象缓冲，可增强应用程序的性能。当超出对象的使用范围时，被缓冲的对象不会真的破坏，只是处于无效状态。当请求这种类型的对象时，可以重新使用这个对象，组件被循环使用。性能增强的原因是当一个对象无效时，对象没有完全损坏，因此就不需执行很多工作。同样，激活一个无效对象也不需执行很多工作。

在创建 MTS/COM+ 对象时，对象缓冲可通过 ATL Object Wizard Properties 对话框启用，如图 18-9 所示。

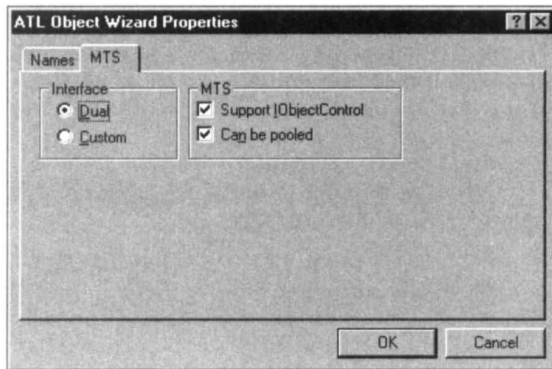


图18-9 启用对象缓冲

注意 `Can be pooled` 选项只能在选定 `Support IObjectControl` 后才能选定。另一种方式是将 `CanBePooled` 方法中的返回值改为 `TRUE`：

```
BOOL CTableStorage2::CanBePooled()
{
    return TRUE;
}
```

使用对象缓冲所需做的工作仅此而已。

上面讲述了组件如何与 ASP 和 COM+ 相互作用，下面我们讲述组件如何访问永久性数据。

## 18.3 数据访问

服务器组件一般用于实现三层应用程序的业务规则，因此需要与数据库交互。C++ 访问数

据有两种方式：ADO和OLE DB消费者模板。第12章我们已经讨论了通用数据访问(UDA)、ADO和OLE DB的作用。

从C++ADO与从ASP或Visual Basic访问ADO非常相似。使用完全一样的ADO COM对象(如ADODB.Recordset)，只是用不同的语言创建和使用ADO COM对象。但使用ADO时没有Visual C++向导，必须编写所有的实现代码。这不仅须手工编写，而且还容易引入代码错误，因此应认真仔细。但是可从C++中得到完全可用的ADO对象模型。

OLE DB消费者使用OLE DB提供者的服务访问数据。ADO是OLE DB消费者的一个例子。使用C++可创建OLE DB消费者，完全跳过ADO层。Visual C++向导可用于创建和使用OLE DB消费者。可以使用向导指向想要访问的数据库和表格，自动生成所需的代码。但是，因为消费者代码直接指向指定的数据库，所以不如使用ADO灵活。

### 18.3.1 通过C++使用ADO

通过C++使用ADO需要两种技能：理解ADO对象模型和在C++中使用COM对象。我们对ADO对象模型已非常熟悉，但现在没有C++类库或模板库支持ADO。因为ADO在COM上构造，使用ADO与使用其他COM对象相同。下面介绍一下使用步骤。

#### 1. 设置使用ADO

C++中使用COM组件最简单的方法是用# import关键字导入组件的类型库。# import 指令围绕ADO类型库产生瘦C++类，可以用接近于VBScript和Visual Basic的语句操作ADO对象和接口。大部分ADO例子用VBScript编写，用# import指令能很容易地将它们转换为C++，比在没有包装类(wrapper class)的情况下使用ADO容易。

使用#import时，可以使用no\_namespace属性，这样ADO类型库内容就没有范围限制。但是，这会引起名称与EOF的冲突，而rename属性可解决这个问题。在TableStorage2.cpp增加下列内容可导入ADO类型库。

```
// Include the ADO type library.
#import "c:\Program Files\Common Files\System\ADO\msado15.dll" \
    no_namespace \
    rename("EOF", "adoEOF")
#include <stdio.h>
```

上述代码生成一个msado15.tlh文件和一个msado15.tli文件。msado15.tlh文件包含前向引用、smart指针声明和typeinfo声明。msado15.tli文件包含编译器产生的几个成员函数的实现。这两个文件用于观察和显示类型库中的可用内容。例如，下面是msado15.tlh文件对ADO记录的smart指针的声明：

```
_COM_SMARTPTR_TYPEDEF(_Recordset, __uuidof(_Recordset));
```

后面的例子将说明如何在代码中使用smart指针。在msado15.tlh中还有一些枚举量的声明，使得使用ADO更为方便。下面是光标位置的枚举量声明：

```
enum CursorLocationEnum
{
    adUseNone = 1,
    adUseServer = 2,
    adUseClient = 3,
    adUseClientBatch = 3
};
```

如上所述，#import使得ADO的使用更为轻松，因为这个指令可自动创建包装类和来自类型库的类型信息。但对C++程序员来说，将ADO返回的VARIANT数据类型转变为C/C++数据类型还是比较麻烦。ADO 2.0和Visual C++ 6.0用附加的ADO VC++扩展解决了这个问题。这一章不讨论ADO VC++扩展，有关内容可参考Visual C++文献。

还增加了一个宏，这个宏可使程序容易理解，减少了嵌套代码的数量：

```
#define SAFE_CALL( expression ) { HRESULT hr; if ( FAILED(hr=expression) ) \
    _com_issue_error( hr ); }
```

通过导入类型库，已可使用所有 ADO对象。下面介绍如何在C++中使用ADO进行读写。我们将以图18-10所示的数据库为例。

假设对这个数据库已建立了一个称为Musician DSN的ODBC连接。

### 2. 读数据库

首先在ITableStorage2中增加Read方法。这个方法从数据库读记录，并将这些记录插入 STL数据结构中，它的参数是 DSN名称、用户名和口令。在对话框中，参数在 Implementation文本框中出现，如图 18-11所示。



Name	Group	Instrument
Ozzy Ozbourne	Black Sabbath	Vocals
Jim Morrison	The Doors	Vocals
Jimi Hendrix		Guitar & Vocals
Keith Moon	The Who	Drums
Lenny Kravitz		Everything
Robert Plant	Led Zeppelin	Vocals

图18-10 示例所用的数据库

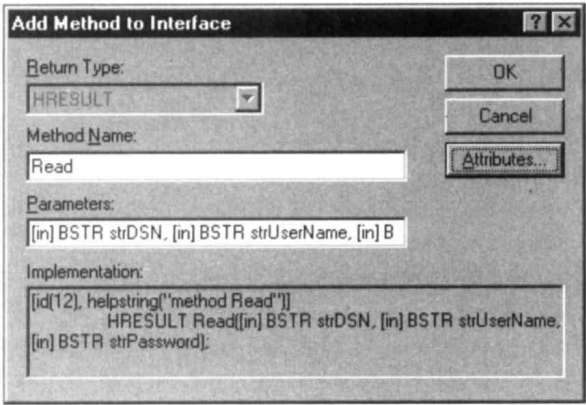


图18-11 Implementation文本框

下面看一下这个方法的实现：

```
STDMETHODIMP CTableStorage2::Read (BSTR strDSN, BSTR strUserName, BSTR strPassword)
{
    HRESULT          hResult = S_OK;
    _ConnectionPtr   pConn( "ADODB.Connection" ); // ADO Connection object
    _RecordsetPtr     pRS( "ADODB.Recordset" );    // ADO Recordset object
    INDEX_FIELD_MAP  fieldMap;
    int              fieldIndex;
    wstring           fieldName;
    wstring           fieldValue;
    CComBSTR          varFieldValue;
```

首先声明方法中使用的变量，注意如何使用 ATL smart指针的创建 ADO Connection和Recordset对象。为在代码中跟踪错误，我们采用了异常处理，因此开启一个 try块。任何ADO

调用都用前面声明的SAFE\_CALL宏包装，如果任一方法调用返回一个表明失败的值，就抛出一个异常：

```
try
{
    // Establish a connection.
    SAFE_CALL( pConn->Open( strDSN, strUserName, strPassword, 0 ) );
```

首先打开到数据库的连接，这个连接用于 Recordset对象的Open方法。

```
// Open the table.
SAFE_CALL( pRS->Open( "Musicians", pConn.GetInterfacePtr(),
                    adOpenDynamic, adLockOptimistic, adCmdTable) );
```

如果成功，可对数据库的记录进行遍历。首先必须确保内存中当前无数据：

```
// Make sure there is no data around.
m_rows.clear();
m_columnIndexMap.clear();

pRS->MoveFirst();
```

然后在STL数据结构中设置列名称：

```
// Create the columns first.
for ( fieldIndex = 0; fieldIndex < pRS->Fields->Count; fieldIndex++ )
{
    fieldName = pRS->Fields->Item[CComVariant(fieldIndex)]->Name;
    m_columnIndexMap[fieldName.c_str()] = m_columnIndexMap.size();
}
```

现在，将数据库中的数据拷贝到内存中的 STL数据结构内，遍历记录直到遇到文件结束标志EOF。只在字段真的有价值时才插入它，这样可以节约内存：

```
// Add data to the rows.
while ( !pRS->adoEOF )
{
    // Set the field values.
    for(fieldIndex = 0; fieldIndex < pRS->Fields->Count; fieldIndex++)
    {
        varValue = pRS->Fields->Item[CComVariant(fieldIndex)]->Value;

        if ( varValue.vt == VT_BSTR )
        {
            fieldValue = _bstr_t( varValue.bstrVal );

            if ( fieldValue.length() > 0 )
            {
                fieldMap[fieldIndex] = fieldValue.c_str();
            }
        }
    }

    m_rows.push_back( fieldMap );
    fieldMap.clear();

    pRS->MoveNext();
}

pRS->Close();
pConn->Close();
}
```

到目前为止，所有一切都在 `try` 块中。如果有错误出现，程序流将进入 `catch` 块。`_com_error` 变量包含了错误消息的细节，我们添加了一个辅助函数将错误的详细数据格式化为字符串。我们将这个错误字符串传送到 `Error` 中，这样，在客户端可以通过 `Error` 对象的 `Description` 属性得到它。

```
catch ( _com_error &theErr )
{
    string      resultMessage;

    FormatErrorMessage( theErr, resultMessage );

    Error( resultMessage.c_str() );
    HRESULT = E_FAIL;
}

return HRESULT;
```

`FormatErrorMessage` 方法如下：

```
void CTableStorage2::FormatErrorMessage(_com_error &theErr,
                                         string &strMessage )
{
    char      tempBuffer[64];

    strMessage = "COM Error object\n";
    sprintf( tempBuffer, "\tCode = %0x\n", theErr.Error() );
    strMessage += tempBuffer;

    _bstr_t bstrDescription(theErr.Description());
    strMessage += "Description = ";
    strMessage += (LPCSTR)bstrDescription;
    strMessage += "\n";
}
```

上面讲述了如何从数据库中读数据，下面讨论如何将数据写入数据库。

### 3. 写数据库

在 `ITableStorage2` 中添加下列方法。它将 STL 数据结构中的信息写入数据库中。操作界面如图18-12所示。

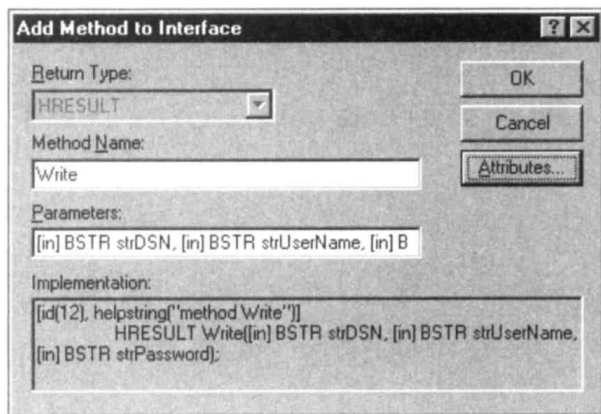


图18-12 添加Write方法



用ADO写数据库的代码如下：

```
STDMETHODIMP CTableStorage2::Write (BSTR strDSN, BSTR strUserName, BSTR
                                     strPassword)
{
    HRESULT          hResult = S_OK;
    _ConnectionPtr   pConn( "ADODB.Connection" );
    _RecordsetPtr     pRS("ADODB.Recordset");
```

和Read方法一样，声明一些ADO对象，建立与数据库的连接，并且打开数据表。

```
try
{
    // Establish a connection.
    SAFE_CALL( pConn->Open( strDSN, strUserName, strPassword, 0 ) );

    // Open the table.
    SAFE_CALL( pRS->Open( "Musicians", pConn.GetInterfacePtr(),
                          adOpenDynamic, adLockOptimistic, adCmdTable) );
```

下面开始写数据库，采用与OutputTable方法相似的风格，遍历迭代STL数据结构完成：

```
// Output the rows.
ROW_VECTOR::iterator      rowIter = m_rows.begin();
INDEX_FIELD_MAP::iterator fieldIter;
COLUMN_INDEX_MAP::iterator mapIter;

while ( rowIter != m_rows.end() )
{
    pRS->AddNew();

    mapIter = m_columnIndexMap.begin();

    // Iterate through the column names.
    while ( mapIter != m_columnIndexMap.end() )
    {
        fieldIter = (*rowIter).find( (*mapIter).second );

        // If the field has a value, add it to the database.
        if ( fieldIter != (*rowIter).end() )
        {
            pRS->Fields->GetItem((*mapIter).first.c_str())->Value =
                (*fieldIter).second.c_str();
        }
        mapIter++;
    }
    pRS->Update();
    rowIter++;
}
pRS->Close();
pConn->Close();
}
```

catch块与Read方法中的完全相同，这里不再重复。

你应该已经熟悉了对ADO的调用，但实现方式有些不同。例如，缺省ADO方法不能直接使用。在VB中，可用下列方法设置字段值：

```
objRS( "Name" ) = "Jimi Hendrix"
```

而在C++中必须完全扩展方法的调用，如：

```
PRS->Fields->GetItem( "Name" )->Value = "Jimi Hendrix"
```

另外一个差异是C++不能很好地支持Variant类型。上面的例子的代码看上去不错，但它仅由字符型数据组成。如果数据类型混用的话，代码将变得混乱。下面介绍的 OLE DB 消费者模板将使代码变得简洁。

### 18.3.2 OLE DB 消费者模板

OLE DB 消费者模板是一个相对新的方法，其目的是改善在 C++ 中进行数据存取的性能。它在 Visual C++ 6.0 中作为 ATL 3.0 的一部分引入使用库扩展了 C++ 功能，OLE DB 消费者模板库是为了通过 OLE DB 进行数据存取而设计的。这个库能够提供：

- 访问 OLE DB 特征。
- 集成 ATL 和 MFC。
- 数据库参数和列的绑定模型。
- 使用原有的 C/C++ 数据类型进行 OLE DB 编程。

另外，Visual C++ 的 AppWizard 可生成基本的 OLE DB 消费者代码。

#### 1. 创建 OLE DB 消费者

在创建 OLE DB 消费者前必须建立数据源，我们采用与 ADO 例子中所用的数据库相似的 Access 数据库。用在 ATL Object Wizard 添加一个新的 ATL 对象，选择 Data Access 类别并选择 Consumer，如图 18-13 所示。

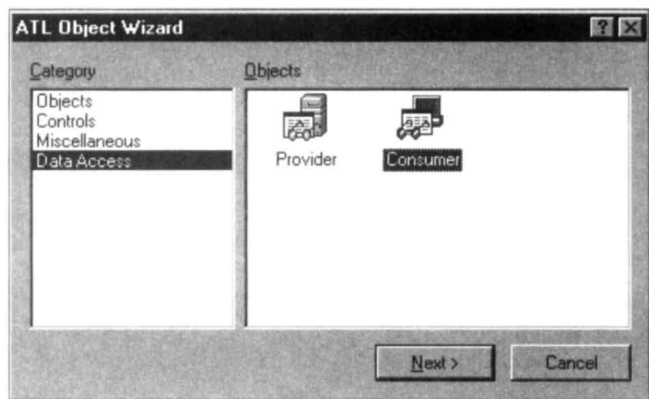


图18-13 在ATL Object Wizard中添加对象的界面1

点击 Next，选择所需的 OLE DB 提供者，因为我们选用了 ODBC 驱动程序，所以选择 Microsoft OLE DB Provider for ODBC Drivers。如图 18-14 所示。

选择所用的数据源名称为 MusicianDSN。用户名和口令没有输入，如果数据库需要用户名和口令，可在此对话框中输入，如图 18-15 所示。

点击 OK 按钮，然后选择所需的数据库表，如图 18-16 所示。

再点击 OK，向导将自动选取一些名称，根据自己的需要可进行修改。如果将要添加记录，可在 Support 部分选择 insert 框，如图 18-17 所示。

点击 OK 后，向导产生两个类：CMusiciansAccessor 和 CMusicians，可完成数据类型转换等工作。我们将只直接使用 CMusicians 类，下一节再做说明。

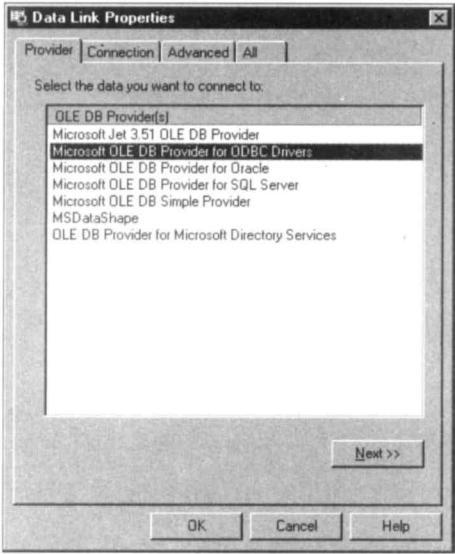


图18-14 在ATL Object Wizard中添加对象的界面2

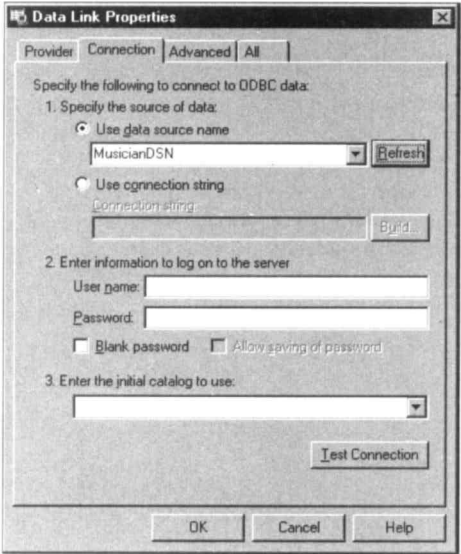


图18-15 在ATL Object Wizard中添加对象的界面3

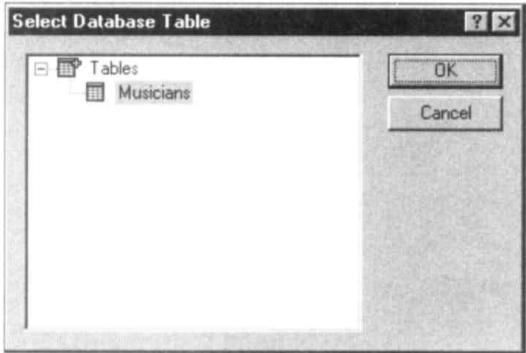


图18-16 在ATL Object Wizard中添加对象的界面4

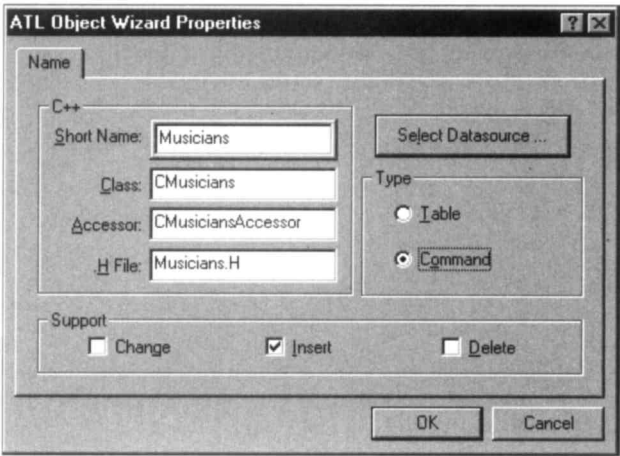


图18-17 在ATL Object Wizard中添加对象的界面5

## 2. 读数据库

下面在ITableStorage2中添加使用新的OLE DB消费者类读数据库的方法，由ATL Object Wizard生成的Read2不带任何参数，因为ATL Object Wizard已经将数据源名称、用户名和口令(如果你提供了)硬编码进CMusicians类，如图18-18所示。

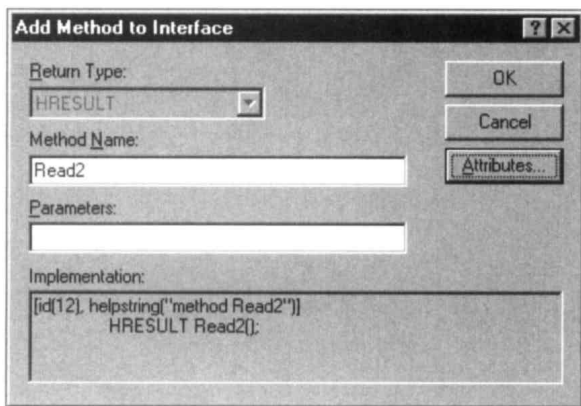


图18-18 生成Read2的屏幕界面

OLE DB 消费者模板版本的Read2方法的代码比ADO版本的代码更为简单：

```
STDMETHODIMP CTableStorage2::Read2()
{
    HRESULT          hResult = S_OK;
    CMusicians        theMusicians;
    INDEX_FIELD_MAP   fieldMap;

    theMusicians.Open();
```

如果上述代码没有编写的话，应在TableStorage2.cpp文件的顶部包含Musicians.h文件。

大部分工作已由theMusicians对象完成。当调用Open后，就创建了与数据库的连接，而且打开了Musicians表。下面的代码建立数据结构，列名称映射由手工创建：

```
// Make sure there is not any data around.
m_rows.clear();
m_columnIndexMap.clear();

// Create the columns first.
m_columnIndexMap[L"Name"] = 0;
m_columnIndexMap[L"Group"] = 1;
m_columnIndexMap[L"Instrument"] = 2;
```

下面遍历数据库记录。在使用OLE DB消费者模板时，要判定记录的结束，应检查MoveNext的返回值而不是检查EOF属性。当调用MoveNext时，theMusicians对象的成员变量自动填充，在读下一个记录前，必须先调用ClearRecord，否则当该记录的字段为NULL时，前一记录的数据仍将占据成员变量。

```
// Add data to the rows.
while ( theMusicians.MoveNext() == S_OK )
{
    // Set the field values.
    fieldMap[0] = _bstr_t( theMusicians.m_Name );
    fieldMap[1] = _bstr_t( theMusicians.m_Group );
```



```
{
    INDEX_FIELD_MAP::iterator   fieldIter;
    COLUMN_INDEX_MAP::iterator  mapIter;

    mapIter = m_columnIndexMap.find( pFieldName );
    fieldIter = (*rowIter).find( (*mapIter).second );

    // If the field has a value, add it to the database.
    if ( fieldIter != (*rowIter).end() )
    {
        USES_CONVERSION;
        strcpy( pBuffer, W2A( (*fieldIter).second.c_str() ) );
    }
}
```

### 18.3.3 使用ADO，还是使用OLE DB消费者模板

进行数据存取时，可选用ADO或OLE DB消费者模板。选用ADO可重新利用你对ADO对象模型的知识，代码修改也较容易，灵活性好。但ADO不能很好地支持绑定到C++的数据类型，并且Visual C++中没有相应的库和AppWizard支持。

选用OLE DB消费者模板可从Visual C++得到库和AppWizard支持。这意味着，Visual C++提供项目的基本结构，代码编写的速度较快，出现的错误也较少，但必须学习新的用于访问数据的API。如果你要编写许多需要访问数据的组件，那么这是值得的。

## 18.4 小结

本章创建了一个简单的C++服务器组件并增强其功能。我们介绍了如何访问ASP内置对象，如何访问各种COM+接口。描述了数据存取的两种方法。C++中可使用ADO对象模型，但需要进行大量的数据类型转换。ATL Object Wizard可创建OLE DB消费者，生成的代码比ADO简洁，但灵活性较差。

有了这些工具，在设计基于网络的应用程序结构时，可以有更大的选择余地。