**Лекция 8. Система клиент-сервер технологии СОМ в C #**

**1. Особенности технологии СОМ в .NET**

**2. Создание сервера**

**3. Регистрация сервера и создание библиотеки типов**

**4. Создание клиента**

**1. Особенности технологии СОМ в .NET**

Для обеспечения совместимости с программами клиент – сервер разработанными в средах программирования не .NET (например на языках DELPHI или С++ Builder)

.NET среды программирования поддерживающие С# позволяют решать следующие задач:

* Использовать в C # - приложениях СОМ –объекты созданные в предшествующих языках, путем их преобразования в сборки  **.NET**
* Создавать в C # - приложениях СОМ- объекты предназначенные для использования в предшествующих языках
* Использовать в C # - приложениях СОМ –объекты созданные в C # - приложениях но при этом они используются как обычные сборки  **.NET**

**2. Создание СОМ сервера**

Создание СОМ сервера в C # сводится к созданию специализированной библиотеки классов (модуля \*.dll) в который включается класс снабженный атрибутом **[ClassInterface(ClassInterfaceType.AutoDual**)] в состав которого включаются интерфейсы содержащие методы предназначенные для вызова пользователями. В классе могут присутствовать также собственные методы(не включенные с состав отдельных интерфейсов. Для поддержки технологии в состав сборки следует включить библиотеку using System.Runtime.InteropServices;

В рассмотренном ниже примере определены два интерфейса:

* Интерфейс IMath1 в составе которого определены методы

int Multiply(int x, int y)

int Divide(int x, int y);

* Интерфейс IDBTab в составе которого определен метод

DataTable getTab(string NameT);

В примере создается сом – сервер CLibCOMtest

содержащий СОМ –объект в виде класса ImpClass в который включены эти интерфейсы и реализованы их методы

Код сборки выглядит следующим образом

using System;

using System.Collections.Generic;

using System.Runtime.InteropServices;

using System.Linq;

using System.Data;

using System.Data.OleDb;

using System.Text;

namespace CLibCOMtest

{

public interface IMath1

{

int Multiply(int x,int y);

int Divide(int x, int y);

}

public interface IDBTab

{

DataTable getTab(string NameT);

}

[ClassInterface(ClassInterfaceType.AutoDual)]

public class ImpClass:IMath1,IDBTab

{

public ImpClass() {}

public int Multiply(int x,int y)

{return x\*y;}

public int Divide(int x,int y)

{

if (y == 0) throw new DivideByZeroException ();

return x/y;

}

public DataTable getTab(string NameT)

{

OleDbConnection cn = new OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\ASOER\_CB\ASOER\_DB\ASOER\_ML.mdb");

cn.Open();

OleDbCommand cmd = cn.CreateCommand();

cmd.CommandText = "Select \* from " + NameT ;

OleDbDataReader rdr = cmd.ExecuteReader();

DataTable dt = new DataTable();

dt.Load(rdr);

return dt;

}

}

}

**Для правильного формирования COM – сервера и его автоматической регистрации**

**необходимо установить два управляющих флага для проекта (Цепочка: Project -> имя\_проектаProperties:**

* **на вкладке Application ->** кнопк**а Assembly Information,** и в окне **Assembly Information** включить флажок **Make Assembly COM –Visible**
* **на вкладке Build** включить флажок **Register for COM interop**

**3. Ручная регистрация сервера и создание библиотеки типов**

**Раннее связывание клиента на основе модели компонентных объектов СОМ с компонентами .NET**

Ранне-связываемые клиенты на основе модели компонентных объектов Microsoft (СОМ) обычно используют информацию библиотеки типов для доступа к компонентам на основе модели компонентных объектов Microsoft (COM). Такая информация представляет собой удобный способ создания экземпляров классов на основе модели компонентных объектов Microsoft (COM), определенных в этих компонентах, созданных на основе модели компонентных объектов Microsoft (COM). Информация а библиотеках может храниться в файлах TLB, динамически подключаемых библиотеках (DLL), специализированных управляющих элементах OLE (OCX) и исполняемых файлах, однако только файлы TLB специально предназначены именно для этой цели.

Упаковщик Callable COM Wrapper, CCW)на основе модели компонентных объектов СОМ формирует из неуправляемого кода СОМ управляемым кодами .NET

Библиотека типов может быть сгенерирована с помощью утилиты Tlbexp.exe (Assembly to Type Library Converter — Конвертер сборки в библиотеку типов) из метаданных в сборке .NET. Библиотека типов позволяет клиентам на основе модели компонентных объектов Microsoft (СОМ) просматривать компоненты .NET так, как будто они являются обычными компонентами, построенными на основе модели компонентных объектов Microsoft (COM). Традиционный клиент на основе модели компонентных объектов СОМ может использовать информацию в полученной библиотеке типов для доступа к компонентам .NET, применяя раннее связывание.

Ниже приведен синтаксис для вызова утилиты командной строки Tlbexp. exe

**Tlbexp AssemblyName [параметры]**

Где AssemblyName – имя DLL (\*.dll)библиотеки содержащей СОМ - объект  
Параметры утилиты:

* /out:FileName Имя файла выходной библиотеки типов (\*.TLB  
  /nologo Подавляет вывод протокола
* / silent (тихий) Подавляет отображение сообщений
* / verbose (подробная) Дополнительная информация
* /? или /help (помощь) Вывести сообщение - справку по использованию

Утилита Tlbexp открывает для модели компонентных объектов СОМ только управляемые общедоступные (public) типы. Клиент на основе модели компонентных объектов СОМ никогда непосредственно не ссылается на СОМ-класс, а вместо этого имеет дело только с интерфейсом класса. Значение AutoDual (Автодуальный), которое задано в атрибуте Classlnterface (ClassInterfaceType: : AutoDual), автоматически генерирует дуальный (двойственный) интерфейс для доступа к СОМ классу . Ранне-связанные клиенты на основе модели компонентных объектов Microsoft (СОМ) могут использовать данный файл \*.tlb во время компиляции

Содержимое файла библиотеки типов можно просмотреть , используя сервисную программу для просмотра объектов OLE/COM — полноекранную утилиту Oleview.exe, расположенную в папке утилит SDK среды NET

// Generated .IDL file (by the OLE/COM Object Viewer)

//

// typelib filename: CLibCOMtest.tlb

[

uuid(0A7D2EFF-3462-46E2-9458-2CEB9E4EE186),

version(1.0),

custom(90883F05-3D28-11D2-8F17-00A0C9A6186D, "CLibCOMtest, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null")

]

library CLibCOMtest

{

// TLib : // TLib : mscorlib.dll : {BED7F4EA-1A96-11D2-8F08-00A0C9A6186D}

importlib("mscorlib.tlb");

// TLib : OLE Automation : {00020430-0000-0000-C000-000000000046}

importlib("STDOLE2.TLB");

// Forward declare all types defined in this typelib

interface IMath1;

interface IDBTab;

interface \_ImpClass;

[

odl,

uuid(DBB27999-6975-3B7F-A0E4-BBBFD924A011),

version(1.0),

dual,

oleautomation,

custom(0F21F359-AB84-41E8-9A78-36D110E6D2F9, "CLibCOMtest.IMath1")

]

interface IMath1 : IDispatch {

[id(0x60020000)]

HRESULT Multiply(

[in] long x,

[in] long y,

[out, retval] long\* pRetVal);

[id(0x60020001)]

HRESULT Divide(

[in] long x,

[in] long y,

[out, retval] long\* pRetVal);

};

[

odl,

uuid(627D823B-52E2-320E-870F-9950C5FB64A9),

version(1.0),

dual,

oleautomation,

custom(0F21F359-AB84-41E8-9A78-36D110E6D2F9, "CLibCOMtest.IDBTab")

]

interface IDBTab : IDispatch {

[id(0x60020000)]

HRESULT getTab(

[in] BSTR NameT,

[out, retval] IUnknown\*\* pRetVal);

};

[

uuid(0DFD18B3-5413-38D8-9CA5-8159F7D7A159),

version(1.0),

custom(0F21F359-AB84-41E8-9A78-36D110E6D2F9, "CLibCOMtest.ImpClass")

]

coclass ImpClass {

[default] interface \_ImpClass;

interface \_Object;

interface IMath1;

interface IDBTab;

};

[

odl,

uuid(E1629780-2939-3F50-A8E2-6430D41086DE),

hidden,

dual,

nonextensible,

oleautomation,

custom(0F21F359-AB84-41E8-9A78-36D110E6D2F9, "CLibCOMtest.ImpClass")

]

interface \_ImpClass : IDispatch {

[id(00000000), propget,

custom(54FC8F55-38DE-4703-9C4E-250351302B1C, 1)]

HRESULT ToString([out, retval] BSTR\* pRetVal);

[id(0x60020001)]

HRESULT Equals(

[in] VARIANT obj,

[out, retval] VARIANT\_BOOL\* pRetVal);

[id(0x60020002)]

HRESULT GetHashCode([out, retval] long\* pRetVal);

[id(0x60020003)]

HRESULT GetType([out, retval] \_Type\*\* pRetVal);

[id(0x60020004)]

HRESULT Multiply(

[in] long x,

[in] long y,

[out, retval] long\* pRetVal);

[id(0x60020005)]

HRESULT Divide(

[in] long x,

[in] long y,

[out, retval] long\* pRetVal);

[id(0x60020006)]

HRESULT getTab(

[in] BSTR NameT,

[out, retval] IUnknown\*\* pRetVal);

};

};

Для того, чтобы среда модели компонентных объектов СОМ могла найти нужную фабрику классов, путь к серверу, и т.п. COM- сервер должен быть зарегистрирован т.е. должна состояться запись информации о СОМ – сервере в системный реестр.

**Утилита регистрации сборки Regasm. exe** (Assembly Registration Utility) прочитывает метаданные в сборке и добавляет необходимые записи в системный реестр, что позволяет клиентам на основе модели компонентных объектов СОМ использовать компоненты сборки .NET так, как будто они являются обычными зарегистрированными компонентами, построенными на основе модели компонентных объектов Microsoft (COM). Конечно, клиенты при этом используют заместитель — вызываемый упаковщик на основе модели компонентных объектов СОМ (Callable COM Wrapper, CCW).   
Синтаксис вызова утилиты регистрации сборки :.

Regasm AssemblyPath [параметры]   
Где AssemblyPath – путь к регистрируемой сборке содержащей СОМ- объект(\*.dll)

Параметры утилиты :

* /unregister Отменить регистрацию типов
* /tlb[:FileName] Указанный файл библиотеки типов
* /regfile[:FileName] Указанное имя выходного файла
* /codebase (кодовая страница) Устанавливает кодовую страницу   
  в системном реестре
* /registered (зарегистрированный) Обращаться только к предварительно   
  зарегистрированным библиотекам типов
* /nologo Предотвращает вывод протокола
* /silent (тихий) Предотвращает отображение сообщений
* /verbose (подробно) Выводит дополнительную информацию
* /? или /help (помощь) Вывести сообщение - справку   
  по использованию

Регистрация позволяет любому клиенту на основе модели компонентных объектов СОМ получать доступ к компонентам .NET так, как будто это обычные компоненты, построенные на основе модели компонентных объектов Microsoft (COM).

Затем можно использовать утилиту Regedt32 . exe для проверки того, что информация была корректно занесена в системный реестр

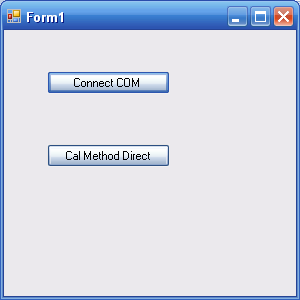
**3. Создание клиента**

Как только закончена ручная или автоматическая регистрация сборки \*. dll в качестве компонента на основе модели компонентных объектов Microsoft (COM), можно начинать разработку клиентского приложения.

Имеются два способа использования СОМ сервера **Клиентами .Net технологий**

1. Использование технологии COM/DCOM

Внешний вид формы программы клиент



Полный код клиента приведен ниже

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Reflection;

using System.Windows.Forms;

namespace WFAppClientCom

{

public partial class Form1 : Form

{

// объявлене СОМ – объекта

Object ComClass;

// обявление типа для получение информации о зарегистрированном СОМ – //объекте (типе)

Type type

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

// получение информации о зарегистрированном СОМ – //объекте (типе)

type = Type.GetTypeFromProgID("CLibComtest.ImpClass");

// создание СОМ – объекта

ComClass = Activator.CreateInstance(type);

if (ComClass != null)

MessageBox.Show("ComClass is created");

else

MessageBox.Show("ComClass is not created" );

}

private void button2\_Click(object sender, EventArgs e)

{

// получение ссылки на метод Multiply

MethodInfo mMulty = type.GetMethod("Multiply");

// вызов метода

int x = 5; int y = 7;

int r = (int)mMulty.Invoke(ComClass, new object[] { x, y });

MessageBox.Show(r.ToString());

// получение ссылки на метод getTab

MethodInfo mGetT = type.GetMethod("getTab");

// первый вызов метода

DataTable dt1 = (DataTable)mGetT.Invoke(ComClass, new object[] { "US\_CITIES" });

DataRow dr1 = dt1.Rows[3];

string name = dr1.Field<string>("NAME\_CITY").ToString();

MessageBox.Show("Назва = " + name);

// второй вызов метода

DataTable dt2 = (DataTable)mGetT.Invoke(ComClass, new object[] { "US\_REGION" });

DataRow dr2 = dt2.Rows[3];

string kilk = dr2.Field<double>("NUM\_APPARTMENT").ToString();

MessageBox.Show("Кількість appartment = " + kilk);

}

}

}

1. **Прямое подключение и использование класса реализации методов**

Полный код программы клиента

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Runtime.InteropServices;

using System.Linq;

using System.Text;

using ImpClass;

using System.Windows.Forms;

using System.Reflection;

namespace MyChrpServerClient

{

public partial class Form1 : Form

{

public object myComObjOB;

public ImpClass myComObjCalc;

IMath i1;

IDBTab i2;

int x = 100 ,y=20, r=0;

DataTable dt1;

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{ // Create OLE As Object and activate interfaces

myComObjOB = Activator.CreateInstance(typeof(ImpClass));

i1 = (IMath1)myComObjOB;

i2 = (IDBTab)myComObjOB;

this.label4.Text = "Creating OBJECT is success";

}

private void button2\_Click(object sender, EventArgs e)

{

// Call metods in firs interface IAdvancedMath i1;

r = i1.Multiply(x, y);

this.label1.Text = this.label1.Text + "="+r.ToString();

r = i1.Divide(x , y);

this.label2.Text = this.label2.Text + "="+r.ToString();

}

private void button3\_Click(object sender, EventArgs e)

{ // Create OLE As class CCharpCalc

myComObjCalc = new CCharpCalc();

this.label5.Text = "Creating CLASS is success";

}

private void button4\_Click(object sender, EventArgs e)

{

// Call metods in second interface IDBTab;

dt1 = i2.GetTab((”US\_SITIES”);

DataRow dr1 = dt1.Rows[3];

string name = dr1.Field<string>("NAME\_CITY").ToString();

MessageBox.Show("Назва = " + name);

}

}

}}

**Клиенты не .Net технологий** , а традиционных СОМ приложений должны создать COM объект традиционным способом, например с помощью импорта библиотеки типов для обеспечения раннего связывания, а затем вызовом CreateOleObject

используют сервер как обычную создают СОМ – объект ( в нашем случае CLibCOMtest)средствами используемого языка.

**Другой пример**

using System;

using System.Collections.Generic;

using System.Text;

using System.Runtime.InteropServices;

using System.Windows.Forms;

namespace Com01

{

[ComVisible(true)]

[Guid("1B97A425-1947-4e05-A0C6-C82B4C317B2A")]

public interface IComClass

{

double Pow(double x, double y);

DialogResult AboutBox();

}

[ComVisible(true)]

[ClassInterface(ClassInterfaceType.AutoDual)]

[Guid("F0635A7A-5C4B-45a1-9297-8471CEF4DEE3")]

public class ComClass : IComClass

{

public double Pow(double x, double y)

{

return Math.Pow(x,y);

}

public DialogResult AboutBox()

{

AboutBox1 box = new AboutBox1();

return box.ShowDialog();

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

using System.Reflection;

namespace ComClient

{

public partial class ComClient : Form

{

Type type;

Object comClass;

public ComClient()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

// button run com dialog about abox

MethodInfo mi = type.GetMethod("AboutBox");

//double f = (double)mi.Invoke(comClass, new object[] { 10, 10 });

mi.Invoke(comClass, null);

//comClass.Pow(2,10);

//comClass c1 = new ComClass();

//IComClass com1 = (IComClass)c1;

//MessageBox.Show(com1.Pow(10, 2).ToString());

/\*catch (Exception ex)

{

Console.WriteLine("Unexpected COM exception: " + ex.Message);

}\*/

}

private void button2\_Click(object sender, EventArgs e)

{

// button Count Power

double x = decimal.ToDouble(numericUpDown1.Value);

double y = decimal.ToDouble(numericUpDown2.Value);

/\* Type type = Type.GetTypeFromProgID("Com01.ComClass");

Object comClass;

comClass = Activator.CreateInstance(type);

\* \*/

MethodInfo mPow = type.GetMethod("Pow");

double f = (double)mPow.Invoke(comClass, new object[] { x, y });

textBox1.Text = f.ToString("0.000");

// double f = (double)mi.Invoke(comClass, new object[] { x, y });

// Com01.IComClass i1 = (Com01.IComClass)comClass;

Com01.IComClass i1 = (Com01.IComClass)comClass;

double f1 = (double)i1.Pow(x, y);

textBox2.Text = f1.ToString("0.000");

}

private void button3\_Click(object sender, EventArgs e)

{

// buttonb Cout Powe 2 couse

/\* Object myComObjOB ;

myComObjOB = Activator.CreateInstance(typeof(Com01.ComClass));

\* \*/

double x = decimal.ToDouble(numericUpDown1.Value);

double y = decimal.ToDouble(numericUpDown2.Value);

Com01.IComClass i1 = (Com01.IComClass)comClass;

// i2 = (IAdvancedMath2)myComObjOB;

double f = (double)i1.Pow(x, y );

textBox1.Text = f.ToString("0.000");

}

private void button4\_Click(object sender, EventArgs e)

{

// connect to Com button

type = Type.GetTypeFromProgID("Com01.ComClass");

comClass = Activator.CreateInstance(type);

if (comClass != null)

MessageBox.Show("Connect to Com is successful");

else

MessageBox.Show("Connect to Com is sunuccessful");

}

}

}