**Лекція 9 . Система клієнт-сервер технології СОМ в C #**

**1. Особливості технології СОМ в. NET**

**2. Створення сервера**

**3. Реєстрація сервера та створення бібліотеки типів**

**4. Створення клієнта**

1. Особливості технології СОМ в. NET

Для забезпечення сумісності з програмами клієнт - сервер розробленими в середовищах програмування не. NET (наприклад на мовах DELPHI або С + + Builder) . NET середовища програмування підтримують С # дозволяють вирішувати наступні завдання

- Використовувати в C # - додатках СОМ-об'єкти створені в попередніх мовах, шляхом їх перетворення в збірки. NET

- Створювати в C # - додатках СОМ-об'єкти призначені для використання в попередніх мовами

- Використовувати в C # - додатках СОМ-об'єкти створені в C # - додатках але при цьому вони використовуються як звичайні збірки. NET

**2. Створення СОМ сервера**

Створення СОМ сервера в C # зводиться до створення спеціалізованої бібліотеки класів (модуля \*. dll) в який включається клас забезпечений атрибутом [ClassInterface (ClassInterfaceType.AutoDual)] до складу якого включаються інтерфейси містять методи призначені для виклику користувачами. До класі можуть бути присутніми також власні методи (не включені до складу окремих інтерфейсів. Для підтримки технології до складу збірки слід включить бібліотеку using System.Runtime.InteropServices;

У розглянутому прикладі визначено два інтерфейсу:

- Інтерфейс IAdvancedMath у складі якого визначені методи int Multiply (int x, int y) і int Divide (int x, int y);

- Інтерфейс IAdvancedMath2 у складі якого визначено метод int Mul10 (int x)

У прикладі створюється сом - сервер MyChrpServerLib СОМ - містить СОМ-об'єкт CCharpCalc в який включені ці інтерфейси і реалізовані їхні методи

   Крім того в складі ласа визначені два власних методу public int Add (int x, int y) і public int Substruct (int x, int y)

Код збірки виглядає наступним чином

using System;

using System.Collections.Generic;

using System.Runtime.InteropServices;

using System.Linq;

using System.Text;

namespace MyChrpServerLib

{

public interface IAdvancedMath

{

int Multiply(int x, int y);

int Divide(int x, int y);

}

public interface IAdvancedMath2

{

int Mul10(int x);

}

[ClassInterface(ClassInterfaceType.AutoDual)]

public class CCharpCalc : IAdvancedMath, IAdvancedMath2

{

public CCharpCalc() { }

public int Add(int x, int y) { return x + y; }

public int Substruct(int x, int y) { return x - y; }

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 int Mul10(int x)

{

return x \* 10;

}

}

}

Для правильного формування COM - сервера і його автоматичної реєстрації необхідно встановити два керуючих прапора для проекту (Ланцюжок: Project -> імя\_проектаProperties:

* на вкладці Application -> кнопка Assembly Information,у вікні Assembly Information включити прапорець Make Assembly COM-Visible
* на вкладці Build включити прапорець Register for COM interop

Приклад иншого сервера з програмно керуємою реестрвціею

using System;

using System.Windows.Forms;

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);

}

[ComVisible(true), Guid("0DFD18B3-5413-38D8-9CA5-8159F7D7A159")]

[ClassInterface(ClassInterfaceType.AutoDual)]

public class ImpClass:IMath1,IDBTab

{

public ImpClass() {}

public int Multiply(int x,int y)

{

MessageBox.Show("A im Multiply");

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();

cn.ConnectionString = @"Provider=Microsoft.Jet.OLEDB.4.00.0000;Data Source=D:\ASOER\_CB\ASOER\_DB\ASOER\_ML.mdb";

cn.Open();

OleDbCommand cmd = cn.CreateCommand();

cmd.CommandText = "Select \* from US\_House";

OleDbDataReader dr = cmd.ExecuteReader();

DataTable dt = new DataTable();

dt.Load(dr);

return dt;

}

}

}

**3. Ручна реєстрація сервера та створення бібліотеки типів**

**Раннє зв'язування клієнта на основі моделі компонентних об'єктів СОМ з компонентами. NET**

Клієнти, створені на основі на основі моделі компонентних об'єктів Microsoft (СОМ) зв’язуються з сервером шляхом попереднього ( раннього) зв’язування зазвичай використовують інформацію бібліотеки типів для доступу до компонентів на основі моделі компонентних об'єктів Microsoft (COM). Така інформація являє собою зручний спосіб створення примірників класів на основі моделі компонентних об'єктів Microsoft (COM), визначених в цих компонентах, створених на основі моделі компонентних об'єктів Microsoft (COM). Інформація а бібліотеках може зберігатися у файлах TLB, динамічно підключаються бібліотеках (DLL), спеціалізованих керуючих елементах OLE (OCX) і виконуваних файлах, однак тільки файли TLB спеціально призначені саме для цієї мети.

Пакувальник Callable COM Wrapper, CCW) на основі моделі компонентних об'єктів СОМ формує з некерованого коду СОМ керованим кодами. NET

Бібліотека типів може бути згенеровано за допомогою утиліти Tlbexp.exe (Assembly to Type Library Converter - Конвертер збірки в бібліотеку типів) з метаданих в збірці. NET. Бібліотека типів дозволяє клієнтам на основі моделі компонентних об'єктів Microsoft (СОМ) переглядати компоненти. NET так, як ніби вони є звичайними компонентами, побудованими на основі моделі компонентних об'єктів Microsoft (COM). Традиційний клієнт на основі моделі компонентних об'єктів СОМ може використовувати інформацію в отриманій бібліотеці типів для доступу до компонентів. NET, застосовуючи раннє зв'язування.

Нижче наведено синтаксис для виклику утиліти командного рядка

Tlbexp. exe

**Tlbexp AssemblyName [параметри]**

де AssemblyName - ім'я DLL (\*. dll) бібліотеки містить СОМ - об'єкт

Параметри утиліти:

- / Out: FileName Файл вихідний бібліотеки типів (\*. TLB

/ Nologo Пригнічує висновок протоколу

- / Silent (тихий) Пригнічує відображення повідомлень

- / Verbose (докладна) Додаткова інформація

- /? або / help (допомога) Вивести повідомлення - довідку з використання

Утиліта Tlbexp відкриває для моделі компонентних об'єктів СОМ тільки керовані загальнодоступні (public) типи. Клієнт на основі моделі компонентних об'єктів СОМ ніколи безпосередньо не посилається на СОМ-клас, а замість цього має справу тільки з інтерфейсом класу. Значення AutoDual (Автодуальний), яке задано в атрибуті Classlnterface (ClassInterfaceType:: AutoDual), автоматично генерує дуальний (двоїстий) інтерфейс для доступу до СОМ класу. Ранньо-зв'язані клієнти на основі моделі компонентних об'єктів Microsoft (СОМ) можуть використовувати даний файл \*. tlb під час компіляції

Вміст файлу бібліотеки типів можна переглянути, використовуючи сервісну програму для перегляду об'єктів OLE / COM - полноекранную утиліту Oleview.exe, розташовану в папці утиліт SDK середовища NET// Generated .IDL file (by the OLE/COM Object Viewer)

//

// typelib filename: MyChrpServerLib.tlb

[

uuid(B1F66461-D68F-4160-8998-5907E92C6E98),

version(1.0),

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

]

library MyChrpServerLib

{

// 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 IAdvancedMath;

interface IAdvancedMath2;

interface \_CCharpCalc;

[

odl,

uuid(2814184F-53A2-365C-BBAF-BC1B5464E5AC),

version(1.0),

dual,

oleautomation,

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

]

interface IAdvancedMath : 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(D0CC968D-2743-3F84-A8BE-566A34EEC30C),

version(1.0),

dual,

oleautomation,

custom(0F21F359-AB84-41E8-9A78-36D110E6D2F9, "MyChrpServerLib.IAdvancedMath2")

]

interface IAdvancedMath2 : IDispatch {

[id(0x60020000)]

HRESULT Mul10(

[in] long x,

[out, retval] long\* pRetVal);

};

[

uuid(479A881E-BBAE-3CC8-9828-8C01AE21CD9C),

version(1.0),

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

]

coclass CCharpCalc {

[default] interface \_CCharpCalc;

interface \_Object;

interface IAdvancedMath;

interface IAdvancedMath2;

};

[

odl,

uuid(6C104F1F-B191-3952-9B63-DD0FE1068FEA),

hidden,

dual,

oleautomation,

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

]

interface \_CCharpCalc : IDispatch {

};

};

Для того, щоб середовище моделі компонентних об'єктів СОМ могла знайти потрібну фабрику класів, шлях до сервера, і т.п. COM-сервер повинен бути зареєстрований тобто повинна відбутися запис інформації про СОМ - сервер до реєстру.

Утиліта реєстрації збірки Regasm. exe (Assembly Registration Utility) прочитує метадані в збірці і додає необхідні записи до реєстру, що дозволяє клієнтам на основі моделі компонентних об'єктів СОМ використовувати компоненти збірки. NET так, як ніби вони є звичайними зареєстрованими компонентами, побудованими на основі моделі компонентних об'єктів Microsoft ( COM). Звичайно, клієнти при цьому використовують заступник - викликається пакувальник на основі моделі компонентних об'єктів СОМ (Callable COM Wrapper, CCW).

Синтаксис виклику утиліти реєстрації збирання:.

Regasm AssemblyPath [параметри]

Де AssemblyPath - шлях до реєстрованої збірці містить СОМ-об'єкт (\*. dll)

Параметри утиліти:

- / Unregister Скасувати реєстрацію типів

- / Tlb [: FileName] Вказаний файл бібліотеки типів

- / Regfile [: FileName] Зазначене ім'я вихідного файлу

- / Codebase (кодова сторінка) Встановлює кодову сторінку

- В системному реєстрі

- / Registered (зареєстрований) Звертатися тільки до попередньо

- Зареєстрованим бібліотекам типів

- / Nologo Запобігає висновок протоколу

- / Silent (тихий) Запобігає відображення повідомлень

- / Verbose (докладно) Виводить додаткову інформацію

- /? або / help (допомога) Вивести повідомлення - довідку

- По використанню

Реєстрація дозволяє будь-якому клієнту на основі моделі компонентних об'єктів СОМ отримувати доступ до компонентів. NET так, як ніби це звичайні компоненти, побудовані на основі моделі компонентних об'єктів Microsoft (COM).

Потім можна використовувати утиліту Regedt32. exe для перевірки того, що інформація була коректно занесена до реєстру

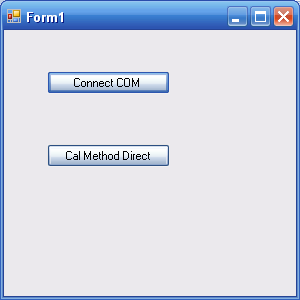
**3. Створення клієнта**

Як тільки закінчена реєстрація збірки \*. dll як компонент на основі моделі компонентних об'єктів Microsoft (COM), можна починати розробку клієнтського додатку. Для початку слід інсталювати збірку в клієнтський проект. Можна, звичайно, все виконати, просто скопіювавши збірку \*. dll в папку клієнта Зробивши це, можна, нарешті, виконати клієнт, побудований на основі моделі компонентних об'єктів COM.

Маються два шляхи використання створеного СОМ сервера клієнтами .Net технологій

**1. Використання технології COM/DCOM**

Зовнішній вигляд інтерфейсу користувача клієнтської програми



Повний програмний код клієнта виглядає наступним чином

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

{

// оголошення СОМ – об’екта

Object ComClass;

// оголошення змінної type типа Type для отримання інформації про зареєстрований СОМ – //обєект (тип )

Type type

public Form1()

{

InitializeComponent();

}

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

{

// отримання інформації про зареєстрований СОМ – //об’ект (тип )

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

// створення СОМ – об’екта

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)

{

// отримання опису метода Multiply

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

// ввиклик метода

int x = 5; int y = 7;

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

MessageBox.Show(r.ToString());

// отримання опису метода getTab

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

// першій виклик 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);

// другій виклик 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);

}

}

}

**2 Пряме використання COM об'єкта як звичайного класу**

Клієнтом не. Net технологій, а традиційних СОМ програм слід створити COM об'єкт традиційним способом, наприклад за допомогою імпорту бібліотеки типів для забезпечення раннього зв'язування, а потім викликом CreateOleObject

Клієнти. Net технологій використовують сервер як звичайну dll бібліотеку, яка підключається до проекту за допомогою оператора using, в нашому випадку бібліотеку MyChrpServerLib;

Клієнти. Net технологій використовують сервер як звичайну dll бібліотеку підключається до СС допомогою оператора using, в нашому випадку MyChrpServerLib;

Далі робота може йти двома шляхами:

З явним використанням наявності інтерфейсів

У цьому випадку активізація сервера виробляється з використанням змінної типу object і активізацією інтерфейсів

public object myComObjOB;

IAdvancedMath i1;

IAdvancedMath2 i2;

// Create OLE As Object and activate interfaces

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

i1 = (IAdvancedMath) myComObjOB;

i2 = (IAdvancedMath2)myComObjOB;

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

Після цього можна викликати методи відповідного інтерфейсу, наприклад:  
           / / Call metods in firs interface IAdvancedMath i1;  
             r = i1.Multiply (x, y);  
             this.label1.Text = this.label1.Text + "=" + r.ToString ();  
  
З використанням усіх методів без явного поділу по інтерфейсах  
У цьому випадку активізація сервера виробляється з використанням змінної типу класа та активізацією інтерфейсів

public CCharpCalc myComObjCalc;

// Create OLE As class CCharpCalc

myComObjCalc = new CCharpCalc();

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

После этого можно вызывать любые методы класса, например:

// Call all methods in class

// Method in interface IAdvancedMath

r = myComObjCalc.Multiply(x, y);

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

// Method in interface IAdvancedMath2

r = myComObjCalc.Mul10(2);

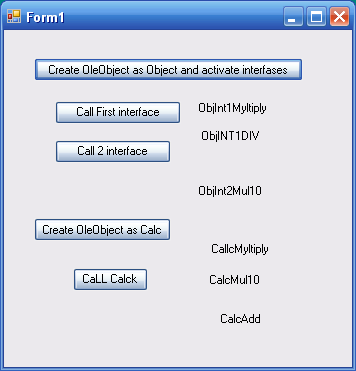
this.CalcMul10.Text = this.CalcMul10.Text + "="+ r.ToString();

// Direct Method in class myComObjCalc

r = myComObjCalc.Add(x,y);

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

Интерфейс программы клиента



Повний код програми клієнта

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 MyChrpServerLib;

using System.Windows.Forms;

using System.Reflection;

namespace MyChrpServerClient

{

public partial class Form1 : Form

{

public object myComObjOB;

public CCharpCalc myComObjCalc;

IAdvancedMath i1;

IAdvancedMath2 i2;

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

public Form1()

{

InitializeComponent();

}

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

{ // Create OLE As Object and activate interfaces

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

i1 = (IAdvancedMath) myComObjOB;

i2 = (IAdvancedMath2)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 IAdvancedMath2 i2;

r = i2.Mul10(x);

this.label3.Text = this.label3.Text + "="+r.ToString();

}

private void button5\_Click(object sender, EventArgs e)

{

// Call all methods in class

// Method in interface IAdvancedMath

r = myComObjCalc.Multiply(x, y);

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

// Method in interface IAdvancedMath2

r = myComObjCalc.Mul10(2);

this.CalcMul10.Text = this.CalcMul10.Text + "="+ r.ToString();

// Direct Method in class myComObjCalc

r = myComObjCalc.Add(x,y);

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

}

}

}}

Інший приклад( и сервер и слиент в одному Solution

Код сервера

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()

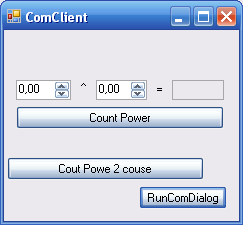
{

AboutBox1 box = new AboutBox1();

return box.ShowDialog();

}

}



Код клиента

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

{

public ComClient()

{

InitializeComponent();

}

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

{

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

Object comClass;

//comClass = (IComClass)Activator.CreateInstance(type);

//comClass = (IComClass)(new ComClass());

comClass = Activator.CreateInstance(type);

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)

{

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 mi = type.GetMethod("Pow");

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

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

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

}

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

{

Object myComObjOB ;

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

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

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

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

// i2 = (IAdvancedMath2)myComObjOB;

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

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

}

}