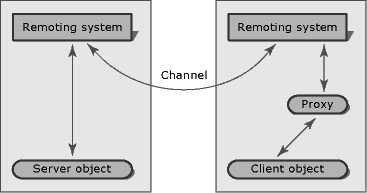
**Лекція 11 Технологія Remoting. Об’єкт що активізується сервером**

Технологія Remoting розроблена для створення розподілених додатків. З її допомогою можна звертатися до екземплярів класів .Net, які перебувають за межами власного домену додатки (application domain, логічний процес). Це може бути інший додаток усередині одного процесу, інший процес на тій же машині, або процес на іншій машині (у тому числі підключеної через Internet).  
  
  
  
 **Рисунок 1. Механизм роботи Remoting**  
  
Клієнт деяким чином отримує об'єктну посилання на серверний об'єкт, з якою взаємодіє через проксі-об'єкт. Коли клієнт викликає метод, інфраструктура віддаленого взаємодії відправляє виклик по каналу в серверний процес. Хто слухає канал приймає запит і направляє його в серверну систему віддаленого взаємодії, яка знаходить (або створює, якщо потрібно) і викликає запитуваний об'єкт. Потім процес протікає в зворотному напрямку: серверна система віддаленого взаємодії упаковує відповідь в повідомлення, яке відправляється серверним каналом клієнтського каналу Типова архітектура триланкової системи з окремо створюваним Remoting-об'єктом у вигляді бібліотеки, Remoting сервером, що надає об'єкт клієнтам, Remoting клієнтом, які отримують об’єкт і сервером бд

Сервер БД

Реализация запросов

1.Создание канала

2.Регистрация канала

3.Регистрация объекта

Remoting-сервер

Remoting-объект

С методами и реализацией

Remoting-клиент

1. Регистрация объекта
2. Создание объекта
3. Вызов методов

створення об'єкта  
Об'єкт можна створити як звичайну dll -бібліотеки класів (Class Library), в якій створюється клас, що містить реалізацію методів призначених для виклику клієнтами. Ім'я класу - ім'я remoting - об'єкта  
Залежно від способу надання об'єкта клієнтам є два варіанти оголошення класу:  
1 варіант: Передача об'єкта за посиланням  
public class RemOb1: System.MarshalByRefObject  
     {  
         public RemOb1 ()  
         {  
           // MessageBox.Show ("Constructor RemOb1 called");  
             
         }  
          
         public DataTable GetADOTable (string name)  
         {String connstr = "Provider = Microsoft.Jet.OLEDB.4.0; Data Source = D: \\ ASOER\_CB \\ ASOER\_DB \\ ASOER\_ML.mdb";  
              DataTable restab; OleDbCommand Ocmd; OleDbConnection Conn;  
           // DataSet XMLDataSet = new DataSet (); // DataRow dr;  
              Conn = new OleDbConnection (connstr);  
              Ocmd = Conn.CreateCommand ();  
              Ocmd.CommandText = name;  
              Ocmd.CommandType = CommandType.Text;  
              Conn.Open ();  
              OleDbDataAdapter da = new OleDbDataAdapter (Ocmd.CommandText, Conn);  
              restab = new DataTable ();  
              da.Fill (restab);  
            // MessageBox.Show ("Metod GetADOTable called");  
             return (restab);  
}  
}  
  
2варіант: Передача об'єкта за значенням (шляхом серіалізації)  
[Serializable]  
public class RemOb1  
     {  
         public RemOb1 ()  
         {// MessageBox.Show ("Constructor RemOb1 called"); }  
          
         public DataTable GetADOTable (string sqlname)  
         {String connstr = "Provider = Microsoft.Jet.OLEDB.4.0; Data Source = D: \\ ASOER\_CB \\ ASOER\_DB \\ ASOER\_ML.mdb";  
              DataTable restab; OleDbCommand Ocmd; OleDbConnection Conn;  
           // DataSet XMLDataSet = new DataSet (); // DataRow dr;  
              Conn = new OleDbConnection (connstr);  
              Ocmd = Conn.CreateCommand ();  
              Ocmd.CommandText = sqlname;  
              Ocmd.CommandType = CommandType.Text;  
              Conn.Open ();  
              OleDbDataAdapter da = new OleDbDataAdapter (Ocmd.CommandText, Conn);  
              restab = new DataTable ();  
              da.Fill (restab);  
            // MessageBox.Show ("Metod GetADOTable called");  
             return (restab);  
}  
}  
  
**Серверний додаток**Об'єкти, що активуються сервером, - це об'єкти, час існування яких безпосередньо контролюється сервером. Домен серверного додатка створює ці об'єкти тільки тоді, коли клієнт викликає метод такого об'єкта, а не коли клієнт використовує оператор new або викликає метод Activator.GetObject; це дозволяє уникнути відправлення запиту по мережі і отримання відповіді тільки з метою створення екземпляра. Коли клієнт запитує екземпляр типу, активируемого сервером, в домені клієнтського додатку створюється тільки проксі. Це означає, що для типів, які активуються сервером, можна використовувати тільки конструктори за умовчанням. Для публікації типу, екземпляри якого будуть створюватися за допомогою особливих конструкторів, які беруть аргументи, повинна використовуватися Активація клієнта або динамічна публікація конкретного екземпляра.  
  
На сервері спочатку необхідно створити і реєструвати канал доступу до об'єкта, наприклад  
// Створюємо Tcp або Http канал channel  
     channel = new TcpChannel (888); // Варіант HttpChannel (888)  
   // Реєструємо канал channel;  
             ChannelServices.RegisterChannel (channel, false);  
   
  Потім, для створення об'єкта активируемого сервером його необхідно зареєструвати, вказавши при цьому режим активації. Для реєстрації використовується два варіанти методу реєстрації: **1 варіант, з** режимом активації Singleton (один об'єкт для обслуговування всіх клієнтів)  
RemotingConfiguration. RegisterWellKnownServiceType (typeof (ServerObject), "uri",  
                WellKnownObjectMode.Singleton);  
У типів Singleton ніколи не може одночасно бути більш одного екземпляра. Якщо екземпляр є, всі запити клієнтів обслуговуються цим екземпляром. Якщо примірника ні, сервер створює його і всі наступні запити клієнтів обслуговуються цим екземпляром. Оскільки для типів Singleton задається час існування за замовчуванням, клієнти не будуть завжди отримувати посилання на один і той же екземпляр віддаленого типу, навіть якщо в кожний конкретний момент часу доступно не більше одного екземпляра.  
**2 варіант,** з режимом активації SingleCall (для обслуговування кожного клієнта створюється свій об'єкт)  
RemotingConfiguration. RegisterWellKnownServiceType (typeof (ServerObject), "uri", WellKnownObjectMode.SingleCall);  
Повний опис методу  
Метод RemotingConfiguration.RegisterWellKnownServiceType (WellKnownServiceTypeEntry) реєструє об'єкт Type в якості добре відомого типу (єдиного виклику або одноелементна множини, активізується сервером за зверненням клієнта).  
список перевантаження

|  |  |
| --- | --- |
| ім'я | Опис |
| RegisterWellKnownServiceType (WellKnownServiceTypeEntry) | Реєструє об'єкт Type, записаний в наданому WellKnownServiceTypeEntry з боку служби як добре відомого типу. |
| RegisterWellKnownServiceType (Type, String, WellKnownObjectMode) | Реєструє об'єкт Type з боку клієнта як добре відомого типу, використовуючи параметри, задані для ініціалізації нового екземпляра класу WellKnownServiceTypeEntry |

Винятки: SecurityException Принаймні в одного з викликають операторів, що знаходяться у верхній частині стека викликів, відсутній дозвіл на налаштування типів і каналів віддаленого взаємодії.  
Процесом реєстрації не створюється екземпляр віддаленого об'єкта. Це відбувається, тільки коли клієнт намагається викликати метод на об'єкт або активувати об'єкт з боку клієнта.  
У типів SingleCall завжди створюється один екземпляр на кожен клієнтський запит. Черговий виклик методу буде обслуговуватися іншим серверним екземпляром, навіть якщо попередній екземпляр ще не був знищений системою. Типи SingleCall не беруть участь у процесах оренди часу існування.  
Використання конфігураційного файлу  
Розглянемо базовий варіант функції Main реалізує сервер:  
{  
HttpChannel oJobChannel = new HttpChannelf 4000);  
ChannelServices.RegisterChannel (oJobChannel);  
// Реєстрація загальновідомого типу.  
RemotingConfiguration.RegisterWellKnownServiceType (typeof (JobServerlmpl),  
            "JobURI", WellKnownObjectMode .Singleton);  
}  
Все добре, але що якщо ми захочемо поміняти номер порту? Тоді доведеться перекомпілювати сервер. Правда «Номер порту можна просто передати в командному рядку». Цей прийом працює, але не вирішує таких проблем, як додавання нових каналів.  
  Необхідний спосіб перенесення деталей налаштування з коду програми в конфігураційний файл. Використання конфігураційного файлу дозволяє адміністратору налаштовувати поведінку програми без необхідності перекомпілювати його. Але найголовніше, моженоте замінити весь попередній код одним рядком! Новий варіант функції Main:  
  
{  
RemotingConfiguration.Configure( @ ". D .. \ JobServer.exe.config");  
  }  
  
Нова версія Main складається з одного рядка. Вся конфігураційна інформація .NET Remoting зберігається тепер у файлі JobServer.exe.config.  
  
     ПРИМІТКА За угодою ім'я конфігураційного файлу утворюється шляхом додавання до імені виконуваного файлу програми рядки «.config».  
Нижче показаний конфігураційний файл JobServer.exe.config:  
  
<configuration>  
      <system.runtime.remoting>  
         < Application name = "JobServer">  
                <service>  
                    <wellknown mode = "Singleton"  
  
                      type = "JobServerLib. JobServerlmpl, JobServerLib"  
                      obJectUri = "JobURI" />  
                </ service>  
                <channels>  
                     <charmel ref = "http"  
                         port = "4000" />  
                </ channels>  
           </ application>  
      </system.runtime.remoting>  
</ configuration>  
  
Зверніть увагу на те, як співвідноситься вміст конфігураційного файлу і код віддаленого взаємодії, доданий раніше. Елемент <channel /> містить ту ж саму інформацію для налаштування каналу, яка була використана в первинному варіанті програмної конфігурації. Програмна реєстрація загальновідомого об'єкта замінена на елемент <wellknown> конфігураційного файлу. Інформація для регістра ції об'єктів з серверної активізацією укладена в елементі <$ ervice>. І елемент <service>, і елемент <channefs> можуть містити кілька вкладених елементів. Як видно з цього прикладу, можливості конфігураційних файлів просто вражаючі  
клієнтську програму  
Щоб створити екземпляр типу, активируемого сервером, клієнтські програми повинні бути налаштовані програмним чином або за допомогою файлу конфігурації. У разі налаштування програми програмним чином незалежно від способу активації є два способи отримання **об'єкта  
1 спосіб - з попередньою реєстрацією**  
Спочатку проводиться реєстрація методом  
RemotingConfiguration.RegisterWellKnownClientType(typeof(имя\_объекта),"протокол://адрес:порт/uri");  
                
             Object obj = null;  
              obj = new ім'я об'єкта ();  
приклад:  
   // Реєстрація WellKnown об'єкта з боку клієнта  
             RemotingConfiguration.RegisterWellKnownClientType (typeof (RemOb1), "TCP: // localhost: 888 / Rserv");  
              this.label1.Text = "реєстрація об'єкта з боку клієнта пройшла";  
             obj = null;  
              obj = new RemOb1 ();  
Повний опис методу  
метод RemotingConfiguration.RegisterWellKnownClientType  
Реєструє об'єкт Type з боку клієнта як добре відомого типу (єдиного виклику або одноелементна множини).  
список перевантаження

|  |  |
| --- | --- |
| ім'я | Опис |
| RegisterWellKnownClientType (WellKnownClientTypeEntry) | Реєструє об'єкт Type, записаний в наданий WellKnownClientTypeEntry з боку клієнта як добре відомого типу, який може бути активований сервером. |
| RegisterWellKnownClientType (Type, String) | Реєструє об'єкт Type з боку клієнта як добре відомого об'єкта, який може бути активований на сервері, використовуючи дані параметри для ініціалізації нового екземпляра класу WellKnownClientTypeEntry |

**2 спосіб - без попередньої реєстрації**  
У цьому випадку для створення на клієнті примірника об'єкта, який активується сервером, використовується метод Activator.GetObject.  
У наступному прикладі коду показаний виклик методу Activator.GetObject в припущенні, що канал TcpChannel зареєстрований для зв'язку через порт 8080. Якщо клієнтові відомий тільки факт, що в серверному об'єкті реалізований певний інтерфейс, необхідно викликати метод Activator.GetObject, оскільки для створення екземпляра класу можна використовувати тільки оператор new (New в Visual Basic).  
Object obj = (імя\_об) Activator.GetObject (typeofімя\_об), "канал: // адреса: порт / Uri");  
приклад  
this.obj = (RemOb1) Activator.GetObject (typeof (RemOb1), "TCP: // localhost: 888 / Rserv");  
  
Слід пам'ятати, що на даному етапі ніякої взаємодії з сервером не відбувається і віддалений об'єкт не створюється. Створюється тільки екземпляр проксі-об'єкта на стороні клієнта. Тепер клієнт може використовувати об'єкт RemOb1 як якщо б у нього була пряме посилання на віддалений об'єкт. Примірник RemOb1, який клієнт фактично використовує для взаємодії при різних викликах методів, залежить від того, до якого типу належить серверний об'єкт - Singleton або SingleCall. Незалежно від того, розкриває цю інформацію видавець серверного об'єкта, клієнт використовує наявну в нього посилання на об'єкт абсолютно однаково.  
виклик методів  
Після отримання об'єкта використовуючи точковий синтаксис можна викликати його методи, наприклад  
DataTabe dt = new DataTable ();  
String sqls = @ "select \* from Personal"  
dt = obj.GetADOTable (sqls)  
   
Примітка. На сервері можна реєструвати також і канал доступу, але можливо, реєструвати канал на стороні клієнта не буде потрібно. Якщо клієнт не реєструє канал, служба віддаленого взаємодії автоматично створює канал для вихідних запитів, використовуючи один з каналів за замовчуванням, заданих у файлі Machine.config. При такому автоматичному виборі каналів на клієнті цей канал не реєструється для очікування функцій зворотного виклику з сервера і, якщо тільки в файл machine.config не додано користувальницький канал, не реєструються ніякі користувальницькі реалізації каналу. У цих випадках потрібно зареєструвати тип каналу, який потрібно Об'єкти Singleton  
Використання конфігураційного файлу для клієнтського застосування  
Другий метод, що надається .NET Remoting для налаштування параметрів віддаленого взаємодії, використовує конфігураційні файли. Раніше було розглянуто використанні конфігураційних файлів в розділі «Реалізація програми JobServer».  
При налаштування клієнтського додатка застосовуються інші теги. Ось так виглядає XML-код конфігураційного файлу програми JobClient:  
<configuration>  
     <system.runtime.remoting>  
        <application name = "JobCHent">  
            <client>  
                <wellknown  
                   type = "JobserverLib.JobServerlmpl, JobServerLib"  
                   url = "http: // localhost: 4000 / JobURI" />  
            </ client>  
            <channels>  
                <channel ref = "http" port = "0" />  
            </ channels>  
        </ application>  
     </system.runtime.remoting>  
</ configuration>  
Використання файлу JobClient.exe.config для настройки віддаленого взаємодії шляхом виклику RemotingConfiguration.Configure:  
private IJobServe r GetJobServerQ  
{  
      RemotingConfiguration.Configure (@ ".. \ .. \ JobClient.exe.config"  
      return (IJobServer) new JobServerImpl;

CALLBACK