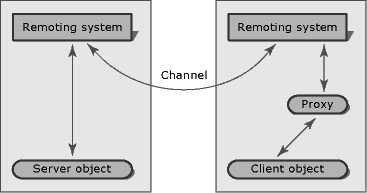
**Лекция 11 Технология Remoting. Объект активизируемый сервером**

Технология Remoting разработана для создания распределенных приложений. С ее помощью можно обращаться к экземплярам классов .Net, находящимся за пределами собственного домена приложения (application domain, логический процесс). Это может быть другое приложение внутри одного процесса, другой процесс на той же машине, или процесс на другой машине (в том числе подключенной через Internet).  
  
Очень упрощенно, механизм .NET Remoting работает следующим образом.   
  
  
  
  
 **Рисунок 1. Механизм работы Remoting**  
  
Клиент некоторым образом получает объектную ссылку на серверный объект, с которой взаимодействует через прокси-объект. Когда клиент вызывает метод, инфраструктура удаленного взаимодействия отправляет вызов по каналу в серверный процесс. Слушающий канал принимает запрос и направляет его в серверную систему удаленного взаимодействия, которая находит (или создает, если нужно) и вызывает запрашиваемый объект. Затем процесс протекает в обратном направлении: серверная система удаленного взаимодействия упаковывает ответ в сообщение, которое отправляется серверным каналом клиентскому каналу. Наконец, клиентская система удаленного взаимодействия возвращает результат вызова клиентскому объекту через прокси-объект.  
Типовая архитектура трехзвенной системы с отдельно создаваемым Remoting-объектом в виде библиотеки, Remoting сервером, предоставляющим объект клиентам, Remoting клиентом, получающим обект и сервером бд

Remoting-клиент

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

Сервер БД

Реализация запросов

1.Создание канала

2.Регистрация канала

3.Регистрация объекта

Remoting-объект

С методами и реализацией

Remoting-сервер

# Создание объекта

Объект можно создать как обычную dll –библиотеку классов(Class Library), в которой создается класс, содержащий реализацию методов предназначенных для вызова клиентами. Имя класса – имя remoting – объекта

В зависимости от способа предоставления объекта клиентам имеются два варианта объявления класса:

1 вариант: Передача объекта по ссылке

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);

}

}

2вариант: Передача объекта по значению( путем сериализации)

[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);

}

}

# Серверное приложение

Объекты, активируемые сервером, — это объекты, время существования которых напрямую контролируется сервером. Домен серверного приложения создает эти объекты только тогда, когда клиент вызывает метод такого объекта, а не когда клиент использует оператор **new** или вызывает метод [Activator.GetObject](http://msdn.microsoft.com/ru-ru/library/system.activator.getobject.aspx); это позволяет избежать отправки запроса по сети и получения ответа только с целью создания экземпляра. Когда клиент запрашивает экземпляр типа, активируемого сервером, в домене клиентского приложения создается только прокси. Это значит, что для типов, активируемых сервером, можно использовать только конструкторы по умолчанию. Для публикации типа, экземпляры которого будут создаваться с помощью особых конструкторов, принимающих аргументы, должна использоваться [Активация клиента](http://msdn.microsoft.com/ru-ru/library/w93betdk.aspx) или динамическая публикация конкретного экземпляра.

На сервере сначала необходимо создать и регистрировать канал доступа до объекта, например

// Создаем Tcp или Http канал channel

channel = new TcpChannel(888); // вариант HttpChannel(888)

// Регистрируем канал channel;

ChannelServices.RegisterChannel(channel, false);

Затем, для создания объекта активируемого сервером его необходимо зарегистрировать, указав при этом режим активации. Для регистрации используется два варианта метода регистрации:

1 вариант, с режимом активации Singleton( один объект для обслуживания всех клиентов)

RemotingConfiguration. RegisterWellKnownServiceType(typeof(ServerObject), "uri",

WellKnownObjectMode.Singleton);

У типов **Singleton** никогда не может одновременно быть более одного экземпляра. Если экземпляр есть, все запросы клиентов обслуживаются этим экземпляром. Если экземпляра нет, сервер создает его и все последующие запросы клиентов обслуживаются этим экземпляром. Поскольку для типов **Singleton** задается время существования по умолчанию, клиенты не будут всегда получать ссылку на один и тот же экземпляр удаленного типа, даже если в каждый конкретный момент времени доступно не более одного экземпляра.

**2 вариант,** с режимом активации **SingleCall** **(для обслуживания каждого клиента создаётся свой объект)**

RemotingConfiguration. RegisterWellKnownServiceType(typeof(ServerObject), "uri", WellKnownObjectMode.**SingleCall**);

**Полное описание метода**

Метод RemotingConfiguration.RegisterWellKnownServiceType (WellKnownServiceTypeEntry) регистрирует объект [Type](http://msdn.microsoft.com/ru-ru/library/system.type(v=VS.90).aspx) в качестве хорошо известного типа (единственного вызова или одноэлементного множества, активизируется сервером по обращению клиента).

Список перегрузки

|  |  |
| --- | --- |
| Имя | Описание |
| [RegisterWellKnownServiceType (WellKnownServiceTypeEntry)](http://msdn.microsoft.com/ru-ru/library/3xzb9z8s(v=VS.90).aspx) | Регистрирует объект [Type](http://msdn.microsoft.com/ru-ru/library/system.type(v=VS.90).aspx), записанный в предоставленном [WellKnownServiceTypeEntry](http://msdn.microsoft.com/ru-ru/library/system.runtime.remoting.wellknownservicetypeentry(v=VS.90).aspx) со стороны службы в качестве хорошо известного типа. |
| [RegisterWellKnownServiceType](http://msdn.microsoft.com/ru-ru/library/czshbka7(v=VS.90).aspx)  [(Type, String, WellKnownObjectMode)](http://msdn.microsoft.com/ru-ru/library/czshbka7(v=VS.90).aspx) | Регистрирует объект [Type](http://msdn.microsoft.com/ru-ru/library/system.type(v=VS.90).aspx) со стороны клиента в качестве хорошо известного типа, используя параметры, заданные для инициализации нового экземпляра класса [WellKnownServiceTypeEntry](http://msdn.microsoft.com/ru-ru/library/system.runtime.remoting.wellknownservicetypeentry(v=VS.90).aspx). |

**Исключения** : [SecurityException](http://msdn.microsoft.com/ru-ru/library/system.security.securityexception.aspx) По крайней мере у одного из вызывающих операторов, находящихся в верхней части стека вызовов, отсутствует разрешение на настройку типов и каналов удаленного взаимодействия.

Процессом регистрации не создается экземпляр удаленного объекта. Это происходит, только когда клиент пытается вызвать метод на объект или активировать объект со стороны клиента.

У типов **SingleCall** всегда создается один экземпляр на каждый клиентский запрос. Очередной вызов метода будет обслуживаться другим серверным экземпляром, даже если предыдущий экземпляр еще не был уничтожен системой. Типы **SingleCall** не участвуют в процессах аренды времени существования.

**Использование конфигурационного файла**

Рассмотрим базовый вариант функции Main реализующий сервер:

{

HttpChannel oJobChannel = new HttpChannelf 4000 );

ChannelServices.RegisterChannel( oJobChannel );

// Регистрация общеизвестного типа.

RemotingConfiguration.RegisterWellKnownServiceType( typeof( JobServerlmpl ),

"JobURI", WellKnownObjectMode .Singleton) ;

}

Все хорошо, но что если мы захотим поменять номер порта? Тогда придется перекомпилировать сервер. Правда «Номер порта можно просто передать в командной строке». Этот прием работает, но не решает таких проблем, как добавление новых каналов.

Необходим способ переноса деталей настройки из кода программы в конфигурационный файл. Использование конфигурационного файла позволяет администратору настраивать поведение приложения без необходимости перекомпилировать его. Но самое главное, моженоте заменить весь предыдущий код одной строкой! Новый вариант функции Main:

{

**RemotingConfiguration.Configuref @".Д..\JobServer.exe.config" );**

}

Новая версия Main состоит из одной строки. Вся конфигурационная информация .NET Remoting хранится теперь в файле JobServer.exe.config.

ПРИМЕЧАНИЕ По соглашению имя конфигурационного файла образуется путем добавления к имени исполняемого файла приложения строки «.config».

Ниже показан конфигурационный 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>

Обратите внимание на то, как соотносится содержимое конфигурационного файла и код удаленного взаимодействия, добавленный ранее. Элемент<channel/> содержит ту же самую информацию для настройки канала, которая была использована в первоначальном варианте программной конфигурации. Программная регистрация общеизвестного объекта заменена на элемент <wellknown> конфигурационного файла. Информация для регистра ции объектов с серверной активизацией заключена в элементе <$ervice>. И элемент <service>, и элемент <channefs> могут содержать несколько вложенных элементов. Как видно из данного примера, возможности конфигурационных файлов просто поразительны

**Клиентское приложение**

Чтобы создать экземпляр типа, активируемого сервером, клиентские приложения должны быть настроены программным образом или с помощью файла конфигурации. В случае настройки приложения программным образом независимо от способа активации имеется два способа получения объекта

**1 способ – с предварительной регистрацией**

Сначала проводится регистрация методом

RemotingConfiguration.RegisterWellKnownClientType(typeof(имя\_объекта),"протокол://адрес:порт/uri");

Object obj = null;

obj = new имя объекта();

Пример:

// регистрация WellKnown объекта со стороны клиента

RemotingConfiguration.RegisterWellKnownClientType(typeof(RemOb1),"TCP://localhost:888/Rserv");

this.label1.Text = " регистрация объекта со стороны клиента прошла";

obj = null;

obj = new RemOb1();

**Полное описание метода**

Метод RemotingConfiguration.RegisterWellKnownClientType

Регистрирует объект [**Type**](http://msdn.microsoft.com/ru-ru/library/system.type.aspx) со стороны клиента в качестве хорошо известного типа (единственного вызова или одноэлементного множества).

Список перегрузки

|  |  |
| --- | --- |
| Имя | Описание |
| [RegisterWellKnownClientType](http://msdn.microsoft.com/ru-ru/library/zx2by16x.aspx)  [(WellKnownClientTypeEntry)](http://msdn.microsoft.com/ru-ru/library/zx2by16x.aspx) | Регистрирует объект [Type](http://msdn.microsoft.com/ru-ru/library/system.type.aspx), записанный в предоставленный [WellKnownClientTypeEntry](http://msdn.microsoft.com/ru-ru/library/system.runtime.remoting.wellknownclienttypeentry.aspx) со стороны клиента в качестве хорошо известного типа, который может быть активирован сервером. |
| [RegisterWellKnownClientType](http://msdn.microsoft.com/ru-ru/library/hcd5d6y7.aspx)  [(Type, String)](http://msdn.microsoft.com/ru-ru/library/hcd5d6y7.aspx) | Регистрирует объект [Type](http://msdn.microsoft.com/ru-ru/library/system.type.aspx) со стороны клиента в качестве хорошо известного объекта, который может быть активирован на сервере, используя данные параметры для инициализации нового экземпляра класса [WellKnownClientTypeEntry](http://msdn.microsoft.com/ru-ru/library/system.runtime.remoting.wellknownclienttypeentry.aspx). |

**2 способ – без предварительной регистрации**

В этом случае для создания на клиенте экземпляра объекта, активируемого сервером, используется метод Activator.GetObject.

В следующем примере кода показан вызов метода **Activator.GetObject** в предположении, что канал [TcpChannel](http://msdn.microsoft.com/ru-ru/library/system.runtime.remoting.channels.tcp.tcpchannel.aspx) зарегистрирован для связи через порт 8080. Если клиенту известен только факт, что в серверном объекте реализован определенный интерфейс, необходимо вызывать метод **Activator.GetObject**, поскольку для создания экземпляра класса можно использовать только оператор **new** (**New** в Visual Basic).

Object obj = (имя\_об)Activator.GetObject(typeofимя\_об), "канал://адрес:порт/Uri");

Пример

this.obj = (RemOb1)Activator.GetObject(typeof(RemOb1), "TCP://localhost:888/Rserv");

Следует помнить, что на данном этапе никакого взаимодействия с сервером не происходит и удаленный объект не создается. Создается только экземпляр прокси-объекта на стороне клиента. Теперь клиент может использовать объект RemOb1 как если бы у него была прямая ссылка на удаленный объект. Экземпляр RemOb1, который клиент фактически использует для взаимодействия при различных вызовах методов, зависит от того, к какому типу относится серверный объект — **Singleton** или **SingleCall**. Независимо от того, раскрывает ли эту информацию издатель серверного объекта, клиент использует имеющуюся у него ссылку на объект совершенно одинаково.

**Вызов методов**

После получения объекта используя точечный синтаксис можно вызывать его методы, например

DataTabe dt = new DataTable();

String sqls = @”select \* from Personal”

dt = obj.GetADOTable(sqls)

**Примечание.** На сервере можно регистрировать также и канал доступа, но возможно, регистрировать канал на стороне клиента не потребуется. Если клиент не регистрирует канал, служба удаленного взаимодействия автоматически создает канал для исходящих запросов, используя один из каналов по умолчанию, заданных в файле Machine.config. При таком автоматическом выборе каналов на клиенте этот канал не регистрируется для ожидания функций обратного вызова с сервера и, если только в файл machine.config не добавлен пользовательский канал, не регистрируются никакие пользовательские реализации канала. В этих случаях нужно зарегистрировать тип канала, который требуется Объекты Singleton

**Использование конфигурационного файла для клиентского приложения**

Второй метод, предоставляемый .NET Remoting для настройки параметров удаленного взаимодействия, использует конфигурационные файлы. Ранее было рассмотрено использовании конфигурационных файлов в разделе «Реализация приложения JobServer».

При настройке клиентского приложения применяются другие теги. Вот так выглядит XML-код конфигурационного файла приложения 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>

Использование файла JobClient.exe.config для настройки удаленного взаимодействия путем вызова RemotingConfiguration.Configure:

private IJobServe r GetJobServerQ

{

RemotingConfiguration.Configure(@"..\..\JobClient.exe.config"

return (IJobServer)new JobServerImpl;