10,833,602 members iagsav 359 Sign out

CODE PROJECT For those who code

home articles quick answers discussions features community help

Search for articles, questions, tips

Articles » General Programming » Internet / Network » Remoting

Simple Sample for .NET Remoting

Manoj Kumar Raju, 14 Feb 2007

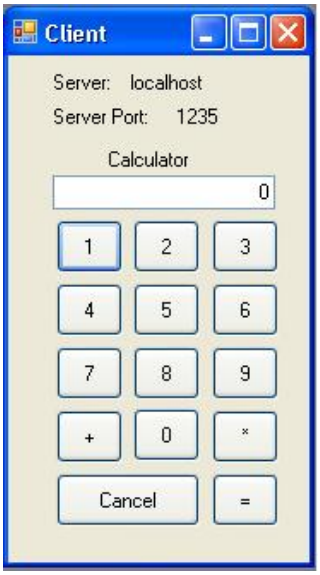
Rate:

4.10 (10 votes)

This article is for beginners

Download demo project - 15.94 KB

Download source code - 64.83 KB



Introduction

The intention of writing this article is to make beginners understand the concept of .NET Remoting. The sample project along with this article is a simple application which clearly gives an idea for a beginner in .NET Remoting.

.NET Remoting

.NET Remoting provides a framework for designing distributed applications. Remoting is an architecture which enables communication between objects living in different application domains. Application domain is a small execution unit of an application. An application domain cannot access the code of another application domain. Communication among the objects across the application boundaries takes place through a process called as Marshalling.

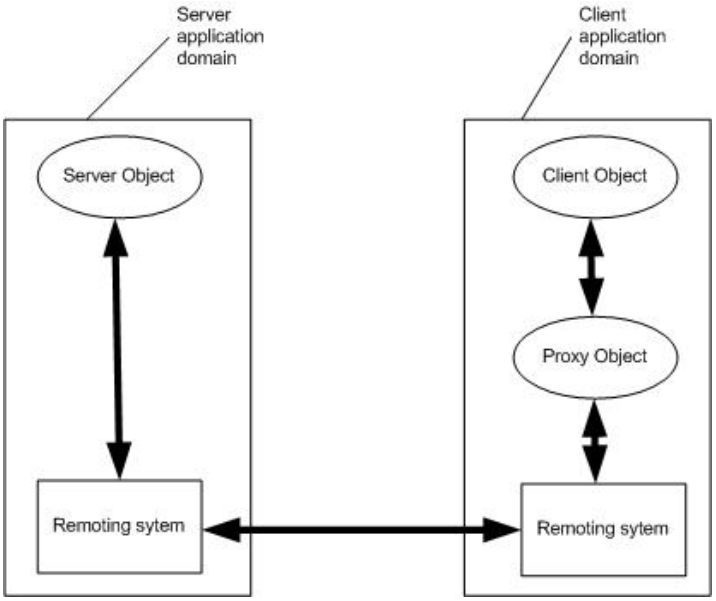
Marshalling

Marshalling is a mechanism of passing objects across the application boundaries by serialization and deserialization.

Communication between two objects takes place through a channel.

Channels

Channels are the objects that transport messages across application boundaries. The remote objects are accessed through channels which are transport protocols which enable communication between the client and a remote object which lies in different application boundaries.



Implementation

Let me directly get into the implementation. The steps involved in implementing remoting are as follows. Create a class which will act as a remote object in the server. This class should be derived from `MarshalByRefObject`. Objects created from this class can be accessed from the client via the reference which is passed to the client. This reference is used by a proxy object which pretends as the remote object.

Collapse | Copy Code

```
using System;
using System.Diagnostics;

namespace Sample.RemoteObject
{
    /// <span class="code-SummaryComment"><summary></span>
    /// Used as the remote component which will be accessed by the clients
    /// <span class="code-SummaryComment"></summary></span>
    public class RemoteCalculator:MarshalByRefObject
    {
        public RemoteCalculator()
        {
            //Check whether a source exists with the name given
            if (!EventLog.SourceExists("RemoteObject"))
            {
                //If not create a source
                EventLog.CreateEventSource("RemoteObject", "Application");
            }
        }
        public int Add(int a, int b)
        {
            //Log the information to keep track of the calls made from the
            //client
            EventLog.WriteEntry("RemoteObject",
                        String.Format("Addition of {0} and {1} ", a, b));
            return a + b;
        }

        public int Multiply(int a, int b)
```

```
        {
            //Log the information to keep track of the calls made from the
            //client
            EventLog.WriteEntry("RemoteObject",
                                String.Format("Addition of {0} and {1} ", a, b));
            return a * b;
        }
    }
}
```

Create the Server application which listens to the request of clients in another project. And make a reference to the DLL which implements the Remote object (`RemoteCalculator`) in the application.

Collapse | Copy Code

```
//Registering the TCP channel
_serverChannel = new TcpChannel(_port);
ChannelServices.RegisterChannel(_serverChannel);
//Registering the server component as a server activated object (SOA)
RemotingConfiguration.RegisterWellKnownServiceType(typeof(RemoteCalculator),
                    "RemoteCalculator", WellKnownObjectMode.Singleton);
btnListen.Text = "Stop Listening";
```

Create a channel where the client requests will be received. The channel which is created has to be registered. Remote objects can be classified as:

- Server activated objects (SAO)
- Client activated objects (CAO)

Server Activated Objects

Server activated objects are remote objects where the lifetime of the objects is controlled by the server. Two possible activation modes of a SAO are **SingleCall** and **Singleton**.

SingleCall activation mode is used for the purpose of responding to just one client request.

Singleton is used where one instance of server remote object is accessed by many clients.

Client Activated Objects

Client activated objects are objects where the lifetime is directly controlled by the client.

After registering the channel, the remotable object has to be registered in the server using `RemotingConfiguration.RegisterWellKnownServiceType` method.

In the sample application, addition and multiplication calculation is done by the remote object residing in the server which is accessed from the client using remoting.

Collapse | Copy Code

```
/// <span class="code-SummaryComment"><summary></span>
/// Addition and Multiplication is done at the server through remoting
/// <span class="code-SummaryComment"></summary></span>
private string Calculate(Operation operation,int number1, int number2)
{
    int result ;
    //check whether channel is created
    if (_clientChannel == null)
    {
        //Create the channel
        _clientChannel = new TcpChannel();
        //Register  the channel
        ChannelServices.RegisterChannel(_clientChannel);
     }
     //Create a proxy object to access the remote calculator
     _remoteCalculator =
           (RemoteCalculator)Activator.GetObject(typeof(RemoteCalculator),
               "tcp://" + _server + ":" + _port + "/RemoteCalculator");
     if (operation.Equals(Operation.Addition))
```

```
    {
        result = _remoteCalculator.Add(number1, number2);
    }
    else
    {
        result = _remoteCalculator.Multiply(number1, number2);
    }
    return Convert.ToString(result);
}
```

On the client side, a channel has to be created to access the remote object. A proxy object is created in the client side using:

Collapse | Copy Code

```
remoteCalculator =
(RemoteCalculator)Activator.GetObject(typeof(RemoteCalculator),
    "tcp://" + _server + ":" + _port + "/RemoteCalculator");
```

`Activator.GetObject` method is used to create a proxy for SAO. In case of CAO, `Activator.CreateInstance` method is used. In the sample application, the server name and the port are picked from the configuration (*App.Config*) file so that the server name and port number can be changed easily.

Collapse | Copy Code

```
//Get the server name and server port from the configuration file
_server = ConfigurationManager.AppSettings["Server"];
_port = ConfigurationManager.AppSettings["ServerPort"];
```

Each call to remote object is tracked in the Event Viewer.You can check the logged information in the eventviewer by using the command `eventvwr.msc` in the command prompt (*cmd.exe*) or by exploring the path "*Control Panel/Administrative Tools/Event Viewer*".Check the logged information in the "Application" Log. You will find that each call to a remote object is logged.

I hope this simple sample application gives you an understanding of Remoting.

License

This article, along with any associated source code and files, is licensed under The Code Project Open License (CPOL)

Share

EMAIL

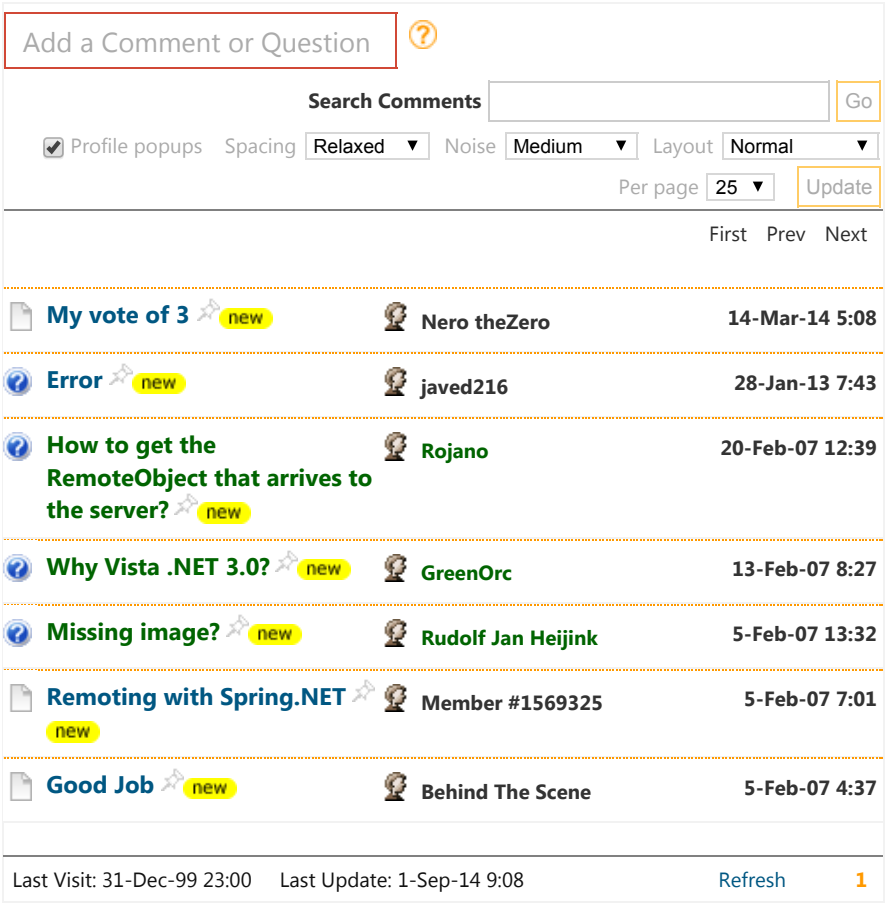
About the Author

Manoj Kumar Raju
Web Developer
India

No Biography provided

Article Top

Comments and Discussions



General News Suggestion Question Bug Answer Joke Rant Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.

Permalink | Advertise | Privacy | Mobile
Web02 | 2.8.140827.1 | Last Updated 14 Feb 2007

Layout: fixed | fluid

Article Copyright 2007 by Manoj Kumar Raju
Everything else Copyright © CodeProject, 1999-2014
Terms of Service