Remoting is a framework built into Common Language Runtime (CLR) in order to provide developers classes to build distributed applications and wide range of network services. Remoting provides various features such as Object Passing, Proxy Objects, Activation, Stateless and Stateful Object, Lease Based LifeTime and Hosting of Objects in IIS. I'm not going into detail of these features because it will take 3 to 4 tutorials.  
  
Here I'm presenting a simple client/server based application in order to provide you easy and fast hands on Remoting.

**Remoting Object**

This is the object to be remotely accessed by network applications. The object to be accessed remotely must be derived by MarshalByRefObject and all the objects passed by value must be serializable.

using System;  
using System.Runtime.Remoting;  
using System.Runtime.Remoting.Channels;  
using System.Runtime.Remoting.Channels.Tcp;  
namespace RemotingSamples  
{  
public class RemoteObject : MarshalByRefObject  
{  
///constructor  
public RemoteObject()  
{  
Console.writeline("Remote object activated");  
}  
///return message reply  
public String ReplyMessage(String msg)  
{  
Console.WriteLine("Client : "+msg);//print given message on console   
return "Server : Yeah! I'm here";  
}  
}  
}

The remote object must be compiled as follows to generate remoteobject.dll which is used to generate server and client executable.

csc /t:library /debug /r:System.Runtime.Remoting.dll remoteobject.cs

**The Server**

This is the server application used to register remote object to be access by client application. First, of all choose channel to use and register it, supported channels are HTTP, TCP and SMTP. I have used here TCP. Than register the remote object specifying its type.

using System;  
using System.Runtime.Remoting;  
using System.Runtime.Remoting.Channels;  
using System.Runtime.Remoting.Channels.Tcp;  
namespace RemotingSamples  
{  
public class Server  
{  
///constructor  
public Server()  
{  
}  
///main method  
public static int Main(string [] args)  
{  
//select channel to communicate  
TcpChannel chan = new TcpChannel(8085);  
ChannelServices.RegisterChannel(chan); //register channel  
//register remote object  
RemotingConfiguration.RegisterWellKnownServiceType(Type.GetType("RemotingSamples.RemoteObject,object"),"RemotingServer",WellKnownObjectMode.SingleCall);  
//inform console  
Console.WriteLine("Server Activated");  
return 0;  
}  
}  
}

The server must be compiled as follows to produce server.exe.

csc /debug /r:remoteobject.dll /r:System.Runtime.Remoting.dll server.cs

**The Client**

This is the client application and it will call remote object method. First, of all client must select the channel on which the remote object is available, activate the remote object and than call proxy's object method return by remote object activation.

using System;  
using System.Runtime.Remoting;  
using System.Runtime.Remoting.Channels;  
using System.Runtime.Remoting.Channels.Tcp;  
using RemotingSamples;  
namespace RemotingSamples  
{  
public class Client  
{  
///constructor  
public Client()  
{  
}  
///main method  
public static int Main(string [] args)  
{  
//select channel to communicate with server  
TcpChannel chan = new TcpChannel();  
ChannelServices.RegisterChannel(chan);  
RemoteObject remObject = (RemoteObject)Activator.GetObject(typeofRemotingSamples.RemoteObject),"tcp://localhost:8085/RemotingServer");  
if (remObject==null)  
Console.WriteLine("cannot locate server");  
else  
remObject.ReplyMessage("You there?");  
return 0;  
}  
}  
}

The client must be compiled as follows in order to produce client.exe

csc /debug /r:remoteobject.dll /r:System.Runtime.Remoting.dll client.cs