

Introduction au .NET Remoting

(Avec code source à télécharger gratuitement en VB.NET et C#)

Mis à jour le 06/07/2003

Par Elise Dupont

Droit de diffusion:

L'ensemble ou partie de ce document ainsi que le code mis à disposition, ne peut être diffusé sur d'autres sites Web sans l'autorisation préalable de son créateur.

Avant Propos :

Ce document a pour but de vous donner un premier aperçu de la technologie Remoting. La solution complète avec le code plus détaillé en VB.NET peut être trouvée **en téléchargement ici** ainsi qu'une version C# **à télécharger ici** (40 Ko environs chacun), et c'est gratuit !)

Sommaire:

1. Qu'est-ce que .NET Remoting ?
2. Les bases
3. Votre première application en remoting

1. Qu'est-ce que .NET Remoting ?

Pour comprendre le concept du remoting, il faut déjà comprendre celui d'une application distribuée.

La définition la plus simple est la suivante :

Application qui, suivant les principes de l'architecture client-serveur, peut tourner de façon transparente sur plusieurs ordinateurs reliés en réseau.

Le remoting .NET est la dernière technologie en matière d'applications distribuées. Je m'explique :

Dans le domaine des applications client serveur, il y a eu une constante évolution. Parmi les "anciens" concepts on peut citer CORBA, COM+, Java RMI ou plus dernièrement les Web Services.

A la différence des Web Services, le .NET remoting n'est pas obligé d'utiliser le protocole SOAP pour communiquer. Il peut utiliser n'importe quel type de protocole.

L'avantage du .NET remoting réside dans la flexibilité du transfert (HTTP ou TCP sont proposés d'office par .NET), de l'encodage (SOAP ou en mode binaire qui sont supportés par défaut). Par contre si l'on recherche avant tout l'interopérabilité, on se tournera plutôt vers les Web Services.

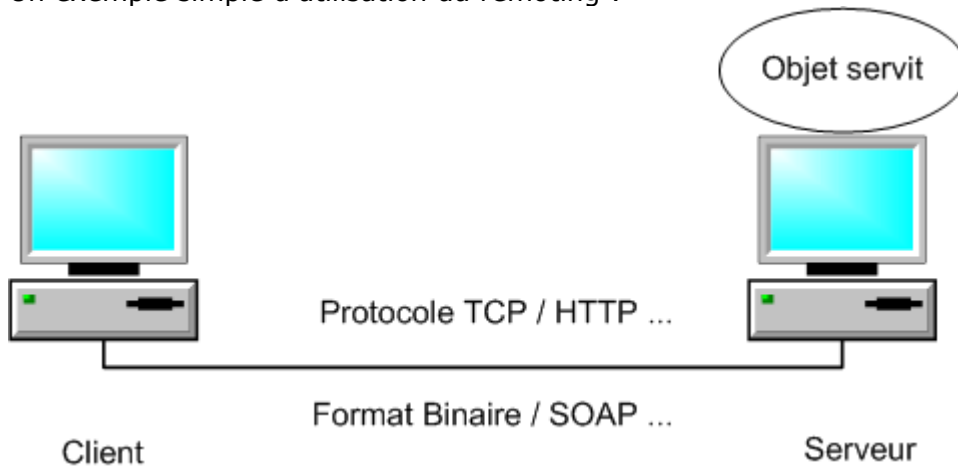
L'utilisation principale en remoting sera donc la distribution d'objets .NET.

Il n'y a pas qu'une seule façon de faire du remoting aussi le choix final d'implémentation dépendra t-il de vos besoins. Ici nous traiterons des cas simples.

Si vous êtes déjà un développeur .NET remoting avertit, je vous conseille le livre

"Advanced .NET Remoting" d'Ingo Rammer (en anglais uniquement à ma connaissance), qui traite le sujet plus en profondeur. Il existe une édition C# et une autre VB.NET.

Un exemple simple d'utilisation du remoting :



Supposons que vous centralisiez toutes les données sur une même base, et que ces données soient utilisées par diverses applications clients qui ne doivent pas avoir un accès direct à la base. Le remoting peut vous aider à servir les objets dont les applications clients ont besoin. Ainsi ce sera la partie serveur du remoting qui aura accès à la base de données et servira les objets aux différentes applications clients.

2. Les bases :

Le plus dur au départ, c'est de se familiariser avec le système de fonctionnement.

2.1 Les éléments :

Un système remoting comporte 3 éléments :

- un serveur
- un client
- une interface qui sera commune au client et au serveur : c'est dans cette interface que l'on définit les types et les méthodes qui seront délivrées par le serveur.

Sans cette interface, la communication entre le client et le serveur est impossible.

2.2 Les Objets :

En remoting, on ne peut communiquer que des objets dits "Sérialisables". Autrement dit, ceux qui implémentent l'interface ISerializable.

La sérialisation est le processus qui permet de transformer l'objet en un format commun, et communicable sur le réseau.

La définition officielle étant un peu plus obscure :

"Sérialiser : Transformer les informations reçues du canal ou du bus d'entrée-sortie de l'équipement informatique émetteur, sous une forme parallèle, afin de pouvoir les transmettre, sous une forme sérielle, sur le support de télécommunications."

Pour savoir si vous pouvez partager un objet (s'il est sérialisable), il suffit de regarder si la classe est <Serializable>, c'est indiqué dans le détail de la classe dans l'aide .NET

En pratique, la majorité des classes (String, Dataset, DataTable, Double etc...) sont Sérialisables. Pour un peu plus de détails, je vous conseille ce lien qui vous explique plus en

détail les objets "remotables".

On peut "servir" (dans le sens publier, mettre à la disponibilité des clients) un objet de plusieurs façons.

- Par référence (MarshalByRef): Les objets vivent sur le serveur, et le client utilise une référence à ces objets. On peut comparer ça à un pointeur sur une variable mais sur le réseau.

- Par valeur (MarshalByValue): Le client manipule une copie de l'objet qui est sur le serveur. Clients et serveurs ont chacun leur copie et l'utilisent indépendamment.

3. Votre première application en remoting :

Pour suivre ce code et mieux visualiser les concepts du remoting et ce que j'explique ici, je vous conseille de **télécharger** le code exemple que j'ai fait et qui est disponible [ici en VB.NET](#) et [là en C#](#), qui vous aideront à mieux comprendre le remoting.

Créez une nouvelle solution. Dans notre exemple elle s'appellera "PremierRemoting". Nos clients et nos serveurs peuvent être des applications consoles, des applications Windows ou encore tout simplement une ligne que l'on ajoute sur une application web. Peu importe leur forme. Ici, nous allons utiliser des applications Consoles.

- Ajoutez dans votre solution une nouvelle Application Console que vous nommerez "Serveur".
- Ajoutez une nouvelle Application Console que vous nommerez "Client".
- Ajoutez enfin un nouveau projet de type "Class library" que vous nommerez "IRemote". Il s'agit là de notre interface.

Nous allons créer un Objet "Employé" comme ceci :

Visual Basic .NET

```
Public Class Employe
    Public IID As Integer
    Public strNom As String
    Public fSalaire As Double
    Public Sub Augmenter( ByVal dMontant As Double )
        Me.fSalaire += dMontant
        Console.WriteLine( " - Employé Augmenté, nouveau salaire : " +
Me .fSalaire.ToString + " -" )
    End Sub
End Class
```

C#

```
public class Employe
{
    public int IID
    public string strNom
    public double fSalaire
    public void Augmenter( double dMontant )
    {
        this.fSalaire += dMontant;
        Console.WriteLine( " - Employé Augmenté, nouveau salaire : " +
this .fSalaire.ToString + " -" );
    }
}
```

On crée une fonction très simple qui permet d'augmenter le salaire de l'employé. Seul le serveur en aura connaissance.
Ensuite, on ajoute cette fonction dans l'interface afin que le client puisse l'utiliser.

Visual Basic .NET

```
Public Interface IEmployee
    Sub Augmenter( ByVal dMontant As Double )
End Interface
```

C#

```
public interface IEmployee
{
    void Augmenter( double dMontant );
}
```

Puis on crée le serveur :

Ajouter une référence au projet IRemote, et ajouter une référence à la dll .NET qui s'appelle System.Runtime.Remoting.
Cette dll contient toutes les classes nécessaires au remoting.

On crée l'Objet qui sera servi : on crée une classe que l'on appelle EmployeeManager.
Cette classe implémente l'interface que l'on a définie plus haut. Elle doit donc implémenter les méthodes qui sont listées dans l'interface, sinon vous aurez une erreur.
Le code est relativement simple.

Visual Basic .NET

```
Public Class EmployeeManager
    Inherits MarshalByRefObject
    Implements IRemote.IEmployee
    'C'est cette classe que l'on distribue sur le reseau
    Public Overrides Function InitializeLifetimeService() As Object
        'Durée de vie de l'objet : pour "toujours"
        Return Nothing
    End Function
    Public Sub Augmenter( ByVal dMontant As Double ) Implements
    IEmployee.Augmenter
        myEmp.Augmenter(dMontant)
    End Sub
End Class
```

C#

```
public class EmployeeManager : MarshalByRefObject, IRemote.IEmployee
{
    //C'est cette classe que l'on distribue sur le reseau
    public override object InitializeLifetimeService()
    {
        //Durée de vie de l'objet : pour "toujours"
        Return null;
    }
    public void Augmenter( double dMontant )
    {
        myEmp.Augmenter(dMontant);
    }
}
```

C'est cette classe que l'on sert.

Créons une classe `RemotingServer`, avec une méthode qui "démontre" le serveur. Il faut spécifier plusieurs choses dans cette méthode :

- le canal de transmission, et quel protocole il utilise (HTTP, TCP etc...)
- le port du canal (éviter le port 80)
- le mode de transmission des données (il y a deux choix possible : Singleton ou SingleCall)
- Singleton : Il n'y a uniquement qu'une seule instance de l'objet au même moment. On peut leur associer une durée de vie (quelques secondes ou éternel). Après la durée de vie écoulée, le serveur recrée une instance de l'objet à nouveau et la servira aux clients jusqu'à la fin de durée de vie.
- SingleCall : une nouvelle instance de l'objet sera créée à chaque appel et détruite juste après.

Voici le code :

Visual Basic .NET

```
Imports System.Runtime.Remoting
Imports System.Runtime.Remoting.Channels
Imports System.Runtime.Remoting.Channels.Tcp.TcpChannel
Imports IRemote

Public Class RemotingServer
    Public Function Start()
        'Ouvre un canal en mode TCP sur le port '1234'
        Dim chnl As New Channels.Tcp.TcpChannel(1234)
        ChannelServices.RegisterChannel(chnl)
        'Publier notre objet EmployeeManager
        RemotingConfiguration.RegisterWellKnownServiceType( GetType
(EmployeeManager),
        "EmployeeManager" , WellKnownObjectMode.Singleton)
    End Function
End Class
```

C#

```
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp.TcpChannel;
using IRemote;

public class RemotingServer {
    public void Start()
    {
        //Ouvre un canal en mode TCP sur le port '1234'
        TcpChannel chnl = New Channels.Tcp.TcpChannel(1234);
        ChannelServices.RegisterChannel(chnl);
        //Publier notre objet EmployeeManager
        RemotingConfiguration.RegisterWellKnownServiceType( typeof
(EmployeeManager),
        "EmployeeManager" , WellKnownObjectMode.Singleton);
    }
}
```

On spécifie dans la méthode `RegisterWellKnownServiceType` que c'est la classe `EmployeeManager` que l'on sert.

Si vous voulez que votre objet dure pour "toujours", il vous suffit de surcharger une méthode

qui s'appelle InitializeLifetimeService.

En retournant "null" dans cette méthode, la durée de vie est ainsi infinie. Mais vous pouvez aussi mettre une durée spécifique.

Voici la durée de vie normale d'un objet (sans surcharge de la classe InitializeLifetimeService)

Appel	Par défaut	Explication
Premier appel	5 minutes	Le temps de vie initial d'un objet après sa création.
Rappel pendant la durée de vie	2 minutes	Temps de sursis rajouté à l'objet si vous le rappelez pendant qu'il vit encore.

Voilà. Il nous suffit de mettre un code qui démarre le serveur et attends des instructions dans un nouveau module :

Visual Basic .NET

```
Module Start
    Private myServer As Serveur.RemotingServer = New
    Serveur.RemotingServer()
    Public myEmp As Employe = New Employe()
    Sub main()
        Try
            Console.WriteLine( "Serveur : Demarrage ..." )
            myServer.Start()
            Console.WriteLine( "Serveur démarré..." )
            Console.ReadLine()
        Catch ex As Exception
            Console.WriteLine( "Serveur : Erreur d'initialisation !!! " +
ex.Message)
        End Try
    End Sub
End Module
```

C#

```
public Employe myEmp = New Employe();
[STAThread]
static void Main( string [] args)
{
    Serveur.RemotingServer myServer = new Serveur.RemotingServer();
    try
    {
        Console.WriteLine( "Serveur : Demarrage ..." );
        myServer.Start();
        Console.WriteLine( "Serveur démarré..." );
        Console.ReadLine();
    }
}
```

```

    }
    catch (Exception ex )
    {
        Console.WriteLine( "Serveur : Erreur d'initialisation !!! " +
ex.Message);
    }
}

```

Dans le sub main, appelez la méthode RemotingServer.Start. Le tour est joué. Compilez et exécutez le serveur, il tourne.

A présent le client qui consomme le service :

Ajouter une référence au projet IRemote, et ajouter une référence à la dll .NET qui s'appelle System.Runtime.Remoting.

Ouvrez le même canal que le serveur, et prenez une référence à l'objet Servit. Ensuite il vous suffit d'appeler les méthodes par cette référence :

Visual Basic .NET

```

Imports System.Runtime.Remoting
Imports System.Runtime.Remoting.Channels
Imports System.Runtime.Remoting.Channels.Tcp.TcpChannel

Module Client
    Public mgr As IRemote.IEmployee
    Sub Main()
        Dim myChnl As TcpChannel
        Try
            'Enregistrer le canal TCP port 1234
            myChnl = New TcpChannel()
            ChannelServices.RegisterChannel(myChnl)
            Console.WriteLine( "Client: Canal enregistré" )
            'Prendre la référence sur le serveur
            (tcp://nommachine:port/EmployeeManager.soap)
            mgr = Activator.GetObject( GetType (IRemote.IEmployee),
"tcp://localhost:1234/EmployeeManager" )
            'Verifier que la référence a été acquise
            If IsNothing(mgr) Then
                Console.WriteLine( "Client: SERVEUR en vacances ...essayez
plus tard " )
            Else
                Console.WriteLine( "Client: Reference au serveur acquise " )
                'On peut a present appeler les méthodes de la façon
mgr.maMethode
                mgr.Augmenter(1000)         'Belle augmentation
                Console.ReadLine()         'Attendre les instructions sur la
console...
            End If
        Catch ex As Exception
            Console.WriteLine( "ERREUR :" & ex.Message)
        End Try
    End Sub
End Module

```

C#

```

using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp.TcpChannel;

class Client
{

```

```

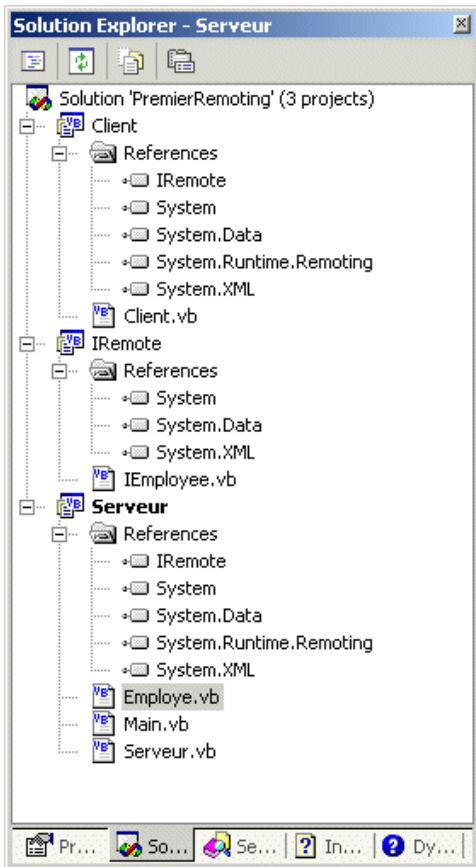
static void Main( string [] args)
    TcpChannel myChnl = new TcpChannel();
    try
    {
        //Enregistrer le canal TCP port 1234
        myChnl = New TcpChannel();
        ChannelServices.RegisterChannel(myChnl);
        Console.WriteLine( "Client: Canal enregistré" );
        //Prendre la référence sur le serveur
        (tcp://nommachine:port/EmployeeManager.soap)
        IRemote.IEmploye mgr = Activator.GetObject( typeof
        (IRemote.IEmployee), "tcp://localhost:1234/EmployeeManager" );
        //Verifier que la reférence a été acquise
        If (mgr == null )
        {
            Console.WriteLine( "Client: SERVEUR en vacances ...essayez
plus tard " );
        }
        else
        {
            Console.WriteLine( "Client: Reference au serveur acquise " );
            //On peut a present appeler les méthodes de la façon
mgr.maMethode
            mgr.Augmenter(1000);    //Belle augmentation
            Console.ReadLine();    //Attendre les instructions sur la
console...
        }
    }
    catch (Exception ex )
    {
        Console.WriteLine( "ERREUR :" & ex.Message);
    }
}

```

Comme vous pouvez le constater, pour les tests, j'ai fait tourné le client et le serveur sur la même machine.

On le voit bien quand je spécifie le serveur "localhost". Mais vous pouvez tester sur deux machines séparées, cela fonctionne aussi.

Voici la structure finale de votre solution :



En lançant le client, on voit bien que le serveur crée une fois l'instance Employé.

```
C:\WINNT\System32\cmd.exe - serveur
C:\Documents and Settings\edupont\My Documents\Visual Studio Projects\PremierRemoting\Serveur\bin>serveur
--- Constructeur Employe : Creation ---
--- Employé ---
ID : 1
Nom : Durand
Salaire : 0
---
Serveur : Demarrage ...
Serveur démarré...
```

Ensuite l'appel des méthodes par le client. On peut voir sur la console du serveur que les appels aux méthodes ont été reçus.

```
C:\WINNT\System32\cmd.exe - serveur
C:\Documents and Settings\edupont\My Documents\Visual Studio Projects\PremierRemoting\Serveur\bin>serveur
-----
- Constructeur Employe : Creation -
-----
- Employé -
- ID : 1
- Nom : Durand
- Salaire : 0
-----
Serveur : Demarrage ...
Serveur démarré...
- Employé Augmenté, nouveau salaire : 8500 -
- Employé Augmenté, nouveau salaire : 9050 -
- Employé Augmenté, nouveau salaire : 9077.15 -
```

Si vous arrêtez le client en appuyant sur une touche et le relancez sans arrêter le serveur, vous verrez qu'il s'agit du même objet employé sur le serveur qui est utilisé depuis le début et qui continue d'être augmenté. Le mode Singleton a donc été prouvé.

```
C:\WINNT\System32\cmd.exe - serveur
oting\Serveur\bin>serveur
-----
- Constructeur Employe : Creation -
-----
- Employé -
- ID : 1
- Nom : Durand
- Salaire : 0
-----
Serveur : Demarrage ...
Serveur démarré...
- Employé Augmenté, nouveau salaire : 8500 -
- Employé Augmenté, nouveau salaire : 9050 -
- Employé Augmenté, nouveau salaire : 9077.15 -
- Employé Augmenté, nouveau salaire : 17577.15 -
- Employé Augmenté, nouveau salaire : 18127.15 -
- Employé Augmenté, nouveau salaire : 18154.3 -
```

Vous avez réalisé votre première application remoting !

Par la suite, je ferais une session avancée pour aller un peu plus dans le détail. Car on peut gérer la sécurité, les événements de client à serveur et bien d'autres choses encore. Si déjà vous avez saisi le système de fonctionnement global du remoting, le reste sera moins difficile.



L'ensemble ou partie de ce document ainsi que le code mis à disposition, ne peut être diffusé ailleurs sans autorisation préalable

elise.dupont@europa.com - www.dotnet-tech.com - 2003