M1
GMIN404 - Février 2013

.NET Remoting

Listing 1 – Share.cs pour exemple Singleton et passage de paramètre

```csharp
using System;
namespace RemotingSamples {
  public class ForwardMe : MarshalByRefObject {
    public void CallMe(String text)
    {
      Console.WriteLine(text);
    }
  }

  public class HelloServer : MarshalByRefObject {
    private int compteur;
    public HelloServer()
    {
      Console.WriteLine("HelloServer activ");
        compteur=0;
    }
    public String HelloMethod(String name,ForwardMe obj)
    {
      obj.CallMe("Message venant du serveur");
      Console.WriteLine("Hello.HelloMethod : {0}", name);
      return "Bonjour " + name;
    }

    public int CountMe()
    {
        compteur++;
        return compteur;
    }
  }
}
```

Listing 2 – Exemple singleton Serveur.cs

```csharp
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
using System.Runtime.Remoting.Channels.Http;

namespace RemotingSamples {
  public class Sample {

    public static int Main(string [] args) {

      TcpChannel chan1 = new TcpChannel(8089);
      ChannelServices.RegisterChannel(chan1, true);
      RemotingConfiguration.RegisterWellKnownServiceType(typeof(HelloServer), "SayHello", WellKnownObjectMode.Singleton);
      System.Console.WriteLine("Appuyez sur <entree> pour sortir...");
      System.Console.ReadLine();
      return 0;
    }
  }
}
```

Listing 3 – Exemple singleton Client.cs

```
using System;
using System.Threading;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
using System.Runtime.Remoting.Channels.Tcp;

namespace RemotingSamples {
  public class Client {

    public bool init = false;
    public static Thread thread1 = null;
    public static Thread thread2 = null;

    public static int Main(string [] args)
    {
        TcpChannel chan = new TcpChannel();
        ChannelServices.RegisterChannel(chan, true);
      Client c = new Client();
      thread1 = new Thread(new ThreadStart(c.RunMe));
      thread2 = new Thread(new ThreadStart(c.RunMe));
      thread1.Start();
      thread2.Start();
      Console.Read();
      return 0;
    }


    public void RunMe()
    {

      if (Thread.CurrentThread == thread1) {

        Console.WriteLine("Ceci est le thread un");
        HelloServer obj = (HelloServer)Activator.GetObject(typeof(HelloServer),
            "tcp://localhost:8089/SayHello");
        for (int i = 0; i < 100; i++) {
          Console.WriteLine(obj.CountMe() + " depuis le thread 1 ");
          Thread.Sleep(0);
        }
      }
      else if (Thread.CurrentThread == thread2) {
        Console.WriteLine("Ceci est le thread deux");
        HelloServer obj = (HelloServer)Activator.GetObject(typeof(HelloServer),
            "tcp://localhost:8089/SayHello");
        for (int i = 0; i < 100; i++) {
          Console.WriteLine(obj.CountMe() + " depuis le thread 2 ");
          Thread.Sleep(0);
        }
      }
    }
  }
}
```

Listing 4 – Exemple passage de référence Server.cs

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
using System.Collections;

namespace RemotingSamples {
  public class Sample {

    public static int Main(string [] args) {

        // on va avoir besoin de passer des ref distantes au serveur
        // on va donc regler le canal, avec le niveau de securite adequate
        // c'est un peu complique ...
        // Plus d'explications : http://msdn.microsoft.com/fr-fr/library/5
            dxse167(v=VS.90).aspx

        // Creating a custom formatter for a TcpChannel sink chain.
        BinaryServerFormatterSinkProvider provider = new
            BinaryServerFormatterSinkProvider();
        provider.TypeFilterLevel = System.Runtime.Serialization.Formatters.
            TypeFilterLevel.Full;
        // Creating the IDictionary to set the port on the channel instance.
        IDictionary props = new Hashtable();
        props["port"] = 8085;
        // Pass the properties for the port setting and the server provider in
            the server chain argument. (Client remains null here.)
        TcpChannel chan = new TcpChannel(props, null, provider);
        // TcpChannel chan = new TcpChannel(8085); et si on faisait juste cette
            ligne au lieu de tout ce qui est au dessus ?

      ChannelServices.RegisterChannel(chan, true);
      Type t = Type.GetType("RemotingSamples.HelloServer,Share");
      RemotingConfiguration.RegisterWellKnownServiceType(Type.GetType("
          RemotingSamples.HelloServer,Share"), "SayHello", WellKnownObjectMode.
          SingleCall);
      System.Console.WriteLine("Appuyez sur <entree> pour sortir...");
      System.Console.ReadLine();
      return 0;
    }
  }
}
```

Listing 5 – Exemple passage de référence Client.cs

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


namespace RemotingSamples
{
    public class Client
    {
        public static int Main(string[] args)
    {
        TcpChannel chan = new TcpChannel(8086);
      ChannelServices.RegisterChannel(chan, true);
      ForwardMe param = new ForwardMe();
      HelloServer obj = (HelloServer)Activator.GetObject(typeof(RemotingSamples.
          HelloServer), "tcp://localhost:8085/SayHello");
      if (obj == null) System.Console.WriteLine("Impossible de trouver le 
          serveur");
      else Console.WriteLine(obj.HelloMethod("Homme des cavernes",param));
      Console.Read();
      return 0;
    }
    }
}
```

Listing 6 – Exemple asynchrone ServiceClass.cs

```
using System;
using System.Runtime.Remoting;

public class ServiceClass : MarshalByRefObject{

  public ServiceClass() {
    Console.WriteLine("ServiceClass created.");
  }

  public string VoidCall(){
    Console.WriteLine("VoidCall called.");
    return "You are calling the void call on the ServiceClass.";
  }

  public int GetServiceCode(){
    return this.GetHashCode();
  }

  public string TimeConsumingRemoteCall(){
    Console.WriteLine("TimeConsumingRemoteCall called.");

    for(int i = 0; i < 20000; i++){
      Console.Write("Counting: " + i.ToString());
      Console.Write("\r");
    }
    return "This is a time-consuming call.";
  }
}
```

Listing 7 – Exemple asynchrone Server.cs

```
using System;
using System.Runtime.Remoting;

public class Server{

  public static void Main(){
    RemotingConfiguration.Configure("server.exe.config", true);
    Console.WriteLine("Waiting...");
    Console.ReadLine();
  }
}
```

Listing 8 – Exemple asynchrone Client.cs

```
using System;
using System.Reflection;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Messaging;
using System.Runtime.Remoting.Channels;
using System.Threading;

public class RemotingDelegates : MarshalByRefObject{


    public static ManualResetEvent e; //Permet d'avertir un ou plusieurs threads en attente qu'un evenement s'est produit

    public delegate string RemoteSyncDelegate();
    public delegate string RemoteAsyncDelegate();

    // This is the call that the AsyncCallBack delegate references.
    [OneWayAttribute]
   public void OurRemoteAsyncCallBack(IAsyncResult ar){
    // ASyncResult encapsule le resultat issu d'un appel asynchrone ;
    // AsyncDelegate permet de recuperer l'objet delegate sur lequel l'appel asynchrone a ete invoque.
    // ligne suivante, on ne fait donc que recuperer dans del le delegate sur lequel l'appel asynchrone a ete effectue
      RemoteAsyncDelegate del = (RemoteAsyncDelegate)((AsyncResult)ar).AsyncDelegate;
      Console.WriteLine("\r\n**SUCCESS**: Result of the remote AsyncCallBack: " + del.EndInvoke(ar) );

        // Signal the thread.
    //Set : evenement signale, les threads en attente peuvent poursuivre
        e.Set();
        return;
   }

    public static void Main(string[] Args){

        // IMPORTANT: .NET Framework remoting does not remote
        // static members. This class must be an instance before
        // the callback from the asynchronous invocation can reach this client.
        RemotingDelegates HandlerInstance = new RemotingDelegates();
        HandlerInstance.Run();
    }

    public void Run(){
        // Enable this and the e.WaitOne call at the bottom if you
        // are going to make more than one asynchronous call.
        e = new ManualResetEvent(false); // false : evt non signale

        Console.WriteLine("Remote synchronous and asynchronous delegates.");
        Console.WriteLine(new String('-',80));
        Console.WriteLine();

        // This is the only thing you must do in a remoting scenario
        // for either synchronous or asynchronous programming
        // configuration.
        RemotingConfiguration.Configure("SyncAsync.exe.config", true);

        // The remaining steps are identical to single-AppDomain programming.
    // Sauf si on veut utiliser la ligne suivante, plus prudente qu'un new
       // ServiceClass obj = (ServiceClass)Activator.GetObject(typeof(ServiceClass), "tcp://localhost:8085/ServiceClass.rem");
        ServiceClass obj = new ServiceClass(); // attention, on recupere un proxy ...

        // This delegate is a remote synchronous delegate.
        RemoteSyncDelegate Remotesyncdel = new RemoteSyncDelegate(obj.VoidCall);

        // When invoked, program execution waits until the method returns.
        // This delegate can be passed to another application domain
        // to be used as a callback to the obj.VoidCall method.
        Console.WriteLine(Remotesyncdel());
        Console.WriteLine("Pause 1");
        Console.Read();
        // This delegate is an asynchronous delegate. Two delegates must
        // be created. The first is the system-defined AsyncCallback
        // delegate, which references the method that the remote type calls
        // back when the remote method is done.

        AsyncCallback RemoteCallback = new AsyncCallback(this.
            OurRemoteAsyncCallBack);

        // Create the delegate to the remote method you want to use
        // asynchronously.
        RemoteAsyncDelegate RemoteDel = new RemoteAsyncDelegate(obj.
            TimeConsumingRemoteCall);

        // Start the method call. Note that execution on this
        // thread continues immediately without waiting for the return of
        // the method call.
        IAsyncResult RemAr = RemoteDel.BeginInvoke(RemoteCallback, null); //
            BeginInvoke est generee : prend d'abord les parametres s'il y en a (
            ici : non, TimeConsumingRemoteCall ne prend pas de param), puis le
            callback, puis un objet qcq, qui peut etre utile par exemple a passer
             des informations d'etat)
        Console.WriteLine("Pause 2");
        Console.Read();
        // If you want to stop execution on this thread to
        // wait for the return from this specific call, retrieve the
        // IAsyncResult returned from the BeginIvoke call, obtain its
        // WaitHandle, and pause the thread, such as the next line:
        // RemAr.AsyncWaitHandle.WaitOne();

        // To wait in general, if, for example, many asynchronous calls
        // have been made and you want notification of any of them, or,
        // like this example, because the application domain can be
        // recycled before the callback can print the result to the
        // console.
        //e.WaitOne();

    // This simulates some other work going on in this thread while the
    // async call has not returned.
    int count = 0;
    while(!RemAr.IsCompleted){
        Console.Write("\rNot completed: " + (++count).ToString());
        // Make sure the callback thread can invoke callback.
        Thread.Sleep(1);
    }
    Console.Read();
     }
}
```

Listing 9 – Config pour serveur asynchrone

```
<configuration>
    <system.runtime.remoting>
        <application>
        <service>
            <wellknown
                type="ServiceClass, ServiceClass"
                mode="Singleton"
                objectUri="ServiceClass.rem"
            />
        </service>
        <channels>
            <channel
                ref="tcp"
                port="8085"
            />
        </channels>
    </application>
  </system.runtime.remoting>
</configuration>
```

Listing 10 – Config pour client asynchrone

```
<configuration>
  <system.runtime.remoting>
    <application>
        <client>
            <wellknown
                type="ServiceClass, ServiceClass"
                url="tcp://localhost:8085/ServiceClass.rem"
            />
        </client>
        <channels>
            <channel
                ref="tcp"
                port="0"
            />
        </channels>
    </application>
  </system.runtime.remoting>
</configuration>
```