

Desarrollo de aplicaciones distribuidas con .NET: Programación distribuida con .NET

Grupo ARCOS

Desarrollo de Aplicaciones Distribuidas

Ingeniería Informática

Universidad Carlos III de Madrid



Contenidos

1. Introducción a la programación en .NET
2. .NET remoting
3. Cola de mensajes

Contenidos

1. **Introducción a la programación en .NET**
2. .NET remoting
3. Cola de mensajes

Programación en .NET

- ▶ .NET ofrece un conjunto de clases comunes a todos los lenguajes
- ▶ Representación consistente
- ▶ Todos los objetos .NET derivan de la clase **Object**

Programación en .NET

Métodos públicos de la clase Object	
Método	Descripción
Equals()	Compara dos objetos y determina si son equivalentes.
ReferenceEquals()	Compara dos referencias de objetos y comprueba si hacen referencia al mismo elemento.
GetHashCode()	Obtiene el código hash del objeto.
GetType()	Obtiene el type del objeto.
ToString()	Obtiene una representación en String del objeto.

Ejemplo en C#: ej1.cs

```
using System;
namespace Cpm
{
    class CPMModel
    {
        public static void Main( )
        {
            CPMModel c = new CPMModel();

            // Test for self equivalence
            Console.WriteLine("Equivalence:\t" + c.Equals(c));

            // Get the hash code from this object
            Console.WriteLine("Object hash:\t" + c.GetHashCode());

            // Use the type to obtain method information
            Console.WriteLine("Object method:\t"+ c.GetType().GetMethods()[1]);

            // Convert the object to a string
            Console.WriteLine("Object dump:\t" + c.ToString());
        }
    }
}
```

Compilación y ejecución

MS Windows XP/Vista/7 con .Net Framework 3.5

▶ Compilación:

```
C:\temp> csc ej1.cs
```

▶ Ejecución:

```
C:\Temp> ej1  
Equivalence:      True  
Object hash:      58225482  
Object method:    System.String ToString()  
Object dump:      Cpm.CPModel
```

Programación en .NET

Algunos espacios de nombres y clases en .NET	
Espacio de nombre	Descripción
System	Incluye clases básicas: Object, Char, String, Array y Exception. Algunas más avanzadas: GC y AppDomain.
System.IO	Clases para la manipulación de datastreams y sistemas de ficheros: FileStream, MemoryStream, Path y Directory.
System.Collections	Clases para el manejo de conjuntos de objetos: ArrayList, DictionaryBase, Hashtable, Queue y Stack.
System.Threading	Clases con soporte multithread: Thread, ThreadPool, Mutex y AutoResetEvent.
System.Reflection	Clases que dan soporte a la inspección de tipo, dynamic y binding: Assembly, Module y MethodInfo.
System.Security	Soporte de seguridad: Cryptography, Permissions, Police y Principal.
System.Net	Incluye clases para programación en red: IPAddress, Dns y HttpWebRequest.
System.Data	
System.Web.Services	

Programación en .NET

- ▶ Todos los lenguajes .NET deben soportar:
 - ▶ Espacio de nombres.
 - ▶ Interfaces.
 - ▶ Encapsulación.
 - ▶ Herencia.
 - ▶ Polimorfismos.
 - ▶ Manejo de excepciones.

Common Type System (CTS) [1 / 2]

- ▶ Define las reglas que han de seguir las definiciones de tipos para que el CLR las acepte.
- ▶ Algunas de las principales reglas son:
 - ▶ Cada tipo de dato puede constar de cero o más miembros.
Cada uno de estos miembros puede ser un campo, un método, una propiedad o un evento.
 - ▶ No puede haber herencia múltiple, y todo tipo de dato ha de heredar directa o indirectamente de `System.Object`.
 - ▶ **Tipos por valor, por referencia, *boxing* y *unboxing***
 - ▶ Etc..

Common Type System (CTS) [2/2]

- ▶ Categoría de tipos:
 - ▶ **Tipos por valor:**
 - ▶ Contiene directamente el dato (con memoria para él)
 - ▶ **Tipo por referencia:**
 - ▶ Almacenan una referencia a la dirección en memoria del valor (~puntero)

- ▶ Conversiones entre categoría de tipos:

- ▶ **Boxing:**

- ▶ Conversión de un tipo por valor a un tipo por referencia

```
Int32 x = 10;  
object o = x; // Implicit boxing
```

- ▶ **Unboxing :**

- ▶ Conversión de un tipo por referencia a un tipo por valor

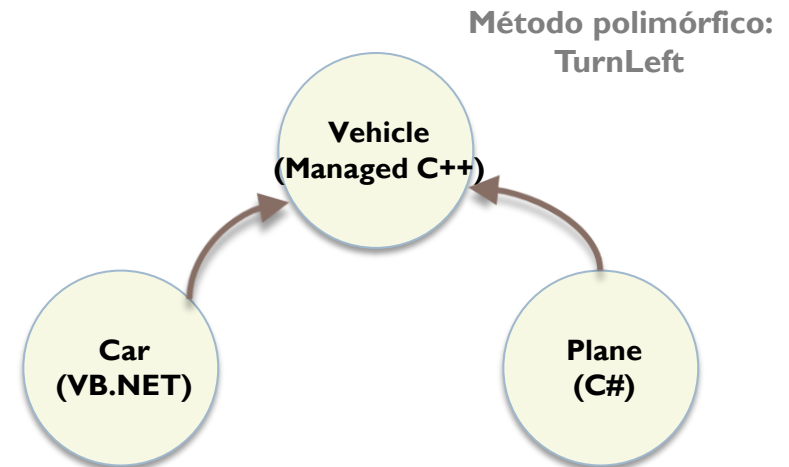
```
Int32 x = 15;  
object o = x; // Implicit Boxing  
x = (int)o;   // Explicit Unboxing
```

Common Language Specification (CLS)

- ▶ Define las reglas que un lenguaje gestionado ha de cumplir para ser accedido desde otro lenguaje gestionado.
- ▶ Algunas de las principales reglas son:
 - ▶ Los **tipos de datos básicos** admitidos son: bool, char, byte, short, int, long, float, double, string y object
 - ▶ En un mismo ámbito **no se pueden definir varios identificadores** cuyos nombres **sólo difieran en la capitalización** usada.
 - ▶ Las excepciones han de derivar de **System.Exception**, los delegados de **System.Delegate**, las enumeraciones de **System.Enum**, y los tipos por valor que no sean enumeraciones de **System.ValueType**.
 - ▶ Etc.

Ejemplo de uso de varios lenguajes

- ▶ **Diseño informal del ejemplo:**
 - ▶ Creación del espacio de nombres **Lang** que encapsula interface **ISteering**
 - ▶ Clase **Vehicle**: clase abstracta que implementa **ISteering**
 - ▶ Clase **Car**: derivada de **Vehicle**
 - ▶ Clase **Plane**: derivada de **Vehicle**
- ▶ **Objetivos del ejemplo:**
 - ▶ Mostrar herencia entre objetos
 - ▶ Manejo de excepciones



Ejemplo: vehicle.cpp

```
#using <mscorlib.dll>
using namespace System;

namespace Lang
{
    __interface ISteering
    {
        void TurnLeft( );
        void TurnRight( );
    };

    class Vehicle : public ISteering
    {
    public:
        void TurnLeft() { Console::WriteLine("Vehicle turns left."); }
        void TurnRight() { Console::WriteLine("Vehicle turns right."); }
        virtual void ApplyBrakes( ) = 0;
    };
}
```

Compilación

► Compilación:

```
C:\temp> cl /clr /c vehicle.cpp  
C:\temp> link -dll /out:vehicle.dll vehicle.obj
```

Generar DLL

Nombre
de la DLL

Ejemplo: car.vb

```
Imports System
```

```
Public Class Car Inherits Vehicle
```

```
    Overrides Public Sub TurnLeft()
```

```
        Console.WriteLine("Car turns left <-")
```

```
    End Sub
```

```
    Overrides Public Sub TurnRight()
```

```
        Console.WriteLine("Car turns right ->")
```

```
    End Sub
```

```
    Overrides Public Sub ApplyBrakes()
```

```
        Console.WriteLine("Car trying to stop...")
```

```
        throw new Exception("Brake failure! :<")
```

```
    End Sub
```

```
End Class
```


Compilación

► Compilación:

```
C:\temp> vbc /r:vehicle.dll /t:library /out:car.dll car.vb
```

Referencia a
otra DLL usada

Generar DLL

Nombre
de la DLL

Ejemplo: plane.cs

```
using System;

public class Plane : Vehicle
{
    override public void TurnLeft()
    {
        Console.WriteLine("Plane turns left <-");
    }

    override public void TurnRight()
    {
        Console.WriteLine("Plane turns right ->");
    }

    override public void ApplyBrakes()
    {
        Console.WriteLine("Air brakes being used...");
    }
}
```

Compilación

► Compilación:

```
C:\temp> csc /r:vehicle.dll /t:library /out:plane.dll plane.cs
```

Referencia a
otra DLL usada

Generar DLL

Nombre
de la DLL

Ejemplo: drive.jsl

```
class TestDrive
{
    public static void main ()
    {
        Vehicle v = null;

        try
        {
            Plane p = new Plane();
            v = p;
            v.TurnLeft();
            v.ApplyBrakes();

            Car c = new Car();
            v = c;
            v.TurnLeft() ;
            v.ApplyBrakes(); // Excepción
        }
        catch (System.Exception e)
        {
            System.Console.WriteLine(e.ToString());
        }
    }
}
```



Compilación

► Compilación:

```
C:\temp> vjc /r:vehicle.dll;car.dll;plane.dll /t:exe /out:drive.exe drive.js1
```



Referencias a otras
DLL usadas

Generar
ejecutable

Contenidos

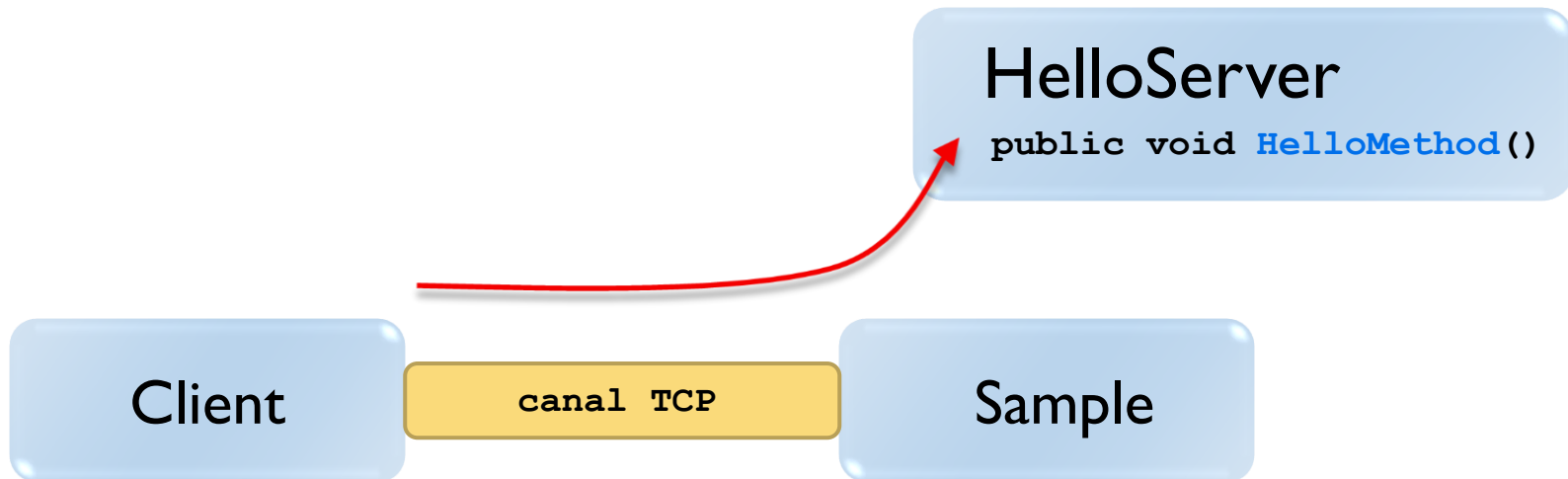
1. Introducción a la programación en .NET
2. .NET remoting
3. Cola de mensajes

.NET *remoting*

- ▶ **.NET *remoting* es definido como:**
 - ▶ “...Microsoft .NET *Remoting* ofrece un *framework* rico y extensible para **objetos residentes** en **diferentes dominios de aplicación** (*AppDomains*), en **diferentes procesos**, e incluso en **diferentes máquinas** para comunicarse unos con otros sin problemas.
 - ▶ “... .NET *Remoting* **ofrece un modelo de programación y entorno de ejecución** potente pero simple para hacer estas interacciones de forma transparente...”

Ejemplo 1: TCP *channel* en .NET

- ▶ Invocación de `HelloMethod()` de `HelloServer`, ofrecido por `Sample`, desde `Client` a través de un canal TCP



object.cs

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

namespace RemotingSamples
{
    public class HelloServer : MarshalByRefObject
    {
        public HelloServer () {
            Console.WriteLine("HelloServer activated");
        }

        public String HelloMethod (String name) {
            Console.WriteLine("Hello.HelloMethod : {0}", name);
            return "Hi there " + name;
        }
    }
}
```

server.cs

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

namespace RemotingSamples {
    public class Sample {
        public static int Main (string [] args) {
            TcpChannel chan = new TcpChannel(9000);
            ChannelServices.RegisterChannel(chan, false);

            RemotingConfiguration.RegisterWellKnownServiceType(
                typeof(RemotingSamples.HelloServer),
                "SayHello",
                WellKnownObjectMode.Singleton
            );
            System.Console.WriteLine("Hit <enter> to exit...");
            System.Console.ReadLine();
            return 0;
        }
    }
}
```

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();
            ChannelServices.RegisterChannel(chan, false);

            HelloServer obj = (HelloServer) Activator.GetObject(
                typeof(RemotingSamples.HelloServer),
                "tcp://localhost:9000/SayHello"
            );
            if (obj == null)
                Console.WriteLine("Could not locate server");
            else Console.WriteLine(obj.HelloMethod("Caveman"));
            return 0;
        }
    }
}
```

Compilación

MS Windows XP/Vista/7 con .Net Framework 3.5

▶ Objeto:

```
C:\Temp> csc /debug+ /target:library →  
→ /out:object.dll object.cs
```

▶ Servidor:

```
C:\Temp> csc /debug+ /r:object.dll →  
→ /r:System.Runtime.Remoting.dll /out:server.exe server.cs
```

▶ Cliente:

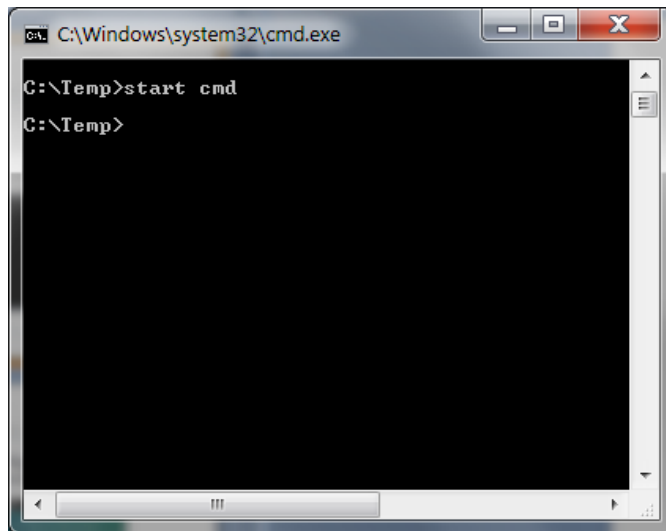
```
C:\Temp> csc /debug+ /r:object.dll →  
→ /r:System.Runtime.Remoting.dll  
→ /r:server.exe /out:client.exe client.cs
```

Ejecución

MS Windows XP/Vista/7 con .Net Framework 3.5

- ▶ Abrir una segunda ventana para el servidor:

```
C:\Temp> start cmd
```



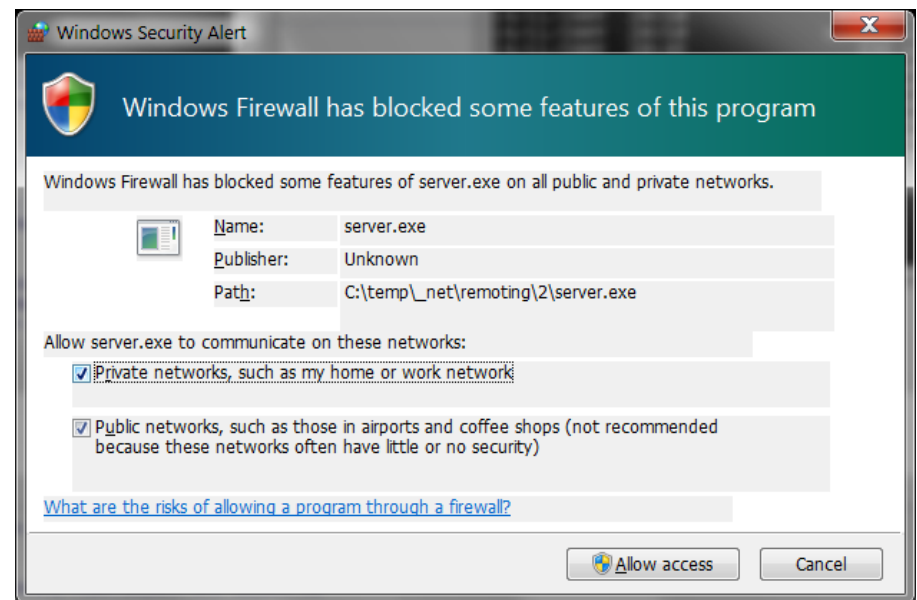
Ejecución

MS Windows XP/Vista/7 con .Net Framework 3.5

- ▶ Ejecutar el servidor en la segunda ventana:

```
C:\Temp> server.exe  
Hit <enter> to exit...
```

- ▶ Puede que haya que configurar el firewall para permitir las comunicaciones a y desde servidor.exe



Ejecución

MS Windows XP/Vista/7 con .Net Framework 3.5

- ▶ Ejecutar el cliente en la primera ventana:

```
C:\Temp> cliente.exe  
Hi there Caveman  
  
C:\Temp>
```

Ejecución

MS Windows XP/Vista/7 con .Net Framework 3.5

- ▶ Ver salida en el servidor y finalizar la ejecución:

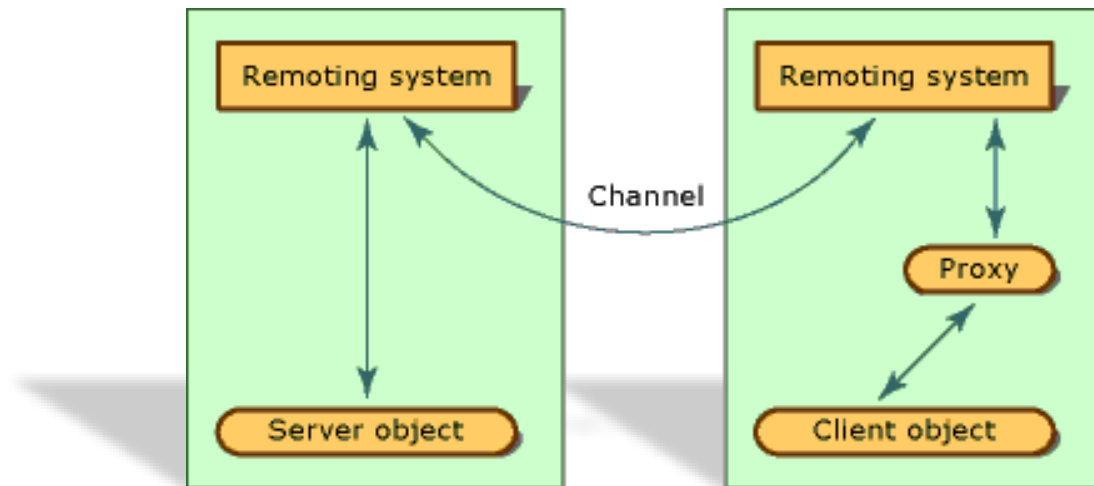
```
C:\Temp> server.exe  
Hit <enter> to exit...  
HelloServer activated  
Hello.HelloMethod : Caveman
```



```
C:\Temp>
```

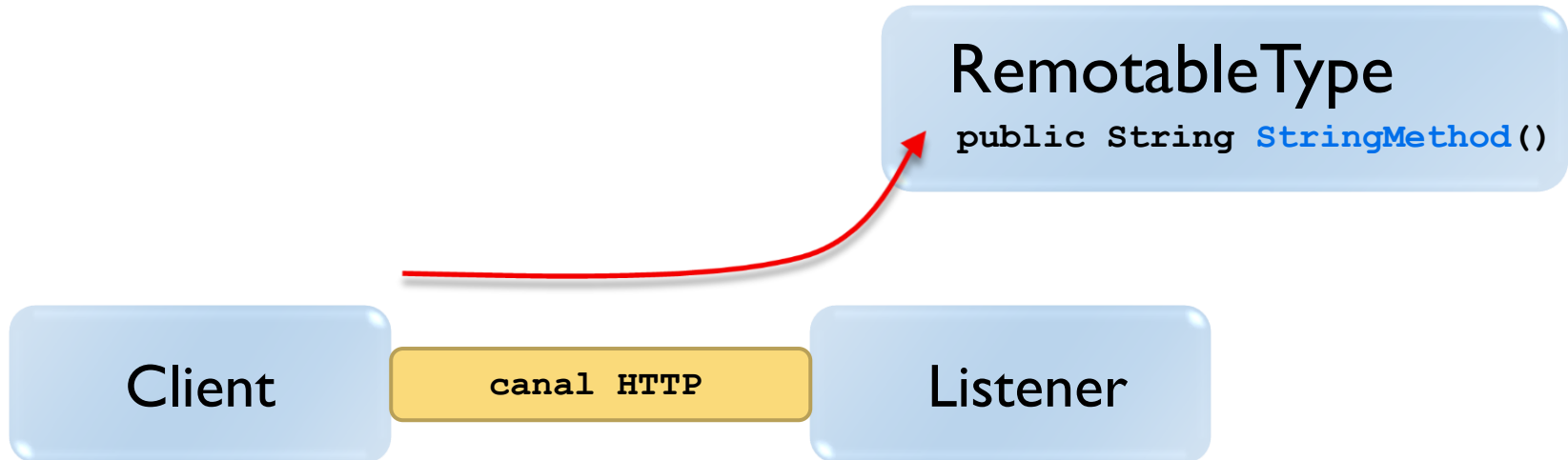

.NET *remoting*

- ▶ **Arquitectura básica de .NET *remoting*:**
 - ▶ Instancia de objeto remoto
 - ▶ *Proxy* cliente
 - ▶ *Remoting system*
 - ▶ Canal



Ejemplo 2: HTTP *channel* en .NET

- ▶ Invocación de `StringMethod()` de `RemotableType` ofrecido por `Listener`, desde `Client` a través de un canal HTTP



1.- Creación de los objetos distribuidos

- ▶ Deben derivar de **MarshalByRefObject**:

```
using System;

public class RemovableType : MarshalByRefObject
{
    private string _internalString = "Esto es un ejemplo.";
    public string StringMethod () { return _internalString; }
}
```

RemovableType.cs

- ▶ **Compilación:**

```
C:\Temp> csc /noconfig /t:library RemovableType.cs
```

2.- Construcción de un servidor

- ▶ Servidor especificado mediante fichero de configuración:

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

public class Listener {
    public static void Main () {
        RemotingConfiguration.Configure("Listener.exe.config",false);
        Console.WriteLine("Esperando solicitudes. Presione Enter para finalizar...");
        Console.ReadLine();
    }
}
```

Listener.cs

- ▶ Compilación:

```
C:\Temp> csc /noconfig /r:RemotableType.dll Listener.cs
```

3.- Fichero de configuración del servidor

Listener.exe.config

```
<configuration>
  <system.runtime.remoting>
    <application>
      <service>
        <wellknown
          mode="Singleton"
          type="RemovableType, RemovableType"
          objectUri="RemovableType.rem"
        />
      </service>
      <channels>
        <channel ref="http" port="8989"/>
      </channels>
    </application>
  </system.runtime.remoting>
</configuration>
```



4.- Construcción del cliente

- ▶ El cliente ha de registrarse por cada objeto remoto:

```
Client.cs
using System;
using System.Runtime.Remoting;

public class Client {
    public static void Main () {
        RemotingConfiguration.Configure("Client.exe.config",false);
        RemotableType remoteObject = new RemotableType();
        Console.WriteLine(remoteObject.StringMethod());
    }
}
```

- ▶ Compilación:

```
C:\Temp> csc /noconfig /r:RemotableType.dll Client.cs
```

5.- Fichero de configuración del cliente

Client.exe.config

```
<configuration>
  <system.runtime.remoting>
    <application>
      <client>
        <wellknown
          type="RemotableType, RemotableType"
          url="http://localhost:8989/RemotableType.rem"
        />
      </client>
    </application>
  </system.runtime.remoting>
</configuration>
```

Resumen del proceso de compilación

MS Windows XP/Vista/7 con .Net Framework 3.5

▶ Objeto:

```
C:\Temp> csc /noconfig /t:library RemotableType.cs
```

▶ Servidor:

```
C:\Temp> csc /noconfig /r:RemotableType.dll Listener.cs
```

▶ Cliente:

```
C:\Temp> csc /noconfig /r:RemotableType.dll Client.cs
```


Resumen del proceso de ejecución (1/2)

MS Windows XP/Vista/7 con .Net Framework 3.5

- ▶ Abrir una segunda ventana para el servidor:

```
C:\Temp> start cmd
```

- ▶ Ejecutar el servidor en la segunda ventana:

- ▶ Puede que haya que configurar el firewall para permitir las comunicaciones a/desde servidor.exe

```
C:\Temp> Listener.exe  
Esperando solicitudes. Presione Enter para finalizar...
```

Resumen del proceso de ejecución (2/2)

MS Windows XP/Vista/7 con .Net Framework 3.5

- ▶ Ejecutar el cliente en la primera ventana:

```
C:\Temp> client.exe  
Esto es un ejemplo.
```

```
C:\Temp>
```

- ▶ Ver salida en el servidor y finalizar la ejecución:

```
C:\Temp> server.exe  
Esperando solicitudes. Presione Enter para finalizar...
```



```
C:\Temp>
```

Detalles de .NET *remoting*

- ▶ Limitaciones en la publicación de objetos
- ▶ Tipo de canales
- ▶ Modos de activación
- ▶ *Application Domains*

Limitaciones en la publicación de objetos

- ▶ Tipos de objetos que no pueden ser publicados:
 1. Miembros estáticos.
 2. *Instance fields*.
 3. Métodos privados.
 4. *Delegates*.
 5. Sobrecarga de métodos:
GetHashCode, *Equals* (versión estática) y *MemberwiseClone*.
Sí pueden ejecutarse como remotos: *Equals* y *ToString*

.NET *remoting*: canales

- ▶ Son **objetos que transportan mensajes** entre aplicaciones.
- ▶ Clientes se pueden comunicar con objetos remotos mediante cualquier canal registrado (por parte del servidor).
- ▶ En el caso de *callback* el cliente debe registrar un canal.
- ▶ Los canales **son multi-hilo**: dan soporte a múltiples conexiones simultáneas.
- ▶ **Distintos tipos: *TcpChannel, HttpChannel, etc.***
- ▶ Muchas funcionalidades: nombre, prioridad, *timeout*, etc.

.NET *remoting*: canales

- ▶ Principales tipos de canales:
 - ▶ *TcpChannel*
 - ▶ *BinaryFormatter Class*
 - ▶ Alto rendimiento
 - ▶ Puede originar problemas con *firewalls*
 - ▶ *HttpChannel*
 - ▶ *SoapFormatter Class*
 - ▶ Menor rendimiento (tamaño de paquetes mayores)
 - ▶ Bajo riesgo de problemas de encaminamiento

.NET *remoting*: canales

- ▶ Otros tipos de canales:
 - ▶ *Genuine Channels*: Canal bidireccional TCP, canal HTTP mejorado, canal UDP y canal basado en memoria compartida
 - ▶ *Jabber Channel*: Basado en el protocolo Jabber XML
 - ▶ *MSMQ Channel*: Basado en el *Microsoft Message Queue Channel*
 - ▶ *Named piped Channel*: basado en *pipes* nombrados
 - ▶ *Secure TCP Channel*: Basado en encriptación RSA
 - ▶ *SMTP Channel*: Utiliza el protocolo SMTP (e-mail)
 - ▶ *TCPEX*: Canal bidireccional TCP

Ejemplo 3: ejemplo 2 con TCP *channel*

- ▶ Cambio del canal de comunicaciones:

- ▶ Fichero `client.exe.config`:

```
<wellknown
  type="RemotableType, RemotableType"
  url="tcp://localhost:8989/RemotableType.rem"
/>
```

- ▶ Fichero `Listener.exe.config`:

```
<channel ref="tcp" port="8989"/>
```


.NET *remoting*: modos de activación

- ▶ **Modos de activación:**
 - ▶ Activados por el servidor (*well-known objects*)
 - ▶ Se activan **cada vez** que **el cliente invoca el objeto**
 - ▶ Reduce tráfico de red
 - ▶ Existen dos clases:
 - *Singleton*:
Existe un **único objeto atendiendo a múltiples clientes**
 - *Singlecall*:
Creados y destruidos en cada invocación de un cliente
 - ▶ Activados por el cliente
 - ▶ Se activa **cada vez** que **el cliente crea el objeto**
 - ▶ Sólo **sirve** a un **único cliente**

.NET *remoting*: modos de activación

▶ Modos de activación:

▶ Activados por el servidor (*well-known objects*)

Invocación
en el cliente

```
object[] url = {new UriAttribute(
    "tcp://computername:8080/RemoteObjectName"
)};

RemoteObjectClass MyRemoteClass = (RemoteObjectClass)
    Activator.CreateInstance(typeof(RemoteObjectClass),
        null,
        url);
```

▶ Activados por el cliente

Invocación
en el cliente

```
RemoteObjectClass MyRemoteClass = (RemoteObjectClass)
    Activator.GetObject(typeof(RemoteObjectClass),
        "tcp://computername:8080/RemoteObjectUri");
```



.NET remoting: *Application Domains*

▶ Procesos y *Application Domains*:

- Sistema operativo Microsoft Windows asocia cada aplicación a un proceso.
- *Application Domains*: proporciona aislamiento y seguridad a la ejecución de aplicaciones.
 - Objetos *Nonremotable*: únicamente son accedidos dentro del dominio.
 - Objetos *Remotable*: pueden ser enviados fuera del dominio por referencia o por valor.

Contenidos

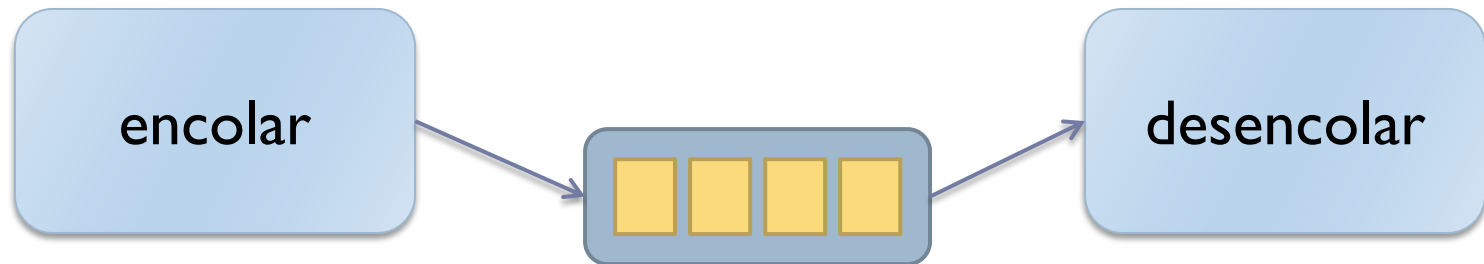
1. Introducción a la programación en .NET
2. .NET remoting
3. Cola de mensajes

Cola de mensajes

- Similar a *Microsoft Message Queuing* (MSMQ).
- *System.Messaging* namespace.
- Serialización en XML.

Ejemplo: cola de mensajes

- ▶ Dos procesos comunicados por un sistema de colas



encolar.cs (1/2)

```
using System;
using System.Messaging;

public struct Customer
{
    public string Last;
    public string First;
}
```

encolar.cs (2/2)

```
public class Enqueue
{
    public static void Main( )
    {
        try
        {
            string path = ".\\PRIVATE$\\NE_queue";
            if (!MessageQueue.Exists(path))
            {
                // Create our private queue.
                MessageQueue.Create(path);
            }
            // Initialize the queue.
            MessageQueue q = new MessageQueue(path);

            // Create our object.
            Customer c = new Customer( );
            c.Last = "Osborn";
            c.First = "John";

            // Send it to the queue.
            q.Send(c);
        }
        catch(Exception e)
        {
            Console.WriteLine(e.ToString( ));
        }
    }
}
```


desencolar.cs (1/2)

```
using System;
using System.Messaging;
using System.Runtime.Serialization;

public struct Customer
{
    public string Last;
    public string First;
}

public class Dequeue
{
    public static void Main( )
    {
        try
        {
            string strQueuePath = ".\\PRIVATE$\\NE_queue";

            // Ensure that the queue exists
            if (!MessageQueue.Exists(strQueuePath))
            {
                throw new Exception(strQueuePath + " doesn't exist!");
            }
        }
    }
}
```

desencolar.cs (2/2)

```
// Initialize the queue
MessageQueue q = new MessageQueue(strQueuePath);

// Specify the types we want to get back
string[] types = {"Customer, dequeue"};
(XmlMessageFormatter)q.Formatter.TargetTypeNames = types;

// Receive the message (5 sec timeout)
Message m = q.Receive(new TimeSpan(0,0,5));

// Convert the body into the type we want
Customer c = (Customer) m.Body;
Console.WriteLine("Customer: {0}, {1}", c.Last, c.First);
}
catch(Exception e)
{
    Console.WriteLine(e.ToString( ));
}
}
```

Compilación

```
C:\Temp> csc /t:exe /out:encolar.exe encolar.cs
```

```
C:\Temp> csc /t:exe /out:desencolar.exe desencolar.cs
```



Desarrollo de aplicaciones distribuidas con .NET:
Programación distribuida con .NET



Grupo ARCOS

Desarrollo de Aplicaciones Distribuidas

Ingeniería Informática

Universidad Carlos III de Madrid