Llamada a Métodos Remotos (Java RMI)

Ejemplos

Sistemas Distribuidos
Grado en Ingeniería Informática

Ejemplo 1. Calculadora Remota mediante RMI con la necesidad de utilizar el programa externo rmiregistry

En este ejemplo vamos a ver como implementar un servicio de calculadora sencilla mediante la invocación o llamadas a métodos remostos (RMI) que proporciona el lenguaje Java. Este servicio necesita que por parte del usuario, se lance la aplicación **rmiregistry [nº de puerto]** en el mismo directorio donde se encuentre el servidor compilado (**ficheros .class**).

Se ha optado por implementar la calculadora de manera modular, es decir, por una parte el cliente, y el servidor y por otra el servicio. La clase servicio debe ser conocida tanto por la clase cliente como por la clase servidor. La definición de un servicio en Java RMI se realizar mediante la definición de una interfaz que extiende o hereda de la clase **remote**. La clase **remote** está definida en **java.rmi**.

Empezamos con la definición e implementación del servicio de calculadora remota.

Fichero calculadora.java

```
import java.rmi.*;
import java.util.LinkedList;

public interface Calculadora extends Remote {
    float Sumatorio (LinkedList l) throws RemoteException;
    float sumar(float a, float b) throws RemoteException;
    float restar(float a, float b) throws RemoteException;
    float multiplicar(float a, float b) throws RemoteException;
    float dividir(float a, float b) throws RemoteException;
    float Operacion(float a, char o, float b) throws RemoteException;
}
```

En caso de que algún método deba ser ejecutado de manera síncrona se incluirá la clausula `synchronized`.

La Implementación de la interfaz puede hacerse de dos maneras que afectaran a la implementación del servidor. La primera, que es la que se muestra a continuación es la más común pero necesita lanzar externamente el comando rmiregistry que habilita el registro del stubs del servidor.

Fichero calculadoraImpl.java

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.LinkedList;
import java.util.Iterator;

public class CalculadoraImpl extends UnicastRemoteObject implements Calculadora{

    public CalculadoraImpl() throws RemoteException {
        super();
    }

    public float sumar(float a, float b) throws RemoteException {
        System.out.println("SERVIDOR ha hecho: "+a+"+"+b+"="+(a+b));
        return(a+b);
    }

    public float restar(float a, float b) throws RemoteException {
        System.out.println("SERVIDOR ha hecho: "+a+"-"+b+"="+(a-b));
        return(a-b);
    }

    public float multiplicar(float a, float b) throws RemoteException {
        System.out.println("SERVIDOR ha hecho: "+a+"*"+b+"="+(a*b));
        return(a*b);
    }

    public float dividir(float a, float b) throws RemoteException {
        System.out.println("SERVIDOR ha hecho: "+a+"/"+b+"="+(a/b));
        return(a/b);
    }
```

Esta primera implementación implementa la interfaz común definida y hereda de la clase `UnicastRemoteObject`.

`super` exporta automáticamente el objeto al registro y lo asocia automáticamente al mecanismo de comunicación del RMI

```
    public float Operacion(float a, char o, float b) throws RemoteException {
        float resultado=0;
        switch(o)
        {
            case '+': resultado=a+b;
                      break;
            case '-': resultado=a-b;
                      break;
            case '*': resultado=a*b;
                      break;
            case '/': resultado=a/b;
                      break;
            default: System.out.println("Error En SERVIDOR Calculadora: Operación "+o+" desconocida.");
        }
        System.out.println("SERVIDOR ha hecho: "+a+o+b+"="+resultado);
        return resultado;
    }

    public float Sumatorio (LinkedList l) throws RemoteException
    {
        System.out.println("Numero de elementos = "+l.size());
        float Suma=0,a,b;
        char operacion;
        Object object;
        for (Iterator it = l.iterator(); it.hasNext();) {
            object = it.next();
            a=Float.parseFloat(object.toString());
            object = it.next();
            b=Float.parseFloat(object.toString());
            object = it.next();
            operacion= object.toString().charAt(0);
            Suma=Suma+Operacion(a, operacion, b);
        }
        System.out.println("SERVIDOR ha hecho: Suma total = "+Suma);
        return Suma;
    }
}
```

RMI permite también utilizar referencia a objetos. Esta referencia no es al objeto almacenado en el cliente, sino que es una referencia a un objeto obtenido por la deserialización del objeto enviado desde el cliente.

El programa servidor, crear un objeto que implementa la interfaz que define el servicio (**calculadora**). Este objeto automáticamente devuelve su **stub** para el cliente y lo almacena en el registro RMI (**rmi registry**).

Fichero RMI_Calculadora_Servidor.java

```
import java.rmi.*;
import java.util.Scanner;

public class RMI_Calculadora_Servidor {

    public static void main(String[] args) {

        try {

            Calculadora calcStub = new CalculadoraImpl();
            int Puerto=0;
            Scanner Teclado=new Scanner(System.in);
            System.out.print("Introduce el nº de puerto para comunicarse: ");
            Puerto=Teclado.nextInt();

            Naming.rebind("rmi://localhost:"+Puerto+"/Calculadora", calcStub);

            System.out.println("Servidor Calculadora esperando peticiones ... ");

           //Naming.unbind("rmi://localhost:"+Puerto+"/Calculadora");

        } catch (Exception e) {
            System.out.println("Error en servidor Calculadora:"+e);
        }
    }

}
```

El servidor, solicita el puerto por el que puede acceder el registro RMI y de de alta el servicio de calculadora mediante el método `Naming.rebind`

En caso de poder cerrar el servidor se puede dar de baja el servicio mediante el método `Naming.unbind`

Fichero RMI_Calculadora_Cliente.java

```
import java.rmi.Naming;
import java.rmi.RMISecurityManager;
import java.util.Random;
import java.util.Scanner;
import java.util.LinkedList;

public class RMI_Calculadora_Cliente {

    public static int MenuPrincipal()
    {
        Scanner Teclado=new Scanner(System.in);
        int Salida;
        do
        {
            System.out.println("\n***************************");
            System.out.println("**");
            System.out.println("** 1.- Sumar");
            System.out.println("** 2.- Restar");
            System.out.println("** 3.- Multiplicar");
            System.out.println("** 4.- Dividir");
            System.out.println("** 5.- Automático");
            System.out.println("** 6.- Sumar Vector de Operaciones");
            System.out.println("** 7.- Salir");
            System.out.println("**");
            System.out.print("** Elige Opcion:");
            Salida=Teclado.nextInt();
        } while (Salida<1 || Salida>7);
        return Salida;
    };
```

El programa cliente implementa un menú con las posibles operaciones a invocar remotamente.

```
public static void main(String[] args) {
    try {
        int Puerto=0;
        String Host;
        Scanner Teclado=new Scanner(System.in);
        System.out.print("Introduce el nº de puerto para comunicarse: ");
        Puerto=Teclado.nextInt();
        System.out.print("Introduce el nombre del host: ");
        Host=Teclado.next();

        // Obtiene el stub del rmiregistry
        Random rnd=new Random(System.nanoTime());
        Calculadora calcStub = (Calculadora) Naming.lookup("rmi://"+Host+":"+Puerto+"/Calculadora");

        boolean OtraOperacion=false;
        float a=0,b=0,resultado=0;
        int Nveces=0, Opcion=0;
        char operacion=' ';
        String SN;
        do
        {
            Opcion = MenuPrincipal();
            if (Opcion<5)
            {
                System.out.print("Introduce el operando 1: ");
                a=Teclado.nextFloat();
                System.out.print("Introduce el operando 2: ");
                b=Teclado.nextFloat();
                switch (Opcion)                    {
                    case 1: resultado=calcStub.sumar(a, b);
                            operacion='+';
                            break;
                    case 2: resultado=calcStub.restar(a, b);
                            operacion='-';
                            break;
                    case 3: resultado=calcStub.multiplicar(a, b);
                            operacion='*';
                            break;
```

El programa cliente solicita el puerto para comunicarse con el rmiregistry y el host donde está implementando el servicio de calculadora

Con el método `Naming.lookup` obtenemos el stub del servidor para el proceso cliente. Mediante este stub el cliente se comunicará con el servidor.

La invocación remota se realiza exactamente igual que la invocación local de un método.

```
                    case 4: resultado=calcStub.dividir(a, b);
                            operacion='/';
                }
                System.out.println("CLIENTE dice: El resultado de "+a+operacion+b+"="+resultado);
            }
            else
                if (Opcion<7)  {
                    LinkedList Lista=new LinkedList();
                    System.out.print("Introduce el de veces a realizar operaciones aleatorias: ");
                    Nveces=Teclado.nextInt();
                    for (int i=0; i<Nveces; i++)    {
                        a=rnd.nextFloat()*2000-1000;
                        b=rnd.nextFloat()*2000-1000;
                        switch(rnd.nextInt(4))   {
                            case 0: operacion='+'; break;
                            case 1: operacion='-'; break;
                            case 2: operacion='*'; break;
                            case 3: operacion='/'; break;
                        };
                        if (Opcion==5)    {
                            resultado=calcStub.Operacion(a, operacion, b);
                            System.out.println("CLIENTE dice "+i+": El resultado de "+a+operacion+b+"="+resultado);
                        }
                        else   {
                            Lista.add(a); Lista.add(b);
                            Lista.add(operacion);
                            System.out.println("CLIENTE dice "+i+": ("+a+operacion+b+") + ....");
                        };
                    }
                    if (Opcion==6)  {
                         System.out.println("-----------------------------------------------------------");
                         System.out.println("CLIENTE dice: Suma Total = "+calcStub.Sumatorio(Lista));
                    }
                };
            } while (Opcion!=7);
        } catch (Exception e) { System.out.println("Error: " + e);  }
    }
}
```

La opción 5, generamos una serie de operaciones aleatorias con números aleatorios. Estas operaciones son invocadas remotamente una a una.

La opción 6, es igual pero almacena la operación en una lista que posteriormente será trasladada al servidor mediante una única invocación remota.

Tanto el cliente como el servidor deben ser lanzados mediante un fichero que contenga los permisos de acceso al registro RMI por parte del servidor y el cliente. El contenido de fichero es el mismo para el servidor y el cliente. Este fichero contiene las siguientes líneas:

```
grant {
      permission java.security.AllPermission;
};
```

La invocación tanto del servidor como la del cliente ser realiza de dos formas:

1. En el directorio donde están los fichero compilados (.class)

```
java –Djava.security.policy=fichero.permiso  RMI_Calculadora_Cliente
```

2. En el direcotorio donde está el fichero comprimido (.jar)

```
java –Djava.security.policy=fichero.permiso  -jar RMI_Calculadora_Cliente.jar
```

Ejemplo 2. Calculadora Remota mediante RMI sin la necesidad de utilizar el programa externo rmiregistry

Este ejemplo es exactamente el mismo que el anterior con la salvedad de que no es necesario que el usuario lance la aplicación **rmiregistry**. En este ejemplo es el servidor el que lanza automáticamente dicha aplicación lo que hace más independiente el código del usuario.

El servicio es exactamente idéntico que el visto en el ejemplo anterior. La implementación del servicio, se realiza de otra forma, si necesitar extender o heredar de la clase `UnicastRemoteObject` por lo que deberá modificarse ligeramente el código del servidor. También el programa cliente es exactamente el

mismo al visto en el ejemplo anterior. A continuación presentamos solo los cambios en el programa servidor, el resto es idéntico como hemos comentado.

FicheroRMI_Calculadora_Servidor.java

```
import java.rmi.RemoteException;
import java.rmi.server.*;
import java.rmi.registry.*;
import java.util.Scanner;

public class RMI_Calculadora_Servidor {

    public static void main(String[] args) throws RemoteException {

        try {
            int Puerto = 0;
            Scanner Teclado=new Scanner(System.in);
            System.out.print("Introduce el nº de puerto para comunicarse: ");
            Puerto=Teclado.nextInt();

            Registry registry = LocateRegistry.createRegistry(Puerto);
            CalculadoraImpl  obj = new CalculadoraImpl();

            Calculadora stub = (Calculadora) UnicastRemoteObject.exportObject(obj,Puerto);

            registry = LocateRegistry.getRegistry(Puerto);
            registry.bind("Calculadora", stub);

            System.out.println("Servidor Calculadora esperando peticiones ... ");
        } catch (Exception e) {
            System.out.println("Error en servidor Calculadora:"+e);
        }
    }
}
```

Este servidor crea localmente un servicio rmiregistry mediante el método `LocateRegistry.creaeRegistry`

Obtiene el stub remoto que gestionará el servicio de calculadora mediante el método `UnicastRemoteObject.exportObject`

Obtiene el registro local y registra el stub calculadora mediante los métodos `getRegistry` y `bind` respectivamente.

En caso de utilizar Netbeans u otro entorno de programación, no hay que olvidar de situar el directorio de trabajo dentro del proyecto, en concreto en el subdirectorio **build\classes** y en opciones para la máquina virtual de java un enlace al fichero que contiene los permisos de acceso al **rmregistry**.

Para el Netbeans, estas modificaciones se realizan en la configuración del proyecto, en la categoría **run** tal como muestra la siguiente figura.