Java RMI

las RPC de Java. Parte I

Luis Fernando Llana Díaz

Departamento de Sistemas Informáticos y Programación
Universidad Complutense de Madrid

21 de marzo de 2006

RMI y RPC

RPC: *Remote Procedure Call*.

RMI y RPC

RPC: *Remote Procedure Call*.

RMI: *Remote Method Invocation*.

RMI y RPC

RPC: *Remote Procedure Call*.

RMI: *Remote Method Invocation*.

Llamada a métodos (procedimientos, funciones) que están en otra máquina abstracta,

RMI y RPC

RPC: *Remote Procedure Call*.

RMI: *Remote Method Invocation*.

Llamada a métodos (procedimientos, funciones) que están en otra máquina abstracta,

- de la misma máquina,
- de una máquina remota.

Definición de objetos remotos

- El cliente y el servidor deben conocer un interfaz común. El cliente sólo conoce el interfaz.
- El servidor debe implementar el interfaz `java.rmi.Remote`.

```
package  ejemplosRMI.hebras;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Prueba1 extends Remote {
    public int mete(String s) throws RemoteException,InterruptedException;
    public int saca()  throws RemoteException;
}
```

Servidor

```
package  ejemplosRMI.hebras;

import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
import java.rmi.Naming;

public class Prueba1Impl extends UnicastRemoteObject implements Prueba1 {
    private int hanEntrado;
    public synchronized int  mete(String s)
        throws RemoteException,InterruptedException {
        hanEntrado++;
        System.out.println("Entrando:"+s+":"+"han entrado:"+hanEntrado);
        wait();
        System.out.println("Saliendo:"+s+":"+"hay dentro:"+hanEntrado);
        return hanEntrado;
      }
    public synchronized int  saca()  throws RemoteException {
        System.out.println("sacando...");
        notify();
        return hanEntrado;
    }
    public Prueba1Impl() throws RemoteException{
        hanEntrado = 0;
    }
}
```

Cliente I

Problema: El cliente sólo conoce un interfaz.

Cliente II

Problema: El cliente sólo conoce un interfaz.¿Cómo construye un objeto al cual invocar?

Cliente III

Problema: El cliente sólo conoce un interfaz.¿Cómo construye un objeto al cual invocar?

1. Debe buscar un servidor

Cliente IV

Problema: El cliente sólo conoce un interfaz.¿Cómo construye un objeto al cual invocar?

1. Debe buscar un servidor y dentro del servidor un nombre: `rmi://127.0.0.1/prueba1`.

Cliente V

Problema: El cliente sólo conoce un interfaz.¿Cómo construye un objeto al cual invocar?

1. Debe buscar un servidor y dentro del servidor un nombre: `rmi://127.0.0.1/prueba1`.
2. En el servidor debe darle el objeto.

Cliente VI

Problema: El cliente sólo conoce un interfaz.¿Cómo construye un objeto al cual invocar?

1. Debe buscar un servidor y dentro del servidor un nombre: `rmi://127.0.0.1/prueba1`.
2. En el servidor debe darle el objeto. Fichero *Stub*.

Cliente VII

```
package  ejemplosRMI.hebras;

import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.Naming;
import java.net.MalformedURLException;
public class Cliente1 {
    public static void main (String args[]) throws NotBoundException,
                                                   MalformedURLException,
                                                   RemoteException,
                                                   InterruptedException {

        String url=args[0];
        String yo=args[1];
        System.out.println("Buscando "+url+"...");
        Prueba1 pr = (Prueba1)Naming.lookup(url);
        int n=pr.mete(yo);
        System.out.println("Han pasado:"+n);
    }
}
```

Cuestiones técnicas

- Poner en marcha un servidor, registar un servicio.
- Localizar la máquina y llamar al servicio.

En el servidor hay que arrancar el *registro*

```
#!/bin/sh
. ./comun.sh
rmiregistry &
echo $! >> $PIDFILE
```

Registrar el servidor

```
public static void main (String[] args) {
    String nombre = args[0];
     if (System.getSecurityManager() == null) {
        System.setSecurityManager(new RMISecurityManager());
     }
     try {
         Prueba1 servidor = new Prueba1Impl();
         Naming.rebind(nombre,servidor);
         System.out.println("Servidor "+ nombre+" funcionando");
     } catch (Exception e) {
         System.err.println("óExcepcin en "+nombre +": " + e.getMessage());
         e.printStackTrace();
     }
}
```

Arrancando todo I

Parte comun

```
#!/bin/sh
controlAcceso="-Djava.security.policy=./servidor.policy"
JAVA="java $controlAcceso"
url="rmi://localhost/prueba1"
export CLASSPATH=~/Java/classes
PIDFILE=pids
```

Servidor

```
#!/bin/sh
. ./comun.sh
$JAVA ejemplosRMI.hebras.Prueba1Impl $url &
echo $! >> $PIDFILE
```

Para finalizar...

```
#!/bin/sh
. ./comun.sh
kill -9 `cat $PIDFILE`
rm $PIDFILE
```

Arrancando todo II

Política de seguridad

```
grant codebase "file:/home/luis/Java/classes/-" {
  permission java.net.SocketPermission "*:1024-65535", "connect,accept,resolve";
  permission java.net.SocketPermission "*:80", "connect";
};
```

Clientes

```
#!/bin/sh
. ./comun.sh
$JAVA ejemplosRMI.hebras.ClienteSaca1 $url
$JAVA ejemplosRMI.hebras.ClienteSaca1 $url
$JAVA ejemplosRMI.hebras.ClienteSaca1 $url
$JAVA ejemplosRMI.hebras.ClienteSaca1 $url
```

```
#!/bin/sh
. ./comun.sh
$JAVA ejemplosRMI.hebras.Cliente1 $url A
$JAVA ejemplosRMI.hebras.Cliente1 $url B
$JAVA ejemplosRMI.hebras.Cliente1 $url C
$JAVA ejemplosRMI.hebras.Cliente1 $url D
```

Arrancando todo III

```
package  ejemplosRMI.hebras;

import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.Naming;
import java.net.MalformedURLException;
public class ClienteSaca1 {
    public static void main (String args[])
            throws NotBoundException,MalformedURLException,
                   RemoteException, InterruptedException {
        String url=args[0];
        Prueba1 pr = (Prueba1)Naming.lookup(url);
        int n=pr.saca();
        System.out.println("Han entrado:"+n);
    }
}
```

Ficheros Stub

El cliente necesita el objeto remoto, *sucedáneo* el fichero Stub.

```
Usage: rmic <options> <class names>

where <options> includes:
  -verbose        Output messages about what the compiler is doing
  -classpath <path>      Specify where to find input class files
  -bootclasspath <path>  Override location of bootstrap class files
  -extdirs <path>        Override location of installed extensions
  -d <directory>         Specify where to place generated class files
  -J<runtime flag>       Pass argument to the java interpreter
```

```
rmic -d classes ejemplosRMI.hebras.Prueba1Impl
```

```
  <target name="hebras">
    <javac destdir="${build}" classpath="${classpath}" listfiles="yes"
           optimize="on" srcdir="${src}" includes="ejemplosRMI/hebras/**">
    </javac>
    <rmic base="${build}"
       includes="ejemplosRMI/hebras/Prueba1Impl.class"/>
  </target>
```

Ficheros Stub

Son necesarios para el

Ficheros Stub

Son necesarios para el

- Registro (`rmiregistry`) y el servidor, deben tenerlo en su `CLASSPATH`.

Ficheros Stub

Son necesarios para el

- Registro (`rmiregistry`) y el servidor, deben tenerlo en su `CLASSPATH`.
- El cliente, ¿Cómo se distribuyen?

Ficheros Stub

Son necesarios para el

- Registro (`rmiregistry`) y el servidor, deben tenerlo en su `CLASSPATH`.
- El cliente, ¿Cómo se distribuyen?
  - El cliente no tiene porqué estar en la máquina del cliente.

Ficheros Stub

Son necesarios para el

- Registro (`rmiregistry`) y el servidor, deben tenerlo en su `CLASSPATH`.
- El cliente, ¿Cómo se distribuyen?
  - El cliente no tiene porqué estar en la máquina del cliente.
  - Si cambia la clase servidor, hay que actualizar la clase Stub.

Ficheros Stub

Son necesarios para el

- Registro (`rmiregistry`) y el servidor, deben tenerlo en su `CLASSPATH`.
- El cliente, ¿Cómo se distribuyen?
  - El cliente no tiene porqué estar en la máquina del cliente.
  - Si cambia la clase servidor, hay que actualizar la clase Stub.
  - El cliente no tiene porqué estar atento de los cambios del servidor.

Ficheros Stub

Son necesarios para el

- Registro (`rmiregistry`) y el servidor, deben tenerlo en su `CLASSPATH`.
- El cliente, ¿Cómo se distribuyen?
  - El cliente no tiene porqué estar en la máquina del cliente.
  - Si cambia la clase servidor, hay que actualizar la clase Stub.
  - El cliente no tiene porqué estar atento de los cambios del servidor.
  - El cliente puede coger las clases de una url (`java.rmi.server.codebase`).

Ficheros Stub

Son necesarios para el

- Registro (`rmiregistry`) y el servidor, deben tenerlo en su `CLASSPATH`.
- El cliente, ¿Cómo se distribuyen?
  - El cliente no tiene porqué estar en la máquina del cliente.
  - Si cambia la clase servidor, hay que actualizar la clase Stub.
  - El cliente no tiene porqué estar atento de los cambios del servidor.
  - El cliente puede coger las clases de una url (`java.rmi.server.codebase`).
  - Problemas de seguridad.

Ficheros Stub

Son necesarios para el

- Registro (`rmiregistry`) y el servidor, deben tenerlo en su `CLASSPATH`.
- El cliente, ¿Cómo se distribuyen?
  - El cliente no tiene porqué estar en la máquina del cliente.
  - Si cambia la clase servidor, hay que actualizar la clase Stub.
  - El cliente no tiene porqué estar atento de los cambios del servidor.
  - El cliente puede coger las clases de una url (`java.rmi.server.codebase`).
  - Problemas de seguridad.

Canales síncronos I

```
public interface Canal extends Remote {
    public void envia(Object o) throws RemoteException;
    public Object recibe() throws RemoteException;
    public String nombre() throws RemoteException;
}
```

```
package ejemplosRMI.canales;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class CanalSinc  extends UnicastRemoteObject implements Canal{
    private Object mensaje;
    private boolean lleno;
    private boolean leido;
    public synchronized void envia(Object param1) throws RemoteException {
        try {
            while ( lleno  ) {wait();}
            lleno = true;
            leido = false;
            mensaje = param1;
            notifyAll();
            while ( !leido ) {wait(); }
        } catch ( InterruptedException e ) {
            throw new RuntimeException(e);
        }
    }
```

Canales síncronos II

```
    public synchronized Object recibe()  throws RemoteException {
        try {
            while ( !lleno ) { wait(); }
            lleno = false;
            leido = true;
            notifyAll();
        } catch ( InterruptedException e ) {
            throw new RuntimeException(e);
        }
        return mensaje;
    }
}
```

Canales síncronos III

- Los procesos A y B quieren comunicar a través del canal c.
- A y B no se conocen entre sí.

Canales síncronos IV

- Los procesos A y B quieren comunicar a través del canal c.
- A y B no se conocen entre sí.
- Hace falta un intermediario que es conocido por los procesos involucrados.
- El gestor registra los canales a través de los que se quieren comunicar los procesos.

Canales síncronos V

```
package ejemplosRMI.canales;                                                    1
import java.rmi.Remote;                                                         2
import java.rmi.RemoteException;                                                3
public interface  Gestor extends Remote {                                       4
    public void registra(String nombre, Canal canal) throws RemoteException;    5
    public boolean existeCanal(String nombre) throws RemoteException;           6
    public Canal canal(String nombre) throws RemoteException, InterruptedException;7
}                                                                               8
```

```
public class GestorAntares extends UnicastRemoteObject implements Gestor  {    1
    private Hashtable tablaCanales;                                             2
    public synchronized Canal canal(String nombre)                              3
        throws RemoteException, InterruptedException {                          4
        Object c=tablaCanales.get(nombre);                                      5
        while (c==null || !(c instanceof Canal)) {                              6
            wait();                                                             7
            c=tablaCanales.get(nombre);                                         8
        }                                                                       9
        return (Canal)c;                                                        10
    }                                                                           11
```

Canales síncronos VI

```
    public synchronized void registra(String nombre, Canal canal)
            throws RemoteException {
        debug("Registrando canal:"+nombre+"...");
        Object c = tablaCanales.get(nombre);
        if (  c != null && ( c instanceof Canal) ) {
            debugln("ya íexista\n");
        } else {
            debugln("nuevo");
            tablaCanales.put(nombre,canal);
            notifyAll();
        }
    }
    public synchronized boolean existeCanal(String nombre)
        throws RemoteException {
        Object c = tablaCanales.get(nombre);
        return c!=null; //Esto áest mal
    }
}
```

Canales síncronos VII

```
public class Prueba{                                                    1
    private static Canal construyeCanal(String nombre, Gestor gestor)   2
        throws RemoteException, InterruptedException {                  3
        Canal canal=null;                                               4
        if (gestor.existeCanal(nombre)) {                               5
            System.out.println("Canal:"+nombre+":existe:");             6
            canal = gestor.canal(nombre);                               7
            System.out.println("Canal:"+nombre+":recibido:");           8
        } else {                                                        9
            System.out.println("Canal:"+nombre+":NO existe:");          10
            canal = new CanalSinc(nombre);                              11
            gestor.registra(nombre,canal);                              12
        }                                                               13
        return canal;                                                   14
    }                                                                   15
    ..................                                                  16
}                                                                       17
```

Canales síncronos VIII

```
public synchronized boolean existeCanal(String nombre)
    throws RemoteException {
    Object c = tablaCanales.get(nombre);
    if (c==null) {
        tablaCanales.put(nombre,new Boolean(false));
        return false;
    } else {
        return true;
    }
```