Sistemas Distribuidos
Java RMI (*Remote Method Invocation*)

Alberto Lafuente

Mikel Larrea

Dpto. ATC, UPV/EHU

Contenido

- Interfaz
- Implementación
- Servidor
- Cliente
- Puesta en marcha de la aplicación:
  - Compilador de Java
  - Compilador de RMI
  - Iniciar el servidor de nombres de RMI
  - Iniciar el servidor
  - Iniciar el cliente

Interfaz

- Definición de los objetos que serán accedidos remotamente:
  - métodos, con sus parámetros
- No se da la implementación de los métodos
- Necesario extender el interfaz java.rmi.Remote
- En todos los métodos remotos debe indicarse la posibilidad de lanzar la excepción java.rmi.RemoteException

Interfaz (Hello.java)

```
package hello;

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.Date;

public interface Hello extends Remote {
   String sayHello() throws RemoteException;
   Date getDate() throws RemoteException;
}
```

Implementación

- Es una clase Java normal
- Extiende la clase java.rmi.server.UnicastRemoteObject, e implementa el interfaz definido en el paso anterior
  - Implementa todos los métodos remotos del interfaz
- En el método constructor de esta clase únicamente se llama al constructor de la clase UnicastRemoteObject: *super()*

Implementación (HelloImpl.java)

```
package hello;

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

public class HelloImpl extends UnicastRemoteObject
   implements Hello {

   public HelloImpl() throws RemoteException {
      super();
   }
   public String sayHello() {
      return "Kaixo Mikel!";
   }
   public Date getDate() {
      return new Date();
   }
   // ...
```

Servidor

- Inicia un gestor de seguridad
- Crea el objeto que será accesible remotamente
  - Se trata de una instancia de la clase implementación
- Registra el objeto remoto (con un nombre dado) en el servidor de nombres de RMI, mediante el método Naming.rebind
- El servidor puede implementarse en una clase independiente, o también dentro de la clase implementación (método *main*)

Servidor (HelloImpl.java)

```
   // ...

   public static void main(String args[]) {

      // Create and install a security manager
      System.setSecurityManager(new RMISecurityManager());

      try {
         Hello obj = new HelloImpl();
         // Bind this object instance to the name "HelloServer"
         Naming.rebind("HelloServer", obj);
         System.out.println("HelloServer bound in registry");
      } catch (Exception e) {
         System.out.println("HelloImpl exception: " +
                            e.getMessage());
         e.printStackTrace();
      }
   }
}
```

Cliente

- Inicia un gestor de seguridad
- Obtiene una referencia del objeto remoto al que desea acceder, mediante una petición al servidor de nombres de RMI
  - Método Naming.lookup, usando el mismo nombre con el que el objeto remoto fue registrado
- Una vez obtenida la referencia, invoca los métodos del objeto remoto como si de un objeto local se tratase, recibiendo los resultados y mostrándolos en la pantalla

Cliente (HelloClient.java)

```
package hello;

import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
import java.util.Date;

public class HelloClient {

   public static void main(String args[]) {

      if (args.length != 1) {
         System.out.println(("Usage: java HelloClient host");
         Systm.exit(1);
      }

      System.setSecurityManager(new RMISecurityManager());
      // ...
```

Cliente (HelloClient.java)

```
        // ...
        try {
           Hello obj = (Hello)Naming.lookup("rmi://" + args[0] +
                                            "/HelloServer");
           String str = obj.sayHello();
           System.out.println(str);
           // invoke method on server object and locally
           // (to compare dates)
           Date d_remote = obj.getDate();
           Date d_local = new Date();
           System.out.println("Date on server is " + d_remote);
           System.out.println("Date on client is " + d_local);
        } catch (Exception e) {
           System.out.println("HelloClient exception: " +
                              e.getMessage());
           e.printStackTrace();
        }
     }
}
```

Puesta en marcha de la aplicación

- Compilador de Java:
  - javac hello/*.java
- Compilador de RMI (sobre la clase implementación):
  - rmic hello.HelloImpl
- Iniciar el servidor de nombres de RMI (incluido en la distribución de Java):
  - rmiregistry
- Iniciar el servidor:
  - java –Djava.security.policy=hello/policy hello.HelloImpl
- Iniciar el cliente:
  - java –Djava.security.policy=hello/policy hello.HelloClient host

Ejercicio 1

- Desarrollar un servicio Java RMI de cálculo de potencias. Las operaciones ofertadas por el servicio serán las siguientes:
  - Operación de cálculo del cuadrado:

```
long square(int n);
```

  - Operación de cálculo de potencias:

```
long power(int n1, int n2);
```

Ejercicio 2

- Probar entre dos máquinas del laboratorio el ejemplo RMI que simula el algoritmo de Cristian de sincronización de relojes
  - Nombre del archivo: `cristian.tar`
- Adaptar el ejemplo anterior para que simule el algoritmo simétrico de NTP
  - Pista: se deben obtener dos tiempos de cada máquina