Práctica 7

Chat mediante uso de Java RMI

Grado Ingeniería Telemática
GSyC, Universidad Rey Juan Carlos

1. Introducción

Como ya hemos visto en clase, una *RPC* o llamada a procedimiento remoto, consiste en procesar de manera remota algún fragmento de código usando la CPU, Memoria, Stack y -en definitiva- recursos remotos sin salir del procesamiento local.

Esto exige considerar una serie de detalles y problemas, como por ejemplo:

1. *Marshalling* y *Unmarshalling*: estandarizar el orden de los datos.
2. Inestabilidad de la Red: cómo retornar a la programación.
3. Seguridad: cualquiera no pueda interrumpir el flujo de procesamiento.
4. Representación de Datos: *charset*, *UTF8*, *ASCII*, etc.
5. Distribucion de Hardware: distinta zona de memoria, stack, clock de CPU, etc.

Todos estos problemas son, evidentemente, también aplicables a *RMI*.

2. RMI

RMI es un paquete de Java que permite manejar objetos (y sus respectivos métodos) de manera remota, de forma que se pueden usar los recursos de un servidor de manera transparente para el usuario local.

Como ya hemos visto, la forma en que *RMI* (y *RPC* en general) logra hacer esto, es por medio de lo que se conoce como *stubs*. En el caso del *stub* servidor, se conoce como *skeleton*. Estos *stubs* y *skeletons* permiten que, al momento de ser invocada la función remota, ésta pueda ser simulada localmente.

2.1. Interfaz

Para la comunicación entre el servidor y el cliente, se trabaja con interfaces, que deben ser implementadas por el servidor y/o cliente, para que los *stubs* puedan realizar la transparencia para ambos. Además, esto evita que deba existir una definición local real de la clase remota; esto es, en el cliente solo debe estar definida la interfaz, no la clase remota.

Otro punto importante en *RMI*, es el como se produce la conectividad entre el cliente y servidor. Para esto se ocupa una herramienta de Java, llamada *RMI Registry*.

El *RMI Registry* puede estar localizado en un lugar distinto al servidor, y se encarga de registrar un determinado objeto y asignarle un servidor que se encargará de procesar dicho objeto.

El funcionamiento general es:

1. Se ejecuta el RMI Registry, en algún lugar de la red.

2. El servidor que desea manejar un objeto, se registra en dicho servidor.
3. El RMI Registry registra el par: objeto/servidor.
4. El cliente que necesita utilizar un determinado objeto hace una consulta al RMI Registry, quien devuelve el stub listo para la comunicación.

3. Especificación de la clase *InterfazChat*

En primer lugar se escribe la interfaz remota del servidor, que en este caso se llamará `InterfazChat`. Toda interfaz remota debe declararse como `public` y debe extender la interfaz `java.rmi.Remote`.

Además, esta interfaz debe definir los métodos que serán accesibles remotamente. Por último cada uno de estos métodos debe manejar la excepción `java.rmi.RemoteException`.

Así, una posible interfaz podría ser el siguiente:

```
package chat;

import java.rmi.*;

public interface InterfazChat extends Remote {
public String getName () throws RemoteException;
public void send (String msg) throws RemoteException;
public void setClient (InterfazChat c)throws RemoteException;
public InterfazChat getClient () throws RemoteException;
}
```

Cuya implementación podría definirse en una clase `Chat` como sigue:

```
package chat;

import java.rmi.*;
import java.rmi.server.*;

public class Chat extends UnicastRemoteObject implements InterfazChat  {

public String name;
public InterfazChat client=null;

public Chat(String n)  throws RemoteException {
this.name=n;
}
public String getName() throws RemoteException {
return this.name;
}

public void setClient(InterfazChat c){
client=c;
}

public InterfazChat getClient(){
return client;
}
```

```
public void send(String s) throws RemoteException{
System.out.println(s);
}
}
```

4. Ejercicios

Ejercicio 1. Chat cliente/servidor básico

Haciendo uso del código facilitado, y tomando como referencia la teoría vista en clase sobre el uso de `Java RMI`, implementa las clases correspondientes a un cliente y un servidor para que, entre ellos, se puedan intercambiar mensajes (`String`) de forma interactiva.

Un posible resultado del funcionamiento del chat sería el siguiente, donde se muestran las salidas del lado cliente y servidor, previo registro (`rmiregistry`) con nombre asociado; en este caso con nombres `julio` y `vega` respectivamente.

Lanzamiento de la parte del servidor:

```
Introduzca su nombre y pulsa Enter:
julio
Objeto Remoto del Chat preparado
```

Lanzamiento de la parte del cliente:

```
Introduzca su nombre y pulsa Enter:
vega
[julio] conectado
[vega] conectado
Comienzo el envío de mensajes
[vega] Comienzo el envío de mensajes
Contesto desde la parte del servidor
[julio] Contesto desde la parte del servidor
```

Problemas con *java.policy* La mayoría de los problemas que nos podemos encontrar a la hora de ejecutar una aplicación de *RMI*, son los concernientes a los permisos que tiene establecido Java en su fichero `java.policy`, que podemos encontrar en `/etc/java-?` según la librería que estemos usando.

Por ejemplo, si usamos la librería proporcionada por *Oracle*, tendremos que acceder al fichero ubicado en `/etc/java-8-oracle/security/java.policy`.

Ejercicio 2. Chat múltiple entre clientes

Implementa la funcionalidad necesaria para que el servidor pueda atender a múltiples clientes de forma concurrente.

5. Entrega

Para implementar cada ejercicio deberás crear su correspondiente proyecto. Los nombres de los proyectos serán, respectivamente:

- `tulogin-p7-ej1`
- `tulogin-p7-ej2`

Para la entrega de esta práctica, deja todo el código fuente de los dos proyectos en un único fichero `chat.zip` que deberás dejar adjunto a la tarea (Moodle).