

SD	Sistemas Distribuidos
17/18	Prácticas
#2	Introducción a la tecnología RMI

Introducción

Se trata de un API del lenguaje Java que nos permite llamar a métodos de objetos que se encuentran en una máquina virtual diferente. Esta puede estar en la misma máquina física o en otra máquina.

Es una solución de más alto nivel que las mencionadas anteriormente (sockets, rpc).

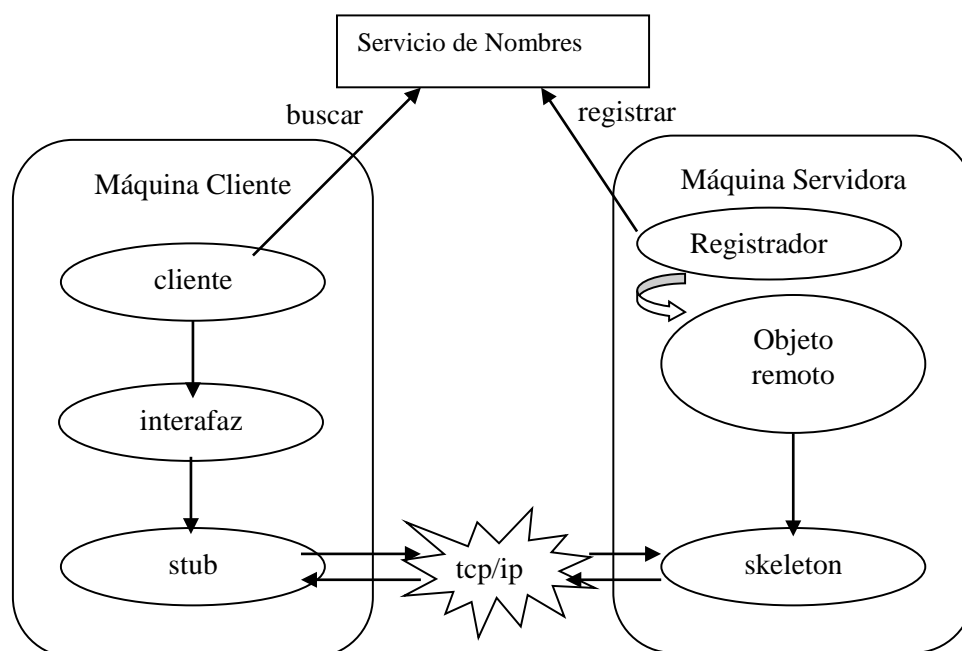


Figura 1. Esquema básico de RMI

El paquete básico para utilizar RMI se llama `java.rmi` y se encuentra en el `j2sdk` que proporciona SUN en el archivo `jdk1.7.0.07\jre\lib\rt.jar`.

Existen diferentes elementos para establecer una comunicación mediante RMI como se muestra en la figura 1.

- **Interfaz** → Define los métodos de nuestro objeto. Debe extender de la interfaz `java.rmi.Remote`.
- **Objeto Remoto** → Objeto que se encuentra localizado en una máquina remota. Implementa la interfaz que define la estructura de nuestro objeto.
- **Cliente** → Objeto desde el que queremos invocar a los métodos de un objeto remoto. Este utilizará la interfaz para comunicarse con el stub.
- **Stub** → Clase proxy que nos permite la comunicación con el objeto remoto. Implementa la interfaz.
- **Skeleton** → Recibe los mensajes y se encarga de invocar al objeto remoto y de devolver el resultado.

- Registrador → Objeto que se encarga de registrar el objeto remoto en el servicio de nombres para que pueda ser accedido desde el cliente.
- Servicio de nombres → Nos permite asociar nombres lógicos a objetos. Independiza la ubicación del objeto respecto al cliente.

Java RMI define un servicio de nombres muy básico y restringido. La sintaxis de nombrado de la URL sigue el siguiente esquema.

`//maquina:puerto/nombreObjeto`

Donde *máquina* y *puerto* hacen referencia a la máquina donde corre el servicio de nombres. Este servicio de nombres se debe ejecutar en la misma máquina que el objeto registrador y el objeto remoto por motivos de seguridad.

Los pasos a realizar para crear una aplicación Cliente/Servidor utilizando RMI son:

1. Definir una interfaz remota
2. Implementar la interfaz remota
3. Generar stubs (proxy en el cliente) y skeleton (proxy en el servidor)
4. Generar el objeto para registrar el objeto remoto
5. Crear el cliente que va a invocar al objeto remoto
6. Iniciar el registro y registrar el objeto
7. Ejecutar el cliente

Definir interfaz remota

Para poder llamar a los métodos de un objeto de forma remota, el objeto remoto debe heredar de la clase `java.rmi.Remote`.

Para ello debemos importar en nuestra clase java la interfaz `Remote`.

Una interfaz no es más que la declaración de métodos que contendrá un objeto. La interfaz remota es un conjunto de métodos que pueden ser invocados remotamente por un cliente.

```
import java.rmi.Remote;

public interface InterfazRemoto extends Remote
{
    public int suma (int a, int b) throws java.rmi.RemoteException;
    public int multiplicacion (int a, int b) throws java.rmi.RemoteException;
}
```

Implementar interfaz remota

A continuación implementaremos la interfaz que hemos definido. Esta será la clase real que proporciona la implementación para métodos definidos en la interfaz remota.

```
import java.io.Serializable;
import java.rmi.*;
import java.rmi.server.*;

public class ObjetoRemoto extends UnicastRemoteObject
implements InterfazRemoto, Serializable
{
    public ObjetoRemoto () throws RemoteException
    {
        super();
    }
    public int suma(int a, int b)
    {
        return a+b;
    }

    public int multiplicacion(int a,int b)
    {
        return a*b;
    }
}
```

Nuestro objeto también debe implementar la interfaz *Serializable* para que podamos enviar el objeto a través de la red si fuera necesario (Serializar consiste en poder guardar el objeto en algún medio, archivo, de tal forma que pueda ser recuperado en el estado en que se encontraba cuando se guardó).

Compilación de la interfaz y del objeto remoto

Primero compilaremos la interfaz. En la variable de entorno PATH se debe incluir la ruta de acceso a los ejecutables del j2sdk.

```
$ export PATH=$PATH:ruta_j2sdk/bin
```

En el caso de que no se sepa la ruta_j2sdk se puede utilizar el siguiente comando que proporcionará de forma automática la ruta.

```
$ which javac
```

Ahora compilamos la interfaz. Nos situamos en el directorio donde se encuentran las clases del servidor.

```
$ javac InterfazRemoto.java
```

Ahora compilaremos el objeto remoto. Debemos incluir en la variable de entorno CLASSPATH la ruta a la interfaz remota para compilar el objeto remoto.

```
$ export CLASSPATH=$CLASSPATH:ruta_interfaz  
$ javac ObjetoRemoto.java
```

(ruta_interfaz es la ruta al directorio donde se encuentra la interfaz compilada. Un ejemplo sería export CLASSPATH=\$CLASSPATH:/Escritorio/rmi/servidor/interfaz

Generar stubs y skeletons

Se pueden generar los stubs y skeletons, después de haber compilado las clases, con la herramienta *rmic*.

```
$ rmic ObjetoRemoto
```

La utilidad *rmic* la podemos encontrar en el directorio *bin* del *j2sdk*. El cliente utilizará la interfaz y el Stub para comunicarse por lo que lo incluiremos en un archivo jar para tenerlo más localizado.

```
$ jar cvf cliente.jar InterfazRemoto.class ObjetoRemoto_Stub.class
```

Generar el objeto para registrar el objeto remoto

A continuación debemos crear una clase java que se encargue del registro de los objetos en el Servidor RMI. Para registrar el objeto utilizaremos la clase *Naming.rebind*.

```
import java.rmi.*;  
  
public class Registro {  
  
    public static void main (String args[])  
    {  
        String URLRegistro;  
        try  
        {  
            System.setSecurityManager(new RMISecurityManager());  
            ObjetoRemoto objetoRemoto = new ObjetoRemoto();  
            URLRegistro = "/ObjetoRemoto";  
            Naming.rebind (URLRegistro, objetoRemoto);  
            System.out.println("Servidor de objeto preparado.");  
        }  
        catch (Exception ex)  
        {  
            System.out.println(ex);  
        }  
    }  
}
```

Al método *rebind* le pasamos una cadena para identificar al objeto y la instancia del objeto. En caso de estar registrado se sustituye por el nuevo objeto.

Debemos instalar un gestor de seguridad para que permita la descarga dinámica del Stub por parte del Servicio de nombres (rmiregistry).

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

Como el registro (rmiregistry) se encuentra localizado en la misma máquina que el registrador y el objeto remoto no hace falta que al hacer el rebind indiquemos el host y el puerto. Por defecto cogerá localhost y el 1099.

Una vez creado el registro se procederá a su compilación.

```
$ javac Registro.java
```

Crear el cliente

El último paso es crear un cliente encargado de invocar los métodos del objeto remoto.

El cliente realizará una búsqueda en el registro del host donde tengamos el objeto remoto y obtiene una referencia a dicho objeto. Para ello lo primero que se debe hacer es localizar el objeto remoto solicitándoselo al Servidor RMI.

```
InterfRemoto objetoRemoto=null;  
String servidor = "rmi://" + host + ":" + port + "/ObjetoRemoto";  
objetoRemoto =(InterfazRemoto) Naming.lookup(servidor);
```

Para ello utilizaremos el método *lookup* de la clase *Naming* pasándole como parámetro la *URL* con el protocolo RMI del servidor remoto e indicando el objeto a instanciar.

Este método devuelve un *Remote* que no es más que una referencia al objeto remoto (stub). Debemos hacer una conversión de tipo a la interfaz que hemos creado.

En este momento podemos llamar a los métodos del objeto remoto como si fuera un objeto local.

```
System.out.print ("2+3=");  
System.out.println (objetoRemoto.suma(2, 3));
```

Como en el caso anterior es necesario crear un contexto de seguridad que me permita bajarme el código del servidor. Esto lo haremos antes de la búsqueda del objeto remoto.

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

Una vez creado el cliente procederemos a su compilación. El cliente utiliza la interfaz y el stub para acceder al objeto remoto. Por este motivo estas clases deben estar localizadas en el

CLASSPATH local del cliente. Estas clases las habíamos agrupado anteriormente en el archivo cliente.jar. Debemos copiar el archivo en la máquina local del cliente e indicar la ruta a él en el CLASSPATH (existe una forma de descargarlo dinámicamente sin tener que introducirlo en el CLASSPATH pero necesitaríamos un servidor Web o FTP). Si el archivo cliente.jar estuviera localizado en /home/guest

```
$ export CLASSPATH=$CLASSPATH:/home/guest/cliente.jar
```

Para asegurarnos que la variable CLASSPATH tiene el contenido deseado ejecutaremos

```
$ echo $CLASSPATH
```

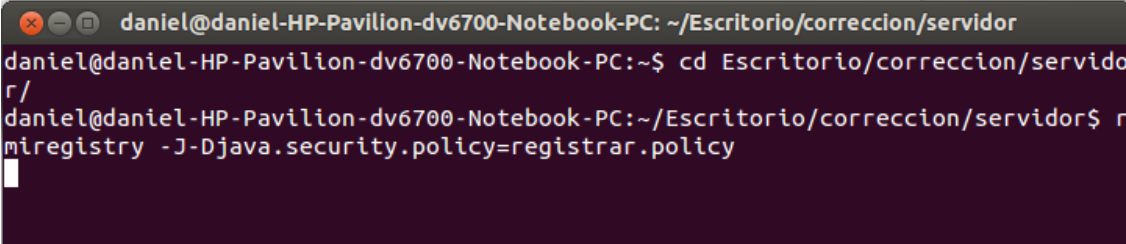
Registrar el objeto

Ahora debemos registrar nuestro objeto remoto. Registraremos el objeto mediante un nombre, con el cual realizaremos la búsqueda del objeto remoto desde el cliente.

Para poder registrar nuestro objeto como remoto debemos lanzar el servidor de nombres RMI que nos proporciona el *J2SDK*. El Servidor se encuentra en *J2SDK_1... /bin* y se llama *rmiregistry*. Para ello abriremos una consola y lanzaremos la siguiente orden en la ruta en la cual está el servidor.

```
$ rmiregistry -J-Djava.security.policy=registrar.policy
```

Cuando se ejecute el comando, se quedará de la siguiente forma:



```
daniel@daniel-HP-Pavilion-dv6700-Notebook-PC: ~/Escritorio/correccion/servidor
daniel@daniel-HP-Pavilion-dv6700-Notebook-PC:~$ cd Escritorio/correccion/servidor/
daniel@daniel-HP-Pavilion-dv6700-Notebook-PC:~/Escritorio/correccion/servidor$ rmiregistry -J-Djava.security.policy=registrar.policy
```

La política de seguridad es necesaria debido al modelo de seguridad de Java 2. Nuestro archivo de políticas se llamará *registrar.policy* con el siguiente contenido:

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

En nuestro ejemplo daremos permisos a todos para poder instanciar nuestro objeto (no recomendable en entornos de producción). En el *j2sdk* existe una utilidad para tratar las políticas de seguridad (*policytool.exe*). Este archivo de seguridad debemos situarlo en el directorio desde el cual lanzamos el *rmiregistry*.

Es importante que en la consola donde vayamos a ejecutar el rmiregistry la variable de entorno CLASSPATH esté vacía.

```
$ unset CLASSPATH
```

A continuación abrimos otra consola y nos situaremos en el directorio donde tengamos el objeto Registro y lo ejecutamos.

```
$ java -Djava.security.policy=registrar.policy Registro
```

Necesitamos una política de seguridad para permitir a las aplicaciones utilizar y acceder a los puertos en los que se conectan los objetos remotos. Por otro lado debemos indicar la URL donde se encuentran los stub para que el registro (rmiregistry) tenga acceso a ellos (/home/alumno/rmi/servidor).

Si se produce un error al ejecutar el Registro lo más probable es que esté mal puesto el CLASSPATH.

Ejecutar el cliente

Una vez tenemos la aplicación para registrar y el Servidor RMI funcionado lanzaremos la aplicación cliente. Para ello lanzaremos una consola y nos aseguraremos que en el CLASSPATH tengamos la ruta completa a cliente.jar con el nombre del archivo incluido.

Primero se producirá la compilación.

```
$javac clienteRMI
```

En caso de que se produzca el error “symbol : class InterfazRemoto” verifica la ruta del CLASSPATH referente a cliente.jar. este error es producido porque no encuentra dicha clase.

```
clienteRMI.java:11: error: cannot find symbol
    private int pedirNumeros(int operacion, int resultado, InterfazRemoto objeto
Remoto)
                                                    ^
    symbol:   class InterfazRemoto
    location: class clienteRMI
clienteRMI.java:69: error: cannot find symbol
    InterfazRemoto objetoRemoto = null;
    ^
    symbol:   class InterfazRemoto
    location: class clienteRMI
clienteRMI.java:77: error: cannot find symbol
        objetoRemoto = (InterfazRemoto) Naming.lookup(servidor);
                        ^
    symbol:   class InterfazRemoto
    location: class clienteRMI
```

```
$ java -Djava.security.policy=registrar.policy clienteRMI localhost 1099
```

En el caso de que el servidor este en otra máquina se cambiaría localhost por la ip máquina de servidor.

Si se produce el error.net.ConnectionException: Conexión rehusada verifica la ip del servidor y el puerto utilizado.

Debemos introducir la política de seguridad utilizada hasta el momento.

Ejemplo de RMI desplegado en local.

The screenshot shows two terminal windows. The left window is the client's terminal, and the right window is the server's terminal. The client terminal shows the execution of the `clienteRMI` program, which prompts for operations (Sumar, Multiplicar) and operands. The server terminal shows the execution of the `servidor` program, which registers the `ObjetoRemoto` class and handles the requests from the client.

```
daniel@daniel-HP-Pavilion-dv6700-Notebook-PC: ~/Escrito
mic ObjetoRemoto.java
error: Class ObjetoRemoto.java not found.
1 error
daniel@daniel-HP-Pavilion-dv6700-Notebook-PC:~/Escrito
mic ObjetoRemoto
error: Class ObjetoRemoto not found.
1 error
daniel@daniel-HP-Pavilion-dv6700-Notebook-PC:~/Escrito
avac ObjetoRemoto.java
daniel@daniel-HP-Pavilion-dv6700-Notebook-PC:~/Escrito
mic ObjetoRemoto
daniel@daniel-HP-Pavilion-dv6700-Notebook-PC:~/Escrito
ar cvf cliente.jar InterfazRemoto.class ObjetoRemoto_S
manifiesto agregado
agregando: InterfazRemoto.class(entrada = 243) (salida 2
agregando: ObjetoRemoto_Stub.class(entrada = 2024) (sa
)
daniel@daniel-HP-Pavilion-dv6700-Notebook-PC:~/Escrito
avac Registro.java
daniel@daniel-HP-Pavilion-dv6700-Notebook-PC:~/Escrito
ava -Djava.security.policy=registrar.policy Registro
Servidor de objeto preparado.

daniel@daniel-HP-Pavilion-dv6700-Notebook-PC: ~/Escritorio/correccion/cliente
va -Djava.security.policy=registrar.policy clienteRMI localhost 1099
[1] Realizar operacion
[2] Salir
Indique la opcion a realizar:
1
[1] Sumar
[2] Multiplicar
Indica la operacion a realizar:
2
Introduzca el primer operando [0-9]:
2
Introduzca el segundo operando [0-9]:
2
El resultado es: 4
Desea realizar otra operacion? [s,n]:
[1] Realizar operacion
[2] Salir
Indique la opcion a realizar:
2
daniel@daniel-HP-Pavilion-dv6700-Notebook-PC:~/Escritorio/correccion/cliente$

daniel@daniel-HP-Pavilion-dv6700-Notebook-PC: ~/Escritorio/correccion/servidor
daniel@daniel-HP-Pavilion-dv6700-Notebook-PC:~$ cd Escritorio/correccion/servido
r/
daniel@daniel-HP-Pavilion-dv6700-Notebook-PC:~/Escritorio/correccion/servidor$ r
miregistry -J-Djava.security.policy=registrar.policy
```

Bibliografía

M.L.Liu. Computación Distribuida. Fundamentos y Aplicaciones. Addison Wesley 2004.
 Allamaraju, S., Beust, C., Davis, J., Jewell, T., Johnson, R., Longshaw, A., Nagappan, R., Dr.
 Sarang, P.G., Toussaint, A. Tyagi, S., Watson, G., Wilcox, M., Williamson, A., O'Connor, D.:
 Programación Java Server con J2EE Edición 1.3. Anaya Multimedia (2002)
<http://www.programacion.com/java/tutorial/rmi>