

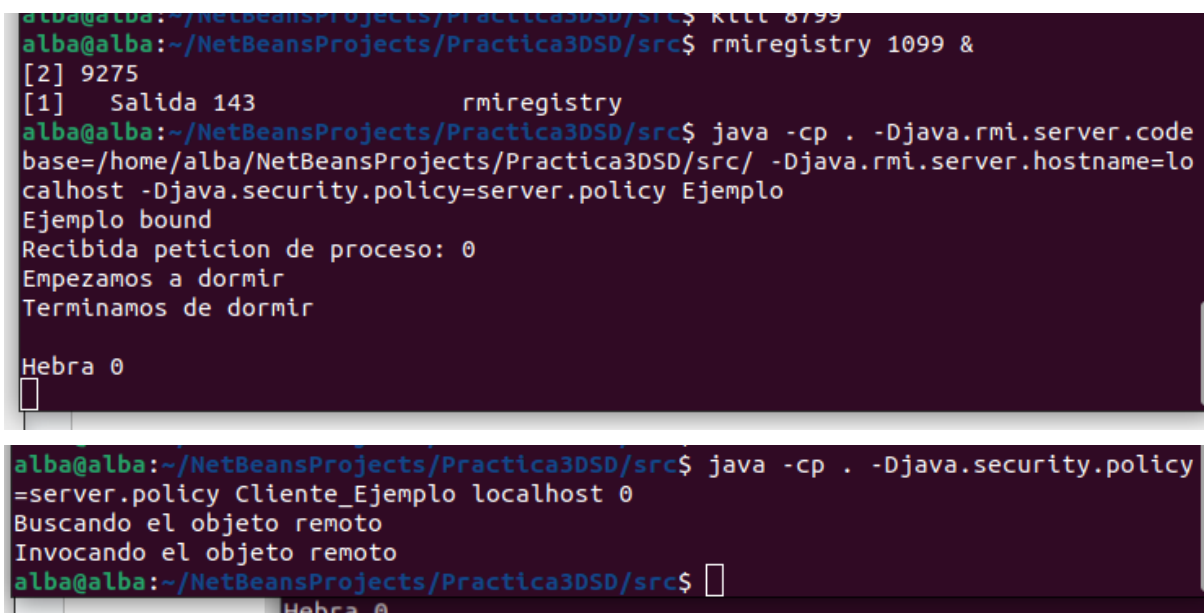
# Práctica 3 DSD: RMI

Alba Guisado Farnes

## 1. Ejemplo

En este ejemplo el servidor imprime el argumento enviado por el cliente en la llamada. En caso de ser un 0 el servidor espera un tiempo de 5 segundos para volver a imprimir el mensaje. Podemos ver su ejecución:

1. rmiregistry
2. Servidor: java -cp . -Djava.rmi.server.codebase=file:///...  
-Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Ejemplo
3. Cliente: java -cp . -Djava.security.policy=server.policy Cliente\_Ejemplo



```
alba@alba:~/NetBeansProjects/Practica3DSD/src$ rmiregistry 1099 &
[2] 9275
[1] Salida 143 rmiregistry
alba@alba:~/NetBeansProjects/Practica3DSD/src$ java -cp . -Djava.rmi.server.code
base=/home/alba/NetBeansProjects/Practica3DSD/src/ -Djava.rmi.server.hostname=lo
calhost -Djava.security.policy=server.policy Ejemplo
Ejemplo bound
Recibida peticion de proceso: 0
Empezamos a dormir
Terminamos de dormir

Hebra 0
[ ]

alba@alba:~/NetBeansProjects/Practica3DSD/src$ java -cp . -Djava.security.policy
=server.policy Cliente_Ejemplo localhost 0
Buscando el objeto remoto
Invocando el objeto remoto
alba@alba:~/NetBeansProjects/Practica3DSD/src$ [ ]
Hebra 0
```

## 2. Ejemplo:Multihebra

En este caso lanzaremos varias hebras para que realicen la tarea. Esto permite la gestión concurrente de las peticiones de los clientes.

He ejecutado primeramente una única hebra, luego 11. Cómo podemos ver en el servidor cuando una hebra termina por 0, ya sea 0 o 10 por ej, esta se duerme durante un tiempo de 5 segundos. Se ve claramente en la primera ejecución de una única hebra.

Las demás hebras únicamente entran y salen del servidor, estos mensajes no se solapan. Cada hebra entra y sale sin solaparse.

Podemos ver su ejecución:

4. rmiregistry

5. Servidor: `java -cp . -Djava.rmi.server.codebase=file:///...  
-Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy` Ejemplo
6. Cliente: `java -cp . -Djava.rmi.server.codebase=file:///...  
-Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy`  
Cliente\_Ejemplo\_Multithread localhost numHebras

[illegible]

```
albagalba:~/NetBeansProjects/Practica3DSD-Ejemplo2/src$ java -cp . -Djava.rmi.se
rver.codebase=/home/alba/NetBeansProjects/Practica3DSD-Ejemplo2/src/ -Djava.rmi.
server.hostname=localhost -Djava.security.policy=server.policy Ejemplo
Ejemplo bound

Entra Hebra Cliente 0
Empezamos a dormir
Terminamos de dormir
Sale Hebra Cliente 0

Entra Hebra Cliente 2
Sale Hebra Cliente 2

Entra Hebra Cliente 6
Sale Hebra Cliente 6

Entra Hebra Cliente 1
Sale Hebra Cliente 1

Entra Hebra Cliente 7
Sale Hebra Cliente 7

Entra Hebra Cliente 0
Empezamos a dormir

Entra Hebra Cliente 5
Sale Hebra Cliente 5

Entra Hebra Cliente 3

Entra Hebra Cliente 8
Sale Hebra Cliente 8
Sale Hebra Cliente 3

Entra Hebra Cliente 10
Empezamos a dormir
```

Cuando añadimos el modificador `synchronized` en el método de la implementación remota espera a que las hebras que duermen terminen y salgan del servidor, mientras que sin este modificador el servidor ejecutaba otras hebras mientras estas dormían. Veamos cómo se ejecutaría el siguiente ejemplo:

```

Terminamos de dormir
Sale Hebra Cliente 0
Terminamos de dormir
Sale Hebra Cliente 10
^Calba@alba:~/NetBeansProjects/Practica3DSD-Ejemplo2/src$ javac *.java
alba@alba:~/NetBeansProjects/Practica3DSD-Ejemplo2/src$ java -cp . -Djava.rmi.se
rver.codebase=/home/alba/NetBeansProjects/Practica3DSD-Ejemplo2/src/ -Djava.rmi.
server.hostname=localhost -Djava.security.policy=server.policy Ejemplo
Ejemplo bound

Entra Hebra Cliente 0
Empezamos a dormir
Terminamos de dormir
Sale Hebra Cliente 0

Entra Hebra Cliente 9
Sale Hebra Cliente 9

Entra Hebra Cliente 1
Sale Hebra Cliente 1

Entra Hebra Cliente 5
Sale Hebra Cliente 5

Entra Hebra Cliente 0
Empezamos a dormir
Terminamos de dormir
Sale Hebra Cliente 0

Entra Hebra Cliente 4
Sale Hebra Cliente 4

Entra Hebra Cliente 3
Sale Hebra Cliente 3

Entra Hebra Cliente 7
Sale Hebra Cliente 7

```

Esto es lo que sucede al ejecutar primero una hebra y luego otro cliente con 11. Se puede ver como se espera a que la hebra 0 termine de dormir, cosa que antes no pasaba.

### 3. Ejemplo

El programa contador es un ejemplo de cliente-servidor. En este ejemplo se crea por una parte el objeto remoto y por otra el servidor. El servidor tiene micontador como objeto instanciado. Este programa pone el valor inicial en el contador del servidor, invoca al método 1000 veces e imprime el valor final del contador junto con el tiempo medio de respuesta.

```

alba@alba:~/NetBeansProjects/Practica3DSD-Ejemplo3/src$
alba@alba:~/NetBeansProjects/Practica3DSD-Ejemplo3/src$ java -cp . -Djava.rmi.se
rver.codebase=file:///home/alba/NetBeansProjects/Practica3DSD-Ejemplo3/src/ -Dja
va.rmi.server.hostname=localhost -Djava.security.policy=server.policy servidor
Servidor RemoteException | MalformedURLExceptionor preparado

```

```

alba@alba:~/NetBeansProjects/Practica3DSD-Ejemplo3/src$ java -cp . -Djava.rmi.se
rver.codebase=file:///home/alba/NetBeansProjects/Practica3DSD-Ejemplo3/src/ -Dja
va.security.policy=server.policy cliente localhost
Poniendo contador a 0
Incrementando...
Media de las RMI realizadas = 0.113 msecs
RMI realizadas = 1000

```

## 4. Ejercicio

La práctica consiste en realizar un servidor replicado para recibir donaciones. Para ello he creado una interfaz `Donaciones_I` de los objetos que controlan las donaciones realizadas por parte de los clientes.

También cuento con una clase cliente donde podremos almacenar su nombre, contraseña y el dinero total donado por él.

```

public class ClienteRegistrado implements Serializable{

    private String nombre;
    private String password;
    private double totalDonado;

    public ClienteRegistrado(String nombre, String password){
        this.nombre = nombre;
        this.password = password;
        this.totalDonado = 0;
    }

    public String getNombre() {
        return nombre;
    }

    public String getPassword() {
        return password;
    }

    public double getTotalDonado() {
        return totalDonado;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public void donar(double cantidad) {
        totalDonado += cantidad;
    }
}

```

Para poder llevar a cabo la actualización de los datos de los dos servidores en cada clase llamamos al otro objeto remoto para consultar sus datos. Para ello le hemos pasado como parámetro al constructor el nombre del objeto de la réplica a la que queremos acceder y el puerto en el que este se encuentra. Nos ayudaremos del método `getReplica` para que podamos acceder a nuestra réplica siempre que lo deseemos.

La clase `Donaciones_I` consta de los siguientes métodos:

```

/**
 * Conseguimos tener el objeto de la otra réplica para futuras actualizaciones
 * @param nombre Nombre de la réplica.
 * @param puerto Puerto donde se encuentra la réplica
 * @return El objeto de la réplica.
 * @throws RemoteException Si ocurre un error durante la comunicación remota.
 */
public Donaciones_I getReplica(String nombre, int puerto) throws RemoteException;

```

Así conseguimos tener acceso al otro objeto remoto.

```

/**
 * Agregamos un nuevo cliente en el servidor
 * @param nombre Nombre del cliente.
 * @param password COntraseña del cliente.
 * @throws RemoteException Si ocurre un error durante la comunicación remota.
 */
public void nuevoCliente(String nombre, String password) throws RemoteException;

```

Añadimos un nuevo cliente a nuestro array de clientes del servidor

```

/**
 * Registra a un cliente
 * @param nombre Nombre del cliente.
 * @param password Contraseña del cliente
 * @return true si se ha registrado, false si ya estaba registrado.
 * @throws RemoteException Si ocurre un error durante la comunicación remota.
 */
public boolean registrarCliente(String nombre, String password) throws RemoteException;

```

Comprobamos previamente que el cliente no existe en ningún servidor y luego llamamos al nuevoCliente del servidor con menos clientes en ese momento.

```

/**
 * Identifica un usuario
 * @param nombre Nombre del cliente.
 * @param password Contraseña del cliente.
 * @return true si se ha identificado, false si el cliente no existe.
 * @throws RemoteException Si ocurre un error durante la comunicación remota.
 */
public boolean identificarCliente(String nombre, String password) throws RemoteException;

```

Comprobamos si existe el cliente en algún servidor. Esto sirve para que el usuario pueda iniciar sesión sin tener que volver a registrarse en el sistema usando su nombre y contraseña.

```

/**
 * Realiza una donación
 * @param nombre Nombre del cliente que realiza la donación.
 * @param password Contraseña del cliente.
 * @param cantidad Cantidad de la donación.
 * @return true si se ha realizado la donación, false si el cliente no está registrado.
 * @throws RemoteException Si ocurre un error durante la comunicación remota.
 */
public boolean donar(String nombre, String password, int cantidad) throws RemoteException;

```

```

/**
 * Obtiene el total del servidor
 * @return Total de donaciones del servidor.
 * @throws RemoteException Si ocurre un error durante la comunicación remota.
 */
public int getTotalDonadoServidor() throws RemoteException;

/**
 * Obtiene el total donado en ambas réplicas del servidor.
 * @return El total donado en ambas réplicas.
 * @throws RemoteException Si ocurre un error durante la comunicación remota.
 */
public int getTotalDonado() throws RemoteException;

/**
 * Obtiene el número de clientes registrados en ambas réplicas del servidor.
 * @return El número de clientes registrados en ambas réplicas.
 * @throws RemoteException Si ocurre un error durante la comunicación remota.
 */
public int getNumClientes() throws RemoteException;

/**
 * Getter de clientes
 * @return Devuelve el número de clientes registrado en la réplica.
 * @throws RemoteException Si ocurre un error durante la comunicación remota.
 */
public ArrayList<ClienteRegistrado> getClientes() throws RemoteException;

```

Nuestro cliente cuenta con un menú para poder realizar las siguientes acciones. Primero nos encontramos con un primer menú para poder registrarte o en caso de ya habernos registrado para iniciar sesión indicando nuestro nombre y contraseña. Si esto ha sido posible pasaremos a un segundo menú donde será posible realizar una nueva donación, comprobar todo el dinero donado en nuestro sistema y por último salir del propio sistema.

Podrá verlo más detalladamente en el propio código aunque le dejo un esquema de cómo funcionaría:

0:Registarse

1:Iniciar sesión

-si ha podido iniciar sesion:

1:donar

2:consultar donaciones

3:salir

Es necesario utilizar el comando `javac *.java` para su compilación.

A continuación realizaré un ejemplo de uso:

- 1\* Activar el primer servidor mediante el comando: `java -cp . -Djava.rmi.server.codebase=file:///... -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Replica1`

```
^Calba@alba:~/NetBeansProjects/Practica3DSD-Ejercicio/src$ java -cp . -Djava.rmi
.server.codebase=file:///home/alba/NetBeansProjects/Practica3DSD-Ejercicio/src/
-Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Repli1
Activo réplica 1
No se ha podido identificado el cliente: a
Registrando cliente: a
Donación del cliente a de : 12
```

- 2\* Activar el segundo servidor mediante el comando: `java -cp . -Djava.rmi.server.codebase=file:///... -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Replica2`

```
^Calba@alba:~/NetBeansProjects/Practica3DSD-Ejercicio/src$ java -cp . -Djava.rmi
.server.codebase=file:///home/alba/NetBeansProjects/Practica3DSD-Ejercicio/src/
-Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Repli2
Activo réplica 2
```

- 3\* Iniciar el cliente mediante el comando: `java -cp . -Djava.rmi.server.codebase=file:///... -Djava.rmi.server.hostname=localhost -Djava.security.policy=server.policy Cliente`

Ya podremos interactuar con el servidor. Primeramente sería necesario registrarse. Vea un ejemplo:



```
alba@alba: ~/NetBeansProjects/Practica3DSD-Ejercicio/src
Conexión establecida con los servidores

0: Registrarse
1: Iniciar sesión
Ingrese una opción: 0

Introduce tu nombre:
a

Introduce una contraseña
a
Usuario registrado
1: Realizar una donación
2: Consultar total donado
3: Salir
Ingrese una opción: 1
Ingrese la cantidad a donar: 12
Donación realizada
1: Realizar una donación
2: Consultar total donado
3: Salir
Ingrese una opción: 2
Total donado en ambas réplicas: 12

1: Realizar una donación
2: Consultar total donado
3: Salir
Ingrese una opción: 3
Saliendo...
alba@alba:~/NetBeansProjects/Practica3DSD-Ejercicio/src$
```

Una vez que ya está registrado en el sistema será posible poder acceder de nuevo únicamente iniciando sesión. Véalo a continuación:

```
alba@alba: ~/NetBeansProjects/Practica3DSD-Ejercicio/src
java.rmi.server.hostname=localhost -Djava.security.policy=server
Conexión establecida con los servidores

0: Registrarse
1: Iniciar sesión
Ingrese una opción: 1

Introduce tu nombre:
a

Introduce una contraseña
a
Usuario identificado
1: Realizar una donación
2: Consultar total donado
3: Salir
Ingrese una opción: 1
Ingrese la cantidad a donar: 45
Donación realizada
1: Realizar una donación
2: Consultar total donado
3: Salir
Ingrese una opción: 2
Total donado en ambas réplicas: 57

1: Realizar una donación
2: Consultar total donado
3: Salir
Ingrese una opción: 3
Saliendo...
alba@alba: ~/NetBeansProjects/Practica3DSD-Ejercicio/src$
```