Centro de Enseñanza Técnica Industrial

Desarrollo de Software

RMI Paso de Mensajes

Jesús Alberto Aréchiga Carrillo

22310439 5N

Profesor

José Francisco Pérez Reyes

Noviembre 2024

Guadalajara, Jalisco

Introducción

Java RMI (Remote Method Invocation) es una tecnología que permite la invocación de métodos en objetos que residen en diferentes máquinas dentro de una red, como si estuvieran en la misma máquina virtual. RMI es una implementación de comunicación distribuida en Java, basada en el paradigma cliente-servidor. Permite a los desarrolladores enfocarse en la lógica de negocio sin preocuparse por los detalles complejos de la comunicación en red.

RMI utiliza un registro centralizado, conocido como **RMI Registry**, donde los objetos remotos se registran para que los clientes puedan localizarlos y acceder a ellos. Los objetos remotos deben implementar una interfaz que extiende Remote y se comunican utilizando la serialización de objetos para enviar datos a través de la red. Además, RMI emplea el protocolo TCP/IP para garantizar una comunicación confiable y permite manejar objetos complejos gracias a la serialización.

RMI es útil para aplicaciones distribuidas, como sistemas cliente-servidor o aplicaciones que necesitan dividir su lógica en múltiples nodos. Aunque es una tecnología eficiente y bien integrada en Java, requiere un manejo cuidadoso de excepciones y la configuración de políticas de seguridad, especialmente cuando se ejecuta en entornos de red abierta.

Desarrollo

Los códigos para utilizar son:

MiInterfazRemota.java:

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

public interface MiInterfazRemota extends Remote {
    public void miMetodo1() throws RemoteException;
    public int miMetodo2() throws RemoteException;
    public String otroMetodo() throws RemoteException;
}
```

MiClaseRemota.java:

```java
import java.net.MalformedURLException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.server.UnicastRemoteObject;

public class MiClaseRemota extends UnicastRemoteObject implements
    MiInterfazRemota {

    public MiClaseRemota() throws RemoteException {
```

```
        // Código del constructor
    }

    @Override
    public void miMetodo1() throws RemoteException {
        // Aquí ponemos el código que queramos
        System.out.println("Estoy en miMetodo1()");
    }

    @Override
    public int miMetodo2() throws RemoteException {
        // Aquí ponemos el código que queramos
        return 5;
    }

    @Override
    public String otroMetodo() {
        return "a";
    }

    public static void main(String[] args) {
        try {

            String ip = "192.168.1.85";
            int port = 1234;

            LocateRegistry.createRegistry(port);

            MiInterfazRemota mir = new MiClaseRemota();

            java.rmi.Naming.rebind("//" + ip + ":" + port + "/PruebaRMI",
mir);
        } catch (MalformedURLException | RemoteException e) {
            System.out.println(e);
        }
    }
}
```

MiClienteRMI.java:

```
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
public class MiClienteRMI {
    public static void main(String[] args) {
    try {
        String ip = "192.168.1.85";
```

```
        String puerto = "1234";
        MiInterfazRemota mir =
        (MiInterfazRemota)Naming.lookup("//" + ip + ":" + puerto +
"/PruebaRMI");

        // Imprimimos miMetodo1() tantas veces como devuelva miMetodo2()
        for (int i = 1; i <= mir.miMetodo2(); i++) mir.miMetodo1();

        } catch (MalformedURLException | NotBoundException |
RemoteException e) {
            System.out.println("Error: " + e);
        }
        System.out.println("Mensajes enviados");

    }
}
```

En este caso MiClaseRemota es la que tiene los métodos públicos que se podrán utilizar desde el cliente.

Se ejecuta el código del servidor.

```
Servidor esperando mensajes...
```

En este punto el servidor ya está esperando a que el cliente se inicie para ejecutar los métodos.

```
Mensajes enviados
```

Ahora los mensajes ya fueron enviados y sólo se necesita revisar que los mensajes estén en la consola del servidor.

```
Estoy en miMetodo1()
Estoy en miMetodo1()
Estoy en miMetodo1()
Estoy en miMetodo1()
Estoy en miMetodo1()
```

Los mensajes fueron enviados desde el cliente y recibidos por el servidor correctamente.

Conclusiones:

Java RMI es una herramienta poderosa para construir sistemas distribuidos que necesitan comunicar procesos en diferentes máquinas. Al abstraer los detalles de la red, RMI permite a los desarrolladores trabajar con objetos remotos de

manera similar a los locales, facilitando la implementación de arquitecturas cliente-servidor complejas.

La seguridad y la gestión de excepciones son aspectos críticos en RMI, ya que la comunicación distribuida puede estar sujeta a problemas como fallos en la red o ataques externos. A pesar de estas consideraciones, RMI sigue siendo una opción sólida para sistemas basados en Java, aunque su popularidad ha disminuido frente a tecnologías modernas como gRPC y REST. Sin embargo, para proyectos Java específicos o legados, RMI proporciona una solución madura y eficiente.