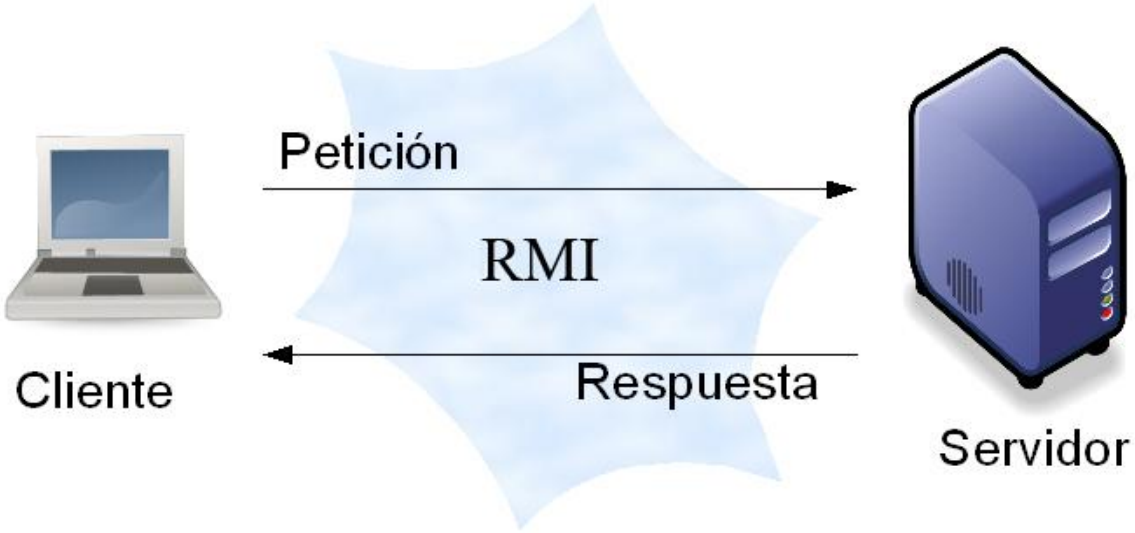
INSTITUTO POLITÉCNICO NACIONAL

Escuela Superior de Cómputo

Sistemas Distribuidos

Profesor: Chadwick Carreto Arrellano



7CM1

Práctica 5 “Objetos distribuidos RMI”

Juan Fernando León Medellín

ANTECEDENTE

1. Sistemas Distribuidos

Un **sistema distribuido** es un conjunto de computadoras interconectadas que cooperan para ejecutar tareas de manera coordinada. Estos sistemas permiten compartir recursos, mejorar la escalabilidad y la tolerancia a fallos. Un reto clave en los sistemas distribuidos es la comunicación entre procesos que pueden ejecutarse en diferentes máquinas.

2. Comunicación en Sistemas Distribuidos

Existen diversas técnicas para lograr la comunicación en sistemas distribuidos, entre ellas:

- **Sockets**: Permiten el envío y recepción de datos a través de una red.
- **RPC (Remote Procedure Call)**: Permite invocar funciones en una máquina remota como si fueran locales.
- **Mensajería y colas de mensajes**: Utilizan intermediarios para el envío de mensajes asincrónicos.
- **RMI (Remote Method Invocation)**: Permite la invocación de métodos en objetos remotos, manteniendo la orientación a objetos.

3. Java RMI (Remote Method Invocation)

Java RMI es una tecnología que permite la comunicación entre objetos ubicados en diferentes máquinas dentro de una red. Se basa en la invocación de métodos remotos, manteniendo el paradigma de programación orientada a objetos.

Características de Java RMI

- **Transparencia de ubicación**: Los objetos remotos se usan como si fueran locales.
- **Uso de interfaces remotas**: Define los métodos que pueden ser invocados remotamente.
- **Comunicación basada en serialización**: Los datos se transmiten a través de la red en formato serializado.
- **Registro y localización de objetos remotos**: A través del servicio rmiregistry.

4. Componentes de un Sistema RMI

Para que una aplicación basada en Java RMI funcione, se requieren los siguientes componentes:

a) Interfaz remota

Define los métodos que pueden ser invocados por clientes remotos. Debe extender Remote y lanzar RemoteException.

b) Implementación del objeto remoto

La clase que implementa la interfaz remota extiende UnicastRemoteObject para permitir la comunicación remota.

c) Servidor RMI

Registra la implementación del objeto remoto en el servicio rmiregistry, permitiendo que los clientes lo encuentren y lo usen.

d) Cliente RMI

Busca el objeto remoto en el rmiregistry y llama a sus métodos como si fuera un objeto local.

5. Funcionamiento de Java RMI

El proceso de ejecución de una aplicación RMI sigue estos pasos:

1. El servidor crea y registra un objeto remoto en el **registro RMI** (rmiregistry).
2. El cliente busca el objeto remoto en el registro.
3. Se establece una conexión entre el cliente y el servidor.
4. El cliente invoca métodos del objeto remoto, y el servidor devuelve los resultados.

6. Aplicaciones de Java RMI

Java RMI es utilizado en diversas aplicaciones distribuidas, como:

- Sistemas de bases de datos distribuidas.
- Aplicaciones cliente-servidor en la nube.
- Plataformas de colaboración en línea.
- Computación distribuida y procesamiento paralelo.

PLANTEAMIENTO DEL PROBLEMA

En esta práctica, se busca desarrollar y analizar una aplicación distribuida basada en Java RMI que permita la ejecución remota de una operación matemática simple (suma de dos números). Con ello, se pretende comprender el funcionamiento de RMI, sus ventajas y limitaciones en un entorno distribuido.

El problema a resolver es cómo establecer una comunicación eficiente entre un **servidor RMI**, que ofrecerá el servicio de suma, y un **cliente RMI**, que consumirá dicho servicio de manera transparente. Para lograrlo, es necesario diseñar e implementar una arquitectura basada en **interfaces remotas**, **objetos distribuidos** y **registro de objetos en el servicio RMI**.

La práctica permitirá responder preguntas clave como:

- ¿Cómo se define e implementa una interfaz remota en Java?
- ¿Qué pasos son necesarios para registrar y localizar un objeto remoto?
- ¿Cómo se establece la comunicación entre un cliente y un servidor en Java RMI?

Mediante esta implementación, se podrá evaluar la efectividad de Java RMI en la resolución de problemas de comunicación en sistemas distribuidos y explorar posibles aplicaciones en escenarios más complejos.

MATERIALES Y MÉTODOS EMPLEADOS

- Java Development Kit (JDK) versión 8 o superior.
- Entorno de Desarrollo Integrado (IDE): Eclipse, IntelliJ IDEA o NetBeans (opcional).
- Consola de comandos o terminal para ejecutar los archivos .class.
- Servicio RMI Registry, ejecutado en el puerto 1099.

PROPUESTA DE SOLUCIÓN

Para abordar el problema de la comunicación remota entre procesos en un entorno distribuido, se propone el uso de **Java Remote Method Invocation (RMI)** como tecnología para permitir la ejecución de métodos en objetos ubicados en diferentes máquinas. La solución consiste en el desarrollo de una aplicación cliente-servidor basada en RMI, donde un servidor ofrece un servicio de suma de dos números y un cliente lo consume de manera remota.

Enfoque de la solución

La solución se basa en los siguientes principios:

1. **Modularidad**: Separar la aplicación en tres componentes principales: la interfaz remota, la implementación del servicio y los programas cliente-servidor.
2. **Transparencia de ubicación**: Los clientes invocarán métodos remotos sin preocuparse por la ubicación física del servidor.
3. **Uso de RMI Registry**: Se utilizará el registro de objetos remotos (rmiregistry) para permitir la localización del servicio.
4. **Seguridad y manejo de excepciones**: Se implementarán mecanismos para manejar posibles errores en la comunicación y asegurar la estabilidad del sistema.

Pasos para implementar la solución

1. **Definir una interfaz remota (SumaRemota.java)**
    - Declarar el método sumar(int a, int b).
    - Hacer que la interfaz extienda Remote y que el método lance RemoteException.
2. **Implementar la lógica del servicio (SumaRemotaImpl.java)**
    - Desarrollar la clase SumaRemotaImpl, que extienda UnicastRemoteObject y que implemente SumaRemota.
    - Definir el método sumar(), que realizará la operación matemática y devolverá el resultado al cliente.

3. **Crear el servidor (Servidor.java)**
    - Instanciar el objeto remoto y registrarlo en rmiregistry usando Naming.rebind().
4. **Crear el cliente (Cliente.java)**
    - Conectar con el servidor utilizando Naming.lookup().
    - Invocar el método sumar() de manera remota y mostrar el resultado.
5. **Ejecutar y probar la aplicación**
    - Iniciar rmiregistry.
    - Ejecutar el servidor y el cliente para comprobar la comunicación.
    - Validar que los datos se transmitan correctamente y que la ejecución sea exitosa.

Ventajas de la solución

- **Simplicidad**: Java RMI proporciona una forma sencilla y nativa de implementar comunicación remota en Java.
- **Escalabilidad**: La solución puede expandirse para soportar operaciones más complejas.
- **Reutilización**: El servicio de suma puede extenderse para otras operaciones matemáticas o aplicaciones distribuidas.

DESARROLLO DE LA SOLUCIÓN

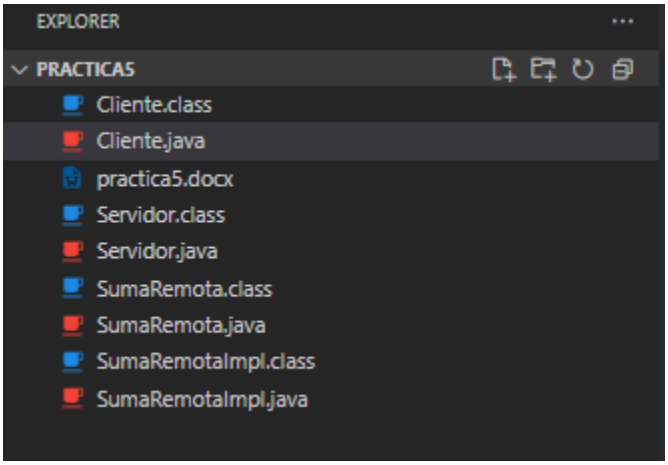
En el siguiente enlace se puede encontrar el código de la práctica:

https://github.com/JFernandoLe/SD_PRACTICA5.git

RESULTADOS

Compilamos los archivos

```
PS C:\Users\leonm\OneDrive\Escritorio\documentos\6 semestre\Sistemas Distribuidos\Practica5> javac *.java
>>
```



Inicializamos el Servidor

```
PS C:\Users\leonm\OneDrive\Escritorio\documentos\6 semestre\Sistemas Distribuidos\Practica5> java Servidor
>>
Servidor RMI listo...
```

Inicializamos el Cliente y obtenemos el resultado

```
PS C:\Users\leonm\OneDrive\Escritorio\documentos\6 semestre\Sistemas Distribuidos\Practica5> java Cliente
>>
Resultado de la suma: 8
```

CONCLUSIÓN

La implementación de **Java Remote Method Invocation (RMI)** en esta práctica permitió comprender el funcionamiento de la comunicación entre procesos en un sistema distribuido, destacando la facilidad con la que se pueden invocar métodos en objetos remotos utilizando el paradigma de programación orientada a objetos.

A través del desarrollo del **servidor RMI**, el **cliente RMI** y el uso del **registro de objetos remotos (rmiregistry)**, se demostró la capacidad de RMI para gestionar la transparencia de ubicación y la serialización de datos de manera automática. La modularidad de la solución permitió estructurar el código en tres partes principales: la interfaz remota, la implementación del servicio y los programas cliente-servidor, facilitando la reutilización y escalabilidad del sistema.

Entre las ventajas observadas se encuentra la simplicidad de la implementación en comparación con otras técnicas de comunicación en sistemas distribuidos, como **sockets** o **mensajería basada en colas**. Sin embargo, se identificaron desafíos como la necesidad de manejar correctamente las excepciones de red y la dependencia del servicio **rmiregistry**, que debe estar en ejecución para que los clientes puedan localizar los objetos remotos.

En conclusión, Java RMI es una herramienta efectiva para la comunicación remota en entornos distribuidos, proporcionando una interfaz clara y sencilla para el desarrollo de aplicaciones cliente-servidor. Esta práctica sienta las bases para la implementación de sistemas más complejos, como servicios web distribuidos, aplicaciones en la nube y computación distribuida.