Instituto Politécnico Nacional

Escuela Superior de Computo

Ingeniería en Sistemas Computacionales.

Unidad de Aprendizaje:
Sistemas Distribuidos.

Nombre del Profesor:
Carreto Arellano Chadwick.

Nombre del Alumno:

Cruz Cubas Ricardo

Grupo:
7CM1.

Practica 5:

Objetos Distribuidos.

Contenido

Introducción

Un objeto distribuido es un objeto en programación que puede ser utilizado por múltiples sistemas computacionales en una red como si estuviera en la misma máquina. Es decir, permite que un programa en una computadora invoque métodos o funciones de un objeto ubicado en otra computadora de manera transparente.

Este paradigma se basa en la programación orientada a objetos, extendiéndola al ámbito de los sistemas distribuidos. En este modelo, los objetos pueden residir en diferentes nodos de una red y comunicarse entre sí a través de protocolos específicos sin que el usuario o el desarrollador necesite preocuparse por los detalles de la comunicación en red.

Por ejemplo, Imaginemos que tienes un objeto que representa una calculadora remota. En lugar de ejecutar los cálculos localmente en tu computadora, puedes hacer que los cálculos se realicen en un servidor y simplemente obtener los resultados en tu dispositivo.

Características de los Objetos Distribuidos:

Los sistemas de objetos distribuidos presentan varias características clave:

Transparencia

La idea principal de los objetos distribuidos es que la ubicación física de un objeto sea transparente para el usuario. Esto se logra a través de:

- Transparencia de acceso: Un objeto remoto se usa de la misma manera que un objeto local.
- Transparencia de ubicación: No importa en qué máquina se encuentre el objeto.
- Transparencia de movilidad: Los objetos pueden moverse entre diferentes servidores sin afectar su funcionamiento.
- Transparencia de concurrencia: Múltiples clientes pueden acceder a un objeto al mismo tiempo.

Comunicación remota

Los objetos distribuidos utilizan diferentes mecanismos para comunicarse a través de una red. Los más comunes incluyen:

- RMI (Remote Method Invocation) en Java.

- CORBA (Common Object Request Broker Architecture) para objetos en diferentes lenguajes.
- DCOM (Distributed Component Object Model) de Microsoft.
- gRPC para comunicación eficiente entre microservicios.

Independencia de plataforma

La comunicación entre objetos distribuidos suele basarse en estándares abiertos, lo que permite que un cliente en Windows pueda interactuar con un servidor en Linux sin problemas.

Mecanismos de serialización

Cuando un objeto se encuentra en una máquina y su método es llamado desde otra, el objeto debe ser serializado (convertido en una secuencia de bytes), enviado a través de la red y luego deserializado en el otro extremo.

Seguridad y control de acceso

Dado que los objetos pueden ser accesibles desde cualquier parte de la red, es fundamental implementar mecanismos de autenticación y autorización para evitar accesos no deseados.

Arquitectura de Objetos Distribuidos

Los sistemas distribuidos basados en objetos generalmente siguen una arquitectura cliente-servidor o una arquitectura peer-to-peer.

3.1 Cliente-Servidor

En esta arquitectura:

1. El cliente solicita servicios a un objeto remoto.
2. El servidor aloja el objeto y responde a las solicitudes del cliente.

Ejemplo:

- Un cajero automático (cliente) consulta el saldo en una base de datos central (servidor).

3.2 Peer-to-Peer (P2P)

En esta arquitectura, no hay una distinción clara entre cliente y servidor, sino que cada nodo puede actuar como ambos.

Tecnologías para Objetos Distribuidos

Existen varias tecnologías que permiten implementar objetos distribuidos:

Java RMI (Remote Method Invocation)

RMI es una tecnología de Java que permite que un objeto en una máquina invoque métodos en un objeto que reside en otra máquina.

CORBA (Common Object Request Broker Architecture)

Es un estándar que permite la comunicación entre objetos en diferentes lenguajes de programación.

DCOM (Distributed Component Object Model)

Es la versión distribuida del modelo de objetos de Microsoft.

gRPC

Usa Protocol Buffers para permitir la comunicación eficiente entre servicios distribuidos.

Web Services (REST & SOAP)

Permiten la comunicación entre objetos a través de la web, utilizando JSON o XML como formato de datos.

Ventajas y Desafíos de los Objetos Distribuidos

Ventajas

- Permiten reutilizar y distribuir componentes en diferentes sistemas.
- Facilitan la escalabilidad de aplicaciones complejas.
- Son flexibles e independientes de la plataforma.

Desafíos

- Requieren una gestión compleja de la comunicación en red.
- Son más vulnerables a problemas de seguridad.
- La latencia de red puede afectar el rendimiento.

Planteamiento del Problema

En entornos distribuidos, muchas aplicaciones requieren que múltiples clientes accedan a un mismo servicio desde diferentes ubicaciones. Sin embargo, gestionar esta comunicación de manera eficiente puede ser un desafío debido a la necesidad de establecer conexiones de red, manejar la concurrencia y garantizar la integridad de los datos.

Un ejemplo común de este problema es la necesidad de realizar cálculos remotos sin que el cliente tenga que preocuparse por dónde se ejecutan estos cálculos. Tradicionalmente, los cálculos se realizan en el mismo dispositivo donde se ejecuta la aplicación, lo que puede ser ineficiente en términos de procesamiento y uso de recursos.

Problema específico:
Queremos desarrollar un sistema en el cual un cliente pueda solicitar operaciones matemáticas a un servidor remoto. El servidor debe procesar la solicitud y devolver el resultado al cliente de manera transparente, como si la operación se ejecutara localmente.

Retos a resolver:

1. Comunicación entre cliente y servidor: ¿Cómo permitir que el cliente invoque métodos en el servidor sin preocuparse por la infraestructura de red?
2. Seguridad y control de acceso: ¿Cómo evitar que usuarios no autorizados accedan al servicio de cálculo?
3. Escalabilidad: ¿Cómo garantizar que varios clientes puedan usar el servicio simultáneamente sin afectar el rendimiento?
4. Manejo de errores: ¿Cómo manejar posibles fallos en la red o en el servidor sin que la aplicación cliente se detenga abruptamente?

Para solucionar este problema, utilizaremos Java RMI (Remote Method Invocation), una tecnología que permite la invocación de métodos en objetos remotos de manera transparente. Implementaremos una calculadora distribuida, en la que un servidor proporciona servicios de cálculo y un cliente puede acceder a ellos como si fueran métodos locales.

Propuesta de Solución

Para abordar el problema de la comunicación remota entre clientes y servidores de manera transparente y eficiente, proponemos desarrollar un sistema de cálculo distribuido utilizando Java RMI (Remote Method Invocation).

El sistema estará compuesto por dos componentes principales:

1. Servidor RMI: Encargado de alojar y gestionar un objeto remoto que ofrecerá servicios de cálculo.
2. Cliente RMI: Permite a los usuarios conectarse al servidor y solicitar cálculos remotos sin preocuparse por la infraestructura de red.

Elementos de la solución

- Implementación de la interfaz remota

Para garantizar que el cliente pueda comunicarse con el servidor, definiremos una interfaz Calculadora que especificará los métodos que podrán ejecutarse de manera remota.

- Creación del Servidor RMI

El servidor implementará la interfaz Calculadora y registrará el objeto en un Registro RMI para que los clientes puedan localizarlo y acceder a sus métodos.

- Desarrollo del Cliente RMI

El cliente buscará el objeto remoto en el registro RMI y lo utilizará como si fuera un objeto local, enviando solicitudes de cálculo y recibiendo los resultados.

- Mecanismo de serialización y comunicación

Se utilizará el mecanismo de serialización de Java para transmitir los datos entre el cliente y el servidor de manera segura y eficiente.

- Manejo de errores y excepciones

Se implementarán capturas de excepciones para manejar posibles fallos en la red, fallas del servidor o problemas con el registro RMI.

Materiales y Métodos

Recurso	Descripción
Java Development Kit (JDK) 8 o superior	Proporciona el entorno de desarrollo para compilar y ejecutar programas en Java, incluyendo soporte para RMI.
Eclipse	IDEs recomendados para facilitar la escritura, depuración y ejecución del código Java.
Consola del sistema operativo (CMD, PowerShell, Terminal, etc.)	Para ejecutar el servidor, el cliente y el registro RMI.
RMI Registry (rmiregistry)	Servicio necesario para registrar los objetos remotos y permitir que los clientes los encuentren.
Sistema Operativo Windows	Cualquier sistema que soporte Java SE es compatible con la ejecución del programa.

Desarrollo de la Solución

Calculadora.java

```java
package Objetos_Distribuidos;

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

public interface Calculadora extends Remote {
    int sumar(int a, int b) throws RemoteException;
    int restar(int a, int b) throws RemoteException;
}
```

En Java, una interfaz es como un contrato que define qué métodos debe tener una clase, sin decir cómo funcionan. En RMI, usamos una interfaz para que tanto el servidor como el cliente sepan qué métodos pueden usar de manera remota.

En esta seccione se define una interfaz llamada Calculadora, con un método sumar que toma dos números enteros y devuelve su suma. La interfaz

extiende Remote, lo que indica que sus métodos pueden ser llamados desde otra computadora.

Servidor.java

El servidor es el programa que ofrece el servicio de suma. Implementa la interfaz Calculadora, definiendo cómo funciona el método sumar.

```
1  package Objetos_Distribuidos;
2
3  import java.rmi.RemoteException;
4  import java.rmi.registry.LocateRegistry;
5  import java.rmi.registry.Registry;
6  import java.rmi.server.UnicastRemoteObject;
7
8  public class Servidor implements Calculadora {
9      public Servidor() {}
10
11     @Override
12     public int sumar(int a, int b) throws RemoteException {
13         return a + b;
14     }
15
16     @Override
17     public int restar(int a, int b) throws RemoteException {
18         return a - b;
19     }
20
21     public static void main(String[] args) {
22         try {
23             Servidor obj = new Servidor();
24             Calculadora stub = (Calculadora)
25 UnicastRemoteObject.exportObject(obj, 0);
26
27             Registry registry = LocateRegistry.createRegistry(1099);
28             registry.rebind("Calculadora", stub);
29
30             System.out.println("Servidor listo...");
31         } catch (Exception e) {
32             e.printStackTrace();
33         }
34     }
   }
```

Primero, se crea un objeto Servidor, que implementa Calculadora. Luego, se convierte en un objeto remoto con UnicastRemoteObject.exportObject(obj, 0). Después, se crea un registro RMI en el puerto 1099, que es donde los clientes buscarán el servicio. Finalmente, el objeto remoto se asocia con el nombre "Calculadora" en el registro.

Cuando el servidor está funcionando, los clientes pueden encontrarlo y llamar al método sumar.

Cliente.java

El cliente es el programa que solicita el servicio de suma al servidor.

```
  package Objetos_Distribuidos;
1
 2 import java.rmi.registry.LocateRegistry;
 3 import java.rmi.registry.Registry;
 4
 5 public class Cliente {
 6     public static void main(String[] args) {
 7         try {
 8             Registry registry =
 9 LocateRegistry.getRegistry("localhost", 1099);
10             Calculadora stub = (Calculadora)
11 registry.lookup("Calculadora");
12
13             int suma = stub.sumar(5, 3);
14             int resta = stub.restar(10, 4);
15
16             System.out.println("Suma: " + suma);
17             System.out.println("Resta: " + resta);
18         } catch (Exception e) {
19             e.printStackTrace();
20         }
21     }
  }
```

Primero, el cliente se conecta al registro RMI en localhost, donde se ejecuta el servidor. Luego, busca el objeto remoto "Calculadora" y lo obtiene. Finalmente, llama al método sumar(5, 3), que en realidad se ejecuta en el servidor, y muestra el resultado.

Cuando ejecutas el cliente, este encuentra el servidor y le pide realizar la suma, demostrando cómo los objetos distribuidos pueden comunicarse a través de la red.

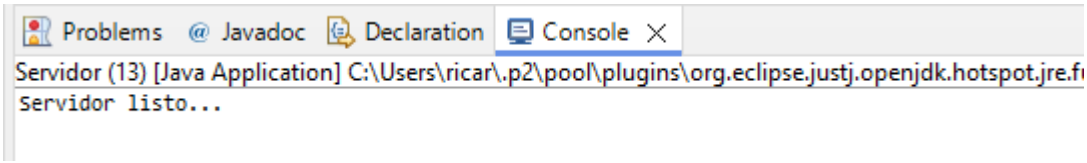
Resultados

Al ejecutar el sistema, se obtienen los siguientes comportamientos y resultados esperados:

Ejecución del Servidor

Cuando iniciamos Servidor.java, este crea un objeto remoto de la calculadora y lo registra en el sistema RMI.

En la consola del servidor, aparece el mensaje:

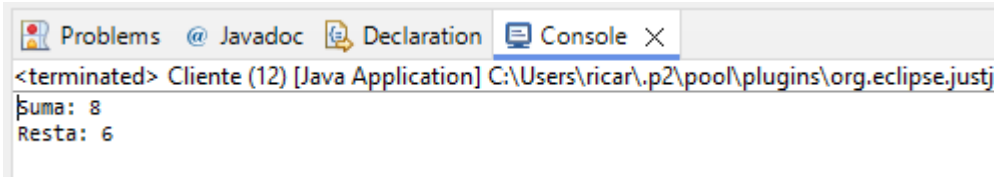


Ejecución del Cliente

Cuando ejecutamos Cliente.java, este se conecta al servidor y solicita la suma de dos números.

El servidor procesa la solicitud y devuelve el resultado al cliente.

En la consola del cliente, aparece el mensaje:



Conclusión

La implementación de objetos distribuidos mediante Java RMI demuestra cómo es posible ejecutar métodos de manera remota, permitiendo que distintas aplicaciones se comuniquen a través de una red como si fueran parte de un mismo sistema.

Durante el desarrollo, se creó una interfaz Calculadora, un servidor que ofrece el servicio de suma y un cliente que lo consume. La ejecución exitosa del sistema confirmó que el cliente puede invocar métodos en el servidor sin necesidad de conocer su implementación interna, lo que es una de las principales ventajas de la programación distribuida.

Se verificó que el sistema funciona correctamente tanto en la misma máquina como en equipos conectados en red, siempre que se configure adecuadamente la dirección IP del servidor. Además, se observó que en caso de no estar activo el servidor, el cliente no puede acceder al servicio, lo que resalta la importancia de un adecuado manejo de errores en aplicaciones distribuidas.

En conclusión, Java RMI proporciona un mecanismo eficiente para la comunicación entre procesos en diferentes computadoras, facilitando el desarrollo de sistemas distribuidos con interacción en tiempo real.