Clase 09/11/2020

Sistemas basados en objetos distribuidos

Paradigma de paz de mensajes

Es el modelo natural para el desarrollo de sistemas distribuidos, reproduce la comunicación entre las personas.

El programador debe serializar los datos antes de enviarlos y des-serializarlos antes de recibirlos.

Este paradigma es orientado a datos.

Objetos locales

Un objeto encapsula variables y funciones. Las variables guardan el estado del objeto y las funciones/metodos permiten modificar y acceder el estado del objeto.

Un objeto es local, si los métodos son evocados por un proceso local (misma PC)

Objetos Remotos

Sus métodos son invocados por procesos remotos (PC remota conectada mediante una red)

Los objetos no comparten el espacio de direcciones, solo valores pero no referencias.

Paradigma de objetos distribuidos

Combina objetos locales y remotos

Representa una atracción sobre el paso de mensajes, por ende el programador no debe preocuparse por controlar el paso de mensaje entre los nodos.

Es orientado a la “acción”, ya que se basa en la acción que realiza el método remoto invocado

Remote Method Invocation

En un sistema que utiliza RMI, existe un proceso llamado registry el caul hace las funciones de servidor de nombres.

En cada nodo existe un proceso servidor el caul registra en el servidor de nombres los objetos que exportara, cada uno esta identificado por una URL.

Para acceder a un objeto remoto, el proceso cliente consulta el servidor de nombres utilizando la URL, si el objeto es encontrado, entonces el servidor de nombres regresa al cliente una referencia que apunta al objeto remoto.

El paso de parámetros y regreso de resultados es manejado por la capa RMI

JAVA RMI

Es una API que implementa la invocación de métodos remotos, JDK incluye un servidor de nombres llamado “rmiregistry”

Como usar JAVA RMI

1. Para cada objeto remoto se debe crear una interface **I** que defina el prototipo de cada método a exportar. Es necesario declarar que los métodos remotos pueden producir la excepción java.rmi.RemoteException. La interface **I** debe heredar de java.rmi.Remote.

2. El código de los métodos remotos se debe escribir en una clase **C** que implemente la interface **I**. La clase **C** debe ser una subclase de java.rmi.server.UnicastRemoteObject. El constructor default de la clase **C** debe invocar el constructor de la superclase. Es necesario declarar que los métodos remotos pueden producir la excepción java.rmi.RemoteException.

3. El proceso servidor deberá registrar la clase **C** invocando el método bind() o el método rebind() de la clase java.rmi.Naming. A los métodos bind() y rebind() se les pasa como parámetros la URL correspondiente al objeto remoto y una instancia de la clase **C**. La URL tiene la siguiente forma: **rmi://***ip***:***puerto***/***nombre,* donde *ip* es la dirección IP de la computadora dónde ejecuta el programa **rmiregistry**, *puerto* es el número de puerto utilizado por rmiregistry (se puede omitir si rmiregistry utiliza el puerto default 1099) y *nombre* es el nombre con el que identificaremos el objeto.

4. El proceso cliente deber invocar el método lookup() de la clase java.rmi.Naming para obtener una referencia al objeto remoto. El método lookup() regresa una instancia de la clase Remote, la cual se debe convertir al tipo de la interface **I** mediante casting. Utilizando la referencia, el proceso cliente invocará los métodos remotos de la clase **C**.

Por razones de seguridad, la aplicación rmiregistry se debe ejecutar en la misma computadora dónde ejecuta el servidor.

Por default la aplicación rmiregistry utiliza el puerto 1099, si se utiliza otro puerto, se deberá pasar el número de puerto como argumento al ejecutar rmiregistry.

Se puede notar que el proceso servidor permanece en ejecución debido a que los métodos bind() y rebind() crean threads que no terminan.