

1

Comprender qué es un servicio en la red.

2


Aprender a planificar un servicio en la red.

3


Recordar el uso de *Socket* para la implementación de servicios en la red.

4

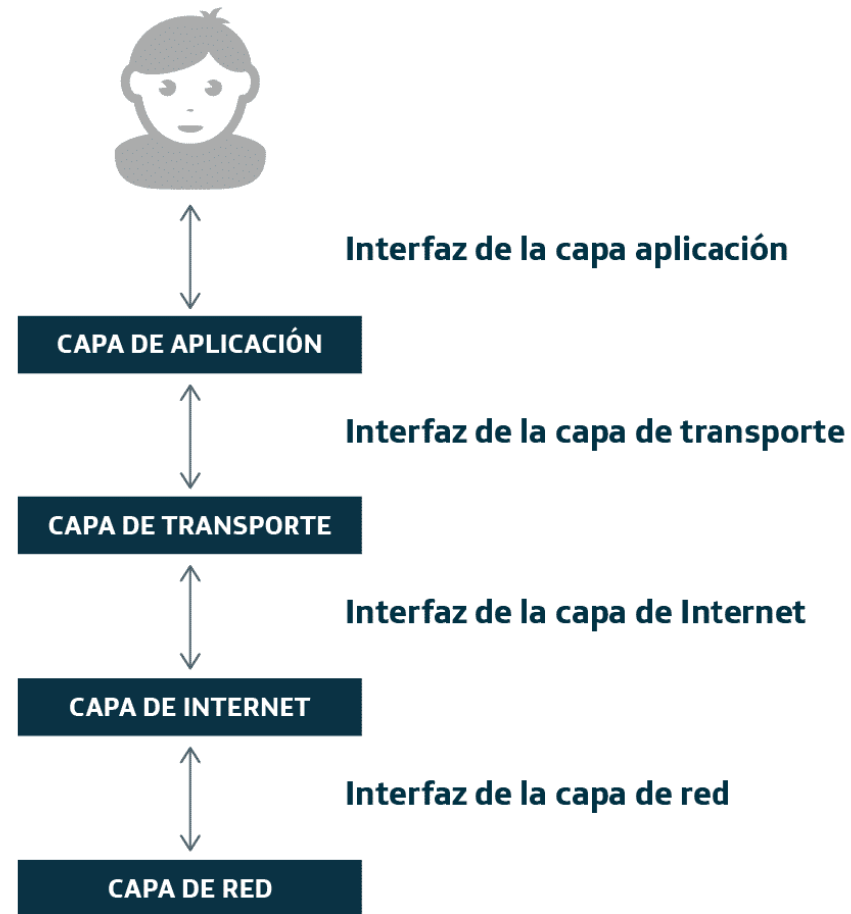
Aprender técnicas avanzadas para la implementación de servicios en la red (RMI).



Un servicio es un programa que se ejecuta en segundo plano y que no es utilizado directamente por el usuario, sino por otros programas que invocan a las funciones que dicho servicio deja disponibles a través de su interfaz.



Cada capa dentro de la pila de protocolos IP proporciona un servicio a la capa de nivel superior y una interfaz, a través de la cual puede comunicarse con ella.



Pila de protocolos IP

Servicios en la capa de aplicación

- Como futuro programador de servicios en la red, la capa que más debe preocuparte es la **capa de aplicación**. Para el resto de las capas contarás con diversos recursos que te permitirán olvidarte de ellas.
- La capa de aplicación debe contar con una interfaz para que el usuario pueda activar el servicio, desactivarlo, con gurararlo, etc.

Con nuestros conocimientos sobre servicios en la red podemos llegar a la conclusión de que la planificación para el desarrollo de un nuevo servicio requiere plantearse las siguientes preguntas:

1

¿Qué servicios debe prestar nuestro servidor?

2

¿Qué formato tiene la petición del cliente?

3

¿Qué formato tiene la respuesta del servidor?

4

¿Se resuelve con una sola petición/respuestas, o requiere de varios mensajes entre cliente y servidor?

5

¿Es necesario que nuestro servidor atienda varias peticiones simultáneas? Si la respuesta es afirmativa, será necesario el uso de hilos.

Desarrollo de una aplicación cliente/servidor para el cambio de divisas

1

¿Qué servicios debe prestar nuestro servidor? Se trata de un servidor que realiza el cálculo de cambio de divisa.

2

¿Qué formato tiene la petición del cliente? El mensaje de la petición del cliente estará formado por los siguientes elementos: una cantidad, divisa de dicha cantidad, divisa para el calculo del cambio. Ejemplo: *100;E;D* (donde E=Euros y D=Dólares).

3

¿Qué formato tiene la respuesta del servidor? La respuesta del servidor será una línea de texto similar a esta: *100 Euros = 115,64 Dólares*.

4

¿Se resuelve con una sola petición/respuestas, o requiere de varios mensajes entre cliente y servidor? Un mismo cliente podrá solicitar varios cambios de divisa, por lo tanto, harán falta varios mensajes entre cliente y servidor.

5

¿Es necesario que nuestro servidor atienda varias peticiones simultáneas? Sí, ya que varios clientes solicitar cambios de divisa al mismo tiempo.

Invocación de métodos remotos (RMI)

Ejemplo práctico RMI: el programa servidor

Desarrollaremos una aplicación distribuida con RMI. El programa servidor funcionará como un buscador de canciones, utilizando como clave de búsqueda el título de la canción, la banda, el álbum o el año de producción.

Crear la interfaz remota.

- Se trata de una interfaz Java que contiene la definición de los métodos que el cliente invocará de manera remota.
 - **Extender la interfaz *java.rmi.Remote***, lo que otorga a sus objetos la capacidad de poder invocar a sus métodos desde cualquier máquina virtual situada en cualquier equipo remoto.
 - Todos los métodos definidos en la interfaz deberán **propagar la excepción *java.rmi.RemoteException***. *RemoteException* es la superclase de la que derivan un conjunto de clases, relacionadas con los errores que pueden ocurrir durante la invocación a métodos remotos.
 - **La interfaz debe ser pública**, al igual que todos los métodos que deseemos exponer.

Se trata de crear una clase que implementará la interfaz remota y que será la encargada de llevar a cabo las tareas de la lógica de negocio del servicio.

Esta clase, además de implementar la interfaz remota, deberá extender de la clase *UnicastRemoteObject*, lo que permitirá a los clientes obtener el *stub* del objeto para establecer la comunicación y poder invocar a los métodos remotos.

Observa, además, que la clase *MusicaRMI* tiene un número de versión (*serialVersionUID*), ya que también es una clase **serializable**.

Cliente y Servidor

Clase LocateRegistry

Descripción General:

La clase LocateRegistry pertenece al paquete `java.rmi.registry`.

Se utiliza para obtener una referencia a un registro de objetos remotos en un host específico (incluyendo el host local) o para crear un registro de objetos remotos que acepte llamadas en un puerto específico¹.

Constructores:

LocateRegistry no tiene constructores públicos. Todos sus métodos son estáticos, lo que significa que se pueden llamar directamente sin crear una instancia de la clase

Métodos Estáticos:

1.createRegistry(int port):

- Crea y exporta una instancia de Registry en el host local que acepta solicitudes en el puerto especificado

.

2.createRegistry(int port, RMIClientSocketFactory csf, RMIServerSocketFactory ssf):

- Crea y exporta una instancia de Registry en el host local utilizando fábricas de sockets personalizadas para la comunicación

.

3.getRegistry():

- Devuelve una referencia al registro de objetos remotos para el host local en el puerto predeterminado (1099)

.

4.getRegistry(int port):

- Devuelve una referencia al registro de objetos remotos para el host local en el puerto especificado

.

5.getRegistry(String host):

- Devuelve una referencia al registro de objetos remotos en el host especificado en el puerto predeterminado (1099)

.

6.getRegistry(String host, int port):

- Devuelve una referencia al registro de objetos remotos en el host y puerto especificados

.

7.getRegistry(String host, int port, RMIClientSocketFactory csf):

- Devuelve una referencia localmente creada al registro de objetos remotos en el host y puerto especificados, utilizando una fábrica de sockets personalizad

1.Creación del Registro RMI:

```
registry registro = LocateRegistry.createRegistry(puerto);
```

Esta línea crea un registro RMI en el puerto especificado (por ejemplo, 5055).

El registro es un servicio que permite a los clientes encontrar objetos remotos por nombre.

2.Inscripción del Objeto Remoto:

```
registro.rebind("miMusica", musica);
```

- rebind(String name, Remote obj):**

- Este método inscribe (o vuelve a inscribir) un objeto remoto en el registro bajo el nombre especificado.
- Si ya existe un objeto registrado con ese nombre, lo reemplaza con el nuevo objeto.

- Parámetros:**

- "miMusica": Es el nombre bajo el cual el objeto remoto musica será accesible para los clientes.
- musica: Es la instancia del objeto remoto que implementa la interfaz MusicalInterfaceRMI.

Propósito: Hacer Disponible el Objeto Remoto:

- Permite que los clientes encuentren y se comuniquen con el objeto remoto musica utilizando el nombre "miMusica".
- Los clientes pueden usar `registry.lookup("miMusica")` para obtener una referencia al objeto remoto y llamar a sus métodos.

Servidor

```
MusicaRMI musica = new MusicaRMI();
```

MusicaRMI musica = new MusicaRMI(); crea una nueva instancia del objeto remoto MusicaRMI, que luego puede ser registrado y puesto a disposición de los clientes para invocaciones remotas.

1.MusicaRMI:

- Es el nombre de la clase que implementa la interfaz remota MusicaInterfaceRMI.
- Esta clase debe extender UnicastRemoteObject para ser exportada como un objeto remoto.

2.musica:

- Es el nombre de la variable que se está declarando.
- Esta variable será una referencia a una instancia de MusicaRMI.

3.new MusicaRMI():

- Crea una nueva instancia de la clase MusicaRMI.
- Llama al constructor de MusicaRMI, que puede incluir la lógica necesaria para inicializar el objeto remoto.

Propósito: Crear una Instancia del Objeto Remoto:

- Esta línea crea una instancia del objeto remoto MusicaRMI, que luego puede ser registrada en el registro RMI.
- Permite que el objeto musica esté disponible para ser invocado remotamente por los clientes.

Cliente

MusicalInterfaceRMI canciones = (MusicalInterfaceRMI) registry.lookup("miMusica");

Propósito:

Esta línea permite al cliente obtener una referencia al objeto remoto MusicaRMI registrado en el servidor bajo el nombre "miMusica".

Con esta referencia, el cliente puede invocar métodos remotos definidos en la interfaz MusicalInterfaceRMI, como buscarTitulo, buscarBanda, etc

MusicalInterfaceRMI canciones:

Declara una variable canciones de tipo MusicalInterfaceRMI y le asigna la referencia al stub del objeto remoto.

(MusicalInterfaceRMI):

Realiza un casting de la referencia devuelta por lookup al tipo MusicalInterfaceRMI.

Esto es necesario porque lookup devuelve una referencia de tipo Remote, y necesitas convertirla al tipo específico de tu interfaz remota para poder invocar sus métodos.

registry.lookup("miMusica"):

Este método busca en el registro RMI un objeto remoto registrado bajo el nombre "miMusica".

Devuelve una referencia al stub del objeto remoto, que es un proxy que permite la comunicación con el objeto remoto.

interfaceRMI

Clase Remote

Descripción General:

La clase Remote pertenece al paquete `java.rmi`.

Es una interfaz que identifica las interfaces cuyos métodos pueden ser invocados desde una máquina virtual no local.

Cualquier objeto que sea un objeto remoto debe implementar directa o indirectamente esta interfaz.

Propósito:

Permitir la invocación de métodos en objetos que residen en diferentes máquinas virtuales Java (JVMs), facilitando la comunicación en aplicaciones distribuidas

Métodos de la Interfaz Remote

Características Principales:

- Identificación de Métodos Remotos:
- Solo los métodos especificados en una interfaz que extiende Remote pueden ser invocados remotamente¹.
- Implementación de Clases:
- Las clases de implementación pueden implementar cualquier número de interfaces remotas y extender otras clases de implementación remota¹.

Conveniencia en la Implementación:

- Java RMI proporciona clases de conveniencia como UnicastRemoteObject que facilitan la creación de objetos remotos.

Ejemplo de Uso:

- En el código, MusicalInterfaceRMI extiende Remote, lo que permite que sus métodos (buscarTitulo, buscarBanda, etc.) **sean invocados desde una JVM diferente**

musicaRMI

Clase **UnicastRemoteObject**

- Descripción General:
- La clase ***UnicastRemoteObject*** pertenece al paquete `java.rmi.server`.
- Se utiliza para exportar un objeto remoto con JRMP (Java Remote Method Protocol) y obtener un stub que se comunica con el objeto remoto.

Características Principales:

- Exportación de Objetos Remotos: Permite que un objeto remoto esté disponible para recibir llamadas remotas.
- Generación de Stubs: Los stubs pueden ser generados dinámicamente en tiempo de ejecución utilizando objetos proxy

Métodos Principales de UnicastRemoteObject

Métodos de Exportación:

- `UnicastRemoteObject()`:

Constructor que exporta el objeto remoto en un puerto anónimo.

- `UnicastRemoteObject(int port)`:

Constructor que exporta el objeto remoto en el puerto especificado.

- `UnicastRemoteObject(int port, RMIClientSocketFactory csf, RMIServerSocketFactory ssf)`:

Constructor que exporta el objeto remoto utilizando fábricas de sockets personalizadas.

- `exportObject(Remote obj)`:

Método estático que exporta el objeto remoto utilizando un puerto anónimo.

- `exportObject(Remote obj, int port)`:

Método estático que exporta el objeto remoto en el puerto especificado

serialVersionUID

Descripción General:

- serialVersionUID es un identificador utilizado durante la serialización y deserialización de objetos en Java.
- Se asegura de que el remitente y el receptor de un objeto serializado tengan clases compatibles con respecto a la serialización.

Propósito:

- Verificar que la versión de la clase utilizada para serializar el objeto es compatible con la versión utilizada para deserializarlo.

Uso de serialVersionUID

Declaración:

- Se declara como un campo static, final y de tipo long3.
- `private static final long serialVersionUID = 1L;`

Generación del Valor:

- El valor puede ser cualquier número, pero debe cambiarse si se realizan cambios incompatibles en la clase3.
- Si no se declara explícitamente, el compilador generará uno automáticamente basado en varios atributos de la clase3.

Ejemplo:

- `private static final long serialVersionUID = -4817856459999901795L;`