SERVICIOS

23-01-2025

Anterior tema: la computación distribuida y mecanismos básicos de comunicación de entorno y en especial el modelo cliente-servidor, que es lo que hicimos un poco a ver con los sockets y eso es que se basa en la interacción entre clientes que solicitan servicios y servidores que los proporcionan.

Qué es un servicio, las características principales y cómo diseñar aplicaciones capaces de ofrecerlo.

Cosas que tenemos que tener en cuenta: **estructura** y **la función del sistema**.

La **estructura** se refiere a los componentes físicos o software que conforman el sistema y la función al propósito o la utilidad, digamos, a lo que hace el servicio. Por ejemplo, la lavadora tiene como estructura sus partes mecánica y electrónica, el tambor o el motor y la función es lavar la ropa.

WhatsApp lo mismo, la estructura está formada por aplicaciones entre clientes y servidores y la función es facilitar la comunicación entre usuarios mediante el mensaje.

El **concepto del servicio** es el conjunto de mecanismos que un sistema proporciona a los usuarios para que los usuarios accedan a su función.

Mientras que la **función define qué hace el sistema de servicio concreto**, ¿cómo se hace accesible esa funcionalidad?. Por ejemplo, la aplicación de mensajería puede tener como función **permitir la comunicación**, pero su servicio se define por las herramientas concretas que ofrece, como el envío de texto o de archivos.

Y en su mismo sistema, puede proporcionar múltiples servicios. El correo electrónico, por ejemplo, puedes enviar fotos, ficheros, un simple mensaje, incluso algunos ya puedes hacer llamadas y cosas así.

Interfaz de servicio.
Es el **punto de contacto entre el usuario y el sistema** y es lo que especifica los procedimientos y las restricciones necesarias. Más sencillo de lo que parece, si pensamos en WhatsApp, la interfaz incluye elementos como la aplicación del móvil, que son los menús, los botones, las pantallas y eso es lo que va guiando al usuario. Eso va dando una coherencia y un orden entre los envíos del mensaje y las respuestas del servidor entre la comunicación entre el cliente y el servidor.

Servicios de comunicaciones en aplicaciones distribuidas.
Durante las comunicaciones en las aplicaciones distribuidas **intervienen múltiples servicios** que trabajan en conjunto y garantizan el paso de mensajes entre la servidora receptiva. Ese servicio se organiza en la **pila de protocolos IP.**

La pila de protocolos digitales.
Organizan los servicios en diferentes niveles jerárquicos, que trabajan de forma cooperativa y garantizan el paso de los mensajes entre el emisor y los receptores.

La **Red** son las tecnologías, las conexiones físicas, la conexión de la cadena de todos. **Internet** son los mecanismos de direccionamiento y encaminamiento de una IP, para ir de una IP a otra IP tengo que ir por este camino de la red.
Y **aplicación**, que ya son los servicios que interactúan directamente con el usuario final, digamos ya, la aplicación que el usuario ve.

Servicios en cada nivel.
Cada nivel proporciona los servicios específicos a su nivel superior mediante **una interfaz de servicio**, por ejemplo, a nivel de red.

Pues servicios de comunicación física, la red es Cernet.
Nivel de internet, los paquetes de IP.
Nivel de transporte, gestión de puertas y transmisión fiable, TCP y UDP.
Y nivel de aplicación, protocolos de usuario como HTTP y FTP.

Protocolo de comunicación.
Cada nivel opera con su propio protocolo y define cómo interactúa con los otros elementos del sistema. Estos protocolos garantizan que los servicios se ejecutan de manera eficiente y ordenada.

¿Qué tenemos que pensar?
A la hora de programar una aplicación, siguiendo el modelo del cliente-servidor. Se pueden definir diferentes formas y aspectos antes de ponerse a programar. Como son las funciones del servidor, la tecnología de comunicación, protocolos de nivel de aplicación.

¿Cuál es la función básica de nuestro servicio? ¿El servicio que proporciona nuestro servidor? El rápido, el lento... Se puede resolver con una simple petición y respuesta o requiere varios momentos de comunicación. Va a atender a varios clientes simultáneamente, solo necesita atender a uno. Son cosas que antes de ponernos a programar, tenemos que saber.

Elección de la tecnología de comunicación.
Los dos mecanismos básicos que se utilizan son los **sockets de stream** y los **sockets de datagram.**

Los **sockets de Stream** están más orientados a la conexión. Esto garantiza que los paquetes lleguen de manera ordenada y fiables. Y son los más adecuados para aplicaciones más complejas que requieren un intercambio masivo de datos entre clientes y servidores.

Se usan cuando necesitan conexiones estables. Cuando las conexiones estables son la prioridad. Y los **datagrams** al final son algo más básicos. No están orientados a toda la conexión, son menos fiables. Es posible que se pierdan. A ver, bueno, depende a qué nivel estemos.

¿Que es un protocolo a nivel de aplicación?
El **conjunto de reglas** que regula la interacción entre los elementos de una aplicación distribuida. En el modelo cliente-servidor, este protocolo define cómo se comunica el cliente y el servidor. Elementos clave, el formato de los mensajes. El servidor necesita saber con anterioridad qué tipo de mensajes va a recibir. Para saber cómo tratarlo, para saber en qué tipo de variables guardarlo.

Un **string** que es un entero, una raíz de bytes. La secuencia de los mensajes, lo que decía, por ejemplo, con WhatsApp o con cualquier aplicación, según cómo tú interactúes con la aplicación vas a enviar un tipo de mensaje u otro. Entonces, tú puedes, por ejemplo, el juego del Pokémon...

Un turno tenía una serie de pasos, que es una secuencia. Entonces, tú ya sabes qué secuencia va a ser el envío de mensajes y la comunicación entre clientes y el servidor. El servidor sabe qué mensajes tiene que recibir primero y qué respuesta tiene que dar a ese mensaje. Y así sucesivamente.

Servidores multihilo
Para atender a múltiples clientes tienes que tener distintos hilos. Cada cliente va a ser un hilo. Hay que asegurar dos condiciones, el aislamiento entre clientes, que cada cliente debe percibir que está operando con el servidor de manera exclusiva. Tú no tienes por qué percibir cuántos hilos está gestionando el servidor, tú estás en tu hilo y es lo que te importa,
y una atención rápida y simultánea. El servidor tiene que ser capaz de atender a todos los clientes, todos los hilos que estén conectados a ese servidor.

Los **sockets de stream** son adecuados para la programación multihilo, lo que hemos dicho, son para comunicaciones más extensas, entonces para tener varios hilos nos conviene que estén orientados a conexiones. El servidor utiliza un socket que espera conexiones entrantes. Cuando llega una conexión, la operación accept crea un nuevo socket para comunicarse con ese cliente en particular.

Asignación de hilos.
Tras una conexión, el servidor arranca un nuevo hilo y se le asigna el socket del cliente, y luego ese nuevo hilo gestiona todas las interacciones con ese cliente. Y la eficiencia del hilo principal, el propio servidor, es un hilo.

Ventajas → atención simultánea, escalabilidad y aislamiento.

Cada cliente tiene un canal exclusivo, está la habilidad que se adecuó para atender a miles de usuarios y atención simultánea que permite gestionar múltiples días.

03-02-2025

Los **protocolos están en el nivel de aplicación**, las aplicaciones distribuidas, como las páginas web, los correos electrónicos, siguen el modelo de cliente-servidor.

El cliente y el servidor suelen ser independientes. Digamos que están desarrollados por distintas personas y en diferentes lenguajes. Entonces, para garantizar que puedan comunicarse de manera correcta, es esencial definir los protocolos del nivel de aplicación. Hay muchísimos protocolos.

Primero, Telnet. En realidad, ya no se usa mucho, pero es un protocolo de nivel de aplicación para comunicación bidireccional en texto plano. Permite enviar y recibir archivos de texto. Estos archivos no tienen por qué ser solo texto; pueden representar distintos tipos de información.

Simula una colección virtual. Utiliza TCP, lo que significa que usa sockets de stream. ¿Cuál era? Utiliza el **puerto 23 por defecto.**

Funciona como una sesión de línea de comandos. Tiene problemas de seguridad porque envía los datos sin cifrar, por lo que su uso ya no es recomendado.

SSH, Secure Shell, es una comunicación segura similar a Telnet pero más moderna, diseñada para resolver los problemas de seguridad de Telnet. **Cifra la información**. Es un protocolo conectado, mantiene la información de la sesión durante toda la comunicación y permite autenticación y seguimiento de comandos. Su **puerto por defecto** es el **22.**

FTP, File Transfer Protocol, para la transferencia de archivos en red. Utiliza TCP/stream. Tiene dos conexiones simultáneas: una de control en el puerto 21 que envía órdenes y recibe información, y otra para datos que transfiere los archivos, permitiendo la subida, descarga y gestión de directorios y permisos, manteniendo los datos de la sesión activa durante la transferencia.

HTTP es para el hipertexto. Es esencial para la web, controla la transferencia de documentos de hipertexto. Hay enlaces a otros documentos, como en una página web. Es un modelo cliente-servidor basado en **petición-respuesta**. Funciona con TCP/IP en el puerto 80.

Sin estado, **no almacena información de la sesión. ¿Por qué?** Porque tenemos las cookies. Las **cookies** son pequeños fragmentos de información que se almacenan en el cliente, permitiendo mantener datos entre peticiones. Cuando te vuelves a conectar, el cliente ya tiene los datos de la sesión almacenados y el servidor puede consultarlos. Enviadas por el servidor y guardadas por el navegador para usos futuros, mejoran la experiencia del usuario pero tienen implicaciones de seguridad y privacidad.

Sesiones HTTP y comunicación entre cliente y servidor. La sesión HTTP es una secuencia de intercambios de petición y respuesta. El cliente inicia la sesión estableciendo una conexión TCP en el puerto 80; el servidor espera las peticiones y responde con el estado de la petición, indicando si fue exitosa o errónea, y luego con el cuerpo del mensaje, que puede ser una página web, un archivo, etc. Los recursos o URLs están almacenados en el servidor y se acceden mediante hiperenlaces.

Métodos HTTP que utilizamos: GET solicita recursos, es la primera petición para cargar una página web. **HEAD** es igual a GET pero solo devuelve metadatos. **POST** envía datos al servidor, como al escribir en un foro.

PUT sube o actualiza recursos en el servidor. **DELETE** elimina el recurso del servidor. **OPTIONS** informa sobre los métodos que el servidor admite en una URL. **TRACE** devuelve la petición original como un eco para depuración. **CONNECT** convierte la conexión en un túnel TCP/IP, útil para enviar datos cifrados.

Compatibilidad y seguridad en HTTP.
No todos los servidores aceptan todos los métodos. HTTPS asegura la comunicación evitando ataques de intercepción.

Códigos de estado en HTTP. Todas las respuestas de HTTP tienen un código de estado de tres dígitos divididos en cinco categorías:

1XX: Información, la petición sigue en proceso.
2XX (OK): Éxito, como 200.
3XX: Redirección, el cliente debe hacer otra petición.
4XX: Error del cliente, como 404 No encontrado.
5XX: Error del servidor, como 500 Error interno.

Otros protocolos de nivel de aplicación específicos para correos electrónicos: **POP3** para acceder a correos almacenados, y **SMTP** para el envío de correos. Ambos funcionan con TCP.

DNS, Domain Name System, traduce nombres de dominio.

Técnicas avanzadas para programar aplicaciones distribuidas.
Programar directamente con sockets es poco práctico. Existen tecnologías que simplifican la comunicación, actuando como una capa intermedia entre el nivel de transporte y el de aplicación.

La **invocación de métodos remotos** permite a un objeto A en un cliente invocar métodos de un objeto B en un servidor. En Java, se conoce como **Remote Method Invocation (RMI).**

Para la invocación remota se necesita una **capa de software adicional** que traduce las llamadas a **métodos en mensajes** enviados por la red y viceversa.

Los **componentes principales** son los **stubs**, que actúan como representantes del objeto remoto en el cliente.

El **registro de objetos remotos** permite localizar y usar objetos remotos en el sistema.

El proceso de invocación de métodos remotos incluye iniciar el registro, crear el objeto en el servidor, inscribirlo en el registro, y luego el cliente busca y usa ese objeto.

El cliente interactúa con el registro a través del stub, que construye el mensaje de petición y lo envía al servidor. La respuesta regresa al cliente a través del stub, convirtiéndose en el valor de retorno.