WebSocket

SISTEMAS DISTRIBUIDOS

Carlos Rojas Sánchez

Licenciatura en Informática

Universidad del Mar

Contenido

Introducción a WebSocket

Definición

WebSocket es un protocolo que permite la comunicación en tiempo real y bidireccional entre el cliente y el servidor a través de una única conexión persistente.

Comparación de Protocolos: HTTP, WebSocket y CORBA

Característica	HTTP	WebSocket	CORBA
Tipo de Comunicación	Unidireccional	Bidireccional	Bidireccional
Conexión	Nueva conexión por solicitud	Conexión persistente	Conexión persistente

Table 1: Comparación entre HTTP, WebSocket y CORBA (Parte 1)

Comparación de Protocolos: HTTP, WebSocket y CORBA

Característica	HTTP	WebSocket	CORBA
Latencia	Mayor	Baja	Baja (pero más pesada que WebSocket)
Protocolo	Basado en texto	Basado en texto/binario	Binario
Escalabilidad	Escalable para contenido estático	Alta escalabilidad en tiempo real	Menos escalable por su complejidad

Table 2: Comparación entre HTTP, WebSocket y CORBA (Parte 2)

Comparación de Protocolos: HTTP, WebSocket y CORBA

Característica	HTTP	WebSocket	CORBA
Facilidad de Implementación	Fácil	Moderada	Compleja (requiere IDL y ORB)
Casos de Uso	Páginas web tradicionales	Chats, juegos en línea	Sistemas distribuidos complejos

Table 3: Comparación entre HTTP, WebSocket y CORBA (Parte 3)

Casos de uso

- Chats en tiempo real
- Juegos en línea
- Notificaciones instantáneas
- Aplicaciones financieras (datos en vivo)
- Control remoto de dispositivos IoT

Cómo Funciona WebSocket

Proceso de conexión (Handshake)

1. El cliente inicia una conexión HTTP estándar.
2. El cliente envía una cabecera Upgrade: websocket para solicitar el cambio de protocolo.
3. El servidor responde con el código 101 Switching Protocols y se establece la conexión WebSocket.

Comunicación Activa

- Mensajes en formato texto o binario.
- Ambos extremos pueden enviar datos de forma asíncrona.

WebSocket en Node.js

Instalación del Entorno WebSocket

1. Inicializa tu proyecto
2. Instala la librería ws

Métodos para el manejo de la comunicación en WebSocket

.on() Escuchar eventos. Se ejecuta cada vez que se recibe el evento especificado.

.emit() Enviar eventos. Se usa para enviar eventos junto con datos opcionales.

Combinación en una App de Chat

[Flujo de datos]

1. Cliente envía mensaje → socket.emit()
2. Servidor recibe → socket.on()
3. Servidor reenvía a otros clientes → io.emit()
4. Cliente recibe el mensaje de vuelta → socket.on()

Eventos en WebSocket y Socket.IO

WebSocket Protocolo nativo.

Socket.IO Una implementación más avanzada de WebSocket.

Eventos en WebSocket

Evento	Descripción
open	Se activa cuando la conexión se establece.
message	Se activa cuando se recibe un mensaje del otro extremo.
error	Se activa cuando ocurre un error en la conexión.
close	Se activa cuando la conexión se cierra.

Eventos en Socket.IO

Evento	Descripción
`connect`	Se activa cuando un cliente se conecta correctamente.
`disconnect`	Se activa cuando un cliente se desconecta del servidor.
`disconnecting`	Se activa justo antes de que un cliente se desconecte.
`error`	Se activa cuando ocurre un error en la conexión.

Eventos en Socket.IO

Evento	Descripción
`connect_error`	Se dispara si ocurre un error durante el intento de conexión.
`connect_timeout`	Se dispara si la conexión excede el tiempo de espera configurado.
`reconnect`	Se activa cuando el cliente se reconecta después de una desconexión.
`reconnect_attempt`	Se dispara cuando el cliente intenta reconectarse.

Eventos en Socket.IO

Evento	Descripción
`reconnect_error`	Se dispara si ocurre un error durante el intento de reconexión.
`reconnect_failed`	Se dispara si el cliente no logra reconectarse después de varios intentos.
`ping`	Se envía automáticamente para comprobar la latencia de la conexión.
`pong`	Responde automáticamente a un `ping`.
Eventos personalizados	Definidos por el desarrollador para manejar lógica específica.