Engineer

WebSocket

↔ protocolo para una conexión bidireccional que sigue abierta

Qué es

WebSocket es un protocolo para una conexión en la que ambas partes pueden escribir en cualquier momento. Un intercambio inicial HTTP la establece, y permanece abierta hasta que una de las partes la cierra o la conexión falla. El IETF lo publicó como RFC 6455 en diciembre de 2011. La parte que el navegador ofrece a los scripts, el objeto `WebSocket`, está definida en WebSockets Standard de WHATWG. En código y documentación también se escribe «websocket» o «web socket», y el plural WebSockets suele referirse a la tecnología en su conjunto.

En el intercambio inicial clásico de HTTP/1.1, la conexión empieza con un `GET` ordinario. El cliente añade `Upgrade: websocket` y `Connection: Upgrade`, un `Sec-WebSocket-Version` de 13 y un `Sec-WebSocket-Key`: un valor aleatorio de 16 bytes codificado en base64. Un servidor que acepta responde con el estado 101 Switching Protocols y una cabecera `Sec-WebSocket-Accept`. Su valor demuestra que el servidor ha leído precisamente ese intercambio, en lugar de reproducir una respuesta antigua: es la clave del cliente con un GUID fijo añadido, procesada con SHA-1 y codificada en base64. Desde ese momento termina HTTP en esa conexión y las dos partes intercambian tramas. En cambio, HTTP/2 y HTTP/3 transportan WebSocket por un único flujo mediante CONNECT extendido, como especifican RFC 8441 y RFC 9220.

Una trama es pequeña. Su cabecera empieza en dos bytes, crece en dos u ocho más cuando la carga supera los 125 bytes y añade otros cuatro en las tramas del cliente, que llevan una clave de enmascaramiento. Cada trama lleva un código de operación: texto, datos binarios o continuación de un mensaje repartido entre varias tramas; otros tres códigos corresponden a las tramas de control de cierre, ping y pong. Un mensaje de texto completo debe ser UTF-8 válido; un fragmento individual puede dividir una secuencia UTF-8. Cada trama que envía el cliente se enmascara con una nueva clave aleatoria de 32 bits, y el servidor nunca

enmascara. El enmascaramiento no es cifrado. Existe para impedir que un script hostil elija los bytes exactos que cruzan la red y los use para contaminar la caché de algún proxy situado en el camino.

Las direcciones usan los esquemas `ws://` y `wss://`, con los puertos 80 y 443 como valores predeterminados, y `wss` ejecuta todo el intercambio dentro de TLS. Son los puertos que ya utiliza HTTP, y es deliberado: para los cortafuegos y proxies por los que pasa, el intercambio inicial parece una petición web cualquiera.

Utilice el concepto en una decisión

Aplique «¿Qué es WebSocket?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. La parte que el usuario no ve: dónde se guardan los datos, qué reglas se aplican y cómo pueden comunicarse otros sistemas con ella.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Diseño y operación de 6 flujos ETL/ELT — 6 flujos ETL en tres motores de bases de datos

Diseño y operación de 6 flujos ETL/ELT Póngase en contacto →

Por qué importa

Con respuestas HTTP cortas, el servidor solo habla cuando se le pregunta. Muchos productos necesitan lo contrario: un mensaje de chat, un precio que acaba de cambiar, un cursor en un documento compartido o una tarea en segundo plano que ha terminado. Sin un canal por el que pueda escribir el servidor, el cliente tiene que realizar sondeos. Pregunta cada pocos segundos; la mayoría de las respuestas solo dice que nada ha cambiado, cada una lleva un juego completo de cabeceras y las novedades esperan hasta un intervalo antes de que alguien las vea. Una conexión WebSocket abierta elimina ambos costes. El servidor puede enviar cuando ocurre algo, sin esperar al siguiente sondeo. Cada mensaje solo necesita unos pocos bytes de estructura de trama en lugar de un juego completo de cabeceras HTTP.

El precio es mantener estado. Una petición HTTP puede llegar a cualquier servidor situado detrás de un balanceador de carga, mientras que una conexión WebSocket pertenece a un proceso durante toda su vida. Ese proceso reserva memoria para cada conexión abierta; un despliegue o reinicio las corta todas a la vez, y cada proxy y balanceador del camino debe admitir el intercambio inicial y tolerar una conexión que puede permanecer en silencio durante minutos. Usar más de un servidor plantea una pregunta que HTTP ordinario no hace: cuando

sucede algo en el servidor A, ¿cómo llega a un cliente cuya conexión está en el servidor B? La respuesta habitual es un canal compartido de publicación y suscripción que escuchan todos los servidores. Eso añade otro sistema que mantener en funcionamiento.

Por eso es una elección, no la opción predeterminada. Cuando solo el servidor tiene novedades que enviar, los eventos enviados por el servidor, definidos en HTML Standard, transmiten eventos mediante una respuesta HTTP ordinaria de tipo `text/event-stream`, y el navegador se reconecta por sí solo cuando la respuesta se interrumpe. WebSocket justifica sus componentes adicionales cuando el cliente tiene tanto que decir como el servidor, como en los chats, los juegos y la edición colaborativa.

Cómo se aplica en la práctica

El intercambio siguiente abre una conexión de chat. La clave es el valor de ejemplo del propio RFC 6455, y el valor de aceptación es el que muestra el estándar y que todo servidor correcto debe devolver para esa clave.

```
GET /chat HTTP/1.1
Host: example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Version: 13
Origin: https://example.com

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
```

Después de la línea en blanco que termina la respuesta del servidor, nada de lo que circula por esa conexión sigue siendo HTTP. El navegador realiza todo el intercambio inicial con una llamada al constructor y deja el resto a los escuchadores de eventos:

```
const socket = new WebSocket("wss://example.com/chat");
socket.addEventListener("open", () => socket.send("hello"));
socket.addEventListener("message", (event) => console.log(event.data));
```

```
socket.addEventListener("close", (event) => {
  console.log(event.code, event.reason, event.wasClean);
});
```

El evento de cierre indica cómo terminó la conexión. El código 1000 es un cierre normal, y el 1001 significa que una de las partes se marcha, como un servidor que se apaga o un navegador que abandona la página. El código 1006 significa que la conexión se cortó sin ninguna trama de cierre. Ninguno de los extremos puede enviarlo: solo existe para que el programa que utiliza el protocolo distinga una terminación anormal de una limpia.

La diferencia entre el modelo de petición y respuesta y un socket se ve con claridad en un proyecto del portafolio de este sitio. Al migrar una API a Huma v2, todas las rutas pasaron al modelo de petición y respuesta de Huma salvo exactamente dos: los puntos de conexión WebSocket siguieron directamente sobre Fiber, porque allí el código necesita realmente el socket y el modelo de Huma no encaja. Un framework construido sobre la estructura de HTTP describe y valida bien peticiones y respuestas; una conexión bidireccional de larga duración es sencillamente otra cosa.

Dónde surgen los problemas

Una página de otro sitio abre una conexión con las cookies de sus visitantes. El navegador envía la petición de intercambio inicial al host del servidor y puede incluir las cookies de ese host, de acuerdo con las políticas de cookies del navegador. Si esas políticas permiten peticiones autenticadas entre sitios, un servidor que autentica mediante cookies e ignora la cabecera `Origin` puede aceptar una conexión abierta por una página de otro sitio en nombre del visitante. RFC 6455 indica que un servidor destinado solo a ciertos sitios debería comparar `Origin` con los esperados y rechazar los demás con 403 Forbidden. Hágalo durante el intercambio inicial, antes de aceptar la conexión.

El token de acceso termina en los registros. El constructor `WebSocket` del navegador solo acepta una URL y una lista opcional de subprotocolos, de modo que un script no puede añadir una cabecera `Authorization`. El atajo es poner un token en la URL, donde los proxies inversos y los registros de acceso lo anotan. El RFC deja abierta la autenticación y menciona como opciones las cookies, la autenticación HTTP y la autenticación TLS. Funciona una cookie con la

comprobación de `Origin` anterior, y también un pase de un solo uso y corta duración que la página obtiene mediante una llamada HTTP autenticada y consume en el intercambio inicial.

Un reinicio convierte a todos los clientes en una avalancha. Un despliegue abrupto o un reinicio puede cerrar todas las conexiones a la vez. Los clientes que no reciben una trama de cierre notifican el código 1006; si todos se reconectan inmediatamente y siguen intentándolo, el servidor vuelve a arrancar ante lo que el RFC compara con un ataque de denegación de servicio. RFC 6455 pide que los clientes esperen un tiempo aleatorio antes del primer intento, sugiere entre 0 y 5 segundos y solicita aumentar la espera tras cada fallo, por ejemplo mediante retroceso exponencial binario truncado. Un servidor que cierra sus conexiones con 1001 antes de detenerse al menos comunica a los clientes que la interrupción estaba prevista.

```
function reconnectDelay(attempt) {
  if (attempt === 0) return Math.random() * 5000;
  const ceiling = Math.min(60_000, 1000 * 2 ** attempt);
  return Math.random() * ceiling;
}
```

Las conexiones mueren y nadie lo advierte. Una conexión sin tráfico no da a ninguna parte una señal de que el camino siga existiendo. Un extremo que desaparece sin trama de cierre parece igual que uno silencioso hasta que falla la siguiente escritura. Para eso existen ping y pong: un extremo debe responder a un ping con un pong, salvo que ya haya recibido una trama de cierre. Si varios pings siguen sin respuesta, puede contestar solo al último. El RFC señala que un ping sirve para mantener viva la conexión o comprobar que la otra parte aún responde. La API del navegador no permite enviar un ping, por lo que el temporizador corresponde al servidor: envíe pings a cada conexión a intervalos fijos y cierre las que no respondan con su pong.

Un mensaje consume toda la memoria del servidor. El campo de longitud de la cabecera de una trama admite cargas mucho mayores que la memoria de cualquier servidor. El RFC advierte que un extremo hostil puede enviar una trama enorme o una sucesión interminable de fragmentos pequeños. Las implementaciones deben protegerse: fije un tamaño máximo de mensaje en la biblioteca del servidor y cierre con el código 1009, message too big, cuando un extremo lo supere. La misma acumulación aparece en sentido contrario. En el navegador, `bufferedAmount` indica los bytes que `send()` ha puesto en cola y que

aún esperan para salir a la red; el ejemplo de WHATWG envía la siguiente actualización solo cuando ese número es cero. En el servidor, un cliente que lee despacio llena del mismo modo el búfer de salida. Cada conexión necesita un límite para ese búfer y una regla sobre qué hacer al alcanzarlo.

Lecturas adicionales

- RFC 6455: The WebSocket Protocol — intercambio inicial, tramas, enmascaramiento y códigos de cierre, con las razones de seguridad que hay detrás de cada uno.
- WebSockets Standard — API del navegador: constructor, `send()`, `close()`, `bufferedAmount` y evento de cierre.
- RFC 9110: HTTP Semantics — cabecera `Upgrade` y estado 101 Switching Protocols en que se apoya el intercambio inicial.
- Server-sent events — alternativa unidireccional de HTML Standard, con reconexión incorporada.
- RFC 8441: Bootstrapping WebSockets with HTTP/2 — CONNECT extendido transporta WebSocket por un flujo HTTP/2 multiplexado.
- RFC 9220: Bootstrapping WebSockets with HTTP/3 — mecanismo correspondiente de CONNECT extendido para HTTP/3.

Glosario

Elija según sus restricciones reales

Describa el flujo de trabajo, las interfaces existentes y los usuarios que necesitan el cambio. Podemos definir una entrega comprobable y un traspaso que permita a su equipo mantener lo entregado.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es WebSocket?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/websocket/