Engineer

WebSocket

↔ протокол постійного двостороннього з'єднання

Що це таке

WebSocket — це протокол з’єднання, у яке обидві сторони можуть писати будь-коли. Його встановлює рукостискання HTTP, і воно лишається відкритим, доки одна зі сторін його не закриє або з’єднання не обірветься. Його описує RFC 6455, який IETF опублікувала у грудні 2011 року, а ту частину, яку браузер відкриває скриптам, — об’єкт `WebSocket` — визначає WebSockets Standard від WHATWG. У коді й документації назву часто пишуть «websocket» або «web socket», а множина WebSockets зазвичай означає технологію загалом.

У класичному рукостисканні HTTP/1.1 з’єднання починається зі звичайного запиту `GET`. Клієнт додає `Upgrade: websocket` і `Connection: Upgrade`, `Sec-WebSocket-Version` зі значенням 13 та `Sec-WebSocket-Key` — випадкові 16 байтів, закодовані в base64. Сервер, який погоджується, відповідає кодом стану 101 Switching Protocols і полем заголовка `Sec-WebSocket-Accept`. Це значення доводить, що сервер прочитав саме це рукостискання, а не повторює стару відповідь: це ключ клієнта з дописаним до нього фіксованим GUID, хешований SHA-1 і закодований у base64. Від цієї миті HTTP на з’єднанні закінчується, і сторони обмінюються кадрами. Натомість HTTP/2 та HTTP/3 передають WebSocket одним потоком за допомогою розширеного CONNECT, як визначають RFC 8441 і RFC 9220.

Кадр невеликий. Його заголовок займає щонайменше два байти, виростає ще на два або вісім, коли корисне навантаження довше за 125 байтів, і ще на чотири в кадрах від клієнта, бо ті несуть ключ маскування. Кожен кадр має код операції: текст, двійкові дані або продовження, коли повідомлення розбито на кілька кадрів, а також три керувальні кадри — для закриття, ping і pong. Повне текстове повідомлення мусить бути коректним UTF-8; окремий фрагмент може розділяти послідовність UTF-8. Кожен кадр, який надсилає клієнт, маскується новим випадковим 32-бітним ключем, а сервер не маскує

ніколи. Маскування — не шифрування. Воно існує для того, щоб ворожий скрипт не міг вибрати точні байти, які підуть мережею, і з їхньою допомогою отруїти кешувальний проксі десь на шляху.

Адреси починаються з `ws://` або `wss://`, за замовчуванням використовують порти 80 і 443, а `wss` веде весь обмін усередині TLS. Це ті самі порти, що й у HTTP, і так задумано: для брандмауерів і проксі, крізь які проходить рукостискання, воно виглядає як будь-який інший вебзапит.

Застосуйте поняття до рішення

Застосуйте «Що таке WebSocket?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Та частина, якої користувач не бачить: де зберігаються дані, які діють правила і як інші системи можуть із нею взаємодіяти.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Спроєктувала й підтримувала 6 пайплайнів ETL/ELT — 6 пайплайнів ETL на трьох рушіях баз даних

Спроєктувала й підтримувала 6 пайплайнів ETL/ELT Зв'язатися з нами →

Чому це важливо

За коротких відповідей HTTP сервер говорить, лише коли його питають. Багатьом продуктам потрібне протилежне: повідомлення в чаті, ціна, що щойно змінилася, курсор у спільному документі, фонове завдання, яке завершилося. Без каналу, у який може писати сервер, клієнтові лишається опитування. Він питає кожні кілька секунд, більшість відповідей лише повідомляє, що нічого не змінилося, кожна несе повний набір полів заголовка, а новина чекає до цілого інтервалу, перш ніж її хтось побачить. Відкрите з’єднання WebSocket прибирає обидві витрати. Сервер може надсилати дані, коли щось стається, не чекаючи наступного опитування. Кожне повідомлення потребує лише кількох байтів оформлення кадру замість повного набору заголовків HTTP.

Ціна — це стан. Запит HTTP може потрапити на будь-який сервер за балансувальником навантаження, а з’єднання WebSocket належить одному процесу, доки живе. Цей процес тримає пам’ять під кожне відкрите з’єднання, розгортання чи перезапуск обриває їх усі разом, а кожен проксі й балансувальник навантаження на шляху має підтримувати рукостискання й терпіти з’єднання, яке може мовчати хвилинами. Коли серверів більше одного, постає питання, якого звичайний HTTP ніколи не ставить: якщо щось

сталося на сервері A, як про це дізнається клієнт, чиє з'єднання живе на сервері B? Звична відповідь — спільний канал publish/subscribe, який слухають усі сервери, і це ще одна система, яку треба підтримувати.

Тож WebSocket — це свідомий вибір, а не відправна точка. Коли новини є лише в сервера, server-sent events, описані в HTML Standard, надсилають події звичайною відповіддю HTTP з типом `text/event-stream`, а браузер сам перепідключається, коли відповідь обривається. WebSocket виправдовує свої додаткові рухомі частини тоді, коли клієнтові є що сказати не менше, ніж серверові: у чатах, іграх і спільному редагуванні документів.

Як це виглядає на практиці

Наведений нижче обмін відкриває з'єднання для чату. Ключ — зразкове значення, узяте з самого RFC 6455, а значення `Sec-WebSocket-Accept` — саме те, яке наводить стандарт і яке на цей ключ мусить повернути будь-який коректний сервер.

```
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=
```

Після порожнього рядка, що завершує відповідь сервера, на з'єднанні вже немає HTTP. Браузер виконує все рукостискання одним викликом конструктора й передає решту обробникам подій:

```
const socket = new WebSocket("wss://example.com/chat");
socket.addEventListener("open", () => socket.send("привіт"));
socket.addEventListener("message", (event) => console.log(event.data));
socket.addEventListener("close", (event) => {
  console.log(event.code, event.reason, event.wasClean);
});
```

Подія закриття повідомляє, як завершилося з'єднання. Код 1000 — нормальне закриття, а 1001 означає, що одна зі сторін іде: наприклад, сервер вимикається або браузер залишає сторінку. Код 1006 означає, що з'єднання обірвалося зовсім без кадру закриття. Жодна сторона не має права його надсилати: він існує лише для того, щоб програма зверху могла відрізнити аварійне завершення від чистого.

Межу між моделлю «запит — відповідь» і сокетом добре видно в одному проєкті з портфоліо сайту. Коли API переводили на Huma v2, усі маршрути перейшли на модель запиту й відповіді Huma, крім рівно двох: ендпоінти WebSocket лишилися на чистому Fiber, бо там коду справді потрібен сам сокет, а модель Huma не пасує. Фреймворк, побудований навколо форми самого HTTP, добре описує й перевіряє запити та відповіді, а довгоживуче двостороннє з'єднання — просто інша річ.

Де виникають проблеми

Сторінка з іншого сайту відкриває з'єднання з cookie ваших відвідувачів. Браузер надсилає запит рукостискання до хоста сервера й може додавати його cookie відповідно до своєї політики щодо cookie. Якщо ця політика дозволяє автентифіковані міжсайтові запити, сервер з автентифікацією за cookie, який ігнорує `Origin`, може прийняти з'єднання, відкрите сторінкою іншого сайту, від імені відвідувача. RFC 6455 радить серверові, призначеному лише для певних сайтів, звіряти `Origin` з очікуваними й відмовляти решті кодом 403 Forbidden. Робіть це під час рукостискання, до прийняття з'єднання.

Токен доступу потрапляє в журнали. Конструктор `WebSocket` у браузері приймає URL і необов'язковий список підпротоколів — і більше нічого, тож скрипт не може додати поле `Authorization`. Швидкий обхід — токен в URL, де його записують зворотні проксі й журнали доступу. RFC залишає автентифікацію відкритою й називає як варіанти cookie, автентифікацію HTTP і автентифікацію TLS. Працює cookie разом із перевіркою `Origin`, описаною вище, і так само працює короткоживучий одноразовий квиток, який сторінка отримує через автентифікований виклик HTTP і витрачає на рукостискання.

Перезапуск перетворює всіх клієнтів на лавину. Раптове розгортання чи перезапуск можуть одночасно закрити всі з'єднання. Клієнти, які не отримали кадру закриття, повідомляють код 1006; якщо всі вони негайно підключаються знову й не припиняють спроб, сервер, піднявшись, отримує

те, що RFC порівнює з атакою на відмову в обслуговуванні. RFC 6455 просить клієнтів почекати випадковий час перед першою спробою, пропонуючи від 0 до 5 секунд, а після кожної невдачі чекати довше, наприклад з усіченою двійковою експоненційною затримкою. Сервер, який перед зупинкою закриває свої з'єднання з кодом 1001, принаймні повідомляє клієнтам, що обрив був запланований.

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

З'єднання вмирають, і ніхто цього не помічає. З'єднання без трафіку не дає жодній стороні знаку, що шлях досі існує, а співрозмовник, який зник без кадру закриття, виглядає точнісінько як той, що просто мовчить, аж до наступного невдалого запису. Для цього й існують ping і pong: сторона мусить відповідати на ping кадром pong, якщо ще не отримала кадр закриття. Якщо кілька ping лишилися без відповіді, вона може відповісти лише на останній. RFC зазначає, що ping може слугувати і keepalive, і перевіркою того, що інша сторона ще відповідає. API браузера не дає способу надіслати ping, тож таймер має жити на сервері: пінгуйте кожне з'єднання з фіксованим інтервалом і закривайте ті, що не відповіли кадром pong.

Одне повідомлення з'їдає пам'ять сервера. Поле довжини в заголовку кадру вміщує розміри, на які не вистачить пам'яті жодного сервера, і RFC попереджає, що ворожа сторона може надіслати один величезний кадр або нескінченну низку дрібних фрагментів. Реалізації мусять себе захищати, тож задайте в серверній бібліотеці максимальний розмір повідомлення й закривайте з'єднання кодом 1009, message too big, коли співрозмовник його перевищує. Те саме накопичення буває й у зворотному напрямку. У браузері `bufferedAmount` показує, скільки переданих у `send()` байтів досі чекають на відправлення в мережу, і приклад у стандарті WHATWG надсилає наступне оновлення лише тоді, коли це число дорівнює нулю. На сервері клієнт, що читає повільно, так само наповнює вихідний буфер, тож кожному з'єднанню потрібна межа для нього й правило, що робити, коли межу досягнуто.

Що почитати далі

- RFC 6455: The WebSocket Protocol — рукостискання, кадри, маскування й коди закриття, а також міркування безпеки, що стоять за кожним із них.
- WebSockets Standard — API браузера: конструктор, `send()`, `close()`, `bufferedAmount` і подія закриття.
- RFC 9110: HTTP Semantics — поле заголовка `Upgrade` і код стану 101 Switching Protocols, на які спирається рукостискання.
- Server-sent events — односпрямована альтернатива з HTML Standard із вбудованим перепідключенням.
- RFC 8441: Bootstrapping WebSockets with HTTP/2 — розширений CONNECT передає WebSocket одним мультиплексованим потоком HTTP/2.
- RFC 9220: Bootstrapping WebSockets with HTTP/3 — відповідний механізм розширеного CONNECT для HTTP/3.

Словник

Обирайте за фактичними умовами

Опишіть робочий процес, наявні інтерфейси та користувачів, яким потрібна зміна. Ми можемо визначити перевірюваний результат і спосіб передачі, який дасть вашій команді змогу підтримувати його.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке WebSocket?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

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

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/websocket/