Engineer

Server-sent events

“A” односпрямований потік подій від сервера через HTTP

Що це таке

Server-sent events — це спосіб, у який вебсервер надсилає сторінці повідомлення через одну звичайну відповідь HTTP: сервер тримає її відкритою й дописує в неї щоразу, коли є новини. Їх визначає HTML Standard, який підтримує WHATWG, і цей стандарт описує як текстовий формат потоку, так і `EventSource`, інтерфейс браузера, що цей потік читає. Назву часто скорочують до SSE, а сам формат нерідко називають за його медіатипом, `text/event-stream`.

Сторінка відкриває потік одним рядком, `new EventSource(url)`. Браузер надсилає запит `GET`, який може містити `Accept: text/event-stream`, і чекає відповіді зі статусом 200 і саме цим типом вмісту. Сервер надсилає заголовки, а тіло відповіді не завершує. Щоразу, коли щось стається, він дописує кілька рядків тексту, і браузер перетворює кожен блок рядків на подію, яку сторінка може слухати.

Формат навмисно простий. Потік завжди в кодуванні UTF-8, і кожен рядок — це одне поле. Рядок `data` несе вміст події, а кілька рядків `data` поспіль з’єднуються через розриви рядків. Рядок `event` задає тип події, а без нього тип — `message`. Рядок `id` задає ID останньої події, а рядок `retry` — скільки мілісекунд браузер чекає, перш ніж під’єднатися знову. Рядок, що починається з двокрапки, — це коментар, і браузер його ігнорує. Порожній рядок завершує блок і передає подію сторінці, тож блок, обірваний до свого порожнього рядка, відкидається.

Повторне під’єднання — частина стандарту. Коли з’єднання обривається, браузер чекає час повторного під’єднання, початкове значення якого вибирає сам, за стандартом, імовірно, кілька секунд, а тоді надсилає запит знову. Якщо сервер давав своїм подіям ID, новий запит несе останній із них у

заголовку `Last-Event-ID` , і сервер може продовжити з того місця. Деякі відповіді натомість завершують потік назавжди: статус, відмінний від 200, або інший тип вмісту змушує браузер здатися й більше не пробувати, а стандарт називає 204 No Content способом, у який сервер повідомляє, що потік закінчено.

Напрямок лише один. Після початкового запиту на цьому з'єднанні від сторінки до сервера не йде нічого; коли сторінці є що сказати, вона надсилає окремий запит HTTP.

Застосуйте поняття до рішення

Застосуйте «Що таке server-sent events?» 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

Багато функцій, які виглядають «живими», потребують лише одного напрямку: поступ довгого завдання, сповіщення, цифра на дашборді, відповідь, що надходить частинами. Без потоку сторінка мусить опитувати сервер. Вона питає кожні кілька секунд, більшість відповідей каже, що нічого не змінилося, а новина чекає до одного інтервалу, перш ніж хтось її побачить. З потоком сервер пише тієї ж миті, як щось стається.

Звична альтернатива для передавання даних від сервера — WebSocket, який перемикає з'єднання з HTTP на власний протокол. Server-sent events залишаються в межах HTTP від початку до кінця, тож проходять через ті самі проксі й балансувальники навантаження, що й будь-який інший запит, використовують ті самі файли cookie й ті самі правила для запитів з іншого джерела, і можуть ділити одне з'єднання HTTP/2 з рештою сторінки. Повторне під'єднання й продовження з останньої події дає сам браузер, а не код, який кожна команда пише для себе. WebSocket лишається правильним вибором, коли клієнтові є що сказати не менше, ніж серверу, як у чаті чи спільному редакторі.

Формат не прив’язаний і до браузера. Його може читати будь-який клієнт HTTP, і його використовують специфікації поза вебом. У Model Context Protocol, через який ШІ-агенти викликають інструменти, транспорт Streamable HTTP дозволяє серверу відповісти на запит або одним об’єктом JSON, або відповіддю `text/event-stream`. У такому потоці сервер може надсилати власні сповіщення й запити ще до фінальної відповіді, а також додавати ID подій, щоб клієнт, який утратив з’єднання, продовжив із `Last-Event-ID`. Streamable HTTP замінив HTTP+SSE, транспорт попередньої версії протоколу, 2024-11-05.

Обмеження варто знати, перш ніж обирати цей спосіб. Потік несе лише текст, тож двійкові дані спершу треба закодувати як текст. Конструктор `EventSource` у браузері приймає URL і один параметр, `withCredentials`, і більше нічого, тож сторінка не може додати до запиту заголовок `Authorization`.

Як це виглядає на практиці

Обмін нижче стежить за довгим завданням експорту. Браузер утратив з’єднання після події 41 і під’єднується знову, тож його запит несе цей ID, а сервер продовжує з події 42:

```
GET /jobs/4471/events HTTP/1.1
Host: example.com
Accept: text/event-stream
Last-Event-ID: 41

HTTP/1.1 200 OK
Content-Type: text/event-stream
Cache-Control: no-store

retry: 5000

: still working

id: 42
event: progress
data: {"done": 120, "total": 400}

id: 43
event: progress
data: {"done": 400, "total": 400}
```

```
id: 44
event: finished
data: {"report": "/jobs/4471/report.pdf"}
```

Рядок `retry` просить браузер чекати п’ять секунд перед кожним наступним повторним під’єднанням, а рядок-коментар не несе жодної події; він потрібен, щоб з’єднання не простоювало, поки завдання мовчить. Сторінка слухає обидва типи подій за назвою:

```
const source = new EventSource("/jobs/4471/events");
source.addEventListener("progress", (event) => {
  const { done, total } = JSON.parse(event.data);
  showProgress(done / total);
});
source.addEventListener("finished", (event) => {
  showReport(JSON.parse(event.data).report);
  source.close();
});
source.addEventListener("error", () => {
  if (source.readyState === EventSource.CLOSED) showRetryButton();
});
```

Виклик `close()` тут важливий. Коли сервер завершує відповідь звичайним чином, браузер сприймає це як обірване з’єднання й під’єднується знову, тож потік для завершеного завдання інакше відкрився б знову за п’ять секунд. Обробник `error` розрізняє два види збою: поки `readyState` має значення `CONNECTING`, браузер ще пробує, а щойно воно стає `CLOSED`, браузер здався.

Робота, що триває після того, як агент отримав відповідь, згадується й на сторінці цього сайту ваші програми як інструменти для ШІ-агентів. Вона описує, як операції компанії стають інструментами, а MCP-сервер надає ті самі операції агентам, що підтримують цей протокол, і як тривалі операції передають у чергу, щоб агент отримував відповідь, поки сама робота виконується у фоні. У специфікації MCP сервер може використати потік подій, подібний до наведеного вище, щоб надсилати власні запити або сповіщення до остаточної відповіді на запит клієнта.

Де виникають проблеми

Події надходять пачками або всі наприкінці. Щось між сервером і браузером притримує відповідь. nginx, наприклад, за замовчуванням буферизує відповіді від сервера, що стоїть за ним, і передає відповідь далі в міру надходження, лише коли буферизацію вимкнено. HTML Standard також попереджає, що поділ HTTP на частини (chunking), який виконує шар, що нічого не знає про часові вимоги потоку, може зробити потік ненадійним. Вимкніть буферизацію для адреси потоку, у nginx через `proxy_buffering off` або заголовок відповіді `X-Accel-Buffering: no`, скидайте вивід після кожної події в коді сервера й тестуйте весь шлях разом із проксі, а не лише сам сервер.

Тихі потоки, обірвані через тайм-аут. Проксі, який якийсь час не бачить жодного байта, вважає з'єднання мертвим. У nginx `proxy_read_timeout` за замовчуванням становить 60 секунд між двома читаннями, після чого з'єднання закривається, а HTML Standard попереджає, що деякі старіші проксі обривають з'єднання після короткого тайм-ауту. Стандарт радить надсилати рядок-коментар приблизно кожні 15 секунд: це коштує кількох байтів, і браузер такий рядок ігнорує.

Події, втрачені під час повторного під'єднання. Браузер надсилає `Last-Event-ID`, лише якщо сервер давав своїм подіям ID, і заголовок допомагає, лише якщо сервер іще може знайти те, що йшло після цього ID. Дайте кожній події ID, що працює як курсор у потоці, тримайте вікно недавніх подій для повторного надсилання й надсилайте повний поточний стан, коли ID надто старий, щоб із нього продовжити. Специфікація MCP додає правило, яке варто перейняти: ID належать одному потоку, і сервер не повинен повторно надсилати в одному потоці повідомлення, призначені для іншого.

Повторне під'єднання, яке не припиняється, або припиняється назавжди. Потік, що завершується звичайним чином, повертається після часу повторного під'єднання знову й знову, тож завершене завдання постійно відкриває свій потік наново, якщо сторінка не викличе `close()` або сервер не відповість на повторне під'єднання статусом 204. Протилежне трапляється під час розгортання. Проксі, що відповідає 502 чи 503, поки сервери перезапускаються, дає кожному відкритому `EventSource` статус, відмінний від 200, і всі вони припиняють спроби назавжди. Слухайте `error`, і коли `readyState` має значення `CLOSED`, створюйте новий `EventSource` після випадкової паузи, яка зростає з кожною невдачею.

Забагато потоків для одного браузера. Браузери обмежують кількість з'єднань, які тримають відкритими до одного сервера, а потік, що йде через HTTP/1.1, займає одне з них, доки триває. HTML Standard попереджає, що сайт, де кожна відкрита сторінка має власний `EventSource`, може впертися в це обмеження. Віддавайте потік через HTTP/2, де потоки ділять одне з'єднання, а RFC 9113 рекомендує дозволяти щонайменше 100 одночасно, або діліть один `EventSource` між вкладками через спільний воркер (shared worker).

Токен в адресі. Оскільки `EventSource` не може надіслати заголовок `Authorization`, швидке рішення — токен в URL, де його записують зворотні проксі та журнали доступу. Автентифікуйте потік сесійним файлом cookie, який сторінка вже має, додаючи `withCredentials`, коли потік надходить з іншого джерела, або короткочасним одноразовим квитком, який сторінка отримує автентифікованим запитом безпосередньо перед тим, як відкрити потік.

Що почитати далі

- HTML Standard: Server-sent events — формат потоку, інтерфейс `EventSource`, повторне під'єднання та поради авторам щодо проксі й обмежень на кількість з'єднань.
- Transports - Model Context Protocol — Streamable HTTP, де сервер може відповісти потоком подій, і те, як обірваний потік продовжується.
- RFC 9113: HTTP/2 — потоки, що ділять одне з'єднання, і параметр, який обмежує, скільки їх може бути відкрито одночасно.
- nginx: Module ngx_http_proxy_module — `proxy_buffering`, заголовок `X-Accel-Buffering` і `proxy_read_timeout`.

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 server-sent events?» 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/server-sent-events/