Engineer

Server-Sent Events

Ein unidirektionaler Ereignisstrom vom Server über HTTP

Was es ist

Server-Sent Events ermöglichen einem Webserver, Nachrichten über eine einzige gewöhnliche HTTP-Antwort an eine Seite zu senden. Er hält sie offen und schreibt hinein, sobald es Neuigkeiten gibt. Definiert sind sie im HTML-Standard der WHATWG, der sowohl das Textformat des Streams als auch `EventSource` beschreibt, die Browser-Schnittstelle zum Lesen dieses Streams. Der Name wird häufig zu SSE verkürzt; das Format wird oft nach seinem Medientyp `text/event-stream` benannt.

Eine Seite öffnet einen Stream mit einer Zeile: `new EventSource(url)` . Der Browser sendet eine `GET` -Anfrage, die `Accept: text/event-stream` enthalten kann, und erwartet eine Antwort mit Status 200 und diesem Inhaltstyp. Der Server sendet die Header und schließt den Antwortkörper danach nicht ab. Sobald etwas geschieht, schreibt er einige Textzeilen hinein. Der Browser macht aus jedem Zeilenblock ein Ereignis, auf das die Seite reagieren kann.

Das Format ist bewusst einfach. Ein Stream ist immer UTF-8, und jede Zeile enthält ein Feld. Eine `data` -Zeile trägt die Nutzdaten; mehrere aufeinanderfolgende `data` -Zeilen werden mit Zeilenumbrüchen verbunden. Eine `event` -Zeile gibt dem Ereignis einen Typ. Fehlt sie, lautet der Typ `message` . Eine `id` -Zeile setzt die ID des letzten Ereignisses, und eine `retry` -Zeile legt fest, wie viele Millisekunden der Browser vor einem erneuten Verbindungsaufbau wartet. Eine mit einem Doppelpunkt beginnende Zeile ist ein Kommentar, den der Browser ignoriert. Eine Leerzeile schließt den Block ab und löst die Zustellung des Ereignisses aus. Ein Block, der vor seiner Leerzeile abbricht, wird deshalb verworfen.

Der erneute Verbindungsaufbau gehört zum Standard. Bricht die Verbindung ab, wartet der Browser die Wiederverbindungszeit ab und sendet die Anfrage erneut. Den Anfangswert dieser Wartezeit wählt er selbst; laut Standard sind es vermutlich einige Sekunden. Hat der Server seinen Ereignissen IDs gegeben, enthält die neue Anfrage die letzte davon im Header `Last-Event-ID`, und der Server kann dort fortsetzen. Manche Antworten beenden den Stream dagegen endgültig: Ein anderer Status als 200 oder ein anderer Inhaltstyp lässt den Browser aufgeben und weitere Versuche einstellen. Der Standard nennt 204 No Content als Möglichkeit für den Server, das Ende des Streams mitzuteilen.

Der Datenfluss ist unidirektional. Nach der einleitenden Anfrage gelangt auf dieser Verbindung nichts von der Seite zum Server. Hat die Seite etwas mitzuteilen, sendet sie eine eigene HTTP-Anfrage.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was sind Server-Sent Events?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Der Teil, den Nutzer nie sehen: wo die Daten liegen, welche Regeln gelten und wie andere Systeme darauf zugreifen dürfen.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. 6 ETL/ELT-Pipelines entworfen und betrieben — 6 ETL-Pipelines über drei Datenbankengines hinweg

6 ETL/ELT-Pipelines entworfen und betrieben Kontakt aufnehmen →

Warum es wichtig ist

Viele Funktionen, die wie eine Live-Ansicht wirken, brauchen nur eine Richtung: den Fortschritt eines langen Jobs, eine Benachrichtigung, eine Kennzahl auf einem Dashboard oder eine stückweise eintreffende Antwort. Ohne Stream muss die Seite regelmäßig nachfragen. Sie fragt alle paar Sekunden, die meisten Antworten melden keine Änderung, und Neuigkeiten warten bis zu einem ganzen Intervall, bevor jemand sie sieht. Mit einem Stream schreibt der Server, sobald etwas geschieht.

Die übliche Alternative zum Senden neuer Daten ist WebSocket. Dabei wechselt die Verbindung von HTTP zu einem eigenen Protokoll. Server-Sent Events bleiben von Anfang bis Ende innerhalb von HTTP. Sie durchlaufen daher dieselben Proxys und Lastverteiler wie jede andere Anfrage, verwenden dieselben Cookies und Regeln für ursprungsübergreifende Anfragen und können sich mit der übrigen Seite eine HTTP/2-Verbindung teilen. Wiederverbindung und Fortsetzen ab dem

letzten Ereignis bringt der Browser mit, statt dass jedes Team den Code selbst schreibt. WebSocket bleibt die richtige Wahl, wenn der Client ebenso viel mitzuteilen hat wie der Server, etwa in einem Chat oder einem gemeinsam genutzten Editor.

Das Format ist auch nicht an den Browser gebunden. Jeder HTTP-Client kann es lesen, und auch Spezifikationen außerhalb des Webs nutzen es. Im Model Context Protocol, über das KI-Agenten Werkzeuge aufrufen, kann ein Server mit dem Transport Streamable HTTP eine Anfrage entweder mit einem JSON-Objekt oder mit einer `text/event-stream`-Antwort beantworten. In diesem Stream darf er vor der abschließenden Antwort eigene Benachrichtigungen und Anfragen senden. Er kann Ereignis-IDs beifügen, damit ein Client nach einem Verbindungsabbruch mit `Last-Event-ID` fortsetzt. Streamable HTTP ersetzte HTTP+SSE, den Transport der früheren Protokollversion 2024-11-05.

Vor der Entscheidung sollte man die Grenzen kennen. Der Stream trägt ausschließlich Text; Binärdaten müssen deshalb zuerst als Text kodiert werden. Der Konstruktor `EventSource` im Browser nimmt nur eine URL und eine Option, `withCredentials`, entgegen. Eine Seite kann der Anfrage daher keinen `Authorization`-Header hinzufügen.

Wie es in der Praxis aussieht

Der folgende Austausch zeigt einen langen Exportjob. Der Browser hat nach Ereignis 41 die Verbindung verloren und verbindet sich erneut. Seine Anfrage enthält deshalb diese ID, und der Server setzt bei Ereignis 42 fort:

```
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"}
```

Die Zeile `retry` fordert den Browser auf, vor jedem späteren erneuten Verbindungsaufbau fünf Sekunden zu warten. Die Kommentarzeile enthält überhaupt kein Ereignis; sie hält die Verbindung beschäftigt, während der Job nichts meldet. Die Seite wartet anhand ihrer Namen auf die beiden Ereignistypen:

```
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();
});
```

Der Aufruf von `close()` ist wichtig. Wenn der Server eine Antwort regulär beendet, behandelt der Browser das als Verbindungsabbruch und verbindet sich erneut. Der Stream eines abgeschlossenen Jobs würde sonst nach fünf Sekunden wieder geöffnet. Der Handler für `error` unterscheidet die beiden Fehlerarten: Solange `readyState` den Wert `CONNECTING` hat, versucht der Browser es weiter; bei `CLOSED` hat er aufgegeben.

Arbeit, die nach der Antwort an den Agenten weiterläuft, behandelt die Seite dieser Website zum Bereitstellen vorhandener Software als Werkzeuge für KI-Agenten. Sie beschreibt, wie die Abläufe eines Unternehmens als Werkzeuge angeboten werden. Ein MCP-Server stellt dieselben Abläufe den Agenten bereit, die dieses Protokoll sprechen. Langsame Arbeit wandert in eine Warteschlange, damit der Agent eine Antwort erhält, während die Arbeit weiterläuft. In der MCP-

Spezifikation kann ein Server einen Ereignisstream wie oben verwenden, um vor der abschließenden Antwort auf die Anfrage des Clients eigene Anfragen oder Benachrichtigungen zu senden.

Wo es schiefgeht

Ereignisse kommen schubweise oder erst ganz am Ende an. Etwas zwischen Server und Browser hält die Antwort zurück. nginx etwa puffert Antworten des dahinterliegenden Servers standardmäßig und leitet eine Antwort nur bei abgeschalteter Pufferung schon während ihres Eintreffens weiter. Auch der HTML-Standard warnt: HTTP-Chunking durch eine Schicht, die den zeitlichen Ablauf des Streams nicht kennt, kann ihn unzuverlässig machen. Schalten Sie die Pufferung für den Pfad des Streams ab, bei nginx mit `proxy_buffering off` oder einem Antwortheader `X-Accel-Buffering: no` . Leeren Sie im Servercode nach jedem Ereignis den Ausgabepuffer und testen Sie den gesamten Weg einschließlich Proxy statt nur den Server.

Ein Timeout trennt stille Streams. Ein Proxy, der eine Zeit lang keine Bytes sieht, hält die Verbindung für tot. Bei nginx beträgt `proxy_read_timeout` standardmäßig 60 Sekunden zwischen zwei Lesevorgängen; danach wird die Verbindung geschlossen. Der HTML-Standard warnt außerdem davor, dass manche älteren Proxys Verbindungen nach einem kurzen Timeout trennen. Er empfiehlt etwa alle 15 Sekunden eine Kommentarzeile. Sie kostet wenige Bytes und wird vom Browser ignoriert.

Bei einer Wiederverbindung gehen Ereignisse verloren. Der Browser sendet `Last-Event-ID` nur, wenn der Server seinen Ereignissen IDs zugewiesen hat. Der Header hilft nur, wenn der Server noch finden kann, was nach dieser ID kam. Geben Sie jedem Ereignis eine ID, die als Cursor im Stream dient, halten Sie ein Fenster jüngerer Ereignisse zur erneuten Zustellung vor und senden Sie den vollständigen aktuellen Zustand, wenn eine ID zum Fortsetzen zu alt ist. Die MCP-Spezifikation ergänzt eine übernehmenswerte Regel: IDs gehören zu einem Stream, und ein Server darf in einem Stream keine Nachrichten erneut zustellen, die für einen anderen bestimmt sind.

Die Wiederverbindung endet nie oder endet endgültig. Ein regulär beendeter Stream kehrt nach Ablauf der Wiederverbindungszeit immer wieder zurück. Ein abgeschlossener Job öffnet seinen Stream deshalb ständig neu, wenn die Seite nicht `close()` aufruft und der Server die erneute Anfrage nicht mit 204 beantwortet. Bei einer Bereitstellung geschieht das Gegenteil. Antwortet ein

Proxy während des Serverneustarts mit 502 oder 503, erhält jede offene `EventSource` einen anderen Status als 200 und gibt endgültig auf. Reagieren Sie auf `error` und erzeugen Sie bei `readyState` gleich `CLOSED` nach einer zufälligen, mit jedem Fehler wachsenden Verzögerung eine neue `EventSource`.

Ein Browser hat zu viele Streams offen. Browser begrenzen die Zahl gleichzeitig offener Verbindungen zu einem Server. Ein über HTTP/1.1 ausgelieferter Stream belegt für seine gesamte Dauer eine davon. Der HTML-Standard warnt, dass eine Website mit einer eigenen `EventSource` je geöffneter Seite an diese Grenze stoßen kann. Liefern Sie den Stream über HTTP/2 aus: Dort teilen sich Streams eine Verbindung, und RFC 9113 empfiehlt, mindestens 100 Streams gleichzeitig zuzulassen. Oder teilen Sie über einen Shared Worker eine `EventSource` zwischen den Tabs.

Das Token steht in der Adresse. Weil `EventSource` keinen `Authorization`-Header senden kann, ist ein Token in der URL die schnelle Lösung. Dort schreiben Reverse-Proxys und Zugriffsprotokolle es jedoch mit. Authentifizieren Sie den Stream über das Sitzungscookie, das die Seite bereits hat, und ergänzen Sie bei einem anderen Ursprung `withCredentials`. Oder verwenden Sie ein kurzlebiges, einmal verwendbares Zugangsticket, das die Seite unmittelbar vor dem Öffnen des Streams über eine authentifizierte Anfrage abruft.

Weiterführende Lektüre

- HTML-Standard: Server-Sent Events — Streamformat, Schnittstelle `EventSource`, Wiederverbindung und Hinweise für Autoren zu Proxys und Verbindungsgrenzen.
- Transports – Model Context Protocol — Streamable HTTP, bei dem ein Server mit einem Ereignisstream antworten kann, und die Fortsetzung eines abgebrochenen Streams.
- RFC 9113: HTTP/2 — Streams, die sich eine Verbindung teilen, und die Einstellung zur Begrenzung gleichzeitig offener Streams.
- nginx: Modul ngx_http_proxy_module — `proxy_buffering`, der Header `X-Accel-Buffering` und `proxy_read_timeout`.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Arbeitsablauf, die vorhandenen Schnittstellen und die Nutzer, die die Veränderung benötigen. Wir können eine prüfbare Lieferung definieren und die Übergabe so gestalten, dass Ihr Team die gelieferten Bestandteile warten kann.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was sind Server-Sent Events?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/server-sent-events/