Engineer

Événements envoyés par le serveur

Flux unidirectionnel d’événements du serveur via HTTP

Ce que c’est

Les événements envoyés par le serveur permettent à un serveur web de pousser des messages vers une page au moyen d’une réponse HTTP ordinaire qu’il garde ouverte et dans laquelle il écrit dès qu’il y a du nouveau. Le standard HTML, maintenu par le WHATWG, définit à la fois le format textuel du flux et `EventSource`, l’interface du navigateur qui le lit. Le nom est souvent abrégé en SSE, et le format est souvent désigné par son type de média, `text/event-stream`.

Une page ouvre un flux avec une seule ligne, `new EventSource(url)`. Le navigateur envoie une requête `GET`, qui peut comporter `Accept: text/event-stream`, et attend une réponse de statut 200 avec ce type de contenu. Le serveur envoie les en-têtes, puis laisse le corps ouvert. Chaque fois qu’il se passe quelque chose, il écrit quelques lignes de texte. Le navigateur transforme chaque bloc de lignes en un événement que la page peut écouter.

Le format est volontairement simple. Un flux est toujours en UTF-8, et chaque ligne correspond à un champ. Une ligne `data` porte la charge utile ; plusieurs lignes `data` consécutives sont réunies avec des sauts de ligne. Une ligne `event` donne un type à l’événement ; sans elle, le type est `message`. Une ligne `id` définit l’identifiant du dernier événement, et une ligne `retry` fixe le nombre de millisecondes que le navigateur attend avant de se reconnecter. Une ligne commençant par deux-points est un commentaire, ignoré par le navigateur. Une ligne vide termine le bloc et déclenche l’événement : un bloc interrompu avant sa ligne vide est donc abandonné.

La reconnexion fait partie du standard. Si la connexion tombe, le navigateur attend le délai de reconnexion, dont il choisit lui-même la valeur initiale, probablement quelques secondes selon le standard, puis renvoie la requête. Si le serveur a attribué des identifiants aux événements, la nouvelle requête porte le

dernier dans un en-tête `Last-Event-ID`, et le serveur peut reprendre à partir de là. Certaines réponses mettent au contraire fin au flux : un statut autre que 200 ou un type de contenu différent amène le navigateur à abandonner ses tentatives. Le standard désigne 204 No Content comme le moyen pour le serveur d'indiquer que le flux est terminé.

La communication est strictement unidirectionnelle. Après la requête d'ouverture, rien ne va de la page vers le serveur sur cette connexion. Lorsque la page a quelque chose à dire, elle envoie une requête HTTP distincte.

Utilisez le concept pour prendre une décision

Utilisez « Que sont les événements envoyés par le serveur ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. La partie invisible pour les utilisateurs : où résident les données, quelles règles s'appliquent et comment les autres systèmes peuvent y accéder.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. 6 pipelines ETL/ELT conçus et exploités — 6 pipelines ETL sur trois moteurs de bases de données

6 pipelines ETL/ELT conçus et exploités Contactez-nous →

Pourquoi c'est important

Beaucoup de fonctionnalités qui semblent fonctionner en direct n'ont besoin que d'un sens de communication : l'avancement d'un traitement long, une notification, un chiffre sur un tableau de bord ou une réponse qui arrive par morceaux. Sans flux, la page doit interroger périodiquement le serveur. Elle renouvelle sa demande à quelques secondes d'intervalle, reçoit le plus souvent une réponse indiquant que rien n'a changé, et une nouveauté peut attendre tout un intervalle avant d'être visible. Avec un flux, le serveur écrit dès qu'un événement se produit.

L'autre solution habituelle pour pousser des données est WebSocket, qui fait passer la connexion de HTTP à son propre protocole. Les événements envoyés par le serveur restent dans HTTP du début à la fin : ils traversent les mêmes mandataires et répartiteurs de charge que toute autre requête, utilisent les mêmes cookies et les mêmes règles de requêtes entre origines, et peuvent partager une connexion HTTP/2 avec le reste de la page. La reconnexion et la reprise au dernier événement sont fournies par le navigateur, au lieu de devoir

être réécrites par chaque équipe. WebSocket reste le bon choix lorsque le client a autant à dire que le serveur, comme dans une messagerie instantanée ou un éditeur partagé.

Le format n'est pas non plus réservé au navigateur. Tout client HTTP peut le lire, et des spécifications extérieures au web l'utilisent. Dans le Model Context Protocol, par lequel les agents d'IA appellent des outils, le transport Streamable HTTP permet au serveur de répondre à une requête soit par un objet JSON, soit par une réponse `text/event-stream`. Sur ce flux, il peut envoyer ses propres notifications et requêtes avant la réponse finale, et ajouter des identifiants d'événements pour qu'un client déconnecté reprenne avec `Last-Event-ID`. Streamable HTTP a remplacé HTTP+SSE, le transport de la version précédente du protocole, 2024-11-05.

Il faut connaître les limites avant de choisir ce mécanisme. Le flux ne transporte que du texte : les données binaires doivent d'abord être encodées sous forme textuelle. Le constructeur `EventSource` du navigateur accepte une URL et une seule option, `withCredentials`, rien d'autre. La page ne peut donc pas ajouter d'en-tête `Authorization` à la requête.

À quoi cela ressemble en pratique

L'échange ci-dessous suit une longue tâche d'export. Le navigateur a perdu sa connexion après l'événement 41 et se reconnecte : sa requête porte donc cet identifiant, et le serveur reprend à l'événement 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"}
```

La ligne `retry` demande au navigateur d'attendre cinq secondes avant toute reconnexion ultérieure. La ligne de commentaire ne porte aucun événement : elle maintient de l'activité sur la connexion lorsque la tâche n'émet rien. La page écoute les deux types d'événements par leur nom :

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

L'appel à `close()` est important. Quand le serveur termine normalement une réponse, le navigateur y voit une connexion interrompue et se reconnecte. Sans cet appel, le flux d'une tâche terminée s'ouvrirait donc à nouveau après cinq secondes. Le gestionnaire `error` distingue les deux types de défaillance : tant que `readyState` vaut `CONNECTING`, le navigateur essaie encore ; lorsqu'il vaut `CLOSED`, il a abandonné.

Le travail qui continue après que l'agent a reçu sa réponse est abordé dans la page de ce site consacrée à la transformation de logiciels existants en outils pour agents d'IA. Elle décrit comment proposer les opérations d'une entreprise sous forme d'outils, un serveur MCP offrant les mêmes opérations aux agents qui parlent ce protocole, et comment déporter les traitements lents vers une file pour que l'agent reçoive une réponse pendant que le travail continue. Dans la spécification MCP, un serveur peut utiliser un flux d'événements comme celui ci-dessus pour envoyer ses propres requêtes ou notifications avant la réponse finale à la requête du client.

Où les problèmes surviennent

Les événements arrivent par paquets, ou tous à la fin. Un élément entre le serveur et le navigateur retient la réponse. nginx, par exemple, met par défaut en tampon les réponses du serveur situé derrière lui. Il ne les transmet au fil de leur arrivée que si cette mise en tampon est désactivée. Le standard HTML avertit aussi qu'un découpage HTTP en blocs effectué par une couche qui ignore le rythme du flux peut le rendre peu fiable. Désactivez la mise en tampon pour l'emplacement du flux, avec `proxy_buffering off` dans nginx ou l'en-tête de réponse `X-Accel-Buffering: no`. Forcez l'envoi de la sortie après chaque événement dans le code serveur et testez le trajet complet, mandataire compris, plutôt que le seul serveur.

Un délai d'inactivité coupe les flux silencieux. Un mandataire qui ne voit passer aucun octet pendant un certain temps suppose que la connexion est morte. Dans nginx, `proxy_read_timeout` vaut par défaut 60 secondes entre deux lectures, après quoi la connexion est fermée. Le standard HTML avertit que certains mandataires anciens coupent les connexions après un délai court. Il suggère une ligne de commentaire toutes les 15 secondes environ, pour quelques octets que le navigateur ignore.

Des événements sont perdus pendant la reconnexion. Le navigateur n'envoie `Last-Event-ID` que si le serveur a donné des identifiants aux événements. Cet en-tête n'est utile que si le serveur retrouve encore ce qui suit cet identifiant. Donnez à chaque événement un identifiant servant de curseur dans le flux, conservez une fenêtre d'événements récents à rejouer et envoyez l'état courant complet lorsqu'un identifiant est trop ancien pour reprendre. La spécification MCP ajoute une règle à retenir : les identifiants appartiennent à un flux, et le serveur ne doit pas rejouer sur un flux des messages destinés à un autre.

La reconnexion ne s'arrête jamais, ou s'arrête définitivement. Un flux qui se termine normalement revient après le délai de reconnexion, encore et encore. Une tâche terminée rouvre donc son flux si la page n'appelle pas `close()` et si le serveur ne répond pas à la reconnexion par un statut 204. Le phénomène inverse survient pendant un déploiement. Un mandataire qui répond 502 ou 503 lors du redémarrage des serveurs donne à chaque `EventSource` ouvert un statut autre que 200 : tous s'arrêtent définitivement. Écoutez `error` et, lorsque `readyState` vaut `CLOSED`, créez un nouvel `EventSource` après un délai aléatoire qui augmente à chaque échec.

Un navigateur ouvre trop de flux. Les navigateurs limitent le nombre de connexions qu'ils gardent ouvertes vers un même serveur. Un flux en HTTP/1.1 en occupe une pendant toute sa durée. Le standard HTML avertit qu'un site où chaque page ouverte possède son propre `EventSource` peut atteindre cette limite. Servez le flux en HTTP/2, où les flux partagent une connexion et où la RFC 9113 recommande d'en autoriser au moins 100 simultanément, ou partagez un `EventSource` entre les onglets au moyen d'un worker partagé.

Le jeton se trouve dans l'adresse. Comme `EventSource` ne peut pas envoyer d'en-tête `Authorization`, la solution rapide consiste à mettre un jeton dans l'URL, où les mandataires inverses et les journaux d'accès le consignent. Authentifiez le flux avec le cookie de session que possède déjà la page, en ajoutant `withCredentials` si le flux provient d'une autre origine, ou utilisez un ticket à usage unique et à courte durée de vie, récupéré par la page au moyen d'une requête authentifiée juste avant l'ouverture du flux.

Pour aller plus loin

- Standard HTML : événements envoyés par le serveur — Le format du flux, l'interface `EventSource`, la reconnexion et les recommandations aux auteurs sur les mandataires et les limites de connexion.
- Transports — Model Context Protocol — Streamable HTTP, qui permet au serveur de répondre par un flux d'événements, et la reprise d'un flux interrompu.
- RFC 9113 : HTTP/2 — Les flux qui partagent une connexion et le paramètre qui limite le nombre de flux ouverts simultanément.
- nginx : module ngx_http_proxy_module — `proxy_buffering`, l'en-tête `X-Accel-Buffering` et `proxy_read_timeout`.



Glossaire

Choisissez selon vos contraintes réelles

Présentez le processus de travail, les interfaces existantes et les utilisateurs qui ont besoin du changement. Nous pouvons définir une livraison testable et une transmission permettant à votre équipe d'assurer la maintenance des éléments livrés.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Que sont les événements envoyés par le serveur ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

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

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

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

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/server-sent-events/