Engineer

サーバー送信イベント

🔠 HTTP経由でサーバーから一方向にイベントを送り続ける仕組み

どのようなものか

サーバー送信イベントは、ウェブサーバーからページへメッセージを送る仕組みです。通常のHTTP応答を1本だけ開いたままにし、新しい情報があるたびに書き込みます。WHATWGが管理するHTML Standardに定義されており、ストリームのテキスト形式と、それを読むブラウザーのインターフェースである `EventSource` の両方が規定されています。名前はSSEと略されることが多く、形式そのものもメディアタイプにちなんで `text/event-stream` と呼ばれることがあります。

ページは `new EventSource(url)` という1行でストリームを開きます。ブラウザーは `GET` リクエストを送り、そこには `Accept: text/event-stream` が含まれる場合があります。期待するのは、ステータス200で、そのコンテンツタイプを持つ応答です。サーバーはヘッダーを送った後、本文を終了しません。何かが起きるたびに数行のテキストを書き込み、ブラウザーはその行のまとまりを、ページ側で受け取れるイベントに変換します。

形式は意図的に単純に作られています。ストリームは常にUTF-8で、各行が1つのフィールドです。 `data` 行はペイロードを運び、複数の `data` 行が続くと改行を挟んで結合されます。 `event` 行はイベントの型を指定し、省略した場合の型は `message` です。 `id` 行は最後のイベントのIDを設定し、 `retry` 行は再接続までにブラウザーが待つ時間をミリ秒単位で指定します。コロンで始まる行はコメントで、ブラウザーは無視します。空行でブロックが終わり、イベントが配信されるため、空行に達する前に途切れたブロックは破棄されます。

再接続も標準の一部です。接続が切れると、ブラウザーは再接続までの時間を待ってから、もう一度リクエストを送ります。この時間の初期値はブラウザーが決めるもので、標準ではおそらく数秒とされています。サーバーがイベントにIDを付けていれば、新しいリクエストの `Last-Event-ID` ヘッダーに最後のIDが入り、サーバーはその続きから送信できます。一方、ストリームを完全に終わらせる応

答もあります。200以外のステータス、または異なるコンテンツタイプを受け取ると、ブラウザーは接続を諦め、再試行をやめます。サーバーからストリームの終了を伝える方法として、標準は204 No Contentを挙げています。

通信方向は一方向だけです。最初のリクエストの後、その接続を通じてページからサーバーへ送るものはありません。ページ側から何かを伝えるときは、別のHTTPリクエストを送ります。

概念を判断に生かす

「サーバー送信イベントとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 データの保存先、処理のルール、他のシステムとの接続方法など、利用者の目に見えない部分をつくります。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 ETL/ELTパイプライン6本を設計・運用 — 6 3種類のDBエンジンにまたがるETLパイプライン

ETL/ELTパイプライン6本を設計・運用 お問い合わせ →

なぜ重要なのか

リアルタイムに見える機能の多くは、一方向の通信だけで足ります。時間のかかる処理の進捗、通知、ダッシュボードの数値、少しずつ届く回答などです。ストリームを使わなければ、ページはポーリングする必要があります。数秒おきに問い合わせても、たいていの応答は変化がないことを知らせるだけで、新しい情報が表示されるまで最大で問い合わせ間隔と同じだけ待つことになります。ストリームなら、何かが起きた時点でサーバーが書き込みます。

データをサーバーから送る際によく使われる別の選択肢がWebSocketです。これは接続をHTTPから独自のプロトコルへ切り替えます。サーバー送信イベントは最初から最後までHTTPの中に収まるため、他のリクエストと同じプロキシやロードバランサーを通り、同じCookieとクロスオリジンリクエストの規則を使い、ページの他の通信と1本のHTTP/2接続を共有できます。再接続や最後のイベントからの再開はブラウザーに備わっており、チームごとに実装する必要がありません。チャットや共同編集のように、クライアントからもサーバーと同程度に送信する場合は、引き続きWebSocketが適しています。

この形式はブラウザー専用でもありません。どのHTTPクライアントでも読め、ウェブ以外の仕様でも使われています。AIエージェントがツールを呼び出すModel Context Protocolでは、Streamable HTTPトランスポートにより、サーバーは1つのJSONオブジェクトか `text/event-stream` 応答のどちらかでリクエストに答えられます。ストリーム上では、最終応答の前にサーバー自身から通知やリ

クエストを送れます。また、イベントIDを付ければ、接続を失ったクライアントが `Last-Event-ID` を使って再開できます。Streamable HTTPは、以前のプロトコルバージョン2024-11-05で使われていたHTTP+SSEトランスポートを置き換えたものです。

選ぶ前に制約も理解しておく必要があります。ストリームが運べるのはテキストだけなので、バイナリデータは先にテキストへ符号化しなければなりません。ブラウザーの `EventSource` コンストラクターが受け取るのはURLと `withCredentials` という1つのオプションだけです。そのため、ページ側からリクエストに `Authorization` ヘッダーを追加することはできません。

実際の活用例

次のやり取りは、時間のかかるエクスポート処理を追うものです。ブラウザーはイベント41の後で接続を失い、再接続しています。そのため、リクエストにはそのIDが含まれ、サーバーはイベント42から送信を続けます。

```
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"}
```

`retry` 行は、それ以降の再接続まで5秒待つようブラウザーに求めます。コメント行はイベントを一切運びません。処理に動きがない間も接続に通信を流しておくためのものです。ページは、2種類のイベントを名前で指定して受け取ります。

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

`close()` の呼び出しが重要です。サーバーが応答を正常に終了しても、ブラウザーは接続が途切れたものとして再接続します。そのため、この呼び出しがなければ、完了した処理のストリームが5秒後にまた開かれます。 `error` ハンドラーは2種類の失敗を区別します。 `readyState` が `CONNECTING` の間はまだ接続を試みていますが、 `CLOSED` になった時点で諦めています。

エージェントが回答を得た後も続く処理は、当サイトの既存ソフトウェアをAIエージェント用のツールにするページでも扱っています。企業の業務操作をツールとして提供し、MCPを扱えるエージェントにはMCPサーバーから同じ操作を提供する仕組みです。時間のかかる処理はキューへ移すことで、処理が続く間にエージェントへ回答を返します。MCP仕様では、サーバーは上の例のようなイベントストリームを使い、クライアントのリクエストに対する最終応答の前に、自身のリクエストや通知を送れます。

問題が起きるところ

イベントがまとめて届く、または最後に全部届きます。 サーバーとブラウザーの間にある何かが、応答をせき止めています。例えばnginxは、背後のサーバーからの応答を既定でバッファリングし、バッファリングを無効にした場合だけ、受け取るそばから応答を転送します。HTML Standardも、ストリームの送信タイミングを知らない層がHTTPのチャンク分割を行うと、信頼性を損なう可能性があると警告しています。ストリームを配信する場所のバッファリングを無効にし

てください。nginxでは `proxy_buffering off` 、または `X-Accel-Buffering: no` 応答ヘッダーを使います。サーバーのコードではイベントごとに出力をフラッシュし、サーバー単体ではなく、プロキシを含む経路全体でテストします。

動きのないストリームがタイムアウトで切れます。 しばらくデータが流れないと、プロキシは接続が失われたと判断します。nginxの `proxy_read_timeout` は、2回の読み取りの間隔について既定値が60秒で、それを過ぎると接続を閉じます。HTML Standardも、一部の古いプロキシは短いタイムアウトで接続を切ると警告しています。対策として、15秒程度おきにコメント行を送ることを勧めています。数バイトで済み、ブラウザーは無視します。

再接続時にイベントが失われます。 ブラウザーが `Last-Event-ID` を送るのは、サーバーがイベントにIDを付けていた場合だけです。また、このヘッダーが役立つのも、サーバーがそのIDの後に起きたことをまだ取り出せる場合だけです。すべてのイベントにストリーム内のカーソルとして使えるIDを付け、最近のイベントを一定範囲保存して再送できるようにし、古すぎるIDからは再開できない場合には現在の状態全体を送ります。MCP仕様には、取り入れる価値のある規則もあります。IDは1つのストリームに属し、別のストリーム向けのメッセージを他のストリームで再送してはいけません。

再接続が止まらないか、完全に止まります。 正常に終了したストリームは、再接続までの時間が過ぎるたびに何度でも再開されます。そのため、ページが `close()` を呼ばず、サーバーも再接続に204で応答しなければ、完了した処理のストリームが開き直され続けます。デプロイ中は逆のことが起きます。サーバーの再起動中にプロキシが502や503を返すと、開いているすべての `EventSource` が200以外のステータスを受け取り、どれも完全に停止します。 `error` を受け取り、 `readyState` が `CLOSED` なら、失敗するたびに長くなるランダムな遅延を挟んで、新しい `EventSource` を作成します。

1つのブラウザーでストリームを開きすぎます。 ブラウザーには、1つのサーバーに対して同時に開ける接続数の上限があります。HTTP/1.1で配信するストリームは、続いている間ずっとそのうち1本を占有します。HTML Standardは、開いているページごとに専用の `EventSource` があるサイトでは、この上限に達し得ると警告しています。ストリームをHTTP/2で配信すれば、複数のストリームで1本の接続を共有でき、RFC 9113は少なくとも100本のストリームを同時に許可することを推奨しています。あるいは、共有ワーカーを介してタブ間で1つの `EventSource` を共有します。

アドレスにトークンが入ります。 `EventSource` は `Authorization` ヘッダーを送れないため、手早い対処としてURLにトークンを入れがちです。しかし、そのURLはリバースプロキシやアクセスログに記録されます。ストリームは、ページがすでに持つセッションCookieで認証し、別オリジンから取得する場合は `withCredentials` を付けます。または、ストリームを開く直前に認証済みリクエストで取得する、有効期間が短く1回だけ使えるチケットで認証します。

参考資料

- HTML Standard：サーバー送信イベント — ストリーム形式、`EventSource` インターフェース、再接続、プロキシと接続数上限に関する作成者向けの注意事項を説明しています。
- トランスポート — Model Context Protocol — サーバーがイベントストリームで応答できるStreamable HTTPと、切れたストリームの再開方法を説明しています。
- RFC 9113：HTTP/2 — 1本の接続を共有するストリームと、同時に開ける本数を制限する設定を説明しています。
- nginx：ngx_http_proxy_moduleモジュール — `proxy_buffering`、`X-Accel-Buffering` ヘッダー、`proxy_read_timeout` を説明しています。

用語集

実際の制約に合わせて選ぶ

業務の流れ、既存のインターフェース、変更を必要とする利用者をお知らせください。検証可能な納品内容と、お客様のチームが保守を続けられる引き継ぎ内容を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「サーバー送信イベントとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

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

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/server-sent-events/