Engineer

WebSocket

↔ 開いた接続を維持し、双方向に通信するプロトコル

どのようなものか

WebSocketは、接続の両側がいつでもデータを送れる通信プロトコルです。HTTPのハンドシェイクで接続を確立し、どちらかが閉じるか、接続に障害が起きるまで開いたままにします。IETFは2011年12月にRFC 6455として公開しました。ブラウザーがスクリプトに公開する `WebSocket` オブジェクトは、WHATWGのWebSockets Standardで定義されています。コードや文書ではwebsocketやweb socketともよく書かれ、複数形のWebSocketsは通常、この技術全体を指します。

従来のHTTP/1.1のハンドシェイクは、通常の `GET` から始まります。クライアントは `Upgrade: websocket` と `Connection: Upgrade` 、値が13の `Sec-WebSocket-Version` 、ランダムな16バイトをbase64で符号化した `Sec-WebSocket-Key` を加えます。受け入れるサーバーは、101 Switching Protocolsと `Sec-WebSocket-Accept` ヘッダーを返します。この値は、古い応答の再送ではなく、今回のハンドシェイクをサーバーが読んだことを示します。クライアントのキーに固定のGUIDを足し、SHA-1でハッシュ化してbase64で符号化したものです。この時点で、その接続上のHTTPは終わり、双方はフレームを交換します。一方、HTTP/2とHTTP/3は、RFC 8441とRFC 9220が定める拡張CONNECTを使い、1本のストリーム上でWebSocketを扱います。

フレームは小さな単位です。ヘッダーは2バイトから始まり、ペイロードが125バイトを超えると、さらに2バイトまたは8バイト増えます。クライアントからのフレームにはマスキングキーがあるため、さらに4バイト加わります。各フレームはオペコードを持ち、テキスト、バイナリ、複数フレームに分割したメッセージの継続を区別します。このほか、終了、ping、pongの3種類の制御フレームがあります。テキストメッセージ全体は正しいUTF-8でなければなりませんが、個々の断片ではUTF-8の並びの途中で分割できます。クライアントが送るすべてのフレームは、新しいランダムな32ビットキーでマスクし、サーバーは一切マ

スクしません。マスキングは暗号化ではありません。悪意のあるスクリプトがネットワークを流れるバイト列を正確に選び、途中のキャッシュプロキシを汚染するために使うことを防ぐ仕組みです。

アドレスのスキームは `ws://` と `wss://` で、指定しなければ、それぞれポート80と443を使います。 `wss` では、やり取り全体をTLSの内側で行います。HTTPと同じポートを使うのは意図的です。途中のファイアウォールやプロキシには、ハンドシェイクがほかのウェブリクエストと同じように見えるためです。

概念を判断に生かす

「WebSocket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

短いHTTP応答では、サーバーは問い合わせられたときにしか話せません。多くの製品は、その逆を必要とします。チャットのメッセージ、変わったばかりの価格、共有文書のカーソル、終了したバックグラウンド処理などです。サーバーから送れる経路がなければ、クライアントはポーリングするしかありません。数秒ごとに問い合わせ、たいていは変化がないという応答を受け取ります。どの応答もヘッダー一式を含み、知らせが届くまで最大で問い合わせ1回分の間隔だけ待ちます。開いたWebSocket接続は、両方の負担を取り除きます。サーバーは次の問い合わせを待たず、何かが起きたときに送れます。各メッセージに必要なのは、HTTPヘッダー一式ではなく、フレーム用の数バイトだけです。

その代償は、状態を持つことです。HTTPリクエストはロードバランサーの背後のどのサーバーに届いても構いませんが、WebSocket接続は、存在する間ずっと1つのプロセスに属します。そのプロセスは、開いている接続ごとにメモリーを確保し、デプロイや再起動で全接続が一度に切れます。途中のすべてのプロキシとロードバランサーも、ハンドシェイクに対応し、何分も無通信になることがある接続を維持しなければなりません。サーバーを複数動かすと、通常のHTTPでは問われない問題も生まれます。サーバーAで起きたことを、サーバーBに接続しているクライアントへ、どう届けるのでしょうか。一般的な答えは、全サーバーが購読する共通のパブリッシュ・サブスクライブ経路です。運用するシステムが、さらに1つ増えます。

したがって、WebSocketは既定の選択肢ではなく、用途に応じて選ぶ方式です。知らせを送るのがサーバーだけなら、HTML標準が定めるServer-Sent Eventsで、通常のHTTP応答を `text/event-stream` として使ってイベントを送れます。応答が途切れると、ブラウザーが自動的に再接続します。WebSocketの追加の仕組みが見合うのは、チャット、ゲーム、共同編集のように、クライアントもサーバーと同程度に送るものがある場合です。

実際の活用例

次のやり取りは、チャット接続を開きます。キーはRFC 6455に載っている例で、受け入れを示す値も、正しいサーバーならこのキーに対して必ず返す、標準に掲載された値です。

```
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=
```

サーバーの応答を終える空行の後は、この接続上にHTTPは流れません。ブラウザーは、コンストラクターを1回呼ぶだけでハンドシェイク全体を処理し、その後はイベントリスナーに引き継ぎます。

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

closeイベントは、接続がどのように終わったかを示します。コード1000は正常終了で、1001は、サーバーの停止やブラウザーでページを離れる場合など、相手が接続から離れることを意味します。1006は、closeフレームを受け取らずに

接続が切れたことを示します。どのエンドポイントも、このコードを送ってはいけません。上位のプログラムが異常終了と正常終了を区別するためだけの値です。

リクエストと応答の仕組みと、ソケットそのものを扱う仕組みの違いは、このサイトの実績にある1つのプロジェクトにはっきり表れています。APIをHuma v2へ移した際、ちょうど2つを除く全ルートをHumaのリクエスト・レスポンスモデルに移しました。WebSocketエンドポイントは、ソケット自体を必要とし、Humaのモデルには合わないため、Fiberを直接使う形で残しました。HTTPの形を基に作られたフレームワークは、リクエストと応答を記述し、検証するのには適しています。長く維持する双方向接続は、それとは別のものです。

問題が起きるところ

別のサイトのページが、訪問者のCookieで接続を開く。 ブラウザーはサーバーのホストにハンドシェイクのリクエストを送り、ブラウザーのCookie方針が認める場合、そのCookieを含めることがあります。認証付きのクロスサイトリクエストが認められる場合、Cookieで認証するサーバーが `Origin` ヘッダーを無視すると、別サイトのページが開いた接続を、訪問者本人のものとして受け入れるおそれがあります。RFC 6455は、特定のサイトだけを想定するサーバーに対し、`Origin` が想定するものか検査し、それ以外は403 Forbiddenで拒否するよう勧めています。接続を受け入れる前のハンドシェイクで行います。

アクセストークンがログに残る。 ブラウザーの `WebSocket` コンストラクターが受け取るのは、URLと、任意指定のサブプロトコル一覧だけです。そのため、スクリプトから `Authorization` ヘッダーを追加できません。手早い回避策はURLにトークンを入れることですが、リバースプロキシやアクセスログにも記録されます。RFCは認証方式を限定せず、Cookie、HTTP認証、TLS認証を選択肢として挙げています。上記の `Origin` 検査を組み合わせたCookieが使えます。また、認証済みのHTTP呼び出しでページが取得し、ハンドシェイクで1回だけ使う、有効期間の短いチケットも使えます。

再起動で全クライアントが集中する。 突然のデプロイや再起動で、全接続が一度に閉じることがあります。closeフレームを受け取らなかったクライアントは、1006を報告します。全員が直ちに再接続し、試行を続けると、復帰したサーバーは、RFCがサービス拒否攻撃に例える集中を受けます。RFC 6455は、最初の試行前にランダムな時間を待つよう求め、0秒から5秒の間を例に挙げています。さらに失敗するたびに、上限付きの二進指数バックオフなどで、待ち時間を長くするよう求めています。停止前に1001で接続を閉じるサーバーなら、少なくとも計画的な切断であることを伝えられます。

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

接続が切れても誰も気付かない。 通信のない接続では、経路がまだ存在するという手掛かりを双方とも得られません。closeフレームなしに消えた相手は、次の書き込みが失敗するまで、静かな相手と区別が付きません。そのためにpingとpongがあります。エンドポイントは、closeフレームをすでに受け取っていない限り、pingにpongで答えなければなりません。未応答のpingが複数あれば、最後のものだけに答えても構いません。RFCは、pingを接続の維持にも、相手が応答するかの確認にも使えるとしています。ブラウザーAPIからpingを送る手段はないため、タイマーはサーバー側に置きます。接続ごとに一定間隔でpingを送り、pongが返らないものを閉じます。

1つのメッセージがサーバーのメモリーを使い尽くす。 フレームヘッダーの長さフィールドは、どんなサーバーのメモリーにも収まらないほど大きなペイロードを表現できます。RFCは、悪意のある相手が、巨大な1フレームや、途切れない小さな断片の列を送るおそれを指摘しています。実装側で守らなければならないため、サーバーのライブラリーにメッセージサイズの上限を設定し、超えた相手には、メッセージが大きすぎることを表す1009で接続を閉じます。逆方向にも同じ滞留が起きます。ブラウザーでは、`bufferedAmount` が、`send()` でキューに入り、まだネットワークへ送られていないバイト数を示します。WHATWGの例は、この値がゼロのときだけ、次の更新を送ります。サーバー側も、読むのが遅いクライアントによって送信バッファーが同じように埋まるため、接続ごとに上限と、それに達したときの規則が必要です。

参考資料

- RFC 6455: WebSocketプロトコル — ハンドシェイク、フレーム、マスキング、終了コードと、それぞれのセキュリティ上の理由。
- WebSockets Standard — ブラウザーAPI。コンストラクター、`send()`、`close()`、`bufferedAmount`、closeイベント。
- RFC 9110: HTTPの意味論 — ハンドシェイクが使う `Upgrade` ヘッダーフィールドと、101 Switching Protocolsステータス。
- Server-Sent Events — 再接続を組み込んだ、HTML標準による単方向の代替手段。
- RFC 8441: HTTP/2でのWebSocketの起動 — 拡張CONNECTで、HTTP/2の多重化された1ストリーム上にWebSocketを載せます。

- RFC 9220: HTTP/3でのWebSocketの起動 — HTTP/3に対応する拡張CONNECTの仕組み。

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

「WebSocket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/websocket/