Möglichkeiten zur asynchronen Kommunikation zwischen Webbrowser und Server

Hendrik Wagner
hendrik.wagner@mni.thm.de
Technische Hochschule Mittelhessen
Gießen, Hessen, Germany

Zusammenfassung

Die Kommunikation zwischen Webbrowser und Server wird klassischerweise durch den Client (in der Regel ein Webbrowser) initiiert. Dieser fragt den Inhalt einer Website an, und kann später weitere Inhalte laden oder Daten übermitteln. Der Server kann aber keine Daten nachträglich an den Client senden, wenn dieser diese nicht aktiv anfragt. In diesem Artikel werden verschiedene Möglichkeiten zur fortlaufenden Kommunikation zwischen Client und Server vorgestellt und analysiert. Dabei werden die Vor- und Nachteile der einzelnen Methoden aufgezeigt.

Keywords: web, http, WebSockets, polling, bidirectional communication, asynchronous communication

Inhaltsverzeichnis

1 Einleitung

Gilt es, eine Website mit bidirektionaler bzw. asynchroner Kommunikation (z. B. vom Server bereitgestellten Aktualisierungen) zu realisieren, so muss eine Auswahl zwischen verschiedenen Ansätzen gewählt werden. Ein solcher Fall kann eine Chatanwendung sein, in welcher der Server von einem Client erhaltene Nachrichten an andere Clients weitersendet. Ein weiteres Beispiel wäre eine rechenintensive Anwendung, in der eine Berechnung nach einiger Zeit erfolgt und der Server das Ergebnis an den Client melden soll. Da der Server unter HTTP nicht ohne weiteres eine Nachricht an einen Client senden kann, muss dieser die Nachricht anfordern – diese simple Art des Nachrichtenempfangs nennt man *Polling*, also die wiederholte Abfrage nach neuen Nachrichten. Neben diesem Ansatz gibt es *WebSockets*, welche eine TCP/UDP-ähnliche Kommunikation im Web ermöglichen.

2 Vorstellung der Kommunikationsvarianten

Zunächst werden die Kommunikationsvarianten in ihrer Funktionsweise erklärt und vorgestellt. In späteren Kapiteln wird ebenfalls auf deren Implementierung sowie auf Frameworks eingegangen.

2.1 Klassisches Polling

Unter dem 1990 eingeführten [1, Abs. 1.2] HTTP-Protokoll gibt es eine Reihe von Anfragen, die der Client an den Server senden kann [2]. Das Protokoll definiert dabei explizit einen Client (in unserem Fall der Browser), welcher Anfragen an den Server (Bereitstellender der Webinhalte) versendet. Dieser wartet auf Anfragen und beantwortet diese [1, Abs. 1.3].

Für das Vorhaben sind GET-Requests besonders relevant, also klassische Abfragen von Inhalten mittels HTTP. Polling, in diesem Anwendungsfall wohl am besten übersetzt mit *zyklische Absuche* oder *Sendeaufruf*, beschreibt in der Webentwicklung das regelmäßige Abrufen (in einem Intervall Δ) von neuen Inhalten. Gibt es Nachrichten, die der Server bereitstellen möchte, beantwortet dieser die Anfrage mit den neuen Inhalten – andernfalls wird der Antwortkörper leer sein.

Das Hauptproblem mit Polling ist der entstehende Header Overhead, der für den gesamten Zeitraum in den regelmäßigen Abfragen besteht. Dadurch entsteht eine Netzwerkbelastung, welche keine tatsächlichen Informationen übermittelt.

In der in späteren Abschnitten vorgestellten Chatanwendung handelt es sich so zum Beispiel um 156 Bytes, die sekündlich vom Server übermittelt werden, aber lediglich ein leeres JSON-Array enthalten.

2.1.1 Spezifikation. Polling ist in der Spezifikation von HTTP nicht definiert, es handelt sich hierbei um eine Implementierung des Clients und des Servers. Meist wird Polling durch JavaScript-Code realisiert, welcher in einem bestimmten Intervall (z. B. alle 5 Sekunden) eine Anfrage an den Server sendet. Diese Anfrage ist ein GET-Request, welcher die URL der Anwendung enthält. Wie der Server auf diese Anfrage antwortet, ist Implementierungsabhängig: So könnte er zum Beispiel den gesamten abgefragten Inhalt bei jeder Anfrage übermitteln, oder nur Änderungen, die dieser Client noch nicht übermittelt bekommen hat. Eine solche Optimierung setzt allerdings voraus, dass der Server in der Lage ist, zu identifizieren, welche Änderungen der Client bereits erhalten hat.

2.2 Long Polling

Eine Optimierung des Polling-Konzepts ist das sog. *Long Polling*. Dabei wird die Antwort auf eine HTTP-Anfrage zurückgehalten, bis der Server eine Nachricht versenden will [3].

Probleme von Long Polling sind unter anderem der (im Vergleich zu klassischem Polling reduzierter, aber weiterhin präsenter) Header Overhead, mögliche Timeouts und Ressourcen, die in Vorbereitung auf eine eingehende Nachricht vom Betriebssystem zu Verfügung gestellt werden. [4, Abs. 2.2].

2.3 Streaming

Eine weitere Optimierung des Polling-Konzepts ist das sog. *Streaming*, welches die HTTP Transferkodierung Chunking (vgl. [1, Abs. 7.1]) verwendet, also dem Aufspalten der Antwort in mehrere Packets, in welchen dann einzelne Nachrichten versendet werden. So kann die Anzahl an Anfragen ausgehend vom Client an den Server auf eine reduziert werden – der Server terminiert nie die Antwort auf die erste Anfrage [4, Abs. 3].

Mit Streaming werden einige Nachteile von Polling beglichen, andere – insb. mögliche Timeouts und unnötig reservierte Ressourcen – bleiben bestehen. Zusätzlich besteht das Risiko, dass diese Form von Kommunikation nicht auf allen Systemen funktioniert – Proxys können Pakete bündeln und erst verzögert weiterleiten, wodurch Pakete ggf. nicht zeitgetreu, gebündelt oder aufgespalten ankommen [4, Abs. 3.2].

2.4 WebSockets

WebSockets sind ein vom IETF[1] entwickeltes Protokoll, welches 2011 als Standard veröffentlicht wurde [5]. Grund für dessen Entstehung ist unter anderem die Erkenntnis, dass der Versuch HTTP zu verwenden, um bidirektionale Kommunikation zu ermöglichen, vermeidbare Komplexität und Ineffizienz mit sich bringt [6, S. 137f]. Es ist anzumerken, dass die vorgestellten Polling-Varianten das HTTP-Protokoll effektiv missbrauchen, um serverseitige Kommunikation zu ermöglichen:

- Polling beinhaltet das Versenden von redundanten Anfragen, welche ohne Inhalt beantwortet werden,
- Long Polling simuliert eine Verbindung mit (sehr) hoher Latenz um eine Antwort herauszuzögern und
- Streaming verwendet HTTP Chunking, um innerhalb einer Response alleinstehende Nachrichten zu senden.

Diesen Missbrauch haben WebSockets nicht – es handelt sich hierbei um ein komplett neues Protokoll, welches diese Problemstellung direkt adressiert (vgl. [5, Abs. 1.1]), indem es eine TCP-Verbindung zwischen Client und Server aufbaut und unterstützt. Genauer baut WebSockets einen Tunnel zwischen TCP und IP auf, sodass darauffolgend TCP-Kommunikation Systemübergreifend erfolgen kann (vgl. [5, Abs. 1.5]). HTTP wird dann lediglich für die initialen Handshakes verwendet – danach nicht mehr.

2.4.1 Spezifikation. Um eine Verbindung über einen WebSocker aufzubauen, wird zunächst vom Client aus eine HTTP-Anfrage gesendet. Gemäß der Spezifikation muss es sich hierbei um eine GET-Request an den Pfad, auf dem der WebSocket hinterlegt ist, handeln.

Wäre die WebSocket URI `ws://example.org/chat`, so wäre die erste Zeile der HTTP-Anfrage `GET /chat HTTP/1.1` [5, Abs. 4.1]. Zusätzlich müssen in der Anfrage unter anderem der Host (hier `Host: example.org`) und die Intention (Felder `Connection: Upgrade` und `Upgrade: websocket`) als Headerfelder übermittelt werden.

Der Server antwortet auf diese Anfrage mit einer HTTP-Response, welche entweder den Statuscode 101 *Switching Protocols* oder einen Fehlercode enthält. Zudem meldet er bei erfolgreichem Protokollwechsel die vom Client angegebene Intention zurück.

Wurde die Verbindung vom Server akzeptiert, können nun Client und Server direkt miteinander kommunizieren. Für die Datenübermittlung sieht das WebSocket Protokoll eine auch in anderen Protokollen ähnlich vertretenes Frame-System vor, also dem Versenden von Daten in Form von Frames, welche jeweils einen Header und eine Payload enthalten. Header enthalten u. A. den Opcode[2] und die Payload-Länge (vgl. [5, Abs. 5.2]).

[1] Internet Engineering Task Force.

[2] Opcodes (*Operation Codes*, dt. *Befehlscode*) können bei WebSockets Frames mit Text- oder Binärdaten ankündigen, eine vorherige Frame fortsetzen, die Verbindung schließen, oder einen Ping/Pong markieren.

3 Beispielhafte Implementierungen

Um die Unterschiede zwischen den Kommunikationsvarianten ersichtlich zu machen, werden diese für eine Chat-Applikation implementiert. Dabei wird jeweils auf die Client- und Serverseitige Implementierung eingegangen.

Die Realisierung erfolgt in TypeScript, wobei die Serverseitige Implementierung auf Node.js basiert. Der Client benutzt das Vue.js-Framework mit Vuetify, um die Darstellung zu vereinfachen. Implementierungen in anderen Frameworks erfolgen analog.

3.1 Implementierung von Polling

Für die Implementierung von Polling verwaltet der Server eine Liste der letzten 100 Nachrichten. Neue von einem Client übermittelte Nachrichten werden mit Autor, Zeitstempel und Text in diese Liste eingefügt. Fragt ein Client die Liste ab, so erhält er die letzten 100 Nachrichten zurück. Der Client übermittelt dabei den Zeitstempel der letzten Nachricht, die er erhalten hat. So kann der Server nur die Nachrichten zurückgeben, die nach diesem Zeitstempel eingegangen sind.

Es wird nicht auf die Darstellung der Nachrichten oder des Chats eingegangen, da dies nicht relevant für die Implementierung ist. Entsprechende Codeabschnitte sind daher ausgelassen. Die vollständige Implementierung ist in der Anlage ?? zu finden.

Listing 1. Beispielhafte Nachricht

```
{
  content: "Hello World!",
  sender: "Alice",
  timestamp: "2022-12-04T19:29:13.455Z"
}
```

Listing 2. Polling Server

```
import * as http from 'http';

type Message = {
  sender: string;
  content: string;
  timestamp: string;
};

const lastMessages: Message[] = [];

const pollingServer = http.createServer((req, res) => {
  if (req.url === '/polling') {
    let body = '';
    req.on('data', (chunk) => {
      body += chunk; // convert Buffer to string
    });
    req.on('end', () => {
      const timestamp: string = body;
      const messages = body ? lastMessages.filter(
        (message) => Date.parse(message.timestamp) > Date.parse(timestamp)
      ) : lastMessages; // return all messages if no timestamp was sent
      res.setHeader('Content-Type', 'application/json');
      if(messages.length > 0) {
        res.end(JSON.stringify(messages)); // return messages as json
      } else {
        res.end(); // no messages to return
      }
    });
  } else if (req.url === '/polling/send') {
    let body = '';
    req.on('data', (chunk) => {
      body += chunk;
    });
    req.on('end', () => {
      const message: Message = JSON.parse(body);
      console.log('Received http message => ' + message);
      lastMessages.push(message);
      if (lastMessages.length > 100) {
        lastMessages.shift();
      }
      res.end(); // send 200 OK
    });
  } else {
    res.end(); // ignore other requests
  }
});

pollingServer.listen(8082, () =>
  console.log(`Polling server started on port 8082`)
);
```

Listing 3. Polling Client

```
import {ref} from 'vue';

type Message = {
  sender: string;
  content: string;
  timestamp: string;
};

const httpMessages = ref<Message[]>([]);
const message = ref('');
const username = ref('');
let lastMessageTimestamp = new Date().toISOString();

const sendMessage = async () => {
  const msg = {
    sender: username.value,
    content: message.value,
    timestamp: new Date().toISOString(),
  };

```

```
21    await fetch('http://localhost:8082/polling/send'
        , {
22      method: 'POST',
23      headers: {
24        'Content-Type': 'application/json',
25      },
26      body: JSON.stringify(msg),
27    });
28
29    message.value = '';
30  };
31
32  // fetch messages from server every 2 seconds
33  const fetchMessages = async () => {
34    if(httpError.value) { // slow down polling if
        there is an error
35      clearInterval(interval);
36      interval = setInterval(fetchMessages, timeout
        *= 2);
37    }
38    const nextTimestamp = new Date().toISOString();
39    const response = await fetch('http://localhost
        :8082/polling', {
40      method: 'POST',
41      headers: {
42        'Content-Type': 'application/json',
43      },
44      body: lastMessageTimestamp,
45    });
46    if (response.ok) {
47      timeout = 2000;
48      if(httpError.value) {
49        clearInterval(interval);
50        interval = setInterval(fetchMessages,
        timeout);
51        httpError.value = false;
52      }
53      const messages = await response.json();
54      httpMessages.value = httpMessages.value.concat
        (messages);
55      lastMessageTimestamp = nextTimestamp;
56      await nextTick()
57      window.scrollTo({
58        top: document.body.scrollHeight,
59        behavior: 'smooth',
60      });
61    } else {
62      httpError.value = true;
63      httpErrorMessage.value = await response.text()
        ;
64    }
65  };
66
67  let timeout = 2000;
68  let interval = setInterval(fetchMessages, timeout)
        ;
```

3.2 Implementierung von Long Polling

...

3.3 Implementierung von WebSockets

3.3.1 WebSocket Client.

3.3.2 WebSocket Server.

4 Vergleiche zwischen Kommunikationsvarianten

4.1 Analyse der Implementierungen

Vor- und Nachteile Performance?

... Die Latenzen überschneiden sich mit Messungen getätigt im Rahmen eines Papers von Pimentel und Nickerson [7].

4.2 Generelle Feststellungen

Notabel ist auch die Reduktion in Overhead, welcher bei klassischem Polling durch HTTP-Header entsteht. Üblicherweise sind HTTP-Header zwischen 200 und 2.000 Bytes groß [8]. Dies kann einen ernsthaften Effekt auf die Netzwerkbelastung haben, besonders wenn die Polling- oder Nachrichtenfrequenz hoch ist. So würden 1.000 Nutzer, die sekündlich Nachrichten anfragen, allein durch die Header etwa 6 Mbps an Netzwerkdurchsatz verursachen [9]. Dieser Netzwerkdurchsatz entfällt durch Websockets.

5 Vorstellung von Frameworks

Es gibt Frameworks, die das Implementieren von asynchroner Kommunikation erleichtern.

5.1 Socket.IO

Was ist das? Wie ist es besser oder schlechter für die Implementierung als Chat-App? Performance?

5.2 Faye

6 Fazit

Literatur

[1] R. T. Fielding, M. Nottingham, and J. Reschke, "HTTP Semantics," Internet Engineering Task Force, Request for Comments RFC 9110, Jun. 2022, num Pages: 194. [Online]. Available: https://datatracker.ietf.org/doc/rfc9110

[2] "HTTP request methods." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

[3] "Long polling," Dec. 2021. [Online]. Available: https://javascript.info/long-polling

[4] P. Saint-Andre, S. Loreto, S. Salsano, and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP," Internet Engineering Task Force, Request for Comments RFC 6202, Apr. 2011, num Pages: 19. [Online]. Available: https://datatracker.ietf.org/doc/rfc6202

[5] A. Melnikov and I. Fette, "The WebSocket Protocol," Internet Engineering Task Force, Request for Comments RFC 6455, Dec. 2011, num Pages: 71. [Online]. Available: https://datatracker.ietf.org/doc/rfc6455

[6] P. Lubbers, B. Albers, and F. Salim, *Pro HTML5 Programming*, 1st ed. Apress Berkeley, CA, 2010. [Online]. Available: https://link.springer.com/book/10.1007/978-1-4302-2791-5

[7] V. Pimentel and B. G. Nickerson, "Communicating and Displaying Real-Time Data with WebSocket," *IEEE Internet Computing*, vol. 16, no. 4, pp. 45–53, Jul. 2012, conference Name: IEEE Internet Computing. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6197172

[8] "SPDY: An experimental protocol for a faster web." [Online]. Available: https://www.chromium.org/spdy/spdy-whitepaper/

[9] P. Lubbers and F. Greco, "HTML5 WebSocket - A Quantum Leap in Scalability for the Web." [Online]. Available: https://web.archive.org/web/20210422023846/http://websocket.org/quantum.html