Protokoly

SSPŠ
MVOP4 WBI
2025 / 2026
Ing. Matěj Cajthaml

HTTP

- Bezestavový protokol
- Požadavek vs odpověď
- Pro výměnu dat ideální
- Problém nastává s oboustrannou komunikací
- Co když chce něco poslat server a klient o tom neví?
- Několik možností, které si ukážeme
- Spíše imitují oboustrannou komunikaci
- A jiný protokol

Polling

- Implementačně pouze na klientovi

- Klient se ptá na určitou adresu opakovaně
- Třeba 1x za 5s

- Server odpovídá zcela obyčejně
- Pokud má data, zašle je
- Pokud ne, vrátí chybu

- Jednoduché na implementaci
- Hrozné na zátěž
- Při 1000 klientech se každý ptá 1x za 5s, to je přes 700k požadavků za hodinu
- A často se ptá na něco, co neexistuje
- Přenášejí se zbytečná data

Polling

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <p>Data:</p>
    <pre id="data"></pre>

    <script>
        const dataElement = document.getElementById('data');

        const interval = setInterval(() => {
```

Long polling

- Navazuje na polling
- Implementačně oproti pollingu změny pouze na serveru

- Klient se ptá na určitou adresu opakovaně
- Nyní ale server čeká s odpovědí
- Má tzv. timeout, po kterém vyhodí chybu, a nebo pokud přišla hodnota, vrátí ji rovnou
- Po obdržení odpovědi klient znovu pošle požadavek

- Timeout většinou třeba 30s
- Problém, když je velký, tak klient, proxy nebo něco mezi tím "ukončí" požadavek
- Stále se přenáší zbytečné data
- Stále velmi hrozné na výkon

Long polling

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <p>Data:</p>
    <pre id="data"></pre>

    <script>
        const dataElement = document.getElementById('data');

        const wait = (ms) => new Promise(resolve => setTimeout(resolve, ms));
```

Server sent events

- Zcela otáčí problém
- Místo volání stejných endpointů budeme pořád připojený na jeden
- Propojení ale jasně označíme jako trvalé

- Sever při nové události (informaci, datech) pošle část dat v těle klientovi
- Klient zpracuje novou část zprávy

- Stejný problém s timeoutem - kdokoliv se může rozhodnout o uzavření spojení
- Stejně je nutné poslat nějakou zprávu např. 1x za 60s, aby všichni věděli, že data pořád pokračují

- Speciální API na klientovi
- Lze se po odpojení navázat jednoduše a získat zprávy od poslední

SSE / server

```
app.get('/events', (req, res) => {
  res.setHeader('Content-Type', 'text/event-stream');
  res.setHeader('Cache-Control', 'no-cache');
  res.setHeader('Connection', 'keep-alive');

  setInterval(() => {
    res.write(`data: ${JSON.stringify({ time: new Date() })}\n\n`);
  }, 1000);

  res.on('close', () => {
    clearInterval(intervalId);
    res.end();
  });
});
```

SSE / klient

```
const eventSource = new EventSource('/events');
eventSource.addEventListener('message', (event) => {
  const data = JSON.parse(event.data);
  console.log('Nová data ze serveru:', data);
});
```

Problémy prohlížečů

- Prohlížeče omezují stránky
- V jeden moment omezuje počet různých propojení

- Moderní prohlížeče mají okolo 5-8 tunelů
- Tunel = jedno možné připojení na další internet

- Pokud máme více, musíme alokovat konkrétní volání a samostatně je omezovat
- Aby se nám nezaplnili všechny

Komunikace mezi servery

- Komunikace mezi jednotlivými servery často probíhá stejně
- Tj. HTTP

- Někdy komunikují pomocí databází
- Velmi logicky dokumentové a relační

- Méně logické jsou databáze typu key-value
- Dovoluji často zamykat různé akce, ukládat cache a podobně
- Nejčastěji redis

- Na pokročilé (a složitější) komunikace existují jiné možnosti
- Např. gRPC, což je formát a protokol na TCP
- Používá protobuf k definici jednotlivýchzpráv a služeb
- Dovoluje vícestrannou komunikace

Nevěřte uživateli

- Jak víme z backendu
- Nikdy nevěříme datům od uživatele
- Vždy kontrolujeme validaci
- Validujeme i délku hodnot
- Ve všech protokolech
- Dokumentace API
- Celkové principy bezpečnosti

WebSocket

Aneb jak používat něco vymyšleného pro vícestranné komunikace

WebSockets

- Dovoluje oboustrannou komunikaci mezi klientem a serverem
- Podpora skoro každého prohlížeče

- TCP vs. UDP
- Trvalé propojení, stejně používá ping

- Chat, notifikace, online hry, ...

- Pro připojení potřebuje HTTP
- Hlavička Upgrade a HTTP status kód 101

- API v DOM
- Na serveru většinou pomocí knihovny

- Protokol ws:// nebo wss://

OTÁZKA

Na co dalšího by se mohli WebSockety používat kromě chatu, her a notifikací?

Použití

- Vytvoříme klienta
- Ten se připojí
- Navážeme na klienta události
- - po připojení
- - po obdržení zprávy
- - při chybě, ...
- Na klientovi můžeme poslat zprávu na server
- WS fungují buď textově nebo binárně
- V základu prostě posíláme texty naproti sobě
- Problém je rozeznání jednotlivých typů zpráv
- Server má událost připojení klienta

Klient

```
const socket = new WebSocket('wss://example.com/socket');
socket.addEventListener('open', (event) => {
  console.log('Připojeno k WebSocket serveru');
  socket.send('Ahoj server!');
});

socket.addEventListener('message', (event) => {
  console.log('Zpráva od serveru:', event.data);
});

socket.addEventListener('close', (event) => {
  console.log('Spojení uzavřeno');
});

socket.addEventListener('error', (error) => {
```

Server

- NPM knihovna ws
- Stejně potřebujeme webový server
- Např. express nebo http

Server

```
1  import http from 'http';
2  import WebSocket, { WebSocketServer } from 'ws';
3  const server = http.createServer();
4
5  const wss = new WebSocketServer({ server });
6
7  wss.on('connection', (ws) => {
8    console.log('Nový klient připojen');
9
10   ws.on('message', (message) => {
11     console.log('Zpráva od klienta:', message);
12     ws.send(`Server obdržel: ${message}`);
13   });
14
15   ws.on('close', () => {
```

Nastavení komunikace

- Velký problém, řešen různě dle potřeb
- Nejčastěji hry - posílají si packety, jasný identifikátor a data
- Můžeme:
- mít id typu zprávy, např. 1, 332 nebo klíče, např. "post_chat_message"
- Pro daný typ poté můžeme očekávat nějaká data
- Validace např. pomocí zod, Joi a dalších
- Binárně můžeme data hodně "nasoukat" do menších čísel

OTÁZKA

Jak uděláme, že pošleme zprávu jen některým klientům? A co všem?

Nástavby

- Knihovna ws je velmi jednoduchá
- Pokud chceme něco implementovat, musíme sami
- Ale v rámci komunikací existuje spousta již existujících problémů
- Např. problém místností
- A DOM ws vůbec neřeší reconnection zařízení
- A co když zařízení ws nepodporuje?
- Co když bude více serverů?

Nástavby

- Socket.IO
- SockJS
- SignalR
- Primus

OTÁZKA

Jak vyřešíme v rámci WS autentizaci a autorizaci?

PRÁCE

Chatovací aplikace

Asynchronní protokol

- Data se na server (klienta) odešlou
- Až dojdou, server reaguje
- My nevíme kdy

- Často potřebujeme čekat "na schválení" či podobné
- Např. chat_message_new
- Server by mohl odpovědět chat_message_processed

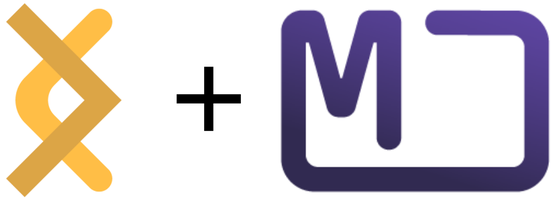
- Co ale když během toho pošleme další? Pro jakou z nich je tento processed?
- Solution => ke každé zprávě dávat klientově unikátní id
- Které server "vrací s odpovědí"

PRÁCE

IO hra za pomocí threejs/canvasu

Děkuji za pozornost

matej.cajthaml@ssps.cz
ssps.cajthaml.eu



Vytvořeno pomocí aplikace Materialist