Възможности за създаване на приложения, използващи WebSocket

Павел Петров

Abstract: Possibilities for Creating Applications Using WebSocket: *The paper address issues associated with the use of DOM object and protocol WebSocket. We developed example application to illustrate the usage of WebSocket in a simple real time web application. Node.js and the program library socket.io have been used.*

Key words: *WebSocket, web applications, Node.js, socket.io.*

ВЪВЕДЕНИЕ

При класическите заявки по протокол HTTP 1.0 [1], уебклиентът изпраща заявка до уебсървъра, получава отговор и TCP/IP връзката между двете страни се прекъсва. Версия 1.1 [3] на HTTP даде възможност в рамките на една TCP/IP връзка, уебклиентът синхронно да изпраща на уебсървъра множество заявки, като по този начин се избягват разходите от време, свързани с отваряне и затваряне на нова мрежова връзка при всяка заявка. В резултат, мрежовата връзка между клиент и сървър остава отворена много по-дълго време спрямо връзките по протокол HTTP 1.0. Това наложи еволюцията на уебсървърите да се развие в посока поддържане на по-голям брой едновременно отворени мрежови връзки, с цел да се поддържат повече уебклиенти едновременно и да няма ситуации на отказ от обслужване при голям брой HTTP заявки за единица време.

В резултат се появиха и наложиха уебсървъри, способни да поддържат десетки хиляди едновременно отворени връзки при сравнително малък обем заета оперативна памет, като например уебсървърите nginx и lighttpd. За сравнение, при най-популярните уебсървъри - Apache и IIS обемът памет заеман за всяка мрежова връзка е много по-голям, което налага и по-големи изисквания към хардуера по отношение на инсталираната оперативна памет.

В допълнение на това, тенденцията за все по-дълго поддържане на отворена мрежова връзка се засили и от налагането на техниките за създаване на приложения в реално време, базирани на особености на HTTP протоколите версии 1.0 и 1.1 - "издърпване" - "pull", "дълго запитване" - "long polling" и "избутване" - "push". Тези техники използват средства, които по принцип отговарят на стандартите, но не са изрично предвидени в тях [5]. Затова през последните години за създаването на приложения в реално време се предпочитаха други подходи, като Java аплети, Flash приложения или Silverlight приложения. Използването на подобни приложения затруднява потребителите, тъй като изисква инсталирането на допълнителни модули (плъгини) към браузърите и това намалява тяхната приложимост.

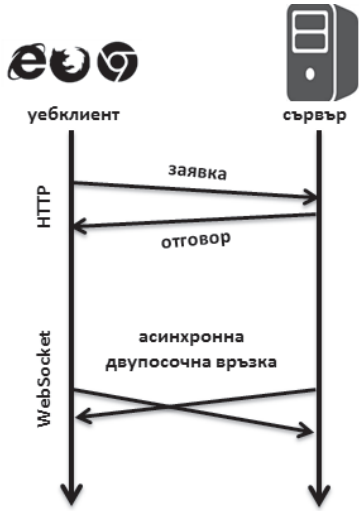
За решаване на тези проблеми, едно от направленията за развитие на протоколите HTTP и документния обектен модел на уебстраниците (DOM) беше добавянето на възможност мрежовата връзка между клиента и сървъра да остава отворена неограничен период от време. Целта на настоящата публикация е да разгледа в приложен аспект новите възможности за поддържане на мрежовата връзка между уебклиента и уебсървъра отворена дълги периоди от време, което дава възможност за създаване на приложения работещи в реално време, в които множество потребители взаимодействат едновременно.

ИЗПОЛЗВАНЕ НА DOM ОБЕКТ И ПРОТОКОЛ WEBSOCKET

За добавяне в документния обектен модел на уебстраниците (DOM) през последните години беше предложен и обектът WebSocket, стандартизиран от W3 Consortium [4]. Също така, специално за WebSocket има и разработен протокол от IETF със същото име [2], поддържащ се от най-популярните браузъри към

настоящия момент. Това осигурява широкото разпространение на WebSocket и той е особено перспективен, тъй като решава всички въпроси, свързани с двупосочната асинхронна връзка между клиента и сървъра. Той опростява значително създаването на уебприложения в реално време.

Редица големи доставчици на облачни услуги поддържат WebSocket, което дава възможност за прилагането му и без пълен административен контрол върху хардуерния сървър. Възможността по ефективен начин (по отношение на обем трафик и натоварване на сървъра) да се предават данни на малки части, с висока честота и асинхронно, създава предпоставки за разработването на приложения, които работят също толкова бързо и интерактивно, както и традиционните десктоп приложения. Подходите, базирани изцяло на особеностите на протокола HTTP не са подходящи за тези цели, тъй като при предаването на заявката и на отговора, към данните се добавят и заглавни части, които съдържат различна служебна информация, като например за типа на данните, кодировка, бисквитки, идентификация на клиента или сървъра, времето и датата и др. Тези служебни данни варират значително по обем и съпоставени с размера на една уебстраница или изображение са сравнително малки, но когато данните се изпращат на малки порции и през малък интервал от време, обемъта на служебните данни започва да става съпоставим и може в пъти да надхвърли обема на "полезните" данни, които всъщност се предават.



Фигура 1. Времедиаграма на комуникацията между уебклиента и уебсървъра при протоколи HTTP и WebSocket

За разлика от сесийния синхронен характер на HTTP (заявка - отговор), при WebSocket след като се осъществи връзката, двете страни са равноправни и използват постоянна мрежова връзка (Фиг.1). Обменът на данни е асинхронен и двете страни на мрежовата връзка не е нужно да се изчакват, за да изпращат данните и това може да става едновременно и двупосочно. Използването на събития от страна на клиента дава възможност без да се прекъсва изпълнението на програмен код, след пристигане на данни от сървъра те да бъдат обработени.

Едно от основните предимства на WebSocket е опростеният приложен програмен интерфейс, който в сравнение с голямото количество библиотеки,

предназначени да улеснят създаването на уебприложения в реално време по техниките "издърпване" - "pull", "дълго запитване" - "long polling" и "избутване" - "push", значително улеснява разработката.

Процедурите от страна на клиента за работа с обекта WebSocket са следните:

- създаване на нов обект с URL с използване на протокол ws или wss (Secure WebSocket):

```
ws = new WebSocket("ws://example.com/test");
```

- функция, извиквана при осъществяване на мрежовата връзка:

```
ws.onopen = function() { alert("OPEN..."); };
```

- функция, извиквана при прекъсване на мрежовата връзка:

```
ws.onclose = function() { alert("CLOSE..."); };
```

- функция, която асинхронно се извиква при получаване на данни от сървъра - те се намират в свойството .data на параметъра:

```
ws.onmessage = function(e) { alert(e.data); };
```

- асинхронно изпращане на данни към сървъра:

```
ws.send("Test");
```

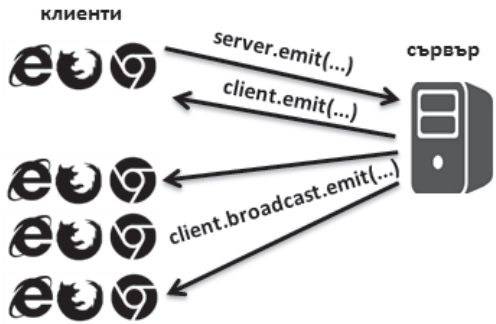
Работата от страна на уебсървъра е по-сложна и изисква повече познания за стандартите на WebSocket. Ще разгледаме пример, разработен на Node.js с използване на WebSocket чрез използването на програмната библиотека socket.io. Примерът представлява едно демонстрационно приложение за онлайн комуникация - за едновременно рисуване от няколко потребителя в една уебстраница.

Програмният код на уебсървърната част на Node.js е във файл с име draw.js:

```
var fs = require('fs');
var txt = fs.readFileSync('draw.html');
var br = 0;

var server = require('http').createServer(function (req, res) {
   res.writeHead(200);
   res.end(txt);
});
server.listen(1234);

var io = require('socket.io')(server);
io.on('connection', function (client) {
  br++;
  console.log("Total connections: " + br);

  client.on('draw', function(data) {
        client.emit('draw', data);
        client.broadcast.emit('draw', data);
  });

  client.on('disconnect', function(){
        br--;
        console.log("Total connections: " + br);
  });
});
```

Както се вижда, програмният код на сървърната част е много кратък, но това е за сметка на клиентската част, където е изнесена по-голяма част от логиката на това разпределено уебприложение. Подобен подход има смисъл, тъй като обикновено машината на уебсървъра е по-натоварена от машината на уебклиента и прехвърлянето на по-голямата част от изчисленията към клиента дава възможност уебсървъра да се разтовари и да обслужва повече уебклиенти.

Основен момент в програмния код е използването на два метода - с метода .emit() се изпраща съобщение само до клиента, а с метода .broadcast.emit() се изпраща съобщение до всички клиенти, свързали се със сървъра и така на практика се реализира "избутване" - "push" на данни от сървъра към клиентите (Фиг.2).



Фигура 2. Схема на комуникацията между уебклиентите и уебсървъра при WebSocket

Програмният код на уебстраницата за приложението за едновременно рисуване е във файл draw.html, разположена в същата директория, в която е и файлът draw.js. Програмата draw.js всъщност е едновременно и уебсървър, който ще върне като отговор draw.html, и /socket.io/socket.io.js, и WebSocket сървър.

```
<!doctype html>
<html>
<head>
  <title>Draw</title>
</head>

<body>
  <button type="button" onclick="color = 'red';">Red</button>
  <button type="button" onclick="color = 'green';">Green</button>
  <button type="button" onclick="color = 'blue';">Blue</button>
  <input type="color" onchange="color = this.value;" >
  0<input type="range" min="1" max="30" value="5" onchange="lw = this.value;">30

   <canvas id="myCanvas" width="800" height="600" style="border: 1px solid #000"></canvas>
   <script>
        var x, y, isMouseDown = false, color = 'black', lw = 5;

        var canvas = document.getElementById('myCanvas');
        var context = canvas.getContext('2d');

        canvas.addEventListener('mousedown', function(evt) {
              isMouseDown = true;
        }, false);
        canvas.addEventListener('mouseup', function(evt) {
              isMouseDown = false;
        }, false);
```

```
        canvas.addEventListener('mousemove', function(evt) {
                x_old = x;
                y_old = y;
                x = evt.clientX - canvas.getBoundingClientRect().left;
                y = evt.clientY - canvas.getBoundingClientRect().top;
                if(isMouseDown) {
                        server.emit('draw', [x_old, y_old, x, y, color, lw]);
                }
    }, false);
  </script>

  <script src="/socket.io/socket.io.js"></script>
  <script>
        var server = io();
        server.on('draw', function (data) {
                context.beginPath();
                context.moveTo(data[0], data[1]);
                context.lineTo(data[2], data[3]);
                context.strokeStyle = data[4];
                context.lineWidth = data[5];
                context.stroke();
                context.closePath();
        });
  </script>
</body>
</html>
```

ЗАКЛЮЧЕНИЕ

Уебтехнологиите се развиват в направление към облекчаване изграждането на уебприложения, работещи в реално време. Подобни приложения се различават съществено от класическите разбирания за уебсайт, доближавайки се като функционалност до десктоп приложенията.

ЛИТЕРАТУРА

[1] Berners-Lee T., R. Fielding R., H. Frystyk RFC1945 Hypertext Transfer Protocol - HTTP/1.0, 1996, <http://www.ietf.org/rfc/rfc1945.txt>

[2] Fette I., A. Melnikov, RFC6455 The WebSocket Protocol, 2011, <http://tools.ietf.org/rfc/rfc6455.txt>

[3] Fielding R., J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee RFC2616 Hypertext Transfer Protocol - HTTP/1.1, 1999, <http://www.rfc-editor.org/rfc/rfc2616.txt>

[4] Hickson, I. The WebSocket API, W3C Candidate Recommendation, 2012, <http://www.w3.org/TR/websockets/>

[5] Loreto, S., Saint-Andre, P., Salsano, S., G. Wilkins, RFC 6202: "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", 2011, <http://www.ietf.org/rfc/rfc6202.txt>

За контакти:

Доц. д-р Павел Петров, Катедра *Информатика*, Икономически университет - Варна, тел.: 0882-164-704, e-mail: petrov@ue-varna.bg