treinaweb.com.br

Aplicações em tempo real com PHP usando WebSockets - Blog da TreinaWeb

Kennedy Tedesco

9-12 minutos

Neste artigo veremos uma introdução a WebSockets com a criação de um servidor em PHP e usando o navegador do usuário como cliente.

Antes, entretanto, recomendo que você assista o vídeo O que são WebSockets? gravado pelo Akira Hanashiro (aqui da TreinaWeb) que explica de uma forma bem sucinta.

E, se você tiver interesse, também recomendo a leitura do artigo Uma introduçaõ a TCP, UDP e Sockets para se ter a base do que é um socket na arquitetura **TCP**/**IP**.

Ok, mas o que é um WebSocket?

WebSocket é um protocolo que permite a criação de um canal de comunicação cliente-servidor com transmissão bidirecional onde ambos os lados (cliente e servidor) podem transmitir dados simultaneamente. WebSocket veio para suprir as deficiências do protocolo Http para esse objetivo. O protocolo Http, que por sinal, é unidirecional (a transmissão ocorre só de uma ponta para a outra), onde o cliente envia a requisição e o servidor retorna a resposta, finalizando ali a conexão. Ou seja, se o cliente envia 5 requisições Http para o servidor, 5 conexões TCP independentes são abertas, enquanto que com WebSocket uma única conexão TCP é aberta e ela fica disponível para troca de dados a qualquer momento (uma conexão

persistente, até que um dos lados decida fechá-la).

É muito importante pontuar, também, que Socket e WebSocket são coisas diferentes. Um WebSocket é um protocolo que roda em cima de sockets TCP, enquanto que um socket é uma abstração, uma porta lógica de comunicação entre duas pontas numa rede.

Benefícios de usar WebSocket em detrimento a Http

Se existe a necessidade de uma conexão permanecer aberta por um longo tempo para uma troca de dados constante, WebSocket é uma ótima escolha. Uma conexão Http é relativamente pesada, ela transmite não só os dados, mas também cabeçalhos. Além disso, possui um curto tempo de vida e não mantém estado (stateless).

Já uma conexão WebSocket, depois do handshake entre o cliente e o servidor, ela permanece aberta até que uma das partes decida fechá-la. E foco dela é na transmissão dos dados, cabeçalhos não são transmitidos pra lá e pra cá.

Então, em resumo, os benefícios em relação ao Http são: baixa latência, conexão persistente e full-duplex (transmissão bidirecional).

Principais casos de uso para WebSockets

Home Brokers (onde cotações são atualizadas a todo instante), feeds de redes sociais, aplicativos de bate-papo, ferramentas de edição colaborativa, jogos multi-player etc.

Criando o primeiro servidor

Neste artigo usaremos a library Ratchet que nos permite

criar um servidor WebSocket assíncrono, e para isso ela utiliza o *event loop* do ReactPHP. Outra opção seria utilizarmos o servidor de WebSocket do Swoole. Mas o bom de usar o ReactPHP é que não precisamos instalar nenhuma extensão específica na nossa instalação do PHP.

Crie uma pasta chamada `websocket-demo` e dentro dela crie o arquivo `composer.json`:

```
{
    "require": {
        "cboden/ratchet": "^0.4.1"
    }
}
```

Instale as dependências:

```
$ composer install
```

Agora, crie um arquivo `server.php` com a seguinte implementação:

```
<?php

require './vendor/autoload.php';

use Ratchet\Server\EchoServer;

$app = new Ratchet\App('localhost', 9980);
$app->route('/echo', new EchoServer, ['*']);
$app->run();
```

Esse é o servidor mais primitivo que existe, é um “Echo Server”, ele envia de volta tudo o que o cliente manda pra

ele. Nem tivemos a necessidade de implementá-lo, pois ele já vem junto com o Ratchet.

Por fim, crie um arquivo `index.html` com a seguinte implementação:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>WebSocket EchoServer</title>
</head>
<body>
<label for="input">Digite aqui: </label>
<input id="input" type="text"
placeholder="Digite aqui"/>

<div id="response"></div>
<script>
    let input =
document.getElementById('input');
    let response =
document.getElementById('response');
    const socket = new
WebSocket('ws://localhost:9980/echo');


    socket.addEventListener('open', function
() {
        socket.send('Conexão
```

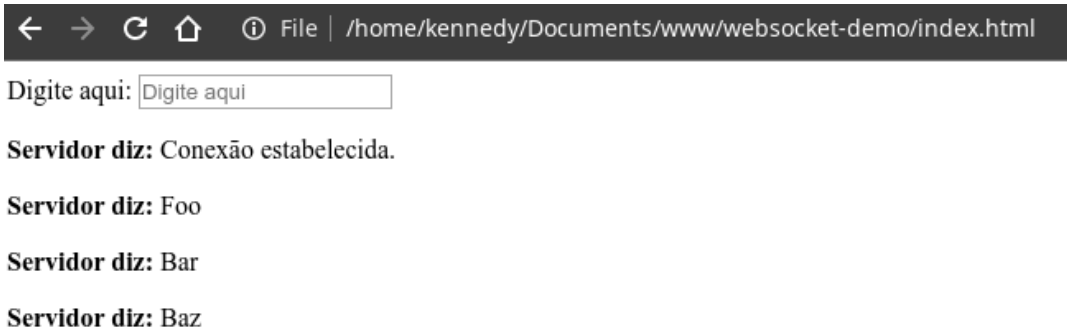
```
estabelecida.');
    });


    socket.addEventListener('message',
function (event) {

response.insertAdjacentHTML('beforeend',
"<p><b>Servidor diz: </b>" + event.data +
"</p>");
    });

    input.addEventListener('keyup', function
(event) {
        if (event.keyCode === 13) {
            socket.send(this.value);
            this.value = '';
        }
    });
</script>
</body>
</html>
```

Para testar, primeiro de tudo temos que ter o servidor rodando. Na raiz do projeto execute:

```
$ php server.php
```

Vai iniciar o servidor na porta `9980`. Por fim, basta executar o arquivo `index.html` e interagir escrevendo mensagens e apertando *enter* para enviá-las. Tudo o que o servidor

receber de input, ele retornará de volta para o cliente.



O objeto WebSocket é nativo e presente em todos os navegadores modernos. Iniciamos o servidor de forma "mágica" usando o Ratchet, mas a realidade é que por debaixo dos panos ele precisa abstrair algumas importantes coisas como o *handshake* inicial, as trocas das mensagens (*Data Frames*) que são criptografadas etc. Toda a dinâmica do funcionamento de um servidor de Websocket pode ser lida nesse documento: Escrevendo um servidor WebSocket.

Agora vamos criar o nosso próprio *wrapper*, nossa própria implementação de um servidor. Primeiro, na raiz do projeto, crie uma pasta chamada `src`. Em seguida, altere o `composer.json` para:

```
{
    "require": {
        "cboden/ratchet": "^0.4.1"
    },
    "autoload": {
        "psr-4": {
            "Chat\\": "src"
        }
```

```
    }
}
```

Agora crie um arquivo `ChatServer.php` dentro da pasta `src` com a seguinte implementação:

```
<?php

namespace Chat;

use Exception;
use SplObjectStorage;
use Ratchet\ConnectionInterface;
use Ratchet\MessageComponentInterface;

final class ChatServer implements
MessageComponentInterface
{
    private $clients;

    public function __construct()
    {
        $this->clients = new
SplObjectStorage();
    }

    public function
onOpen(ConnectionInterface $conn): void
    {
        $this->clients->attach($conn);
```

```
    }

    public function
onMessage(ConnectionInterface $from, $msg):
void
    {
        foreach ($this->clients as $client)
{
            $client->send($msg);
        }
    }

    public function
onClose(ConnectionInterface $conn): void
    {
        $this->clients->detach($conn);
    }

    public function
onError(ConnectionInterface $conn, Exception
$exception): void
    {
        $conn->close();
    }
}
```

Estamos implementando a interface `MessageComponentInterface`. A diferença desse servidor pro *EchoServer*, é que neste estamos guardando as conexões que são estabelecidas e, quando uma

mensagem é recebida, enviamos ela de volta para toda as conexões abertas (que é o que o fazemos no método *onMessage*).

Para testar a implementação, crie um arquivo `chat.html` no projeto:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>WebSocket Simple Chat</title>
</head>
<body>
<p>
    <label for="nome">Seu nome: </label>
    <input id="nome" type="text"
placeholder="Seu nome"/>
</p>
<p>
    <label for="input">Sua mensagem:
</label>
    <input id="input" type="text"
placeholder="Sua mensagem"/>
</p>
<hr>
<div id="chat"></div>
<script>
    let chat =
document.getElementById('chat');
```

```
    let input =
document.getElementById('input');
    const nome =
document.getElementById('nome');
    const socket = new
WebSocket('ws://localhost:9990/chat');


    socket.addEventListener('message',
function (event) {

        const data = JSON.parse(event.data);

        chat.insertAdjacentHTML('beforeend',
"<p><b>" + data.nome + " diz: </b>" +
data.mensagem + "</p>");
    });


    input.addEventListener('keyup', function
(event) {
        if (event.keyCode === 13) {

            const data = {
                nome: nome.value,
                mensagem: this.value,
            };
```

```
socket.send(JSON.stringify(data));

            this.value = '';
        }
    });
</script>
</body>
</html>
```

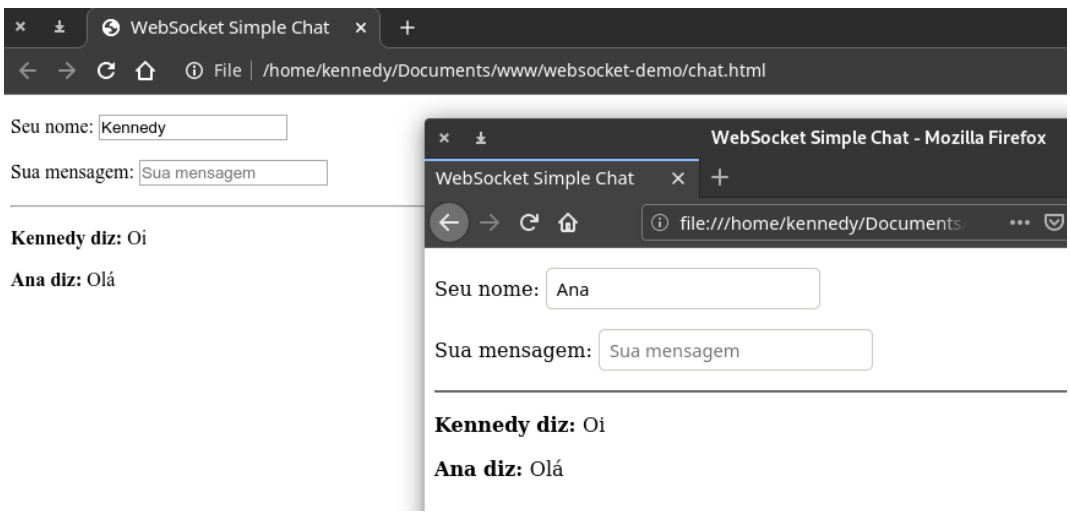
Antes de iniciar o servidor é importante executar o `dump-autoload` do Composer, para que a classe `ChatServer` seja reconhecida:

```
$ composer dump-autoload
```

Por fim, inicie o servidor:

```
$ php chat.php
```

Você pode testar abrindo duas instâncias do chat e usando nomes diferentes:



Os exemplos completos você pode ver nesse repositório: https://github.com/KennedyTedesco/websocket-demo

Só o navegador pode ser cliente do meu servidor?

Não, qualquer outra aplicação pode ser cliente de um servidor WebSocket. Você pode, por exemplo, criar uma aplicação cliente em PHP usando a library Pawl.

Palavras finais

O *Ratchet* simplifica e muito a criação de servidores de WebSocket em PHP, não obstante, ele também suporta sub-protocolos como o Wamp. Em um próximo artigo pretendo abordar um exemplo utilizando-o.

Até a próxima!