Spring

WebSockets
(für Events und Chat im Spiel)

von Tobias Herzog und Alexander Hildebrandt

Übersicht

- Motivation
- Einführung
- Das Öffnen eines WebSockets
- Vor- und Nachteile
- Probleme – Proxys
- Fallback
- Unterstützende Browser
- Zusammenfassung
- Quellen

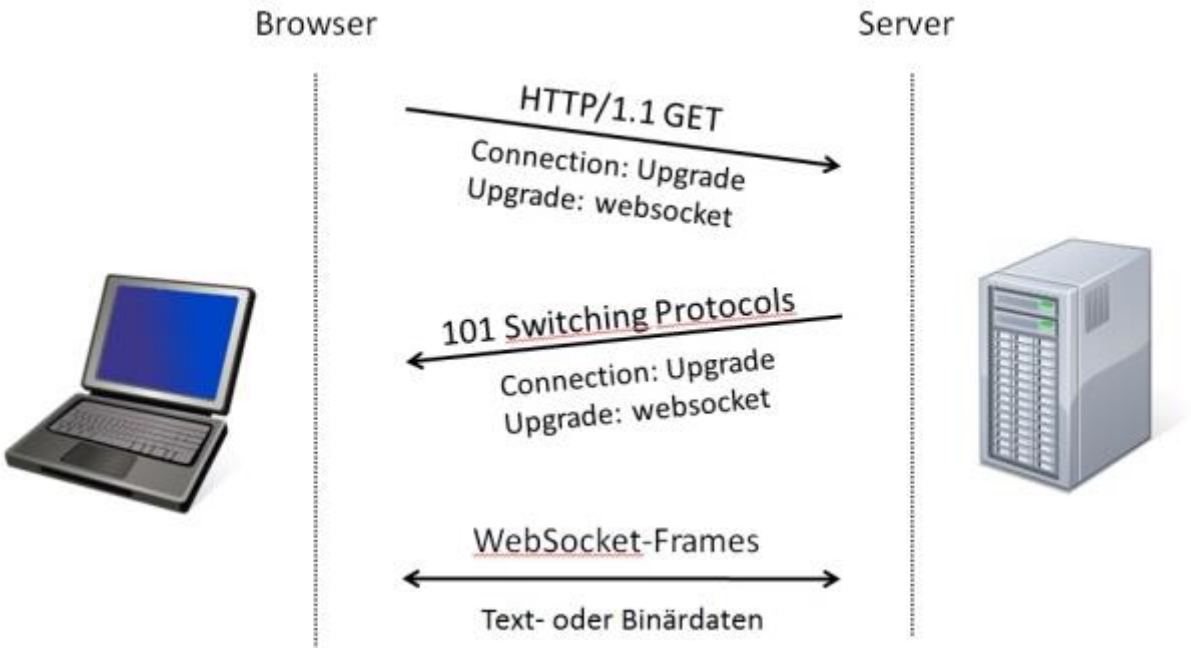
Motivation – Warum WebSockets?

- Traditionelle Kommunikation im Web geht vom Client aus.
- Der Server kann eigenständig keine Mitteilungen senden.
- Workarounds für: low-latency-low-frequency und high-latency-high-frequency
- Wie aber löst man low-latency-high-frequency?

WebSockets – Eine Einführung

- WebSockets bieten eine Möglichkeit der bi-direktionalen Verbindung zwischen Server und Client
- Sie erlauben beiden Seiten das Senden von Informationen
- Sie eignen sich daher gut für Kommunikation, die häufig und in Echtzeit Erfolgen muss.

WebSockets – Eine Einführung



https://entwickler.de/online/web/server-push-und-bidirektionale-kommunikation-mit-websockets-136313.html

Das Öffnen eines WebSockets

- Dem Öffnen eines WebSockets geht immer ein sog. Handshake voraus.
- Der Client nutzt eine normale HTTP-Verbindung um den ersten Kontakt aufzunehmen.
- Diese Verbindung wird daraufhin zu einem WebSocket aufgewertet.

Das Öffnen eines WebSockets

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

Anfrage des Users

Antwort des Servers

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

https://de.wikipedia.org/wiki/WebSocket

Das Öffnen eines WebSockets

```java
package hello;

public class HelloMessage {

    private String name;

    public String getName() {
        return name;
    }

}
```

```java
package hello;

public class Greeting {

    private String content;

    public Greeting(String content) {
        this.content = content;
    }

    public String getContent() {
        return content;
    }

}
```

- Code aus plain old java objects
- HelloMessage: Seite des Users, der sich mit Namen anmeldet
- Greeting: Seite des Servers, der jedem User eine persönliche Begrüßung schickt

Das Öffnen eines WebSockets

```java
package hello;

import org.springframework.messaging.handler.annotation.MessageMapping;
import org.springframework.messaging.handler.annotation.SendTo;
import org.springframework.stereotype.Controller;

@Controller
public class GreetingController {


    @MessageMapping("/hello")
    @SendTo("/topic/greetings")
    public Greeting greeting(HelloMessage message) throws Exception {
        Thread.sleep(3000); // simulated delay
        return new Greeting("Hello, " + message.getName() + "!");
    }

}
```

- Spring-Controller für das Senden einer Begrüßung nach Erhalt einer Nachricht
- @MessageMapping(x): alles, was an x geschickt wird, löst die Methode greeting() aus
- @SendTo(y): schickt das return-Element an alle Abonnenten von y
- “/topic/z“, um an alle Abonnenten zu schicken, “/queue/z“ für einzelne User

Das Öffnen eines WebSockets

```java
package hello;

import org.springframework.context.annotation.Configuration;
import org.springframework.messaging.simp.config.MessageBrokerRegistry;
import org.springframework.web.socket.config.annotation.AbstractWebSocketMessageBroker
import org.springframework.web.socket.config.annotation.EnableWebSocketMessageBroker;
import org.springframework.web.socket.config.annotation.StompEndpointRegistry;

@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig extends AbstractWebSocketMessageBrokerConfigurer {

	@Override
	public void configureMessageBroker(MessageBrokerRegistry config) {
		config.enableSimpleBroker("/topic");
		config.setApplicationDestinationPrefixes("/app");
	}

	@Override
	public void registerStompEndpoints(StompEndpointRegistry registry) {
		registry.addEndpoint("/hello").withSockJS();
	}

}
```

- Konfigurierung von Spring, um WebSockets und STOMP zu erlauben
- @Configuration: Markierung als Spring Konfigurations-Klasse
- @EnableWebSocketBroker: erlaubt den Einsatz von Brokern zur Übersetzung von Nachrichten
- registerStompEndpoints(): Erlaubt Fallback Options, falls WebSockets nicht verfügbar sind

Das Öffnen eines WebSockets

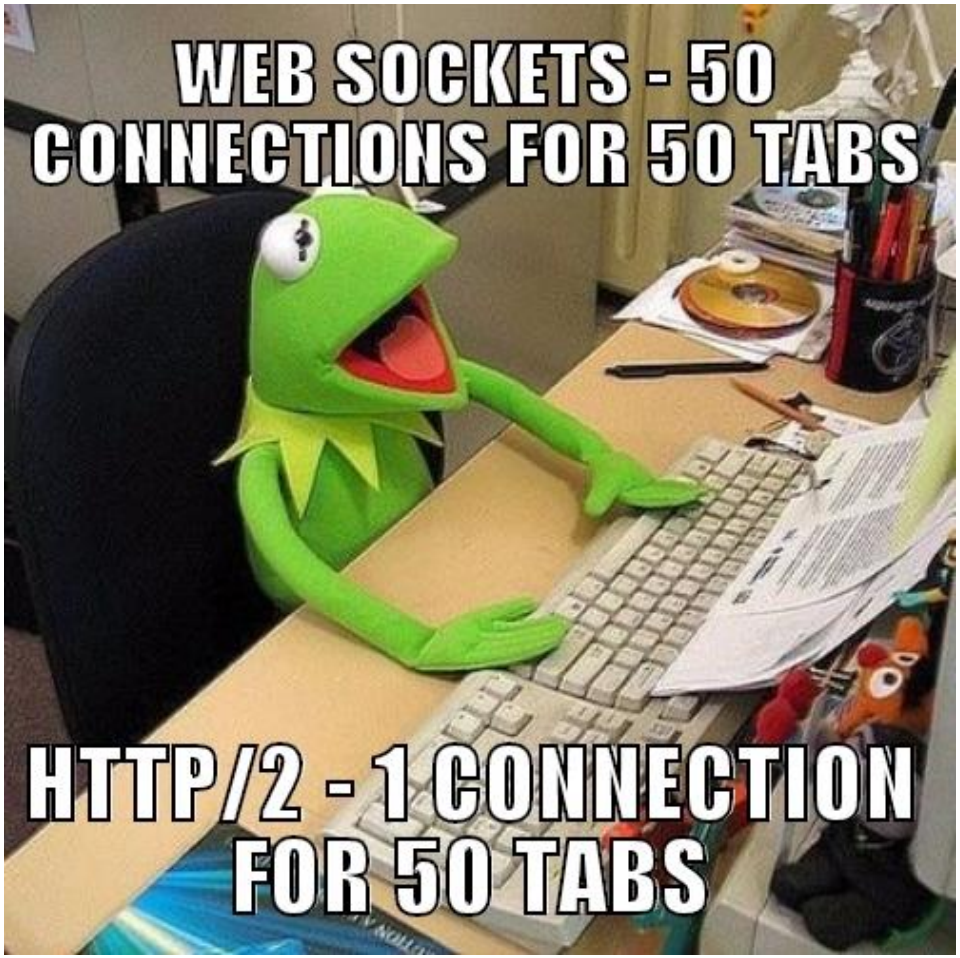
```
function connect() {
    var socket = new SockJS('/hello');
    stompClient = Stomp.over(socket);
    stompClient.connect({}, function(frame) {
        setConnected(true);
        console.log('Connected: ' + frame);
        stompClient.subscribe('/topic/greetings', function(greeting){
            showGreeting(JSON.parse(greeting.body).content);
        });
    });
}

function sendName() {
    var name = document.getElementById('name').value;
    stompClient.send("/app/hello", {}, JSON.stringify({ 'name': name }));
}
```

- Relevante JavaScript Funktionen aus der HTML Datei
- connect(): öffnet Verbindung zu “/hello“, wo GreetingsController wartet. Falls erfolgreich, wird “topics/greetings“ abonniert
- sendName(): nimmt den eingegebenen Namen und sendet ihn an “/app/hello“, wo GreetingsController.greeting() ihn erhält

Nachteile

- Einige (ältere) Browserversionen können nicht damit arbeiten
- Proxy-Server benötigen Fallback Methoden
- Alte (und schlechte) Protokolle müssen für hohe Browser Abdeckung genutzt werden
- WebSockets erlauben eine hohe Menge an gleichzeitig offenen Sockets

Nachteile



https://samsaffron.com/archive/2015/12/29/websockets-caution-required

Nachteile

- WebSockets und HTTP/2 funktionieren nicht zusammen
- Load-Balancing ist sehr schwer möglich

Vorteile

- Reale zweiseitige Kommunikation
- Erhöhte Kommunikationseffizienz
- Simple API
- Durch TCP sehr stabil

Probleme - Proxys

- WebSockets und Proxys haben Schwierigkeiten miteinander zu arbeiten
- HTTPS läuft relativ stabil – HTTP hingegen kann verschiedene Probleme verursachen
  - Verbindungsabbrüche trotz Traffic
  - Das nicht Senden von Daten

Fallback

- Fallbacks sind automatisierte Abläufe, welche unter anderem bei Proxy Problemen ausgeführt werden.
- Sie sollen die Funktionalität der Anwendung gewährleisten, wenn WebSockets nicht richtig funktionieren.

Fallback

- Fallbacks nutzen bewährte Methoden um Echtzeitkommunikation zu simulieren
  - Long-polling
  - Polling
- Über Fallbacks werden häufig auftretende Probleme versucht abzufangen

Unterstützende Browser

- Google Chrome
- Safari
- Opera, ab Version 10.70
- Firefox, ab Version 4
- Internet Explorer, ab Version 10.0

Zusammenfassung

- WebSockets erlauben eine bi-direktionale Verbindung zwischen Server und Client
- Trotz ihrer Nützlichkeit sind sie mit Vorsicht zu nutzen, da sie einige Probleme/Nachteile mit sich bringen
- Diese Probleme werden versucht mit guten Fallback Libraries zu fangen

Quellen

- http://docs.spring.io/spring/docs/current/spring-framework-reference/html/websocket.html
- http://www.html5rocks.com/de/tutorials/websockets/basics/
- http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/HomeWebsocket/WebsocketHome.html

Quellen

- https://de.wikipedia.org/wiki/Spring_(Framework)
- https://de.wikipedia.org/wiki/WebSocket
- https://www.engineyard.com/articles/websocket
- https://banksco.de/p/state-of-realtime-web-2016.html

Quellen

- https://samsaffron.com/archive/2015/12/29/websockets-caution-required
- https://spring.io/guides/gs/messaging-stomp-websocket/