DARIAWAN

Spring →

Spring Boot + WebSocket With STOMP Tutorial

In previous article we learn on how to create a simple broadcast application using Spring Boot and plain WebSocket. In this article, we will create similar application not only using **WebSocket**, but adding **STOMP** on top of it.

STOMP

Simple Text Oriented Messaging Protocol (**STOMP**), is a simple text-based protocol, designed for working with message-oriented middleware (MOM). Any STOMP client can communicate with any STOMP message broker and be interoperable among languages and platforms.

So, why using **STOMP** if we are already using **WebSocket**? Or vice-versa, why using **WebSocket** if we are using **STOMP**?

- **STOMP** describes the message format exchanged between clients and servers. On another hand, **WebSocket** is nothing but a communication protocol.
- You can't "just use" **STOMP** to communicate with a server or a message broker. You have to use a transport to send those STOMP messages, one of them is **WebSocket**.
- **STOMP** doesn't take care of the **WebSocket** handshake, in fact, it's not aware of it at all. We can use STOMP on top of another transport protocol (example: HTTP) and see no difference from the STOMP perspective.

- Feature like to send a message only to users who are subscribed to a particular topic, or to send a message to a particular user is harder to implement with plain **WebSocket**, but **STOMP** has all this features, since it's designed to interact with message broker.

Let's start dig into the project. We still use the same project that we created in previous article. The Spring Boot's main entry point also still `WebSocketExampleApplication.`

Create a DTO

Since we will exchanging messages in JSON format, we need to a data transfer object (DTO) class. The DTO class is `ChatMessage` :

ChatMessage.java

```java
package com.dariawan.websocket.dto;

import com.dariawan.websocket.util.StringUtils;
import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
public class ChatMessage {

    private String from;
    private String text;
    private String recipient;
    private String time;

    public ChatMessage() {

    }

    public ChatMessage(String from, String text, String recipient) {
        this.from = from;
        this.text = text;
        this.recipient = recipient;
        this.time = StringUtils.getCurrentTimeStamp();
    }
}
```

We are using `lombok` for setter and getter. `StringUtils` is our small utility class:

StringUtils.java

```
package com.dariawan.websocket.util;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class StringUtils {

    private static final String TIME_FORMATTER= "HH:mm:ss";

    public static String getCurrentTimeStamp() {
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern(TIME_FORMATTER);
        return LocalDateTime.now().format(formatter);
    }
}
```

Create Controller Class

`WebSocketBroadcastController` is our controller for this sample:

WebSocketBroadcastController.java

```
package com.dariawan.websocket.controller;

import com.dariawan.websocket.dto.ChatMessage;
import org.springframework.messaging.handler.annotation.MessageMapping;
import org.springframework.messaging.handler.annotation.SendTo;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class WebSocketBroadcastController {

```

```java

    @GetMapping("/stomp-broadcast")
    public String getWebSocketBroadcast() {
        return "stomp-broadcast";
    }

    @MessageMapping("/broadcast")
    @SendTo("/topic/messages")
    public ChatMessage send(ChatMessage chatMessage) throws Exception {
        return new ChatMessage(chatMessage.getFrom(), chatMessage.getText(), "ALL");
    }
}
```

Function `getWebSocketBroadcast()` will return the name of html template, `stomp-broadcast.html` that will be rendered by Thymeleaf engine. We will revisit this later. But let's focus on function `send(ChatMessage)`. This is will relate to our configuration later in next section.

Handling **WebSocket** requests happens in a similar way to normal HTTP requests, but we are not using `@RequestMapping` or `@GetMapping` , but `@SubscribeMapping` and `@MessageMapping` depending on the case. We are using `@MessageMapping` to map messages headed for the `/broadcast` . Check application destination prefixes in configuration section below.

`@SendTo` indicates that the return value of a message-handling method should be sent as a Message to the specified destination, which in our case is `/topic/broadcast`. Check about enable a simple message broker for subscription in configuration section below.

So in above example, function `send(ChatMessage)` will converted messages that headed to `/broadcast` endpoint (to be precise: `/app/broadcast`), convert it to new ChatMessage and send to `/topic/messages`, so all subscribers for `/topic/messages` will receive this broadcast message.

WebSocket Configuration for STOMP Messaging

Configure Spring to enable **WebSocket** and **STOMP** messaging by creating `WebSocketMessageBrokerConfig`:

WebSocketMessageBrokerConfig.java

```java
package com.dariawan.websocket.config;

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

@Configuration
@EnableWebSocketMessageBroker
public class WebSocketMessageBrokerConfig implements WebSocketMessageBrokerConfigurer {

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

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

Let's check several items in `WebSocketMessageBrokerConfig` :

- Annotate with `@Configuration` to indicate that this is a Spring configuration class.
- Annotate with `@EnableWebSocketMessageBroker` to enables **WebSocket** message handling, backed by a message broker.
- Enable a simple message broker and configure destination prefix(es). Simple broker means a simple in-memory broker, and in our example the destination prefix is `/topic` . The client app will subscribe messages at endpoints starting with these configured prefix(es), in our example: `/topic/broadcast` .
- Set application destination prefixes, in our sample is `/app` . The client will send messages at this endpoint. For example, if client sends message at `/app/broadcast` , the endpoint

configured at `/broadcast` in the spring controller will be invoked.

- Enable **STOMP** support by register **STOMP** endpoint at `/broadcast`. This is the endpoint used by clients to connect to **STOMP**.

HTML Template (and JavaScript Client)

Now time to go back to `stomp-broadcast.html`. Here the content of this html:

stomp-broadcast.html

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>WebSocket With STOMP Broadcast Example</title>
        <th:block th:include="fragments/common.html :: headerfiles"></th:block>
    </head>
    <body>
        <div class="container">
            <div class="py-5 text-center">
                <a href="/"><h2>WebSocket</h2></a>
                <p class="lead">WebSocket Broadcast - with STOMP</p>
            </div>
            <div class="row">
                <div class="col-md-6">
                    <div class="mb-3">
                        <div class="input-group">
                            <input type="text" id="from" class="form-control" placeholde
                            <div class="btn-group">
                                <button type="button" id="connect" class="btn btn-sm btn
                            <button type="button" id="disconnect" class="btn btn-sm btn-
                            </div>
                        </div>
                    </div>
                    <div class="mb-3">
                        <div class="input-group" id="sendmessage" style="display: none;"
                            <input type="text" id="message" class="form-control" placeho
                            <div class="input-group-append">
                                <button id="send" class="btn btn-primary" onclick="send(
                            </div>
                        </div>
                    </div>
                </div>
                <div class="col-md-6">
                    <div id="content"></div>
                    <div>
                        <span class="float-right">
                            <button id="clear" class="btn btn-primary" onclick="clearBro
                        </span>
                    </div>
                </div>
            </div>
        </div>

        <footer th:insert="fragments/common.html :: footer"></footer>

        <script th:src="@{/webjars/stomp-websocket/2.3.3-1/stomp.js}" type="text/javascr
        <script type="text/javascript">
            var stompClient = null;
            var userName = $("#from").val();

            function setConnected(connected) {
                $("#from").prop("disabled", connected);
                $("#connect").prop("disabled", connected);
                $("#disconnect").prop("disabled", !connected);
                if (connected) {
                    $("#sendmessage").show();
                } else {
                    $("#sendmessage").hide();
                }
            }

            function connect() {
                userName = $("#from").val();
                if (userName == null || userName === "") {
                    alert('Please input a nickname!');
                    return;
                }
                 /*<![CDATA[*/
                var url = /*[['ws://'+${#httpServletRequest.serverName}+':'+${#httpServl
                /*]]>*/
                stompClient = Stomp.client(url);
                stompClient.connect({}, function () {
                    stompClient.subscribe('/topic/broadcast', function (output) {
                        showBroadcastMessage(createTextNode(JSON.parse(output.body)));
                    });

                    sendConnection(' connected to server');
                    setConnected(true);
                }, function (err) {
                    alert('error' + err);
                });
            }

            function disconnect() {
                if (stompClient != null) {
                    sendConnection(' disconnected from server');

                    stompClient.disconnect(function() {
                        console.log('disconnected...');
                        setConnected(false);
                    });
                }
            }

            function sendConnection(message) {
                var text = userName + message;
                sendBroadcast({'from': 'server', 'text': text});
            }

            function sendBroadcast(json) {
                stompClient.send("/app/broadcast", {}, JSON.stringify(json));
            }

            function send() {
                var text = $("#message").val();
                sendBroadcast({'from': userName, 'text': text});
                $("#message").val("");
            }

```

```
            function createTextNode(messageObj) {
                return '<div class="row alert alert-info"><div class="col-md-8">' +
                        messageObj.text +
                        '</div><div class="col-md-4 text-right"><small>[<b>' +
                        messageObj.from +
                        '</b> ' +
                        messageObj.time +
                        ']</small>' +
                        '</div></div>';
            }

            function showBroadcastMessage(message) {
                $("#content").html($("#content").html() + message);
                $("#clear").show();
            }

            function clearBroadcast() {
                $("#content").html("");
                $("#clear").hide();
            }
        </script>
    </body>
</html>
```

Notice on how we include `stomp.js` from webjars:

```
<script th:src="@{/webjars/stomp-websocket/2.3.3-1/stomp.js}" type="text/javascript"></script
```

You can get this jar by declare it as dependency in your Maven's pom.xml:

```
<!-- https://mvnrepository.com/artifact/org.webjars/stomp-websocket -->
<dependency>
    <groupId>org.webjars</groupId>
    <artifactId>stomp-websocket</artifactId>
    <version>2.3.3-1</version>
</dependency>
```

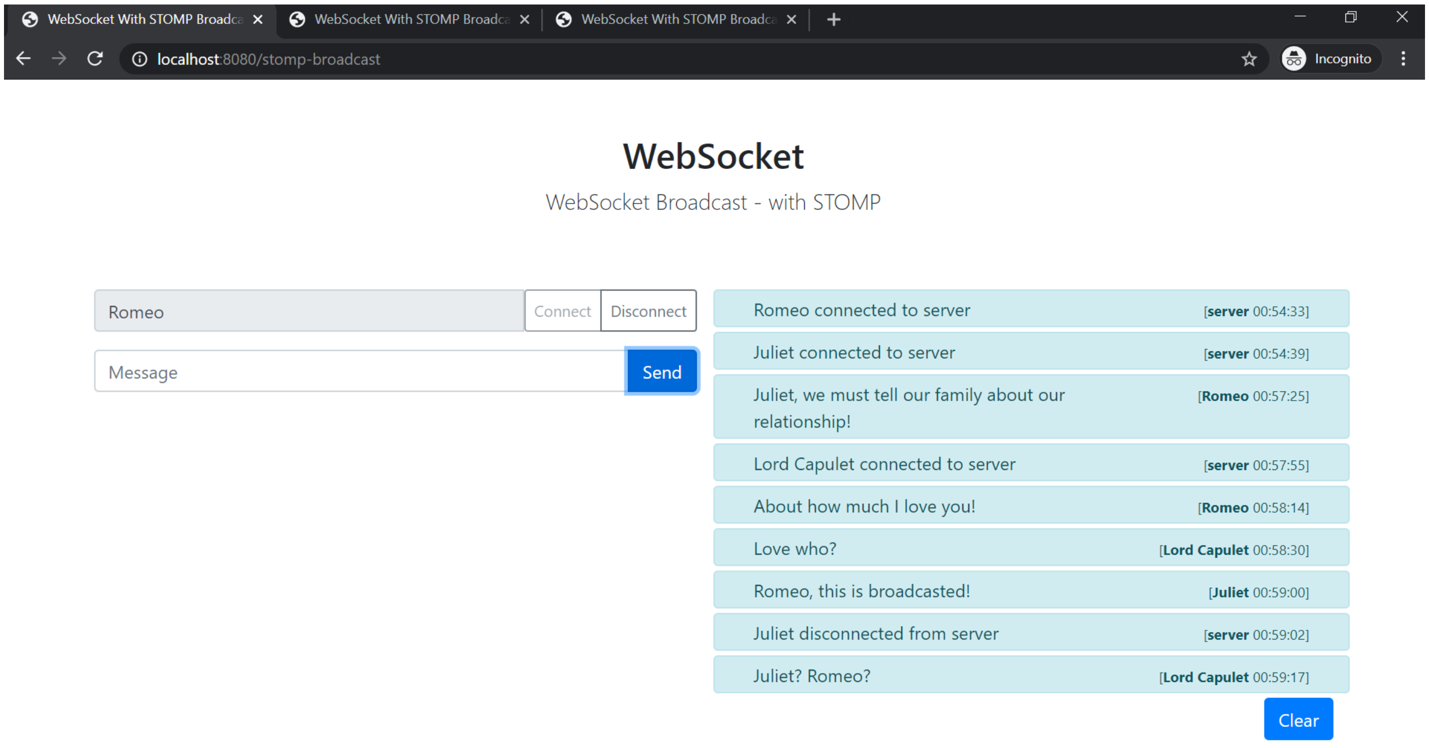
And you sure easily able to identify, on how to make a **STOMP** connection via javascript (stomp.js). Here the snippet:

```
stompClient = Stomp.client(url);
stompClient.connect({}, function () {
        stompClient.subscribe('/topic/broadcast', function (output) {
                // what happen if we got message?
        });


        ...
}, function (err) {
        // what happen if error occurs?
});
```

Running the Application

Let's run our application, and open http://localhost:8080/stomp-broadcast in the browser. Here a screen shot of the application that we completed in this tutorial:


WebSocket With STOMP Broadca
WebSocket With STOMP Broadca
WebSocket With STOMP Broadca
localhost:8080/stomp-broadcast
Incognito
WebSocket
WebSocket Broadcast - with STOMP
Romeo
Connect
Disconnect
Message
Send
Clear

Romeo connected to server [**server** 00:54:33]
Juliet connected to server [**server** 00:54:39]
Juliet, we must tell our family about our relationship! [**Romeo** 00:57:25]
Lord Capulet connected to server [**server** 00:57:55]
About how much I love you! [**Romeo** 00:58:14]
Love who? [**Lord Capulet** 00:58:30]
Romeo, this is broadcasted! [**Juliet** 00:59:00]
Juliet disconnected from server [**server** 00:59:02]
Juliet? Romeo? [**Lord Capulet** 00:59:17]

http://localhost:8080/stomp-broadcast

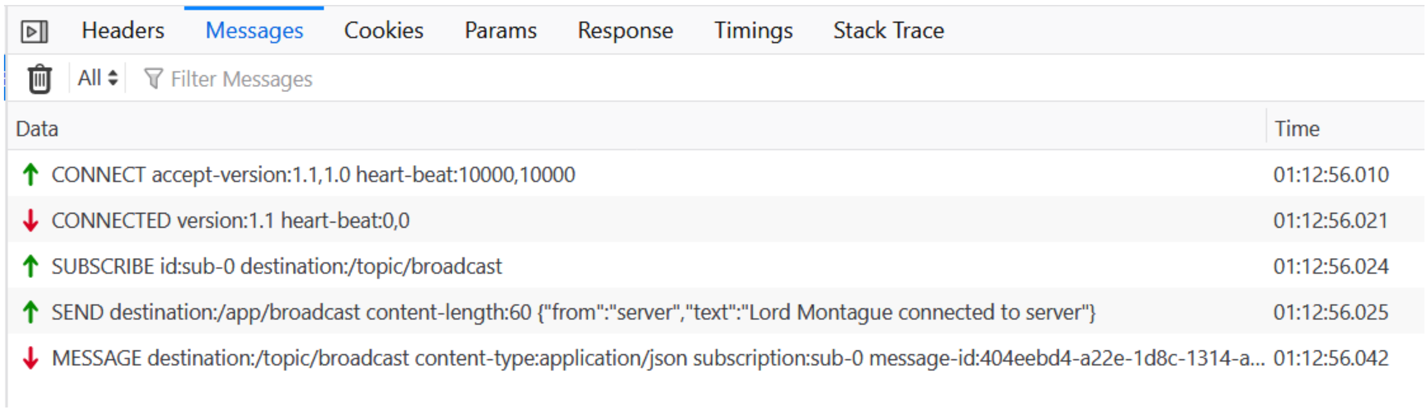
Let's check the connection when make connection by clicking "Connect" button. Request headers:

```
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:72.0) Gecko/20100101 Firefox/72.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Sec-WebSocket-Version: 13
Origin: http://localhost:8080
Sec-WebSocket-Protocol: v10.stomp, v11.stomp
Sec-WebSocket-Extensions: permessage-deflate
Sec-WebSocket-Key: 0566sTUvnNECJ7dWhLLr7g==
Connection: keep-alive, Upgrade
Pragma: no-cache
Cache-Control: no-cache
Upgrade: websocket
```

And the Response Headers:

```
HTTP/1.1 101
Upgrade: websocket
Connection: upgrade, keep-alive
Sec-WebSocket-Accept: x9YEV1lGVGHmwbdMq4DZi55HC4M=
Sec-WebSocket-Protocol: v10.stomp
Sec-WebSocket-Extensions: permessage-deflate
Date: Thu, 13 Feb 2020 17:12:56 GMT
Keep-Alive: timeout=60
```

We can see that stomp used in `Sec-WebSocket-Protocol` . And here the messages that happen when "Connect" button is clicked (from Mozilla):



Data	Time
↑ CONNECT accept-version:1.1,1.0 heart-beat:10000,10000	01:12:56.010
↓ CONNECTED version:1.1 heart-beat:0,0	01:12:56.021
↑ SUBSCRIBE id:sub-0 destination:/topic/broadcast	01:12:56.024
↑ SEND destination:/app/broadcast content-length:60 {"from":"server","text":"Lord Montague connected to server"}	01:12:56.025
↓ MESSAGE destination:/topic/broadcast content-type:application/json subscription:sub-0 message-id:404eebd4-a22e-1d8c-1314-a...	01:12:56.042

Firefox Web Developer - Network - Messages

Based on our code flow, we can see on how the client make connection, and subscribe to `/topic/broadcast` after connected, send a message, and receive the message (broadcast).

References:

- STOMP over websockets vs plain STOMP. Which one is better?

Liked this Tutorial? Share it on Social media!

Twitter Facebook Linkedin Reddit WhatsApp

This article is part of Build Spring WebSocket Application Series.

Other articles in this series:

- Spring Boot + WebSocket Basic Example
- Create Spring Boot + WebSocket Application using STOMP and SockJS
- Build a Chat Application Using Spring Boot and WebSocket

← Spring Boot + WebSocket Basic Example

Published 13 February 2020

Desson Ariawan

Programmer

TAGS

Java

Spring

Spring Boot

Thymeleaf

NEXT

Create Spring Boot + WebSocket Application using STOMP and SockJS

DESSON ARIAWAN

Sign up to our newsletter Your email Sign up

Oh... In my spare time, I love to travel, take photos, and exploring new technology

Got exciting ideas? Let's get in touch

hello@dariawan.com

@dariawantech

dariawantech

Do One Thing Well | Source Codes | Privacy Policy