链滴

SpringBoot 入门教程 (十六) | 整合 WebSocket 实现广播

作者：JavaFish

原文链接：https://ld246.com/article/1621600454949

来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

01 前言

如题，今天介绍的是 SpringBoot 整合 WebSocket 实现广播消息。

02 什么是 WebSocket ?

WebSocket 为浏览器和服务器提供了双工异步通信的功能，即浏览器可以向服务器发送信息，反之成立。

WebSocket 是通过一个 socket 来实现双工异步通信能力的，但直接使用 WebSocket (或者 SockJ : WebSocket 协议的模拟，增加了当前浏览器不支持使用 WebSocket 的兼容支持) 协议开发程序得十分繁琐，所以使用它的子协议 STOMP。

03 STOMP 协议简介

它是高级的流文本定向消息协议，是一种为 MOM (Message Oriented Middleware，面向消息的间件) 设计的简单文本协议。

它提供了一个可互操作的连接格式，允许 STOMP 客户端与任意 STOMP 消息代理 (Broker) 进行交，类似于 OpenWire (一种二进制协议)。

由于其设计简单，很容易开发客户端，因此在多种语言和多种平台上得到广泛应用。其中最流行的 ST MP 消息代理是 Apache ActiveMQ。

STOMP 协议使用一个基于 (frame) 的格式来定义消息，与 Http 的 request 和 response 类似 。

04 广播

接下来，实现一个广播消息的 demo。即服务端有消息时，将消息发送给所有连接了当前 endpoint 浏览器。

05 准备工作

- SpringBoot 2.1.3
- IDEA
- JDK8

06 Pom 依赖配置

```
<dependencies>
    <!-- thymeleaf 模板引擎 -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <!-- web 启动类 -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
```

```
        </dependency>
        <!-- WebSocket 依赖 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-websocket</artifactId>
        </dependency>
        <!-- test 单元测试 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

代码注释很详细，不多说。

07 配置 WebSocket

实现 WebSocketMessageBrokerConfigurer 接口，注册一个 STOMP 节点，配置一个广播消息代理

```
@Configuration
// @EnableWebSocketMessageBroker注解用于开启使用STOMP协议来传输基于代理（MessageBroker）的消息，这时候控制器（controller）
// 开始支持@MessageMapping,就像是使用@requestMapping一样。
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        //注册一个 Stomp 的节点(endpoint),并指定使用 SockJS 协议。
        registry.addEndpoint("/endpointNasus").withSockJS();
    }

    @Override
    public void configureMessageBroker(MessageBrokerRegistry registry) {
        // 广播式配置名为 /nasus 消息代理 , 这个消息代理必须和 controller 中的 @SendTo 配置的址前缀一样或者全匹配
        registry.enableSimpleBroker("/nasus");
    }
}
```

7.1 消息类

客户端发送给服务器：

```
public class Client2ServerMessage {

    private String name;

    public String getName() {
        return name;
    }
```

```
    public void setName(String name) {
        this.name = name;
    }
}
```

服务器发送给客户端：

```
public class Server2ClientMessage {

    private String responseMessage;

    public Server2ClientMessage(String responseMessage) {
        this.responseMessage = responseMessage;
    }

    public String getResponseMessage() {
        return responseMessage;
    }

    public void setResponseMessage(String responseMessage) {
        this.responseMessage = responseMessage;
    }
}
```

7.2 演示控制器代码

```
@RestController
public class WebSocketController {

    @MessageMapping("/hello") // @MessageMapping 和 @RequestMapping 功能类似，浏览
向服务器发起消息，映射到该地址。
    @SendTo("/nasus/getResponse") // 如果服务器接受到了消息，就会对订阅了 @SendTo 括号
的地址的浏览器发送消息。
    public Server2ClientMessage say(Client2ServerMessage message) throws Exception {
        Thread.sleep(3000);
        return new Server2ClientMessage("Hello," + message.getName() + "!");
    }

}
```

7.3 引入 STOMP 脚本

将 stomp.min.js (STOMP 客户端脚本) 和 sockJS.min.js (sockJS 客户端脚本) 以及 Jquery 放在 reso rce 文件夹的 static 目录下。

7.4 演示页面

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8" />
    <title>Spring Boot+WebSocket+广播式</title>
```

```
</head>
<body onload="disconnect()">
<noscript><h2 style="color: #ff0000">貌似你的浏览器不支持websocket</h2></noscript>
<div>
    <div>
        <button id="connect" onclick="connect();">连接</button>
        <button id="disconnect" disabled="disabled" onclick="disconnect();">断开连接</butto
>
    </div>
    <div id="conversationDiv">
        <label>输入你的名字</label><input type="text" id="name" />
        <button id="sendName" onclick="sendName();">发送</button>
        <p id="response"></p>
    </div>
</div>
<script th:src="@{sockjs.min.js}"></script>
<script th:src="@{stomp.min.js}"></script>
<script th:src="@{jquery.js}"></script>
<script type="text/javascript">
    var stompClient = null;

    function setConnected(connected) {
        document.getElementById('connect').disabled = connected;
        document.getElementById('disconnect').disabled = !connected;
        document.getElementById('conversationDiv').style.visibility = connected ? 'visible' : 'hidd
n';
        $('#response').html();
    }

    function connect() {
        // 连接 SockJs 的 endpoint 名称为 "/endpointNasus"
        var socket = new SockJS('/endpointNasus');
        // 使用 STOMP 子协议的 WebSocket 客户端
        stompClient = Stomp.over(socket);
        stompClient.connect({}, function(frame) {
            setConnected(true);
            console.log('Connected: ' + frame);
            // 通过 stompClient.subscribe 订阅 /nasus/getResponse 目标发送的信息，对应控制器的
endTo 定义
            stompClient.subscribe('/nasus/getResponse', function(respnose){
            // 展示返回的信息，只要订阅了 /nasus/getResponse 目标，都可以接收到服务端返回的信

            showResponse(JSON.parse(respnose.body).responseMessage);
            });
        });
    }


    function disconnect() {
        // 断开连接
        if (stompClient != null) {
            stompClient.disconnect();
        }
```

```
        setConnected(false);
        console.log("Disconnected");
    }

    function sendName() {
        // 向服务端发送消息
        var name = $('#name').val();
        // 通过 stompClient.send 向 /hello （服务端）发送信息，对应控制器 @MessageMapping 的定义
        stompClient.send("/hello", {}, JSON.stringify({ 'name': name }));
    }

    function showResponse(message) {
          // 接收返回的消息
          var response = $("#response");
          response.html(message);
    }
</script>
</body>
</html>
```

7.5 页面 Controller

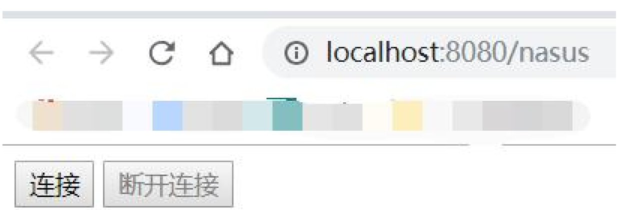
注意，这里使用的是 @Controller 注解，用于匹配 html 前缀，加载页面。

```
@Controller
public class ViewController {

    @GetMapping("/nasus")
    public String getView(){
        return "nasus";
    }
}
```

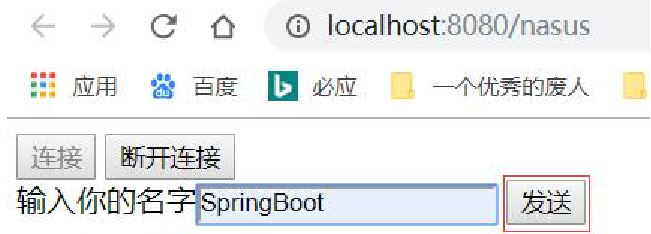
08 测试结果

打开三个窗口访问 http://localhost:8080/nasus ，初始页面长这样：

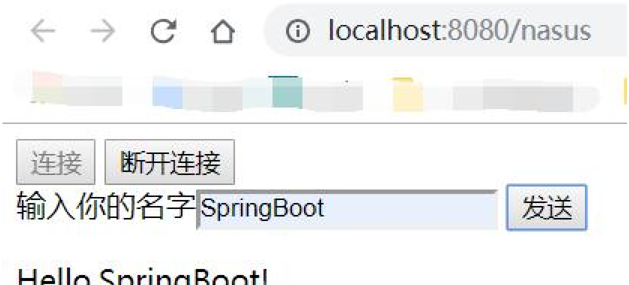


三个页面全部点连接，点击连接订阅 endpoint ，如下图：

在第一个页面，输入名字，点发送 ，如下图：



在第一个页面发送消息，等待 3 秒，结果是 3 个页面都接受到了服务端返回的信息，广播成功。



原文链接：SpringBoot 入门教程 (十六) | 整合 WebSocket 实现广播

源码下载：

https://github.com/turoDog/Demo/tree/master/springboot_websocket_demo

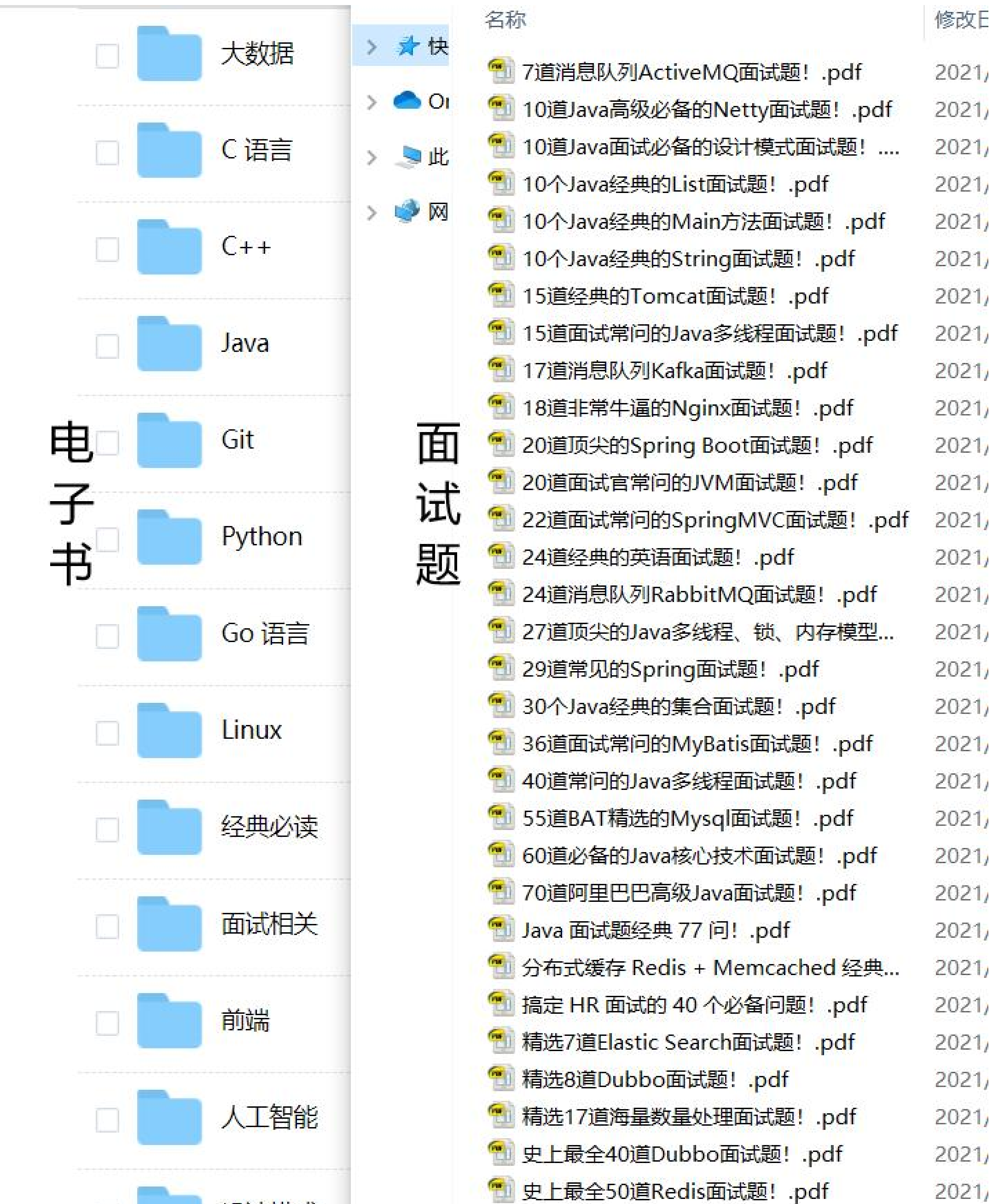
如果觉得对你有帮助，请给个 Star 再走呗，非常感谢。

09 大厂面试题

如果本文对你哪怕有一丁点帮助，请帮忙点好看。你的好看是我坚持写作的动力。

初次见面，也不知道送你们啥。干脆就送**几百本电子书**和**2021最新面试资料**吧。微信搜索**JavaFish**复**电子书**送你 1000+ 本编程电子书；回复**面试**获取 50 套大厂面试题；回复**1024**送你一套完整的 jav 视频教程。

面试题都是有答案的，如下所示：有需要的就来拿吧，**绝对免费，无套路获取。**



原文链接：SpringBoot 入门教程 (十六) | 整合 WebSocket 实现广播

JavaFish

微信扫描二维码，关注我的公众号