

🏠 首页 (<https://www.zifangsky.cn/>) » Java (<https://www.zifangsky.cn/java/>) » Spring (<https://www.zifangsky.cn/java/spring/>) » 正文

## Spring Boot中使用WebSocket总结（二）：向指定用户发送WebSocket消息并处理对方不在线的情况

📅 2018/11/20 | 📁 Spring (<https://www.zifangsky.cn/java/spring/>) | 👤 admin (<https://www.zifangsky.cn/author/zifangskyw/>)

在上一篇文章 (<https://www.zifangsky.cn/1355.html>) 中我介绍了在Spring项目中使用WebSocket的几种实现方式。但是，上篇文章中只介绍了服务端采用广播模式给所有客户端发送消息，然而我们有时需要服务端给指定用户的客户端发送消息（比如：发送Web通知、实时打印用户任务的日志、两个用户点对点聊天等）。

关于服务端如何给指定用户的客户端发送消息，一般可以通过以下三种方案来实现：

- **方案一：** WebSocket使用“Java提供的@ServerEndpoint注解”实现或者使用“Spring低层级API”实现，在建立连接时从 HttpSession 中获取用户登录后的用户名，然后把“**用户名+该WebSocket连接**”存储到 ConcurrentHashMap 。给指定用户发送消息，只需要**根据接收者的用户名获取对方已经建立的WebSocket连接，接着给他发送消息即可。**
- **方案二：** 在页面的监听路径前面动态添加当前登录的“**用户ID/用户名**”，这样给指定用户发送消息，只需要发送广播消息到监听了前面那个路径的客户端即可。
- **方案三：** 这种方案类似于方案一。使用Spring的高级API实现WebSocket，然后自定义 HandshakeHandler 类并重写 determineUser 方法，其目的是为了在建立连接时使用用户登录后的用户名作为此次WebSocket的凭证，最后我们就可以使用 `messagingTemplate.convertAndSendToUser` 方法给指定用户发送消息了。

注：本篇文章的完整源码可以参考：<https://github.com/zifangsky/WebSocketDemo>  
(<https://github.com/zifangsky/WebSocketDemo>)

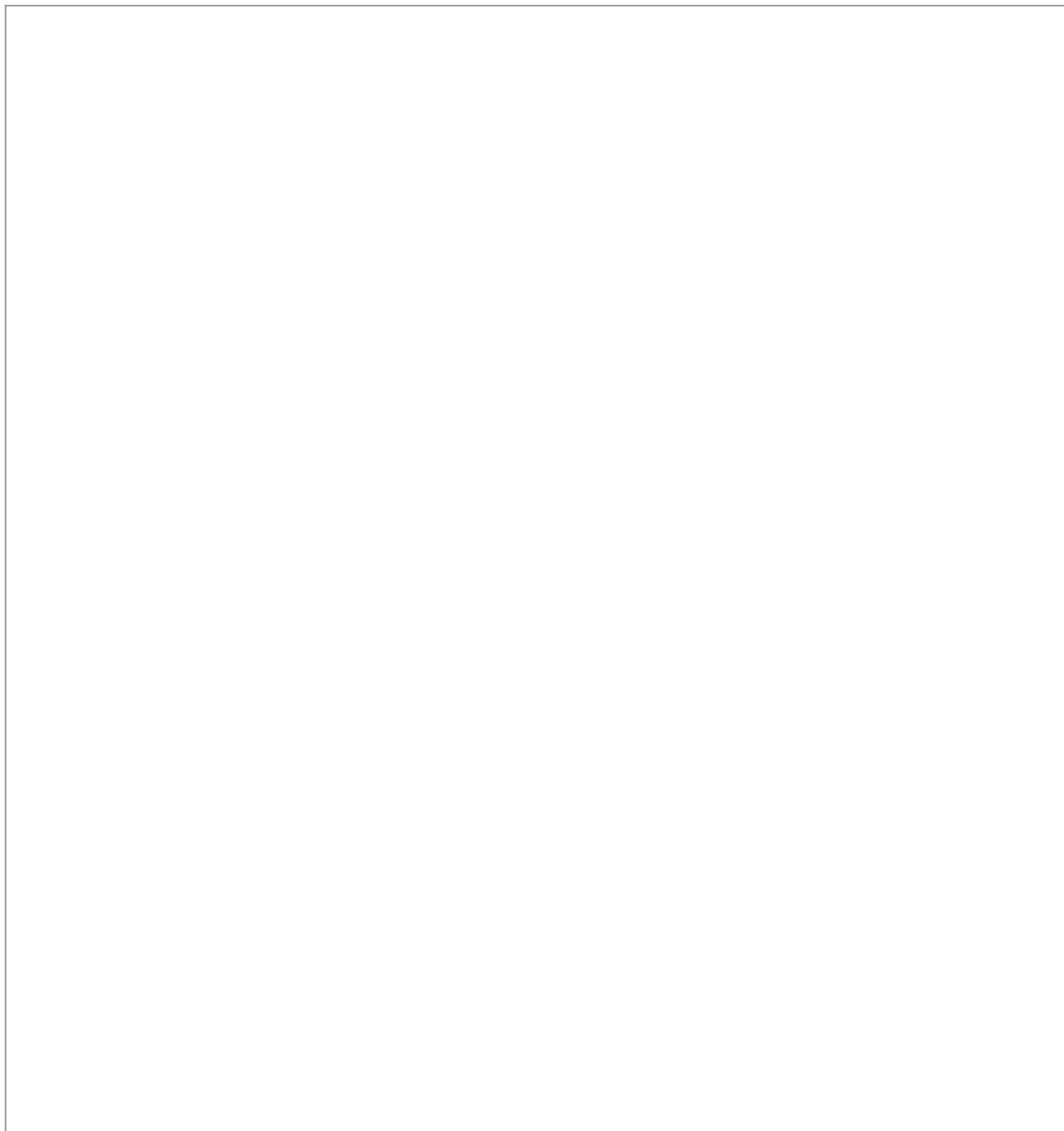
## 使用SimpMessagingTemplate发送消息

使用 `org.springframework.messaging.simp.SimpMessagingTemplate` 类可以在服务端的任意地方给客户端发送消息。此外，在我们配置Spring支持STOMP后 `SimpMessagingTemplate` 类就会被自动装配到Spring的上下文中，因此我们只需要在想要使用的地方使用 `@Autowired` 注解注入 `SimpMessagingTemplate`即可使用。

需要说明的是， `SimpMessagingTemplate` 类有两个重要的方法，它们分别是：

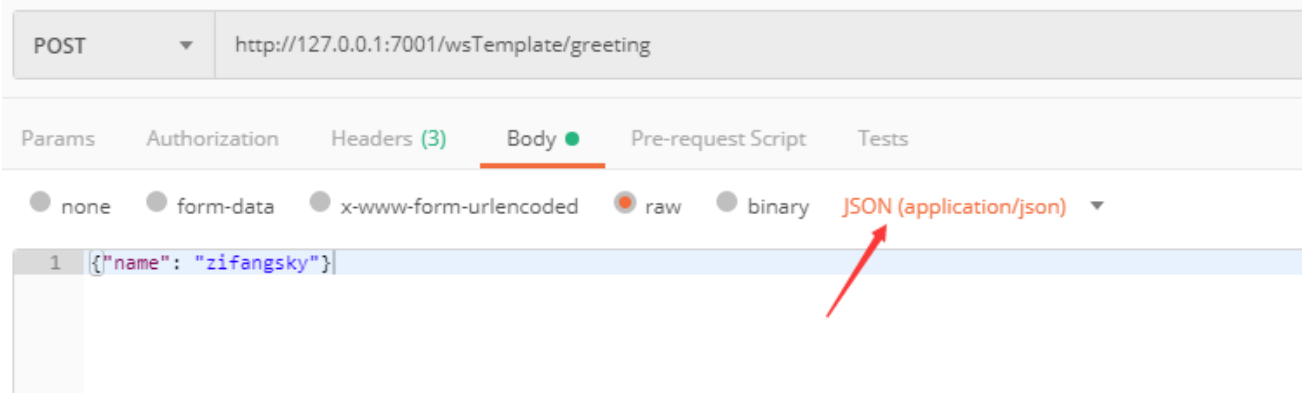
- `public void convertAndSend(D destination, Object payload)`：给监听了路径 `destination` 的所有客户端发送消息 `payload`
- `public void convertAndSendToUser(String user, String destination, Object payload)`：给监听了路径 `destination` 的用户 `user` 发送消息 `payload`

## 一个简单示例：



```
1 package cn.zifangsky.stompwebsocket.controller;
2
3 import cn.zifangsky.stompwebsocket.model.websocket.Greeting;
4 import cn.zifangsky.stompwebsocket.model.websocket.HelloMessage;
5 import cn.zifangsky.stompwebsocket.service.RedisService;
6 import org.slf4j.Logger;
7 import org.slf4j.LoggerFactory;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.messaging.simp.SimpMessagingTemplate;
10 import org.springframework.messaging.simp.user.SimpUserRegistry;
11 import org.springframework.stereotype.Controller;
12 import org.springframework.web.bind.annotation.PostMapping;
13 import org.springframework.web.bind.annotation.RequestBody;
14 import org.springframework.web.bind.annotation.RequestMapping;
15 import org.springframework.web.bind.annotation.ResponseBody;
16
17 import javax.annotation.Resource;
18
19 /**
20  * 测试{@link org.springframework.messaging.simp.SimpMessagingTemplate}类的基本用法
21  * @author zifangsky
22  * @date 2018/10/10
23  * @since 1.0.0
24  */
25 @Controller
26 @RequestMapping("/wsTemplate")
27 public class MessageTemplateController {
28     private final Logger logger = LoggerFactory.getLogger(getClass());
29
30     @Autowired
31     private SimpMessagingTemplate messagingTemplate;
32
33     @Autowired
34     private SimpUserRegistry userRegistry;
35
36     @Resource(name = "redisServiceImpl")
37     private RedisService redisService;
38
39     /**
40      * 简单测试SimpMessagingTemplate的用法
41      */
42     @PostMapping("/greeting")
43     @ResponseBody
44     public String greeting(@RequestBody Greeting greeting) {
45         this.messagingTemplate.convertAndSend("/topic/greeting", new HelloMessage("Hel
46
47         return "ok";
48     }
49
50 }
```

很显然，这里发送的地址是上篇文章中最后那个示例监听的地址，在客户端页面建立连接后，我们使用 Postman 请求一下上面这个方法，效果如下：



然后我们可以发现页面中也收到消息了：

```
<<< MESSAGE
destination:/topic/greeting
content-type:application/json;charset=UTF-8
subscription:sub-0
message-id:oja2jrwg-3
content-length:30

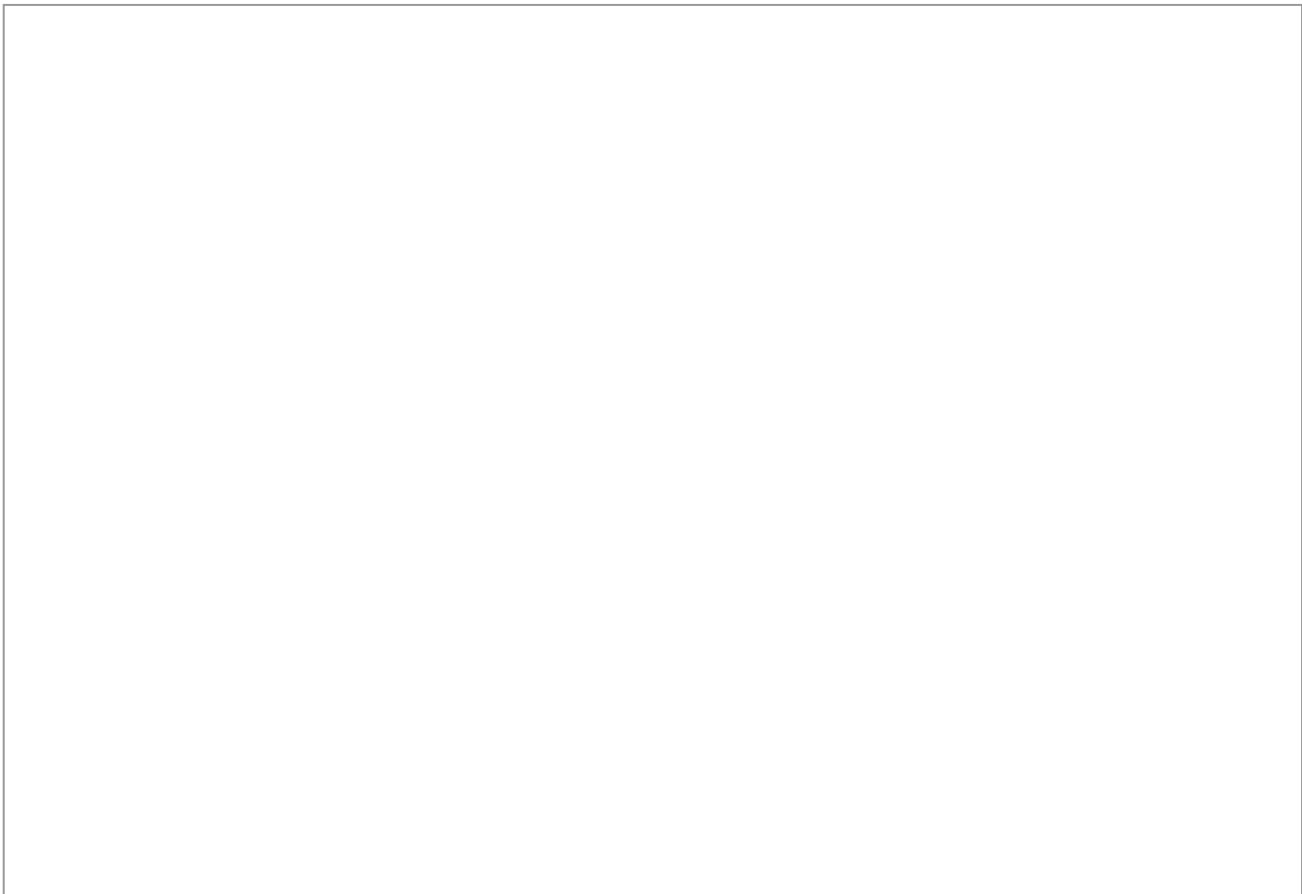
{"content":"Hello,zifangsky!"}
Received: Hello,zifangsky!
```

## 向指定用户发送WebSocket消息并处理对方不在线的情况

给指定用户发送消息：

- 如果接收者在线，则直接发送消息；
- 否则将消息存储到redis，等用户上线后主动拉取未读消息。

### (1) 自定义HandshakeInterceptor，用于禁止未登录用户连接WebSocket：



```
1 package cn.zifangsky.stompwebsocket.interceptor.websocket;
2
3 import cn.zifangsky.stompwebsocket.common.Constants;
4 import cn.zifangsky.stompwebsocket.common.SpringContextUtils;
5 import cn.zifangsky.stompwebsocket.model.User;
6 import org.slf4j.Logger;
7 import org.slf4j.LoggerFactory;
8 import org.springframework.http.server.ServerHttpRequest;
9 import org.springframework.http.server.ServerHttpResponse;
10 import org.springframework.stereotype.Component;
11 import org.springframework.web.socket.WebSocketHandler;
12 import org.springframework.web.socket.server.HandshakeInterceptor;
13
14 import javax.servlet.http.HttpSession;
15 import java.text.MessageFormat;
16 import java.util.Map;
17
18 /**
19  * 自定义{@link org.springframework.web.socket.server.HandshakeInterceptor}，实现“需要登录”
20  *
21  * @author zifangsky
22  * @date 2018/10/11
23  * @since 1.0.0
24  */
25 @Component
26 public class AuthHandshakeInterceptor implements HandshakeInterceptor {
27     private final Logger logger = LoggerFactory.getLogger(getClass());
28
29     @Override
30     public boolean beforeHandshake(ServerHttpRequest serverHttpRequest, ServerHttpResponse serverHttpResponse,
31         HttpSession session = SpringContextUtils.getSession();
32         User loginUser = (User) session.getAttribute(Constants.SESSION_USER);
33
34         if(loginUser != null){
35             logger.debug(MessageFormat.format("用户{0}请求建立WebSocket连接", loginUser.getUsername()));
36             return true;
37         }else{
38             logger.error("未登录系统，禁止连接WebSocket");
39             return false;
40         }
41     }
42
43     @Override
44     public void afterHandshake(ServerHttpRequest serverHttpRequest, ServerHttpResponse serverHttpResponse,
45         WebSocketHandler webSocketHandler, Map<String, Object> attributes) throws Exception {
46
47     }
48
49 }
```

## (2) 自定义HandshakeHandler，用于在建立WebSocket的时候使用自定义的Principal：

```
1 package cn.zifangsky.stompwebsocket.interceptor.websocket;
2
3 import cn.zifangsky.stompwebsocket.common.Constants;
4 import cn.zifangsky.stompwebsocket.common.SpringContextUtils;
5 import cn.zifangsky.stompwebsocket.model.User;
6 import org.slf4j.Logger;
7 import org.slf4j.LoggerFactory;
8 import org.springframework.http.server.ServerHttpRequest;
9 import org.springframework.stereotype.Component;
10 import org.springframework.web.socket.WebSocketHandler;
11 import org.springframework.web.socket.server.support.DefaultHandshakeHandler;
12
13 import javax.servlet.http.HttpSession;
14 import java.security.Principal;
15 import java.text.MessageFormat;
16 import java.util.Map;
17
18 /**
19  * 自定义{@link org.springframework.web.socket.server.support.DefaultHandshakeHandler},
20  *
21  * @author zifangsky
22  * @date 2018/10/11
23  * @since 1.0.0
24  */
25 @Component
26 public class MyHandshakeHandler extends DefaultHandshakeHandler{
27     private final Logger logger = LoggerFactory.getLogger(getClass());
28
29     @Override
30     protected Principal determineUser(ServerHttpRequest request, WebSocketHandler wsHandler,
31         HttpSession session = SpringContextUtils.getSession();
32         User loginUser = (User) session.getAttribute(Constants.SESSION_USER);
33
34         if(loginUser != null){
35             logger.debug(MessageFormat.format("WebSocket连接开始创建Principal, 用户: {0}",
36                 loginUser.getUsername()));
37             return new MyPrincipal(loginUser.getUsername());
38         }else{
39             logger.error("未登录系统, 禁止连接WebSocket");
40             return null;
41         }
42     }
43 }
```

相应地，这里的 MyPrincipal 继承了 java.security.Principal 类：

```
1 package cn.zifangsky.stompwebsocket.interceptor.websocket;
2
3 import java.security.Principal;
4
5 /**
6  * 自定义{@link java.security.Principal}
7  *
8  * @author zifangsky
9  * @date 2018/10/11
10  * @since 1.0.0
11  */
12 public class MyPrincipal implements Principal {
13     private String loginName;
14
15     public MyPrincipal(String loginName) {
16         this.loginName = loginName;
17     }
18
19     @Override
20     public String getName() {
21         return loginName;
22     }
23 }
```

### (3) 自定义ChannelInterceptor，用于在用户断开连接的时候记录日志：

```
1 package cn.zifangsky.stompwebsocket.interceptor.websocket;
2
3 import org.apache.commons.lang3.StringUtils;
4 import org.slf4j.Logger;
5 import org.slf4j.LoggerFactory;
6 import org.springframework.messaging.Message;
7 import org.springframework.messaging.MessageChannel;
8 import org.springframework.messaging.simp.stomp.StompCommand;
9 import org.springframework.messaging.simp.stomp.StompHeaderAccessor;
10 import org.springframework.messaging.support.ChannelInterceptor;
11 import org.springframework.stereotype.Component;
12
13 import java.security.Principal;
14 import java.text.MessageFormat;
15
16 /**
17  * 自定义{@link org.springframework.messaging.support.ChannelInterceptor}，实现断开连接的
18  *
19  * @author zifangsky
20  * @date 2018/10/10
21  * @since 1.0.0
22  */
23 @Component
24 public class MyChannelInterceptor implements ChannelInterceptor{
25     private final Logger logger = LoggerFactory.getLogger(getClass());
26
27     @Override
28     public void afterSendCompletion(Message<?> message, MessageChannel channel, boolean
29         StompHeaderAccessor accessor = StompHeaderAccessor.wrap(message);
30         StompCommand command = accessor.getCommand();
31
32         //用户已经断开连接
33         if(StompCommand.DISCONNECT.equals(command)){
34             String user = "";
35             Principal principal = accessor.getUser();
36             if(principal != null && StringUtils.isNotBlank(principal.getName())){
37                 user = principal.getName();
38             }else{
39                 user = accessor.getSessionId();
40             }
41
42             logger.debug(MessageFormat.format("用户{0}的WebSocket连接已经断开", user));
43         }
44     }
45 }
46 }
```

#### (4) WebSocket相关的完整配置：



```
1 package cn.zifangsky.stompwebsocket.config;
2
3 import cn.zifangsky.stompwebsocket.interceptor.websocket.AuthHandshakeInterceptor;
4 import cn.zifangsky.stompwebsocket.interceptor.websocket.MyChannelInterceptor;
5 import cn.zifangsky.stompwebsocket.interceptor.websocket.MyHandshakeHandler;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.context.annotation.Configuration;
8 import org.springframework.messaging.simp.config.ChannelRegistration;
9 import org.springframework.messaging.simp.config.MessageBrokerRegistry;
10 import org.springframework.web.socket.config.annotation.EnableWebSocketMessageBroker;
11 import org.springframework.web.socket.config.annotation.StompEndpointRegistry;
12 import org.springframework.web.socket.config.annotation.WebSocketMessageBrokerConfigurer;
13
14 /**
15  * WebSocket相关配置
16  *
17  * @author zifangsky
18  * @date 2018/9/30
19  * @since 1.0.0
20  */
21 @Configuration
22 @EnableWebSocketMessageBroker
23 public class WebSocketConfig implements WebSocketMessageBrokerConfigurer{
24     @Autowired
25     private AuthHandshakeInterceptor authHandshakeInterceptor;
26
27     @Autowired
28     private MyHandshakeHandler myHandshakeHandler;
29
30     @Autowired
31     private MyChannelInterceptor myChannelInterceptor;
32
33     @Override
34     public void registerStompEndpoints(StompEndpointRegistry registry) {
35         registry.addEndpoint("/stomp-websocket").withSockJS();
36
37         registry.addEndpoint("/chat-websocket")
38             .addInterceptors(authHandshakeInterceptor)
39             .setHandshakeHandler(myHandshakeHandler)
40             .withSockJS();
41     }
42
43     @Override
44     public void configureMessageBroker(MessageBrokerRegistry registry) {
45         //客户端需要把消息发送到/message/xxx地址
46         registry.setApplicationDestinationPrefixes("/message");
47         //服务端广播消息的路径前缀，客户端需要相应订阅/topic/yyy这个消息
48         registry.enableSimpleBroker("/topic");
49     }
50
51     @Override
52     public void configureClientInboundChannel(ChannelRegistration registration) {
53         registration.interceptors(myChannelInterceptor);
54     }
55
56 }
```

## (5) Controller中的消息处理如下：

```
1 package cn.zifangsky.stompwebsocket.controller;
2
3 import cn.zifangsky.stompwebsocket.common.Constants;
4 import cn.zifangsky.stompwebsocket.common.SpringContextUtils;
5 import cn.zifangsky.stompwebsocket.enums.ExpireEnum;
6 import cn.zifangsky.stompwebsocket.model.User;
7 import cn.zifangsky.stompwebsocket.model.websocket.Greeting;
8 import cn.zifangsky.stompwebsocket.model.websocket.HelloMessage;
9 import cn.zifangsky.stompwebsocket.service.RedisService;
10 import cn.zifangsky.stompwebsocket.utils.JsonUtils;
11 import org.apache.commons.lang3.StringUtils;
12 import org.slf4j.Logger;
13 import org.slf4j.LoggerFactory;
14 import org.springframework.beans.factory.annotation.Autowired;
15 import org.springframework.messaging.simp.SimpMessagingTemplate;
16 import org.springframework.messaging.simp.user.SimpUser;
17 import org.springframework.messaging.simp.user.SimpUserRegistry;
18 import org.springframework.stereotype.Controller;
19 import org.springframework.web.bind.annotation.PostMapping;
20 import org.springframework.web.bind.annotation.RequestBody;
21 import org.springframework.web.bind.annotation.RequestMapping;
22 import org.springframework.web.bind.annotation.ResponseBody;
23
24 import javax.annotation.Resource;
25 import javax.servlet.http.HttpServletRequest;
26 import javax.servlet.http.HttpSession;
27 import java.text.MessageFormat;
28 import java.util.HashMap;
29 import java.util.List;
30 import java.util.Map;
31
32 /**
33  * 测试{@link org.springframework.messaging.simp.SimpMessagingTemplate}类的基本用法
34  * @author zifangsky
35  * @date 2018/10/10
36  * @since 1.0.0
37  */
38 @Controller
39 @RequestMapping("/wsTemplate")
40 public class MessageTemplateController {
41     private final Logger logger = LoggerFactory.getLogger(getClass());
42
43     @Autowired
44     private SimpMessagingTemplate messagingTemplate;
45
46     @Autowired
47     private SimpUserRegistry userRegistry;
48
49     @Resource(name = "redisServiceImpl")
50     private RedisService redisService;
51
52     /**
53      * 简单测试SimpMessagingTemplate的用法
54      */
55     @PostMapping("/greeting")
56     @ResponseBody
```

```
57 public String greeting(@RequestBody Greeting greeting) {
58     this.messagingTemplate.convertAndSend("/topic/greeting", new HelloMessage("He
59
60     return "ok";
61 }
62
63 /**
64  * 给指定用户发送WebSocket消息
65  */
66 @PostMapping("/sendToUser")
67 @ResponseBody
68 public String chat(HttpServletRequest request) {
69     //消息接收者
70     String receiver = request.getParameter("receiver");
71     //消息内容
72     String msg = request.getParameter("msg");
73     HttpSession session = SpringContextUtils.getSession();
74     User loginUser = (User) session.getAttribute(Constants.SESSION_USER);
75
76     HelloMessage resultData = new HelloMessage(MessageFormat.format("{0} say: {1}
77     this.sendToUser(loginUser.getUsername(), receiver, "/topic/reply", JsonUtils.
78
79     return "ok";
80 }
81
82 /**
83  * 给指定用户发送消息，并处理接收者不在线的情况
84  * @param sender 消息发送者
85  * @param receiver 消息接收者
86  * @param destination 目的地
87  * @param payload 消息正文
88  */
89 private void sendToUser(String sender, String receiver, String destination, Strin
90     SimpUser simpUser = userRegistry.getUser(receiver);
91
92     //如果接收者存在，则发送消息
93     if(simpUser != null && StringUtils.isNotBlank(simpUser.getName())){
94         this.messagingTemplate.convertAndSendToUser(receiver, destination, payload
95     }
96     //否则将消息存储到redis，等用户上线后主动拉取未读消息
97     else{
98         //存储消息的Redis列表名
99         String listKey = Constants.REDIS_UNREAD_MSG_PREFIX + receiver + ":" + des
100         logger.info(MessageFormat.format("消息接收者{0}还未建立WebSocket连接, {1}发送
101
102         //存储消息到Redis中
103         redisService.addToListRight(listKey, ExpireEnum.UNREAD_MSG, payload);
104     }
105
106 }
107
108
109 /**
110  * 拉取指定监听路径的未读的WebSocket消息
111  * @param destination 指定监听路径
112  * @return java.util.Map<java.lang.String,java.lang.Object>
113  */
114 @PostMapping("/pullUnreadMessage")
```

```
115 @ResponseBody
116 public Map<String, Object> pullUnreadMessage(String destination){
117     Map<String, Object> result = new HashMap<>();
118     try {
119         HttpSession session = SpringContextUtils.getSession();
120         //当前登录用户
121         User loginUser = (User) session.getAttribute(Constants.SESSION_USER);
122
123         //存储消息的Redis列表名
124         String listKey = Constants.REDIS_UNREAD_MSG_PREFIX + loginUser.getUserName();
125         //从Redis中拉取所有未读消息
126         List<Object> messageList = redisService.rangelist(listKey, 0, -1);
127
128         result.put("code", "200");
129         if(messageList !=null && messageList.size() > 0){
130             //删除Redis中的这个未读消息列表
131             redisService.delete(listKey);
132             //将数据添加到返回集，供前台页面展示
133             result.put("result", messageList);
134         }
135     }catch (Exception e){
136         result.put("code", "500");
137         result.put("msg", e.getMessage());
138     }
139
140     return result;
141 }
142
143 }
```

注：这里对应的几个Redis操作的方法如下：

```

1  @Override
2  public boolean delete(String key) {
3      return redisTemplate.delete(key);
4  }
5
6  @Override
7  public void addToListLeft(String listKey, ExpireEnum expireEnum, Object... values) {
8      //绑定操作
9      BoundListOperations<String, Object> boundValueOperations = redisTemplate.boundListOperations(listKey);
10     //插入数据
11     boundValueOperations.leftPushAll(values);
12     //设置过期时间
13     boundValueOperations.expire(expireEnum.getTime(), expireEnum.getTimeUnit());
14 }
15
16 @Override
17 public void addToListRight(String listKey, ExpireEnum expireEnum, Object... values) {
18     //绑定操作
19     BoundListOperations<String, Object> boundValueOperations = redisTemplate.boundListOperations(listKey);
20     //插入数据
21     boundValueOperations.rightPushAll(values);
22     //设置过期时间
23     boundValueOperations.expire(expireEnum.getTime(), expireEnum.getTimeUnit());
24 }
25
26 @Override
27 public List<Object> rangeList(String listKey, long start, long end) {
28     //绑定操作
29     BoundListOperations<String, Object> boundValueOperations = redisTemplate.boundListOperations(listKey);
30     //查询数据
31     return boundValueOperations.range(start, end);
32 }

```

## (6) 示例页面:

```

1  <html xmlns:th="http://www.thymeleaf.org">
2  <head>
3      <meta content="text/html; charset=UTF-8"/>
4      <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
5      <meta http-equiv="X-UA-Compatible" content="IE=edge"/>
6      <meta name="viewport" content="width=device-width, initial-scale=1"/>
7      <title>Chat With STOMP Message</title>
8      <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
9      <script src="https://cdnjs.cloudflare.com/ajax/libs/sockjs-client/1.1.4/sockjs.min.js"></script>
10     <script src="https://cdnjs.cloudflare.com/ajax/libs/stomp.js/2.3.3/stomp.min.js"></script>
11     <script th:src="@{/layui/layui.js}"></script>
12     <script th:src="@{/layui/lay/modules/layer.js}"></script>
13     <link th:href="@{/layui/css/layui.css}" rel="stylesheet">
14     <link th:href="@{/layui/css/modules/layer/default/layer.css}" rel="stylesheet">
15     <link th:href="@{/css/style.css}" rel="stylesheet">
16     <style type="text/css">
17         #connect-container {
18             margin: 0 auto;
19             width: 400px;
20         }
21
22         #connect-container div {
23             padding: 5px;

```

```
24     margin: 0 7px 10px 0;
25 }
26
27 .message input {
28     padding: 5px;
29     margin: 0 7px 10px 0;
30 }
31
32 .layui-btn {
33     display: inline-block;
34 }
35 </style>
36 <script type="text/javascript">
37     var stompClient = null;
38
39     $(function () {
40         var target = $("#target");
41         if (window.location.protocol === 'http:') {
42             target.val('http://' + window.location.host + target.val());
43         } else {
44             target.val('https://' + window.location.host + target.val());
45         }
46     });
47
48     function setConnected(connected) {
49         var connect = $("#connect");
50         var disconnect = $("#disconnect");
51         var echo = $("#echo");
52
53         if (connected) {
54             connect.addClass("layui-btn-disabled");
55             disconnect.removeClass("layui-btn-disabled");
56             echo.removeClass("layui-btn-disabled");
57         } else {
58             connect.removeClass("layui-btn-disabled");
59             disconnect.addClass("layui-btn-disabled");
60             echo.addClass("layui-btn-disabled");
61         }
62
63         connect.attr("disabled", connected);
64         disconnect.attr("disabled", !connected);
65         echo.attr("disabled", !connected);
66     }
67
68     //连接
69     function connect() {
70         var target = $("#target").val();
71
72         var ws = new SockJS(target);
73         stompClient = Stomp.over(ws);
74
75         stompClient.connect({}, function () {
76             setConnected(true);
77             log('Info: STOMP connection opened.');
```

//连接成功后，主动拉取未读消息

```
80             pullUnreadMessage("/topic/reply");
81         });
82     }
83 }
```

```
82 //订阅服务端的/topic/reply地址
83 stompClient.subscribe("/user/topic/reply", function (response) {
84     log(JSON.parse(response.body).content);
85 })
86 },function () {
87     //断开处理
88     setConnected(false);
89     log('Info: STOMP connection closed.');
```

```
90 });
91 }
92
93 //断开连接
94 function disconnect() {
95     if (stompClient != null) {
96         stompClient.disconnect();
97         stompClient = null;
98     }
99     setConnected(false);
100     log('Info: STOMP connection closed.');
```

```
101 }
102
103 //向指定用户发送消息
104 function sendMessage() {
105     if (stompClient != null) {
106         var receiver = $("#receiver").val();
107         var msg = $("#message").val();
108         log('Sent: ' + JSON.stringify({'receiver': receiver, 'msg':msg}));
109
110         $.ajax({
111             url: "/wsTemplate/sendToUser",
112             type: "POST",
113             dataType: "json",
114             async: true,
115             data: {
116                 "receiver": receiver,
117                 "msg": msg
118             },
119             success: function (data) {
120
121             }
122         });
123     } else {
124         layer.msg('STOMP connection not established, please connect.', {
125             offset: 'auto'
126             ,icon: 2
127         });
128     }
129 }
130
131 //从服务器拉取未读消息
132 function pullUnreadMessage(destination) {
133     $.ajax({
134         url: "/wsTemplate/pullUnreadMessage",
135         type: "POST",
136         dataType: "json",
137         async: true,
138         data: {
139             "destination": destination
```

```

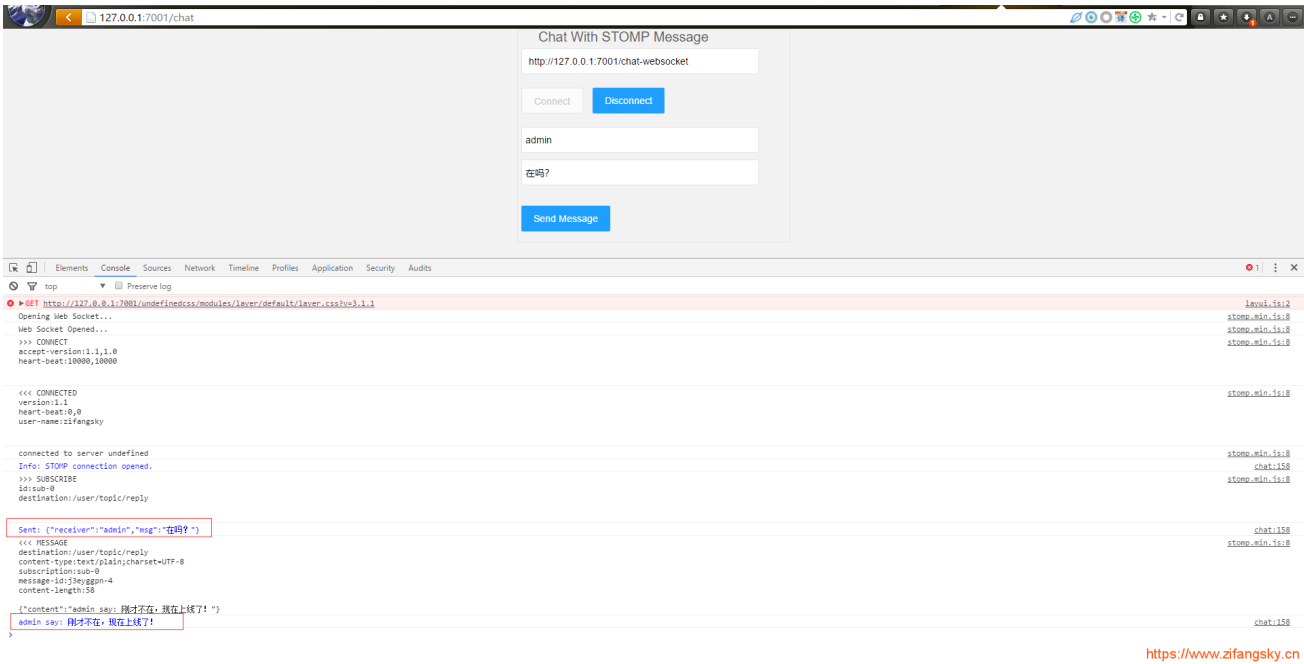
140     },
141     success: function (data) {
142         if (data.result != null) {
143             $.each(data.result, function (i, item) {
144                 log(JSON.parse(item).content);
145             })
146         } else if (data.code != null && data.code == "500") {
147             layer.msg(data.msg, {
148                 offset: 'auto'
149                 , icon: 2
150             });
151         }
152     }
153 });
154 }
155
156 //日志输出
157 function log(message) {
158     console.debug(message);
159 }
160 </script>
161 </head>
162 <body>
163     <noscript><h2 style="color: #ff0000">Seems your browser doesn't support Javascript
164     enabled. Please enable
165     Javascript and reload this page!</h2></noscript>
166     <div>
167         <div id="connect-container" class="layui-elem-field">
168             <legend>Chat With STOMP Message</legend>
169             <div>
170                 <input id="target" type="text" class="layui-input" size="40" style="width: 100%; border: 1px solid #ccc;" />
171             </div>
172             <div>
173                 <button id="connect" class="layui-btn layui-btn-normal" onclick="connect();" >Connect</button>
174                 <button id="disconnect" class="layui-btn layui-btn-normal layui-btn-disabled" onclick="disconnect();" >Disconnect</button>
175             </div>
176         </div>
177         <div class="message">
178             <input id="receiver" type="text" class="layui-input" size="40" style="width: 100%; border: 1px solid #ccc;" />
179             <input id="message" type="text" class="layui-input" size="40" style="width: 100%; border: 1px solid #ccc;" />
180         </div>
181         <div>
182             <button id="echo" class="layui-btn layui-btn-normal layui-btn-disabled" onclick="sendMessage();" >Send Message</button>
183         </div>
184     </div>
185 </body>
186 </html>

```

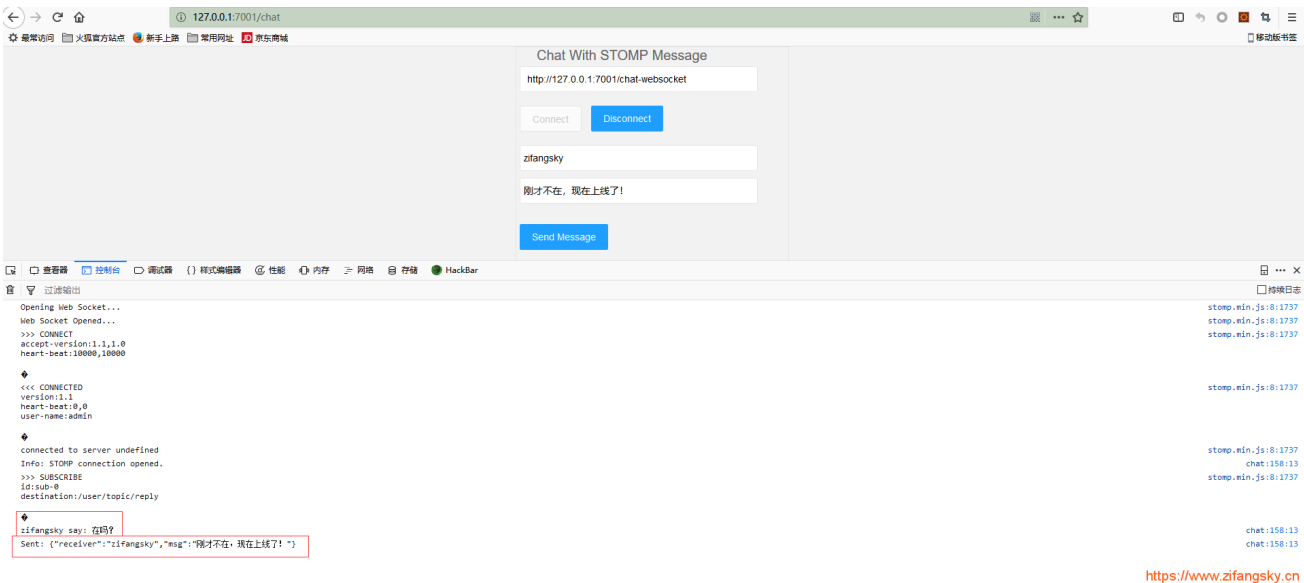
启动项目后，分别在两个浏览器中使用不同的账号登录，接着互相给对方发送消息，效果如下：

界面一：





## 界面二：



👍 赞 (16)

#Spring Boot (<https://www.zifangsky.cn/tag/spring-boot>) #WebSocket (<https://www.zifangsky.cn/tag/websocket>)

©版权声明：原创作品，允许转载，转载时请务必以超链接形式标明文章 原始出处 (<https://www.zifangsky.cn/1359.html>)、作者信息和本声明。否则将追究法律责任。

转载请注明来源：Spring Boot中使用WebSocket总结（二）：向指定用户发送WebSocket消息并对方不在线的情况 (<https://www.zifangsky.cn/1359.html>) - zifangsky的个人博客 (<https://www.zifangsky.cn>)

上一篇 (<https://www.zifangsky.cn/1355.html>)

下一篇 (<https://www.zifangsky.cn/1364.html>)

你可能也喜欢：

- 如何在普通Spring项目中手动实现类似Spring Boot中有条件生成Bean?  
(<https://www.zifangsky.cn/1416.html>)
- 基于Spring的项目中Redis存储对象使用Jackson序列化方式 (<https://www.zifangsky.cn/1366.html>)
- Spring Boot中使用WebSocket总结（三）：使用消息队列实现分布式WebSocket  
(<https://www.zifangsky.cn/1364.html>)
- Spring Boot中使用WebSocket总结（一）：几种实现方式详解 (<https://www.zifangsky.cn/1355.html>)
- 在Spring Boot中使用Spring Data Redis实现基于“发布/订阅”模型的消息队列  
(<https://www.zifangsky.cn/1347.html>)

## 发表评论

来都来了，何不留个足迹~



昵称

\*



邮箱

\*



网址




验证码 \*

发表评论

## 友情链接

iceH's Blog (<http://www.secice.cn>) 业余草 (<http://www.xttblog.com/>) 仲威的博客 (<https://www.blogme.top>)  
俄罗斯方块 (<https://sale.hacker.bid/>) 六阿哥博客 (<https://blog.6ag.cn>)  
太空船博客 (<https://www.boatsky.com/>) 掘金专栏 (<https://juejin.im/user/5819f202d203090055df470e>)  
朴实的追梦者 (<http://www.zmzblog.com>) 青木（简书） (<https://www.jianshu.com/u/cedd62e70951>)

Copyright © 2018 zifangsky的个人博客 (<https://www.zifangsky.cn/>) | 

([https://www.cnzz.com/stat/website.php?web\\_id=1256860929](https://www.cnzz.com/stat/website.php?web_id=1256860929)) | Theme By Specs (<http://9iphp.com>)