脅首页 (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(三): 使用消息队列实现分布式WebSocket

- admin (https://www.zifangsky.cn/author/zifangskyw)

在上一篇文章(https://www.zifangsky.cn/1359.html (https://www.zifangsky.cn/1359.html))中我介绍了服务端如何给指定用户的客户端发送消息,并如何处理对方不在线的情况。在这篇文章中我们继续思考另外一个重要的问题,那就是:如果我们的项目是分布式环境,登录的用户被Nginx的反向代理分配到多个不同服务器,那么在其中一个服务器建立了WebSocket连接的用户如何给在另外一个服务器上建立了WebSocket连接的用户发送消息呢?

其实,要解决这个问题就需要实现分布式WebSocket,而分布式WebSocket一般可以通过以下两种方案来实现:

- 方案一:将消息(<用户id,消息内容>)统一推送到一个消息队列(Redis、Kafka等)的的topic,然后每个应用节点都订阅这个topic,在接收到WebSocket消息后取出这个消息的"**消息接收者的用户ID/用户名**",然后再比对自身是否存在相应用户的连接,如果存在则推送消息,否则丢弃接收到的这个消息(这个消息接收者所在的应用节点会处理)
- 方案二:在用户建立WebSocket连接后,使用Redis缓存记录用户的WebSocket建立在哪个应用节点上,然后同样使用消息队列将消息推送到接收者所在的应用节点上面(实现上比方案一要复杂,但是网络流量会更低)

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

在下面的示例中, 我将根据相对简单的方案一来是实现, 具体实现方式如下:

# (1) 定义一个WebSocket Channel枚举类:

- 1 package cn.zifangsky.mqwebsocket.enums;
- 2
- 3 import org.apache.commons.lang3.StringUtils;

```
4
5
6
    * WebSocket Channel枚举类
7
    * @author zifangsky
8
9
    * @date 2018/10/16
    * @since 1.0.0
10
11
12
   public enum WebSocketChannelEnum {
13
       //测试使用的简易点对点聊天
14
       CHAT("CHAT", "测试使用的简易点对点聊天", "/topic/reply");
15
       WebSocketChannelEnum(String code, String description, String subscribeUrl) {
16
17
           this.code = code;
18
           this.description = description;
           this.subscribeUrl = subscribeUrl;
19
20
       }
21
       /**
22
        * 唯一CODE
23
24
25
       private String code;
26
27
        * 描述
28
29
       private String description;
30
31
        * WebSocket客户端订阅的URL
32
33
       private String subscribeUrl;
34
35
       public String getCode() {
36
           return code;
37
       }
38
       public String getDescription() {
39
40
           return description;
       }
41
42
43
       public String getSubscribeUrl() {
44
           return subscribeUrl;
45
       }
46
47
       /**
        * 通过CODE查找枚举类
48
49
50
       public static WebSocketChannelEnum fromCode(String code){
51
           if(StringUtils.isNoneBlank(code)){
52
                for(WebSocketChannelEnum channelEnum : values()){
53
                    if(channelEnum.code.equals(code)){
54
                        return channelEnum;
55
                    }
56
                }
57
           }
58
59
           return null;
60
       }
```

62 }

# (2) 配置基于Redis的消息队列:

关于Redis实现的消息队列可以参考我之前的这篇文章: https://www.zifangsky.cn/1347.html (https://www.zifangsky.cn/1347.html)

需要注意的是,在大中型正式项目中并不推荐使用Redis实现的消息队列,因为经过测试它并不是特别可靠,所以应该考虑使用 Kafka 、 rabbitMQ 等专业的消息队列中间件(PS: 据说Redis 5.0全新的数据结构 Streams 极大增强了Redis的消息队列功能?)

```
package cn.zifangsky.mqwebsocket.config;
1
2
3
    import cn.zifangsky.mgwebsocket.mg.MessageReceiver;
4
    import com.fasterxml.jackson.annotation.JsonAutoDetect;
5
    import com.fasterxml.jackson.annotation.PropertyAccessor;
6
    import com.fasterxml.jackson.databind.ObjectMapper;
7
    import org.springframework.beans.factory.annotation.Autowired;
8
    import org.springframework.beans.factory.annotation.Value;
9
    import org.springframework.boot.autoconfigure.condition.ConditionalOnClass;
10
    import org.springframework.context.annotation.Bean;
11
    import org.springframework.context.annotation.Configuration;
    import org.springframework.data.redis.connection.RedisClusterConfiguration;
12
13
    import org.springframework.data.redis.connection.RedisConnectionFactory;
    import org.springframework.data.redis.connection.jedis.JedisConnectionFactory;
14
15
    import org.springframework.data.redis.core.RedisTemplate;
    import org.springframework.data.redis.listener.PatternTopic;
16
    import org.springframework.data.redis.listener.RedisMessageListenerContainer;
17
    import org.springframework.data.redis.listener.adapter.MessageListenerAdapter;
18
    import org.springframework.data.redis.serializer.Jackson2JsonRedisSerializer;
19
20
    import org.springframework.data.redis.serializer.StringRedisSerializer;
    import redis.clients.jedis.JedisCluster;
21
22
    import redis.clients.jedis.JedisPoolConfig;
23
24
    import java.util.Arrays;
25
26
27
    * Redis相关配置
28
29
    * @author zifangsky
    * @date 2018/7/30
30
     * @since 1.0.0
31
32
    */
33
    @Configuration
34
    @ConditionalOnClass({JedisCluster.class})
35
    public class RedisConfig {
36
37
        @Value("${spring.redis.timeout}")
38
        private String timeOut;
39
40
        @Value("${spring.redis.cluster.nodes}")
        private String nodes;
41
42
43
        @Value("${spring.redis.cluster.max-redirects}")
        private int maxRedirects;
```

```
45
46
         @Value("${spring.redis.jedis.pool.max-active}")
47
         private int maxActive;
48
49
         @Value("${spring.redis.jedis.pool.max-wait}")
         private int maxWait;
50
51
52
         @Value("${spring.redis.jedis.pool.max-idle}")
53
         private int maxIdle;
54
55
         @Value("${spring.redis.jedis.pool.min-idle}")
56
        private int minIdle;
57
58
         @Value("${spring.redis.message.topic-name}")
59
        private String topicName;
60
61
         @Rean
         public JedisPoolConfig jedisPoolConfig(){
62
63
             JedisPoolConfig config = new JedisPoolConfig();
             config.setMaxTotal(maxActive);
64
             config.setMaxIdle(maxIdle);
65
66
             config.setMinIdle(minIdle);
67
             config.setMaxWaitMillis(maxWait);
68
69
            return config;
70
        }
71
72
         @Bean
         public RedisClusterConfiguration redisClusterConfiguration(){
73
74
             RedisClusterConfiguration configuration = new RedisClusterConfiguration(Array
75
             configuration.setMaxRedirects(maxRedirects);
76
77
             return configuration;
78
        }
79
         /**
80
81
          * JedisConnectionFactory
         */
82
83
         @Bean
         public JedisConnectionFactory jedisConnectionFactory(RedisClusterConfiguration co
84
             return new JedisConnectionFactory(configuration, jedisPoolConfig);
85
86
        }
87
88
         /**
          * 使用Jackson序列化对象
89
         */
90
91
         @Bean
         public Jackson2JsonRedisSerializer<0bject> jackson2JsonRedisSerializer(){
92
             Jackson2JsonRedisSerializer<0bject> serializer = new Jackson2JsonRedisSeriali
93
94
95
             ObjectMapper objectMapper = new ObjectMapper();
             objectMapper.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.AN
96
97
             objectMapper.enableDefaultTyping(ObjectMapper.DefaultTyping.NON_FINAL);
98
             serializer.setObjectMapper(objectMapper);
99
100
             return serializer;
101
        }
102
```

```
/**
103
104
         * RedisTemplate
         */
105
106
        @Bean
        public RedisTemplate<String, Object> redisTemplate(JedisConnectionFactory factory)
107
            RedisTemplate<String, Object> redisTemplate = new RedisTemplate<>();
108
109
            redisTemplate.setConnectionFactory(factory);
110
            //字符串方式序列化KEY
111
            StringRedisSerializer stringRedisSerializer = new StringRedisSerializer();
112
            redisTemplate.setKeySerializer(stringRedisSerializer);
113
114
            redisTemplate.setHashKeySerializer(stringRedisSerializer);
115
116
            //JSON方式序列化VALUE
            redisTemplate.setValueSerializer(jackson2JsonRedisSerializer);
117
            redisTemplate.setHashValueSerializer(jackson2JsonRedisSerializer);
118
119
            redisTemplate.afterPropertiesSet();
120
121
            return redisTemplate;
122
123
        }
124
         /**
125
126
         * 消息监听器
         */
127
128
129
        MessageListenerAdapter messageListenerAdapter(MessageReceiver messageReceiver, Ja
130
            //消息接收者以及对应的默认处理方法
131
            MessageListenerAdapter messageListenerAdapter = new MessageListenerAdapter(me
132
            //消息的反序列化方式
133
            messageListenerAdapter.setSerializer(jackson2JsonRedisSerializer);
134
135
            return messageListenerAdapter;
        }
136
137
        /**
138
139
         * message listener container
         */
140
141
        @Bean
142
        RedisMessageListenerContainer container(RedisConnectionFactory connectionFactory
143
                 , MessageListenerAdapter messageListenerAdapter){
144
            RedisMessageListenerContainer container = new RedisMessageListenerContainer()
            container.setConnectionFactory(connectionFactory);
145
146
            //添加消息监听器
147
            container.addMessageListener(messageListenerAdapter, new PatternTopic(topicNa
148
149
            return container;
150
        }
151
152 | }
```

#### 需要注意的是,这里使用的配置如下所示:

```
spring:
1
2
     . . .
3
     #redis
4
     redis:
5
         cluster:
6
           nodes: namenode22:6379, datanode23:6379, datanode24:6379
7
           max-redirects: 6
8
         timeout: 300000
9
         jedis:
10
           pool:
11
             max-active: 8
12
             max-wait: 100000
             max-idle: 8
13
14
             min-idle: 0
         #自定义的监听的TOPIC路径
15
16
         message:
17
           topic-name: topic-test
```

# (3) 定义一个Redis消息的处理者:

https://www.zifangsky.cn/1364.html

```
package cn.zifangsky.mqwebsocket.mq;
2
3
   import cn.zifangsky.mgwebsocket.enums.WebSocketChannelEnum;
   import cn.zifangsky.mgwebsocket.model.websocket.RedisWebsocketMsg;
   import org.apache.commons.lang3.StringUtils;
5
   import org.slf4j.Logger;
6
7
   import org.slf4j.LoggerFactory;
   import org.springframework.beans.factory.annotation.Autowired;
   import org.springframework.messaging.simp.SimpMessagingTemplate;
9
   import org.springframework.messaging.simp.user.SimpUser;
10
   import org.springframework.messaging.simp.user.SimpUserRegistry;
11
12
   import org.springframework.stereotype.Component;
13
14
   import java.text.MessageFormat;
15
   /**
16
17
    * Redis中的WebSocket消息的处理者
18
19
    * @author zifanaskv
20
    * @date 2018/10/16
    * @since 1.0.0
21
    */
22
23
   @Component
   public class MessageReceiver {
24
25
       private final Logger logger = LoggerFactory.getLogger(getClass());
26
27
       @Autowired
28
       private SimpMessagingTemplate messagingTemplate;
29
30
       @Autowired
       private SimpUserRegistry userRegistry;
31
32
33
34
        * 处理WebSocket消息
35
       public void receiveMessage(RedisWebsocketMsg redisWebsocketMsg) {
36
37
           logger.info(MessageFormat.format("Received Message: {0}", redisWebsocketMsg));
38
           //1. 取出用户名并判断是否连接到当前应用节点的WebSocket
39
           SimpUser simpUser = userRegistry.getUser(redisWebsocketMsg.getReceiver());
40
41
           if(simpUser != null && StringUtils.isNoneBlank(simpUser.getName())){
42
               //2. 获取WebSocket客户端的订阅地址
43
               WebSocketChannelEnum channelEnum = WebSocketChannelEnum.fromCode(redisWebs
44
               if(channelEnum != null){
45
46
                   //3. 给WebSocket客户端发送消息
47
                    messagingTemplate.convertAndSendToUser(redisWebsocketMsg.getReceiver()
48
               }
           }
49
50
51
       }
52 }
```

# (4) 在Controller中发送WebSocket消息:

```
package cn.zifangsky.mqwebsocket.controller;

import cn.zifangsky.mqwebsocket.common.Constants;
```

```
4
    import cn.zifangsky.mqwebsocket.common.SpringContextUtils;
5
    import cn.zifangsky.mqwebsocket.enums.ExpireEnum;
6
    import cn.zifangsky.mgwebsocket.enums.WebSocketChannelEnum;
7
    import cn.zifangsky.mgwebsocket.model.User;
    import cn.zifangsky.mqwebsocket.model.websocket.HelloMessage;
8
9
    import cn.zifangsky.mqwebsocket.model.websocket.RedisWebsocketMsg;
10
    import cn.zifangsky.mqwebsocket.service.RedisService;
    import cn.zifangsky.mqwebsocket.utils.JsonUtils;
11
12
    import org.apache.commons.lang3.StringUtils;
13
    import org.slf4j.Logger;
14
    import org.slf4j.LoggerFactory;
15
    import org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.beans.factory.annotation.Value;
16
17
    import org.springframework.messaging.simp.SimpMessagingTemplate;
    import org.springframework.messaging.simp.user.SimpUser;
18
19
    import org.springframework.messaging.simp.user.SimpUserRegistry;
20
    import org.springframework.stereotype.Controller;
21
    import org.springframework.web.bind.annotation.PostMapping;
22
    import org.springframework.web.bind.annotation.RequestMapping;
23
    import org.springframework.web.bind.annotation.ResponseBody;
24
25
    import javax.annotation.Resource;
26
    import javax.servlet.http.HttpServletRequest;
    import javax.servlet.http.HttpSession;
27
28
    import java.text.MessageFormat;
29
    import java.util.HashMap;
30
    import java.util.List;
31
    import java.util.Map;
32
33
    * 测试{@link org.springframework.messaging.simp.SimpMessagingTemplate}类的基本用法
34
35
    * @author zifangsky
     * @date 2018/10/10
36
37
    * @since 1.0.0
    */
38
39
    @Controller
    @RequestMapping(("/wsTemplate"))
40
41
    public class RedisMessageController {
42
        private final Logger logger = LoggerFactory.getLogger(getClass());
43
44
        @Value("${spring.redis.message.topic-name}")
45
        private String topicName;
46
47
        @Autowired
48
        private SimpMessagingTemplate messagingTemplate;
49
50
        @Autowired
51
        private SimpUserRegistry userRegistry;
52
53
        @Resource(name = "redisServiceImpl")
        private RedisService redisService;
54
55
        /**
56
57
         * 给指定用户发送WebSocket消息
58
        @PostMapping("/sendToUser")
59
60
        @ResponseBody
        public String chat(HttpServletRequest request) {
```

```
62
            //消息接收者
63
            String receiver = request.getParameter("receiver");
64
            //消息内容
65
            String msg = request.getParameter("msg");
66
            HttpSession session = SpringContextUtils.getSession();
            User loginUser = (User) session.getAttribute(Constants.SESSION_USER);
67
68
69
            HelloMessage resultData = new HelloMessage(MessageFormat.format("{0} say: {1}
70
            this.sendToUser(loginUser.getUsername(), receiver, WebSocketChannelEnum.CHAT.
71
            return "ok";
72
73
        }
74
75
76
         * 给指定用户发送消息,并处理接收者不在线的情况
77
         * @param sender 消息发送者
78
         * @param receiver 消息接收者
79
         * @param destination 目的地
80
         * @param payload 消息正文
81
82
        private void sendToUser(String sender, String receiver, String destination, String
83
            SimpUser simpUser = userRegistry.getUser(receiver);
84
            //如果接收者存在,则发送消息
85
86
            if(simpUser != null && StringUtils.isNoneBlank(simpUser.getName())){
87
                messagingTemplate.convertAndSendToUser(receiver, destination, payload);
88
            }
89
            //如果接收者在线,则说明接收者连接了集群的其他节点,需要通知接收者连接的那个节点发送消息
90
            else if(redisService.isSetMember(Constants.REDIS_WEBSOCKET_USER_SET, receiver
91
                RedisWebsocketMsg<String> redisWebsocketMsg = new RedisWebsocketMsg<>(red
92
93
                redisService.convertAndSend(topicName, redisWebsocketMsg);
94
            }
95
            //否则将消息存储到redis,等用户上线后主动拉取未读消息
96
            else{
97
                //存储消息的Redis列表名
                String listKey = Constants.REDIS_UNREAD_MSG_PREFIX + receiver + ":" + des
98
99
                logger.info(MessageFormat.format("消息接收者{0}还未建立WebSocket连接, {1}发送
100
                //存储消息到Redis中
101
102
                redisService.addToListRight(listKey, ExpireEnum.UNREAD_MSG, payload);
103
            }
104
105
        }
106
107
        /**
108
109
         * 拉取指定监听路径的未读的WebSocket消息
         * @param destination 指定监听路径
110
111
         * @return java.util.Map<java.lang.String,java.lang.Object>
112
113
        @PostMapping("/pullUnreadMessage")
114
        @ResponseBody
115
        public Map<String, Object> pullUnreadMessage(String destination){
116
            Map<String, Object> result = new HashMap<>();
117
118
                HttpSession session = SpringContextUtils.getSession();
119
                //当前登录用户
```

```
120
                User loginUser = (User) session.getAttribute(Constants.SESSION_USER);
121
122
                //存储消息的Redis列表名
123
                String listKey = Constants.REDIS_UNREAD_MSG_PREFIX + loginUser.getUsernam
124
                //从Redis中拉取所有未读消息
125
                List<Object> messageList = redisService.rangeList(listKey, 0, -1);
126
                result.put("code", "200");
127
128
                if(messageList !=null && messageList.size() > 0){
129
                    //删除Redis中的这个未读消息列表
130
                    redisService.delete(listKey);
131
                    //将数据添加到返回集,供前台页面展示
132
                    result.put("result", messageList);
133
                }
134
            }catch (Exception e){
                result.put("code", "500");
135
                result.put("msg", e.getMessage());
136
137
            }
138
139
            return result;
140
        }
141
142 }
```

# (5) 其他拦截器处理WebSocket连接相关问题:

i) AuthHandshakeInterceptor:

```
package cn.zifangsky.mqwebsocket.interceptor.websocket;
 2
 3
       import cn.zifangsky.mgwebsocket.common.Constants;
       import cn.zifangsky.mgwebsocket.common.SpringContextUtils;
       import cn.zifangsky.mqwebsocket.model.User;
 5
       import cn.zifangsky.mqwebsocket.service.RedisService;
 6
 7
       import org.apache.commons.lang3.StringUtils;
       import org.slf4j.Logger;
 9
       import org.slf4j.LoggerFactory;
      import org.springframework.http.server.ServerHttpRequest;
10
       import org.springframework.http.server.ServerHttpResponse;
11
12
       import org.springframework.stereotype.Component;
       import org.springframework.web.socket.WebSocketHandler;
13
       import org.springframework.web.socket.server.HandshakeInterceptor;
15
       import javax.annotation.Resource;
16
17
       import javax.servlet.http.HttpSession;
18
       import java.text.MessageFormat;
19
       import java.util.Map;
20
       /**
21
        * 自定义{@link org.springframework.web.socket.server.HandshakeInterceptor}, 实现"需要登录
22
23
24
       * @author zifangsky
25
       * @date 2018/10/11
        * @since 1.0.0
26
27
         */
28
       @Component
       public class AuthHandshakeInterceptor implements HandshakeInterceptor {
29
30
               private final Logger logger = LoggerFactory.getLogger(getClass());
31
               @Resource(name = "redisServiceImpl")
32
33
               private RedisService redisService;
34
35
               @Override
               public boolean beforeHandshake(ServerHttpRequest serverHttpRequest, ServerHttpResp
36
37
                       HttpSession session = SpringContextUtils.getSession();
38
                       User loginUser = (User) session.getAttribute(Constants.SESSION_USER);
39
                       if(redisService.isSetMember(Constants.REDIS_WEBSOCKET_USER_SET, loginUser.getU
40
41
                               logger.error("同一个用户不准建立多个连接WebSocket");
42
                               return false;
43
                       }else if(loginUser == null || StringUtils.isBlank(loginUser.getUsername())){
                               logger.error("未登录系统,禁止连接WebSocket");
44
                               return false;
45
46
                       }else{
47
                               logger.debug(MessageFormat.format("用户{0}请求建立WebSocket连接", loginUser.debug(MessageFormat.format("用户{0}请求建立WebSocket连接", loginUser.debug(MessageFormat.format("用户{0}请求建立WebSocket连接", loginUser.debug(MessageFormat.format("用户{0}请求建立WebSocket连接", loginUser.debug(MessageFormat.format("用户{0}请求建立WebSocket连接", loginUser.debug(MessageFormat.format("用户{0})请求建立WebSocket连接", loginUser.debug(MessageFormat.format("用户{0})请求建立WebSocket连接", loginUser.debug(MessageFormat.format("用户{0})请求建立WebSocket连接", loginUser.debug(MessageFormat.format("用户{0}) 请求建立WebSocket连接", loginUser.debug(MessageFormat.format("用户{0}) 请求建立WebSocket连接", loginUser.debug(MessageFormat.format("用户{0}) 请求建立WebSocket连接", loginUser.debug(MessageFormat.format("用户{0}) in the format in the 
48
                               return true;
                       }
49
50
               }
51
52
               @Override
53
               public void afterHandshake(ServerHttpRequest serverHttpRequest, ServerHttpResponse)
54
55
               }
56
57 | }
```

#### ii) MvHandshakeHandler:

```
package cn.zifangsky.mgwebsocket.interceptor.websocket;
2
3
   import cn.zifangsky.mgwebsocket.common.Constants;
4
   import cn.zifangsky.mgwebsocket.common.SpringContextUtils;
   import cn.zifangsky.mqwebsocket.model.User;
5
6
   import cn.zifangsky.mgwebsocket.service.RedisService;
7
   import org.slf4j.Logger;
   import org.slf4j.LoggerFactory;
9
   import org.springframework.http.server.ServerHttpRequest;
10 import org.springframework.stereotype.Component;
   import org.springframework.web.socket.WebSocketHandler;
11
   import org.springframework.web.socket.server.support.DefaultHandshakeHandler;
12
13
14
   import javax.annotation.Resource;
15 import javax.servlet.http.HttpSession;
   import java.security.Principal;
16
   import java.text.MessageFormat;
17
18
   import java.util.Map;
19
   /**
20
21
    * 自定义{@link org.springframework.web.socket.server.support.DefaultHandshakeHandler},
22
23
    * @author zifangsky
24
    * @date 2018/10/11
25
    * @since 1.0.0
26
    */
27
   @Component
   public class MyHandshakeHandler extends DefaultHandshakeHandler{
28
29
       private final Logger logger = LoggerFactory.getLogger(getClass());
30
31
       @Resource(name = "redisServiceImpl")
32
       private RedisService redisService;
33
34
       @Override
35
       protected Principal determineUser(ServerHttpRequest request, WebSocketHandler wsHd
           HttpSession session = SpringContextUtils.getSession();
36
37
           User loginUser = (User) session.getAttribute(Constants.SESSION_USER);
38
39
           if(loginUser != null){
               logger.debug(MessageFormat.format("WebSocket连接开始创建Principal, 用户: {0}"
40
41
               //1. 将用户名存到Redis中
42
               redisService.addToSet(Constants.REDIS_WEBSOCKET_USER_SET, loginUser.getUse
43
44
               //2. 返回自定义的Principal
45
               return new MyPrincipal(loginUser.getUsername());
46
           }else{
47
               logger.error("未登录系统,禁止连接WebSocket");
               return null;
48
49
           }
       }
50
51
52 | }
```

#### iii) MyChannelInterceptor:

```
package cn.zifangsky.mqwebsocket.interceptor.websocket;
2
3
   import cn.zifangsky.mgwebsocket.common.Constants;
   import cn.zifangsky.mawebsocket.service.RedisService;
   import org.apache.commons.lang3.StringUtils;
5
   import org.slf4j.Logger;
6
7
   import org.slf4j.LoggerFactory;
   import org.springframework.messaging.Message;
9
   import org.springframework.messaging.MessageChannel;
   import org.springframework.messaging.simp.stomp.StompCommand;
10
   import org.springframework.messaging.simp.stomp.StompHeaderAccessor;
11
   import org.springframework.messaging.support.ChannelInterceptor;
   import org.springframework.stereotype.Component;
13
14
15
   import javax.annotation.Resource;
   import java.security.Principal;
16
17
   import java.text.MessageFormat;
18
   /**
19
20
    * 自定义{@link org.springframework.messaging.support.ChannelInterceptor}, 实现断开连接的处
21
    * @author zifangsky
22
    * @date 2018/10/10
23
24
    * @since 1.0.0
25
   @Component
26
   public class MyChannelInterceptor implements ChannelInterceptor{
27
28
       private final Logger logger = LoggerFactory.getLogger(getClass());
29
30
       @Resource(name = "redisServiceImpl")
       private RedisService redisService;
31
32
33
       @Override
34
       public void afterSendCompletion(Message<?> message, MessageChannel channel, booled
35
           StompHeaderAccessor accessor = StompHeaderAccessor.wrap(message);
36
           StompCommand command = accessor.getCommand();
37
           //用户已经断开连接
38
39
           if(StompCommand.DISCONNECT.equals(command)){
               String user = "";
40
41
               Principal principal = accessor.getUser();
42
               if(principal != null && StringUtils.isNoneBlank(principal.getName())){
43
                   user = principal.getName();
44
45
                    //从Redis中移除用户
                    redisService.removeFromSet(Constants.REDIS_WEBSOCKET_USER_SET, user);
46
47
               }else{
48
                   user = accessor.getSessionId();
               }
49
50
51
               logger.debug(MessageFormat.format("用户{0}的WebSocket连接已经断开", user));
52
           }
53
       }
54
55 | }
```

### (6) WebSocket相关配置:

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

# (7)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"/>
        <meta name="viewport" content="width=device-width, initial-scale=1"/>
6
7
        <title>Chat With STOMP Message</title>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"><</pre>
8
        <script src="https://cdnjs.cloudflare.com/ajax/libs/sockjs-client/1.1.4/sockjs.mi</pre>
9
10
        <script src="https://cdnjs.cloudflare.com/ajax/libs/stomp.js/2.3.3/stomp.min.js">
        <script th:src="@{/layui/layui.js}"></script>
11
12
        <script th:src="@{/layui/lay/modules/layer.js}"></script>
13
        <link th:href="@{/layui/css/layui.css}" rel="stylesheet">
        <link th:href="@{/layui/css/modules/layer/default/layer.css}" rel="stylesheet">
14
15
        <link th:href="@{/css/style.css}" rel="stylesheet">
16
        <style type="text/css">
17
            #connect-container {
                margin: 0 auto;
18
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>
        <script type="text/javascript">
36
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
                    target.val('https://' + window.location.host + target.val());
44
45
                }
46
            });
47
48
            function setConnected(connected) {
                var connect = $("#connect");
49
50
                var disconnect = $("#disconnect");
51
                var echo = $("#echo");
52
53
                if (connected) {
                     connect.addClass("layui-btn-disabled");
54
55
                    disconnect.removeClass("layui-btn-disabled");
                     echo.removeClass("layui-btn-disabled");
56
57
                } else {
                     connect.removeClass("layui-btn-disabled");
58
```

```
59
                     disconnect.addClass("layui-btn-disabled");
60
                     echo.addClass("layui-btn-disabled");
                }
61
62
                 connect.attr("disabled", connected);
63
                disconnect.attr("disabled", !connected);
64
65
                echo.attr("disabled", !connected);
            }
66
67
            //连接
68
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.');
78
79
                     //连接成功后,主动拉取未读消息
80
                     pullUnreadMessage("/topic/reply");
81
82
                     //订阅服务端的/topic/reply地址
                     stompClient.subscribe("/user/topic/reply", function (response) {
83
                         log(JSON.parse(response.body).content);
84
85
                     })
                },function () {
86
                     //断开处理
87
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
            //向指定用户发送消息
103
             function sendMessage() {
104
105
                 if (stompClient != null) {
                     var receiver = $("#receiver").val();
106
                     var msg = $("#message").val();
107
108
                     log('Sent: ' + JSON.stringify({'receiver': receiver, 'msg':msg}));
109
110
                     $.ajax({
                         url: "/wsTemplate/sendToUser",
111
                         type: "POST",
112
113
                         dataType: "json",
114
                         async: true,
115
                         data: {
                             "receiver": receiver,
116
```

```
117
                              "msg": msg
118
                          },
119
                          success: function (data) {
120
121
                          }
122
                     });
123
                 } else {
                     layer.msg('STOMP connection not established, please connect.', {
124
125
                          offset: 'auto'
                          ,icon: 2
126
127
                     });
128
                 }
129
             }
130
             //从服务器拉取未读消息
131
132
             function pullUnreadMessage(destination) {
133
                 $.ajax({
134
                     url: "/wsTemplate/pullUnreadMessage",
135
                     type: "POST",
136
                     dataType: "json",
                     async: true,
137
                     data: {
138
139
                          "destination": destination
140
                     },
141
                     success: function (data) {
                          if (data.result != null) {
142
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.msq(data.msq, {
                                  offset: 'auto'
148
                                  ,icon: 2
149
150
                              });
151
                          }
152
                     }
                 });
153
154
             }
155
156
             //日志输出
157
             function log(message) {
158
                 console.debug(message);
159
             }
         </script>
160
161
    </head>
162
     <body>
163
         <noscript><h2 style="color: #ff0000">Seems your browser doesn't support Javascrip
164
             enabled. Please enable
165
             Javascript and reload this page!</h2></noscript>
166
         <div>
             <div id="connect-container" class="layui-elem-field">
167
168
                 <legend>Chat With STOMP Message</legend>
169
170
                      <input id="target" type="text" class="layui-input" size="40" style="w</pre>
171
                 </div>
172
                 <div>
173
                      <button id="connect" class="layui-btn layui-btn-normal" onclick="conn</pre>
                     <button id="disconnect" class="layui-btn layui-btn-normal layui-btn-d</pre>
174
```

```
175
                              onclick="disconnect();">Disconnect
176
                      </button>
177
                 </div>
178
179
                  <div class="message">
180
                      <input id="receiver" type="text" class="layui-input" size="40" style=</pre>
                      <input id="message" type="text" class="layui-input" size="40" style="</pre>
181
182
                  <div>
183
                      <button id="echo" class="layui-btn layui-btn-normal layui-btn-disable</pre>
184
185
                              onclick="sendMessage();">Send Message
                      </button>
186
187
                  </div>
188
             </div>
189
         </div>
190 </body>
191 </html>
```

测试效果省略, 具体效果可以自行在两台不同服务器上面运行示例源码查看。

■ 赞 (9)

#Spring Boot (https://www.zifangsky.cn/tag/spring-boot) #WebSocket (https://www.zifangsky.cn/tag/websocket) #消息队列 (https://www.zifangsky.cn/tag/%e6%b6%88%e6%81%af%e9%98%9f%e5%88%97)

**©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 <u>原始出处</u> (https://www.zifangsky.cn/1364.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(三):使用消息队列实现分布式WebSocket (https://www.zifangsky.cn/1364.html) - zifangsky的个人博客 (https://www.zifangsky.cn)

上一篇 (https://www.zifangsky.cn/1359.html)

下一篇 (https://www.zifangsky.cn/1366.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(二):向指定用户发送WebSocket消息并处理对方不在线的情况 (https://www.zifangsky.cn/1359.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)

#### 本文共 1 个回复



董佩力 2019/09/11 11:16

感谢, 思路有用

(https://www.zifangsky.cn/1364.html?replytocom=6505#respond)

回复

来都	张了,何不留个足迹~
) <b>;</b>	_ <del>- </del>
•	昵称 ————————————————————————————————————
	邮箱
90	网址
7	mm9 ®

## 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) 大空船博客 (https://www.boatsky.com/) 押余专栏 (https://iueiin.im/user/5819f202d203090055df470e)

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) | **请** (https://www.cnzz.com/stat/website.php?web\_id=1256860929) | Theme By Specs (http://9iphp.com)