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

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/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：

```
package cn.zifangsky.mqwebsocket.enums;

import org.apache.commons.lang3.StringUtils;
```

```java

/**
 * WebSocket Channel枚举类
 *
 * @author zifangsky
 * @date 2018/10/16
 * @since 1.0.0
 */
public enum WebSocketChannelEnum {
    //测试使用的简易点对点聊天
    CHAT("CHAT", "测试使用的简易点对点聊天", "/topic/reply");

    WebSocketChannelEnum(String code, String description, String subscribeUrl) {
        this.code = code;
        this.description = description;
        this.subscribeUrl = subscribeUrl;
    }

    /**
     * 唯一CODE
     */
    private String code;
    /**
     * 描述
     */
    private String description;
    /**
     * WebSocket客户端订阅的URL
     */
    private String subscribeUrl;

    public String getCode() {
        return code;
    }

    public String getDescription() {
        return description;
    }

    public String getSubscribeUrl() {
        return subscribeUrl;
    }

    /**
     * 通过CODE查找枚举类
     */
    public static WebSocketChannelEnum fromCode(String code){
        if(StringUtils.isNoneBlank(code)){
            for(WebSocketChannelEnum channelEnum : values()){
                if(channelEnum.code.equals(code)){
                    return channelEnum;
                }
            }
        }

        return null;
    }

}
```

（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？）

```java
package cn.zifangsky.mqwebsocket.config;

import cn.zifangsky.mqwebsocket.mq.MessageReceiver;
import com.fasterxml.jackson.annotation.JsonAutoDetect;
import com.fasterxml.jackson.annotation.PropertyAccessor;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.autoconfigure.condition.ConditionalOnClass;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisClusterConfiguration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.connection.jedis.JedisConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.listener.PatternTopic;
import org.springframework.data.redis.listener.RedisMessageListenerContainer;
import org.springframework.data.redis.listener.adapter.MessageListenerAdapter;
import org.springframework.data.redis.serializer.Jackson2JsonRedisSerializer;
import org.springframework.data.redis.serializer.StringRedisSerializer;
import redis.clients.jedis.JedisCluster;
import redis.clients.jedis.JedisPoolConfig;

import java.util.Arrays;

/**
 * Redis相关配置
 *
 * @author zifangsky
 * @date 2018/7/30
 * @since 1.0.0
 */
@Configuration
@ConditionalOnClass({JedisCluster.class})
public class RedisConfig {

    @Value("${spring.redis.timeout}")
    private String timeOut;

    @Value("${spring.redis.cluster.nodes}")
    private String nodes;

    @Value("${spring.redis.cluster.max-redirects}")
    private int maxRedirects;

    @Value("${spring.redis.jedis.pool.max-active}")
    private int maxActive;

    @Value("${spring.redis.jedis.pool.max-wait}")
    private int maxWait;

    @Value("${spring.redis.jedis.pool.max-idle}")
    private int maxIdle;

    @Value("${spring.redis.jedis.pool.min-idle}")
    private int minIdle;

    @Value("${spring.redis.message.topic-name}")
    private String topicName;

    @Bean
    public JedisPoolConfig jedisPoolConfig(){
        JedisPoolConfig config = new JedisPoolConfig();
        config.setMaxTotal(maxActive);
        config.setMaxIdle(maxIdle);
        config.setMinIdle(minIdle);
        config.setMaxWaitMillis(maxWait);

        return config;
    }

    @Bean
    public RedisClusterConfiguration redisClusterConfiguration(){
        RedisClusterConfiguration configuration = new RedisClusterConfiguration(Array
        configuration.setMaxRedirects(maxRedirects);

        return configuration;
    }

    /**
     * JedisConnectionFactory
     */
    @Bean
    public JedisConnectionFactory jedisConnectionFactory(RedisClusterConfiguration co
        return new JedisConnectionFactory(configuration,jedisPoolConfig);
    }

    /**
     * 使用Jackson序列化对象
     */
    @Bean
    public Jackson2JsonRedisSerializer<Object> jackson2JsonRedisSerializer(){
        Jackson2JsonRedisSerializer<Object> serializer = new Jackson2JsonRedisSeriali

        ObjectMapper objectMapper = new ObjectMapper();
        objectMapper.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.AN
        objectMapper.enableDefaultTyping(ObjectMapper.DefaultTyping.NON_FINAL);
        serializer.setObjectMapper(objectMapper);

        return serializer;
    }

    /**
     * RedisTemplate
     */
    @Bean
    public RedisTemplate<String, Object> redisTemplate(JedisConnectionFactory factory
        RedisTemplate<String, Object> redisTemplate = new RedisTemplate<>();
        redisTemplate.setConnectionFactory(factory);

        //字符串方式序列化KEY
        StringRedisSerializer stringRedisSerializer = new StringRedisSerializer();
        redisTemplate.setKeySerializer(stringRedisSerializer);
        redisTemplate.setHashKeySerializer(stringRedisSerializer);

        //JSON方式序列化VALUE
        redisTemplate.setValueSerializer(jackson2JsonRedisSerializer);
        redisTemplate.setHashValueSerializer(jackson2JsonRedisSerializer);

        redisTemplate.afterPropertiesSet();

        return redisTemplate;
    }

    /**
     * 消息监听器
     */
    @Bean
    MessageListenerAdapter messageListenerAdapter(MessageReceiver messageReceiver, Ja
        //消息接收者以及对应的默认处理方法
        MessageListenerAdapter messageListenerAdapter = new MessageListenerAdapter(me
        //消息的反序列化方式
        messageListenerAdapter.setSerializer(jackson2JsonRedisSerializer);

        return messageListenerAdapter;
    }

    /**
     * message listener container
     */
    @Bean
    RedisMessageListenerContainer container(RedisConnectionFactory connectionFactory
            , MessageListenerAdapter messageListenerAdapter){
        RedisMessageListenerContainer container = new RedisMessageListenerContainer()
        container.setConnectionFactory(connectionFactory);
        //添加消息监听器
        container.addMessageListener(messageListenerAdapter, new PatternTopic(topicNa

        return container;
    }

}
```

需要注意的是，这里使用的配置如下所示：

```yaml
spring:
  ...
  #redis
  redis:
      cluster:
        nodes: namenode22:6379,datanode23:6379,datanode24:6379
        max-redirects: 6
      timeout: 300000
      jedis:
        pool:
          max-active: 8
          max-wait: 100000
          max-idle: 8
          min-idle: 0
      #自定义的监听的TOPIC路径
      message:
        topic-name: topic-test
```

（3）定义一个Redis消息的处理者：

```java
package cn.zifangsky.mqwebsocket.mq;

import cn.zifangsky.mqwebsocket.enums.WebSocketChannelEnum;
import cn.zifangsky.mqwebsocket.model.websocket.RedisWebsocketMsg;
import org.apache.commons.lang3.StringUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.messaging.simp.SimpMessagingTemplate;
import org.springframework.messaging.simp.user.SimpUser;
import org.springframework.messaging.simp.user.SimpUserRegistry;
import org.springframework.stereotype.Component;

import java.text.MessageFormat;

/**
 * Redis中的WebSocket消息的处理者
 *
 * @author zifangsky
 * @date 2018/10/16
 * @since 1.0.0
 */
@Component
public class MessageReceiver {
    private final Logger logger = LoggerFactory.getLogger(getClass());

    @Autowired
    private SimpMessagingTemplate messagingTemplate;

    @Autowired
    private SimpUserRegistry userRegistry;

    /**
     * 处理WebSocket消息
     */
    public void receiveMessage(RedisWebsocketMsg redisWebsocketMsg) {
        logger.info(MessageFormat.format("Received Message: {0}", redisWebsocketMsg));
        //1. 取出用户名并判断是否连接到当前应用节点的WebSocket
        SimpUser simpUser = userRegistry.getUser(redisWebsocketMsg.getReceiver());

        if(simpUser != null && StringUtils.isNoneBlank(simpUser.getName())){
            //2. 获取WebSocket客户端的订阅地址
            WebSocketChannelEnum channelEnum = WebSocketChannelEnum.fromCode(redisWebs

            if(channelEnum != null){
                //3. 给WebSocket客户端发送消息
                messagingTemplate.convertAndSendToUser(redisWebsocketMsg.getReceiver()
            }
        }

    }
}
```

(4) 在Controller中发送WebSocket消息：

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

import cn.zifangsky.mqwebsocket.common.Constants;
 import cn.zifangsky.mqwebsocket.common.SpringContextUtils;
 import cn.zifangsky.mqwebsocket.enums.ExpireEnum;
 import cn.zifangsky.mqwebsocket.enums.WebSocketChannelEnum;
 import cn.zifangsky.mqwebsocket.model.User;
 import cn.zifangsky.mqwebsocket.model.websocket.HelloMessage;
 import cn.zifangsky.mqwebsocket.model.websocket.RedisWebsocketMsg;
 import cn.zifangsky.mqwebsocket.service.RedisService;
 import cn.zifangsky.mqwebsocket.utils.JsonUtils;
 import org.apache.commons.lang3.StringUtils;
 import org.slf4j.Logger;
 import org.slf4j.LoggerFactory;
 import org.springframework.beans.factory.annotation.Autowired;
 import org.springframework.beans.factory.annotation.Value;
 import org.springframework.messaging.simp.SimpMessagingTemplate;
 import org.springframework.messaging.simp.user.SimpUser;
 import org.springframework.messaging.simp.user.SimpUserRegistry;
 import org.springframework.stereotype.Controller;
 import org.springframework.web.bind.annotation.PostMapping;
 import org.springframework.web.bind.annotation.RequestMapping;
 import org.springframework.web.bind.annotation.ResponseBody;

 import javax.annotation.Resource;
 import javax.servlet.http.HttpServletRequest;
 import javax.servlet.http.HttpSession;
 import java.text.MessageFormat;
 import java.util.HashMap;
 import java.util.List;
 import java.util.Map;

 /**
  * 测试{@link org.springframework.messaging.simp.SimpMessagingTemplate}类的基本用法
  * @author zifangsky
  * @date 2018/10/10
  * @since 1.0.0
  */
 @Controller
 @RequestMapping(("/wsTemplate"))
 public class RedisMessageController {
     private final Logger logger = LoggerFactory.getLogger(getClass());

     @Value("${spring.redis.message.topic-name}")
     private String topicName;

     @Autowired
     private SimpMessagingTemplate messagingTemplate;

     @Autowired
     private SimpUserRegistry userRegistry;

     @Resource(name = "redisServiceImpl")
     private RedisService redisService;

     /**
      * 给指定用户发送WebSocket消息
      */
     @PostMapping("/sendToUser")
     @ResponseBody
     public String chat(HttpServletRequest request) {
         //消息接收者
         String receiver = request.getParameter("receiver");
         //消息内容
         String msg = request.getParameter("msg");
         HttpSession session = SpringContextUtils.getSession();
         User loginUser = (User) session.getAttribute(Constants.SESSION_USER);

         HelloMessage resultData = new HelloMessage(MessageFormat.format("{0} say: {1}
         this.sendToUser(loginUser.getUsername(), receiver, WebSocketChannelEnum.CHAT.

         return "ok";
     }

     /**
      * 给指定用户发送消息，并处理接收者不在线的情况
      * @param sender 消息发送者
      * @param receiver 消息接收者
      * @param destination 目的地
      * @param payload 消息正文
      */
     private void sendToUser(String sender, String receiver, String destination, Strin
         SimpUser simpUser = userRegistry.getUser(receiver);

         //如果接收者存在，则发送消息
         if(simpUser != null && StringUtils.isNoneBlank(simpUser.getName())){
             messagingTemplate.convertAndSendToUser(receiver, destination, payload);
         }
         //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
         else if(redisService.isSetMember(Constants.REDIS_WEBSOCKET_USER_SET, receiver
             RedisWebsocketMsg<String> redisWebsocketMsg = new RedisWebsocketMsg<>(rec

             redisService.convertAndSend(topicName, redisWebsocketMsg);
         }
         //否则将消息存储到redis，等用户上线后主动拉取未读消息
         else{
             //存储消息的Redis列表名
             String listKey = Constants.REDIS_UNREAD_MSG_PREFIX + receiver + ":" + des
             logger.info(MessageFormat.format("消息接收者{0}还未建立WebSocket连接，{1}发送

             //存储消息到Redis中
             redisService.addToListRight(listKey, ExpireEnum.UNREAD_MSG, payload);
         }

     }


     /**
      * 拉取指定监听路径的未读的WebSocket消息
      * @param destination 指定监听路径
      * @return java.util.Map<java.lang.String,java.lang.Object>
      */
     @PostMapping("/pullUnreadMessage")
     @ResponseBody
     public Map<String, Object> pullUnreadMessage(String destination){
         Map<String, Object> result = new HashMap<>();
         try {
             HttpSession session = SpringContextUtils.getSession();
             //当前登录用户
             User loginUser = (User) session.getAttribute(Constants.SESSION_USER);

             //存储消息的Redis列表名
             String listKey = Constants.REDIS_UNREAD_MSG_PREFIX + loginUser.getUsernam
             //从Redis中拉取所有未读消息
             List<Object> messageList = redisService.rangeList(listKey, 0, -1);

             result.put("code", "200");
             if(messageList !=null && messageList.size() > 0){
                 //删除Redis中的这个未读消息列表
                 redisService.delete(listKey);
                 //将数据添加到返回集，供前台页面展示
                 result.put("result", messageList);
             }
         }catch (Exception e){
             result.put("code", "500");
             result.put("msg", e.getMessage());
         }

         return result;
     }

 }
```

（5）其他拦截器处理WebSocket连接相关问题：

i）AuthHandshakeInterceptor:

```java
package cn.zifangsky.mqwebsocket.interceptor.websocket;

import cn.zifangsky.mqwebsocket.common.Constants;
import cn.zifangsky.mqwebsocket.common.SpringContextUtils;
import cn.zifangsky.mqwebsocket.model.User;
import cn.zifangsky.mqwebsocket.service.RedisService;
import org.apache.commons.lang3.StringUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.server.ServerHttpRequest;
import org.springframework.http.server.ServerHttpResponse;
import org.springframework.stereotype.Component;
import org.springframework.web.socket.WebSocketHandler;
import org.springframework.web.socket.server.HandshakeInterceptor;

import javax.annotation.Resource;
import javax.servlet.http.HttpSession;
import java.text.MessageFormat;
import java.util.Map;

/**
 * 自定义{@link org.springframework.web.socket.server.HandshakeInterceptor}，实现“需要登录
 *
 * @author zifangsky
 * @date 2018/10/11
 * @since 1.0.0
 */
@Component
public class AuthHandshakeInterceptor implements HandshakeInterceptor {
    private final Logger logger = LoggerFactory.getLogger(getClass());

    @Resource(name = "redisServiceImpl")
    private RedisService redisService;

    @Override
    public boolean beforeHandshake(ServerHttpRequest serverHttpRequest, ServerHttpResp
        HttpSession session = SpringContextUtils.getSession();
        User loginUser = (User) session.getAttribute(Constants.SESSION_USER);

        if(redisService.isSetMember(Constants.REDIS_WEBSOCKET_USER_SET, loginUser.getU
            logger.error("同一个用户不准建立多个连接WebSocket");
            return false;
        }else if(loginUser == null || StringUtils.isBlank(loginUser.getUsername())){
            logger.error("未登录系统，禁止连接WebSocket");
            return false;
        }else{
            logger.debug(MessageFormat.format("用户{0}请求建立WebSocket连接", loginUser.
            return true;
        }
    }

    @Override
    public void afterHandshake(ServerHttpRequest serverHttpRequest, ServerHttpResponse

    }

}
```

ii）MyHandshakeHandler：

```java
package cn.zifangsky.mqwebsocket.interceptor.websocket;

import cn.zifangsky.mqwebsocket.common.Constants;
import cn.zifangsky.mqwebsocket.common.SpringContextUtils;
import cn.zifangsky.mqwebsocket.model.User;
import cn.zifangsky.mqwebsocket.service.RedisService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.server.ServerHttpRequest;
import org.springframework.stereotype.Component;
import org.springframework.web.socket.WebSocketHandler;
import org.springframework.web.socket.server.support.DefaultHandshakeHandler;

import javax.annotation.Resource;
import javax.servlet.http.HttpSession;
import java.security.Principal;
import java.text.MessageFormat;
import java.util.Map;

/**
 * 自定义{@link org.springframework.web.socket.server.support.DefaultHandshakeHandler},
 *
 * @author zifangsky
 * @date 2018/10/11
 * @since 1.0.0
 */
@Component
public class MyHandshakeHandler extends DefaultHandshakeHandler{
    private final Logger logger = LoggerFactory.getLogger(getClass());

    @Resource(name = "redisServiceImpl")
    private RedisService redisService;

    @Override
    protected Principal determineUser(ServerHttpRequest request, WebSocketHandler wsHa
        HttpSession session = SpringContextUtils.getSession();
        User loginUser = (User) session.getAttribute(Constants.SESSION_USER);

        if(loginUser != null){
            logger.debug(MessageFormat.format("WebSocket连接开始创建Principal，用户：{0}"
            //1. 将用户名存到Redis中
            redisService.addToSet(Constants.REDIS_WEBSOCKET_USER_SET, loginUser.getUse

            //2. 返回自定义的Principal
            return new MyPrincipal(loginUser.getUsername());
        }else{
            logger.error("未登录系统，禁止连接WebSocket");
            return null;
        }
    }

}
```

iii）MyChannelInterceptor：

```java
package cn.zifangsky.mqwebsocket.interceptor.websocket;

import cn.zifangsky.mqwebsocket.common.Constants;
import cn.zifangsky.mqwebsocket.service.RedisService;
import org.apache.commons.lang3.StringUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.messaging.Message;
import org.springframework.messaging.MessageChannel;
import org.springframework.messaging.simp.stomp.StompCommand;
import org.springframework.messaging.simp.stomp.StompHeaderAccessor;
import org.springframework.messaging.support.ChannelInterceptor;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;
import java.security.Principal;
import java.text.MessageFormat;

/**
 * 自定义{@link org.springframework.messaging.support.ChannelInterceptor}，实现断开连接的
 *
 * @author zifangsky
 * @date 2018/10/10
 * @since 1.0.0
 */
@Component
public class MyChannelInterceptor implements ChannelInterceptor{
    private final Logger logger = LoggerFactory.getLogger(getClass());

    @Resource(name = "redisServiceImpl")
    private RedisService redisService;

    @Override
    public void afterSendCompletion(Message<?> message, MessageChannel channel, boolea
        StompHeaderAccessor accessor = StompHeaderAccessor.wrap(message);
        StompCommand command = accessor.getCommand();

        //用户已经断开连接
        if(StompCommand.DISCONNECT.equals(command)){
            String user = "";
            Principal principal = accessor.getUser();
            if(principal != null && StringUtils.isNoneBlank(principal.getName())){
                user = principal.getName();

                //从Redis中移除用户
                redisService.removeFromSet(Constants.REDIS_WEBSOCKET_USER_SET, user);
            }else{
                user = accessor.getSessionId();
            }

            logger.debug(MessageFormat.format("用户{0}的WebSocket连接已经断开", user));
        }
    }

}
```

（6）WebSocket相关配置：

```java
package cn.zifangsky.mqwebsocket.config;

import cn.zifangsky.mqwebsocket.interceptor.websocket.MyHandshakeHandler;
import cn.zifangsky.mqwebsocket.interceptor.websocket.AuthHandshakeInterceptor;
import cn.zifangsky.mqwebsocket.interceptor.websocket.MyChannelInterceptor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.messaging.simp.config.ChannelRegistration;
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.WebSocketMessageBrokerConfigur

/**
 * WebSocket相关配置
 *
 * @author zifangsky
 * @date 2018/9/30
 * @since 1.0.0
 */
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer{
    @Autowired
    private AuthHandshakeInterceptor authHandshakeInterceptor;

    @Autowired
    private MyHandshakeHandler myHandshakeHandler;

    @Autowired
    private MyChannelInterceptor myChannelInterceptor;

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/chat-websocket")
                .addInterceptors(authHandshakeInterceptor)
                .setHandshakeHandler(myHandshakeHandler)
                .withSockJS();
    }

    @Override
    public void configureMessageBroker(MessageBrokerRegistry registry) {
        //客户端需要把消息发送到/message/xxx地址
        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
        registry.enableSimpleBroker("/topic");
        //给指定用户发送消息的路径前缀，默认值是/user/
        registry.setUserDestinationPrefix("/user/");
    }

    @Override
    public void configureClientInboundChannel(ChannelRegistration registration) {
        registration.interceptors(myChannelInterceptor);
    }

}
```

（7）示例页面：

```
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta content="text/html;charset=UTF-8"/>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <meta http-equiv="X-UA-Compatible" content="IE=edge"/>
    <meta name="viewport" content="width=device-width, initial-scale=1"/>
    <title>Chat With STOMP Message</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"><
    <script src="https://cdnjs.cloudflare.com/ajax/libs/sockjs-client/1.1.4/sockjs.mi
    <script src="https://cdnjs.cloudflare.com/ajax/libs/stomp.js/2.3.3/stomp.min.js">
    <script th:src="@{/layui/layui.js}"></script>
    <script th:src="@{/layui/lay/modules/layer.js}"></script>
    <link th:href="@{/layui/css/layui.css}" rel="stylesheet">
    <link th:href="@{/layui/css/modules/layer/default/layer.css}" rel="stylesheet">
    <link th:href="@{/css/style.css}" rel="stylesheet">
    <style type="text/css">
        #connect-container {
            margin: 0 auto;
            width: 400px;
        }

        #connect-container div {
            padding: 5px;
            margin: 0 7px 10px 0;
        }

        .message input {
            padding: 5px;
            margin: 0 7px 10px 0;
        }

        .layui-btn {
            display: inline-block;
        }
    </style>
    <script type="text/javascript">
        var stompClient = null;

        $(function () {
            var target = $("#target");
            if (window.location.protocol === 'http:') {
                target.val('http://' + window.location.host + target.val());
            } else {
                target.val('https://' + window.location.host + target.val());
            }
        });

        function setConnected(connected) {
            var connect = $("#connect");
            var disconnect = $("#disconnect");
            var echo = $("#echo");

            if (connected) {
                connect.addClass("layui-btn-disabled");
                disconnect.removeClass("layui-btn-disabled");
                echo.removeClass("layui-btn-disabled");
            } else {
                connect.removeClass("layui-btn-disabled");
                disconnect.addClass("layui-btn-disabled");
                echo.addClass("layui-btn-disabled");
            }

            connect.attr("disabled", connected);
            disconnect.attr("disabled", !connected);
            echo.attr("disabled", !connected);
        }

        //连接
        function connect() {
            var target = $("#target").val();

            var ws = new SockJS(target);
            stompClient = Stomp.over(ws);

            stompClient.connect({}, function () {
                setConnected(true);
                log('Info: STOMP connection opened.');

                //连接成功后，主动拉取未读消息
                pullUnreadMessage("/topic/reply");

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

        //断开连接
        function disconnect() {
            if (stompClient != null) {
                stompClient.disconnect();
                stompClient = null;
            }
            setConnected(false);
            log('Info: STOMP connection closed.');
        }

        //向指定用户发送消息
        function sendMessage() {
            if (stompClient != null) {
                var receiver = $("#receiver").val();
                var msg = $("#message").val();
                log('Sent: ' + JSON.stringify({'receiver': receiver, 'msg':msg}));

                $.ajax({
                    url: "/wsTemplate/sendToUser",
                    type: "POST",
                    dataType: "json",
                    async: true,
                    data: {
                        "receiver": receiver,
                        "msg": msg
                    },
                    success: function (data) {

                    }
                });
            } else {
                layer.msg('STOMP connection not established, please connect.', {
                    offset: 'auto'
                    ,icon: 2
                });
            }
        }

        //从服务器拉取未读消息
        function pullUnreadMessage(destination) {
            $.ajax({
                url: "/wsTemplate/pullUnreadMessage",
                type: "POST",
                dataType: "json",
                async: true,
                data: {
                    "destination": destination
                },
                success: function (data) {
                    if (data.result != null) {
                        $.each(data.result, function (i, item) {
                            log(JSON.parse(item).content);
                        })
                    } else if (data.code !=null && data.code == "500") {
                        layer.msg(data.msg, {
                            offset: 'auto'
                            ,icon: 2
                        });
                    }
                }
            });
        }

        //日志输出
        function log(message) {
            console.debug(message);
        }
    </script>
</head>
<body>
    <noscript><h2 style="color: #ff0000">Seems your browser doesn't support Javascrip
        enabled. Please enable
        Javascript and reload this page!</h2></noscript>
    <div>
        <div id="connect-container" class="layui-elem-field">
            <legend>Chat With STOMP Message</legend>
            <div>
                <input id="target" type="text" class="layui-input" size="40" style="w
            </div>
            <div>
                <button id="connect" class="layui-btn layui-btn-normal" onclick="conn
                <button id="disconnect" class="layui-btn layui-btn-normal layui-btn-d
```

```
                        onclick="disconnect();">Disconnect
                    </button>

                </div>
                <div class="message">
                    <input id="receiver" type="text" class="layui-input" size="40" style=
                    <input id="message" type="text" class="layui-input" size="40" style="
                </div>
                <div>
                    <button id="echo" class="layui-btn layui-btn-normal layui-btn-disable
                            onclick="sendMessage();">Send Message
                    </button>
                </div>
            </div>
        </div>
</body>
</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)

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

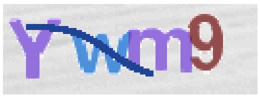
发表评论

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)