链滴

Spring Websocket 配置

作者：714593351

原文链接：https://ld246.com/article/1512177127884

来源网站：链滴

1. 依赖包

```
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-websocket</artifactId>
   <version>4.1.4.RELEASE</version>
</dependency>
```

2. spring 配置

我关闭了 sockjs，因为我的系统里使用了异步，打开了 <task:scheduler id="taskScheduler" pool-size="10"/> 和 <task:executor id="async_executor" pool-size="10"/> ，开启 sockjs 的时候会和 sync_executor 这个 bean 冲突，没有找到解决办法，所以就先关闭了。。

```
<websocket:handlers allowed-origins="*">
      <websocket:mapping path="/ws/user-channel" handler="userMessagesHandler"/>
      <websocket:handshake-interceptors>
         <bean class="com.xx.xx.websocket.interceptor.HandshakeInterceptor"/>
      </websocket:handshake-interceptors>
      <!-- 开启sockjs，去掉则关闭sockjs -->
      <!--<websocket:sockjs/>-->
   </websocket:handlers>
```

3. 拦截器实现

HandshakeInterceptor 这个拦截器很简单，我只做了把系统 session 里的用户名取出来放到 WebSocketSession 里，以便在服务端主动发消息时，能找到对应用户。

```
public class HandshakeInterceptor extends HttpSessionHandshakeInterceptor {

   @Override
   public boolean beforeHandshake(ServerHttpRequest request,
                                  ServerHttpResponse response, WebSocketHandler wsHandler,
                                  Map<String, Object> attributes) throws Exception {
      // 解决The extension [x-webkit-deflate-frame] is not supported问题
      if (request.getHeaders().containsKey("Sec-WebSocket-Extensions")) {
         request.getHeaders().set("Sec-WebSocket-Extensions",
               "permessage-deflate");
      }
      if (request instanceof ServletServerHttpRequest) {
         ServletServerHttpRequest servletRequest = (ServletServerHttpRequest) request;
         HttpSession session = servletRequest.getServletRequest().getSession(false);
         if (session != null) {
            //使用userName区分WebSocketHandler，以便定向发送消息
            User user = (User) session.getAttribute(Constants.SessionAttr.USER.getValue());
            String userName = user.getLoginName();
            attributes.put(WebsocketEndPoint.WEBSOCKET_USERNAME, userName);
         }
      }
      return super.beforeHandshake(request, response, wsHandler, attributes);
   }
```

```
    @Override
    public void afterHandshake(ServerHttpRequest request,
                               ServerHttpResponse response, WebSocketHandler wsHandler,
                               Exception ex) {
        super.afterHandshake(request, response, wsHandler, ex);
    }

}
```

4. 消息处理类实现

UserMessagesHandler 是 websocket 消息处理类。

其中 afterConnectionEstablished 方法主要是将请求 session 缓存到 CopyOnWriteArraySet 。这有个特殊处理，如果 sessionId 已存在缓存里，则将之前是连接关闭，并将之移出 CopyOnWriteArraSet 。

sendMessageToUser 方法是发送消息到具体用户的方法。

```
@Component
public class UserMessagesHandler implements WebSocketHandler {

    private static final Logger logger = LoggerFactory.getLogger(UserMessagesHandler.class);

    public static final String WEBSOCKET_USERNAME = "WS_USERNAME";
    public static final String SESSION_ID_ATTRIBUTE = "HTTP.SESSION.ID";

//    private static final ArrayList<WebSocketSession> users;

    public static CopyOnWriteArraySet<WebSocketSession> users = new CopyOnWriteArraySe
<WebSocketSession>();

    @Resource
    private UserService userService;
    @Resource
    private UserNoticeDAO userNoticeDAO;

    @Override
//    @Transactional(readOnly = true)
    public void afterConnectionEstablished(WebSocketSession session) {
        try {
            String sessionId = (String) session.getAttributes().get(SESSION_ID_ATTRIBUTE);
            if (StringUtils.hasText(sessionId)) {
                for (WebSocketSession user : users) {
                    if (sessionId.equals(user.getAttributes().get(SESSION_ID_ATTRIBUTE))) {
                        user.close(new CloseStatus(CloseStatus.SESSION_NOT_RELIABLE.getCode(), "M
ltiple tab websocket connect"));
                        users.remove(user);
                    }
                }
            }
            logger.debug("connect to the websocket success.....");
            users.add(session);
        } catch (Exception e) {
```

```
            e.printStackTrace();
        }
    }

    @Override
    public void handleMessage(WebSocketSession session, WebSocketMessage<?> message)
hrows Exception {

        //sendMessageToUsers();
        TextMessage returnMessage = new TextMessage(message.getPayload()+" received at se
ver");
//        session.sendMessage(returnMessage);
    }

    @Override
    public void handleTransportError(WebSocketSession session, Throwable exception) throws
xception {
        if(session.isOpen()){
            session.close();
        }
        logger.debug("websocket connection closed......");
        users.remove(session);
    }

    @Override
    public void afterConnectionClosed(WebSocketSession session, CloseStatus closeStatus) thr
ws Exception {
        logger.debug("websocket connection closed......");
        users.remove(session);
    }

    @Override
    public boolean supportsPartialMessages() {
        return false;
    }

    /**
     * 给所有在线用户发送消息
     *
     * @param message
     */
    public void sendMessageToUsers(TextMessage message) {
        for (WebSocketSession user : users) {
            try {
                if (user.isOpen()) {
                    user.sendMessage(message);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    /**
```

```
     * 给某个用户发送消息
     *
     * @param userName
     * @param message
     */
    public void sendMessageToUser(String userName, TextMessage message) {
        for (WebSocketSession user : users) {
            if (user.getAttributes().get(WEBSOCKET_USERNAME).equals(userName)) {
                try {
                    if (user.isOpen()) {
                        user.sendMessage(message);
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
                break;
            }
        }
    }
}
```