

# websocket 记录

## 实用工具类

<http://websocket.jsonin.com/>

## 1.spring 整合websocket

### 1.不用注解的方式

#### 1.简单说明

1.spring整合websocket有两种方式

##### 1.1 实现org.springframework.web.socket.WebSocketHandler

重写里面的方法

1. 方法 `afterConnectionEstablished()`，是连接之后进行的操作，类似于以前的 `onopen` 方法。里面有一个参数 `WebSocketSession`，表示连接进来的那一个 `Session`。可以通过 `getAttributes()` 方法，获取 `HandshakeInterceptor` 拦截器放置的 `paramMap` 集合。

2. 方法 `handleMessage()`，是服务器接收浏览器发送过来的消息进行的操作，类似于以前的 `onmessage` 方法。`WebSocketSession` 对象表示 发送消息的那一个`Session`，`WebSocketMessage` 表示发送的消息主体。

3. 方法 `handleTransportError()` 是出现异常时进行的操作，类似于以前的 `onerror` 方法。`WebSocketSession` 对象表示哪一个`Session` 出现了错误异常。

4. 方法 `afterConnectionClosed()`，是浏览器断开连接或者服务器断开连接的操作，类似于以前的 `onclose` 方法，`WebSocketSession` 表示 断开的是哪一个`Session`

5. 方法 `supportsPartialMessages()` 表示是否支持拆分。当浏览器输入的内容过多时，允不允许将接收到的内容，进行拆分处理。通常不允许拆分，返回 `false` 即可。

##### 1.2 利用注解注释类方法,这种是用java自带的包

`javax.websocket.ClientEndpoint`

`javax.websocket.ServerEndpoint`

`javax.websocket.Session`

`@OnOpen`

`@OnMessage`

`@OnError`

`@OnClose`

```
public void send(String message) {  
    this.session.getAsyncRemote().sendText(message);  
}
```

## 2.服务端

### 1.pom.xml

```
<dependencies>  
    <!--<-->  
    <dependency>
```

```

        <groupId>org.springframework</groupId>
        <artifactId>spring-websocket</artifactId>
        <version>4.2.4.RELEASE</version>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <excludes>
                    <exclude>
                        <groupId>org.projectlombok</groupId>
                        <artifactId>lombok</artifactId>
                    </exclude>
                </excludes>
            </configuration>
        </plugin>
    </plugins>
</build>xml

```

## 2. 拦截器

1. 继承spring提供的拦截器接口

`org.springframework.web.socket.server.support.HttpSessionHandshakeInterceptor`

2. 内容

```

public class WebSocketInterceptor extends HttpSessionHandshakeInterceptor {
    @Override
    public boolean beforeHandshake(ServerHttpRequest request,
                                   ServerHttpResponse response, WebSocketHandler
wsHandler,

```

```

        Map<String, Object> attributes) throws
Exception {
    if (request instanceof ServletServerHttpRequest) {
        ServletServerHttpRequest serverHttpRequest =
(ServletServerHttpRequest) request;
        // 获取参数
        String userId = serverHttpRequest.getServletRequest().getParameter(
            "userId");
        attributes.put("currentUser", userId);
    }

    return true;
}

// 初次握手访问后
@Override
public void afterHandshake(ServerHttpRequest serverHttpRequest,
    ServerHttpResponse serverHttpResponse,
    WebSocketHandler webSocketHandler, Exception e) {

}

```

### 3.方法说明

beforeHandshake() 方法，是请求连接之前的处理方法，  
afterHandshake() 方法，是请求连接成功之后的处理方法。

## 3.消息处理器

1.实现spring提供的接口org.springframework.web.socket.WebSocketHandler

2.内容

```

public class MyMessageHandler implements WebSocketHandler {

    /**
     * userMap:使用线程安全map存储用户连接websocket信息
     *
     * @since JDK 1.7
     */
    private final static Map<String, WebSocketSession> userMap = new
ConcurrentHashMap<String, WebSocketSession>();

    /**
     * 关闭websocket时调用该方法
     *
     * @see
org.springframework.web.socket.WebSocketHandler#afterConnectionClosed(org.spring
framework.web.socket.WebSocketSession,
    * org.springframework.web.socket.CloseStatus)
     */
    @Override
    public void afterConnectionClosed(WebSocketSession session,
        CloseStatus status) throws Exception {
        String userId = this.getUserId(session);
        if (!StringUtils.isEmpty(userId)) {
            userMap.remove(userId);
            System.err.println("该" + userId + "用户已成功关闭");
        } else {
            System.err.println("关闭时，获取用户id为空");
        }
    }
}

```

```

    }

    /**
     * 建立websocket连接时调用该方法
     * <p>
     *
     org.springframework.web.socket.WebSocketHandler#afterConnectionEstablished(org.s
     pringframework.web.socket.WebSocketSession)
     */
    @Override
    public void afterConnectionEstablished(WebSocketSession session)
        throws Exception {
        String userId = this.getUserId(session);
        if (!StringUtils.isEmpty(userId)) {
            userMap.put(userId, session);
            System.out.println("用户:" + userId + "连接成功! ");
            session.sendMessage(new TextMessage("建立服务端连接成功! "));
        }
    }

}

/**
 * 客户端调用websocket.send时候, 会调用该方法, 进行数据通信
 * <p>
 *
     org.springframework.web.socket.WebSocketHandler#handleMessage(org.springframewor
     k.web.socket.WebSocketSession,
     * org.springframework.web.socket.WebSocketMessage)
     */
    @Override
    public void handleMessage(WebSocketSession session,
        WebSocketMessage<?> message) throws Exception {
        String msg = message.toString();
        String userId = this.getUserId(session);
        System.err.println("该" + userId + "用户发送的消息是: " + msg);
        message = new TextMessage("服务端已经接收到消息, msg=" + msg);
        session.sendMessage(message);
    }

}

/**
 * 传输过程出现异常时, 调用该方法
 * <p>
 *
     org.springframework.web.socket.WebSocketHandler#handleTransportError(org.springf
     ramework.web.socket.WebSocketSession,
     * java.lang.Throwable)
     */
    @Override
    public void handleTransportError(WebSocketSession session, Throwable e)
        throws Exception {
        WebSocketMessage<String> message = new TextMessage("异常信息: "
            + e.getMessage());
        session.sendMessage(message);
    }

}

/**
 * org.springframework.web.socket.WebSocketHandler#supportsPartialMessages()

```

```

    */
    @Override
    public boolean supportsPartialMessages() {

        return false;
    }

    /**
     * sendMessageToUser:发给指定用户
     */
    public void sendMessageToUser(String userId, String contents) {
        WebSocketSession session = userMap.get(userId);
        if (session != null && session.isOpen()) {
            try {
                TextMessage message = new TextMessage(contents);
                session.sendMessage(message);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    /**
     * sendMessageToAllUsers:发给所有的用户
     */
    public void sendMessageToAllUsers(String contents) {
        Set<String> userIds = userMap.keySet();
        for (String userId : userIds) {
            this.sendMessageToUser(userId, contents);
        }
    }

    /**
     * getUserId:获取用户id
     *
     * @param session
     * @return
     * @author liuchao
     * @since JDK 1.7
     */
    private String getUserId(WebSocketSession session) {
        try {
            String userId = (String) session.getAttributes().get("currentUser");
            return userId;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
}

```

### 3. 说明

这里除了handler的功能，同时也肩负着连接管理的功能，一般可以将其分开成两个类

## 4.websocketConfig

### 1. 实现spring提供的接口

`org.springframework.web.socket.config.annotation.WebSocketConfigurer`

打上配置注解和启用websocket的注解

`org.springframework.web.socket.config.annotation.EnableWebSocket`

### 2. 内容

@Configuration

@EnableWebSocket

```
public class WebSocketConfig implements WebSocketConfigurer {  
    /**  
     * 注册handle  
     * @see  
     org.springframework.web.socket.config.annotation.WebSocketConfigurer#registerWeb  
     SocketHandlers(org.springframework.web.socket.config.annotation.WebSocketHandler  
     Registry)  
     */  
    @Override  
    public void registerWebSocketHandlers(WebSocketHandlerRegistry registry) {  
        registry.addHandler(myHandler(),  
            "/testHandler.do").addInterceptors(new  
            WebSocketInterceptor()).setAllowedOrigins("*");  
        registry.addHandler(myHandler(),  
            "/socketJs/testHandler.do").addInterceptors(new  
            WebSocketInterceptor()).setAllowedOrigins("*").withSockJS();  
    }  
  
    @Bean  
    public WebSocketHandler myHandler(){  
        return new MyMessageHandler();  
    }  
}
```

3.

## 2.使用注解的方式

### 1.简单说明

这里就以客户端演示如何使用java jdk 自带的方式使用websocket

### 2.客户端

#### 1.pom.xml

```
</properties>  
<dependencies>  
    <dependency>  
        <groupId>org.springframework.boot</groupId>  
        <artifactId>spring-boot-starter-websocket</artifactId>  
    </dependency>  
    <dependency>  
        <groupId>org.springframework.boot</groupId>  
        <artifactId>spring-boot-starter-web</artifactId>  
    </dependency>  
    <dependency>
```

```

        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>

```

## 2.配置WebSocketClient

1.不用像服务端那样，需要继承或者实现spring提供的接口，自己创建类，然后使用jdk中自带的类进行处理连接

2.内容

@Component

@ClientEndpoint

```
public class WebSocketClient {
```

```
    @Value("${websocket.server.url:localhost:8080/testHandler.do}")
```

```
    private String serverUrl;
```

```
    @Value("${websocket.server.user:system@1}")
```

```
    private String user;
```

```
    private Session session;
```

```
@PostConstruct
```

```
void init() {
```

```
    try {
```

```
        // 本机地址
```

```
        WebSocketContainer container =
```

```
ContainerProvider.getWebSocketContainer();
```

```
        String wsUrl = "ws://" + serverUrl + "?userId=" + user;
```

```
        URI uri = URI.create(wsUrl);
```

```
        session = container.connectToServer(WebSocketClient.class, uri);
```

```
    } catch (DeploymentException | IOException e) {
```

```
        e.printStackTrace();
```

```
    }
```

```
}
```

```
/**
```

```
 * 打开连接
```

```
 * @param session
```

```
 */
```

```
@OnOpen
```

```
public void onOpen(Session session) {
```

```
    System.out.println("打开");
```

```
    this.session = session;
```

```
}
```

```

/**
 * 接收消息
 * @param text
 */
@OnMessage
public void onMessage(String text) {
    System.out.println(text);
}

/**
 * 异常处理
 * @param throwable
 */
@OnError
public void onError(Throwable throwable) {
    throwable.printStackTrace();
}

/**
 * 关闭连接
 */
@OnClose
public void onClose() throws IOException {
    System.out.println("关闭");
    session.close();
}

/**
 * 主动发送消息
 */
public void send(String message) {
    this.session.getAsyncRemote().sendText(message);
}

public void close() throws IOException{
    if(this.session.isOpen()){
        this.session.close();
    }
}
}

```

### 3.说明

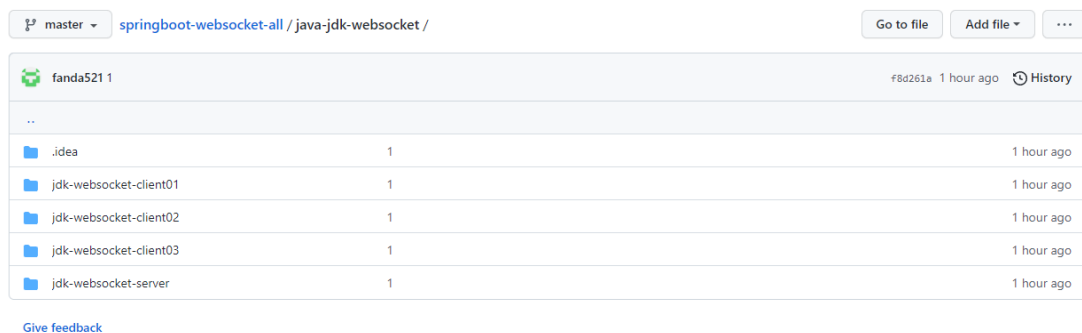
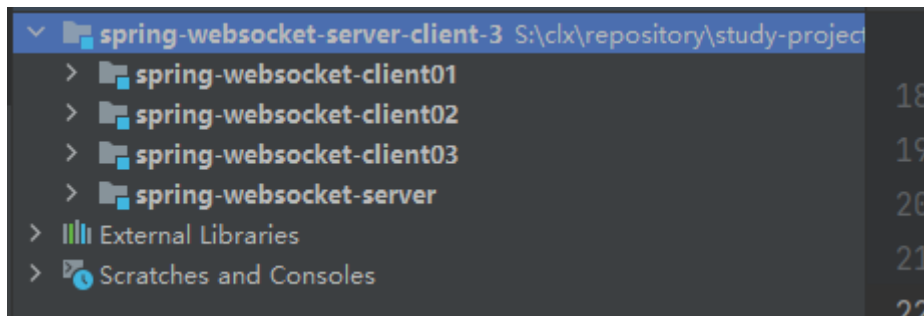
spring-websocket: 是服务端代码  
 spring-client01:是客户端01代码  
 client02:是客户端02代码  
 测试组消息发送的时候可以同时启动这三个进行测试

## 2.spring 和jdk



# 1.项目说明

如果想看单独的jdk 和 spring 客户端和服务端分别用同一种的  
spring:springboot-websocket-all/spring-websocket-server-client-3  
jdk:springboot-websocket-all/java-jdk-websocket/



## 3.webflux整合websocket

### 1.springboot方式

#### 1.server端

##### 1.pom.xml

1.注意点，不要添加普通web的jar,否则会默认启动普通的tomcat,而不是webflux的netty,导致ws启动不了

自己就在这里栽了跟头

2.内容

```
<properties>
    <java.version>1.8</java.version>
</properties>
<dependencies>
    <!--webflux-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-webflux</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
```

```

        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <excludes>
                    <exclude>
                        <groupId>org.projectlombok</groupId>
                        <artifactId>lombok</artifactId>
                    </exclude>
                </excludes>
            </configuration>
        </plugin>
    </plugins>
</build>

```

## 2.websocketConfig

```

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.stereotype.Component;
import org.springframework.web.reactive.handler.SimpleUrlHandlerMapping;
import org.springframework.web.reactive.socket.WebSocketHandler;
import org.springframework.web.reactive.socket.server.support.WebSocketHandlerAdapter;

import java.util.LinkedHashMap;
import java.util.Map;

@Configuration
public class WebSocketConfig {
    @Bean
    public HandlerMapping websocketMapping() {
        SimpleUrlHandlerMapping simpleUrlHandlerMapping = new
SimpleUrlHandlerMapping();
        Map<String, WebSocketHandler> handlerMap = new LinkedHashMap<>();
        handlerMap.put("/path", new EchoHandler());
        simpleUrlHandlerMapping.setUrlMap(handlerMap);
        simpleUrlHandlerMapping.setOrder(-1);
    }
}

```

```

        return simpleUrlHandlerMapping;
    }

    @Bean
    public WebSocketHandlerAdapter handlerAdapter() {
        return new WebSocketHandlerAdapter();
    }
}

```

### 3.handler

```

import org.springframework.stereotype.Component;
import org.springframework.web.reactive.socket.WebSocketHandler;
import org.springframework.web.reactive.socket.WebSocketMessage;
import org.springframework.web.reactive.socket.WebSocketSession;
import reactor.core.publisher.Flux;
import reactor.core.publisher.Mono;

@Component
public class EchoHandler implements WebSocketHandler {
    @Override
    public Mono<Void> handle(WebSocketSession session) {
        Flux<WebSocketMessage> messageFlux = session.receive().map(message -> {
            String payload = message.getPayloadAsText();
            return "Received: " + payload;
        }).map(session::textMessage).log();
        return session.send(messageFlux);
    }
}

```