KB금융그룹 | 국민의 평생 금융파트너

2025년 상반기 K-디지털 트레이닝

Spring Web Socket, STOMP

[KB] IT's Your Life

KB 국민은행

Spring Web Socket, STOMP

HTTP 통신 특징

- 비연결성 (connectionless) : 연결을 맺고 요청을 하고 응답을 받으면 연결을 끊김
- 무상태성 (stateless) : 서버가 클라이언트의 상태를 가지고 있지 음
- 단방향 통신

→ 채팅과 같은 실시간 통신에 적합하지 않음

Spring Web Socket, STOMP

웹소켓 프로토콜

- 클라이언트 주도 양방향 통신
- HTTP에서 동작 가능하게 디자인 되었고 80, 443 포트(ws 프로토콜)를 사용
- 일반 HTTP 요청에 Upgrade 헤더를 포함한 reqeust를 전송하면 WebSocket protocol로 변환되며 Web Socket interaction이 시작

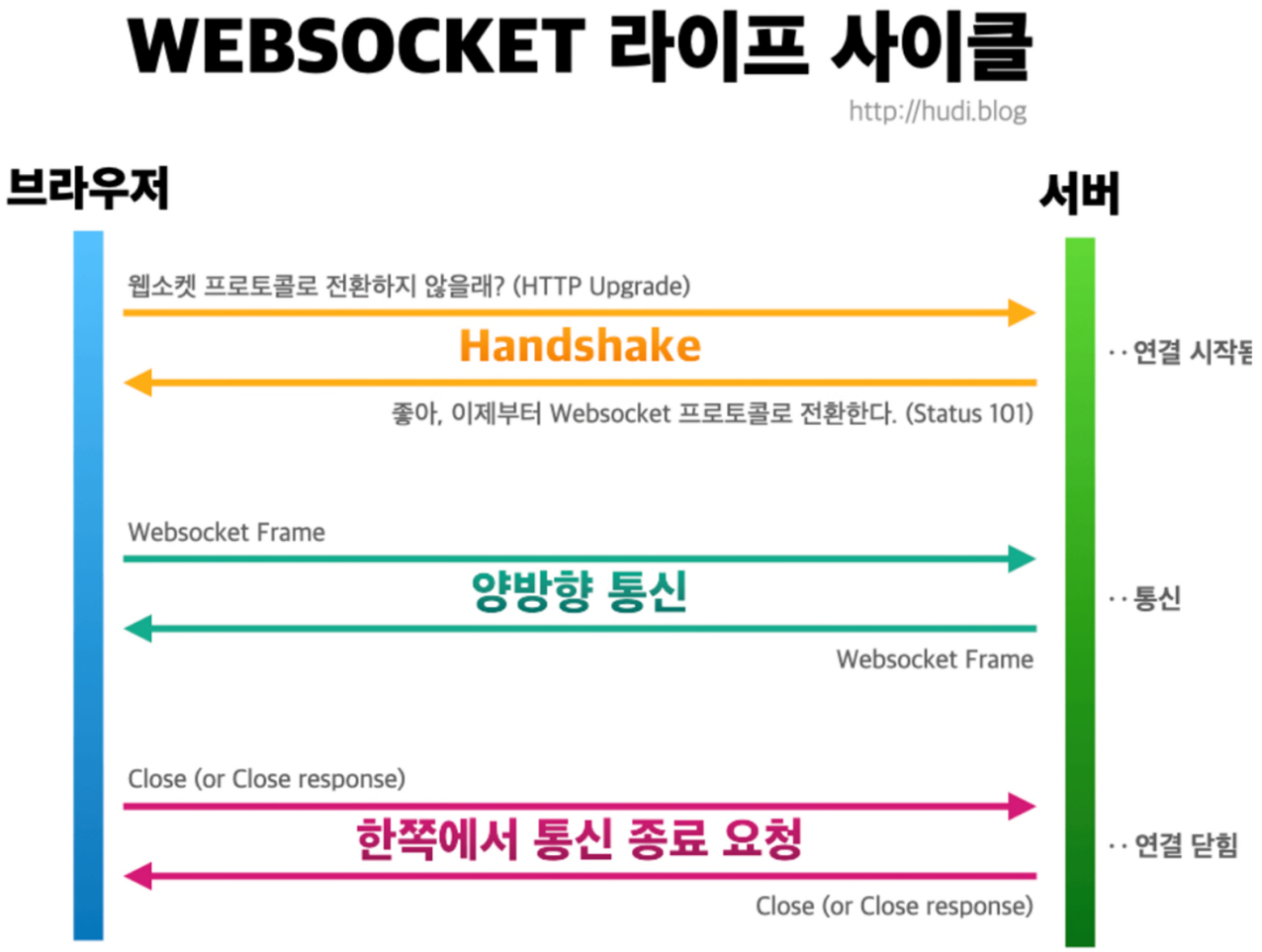
웹소켓 프로토콜

요청

GET /spring-websocket-portfolio/portfolio HTTP/1.1
Host: localhost:8080
Upgrade: websocket ← Upgrade 헤더
Connection: Upgrade ← Upgrade connection 사용
Sec-WebSocket-Key: Uc9l9TMkWGbHFD2qnFHltg==
Sec-WebSocket-Protocol: v10.stomp, v11.stomp
Sec-WebSocket-Version: 13
Origin: http://localhost:8080
Upgrade 헤더
Upgrade connection 사용

응답

HTTP/1.1 101 Switching Protocols ← 200 코드 대신 101 상태 코드 , 프로토콜 스위칭 - tcp 소켓 계속 유지
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: 1qVdfYHU9hPOl4JYYNXF623Gzn0=
Sec-WebSocket-Protocol: v10.stomp


WEBSOCKET 라이프 사이클
http://hudi.blog
브라우저
서버
웹소켓 프로토콜로 전환하지 않을래? (HTTP Upgrade)
Handshake
좋아, 이제부터 Websocket 프로토콜로 전환한다. (Status 101)
연결 시작됨
Websocket Frame
양방향 통신
Websocket Frame
통신
Close (or Close response)
한쪽에서 통신 종료 요청
Close (or Close response)
연결 닫힘

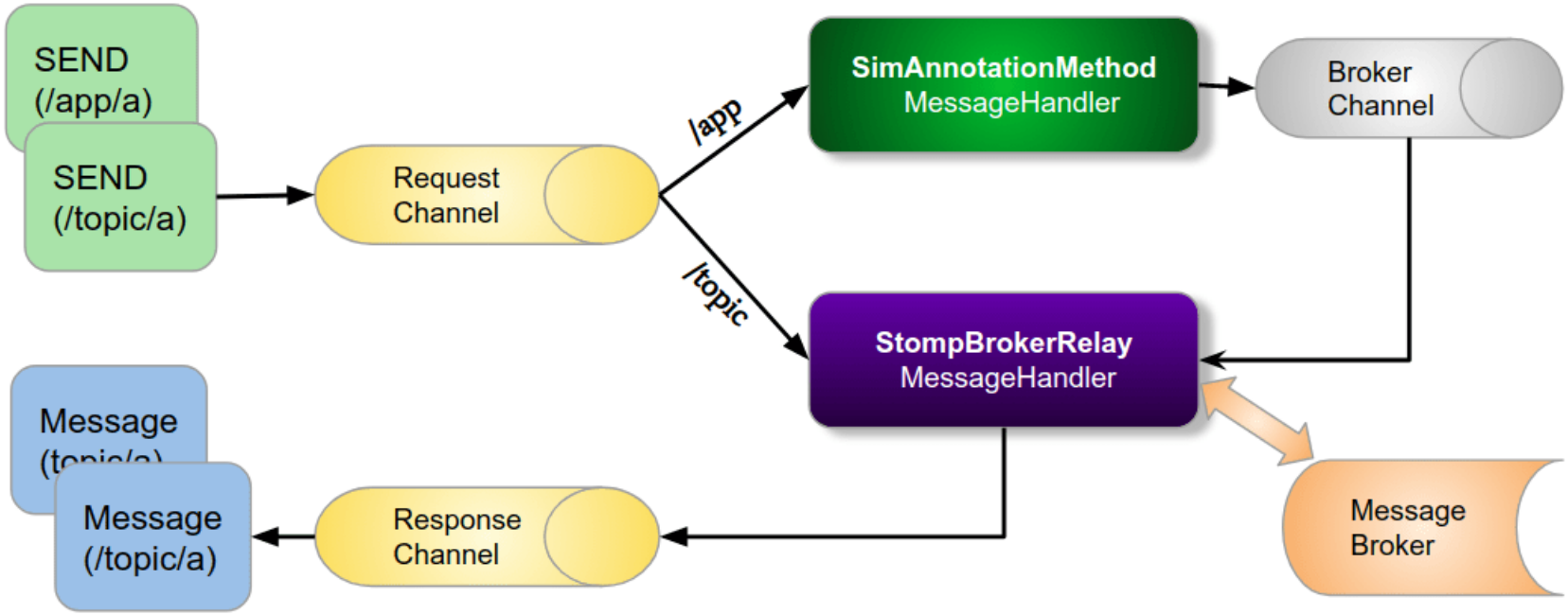
Spring Web Socket, STOMP

STOMP: Simple Text Oriented Messaging Protocol

- 간단한 메시지를 전송하기 위한 프로토콜
- 메시지 브로커와 publisher – subscriber 방식을 사용
  - publisher: 메시지 전송자
  - subscriber: 메시지 수신자
  - broker: publisher가 발행한 메시지를 subscriber에게 전달
- frame 기반 프로토콜
  - command, header, body로 구성

<STOMP frame 구조>
COMMAND
header1:value1
header2:value2
Body^@

STOMP 기반 통신 과정



- topic

Spring Web Socket, STOMP

build.gradle

```
implementation "org.springframework:spring-websocket:${springVersion}"
implementation "org.springframework:spring-messaging:${springVersion}"
implementation "org.springframework.integration:spring-integration-stomp:5.5.20"
```

메시지 브로커 설정

WebSocketMessageBrokerConfigurer 인터페이스 구현

- @EnableWebSocketMessageBroker 어노테이션으로 WebSocketMessageBroker 활성화
- **구현 메서드**

void configureMessageBroker(MessageBrokerRegistry config)

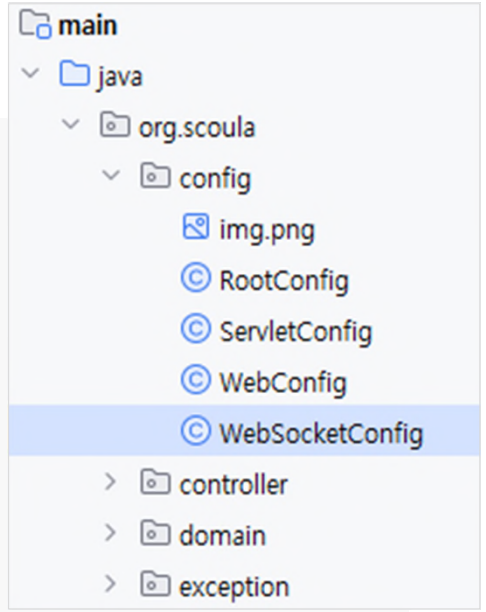
- 메시지 브로커 구성
- config.enableSimpleBroker("/topic");
  메모리 기반 브로커 활성화
  처리할 토픽을 매개변수로 지정
- config.setApplicationDestinationPrefixes("/app");
  메시지 경로의 접두어
  @MessageMapping("/hello") → /app/hello

void registerStompEndpoints(StompEndpointRegistry registry)

- registry.addEndpoint("/chat-app");
  클라이언트가 접속할 때 사용할 url 경로(브로커 url)
  ws://localhost:8080/chat-app

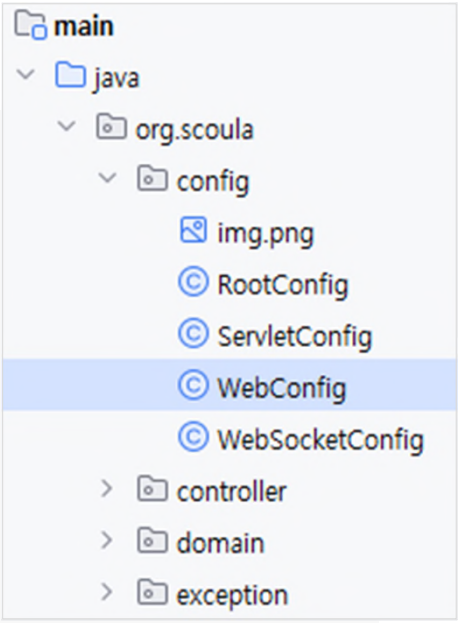
config.WebSocketConfig

```
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

 @Override
 public void configureMessageBroker(MessageBrokerRegistry config) {
  config.enableSimpleBroker("/topic");                    // 구독시 사용할 토픽 접두어
  // 클라이언트가 발행 시 사용해야하는 접두어
  config.setApplicationDestinationPrefixes("/app");
 }

 @Override
 public void registerStompEndpoints(StompEndpointRegistry registry) {
  registry.addEndpoint("/chat-app")  // 접속 엔드포인트, ws://localhost:8080/chat-app
        .setAllowedOrigins("*");                  // CORS 허용
 }

}
```



config.WebConfig :: WebSocketConfig 등록

```
public class WebConfig extends AbstractAnnotationConfigDispatcherServletInitializer {

   @Override
   protected Class<?>[] getRootConfigClasses() {
      return new Class[] { RootConfig.class };
   }

   @Override
   protected Class<?>[] getServletConfigClasses() {
      return new Class[] { ServletConfig.class, WebSocketConfig.class };
   }
  ...

}
```



Spring Web Socket, STOMP

Stomp 컨트롤러

- @MessageMapping(메시지경로)
  - 클라이언트가 보낸 메시지의 경로와 일치하는 경우 해당 메서드 호출
  - 메시지의 Body를 매개변수의 객체로 변환하여 전달
  - setApplicationDestinationPrefixes()에서 지정한 prefix는 제외하고 지정

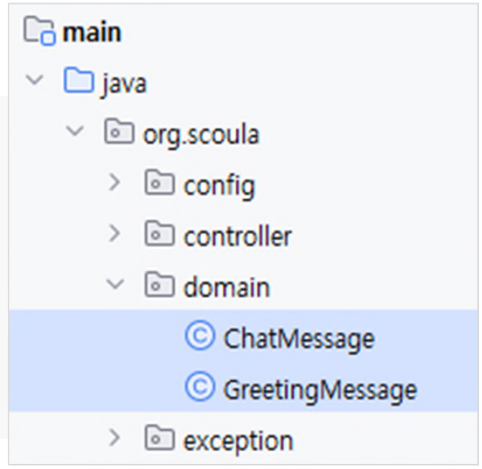
- @SendTo(토픽)
  - 해당 메서드의 리턴값을 지정한 토픽으로 전송

메시지

- GreetingMessage 입장 메시지
- ChatMessage 채팅 문자열 메시지

domain.GreetingMessage

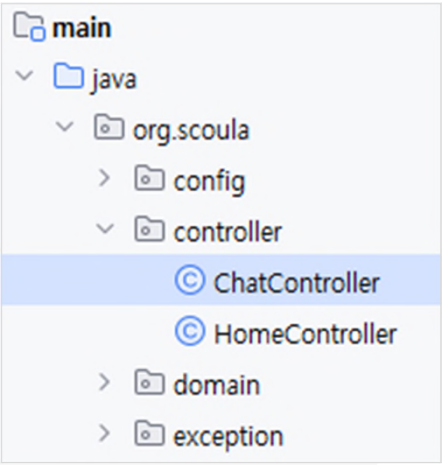
```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class GreetingMessage {
    private String name;
}
```



domain.ChatMessage

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class ChatMessage {
    private String name;
    private String content;
}
```

controller.ChatController.java

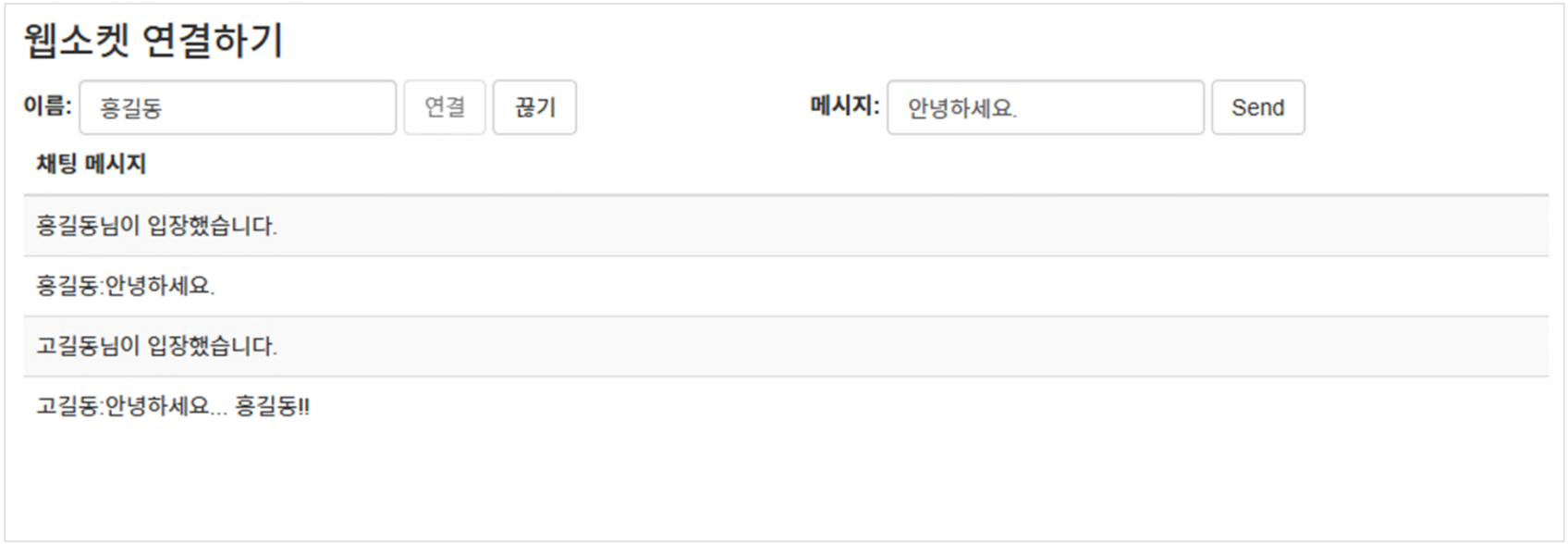



```
@Controller
@Log4j2
public class ChatController {

    @MessageMapping("/hello")
    @SendTo("/topic/greetings")
    public GreetingMessage greeting(GreetingMessage message) throws Exception {
        log.info("greeting: " + message);
        return message;
    }

    @MessageMapping("/chat")
    @SendTo("/topic/chat")
    public ChatMessage chat(ChatMessage message) throws Exception {
        log.info("chat received: " + message);
        return message;
    }
}
```

Spring Web Socket, STOMP

클라이언트

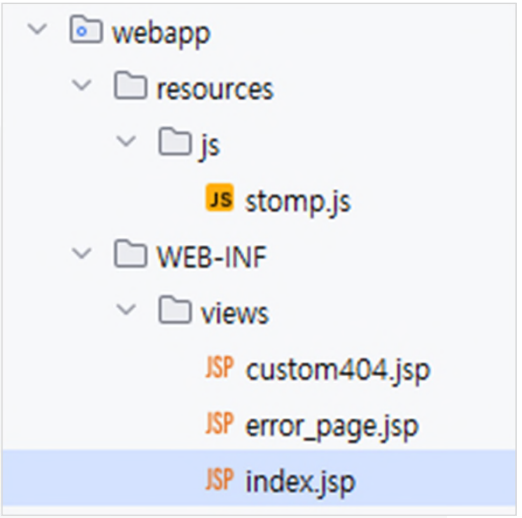
화면



Stomp 라이브러리 CDN

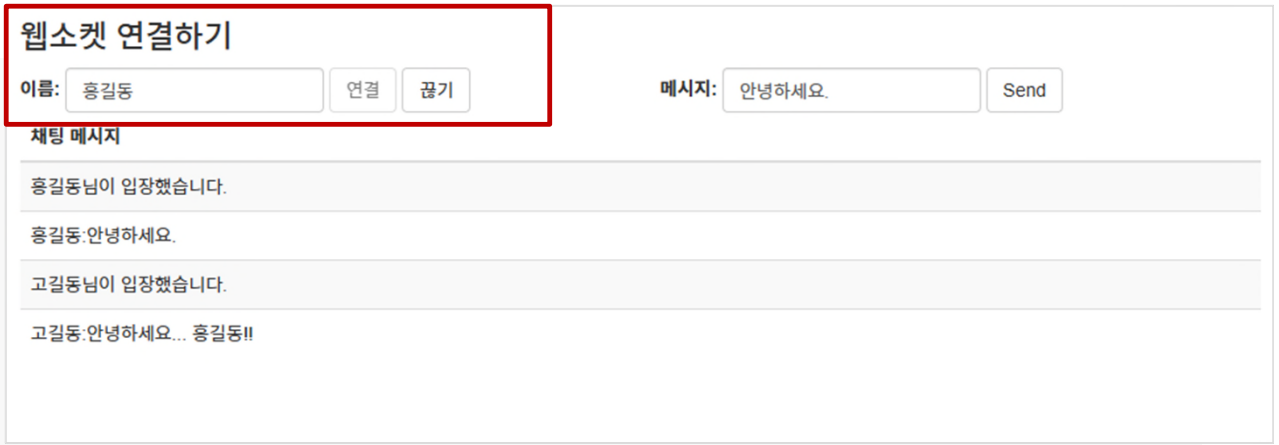
- https://cdn.jsdelivr.net/npm/@stomp/stompjs@7.0.0/bundles/stomp.umd.min.js

index.jsp 기본 골격




```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

<!DOCTYPE html>
<html>
<head>
  <title>Hello WebSocket</title>
  <link rel="stylesheet"
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
    integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
    crossorigin="anonymous">
  <script src="https://cdn.jsdelivr.net/npm/@stomp/stompjs@7.0.0/bundles/stomp.umd.min.js"></script>
</head>
<body>

  <script src="/resources/js/stomp.js"></script>
</body>
</html>
```

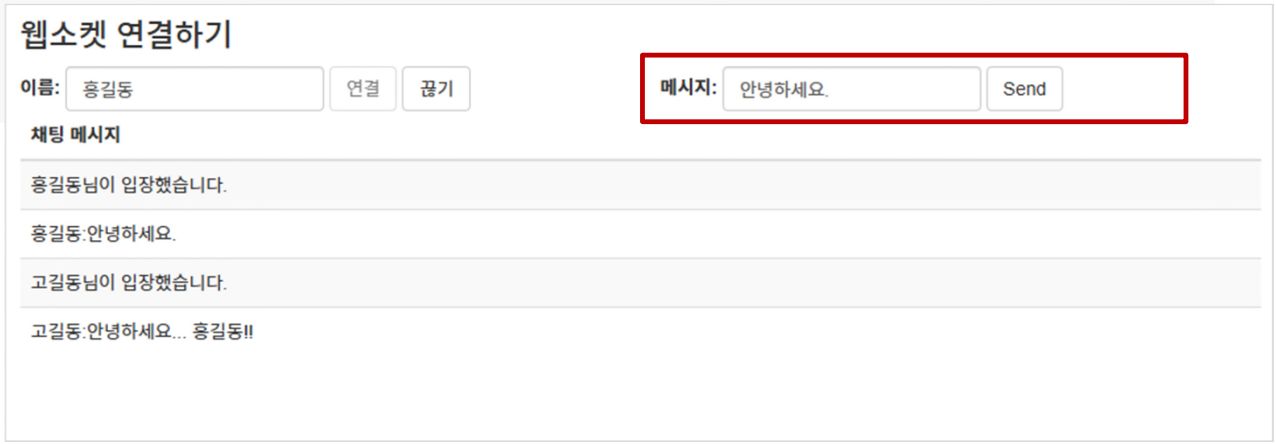
index.jsp




```
<body>
<div id="main-content" class="container">
 <h3>웹소켓 연결하기</h3>
 <div class="row">
  <div class="col-md-6">
   <form class="form-inline">
    <div class="form-group">
     <div class="form-group">
      <label for="name">이름: </label>
      <input type="text" id="name" class="form-control" placeholder="이름을 이력하세요.">
     </div>
     <button id="connect" class="btn btn-default" type="submit">연결</button>
     <button id="disconnect" class="btn btn-default" type="submit" disabled="disabled">끊기
     </button>
    </div>
   </form>
  </div>
```

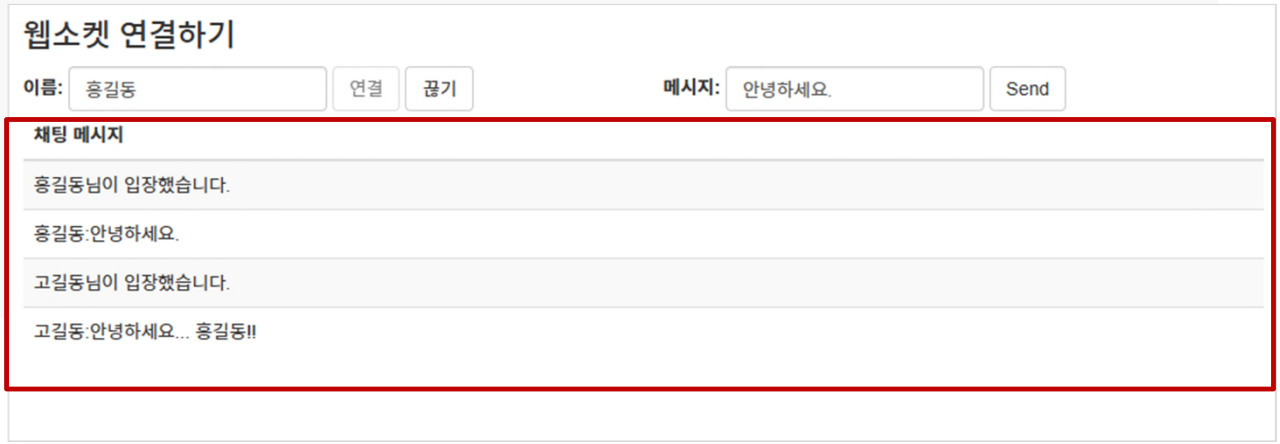
index.jsp

```
<div class="col-md-6">
 <form class="form-inline">
  <div class="form-group">
   <label for="content">메시지:</label>
   <input type="text" id="content" class="form-control" placeholder="메시지를 입력하세요...">
  </div>
  <button id="send" class="btn btn-default" type="submit">Send</button>
 </form>
</div>
</div>
```



index.jsp

```
<div class="row">
  <div class="col-md-12">
    <table class="table table-striped">
      <thead>
      <tr>
        <th>채팅 메시지</th>
      </tr>
      </thead>
      <tbody id="chat-messages">
      </tbody>
    </table>
  </div>
</div>
</div>
```



Stomp 라이브러리

StompClient

- StompJs.Client 클래스로 생성

```
const stompClient = new StompJs.Client({
    brokerURL: 'Web Socket 엔드 포인트'
});
```

연결 관리 메서드

- .activate() 연결하기
- .deactivate() 연결끊기

주요 이벤트 핸들러

- onConnect 연결 성공시 콜백 등록
- onWebSocketError웹 소켓 에러 발생시 콜백 등록
- onStompError Stomp 에러 발생시 콜백 등록

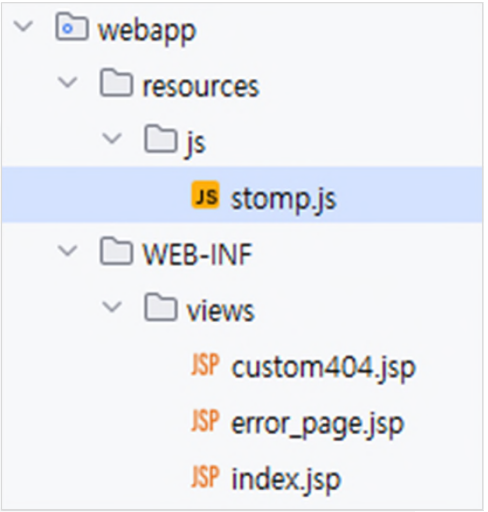
Spring Web Socket, STOMP

- **구독할 토픽 등록하기**
  - .subscribe(구독 토픽 문자열, 수신 핸들러 함수)

- **메시지 발행(전송) 하기**
  - .publish({ destination: 토픽 문자열, body: 전송할 메시지 })

resources/js/stomp.js

```
// StompJs.Client 객체 생성
const stompClient = new StompJs.Client({
    brokerURL: 'ws://localhost:8080/chat-app'
});


// 웹 소켓 에러 발생시 콜백
stompClient.onWebSocketError = (error) => {
    console.error('Error with websocket', error);
};

// Stomp 에러 발생시 콜백
stompClient.onStompError = (frame) => {
    console.error('Broker reported error: ' + frame.headers['message']);
    console.error('Additional details: ' + frame.body);
};
```



resources/js/stomp.js

```
// 연결 성공시 콜백
// 구독 토픽 등록
stompClient.onConnect = (frame) => {
    console.log(frame)
    setConnected(true);
    // 구독 토픽 등록 및 수신 처리 핸들러 등록
    // 토픽 문자열: '/topic/greetings' - 입장 메시지
    stompClient.subscribe('/topic/greetings', (greeting) => {
        console.log('/topic/greetings', greeting.body)
        showMessage(JSON.parse(greeting.body).name + '님이 입장했습니다.');
    });
    // 토픽 문자열: '/topic/chat' - chat 메시지
    stompClient.subscribe('/topic/chat', (chat) => {
        console.log('/topic/chat', chat.body)
        const message = JSON.parse(chat.body);
        showMessage(`${message.name}:${message.content}`);
    });
```

resources/js/stomp.js

```
    // 연결 성공시 입장 메시지 보내기
    const name =  document.getElementById('name').value;
    stompClient.publish({
        destination: '/app/hello',
        body: JSON.stringify({name})                    // GreetingMessage에 대응
    });

};

// 연결됐을 때 엘리먼트 프로퍼티 변경
function setConnected(connected) {
    const connectBtn = document.getElementById('connect');
    const disconnectBtn = document.getElementById('disconnect');
    const messages = document.getElementById('chat-messages');

    connectBtn.disabled = connected;
    disconnectBtn.disabled = !connected;
    messages.innerHTML = '';
}
```

resources/js/stomp.js

```
// 연결하기
function connect() {
    stompClient.activate();
}

// 연결 끊기
function disconnect() {
    stompClient.deactivate();
    setConnected(false);
    console.log('Disconnected');
}

// 메시지 전송하기
function sendMessage() {
    const name =  document.getElementById('name').value;
    const content =  document.getElementById('content').value;
    console.log({name, content})
    stompClient.publish({
        destination: '/app/chat',
        body: JSON.stringify({name, content});   // ChatMessage에 대응
    });
}
```

resources/js/stomp.js

```
// 수신 메시지 출력하기
function showMessage(message) {
    const messages = document.getElementById('chat-messages');
    messages.innerHTML += '<tr><td>' + message + '</td></tr>'
}

// 이벤트 핸들러 설정
window.addEventListener("DOMContentLoaded", (event) => {
    const forms = document.querySelectorAll('.form-inline');
    const connectBtn = document.getElementById('connect');
    const disconnectBtn = document.getElementById('disconnect');
    const sendBtn = document.getElementById('send');

    connectBtn.addEventListener('click', () => connect());
    disconnectBtn.addEventListener('click',  () => disconnect());
    sendBtn.addEventListener('click', () => sendMessage());
    for(const form of forms) {
        console.log(form)
        form.addEventListener('submit', (e) => e.preventDefault());
    }
});
```