보통 클라 요청으로부터 시작되는데

이번에는 서버가 시작하고 싶음.
그리고 서버가 지속적으로 시작하고 쌍방향 소통을 하고 싶다면...

지속적인 연결이 유지가 되어야 하는게 전제

클라에서는 js에 서버에서는 프레임웍을 통해 웹소켓을 이용해야함.

채팅 시나리오에서 많이 사용되는데 이것도 전형화된 패턴이 있어서
라이브러리? 프로토콜이 나옴. 이는 stomp.

KB금융그룹 | 국민의 평생 금융파트너

2025년 상반기 K-디지털 트레이닝

Spring Web Socket, STOMP

챗봇 만들때 많이 써.
ai랑 연계 많이 함.

[KB] IT's Your Life

KB 국민은행

등장하게된 배경

Spring Web Socket, STOMP

KB국민은행

HTTP 통신 특징

- 비연결성 (connectionless) : 연결을 맺고 요청을 하고 응답을 받으면 연결을 끊김
- 무상태성 (stateless) : 서버가 클라이언트의 상태를 가지고 있지 않음 이걸 보완하는게 세션
- 단방향 통신 클라한번 요청 서버 한번 응답 끝. 그리고 서버가 먼저 시작 X

→ 채팅과 같은 실시간 통신에 적합하지 않음

Spring Web Socket, STOMP

연결자체는 클라이언트가 해야하긴함. 통신은 양쪽에서 할 수 있어.
채널 만드는 것은 클라이언트가 해야하긴해.

- **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

ws 연결:
http로 연결 요청.하지만 헤더에 나 이제 http프로토콜이아닌 websocket프로토콜로 업그레이드해줘.
서버가 ws protocol로 바꿔 응답함
그다음부터는 swprotocol

Spring Web Socket, STOMP

웹소켓 프로토콜

요청

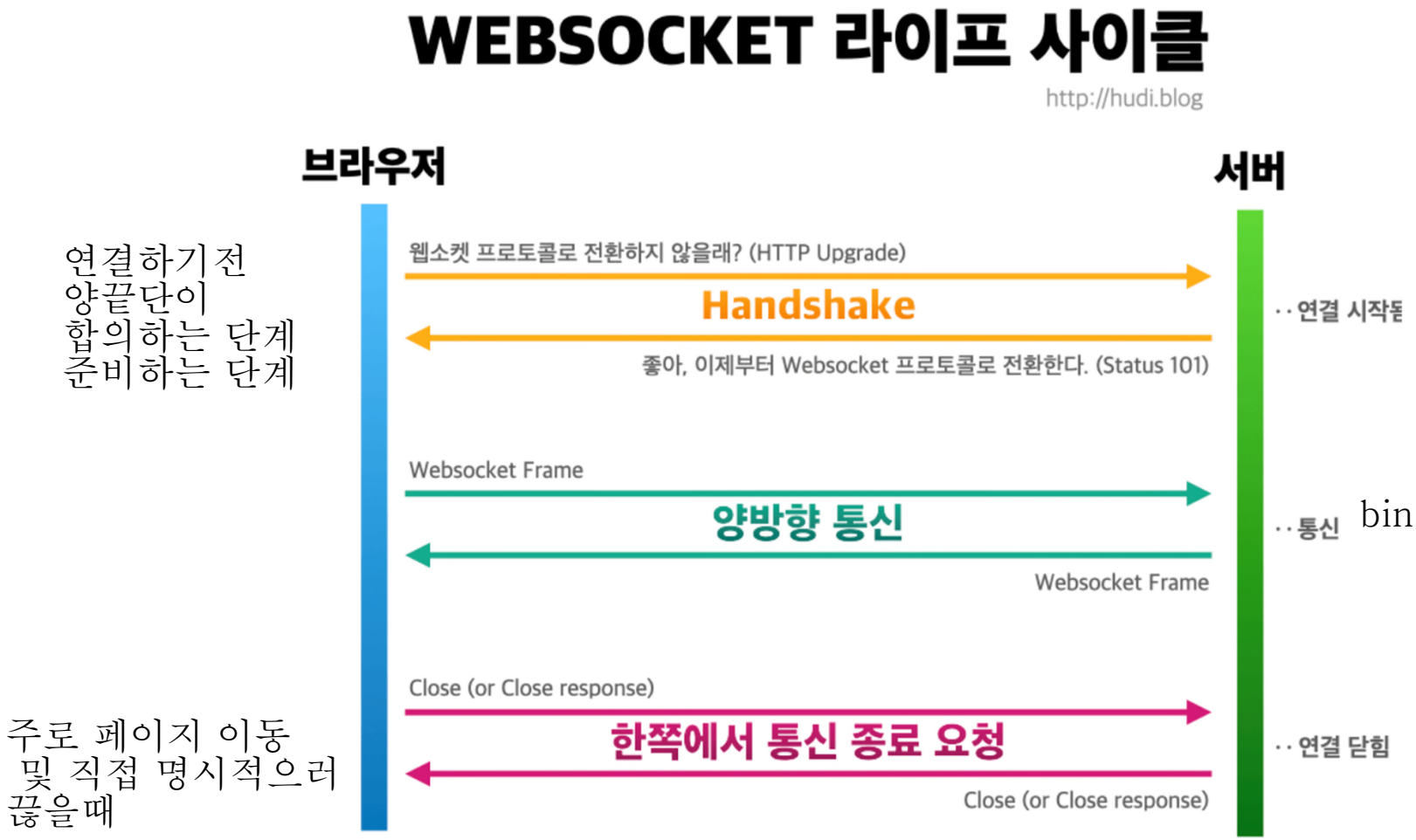
GET /spring-websocket-portfolio/portfolio HTTP/1.1
Host: localhost:8080
Upgrade: websocket ← Upgrade 헤더
Connection: Upgrade ← Upgrade connection 사용
Sec-WebSocket-Key: Uc9l9TMkWGbHFD2qnFHltg==
Sec-WebSocket-Protocol: v10.stomp, v11.stomp
Sec-WebSocket-Version: 13
Origin: http://localhost:8080
Upgrade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

별도의 페이지 이동이나
명시적인 disconet하기 전가지

연결이 유지된다!



연결하기 전
양끝단이
합의하는 단계
준비하는 단계

binary text로 할거냐

주로 페이지 이동
및 직접 명시적으로
끊을때

원래는 웹소켓에 대한 코드를 개발자가 다짰어야 함

subscribe 패턴. 활용함.

정보제공자는 누가 수신할지
몰라.

수신자(구독자)
등록만 해두면 알아서 메세지가
날라옴,

정보제공자를 구독해놓으면
메세지 발생시
알아서 구독자에게

Spring Web Socket, STOMP

KB국민은행

STOMP: Simple Text Oriented Messaging Protocol

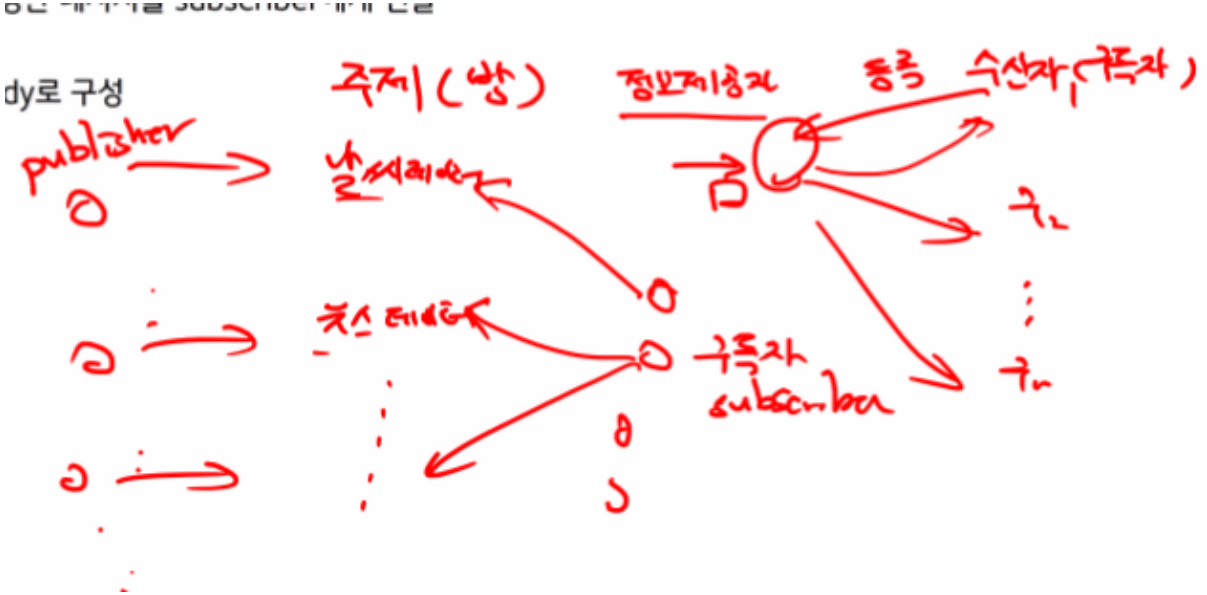
- 간단한 메시지를 전송하기 위한 프로토콜
- 메시지 브로커와 publisher – subscriber 방식을 사용
  - publisher: 메시지 전송자
  - subscriber: 메시지 수신자
  - broker: publisher가 발행한 메시지를 subscriber에게 전달
- frame 기반 프로토콜
  - command, header, body로 구성

구독자가 어떤 주제에 관심이 있다고
알리는 것은
구독자가 해당 방에 들어가는 것.
publisher는 주제 방에 들어가 있는
구독자들에게 전파.
그렇다면 방은 무라고 부를까
broker라고 한다. 방관리자.

이거는 어차피
스톰프
프로토콜이
알아서
할일이고

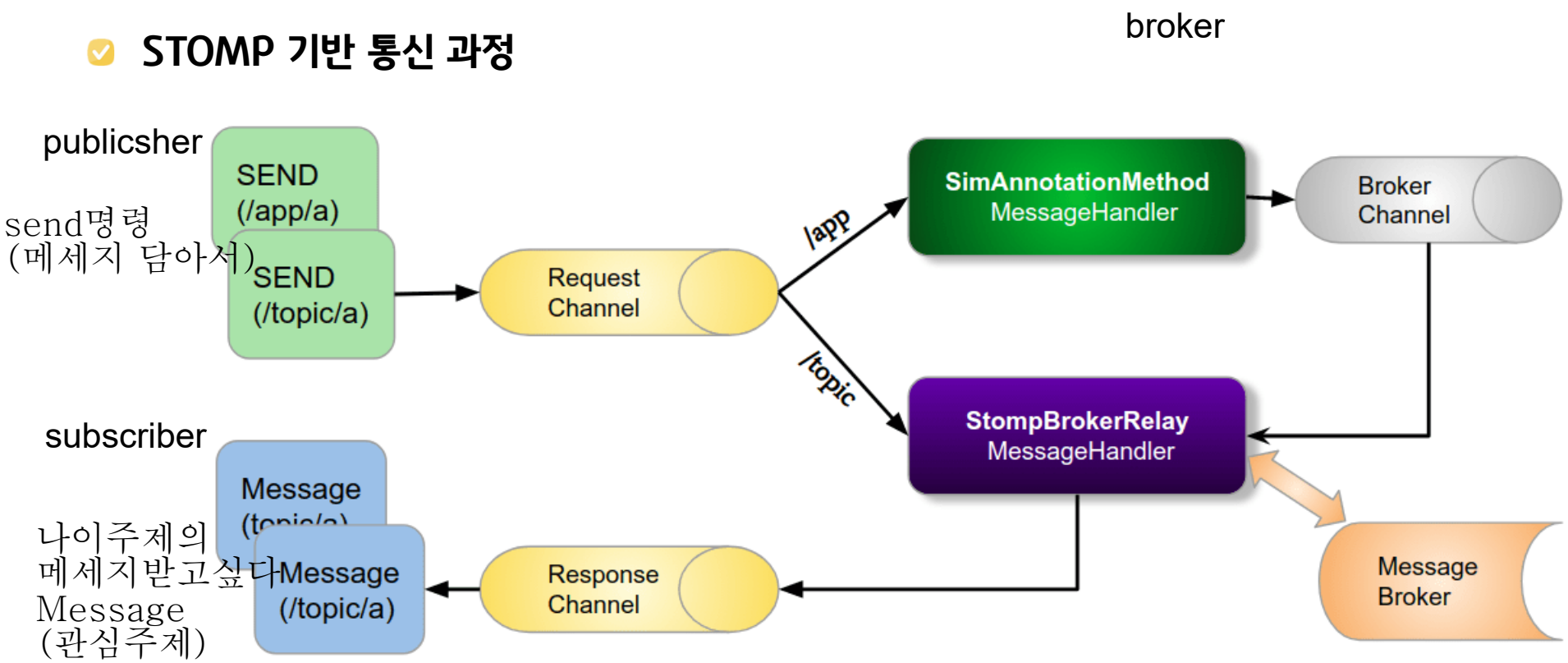
<STOMP frame 구조>
COMMAND
header1:value1
header2:value2
Body^@

우리는 바디의 내용을
뭘 넣을지
JSON타입임



Spring Web Socket, STOMP

STOMP 기반 통신 과정



브로커에 다가 토픽문자열 등록
/news
/chats
/room1 /room2
이렇게 등록. 미리하든 도적이든

- topic

Spring Web Socket, STOMP

build.gradle

둘만 잇었다면 개발자가 처음부터 다해야함

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

챗 관련된 것은 stomp사용이 대세

Spring Web Socket, STOMP 브로커에게 알릴 설정

메시지 브로커 설정

WebSocketMessageBrokerConfigurer 인터페이스 구현

- @EnableWebSocketMessageBroker 어노테이션으로 WebSocketMessageBroker 활성화
- 구현 메서드

void configureMessageBroker(MessageBrokerRegistry config)

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

void registerStompEndpoints(StompEndpointRegistry registry)

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

접속할때 사용할 url

클라이언트 쪽에서 접속하려할때 이런형식으로해야함

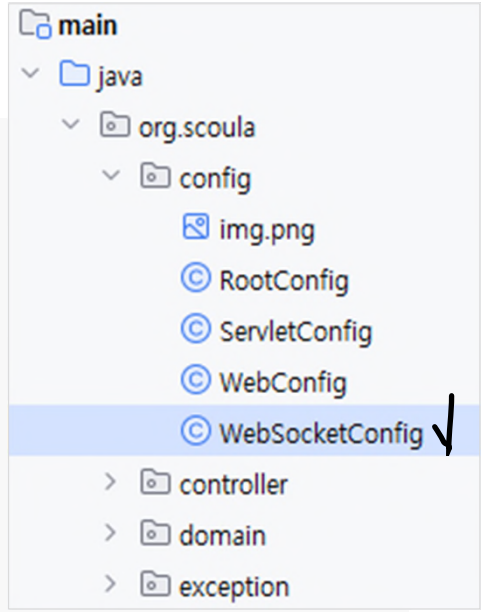
config.WebSocketConfig

템플릿에 추가할 내용

```
@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
 }

}
```

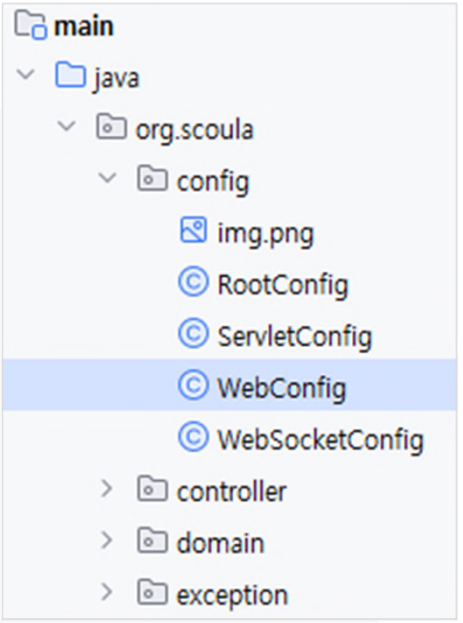
origin이 달라도
웹소켓 연결을 허용하겠다 라는 설정

/topic 구독시
/app 클라발행시
/chat-app 접속엔드포인트



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 };
   }
  ...

}
```



운영은 컨트롤러가 담당!. stomp컨트롤러가 담당

Spring Web Socket, STOMP

어떤 요청이 아닌 메세지냐?

Stomp 컨트롤러

토픽 경로. 해당 경로로 메세지를 보내면. 어떻게 하겠다!

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

/app 제외한 경로를 메시지경로로 지정해라.

전송한 메세지는 DI로 전달됨

메세지를 수신했는데 이 메세지를 누구한테 보낼 것인가

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

publisher전송한 메세지를 매핑해서 브로커가 수신함.
그럼 브로커가 누구에게 전송할거냐 다른말로 subscriber가 누구냐?
그것을 지정하는게 sendto 토픽에 쏴서
해당 토픽을 구독한 애들한테 보낸다.

메시지

- GreetingMessage 입장 메시지
- ChatMessage 채팅 문자열 메시지

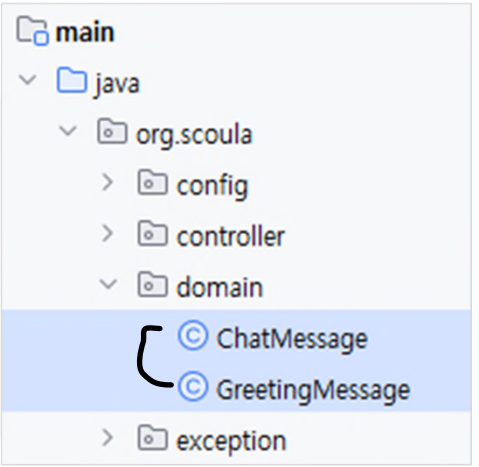
메세지 DTO만들기

Spring Web Socket, STOMP

domain.GreetingMessage

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

입장 인사 메세지

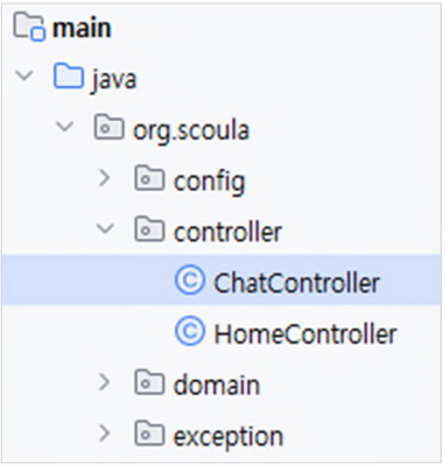


domain.ChatMessage

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

송신자와 내용

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;
  }
}
```

"/app/hello 메세지 내용을 받고

해당 토픽 경로의 구독자들에게 보내겠다.

request body 어노텡션을 붙여야 했지만 디폴트가 jSON이므로 굳이 안붙임

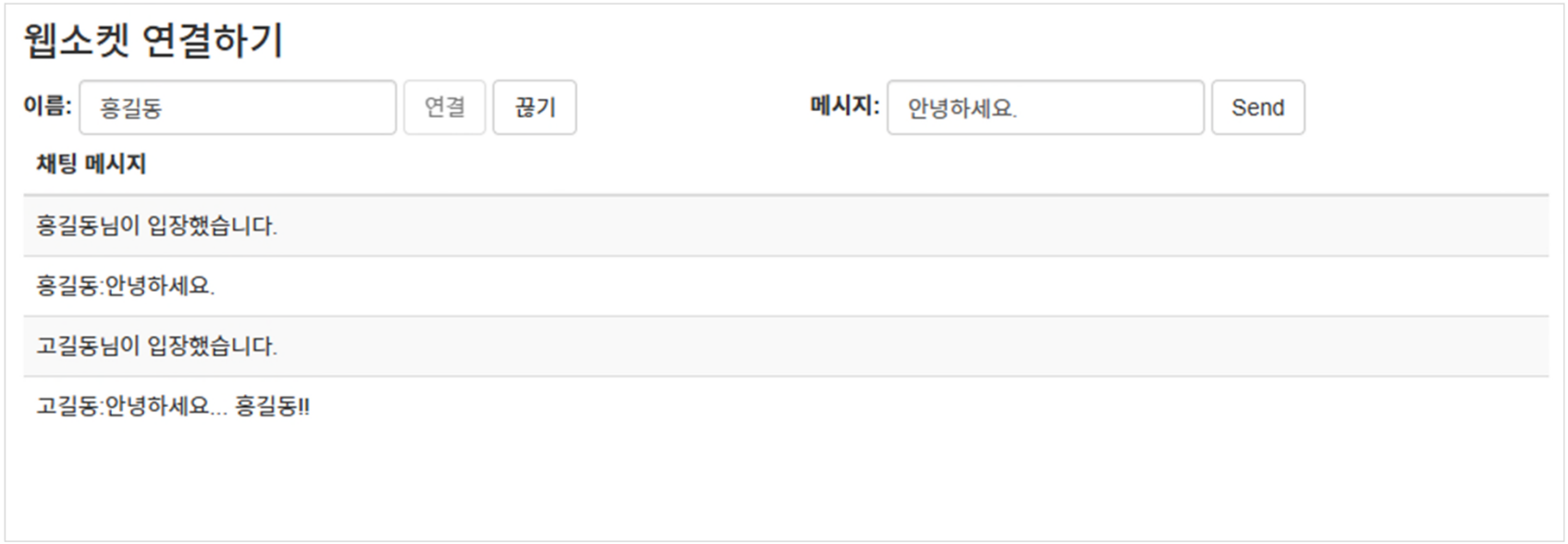
리턴 값을 지정한 토픽의 구독자들에게 전송

/app/chat로 메세지 내용을 받고

/topic/chat 의 구독자들에게 전송

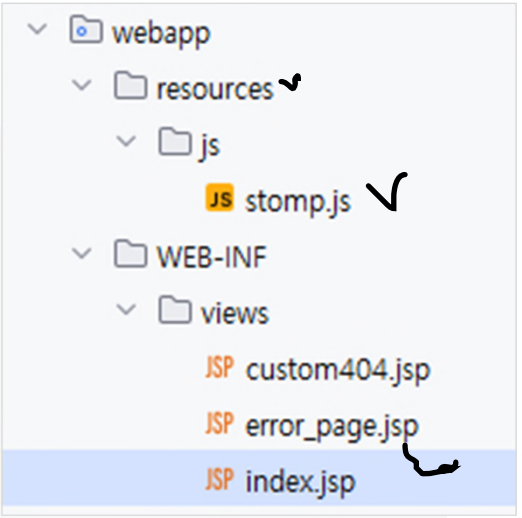
Spring Web Socket, STOMP

- **클라이언트** 클라가 할게 많아. 접속처리, ui작업, 등등
  - 화면



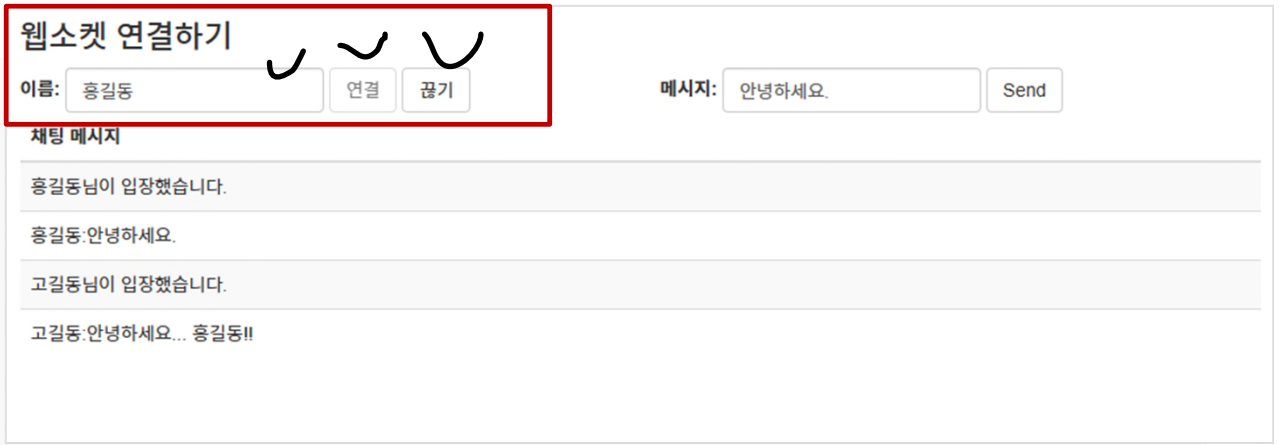
  - Stomp 라이브러리 CDN
    - https://cdn.jsdelivr.net/npm/@stomp/stompjs@7.0.0/bundles/stomp.umd.min.js

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

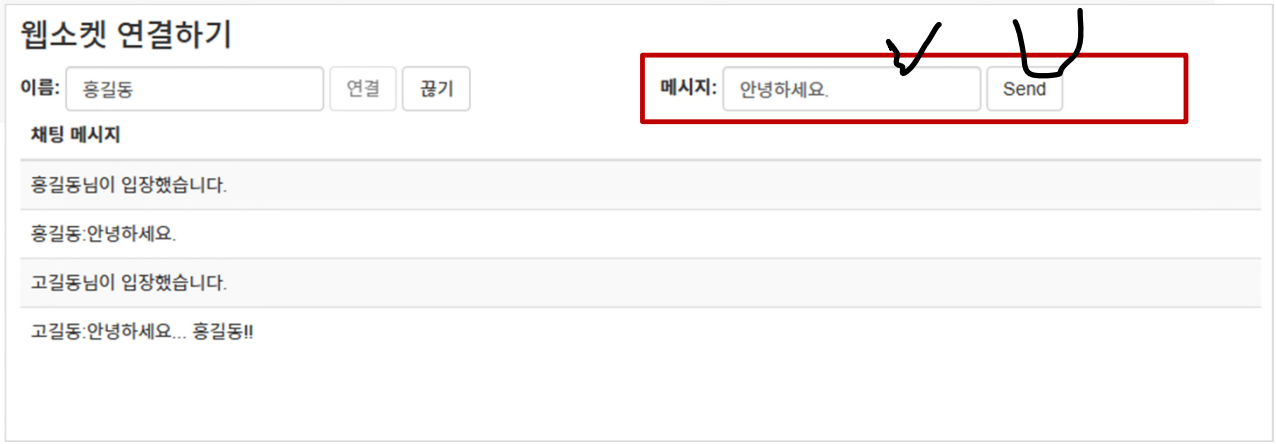



```
<body>
<div id="main-content" class="container">
 <h3>웹소켓 연결하기</h3>
 <div class="row">
  <div class="col-md-6">
   <form class="form-inline">
    <div class="form-group">
     <div class="form-group">
      <label for="name">이름: </label>
      <input type="text" id="name" class="form-control" placeholder="이름을 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>
```

```
<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>
```

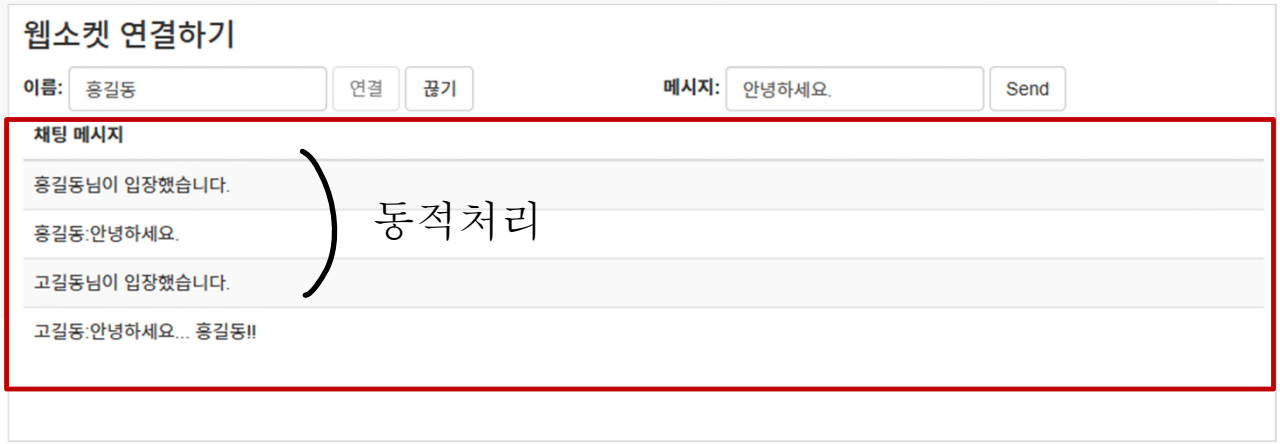
index.jsp

```
<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>
```



Spring Web Socket, STOMP

Stomp 라이브러리

StompClient

- StompJs.Client 클래스로 생성

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

옵션객체 == 설정객체

연결 관리 메서드

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

주요 이벤트 핸들러

인사메세지 전송. 수신토픽등록하는 동작

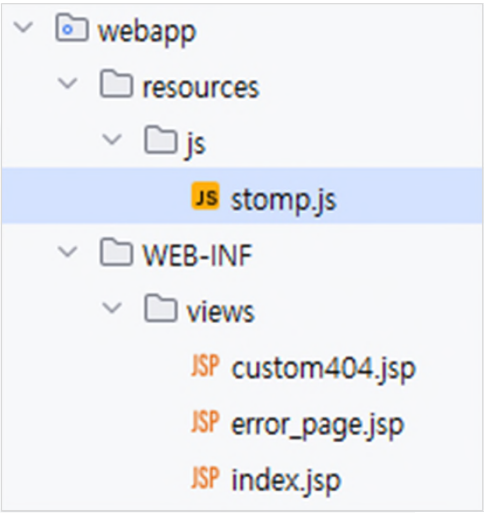
- onConnect 연결 성공시 콜백 등록
- onWebSocketError웹 소켓 에러 발생시 콜백 등록
- onStompError Stomp 에러 발생시 콜백 등록

Spring Web Socket, STOMP

- **구독할 토픽 등록하기** 수신 == 구독
  - .subscribe(구독 토픽 문자열, 수신 핸들러 함수)
    어떤 토픽에 관심이 있는지 그때 이 함수 호출해줘 등록
- **메시지 발행(전송) 하기**
  - .publish({ destination: 토픽 문자열, body: 전송할 메시지 })
    목적지 메세지

resources/js/stomp.js

이것도 템플릿화 할만해




```
// 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);
};
```

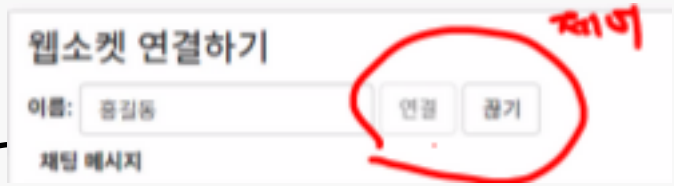
접속 엔트포인트 필드가 있는
설정 객체 생성자에 넘겨주며 생성

Spring Web Socket, STOMP

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}`);
    });
```

UI업데이트



ui활성화 업데이트

@SendTo에 맞춤. sendto와 같으면 해당 메소드가 동작함

메세지 수신처리

테이블에 행추가 뒤에 정의 나와있으음

해당 토픽에 관심있고 그 토픽에 메세지 왔을때 내가 등록한 콜백함수를 호출해줘

resources/js/stomp.js

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

};

// 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 = '';
}
```

입력한 사용자 이름 추출

전송

@MessageMapping에 맞춰 단 /app빼고 연동된 컨트롤러의 메서드 호출됨

// GreetingMessage에 대응

전송할 메세지

버튼 비활성화 UI

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
    });
}
```

@MEssageMapping 에맞추기 /app빼고

보낼 메세지

Spring Web Socket, STOMP

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());
    }
});
```

번외
서버가 클라에게 단방향으로
서버가 주도하에 송신하는거.

대표적인 시나리오 알림 메세지.

이런건 웹소켓이 하는거 아니야
SSE기술이다. server send event
이것도 스프링에 준비가 되어있따
stopmq보단 간단함

웹소켓은 양방향.

subscribe는 sendto에 맞추고

publish는 ㅡmessagemapping에맞추고

우리는 나중에 vue3 랑 하기에
js에서 썻던 코드는 사용안함
vue3에서 stomp와 맞는 라이브러리를 쓰면된다.

매커니즘은 같다.