Web-Socket

웹소켓은 ws 프로토콜을 기반으로 포트 80을 사용하고 클라이언트와 서버 사이의 지속적인 양방향 연결 스트림을 만들어 주는 기술이다. TCP에 의존하며 HTTP와 마찬가지로 보안적인 측면을 고려한다면 wss를 사용하면 된다. 하지만 기존 TCP Socket과 다른 점이 하나 있다면 http requeste를 통해 handshaking 과정을 통해 이루어진다는 점이다. 그래서 기존 80이나 443 포트로 접속을 하므로 추가로 방화벽을 열지 않고도 양방향 통신이 가능하고 CORS 적용, 인증 등의 과정을 기존과 동일하게 가져갈 수 있다는 장점이 있다.

즉, HTTP의 HandShaking 과정으로 통신을 하며 Binary Data만 통신할 수 있는 TCP와 달리 Text까지 통신할 수 있다. 또한, 기존 포트로 접속할 경우 방화벽, CORS, 인증 등의 방법을 새로 구현할 필요가 없다는 점이 매력적이다.

Spring WebSocket

ref.

'Websocket + STOMP' 태그의 글 목록
레퍼런스에 따르면, websocket은 웹 브라우저(클라이언트)와 서버간의 full-duplex(양방향), bi-directional(전이중적), persistent connection(지속적인 연결) 의 특징을 갖는 프로토콜이라고 규정한다. 여기서 핵심은 클라이언트와 서버간의 지속적인 연결을 한다는 점이다.
https://swiftymind.tistory.com/tag/Websocket%20%2B%20STOMP

스프링 웹 소켓은 Spring 4.0 이상 지원하며 Maven 3.2+, gradle 4+, jdk8 이상 필요하다. 스프링에서는 웹 소켓을 사용하는 방식이 크게 2가지로 구분된다고 한다.

1) Websocket Data 직접 처리

Socket.io

node.js 기반으로 만들어진 기술로 자체 스펙으로 'socket.io' 서버를 구축하고 클라이언트와 브라우저에 구애받지 않고 실시간 통신이 가능하게 만든 오픈소스 라이브러리다. 사용언어가 JavaScript이므로 자바 개발자들은 다른 방법을 사용하는 것을 추천한다고 한다. (하지만, 자바로 바꿔서 사용하는 방법이 있지만 많이 추천하지는 않는다고 한다.)

SockJS

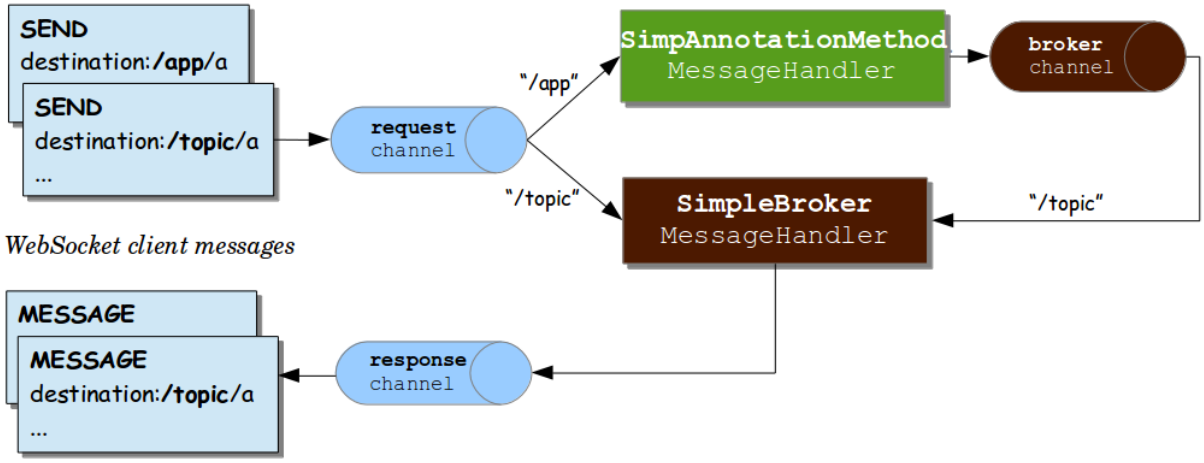
SpringFrameWork에서 지원하는 WebSocket 라이브러리이다. 'socket.io'와 마찬가지로 자체 서버를 구축해 브라우저에 구애받지 않는다. 또한, 서버 개발 시 스프링 설정을 통해 일반 웹소켓으로 통신할지 SockJS 호환으로 통신할지 결정할 수 있다고 한다.

```
@Configuration
@EnableWebSocket
public class WebSocketConfig implements WebSocketConfigurer {
    @Override
    public void registerWebSocketHandlers(WebSocketHandlerRegistry registry) {
//        WebSocket을 사용할 수없는 경우 대체 전송을 사용할 수 있도록 SockJS 폴백 옵션을 활성화합니다.
//        SockJS 클라이언트는 "/ws"에 연결하여 사용 가능한 최상의 전송 (websocket, xhr-streaming, xhr-polling 등)을 시도.
        registry.addHandler(new WsTranportHandler(), "/ws").setAllowedOrigins("*").withSockJS();
    }
}
```

```
@Component
public class WsTranportHandler extends TextWebSocketHandler {

    // connection이 맺어진 후 실행된다
    @Override
    public void afterConnectionEstablished(WebSocketSession session) throws Exception {
        System.err.println("session connected +=" + session);
    }
    // 메세지 수신
    @Override
    public void handleMessage(WebSocketSession session, WebSocketMessage<?> message) throws Exception {
        System.err.println("handle message +=" + session) + ", message=" + message);

        //echo Message
        session.sendMessage(message);

    }
    // transport 중 error
    @Override
    public void handleTransportError(WebSocketSession session, Throwable exception) throws Exception {
        System.err.println("transport error =" + session +", exception =" + exception);
    }
    // connection close
    @Override
    public void afterConnectionClosed(WebSocketSession session, CloseStatus closeStatus) throws Exception {

        System.err.println("session close -=" + session);

    }
}
```

2) STOMP 프로토콜을 사용하여 메시징 처리

STOMP는 (Simple Text Oriented Messaging Protocol)의 약자이며, 텍스트 기반의 프로토콜이다.



스프링 문서에서 WebSocket과 STOMP 프로토콜을 사용하는 방법은 위와 같은데, Spring 내부의 In Memory Blocker를 통해 메시지를 처리한다. 위 그림의 3가지를 주목해야 한다.

1) Receive Client

▼ 메시지를 받기 위해 특정 토픽이 사전에 서버에 Subscribe되어야 한다.

2) Send Client

▼ 서버와 연결된 클라이언트는 특정 PATH로 전달한다.

ex) /app/message, /ws/haha ...

3) Broker

▼ 메시지 브로커는 Kafka, RabbitMQ 등 오픈소스처럼 MQ 이며, Pub/Sub 모델을 따른다. 토픽에 따라 메시지를 전달해야 하는 사용자를 구분한다.

MQ(Message Queue) : 메시지 기반의 미들웨어로 메시지를 이용하여 여러 어플리케이션, 시스템, 서비스들을 연결해주는 솔루션이다.

▼ 연결된 클라이언트의 세션을 관리한다.

▼ 특정 토픽과 메시지를 Mapping 하여 토픽을 구독하는 세션에 존재하는 클라이언트에게 메시지를 전달한다.

설정

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

    //messageBroker config
    @Override
    public void configureMessageBroker(MessageBrokerRegistry config) {

        //in-memory message-broker, topic에 대한 prefix 설정
         config.enableSimpleBroker("/topic");
```

```
        //메세지를 수신하는 handler의 메세지 prefix 설정
        config.setApplicationDestinationPrefixes("/api");

    };

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
       registry.addEndpoint("/ws").setAllowedOrigins("*").withSockJS();
    }
}
```

Controller

```
@RestController
public class MessageHandleController {

    @MessageMapping("/echo")
    @SendTo("/topic/messages")
    public EchoMessage echo(String message) throws Exception {
        System.err.println(message);
        return  new EchoMessage(message,LocalDateTime.now());
    }
}
```

REFERENCE

1). 배달의 민족에서 WebSocket을 사용했을 때 브라우저 Delay에 관한 이슈 및 해결 방안

실시간 서비스 경험기(배달운영시스템) - 우아한형제들 기술 블로그

들어가기 앞서 이 글은 신기술 사용기 또는 소개가 아닌 실시간 서비스 즉 배민라이더스 BROS 1.0 을 개발 하면서 겪어왔던 다소 특별한 개발 및 운영 경험기 입니다. BROS 2.0이 나온 상황에서 1.0을 이야기 하는 것이 다소 순서가 맞지 않지만 그 때 당시 경험기를 남겨놓지 못 한 것을 이

https://woowabros.github.io/woowabros/2017/09/12/realtime-service.html