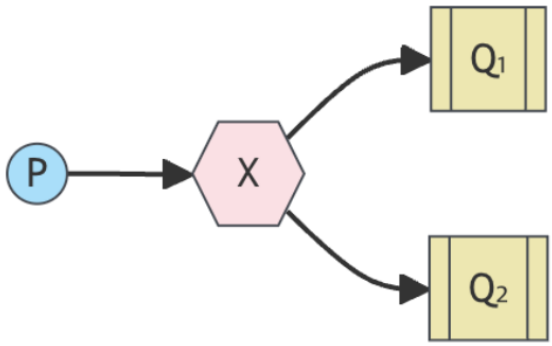
Pub/Sub 모델을 이용한 실시간 알림과 뉴스 구독 (WebSocket, STOMP 활용)

Publish / Subscribe 모델

Pub/Sub 은 메시지 발행(Publish)과 구독(Subscribe)의 개념을 기반으로 하는 메시징 패턴으로 메시지를 **중간 브로커**(Exchange)를 통해 **구독자들(Subscriber)**에게 메시지를 전달한다.

RabbitMQ에서는 Fanout Exchange를 통해 연결된 모든 Queue에게 메시지를 전달하므로 Binding(exchange와 큐와의 관계를 정의한 일종의 라우팅 테이블)을 통해 익스체인지를 연결하여 동시에 메시지를 받을 수 있다.



Pub/Sub 모델의 주요 특징

유연성과 확장성이 좋아 여러 subscriber를 쉽게 추가하여도 서로 독립적으로 동작이 가능하다.

1. **다대다 메시징**:

- 하나의 메시지가 여러 Subscriber에게 전달
- 메시지 복사가 이루어지므로 Subscriber는 동일한 메시지를 수신. 동일한 메시지가 여러 큐에 처리 되므로 중복 처리 로직이 필요할 수 있음

2. **구독자 독립성**:

- Publisher는 메시지가 어떤 Subscriber에게 전달될지 알 필요가 없음
- 메시지의 전달은 브로커가 처리

3. **비동기 메시징**:

- Publisher와 Subscriber는 서로 독립적으로 동작하며, 동시에 실행될 필요가 없음

4. **확장성**:

- 여러 Subscriber를 추가하거나 제거해도 시스템이 영향을 받지 않음

5. **구독 제어**:

- 구독자는 특정 조건(예: 라우팅 키, 토픽)을 기반으로 메시지를 필터링하여 수신할 수도 있다.
- Fanout Exchange는 모든 구독자에게 메시지를 브로드캐스트하는 반면(Routing Key는 필요하지 않음), Topic Exchange나 Direct Exchange는 메시지를 선택적으로 전달 가능
- 구독자가 많을수록 복잡도가 증가

튜토리얼 Step 3. 프로세스 - WebSocket과 Pub/Sub을 통한 실시간 웹 알림 구현하기

웹 페이지를 하나 만들고, WebSocket을 이용하여 API를 연동한 뒤, 서버에서 publish 한 메시지를 web 알림 영역에 실시간으로 표시한다.

프로젝트 구조는 아래와 같다.

실시간 웹 알림 Package Structure

```
src/main/resources/
├── static/      # CSS, JS, 이미지 파일을 보관
└── templates/   # Thymeleaf, HTML 템플릿 파일 보관
```

```
HelloMessageQueue/src/main/java/net/harunote/hellomessagequeue
├── RabbitMQConfig.java
├── NotificationController.java
├── NotificationPublisher.java
├── WebSocketConfig.java
└── NotificationSubscriber.java
```

build.gradle 에 thymeleaf 템플릿 엔진을 추가한 뒤 gradle refresh 실행

```
    implementation 'org.springframework.boot:spring-boot-starte
```

1. 클라이언트 페이지 연동 확인

- HomeController 작성
- templates/home.html 작성
- static/css/style.css 작성

http://localhost:8080/home 접속하여 첫 페이지 로드 확인

2. WebSocket을 이용한 Notification 구현

build.gradle에 websocket 디펜던시 추가

```
    implementation 'org.springframework.boot:spring-boot-starte
```

3. notification publisher/subscriber 개발 순서

- RabbitMQConfig ：3개의 Bean(Queue, Exchange, Binding)을 설정
- FanoutExchange 설정
- QueueName 설정

- BindingBuilder를 이용해 큐와 익스체인지 연결

4. Publisher

- RabbitTemplate을 이용해 convertAndSend

5. Subscriber

- RabbitListener로 큐 네임 지정
    - 내부적으로 Spring의 MessageListenerContainer를 사용하여 Queue를 지속적으로 모니터링
    - Spring이 메시지 수신을 자동화하고, 비동기적으로 처리. 코드가 간결
    - 메시지 수신, 메서드 호출과 변환, Ack 전송 등의 역할
- SimpMessageingTemplate을 통해 특정 경로에 메시지 전달
    - WebSocket 메시지 브로커와 통신하며, 클라이언트가 구독하는 특정 경로로 메시지를 전송
    - 특정 클라이언트나, 클라이언트 그룹에게 메시지 전송 가능
    - STOMP의 경로를 지정

```
simpMessagingTemplate.convertAndSend("/topic/notifications",
```

6. WebSocketConfig

- WebSocketMessageBroker 활성화
- WebSocketMessageBrokerConfigurer를 구현
    - Spring에서 WebSocket 메시지 브로커를 구성하기 위한 인터페이스로 웹소켓 연결, 메시지 브로커 설정 및 라우팅 등의 웹 소켓 관련 확장기능을 제공함
        - 주요 메서드
        - registerStompEndpoints(StompEndpointRegistry registry) // STOMP 엔드 포인트 등록

```
@Override
public void registerStompEndpoints(StompEndpointRegistry r
```

```
    registry.addEndpoint("/ws") // WebSocket 연결 엔드포인트
            .setAllowedOriginPatterns("*") // CORS 설정
            .withSockJS(); // SockJS 지원
}
```

- configureWebSocketTransport(WebSocketTransportRegistration registry) // 웹소켓 전송 설정

```
@Override
public void configureWebSocketTransport(WebSocketTransport
    registry.setMessageSizeLimit(2048); // 메시지 크기 제한 (
    registry.setSendTimeLimit(5 * 1000); // 전송 제한 시간 (
    registry.setSendBufferSizeLimit(512 * 1024); // 버퍼 크
}
```

- 기타 메시지 변환을 위한 컨버터나 메시지 전송 브로커 경로 등을 설정할 수 있음

- 클라이언트 구독 경로 설정
  - `config.enableSimpleBroker( "/topic" ); // 클라이언트 구독 경로`
- 서버의 발행 경로 설정
  - `config.setApplicationDestinationPrefixes( "/app" ); // 서버 발행 경로`
- WebSocket endpoint 설정
  - `registry.addEndpoint( "/ws" ).setAllowedOriginPatterns( "*" ).withSockJS(); // WebSocket 엔드포인트`

7. Controller 작성
8. HTML 작성

- 필요 라이브러리
  - sock.js : WebSocket 연결을 지원하지 않는 브라우저에서도 동작하도록 폴백(fallback) 기능을 제공하는 라이브러리로 아래와 같은 전송 타입을 지원
    - WebSocket

        - HTTP Streaming
        - HTTP Long Polling
    - stomp.js
        - STOMP(Simple/Streaming Text Oriented Messaging Protocol) 프로토콜을 지원하는 JavaScript 클라이언트 라이브러리
            - https://stomp.github.io/stomp-specification-1.2.html
            - 텍스트 기반 프로토콜로, 메시지의 유형, 내용을 정의하여 클라이언트와 서버 간 메시지를 주고 받음
            - RabbitMQ와 같은 브로커와 통신하는 데 사용
            - 각 프레임은 명령(Command), 헤더(Header), 바디(Body)로 구성
            - 이기종 시스템 간의 메시징에 유용

9. cURL 테스트 (http client)

- curl 호출 (zsh 표기상 !가 명령어라서 싱글 쿼테이션으로 표기)

```
curl -X POST 'http://localhost:8080/notifications' -H 'Content-T
```

10. html 영역에서 input 폼을 통해 서버의 /app 에 전송하여 실시간으로 반영되는지 기능 추가

✔️ 개발중에 발생한 에러 정리

클라이언트의 JSON 객체를 서버에 전송할 때 매핑되는 객체의 기본 생성자가 없을 경우에는 어떤 문제가 발생하는가?
관련 에러 : Could not read JSON: Cannot construct instance of `NotificationMessage` ... cannot deserialize from Object value
- Jackson 역직렬화 과정

Jackson이 JSON 데이터를 역직렬화(Deserialization)할 때 프로세스

1. JSON 데이터 조회
2. Jackson이 역직렬화 대상 클래스(여기서는 NotificationMessage)의 객체를 찾음 -> 객체를 생성 (생성시 기본 생성자 호출 new NotificationMessage(); // 없으면 error)
3. JSON 필드 이름과 클래스의 필드 이름을 매핑 -> Spring 에서 내부적으로 Jackson 을 사용해서 JSON 데이터를 NotificationMessage 객체로 필드 매핑
4. 기본 생성자가 없으면 Jackson은 객체를 생성할 수 없어서 에러가 발생 (Setter를 통해 JSON 필드값을 클래스 필드에 설정하므로 setter도 필수)
5. 필드를 final 로 선언할 경우 객체 생성시 초기화 해야 하므로 기본 생성자가 자동으로 생성되지 않아 MessageConversionException이 발생함

예외 케이스 검증

```
public class NotificationMessage {
    private final String message; // 'final' 키워드로 정의
}
```

- final 필드는 객체 생성 시 초기화 해야하므로 기본 생성자가 자동으로 생성되지 않음 (
  - final 필드는 반드시 선언과 동시에 초기화하거나 생성자를 통해 초기화 해야함
  - 한번 초기화 된 후에는 값을 변경할 수 없음
- setter가 없으므로 JSON 데이터에서 message 필드를 매핑할 수 없음

따라서 이 경우는 기본 생성자와 setter를 추가해서 해결하거나 JSON Parsing을 통해서 해결이 가능

혹은, @JsonCreator 와 @JsonProperty 로 특정 생성자를 호출하도록 명시해야함

```
@MessageMapping("/send")
public void handleNotification(String jsonMessage) {
    // JSON 파싱
    ObjectMapper objectMapper = new ObjectMapper();
    try {
        NotificationMessage message = objectMapper.readValue(jso
        System.out.println("Received message: " + message.getCon
    } catch (JsonProcessingException e) {
        System.err.println("Invalid JSON format: " + jsonMessage
```

```
    }
}
```

WebSocket

WebSocket은 **양방향 통신**을 가능하게 하는 표준 프로토콜(RFC 6455)로 클라이언트와 서버 간에 **실시간 데이터**를 주고받는 데 적합하다.

데이터를 프레임 단위로 전송하며, 오버헤드가 낮음.

- 기존의 HTTP 기반 통신과 달리, 연결이 초기화된 후에는 **상태를 유지**하며 데이터 교환이 가능
- ws://호스트:포트/경로 형태로 작성 (ws://example.com/chat)

특징

1. 실시간성

- 클라이언트와 서버 간 실시간으로 데이터를 주고받을 수 있음
- 예: (주식거래, 채팅, 실시간 알림)

2. 효율성:

- HTTP 기반 폴링(Polling)보다 네트워크 자원과 대역폭을 절약.
- 연결이 열려 있는 동안 별도의 요청/응답 없이 데이터를 지속적으로 전송 가능.

3. 양방향 통신 (Full-Duplex):

- 클라이언트에서 서버로 요청을 보낼 필요 없이, 서버가 클라이언트로 데이터를 푸시.

4. 낮은 지연 시간:

- 한 번 연결이 설정되면 데이터 전송 속도가 매우 빠름

5. 상태 유지 (Persistent Connection):

- 연결이 끊기지 않는 동안 클라이언트의 상태를 서버에서 유지 가능.

NotificationPublisher 부터의 전체 흐름

Binding:

- RabbitMQ에서 **Exchange**와 **Queue** 간의 관계를 정의
- 메시지가 Exchange에 도착했을 때, 어떤 Queue로 전달할지를 결정

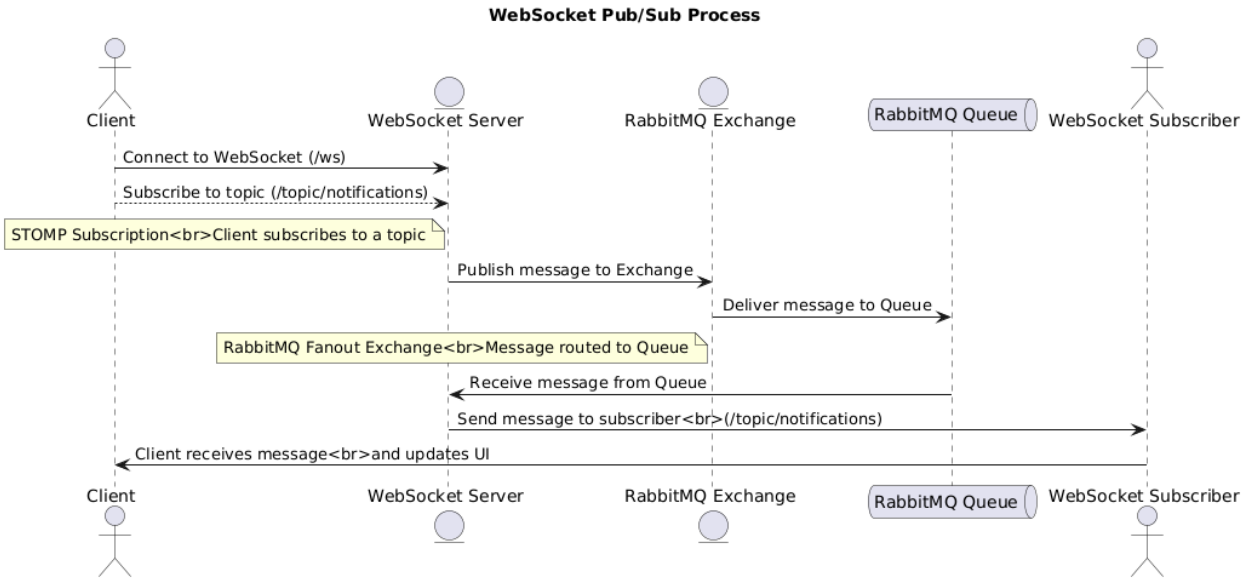
BindingBuilder.bind(notificationQueue).to(fanoutExchange):

- notificationQueue를 fanoutExchange에 연결
- Fanout Exchange의 메시지를 해당 Queue로 전달하도록 설정

publisher 는 exchange 에 메시지를 보내고 exchange는 binding 으로 인해 queue와 연결되어 있으므로

subscriber에서 RabbitListener 에 의해 QUEUE_NAME을 바라보다가 exchange에 메시지가 도착하면 Queue로 발행되고 이 Queue 가 메시지를 수신

즉, Publisher → Fanout Exchange → 모든 연결된 Queue라는 흐름으로 메시지가 전달하고 이를 시퀀스 다이어그램으로 정리하면 아래와 같다.



pub/sub의 활용 영역 예시

- 실시간 채팅
- 실시간 알림 (실시간 상태 변경 및 수신)
- IoT 데이터 스트리밍 (중앙 서버에 발행)
- 위치 추적, 이벤트 트래킹, 분산 이벤트 통신 (메시지 전달)
- SNS의 좋아요, 새 게시물, 댓글, 팔로워 등

튜토리얼 Step 4. 여러 큐를 소비하는 Fanout Exchange 예제 (관심사 기반의 뉴스 레터 발행/구독 모델)

- 개발 뉴스 레터 구독시 관심사 체크 (Java/Spring/Vue ...)
- 각각의 관심사별 Exchange 연결에 따른 메시지 발행과 구독

개발 프로세스

1. 3개의 Queue Bean(Java, Spring, Vue 뉴스 큐 선언), 1개의 Exchange Bean(FanoutExchange), 3개의 큐 Binding에 대한 bind().to() 설정
    - 매개변수 이름은 실제로 주입되는 빈 이름과 **일치하지 않아도 동작하지만, 여러개의 빈이 존재하면 이름으로 구분 지어야 함**
        - Spring은 **타입 기반으로 매칭**하며, 충돌이 없으면 주입 성공.
        - 동일 타입의 빈이 여러 개 존재할 경우는 @Qualifier로 명확히 지정.

        ```
        @Bean
        public Binding javaBinding(@Qualifier("javaQueue") Queu
            return BindingBuilder.bind(queue).to(fanoutExchange
        }
        ```

        - 또는 매개변수 이름을 빈 이름과 동일하게 설정하여 가독성을 높임.
2. 컨슈머 (subscriber) 생성
3. 메시지 발행 (publisher) 생성
4. HTML 화면 변경 후 WebSocket을 통한 호출 테스트

5. REST API 작성 & 테스트도 가능
    - REST Controller 작성 뉴스 타입별 파라미터를 받아 publisher 호출
    - cURL 호출 테스트

```
curl -X POST "http://localhost:8080/news/api/publish?newsType=ja
curl -X POST "http://localhost:8080/news/api/publish?newsType=sp
curl -X POST "http://localhost:8080/news/api/publish?newsType=vu
```