**Http의 한계**

1 요청-응답 기반이이다 : 클라가 요청해야만 서버가 응답할 수 있다. 클라가 요청하기 전까진 서버는 뭘 할 수 없음.

2 비연결성 : 요청마다 TCP 연결이 끊겼다 다시 열린다.

3 일방향 통신 : 실질적으로 클라에서 서버 방향으로 통신한다. 반대도 하지만 여튼 단방향임.

🡺실시간 알림, 실시간 데이터, 채팅, 온라인 게임 서비스처럼 서버가 먼저 데이터를 보내고 실시간 양방향 통신하는 상황에 적합하지 않다.

**WebSocket의 특징**

1 양방향 통신이다 : 클라와 서버가 자유롭게 메시지를 주고 받을 수 있다.

2 지속적 연결 : 한 번 연결되면 끊기지 않고 계속 유지. 다시 연결해야하는 오버헤드 없음. 그로 인한 딜레이도 없음!

3 낮은 오버헤드 : http 헤더 없이 프레임으로 통신하므로 네트워크 효율이 좋음. 그렇구만.

4 이벤트 기반 메시징 가능 : 이벤트 발생시 실시간 메시지 전송 기능

**🡺 spring에서 WebSocket을 사용하는 이유**

1 실시간 데이터 처리 : 채팅, 알림, 주식 시세, 게임, 실시간 모니터링 등 실시간 서비스에 적합

2 효율적인 서버 리소스 사용

3 양방향 통신이 필요할 때 : 서버🡪클라, 클라🡪서버 모두 자유롭게 가능. 서버에서 먼저 통신 할 수 있다.

4 STOMP 프로토콜과 메시지 브로커를 활용하면 추상화 수준이 높고 관리를 쉽게 할 수 있다.

5 보안/확장성 좋은 구조 : 인증 처리, 사용자 구분, 토픽 분리, 토픽 구독 등 편리하게 구성할 수 있다.

🡺서버가 클라에게 실시간으로 데이터를 푸시할 수 있게 해주고/클라도 서버에게 “언제든지” 메시지를 전송할 수 있게 해준다.

웹소켓 프로토콜

클라이언트 주도 양방향 통신 : 채널 생성과 연결자체는 클라가. 통신은 양쪽에서.

http에서 동작 가능하게 디자인 됐고 80, 443포트 사용.

Handshake : 일반 http 요청에 upgrade 헤더를 포함한 request를 전송하면 WebSocket 프로토콜로 변환되며 웹소켓 통신이 시작된다.

Close : 양방향 통신하다가 한쪽으로 명시적으로 직접 끊으면 연결이 끊긴다.

**WebSocket이 활용되는 분야 (대표 사례)**

실시간 채팅 : 카톡, 슬랙, 디코 … : 사용자 간 메시지를 지연없이 주고 받아

실시간 알림 : 이메일 알림, 웹 알림, 푸시 알림 : 서버에서 이벤트 발생 시 클라에게 즉시 전송

데이터 스트리밍 : 주식 시세, 환율, 비트 코인 등 // 동영상 스트리밍은 다른 기술.

온라인 게임

실시간 위치 : 실시간 차량 위치 추적, 물류/배송 추적 시스템/친구 위치 공유 앱 : 실시간 위치 반영

콜라보 도구 : 실시간 문서 편집(google docs) : 여러 사용자의 편집 정보를 실시간 반영

**🡺WebSocket을 써야 하는 상황 판단 기준! 아래 중에 하나라도 충족되면 WebSocket고려**

서버가 먼저 클라에게 데이터를 보내야 한다. 예) 알림, 방송

데이터가 자주/빠르게 바뀌며 이를 즉시 반영해야 한다. 예) 대시보드

지연에 민감하다. 예) 온라인 게임

사용자가 서로에게 실시간으로 데이터를 주고 받는다. 예) 채팅

**번외1) SSE == Server-Sent Events**

서버에서 클라로 단방향 실시간으로 http를 통해 데이터를 푸시할 수 있는 표준 기술.

websocket까지 필요하지 않은 경우에 매우 유용. **서버🡪클라.**

EventSource객체로 쉽게 구현 가능하다.

예시) 실시간 알림, 뉴스 피드

**번외2) 실시간 영상 스트리밍**

SSE나 WebSocket을 사용하지 않고 HttpLiveStreaming이나 MPEG-DASH 기술을 사용한다.

**WebSocket의 한계**

연결은 잘 만들어주지만, 그 위에서 직접 메시지를 주고 받을 때는 :

어떤 메시지가 어떤 목적(구독, 전송 등)인지 명확하지 않고, 메시지 형식을 개발자가 직접 설계해야하고, 메시지 라우팅(구독자 관리, 주제별 발행 등)을 모두 수동으로 구현해야 한다.

🡺즉, WebSocket은 파이프는 만들어주지만, 그 파이프 안의 통신 규칙은 알아서 만들어야 합니다.

**STOMP**

STOMP : Simple Text Oriented Messaging Protocol : WebSocket같은 이진 기반 전송 기술 위에서 텍스트 기반 메시징을 쉽고 구조적으로 하기 위해서 사용한다.

WebSocket위에서 구조화된 메시지를 주고 받기 위한 메시징 프로토콜이다.

기능들 :

메시지 목적 정의 🡺 명령 프레임으로 지원

메시지 라우팅 🡺 destination으로 자동화

클라이언트 관리 🡺 클라이언트 구독/해지 모델 지원. destination으로 자동화.

에러 처리 🡺 ERROR 프레임 제공

메타데이터 🡺 헤더 기반 지원

텍스트 명령어로 처리.

🡺 STOMP는 WebSocket통신을 HTTP처럼 구조화된 메시징 방식을 만들기 위해 존재.  
🡺 STOMP는 WebSocket의 단순한 통신 채널에 메시지 목적, 라우팅, 구독, 헤더 등을 명확히 정한 프로토콜을 입힌 것.

🡺 STOMP는 복잡한 메시징 시스템을 안정적이고 일관된 방식으로 구현하게 해주는 것.

**STOMP 개발자 생산성 향상**

클라이언트는 단순히 SUBSCRIBE, SEND명령만 보내고 되고, 서버(spring 등)은 목적지(destination)만 매핑하면  
되며 서로 메시지 구조를 맞추느라 생기는 혼란이 줄어든다.

**방식 :**

클라 (프론트에서 js로)

/브로커대상/토픽명 을 구독! //subscribe( “/브로커대상/토픽명“ );

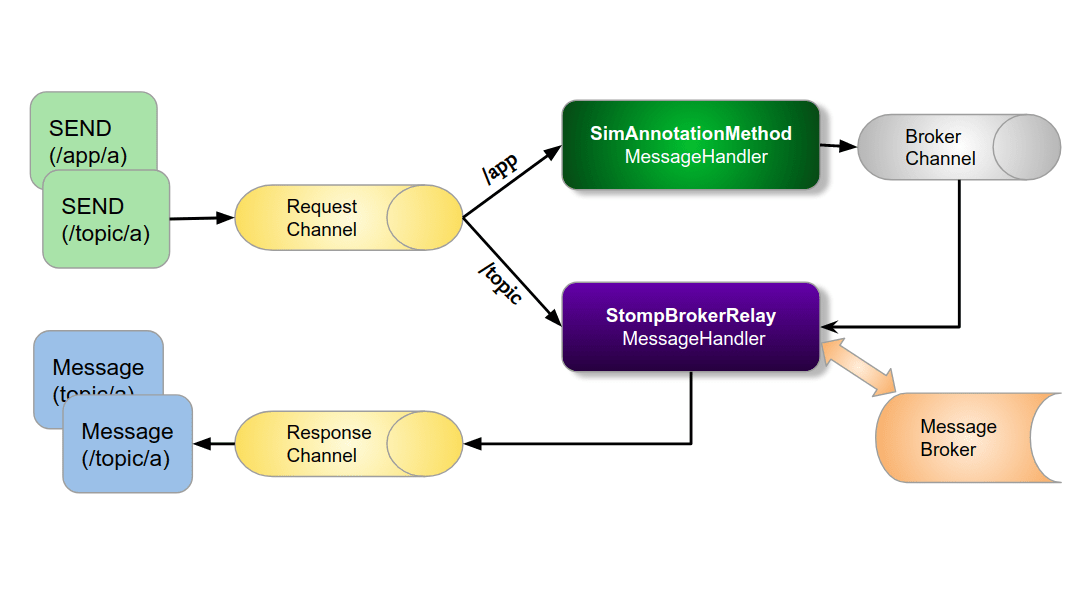
/컨트롤러매핑destination/컨트롤러메서드매핑경로으로 내용을 보내   
//send( “/컨트롤러매핑destination/컨트롤러메서드매핑경로”, 내용 );

서버 (백에서 spring controller로)

@MessageMapping( “/컨트롤러매핑destination/컨트롤러메서드매핑경로” )  
//연관된 메서드는 클라가 해당 경로로 보내는 메시지를 처리함

@SendTo( “/브로커대상/토픽명” ) //해당 경로를 구독한 클라들에게 연관된 메서드가 메시지를 브로드 캐스트함.

**구도 정리**



특정 /topic을 구독한 사람이

/topic으로 시작하는 장소로 메시지를 보내면 message broker가 ‘바로’ 해당 장소를 구독한 사람들에게 메시지 전달.

/app으로 시작하는 곳으로 메시지를 보내면 @MessageMapping과 연동된 기능에서 메시지를 가공하고 message broker가 @SendTo의 값인 장소를 구독한 사람들에게 메시지 전달.

구성 준비 :

Config>WebSocketConfig.java  
@Configuration  
@EnableWebSocketMessageBroker //웹소켓 메시지 브로커 활성화   
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer{  
  
}