SGS – Smooth

Smooth

backend – 박병찬

코드 리뷰 가이드 목록

서버리스트 및 요청 서버

유저 서버

서버 링크

API Gateway

서버 링크

채팅 서버

서버 링크 (핵심 서버) 코드 리뷰 가이드

상태 관리 서버

서버 링크 (핵심 서버) 코드 리뷰 가이드

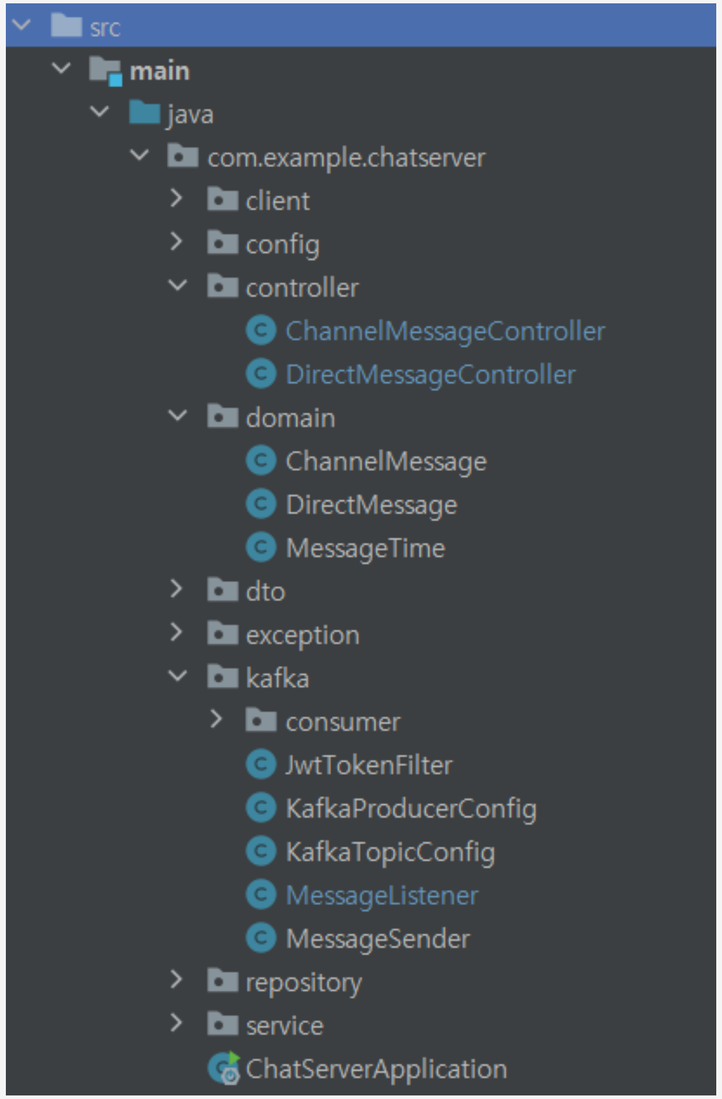
채팅 서버

채팅 서버 링크

채팅 서버 기술 스택

Java 11	java 8 이상 stream을 지원함, 작업 환경이 java 11 개발 환경에 익숙함
Spring Boot (2.6.2)	spring 설정을 자동으로 처리, jar 파일로 배포 가능, Spring boot 개발 환경에 익숙함
Spring Data MongoDB	자유로운 데이터 구조 생성 가능, RDB 보다 더 빠른 읽기 성능
Spring Integration	spring integration tcp 통신 이용
WebSocket, STOMP, SockJS	TCP지만 80,443 사용, 메세지 타입 정의 필요 없음, 세션 관리 대체, sockjs로 websocket 예외 상황 처리
Kafka, Zookeeper	pub/sub 구조 메세지 브로커, Disk에 메시지를 저장하고 유지, 확장성 용이
Spring Data Redis	api call을 줄이기 위한 비즈니스 로직용 휘발성 메모리

채팅 서버 구조



client – feignClient를 위한 api 정의

config – websocket, tcp 통신을 위한 config

controller – 다이렉트 채팅, 커뮤니티 채팅 컨트롤러

domain – 다이렉트 채팅, 커뮤니티 채팅, 채팅방 접속 기록 document들

kafka – 카프카 설정 파일, sender, listener

service – 채팅 관련 비즈니스 로직

repository – MongoRepository 이용한 repository들

WebSocketConfig(링크)

WebSocket Interceptor(링크) - preSend와 postSend를 나눠 웹 소켓 연결 구분

```java
@Override
public Message<?> preSend(Message<?> message, MessageChannel channel) {
    StompHeaderAccessor headerAccessor = StompHeaderAccessor.wrap(message);
    if (StompCommand.CONNECT.equals(headerAccessor.getCommand())) {
        if (!jwtTokenFilter.isJwtValid(Objects.requireNonNull(headerAccessor.getFirstNativeHeader( headerName: "access-token")))) {
            log.info("실패");
            throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
        }
    }

    return message;

}
```

채팅 서버가 하나의 로비 역할을 하게 만든다. 즉 로그인 됐을 때 소켓을 연결하여 관리 하도록 한다.

Web socket에 connect 되기 전, jwt 토큰 인증을 진행한다.

Front에서는 Web socket 연결 헤더에 토큰 값을 실어서 보낸다

채팅 서버 - 웹 소켓

WebSocket Interceptor(링크) - preSend와 postSend를 나눠 웹 소켓 연결 구분

```java
@Override
public void postSend(Message<?> message, MessageChannel channel, boolean sent) {
    StompHeaderAccessor accessor = StompHeaderAccessor.wrap(message);
    switch (accessor.getCommand()) {
        case CONNECT:
            String session_id = accessor.getSessionId();
            String user_id = Objects.requireNonNull(accessor.getFirstNativeHeader( headerName: "user-id"));
            LoginSessionRequest loginSessionRequest = LoginSessionRequest.builder()
                    .type("login")
                    .session_id(session_id).user_id(user_id).build();
            tcpClientGateway.send(loginSessionRequest.toString());
            break;
            // todo disconnect 여러가지 테스트 (전원끄기, 랜선뽑기 등)
        case DISCONNECT:
            String sessionId = accessor.getSessionId();
            LoginSessionRequest logoutSessionRequest = LoginSessionRequest.builder()
                    .type("logout")
                    .session_id(sessionId).build();
            String id = tcpClientGateway.send(logoutSessionRequest.toString());
            String[] items = id.split( regex: ",");
            setRoomTime(items[0],items[1]);
            break;
        default:
            break;
    }
}
```

```java
public void setRoomTime(String userId, String lastRoom) {
    MessageTime result = messageTimeRepository.findByChannelId(lastRoom);
    if (result == null) {
        Map<String, LocalDateTime> users = new HashMap<>();
        users.put(userId, LocalDateTime.now());
        MessageTime messageTime = MessageTime.builder()
                .channelId(lastRoom)
                .read(users).build();
        messageTimeRepository.save(messageTime);
    } else {
        Map<String, LocalDateTime> read = result.getRead();
        read.put(userId,LocalDateTime.now());
        result.setRead(read);
        messageTimeRepository.save(result);
    }
}
```

CONNECT시 tcp를 통해 상태관리 서버로 사용자 정보를 보낸다.

DISCONNECT시 tcp를 통해 상태관리 서버로 세션 값을 보내,
상태관리 서버에 있는 사용자 정보를 삭제 시킨다.

tcp 통신의 반환 값으로 해당 채팅방에서 유저의
마지막 사용자 접속 기록을 저장한다.

채팅 서버 - 채팅 로직(웹 소켓)

* 다이렉트 채팅으로 예시를 들겠습니다.

client 제외, 각각을 클릭 하시면 해당 소스로 이동이 됩니다.

Client(채팅 입력) → Controller(messageMapping으로 수신, kafka sender로 전송) → Sender(해당 kafka topic으로 전송)

↓

Client(구독된 유저가 메세지 확인) ← Listener(해당 topic으로 들어온 메세지들을 수신, 메세지 로직 처리 , 구독된 유저에게 전송)

채팅 서버 - 카프카

kafka topic

chat-server-topic : 다이렉트 채팅을 위한 토픽

channel-server-topic : 커뮤니티 채팅을 위한 토픽

etc-direct-topic : 다이렉트 채팅 부가 기능을 위한 토픽

etc-community-topic : 커뮤니티 채팅 부가 기능을 위한 토픽

file-topic : 다이렉트,커뮤니티 채팅 파일 업로드를 위한 토픽

kafka consumer group

direct-server-group: 다이렉트 채팅을 위한 컨수머 그룹

channel-server-group : 커뮤니티 채팅을 위한 컨수머 그룹

direct-etc-server-group : 다이렉트 채팅 부가 기능을 위한 컨수머 그룹

channel-etc-server-group : 커뮤니티 채팅 부가 기능을 위한 컨수머 그룹

file-server-group : 다이렉트,커뮤니티 채팅 파일 업로드를 위한 컨수머 그룹

채팅 서버 - 카프카 리스너

1. public void directMessageListener(DirectMessage directChat)
- 다이렉트 채팅 카프카 리스너

2. public void directEctListener(DirectMessage directChat)
- 다이렉트 채팅 부가 기능 카프카 리스너

3. public void communityChatListener(ChannelMessage channelMessage)
- 커뮤니티 채팅 카프카 리스너

4. public void CommunityEctListener(ChannelMessage channelMessage)
- 커뮤니티 채팅 부가 기능 카프카 리스너

5. public void fileUpload(FileUploadResponse fileUploadResponse)
- 채팅방 파일 업로드 카프카 리스너

부가 기능

- typing (ex "병찬"님이 채팅을 입력하고 있습니다.)

- reply (채팅의 답장 기능)

- modify(채팅의 수정 기능)

- delete(채팅의 삭제 기능)

* 카프카 리스너에 websocket 관련 채팅 로직을 포함시켰습니다. * redis를 이용하여 최대한 api call을 줄이고자 했습니다. 카프카 리스너 링크입니다.(클릭)

채팅 서버 - TCP

채팅 서버와 상태관리 서버는 TCP 통신을 이용하여 데이터를 주고 받고 있습니다.

채팅 서버(Client) , 상태관리 서버(Server) 구조 입니다. (10001번 포트 이용)

채팅 서버 : outboundChannel

상태 관리 서버 : inboundChannel

String 타입을 이용하여 메세지를 전송합니다.

tcp config 링크입니다.(클릭)

tcp gateway 링크입니다.(클릭)

채팅 서버 - 채팅방 로직

ChannelMessageService (커뮤니티 채팅방 읽어오기, 파일 전송) 링크 입니다.(클릭)

DirectMessageService (다이렉트 채팅방 읽어오기, 파일 전송, 채팅방 리스트 최신화 로직) 링크 입니다.(클릭)

DirectMessageService 의 채팅방 리스트 최신화 로직

public List<MessageCountResponse> messageCount(MessageCountRequest messageCountRequest)

- 안 읽은 사람에게 뜨는 채팅 누적 개수
- 유저 별 마지막 채팅방 입장 시간을 이용하여 그 이후에 온 메세지들을 counting

- 채팅이 온 순서대로 최신화 시키는 채팅방 리스트
- 채팅방 마지막 메세지의 date 값을 이용한 정렬

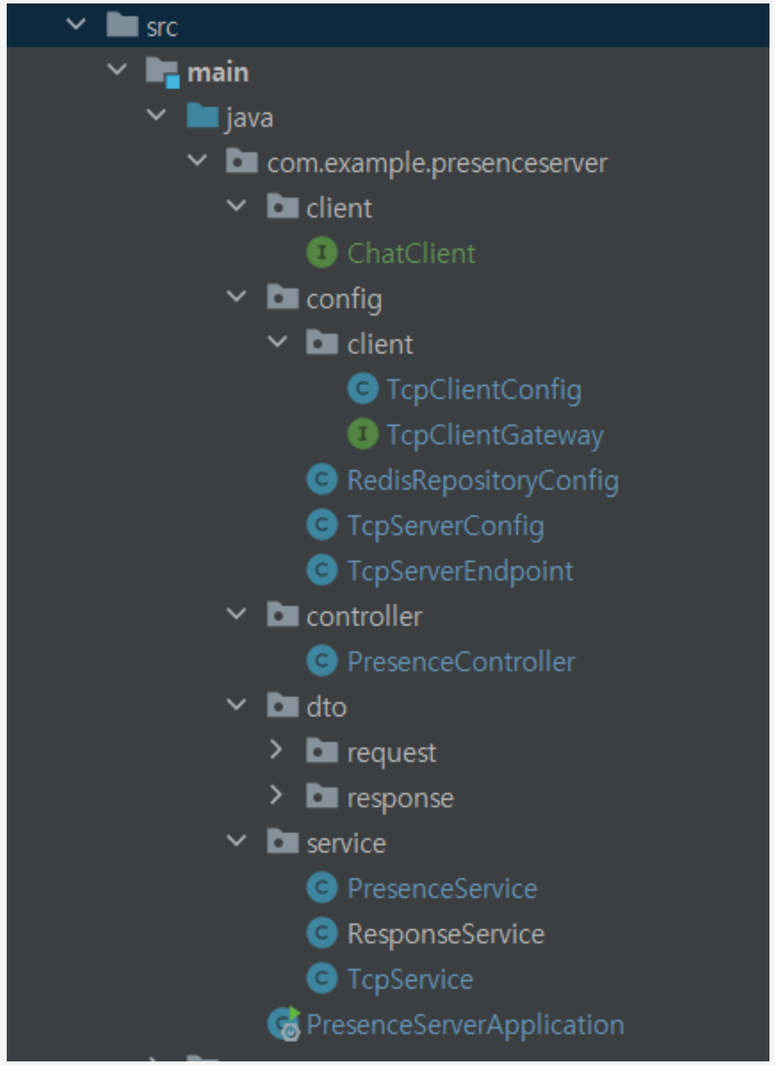
상태 관리 서버

상태 관리 서버 링크

상태 관리 서버 기술 스택

Java 11	java 8 이상 stream을 지원함, 작업 환경이 java 11 개발 환경에 익숙함
Spring Boot (2.6.2)	spring 설정을 자동으로 처리, jar 파일로 배포 가능, Spring boot 개발 환경에 익숙함
Spring Data Redis	실시간으로 변하는 사용자 정보를 저장하기 위한 휘발성 메모리
Spring Integration	spring integration tcp 통신 이용

상태 관리 서버 구조



client – feignClient를 위한 api 정의

config – tcp 통신을 위한 config

controller – 상태 관리 controller

service – 상태 관리 비즈니스 로직

상태 관리 서버 TCP 통신

상태관리 서버와 다른 서버는 TCP 통신을 이용하여 데이터를 주고 받고 있습니다.

상태관리 서버(Server)

채팅 서버(Client) 시그널링 서버(Client) 채팅 룸서버(Client) 구조 입니다. (10001번 포트 이용)

상태 관리 서버 : inboundChannel

채팅 서버 : outboundChannel

시그널링서버 : outboundChannel

채팅 룸서버 : outboundChannel

inboundChannel로 들어오는 메세지들을 @ServiceActivator를 통해 읽습니다.

tcp server config 링크입니다.(클릭)

tcp server endpoint 링크입니다.(클릭)

* @ServiceActivator란? 서비스를 메시징 시스템에 연결하기 위한 엔드포인트입니다. 입력 채널(inboundChannel) 출력 채널(outboundChannel)로 구성되어 있습니다.

상태 관리 서버 TCP 통신

TCP 통신을 통해 상태관리 서버로 들어오는 메세지들은 Type을 정의하여 관리하고 있습니다.

Type에 따라 유저 상태 정보값을 Redis에 저장하고 있습니다.

레디스 저장 기한은 2주입니다.

TYPE 정의

login - websocket connect 됐을 때 유저 상태 정보 저장

logout - websocket disconnect 됐을 때 유저

state - 유저가 방, 로비 등을 이동할 때마다 상태값 변경, 저장

direct, community - 채팅 읽은 사람, 안 읽은 사람 파악을 위한 상태값 호출

before-enter - 커뮤니티에 입장 했을 때, 음성,영상 채팅방에 있는 유저 리스트들을 반환

enter - 음성,영상 채팅방에 입장(connect)할 때마다 유저 리스트 갱신, 반환

signaling - 음성,영상 채팅방에 퇴장(disconnect할 때마다 유저 리스트 갱신, 반환

TcpService 링크입니다.(클릭)

* 유저는 websocket(로그인 할 때) 연결을 하고있고, signaling websocket(음성,영상 채팅방 입장 시) 최대 2개의 웹소켓을 연결할 수 있습니다.

감사합니다