2025.03.25

김희진

스프링 이벤트

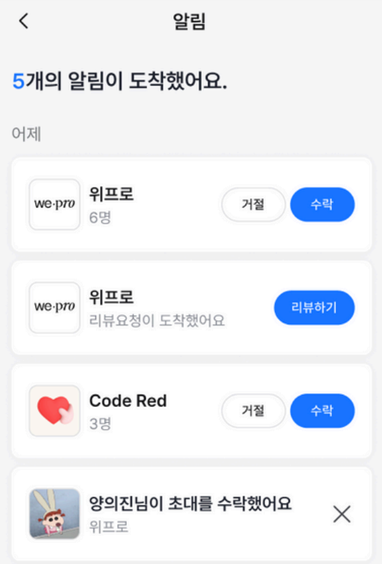
목차

01

프로젝트 상황

Alram 엔티티

Alarm 생성되는 상황: 리뷰 요청 / 프로젝트 팀원 초대 / 초대 수락



[화면]

알람					Alarm
아이디	id	Domain	bigint	NOT NULL	Default value
생성날짜	created_at	Domain	timestamp	NOT NULL	Default value
수정날짜	updated_at	Domain	timestamp	NULL	Default value
받은 사람	receiver_id	Domain	bigint	NOT NULL	Default value
보낸 사람	sender_id	Domain	bigint	NOT NULL	Default value
project, reviewForm, reviewRecord의 ID	targetId	Domain	bigint	NOT NULL	Default value
알림 멘트	message	Domain	varchar(50)	NOT NULL	Default value
알림 타입	alarmType	REVIEW_REQUEST, 등	Type	NULL	Default value
확인했는지 여부	readFlag	Domain	bool	NOT NULL	false

[Alarm 테이블]

```java
@Entity  6개 사용 위치  heejjinkim +1
@Getter
@NoArgsConstructor(access = AccessLevel.PROTECTED)
@AllArgsConstructor
@Builder
public class Alarm extends BaseEntity {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  @ManyToOne(fetch = FetchType.LAZY)
  @JoinColumn(name = "sender_id")
  private Member sender;

  @ManyToOne(fetch = FetchType.LAZY)
  @JoinColumn(name = "receiver_id")
  private Member receiver;

  private Long targetId;

  @Column(length = 50)
  private String message;

  @Enumerated(EnumType.STRING)
  private AlarmType alarmType;

  @Builder.Default
  private boolean readFlag = false;
```

[Alarm 엔티티]

01

프로젝트 상황

리뷰 요청 알람 생성

사용자가 리뷰 요청 -> 리뷰 요청을 받은 사람에게 알림 생성

```java
@Slf4j  2개 사용 위치  heejjinkim +1 *
@Service
@RequiredArgsConstructor
public class ReviewService {

  private final AlarmService alarmService;
  private final MemberRepository memberRepository;

  public void requestReview(ReviewAskRequest request, Long memberId) {  1개 사용 위치  신규 *

    Member member = memberRepository.findByIdOrThrow(memberId);
    Long reviewerId = request.getReviewerId();
    Long reviewFormId = request.getReviewFormId();
    alarmService.createAlarm(member, reviewerId, AlarmType.REVIEW_REQUEST, reviewFormId);
  }
```

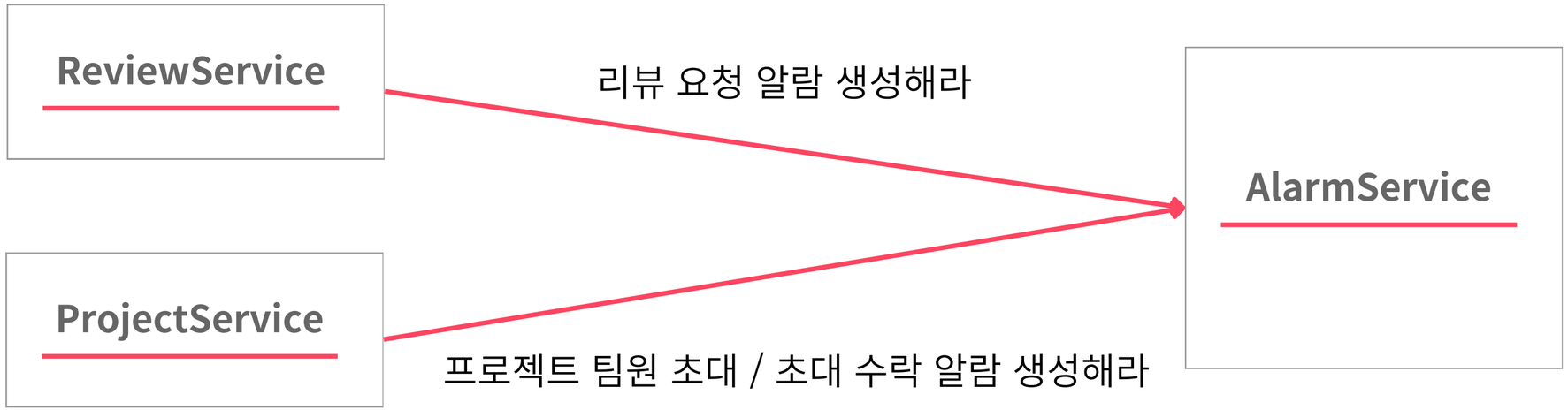
[ReviewService]

```java
@Service  2개 사용 위치  heejjinkim
@RequiredArgsConstructor
public class AlarmService {

  private final MemberRepository memberRepository;
  private final AlarmRepository alarmRepository;

  @Transactional  1개 사용 위치  heejjinkim
  public void createAlarm(Member sender, Long receiverId, AlarmType alarmType, Long targetId) {
    Member receiver = memberRepository.findById(receiverId).orElseThrow(()
        -> new RestApiException(UserErrorCode.USER_NOT_FOUND));

    alarmRepository.save(Alarm.of(sender, receiver, alarmType, targetId));
  }

}
```

[AlarmService]

01

문제점



- Review/Project와 Alaram 도메인 간의 **강한 결합도**
  - Alarm 생성 로직 변경 시, ReviewService와 ProjectService에 영향
- **단일 책임 원칙(SRP) 위배**
  - 단일 책임 원칙: 어떤 클래스를 변경해야 하는 이유는 오직 하나
  - 현재 Review: Review 관련 서비스 로직 + Alarm을 생성하는 책임
- **좋지 않은 확장성**
  - 서비스 내부 알람 뿐만이 아니라, 이메일 알람도 보내고 싶다면 emailService.createEmailAlarm()을 ReviewService와 ProjectService에서 호출해야 함

02

스프링 이벤트란?

구성 요소

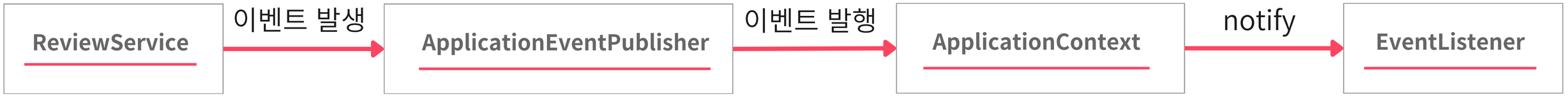
- **이벤트 :** 특정 조건이 충족되었을 때 발생하는 사건
- **발행자(publisher)**: 이벤트를 발행시킴
- **구독자(subscriber)**: 이벤트를 감지하고 특정 동작 수행

목적

의존성 분리를 위해

흐름

1. ApplicationEventPublisher가 publishEvent()로 이벤트 발행
2. ApplicationContext가 발행된 이벤트를 @EventListener가 붙어있는 메소드에 전달
3. @EventListener 메소드 로직 실행



02

스프링 이벤트 적용

- 이벤트 종류: 리뷰 요청, 팀원 초대, 초대 수락

02

문제 해결: 강한 결합도

- Review/Project와 Alaram 도메인 간의 **강한 결합도**
  - 중간에 EventListener가 생기면서,
  - ReviewService/ProjectService는 AlarmService에 의존하지 않아도 됨
  - Alarm 생성 로직 변경 시, ReviewService와 TeamService에 영향 X

02

문제 해결: 단일 책임 원칙

- **단일 책임의 원칙(SRP)**
  - 기존: “알람 생성해라"
  - 수정: “리뷰 요청/팀원 초대/초대 수락 이벤트가 발생했다!”

02

문제 해결: 좋지 않은 확장성

- 지금은 이벤트 처리 결과가 ‘서비스 내부 알림' 밖에 없지만
- 추후에 ‘이메일 알림’, ‘문자 알림’ 등을 추가하고 싶다면
- 쉽게 확장 가능

03

코드 수정

Event 클래스 생성

```
@Getter  5개 사용 위치  신규 *
@RequiredArgsConstructor
public class ReviewRequestEvent {
    private final Long senderId;
    private final Long receiverId;
    private final Long reviewFormId;
}
```

```
@Getter  1개 사용 위치  신규 *
@RequiredArgsConstructor
public class TeamInviteEvent {
    private final Long inviterId;
    private final Long inviteeId;
    private final Long projectId;
}
```

```
@Getter  1개 사용 위치  신규 *
@RequiredArgsConstructor
public class TeamInviteAcceptanceEvent {
    private final Long inviterId;
    private final Long inviteeId;
    private final Long projectId;
}
```

03

코드 수정

ReviewService에 eventPublisher 주입

```
@Slf4j  2개 사용 위치  heejjinkim +1 *
@Service
@RequiredArgsConstructor
public class ReviewService {

  private final ApplicationEventPublisher eventPublisher;
  private final MemberRepository memberRepository;

  public void requestReview(ReviewAskRequest request, Long memberId) {  1개 사용 위치  heejjinkim
    Member member = memberRepository.findByIdOrThrow(memberId);
    Long reviewerId = request.getReviewerId();
    Long reviewFormId = request.getReviewFormId();

    ReviewRequestEvent event = new ReviewRequestEvent(memberId, reviewerId, reviewFormId);
    eventPublisher.publishEvent(event);
  }
```

02

코드 수정

@EventListener 사용

```java
@Component
@RequiredArgsConstructor
public class AlarmEventListener {
    private final AlarmService alarmService;
//    private final EmailService emailService; // 확장 가능성

    @EventListener 신규 *
    public void createReviewRequestAlarm(ReviewRequestEvent event) {
        alarmService.createAlarm(event.getSenderId(), event.getReceiverId(), AlarmType.REVIEW_REQUEST, event.getReviewFormId());
    }

    @EventListener 신규 *
    public void createReviewRequestEmail(ReviewRequestEvent event) {
        // emailService.sendEmail(); // 확장 가능성
    }

    @EventListener 신규 *
    public void handleTeamInviteEvent(TeamInviteEvent event) {
        alarmService.createAlarm(event.getInviterId(), event.getInviteeId(), AlarmType.TEAM_INVITE, event.getProjectId());
    }

    @EventListener 신규 *
    public void handleTeamInviteAcceptanceEvent(TeamInviteAcceptanceEvent event) {
        alarmService.createAlarm(event.getInviterId(), event.getInviteeId(), AlarmType.TEAM_INVITE_ACCEPTANCE, event.getProjectId());
    }
}
```

03

@EventListener 문제

1. 이벤트는 기본적으로 동기로 동작 => 비동기

1. 서비스 내부 알림 + 이메일 알림 보낸다고 가정
2. 동기로 동작하면 서비스 내부 알림 생성 후, 이메일 알림 보냄
3. 이메일 알림이 오래 걸리면 사용자는 이 시간을 모두 기다려야 함

2. EventListener의 로직에서 예외가 발생하면 Event를 발행시킨 로직도 같이 롤백 => @TransactionalEventListener

[ex. 초대 수락 알림]

1. 프로젝트 멤버 초대를 받은 유저가 초대 수락
2. ProjectService에서 팀 멤버를 새로 생성하여 저장
3. 초대를 수락했다는 이벤트를 발생시킴
4. AlarmEventListener에서 알람 생성하여 저장하려 했는데, Exception 발생
5. 그럼 **팀 멤버를 새로 생성하는 과정도 같이 롤백**됨

>>> **초대를 수락한 사용자 입장에서, 수락했음에도 수락되지 않은 상황 발생**

03

@EventListener vs. @TransactionalEventListener

@EventListener: 이벤트가 발행되는 시점에 바로 EventListener의 로직이 실행

@TransactionalEventListener: 트랜잭션의 어떤 타이밍에 이벤트를 발생시킬 지 정할 수 있다

- AFTER_COMMIT (기본값) - 트랜잭션이 성공적으로 마무리(commit)됐을 때 이벤트 실행
- AFTER_ROLLBACK – 트랜잭션이 rollback 됐을 때 이벤트 실행
- AFTER_COMPLETION – 트랜잭션이 마무리 됐을 때(commit or rollback) 이벤트 실행
- BEFORE_COMMIT - 트랜잭션의 커밋 전에 이벤트 실행

[ex. 초대 수락 알림] - @TransactionalEventListener(phase = TransactionPhase.AFTER_COMMIT)

1. ProjectService에서 팀 멤버를 새로 생성하여 저장
2. 초대를 수락했다는 이벤트를 발생시킴
3. 트랜잭션 커밋
4. AlarmEventListener에서 알람 생성하여 저장하려 했는데, Exception 발생
5. 팀 멤버는 이미 커밋되었기 때문에 잘 저장됨

03

코드 수정 @Async

```java
@Component
@RequiredArgsConstructor
public class AlarmEventListener {
    private final AlarmService alarmService;
//    private final EmailService emailService; // 확장 가능성

    // 리뷰 요청 보내기 이벤트
    @Async 신규 *
    @EventListener
    public void createReviewRequestAlarm(ReviewRequestEvent event) {
        alarmService.createAlarm(event.getSenderId(), event.getReceiverId(), AlarmType.REVIEW_REQUEST, event.getReviewFormId());
    }

    @Async 신규 *
    @EventListener
    public void createReviewRequestEmail(ReviewRequestEvent event) {
        // emailService.sendEmail(); // 확장 가능성
    }
```

03

코드 수정 @Async + @TransactionalEventListener

```java
// 팀 초대 보내기 이벤트
@Async 신규 *
@TransactionalEventListener
public void createTeamInviteAcceptanceEventAlarm(TeamInviteAcceptanceEvent event) {
    alarmService.createAlarm(event.getInviterId(), event.getInviteeId(), AlarmType.TEAM_INVITE_ACCEPTANCE, event.getProjectId());
}

@Async 신규 *
@TransactionalEventListener
public void createTeamInviteAcceptanceEventEmail(TeamInviteAcceptanceEvent event) {
    // emailService.sendEmail(); // 확장 가능성
}
```

03

@TransactionalEventListener 만 쓰면 안됨!

[문제]

팀 멤버 생성 후 트랜잭션에서 이미 Commit이 되었기 때문에 해당 트랜잭션에서는 조회밖에 수행할 수 없다!

= EventListener에서 데이터를 insert, delete, update하면 반영이 안된다

[해결]

@Transactional의 propagation을 REQUIRES_NEW로 설정하기

```
// 팀 초대 보내기 이벤트
@Async 신규 *
@Transactional(propagation = Propagation.REQUIRES_NEW)
@TransactionalEventListener
public void createTeamInviteAcceptanceEventAlarm(TeamInviteAcceptanceEvent event) {
    alarmService.createAlarm(event.getInviterId(), event.getInviteeId(), AlarmType.TEAM_INVITE_ACCEPTANCE, event.getProjectId());
}

@Async 신규 *
@Transactional(propagation = Propagation.REQUIRES_NEW)
@TransactionalEventListener
public void createTeamInviteAcceptanceEventEmail(TeamInviteAcceptanceEvent event) {
    // emailService.sendEmail(); // 확장 가능성
}
```

04

AOP vs. 스프링 이벤트

AOP를 써서 “알림 생성" 이라는 공통 관심사를 분리하면 되지 않나?

- AOP는 **공통**된 **부가적인 기능**(ex. 로깅)을 하나의 Aspect로 분리하는 것이다
- “알람 생성"이 리뷰 요청 / 프로젝트 팀원 초대 / 초대 수락에서 **공통된 부분은 맞다**
- 하지만 **부가적인 기능은 아니다**
  - 부가 기능 = 핵심 기능을 보조하는 기능(ex. 로그, 수행 시간 저장)
- “알림 생성"은 DB 처리 로직으로, Aspect으로는 부적절

>>> “두 도메인 간의 의존성을 분리”하는 목적에는 스프링 이벤트가 적절

참고자료

- https://supawer0728.github.io/2018/03/24/spring-event/
- https://mangkyu.tistory.com/292
- https://wildeveloperetrain.tistory.com/217
- https://velog.io/@ming_gry/%ED%95%AD%ED%95%B499-9%EC%A3%BC%EC%B0%A8-WIL
- https://ksh-coding.tistory.com/111#1.%20%EC%8A%A4%ED%94%84%EB%A7%81%20%EC%9D%B4%EB%B2%A4%ED%8A%B8%EB%A5%BC%20%EC%82%AC%EC%9A%A9%ED%95%B4%EC%84%9C%20%EC%9D%98%EC%A1%B4%EC%84%B1%20%EB%B6%84%EB%A6%AC%ED%95%98%EA%B8%B0-1

Q & A