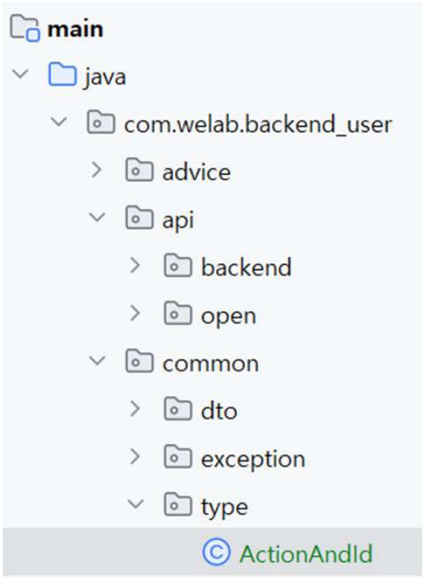
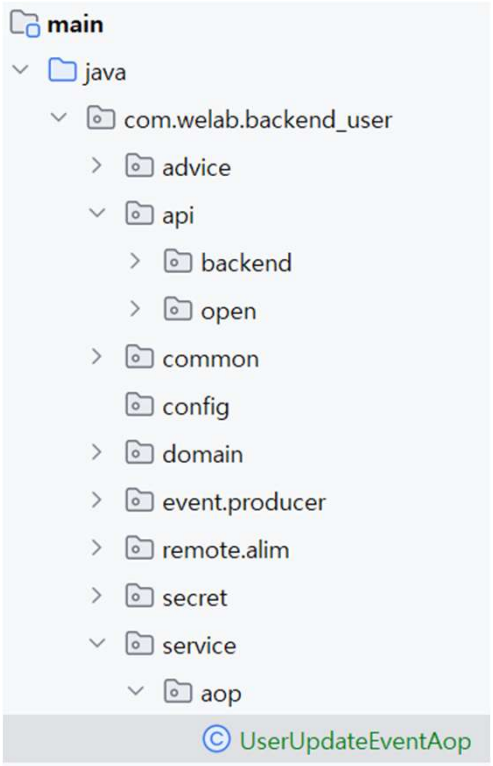
메시지 발행 AOP 적용하기

Alim 프로젝트 코드 작성 – ActionAndId




```java
@Getter
@Setter
public class ActionAndId {
    private String action;
    private Long id;

    public static ActionAndId of(String action, Long id) {
        ActionAndId actionAndId = new ActionAndId();

        actionAndId.action = action;
        actionAndId.id = id;

        return actionAndId;
    }
}
```

Alim 프로젝트 코드 작성 – UserUpdateEventAop



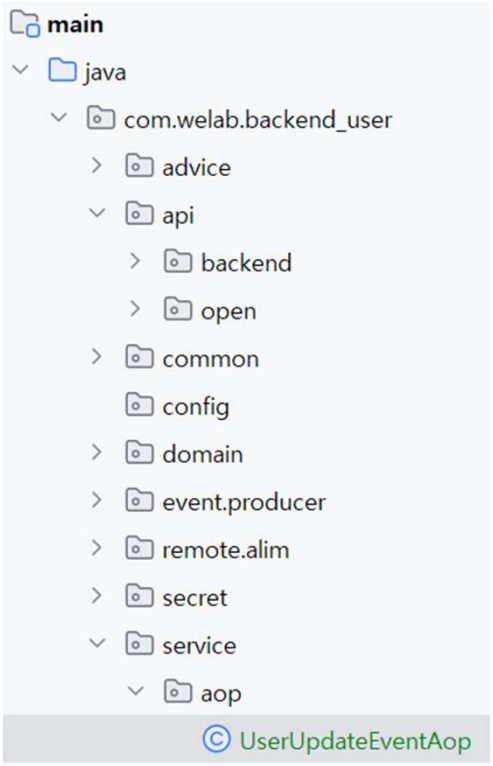

```java
@Slf4j
@Aspect
@Component
@RequiredArgsConstructor
public class UserUpdateEventAop {
    private final SiteUserRepository siteUserRepository;
    private final KafkaMessageProducer kafkaMessageProducer;

    @AfterReturning(
            value = "execution(* com.welab.backend_user.service.SiteUserService.*AndNotify(..))",
            returning = "actionAndId"
    )
    public void publishUserUpdateEvent(JoinPoint joinPoint, ActionAndId actionAndId) {
        publishUserUpdateEvent(actionAndId);
    }
```

메소드 실행 후 값이 반환되면, publishUserUpdateEvent() 메소드 실행

...AndNotify로 끝나는 메소드들에 대해서 실행

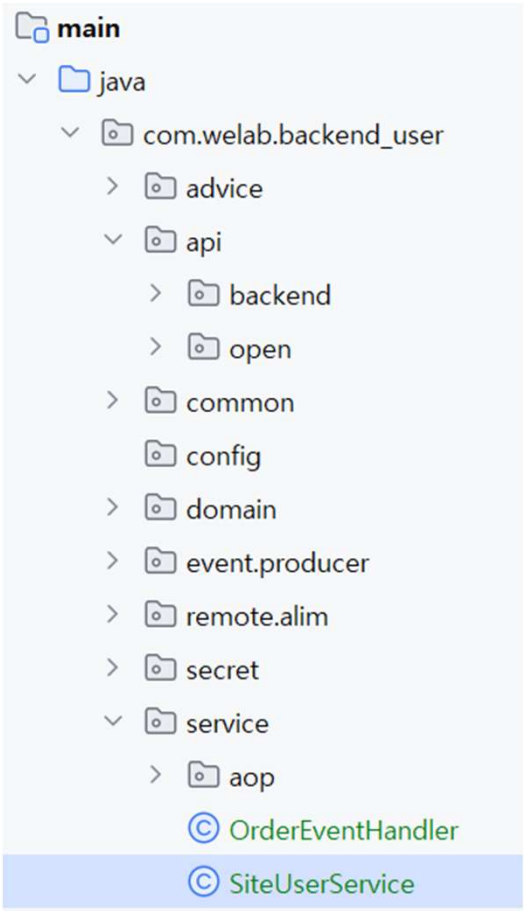
`actionAndId` 값이 반환되어야 함

Alim 프로젝트 코드 작성 – UserUpdateEventAop (계속)




```
    public void publishUserUpdateEvent(ActionAndId actionAndId) {
        try {
            SiteUser siteUser = siteUserRepository.findById(actionAndId.getId())
                    .orElse(null);

            if (siteUser == null) {
                return;
            }

            SiteUserInfoEvent event = SiteUserInfoEvent.fromEntity(actionAndId.getAction(), siteUser);

            kafkaMessageProducer.send(SiteUserInfoEvent.Topic, event);
        } catch (Exception e) {
            log.warn("사용자 정보 업데이트 이벤트를 전송하지 못하였습니다. id={}",
                    actionAndId.getId());
        }
    }
}
```

Alim 프로젝트 코드 작성 – SiteUserService 수정




```java
@Slf4j
@Service
@RequiredArgsConstructor
public class SiteUserService {
    private final SiteUserRepository siteUserRepository;
    private final TokenGenerator tokenGenerator;

    @Transactional
    public ActionAndId registerUserAndNotify(SiteUserRegisterDto registerDto) {
        SiteUser siteUser = registerDto.toEntity();

        siteUserRepository.save(siteUser);

        return ActionAndId.of("Create", siteUser.getId());
    }

    …
```

action = "Create"
id = 사용자 아이디

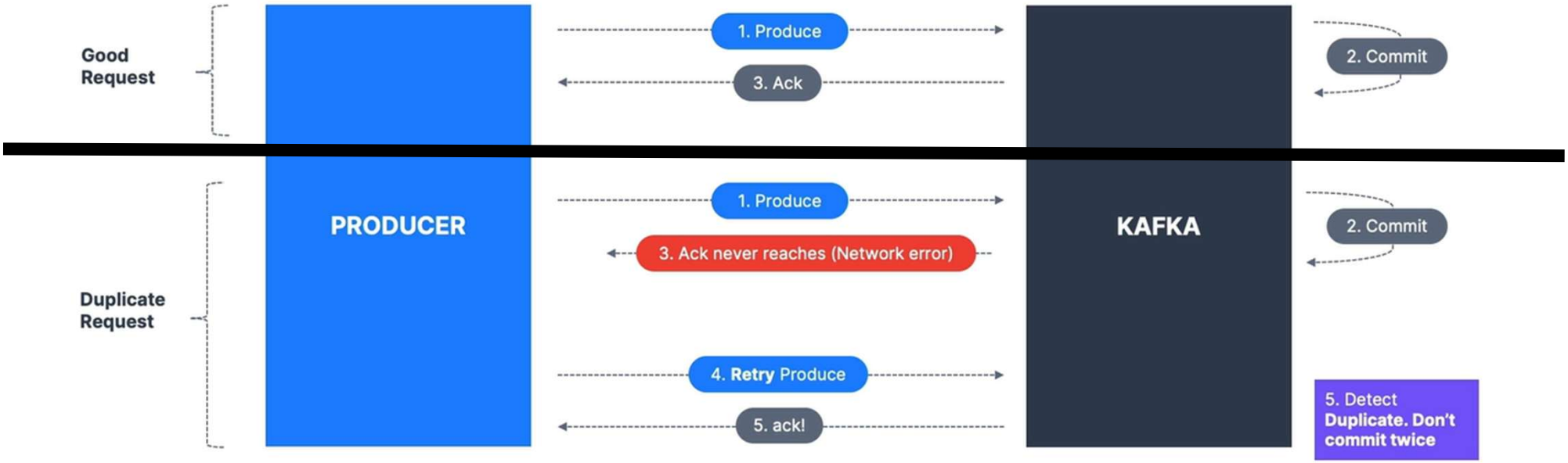
Kafka 멱등성

멱등성(Idempotence)이란?

- 특정 연산을 여러 번 수행해도 최초 한 번 수행한 이후의 결과가 변하지 않는 성질을 의미
- 주로 데이터베이스, API 설계, 메시지 시스템 등에서 중요
- 중복 처리를 방지하여 시스템의 안정성과 데이터 일관성을 유지하는 데 기여

Kafka 멱등성 Producer

- 중복 메시지가 전송되더라도 카프카에서는 단 한 번만 커밋되도록 함으로써 데이터의 중복을 방지
- Kafka 3.0 버전부터는 이 옵션이 기본적으로 활성화됨



Kafka Consumer는?

- Kafka의 Consumer는 기본적으로 "적어도 한 번(at-least-once)" 전송을 보장 -> 1회 이상 전송 가능
- Consumer의 중복 처리 설계 필요

-> 멱등성 저장소를 이용하여 중복 문제 해결

Kafka Consumer 중복 처리 방지 중복 처리 검증 Entitiy

```java
@Entity
@Table(name = "processed_event",
        uniqueConstraints = {
                @UniqueConstraint(columnNames = {"topic", "event_id"})
        })
public class ProcessedEvent {
    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "topic", nullable = false)
    private String topic;

    @Column(name = "event_id", nullable = false)
    private String eventId;
}
```

Kafka Consumer 중복 처리 방지 (계속)

```java
@Slf4j
@Service
@RequiredArgsConstructor
public class OrderEventHandler {
    private final ProcessedEventRepository processedEventRepository;

    @Transactional
    public void handleOrderEvent(OrderEvent orderEvent) {
        // 1. 중복 이벤트 검증
        if (processedEventRepository.existsByTopicAndEventId(OrderEvent.Topic, orderEvent.getEventId())) {
            log.warn("중복 이벤트 무시: {}", orderEvent.getEventId());
            return; // 이미 처리된 이벤트
        }

        // 2. 비즈니스 로직 실행 (멱등성 보장된 작업)
        ...

        // 3. 처리 완료 기록 저장
        ProcessedEvent processedEvent = new ProcessedEvent();
        processedEvent.setTopic(OrderEvent.Topic);
        processedEvent.setEventId(orderEvent.getEventId());
        processedEventRepository.save(processedEvent);
    }
}
```

가장 간단한 방법.
다른 방법들도 존재.