[Spring]ApplicationEvent (Publisher, Listener)

생성일	@2025년 11월 29일
카테고리	Study
태그	Database JPA Java

[Spring]ApplicationEvent (Publisher, Listener)이란?

스프링에서 이벤트를 발행하고 처리하는 패턴입니다.

도메인/애플리케이션 이벤트를 다른 컴포넌트들에게 느슨하게 전달하는 용도로 사용합니다.

즉 관심 있는 객체(Listener)들한테 broadcast해주는 이벤트 버스 역할을 합니다

본 기술로 어떤 문제를 해소하나요?
“직접 호출의 결합도를 끊어냅니다.”

[가정 상황]

- A 서비스에서 ‘상태 변경’이 일어남
- 그 후에 해야 할 작업이 많다는 가정이있습니다.
    - **알림 발송**
    - **로그/감사 기록**

- **통계 적재**
- **외부 API 호출**
- **기타 후속 처리**

이걸 전부 A 서비스 안에서 다른 서비스들 직접 주입해서 메서드 호출로 처리하면 어떤 문제가 있을까요?

- A 서비스가 **후처리 관심사를 전부 알고 있어야 함**
- 새로운 후처리 로직이 생기면:
  - 코드 수정 범위가 커짐 (A가 계속 비대해짐)
  - 테스트/리팩터링 난이도 증가
- "상태 변경"이라는 핵심 도메인 로직과

  "그 이후에 따라붙는 후속 작업"의 **관심사가 섞여버림**

> 💡 ***즉 상태가 적으면 문제가 생기지 않지만 로직이 복잡해지고 변경 케이스가 많아질수록 관리가 힘들어집니다.***

`ApplicationEventPublisher` 가 해주는 것

- A는 그냥

  "**지금 이 도메인 이벤트가 발생했다**"라는 사실만 던짐
- "그다음에 누가 무엇을 할지"는

  이벤트 리스너들이 알아서 처리
- 결과적으로
  - **발행자(Producer)** 와
  - **구독자(Listener)** 사이를 **느슨하게 결합하여 해소가 가능합니다.**

Spring Application Event의 구성요소 (Publisher, Event, Listener)

1. **이벤트 발행자 (Publisher)**
    - 도메인/애플리케이션 로직을 실행하다가

      "의미 있는 상태 변화가 발생했다"라고 판단하는 쪽
    - 이 쪽은 *"누가 이걸 듣고 있는지"* 몰라도 됨
2. **이벤트 객체 (Event)**
    - "무슨 일이 일어났는지"를 표현하는 데이터 Class
3. **이벤트 리스너 (Listener)**
    - 이벤트가 발생했을 때 필요한 후속 작업을 수행

그리고 이들 사이를 연결해주는 **중앙 허브**가 바로:

- `ApplicationEventPublisher` (역할)
- `ApplicationContext` (구현체, 실질적인 이벤트 버스)

즉:

> 발행자 → ApplicationEventPublisher → (타입 기반 매칭) → 리스너들

이라는 **간접 호출 구조**가 만들어집니다.

Example Code

```
// OrderCreatedEvent.java
public class OrderCreatedEvent {
    private final Long orderId;

    public OrderCreatedEvent(Long orderId) {
        this.orderId = orderId;
    }
```

```
    public Long getOrderId() {
        return orderId;
    }
}



@Service
@RequiredArgsConstructor
public class OrderService {

    private final ApplicationEventPublisher eventPublisher;

    public void createOrder(Long orderId) {
        // 1. 주문 생성 비즈니스 로직 (생략)
        // ...

        // 2. 주문 생성 이벤트 발행
        eventPublisher.publishEvent(new OrderCreatedEvent(orderId));
    }
}



@Slf4j
@Component
public class OrderEventListener {

    @EventListener
    public void handleOrderCreated(OrderCreatedEvent event) {
        log.info("주문 생성됨. orderId={}", event.getOrderId());
        // 여기서 알림 발송, 로그 적재, 외부 호출 등 후처리
    }
}

@RestController
@RequiredArgsConstructor
public class OrderController {
```

```
    private final OrderService orderService;

    @PostMapping("/orders/{orderId}")
    public String create(@PathVariable Long orderId) {
        orderService.createOrder(orderId);
        return "ok";
    }
}
```