1. Event

개념	설명
이벤트(Event)	어떤 상황이 발생했다는 신호 또는 메시지
이벤트 발행(Publish)	이벤트를 발생시키는 행위
이벤트 리스너(Listener)	이벤트가 발생했을 때 처리하는 코드 (구독자 역할)
이벤트 핸들링	이벤트 발생 시 특정 로직을 실행하는 것

2. Spring 이벤트 구성요소

구성요소	설명	주요 메서드
org.springframework.context.ApplicationEvent	이벤트 객체 (상속 가능)	getSource(), getTimestamp()
ApplicationEventPublisher	이벤트 발행자	publishEvent(Object event)
@EventListener	이벤트 리스너 선언	없음 (메서드 어노테이션, 조건 처리 가능)

Spring 의 Built-in Events (내장 이벤트)

ApplicationContext 의 생명주기나 웹 요청 처리 과정에서 자동으로 이벤트를 발생

특별한 설정 없이 @EventListener 를 통해 사용

이벤트명	설명	발생 시점	사용 예시
ContextRefreshedEvent	ApplicationContext 가 초기화되거나 refresh()가 호출될 때 발생	앱 실행 시, context.refresh() 호출 시	초기 설정 로직 실행, 캐시 로딩 등
ContextStartedEvent	ApplicationContext 에서 start()가 호출될 때 발생	수동으로 start() 호출 시	수동 시작 대상 Bean 실행
ContextStoppedEvent	ApplicationContext 에서 stop()이 호출될 때 발생	수동으로 stop() 호출 시	Bean 중지, 스케줄러 중단 등

ContextClosedEvent	ApplicationContext 가 종료될 때 (close() 호출, JVM 종료 등)	앱 종료 시 또는 수동 close() 시	자원 정리, 커넥션 종료 등
RequestHandledEvent	Spring MVC 가 HTTP 요청을 처리한 직후 발생	웹 요청 처리 후	요청 완료 로그 기록, 모니터링
ServletRequestHandledEvent	RequestHandledEvent 의 서브클래스로, 서블릿 요청에 특화된 정보 포함	Servlet 요청 처리 후	추가 컨텍스트 정보를 활용한 로깅

Built-in Events 를 사용하는 이유

사용목적	설명
라이프사이클 관리	애플리케이션의 시작, 중지, 종료 시점에 맞춰 자동 처리 가능
유지보수성 향상	비즈니스 로직과 시스템 관리 로직을 분리
테스트 용이성	테스트 컨텍스트에 리스너를 부착하여 흐름 검증 가능

https://docs.spring.io/spring-framework/reference/core/beans/context-introduction.html

3. 이벤트의 종류

① 동기 이벤트 (defalut)

- 이벤트를 발행하면, 리스너가 처리할 때까지 기다림
- 메서드가 순차적으로 동작

② 비동기 이벤트

- 리스너에 @Async 를 붙이면, 이벤트가 **다른 스레드에서 처리됨**
- @EnableAsync 필요

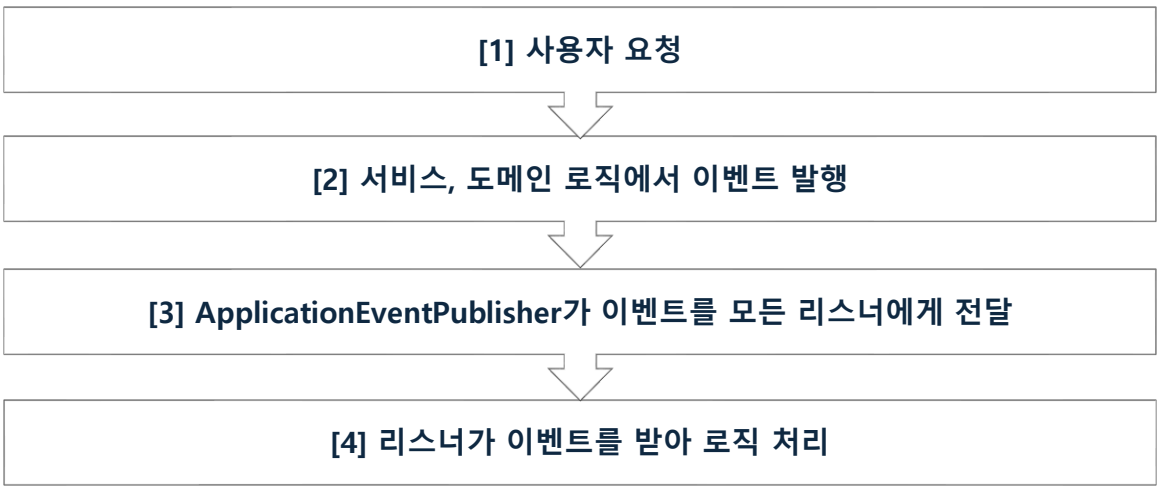
③ 커스텀 이벤트

- ApplicationEvent 를 상속하거나 단순 POJO 사용 가능

④ 외부 이벤트 (Kafka 등)

- 시스템 간 통신용 이벤트

4. Spring Boot 이벤트 처리 흐름



5. 동기 이벤트

이벤트를 발행한 쪽이 이벤트 리스너의 처리가 끝날 때까지 기다리는 방식

- 이벤트 → 리스너 실행까지 **같은 스레드에서 연속적으로 처리**됨
- 예외 발생 시 상위 호출 스택으로 전달되며, **트랜잭션과 같은 흐름 제어가 가능**
- **성능보다 신뢰성과 제어 흐름 유지가 중요할 때 사용**

동기 이벤트의 특징

항목	설명
실행 흐름	이벤트 발행 → 리스너 실행 → 다음 코드 진행
처리 방식	**동기적 (blocking)**
스레드	호출자와 **같은 스레드**에서 처리
사용 사례	데이터 저장 후 로깅, 검증, 트랜잭션 내 후속처리

Synchronous Event Structure

[1] 이벤트 클래스

```
public class @Data UserRegisteredEvent {
    private final String username;
}
```

[2] 서비스에서 이벤트 발생

```
@Service
public class UserService {
    @Autowired
    private ApplicationEventPublisher publisher;

    public void registerUser(String username) {
            publisher.publishEvent(new UserRegisteredEvent(username));
        System.out.println("사용자 등록 완료"); -> 실행 2
    }
}
```

[3] 이벤트 리스너 (동기 처리)

```
@Component
public class UserEventListener {
    @EventListener
    public void handleUserRegistered(UserRegisteredEvent event) {
        System.out.println("환영 메시지 전송: " + event.getUsername()); -> 실행 1
    }
}
```

6. 비동기 이벤트

비동기 이벤트는 **이벤트를 발행한 이후에, 리스너가 별도의 스레드에서 이벤트를 처리**하는 방식

- 발행자와 리스너가 **다른 스레드**에서 동작
- 리스너의 실행 여부와 관계없이 발행자는 다음 코드로 진행함
- 병렬 처리, 대량 작업 분산 처리 등에 유용

비동기 이벤트의 특징

항목	설명
실행 흐름	이벤트 발행 → **바로 다음 코드 진행** → 리스너는 나중에 실행
처리 방식	비동기 (non-blocking)
스레드	리스너는 별도 스레드에서 실행
사용 사례	이메일 발송, 슬랙 알림, 대용량 로그 저장, 알림 발송 등
설정 필수	@Async + @EnableAsync

Asynchronous Event Structure

[3] 이벤트 리스너 (비동기 처리) [1][2]는 동기 이벤트 동일

```
@Component
public class UserEventListener {
    @Async
    @EventListener
    public void handleUserRegistered(UserRegisteredEvent event) {
System.out.println("환영 메시지 전송: " + event.getUsername()); // 실행 2     } }
```

[4] 설정 클래스 (@EnableAsync 추가)

```
@Configuration
@EnableAsync
public class AsyncConfig {
    // 비동기 처리 활성화 설정
}
```

7. 커스텀 이벤트 (POJO 기반)

Spring 4.2 이후부터는 ApplicationEvent 를 상속하지 않고도 **일반 클래스(POJO)** 만으로 이벤트 객체를 만들 수 있다.

커스텀 이벤트의 특징

항 목	설 명
이벤트 정의	POJO 클래스 (일반 클래스)
상속 필요 여부	ApplicationEvent 상속 ❌
처리 방식	기존과 동일하게 publishEvent() + @EventListener 사용
장점	도메인에 가까운 객체로 이벤트 구성 가능, 간결함

Custom Event Structure

[1] 이벤트 클래스

```
@Data
@AllArgsConstructor
public class OrderCreatedEvent {
    private String orderNumber;
}
```

[2] 서비스 클래스 (이벤트 발행)

```
@Service
public class OrderService {
    @Autowired
    private ApplicationEventPublisher publisher;

    public void createOrder(String orderNumber) {
        // 주문 저장 등 로직 생략 ...
        publisher.publishEvent(new OrderCreatedEvent(orderNumber));
        System.out.println("주문 처리 완료"); // 실행 1
    }
}
```

[3] 리스너 클래스 (이벤트 처리)

```
@Component
public class OrderEventListener {

    @EventListener
    public void handle(OrderCreatedEvent event) {
     System.out.println("주문 확인 알림 발송: " + event.getOrderNumber()); // 실행 2
    }
}
```

8. 조건부 이벤트 처리 (@EventListener(condition = "..."))

- Spring 의 @EventListener 는 condition 속성을 통해 **특정 조건일 때만 이벤트를 처리**하도록 제어
- SpEL(Spring Expression Language)을 사용하여 이벤트 객체의 값을 평가하고, **필터링된 리스닝**을 구현.
- **하나의 이벤트 객체를 여러 조건별로 분리하여 처리할 때 사용**

조건부 이벤트의 특징

항 목	설 명
조건식	SpEL 사용 (#event.필드명, T(java.lang.String).valueOf(...) 등)
실행 흐름	이벤트 발행 → 조건 평가 → true 일 때만 리스너 실행
장점	리스너 클래스 분리 없이 내부에서 조건 기반 처리 가능
사용 사례	특정 이메일 도메인일 때만 발송, VIP 사용자만 알림 처리 등

Conditional Event Structure

[1] 이벤트 클래스

```
@Data
@AllArgsConstructor
public class UserRegisteredEvent {
    private String username,  email;
}
```

[2] 이벤트 발행

```
@Service
public class UserService {
    @Autowired
    private ApplicationEventPublisher publisher;

    public void registerUser(String username, String email) {
        publisher.publishEvent(new UserRegisteredEvent(username, email));
        System.out.println("회원가입 처리 완료");
    } }
```

[3] 조건부 이벤트 리스너

```
@Component
public class ConditionalUserEventListener {

    @EventListener(condition = "#event.email.endsWith('.com')")
    public void handleComUsers(UserRegisteredEvent event) {
        System.out.println(".com 도메인 사용자 환영: " + event.getEmail());
    }

    @EventListener(condition = "#event.email.endsWith('.co.kr')")
    public void handleKrUsers(UserRegisteredEvent event) {
        System.out.println(".co.kr 도메인 사용자 환영: " + event.getEmail());
    }
}
```

[4] 실행 결과

registerUser("kim", "kim@gmail.com")
// 출력 → .com 도메인 사용자 환영: kim@gmail.com

registerUser("Dominica", " Dominica @company.co.kr")
// 출력 → .co.kr 도메인 사용자 환영: Dominica @company.co.kr

SpEL 조건 작성 팁

예시 조건	의미
#event.username == 'admin'	admin 유저일 경우에만 처리
#event.email.contains('naver')	이메일에 'naver' 포함될 경우
#event.email.matches('.*@gmail₩₩.com')	정규표현식으로 조건 지정

9. 이벤트 리스너 순서 제어 (@Order)

- 동일한 이벤트를 여러 리스너가 처리하는 경우, Spring 에서는 기본적으로 **리스너 실행 순서가 보장되지 않는다**.
- @Order 어노테이션을 사용하면 **리스너의 실행 우선순서를 지정**할 수 있다.

리스너 순서 제어의 특징

항목	설명
어노테이션	@Order(int value)
숫자가 작을수록	먼저 실행됨 (@Order(1) → @Order(2))
적용 위치	@EventListener 메서드 또는 클래스 위
주의점	기본값은 우선순위 없음 (정렬되지 않음)

Ordered Event Structure

- @Order 는 **클래스 전체에도 부여 가능**하나, 보통은 **각 메서드에 부여**하는 것이 명확함
- 비동기(@Async)와 함께 사용할 경우 순서 보장 X (스레드 경쟁 발생 가능)

[1] 이벤트 클래스

```
@AllArgsConstructor
public class   @Data   NotificationEvent {
    private String message;
}
```

[2] 서비스에서 이벤트 발행

```
@Service
public class NotificationService {
```

```
    @Autowired
    private ApplicationEventPublisher publisher;

    public void sendNotification(String message) {
        publisher.publishEvent(new NotificationEvent(message));
        System.out.println("이벤트 발행 완료");
    }
}
```

[3] 순서가 지정된 리스너들

```
@Component
public class NotificationEventListener {

    @Order(1)
    @EventListener
    public void logToConsole(NotificationEvent event) {
        System.out.println("[1] 콘솔 로그: " + event.getMessage());
    }

    @Order(2)
    @EventListener
    public void logToDatabase(NotificationEvent event) {
        System.out.println("[2] DB 저장: " + event.getMessage());
    }

    @Order(3)
    @EventListener
    public void sendToSlack(NotificationEvent event) {
        System.out.println("[3] 슬랙 전송: " + event.getMessage());
    }
}
```

10. @TransactionalEventListener – 트랜잭션 이후 이벤트 처리

@TransactionalEventListener 는 **트랜잭션의 커밋(commit)이 완료된 후에만 이벤트를 처리**하도록 보장해주는 어노테이션

트랜잭션 이벤트 리스너의 특징

항목	설명
실행 시점	트랜잭션 커밋 이후 (기본 동작)
트랜잭션 의존	YES, @Transactional 내에서만 동작
실무 예시	DB 저장 후 이메일, 슬랙 전송, 알림 발송 등
지원 모드	상수 / 설명
	AFTER_COMMIT / 트랜잭션 **성공 후 (기본값)**
	BEFORE_COMMIT / 커밋 직전
	AFTER_ROLLBACK / 롤백 시
	AFTER_COMPLETION / 트랜잭션 종료 시 (성공/실패 무관)

구현 주의 사항

- @Transactional 메서드 내부에서 publishEvent()가 호출되어야 한다.
- 테스트 코드 등에서는 트랜잭션이 제대로 적용되지 않으면 리스너가 실행되지 않을 수 있음.
- 비동기 처리와 함께 쓸 경우, @Async 는 트랜잭션 이후 동작을 더 분리할 수 있음.

Transactional Event Structure

[1] 이벤트 클래스

```
@Data
@AllArgsConstructor
public class PurchaseCompletedEvent {
    private String userEmail;
    private String productName;
}
```

[2] 서비스에서 트랜잭션 처리 + 이벤트 발행

```
@Service
public class PurchaseService {
    @Autowired
    private ApplicationEventPublisher publisher;

    @Transactional
    public void completePurchase(String email, String product) {
        //  DB 에 구매 정보 저장 (생략)
        publisher.publishEvent(new PurchaseCompletedEvent(email, product));
        System.out.println("구매 완료 처리 중...");
    }
}
```

[3] 트랜잭션 이벤트 리스너

```
@Component
public class PurchaseEventListener {

    @TransactionalEventListener(phase = TransactionPhase.AFTER_COMMIT)
    public void sendReceipt(PurchaseCompletedEvent event) {

        System.out.println("이메일 전송: " +
event.getUserEmail() + " - " + event.getProductName());
    }
}
```

구매 완료 처리 중...

(트랜잭션 커밋 후) 이메일 전송: user@domain.com - MacBook Air

[실습 SpringBootLab09 이벤트 구현]

패키지	주제	특징
com.sec01	동기 이벤트	기본 이벤트 흐름 학습용
com.sec02	비동기 이벤트	@Async, @EnableAsync 사용
com.sec03	커스텀 이벤트	POJO + Lombok 기반
com.sec04	Built-in 이벤트	Spring 자체 이벤트 리스닝 (ContextRefreshedEvent, 등)

디렉토리구조

```
SpringBootLab09/
├─── src/ main/ java/
│        └─── com/
│             └─── sec/
│                  ├─── sec01/              # ① 동기 이벤트 (Synchronous)
│                  │    ├─── SyncController.java
│                  │    ├─── SyncUserService.java
│                  │    ├─── UserRegisteredEvent.java
│                  │    └─── SyncUserEventListener.java
│                  │
│                  ├─── sec02/              # ② 비동기 이벤트 (Asynchronous)
│                  │    ├─── AsyncController.java
│                  │    ├─── AsyncUserService.java
│                  │    ├─── UserRegisteredEvent.java
│                  │    ├─── AsyncUserEventListener.java
│                  │    └─── AsyncConfig.java
│                  │
│                  ├─── sec03/              # ③ 커스텀 이벤트 (Custom POJO)
│                  │    ├─── CustomEventController.java
│                  │    ├─── OrderService.java
│                  │    ├─── OrderCreatedEvent.java
│                  │    └─── OrderEventListener.java
│                  │
│                  └─── sec04/              # ④ Built-in 이벤트 (Spring 내장 이벤트)
│                       └─── LifecycleEventListener.java
```

테스트 예시

URL 예시	설명
POST /sync/register?username=kim	동기 이벤트 발행
POST /async/register?username=lee	비동기 이벤트 발행
POST /custom/order?name=keyboard	커스텀 이벤트 발행
(앱 실행/종료)	Built-in 이벤트 자동 감지 로그 확인