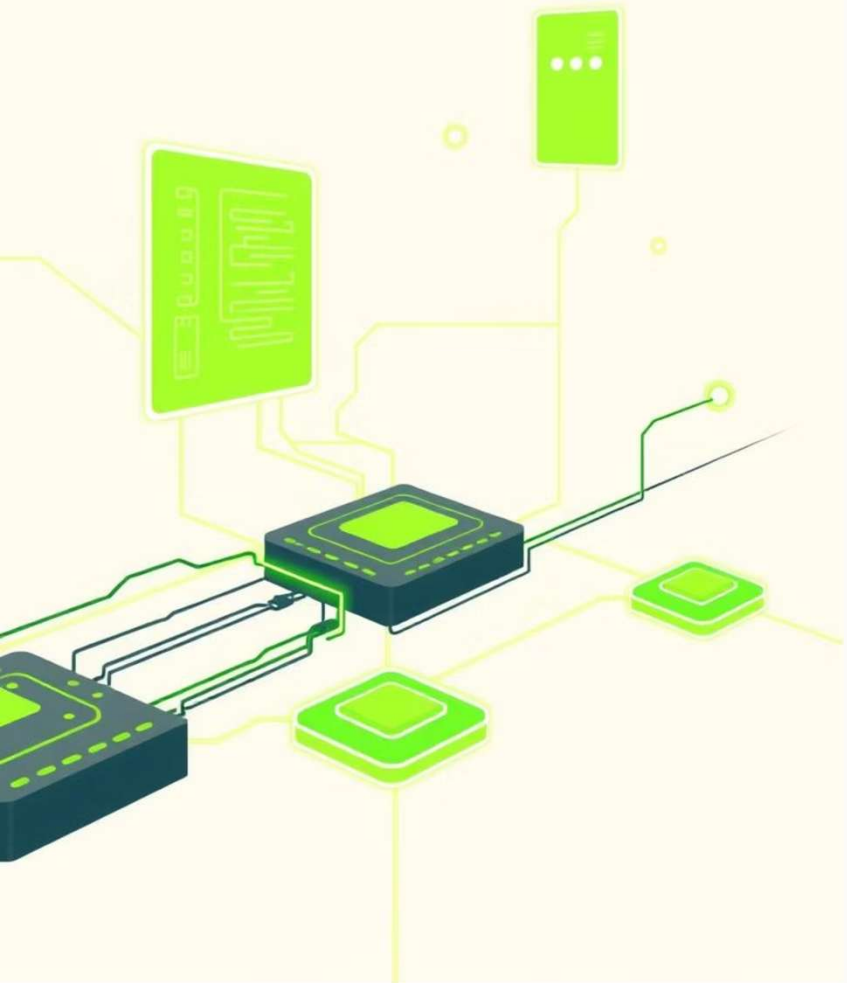


관점 지향 프로그래밍 (AOP)

소프트웨어 개발의 효율성을 높이는 강력한 패러다임

박건영 이우성 홍석민

발표 목차

1

개요

AOP의 기본 개념

2

용어 정리

주요 용어 설명

3

핵심 원리

AOP의 동작 방식

4

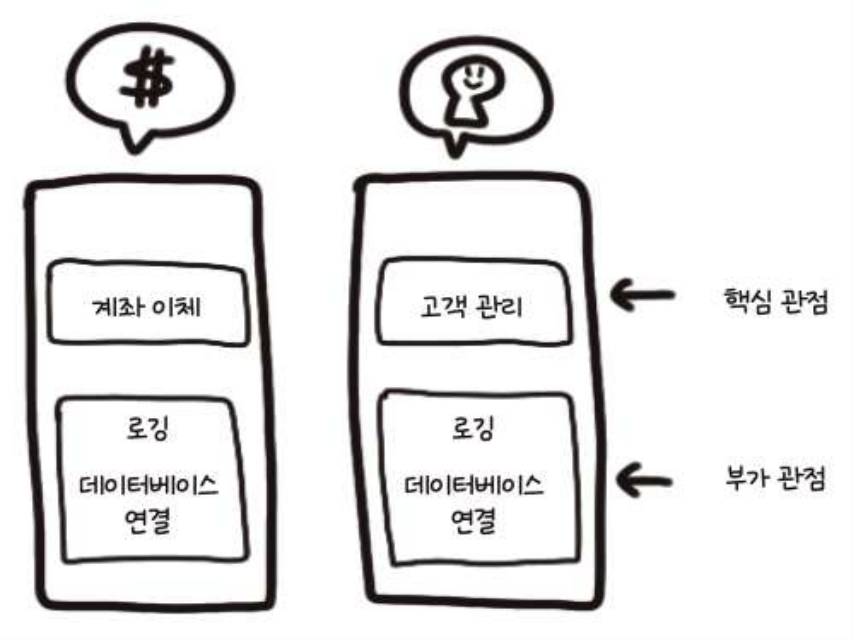
적용 사례

실제 활용 예시

5

추가 학습

주제를 통해 알게 된 점

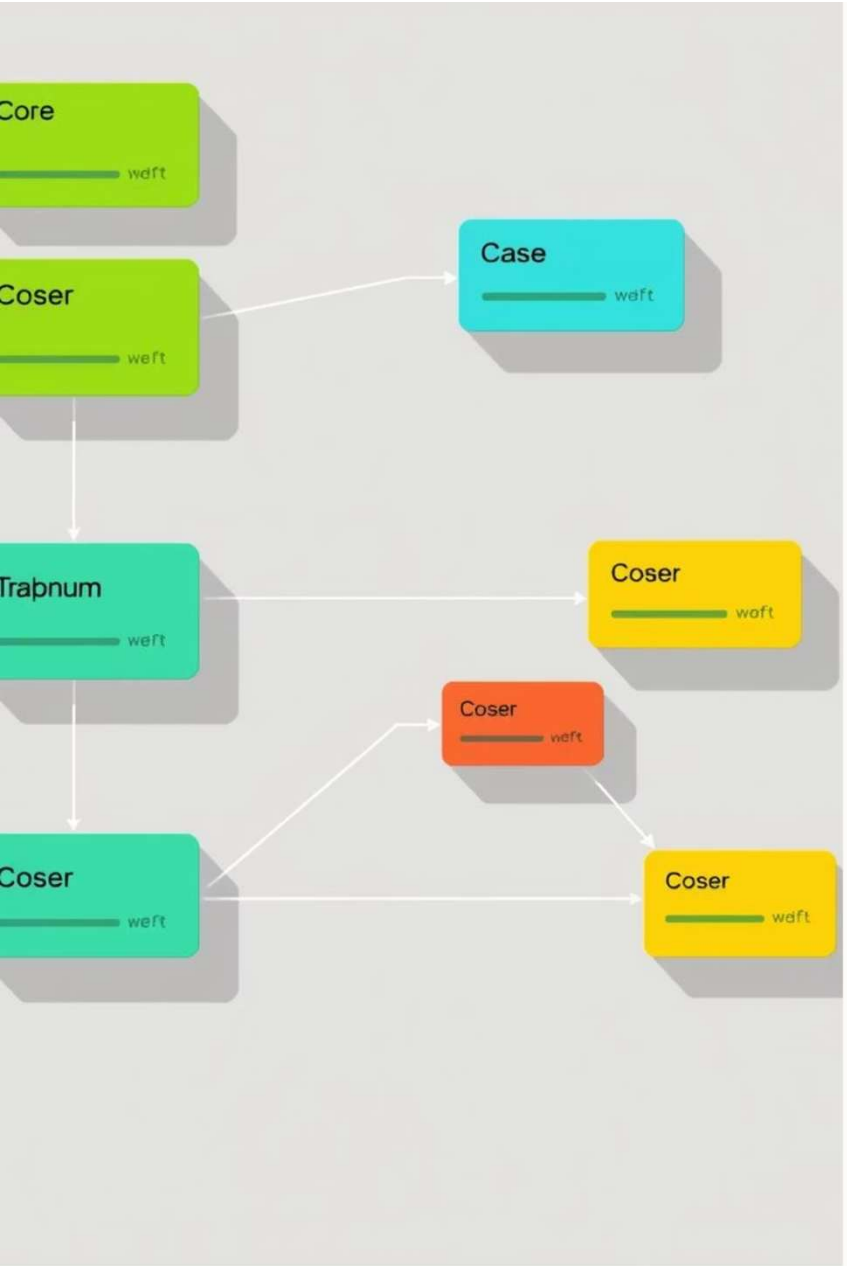


AOP 개요

관점 지향 프로그래밍 (AOP)은 핵심 기능은 그대로 유지하면서, 로깅, 보안 등 **공통 부가 기능을 별도의 Aspect로 모아** 유지보수 및 확장을 쉽게 관리하는 개념입니다.

핵심 개념 용어 정리

Aspect	공통 기능을 담은 모듈 (예: 로깅, 보안)
Advice	Aspect가 언제 실행될지 정의 (예: 메서드 실행 전/후)
JoinPoint	Advice가 적용될 수 있는 지점 (예: 메서드 호출)
Pointcut	Advice를 적용할 JoinPoint를 선별하는 규칙
Target	핵심 기능을 가진 클래스
Weaving	Advice를 Target에 적용하는 과정



AOP의 주요 개념 및 원리

AOP는 소프트웨어 개발에서 **공통 기능을 핵심 로직과 분리하여 모듈화**하는 프로그래밍 패러다임입니다. 로깅, 보안, 트랜잭션 관리, 예외 처리 같은 기능을 여러 클래스에 반복하지 않고 한 곳에서 정의하고 자동으로 적용할 수 있게 해줍니다.

AOP 동작 원리

1

클라이언트 호출

클라이언트가 Target 객체의 메서드를 호출합니다.

2

프록시 개입

Spring이 생성한 프록시 객체가 호출을 가로챕니다.

3

Advice 실행

트랜잭션 시작, 로그 기록 등 Advice가 실행됩니다.

4

메서드 실행

실제 Target 메서드가 실행됩니다.

5

후처리 Advice

커밋, 알림 전송 등 후처리 Advice가 실행됩니다.

Spring AOP는 프록시 기반으로 동작하며, 메서드 수준 JoinPoint만 지원합니다.

AOP의 장점

중복 코드 제거

공통 기능을 한 곳에 정의하여 코드 중복을 줄입니다.

유지보수 용이

핵심 로직과 공통 로직을 분리하여 유지보수를 쉽게 합니다.

생산성 향상

개발자가 핵심 로직에 집중할 수 있게 합니다.

재사용성 증가

Aspect를 여러 클래스에 적용하여 재사용성을 높입니다.

AOP 사용 시 주의할 점

복잡성 증가

과도한 사용은 오히려 시스템의 복잡성을 증가시킬 수 있습니다.

디버깅 어려움

런타임에 프록시가 개입하여 흐름 추적이 까다로울 수 있습니다.

private 메서드 적용 불가

프록시는 외부 호출만 감지 가능하므로 private 메서드에는 적용할 수 없습니다.

AOP 활용 사례

트랜잭션 관리

DB 작업 전후에 통해 자동 처리.

보안 검사

메서드 실행 전에 사용자 권한을 자동으로 확인합니다.

로깅

모든 메서드 실행 정보를 자동으로 기록하여 시스템 상태를 파악합니다.

성능 모니터링

메서드 실행 시간을 측정하고 분석하여 성능 병목을 식별합니다.

java

```java
@SpringBootApplication
@EnableAspectJAutoProxy          // ← 프록시 생성기 활성화
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

java

```java
@Aspect
@Component
public class LoggingAspect {
    @Before("execution(* com.example.service.*.*(..))")
    public void logBefore() {
        System.out.println("로깅: 메서드 실행 전");
    }

    @After("execution(* com.example.service.*.*(..))")
    public void logAfter() {
        System.out.println("로깅: 메서드 실행 후");
    }
}
```

JAVA

```java
@Service
public class ProductService {
  public void addProduct(Product product) {
    System.out.println("상품이 추가되었습니다: " + product.getName());
  }
}
```

결론 및 추가 학습

AOP는 공통 기능을 효과적으로 관리하여 코드의 중복을 줄이고 유지보수성을 높이는 강력한 도구입니다. 하지만 과도한 사용은 복잡성을 증가시킬 수 있으므로 적절한 활용이 중요합니다.

추가적으로 알게 된 점

AOP는 Spring 프레임워크에서 특히 강력하게 활용되며, 개발 생산성 향상에 크게 기여합니다. 프록시 기반 동작 방식과 메서드 수준 JoinPoint 지원을 이해하는 것이 중요합니다.

감사합니다