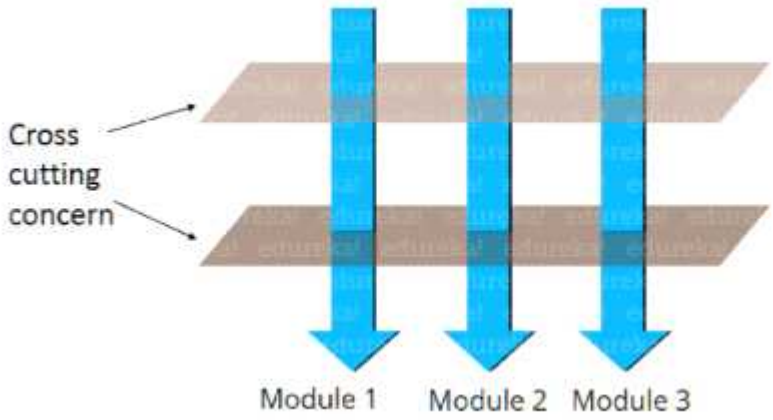
AOP (Aspect-Oriented Programming)



이미지출처 : https://www.edureka.co/blog/spring-aop-tutorial/

1. 개요

Aspect-Oriented Programming (AOP)는 소프트웨어 개발에서 관심사(기능)를 모듈화하여 코드의 재사용성과 유지보수성을 향상시키는 프로그래밍 패러다임이다. 주로 로깅, 보안, 트랜잭션 관리와 같은 횡단 관심사(Cross-Cutting Concerns)를 분리하여 코드의 핵심 기능과 보조 기능을 분리하는 데 사용된다.

2. AOP의 장점

AOP는 특히 대규모 시스템에서 공통 관심사를 효율적으로 관리하는 데 유용하며 코드의 가독성과 유지보수성을 크게 향상시킨다.

2-1) 모듈화
- 공통 관심사를 분리하여 코드의 모듈화를 높인다.

2-2) 유지보수성
- 공통 로직을 한 곳에 모아 관리하기 때문에 유지보수가 쉬워진다.

2-3) 재사용성
- 여러 곳에서 공통 로직을 재사용할 수 있다.

3. 핵심 개념

3-1) 관심사(Concern)

프로그램의 특정 기능이나 작업. 예를 들어, 로깅(logging), 보안(security), 트랜잭션 관리(transaction management) 등이 있다.

3-2) 횡단 관심사(Cross-Cutting Concerns)

여러 모듈에 걸쳐 있는 관심사. 예를 들어, 로깅은 프로그램의 여러 부분에서 필요하지만, 각 부분마다 로깅 코드를 추가하는 것은 비효율적이다.

3-3) 애스펙트(Aspect)

횡단 관심사를 모듈화한 것이다. 예를 들어, 로깅 기능을 하나의 애스펙트로 작성한다.

3-4) 조인 포인트(Join Point)

프로그램 실행 중에 애스펙트 코드를 삽입할 수 있는 지점이다. 예를 들어, 메소드 호출, 예외 발생 등이 조인 포인트가 될 수 있다.

3-5) 어드바이스(Advice)

조인 포인트에서 실행되는 코드이다. 애스펙트가 무엇을, 언제, 어디서 할지 정의한다.

@Around : 대상 객체의 메서드 실행 전,후 및 예외 발생 모두 실행한다.
@Before : 대상 객체의 메서드 호출전에 수행한다.
@After : 대상 객체의 메서드 실행도중 예외 발생 여부와 상관없이 메서드 실행 후 실행한다.
@AfterReturning : 대상 객체의 메서드가 실행 도중 예외없이 실행 성공한 경우에 실행한다.
@AfterThrowing : 대상 객체의 메서드가 실행 도중 예외가 발생한 경우에 실행한다.

3-6) 포인트컷(Pointcut): 어드바이스가 적용될 조인 포인트를 정의한다. 예를 들어, 특정 메소드나 클래스에서만 어드바이스를 적용할 수 있다.

*** execution 표현식**

execution 표현식은 포인트컷을 정의하기 위해 사용되는 구체적인 표현식이다. execution 표현식은 메소드의 시그니처를 기준으로 어드바이스를 적용할 메소드를 지정한다. 스프링 AOP에서 가장 많이 사용되는 포인트컷 표현식 중 하나이다.

execution(modifiers-pattern ret-type-pattern declaring-type-pattern name-pattern(param-pattern) throws-pattern)

modifiers-pattern: 접근 제어자 (public, protected 등) 패턴 (선택 사항)
ret-type-pattern: 반환 타입 패턴 (예: *, void, java.lang.String 등)
declaring-type-pattern: 메소드를 선언한 타입 패턴 (선택 사항)
name-pattern: 메소드 이름 패턴
param-pattern: 파라미터 패턴 (예: .., String, int, String 등)
throws-pattern: 예외 패턴 (선택 사항)

execution(접근제어자패턴 리턴타입패턴 패키지이름패턴.클래스이름패턴.메서드이름패턴(파라미터패턴) 예외처리패턴)

패키지
- '..'을 사용하여 해당 패키지의 하위 패키지까지 타겟할 수 있다.
com.spring.aop > com.spring.aop패키지를 타겟
com.spring.aop.. > com.spring.aop로 시작하는 하위의 모든 패키지를 타겟

리턴타입
- '*'을 사용하여 모든 데이터 타입을 타겟할 수 있다.

* > 모든 리턴 타입 타겟
void > 리턴 타입이 void인 메서드만 타겟
!void > 리턴 타입이 void가 아닌 메서드만 타겟

매개 변수
- '*'을 사용하여 1개의 파라메타의 모든데이터를 타겟할 수 있다.
- '..'을 사용하여 0개 이상의 모든 파라메타를 타겟할 수 있다.
(..) > 0개 이상의 모든 파라미터 타겟
(*) > 1개의 파라미터만 타겟
(*,*) > 2개의 파라미터만 타겟
(String,*) > 2개의 파라미터중 첫번째 파라미터가 String타입만 타겟

샘플 예시

execution(void set*(..))
리턴 타입이 void이고 메서드 이름이 set으로 시작하고 파라미터가 0개 이상인 메서드 타겟

execution(* abc.*.*())
모든 리턴타입 매칭 , abc패키지에 속한 모든 클래스의 파라미터가 없는 모든 메서드 타겟

execution(* abc..*.*(..))
모든 리턴타입 매칭 , abc패키지 및 하위 패키지에 있는 모든 클래스의 파라미터가 0개 이상인 메서드 타겟

execution(long com.spring.aop.Boss.work(..))
리턴 타입인 long이며 com.spring.aop 패키지 안의 Boss클래스의 work 메서드 타겟

execution (* get*(*))
모든 리턴타입 매칭 , 이름이 get으로 시작하고 파라미터가 한 개(모든타입)인 메서드 타겟

execution(* get*(*,*))
모든 리턴타입 매칭 , 이름이 get으로 시작하고 파라미터가 2개인(모든타임) 메서드 타겟

execution(* read*(int,..))
메서드 이름이 read로 시작하고 첫번째 파라미터 타입이 int이며 한개 이상의 파라미터를 갖는 메서드 타겟

3-7) 위빙(Weaving)
- 애스펙트를 프로그램의 특정 지점에 적용하는 과정이다. 컴파일 시점, 클래스 로드 시점, 런타임 시점 등에서 이루어질 수 있다.

4. 예시를 통한 설명

예시 시나리오: 은행 시스템의 트랜잭션 관리
관심사 분리: 트랜잭션 관리, 로깅, 보안 등의 기능을 각각 분리한다.
횡단 관심사: 트랜잭션 관리는 여러 서비스 메소드에서 공통으로 필요로한다.
애스펙트 정의: 트랜잭션 관리를 하나의 애스펙트로 작성한다.

```
@Aspect
public class TransactionAspect {
    @Before("execution(* com.bank.service.*.*(..))")
    public void beginTransaction() {
        System.out.println("트랜잭션 시작");
        // 트랜잭션 시작 로직
    }

    @After("execution(* com.bank.service.*.*(..))")
    public void commitTransaction() {
        System.out.println("트랜잭션 커밋");
        // 트랜잭션 커밋 로직
    }

    @AfterThrowing("execution(* com.bank.service.*.*(..))")
    public void rollbackTransaction() {
        System.out.println("트랜잭션 롤백");
        // 트랜잭션 롤백 로직
    }
}
```

- 조인 포인트와 어드바이스
execution(* com.bank.service.*.*(..))는 서비스 패키지의 모든 메소드 호출을 조인 포인트로 지정한다. 각 메소드 호출 전에 트랜잭션을 시작하고, 메소드가 성공적으로 종료되면 트랜잭션을 커밋하며, 예외가 발생하면 트랜잭션을 롤백한다.

- 포인트컷
execution(* com.bank.service.*.*(..))는 서비스 패키지의 모든 메소드를 대상으로 한다.

- 위빙
위 애스펙트는 런타임에 적용되어 트랜잭션 관리 로직을 각 메소드 호출 시점에 삽입한다.