



스프링 **AOP**

# AOP (Aspect Oriented Programming)

- 횡단 관심사 (crosscutting concerns)를 구현하는 도구
- 동일한 구현을 효과적으로 다수의 객체에 적용할 수 있는 방법
- 객체지향 프로그래밍 기법 보완
- 구현 방법에 따라 정적 AOP와 동적 AOP로 구분
  - 정적 AOP : 컴파일 시점에 대상 객체에 바이트코드 수준의 변경을 적용
  - 동적 AOP : 런타임에 프록시 객체 생성

# 용어

## ■ 조인포인트

- 애플리케이션 실행의 특정 지점을 의미
- AOP에서 횡단 관심사를 적용하는 위치를 표시하는데 사용

## ■ 어드바이스

- 특정 조인포인트에 적용(실행)할 코드
- 실행시점에 따라 Before Advice, After Advice 등으로 구현

## ■ 포인트컷

- 여러 조인포인트의 집합으로 어드바이스를 실행하는 위치 표시

## ■ 애스팩트

- 어드바이스와 포인트컷을 조합해서 횡단 관심사에 대한 코드와 그것을 적용할 지점을 정의한 것

## ■ 위빙

- 애스팩트를 실제 적용하는 과정 (정적 / 동적 방식이 구분되는 조건)

## ■ 타겟

- 어드바이스가 적용된 객체

# 스프링 AOP

- 스프링은 프록시 기반의 동적 AOP 지원
  - 메서드 호출 Joinpoint만 지원
  - 대상 객체가 인터페이스를 구현한 경우 자바 리플렉션 API를 이용해서 프록시 생성
  - 대상 객체가 인터페이스를 구현하지 않은 경우 CGLIB를 이용해서 프록시 생성
- 완전한 AOP 기능을 제공하지 않음 → JEE 애플리케이션 구현에 필요한 수준의 기능만 제공
- 지원하는 구현 방식
  - XML 스키마 기반 POJO 클래스를 이용한 AOP 구현
  - @Aspect Annotation 기반 AOP 구현
  - 스프링 API를 이용한 AOP 구현

# 스프링 AOP (continued)

## ▪ 구현 가능한 Advice 종류

종류	설명
Before Advice	메서드 호출 전 공통 기능 수행
After Returning Advice	메서드가 정상적으로 반환한 후 공통 기능 수행
After Throwing Advice	메서드 실행 중 예외가 발생하는 경우 공통 기능 수행
After Advice	예외 발생 여부와 상관 없이 메서드 실행 후 공통 기능 수행
Around Advice	메서드 실행 전, 후 또는 예외 발생 시점에 공통 기능 수행

## 스프링 AOP 의존성 패키지 정의 (maven build 설정)

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-aop</artifactId>
  <version>${spring-framework.version}</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-aspects</artifactId>
  <version>${spring-framework.version}</version>
</dependency>
```

# XML 기반 AOP 설정

## ▪ Advice 클래스 정의

```
public class LoggingAspect {  
    |  
    private Logger log = LoggerFactory.getLogger(getClass());  
}
```

## ▪ Advice 메서드 정의

```
// before 어드바이스  
public void logBefore(JoinPoint joinpoint) {  
    String message = buildJoinpoint(joinpoint);  
    message += "메서드 실행 시작";  
    log.info(message);  
}  
// after 어드바이스  
public void logAfter(JoinPoint joinpoint) {  
    String message = buildJoinpoint(joinpoint);  
    message += "메서드 실행 공통 종료";  
    log.info(message);  
}  
//after-returning 어드바이스  
public void logAfterReturning(JoinPoint joinpoint) {  
    String message = buildJoinpoint(joinpoint);  
    message += "메서드 실행 정상 종료";  
    log.info(message);  
}
```

```
// after-throwing 어드바이스  
public void logAfterThrowing(JoinPoint joinpoint) {  
    String message = buildJoinpoint(joinpoint);  
    message += "메서드 실행 예러";  
    log.info(message);  
}  
// around 어드바이스  
public void logAround(ProceedingJoinPoint joinpoint) throws Throwable {  
    long start = System.currentTimeMillis();  
    log.info("===== 시작 =====");  
    String message = buildJoinpoint(joinpoint);  
    message += "메서드 실행 시작";  
    log.info(message);  
    joinpoint.proceed(); // 메서드 호출  
    message = buildJoinpoint(joinpoint);  
    message += "메서드 실행 종료";  
    long end = System.currentTimeMillis();  
    long duration = end - start;  
    log.info("실행 시간 : " + duration + " 밀리초");  
    log.info("===== 종료 =====");  
}
```

# XML 기반 AOP 설정

- Advice 메서드의 JoinPoint 전달인자를 이용해서 호출된 메서드 정보 접근
  - Around Advice 메서드는 ProceedingJoinPoint 형식

```
// around 어드바이스
public void logAround(ProceedingJoinPoint joinpoint) throws Throwable {
    long start = System.currentTimeMillis();
    log.info("===== 시작 =====");
    String message = buildJoinpoint(joinpoint);
    message += "메서드 실행 시작";
    log.info(message);
    joinpoint.proceed();           // 메서드 호출
    message = buildJoinpoint(joinpoint);
    message += "메서드 실행 종료";
    long end = System.currentTimeMillis();
    long duration = end - start;
    log.info("실행 시간 : " + duration + " 밀리초");
    log.info("===== 종료 =====");
}
```

```
private String buildJoinpoint(JoinPoint joinpoint) {
    String className = joinpoint.getTarget().getClass().getName();
    String methodName = joinpoint.getSignature().getName();
    String message = className + " 클래스의 " + methodName + "(" ;
    Object [] args = joinpoint.getArgs();
    for(int i = 0; i < args.length; ++i){
        Object arg = args[i];
        message += arg.getClass().getTypeName();
        if(i != args.length - 1 )
            message += ", ";
    }
    message += " ) ";
    return message;
}
```



# XML 기반 AOP 설정

## ▪ Aspect 설정

```
<bean id="loggingAspect" class="com.ensoa.order.aspect.LoggingAspect"/>
```

```
<aop:config>
```

AOP 설정 영역 지정

```
<aop:aspect ref="loggingAspect">
```

Aspect 설정

Pointcut 설정

```
<aop:pointcut expression="execution(* com.ensoa.order.*.*.get*(..))  
                || execution(* com.ensoa.order.*.*.find*(..))" id="getLogging"/>  
<aop:pointcut expression="execution(* com.ensoa.order.*.*.save*(..))" id="saveLogging"/>  
<aop:pointcut expression="execution(* com.ensoa.order.service.*.update*(com.ensoa.order.domain.Customer))  
                and args(customer)" id="updateLogging"/>
```

```
<aop:before method="logBefore" pointcut-ref="getLogging"/>  
<aop:before method="logBeforeUpdate" pointcut-ref="updateLogging"/>  
<aop:after method="logAfter" pointcut-ref="getLogging"/>  
<aop:after-returning method="logAfterReturning" pointcut-ref="getLogging"/>  
<aop:after-throwing method="logAfterThrowing" pointcut-ref="getLogging"/>  
<aop:around method="logAround" pointcut-ref="saveLogging"/>
```

Advice 설정

```
</aop:aspect>
```

```
</aop:config>
```

# XML 기반 AOP 설정

## ▪ Pointcut 구문

지명자	설명
execution()	메서드를 조인포인트 매치 (메서드 패턴)
within()	타입 범위로 조인포인트 매치 (패키지 또는 클래스 패턴)
bean()	빈 이름으로 조인포인트 매치
target()	특정 타입을 대상으로 조인포인트 매치 (명시적 타입)
this()	AOP 프록시 빈 인스턴스를 대상으로 조인포인트 매치
args()	전달인자가 해당 타입인 메서드에 조인포인트 매치

# Pointcut 구문 형식

## ■ 기본형식

- execution(수식어패턴? 리턴타입패턴 클래스이름패턴?이름패턴(전달인자패턴))
- bean(Beans 이름)
- within(클래스 또는 패키지 패턴)

## ■ 와일드카드

- ?는 선택적 사용 항목
- \*는 All
- ..은 0 or more

- 두 개 이상의 Pointcut을 and, or, not, &&, ||, ! 으로 조합 및 수식 가능

# Pointcut 사용 사례

`execution(public void set*(..))`

public 접근성, void 반환, 이름이 set으로 시작, 전달인자 0개 이상

`execution(* com.example.springaop.*.*())`

com.example.springaop 패키지의 모든 클래스의 전달인자 없는 모든 메서드

`execution(* com.example.springaop..*.*(..))`

com.example.springaop 및 하위 패키지 내 모든 클래스의 모든 메서드

`execution(* com.example.springaop..ClassName.method(..))`

com.example.springaop 패키지 및 모든 하위 패키지 내의 ClassName 클래스의 모든 오버로딩된 이름이 method인 메서드

# Annotation을 이용한 AOP 설정

## ▪ AOP Annotation 종류

Annotation	XML 설정
@Aspect	<aop:aspect>
@Pointcut	<aop:pointcut>
@Before	<aop:before>
@After	<aop:after>
@AfterReturning	<aop:afterReturning>
@AfterThrowing	<aop:afterThrowing>
@Around	<aop:around>

# Annotation을 이용한 AOP 설정

## ▪ AOP Annotation 활성화

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:aop="http://www.springframework.org/schema/aop" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">

  <aop:aspectj-autoproxy />
```

# AspectJ Annotation 기반 설정 1

Aspect 등록

Pointcut & Advice 등록

```
@Aspect
public class ArticleCacheAspect implements Ordered {

    private Map<Integer, Article> cache = new HashMap<Integer, Article>();

    @Around("execution(public * *..ReadArticleService.*(..))")
    public Article cache(ProceedingJoinPoint joinPoint) throws Throwable {
        Integer id = (Integer) joinPoint.getArgs()[0];
        Article article = cache.get(id);
        if (article != null) {
            System.out.println("[ACA] 캐시에서 Article[" + id + "] 검색");
            return article;
        }
        Article ret = (Article) joinPoint.proceed();
        if (ret != null) {
            cache.put(id, ret);
            System.out.println("[ACA] 캐시에 Article[" + id + "] 추가");
        }
        return ret;
    }

    @Override
    public int getOrder() {
        return 2;
    }
}
```

# AspectJ Annotation 기반 설정 2

```
@Aspect
@Order(3)
public class ProfilingAspect {
```

Aspect 등록

Pointcut 등록

```
@Pointcut("execution(public * madvirus.spring.chap05.board..*(..))")
private void profileTarget() {}
```

```
@Around("profileTarget()")
public Object trace(ProceedingJoinPoint joinPoint) throws Throwable {
    String signatureString = joinPoint.getSignature().toShortString();
    System.out.println(signatureString + " 시작");
    long start = System.currentTimeMillis();
    try {
        Object result = joinPoint.proceed();
        return result;
    } finally {
        long finish = System.currentTimeMillis();
        System.out.println(signatureString + " 종료");
        System.out.println(signatureString + " 실행 시간 : " + (finish - start) + "ms");
    }
}
```

Advice 등록