스프링 AOP

AOP (Aspect Oriented Programming)

- 횡단 관심사 (crosscutting concerns)를 구현하는 도구

  횡단 관심사는 시스템의 다른 부분에 의존하거나 영향을 미쳐야 하는 프로그램의 일부분. 이 관심사들은 디자인과 구현 면에서 시스템의 나머지 부분으로부터 깨끗하게 분해되지 못하는 경우가 있을 수 있으며 분산(코드 중복)되거나 얽히는(시스템 간의 상당한 의존성 존재) 일이 일어날 수 있다.

- 동일한 구현을 효과적으로 다수의 객체에 적용할 수 있는 방법
  - 여러 객체에 공통으로 적용할 기능을 분리해서 재사용성을 높여주는 프로그래밍 기법
  - 핵심 기능과 공통 기능의 구현 분리

- 구현 방법
  - 컴파일 시점에 코드에 공통 기능 삽입
  - 클래스 로딩 시점에 바이트 코드에 공통 기능 삽입
  - 런타임에 프록시 객체를 생성해서 공통 기능 삽입

AOP 주요 용어

용어	설명
Advice	Joinpoint에 적용할 코드 실행 시점에 따라 Before Advice, After Advice 등으로 구현
Joinpoint	애플리케이션 실행의 특정 지점 횡단 관심사를 적용하는 구체적인 위치 표시
Pointcut	여러 Joinpoint의 집합으로 Advice를 실행하는 위치 표시
Aspect	Advice와 Pointcut을 조합해서 횡단 관심사에 대한 코드와 그것을 적용할 지점을 정의한 것
Weaving	Aspect를 적용하는 과정 프록시 기반 구현 등의 AOP 구현 방식이 구분되는 기준

스프링 AOP

- 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

- 완전한 AOP 기능을 제공하지 않음
  - JEE 애플리케이션 구현에 필요한 수준의 기능만 제공
  - 프록시 기반의 동적 AOP 지원
  - 메서드 호출 Joinpoint 지원

- 지원하는 구현 방식
  - XML 설정 기반 POJO 클래스를 이용한 AOP 구현
  - @Aspect, @Pointcut, @Around 등 Annotation 기반 AOP 구현

스프링 AOP 의존성 패키지 정의 (maven build 설정)

- spring-aop와 aspectjweaver 패키지 필요

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.2.8.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.aspectj</groupId>
        <artifactId>aspectjweaver</artifactId>
        <version>1.9.6</version>
    </dependency>
</dependencies>
```

spring-context 패키지에 대한 의존성을 추가하면 spring-aop 패키지 의존성이 자동으로 추가됨

XML 기반 AOP 설정

```java
public class ExeTimeAspect {

    public Object measure(ProceedingJoinPoint joinPoint) throws Throwable {
        long start = System.nanoTime();
        try {
            Object result = joinPoint.proceed();
            return result;
        } finally {
            long finish = System.nanoTime();
            Signature sig = joinPoint.getSignature();
            System.out.printf("%s.%s(%s) 실행 시간 : %d ns\n",
                    joinPoint.getTarget().getClass().getSimpleName(),
                    sig.getName(), Arrays.toString(joinPoint.getArgs()),
                    (finish - start));
        }
    }
```

Advice 클래스 및 메서드 정의

AOP Namespace 추가

```xml
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.3.xsd">
```

AOP 설정 (pointcut, advice, aspect)

```xml
<bean id="exeTimeAdvice" class="aspect.ExeTimeAspect" />

<aop:config>
    <aop:aspect id="exeTimeAspect" ref="exeTimeAdvice">
        <aop:pointcut id="publicTarget" expression="execution(public * chap07..*(..))" />
        <aop:around method="measure" pointcut-ref="publicTarget" />
    </aop:aspect>
</aop:config>
```

AOP 설정 영역 지정

Aspect 설정

Pointcut 설정

Advice 설정

Annotation 기반 AOP 설정

Aspect 설정

```
@Aspect
public class ExeTimeAspect {

    @Pointcut("execution(public * chap07..*(..))")
    private void publicTarget() {
    }

    @Around("publicTarget()")
    public Object measure(ProceedingJoinPoint joinPoint) throws Throwable {
        long start = System.nanoTime();
        try {
            Object result = joinPoint.proceed();
            return result;
        } finally {
            long finish = System.nanoTime();
            Signature sig = joinPoint.getSignature();
            System.out.printf("%s.%s(%s) 실행 시간 : %d ns\n",
                    joinPoint.getTarget().getClass().getSimpleName(),
                    sig.getName(), Arrays.toString(joinPoint.getArgs()),
                    (finish - start));
        }
    }

}
```

Pointcut 설정

Advice 설정

AOP Annotation 활성화

```
@Configuration
@EnableAspectJAutoProxy
public class AppCtx {
    @Bean
    public ExeTimeAspect exeTimeAspect() {
        return new ExeTimeAspect();
    }
```

Aspect bean 등록

XML 설정과 Annotation 비교

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

JoinPoint 전달인자

- Advice 메서드의 JoinPoint 전달인자를 이용해서 호출된 메서드 정보 접근
- Around Advice 메서드는 ProceedingJoinPoint 형식의 전달인자를 필수적으로 사용
- Around Advice 이외의 메서드는 JoinPoint 형식의 전달인자를 선택적으로 사용

```
public class ExeTimeAspect {

    public Object measure(ProceedingJoinPoint joinPoint) throws Throwable {
        long start = System.nanoTime();
        try {
            Object result = joinPoint.proceed();
            return result;
        } finally {
            long finish = System.nanoTime();
            Signature sig = joinPoint.getSignature();
            System.out.printf("%s.%s(%s) 실행 시간 : %d ns\n",
                    joinPoint.getTarget().getClass().getSimpleName(),
                    sig.getName(), Arrays.toString(joinPoint.getArgs()),
                    (finish - start));
        }
    }
```

프록시 생성 방식

- 스프링은 Advice가 적용되는 타겟 클래스가 인터페이스를 구현한 경우 타겟 클래스 상속 타입이 아닌 인터페이스 구현 타입의 프록시 객체를 반환

- @EnableAspectJAutoProxy의 proxyTargetClass 속성을 true로 설정하면 스프링은 프록시 객체를 인터페이스 구현 타입이 아닌 타겟 클래스를 상속한 타입으로 구현

```
@Configuration
@EnableAspectJAutoProxy(proxyTargetClass = true)
public class AppCtxWithClassProxy {
    @Bean
    public ExeTimeAspect exeTimeAspect() {
        return new ExeTimeAspect();
    }
```

```
AnnotationConfigApplicationContext ctx =
        new AnnotationConfigApplicationContext(AppCtxWithClassProxy.class);

RecCalculator cal = ctx.getBean("calculator", RecCalculator.class);
long fiveFact = cal.factorial(5);
System.out.println("cal.factorial(5) = " + fiveFact);
System.out.println(cal.getClass().getName());
ctx.close();
```

Pointcut 구문 종류

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?메서드이름패턴(전달인자패턴))
  - bean(Bean 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 구문 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

Advice 적용 순서

- 동일한 PointCut에 여러 개의 Advice가 적용될 경우 호출 순서는 상황에 따라 다를 수 있음.

- 필요한 경우 @Order Annotation으로 @Advice의 적용 순서를 명시적으로 설정할 수 있음

```
@Aspect
@Order(1)
public class ExeTimeAspect {
```

```
@Aspect
@Order(2)
public class CacheAspect {
```

```
<aop:aspect id="exeTimeAspect" ref="exeTimeAdvice" order="1">
        <aop:pointcut id="publicTarget" expression="execution(public * chap07..*(..))" />
        <aop:around method="measure" pointcut-ref="publicTarget" />
</aop:aspect>

<aop:aspect id="cacheAspect" ref="cacheAdvice" order="2">
        <aop:pointcut id="cacheTarget" expression="execution(public * chap07..*(long))" />
        <aop:around method="execute" pointcut-ref="cacheTarget" />
</aop:aspect>
```

@Pointcut 재사용

```
@Aspect
public class CacheAspect2 {

    private Map<Long, Object> cache = new HashMap<>();

    @Around("aspect2.CommonPointcut.commonTarget()")
    public Object execute(ProceedingJoinPoint joinPoint) throws Throwable {
```

```
public class CommonPointcut {

    @Pointcut("execution(public * chap07..*(..))")
    public void commonTarget() {
    }

}
```

```
@Aspect
public class ExeTimeAspect2 {

    @Around("aspect2.CommonPointcut.commonTarget()")
    public Object measure(ProceedingJoinPoint joinPoint) throws Throwable {
```