Spring AOP

AOP

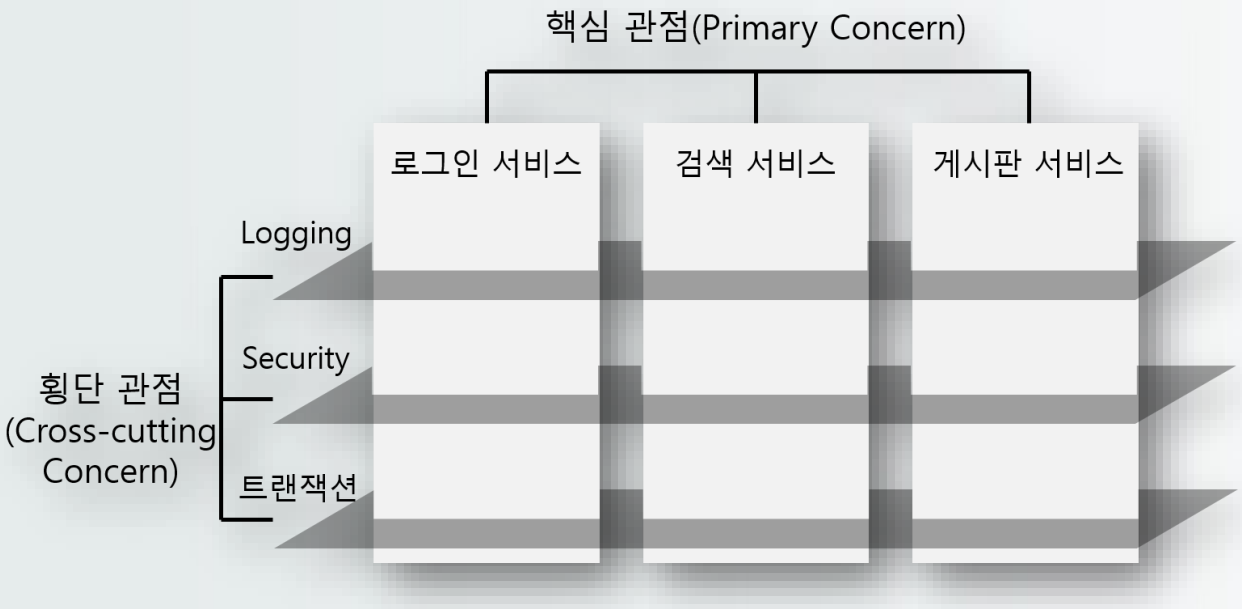
Spring AOP

Spring AOP

Spring AOP란, 관점지향 프로그래밍의 약자로 일반적으로 사용하는 클래스(Service, DAO)에서 중복되는 공통 코드 부분(ex. commit, rollback, logging..)을 별도의 영역으로 분리해 내고, 코드가 실행되기 전이나 이 후의 시점에 해당 코드를 붙여 넣음으로써 소스코드의 중복을 줄이고, 필요할 때마다 가져다 쓸 수 있게 객체화 하는 기술

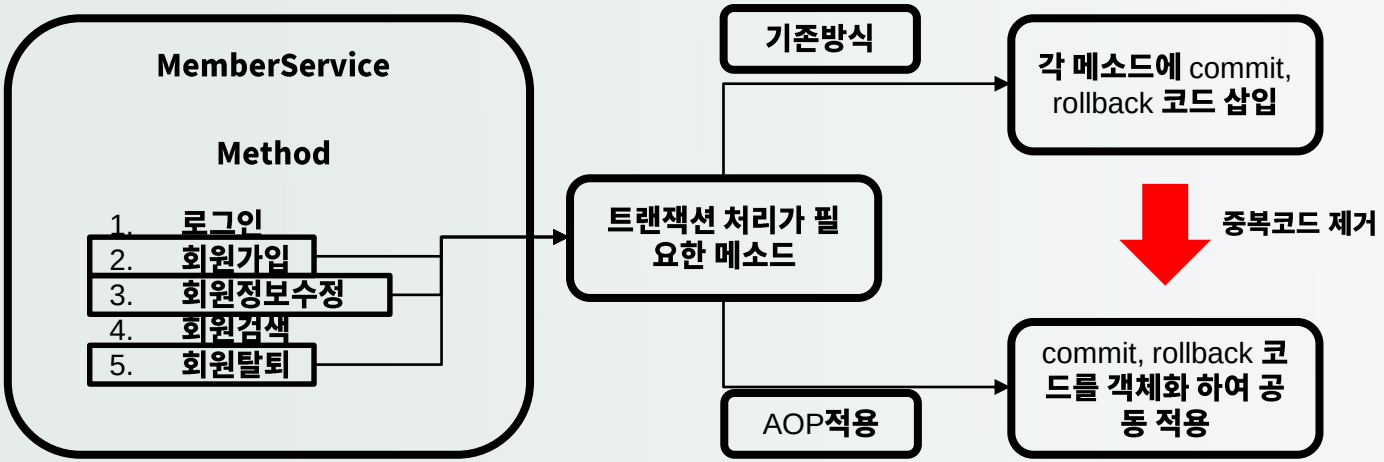
Spring AOP

Spring AOP



Spring AOP

Spring AOP



Spring AOP

Spring AOP 용어

용어	설명
Joinpoint	- **클라이언트가 호출하는 모든 비즈니스 메소드** - **일반적으로** Service**의 모든 클래스**
Pointcut	- **필터링 된 조인 포인트** - Joinpoint **중 공통기능을 적용 할 선택된 메소드**
Advice	- Pointcut**에 적용할 공통 기능의 코드**
Aspect or Advisor	- Pointcut + Advice = Aspect - **어떤** Pointcut**에 어떤** Advice**를 적용할지 결정** - Advisor**는** Aspect**와 같지만 몇몇 특수한 경우에 사용(트랜잭션 처리)**

Spring AOP

Spring AOP – Pointcut 표현식

Joinpoint 중 Advice 적용을 원하는 메소드를 필터링 할 때 사용하는 표현식

Pointcut 표현식	execution(* member.model.service..*Service.*(..))
*	- 메소드 리턴 타입
member.model.service	- 패키지 경로
*Service	- 클래스명(Service로 끝나는 모든 클래스)
*	- 메소드 명
(..)	- 매개변수

Spring AOP

Spring AOP – Pointcut 표현식

형식	예	설명
리턴 타입	*	- 모든 리턴 타입 허용
	void	- 리턴 타입이 void인 메소드만 선택
	!void	- 리턴 타입이 void가 아닌 메소드만 선택
패키지	org.kh.test	- 정확하게 org.kh.test 패키지만 선택
	org.kh.test..	- org.kh.test패키지 및 모든 하위 패키지 선택
	org.kh.test.*vice	- 패키지명이 org.kh.test로 시작하면서 마지막 패키지 이름임 vice로 끝나는 패키지

Spring AOP

Spring AOP – Pointcut 표현식

형식	예	설명
클래스	MemberService	- 정확하게 MemberService 클래스만 선택
	*Service	- 클래스 이름이 Service로 끝나는 클래스만 선택
	MemberService+	- 클래스 이름 뒤에'+'가 붙으면 해당 클래스로 부터 파생된 모든 자식클래스를 선택 - 인터페이스 뒤에 '+'가 붙으면 해당 인터페이스를 implement한 모든 클래스 선택
메소드	*(..)	- 가장 기본 설정으로 모든 메소드 선택
	*Member(..)	- 메소드 이름이 Member로 끝나는 모든 메소드 선택

Spring AOP

Spring AOP – Pointcut 표현식

형식	예	설명
매개변수	(..)	- 가장 기본 설정으로 매개변수 타입과 개수에 제약이 없음을 의미
	(*)	- 반드시 1개의 매개변수를 가지는 메소드
	(member.vo.Member)	- 매개변수로 Member를 가지는 메소드 선택, 이때 클래스의 패키지 경로를 포함해야함
	(!member.vo.Member)	- 매개변수로 Member를 가지지 않는 메소드를 선택
	(Integer,..)	- 한 개 이상의 매개변수를 가지되, 첫번째 매개변수 타입이 Integer인 메소드
	(Integer,*)	- 반드시 2 개의 매개변수를 가지되, 첫번째 매개변수 타입이 Integer인 메소드

Spring AOP

Spring AOP – Advice 동작시점

Pointcut(선택된 비지니스 메소드)이 수행될 때 Advice를 동작 시킬 시점

용어	설명
Before	- 비지니스 메소드 실행 전 동작
After Returning	- 비지니스 메소드가 성공적으로 리턴 되면 동작
After Throwing	- 비즈니스 메소드 실행 중 예외가 발생하면 동작
After	- 비즈니스 메소드 실행 된 후 무조건 동작(에러,성공 상관없음)
Around	- 메소드 호출 자체를 가로채 비즈니스 메소드 실행 전후에 처리할 로직을 삽입 가능

Spring AOP

Spring AOP – JoinPoint Interface

JoinPoint는 Spring AOP 혹은 AspectJ에서 AOP의 부가기능을 지닌 코드가 적용되는 지점을 뜻하며, Advice는 org.aspectj.lang.JoinPoint 타입의 파라미터를 어드바이스 메소드의 첫번째 매개변수로 선언해야 한다.
단, Around의 경우 JoinPoint의 하위 클래스인 ProceedingJoinPoint타입의 파라미터를 필수적으로 선언해야 한다.

Spring AOP

Spring AOP – JoinPoint Interface 메소드

메소드	설명
getArgs()	- 메소드의 매개 변수를 반환
getSignature()	- 대상 객체 메소드의 설명(메소드 명, 리턴 타입 등) 반환

※ 더 다양한 메소드가 존재하지만 상세 사항은 따로 확인
https://www.eclipse.org/aspectj/doc/next/runtime-api/index.html

Spring AOP – Signatur 객체 메소드

메소드	설명
getName()	- 클라이언트가 호출한 메소드 이름 리턴
toLongString()	- 클라이언트가 호출한 메소드의 리턴 타입, 이름, 매개변수를 리턴
toShortString()	- 클라이언트가 호출한 메소드 시그니처를 축약한 문자열로 리턴

Spring AOP

Spring AOP – ProceedingJoinPoint 메소드

ProceedingJoinPoint 인터페이스는 JoinPoint를 상속한 인터페이스로 JoinPoint 가 가진 모든 메소드를 지원하고, 추가적으로 proceed()메소드를 제공
단, ProceedingJoinPoint는 Around에서만 사용!(다른 시점에서는 JoinPoint사용)

메소드	설명
proceed()	- proceed() 메소드는 비즈니스 메소드를 수행하는 메소드로 Object 타입 객체를 리턴 하는데 이 Object 객체가 비즈니스 메소드 수행 후 리턴 하는 객체 - 다른 advice는 proceed()메소드가 필요 없지만, Around의 경우 비즈니스 로직 수행 전/후 로직을 모두 처리하기 때문에 비즈니스 메소드를 수행하는 proceed() 메소드가 필요하여 반드시 ProceedingJoinPoint 인터페이스가 필요

AOP 적용하기

Spring AOP

pom.xml을 이용한 라이브러리 추가

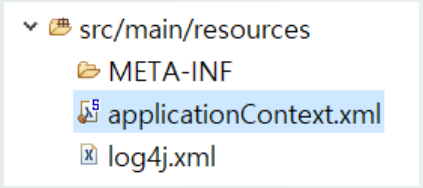
AspectJ Weaver : AOP에서 advice의 핵심 기능에 적용하는 설정파일

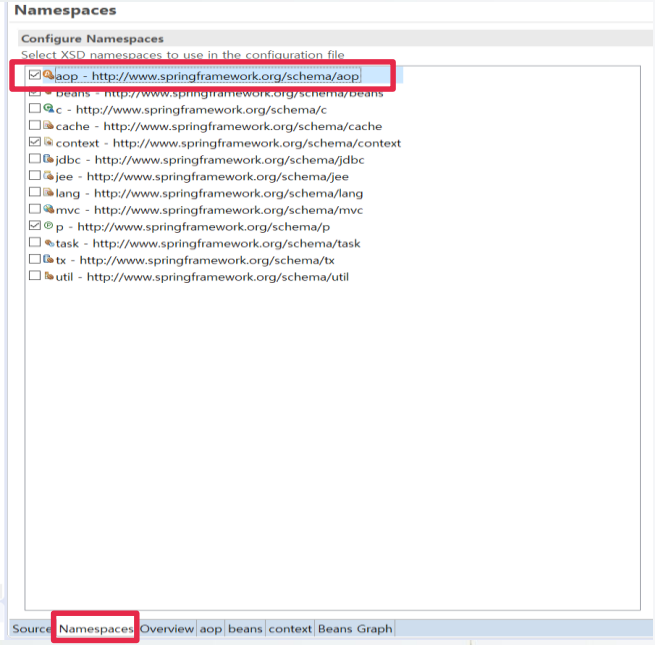
```
<!-- AspectJ -->
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjrt</artifactId>
    <version>${org.aspectj-version}</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.8.8</version>
</dependency>
```

Spring AOP

xml파일에서 aop 설정 추가

aop에서 제공하는 엘리먼트들을 사용하기 위한 namespace 추가





Spring AOP

AOP 설정 적용

```
<bean id="testAop"
      class="org.kh.firstSpring.service.common.AroundLog"/>
<!-- AOP 설정 -->
<aop:config>
   <aop:aspect id="testAspect" ref="testAop">
      <aop:around pointcut="execution(public * org.kh.firstSpring..*(..))"
            method="aroundLog" />
   </aop:aspect>
</aop:config>
```

1. AroundLog 클래스를 testAop라는 id로 bean 생성(AOP로 적용할 기능을 작성한 클래스)
2. <aop:config> : AOP 설정정보임을 나타냄
3. <aop:aspect> : aspect를 설정 → testAop를 이용하여 기능을 적용
4. <aop:around> : around 적용
   - pointcut : 적용할 비즈니스 메소드를 선택 할 포인트컷 표현식(적용 할 메소드 선택)
   - method : testAop객체 중 기능이 작성되어 있는 메소드명 선택

Spring AOP

Spring AOP – Advice 정의하는 태그

용어	설명
<aop:before>	- 메소드 실행 전에 적용되는 어드바이스 정의
<aop:around>	- 메소드 호출 이전, 이후, 예외 발생등 모든시점에 적용 가능한 어드바이스 정의
<aop:after>	- 메소드가 정상적으로 실행되는지 또는 예외를 발생시키는지 여부에 관계없이 실행되는 어드바이스 정의
<aop:after-returning>	- 메소드가 정상 실행된 후 적용되는 어드바이스 정의
<aop-throwing>	- 메소드가 예외를 발생시킬 때 적용되는 어드바이스 정의

Spring AOP

Annotation을 이용한 AOP 설정

1. 클래스 선언부에 @Aspect 어노테이션 정의
2. 해당 클래스를 객체생성해야 사용이 가능하므로 @Component 어노테이션도 함께 정의
3. xml파일에 annotation 설정 입력(<aop:aspectj-autoproxy/>)

```
@Component
@Aspect
public class AroundLog {
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.3.xsd
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.3.xsd
        http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.3.xsd">


    <aop:aspectj-autoproxy/>
```

Spring AOP

Spring AOP – Advice 정의하는 어노테이션

용어	설명
@Before(“pointcut”)	- 타겟 객체의 메소드가 실행되기 전에 실행되는 어드바이스 - JoinPoint를 통해 파라미터 정보 참조 가능
@After(“pointcut”)	- 타겟 객체 메소드가 실행되고나면 성공여부와 관계없이 모두 호출되는 어드바이스로, 반환값을 받을 수 없다.
@Around(“pointcut”)	- 타겍 객체의 메소드 호출 전과 후에 실행될 코드를 구현할 어드바이스 - ProceedingJoinPoint를 통해 파라미터와, 반환값 모두 참조 가능
@AfterReturning(“pointcut”, returning=“”)	- 타겟 객체의 메소드가 정상 실행을 마친 후 호출되는 어드바이스 - 리턴값을 참조할 때는 returning 속성에 리턴값을 저장할 변수명을 지정
@AfterThrowing(“pointcut”, throwing=“”)	- 타겟 객체의 메소드가 예외가 발생하면 호출되는 어드바이스 - 발생된 예외를 참조할 때는 throwing 속성에 발생한 예외를 저장할 변수 이름을 지정

Spring AOP

Annotation을 이용한 AOP 설정 예(Around)

```
@Around("execution(* org.kh.firstSpring..*Impl.*(..))")
public Object aroundLog(ProceedingJoinPoint pp) throws Throwable{
   //사전, 사후 처리를 모두 해결하고자 할 때 사용하는 어드바이스이다.
   String methodName = pp.getSignature().getName();

   StopWatch stopWatch = new StopWatch();
   stopWatch.start();

   Object obj = pp.proceed();

   stopWatch.stop();

   System.out.println(methodName + "() 메소드 수행에 걸린 시간 : " +
   stopWatch.getTotalTimeMillis() + "(ms)초");

   return obj;
}
```