KH정보교육원

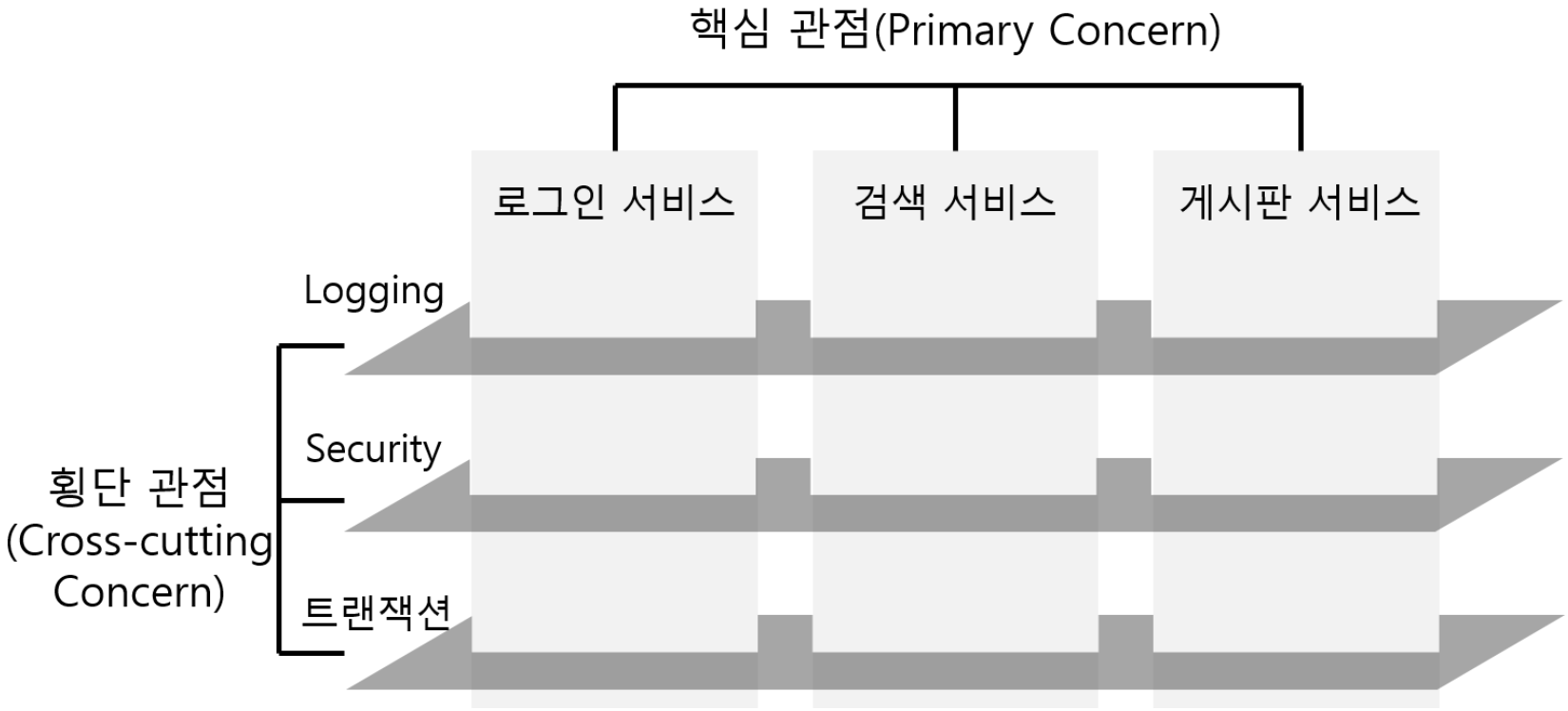
Spring AOP

Spring AOP란

Spring AOP 란, **관점 지향 프로그래밍의 약자**로 일반적으로 사용하는 클래스(Service, Dao 등) 에서 **중복되는 공통 코드** 부분(commit, rollback, log 처리)을 **별도의 영역으로 분리**해 내고, 코드가 실행 되기 전이나 이 후의 시점에 해당 코드를 붙여 넣음으로써 **소스 코드의** 중복을 줄이고, 필요할 때마다 가져다 쓸 수 있게 **객체화하는 기술**을 말한다.

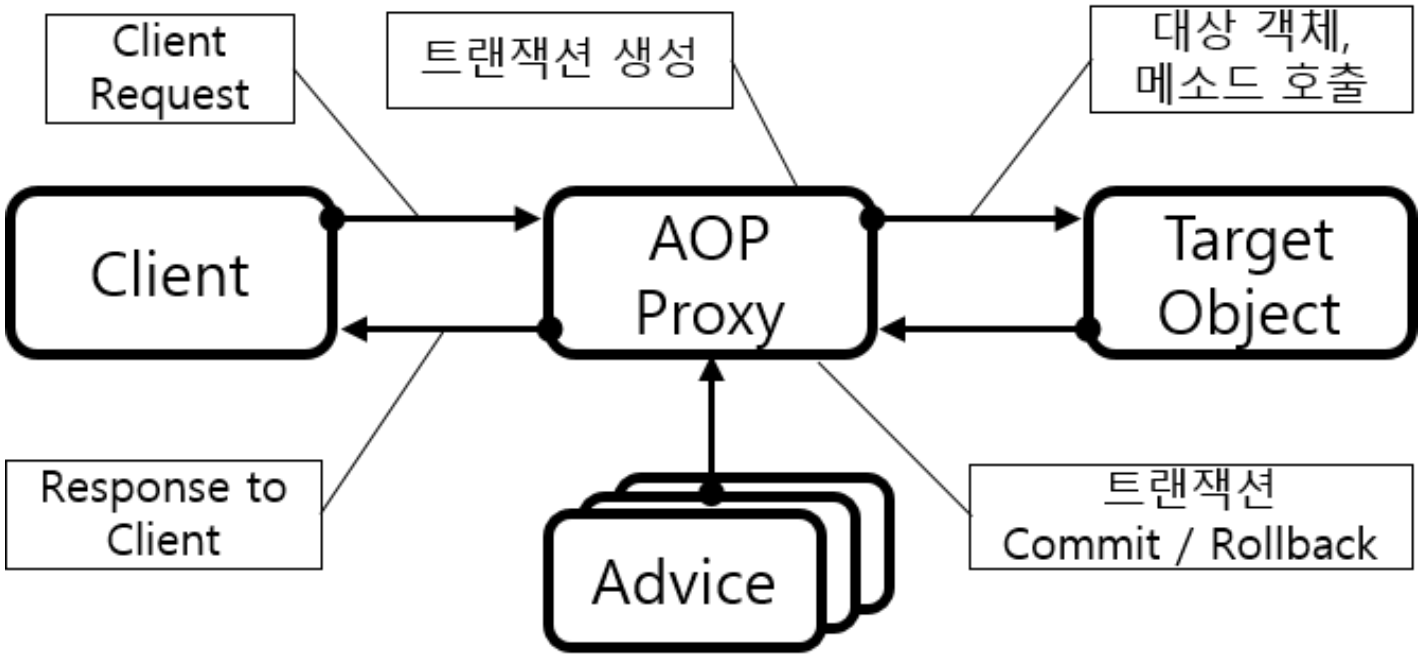
KH정보교육원

Spring AOP 개요



※ 위 이미지와 같이 공통되는 부분을 따로 빼내어 필요한 시점에 해당 코드를 추가해주는 기술을 AOP라고 말한다.

Spring AOP의 구조



※ **공통되는 부분**을 따로 빼내어 작성하는 클래스를 **Advice**라고 이야기 하며, 해당 **시점을 Joinpoint**, 그리고 그 시점에 **공통 코드를 끼워 넣는 작업을 Weaving** 이라고 말한다.

Aspect 란?

아스펙트(Aspect)는 부가기능을 정의한 코드인 어드바이스(Advice)와 어드바이스를 어디에 적용하지를 결정하는 포인트컷(PointCut)을 합친 개념이다.

Advice + PointCut = Aspect

AOP 개념을 적용하면 핵심기능 코드 사이에 끼어있는 부가기능을 독립적인 요소로 구분해 낼 수 있으며, 이렇게 구분된 부가기능 아스펙트는 런타임 시에 필요한 위치에 동적으로 참여하게 할 수 있다.

Spring AOP 의 핵심 용어

용 어	설 명	
Aspect	여러 객체에 공통으로 적용되는 기능을 분리하여 작성한 클래스	
Joinpoint	클래스의 객체(인스턴스) 생성 지점, 메소드 호출 시점, 예외 발생 시점 등 특정 작업이 시작되는 시점	
Advice	Joinpoint에 삽입되어 동작될 코드, 메소드	
	Before advice	Joinpoint 앞에서 실행
	Around Advice	Joinpoint 앞과 뒤에서 실행
	After Advice	Joinpoint 호출이 리턴되기 직전에 실행
	After Returning Advice	Joinpoint 메소드 호출이 정상적으로 종료된 후에 실행
	After Throwing Advice	예외가 발생했을 때 실행

Spring AOP 의 핵심 용어

용 어	설 명	
Pointcut	조인 포인트의 부분 집합 / 실제 Advice가 적용되는 부분	
Introduction	정적인 방식의 AOP 기술	
Weaving	작성한 Advice (공통 코드)를 핵심 로직 코드에 삽입	
	컴파일 시 위빙	컴파일 시 AOP가 적용된 클래스 파일이 새로 생성 (AspectJ)
	클래스 로딩 시 위빙	JVM에서 로딩한 클래스의 바이트 코드를 AOP가 변경하여 사용
	런타임 시 위빙	클래스 정보 자체를 변경하지 않고, 중간에 프록시를 생성하여 경유 (스프링)
Proxy	대상 객체에 Advice가 적용된 후 생성되는 객체	
Target Object	Advice를 삽입할 대상 객체	

Spring AOP

KH정보교육원

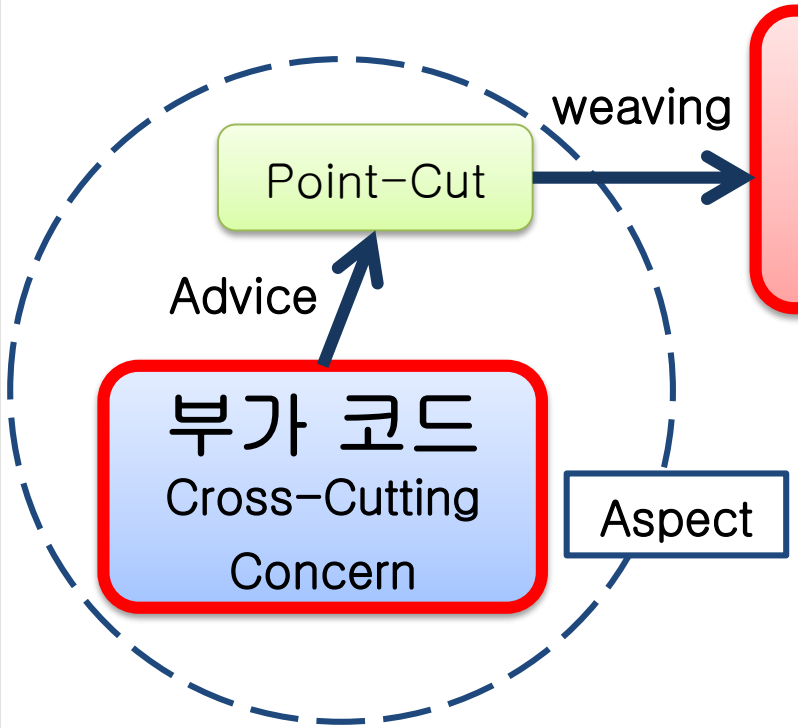
AOP 구조 정리

<기존>

주요 코드
Primary
Concern

부가 코드
Cross-Cutting
Concern

<AOP>



주요 코드
Primary
Concern

해당 업무 로직이
구현된 코드

보안, 인증 등의 부가 기능 및
시스템 전반에 걸쳐 사용되는 코드들

KH정보교육원

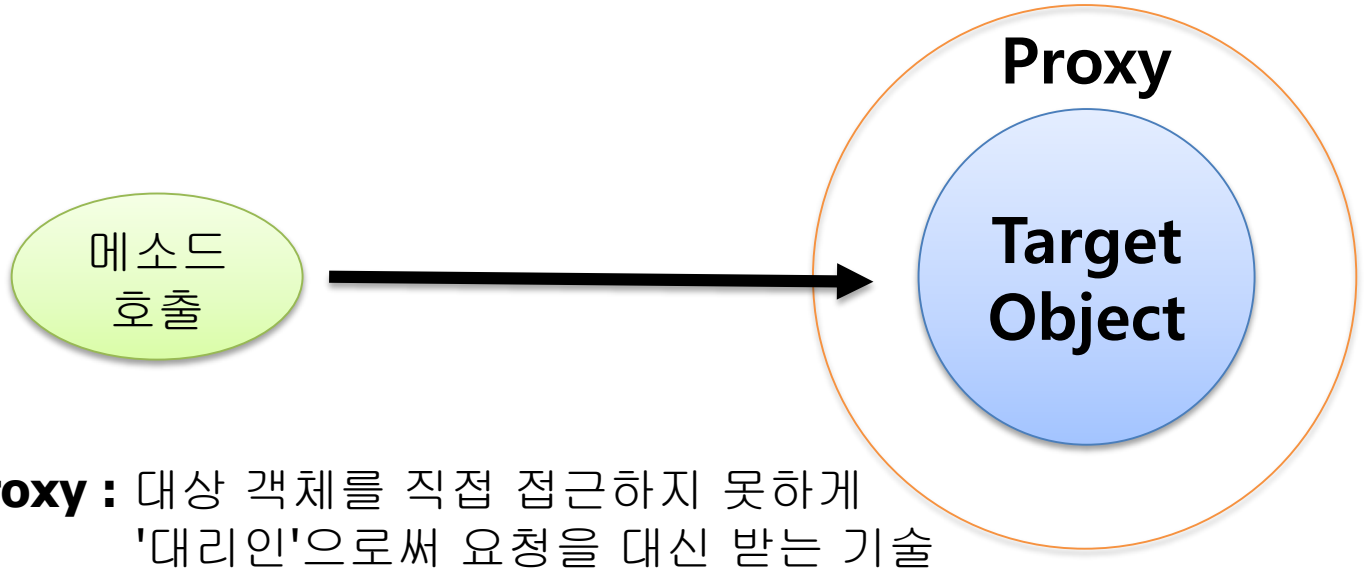
Spring AOP의 특징 및 구현 방식

Spring AOP의 특징

KH정보교육원

1) Spring은 프록시(Proxy) 기반 AOP를 지원한다.

Spring은 대상 객체(Target Object)에 대한 프록시를 만들어 제공하며, 타겟을 감싸는 프록시는 서버 Runtime 시에 생성된다. 이렇게 생성된 프록시는 대상 객체를 호출 할 때 먼저 호출되어 어드바이스의 로직을 처리 후 대상 객체를 호출한다.



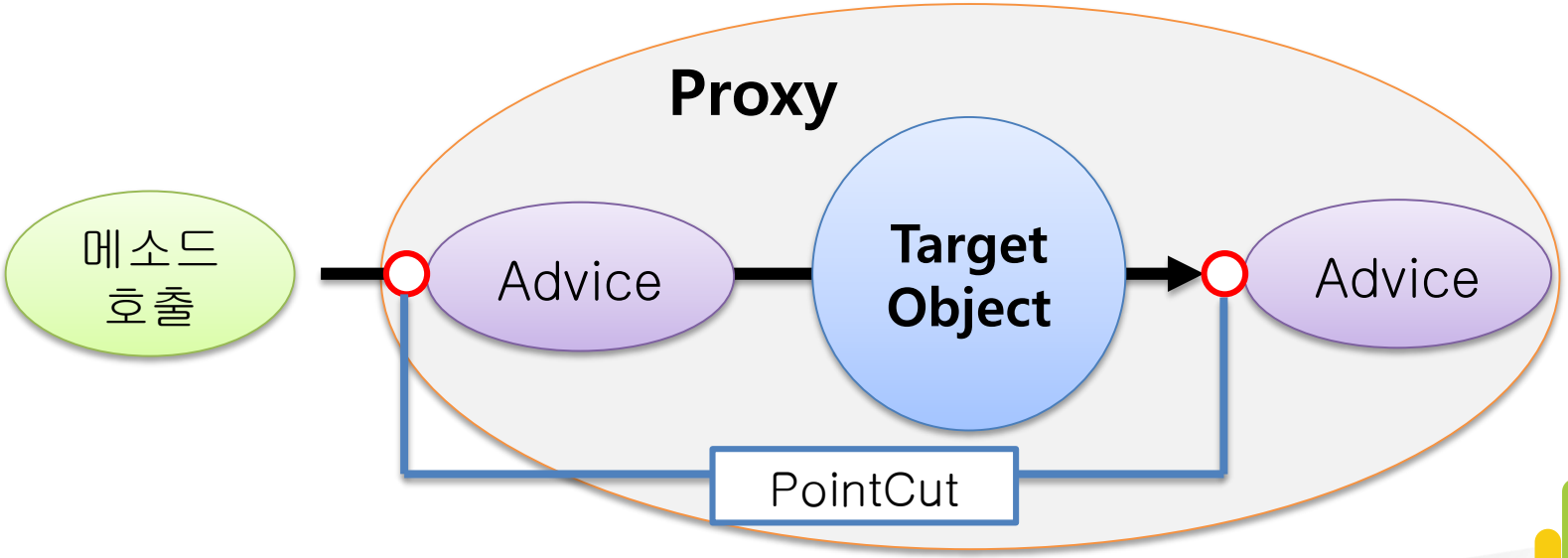
※ **Proxy :** 대상 객체를 직접 접근하지 못하게
'대리인'으로써 요청을 대신 받는 기술

Spring AOP의 특징

KH정보교육원

2) Proxy는 대상 객체의 호출을 가로챈다(Intercept).

Proxy는 그 역할에 따라 타겟 객체에 대한 호출을 가로챈 다음 어드바이스의 부가기능 로직을 수행하고 난 후에 타겟의 핵심기능 로직을 호출하거나 (전처리 어드바이스) 타겟의 핵심기능 로직 메서드를 호출한 후에 부가기능(어드바이스)을 수행한다.(후처리 어드바이스).



Spring AOP의 특징

KH정보교육원

3) Spring AOP는 메소드 조인 포인트만 지원한다.

Spring은 동적 프록시를 기반으로 AOP를 구현하기 때문에 메소드 조인 포인트만 지원한다. 즉, 핵심기능(대상 객체)의 메소드가 호출되는 런타임 시점에만 부가기능(어드바이스)을 적용할 수 있다.

하지만, AspectJ 같은 고급 AOP 프레임워크를 사용하면 객체의 생성, 필드값의 조회와 조작, static 메서드 호출 및 초기화 등의 다양한 작업에 부가기능을 적용할 수 있다.

Spring AOP의 구현 방식

KH정보교육원

XML 기반의 aop 네임스페이스를 통한 AOP 구현

부가기능을 제공하는 Advice 클래스를 작성한다.
XML 설정 파일에 <aop:config>를 이용해서 아스펙트를 설정한다.
(즉, 어드바이스와 포인트컷을 설정함)

@Aspect 어노테이션을 이용한 AOP 구현

@Aspect 어노테이션을 이용해서 부가기능을 제공하는 Aspect 클래스를 작성한다. 이때 Aspect 클래스는 어드바이스를 구현하는 메서드와 포인트컷을 포함한다.
XML 설정 파일에 <aop:aspectj-autoproxy />를 설정한다.

KH정보교육원

Advice 작성하기

Advice 작성하기

KH정보교육원

Advice의 종류

Before Advice	타겟의 메소드가 실행되기 이전(before) 시점에 처리해야 할 필요가 있는 부가기능을 정의한다 -> Joinpoint 앞에서 실행되는 Advice
Around Advice	타겟의 메소드가 호출되기 이전(before) 시점과 이후(after) 시점에 모두 처리해야 할 필요가 있는 부가기능을 정의한다. -> Joinpoint 앞과 뒤에서 실행되는 Advice
After Returning Advice	타겟의 메서드가 정상적으로 실행된 이후(after) 시점에 처리해야 할 필요가 있는 부가기능을 정의한다. -> Jointpoint 메서드 호출이 정상적으로 종료된 뒤에 실행되는 Advice
After Throwing Advice	타겟의 메서드가 예외를 발생된 이후(after) 시점에 처리해야 할 필요가 있는 부가기능을 정의한다. -> 예외가 던져질 때 실행되는 Advice

Advice 작성하기

KH정보교육원

JoinPoint Interface

JoinPoint는 Spring AOP 혹은 AspectJ에서 AOP의 부가 기능을 지닌 코드가 적용되는 지점을 뜻하며, 모든 어드바이스는 org.aspectj.lang.JoinPoint 타입의 파라미터를 어드바이스 메소드에 첫 번째 매개변수로 선언해야 한다.
단, Around 어드바이스는 JoinPoint의 하위 클래스인 ProceedingJoinPoint 타입의 파라미터를 필수적으로 선언해야 한다.

Advice 작성하기

KH정보교육원

JoinPoint Interface 메소드

getArgs()	메소드의 매개 변수를 반환한다
getThis()	현재 사용 중인 프록시 객체를 반환한다
getTarget()	대상 객체를 반환한다
getSignature()	대상 객체 메소드의 설명(메소드 명, 리턴 타입 등)을 반환한다
toString()	대상 객체 메소드의 정보를 출력한다

※ JoinPoint에 대한 상세한 내용은 아래 링크를 참조하자.
https://www.eclipse.org/aspectj/doc/next/runtime-api/index.html

Advice 작성하기

KH정보교육원

Advice 예시

```
import org.aspectj.lang.ProceedingJoinPoint;
import org.springframework.util.StopWatch;

public class AroundLog {
   public Object aroundLog(ProceedingJoinPoint pp) throws Throwable{
      //사전, 사후 처리를 모두 해결하고자 할 때 사용하는 어드바이스
      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;
   }
}
```

KH정보교육원

Spring AOP 사용하기

Spring AOP 사용하기

KH정보교육원

maven : pom.xml에 라이브러리 추가

AOP 사용을 위한 라이브러리를 pom.xml에 등록한다.

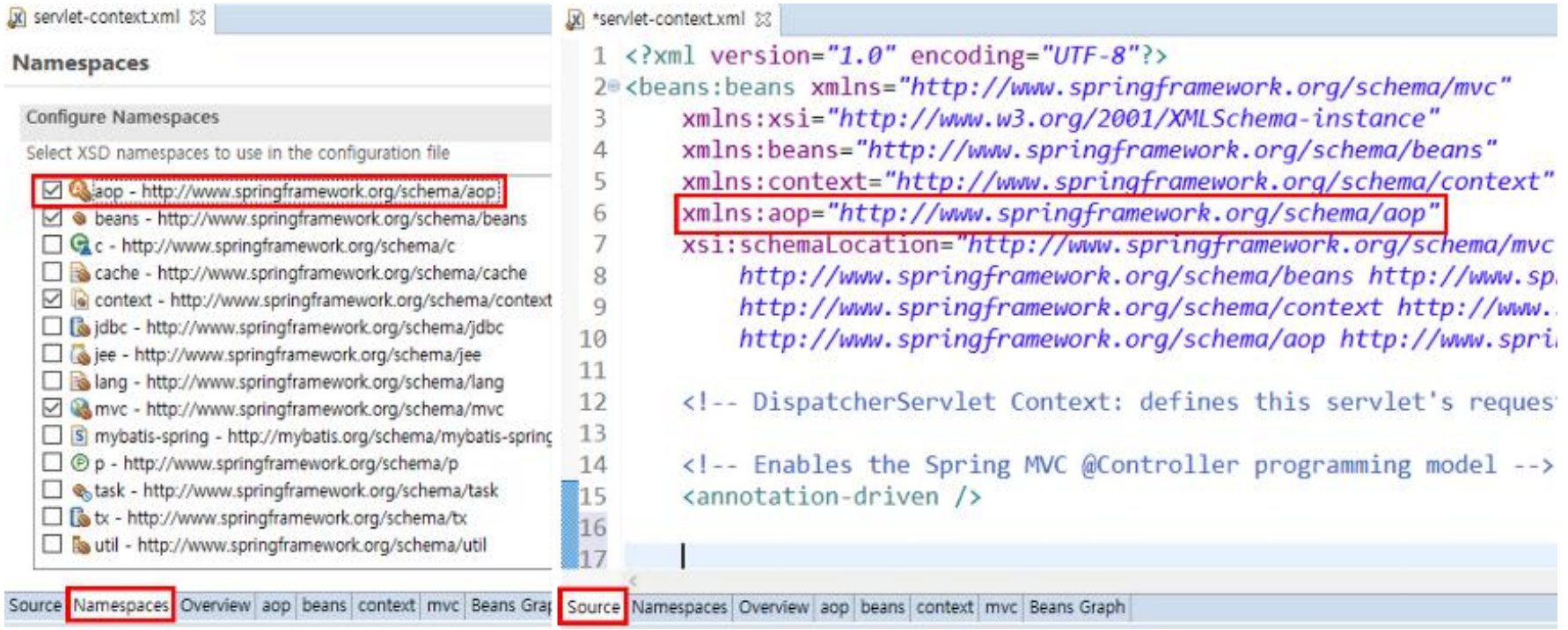
```
<!-- AspectJ RunTime -->
<dependency>
   <groupId>org.aspectj</groupId>
   <artifactId>aspectjrt</artifactId>
   <version>${org.aspectj-version}</version>
</dependency>
<!-- AspectJ Weaver -->
<!-- weaver는 AOP에서 advice를 핵심 기능에 적용하는 설정 파일이다 -->
<dependency>
   <groupId>org.aspectj</groupId>
   <artifactId>aspectjweaver</artifactId>
   <version>${org.aspectj-version}</version>
</dependency>
<!-- AOP Proxy Library -->
<dependency>
   <groupId>cglib</groupId>
   <artifactId>cglib</artifactId>
   <version>2.2</version>
</dependency>
```

Spring AOP 사용하기

KH정보교육원

servlet-context.xml 에 xmlns : aop 등록

스프링 설정 파일(servlet-context.xml)을 클릭 후 하단의 Namespaces 탭을 클릭하여 aop을 선택하여 해당 요소를 사용하기 위한 설정을 추가



Spring AOP 사용하기

KH정보교육원

XML : <aop: ??? >

<aop:config> : AOP 설정 정보임을 나타낸다.

<aop:aspect> : 아스펙트를 설정한다.

<aop:around pointcut="execution()">

: Around 어드바이스와 포인트컷을 설정한다

```
<bean id="testAop"
     class="org.kh.firstSpring.service.common.AroundLog"/>

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

Spring AOP 사용하기

KH정보교육원

XML : <aop: ??? >

<aop:aspect> 태그의 ref 속성은 아스팩트로서 기능을 제공할 Bean을 설정할 때 사용함

<aop:around> 태그의 pointcut 속성의 execution 지시자(designator)는 어드바이스를 적용할 패키지, 클래스, 메소드를 표현할 때 사용됨

```
<bean id="testAop"
      class="org.kh.firstSpring.service.common.AroundLog"/>

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

Spring AOP 사용하기

KH정보교육원

Advice를 정의하는 태그

<aop:before>	- 메소드 실행 전에 적용되는 어드바이스를 정의
<aop:around>	- 메소드 호출 이전, 이후, 예외 발생 등 모든 시점에 적용 가능한 어드바이스를 정의
<aop:after>	- 메소드가 정상적으로 실행되는지 또는 예외를 발생시키는지 여부에 상관없는 어드바이스를 정의
<aop:after-returning>	- 메소드가 정상적으로 실행된 후에 적용되는 어드바이스를 정의
<aop:after-throwing>	- 메소드가 예외를 발생시킬 때 적용되는 어드바이스를 정의 - **try-catch 블록에서 catch 블록과 비슷함**

Spring AOP 사용하기

KH정보교육원

Annotation : @Aspect

1. 클래스 선언부에 @Aspect 어노테이션을 정의한다.

2. 이 클래스를 아스펙트로 사용하려면 Bean으로 등록해야 하므로 @Component 어노테이션도 함께 정의한다.

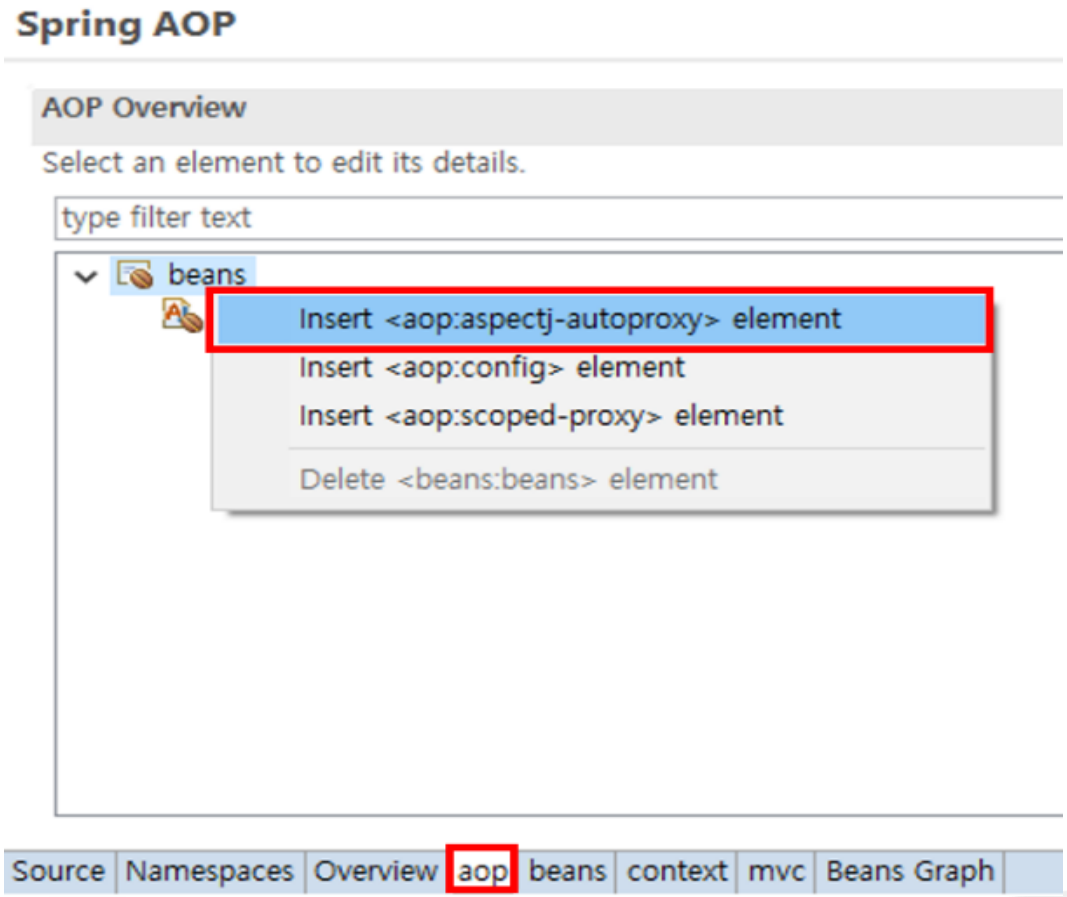
```
@Component
@Aspect
public class AroundLog {
```

Spring AOP 사용하기

KH정보교육원

Annotation : <aop:aspect-autoproxy>

3. aop 탭으로 가서 beans를 우클릭하고 'Insert <aop:aspect-autoproxy> element'를 클릭하여 <aop:aspect-autoproxy> 요소를 추가한다



Spring AOP 사용하기

KH정보교육원

Advice를 정의하는 어노테이션

@Before(“pointcut”)	- 타겟 객체의 메소드가 실행되기 전에 호출되는 어드바이스 - JoinPoint를 통해 파라미터 정보를 참조할 수 있다.
@After(“pointcut”)	- 타겟 객체의 메소드가 정상 종료됐을 때와 예외가 발생했을 때 모두 호출되는 어드바이스로, 반환 값을 받을 수 없다
@Around(“pointcut”)	- 타겟 객체의 메소드 호출 전과 후에 실행 될 코드를 구현할 어드바이스.
@AfterReturning(pointcut=“”, returning=“”)	- 타겟 객체의 메서드가 정상적으로 실행을 마친 후에 호출되는 어드바이스 - 리턴값을 참조할 때는 returning 속성에 리턴값을 저장할 변수 이름을 지정해야 한다.
@AfterThrowing(pointcut=“”, throwing=“”)	- 타겟 객체의 메소드 예외가 발생하면 호출되는 어드바이스 - 발생된 예외를 참조할 때는 throwing 속성에 발생한 예외를 저장할 변수 이름을 지정해야 한다.

Spring AOP 사용하기

KH정보교육원

Annotation Advice 예시

```
@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;
}
```

KH정보교육원

@Pointcut 어노테이션 표현식

Poincut 어노테이션을 사용할 때 execution 속성에 표현식을 설정해 주어야 하는데, 다음과 같은 형식으로 지정해 주어야 한다.

Pointcut 표현식 / execution(* com.kh.biz. .*Impl.*(..))			
*	com.kh.biz.	.*Impl.	*(..)
리턴 타입	패키지 경로	클래스명	메소드명 및 매개변수

※ 예시의 표현식은 리턴 타입과 매개변수를 무시하고, com.kh.biz. 패키지로 시작하는 클래스 중 이름이 Impl로 끝나는 클래스의 모든 메소드를 뜻한다.

@Pointcut 어노테이션 주요 표현식

형식	설명	
리턴타입	*	모든 리턴 타입 허용
	void	리턴타입이 void인 메소드만 선택
	!void	리턴타입이 void가 아닌 메소드 선택
패키지	com.kh.test	정확하게 com.kh.test 패키지만 선택
	com.kh.test..	com.kh.test 로 시작하는 모든 패키지 선택
	com.kh.test..*impl	com.kh.test 로 시작해서 마지막 패키지 이름이 impl로 끝나는 패키지 선택
클래스	BoardServiceImpl	정확하게 BoardServiceImpl 클래스만 선택
	*Impl	클래스 이름이 Impl로 끝나는 클래스만 선택
	BoardService+	클래스 이름 뒤에 '+'가 붙으면 해당 클래스로부터 파생된 모든 자식 클래스 선택 인터페이스 뒤에 '+'가 붙으면 해당 인터페이스를 구현한 모든 클래스 선택

KH정보교육원

@Pointcut 어노테이션 주요 표현식

형식	설명	
메소드	*(..)	가장 기본 설정으로 모든 메소드 선택
	get*(..)	메소드 이름이 get으로 시작하는 모든 메소드 선택
매개변수	(..)	가장 기본 설정으로 '..'은 매개변수의 개수와 타입에 제약이 없음을 의미
	(*)	반드시 1개의 매개변수를 가진 메소드 선택
	(com.kh.user.UserVO)	매개변수로 UserVO를 가지는 메소드만 선택, 이때 클래스의 패키지 경로는 반드시 포함
	(!com.kh.user.UserVO)	매개변수로 UserVO를 가지지 않는 메소드
	(Integer, ..)	한 개이상의 매개변수를 가지되, 첫번째 매개변수의 타입이 Integer 인 메소드만 선택
	(Integer, *)	반드시 두 개의 매개변수를 가지되, 첫번째 매개변수의 타입이 Integer 인 메소드만 선택