

Spring AOP

Spring AOP 개요 (1/2)

- 핵심 관심 사항(core concern) 과 공통 관심 사항 (cross-cutting concern)
- 기존 OOP 에서는 공통관심사항을 여러 모듈에서 적용하는데 중복된 코드를 양산과 같은 한계가 존재 - 이를 극복하기 위해 AOP 가 등장
- Aspect Oriented Programming은 문제를 해결하기 위한 핵심 관심 사항과 전체에 적용되는 공통관심 사항을 기준으로 프로그래밍함으로써 공통 모듈을 손쉽게 적용할 수 있게 해준다.

Spring AOP 개요 (2/2)

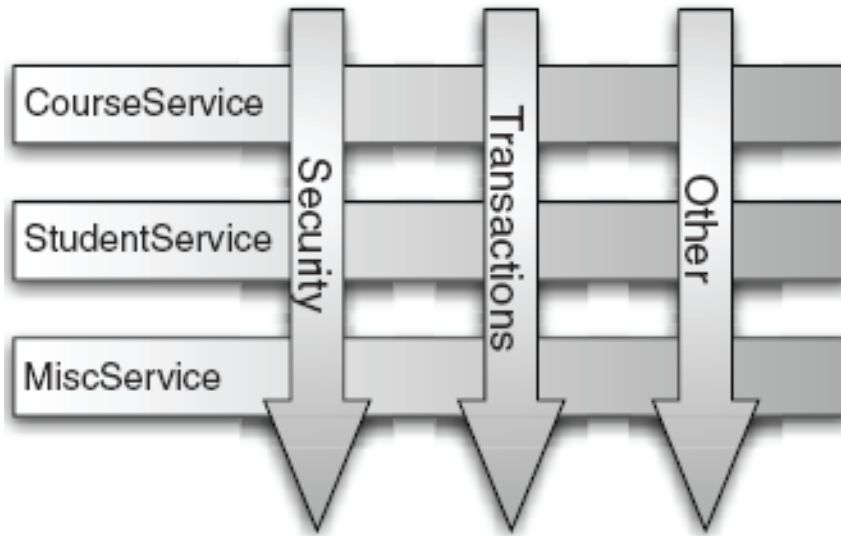


Figure 4.1
Aspects modularize cross-cutting concerns, applying logic that spans multiple application objects.

핵심관심사항 : CourseService, StudentService, MiscService
공통관심사항 : Security, Transactions, Other

- 핵심관심사항에 공통관심사항을 어떻게 적용시킬 것인가...

Spring AOP 용어

- Aspect - 여러 객체에서 공통으로 적용되는 공통 관심 사항(ex:트랜잭션, 로깅, 보안)
- JoinPoint – Aspect가 적용 될 수 있는 지점(ex:메소드, 필드)
- Pointcut – 공통 관심 사항이 적용 될 Joinpoint
- Advice – 어느 시점(ex: 메소드 수행 전/후, 예외발생 후 등)에 어떤 공통 관심 기능(Aspect)을 적용할 지 정의한 것.
- Weaving– 어떤 Advice를 어떤 Pointcut(핵심사항)에 적용시킬 것인지에 대한 설정 (Advisor)

Spring에서 AOP 구현 방법

- AOP 구현의 세가지 방법
 - POJO Class를 이용한 AOP구현
 - 스프링 API를 이용한 AOP구현
 - 어노테이션(Annotation) 을 이용한 AOP 구현

POJO 기반 AOP구현

- XML 스키마 확장기법을 통해 설정파일을 작성한다.
- POJO 기반 Advice 클래스 작성

POJO 기반 AOP구현 - 설정파일 작성 (1/5)

- XML 스키마를 이용한 AOP 설정
 - aop 네임스페이스와 XML 스키마 추가

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:p="http://www.springframework.org/schema/p"
  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-2.5.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

</beans>
```

POJO 기반 AOP구현 - 설정파일 작성 (2/5)

- XML 스키마를 이용한 AOP 설정

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:p="http://www.springframework.org/schema/p"
  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-2.5.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

  <bean id="writelog" class="org.kosta.spring.LogAspect"/>

  <aop:config>
    <aop:pointcut id="publicmethod" expression="execution(public * org.kosta.spring..*(..))"/>
    <aop:aspect id="loggingAspect" ref="writelog">
      <aop:around pointcut-ref="publicmethod" method="logging"/>
    </aop:aspect>
  </aop:config>

  <bean id="targetclass" class="org.kosta.spring.TargetClass"/>

</beans>
```


POJO 기반 AOP구현 - 설정파일 작성 (3/5)

- AOP 설정 태그

1. <aop:config> : aop 설정의 root 태그. – weaving들의 묶음.
2. <aop:aspect> : Aspect 설정 – 하나의 weaving에 대한 설정
3. <aop:pointcut> : Pointcut 설정
4. Advice 설정 태그들
 - A. <aop:before> - 메소드 실행 전 실행 될 Advice
 - B. <aop:after-returning> - 메소드 정상 실행 후 실행 될 Advice
 - C. <aop:after-throwing> - 메소드에서 예외 발생시 실행 될 Advice
 - D. <aop:after> - 메소드 정상 또는 예외 발생 상관없이 실행 될 Advice – finally
 - E. <aop:around> - 모든 시점에서 적용시킬 수 있는 Advice 구현

POJO 기반 AOP구현 - <aop:aspect> (4/5)

- 한 개의 Aspect (공통 관심기능)을 설정
- ref 속성을 통해 공통기능을 가지고 있는 bean을 연결 한다.
- id는 이 태그의 식별자를 설정
- 자식 태그로 <aop:pointcut> advice관련 태그가 올 수 있다.

```
<aop:config>
  <aop:aspect id="loggingAspect" ref="writelog">
    <aop:pointcut id="publicmethod" expression="execution(public * public * org.myspring..*.* (..))"/>
    <aop:around pointcut-ref="publicmethod" method="logging"/>
  </aop:aspect>
</aop:config>
```

POJO 기반 AOP구현 - <aop:pointcut> (5/5)

- Pointcut(공통기능이 적용될 곳)을 지정하는 태그
- <aop:config>나 <aop:aspect>의 자식 태그
 - <aop:config> 전역적으로 사용
 - <aop:aspect> 내부에서 사용
- AspectJ 표현식을 통해 pointcut 지정
- 속성 :
 - id : 식별자로 advice 태그에서 사용됨
 - expression : pointcut 지정

<aop:pointcut id="publicmethod" expression="execution(public * org.myspring..*.*(..))"/>

```
<aop:config>
  <aop:aspect id="loggingAspect" ref="writelog">
    <aop:pointcut id="publicmethod" expression="execution(public * org.myspring..*.*(..))"/>
    <aop:around pointcut-ref="publicmethod" method="logging"/>
  </aop:aspect>
</aop:config>
```

POJO 기반 AOP구현 - AspectJ 표현식 (1/3)

- AspectJ에서 지원하는 패턴 표현식
- 스프링은 메서드 호출관련 명시자만 지원

명시자(제한자패턴? 리턴타입패턴 패키지패턴?이름패턴(파라미터패턴))
-?는 생략가능

- 명시자
 - execution : 실행시킬 메소드 패턴을 직접 입력하는 경우
 - within : 메소드가 아닌 특정 타입에 속하는 메서드들을 설정할 경우
 - bean : 2.5버전에 추가됨. 설정파일에 지정된 빈의 이름(name속성)을 이용해 pointcut설정

POJO 기반 AOP구현 - AspectJ 표현식 (2/3)

- 표현

명시자(수식어패턴? 리턴타입패턴 패키지패턴? 클래스이름패턴.메소드이름패턴(파라미터패턴))

-?는 생략가능

예) **execution(public * abc.def.*Service.set*(..))**

- 수식어 패턴에는 public, protected 또는 생략한다.

- * : 1개의 모든 값을 표현

- argument에서 쓰인 경우 : 1개의 argument
 - package에 쓰인 경우 : 1개의 하위 package

- .. : 0개 이상

- argument에서 쓰인 경우 : 0개 이상의 argument
 - package에 쓰인 경우 : 0개의 이상의 하위 package

- 위 예 설명

적용 하려는 메소드들의 패턴은 public 제한자를 가지며 리턴 타입에는 모든 타입이 다 올 수 있다. 이름은 abc.def 패키지와 그 하위 패키지에 있는 모든 클래스 중 Service로 끝나는 클래스들에서 set으로 시작하는 메소드이며 argument는 0개 이상 오며 타입은 상관 없다.

POJO 기반 AOP구현 - AspectJ 표현식 (3/3)

- 예

```
execution(* test.spring.*.*())  
execution(public * test.spring..*.*())  
execution(public * test.*.*.get*(*))  
execution(String test.spring.MemberService.registMember(..))  
execution(* test.spring..*Service.regist*(..))  
execution(public * test.spring..*Service.regist*(String, ..))  
  
within(test.spring.service.MemberService)  
within(test.spring..MemberService)  
within(test.spring.aop..*)  
  
bean(memberService)  
bean(*Service)
```

POJO 기반 AOP구현 - Advice 작성

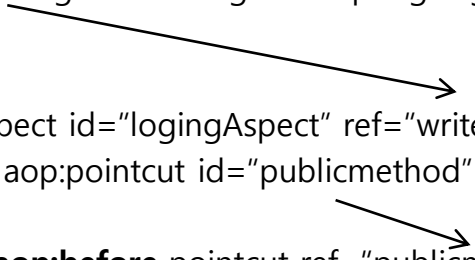
- POJO 기반 Aspect클래스 작성
 - 설정 파일의 advice 관련 태그에 맞게 작성한다.
 - <bean>으로 등록 하며 <aop:aspect> 의 ref 속성으로 참조한다.
 - 공통 기능 메소드 : advice 관련 태그들의 method 속성의 값이 메소드의 이름이 된다.

POJO 기반 AOP구현 - Advice 정의 관련 태그

- 속성

- pointcut-ref : <aop:pointcut>태그의 id명을 넣어 pointcut지정
- pointcut : 직접 pointcut을 설정 한다.
- method : Aspect bean에서 호출할 메소드명 지정

```
<bean id="writelog" class="org.kosta.spring.LogAspect"/>
<aop:config>
  <aop:aspect id="loggingAspect" ref="writelog">
    <aop:pointcut id="publicmethod" expression="execution(public * org.my.spring..*(..))"/>
    <aop:before pointcut-ref="publicmethod" method="logging"/>
  </aop:aspect>
</aop:config>
```



POJO 기반 AOP구현 - Aspect클래스 작성 (1/4)

- POJO 기반의 클래스로 작성한다.
 - 클래스 명이나 메서드 명에 대한 제한은 없다.
 - 메소드 구문은 호출되는 시점에 따라 달라 질 수 있다.
 - 메소드의 이름은 advice 태그(<aop:before/>)에서 method 속성의 값이 메소드 명이 된다.
- before 메소드
 - 대상 객체의 메소드가 실행되기 전에 실행됨
 - return type : void
 - argument : 없거나 JoinPoint 객체를 받는다.
 - ex)

```
public void beforeLogging(JoinPoint jp) {  
  
}
```

POJO 기반 AOP구현 - Aspect클래스 작성 (2/4)

- After Returning Advice

- 대상객체의 메소드 실행이 정상적으로 끝난 뒤 실행됨
- return type : void
- argument :
 - 없거나 JoinPoint 객체를 받는다. JoinPoint는 항상 첫 argument로 사용된다.
 - 대상 메소드에서 리턴되는 값을 argument로 받을 수 있다.

```
<aop:after-returning pointcut-ref="publicmethod" method="returnLogging" returning="retValue"/>
```

```
public void returnLogging(Object retValue) {
```

```
    //대상객체에서 리턴되는 값을 받을 수는 있지만 수정할 수는 없다.
```

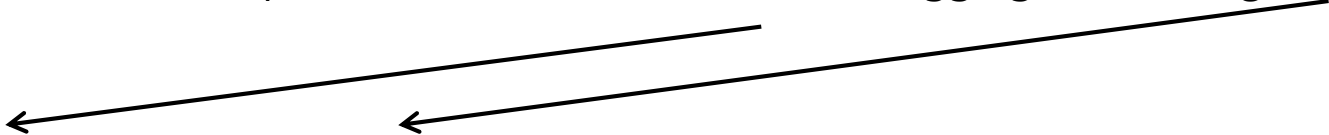
```
}
```

POJO 기반 AOP구현 - Aspect클래스 작성 (3/4)

- After Throwing Advice
 - 대상객체의 메소드 실행 중 예외가 발생한 경우 실행됨
 - return type : void
 - argument :
 - 없거나 JoinPoint 객체를 받는다. JoinPoint는 항상 첫 argument로 사용된다.
 - 대상 메소드에서 전달되는 예외객체를 argument로 받을 수 있다.

```
<aop:after-throwing pointcut-ref="publicmethod" method="returnLogging" throwing="ex"/>>
```

```
public void returnLogging(MyException ex){  
    //대상객체에서 리턴되는 값을 받을 수는 있지만 수정할 수는 없다.  
}
```



POJO 기반 AOP구현 - Aspect클래스 작성 (4/4)

- Around Advice
 - 위의 네 가지 Advice를 다 구현 할 수 있는 Advice.
 - return type : Object
 - argument
 - org.aspectj.lang.ProceedingJoinPoint 를 반드시 첫 argument로 지정한다.

```
<aop:around pointcut-ref="publicmethod" method="returnLogging" />

public Object returnLogging(ProceedingJoinPoint joinPoint) throws Throwable{
    //대상 객체의 메소드 호출 전 해야 할 전 처리 코드
    try{
        Object retValue = joinPoint.proceed(); //대상객체의 메소드 호출
        //대상 객체 처리 이후 해야 할 후처리 코드
        return retValue; //호출 한 곳으로 리턴 값 넘긴다. - 넘기기 전 수정 가능
    }catch(Throwable e){
        throw e; //예외 처리
    }
}
```

JoinPoint

- 대상객체에 대한 정보를 가지고 있는 객체로 Spring container로 부터 받는다.
- org.aspectj.lang 패키지에 있음
- 받듯이 Aspect 메소드의 첫 argument로 와야 한다.
- 메소드들

Object getTarget() : 대상객체를 리턴

Object[] getArgs() : 파라미터로 넘겨진 값들을 배열로 리턴. 넘어온 값이 없으면 빈 배열객체가 return 됨.

Signature getSignature () : 호출 되는 메소드의 정보

- Signature : 호출 되는 메소드에 대한 정보를 가진 객체

String getName() : 메소드 명

String toLongString() : 메서드 전체 syntax를 리턴

String toShorString() : 메소드를 축약해서 return – 기본은 메소드 이름만 리턴

@Aspect 어노테이션을 이용한 AOP

- @Aspect 어노테이션을 이용하여 Aspect 클래스에 직접 Advice 및 Pointcut등을 직접 설정
- 설정파일에 <aop:aspectj-autoproxy/> 를 추가 필요
- Aspect class를 <bean>으로 등록
- 어노테이션(Annotation)
 - @Aspect : Aspect 클래스 선언
 - @Before("pointcut")
 - @AfterReturning(pointcut="", returning="")
 - @AfterThrowing(pointcut="", throwing="")
 - @After("pointcut")
 - @Around("pointcut")
- Around를 제외한 나머지 메소드들은 첫 argument로 JoinPoint를 가질 수 있다.
- Around 메소드는 argument로 ProceedingJoinPoint를 가질 수 있다.