# 애스펙트(관점) 지향 프로그래밍

## AOP란 무엇인가?!

- 한 애플리케이션에서 여러부분에 걸쳐 있는 기능을 가리켜 횡단 관심사(cross-cutting concerns) 라 한다.

- 횡단 관심사란하나의 애플리케이션이

주요관심사에 대해서 보조기능을 모듈화 시킨것이라 볼 수 있다.

- 횡단관심사의 주요 목적은 모듈화에 있다. 횡단관심사를 하나의 어플리케이션의 여러부분에 영향을 주는 기능이라고 정리할 수있는데 이 부분을 하나로 모듈화시켜 관리 할 수 있는 것이 AOP인 것이다.

- 애스펙트 지향 프로그래밍(AOP)은 바로 이러한 횡단관심사의 분리를 위한 것이다.

- 공통적인 기능(횡단관심사)를 재사용 하기 위해선 객체지향에서 상속(inheritance)나 위임(delegation)을 사용하는데 상속은 의존성이 높아지고 위임은 코드의 복잡성이 증가했었다. 하지만 AOP는 이런 부분을 해소시켜준다.

- AOP는 기본적으로 공통기능을 한 곳에 정의하는 것은 동일하다. 하지만 이 기능을 어디에 어떻게 적용할지를 “선언적 정의”할 수 있다는 것이 강점이다.

AOP에서는 횡단관심사를 Aspect(애스펙트)라는 특별한 클래스로 모듈화 한다.

AOP가 다른 기법과 차별화되는 장점은 두가지이다. 첫 번째 장점은 여타 서비스모듈이 자신의 주요관심사항에 대한 코드만 포함하고 그 외의 관심사항은 모드 애스펙트로 옮겨지므로 코드가 깔끔해진다. 두번째 장점은 전체코드기반에 흩어져 있는 관심사항이 하나의 장소로 응집된다

### AOP 용어 정리

AOP의 용어의 종류는 어드바이스, 포인트 컷, 조인포인트 세개 이다.

###### 1. 어드바이스 (ADVICE)

- 애스펙트는 자신이 무엇을 해야하는지를 알고 있어야 한다. 애스펙트가 해야할 작업을 **어드바이스** 라고 한다.

- ‘무엇**’**을 ‘언제’할 지 정의한다. Before , After , after-returning , after-throwing , around 등의 수행동작 시점을 선정할 수 있다.

###### 2. 조인포인트(point cut)

- 애플리케이션에서 어드바이스를 적용할 수 있는 곳을 말한다. 이와 같이 어드바이스를 적용할 수 있는 곳을 “조인포인트”라 한다. 애플리케이션 조인포인트 지점에 애스펙트를 끼워 넣을 수 있는 지점을 말한다. 이러한 지점에는 메소드 호출 지점 **,** 예외 발생 , 필드 값 수정 등이 있다.

###### 3. 포인트 컷(point cut)

- 포인트 컷은 애스펙트가 어드바이스할 조인포인트의 영역을 좁히는 일을 한다.

- 포인트 컷은 “어디서” 할 지를 정의한다.

- 각 포인트 컷은 어드바이스가 위빙(weaving)돼야 하는 하나 이상의 조인포인트를 정의한다.

- 클래스나 메소드명을 직접 사용하여 지정할 수도 잇고 정규식으로 정의할 수도 있다.

###### 4. 애스펙트(Aspect, 관점)

- 애스펙트는 어드바이스와 포인트 컷을 합친(merge) 것이다.

###### 5. 인트로 덕션

- 인트로덕션은 클래스의 코드 변경 없이 메소드나 멤버변수를 추가하는 기능이다.

###### 6. 위빙(weaving)

- 위빙은 타깃 객체(어드바이스 대상)에 애스펙트를 적용해서 새로운 프록시 객체를 생성하는 절차다. 애스펙트는 타깃 객체의 조인포인트로 위빙된다.

위빙은 대상객체의 생애 중 다음과 같은 몇 가지 시점에서 수행될 수 있다.

1. 컴파일 시간(compile time) – 타깃 클래스가 컴파일 될 때 애스펙트가 위빙(프록시 객체 생성)된다.

2. 클래스 로드 시간(classload time) – 클래스가 jvm에 로드 될 때 애스펙트가 위빙된다. (aspectJ 가 필요)

3. 실행 시간(runtime time) – 애플리케이션 실행 중에 애스펙트가 위빙된다.

### 스프링의 AOP 지원

AOP 프레임워크는 프레임워크마다 제공하는 조인포인트 모델 수준이 다르다.

스프링 프레임워크는 메소드 조인포인트만 지원한다.

AspectJ나 JBoss 같은 AOP프레임워크는 필드나 생성자 조인포인트까지 제공한다.

스프링 AOP로도 대부분의 대부분 충족할 수 있으나 그렇지 못한 부분에 대해서 AspectJ 를 이용하여 보충하면 된다.

가장 널리 사용하는 AOP프레임워크는 아래와 같다. (스프링은 스프링AOP만 지원)

* AspectJ (<http://www.eclipse.org/aspectJ>)
* JBoss AOP (<http://www.jboss.org/jbossaop>)
* 스프링 AOP (<http://www.springframework.org>)

스프링은 네가지 형태의 AOP를 지원한다.

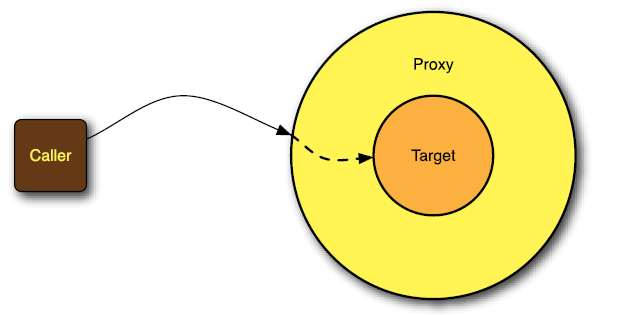
* 고전적인(classic) 스프링 프록시 기반 AOP
* @AspectJ 애너테이션 기반 애스펙트
* Pure-POJO 애스팩트
* AspectJ 애스펙트에 빈 주입 (스프링의 모든 버전에서 지원 )

스프링의 어드바이스는 자바로 작성한다.

- 스프링에서 생성하는 advice는 자바클래스로 작성한다.

스프링 애스펙트는 실행시간(Runtime)에 만들어 진다.

- 스프링에서 빈을 감싸는 프록시 객체를 실행시간(Runtime)에 생성함으로써 애스펙트가 스프링과니 빈에 위빙된다.

- 프록시는 caller 의 호출을 가로채고 나서 실행되고 애스펙트 로직을 수행하고 나서 타깃 빈의 메소드가 호출된다. AspectJ

- 프록시는 타겟으로 위장하고 있다.

## 포인트컷을 이용한 조인포인트 선택

스프링 AOP에서 포인트컷은 “AspectJ의 포인트컷 표현식 언어를 이용해 정의” 된다.

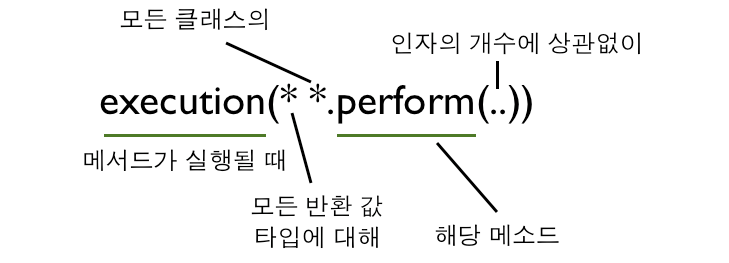
스프링은 AspectJ에서 사용할 수 있는 포인트컷 지정자에 속하는 것만 지원한다

p.121 표 4.1참고

AspectJ의 다른 지정자를 사용하면 IllegalArgumentException 이 던져진다.

### 포인트컷 작성

포인트컷 표현식 예제: **execution \* com.springinaction.springidol.Istrument.play( . . )**



설명

execution : 메소드의 실행 시작

\* : 모든 타입 리턴

com.springinaction.springidol.Istrument. : 메소드가 속하는 타입

play() : 메소드

( . . ) : 모든 인자 받기

\* com.springinaction.springidol.Istrument.play( . . ) : 메소드 명세

### 스프링의 Bean() 지정자 사용

execution (\* com.springinaction.springidol.Instrument.play()) and bean(eddie)

여기서는 애스펙트 어드바이스를 Instrument 에 있는 play() 메소드의 실행에 적용한다. 하지만 ID가 eddie인 빈으로 제한한다.

Bean()은 인자로 빈ID나 이름을 바고 특정 빈에 대한 포인트컷의 영향을 제한한다.

## XML에 애스펙트 선언

스프링의aop설정 엘리먼트는 POJO 기반의 애스펙트 선언을 간소화한다.

|  |  |
| --- | --- |
| AOP 설정 엘리먼트 | 목적 |
| <aop:advisor> | AOP 어드바이저를 정의 한다 |
| <aop:after> | After어드바이스를 정의한다.(이 어드바이스는 메서드의 예외발생 여부와 관계 없다.) |
| <aop:after-returning> | After-returning 어드바이스를 정의한다. |
| <aop:after-throwing> | After-throwing 어드바이스를 정의한다. |
| <aop:around> | Around 어드바이스를 정의한다. |
| <aop:aspect> | 애스펙트를 정의한다. |
| <aop:aspectj-autoproxy> | @AspectJ를 이용하여 애너테이션 기반 애스펙트를 가능 하게 한다. |
| <aop:before> | Before 어드바이스를 정의한다. |
| <aop:config> | 최상위 AOP엘리번트. 대부분의 <aop:\*> 엘리먼트는 이 <aop:config>안에 포함되야 한다. |
| <aop:declare-parents> | 투명하게 구현된 어드바이스 대상 객체에 추가적인 인터페이스를 도입한다. |
| <aop:pointcut> | 포인트컷을 정의한다. |

<스프링 AOP설명하기 위한 클래스 Audience 클래스 생성해 보자 >

Public class Audience {

Public void takeSeats() { …. } // 공연 전

Public void turnOffCellPhones(){ …. } //공연 전

Public void applaud() { …. } // 공연 후

Public void demandRefund() { …. } // 공연 실패 후

}

### Before , After 어드바이스 선언

스프링의 AOP 설정 엘리먼트를 이용하여 audience 빈을 애스펙트로 변환할 수 있다.

<aop:config>

<aop:aspect ref=”audience” >//ref=”빈ID”

<aop:before pointcut=”execution(\* com.springinaction.springido.Performer.perform(..))”

method=”takeSeats” /> //공연 전

<aop:before pointcut=”execution(\* com.springinaction.springido.Performer.perform(..))”

method=”turnOffCellPhones” /> //공연 전

<aop:after-returning pointcut=”execution(\* com.springinaction.springido.Performer.perform(..))”

method=”applaud” /> //공연 후

<aop:after-throwing pointcut=”execution(\* com.springinaction.springido.Performer.perform(..))”

method=”demandRefund” /> //공연 실패 후

</aop:aspect>

</aop:config>

AOP 설정 엘리먼트는 <aop:config>내에 사용되야 하는데 이 규칙에 몇가지 예외는 있지만 빈을 애스펙트로 선언할 때는 항상 <aop:config>엘리먼트로 시작한다.

위 코드는 하나의 애스펙트만을 선언하였으며, 네개의 어드바이스를 갖고 있다.

<aop:before> 엘리먼트는 audience빈의 takeSeats()와 turnOffCellPhones()메소드를 해당 포인트컷에 일치하는 메소드 실행 이전에 호출하는 어드바이스

<aop:after-returning> 엘리먼트는 포인트컷 다음에 applaud90메소드를 호출하는 어드바이스

<aop:after-throwing> 은 예외발생시에 demandRefund()를 호출하는 어드바이스를 정의 한다.

모든 어드바이스 엘리먼트에서 pointcut 애트리뷰트는 어드바이스 적용대상이 어딘지를 나타내는 포인트 컷을 정의한다.

Pointcut 애트리뷰트에 설정한 값은 AspectJ의 포인트컷 표현식으로 정의한다.

포인트컷을 변경한다면 네 군데 모두를 같이 변경해야 하는데 이는 동일한 포인트컷이 같은곳에 적요되어 있기 때문이다. 이는 DRY(Don’t Repeat Yoursele) 에 위반한다.

어드바이스가 적용된 로직은 아래와 같다.

try {

audience.takeSeats();

audience.turnOffCellPhones();

perform(); //비즈니스 로직

audience.applaud()

}catch(Exception e){

audience.demandRefund();

}

포인트컷 정의 중복을 피하려면 포인트컷에 이름을 부여할 수 잇는<aop:pointcut>엘리먼트가 좋은 선택이다.

### Around 어드바이스 선언

### 어드바이스에 파라미터 전달

### 애스펙트를 이용한 새로운 기능 도입

인트로덕션(introduction)으로 알려진 AOP 개념을 이용하면, 애스펙트는 스픵 빈에 새로운 메소드를 추가 할 수 있다.

--위의 세 문제는 추후에 개념을 잡기로 해야겠다.

## 애스펙트 애너테이션

- AspectJ 5에 도입된 주요 기능은 애너테이션을 사용하여 애스펙트를 만들 수 있는 기능이다.

이 새로운 기능을 보통 @AspectJ 라고 한다.

@AspectJ 애너테이션을 이용하면 audience 클래스를 잡다한 추가 클래스나 빈 설정 없이도 애스펙트로 만들 수 있다.

Public class audience {

@Pointcut(“execution(\* com.springinaction.springidol.Perfomer.perform(..))”)

Public void performance() { …. } //포인트컷 정의

@Before(“performance()”)

Public void takeSeats(){ …… }//공연 전

}

위 Audience 클래스에는 @AspectJ 애너테이션이 붙어 있어서 Audience 클래스는 더 이상 POJO가 아닌 애스펙트임을 나타낸다. (어차피 어드바이스는 하나의 메소드에만 적용한다. )

포인트컷의 이름은 애너테이션이 적용된 메소드의 이름으로 결정된다.

포인트 컷을 위한 메소드는 단순한 표식이라서 @Pointcut 메소드의 구현부는 @AspectJ에서는 아무상관 없으므로 그냥 비워두면 된다. (애너테이션 포인트컷은 장소나 이름을 제공하는것만이 목적이다. )

Performace()포인트 컷의 이름은 다른 어드바이스 애너테이션에 인자로 전달 되는데 이를 통해 어드바이스가 어디에 적용돼야 하는지를 알 수 있다.

마지막으로 스프링이 audience를 애스펙트로 사용하기 위해 남은 일은, 스프링 컨텍스트에 @AspectJ 애너테이션 기반의 빈을 프록시 어드바이스로 변환하는 방법을 아는 오토프록시 빈을 선언하는 것 뿐이다.

<aop:aspectj-autoproxy />

<aop:aspectj-autoproxt/>는 스프링 컨텍스트에 AnnotationAwareAspectJAutoProxyCreator 를 생성하고 자동으로 @AspectJ 애너테이션 붙은 빈의 @Pointcut으로 정의한 포인트컷에 일치하는 메소드를 프록시 하는 빈이 생성된다.

<aop:aspectj-autoproxt/>는 @AspectJ의 애너테이션을 가이드로만 이용해서 프록시기반 애스페트를 생성하는 것이라 여전히 메소드 호출에만 제한된 프록시를 만들 수 있는 스프링 애스펙트이지 AspectJ가 아니다.

### Around 어드바이스 애너테이션

### 애너테이션이 적용된 어드바이스에 인자 전달

### 인트로 덕션 애너테이션

-- 위 주제는 차후에 적용하기로 한다.

## AspectJ 애스펙트와 종속객체 주입

이 부분은 AspectJ로 다른 종속객체를 활용하여 처리하는 부분이다.

스프링 AOP에서는 지원하지 않지만 다른 클래스와 협력하여 사용할 때 쓰인다고만 알고 넘어가자