

5장. spring_step03_di

1. DI (Dependency Injection : 의존 주입)

=> 객체간의 결합을 느슨하게 하는 스프링의 핵심 기술이다.

- 객체사이의 의존관계를 자기 자신이 아닌 외부에 의해서 설정된다는 개념이다.
- DI컨테이너는 어떤 클래스가 필요로 하는 인스턴스를 자동으로 생성, 취득하여 연결시켜주는 역할을 한다.

(1) 의존

- 변경에 의해 영향을 받는 관계
- 의존하는 대상이 있으면, 그 대상을 구하는 방법
 - ① 의존 대상 객체를 직접 생성
 - ② DI
 - ③ 서비스 로케이터

(2) DI를 사용하는 이유

- 변경의 유연함

(3) DI 방식 (의존관계를 관리하기 위한 방법)

① 생성자 방식 (Construction Injection)

- 빈 객체를 생성하는 시점에 모든 의존 객체가 주입된다.
- 생성자를 통해서 의존관계를 연결시키는 것이다.

② 설정 메소드 방식 (Setter Injection)

- <property> 태그의 name 속성을 통해 어떤 의존 객체가 주입되는지 알 수 있다.
- 클래스 사이의 의존관계를 연결시키기 위해서 setter 메소드를 이용하는 방법이다.

(4) BeanFactory 인터페이스

- Object getBean()
=> 인수에 지정된 이름의 Bean인스턴스를 생성해서 반환
- boolean containsBean(String name)
=> 인수에 지정된 이름의 Bean이 정의 되어 있는지 여부를 반환
- String[] getAliases(String name)
=> Bean이름에 알리아스가 정의 되어 있는 경우, 그 알리아스를 반환한다

(5) ApplicationContext.xml

(1) bean 요소의 설정

id : 식별자(고유값)

name : id에 대한 별칭, 복수정의 가능

class : Bean클래스 이름, 완전한 형태의 클래스이름을 기술한다

parent : Bean정의를 상속하는 경우 지저안 새로운 Bean의 id

abstract : Bean클래스가 추상 클래스인지 여부

singleton : Bean이 싱글톤을 관리하는지 여부

lazy-init : Bean의 로딩을 지연시킬지 여부

autowire : 오토와이즈여부

dependency-check : 의존관계확인 방법

depends-on : 이 Bean이 의존할 Bean이름, 먼저 초기화 되는 것이 보장된다

init-method : Bean초기화시 실행시킬 메서드

destroy-method : Bean컨테이너 종료시 실행시킬 메서드

scope : 객체생성을 어떻게 할지 여부

ex)

```
<bean... scope="singleton">
```

- 객체를 한번만 생성하고, 그 후부터는 이미 생성된 객체를 재활용
- 객체 생성 시점 :

```
GenericXmlApplicationContext context =
```

```
new GenericXmlApplicationContext("xml파일");
```

할 때 객체를 공유

```
<bean... scope="prototype">
```

- 객체를 매번 new instance를 생성해서 반환함
- 객체 생성 시점 : Object obj = (Object) context.getBean() 시 매번 생성.

other scopes : (web관련 scope)

- scope="request"
- scope="session"
- scope="globalSession"

factory-method : 속성값으로 static method를 지정해서 해당 method를 이용하여 빈을 생성하도록 설정

(2) constructor-arg 요소속성

속성	설 명
----	-----

index	constructor의 몇번째의 인수에 값을 전달할 것인지 지정
-------	-------------------------------------

type	constructor의 어느 데이터형의 인수에 값을 전달할 것인지 지정
------	---

ref	자식요소 <ref bean="빈이름"/> 대신에 사용할수 있다
-----	------------------------------------

value	자식요소 <value>값</value> 대신에 사용할수 있다
-------	-----------------------------------

(3) property 요소의 속성

속성	설 명
----	-----

ref	자식요소 <ref bean="빈이름"/> 대신에 사용할수 있다
-----	------------------------------------

value	자식요소 <value>값</value> 대신에 사용할수 있다
-------	-----------------------------------

예제1. bean.xml에서 생성자와 setter를 통한 bean 객체 초기화

Project Name : step03
package Name : sample1
interface Name : src/main/java/sample1/MessageBean.java (interface)
class Name : src/main/java/sample1/MessageBeanImpl.java
 src/main/java/sample1/HelloSpring.java (main)
XML File : src/main/java/sample1/bean.xml

<작업 순서>

1. 프로젝트 만들기
2. JRE System Library 버전 변경하기
3. pom.xml 파일 수정하기
4. Class 파일 추가하기
5. XML 파일 만들기

<실행 결과>

MessageBeanImpl 생성자 호출
setCost() 호출
** Container 초기화 작업 **
strawberry 3000
banana 5500

예제2. applicationContext.xml로 생성자를 통한 bean 객체 초기화

Project Name : step03
package Name : sample2
interface Name : src/main/java/sample2/InterFoo.java (interface)
class Name : src/main/java/sample2/Bar.java
 src/main/java/sample2/Foo.java
 src/main/java/sample2/FooTestApp.java (main)
XML File : src/main/java/sample2/applicationContext.xml

<작업 순서>

1. Class 파일 추가하기
2. XML 파일 만들기

<실행 결과>

Foo객체 생성 : Foo() 호출

Foo객체 생성 : Foo(String)호출
전달된 String객체 : 테스트

Foo객체생성 : Foo(int, String)호출
num : 25
str : Hello

Foo객체생성 : Foo(int, String, boolean)호출
num : 50
str : 안녕
flag : true

Foo객체생성 : Foo(int, String, boolean)호출
num : 10
str : Hi, Spring
flag : true

Bar객체생성
Foo객체 생성 : Foo(bar)호출
전달된 Bar객체 : sample2.Bar@ae45eb6

Foo객체 생성 : Foo(bar)호출
전달된 Bar객체 : sample2.Bar@ae45eb6

** Container 초기화 완료 **

예제3. applicationContext.xml로 Singleton 객체 만들기

Project Name : step03

package Name : sample3

interface Name : src/main/java/sample3/UserService.java (interface)

class Name : src/main/java/sample3/UserServiceImpl.java

src/main/java/sample3/UserVo.java

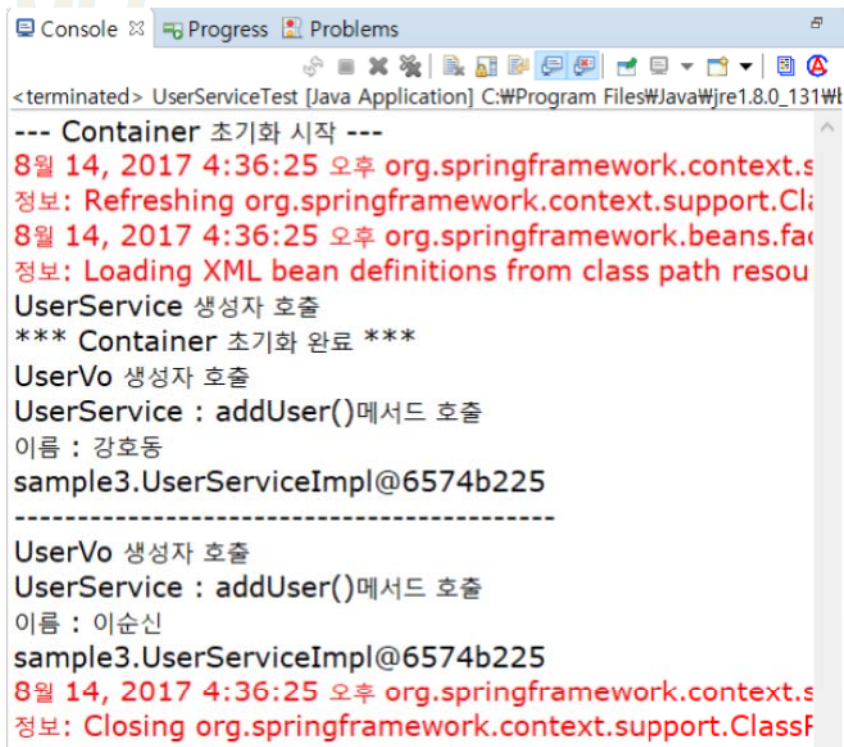
src/main/java/sample3/UserServiceTest.java (main)

XML File : src/main/java/sample3/applicationContext.xml

<작업 순서>

1. Class 파일 추가하기
2. XML 파일 만들기

<실행 결과>



```
<terminated> UserServiceTest [Java Application] C:\Program Files\Java\jre1.8.0_131\bin
--- Container 초기화 시작 ---
8월 14, 2017 4:36:25 오후 org.springframework.context.support.ClassPathXmlApplicationContext:
정보: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext
8월 14, 2017 4:36:25 오후 org.springframework.beans.factory.xml.XmlBeanDefinitionReader:
정보: Loading XML bean definitions from class path resource [applicationContext.xml]
UserService 생성자 호출
*** Container 초기화 완료 ***
UserVo 생성자 호출
UserService : addUser()메서드 호출
이름 : 강호동
sample3.UserServiceImpl@6574b225
-----
UserVo 생성자 호출
UserService : addUser()메서드 호출
이름 : 이순신
sample3.UserServiceImpl@6574b225
8월 14, 2017 4:36:25 오후 org.springframework.context.support.ClassPathXmlApplicationContext:
정보: Closing org.springframework.context.support.ClassPathXmlApplicationContext
```

예제4. <bean> 태그의 factory-method 속성값을 이용해서 singleton 객체 생성하기

Project Name : step03

package Name : sample4

interface Name : src/main/java/sample4/AbstractTest.java (abstract class)

class Name : src/main/java/sample4/Sunday.java

src/main/java/sample4/Monday.java

src/main/java/sample4/Tuesday.java

src/main/java/sample4/Wednesday.java

src/main/java/sample4/Thursday.java

src/main/java/sample4/Friday.java

src/main/java/sample4/Saturday.java

src/main/java/sample4/TestApp.java (main)

XML File : src/main/java/sample4/applicationContext.xml

<작업 순서>

1. Class 파일 추가하기
2. XML 파일 만들기

<실행 결과>

오늘은 월요일 입니다

예제5. applicationContext.xml로 <property> 태그를 이용한 bean 객체 초기화

Project Name : step03

package Name : sample5

class Name : src/main/java/sample5/DateVo.java

src/main/java/sample5/BirthdayEx.java (main)

XML File : src/main/java/sample5/applicationContext.xml

<작업 순서>

1. Class 파일 추가하기
2. XML 파일 만들기

<실행 결과>

홍길동의 생일 : 1991-10-21

이순신의 생일 : 1992-05-15

예제6. applicationContext.xml로 bean 객체 초기화 (1)

Project Name : step03
package Name : sample6
interface Name : src/main/java/sample6/Outputter.java
 src/main/java/sample6/MessageBean.java
class Name : src/main/java/sample6/FileOutput.java
 src/main/java/sample6/MessageBeanImpl.java
 src/main/java/sample6/HelloSpring.java (main)
XML File : src/main/java/sample6/applicationContext.xml

<작업 순서>

1. Class 파일 추가하기
2. XML 파일 만들기

<실행 결과>

1. Bean의 생성자 호출
2. 파일 경로와 파일 이름설정
3. phone를 입력받음
4. outputter를 입력받음

** End **

이순신 : 123-4567

5. 파일 전송 성공

6. 작업끝

=> step3/data.txt 파일이 만들어짐

이순신: 123-4567

예제7. applicationContext.xml로 bean 객체 초기화 (2)

Project Name : step03
package Name : sample7
class Name : src/main/java/sample7/Emp.java
src/main/java/sample7/Developer.java
src/main/java/sample7/Engineer.java
src/main/java/sample7/EmpMain.java (main)
XML File : src/main/java/sample7/applicationContext.xml

<작업 순서>

1. Class 파일 추가하기
2. XML 파일 만들기

<실행 결과>

이름 : 강호동 급여 : 1500000 부서 : 개발1팀(개발부)
이름 : 이순신 급여 : 2500000 부서 : 기술1팀(기술부)

예제8. applicationContext.xml로 bean 객체 초기화 - Spring DI (1)

Project Name : step03
package Name : sample8
class Name : src/main/java/sample8/Emp.java
src/main/java/sample8/Developer.java
src/main/java/sample8/Engineer.java
src/main/java/sample8/EmpMain.java (main)
XML File : src/main/java/sample8/applicationContext.xml

<작업 순서>

1. Class 파일 추가하기
2. XML 파일 만들기

<실행 결과>

이름 : 강호동 급여 : 1500000 부서 : 개발1팀(개발부)
이름 : 이순신 급여 : 2500000 부서 : 기술1팀(기술부)

예제9. applicationContext.xml로 bean 객체 초기화 - Spring DI (2)

Project Name : step03
package Name : sample9
class Name : src/main/java/sample9/Emp.java
src/main/java/sample9/Developer.java
src/main/java/sample9/Engineer.java
src/main/java/sample9/EmpMain.java (main)
XML File : src/main/java/sample9/applicationContext1.xml
src/main/java/sample9/applicationContext2.xml

<작업 순서>

1. Class 파일 추가하기
2. XML 파일 만들기

<실행 결과>

이름 : 강호동 급여 : 1500000 부서 : 개발1팀(개발부)
이름 : 이순신 급여 : 2500000 부서 : 기술1팀(기술부)

6장. spring_step04_annotation

1. Spring annotation

- 어노테이션은 자바1.5부터 지원한다
- 스프링은 어노테이션을 이용하여 빈과 관련된 정보를 설정할 수 있다.
- 스프링 프레임워크를 구현할 때 annotation 구문을 사용하여 구현하려면 다음 설정들을 필요로 한다.
현재는 주로 ②번과 ③번을 사용한다.

① CommonAnnotationBeanPostProcessor 클래스를 설정파일에 빈객체로 등록하여 어노테이션을 적용시킨다.

```
<bean
```

```
class="org.springframework.beans.factory.annotation.CommonAnnotationBeanPostProcessor"/>
```

② <context:annotation-config> 태그를 이용한다.

③ <context:component-scan base-package="xxx" /> 태그를 이용한다.

2. 주요 annotation

(1) @Autowired

- Spring에서 의존관계를 자동으로 설정할 때 사용된다.
Autowired 어노테이션이 붙은 인스턴스 변수는 해당 변수 타입과 일치하는 컨텍스트 내의 빈을 찾아 인스턴스 변수에 주입해 준다.
- 타입을 이용하여 의존객체를 자동으로 설정한다.
- 이 어노테이션은 생성자, 필드, 메소드 세곳에 적용이 가능하며, 타입을 이용한 프로퍼티 자동 설정기능을 제공한다.
의존성 주입을 위해선 생성자나 setter가 필요한데, 이 어노테이션을 사용할 경우 없어도 가능하다
- 해당 타입의 빈 객체가 없거나 2개 이상일 경우 예외를 발생시킨다.
- 프로퍼티 설정 메소드(ex: setXXX()) 형식이 아닌 일반메소드에도 적용가능하다.
- 프로퍼티 설정이 필수가 아닐 경우 @Autowired(required=false)로 선언한다. (기본값은 true)
- byType으로 의존관계를 자동으로 설정할 경우 같은 타입의 빈이 2개 이상 존재하게 되면 예외가 발생하는데, Autowired도 이러한 문제가 발생한다.
이럴 때 @Qualifier를 사용하면 동일한 타입의 빈 중 특정 빈을 사용하도록 하여 문제를 해결할 수 있다.

ex)

```
@Autowired
@Qualifier("test")
private Test test;
```

(2) @Qualifier

- 빈 객체 중, 특정 빈을 사용하도록 선언
- @Autowired이 타입기반이기 때문에 2개이상의 동일타입 빈 객체중 특정 빈을 사용하도록 선언한다.
- @Qualifier("mainBean")의 형태로 @Autowired와 같이 사용하며 해당 <bean>태그에
<qualifire value="mainBean" /> 태그를 선언해주어야 한다.
- Method에서 두개이상의 파라미터를 사용할 경우는 파라미터 앞에 선언해야한다.

(3) @Component

- 클래스에 선언하며 해당 클래스를 자동으로 bean으로 등록한다.
- bean의 이름은 해당 클래스명(첫글자는 소문자)이 사용된다.
- 범위는 디폴트로 singleton이며 @Scope를 사용하여 지정할 수 있다.

(4) @Resource

- 어플리케이션에서 필요로 하는 자원을 자동 연결할 때 사용 한다.
- 스프링 2.5 부터 지원하는 어노테이션으로 스프링에서는 의존하는 빈 객체를 전달할 때 사용한다.
name 속성에 자동으로 연결될 빈 객체의 이름을 입력한다.
ex) @Resource(name="testDao")

(5) 빈 객체 간의 의존관계 설정

- 빈 객체 간의 의존관계를 설정하기 위해 <property>를 통해서 설정
- autowire=""을 이용
- byName : 속성의 이름과 빈의 이름이 동일한 빈을 찾아서 해당 빈의 속성에 빈 객체를 설정한다.
- byType : 속성의 타입과 빈의 타입이 동일한 빈을 찾아서 해당 빈의 속성에 빈 객체를 설정한다.
- constructor : 생성자의 파라미터 타입과 동일한 타입의 빈을 찾아서 생성자의 파라미터에 설정한다.
- autodetect : 먼저 constructor 방식을 적용하고 실패한 경우에는 byType 식을 적용한다.

예제1. @Component 어노테이션으로 <bean> 객체 생성

Project Name : step04

package Name : anno1

interface Name : src/main/java/anno1/Tire.java

class Name : src/main/java/anno1/KoreaTire.java

src/main/java/anno1/Car.java

src/main/java/anno1/DriverCar.java (main)

XML File : src/main/java/anno1/bean.xml

<작업 순서>

1. 프로젝트 만들기
2. JRE System Library 버전 변경하기
3. pom.xml 파일 수정하기
4. Class 파일 추가하기
5. XML 파일 만들기

<실행 결과>

금호 타이어

금호 타이어로 만들어진 국산 자동차

예제2. <bean> 객체를 이용한 DI 자동 설정

Project Name : step04
package Name : anno2
class Name : src/main/java/anno2/Food.java
src/main/java/anno2/MyFoodMgr.java
src/main/java/anno2/FoodTest.java (main)
XML File : src/main/java/anno2/bean1.xml
src/main/java/anno2/bean2.xml

<작업 순서>

1. Class 파일 추가하기
2. XML 파일 만들기

<실행 결과>

** Container 초기화 종료 **

[좋아하는 음식=Food [foodName=스파게티, foodPrice=7500]

싫어하는 음식=Food [foodName=보신탕, foodPrice=15000]]

예제3. @Autowired 어노테이션을 이용한 자동 설정

Project Name : step04
package Name : anno3
class Name : src/main/java/anno3/Food.java
src/main/java/anno3/MyFoodMgr.java
src/main/java/anno3/FoodTest.java (main)
XML File : src/main/java/anno3/bean.xml

<작업 순서>

1. Class 파일 추가하기
2. XML 파일 만들기

<실행 결과>

** Container 초기화 종료 **

[좋아하는 음식=Food [foodName=스파게티, foodPrice=7500]

싫어하는 음식=Food [foodName=보신탕, foodPrice=15000]]

예제4. @Resource 어노테이션을 이용해서 의존하는 빈 객체를 전달

Project Name : step04
package Name : anno4
class Name : src/main/java/anno4/Student.java
 src/main/java/anno4/School.java
 src/main/java/anno4/SchoolMain.java (main)
XML File : src/main/java/anno4/bean.xml

<작업 순서>

1. Class 파일 추가하기
2. XML 파일 만들기

<실행 결과>

ABC 고등학교[학생정보 = Student [name=홍길동, address=부산, age=17] 학년 = 3]

예제5. @Service, @Scope 어노테이션

Project Name : step04
package Name : anno5
class Name : src/main/java/anno5/MyMessage.java
src/main/java/anno5/MTest.java (main)
XML File : src/main/java/anno5/bean.xml

<작업 순서>

1. Class 파일 추가하기
2. XML 파일 만들기

<실행 결과>

즐겁게 삽시다 anno5.MyMessage@3ecd23d9
즐겁게 삽시다 anno5.MyMessage@569cfc36

3. Configuration 어노테이션

(1) Configuration 어노테이션과 Bean 어노테이션

org.springframework.context.annotation의 Configuration 어노테이션과 Bean 어노테이션 코드를 이용하여 스프링 컨테이너에 새로운 빈 객체를 제공할 수 있다.

@Configuration

```
public class SpringConfig {
```

```
    @Bean
```

```
    public Greet greet(){  
        return new Hello();  
    }
```

```
}
```

위 source에서 @Bean 어노테이션은 새로운 빈 객체를 제공할때 사용되며 아래 XML 과 동일한 설정이다.

```
<bean id="greet" class="com.interwater.Hello" />
```

만약 bean id를 greet가 아닌 다른 것으로 변경하고 싶다면 @Bean(name="hello")와 같이 설정하면 된다.

(2) @Configuration 어노테이션 설정정보 사용

위에서 적용된 어노테이션을 프로그램에 활용하기 위해서는 2가지 방법이 있다.

첫째로는 AnnotationConfigApplicationContext를 이용하는 방법이 있다.

```
public static void main(String[] args) {
```

```
    ApplicationContext context = new AnnotationConfigApplicationContext(GreetConfig.class);
```

```
    GreetController greetController = context.getBean("homeController", HomeController.class);
```

```
    ....
```

```
}
```

두번째 방법으로 xml 설정을 활용하는 방법이 있다.

```
<bean class="org.springframework.context.annotation.ConfigurationClassPostProcessor" />
```

```
<bean class="com.interwater.GreetConfig"/>
```

또는

```
<context:annotation-config/>
```

```
<bean class="com.interwater.GreetConfig"/>
```

그리고 ImportResource를 통해서 Configuration 설정 클래스에서 xml 를 활용할수도 있다.

@configuration 이노테이션 비로 하단에 이레 코드를 넣어 주면 된다.

```
@importResource({"classpath://greet-repository.xml", "classpath://greet-datasource.xml"})
```

예제6. @Configuration 어노테이션과 @Bean 어노테이션

Project Name : step04
package Name : anno6
interface Name : src/main/java/anno6/UserService.java
class Name : src/main/java/anno6/UserServiceImpl.java
src/main/java/anno6/AppConfig.java
src/main/java/anno6/UserServiceTest.java (main)

<작업 순서>

1. Class 파일 추가하기

<실행 결과>

어노테이션 연습중....

7장. spring_step05_aop

1. AOP(Aspect Oriented Programming : 관점 지향 프로그래밍)

- 절차적 프로그래밍에서는 분리된 관점을 프로시저로 구성하고, OOP에서는 이를 클래스로 작성한다.
- 여기서 AOP는 OOP를 적용해도 분리해 낼수 없는 부분이 있다는 문제가 있다.
- 횡단 관심사(cross-cutting concern)의 분리를 허용함으로써 모듈성을 증가시키는 것이 목적인 프로그래밍 패러다임이다

(1) AOP 개념 및 정의

- 하나의 어플리케이션을 만들기 위해 공통으로 필요한 것이 있다.
로깅, 트랜잭션, 보안 이런 것들이 전반적으로 필요하다.
만약에 이런 것들을 extends(상속)으로 구현하기에는 한계가 있다.
- 이런 한계를 극복하기위해 AOP(Aspect Oriented Programming)기법이 필요하다
- 스프링의 DI가 클래스간의 관계에서 결합도를 낮추는데 있다면, AOP는 공통관심사의 분리에 있다.
- 쉽게 말해 클래스들이 공통으로 갖는 기능이나 절차, 서비스 등을 한데 묶어 따로 빼내겠다는 취지이다.

(2) AOP 특징점

- aop 라이브러리(spring-aop.jar)가 공통기능을 아주 알맞게 삽입해준다.
- 핵심 로직을 구현할 때 트랜잭션 적용이나 로깅, 보안검사코드를 따로 삽입할 필요가 없다.
- 전체 코드 기반에 흩어져 있는 관심사항이 하나의 장소로 응집된다.
- 서비스 모듈이 자신의 주요 관심사항(또는 핵심 기능)에 대한 코드만 포함하고
그 외의 관심사항은 모두 애스펙트로 옮겨지므로 코드가 깔끔해진다.

(3) AOP 용어

1) Aspect

- 여러 객체에 공통적으로 적용되는 공통관심사항을 Aspect라고 한다.
- 트랜잭션이나 보안등이 Aspect의 좋은 예이다.

ex) 사용 예

- 대상 정의
- 대상이 특정 액션을 취할 때 수행할 (advice) 정의
- 대상이 특정 액션을 정의 (pointcut)
- 대상과 pointcut과 advice를 조합해 advisor 정의
- advisor를 적용

2) Advice

- 언제 공통 관심기능을 핵심 로직에 적용할 지를 정의하고 있다.
- 조인포인트에 삽입되어져 동작할수 있는 코드를 '어드바이스'라 한다.
- 예를 들어 '메서드를 호출하기 전'(언제)에 '트랜잭션을 시작한다'(공통기능) 기능을 적용한다는 것을 정의하고 있다.
- 관점으로서 분리되고 실행시 모듈에 위빙된 구체적인 처리를 담당한다.
- Around Advice : Joinpoint 앞과 뒤에 실행되는 Advice
- Before Advice : Joinpoint 앞에서 실행되는 Advice
- After Running Advice : Joinpoint 정상 종료 후, 실행 되는 Advice
- After Throwing Advice : 예외가 던져질 때 (throw e 와 같은) 실행되는 Advice
- After Advice : Joinpoint 정상 종료 후, 무조건 실행 되는 Advice

3) Joinpoint

- Advice를 적용 가능한 지점을 의미한다.
- 메서드호출, 필드값 변경등이 Joinpoint에 해당한다.
- 스프링은 프록시를 이용해서 AOP를 구현하기 때문에 메서드 호출에 대한 Joinpoint만 지원한다.
- '클래스의 인스턴스 생성시점', '메소드 호출시점' 및 '예외발생시점'과 같이 어플리케이션을 실행할때 특정 작업이 시작되는 시점에서 Advice를 위빙하는 포인트를 '조인포인트'라 한다.

4) Pointcut

- 실제로 Advice가 적용되는 Joinpoint를 나타낸다.
- Advice의 정의는 Pointcut 대상으로 설정
- 하나의 Pointcut을 여러 Advice에 연결 가능, 반대로 여러 Advice를 하나의 Pointcut에 연결 가능
- 스프링에서는 정규표현식이나 AspectJ의 문법을 이용하여 Pointcut을 정의할 수 있다.

5) Advisor

- Advice와 Pointcut을 하나로 묶은 것
(스프링 AOP에만 존재 관점 지향에서 관점을 나타내는 개념)

6) Weaving

- 공통코드를 핵심로직코드에 삽입하는 것을 Weaving이라고 한다.

7) Target

- 핵심 로직을 구현하는 클래스를 말한다.

(4) AOP표현식

1) AspectJ

- PARC에서 개발한 자바 프로그래밍 언어용 관점 지향 프로그래밍 (AOP) 확장 기능이다.
- 이클립스 재단 오픈 소스 프로젝트에서 독립형 또는 이클립스로 통합하여 이용 가능하다.
- AspectJ는 최종 사용자를 위한 단순함과 이용성을 강조함으로써 폭넓게 사용되는 AOP에 대한 팩토 표준이 되었다.
- 자바 계열 문법을 사용하며 2001년 초기 출시이후 횡단구조를 표시하기 위한 IDE 연동을 포함하였다.

2) AspectJ 표현식에서 사용되는 와일드 카드

*	*는 마침표를 제외한 문자가 0개 이상 나열된것을 의미한다. (.을 포함하지 않는 모든 문자열)
..	..은 마침표를 포함한 문자가 0개 이상 나열된것을 의미한다. (.을 포함한 모든 문자열)
+	+는 주어진 타입의 임의의 서브 클래스나 서브 인터페이스를 의미한다. (하위클래스 또는 하위 인터페이스)

3) AspectJ에서 포인트컷 지정시 사용되는 논리연산자

!	표시된 조인 포인트를 제외한 모든 조인 포인트를 지정할 때 사용된다.
&&	두 조건을 모두 만족시키는 조인 포인트를 지정할 때 사용된다.
	주 조건중 하나만 만족하는 조인 포인트를 지정할 때 사용한다.

4) 매핑의 기본

get*(..)	get으로 시작하는 모든 메서드를 지정한다 ex) get(), getUser(), getUser(), getUser(UserVo vo), getUserList(), getUserList(UserVo vo, int num)
get(..)	리턴타입이 *이면 어떤 타입이 오더라도 상관없다 ex) void getUser(), int getUser(), UserVo getUser(), void getUser(UserVo vo), vo, int num)
execution(public * set*(..))	public 이면서 set으로 시작하는 메서드
execution(* sample1.*.*(..))	sample1 패키지에 있는 모든 클래스의 모든 메서드
execution(* sample1.*.*(..))	sample1 패키지 하위의 모든 클래스의 모든 메서드
execution(public * set*(..)) execution(* sample1.*.*(..))	

5) 클래스지정

com.elitebiz.*	*를 이용하면 특정패키지의 모든 클래스를 지정할수 있다. com.elite.biz 패키지의 모든 클래스를 지정한다.
com.elite..*	com.elite패키지 및 그 서브 패키지의 모든 클래스를 지정한다.
UserService+	+는 서브타입을 표현할때 사용한다. UserService클래스 혹은 UserService인터페이스로 부터 파생된 모든 서브클래스를 지정한다.
!UserService	!은 논리부정 조건을 의미한다. UserService클래스 혹은 UserService인터페이스로 부터 파생되지 않은 모든 서브클래스를 지정한다.
int Integer	연산자는 or 연산을 표현함으로 int타입으로 표현되거나 Integer 타입으로 표현되는 요소를 지칭한다.

6) 매개변수 지정

addUser(..)	매개변수의 개수와 타입이 뭐가 오든 상관없이 처리한다.
addUser(*)	반드시 1개의 임의의 매개변수가 선언되어야 한다.
addUser(*, ..)	최소 1개의 매개변수가 선언되어야 하며, 이후 다른 매개변수가 있어도 되고 없어도 된다.

ex) addUser(UserVo vo) , addUser(String id, String pwd, int no)

7) 생성자 지정

new(..)	new는 메서드의 이름을 지정하는게 아니라 생성자를 지정한다. 매개변수 상관없이 생성자를 지정한다.
UserServiceImpl.new(..)	UserServiceImpl 클래스의 new메서드가 아니라 UserServiceImpl 클래스의 생성자를 지정한다.

(5) AOP구현하는과정

- 1) Advice클래스를 작성한다.
- 2) 설정파일에 Pointcut을 설정한다.
- 3) 설정파일에 Advice와 Pointcut을 묶어 놓는 Advisor를 설정한다.
- 4) 설정파일에 ProxyFactoryBean 클래스를 이용하여 대상 객체에 Advisor를 적용한다.
- 5) getBean() 메소드로 빈 객체를 가져와 사용한다.

예제1. 공통관심사항을 개별적으로 사용

Project Name : step05

package Name : aop01

class Name : src/main/java/aop01/Man.java

src/main/java/aop01/Woman.java

src/main/java/aop01/StartMain.java (main)

<작업 순서>

1. 프로젝트 만들기
2. JRE System Library 버전 변경하기
3. pom.xml 파일 수정하기
4. Class 파일 추가하기

<실행 결과>

**** 여학생 입장 ****

교실문을 연다

컴퓨터를 켜고 Shopping을 시작한다

전등이 켜져있는지 확인한다

교실문을 잠근다

**** 남학생 입장 ****

교실문을 연다

컴퓨터를 켜고 Game을 시작한다

전등이 켜져있는지 확인한다

교실문을 잠근다

* 순수 XML기반의 설정 파일

- 개발시점에 편집기에서 xml과 java(POJO: Plain Old Java Object 방식)파일을 오가면서 작성
- 유지보수시점에 xml파일만 보면 Bean들 사이의 관계를 쉽게 알 수 있다.

예제2. 순수 XML 설정파일 기반의 AOP 동작 구현 - MethodInterceptor

Project Name : step05
package Name : aop02
interface Name : src/main/java/aop02/Person.java
class Name : src/main/java/aop02/Man.java
src/main/java/aop02/Woman.java
src/main/java/aop02/MyAspect.java
src/main/java/aop02/StartMain.java (main)
XML File : src/main/java/aop02/applicationContext.xml

<작업 순서>

1. pom.xml 파일 수정하기
2. Class 파일 추가하기
3. XML 파일 만들기

<실행 결과>

- man

교실문을 연다
컴퓨터를 켜고 Game을 시작한다
전등이 꺼져 있는지 확인한다
교실문을 잠근다

- woman

교실문을 연다
컴퓨터를 켜고 Shopping을 시작한다
전등이 꺼져 있는지 확인한다
교실문을 잠근다

예제3. XML설정 파일에서 <aop: > 태그로 구현한 AOP

Project Name : step05

package Name : aop03

interface Name : src/main/java/aop03/Person.java

class Name : src/main/java/aop03/Man.java

src/main/java/aop03/Woman.java

src/main/java/aop03/MyAspect.java

src/main/java/aop03/StartMain.java (main)

XML File : src/main/java/aop03/applicationContext.xml

<작업 순서>

1. Class 파일 추가하기
2. XML 파일 만들기

<실행 결과>

- man

교실문을 연다

컴퓨터를 켜고 Game을 시작한다

교실문을 닫는다

- woman

교실문을 연다

컴퓨터를 켜고 Shopping을 시작한다

교실문을 닫는다

* Annotation기반의 XML 설정 파일

- 일부 java파일은 스프링프레임워크에 의존하며, 개발 시점에 xml파일을 한번 설정한 후에 다시는 볼일이 없다.
- 유지보수시점에 xml파일만 봐서는 Bean들 사이의 관계를 알 수가 없다. 모든 java파일을 열어서 확인해야 정확한 구조를 파악할 수 있다. (Annotation 확인)

예제4. 어노테이션으로 구현한 AOP (1)

Project Name : step05
package Name : aop04
interface Name : src/main/java/aop04/Person.java
class Name : src/main/java/aop04/Man.java
src/main/java/aop04/Woman.java
src/main/java/aop04/MyAspect.java
src/main/java/aop04/StartMain.java (main)
XML File : src/main/java/aop04/applicationContext.xml

<작업 순서>

1. Class 파일 추가하기
2. XML 파일 만들기

<실행 결과>

- man
교실문을 연다
컴퓨터를 켜고 Game을 시작한다
교실문을 닫는다
- woman
교실문을 연다
컴퓨터를 켜고 Shopping을 시작한다
교실문을 닫는다

예제5. 어노테이션으로 구현한 AOP (2)

Project Name : step05
package Name : aop05
interface Name : src/main/java/aop05/Person.java
class Name : src/main/java/aop05/Man.java
src/main/java/aop05/Woman.java
src/main/java/aop05/MyAspect.java
src/main/java/aop05/StartMain.java (main)
XML File : src/main/java/aop05/applicationContext.xml

<작업 순서>

1. Class 파일 추가하기
2. XML 파일 만들기

<실행 결과>

- man
교실문을 연다
컴퓨터를 켜고 Game을 시작한다
교실문을 닫는다

- woman
교실문을 연다
컴퓨터를 켜고 Shopping을 시작한다
교실문을 닫는다

예제6. @Component 어노테이션으로 구현한 AOP

Project Name : step05
package Name : aop06
interface Name : src/main/java/aop06/Person.java
class Name : src/main/java/aop06/Man.java
src/main/java/aop06/Woman.java
src/main/java/aop06/MyAspect.java
src/main/java/aop06/StartMain.java (main)
XML File : src/main/java/aop06/applicationContext.xml

<작업 순서>

1. Class 파일 추가하기
2. XML 파일 만들기

<실행 결과>

- man
교실문을 연다
컴퓨터를 켜고 Game을 시작한다
교실문을 닫는다

- woman
교실문을 연다
컴퓨터를 켜고 Shopping을 시작한다
교실문을 닫는다

예제7. AOP around 동작 구현 - 순수 XML 설정파일 기반

- around는 메소드의 호출자체를 가로채 비즈니스 메소드 실행 전후에 처리할 로직을 삽입할 수 있다.

Project Name : step05

package Name : aop07

interface Name : src/main/java/aop07/MessageBean.java

class Name : src/main/java/aop07/MessageBeanImpl.java

src/main/java/aop07/LoggingAdvice.java

src/main/java/aop07/HelloApp.java (main)

XML File : src/main/java/aop07/applicationContext.xml

<작업 순서>

1. Class 파일 추가하기
2. XML 파일 만들기

<실행 결과>

- applicationContext.xml

From [applicationContext.xml]

[LOG]METHOD : sayHello is calling
Hello, Spring!!

[LOG]METHOD : sayHello was done

[LOG]처리시간 : 3 초

예제8. 어노테이션으로 구현한 AOP (3)

Project Name : step05
package Name : aop08
class Name : src/main/java/aop08/CoreEx.java
 src/main/java/aop08/AdviceEx.java
 src/main/java/aop08/CoreMain.java (main)
XML File : src/main/java/aop08/applicationContext.xml

<작업 순서>

1. Class 파일 추가하기
2. XML 파일 만들기

<실행 결과>

Joinpoint 양쪽의 전에 실행되는 Advice
Joinpoint앞에서 실행되는 Advice
5 / 2 = 2
Joinpoint 양쪽의 후에 실행되는 Advice
Joinpoint뒤에서 실행되는 Advice
Joinpoint가 정상 종료후 실행되는 Advice

Joinpoint 양쪽의 전에 실행되는 Advice
Joinpoint앞에서 실행되는 Advice
Joinpoint뒤에서 실행되는 Advice
예외가 실행될때 호출되는 Advice
0으로 나눌수 없습니다

예제9. 어노테이션으로 구현한 AOP (4)

Project Name : step05
package Name : aop09
class Name : src/main/java/aop09/CalcMethod.java
src/main/java/aop09/CalcAdvice.java
src/main/java/aop09/CalcMain.java (main)
XML File : src/main/java/aop09/applicationContext.xml

<작업 순서>

1. Class 파일 추가하기
2. XML 파일 만들기

<실행 결과>

** 연산시작 **
5 + 4 = 9
** 연산종료 **

** 연산시작 **
0으로 나눌수 없습니다.
** 연산종료 **