Spring Framework

14. AOP 어플리케이션 작성(1)

Tacademy

CONTENTS

1 Advice 클래스 작성

2 AOP 설정 및 테스트

3 PointCut 표현식

학습 목표

- Advice 클래스 작성에 대하여 이해할 수 있습니다.
- AOP 설정 및 테스트에 대하여 이해할 수 있습니다.
- PointCut 표현식에 대하여 이해할 수 있습니다.

1. Advice 클래스 작성

Spring AOP의 구현 방식

01 XML 기반의 POJO 클래스를 이용한 AOP 구현

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

02 @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.

Advice의 종류

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

1. Advice 클래스 작성

Advice 클래스 정보

- **클래스명** : PerformanceTraceAdvice.java

- **클래스 기능** : 이 어드바이스는 타겟 객체의 메서드 실행 시간을 계산해서 출력해주는 부가기능을 제공한다.

- **Advice 유형** : Around 어드바이스
  (타겟 객체의 메서드 실행 전, 후의 시간을 측정하여 계산하면 타겟 객체의 메서드 실행 시간을 알 수 있음)

- **구현 메서드명** : trace(ProceedingJoinPoint joinPoint)

JoinPoint 인터페이스

- JoinPoint는 Spring AOP 혹은 AspectJ에서 AOP가 적용되는 지점을 뜻한다.
- 해당 지점을 AspectJ에서 JoinPoint라는 인터페이스로 나타낸다.
- JoinPoint 인터페이스는 getArgs() (메서드 아규먼트를 반환한다), getThis() (프록시 객체를 반환한다), getTarget() (대상 객체를 반환한다), getSignature() (어드바이즈 되는 메서드의 설명(description)을 반환한다), toString() (어드바이즈 되는 메서드의 설명을 출력한다)과 같은 다수의 유용한 메서드를 제공한다.
- 모든 어드바이스는 org.aspectj.lang.JoinPoint 타입의 파라미터를 어드바이스 메서드에 첫 번째 매개변수로 선언할 수 있다.
- Around 어드바이스는 JoinPoint의 하위 클래스인 ProceedingJoinPoint 타입의 파라미터를 필수적으로 선언해야 한다.

JoinPoint 인터페이스

org.aspectj.lang

Interface JoinPoint

All Known Subinterfaces:

ProceedingJoinPoint

---

public interface **JoinPoint**

Provides reflective access to both the state available at a join point and static information about it.
This information is available from the body of advice using the special form thisJoinPoint.
The primary use of this reflective information is for tracing and logging applications.

Method Summary	
java.lang.Object[]	**getArgs**() Returns the arguments at this join point.
Signature	**getSignature**() Returns the signature at the join point.
java.lang.Object	**getTarget**() Returns the target object.
java.lang.String	**toString**()

ProceedingJoinPoint 인터페이스

org.aspectj.lang

Interface ProceedingJoinPoint

All Superinterfaces:

JoinPoint

```
public interface ProceedingJoinPoint
extends JoinPoint
```

ProceedingJoinPoint exposes the proceed(..) method in order to support around advice in @AJ aspects

Method Summary	
java.lang.Object	**proceed**() Proceed with the next advice or target method invocation
java.lang.Object	**proceed**(java.lang.Object[] args) Proceed with the next advice or target method invocation The given args Object[] must be in the same order and size as the advice signature but without the actual joinpoint instance

1. Advice 클래스 작성

PerformanceTraceAdvice.java

*PerformanceTraceAdvice.java

```
1  package myspring.aop.xml;
2  import org.aspectj.lang.ProceedingJoinPoint;
3
4  public class PerformanceTraceAdvice {
5      public Object trace(ProceedingJoinPoint joinPoint) throws Throwable {
6          //타겟 메서드의 signature 정보
7          String signatureString = joinPoint.getSignature().toShortString();
8          System.out.println(signatureString + " 시작");
9          //타겟의 메서드가 호출되기 전의 시간
10         long start = System.currentTimeMillis();
11         try {
12             //타겟의 메서드 호출
13             Object result = joinPoint.proceed();
14             return result;
15         } finally {
16             //타겟의 메서드가 호출된 후의 시간
17             long finish = System.currentTimeMillis();
18             System.out.println(signatureString + " 종료");
19             System.out.println(signatureString + " 실행 시간 : " +
20                (finish - start) + " ms");
21         }
22     }
23 }
```

2. AOP 설정 및 테스트

Advice 클래스를 Bean으로 등록

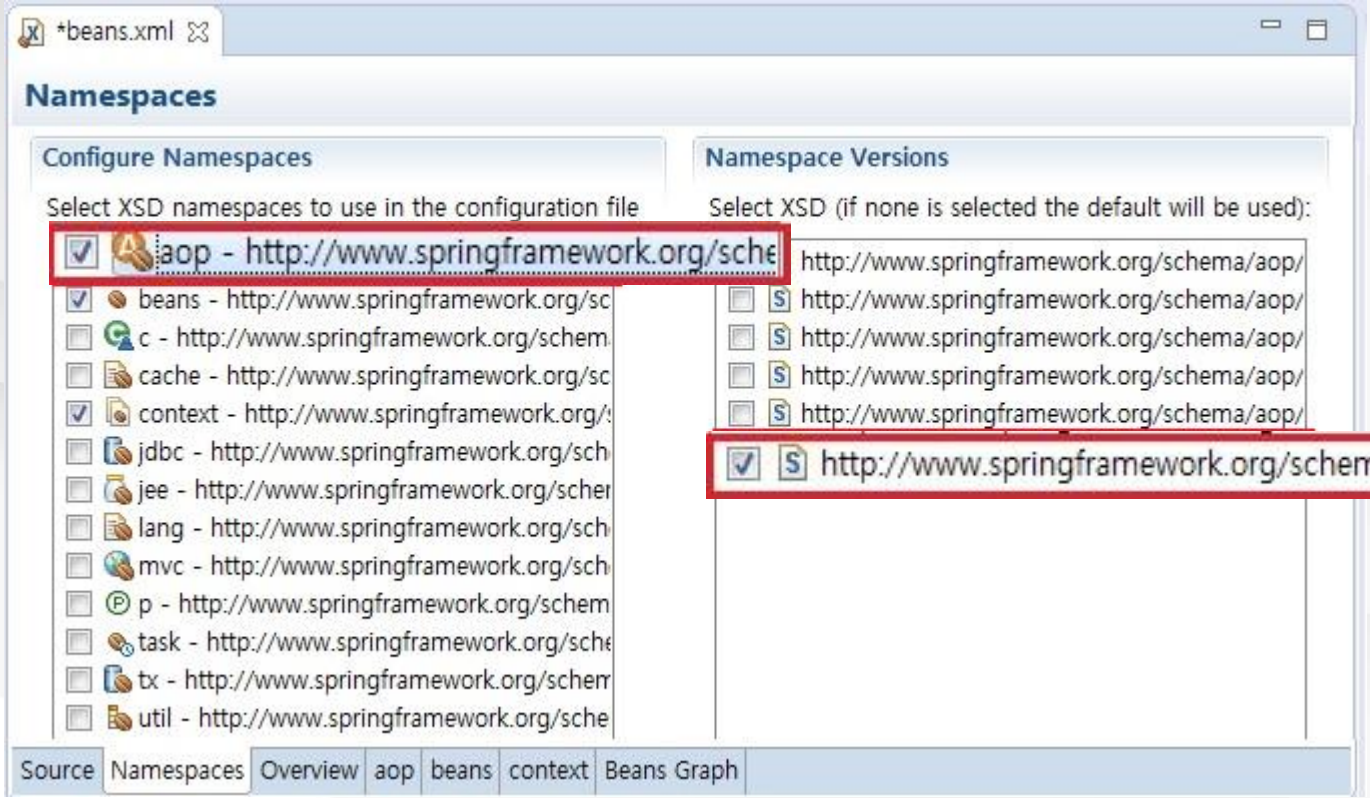
◉ 작성한 Advice 클래스를 XML 설정파일에 Bean으로 등록해야 한다.

*beans.xml

```
8  <!-- Advice 클래스를 Bean으로 등록 -->
9  <bean id="performanceTraceAdvice"
10     class="myspring.aop.xml.PerformanceTraceAdvice" />
```

AOP 네임스페이스 추가

◉ 설정 파일에 AOP와 관련된 내용을 설정하려면 aop 네임스페이스를 추가해 주어야 한다.



AOP 설정

- <aop:config> : AOP 설정 정보임을 나타낸다.
- <aop:aspect> : 애스펙트를 설정한다.
- <aop:around pointcut="execution()"> : Around 어드바이스와 포인트컷을 설정한다.

beans.xml

```
10 <!-- AOP 설정 -->
11  <aop:config>
12    <aop:aspect id="traceAspect" ref="performanceTraceAdvice">
13        <aop:around pointcut="execution(public * myspring.user.service..*(..))"
14              method="trace" />
15    </aop:aspect>
16 </aop:config>
17
18 <!-- Advice 클래스를 Bean으로 등록 -->
19 <bean id="performanceTraceAdvice"
20     class="myspring.aop.xml.PerformanceTraceAdvice" />
```

AOP 설정에 대한 설명

- <aop:aspect> 태그의 ref 속성은 애스팩트로서 기능을 제공할 Bean을 설정할 때 사용함
- <aop:around> 태그의 pointcut 속성의 execution 지시자(designator)는 어드바이스를 적용할 패키지, 클래스, 메서드를 표현할 때 사용됨
- myspring.user.service 패키지 및 그 하위 패키지에 있는 모든 public 메서드를 포인트컷으로 설정하고 있음
- UserServiceImpl의 public 메서드가 호출될 때 PerformanceTraceAdvice Bean의 trace() 메서드가 호출 되도록 설정하고 있음

```
<!-- AOP 설정 -->
 <aop:config>
   <aop:aspect id="traceAspect" ref="performanceTraceAdvice">
       <aop:around pointcut="execution(public * myspring.user.service..*(..))"
           method="trace" />
  </aop:aspect>
</aop:config>

<!-- Advice 클래스를 Bean으로 등록 -->
<bean id="performanceTraceAdvice"
    class="myspring.aop.xml.PerformanceTraceAdvice" />
```

Around Advice와 AOP 설정 테스트

◉ UserService Bean의 메서드를 호출하면, Around Advice가 적용된 것을 확인해 볼 수 있다.

UserClient.java

```
22 public class UserClient {
23     @Autowired
24     ApplicationContext context;
25
26     @Test
27     public void getUserTest() {
28         service = context.getBean(UserService.class);
29         UserVO user = service.getUser("gildong");
30         System.out.println(user);
31         assertEquals("홍길동",user.getName());
32     }
```

Markers | Console | Maven Repositories | JUnit | Properties | Dat

```
<terminated> UserClient (2) [JUnit] C:₩Program Files (x86)₩Java₩jre1.8.0_91₩bin₩javaw
UserService.getUser(..) 시작
UserService.getUser(..) 종료
UserService.getUser(..) 실행 시간 : 1919 ms
User [userId=gildong, name=홍길동, gender=남, city=서울]
```

Advice를 정의하는 태그

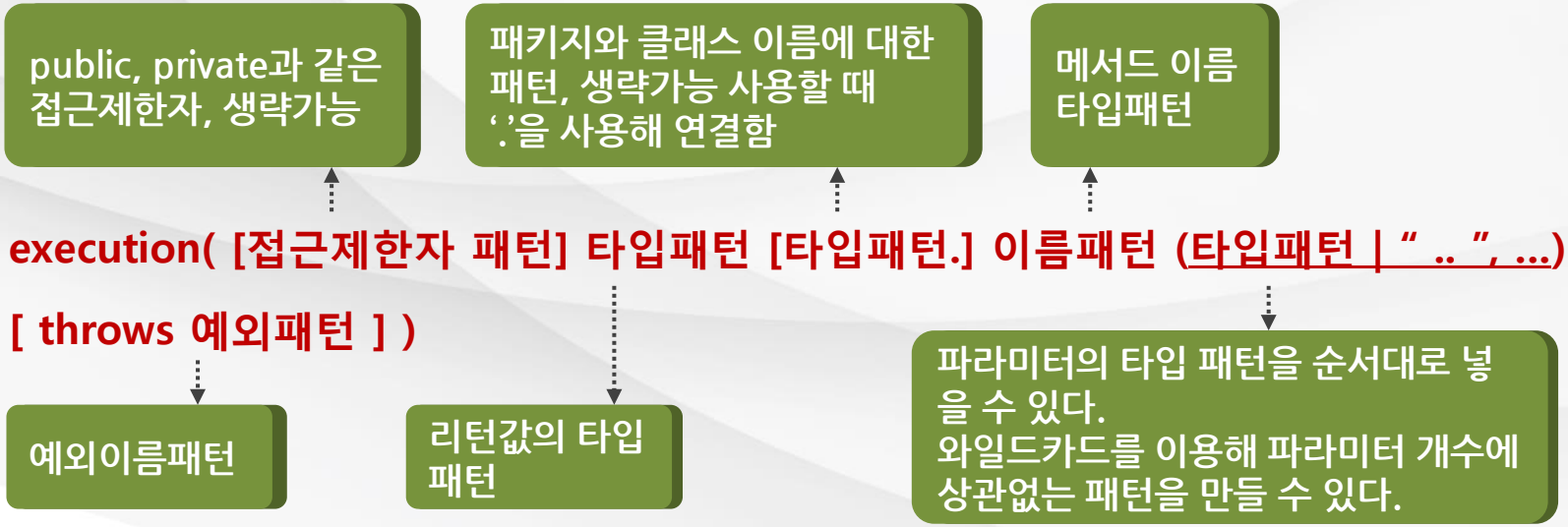
◉ 각 타입의 Advice를 정의하기 위해 아래와 같은 태그를 제공한다.

〈aop:before〉	▪ 메서드 실행 전에 적용되는 어드바이스를 정의한다.
〈aop:after-returning〉	▪ 메서드가 정상적으로 실행된 후에 적용되는 어드바이스를 정의한다.
〈aop:after-throwing〉	▪ 메서드가 예외를 발생시킬 때 적용되는 어드바이스를 정의한다. try-catch 블록에서 catch 블록과 비슷하다.
〈aop:after〉	▪ 메서드가 정상적으로 실행되는지 또는 예외를 발생시키는지 여부에 상관없이 어드바이스를 정의한다. ▪ try-catch-finally에서 finally 블록과 비슷하다.
〈aop:around〉	▪ 메서드 호출 이전, 이후, 예외발생 등 모든 시점에 적용 가능한 어드바이스를 정의한다.

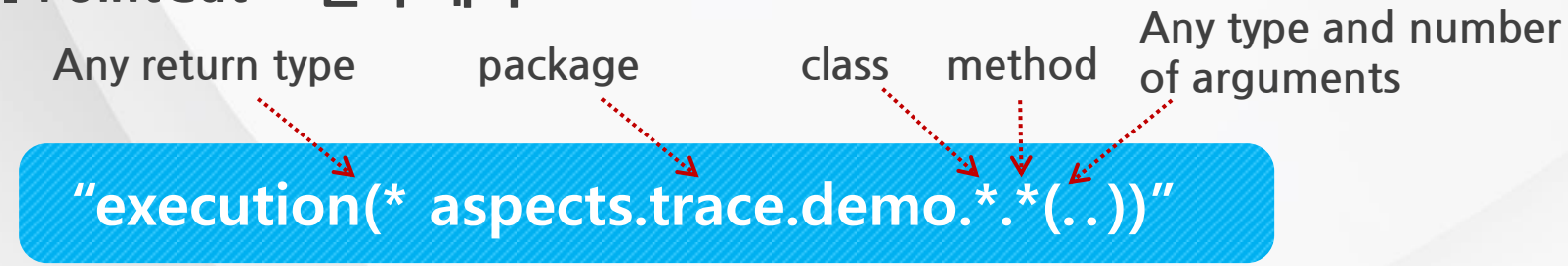
3. PointCut 표현식

PointCut 표현식 문법

- AspectJ 포인트컷 표현식은 포인트컷 지시자를 이용하여 작성한다.
- 포인트컷 지시자 중에서 가장 대표적으로 사용되는 것은 execution()이다.
- execution() 지시자를 사용한 포인트컷 표현식의 문법구조는 다음과 같다.



PointCut 표현식 예시



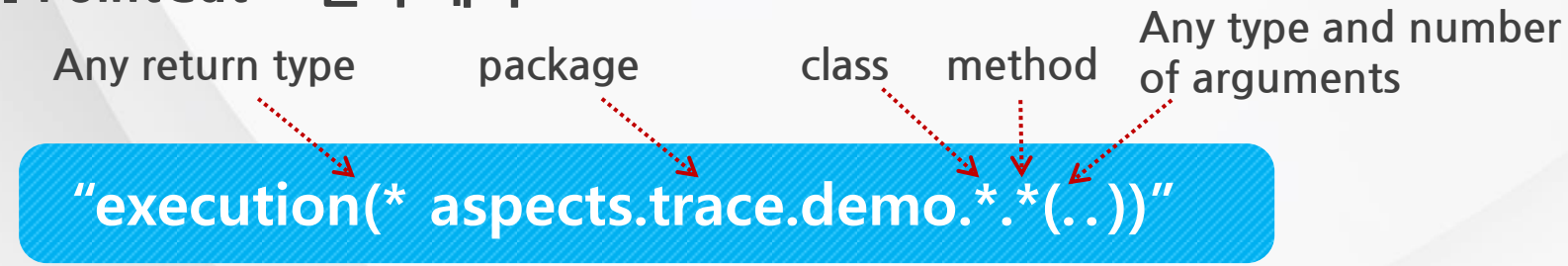
01 execution(* hello(..))

- hello라는 이름을 가진 메서드를 선정하는 것이다.
  파라미터는 모든 종류를 다 허용한다.

02 execution(* hello())

- 파라미터 패턴이 ()로 되어 있으니
  hello 메서드 중에서 파라미터가 없는 것만 선택한다.

PointCut 표현식 예시



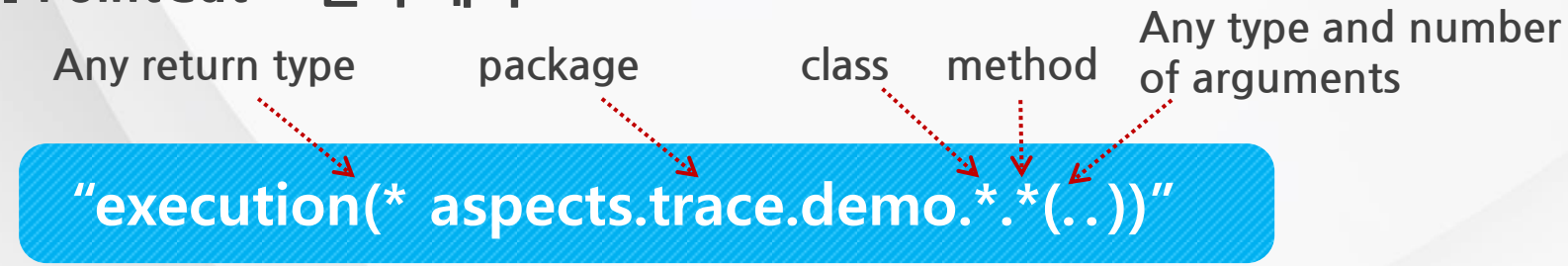
03 execution(* myspring.user.service.UserServiceImpl.*(..))

- myspring.user.service.UserServiceImpl 클래스를 직접 지정하여 이 클래스가 가진 모든 메서드를 선택한다.

04 execution(* myspring.user.service.*.*(..))

- myspring.user.service 패키지의 모든 클래스에 적용된다. 하지만 서브패키지의 클래스는 포함되지 않는다.

PointCut 표현식 예시



05 execution(* myspring.user.service..*.*(..))

- myspring.user.service 패키지의 모든 클래스에 적용된다.
  그리고 ‘..’ 를 사용해서 서브패키지의 모든 클래스까지 포함한다.

06 execution(* *..Target.*(..))

- 패키지에 상관없이 Target이라는 이름의 모든 클래스에 적용된다.
  다른 패키지에 같은 이름의 클래스가 있어도 적용이 된다는 점에
  유의해야 함

학습정리

지금까지 [AOP 어플리케이션 작성(1)]에 대해서 살펴보았습니다.

Advice 클래스 작성

Advice 종류, PointCut 인터페이스, PerformanceTraceAdvice.java

AOP설정 및 테스트

- Advice 클래스를 Bean으로 등록
- <aop:config> <aop:aspect> <aop:around> 설정

PointCut 표현식

- execution() 지시자
- execution(리턴타입패턴 패키지패턴 메서드패턴(파라미터패턴))