AOP의 개념 파악하기

한국기술교육대학교
온라인평생교육원

학습내용

- AOP의 개념
- AspectJ 활용하기
- Pointcut 활용하기

학습목표

- AOP에 대해 설명할 수 있다.
- AspectJ 활용법을 활용할 수 있다.
- Pointcut 활용법을 활용할 수 있다.

AOP의 개념

1 Aspect란

Aspect

- 객체 지향 기술에서 부가기능 모듈을 부르는 이름
- Aspect 자체로는 애플리케이션의 핵심기능을 담고 있지는 않지만 요소요소마다 공통 관심사항이 될 수 있음
- 애플리케이션을 구성하는 한 가지 측면

1 Aspect 개념이 적용되어 있지 않은 어플리케이션

부가기능이 핵심기능의 모듈에 침투해 들어가면서 설계와 코드가 지저분해짐

```
public void f1() {
  //시간체크
  //필수기능1
  //필수기능3
  //시간체크종료
  //로그 기록
}
```

부가기능 코드가 여기저기 메소드에 마구 흩어져서 나타나고 코드는 중복됨

```
public void f2() {
  //시간체크
  //필수기능 1
  //필수기능 2
  //시간체크종료
  //로그 기록
}
```

AOP의 개념

《2》 Aspect 개념을 염두에 두고 분리한 어플리케이션

코드 사이에 침투한 부가기능을 독립적인 모듈인 Aspect로 구분

각각 성격이 다른 부가기능은 다른 측면에 존재하여 독립적으로 그 코드를 살펴볼 수 있음

시간 체크

```
public void f1() {
  //필수기능1
  //필수기능3
}
```

로그

시간 체크

```
public void f2() {
  //필수기능 1
  //필수기능 2
}
```

로그

AOP의 개념

2 Aspect Oriented Programming

Aspect Oriented Programming

- 관점 지향 프로그래밍
- 애플리케이션의 핵심적인 기능에서 부가적인 기능을 분리하여 Aspect라는 모듈로 구분하여 설계하고 개발하는 방법
- AOP는 OOP를 돕는 보조적인 기술이며 OOP를 대체하는 기술은 아님
- 애플리케이션을 다양한 측면에서 독립적으로 모델링하고, 설계하고, 개발할 수 있도록 함

용어	설명
타겟 (Target)	부가기능을 부여할 대상(예 : 클래스)
어드바이스 (Advice)	▪ 타겟에게 제공할 부가기능을 담은 모듈 ▪ 객체로 정의하기도 하지만 메소드 레벨에서 정의하기도 함
조인 포인트 (Join Point)	▪ 어드바이스가 적용될 수 있는 위치 ▪ 스프링 프록시 AOP에서 조인 포인트는 메소드의 실행 단계임
포인트컷 (Pointcut)	▪ 어드바이스를 적용할 조인 포인트를 선별하는 작업 또는 그 기능을 정의한 모듈 ▪ 스프링의 포인트컷은 특정 타입 객체의 특정 메소드를 선정하는 기능

AspectJ 활용하기

1 AspectJ 설정하기

《1》 AspectJ란?

AspectJ

- @AspectJ는 Java Annotation을 사용한 일반 Java 클래스로 관점(Aspect)을 정의하는 방식
- @AspectJ 방식은 AspectJ 5 버전에서 소개되었으며, Spring은 2.0 버전부터 AspectJ 5 annotation을 지원

《2》 AspectJ 설정(pom.xml)

```
<!-- Spring AOP + AspectJ -->
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-aop</artifactId>
   <version>4.2.0.RELEASE</version>
</dependency>

<dependency>
   <groupId>org.aspectj</groupId>
   <artifactId>aspectjrt</artifactId>
   <version>1.8.0</version>
</dependency>

<dependency>
   <groupId>org.aspectj</groupId>
   <artifactId>aspectjweaver</artifactId>
   <version>1.8.6</version>
</dependency>
```

AspectJ 활용하기

1 AspectJ 설정하기

3 실습 환경 구성(com.spring.service.UserService)

```
package com.spring.service;

import com.spring.domain.User;
import com.spring.repository.UserMapper;
import org.springframework.stereotype.Service;

import javax.inject.Inject;
import java.util.Date;

@Service
public class UserService {  //타겟
    @Inject
    private UserMapper userMapper;

 public Boolean signup(User user) {
        if(user.getEmail() == null || user.getPassword() ==  null)
            return false;

        userMapper.insert(user);

        System.out.println("User created: " + new Date());
        return true;
    }
}
```

AspectJ 활용하기

1 AspectJ 설정하기

3 실습 환경 구성(com.spring.aop.AopTester)

```
package com.spring.aop;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

import java.util.Date;
@Aspect
public class AopTester {
   @Before("execution(* com.spring.service.UserService.signup(..))") //포인트컷
    public void beforeMethod(JoinPoint joinPoint) { //어드바이스
        System.out.println("Before " + joinPoint.getSignature().getName() + ": " + new
Date());
    }

    @After("execution(* com.spring.service.UserService.signup(..))")
    public void afterMethod(JoinPoint joinPoint) {
        System.out.println("After " + joinPoint.getSignature().getName() + ": " + new
Date());
    }
}
```

3 실습 환경 구성(스프링 설정파일(services.xml))

```
<aop:aspectj-autoproxy />
 <bean id="userServiceAspect" class="com.spring.aop.AopTester" />
```

AspectJ 활용하기

1 AspectJ 설정하기

3 실습 환경 구성 (com.spring.controller.UserController)

```
@Transactional
  @RequestMapping(value="/signup", method= RequestMethod.POST)
  @ResponseBody
  public String signup(@ModelAttribute User user) {
      userService.signup(user);
      return "success";
  }
```

실행 결과

```
Before: Sun Oct 09 14:36:41 JST 2020
User created at Sun Oct 09 14:36:41 JST 2020
After: Sun Oct 09 14:36:41 JST 2020
```

AspectJ 활용하기

2 AOP 적용하기

〈1〉 @AfterReturning

Pointcut에 지정한 함수가 실행 완료 후 결과값을 활용

```
@AfterReturning(pointcut = "execution(*
com.spring.service.UserService.signup(..))",
        returning= "result")
  public void afterReturningMethod(JoinPoint joinPoint, Object result) {
     System.out.println("After Returning " + joinPoint.getSignature().getName()
     + ": " + new Date() + ", Value = " + result);
  }
```

실행 결과

```
User created at Sun Oct 09 14:36:41 JST 2020
After Returning signup: Before: Sun Oct 09 14:36:41 JST 2020, Value = true
```

〈2〉 @Around

@Before와 @After를 통합하여 실행 가능

```
@Around("execution(* com.spring.cse.service.UserService.signup(..))")
  public void aroundMethod(ProceedingJoinPoint joinPoint) throws Throwable
{
     System.out.println("@Around is running!");
     System.out.println("Before " + joinPoint.getSignature().getName());
     joinPoint.proceed();
     System.out.println("After " + joinPoint.getSignature().getName());
  }
```

실행 결과

```
@Around is running!
Before signup
User created at Sun Oct 09 14:36:41 JST 2020
After signup
```

Pointcut 활용하기

1 Pointcut 표현식

⟨1⟩ 보다 복잡하고 세밀한 Pointcut 필터 기준 설정

1 클래스와 메소드 이름, 정의된 패키지, 메소드 파라미터, 리턴 값, 부여된 annotation, 구현한 인터페이스, 상속한 클래스

⟨2⟩ Pointcut 표현식

1 효과적으로 Pointcut의 클래스와 메소드는 선정하는 언어

2 문법은 다음과 같음

execution([접근제한자 패턴] 리턴 타입 패턴 [패키지 타입 패턴]이름 패턴 (파라미터 타입패턴) [throws 예외 패턴])

접근제한자 패턴	Private, Public과 같은 접근제한자로 생략 가능
리턴 타입 패턴	Return값의 타입 패턴
패키지 타입 패턴	패키지와 클래스 이름에 대한 패턴 사용할 때 '.' 를 두어 연결해야 하며, 생략 가능

execution([접근제한자 패턴] 리턴 타입 패턴 [패키지 타입 패턴]이름 패턴 (파라미터 타입 패턴) [throws 예외 패턴])

Pointcut 활용하기

1 Pointcut 표현식

이름 패턴	메소드 이름 패턴
파라미터 타입 패턴	파라미터의 타입 패턴을 순서대로 넣을 수 있으며, 와일드카드 활용 가능
예외 패턴	예외 이름에 대한 패턴

execution([접근제한자 패턴] 리턴 타입 패턴 [패키지 타입 패턴]이름 패턴 (파라미터 타입 패턴) [throws 예외 패턴])

2 Pointcut 활용

UserService의 함수에 대한 적용 결과

execution(* signup(..))	O
execution(* signup())	X
execution(* sign*(..))	O
execution(* *())	O
execution(Boolean *(..))	O
execution(* com.spring.*.*(..))	O

학습정리

1. AOP의 개념

- Aspect
  - 객체 지향 기술에서 부가기능 모듈을 부르는 이름
  - Aspect 자체로는 애플리케이션의 핵심기능을 담고 있지 않지만 요소마다 공통 관심사항이 될 수 있음
  - 애플리케이션을 구성하는 한 가지 측면

- Aspect Oriented Programming
  - 관점 지향 프로그래밍
  - 애플리케이션의 핵심적인 기능에서 부가적인 기능을 분리하여 Aspect라는 모듈로 구분하여 설계하고 개발하는 방법
  - AOP는 OOP를 돕는 보조적인 기술이며 OOP를 대체하는 기술은 아님
  - 애플리케이션을 다양한 측면에서 독립적으로 모델링하고, 설계하고, 개발할 수 있도록 함

학습정리

2. AspectJ 활용하기

- AspectJ
  - @AspectJ는 Java annotation을 사용한 일반 Java 클래스로 관점(Aspect)를 정의하는 방식
  - @AspectJ 방식은 AspectJ 5 버전에서 소개되었으며, Spring은 2.0 버전부터 AspectJ 5 Annotation을 지원함

- AOP 적용
  - @AfterReturning : Pointcut에 지정한 함수가 실행 완료 후 결과값을 활용
  - @Around : @Before와 @After를 통합하여 실행 가능

학습정리

3. Pointcut 활용하기

- Pointcut 표현식
  - 보다 복잡하고 세밀한 Pointcut 필터 기준 설정
  - 효과적으로 Pointcut의 클래스와 메소드를 선정하는 언어

- Pointcut 표현식
  - 접근제한자 패턴 : private, public과 같은 접근제한자로 생략 가능
  - 리턴 타입 패턴 : Return값의 타입 패턴
  - 패키지 타입 패턴 : 패키지와 클래스 이름에 대한 패턴. 사용할 때 '.'를 두어 연결해야 하며, 생략 가능
  - 이름 패턴 : 메소드 이름 패턴
  - 파라미터 타입 패턴 : 파라미터의 타입 패턴을 순서대로 넣을 수 있으며, 와일드카드 활용 가능
  - 예외 패턴 : 예외 이름에 대한 패턴