spring은 aop를 중요시한다.

핵심 로직과

부수적인 로직은 나누는 역할.

KB금융그룹 | 국민의 평생 금융파트너

2025년 상반기 K-디지털 트레이닝

spring aop 기본 매카니즘

AOP

[KB] IT's Your Life

KB 국민은행

번외 @Transactional도 AOP 활용. @Around 로 구성됨

spring나오기 전부터 존재. 원래 aspectj라는 라이브러리로 존재했음.

1 AOP

AOP Aspect Oriented Programming

- 관점(Aspect)지향 프로그래밍

  관심사 == 뭘 처리, 뭘해결, 뭘테스트
  뭘 위해서 할 것인지

- 관심사의 예
  - 파라미터가 올바르게 들어왔을까?
  - 이 작업을 하는 사용자가 적절한 권한을 가진 사용자인가?
  - 이 작업에서 발생할 수 있는 모든 예외는 어떻게 처리해야 하는가?

어떤 관점에서 처리하느냐가
보안
성능
...

핵심로직에 위와 관련된
코드를 종속시키면
유지보수 및 SRP원칙에도 어긋남.

- 관심사의 분리
  - 개발자가 염두에 두어야하는 일들은 별도의 관심사로 분리하고,
  - 핵심 비즈니스 로직만을 작성할 것을 권장

어느 시점에
이런 관심사가 들어갈지
동적으로 끄고 킬수
동작하는 범위도 지정할수있고

조립할 수 있다.

→ 기존의 코드(핵심 비즈니스 로직)를 수정하지 않고, 원하는 기능(관심사)들과 결합

그 관점을 기준으로 각각 분리하고 모듈화하여 재사용하겠다는 말이다.

실질적인 부가기능을 담은 구현체

AOP

KB 국민은행

AOP 용어들

컨트롤러 서비스 ... 우리가 만든 로직, 클래스...

- Target: 개발자가 작성한 핵심 비즈니스 로직을 가지는 객체
- Advice: 관심사 로직 — 성능처리할거야 보안검사할거야 예외처리할거야. 해당일을 하는 메서드를 가진 클래스가 우리가 해봤던것은 controlleradvice
- Proxy: 타겟을 감싸는 래퍼 클래스, 내부에서 타겟을 호출. 그 과정에서 관심사(adice)를 거치도록 작성
- JoinPoint: AOP를 적용할 타겟 객체의 메서드 — 클래스, 필드, 메서드 레벨에서 시점은 컴파일, 로딩, 런타임
- 외부에서의 호출은 Proxy 객체를 통해서 Target 객체의 JoinPoint를 호출

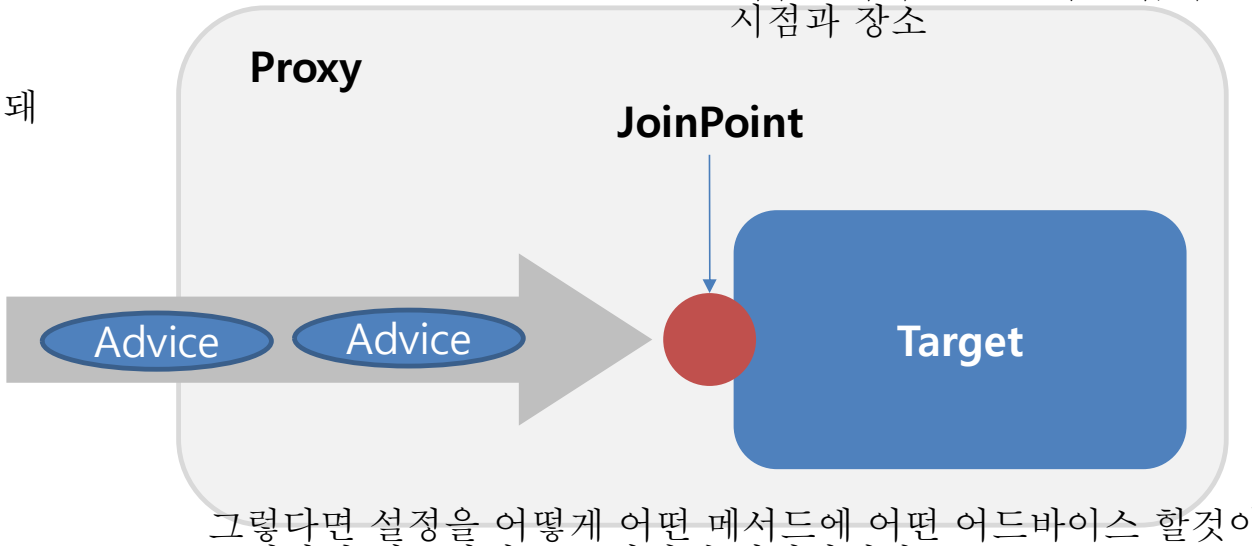
스프링 aop는
메소드 런타임(호출) 만 지원

spring aop joinpoint는 method

핵심로직과 advice가 결합되는
시점과 장소

프록시를 활용하여 원본에 대한 코드를 편집 안해도 돼

그리고 스플링의 빈이 다 프록시긴해



연결은 설정에서
이 메소드에 이 어드바이스 붙여라고 지정하고

나중에 해당 메소드(조인포인트) 호출될때
붙인 어드바이스가 호출된다

그렇다면 설정을 어떻게 어떤 메서드에 어떤 어드바이스 할것이냐
그런하면 어드바이스는 어떻게 정의할거냐

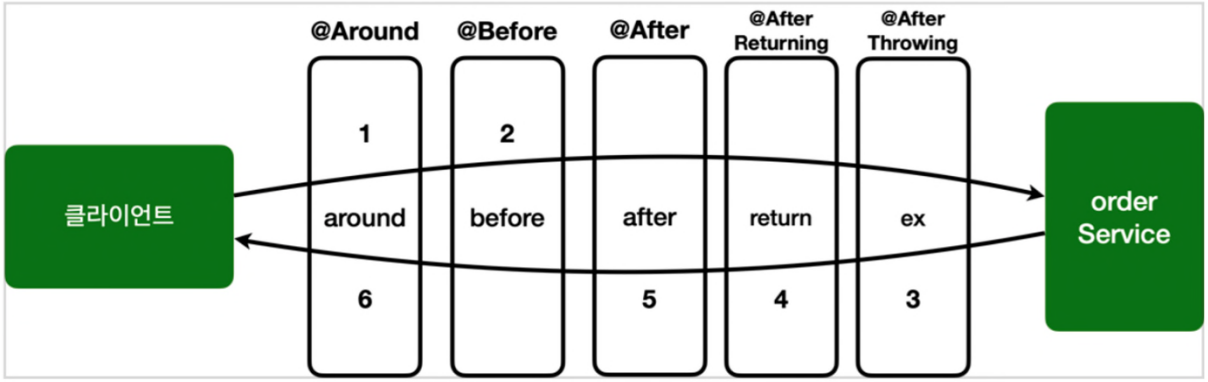
AOP 용어들

관심사 코드. 보안 유효성 성능 예외 ...

- Advice
  - 분리된 관심사 코드(메서드)
  - Adice가 동작되는 위치를 어노테이션으로 지정

어노테이션	설명
@Around	메서드 실행 전/후로 실행
@Before	JointPoint 호출 전 실행 로그인 했니? 안했으면 리다이렉트
@AfterReturning	모든 실행이 정상적으로 이루어진 후에 실행
@AfterThrowing	예외가 발생한 뒤에 실행
@After	정상적으로 실행되거나 예외가 발생했을 때 구분없이 실행되는 코드

언제 어드바이스 코드 동작하게 할까에 대하여



AOP 용어들

연결하는 것을 선택하는 것 == point cut. 문자열로된 표현식. 적용대상을 나타내는 문자열 표현식!

- Pointcut : Advice를 어떤 JointPoint에 결합할 것인지 결정하는 표현식

구분	설명
execution(@execution)	메서드를 기준으로 Pointcut을 설정
within(@within)	특정 타입(클래스)을 기준으로 Pointcut을 설정
this	주어진 인터페애스를 구현한 객체를 대상으로 Pointcut을 지정
args(@args)	특정 파라미터를 가지는 대상들만 Pointcut으로 설정
@annotation	특정한 어노테이션이 적용된 대상들만을 Pointcut으로 설정

리턴타입으로 적용대상들 지정할수있음

리턴타입 제한없음

지정한 이름으로 시작하는 클래스명

해당 이름으로 시작하는 타입!

매개변수 제한없음

메서드명 제한없음

- 예: Advice + Pointcut

```
@Before("execution(* org.scoula.sample.service.SampleService*.*(..))")
public void logBefore() {
    ... 어드바이스 로직.

    관심사 로직.
```

접미어 접두어로 메소드명, 매개변수 객체, 리턴 타입. 이름문자열 표현식으로 제한 걸 수 있다.

의존 라이브러리

```
implementation 'org.aspectj:aspectjrt:1.9.20'
implementation 'org.aspectj:aspectjweaver:1.9.20'
```

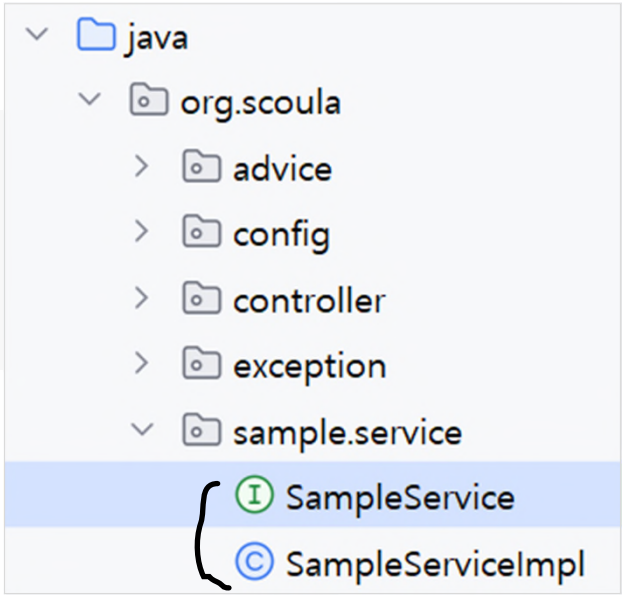
코드를 삽입하는 행위 == weaving

프로젝트

- 템플릿: SpringLegacy
- name: aopex

SampleService.java

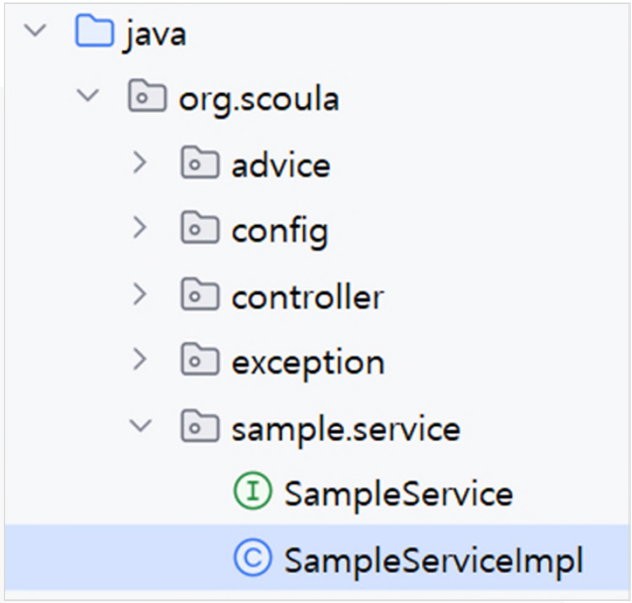
```
package org.scoula.sample.service;

public interface SampleService {
    public Integer doAdd(String str1, String str2) throws Exception;
}
```



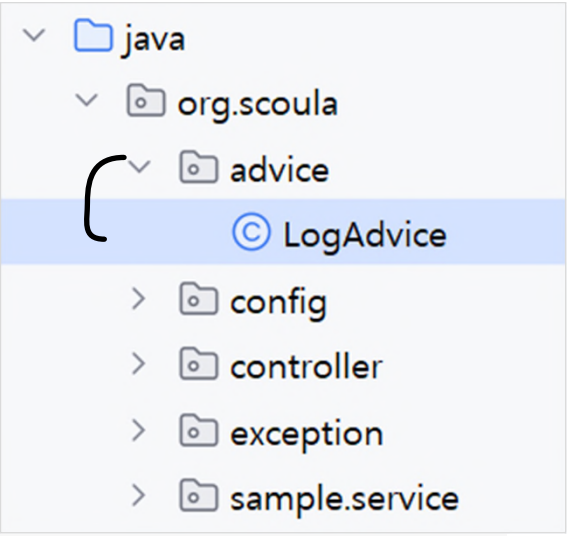
SampleServiceImpl.java

```
package org.scoula.sample.service;

import org.springframework.stereotype.Service;

@Service
public class SampleServiceImpl implements SampleService {

    @Override
    public Integer doAdd(String str1, String str2) throws Exception {
        return Integer.parseInt(str1) + Integer.parseInt(str2);
    }
}
```

넘버 포맷 예외 생길수도



LogAdvice




```
package org.scoula.advice;

import lombok.extern.log4j.Log4j;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;

@Aspect
@Log4j
@Component
public class LogAdvice {

  @Before("execution(* org.scoula.sample.service.SampleService*.*(..))")
  public void logBefore() {
    log.info("================================");
  }

}
```

포인트컷

sambleservice로 시작하는 타입의
반환 상관 없고 매개변수 상관 없고
멧드이름상관없이 == 모든 메서드.
메서드 호출되기전에 logBefore(advice)를 호출해라.

연결했으니 스프링이 해당 연겨을 알아차리게 설정하자

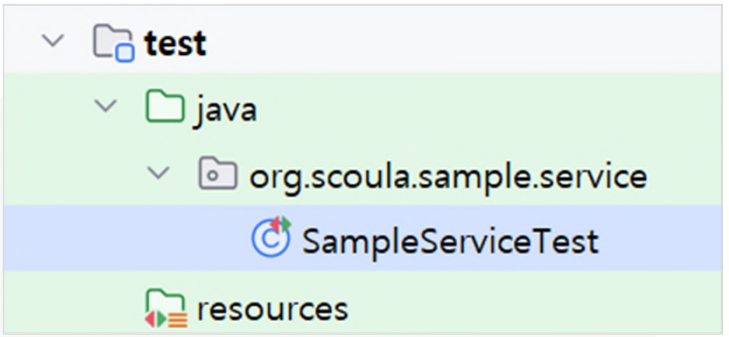
Rootconfig

```
@Configuration
@ComponentScan(basePackages = {
      "org.scoula.advice",
      "org.scoula.sample.service"
})
@EnableAspectJAutoProxy
public class RootConfig {


}
```

AOP관련된 프록시를 자동으로 만들어달라.

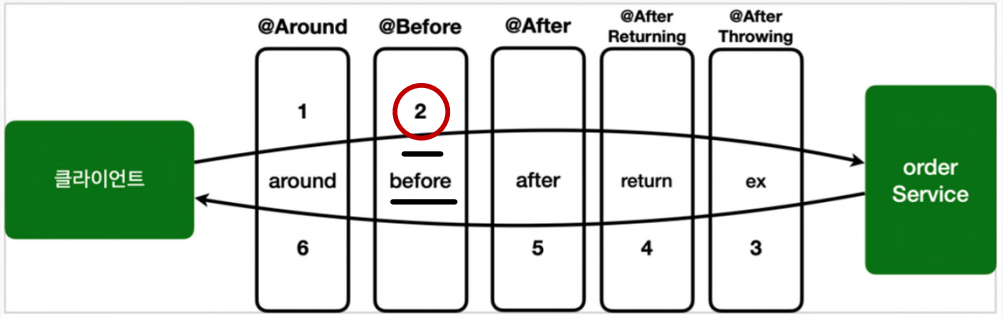
test::SampleServiceTest.java

테스트해보자




```
package org.scoula.sample.service;
...
@ExtendWith(SpringExtension.class)
@ContextConfiguration(classes = { RootConfig.class })
@Log4j2
class SampleServiceTest {
    @Autowired
    private SampleService service;

    @Test
    public void doAdd() throws Exception {
        log.info(service.doAdd("123", "456"));
    }
}
```

계속 반복되는것은
라이브 템플릿에




```
INFO : org.scoula.advice.LogAdvice - ==================================
INFO : org.scoula.sample.service.SampleServiceTest - 579
> Task :test
```

넘어오는 매개변수를 검증하는 aop를 해보자

args를 이용한 파라미터 추적

- 해당 메서드에 전달되는 파라미터가 무엇인지 기록하건, 예외가 발생했을 때 어떤 파라미터에 문제가 있는지 알고 싶은 경우
- Pointcut에 args를 이용한 파라미터 추적 설정 추가

```
...
public class LogAdvice {

 @Before("execution(* org.scoula.sample.service.SampleService*.doAdd(String, String)) && args(str1, str2)")
 public void logBeforeWithParam(String str1, String str2) {
  log.info("str1:" + str1);
  log.info("str2:" + str2);
 }

}
```

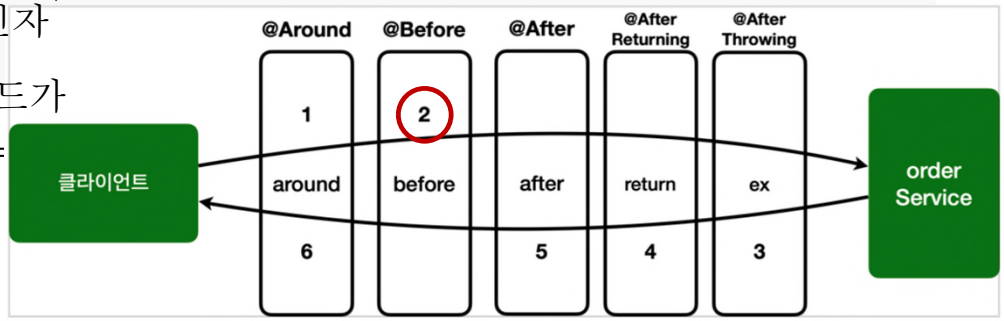
메서드명지정

면서 해당 매개변수 타입인 경우

advice메소드에
매개변수 2개를 받고싶다.
전달받고 싶다 의미

doAdd메소드가
받은 실제 인자
전달받음
advice메소드가

```
INFO : org.scoula.advice.LogAdvice - ==========================
INFO : org.scoula.advice.LogAdvice - str1:123
INFO : org.scoula.advice.LogAdvice - str2:456
INFO : org.scoula.sample.service.SampleServiceTest - 579
```



@AfterThrowing

관심사가 예외처리일때.

- 지정된 대상이 예외를 발생한 후에 동작

```
@Aspect
@Log4j
@Component
public class LogAdvice {
 ...

 @AfterThrowing(pointcut = "execution(* org.scoula.sample.service.SampleService*.*(..))", throwing="exception")
 public void logException(Exception exception) {
   log.info("Exception...!!!!");
   log.info("exception: " + exception);
 }
}
```

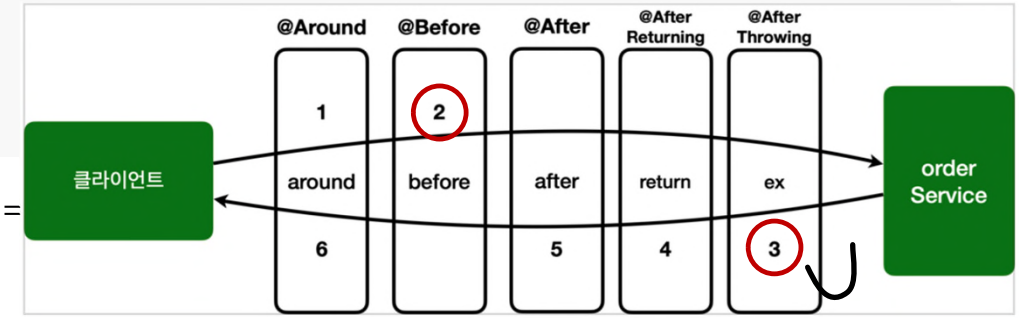
이름이저러한 클래스의
모든 메서드에서

예외매개변수 advice메서드에 던져달라

test::SampleServiceTest.java

```
public class SampleServiceTest {
   ...

   @Test
   public void addError() throws Exception {
      log.info(service.doAdd("123", "ABC"));
   }
}
```

Integer.parseInt안됨.




```
INFO : org.scoula.advice.LogAdvice - ==========================
INFO : org.scoula.advice.LogAdvice - str1:123
INFO : org.scoula.advice.LogAdvice - str2:ABC
INFO : org.scoula.advice.LogAdvice - Exception...!!!!
INFO : org.scoula.advice.LogAdvice - exception: java.lang.NumberFormatException: For input string: "ABC"
```

@ControllerAdvice기억나죠?
사실그게 AOP였어요

호출전에 호출후에. 제일 까다로움

@Around와 ProceedingJoinPoint

- 메서드의 실행 전과 실행 후에 처리가 가능

```
...
public class LogAdvice {

 @Around("execution(* org.scoula.sample.service.SampleService*.*(..))")
 public Object logTime(ProceedingJoinPoint pjp) {
   long start = System.currentTimeMillis(); 시작 시간 측정

   log.info("Target: " + pjp.getTarget());
   log.info("Param: " + Arrays.toString(pjp.getArgs()));

   Object result = null;
   try {
      result = pjp.proceed(); // 실제 메서드 호출 실제 메서드 호출.
   } catch(Throwable e) {
      e.printStackTrace();
   }
```

성능 모니터링이 목적

advice가 연결된 실제 메소드가 joinpoint잖아
매개변수가 연결된 메소드의 대한 정보

객체확인 및 매개변수 확인.

메소드를 호출한다면 @Around 사용시에는
저렇게 수동적으로 호출해야하나?
자동으로 호출이 안되나

저렇게 수동적으로 호출하는 것은 테스트를 위해서인가
ㄷ아니면 실제 동작 수행을 하는 것인가

@Around와 ProceedingJoinPoint

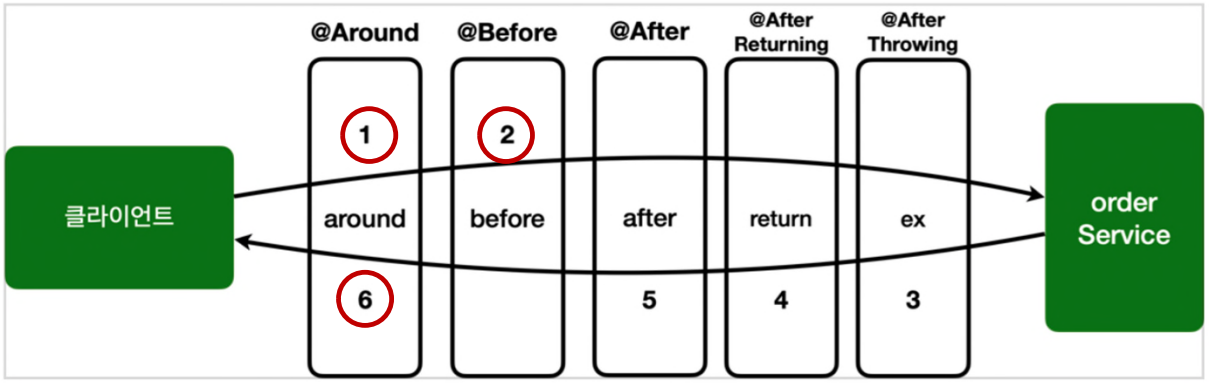
끝시간 측정

```
    long end = System.currentTimeMillis();

    log.info("TIME: " + (end - start));

    return result;
  }
}
```

INFO : org.scoula.advice.LogAdvice - Target: org.scoula.sample.service.SampleServiceImpl@74971ed9
INFO : org.scoula.advice.LogAdvice - Param: [123, 456]
INFO : org.scoula.advice.LogAdvice - ==========
INFO : org.scoula.advice.LogAdvice - str1:123
INFO : org.scoula.advice.LogAdvice - str2:456
INFO : org.scoula.advice.LogAdvice - TIME: 3
INFO : org.scoula.sample.service.SampleServiceTest - 579

before전에 around가 먼저 출력됨

그리고 호출후에도 출력됨.



하지만 예시들처럼 우리가 직접 이렇게 만들어서 사용하는 것은 드물다. 아주 자주쓰는 것은 별도의 매카니즘으로 분리되어서 사용됨. 그것들이 controlleradvice restcontrolleradvice 들이다.
또한 프레임웍 기능으로서 많이 활용된다.
일반 개발자들이 AOP를 직접 건드릴 일이 보편적이진 않다.