

제 4 강 Spring AOP

어플리케이션은 다양한 공통 기능을 필요로 한다. 로깅과 같은 기본적인 기능에서부터 트랜잭션이나 보안과 같은 기능에 이르기까지 어플리케이션 전반에 걸쳐 적용되는 공통기능이 존재한다.

이런 공통기능들은 어플리케이션의 핵심 비즈니스 로직과는 구분되는 기능이다. 핵심 비즈니스 로직과 구분하기 위해 **공통 기능을 공통 관심 사항(Cross-Cutting Concern)**이라고 표현하며, 핵심 로직을 핵심 관심 사항(Core Concern)이라고 표현한다.

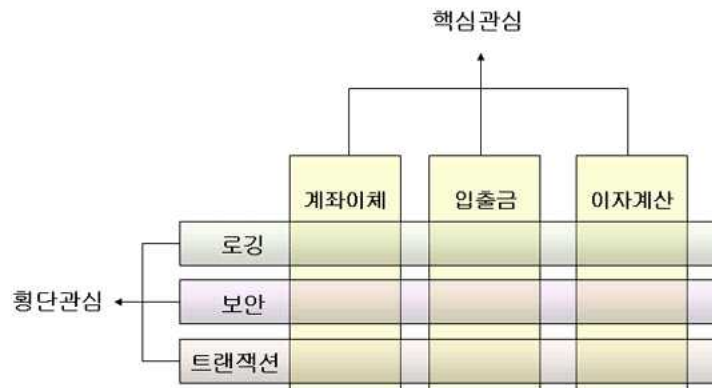
공통 관심사항들을 객체 지향 기법(상속이나 위임 등)을 사용해서 여러 모듈에 효과적으로 적용하는데 한계가 있으며 **중복된 코드를 양산**하곤 한다. 이런 한계를 극복하기 위해 AOP라는 기법이 필요하게 되었다.

비즈니스 컴포넌트 개발에서 가장 중요한 두 가지 원칙은 **낮은 결합도와 높은 응집도를 유지**하는 것이다. 스프링의 의존성 주입을 이용하면 비즈니스 컴포넌트를 구성하는 객체들의 결합도를 떨어뜨릴 수 있어 의존관계를 쉽게 변경할 수 있다.

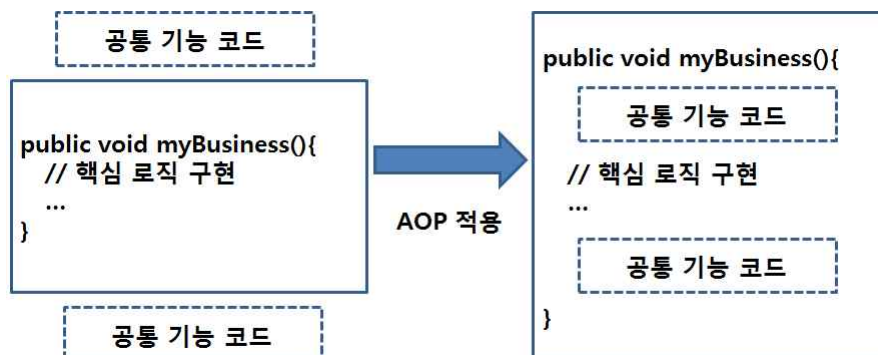
스프링의 IoC가 결합도와 관련된 기능이라면 AOP는 응집도와 관련된 기능이라고 할 수 있다.

1. AOP 소개

- Aspect Oriented Programming은 문제를 바라보는 관점을 기준으로 프로그래밍 하는 기법을 말한다.



- 핵심 로직을 구현한 코드에 공통 기능 관련 코드가 포함되어 있지 않기 때문에 적용해야 할 **공통 기능이 변경되더라도 핵심 로직을 구현한 코드를 변경할 필요가 없다**. 단지 공통기능 코드를 변경한 뒤 핵심 로직 구현 코드에 적용만 하면 된다.



① AOP 용어의 개요

- Advice : 언제 공통 관심 기능을 핵심 로직에 적용할 지를 정의하고 있다. 예를 들어 “메서드를 호출하기 전” (언제)에 “트랜잭션을 시작한다.” (공통기능)을 적용한다는 것을 정의하고 있다.
- Joinpoint : Advice를 적용 가능한 지점을 의미한다. 메서드 호출, 필드 값 변경 등이 이에 해당한다.
- Pointcut : Joinpoint의 부분 집합으로서 실제로 Advice가 적용되는 Joinpoint를 나타낸다. 스프링에서는 정규표현식이나 AspectJ의 문법을 이용하여 Pointcut을 정의할 수 있다.
- Weaving : Advice를 핵심 로직 코드에 적용하는 것을 weaving이라고 한다. 즉 공통 코드를 핵심 로직 코드에 삽입하는 것이 weaving이다.
- Aspect : 여러 객체에 공통으로 적용되는 공통 관심 사항을 Aspect라고 한다. 트랜잭션이나 보안, 로깅 등이 Aspect의 좋은 예이다.

② Weaving 방식

① 컴파일 시에 Weaving 하기

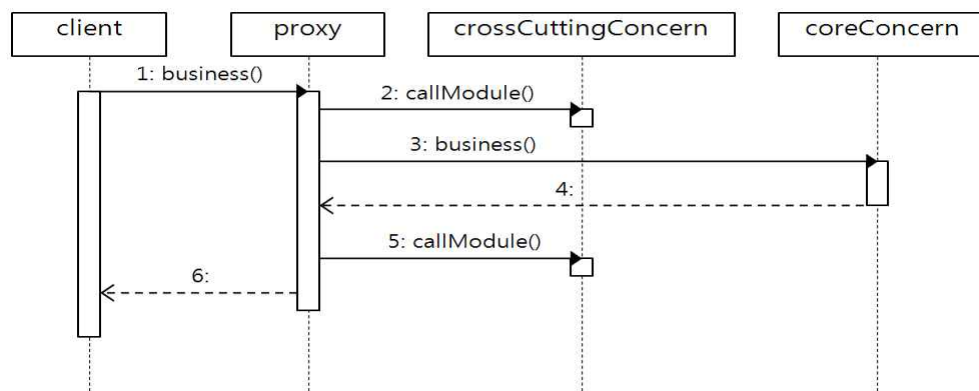
- AspectJ에서 사용하는 방식.
- 핵심 로직을 구현한 자바 소스 코드를 컴파일 할 때에 알맞은 위치에 공통 코드를 삽입. (AOP가 적용된 클래스 파일이 생성.)
- 컴파일 방식을 제공하는 AOP 도구는 공통 코드를 알맞은 위치에 삽입할 수 있도록 도와주는 컴파일러나 IDE를 함께 제공.

② 클래스 로딩 시에 Weaving 하기

- AOP 라이브러리는 JVM이 클래스를 로딩 할 때 클래스 정보를 변경할 수 있는 에이전트를 제공.
- 에이전트는 로딩한 클래스의 바이너리 정보를 변경하여 알맞은 위치에 공통 코드를 삽입한 새로운 클래스 바이너리 코드를 사용.
- 원본 클래스 파일은 변경하지 않고 클래스를 로딩 할 때에 JVM이 변경된 바이트 코드를 사용하도록 함으로써 AOP를 적용.
- AspectJ 5 버전이 컴파일 방식과 더불어 클래스 로딩 방식을 함께 지원.

③ 런타임 시에 Weaving 하기

- 소스 코드나 클래스 정보 자체를 변경하지 않음.
- 프록시를 이용하여 AOP를 적용.
- 핵심 로직을 구현한 객체에 직접 접근하는 것이 아니라 중간에 프록시를 생성하여 프록시를 통해서 핵심 로직을 구현한 객체에 접근.
- 프록시 기반의 AOP 적용 과정



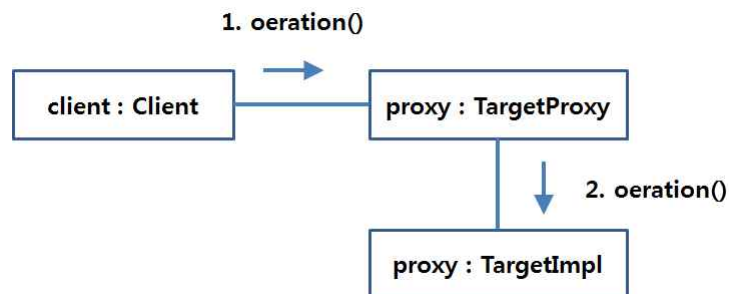
- 프록시는 핵심 로직을 실행하기 전 또는 후에 공통 기능을 적용하는 방식으로 AOP를 적용.
- 메서드가 호출될 때에만 Advice를 적용할 수 있기 때문에 필드 값 변경과 같은 Joinpoint에 대해서는 적용할 수 없는 한계가 있음

2. 스프링에서의 AOP

- 스프링은 자체적으로 프록시 기반의 AOP를 지원하고 있다.
- 스프링 AOP의 또 다른 특징은 자바 기반이라는 점이다.
- 스프링은 세가지 방식으로 AOP를 구현할 수 있도록 하고 있다.
 - XML 스키마 기반의 POJO 클래스를 이용한 AOP 구현
 - AspectJ 5/6에서 정의한 @Aspect 어노테이션 기반의 AOP 구현
 - 스프링 API를 이용한 AOP 구현
- 위의 어떤 방식을 사용하더라도 내부적으로는 프록시를 이용하여 AOP가 구현되므로 메서드 호출에 대해서만 AOP를 적용할 수 있다.

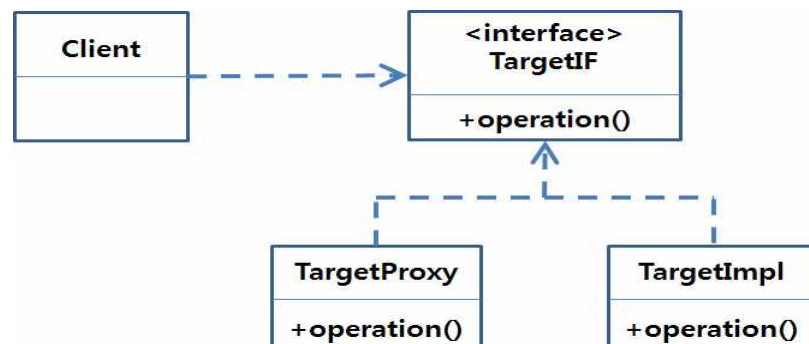
① 프록시를 이용한 AOP 구현

- 스프링은 Aspect의 적용 대상이 되는 객체에 대한 프록시를 만들어 제공하고 있으며 대상 객체를 사용하는 코드는 대상 객체에 직접 접근하기 보다는 프록시를 통해서 간접적으로 접근하게 된다.
- 프록시를 통한 AOP (아래 그림 참조)



- 어떤 대상 객체에 대해 AOP를 적용할 지의 여부는 설정파일을 통해서 지정할 수 있으며 스프링은 설정 정보를 이용하여 런타임 시에 대상 객체에 대한 프록시 객체를 생성하게 된다.
- 대상 객체가 인터페이스를 구현하고 있다면 스프링은 자바 리플렉션 API가 제공하는 java.lang.reflect.Proxy를 이용하여 프록시 객체를 생성한다.
- 인터페이스를 기반으로 프록시 객체를 생성하기 때문에 인터페이스에 정의되어 있지 않은 메서드에 대해서는 AOP가 적용되지 않는다.

- 인터페이스 기반 프록시 구조 (아래 그림 참조)



- 대상 객체가 인터페이스를 구현하고 있지 않다면 스프링은 CGLIB를 이용하여 클래스에 대한 프록시 객체를 생성한다. CGLIB는 대상 클래스를 상속받아 프록시를 구현한다. 따라서 **대상 프록시가 final인 경우는 프록시를 생성할 수 없으며 final 메서드에 대해서도 AOP를 적용할 수 없다.**

② OOP를 활용한 공통기능 처리 : 공통코드 양산과정

■ 공통으로 처리할 로직 : LogAdvice 클래스 작성하기

```
1 package tommy.spring.web.common;
2 public class LogAdvice {
3     public void printLog() {
4         System.out.println("[로그] : 비즈니스 로직 수행 전 동작");
5     }
6 }
```

■ BoardServiceImpl 클래스 수정하기.

```
1 package tommy.spring.web.board.impl;
2 import java.util.List;
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Service;
5 import tommy.spring.web.board.BoardService;
6 import tommy.spring.web.board.BoardVO;
7 import tommy.spring.web.common.LogAdvice;
8 @Service("boardService")
9 public class BoardServiceImpl implements BoardService {
10     @Autowired
11     private BoardDAO boardDAO;
12     private LogAdvice log;
13     public BoardServiceImpl() {
14         log = new LogAdvice();
15     }
16     public void insertBoard(BoardVO vo) {
17         log.printLog();
18         boardDAO.insertBoard(vo);
19     }
20     public void updateBoard(BoardVO vo) {
21         log.printLog();
22         boardDAO.updateBoard(vo);
23     }
24     public void deleteBoard(BoardVO vo) {
25         log.printLog();
26         boardDAO.deleteBoard(vo);
27     }
28     public BoardVO getBoard(BoardVO vo) {
29         log.printLog();
30         return boardDAO.getBoard(vo);
31     }
32     public List<BoardVO> getBoardList(BoardVO vo) {
33         log.printLog();
```

| | |
|----|-----------------------------------|
| 34 | return boardDAO.getBoardList(vo); |
| 35 | } |
| 36 | } |

■ 이제 BoardServiceClient 클래스를 실행하여 결과를 확인해 보고 무엇이 문제인지 생각해 보자.

```

<terminated> BoardServiceClient [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (2019. 9. 12. 오후 1:
[로그] : 비즈니스 로직 수행 전 동작
JDBC => insertBoard() 기능 처리
[로그] : 비즈니스 로직 수행 전 동작
JDBC => getBoardList() 기능 처리
--> BoardVO [seq=2, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
--> BoardVO [seq=1, title=임시제목, writer=홍길동, content=일찍..., regDate=2019-09-08, cnt=0]

```

□ 결론 : 위와 같이 작성된 프로그램은 BoardServiceImpl 클래스와 LogAdvice 객체가 소스코드에 강력하게 결합되어 있어서 LogAdvice 클래스를 다른 클래스로 변경해야 하거나 공통기능에 해당하는 printLog() 메서드의 시그니처가 변경되는 상황에서 유연하게 대처할 수 없다.

③ 문제점 확대 : 로그기능이 향상된 Log4jAdvice 클래스로 변경

■ Log4jAdvice 클래스 작성

| | |
|---|---|
| 1 | package tommy.spring.web.common; |
| 2 | public class Log4jAdvice { |
| 3 | public void printLogging() { |
| 4 | System.out.println("[로그 - Log4jAdvice] : 비즈니스 로직 수행 전 동작"); |
| 5 | } |
| 6 | } |

■ BoardServiceImpl 클래스 수정

| | |
|----|--|
| 1 | package tommy.spring.web.board.impl; |
| 2 | import java.util.List; |
| 3 | import org.springframework.beans.factory.annotation.Autowired; |
| 4 | import org.springframework.stereotype.Service; |
| 5 | import tommy.spring.web.board.BoardService; |
| 6 | import tommy.spring.web.board.BoardVO; |
| 7 | import tommy.spring.web.common.Log4jAdvice; |
| 8 | @Service("boardService") |
| 9 | public class BoardServiceImpl implements BoardService { |
| 10 | @Autowired |
| 11 | private BoardDAO boardDAO; |
| 12 | private Log4jAdvice log; |
| 13 | public BoardServiceImpl() { |
| 14 | log = new Log4jAdvice(); |
| 15 | } |
| 16 | public void insertBoard(BoardVO vo) { |
| 17 | log.printLogging(); |
| 18 | boardDAO.insertBoard(vo); |
| 19 | } |
| 20 | public void updateBoard(BoardVO vo) { |
| 21 | log.printLogging(); |

```

22         boardDAO.updateBoard(vo);
23     }
24     public void deleteBoard(BoardVO vo) {
25         log.printLogging();
26         boardDAO.deleteBoard(vo);
27     }
28     public BoardVO getBoard(BoardVO vo) {
29         log.printLogging();
30         return boardDAO.getBoard(vo);
31     }
32     public List<BoardVO> getBoardList(BoardVO vo) {
33         log.printLogging();
34         return boardDAO.getBoardList(vo);
35     }
36 }

```

■ BoardServiceClient 실행

```

<terminated> BoardServiceClient [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (2019. 9. 12. 오후
[로그 - Log4jAdvice] : 비즈니스 로직 수행 전 동작
JDBC로 insertBoard() 기능 처리
[로그 - Log4jAdvice] : 비즈니스 로직 수행 전 동작
JDBC로 getBoardList() 기능 처리
---> BoardVO [seq=3, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
---> BoardVO [seq=2, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
---> BoardVO [seq=1, title=임시제목, writer=홍길동, content=일박..., regDate=2019-09-08, cnt=0]

```

- 결론 : OOP처럼 모듈화가 뛰어난 언어를 사용하여 개발을 하더라도 공통모듈에 해당하는 Advice 클래스 객체를 생성하고 공통 메서드를 호출하는 코드가 비즈니스 메서드에 있다면 **핵심관심과 횡단관심을 완벽하게 분리할 수 없다**. 스프링 AOP는 이런 OOP의 한계를 극복할 수 있게 도와준다.

④ AOP 적용해 보기.

■ 비즈니스 클래스 수정 : BoardServiceImpl 클래스를 원상태로 되돌린다.

```

1 package tommy.spring.web.board.impl;
2 import java.util.List;
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Service;
5 import tommy.spring.web.board.BoardService;
6 import tommy.spring.web.board.BoardVO;
7 @Service("boardService")
8 public class BoardServiceImpl implements BoardService {
9     @Autowired
10     private BoardDAO boardDAO;
11     public void insertBoard(BoardVO vo) {
12         boardDAO.insertBoard(vo);
13     }
14     public void updateBoard(BoardVO vo) {
15         boardDAO.updateBoard(vo);
16     }

```

```

17         public void deleteBoard(BoardVO vo) {
18             boardDAO.deleteBoard(vo);
19         }
20         public BoardVO getBoard(BoardVO vo) {
21             return boardDAO.getBoard(vo);
22         }
23         public List<BoardVO> getBoardList(BoardVO vo) {
24             return boardDAO.getBoardList(vo);
25         }
26     }

```

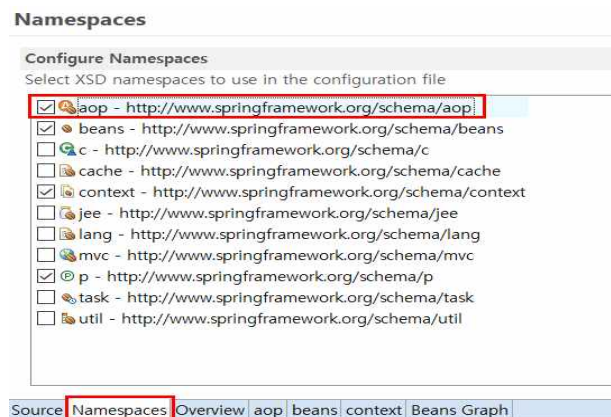
- AOP 라이브러리 추가 : mvnrepository.com에서 aspectjweaver 검색해서 pom.xml에 추가

```

<!-- AspectJ -->
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjrt</artifactId>
    <version>${org.aspectj-version}</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.9.4</version>
</dependency>
<!-- Logging -->

```

- 스프링 설정파일 수정 : 네임스페이스에 AOP 추가



- 스프링 설정파일에 LogAdvice 등록 및 AOP 설정

```

<!-- 상단부분 생략 -->
1 <context:component-scan base-package="tommy.spring.web"></context:component-scan>
2 <bean id="log" class="tommy.spring.web.common.LogAdvice"></bean>
3 <aop:config>
4     <aop:pointcut expression="execution(* tommy.spring.web.*Impl.*(..))" id="allPointcut"/>
5     <aop:aspect ref="log">
6         <aop:before method="printLog" pointcut-ref="allPointcut"/>
7     </aop:aspect>
8 </aop:config>
<!-- 하단부분 생략 -->

```


- BoardServiceClient 클래스를 실행하여 결과를 확인해 보자.

```

Markers Properties Servers Data Source Explorer Snippets Console Progress
<terminated> BoardServiceClient [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (2019. 9. 12. 오후 2:12:11)
[로그] : 비즈니스 로직 수행 전 동작
JDBC로 insertBoard() 기능 처리
[로그] : 비즈니스 로직 수행 전 동작
JDBC로 getBoardList() 기능 처리
---> BoardVO [seq=4, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
---> BoardVO [seq=3, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
---> BoardVO [seq=2, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
---> BoardVO [seq=1, title=임시제목, writer=홍길동, content=일찍..., regDate=2019-09-08, cnt=0]

```

- applicationContext.xml 설정파일에서 아래와 같이 LogAdvice를 Log4jAdvice로 수정하고 printLog를 printLogging으로 수정하여 다시 실행해 보자.

```

<!-- 상단부분 생략 -->
1 <context:component-scan base-package="tommy.spring.web"></context:component-scan>

2 <bean id="log" class="tommy.spring.web.common.Log4jAdvice"></bean>

3 <aop:config>
4     <aop:pointcut expression="execution(* tommy.spring.web..*Impl.*(..))" id="allPointcut"/>
5     <aop:aspect ref="log">
6         <aop:before method="printLogging" pointcut-ref="allPointcut"/>
7     </aop:aspect>
8 </aop:config>
<!-- 하단부분 생략 -->

```

```

Markers Properties Servers Data Source Explorer Snippets Console Progress
<terminated> BoardServiceClient [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (2019. 9. 12. 오후
[로그 - Log4jAdvice] : 비즈니스 로직 수행 전 동작
JDBC로 insertBoard() 기능 처리
[로그 - Log4jAdvice] : 비즈니스 로직 수행 전 동작
JDBC로 getBoardList() 기능 처리
---> BoardVO [seq=5, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
---> BoardVO [seq=4, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
---> BoardVO [seq=3, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
---> BoardVO [seq=2, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
---> BoardVO [seq=1, title=임시제목, writer=홍길동, content=일찍..., regDate=2019-09-08, cnt=0]

```

- 결론 : 스프링 AOP는 클라이언트가 핵심관심에 해당하는 비즈니스 메서드를 호출할 때 횡단관심에 대한 메서드를 적절하게 실행해 준다. 이때 **핵심관심 메서드와 횡단관심 메서드 사이에서 소스코드상의 결합은 발생하지 않으며** 이것이 우리가 AOP를 사용하는 주된 목적이다.

3. AOP 용어와 기본설정

① Joinpoint

- 조인포인트는 클라이언트가 호출하는 모든 비즈니스 메소드로 포인트컷의 대상 또는 포인트컷 후 보라고도 하는데 이는 조인포인트 중에서 포인트컷이 선택되기 때문이다.

② Pointcut

- 클라이언트가 호출하는 모든 메서드가 조인포인트라면 포인트컷은 필터링된 조인포인트를 의미한다. 수많은 비즈니스 메서드 중에서 우리가 원하는 특정 메서드에서만 횡단관심에 해당하는 공통 기능을 수행시키기 위해서 포인트 컷이 필요하다.

- 포인트컷 실습 : applicationContext.xml 수정

| | |
|---|---|
| | <!-- 상단부분 생략 --> |
| 1 | <aop:config> |
| 2 | <aop:pointcut expression="execution(* tommy.spring.web..*Impl.*(..))" id="allPointcut"/> |
| 3 | <aop:pointcut expression="execution(* tommy.spring.web..*Impl.get*(..))" id="getPointcut"/> |
| 4 | <aop:aspect ref="log"> |
| 5 | <aop:before method="printLogging" pointcut-ref="getPointcut"/> |
| 6 | </aop:aspect> |
| 7 | </aop:config> |
| | <!-- 하단부분 생략 --> |

- 포인트컷은 <aop:pointcut> 엘리먼트로 선언하며 id 속성으로 포인트컷을 식별하기 위한 유일한 문자열을 선언한다.

■ BoardServiceClient 실행

```

<terminated> BoardServiceClient [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (2019. 9. 12. 오후 2:33)
JDBC로 insertBoard() 기능 처리
[로그 - Log4jAdvice] : 비즈니스 로직 수행 전 동작
JDBC로 getBoardList() 기능 처리
--> BoardVO [seq=6, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
--> BoardVO [seq=5, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
--> BoardVO [seq=4, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
--> BoardVO [seq=3, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
--> BoardVO [seq=2, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
--> BoardVO [seq=1, title=임시제목, writer=홍길동, content=일찍..., regDate=2019-09-08, cnt=0]

```

- 결론 : get으로 시작하는 getBoardList() 메서드 호출에만 반응하는 것을 확인할 수 있다.

③ AspectJ의 Pointcut표현식

- XML 스키마를 이용해서 Aspect를 설정하든 @Aspect 어노테이션을 이용해서 설정하든지 AspectJ 문법을 이용하여 Pointcut을 설정한다.
- AspectJ에서는 다양한 Pointcut 명시자를 제공하는데 스프링에서는 메서드 호출과 관련된 명시자만을 지원하고 있다.

④ execution 명시자

- execution 명시자는 Advice를 적용할 메서드를 명시할 때 사용한다.
- 기본형식

execution(리턴타입패턴 패키지경로패턴.클래스이름패턴.메서드이름패턴(파라미터패턴))

- 각 패턴은 "*"을 이용하여 모든 값을 표현할 수 있다.
- "."을 이용해서 0개 이상이라는 의미를 표현할 수 있다.
- 클래스 이름 뒤에 +를 사용하여 해당 클래스로부터 파생된 모든 자식클래스를 선택.
- 인터페이스 이름 뒤에 +를 사용하면 해당 인터페이스를 구현한 모든 클래스 선택.

□ 예시와 해설

- execution(public void set*(..))
 - ▷ 리턴타입이 void 이고 메서드 이름이 set으로 시작하고 파라미터가 0개 이상인 메서드를 호출
- execution(* tommy.spring.board.*.*())

- ▷ tommy.spring.board 패키지의 파라미터가 없는 모든 메서드 호출
- execution(* tommy.spring.board..*.*(..))
- ▷ tommy.spring.board 패키지 및 하위 패키지의 파라미터가 0개 이상인 모든 메서드 호출
- execution(Integer tommy.spring.board.WriteArticleService.write(..))
- ▷ 리턴 타입이 integer인 tommy.spring.board 패키지의 WriteArticleService 인터페이스의 write() 메서드 호출
- execution(* get*(*))
- ▷ 이름이 get으로 시작하고 반드시 1개의 파라미터를 갖는 메서드 호출
- execution(* get*(*,*))
- ▷ 이름이 get으로 시작하고 2개의 파라미터를 갖는 메서드 호출
- execution(* read*(Integer, ..))
- ▷ 이름이 read로 시작하고 첫 번째 파라미터의 타입이 integer이며 1개 이상의 파라미터를 갖는 메서드 호출

㉞ within 명시자

- ☐ within 명시자는 메서드가 아닌 **특정 타입에 속하는 메서드를 Pointcut으로 설정할 때 사용**한다.
- ☐ 예시와 해설
 - within(tommy.spring.board.service.WriteArticleService)
 - ▷ WriteArticleService 인터페이스의 모든 메서드 호출
 - within(tommy.spring.board.service.*)
 - ▷ tommy.spring.board.service 패키지에 있는 모든 메서드 호출
 - within(tommy.spring.board..*)
 - ▷ tommy.spring.board 패키지 및 그 하위 패키지에 있는 모든 메서드 호출

㉟ bean 명시자

- ☐ **스프링 빈이름을 이용하여 Pointcut을 정의**한다.
- ☐ 예시와 해설
 - bean(writeArticleService) : 이름이 writeArticleService인 빈의 메서드 호출
 - bean(*ArticleService) : 이름이 ArticleService로 끝나는 빈의 메서드 호출

㊱ Pointcut의 조합

- ☐ 각각의 표현식은 "&&" 나 "||" 연산자를 이용하여 연결할 수 있다
- ☐ **스프링 설정 파일에서 "&&" 나 "||" 대신 "and" 와 "or"를 사용**하도록 하고 있다.

④ Advice

- 어드바이스는 **횡단관심에** 해당하는 **공통기능의 코드를** 의미하며 독립된 클래스와 메서드로 작성된다. 스프링에서는 어드바이스의 동작시점을 “before”, “after”, “after-returning”, “after-throwing”, “around” 등 다섯 가지로 지정할 수 있다.

■ Advice 실습 : 비즈니스 메서드 실행 후에 동작하도록 변경

```

<!-- 상단부분 생략 -->
1 <aop:config>
2   <aop:pointcut expression="execution(* tommy.spring.web..*Impl.*(..))" id="allPointcut"/>
3   <aop:pointcut expression="execution(* tommy.spring.web..*Impl.get*(..))" id="getPointcut"/>
4   <aop:aspect ref="log">
5     <aop:after method="printLogging" pointcut-ref="getPointcut"/>
6   </aop:aspect>
7 </aop:config>
<!-- 하단부분 생략 -->

```

■ BoardServiceClient 실행 및 결과 확인

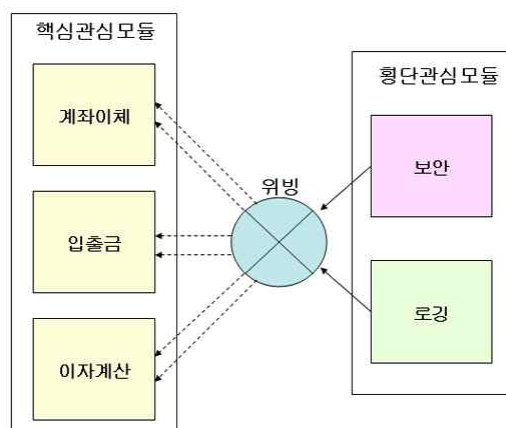
```

Markers Properties Servers Data Source Explorer Snippets Console Progress
<terminated> BoardServiceClient [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (2019. 9. 12. 오후 1:00)
JDBC => insertBoard() 기능 처리
JDBC => getBoardList() 기능 처리
[로그 - Log4jAdvice] : 비즈니스 로직 수행 전 동작
---> BoardVO [seq=7, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
---> BoardVO [seq=6, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
---> BoardVO [seq=5, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
---> BoardVO [seq=4, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
---> BoardVO [seq=3, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
---> BoardVO [seq=2, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
---> BoardVO [seq=1, title=임시제목, writer=홍길동, content=일백..., regDate=2019-09-08, cnt=0]

```

⑤ Weaving

- 위빙은 포인트컷으로 지정한 **핵심관심 메서드가 호출될 때** 어드바이스에 해당하는 **횡단관심 메서드가 삽입되는 과정**을 의미한다.



- 위빙을 처리하는 방식은 크게 컴파일타임 위빙, 로딩타임위빙, 런타임 위빙이 있지만 스프링에서는 런타임 위빙 방식만 지원한다.

⑥ Aspect or Advisor

- 애스팩트는 포인트컷과 어드바이스의 결합으로서 어떤 포인트컷 메서드에 대해서 어떤 어드바이스 메서드를 실행할지 결정한다.

```
package tommy.spring.web.common;

public class Log4jAdvice {
    public void printLogging() {
        System.out.println("[로그 - Log4jAdvice] : 비즈니스 로직 수행 전 동작");
    }
}

<bean id="log" class="tommy.spring.web.common.Log4jAdvice"></bean>

<aop:config>
    <aop:pointcut expression="execution(* tommy.spring.web.*Impl.*(..))" id="allPointcut"/>
    <aop:pointcut expression="execution(* tommy.spring.web.*Impl.get*(..))" id="getPointcut"/>
    <aop:aspect ref="log">
        <aop:after method="printLogging" pointcut-ref="getPointcut"/>
    </aop:aspect>
</aop:config>
```

- 위 그림은 getPointcut으로 설정한 포인트컷 메서드가 호출될 때(①) log라는 어드바이스 객체(②)의 printLogging() 메서드가 실행되고(③) 이때 printLogging() 메서드 동작 시점이 <aop:after>라는 내용의 설정이다.

⑦ AOP 엘리먼트

□ <aop:config> 엘리먼트

AOP 설정에서 <aop:config>는 루트 엘리먼트 이다. 스프링 설정파일 내에 여러 번 사용될 수 있으며 하위에 <aop:pointcut>, <aop:aspect>엘리먼트가 위치할 수 있다.

□ <aop:pointcut> 엘리먼트

<aop:pointcut> 엘리먼트는 포인트컷을 지정하기 위해 사용하며 <aop:config>나 <aop:aspect>의 자식 엘리먼트로 사용될 수 있다. <aop:pointcut>은 여러 개 정의할 수 있으며 유일한 아이디를 할당하여 애스팩트를 설정할 때 포인트컷을 참조하는 용도로 사용한다.

□ <aop:aspect> 엘리먼트

애스팩트는 <aop:aspect> 엘리먼트로 설정하며 핵심관심에 해당하는 포인트컷 메서드와 횡단관심에 해당하는 어드바이스 메서드를 결합하기 위해 사용한다.

□ <aop:advisor> 엘리먼트

<aop:advisor> 엘리먼트는 포인트컷과 어드바이스를 결합한다는 의미에서 애스팩트와 같은 기능을 한다. 하지만 트랜잭션 설정 같은 몇몇 특수한 경우에는 애스팩트가 아닌 어드바이저를 사용해야 한다. AOP 설정에서 애스팩트를 사용하려면 어드바이스의 아이디와 메서드 이름을 알아야 한다. 그런데 어드바이스 객체의 아이디를 모르거나 메서드 이름을 확인할 수 없을 때는 애스팩트를 설정할 수 없다.

4. Advice 동작 시점

| 종 류 | 설 명 |
|------------------------|---|
| Before Advice | 대상 객체의 메서드 호출 전에 공통 기능을 실행한다. |
| After Returning Advice | 대상 객체의 메서드가 예외 없이 실행한 이후에 공통기능을 실행한다. |
| After Throwing Advice | 대상 객체의 메서드를 실행하는 도중 예외가 발생한 경우에 공통기능을 실행한다. try~catch 블록에서 catch 블록에 해당함. |
| After Advice | 대상 객체의 메서드를 실행하는 도중에 예외가 발생했는지의 여부와 상관없이 메서드 실행 후 공통 기능을 실행한다. (try~catch~finally)의 finally와 비슷하다 |
| Around Advice | 대상 객체의 메서드 실행 전 후 또는 예외 발생 시점에 공통 기능을 실행하는데 사용된다. |

- ☐ 어드바이스 메서드의 동작시점은 <aop:aspect> 엘리먼트 하위에 각각 <aop:before>, <aop:after>, <aop:after-returning>, <aop:after-throwing>, <aop:around> 엘리먼트를 이용하여 지정한다.

① Before Advice

- ☐ Before Advice는 포인트컷으로 지정된 메서드를 호출 시 메서드가 실행되기 전에 처리되는 내용을 기술하기 위해 사용된다.

■ Before Advice 실습 : BeforeAdvice 클래스를 작성한다.

```

1 package tommy.spring.web.common;
2 public class BeforeAdvice {
3     public void beforeLog() {
4         System.out.println("[사전처리] : 비즈니스 로직 수행 전 동작");
5     }
6 }
```

■ applicationContext.xml 수정

```

<!-- 상단부분 생략 -->
1 <!-- <bean id="log" class="tommy.spring.web.common.Log4jAdvice"></bean> -->
2 <bean id="before" class="tommy.spring.web.common.BeforeAdvice"></bean>
3 <aop:config>
4     <aop:pointcut expression="execution(* tommy.spring.web..*Impl.*(..))" id="allPointcut"/>
5     <!-- getPointcut은 잠시 주석처리-->
6     <aop:aspect ref="before">
7         <aop:before method="beforeLog" pointcut-ref="allPointcut"/>
8     </aop:aspect>
9 </aop:config>
<!-- 하단부분 생략 -->
```

■ BoardServiceClient 실행 및 결과 확인

```

<terminated> BoardServiceClient [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (2019. 9. 12. 오후
[사전처리] : 비즈니스 로직 수행 전 동작
JDBC로 insertBoard() 기능 처리
[사후처리] : 비즈니스 로직 수행 후 동작
JDBC로 getBoardList() 기능 처리
---> BoardVO [seq=8, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
---> BoardVO [seq=7, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
---> BoardVO [seq=6, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
---> BoardVO [seq=5, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
---> BoardVO [seq=4, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
---> BoardVO [seq=3, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
---> BoardVO [seq=2, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
---> BoardVO [seq=1, title=임시제목, writer=홍길동, content=일찍..., regDate=2019-09-08, cnt=0]

```

② After Returning Advice

- After Returning Advice는 포인트컷으로 지정된 메서드가 정상적으로 실행되고 나서 메서드 수행 결과로 생성된 데이터를 리턴하는 시점에 동작한다.

■ After Returning Advice 실습 : AfterReturningAdvice 클래스 작성

```

1 package tommy.spring.web.common;
2 public class AfterReturningAdvice {
3     public void afterLog() {
4         System.out.println("[사후처리] : 비즈니스 로직 수행 후 처리");
5     }
6 }

```

■ 스프링 설정파일 수정

```

<!-- 상단부분 생략 -->
1 <bean id="afterReturning" class="tommy.spring.web.common.AfterReturningAdvice"></bean>
2 <aop:config>
3     <!-- allPointcut은 잠시 주석처리 -->
4     <aop:pointcut expression="execution(* tommy.spring.web..*Impl.get*(..))" id="getPointcut"/>
5     <aop:aspect ref="afterReturning">
6         <aop:after-returning method="afterLog" pointcut-ref="getPointcut"/>
7     </aop:aspect>
8 </aop:config>
<!-- 하단부분 생략 -->

```

■ BoardServiceClient 클래스 실행 및 결과확인

: 데이터가 너무 많다고 생각되면 일부를 삭제하고 실행하자.

```

<terminated> BoardServiceClient [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (2019. 9. 12. 오후 5:22
JDBC로 insertBoard() 기능 처리
JDBC로 getBoardList() 기능 처리
[사후처리] : 비즈니스 로직 수행 후 처리
---> BoardVO [seq=4, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
---> BoardVO [seq=3, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
---> BoardVO [seq=2, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
---> BoardVO [seq=1, title=임시제목, writer=홍길동, content=일찍..., regDate=2019-09-08, cnt=0]

```

③ After Throwing Advice

- After Throwing Advice는 포인트컷으로 지정한 메서드가 실행되다가 예외가 발생하는 시점에 동작한다. 따라서 예외처리 어드바이스를 설정할 때 사용한다.

■ After Throwing Advice 실습 : AfterThrowingAdvice 클래스 작성

```

1 package tommy.spring.web.common;
2 public class AfterThrowingAdvice {
3     public void exceptionLog() {
4         System.out.println("[예외처리] : 비즈니스 로직 수행 중 예외 발생");
5     }
6 }

```

■ 스프링 설정파일 수정

```

<!-- 상단부분 생략 -->
1 <bean id="afterThrowing" class="tommy.spring.web.common.AfterThrowingAdvice"></bean>
2 <aop:config>
3     <aop:pointcut expression="execution(* tommy.spring.web.*Impl.*(..))" id="allPointcut"/>
4     <!-- getPointcut은 잠시 주석처리-->
5     <aop:aspect ref="afterThrowing">
6         <aop:after-throwing method="exceptionLog" pointcut-ref="allPointcut"/>
7     </aop:aspect>
8 </aop:config>
<!-- 하단부분 생략 -->

```

■ 예외를 발생시키기 위하여 BoardServiceImpl 클래스의 insertBoard 메서드 수정

```

<!-- 상단부분 생략 -->
1 @Service("boardService")
2 public class BoardServiceImpl implements BoardService {
3     @Autowired
4     private BoardDAO boardDAO;
5     @Override
6     public void insertBoard(BoardVO vo) {
7         if(vo.getSeq() == 0) {
8             throw new IllegalArgumentException("0번 글은 등록할 수 없습니다.");
9         }
10        boardDAO.insertBoard(vo);
11    }
<!-- 하단부분 생략 -->

```

■ BoardServiceClient 클래스 실행 및 결과 확인

```

<terminated> BoardServiceClient [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (2019. 9. 12. 오후 5:38:02)
Exception in thread "main" [예외처리] : 비즈니스 로직 수행 중 예외 발생
java.lang.IllegalArgumentException: 0번 글은 등록할 수 없습니다.
    at tommy.spring.web.board.impl.BoardServiceImpl.insertBoard(BoardServiceImpl.java:19)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
    at java.lang.reflect.Method.invoke(Unknown Source)

```


④ After Advice

- ☐ try~catch~finally 구문에서 finally 블록처럼 예외 발생 여부에 상관없이 무조건 수행되는 어드바이스를 등록할 때 After Advice를 사용한다.

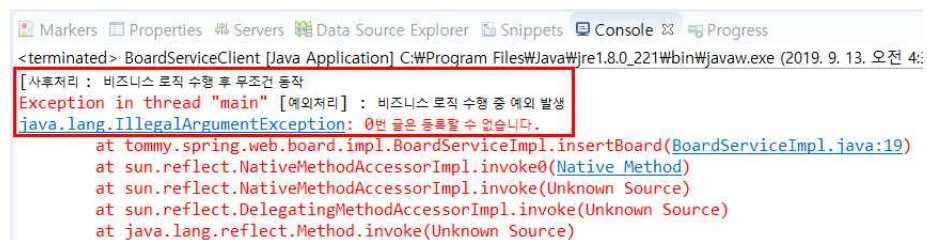
■ After Advice 실습 : AfterAdvice 클래스 작성하기

```
1 package tommy.spring.web.common;
2 public class AfterAdvice {
3     public void finallyLog() {
4         System.out.println("[사후처리] : 비즈니스 로직 수행 후 무조건 동작");
5     }
6 }
```

- 스프링 설정 파일 수정 : 무조건 수행된다는 것을 확인하기 위하여 예외발생 부분을 그대로 둔 상태에서 작업을 진행해 보자.

```
<!-- 상단부분 생략 -->
1 <bean id="afterThrowing" class="tommy.spring.web.common.AfterThrowingAdvice"></bean>
2 <bean id="after" class="tommy.spring.web.common.AfterAdvice"></bean>
3 <aop:config>
4     <aop:pointcut expression="execution(* tommy.spring.web.*Impl.*(..))" id="allPointcut"/>
5     <!-- getPointcut은 잠시 주석처리 -->
6     <aop:aspect ref="afterThrowing">
7         <aop:after-throwing method="exceptionLog" pointcut-ref="allPointcut"/>
8     </aop:aspect>
9     <aop:aspect ref="after">
10         <aop:after method="finallyLog" pointcut-ref="allPointcut"/>
11     </aop:aspect>
12 </aop:config>
<!-- 하단부분 생략 -->
```

■ BoardServiceClient 실행 및 결과 확인



```
<terminated> BoardServiceClient [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (2019. 9. 13. 오전 4:
[사후처리] : 비즈니스 로직 수행 후 무조건 동작
Exception in thread "main" [예외처리] : 비즈니스 로직 수행 중 예외 발생
java.lang.IllegalArgumentException: 0번 글은 등록할 수 없습니다.
    at tommy.spring.web.board.impl.BoardServiceImpl.insertBoard(BoardServiceImpl.java:19)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
    at java.lang.reflect.Method.invoke(Unknown Source)
```

⑤ Around Advice

- ☐ Around Advice는 비즈니스 메서드 실행전과 후에 모두 동작하여 로직을 처리한다.

■ Around Advice 실습 : AroundAdvice 클래스 작성

```
1 package tommy.spring.web.common;
2 import org.aspectj.lang.ProceedingJoinPoint;
3 public class AroundAdvice {
```

```

4      public Object aroundLog(ProceedingJoinPoint joinPoint) throws Throwable {
5          System.out.println("[BEFORE] : 비즈니스 메서드 수행 전에 처리할 내용 ...");
6          Object returnObj = joinPoint.proceed();
7          System.out.println("[AFTER] 비즈니스 메서드 수행 후에 처리할 내용 ...");
8          return returnObj;
9      }
10 }

```

■ 스프링 설정 파일 수정

```

<!-- 상단부분 생략 -->
1 <bean id="around" class="tommy.spring.web.common.AroundAdvice"></bean>
2 <aop:config>
3     <aop:pointcut expression="execution(* tommy.spring.web.*Impl.*(..))" id="allPointcut"/>
4     <!-- getPointcut 잠시 주석 처리 -->
5     <aop:aspect ref="around">
6         <aop:around method="aroundLog" pointcut-ref="allPointcut"/>
7     </aop:aspect>
8 </aop:config>
<!-- 하단부분 생략 -->

```

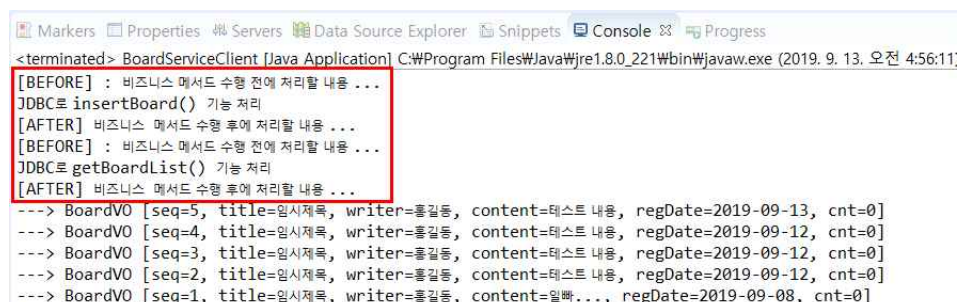
■ BoardServiceImpl 클래스 수정 : 예외발생 코드 주석처리

```

<!-- 상단부분 생략 -->
1 @Service("boardService")
2 public class BoardServiceImpl implements BoardService {
3     @Autowired
4     private BoardDAO boardDAO;
5     @Override
6     public void insertBoard(BoardVO vo) {
7         //if(vo.getSeq() == 0) {
8         //     throw new IllegalArgumentException("0번 글은 등록할 수 없습니다.");
9         //}
10        boardDAO.insertBoard(vo);
11    }
<!-- 하단부분 생략 -->

```

■ BoardServiceClient 클래스 실행 및 결과확인



```

<terminated> BoardServiceClient [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (2019. 9. 13. 오전 4:56:11)
[BEFORE] : 비즈니스 메서드 수행 전에 처리할 내용 ...
JDBC로 insertBoard() 기능 처리
[AFTER] 비즈니스 메서드 수행 후에 처리할 내용 ...
[BEFORE] : 비즈니스 메서드 수행 전에 처리할 내용 ...
JDBC로 getBoardList() 기능 처리
[AFTER] 비즈니스 메서드 수행 후에 처리할 내용 ...
--> BoardVO [seq=5, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-13, cnt=0]
--> BoardVO [seq=4, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
--> BoardVO [seq=3, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
--> BoardVO [seq=2, title=임시제목, writer=홍길동, content=테스트 내용, regDate=2019-09-12, cnt=0]
--> BoardVO [seq=1, title=임시제목, writer=홍길동, content=일찍..., regDate=2019-09-08, cnt=0]

```

5. JoinPoint와 바인드 변수

횡단관심에 해당하는 어드바이스 메서드를 의미 있게 구현하려면 클라이언트가 호출한 비즈니스 메서드의 정보가 필요하다. 스프링에서는 이런 다양한 정보들을 이용할 수 있도록 JoinPoint 인터페이스를 제공한다.

① JoinPoint 메서드

| 메서드 | 설 명 |
|--------------------------|---|
| Signature getSignature() | 클라이언트가 호출한 메서드의 시그니처(리턴타입, 이름, 매개변수) 정보가 저장된 Signature 객체를 리턴 |
| Object getTarget() | 클라이언트가 호출한 비즈니스 메서드를 포함하는 비즈니스 객체 리턴 |
| Object[] getArgs() | 클라이언트가 메서드를 호출할 때 넘겨준 인자 목록을 Object 배열로 리턴 |

- ☐ Before, After Returning, After Throwing, After 어드바이스는 JoinPoint를 사용하고 Around 어드바이스에서만 ProceedingJoinPoint를 매개변수로 사용한다. 이는 Around 어드바이스에서만 proceed() 메서드가 필요하기 때문이다.

☐ Signature 제공하는 메서드

| 메서드 | 설 명 |
|------------------------|--|
| String getName() | 클라이언트가 호출한 메서드 이름을 리턴 |
| String toLongString() | 클라이언트가 호출한 메서드의 리턴타입, 이름, 매개변수를 패키지 경로까지 포함하여 리턴 |
| String toShortString() | 클라이언트가 메서드를 메서드 시그니처를 축약한 문자열로 리턴 |

- ☐ JoinPoint 객체를 사용하려면 단지 JoinPoint를 어드바이스 메서드 매개변수로 선언만 해 주면 된다. 그러면 클라이언트가 비즈니스 메서드를 호출할 때 스프링 컨테이너가 JoinPoint 객체를 생성한다. 그리고 메서드 호출과 관련된 모든 정보를 JoinPoint 객체에 저장하여 어드바이스 메서드를 호출할 때 인자로 넘겨준다.

② Before Advice

- ☐ Before Advice는 비즈니스 메서드가 실행되기 전에 동작할 로직을 구현한다. 따라서 호출된 메서드 시그니처만 알면 다양한 사전처리 로직을 구현할 수 있다.

■ BeforeAdvice 클래스 수정

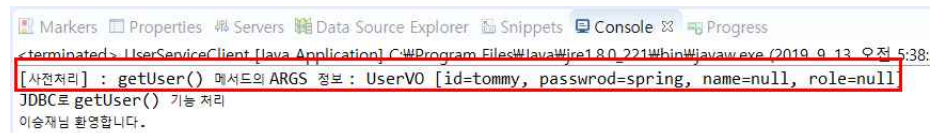
```
1 package tommy.spring.web.common;
2 import org.aspectj.lang.JoinPoint;
3 public class BeforeAdvice {
4     public void beforeLog(JoinPoint joinPoint) {
5         String method = joinPoint.getSignature().getName();
6         Object[] args = joinPoint.getArgs();
7         System.out.println("[사전처리] : " + method +
8                             "() 메서드의 ARGS 정보 : " + args[0].toString());
```

| | |
|---|---|
| 8 | } |
| 9 | } |

■ 스프링 설정 파일 수정 : 기존의 설정을 주석처리하고 아래와 같이 수정한다.

| | |
|---|---|
| | <!-- 상단 부분 생략 --> |
| 1 | <context:component-scan base-package="tommy.spring.web"></context:component-scan> |
| 2 | <bean id="before" class="tommy.spring.web.common.BeforeAdvice"></bean> |
| 3 | <aop:config> |
| | <!-- allPointcut 잠시 주석처리 --> |
| 4 | <aop:pointcut |
| | expression="execution(* tommy.spring.web..*Impl.get*(..))" id="getPointcut"/> |
| 5 | <aop:aspect ref="before"> |
| 6 | <aop:before method="beforeLog" pointcut-ref="getPointcut"/> |
| 7 | </aop:aspect> |
| 8 | </aop:config> |

■ UserServiceClient 클래스 실행 및 결과 확인



③ After Returning Advice

- After Returning Advice는 비즈니스 메서드가 수행되고 나서 결과 데이터를 리턴 할 때 동작하는 어드바이스 이다. 따라서 어떤 메서드가 어떤 값을 리턴 했는지를 알아야 사후처리 기능을 다양하게 구현할 수 있다.

■ AfterReturningAdvice 클래스 수정

| | |
|----|--|
| 1 | package tommy.spring.web.common; |
| 2 | import org.aspectj.lang.JoinPoint; |
| 3 | import tommy.spring.web.user.UserVO; |
| 4 | public class AfterReturningAdvice { |
| 5 | public void afterLog(JoinPoint joinPoint, Object returnObj) { |
| 6 | String method = joinPoint.getSignature().getName(); |
| 7 | if (returnObj instanceof UserVO) { |
| 8 | UserVO user = (UserVO) returnObj; |
| 9 | if (user.getRole().equals("admin")) { |
| 10 | System.out.println(user.getName() + " 로그인(Admin)"); |
| 11 | } |
| 12 | } |
| 13 | System.out.println("[사후처리] : " + method + "() 메서드 리턴값 : " + returnObj.toString()); |
| 14 | } |
| 15 | } |

- 위의 예제에서 afterLog() 메서드에서 클라이언트가 호출한 비즈니스 메서드 정보를 알아내기 위

해서 JoinPoint 객체를 첫 번째 매개변수로 선언하였다. 그리고 Object 타입의 변수도 두 번째 매개변수로 선언하였는데 이를 “바인딩 변수” 라고 한다. 바인딩 변수는 비즈니스 메서드가 리턴한 결과 값을 바인딩 할 목적으로 사용하며 어떤 값이 리턴 될지 모르기 때문에 Object 타입으로 선언한다.

■ 스프링 설정파일 수정

```

1 <!-- 상단 부분 생략 -->
2 <context:component-scan base-package="tommy.spring.web"></context:component-scan>
3 <bean id="afterReturning" class="tommy.spring.web.common.AfterReturningAdvice"></bean>
4 <aop:config>
5     <!-- allPointcut 잠시 주석처리 -->
6     <aop:pointcut expression="execution(* tommy.spring.web..*Impl.get*(..))"
7         id="getPointcut"/>
8     <aop:aspect ref="afterReturning">
9         <aop:after-returning method="afterLog" pointcut-ref="getPointcut"
10             returning="returnObj"/>
11     </aop:aspect>
12 </aop:config>

```

■ UserServiceClient 클래스 실행 및 결과 확인

```

<terminated> UserServiceClient [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (2019. 9. 13. 오전 10:00)
JDBC로 getUser() 기능 처리
이승재 로그인 (Admin)
[사후처리] : getUser() 메서드 리턴값 : UserVO [id=tommy, password=spring, name=이승재, role=admin]
이승재님 환영합니다.

```

④ After Throwing Advice

- ☐ After Throwing Advice는 비즈니스 메서드가 수행되다가 예외가 발생할 때 동작하는 어드바이스이다. 따라서 어떤 메서드에서 어떤 예외가 발생했는지를 알아야 한다.

■ AfterThrowingAdvice 클래스 수정

```

1 package tommy.spring.web.common;
2 import org.aspectj.lang.JoinPoint;
3 public class AfterThrowingAdvice {
4     public void exceptionLog(JoinPoint joinPoint, Exception e) {
5         String method = joinPoint.getSignature().getName();
6         System.out.println("[예외처리] : " + method +
7             "() 메서드 수행 중 발생한 예외 메시지 : " + e.getMessage());
8     }
9 }

```

■ 스프링 설정 파일 수정

```

1 <!-- 상단 부분 생략 -->
2 <context:component-scan base-package="tommy.spring.web"></context:component-scan>
3 <bean id="afterThrowing" class="tommy.spring.web.common.AfterThrowingAdvice"></bean>
4 <aop:config>

```

```

4      <aop:pointcut expression="execution(* tommy.spring.web..*Impl.*(..))" id="allPointcut"/>
      <!-- getPointcut 잠시 주석 처리 -->
5      <aop:aspect ref="afterThrowing">
6          <aop:after-throwing method="exceptionLog" pointcut-ref="allPointcut"
                                                    throwing="e"/>
7      </aop:aspect>
8  </aop:config>

```

■ BoardServiceImpl 클래스 수정 : 예외 발생 추적부분 해제

```

<!-- 상단부분 생략 -->
1  @Service("boardService")
2  public class BoardServiceImpl implements BoardService {
3      @Autowired
4      private BoardDAO boardDAO;
5      @Override
6      public void insertBoard(BoardVO vo) {
7          if(vo.getSeq() == 0) {
8              throw new IllegalArgumentException("0번 글은 등록할 수 없습니다.");
9          }
10         boardDAO.insertBoard(vo);
11     }
    <!-- 하단부분 생략 -->

```

■ BoardServiceClient 클래스 실행 : 결과를 확인한 후 위의 붉은 색 부분을 다시 주석처리 할 것.

```

<terminated> BoardServiceClient [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (2019. 9. 13. 오전 6:18:34)
[예외처리] : insertBoard() 메서드 수행 중 발생된 예외 메시지 : 0번 글은 등록할 수 없습니다.
Exception in thread "main" java.lang.IllegalArgumentException: 0번 글은 등록할 수 없습니다.
    at tommy.spring.web.board.impl.BoardServiceImpl.insertBoard(BoardServiceImpl.java:19)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
    at java.lang.reflect.Method.invoke(Unknown Source)

```

⑤ Around Advice

- Around Advice는 다른 어드바이스와는 다르게 반드시 **ProceedingJoinPoint** 객체를 매개변수로 받아야 한다. **ProceedingJoinPoint** 객체는 비즈니스 메서드를 호출하는 **proceed()** 메서드를 가지고 있으며 **JoinPoint**를 상속했다.

■ AroundAdvice 클래스 수정

```

package tommy.spring.web.common;
1  import org.aspectj.lang.ProceedingJoinPoint;
2  import org.springframework.util.StopWatch;
3  public class AroundAdvice {
4      public Object aroundLog(ProceedingJoinPoint joinPoint) throws Throwable {
5          String method = joinPoint.getSignature().getName();
6          StopWatch stopWatch = new StopWatch();
7          stopWatch.start();
8          Object returnObj = joinPoint.proceed();

```

```

9      stopWatch.stop();
10     System.out.println(method + "() 메서드 수행에 걸린 시간 : " +
                          stopWatch.getTotalTimeMillis() + "(ms)초");
11     return returnObj;
12 }
13 }

```

■ 스프링 설정 파일 수정

```

<!-- 상단 부분 생략 -->
1 <context:component-scan base-package="tommy.spring.web"></context:component-scan>
2 <bean id="around" class="tommy.spring.web.common.AroundAdvice"></bean>
3 <aop:config>
4     <!-- allPointcut 주석처리 -->
5     <aop:pointcut expression="execution(* tommy.spring.web..*Impl.get*(..))"
6                                     id="getPointcut"/>
7     <aop:aspect ref="around">
8         <aop:around method="aroundLog" pointcut-ref="getPointcut"/>
9     </aop:aspect>
10 </aop:config>

```

■ UserServiceClient 클래스 실행 및 결과 확인

```

<terminated> UserServiceClient [Java Application] C:
JDBC로 getUser() 기능 처리
getUser() 메서드 수행에 걸린 시간 : 2332(ms)초
이승재님 환영합니다.

```

6. 어노테이션 기반의 AOP

① 어노테이션 기반의 AOP 설정

- ☐ AOP를 어노테이션으로 설정하려면 가장 먼저 스프링 설정 파일에 `<aop:aspectj-autoproxy>` 엘리먼트를 선언해야 한다. `<aop:aspectj-autoproxy>` 설정만으로도 스프링 컨테이너는 AOP 관련 어노테이션을 인식하고 용도에 맞게 처리해 준다.

```

<!-- 상단 부분 생략 -->
1 <context:component-scan base-package="tommy.spring.web"></context:component-scan>
2 <aop:aspectj-autoproxy></aop:aspectj-autoproxy>
3 <!-- 나머지 모두 주석 처리 -->
4 </beans>

```

- ☐ AOP 관련 어노테이션들은 어드바이스 클래스에 설정한다. 그리고 어드바이스 클래스에 선언된 어노테이션들을 스프링 컨테이너가 처리하게 하려면 반드시 어드바이스 객체가 생성되어 있어야 한다. 따라서 어드바이스 클래스는 반드시 스프링 설정 파일에 `<bean>`으로 등록되거나 `@Service` 어노테이션을 사용하여 컴포넌트가 검색될 수 있도록 해야 한다.

| | |
|---------------|---|
| Annotation 설정 | @Service public class LogAdvice { ... } |
| XML 설정 | <bean id="log" class="tommy.spring.web.common.LogAdvice" /> |

② 포인트컷 설정

- ☐ 어노테이션 설정으로 포인트컷을 선언할 때는 @Pointcut을 사용하며 참조 메서드는 메서드 몸체가 비어있는 즉 구현 로직이 없는 메서드 이다.
- ☐ @Aspect 어노테이션을 사용하는 경우에 @Pointcut 어노테이션을 이용해서 Pointcut 설정을 재사용할 수 있다.
- ☐ @Pointcut 어노테이션은 Pointcut 표현식을 값으로 가지며 @Pointcut 어노테이션이 적용된 메서드는 리턴 타입이 반드시 void 여야 한다.
- ☐ @Pointcut 어노테이션이 적용된 메서드의 이름을 이용해서 Pointcut을 참조할 수 있다. 이때 메서드 이름은 범위에 맞게 알맞게 입력해야 한다.
 - 같은 클래스에 위치한 @Pointcut 메서드는 "메서드이름"만 입력
 - 같은 패키지에 위치한 @Pointcut 메서드는 "클래스단순이름.메서드이름"을 입력
 - 다른 패키지에 위치한 @Pointcut 메서드는 "완전한클래스이름.메서드이름"을 입력
- ☐ @Pointcut 메서드를 사용할 때 주의 점은 메서드의 접근제어가 그대로 적용된다는 점이다.
- ☐ @Pointcut 메서드는 @Aspect 기반의 Aspect 구현뿐만 아니라 XML 스키마의 pointcut 속성 값으로도 사용할 수 있다.

■ XML 설정에서 사용했던 getPointcut과 allPointcut을 어노테이션으로 변경한 예

```

1 package tommy.spring.web.common;
2 import org.aspectj.lang.annotation.Pointcut;
3 import org.springframework.stereotype.Service;
4 @Service
5 public class LogAdvice {
6     @Pointcut("execution(* tommy.spring.web..*Impl.*(..))")
7     public void allPointcut() {
8     }
9     @Pointcut("execution(* tommy.spring.web..*Impl.get*(..))")
10    public void getPointcut() {
11    }
12 }
```

③ 어드바이스 설정

- ☐ 어드바이스 클래스에는 횡단관심에 해당하는 어드바이스 메서드가 구현되어 있다. 어드바이스의 동작시점은 XML과 마찬가지로 다섯 가지가 제공된다. 이때 반드시 어드바이스가 결합된 포인트컷을 참조해야 한다. 포인트컷을 참조하는 방법은 어드바이스 어노테이션 뒤에 괄호를 추가하고 포인트컷 참조메서드를 지정하면 된다.

■ 어드바이스 설정의 예

```

1 package tommy.spring.web.common;
2 import org.aspectj.lang.annotation.Before;
```

```

3 import org.aspectj.lang.annotation.Pointcut;
4 import org.springframework.stereotype.Service;
5 @Service
6 public class LogAdvice {
7     @Pointcut("execution(* tommy.spring.web..*Impl.*(..))")
8     public void allPointcut() {
9     }
10    @Before("allPointcut()")
11    public void printLog() {
12        System.out.println("[공통 로그] : 비즈니스 로직 수행 전 동작");
13    }
14 }

```

□ 어드바이스 동작 시점

| 종 류 | 설 명 |
|-----------------|---|
| @Before | 대상 객체의 메서드 호출 전에 공통 기능을 실행한다. |
| @AfterReturning | 대상 객체의 메서드가 예외 없이 실행한 이후에 공통기능을 실행한다. |
| @AfterThrowing | 대상 객체의 메서드를 실행하는 도중 예외가 발생한 경우에 공통기능을 실행한다. try~catch 블록에서 catch 블록에 해당함. |
| @After | 대상 객체의 메서드를 실행하는 도중에 예외가 발생했는지의 여부와 상관없이 메서드 실행 후 공통 기능을 실행한다. (try~catch~finally)의 finally와 비슷하다 |
| @Around | 대상 객체의 메서드 실행 전 후 또는 예외 발생 시점에 공통 기능을 실행하는데 사용된다. |

④ 애스팩트 설정

- AOP 설정에서 가장 중요한 애스팩트는 @Aspect를 이용하여 설정한다. 애스팩트는 포인트컷과 어드바이스의 결합이다. 따라서 @Aspect가 설정된 애스팩트 객체에는 반드시 포인트컷과 어드바이스를 결합하는 설정이 있어야 한다.

```

1 package tommy.spring.web.common;
2 import org.aspectj.lang.annotation.Aspect;
3 import org.aspectj.lang.annotation.Before;
4 import org.aspectj.lang.annotation.Pointcut;
5 import org.springframework.stereotype.Service;
6 @Service
7 @Aspect // Asepct = Pointcut + Advice
8 public class LogAdvice {
9     @Pointcut("execution(* tommy.spring.web..*Impl.*(..))")
10    public void allPointcut() {           포인트컷
11    }                                     +
12    @Before("allPointcut()")             어드바이스
13    public void printLog() {
14        System.out.println("[공통 로그] : 비즈니스 로직 수행 전 동작");
15    }
16 }

```

7. 어드바이스 동작 실습

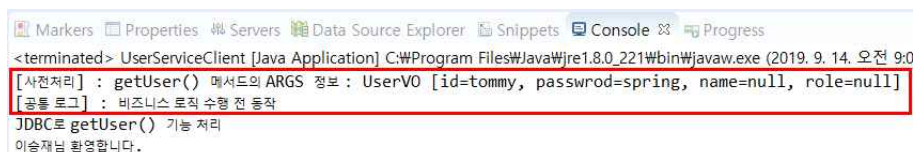
□ 지금부터 이전에 XML 설정했던 각 어드바이스를 어노테이션으로 변경하여 실습해 보자.

① Before Advice

■ BeforeAdvice 클래스 수정

```
1 package tommy.spring.web.common;
2 import org.aspectj.lang.JoinPoint;
3 import org.aspectj.lang.annotation.Aspect;
4 import org.aspectj.lang.annotation.Before;
5 import org.aspectj.lang.annotation.Pointcut;
6 import org.springframework.stereotype.Service;
7 @Service
8 @Aspect
9 public class BeforeAdvice {
10     @Pointcut("execution(* tommy.spring.web..*Impl.*(..))")
11     public void allPointcut() {
12     }
13     @Before("allPointcut()")
14     public void beforeLog(JoinPoint joinPoint) {
15         String method = joinPoint.getSignature().getName();
16         Object[] args = joinPoint.getArgs();
17         System.out.println("[사전처리] : " + method +
18             "() 메서드의 ARGS 정보 : " + args[0].toString());
19     }
20 }
```

■ UserServiceClient 클래스 실행 및 결과 확인



<terminated> UserServiceClient [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (2019. 9. 14. 오전 9:00)
[사전처리] : getUser() 메서드의 ARGS 정보 : UserVO [id=tommy, password=spring, name=null, role=null]
[공통 로그] : 비즈니스 로직 수행 전 동작
JDBC로 getUser() 가능 처리
이승재님 환영합니다.

□ 결론 : 정상적으로 비즈니스 로직 수행 전에 @Before Advice가 동작하는 것을 볼 수 있다. 그런데 앞에서 샘플로 작성한 LogAdvice 클래스의 어노테이션도 동작하는 것을 알 수 있다. 즉 <aop:aspectj-autoproxy> 설정만으로도 스프링 컨테이너가 모든 어노테이션을 알아서 처리해 준다는 것을 확인할 수 있다. 앞으로 예제에서는 LogAdvice 클래스의 어노테이션을 주석처리한 후 실습해 보도록 하자.

② After Returning Advice

■ AfterReturningAdvice 클래스 수정

```
1 package tommy.spring.web.common;
2 import org.aspectj.lang.JoinPoint;
3 import org.aspectj.lang.annotation.AfterReturning;
4 import org.aspectj.lang.annotation.Aspect;
5 import org.aspectj.lang.annotation.Pointcut;
```

```

6 import org.springframework.stereotype.Service;
7 import tommy.spring.web.user.UserVO;
8 @Service
7 @Aspect
8 public class AfterReturningAdvice {
9     @Pointcut("execution(* tommy.spring.web..*Impl.get*(..))")
10    public void getPointcut() {
11    }
12    @AfterReturning(pointcut = "getPointcut()", returning = "returnObj")
13    public void afterLog(JoinPoint joinPoint, Object returnObj) {
14        String method = joinPoint.getSignature().getName();
15        if (returnObj instanceof UserVO) {
16            UserVO user = (UserVO) returnObj;
17            if (user.getRole().equals("admin")) {
18                System.out.println(user.getName() + " 로그인(Admin)");
19            }
20        }
21        System.out.println("[사후처리] : " + method +
22                                "() 메서드 리턴값 : " + returnObj.toString());
23    }
24 }

```

■ UserServiceClient 클래스 실행 및 결과 확인

```

<terminated> UserServiceClient [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (2019. 9. 14. 오전 9:14)
[사전처리] : getUser() 메서드의 ARGS 정보 : UserVO [id=tommy, password=spring, name=null, role=null]
JDBC로 getUser() 가능 처리
이승재 로그인(Admin)
[사후처리] : getUser() 메서드 리턴값 : UserVO [id=tommy, password=spring, name=이승재, role=admin]
이승재님 환영합니다.

```

③ After Throwing Advice

■ AfterThrowingAdvice 클래스 수정

```

1 package tommy.spring.web.common;
2 import org.aspectj.lang.JoinPoint;
3 import org.aspectj.lang.annotation.AfterThrowing;
4 import org.aspectj.lang.annotation.Aspect;
5 import org.aspectj.lang.annotation.Pointcut;
6 import org.springframework.stereotype.Service;
7 @Service
8 @Aspect
7 public class AfterThrowingAdvice {
8     @Pointcut("execution(* tommy.spring.web..*Impl.*(..))")
9     public void allPointcut() {
10    }
11    @AfterThrowing(pointcut = "allPointcut()", throwing = "e")
12    public void exceptionLog(JoinPoint joinPoint, Exception e) {
13        String method = joinPoint.getSignature().getName();
14        System.out.println("[예외처리] : " + method +
15                                "() 메서드 수행 중 발생한 예외 메시지 : " + e.getMessage());

```

| | |
|----|---|
| 15 | } |
| 16 | } |

■ BoardServiceImpl 클래스의 insertBoard() 수정

| | |
|----|--|
| | <!-- 상단부분 생략 --> |
| 1 | @Service("boardService") |
| 2 | public class BoardServiceImpl implements BoardService { |
| 3 | @Autowired |
| 4 | private BoardDAO boardDAO; |
| 5 | @Override |
| 6 | public void insertBoard(BoardVO vo) { |
| 7 | if(vo.getSeq() == 0) { |
| 8 | throw new IllegalArgumentException("0번 글은 등록할 수 없습니다."); |
| 9 | } |
| 10 | boardDAO.insertBoard(vo); |
| 11 | } |
| | <!-- 하단부분 생략 --> |

■ BoardServiceClient 클래스 실행 및 결과 확인

| | |
|--|--|
| Markers Properties Servers Data Source Explorer Snippets Console Progress | |
| <terminated> BoardServiceClient [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (2019. 9. 14. 오전 10:31:12) | |
| [사후처리] : insertBoard() 메서드의 ARGS 정보 : BoardVO [seq=0, title=임시제목, writer=홍길동, content=테스트 내용, regDate=null, cnt=0] | |
| Exception in thread "main" [예외처리] : insertBoard() 메서드 수행 중 발생된 예외 메세지 : 0번 글은 등록할 수 없습니다. | |
| java.lang.IllegalArgumentException: 0번 글은 등록할 수 없습니다. | |
| at tommy.spring.web.board.impl.BoardServiceImpl.insertBoard(BoardServiceImpl.java:19) | |
| at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method) | |
| at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source) | |
| at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source) | |
| at java.lang.reflect.Method.invoke(Unknown Source) | |

④ After Advice

■ AfterAdvice 클래스 수정

| | |
|----|---|
| 1 | package tommy.spring.web.common; |
| 2 | import org.aspectj.lang.annotation.After; |
| 3 | import org.aspectj.lang.annotation.Aspect; |
| 4 | import org.aspectj.lang.annotation.Pointcut; |
| 5 | import org.springframework.stereotype.Service; |
| 6 | @Service |
| 7 | @Aspect |
| 8 | public class AfterAdvice { |
| 9 | @Pointcut("execution(* tommy.spring.web..*Impl.*(..))") |
| 10 | public void allPointcut() { |
| 11 | } |
| 12 | @After("allPointcut()") |
| 13 | public void finallyLog() { |
| 14 | System.out.println("[사후처리 : 비즈니스 로직 수행 후 무조건 동작]; |
| 15 | } |
| 16 | } |

■ BoardServiceClient 클래스 실행 및 결과 확인

```

Markers Properties Servers Data Source Explorer Snippets Console Progress
<terminated> BoardServiceClient [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (2019. 9. 14. 오전 10:35:55)
[사전처리] : insertBoard() 메서드의 ARGS 정보 : BoardVO [seq=0, title=임시제목, writer=홍길동, content=테스트 내용, regDate=null, cnt=0]
Exception in thread "main" [예외처리] : insertBoard() 메서드 수행 중 발생한 예외 메시지 : 0번 글은 등록할 수 없습니다.
[사후처리 : 비즈니스 로직 수행 후 무조건 동작]
java.lang.IllegalArgumentException: 0번 글은 등록할 수 없습니다.
    at tommy.spring.web.board.impl.BoardServiceImpl.insertBoard(BoardServiceImpl.java:19)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
    at java.lang.reflect.Method.invoke(Unknown Source)

```

⑤ Around Advice

■ 우선 BoardServiceImpl 클래스의 insertBoard의 붉은색 부분을 다시 주석처리하자.

```

<!-- 상단부분 생략 -->
1  @Service("boardService")
2  public class BoardServiceImpl implements BoardService {
3      @Autowired
4      private BoardDAO boardDAO;
5      @Override
6      public void insertBoard(BoardVO vo) {
7          //if(vo.getSeq() == 0) {
8              //    throw new IllegalArgumentException("0번 글은 등록할 수 없습니다.");
9          //}
10         boardDAO.insertBoard(vo);
11     }
<!-- 하단부분 생략 -->

```

■ AroundAdvice 클래스 수정

```

1  package tommy.spring.web.common;
2  import org.aspectj.lang.ProceedingJoinPoint;
3  import org.aspectj.lang.annotation.Around;
4  import org.aspectj.lang.annotation.Aspect;
5  import org.aspectj.lang.annotation.Pointcut;
6  import org.springframework.stereotype.Service;
7  import org.springframework.util.StopWatch;
8  @Service
9  @Aspect
10 public class AroundAdvice {
11     @Pointcut("execution(* tommy.spring.web.*Impl.*(..))")
12     public void allPointcut() {
13     }
14     @Around("allPointcut()")
15     public Object aroundLog(ProceedingJoinPoint joinPoint) throws Throwable {
16         String method = joinPoint.getSignature().getName();
17         StopWatch stopWatch = new StopWatch();
18         stopWatch.start();
19         Object returnObj = joinPoint.proceed();
20         stopWatch.stop();
21         System.out.println(method + "() 메서드 수행에 걸린 시간 : " +
22             stopWatch.getTotalTimeMillis() + "(ms)초");
23     }
24 }

```

| | |
|----|-------------------|
| 20 | return returnObj; |
| 21 | } |
| 22 | } |

■ UserServiceClient 클래스 실행 및 결과 확인

```

<terminated> UserServiceClient [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (2019. 9. 14. 오전 10
[사전처리] : getUser() 메서드의 ARGS 정보 : UserVO [id=tommy, passwrod=spring, name=null, role=null]
JDBC로 getUser() 가능 처리
getUser() 메서드 수행에 걸린 시간 : 2268(ms)초
이승재 로그인(Admin)
[사후처리] : getUser() 메서드 리턴값 : UserVO [id=tommy, passwrod=spring, name=이승재, role=admin]
[사후처리] : 비즈니스 로직 수행 후 무조건 동작
이승재님 환영합니다.

```

⑥ 외부 Pointcut 참조하기

- ☐ 어노테이션으로 설정을 변경하면서 어드바이스 클래스마다 포인트컷을 포함시켰다. 이렇다보니 비슷하거나 같은 포인트컷이 중복되는 문제가 발생하였다. 스프링은 이러한 문제를 해결하고자 **포인트컷을 외부에 독립된 클래스에 따로 설정**할 수 있도록 한다.

■ PointcutCommon 클래스 작성하기

| | |
|----|--|
| 1 | package tommy.spring.web.common; |
| 2 | import org.aspectj.lang.annotation.Aspect; |
| 3 | import org.aspectj.lang.annotation.Pointcut; |
| 4 | @Aspect |
| 5 | public class PointcutCommon { |
| 6 | @Pointcut("execution(* tommy.spring.web..*Impl.*(..))") |
| 7 | public void allPointcut() { |
| 8 | } |
| 9 | @Pointcut("execution(* tommy.spring.web..*Impl.get*(..))") |
| 10 | public void getPointcut() { |
| 11 | } |
| 12 | } |

- 위와 같이 작성했을 경우 BeforeAdvice 클래스는 아래와 같이 변경할 수 있다.

| | |
|---|--|
| 1 | public class BeforeAdvice { |
| 2 | @Before("PointcutCommon.allPointcut()") |
| 3 | public void beforeLog(JoinPoint joinPoint) { |
| | <!-- 하단 부분 생략 --> |

■ AfterReturningAdvice 클래스 변경

| | |
|---|---|
| 1 | public class AfterReturningAdvice { |
| 2 | @AfterReturning(pointcut = "PointcutCommon.getPointcut()", returning = "returnObj") |
| 3 | public void afterLog(JoinPoint joinPoint, Object returnObj) { |
| | <!-- 하단 부분 생략 --> |

■ AfterThrowingAdvice 클래스 변경


```

1 public class AfterThrowingAdvice {
2     @AfterThrowing(pointcut = "PointcutCommon.allPointcut()", throwing = "e")
3     public void exceptionLog(JoinPoint joinPoint, Exception e) {
<!-- 하단 부분 생략 -->

```

■ AfterAdvice 클래스 변경

```

1 public class AfterAdvice {
2     @After("PointcutCommon.allPointcut()")
3     public void finallyLog() {
<!-- 하단 부분 생략 -->

```

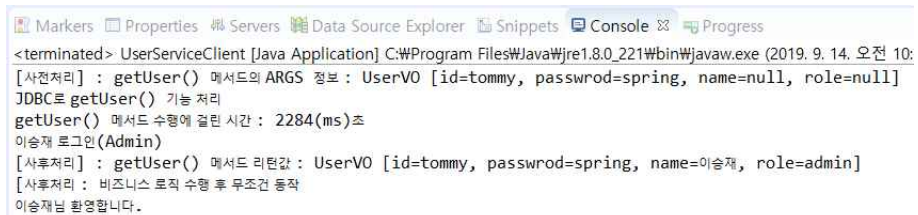
■ AroundAdvice 클래스 변경

```

1 public class AroundAdvice {
2     @Around("PointcutCommon.allPointcut()")
3     public Object aroundLog(ProceedingJoinPoint joinPoint) throws Throwable {
<!-- 하단 부분 생략 -->

```

■ 모두 변경하였으면 UserServiceClient 클래스 실행 및 결과 확인



```

<terminated> UserServiceClient [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (2019. 9. 14. 오전 10:
[사전처리] : getUser() 메서드의 ARGS 정보 : UserVO [id=tommy, password=spring, name=null, role=null]
JDBC로 getUser() 가능 처리
getUser() 메서드 수행에 걸린 시간 : 2284(ms)초
이송자 로그인(Admin)
[사후처리] : getUser() 메서드 리턴값 : UserVO [id=tommy, password=spring, name=이송자, role=admin]
[사후처리] : 비즈니스 로직 수행 후 무조건 동작
이송자님 환영합니다.

```

- 결론 : 예외가 발생되지 않으므로 After Throwing Advice 부분을 제외하고 나머지 Advice는 정상적으로 동작되는 것을 확인할 수 있다. After Throwing Advice를 확인하고 싶다면 BoardServiceImpl 클래스의 insertBoard() 메서드를 수정하고 실행 해 보면 된다.