1. AOP What?
  1)Business Component 개발에서 가장 중요한 2가지 원칙
    -낮은 결합도
    -높은 응집도
  2)Spring의 의존성 주입(DI)을 이용하면 Business Component를 구성하는 객체들의 결합도를 떨어뜨릴 수 있어서 의존 관계를 쉽게 변경할 수 있다.
  3)반면, 응집도와 관련이 있는 기능은 바로, AOP(Aspect Oriented Programming)이다.

  4)Application은 다양한 공통 기능을 필요로 한다.
  5)Logging과 같은 기본적인 기능에서부터 Transaction이나 보안과 같은 기능에 이르기까지 Application 전반에 걸쳐 적용되는 공통 기능이 존재한다.
  6)공통기능은 Application의 핵심 Business logic과는 구분되고 핵심기능을 도와 주는 부가적인 기능(logging, 보안 등)이다.
  7)핵심 Business 기능과 구분하기 위해 공통 기능을 공통 관심 사항(Cross-cutting Concern)이라고 표현한다.
  8)핵심 Logic(Biz logic을 포함하는 기능)을 핵심 관심 사항(Core Concern)이라고 표현한다.
    -핵심기능
      --계좌 이체, 대출 승인, 이자 계산
    -부가기능
      --Logging, 보안, transaction
    -그림 참조
    http://dev.anyframejava.org/docs/anyframe/plugin/foundation/4.6.0/reference/html/ch05.html

  9)객체지향의 기본 원칙(OOP)을 적용하면서도 핵심기능에서 부가기능을 분리해서 모듈화하는 것은 매우 어렵다.
  10)Programming에서 공통적인 기능을 모든 module에 적용하기 위한 방법으로 상속이 있다.
  11)하지만 Java는 다중상속을 하지 않기 때문에 다양한 module에 상속 기법을 통한 공통 기능 부여에는 한계가 있다.
  12)AOP는 핵심기능과 공통 기능을 분리시켜놓고, 공통 기능을 필요로 하는 핵심 기능들에서 사용하는 방식이다.
  13)AOP(Aspect Oriented Programming)은 문제를 바라보는 관점(시점)을 기준으로 Programming하는 기법이다.
  14)분리한 부가기능(공통 기능)을 Aspect라는 독특한 module 형태로 만들어서 설계하고 개발하는 방법이다.
  15)기본적인 개념은 공통 관심 사항을 구현한 Code를 핵심 Logic을 구현한 Code 안에 삽입한다는 것이다.
  16)OOP를 적용하여도 핵심기능에서 부가기능을 쉽게 분리된 module로 작성하기 어려운 문제점을 AOP가 해결해 준다고 볼 수 있다.
  17)즉, AOP는 부가기능을 Aspect로 정의하여, 핵심기능에서 부가기능을 분리함으로써 핵심 기능을 설계하고 구현할 때 객체지향적인 가치를 지킬 수 있도록 도와주는 개념이다.

  18)AOP 기법에서는 핵심 Logic을 구현한 code에서 공통 기능을 직접적으로 호출하지 않는다.
  19)핵심 Logic을 구현한 Code를 Compile하거나, Compile된 Class를 Loading하거나, Loading한 Class의 객체를 생성할 때 AOP가 적용되어 핵심 logic 구현 code안에 공통 기능이 삽입된다.
  20)AOP에서는 AOP library가 공통 기능을 알맞게 삽입해주기 때문에 개발자는 게시글 쓰기나 목록 읽기와 같은 핵심 Logic을 구현할 때 Transaction 적용이나 보안검사와 같은 공통 기능을 처리하기 위한 code를 핵심 logic code에 삽입할 필요가 없다.
  21)핵심 Logic을 구현한 Code에 공통 기능 관련 Code가 포함되어 있지 않기 때문에 적용해야 할 공통 기능이 변경되더라도 핵심 Logic을 구현한 code를 변경할 필요가 없다.
  22)단지, 공통 기능 code를 변경한 뒤 핵심 Logic 구현 code에 적용만 하면 된다.
  23)AOP 개념을 적용하면 핵심기능 Code 사이에 침투된 부가기능을 독립적인 Aspect로 구분해 낼 수 있다.
  24)구분된 부가기능 Aspect를 runtime 시에 필요한 위치에 동적으로 참여하게 할 수 있다.
  25)AspectJ Homepage : https://www.eclipse.org/aspectj/

2. BeforeAop
  1)Project 생성하기
    -Package Explorer > right-click > New > Java Project
    -Project name : BeforeAop > Finish
  2)Maven Project로 변환
  3)Spring Project로 변환
  4)Version Check

```
5)pom.xml update
  <dependencies>
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>5.2.7.RELEASE</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-api -->
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-api</artifactId>
      <version>5.6.2</version>
      <scope>test</scope>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-test -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-test</artifactId>
      <version>5.2.7.RELEASE</version>
      <scope>test</scope>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->
    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
      <version>1.18.12</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>

6)Config folder 생성
  -Project > right-click > New > Source Folder
  -Folder name : config  > Finish

7)applicationContext.xml 생성
  -config > righti-click > New > Spring Bean Configuration File
  -File name : applicationContext.xml > Finish
  -Namespace tab > context check

    <context:component-scan base-package="com.example" />

8)package 생성
  -com.example.vo
    package com.example.vo;

    import lombok.AllArgsConstructor;
    import lombok.Getter;
    import lombok.Setter;
    import lombok.ToString;

    @Setter
    @Getter
    @AllArgsConstructor
    @ToString
    public class MemberVO {
      private String userid, name, city, gender;
      private int age;
```

```
105       }
106
107     -com.example.dao
108       package com.example.dao;
109
110       import java.util.List;
111
112       import org.springframework.stereotype.Repository;
113
114       import com.example.vo.MemberVO;
115
116       import lombok.extern.slf4j.Slf4j;
117
118       @Repository("memberDao")
119       public class MemberDao {
120         public void insertMember(MemberVO memberVo) {
121           System.out.println("called insertMember()");
122         }
123
124         public MemberVO selectMember(String userid) {
125           System.out.println("called selectMember()");
126           return null;
127         }
128
129         public List<MemberVO> select(){
130           System.out.println("called selectAllMember()");
131           return null;
132         }
133
134         public void updateMember(MemberVO memberVo) {
135           System.out.println("called updateMember()");
136         }
137
138         public void deleteMember(String userid) {
139           System.out.println("called deleteMember()");
140         }
141       }
142
143     -com.example.common
144       package com.example.common;
145
146       public class LogAdvice {
147         public void printLog() {
148           System.out.println("[Common Log] 비즈니스 로직 수행 전 동작");
149         }
150       }
151
152     -com.example.service
153       package com.example.service;
154
155       import java.util.List;
156
157       import org.springframework.beans.factory.annotation.Autowired;
158       import org.springframework.stereotype.Service;
159
160       import com.example.common.LogAdvice;
161       import com.example.dao.MemberDao;
162       import com.example.vo.MemberVO;
```

```
163
164         @Service("memberService")
165         public class MemberService {
166           @Autowired
167           private MemberDao memberDao;
168           private LogAdvice log;
169
170           public MemberService() {
171             log = new LogAdvice();
172           }
173
174           public void create(MemberVO memberVo) {
175             this.log.printLog();
176             this.memberDao.insertMember(memberVo);
177           }
178
179           public MemberVO select(String userid) {
180             this.log.printLog();
181             return this.memberDao.selectMember(userid);
182           }
183
184           public List<MemberVO> select(){
185             this.log.printLog();
186             return this.memberDao.select();
187           }
188
189           public void update(MemberVO memberVo) {
190             this.log.printLog();
191             this.memberDao.updateMember(memberVo);
192           }
193
194           public void delete(String userid) {
195             this.log.printLog();
196             this.memberDao.deleteMember(userid);
197           }
198         }
199
200     9)TestApp.java 생성
201       -com.example.test package
202       -com.example.test > New > JUnit Test Case
203       -Select New JUnit Jupiter test
204       -Name : TestApp > Finish
205
206         package com.example.test;
207
208         import org.junit.jupiter.api.Test;
209         import org.junit.jupiter.api.extension.ExtendWith;
210         import org.springframework.beans.factory.annotation.Autowired;
211         import org.springframework.test.context.ContextConfiguration;
212         import org.springframework.test.context.junit.jupiter.SpringExtension;
213
214         import com.example.service.MemberService;
215         import com.example.vo.MemberVO;
216
217         @ExtendWith(SpringExtension.class)
218         @ContextConfiguration(locations = {"classpath:applicationContext.xml"})
219         class TestApp {
220           @Autowired
```

```
221         private MemberService memberService;
222
223         @Test
224         void test() {
225           MemberVO memberVo = new MemberVO("jimin", "한지민", "Seoul", "female", 24);
226           memberService.create(memberVo);
227           memberService.select("jimin");
228           memberService.select();
229           memberService.update(memberVo);
230           memberService.delete("jimin");
231         }
232       }
233
234   10)TestApp 실행
235     -TestApp > right-click > Run As > JUnit Test
236
237       Greenbar
238       [Common Log] 비즈니스 로직 수행 전 동작
239       called insertMember()
240       [Common Log] 비즈니스 로직 수행 전 동작
241       called selectMember()
242       [Common Log] 비즈니스 로직 수행 전 동작
243       called selectAllMember()
244       [Common Log] 비즈니스 로직 수행 전 동작
245       called updateMember()
246       [Common Log] 비즈니스 로직 수행 전 동작
247       called deleteMember()
248
249   11)Service 객체가 생성될 때, 생성자에서 LogAdvice 객체도 같이 생성된다.
250   12)그리고 각 Business method에서 Business Logic을 수행하기 전에 LogAdvice의 printLog() method
      를 호출하기만 하면 된다.
251   13)이후에 공통 기능을 수행할 때는 LogAdvice class의 printLog() method만 수정하면 되므로 관리가 편해졌
      다.
252   14)하지만, 이렇게 작성된 프로그램은 Service객체와 LogAdvice 객체가 소스코드에서 강하게 결합되어 있어서,
      LogAdvice class를 다른 Class로 변경해야 하거나 공통 기능에 해당하는 printLog() method의 Signature가
      변경되는 상황에서는 유연하게 대처할 수 없다.
253   15)다음과 같이 LogAdvice class를 대체할 Log4jAdvice class를 생성하기로 한다.
254   16)Method의 이름도 printLogging()으로 변경하기로 한다.
255
256     package com.example.common;
257
258     public class Log4jAdvice {
259       public void printLogging() {
260         System.out.println("[Common Log4j] 비스니스 로직 수행 전 동작");
261       }
262     }
263
264   17)LogAdvice에서 Log4jAdvice로 변경됐으므로 Service에서 일일히 변경해야 한다.
265
266     package com.example.service;
267
268     import java.util.List;
269
270     import org.springframework.beans.factory.annotation.Autowired;
271     import org.springframework.stereotype.Service;
272
273     import com.example.common.Log4jAdvice;
274     import com.example.common.LogAdvice;
```

```
import com.example.dao.MemberDao;
import com.example.vo.MemberVO;

@Service("memberService")
public class MemberService {
  @Autowired
  private MemberDao memberDao;
  private Log4jAdvice log;

  public MemberService() {
    log = new Log4jAdvice();
  }

  public void create(MemberVO memberVo) {
    this.log.printLogging();
    this.memberDao.insertMember(memberVo);
  }

  public MemberVO select(String userid) {
    this.log.printLogging();
    return this.memberDao.selectMember(userid);
  }

  public List<MemberVO> select(){
    this.log.printLogging();
    return this.memberDao.select();
  }

  public void update(MemberVO memberVo) {
    this.log.printLogging();
    this.memberDao.updateMember(memberVo);
  }

  public void delete(String userid) {
    this.log.printLogging();
    this.memberDao.deleteMember(userid);
  }
}
```

18)결국 Advice class가 LogAdvice에서 Log4jAdvice로 변경되는 순간 Service class의 생성자도 수정하고 method도 printLog()에서 printLoggin()으로 모두 변경해야 한다.
19)TestApp을 실행해 본다.

```
[Common Log4j] 비스니스 로직 수행 전 동작
called insertMember()
[Common Log4j] 비스니스 로직 수행 전 동작
called selectMember()
[Common Log4j] 비스니스 로직 수행 전 동작
called selectAllMember()
[Common Log4j] 비스니스 로직 수행 전 동작
called updateMember()
[Common Log4j] 비스니스 로직 수행 전 동작
called deleteMember()
```

20)물론 변경된 사항이 잘 적용되는 것을 알 수 있다.
21)정리하면, OOP처럼 module화가 뛰어난 언어를 사용하여 개발하더라도 공통 module에 해당하는 Advice class 객체를 생성하고 공통 method를 호출하는 코드가 Business method에 있다면, 핵심 관심(Service)과 횡단 관심(LogAdvice)을 완벽하게 분리할 수 없다.

22)그래서 Spring의 AOP는 이런 OOP의 한계를 극복할 수 있도록 한다.

3. StartAop project
1)위 BeforeAop project를 그대로 사용한다.
2)일단 Service 객체에서 Log4jAdvice를 떼어내고, printLoggin()도 모두 지운다.

```
package com.example.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.example.dao.MemberDao;
import com.example.vo.MemberVO;

@Service("memberService")
public class MemberService {
  @Autowired
  private MemberDao memberDao;

  public void create(MemberVO memberVo) {
    this.memberDao.insertMember(memberVo);
  }

  public MemberVO select(String userid) {
    return this.memberDao.selectMember(userid);
  }

  public List<MemberVO> select(){
    return this.memberDao.select();
  }

  public void update(MemberVO memberVo) {
    this.memberDao.updateMember(memberVo);
  }

  public void delete(String userid) {
    this.memberDao.deleteMember(userid);
  }
}
```

3)이제 Service객체와 LogAdvice 혹은 Log4jAdvice와는 아무 상관이 없는 객체가 되었다.
4)pom.xml에 AOP library 추가
-mvnrepository.com에서 'aspectj'로 검색한다.
--AspectJ Weaver

```
<!-- https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->
<dependency>
   <groupId>org.aspectj</groupId>
   <artifactId>aspectjweaver</artifactId>
   <version>1.9.5</version>
</dependency>
```

--AspectJ Runtime

```
<!-- https://mvnrepository.com/artifact/org.aspectj/aspectjrt -->
<dependency>
```

```
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjrt</artifactId>
            <version>1.9.5</version>
        </dependency>

  -pom.xml에 붙여넣고 maven install 한다.

5)applicationContext.xml에 추가하기
  -Namespaces tab > aop check

    <bean id="log" class="com.example.common.LogAdvice" />
    <aop:config>
      <aop:pointcut expression="execution(* com.example.service..*Service.*(..))"
      id="allPointcut"/>
      <aop:aspect ref="log">
        <aop:before pointcut-ref="allPointcut" method="printLog"/>
      </aop:aspect>
    </aop:config>

6)TestApp 수행

  [Common Log] 비즈니스 로직 수행 전 동작
  called insertMember()
  [Common Log] 비즈니스 로직 수행 전 동작
  called selectMember()
  [Common Log] 비즈니스 로직 수행 전 동작
  called selectAllMember()
  [Common Log] 비즈니스 로직 수행 전 동작
  called updateMember()
  [Common Log] 비즈니스 로직 수행 전 동작
  called deleteMember()

7)이제는 Service를 건드리지 않고 단지 LogAdvice에서 Log4jAdvice로 변경하고 다시 TestApp을 수행한다.

  <context:component-scan base-package="com.example" />
  <bean id="log" class="com.example.common.Log4jAdvice" />
  <aop:config>
    <aop:pointcut expression="execution(* com.example.service..*Service.*(..))"
    id="allPointcut"/>
    <aop:aspect ref="log">
      <aop:before pointcut-ref="allPointcut" method="printLogging"/>
    </aop:aspect>
  </aop:config>

  [Common Log4j] 비스니스 로직 수행 전 동작
  called insertMember()
  [Common Log4j] 비스니스 로직 수행 전 동작
  called selectMember()
  [Common Log4j] 비스니스 로직 수행 전 동작
  called selectAllMember()
  [Common Log4j] 비스니스 로직 수행 전 동작
  called updateMember()
  [Common Log4j] 비스니스 로직 수행 전 동작
  called deleteMember()

8)분명 Service 객체를 수정하지 않고도 LogAdvice에서 Log4jAdvice로 변경할 수 있었다.
9)이 때, 핵심 관심 method와 횡단 관심 method 사이에서 소스상의 결합은 발생하지 않았다.
```

4. AOP 용어
1)Aspect : 여러 객체에 공통으로 적용되는 공통 관심 사항. 예)transaction이나 보안, logging 등...
-Advice와 pointcut을 합친 것이다.
-구현 하고자 하는 Cross-cutting Concern의 기능.
-Application의 module화 하고자 하는 부분

2)Target : 핵심 기능을 담고 있는 모듈로, Target은 부가기능을 부여할 대상이 된다.
-Advice를 받는 객체.
-Target은 우리가 작성한 class는 물론, 별도의 기능을 추가하고자 하는 third-party class가 될 수 있다.

3)Advice : 언제 공통관심 기능을 핵심 logic에 적용할지를 정의한다.
-즉, 부가기능을 정의한 code.
-Target에 제공할 부가기능을 담고 있는 module.
-PointCut에서 지정한 JoinPoint에서 실행(삽입)되어야할 code이다.
-Aspect의 실제 구현체
-예)'method를 호출하기 전'(언제)에 'transaction을 시작한다.'(공통기능) 기능을 적용한다는 것을 정의

4)JoinPoint : Advice를 적용해야 되는 지점
-Instance의 생성시점, method를 호출하는 시점, Exception이 발생하는 시점과 같이 application이 실행될 때 특정작업이 실행되는 시점을 의미한다.
-Aspect를 plugin 할 수 있는 application의 실행 지점
-즉, Target 객체가 구현한 interface의 모든 method는 JoinPoint가 된다.
-ex. field값 변경, method호출
-Spring에서는 method 호출만 해당

5)Pointcut : JoinPoint의 부분집합으로 실제로 Advice가 적용되는 JoinPoint 부분.
-Advice가 어떤 JoinPoint에 적용되어야 하는지 정의.
-명시적인 class의 이름, method의 이름이나 class나 method의 이름과 pattern이 일치하는 결합점을 지정 가능토록 해준다.
-Spring에서는 정규 표현식이나 AspectJ의 문법을 이용하여 정의한다.
-표현식은 execution으로 시작하고, method의 Signature를 비교하는 방법을 주로 이용

6)Weaving : Advice를 핵심 code에 적용하는 행위.
-공통 코드를 핵심 logic code에 삽입하는 것.
-AOP가 핵심기능(Target)의 code에 영향을 주지 않으면서 필요한 부가기능(Advice)을 추가할 수 있도록 해주는 핵심적인 처리과정
-Aspect를 Target 객체에 적용하여 새로운 proxy 객체를 생성하는 과정을 말한다.
-Aspect는 Target 객체의 지정된 JoinPoint에 엮인다.

7)즉, Aspect = Advice + PointCut 이다.

8)Aspect는 AOP의 기본 module

9)Aspect는 Singleton 형태의 객체로 존재한다.

10)Advisor = Advice + Pointcut
-Spring AOP에서만 사용되는 특별한 용어

11)그림참조
http://isstory83.tistory.com/90

5. 3 가지 Weaving 방식
1)Compile 시 : AspectJ에서 사용하는 방식
2)Class loading 시
3)Runtime 시 : Proxy를 이용. 핵심 logic을 구현한 객체에 직접 접근하는 것이 아니라 중간에 Proxy를 생성하여 Proxy를 통해서 핵심 logic을 구현한 객체에 접근

-Spring에서 AOP를 구현하는 방법 : Proxy를 이용한다.
-호출부(Client) --> Proxy(대행) --> Target(핵심기능)

6. Spring AOP의 특징
1)Spring은 Proxy 기반 AOP를 지원한다.
-Spring은 Target 객체에 대한 Proxy를 만들어 제공한다.
-Target을 감싸는 Proxy는 실행시간(Runtime)에 생성된다.
-Proxy는 Advice를 Target객체에 적용하면서 생성되는 객체이다.

2)Proxy가 호출을 intercept한다.
-Proxy는 Target 객체에 대한 호출을 가로챈 다음, Advice의 부가기능 logic을 수행하고 난 후에 Target의 핵심 기능 logic을 호출한다.(전처리 Advice)
-또는 Target의 핵심기능 logic method를 호출한 뒤에 부가기능(Advice)을 수행하는 경우도 있다.(후처리 Advice)

3)Spring AOP는 method JoinPoint만 지원한다.
-Spring은 동적 Proxy를 기반으로 AOP를 구현하기 때문에 method JoinPoint만 지원한다.
-즉, 핵심기능(Target)의 method가 호출되는 runtime 시점에만 부가기능(Advice)을 적용할 수 있다.
-반면에, AspectJ 같은 고급 AOP framework를 사용하면 객체의 생성, field값의 조회와 조작, static method 호출 및 초기화 등의 다양한 작업에 부가기능을 적용할 수 있다.

7. Spring에서 AOP 구현 방식
1)Spring API를 이용한 AOP 구현

2)XML schema 기반의 POJO class를 이용한 AOP구현 : Spring 2부터 사용
-부가기능을 제공하는 Advice class를 작성
-XML 설정 file에 <aop:config>를 이용해서 Aspect를 설정
-즉, Advice와 Pointcut을 설정

3)@Aspect annotation 기반의 AOP 구현
-@Aspect annotation을 이용해서 부가기능을 제공하는 Aspect class를 작성
-이때, Aspect class는 Advice를 구현하는 method와 Pointcut을 포함한다.
-XML 설정 file에 <aop:aspectj-autoproxy />를 설정

4)@Aspect annotation
-Aspect class를 선언할 때 @Aspect annotation을 사용한다.
-AspectJ 5 버전에 새롭게 추가된 annotaion이다.
-@Aspect annotaion을 이용할 경우 XML 설정 file에 Advice와 Pointcut을 설정하는 것이 아니라 class 내부에 정의할 수 있다.
-<aop:aspectj-autoproxy> tag를 설정file에 추가하면 @Aspect annotaion이 적용된 Bean을 Aspect로 사용 가능하다.

8. AspectJ와 Spring AOP library 설치
1)Runtime library 설치
-Maven Repository에서 'aspectj runtime'으로 검색
-aspectj runtime 1.8.10 버전을 pom.xml에 추가

```
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjrt</artifactId>
    <version>1.8.10</version>
</dependency>
```

2)AspectJ Weaver library 설치
-Maven Repository에서 'aspectj weaver'으로 검색

```
    -aspectj weaver 1.8.10 버전을 pom.xml에 추가

      <dependency>
          <groupId>org.aspectj</groupId>
          <artifactId>aspectjweaver</artifactId>
          <version>1.8.10</version>
      </dependency>

  3)Spring AOP library 설치
    -Maven Repository에서 'spring aop'으로 검색
    -aspectj weaver 4.3.9 버전을 pom.xml에 추가

      <dependency>
          <groupId>org.springframework</groupId>
          <artifactId>spring-aop</artifactId>
          <version>4.3.9.RELEASE</version>
      </dependency>

  4)AspectJ Runtime API 문서
    -Google에서 'aspectj runtime aip doc'로 검색
    -https://eclipse.org/aspectj/doc/released/runtime-api/index.html


9. Advice의 종류
  1)<aop:before>
    -Method 실행 전에 Advice실행
    -JoinPoint 앞에서 실행되는 Advice

  2)<aop:after-returning>
    -정상적으로 method 실행 후에 Advice 실행
    -JoinPoint method 호출이 정상적으로 종료된 뒤에 실행되는 Advice

  3)<aop:after-throwing>
    -Method 실행 중 exception 발생시 Advice 실행
    -try-catch의 catch와 비슷

  4)<aop:after>
    -Method 실행 중 exception 이 발생하여도 Advice 실행
    -try-catch-finally에서 finally와 비슷

  5)<aop:around>
    -Target의 method 실행 전/후 및 exception 발생시 Advice 실행
    -JoinPoint 앞과 뒤에서 실행되는 Advice


10. Advice를 정의하는 annotation
  1)@Before("pointcut")
    -Target 객체의 method가 실행되기 전에 호출되는 Advice
    -JoinPoint를 통해 parameter 정보를 참조할 수 있다.

  2)@After("pointcut")
    -Target 객체의 method가 정상 종료됐을 때와 예외가 발생했을 때 모두 호출되는 Advice
    -Return값이나 예외를 직접 전달 받을 수는 없다.

  3)@Around("pointcut")
    -Target 객체의 method가 호출되는 전 과정을 모두 담을 수 있는 가장 강력한 기능을 가진 Advice

  4)@AfterReturning(pointcut="", returning="")
```

609 -Target 객체의 method가 정상적으로 실행을 마친 후에 호출되는 Advice
610 -Return값을 참조할 때는 returning 속성에 Return값을 저장할 변수 이름을 지정해야 한다.
611
612 5)@AfterThrowing(pointcut="", throwing="")
613 -Target 객체의 method가 예외가 발생하면 호출되는 Advice
614 -발생된 예외를 참조할 때는 throwing 속성에 발생한 예외를 저장할 변수 이름을 지정해야 한다.
615
616
617 11. JoinPoint Interace
618 1)JoinPoint 는 Spring AOP 혹은 AspectJ에서 AOP가 적용되는 지점을 뜻한다.
619
620 2)해당 지점을 AspectJ에서 JointPoint라는 interface로 나타낸다.
621
622 3)Methos
623 -getArgs() : method argument 반환
624 -getThis() : Proxy 객체를 반환
625 -getTarget() : 대상 객체를 반환
626 -getSignature() : Advice되는 method의 설명(description)을 반환
627 -toString() : Advice되는 method의 설명을 출력
628
629 4)모든 Advice는 org.aspectj.lang.JoinPoint type의 parameter를 Advice method에 첫 번째 매개변수로 선언 가능
630
631 5)Around Advice는 JoinPoint의 하위 class인 ProceedingJoinPoint 타입의 parameter를 필수적으로 선언해야 함.
632
633 6)AspectJ Runtime API의 org.aspectj.lang의 JoinPoint interface 참조할 것
634
635 7)AspectJ Runtime API의 org.aspectj.lang의 ProceedingJoinPoint interface 참조할 것
636
637
638 12. AOP 설정
639 1)<aop:config> : AOP의 설정 정보임을 나타낸다.
640 2)<aop:aspect> : Aspect를 설정한다.
641 3)<aop:around pointcut="execution()"> : Around Advice와 Pointcut을 설정한다.
642 4)<aop:aspect> tag의 ref속성은 Aspect로서 기능을 제공할 bean을 설정할 때 사용함.
643 5)<aop:around> tag의 pointcut 속성의 execution 지시자(designator)는 Advice를 적용할 package, class, method를 표현할 때 사용됨.
644 6)com.example.service package 및 그 하위 package에 있는 모든 public method를 Pointcut으로 설정하고 있다.
645 7)UserServiceImpl의 public method가 호출될 때 PerformanceTraceAdvice Bean의 trace() method가 호출되도록 설정하고 있다.
646
647
648 13. Lab : XML schema 기반의 AOP 구현
649 1)In Java Perspective, New > Java Project >
650 Project Name : AopDemo
651 -JRE : Use default JRE (currently 'jdk 1.8.0_192')
652 -Maven Project Convert : project right-click > Configure > Convert to Maven Project
653 -Spring Project Convert : project right-click > Spring Tools > Add Spring Project Nature
654
655 2)Create Package : src/com.example
656
657 3)com.example.Student.java
658 package com.example;
659
660 public class Student{
661 private String name;

```
662         private int age;
663         private int grade;
664         private int classNum;
665         public String getName() {
666           return name;
667         }
668         public void setName(String name) {
669           this.name = name;
670         }
671         public int getAge() {
672           return age;
673         }
674         public void setAge(int age) {
675           this.age = age;
676         }
677         public int getGrade() {
678           return grade;
679         }
680         public void setGrade(int grade) {
681           this.grade = grade;
682         }
683         public int getClassNum() {
684           return classNum;
685         }
686         public void setClassNum(int classNum) {
687           this.classNum = classNum;
688         }
689         public void getStudentInfo(){
690           System.out.println("Name : " + this.name);
691           System.out.println("Age : " + this.age);
692           System.out.println("Grade : " + this.grade);
693           System.out.println("Class : " + this.classNum);
694         }
695       }
696
697     4)com.example.Worker.java
698       package com.example;
699
700       public class Worker {
701         private String name;
702         private int age;
703         private String job;
704         public String getName() {
705           return name;
706         }
707         public void setName(String name) {
708           this.name = name;
709         }
710         public int getAge() {
711           return age;
712         }
713         public void setAge(int age) {
714           this.age = age;
715         }
716         public String getJob() {
717           return job;
718         }
719         public void setJob(String job) {
```

```
720         this.job = job;
721       }
722       public void getWorkerInfo(){
723         System.out.println("Name : " + this.name);
724         System.out.println("Age : " + this.age);
725         System.out.println("Job : " + this.job);
726       }
727     }
728
729   5)Spring Context 설치
730     -Maven Repository 에서 'Spring Context'로 검색하여 dependency 추가하고 설치
731
732       <dependency>
733         <groupId>org.springframework</groupId>
734         <artifactId>spring-context</artifactId>
735         <version>4.3.20.RELEASE</version>
736       </dependency>
737
738     -'junit' 검색해서 추가
739
740       <dependency>
741          <groupId>junit</groupId>
742          <artifactId>junit</artifactId>
743          <version>4.12</version>
744          <scope>test</scope>
745       </dependency>
746
747     -pom.xml : aop code 추가
748     -Runtime library 설치
749       --Maven Repository에서 'aspectj runtime'으로 검색
750       --aspectj runtime 1.9.2 version을 pom.xml에 추가
751
752         <dependency>
753             <groupId>org.aspectj</groupId>
754             <artifactId>aspectjrt</artifactId>
755             <version>1.9.2</version>
756         </dependency>
757
758     -AspectJ Weaver library 설치
759       --Maven Repository에서 'aspectj weaver'으로 검색
760       --aspectj weaver 1.8.10 version을 pom.xml에 추가
761
762         <dependency>
763             <groupId>org.aspectj</groupId>
764             <artifactId>aspectjweaver</artifactId>
765             <version>1.9.2</version>
766         </dependency>
767
768     -Spring AOP library 설치
769       --Maven Repository에서 'spring aop'으로 검색
770       --aspectj weaver 4.3.9 version을 pom.xml에 추가
771
772         <dependency>
773             <groupId>org.springframework</groupId>
774             <artifactId>spring-aop</artifactId>
775             <version>4.3.20.RELEASE</version>
776         </dependency>
777
```

```
778     -Maven Install
779
780   6)com.example.LogAop.java
781     package com.example;
782
783     import org.aspectj.lang.ProceedingJoinPoint;
784
785     public class LogAop {
786     //joinpoint 객체를 전달 받을 때에는 반드시 첫번째 parameter여야 한다.
787       public Object loggerAop(ProceedingJoinPoint joinpoint) throws Throwable{
788         String signatureStr = joinpoint.getSignature().toShortString();
789         //Signature getSignature() : 호출되는 method에대한 정보를 구한다.
790         //cf)Object getTarget() : 대상 객체를 구한다.
791         //cf)Object [] getArgs() : parameter 목록을 구한다.
792
793         //toShortString() : method를 축약해서 표현한 문장을 구한다. method의 이름만 구한다.
794         //cf)toLongString() : 완전하게 표현된 문장. return type, method이름, parameter type 모두
795         //cf)getName() : method의 이름을 구한다.
796         System.out.println(signatureStr + " is start.");
797         long start = System.currentTimeMillis();
798
799         try{
800           Object obj = joinpoint.proceed(); //대상객체의 실제 method 호출
801           return obj;
802         }finally{
803           long end = System.currentTimeMillis();
804           System.out.println(signatureStr + " is finished.");
805           System.out.println(signatureStr + " 경과시간 : " + (end - start));
806         }
807       }
808     }
809
810   7)beans.xml 설정
811     -Project right-click > Build Path > Configure Build Path.. > Source tab
812     -Add Folder > Select AopDemo > Click Create New Folder... >
813     -Folder name : config > Finish > OK > Apply and Close
814
815     -config right-click > New > Spring Bean Configuration File > File name : beans.xml
816     -Namespace tab
817       --aop Check
818
819     <?xml version="1.0" encoding="UTF-8"?>
820     <beans xmlns="http://www.springframework.org/schema/beans"
821       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
822       xmlns:aop="http://www.springframework.org/schema/aop"
823       xsi:schemaLocation="http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans.xsd
824         http://www.springframework.org/schema/aop
            http://www.springframework.org/schema/aop/spring-aop-4.3.xsd">
825
826       <bean id="logAop" class="com.example.LogAop" />
827
828       <aop:config>
829         <aop:aspect id="logger" ref="logAop">
830           <aop:pointcut expression="within(com.example.*)*" id="publicMethod"/>
831           <aop:around method="loggerAop" pointcut-ref="publicMethod"/>
832         </aop:aspect>
833       </aop:config>
```

```
834
835         <!-- com.example아래 모든 class의 public Method를 호출할 때 LogAop의 loggerAop method가
            실행된다는 뜻 -->
836
837         <bean id="student" class="com.example.Student">
838           <property name="name" value="한지민" />
839           <property name="age" value="15" />
840           <property name="grade" value="3" />
841           <property name="classNum" value="5" />
842         </bean>
843
844         <bean id="worker" class="com.example.Worker">
845           <property name="name" value="설운도" />
846           <property name="age" value="50" />
847           <property name="job" value="개발자" />
848         </bean>
849       </beans>
850
851     8)Junit Test Case 추가
852       -src > com.example.test package 추가
853       -com.example.test > right-click > New > Junit Test Case
854       -Select New JUnit 4 test
855       -Name : TestApp > Finish
856
857         package com.example.test;
858
859         import static org.junit.Assert.assertNotNull;
860         import static org.junit.Assert.fail;
861
862         import org.junit.Before;
863         import org.junit.Test;
864         import org.springframework.context.ApplicationContext;
865         import org.springframework.context.support.GenericXmlApplicationContext;
866
867         public class TestApp {
868           private ApplicationContext ctx;
869
870           @Before
871           public void init() {
872             this.ctx = new GenericXmlApplicationContext("classpath:beans.xml");
873           }
874           @Test
875           public void test() {
876             assertNotNull(this.ctx);
877           }
878         }
879
880       -right-click > Run as > JUnit test
881       -green bar
882
883     9)test() 수정
884       @Test
885       public void test() {
886         Student student = this.ctx.getBean("student", Student.class);
887         student.getStudentInfo();
888
889         Worker worker = this.ctx.getBean("worker", Worker.class);
890         worker.getWorkerInfo();
```

```
891     }
892
893   10)결과
894     -right-click > Run as > JUnit test
895     -green bar
896     Student.getStudentInfo() is start.
897     Name : 한지민
898     Age : 15
899     Grade : 3
900     Class : 5
901     Student.getStudentInfo() is finished.
902     Student.getStudentInfo() 경과시간 : 14
903     Worker.getWorkerInfo() is start.
904     Name : 설운도
905     Age : 50
906     Job : 개발자
907     Worker.getWorkerInfo() is finished.
908     Worker.getWorkerInfo() 경과시간 : 7
909
910
911 14. Lab : XML schema 기반의 AOP 구현
912   1)In Java Perspective, New > Java Project >
913     Project Name : AopDemo1
914     -JRE : Use default JRE (currently 'jdk 1.8.0_192')
915     -Maven Project Convert : project right-click > Configure > Convert to Maven Project
916     -Spring Project Convert : project right-click > Spring Tools > Add Spring Project Nature
917
918   2)Spring Context 설치
919     -Maven Repository 에서 'Spring Context'로 검색하여 dependency 추가하고 설치
920
921       <dependency>
922         <groupId>org.springframework</groupId>
923         <artifactId>spring-context</artifactId>
924         <version>4.3.20.RELEASE</version>
925       </dependency>
926
927     -'junit' 검색해서 추가
928
929       <dependency>
930          <groupId>junit</groupId>
931          <artifactId>junit</artifactId>
932          <version>4.12</version>
933          <scope>test</scope>
934       </dependency>
935
936     -Run as > Maven Install
937
938   3)Create Package : src/com.example
939   4)src/com.example.Animal interface
940
941     package com.example;
942
943     public interface Animal {
944       void walwal();
945     }
946
947   5)src/com.example.TomDog.java
948
```

```
949     package com.example;
950
951     public class TomDog implements Animal {
952
953         public void walwal() {
954             System.out.println("I'm Tomdog...");
955         }
956     }
957
958   6)src/com.example.AnimalAOP.java
959
960     package com.example;
961
962     public class AnimalAOP {
963         public void beforeWalwal() {
964             System.out.println("Hi~ Dog...");
965         }
966
967         public void afterWalwal() {
968             System.out.println("Good Bye Dog...");
969         }
970     }
971
972   7)src/com.example.TestClient.java
973
974     package com.example;
975
976     import org.springframework.beans.factory.BeanFactory;
977     import org.springframework.context.support.FileSystemXmlApplicationContext;
978
979     public class TestClient {
980
981       public static void main(String[] args) {
982         BeanFactory bean = new
            FileSystemXmlApplicationContext("classpath:applicationContext.xml");
983
984         Animal tomdog = (Animal)bean.getBean("tomdog");
985         tomdog.walwal();
986       }
987     }
988
989   8)-pom.xml : aop code 추가
990     -Runtime library 설치
991       --Maven Repository에서 'aspectj runtime'으로 검색
992       --aspectj runtime 1.9.2 version을 pom.xml에 추가
993
994         <dependency>
995             <groupId>org.aspectj</groupId>
996             <artifactId>aspectjrt</artifactId>
997             <version>1.9.2</version>
998         </dependency>
999
1000    -AspectJ Weaver library 설치
1001      --Maven Repository에서 'aspectj weaver'으로 검색
1002      --aspectj weaver 1.8.10 version을 pom.xml에 추가
1003
1004        <dependency>
1005            <groupId>org.aspectj</groupId>
```

```
1006              <artifactId>aspectjweaver</artifactId>
1007              <version>1.9.2</version>
1008          </dependency>
1009
1010      -Spring AOP library 설치
1011        --Maven Repository에서 'spring aop'으로 검색
1012        --aspectj weaver 4.3.9 version을 pom.xml에 추가
1013
1014          <dependency>
1015              <groupId>org.springframework</groupId>
1016              <artifactId>spring-aop</artifactId>
1017              <version>4.3.20.RELEASE</version>
1018          </dependency>
1019
1020      -Maven Install
1021
1022    9)applicationContext.xml 설정
1023      -Project right-click > Build Path > Configure Build Path.. > Source tab
1024      -Add Folder > Select AopDemo > Click Create New Folder... >
1025      -Folder name : config > Finish > OK > Apply and Close
1026
1027      -config right-click > New > Spring Bean Configuration File > File name : beans.xml
1028      -Namespace tab
1029        --aop Check
1030
1031        <bean id="tomdog" class="com.example.TomDog" />
1032
1033        <aop:config>
1034          <aop:aspect ref="animalAOP">
1035            <aop:pointcut id="greeting"
1036              expression="execution(public * com.example.Animal.walwal(..))" />
1037            <aop:before pointcut-ref="greeting" method="beforeWalwal" />
1038            <aop:after-returning pointcut-ref="greeting"
1039              method="afterWalwal" />
1040          </aop:aspect>
1041        </aop:config>
1042
1043        <bean id="animalAOP" class="com.example.AnimalAOP" />
1044
1045    10)TestClient 실행 결과
1046
1047      Hi~ Dog...
1048      I'm Tomdog...
1049      Good Bye Dog...
1050
1051
1052  15. Lab
1053    1)In Spring Perspective, New > Spring Legacy Project > Simple Spring Maven
1054      -Project name: AopDemo2
1055      -Finish
1056
1057    2)pom.xml의 Dependencies tab에서
1058      -spring-context, spring-test, junit을 제외하고 모두 제거
1059      -pom.xml source에서
1060        <java.version>1.8</java.version> 수정
1061        <!-- Spring -->
1062        <spring-framework.version>4.3.20.RELEASE</spring-framework.version> 수정
1063
```

```
1064        <!-- Hibernate / JPA --> 제거
1065        <hibernate.version>4.2.1.Final</hibernate.version> 제거
1066        <!-- Logging --> 제거
1067        <logback.version>1.0.13</logback.version> 제거
1068        <slf4j.version>1.7.5</slf4j.version> 제거
1069
1070        <junit.version>4.12</junit.version> 수정
1071
1072      -Maven Install
1073      -Meven > Update Project
1074      -Project right-click > Properties > Project facet > Java version을 최신 jdk로 변경
1075
1076    3)Create Package : src/main/java/com.example
1077
1078    4)src/main/java/com.example.TV.java
1079
1080      public interface TV {
1081        void powerOn();
1082        void powerOff();
1083        void soundUp();
1084        void soundDown();
1085      }
1086
1087    5)src/main/java/com.example.SamsungTV.java
1088
1089      package com.javasoft;
1090
1091      public class SamsungTV implements TV{
1092        private String name;
1093        public SamsungTV(String name) {
1094          this.name = name;
1095          System.out.println(name + " : 방금 객체가 생성됐습니다.");
1096        }
1097
1098        public void powerOn() {
1099          System.out.println(name + " : 전원을 킨다.");
1100        }
1101        public void powerOff() {
1102          System.out.println(name + " : 전원을 끈다.");
1103        }
1104        public void soundUp() {
1105          System.out.println(name + " : 볼륨을 올린다.");
1106        }
1107        public void soundDown() {
1108          System.out.println(name + " : 볼륨을 내린다.");
1109        }
1110      }
1111
1112    6)src/main/java/com.example.LgTV.java
1113
1114      package com.javasoft;
1115
1116      public class LgTV implements TV{
1117        private String name;
1118        public LgTV(String name) {
1119          this.name = name;
1120          System.out.println(name + " : 방금 객체가 생성됐습니다.");
1121        }
```

```
1122        public void powerOn() {
1123          System.out.println(name + " : 전원을 킨다.");
1124        }
1125        public void powerOff() {
1126          System.out.println(name + " : 전원을 끈다.");
1127        }
1128        public void soundUp() {
1129          System.out.println(name + " : 볼륨을 올린다.");
1130        }
1131        public void soundDown() {
1132          System.out.println(name + " : 볼륨을 내린다.");
1133        }
1134      }
1135
1136    7)src/main/java/com.example.LogAdvice.java
1137
1138      package com.javasoft;
1139
1140      import org.aspectj.lang.JoinPoint;
1141
1142      public class LogAdvice {
1143        public void printLog(JoinPoint thisJoinPoint){
1144          System.out.println("[Core Concern] 수행전 로그하기");
1145        }
1146      }
1147
1148    8)src/main/resources/beans.xml
1149
1150      <?xml version="1.0" encoding="UTF-8"?>
1151      <beans xmlns="http://www.springframework.org/schema/beans"
1152        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1153        xmlns:context="http://www.springframework.org/schema/context"
1154        xmlns:aop="http://www.springframework.org/schema/aop"
1155        xsi:schemaLocation="http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans.xsd
1156          http://www.springframework.org/schema/context
              http://www.springframework.org/schema/context/spring-context-3.2.xsd
1157          http://www.springframework.org/schema/aop
              http://www.springframework.org/schema/aop/spring-aop-4.3.xsd">
1158
1159
1160        <aop:config proxy-target-class="true">
1161          <aop:aspect ref="logAdvice" id="logger">
1162            <aop:pointcut expression="within(com.javasoft.*)*" id="myPointcut" />
1163            <aop:before method="printLog" pointcut-ref="myPointcut"/>
1164          </aop:aspect>
1165        </aop:config>
1166
1167        <bean id="samsungTV" class="com.javasoft.SamsungTV">
1168          <constructor-arg value="samsungTV" />
1169        </bean>
1170        <bean id="lgTV" class="com.javasoft.LgTV">
1171          <constructor-arg value="lgTV" />
1172        </bean>
1173      </beans>
1174
1175    9)src/main/java/test package 생성
1176
```

10)src/main/java/test.Main.java 생성

```
package test;

import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.GenericXmlApplicationContext;

import com.javasoft.LgTV;
import com.javasoft.SamsungTV;
import com.javasoft.TV;

public class Main {
  public static void main(String[] args) {
    AbstractApplicationContext ctx =
        new GenericXmlApplicationContext("classpath:beans.xml");

    TV tv = ctx.getBean("lgTV", LgTV.class);
    tv.powerOn();
    tv.powerOff();
    tv.soundUp();
    tv.soundDown();
    ctx.close();
  }
}
```

11)결과

```
samsungTV : 방금 객체가 생성됐습니다.
lgTV : 방금 객체가 생성됐습니다.
[Core Concern] 수행전 로그하기
lgTV : 전원을 킨다.
[Core Concern] 수행전 로그하기
lgTV : 전원을 끈다.
[Core Concern] 수행전 로그하기
lgTV : 볼륨을 올린다.
[Core Concern] 수행전 로그하기
lgTV : 볼륨을 내린다.
```

16. Lab

1)Advice class 정보

-Class name : PerformanceTraceAdvice.java

-Class 기능 : Target 객체의 method 실행 시간을 계산해서 출력해 주는 부가기능 제공

-Advice 유형 : Around advice

--Target 객체의 method실행 전, 후의 시간을 측정하여 계산하면 Target 객체의 method 실행 시간을 알 수 있다.

-구현 method 이름 : trace(ProceedingJoinPoint joinPoint)

2)New > Spring Legacy Project > Simple Projects > Simple Spring Maven

Project Name : AopDemo2

3)Create Package : src/main/java/com.example

4)src/main/java/com.example.PerformanceTraceAdvice.java

```
package com.example;
import org.aspectj.lang.ProceedingJoinPoint;
```

```
1234        public class PerformanceTraceAdvice {
1235          public Object trace(ProceedingJoinPoint joinPoint) throws Throwable {
1236            //타겟 method의 signature 정보
1237            String signatureString = joinPoint.getSignature().toShortString();
1238            System.out.println(signatureString + " 시작");
1239            //타겟의 method가 호출되기 전의 시간
1240            long start = System.currentTimeMillis();
1241            try {
1242              //타겟의 method 호출
1243              Object result = joinPoint.proceed();
1244              return result;
1245            } finally {
1246              //타겟의 method가 호출된 후의 시간
1247              long finish = System.currentTimeMillis();
1248              System.out.println(signatureString + " 종료");
1249              System.out.println(signatureString + " 실행 시간 : " +
1250                 (finish - start) + " ms");
1251            }
1252          }
1253        }
1254
1255    5)pom.xml : aop code 추가
1256      pom.xml : aop code 추가
1257      -Runtime library 설치
1258        --Maven Repository에서 'aspectj runtime'으로 검색
1259        --aspectj runtime 1.8.10 version을 pom.xml에 추가
1260
1261          <dependency>
1262              <groupId>org.aspectj</groupId>
1263              <artifactId>aspectjrt</artifactId>
1264              <version>1.8.10</version>
1265          </dependency>
1266
1267      -AspectJ Weaver library 설치
1268        --Maven Repository에서 'aspectj weaver'으로 검색
1269        --aspectj weaver 1.8.10 version을 pom.xml에 추가
1270
1271          <dependency>
1272              <groupId>org.aspectj</groupId>
1273              <artifactId>aspectjweaver</artifactId>
1274              <version>1.8.10</version>
1275          </dependency>
1276
1277      -Spring AOP library 설치
1278        --Maven Repository에서 'spring aop'으로 검색
1279        --aspectj weaver 4.3.9 version을 pom.xml에 추가
1280
1281          <dependency>
1282              <groupId>org.springframework</groupId>
1283              <artifactId>spring-aop</artifactId>
1284              <version>4.3.9.RELEASE</version>
1285          </dependency>
1286
1287      -Maven Install
1288
1289          <dependency>
1290              <groupId>org.springframework</groupId>
1291              <artifactId>spring-context</artifactId>
```

```
            <version>4.3.9.RELEASE</version>     <--여기를 수정
          </dependency>

    -Maven Clean -> Maven Install

  6)Advice class를 Bean으로 등록
    -src/main/resources/beans.xml

      <!-- Advice class를 Bean으로 등록 -->
      <bean id="performanceTraceAdvice" class="com.example.PerformanceTraceAdvice" />

  7)beans.xml에 AOP namespace 추가
    -aop - http://www.springframework.org/schema/aop check

  8)AOP 설정

      <aop:config>
        <aop:aspect id="traceAspect"   ref="performanceTraceAdvice">
          <aop:around  pointcut="execution(public * com.example.Hello.*(..))"
          method="trace" />
        </aop:aspect>
      </aop:config>

    -<aop:config> : AOP 설정 정보임을 나타낸다.
    -<aop:aspect> : Aspect를 설정한다.
    -<aop:around pointcut="execution()"> : Around Advice와 Pointcut을 설정한다.

  9)Target Class 작성
    -/src/main/java/com.example.Hello.java

      package com.example;

      import org.springframework.beans.factory.annotation.Value;
      import org.springframework.stereotype.Component;

      @Component("hello")
      public class Hello {
        @Value("Spring")
        private String name;

        @Value("25")
        private int age;

        @Override
        public String toString() {
          return String.format("Hello [name=%s, age=%s]", name, age);
        }
      }

  10)beans.xml 설정
    -Namespace Tab
    -Check context - http://www.springframework.org/schema/context

      <context:component-scan base-package="com.example" />

  11)Around Advice와 AOP 설정 test
    -/src/main/java/com.example.MainClass.java
```

```
1349        package com.example;
1350
1351        import org.springframework.context.ApplicationContext;
1352        import org.springframework.context.support.GenericXmlApplicationContext;
1353
1354        public class MainClass {
1355          public static void main(String[] args) {
1356            ApplicationContext ctx = new
                GenericXmlApplicationContext("classpath:beans.xml");
1357
1358            Hello hello = ctx.getBean("hello", Hello.class);
1359            System.out.println(hello);
1360          }
1361        }
1362
1363    12)결과
1364
1365        Hello.toString() 시작
1366        Hello.toString() 종료
1367        Hello.toString() 실행 시간 : 50 ms
1368        Hello [name=Spring, age=25]
1369
1370
1371  17. Lab : @Aspect annotation 기반의 AOP 구현
1372    1)New > Spring Legacy Project > Simple Projects > Simple Spring Maven
1373      Project Name : AopDemo3
1374
1375    2)Create Package : src/main/java/com.example
1376
1377    3)com.example.Student.java
1378      package com.example;
1379
1380      import java.util.ArrayList;
1381
1382      import org.springframework.beans.factory.DisposableBean;
1383      import org.springframework.beans.factory.InitializingBean;
1384
1385      public class Student{
1386        private String name;
1387        private int age;
1388        private int grade;
1389        private int classNum;
1390        public String getName() {
1391          return name;
1392        }
1393        public void setName(String name) {
1394          this.name = name;
1395        }
1396        public int getAge() {
1397          return age;
1398        }
1399        public void setAge(int age) {
1400          this.age = age;
1401        }
1402        public int getGrade() {
1403          return grade;
1404        }
1405        public void setGrade(int grade) {
```

```
1406          this.grade = grade;
1407        }
1408        public int getClassNum() {
1409          return classNum;
1410        }
1411        public void setClassNum(int classNum) {
1412          this.classNum = classNum;
1413        }
1414        public void getStudentInfo(){
1415          System.out.println("Name : " + this.name);
1416          System.out.println("Age : " + this.age);
1417          System.out.println("Grade : " + this.grade);
1418          System.out.println("Class : " + this.classNum);
1419        }
1420      }
1421
1422    4)com.example.Worker.java
1423      package com.example;
1424
1425      public class Worker {
1426        private String name;
1427        private int age;
1428        private String job;
1429        public String getName() {
1430          return name;
1431        }
1432        public void setName(String name) {
1433          this.name = name;
1434        }
1435        public int getAge() {
1436          return age;
1437        }
1438        public void setAge(int age) {
1439          this.age = age;
1440        }
1441        public String getJob() {
1442          return job;
1443        }
1444        public void setJob(String job) {
1445          this.job = job;
1446        }
1447        public void getWorkerInfo(){
1448          System.out.println("Name : " + this.name);
1449          System.out.println("Age : " + this.age);
1450          System.out.println("Job : " + this.job);
1451        }
1452      }
1453
1454    5)pom.xml : 아래 code 추가
1455      <!-- AOP -->
1456      <dependency>
1457        <groupId>org.aspectj</groupId>
1458        <artifactId>aspectjweaver</artifactId>
1459        <version>1.8.10</version>
1460      </dependency>
1461
1462    6)com.example.LogAop.java
1463      package com.example;
```

```
1464
1465      import org.aspectj.lang.ProceedingJoinPoint;
1466      import org.aspectj.lang.annotation.Around;
1467      import org.aspectj.lang.annotation.Aspect;
1468      import org.aspectj.lang.annotation.Before;
1469      import org.aspectj.lang.annotation.Pointcut;
1470
1471      @Aspect
1472      public class LogAop {
1473
1474        @Pointcut("within(com.example.*)*")
1475        private void pointcutMethod(){}
1476
1477        @Around("pointcutMethod()")
1478        public Object loggerAop(ProceedingJoinPoint joinpoint) throws Throwable{
1479          String signatureStr = joinpoint.getSignature().toShortString();
1480          System.out.println(signatureStr + " is start.");
1481          long start = System.currentTimeMillis();
1482
1483          try{
1484            Object obj = joinpoint.proceed();
1485            return obj;
1486          }finally{
1487            long end = System.currentTimeMillis();
1488            System.out.println(signatureStr + " is finished.");
1489            System.out.println(signatureStr + " 경과시간 : " + (end - start));
1490          }
1491        }
1492
1493        @Before("within(kr.co.javaexpert.*)*")
1494        public void beforeAdvice(){
1495          System.out.println("Called beforeAdvice()");
1496        }
1497      }
1498
1499    7)src/main/resources/beans.xml
1500      <?xml version="1.0" encoding="UTF-8"?>
1501      <beans xmlns="http://www.springframework.org/schema/beans"
1502        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1503        xmlns:aop="http://www.springframework.org/schema/aop"
1504        xsi:schemaLocation="http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans.xsd
1505          http://www.springframework.org/schema/aop
              http://www.springframework.org/schema/aop/spring-aop-3.2.xsd">
1506
1507        <bean id="logAop" class="com.example.LogAop" />
1508
1509        <aop:config>
1510          <aop:aspect id="logger" ref="logAop">
1511            <aop:pointcut expression="within(com.example.*)*" id="publicM"/>
1512            <aop:around method="loggerAop" pointcut-ref="publicM"/>
1513          </aop:aspect>
1514        </aop:config>
1515
1516        <bean id="student" class="com.example.Student">
1517          <property name="name" value="한지민" />
1518          <property name="age" value="15" />
1519          <property name="grade" value="3" />
```

```
        <property name="classNum" value="5" />
      </bean>

      <bean id="worker" class="com.example.Worker">
        <property name="name" value="설운도" />
        <property name="age" value="50" />
        <property name="job" value="개발자" />
      </bean>
    </beans>
```

8)com.example.MainClass.java

```
package com.example;

import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.GenericXmlApplicationContext;

public class MainClass {
  public static void main(String[] args) {
    AbstractApplicationContext context = new
    GenericXmlApplicationContext("classpath:beans.xml");
    Student student = context.getBean("student", Student.class);
    student.getStudentInfo();

    Worker worker = context.getBean("worker", Worker.class);
    worker.getWorkerInfo();

    context.close();
  }
}
```

9)결과

```
Student.getStudentInfo() is start.
Name : 한지민
Age : 15
Grade : 3
Class : 5
Student.getStudentInfo() is finished.
Student.getStudentInfo() 경과시간 : 12 ms
Worker.getWorkerInfo() is start.
Name : 설운도
Age : 50
Job : 개발자
Worker.getWorkerInfo() is finished.
Worker.getWorkerInfo() 경과시간 : 7 ms
```

18. Lab

1)Aspect class 정보

-Class명 : LoggingAspect.java

-Class 기능 : 이 Aspect class는 4가지 유형의 Advice와 Pointcut을 설정하여 Target 객체의 parameter와 return값, 예외 발생 시 예외 message를 출력하는 기능을 제공

-Advice 유형 : Before, AfterReturning, AfterThrowing, After

-구현 method명 : before(JoinPoint joinPoint), afterReturing(JoinPoint joinPoint, Object ret), afterThrowing(JoinPoint joinPoint, Throwable ex), afterFinally(JoinPoint joinPoint)

2)Aspect class 선언 및 설정

-Class 선언부에 @Aspect annotation을 정의한다.

-이 class를 Aspect로 사용하려면 Bean으로 등록해야 하므로 @Component annotation도 함께 정의한다.

```
1575
1576      package com.example;
1577
1578      import org.aspectj.lang.JoinPoint;
1579
1580      @Component
1581      @Aspect
1582      public class LoggingAspect {
1583      ...
1584
1585      <context:component-scan base-package="com.example" />
1586
1587    3)XML 설정파일에 <aop:aspectj-autoproxy /> 선언
1588      -이 선언은 bean으로 등록된 class 중에서 @Aspect가 선언된 class를 모두 Aspect로 자동 등록해주는 역할을
          한다.
1589
1590      <aop:aspectj-autoproxy />
1591
1592    4)AopDemo4 Project 생성
1593      -Spring Legacy Project > Simple Maven Project
1594
1595    5)com.example package 생성
1596      -/src/main/java/com.example
1597
1598    6)pom.xml에 Aspectj 종속성 추가 및 설치
1599
1600        <dependency>
1601          <groupId>org.aspectj</groupId>
1602          <artifactId>aspectjweaver</artifactId>
1603          <version>1.8.10</version>
1604        </dependency>
1605
1606      -Maven Install
1607
1608    7)/src/main/java/com.example.LoggingAspect.java 생성
1609
1610        package com.example;
1611
1612        import org.aspectj.lang.JoinPoint;
1613        import org.aspectj.lang.annotation.After;
1614        import org.aspectj.lang.annotation.AfterReturning;
1615        import org.aspectj.lang.annotation.AfterThrowing;
1616        import org.aspectj.lang.annotation.Aspect;
1617        import org.aspectj.lang.annotation.Before;
1618        import org.springframework.stereotype.Component;
1619        @Component
1620        @Aspect
1621        public class LoggingAspect {
1622          @Before("execution(public * com.example..*(..))")
1623          public void before(JoinPoint joinPoint) {
1624            String signatureString = joinPoint.getSignature().getName();
1625            System.out.println("@Before [ " + signatureString + " ] 메서드 실행 전처리 수행");
1626            for (Object arg : joinPoint.getArgs()) {
1627              System.out.println("@Before [ " + signatureString + " ] 아규먼트 " + arg);
1628            }
1629          }
1630            @AfterReturning(pointcut="execution(public * com.example..*(..))",
              returning="ret")
```

```
1631          public void afterReturning(JoinPoint joinPoint, Object ret) {
1632            String signatureString = joinPoint.getSignature().getName();
1633            System.out.println("@AfterReturing [ " + signatureString + " ] method 실행 후처리 수
                행");
1634            System.out.println("@AfterReturing [ " + signatureString + " ] 리턴값=" + ret);
1635
1636          }
1637
1638            @AfterThrowing(pointcut="execution(public * com.example..*(..))",
1639                throwing="ex")
1640          public void afterThrowing(JoinPoint joinPoint, Throwable ex) {
1641            String signatureString = joinPoint.getSignature().getName();
1642            System.out.println("@AfterThrowing [ " + signatureString + " ] method 실행 중 예외
                발생");
1643            System.out.println("@AfterThrowing [ " + signatureString + " ] 예외=" +
                ex.getMessage());
1644          }
1645
1646            @After("execution(public * com.example..*(..))")
1647          public void afterFinally(JoinPoint joinPoint) {
1648            String signatureString = joinPoint.getSignature().getName();
1649            System.out.println("@After [ " + signatureString + " ] method 실행 완료");
1650          }
1651        }
1652
1653    8)beans.xml 파일 생성
1654      -/src/main/resources/beans.xml
1655      -Namespace Tab에서 'aop'와 'context' 체크할 것
1656
1657        <?xml version="1.0" encoding="UTF-8"?>
1658        <beans xmlns="http://www.springframework.org/schema/beans"
1659          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1660          xmlns:aop="http://www.springframework.org/schema/aop"
1661          xmlns:context="http://www.springframework.org/schema/context"
1662          xsi:schemaLocation="http://www.springframework.org/schema/beans
              http://www.springframework.org/schema/beans/spring-beans.xsd
1663              http://www.springframework.org/schema/context
                  http://www.springframework.org/schema/context/spring-context-3.2.xsd
1664              http://www.springframework.org/schema/aop
                  http://www.springframework.org/schema/aop/spring-aop-3.2.xsd">
1665
1666          <context:component-scan base-package="com.example" />
1667          <aop:aspectj-autoproxy />
1668        </beans>
1669
1670    9)Target 객체 생성
1671      -/src/main/java/com.example.Hello.java
1672
1673        package com.example;
1674
1675        import org.springframework.beans.factory.annotation.Value;
1676        import org.springframework.stereotype.Component;
1677
1678        @Component("hello")
1679        public class Hello {
1680          @Value("Spring")
1681          private String name;
1682
```

```
1683          @Value("25")
1684          private int age;
1685
1686          @Override
1687          public String toString() {
1688            return String.format("Hello [name=%s, age=%s]", name, age);
1689          }
1690
1691          public void calculation(){
1692            System.out.println(5 / 0);
1693          }
1694        }
1695
1696    10)테스트 class 작성
1697      -src/main/java/com.example.MainClass.java
1698
1699        package com.example;
1700
1701        import org.springframework.context.ApplicationContext;
1702        import org.springframework.context.support.GenericXmlApplicationContext;
1703
1704        public class MainClass {
1705          public static void main(String[] args) {
1706            ApplicationContext ctx = new
                GenericXmlApplicationContext("classpath:beans.xml");
1707
1708            Hello hello = ctx.getBean("hello", Hello.class);
1709            System.out.println(hello);
1710            hello.calculation();
1711          }
1712        }
1713
1714    11)결과
1715        @Before [ toString ] method 실행 전처리 수행
1716        @After [ toString ] method 실행 완료
1717        @AfterReturing [ toString ] method 실행 후처리 수행
1718        @AfterReturing [ toString ] 리턴값=Hello [name=Spring, age=25]
1719        Hello [name=Spring, age=25]
1720        @Before [ calculation ] method 실행 전처리 수행
1721        @After [ calculation ] method 실행 완료
1722        @AfterThrowing [ calculation ] method 실행 중 예외 발생
1723        @AfterThrowing [ calculation ] 예외=/ by zero
1724
1725
1726  19. AspectJ Pointcut 표현식
1727    1)표현식은 Pointcut 지시자를 이용하여 작성
1728
1729    2)가장 대표적인 지시자는 execution()이다.
1730
1731    3)Pointcut 을 지정할 때 사용하는 표현식으로 AspectJ 문법을 사용한다.
1732      -* : 모든
1733      -. : 현재
1734      -.. : 0개 이상
1735
1736    4)execution
1737      -Usage
1738        execution([접근제한자 pattern] return type pattern [type pattern.] 이름 pattern(parameter
          type pattern | "..", ...) [throws 예외pattern])
```

1739 --접근제한자 pattern : public, private과 같은 접근 제한자, 생략가능
1740 --Return type pattern : return값의 type pattern
1741 --Type pattern : 패키지와 class 이름에 대한 pattern, 생략가능. 사용할 때 "."를 사용해 연결함.
1742 --이름 pattern : method 이름 type pattern
1743 --Parameter 타입pattern : parameter의 type pattern을 순서대로 넣을 수 있다. wildcard를 이용해서 parameter 갯수에 상관없는 pattern을 만들 수 있다.
1744 --예외 pattern : 예외 이름 pattern
1745
1746 -예
1747 "execution(* aspects.trace.demo.*.*(..))"
1748 -* : Any return type
1749 -aspects.trace.demo : package
1750 -* : class
1751 -* : method
1752 -(..) : Any type and number of arguments
1753
1754 -execution(* hello(..))
1755 --hello라는 이름을 가진 method를 선정
1756 --Parameter는 모든 종류를 다 허용
1757
1758 -execution(* hello())
1759 --hello method 중에서 parameter가 없는 것만 선택함.
1760
1761 -execution(* com.example.service.UserServiceImpl.*(..))
1762 --com.example.service.UserServiceImpl class를 직접 지정
1763 --이 class가 가진 모든 method를 선택
1764
1765 -execution(* com.example.user.service.*.*(..))
1766 --com.example.user.service package의 모든 class에 적용
1767 --하지만 sub-package의 class는 포함하지 않는다.
1768
1769 -execution(* com.example.user.service..*.*(..))
1770 --com.example.user.service package의 모든 class에 적용
1771 --그리고 '..'를 사용해서 sub-package의 모든 class까지 포함
1772
1773 -execution(* *.. Target.*(..))
1774 --Package에 상관없이 Target이라는 이름의 모든 class에 적용
1775 --다른 package의 같은 이름의 class가 있어도 적용이 된다는 점에 유의해야 함.
1776
1777 @Pointcut("executeion(public void get*(..))") : public void인 모든 get method
1778 @Pointcut("executeion(* com.example.*.*())") : com.example package에 parameter가 없는 모든 method
1779 @Pointcut("executeion(* com.example..*.*())") : com.example package & kr.co.javaexpert 하위 package에 parameter가 없는 모든 method
1780 @Pointcut("executeion(* com.example.Worker.*())") : com.example.Worker 안의 모든 method
1781
1782 2)within
1783 @Pointcut("within(com.example.*)") : com.example package 안에 있는 모든 method
1784 @Pointcut("within(com.example..*)") : com.example package 및 하위 package 안에 있는 모든 method
1785 @Pointcut("within(com.example.Worker)") : com.example.Worker 모든 method
1786
1787 3)bean
1788 @Pointcut("bean(student)") : student bean에만 적용
1789 @Pointcut("bean(*ker)") : ~ker로 끝나는 bean에만 적용