1 1. AOP What?

- 1)어플리케이션은 다양한 공통 기능을 필요로 한다.
- 3 **2)**로깅과 같은 기본적인 기능에서부터 트랜잭션이나 보안과 같은 기능에 이르기까지 어플리케이션 전반에 걸쳐 적용되는 공통 기능이 존재한다.
- 4 3)공통기능은 어플리케이션의 핵심 비지니스 로직과는 구분되고 핵심기능을 도와 주는 부가적인 기능(로깅, 보안 등)이다.
 - 4)핵심 비즈니스 기능과 구분하기 위해 공통 기능을 공통 관심 사항(Cross-cutting Concern)이라고 표현한다.
 - 5)핵심 로직(Biz 로직을 포함하는 기능)을 핵심 관심 사항(Core Concern)이라고 표현한다.
 - -핵심기능
 - --계좌 이체, 대출 승인, 이자 계산
 - -부가기능
 - --로깅, 보안, 트렌젝션
- 11 -그림 참조

http://dev.anyframejava.org/docs/anyframe/plugin/foundation/4.6.0/reference/html/ch05.html

12 13 14

2

5

6

8

9

10

- 6)객체지향의 기본 원칙을 적용하면서도 핵심기능에서 부가기능을 분리해서 모듈화하는 것은 매우 어렵다.
- 15 7)프로그래밍에서 공통적인 기능을 모든 모듈에 적용하기 위한 방법으로 상속이 있다.
- 16 8)하지만 자바는 다중상속을 하지 않기 때문에 다양한 모듈에 상속 기법을 통한 공통 기능 부여에는 한계가 있다.
- 17 9)AOP는 핵심기능과 공통 기능을 분리시켜놓고, 공통 기능을 필요로 하는 핵심 기능들에서 사용하는 방식이다.
- 18 10)AOP(Aspect Oriented Programming)은 문제를 바라보는 관점(시점)을 기준으로 프로그래밍하는 기법이다.
- 19 11)분리한 부가기능(공통 기능)을 Aspect라는 독특한 모듈 형태로 만들어서 설계하고 개발하는 방법이다.
- 20 12)기본적인 개념은 공통 관심 사항을 구현한 코드를 핵심 로직을 구현한 코드 안에 삽입한다는 것이다.
- 21 13)OOP를 적용하여도 핵심기능에서 부가기능을 쉽게 분리된 모듈로 작성하기 어려운 문제점을 AOP가 해결해 준다고 볼 수 있다.
- 22 **14)**즉, AOP는 부가기능을 Aspect로 정의하여, 핵심기능에서 부가기능을 분리함으로써 핵심 기능을 설계하고 구현할 때 객체지향적인 가치를 지킬 수 있도록 도와주는 개념이다.
- 23 15)AOP 기법에서는 핵심 로직을 구현한 코드에서 공통 기능을 직접적으로 호출하지 않는다.
- 24 16)핵심 로직을 구현한 코드를 컴파일하거나, 컴파일된 클래스를 로딩하거나, 로딩한 클래스의 객체를 생성할 때 AOP가 적용되어 핵심 로직 구현 코드안에 공통 기능이 삽입된다.
- 25 **17)AOP**에서는 **AOP** 라이브러리가 공통 기능을 알맞게 삽입해주기 때문에 개발자는 게시글 쓰기나 목록 읽기와 같은 핵심 로직을 구현할 때 트랜잭션 적용이나 보안검사와 같은 공통 기능을 처리하기위한 코드를 핵심 로직 코드에 삽입할 필요가 없다.
- 26 **18)**핵심 로직을 구현한 코드에 공통 기능 관련 코드가 포함되어 있지 않기 때문에 적용해야 할 공통 기능이 변경되더라도 핵심 로직을 구현한 코드를 변경할 필요가 없다.
- 27 19)단지, 공통 기능 코드를 변경한 뒤 핵심 로직 구현 코드에 적용만 하면 된다.
- 28 20)AOP 개념을 적용하면 핵심기능 코드 사이에 침투된 부가기능을 독립적인 Aspect로 구분해 낼 수 있다.
 - 21)구분된 부가기능 Aspect를 런타임 시에 필요한 위치에 동적으로 참여하게 할 수 있다.

30 31

33

34

29

- 32 **2. AOP** 용어
 - 1)Aspect : 여러 객체에 공통으로 적용되는 공통 관심 사항. 예)트랜잭션이나 보안, 로깅 등...
 - 2)Target : 핵심 기능을 담고 있는 모듈로, Target은 부가기능을 부여할 대상이 된다.

35 36 37

38

39

- 3)Advice : 언제 공통관심 기능을 핵심 로직에 적용할지를 정의한다.
- -즉, 부가기능을 정의한 코드.
 - -Target에 제공할 부가기능을 담고 있는 모듈.
 - -예)'메소드를 호출하기 전'(언제)에 '트랜잭션을 시작한다.'(공통기능) 기능을 적용한다는 것을 정의

40 41 42

43

44

45

- 4)JoinPoint: Advice를 적용해야 되는 지점
 - -즉, Target 객체가 구현한 인터페이스의 모든 메소드는 JoinPoint가 된다.
 - -ex. 필드값 변경, 메소드호출
 - -Spring에서는 메소드호출만 해당

46 47 48

- 5)Pointcut: JoinPoint의 부분집합으로 실제로 Advice가 적용되는 JoinPoint 부분.
 - -Spring에서는 정규 표현식이나 AspectJ의 문법을 이용하여 정의한다.
 - -표현식은 execution으로 시작하고, 메소드의 Signature를 비교하는 방법을 주로 이용

49 50 51

52

53

- 6)Weaving: Advice를 핵심 코드에 적용하는 행위.
 - -공통 코드를 핵심 로직 코드에 삽입하는 것.
 - -AOP가 핵심기능(Target)의 코드에 영향을 주지 않으면서 필요한 부가기능(Advice)을 추가할 수 있도록 해주는 핵심적인 처리과정

54 55

7)즉, Aspect = Advice + PointCut 이다.

56 57 58

8)Aspect는 AOP의 기본 모듈

```
59
      9)Aspect는 Singleton 형태의 객체로 존재한다.
 60
 61
      10)Advisor = Advice + Pointcut
 62
        -Spring AOP에서만 사용되는 특별한 용어
 63
      11)그림참조
 64
 65
        http://isstory83.tistory.com/90
 66
 67
 68
    3. 3 가지 Weaving 방식
 69
      1)컴파일 시: AspectJ에서 사용하는 방식
 70
      2)클래스 로딩 시
 71
      3)런타임 시 : Proxy를 이용. 핵심 로직을 구현한 객체에 직접 접근하는 것이 아니라 중간에 Proxy를 생성하여
      Proxy를 통해서 핵심 로직을 구현한 객체에 접근
 72
        -Spring에서 AOP를 구현하는 방법 : Proxy를 이용한다.
 73
        -호출부(Client) --> Proxy(대행) --> Target(핵심기능)
 74
 75
 76
    4. Spring AOP의 특징
 77
      1)Spring은 Proxy 기반 AOP를 지원한다.
 78
        -Spring은 Target 객체에 대한 Proxy를 만들어 제공한다.
 79
        -Target을 감싸는 Proxy는 실행시간(Runtime)에 생성된다.
 80
        -Proxy는 Advice를 Target객체에 적용하면서 생성되는 객체이다.
 81
 82
      2)Proxy가 호출을 intercept한다.
 83
        -Proxy는 Target 객체에 대한 호출을 가로챈 다음, Advice의 부가기능 로직을 수행하고 난 후에 Target의
        핵심 기능 로직을 호출한다.(전처리 Advice)
 84
        -또는 Target의 핵심기능 로직 메소드를 호출한 뒤에 부가기능(Advice)을 수행하는 경우도 있다.(후처리
        Advice)
 85
 86
      3)Spring AOP는 메소드 JoinPoint만 지원한다.
 87
        -Spring은 동적 Proxy를 기반으로 AOP를 구현하기 때문에 메소드 JoinPoint만 지원한다.
 88
        -즉, 핵심기능(Target)의 메소드가 호출되는 런타임 시점에만 부가기능(Advice)을 적용할 수 있다.
 89
        -반면에, Aspect 같은 고급 AOP 프레임워크를 사용하면 객체의 생성, 필드값의 조회와 조작, static
        메소드 호출 및 초기화 등의 다양한 작업에 부가기능을 적용할 수 있다.
 90
 91
 92
    5. 스프링에서 AOP 구현 방식
 93
      1)Spring API를 이용한 AOP 구현
 94
 95
      2)XML 스키마 기반의 POJO 클래스를 이용한 AOP구현: Spring 2부터 사용
 96
        -부가기능을 제공하는 Advice 클래스를 작성
 97
        -XML 설정 파일에 <aop:config>를 이용해서 Aspect를 설정
 98
        -즉, Advice와 Pointcut을 설정
 99
100
      3)@Aspect 어노테이션 기반의 AOP 구현
101
        -@Aspect 어노테이션을 이용해서 부가기능을 제공하는 Aspect 클래스를 작성
102
        -이때, Aspect 클래스는 Advice를 구현하는 메소드와 Pointcut을 포함한다.
103
        -XML 설정 파일에 <aop:aspectj-autoproxy />를 설정
104
105
      4)@Aspect annotation
106
        -Aspect 클래스를 선언할 때 @Aspect 어노테이션을 사용한다.
107
        -AspectJ 5 버전에 새롭게 추가된 어노테이션이다.
108
        -@Aspect 어노테이션을 이용할 경우 XML 설정 파일에 Advice와 Pointcut을 설정하는 것이 아니라 클래스
        내부에 정의할 수 있다.
109
        -<aop:aspectj-autoproxy> 태그를 설정파일에 추가하면 @Aspect 어노테이션이 적용된 Bean을
        Aspect로 사용 가능하다.
110
111
    6. AspectJ와 Spring AOP library 설치
112
      1)Runtime library 설치
113
        -Maven Repository에서 'aspectj runtime'으로 검색
114
        -aspectj runtime 1.8.10 버전을 pom.xml에 추가
115
116
           <dependency>
117
               <groupId>org.aspectj</groupId>
118
               <artifactId>aspectjrt</artifactId>
119
               <version>1.8.10</version>
```

```
120
           </dependency>
121
       2)AspectJ Weaver library 설치
122
123
         -Maven Repository에서 'aspectj weaver'으로 검색
124
         -aspectj weaver 1.8.10 버전을 pom.xml에 추가
125
126
            <dependency>
127
                <groupId>org.aspectj</groupId>
128
                <artifactId>aspectjweaver</artifactId>
129
                <version>1.8.10</version>
130
           </dependency>
131
       3)Spring AOP library 설치
132
133
         -Maven Repository에서 'spring aop'으로 검색
134
         -aspectj weaver 4.3.9 버전을 pom.xml에 추가
135
136
           <dependency>
137
                <groupId>org.springframework</groupId>
138
                <artifactId>spring-aop</artifactId>
139
                <version>4.3.9.RELEASE
140
            </dependency>
141
142
       4)AspectJ Runtime API 문서
143
         -Google에서 'aspectj runtime aip doc'로 검색
144
         -https://eclipse.org/aspectj/doc/released/runtime-api/index.html
145
146
    7. Advice의 종류
147
148
       1)<aop:before>
149
         -메소드 실행 전에 Advice실행
150
         -JoinPoint 앞에서 실행되는 Advice
151
152
       2)<aop:after-returning>
153
         -정상적으로 메소드 실행 후에 Advice 실행
154
         -JoinPoint 메소드 호출이 정상적으로 종료된 뒤에 실행되는 Advice
155
156
       3)<aop:after-throwing>
157
         -메소드 실행 중 exception 발생시 Advice 실행
158
         -try-catch의 catch와 비슷
159
160
       4)<aop:after>
161
         -메소드 실행 중 exception 이 발생하여도 Advice 실행
162
         -try-catch-finally에서 finally와 비슷
163
164
       5)<aop:around>
         -Target의 메소드 실행 전/후 및 exception 발생시 Advice 실행
165
166
         -JoinPoint 앞과 뒤에서 실행되는 Advice
167
168
    8. Advice를 정의하는 어노테이션
169
       1)@Before("pointcut")
170
         -Target 객체의 메소드가 실행되기 전에 호출되는 Advice
171
         -JoinPoint를 통해 파라미터 정보를 참조할 수 있다.
172
173
       2)@After("pointcut")
174
         -Target 객체의 메소드가 정상 종료됐을 때와 예외가 발생했을 때 모두 호출되는 Advice
175
         -리턴값이나 예외를 직접 전달 받을 수는 없다.
176
177
       3)@Around("pointcut")
178
         -Target 객체의 메소드가 호출되는 전 과정을 모두 담을 수 있는 가장 강력한 기능을 가진 Advice
179
       4)@AfterReturning(pointcut="", returning="")
180
181
         -Target 객체의 메소드가 정상적으로 실행을 마친 후에 호출되는 Advice
182
         -리턴값을 참조할 때는 returning 속성에 리턴값을 저장할 변수 이름을 지정해야 한다.
183
       5)@AfterThrowing(pointcut="", throwing="")
184
         -Target 객체의 메소드가 예외가 발생하면 호출되는 Advice
185
         -발생된 예외를 참조할 때는 throwing 속성에 발생한 예외를 저장할 변수 이름을 지정해야 한다.
186
```

```
187
188
189
     9. JoinPoint Interace
190
       1)JoinPoint 는 Spring AOP 혹은 AspectJ에서 AOP가 적용되는 지점을 뜻한다.
191
192
       2)해당 지점을 AspectJ에서 JointPoint라는 인터페이스로 나타낸다.
193
194
       3)Methos
195
         -getArgs(): 메소드 argument 반환
196
          -getThis(): Proxy 객체를 반환
197
         -getTarget(): 대상 객체를 반환
198
         -getSignature(): Advice되는 메소드의 설명(description)을 반환
199
         -toString(): Advice되는 메소드의 설명을 출력
200
       4)모든 Advice는 org.aspectj.lang.JoinPoint 타입의 파라미터를 Advice 메소드에 첫 번째 매개변수로 선언
201
       가능
202
203
       5)Around Advice는 JoinPoint의 하위 클래스인 ProceedingJoinPoint 타입의 파라미터를 필수적으로
       선언해야 함.
204
205
       6)AspectJ Runtime API의 orq.aspectj.lanq의 JoinPoint interface 참조할 것
206
207
       7)AspectJ Runtime API의 org.aspectj.lang의 ProceedingJoinPoint interface 참조할 것
208
209
210
     10. AOP 설정
211
       1)<aop:config>: AOP의 설정 정보임을 나타낸다.
212
       2)<aop:aspect> : Aspect를 설정한다.
213
       3)<aop:around pointcut="execution()"> : Around Advice와 Pointcut을 설정한다.
       4)<aop:aspect> 태그의 ref속성은 Aspect로서 기능을 제공할 Bean을 설정할 때 사용함.
214
215
       5)<aop:around> 태그의 pointcut 속성의 execution 지시자(designator)는 Advice를 적용할 패키지,
       클래스, 메소드를 표현할 때 사용됨.
       6)info.javaexpert.service 패키지 및 그 하위 패키지에 있는 모든 public 메소드를 Pointcut으로 설정하고 있다.
216
217
       7)UserServiceImpl의 public 메소드가 호출될 때 PerformanceTraceAdvice Bean의 trace() 메소드가
       호출되도록 설정하고 있다.
218
219
220
     11. Lab: XML 스키마 기반의 AOP 구현
221
       1)New > Spring Legacy Project > Simple Projects > Simple Spring Maven
222
        Project Name: AopDemo
223
224
       2)Create Package: src/main/java/info.javaexpert
225
226
       3)info.javaexpert.Student.java
227
         package info.javaexpert;
228
229
         import java.util.ArrayList;
230
231
         import org.springframework.beans.factory.DisposableBean;
232
         import org.springframework.beans.factory.InitializingBean;
233
234
         public class Student{
235
            private String name;
236
            private int age;
237
            private int grade;
238
            private int classNum;
239
            public String getName() {
240
              return name;
241
242
            public void setName(String name) {
243
              this.name = name;
244
245
            public int getAge() {
246
              return age;
247
            }
248
            public void setAge(int age) {
249
              this.age = age;
```

```
250
251
             public int getGrade() {
252
                return grade;
253
             }
254
             public void setGrade(int grade) {
255
                this.grade = grade;
256
257
             public int getClassNum() {
258
                return classNum;
259
             }
260
             public void setClassNum(int classNum) {
261
                this.classNum = classNum;
262
             }
263
             public void getStudentInfo(){
                System.out.println("Name : " + this.name);
264
265
                System.out.println("Age : " + this.age);
                System.out.println("Grade : " + this.grade);
266
267
                System.out.println("Class : " + this.classNum);
268
             }
269
          }
270
271
        4)info.javaexpert.Worker.java
272
          package info.javaexpert;
273
274
          public class Worker {
275
             private String name;
276
             private int age;
277
             private String job;
278
             public String getName() {
279
                return name;
280
281
             public void setName(String name) {
282
                this.name = name;
283
284
             public int getAge() {
285
                return age;
286
287
             public void setAge(int age) {
288
                this.age = age;
289
290
             public String getJob() {
291
                return job;
292
293
             public void setJob(String job) {
294
                this.job = job;
295
296
             public void getWorkerInfo(){
297
                System.out.println("Name : " + this.name);
298
                System.out.println("Age : " + this.age);
                System.out.println("Job: " + this.job);
299
300
301
          }
302
        5)pom.xml: aop 코드 추가
303
304
          -Runtime library 설치
             --Maven Repository에서 'aspectj runtime'으로 검색
305
306
             --aspectj runtime 1.8.10 버전을 pom.xml에 추가
307
308
                <dependency>
309
                     <groupId>org.aspectj</groupId>
310
                     <artifactId>aspectirt</artifactId>
311
                     <version>1.8.10</version>
312
                </dependency>
313
314
          -AspectJ Weaver library 설치
315
             --Maven Repository에서 'aspectj weaver'으로 검색
316
             --aspectj weaver 1.8.10 버전을 pom.xml에 추가
```

```
317
318
               <dependency>
319
                    <groupId>org.aspectj</groupId>
320
                    <artifactId>aspectjweaver</artifactId>
321
                    <version>1.8.10</version>
322
               </dependency>
323
324
          -Spring AOP library 설치
325
            --Maven Repository에서 'spring aop'으로 검색
326
            --aspectj weaver 4.3.9 버전을 pom.xml에 추가
327
328
               <dependency>
329
                    <groupId>org.springframework</groupId>
330
                    <artifactId>spring-aop</artifactId>
331
                    <version>4.3.9.RELEASE
332
               </dependency>
333
334
          -Maven Install
335
336
       6)info.javaexpert.LogAop.java
337
          package info.javaexpert;
338
339
          import org.aspectj.lang.ProceedingJoinPoint;
340
341
          public class LogAop {
342
          //joinpoint 객체를 전달 받을 때에는 반드시 첫번째 파라미터여야 한다.
343
            public Object loggerAop(ProceedingJoinPoint joinpoint) throws Throwable{
344
               String signatureStr = joinpoint.getSignature().toShortString();
345
               //Signature getSignature() : 호출되는 메소드에대한 정보를 구한다.
346
               //cf)Object getTarget() : 대상 객체를 구한다.
347
               //cf)Object [] getArgs() : 파라미터 목록을 구한다.
348
349
               //toShortString() : 메소드를 축약해서 표현한 문장을 구한다. 메소드의 이름만 구한다.
350
               //cf)toLongString() : 완전하게 표현된 문장. 리턴타입, 메소드이름, 파라미터 타입 모두
               //cf)getName() : 메소드의 이름을 구한다.
351
352
               System.out.println(signatureStr + " is start.");
353
               long start = System.currentTimeMillis();
354
355
               try{
356
                 Object obj = joinpoint.proceed(); //대상객체의 실제 메소드 호출
357
                 return obj;
358
               }finally{
359
                 long end = System.currentTimeMillis();
360
                 System.out.println(signatureStr + " is finished.");
                 System.out.println(signatureStr + " 경과시간 : " + (end - start));
361
362
               }
363
            }
          }
364
365
366
       7)src/main/resources/beans.xml
367
          -Namespace tab
368
            --aop Check
369
370
          <?xml version="1.0" encoding="UTF-8"?>
371
          <beans xmlns="http://www.springframework.org/schema/beans"</pre>
372
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
373
            xmlns:aop="http://www.springframework.org/schema/aop"
374
            xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans.xsd
375
               http://www.springframework.org/schema/aop
               http://www.springframework.org/schema/aop/spring-aop-3.2.xsd">
376
377
            <bean id="logAop" class="info.javaexpert.LogAop" />
378
379
            <aop:config>
               <aop:aspect id="logger" ref="logAop">
380
                  <aop:pointcut expression="within(info.javaexpert.*)*" id="publicMethod"/>
381
```

```
<aop:around method="loggerAop" pointcut-ref="publicMethod"/>
382
383
               </aop:aspect>
384
             </aop:config>
385
             <!-- info.javaexpert아래 모든 클래스의 public Method를 호출할 때 LogAop의 loggerAop 메소드가
386
            실행된다는 뜻 -->
387
             <bean id="student" class="info.javaexpert.Student">
388
389
               roperty name="name" value="한지민" />
               cproperty name="age" value="15" />
390
391
               cproperty name="grade" value="3" />
392
               cproperty name="classNum" value="5" />
            </bean>
393
394
            <bean id="worker" class="info.javaexpert.Worker">
395
396
               roperty name="name" value="설운도" />
397
               cproperty name="age" value="50" />
               property name="job" value="개발자" />
398
399
             </bean>
400
          </beans>
401
402
       8)info.javaexpert.MainClass.java
403
          package info.javaexpert;
404
405
          import org.springframework.context.support.AbstractApplicationContext;
406
          import org.springframework.context.support.GenericXmlApplicationContext;
407
408
          public class MainClass {
409
            public static void main(String[] args) {
410
               AbstractApplicationContext context = new
               GenericXmlApplicationContext("classpath:beans.xml");
               Student student = context.getBean("student", Student.class);
411
412
               student.getStudentInfo();
413
414
               Worker worker = context.getBean("worker", Worker.class);
415
               worker.getWorkerInfo();
416
417
               context.close();
418
            }
419
          }
420
421
       9)결과
422
          Student.getStudentInfo() is start.
423
          Name: 한지민
424
          Age: 15
425
          Grade: 3
426
          Class: 5
          Student.getStudentInfo() is finished.
427
428
          Student.getStudentInfo() 경과시간: 14
429
          Worker.getWorkerInfo() is start.
430
          Name: 설운도
431
          Age: 50
432
          Job:개발자
433
          Worker.getWorkerInfo() is finished.
434
          Worker.getWorkerInfo() 경과시간: 7
435
436
       10)Error 발생시 아래와 같이 조치한다.
437
          -pom.xml 파일에서 아래를 수정한다.
438
             <dependencies>
               <!-- Spring and Transactions -->
439
440
               <dependency>
441
                  <groupId>org.springframework</groupId>
442
                  <artifactId>spring-context</artifactId>
443
                  <version>4.3.9.RELEASE
                                                      <--여기를 수정
444
               </dependency>
445
446
```

```
447
     12. Lab
448
       1)Advice 클래스 정보
449
          -Class name : PerformanceTraceAdvice.java
450
          -Class 기능 : Target 객체의 메소드 실행 시간을 계산해서 출력해 주는 부가기능 제공
451
          -Advice 유형: Around advice
452
            --Target 객체의 메소드실행 전, 후의 시간을 측정하여 계산하면 Target 객체의 메소드 실행 시간을 알
            수 있다.
453
          -구현 메소드 이름: trace(ProceedingJoinPoint joinPoint)
454
455
       2) New > Spring Legacy Project > Simple Projects > Simple Spring Maven
         Project Name: AopDemo1
456
457
458
       3)Create Package: src/main/java/info.javaexpert
459
460
       4)src/main/java/info.javaexpert.PerformanceTraceAdvice.java
461
462
            package info.javaexpert;
463
            import org.aspectj.lang.ProceedingJoinPoint;
464
465
            public class PerformanceTraceAdvice {
466
               public Object trace(ProceedingJoinPoint joinPoint) throws Throwable {
                 //타겟 메서드의 signature 정보
467
468
                 String signatureString = joinPoint.getSignature().toShortString();
469
                 System.out.println(signatureString + " 시작");
470
                 //타겟의 메서드가 호출되기 전의 시간
471
                 long start = System.currentTimeMillis();
472
                 try {
473
                    //타겟의 메서드 호출
474
                    Object result = joinPoint.proceed();
475
                    return result;
476
                 } finally {
477
                    //타겟의 메서드가 호출된 후의 시간
478
                    long finish = System.currentTimeMillis();
479
                    System.out.println(signatureString + " 종료");
                    System.out.println(signatureString + " 실행 시간 : " +
480
481
                       (finish - start) + " ms");
482
                 }
483
               }
484
            }
485
486
       5)pom.xml: aop 코드 추가
          pom.xml: aop 코드 추가
487
488
          -Runtime library 설치
            --Maven Repository에서 'aspectj runtime'으로 검색
489
490
            --aspectj runtime 1.8.10 버전을 pom.xml에 추가
491
492
               <dependency>
493
                    <groupId>org.aspectj</groupId>
494
                    <artifactId>aspectjrt</artifactId>
495
                    <version>1.8.10</version>
496
               </dependency>
497
498
          -AspectJ Weaver library 설치
499
            --Maven Repository에서 'aspectj weaver'으로 검색
500
            --aspectj weaver 1.8.10 버전을 pom.xml에 추가
501
502
               <dependency>
503
                    <groupId>org.aspectj</groupId>
504
                    <artifactId>aspectjweaver</artifactId>
505
                    <version>1.8.10</version>
506
               </dependency>
507
508
          -Spring AOP library 설치
509
            --Maven Repository에서 'spring aop'으로 검색
510
            --aspectj weaver 4.3.9 버전을 pom.xml에 추가
511
```

<dependency>

```
513
                    <groupId>org.springframework</groupId>
514
                    <artifactId>spring-aop</artifactId>
515
                    <version>4.3.9.RELEASE
516
               </dependency>
517
518
          -Maven Install
519
520
             <dependency>
521
                  <groupId>org.springframework</groupId>
522
                  <artifactId>spring-context</artifactId>
523
                  <version>4.3.9.RELEASE
                                                      <--여기를 수정
524
               </dependency>
525
          -Maven Clean -> Maven Install
526
527
528
       6)Advice 클래스를 Bean으로 등록
529
          -src/main/resources/beans.xml
530
531
             <!-- Advice 클래스를 Bean으로 등록 -->
             <bean id="performanceTraceAdvice" class="info.javaexpert.PerformanceTraceAdvice" />
532
533
534
       7)beans.xml에 AOP 네임스페이스 추가
535
          -aop - http://www.springframework.org/schema/aop check
536
537
       8)AOP 설정
538
539
            <aop:config>
               <aop:aspect id="traceAspect" ref="performanceTraceAdvice">
540
541
                  <aop:around pointcut="execution(public * info.javaexpert.Hello.*(..))" method="trace"</p>
542
               </aop:aspect>
543
             </aop:config>
544
545
          -<aop:config> : AOP 설정 정보임을 나타낸다.
546
          -<aop:aspect> : Aspect를 설정한다.
          -<aop:around pointcut="execution()"> : Around Advice와 Pointcut을 설정한다.
547
548
549
       9)Target Class 작성
550
          -/src/main/java/info.javaexpert.Hello.java
551
552
            package info.javaexpert;
553
554
            import org.springframework.beans.factory.annotation.Value;
555
            import org.springframework.stereotype.Component;
556
            @Component("hello")
557
558
            public class Hello {
               @Value("Spring")
559
560
               private String name;
561
562
               @Value("25")
563
               private int age;
564
565
               @Override
566
               public String toString() {
567
                 return String.format("Hello [name=%s, age=%s]", name, age);
568
               }
569
            }
570
571
       10)beans.xml 설정
572
          -Namespace Tab
573
          -Check context - http://www.springframework.org/schema/context
574
575
             <context:component-scan base-package="info.javaexpert" />
576
577
       11)Around Advice와 AOP 설정 테스트
578
          -/src/main/java/info.javaexpert.MainClass.java
```

```
579
580
             package info.javaexpert;
581
582
             import org.springframework.context.ApplicationContext;
583
             import org.springframework.context.support.GenericXmlApplicationContext;
584
585
             public class MainClass {
586
                public static void main(String[] args) {
587
                   ApplicationContext ctx = new GenericXmlApplicationContext("classpath:beans.xml");
588
589
                   Hello hello = ctx.getBean("hello", Hello.class);
590
                   System.out.println(hello);
591
592
             }
593
594
        12)결과
595
596
             Hello.toString() 시작
597
             Hello.toString() 종료
598
             Hello.toString() 실행 시간: 50 ms
599
             Hello [name=Spring, age=25]
600
601
602
     13. Lab: @Aspect 어노테이션 기반의 AOP 구현
603
        1)New > Spring Legacy Project > Simple Projects > Simple Spring Maven
604
           Project Name: AopDemo2
605
606
        2)Create Package: src/main/java/info.javaexpert
607
608
        3)info.javaexpert.Student.java
609
           package info.javaexpert;
610
611
           import java.util.ArrayList;
612
613
           import org.springframework.beans.factory.DisposableBean;
614
           import org.springframework.beans.factory.InitializingBean;
615
616
           public class Student{
617
             private String name;
618
             private int age;
619
             private int grade;
620
             private int classNum;
621
             public String getName() {
622
                return name;
623
             }
624
             public void setName(String name) {
625
                this.name = name;
626
             }
627
             public int getAge() {
628
                return age;
629
630
             public void setAge(int age) {
631
                this.age = age;
632
633
             public int getGrade() {
634
                return grade;
635
636
             public void setGrade(int grade) {
637
                this.grade = grade;
638
639
             public int getClassNum() {
640
                return classNum;
641
             }
642
             public void setClassNum(int classNum) {
643
                this.classNum = classNum;
644
645
             public void getStudentInfo(){
```

```
System.out.println("Name: " + this.name);
646
647
                System.out.println("Age: " + this.age);
                System.out.println("Grade: " + this.grade);
648
649
                System.out.println("Class: " + this.classNum);
650
             }
           }
651
652
653
        4)info.javaexpert.Worker.java
654
           package info.javaexpert;
655
656
           public class Worker {
657
             private String name;
658
             private int age;
659
             private String job;
660
             public String getName() {
661
                return name;
662
663
             public void setName(String name) {
664
                this.name = name;
665
666
             public int getAge() {
667
                return age;
668
669
             public void setAge(int age) {
670
                this.age = age;
671
672
             public String getJob() {
673
                return job;
674
675
             public void setJob(String job) {
676
                this.job = job;
677
678
             public void getWorkerInfo(){
679
                System.out.println("Name : " + this.name);
                System.out.println("Age : " + this.age);
680
                System.out.println("Job : " + this.job);
681
682
             }
           }
683
684
685
        5)pom.xml : 아래 코드 추가
686
           <!-- AOP -->
687
           <dependency>
688
              <groupId>org.aspectj</groupId>
689
              <artifactId>aspectjweaver</artifactId>
690
              <version>1.8.10</version>
691
           </dependency>
692
693
        6)info.javaexpert.LogAop.java
694
           package info.javaexpert;
695
696
           import org.aspectj.lang.ProceedingJoinPoint;
697
           import org.aspectj.lang.annotation.Around;
698
           import org.aspectj.lang.annotation.Aspect;
699
           import org.aspectj.lang.annotation.Before;
700
           import org.aspectj.lang.annotation.Pointcut;
701
702
           @Aspect
703
           public class LogAop {
704
705
             @Pointcut("within(info.javaexpert.*)*")
706
             private void pointcutMethod(){}
707
708
             @Around("pointcutMethod()")
709
             public Object loggerAop(ProceedingJoinPoint joinpoint) throws Throwable{
710
                String signatureStr = joinpoint.getSignature().toShortString();
                System.out.println(signatureStr + " is start.");
711
712
                long start = System.currentTimeMillis();
```

```
714
                try{
715
                  Object obj = joinpoint.proceed();
716
                  return obj;
717
                }finally{
718
                  long end = System.currentTimeMillis();
719
                  System.out.println(signatureStr + " is finished.");
                  System.out.println(signatureStr + " 경과시간 : " + (end - start));
720
721
                }
722
             }
723
724
             @Before("within(kr.co.javaexpert.*)*")
725
             public void beforeAdvice(){
726
                System.out.println("Called beforeAdvice()");
727
             }
728
          }
729
730
        7)src/main/resources/beans.xml
731
           <?xml version="1.0" encoding="UTF-8"?>
          <beans xmlns="http://www.springframework.org/schema/beans"</pre>
732
733
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
734
             xmlns:aop="http://www.springframework.org/schema/aop"
735
             xsi:schemaLocation="http://www.springframework.org/schema/beans
             http://www.springframework.org/schema/beans/spring-beans.xsd
736
                http://www.springframework.org/schema/aop
                http://www.springframework.org/schema/aop/spring-aop-3.2.xsd">
737
738
             <bean id="logAop" class="info.javaexpert.LogAop" />
739
740
             <aop:config>
741
                <aop:aspect id="logger" ref="logAop">
742
                  <aop:pointcut expression="within(info.javaexpert.*)*" id="publicM"/>
743
                  <aop:around method="loggerAop" pointcut-ref="publicM"/>
744
                </aop:aspect>
745
             </aop:config>
746
             <bean id="student" class="info.javaexpert.Student">
747
748
                roperty name="name" value="한지민" />
749
                cproperty name="age" value="15" />
750
                cproperty name="grade" value="3" />
751
                cproperty name="classNum" value="5" />
752
             </bean>
753
754
             <bean id="worker" class="info.javaexpert.Worker">
755
                roperty name="name" value="설운도" />
756
                cproperty name="age" value="50" />
                <property name="job" value="개발자" />
757
758
             </bean>
759
          </beans>
760
761
        8)info.javaexpert.MainClass.java
762
          package info.javaexpert;
763
764
          import org.springframework.context.support.AbstractApplicationContext;
765
          import org.springframework.context.support.GenericXmlApplicationContext;
766
767
          public class MainClass {
             public static void main(String[] args) {
768
769
                AbstractApplicationContext context = new
                GenericXmlApplicationContext("classpath:beans.xml");
770
                Student student = context.getBean("student", Student.class);
771
                student.getStudentInfo();
772
773
                Worker worker = context.getBean("worker", Worker.class);
774
                worker.getWorkerInfo();
775
776
                context.close();
```

```
778
779
780
       9)결과
          Student.getStudentInfo() is start.
781
782
          Called beforeAdvice()
783
          Name: 한지민
          Age: 15
784
          Grade: 3
785
          Class: 5
786
787
          Student.getStudentInfo() is finished.
788
          Student.getStudentInfo() 경과시간: 12
789
          Worker.getWorkerInfo() is start.
790
          Called beforeAdvice()
791
          Name: 설운도
792
          Age: 50
793
          Job:개발자
794
          Worker.getWorkerInfo() is finished.
795
          Worker.getWorkerInfo() 경과시간: 7
796
797
798
     14. Lab
799
       1)Aspect 클래스 정보
800
          -클래스명 : LoggingAspect.java
801
          -클래스 기능 : 이 Aspect 클래스는 4가지 유형의 Advice와 Pointcut을 설정하여 Target 객체의
          파라미터와 리턴값, 예외 발생 시 예외 메시지를 출력하는 기능을 제공
802
          -Advice 유형: Before, AfterReturning, AfterThrowing, After
803
          -구현 메소드명: before(JoinPoint joinPoint), afterReturing(JoinPoint joinPoint, Object ret),
          afterThrowing(JoinPoint joinPoint, Throwable ex), afterFinally(JoinPoint joinPoint)
804
805
       2)Aspect 클래스 선언 및 설정
          -클래스 선언부에 @Aspect 어노테이션을 정의한다.
806
807
          -이 클래스를 Aspect로 사용하려면 Bean으로 등록해야 하므로 @Component 어노테이션도 함께 정의한다.
808
809
          package info.javaexpert;
810
811
          import org.aspectj.lang.JoinPoint;
812
813
          @Component
814
          @Aspect
815
          public class LoggingAspect {
816
817
818
          <context:component-scan base-package="info.javaexpert" />
819
820
       3)XML 설정파일에 <aop:aspectj-autoproxy /> 선언
821
          -이 선언은 Bean으로 등록된 클래스 중에서 @Aspect가 선언된 클래스를 모두 Aspect로 자동 등록해주는
          역할을 한다.
822
823
          <aop:aspectj-autoproxy />
824
825
       4)AopDemo4 Project 생성
826
          -Spring Legacy Project > Simple Maven Project
827
828
       5)info.javaexpert package 생성
829
          -/src/main/java/info.javaexpert
830
831
       6)pom.xml에 Aspectj 종속성 추가 및 설치
832
833
            <dependency>
834
               <groupId>org.aspectj</groupId>
835
               <artifactId>aspectjweaver</artifactId>
836
               <version>1.8.10</version>
837
            </dependency>
838
839
          -Maven Install
840
```

}

```
841
        7)/src/main/java/info.javaexpert.LoggingAspect.java 생성
842
843
             package info.javaexpert;
844
845
             import org.aspectj.lang.JoinPoint;
846
             import org.aspectj.lang.annotation.After;
847
             import org.aspectj.lang.annotation.AfterReturning;
848
             import org.aspectj.lang.annotation.AfterThrowing;
849
             import org.aspectj.lang.annotation.Aspect;
850
             import org.aspectj.lang.annotation.Before;
851
             import org.springframework.stereotype.Component;
852
             @Component
853
             @Aspect
854
             public class LoggingAspect {
855
                @Before("execution(public * info.javaexpert..*(..))")
856
                public void before(JoinPoint joinPoint) {
857
                  String signatureString = joinPoint.getSignature().getName();
                  System.out.println("@Before [ " + signatureString + " ] 메서드 실행 전처리 수행");
858
859
                  for (Object arg : joinPoint.getArgs()) {
860
                     System.out.println("@Before [ " + signatureString + " ] 아큐먼트 " + arg);
861
                  }
862
                }
863
                  @AfterReturning(pointcut="execution(public * info.javaexpert..*(..))", returning="ret")
864
                public void afterReturning(JoinPoint joinPoint, Object ret) {
865
                  String signatureString = joinPoint.getSignature().getName();
                  System.out.println("@AfterReturing [ " + signatureString + " ] 메서드 실행 후처리 수행");
866
867
                  System.out.println("@AfterReturing [ " + signatureString + " ] 리턴값=" + ret);
868
869
                }
870
871
                  @AfterThrowing(pointcut="execution(public * info.javaexpert..*(..))",
872
                        throwing="ex")
873
                public void afterThrowing(JoinPoint joinPoint, Throwable ex) {
874
                  String signatureString = joinPoint.getSignature().getName();
                  System.out.println("@AfterThrowing [ " + signatureString + " ] 메서드 실행 중 예외 발생");
875
                  System.out.println("@AfterThrowing [ " + signatureString + " ] 예외=" +
876
                  ex.getMessage());
877
                }
878
879
                  @After("execution(public * info.javaexpert..*(..))")
088
                public void afterFinally(JoinPoint joinPoint) {
881
                  String signatureString = joinPoint.getSignature().getName();
882
                  System.out.println("@After [ " + signatureString + " ] 메서드 실행 완료");
883
                }
884
             }
885
886
        8)beans.xml 파일 생성
887
          -/src/main/resources/beans.xml
888
          -Namespace Tab에서 'aop'와 'context' 체크할 것
889
890
             <?xml version="1.0" encoding="UTF-8"?>
891
             <beans xmlns="http://www.springframework.org/schema/beans"</pre>
892
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
893
                xmlns:aop="http://www.springframework.org/schema/aop"
894
                xmlns:context="http://www.springframework.org/schema/context"
895
                xsi:schemaLocation="http://www.springframework.org/schema/beans
                http://www.springframework.org/schema/beans/spring-beans.xsd
896
                  http://www.springframework.org/schema/context
                  http://www.springframework.org/schema/context/spring-context-3.2.xsd
897
                  http://www.springframework.org/schema/aop
                  http://www.springframework.org/schema/aop/spring-aop-3.2.xsd">
898
899
                <context:component-scan base-package="info.javaexpert" />
900
                <aop:aspectj-autoproxy />
901
             </beans>
902
        9)Target 객체 생성
903
```

```
904
          -/src/main/java/info.javaexpert.Hello.java
905
906
             package info.javaexpert;
907
908
             import org.springframework.beans.factory.annotation.Value;
909
             import org.springframework.stereotype.Component;
910
911
             @Component("hello")
912
             public class Hello {
               @Value("Spring")
913
914
               private String name;
915
916
               @Value("25")
917
               private int age;
918
919
               @Override
               public String toString() {
920
921
                  return String.format("Hello [name=%s, age=%s]", name, age);
922
               }
923
924
               public void calculation(){
925
                  System.out.println(5 / 0);
926
927
             }
928
929
        10)테스트 클래스 작성
930
          -src/main/java/info.javaexpert.MainClass.java
931
932
             package info.javaexpert;
933
934
             import org.springframework.context.ApplicationContext;
935
             import org.springframework.context.support.GenericXmlApplicationContext;
936
937
             public class MainClass {
938
               public static void main(String[] args) {
                  ApplicationContext ctx = new GenericXmlApplicationContext("classpath:beans.xml");
939
940
                  Hello hello = ctx.getBean("hello", Hello.class);
941
942
                  System.out.println(hello);
943
                  hello.calculation();
944
               }
945
             }
946
947
       11)결과
948
             @Before [toString]메서드 실행 전처리 수행
             @After [ toString ] 메서드 실행 완료
949
950
             @AfterReturing [ toString ] 메서드 실행 후처리 수행
             @AfterReturing [ toString ] 리턴값=Hello [name=Spring, age=25]
951
952
             Hello [name=Spring, age=25]
953
             @Before [ calculation ] 메서드 실행 전처리 수행
954
             @After [ calculation ] 메서드 실행 완료
955
             @AfterThrowing [ calculation ] 메서드 실행 중 예외 발생
956
             @AfterThrowing [ calculation ] 예외=/ by zero
957
958
959
     15. AspectJ Pointcut 표현식
960
       1)표현식은 Pointcut 지시자를 이용하여 작성
961
962
       2)가장 대표적인 지시자는 execution()이다.
963
964
       3)Pointcut 을 지정할 때 사용하는 표현식으로 AspectJ 문법을 사용한다.
965
          -*: 모든
966
          -.:현재
967
          -..: 0개 이상
968
969
       4)execution
970
          -Usage
```

```
971
            execution([접근제한자 패턴] 리턴타입패턴 [타입패턴.] 이름패턴(파라미터타입패턴 | "..", ...)
            [throws 예외패턴])
972
            --접근제한자 패턴: public, private과 같은 접근 제한자, 생략가능
 973
            --리턴타입패턴 : 리턴값의 타입 패턴
 974
            --타입패턴 : 패키지와 클래스 이름에 대한 패턴, 생략가능. 사용할 때 "."를 사용해 연결함.
975
            --이름패턴 : 메소드 이름 타입 패턴
976
            --파라미터 타입패턴 : 파라미터의 타입 패턴을 순서대로 넣을 수 있다. 와일드카드를 이용해서 파라미터
            갯수에 상관없는 패턴을 만들 수 있다.
977
            --예외패턴: 예외 이름 패턴
978
 979
          -예
980
            "execution(* aspects.trace.demo.*.*(..))"
981
            -* : Any return type
            -aspects.trace.demo: package
982
 983
            -*: class
            -*: method
984
985
            -(..): Any type and number of arguments
 986
          -execution(* hello(..))
987
988
            --hello라는 이름을 가진 메소드를 선정
989
            --파라미터는 모든 종류를 다 허용
990
991
          -execution(* hello())
992
            --hello 메소드 중에서 파라미터가 없는 것만 선택함.
993
 994
          -execution(* info.javaexpert.service.UserServiceImpl.*(..))
 995
            --info.javaexpert.service.UserServiceImpl 클래스를 직접 지정
996
            --이 클래스가 가진 모든 메소드를 선택
997
998
          -execution(* info.javaexpert.user.service.*.*(..))
999
            --info.javaexpert.user.service 패키지의 모든 클래스에 적용
1000
            --하지만 서브패키지의 클래스는 포함하지 않는다.
1001
1002
          -execution(* info.javaexpert.user.service..*.*(..))
1003
            --info.javaexpert.user.service 패키지의 모든 클래스에 적용
            --그리고 '..'를 사용해서 서브패키지의 모든 클래스까지 포함
1004
1005
          -execution(* *.. Target.*(..))
1006
1007
            --패키지에 상관없이 Target이라는 이름의 모든 클래스에 적용
1008
            --다른 패키지의 같은 이름의 클래스가 있어도 적용이 된다는 점에 유의해야 함.
1009
          @Pointcut("executeion(public void get*(..))"): public void인 모든 get메소드
1010
1011
          @Pointcut("executeion(* info.javaexpert.*.*())"): info.javaexpert 패키지에 파라미터가 없는 모든
          메소드
          @Pointcut("executeion(* info.javaexpert..*.*())"): info.javaexpert 패키지 & kr.co.javaexpert
1012
          하위 패키지에 파라미터가 없는 모든 메소드
1013
          @Pointcut("executeion(* info.javaexpert.Worker.*())") : info.javaexpert.Worker 안의 모든 메소드
1014
1015
        2)within
          @Pointcut("within(info.javaexpert.*)"): info.javaexpert 패키지 안에 있는 모든 메소드
1016
1017
          @Pointcut("within(info.javaexpert...*)"): info.javaexpert 패키지 및 하위 패키지 안에 있는 모든 메소드
          @Pointcut("within(info.javaexpert.Worker)"): info.javaexpert.Worker 모든 메소드
1018
1019
```

1021

1022

3)bean

@Pointcut("bean(student)"): student 빈에만 적용

@Pointcut("bean(*ker)"): ~ker로 끝나는 빈에만 적용