[ 06 ] AOP(Aspect Oriented Programming)

1. AOP란?
   * 프로그래밍을 하다 보면, 공통적인 기능이 많이 발생합니다. 이러한 공통 기능을 모든 모듈에 적용하기 위한 방법으로 상속을 통한 방법이 있습니다. 상속도 좋은 방법이기는 하지만 몇 가지 문제가 있습니다. 우선 JAVA에서는 다중 상속이 불가하므로 다양한 모듈에 상속기법을 통한 공통 기능 부여는 한계가 있습니다. 그리고, 기능 구현부분에 핵심 기능 코드와 공통 기능 코드가 섞여 있어 효율성이 떨어집니다.
   * 위의 상속을 통한 방법에 한계가 있어 AOP가 등장하게 되었습니다.
   * 예를 들어 쇼핑몰 A와 쇼핑몰 B가 있다고 가정해 보겠습니다. 만일 A는 구매 시 고객의 포인트 점수를 올려주지만, B의 경우 직접 결재 시 가격을 낮춰주는 정책이 존재한다고 생각해 봅시다. 이 경우 사용자가 하는 행위는 ‘쇼핑몰에서 물건을 구입한다’이지만 부가적으로 A사와 B사의 정책 차이 때문에 코드를 변경해 줘야만 합니다.
   * AOP는 사전적 의미로는 ‘측면 혹은 관점’을 의미하지만, 실제로 프로그램 개발에서 의미하는 것은 ‘비즈니스 로직은 아니지만 반드시 해야 하는 작업. 필요한 기능들’로 해석될 수 있습니다
   * **AOP방법은 핵심 기능과 공통 기능을 분리 시켜놓고, 공통 기능을 필요로 하는 핵심 기능들에서 사용하는 방식 입니다.**
   * 특정 시스템 내에서 이러한 예로는 보안이나 성능 모니터링과 같은 작업을 들 수 있습니다. 보안 검증이 된 사람에 의해서만 특정 비즈니스 로직이 이뤄져야 하지만 그 자체가 고객의 비즈니스는 아닙니다. 그 자체가 목적은 아니고 시스템의 완성도를 높여주는 역할을 합니다.
2. AOP와 관련된 용어
   * **Aspect** : 공통 기능. 예를 들어 로깅같은 기능 자체에 대한 용어
   * **Advice** : 공통기능을 구현한 객체 메소드
   * **Join Point** : 핵심기능. 공통 기능을 적용할 수 있는 대상.
   * **PointCuts** : Join Point의 부분으로 실제 Advice를 적용해야 되는 부분
   * Proxy : Advice가 적용되었을 때 만들어지는 객체
   * Weaving : Advice와 target이 결합되어서 프록시 객체를 만드는 과정(advice를 핵심기능에 적용하는 행위)
3. 스프링에서 AOP 구현
   * AOP의 구현은 proxy를 이용한다.
   * Weaving 방식은 두가지
     1. XML을 이용
     2. @Aspect 어노테이션 이용
4. XML 기반의 AOP 구현
   * 작업순서
     1. Pom.xml에서 의존 설정

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.7.4</version>

</dependency>

<dependency>

<groupId>cglib</groupId>

<artifactId>cglib</artifactId>

<version>2.2</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>1.18.24</version>

<scope>provided</scope>

</dependency>

* + 1. Proxy 클래스 제작(공통기능의 클래스 제작 – Advice 역할 클래스)

**public** **class** ProxyClass {

**public** Object aroundAdvice(ProceedingJoinPoint joinPoint) **throws** Throwable {//joinPoint:핵심기능

// aroundAdvice : 핵심기능 전후로 공통기능 수행

System.***out***.println("★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★");

String signatureName = joinPoint.getSignature().toShortString(); // 핵심기능메소드명

System.***out***.println(signatureName + "가 시작되었습니다");

**long** startTime = System.*currentTimeMillis*(); // 시작시점

**try** {

Object obj = joinPoint.proceed(); // 실제 핵심기능 실행

**return** obj;

} **finally** {

**long** endTime = System.*currentTimeMillis*(); // 핵심기능 수행 후 시점

System.***out***.println(signatureName+"가 수행끝났습니다. 경과 시간 : "+(endTime-startTime));

}

}

**public** **void** beforeAdvice() {

System.***out***.println("☆ ☆ ☆ 이 공통기능 수행하고 알아서 핵심기능 수행하고 끝");

}

**public** **void** afterAdvice() {

System.***out***.println("핵심기능 수행 후 이 공통기능 수행");

}

**public** **void** afterReturningAdvice() {

System.***out***.println("정상적으로(예외발생되지 않고) 핵심기능이 수행된 후 이 공통기능 수행");

}

**public** **void** afterThrowingAdvice() {

System.***out***.println("예외가 발생된 핵심기능이 수행된 후 이 공통기능 수행");

}

1. }XML 설정 파일에 Aspect 설정

<bean id=*"apectClass"* class=*"com.tj.ex1.AspectClass"*/>

<aop:config>

<aop:aspect id=*"aroundAspect"* ref=*"apectClass"*>

<aop:pointcut expression=*"within(com.tj.ex1.\*)"* id=*"aroundM"*/>

<aop:around method=*"aroundAdvice"* pointcut-ref=*"aroundM"*/>

</aop:aspect>

</aop:config>

* + Advice의 종류

<aop:before> 핵심기능 메소드 실행 전에 advice 실행

<aop:after-returning> 정상적으로 핵심기능 메소드 실행 후에 advice 실행

<aop:after-throwing> 핵심기능 메소드 실행 중 exception 발생시 advice 실행

<aop:after> 핵심기능 메소드 실행 후( exception이 발생여부에 상관없이 반드시 ) advice 실행

<aop:around> 핵심기능 메소드 실행 전/후 및 exception 발생시 advice 실행 (가장 광범위하게 사용)

* + 예제

Student.java

@Data

**public** **class** Student {

**private** String name;

**private** **int** age;

**private** **int** gradeNum;

**private** **int** classNum;

**public** **void** getStudentInfo() { // 호출될 핵심기능(타켓메소드=비즈니스로직

System.***out***.println("이름 : " + name);

System.***out***.println("나이 : " + age);

System.***out***.println("학년 : " + gradeNum);

System.***out***.println("반 : " + classNum);

}

}

**Worker.java**

Data

**public** **class** Worker {

**private** String name;

**private** **int** age;

**private** String job;

**public** **void** getWorkerInfo() { // 호출될 핵심기능(=타겟메소드=비즈니스로직)

System.***out***.println("이름 : "+name);

System.***out***.println("나이 : "+age);

System.***out***.println("직업 : "+job);

}

}

**MainClass.java**

**public** **class** MainClass {

**public** **static** **void** main(String[] args) {

String location = "classpath:applicationCTX1.xml";

AbstractApplicationContext ctx =

**new** GenericXmlApplicationContext(location);

Student student = ctx.getBean("student", Student.**class**);

Worker worker = ctx.getBean("worker",Worker.**class**);

student.getStudentInfo();

worker.getWorkerInfo();

ctx.close();

}

}

**AspectClass.java**

**package** com.tj.ex1;

**import** org.aspectj.lang.ProceedingJoinPoint;

**public** **class** AspectClass { // aspect

**public** Object aroundAdvice(ProceedingJoinPoint joinpoint) **throws** Throwable {//공통기능

System.***out***.println("공통기능 시작");

String signture = joinpoint.getSignature().toShortString();

System.***out***.println(signture+"is start");

**long** st = System.*currentTimeMillis*(); //시작시점시간

**try** {

Object obj = joinpoint.proceed();//핵심기능 수행

**return** obj;

} **finally** {

**long** et = System.*currentTimeMillis*(); // 종료시점 시간

System.***out***.println(signture+"수행하는데 걸린 시간 :"+(et-st));

System.***out***.println("공통기능 수행 끝");

}

}

**public** **void** beforeAdvice() {

System.***out***.println("핵심기능 수행전 before공통기능 수행함");

}

**public** **void** afterAdvice() {

System.***out***.println("Worker 핵심기능 후 공통기능 수행");

}

**public** **void** afterReturningAdvice() {

System.***out***.println("핵심기능이 예외없이 수행 후 공통기능 수행");

}

**public** **void** afterThrowingAdvice() {

System.***out***.println("핵심기능에 예외발생 후 공통기능 수행");

}

}

**applicationCTX.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:aop=*"http://www.springframework.org/schema/aop"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.1.xsd"*>

<bean id=*"proxyClass"* class=*"com.lec.ch05.ex1.ProxyClass"*/>

<aop:config>

<aop:aspect id=*"aroundAspect"* ref=*"proxyClass"*>

<aop:pointcut expression=*"within(com.lec.ch05.ex1.\*)"* id=*"aroundM"*/>

<aop:around method=*"aroundAdvice"* pointcut-ref=*"aroundM"*/>

</aop:aspect>

<aop:aspect id=*"beforeAspect"* ref=*"proxyClass"*>

<aop:pointcut expression=*"within(com.lec.ch05.ex1.Worker)"* id=*"beforeM"*/>

<aop:before method=*"beforeAdvice"* pointcut-ref=*"beforeM"*/>

</aop:aspect>

<aop:aspect id=*"afterAspect"* ref=*"proxyClass"*>

<aop:pointcut expression=*"execution(\* com.lec.ch05.ex1.Worker.get\*())"* id=*"afterM"*/>

<aop:after method=*"afterAdvice"* pointcut-ref=*"afterM"*/>

</aop:aspect>

<aop:aspect id=*"afterReturnAspect"* ref=*"proxyClass"*>

<aop:pointcut expression=*"bean(stude\*)"* id=*"afterReturnM"*/>

<aop:after-returning method=*"afterReturningAdvice"* pointcut-ref=*"afterReturnM"*/>

</aop:aspect>

</aop:config>

<bean id=*"student"* class=*"com.lec.ch05.ex1.Student"*>

<property name=*"name"* value=*"홍길동"*/>

<property name=*"age"* value=*"31"*/>

<property name=*"gradeNum"* value=*"3"*/>

<property name=*"classNum"* value=*"1"*/>

</bean>

<bean id=*"worker"* class=*"com.lec.ch05.ex1.Worker"*>

<property name=*"name"* value=*"홍성실"*/>

<property name=*"age"* value=*"40"*/>

<property name=*"job"* value=*"개발자"*/>

</bean>

</beans>

1. @Aspect를 이용한 AOP구현
   * 작업순서
     1. 의존 설정
     2. @Aspect 어노테이션을 이용한 Aspect class 제작

@pointcut

@Around @Before @AfterReurning @AfterThrowing @After

* + 1. XML 설정 파일에 <aop:aspectj-autoproxy/>설정
  + 예제

**Student.java**

@Data

**public** **class** Student2 {

**private** String name;

**private** **int** age;

**private** **int** gradeNum;

**private** **int** classNum;

**public** **void** getStudentInfo() { // 호출될 핵심기능(타켓메소드=비즈니스로직)

System.***out***.println("이름 : " + name);

System.***out***.println("나이 : " + age);

System.***out***.println("학년 : " + gradeNum);

System.***out***.println("반 : " + classNum);

}

}

**Worker.java**

@Data

**public** **class** Worker2 {

**private** String name;

**private** **int** age;

**private** String job;

**public** **void** getWorkerInfo() { // 호출될 핵심기능(=타겟메소드=비즈니스로직)

System.***out***.println("이름 : "+name);

System.***out***.println("나이 : "+age);

System.***out***.println("직업 : "+job);

}

}

**AspectClass.java**

@Aspect

**public** **class** ProxyClass2 {

// @Pointcut("within(com.lec.ch05.ex2.\*)")

// private void aroundM() {}

// @Around("aroundM()")

@Around("within(com.lec.ch05.ex2.\*)")

**public** Object aroundAdvice(ProceedingJoinPoint joinPoint) **throws** Throwable {

System.***out***.println("☆ 공통기능 around Advice 시작");

String signatureName = joinPoint.getSignature().toShortString();

System.***out***.println(signatureName+" 기능 시작");

**try** {

Object obj = joinPoint.proceed();

**return** obj;

} **finally** {

System.***out***.println(signatureName+" 기능 종료 ☆ ");

}

}

@Before("within(com.lec.ch05.ex2.Worker2)")

**public** **void** beforeAdvice() {

System.***out***.println("핵심기능 수행 전 beforeAdvice()");

}

@After("execution(\* com.lec.ch05.ex2.Student2.\*())")

**public** **void** afterAdvice() {

System.***out***.println("핵심기능 수행 후 afterAdvice()");

}

@AfterThrowing("within(com.lec.ch05.ex2.Student2)")

**public** **void** afterthrowing() {

System.***out***.println("예외가 발생한 핵심기능 수행 후 afterthrowing()");

}

@AfterReturning("execution(public void com.lec.ch05.ex2.Student2.getStudentInfo())")

**public** **void** afterReturing() {

System.***out***.println("예외가 발생되지 않는 핵심기능 수행후 afterReturing()");

}

}

**applicationCTX2.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:aop=*"http://www.springframework.org/schema/aop"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.1.xsd"*>

<aop:aspectj-autoproxy/>

<bean id=*"proxyClass2"* class=*"com.lec.ch05.ex2.ProxyClass2"*/>

<bean id=*"student2"* class=*"com.lec.ch05.ex2.Student2"*>

<property name=*"name"* value=*"홍길동"*/>

<property name=*"age"* value=*"20"*/>

<property name=*"gradeNum"* value=*"4"*/>

<property name=*"classNum"* value=*"3"*/>

</bean>

<bean id=*"worker2"* class=*"com.lec.ch05.ex2.Worker2"*>

<property name=*"name"* value=*"김일꾼"*/>

<property name=*"age"* value=*"40"*/>

<property name=*"job"* value=*"가수"*/>

</bean>

</beans>

**MainClass.java**

**public** **class** TestMain {

**public** **static** **void** main(String[] args) {

String location = "classpath:applicationCTX2.xml";

AbstractApplicationContext ctx = **new** GenericXmlApplicationContext(location);

Student2 student2 = ctx.getBean("student2", Student2.**class**);

Worker2 worker2 = ctx.getBean("worker2", Worker2.**class**);

student2.getStudentInfo();

worker2.getWorkerInfo();

ctx.close();

}

}

1. AspectJ Pointcut 표현식
   * Pointcut을 지정할 때 사용하는 표현식으로 AspectJ 문법을 사용 합니다.

\* : 모든

. : 현재

.. : 0개 이상

1. execution

@Pointcut("execution(public void get\*(..))") // public void인 모든 get메소드

@Pointcut("execution(\* com.lec.ch05.ex.\*.\*())") // com.ch.ex 패키지에 파라미터가 없는 모든 메소드

@Pointcut("execution(\* com.lec.ch05.ex..\*.\*())")

// com.ch.ex 패키지 & com.ch.ex 하위 패키지에 파라미터가 없는 모든 메소드

@Pointcut("execution(\* com.lec.ch05.ex.Worker.\*())") // com.ch.ex.Worker 안의 모든 메소드

1. within

@Pointcut("within(com.lec.ch05.ex.\*)") //com.ch.ex 패키지 안에 있는 모든 메소드

@Pointcut("within(com.lec.ch05.ex..\*)") // com.ch.ex 패키지 및 하위 패키지 안에 있는 모든 메소드

@Pointcut("within(com.lec.ch05.ex.Worker)") // com.ch.ex.Worker 모든 메소드

1. bean

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

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