관점지향프로그래밍(AOP) , 횡단관심사 Cross-cutting Concern
Aspect Object Programming

Advice: 부가기능( 다른 관점의 코드)
target: 핵심기능이 있는 객체
weaving: 꿰매기

| 용 어 | 설 명 |
| --- | --- |
| target | advice가 추가될 객체 |
| advice | target에 동적으로 추가될 부가 기능(코드) |
| join point | advice가 추가(join)될 대상(메서드) |
| pointcut | join point들을 정의한 패턴. 예) execution(* com.acorn.*.*(..)) |
| proxy | target에 advice가 동적으로 추가되어 생성된 객체 |
| weaving | target에 advice를 추가해서 proxy를 생성하는 것 |

advice + target = proxy

코드의 분리

```
class MyClass {
    void aaa() {
        System.out.println("[before]{");
        System.out.println("aaa() is called.");
        System.out.println("}[after]");
    }

    void aaa2() {
        System.out.println("[before]{");
        System.out.println("aaa2() is called.");
        System.out.println("}[after]");
    }

    void bbb() {
        System.out.println("[before]{");
        System.out.println("bbb() is called.");
        System.out.println("}[after]");
    }
}
```

```
class MyAdvice {
    void invoke(Method m, Object obj, Object... args) throws Exceptior
        System.out.println("[before]{");
        m.invoke(obj,args);  // 메서드 호출
        System.out.println("}[after]");
    }
}
```

```
class MyClass {
    void aaa() {
        System.out.println("aaa() is called.");
    }

    void aaa2() {
        System.out.println("aaa2() is called.");
    }

    void bbb() {
        System.out.println("bbb() is called.");
    }
```

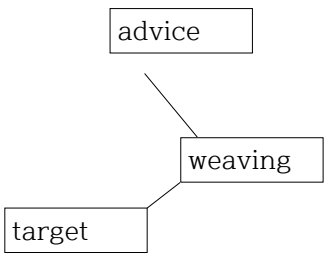
advice가 추가될 매서드를 지정하기 위한 패턴

execution( 반환타입 패키지명.클래스명.매서드명(매개변수 목록))

```
public class AroundAdvice {
    @Around("exec @Around("execute(* com.acorn.aop.*.*(**))")
    public Object methodCallLog(ProceedingJoinPoint pjp) throws Throwable {
        long start = System.currentTimeMillis();
        System.out.println("<<[start] "
                + pjp.getSignature().getName()+Arrays.deepToString(pjp.getArgs()));

        Object result = pjp.proceed();  // 메서드 호출

        System.out.println("result = " + result);
        System.out.println("[end]>> "+ (System.currentTimeMillis() - start)+"ms");
        return result; // 메서드 호출결과 반환
    }
}
```



pom.xml

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aop</artifactId>
    <version>${org.springframework-version}</version>
</dependency>

<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.9.8</version>
    <scope>runtime</scope>
</dependency>
```

root-context.xml 추가

```
<aop:aspectj-autoproxy />
<context:component-scan base-package="com.acorn.aop" />
```

```
///
public class AopTest3 {

    public static void main(String[] args) {

        ApplicationContext ac = new GenericXmlApplicationContext("file:src/main/webapp/WEB-INF/spring/**/root-context.xml");

        MyCalculator cal = (MyCalculator) ac.getBean("myCalculator");
        cal.add(5, 3);
        cal.sub(5, 3);
        System.out.println(cal.test());      //3
    }

}
```

```
//
@Component
@Aspect
public class LoginAdvice {

 @Around("execution(* com.acorn.aop.*.*(int, int))")
 public Object method( ProceedingJoinPoint pjp ) throws Throwable {
                System.out.println("before ");
                 Object  result  = pjp.proceed();
                 System.out.println( result);
                System.out.println("  after");
                return result;
 }

////
@Component
public class MyCalculator {

            public int add( int x, int y) {
                        return x+y ;
            }

            public int sub( int x, int  y) {
                        return x-y;
            }

            public int test(  ) {
                        return 3;
            }

}
```