SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

삼성 청년 SW 아카데미

Spring Framework

SAMSUNG SW ACADEMY FOR YOUTH

Spring AOP

- AOP

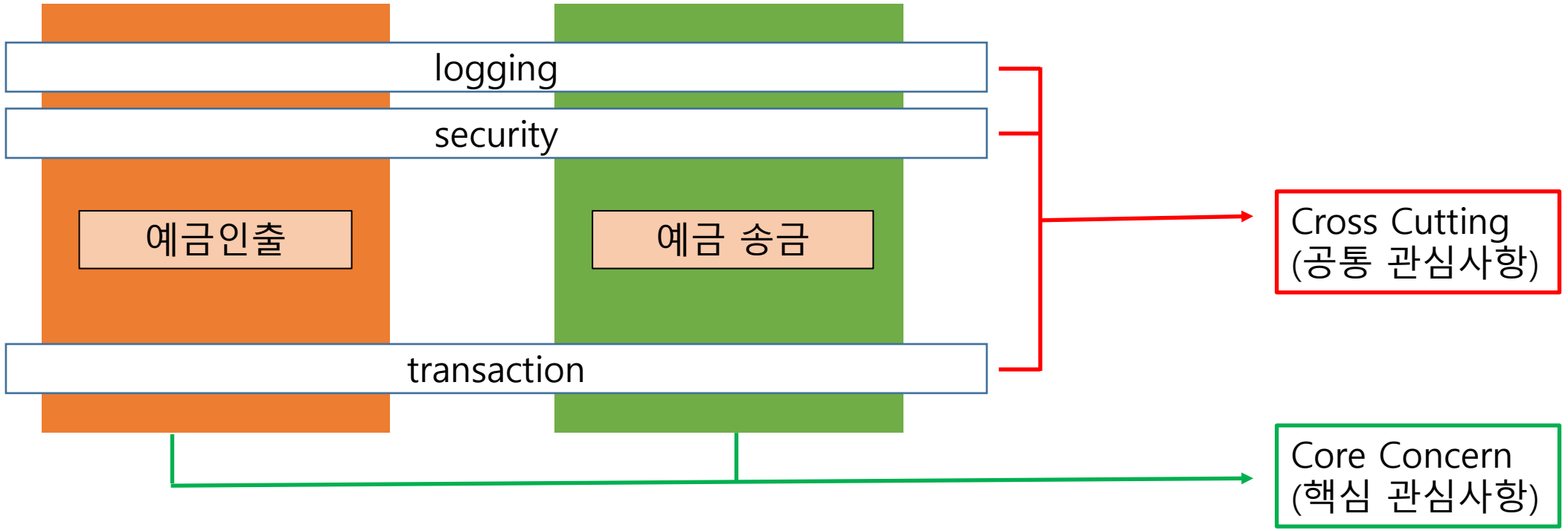
SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

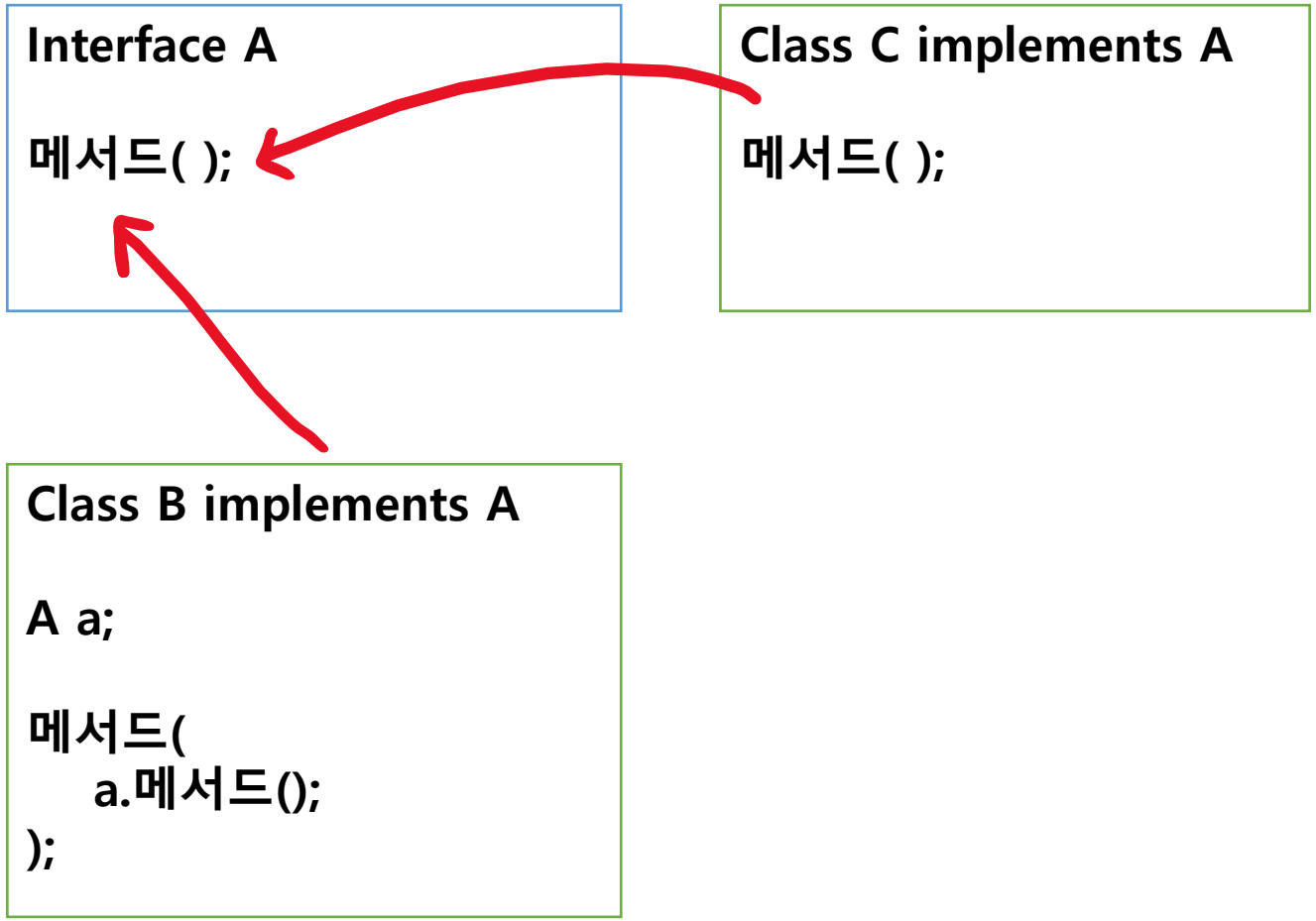
AOP
(Aspect Oriented Programming)

AOP (Aspect Oriented Programming)

- OOP에서 모듈화의 핵심 단위는 클래스인 반면, AOP 에서 모듈화의 단위는 Aspect.
- Aspect는 여러 타입과 객체에 거쳐서 사용되는 기능 (Cross Cutting, 트랜잭션 관리 등)의 모듈화
- Spring framework의 필수요소는 아니지만, AOP 프레임워크는 Spring IoC를 보완한다.



프록시 패턴

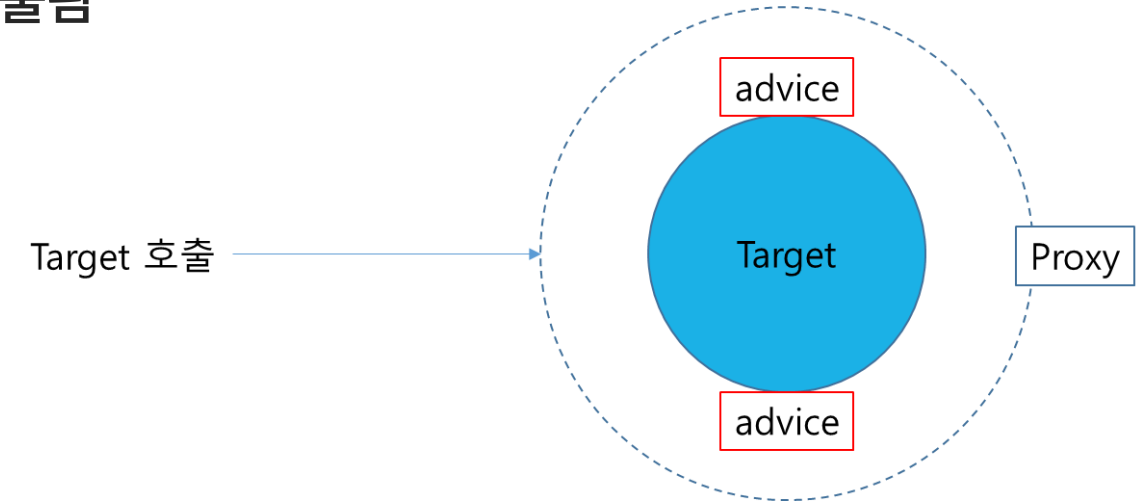


AOP 용어

용어	내용
Aspect	• 여러 클래스에 공통적으로 구현되는 관심사(Concern)의 모듈화
Join Point	• 메서드 실행이나 예외처리와 같은 프로그램 실행중의 특정 지점. • Spring에서는 메서드 실행을 의미한다.
PointCut	• Join Point에 Aspect를 적용하기 위한 조건 서술. • Aspect는 지정한 pointcut 에 일치하는 모든 join point에서 실행된다.
Advice	• 특정 조인포인트(Join Point)에서 Aspect에 의해서 취해진 행동. • Around, Before, After 등의 Advice 타입이 존재
Target 객체	• 하나 이상의 advice가 적용될 객체. • Spring AOP는 Runtime Proxy를 사용하여 구현되므로 객체는 항상 Proxy 객체가 된다.
AOP Proxy	• AOP를 구현하기 위해 AOP 프레임워크에 의해 생성된 객체. • Spring Framework에서 AOP 프록시는 JDK dynamic proxy 또는 CGLIB proxy이다.
Weaving	• Aspect를 다른 객체와 연결하여 Advice 객체를 생성. • 런타임 또는 로딩 시 수행할 수 있지만 Spring AOP는 런타임에 위빙을 수행

Spring AOP Proxy

- 실제 기능이 구현된 Target 객체를 호출하면, target이 호출 되는 것이 아니라 advice 가 적용된 Proxy 객체가 호출됨



- Spring AOP 는 기본값으로 표준 JDK dynamic proxy를 사용.
- 인터페이스를 구현한 클래스가 아닌 경우 CGLIB 프록시를 사용.

Spring AOP

- @AspectJ : 일반 Java 클래스를 Aspect로 선언하는 스타일, AspectJ 프로젝트에 의해 소개되었음
- Spring AOP 에서는 pointcut 구문 분석, 매핑을 위해서 AspectJ 라이브러리를 사용함
- 하지만 AOP runtime은 순수 Spring AOP 이며, AspectJ 에 대한 종속성은 없음

Spring AOP 시작하기 - xml

- Aspect 선언하기 - xml 방식
- Aspect는 핵심 관심사항 (Core concern)
- @Aspect annotation 또는 xml bean 등록하기

com.ssafy.pjt02.aop.ServiceAspect.java

```java
public class ServiceAspect {
    public void method1() {
        System.out.println("method1 입니다.");
    }
    public void method2() {
        System.out.println("method2 입니다.");
    }
    public void method3() {
        System.out.println("method3 입니다.");
    }
}
```

root-context.xml

```xml
<bean id="myAspect" class="com.ssafy.pjt02.aop.ServiceAspect">

</bean>
```

Spring AOP 시작하기 - xml

- Pointcut 선언
- 포인트 컷은 어떤 조인포인트를 사용할지 결정한다. Spring AOP는 메서드 실행만 지원한다
- 포인트 컷 선언은 두 내용을 포함한다.
  - 조인포인트에 대한 표현식
  - 포인트 컷의 이름

root-context.xml

```
<aop:config> <!-- 어떤 aspect가 언제 적용될지 결정 -->
    <aop:aspect ref="myAspect">
        <aop:pointcut
            expression="execution(public * ssafy.com.pjt02.service.*.doSomething())" id="mypt"/>
        <aop:before method="method1" pointcut-ref="mypt"/>
        <aop:after method="method2" pointcut-ref="mypt"/>
    </aop:aspect>
</aop:config>
```

ssafy.com.pjt02.service 패키지의 모든 클래스의 doSomething 메서드를 선택
execution : 해당 메서드가 실행되었을 경우

Spring AOP 시작하기 - xml

root-context.xml

```
<aop:config> <!-- 어떤 aspect가 언제 적용될지 결정 -->
    <aop:aspect ref="myAspect">
        <aop:pointcut
            expression="execution(public * ssafy.com.pjt02.service.*.doSomething())" id="mypt"/>
        <aop:before method="method1" pointcut-ref="mypt"/>
        <aop:after method="method2" pointcut-ref="mypt"/>
    </aop:aspect>
</aop:config>
```

advice type에 따른 메서드 지정
before : target 메서드 실행 전
after : target 메서드 실행 후

Spring AOP 시작하기 - AOP 적용 확인하기

root-context.xml

```
15    <aop:config> <!-- 어떤 aspect가 언제 적용될지 결정 -->
16        <aop:aspect ref="myAspect">
17            <aop:pointcut
18               expression="execution(public * com.ssafy.pjt02.service.*.doSomething())" id="mypt"/>
19            <aop:before method="method1" pointcut-ref="mypt"/>
20            <aop:after method="method2" pointcut-ref="mypt"/>
21        </aop:aspect>
22    </aop:config>
```

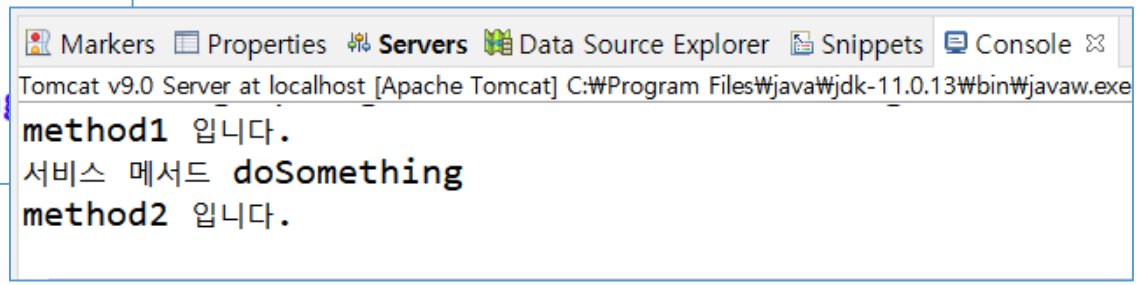
ServiceAspect.java

```
5     public void method1() {
6         System.out.println("method1 입니다.");
7     }
8     public void method2() {
9         System.out.println("method2 입니다.");
10    }
```

aop가 적용되면 eclipse에서 화살표로 적용 확인 가능하다.

BoardServiceImp.java

```
49    @Override
50    public void doSomething() {
51        System.out.println("서비스 메서드 doSomethin
52    }
```



Spring AOP 시작하기 - Annotation

- @AspectJ 활성화 - aop:aspectj-autoproxy

root-context.xml

```
<aop:aspectj-autoproxy/>

<!-- Root Context: defines shared resources visible to all other web components -->
<context:component-scan base-package="com.ssafy.pjt02.dao"/>
<context:component-scan base-package="com.ssafy.pjt02.aop"/>
<context:component-scan base-package="com.ssafy.pjt02.service"/>
```

Spring AOP 시작하기 - Annotation

- Bean 등록 및 @Aspect

com.ssafy.pjt02.aop.ServiceAspect.java

```
 9 @Component
10 @Aspect
11 public class ServiceAspect {
12
13     @Pointcut("execution(public * com.ssafy.pjt02.service.*.doSomething())")
14     public void servicepointcut(){}
15
16     @Before("servicepointcut()")
17     public void method1() {
18         System.out.println("method1 입니다.");
19     }
20
21     @After("servicepointcut()")
22     public void method2() {
23         System.out.println("method2 입니다.");
24     }
```

Aspect 클래스를 Bean 등록 하기 위해서 @Component annotation 사용
@Aspect Annotation으로 해당 클래스가 Aspect임을 선언

Spring AOP 시작하기 - Annotation

- Pointcut 설정

com.ssafy.pjt02.aop.ServiceAspect.java

```
 9 @Component
10 @Aspect
11 public class ServiceAspect {
12
13     @Pointcut("execution(public * com.ssafy.pjt02.service.*.doSomething())")
14     public void servicepointcut(){}
15
16     @Before("servicepointcut()")
17     public void method1() {
18         System.out.println("method1 입니다.");
19     }
20
21     @After("servicepointcut()")
22     public void method2() {
23         System.out.println("method2 입니다.");
24     }
```

@Pointcut annotation을 이용하여 Pointcut을 설정할 수 있다. 이때, pointcut의 이름 설정은 @Pointcut annotation이 설정된 메서드의 이름이 된다. 이 메서드는 void 유형의 body가 비어있는 메서드로 생성한다.

Spring AOP 시작하기 - Annotation

- advice type 지정

com.ssafy.pjt02.aop.ServiceAspect.java

```
 9 @Component
10 @Aspect
11 public class ServiceAspect {
12
13     @Pointcut("execution(public * com.ssafy.pjt02.service.*.doSomething())")
14     public void servicepointcut(){}
15
16     @Before("servicepointcut()")
17     public void method1() {
18         System.out.println("method1 입니다.");
19     }
20
21     @After("servicepointcut()")
22     public void method2() {
23         System.out.println("method2 입니다.");
24     }
```

@Before, @After 등의 advice type annotation을 설정하고 인자로 pointcut 이름을 넣어준다.

Advice Type

- before - target 메서드 호출 이전
- after - target 메서드 호출 이후, java exception 문장의 finally와 같이 동작
- after returning - target 메서드 정상 동작 후
- after throwing - target 메서드 에러 발생 후
- around - target 메서드 의 실행 시기, 방법, 실행 여부를 결정

Advice Type - after returning , after throwing

- after returning , after throwing은 각각 실행 결과와 에러를 인자로 받음

root-context.xml

```
<bean id="myAspect" class="com.ssafy.pjt02.aop.ServiceAspect">

</bean>
<aop:config>
    <aop:aspect ref="myAspect">
        <aop:pointcut
           expression="execution(public * com.ssafy.pjt02.service.*.doSomething2())" id="mypt"/>
        <aop:after-returning method="method3" returning="message" pointcut-ref="mypt" />
        <aop:after-throwing method="method4" throwing="err" pointcut-ref="mypt"/>
    </aop:aspect>
</aop:config>
```

returning 속성과 throwing 속성에 지정한 이름으로 각 메서드에 매개변수를 선언한다.
After returning의 경우 target이 정상 실행했을 때 실행되는 메서드 이므로, 타입은 targe의 반환형과 같아야 한다.
After Throwing의 경우, 에러가 발생하면 수행되는 메서드 이므로 타입은 Throwable

```
public void method3(String message) {
    System.out.println("msg : " + message);
    System.out.println("method3 입니다.");
}
public void method4(Throwable err) {
    System.out.println("th :" + err.getMessage());
    System.out.println("method3 입니다.");
}
```

ServiceAspect.java

Advice Type - after returning , after throwing

annotation 설정

```
@AfterReturning(pointcut = "servicepointcut()", returning = "message")
public void method3(String message) {
    System.out.println("method3 입니다.");
}
@AfterThrowing(pointcut = "servicepointcut()",throwing = "th")
public void method4(Throwable th) {
    System.out.println("method3 입니다.");
}
```

ServiceAspect.java

Advice Type - around

root-context.xml

```xml
<bean id="myAspect" class="com.ssafy.pjt02.aop.ServiceAspect">

</bean>
<aop:config>
    <aop:aspect ref="myAspect">
        <aop:pointcut
           expression="execution(public * com.ssafy.pjt02.service.*.doSomething2())" id="mypt"/>
        <aop:around method="method5" pointcut-ref="mypt"/>
    </aop:aspect>
</aop:config>
```

ServiceAspect.java

```java
public void method5(ProceedingJoinPoint jp) {
    method1();
    try {
        Object message = jp.proceed();
        method3((String)message);
    } catch (Throwable e) {
        method4(e);
    }finally {
        method2();
    }
}
```

1. around 메서드는 target 메서드의 실행을 직접 수행한다.
2. advice 내부에서 targe에 접근하기 위해서 메서드의 첫번째 인자로 ProceedingJoinPoint를 받아온다.
3. targe메서드의 실행은 proceed() 메서드를 호출한다.

그 외 나머지 target 메서드 이전, 이후에 실행할 코드들을 작성해준다.

Advice Type - around

```
@Around("servicepointcut()")
public void method5(ProceedingJoinPoint jp) {
    method1();
    try {
        Object message = jp.proceed();
        method3((String)message);
    } catch (Throwable e) {
        method4(e);
    }finally {
        method2();
    }
}
```

Pointcut Expression 패턴 예시

패턴	설명
execution(public * *(..))	public 메서드 실행
execution(* set*(..))	메서드 이름이 set으로 시작하는 메서드 실행
executioin(* com.xyz.service.AccoutService.*(..))	AccountService Interface에 정의된 모든 메서드
execution(* com.xyz.service.*.*(..))	service package에 선언된 모든 메서드
execution(* com.xyz.service..*.*(..))	service package 또는 그 하위 패키지에 선언된 모든 메서드 실행
within(com.xyz.service.*)	service package 또는 그 하위 패키지의 모든 join point(Spring AOP에서는 메서드 실행)

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

다음 방송에서 만나요!

삼성 청년 SW 아카데미