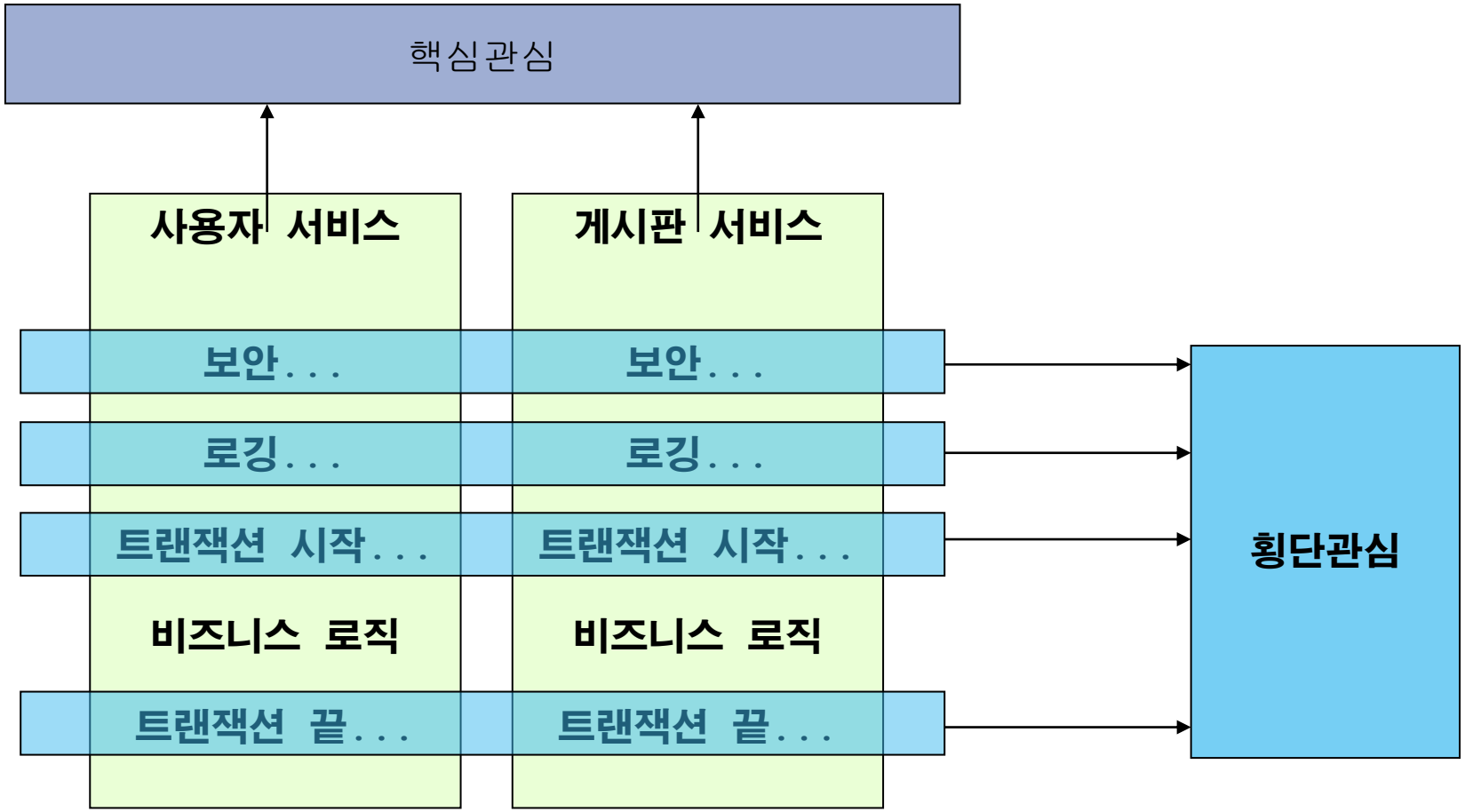
Spring AOP Programming

Aspect Oriented Programming

AOP 개요

- Aspect Oriented Programming : 관점 지향 프로그래밍
- 가장 기초가 되는 개념은 관심의 분리 (Separation of Concern)



AOP 개요

- Aspect Oriented Programming : 관점 지향 프로그래밍
- 가장 기초가 되는 개념은 관심의 분리 (Separation of Concern)
  1. 핵심 관심 : 시스템의 핵심 가치와 목적이 그대로 드러난 관심 영역
  2. 횡단 관심 : 핵심 관심 전반에 걸쳐 반복적으로 나타나는 로깅, 트랜잭션, 보안, 인증,
     리소스 풀링, 에러 체크 등의 관심 영역
  3. 관심의 분리 : 여러 핵심관심에 걸쳐 등장하는 횡단 관심을 분리하여 독립적인 모듈로 만들고
     핵심 관심이 실행되는 동안 횡단 관심을 호출하는 코드를 직접 명시하지 않고 선언적으로 처리
  4. 핵심 관심 모듈의 중간중간에 필요한 횡단 관심 모듈을 직접 호출하지 않고 위빙(Weaving)이라 불리는
     작업을 이용하여 횡단 관심 코드가 삽입되도록 만든다
  5. 핵심 관심 모듈에서는 횡단 관심 모듈이 무엇인지 인식할 필요조차 없음

AOP 개요
: AspectJ

- 1990년대 후반, Java를 확장하여 AOP를 지원하도록 만들어진 최초이며 대표적인 프레임워크

- 자바의 클래스 파일과 비슷한 Aspect를 작성한 후 AspectJ 컴파일러로 컴파일을 하면 관심 모듈의 사이사이에 횡단 관심 모듈들이 삽입되는 위빙 작업이 일어나고 이때 Java VM에서 사용할 수 있는 클래스 파일이 만들어지는 원리

- 2005년 12월, AspectJ 5.0이 발표되었고, Java 5.0에서 추가된 제네릭, 어노테이션 등을 활용하게 되면서 자바 정규 문법을 이용하여 작성이 가능하게 됨. 즉, 자바 컴파일러로 개발 가능

- Spring 2.0의 경우, AspectJ 라이브러리를 활용하여 AspectJ 5.0과 동일하게 어노테이션 해석 기능을 수행

AOP 개요
: 구성 요소

- JoinPoint (언제)
  - 횡단 관심 모듈은 코드의 아무 때나 삽입되는 것은 아니다
  - 조인포인트라 불리는 특정 시점에서만 삽입이 가능
  - 예
    - 메서드가 호출되거나 리턴되는 시점
    - 필드를 액세스하는 시점
    - 객체가 생성되는 시점
    - 예외가 던져지는 시점
    - 예외 핸들러가 작동하는 시점
    - 클래스가 초기화되는 시점 등
  - AOP 프레임워크에 따라 제공되는 조인포인트는 다르며 스프링은 메서드 조인포인트만 제공

AOP 개요
: 구성 요소

- PointCut (어디에)
  - 어느 조인포인트에 횡단 관심 모듈을 삽입할지를 결정하는 기능
  - 횡단 관심이 삽입될 특정 클래스의 특정 메서드를 정의하는 방법 정의
- Advice (or Interceptor, 무엇을)
  - 횡단 관심 모듈 (로깅, 보안, 트랜잭션 등)
- Weaving (위빙)
  - 어드바이스 (횡단 관심)를 삽입하는 과정
  - 위빙 작업이 일어나는 시간

    컴파일시 - 특별한 컴파일러 필요

    클래스 로딩시 - 특별한 클래스 로더 필요

    런타임시 - 프록시를 이용한 방법 (스프링)

AOP 개요
: 구성 요소

- Aspect (Advisor)
  1. 어디에서 무엇을 언제 할 것인가?
  2. PointCut + Advice를 정의

AOP 개요
: Spring AOP

- Spring AOP 연습
  - ProductService 객체의 findProduct (String) 메서드가
    다음 5가지 지점에 해당 Advice가 실행되도록 코드를 작성하는 연습

1) 시작 지점
2) 종료 지점
3) 예외를 보내지 않고 정상 종료되는 지점
4) 메서드 시작/종료 지점
5) 메서드에서 예외가 발생한 시점

Spring AOP
: 연습

- Dependency 추가

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <org.springframework-version>6.1.2</org.springframework-version>
</properties>

<dependencies>
  <!-- Spring Context -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${org.springframework-version}</version>
  </dependency>

  <!-- Spring Aspect -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aspects</artifactId>
    <version>${org.springframework-version}</version>
  </dependency>
</dependencies>
```

Spring AOP
: 연습

applicationContext.xml

```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"
    xmlns:context="http://www.springframework.org/schema/context"
    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-4.2.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-4.2.xsd">
    <context:annotation-config />
    <context:component-scan base-package="com.example.aoptest">
        <context:include-filter type="annotation"
            expression="org.springframework.stereotype.Repository"/>
        <context:include-filter type="annotation"
            expression="org.springframework.stereotype.Service"/>
        <context:include-filter type="annotation"
            expression="org.springframework.stereotype.Component"/>
    </context:component-scan>
    <aop:aspectj-autoproxy />
</beans>
```

Spring AOP
: 연습

- ProductService.java

```
@Service
public class ProductService {
    public ProductVo findProduct(String name) {
        System.out.println("Finding " + name + " ..." );
        return new ProductVo(name);
    }
}
```

Spring AOP
: 연습

- ProductVo.java

```
public class ProductVo {
    private String name;
    public ProductVo(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    @Override
    public String toString() {
        return "ProductVo [name=" + name + "]";
    }
}
```

Spring AOP
: 연습

- App.java

```
public class App {
    public static void main(String[] args) {
        ApplicationContext appContext =
            new ClassPathXmlApplicationContext("config/applicationContext.xml");

        ProductService pService =
            (ProductService)appContext.getBean("productService");

        pService.findProduct("SmartPhone");
    }
}
```

Spring AOP
: 연습

- Aspect 클래스 구현

```
@Aspect
@Component
public class MyAspect {

}
```

- Before Advice 구현

```
@Before( "execution(* *..*.findProduct(..))")
    public void before() {
    //메서드 시작 시점에 동작하는 Advice
    System.out.println("call [before advice]");
}
```

Spring AOP
: PointCut의 기술 방법

- 가장 많이 사용하는 execution 포인트컷은 호출되는 쪽의 메서드를 조건으로 포인트컷을 기술

```
execution( 접근자 반환타입 패키지.클래스(인터페이스).메서드(인수) throws 예외)
```

- 기술 방법
  1. 접근자 생략 가능
  2. throws 예외 생략 가능
  3. * (와일드카드) 사용 가능
  4. 패키지에서의 와일드 카드는 .. 을 사용
  5. 메서드 인수에서 .. 을 사용하면 모든 인수를 의미
  6. 패키지명.클래스명은 생략할 수 있음

Spring AOP
: 연습

- After Advice 구현

```
@After("execution(public * com.example.aoptest..Product*.findProduct(..))")
public void after() {
    //메서드 종료 시점에 호출
    System.out.println("call [after advice]");
}
```

- AfterReturning Advice 구현

```
@AfterReturning(value="execution(* *..ProductService.*(..))", returning="vo")
public void afterReturning(ProductVo vo) {
    //메서드 호출이 예외를 보내지 않고 정상 종료했을 때 동작하는 Advice
    System.out.println("call [afterReturning advice] : " + vo);
}
```

Spring AOP
: 연습

- Around Advice 구현

```
@Around("execution(* findProduct(String))")
public ProductVo around(ProceedingJoinPoint pjp) throws Throwable {
    //메서드 호출 전후에 동작하는 Advice
    System.out.println("call [around advice] : before");
    Object[] a = { "Camera" };
    ProductVo vo = (ProductVo)pjp.proceed(a);
    System.out.println("call [around advice] : after");

    return vo;
}
```

- AfterThrowing Advice 구현

```
@AfterThrowing( value="execution(* findProduct(String))", throwing="ex")
public void afterThrowing(Throwable ex) {
    System.out.println("call [afterThrowing] : " + ex.toString());
}
```