6. AOP(Aspect Oriented programming)

1. AOP의 개념?
2. XML 기반 AOP 구현

1. AOP의 개념

- **소프트웨어 개발에서 공통문제점**
  - 로깅(Logging)
  - 보안/인증(security/authentication)
  - 트랜잭션(transaction)
  - 리소스 폴링(resource)
  - 에러 검사(error checking)
  - 정책 적용(policy enforcement)
  - 멀티 쓰레드 안전 관리(Multithread safety)
  - 데이터 퍼시스턴스(data persistence)

1. AOP 개념

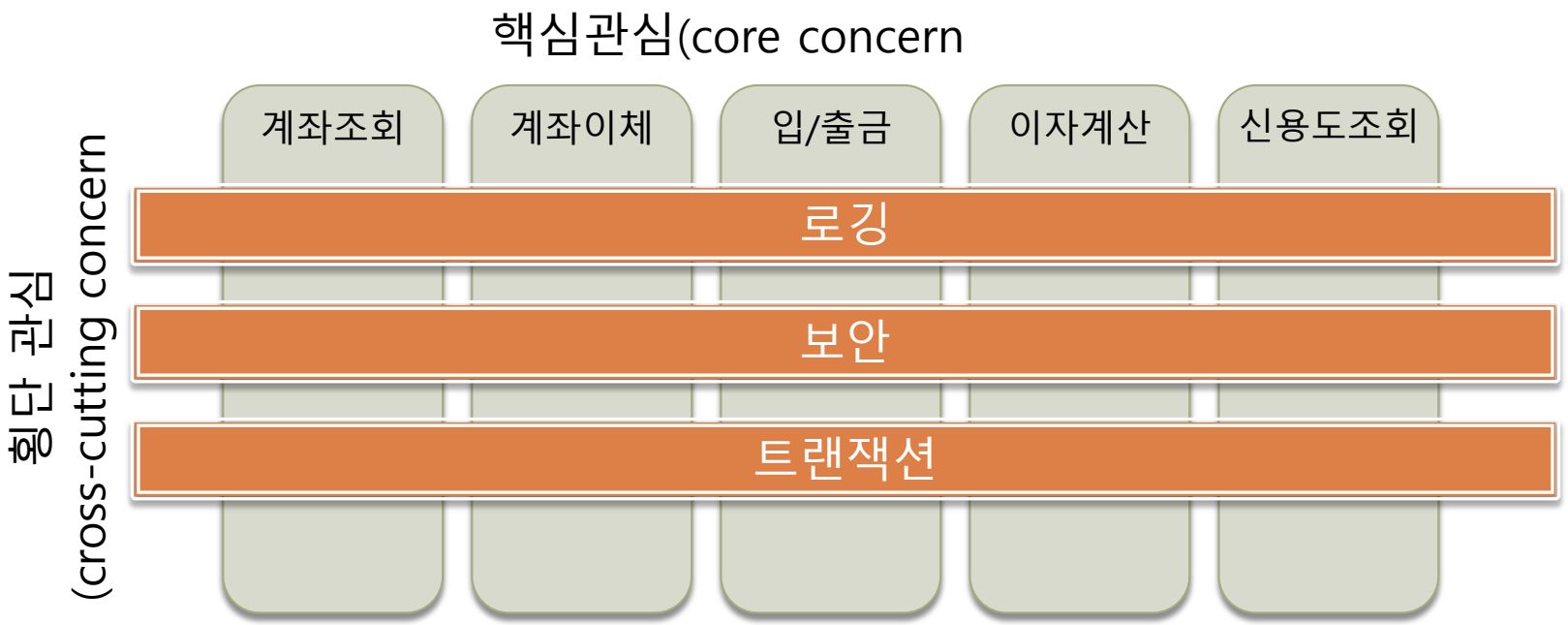
문제 영역(poroblem domain)

- 핵심관심(core concern)
  - 프로그램 작성의 핵심 가치와 목적이 드러나는 관심 영역
  - 핵심 업무 기능에 해당
  - 객체 지향 패러다임을 사용하여 클래스, 컴포넌트, 서비스로 구현
- 횡단 관심(cross cutting concern)
  - 핵심 관심에 영향을 주는 프로그램 영역,
  - 로깅, 트랜잭션, 인증처리와 같은 시스템 공통 처리 영역
  - 관점지향 패러다임을 사용하여 관점으로 구현

관심	패러다임	모듈
핵심관심	OOP(객체지향 프로그래밍)	클래스/컴포넌트/서비스
횡단관심	AOP(관점지향 프로그래밍)	관점(aspect)

1. AOP의 개념

문제 영역



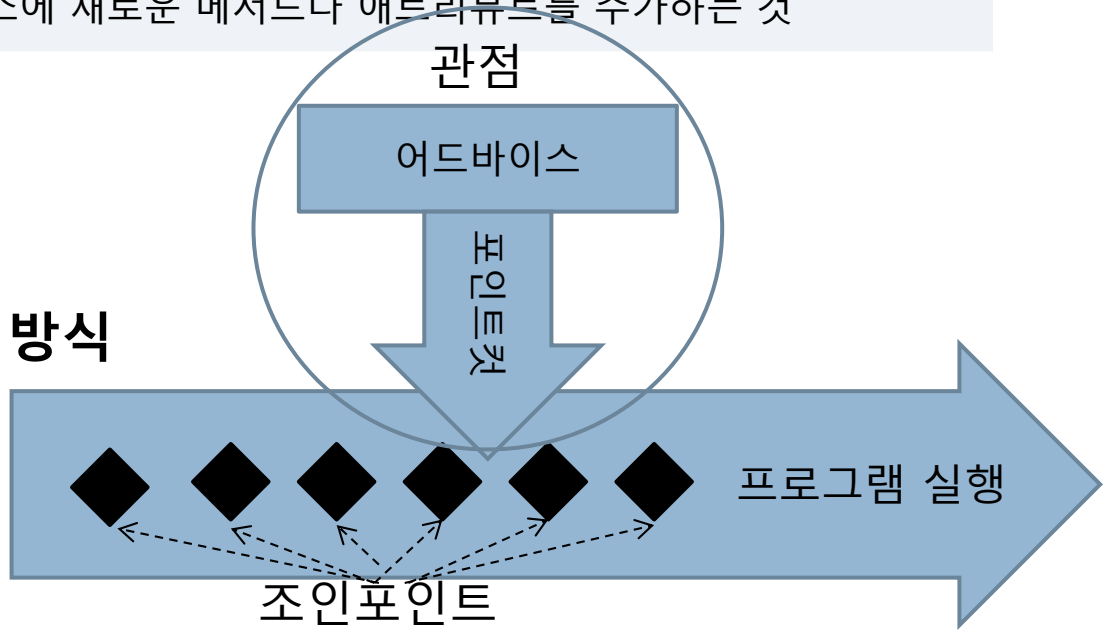
관점지향 프로그래밍

- 관심의 분리를 통해 문제 영역을 핵심 관심과 횡단 관심의 독립적인 모듈로 분해하는 프로그래밍 패러다임
- 이점
  - 관심의 분리 도출
  - 비즈니스 로직 이해도 향상
  - 생산성 향상
  - 비지니즈 코드와 횡단 관삼사들 간의 결합성 제거
  - 비즈니스 코드 재사용성 향상
  - 확장 용이

관점지향 용어

용어	설명
어드바이스(advice)	Aspect의 기능 자체로, 관점이 언제, 무엇을 하는지를 정의함
조인포인트(joinpoint)	관점이 플러그인되는 애플리케이션의 실행 위치
포인트컷(pointcut)	Advice를 적용해야되는 부분, 관점이 어드바이스하는 위치(어디서)
관점(aspect)	공통 기능 ,어드바이스와 포인트 컷 결합, 무엇을 언제, 어디서 하는지를 정의함
엮기(weaving)	Advice를 핵심 기능에 적용하는 행위, 관점을 대상 객체에 적용시키는 것, 프록시 객체(proxied object) 생성
도입(introduction)	기존 클래스에 새로운 메서드나 애트리뷰트를 추가하는 것

- **관점지향 프로그램 실행 방식**



스프링에서 AOP 구현 방법 : proxy를 이용 합니다.

스프링에서 AOP 구현 방식

2. XML 기반 AOP 구현

- **순서**
  - 의존성 설정(pom.xml)
  - 공통 기능의 클래스 제작- Advice 역할 클래스
  - xml 설정 파일에 Aspect 설정

의존성 설정

- maven repository 에서 검색하여 pom.xml에 복사

```
<!-- AOP -->
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.7.4</version>
</dependency>
```

공통 기능 클래스

```
public class LogAop {
  public Object loggerAop(ProceedingJoinPoint jointpoint) throws Throwable{
    String signatureStr=jointpoint.getSignature().toString();
    System.out.println(signatureStr + "is start");
    long st=System.currentTimeMillis();
    try{
      Object obj=jointpoint.proceed();
      return obj;
    }finally{
      long et=System.currentTimeMillis();
      System.out.println(signatureStr+"is finished");
      System.out.println(signatureStr + "경과시간: " +(et-st));
    }
  }
}
```

xml 파일 설정

- aop 네임스페이스 설정

```
<bean id="logAop" class="com.example.pgm.LogAop" />

  <aop:config>
    <aop:aspect id="logger" ref="logAop">
      <aop:pointcut id="publicM" expression="within(com.example.pgm.*)"/>
      <aop:around pointcut-ref="publicM" method="loggerAop"/>
    </aop:aspect>
  </aop:config>
```

어드바이스

- 관점의 실제 구현체로 포인터컷에 삽입되어 동작할 수 이쓴 코드
- 관점이 언제 무엇을 하는지 정의
- 어드바이스는 포인트컷과 결합하여 동작하는 시점에 따라 구분

어드바이스	설명
befor	메서드가 호출되기 전에 어드바이스 기능이 발생
after	메서드의 실행이 완료된 후에 결과와 관계없이 어드바이스 기능이 발생함
after-returning	메서드가 성공적으로 완료된 후 어드바이스 기능이 발생함
after-throwing	메서드가 예외를 던진 후에 어드바바이스 기능이 발생함
around	메서드가 호출되기 전과 후에 어드바이스 기능이 발생함

- **조인포인트**
  - 관점이 플로그인되는 애플리케이션의 실행 위치 즉, 관점의 코드가 애플리케이션의 정상적인 흐름속에 삽입되어 새로운 행위가 추가되는 지점
  - 호출되는 메서드
  - 예외가 던져지는 위치
  - 필드 값이 수정될때
- **조인트컷**
  - 관점이 어드바이스하는 조인포인트의 범위 축소
  - 관점의 어디서를 정의
  - 명확하게 클래스와 메서드 이름을 사용하거나 매치되는 클래스와 메서드 이름 패턴을 정의하는 정규식 이름으로 지정하여 어드바이스가 엮여하는 위치에 하나 이상의 조인포인트를 매치시킴

관점

- 구현하고자 하는 횡단 관심사의 기능으로 어드바이스와 포인트컷을 결합한 것
- 어드바이스와 포인트컷은 관점에 대하여 알기 위한 모든것 즉, 무엇을 언제 어디서 하는지를 정의

엮기

- 관점을 대상 객체에 적용시켜 새로운 프록시 객체를 생성하는 과정, 관점은 명시된 조인포인트에서 대상 객체에 엮임
- 엮기 방식

방식	설명	사용
컴파일 시	대상 클래스가 컴파일 시	Aspectj
클래스 로드시	대상 클래스가 JVM 안에로드될때	Aspectj 5의 LTW(load-time weaving)
실행시	애플리케이션 실행시 동적으로 프록시 개체를 생성시켜 대상 객체에 전달함	Spring AOP

- **도입**
  - 기존 클래스에 새로운 메서드나 애트리부트를 추가하는 것

2. Spring AOP 구현- Spring AOP 설정

- **Spring AOP 설정 모듈**
  - spring-aop.jar
  - spring-sapects.jar
- **pom.xml 파일에 종속성 추가**

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

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

관점 클래스 정의

- 새 패키지 작성(com.miya.order.aspect)
- 새 키지에 관점 클래스 작성

```
// 관점 클래스
public class LoggingAspect {
    private Logger log=LoggerFactory.getLogger(getClass());
    //.....
}
```

```
private String buildJoinPoint(JoinPoint joinpoint){
```

- 관첨 클래스에 buildJoinPoint 메소드 구현

- **joinPoint 인터페이스의 getTarget()메소드는 대상 조인포인트 객체 리턴**

```
String classname=joinpoint.getTarget().getClass.getName();
```

- **joinPoint 인터페이스의 getSignature()메소드는 조인포인트 메소드 시**

```
String methodname=joinpoint.getSignature().getClass.getName();
```

```
Object[] args=joinpoint.getArgs();
```

- **JoinPoint 인터페이스의 getArgs() 메소드조인 포인트 메서드의 매개 변수 목록을 리턴**

메소드를 조합하여 대상 조인포이트 정보를 구함

```
private String buildJoinPoint(JoinPoint joinpoint){
    String className=joinpoint.getTarget().getClass().getName();
    String methodName=joinpoint.getSignature().getName();
    String message=className + "클래스의 "+methodName + "(";
    Object[] args=joinpoint.getArgs();
    for(int i=0; i<args.length; ++i){
        Object arg=args[i];
        message+=arg.getClass().getName();
        if(i != args.length-1){
            message +=".";
        }
    }
    message +=")";
    return message;
}
```

관점 설정

- **Berfor 어드바이스 : 어드바이스하는 조인 포인트 메소드 호출되기 전에 호출되어 실행되는어드바이스**

```
public void logBefore(JoinPoint joinpoint) {
    String message = buildJoinpoint(joinpoint);
    message += "메서드 실행 시작";
    log.info(message);
}
```

- **A**
  어

```
public void logAfter(JoinPoint joinpoint) {
    String message = buildJoinPoint(joinpoint);
    message += "메서드 실행 공통 종료";
    log.info(message);
}
```

- **after-returning : 어드바이스하는 조인포인트 메서드 실행이 성공적으로 완료된 후 호출되어 실행되는 어드바이스**

```
public void logAfterReturning(JoinPoint joinpoint) {
    String message = buildJoinPoint(joinpoint);
    message += "메서드 실행 정상 종료";
    log.info(message);
}
```

- **a** **해하**

```
public void logAfterThrowing(JoinPoint joinpoint) {
    String message = buildJoinPoint(joinpoint);
    message += "메서드 실행 에러";
    log.info(message);
}
```

- **around 어드바이스: 조인포인트 메서드가 호출되기 전과 후에 어드바이스 기능을 수행할 수 있도록 하는 어드바이스around 어드바이스: ProcedingJoinPoint 타입 매개변수를 받음**

```
public void logAround(ProceedingJoinPoint joinpoint) throws Throwable{
    long start = System.currentTimeMillis();
    log.info("============  시작 ============");
    String message = buildJoinPoint(joinpoint);
    message += "메서드 실행 시작";
    log.info(message);
    joinpoint.proceed();// 조인포인트 메서드 호출
    message = buildJoinPoint(joinpoint);
    message += "메서드 실행 종료";
    long end = System.currentTimeMillis();
    long duration = end - start;
    log.info("실행 시간 : " + duration + " 밀리초");
    log.info("============  종료 ============");
}
```

포인트컷 설정

- 관점 클래스 Sping 빈으로설정

```
<bean id="loggingAspect" class="com.miya.order.aspet.LogingAspect"/>
```

- 관점을 설정하기 위해 <beans> 태그 안에 aop 네이스페이스 추가

설정	설명
<aop:config>	최상위 AOP 요소
<aop:aspect>	관점 설정
<aop:pointcut>	포인트컷 설정
<aop:before>	before 어드바이스 설정
<aop:after>	after 어드바이스 설정
<aop:after-returning>	after-returning 어드바이스 설정
<aop:after-throwing>	after-throwing 어드바이스 설정
<aop:around>	around 어드바이스 설정

- **AOP 설정 정의**

```
<aop:config>
</aop:config>
```

- **A**

```
<aop:config>
    <aop:aspect ref="loggingAspect">
    </aop:aspect>
</aop:config>
```

관점에 포인트컷 설정

```
<aop:config>
    <aop:aspect ref="loggingAspect">
        <aop:pointcut expression="execution(* com.miya.order.*.*.get*(..))
            || execution(* com.miya.order.*.*.find*(..))" id="getLogging"/>
        <aop:pointcut expression="execution(* com.miya.order.*.*.save*(..))"
            id="saveLogging"/>
    </aop:aspect>
</aop:config>
```

- 관점에 포함된 어드바이스 설정

- AOP 설정을 위한 <aop:config>태그 추가

```
<aop:config>
    <aop:aspect ref="loggogAspect">
        <aop:pointcut expression="execution(* com.miya.order.*.*.get*(..))
            || execution(* com.miya.order.*.*.find*(..))" id="getLogging />
        <aop:pointcut expression="execution(* com.miya.order.*.*.save*(..))
            id="saveLogging />
        <aop:before method="logBefore" pointcut-ref="getLogging"/>
        <aop:after method="logAfter" pointcut-ref="getLogging"/>
        <aop:after-returning method="logAfterReturning" pointcut-ref="getLogging/>
        <aop:after-throwing method="logAfterThrowing" pointcut-ref="getLogging/>
        <aop:around method="logAround" pointcut-ref="getLogging/>
    </aop:aspect>
</aop:config>
```

관점 적용

- **표현식**


메서드가 속한 클래스
반환타입
메서드
인수
execution(* com.miya.order.*.*.get*(..))
지명자
메서드 명세

요소 지명자

지명자	설명
execution()	메서드 실행인 조인포인트를 매치함
within()	일정한 타입 별로 조인포인트 매치를 제한함
bean()	빈 이름으로 조인포인트 매치를 제한함
target()	대상 객체가 해당 타입인 것으로 조인포인트 매치를 제한함
this()	AOP 프록시 빈 인스턴스가 해당 타입인 것으로 조인포인트 매치를 제한함
args()	인수가 해당 타입의 인스턴스인 메서드 실행으로 조인포인트 매치를 제한함

- 지명자는 and, or, not (&&, ||, !) 연산자를 사용하여 결합 가능

요소 지명자 사용

- 어드바이스 실행될 때 포인트컷 메서드 선택

```
execution(* com.ensoa.order.*.*.get*(..))
```

- 지명자 묶기 : get으로 시작하는 메소드와 find로 시작하는 메서드 포인트컷을 선택

```
"execution(* com.ensoa.order.*.*.get*(..))
            || execution(* com.ensoa.order.*.*.find*(..))"
```

- 포인트컷 메서드가 실행되는 클래스를 설정

```
within(* com.ensoa.order.service.*)
```

- com.ensoa.order.service 패키지 안에 있는 save로 시작함수로 제한

```
execution(* com.ensoa.order.*.*.save*(..))
 and within(com.ensoa.order.service.*)
```

요소 지정자 사용 예-2

- 포인트컷 메서드가 실행되는 Spring 빈 이름을 설정

```
bean(customer*)
```

- 어드바이스가 실행될 대상 객체의 타입을 제한

```
target(com.ensoa.order.service.CustomerService)
```

- 어드바이스가 실행 될 AOP 프록시 타입을 제한

```
this(com.ensoa.order.service.CustomerService)
```

- 어드바이스가 실행될 메서드 매개 변수 타입을 제한

```
args(long)
```

관점에 데이터 수정 포인트컷 추가

```
<aop:pointcut expression="execution(*
        com.miya.order.service.*.update*(com.miya.order.domain.Customer))
    and args(customer)" id="updateLogging"/>
```

```
// getLogging 포인트컷
public void getLogging() {}

// 매개변수 포인트 컷
public void updateLogging(Customer customer) {}


.....
public void logBeforeUpdate(JoinPoint joinpoint, Customer customer) {
    String message = buildJoinPoint(joinpoint);
    message += "메서드 실행 시작";
    log.info(message);
        log.info("변경 정보 : " + customer.getName() + ", "
        + customer.getAddress() + ", " + customer.getEmail());
}
```

3. @AspectJ 어노테이션

@AspectJ 어노테이션 AOP 설정

설정	설명
<aop:config>	최상위 AOP 요소
<aop:aspect>	관점 설정
<aop:pointcut>	포인트컷 설정
<aop:before>	before 어드바이스 설정
<aop:after>	after 어드바이스 설정
<aop:after-returning>	after-returning 어드바이스 설정
<aop:after-throwing>	after-throwing 어드바이스 설정
<aop:around>	around 어드바이스 설정

관점 클래스에 @Aspect 어노테이션 지정

- XML 파일에 <aop:sapect> 에 해당

```
@Aspect
@Component
public class LoggingAspect {
}
```

```
// getLogging 포인트컷
        @Pointcut("execution(* com.ensoa.order.*.*.get*(..))"
                        + " || execution(* com.ensoa.order.*.*.find*(..))")
    public void getLogging() {}
```

```
public void updateLogging(Customer customer) {}
// before 어드바이스
@Before("getLogging()")
public void logBefore(JoinPoint joinpoint) {
    String message = buildJoinpoint(joinpoint);
    message += "메서드 실행 시작";
    log.info(message);
}
// after 어드바이스
@After("getLogging()")
public void logAfter(JoinPoint joinpoint) {
    String message = buildJoinpoint(joinpoint);
    message += "메서드 실행 공통 종료";
    log.info(message);
}
```

```
//after-returning 어드바이스
@AfterReturning("getLogging()")
public void logAfterReturning(JoinPoint joinpoint) {
   String message = buildJoinpoint(joinpoint);
   message += "메서드 실행 정상 종료";
   log.info(message);
}
// after-throwing 어드바이스
@AfterThrowing("getLogging()")
public void logAfterThrowing(JoinPoint joinpoint) {
   String message = buildJoinpoint(joinpoint);
   message += "메서드 실행 에러";
   log.info(message);
}
```

```
// around 어드바이스
@Around("execution(* com.ensoa.order.*.*.save*(..))")
public void logAround(ProceedingJoinPoint joinpoint) throws Throwable {
      long start = System.currentTimeMillis();
      log.info("===========  시작 ===========");
      String message = buildJoinpoint(joinpoint);
      message += "메서드 실행 시작";
      log.info(message);
      joinpoint.proceed();                    // 메서드 호출
      message = buildJoinpoint(joinpoint);
      message += "메서드 실행 종료";
      long end = System.currentTimeMillis();
      long duration = end - start;
      log.info("실행 시간 : " + duration + " 밀리초");
      log.info("===========  종료 ===========");
}
```

- **XML 파일에 @Aspect 알림**

```
<aop:aspectj-autoproxy>
```

어드바이스에 매개변수 전달

```
public void logBeforeUpdate(JoinPoint joinpoint, Customer customer) {
        String message = buildJoinpoint(joinpoint);
        message += "메서드 실행 시작";
        log.info(message);
        log.info("변경 정보 : " + customer.getName() + ", "
            + customer.getAddress() + ", " + customer.getEmail());
    }
```

```
// 매개변수 포인트 컷
@Pointcut("execution(*
    com.ensoa.order.service.*.update*(com.ensoa.order.domain.Customer))"
                        + " && args(customer)")
public void updateLogging(Customer customer) {}
```

```
@Before("updateLogging(customer)")
        public void logBeforeUpdate(JoinPoint joinpoint, Customer customer) {
                String message = buildJoinpoint(joinpoint);
                message += "메서드 실행 시작";
                log.info(message);
                log.info("변경 정보 : " + customer.getName() + ", "
                    + customer.getAddress() + ", " + customer.getEmail());
        }
```