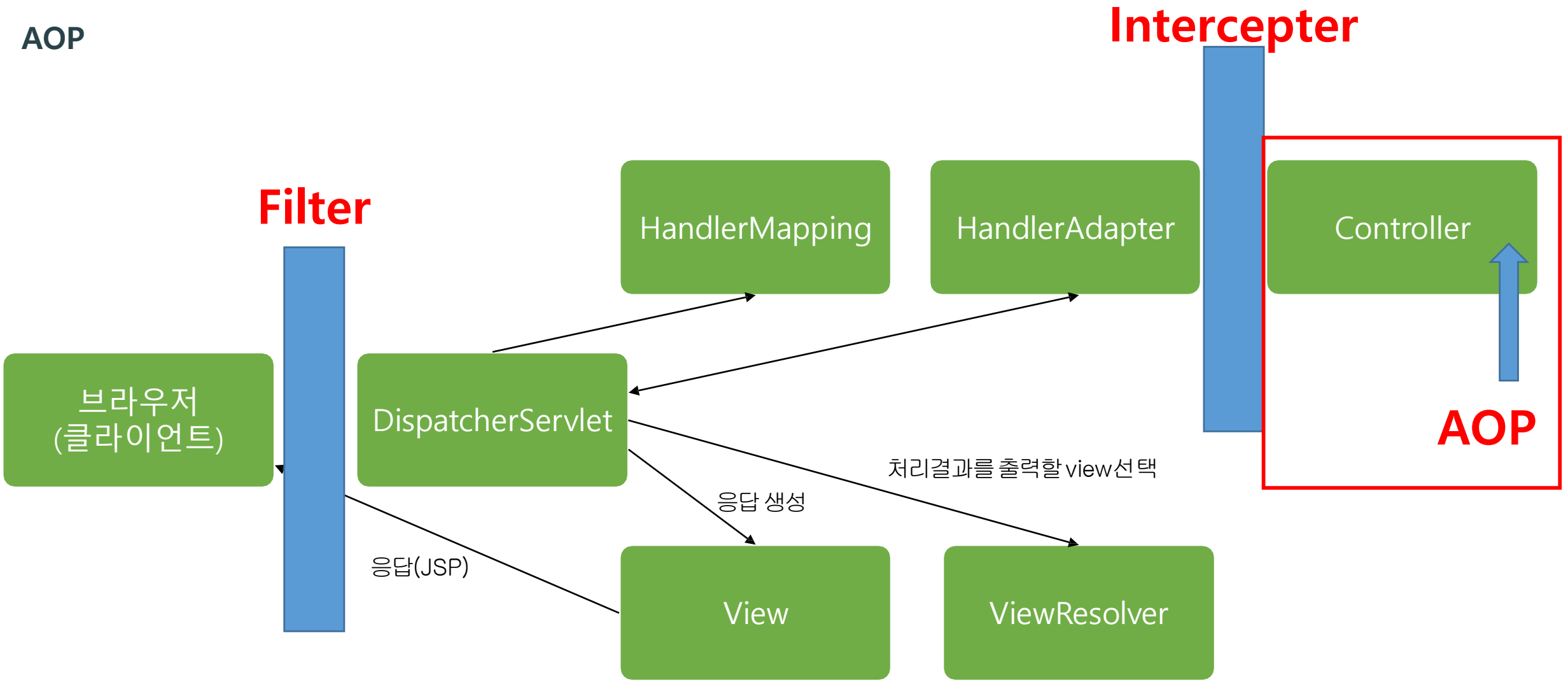
Spring Framework

-AOP
1. AOP란
2. AOP의 용어
3. AOP설정

AOP

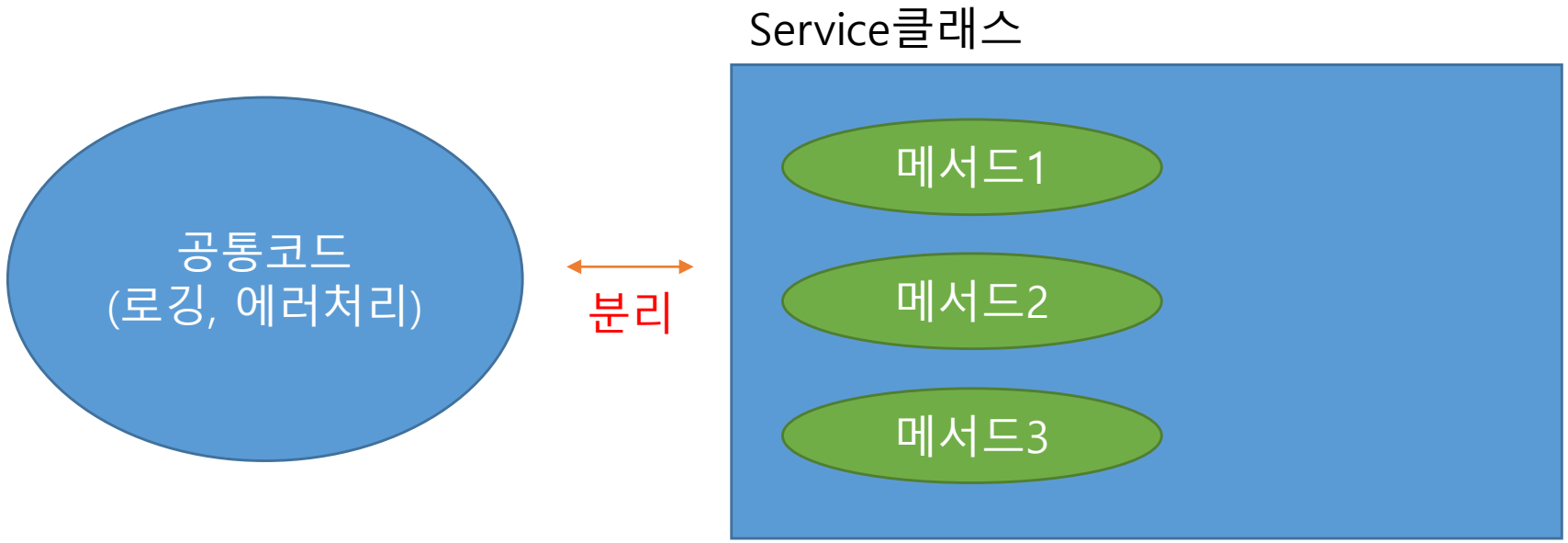


Filter- 디스패쳐 서블릿을 거치기 전에 처리
Interceptor – 컨트롤러 전 후로 요청을 처리
AOP – 비즈니스 로직에서(메서드) 에서 세밀하게 처리

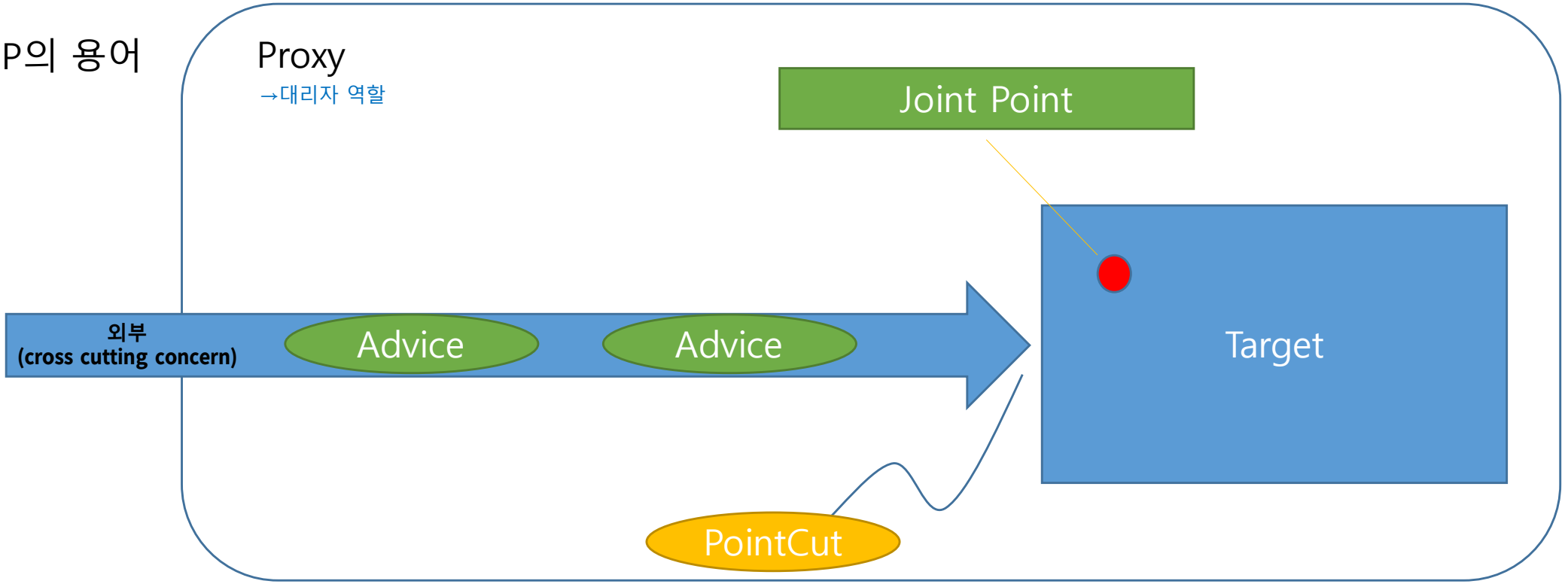
1.AOP란

AOP란 (Aspect-Oriented-Programming)
-(관점지향프로그래밍)더욱 객체지향답게 의미를 갖는다
-공통 코드, 개별코드(비즈니스 로직) 를 분리해서 작성한다.
-ex) java의 공통 기능을 부모클래스로 정의하고 상속 관계로 사용

-기존 코드를 수정하지 않고, 외부에서 원하는 기능에 접근하여 결합



2.AOP의 용어



Target (=core concern)
-개발자가 작성한 Service클래스라 보면 된다
Joint Point
-Target의 메서드 (Service클래스 안에 메서드) →Target에 실제 적용할 메서드
Advice (=cross cutting concern)
-공통 코드(로깅, 에러처리 등) →부가기능에 집중된 구현체
PointCut
-Advice를 어떤 Joint Point에 결합할 것인지 설정(공통 코드를 어떤 메서드에 적용할 것인가)

3.AOP설정
- 스프링설정파일의 위치는 디스패처 서블릿 생성 이후의 파일과 같은 라인에 생성

-AOP사용 설정

pom.xml 라이브러리 다운

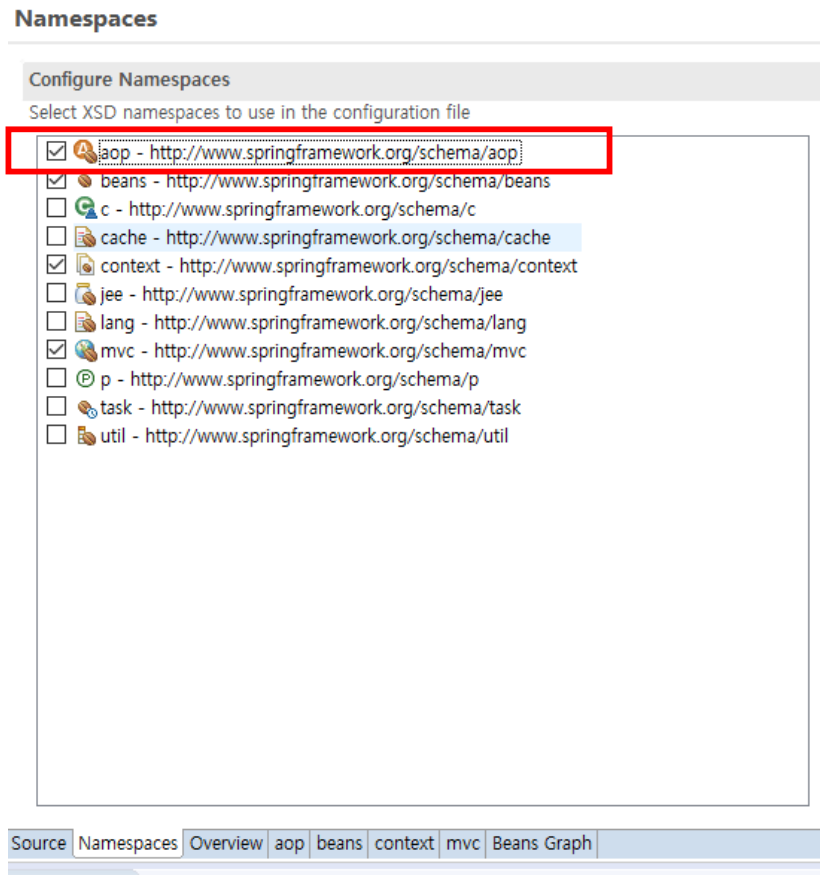
```
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>${org.aspectj-version}</version>
</dependency>
```

servlet-xml 태그 추가

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

-위 태그 선언으로 aop 어노테이션을 사용할 수 있습니다

Namespace설정



2.AOP의 용어

Advice
-공통 코드(로깅, 에러처리 등)

PointCut
-Advice를 어떤 Joint Point에 결합할 것인지 설정
(공통 코드를 어떤 메서드에 적용할 것인가)

-execution(표현식) : 해당 메서드를 기준으로 PointCut 실행

Advice 동작위치

구분	설명
@Before	메서드를 호출 전 실행
@After	메서드를 호출 후 실행
@AfterThrowing	메서드 예외 발생시 동작
@Around	메서드에 결합해서 처리

```
@Aspect
@Component
public class LogAdvice {

@Before("execution(*com.zerock.service.memberServiceImpl*.*(..))")
public void beforeLog() {

}

@After("execution(*com.zerock.service.memberServiceImpl*.*(..))")
public void afterLog() {

}
```

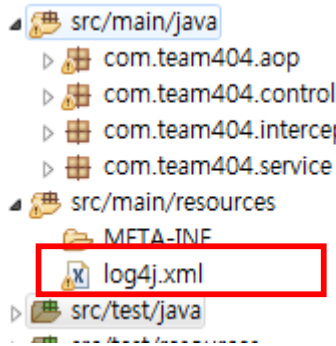
2.AOP의 용어

1. @Around – 메서드와 결합해서 실행 구조를 바꾸는 강력한 기능

2. Before, after와 다르게 반드시 반환 유형이 존재 해야함
3. ProceedingJointPoint 타입 은 타겟 메서드의 직접 관여할 수 있음
4. proceed() 메서드의 실행이 없다면, 타겟 메서드는 실행되지 않음

```
@Aspect
@Component
public class LogAdvice {

1 @Around("execution(* com.zerock.service.memberServiceImpl.*(..))")
public 2 Object aroundLog(ProceedingJoinPoint jp) 3 {

        Object result = null;

        try {
                4
                result = jp.proceed();
        } catch (Throwable e) {
                e.printStackTrace();
        }

        return result;
}
```

2.AOP설정 후 log설정



적용 패키지단위를 추가

```
<!-- aop패키지를 로그레벨로 추가 -->
<logger name="com.team404.aop">
    <level value="info" />
</logger>
```

```
@Aspect
@Component
public class LogAdvice {

    private static final Logger log = LoggerFactory.getLogger(LogAdvice.class);

    @Around("execution(* com.team404.service.memberServiceImpl.*(..))")
    public Object aroundLog(ProceedingJoinPoint jp) {
        long start = System.currentTimeMillis();

        log.info("적용클래스:" + jp.getTarget());
        log.info("적용파리미터:" + Arrays.toString(jp.getArgs()) );

        Object result = null;

        try {

            result = jp.proceed();
        } catch (Throwable e) {
            e.printStackTrace();
        }

        long end = System.currentTimeMillis();

        log.info("메서드수행에걸린시간:" + (end - start));


        return result;
    }

}
```