

# chapter07

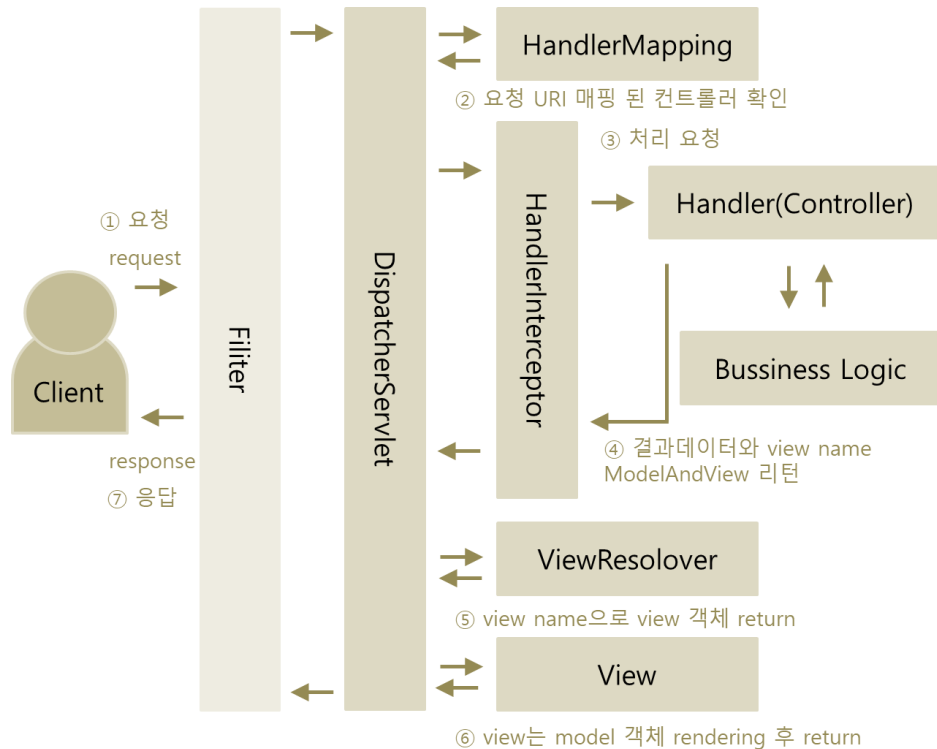
## Spring Boot

1. **Interceptor** 처리
2. AOP 설정

# 01 Interceptor 처리

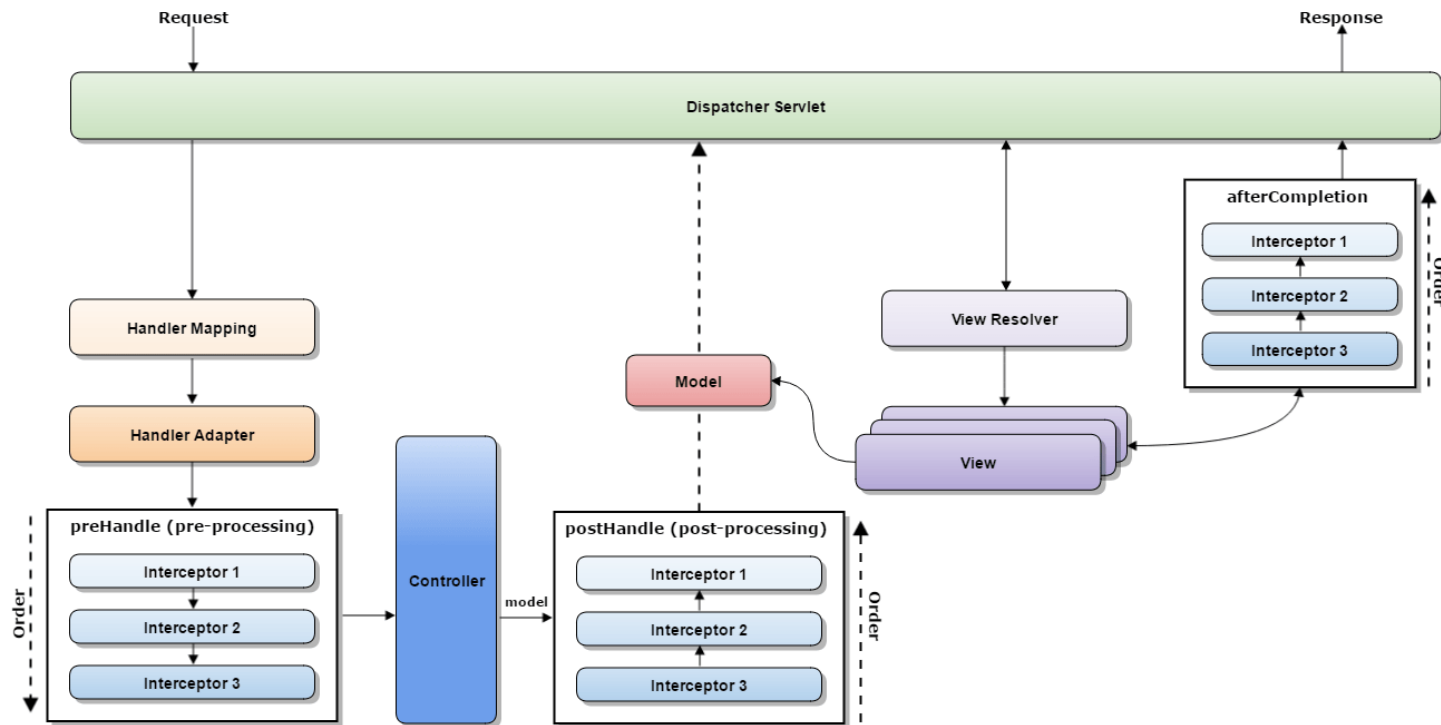
## ■ 인터셉터(Interceptor)란?

- Interceptor는 Spring Context(ApplicationContext) 기능으로 임의의 URI를 호출시 DispatcherServlet에서 해당 Controller가 처리되기 전과 후에 발생하는 이벤트이다.



## ■ HandlerInterceptorAdapter를 상속받아 인터셉터 구현 할 수 있다

- 스프링에서 인터셉터는 HandlerInterceptorAdapter를 상속받아 구현할 수 있는데 해당 클래스는
- preHandle**, **postHandle**, **afterCompletion**, **afterConcurrentHandlingStarted** 네 개의 메서드를 포함



# 01 Interceptor 처리

## ■ HandlerInterceptor를 구현 하는 Interceptor 클래스 작성

- com.board 패키지에 interceptor 패키지를 추가하고, LoggerInterceptor 클래스를 추가하고 HandlerInterceptor를 구현

```
package com.board.interceptor;

import javax.servlet.http.*;
import org.slf4j.*;
import org.springframework.web.servlet.*;

public class LoggerInterceptor implements HandlerInterceptor {

    private final Logger logger = LoggerFactory.getLogger(this.getClass());

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {
        logger.debug("=====");
        logger.debug("===== BEGIN =====");
        logger.debug("Request URI ==> " + request.getRequestURI());
        return true;
    }

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView mav) throws Exception {
        logger.debug("===== END =====");
        logger.debug("=====");
    }

}
```

# 01 Interceptor 처리

## ■ LoggerInterceptor 클래스를 빈(Beans)으로 등록하기

- com.board.configuration 패키지에 MvcConfiguration 클래스를 생성하고, WebMvcConfigurer 인터페이스를 구현

```
package com.board.configuration;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

import com.board.interceptor.LoggerInterceptor;

@Configuration
public class MvcConfiguration implements WebMvcConfigurer {

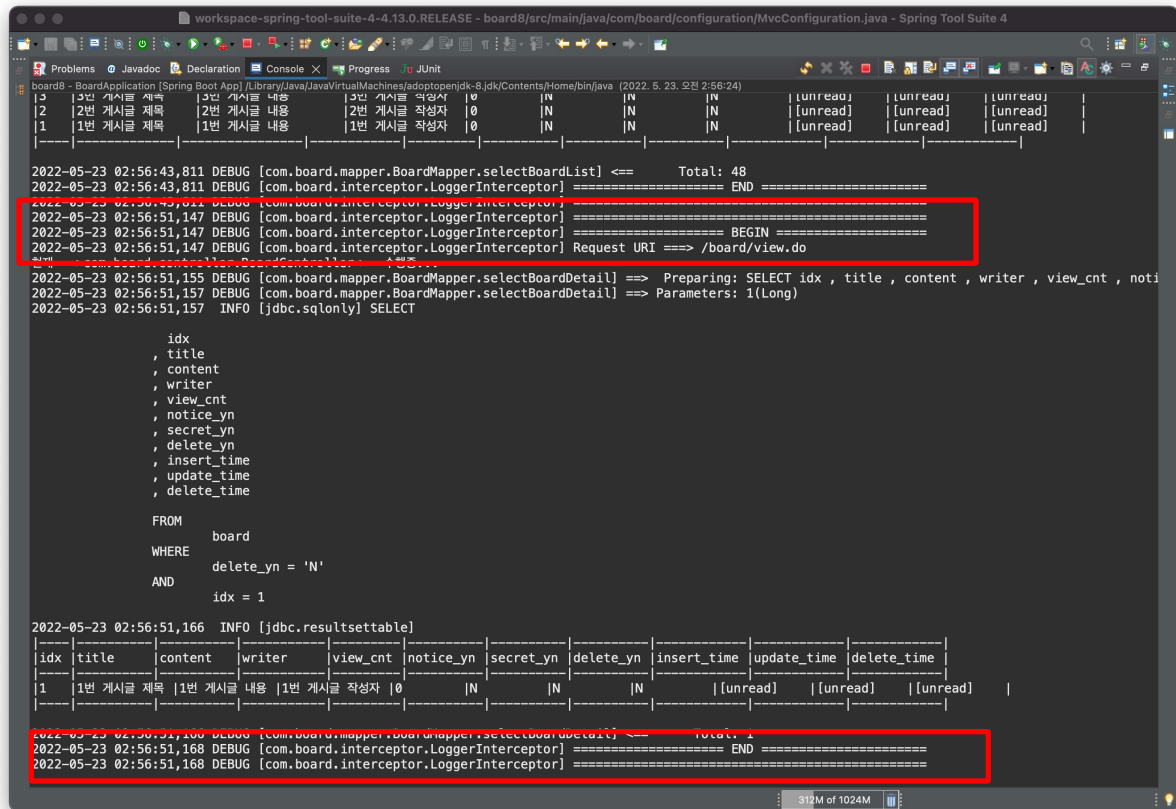
    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(new LoggerInterceptor()).excludePathPatterns("/css/**", "/fonts/**",
        "/plugin/**", "/scripts/**");
    }
}
```

- src/main/resources 디렉터리의 static 폴더에 포함된 정적 리소스 파일을 제외 시켜야 함

# 01 Interceptor 처리

## ■ Run App

- 애플리케이션 실행하고 게시글 조회시 STS 의 console 창에 로그가 기록 된다



```
workspace-spring-tool-suite-4-4.13.0.RELEASE - board8/src/main/java/com/board/configuration/MvcConfiguration.java - Spring Tool Suite 4

board8 - BoardApplication [Spring Boot App] /Library/Java/JavaVirtualMachines/adoptopenjdk-8.jdk/Contents/Home/bin/java (2022. 5. 23. 오전 2:56:24)

1 | 1번 게시글 제목 | 1번 게시글 내용 | 1번 게시글 작성자 | 0 | IN | IN | IN | [unread] | [unread] | [unread] |
2 | 2번 게시글 제목 | 2번 게시글 내용 | 2번 게시글 작성자 | 0 | IN | IN | IN | [unread] | [unread] | [unread] |
1 | 1번 게시글 제목 | 1번 게시글 내용 | 1번 게시글 작성자 | 0 | IN | IN | IN | [unread] | [unread] | [unread] |

-----
2022-05-23 02:56:43,811 DEBUG [com.board.mapper.BoardMapper.selectBoardList] <== Total: 48
2022-05-23 02:56:43,811 DEBUG [com.board.interceptor.LoggerInterceptor] ===== END =====
2022-05-23 02:56:51,147 DEBUG [com.board.interceptor.LoggerInterceptor] ===== BEGIN =====
2022-05-23 02:56:51,147 DEBUG [com.board.interceptor.LoggerInterceptor] Request URI ==> /board/view.do
2022-05-23 02:56:51,155 DEBUG [com.board.mapper.BoardMapper.selectBoardDetail] ==> Preparing: SELECT idx , title , content , writer , view_cnt , notice_yn , secret_yn , delete_yn , insert_time , update_time , delete_time
2022-05-23 02:56:51,157 DEBUG [com.board.mapper.BoardMapper.selectBoardDetail] ==> Parameters: 1(Long)
2022-05-23 02:56:51,157 INFO [jdbc.sqlonly] SELECT

idx
, title
, content
, writer
, view_cnt
, notice_yn
, secret_yn
, delete_yn
, insert_time
, update_time
, delete_time

FROM
board
WHERE
delete_yn = 'N'
AND
idx = 1

2022-05-23 02:56:51,166 INFO [jdbc.resultsettable]
idx | title | content | writer | view_cnt | notice_yn | secret_yn | delete_yn | insert_time | update_time | delete_time |
1 | 1번 게시글 제목 | 1번 게시글 내용 | 1번 게시글 작성자 | 0 | IN | IN | IN | [unread] | [unread] | [unread] |

-----
2022-05-23 02:56:51,168 DEBUG [com.board.mapper.BoardMapper.selectBoardDetail] <== Total: 1
2022-05-23 02:56:51,168 DEBUG [com.board.interceptor.LoggerInterceptor] ===== END =====
2022-05-23 02:56:51,168 DEBUG [com.board.interceptor.LoggerInterceptor] =====
```

# chapter07

## Spring Boot

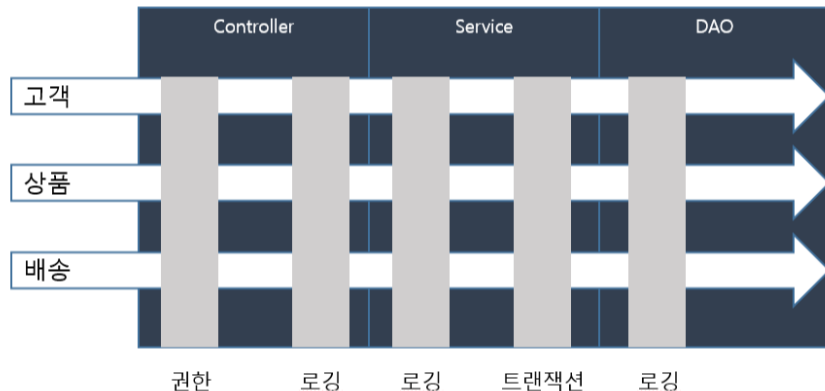
1. Interceptor 처리
2. **AOP 설정**

### ■ AOP(Aspect Oriented Programming)란?

- AOP는 관점 지향 프로그래밍

공통으로 처리되어야 하는 로깅, 보안, 예외 처리 등의 코드를 핵심로직에서 분리하여 하나의 단위로 묶는 모듈화의 개념

AOP에서 관점은 핵심적인 관점과 부가적인 관점으로 나눌 수 있습니다. 핵심적인 관점은 핵심 비즈니스 로직을 의미하고 부가적인 관점은 공통으로 처리되어야 하는 코드를 의미합니다.





### ■ (Aspect Oriented Programming)란?

- AOP 용어

용어	설명
관점(Aspect)	공통적으로 적용될 기능을 의미 부가적인 기능을 정의한 코드인 어드바이스와 어드바이스를 어느 곳에 적용할지 결정하는 포인트컷의 조합으로 만들어짐
어드바이스(Advice)	실제로 부가적인 기능을 구현한 객체를 의미
조인포인트(Join point)	어드바이스를 적용할 위치를 의미 예를 들어, BoardService에서 CRUD를 처리하는 메서드 중 원하는 메서드를 골라서 어드바 이스를 적용할 수 있고, 이때 BoardService의 모든 메서드는 조인 포인트가 됨
포인트컷(Pointcut)	어드바이스를 적용할 조인 포인트를 선별하는 과정이나, 그 기능을 정의한 모듈을 의미 정규 표현식이나 AspectJ 문법을 사용해서 어떤 조인 포인트를 사용할지 결정
타겟(Target)	실제로 비즈니스 로직을 수행하는 객체를 의미 즉, 어드바이스를 적용할 대상
프록시(Proxy)	어드바이스가 적용되었을 때 생성되는 객체를 의미
인트로덕션(Introduction)	타겟에는 없는 새로운 메서드나 인스턴스 변수를 추가하는 기능
위빙(Weaving)	포인트컷에 의해서 결정된 타겟의 조인 포인트에 어드바이스를 적용하는 것을 의미

## 02 AOP 설정

### ■ LoggerAspect 클래스 생성

- com.board.aop 패키지를 추가하고, aop 패키지에 LoggerAspect 클래스를 추가

```
package com.board.aop;

import org.aspectj.lang.ProceedingJoinPoint;

@Component
@Aspect
public class LoggerAspect {

    private final Logger logger = LoggerFactory.getLogger(this.getClass());

    @Around("execution(* com.board..controller.*Controller.*(..)) or "
            + "execution(* com.board..service.*Impl.*(..)) or "
            + "execution(* com.board..mapper.*Mapper.*(..))")
    public Object printLog(ProceedingJoinPoint joinPoint) throws Throwable {

        String type = "";
        String name = joinPoint.getSignature().getDeclaringTypeName();

        if (name.contains("Controller") == true) {
            type = "Controller ==> ";
        }
        else if (name.contains("Service") == true) {
            type = "ServiceImpl ==> ";
        }
        else if (name.contains("Mapper") == true) {
            type = "Mapper ==> ";
        }

        logger.debug(type + name + "." + joinPoint.getSignature().getName() + "()");
        return joinPoint.proceed();
    }
}
```

- (String) : 정확하게 String 타입 파라미터
- () : 파라미터가 없어야 한다
- (\*) : 정확히 하나의 파라미터, 단 모든 타입을 허용
- (\*, \*) : 정확히 두 개의 파라미터, 단 모든 타입을 허용
- (..) : 갯수와 무관, 모든 타입 허용  
(파라미터가 없어도 된다. 0..\*로 이해하면 된다)
- (String, ..) : String 타입으로 시작해야 한다.  
갯수와 무관, 모든 파라미터, 모든 타입을 허용한다.
  - (String), (String, Xxx), (String, Xxx, Xxx) 허용

### ■ LoggerAspect 클래스 생성

애너테이션	설명
@Component	스프링 컨테이너에 빈(Been)으로 등록하기 위한 애너테이션입니다. @Bean은 개발자가 제어할 수 없는 외부 라이브러리를 빈(Been)으로 등록할 때 사용하고, @Component는 개발자가 직접 정의한 클래스를 빈(Been)으로 등록할 때 사용합니다.
@Aspect	AOP 기능을 하는 클래스의 클래스 레벨에 지정하는 애너테이션입니다.
@Around	어드바이스(Advice)의 종류 중 한 가지로 어드바이스는 모두 다섯 가지의 타입이 있습니다. 다섯 가지 중 어라운드(Around)는 메서드의 호출 자체를 제어할 수 있기 때문에 어드바이스 중 가장 강력한 기능이라고 볼 수 있습니다.

### ■ @Around

- 어드바이스(Advice)의 종류 중 한 가지로 어드바이스는 5 가지의 타입이 존재하며, 그 중 어라운드(Around)는 메서드의 호출 자체를 제어할 수 있기 때문에 어드바이스 중 가장 강력한 기능

타입	애너테이션	기능
Before Advice	@Before	Target 메서드 호출 이전에 적용할 어드바이스 정의
After returning	@AfterReturning	Target 메서드가 성공적으로 실행되고, 결과값을 반환한 뒤에 적용
After throwing	@AfterThrowing	Target 메서드에서 예외 발생 이후에 적용 (try/catch 문의 catch와 유사)
After	@After	Target 메서드에서 예외 발생에 관계없이 적용 (try/catch 문의 finally와 유사)
Around	@Around	Target 메서드 호출 이전과 이후 모두 적용 (가장 광범위하게 사용됨)

## 02 AOP 설정

### ■ execution

- @Around 안에서 execution으로 시작하는 구문은 포인트컷을 지정하는 문법으로 **execution**, within, bean이 있음  
execution으로 접근 제어자, 리턴 타입, 타입 패턴, 메서드, 파라미터 타입, 예외 타입 등을 조합해서 정교한 포인트컷을 만들 수 있음

예시	설명
execution(BoardDTO select*(..))	리턴 타입이 BoardDTO 클래스이고, 메서드의 이름이 select로 시작하며, 파라미터가 0개 이상인 모든 메서드가 호출될 때 (0개 이상은 패키지, 메서드, 파라미터 등 모든 것을 의미)
execution(* com.board.controller.*())	해당 패키지 밑에 파라미터가 없는 모든 메서드가 호출될 때
execution(* com.board.controller.*(..))	해당 패키지 밑에 파라미터가 0개 이상인 모든 메서드가 호출될 때
execution(* com.board..select(*))	com.board 패키지의 모든 하위 패키지에 존재하는 select로 시작하고, 파라미터가 한 개인 모든 메서드가 호출될 때
execution(* com.board..select(*, *))	com.board 패키지의 모든 하위 패키지에 존재하는 select로 시작하고, 파라미터가 두 개인 모든 메서드가 호출될 때

### ■ execution

- LoggerAspect 클래스의 포인트컷 표현식은 and( && ) 또는 or( || )를 조합해서 사용
- execution 포인트컷은 AOP에서 중요

예시	설명
* com.board..controller.*Controller.*(..)	com.board 패키지의 모든 하위 패키지 중 controller로 시작하는 패키지에서 xxxController와 같은 패턴의 이름을 가진 클래스에서 파라미터가 0개 이상인 메서드를 의미
* com.board..service.*Impl.*(..)	com.board 패키지의 모든 하위 패키지 중 service로 시작하는 패키지에서 xxxServiceImpl과 같은 패턴의 이름을 가진 클래스에서 파라미터가 0개 이상인 메서드를 의미
* com.board..mapper.*Mapper.*(..)	com.board 패키지에서 모든 하위 패키지 중 mapper로 시작하는 패키지에서 xxxMapper와 같은 패턴의 이름을 가진 인터페이스에서 파라미터가 0개 이상인 메서드를 의미

### ■ ProceedingJoinPoint와 나머지

- **전체 로직의 역할**은 메서드에 대한 정보를 가지고 있는 Signature 객체에 담겨 있는 파일명을 포함한 전체 패키지 경로를 name에 담아 **어떤 클래스의 어떤 메서드를 호출하는지를 로그에 출력**
- printLog 메서드의 파라미터인 ProceedingJoinPoint 인터페이스가 상속받는 클래스는 다음의 메서드를 포함합니다.

메서드	설명
Object[] getArgs()	전달되는 모든 파라미터들을 Object 타입의 배열로 리턴
String getKind()	해당 어드바이스(Advice)의 타입을 리턴
Signature getSignature()	실행되는 대상 객체의 메서드에 대한 정보를 리턴
Object getTarget()	타겟(Target) 객체를 리턴
Object getThis()	어드바이스(Advice)를 행하는 객체를 리턴

## 02 AOP 설정

### ■ Run App

- 애플리케이션 실행하고 게시글 조회시 STS 의 console 창에 로그가 기록 된다

```

7:50,578 DEBUG [com.board.interceptor.LoggerInterceptor] ===== BEGIN =====
7:50,578 DEBUG [com.board.interceptor.LoggerInterceptor] Request URI ==> /favicon.ico
7:50,579 DEBUG [com.board.interceptor.LoggerInterceptor] ===== END =====
7:50,579 DEBUG [com.board.interceptor.LoggerInterceptor] ===== BEGIN =====
7:50,586 DEBUG [com.board.interceptor.LoggerInterceptor] Request URI ==> /error
7:50,631 DEBUG [com.board.interceptor.LoggerInterceptor] ===== END =====
7:51,981 DEBUG [com.board.interceptor.LoggerInterceptor] ===== BEGIN =====
7:51,981 DEBUG [com.board.interceptor.LoggerInterceptor] Request URI ==> /board/view.do
7:51,986 DEBUG [com.board.aop.LoggerAspect] Controller ==> com.board.controller.BoardController.openBoardDetail()
d.controller.BoardController<-- 수행중...
7:51,989 DEBUG [com.board.aop.LoggerAspect] ServiceImpl ==> com.board.service.BoardServiceImpl.getBoardDetail()
7:51,990 DEBUG [com.board.aop.LoggerAspect] Mapper ==> com.board.mapper.BoardMapper.selectBoardDetail()
7:51,991 DEBUG [com.board.mapper.BoardMapper.selectBoardDetail] ==> Preparing: SELECT idx , title , content , writer , view_cnt , notice_yn , secret_yn , delete_yn , insert_time , update_time , delete_time
7:51,992 DEBUG [com.board.mapper.BoardMapper.selectBoardDetail] ==> Parameters: 49(Long)
7:51,992 INFO [jdbc.sqlonly] SELECT

    idx
    , title
    , content
    , writer
    , view_cnt
    , notice_yn
    , secret_yn
    , delete_yn
    , insert_time
    , update_time
    , delete_time

FROM
    board

WHERE
    delete_yn = 'N'

AND
    idx = 49
  
```