

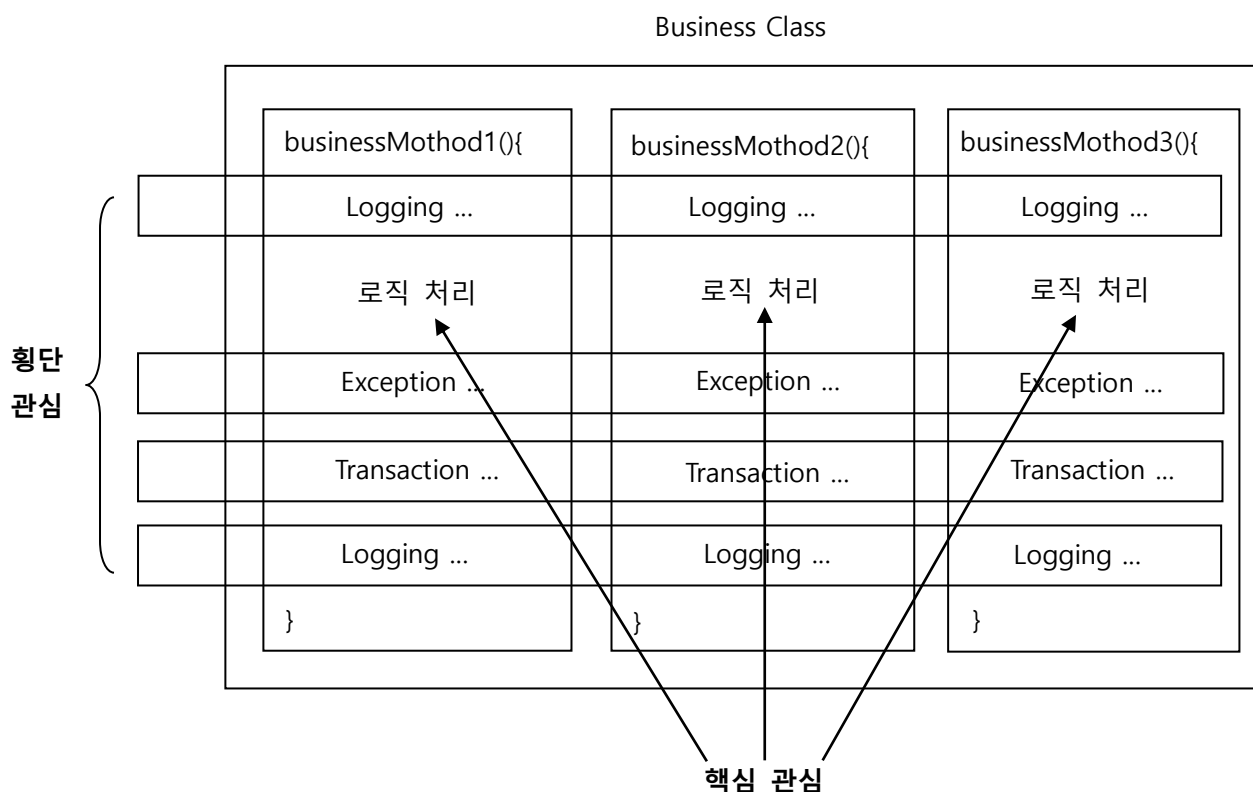
비즈니스 컴포넌트 개발에서 가장 중요한 두 가지 원칙은 낮은 결합도와 높은 응집도를 유지하는 것이다. 스프링의 의존성 주입(Dependency Injection)을 이용하면 비즈니스 컴포넌트를 구성하는 객체들의 결합도를 떨어뜨릴 수 있어서 의존관계를 쉽게 변경할 수 있다. 스프링의 IoC(Inversion Of Control: 제어의 역전)가 결합도와 관련된 기능이라면, AOP(Aspect Oriented Programming)는 응집도와 관련된 기능이라 할 수 있다.

1.1 AOP 이해

AOP는 흔히 '관점 지향 프로그래밍'이라는 용어로 번역되는데 '관점'이라는 용어는 개발자들에게 '관심사(concern)'라는 말로 통용된다. '관심사'는 개발 시 필요한 고민이나 염두에 두어야 하는 일이라고 생각할 수 있는데 코드상에서는 파라미터가 올바르게 들어왔는지, 이작업을 하는 사용자가 적절한 권한을 가진 사용자인지, 모든 예외를 어떻게 처리해야 하는지 등의 일이다.

위 고민들은 '핵심 로직'이 아니지만 코드를 온전하게 만들기 위해서 필요한 고민들인데 전통적인 방식에서는 개발자가 반복적으로 이러한 코드들을 명시해 주어야 한다. 하지만 스프링에서는 AOP가 추구하는 것은 '관심사의 분리(separate concerns)'이다. AOP는 개발자가 염두에 두어야 하는 일들은 별도의 '관심사'로 분리하고, 핵심 비즈니스 로직만을 작성할 것을 권장한다.

그래서 AOP를 이용하면 작성된 모든 메서드가 실행 시간이 얼마인지를 기록하는 기능을 기존 코드의 수정 없이도 작성할 수 있고, 잘못된 파라미터가 들어와서 예외가 발생하는 상황을 기존 코드의 수정 없이도 제어할 수 있다.



이 두 관심을 완벽하게 분리할 수 있다면, 구현하는 메소드에는 실제 비즈니스 로직만으로 구성할 수 있으므로 더욱 간결하고 응집도 높은 코드를 유지할 수 있다. 문제는 기존의 OOP 언어에서는 횡단 관심에 해당하는 공통 코드를 완벽하게 독립적인 모듈로 분리하기가 어렵다.

기존 OOP에서 관심 분리가 어려운지 확인하고 스프링의 AOP가 관심 분리를 해결하는 과정을 살펴본다. BoardServiceImpl 구현 클래스의 모든 비즈니스 메소드가 실행되기 직전에 공통으로 처리할 로직을 LoggerInfo 클래스에 printLogging() 메소드로 구현한다.

LoggerInfo.java

```
package com.spring.common.log;

public class LoggerInfo{
    public void printLogging(){
        logger.info("-----");
        logger.info("[공통 로그 Log] 비즈니스 로직 수행 전 동작");
        logger.info("-----");
    }
}
```

BoardServiceImpl 클래스의 모든 메소드는 LoggerAdvice를 이용하도록 수정한다.
그리고 다시 BoardServiceImpl 클래스의 모든 메소드는 LoggerAdvice를 이용하도록 수정한다.

BoardServiceImpl.java (설명을 위해 작성한 코드로 코딩 생략.)

```
package com.spring.client.board.service;

import java.util.List;
...

@Service
public class BoardServiceImpl implements BoardService {
    ... 중략
    private LoggerInfo log;

    public BoardServiceImpl() {
        log = new LoggerInfo();
    }

    // 글목록 구현
    @Override
    public List<BoardVO> boardList(BoardVO bvo) {
```

```

        log.printLogging();
        List<BoardVO> myList = null;
        myList = boardDao.boardList(bvo);
        return myList;
    }

    // 글입력 구현
    @Override
    public int boardInsert(BoardVO bvo) {
        log.printLogging();
        int result = 0;
        result = boardDao.boardInsert(bvo);
        return result;
    }

    // 글상세 구현
    @Override
    public BoardVO boardDetail(BoardVO bvo) {
        log.printLogging();
        BoardVO detail = null;
        detail = boardDao.boardDetail(bvo);
        return detail;
    }

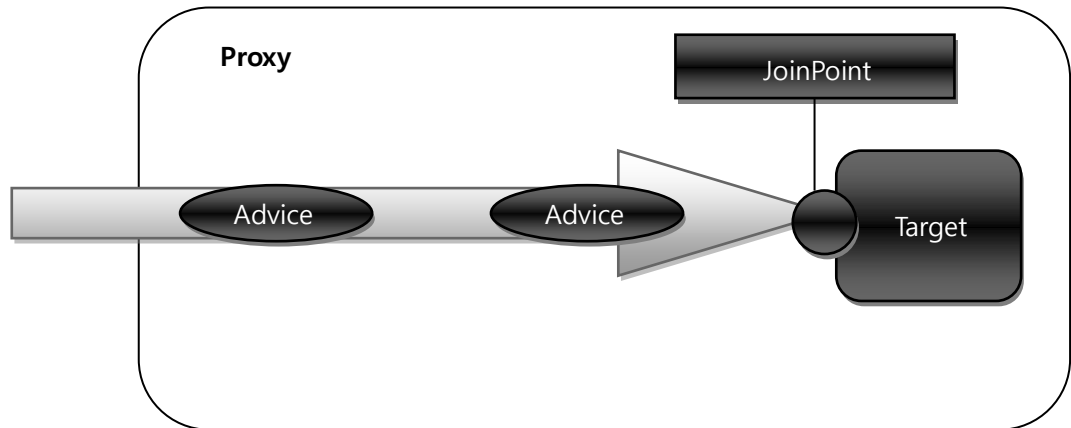
    ...
}

```

OOP처럼 모듈화가 뛰어난 언어를 사용하여 개발하더라도 공통 모듈에 해당하는 클래스의 객체를 생성하고 공통 메소드를 호출하는 코드가 비즈니스 메소드에 있다면, 핵심 관심과 횡단 관심을 완벽하게 분리할 수 없다. 하지만 스프링의 AOP는 이런 OOP의 한계를 극복할 수 있도록 도와준다.

2. AOP 용어들

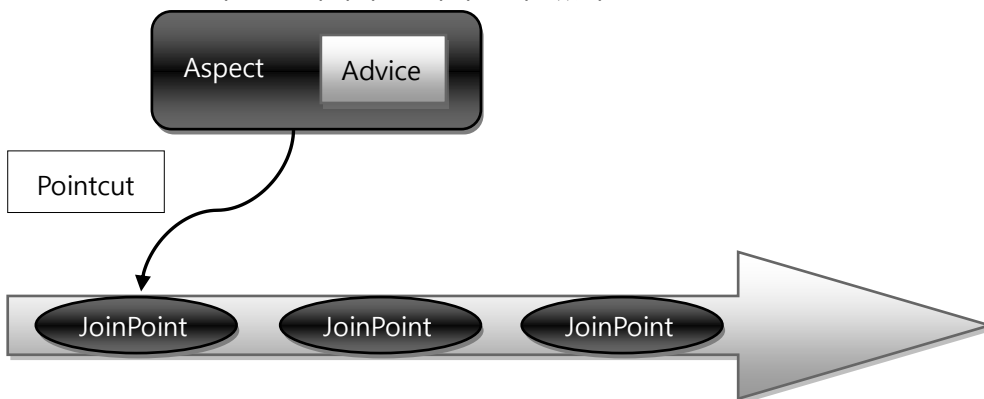
AOP는 기존의 코드를 수정하지 않고, 원하는 기능들과 결합할 수 있는 패러다임이다. AOP를 구현하기 위해서는 다음과 같은 핵심적인 그림들을 이해할 필요가 있다.



개발자의 입장에서 AOP를 적용한다는 것은 기존의 코드를 수정하지 않고도 원하는 관심사(cross-concern)들을 엮을 수 있다는 점이다. **Target에 해당하는 것이 바로 개발자가 작성한 핵심 비즈니스 로직을 가지는 객체이다.** (순수한 비즈니스 로직을 의미)

Target을 전체적으로 감싸고 있는 존재를 Proxy라고 한다. Proxy는 내부적으로 Target을 호출하지만, 중간에 필요한 관심사들을 거쳐서 Target을 호출하도록 자동 혹은 수동으로 작성한다. Proxy의 존재는 직접 코드를 통해서 구현하는 경우도 있지만, 대부분의 경우 스프링 AOP기능을 이용해서 자동으로 생성되는 (auto-proxy) 방식을 이용한다.

JoinPoint는 Target 객체가 가진 메서드이다. 외부에서의 호출은 Proxy 객체를 통해서 Target 객체의 JoinPoint를 호출하는 방식이라고 이해할 수 있다.



JoinPoint는 Target이 가진 여러 메서드라고 보면 된다. Target에는 여러 메서드가 존재하기 때문에 **어떤 메서드에 관심사를 결합할 것인지를 결정해야 하는데 이 결정을 'Pointcut'**이라고 한다.

Pointcut은 관심사와 비즈니스 로직이 결합되는 지점을 결정하는 것이다. Proxy는 이 결합이 완성된 상태이므로 메서드를 호출하게 되면 자동으로 관심사가 결합된 상태로 동작하게 된다. 관심사(concern)는 Aspect와 Advice라는 용어로 표현되어 있다. **Aspect는 관심사 자체를 의미하는 추상명사라고 볼 수 있고, Advice는 Aspect를 구현한 코드이다.** Advice는 실제 공통 기능의 코드를 의미한다.

Advice는 그 동작 위치에 따라 다음과 같이 구분된다.

구분	실행 시점	설명
Before Advice	비즈니스 메소드 실행 전 동작	Target의 JoinPoint를 호출하기 전에 실행되는 코드이다. 코드의 실행 자체에는 관여할 수 없다.
After Returning Advice	비즈니스 메소드가 성공적으로 리턴되면 동작	모든 실행이 정상적으로 이루어진 후에 동작하는 코드이다.
After Throwing Advice	비즈니스 메소드 실행 중 예외가 발생하면 동작(catch 블록)	예외가 발생한 뒤에 동작하는 코드이다
After Advice	비즈니스 메소드가 실행된 후 무조건 실행(finally 블록)	정상적으로 실행되거나 예외가 발생했을 때 구분 없이 실행되는 코드이다
Around	메소드 호출 자체를 가로채 비즈니스 메소드 실행 전후에 처리할 로직을 삽입.	메서드의 실행 자체를 제어할 수 있는 가장 강력한 코드이다. 직접 대상 메서드를 호출하고 결과나 예외를 처리할 수 있다.

Pointcut은 Advice를 어떤 JoinPoint에 결합할 것인지를 결정하는 설정이다. AOP에서 Target은 결과적으로 Pointcut에 의해서 자신에게는 없는 기능들을 가지게 된다. Pointcut은 다양한 형태로 선언해서 사용할 수 있는데 주로 사용되는 설정은 다음과 같다.

구분	실행 시점
execution(@execution)	메서드를 기준으로 Pointcut 을 설정한다
within(@within)	특정한 타입(클래스)을 기준으로 Pointcut을 설정한다.
args(@args)	특정한 파라미터를 가지는 대상들만을 Pointcut으로 설정한다
@annotation	특정한 어노테이션이 적용된 대상들만을 Pointcut으로 설정한다.

3. AOP 시작

스프링의 AOP를 이용해서 핵심 관심과 횡단 관심을 분리한다. 그래서 BoardServiceImpl 소스와 무관하게 LoggerAdvice 클래스의 메소드를 실행할 수 있다.

3.1 AOP 라이브러리 추가

AOP를 적용하기 위해서 우선 pom.xml 파일에 AOP 관련 라이브러리를 추가한다.

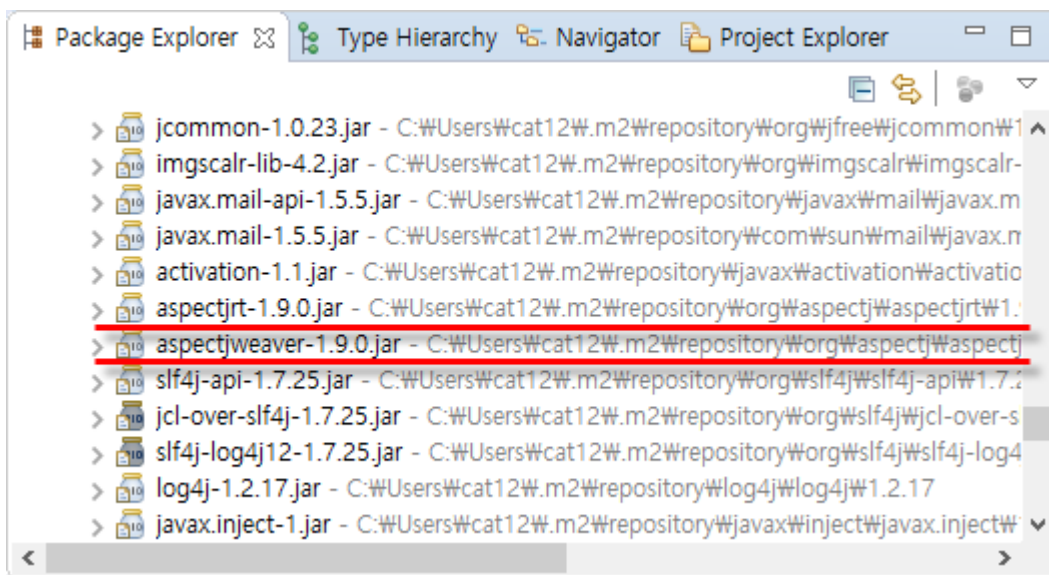
pom.xml
<pre> <org.aspectj-version>1.9.0</org.aspectj-version> 생략 <!-- AspectJ --> <dependency> <groupId>org.aspectj</groupId> <artifactId>aspectjrt</artifactId> <version>\${org.aspectj-version}</version> </dependency> </pre>

```

<!-- AOP 관련 라이브러리 추가 -->
<!-- https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>${org.aspectj-version}</version>
</dependency>

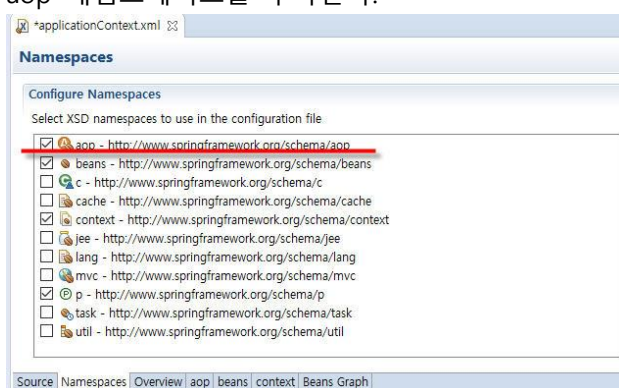
```

...생략



3.2 네임스페이스 추가 및 AOP 설정

AOP 설정을 추가하려면 AOP에서 제공하는 엘리먼트들을 사용해야 한다. 따라서 스프링 설정 파일에서 aop 네임스페이스를 추가한다.



자동으로 Proxy 객체를 만들어주는 설정을 추가해 주면 된다.

/WEB-INF/spring/appServlet/servlet-context.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"

```

```

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:beans="http://www.springframework.org/schema/beans"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:aop="http://www.springframework.org/schema/aop"
xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd">

<!-- DispatcherServlet Context: defines this servlet's request-processing infrastructure -->

<!-- Enables the Spring MVC @Controller programming model -->
<annotation-driven />

<!-- Handles HTTP GET requests for /resources/** by efficiently serving up static resources in
the ${webappRoot}/resources directory -->
<resources mapping="/resources/**" location="/resources/" />

<context:component-scan base-package="com.spring.**" />

<!-- 타일즈(tiles) 설정 - ViewResolver은 타일즈를 통해서, 설정파일에 존재하지 않으면
우선순위2로 -->
<beans:bean id="tilesConfigurer"
class="org.springframework.web.servlet.view.tiles3.TilesConfigurer">
    <beans:property name="definitions">
        <beans:list>
            <beans:value>/WEB-INF/tiles/tiles-setting.xml</beans:value>
        </beans:list>
    </beans:property>
</beans:bean>

<beans:bean id="tilesViewResolver"
class="org.springframework.web.servlet.view.UrlBasedViewResolver">
    <beans:property name="viewClass"
value="org.springframework.web.servlet.view.tiles3.TilesView" />
    <beans:property name="order" value="1" /> <!-- 우선순위설정 -->

```

```

</beans:bean>

<beans:bean id="viewResolver"
  class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <beans:property name="prefix" value="/WEB-INF/views/" />
    <beans:property name="suffix" value=".jsp" />
    <beans:property name="order" value="2" />
</beans:bean>

<!-- 자동으로 AspectJ 라이브러리를 이용해서 Proxy 객체를 생성해 내는 용도로 사용. -->
<aop:aspectj-autoproxy />
</beans:beans>

```

3.3 Advice 작성

3.3.1 @Before

com.spring.common.log.LoggerAdvice

```

package com.spring.common.log;

import lombok.extern.log4j.Log4j;

import org.springframework.stereotype.Component;

import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

@Log4j
/* 스프링에서 빈(bean)으로 인식하기 위해서 사용*/
@Component
/* 해당 클래스의 객체가 Aspect를 구현한 것임으로 나타내기 위해서 사용*/
@Aspect
public class LoggerAdvice {

    /* execution(@execution) 메서드를 기준으로 Pointcut을 설정 */
    @Before("execution(* com.spring..*Impl.*(..))")
    public void printLogging(){
        log.info("-----");
        log.info("[공통 로그 Log] 비즈니스 로직 수행 전 동작");
        log.info("-----");
    }
}

```


Advice와 관련된 어노테이션들은 내부적으로 Pointcut을 지정한다. Pointcut은 별도의 어노테이션으로 지정해서 사용할 수 있다. @Before내부의 "execution(* com.spring..*Impl.*(..))" 문자열은 Aspect의 표현식이다. 'execution'의 경우 리턴타입과 특정 클래스의 메서드를 지정할 수 있다. 맨 앞의 '*'는 보통 리턴타입을 의미하고, 맨 뒤의 '*'는 클래스의 이름과 메서드이름을 의미한다.

*	com.spring..	*Impl	.	*(..)
*	com.spring..	*Impl	.	*List(..)
리턴 타입	패키지 경로	클래스명		메소드명 및 매개변수

```

Tomcat v8.5 Server at localhost [Apache Tomcat] C:\Java\jdk1.8\bin\javaw.exe (2020. 3. 26. 오후 3:57:29)
INFO : com.spring.client.board.controller.BoardController - boardList 호출 성공
INFO : com.spring.common.log.LoggerAdvice - -----
INFO : com.spring.common.log.LoggerAdvice - [공통 로그 Log] 비즈니스 로직 수행 전 동작
INFO : com.spring.common.log.LoggerAdvice - -----
INFO : com.spring.common.log.LoggerAdvice - -----
INFO : com.spring.common.log.LoggerAdvice - [공통 로그 Log] 비즈니스 로직 수행 전 동작
INFO : com.spring.common.log.LoggerAdvice - -----
  
```

com.spring.common.log.LoggerAdvice

```

package com.spring.common.log;

import com.spring.client.board.vo.BoardVO;
import lombok.extern.log4j.Log4j;
import org.springframework.stereotype.Component;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

@Log4j
/* 스프링에서 빈(bean)으로 인식하기 위해서 사용*/
@Component
/* 해당 클래스의 객체가 Aspect를 구현한 것임으로 나타내기 위해서 사용*/
@Aspect
public class LoggerAdvice {
    //조금 전에 수행한 메서드는 주석 또는 아래의 내용으로 변경하면 된다.
    @Before("execution(* com.spring..*Impl.*(..)) && args(bvo)")
    public void printLogging(BoardVO bvo){
        log.info("-----");
        log.info("[공통 로그 Log] 비즈니스 로직 수행 전 동작");
        log.info("-----");
        log.info("BoardVO 타입의 bvo 파라미터 값 : " + bvo);
    }
}
  
```

설정 시에 args를 이용하면 간단히 파라미터를 구할 수 있다.

```

Tomcat v8.5 Server at localhost [Apache Tomcat] C:\java\jdk1.8\bin\javaw.exe (2020. 3. 26. 오후 3:57:29)
INFO : com.spring.client.board.controller.BoardController - boardDetail 호출 성공
INFO : com.spring.common.log.LoggerAdvice - -----
INFO : com.spring.common.log.LoggerAdvice - [공통 로그 Log] 비즈니스 로직 수행 전 동작
INFO : com.spring.common.log.LoggerAdvice - -----
INFO : com.spring.common.log.LoggerAdvice - BoardVO 타입의 bvo 파라미터 값 : BoardVO(b_num=143, b_name=
INFO : com.spring.client.reply.controller.ReplyController - list 호출 성공
  
```

스프링에서는 다양한 정보들을 이용할 수 있도록 JoinPoint 인터페이스를 제공한다.

JoinPoint가 제공하는 메소드

메소드	설명
Signature getSignature()	클라이언트가 호출한 메소드의 시그니처(리턴 타입, 이름, 매개변수) 정보가 저장된 Signature 객체 리턴
Object getTarget()	클라이언트가 호출한 비즈니스 메소드를 포함하는 비즈니스 객체 리턴
Object[] getArgs()	클라이언트가 메소드를 호출할 때 넘겨준 인자목록을 Object 배열로 리턴

Signature가 제공하는 메소드

메소드	설명
String getName()	클라이언트가 호출한 메소드 이름 리턴
String toLongString()	클라이언트가 호출한 메소드의 리턴 타입, 이름, 매개변수를 패키지 경로까지 포함하여 리턴
String toShortString()	클라이언트가 호출한 메소드 시그니처를 축약한 문자열로 리턴

```

com.spring.common.log.LoggerAdvice
package com.spring.common.log;

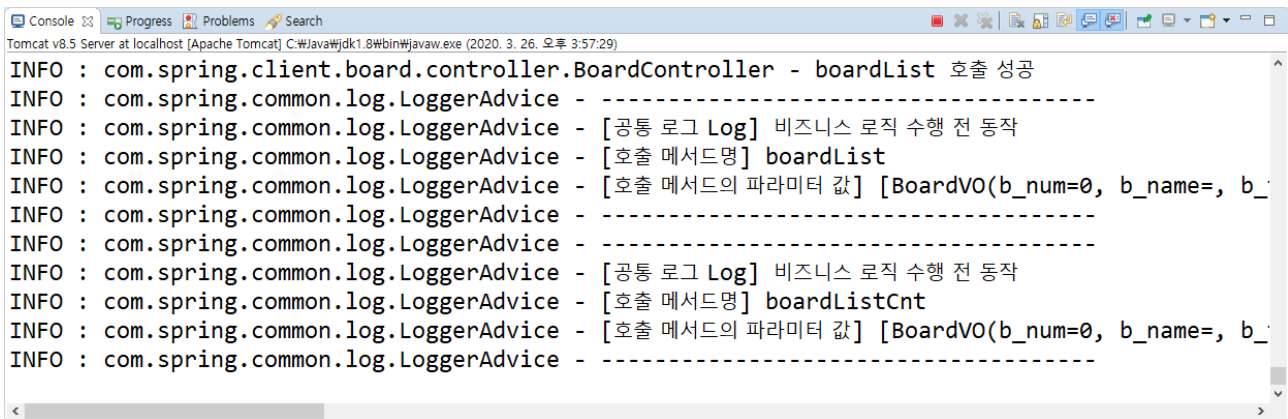
//import com.spring.client.board.vo.BoardVO;
import lombok.extern.log4j.Log4j;
import org.springframework.stereotype.Component;

import java.util.Arrays;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

@Log4j
/* 스프링에서 빈(bean)으로 인식하기 위해서 사용*/
@Component
/* 해당 클래스의 객체가 Aspect를 구현한 것임으로 나타내기 위해서 사용*/
  
```

@Aspect

```
public class LoggerAdvice {  
    // 조금 전에 수행한 메서드는 주석 또는 아래의 내용으로 변경하면 된다.  
    @Before("execution(* com.spring.*Impl.*(..))")  
    public void printLogging(JoinPoint jp){  
        log.info("-----");  
        log.info("[공통 로그 Log] 비즈니스 로직 수행 전 동작");  
        // getArgs() : 전달되는 모든 파라미터들을 Object의 배열로 가져온다.  
        // getSignature() : 실행하는 대상 객체의 대한 정보를 알아낼 때 사용.  
        log.info("[호출 메서드명] " + jp.getSignature().getName());  
        log.info("[호출 메서드의 파라미터 값] " + Arrays.toString(jp.getArgs()));  
        log.info("-----");  
    }  
}
```



The screenshot shows a console window with the following log output:

```
INFO : com.spring.client.board.controller.BoardController - boardList 호출 성공  
INFO : com.spring.common.log.LoggerAdvice - -----  
INFO : com.spring.common.log.LoggerAdvice - [공통 로그 Log] 비즈니스 로직 수행 전 동작  
INFO : com.spring.common.log.LoggerAdvice - [호출 메서드명] boardList  
INFO : com.spring.common.log.LoggerAdvice - [호출 메서드의 파라미터 값] [BoardVO(b_num=0, b_name=, b_...]  
INFO : com.spring.common.log.LoggerAdvice - -----  
INFO : com.spring.common.log.LoggerAdvice - [공통 로그 Log] 비즈니스 로직 수행 전 동작  
INFO : com.spring.common.log.LoggerAdvice - [호출 메서드명] boardListCnt  
INFO : com.spring.common.log.LoggerAdvice - [호출 메서드의 파라미터 값] [BoardVO(b_num=0, b_name=, b_...]  
INFO : com.spring.common.log.LoggerAdvice - -----
```

3.3.2 @AfterThrowing

코드를 실행하다 보면 파라미터의 값이 잘못되어서 예외가 발생하는 경우에 AOP의 @AfterThrowing 어노테이션은 지정된 대상이 예외를 발생한 후에 동작하면서 문제를 찾을 수 있도록 도와준다.

com.spring.common.log.LoggerAdvice

```
package com.spring.common.log;  
  
//import com.spring.client.board.vo.BoardVO;  
import lombok.extern.log4j.Log4j;  
import org.springframework.stereotype.Component;  
  
//import java.util.Arrays;  
  
import org.aspectj.lang.JoinPoint;  
import org.aspectj.lang.annotation.AfterThrowing;  
import org.aspectj.lang.annotation.Aspect;
```

```

//import org.aspectj.lang.annotation.Before;

@Log4j
/* 스프링에서 빈(bean)으로 인식하기 위해서 사용*/
@Component
/* 해당 클래스의 객체가 Aspect를 구현한 것임으로 나타내기 위해서 사용*/
@Aspect
public class LoggerAdvice {
    //조금 전에 수행한 메서드는 주석을 주거나 그대로 명시해 두어도 된다.

    /* 예외가 발생한 시점에 동작 */
    @AfterThrowing(pointcut="execution(* com.spring..*Impl.*(..)", throwing="exception")
    public void exceptionLogging(JoinPoint jp, Throwable exception){
        log.info("-----");
        log.info("[예외발생] ");
        log.info("[예외발생 메서드명] " + jp.getSignature().getName());
        //exception.printStackTrace();
        log.info("[예외 메시지] " + exception);
        log.info("-----");
    }
}

```

위의 내용을 확인하기 위해서는 예외를 발생시켜야 하기에 아래의 내용을 추가하여 강제로 예외가 발생하도록 한다.

```

com.spring.client.board.service.BoardServiceImpl

package com.spring.client.board.service;

... 중략
@Service
public class BoardServiceImpl implements BoardService {
    ... 중략

    // 글입력 구현
    @Override
    public int boardInsert(BoardVO bvo) {
        int result = 0;

        /* 예외를 발생시킬 코드 작성 */
        bvo.setB_num(0);
        if(bvo.getB_num() == 0){
            throw new IllegalArgumentException("0번 글은 등록할 수 없습니다.");
        }
    }
}

```

```

    }

    result = boardDao.boardInsert(bvo);
    return result;
}
... 중략
}

```

```

Tomcat v8.5 Server at localhost (Apache Tomcat) C:\Java\jdk1.8\bin\javaw.exe (2020. 3. 26. 오후 3:57:29)
INFO : com.spring.client.board.controller.BoardController - writeForm 호출 성공
INFO : com.spring.client.board.controller.BoardController - boardInsert 호출 성공
INFO : com.spring.common.log.LoggerAdvice - -----
INFO : com.spring.common.log.LoggerAdvice - [예외발생]
INFO : com.spring.common.log.LoggerAdvice - [예외발생 메서드명] boardInsert
INFO : com.spring.common.log.LoggerAdvice - [예외 메시지] java.lang.IllegalArgumentException: 0번 글은 등록할 수 없습니다.
INFO : com.spring.common.log.LoggerAdvice - -----
ERROR: com.spring.common.exception.CommonExceptionAdvice - Exception .....0번 글은 등록할 수 없습니다.
ERROR: com.spring.common.exception.CommonExceptionAdvice - {exception=java.lang.IllegalArgumentException: 0번 글은 등

```

3.3.3 @AfterReturning

com.spring.common.log.LoggerAdvice

```

package com.spring.common.log;

//import com.spring.client.board.vo.BoardVO;
import lombok.extern.log4j.Log4j;
import org.springframework.stereotype.Component;

//import java.util.Arrays;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Aspect;
//import org.aspectj.lang.annotation.Before;

@Log4j
/* 스프링에서 빈(bean)으로 인식하기 위해서 사용*/
@Component
/* 해당 클래스의 객체가 Aspect를 구현한 것임으로 나타내기 위해서 사용*/
@Aspect
public class LoggerAdvice {
    /* 예외가 발생한 시점에 동작 */
    @AfterThrowing(pointcut="execution(* com.spring.*.Impl.*(..))", throwing="exception")
    public void exceptionLogging(JoinPoint jp, Throwable exception){
        log.info("-----");
    }
}

```

```

        log.info("[예외발생] ");
        log.info("[예외발생 메서드명] " + jp.getSignature().getName());
        //exception.printStackTrace();
        log.info("[예외 메시지] " + exception);
        log.info("-----");
    }
    /* 비즈니스 로직 메서드가 정상적으로 수행 된 후 동작 */
    @AfterReturning(pointcut="execution(* com.spring..service.*Impl.*(..))",
                    returning = "returnValue")
    public void afterReturningMethod(JoinPoint jp, Object returnValue) {
        log.info("-----");
        log.info("[공통 로그 Log] 비즈니스 로직 수행 후 동작");
        log.info("afterReturningMethod() called....." + jp.getSignature().getName());
        log.info("[리턴 결과] " + returnValue);
        log.info("-----");
    }
}

```

```

Tomcat v8.5 Server at localhost [Apache Tomcat] C:\Java\jdk1.8\bin\javaw.exe (2020. 3. 26. 오후 3:57:29)
INFO : com.spring.client.board.controller.BoardController - boardList 호출 성공
INFO : com.spring.common.log.LoggerAdvice - -----
INFO : com.spring.common.log.LoggerAdvice - [공통 로그 Log] 비즈니스 로직 수행 후 동작
INFO : com.spring.common.log.LoggerAdvice - afterReturningMethod() called.....boardList
INFO : com.spring.common.log.LoggerAdvice - [리턴 결과] [BoardVO(b_num=143, b_name=홍길동120, b_title=페0
INFO : com.spring.common.log.LoggerAdvice - -----
INFO : com.spring.common.log.LoggerAdvice - -----
INFO : com.spring.common.log.LoggerAdvice - [공통 로그 Log] 비즈니스 로직 수행 후 동작
INFO : com.spring.common.log.LoggerAdvice - afterReturningMethod() called.....boardListCnt
INFO : com.spring.common.log.LoggerAdvice - [리턴 결과] 131
INFO : com.spring.common.log.LoggerAdvice - -----

```

3.3.4 @Around와 ProceedingJoinPoint

@Around는 직접 대상 메서드를 실행할 수 있는 권한을 가지고 있고, 메서드의 실행 전과 실행 후에 처리가 가능하다. ProceedingJoinPoint는 @Around와 같이 결합해서 파라미터나 예외 등을 처리할 수 있다.

com.spring.common.log.LoggerAdvice

```

package com.spring.common.log;

//import com.spring.client.board.vo.BoardVO;
import lombok.extern.log4j.Log4j;
import org.springframework.stereotype.Component;
import org.springframework.util.StopWatch;

import java.util.Arrays;
//import org.aspectj.lang.JoinPoint;

```

```

import org.aspectj.lang.ProceedingJoinPoint;
//import org.aspectj.lang.annotation.AfterReturning;
//import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
//import org.aspectj.lang.annotation.Before;

@Log4j
/* 스프링에서 빈(bean)으로 인식하기 위해서 사용*/
@Component
/* 해당 클래스의 객체가 Aspect를 구현한 것임으로 나타내기 위해서 사용*/
@Aspect
public class LoggerAdvice {

    //조금 전에 수행한 메서드는 주석 처리하면 된다.
    @Around("execution(* com.spring..*Impl.*(..))")
    public Object timeLogging(ProceedingJoinPoint pjp) throws Throwable{
        log.info("-----");
        log.info("[공통 로그 Log] 비즈니스 로직 수행 전 동작");

        //long startTime = System.currentTimeMillis();
        Stopwatch watch = new Stopwatch();
        watch.start();
        log.info("[호출 메서드의 파라미터 값] " + Arrays.toString(pjp.getArgs()));

        Object result = null;
        // proceed() : 실제 target 객체의 메서드를 실행하는 기능.
        result = pjp.proceed();

        //long endTime = System.currentTimeMillis();
        watch.stop();

        log.info("[Class] " + pjp.getTarget().getClass());

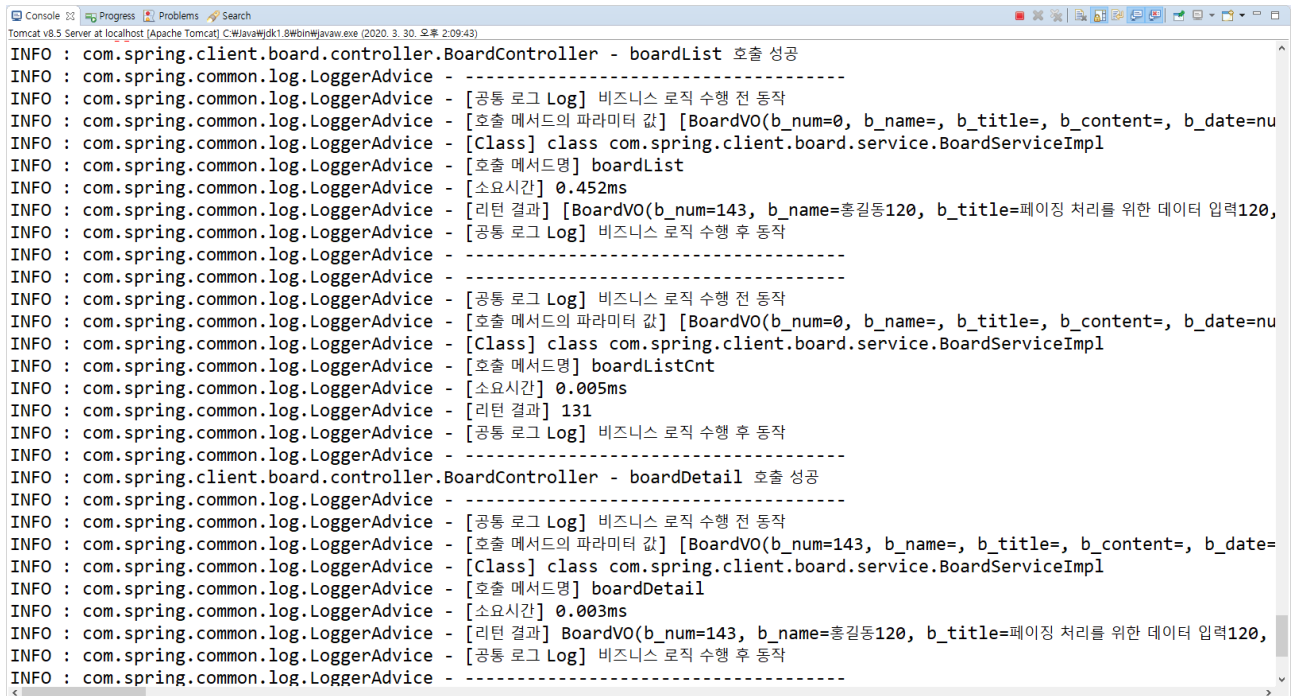
        //logger.info(pjp.getSignature().getName()+":소요시간 " + (endTime-startTime)+"ms");
        log.info("[호출 메서드명] " + pjp.getSignature().getName() );
        log.info("[소요시간] " + watch.getTotalTimeSeconds() +"ms");
        log.info("[리턴 결과] " + result);
        log.info("[공통 로그 Log] 비즈니스 로직 수행 후 동작");
        log.info("-----");
    }
}

```

```

        return result;
    }
}

```



Controller의 Exception 처리

Controller를 작성할 때 예외 상황에 대해 Spring MVC에서는 다음과 같은 방식으로 처리할 수 있다.

- @ControllerAdvice와 @ExceptionHandler를 이용한 처리.

```

com.spring.common.exception.CommonExceptionAdvice
package com.spring.common.exception;

import org.springframework.http.HttpStatus;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.servlet.NoHandlerFoundException;

import lombok.extern.log4j.Log4j;

/* 해당 객체가 컨트롤러에서 발생하는 예외를 처리하는 존재임을 명시하는 용도로 사용. */
@ControllerAdvice
@Log4j

```



```

public class CommonExceptionHandler {
    /* @ExceptionHandler는 해당 메서드가 () 들어가는 예외 타입을 처리. */
    @ExceptionHandler(Exception.class)
    public String exceptionMethod(Exception ex, Model model) {
        log.error("Exception ..... " + ex.getMessage());
        model.addAttribute("exception", ex);
        log.error(model);
        return "error/error_page";
    }
}

```

/WEB-INF/views/error/error_page.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="java.util.*"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0,
            maximum-scale=1.0, minimum-scale=1.0, user-scalable=no" />
        <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />

        <!-- 모바일 웹 페이지 설정 -->
        <link rel="shortcut icon" href="/resources/images/icon.png" />
        <link rel="apple-touch-icon" href="/resources/images/icon.png" />
        <!-- 모바일 웹 페이지 설정 끝 -->

        <title>에러 페이지</title>

        <!--[if lt IE 9]>
        <script src="/resources/include/js/html5shiv.js"> </script>
        <![endif]-->

        <link rel="stylesheet" type="text/css"
            href="/resources/include/dist/css/bootstrap.min.css" />
        <link rel="stylesheet" type="text/css"
            href="/resources/include/dist/css/bootstrap-theme.min.css" />

```

```

<script type="text/javascript"
    src="/resources/include/js/jquery-1.12.4.min.js"> </script>
<script type="text/javascript" src="/resources/include/js/jquery.form.min.js"> </script>
<script type="text/javascript" src="/resources/include/js/common.js"> </script>

<script type="text/javascript">
    $(function(){
        //close 인스턴스 메소드가 호출되는 즉시 실행
        $('#errorAlert').on('closed.bs.alert', function () {
            location.href = "/";
        });
        $("#main").click(function(){
            location.href = "/";
        });
    });
</script>
</head>
<body>
    <!-- 이 부분은 개발 당시에는 사용하면 된다. (예외 메시지를 확인하기 위해서)
    <h4><c:out value="${exception.getMessage()}"> </c:out> </h4>
    <ul>
        <c:forEach items="${exception.getStackTrace()}" var="stack">
            <li><c:out value="${stack}"> </c:out> </li>
        </c:forEach>
    </ul> --%>

    <!-- 개발이 완료되면 이 부분을 주석해제 하면 된다. --%>
    <div class="alert alert-danger alert-dismissible fade in" role="alert" id="errorAlert">
        <button type="button" class="close" data-dismiss="alert" aria-label="Close">
            <span aria-hidden="true">&times;</span></button>
        <h4>예상하지 못한 오류가 발생했습니다.</h4>
        <p>
            일시적인 현상이거나 네트워크 문제일 수 있으니, 잠시 후 다시
            시도해 주세요.<br />
            계속 발생하는 경우 홈페이지를 통해 문의해 주세요.<br />
            감사합니다<br />
        </p>
        <p>
            <button type="button" class="btn btn-danger" id="main">홈으로</button>
        </p>
    </div>

```

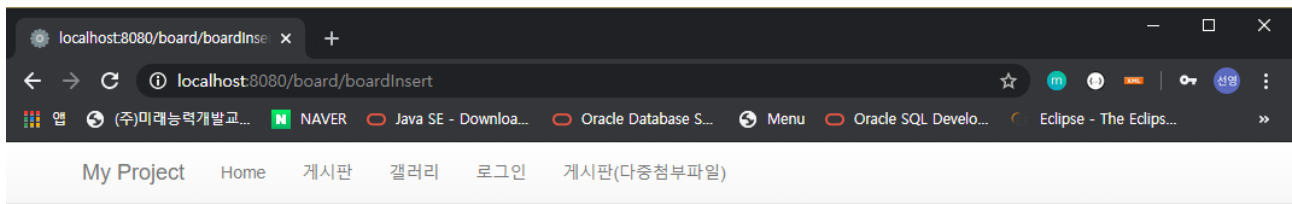
```

        </div>

    </body>

</html>

```



예상하지 못한 오류가 발생했습니다.

일시적인 현상이거나 네트워크 문제일 수 있으니, 잠시 후 다시 시도해 주세요.
계속 발생하는 경우 홈페이지를 통해 문의해 주세요.
감사합니다

홈으로

클릭시 메인(<http://localhost:8080>)으로 이동한다.

사업자등록번호: 128-23-45267 | 대표: 홍길동 | 개인정보보호책임자: 김철수 | 이메일 : javauser1234@naver.com | 서울특별시 성동구 왕십리로 303 4층

com.spring.common.exception.CommonExceptionAdvice

```

package com.spring.common.exception;

import org.springframework.http.HttpStatus;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.servlet.NoHandlerFoundException;

import lombok.extern.log4j.Log4j;

/* 해당 객체가 컨트롤러에서 발생하는 예외를 처리하는 존재임을 명시하는 용도로 사용. */
@ControllerAdvice
@Log4j
public class CommonExceptionAdvice {
    //기존 코드에 아래의 내용을 추가한다.

    /* 매핑 정보나 파일을 찾지 못했을 경우에 대한 예외 처리. */
    @ExceptionHandler(NoHandlerFoundException.class)
    @ResponseStatus(HttpStatus.NOT_FOUND)
    public String handle404(NoHandlerFoundException ex) {

```

```

        return "error/custom404";
    }
}

```

스프링 MVC의 모든 요청은 DispatcherServlet을 이용해서 처리하므로 404 에러도 같이 처리할 수 있도록 web.xml을 수정해야 한다

/WEB-INF/web.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" ...중략>

    ...중략
    <!-- Processes application requests -->
    <servlet>
        <servlet-name>appServlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
        </init-param>
        <init-param>
            <param-name>throwExceptionIfNoHandlerFound</param-name>
            <param-value>true</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>appServlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

    ...중략
</web-app>

```

/WEB-INF/views/error/custom404.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>

```

```

<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0,
    maximum-scale=1.0, minimum-scale=1.0, user-scalable=no" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />

    <!-- 모바일 웹 페이지 설정 -->
    <link rel="shortcut icon" href="/resources/images/icon.png" />
    <link rel="apple-touch-icon" href="/resources/images/icon.png" />
    <!-- 모바일 웹 페이지 설정 끝 -->

    <title>에러 페이지</title>

    <!--[if lt IE 9]>
    <script src="/resources/include/js/html5shiv.js"> </script>
    <![endif]-->

    <link rel="stylesheet" type="text/css"
    href="/resources/include/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" type="text/css"
    href="/resources/include/dist/css/bootstrap-theme.min.css" />

    <script type="text/javascript"
    src="/resources/include/js/jquery-1.12.4.min.js"> </script>
    <script type="text/javascript" src="/resources/include/js/jquery.form.min.js"> </script>
    <script type="text/javascript" src="/resources/include/js/common.js"> </script>

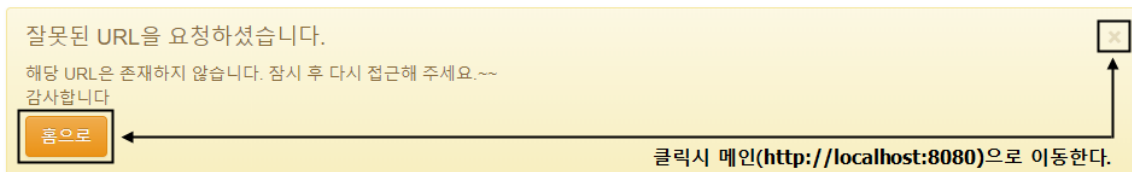
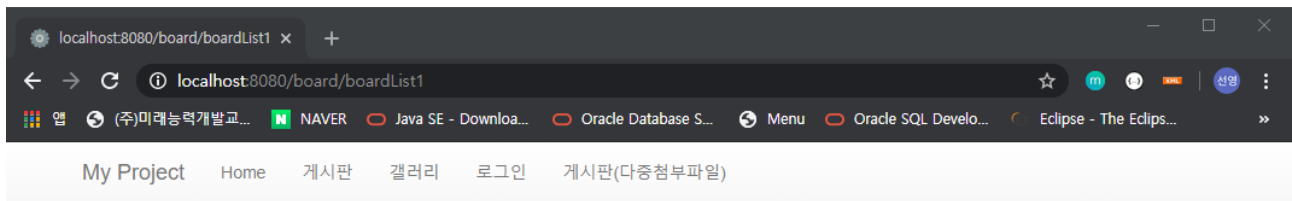
    <script type="text/javascript">
        $(function(){
            //close 인스턴스 메소드가 호출되는 즉시 실행
            $('#errorAlert').on('closed.bs.alert', function () {
                location.href = "/";
            });
            $("#main").click(function(){
                location.href = "/";
            });
        });
    </script>
</head>
<body>

```

```

<div class="alert alert-warning alert-dismissible fade in" role="alert" id="errorAlert">
    <button type="button" class="close" data-dismiss="alert" aria-label="Close">
        <span aria-hidden="true">&times;</span> </button>
    <h4>잘못된 URL을 요청하셨습니다. </h4>
    <p>
        해당 URL은 존재하지 않습니다. 잠시 후 다시 접근해 주세요.~~<br />
        감사합니다<br />
    </p>
    <p>
        <button type="button" class="btn alert-warning" id="main">홈으로</button>
    </p>
</div>
</body>
</html>

```



사업자등록번호: 128-23-45267 | 대표: 홍길동 | 개인정보보호책임자: 김철수 | 이메일 : javauser1234@naver.com | 서울특별시 성동구 왕십리로 303 4층