

자바 - 프레젠테이션(웹) 계층의 구현

1. Controller 의 작성

스프링 MVC 의 Controller는 하나의 클래스 내에서 여러 메서드를 작성하고,

@RequestMapping 등을 이용하여 URLdf 분기하는 구조로 작성할 수 있기 때문에

하나의 클래스에서 필요한 만큼 메서드의 분기를 이용하는 구조로 작성

이전방식 : Tomcat(WAS)을 실행하고,

웹 화면을 만들어서 결과를 확인하는 방식의 코드를 작성

이전방식의 단점 : 오래 걸리고, 테스트를 자동화 하기가 어렵다.

WAS를 실행하지 않고 Controller 를 테스트 할 수 있는 방법으로 진행

1. BoardController의 분석

작성하기 전에 이루어지는 사전 작업

반드시 현재 원하는 기능을 호출하는 방식에 대하여 테이블(표)로 정리한 후, 작성

테이블에서 from 항목은 해당 url 호출하기 위해 별도의 입력 화면이 필요하다는 것을 의미

Task	URL	Method	Parameter	From	URL이름
전체 목록	/board/list	GET			
등록 처리	/board/register	POST	모든 항목	입력화면 필요	이동
조회	/board/read	GET	bno=123		
삭제 처리	/board/remove	POST	bno	입력화면 필요	이동
수정 처리	/board/modify	POST	모든 항목	입력화면 필요	이동

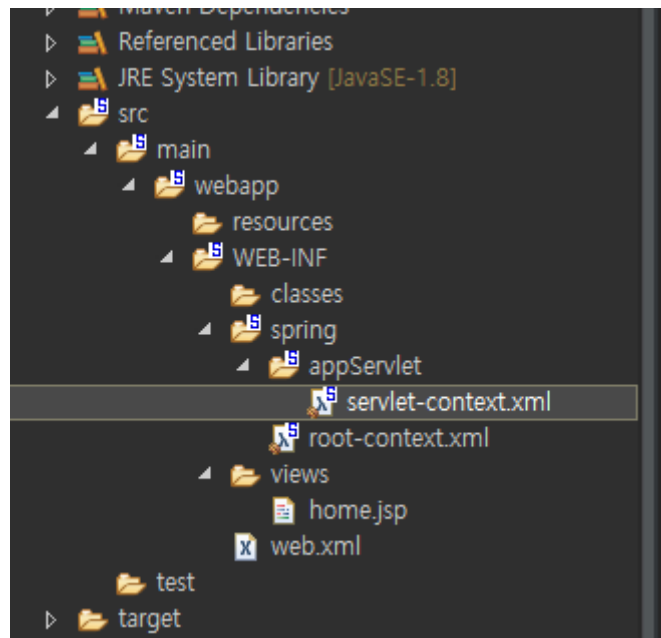
BoardController 는 com.이니셜.controller 패키지에 선언.

URL 분석된 내용들을 반영하는 메서드를 설계.

@Controller 어노테이션을 추가하여 스프링의 빈으로 인식할 수 있도록 한다.

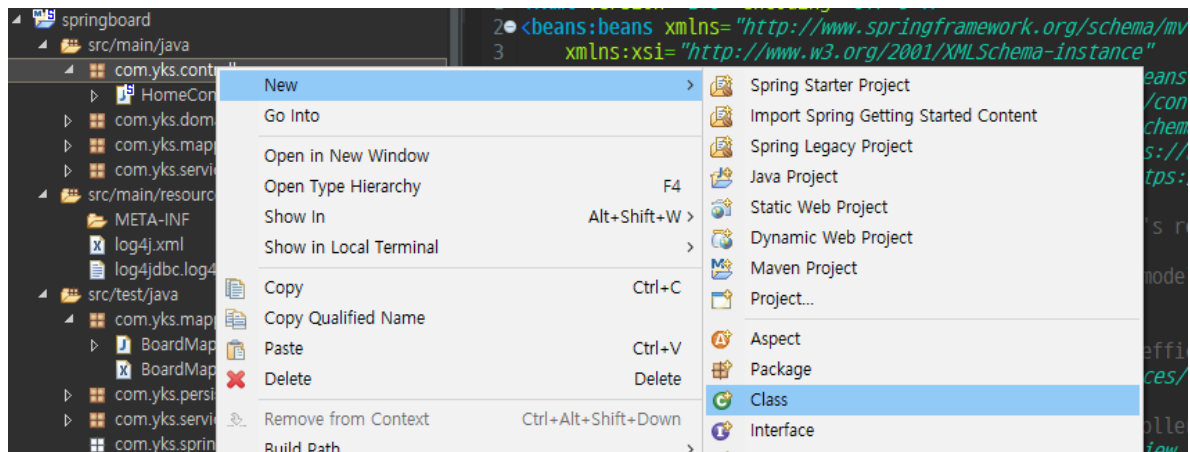
@RequestMapping 을 통하여 '/board' 로 시작하는 모든 처리를 BoardController 가 담당하도록 설정.


BoardController 가 속한 패키지는 servlet-context.xml 에 기본적으로 설정되어 있으므로 별도의 설정이 필요없다.



```
<context:component-scan base-package="com.yks.controller" />
```

위 코드가 들어가있는지 확인한다.



 New Java Class

Java Class
Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:


Modifiers: ☐ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?
☐ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments



```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import com.yks.domain.BoardVO;
import com.yks.service.BoardService;

import lombok.AllArgsConstructor;
import lombok.extern.log4j.Log4j;
```

클래스 선언부에 어노테이션 추가

BoardController는 BoardService 에 대하여 의존적이므로

@AllArgsConstructor 를 이용하여 생성자를 만들고, 자동 주입

만약, 생성자를 만들지 않을 경우에는 반드시

@Setter(onMethod_ = {@Autowired}) 를 이용하여 처리

```
@Controller
@Log4j
@RequestMapping("/board/*")    // 요청하는 URL이 /board/로 시작하면 무조건 이 컨트롤러 실행
@AllArgsConstructor           // 생성자를 통한 자동 주입
public class BoardController {
```

조회를 위한 list() 메서드 추가

list()는 나중에 게시물의 목록을 전달해야 하므로 Model 객체를 파라미터

이를 통하여 BoardServiceImpl 객체의 getList() 결과를 담아 전달

```
/springboard/src/main/java/com/yks/controller/BoardController.java

public class BoardController {

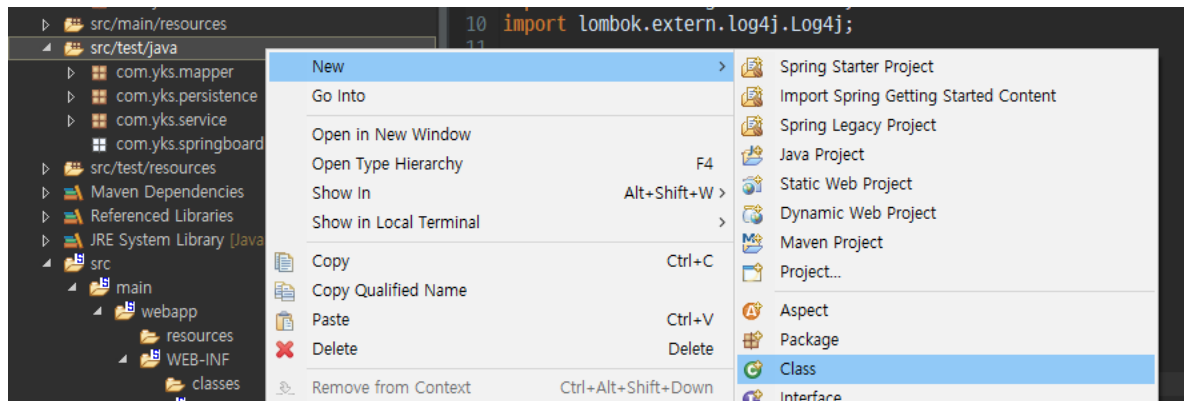
    private BoardService service;

    @GetMapping("/list")
    public void list(Model model) {

        log.info("list");
        model.addAttribute("list", service.getList());
    } // 이 리스트 메소드가 호출되려면 /board/list 로 되어야 한다

}
```

테스트를 위한 BoardControllerTests 클래스 작성



Java Class
Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?
☐ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments

테스트 코드는

웹을 개발하면서 매번 URL을 테스트하기 위해

Tomcat과 같은 WAS를 실행하는 불편한 단계를 생략할 수 있도록 코드를 작성.

이 때 필요한 클래스

- WebAppConfiguration
- WebApplicationContext
- MockMvc
- MockMvcBuilders
- MockMvcRequestBuilders

필요한 클래스 import

```
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.web.WebAppConfiguration;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;
import org.springframework.web.context.WebApplicationContext;

import lombok.Setter;
import lombok.extern.log4j.Log4j;
```

클래스 선언 부에 어노테이션 추가

@WebAppConfiguration : Servlet의 ServletContext를 이용하기 위함.

즉, 스프링에서는 WebApplicationContext 라는 객체를 이용하기 위함이다.

```
@RunWith(SpringJUnit4ClassRunner.class)

// Test for Controller
@WebAppConfiguration

@ContextConfiguration({ "file:src/main/webapp/WEB-INF/spring/root-context.xml",
    "file:src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml" })
@Log4j
public class BoardControllerTests {
```

의존성 주입을 위한 멤버(인스턴스) 변수 및 메서드 추가

MockMvc : 가짜 mvc

가상으로 URL 과 파라미터 등을 마치 브라우저에서 사용하는 것처럼 만들어서 Controller 를 실행해 볼 수 있다.

```
public class BoardControllerTests {

    @Setter(onMethod_ = { @Autowired })
    private WebApplicationContext ctx;

    private MockMvc mockMvc;

}
```

@Before : import 할 때 JUnit을 이용.

이 어노테이션이 적용된 메서드는 모든 테스트 전에 매번 실행되는 메서드가 된다.

```
@Before
public void setup() {
    this.mockMvc = MockMvcBuilders.webAppContextSetup(ctx).build();
}
```

MockMvcRequestBuilder 를 이용하면 마치 GET방식의 호출을 할 수 있다.

```
@Test
public void testList() throws Exception {

    log.info(mockMvc.perform(MockMvcRequestBuilders.get("/board/list")).andReturn()
        .getModelAndView().getModelMap());
}
```

bno	title	content	writer	regdate	update date	
1	제목을 수정합니다.	{테스트 내용3	user03	{2020-05-13 09:46:33.0	{2020-05-13 16:21:50.0	
2	테스트제목	{테스트 내용	user00	{2020-05-13 09:49:39.0	{2020-05-13 09:49:39.0	
4	테스트제목2	{테스트 내용2	user02	{2020-05-13 09:49:52.0	{2020-05-13 09:49:52.0	
5	수정된 제목	{수정된 내용	user00	{2020-05-13 09:49:57.0	{2020-05-13 14:08:23.0	
6	테스트제목4	{테스트 내용4	user04	{2020-05-13 09:50:02.0	{2020-05-13 09:50:02.0	
8	새로 작성하는 글	새로 작성하는 내용	newbie	{2020-05-13 11:35:55.0	{2020-05-13 11:35:55.0	
9	새로 작성하는 글	새로 작성하는 내용	newbie	{2020-05-13 12:51:19.0	{2020-05-13 12:51:19.0	
10	새로 작성하는 글	새로 작성하는 내용	newbie	{2020-05-13 12:51:19.0	{2020-05-13 12:51:19.0	
11	새로 작성하는 글	새로 작성하는 내용	newbie	{2020-05-13 12:52:01.0	{2020-05-13 12:52:01.0	
12	새로 작성하는 글	새로 작성하는 내용	newbie	{2020-05-13 12:52:01.0	{2020-05-13 12:52:01.0	
13	새로 작성하는 글	새로 작성하는 내용	newbie	{2020-05-13 14:05:52.0	{2020-05-13 14:05:52.0	
14	새로 작성하는 글	새로 작성하는 내용	newbie	{2020-05-13 14:05:52.0	{2020-05-13 14:05:52.0	
15	새로 작성하는 글	새로 작성하는 내용	newbie	{2020-05-13 14:08:23.0	{2020-05-13 14:08:23.0	
16	새로 작성하는 글	새로 작성하는 내용	newbie	{2020-05-13 14:08:23.0	{2020-05-13 14:08:23.0	
18	새로 작성하는 글	테스트 새로 작성하는 내용	테스트 newbie	test {2020-05-13 16:14:18.0	{2020-05-13 16:14:18.0	
19	새로 작성하는 글	테스트 새로 작성하는 내용	테스트 newbie	test {2020-05-13 16:17:11.0	{2020-05-13 16:17:11.0	
20	새로 작성하는 글	테스트 새로 작성하는 내용	테스트 newbie	test {2020-05-13 16:21:50.0	{2020-05-13 16:21:50.0	

등록 처리와 테스트

BoardController 에 POST 방식으로 처리되는 register() 추가

리턴 타입 : String

RedirectAttributes 파라미터 : 등록 작업이 끝난 후, 다시 목록화면으로 이동하기 위해 추가적으로 새롭게 등록된 게시물의 번호를 같이 전달하기 위해서 이용.

리턴 할 때에는 'redirect:' 라는 접두어를 사용. 스프링 MVC가 내부적으로 response.sendRedirect()를 처리해주기 때문에 편리.

```
/springboard/src/main/java/com/yks/controller/BoardController.java

@PostMapping("/register")
public String register(BoardVO board, RedirectAttributes rtrr) {

    log.info("register: " + board);

    service.register(board);

    rtrr.addFlashAttribute("result", board.getBno());

    return "redirect/board/list";
}
```

테스트 코드

MockMvcRequestBuilder 의 post() : POST 방식으로 데이터를 전달할 수 있다.

param() : 전달해야 하는 파라미터들을 지정할 수 있다.

마치 <input> 처럼

이와 같은 방식으로 코드를 작성하면 최초 작성 시에는 일이 많이 느껴지지만, 매번 입력할 필요가 없기 때문에 오류 발생 및 수정하는 경우, 반복적인 테스트가 수월함

```
/springboard/src/test/java/com/yks/controller/BoardControllerTests.java

@Test
public void testRegister() throws Exception {

    String resultPage = mockMvc
        .perform(MockMvcRequestBuilders
            .post("/board/register")
            .param("title", "테스트 새글 제목")
            .param("content", "테스트 새글 내용")
            .param("writer", "user00"))
        .andReturn().getModelAndView().getViewName();

    log.info(resultPage);
}
```

조회 처리와 테스트

@GetMapping : 특별한 경우가 아니라면, 조회는 GET 방식으로 처리.

@RequestParam : bno 값을 좀 더 명시적으로 처리하기 위함.

Model 파라미터 : 화면쪽으로 해당 게시번호의 게시물을 전달하기 위함.

```
/springboard/src/main/java/com/yks/controller/BoardController.java

@GetMapping("/get")
public void get(@RequestParam("bno") Long bno, Model model) {

    log.info("/get");
    model.addAttribute("board", service.get(bno));
}
```

```
/springboard/src/test/java/com/yks/controller/BoardControllerTests.java

@Test
public void testGet() throws Exception {

    log.info(mockMvc.perform(MockMvcRequestBuilders
        .get("/board/get")
        .param("bno", "2"))
        .andReturn().getModelAndView().getModelMap());
}
```

수정 처리와 테스트

변경된 내용을 수집해서 BoardVO 파라미터로 처리한 후, BoardService 를 호출,

수정 작업을 시작하는 화면의 경우 GET 방식으로 접근하지만, 실제 작업은 POST 방식으로 동작하므로 @PostMapping 을 이용하여 처리

service.modify() 는 수정 여부를 boolean 으로 처리하므로 성공한 경우에만 RedirectAttributes 에 추가.

```
/springboard/src/main/java/com/yks/controller/BoardController.java
```

```
@PostMapping("/modify")
public String modify(BoardVO board, RedirectAttributes rttr) {

    log.info("modify: " + board);

    if (service.modify(board)) {
        rttr.addFlashAttribute("result", "success");
    }
    return "redirect/board/list";
}
```

```
/springboard/src/test/java/com/yks/controller/BoardControllerTests.java
```

```
@Test
public void testModify() throws Exception {

    String resultPage = mockMvc.perform(MockMvcRequestBuilders
                                    .post("/board/modify")
                                    .param("bno", "1")
                                    .param("title", "수정된 테스트 새글 제목")
                                    .param("content", "수정된 테스트 새글 내용")
                                    .param("writer", "user00"))
                                .andReturn()
                                .getModelAndView()
                                .getViewName();

    log.info(resultPage);
}
```

```
INFO : jdbc.audit - 1. Connection.setAutoCommit() returned true
INFO : jdbc.audit - 1. PreparedStatement.new PreparedStatement returned
INFO : jdbc.audit - 1. Connection.prepareStatement(UPDATE tbl_board
      SET title = ?, content = ?, writer = ?, updateDate = sysdate
      WHERE bno = ?) returned net.sf.log4jdbc.sql.jdbcapi.PreparedStatementSpy@456be73c
INFO : jdbc.audit - 1. PreparedStatement.setString(1, "수정된 테스트 새글 제목") returned
INFO : jdbc.audit - 1. PreparedStatement.setString(2, "수정된 테스트 새글 내용") returned
INFO : jdbc.audit - 1. PreparedStatement.setString(3, "user00") returned
INFO : jdbc.audit - 1. PreparedStatement.setLong(4, 1) returned
INFO : jdbc.sqlonly - UPDATE tbl_board SET title = '수정된 테스트 새글 제목', content = '수정된 테스트 새글 내용', writer = 'user00',
updateDate = sysdate WHERE bno = 1
|
INFO : jdbc.sqltiming - UPDATE tbl_board SET title = '수정된 테스트 새글 제목', content = '수정된 테스트 새글 내용', writer = 'user00',
updateDate = sysdate WHERE bno = 1
{executed in 37 msec}
INFO : jdbc.audit - 1. PreparedStatement.execute() returned false
INFO : jdbc.audit - 1. PreparedStatement.getUpdateCount() returned 1
INFO : jdbc.audit - 1. PreparedStatement.isClosed() returned false
INFO : jdbc.audit - 1. PreparedStatement.close() returned
INFO : jdbc.audit - 1. Connection.clearWarnings() returned
```

삭제 처리와 테스트

삭제는 반드시 POST 방식으로만 처리

```

/springboard/src/main/java/com/yks/controller/BoardController.java

@PostMapping("/remove")
public String remove(@RequestParam("bno") Long bno, RedirectAttributes rttr) {

    log.info("remove..." + bno);
    if (service.remove(bno)) {
        rttr.addFlashAttribute("result", "success");
    }
    return "redirect/board/list";
}

```

```

/springboard/src/test/java/com/yks/controller/BoardControllerTests.java

@Test
public void testRemove() throws Exception {

    // 삭제 전 데이터베이스에 게시물 번호 확인할 것
    String resultPage = mockMvc.perform(MockMvcRequestBuilders
                                    .post("/board/remove")
                                    .param("bno", "13"))
        .andReturn().getModelAndView().getViewName();

    log.info(resultPage);
}

```

```

INFO : com.yks.controller.BoardController - remove...13
INFO : com.yks.service.BoardServiceImpl - remove...13
INFO : jdbc.audit - 1. Connection.getAutoCommit() returned true
INFO : jdbc.audit - 1. PreparedStatement.new PreparedStatement returned
INFO : jdbc.audit - 1. Connection.prepareStatement(DELETE FROM tbl_board WHERE bno = ?) returned net.sf.log4
INFO : jdbc.audit - 1. PreparedStatement.setLong(1, 13) returned
INFO : jdbc.sqlonly - DELETE FROM tbl_board WHERE bno = 13

INFO : jdbc.sqltiming - DELETE FROM tbl_board WHERE bno = 13
{executed in 1 msec}
INFO : jdbc.audit - 1. PreparedStatement.execute() returned false
INFO : jdbc.audit - 1. PreparedStatement.getUpdateCount() returned 1
INFO : jdbc.audit - 1. PreparedStatement.isClosed() returned false
INFO : jdbc.audit - 1. PreparedStatement.close() returned
INFO : jdbc.audit - 1. Connection.clearWarnings() returned
INFO : com.yks.controller.BoardControllerTests - redirect/board/list
INFO : org.springframework.web.context.support.GenericWebApplicationContext - Closing org.springframework.we
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Shutdown initiated...

```