

Chap10

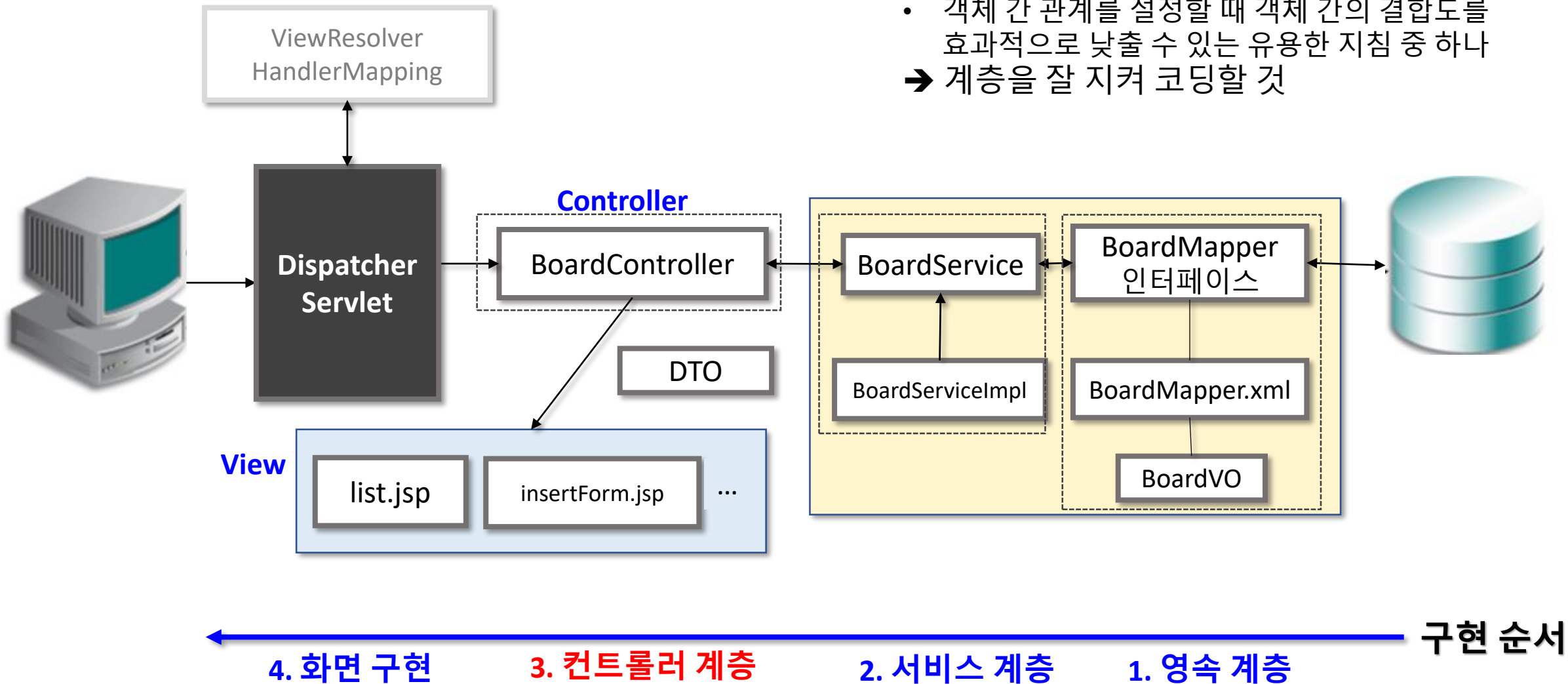
프레젠테이션(웹) 계층의 CRUD 구현

동의과학대학교

컴퓨터정보과

김진숙

구성도



[디미터 법칙(Law of Demeter)]

- 객체 구조의 경로를 따라 멀리 떨어져 있는 낯선 객체에 메시지를 보내는 설계는 피하라는 것
- 객체 간 관계를 설정할 때 객체 간의 결합도를 효과적으로 낮출 수 있는 유용한 지침 중 하나
➔ 계층을 잘 지켜 코딩할 것

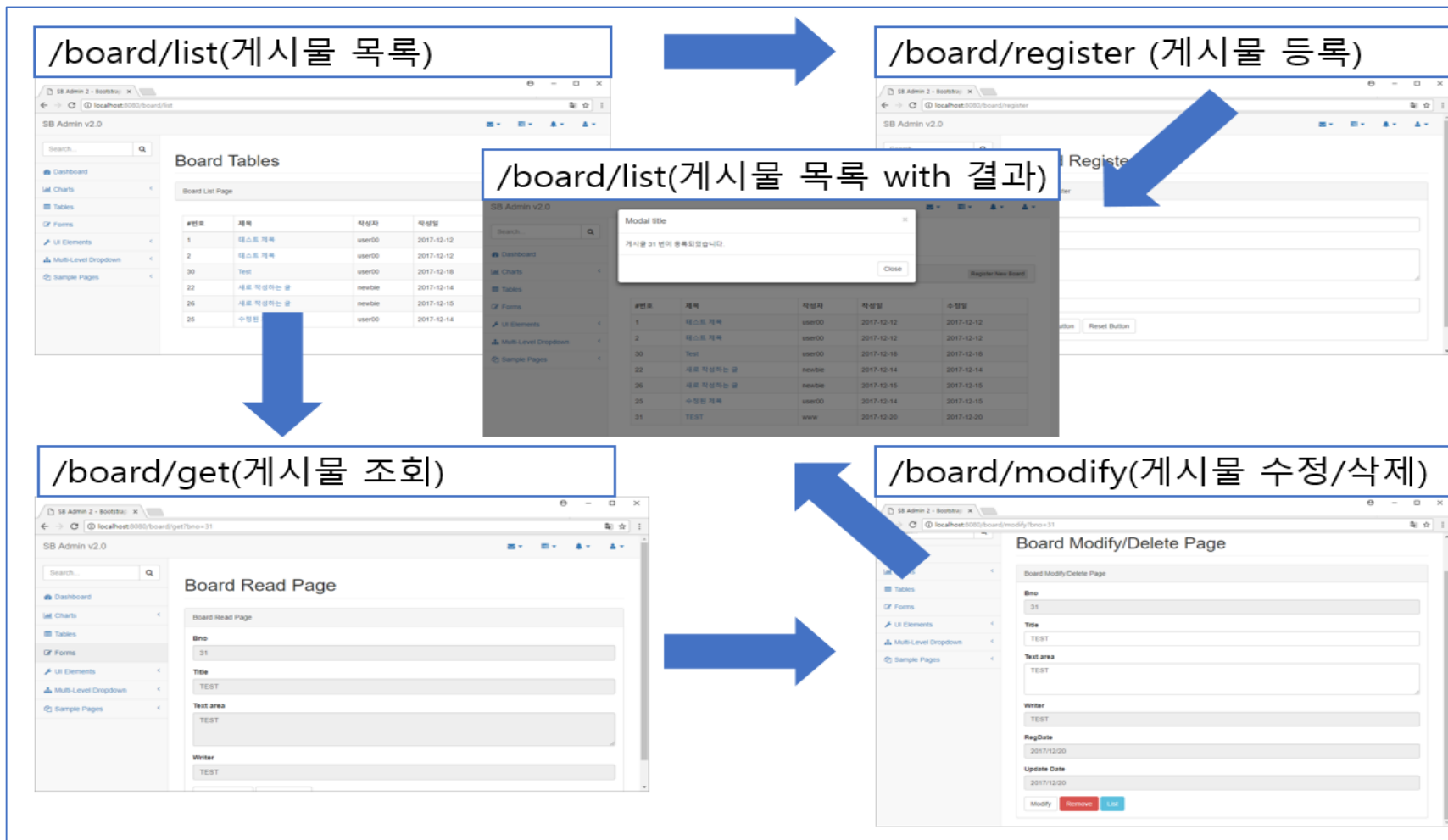
URI 설계

- 웹 계층 구현에서 가장 먼저 설계하는 것은 URI의 설계

해당 URI를 호출하기
위해서는 별도의
입력화면 필요

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

스토리보드

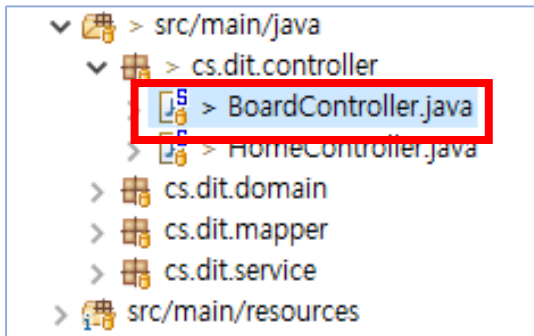


구현 작업의 순서

1. 전체 목록
 - 모든 진입 경로인 동시에 입력을 가는 링크
2. 등록 입력/처리
 - 게시물 등록 및 처리, 처리 후 이동
3. 조회
 - 목록 페이지에서 특정 게시물의 자세히 보기로 이동
4. 수정 처리
 - 조회 페이지에서 수정, 처리 후 이동
5. 삭제 처리
 - 조회 페이지에서 삭제, 처리 후 이동

BoardController.java

- Cs.dit.controller 패키지에 BoardController 생성하기
- @Controller 어노테이션으로 bean객체 생성 요청
- 웹 관련 내용임으로 servlet-context.xml 파일에서 해당 패키지를 component-scan 함



servlet-context.xml의 일부

```
<context:component-scan base-package="cs.dit.controller" />
```

@Controller

@Log4j

@RequestMapping("/board/*")

@RequiredArgsConstructor

```
public class BoardController {  
    private final BoardService service;
```

```
}
```

게시물 조회

- 게시물(BoardVO)의 목록을 이미 생성되어 있는 Model 객체에 담아서 전달
 - BoardController는 BoardService에 대해 의존적
 - 전달 당시에는 없으나 화면에서 추가적인 데이터가 필요할 때 Model 객체 사용
 - 의존성 주입
 - Final 주입
(@RequiredArgsConstructor)을 이용하여 자동으로 주입됨

```
@Controller
@Log4j
@RequestMapping("/board/*")
@RequiredArgsConstructor
public class BoardController {

    private final BoardService service;

    @GetMapping("/list")
    public void list(Model model) {
        Log.info("list");
        model.addAttribute("list", service.getList());
    }
}
```

Key value

게시물 조회 테스트

```
@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration //Test for Controller
@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 {

    @Autowired
    private WebApplicationContext ctx;

    private MockMvc mockMvc;
```

화면이 개발되지 않은 상태에서
구현내용을 테스트 할 수 있음

[웹어플리케이션 테스트 중 에러 시]

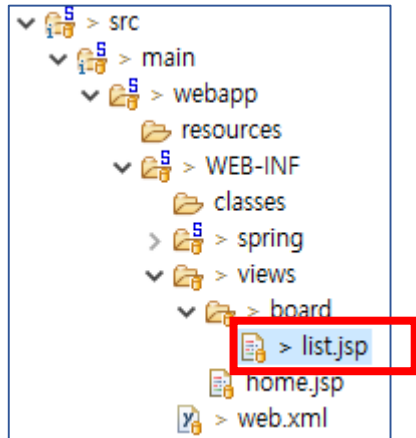
- 에러 : java.lang.NoClassDefFoundError:
javax/servlet/SessionCookieConfig
- 해결방안 : 서블릿 버전을 3.1 이상으로 변경할 것

테스트 환경과 실행환경의 서버가 다름

```
@Before
public void setup() {
    this.mockMvc = MockMvcBuilders.webAppContextSetup(ctx).build();
}
@Test
public void testList() throws Exception {
    Log.info(
        mockMvc.perform(
            MockMvcRequestBuilders.get("/board/list"))
            .andReturn()
            .getModelAndView()
            .getModelMap());
}
}
```


게시물 조회 테스트

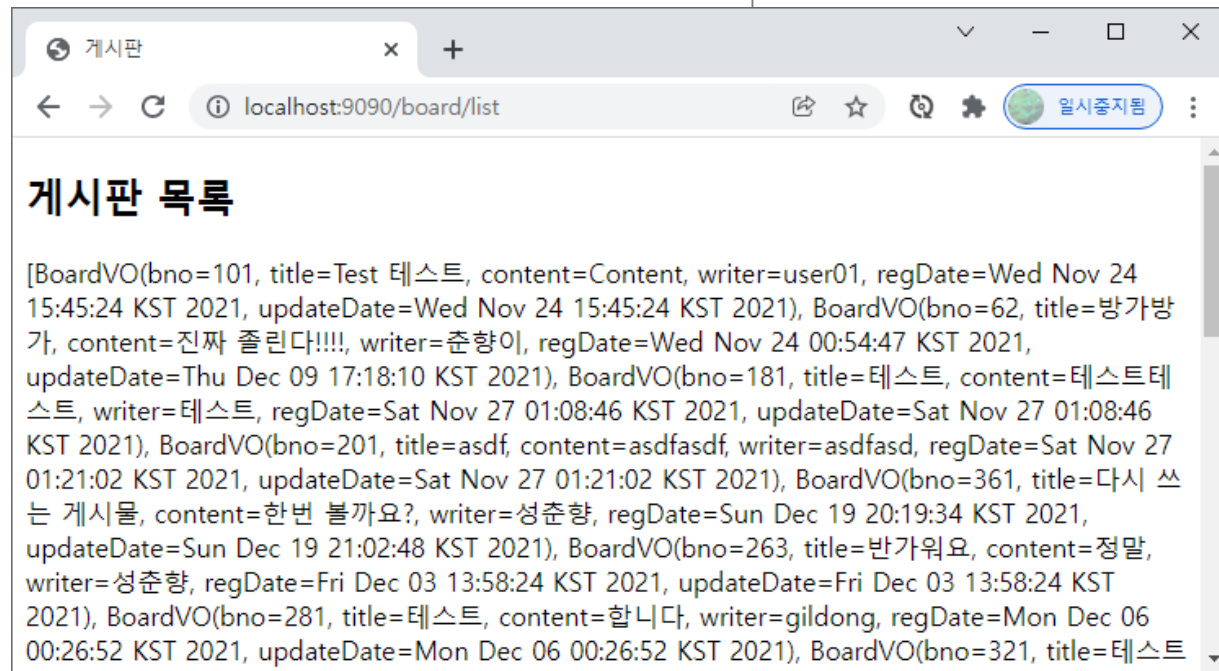
- JSP로 뷰 페이지를 작성하여 확인(list.jsp)
 - tomcat 실행 – chrome 실행
 - <http://localhost:8080/board/list>



```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
```

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">

    <title>게시판</title>
</head>
<body>
    <h2>게시판 목록</h2>
    ${list}
</body>
</html>
```



[참고] MockMvc 객체

- MockMvc 객체
 - MockMvc는 서블릿 컨테이너의 구동 없이, 시뮬레이션된 MVC 환경에 모의 HTTP 서블릿 요청을 전송하는 기능을 제공하는 유틸리티 클래스
 - 컨트롤러 테스트에 사용됨
- MockMvc 설정
 - 스프링 MVC의 설정을 적용한 DI 컨테이너를 만들고 사용해 스프링 MVC 동작을 재현함으로써 애플리케이션 서버에 배포한 것과 같은 것 처럼 테스트 가능

```
@Autowired
```

```
    private WebApplicationContext ctx;
```

```
@Before
```

```
    public void setup() {
```

```
        this.mockMvc = MockMvcBuilders.webAppContextSetup(ctx).build();
```

```
    }
```

[참고] MockMvc 객체

- 주요 메소드
 - perform() : 브라우저에서 서버에 URL 요청을 하듯 컨트롤러를 실행
 - perform() 메소드는 RequestBuilder 객체를 인자로 받고, 이는 MockMvcRequestBuilders의 정적 메소드를 이용해서 생성

- MockMvcRequestBuilders

- GET, POST, PUT, DELETE 요청 방식과 매핑되는 get(), post(), put(), delete() 메소드를 제공
- 메소드들은 MockHttpServletRequestBuilder 객체를 리턴하고, HTTP 요청 관련 정보 (파라미터, 헤더, 쿠키 등) 설정 가능

URI 설계

- 웹 계층 구현에서 가장 먼저 설계하는 것은 URI의 설계

해당 URI를 호출하기
위해서는 별도의
입력화면 필요

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

POST 방식 후 처리

- 등록/수정/삭제의 최종 처리는 POST 방식을 이용
- 브라우저에는 어떤 결과를 보여줄 것인가?
 1. 별도의 결과 페이지를 만들어서 보여주는 방식
 - 회원 가입 완료 메시지 출력 페이지 등
 2. 목록 페이지로 이동하는 방식(많이 사용하는 방식)
 - 목록 페이지에서 알림 페이지를 보여주는 방식
- POST 방식 후에는 'redirect:/ ' 를 고려할 것
 - 'redirect:/ ' 를 이용하여 브라우저와 연결을 한번에 종료
 - 브라우저는 새롭게 특정 URI를 요구

RedirectAttribute

- 시나리오
 - GET 방식으로 입력페이지 : /board/register
 - POST 방식으로 입력 처리 : /board/register
 - 처리가 끝난 후 화면 이동을 해도 브라우저의 URL은 POST 방식 처리URL 그대로
 - 만일 브라우저를 새로 고침 하면? => 잘못하면 계속 같은 게시물이 등록됨
- 'redirect:/...'을 이용하여 브라우저와 연결을 한 번 종료
 - 내부적으로 response.sendRedirect() 처리
 - 브라우저는 새롭게 특정 url을 요구
- 관련 속성
 - addAttribute : 브라우저의 링크와 연결되어 전송
 - addFlashAttribute : 한번만 전송되고 브라우저 주소창에는 남지 않음. 내부적으로 HttpSession에 저장했다가 삭제함

게시물 등록

- 게시물 등록 화면 출력
- 게시물 등록 처리

```
@PostMapping("/register")
public String register(BoardVO board, RedirectAttributes rttr) {
    Log.info("register : " + board);

    int count = service.register(board);

    if(count==1)
        rttr.addFlashAttribute("result", "registered");//한번만 사용하는 값을 보냄

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


게시물 등록 테스트

- MockMvcRequestBuilders의 post()
 - POST 방식으로 데이터 전달
 - param()을 이용하여 전달할 파라미터 지정 가능

```
@Test
public void testRegister() throws Exception{
    Log.info(mockMvc.perform(
        MockMvcRequestBuilders.post("/board/register")
            .param("title", "테스트")
            .param("content", "content")
            .param("writer", "작성자")
        )
        .andReturn()
        .getModelAndView()
        .getModelMap()
    );
}
```

```
INFO : cs.dit.controller.BoardController - register : BoardVO(bno=null, title=테스트,
content=content, writer=작성자, regDate=null, updateDate=null)
INFO : jdbc.audit - 1. Connection.isValid(5) returned true
INFO : jdbc.audit - 1. Connection.getAutoCommit() returned true
INFO : jdbc.audit - 1. PreparedStatement.new PreparedStatement returned
INFO : jdbc.audit - 1. Connection.prepareStatement(insert into tbl_board(title, content,
writer)
                                values(?, ?, ?)) returned
net.sf.log4jdbc.sql.jdbcapi.PreparedStatementSpy@383790cf
INFO : jdbc.audit - 1. PreparedStatement.setString(1, "테스트") returned
INFO : jdbc.audit - 1. PreparedStatement.setString(2, "content") returned
INFO : jdbc.audit - 1. PreparedStatement.setString(3, "작성자") returned
INFO : jdbc.sqlonly - insert into tbl_board(title, content, writer) values('테스트',
'content', '작성자')

INFO : jdbc.sqltiming - insert into tbl_board(title, content, writer) values('테스트',
'content', '작성자')
{executed in 4 msec}
```

게시글 조회(한 개)/테스트

@RequestParam("bno")
• bno 값을 좀 더 명시적으로
처리하기 위해 사용

```
@GetMapping("/get")
public void get(@RequestParam("bno") Long bno, Model model) {
    log.info("/get");
    model.addAttribute("board", service.get(bno));
}
```

```
@Test
public void testGet() throws Exception {
    log.info("get test-----");
    log.info(mockMvc.perform(MockMvcRequestBuilders.get("/board/get")
        .param("bno", "361")
        ).andReturn().getModelAndView().getModelMap()
    );
}
```

게시글 수정

```
@PostMapping("/modify")
public String modify(BoardVO board, RedirectAttributes rttr) {
    Log.info("modify : " + board);

    int count = service.modify(board);

    if(count==1)
        rttr.addFlashAttribute("result", "modified");

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

BoardController.java

게시글 삭제

```
@PostMapping("/remove")
public String remove(@RequestParam("bno") Long bno, RedirectAttributes rttr) {
    Log.info("remove : " + bno);

    int count = service.remove(bno);

    if(count==1)
        rttr.addFlashAttribute("result", "removed");

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

BoardController.java

게시글 수정/삭제 테스트

```
@Test
public void testModify() throws Exception {
    log.info("modify test-----");

    String resultPage = mockMvc.perform(MockMvcRequestBuilders.post("/board/modify")
        .param("bno", "341")
        .param("title", "다시 쓰는 게시물")
        .param("content", "한번 볼까요?")
        .param("writer", "성춘향")
        ).andReturn().getModelAndView().getViewName();
    log.info(resultPage);
}
```

BoardControllerTests.java

```
@Test
public void testRemove() throws Exception {
    log.info("remove test-----");

    String resultPage = mockMvc.perform(MockMvcRequestBuilders.post("/board/remove")
        .param("bno", "341")
        ).andReturn().getModelAndView().getViewName();
    log.info(resultPage);
}
```