제 10 강 프레임워크 개발하기

1. 프레임워크 개발

① MVC 프레임워크 구조

▢ 앞장에서 우리가 개발한 게시판 프로그램은 MVC 아키텍처를 적용하였다.
하지만 DispatcherServlet 클래스 하나로 Controller 기능을 구현하여 처리하였다.

▢ 이렇게 하나의 서블릿 클래스로 Controller를 구현하면 클라이언트의 모든 요청을 하나의 서블릿이 처리하게 된다. 따라서 수많은 분기처리 로직을 가질 수밖에 없고 이는 오히려 개발과 유지보수를 어렵게 만든다.

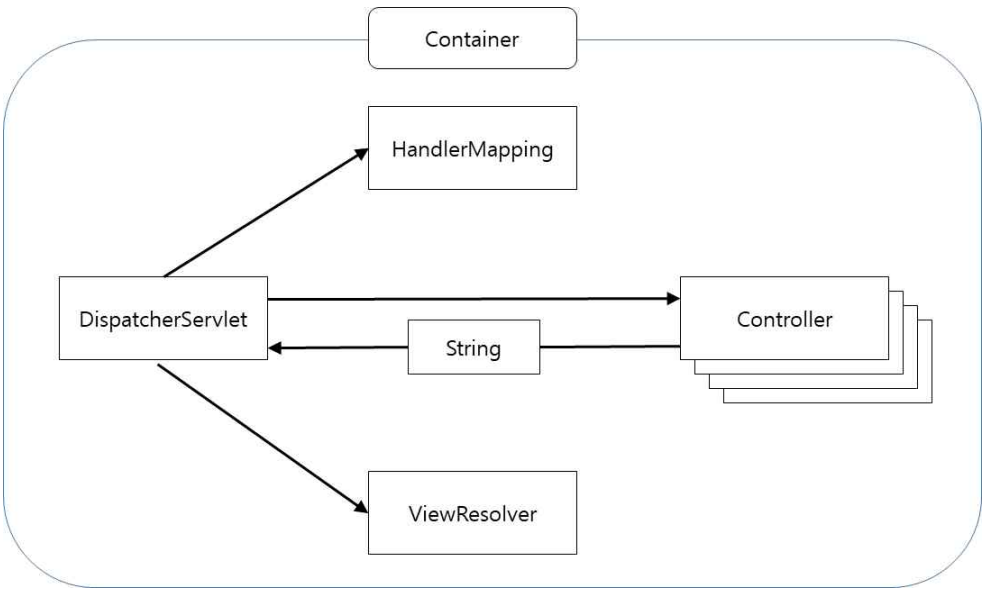
▢ 따라서 Controller 구현 시 다양한 디자인 패턴을 결합하여 개발과 유지보수의 편의성이 보장되도록 잘 만들어야 한다.

▢ 이러한 점 때문에 대부분의 개발 시 Struts나 Spring 같은 MVC 프레임워크를 사용하는 것이다. 이런 프레임워크들이 효율적인 Controller를 제공해 주기 때문이다.

▢ 결국 우리는 최종적으로 Spring 프레임워크를 적용할 것이다. Spring MVC는 DispatcherServlet을 시작으로 다양한 객체들이 상호작용하면서 클라이언트 요청을 처리한다. 하지만 MVC 프레임워크에서 제공하는 Controller의 기능과 구조가 복잡해서 바로 살펴보기 힘들다.

▢ 따라서 Spring MVC를 적용하기 전에 Spring MVC와 동일한 구조의 프레임워크를 직접 구현해 보고자 한다. 이 단계를 통해서 Spring MVC의 구조와 원리를 더욱 쉽게 이해할 수 있을 것이다.

② MVC 프레임워크의 구조



▢ 클라이언트 요청을 받은 DispatcherServlet은 HandlerMapping을 통해 Controller 객체를 검색하고 검색된 Controller를 실행한다.

③ MVC 프레임워크에 사용된 클래스의 기능

클래스	기능
DispatcherServlet	유일한 서블릿 클래스로서 모든 클라이언트의 요청을 가장 먼저 처리하는 Front Controller
HandlerMapping	클라이언트 요청을 처리할 Controller 매핑
Controller	실질적인 클라이언트의 요청을 처리
ViewResolver	Controller가 리턴한 View 이름으로 실행될 JSP 경로를 완성

2. 프레임워크 구현

① Controller 인터페이스 작성

▢ Controller를 구성하는 요소 중에서 DispatcherServlet은 클라이언트의 요청을 가장먼저 받아들이는 Front Controller이다. 하지만 클라이언트 요청을 처리하기 위해서 DispatcherServlet이 하는 일은 거의 없으며 실제적인 작업은 각 Controller에서 처리한다. 따라서 각 Controller를 같은 타입으로 관리하기 위하여 인터페이스를 작성해야 한다.

```
1  package tommy.spring.web.controller;
2  import javax.servlet.http.HttpServletRequest;
3  import javax.servlet.http.HttpServletResponse;
4  public interface Controller {
5          String handleRequest(HttpServletRequest request, HttpServletResponse response);
6  }
```

② LoginController 구현

: DispatcherServlet의 login.do에 해당하는 로직을 복사

```
1  package tommy.spring.web.user;
2  import javax.servlet.http.HttpServletRequest;
3  import javax.servlet.http.HttpServletResponse;
4  import tommy.spring.web.controller.Controller;
5  import tommy.spring.web.user.impl.UserDAO;
6  public class LoginController implements Controller {
7      @Override
8      public String handleRequest(HttpServletRequest request, HttpServletResponse response) {
9                  System.out.println("로그인 처리");
10                 // 1. 사용자 입력 정보 추출
11                 String id = request.getParameter("id");
12                 String password = request.getParameter("password");
13                 // 2. 데이터베이스 연동 처리
14                 UserVO vo = new UserVO();
15                 vo.setId(id);
16                 vo.setPassword(password);
17                 UserDAO userDAO = new UserDAO();
18                 UserVO user = userDAO.getUser(vo);
19                 // 3. 화면 네비게이션
20                 if (user != null) {
```

```
21                              return "getBoardList.do";
22                      } else {
23                              return "login";
24                      }
25      }
26  }
```

▢ 마지막에 이동할 화면을 리다이렉트 하지 않고 리턴하는 것으로 처리하였는데 화면정보가 login.jsp가 아니라 그냥 login이다. 이는 나중에 추가할 ViewResolver 클래스를 만들어야 정확하게 이해할 수 있다. 지금은 그냥 handleRequest() 메서드가 확장자 없는 문자열을 리턴하면 자동으로 ".jsp" 확장자가 붙어서 처리된다는 것으로 이해하고 넘어가자.

③ HandlerMapping 클래스 작성

▢ HandlerMapping은 모든 Controller 객체를 저장하고 있다가 클라이언트 요청이 들어오면 요청을 처리할 특정 Controller를 검색하는 기능을 제공한다.

▢ HandlerMapping은 DispatcherServlet이 사용하는 객체이다. DispatcherServlet이 생성되고 init() 메서드가 호출될 때 한 번 생성된다.

▢ HandlerMapping은 게시판 프로그램에 필요한 모든 Controller 객체를 등록하고 관리한다.

```
1   package tommy.spring.web.controller;
2   import java.util.HashMap;
3   import java.util.Map;
4   import tommy.spring.web.user.LoginController;
5   public class HandlerMapping {
6           private Map<String, Controller> mappings;
7           public HandlerMapping() {
8                   mappings = new HashMap<String, Controller>();
9                   mappings.put("/login.do", new LoginController());
10                  // 나중에 이 부분에 명령어(path)와 Controller 객체가 추가됨.
11          }
12          public Controller getController(String path) {
13                  return mappings.get(path);
14          }
15  }
```

④ ViewResolver 클래스 작성

▢ ViewResolver 클래스는 Controller가 리턴한 View 이름에 접두사(prefix)와 접미사(suffix)를 결합하여 최종적으로 실행될 View 경로와 파일명을 완성한다.

▢ DispatcherServlet이 생성되고 init() 메서드가 호출될 때 생성된다.

```
1   package tommy.spring.web.controller;
2   public class ViewResolver {
3           private String prefix;
4           private String suffix;
5           public void setPrefix(String prefix) {
6                   this.prefix = prefix;
7           }
```

```
8           public void setSuffix(String suffix) {
9                   this.suffix = suffix;
10          }
11          public String getView(String viewName) {
12                  return prefix + viewName + suffix;
13          }
14  }
```

⑤ DispatcherServlet 수정

```
1   package tommy.spring.web.controller;
2   import java.io.IOException;
3   import javax.servlet.ServletException;
4   import javax.servlet.annotation.WebServlet;
5   import javax.servlet.http.HttpServlet;
6   import javax.servlet.http.HttpServletRequest;
7   import javax.servlet.http.HttpServletResponse;
8   @WebServlet(name = "action", urlPatterns = { "*.do" })
9   public class DispatcherServlet extends HttpServlet {
10          private static final long serialVersionUID = 1L;
11          private HandlerMapping handlerMapping;
12          private ViewResolver viewResolver;
13          public void init() throws ServletException {
14                  handlerMapping = new HandlerMapping();
15                  viewResolver = new ViewResolver();
16                  viewResolver.setPrefix("./");
17                  viewResolver.setSuffix(".jsp");
18          }
19          protected void doGet(HttpServletRequest request, HttpServletResponse response)
                                                    throws ServletException, IOException {
20                  processRequest(request, response);
21          }
22          protected void doPost(HttpServletRequest request, HttpServletResponse response)
                                                    throws ServletException, IOException {
23                  request.setCharacterEncoding("UTF-8");
24                  processRequest(request, response);
25          }
26          private void processRequest(HttpServletRequest request, HttpServletResponse response)
                                                                        throws IOException {
                    // 1. 클라이언트 정보를 추출한다.
27                  String uri = request.getRequestURI();
28                  String path = uri.substring(uri.lastIndexOf("/"));
29                  System.out.println(path);
                    // 2. HandlerMapping을 통해 path에 해당하는 Controller를 검색한다.
30                  Controller controller = handlerMapping.getController(path);
                    // 3. 검색된 Controller를 실행한다.
31                  String viewName = controller.handleRequest(request, response);
                    // 4. ViewResolver를 통해 viewName에 해당하는 화면을 검색한다.
32                  String view = null;
```

```
33                      if (!viewName.contains(".do")) {
34                              view = viewResolver.getView(viewName);
35                      } else {
36                              view = viewName;
37                      }
                        // 5. 검색된 화면으로 이동한다.
38                      response.sendRedirect(view);
39              }
40      }
```

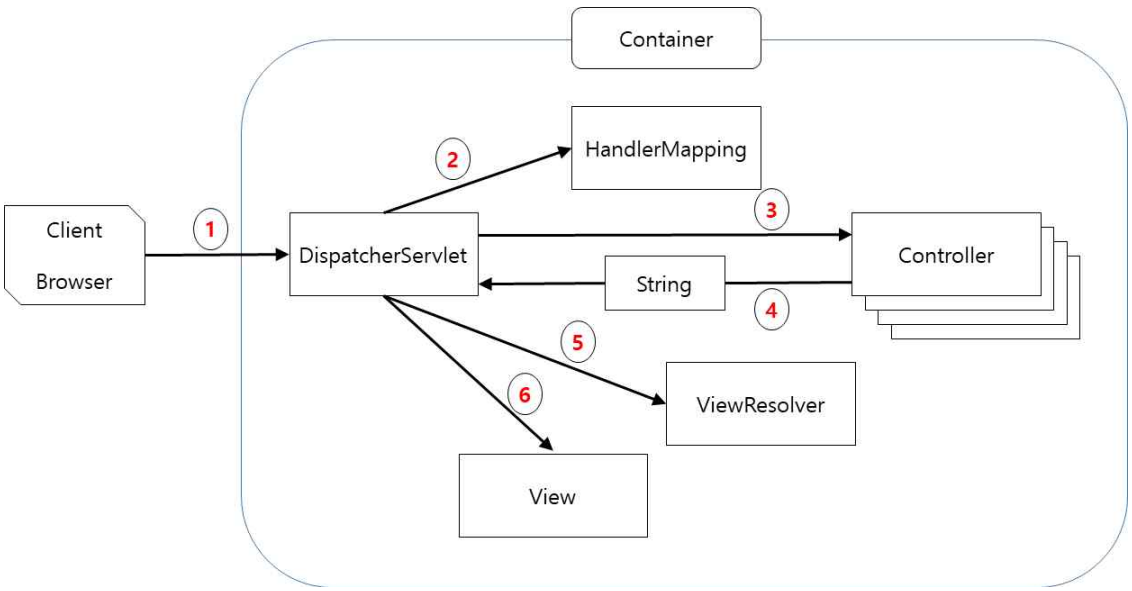
☐ 수정된 DispatcherServlet 클래스에는 init() 메서드가 재정의 되어있다. 서블릿의 init() 메서드는 서블릿 객체가 생성된 후 멤버변수를 초기화하기 위해 자동으로 실행된다.

☐ 따라서 init() 메서드에서 DispatcherServlet이 사용할 HandlerMapping와 ViewResolver 객체를 초기화한다. 그리고 DispatcherServlet은 이렇게 생성된 HandlerMapping와 ViewResolver를 이용하여 사용자의 요청을 처리한다.

⑥ 실행 및 결과확인

☐ http://localhost:8080/myboard/login.do 입력하여 정상적으로 실행되는지 확인해 보자.

3. 로그인 기능이 동작하는 과정



① 클라이언트가 로그인 버튼을 클릭하여 "/login.do" 요청을 전송하면 DispatcherServlet 요청을 받는다.

② DispatcherServlet은 HandlerMapping 객체를 통해 로그인 요청을 처리할 LoginController를 검색하고

③ 검색된 LoginController의 handleRequest() 메서드를 호출하면 로그인 로직이 처리된다.
④ 로그인 처리 후에 이동할 화면정보가 리턴되면
⑤ DispatcherServlet은 ViewResolver를 통해 접두사와 접미사가 붙은 JSP 파일의 이름과 경로를 리턴 받는다.
⑥ 최종적으로 JSP를 실행하고 실행결과가 브라우저에 응답된다.

4. MVC 프레임워크 적용

① 글 목록 검색구현

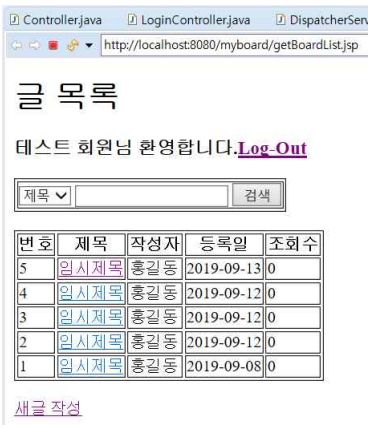
☐ GetBoardListController 구현 : DispatcherServlet의 글 목록 검색 부분 복사

```
1  package tommy.spring.web.board;
2  import java.util.List;
3  import javax.servlet.http.HttpServletRequest;
4  import javax.servlet.http.HttpServletResponse;
5  import javax.servlet.http.HttpSession;
6  import tommy.spring.web.controller.Controller;
7  import tommy.spring.web.board.impl.BoardDAO;
8  public class GetBoardListController implements Controller {
9      @Override
10     public String handleRequest(HttpServletRequest request, HttpServletResponse response) {
11                 System.out.println("글 목록 검색 처리");
12                 // 1. 사용자 입력 정보 추출 : 검색 기능은 나중에 구현
13                 // 2. 데이터베이스 연동 처리
14                 BoardVO vo = new BoardVO();
15                 BoardDAO boardDAO = new BoardDAO();
16                 List<BoardVO> boardList = boardDAO.getBoardList(vo);
17                 // 3. 응답 화면 구성
18                 HttpSession session = request.getSession();
19                 session.setAttribute("boardList", boardList);
20                 return "getBoardList";
21     }
22 }
```

☐ HandlerMapping에 GetBoardListController 등록

```
1  <!-- 상단 부분 생략 -->
2  public class HandlerMapping {
3          private Map<String, Controller> mappings;
4          public HandlerMapping() {
5                  mappings = new HashMap<String, Controller>();
6                  mappings.put("/login.do", new LoginController());
7                  mappings.put("/getBoardList.do", new GetBoardListController());
8          }
9  <!-- 하단 부분 생략 -->
```

☐ 실행 및 결과 확인 : http://localhost:8080/myboard/getBoardList.do



② 글 상세 보기 구현

☐ GetBoardController 구현 : DispatcherServlet의 글 상세 보기 부분 복사

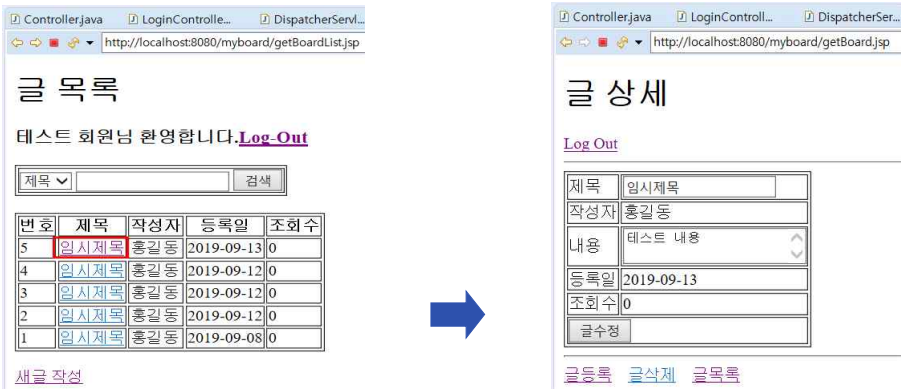
```
1  package tommy.spring.web.board;
2  import javax.servlet.http.HttpServletRequest;
3  import javax.servlet.http.HttpServletResponse;
4  import javax.servlet.http.HttpSession;
5  import tommy.spring.web.controller.Controller;
6  import tommy.spring.web.board.impl.BoardDAO;
7  public class GetBoardController implements Controller {
8      @Override
9      public String handleRequest(HttpServletRequest request, HttpServletResponse response) {
10                 System.out.println("글 상세 보기 처리");
11                 // 1. 검색할 게시글 번호 추출
12                 String seq = request.getParameter("seq");
13                 // 2. 데이터베이스 연동 처리
14                 BoardVO vo = new BoardVO();
15                 vo.setSeq(Integer.parseInt(seq));
16                 BoardDAO boardDAO = new BoardDAO();
17                 BoardVO board = boardDAO.getBoard(vo);
18                 // 3. 응답 화면 구현
19                 HttpSession session = request.getSession();
20                 session.setAttribute("board", board);
21                 return "getBoard";
22     }
23 }
```

☐ HandlerMapping에 GetBoardController 등록

```
1  <!-- 상단 부분 생략 -->
2  public class HandlerMapping {
3          private Map<String, Controller> mappings;
4          public HandlerMapping() {
5                  mappings = new HashMap<String, Controller>();
6                  mappings.put("/login.do", new LoginController());
7                  mappings.put("/getBoardList.do", new GetBoardListController());
8                  mappings.put("/getBoard.do", new GetBoardController());
```

```
9           }
10  <!-- 하단 부분 생략 -->
```

☐ 실행 및 결과 확인 : 글 목록 화면에서 제목을 클릭했을 때 상세화면이 나오는지 확인.



③ 글 등록 구현

☐ InsertBoardController 구현 : DispatcherServlet의 글 등록 부분 복사

```
1   package tommy.spring.web.board;
2   import javax.servlet.http.HttpServletRequest;
3   import javax.servlet.http.HttpServletResponse;
4   import tommy.spring.web.controller.Controller;
5   import tommy.spring.web.board.impl.BoardDAO;
6   public class InsertBoardController implements Controller {
7       @Override
8       public String handleRequest(HttpServletRequest request, HttpServletResponse response) {
9                   System.out.println("글 등록 처리");
10                  // 1. 사용자 입력 정보 추출
11                  // request.setCharacterEncoding("UTF-8");
12                  String title = request.getParameter("title");
13                  String writer = request.getParameter("writer");
14                  String content = request.getParameter("content");
15                  // 2. 데이터베이스 연동 처리
16                  BoardVO vo = new BoardVO();
17                  vo.setTitle(title);
18                  vo.setWriter(writer);
19                  vo.setContent(content);
20                  BoardDAO boardDAO = new BoardDAO();
21                  boardDAO.insertBoard(vo);
22                  // 3. 화면 네비게이션
23                  return "getBoardList.do";
24      }
25  }
```

☐ HandlerMapping에 InsertBoardController 등록

```
1   <!-- 상단 부분 생략 -->
2   public class HandlerMapping {
```

```
3           private Map<String, Controller> mappings;
4           public HandlerMapping() {
5                   mappings = new HashMap<String, Controller>();
6                   mappings.put("/login.do", new LoginController());
7                   mappings.put("/getBoardList.do", new GetBoardListController());
8                   mappings.put("/getBoard.do", new GetBoardController());
9                   mappings.put("/insertBoard.do", new InsertBoardController());
10          }
11  <!-- 하단 부분 생략 -->
```

▢ 실행 및 결과 확인 : 글 상세보기 화면에서 [글 등록]을 클릭하여 확인

④ 글 수정 구현

▢ UpdateBoardController 구현 : DispatcherServlet의 글 수정 부분 복사

```
1   package tommy.spring.web.board;
2   import javax.servlet.http.HttpServletRequest;
3   import javax.servlet.http.HttpServletResponse;
4   import tommy.spring.web.controller.Controller;
5   import tommy.spring.web.board.impl.BoardDAO;
6   public class UpdateBoardController implements Controller {
7       @Override
8       public String handleRequest(HttpServletRequest request, HttpServletResponse response) {
9                   // 1. 사용자 입력 정보 추출
10                  // request.setCharacterEncoding("UTF-8");
11                  String title = request.getParameter("title");
12                  String content = request.getParameter("content");
13                  String seq = request.getParameter("seq");
14                  // 2. 데이터베이스 연동 처리
15                  BoardVO vo = new BoardVO();
16                  vo.setTitle(title);
17                  vo.setContent(content);
18                  vo.setSeq(Integer.parseInt(seq));
19                  BoardDAO boardDAO = new BoardDAO();
20                  boardDAO.updateBoard(vo);
21                  // 3. 화면 네비게이션
22                  return "getBoardList.do";
```

```
23        }
24  }
```

▢ HandlerMapping에 UpdateBoardController 등록

```
1   <!-- 상단 부분 생략 -->
2   public class HandlerMapping {
3           private Map<String, Controller> mappings;
4           public HandlerMapping() {
5                   mappings = new HashMap<String, Controller>();
6                   mappings.put("/login.do", new LoginController());
7                   mappings.put("/getBoardList.do", new GetBoardListController());
8                   mappings.put("/getBoard.do", new GetBoardController());
9                   mappings.put("/insertBoard.do", new InsertBoardController());
10                  mappings.put("/updateBoard.do", new UpdateBoardController());
11          }
12  <!-- 하단 부분 생략 -->
```

▢ 실행 및 결과 확인 : 글 상세보기 화면에서 [글 수정]을 클릭하여 확인

⑤ 글 삭제 구현하기

▢ DeleteBoardController 구현 : DispatcherServlet의 글 삭제 부분 복사

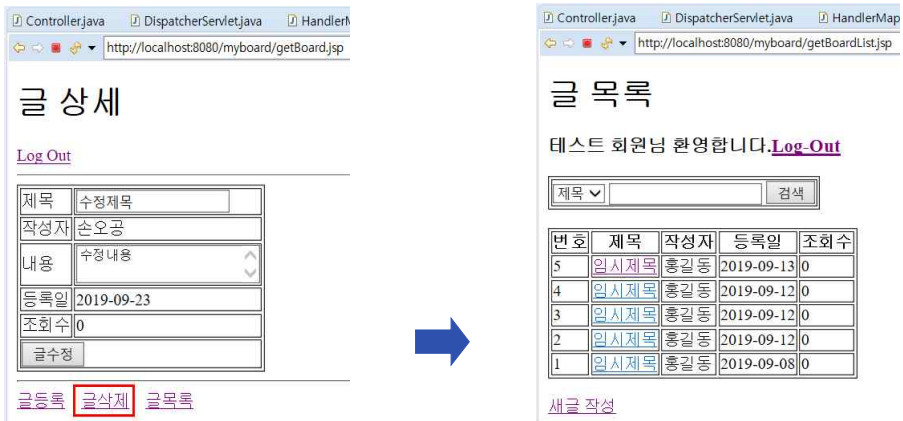
```
1   package tommy.spring.web.board;
2   import javax.servlet.http.HttpServletRequest;
3   import javax.servlet.http.HttpServletResponse;
4   import tommy.spring.web.controller.Controller;
5   import tommy.spring.web.board.impl.BoardDAO;
6   public class DeleteBoardController implements Controller {
7       @Override
8       public String handleRequest(HttpServletRequest request, HttpServletResponse response) {
9                   System.out.println("글 삭제 처리");
10                  // 1. 사용자 입력 정보 추출
11                  String seq = request.getParameter("seq");
12                  // 2. 데이터베이스 연동 처리
13                  BoardVO vo = new BoardVO();
14                  vo.setSeq(Integer.parseInt(seq));
```

```
15                      BoardDAO boardDAO = new BoardDAO();
16                      boardDAO.deleteBoard(vo);
17                      // 3. 화면 네비게이션
18                      return "getBoardList.do";
19          }
20  }
```

☐ HandlerMapping에 DeleteBoardController 등록

```
1   <!-- 상단 부분 생략 -->
2   public class HandlerMapping {
3           private Map<String, Controller> mappings;
4           public HandlerMapping() {
5                   mappings = new HashMap<String, Controller>();
6                   mappings.put("/login.do", new LoginController());
7                   mappings.put("/getBoardList.do", new GetBoardListController());
8                   mappings.put("/getBoard.do", new GetBoardController());
9                   mappings.put("/insertBoard.do", new InsertBoardController());
10                  mappings.put("/updateBoard.do", new UpdateBoardController());
11                  mappings.put("/deleteBoard.do", new DeleteBoardController());
12          }
13  <!-- 하단 부분 생략 -->
```

☐ 실행 및 결과 확인 : 글 상세보기 화면에서 [글 삭제]을 클릭하여 확인



⑥ 로그아웃 구현하기

☐ LogoutController 구현 : DispatcherServlet의 로그아웃 부분 복사

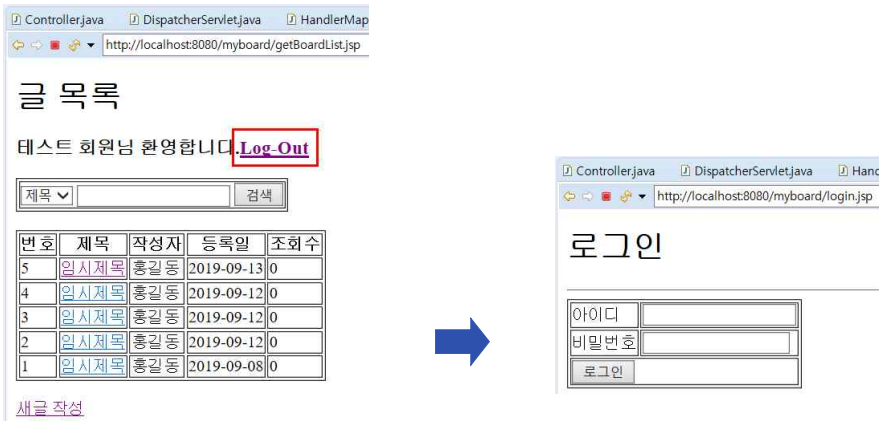
```
1   package tommy.spring.web.user;
2   import javax.servlet.http.HttpServletRequest;
3   import javax.servlet.http.HttpServletResponse;
4   import javax.servlet.http.HttpSession;
5   import tommy.spring.web.controller.Controller;
6   public class LogoutController implements Controller {
7           @Override
8           public String handleRequest(HttpServletRequest request, HttpServletResponse response) {
9                   System.out.println("로그아웃 처리");
```

```
10                      // 1. 브라우저와 연결된 세션 객체를 종료
11                      HttpSession session = request.getSession(false);
12                      session.invalidate();
13                      // 2. 세션 종료 후 메인 화면으로 이동
14                      return "login";
15          }
16  }
```

☐ HandlerMapping에 LogoutController 등록

```
1   <!-- 상단 부분 생략 -->
2   public class HandlerMapping {
3           private Map<String, Controller> mappings;
4           public HandlerMapping() {
5                   mappings = new HashMap<String, Controller>();
6                   mappings.put("/login.do", new LoginController());
7                   mappings.put("/getBoardList.do", new GetBoardListController());
8                   mappings.put("/getBoard.do", new GetBoardController());
9                   mappings.put("/insertBoard.do", new InsertBoardController());
10                  mappings.put("/updateBoard.do", new UpdateBoardController());
11                  mappings.put("/deleteBoard.do", new DeleteBoardController());
12                  mappings.put("/logout.do", new LogoutController());
13          }
14  <!-- 하단 부분 생략 -->
```

☐ 실행 및 결과 확인 : 글 목록 보기 화면에서 [로그아웃]을 클릭하여 확인



5. 결론

완성된 프로그램을 전체적으로 실행해보자. 수행이 잘 된다면 이전 MVC 2 패턴형태와 비교해 보자. MVC 구조보다 더 많은 클래스와 복잡한 구조를 가지게 되었을 것이다.

도대체 왜 이렇게 프로그램을 구성하는 것일까?

그것은 Controller에서 가장 중요한 DispatcherServlet 클래스는 유지보수 과정에서 기존의 기능을 수정하거나 새로운 기능을 추가한다 하여도 절대 수정되지 않는다.

예를 들어 회원가입 기능을 추가한다고 하면 InsertUserController 클래스를 새로 작성하고

HandlerMapping에 InsertUserController 클래스를 등록하면 된다.

이러한 과정에서 DispatcherServlet은 어떠한 수정작업도 생기지 않는다.

이렇게 기능 추가나 수정에 대해서 DispatcherServlet을 수정하지 않도록 해야 프레임워크에서 DispatcherServlet을 제공할 수 있는 것이다.

최종적으로 우리는 스프링 프레임워크를 사용할 것인데 결국 스프링 프레임워크에서 제공하는 DispatcherServlet 클래스를 사용하려면 새로운 기능이 추가되더라도 DispatcherServlet 클래스의 소스코드는 변경할 필요가 없도록 개발해야 한다.

6. EL/JSTL을 이용한 화면처리

- ▢ Model 1에서 Model 2 구조로 변경했던 결정적인 이유는 JSP 파일에서 Controller 로직에 해당하는 자바 소스코드를 제거하기 위해서이다. 그런데 아직 getBoard.jsp와 getBoardList.jsp 파일을 보면 여전히 자바 소스코드가 남아있다.
- ▢ 현재 남아있는 자바 코드는 사실 Controller의 로직이 아니다. Controller 로직은 사용자 입력 정보 추출, 데이터베이스 연동, 화면 내비게이션에 해당하는 로직이다.
- ▢ 만약 이러한 자바코드 조차 삭제하고 싶다면 EL/JSTL을 이용하면 된다.

① 상세화면 수정

▢ getBoard.jsp

```
1  <%@page import="tommy.spring.web.board.impl.BoardDAO"%>
2  <%@page import="tommy.spring.web.board.BoardVO"%>
3  <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
4  <!DOCTYPE html>
5  <html>
6  <head>
7  <meta charset="UTF-8">
8  <title>Board Article Content</title>
9  </head>
10 <body>
11 <h1>글 상세</h1>
12 <a href="logout.do">Log Out</a><hr>
13 <form action="updateBoard.do" method="post">
14 <input name="seq" type="hidden" value="${board.seq }" />
15 <table border="1">
16 <tr>
17          <td>제목</td>
18          <td><input name="title" type="text" value="${board.title }" /></td>
19 </tr>
20 <tr>
21          <td>작성자</td>
22          <td>${board.writer }</td>
23 </tr>
24 <tr>
25          <td>내용</td>
26          <td><textarea name="content">${board.content }</textarea></td>
27 </tr>
```

```
28  <tr>
29          <td>등록일</td>
30          <td>${board.regDate }</td>
31  </tr>
32  <tr>
33          <td>조회수</td>
34          <td>${board.cnt }</td>
35  </tr>
36  <tr>
37          <td colspan="2"><input type="submit" value="글수정" /></td>
38  </tr>
39  </table>
40  </form>
41  <hr>
42  <a href="insertBoard.jsp">글등록</a>   
43  <a href="deleteBoard.do?seq=${board.seq }">글삭제</a>   
44  <a href="getBoardList.do">글목록</a>
45  </body>
46  </html>
```

② 글 목록 화면 수정

☐ getBoardList.jsp

```
1   <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
2   <%@ page import="tommy.spring.web.board.BoardVO"%>
3   <%@ page import="java.util.List"%>
4   <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
5   <!DOCTYPE html>
    <!-- 중간 부분 생략 -->
            - - -
    <!-- 검색 종료 -->
6   <table border="1">
7   <tr>
8           <th>번호</th>
9           <th>제목</th>
10          <th>작성자</th>
11          <th>등록일</th>
12          <th>조회수</th>
13  </tr>
14  <c:forEach var="board" items="${boardList }">
15  <tr>
16          <td>${board.seq }</td>
17          <td><a href="getBoard.do?seq=${board.seq }">${board.title }</a></td>
18          <td>${board.writer }</td>
19          <td>${board.regDate }</td>
20          <td>${board.cnt }</td>
21  </tr>
22  </c:forEach>
23  </table><br/>
```

```
24  <a href="insertBoard.jsp">새글 작성</a>
25  </body>
26  </html>
```

▢ 이제 모든 작업이 완료되었다. 전체적으로 수행이 잘 되는지 확인해 보자.