스프링 실습 프로젝트 정리 실습

- 1주차 ~ 4주차 : //스프링을 위한 선수 지식 : Servlet 요점 정리
- SpringProject1: (5주차) //개발자에 의한 객체 결합 (교재 p39~48)
- SpringProject2: (6주차) //스프링컨테이너에 의한 객체 결합 (XML 방식) - IoC- (교재 p49~91)
- SpringProject3: (7주차) //스프링컨테이너에 의한 객체 결합 (어노테이션 방식) - IoC- (교재 p93~108)
- SpringProject4_BizStep1: (8주차) //비즈니스 컴포넌트 실습 1 (게시판) -Business Layer- (교재 p109~127)
- SpringProject4_BizStep2: (9주차) // 비즈니스 컴포넌트 실습 2 (User추가) -Business Layer- (교재 p129~139)
- SpringProject5_AOP_step1: (9주차) // 스프링 AOP 개념 파악 - AOP - (교재 p143~170)

9주 과제

- SpringProject5_AOP_step2: (10주차) // AOP 동작 시점 (5개) 실습- AOP - (교재 p171~182)
- SpringProject5_AOP_step3: (10주차) // AOP 동작 시점과 JointPoint/바인드변수 실습- AOP - (교재 p183~192)
- SpringProject5_AOP_step4: (10주차) // 어노테이션 방식의 AOP 실습- AOP - (교재 p193~192)

10주 과제

- SpringProject6_Jdbc_step1: (11주차) // Spring JDBC -스프링JDBC - (교재 p209~226)
- SpringProject6_Jdbc_step2: (11주차) // 트랜잭션 처리-트랜잭션 - (교재 p227~237)

11주 과제

스프링 실습 프로젝트 정리 실습

- SpringProject7_MVC_step1 : (12주차) // MVC 패턴 - **Model 1 아키텍처** – (교재 p241~261)
- SpringProject7_MVC_step2 : (12주차) // MVC 패턴 - **Model 2 아키텍처** – (교재 p263~283)
- SpringProject7_MVC_step3 : (12주차) // MVC 패턴 - **MVC 프레임워크 아키텍처** – (교재 p285~311)
- SpringProject7_MVC_step4 : (12주차) // 스프링 MVC - **Spring MVC 구조 (xml)** – (교재 p313~345)
- SpringProject7_MVC_step5_1: (12주차) // 스프링 MVC - **Spring MVC 구조(어노테이션 1)**– (교재 p349~370)
- SpringProject7_MVC_step5_2: (12주차) // 스프링 MVC - **Spring MVC 구조(어노테이션 2)**– (교재 p371~394)

Spring Layered architecture 구조 – MVC 패턴 p241

1) Presentation Layer

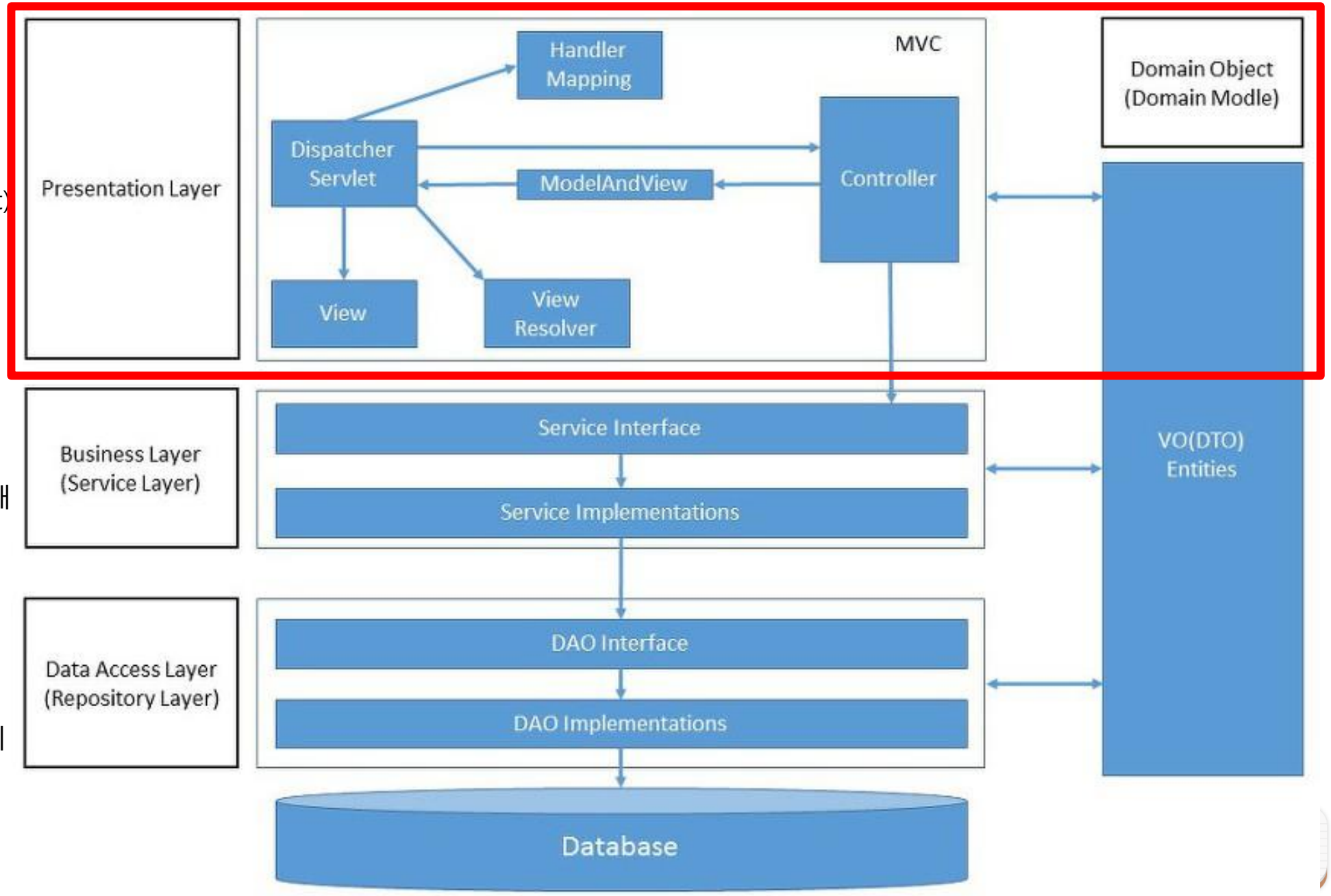
Spring MVC 객체를 말한다.
프론트 컨트롤러(DispatcherServlet)
컨트롤러, 뷰, 모델이 포함

2) Service Layer(Business Layer)

presentation layer에서 요청을 보내면 실제로 비지니스로직 수행

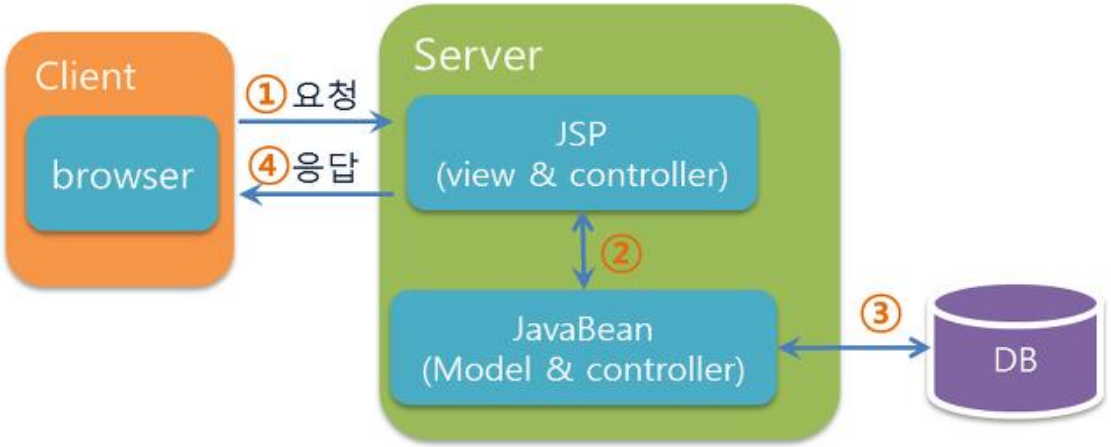
3) Data Access Layer(Repository Layer)

DB에 값을 저장하거나 가져오기 위해 JDBC, Mybatis, JPA 등을 사용해 구현한 DAO

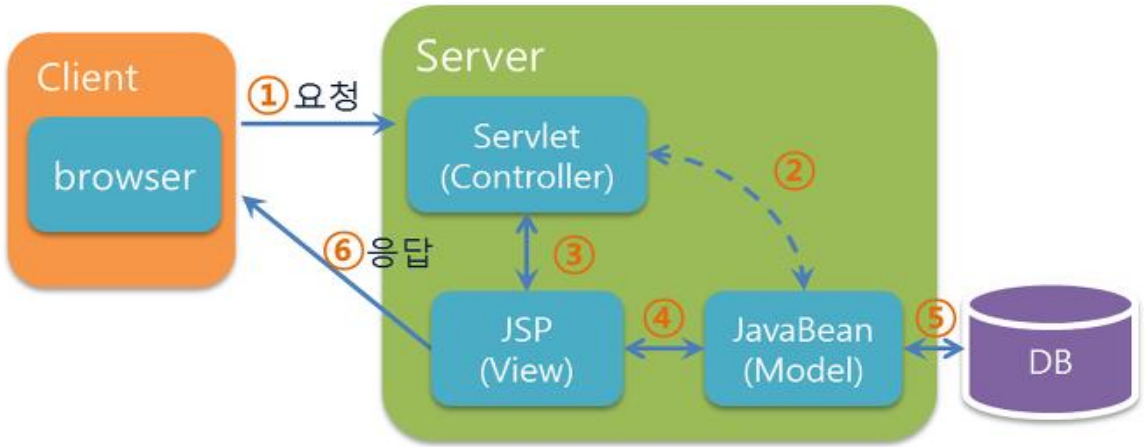


MVC 패턴 : 모델 1과 모델 2의 비교 p241

- **MVC 패턴 :**
  MVC패턴의 최대 장점은 사용자에게 보여지는 프레젠테이션 영역과 비즈니스 로직, 데이터 구조가 서로 완전히 분리되어 있다는 점



<MVC – Model 1>



<MVC – Model 2>

모델1: 비즈니스 로직 영역(Controller)에 프레젠테이션 영역(View)을 같이 구현하는 방식

모델2: 비즈니스 로직 영역과 프레젠테이션 영역이 분리되어 있는 구현 방식

	모델1	모델2
컨트롤러와 뷰의 분리 여부	통합(JSP파일)	분리(JSP, Servlet)
장점	쉽고 빠른 개발	디자이너/개발자 분업 유리 유지보수에 유리
단점	유지보수가 어려움	설계가 어려움 개발 난이도가 높음

MVC 패턴 : Model 1으로 게시판 p241

SpringProject7_MVC_step1 프로젝트

- **모델 1 :** 적은 개발인력으로 간단한 프로젝트 수행시 사용
  - JSP
    - View 역할 : HTML, CSS
    - Controller 역할 : 사용자의 요청 처리와 관련된 자바 코드 의미 (jsp내의 모든 자바코드가 controller는 아님)

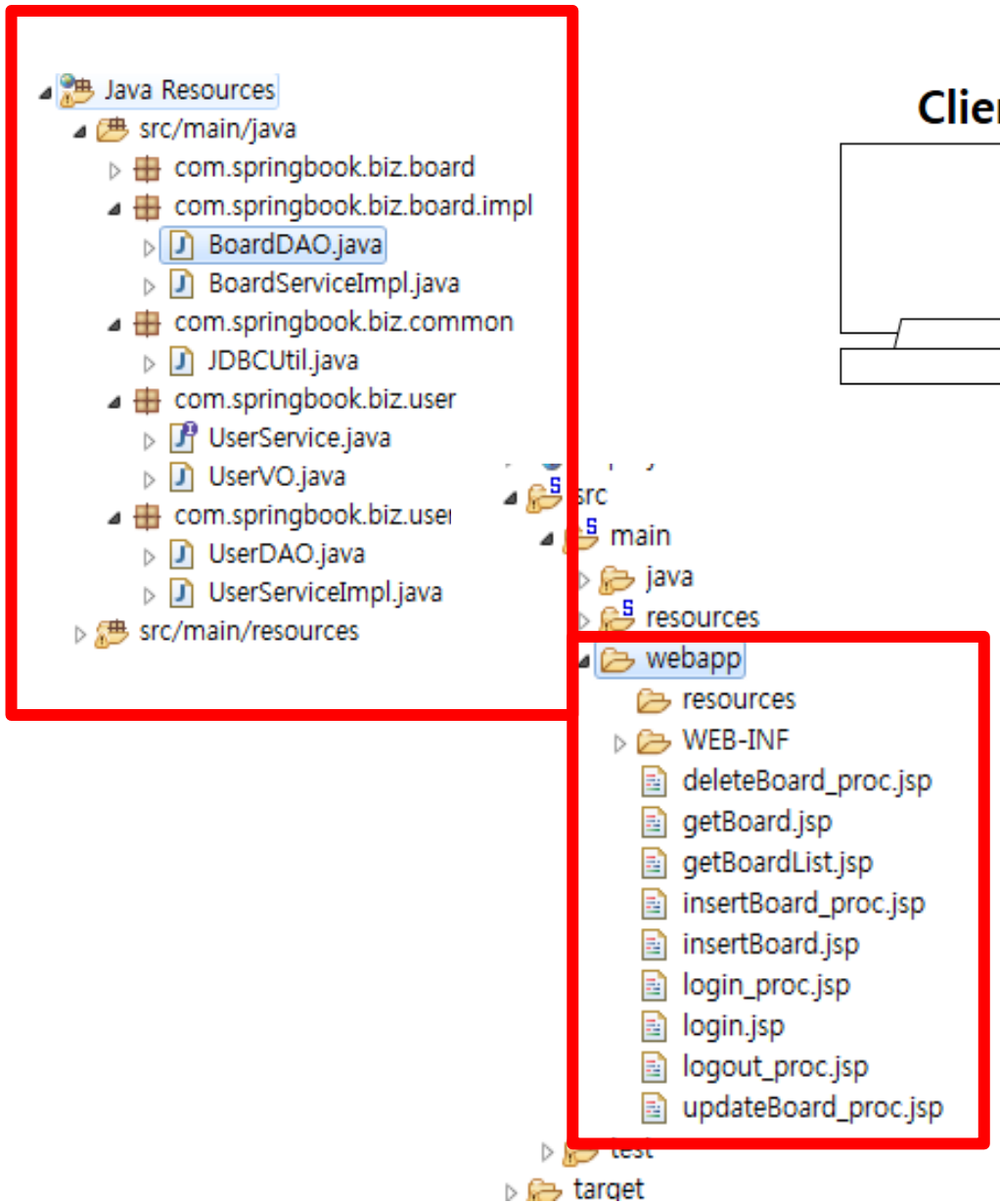
기 능	사 용 예
사용자 입력 정보 추출	String id = request.getParameter("userId")
DB 연동 처리	UserVO vo = new UserVO(); vo.setId(id); UserDAO userDAO = new UserDAO(); UserVO user = userDAO.getUser(vo);

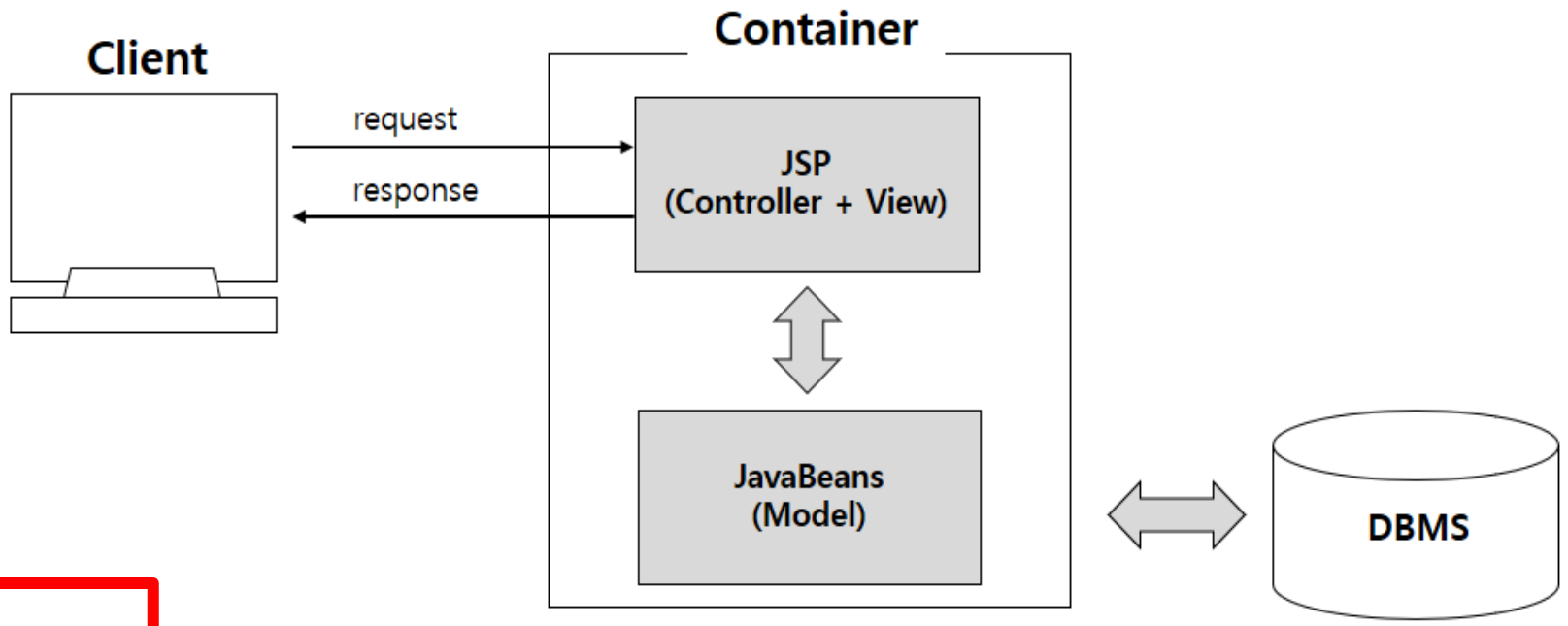
화면 내비게이션	if(user != null) { // 로그인 성공
	response.sendRedirect("getBoardList.jsp");
	} else { // 로그인 실패
	response.sendRedirect("login.jsp");
	}

  - JavaBean
    - Model 역할 : DB연동 로직을 제공하며, DB에서 검색한 데이터가 저장되는 자바 객체. VO와 DTO

MVC 패턴 : Model 1으로 게시판 p241

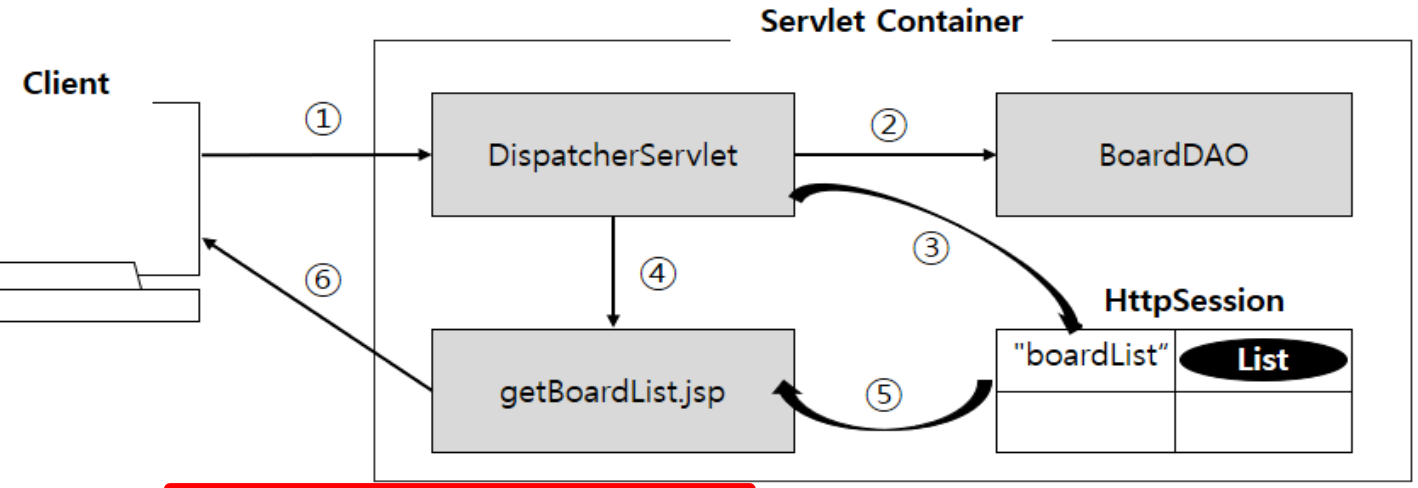
SpringProject7_MVC_step1 프로젝트



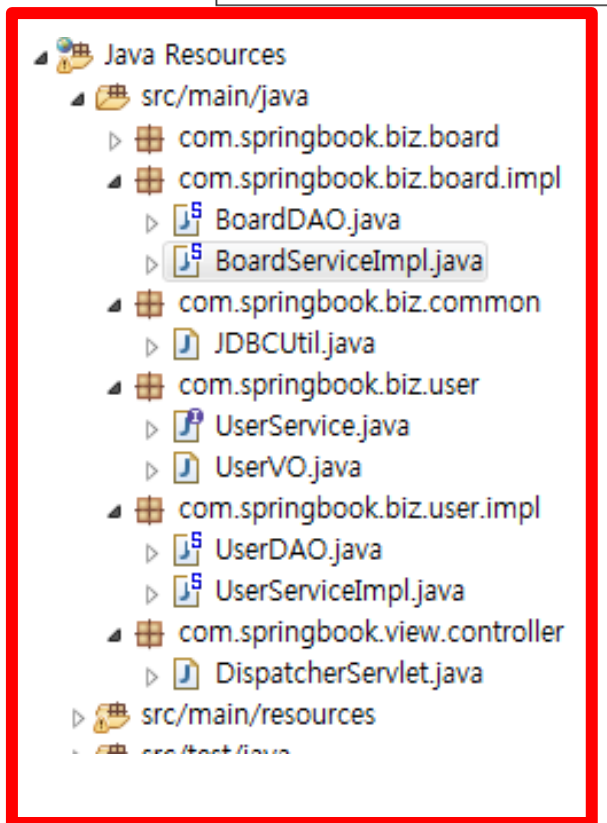


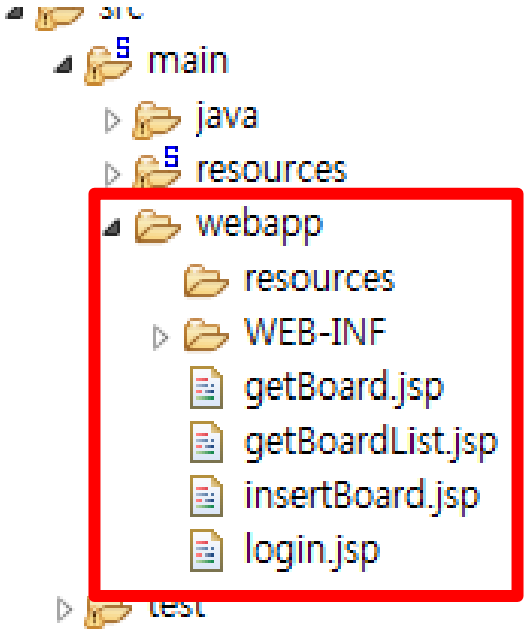
MVC 패턴 : Model 2으로 게시판 p263

SpringProject7_MVC_step2 프로젝트



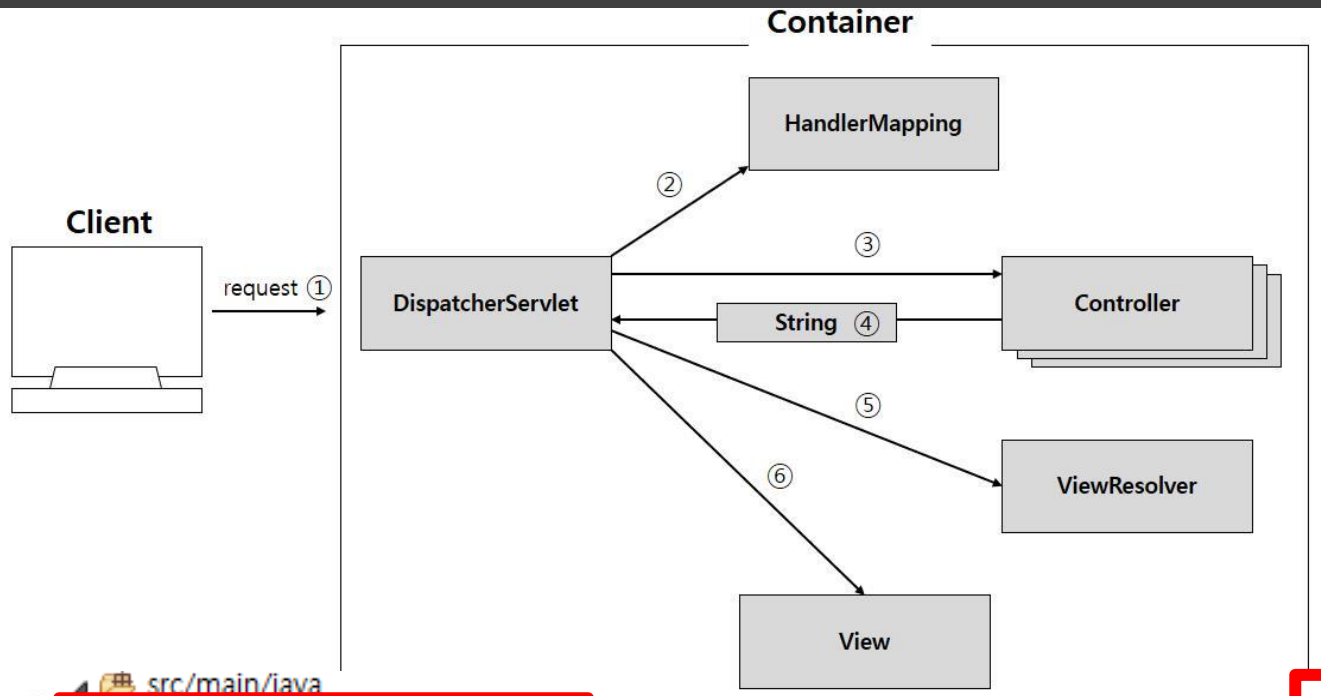
기 능	구성 요소	개발 주체
Model	VO, DAO 클래스	자바 개발자
View	JSP 페이지	웹 디자이너
Controller	Servlet 클래스	자바 개발자 또는 MVC 프레임워크



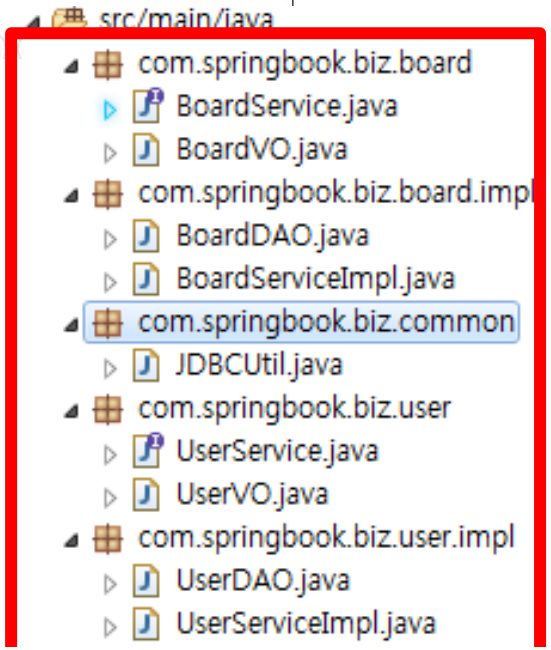


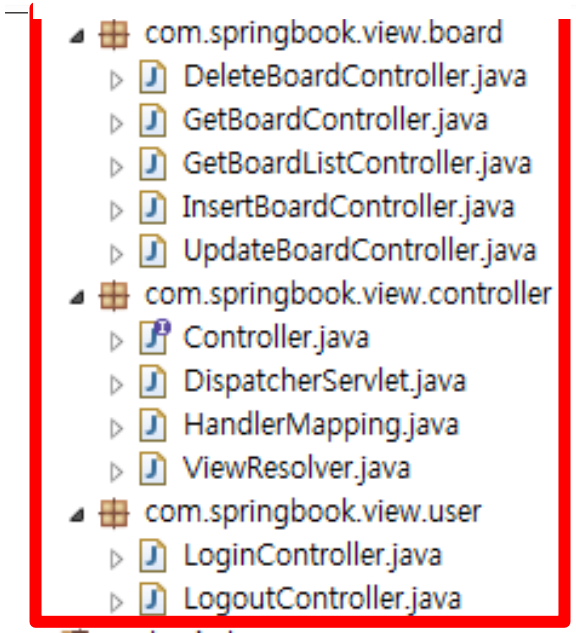
MVC 패턴 : MVC프레임워크로 게시판 p285

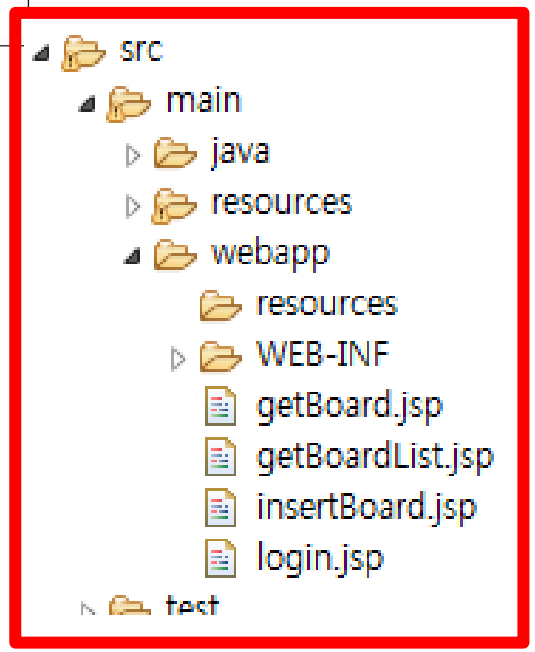
SpringProject7_MVC_step3 프로젝트



클 래 스	기 능
DispatcherServlet	유일한 서블릿 클래스로서 모든 클라이언트의 요청을 가장 먼저 처리하는 Front Controller
HandlerMapping	클라이언트의 요청을 처리할 Controller 매핑
Controller	실질적인 클라이언트의 요청 처리
ViewResolver	Controller가 반환한 View 이름으로 실행될 JSP 경로 완성

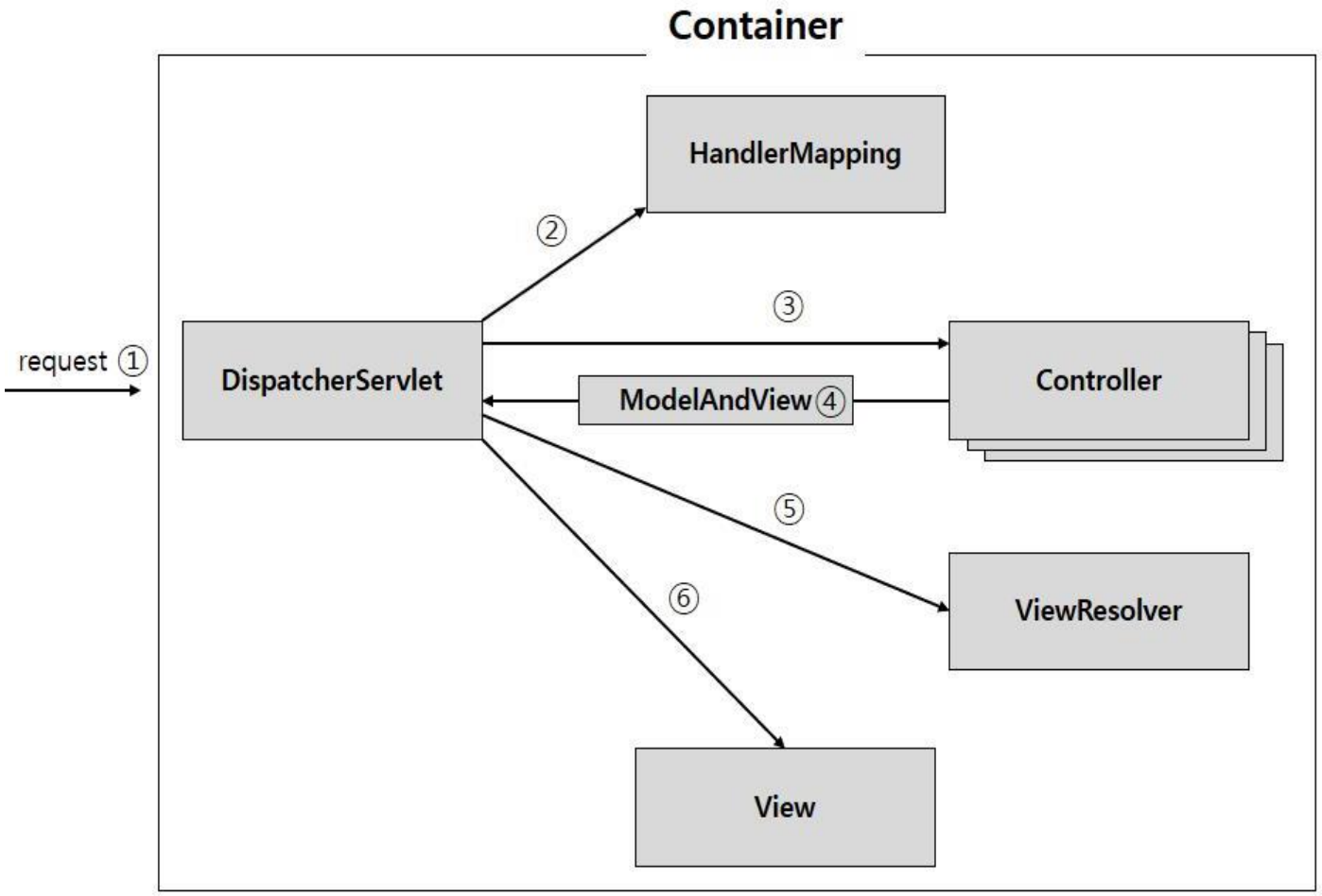


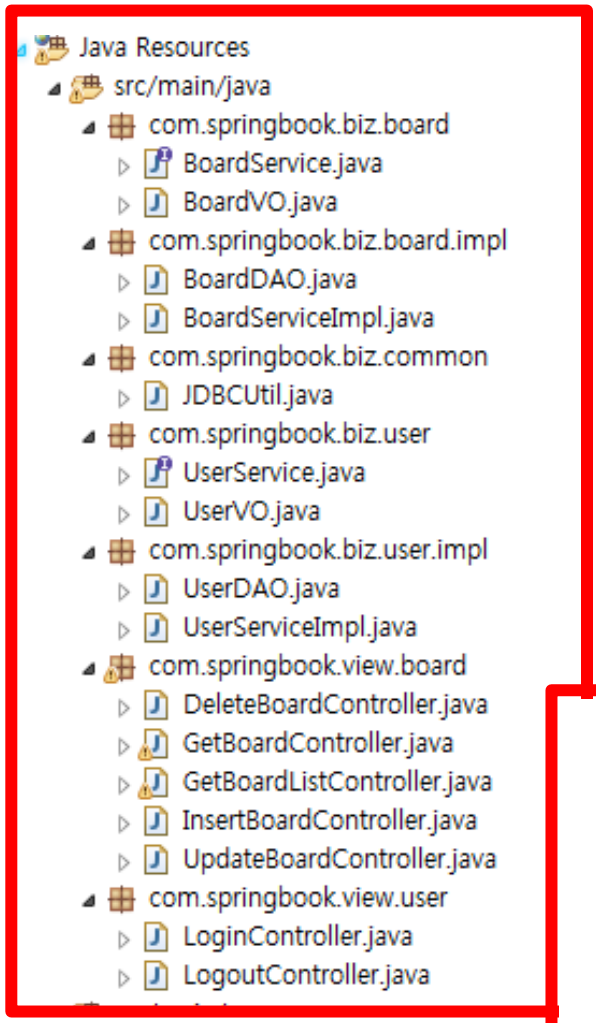


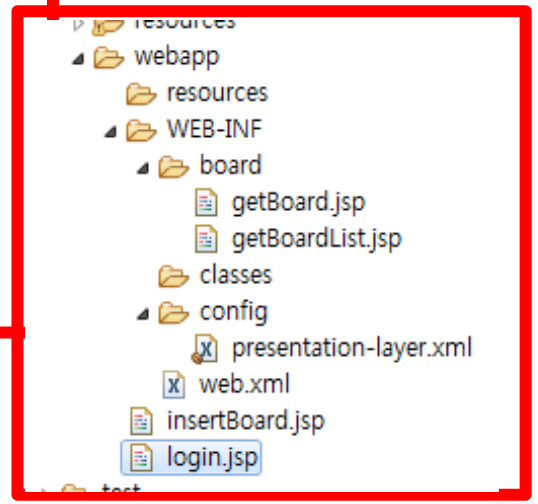


Spring MVC(xml)로 적용 p325

SpringProject7_MVC_step4 프로젝트

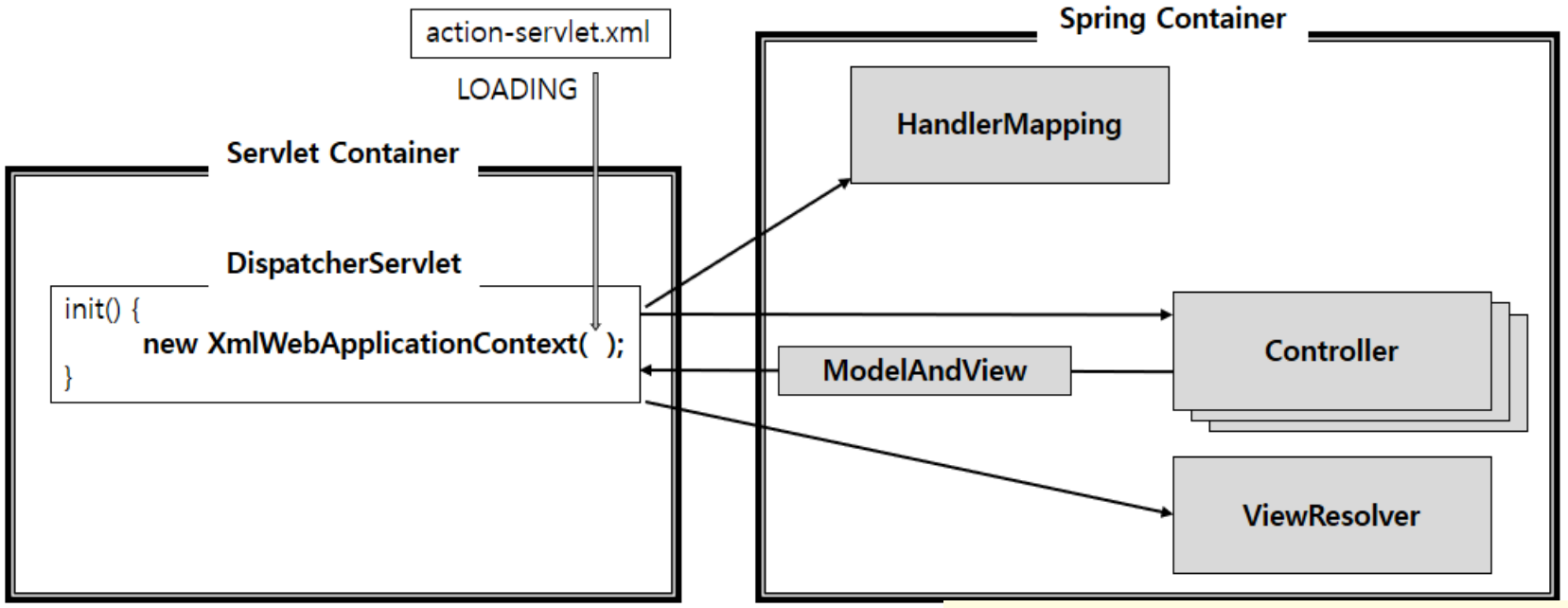






Spring MVC(xml)로 적용 p325

SpringProject7_MVC_step4 프로젝트



DispatcherServlet 등록 (web.xml)

```
<servlet>
        <servlet-name>action</servlet-name>
        <servlet-class>
                org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <init-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>/WEB-INF/config/presentation-layer.xml</param-value>
        </init-param>
</servlet>
 <servlet-mapping>
        <servlet-name>action</servlet-name>
        <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

Spring 설정파일 (presentation-layer.xml)

```
<!-- HandlerMapping 등록 -->
<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
        <property name="mappings">
                <props>
                        <prop key="/login.do">login</prop>
                </props>
        </property>
</bean>

<!-- Controller 등록 -->
<bean id="login" class="com.springbook.view.user.LoginController"></bean>
```

```
<!-- ViewResolver 등록 -->
<bean id="viewResolver"
         class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/board/"></property>
    <property name="suffix" value=".jsp"></property>
</bean>
```

Controller 클래스 구현

```
public class LoginController implements Controller {

	@Override
	public ModelAndView handleRequest(HttpServletRequest request,
					HttpServletResponse response) {
		// 1. 사
		// 2. DB 연 혜동
		// 3. 화 면
		ModelAndView mav = new ModelAndView();
		if(user != null) {
			mav.setViewName("redirect:getBoardList.do");
		 } else {
			mav.setViewName("login");
		 }
		return mav;
	}
}
```

xxx.do로 이동할 때는 ViewResolver를 적용하기 않기 위해 “redirect:”을 View 이름 앞에 붙인다.

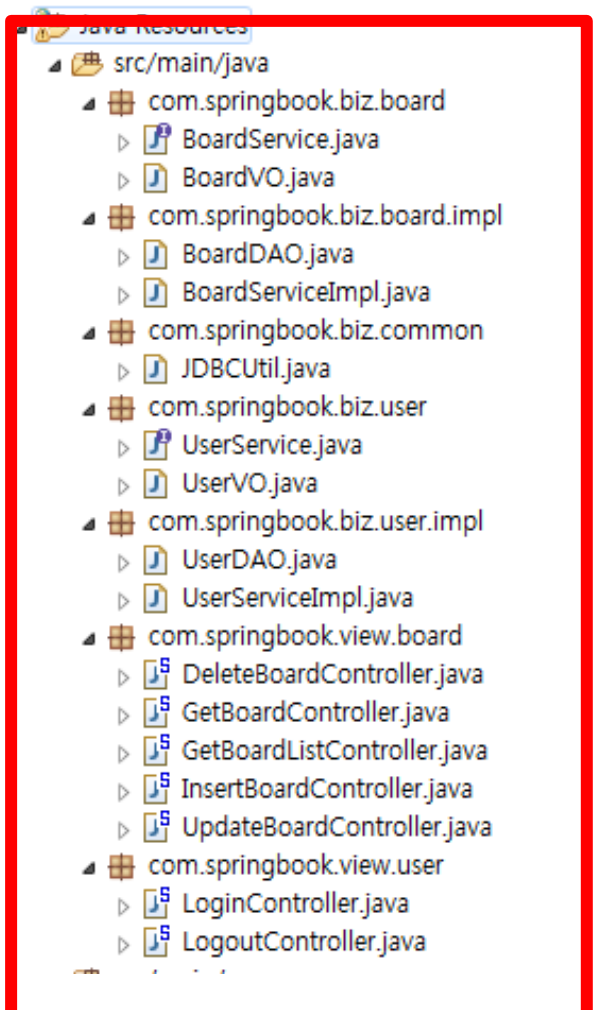
xxx.jsp로 이동할 때는 확장자 “.jsp”를 제거한다.

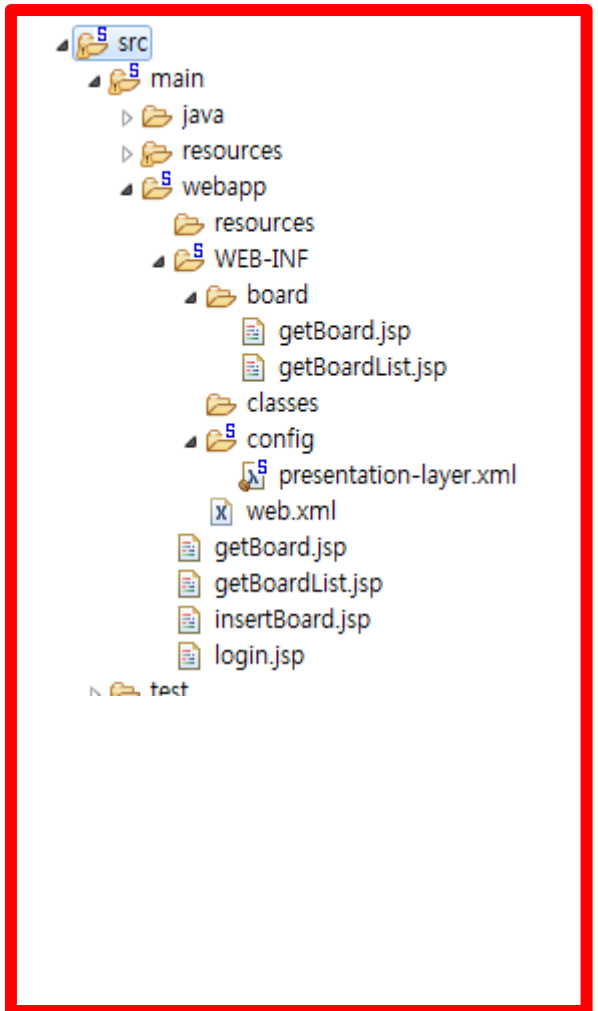
게시판 글 목록

```
public class GetBoardListController implements Controller {

	@Override
	public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) {

		// 1. DB 연 혜동
		BoardVO vo = new BoardVO();
		BoardDAO boardDAO = new BoardDAO();
		List<BoardVO> boardList = boardDAO.getBoardList(vo);

		// 2. DB 쿼리 결과와 화면 정 를뵈ModelAndView에 저장하여 리턴
		ModelAndView mav = new ModelAndView();
		mav.addObject("boardList", boardList);   // Model 정 쟁보
		mav.setViewName("getBoardList.jsp");   // View 정 쟁보
		return mav;
	}

}
```

Annotation 설정을 위한 준비 (presentation-layer.xml)

```
<context:component-scan base-package="com.springbook.view“/>
```





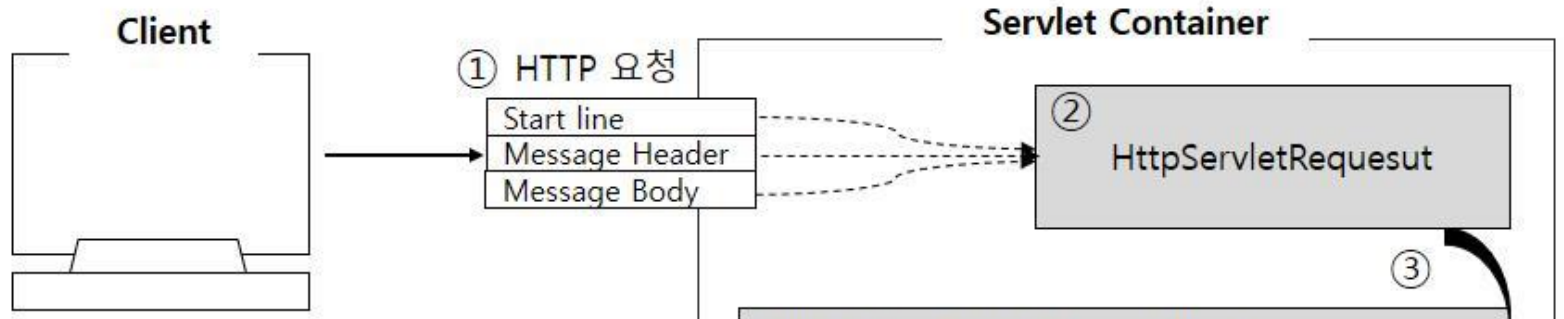
- **@Controller**
  - @Controller가 붙은 클래스를 메모리에 생성하고 Controller 객체로 인식하도록 한다.
  - Controller를 POJO(Plain Old Java Object) 스타일로 코딩할 수 있다.

```
@Controller
public class InsertBoardController {

}
```

- **@RequestMapping**
  - 클라이언트의 요청 path에 대해 실행될 메소드를 매핑한다.

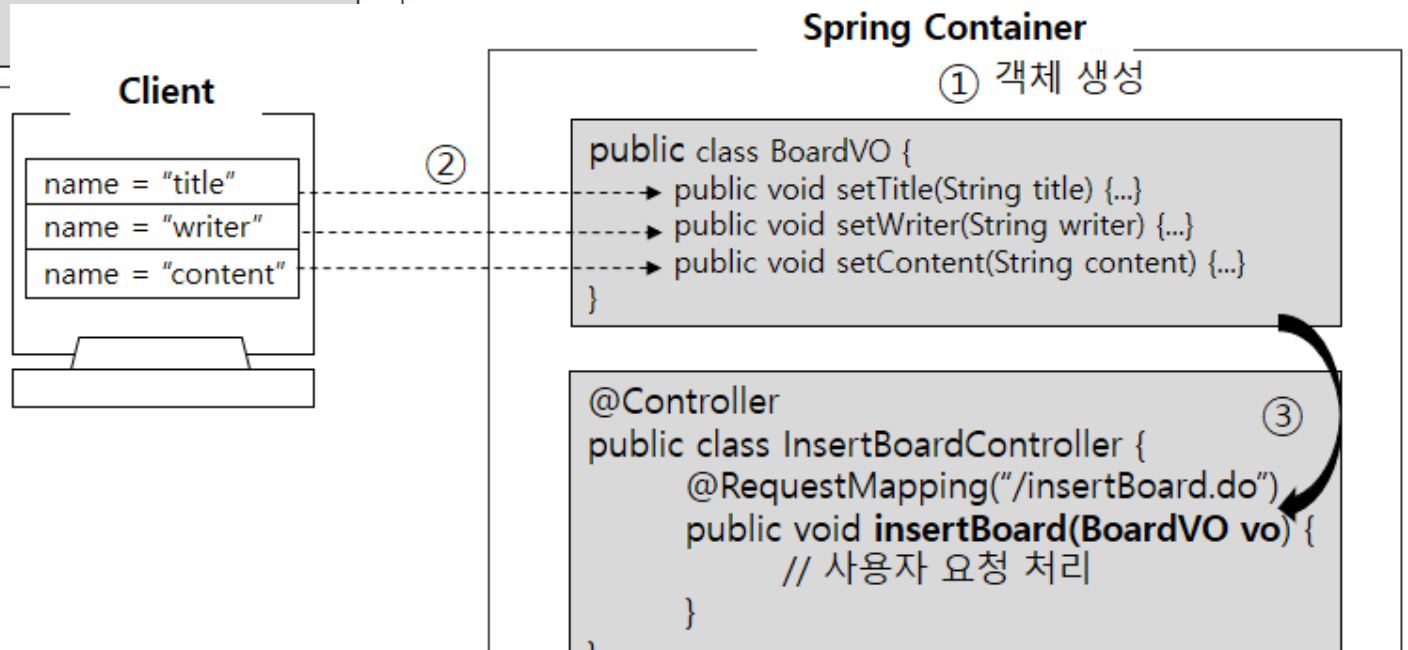
```
@Controller
public class InsertBoardController {
        @RequestMapping(value="/insertBoard.do")
        public void insertBoard(HttpServletRequest request) {
        }
}
```

클라이언트의 요청 처리를 위한 입력정보 추출

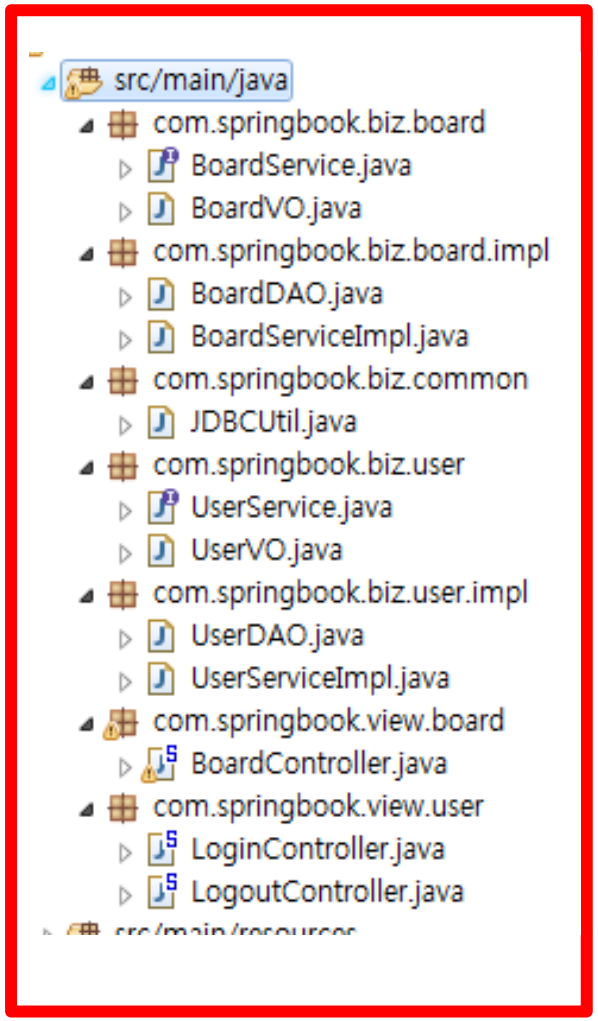


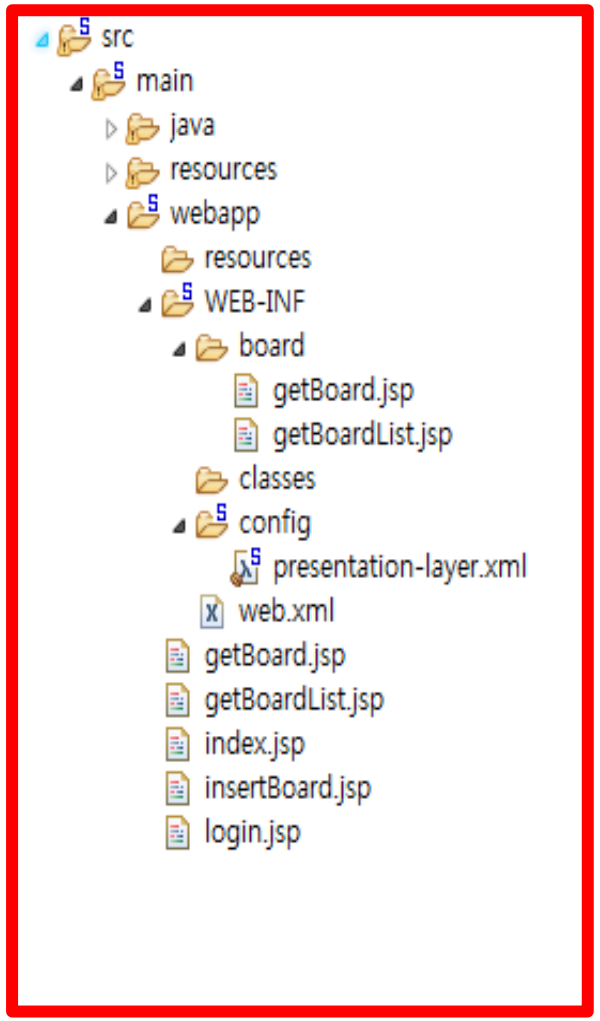
입력정보를 추출해서 VO 객체에 저장하고,
비즈니스 컴포넌트의 메소드를 호출할때
VO 객체를 매개변수로 전달
--→ 문제점 : 입력정보가 많으면 자바코드 증가, 입력정보가 변경시 controller수정해야함

Command 객체 사용!!!



컨트롤러 통합하기


src/main/java
com.springbook.biz.board
BoardService.java
BoardVO.java
com.springbook.biz.board.impl
BoardDAO.java
BoardServiceImpl.java
com.springbook.biz.common
JDBCUtil.java
com.springbook.biz.user
UserService.java
UserVO.java
com.springbook.biz.user.impl
UserDAO.java
UserServiceImpl.java
com.springbook.view.board
BoardController.java
com.springbook.view.user
LoginController.java
LogoutController.java


src
main
java
resources
webapp
resources
WEB-INF
board
getBoard.jsp
getBoardList.jsp
classes
config
presentation-layer.xml
web.xml
getBoard.jsp
getBoardList.jsp
index.jsp
insertBoard.jsp
login.jsp

- **요청 방식에 따른 @RequestMapping 사용**

```
@Controller
public class LoginController {
	@RequestMapping(value="/login.do", method=RequestMethod.GET)
	public String loginView(UserVO vo) {
		vo.setId("test");
		vo.setPassword("test123");
		return "login.jsp";
	}

	@RequestMapping(value="/login.do", method=RequestMethod.POST)
	public String login(UserVO vo, UserDAO userDAO) {
		if(userDAO.getUser(vo) != null) return "getBoardList.do";
		else return "login.jsp";
	}
}
```

- **JSP에서 Command 객체 사용하기**

```
<table border="1" cellpadding="0" cellspacing="0">
        <tr>
                <td bgcolor="orange">아  디</td>
                <td><input type="text" name="id" value="${userVO.id }"/></td>
        </tr>
        <tr>
                <td bgcolor="orange">비      호</td>
                <td><input type="password" name="password" value="${userVO.password }"/></td>
        </tr>
        <tr>
                <td colspan="2"><input type="submit" value="로  인"></td>
        </tr>
</table>
```

- **Command객체의 이름을 바꾸고 싶으면....**

```
@RequestMapping(value = "/login.do", method = RequestMethod.GET)
public String loginView(@ModelAttribute("user") UserVO vo) {
        System.out.println("로그인 화면으로 이동...");
```

```
<table border="1" cellpadding="0" cellspacing="0">
        <tr>
                <td bgcolor="orange">아  디</td>
                <td><input type="text" name="id" value="${user.id }"/></td>
```

- **Servlet API 사용**
  - **Servlet 에서 제공하는 HttpServletRequest, HttpServletResponse, HttpSession, Locale 등 다양한 객체를 매개 변수로 받을 수 있다.**

```
@RequestMapping(value="/login.do", method=RequestMethod.POST)
public String login(UserVO vo, UserDAO userDAO, HttpSession session) {
	UserVO user = userDAO.getUser(vo);
	if(user != null) {
		session.setAttribute("userName", user.getName());
		return "getBoardList.do";
	}
	else return "login.jsp";
}
```

- **Command객체에는 없는 파라미터를 Controller클래스에서 사용하려면?**

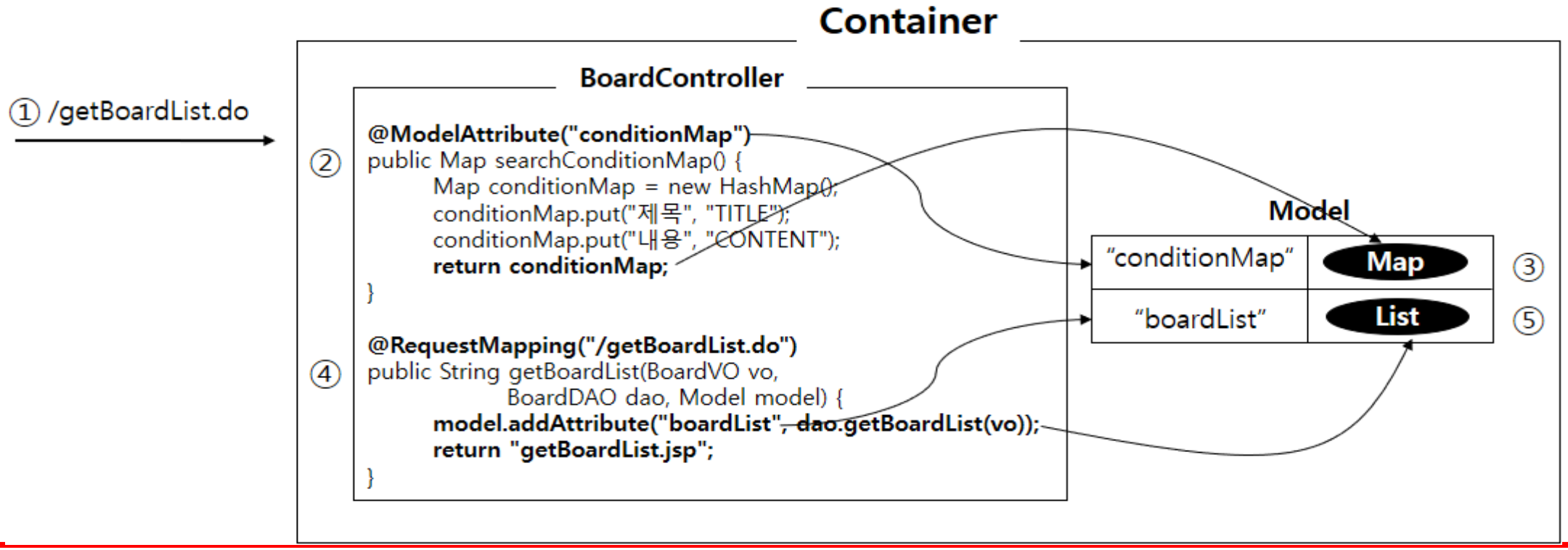
→ HTTP 요청 파라미터 정보를 추출하기 위한 @RequestParam 이용

```
@Controller
public class BoardController {
        @RequestMapping("/getBoardList.do")
        public String getBoardList(
                @RequestParam(value="searchCondition", defaultValue="TITLE", required=false) String condition,
                @RequestParam(value="searchKeyword", defaultValue="", required=false) String keyword) {
                System.out.println("검색" + condition);
                System.out.println("검색" + keyword);
                return "getBoardList.jsp";
        }
}
```

value : 화면으로부터 전달될 파라미터 이름
defaultValue : 화면으로부터 전달될 파라미터 정보가 없을 때의 설정할 기본값
required : 파라미터의 생략 여부

```
<!-- 검색 시작 -->
<form action="getBoardList.do" method="post">
  <table border="1" cellpadding="0" cellspacing="0" width="700">
   <tr>
     <td align="right">
       <select name="searchCondition">
           <c:forEach items="${conditionMap }" var="option">
               <option value="${option.value }">${option.key }
            </c:forEach>
       </select>
       <input name="searchKeyword" type="text" />
       <input type="submit" value="검색" /></td>
    </tr>
  </table>
</form>
```

- **@ModelAttribute 사용하기 (1)**
  - Command객체의 이름을 변경할 목적으로 사용
  - View에서 사용할 데이터를 설정하는 용도로도 사용
  - @RequestMappaing어노테이션이 적용된 메소드보다 먼저 호출
  - @ModelAttribute메소드의 실행결과로 리턴된 객체는 자동으로 Model에 저장



- **@ModelAttribute 사용하기 (2)**

```
@ModelAttribute("conditionMap")
public Map<String, String> searchConditionMap() {
        Map<String, String> conditionMap = new HashMap<String, String>();
        conditionMap.put("제목", "TITLE");
        conditionMap.put("내용", "CONTENT");
        return conditionMap;
}
@RequestMapping("/getBoardList.do")
public String getBoardList(BoardVO vo, BoardDAO, boardDAO, Model model) {
        // Model 정보 저장
        model.addAttribute("boardList", boardDAO.getBoardList(vo));
        return "getBoardList.jsp";
}
```

```
<table border="1" cellpadding="0" cellspacing="0" width="700">
<tr>
        <td align="right">
                <select name="searchCondition">
                <c:forEach items="${conditionMap}" var="option">
                <option value="${option.value}">${option.key}
                </c:forEach>
                </select>
                <input name="searchKeyword" type="text"/>
                <input type="submit" value="검색"/>
        </td>
</tr></table>
```

- **@SessionAttributes 사용하기 (1)**

@SessionAttributes("board")

Model에 SessionAttribute로 저장된 이름의 데이터가 있을 경우,

그 데이터를 세션(HttpSession)에도 자동으로 저장하는 설정

```
@Controller
@SessionAttributes("board") ...
@RequestMapping("/updateBoard.do")
public String updateBoard(@ModelAttribute("board") BoardVO vo) {
        ...
        boardDAO.updateBoard(vo);
        ...
}
```

Controller의 updateBoard() 메소드가 호출될 때, 스프링 컨테이너는 우선 @ModelAttribute("board") 설정을 해석하여 세션에 'board'라는 이름으로 저장된 데이터가 있는지 확인한다.
만약 있다면 해당 객체를 세션에서 꺼내서 매개변수로 선언된 vo 변수에 할당한다.
그리고 사용자가 입력한 파라미터 값을 vo 객체에 할당한다.
이때 사용자가 입력한 수정 정보(title, content...) 값만 새롭게 할당되고, 나머지(seq, writer, regDate...)는 세센에 저장된 데이터가 유지된다.