15. 모델 2(Model2)

1) MVC 구조 (모델 2)

(1) **Model(모델)** : 데이터가 저장되는 부분 → 빈즈부분(DB : BoardDTO, BoardDAO)

(2) **View(화면 출력)** : html, css, js, jsp 파일 + 자바코드 <%~%>

(3) **Controller** : Model →→→ View 전달해서 보여줄 것인가를 결정

※ Model2 가 무조건 el, jstl 태그를 사용한 다는 것을 의미하지는 않는다.
➔ 화면 구현의 편리성을 추구한다.
➔ Model2 를 배운다는 말은 컨트롤러를 배운다는 말과 같다.

2) 모델 1 의 장단점

(1) 장점
: 중소규모의 사이트 작성에 적합하다
: 적은 인원으로도 구성이 가능하다 (4~5 명 가능)

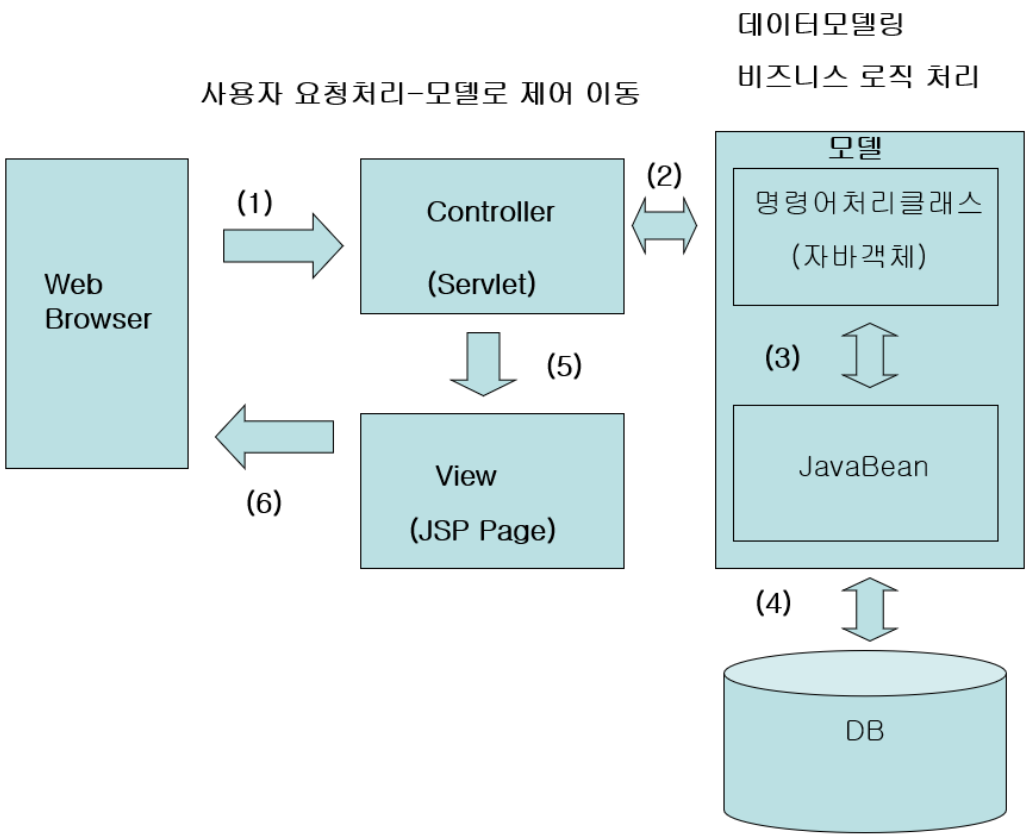
(2) 단점
: 페이지가 많아지면 유지보수가 어려워진다
→ jsp 는 자바소스 코드 + 화면구현 부분이 혼합되어 있어 유지보수가 어려워진다.
→ 중복된 코드가 많아진다.(페이지별로)
➔ 모델 1 요청 (jsp, html) ---→ 처리하는 페이지(jsp)
: 버튼, 링크문자열 클릭 ------→ list.jsp(자바코드 + 화면출력)

3) 모델 2 의 장단점

(1) 장점
: 대규모 사이트 작성하는데 적합한 구조
: 역할 분담이 이루어져 유지보수가 쉽다

(2) 단점
: 개개인의 실력이 높아야 가능 (서블릿:자바 중심)
: 구성원이 10 인 이상을 요구함

4) 모델 2 의 구조(상세)



다시 한번!

M → 데이터를 저장부분

V → 처리결과를 받아서 출력만 담당

1)요청을 받는 부분->Controller 에게 전담

2)뷰에서 존재하는 자바코드->요청명령어 클래스에서 처리

] (=액션클래스(모델 2)->컨트롤러 클래스(스프링))

C-→ 1)요청을 받아서 **그 요청에 맞는 요청명령어 클래스를 선택**->처리

|

빈즈메서드 호출(getArticles())

DB 접속

SimpleContoller.java (servlet / in test Package)

요청이 들어오면 → 서블릿이 요청을 받아서 그 요청에 해당하는 jsp 로 페이지를 이동

```
import java.io.IOException; // 입출력
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException; // 서블릿을 실행
import javax.servlet.annotation.WebServlet; // 어노테이션 때문에 필요
import javax.servlet.http.*; // 웹상에서 어떻게 서블릿을 요청하는지 (경로, 프로토콜 정보)

@WebServlet("/SimpleController")
public class SimpleController extends HttpServlet {

	private static final long serialVersionUID = 1L;
	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
		processRequest(request,response);
	}
	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
		processRequest(request,response);
	}
	// Get or Post 방식으로 요청에 상관없이 둘 다 처리할 수 있도록 method를 따로 작성
	private void processRequest(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException{
		//1. 요청명령어를 입력받아서 분석 -> 매개변수 :type
		String type = request.getParameter("type");
		System.out.println("type:"+type);
		//ex) greeting : 안녕하세요 하고 왔는데 date=> 오늘날짜 출력 하는건 맞지 않다. > invalid Type 메세지를 출력
		//2. 요청명령어에 따른 요청 기능 구현-> 요청처리 클래스의 객체생성 -> 처리(화면에 보여주는 것)
		Object resultObject = null; // String or java.util.Date 둘다 처리하기 위한 Object 선언
		if((type==null)||(type.contentEquals("greeting"))) {
			resultObject = "안녕하세요"; // 자식(String) -> Object 로 자동 형변환
		}else if(type.contentEquals("date")) {
			resultObject = new java.util.Date();
		}else {
			resultObject = "정확하게 다시 한번 알려주세요!!";
		}

		//3. 처리결과를 simpleview.jsp에 전송 --> 화면에 출력
		request.setAttribute("result",resultObject);

		//4. forward 액션태그 사용(jsp) 할 수 없기때문에, forward 시켜주는 객체가 필요
		//: RequestDispatcher
		// dispatcher : 데이터를 공유 받아서 이동할 페이지의 정보를 가진 객체
		RequestDispatcher dispatcher = request.getRequestDispatcher("/simpleview.jsp"); // 전송할 페이지명

		//5. forward : 데이터를 공유하면서 페이지를 이동시키는 기능
		// dispatcher 객체를 통해서만 사용이 가능하다 (모델2)
		dispatcher.forward(request, response);
	}
}
```

SimpleController.java---------->simpleview.jsp

요청에 따라서 ---------->화면에 보여주는 부분이 다르다.

Simpleview.jsp :

요청을 할때 요청명령어에 따라서 처리되는 결과도 다르고, 보여주는 페이지도 다르다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
        //  request.setAttribute("result", resultObject) ➔ SimpleController.java 에서 set 되었음
        // -> ${param.매개변수명} => request.getParameter("매개변수");
        // 형식) ${키명} = request.getAttribute("키명");
%>
<h1>처리결과</h1>
<h3>
        처리결과 1: <%=request.getAttribute("result") %><br>
        처리결과 2: ${result}
</h3>
</body>
</html>
```

요청측의 result 값을 가져옴
1. request.getAttribute
2. ${키이름}
같은 동작임

url ➔ http://localhost:8090/ServletTest2/SimpleController?type=imsi 입력시 다른 결과가 나온다.

16. 모델 1 → 모델 2 : 모델 1 으로 만들어진 게시판을 모델 2 로 변경

(1) 인터페이스 이용

: 클래스는 다 다르지만 요청을 받아서 처리해주는 method 가 필요

(2) 요청명령어

① 게시판 글목록 보기 : /list.do → ListAction

② 글쓰기 : /writeForm.do → WriteFormAction

③ 상세보기

④ 글수정

⑤ 글삭제

⑥ 글검색 → 6 개 → 처리해주는 클래스도 6 개 → 객체 → method 필요

(=액션클래스=컨트롤러 클래스)

주제->영화예매,사이트

->~.do(스프링)->디폴트

~.mov

~.nhn

※ 요청명령어 → 컨트롤러에서 요청명령어를 구분 (String 을 각각 구분하는 개념)

(1) 요청명령어가 컨트롤러에 전달되고, 요청명령어를 구분해 나가는 소스가 이어짐

```
if(글목록을 요청) { if(type.equals("/list.do") // /list.jsp - (X, jsp 가 아니고 do)
        글목록에 해당클래스 생성->메서드호출
    new ListAction();   ->list()
  }else if(type.equals("/writePro.do") {
        글쓰기에 해당클래스 생성->메서드호출
    new WriteAction()->write()
  }else {
        resultObject="Invalid Type";
  }
''''
```

(2)컨트롤러 하나 ->소스계속해서 X

: 동일한 소스를 그대로 계속 사용한다 (**인터페이스**를 이용하기 때문: 요청명령어만 따로 작성)

JspBoard2

|

-src

controller

ControllerAction(4)=>시작

action (인터페이스를 작성)

CommandAction->requestPro(request,response)

상속-------->계속사용(5)

|

-WebContent->list.jsp~(el,jstl 로 모두 변경)

| 자바에서 환경설정~.properties

-WEB-INF->commandPro.properties(2) ->요청명령어를 등록

/가상경로/~/요청명령어=패키지명.명령어처리클래스

순서대로 진행

#key(command~.do)=value(package...classname)

/list.do=action.ListAction

#/notice/list.do= >공지사항

#/member/list.do =>회원리스트

web.xml(3)->컨트롤러 역할을 해주는 서블릿을 지정(p370)

->commandPro.properties 읽어들일 수 있도록

환경설정을 해야 한다.

|

-lib->standard.jar,jstl.jar 복사(1)=>el,jstl 문법을 사용하기위해서

==========

!! 우선 list.jsp 를 먼저 띄워본다. !! (글쓰기, 글수정 등은 나중에) !!

web.xml : config 하는 부분

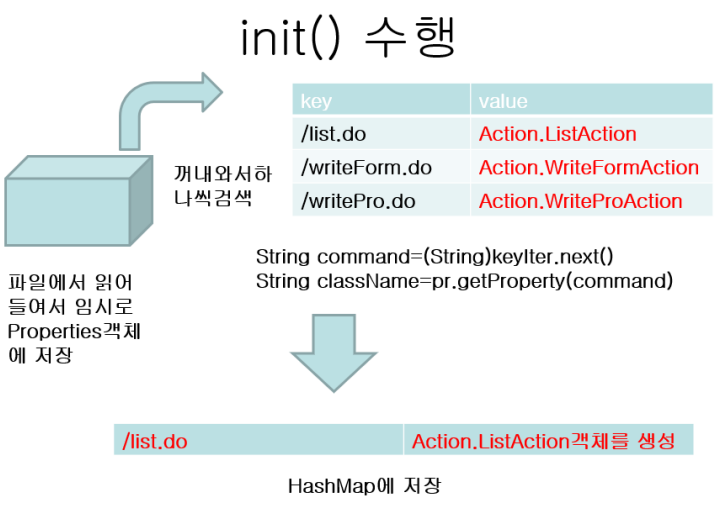
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID" version="3.1">
  <display-name>JspBoard</display-name>
  <!-- 요청(web.xml) 컨트롤러 역할을 하는 서블릿 등록  -->
  <servlet>
        <!-- do명령어가 들어오고 maapping 되서 ControllerAction 을 여기에서찾는다.  -->
        <servlet-name>ControllerAction</servlet-name>
        <servlet-class>controller.ControllerAction</servlet-class>
                                        <!-- 패지키이름.클래스명 -->
        <init-param>
              <param-name>propertyConfig</param-name>
              <param-
value>F:/02.Web_Develope/02.KIC_web_develope_webtest/4.jsp/sou/JspBoard2/WebContent/WEB-
INF/commandPro.properties</param-value>
        </init-param>
  </servlet>

  <!-- 요청의 방식에 따라, 처리할 수 있도록 설정(요청명령어를 등록)  -->
  <servlet-mapping>
        <servlet-name>ControllerAction</servlet-name>
        <url-pattern>*.do</url-pattern>
  </servlet-mapping>
```

③ ② ①

commandPro.properties

```
#commandPro.properties
#key(command)=value(packagename.classname)
/list.do=action.ListAction
```



ControllerAction.java (servlet)

```
package controller;
import java.io.*;//FileInputStream
import java.util.*;//Map,Properties (DB연동에 관련된 환결설정파일)
import javax.servlet.*;
import javax.servlet.http.*;
//추가->다른 패키지의 클래스나 인터페이스를 참조
import action.CommandAction; // 요청을 받아서 처리해주는 클래스를 사용(객체형변환)

public class ControllerAction extends HttpServlet { // Httpservlet 클래스 상속 필수
    //명령어와 명령어 처리클래스를 쌍(HashMap) 으로 저장
    private Map commandMap = new HashMap();

    //서블릿을 실행시 서블릿의 초기화 작업->생성자
    public void init(ServletConfig config) throws ServletException {

    //경로에 맞는 CommandPro.properties파일을 불러옴
    String props = config.getInitParameter("propertyConfig"); // web.xml 에서 전달받음

    //명령어와 처리클래스의 매핑정보를 저장할 Properties객체 생성
    Properties pr = new Properties();
    FileInputStream f = null;
        try { // IO 이기 때문에 Excedption 처리
            //CommandPro.properties 파일의 내용을 읽어옴
                f=new FileInputStream(props);   //props : 경로 , 파일을 불러오는 것

    pr.load(f); // 메모리에 로드 시킴
    }catch(IOException e){
       throw new ServletException(e);
    }
    ,,,,,,,,
    ,,,,,,,,
    try{
    //그 클래스의 객체를 얻어오기위해 메모리에 로드 -> JDBC에서 드라이버를 올리듯, 같은 처리
    Class commandClass = Class.forName(className); // 요청클래스를 얻어옴
    // 요청클래스명.newInstance(); ← 요청에 따른 객체를 얻어올 수 있음.
    Object commandInstance = commandClass.newInstance();
     //Map객체 commandMap에 저장
     → 요청을 처리할 때 필요로 하는 객체를 바로 사용하기 쉽게 미리 만들어 넣어주는 역할 (DI)
     commandMap.put(command, commandInstance);   //HashMap 객체에 Key와 value 넣어줌
```

초기화 부분

pr 에서 가져온 요청 **command : /list.do**
className=action.ListAction
commandClass=class action.ListAction
commandInstance=action.ListAction@4bd061e7
command**Map**={/list.do=action.ListAction@4bd061e7}
request.getRequestURI():/JspBoard2/list.do
request.getContextPath():/JspBoard2
실질적인 요청명령어 command:/list.do

```
//*** 사용자의 요청을 분석해서 해당 작업을 처리 ****
    private void requestPro(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {
        String view = null; // 요청명령어에 따라서 이동할 페이지의 이름을 저장-> ex) list.jsp
        /*
         * ListAction com = null; ListAction com = new ListAction(),,,
         * CommandAction com = new ListAction();  // 글 목록보기
         * CommandAction com = new WriteFormAction();  // 글쓰기
         * CommandAction 부모형으로 생성하여 어떤 자시객체라도 받아서 사용할 수 있게 선언
         */
        CommandAction com = null;
        try {
                // 요청명령어를 분리해주는 코드 : list.jsp 만 가져와야함
                String command = request.getRequestURI(); // 프로젝트명/요청명
                // /JspBoard2/list.do → command
                // /JspBoard2 → request.getContextPath()
                // 요청명령어를 얻어오는 방법
                if(command.indexOf(request.getContextPath())==0) {
                    //indexOf() 가 0 : 일치하는 부분
                    command = command.substring(request.getContextPath().length());
                }
                // 요청명령어 -> /list.do 는 action.ListAction 의 객체
                    com = (CommandAction)commandMap.get(command); // ~get(/list.do);
                    System.out.println("com:"+com); // action.ListAction@주소값(처리객체명)
                    // 이동할페이지명 얻어오기
                    view = com.requestPro(request, response);
                    System.out.println("이동할페이지명(view):"+view); // /list.jsp
        }catch(Throwable e) { // Exception의 상위 클래스 Throwable
                throw new ServletException(e);
        }
        // 위에서 요청명령어에 해당하는 view 로 데이터를 공유시키면서 이동

        RequestDispatcher dispatcher = request.getRequestDispatcher(view);
        dispatcher.forward(request,  response); //forward method를 이용, 데이터를 공유하면서 이동
        }
```

command 에서 공통부분인 /JspBoard2 를 제거하고 넣어줌

형변환

com:action.ListAction@4bd061e7

최종값 view /list.jsp

ListAction.java의 requestPro

list.jsp 를 뿌려주는 부분

requestPro() 수행

http://localhost:8080/Jspboard2/list.do
↓ Request.getRequestURI()
Command=/Jspboard2/list.do분리
↓ Reguset.getContextPath()
Command=/Jspboard2분리
↓ Command.substring(길이)
Command=/list.do분리
↓ Command에 대한 객체구함
Action.ListAction@주소값에 해당하는 객체

requestPro()수행2

com.requestPro(request,response)
↓ View를 얻어옴(이동할페이지)
/list.jsp
Request.getRequestDispatcher(list.jsp)
↓
dispatcher.forward(request,response)
↓
http://localhost:8080/board2/list.jsp로 이동

ListAction.java (list.jsp 부분을 구현하는 클래스)

기존 list.jsp 의 초반 <% %> 내에 설정부분을 가져와서 넣어준다

```
package action;
import javax.servlet.http.*;
import pjh.board.BoardDAO; import java.util.*;
//list.do = action.ListAction (commandPro.properties 에서 넘겨받은 값)
//각 액션클래스에서 공통으로 사용하는 메서드를 사용하기 위해서 상속(CommandAction)
public class ListAction implements CommandAction {
	@Override
	public String requestPro(HttpServletRequest request, HttpServletResponse response) throws Throwable {
		// 1 list.jsp에서 처리했던 자바코드를 액션클래스에서 대신 처리함
		int pageSize = 10;// numPerPage(페이지당 보여주는 게시물수(=레코드수))
		int blockSize = 3;// pagePerBlock(블럭당 보여주는 페이지수)
		// 페이징 처리에 해당하는 환경설정을 다 코딩
		→게시판은 처음 실행시키면 ->무조건 1페이지부터 출력
		String pageNum = request.getParameter("pageNum");
		if (pageNum == null) {
			pageNum = "1
		}
		int currentPage = Integer.parseInt(pageNum);// 현재페이지->nowPage
		int startRow = (currentPage - 1) * pageSize + 1;// 시작 레코드번호
		int endRow = currentPage * pageSize;// 1*10=10,2*1=20,30
		int count = 0;// 총레코드수
		int number = 0;// beginPerPage;(페이지별로 시작하는 맨 처음에 나오는 게시물번호)
		List articleList = null;// 화면에 출력할 레코드를 저장할 변수
		BoardDAO dbPro = new BoardDAO();
		count = dbPro.getArticleCount();// select count(*) from board=>getMemberList()
		System.out.println("현재 레코드수(count)=>" + count);

		if (count > 0) { // startRow,endRow(X)=>pageSize(O)
			articleList = dbPro.getArticles(startRow, pageSize);
		}else {
			//count=0 일 때
			articleList = Collections.EMPTY_LIST;
		}
		number = count - (currentPage - 1) * pageSize;
		System.out.println("페이지별 number=>" + number);
		// 2.처리결과를 list.jsp에 전달 (메모리에 저장, setAttribute / 공유시키기 위해서)
		request.setAttribute("currentPage",currentPage);
		request.setAttribute("startRow",startRow);
		request.setAttribute("count",count);
		request.setAttribute("pageSize",pageSize);
		request.setAttribute("blockSize",blockSize);
		request.setAttribute("number",number);
		request.setAttribute("articleList",articleList);
		// 3.이동할 페이지명을 지정 -> return "경로를 포함해서 이동할 페이지명"
		return "/list.jsp";
	}
```

CommandAction.java (interface 를 선언하는 servlet)

```
package action;
// 기능은 다르지만 요청을 받아서 처리해주는 method가 필요 → 공통의 method로 작성 (interface)

import javax.servlet.http.*; // request, response 객체가 필요하기 때문

public interface CommandAction {
// 이동할 페이지의 경로와 페이지명이 필요(요청에 따른) → 반환값이 String (Spring : Model and View)
        public String requestPro(HttpServletRequest request, HttpServletResponse response)
                throws Throwable;
}
```

!! 이제 글쓰기를 구현해보자 !!

: wirteForm.jsp 를 모델 2 로 구현하기

(1) 요청명령어를 등록한다. (list.jsp 와 과정은 비슷하다)

commandPro.properties : properties 설정에 write 와 관련된 속성을 추가

```
#commandPro.properties
#key(command)=value(packagename.classname)
/list.do=action.ListAction
/writeForm.do = action.WriteFormAction
/wirtePro.do = action.WriteProAction
```

writeForm.jsp → writeForm.do
writePro.jsp → writePro.do

흐름)

(1) Index.jsp → list.do → (내부처리, 윗 내용 참조) → list.jsp → 글쓰기 버튼 클릭 →writeForm.do 이동
(2) writeForm.do → WriteFromAction.java 클래스 참조 → (내부처리) → writeForm.jsp 로 이동
(3) WriteFrom.jsp 에서 글쓰기 버튼 클릭 →writePro.do 이동
(4) writePro.do → WirtePro.Action.java클래스 참조 → (내부처리) → writePro.jsp로 이동
(5) writePro.jsp 에서는 다른 처리가 없이 바로 list.do를 호출

WriteFormAction.java (writeForm.jsp 부분을 구현하는 클래스)

```
package action;
import javax.servlet.*;
// 글쓰기 버튼을 클릭하면  /writeForm.do ➔  action.WriteFormAction 호출
public class WriteFormAction implements CommandAction {
        // 모델1에서 대신 처리해줬던 자바코드를 대행해주는 액션 클래스 -> writeForm.jsp
        @Override // CommandAction 인터페이스를 상속했기 때문에 Override
        public String requestPro(HttpServletRequest request, HttpServletResponse response) throws Throwable {
            //1.jsp 가 처리할 문장 대신 수행
            int num=0, ref=1, re_step=0,re_level=0; // inesrtArticle(BoardDTO article);
            // content.jsp에서 매개변수가 전달되어 받은거라면, (답변 클릭해서 넘어온거라면)
            if(request.getParameter("num")!=null){ // 양수라면
                    num=Integer.parseInt(request.getParameter("num"));
                    ref=Integer.parseInt(request.getParameter("ref"));
                    re_step=Integer.parseInt(request.getParameter("re_step"));
                    re_level=Integer.parseInt(request.getParameter("re_level"));
            }
            //2.처리결과 -> request 객체에 저장 (spring 에서는 ModelAndView 객체에 저장됨)
            request.setAttribute("num",num); // <-> request.getAttribute("num") -> ${num}
            request.setAttribute("ref",ref);
            request.setAttribute("re_step",re_step);
            request.setAttribute("re_level",re_level);
            //3.이동할 페이지를 지정 (String 값을 넘김)
            return "/writeForm.jsp"; // ControllerAction 클래스가 받아서 리턴값/writeForm.jsp 로 이동시켜줌
        }
}
```

기존 WriteForm.jsp 의 초반 <% %> 내에 설정부분

WriteForm.jsp (WriteFormAction 에서 저장된 값을 뿌려주는 jsp 파일) : 글쓰기 페이지 라는 소리

```
<body bgcolor="#e0ffff">
<b>글쓰기</b>
<form method="post" name="writeform" action="/JspBoard2/writePro.do" onsubmit="return writeSave()">
<input type="hidden" name="num" value="${num}"/>
<input type="hidden" name="ref" value="${ref}"/>
<input type="hidden" name="re_step" value="${re_step}"/>
<input type="hidden" name="re_level" value="${re_level}"/>
<table width="400" border="1" cellspacing="0" cellpadding="0"   bgcolor="#e0ffff" align="center">
,,,
,,, 테이블 구현 부분 <table><tr><td> 블라블라~~~~
,,,
<td colspan=2 bgcolor="#b0e0e6" align="center">
   <input type="submit" value="글쓰기" >
   <input type="reset" value="다시작성">
   <input type="button" value="목록보기" OnClick="window.location='/JspBoard2/list.do'">
</td></tr></table>
</form>
</body>
```

WriteFormAction 으로부터 처리된 값들을 hidden으로 넘긴다

WriteProAction.java (글쓰기 버튼을 눌렀을 때 넘겨받은 클래스) : 글쓰기를 처리해주는 부분

```
package action;
import javax.servlet.http.*;
import java.sql.Timestamp;
import pjh.board.*;
public class WriteProAction implements CommandAction {
	@Override
	public String requestPro(HttpServletRequest request, HttpServletResponse response) throws Throwable {
	request.setCharacterEncoding("utf-8");  // 한글처리
	/*
		BoardDTO 의 setter method 5개 + hidden 4개 + 2개(작성)
		BoardDTO article = new BoardDTO();
		<jsp:useBean id="article" class="pjh.board.BoardDTO"/>
		<jsp:setProperty name="article" property="*"/>
	*/

	BoardDTO article = new BoardDTO();
	article.setNum(Integer.parseInt(request.getParameter("num")));
	article.setWriter(request.getParameter("writer"));
	article.setEmail(request.getParameter("email"));
	article.setSubject(request.getParameter("subject"));
	article.setPasswd(request.getParameter("passwd"));
	article.setReg_date(new Timestamp(System.currentTimeMillis()));
	//ref,re_step,re_level
	article.setRef(Integer.parseInt(request.getParameter("ref")));
	article.setRe_step(Integer.parseInt(request.getParameter("re_step")));
	article.setRe_level(Integer.parseInt(request.getParameter("re_level")));
	article.setContent(request.getParameter("content"));
	article.setIp(request.getRemoteAddr());
	//readcount --> default로 설정한 관계로 자동으로 0이 들어감

	BoardDAO dbPro = new BoardDAO();
	dbPro.insertArticle(article);
	return "/writePro.jsp";
	}
}
```

데이터를 article 객체에 넣고, 그 객체를 BoardDAO에 결합 → DB에 저장

여기선 데이터가 공유되지 않는다.
DB에 저장되고, WirtePro.jsp 로 가면 단지 list.jsp 로 이동하는 코드만 있을 뿐이다.

WritePro.jsp (WriteProAction.java 에서의 목적지 : list.do 로 바로 연결만 해주는 기능)

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%
	// 데이터를 공유하면서 이동할 필요 없이, 그냥 이동
	//response.sendRedirect("http://localhost:8090/JsapBoard2/list.do");
%>
<meta http-equiv ="Refresh" content="0;url=/JspBoard2/list.do">
```

Content(게시물 내용표시) /Update(수정)/ Delete(삭제) 의 과정도 비슷하다.

commandPro.properties : properties 설정에 각 기능과 관련된 속성을 추가

```
/content.do = action.ContentAction
/updateForm.do = action.UpdateFormAction
/updatePro.do = action.UpdateProAction
/deleteForm.do = action.DeleteFormAction
/deletePro.do = action.DeleteProAction
```

~Action.java 에서는 대부분 기존 모델 1 에서 jsp 페이지 상단의 "기능" 부분을 구현하고 구현된 기능을 수행하고 메모리에 값을 저장한다.

~.jsp 에서는 ~Action.java 에서 저장된 값들을 표현언어 ${ } 를 이용해서 표시해준다.
(액션태그를 이용하여 '조건문'을 활용하기도 한다.)