Chapter

08

모델1 방식의 회원제 게시판 만들기

학습 목표 및 순서

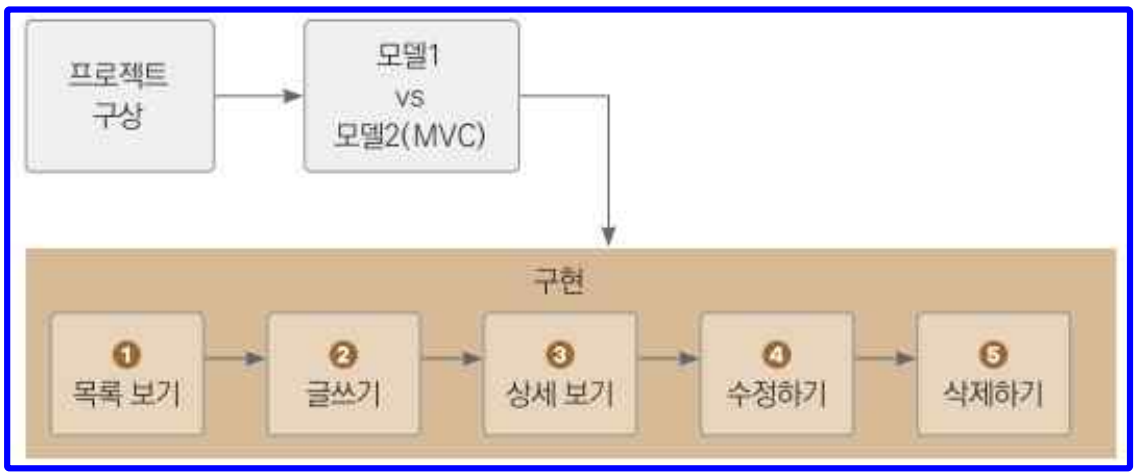
▣ 학습 목표

. 데이터베이스와 연동하여 입력, 수정, 삭제, 조회할 수 있는 모델1 방식의 회원제 게시판 제작

. 폼값 처리에는 내장 객체를 사용하고, 로그인은 session 영역을 사용

. 즉 7장까지의 모든 내용을 통합하여 게시판을 구현함으로써 학습한 내용을 복습

▣ 학습 순서



학습 목표 및 순서

▣ 활용 사례

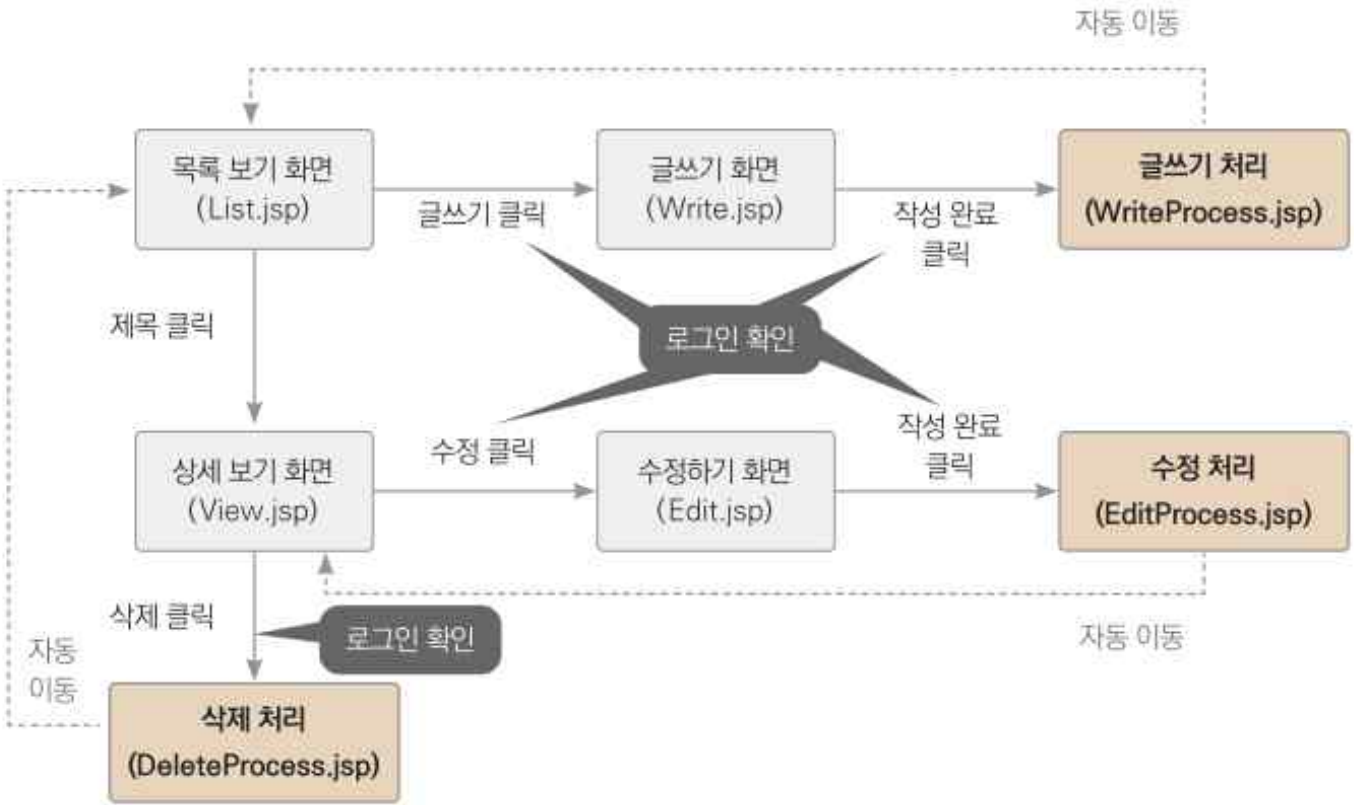
. 게시판의 가장 기본적인 기능은 단순히 글 하나를 저장하는것

. 저장되는 데이터를 다양하게 변경하면 쇼핑몰의 상품 리스트, 재고 관리, 회원 관리 등에서도 게시판을 활용할 수 있음

. 이처럼 게시판은 여러 형태의 데이터를 관리하기 위해 반드시 필요

8.1 프로젝트 구상

▣ 회원제 게시판의 프로세스



- 게시판은 목록부터 출발
- 제목을 클릭하여 상세보기
- 내용 확인 후 수정 및 삭제
- 위 기능을 회원제로 제작

- 비회원(로그아웃) 상태
  - 목록 보기
  - 상세 보기
- 회원(로그인) 상태
  - 글쓰기
  - 수정하기
  - 삭제하기

8.1 프로젝트 구상

▣ 테이블 및 시퀀스 생성

- 5장에서 생성했던 2개의 테이블을 사용
- 회원관리 : member 테이블

컬럼명	데이터 타입	null 허용	키	기본값	설명
id	varchar2(10)	N	기본키		아이디
pass	varchar2(10)	N			패스워드
name	varchar2(30)	N			이름
regidate	date	N		sysdate	가입날짜

- 아이디 컬럼은 기본키(Primary key)로 지정
- 게시판에 글을 쓰거나 수정, 삭제를 위한 회원인증에 사용

8.1 프로젝트 구상

▣ 테이블 및 시퀀스 생성(계속)

. 게시물 관리 : board 테이블

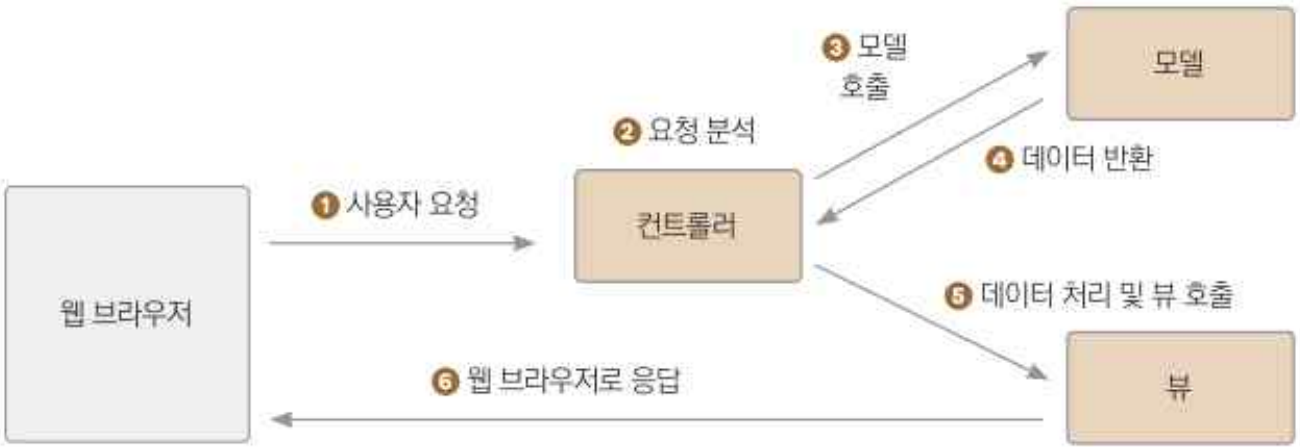
컬럼명	데이터 타입	null 허용	키	기본값	설명
num	number	N	기본키		일련번호. 기본키
title	varchar2(200)	N			게시물의 제목
content	varchar2(2000)	N			내용
id	varchar2(10)	N	외래키		작성자의 아이디. member 테이블의 id를 참조하는 외래키
postdate	date	N		sysdate	작성일
visitcount	number(6)				조회수

. 사용자가 입력한 게시물을 저장
. 일련번호 컬럼을 기본키(Primary key)로 지정
. 회원제이므로 member 테이블의 id 컬럼과 board 테이블의 id 컬럼은 외래키(foreign key)로 지정

8.2 모델1 구조와 모델2 구조(MVC 패턴)

▣ MVC 패턴

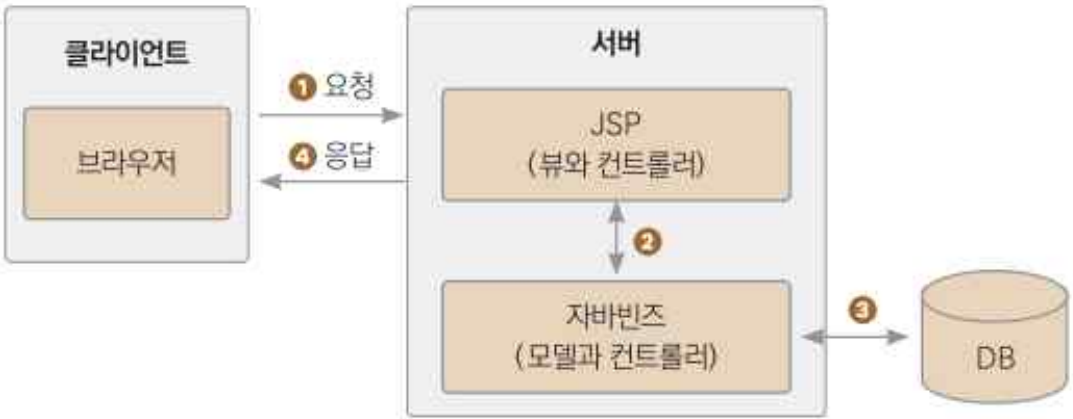
- 모델1과 모델2의 차이를 이해하기 위해 MVC 패턴을 알아야 함
- MVC는 모델(Model), 뷰(View), 컨트롤러(Controller)의 약자로, 소프트웨어를 개발하는 방법론의 일종
  - Model : 비즈니스 로직 혹은 데이터베이스 관련 작업 담당
  - View : 사용자에게 보여주기 위한 작업 담당
  - Controller : 사용자의 요청을 받아 분석 및 처리를 담당. 이 과정에서 Model과 View를 호출.



8.2 모델1 구조와 모델2 구조(MVC 패턴)

▣ 모델1

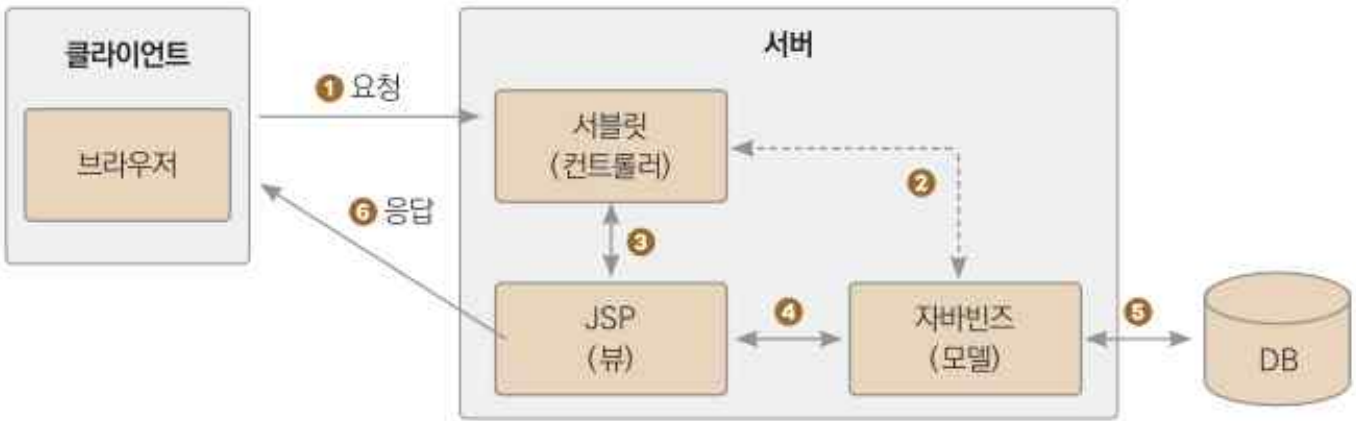
- JSP로 개발하는 방식
- 사용자의 요청을 JSP가 받아서 처리한 후 응답
- 따라서 JSP에 Controller 와 View가 혼재되어 있음
- 장점 : 개발속도가 빠르고, 학습하기 쉬움
- 단점 : 뷰와 컨트롤러를 JSP에서 구현해야 하므로 코드가 복잡해지고 유지보수가 어려움



8.2 모델1 구조와 모델2 구조(MVC 패턴)

▣ 모델2

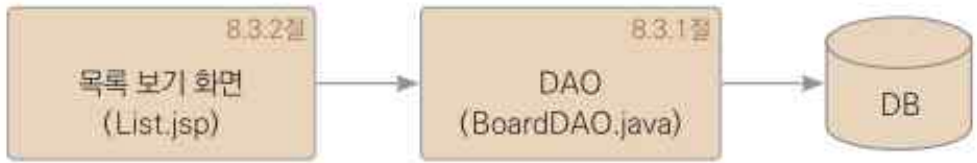
- Servlet으로 개발하는 방식
- 사용자의 요청을 Controller인 Servlet이 직접 받음
- 요청을 분석한 후 Model을 호출하여 데이터를 받음
- 데이터를 최종적으로 View로 전달하여 응답
- 장점 : MVC가 각 역할을 수행하므로 업무 분담이 명확하고 코드가 간결해짐
- 단점 : 구조가 복잡하여 학습하기 어렵고, 개발기간이 길어질 수 있음



8.3 목록 보기

▣ 목록 보기

. 페이지 기능없이 전체 게시물을 한꺼번에 출력하는 형태로 제작
. 페이지로 나누는 기능은 9장에서 추가



▣ **DAO 와 DTO 준비**

```
package model1.board;  ❶

public class BoardDTO {
    // 멤버 변수 선언 ❷
    private String num;
    private String title;
    private String content;
    private String id;
    private java.sql.Date postdate;
    private String visitcount;
    private String name;  ❸
```

예제 8-1] Java Resources/model1/board/BoardDTO.java

. board 테이블에 데이터 저장 및 전송 담당
. 멤버변수 생성 후 getter / setter 자동생성
. 생성자는 별도로 생성하지 않음

8.3 목록 보기

▣ DAO 와 DTO 준비(계속)

```
package model1.board;  ❶

import jakarta.servlet.ServletContext;
import common.JDBConnect;

public class BoardDAO extends JDBConnect {  ❷
    public BoardDAO(ServletContext application) {
        super(application);  ❸
    }
}
```

예제 8-2] Java Resources/model1/board/BoardDAO.java

- JDBConnect 클래스를 상속
- 부모 클래스의 생성자 중 application 내장 객체를 인수로 받아 DB에 연결하는 생성자 호출

▣ JSP 페이지 구현을 위한 메서드 추가

- selectCount( ) : board 테이블에 저장된 게시물의 개수를 반환. 목록에서 번호를 출력 하기 위해 사용.
- selectList( ) : board 테이블의 레코드를 가져와서 반환. 반환한 ResultSet 객체로부터 게시물 목록을 반복하여 출력

8.3 목록 보기

▣ 게시물 개수 세기

예제 8-3] Java Resources/model1/board/BoardDAO.java

```
// 검색 조건에 맞는 게시물의 개수를 반환합니다.
public int selectCount(Map<String, Object> map) {  ❷
    int totalCount = 0; // 결과(게시물 수)를 담을 변수

    // 게시물 수를 얻어오는 쿼리문 작성
    String query = "SELECT COUNT(*) FROM board";
    if (map.get("searchWord") != null) {  ❹
❸       query += " WHERE " + map.get("searchField") + " "
              + " LIKE '%" + map.get("searchWord") + "%'";
    }

    try {  ❺
        stmt = con.createStatement();  // 쿼리문 생성 ❻
        rs = stmt.executeQuery(query); // 쿼리 실행 ❼
        rs.next();  // 커서를 첫 번째 행으로 이동 ❽
        totalCount = rs.getInt(1);  // 첫 번째 컬럼 값을 가져옴 ❾
    }
    catch (Exception e) {
        System.out.println("게시물 수를 구하는 중 예외 발생");
        e.printStackTrace();
    }

    return totalCount;  ❿
}
```

❸ count() 함수를 이용해서 게시물의 갯수 카운트
❹ 검색어가 있다면 where절 동적 추가
❺ 쿼리문 실행 및 결과 얻어옴

8.3 목록 보기

▣ 게시물 목록 가져오기

예제 8-4] Java Resources/model1/board/BoardDAO.java

```
// 검색 조건에 맞는 게시물 목록을 반환합니다.
public List<BoardDTO> selectList(Map<String, Object> map) {  ❶
    List<BoardDTO> bbs = new Vector<BoardDTO>();
                                        // 결과(게시물 목록)를 담을 변수 ❷

    String query = "SELECT * FROM board ";  ❸
    if (map.get("searchWord") != null) {  ❹
        query += " WHERE " + map.get("searchField") + " "
               + " LIKE '%" + map.get("searchWord") + "%' ";
    }
    query += " ORDER BY num DESC ";  ❺

    try {
        stmt = con.createStatement();    // 쿼리문 생성 ❻
        rs = stmt.executeQuery(query);   // 쿼리 실행 ❼
```

❸ 목록을 가져오기 위한 쿼리문
❹ 검색어가 있다면 where절 동적 추가
❺ 게시물은 일련번호의 내림차순 정렬

8.3 목록 보기

▣ 게시물 목록 가져오기(계속)

```
        while (rs.next()) {  // 결과를 순화하며... ❽
            // 한 행(게시물 하나)의 내용을 DTO에 저장
            BoardDTO dto = new BoardDTO();

            dto.setNum(rs.getString("num"));           // 일련번호
            dto.setTitle(rs.getString("title"));       // 제목
            dto.setContent(rs.getString("content"));   // 내용
            dto.setPostdate(rs.getDate("postdate"));   // 작성일
            dto.setId(rs.getString("id"));             // 작성자 아이디
            dto.setVisitcount(rs.getString("visitcount"));  // 조회수

            bbs.add(dto);  // 결과 목록에 저장 ❿
        }
    }
    catch (Exception e) {
        System.out.println("게시물 조회 중 예외 발생");
        e.printStackTrace();
    }

    return bbs;  ⓫
}
```

❽ ResultSet에 저장된 레코드의 갯수만큼 반복

❾ DTO에 레코드를 저장한 후

❿ List에 추가

8.3 목록 보기

▣ 게시물 목록 출력하기

예제 8-5] 08Board/List.jsp

```
<%
// DAO를 생성해 DB에 연결 ❶
BoardDAO dao = new BoardDAO(application);

// 사용자가 입력한 검색 조건을 Map에 저장 ❷
Map<String, Object> param = new HashMap<String, Object>();
String searchField = request.getParameter("searchField");
String searchWord = request.getParameter("searchWord");
if (searchWord != null) {
    param.put("searchField", searchField);
    param.put("searchWord", searchWord);
}

int totalCount = dao.selectCount(param);  // 게시물 수 확인 ❸
List<BoardDTO> boardLists = dao.selectList(param);  // 게시물 목록 받기 ❹
dao.close();  // DB 연결 닫기
%>
<!DOCTYPE html>
```

❷ 검색을 위한 파라미터가 있다면 DAO로 전달하기 위해 Map컬렉션에 저장

❸❹ 앞에서 작성한 DAO의 메서드를 호출하여 게시물수와 목록을 반환받음

8.3 목록 보기

▣ 게시물 목록 출력하기(계속)

```
<%
if (boardLists.isEmpty()) {
    // 게시물이 하나도 없을 때 ❼
%>
        <tr>
            <td colspan="5" align="center">
                등록된 게시물이 없습니다^^*
            </td>
        </tr>
<%
}
else {
    // 게시물이 있을 때 ❽
    int virtualNum = 0;  // 화면상에서의 게시물 번호
    for (BoardDTO dto : boardLists)
    {
        virtualNum = totalCount--;  // 전체 게시물 수에서 시작해 1씩 감소
%>
        <tr align="center">
            <td><%= virtualNum %></td>  <!--게시물 번호-->
            <td align="left">  <!--제목(+ 하이퍼링크)-->
                <a href="View.jsp?num=<%= dto.getNum() %>"><%= dto.getTitle() %>
</a> ❾
            </td>
            <td align="center"><%= dto.getId() %></td>      <!--작성자 아이디-->
            <td align="center"><%= dto.getVisitcount() %></td>  <!--조회수-->
            <td align="center"><%= dto.getPostdate() %></td>    <!--작성일-->
        </tr>
<%
    }
}
%>
```

❽ List 컬렉션에 저장된 게시물이 있다면 갯수만큼 반복하여 <tr>태그를 출력

8.3 목록 보기

▣ 게시물 목록 출력하기(계속)

목록 보기(List)

제목 검색하기

번호	제목	작성자	조회수	작성일
5	지금은 겨울입니다	musthave	0	2021-08-21
4	지금은 가을입니다	musthave	0	2021-08-21
3	지금은 여름입니다	musthave	0	2021-08-21
2	지금은 봄입니다	musthave	0	2021-08-21
1	제목1입니다	mu		

목록에 진입했을때는 전체게시물
이
출력됨

검색어를 입력한 경우 쿼리스트링으로 파라미터가 전송되고 DAO에서는 where절이 동적으로 추가됨

localhost:8081/MustHaveJSP/08Board/List.jsp?searchField=title&searchWord=겨울

로그인 게시판(페이징X) 게시판(페이징O)

목록 보기(List)

제목 검색하기

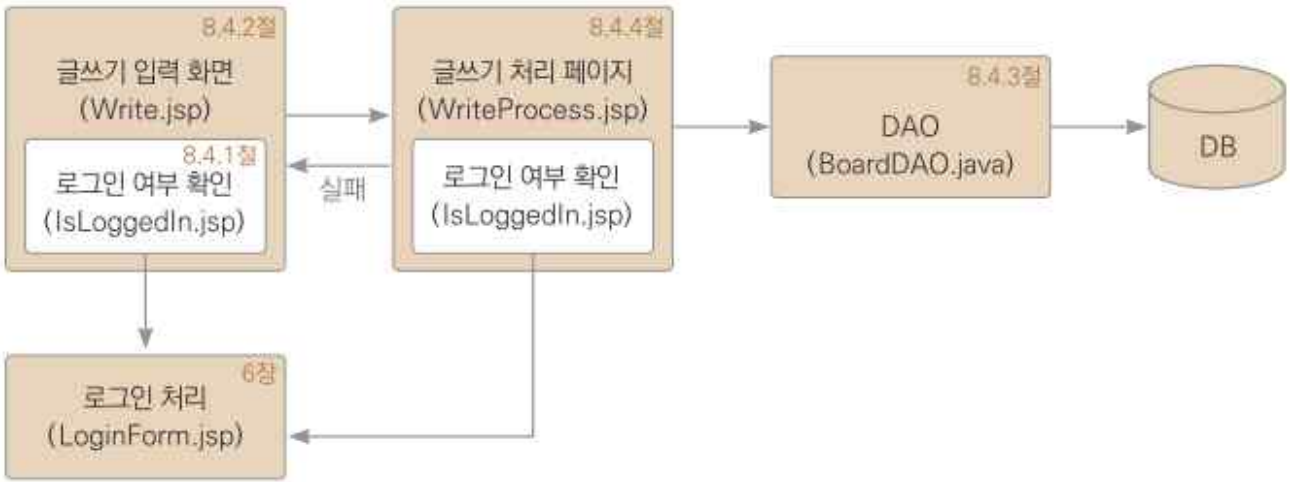
번호	제목	작성자	조회수	작성일
1	지금은 겨울입니다	musthave	0	2021-08-21

글쓰기

8.4 글쓰기

▣ 글쓰기 처리 프로세스

- 회원제 게시판이므로 로그인 후 글쓰기 페이지로 진입 가능
- 로그인이 안된 상태라면 로그인 페이지로 이동



8.4 글쓰기

▣ 로그인 여부 확인

. 로그인 정보가 없을때 로그인 페이지로 이동시키는 페이지 작성

예제 8-7] 08Board/IsLoggedIn.jsp

```
<%@ page import="utils.JSFunction"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%
if (session.getAttribute("UserId") == null) {  ❶
    JSFunction.alertLocation("로그인 후 이용해주십시오.",
                             "../06Session/LoginForm.jsp", out);  ❷
    return;  ❸
}
%>
```

❶ 세션 영역에 저장된 회원아이디가 없다면 로그아웃 상태이므로 로그인 페이지로 이동

뒷부분에서 로그인이 필요한 모든 페이지 상단에 인클루드(include)

8.4 글쓰기

▣ 글쓰기 페이지

- <form> 태그 하위에 <input> 태그로 구성된 일반적인 작성폼
- 입력값이 있는지 확인하기 위한 Javascript 함수로 구성

예제 8-8] 08Board/Write.jsp

```
<script type="text/javascript">
function validateForm(form) {  // 폼 내용 검증 ❷
    if (form.title.value == "") {
        alert("제목을 입력하세요.");
        form.title.focus();
        return false;
    }
```

```
<form name="writeFrm" method="post" action="WriteProcess.jsp"  ❸
      onsubmit="return validateForm(this);">  ❹
    <table border="1" width="90%">
        <tr>
            <td>제목</td>
            <td>  ❺
                <input type="text" name="title" style="width: 90%;" />
            </td>
```

❷ 입력값 검증을 위한 Javascript
❸ 게시물 작성을 위한 <form> 태그 및 하위 요소들로 구성

8.4 글쓰기

▣ DAO에 글쓰기 메서드 추가

예제 8-9] Java Resources/model1/board/BoardDAO.java

```
// 게시글 데이터를 받아 DB에 추가합니다. ❶
public int insertWrite(BoardDTO dto) {
    int result = 0;

    try {
        // INSERT 쿼리문 작성 ❷
        String query = "INSERT INTO board ( "
                     + " num,title,content,id,visitcount) "
                     + " VALUES ( "
                     + " seq_board_num.NEXTVAL, ?, ?, ?, 0)";  ❸

        psmt = con.prepareStatement(query);  // 동적 쿼리 ❹
        psmt.setString(1, dto.getTitle());
        psmt.setString(2, dto.getContent());                   ❺
        psmt.setString(3, dto.getId());

        result = psmt.executeUpdate();  ❻
    }
    catch (Exception e) {
```

❷ 인파라미터가 있는 동적 insert 쿼리문 작성

❸ 일련번호는 시퀀스를 통해 입력

❹❺❻ 동적 쿼리문에 인파라미터를 설정한 후 실행. 결과는 1 혹은 0 반환됨.

8.4 글쓰기

▣ 글쓰기 처리 페이지 작성

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ include file="./IsLoggedIn.jsp"%> ❶

<%
// 폼값 받기 ❷
String title = request.getParameter("title");
String content = request.getParameter("content");

// 폼값을 DTO 객체에 저장 ❸
BoardDTO dto = new BoardDTO();
dto.setTitle(title);
dto.setContent(content);
dto.setId(session.getAttribute("UserId").toString()); ❹

// DAO 객체를 통해 DB에 DTO 저장 ❺
BoardDAO dao = new BoardDAO(application);
int iResult = dao.insertWrite(dto);
dao.close();

// 성공 or 실패?
if (iResult == 1) {
    response.sendRedirect("List.jsp"); ❻
} else {
    JSFunction.alertBack("글쓰기에 실패하였습니다.", out); ❼
}
%>
```

예제 8-10] webapp/08Board/WriteProcess.jsp

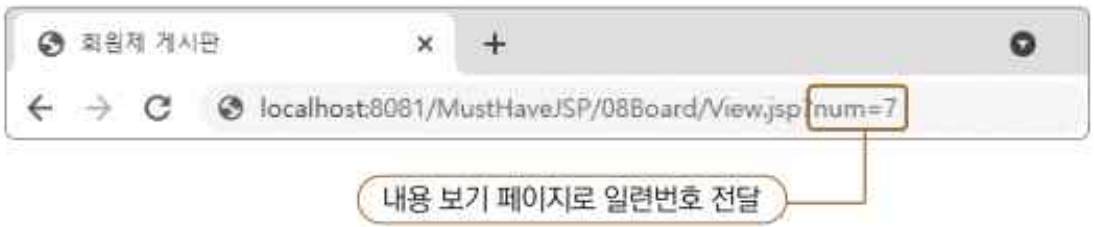
❶ 로그인 확인을 위한 인클루드
❷❸ 전송된 폼값을 받아 DTO에 저장
❺ 입력을 위해 insertWrite() 호출
❻ insert에 성공하면 목록으로 이동

Note 작성한 글이 제대로 등록되는지 테스트

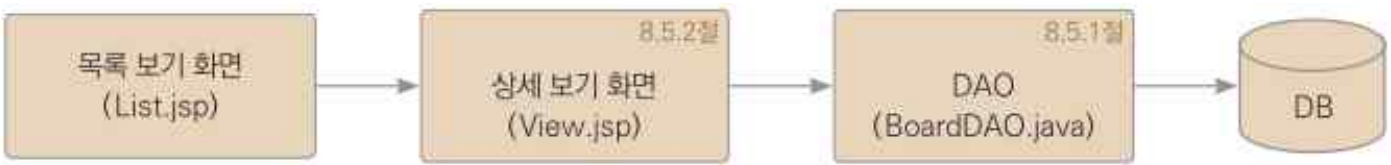
8.5 상세 보기

▣ 상세 보기

- 사용자가 선택한 게시물 하나를 조회하여 보여주는 기능
- 게시물 클릭 시 게시물의 일련번호(num 컬럼)를 매개변수로 전달
- 일련번호를 통해 데이터베이스에서 게시물 내용을 가져오고, 조회수(visitcount 컬럼)를 증가



▣ 프로세스



8.5 상세 보기

▣ DAO 준비

. 게시물 조회를 위한 메서드 추가

예제 8-11] Java Resources/model1/board/BoardDAO.java

```
public BoardDTO selectView(String num) {  ❶
    BoardDTO dto = new BoardDTO();

    // 쿼리문 준비 ❷
    String query = "SELECT B.*, M.name "  ❸
                 + " FROM member M INNER JOIN board B "  ❹
                 + " ON M.id=B.id "
                 + " WHERE num=?";
    try {
        psmt = con.prepareStatement(query);
        psmt.setString(1, num);     // 인파라미터를 일련번호로 설정 ❺
        rs = psmt.executeQuery();  // 쿼리 실행 ❻

        // 결과 처리
        if (rs.next()) {  ❼
            dto.setNum(rs.getString(1));
```

❷ board 와 member 테이블을 서로 조인하여 회원의 이름까지 select

❼ 레코드를 ResultSet으로 반환받은 후 DTO객체에 저장

[Note] DTO에 저장하는 부분은 동일한 패턴이므로 생략함

8.5 상세 보기

▣ DAO 준비

. 게시물 조회수 증가를 위한 메서드 추가

예제 8-12] Java Resources/model1/board/BoardDAO.java

```
// 지정한 게시물의 조회수를 1 증가시킵니다.
public void updateVisitCount(String num) {  ❶
    // 쿼리문 준비 ❷
    String query = "UPDATE board SET "
                 + " visitcount=visitcount+1 "
                 + " WHERE num=?";

    try {
        psmt = con.prepareStatement(query);
        psmt.setString(1, num);  // 인파라미터를 일련번호로 설정 ❸
        psmt.executeQuery();     // 쿼리 실행 ❹
    }
    catch (Exception e) {
        System.out.println("게시물 조회수 증가 중 예외 발생");
        e.printStackTrace();
    }
}
```

❷ number 타입의 컬럼인 visitcount를 1증가 시키는 update 쿼리문 작성
❸ 쿼리문 실행

[Note] 단순히 조회수만 1증가 시키므로 별도로 반환값은 사용하지 않음

8.5 상세 보기

▣ 상세 보기 화면 작성

예제 8-13] webapp/08Board/View.jsp

```
<%
String num = request.getParameter("num");  // 일련번호 받기 ❶

BoardDAO dao = new BoardDAO(application);  // DAO 생성 ❷
dao.updateVisitCount(num);                 // 조회수 증가 ❸
BoardDTO dto = dao.selectView(num);        // 게시물 가져오기 ❹
dao.close();                               // DB 연결 해제
%>
```

```
<table border="1" width="90%">
    <tr>
        <td>번호</td>
        <td><%= dto.getNum() %></td>
        <td>작성자</td>
        <td><%= dto.getName() %></td>
    </tr>
    <tr>
```

❶ 파라미터로 전달된 일련번호를 받은 후 게시물 select 및 조회수 증가

❺ DTO에 저장된 레코드를 getter()를 통해 웹 브라우저에 출력

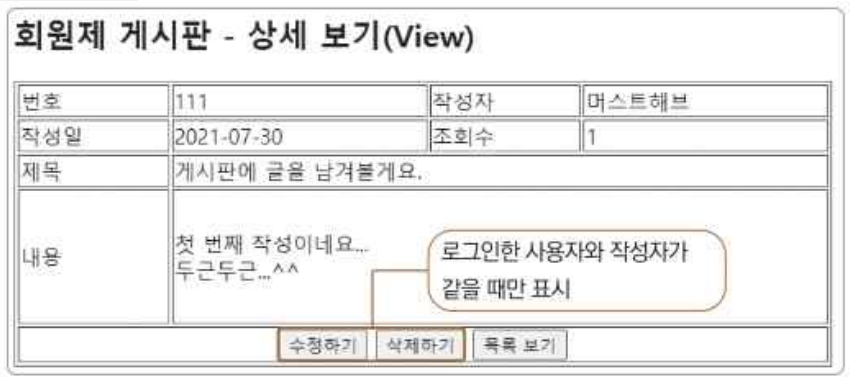
8.5 상세 보기

▣ 상세 보기 화면 작성(계속)

```
<tr>
    <td colspan="4" align="center">
    <%
    if (session.getAttribute("UserId") != null
        && session.getAttribute("UserId").toString().equals(
                                                dto.getId())) {
    %>
    <button type="button"
⑦           onclick="location.href='Edit.jsp?num=<%= dto.getNum() %>';">
        수정하기</button>
    <button type="button" onclick="deletePost();">삭제하기</button>
    <%
    }
    %>
```

❼ 세션영역에 저장된 아이디와 게시물에 저장된 아이디를 비교.
즉 작성자 본인이게만 수정, 삭제 버튼이 보임.

[Note] View.jsp는 단독으로 실행하면 500에러 발생됨. 반드시 목록에서 제목을 클릭해서 실행해야함.

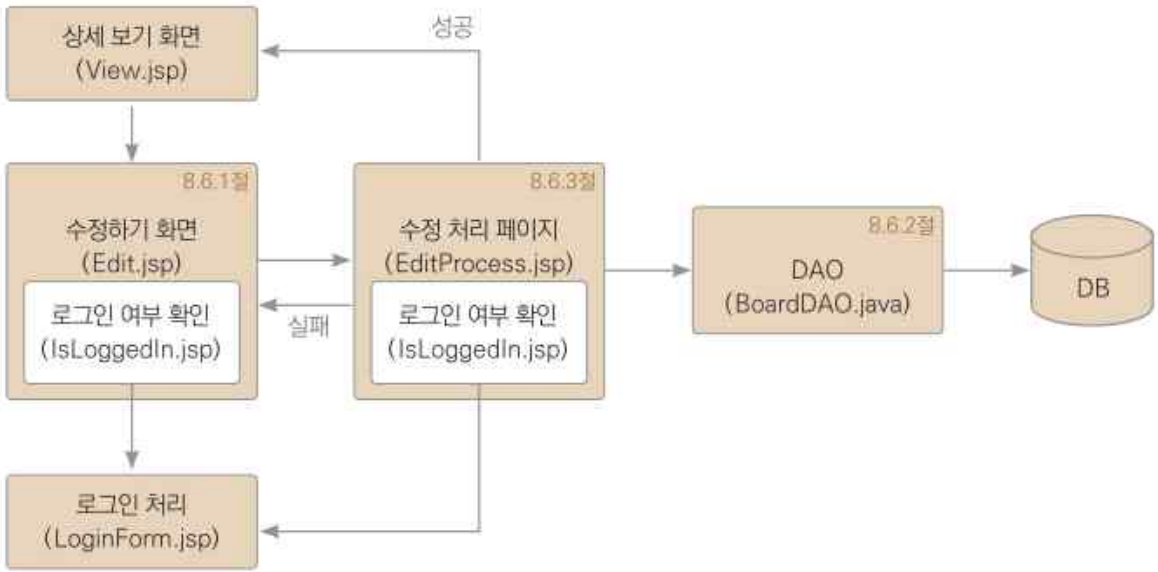


8.6 수정하기

▣ 수정하기

- 수정하기는 상세보기와 글쓰기를 합쳐놓은 형태
- 본인이 작성했던 글을 DB에서 가져와서 글쓰기 폼에 채워서 보여주고
- 내용을 수정해 전송하면 수정한 내용으로 DB를 갱신

▣ 프로세스



8.6 수정하기

▣ 수정 폼 화면 작성

. 기존의 글쓰기 페이지를 복사한 후 몇가지만 수정하면 됨

예제 8-14] webapp/08Board/Edit.jsp

```
<%@ include file="./IsLoggedIn.jsp"%>  ❶
<%
String num = request.getParameter("num");  // 일련번호 받기
BoardDAO dao = new BoardDAO(application);  // DAO 생성
BoardDTO dto = dao.selectView(num);        // 게시물 가져오기
String sessionId = session.getAttribute("UserId").toString();  // 로그인 ID 얻기
if (!sessionId.equals(dto.getId())) {      // 본인인지 확인
    JSFunction.alertBack("작성자 본인만 수정할 수 있습니다.", out);
    return;
}
```

❶ 수정페이지 진입시 로그인 확인

❷ 상세보기와 동일하게 게시물의 내용을 DB로부터 인출

❸ 세션을 통해 본인여부 확인 후 JS의 경고창 및 뒤로이동

8.6 수정하기

▣ 수정 폼 화면 작성(계속)

```
<h2>회원제 게시판 - 수정하기(Edit)</h2>
<form name="writeFrm" method="post" action="EditProcess.jsp"  ❹
      onsubmit="return validateForm(this);">
    <input type="hidden" name="num" value="<%= dto.getNum() %>" />  ❺
    <table border="1" width="90%">
        <tr>
            <td>제목</td>
            <td>
                <input type="text" name="title" style="width: 90%;"
                       value="<%= dto.getTitle() %>"/>  ❻
            </td>
        </tr>
        <tr>
            <td>내용</td>
            <td>
                <textarea name="content" style="width: 90%; height: 100px;">
<%= dto.getContent() %></textarea>  ❼
            </td>
        </tr>
```

❹ action 속성은 EditProcess.jsp로 변경

❺ 수정할 게시물의 일련번호도 함께 전송되야 하므로 hidden 상자 추가

❻ 기존 내용을 value 속성에 삽입

8.6 수정하기

▣ DAO 준비

예제 8-15] Java Resources/model1/board/BoardDAO.java

```
public int updateEdit(BoardDTO dto) {  ❶
    int result = 0;

    try {
        // 쿼리문 템플릿 ❷
        String query = "UPDATE board SET "
                     + " title=?, content=? "
                     + " WHERE num=?";
        // 쿼리문 완성 ❸
        psmt = con.prepareStatement(query);
        psmt.setString(1, dto.getTitle());
        psmt.setString(2, dto.getContent());
        psmt.setString(3, dto.getNum());
        // 쿼리문 실행 ❹
        result = psmt.executeUpdate();
    }
    catch (Exception e) {
```

❷ 제목과 내용을 수정할 수 있는 update 쿼리문 작성

❸ 쿼리문의 인파라미터 설정 및 실행

8.6 수정하기

▣ 수정 처리 페이지 작성

예제 8-16] webapp/08Board/EditProcess.jsp

```
<%@ include file="./IsLoggedIn.jsp"%>  ❶
<%
// 수정 내용 얻기
String num = request.getParameter("num");
String title = request.getParameter("title");
String content = request.getParameter("content");

// DTO에 저장
BoardDTO dto = new BoardDTO();
dto.setNum(num);
dto.setTitle(title);
dto.setContent(content);

// DB에 반영
BoardDAO dao = new BoardDAO(application);
int affected = dao.updateEdit(dto);
dao.close();
```

❶ 로그인 확인
❷ 전송된 파라미터를 받아 DTO 객체에 저장
❸ DAO 객체 생성 후 update를 위한 메서드 호출

8.6 수정하기

▣ 수정 처리 페이지 작성(계속)

```
// 성공/실패 처리
if (affected == 1) {
    // 성공 시 상세 보기 페이지로 이동
    response.sendRedirect("View.jsp?num=" + dto.getNum());  ❹
}
else {
    // 실패 시 이전 페이지로 이동
    JSFunction.alertBack("수정하기에 실패하였습니다.", out);  ❺
}
%>
```

❹ 수정에 성공한 경우 상세보기 페이지로 이동

❺ 실패인 경우 경고창을 띄운 후 뒤로 이동

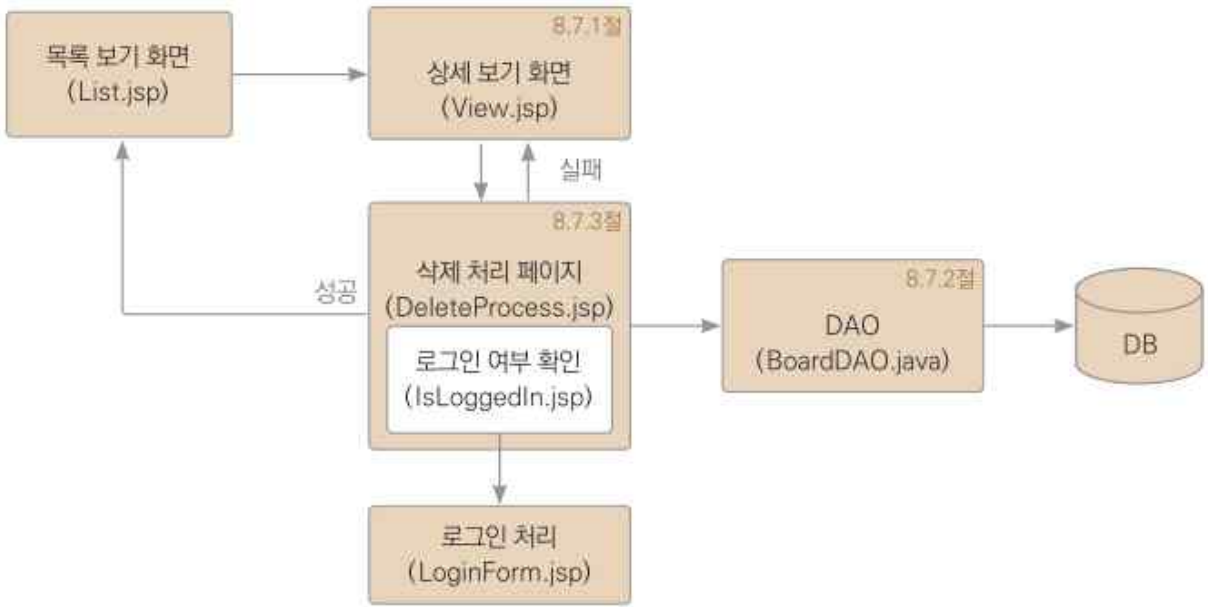
수정성공

8.7 삭제하기

▣ 삭제하기

- 삭제는 쓰기나 수정과는 달리 폼이 필요없음
- 작성자 본인인지 확인되면 즉시 게시물을 삭제하면 됨

▣ 프로세스



8.7 삭제하기

▣ 삭제하기 버튼에 삭제 요청 로직 달기

. 삭제 버튼 클릭시 요청을 보내는 Javascript 코드를 상세보기 페이지에 추가

예제 8-17] webapp/08Board/View.jsp

```
<script>
function deletePost() {  ❶
    var confirmed = confirm("정말로 삭제하겠습니까?");  ❷
    if (confirmed) {
        var form = document.writeFrm;        // 이름(name)이 "writeFrm"인 폼 선택
        form.method = "post";                // 전송 방식 ❸
        form.action = "DeleteProcess.jsp";   // 전송 경로 ❹
        form.submit();                       // 폼값 전송 ❺
    }
}
</script>
```

❶ 삭제하기 버튼을 눌렀을때 호출되는 함수
❷ confirm 대화상자를 띄워 확인 후
❸~❺ 전송방식, 경로를 설정 후 전송

```
<form name="writeFrm">
  <input type="hidden" name="num" value="<%= num %>" />  ❻
... 생략 ...
          <button type="button" onclick="deletePost();">삭제하기</button>  ❼
... 생략 ...
</form>
```

8.7 삭제하기

▣ DAO 준비

예제 8-18] Java Resources/model1/board/BoardDAO.java

```
public int deletePost(BoardDTO dto) {  ❶
    int result = 0;
    try {
        // 쿼리문 템플릿
        String query = "DELETE FROM board WHERE num=?";  ❷

        // 쿼리문 완성
        psmt = con.prepareStatement(query);
        psmt.setString(1, dto.getNum());  ❸

        // 쿼리문 실행
        result = psmt.executeUpdate();  ❹
    }
    catch (Exception e) {
        System.out.println("게시물 삭제 중 예외 발생");
        e.printStackTrace();
    }

    return result; // 결과 반환
}
```

❷ 삭제를 위한 delete 쿼리문 작성
❸ 인파라미터를 설정한 후 실행

8.7 삭제하기

▣ 삭제 처리 페이지 작성

예제 8-19] webapp/08Board/DeleteProcess.jsp

```
<%@ include file="./IsLoggedIn.jsp"%>  ❶
<%
String num = request.getParameter("num");  // 일련번호 얻기 ❷
BoardDTO dto = new BoardDTO();             // DTO 객체 생성
BoardDAO dao = new BoardDAO(application);  // DAO 객체 생성
dto = dao.selectView(num);  // 주어진 일련번호에 해당하는 기존 게시물 얻기 ❸

// 로그인된 사용자 ID 얻기
String sessionId = session.getAttribute("UserId").toString();  ❹

int delResult = 0;

if (sessionId.equals(dto.getId())) {  // 작성자가 본인인지 확인 ❺
    // 작성자가 본인이면...
    dto.setNum(num);
    delResult = dao.deletePost(dto);  // 삭제!!! ❻
    dao.close();
```

❶ 로그인 확인
❷❸ 일련번호로 기존 게시물 인출
❸ 인파라미터를 설정한 후 실행
❹❺ 세션에 저장된 아이디를 읽은 후 게시물의 작성자와 비교
❻ 작성자 본인이 맞으면 게시물 삭제

8.7 삭제하기

▣ 삭제 처리 페이지 작성(계속)

```
    // 성공/실패 처리
    if (delResult == 1) {
        // 성공 시 목록 페이지로 이동 ❼
        JSFunction.alertLocation("삭제되었습니다.", "List.jsp", out);
    } else {
        // 실패 시 이전 페이지로 이동 ❽
        JSFunction.alertBack("삭제에 실패하였습니다.", out);
    }
}
else {
    // 작성자 본인이 아니라면 이전 페이지로 이동 ❾
    JSFunction.alertBack("본인만 삭제할 수 있습니다.", out);

    return;
}
%>
```

❼ 게시물 삭제에 성공하면 목록으로 이동
❽ 삭제에 실패하면 뒤로 이동
❾ 작성자 본인이 아니면 뒤로 이동

학습 마무리

▣ **핵심요약**

- 목록 보기와 상세 보기는 로그인 없이 접근할 수 있음
- 글쓰기는 로그인 후 할 수 있음
- 수정과 삭제는 로그인 후 본인이 작성한 게시물에 한해서만 가능
- 로그인 시 session 내장 객체와 session 영역을 사용
- DB연결을 위한 설정값은 web.xml에 주로 저장
- 사용자가 입력한 내용을 쿼리에 반영하기 위해 동적 쿼리를 뜻하는 PreparedStatement 인터페이스를 주로 사용
- 경고창(alert), 페이지 이동(location.href)과 같이 자주 사용하는 자바스크립트는 별도의 유틸리티 클래스로 만들어두면 재사용성이 높아짐