

JSP 게시판 만들기

1. 모델1 기반의 게시판 구현하기

1) 게시판 CRUD(Create, Retrieve(Read), Update, Delete) 구현

게시판의 가장 기본적인 CRUD 기능을 모델1 방식으로 구현하면서 JSP를 활용해 웹 애플리케이션을 구현하는 방법에 대해서 알아보자. 이번에 구현하는 게시판의 화면(View)을 구현할 때 부트스트랩(Bootstrap)이라는 웹 사이트 개발 프레임워크(Front-end Framework)를 사용할 것이다. 부트스트랩은 트위터에서 사용하는 각종 레이아웃, 버튼, 입력창 등의 디자인을 CSS와 Javascript로 만들어 오픈소스로 공개한 것이다. 부트스트랩은 HTML, CSS 기반의 템플릿 양식, 버튼, 내비게이션 및 웹 페이지를 구성하는 다양한 요소 등을 포함하고 있어서 버튼, 아이콘, 목록, 타이포그래피, 등과 더불어 탭, 드롭다운 메뉴, 메뉴 바, 페이지 이동 바, 툴팁, 썸네일 등과 같은 웹 페이지에서 많이 사용되는 구성 요소를 거의 대부분 내장하고 있어서 웹 페이지를 구현할 때 편리하게 사용할 수 있다.

1-1) DB TABLE 생성 및 설정

▶ 프로젝트 생성 및 환경설정

- 실습용 프로젝트 : JSPClassBBS01
- 완성 프로젝트 : JSPStudyBBS01

▶ 게시판 테이블 생성

/WEB-INF/sql/JSPStudyBBS.sql 파일을 참고해 데이터를 테이블에 추가

```
-- 게시글 번호, 제목, 내용, 글쓴이, 날짜, 조회수, 비밀번호, 파일정보,  
-- no, title, writer, content, reg_date, read_count, pass, file1
```

```
CREATE TABLE jspbbs(  
  no NUMBER PRIMARY KEY,  
  title VARCHAR2(50 CHAR) NOT NULL,  
  writer VARCHAR2(20 CHAR) NOT NULL,  
  content VARCHAR2(1000 CHAR) NOT NULL,  
  reg_date TIMESTAMP NOT NULL,  
  read_count NUMBER(5) NOT NULL,  
  pass VARCHAR2(20 CHAR) NOT NULL,  
  file1 VARCHAR2(100 CHAR)  
);
```

```
CREATE SEQUENCE jspbbs_seq;
```

▶ DBCP(Database Connection Pool) 설정

- webapp/META-INF/context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
  <Resource
    name="jdbc/bbsDBPool"
    auth="Container"
    type="javax.sql.DataSource"
    driverClassName="oracle.jdbc.driver.OracleDriver"
    url="jdbc:oracle:thin:@localhost:1521:xe"
    username="hr"
    password="hr"
    factory="org.apache.commons.dbcp2.BasicDataSourceFactory"
    maxActive="10"
    maxIdle="5" />
</Context>
```

▶ 백엔드 라이브러리 추가

* webapp/WEB-INF/lib/ 폴더에 아래의 라이브러리를 다운로드 받아서 추가

- Oracle 접속 드라이버

ojdbc6.jar

- commons dbcp2 라이브러리

commons-dbcp2-2.8.0.jar, commons-pool2-2.10.0.jar, commons-logging-1.2.jar

- 표준 태그(Java Standard Tag Library - JSTL) 라이브러리

taglibs-standard-impl-1.2.5.jar, taglibs-standard-spec-1.2.5.jar

▶ 프론트엔드 라이브러리 추가

- 부트스트랩 : webapp/bootstrap 폴더에 추가

bootstrap.bundle.min.js, bootstrap.min.css

- jQuery : webapp/js 폴더에 추가

jquery-3.3.1.min.js

1-2) VO(Value Object) 클래스

먼저 com.jspstudy.bbs.vo 패키지를 만들고 jspbbs 테이블에서 읽어온 한 행의 데이터 즉 하나의 게시 글 정보를 저장하는 다음과 같은 Board 클래스를 작성하자.

- com.jspstudy.bbs.vo

/* 하나의 게시 글 정보를 저장하는 VO(Value Object)

* VO 객체에 저장될 데이터는 테이블에서 읽어오기 때문에 각각의 변수는
* 테이블에서 컬럼이 가지는 데이터 형식과 같거나 자동 형 변환이 가능해야 한다.
**/

```
public class Board {  
    // no, title, writer, content, reg_date, read_count, pass, file1  
    private int no;  
    private String title;  
    private String writer;  
    private String content;  
    private Timestamp regDate;  
    private int readCount;  
    private String pass;  
    private String file1;  
  
    public Board() { }  
    public Board(int no, String title, String writer, String content,  
        Timestamp regDate, int readCount, String pass, String file1) {  
        this.no = no;  
        this.title = title;  
        this.content = content;  
        this.writer = writer;  
        this.regDate = regDate;  
        this.readCount = readCount;  
        this.pass = pass;  
        this.file1 = file1;  
    }  
  
    public int getNo() {  
        return no;  
    }  
    public void setNo(int no) {  
        this.no = no;  
    }  
    public String getTitle() {  
        return title;  
    }  
    public void setTitle(String title) {  
        this.title = title;  
    }  
    public String getWriter() {  
        return writer;  
    }  
    public void setWriter(String writer) {  
        this.writer = writer;  
    }  
}
```

```

public String getContent() {
    return content;
}
public void setContent(String content) {
    this.content = content;
}
public Timestamp getRegDate() {
    return regDate;
}
public void setRegDate(Timestamp regDate) {
    this.regDate = regDate;
}
public int getReadCount() {
    return readCount;
}
public void setReadCount(int readCount) {
    this.readCount = readCount;
}
public String getPass() {
    return pass;
}
public void setPass(String pass) {
    this.pass = pass;
}
public String getFile1() {
    return file1;
}
public void setFile1(String file1) {
    this.file1 = file1;
}
}

```

1-3) 게시 글 리스트

▶ DAO(Data Access Object) 클래스

프로젝트에 com.jspstudy.bbs.dao 패키지를 만들고 다음과 같이 BoardDao 클래스를 생성하여 게시 글 리스트를 DB에서 읽어오는 메서드를 추가하자.

- com.jspstudy.bbs.dao.BoardDao

// DBCP를 활용한 DAO(Data Access Object) 클래스

```
public class BoardDao {
```

```
    // 데이터베이스 작업에 필요한 객체 타입으로 변수를 선언
```

```
    // Connection 객체는 DB에 연결해 작업을 수행할 수 있도록 지원하는 객체
```

```

private Connection conn;

// Statement, PreparedStatement 객체는 DB에 쿼리를 발행하는 객체
private PreparedStatement pstmt;

// ResultSet 객체는 DB에 SELECT 쿼리를 발행한 결과를 저장하는 객체
private ResultSet rs;

/* DataSource 객체는 데이터 원본과 연결할 수 있도록 지원하는 객체
 * JNDI 방식으로 DBCP를 찾아 DBCP에서 Connection 객체를 대여하는 객체
 */
private static DataSource ds;

/* 기본 생성자가 호출될 때 마다 ConnectionPool에 접근해 Connection 객체를
 * 얻어 올 수 있는 DataSource 객체를 생성한다.
 */
public BoardDao() {
    try {
        /* 1. 자바 네이밍 서비스를 사용하기 위해
         * javax.naming 패키지의 InitialContext 객체를 생성한다.
         */
        Context initContext = new InitialContext();

        /* 2. InitialContext 객체를 이용해 디렉토리 서비스에서 필요한 객체를
         * 찾기 위해 기본 네임스페이스를 인자로 지정해 Context 객체를 얻는다.
         *
         * 디렉토리 서비스에서 필요한 객체를 찾기 위한 일종의 URL 개념으로
         * 디렉토리 방식을 사용하므로 java:comp/env와 같이 지정한다.
         */
        Context envContext = (Context) initContext.lookup("java:/comp/env");

        /* 3. "jdbc/bbsDBPool"이라는 이름을 가진 DBCP에 접근하기 위한
         * DataSource 객체를 얻는다.
         *
         * context.xml 파일에서 지정한 수의 커넥션을 생성해 커넥션 풀에 저장한다.
         * "java:/comp/env"는 JNDI에서 기본적으로 사용하는 네임스페이스 이고
         * "jdbc/bbsDBPool"은 DBCP 이름으로 임의로 지정하여 사용할 수 있다.
         *
         * BoardDao 클래스가 사용되면 클래스 정보가 메모리에 로딩되면서
         * new 연산자에 의해서 BoardDao 클래스의 인스턴스가 생성된다.
         * 그리고 이 생성자가 호출되면서 DBCP를 찾는 코드가 실행되고 DBCP에
         * 접근해 Connection 객체를 대여할 수 있는 DataSource 객체를 구한다.
         */
        ds = (DataSource) envContext.lookup("jdbc/bbsDBPool");
    }
}

```

```

    } catch(NamingException e) {
        System.out.println("BoardDao() : NamingException");
        e.printStackTrace();
    }
}

```

// DB에 등록된 전체 게시 글을 읽어와 ArrayList로 반환하는 메서드

```

public ArrayList<Board> boardList() {

    String sqlBoardList = "SELECT * FROM jspbbs ORDER BY no DESC";
    ArrayList<Board> boardList = null;

    try {
        // 4. DataSource 객체를 이용해 커넥션을 대여한다.
        conn = ds.getConnection();

        /* 5. DBMS에 SQL 쿼리를 발생하기 위해 활성화된
         * Connection 객체로 부터 PreparedStatement 객체를 얻는다.
         *
         * PreparedStatement는 SQL 명령을 캐싱하여(저장하여) 반복적으로 사용하기
         * 때문에 preparedStatement()를 호출할 때 SQL 쿼리 문을 지정해야 한다.
         */
        pstmt = conn.prepareStatement(sqlBoardList);

        /* 6. PreparedStatement를 이용해 DB에 SELECT 쿼리를 발행하고
         * 그 결과로 ResultSet을 얻는다.
         *
         * executeQuery()는 실제 DBMS에 SELECT 쿼리를 발행하는 메소드로
         * DB에서 검색된 데이터를 가상의 테이블 형태인 ResultSet 객체로 반환한다.
         */
        rs = pstmt.executeQuery();

        // 게시 글 리스트를 저장할 ArrayList 객체 생성
        boardList = new ArrayList<Board>();

        /* 7. 쿼리 실행 결과를 바탕으로 while문 안에서 하나의 게시 글을 저장할
         * Board 객체를 생성하고 이 객체에 하나의 게시 글 정보를 저장하고
         * Board 객체를 ArrayList에 저장한다.
         *
         * ResultSet 객체는 DB로 부터 읽어온 데이터에 접근하기 위해 테이블의
         * 행을 가리키는 cursor를 제공한다. 맨 처음 ResultSet 객체를 반환
         * 받으면 cursor는 첫 번째 행 바로 이전을 가리키고 있다. ResultSet의
         * cursor가 맨 마지막 행에 도달하면 while문을 빠져 나온다.
         *
         */
    }
}

```

```

* ResultSet에는 다양한 데이터 타입에 대응하는 getter 메소드를
* 지원하고 있으며 SELECT 문장에서 지정한 컬럼의 index 또는
* 컬럼명으로 테이블의 필드 값을 가져올 수 있도록 getXxx() 메소드가
* 오버로딩 되어 있어 index와 컬럼명 둘 다 사용이 가능하다.
* 여기에 지정하는 index는 배열에서 사용되는 index의 개념이 아니라
* 첫 번째 컬럼, 두 번째 컬럼과 같이 위치의 개념으로 1부터 시작된다.
**/
while(rs.next()) {

    /* 반복문을 돌 때마다 Board 객체를 생성해 DB로부터 읽어온 한 행의
    * 데이터를 읽어 Board 객체에 저장하고 다시 ArrayList에 담는다.
    */
    Board b = new Board();

    /* ResultSet 객체의 getXXX() 메서드에 컬럼 위치에 대한 index 값을
    * 1부터 지정할 수도 있고 컬럼 이름을 지정해 데이터를 읽어 올 수 있다.
    */
    b.setNo(rs.getInt("no"));
    b.setTitle(rs.getString("title"));
    b.setWriter(rs.getString("writer"));
    b.setContent(rs.getString("content"));
    b.setRegDate(rs.getTimestamp("reg_date"));
    b.setReadCount(rs.getInt("read_count"));
    b.setPass(rs.getString("pass"));
    b.setFile1(rs.getString("file1"));

    boardList.add(b);
}
} catch(SQLException e) {
    System.out.println("BoardDao - boardList() - SQLException");
    e.printStackTrace();
} finally {
    try {
        // 8. 사용한 ResultSet과 PreparedStatement를 종료한다.
        if(rs != null) rs.close();
        if(pstmt != null) pstmt.close();

        // 9. 커넥션 풀로 Connection 객체를 반납한다.
        if(conn != null) conn.close();
    } catch(SQLException e) {}
}

// 10. 데이터베이스로 부터 읽어온 게시 글 리스트를 반환한다.
return boardList;

```



```

    } // end boardList();
}

```

▶ 여러 페이지에서 중복되는 header와 footer 페이지

웹 페이지에서 공통적으로 표현되는 부분은 여러 페이지에 걸쳐서 코드가 중복되는데 이렇게 중복되는 내용을 별도의 JSP 페이지로 작성하고 별도로 작성한 JSP페이지의 내용이 필요한 곳에서 include 지시자를 사용하면 중복 코드를 최소화 할 수 있다.

- webapp/pages/header.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!-- header -->
<div class="row border-bottom border-primary">
    <div class="col-4">
        <p></p>
    </div>
    <div class="col-8">
        <div class="row">
            <div class="col">
                <ul class="nav justify-content-end">
                    <li class="nav-item">
                        <a class="nav-link" href="#">로그인-폼</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="#">로그인-모달</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="boardList.mvc">게시 글 리스트</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="#">회원가입</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="#">주문/배송조회</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="#">고객센터</a>
                    </li>
                </ul>
            </div>
        </div>
    </div>
    <div class="row">

```

```

        <div class="col text-end">로그인시 인사말 출력</div>
    </div>
</div>
</div>

```

- webapp/pages/footer.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!-- footer -->
<div class="row border-top border-primary my-5" id="global-footer">
    <div class="col text-center py-3">
        <p>고객상담 전화주문:1234-5678 사업자등록번호 :111-11-123456
        대표이사: 홍길동 통신판매업 서울 제 000000호<br/>
        개인정보관리책임자:임격정 분쟁조정기관표시 : 소비자보호원, 전자거래분쟁중재위원회<br/>

        Copyright (c) 2023 JSP2U Corp. All right Reserved.
    </p>
    </div>
</div>

```

▶ 게시 글 리스트 요청을 처리하는 JSP

- webapp/board/boardList.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="com.jspstudy.bbs.dao.*, com.jspstudy.bbs.vo.*" %>
<%@ page import="java.util.*" %>
<%--
    JSP 표준 태그 라이브러리(JSTL)를 사용하기 위한 taglib 지시자
    http://jakarta.apache.org, http://tomcat.apache.org 에서
    다운로드 하여 WEB-INF/lib 폴더에 추가해야 표준 태그를 사용할 수 있다.
    이 페이지에서는 JSTL의 코어 라이브러리에 속한 <c:set> 태그를 사용하여
    pageContext, request, session, application 4개의 영역에
    속성으로 데이터를 저장하고 EL 식을 이용해 출력하는 기법을 소개하고 있다.
    JSTL의 코어 라이브러리는 말 그대로 JSTL의 가장 핵심적인 기능을 제공하는
    라이브러리로 프로그래밍 언어에서 일반적으로 제공하고 있는 변수 선언, 조건문,
    반복문에 해당하는 태그를 지원한다. 또한 익셉션, URL 저장, 데이터 출력과 관련된
    태그와 다른 JSP 페이지 호출(import, redirect)과 관련된 태그를 지원한다.
--%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<%
    // BoardDao 객체를 생성하고 게시 글 리스트를 읽어온다.
    BoardDao dao = new BoardDao();

```

```
ArrayList<Board> bList = dao.boardList();
```

```
%>
```

```
<%--
```

JSP 페이지에서 사용할 변수를 선언하고 초기 값을 설정하고 있다.

<c:set> 태그로 변수를 선언할 때는 변수의 타입은 지정하지 않으며 var 속성에 변수의 이름을 지정하고 value 속성에 변수의 초기 값을 필히 지정해야 한다.

scope 속성에는 page, request, session, application 중 하나를

지정할 수 있으며 생략 가능하다. 생략하게 되면 기본 값은 page로 지정된다.

scope 속성을 지정하는 것에서 알 수 있듯이 여기에 선언한 변수는 EL식 안에서 사용할 수 있고 스크립팅 요소에서는 사용할 수 없다. 즉 스크립팅 요소에서 선언한 자바 코드의 변수가 되는 것이 아니라 위의 4개 영역에 속성으로 저장하는 방식인 setAttribute()가 호출되어 scope에 지정한 영역에 속성으로 저장된다.

```
--%>
```

```
<!-- 아래에서 EL로 접근하기 위해서 pageContext 영역의 속성으로 저장 -->
```

```
<c:set var="bList" value="%= bList %>" />
```

```
<!DOCTYPE html>
```

```
<html lang="ko">
```

```
<head>
```

```
<meta charset="utf-8">
```

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

```
<title>게시 글 리스트</title>
```

```
<link href="../bootstrap/bootstrap.min.css" rel="stylesheet" >
```

```
<script src="../js/jquery-3.3.1.min.js"></script>
```

```
<script src="../js/formcheck.js"></script>
```

```
</head>
```

```
<body>
```

```
<div class="container">
```

```
<!--
```

여러 페이지에 걸쳐서 중복되는 내용을 별도의 JSP 페이지로 만들고 include 지시자를 이용해 현재 JSP 페이지에 포함 시킬 수 있다. include 지시자를 이용해 외부의 JSP 페이지를 포함 시키면 컴파일 타임에 현재 JSP 페이지에 하나로 합쳐져서 컴파일 된다. 중복되는 내용을 별도의 JSP 페이지로 만드는 것을 페이지 모듈화라고 한다.

```
-->
```

```
<%@ include file="../pages/header.jsp" %>
```

```
<!-- content -->
```

```
<div class="row my-5 text-center">
```

```
<div class="col">
```

```
<h2 class="fs-3 fw-bold">게시 글 리스트</h2>
```

```
</div>
```

```
</div>
```

```
<form name="searchForm" id="searchForm" action="#"
```

```
class="row justify-content-center my-3">
```

```
<div class="col-auto">
```

```
<select name="type" class="form-select">
```

```
<option value="title">제목</option>
```

```

        <option value="writer">작성자</option>
        <option value="content">내용</option>
    </select>
</div>
<div class="col-4">
    <input type="text" name="keyword" class="form-control"/>
</div>
<div class="col-auto">
    <input type="submit" value="검 색" class="btn btn-primary"/>
</div>
</form>
<div class="row">
    <div class="col text-end">
        <a href="writeForm.jsp" class="btn btn-outline-success">글쓰기</a>
    </div>
</div>
<div class="row my-3">
    <div class="col">
        <table class="table table-hover">
            <thead>
                <tr class="table-dark">
                    <th>NO</th>
                    <th>제목</th>
                    <th>작성자</th>
                    <th>작성일</th>
                    <th>조회수</th>
                </tr>
            </thead>
            <tbody class="text-secondary">
<%--

```

표현언어(EL)를 사용해 내장객체의 속성에 저장할 때 사용한 속성 이름을 지정하면 내장객체의 속성에 저장된 데이터를 읽어 올 수 있다. 스크립팅요소를 사용하는 것에 비해 더 간단히 내장객체의 속성에 저장된 값을 읽을 수 있다.

표현언어(EL)를 사용해 속성 이름을 지정하면 `pageContext`, `request`, `session`, `application` 4개의 영역에 저장된 속성을 작은 범위에서 큰 범위 순으로 검색하여 지정한 이름의 속성에 대한 값을 얻어 올 수 있다. 지정한 속성 이름이 존재하지 않아도 `NullPointerException`은 발생하지 않고 다만 아무것도 출력되지 않는다.

내장객체의 속성에 객체가 저장되어 있으면 내장객체의 속성 명과 dot 연자자(.)를 사용해 객체의 프로퍼티(인스턴스 변수) 값을 읽어 올 수 있다. 표현언어(EL)로 객체의 프로퍼티를 읽기 위해서는 그 객체의 클래스에 읽어 올 프로퍼티에 대한 getter 메서드가 반드시 존재해야 한다. 그렇지 않으면 `JasperException`이 발생한다.

```
--%>
```

```
<%--
```

게시 글이 있는 경우 게시 글 수만큼 반복하면서 게시 글을 출력

```
--%>
```

```

<c:if test="${ not empty bList }">
  <c:forEach var="b" items="${bList}">
    <tr>
      <td>${ b.no }</td>
      <%--
        반복문 안에서 한 행의 게시 글을 출력하면서
        게시 글 상세보기로 넘어갈 수 있도록 링크를 설정
      --%>
      <td><a href="boardDetail.jsp?no=${b.no}" class="text-decoration-none
link-secondary">${ b.title }</a></td>
      <td>${ b.writer }</td>
      <td>${ b.regDate }</td>
      <td>${ b.readCount }</td>
    </tr>
  </c:forEach>
</c:if>
<!-- 게시 글이 없는 경우 -->
<c:if test="${ empty bList }">
  <tr>
    <td colspan="5" class="text-center">게시 글이 존재하지 않습니다.</td>
  </tr>
</c:if>
</tbody>
</table>
</div>
</div>
<%@ include file="../pages/footer.jsp" %>
</div>
<script src="../bootstrap/bootstrap.bundle.min.js"></script>
</body>
</html>

```

1-4) 게시 글 상세 보기

▶ DAO(Data Access Object) 클래스에 메서드 추가

BoardDao 클래스에 하나의 게시 글 정보를 DB로부터 읽어오는 다음 메서드를 추가한다.

- com.jspstudy.bbs.dao.BoardDao

/* 게시 글 상세보기 요청 시 호출되는 메서드

* no에 해당하는 게시 글 을 DB에서 읽어와 Board 객체로 반환하는 메서드

*/

```
public Board getBoard(int no) {
```

```
String sqlBoard = "SELECT * FROM jspbbs WHERE no=?";
```

```
Board board = null;
```

```
try {
```

```
    // 4. DataSource 객체를 이용해 커넥션을 대여한다.
```

```
    conn = ds.getConnection();
```

```
    /* 5. DBMS에 SQL 쿼리를 발생하기 위해 활성화된
```

```
    * Connection 객체로 부터 PreparedStatement 객체를 얻는다.
```

```
    *
```

```
    * PreparedStatement는 SQL 명령을 캐싱하여(저장하여) 반복적으로 사용하기
```

```
    * 때문에 preparedStatement()를 호출할 때 SQL 쿼리 문을 지정해야 한다.
```

```
    * PreparedStatement 객체는 반복적으로 사용되는 SQL 명령 외에
```

```
    * SQL 문장에 포함되어 변경되는 데이터를 Placeholder(?)로 지정할 수 있다.
```

```
    * SQL 문장의 조건식에 사용되는 데이터나 실제 테이블에 입력되는 데이터는
```

```
    * SQL 명령이 동일하나 상황에 따라 조건식에 사용되는 데이터가 변경될 수
```

```
    * 있고 또한 테이블에 새롭게 추가되는 데이터가 다르게 입력된다. 이렇게
```

```
    * 변경되는 데이터가 SQL 문장에서 기술되어야 할 위치에 Placeholder(?)를
```

```
    * 지정하고 실제 DB에 SQL 쿼리 문을 전송하기 전에 PreparedStatement
```

```
    * 객체의 setXxx()를 이용해 데이터를 변경해 가며 질의 할 수 있다.
```

```
    * 여기서 주의할 점은 setXxx()에 지정하는 index의 개념이 배열에서
```

```
    * 사용되는 index의 개념이 아니라 첫 번째 Placeholder(?), 두 번째
```

```
    * Placeholder(?)와 같이 위치의 개념으로 시작 번호가 1부터 시작된다.
```

```
    **/
```

```
    pstmt = conn.prepareStatement(sqlBoard);
```

```
    /* 6. PreparedStatement 객체에 SELECT 쿼리의 Placeholder(?)와
```

```
    * 데이터를 맵핑 한다.
```

```
    **/
```

```
    pstmt.setInt(1, no);
```

```
    // 7. 데이터베이스에 SELECT 쿼리를 발행하고 ResultSet 객체로 받는다.
```

```
    rs = pstmt.executeQuery();
```

```
    /* 8. Board 객체를 생성하고 ResultSet으로 부터 데이터를 읽어
```

```
    * Board 객체의 각 프로퍼티에 값을 설정한다.
```

```
    * no가 Primary Key 이므로 no에 해당하는 게시 글을 DB에서 읽으면
```

```
    * 없거나 또는 하나의 게시 글 정보를 얻을 수 있으므로 if문을 사용했다.
```

```
    *
```

```
    * ResultSet 객체는 DB로 부터 읽어온 데이터에 접근하기 위해 테이블의
```

```
    * 행을 가리키는 cursor를 제공한다. 맨 처음 ResultSet 객체를 반환
```

```
    * 받으면 cursor는 첫 번째 행 바로 이전을 가리키고 있다. ResultSet의
```

```
    * cursor가 맨 마지막 행에 도달하면 while 문을 빠져 나온다.
```

```
    * 여기서는 결과가 많아야 한 행이므로 한 번 실행되고 if문을 빠져 나온다.
```

```
    *
```

```
    * ResultSet에는 다양한 데이터 타입에 대응하는 getter 메소드를
```

```

* 지원하고 있으며 SELECT 문장에서 지정한 컬럼의 index 또는
* 컬럼명으로 테이블의 필드 값을 가져올 수 있도록 getXxx() 메소드가
* 오버로딩 되어 있어 index와 컬럼명 둘 다 사용이 가능하다.
* 여기에 지정하는 index는 배열에서 사용되는 index의 개념이 아니라
* 첫 번째 컬럼, 두 번째 컬럼과 같이 위치의 개념으로 1부터 시작된다.
**/
if(rs.next()) {
    board = new Board();

    /* ResultSet 객체의 getXXX() 메서드에 컬럼 위치에 대한 index 값을
    * 1부터 지정할 수도 있고 컬럼 이름을 지정해 데이터를 읽어 올 수 있다.
    */
    board.setNo(rs.getInt("no"));
    board.setTitle(rs.getString("title"));
    board.setWriter(rs.getString("writer"));
    board.setContent(rs.getString("content"));
    board.setRegDate(rs.getTimestamp("reg_date"));
    board.setReadCount(rs.getInt("read_count"));
    board.setPass(rs.getString("pass"));
    board.setFile1(rs.getString("file1"));
}
} catch(SQLException e) {
    System.out.println("BoardDao - getBoard() : SQLException");
    e.printStackTrace();

} finally {
    try {
        // 9. 사용한 ResultSet과 PreparedStatement 객체를 닫는다.
        if(rs != null) rs.close();
        if(pstmt != null) pstmt.close();

        // 10. 커넥션 풀로 Connection 객체를 반납한다.
        if(conn != null) conn.close();
    } catch(SQLException e) {}
}

// 11. 데이터베이스로 부터 읽어온 no에 해당하는 게시 글 정보를 반환한다.
return board;

} // end getBoard(int no);

```

▶ 게시 글 상세보기 요청을 처리하는 JSP

- webapp/board/boardDetail.jsp

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

```

    pageEncoding="UTF-8"%>
<%@ page import="com.jspstudy.bbs.vo.*, com.jspstudy.bbs.dao.*" %>
<%--
JSP 표준 태그 라이브러리(JSTL)를 사용하기 위한 taglib 지시자
http://jakarta.apache.org, http://tomcat.apache.org 에서
다운로드 하여 WEB-INF/lib 폴더에 추가해야 표준 태그를 사용할 수 있다.
이 페이지에서는 JSTL의 코어 라이브러리에 속한 <c:set> 태그를 사용하여
pageContext, request, session, application 4개의 영역에
속성으로 데이터를 저장하고 EL 식을 이용해 출력하는 기법을 소개하고 있다.
JSTL의 코어 라이브러리는 말 그대로 JSTL의 가장 핵심적인 기능을 제공하는
라이브러리로 프로그래밍 언어에서 일반적으로 제공하고 있는 변수 선언, 조건문,
반복문에 해당하는 태그를 지원한다. 또한 익셉션, URL 저장, 데이터 출력과 관련된
태그와 다른 JSP 페이지 호출(import, redirect)과 관련된 태그를 지원한다.
--%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%
/* 테이블에서 하나의 게시 글 정보를 읽어오기 위해서 boardList.jsp에서
* 게시 글 상세보기 요청을 하면서 테이블에서 게시 글 하나를 유일하게 구분할 수 있는
* 게시 글 번호를 요청 파라미터로 보냈기 때문에 이 게시 글 번호를 요청 파라미터로
* 부터 읽어 BoardDao를 통해서 no에 해당하는 게시 글 정보를 읽을 수 있다.
*
* 아래에서 no라는 요청 파라미터가 없다면 NumberFormatException 발생
**/
String no = request.getParameter("no");

// BoardDao 객체 구하고 게시 글 번호(no)에 해당하는 게시 글을 읽어온다.
BoardDao dao = new BoardDao();
Board board = dao.getBoard(Integer.valueOf(no));
%>
<%--
JSP 페이지에서 사용할 변수를 선언하고 초기 값을 설정하고 있다.
<c:set> 태그로 변수를 선언할 때는 변수의 타입은 지정하지 않으며 var 속성에
변수의 이름을 지정하고 value 속성에 변수의 초기 값을 필히 지정해야 한다.
scope 속성에는 page, request, session, application 중 하나를
지정할 수 있으며 생략 가능하다. 생략하게 되면 기본 값은 page로 지정된다.
scope 속성을 지정하는 것에서 알 수 있듯이 여기에 선언한 변수는 EL식 안에서
사용할 수 있고 스크립팅 요소에서는 사용할 수 없다. 즉 스크립팅 요소에서 선언한
자바 코드의 변수가 되는 것이 아니라 위의 4개 영역에 속성으로 저장하는 방식인
setAttribute()가 호출되어 scope에 지정한 영역에 속성으로 저장된다.

아래는 pageContext.setAttribute("board", board)와 동일한 코드이다.
--%>
<c:set var="board" value="<%= board %>" scope="page" />
<!DOCTYPE html>

```



```

<html lang="ko">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>게시 글 상세보기</title>
  <link href="../bootstrap/bootstrap.min.css" rel="stylesheet" >
  <script src="../js/jquery-3.3.1.min.js"></script>
<script src="../js/formcheck.js"></script>
</head>
<body>
  <div class="container">
    <%@ include file="../pages/header.jsp" %>
    <!-- content -->
    <div class="row my-5" id="global-content">
      <div class="col">
        <div class="row text-center">
          <div class="col">
            <h2 class="fs-3 fw-bold">게시 글 상세보기</h2>
          </div>
        </div>
        <div class="row my-3">
          <div class="col">
            <table class="table table-bordered" >
              <tbody>
                <tr>
                  <th class="table-secondary">제 목</th>
                  <td colspan="3">${ board.title }</td>
                </tr>
                <tr>
                  <th>글쓴이</th>
                  <td>${ board.writer }</td>
                  <th>작성일</th>
                  <td><fmt:formatDate value="${ board.regDate }"
                    pattern="yyyy-MM-dd HH:mm:ss" /></td>
                </tr>
                <tr>
                  <th>비밀번호</th>
                  <td>
                    <div class="col-sm-8">
                      <input class="form-control" type="password" name="pass" id="pass">
                    </div>
                  </td>
                  <th>조회수</th>
                  <td>${ board.readCount }</td>
                </tr>
              </tbody>
            </table>
          </div>
        </div>
      </div>
    </div>
  </body>
</html>

```

```

        <tr>
            <th>파일</th>
            <td colspan="3">
                <c:if test="${ empty board.file1 }">
                    첨부파일 없음
                </c:if>
                <c:if test="${ not empty board.file1 }">
                    <a href="../../upload/${ board.file1 }">파일 다운로드</a>
                </c:if>
            </td>
        </tr>
        <tr>
            <td colspan="4">
                <pre>${ board.content }</pre>
            </td>
        </tr>
    </tbody>
</table>
</div>
</div>
<div class="row my-3">
    <div class="col text-center">
        <input class="btn btn-warning" type="button" id="detailUpdate" value="수정하기"
"/>
        &nbsp;&nbsp;<input class="btn btn-danger" type="button" id="detailDelete"
value="삭제하기" />
        &nbsp;&nbsp;<input class="btn btn-primary" type="button" value="목록보기"
onclick="location.href='boardList.jsp'"/>
    </div>
</div>
</div>
</div>
<%@ include file="../../pages/footer.jsp" %>
</div>
<script src="../../bootstrap/bootstrap.bundle.min.js"></script>
</body>
</html>

```

1-5) 게시 글쓰기

▶ 게시 글쓰기 폼 요청을 처리하는 JSP

- webapp/board/writeForm.jsp

```

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

```

```

<!DOCTYPE html>
<html lang="ko">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>게시 글쓰기</title>
    <link href="../bootstrap/bootstrap.min.css" rel="stylesheet" >
    <script src="../js/jquery-3.3.1.min.js"></script>
    <script src="../js/formcheck.js"></script>
  </head>
  <body>
    <div class="container">
      <%@ include file="../pages/header.jsp" %>
      <!-- content -->
      <div class="row my-5" id="global-content">
        <div class="col">
          <div class="row text-center">
            <div class="col">
              <h2 class="fs-3 fw-bold">게시 글쓰기</h2>
            </div>
          </div>
        </div>
        <form name="writeForm" action="writeProcess.jsp" id="writeForm"
          class="row g-3 border-primary" method="post">
          <div class="col-4 offset-md-2">
            <label for="writer" class="form-label">글쓴이</label>
            <input type="text" class="form-control" name="writer" id="writer"
              placeholder="작성자를 입력해 주세요">
          </div>
          <div class="col-4 ">
            <label for="pass" class="form-label">비밀번호</label>
            <input type="password" class="form-control" name="pass" id="pass" >
          </div>
          <div class="col-8 offset-md-2">
            <label for="title" class="form-label">제 목</label>
            <input type="text" class="form-control" name="title" id="title" >
          </div>
          <div class="col-8 offset-md-2">
            <label for="content" class="form-label">내 용</label>
            <textarea class="form-control" name="content" id="content"
              rows="10"></textarea>
          </div>
          <div class="col-8 offset-md-2 text-center mt-5">
            <input type="submit" value="등록하기" class="btn btn-primary"/>
            &nbsp;&nbsp;&nbsp;<input type="button" value="목록보기"
              onclick="location.href='boardList.jsp'" class="btn btn-primary"/>
          </div>
        </form>
      </div>
    </div>
  </body>
</html>

```

```

        </div>
    </form>
</div>
</div>
<%@ include file="../pages/footer.jsp" %>
</div>
<script src="../bootstrap/bootstrap.bundle.min.js"></script>
</body>
</html>

```

▶ 게시 글쓰기 폼 유효성 검사 스크립트

게시 글쓰기 폼에서 submit 버튼인 등록하기 버튼이 클릭되면 폼이 submit 되는데 폼이 서버로 전송되기 직전에 게시 글쓰기 폼 안에 있는 각각의 폼 컨트롤에 데이터가 제대로 입력되었는지 체크해서 데이터가 입력되지 않았으면 폼이 전송되지 않도록 해야 불필요한 요청으로 인한 서버의 부하를 막을 수 있다. 이렇게 폼에 입력되어야 할 데이터가 제대로 입력되었는지 사전에 체크하는 것을 폼 유효성 검사라고 한다. webapp/js/ 디렉터리에 다음과 같이 formCheck.js 파일을 만들고 아래 코드를 참고해서 등록하기 버튼이 클릭되어 폼이 submit 될 때 폼 유효성 검사를 할 수 있도록 자바스크립트 코드를 추가하자.

- webapp/js/formcheck.js

`$(function() { });` 안에 아래 이벤트 처리 코드를 기술해야 함

```

// 게시 글쓰기 폼 유효성 검사
$("#writeForm").on("submit", function() {
    if($("#writer").val().length <= 0) {
        alert("작성자가 입력되지 않았습니다.\n작성자를 입력해주세요");
        $("#writer").focus();
        return false;
    }
    if($("#title").val().length <= 0) {
        alert("제목이 입력되지 않았습니다.\n제목을 입력해주세요");
        $("#title").focus();
        return false;
    }
    if($("#pass").val().length <= 0) {
        alert("비밀번호가 입력되지 않았습니다.\n비밀번호를 입력해주세요");
        $("#pass").focus();
        return false;
    }
    if($("#content").val().length <= 0) {
        alert("내용이 입력되지 않았습니다.\n내용을 입력해주세요");
        $("#content").focus();
        return false;
    }
}

```

```
});
```

▶ DAO(Data Access Object) 클래스에 메서드 추가

BoardDao 클래스에 하나의 게시 글 정보를 DB에 추가하는 다음 메서드를 추가한다.

- com.jspstudy.bbs.dao.BoardDao

```
/* 게시 글쓰기 요청시 호출되는 메서드
 * 게시 글을 작성하고 등록하기 버튼을 클릭하면 게시 글을 DB에 추가하는 메서드
 */
public void insertBoard(Board board) {

    // 아래에서 file1은 아직 사용하지 않고 파일 업로드를 구현할 때 사용할 것임
    String sqlInsert = "INSERT INTO jspbbs(no, title, writer, content,"
        + " reg_date, read_count, pass, file1) "
        + " VALUES(jspbbs_seq.NEXTVAL, ?, ?, ?, SYSDATE, 0, ?, ?)";

    try {
        // 4. DataSource 객체를 이용해 커넥션을 대여한다.
        conn = ds.getConnection();

        /* 5. DBMS에 SQL 쿼리를 발생하기 위해 활성화된
         * Connection 객체로 부터 PreparedStatement 객체를 얻는다.
         *
         * PreparedStatement는 SQL 명령을 캐싱하여(저장하여) 반복적으로 사용하기
         * 때문에 preparedStatement()를 호출할 때 SQL 쿼리 문을 지정해야 한다.
         * PreparedStatement 객체는 반복적으로 사용되는 SQL 명령 외에
         * SQL 문장에 포함되어 변경되는 데이터를 Placeholder(?)로 지정할 수 있다.
         * SQL 문장의 조건식에 사용되는 데이터나 실제 테이블에 입력되는 데이터는
         * SQL 명령이 동일하나 상황에 따라 조건식에 사용되는 데이터가 변경될 수
         * 있고 또한 테이블에 새롭게 추가되는 데이터가 다르게 입력된다. 이렇게
         * 변경되는 데이터가 SQL 문장에서 기술되어야 할 위치에 Placeholder(?)를
         * 지정하고 실제 DB에 SQL 쿼리 문을 전송하기 전에 PreparedStatement
         * 객체의 setXxx()를 이용해 데이터를 변경해 가며 질의 할 수 있다.
         * 여기서 주의할 점은 setXxx()에 지정하는 index의 개념이 배열에서
         * 사용되는 index의 개념이 아니라 첫 번째 Placeholder(?), 두 번째
         * Placeholder(?)와 같이 위치의 개념으로 시작 번호가 1부터 시작된다.
         */
        pstmt = conn.prepareStatement(sqlInsert);

        /* 6. PreparedStatement 객체의 Placeholder(?)에 대응하는
         * 값을 Board 객체의 데이터를 사용해 순서에 맞게 설정하고 있다.
         */
        pstmt.setString(1, board.getTitle());
        pstmt.setString(2, board.getWriter());
```

```

pstmt.setString(3, board.getContent());
pstmt.setString(4, board.getPass());
pstmt.setString(5, board.getFile1());

/* 7. 데이터베이스에 INSERT 쿼리를 발행하여 게시 글 정보를 테이블에 추가한다.
 *
 * executeUpdate()는 DBMS에 INSERT, UPDATE, DELETE 쿼리를
 * 발행하는 메소드로 추가, 수정, 삭제된 레코드의 개수를 반환한다.
 */
pstmt.executeUpdate();

} catch(Exception e) {
    System.out.println("BoardDao - insertBoard() : SQLException");
    e.printStackTrace();

} finally {
    try {
        // 8. 사용한 PreparedStatement 객체를 닫는다.
        if(pstmt != null) pstmt.close();

        // 9. 커넥션 풀로 Connection 객체를 반납한다.
        if(conn != null) conn.close();
    } catch(SQLException se) {}
}

} // end insertBoard(Board board);

```

▶ 게시 글 등록하기 요청을 처리하는 JSP

게시 글쓰기 폼으로부터 들어오는 게시 글 등록 요청을 처리하는 JSP 페이지

- webapp/board/writeProcess.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="com.jspstudy.bbs.vo.*, com.jspstudy.bbs.dao.*" %>
<%
    //POST 요청 방식의 문자셋 처리
    request.setCharacterEncoding("utf-8");

    /* 사용자가 폼에 입력한 데이터 읽어오기
     * HttpServletRequest 객체를 통해 파라미터를 읽어 변수에 저장한다.
     */
    String title = request.getParameter("title");
    String writer = request.getParameter("writer");
    String pass = request.getParameter("pass");

```

```
String content = request.getParameter("content");

/* 하나의 게시 글 정보를 저장하는 자바빈 객체를 생성하고 요청 파라미터로 받은
 * 데이터를 Board 객체에 저장한다.
 */
Board board = new Board();
board.setTitle(title);
board.setWriter(writer);
board.setPass(pass);
board.setContent(content);

// BoardDao 객체 생성하고 게시 글을 DB에 추가한다.
BoardDao dao = new BoardDao();
dao.insertBoard(board);

/* 게시 글쓰기가 완료된 후 response 내장객체의 sendRedirect() 메서드를
 * 이용해 게시 글 리스트로 Redirect 시킨다. response 내장객체의 sendRedirect()
 * 메서드는 요청한 자원이 다른 곳으로 이동되었다고 응답하면서 URL을 알려주고
 * 그 쪽으로 다시 요청하라고 응답하는 메소드이다. 브라우저가 요청한 콘텐츠가
 * 이동했으니 그 쪽으로 다시 요청하라고 응답 데이터로 웹 주소를 알려주면
 * 브라우저는 그 웹 주소로 다시 요청하게 되는데 이를 리다이렉션이라고 한다.
 *
 * Redirect 기법은 웹 브라우저를 새로 고침(F5) 했을 때 동일한 코드가 다시
 * 실행되면 문제가 될 수 있는 경우 클라이언트의 요청을 처리한 후 특정 URL로
 * 이동시키기 위해 사용하는 기법이다. 예를 들어 게시 글쓰기에 대한 요청을 처리한
 * 후 Redirect 시키지 않으면 게시 글쓰기 후에 사용자가 새로 고침(F5) 동작을
 * 하면 바로 이전에 작성한 게시 글 내용과 동일한 내용을 다시 DB에 등록하는 작업을
 * 하게 되는데 이렇게 되면 중복된 데이터를 계속해서 저장하는 문제가 발생한다.
 * 이를 방지하기 위해서 게시 글쓰기가 완료되면 게시 글 리스트(select 문은 반복
 * 사용해도 중복된 데이터가 발생하지 않음)로 이동시키기 위해서 response
 * 내장객체의 sendRedirect() 메소드를 사용해 게시 글 리스트의 URL을
 * 웹 브라우저에게 응답하고 웹 브라우저는 응답 받은 URL로 다시 요청하도록 하는
 * 것이다. 이렇게 게시 글쓰기와 같이 DB 입력 작업이 연동되는 경우 사용자의
 * 새로 고침(F5) 동작에 의해 동일한 요청이 다시 발생하여 DB에 입력되는 데이터의
 * 중복이 발생하거나 SQLException을 발생 시킬 수 있어 Redirect 기법을
 * 사용한다. 이외에 다른 사이트로 이동시킬 때 Redirect 기법을 사용 한다.
 */
response.sendRedirect("boardList.jsp");
%>
```

1-6) 게시 글 수정하기

게시 글 수정은 게시 글 등록과는 다르게 사용자가 작성한 기존 게시 글의 내용을 수정 폼에 보여
줘야 한다. 게시 글 수정 폼을 요청할 때 DB 테이블에서 수정하고자 하는 게시 글을 읽어 와서 폼

에 출력해야 하므로 테이블에서 게시 글을 유일하게 구분할 수 있는 데이터를 요청 파라미터로 보내야 한다. 그래야 서버에서 게시 글 번호에 해당하는 게시 글의 내용을 DB 테이블로부터 읽어와 수정 폼에 출력해 줄 수 있다. 그리고 게시 글을 작성한 사람만 해당 게시 글을 수정할 수 있도록 해야 하므로 게시 글 수정 폼에 기존 게시 글의 내용을 출력하기 전에 게시 글의 비밀번호가 맞는지 먼저 체크해서 맞으면 게시 글 수정 폼에 게시 글 번호에 해당하는 게시 글 내용을 출력하고 비밀번호가 맞지 않으면 적절한 처리를 해야 한다. 우리는 게시 글 비밀번호가 맞지 않으면 게시 글 수정 폼을 요청하기 바로 직전으로 돌려보내도록 구현할 것이다.

▶ 게시 글 수정 폼 요청에 사용할 숨김 form 추가

게시 글 수정 폼 요청에 사용할 form을 boardDetail.jsp에 다음과 같이 추가하자.

- webapp/board/boardDetail.jsp에 아래 코드를 추가

... 종략 ...

```
<!-- content -->
```

```
<div class="row my-5" id="global-content">
```

```
<div class="col">
```

```
<%--
```

아래는 게시 글 수정 폼 요청과 게시 글 삭제 요청에 사용할 숨김 폼 이다.

아래의 수정하기 버튼과 삭제하기 버튼이 클릭되면 jQuery를 이용해 게시 글 비밀번호가 입력되었는지 체크하여 입력되지 않았으면 게시 글 비밀번호를 입력해 달라고 팝업 창을 띄우고 비밀번호가 입력되었으면 게시 글 번호와 비밀번호를 각각 no와 pass에 설정하고 게시 글 수정 폼페이지 요청과 삭제하기 요청을 수행한다.

게시 글 수정은 게시 글 등록과는 다르게 기존의 게시 글을 수정 폼에 보여줘야 한다. 게시 글 수정 폼 요청시 테이블에서 수정하고자 하는 게시 글을 유일하게 구분할 수 있는 데이터를 요청 파라미터로 보내야 그 게시 글에 해당하는 정보를 읽어와 게시 글 수정 폼에 출력할 수 있다. 또한 게시 글 삭제 요청도 테이블에서 삭제하고자 하는 게시 글의 정보를 유일하게 구분할 수 있는 게시 글 번호를 요청 파라미터로 보내야 그 게시 글에 해당하는 정보를 테이블 삭제할 수 있다.

```
--%>
```

```
<form name="checkForm" id="checkForm">
```

```
<input type="hidden" name="no" id="no" value="${ board.no }"/>
```

```
<input type="hidden" name="pass" id="rPass" />
```

```
</form>
```

```
<div class="row text-center">
```

```
<div class="col">
```

```
<h2 class="fs-3 fw-bold">게시 글 상세보기</h2>
```

```
</div>
```

```
</div>
```


... 중략 ...

▶ 게시 글 상세 페이지에서 게시 글 수정 폼 요청 스크립트

게시 글 상세보기(boardDetail.jsp)에서 수정하기 버튼이 클릭되면 비밀번호 입력란에 입력된 데이터를 읽어와 checkForm의 hidden 컨트롤인 비밀번호 입력란에 추가하고 서버로 submit 하는 자바스크립트 코드를 다음을 참고해서 formcheck.js 파일에 추가하자. 자바스크립트 코드는 DOM이 준비되었을 때 문서 객체에 접근해야 하므로 `$(function() { });` 코드 안쪽에 기술해야 한다.

- webapp/js/formcheck.js

```
/* 게시 글 상세 보기에서 게시 글 수정 폼 요청 처리
 * 아래와 같이 hidden 폼을 통해 get 방식으로 요청할 수 있다.
 */
$("#detailUpdate").on("click", function() {

    var pass = $("#pass").val();
    if(pass.length <= 0) {
        alert("게시 글을 수정하려면 비밀번호를 입력해주세요!");
        return false;
    }

    $("#rPass").val(pass);
    $("#checkForm").attr("action", "updateForm.jsp");
    $("#checkForm").submit();
});
```

▶ DAO(Data Access Object) 클래스에 메서드 추가

BoardDao 클래스에 게시 글 번호에 해당하는 게시 글 비밀번호가 맞는지 체크해서 맞으면 true 를 리턴 false를 반환하는 다음 메서드를 추가한다.

- com.jspstudy.bbs.dao.BoardDao

```
/* 게시 글 수정, 게시 글 삭제 시 비밀번호 입력을 체크하는 메서드
 */
public boolean isPassCheck(int no, String pass) {
    boolean isPass = false;
    String sqlPass = "SELECT pass FROM jsbbbs WHERE no=?";
    try {
        // 4. DataSource 객체를 이용해 커넥션을 대여한다.
        conn = ds.getConnection();

        /* 5. DBMS에 SQL 쿼리를 발생하기 위해 활성화된
         * Connection 객체로 부터 PreparedStatement 객체를 얻는다.
         *
         * PreparedStatement는 SQL 명령을 캐싱하여(저장하여) 반복적으로 사용하기
```

- * 때문에 preparedStatement()를 호출할 때 SQL 쿼리 문을 지정해야 한다.
- * PreparedStatement 객체는 반복적으로 사용되는 SQL 명령 외에
- * SQL 문장에 포함되어 변경되는 데이터를 Placeholder(?)로 지정할 수 있다.
- * SQL 문장의 조건식에 사용되는 데이터나 실제 테이블에 입력되는 데이터는
- * SQL 명령이 동일하나 상황에 따라 조건식에 사용되는 데이터가 변경될 수
- * 있고 또한 테이블에 새롭게 추가되는 데이터가 다르게 입력된다. 이렇게
- * 변경되는 데이터가 SQL 문장에서 기술되어야 할 위치에 Placeholder(?)를
- * 지정하고 실제 DB에 SQL 쿼리 문을 전송하기 전에 PreparedStatement
- * 객체의 setXxx()를 이용해 데이터를 변경해 가며 질의 할 수 있다.
- * 여기서 주의할 점은 setXxx()에 지정하는 index의 개념이 배열에서
- * 사용되는 index의 개념이 아니라 첫 번째 Placeholder(?), 두 번째
- * Placeholder(?)와 같이 위치의 개념으로 시작 번호가 1부터 시작된다.

```

**/
pstmt = conn.prepareStatement(sqlPass);

/* 6. PreparedStatement 객체의 Placeholder(?)에 대응하는
* 값을 매개변수로 받은 데이터를 사용해 순서에 맞게 설정하고 있다.
**/
pstmt.setInt(1, no);

// 7. 데이터베이스에 SELECT 쿼리를 발행하고 ResultSet 객체로 받는다.
rs = pstmt.executeQuery();

/* 8. no가 Primary Key 이므로 no에 해당하는 게시 글을 DB에서 읽으면
* 없거나 또는 하나의 게시 글 정보를 얻을 수 있으므로 if문을 사용했다.
*
* ResultSet 객체는 DB로 부터 읽어온 데이터에 접근하기 위해 테이블의
* 행을 가리키는 cursor를 제공한다. 맨 처음 ResultSet 객체를 반환
* 받으면 cursor는 첫 번째 행 바로 이전을 가리키고 있다. ResultSet의
* cursor가 맨 마지막 행에 도달하면 while문을 빠져 나온다.
* 여기서는 결과가 많아야 한 행이므로 한 번 실행되고 if문을 빠져 나온다.
*
* ResultSet에는 다양한 데이터 타입에 대응하는 getter 메소드를
* 지원하고 있으며 SELECT 문장에서 지정한 컬럼의 index 또는
* 컬럼명으로 테이블의 필드 값을 가져올 수 있도록 getXxx() 메소드가
* 오버로딩 되어 있어 index와 컬럼명 둘 다 사용이 가능하다.
* 여기에 지정하는 index는 배열에서 사용되는 index의 개념이 아니라
* 첫 번째 컬럼, 두 번째 컬럼과 같이 위치의 개념으로 1부터 시작된다.
**/
if(rs.next()) {
    isPass = rs.getString(1).equals(pass);
}
} catch(SQLException e) {
    System.out.println("BoardDao - isPassCheck() : SQLException");
    e.printStackTrace();
}

```

```

    } finally {
        try {
            // 9. 사용한 ResultSet과 PreparedStatement 객체를 닫는다.
            if(rs != null) rs.close();
            if(pstmt != null) pstmt.close();

            // 10. 커넥션 풀로 Connection 객체를 반환한다.
            if(conn != null) conn.close();
        } catch(SQLException e) {}
    }

    // 11. 메서드 실행 결과로 게시 글 비밀번호가 맞는지 여부를 반환한다.
    return isPass;

} // end isPassCheck();

```

▶ 게시 글 수정 폼 요청을 처리하는 JSP

- webapp/board/updateForm.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="com.jspstudy.bbs.dao.*, com.jspstudy.bbs.vo.*" %>
<%
    /* 게시 글 수정 폼에는 게시 글 등록 폼과는 다르게 기존의 게시 글 정보를 출력해야 한다.
     * 테이블에서 하나의 게시 글 정보를 읽어오기 위해서 boardDetail.jsp에서
     * 게시 글 수정 폼 요청을 하면서 테이블에서 하나의 게시 글을 유일하게 구분할 수 있는
     * 게시 글 번호를 요청 파라미터로 보냈기 때문에 이 게시 글 번호를 요청 파라미터로
     * 부터 읽어 BoardDao를 통해서 게시 글 번호에 해당하는 게시 글을 읽을 수 있다.
     *
     * 아래에서 no라는 요청 파라미터가 없다면 NumberFormatException 발생
     */
    String sNo = request.getParameter("no");
    String pass = request.getParameter("pass");

    /* BoardDao 객체를 생성하고 DB에서 게시 글 번호와 사용자가 입력한 게시 글
     * 비밀번호가 맞는지를 체크하여 맞으면 게시 글 번호에 해당하는 게시 글을 읽어온다.
     */
    BoardDao dao = new BoardDao();
    int no = Integer.parseInt(sNo);
    boolean isPassCheck = dao.isPassCheck(no, pass);

    if(isPassCheck) {

        // BoardDao 객체를 이용해 no에 해당하는 게시 글 정보를 읽어온다.

```

```

Board board = dao.getBoard(no);
%>
<!DOCTYPE html>
<html lang="ko">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>게시 글 수정하기</title>
    <link href="../bootstrap/bootstrap.min.css" rel="stylesheet" >
    <script src="../js/jquery-3.3.1.min.js"></script>
    <script src="../js/formcheck.js"></script>
  </head>
  <body>
    <div class="container">
      <%@ include file="../pages/header.jsp" %>
      <!-- content -->
      <div class="row my-5" id="global-content">
        <div class="col">
          <div class="row text-center">
            <div class="col">
              <h2 class="fs-3 fw-bold">게시 글 수정하기</h2>
            </div>
          </div>
        </div>
        <!--
          게시 글을 수정하기 위해서는 테이블에서 게시 글을 유일하게 구분할 수 있는
          데이터가 필요하다. 아래에서 no는 테이블에서 하나의 게시 글을 유일하게
          구분할 수 있는 데이터로 아래 폼이 서버로 전송될 때 이 no도 같이 서버로
          전송되어야 게시 글 정보를 제대로 수정할 수 있기 때문에 화면에는 보이지
          않고 폼이 전송될 때 요청 파라미터에 추가될 수 있도록 hidden 폼 컨트롤로
          폼에 추가하였다.
        -->
        <form name="updateForm" action="updateProcess.jsp" id="updateForm"
          class="row g-3 border-primary" method="post">
          <input type="hidden" name="no" value="<%= board.getNo() %>">
          <div class="col-4 offset-md-2">
            <label for="writer" class="form-label">글쓴이</label>
            <input type="text" class="form-control" name="writer" id="writer"
              placeholder="작성자를 입력해 주세요" value="<%= board.getWriter() %>">
          </div>
          <div class="col-4 ">
            <label for="pass" class="form-label">비밀번호</label>
            <input type="password" class="form-control" name="pass"
              id="pass">
          </div>
          <div class="col-8 offset-md-2">

```

```

        <label for="title" class="form-label">제 목</label>
        <input type="text" class="form-control" name="title"
            id="title" value="<%= board.getTitle() %>">
    </div>
    <div class="col-8 offset-md-2">
        <label for="content" class="form-label">내 용</label>
        <textarea class="form-control" name="content" id="content"
            rows="10"><%= board.getContent() %></textarea>
    </div>
    <div class="col-8 offset-md-2 text-center mt-5">
        <input type="submit" value="수정하기" class="btn btn-primary"/>
        &nbsp;&nbsp;&nbsp;<input type="button" value="목록보기"
            onclick="location.href='boardList.jsp'" class="btn btn-primary"/>
    </div>
    </form>
</div>
</div>
<%@ include file="../pages/footer.jsp" %>
</div>
<script src="../bootstrap/bootstrap.bundle.min.js"></script>
</body>
<%
/* 게시 글 번호에 해당하는 비밀번호가 틀리면 비밀번호가 틀리다. 라고 알려주고
 * 브라우저의 history 객체를 이용해 바로 이전에 있었던 주소로 돌려보낸다.
 */
} else {
%>
    <script>
        alert("비밀번호가 다릅니다.");
        history.back();
    </script>
    <%}%>
</html>

```

▶ 게시 글 수정 폼 유효성 검사 스크립트

`$(function() { });` 안에 아래 이벤트 처리 코드를 기술해야 함

- webapp/js/formcheck.js

// 게시 글 수정 폼 유효성 검사

```

$(("#updateForm").on("submit", function() {
    if($("#writer").val().length <= 0) {
        alert("작성자가 입력되지 않았습니다.\n작성자를 입력해주세요");
        $("#writer").focus();
        return false;
    }

```

```

}
if($("#title").val().length <= 0) {
    alert("제목이 입력되지 않았습니다.\n제목을 입력해주세요");
    $("#title").focus();
    return false;
}
if($("#pass").val().length <= 0) {
    alert("비밀번호가 입력되지 않았습니다.\n비밀번호를 입력해주세요");
    $("#pass").focus();
    return false;
}
if($("#content").val().length <= 0) {
    alert("내용이 입력되지 않았습니다.\n내용을 입력해주세요");
    $("#content").focus();
    return false;
}
});

```

▶ DAO(Data Access Object) 클래스에 메서드 추가

게시 글 수정 폼으로부터 들어오는 데이터를 받아서 게시 글 번호에 해당하는 게시 글을 DB에서 수정하는 다음 메서드를 추가한다.

- com.jspstudy.bbs.dao.BoardDao

```

/* 게시 글 수정 요청시 호출되는 메서드
 * 게시 글 수정 폼에서 수정하기 버튼이 클릭되면 게시 글을 DB에 수정하는 메서드
 */
public void updateBoard(Board board) {

    String sqlUpdate = "UPDATE jspbbs set title=?, writer=?, content=?"
        + " reg_date=SYSDATE, file1=? WHERE no=?";

    try {
        // 4. DataSource 객체를 이용해 커넥션을 대여한다.
        conn = ds.getConnection();

        /* 5. DBMS에 SQL 쿼리를 발생하기 위해 활성화된
         * Connection 객체로 부터 PreparedStatement 객체를 얻는다.
         *
         * PreparedStatement는 SQL 명령을 캐싱하여(저장하여) 반복적으로 사용하기
         * 때문에 preparedStatement()를 호출할 때 SQL 쿼리 문을 지정해야 한다.
         * PreparedStatement 객체는 반복적으로 사용되는 SQL 명령 외에
         * SQL 문장에 포함되어 변경되는 데이터를 Placeholder(?)로 지정할 수 있다.
         * SQL 문장의 조건식에 사용되는 데이터나 실제 테이블에 입력되는 데이터는
         * SQL 명령이 동일하나 상황에 따라 조건식에 사용되는 데이터가 변경될 수

```

```

* 있고 또한 테이블에 새롭게 추가되는 데이터가 다르게 입력된다. 이렇게
* 변경되는 데이터가 SQL 문장에서 기술되어야 할 위치에 Placeholder(?)를
* 지정하고 실제 DB에 SQL 쿼리 문을 전송하기 전에 PreparedStatement
* 객체의 setXxx()를 이용해 데이터를 변경해 가며 질의 할 수 있다.
* 여기서 주의할 점은 setXxx()에 지정하는 index의 개념이 배열에서
* 사용되는 index의 개념이 아니라 첫 번째 Placeholder(?), 두 번째
* Placeholder(?)와 같이 위치의 개념으로 시작 번호가 1부터 시작된다.
**/
pstmt = conn.prepareStatement(sqlUpdate);

/* 6. PreparedStatement 객체의 Placeholder(?)에 대응하는
* 값을 매개변수로 받은 데이터를 사용해 순서에 맞게 설정하고 있다.
**/
pstmt.setString(1, board.getTitle());
pstmt.setString(2, board.getWriter());
pstmt.setString(3, board.getContent());
pstmt.setString(4, board.getFile1());
pstmt.setInt(5, board.getNo());

/* 7. 데이터베이스에 INSERT 쿼리를 발행하여 게시 글 정보를 테이블에 추가한다.
*
* executeUpdate()는 DBMS에 INSERT, UPDATE, DELETE 쿼리를
* 발행하는 메소드로 추가, 수정, 삭제된 레코드의 개수를 반환한다.
**/
pstmt.executeUpdate();

} catch(Exception e) {
    System.out.println("BoardDao - updateBoard() : SQLException");
    e.printStackTrace();

} finally {
    try {
        // 8. 사용한 PreparedStatement 객체를 닫는다.
        if(pstmt != null) pstmt.close();

        // 9. 커넥션 풀로 Connection 객체를 반납한다.
        if(conn != null) conn.close();
    } catch(SQLException se) {}
}
} // end updateBoard(Board board);

```

▶ 게시 글 수정하기 요청을 처리하는 JSP

게시 글 수정 폼으로부터 들어오는 게시 글 수정 요청을 처리하는 JSP 페이지

- webapp/board/updateProcess.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="com.jspstudy.bbs.vo.*, com.jspstudy.bbs.dao.*" %>
<%
    // POST 방식의 요청에 대한 문자셋 처리
    request.setCharacterEncoding("utf-8");

    // 요청 파라미터를 저장할 변수 선언
    String pass= null, title = null, writer = null, content = null;
    int no = 0;

    /* 사용자가 폼에서 수정한 데이터를 요청 파라미터로 부터 읽어온다.
    *
    * 게시 글을 수정하기 위해서 updateForm.jsp에서 게시 글 수정 요청을
    * 하면서 테이블에서 게시 글 하나를 유일하게 구분할 수 있는 게시 글 번호를
    * 요청 파라미터로 보냈기 때문에 이 게시 글 번호를 요청 파라미터로부터 읽어
    * BoardDao를 통해서 게시 글 번호에 해당하는 게시 글의 내용을 수정 할 수 있다.
    *
    * 아래에서 no라는 요청 파라미터가 없다면 NumberFormatException 발생
    */
    no = Integer.parseInt(request.getParameter("no"));
    pass = request.getParameter("pass");

    // BoardDao 객체를 생성하고 게시 글 비밀번호를 체크해 맞지 않으면 이전으로 돌려보낸다.
    BoardDao dao = new BoardDao();
    boolean isPassCheck = dao.isPassCheck(no, pass);

    // 게시 글 번호에 해당하는 게시글 비밀번호가 틀리다면
    if(! isPassCheck) {

        /* 문자열을 보다 효율적으로 다루기 위해서 StringBuilder 객체를 이용해
        * 응답 데이터를 작성하고 있다. 아래에서는 비밀번호가 틀리면 사용자에게
        * 경고 창을 띄우기 위해서 자바스크립트의 alert() 함수를 실행하는 코드를
        * 응답 데이터로 작성하고 있다.
        */
        StringBuilder sb = new StringBuilder();
        sb.append("<script>");
        sb.append(" alert('비밀번호가 맞지 않습니다.');");
        sb.append(" history.back();");
        sb.append("</script>");

        /* 응답 객체에 연결된 JspWriter 객체를 이용해 응답 데이터를 전송하고
        * 더 이상 실행할 필요가 없으므로 return 문을 이용해 현재 메서드를 종료한다.
        */
```



```

        out.println(sb.toString());
        System.out.println("비밀번호 맞지 않음");
        return;
    }

```

// 비밀번호가 맞으면 사용자가 게시 글 수정 폼에 입력한 데이터를 읽어온다.

```

title = request.getParameter("title");
writer = request.getParameter("writer");
content = request.getParameter("content");

```

/* 하나의 게시 글 정보를 저장하는 자바빈 객체를 생성하고 파라미터로

* 넘겨받은 요청 데이터를 Board 객체에 저장한다.

*/

```
Board board = new Board();
```

```
board.setNo(no);
```

```
board.setTitle(title);
```

```
board.setWriter(writer);
```

```
board.setPass(pass);
```

```
board.setContent(content);
```

// BoardDao의 updateBoard() 메서드를 이용해 DB에서 게시 글을 수정한다.

```
dao.updateBoard(board);
```

/* 게시 글 수정이 완료된 후 response 내장객체의 sendRedirect() 메서드를

* 이용해 게시 글 리스트로 Redirect 시킨다. response 내장객체의 sendRedirect()

* 메서드는 요청한 자원이 다른 곳으로 이동되었다고 응답하면서 URL을 알려주고

* 그 쪽으로 다시 요청하라고 응답하는 메소드이다. 브라우저가 요청한 콘텐츠가

* 이동했으니 그 쪽으로 다시 요청하라고 응답 데이터로 웹 주소를 알려주면

* 브라우저는 그 웹 주소로 다시 요청하게 되는데 이를 리다이렉션이라고 한다.

*

* Redirect 기법은 웹 브라우저를 새로 고침(F5) 했을 때 동일한 코드가 다시

* 실행되면 문제가 될 수 있는 경우 클라이언트의 요청을 처리한 후 특정 URL로

* 이동시키기 위해 사용하는 기법이다. 예를 들어 게시 글 수정하기 요청을 처리한

* 후 Redirect 시키지 않으면 게시 글 수정 후에 사용자가 새로 고침(F5) 동작을

* 하면 바로 이전에 수정한 게시 글 내용과 동일한 내용을 다시 DB에 수정하는 작업을

* 하게 되는데 이렇게 되면 계속해서 같은 데이터를 수정하려고 하는 문제가 발생한다.

* 이를 방지하기 위해서 게시 글 수정이 완료되면 게시 글 리스트(select 문은 반복

* 사용해도 중복된 데이터가 발생하지 않음)로 이동시키기 위해서 response

* 내장객체의 sendRedirect() 메소드를 사용해 게시 글 리스트의 URL을

* 웹 브라우저에게 응답하고 웹 브라우저는 응답 받은 URL로 다시 요청하도록 하는

* 것이다. 이렇게 게시 글 수정과 같이 DB 입력 작업이 연동되는 경우 사용자의

* 새로 고침(F5) 동작에 의해 동일한 요청이 다시 발생하여 DB에서 이미 수정된

* 게시 글을 수정하거나 SQLException을 발생 시킬 수 있어 Redirect 기법을

* 사용한다. 이외에 다른 사이트로 이동시킬 때 Redirect 기법을 사용 한다.

*/

```
response.sendRedirect("boardList.jsp");
%>
```

1-7) 게시 글 삭제하기

▶ DAO(Data Access Object) 클래스에 메서드 추가

BoardDao 클래스에 no에 해당하는 하나의 게시 글을 DB에서 삭제하는 다음 메서드를 추가한다.

- com.jspstudy.bbs.dao.BoardDao

```
/* 게시 글 삭제 요청 시 호출되는 메서드
 * no에 해당 하는 게시 글을 DB에서 삭제하는 메서드
 */
public void deleteBoard(int no) {

    String sqlDelete = "DELETE FROM jspbbs WHERE no=?";
    try {
        // 4. DataSource 객체를 이용해 커넥션을 대여한다.
        conn = ds.getConnection();

        /* 5. DBMS에 SQL 쿼리를 발생하기 위해 활성화된
         * Connection 객체로 부터 PreparedStatement 객체를 얻는다.
         *
         * PreparedStatement는 SQL 명령을 캐싱하여(저장하여) 반복적으로 사용하기
         * 때문에 preparedStatement()를 호출할 때 SQL 쿼리 문을 지정해야 한다.
         * PreparedStatement 객체는 반복적으로 사용되는 SQL 명령 외에
         * SQL 문장에 포함되어 변경되는 데이터를 Placeholder(?)로 지정할 수 있다.
         * SQL 문장의 조건식에 사용되는 데이터나 실제 테이블에 입력되는 데이터는
         * SQL 명령이 동일하나 상황에 따라 조건식에 사용되는 데이터가 변경될 수
         * 있고 또한 테이블에 새롭게 추가되는 데이터가 다르게 입력된다. 이렇게
         * 변경되는 데이터가 SQL 문장에서 기술되어야 할 위치에 Placeholder(?)를
         * 지정하고 실제 DB에 SQL 쿼리 문을 전송하기 전에 PreparedStatement
         * 객체의 setXxx()를 이용해 데이터를 변경해 가며 질의 할 수 있다.
         * 여기서 주의할 점은 setXxx()에 지정하는 index의 개념이 배열에서
         * 사용되는 index의 개념이 아니라 첫 번째 Placeholder(?), 두 번째
         * Placeholder(?)와 같이 위치의 개념으로 시작 번호가 1부터 시작된다.
         */
        pstmt = conn.prepareStatement(sqlDelete);

        /* 6. PreparedStatement 객체의 Placeholder(?)에 대응하는
         * 값을 매개변수로 받은 데이터를 사용해 순서에 맞게 설정하고 있다.
         */
        pstmt.setInt(1, no);

        /* 7. 데이터베이스에 INSERT 쿼리를 발행하여 게시 글 정보를 테이블에 추가한다.
```

```

*
* executeUpdate()는 DBMS에 INSERT, UPDATE, DELETE 쿼리를
* 발행하는 메소드로 추가, 수정, 삭제된 레코드의 개수를 반환한다.
**/
pstmt.executeUpdate();

} catch(Exception e) {
    e.printStackTrace();
} finally {
    try {
        // 8. 사용한 PreparedStatement 객체를 닫는다.
        if(pstmt != null) pstmt.close();

        // 9. 커넥션 풀로 Connection 객체를 반납한다.
        if(conn != null) conn.close();
    } catch(SQLException se) {}
}
} // end deleteBoard(int no);

```

▶ 게시 글 상세 페이지에서 게시 글 삭제 요청 스크립트

게시 글 상세보기(boardDetail.jsp)에서 삭제하기 버튼이 클릭되면 비밀번호 입력란에 입력된 데이터를 읽어와 checkForm의 hidden 컨트롤인 비밀번호 입력란에 추가하고 서버로 submit 하는 자바스크립트 코드를 다음을 참고해서 formcheck.js 파일에 추가하자. 자바스크립트 코드는 DOM이 준비되었을 때 문서 객체에 접근해야 하므로 \$(function() { }); 코드 안쪽에 기술해야 한다.

- webapp/js/formcheck.js

```

/* 게시 글 상세 보기에서 게시 글 수정 폼 요청 처리
* 아래와 같이 hidden 폼을 통해 get 방식으로 요청할 수 있다.
**/
$("#detailUpdate").on("click", function() {

    var pass = $("#pass").val();
    if(pass.length <= 0) {
        alert("게시 글을 수정하려면 비밀번호를 입력해주세요");
        return false;
    }

    $("#rPass").val(pass);
    $("#checkForm").attr("action", "updateForm.jsp");
    $("#checkForm").submit();
});

```

▶ 게시 글 삭제하기 요청을 처리하는 JSP

- webapp/board/deleteProcess.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="com.jspstudy.bbs.vo.*, com.jspstudy.bbs.dao.*" %>
<%
    /* 게시 글을 삭제하기 위해서 boardDetail.jsp에서 게시 글 삭제 요청을
    * 하면서 테이블에서 게시 글 하나를 유일하게 구분할 수 있는 게시 글 번호를
    * 요청 파라미터로 보냈기 때문에 이 게시 글 번호를 요청 파라미터로부터 읽어
    * BoardDao를 통해서 게시 글 번호에 해당하는 게시 글을 DB에서 삭제할 수 있다.
    *
    * 아래에서 no라는 요청 파라미터가 없다면 NumberFormatException 발생
    */
    String sNo = request.getParameter("no");
    String pass = request.getParameter("pass");
    int no = Integer.parseInt(sNo);

    /* BoardDao 객체 생성하고 다시 한 번 게시 글의
    * 비밀번호를 체크해 맞지 않으면 이전으로 돌려보낸다.
    */
    BoardDao dao = new BoardDao();
    boolean isPassCheck = dao.isPassCheck(no, pass);
    if(! isPassCheck) {

        /* 문자열을 보다 효율적으로 다루기 위해서 StringBuilder 객체를 이용해
        * 응답 데이터를 작성하고 있다. 아래에서는 비밀번호가 틀리면 사용자에게
        * 경고 창을 띄우기 위해서 자바스크립트의 alert() 함수를 실행하는 코드를
        * 응답 데이터로 작성하고 있다.
        */
        StringBuilder sb = new StringBuilder();
        sb.append("<script>");
        sb.append(" alert('비밀번호가 맞지 않습니다.');");
        sb.append(" history.back();");
        sb.append("</script>");

        /* 응답 객체에 연결된 JspWriter 객체를 이용해 응답 데이터를 전송하고
        * 더 이상 실행할 필요가 없으므로 return 문을 이용해 현재 메서드를 종료한다.
        */
        out.println(sb.toString());
        System.out.println("비밀번호 맞지 않음");
        return;
    }

    // BoardDao의 deleteBoard() 메서드를 이용해 DB에서 게시 글을 삭제한다.
    dao.deleteBoard(no);
```

```

/* 게시 글 삭제가 완료된 후 response 내장객체의 sendRedirect() 메서드를
 * 이용해 게시 글 리스트로 Redirect 시킨다. response 내장객체의 sendRedirect()
 * 메서드는 요청한 자원이 다른 곳으로 이동되었다고 응답하면서 URL을 알려주고
 * 그 쪽으로 다시 요청하라고 응답하는 메소드이다. 브라우저가 요청한 콘텐츠가
 * 이동했으니 그 쪽으로 다시 요청하라고 응답 데이터로 웹 주소를 알려주면
 * 브라우저는 그 웹 주소로 다시 요청하게 되는데 이를 리다이렉션이라고 한다.
 *
 * Redirect 기법은 웹 브라우저를 새로 고침(F5) 했을 때 동일한 코드가 다시
 * 실행되면 문제가 될 수 있는 경우 클라이언트의 요청을 처리한 후 특정 URL로
 * 이동시키기 위해 사용하는 기법이다. 예를 들어 게시 글 삭제하기 요청을 처리한
 * 후 Redirect 시키지 않으면 게시 글 삭제 후에 사용자가 새로 고침(F5) 동작을
 * 하면 바로 이전에 삭제한 게시 글과 동일한 게시 글을 다시 DB에 삭제하는 작업을
 * 하게 되는데 이렇게 되면 없는 게시 글을 계속해서 삭제하려고 하는 문제가 발생한다.
 * 이를 방지하기 위해서 게시 글 삭제가 완료되면 게시 글 리스트(select 문은 반복
 * 사용해도 중복된 데이터가 발생하지 않음)로 이동시키기 위해서 response
 * 내장객체의 sendRedirect() 메소드를 사용해 게시 글 리스트의 URL을
 * 웹 브라우저에게 응답하고 웹 브라우저는 응답 받은 URL로 다시 요청하도록 하는
 * 것이다. 이렇게 게시 글 삭제와 같이 DB 입력 작업이 연동되는 경우 사용자의
 * 새로 고침(F5) 동작에 의해 동일한 요청이 다시 발생하여 DB에 없는 데이터를
 * 삭제하려고 하거나 SQLException을 발생 시킬 수 있어 Redirect 기법을
 * 사용한다. 이외에 다른 사이트로 이동시킬 때 Redirect 기법을 사용 한다.
 */
response.sendRedirect("boardList.jsp");
%>

```

2) 파일 업로드 구현

앞에서 구현한 CRUD 게시판에 파일 업로드 기능을 추가해 자료실 게시판을 만들어 보자.

파일 업로드 라이브러리에는 apache 그룹의 commons FileUpload와 cos 라이브러리가 많이 사용된다. 앞에서 JDBC 수업을 진행할 때 commons FileUpload와 cos 라이브러리에 대해서 알아보았다. 이번에는 게시 글쓰기에서 cos 라이브러리를 이용해 파일을 업로드 하는 방법에 대해 알아볼 것이다.

먼저 <http://www.servlets.com/cos>에 접속해 cos-26Dec2008.zip 파일을 다운로드 받아 압축을 풀고 cos.jar 파일을 웹 프로젝트의 WEB-INF/lib 폴더에 복사 하고 파일을 업로드해 저장할 webapp/upload 폴더를 만들어 놓자

이번 예제는 기존에 구현한 게시판에 파일 업로드 기능을 추가하는 것이므로 기존의 게시 글쓰기에서 사용자가 파일을 선택할 수 있도록 파일 선택 상자를 추가해야 하지만 앞에서 구현할 당시에 이미 파일선택 상자는 추가된 상태이다. 그래서 파일 업로드에 필요한 writeForm.jsp 파일은 form 태그의 속성에 파일 업로드에 맞게 enctype 속성을 추가하면 되고 BoardDao의 insertBoard() 메서드는 수정할 필요 없이 그대로 사용하면 된다. 그리고 클라이언트가 보낸 파일을 받아 cos 라이브러리를 이용해 파일 업로드를 처리하는 writeProcess.jsp를 파일 업로드에 맞게 작성하면 된다.

- 실습용 프로젝트 : JSPClassBBS02
- 완성 프로젝트 : JSPStudyBBS02

▶ 파일 업로드에 필요한 라이브러리 추가

- webapp/WEB-INF/lib/ 폴더에 다운로드 받아 복사
 - cos 파일 업로드 라이브러리
- cos.jar

▶ web.xml에 애플리케이션 초기화 파라미터 추가

```
<!-- 파일을 업로드할 폴더를 애플리케이션 초기화 파라미터로 설정 -->
<context-param>
  <param-name>uploadDir</param-name>
  <param-value>/upload</param-value>
</context-param>
```

▶ 게시 글쓰기 폼 요청을 처리하는 JSP 페이지

게시 글쓰기 폼 페이지도 앞에서 작성한 페이지와 거의 동일하며 파일 업로드를 위해서 <form> 태그 부분의 enctype이 추가되어 빨간색 볼드체로 표현하였다.

- webapp/board/writeForm.jsp

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

```

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>게시 글쓰기</title>
  <link href="../bootstrap/bootstrap.min.css" rel="stylesheet" >
  <script src="../js/jquery-3.3.1.min.js"></script>
<script src="../js/formcheck.js"></script>
</head>
<body>
  <div class="container">
    <%@ include file="../pages/header.jsp" %>
    <!-- content -->
    <div class="row my-5" id="global-content">
      <div class="col">
        <div class="row text-center">
          <div class="col">
            <h2 class="fs-3 fw-bold">게시 글쓰기</h2>
          </div>
        </div>
      </div>
      <!--
      form은 사용자로부터 데이터를 입력받기 위한 폼 컨트롤들로 구성된다.
      사용자가 입력한 일반적인 데이터 즉 이름, 직업 등의 데이터는 문자열로
      이루어진 데이터로 사용자가 submit 버튼을 클릭하게 되면 브라우저는
      이 데이터를 서버로 보내기 전에 페이지의 문자 셋을 기준으로 인코딩을 한 후
      서버로 전송하게 된다. 문자열로 이루어진 데이터를 브라우저가 인코딩 하는
      방식은 application/x-www-form-urlencoded 방식으로 form의
      기본 enctype 이다. 그래서 enctype을 생략할 수 있지만 파일과 같은
      데이터를 전송하기 위해서는 위의 인코딩 방식으로는 서버로 전송할 수 없다.
      파일은 문자열 형태의 데이터가 아닌 바이너리 형태의 데이터로 파일을 업로드
      하기 위해서는 form 태그의 전송방식을 method 속성에 post를 지정하고
      인코딩 타입(enctype)을 multipart/form-data로 지정해야 한다.
      -->
      <form name="writeForm" action="writeProcess.jsp" id="writeForm"
        class="row g-3 border-primary" method="post" enctype="multipart/form-data">
        <div class="col-4 offset-md-2">
          <label for="writer" class="form-label">글쓴이</label>
          <input type="text" class="form-control" name="writer" id="writer"
            placeholder="작성자를 입력해 주세요">
        </div>
        <div class="col-4 ">
          <label for="pass" class="form-label">비밀번호</label>
          <input type="password" class="form-control" name="pass" id="pass" >
        </div>
        <div class="col-8 offset-md-2">
          <label for="title" class="form-label">제 목</label>

```

```

        <input type="text" class="form-control" name="title" id="title" >
    </div>
    <div class="col-8 offset-md-2">
        <label for="content" class="form-label">내용</label>
        <textarea class="form-control" name="content" id="content"
rows="10"></textarea>
    </div>
    <div class="col-8 offset-md-2">
        <label for="file1" class="form-label">파일</label>
        <input type="file" class="form-control" name="file1" id="file1" >
    </div>
    <div class="col-8 offset-md-2 text-center mt-5">
        <input type="submit" value="등록하기" class="btn btn-primary"/>
        &nbsp;&nbsp;&nbsp;<input type="button" value="목록보기"
onclick="location.href='boardList.jsp'" class="btn btn-primary"/>
    </div>
</form>
</div>
</div>
<%@ include file="../pages/footer.jsp" %>
</div>
<script src="../bootstrap/bootstrap.bundle.min.js"></script>
</body>
</html>

```

▶ DAO(Data Access Object) 객체

파일 자체를 DB에 저장하는 것이 아니라 파일의 경로와 이름을 문자열로 저장할 것이므로 앞에서 작성한 게시판에서 사용했던 BoardDao 클래스를 수정 없이 그대로 사용하면 되기 때문에 DAO 클래스의 소스 코드는 교안에 추가하지 않을 것이니 앞의 코드를 참고하기 바란다.

▶ 게시 글쓰기 폼에서 요청한 파일 업로드를 처리하는 JSP 페이지

파일 업로드 작업을 구현하는 페이지로 새롭게 추가되거나 변경된 부분이 많기 때문에 처음부터 코드를 보면서 학습하면 된다.

- webapp/board/writeProcess.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@page import="java.util.*, java.io.*"%>
<%@ page import="com.jspstudy.bbs.vo.*, com.jspstudy.bbs.dao.*" %>

<%-- 파일 업로드 cos 라이브러리 관련 импорт --%>
<%@ page import="com.oreilly.servlet.*, com.oreilly.servlet.multipart.*" %>

```



```

<%!
/* JSP 선언부를 이용해 JSP 초기화 메서드를 정의하고 애플리케이션
 * 초기화 파라미터인 uploadDir을 읽어서 파일이 저장되는 폴더로 사용할 것임
 */
static String uploadDir;
static File parentFile;

public void jsplnit() {
    // web.xml에 지정한 웹 어플리케이션 초기화 파라미터를 읽는다.
    uploadDir = getServletContext().getInitParameter("uploadDir");

    /* 웹 어플리케이션 초기화 파라미터로 읽어온 이미지가 저장될 폴더의
     * 로컬 경로를 구하여 그 경로와 파일명으로 File 객체를 생성한다.
     */
    String realPath = getServletContext().getRealPath(uploadDir);
    parentFile = new File(realPath);

    /* 파일 객체에 지정한 위치에 디렉토리가 존재하지 않거나
     * 파일 객체가 디렉토리가 아니라면 디렉토리를 생성한다.
     */
    if(! (parentFile.exists() && parentFile.isDirectory())) {
        parentFile.mkdir();
    }
    System.out.println("init - " + parentFile);
}
%>
<%
/* cos 라이브러리를 이용한 파일 업로드 구현하기
 *
 * 1. MultipartRequest의 생성자 매개변수에 지정할 데이터를 설정
 *
 * 파일이 저장될 폴더의 로컬 경로를 구한다.
 */
String realPath = application.getRealPath(uploadDir);

// 업로드 파일의 최대 크기를 100MB로 지정
int maxFileSize = 100 * 1024 * 1024;

// 파일의 인코딩 타입을 UTF-8로 지정
String encoding = "UTF-8";

/* 2. 파일 업로드를 처리할 MultipartRequest 객체 생성
 *
 * WEB-INF/lib/cos.jar 파일을 살펴보면 MultipartRequet 클래스는
 * com.oreilly.servlet 패키지에 위치하며 파일 업로드를 직접적으로 처리하는

```

```

* 역할을 담당하는 클래스로 파일 업로드와 관련된 다양한 메소드를 정의하고 있다.
* 생성자는 5개로 오버로딩 되어 있고 아래 생성자가 호출되도록 정의되어 있다.
*
* public MultipartRequest(HttpServletRequest request,
*     String saveDirectory,
*     int maxPostSize,
*     String encoding,
*     FileRenamePolicy policy) throws IOException {...}
*
* 이 생성자를 살펴보면 request, saveDirectory, maxPostSize는 필수사항으로
* 이 매개변수가 null이거나 0보다 작다면 생성자 안에서 예외를 발생시킨다.
*
* request : MultipartRequest에 연결할 사용자의 요청 정보가 담긴 객체
* saveDirectory : 업로드 된 파일을 저장할 서버의 디렉토리 지정
* maxPostSize : 업로드 파일의 최대 크기 지정
* encoding : 파일의 인코딩 방식 지정, 파일 이름이 한글일 경우 필히 utf-8 지정
* policy : 사용자가 업로드 한 파일을 저장할 서버의 디렉토리에 현재 업로드 되는
*     파일과 이름이 중복된 파일이 존재할 경우 현재 업로드 되는 파일의
*     이름을 어떻게 변경할지에 대한 정책을 지정하는 매개변수 이다.
*     일반적으로 new DefaultFileRenamePolicy()를 사용하며
*     이 클래스는 abc.jpg 파일을 업로드 할때 이미 같은 이름의 파일이
*     존재하면 자동으로 abc1.jpg와 같이 파일을 변경해 준다.
*
* 아래와 같이 MultipartRequest 객체를 생성하면 saveDirectory에 지정한
* 서버의 디렉토리로 파일이 바로 업로드 된다.
**/

```

```

MultipartRequest multi = new MultipartRequest(request, realPath,
    maxFileSize, encoding, new DefaultFileRenamePolicy());

```

```

/* 3. MultipartRequest 객체를 이용해 클라이언트로부터 요청된 데이터를 처리

```

```

*
* 파일 업로드 처리를 위해서는 모든 요청에 대한 처리를 MultipartRequest 객체를
* 이용해 접근해야 한다. 위에서 MultipartRequest 객체를 생성할 때 요청에 대한
* 정보를 담고 있는 request를 생성자의 매개변수로 지정해 MultipartRequest를
* 통해 사용자의 요청 정보에 접근할 수 있다.
*
* MultipartRequest 클래스에 정의된 주요 메소드는 아래와 같다.
* getParameter(name) : name에 지정한 파라미터 값을 반환
* getParameterNames() : 폼에서 전송된 모든 파라미터 이름을
*     Enumeration 타입으로 반환
* getParameterValues(name) : name에 지정한 파라미터 값을 String 배열로 반환
* getFile(fileName) : 업로드 된 파일 중에서 fileName에 지정한 파라미터
*     이름을 가진 파일의 정보를 File 객체로 반환
* getFileNames() : 폼에서 전송된 모든 파일의 이름을 Enumeration 타입으로 반환
* getFileSystemName(name) : name에 지정한 파라미터 이름을 가진

```

```

*                                     파일의 이름을 반환
* getOriginalFileName() : 사용자가 업로드 한 파일의 원본 이름을 반환
* getContentType() : 사용자가 업로드 한 파일의 컨텐트 타입을 반환
**/

/* 사용자가 폼에 입력한 데이터 처리
* MultipartRequest 객체를 통해 파라미터를 읽어 변수에 저장한다.
**/
String title = multi.getParameter("title");
String writer = multi.getParameter("writer");
String pass = multi.getParameter("pass");
String content = multi.getParameter("content");

/* 하나의 게시 글 정보를 저장하는 자바빈 객체를 생성하고 요청 파라미터로 받은
* 데이터를 Board 객체에 저장한다.
**/
Board board = new Board();
board.setTitle(title);
board.setWriter(writer);
board.setPass(pass);
board.setContent(content);

/* 사용자가 업로드한 파일 데이터 처리
* MultipartRequest 객체를 통해 파일 이름을 구하여 변수에 저장한다.
* 파일이 업로드 되지 않으면 fileName은 null 값을 받는다.
**/
String fileName = multi.getFilesystemName("file1");
System.out.println("업로드 된 파일명 : " + fileName);
System.out.println("원본 파일명 : " + multi.getOriginalFileName("file1"));

board.setFile1(fileName);
if(board.getFile1() == null) {
    System.out.println("파일이 업로드 되지 않았음");
}

// BoardDao 객체를 생성하고 DB에 새로운 게시 글을 추가한다.
BoardDao dao = new BoardDao();
dao.insertBoard(board);

/* 게시 글쓰기가 완료된 후 response 내장객체의 sendRedirect() 메서드를
* 이용해 게시 글 리스트로 Redirect 시킨다. response 내장객체의 sendRedirect()
* 메서드는 요청한 자원이 다른 곳으로 이동되었다고 응답하면서 URL을 알려주고
* 그 쪽으로 다시 요청하라고 응답하는 메소드이다. 브라우저가 요청한 컨텐츠가
* 이동했으니 그 쪽으로 다시 요청하라고 응답 데이터로 웹 주소를 알려주면
* 브라우저는 그 웹 주소로 다시 요청하게 되는데 이를 리다이렉션이라고 한다.

```

*

- * Redirect 기법은 웹 브라우저를 새로 고침(F5) 했을 때 동일한 코드가 다시
- * 실행되면 문제가 될 수 있는 경우 클라이언트의 요청을 처리한 후 특정 URL로
- * 이동시키기 위해 사용하는 기법이다. 예를 들어 게시 글쓰기에 대한 요청을 처리한
- * 후 Redirect 시키지 않으면 게시 글쓰기 후에 사용자가 새로 고침(F5) 동작을
- * 하면 바로 이전에 작성한 게시 글 내용과 동일한 내용을 다시 DB에 등록하는 작업을
- * 하게 되는데 이렇게 되면 중복된 데이터를 계속해서 저장하는 문제가 발생한다.
- * 이를 방지하기 위해서 게시 글쓰기가 완료되면 게시 글 리스트(select 문은 반복
- * 사용해도 중복된 데이터가 발생하지 않음)로 이동시키기 위해서 response
- * 내장객체의 sendRedirect() 메소드를 사용해 게시 글 리스트의 URL을
- * 웹 브라우저에게 응답하고 웹 브라우저는 응답 받은 URL로 다시 요청하도록 하는
- * 것이다. 이렇게 게시 글쓰기와 같이 DB 입력 작업이 연동되는 경우 사용자의
- * 새로 고침(F5) 동작에 의해 동일한 요청이 다시 발생하여 DB에 입력되는 데이터의
- * 중복이 발생하거나 SQLException을 발생 시킬 수 있어 Redirect 기법을
- * 사용한다. 이외에 다른 사이트로 이동시킬 때 Redirect 기법을 사용 한다.

**/

```
response.sendRedirect("boardList.jsp");
```

%>

3) 페이징 처리 구현

앞에서 구현한 게시 글 리스트는 DB 테이블에 존재하는 모든 게시 글을 가져와 화면에 출력하는 것이었다. 하지만 실제 게시 글의 수가 한 페이지에 표현하기에 많은 경우 한 페이지에 출력되는 게시 글의 수를 정하여 여러 페이지에 걸쳐서 출력될 수 있도록 프로그램을 설계해야 한다.

요구사항에 따라서 다양한 페이징 처리 기법이 나올 수 있지만 그 동안 많이 사용된 가장 보편적인 페이징 처리 기법에 대해서 알아볼 것이다. 페이징 처리 구현에 앞서 먼저 DB 테이블에 게시 글을 추가해야 하므로 JSPStudyBBS03 프로젝트의 webapp/WEB-INF/sql/JSPStudyBBS.sql 파일을 이용해 기존의 테이블과 시퀀스를 삭제하고 다시 게시 글 정보를 추가하자.

- 실습용 프로젝트 : JSPClassBBS03

- 완성 프로젝트 : JSPStudyBBS03

3-1) 페이징 처리 구현

게시 글 리스트의 페이징 처리를 위해서는 전체 게시 글의 개수가 몇 개인지 알아야 전체 페이지가 몇 페이지인지를 계산할 수 있으며 이를 이용해 페이지 내비게이션(페이지 링크)을 구현할 수 있다. 또한 이전에는 boardList.jsp에서 테이블에 저장된 모든 게시 글을 한 페이지 출력했지만 페이지 처리를 적용하면 한 페이지에 몇 개의 게시 글을 출력할지 결정하여 그 개수만큼 게시 글 리스트를 출력해야 한다. 그러므로 BoardDao의 boardList() 메서드도 한 페이지에 보여줄 게시 글 개수만큼만 DB에서 읽어와 반환하도록 구현해야 한다.

▶ BoardDao 클래스에 메서드 추가

- com.jspstudy.bbs.dao.BoardDao

```
/* 전체 게시 글 수를 계산하기 위해 호출되는 메서드 - paging 처리에 사용
 * DB 테이블에 등록된 모든 게시 글의 수를 반환하는 메서드
 **/
```

```
public int getBoardCount() {

    String sqlCount = "SELECT COUNT(*) FROM jspbbs";
    int count = 0;

    try{
        conn = ds.getConnection();
        pstmt = conn.prepareStatement(sqlCount);
        rs = pstmt.executeQuery();
        while(rs.next()) {
            count = rs.getInt(1);
        }
    } catch(Exception e) {
        e.printStackTrace();
    } finally {
```

```

    try {
        if(rs != null) rs.close();
        if(pstmt != null) pstmt.close();
        if(conn != null) conn.close();
    } catch(SQLException se) {}
}
return count;
}

```

▶ BoardDao 클래스에서 boardList() 메서드 수정

- com.jspstudy.bbs.dao.BoardDao

```

/* 한 페이지에 보여 질 게시 글 리스트 요청시 호출되는 메소드
 * 요청한 페이지에 해당하는 게시 글 리스트를 DB에서 읽어와 반환하는 메소드
 */
public ArrayList<Board> boardList(int startRow, int endRow) {

    /* 요청한 페이지에 해당하는 게시 글 리스트를 추출하기 위한 쿼리
     * 테이블에서 현재 페이지에 해당하는 게시 글을 검색할 때 ROWNUM을 사용했다.
     * ROWNUM은 쿼리의 결과로 검색되는 행들의 순서 값을 가진 의사컬럼으로
     * 1부터 시작한다. 최신 게시 글을 먼저 보여주기 위해 ORDER BY DESC를
     * 지정하고 요청된 페이지에 보여줄 게시 글의 시작 행과 마지막 행을 지정한다.
     *
     * 아래의 쿼리로 질의하게 되면 게시 글 전체를 글 번호에 해당하는 no를 기준으로
     * 내림차순 정렬하여 검색하고 WHERE 절에 지정한 첫 번째 Placeholder(?)에
     * 해당하는 시작 행 부터 두 번째 Placeholder(?)에 해당하는 마지막 행까지의
     * 게시 글을 추출할 수 있다.
     */
    String sqlBoardList = "SELECT * FROM (SELECT ROWNUM num,"
        + " no, title, writer, content, reg_date, read_count, pass, file1 FROM"
        + " (SELECT * FROM jspbbs ORDER BY no DESC)) "
        + " WHERE num >= ? AND num <= ?";

    ArrayList<Board> boardList = null;

    try{
        conn = ds.getConnection();
        pstmt = conn.prepareStatement(sqlBoardList);
        pstmt.setInt(1, startRow);
        pstmt.setInt(2, endRow);
        rs = pstmt.executeQuery();

        if(rs.next()) {

            boardList = new ArrayList<Board>();

```

```

do {
    Board board = new Board();
    board.setNo(rs.getInt("no"));
    board.setTitle(rs.getString("title"));
    board.setWriter(rs.getString("writer"));
    board.setContent(rs.getString("content"));
    board.setRegDate(rs.getTimestamp("reg_date"));
    board.setReadCount(rs.getInt("read_count"));
    board.setPass(rs.getString("pass"));
    board.setFile1(rs.getString("file1"));

    boardList.add(board);

} while(rs.next());
}
} catch(Exception e) {
    e.printStackTrace();

} finally {
    try {
        if(rs != null) rs.close();
        if(pstmt != null) pstmt.close();
        if(conn != null) conn.close();
    } catch(SQLException se) {}
}
return boardList;
}

```

▶ 게시 글 리스트 요청을 처리하는 JSP 페이지

- webapp/board/boardList.jsp 수정

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="com.jspstudy.bbs.dao.*,com.jspstudy.bbs.vo.*" %>
<%@ page import="java.util.*" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<%!
    // 한 페이지에 보여 줄 게시 글의 수를 상수로 선언하고 있다.
    final int PAGE_SIZE = 5;

    /* 한 페이지에 보여 질 페이지 그룹의 수를 상수로 선언하고 있다.
     * [이전] 1 2 3 4 5 6 7 8 9 10 [다음]
     */
    final int PAGE_GROUP = 10;

```

%>

<%

// 요청 파라미터로 넘어 온 페이지 번호를 읽어온다.

String pageNum = request.getParameter("pageNum");

/* pageNum이 null 이면 처음 게시 글 리스트를 요청하거나 게시 글쓰기에서

* Redirect 되어 넘어온 요청으로 pageNum을 1페이지로 설정한다.

* 즉 첫 번째 페이지에 해당하는 게시 글 리스트를 화면에 출력한다.

*/

if(pageNum == null) {

pageNum = "1";

}

// 요청 파라미터의 pageNum을 int 형으로 변환하여 현재 페이지로 설정

int currentPage = Integer.parseInt(pageNum);

/* 요청한 페이지에 해당하는 게시 글의 첫 번째 행의 값을 계산한다.

* 현재 페이지가 1일 경우 startRow는 1, 2페이지 일 경우 startRow는 6이 된다.

*

* 테이블에서 현재 페이지에 해당하는 게시 글을 검색할 때 ROWNUM을 사용했다.

* ROWNUM은 쿼리의 결과로 검색되는 행들의 순서 값을 가진 의사컬럼으로

* 1부터 시작한다. 예를 들어 3페이지에 해당하는 게시 글 리스트를 가져 온다면

* 한 페이지에 보여줄 게시 글의 수가 5개로 지정되어 있으므로 startRow는 11이 된다.

* 즉 아래의 공식에 의해 startRow(11) = 3 * 5 - (5 - 1);

* 첫 번째 페이지 startRow = 1, 두 번째 페이지 startRow = 6이 된다.

*/

int startRow = currentPage * PAGE_SIZE - (PAGE_SIZE - 1);

//int startRow = (currentPage - 1) * PAGE_SIZE + 1;

int endRow = startRow + PAGE_SIZE - 1;

int listCount = 0;

// BoardDao 객체를 생성하고 게시 글 개수를 읽어온다.

BoardDao dao = new BoardDao();

listCount = dao.getBoardCount();

// 현재 페이지에 해당하는 게시 글 리스트를 DB에서 읽어온다.

ArrayList<Board> bList = dao.boardList(startRow, endRow);

/* 페이지 그룹 이동 처리를 위해 게시판의 전체 페이지 수를 계산한다.

* [이전] 11 12 13... 또는 ... 8 9 10 [다음] 처리

* 전체 페이지 = 전체 게시 글 수 / 한 페이지에 표시할 게시 글 수가 되는데

* 이 계산식에서 나머지가 존재하면 전체 페이지 수는 전체 페이지 + 1이 된다.

*/

int pageCount = listCount / PAGE_SIZE


```

+ (listCount % PAGE_SIZE == 0 ? 0 : 1);

/* 페이지 그룹 처리를 위해 페이지 그룹별 시작 페이지와 마지막 페이지를 계산하여
 * 페이지 그룹의 시작 페이지 : 1, 11, 21, 31...
 * 첫 번째 페이지 그룹에서 페이지 리스트는 1 ~ 10 되므로 currentPage가
 * 1 ~ 10 사이에 있으면 startPage는 1이 되고 11 ~ 20 사이는 11이 된다.
 *
 * 정수형 연산의 특징을 이용해 startPage를 아래와 같이 구할 수 있다.
 * 아래 계산식으로 계산된 결과를 보면 현재 그룹의 마지막 페이지일 경우
 * startPage가 다음 그룹의 시작 페이지가 나오게 되므로 삼항 연산자를
 * 사용해 현재 페이지가 속한 그룹의 startPage가 되도록 조정 하였다.
 */
int startPage = (currentPage / PAGE_GROUP) * PAGE_GROUP + 1
- (currentPage % PAGE_GROUP == 0 ? PAGE_GROUP : 0);

// 현재 페이지 그룹의 마지막 페이지 : 10, 20, 30...
int endPage = startPage + PAGE_GROUP - 1;

/* 위의 식에서 endPage를 구하게 되면 endPage는 항상 PAGE_GROUP의
 * 크기만큼 증가(10, 20, 30 ...) 되므로 맨 마지막 페이지 그룹의 endPage가
 * 정확하지 못할 경우가 발생하게 된다. 다시 말해 전체 페이지가 53페이지 라고
 * 가정하면 위의 식에서 계산된 endPage는 60 페이지가 되지만 실제로
 * 60페이지는 존재하지 않는 페이지이므로 문제가 발생하게 된다.
 * 그래서 맨 마지막 페이지에 대한 보정이 필요하여 아래와 같이 endPage와
 * pageCount를 비교하여 현재 페이지 그룹에서 endPage가 pageCount 보다
 * 크다면 pageCount를 endPage로 지정 하였다. 즉 현재 페이지 그룹이
 * 마지막 페이지 그룹이면 endPage는 전체 페이지 수가 되도록 지정한 것이다.
 */
if(endPage > pageCount) {
    endPage = pageCount;
}

%>
<!-- 아래에서 EL로 접근하기 위해서 pageContext 영역의 속성으로 저장 -->
<c:set var="bList" value="%= bList %>" />
<c:set var="currentPage" value="%= currentPage %>" />
<c:set var="pageGroup" value="%= PAGE_GROUP %>" />
<c:set var="pageCount" value="%= pageCount %>" />
<c:set var="startPage" value="%= startPage %>" />
<c:set var="endPage" value="%= endPage %>" />
<!DOCTYPE html>
<html lang="ko">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>게시 글 리스트</title>

```

```

<link href="../../bootstrap/bootstrap.min.css" rel="stylesheet" >
<script src="../../js/jquery-3.3.1.min.js"></script>
<script src="../../js/formcheck.js"></script>
</head>
<body>
<div class="container">
  <%@ include file="../../pages/header.jsp" %>
  <!-- content -->
<div class="row my-5" id="global-content">
  <div class="col">
    <div class="row text-center">
      <div class="col">
        <h2 class="fs-3 fw-bold">게시 글 리스트</h2>
      </div>
    </div>
    <form name="searchForm" id="searchForm" action="#"
      class="row justify-content-center my-3">
      <div class="col-auto">
        <select name="type" class="form-select">
          <option value="title">제목</option>
          <option value="writer">작성자</option>
          <option value="content">내용</option>
        </select>
      </div>
      <div class="col-4">
        <input type="text" name="keyword" class="form-control"/>
      </div>
      <div class="col-auto">
        <input type="submit" value="검 색" class="btn btn-primary"/>
      </div>
    </form>
    <div class="row">
      <div class="col text-end">
        <a href="writeForm.jsp" class="btn btn-outline-success">글쓰기</a>
      </div>
    </div>
    <div class="row my-3">
      <div class="col">
        <table class="table table-hover">
          <thead>
            <tr class="table-dark">
              <th>NO</th>
              <th>제목</th>
              <th>작성자</th>
              <th>작성일</th>
            </tr>
          </thead>
        </table>
      </div>
    </div>
  </div>
</div>

```

```

        <th>조회수</th>
    </tr>
</thead>
<tbody class="text-secondary">
    <!-- 게시 글이 있는 경우 -->
    <c:if test="${ not empty bList }">
        <c:forEach var="b" items="${bList}" varStatus="status">
            <tr>
                <td>${ b.no }</td>
                <td><a href="boardDetail.jsp?no=${b.no}&pageNum=${currentPage}"
                    class="text-decoration-none link-secondary">${ b.title }</a></td>
                <td>${ b.writer }</td>
                <td>${ b.regDate }</td>
                <td>${ b.readCount }</td>
            </tr>
        </c:forEach>
    </c:if>
    <!-- 게시 글이 없는 경우 -->
    <c:if test="${ empty bList }">
        <tr>
            <td colspan="5" class="text-center">게시 글이 존재하지 않습니다.</td>
        </tr>
    </c:if>
</tbody>
</table>
</div>
</div>
<div class="row">
    <div class="col">
        <nav aria-label="Page navigation">
            <ul class="pagination justify-content-center">
                <%--
                /* 현재 페이지 그룹의 시작 페이지가 pageGroup보다 크다는 것은
                * 이전 페이지 그룹이 존재한다는 것으로 현재 페이지 그룹의 시작 페이지에
                * pageGroup을 마이너스 하여 링크를 설정하면 이전 페이지 그룹의
                * startPage로 이동할 수 있다.
                **/
                --%>
                <c:if test="${ startPage > pageGroup }">
                    <li class="page-item">
                        <a class="page-link" href="boardList.jsp?pageNum=${ startPage -
pageGroup }">Pre</a>
                    </li>
                </c:if>
                <%--

```

```

/* 현재 페이지 그룹의 startPage 부터 endPage 만큼 반복하면서
 * 현재 페이지와 같은 그룹에 속한 페이지를 출력하고 링크를 설정한다.
 * 현재 페이지는 링크를 설정하지 않는다.
 */
--%>
<c:forEach var="i" begin="${startPage}" end="${endPage}">
    <c:if test="${i == currentPage}">
        <li class="page-item active" aria-current="page">
            <span class="page-link">${i}</span>
        </li>
    </c:if>
    <c:if test="${i != currentPage}">
        <li class="page-item"><a class="page-link"
href="boardList.jsp?pageNum=${ i }">${i}</a></li>
    </c:if>
</c:forEach>

<%--
/* 현재 페이지 그룹의 마지막 페이지가 전체 페이지 보다 작다는 것은
 * 다음 페이지 그룹이 존재한다는 것으로 현재 페이지 그룹의 시작 페이지에
 * pageGroup을 플러스 하여 링크를 설정하면 다음 페이지 그룹의
 * startPage로 이동할 수 있다.
 */
--%>
<c:if test="${endPage < pageCount}">
    <li class="page-item">
        <a class="page-link" href="boardList.jsp?pageNum=${ startPage +
pageGroup }">Next</a>
    </li>
</c:if>
</ul>
</nav>
</div>
</div>
</div>
</div>
<%@ include file="../pages/footer.jsp" %>
</div>
<script src="../bootstrap/bootstrap.bundle.min.js"></script>
</body>
</html>

```

▶ 게시 글 상세보기

게시 글 리스트 페이지링 처리에 따라서 게시 글 상세보기에서 게시 글 리스트로 이동할 때 페이지

번호에 해당하는 게시 글 리스트로 이동시켜야 한다. 그렇지 않으면 사용자는 게시 글 리스트의 첫 페이지에서 다시 해당 페이지로 이동하는 수고를 해야 하기 때문이다. 그래서 게시 글 목록으로 이동하는 링크를 아래와 같이 수정해야 한다. 또한 게시 글 상세페이지를 요청할 때 no와 pageNum에 대한 정보가 요청 파라미터로 넘어오지 않는다면 이는 의도하지 않은 요청이기 때문에 이 정보를 체크해서 no와 pageNum이 비어 있으면 비정상적인 요청이므로 이전 페이지로 돌려보내는 코드도 추가하였다. BoardDao의 getBoard() 메서드는 앞의 예제와 동일하다.

아래 코드에서 볼드체로 표시된 부분이 추가되거나 수정된 코드이다.

- webapp/board/boardDetail.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="com.jspstudy.bbs.vo.*, com.jspstudy.bbs.dao.*" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%
    //요청 파라미터로 넘어 온 게시 글 번호와 페이지 번호를 읽어온다.
    String no = request.getParameter("no");
    String pageNum = request.getParameter("pageNum");

    // no와 pageNum이 비어 있으면 비정상 요청임
    if(no == null || no.equals("") || pageNum == null || pageNum.equals("")) {
        out.println("<script>");
        out.println(" alert('잘못된 접근입니다.');"");
        out.println(" history.back();"");
        out.println("</script>");
        return;
    }

    // BoardDao 객체를 생성하고 no에 해당하는 게시 글 하나를 가져온다.
    BoardDao dao = new BoardDao();
    Board board = dao.getBoard(Integer.valueOf(no));
%>
<c:set var="board" value="<%= board %>" scope="page" />
<!DOCTYPE html>
<html lang="ko">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title>게시 글 상세보기</title>
        <link href="../bootstrap/bootstrap.min.css" rel="stylesheet" >
        <script src="../js/jquery-3.3.1.min.js"></script>
        <script src="../js/formcheck.js"></script>
    </head>
    <body>
        <div class="container">
```

```
<%@ include file="../pages/header.jsp" %>
<!-- content -->
<div class="row my-5" id="global-content">
    <div class="col">
        <form name="checkForm" id="checkForm">
            <input type="hidden" name="no" id="no" value="{ board.no }"/>
            <input type="hidden" name="pass" id="rPass" />
            <input type="hidden" name="pageNum" value="{%= pageNum %}" />
        </form>
        <div class="row text-center">
            <div class="col">
                <h2 class="fs-3 fw-bold">게시 글 상세보기</h2>
            </div>
        </div>
    </div>
    <div class="row my-3">
        <div class="col">
            <table class="table table-bordered" >
                <tbody>
                    <tr>
                        <th class="table-secondary">제 목</th>
                        <td colspan="3">{ board.title }</td>
                    </tr>
                    <tr>
                        <th>글쓴이</th>
                        <td>{ board.writer }</td>
                        <th>작성일</th>
                        <td>{fmt:formatDate value="{ board.regDate }"
                            pattern="yyyy-MM-dd HH:mm:ss" /}</td>
                    </tr>
                    <tr>
                        <th>비밀번호</th>
                        <td>
                            <div class="col-sm-8">
                                <input class="form-control" type="password" name="pass" id="pass">
                            </div>
                        </td>
                        <th>조회수</th>
                        <td>{ board.readCount }</td>
                    </tr>
                    <tr>
                        <th>파일</th>
                        <td colspan="3">
                            <c:if test="{ empty board.file1 }">
                                첨부파일 없음
                            </c:if>
                        </td>
                    </tr>
                </tbody>
            </table>
        </div>
    </div>

```

```

        <c:if test="${ not empty board.file1 }">
            <a href="../../upload/${ board.file1 }">파일 다운로드</a>
        </c:if>
    </td>
</tr>
<tr>
    <td colspan="4">
        <pre>${ board.content }</pre>
    </td>
</tr>
</tbody>
</table>
</div>
</div>
<div class="row my-3">
    <div class="col text-center">
        <input class="btn btn-warning" type="button" id="detailUpdate" value="수정하기" />

        &nbsp;&nbsp;&nbsp;<input class="btn btn-danger" type="button" id="detailDelete"
value="삭제하기" />

        &nbsp;&nbsp;&<input class="btn btn-primary" type="button" value="목록보기"
onclick="location.href='boardList.jsp?pageNum=<%= pageNum %>'" />
    </div>
</div>
</div>
</div>
<div>
    <%@ include file="../../pages/footer.jsp" %>
</div>
<script src="../../bootstrap/bootstrap.bundle.min.js"></script>
</body>
</html>

```

▶ 게시 글 수정하기

게시 글 리스트 페이지 처리에 따라서 게시 글 상세보기와 마찬가지로 게시 글이 수정된 후에 사용자가 이전에 머물렀던 게시 글 리스트의 동일한 페이지로 이동시키기 위해서 pageNum 이라는 파라미터가 필요하다. 앞에서는 게시 글 수정에서 파일 업로드 기능을 구현하지 않았었는데 이번 예제는 게시 글 수정시에 기존에 첨부 파일이 존재하지 않으면 파일 업로드를 할 수 있도록 추가되었다. 사용자가 게시 글 수정 폼에서 파일을 선택한 경우와 파일을 선택하지 않고 폼을 전송하는 두 가지 경우에 대해서 updateProcess.jsp에서 처리하였다. 이번에도 BoardDao의 getBoard() 메서드는 앞의 예제와 동일하다.

아래에서 볼드체로 표시되는 부분이 추가되거나 수정된 코드이다.

- webapp/board/updateForm.jsp

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

```

        pageEncoding="UTF-8"%>
<%@ page import="com.jspstudy.bbs.dao.*, com.jspstudy.bbs.vo.*" %>
<%
    String sNo = request.getParameter("no");
    String pass = request.getParameter("pass");
    String pageNum = request.getParameter("pageNum");

    // no와 pass 그리고 pageNum이 비어 있으면 비정상 요청임
    if(sNo == null || sNo.equals("") || pass == null || pass.equals("")
        || pageNum == null || pageNum.equals("")) {
        out.println("<script>");
        out.println(" alert('잘못된 접근입니다.');
```



```

<div class="col">
  <h2 class="fs-3 fw-bold">게시 글 수정하기</h2>
</div>
</div>
<form name="updateForm" action="updateProcess.jsp" id="updateForm"
  class="row g-3 border-primary" method="post"
  ${ not empty board.file1 ? "" : "enctype='multipart/form-data'"}>

  <!--
    no는 DB에서 게시 글을 수정하기 위해 필요하고 pageNum은 게시 글이
    수정된 후에 이전에 사용자가 머물렀던 게시 글 리스트의 동일한 페이지로
    보내기 위해 필요한 정보이다.
  -->
  <input type="hidden" name="no" value="<%= board.getNo() %>">
  <input type="hidden" name="pageNum" value="<%= pageNum %>" />
  <div class="col-4 offset-md-2">
    <label for="writer" class="form-label">글쓴이</label>
    <input type="text" class="form-control" name="writer" id="writer"
      placeholder="작성자 입력해 주세요" value="<%= board.getWriter() %>">
  </div>
  <div class="col-4 ">
    <label for="pass" class="form-label">비밀번호</label>
    <input type="password" class="form-control" name="pass"
      id="pass">
  </div>
  <div class="col-8 offset-md-2">
    <label for="title" class="form-label">제 목</label>
    <input type="text" class="form-control" name="title"
      id="title" value="<%= board.getTitle() %>">
  </div>
  <div class="col-8 offset-md-2">
    <label for="content" class="form-label">내 용</label>
    <textarea class="form-control" name="content" id="content"
      rows="10"><%= board.getContent() %></textarea>
  </div>
  <div class="col-8 offset-md-2">
    <label for="file1" class="form-label">파 일</label>
    <input type="file" class="form-control" name="file1" id="file1"
      <%= board.getFile1() == null ? "" : "disabled" %>>
  </div>
  <div class="col-8 offset-md-2 text-center mt-5">
    <input type="submit" value="수정하기" class="btn btn-primary"/>
    &nbsp;&nbsp;&nbsp;<input type="button" value="목록보기"
      onclick="location.href='boardList.jsp?pageNum=<%=
      pageNum      %>'
class="btn btn-primary"/>

```

```

        </div>
    </form>
</div>
</div>
<%@ include file="../pages/footer.jsp" %>
</div>
<script src="../bootstrap/bootstrap.bundle.min.js"></script>
</body>
<%
/* 게시 글 번호에 해당하는 비밀번호가 틀리면 비밀번호가 틀리다. 라고 알려주고
 * 브라우저의 history 객체를 이용해 바로 이전에 있었던 주소로 돌려보낸다.
 */
} else {
%>
<script>
    alert("비밀번호가 다릅니다.");
    history.back();
</script>
<%}%>
</html>

```

- webapp/board/updateProcess.jsp

게시 글 리스트 페이징 처리에 따라서 게시 글 상세보기와 마찬가지로 게시 글이 수정된 후에 사용자가 이전에 머물렀던 게시 글 리스트의 동일한 페이지로 이동시키기 위해서 pageNum 이라는 파라미터가 필요하다. 그리고 앞에서는 게시 글쓰기에만 파일 업로드를 구현했지만 이번 예제에서는 기존에 첨부파일이 존재하지 않으면 첨부 파일을 선택할 수 있도록 하여 사용자가 게시 글 수정 폼에서 파일을 선택한 경우와 파일을 선택하지 않고 폼을 전송하는 경우의 두 가지 상황에 대해서 처리하였다. BoardDao 클래스의 updateBoard() 메서드는 앞의 예제와 동일하다.

앞의 예제에서 많은 부분이 추가되거나 수정되었기 때문에 별도로 강조하지 않았다.

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="com.oreilly.servlet.*, com.oreilly.servlet.multipart.*" %>
<%@ page import="java.sql.*, java.io.*" %>
<%@ page import="com.jspstudy.bbs.vo.*, com.jspstudy.bbs.dao.*" %>
<%!
    static String uploadDir;
    static File parentFile;

    public void jsplnit() {
        // web.xml에 지정한 웹 어플리케이션 초기화 파라미터를 읽는다.
        uploadDir = getServletContext().getInitParameter("uploadDir");

        /* 웹 어플리케이션 초기화 파라미터로 읽어온 이미지가 저장될 폴더의

```

```

    * 로컬 경로를 구하여 그 경로와 파일명으로 File 객체를 생성한다.
    **/
String realPath = getServletContext().getRealPath(uploadDir);
parentFile = new File(realPath);

/* 파일 객체에 지정한 위치에 디렉토리가 존재하지 않거나
 * 파일 객체가 디렉토리가 아니라면 디렉토리를 생성한다.
 **/
if(! (parentFile.exists() && parentFile.isDirectory())) {
    parentFile.mkdir();
}
System.out.println("init - " + parentFile);
}
%>
<%

String contentType = request.getHeader("Content-Type");
System.out.println("contentType : " + contentType);

// BoardDao 객체 생성
BoardDao dao = new BoardDao();
Board board = null;
String pass= null, title = null, writer = null, content = null,
    sNo = null, pageNum = null;
int no = 0;

/* 비밀번호가 맞지 않을 경우 먼저 처리해야 하나 Content-Type에
 * 따라서 파라미터를 읽을 수 있는 객체가 다르기 때문에 각각 처리해야 한다.
 * Content-Type이 application/x-www-form-urlencoded 일 경우
 * request.getParameter() 메서드로 파라미터를 읽을 수 있지만
 * multipart/form-data 일 경우에는 request 객체를 이용해
 * 파라미터를 읽을 수 없기 때문에 각각 처리하는 코드를 기술해야 한다.
 **/
// 요청이 multipart/form-data 일 경우
if(contentType.contains("multipart/form-data")) {

    /* cos 라이브러리를 이용한 파일 업로드 구현하기
    *
    * 1. MultipartRequest의 생성자 매개변수에 지정할 데이터를 설정
    *
    * 파일이 저장될 폴더의 로컬 경로를 구한다.
    **/
String realPath = application.getRealPath(uploadDir);

// 업로드 파일의 최대 크기를 100MB로 지정
int maxFileSize = 100 * 1024 * 1024;

```

// 파일의 인코딩 타입을 UTF-8로 지정

String encoding = "UTF-8";

/* 2. 파일 업로드를 처리할 MultipartRequest 객체 생성

*

* WEB-INF/lib/cos.jar 파일을 살펴보면 MultipartRequest 클래스는

* com.oreilly.servlet 패키지에 위치하며 파일 업로드를 직접적으로 처리하는

* 역할을 담당하는 클래스로 파일 업로드와 관련된 다양한 메소드를 정의하고 있다.

* 생성자는 5개로 오버로딩 되어 있고 아래 생성자가 호출되도록 정의되어 있다.

*

* public MultipartRequest(HttpServletRequest request,

* String saveDirectory,

* int maxPostSize,

* String encoding,

* FileRenamePolicy policy) throws IOException {...}

*

* 이 생성자를 살펴보면 request, saveDirectory, maxPostSize는 필수사항으로

* 이 매개변수가 null이거나 0보다 작다면 생성자 안에서 예외를 발생시킨다.

*

* request : MultipartRequest에 연결할 사용자의 요청 정보가 담긴 객체

* saveDirectory : 업로드 된 파일을 저장할 서버의 디렉토리 지정

* maxPostSize : 업로드 파일의 최대 크기 지정

* encoding : 파일의 인코딩 방식 지정, 파일 이름이 한글일 경우 필히 utf-8 지정

* policy : 사용자가 업로드 한 파일을 저장할 서버의 디렉토리에 현재 업로드 되는

* 파일과 이름이 중복된 파일이 존재할 경우 현재 업로드 되는 파일의

* 이름을 어떻게 변경할지에 대한 정책을 지정하는 매개변수 이다.

* 일반적으로 new DefaultFileRenamePolicy()를 사용하며

* 이 클래스는 abc.jpg 파일을 업로드 할때 이미 같은 이름의 파일이

* 존재하면 자동으로 abc1.jpg와 같이 파일을 변경해 준다.

*

* 아래와 같이 MultipartRequest 객체를 생성하면 saveDirectory에 지정한

* 서버의 디렉토리로 파일이 바로 업로드 된다.

**/

MultipartRequest multi = new MultipartRequest(request, realPath,
maxFileSize, encoding, new DefaultFileRenamePolicy());

/* 3. MultipartRequest 객체를 이용해 클라이언트로부터 요청된 데이터를 처리

*

* 파일 업로드 처리를 위해서는 모든 요청에 대한 처리를 MultipartRequest 객체를

* 이용해 접근해야 한다. 위에서 MultipartRequest 객체를 생성할 때 요청에 대한

* 정보를 담고 있는 request를 생성자의 매개변수로 지정해 MultipartRequest를

* 통해 사용자의 요청 정보에 접근할 수 있다.

*

* MultipartRequest 클래스에 정의된 주요 메소드는 아래와 같다.

```

* getParameter(name) : name에 지정한 파라미터 값을 반환
* getParameterNames() : 폼에서 전송된 모든 파라미터 이름을
*
Enumeration 타입으로 반환
* getParameterValues(name) : name에 지정한 파라미터 값을 String 배열로 반환
* getFile(fileName) : 업로드 된 파일 중에서 fileName에 지정한 파라미터
*
이름을 가진 파일의 정보를 File 객체로 반환
* getFileNames() : 폼에서 전송된 모든 파일의 이름을 Enumeration 타입으로 반환
* getFileSystemName(name) : name에 지정한 파라미터 이름을 가진
*
파일의 이름을 반환
* getOriginalFileName() : 사용자가 업로드 한 파일의 원본 이름을 반환
* getContentType() : 사용자가 업로드 한 파일의 콘텐츠 타입을 반환
**/

/* 사용자가 폼에 입력한 데이터 처리
* MultipartRequest 객체를 통해 파라미터를 읽어 변수에 저장한다.
**/
sNo = multi.getParameter("no");
pass = multi.getParameter("pass");
pageNum = multi.getParameter("pageNum");

// no와 pass 그리고 pageNum이 비어 있으면 비정상 요청임
if(sNo == null || sNo.equals("") || pass == null || pass.equals("")
|| pageNum == null || pageNum.equals("")) {
    out.println("<script>");
    out.println(" alert('잘못된 접근입니다.');

```

```

title = multi.getParameter("title");
writer = multi.getParameter("writer");
content = multi.getParameter("content");

/* 하나의 게시 글 정보를 저장하는 자바빈 객체를 생성하고 파라미터로
 * 넘겨받은 요청 데이터를 Board 객체에 저장한다.
 */
board = new Board();
board.setNo(no);
board.setTitle(title);
board.setWriter(writer);
board.setPass(pass);
board.setContent(content);

/* 사용자가 업로드한 파일 데이터 처리
 * MultipartRequest 객체를 통해 파일 이름을 구하여 변수에 저장한다.
 */
String fileName = multi.getFilesystemName("file1");
System.out.println("업로드 된 파일명 : " + fileName);
System.out.println("원본 파일명 : " + multi.getOriginalFileName("file1"));

board.setFile1(fileName);
if(board.getFile1() == null) {
    System.out.println("파일이 업로드 되지 않았음");
}

// 요청이 multipart/form-data 아닌 경우
} else {

    request.setCharacterEncoding("utf-8");
    sNo = request.getParameter("no");
    pass = request.getParameter("pass");
    pageNum = request.getParameter("pageNum");

    // no와 pass 그리고 pageNum이 비어 있으면 비정상 요청임
    if(sNo == null || sNo.equals("") || pass == null || pass.equals("")
        || pageNum == null || pageNum.equals("")) {
        out.println("<script>");
        out.println("alert('잘못된 접근입니다.');

```

```

// 게시 글의 비밀번호를 체크해 맞지 않으면 이전으로 돌려보낸다.
boolean isPassCheck = dao.isPassCheck(no, pass);
if(! isPassCheck) {
    System.out.println("비밀번호 맞지 않음");
    StringBuilder sb = new StringBuilder();
    sb.append("<script>");
    sb.append(" alert('비밀번호가 맞지 않습니다.');

```

▶ 게시 글 삭제하기

게시 글 리스트 페이지징 처리에 따라서 게시 글 상세보기와 마찬가지로 게시 글이 삭제된 후에 사용자가 이전에 머물렀던 게시 글 리스트의 동일한 페이지로 이동시키기 위해서 pageNum 이라는 파

라미터가 필요하다. BoardDao 클래스의 deleteBoard() 메서드는 앞의 예제와 동일하다. 아래에서 볼드체로 표시되는 부분이 추가되거나 수정된 코드이다.

- webapp/board/deleteProcess.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="java.sql.*, java.io.*" %>
<%@ page import="com.jspstudy.bbs.vo.*, com.jspstudy.bbs.dao.*" %>
<%

String sNo = request.getParameter("no");
String pass = request.getParameter("pass");
String pageNum = request.getParameter("pageNum");

// no와 pass 그리고 pageNum이 비어 있으면 비정상 요청임
if(sNo == null || sNo.equals("") || pass == null || pass.equals("")
    || pageNum == null || pageNum.equals("")) {
    out.println("<script>");
    out.println(" alert('잘못된 접근입니다.');"");
    out.println(" history.back();"");
    out.println("</script>");
    return;
}

// BoardDao 객체 생성
BoardDao dao = new BoardDao();
int no = Integer.parseInt(sNo);

// 게시 글의 비밀번호를 체크해 맞지 않으면 이전으로 돌려보낸다.
boolean isPassCheck = dao.isPassCheck(no, pass);
if(! isPassCheck) {
    System.out.println("비밀번호 맞지 않음");
    StringBuilder sb = new StringBuilder();
    sb.append("<script>");
    sb.append(" alert('비밀번호가 맞지 않습니다.');"");
    sb.append(" history.back();"");
    sb.append("</script>");
    out.println(sb.toString());
    return;
}

// BoardDao 객체를 이용해 게시 글을 삭제한다.
dao.deleteBoard(no);

/* DB에 게시 글을 삭제하고 브라우저에게 게시 글 리스트를 요청하라고 응답
 * 게시 글 삭제가 완료된 후 Redirect 시키지 않으면 이 페이지를 새로 고침 하여
```


- * 재요청 할 때 마다 이미 삭제된 게시 글을 계속해서 삭제하려고 하는 문제가 발생한다.
 - *
 - * 리다이렉트 할 때 게시 글 리스트의 페이지 번호를 파라미터로 넘기고 있다.
- **/

```
response.sendRedirect("boardList.jsp?pageNum=" + pageNum);
```

```
%>
```

4) 게시판 검색 기능 추가하기

- 실습용 프로젝트 : JSPClassBBS04
- 완성 프로젝트 : JSPStudyBBS04

4-1) 게시판 검색 기능 구현

이번에는 게시 글 리스트에 검색 기능을 추가해 사용자가 원하는 검색어를 입력해 검색하면 해당 검색어가 포함된 게시 글 리스트만 볼 수 있도록 구현해 보자. 검색 기능은 게시 글 제목, 글쓴이, 내용을 선택하고 검색어를 입력하면 각각에 해당하는 부분에 사용자가 입력한 검색어가 포함된 게시 글만 리스트로 보여주는 기능을 구현할 것이다.

앞에서 구현한 boardList.jsp에 게시 글 검색 기능을 추가하게 되면 일반적인 게시 글 리스트인지 아니면 검색 결과에 대한 게시 글 리스트인지를 구분하여 처리해야 한다. 예를 들면 일반 게시 글 리스트에서 게시 글 상세보기로 갔다가 다시 오는 것은 별로 문제될 것이 없지만 특정 검색어로 검색된 게시 글 리스트에서 상세보기로 갔다가 다시 리스트로 돌아가기 위해서는 이전의 게시 글 리스트 즉 특정 검색어에 해당하는 게시 글 리스트여야 하므로 이런 부분의 처리를 위해서 링크와 같은 부분이 조금은 복잡해 질 수 있고 보다 세밀한 처리가 필요하다.

앞에서 구현한 게시 글 리스트의 페이징 처리와 마찬가지로 게시 글 검색 리스트도 해당 검색어에 해당하는 페이징 처리를 위해서는 해당 검색어에 해당하는 게시 글의 개수를 알아야 한다. 그리고 한 페이지에 몇 개의 게시 글을 출력할지 결정하여 그 개수만큼 해당 검색어에 해당하는 게시 글을 화면에 출력해야 한다. 그러므로 DB에 해당 검색어가 포함된 게시 글 수가 몇 개인지를 반환하는 메서드와 해당 검색어가 포함된 한 페이지에 보여줄 게시 글 리스트를 DB에서 읽어와 반환하는 메서드가 필요하다.

▶ BoardDao 클래스에 메서드 추가

- com.jspstudy.bbs.dao.BoardDao

```
/* 검색어에 해당하는 게시 글 수를 계산하기 위해 호출되는 메서드
 * DB 테이블에서 해당 검색어가 포함된 게시 글의 수를 반환하는 메서드
 */
public int getBoardCount(String type, String keyword) {
    System.out.println(type + " - " + keyword);

    /* 이 부분에서 우리는 SQL 쿼리를 작성하는데 주의를 기울여야 한다.
     * 검색 옵션에 따라서 검색하는 컬럼이 다르기 때문에 파라미터로 받은 type을
     * 아래와 같이 쿼리의 파라미터(?)로 지정해 검색 옵션의 변경에 따라 동적으로
     * 다른 쿼리가 발행되게 하고 싶지만 이 방법은 제대로 동작하지 못한다.
     *
     * "SELECT COUNT(*) FROM jspbbs WHERE ? LIKE '%' || ? || '%"
     *
     *
     * PreparedStatement는 기준이 되는 SQL 쿼리를 캐싱하기 때문에
     * 이 객체를 Connection 객체로부터 받아 올 때 기준 쿼리를 인수로 지정하는데
     * 이 때 WHERE에 검색 대상이 되는 컬럼명을 파라미터로(?) 지정하게 되면
     * 쿼리가 캐싱될 때 검색 대상이 되는 컬럼이 지정되지 못하게 되므로 이 쿼리의
```

```

* 검색 결과는 항상 0이 된다.
*
* 기준 쿼리에서 사용되는 컬럼은 PreparedStatement 객체를 생성할 때
* 정해져 있어야하기 때문에 아래와 같이 type을 지정하면 된다.
**/
String sqlCount = "SELECT COUNT(*) FROM jspbbs WHERE "
    + type + " LIKE '%" + keyword + "%'";
//String sqlCount =
// "SELECT COUNT(*) FROM jspbbs WHERE " + type + " LIKE ?";
int count = 0;

try{
    conn = ds.getConnection();
    pstmt = conn.prepareStatement(sqlCount);
    pstmt.setString(1, keyword);
    //pstmt.setString(1, "%" + keyword + "%");
    rs = pstmt.executeQuery();
    if(rs.next()) {
        count = rs.getInt(1);
    }
} catch(Exception e) {
    e.printStackTrace();
} finally {
    try {
        if(rs != null) rs.close();
        if(pstmt != null) pstmt.close();
        if(conn != null) conn.close();
    } catch(SQLException se) {}
}
return count;

} // end getBoardCount(String type, String keyword);

/* 제목, 작성자, 내용에서 검색어가 포함된 게시 글 검색 시 호출되는 메서드
* 요청한 페이지에 해당하는 검색 결과를 DB에서 읽어와 반환하는 메서드
**/
public ArrayList<Board> searchList(
    String type, String keyword, int startRow, int endRow) {

    /* 검색어가 포함된 게시 글 리스트를 추출하기 위한 쿼리
    * 테이블에서 현재 페이지에 해당하는 게시 글을 검색할 때 ROWNUM을 사용했다.
    * ROWNUM은 쿼리의 결과로 검색되는 행들의 순서 값을 가진 의사컬럼으로
    * 1부터 시작한다. 최신 게시 글을 먼저 보여주기 위해 ORDER BY DESC를
    * 지정하고 요청된 페이지에 보여줄 게시 글의 시작 행과 마지막 행을 지정한다.

```

```

*
* 아래의 쿼리로 질의하게 되면 게시 글 전체를 글 번호에 해당하는 no를 기준으로
* 내림차순 정렬하여 검색하고 WHERE 절에 지정한 첫 번째 Placeholder(?)에
* 해당하는 시작 행부터 두 번째 Placeholder(?)에 해당하는 마지막 행까지의
* 게시 글을 추출할 수 있다.
**/

```

```

String sqlSearchList = "SELECT * FROM (SELECT ROWNUM num, no, title,"
    + " writer, content, reg_date, read_count, pass, file1 FROM"
    + " (SELECT * FROM jspbbs WHERE " + type + " LIKE ?"
    + " ORDER BY no DESC)) WHERE num >= ? AND num <= ?";

```

```

ArrayList<Board> boardList = null;

```

```

try{
    conn = ds.getConnection();
    pstmt = conn.prepareStatement(sqlSearchList);
    pstmt.setString(1, "%" + keyword + "%");
    pstmt.setInt(2, startRow);
    pstmt.setInt(3, endRow);
    rs = pstmt.executeQuery();

    if(rs.next()) {

        boardList = new ArrayList<Board>();

        do {
            Board board = new Board();
            board.setNo(rs.getInt("no"));
            board.setTitle(rs.getString("title"));
            board.setWriter(rs.getString("writer"));
            board.setContent(rs.getString("content"));
            board.setRegDate(rs.getTimestamp("reg_date"));
            board.setReadCount(rs.getInt("read_count"));
            board.setPass(rs.getString("pass"));
            board.setFile1(rs.getString("file1"));

            boardList.add(board);

        } while(rs.next());
    }
} catch(Exception e) {
    e.printStackTrace();
} finally {
    try {

```

```

        if(rs != null) rs.close();
        if(pstmt != null) pstmt.close();
        if(conn != null) conn.close();
    } catch(SQLException se) {}
}
return boardList;

} // end searchList(String type, String keyword, int startRow, int endRow);

```

▶ 게시 글 검색 리스트 요청을 처리하는 JSP 페이지

게시 글 리스트 요청과 게시 글 검색 리스트 요청을 별개의 jsp 페이지에서 처리하면 더 효율적일 것 같지만 실제로 구현해 보면 그렇지 않기 때문에 boardList.jsp에서 게시 글 리스트와 게시 글 검색 리스트 요청 모두를 처리하도록 구현하였다.

아래에서 **빨간색 볼드체로 표현된** 부분이 이전 소스에서 추가되거나 수정된 내용이다.

- webapp/board/boardList.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="com.jspstudy.bbs.dao.*,com.jspstudy.bbs.vo.*" %>
<%@ page import="java.util.*" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<%!
    // 한 페이지에 보여 줄 게시 글의 수를 상수로 선언하고 있다.
    final int PAGE_SIZE = 5;

    /* 한 페이지에 보여 질 페이지 그룹의 수를 상수로 선언하고 있다.
    * [이전] 1 2 3 4 5 6 7 8 9 10 [다음]
    */
    final int PAGE_GROUP = 10;
%>
<%
    /* 검색 기능의 POST 방식 요청에 따른 문자 셋 처리
    * 게시 글 리스트에서 검색 버튼이 클릭되면 자바스크립트로 유효성 검사를 한 후
    * POST 방식으로 요청하게 되는데 검색어에 한글이 포함되어 있을 경우 적절한
    * 문자 셋 처리를 하지 않으면 한글이 깨지므로 DB에서 검색이 제대로 되지 못한다.
    */
    request.setCharacterEncoding("utf-8");

    // 요청 파라미터로 넘어 온 페이지 번호를 읽어온다.
    String pageNum = request.getParameter("pageNum");
    String type = request.getParameter("type");
    String keyword = request.getParameter("keyword");

```

```

/* pageNum이 null 이면 처음 게시 글 리스트를 요청하거나 게시 글쓰기에서
 * Redirect 되어 넘어온 요청으로 pageNum을 1페이지로 설정한다.
 * 즉 첫 번째 페이지에 해당하는 게시 글 리스트를 화면에 출력한다.
 */
if(pageNum == null) {
    pageNum = "1";
}

// 요청 파라미터의 pageNum을 int 형으로 변환하여 현재 페이지로 설정
int currentPage = Integer.parseInt(pageNum);

/* 요청한 페이지에 해당하는 게시 글의 첫 번째 행의 값을 계산한다.
 * 현재 페이지가 1일 경우 startRow는 1, 2페이지 일 경우 startRow는 6이 된다.
 *
 * 테이블에서 현재 페이지에 해당하는 게시 글을 검색할 때 ROWNUM을 사용했다.
 * ROWNUM은 쿼리의 결과로 검색되는 행들의 순서 값을 가진 의사컬럼으로
 * 1부터 시작한다. 예를 들어 3페이지에 해당하는 게시 글 리스트를 가져 온다면
 * 한 페이지에 보여줄 게시 글의 수가 5개로 지정되어 있으므로 startRow는 11이 된다.
 * 즉 아래의 공식에 의해 startRow(11) = 3 * 5 - (5 - 1);
 * 첫 번째 페이지 startRow = 1, 두 번째 페이지 startRow = 6이 된다.
 */
int startRow = currentPage * PAGE_SIZE - (PAGE_SIZE - 1);
//int startRow = (currentPage - 1) * PAGE_SIZE + 1;

int endRow = startRow + PAGE_SIZE - 1;
int listCount = 0;
ArrayList<Board> bList = null;

// BoardDao 객체 생성
BoardDao dao = new BoardDao();

/* 요청 파라미터에서 type이나 keyword가 비어 있으면 일반
 * 게시 글 리스트를 요청하는 것으로 간주하여 false 값을 갖게 한다.
 */
boolean searchOption = (type == null || type.equals(""))
    || keyword == null || keyword.equals("") ? false : true;

// 검색 요청이 아니면
if(! searchOption) {
    // 전체 게시 글 수를 구한다.
    listCount = dao.getBoardCount();
    bList = dao.boardList(startRow, endRow);
} else { // 검색 요청이면
    // 검색어에 해당하는 게시 글 수를 구한다.

```

```

listCount = dao.getBoardCount(type, keyword);
bList = dao.searchList(type, keyword, startRow, endRow);
}
System.out.println("listCount : " + listCount);

/* 페이지 그룹 이동 처리를 위해 게시판의 전체 페이지 수를 계산한다.
 * [이전] 11 12 13... 또는 ... 8 9 10 [다음] 처리
 * 전체 페이지 = 전체 게시 글 수 / 한 페이지에 표시할 게시 글 수가 되는데
 * 이 계산식에서 나머지가 존재하면 전체 페이지 수는 전체 페이지 + 1이 된다.
 */
int pageCount = listCount / PAGE_SIZE
    + (listCount % PAGE_SIZE == 0 ? 0 : 1);

/* 페이지 그룹 처리를 위해 페이지 그룹별 시작 페이지와 마지막 페이지를 계산하여
 * 페이지 그룹의 시작 페이지 : 1, 11, 21, 31...
 * 첫 번째 페이지 그룹에서 페이지 리스트는 1 ~ 10 되므로 currentPage가
 * 1 ~ 10 사이에 있으면 startPage는 1이 되고 11 ~ 20 사이는 11이 된다.
 *
 * 정수형 연산의 특징을 이용해 startPage를 아래와 같이 구할 수 있다.
 * 아래 계산식으로 계산된 결과를 보면 현재 그룹의 마지막 페이지일 경우
 * startPage가 다음 그룹의 시작 페이지가 나오게 되므로 삼항 연산자를
 * 사용해 현재 페이지가 속한 그룹의 startPage가 되도록 조정 하였다.
 */
int startPage = (currentPage / PAGE_GROUP) * PAGE_GROUP + 1
    - (currentPage % PAGE_GROUP == 0 ? PAGE_GROUP : 0);

// 현재 페이지 그룹의 마지막 페이지 : 10, 20, 30...
int endPage = startPage + PAGE_GROUP - 1;

/* 위의 식에서 endPage를 구하게 되면 endPage는 항상 PAGE_GROUP의
 * 크기만큼 증가(10, 20, 30 ...) 되므로 맨 마지막 페이지 그룹의 endPage가
 * 정확하지 못할 경우가 발생하게 된다. 다시 말해 전체 페이지가 53페이지 라고
 * 가정하면 위의 식에서 계산된 endPage는 60 페이지가 되지만 실제로
 * 60페이지는 존재하지 않는 페이지이므로 문제가 발생하게 된다.
 * 그래서 맨 마지막 페이지에 대한 보정이 필요하여 아래와 같이 endPage와
 * pageCount를 비교하여 현재 페이지 그룹에서 endPage가 pageCount 보다
 * 크다면 pageCount를 endPage로 지정 하였다. 즉 현재 페이지 그룹이
 * 마지막 페이지 그룹이면 endPage는 전체 페이지 수가 되도록 지정한 것이다.
 */
if(endPage > pageCount) {
    endPage = pageCount;
}
%>

<!-- 아래에서 EL로 접근하기 위해서 pageContext 영역의 속성으로 저장 -->

```

```

<c:set var="bList" value="<%= bList %> " />
<c:set var="currentPage" value="<%= currentPage %> " />
<c:set var="pageGroup" value="<%= PAGE_GROUP %> " />
<c:set var="pageCount" value="<%= pageCount %> " />
<c:set var="startPage" value="<%= startPage %> " />
<c:set var="endPage" value="<%= endPage %> " />
<c:set var="searchOption" value="<%= searchOption %> " />
<!DOCTYPE html>
<html lang="ko">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>게시 글 리스트</title>
    <link href="../bootstrap/bootstrap.min.css" rel="stylesheet" >
    <script src="../js/jquery-3.3.1.min.js"></script>
    <script src="../js/formcheck.js"></script>
  </head>
  <body>
    <div class="container">
      <%@ include file="../pages/header.jsp" %>
      <!-- content -->
      <div class="row my-5" id="global-content">
        <div class="col">
          <div class="row text-center">
            <div class="col">
              <h2 class="fs-3 fw-bold">게시 글 리스트</h2>
            </div>
          </div>
          <div class="row">
            <div class="col-auto">
              <select name="type" class="form-select">
                <option value="title">제목</option>
                <option value="writer">작성자</option>
                <option value="content">내용</option>
              </select>
            </div>
            <div class="col-4">
              <input type="text" name="keyword" class="form-control"/>
            </div>
            <div class="col-auto">
              <input type="submit" value="검색" class="btn btn-primary"/>
            </div>
          </div>
        </div>
      </div>
    </body>
  </html>

```



```

<!-- 검색 요청일 경우 아래를 화면에 표시 -->
<c:if test="${ searchOption }">
    <div class="row my-3">
        <div class="col text-center">
            "${ keyword }" 검색 결과
        </div>
    </div>

    <%--
        검색 요청일 경우 일반 게시 글 리스트로 이동할 수 있도록 링크를 설정했다.
    --%>

    <div class="row my-3">
        <div class="col-6">
            <a href="boardList.jsp" class="btn btn-outline-success">리스트</a>
        </div>
        <div class="col-6 text-end">
            <a href="writeForm.jsp" class="btn btn-outline-success">글쓰기</a>
        </div>
    </div>
</c:if>

<!-- 검색 요청이 아닐 경우 아래를 화면에 표시 -->
<c:if test="${ not searchOption }">
    <div class="row my-3">
        <div class="col text-end">
            <a href="writeForm.jsp" class="btn btn-outline-success">글쓰기</a>
        </div>
    </div>
</c:if>

<div class="row my-3">
    <div class="col">
        <table class="table table-hover">
            <thead>
                <tr class="table-dark">
                    <th>NO</th>
                    <th>제목</th>
                    <th>작성자</th>
                    <th>작성일</th>
                    <th>조회수</th>
                </tr>
            </thead>
            <tbody class="text-secondary">
                <%--
                    검색 요청 이면서 검색된 리스트가 존재할 경우

```

게시 글 상세보기로 링크를 적용할 때 type과 keyword
파라미터를 적용해 링크를 설정한다.

--%>

<c:if test="\${ searchOption and not empty bList }">

<c:forEach var="b" items="\${bList}" varStatus="status">

<tr>

<td>\${ b.no }</td>

<td>

<a href="boardDetail.jsp?no=\${b.no}&pageNum=\${currentPage}&type=\${
type }&keyword=\${ keyword }" class="text-decoration-none link-secondary">\${ b.title }

</td>

<td>\${ b.writer }</td>

<td>\${ b.regDate }</td>

<td>\${ b.readCount }</td>

</tr>

</c:forEach>

</c:if>

<%--

일반 게시 글 리스트 요청 이면서 게시 글 리스트가 존재할 경우
게시 글 상세보기로 링크를 적용할 때 type과 keyword
파라미터는 필요 없다.

--%>

<c:if test="\${ not searchOption and not empty bList }">

<c:forEach var="b" items="\${bList}" varStatus="status">

<tr>

<td>\${ b.no }</td>

<td><a href="boardDetail.jsp?no=\${b.no}&pageNum=\${currentPage}"
class="text-decoration-none link-secondary">\${ b.title }</td>

<td>\${ b.writer }</td>

<td>\${ b.regDate }</td>

<td>\${ b.readCount }</td>

</tr>

</c:forEach>

</c:if>

<%-- 검색 요청이면서 검색된 리스트가 존재하지 않을 경우 --%>

<c:if test="\${ searchOption and empty bList }">

<tr>

<td colspan="5" class="text-center">

"\${ keyword }"가 포함된 게시 글이 존재하지 않습니다.

</td>

</tr>

</c:if>

<%-- 일반 게시 글 리스트 요청이면서 게시 글 리스트가 존재하지 않을 경우 --%>

```

<c:if test="${ not searchOption and empty bList }">
    <tr>
        <td colspan="5" class="text-center">게시 글이 존재하지 않습니다.</td>
    </tr>
</c:if>
</tbody>
</table>
</div>
</div>

```

<!-- 검색 요청이면서 검색된 리스트가 존재할 경우 페이지네이션 -->

```

<c:if test="${ searchOption and not empty bList }">
    <div class="row">
        <div class="col">
            <nav aria-label="Page navigation">
                <ul class="pagination justify-content-center">
                    <%--
/* 현재 페이지 그룹의 시작 페이지가 pageGroup보다 크다는 것은
* 이전 페이지 그룹이 존재한다는 것으로 현재 페이지 그룹의 시작 페이지에
* pageGroup을 마이너스 하여 링크를 설정하면 이전 페이지 그룹의
* startPage로 이동할 수 있다.
**/
--%>
                    <c:if test="${ startPage > pageGroup }">
                        <li class="page-item">
                            <a class="page-link" href="boardList.jsp?pageNum=${ startPage -
pageGroup }&type=${ type }&keyword=${ keyword }">Pre</a>
                        </li>
                    </c:if>

                    <%--
/* 현재 페이지 그룹의 startPage 부터 endPage 만큼 반복하면서
* 현재 페이지와 같은 그룹에 속한 페이지를 출력하고 링크를 설정한다.
* 현재 페이지는 링크를 설정하지 않는다.
**/
--%>
                    <c:forEach var="i" begin="${startPage}" end="${endPage}">
                        <c:if test="${i == currentPage }">
                            <li class="page-item active" aria-current="page">
                                <span class="page-link">${i}</span>
                            </li>
                        </c:if>
                        <c:if test="${i != currentPage }">
                            <li class="page-item">
                                <a class="page-link" href="boardList.jsp?pageNum=${ i }&type=${

```

```

type }&keyword=${ keyword }">${i}</a>
    </li>
</c:if>
</c:forEach>

<%--
/* 현재 페이지 그룹의 마지막 페이지가 전체 페이지 보다 작다는 것은
* 다음 페이지 그룹이 존재한다는 것으로 현재 페이지 그룹의 시작 페이지에
* pageGroup을 플러스 하여 링크를 설정하면 다음 페이지 그룹의
* startPage로 이동할 수 있다.
**/
--%>
<c:if test="${ endDate < pageCount }">
    <li class="page-item">
        <a class="page-link" href="boardList.jsp?pageNum=${ startPage +
pageGroup }&type=${ type }&keyword=${ keyword }">Next</a>
    </li>
</c:if>
</ul>
</nav>
</div>
</div>
</c:if>

<!-- 일반 게시글 요청이면서 검색된 리스트가 존재할 경우 페이지네이션 -->
<c:if test="${ not searchOption and not empty bList }">
    <div class="row">
        <div class="col">
            <nav aria-label="Page navigation">
                <ul class="pagination justify-content-center">
                    <%--
/* 현재 페이지 그룹의 시작 페이지가 pageGroup보다 크다는 것은
* 이전 페이지 그룹이 존재한다는 것으로 현재 페이지 그룹의 시작 페이지에
* pageGroup을 마이너스 하여 링크를 설정하면 이전 페이지 그룹의
* startPage로 이동할 수 있다.
**/
--%>
                    <c:if test="${ startPage > pageGroup }">
                        <li class="page-item">
                            <a class="page-link" href="boardList.jsp?pageNum=${ startPage -
pageGroup }">Pre</a>
                        </li>
                    </c:if>
                    <%--
/* 현재 페이지 그룹의 startPage 부터 endDate 만큼 반복하면서

```

```

* 현재 페이지와 같은 그룹에 속한 페이지를 출력하고 링크를 설정한다.
* 현재 페이지는 링크를 설정하지 않는다.
**/
--%>
<c:forEach var="i" begin="${startPage}" end="${endPage}">
    <c:if test="${i == currentPage }">
        <li class="page-item active" aria-current="page">
            <span class="page-link">${i}</span>
        </li>
    </c:if>
    <c:if test="${i != currentPage }">
        <li class="page-item">
            <a class="page-link" href="boardList.jsp?pageNum=${ i }">${i}</a>
        </li>
    </c:if>
</c:forEach>

<%--
/* 현재 페이지 그룹의 마지막 페이지가 전체 페이지 보다 작다는 것은
* 다음 페이지 그룹이 존재한다는 것으로 현재 페이지 그룹의 시작 페이지에
* pageGroup을 플러스 하여 링크를 설정하면 다음 페이지 그룹의
* startPage로 이동할 수 있다.
**/
--%>
<c:if test="${ endPage < pageCount }">
    <li class="page-item">
        <a class="page-link" href="boardList.jsp?pageNum=${ startPage +
pageGroup }">Next</a>
    </li>
</c:if>
</ul>
</nav>
</div>
</div>
</div>
</c:if>
</div>
</div>
<%@ include file="../pages/footer.jsp" %>
</div>
<script src="../bootstrap/bootstrap.bundle.min.js"></script>
</body>
</html>

```

- webapp/js/formcheck.js에 추가

```
// 게시 글 리스트, 검색 결과 페이지에서 검색 요청 처리
$("#searchForm").on("submit", function() {
    var keyword = $("#keyword").val();
    if(keyword.length <= 0) {
        alert("검색어가 입력되지 않았습니다.\n검색어를 입력해주세요");
        return false;
    }
    $(this).attr("method", "post");
    $(this).attr("action", "boardList.jsp");
});
```

▶ 게시 글 상세보기 요청을 처리하는 JSP 페이지

- webapp/board/boardDetail.jsp

아래에서 **빨간색 볼드체**로 표현된 부분이 이전 소스에서 추가되거나 수정된 내용이다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="com.jspstudy.bbs.vo.*, com.jspstudy.bbs.dao.*" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%
    //요청 파라미터로 넘어 온 게시 글 번호와 페이지 번호를 읽어온다.
    String no = request.getParameter("no");
    String pageNum = request.getParameter("pageNum");
    String type = request.getParameter("type");
    String keyword = request.getParameter("keyword");

    // no와 pageNum이 비어 있으면 비정상 요청임
    if(no == null || no.equals("") || pageNum == null || pageNum.equals("")) {
        out.println("<script>");
        out.println(" alert('잘못된 접근입니다.'):");
        out.println(" history.back();");
        out.println("</script>");
        return;
    }

    /* 요청 파라미터에서 type이나 keyword가 비어 있으면 일반
     * 게시 글 리스트에서 넘어온 요청으로 간주하여 false 값을 갖게 한다.
     * 이 정보는 게시 글 리스트와 검색 리스트로 구분해 돌려보내기 위해 필요하다.
     */
    boolean searchOption = (type == null || type.equals(""))
        || keyword == null || keyword.equals("")) ? false : true;

    // BoardDao 객체를 생성하고 no에 해당하는 게시 글 하나를 가져온다.
```

```

BoardDao dao = new BoardDao();
Board board = dao.getBoard(Integer.valueOf(no));
%>
<!DOCTYPE html>
<html lang="ko">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>게시 글 상세보기</title>
    <link href="../bootstrap/bootstrap.min.css" rel="stylesheet" >
    <script src="../js/jquery-3.3.1.min.js"></script>
    <script src="../js/formcheck.js"></script>
  </head>
  <body>
    <div class="container">
      <%@ include file="../pages/header.jsp" %>
      <!-- content -->
      <div class="row my-5" id="global-content">
        <div class="col">
          <form name="checkForm" id="checkForm">
            <input type="hidden" name="no" id="no" value="<%= board.getNo() %>" />
            <input type="hidden" name="pass" id="rPass" />
            <input type="hidden" name="pageNum" value="<%= pageNum %>" />

            <%--
              검색 리스트에서 들어온 요청일 경우 다시 keyword에 해당하는
              검색 리스트로 돌려보내기 위해서 아래의 파라미터가 필요하다.
            --%>

            <c:if test="<%= searchOption %>">
              <input type="hidden" name="type" value="<%= type %>" />
              <input type="hidden" name="keyword" value="<%= keyword %>" />
            </c:if>
          </form>
          <div class="row text-center">
            <div class="col">
              <h2 class="fs-3 fw-bold">게시 글 상세보기</h2>
            </div>
          </div>
          <div class="row my-3">
            <div class="col">
              <table class="table table-bordered" >
                <tbody>
                  <tr>
                    <th class="table-secondary">제 목</th>
                    <td colspan="3"><%= board.getTitle() %></td>

```



```

        일반 게시 글 리스트에서 온 요청이면 일반 게시 글 리스트로 돌려보낸다.
--%>
<c:if test="<%= ! searchOption %>">
    &nbsp;&nbsp;&nbsp;<input class="btn btn-primary" type="button" value="목록보기"
        onclick="location.href='boardList.jsp?pageNum=<%= pageNum %>'" />
</c:if>
<%--
        검색 리스트에서 온 요청이면 검색 리스트의 동일한 페이지로 돌려보낸다.
--%>
<c:if test="<%= searchOption %>">
    &nbsp;&nbsp;&nbsp;<input class="btn btn-primary" type="button" value="목록보기"
        onclick="location.href='boardList.jsp?pageNum=<%= pageNum
%>&type=<%= type %>&keyword=<%= keyword %>'" />
<%--
        위의 쿼리 스트링을 작성할 때 같은 줄에서 띄어쓰기 하는 것은 문제되지
        않지만 줄 바뀌서 작성하게 되면 스크립트 에러가 발생한다.
--%>
</c:if>
</div>
</div>
</div>
</div>
<%@ include file="../pages/footer.jsp" %>
</div>
<script src="../bootstrap/bootstrap.bundle.min.js"></script>
</body>
</html>

```

▶ 게시 글 수정 폼 요청을 처리하는 JSP 페이지

- webapp/board/updateForm.jsp

아래에서 빨간색 볼드체로 표현된 부분이 이전 소스에서 추가되거나 수정된 내용이다.

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="com.jspstudy.bbs.dao.*,com.jspstudy.bbs.vo.*" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
    /* boardDetail.jsp에서 요청시 post로 넘어오기 때문에
    * request 영역의 문자 셋 처리가 필요하다.
    */
    request.setCharacterEncoding("utf-8");

    String sNo = request.getParameter("no");
    String pass = request.getParameter("pass");

```

```

String pageNum = request.getParameter("pageNum");
String type = request.getParameter("type");
String keyword = request.getParameter("keyword");

// no와 pass 그리고 pageNum이 비어 있으면 비정상 요청임
if(sNo == null || sNo.equals("") || pass == null || pass.equals("")
    || pageNum == null || pageNum.equals("")) {
    out.println("<script>");
    out.println(" alert('잘못된 접근입니다.');

```

```

<div class="row text-center">
  <div class="col">
    <h2 class="fs-3 fw-bold">게시 글 수정하기</h2>
  </div>
</div>
<form name="updateForm" id="updateForm" action="updateProcess.jsp"
  class="row g-3 border-primary" method="post"
  <%= board.getFile1() != null ? "" : "enctype='multipart/form-data'" %>>
  <!--
    no는 DB에서 게시 글을 수정하기 위해 필요하고 pageNum은 게시 글이
    수정된 후에 이전에 사용자가 머물렀던 게시 글 리스트의 동일한 페이지로
    보내기 위해 필요한 정보이다.
  -->
  <input type="hidden" name="no" value="<%= board.getNo() %>" />
  <input type="hidden" name="pageNum" value="<%= pageNum %>" />
  <!--
    검색 요청일 경우 다시 keyword에 해당하는 검색 리스트로
    돌려보내기 위해서 아래의 파라미터가 필요하다.
  -->
  <c:if test="<%= searchOption %>">
    <input type="hidden" name="type" value="<%= type %>" />
    <input type="hidden" name="keyword" value="<%= keyword %>" />
  </c:if>

  <div class="col-4 offset-md-2">
    <label for="writer" class="form-label">글쓴이</label>
    <input type="text" class="form-control" name="writer" id="writer"
      placeholder="작성자를 입력해 주세요" value="<%= board.getWriter() %>" />
  </div>
  <div class="col-4 ">
    <label for="pass" class="form-label">비밀번호</label>
    <input type="password" class="form-control" name="pass" id="pass">
  </div>
  <div class="col-8 offset-md-2">
    <label for="title" class="form-label">제 목</label>
    <input type="text" class="form-control" name="title" id="title"
      value="<%= board.getTitle() %>" />
  </div>
  <div class="col-8 offset-md-2">
    <label for="content" class="form-label">내 용</label>
    <textarea class="form-control" name="content" id="content"
      rows="10"><%= board.getContent() %></textarea>
  </div>
  <div class="col-8 offset-md-2">
    <label for="file1" class="form-label">파 일</label>

```

```

        <input type="file" class="form-control" name="file1" id="file1"
            <%= board.getFile1() == null ? "" : "disabled" %>>
    </div>
    <div class="col-8 offset-md-2 text-center mt-5">
        <input type="submit" value="수정하기" class="btn btn-primary"/>
        <!--
        일반 게시 글 리스트에서 온 요청이면 일반 게시 글 리스트로 돌려보낸다.
        --%>
        <c:if test="<%= ! searchOption %">
            &nbsp;&nbsp;&nbsp;<input class="btn btn-primary" type="button" value="목록보기"
                onclick="location.href='boardList.jsp?pageNum=<%= pageNum %>'" />
        </c:if>
        <!--
        검색 리스트에서 온 요청이면 검색 리스트의 동일한 페이지로 돌려보낸다.
        --%>
        <c:if test="${ searchOption }">
            &nbsp;&nbsp;&nbsp;<input class="btn btn-primary" type="button" value="목록보기"
                onclick="location.href='boardList.jsp?pageNum=<%=
                                                                    pageNum
%>&type=<%= type %>&keyword=<%= keyword %>'" />
        </c:if>
    </div>
</form>
</div>
</div>
    <%@ include file="../pages/footer.jsp" %>
</div>
    <script src="../bootstrap/bootstrap.bundle.min.js"></script>
</body>
<%
/* 게시 글 번호에 해당하는 비밀번호가 틀리면 비밀번호가 틀리다. 라고 알려주고
 * 브라우저의 history 객체를 이용해 바로 이전에 있었던 주소로 돌려보낸다.
 */
} else {
%>
    <script>
        alert("비밀번호가 다릅니다.");
        history.back();
    </script>
    <%}%>
</html>

```

▶ 게시 글 수정 요청을 처리하는 JSP 페이지

- webapp/board/updateProcess.jsp

아래에서 **빨간색 볼드체로 표현된** 부분이 이전 소스에서 추가되거나 수정된 내용이다.

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="com.oreilly.servlet.*, com.oreilly.servlet.multipart.*" %>
<%@ page import="java.sql.*, java.io.*, java.net.*" %>
<%@ page import="com.jspstudy.bbs.vo.*, com.jspstudy.bbs.dao.*" %>
<%!

    static String uploadDir;
    static File parentFile;

    public void jspInit() {
        // web.xml에 지정한 웹 어플리케이션 초기화 파라미터를 읽는다.
        uploadDir = getServletContext().getInitParameter("uploadDir");

        /* 웹 어플리케이션 초기화 파라미터로 읽어온 이미지가 저장될 폴더의
         * 로컬 경로를 구하여 그 경로와 파일명으로 File 객체를 생성한다.
         */
        String realPath = getServletContext().getRealPath(uploadDir);
        parentFile = new File(realPath);

        /* 파일 객체에 지정한 위치에 디렉토리가 존재하지 않거나
         * 파일 객체가 디렉토리가 아니라면 디렉토리를 생성한다.
         */
        if(! (parentFile.exists() && parentFile.isDirectory())) {
            parentFile.mkdir();
        }
        System.out.println("init - " + parentFile);
    }
%>
<%

String contentType = request.getHeader("Content-Type");
System.out.println("contentType : " + contentType);

// BoardDao 객체 생성
BoardDao dao = new BoardDao();
Board board = null;
String pass= null, title = null, writer = null, content = null,
    sNo = null, pageNum = null, type=null, keyword=null;
int no = 0;

/* 비밀번호가 맞지 않을 경우 먼저 처리해야 하나 Content-Type에
 * 따라서 파라미터를 읽을 수 있는 객체가 다르기 때문에 각각 처리해야 한다.
 * Content-Type이 application/x-www-form-urlencoded 일 경우
 * request.getParameter() 메서드로 파라미터를 읽을 수 있지만
 * multipart/form-data 일 경우에는 request 객체를 이용해

```

* 파라미터를 읽을 수 없기 때문에 각각 처리하는 코드를 기술해야 한다.

*/

// 요청이 multipart/form-data 일 경우

if(contentType.contains("multipart/form-data")) {

/* cos 라이브러리를 이용한 파일 업로드 구현하기

*

* 1. MultipartRequest의 생성자 매개변수에 지정할 데이터를 설정

*

* 파일이 저장될 폴더의 로컬 경로를 구한다.

*/

String realPath = application.getRealPath(uploadDir);

// 업로드 파일의 최대 크기를 100MB로 지정

int maxFileSize = 100 * 1024 * 1024;

// 파일의 인코딩 타입을 UTF-8로 지정

String encoding = "UTF-8";

/* 2. 파일 업로드를 처리할 MultipartRequest 객체 생성

*

* WEB-INF/lib/cos.jar 파일을 살펴보면 MultipartRequest 클래스는

* com.oreilly.servlet 패키지에 위치하며 파일 업로드를 직접적으로 처리하는

* 역할을 담당하는 클래스로 파일 업로드와 관련된 다양한 메소드를 정의하고 있다.

* 생성자는 5개로 오버로딩 되어 있고 아래 생성자가 호출되도록 정의되어 있다.

*

* public MultipartRequest(HttpServletRequest request,

* String saveDirectory,

* int maxPostSize,

* String encoding,

* FileRenamePolicy policy) throws IOException {...}

*

* 이 생성자를 살펴보면 request, saveDirectory, maxPostSize는 필수사항으로

* 이 매개변수가 null이거나 0보다 작다면 생성자 안에서 예외를 발생시킨다.

*

* request : MultipartRequest에 연결할 사용자의 요청 정보가 담긴 객체

* saveDirectory : 업로드 된 파일을 저장할 서버의 디렉토리 지정

* maxPostSize : 업로드 파일의 최대 크기 지정

* encoding : 파일의 인코딩 방식 지정, 파일 이름이 한글일 경우 필히 utf-8 지정

* policy : 사용자가 업로드 한 파일을 저장할 서버의 디렉토리에 현재 업로드 되는

* 파일과 이름이 중복된 파일이 존재할 경우 현재 업로드 되는 파일의

* 이름을 어떻게 변경할지에 대한 정책을 지정하는 매개변수 이다.

* 일반적으로 new DefaultFileRenamePolicy()를 사용하며

* 이 클래스는 abc.jpg 파일을 업로드 할때 이미 같은 이름의 파일이

```

*           존재하면 자동으로 abc1.jpg와 같이 파일을 변경해 준다.
*
* 아래와 같이 MultipartRequest 객체를 생성하면 saveDirectory에 지정한
* 서버의 디렉토리로 파일이 바로 업로드 된다.
**/
MultipartRequest multi = new MultipartRequest(request, realPath,
        maxFileSize, encoding, new DefaultFileRenamePolicy());

/* 3. MultipartRequest 객체를 이용해 클라이언트로부터 요청된 데이터를 처리
*
* 파일 업로드 처리를 위해서는 모든 요청에 대한 처리를 MultipartRequest 객체를
* 이용해 접근해야 한다. 위에서 MultipartRequest 객체를 생성할 때 요청에 대한
* 정보를 담고 있는 request를 생성자의 매개변수로 지정해 MultipartRequest를
* 통해 사용자의 요청 정보에 접근할 수 있다.
*
* MultipartRequest 클래스에 정의된 주요 메소드는 아래와 같다.
* getParameter(name) : name에 지정한 파라미터 값을 반환
* getParameterNames() : 폼에서 전송된 모든 파라미터 이름을
*                       Enumeration 타입으로 반환
* getParameterValues(name) : name에 지정한 파라미터 값을 String 배열로 반환
* getFile(fileName) : 업로드 된 파일 중에서 fileName에 지정한 파라미터
*                    이름을 가진 파일의 정보를 File 객체로 반환
* getFilesNames() : 폼에서 전송된 모든 파일의 이름을 Enumeration 타입으로 반환
* getFileSystemName(name) : name에 지정한 파라미터 이름을 가진
*                          파일의 이름을 반환
* getOriginalFileName() : 사용자가 업로드 한 파일의 원본 이름을 반환
* getContentType() : 사용자가 업로드 한 파일의 콘텐츠 타입을 반환
**/

/* 사용자가 폼에 입력한 데이터 처리
* MultipartRequest 객체를 통해 파라미터를 읽어 변수에 저장한다.
**/
sNo = multi.getParameter("no");
pass = multi.getParameter("pass");
pageNum = multi.getParameter("pageNum");
type = multi.getParameter("type");
keyword = multi.getParameter("keyword");

// no와 pass 그리고 pageNum이 비어 있으면 비정상 요청임
if(sNo == null || sNo.equals("") || pass == null || pass.equals("")
    || pageNum == null || pageNum.equals("")) {
    out.println("<script>");
    out.println(" alert('잘못된 접근입니다.');

```

```

        return;
    }

    no = Integer.parseInt(sNo);

    // 게시 글의 비밀번호를 체크해 맞지 않으면 이전으로 돌려보낸다.
    boolean isPassCheck = dao.isPassCheck(no, pass);
    if(! isPassCheck) {
        System.out.println("비밀번호 맞지 않음");
        StringBuilder sb = new StringBuilder();
        sb.append("<script>");
        sb.append(" alert('비밀번호가 맞지 않습니다.');");
        sb.append(" history.back();");
        sb.append("</script>");
        out.println(sb.toString());
        return;
    }

    title = multi.getParameter("title");
    writer = multi.getParameter("writer");
    content = multi.getParameter("content");

    /* 하나의 게시 글 정보를 저장하는 자바빈 객체를 생성하고 파라미터로
     * 넘겨받은 요청 데이터를 Board 객체에 저장한다.
     */
    board = new Board();
    board.setNo(no);
    board.setTitle(title);
    board.setWriter(writer);
    board.setPass(pass);
    board.setContent(content);

    /* 사용자가 업로드한 파일 데이터 처리
     * MultipartRequest 객체를 통해 파일 이름을 구하여 변수에 저장한다.
     */
    String fileName = multi.getFilesystemName("file1");
    System.out.println("업로드 된 파일명 : " + fileName);
    System.out.println("원본 파일명 : " + multi.getOriginalFileName("file1"));

    board.setFile1(fileName);
    if(board.getFile1() == null) {
        System.out.println("파일이 업로드 되지 않았음");
    }

    // 요청이 multipart/form-data 아닌 경우

```



```

} else {
    /* updateForm.jsp에서 요청시 post로 넘어오기 때문에
     * request 영역의 문자 셋 처리가 필요하다.
     */
    request.setCharacterEncoding("utf-8");

    sNo = request.getParameter("no");
    pass = request.getParameter("pass");
    pageNum = request.getParameter("pageNum");
    type = request.getParameter("type");
    keyword = request.getParameter("keyword");

    // no와 pass 그리고 pageNum이 비어 있으면 비정상 요청임
    if(sNo == null || sNo.equals("") || pass == null || pass.equals("")
        || pageNum == null || pageNum.equals("")) {
        out.println("<script>");
        out.println(" alert('잘못된 접근입니다.');

```

```

board.setNo(no);
board.setTitle(title);
board.setWriter(writer);
board.setPass(pass);
board.setContent(content);
}

// BoardDao 객체를 이용해 게시 글을 수정한다.
dao.updateBoard(board);

/* 요청 파라미터에서 type이나 keyword가 비어 있으면 일반
 * 게시 글 리스트에서 넘어온 요청으로 간주하여 false 값을 갖게 한다.
 * 이 정보는 게시 글 리스트와 검색 리스트로 구분해 돌려보내기 위해 필요하다.
 */
boolean searchOption = (type == null || type.equals(""))
    || keyword == null || keyword.equals("") ? false : true;

/* 리다이렉트 할 때 게시 글 리스트의 페이지 번호를 파라미터로 넘겨 사용자가
 * 게시 글 수정을 요청한 페이지와 동일한 페이지로 리다이렉트 시킨다.
 */
String url = "boardList.jsp?pageNum=" + pageNum;

/* 검색 리스트 상태에서 게시 글 상세보기로 들어와 게시 글을 수정하는 것이라면
 * 검색 옵션에 해당하는 검색한 결과에 대한 게시 글 리스트 페이지로 Redirect
 * 시켜야 하므로 type과 keyword를 Redirect 주소에 추가한다.
 * Redirect 기법은 요청한 결과가 이동했다고 브라우저에게 이동할 주소를 응답하는
 * 것으로 브라우저는 주소 표시줄에 주소를 입력해 요청하게 되므로 GET 방식 요청이다.
 */
if(searchOption) {

    /* 리다이렉트 할 때 파라미터에 한글이 포함되어 있으면 한글로 된 파라미터 값은
     * 공백문자로 변경되어 리다이렉트 되기 때문에 한글 데이터는 깨지게 된다.
     * 이런 경우에는 java.net 패키지의 URLEncoder 클래스를 이용해 아래와
     * 같이 수동으로 URL 인코딩을 하면 이 문제를 해결할 수 있다.
     */
    keyword = URLEncoder.encode(keyword, "utf-8");
    url += "&type=" + type + "&keyword=" + keyword;
}

System.out.println("keyword : " + keyword);
System.out.println("url : " + url);

/* DB에 게시 글을 수정하고 브라우저에게 게시 글 리스트를 요청하라고 응답
 * 게시 글 수정이 완료된 후 Redirect 시키지 않으면 이 페이지를 새로 고침 하여
 * 재요청 할 때 마다 이미 수정된 게시 글을 계속해서 수정하려고 하는 문제가 발생한다.
 * 이런 경우에는 Redirect 기법을 이용해 DB에 추가, 수정, 삭제하는 동작이 아닌

```

```

* 조회하는 곳으로 이동하도록 하면 문제를 해결 할 수 있다.
**/
response.sendRedirect(url);
%>

```

▶ 게시 글 삭제 요청을 처리하는 JSP 페이지

- webapp/board/deleteProcess.jsp

아래에서 **빨간색 볼드체**로 표현된 부분이 이전 소스에서 추가되거나 수정된 내용이다.

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="java.sql.*, java.io.*, java.net.*" %>
<%@ page import="com.jspstudy.bbs.vo.*, com.jspstudy.bbs.dao.*" %>
<%
    /* boardDetail.jsp에서 요청시 post로 넘어오기 때문에
    * request 영역의 문자 셋 처리가 필요하다.
    **/
    request.setCharacterEncoding("utf-8");

    String sNo = request.getParameter("no");
    String pass = request.getParameter("pass");
    String pageNum = request.getParameter("pageNum");
    String type = request.getParameter("type");
    String keyword = request.getParameter("keyword");

    // no와 pass 그리고 pageNum이 비어 있으면 비정상 요청임
    if(sNo == null || sNo.equals("") || pass == null || pass.equals("")
        || pageNum == null || pageNum.equals("")) {
        out.println("<script>");
        out.println(" alert('잘못된 접근입니다.');");
        out.println(" history.back();");
        out.println("</script>");
        return;
    }

    // BoardDao 객체 생성
    BoardDao dao = new BoardDao();
    int no = Integer.parseInt(sNo);

    // 게시 글의 비밀번호를 체크해 맞지 않으면 이전으로 돌려보낸다.
    boolean isPassCheck = dao.isPassCheck(no, pass);
    if(! isPassCheck) {
        System.out.println("비밀번호 맞지 않음");
        StringBuilder sb = new StringBuilder();

```

```

sb.append("<script>");
sb.append(" alert('비밀번호가 맞지 않습니다.');
```

```

sb.append(" history.back();");
sb.append("</script>");
out.println(sb.toString());
return;
}

// BoardDao 객체를 이용해 게시 글을 삭제한다.
dao.deleteBoard(no);

/* 요청 파라미터에서 type이나 keyword가 비어 있으면 일반
 * 게시 글 리스트에서 넘어온 요청으로 간주하여 false 값을 갖게 한다.
 * 이 정보는 게시 글 리스트와 검색 리스트로 구분해 돌려보내기 위해 필요하다.
 */
boolean searchOption = (type == null || type.equals(""))
    || keyword == null || keyword.equals("")) ? false : true;

/* 리다이렉트 할 때 게시 글 리스트의 페이지 번호를 파라미터로 넘겨 사용자가
 * 게시 글 수정을 요청한 페이지와 동일한 페이지로 리다이렉트 시킨다.
 */
String url = "boardList.jsp?pageNum=" + pageNum;

/* 검색 리스트 상태에서 게시 글 상세보기로 들어와 게시 글을 삭제하는 것이라면
 * 검색 옵션에 해당하는 검색한 결과에 대한 게시 글 리스트 페이지로 Redirect
 * 시켜야 하므로 type과 keyword를 Redirect 주소에 추가한다.
 * Redirect 기법은 요청한 결과가 이동했다고 브라우저에게 이동할 주소를 응답하는
 * 것으로 브라우저는 주소 표시줄에 주소를 입력해 요청하게 되므로 GET 방식 요청이다.
 */
if(searchOption) {

    /* 리다이렉트 할 때 파라미터에 한글이 포함되어 있으면 한글로 된 파라미터 값은
     * 공백문자로 변경되어 리다이렉트 되기 때문에 한글 데이터는 깨지게 된다.
     * 이런 경우에는 java.net 패키지의 URLEncoder 클래스를 이용해 아래와
     * 같이 수동으로 URL 인코딩을 하면 이 문제를 해결할 수 있다.
     */
    keyword = URLEncoder.encode(keyword, "utf-8");
    url += "&type=" + type + "&keyword=" + keyword;
}

System.out.println("keyword : " + keyword);
System.out.println("url : " + url);

/* DB에 게시 글을 삭제하고 브라우저에게 게시 글 리스트를 요청하라고 응답
 * 게시 글 삭제가 완료된 후 Redirect 시키지 않으면 이 페이지를 새로 고침 하여
 * 재요청 할 때 마다 이미 삭제된 게시 글을 계속해서 수정하려고 하는 문제가 발생한다.

```

- * 이런 경우에는 Redirect 기법을 이용해 DB에 추가, 수정, 삭제하는 동작이 아닌
- * 조회하는 곳으로 이동하도록 하면 문제를 해결 할 수 있다.

**/

response.sendRedirect(url);

%>

2. 모델2 기반의 MVC패턴 게시판 구현

앞에서 JSP 페이지를 사용해 웹 애플리케이션을 구현했던 방식은 클라이언트의 요청을 JSP 페이지에서 받아서 처리하고 화면에 출력되는 부분도 모두 JSP에서 처리하는 방식으로 자바의 애플리케이션 설계 모델로 구분하자면 모델1 방식이라고 한다. 자바 프로그램으로 애플리케이션을 구현하는 방식에는 JSP 규격서에서 제안하고 있는 설계모델(Design Model)에 따라 모델1 방식과 모델2 방식으로 나눈다. 이 설계모델은 애플리케이션의 기능을 데이터의 입력, 데이터 처리, 데이터 출력으로 구분하여 세 가지 형태로 모듈화 하는 방법을 제시하고 있다.

모델1 방식의 설계모델은 그림 2-1와 같이 JSP 페이지와 자바빈 클래스 두 가지 모듈로 나누어 웹 애플리케이션을 구현하도록 제안하고 있다.

JSP 페이지는 클라이언트의 요청과 함께 입력된 데이터를 받고 그 요청에 대한 처리 결과를 출력하는 역할을 담당하고 자바빈 클래스는 비즈니스 로직 처리를 담당한다.

클라이언트의 요청을 받은 JSP 페이지에서 요청에 대한 처리를 자바빈 클래스나 DAO(Data Access Object) 클래스를 이용해 데이터베이스에서 데이터를 읽어오거나 데이터베이스에 저장하고 그 결과를 현재 JSP 페이지 또는 forward 기법을 통해 다른 JSP 페이지로 제어를 이동해 출력하게 된다.

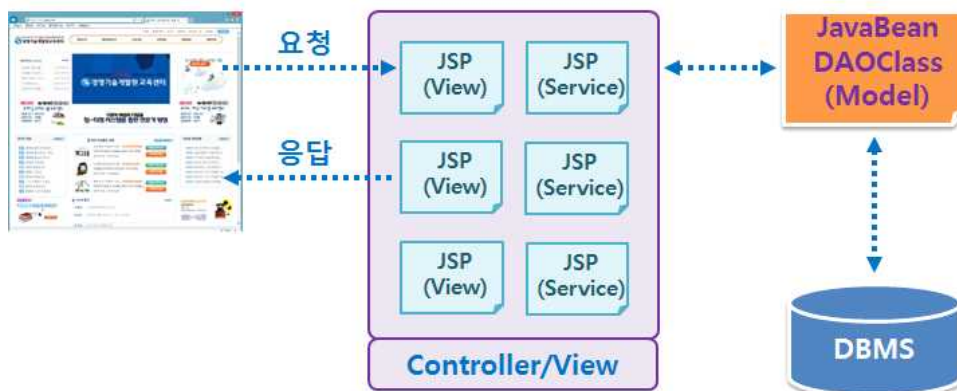


그림 2-1 모델1 아키텍처

모델2 방식의 설계모델은 그림 2-2와 같이 서블릿 클래스, JSP 페이지, 모델 클래스 세 가지 모듈로 나누어 구현하도록 제안하고 있다.

서블릿 클래스는 클라이언트의 요청과 함께 입력된 데이터를 받고 그 요청을 처리하기 위한 애플리케이션의 흐름을 제어하는 컨트롤러(Controller) 역할을 담당하고 JSP 페이지는 사용자 요청에 대한 처리 결과를 출력하는 뷰(View) 역할을 담당하며 모델 클래스는 데이터 저장소(디렉토리 서비스, 데이터베이스 등)와 연동해 클라이언트로부터 전달된 데이터를 저장하거나 필요한 데이터를 읽어와 뷰에 출력할 결과를 만드는 모델(Model) 역할을 담당한다.

MVC 패턴은 모델(Model) - 뷰(View) - 컨트롤러(Controller)의 세 영역으로 나누어 애플리케이션을 구현하는 패턴으로 현재 대부분의 상용 애플리케이션을 구현할 때 사용되는 소프트웨어 구현 패턴이다.

JSP 규격서에서 제안하고 있는 모델2 방식의 웹 애플리케이션 설계모델은 MVC 패턴과 거의 같기 때문에 모델2 기반의 MVC 패턴이라고 부르기도 한다.

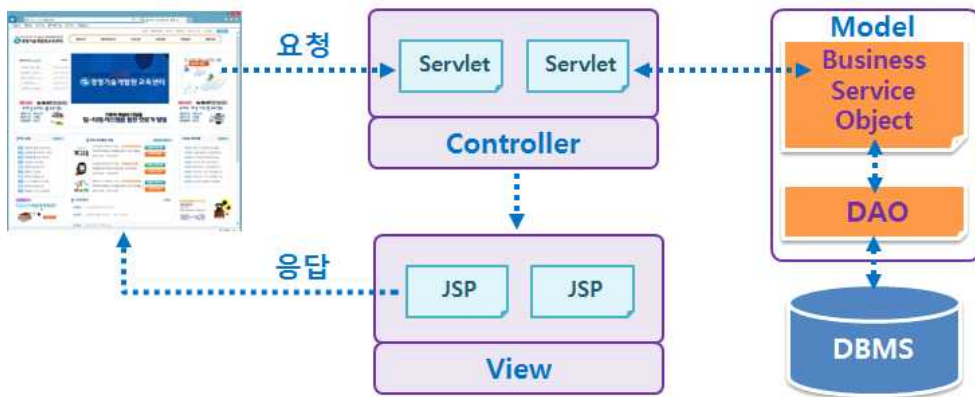


그림 2-2 모델2 아키텍처

◆ 컨트롤러(Controller)

클라이언트의 요청과 함께 입력된 데이터를 받아 분석한 후 어떤 모델 클래스가 그 요청을 처리할지 결정하고 모델 클래스를 이용해 클라이언트의 요청을 처리한 후 결과를 출력할 뷰를 호출하는 역할을 담당한다. 컨트롤러는 클라이언트의 요청을 처리하기 위해 모델과 뷰 사이에서 웹 애플리케이션의 흐름을 제어한다고 할 수 있다.

◆ 모델(Model)

클라이언트의 요청에 대한 처리 결과를 만드는 역할을 담당한다. 이 과정에서 데이터 저장소(디렉토리 서비스, 데이터베이스 등)에 접근해 데이터를 저장하거나 읽어오는 역할을 담당하는 객체를 DAO(Data Access Object)라고 하며 DAO를 퍼시스턴스 계층(Persistence Layer)이라고 구분한다. 또한 DAO를 사용해 클라이언트 요청에 대한 결과를 만드는 객체를 서비스 객체라고 하며 이 서비스 객체에서 실제 업무 대상이 되는 객체(예: 사원, 게시 글, 주문 등)를 생성하게 되는데 이 객체를 도메인 객체라 부른다. 도메인 객체는 클라이언트의 요청에 대한 처리 결과의 구성원이 되는 객체로 경우에 따라서 VO(Value Object), 자바 빈(Java Beans), DTO(Data Transfer Object)라고 부르기도 한다. 서비스 객체와 도메인 객체를 합해서 비즈니스 계층(Business Layer)이라고 구분한다.

◆ 뷰(View)

클라이언트 요청에 대한 처리 결과를 모델이라고도 하며 이 모델을 컨트롤러로부터 넘겨받아 브라우저로 전송할 화면을 만드는 역할을 담당한다.

사용자 인터페이스(UI)를 표현하기 위해 사용되는 객체를 프리젠테이션 계층(Presentation Layer)으로 구분 한다.

◆ 클라이언트 요청을 처리하는 명령처리 방식

컨트롤러에서 클라이언트가 어떤 요청을 했는지 판단하기 위해 사용되는 방식은 여러 가지를 생각할 수 있으나 요청 파라미터를 분석해 명령을 처리하는 방식과 요청 URI 자체를 분석해 명령을 처리하

는 방식이 가장 일반적으로 사용된다.

요청 파라미터를 분석해 클라이언트의 요청을 판단하는 방식은 클라이언트가 서버로 요청을 보낼 때 URL에 특정 파라미터를 추가해 요청하고 서버에서 이 파라미터를 분석해 요청을 처리하는 방식이다. 아래와 같이 요청 URL에서 command라는 파라미터를 읽어서 파라미터 값에 따라 클라이언트의 요청을 처리하는 방식이다.

`http://localhost:8080/JSPStudyMvcBBS/index?command=boardList`

`http://localhost:8080/JSPStudyMvcBBS/index?command=boardDetail`

위에서 언급한 요청 파라미터를 이용한 명령어 처리 방식은 요청할 때 마다 항상 파라미터를 붙여 처리해야 하는 번거로움이 따른다. 하지만 URI 명령어 처리 방식은 요청 URI를 분석해 클라이언트의 요청을 처리하는 방식으로 요청된 URL에서 명령을 분석해 처리하기 때문에 별도의 파라미터가 필요 없다.

요즘은 URI 명령어 처리 방식이 가장 일반적으로 사용되면 이 방식은 요청 URI에서 컨텍스트 패스(ContextPath)를 제외한 나머지 문자열을 추출해서 클라이언트의 요청이 무엇인지 분석하여 요청을 처리하는 방식이다. 아래와 같은 요청 URL에서 요청 URI(/JSPStudyMvcBBS/boardList.mvc)를 추출하고 추출한 문자열에서 다시 컨텍스트 패스(/JSPStudyMvcBBS)를 제외하여 나머지 문자열을 추출하면 /boardList.mvc와 /boardDetail.mvc가 남는데 이 문자열을 가지고 클라이언트의 요청이 어떤 요청인지 분석해서 처리하는 방식이 바로 URI 명령어 처리 방식이다. 이 방식을 URI 커맨드 패턴(URI Command Pattern)이라고도 부른다.

`http://localhost:8080/JSPStudyMvcBBS/boardList.mvc`

`http://localhost:8080/JSPStudyMvcBBS/boardDetail.mvc`

1) MVC 패턴 게시판

일반적으로 MVC 패턴에서 서블릿 클래스를 사용해 컨트롤러를 구현하는데 각각의 요청마다 서블릿 클래스를 작성해 컨트롤러로 사용할 수 있지만 이렇게 되면 코드의 중복이 많이 발생하기 때문에 보통은 하나의 서블릿이 모든 요청을 받을 수 있도록 컨트롤러를 구현한다. 이렇게 하나의 서블릿이 모든 요청을 처리하는 형태가 되기 때문에 이런 애플리케이션 구현 패턴을 프론트 컨트롤러 패턴이라고 부르기도 한다.

프론트 컨트롤러 패턴은 꼭 하나의 서블릿이 모든 요청을 처리하도록 구현해야만 하는 것은 아니며 각각 카테고리 별로 나눠서 주문관련 요청을 처리하는 컨트롤러, 회원관련 요청을 처리하는 컨트롤러, 정산관련 요청을 처리하는 컨트롤러 등으로 나눠서 구현할 수도 있다.

1-1) if문을 이용한 프론트 컨트롤러 구현

- 실습용 프로젝트 : JSPClassMvcBBS01
- 완성 프로젝트 : JSPStudyMvcBBS01

그 동안 모델1 방식으로 웹 애플리케이션을 구현하는 방법을 학습해 온 입문자들이 모델2 기반의 MVC 패턴을 접하게 되면 많이 어려워하는 모습을 보게 된다. 그도 그럴 것이 프로그래밍을 처음 배우는 입문자들이 업무절차에 따라서 순서대로 JSP 페이지를 구성해 요청을 처리하는 모델1 방식도 어려워하고 있는데 각각의 역할을 나눠서 클래스를 분리하고 분리된 클래스의 객체를 활용해 MVC 패턴으로 웹 애플리케이션을 구현하는 프로그래밍 방식은 전체 웹 애플리케이션의 흐름을 이해하는데 보다 많은 노력이 필요하다. 하지만 실무에서는 MVC 패턴을 활용한 웹 애플리케이션 구현 방식이 대세를 이루고 있기 때문에 MVC 패턴을 활용한 웹 애플리케이션 구현 능력을 향상시키는데 노력해야 한다. 그래서 조금은 이해하기 쉽도록 첫 번째 MVC 패턴 예제는 우리가 익숙하게 사용했던 if문을 사용해 요청 명령을 분석하여 어떤 모델 클래스가 실행될지를 결정하는 컨트롤러를 구현할 것이다.

이번 예제는 모델1 방식으로 검색 기능까지 구현한 JSPStudyBBS04 프로젝트를 MVC 패턴 게시판으로 변경해 구현할 것이다. 먼저 JSPStudyBBS04 프로젝트를 JSPClassMvcBBS01로 프로젝트 명을 변경하고 게시 글 리스트 요청을 처리하는 컨트롤러와 모델 클래스 그리고 뷰를 구현하면서 모델2 방식의 MVC 패턴으로 웹 애플리케이션을 구현하는 방법에 대해 구체적으로 알아볼 것이다.

1-1-1) DBCP 설정

- META-INF/context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
  <Resource
    name="jdbc/bbsDBPool"
    auth="Container"
    type="javax.sql.DataSource"
    driverClassName="oracle.jdbc.driver.OracleDriver"
    url="jdbc:oracle:thin:@localhost:1521:xe"
    username="hr"
    password="hr"
    factory="org.apache.commons.dbcp2.BasicDataSourceFactory"
    maxActive="10"
    maxIdle="5" />
</Context>
```

1-1-2) DB Connection을 관리하는 유틸 클래스

JDBC 프로그래밍을 하다보면 SQLException을 처리하면서 중복되는 try{} catch(); 코드를 많이 사용하게 된다. 그래서 중복 코드를 없애고 데이터베이스 커넥션과 데이터베이스 자원해제 등의 작업을 효율적으로 관리하기 위해서 DB Connection을 관리해 주는 DBManager라는 유틸 클래스를 새롭게 만들어 사용할 것이다. 앞에서는 BoardDao 클래스의 생성자에서 직접 DataSource를 구하고 각각의 메서드에서 이 DataSource로부터 Connection 객체를 구하여 데이터베이스 작업을 하였지만 이제 부터는 BoardDao에서 아래에 정의하는 DBManager 클래스를 통해서 DB Connection 객체를 구하여 사용할 것이다.

- com.jspstudy.bbs.dao.DBManager

//DB Connection을 관리하는 클래스

```
public class DBManager {
```

```
/* DataSource 객체는 데이터 원본과 연결할 수 있도록 지원하는 객체
 * JNDI 방식으로 DBCP를 찾아 DBCP에서 Connection 객체를 대여하는 객체
 */
```

```
private static DataSource DS = null;
```

```
// Connection 객체는 DB에 연결해 작업을 수행할 수 있도록 지원하는 객체
```

```
private static Connection CONN = null;
```

```
/* 외부에서 new 연산자를 이용해 인스턴스를 생성하지 못하도록
 * 접근 제어자를 private으로 지정해 생성자를 선언하고 있다.
 *
 * 이 클래스의 모든 메소드는 static으로 선언되어 있으므로 외부에서
 * 인스턴스를 생성할 필요없이 클래스 이름으로 메소드에 접근할 수 있다.
 */
```

```
private DBManager() { }
```

```
/* static 초기화 블록은 클래스가 메모리에 로딩된 직후에 실행된다.
```

```
 * static 초기화 블록은 생성자 보다 먼저 실행된다.
```

```
*/
```

```
static {
```

```
    try {
```

```
        /* 1. 자바 네이밍 서비스를 사용하기 위해
```

```
        * javax.naming 패키지의 InitialContext 객체를 생성한다.
```

```
        *
```

```
        * 이 예제는 commons-dbc2-2.8.0.jar, commons-pool2-2.10.0.jar
```

```
        * 그리고 commons-logging-1.2.jar를 참조하므로 이들 라이브러리를
```

```
        * http://commons.apache.org에서 다운 받아 WEB-INF/lib에 추가해야 한다.
```

```
        * 또한 Oracle 접속 드라이버도 필요하므로 http://www.oracle.com에서
```

```
        * 다운로드 받아 WEB-INF/lib 폴더에 추가해야 한다.
```

```
        */
```

```
        Context initContext = new InitialContext();
```

```
        /* 2. InitialContext 객체를 이용해 디렉토리 서비스에서 필요한 객체를
```

```
        * 찾기 위해 기본 네임스페이스를 인자로 지정해 Context 객체를 얻는다.
```

```
        * 디렉토리 서비스에서 필요한 객체를 찾기 위한 일종의 URL 개념으로
```

```
        * 디렉토리 방식을 사용하므로 "java:comp/env"와 같이 지정한다.
```

```
        *
```

```
        * java:comp/env는 모든 설정된 엔트리와 자원(Resource)이 JNDI namespace의
```

```
        * java:comp/env라는 부분에 놓이게 되고 자원에 대해 접근하려면
```

```
        * lookup("java/comp/env") 메소드를 통해 로컬 리소스에 접근 한다.
```

```
        * 즉 java:comp/env는 JNDI에서 록업할 때 사용하는 빈(객체)의 가상 디렉토리
```

```
        * 공간으로 기본 가상 디렉토리를 java:comp/env로 정해 놓은 것이다.
```

```

    /**/
    Context envContext = (Context) initContext.lookup("java:comp/env");

    /* 3. "jdbc/bbsDBPool" 이름을 가진 DBCP에서 DataSource 객체를 얻는다.
     * context.xml 파일에서 지정한 수의 커넥션을 생성해 커넥션 풀에 저장한다.
     * "java:/comp/env"는 JNDI에서 기본적으로 사용하는 네임스페이스 이고
     * "jdbc/mallDBPool"은 DBCP 이름으로 임의로 지정하여 사용할 수 있다.
     */
    DS = (DataSource) envContext.lookup("jdbc/bbsDBPool");

} catch(Exception e) {
    e.printStackTrace();
}
}

// DBMS에 접속하고 활성화된 Connection 객체를 반환하는 메소드
public static Connection getConnection() {
    try {
        // 4. DataSource 객체를 이용해 DBCP로 부터 커넥션을 대여하여 반환 한다.
        CONN = DS.getConnection();

    } catch(Exception e) {
        e.printStackTrace();
    }
    return CONN;
}

// DB 작업에 사용된 자원을 해제하는 메소드
public static void close(Connection conn, PreparedStatement pstmt) {

    try {
        /* 매개변수로 넘어온 pstmt, conn이 null이 아니면 자원을 해제 한다.
         * DB 작업과 관련된 개체를 닫을 때는 객체가 생성된 역순으로 닫는다.
         */
        if(pstmt != null) pstmt.close();
        if(conn != null) conn.close();

    } catch(SQLException e) { e.printStackTrace(); }
}

// DB 작업에 사용된 자원을 해제하는 메소드
public static void close(Connection conn,

```

```

        PreparedStatement pstmt, ResultSet rs) {
    try {

        /* 매개변수로 넘어온 rs, pstmt, conn이 null이 아니면 자원을 해제 한다.
         * DB 작업과 관련된 개체를 닫을 때는 객체가 생성된 역순으로 닫는다.
         */
        if(rs != null) rs.close();
        if(pstmt != null) pstmt.close();
        if(conn != null) conn.close();

    } catch(SQLException e) { e.printStackTrace(); }
}

// Transaction 시작
public static void setAutoCommit(Connection conn, boolean isAutoCommit) {
    try {
        if(conn != null) conn.setAutoCommit(isAutoCommit);

    } catch(SQLException e) {
        e.printStackTrace();
    }
}

// Transaction 종료
public static void commit(Connection conn) {
    try {
        if(conn != null) conn.commit();

    } catch(SQLException e) {
        e.printStackTrace();
    }
}

// Transaction 롤백
public static void rollback(Connection conn) {
    try {
        if(conn != null) conn.rollback();

    } catch(SQLException e) {
        e.printStackTrace();
    }
}

```

```
}
```

1-1-3) VO(Value Object) 클래스

앞의 모델1 게시판인 JSPStudyBBS04 프로젝트의 Board 클래스와 동일한 클래스이다.

- com.jspstudy.bbs.vo.Board

```
/* 하나의 게시 글 정보를 저장하는 VO(Value Object)
 * VO 객체에 저장될 데이터는 테이블에서 읽어오기 때문에 각각의 변수는
 * 테이블의 컬럼이 가지는 데이터 형식과 같거나 자동 형 변환이 가능해야 한다.
 */
public class Board {
    // no, title, writer, content, reg_date, read_count, pass, file1
    private int no;
    private String title;
    private String writer;
    private String content;
    private Timestamp regDate;
    private int readCount;
    private String pass;
    private String file1;

    public Board() { }
    public Board(int no, String title, String writer, String content,
        Timestamp regDate, int readCount, String pass, String file1) {
        this.no = no;
        this.title = title;
        this.writer = writer;
        this.content = content;
        this.regDate = regDate;
        this.readCount = readCount;
        this.pass = pass;
        this.file1 = file1;
    }

    public int getNo() {
        return no;
    }
    public void setNo(int no) {
        this.no = no;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
```

```

        this.title = title;
    }
    public String getWriter() {
        return writer;
    }
    public void setWriter(String writer) {
        this.writer = writer;
    }
    public String getContent() {
        return content;
    }
    public void setContent(String content) {
        this.content = content;
    }
    public Timestamp getRegDate() {
        return regDate;
    }
    public void setRegDate(Timestamp regDate) {
        this.regDate = regDate;
    }
    public int getReadCount() {
        return readCount;
    }
    public void setReadCount(int readCount) {
        this.readCount = readCount;
    }
    public String getPass() {
        return pass;
    }
    public void setPass(String pass) {
        this.pass = pass;
    }
    public String getFile1() {
        return file1;
    }
    public void setFile1(String file1) {
        this.file1 = file1;
    }
}

```

1-1-4) DAO(DataAccessObject) 클래스

BoardDao 클래스는 앞에서 모델1 방식의 게시판을 구현할 때는 기본 생성자 안에서 DBCP에 접근할 수 있는 DataSource 객체를 구하고 BoardDao 클래스의 각각의 메서드에서 DataSource 객체를 이용해 Connection 객체를 구하여 데이터베이스 작업을 진행하였다. 하지만 이번 예제부터 새롭

게 추가된 데이터베이스 접속과 자원 해제 등을 관리해 주는 DBManager 클래스를 사용할 것이므로 앞에서 사용했던 BoardDao의 기본 생성자는 삭제할 것이다. 또한 BoardDao의 각각의 메서드에서 DBManager 클래스를 이용해 Connection 객체를 구하는 부분과 데이터베이스 작업이 완료되면 데이터베이스 작업에 사용했던 자원을 해제하는 부분은 DBManager 클래스를 이용할 것이므로 이 부분의 코드를 수정하면 되고 나머지는 앞의 모델1 예제와 거의 동일한 코드가 될 것이다.

- com.jspstudy.bbs.dao.BoardDao

// DB 작업을 전담하는 DAO(Data Access Object) 클래스

```
public class BoardDao {
```

```
    // 데이터베이스 작업에 필요한 객체 타입으로 변수를 선언
```

```
    // Connection 객체는 DB에 연결해 작업을 수행할 수 있도록 지원하는 객체
```

```
    private Connection conn;
```

```
    // Statement, PreparedStatement 객체는 DB에 쿼리를 발행하는 객체
```

```
    private PreparedStatement pstmt;
```

```
    // ResultSet 객체는 DB에 SELECT 쿼리를 발행한 결과를 저장하는 객체
```

```
    private ResultSet rs;
```

```
    /* 검색어에 해당하는 게시 글 수를 계산하기 위해 호출되는 메서드
```

```
    * DB 테이블에 등록된 모든 게시 글의 수를 반환하는 메서드
```

```
    */
```

```
    public int getBoardCount(String type, String keyword) {
```

```
        System.out.println(type + " - " + keyword);
```

```
    /* 이 부분에서 우리는 SQL 쿼리를 작성하는데 주의를 기울여야 한다.
```

```
    * 검색 옵션에 따라서 검색하는 컬럼이 다르기 때문에 파라미터로 받은 type을
```

```
    * 아래와 같이 쿼리의 파라미터(?)로 지정해 검색 옵션의 변경에 따라 동적으로
```

```
    * 다른 쿼리가 발행되게 하고 싶지만 이 방법은 제대로 동작하지 못한다.
```

```
    *
```

```
    * "SELECT COUNT(*) FROM jspbbs WHERE ? LIKE '%' || ? || '%" 
```

```
    *
```

```
    *
```

```
    * PreparedStatement는 기준이 되는 SQL 쿼리를 캐싱하기 때문에
```

```
    * 이 객체를 Connection 객체로부터 받아 올 때 기준 쿼리를 인수로 지정하는데
```

```
    * 이 때 WHERE에 검색 대상이 되는 컬럼명을 파라미터(?)로 지정하게 되면
```

```
    * 쿼리가 캐싱될 때 검색 대상이 되는 컬럼이 지정되지 못하게 되므로 이 쿼리의
```

```
    * 검색 결과는 항상 0이 된다.
```

```
    *
```

```
    * 기준 쿼리에서 사용되는 컬럼은 PreparedStatement 객체를 생성할 때
```

```
    * 정해져 있어야하기 때문에 아래와 같이 type을 지정하면 된다.
```

```
    */
```

```
    String sqlCount = "SELECT COUNT(*) FROM jspbbs WHERE "
```

```
        + type + " LIKE '%' || ? || '%";
```

```

//String sqlCount =
// "SELECT COUNT(*) FROM jspbbs WHERE " + type + " LIKE ?";
int count = 0;

try{
    // 1. DBManager을 이용해 DBCP로 부터 Connection을 대여한다.
    conn = DBManager.getConnection();

    /* 2. DBMS에 SQL 쿼리를 발생하기 위해 활성화된
     * Connection 객체로 부터 PreparedStatement 객체를 얻는다.
     */
    pstmt = conn.prepareStatement(sqlCount);

    /* 3. PreparedStatement 객체의 Placeholder(?)에 대응하는
     * 값을 순서에 맞게 지정하고 있다.
     */
    pstmt.setString(1, keyword);
    //pstmt.setString(1, "%" + keyword + "%");

    /* 4. PreparedStatement 객체를 이용해 DB에 SELECT 쿼리를 발행하고
     * 그 결과로 ResultSet 객체를 얻는다.
     *
     * executeQuery()는 실제 DBMS에 SELECT 쿼리를 발행하는 메소드로
     * DB에서 검색된 데이터를 가상의 테이블 형태인 ResultSet 객체로 반환한다.
     */
    rs = pstmt.executeQuery();

    /* 5. 쿼리 실행 결과를 바탕으로 요청한 게시 글 정보를 구한다.
     *
     * ResultSet 객체는 DB로 부터 읽어온 데이터에 접근하기 위해 테이블의
     * 행을 가리키는 cursor를 제공한다. 맨 처음 ResultSet 객체를 반환
     * 받으면 cursor는 첫 번째 행 바로 이전을 가리키고 있다.
     *
     * 집계함수를 사용하용하기 때문에 쿼리 결과가 단일 행이므로 if문을 사용했다.
     */
    if(rs.next()) {
        count = rs.getInt(1);
    }
} catch(Exception e) {
    e.printStackTrace();
} finally {
    // 6. DBManager를 이용해 Connection을 DBCP에 반납한다.
    DBManager.close(conn, pstmt, rs);
}

```



```

    return count;
}

/* 제목, 작성자, 내용에서 검색어가 포함된 게시 글 검색 시 호출되는 메서드
 * 요청한 페이지에 해당하는 검색 결과를 DB에서 읽어와 반환하는 메서드
 **/
public ArrayList<Board> searchList(
    String type, String keyword, int startRow, int endRow) {

    /* 검색어가 포함된 게시 글 리스트를 추출하기 위한 쿼리
     * 테이블에서 현재 페이지에 해당하는 게시 글을 검색할 때 ROWNUM을 사용했다.
     * ROWNUM은 쿼리의 결과로 검색되는 행들의 순서 값을 가진 의사컬럼으로
     * 1부터 시작한다. 최신 게시 글을 먼저 보여주기 위해 ORDER BY DESC를
     * 지정하고 요청된 페이지에 보여줄 게시 글의 시작 행과 마지막 행을 지정한다.
     *
     * 아래의 쿼리로 질의하게 되면 게시 글 전체를 글 번호에 해당하는 no를 기준으로
     * 내림차순 정렬하여 검색하고 WHERE 절에 지정한 첫 번째 Placeholder(?)에
     * 해당하는 시작 행부터 두 번째 Placeholder(?)에 해당하는 마지막 행까지의
     * 게시 글을 추출할 수 있다.
     **/
    String sqlSearchList = "SELECT * FROM (SELECT ROWNUM num, no, title,"
        + " writer, content, reg_date, read_count, pass, file1 FROM"
        + " (SELECT * FROM jsppbbs WHERE " + type + " LIKE ?"
        + " ORDER BY no DESC)) WHERE num >= ? AND num <= ?";

    ArrayList<Board> boardList = null;

    try{
        conn = DBManager.getConnection();

        pstmt = conn.prepareStatement(sqlSearchList);
        pstmt.setString(1, "%" + keyword + "%");
        pstmt.setInt(2, startRow);
        pstmt.setInt(3, endRow);

        rs = pstmt.executeQuery();

        if(rs.next()) {

            boardList = new ArrayList<Board>();

            do {
                Board board = new Board();
                board.setNo(rs.getInt("no"));

```

```

        board.setTitle(rs.getString("title"));
        board.setContent(rs.getString("content"));
        board.setWriter(rs.getString("writer"));
        board.setRegDate(rs.getTimestamp("reg_date"));
        board.setReadCount(rs.getInt("read_count"));
        board.setPass(rs.getString("pass"));
        board.setFile1(rs.getString("file1"));

        boardList.add(board);

    } while(rs.next());
}
} catch(Exception e) {
    e.printStackTrace();

} finally {
    DBManager.close(conn, pstmt, rs);
}
return boardList;
}

/* 전체 게시 글 수를 계산하기 위해 호출되는 메서드 - paging 처리에 사용
 * DB 테이블에 등록된 모든 게시 글의 수를 반환하는 메서드
 **/
public int getBoardCount() {

    String sqlCount = "SELECT COUNT(*) FROM jspbbs";
    int count = 0;

    try{
        conn = DBManager.getConnection();
        pstmt = conn.prepareStatement(sqlCount);
        rs = pstmt.executeQuery();
        while(rs.next()) {
            count = rs.getInt(1);
        }
    } catch(Exception e) {
        e.printStackTrace();
    } finally {
        DBManager.close(conn, pstmt, rs);
    }
    return count;
}

```

```

/* 한 페이지에 보여 질 게시 글 리스트 요청시 호출되는 메소드
 * 요청한 페이지에 해당하는 게시 글 리스트를 DB에서 읽어와 반환하는 메소드
 */
public ArrayList<Board> boardList(int startRow, int endRow) {

    /* 요청한 페이지에 해당하는 게시 글 리스트를 추출하기 위한 쿼리
     * 테이블에서 현재 페이지에 해당하는 게시 글을 검색할 때 ROWNUM을 사용했다.
     * ROWNUM은 쿼리의 결과로 검색되는 행들의 순서 값을 가진 의사컬럼으로
     * 1부터 시작한다. 최신 게시 글을 먼저 보여주기 위해 ORDER BY DESC를
     * 지정하고 요청된 페이지에 보여줄 게시 글의 시작 행과 마지막 행을 지정한다.
     *
     * 아래의 쿼리로 질의하게 되면 게시 글 전체를 글 번호에 해당하는 no를 기준으로
     * 내림차순 정렬하여 검색하고 WHERE 절에 지정한 첫 번째 Placeholder(?)에
     * 해당하는 시작 행 부터 두 번째 Placeholder(?)에 해당하는 마지막 행까지의
     * 게시 글을 추출할 수 있다.
     */
    String sqlBoardList = "SELECT * FROM (SELECT ROWNUM num,"
        + " no, title, writer, content, reg_date, read_count, pass, file1 FROM"
        + " (SELECT * FROM jspbbs ORDER BY no DESC)) "
        + " WHERE num >= ? AND num <= ?";

    ArrayList<Board> boardList = null;

    try{
        conn = DBManager.getConnection();
        pstmt = conn.prepareStatement(sqlBoardList);
        pstmt.setInt(1, startRow);
        pstmt.setInt(2, endRow);
        rs = pstmt.executeQuery();

        if(rs.next()) {

            boardList = new ArrayList<Board>();

            do {
                Board board = new Board();
                board.setNo(rs.getInt("no"));
                board.setTitle(rs.getString("title"));
                board.setContent(rs.getString("content"));
                board.setWriter(rs.getString("writer"));
                board.setRegDate(rs.getTimestamp("reg_date"));
                board.setReadCount(rs.getInt("read_count"));
                board.setPass(rs.getString("pass"));
                board.setFile1(rs.getString("file1"));
            } while (rs.next());
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

```

        boardList.add(board);

    } while(rs.next());
}
} catch(Exception e) {
    e.printStackTrace();

} finally {
    DBManager.close(conn, pstmt, rs);
}
return boardList;
}

```

/* 게시 글 내용 보기 요청 시 호출되는 메서드

* no에 해당하는 게시 글 을 DB에서 읽어와 Board 객체로 반환하는 메서드

*

* 이 메서드는 앞의 예제에서 파라미터를 게시 글 번호에 해당하는 no만 받았지만

* 이 번 예제에서는 no와 함께 게시 글 읽은 횟수의 증가 여부를 boolean 형으로

* 두 번째 파라미터로 받아서 이 값이 true 면 게시 글 읽을 횟수를 하나 증가하고

* false 면 증가하지 않는 코드가 추가 되었다. 아래와 같이 하나의 논리적인 작업

* 안에서 DB에 쿼리가 여러 번 발행되게 되면 SELECT 쿼리인 경우에는 큰 문제가

* 없겠지만 하나의 논리적인 작업 단위에서 추가, 수정, 삭제 작업을 여러 번 해야 할

* 경우에 이 과정에서 중간에 에러가 발생되면 앞의 쿼리는 DB에 적용되고 뒤에 쿼리는

* 오류로 인해서 제대로 DB에 적용되지 못하는 문제가 발생한다. 이렇게 하나의 작업

* 안에서 쿼리가 여러 번 발행될 때 모두 DB에 반영되거나 또는 중간에 에러가 발생하면

* 현재 작업 이전의 상태로 되돌려서 문제가 발생되지 않도록 해야 하는데 여러 번의

* 쿼리 발행을 하나의 논리적인 작업 단위로 묶어서 전체를 적용하여 commit 하거나

* 또는 중간에 문제가 생겨서 이전 작업 상태로 되돌려 rollback 해서 DB에서 문제가

* 발생되지 않도록 처리하는 것을 트랜잭션(Transaction) 처리라고 한다.

**/

```

public Board getBoard(int no, boolean state) {

```

```

    String boardSql = "SELECT * FROM jspbbs WHERE no=?";

```

```

    String countSql = "UPDATE jspbbs set read_count = read_count + 1 "
        + "WHERE no = ?";

```

```

    Board board = null;

```

```

    try{

```

```

        // 1. DBManager을 이용해 DBCP로 부터 Connection을 대여한다.

```

```

        conn = DBManager.getConnection();

```

```

        // 활성화된 Connection에 트랜잭션을 시작한다.

```

```

        DBManager.setAutoCommit(conn, false);

```

```

// 게시 글 조회 요청일 때 state는 true로 게시 글 조회 수를 1증가 시킨다.
if(state) {
    /* 2. DBMS에 SQL 쿼리를 발생하기 위해 활성화된
     * Connection 객체로 부터 PreparedStatement 객체를 얻는다.
     */
    pstmt = conn.prepareStatement(countSql);

    /* 3. PreparedStatement 객체의 Placeholder(?)에 대응하는
     * 값을 순서에 맞게 지정하고 있다.
     */
    pstmt.setInt(1, no);

    /* 4. 데이터베이스에 UPDATE 쿼리를 발행해 조회수를 1증가 시킨다.
     *
     * executeUpdate()는 DBMS에 INSERT, UPDATE, DELETE 쿼리를
     * 발행하는 메소드로 추가, 수정, 삭제된 레코드의 개수를 반환 한다.
     */
    pstmt.executeUpdate();
}

/* 2. DBMS에 SQL 쿼리를 발생하기 위해 활성화된
 * Connection 객체로 부터 PreparedStatement 객체를 얻는다.
 */
pstmt = conn.prepareStatement(boardSql);

/* 3. PreparedStatement 객체의 Placeholder(?)에 대응하는
 * 값을 순서에 맞게 지정하고 있다.
 */
pstmt.setInt(1, no);

/* 4. PreparedStatement를 이용해 SELECT 쿼리를 발행한다.
 *
 * executeQuery()는 실제 DBMS에 SELECT 쿼리를 발행하는 메소드로
 * DB에서 검색된 데이터를 가상의 테이블 형태인 ResultSet 객체로 반환 한다.
 */
rs = pstmt.executeQuery();

/* 5. 쿼리 실행 결과를 바탕으로 요청한 게시 글 정보를 구한다.
 *
 * ResultSet 객체는 DB로 부터 읽어온 데이터에 접근하기 위해 테이블의
 * 행을 가리키는 cursor를 제공한다. 맨 처음 ResultSet 객체를 반환
 * 받으면 cursor는 첫 번째 행 바로 이전을 가리키고 있다.
 *
 * 테이블의 PRIMARY KEY인 no에 해당하는 게시 글을 SELECT 해서
 * ResultSet에는 게시 글 하나의 정보만 존재하기 때문에 if 문을 사용했다.

```

```

    **/
    if(rs.next()) {
        board = new Board();
        board.setNo(rs.getInt("no"));
        board.setTitle(rs.getString("title"));
        board.setContent(rs.getString("content"));
        board.setWriter(rs.getString("writer"));
        board.setRegDate(rs.getTimestamp("reg_date"));
        board.setReadCount(rs.getInt("read_count"));
        board.setPass(rs.getString("pass"));
        board.setFile1(rs.getString("file1"));

        // 모든 작업이 완료되면 커밋하여 트랜잭션을 종료한다.
        DBManager.commit(conn);

    } catch(Exception e) {
        // DB 작업이 하나라도 에러가 발생하면 롤백하고 트랜잭션을 종료한다.
        DBManager.rollback(conn);

        System.out.println("BoardDao - getBoard(no, state)");
        e.printStackTrace();
    } finally {
        // 6. DBManager를 이용해 Connection을 DBCP에 반납한다.
        DBManager.close(conn, pstmt, rs);
    }
    // 요청한 하나의 게시 글을 반환 한다.
    return board;
}

/* 게시 글쓰기 요청시 호출되는 메서드
 * 게시 글을 작성하고 등록하기 버튼을 클릭하면 게시 글을 DB에 추가하는 메서드
 */
public void insertBoard(Board board) {

    String sqlInsert = "INSERT INTO jspbbs(no, title, writer, content,"
        + " reg_date, read_count, pass, file1) "
        + " VALUES(jspbbs_seq.NEXTVAL, ?, ?, ?, SYSDATE, 0, ?, ?)";

    try {
        conn = DBManager.getConnection();

        pstmt = conn.prepareStatement(sqlInsert);
        pstmt.setString(1, board.getTitle());
        pstmt.setString(2, board.getWriter());
    }

```

```

        pstmt.setString(3, board.getContent());
        pstmt.setString(4, board.getPass());
        pstmt.setString(5, board.getFile1());

        pstmt.executeUpdate();

    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        DBManager.close(conn, pstmt, rs);
    }
}

/* 게시 글 수정, 게시 글 삭제 시 비밀번호 입력을 체크하는 메서드
**/
public boolean isPassCheck(int no, String pass) {
    boolean isPass = false;
    String sqlPass = "SELECT pass FROM jspbbs WHERE no=?";
    try {
        conn = DBManager.getConnection();

        pstmt = conn.prepareStatement(sqlPass);
        pstmt.setInt(1, no);

        rs = pstmt.executeQuery();

        if(rs.next()) {
            isPass = rs.getString(1).equals(pass);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        DBManager.close(conn, pstmt, rs);
    }
    return isPass;
}

/* 게시 글 수정 요청시 호출되는 메서드
* 게시 글을 수정하고 수정하기 버튼을 클릭하면 게시 글을 DB에 수정하는 메서드
**/
public void updateBoard(Board board) {

    String sqlUpdate = "UPDATE jspbbs set title=?, writer=?, content=?"

```

```

        + " reg_date=SYSDATE, file1=? WHERE no=?";

try {
    conn = DBManager.getConnection();

    pstmt = conn.prepareStatement(sqlUpdate);
    pstmt.setString(1, board.getTitle());
    pstmt.setString(2, board.getWriter());
    pstmt.setString(3, board.getContent());
    pstmt.setString(4, board.getFile1());
    pstmt.setInt(5, board.getNo());

    pstmt.executeUpdate();

} catch(Exception e) {
    e.printStackTrace();
} finally {
    DBManager.close(conn, pstmt);
}
} // end updateBoard(Board board)

/* 게시 글 삭제 요청 시 호출되는 메서드
 * no에 해당 하는 게시 글을 DB에서 삭제하는 메서드
 */
public void deleteBoard(int no) {

    String sqlDelete = "DELETE FROM jspbbs WHERE no=?";
    try {
        conn = DBManager.getConnection();

        pstmt = conn.prepareStatement(sqlDelete);
        pstmt.setInt(1, no);

        pstmt.executeUpdate();

    } catch(Exception e) {
        e.printStackTrace();
    } finally {
        DBManager.close(conn, pstmt);
    }
} // end deleteBoard(int no);
}

```


1-1-5) 게시판 관련 요청을 처리하는 프런트 컨트롤러 클래스

- com.jspstudy.bbs.controller.BBSController

/* 모델2 방식으로 게시판 구현하기

* 게시판과 관련된 요청을 if문을 사용해 처리하는 Controller

*/

```
@WebServlet(name="bbsController", urlPatterns="*.mvc",
    initParams=@WebInitParam(name="uploadDir", value="upload"))
public class BBSController extends HttpServlet {
```

/* 뷰 페이지 정보 중에서 앞부분과 뒷부분에서 중복되는 데이터를

* 최소화하기 위해서 사용하는 접두어와 접미어를 상수로 설정

*/

```
private final String PREFIX = "/WEB-INF/";
```

```
private final String SUFFIX = ".jsp";
```

// 서블릿 초기화 메서드

```
@Override
```

```
public void init() throws ServletException {
```

/* 이 클래스에 애노테이션으로 선언한 서블릿 초기화 파라미터를 읽는다.

```
String uploadDir = getInitParameter("uploadDir");
```

/* 서블릿 초기화 파라미터로 읽어온 이미지가 저장될 폴더의

* 로컬 경로를 구하여 그 경로와 파일명으로 File 객체를 생성한다.

*/

```
String realPath = getServletContext().getRealPath(uploadDir);
```

```
File parentFile = new File(realPath);
```

/* 파일 객체에 지정한 위치에 디렉토리가 존재하지 않거나

* 파일 객체가 디렉토리가 아니라면 디렉토리를 생성한다.

*/

```
if(! (parentFile.exists() && parentFile.isDirectory())) {
```

```
    parentFile.mkdir();
```

```
}
```

/* ServletContext 객체의 속성으로 저장해 파일 업로드

* 디렉토리 정보를 다른 컴포넌트에서 사용할 수 있도록 하였다.

*/

```
getServletContext().setAttribute("uploadDir", uploadDir);
```

```
getServletContext().setAttribute("parentFile", parentFile);
```

```
System.out.println("init - " + parentFile);
```

```
}
```

// get 방식의 요청을 처리하는 메소드

```
protected void doGet(
```

```

        HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
doProcess(request, response);
}

// post 방식의 요청을 처리하는 메소드
protected void doPost(
    HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    request.setCharacterEncoding("utf-8");
    doProcess(request, response);
}

/* doGet(), doPost()에서 호출하는 메소드
 * 즉 get방식과 post방식 요청을 모두 처리하는 메소드
 * 컨트롤러는 이 메소드 안에서 브라우저의 요청에 대한 처리를 요청 URL을 분석해
 * 요청을 처리할 모델 클래스를 결정하고 해당 모델 클래스의 객체를 사용해(위임)
 * 클라이언트의 요청을 처리한 후 그 결과를 뷰로 전달해 결과 화면을 만들게 된다.
 * 뷰로 전달된 데이터는 html 형식의 문서에 출력하여 브라우저에게 응답한다.
 */
protected void doProcess(
    HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    /* 컨트롤러에 어떤 요청이 들어왔는지 파악하기 위해 Request 객체로 부터
     * 웹 어플리케이션의 컨텍스트 루트를 포함한 요청 페이지의 URI 정보와
     * ContextPath를 얻어와 String 클래스의 substring()을 이용해
     * 웹 어플리케이션 루트부터 요청한 페이지의 정보를 추출 한다.
     */
    /* 요청 정보를 담고 있는 Request 객체로 부터 요청 URI를 구한다.
     * /JSPStudyMvcBBS01/boardList.mvc
     */
    String requestURI = request.getRequestURI();

    /* 요청 정보를 담고 있는 Request 객체로 부터 ContextPath를 구한다.
     * /JSPStudyMvcBBS01
     */
    String contextPath = request.getContextPath();
    System.out.println("uri : " + requestURI + ", ctxPath : " + contextPath);

    /* 요청 URI에서 ContextPath를 제외한 요청 명령을 추출 한다.
     * /boardList.mvc
     */
    String command = requestURI.substring(contextPath.length());
    System.out.println("command : " + command);

```

```

/* 뷰 페이지 정보를 저장할 변수
**/
String viewPage = null;

/* 요청 URI에서 추출한 명령을 비교해 요청을 처리할 모델 클래스를
 * 결정하여 객체를 생성하고 메서드를 호출해 요청을 처리한다.
**/
if(command.equals("/boardList.mvc")
    || command.equals("/*.mvc")) {

    /* 게시 글 리스트 보기가 요청된 경우의 처리
    * 한 페이지에 출력 할 게시 글 리스트를 DB로 부터 읽어오는
    * BoardListService 클래스의 인스턴스를 생성한 후 Request와
    * Response 객체를 매개변수로 requestProcess()를
    * 호출하여 게시 글 리스트 보기에 대한 요청을 처리 한다.
    **/
    BoardListService list = new BoardListService();
    viewPage = list.requestProcess(request, response);

} else if(command.equals("/boardDetail.mvc")) {

    /* 게시 글 내용보기가 요청된 경우의 처리
    * 게시 글 하나의 내용을 DB로 부터 읽어오는 BoardDetailService
    * 클래스의 인스턴스를 생성한 후 Request와 Response 객체를
    * 매개변수로 requestProcess()를 호출하여 게시 글 내용보기에
    * 대한 요청을 처리 한다.
    **/
    BoardDetailService service = new BoardDetailService();
    viewPage = service.requestProcess(request, response);

} else if (command.equals("/writeForm.mvc")) {

    /* 게시 글쓰기 폼을 요청한 경우의 처리
    * 게시 글쓰기는 뷰 페이지에 출력할 모델이 없기 때문에 별도의 Service
    * 클래스를 만들지 않고 아래에서 바로 Redirect와 View 페이지 정보를
    * 저장하는 viewPage에 게시 글쓰기 폼을 출력하는 JSP의 경로를 지정 했다.
    **/
    viewPage = "board/writeForm";

} else if(command.equals("/writeProcess.mvc")) {

    /* 게시 글쓰기 폼에서 등록하기 버튼이 클릭된 경우의 처리
    * 사용자가 작성한 게시 글을 DB에 저장하는 BoardWriteService
    * 클래스의 인스턴스를 생성한 후 Request와 Response 객체를

```

```

        * 매개변수로 requestProcess()를 호출하여 새로운 게시 글을
        * DB에 저장 한다.
        */
BoardWriteService service = new BoardWriteService();
viewPage = service.requestProcess(request, response);

} else if(command.equals("/updateForm.mvc")) {

    /* 게시 글 내용보기에서 수정하기 버튼이 클릭된 경우의 처리
    * 게시 글 수정 폼 요청을 처리하는 UpdateFormService 클래스의
    * 인스턴스를 생성한 후 Request와 Response 객체를 매개변수로
    * requestProcess()를 호출하여 게시 글 수정 폼 요청을 처리한다.
    */
    UpdateFormService service = new UpdateFormService();
    viewPage = service.requestProcess(request, response);

} else if(command.equals("/updateProcess.mvc")) {

    /* 게시 글 수정 폼에서 수정하기 버튼이 클릭된 경우의 처리
    * 게시 글 수정 요청을 처리하는 UpdateService 클래스의
    * 인스턴스를 생성한 후 Request와 Response 객체를 매개변수로
    * requestProcess()를 호출하여 게시 글을 DB에서 수정한다.
    */
    UpdateService service = new UpdateService();
    viewPage = service.requestProcess(request, response);

} else if(command.equals("/deleteProcess.mvc")) {

    /* 게시 글 내용보기에서 수정하기 버튼이 클릭된 경우의 처리
    * 게시 글 수정 폼 요청을 처리하는 UpdateFormService 클래스의
    * 인스턴스를 생성한 후 Request와 Response 객체를 매개변수로
    * requestProcess()를 호출하여 게시 글을 DB에서 삭제한다.
    */
    DeleteService service = new DeleteService();
    viewPage = service.requestProcess(request, response);
}

/* Redirect 정보와 View 페이지의 경로 정보를 저장하는 viewPage가
* null이 아니면 Redirect 여부를 판단하여 Redirect라면 Response 객체의
* sendRedirect()를 이용해 Redirect 시키고 Redirect가 아니라면
* RequestDispatcher를 이용해 View 페이지로 포워딩 시킨다.
**/
if(viewPage != null) {

    /* 모델 클래스가 반환한 viewPage에 "redirect" 또는 "r" 접두어가

```

```

* 존재하면 아래의 viewPage.split(":")[0] 코드에서 "redirect" 또는
* "r" 문자열이 반환되고 그렇지 않으면 Forward 할 뷰 페이지의 경로가
* 반환되므로 다음과 같이 Redirect와 Forward를 구분하여 처리할 수 있다.
**/
String view = viewPage.split(":")[0];
System.out.println("view : " + view);

if(view.equals("r") || view.equals("redirect")) {
    response.sendRedirect(viewPage.split(":")[1]);

} else {

    /* PREFIX는 view 정보 중 앞에서 중복되는 부분을 없애기 위해 사용
    * SUFFIX는 view 정보 중 뒤에서 중복되는 부분을 없애기 위해 사용
    **/
    RequestDispatcher rd =
        request.getRequestDispatcher(PREFIX + view + SUFFIX);
    rd.forward(request, response);
}
}
}
}

```

1-1-6) 게시 글 리스트

모델1에서는 요청을 받고 그 요청에 대한 결과를 출력할 데이터(DB 작업을 통해 만들어지는 결과 데이터, 요청을 처리하고 만들어진 결과 데이터를 모델이라고 부른다.)를 만들어 HTML 문서로 화면에 출력하는 모든 작업들을 JSP 페이지에서 처리했었다. 하지만 MVC 패턴은 요청을 받아 그 요청을 어떤 클래스가 처리할지를 결정하는 Controller, 요청에 대한 결과를 만드는 Model, 그리고 요청을 처리한 결과를 HTML 문서로 작성하여 브라우저 화면에 제대로 출력되도록 만드는 View를 각각 구분해서 구현하도록 제안하고 있다. 그러므로 모델1 게시판(JSPStudyBBS04)에서 게시 글 리스트 요청을 처리할 때 boardList.jsp에서 작업했던 모든 내용 중에서 실제 요청을 받는 부분은 Controller(boardList.mvc 요청을 BBSController 클래스 받음)가 받도록 구현하고 JSP 페이지에서 BoardDao를 이용해서 게시 글 리스트의 결과 데이터를 만들었던 코드 부분은 Model 클래스(Model 클래스를 비즈니스 로직 계층 클래스 또는 Service 계층 클래스라고도 부른다. 그래서 클래스 이름에 Service 또는 Action 이라는 단어가 붙은 클래스 명을 많이 사용한다.)를 별도로 만들어 구현하면 된다. 또한 클라이언트의 요청이 들어오면 Controller 클래스는 요청을 분석해 어떤 Model(Service) 클래스가 실행될지를 결정하고 그 Model 클래스를 Controller 클래스에서 호출해서 필요한 요청을 처리한 후 최종적인 결과는 서버 내부에서 JSP 페이지로 제어를 이동시켜(Forward) HTML 형식으로 화면에 출력되도록 구현하면 된다. 이렇게 Controller 클래스는 클라이언트의 요청을 분석해서 Model 클래스를 호출하여 요청을 처리한 후에 Forward 또는 Redirect를 통해 뷰로 연결될 수 있도록 애플리케이션의 흐름을 제어하는 역할을 한다.

앞의 모델1 게시판인 JSPStudyBBS04 프로젝트의 boardList.jsp에서 구현했던 자바코드는 Model 클래스인 BoardListService 클래스로 옮겨서 구현하고 HTML 형식으로 화면에 출력되는 부분만 boardList.jsp에 나눠서 구현하면 된다. 게시 글 상세보기 등의 나머지 기능도 위와 마찬가지로

Model 클래스와 JSP 페이지로 나눠서 구현하면 된다. 앞의 JSPStudyBBS04 프로젝트에서 사용한 JSP 페이지의 코드가 서비스 클래스와 JSP 페이지로 나눠져서 구현되었을 뿐 코드 부분은 거의 대부분 동일하다.

◆ 게시 글 리스트 요청을 처리할 모델 클래스

- com.jspstudy.bbs.service.BoardListService

// 게시 글 리스트 보기 요청을 처리하는 서비스 클래스

```
public class BoardListService {

    // 한 페이지에 보여 줄 게시 글의 수를 상수로 선언하고 있다.
    private static final int PAGE_SIZE = 5;

    /* 한 페이지에 보여 질 페이지 그룹의 수를 상수로 선언하고 있다.
     * [이전] 1 2 3 4 5 6 7 8 9 10 [다음]
     */
    private static final int PAGE_GROUP = 10;

    public String requestProcess(
        HttpServletRequest request, HttpServletResponse response)
        throws ServletException {

        // 요청 파라미터로 넘어 온 페이지 번호를 읽어온다.
        String pageNum = request.getParameter("pageNum");
        String type = request.getParameter("type");
        String keyword = request.getParameter("keyword");

        /* pageNum이 null 이면 처음 게시 글 리스트를 요청하거나 게시 글쓰기에서
         * Redirect 되어 넘어온 요청으로 pageNum을 1페이지로 설정한다.
         * 즉 최신 게시 글의 첫 번째 페이지에 해당하는 게시 글 리스트를 화면에 출력한다.
         */
        if(pageNum == null) {
            pageNum = "1";
        }

        // 요청 파라미터의 pageNum을 int 형으로 변환하여 현재 페이지로 설정하고 있다.
        int currentPage = Integer.parseInt(pageNum);

        /* 요청한 페이지에 해당하는 게시 글의 첫 번째 행의 값을 계산한다.
         * 현재 페이지가 1일 경우 startRow는 1, 2페이지 일 경우 startRow는 6이 된다.
         *
         * 테이블에서 현재 페이지에 해당하는 게시 글을 검색할 때 ROWNUM을 사용했다.
         * ROWNUM은 쿼리의 결과로 검색되는 행들의 순서 값을 가진 의사컬럼으로
         * 1부터 시작한다. 예를 들어 3페이지에 해당하는 게시 글 리스트를 가져 온다면
         * 한 페이지에 보여줄 게시 글의 수가 5개로 지정되어 있으므로 startRow는 11이 된다.
        */
    }
}
```

```

* 즉 아래의 공식에 의해 startRow(11) = 3 * 5 - (5 - 1);
* 첫 번째 페이지 startRow = 1, 두 번째 페이지 startRow = 6이 된다.
**/
int startRow = currentPage * PAGE_SIZE - (PAGE_SIZE - 1);
//int startRow = (currentPage - 1) * PAGE_SIZE + 1;

int endRow = startRow + PAGE_SIZE - 1;

int listCount = 0;
ArrayList<Board> boardList = null;

// BoardDao 객체를 생성하고 전체 게시 글의 수를 얻어온다.
BoardDao dao = new BoardDao();

/* 요청 파라미터에서 type이나 keyword가 비어 있으면 일반
* 게시 글 리스트를 요청하는 것으로 간주하여 false 값을 갖게 한다.
**/
boolean searchOption = (type == null || type.equals(""))
    || keyword == null || keyword.equals("")) ? false : true;

// 검색 요청이 아니면
if(! searchOption) {
    // 전체 게시 글 수를 구한다.
    listCount = dao.getBoardCount();

    // 현재 페이지에 해당 하는 게시 글 리스트를 DB로부터 읽어온다.
    boardList = dao.boardList(startRow, endRow);

} else {
    // 검색어에 해당하는 게시 글 수를 구한다.
    listCount = dao.getBoardCount(type, keyword);

    // 검색 결과에 대한 현재 페이지에 해당 하는 게시 글 리스트를 DB로부터 읽어온다.
    boardList = dao.searchList(type, keyword, startRow, endRow);
}
System.out.println("listCount : " + listCount);

/* 페이지 그룹 이동 처리를 위해 전체 페이지를 계산하고 있다.
* [이전] 11 12 13... 또는 ... 8 9 10 [다음] 처리
* 전체 페이지 = 전체 게시 글 수 / 한 페이지에 표시할 게시 글 수가 되는데
* 이 계산식에서 나머지가 존재하면 전체 페이지 수 : 전체 페이지 + 1이 된다.
**/
int pageCount = listCount / PAGE_SIZE
    + (listCount % PAGE_SIZE == 0 ? 0 : 1);

```

```

/* 페이지 그룹 처리를 위해 페이지 그룹별 시작 페이지와 마지막 페이지를 계산
 * 페이지 그룹의 시작 페이지 : 1, 11, 21, 31...
 * 첫 번째 페이지 그룹에서 페이지 리스트는 1 ~ 10 되므로 currentPage가
 * 1 ~ 10 사이에 있으면 startPage는 1이 되고 11 ~ 20 사이는
 * 11이 된다.
 *
 * 정수형 연산의 특징을 이용해 startPage를 아래와 같이 구할 수 있다.
 * 아래 계산식으로 계산된 결과를 보면 현재 그룹의 마지막 페이지일 경우
 * startPage가 다음 그룹의 시작 페이지가 나오게 되므로 삼항 연산자를
 * 사용해 현재 페이지가 속한 그룹의 startPage가 되도록 조정 하였다.
 */
int startPage = (currentPage / PAGE_GROUP) * PAGE_GROUP + 1
    - (currentPage % PAGE_GROUP == 0 ? PAGE_GROUP : 0);

// 아래와 같이 startPage를 구할 수도 있다.
//int startPage = (currentPage / PAGE_GROUP) * PAGE_GROUP
//    + (currentPage % PAGE_GROUP == 0 ? -PAGE_GROUP + 1 : 1);

// 현재 페이지 그룹의 마지막 페이지 : 10, 20, 30...
int endPage = startPage + PAGE_GROUP - 1;

/* 위의 식에서 endPage를 구하게 되면 endPage는 항상 PAGE_GROUP의
 * 크기만큼 증가(10, 20, 30 ...) 되므로 맨 마지막 페이지 그룹의 endPage가
 * 정확하지 못할 경우가 발생하게 된다. 다시 말해 전체 페이지가 53페이지라고
 * 가정하면 위의 식에서 계산된 endPage는 60 페이지가 되지만 실제로
 * 60페이지는 존재하지 않는 페이지이므로 문제가 발생하게 된다.
 * 그래서 맨 마지막 페이지에 대한 보정이 필요하여 아래와 같이 endPage와
 * pageCount를 비교하여 현재 페이지 그룹에서 endPage가 pageCount 보다
 * 크다면 pageCount를 endPage로 지정 하였다. 즉 현재 페이지 그룹이
 * 마지막 페이지 그룹이면 endPage는 전체 페이지 수가 되도록 지정한 것이다.
 */
if(endPage > pageCount) {
    endPage = pageCount;
}

/* View 페이지에서 필요한 데이터를 Request 영역의 속성에 저장한다.
 * 현재 페이지, 전체 페이지 수, 페이지 그룹의 시작 페이지와 마지막 페이지
 * 게시 글의 수, 한 페이지에 보여 줄 게시 글 리스트의 데이터를 속성에 저장
 */
request.setAttribute("currentPage", currentPage);
request.setAttribute("pageCount", pageCount);
request.setAttribute("startPage", startPage);
request.setAttribute("endPage", endPage);
request.setAttribute("pageGroup", PAGE_GROUP);
request.setAttribute("listCount", listCount);

```



```

request.setAttribute("boardList", boardList);
request.setAttribute("searchOption", searchOption);

// 검색 요청이면 type과 keyword를 request 영역에 저장한다.
if(searchOption) {
    request.setAttribute("keyword", keyword);
    request.setAttribute("type", type);
}

/* 최종적으로 Redirect 정보와 View 페이지 정보를 문자열로 반환하면 된다.
 *
 * 게시 글 리스트 요청에 대한 결과(모델)를 request 영역의 속성에 저장하고
 * 요청에 대한 결과(모델)를 출력할 View 페이지와 View 페이지를 호출하는 방식을
 * 아래와 같이 문자열로 지정하면 된다. 현재 요청을 처리한 후에 Redirect 하려면
 * 뷰 페이지를 지정하는 문자열 맨 앞에 "r:" 또는 "redirect:"를 접두어로 붙여서
 * 반환하고 Redirect가 아니라 Forward 하려면 뷰 페이지의 경로만 지정하여
 * 문자열로 반환하면 Controller에서 판단하여 Redirect 또는 Forward로 연결된다.
 * 또한 Forward 할 때 뷰 페이지의 정보 중에서 앞부분과 뒷부분에서 중복되는
 * 정보를 줄이기 위해서 Controller에서 PREFIX와 SUFFIX를 지정해 사용하기
 * 때문에 매번 중복되는 부분을 제외하고 뷰 페이지의 정보를 지정하면 된다.
 */
return "board/boardList";
}
}

```

◆ 게시 글 리스트 요청 결과를 출력할 뷰

- webapp/WEB-INF/board/boardList.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<!DOCTYPE html>
<html lang="ko">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>게시 글 리스트</title>
    <link href="bootstrap/bootstrap.min.css" rel="stylesheet" >
    <script src="js/jquery-3.3.1.min.js"></script>
    <script src="js/formcheck.js"></script>
</head>
<body>
    <div class="container">
        <%@ include file="../pages/header.jsp" %>

```

```

<!-- content -->
<div class="row my-5" id="global-content">
  <div class="col">
    <div class="row text-center">
      <div class="col">
        <h2 class="fs-3 fw-bold">게시 글 리스트</h2>
      </div>
    </div>
  </div>
  <form name="searchForm" id="searchForm" action="#"
    class="row justify-content-center my-3">
    <div class="col-auto">
      <select name="type" class="form-select">
        <option value="title">제목</option>
        <option value="writer">작성자</option>
        <option value="content">내용</option>
      </select>
    </div>
    <div class="col-4">
      <input type="text" name="keyword" class="form-control"/>
    </div>
    <div class="col-auto">
      <input type="submit" value="검 색" class="btn btn-primary"/>
    </div>
  </form>

  <!-- 검색 요청일 경우 아래를 화면에 표시 -->
  <c:if test="${ searchOption }">
    <div class="row my-3">
      <div class="col text-center">
        "${ keyword }" 검색 결과
      </div>
    </div>

    <!-- 검색 요청일 경우 일반 게시 글 리스트로 이동할 수 있도록 링크를 설정했다.
    --%>
    <div class="row my-3">
      <div class="col-6">
        <a href="boardList.mvc" class="btn btn-outline-success">리스트</a>
      </div>
      <div class="col-6 text-end">
        <a href="writeForm.mvc" class="btn btn-outline-success">글쓰기</a>
      </div>
    </div>
  </c:if>

```

```

<!-- 검색 요청이 아닐 경우 아래를 화면에 표시 -->
<c:if test="${ not searchOption }">
    <div class="row my-3">
        <div class="col text-end">
            <a href="writeForm.mvc" class="btn btn-outline-success">글쓰기</a>
        </div>
    </div>
</c:if>

<div class="row my-3">
    <div class="col">
        <table class="table table-hover">
            <thead>
                <tr class="table-dark">
                    <th>NO</th>
                    <th>제목</th>
                    <th>작성자</th>
                    <th>작성일</th>
                    <th>조회수</th>
                </tr>
            </thead>
            <tbody class="text-secondary">
                <!--
                    검색 요청 이면서 검색된 리스트가 존재할 경우
                    게시 글 상세보기로 링크를 적용할 때 type과 keyword
                    파라미터를 적용해 링크를 설정한다.
                --%>
                <c:if test="${ searchOption and not empty boardList }">
                    <c:forEach var="b" items="${boardList}" varStatus="status">
                        <tr>
                            <td>${ b.no }</td>
                            <td>
                                <a
                                    href="boardDetail.mvc?no=${b.no}&pageNum=${currentPage}&type=${ type }&keyword=${
keyword }" class="text-decoration-none link-secondary">${ b.title }</a>
                            </td>
                            <td>${ b.writer }</td>
                            <td>${ b.regDate }</td>
                            <td>${ b.readCount }</td>
                        </tr>
                    </c:forEach>
                </c:if>

                <!--
                    일반 게시 글 리스트 요청 이면서 게시 글 리스트가 존재할 경우

```

게시 글 상세보기로 링크를 적용할 때 type과 keyword
파라미터는 필요 없다.

```
--%>
<c:if test="${ not searchOption and not empty boardList }">
  <c:forEach var="b" items="${boardList}" varStatus="status">
    <tr>
      <td>${ b.no }</td>
      <td><a href="boardDetail.mvc?no=${b.no}&pageNum=${currentPage}"
class="text-decoration-none link-secondary">${ b.title }</a></td>
      <td>${ b.writer }</td>
      <td>${ b.regDate }</td>
      <td>${ b.readCount }</td>
    </tr>
  </c:forEach>
</c:if>
<%-- 검색 요청이면서 검색된 리스트가 존재하지 않을 경우 --%>
<c:if test="${ searchOption and empty boardList }">
  <tr>
    <td colspan="5" class="text-center">
      "${ keyword }"가 포함된 게시 글이 존재하지 않습니다.
    </td>
  </tr>
</c:if>

<%-- 일반 게시 글 리스트 요청이면서 게시 글 리스트가 존재하지 않을 경우 --%>
<c:if test="${ not searchOption and empty boardList }">
  <tr>
    <td colspan="5" class="text-center">게시 글이 존재하지 않습니다.</td>
  </tr>
</c:if>
</tbody>
</table>
</div>
</div>

<!-- 검색 요청이면서 검색된 리스트가 존재할 경우 페이지네이션 -->
<c:if test="${ searchOption and not empty boardList }">
  <div class="row">
    <div class="col">
      <nav aria-label="Page navigation">
        <ul class="pagination justify-content-center">
          <%--
/* 현재 페이지 그룹의 시작 페이지가 pageGroup보다 크다는 것은
* 이전 페이지 그룹이 존재한다는 것으로 현재 페이지 그룹의 시작 페이지에
* pageGroup을 마이너스 하여 링크를 설정하면 이전 페이지 그룹의
```

```

* startPage로 이동할 수 있다.
**/
--%>
<c:if test="${ startPage > pageGroup }">
    <li class="page-item">
        <a class="page-link" href="boardList.mvc?pageNum=${ startPage -
pageGroup }&type=${ type }&keyword=${ keyword }">Pre</a>
    </li>
</c:if>

<%--
/* 현재 페이지 그룹의 startPage 부터 endPage 만큼 반복하면서
* 현재 페이지와 같은 그룹에 속한 페이지를 출력하고 링크를 설정한다.
* 현재 페이지는 링크를 설정하지 않는다.
**/
--%>
<c:forEach var="i" begin="${startPage}" end="${endPage}">
    <c:if test="{i == currentPage }">
        <li class="page-item active" aria-current="page">
            <span class="page-link">{i}</span>
        </li>
    </c:if>
    <c:if test="{i != currentPage }">
        <li class="page-item">
            <a class="page-link" href="boardList.mvc?pageNum={ i }&type=${
type }&keyword=${ keyword }">{i}</a>
        </li>
    </c:if>
</c:forEach>

<%--
/* 현재 페이지 그룹의 마지막 페이지가 전체 페이지 보다 작다는 것은
* 다음 페이지 그룹이 존재한다는 것으로 현재 페이지 그룹의 시작 페이지에
* pageGroup을 플러스 하여 링크를 설정하면 다음 페이지 그룹의
* startPage로 이동할 수 있다.
**/
--%>
<c:if test="{ endPage < pageCount }">
    <li class="page-item">
        <a class="page-link" href="boardList.mvc?pageNum=${ startPage +
pageGroup }&type=${ type }&keyword=${ keyword }">Next</a>
    </li>
</c:if>
</ul>
</nav>

```

```

        </div>
    </div>
</c:if>

<!-- 일반 게시글 요청이면서 검색된 리스트가 존재할 경우 페이지네이션 -->
<c:if test="{ not searchOption and not empty boardList }">
    <div class="row">
        <div class="col">
            <nav aria-label="Page navigation">
                <ul class="pagination justify-content-center">
                    <%--
/* 현재 페이지 그룹의 시작 페이지가 pageGroup보다 크다는 것은
* 이전 페이지 그룹이 존재한다는 것으로 현재 페이지 그룹의 시작 페이지에
* pageGroup을 마이너스 하여 링크를 설정하면 이전 페이지 그룹의
* startPage로 이동할 수 있다.
**/
--%>
                    <c:if test="{ startPage > pageGroup }">
                        <li class="page-item">
                            <a class="page-link" href="boardList.mvc?pageNum={ startPage -
pageGroup }">Pre</a>
                        </li>
                    </c:if>
                    <%--
/* 현재 페이지 그룹의 startPage 부터 endPage 만큼 반복하면서
* 현재 페이지와 같은 그룹에 속한 페이지를 출력하고 링크를 설정한다.
* 현재 페이지는 링크를 설정하지 않는다.
**/
--%>
                    <c:forEach var="i" begin="{startPage}" end="{endPage}">
                        <c:if test="{i == currentPage }">
                            <li class="page-item active" aria-current="page">
                                <span class="page-link">{i}</span>
                            </li>
                        </c:if>
                        <c:if test="{i != currentPage }">
                            <li class="page-item">
                                <a class="page-link" href="boardList.mvc?pageNum={ i }">{i}</a>
                            </li>
                        </c:if>
                    </c:forEach>

                    <%--
/* 현재 페이지 그룹의 마지막 페이지가 전체 페이지 보다 작다는 것은
* 다음 페이지 그룹이 존재한다는 것으로 현재 페이지 그룹의 시작 페이지에

```

```

        * pageGroup을 플러스 하여 링크를 설정하면 다음 페이지 그룹의
        * startPage로 이동할 수 있다.
    **/
    --%>
    <c:if test="${ endPage < pageCount }">
        <li class="page-item">
            <a class="page-link" href="boardList.mvc?pageNum=${ startPage +
pageGroup }">Next</a>
        </li>
    </c:if>
</ul>
</nav>
</div>
</div>
</c:if>
</div>
</div>
<%@ include file="../pages/footer.jsp" %>
</div>
<script src="bootstrap/bootstrap.bundle.min.js"></script>
</body>
</html>

```

1-1-7) 게시 글 상세보기

◆ 게시 글 상세보기 요청을 처리할 모델 클래스

- com.jspstudy.bbs.service.BoardDetailService

// 게시 글 상세보기 요청을 처리하는 서비스 클래스

```

public class BoardDetailService {

    public String requestProcess(
        HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        //요청 파라미터로 넘어 온 게시 글 번호와 페이지 번호를 읽어온다.
        String no = request.getParameter("no");
        String pageNum = request.getParameter("pageNum");
        String type = request.getParameter("type");
        String keyword = request.getParameter("keyword");

        // no와 pageNum이 비어 있으면 비정상 요청임
        if(no == null || no.equals("") || pageNum == null || pageNum.equals("")) {

            /* 스트림에 직접 쓰기위해 응답 객체로 부터 스트림을 구한다.
            * 응답 객체의 스트림을 구하기 전에 ContentType이 설정되어야 한다.

```

```

    * 그렇지 않으면 한글과 같은 데이터는 깨져서 출력된다.
    **/
response.setContentType("text/html; charset=utf-8");
PrintWriter out = response.getWriter();

out.println("<script>");
out.println(" alert('정상적인 접근이 아닙니다.');

```


- * 게시 글 상세보기 요청에 대한 결과(모델)를 request 영역의 속성에 저장하고
- * 요청에 대한 결과(모델)를 출력할 View 페이지와 View 페이지를 호출하는 방식을
- * 아래와 같이 문자열로 지정하면 된다. 현재 요청을 처리한 후에 Redirect 하려면
- * 뷰 페이지를 지정하는 문자열 맨 앞에 "r:" 또는 "redirect:"를 접두어로 붙여서
- * 반환하고 Redirect가 아니라 Forward 하려면 뷰 페이지의 경로만 지정하여
- * 문자열로 반환하면 Controller에서 판단하여 Redirect 또는 Forward로 연결된다.
- * 또한 Forward 할 때 뷰 페이지의 정보 중에서 앞부분과 뒷부분에서 중복되는
- * 정보를 줄이기 위해서 Controller에서 PREFIX와 SUFFIX를 지정해 사용하기
- * 때문에 매번 중복되는 부분을 제외하고 뷰 페이지의 정보를 지정하면 된다.

```

**/
return "board/boardDetail";
}
}

```

◆ 게시 글 상세보기 요청 결과를 출력할 뷰

- webapp/WEB-INF/board/boardDetail.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<!DOCTYPE html>
<html lang="ko">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>게시 글 상세보기</title>
<link href="bootstrap/bootstrap.min.css" rel="stylesheet" >
<script src="js/jquery-3.3.1.min.js"></script>
<script src="js/formcheck.js"></script>
</head>
<body>
<div class="container">
    <%@ include file="../pages/header.jsp" %>
    <!-- content -->
    <div class="row my-5" id="global-content">
        <div class="col">
            <form name="checkForm" id="checkForm">
                <input type="hidden" name="no" id="no" value="{ board.no }"/>
                <input type="hidden" name="pass" id="rPass" />
                <input type="hidden" name="pageNum" value="{ pageNum }" />

            <%--
                검색 리스트에서 들어온 요청일 경우 다시 keyword에 해당하는
                검색 리스트로 돌려보내기 위해서 아래의 파라미터가 필요하다.
            %--

```

[illegible]

```

        </td>
    </tr>
    <tr>
        <td colspan="4">
            <pre>${ board.content }</pre>
        </td>
    </tr>
</tbody>
</table>
</div>
</div>
<div class="row my-3">
    <div class="col text-center">
        <input class="btn btn-warning" type="button" id="detailUpdate" value="수정하기
"/>

        &nbsp;&nbsp;&nbsp;<input class="btn btn-danger" type="button" id="detailDelete"
value="삭제하기" />
        <!--
            일반 게시 글 리스트에서 온 요청이면 일반 게시 글 리스트로 돌려보낸다.
        --%>
        <c:if test="${ not searchOption }">
            &nbsp;&nbsp;&nbsp;<input class="btn btn-primary" type="button" value="목록보기"
                onclick="location.href='boardList.mvc?pageNum=${pageNum}'"/>
        </c:if>
        <!--
            검색 리스트에서 온 요청이면 검색 리스트의 동일한 페이지로 돌려보낸다.
        --%>
        <c:if test="${ searchOption }">
            &nbsp;&nbsp;&nbsp;<input class="btn btn-primary" type="button" value="목록보기"
                onclick="location.href='boardList.mvc?pageNum=${pageNum}&type=${ type
}&keyword=${ keyword }'"/>
        </c:if>
    </div>
</div>
</div>
</div>
<div>
    <%@ include file="../pages/footer.jsp" %>
</div>
<script src="bootstrap/bootstrap.bundle.min.js"></script>
</body>
</html>

```

1-1-8) 게시 글쓰기

◆ 게시 글쓰기 폼

- webapp/WEB-INF/board/writeForm.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html lang="ko">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title>게시 글 쓰기 폼</title>
        <link href="bootstrap/bootstrap.min.css" rel="stylesheet" >
        <script src="js/jquery-3.3.1.min.js"></script>
        <script src="js/formcheck.js"></script>
    </head>
    <body>
        <div class="container">
            <%@ include file="../pages/header.jsp" %>
            <!-- content -->
            <div class="row my-5" id="global-content">
                <div class="col">
                    <div class="row text-center">
                        <div class="col">
                            <h2 class="fs-3 fw-bold">게시 글쓰기</h2>
                        </div>
                    </div>
                    <form name="writeForm" action="writeProcess.mvc" id="writeForm"
                        class="row g-3 border-primary" method="post" enctype="multipart/form-data">
                        <div class="col-4 offset-md-2">
                            <label for="writer" class="form-label">글쓴이</label>
                            <input type="text" class="form-control" name="writer" id="writer"
                                placeholder="작성자를 입력해 주세요">
                        </div>
                        <div class="col-4 ">
                            <label for="pass" class="form-label">비밀번호</label>
                            <input type="password" class="form-control" name="pass" id="pass" >
                        </div>
                        <div class="col-8 offset-md-2">
                            <label for="title" class="form-label">제 목</label>
                            <input type="text" class="form-control" name="title" id="title" >
                        </div>
                        <div class="col-8 offset-md-2">
                            <label for="content" class="form-label">내 용</label>
                            <textarea class="form-control" name="content" id="content"
                                rows="10"></textarea>
                        </div>
                        <div class="col-8 offset-md-2">
```

```

        <label for="file1" class="form-label">파일</label>
        <input type="file" class="form-control" name="file1" id="file1" >
    </div>
    <div class="col-8 offset-md-2 text-center mt-5">
        <input type="submit" value="등록하기" class="btn btn-primary"/>
        &nbsp;&nbsp;&nbsp;<input type="button" value="목록보기"
            onclick="location.href='boardList.mvc'" class="btn btn-primary"/>
    </div>
</form>
</div>
</div>
<%@ include file="../pages/footer.jsp" %>
</div>
<script src="bootstrap/bootstrap.bundle.min.js"></script>
</body>
</html>

```

◆ 게시 글쓰기 요청을 처리할 모델 클래스

- com.jspstudy.bbs.service.BoardWriteService

// 게시 글쓰기 요청을 처리하는 서비스 클래스

```

public class BoardWriteService {

    public String requestProcess(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {

        /* cos 라이브러리를 이용한 파일 업로드 구현하기
        *
        * 1. MultipartRequest의 생성자 매개변수에 지정할 데이터를 설정
        *
        * ServletContext 객체의 속성에 저장된 파일을 업로드할
        * 디렉터리 정보를 읽어와 시스템의 로컬 경로를 구한다.
        */
        String uploadDir =
            (String) request.getServletContext().getAttribute("uploadDir");
        String realPath = request.getServletContext().getRealPath(uploadDir);

        // 업로드 파일의 최대 크기를 100MB로 지정
        int maxFileSize = 100 * 1024 * 1024;

        // 파일의 인코딩 타입을 UTF-8로 지정
        String encoding = "UTF-8";

        /* 2. 파일 업로드를 처리할 MultipartRequest 객체 생성
        *

```

- * WEB-INF/lib/cos.jar 파일을 살펴보면 MultipartRequest 클래스는
- * com.oreilly.servlet 패키지에 위치하며 파일 업로드를 직접적으로 처리하는
- * 역할을 담당하는 클래스로 파일 업로드와 관련된 다양한 메소드를 정의하고 있다.
- * 생성자는 5개로 오버로딩 되어 있고 아래 생성자가 호출되도록 정의되어 있다.

```

*
* public MultipartRequest(HttpServletRequest request,
*     String saveDirectory,
*     int maxPostSize,
*     String encoding,
*     FileRenamePolicy policy) throws IOException {...}
*

```

- * 이 생성자를 살펴보면 request, saveDirectory, maxPostSize는 필수사항으로
- * 이 매개변수가 null이거나 0보다 작다면 생성자 안에서 예외를 발생시킨다.

- * request : MultipartRequest에 연결할 사용자의 요청 정보가 담긴 객체
- * saveDirectory : 업로드 된 파일을 저장할 서버의 디렉터리 지정
- * maxPostSize : 업로드 파일의 최대 크기 지정
- * encoding : 파일의 인코딩 방식 지정, 파일 이름이 한글일 경우 필히 utf-8 지정
- * policy : 사용자가 업로드 한 파일을 저장할 서버의 디렉터리에 현재 업로드 되는
- * 파일과 이름이 중복된 파일이 존재할 경우 현재 업로드 되는 파일의
- * 이름을 어떻게 변경할지에 대한 정책을 지정하는 매개변수 이다.
- * 일반적으로 new DefaultFileRenamePolicy()를 사용하며
- * 이 클래스는 abc.jpg 파일을 업로드 할때 이미 같은 이름의 파일이
- * 존재하면 자동으로 abc1.jpg와 같이 파일을 변경해 준다.

- * 아래와 같이 MultipartRequest 객체를 생성하면 saveDirectory에 지정한
- * 서버의 디렉터리로 파일이 바로 업로드 된다.

*/

```

MultipartRequest multi = new MultipartRequest(request, realPath,
    maxFileSize, encoding, new DefaultFileRenamePolicy());

```

/* 3. MultipartRequest 객체를 이용해 클라이언트로부터 요청된 데이터를 처리

- * 파일 업로드 처리를 위해서는 모든 요청에 대한 처리를 MultipartRequest 객체를
- * 이용해 접근해야 한다. 위에서 MultipartRequest 객체를 생성할 때 요청에 대한
- * 정보를 담고 있는 request를 생성자의 매개변수로 지정해 MultipartRequest를
- * 통해 사용자의 요청 정보에 접근할 수 있다.

- * MultipartRequest 클래스에 정의된 주요 메소드는 아래와 같다.
- * getParameter(name) : name에 지정한 파라미터 값을 반환
- * getParameterNames() : 폼에서 전송된 모든 파라미터 이름을
- * Enumeration 타입으로 반환
- * getParameterValues(name) : name에 지정한 파라미터 값을 String 배열로 반환
- * getFile(fileName) : 업로드 된 파일 중에서 fileName에 지정한 파라미터
- * 이름을 가진 파일의 정보를 File 객체로 반환

```

* getFileNames() : 폼에서 전송된 모든 파일의 이름을 Enumeration 타입으로 반환
* getFileSystemName(name) : name에 지정한 파라미터 이름을 가진
*
* 파일의 이름을 반환
* getOriginalFileName() : 사용자가 업로드 한 파일의 원본 이름을 반환
* getContentType() : 사용자가 업로드 한 파일의 콘텐츠 타입을 반환
**/

/* 사용자가 폼에 입력한 데이터 처리
* MultipartRequest 객체를 통해 파라미터를 읽어 변수에 저장한다.
**/
String title = multi.getParameter("title");
String writer = multi.getParameter("writer");
String pass = multi.getParameter("pass");
String content = multi.getParameter("content");

/* 하나의 게시 글 정보를 저장하는 자바빈 객체를 생성하고 요청 데이터를
* Board 객체에 저장한다.
**/
Board board = new Board();
board.setTitle(title);
board.setWriter(writer);
board.setPass(pass);
board.setContent(content);

/* 사용자가 업로드한 파일 데이터 처리
* MultipartRequest 객체를 통해 파일 이름을 구하여 변수에 저장한다.
* 파일이 업로드 되지 않으면 fileName은 null 값을 받는다.
**/
String fileName = multi.getFileSystemName("file1");
System.out.println("업로드 된 파일명 : " + fileName);
System.out.println("원본 파일명 : " + multi.getOriginalFileName("file1"));

// 파일명이 존재하면 파일명을 지정하고 존재하지 않으면 null로 지정 한다.
board.setFile1(fileName != null ? fileName : null);

if(board.getFile1() == null) {
    System.out.println("파일이 업로드 되지 않았음");
}

/* BoardDao 객체를 얻어 게시 글을 DB에 추가하기
* DB에 게시 글을 추가하고 브라우저에게 게시 글 리스트를 요청하라고 응답
* 게시 글쓰기가 완료된 후 Redirect 시키지 않으면 이 페이지를 새로 고침 하여
* 재요청 할 때 마다 이미 추가된 게시 글을 계속하여 추가하는 문제가 발생 한다.
**/
BoardDao dao = new BoardDao();

```

```
dao.insertBoard(board);
```

```
/* 최종적으로 Redirect 정보와 View 페이지 정보를 문자열로 반환하면 된다.
```

```
*
```

- * 게시 글쓰기 요청을 처리하고 Redirect 시키지 않으면 사용자가 브라우저를
- * 새로 고침 하거나 재요청할 때 마다 이미 DB에 추가된 게시 글을 계속 추가하려는
- * 동작으로 인해서 중복된 데이터가 저장되거나 또 다른 문제가 발생할 수 있다.
- * 이런 경우에는 Redirect 기법을 이용해 DB에 추가, 수정, 삭제하는 동작이 아닌
- * 조회하는 곳으로 이동하도록 하면 문제를 해결 할 수 있다.

```
*
```

- * 현재 요청을 처리한 후에 Redirect 하려면 뷰 페이지를 지정하는 문자열 맨 앞에
- * "r:" 또는 "redirect:"를 접두어로 붙여서 반환하고 Redirect가 아니라 Forward
- * 하려면 뷰 페이지의 경로만 지정하여 문자열로 반환하면 Controller에서 판단하여
- * Redirect 또는 Forward로 연결된다.

```
*
```

- * 게시 글쓰기 폼으로 부터 넘어온 신규 게시 글을 DB에 저장한 후 게시 글 리스트
- * 페이지로 이동시키기 위해 View 페이지 정보를 반환할 때 맨 앞에 "r:" 접두어를
- * 붙여서 게시 글 리스트 보기 요청을 처리하는 URL을 지정하여 Controller로 넘기면
- * Controller는 넘겨받은 View 페이지 정보를 분석하여 Redirect 시키게 된다.

```
*
```

- * Redirect는 클라이언트 요청에 대한 결과 페이지가 다른 곳으로 이동되었다고
- * 브라우저에게 알려주고 그 이동된 주소로 다시 요청하라고 브라우저에게 URL을
- * 보내서 브라우저가 그 URL로 다시 응답하도록 처리하는 것으로 아래와 같이
- * View 페이지 정보의 맨 앞에 "r:" 또는 "redirect:"를 접두어로 붙여서 반환하면
- * Controller에서 View 페이지 정보를 분석해 Redirect 시키고 이 응답을 받은
- * 브라우저는 게시 글 리스트를 보여주는 페이지를 다시 요청하게 된다.

```
**/
```

```
return "r:boardList.mvc";
```

```
}
```

```
}
```

1-1-9) 게시 글 수정하기

◆ 게시 글 수정 폼 요청을 처리할 모델 클래스

- com.jspstudy.bbs.service.UpdateFormService

// 게시 글 수정 폼 요청을 처리하는 모델 클래스

```
public class UpdateFormService {
```

```
    public String requestProcess(
```

```
        HttpServletRequest request, HttpServletResponse response)
```

```
        throws ServletException, IOException {
```

```
        String sNo = request.getParameter("no");
```

```
        String pass = request.getParameter("pass");
```

```
        String pageNum = request.getParameter("pageNum");
```



```

String type = request.getParameter("type");
String keyword = request.getParameter("keyword");

/* 글 번호와 페이지 번호가 파라미터로 전달되지 않는 요청은 정상적인 접근이
 * 아니므로 자바스크립트를 사용해 경고 창을 띄우고 브라우저에 저장된
 * 이전 페이지로 돌려보낸다. Controller로 viewPage 정보를 반환해야
 * 하지만 이 경우 viewPage 정보가 없으므로 PrintWriter 객체를
 * 이용해 클라이언트로 응답할 자바스크립트 코드를 출력하고 null을
 * 반환하면 Controller에서는 viewPage 정보가 null이 아닐 경우만
 * 처리하게 되므로 자바스크립트가 브라우저로 전송되어 경고 창이 뜨게 된다.
 */
if(sNo == null || sNo.equals("") || pass == null || pass.equals("")
    || pageNum == null || pageNum.equals("")) {

    /* 스트림에 직접 쓰기위해 응답 객체로 부터 스트림을 구한다.
     * 응답 객체의 스트림을 구하기 전제 ContentType이 설정되어야 한다.
     * 그렇지 않으면 한글과 같은 데이터는 깨져서 출력된다.
     */
    response.setContentType("text/html; charset=utf-8");
    PrintWriter out = response.getWriter();
    out.println("<script>");
    out.println(" alert('정상적인 접근이 아닙니다.');

```

```

/* 요청 파라미터에서 type이나 keyword가 비어 있으면 일반
 * 게시 글 리스트에서 넘어온 요청으로 간주하여 false 값을 갖게 한다.
 * 이 정보는 게시 글 리스트와 검색 리스트로 구분해 돌려보내기 위해 필요하다.
 */
boolean searchOption = (type == null || type.equals(""))
    || keyword == null || keyword.equals("")) ? false : true;

/* BoardDAO 객체를 얻어 수정할 게시글 하나를 읽어와 Board
 * 타입의 변수에 저장한다. BoardDAO 클래스의 getBoard()는 첫 번째
 * 매개변수에 지정한 글 번호에 해당하는 게시글 하나를 반환하는 메소드로
 * 두 번째 인자에 false를 지정하면 게시 글 읽은 횟수는 증가되지 않는다.
 */
Board board = dao.getBoard(Integer.valueOf(no), false);

// 게시글 정보와 페이지 정보를 Request 속성 영역에 저장 한다.
request.setAttribute("board", board);
request.setAttribute("pageNum", pageNum);
request.setAttribute("searchOption", searchOption);

// 검색 요청이면 type과 keyword를 request 영역에 저장한다.
if(searchOption) {
    request.setAttribute("type", type);
    request.setAttribute("keyword", keyword);
}

/* 최종적으로 Redirect 정보와 View 페이지 정보를 문자열로 반환하면 된다.
 *
 * 게시 글 수정 폼 요청에 대한 결과(모델)를 request 영역의 속성에 저장하고
 * 요청에 대한 결과(모델)를 출력할 View 페이지와 View 페이지를 호출하는 방식을
 * 아래와 같이 문자열로 지정하면 된다. 현재 요청을 처리한 후에 Redirect 하려면
 * 뷰 페이지를 지정하는 문자열 맨 앞에 "r:" 또는 "redirect:"를 접두어로 붙여서
 * 반환하고 Redirect가 아니라 Forward 하려면 뷰 페이지의 경로만 지정하여
 * 문자열로 반환하면 Controller에서 판단하여 Redirect 또는 Forward로 연결된다.
 * 또한 Forward 할 때 뷰 페이지의 정보 중에서 앞부분과 뒷부분에서 중복되는
 * 정보를 줄이기 위해서 Controller에서 PREFIX와 SUFFIX를 지정해 사용하기
 * 때문에 매번 중복되는 부분을 제외하고 뷰 페이지의 정보를 지정하면 된다.
 */
return "board/updateForm";
}
}

```

◆ 게시 글 수정 폼

- webapp/WEB-INF/board/updateForm.jsp

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

```

    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html lang="ko">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>게시 글 수정 폼</title>
    <link href="bootstrap/bootstrap.min.css" rel="stylesheet" >
    <script src="js/jquery-3.3.1.min.js"></script>
    <script src="js/formcheck.js"></script>
  </head>
  <body>
    <div class="container">
      <%@ include file="../pages/header.jsp" %>
      <!-- content -->
      <div class="row my-5" id="global-content">
        <div class="col">
          <div class="row text-center">
            <div class="col">
              <h2 class="fs-3 fw-bold">게시 글 수정하기</h2>
            </div>
          </div>
          <form name="updateForm" action="updateProcess.mvc" id="updateForm"
            class="row g-3 border-primary" method="post"
            ${ not empty board.file1 ? "" : "enctype='multipart/form-data'"}>
            <!--
              no는 DB에서 게시 글을 수정하기 위해 필요하고 pageNum은 게시 글이
              수정된 후에 이전에 사용자가 머물렀던 게시 글 리스트의 동일한 페이지로
              보내기 위해 필요한 정보이다.
            --%>
            <input type="hidden" name="no" value="${board.no}">
            <input type="hidden" name="pageNum" value="${ pageNum }" />

            <!--
              검색 요청일 경우 다시 keyword에 해당하는 검색 리스트로
              돌려보내기 위해서 아래의 파라미터가 필요하다.
            --%>
            <c:if test="${ searchOption }">
              <input type="hidden" name="type" value="${ type }" />
              <input type="hidden" name="keyword" value="${ keyword }" />
            </c:if>

            <div class="col-4 offset-md-2">
              <label for="writer" class="form-label">글쓴이</label>

```

```

        <input type="text" class="form-control" name="writer" id="writer"
            placeholder="작성자를 입력해 주세요" value="{ board.writer }">
    </div>
    <div class="col-4 ">
        <label for="pass" class="form-label">비밀번호</label>
        <input type="password" class="form-control" name="pass"
            id="pass">
    </div>
    <div class="col-8 offset-md-2">
        <label for="title" class="form-label">제 목</label>
        <input type="text" class="form-control" name="title"
            id="title" value="{board.title}">
    </div>
    <div class="col-8 offset-md-2">
        <label for="content" class="form-label">내 용</label>
        <textarea class="form-control" name="content" id="content"
            rows="10">{ board.content }</textarea>
    </div>
    <div class="col-8 offset-md-2">
        <label for="file1" class="form-label">파 일</label>
        <input type="file" class="form-control" name="file1" id="file1"
            { empty board.file1 ? "" : "disabled" }>
    </div>
    <div class="col-8 offset-md-2 text-center mt-5">
        <input type="submit" value="수정하기" class="btn btn-primary"/>
        <!--
        일반 게시 글 리스트에서 온 요청이면 일반 게시 글 리스트로 돌려보낸다.
        --%>
        <c:if test="{ not searchOption }">
            &nbsp;&nbsp;&nbsp;<input class="btn btn-primary" type="button" value="목록보기"
                onclick="location.href='boardList.mvc?pageNum={pageNum}'"/>
        </c:if>
        <!--
        검색 리스트에서 온 요청이면 검색 리스트의 동일한 페이지로 돌려보낸다.
        --%>
        <c:if test="{ searchOption }">
            &nbsp;&nbsp;&nbsp;<input class="btn btn-primary" type="button" value="목록보기"
                onclick="location.href='boardList.mvc?pageNum={pageNum}&type={ type
}&keyword={ keyword }'"/>
        </c:if>
    </div>
</form>
</div>
</div>
<%@ include file="../pages/footer.jsp" %>

```

```

</div>
<script src="bootstrap/bootstrap.bundle.min.js"></script>
</body>
</html>

```

◆ 게시 글 수정 요청을 처리할 모델 클래스

- com.jspstudy.bbs.service.UpdateService

// 게시 글 수정 폼에서 요청한 데이터를 받아 DB에 수정하는 모델 클래스

```

public class UpdateService {

    public String requestProcess(
        HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        String contentType = request.getHeader("Content-Type");
        System.out.println("contentType : " + contentType);
        BoardDao dao = new BoardDao();
        Board board = null;
        String pass= null, title = null, writer = null, content = null,
            sNo = null, pageNum = null, type=null, keyword=null;
        int no = 0;

        /* 비밀번호가 맞지 않을 경우 먼저 처리해야 하나 Content-Type에
        * 따라서 파라미터를 읽을 수 있는 객체가 다르기 때문에 각각 처리해야 한다.
        * Content-Type이 application/x-www-form-urlencoded 일 경우
        * request.getParameter() 메서드로 파라미터를 읽을 수 있지만
        * multipart/form-data 일 경우에는 request 객체를 이용해
        * 파라미터를 읽을 수 없기 때문에 각각 처리하는 코드를 기술해야 한다.
        */
        // 요청이 multipart/form-data 일 경우
        if(contentType.contains("multipart/form-data")) {

            /* cos 라이브러리를 이용한 파일 업로드 구현하기
            *
            * 1. MultipartRequest의 생성자 매개변수에 지정할 데이터를 설정
            *
            * ServletContext 객체의 속성에 저장된 파일을 업로드할
            * 디렉터리 정보를 읽어와 시스템의 로컬 경로를 구한다.
            */
            String uploadDir =
                (String) request.getServletContext().getAttribute("uploadDir");
            String realPath = request.getServletContext().getRealPath(uploadDir);

```

```
// 업로드 파일의 최대 크기를 100MB로 지정
int maxFileSize = 100 * 1024 * 1024;

// 파일의 인코딩 타입을 UTF-8로 지정
String encoding = "UTF-8";

/* 2. 파일 업로드를 처리할 MultipartRequest 객체 생성
 *
 * WEB-INF/lib/cos.jar 파일을 살펴보면 MultipartRequest 클래스는
 * com.oreilly.servlet 패키지에 위치하며 파일 업로드를 직접적으로 처리하는
 * 역할을 담당하는 클래스로 파일 업로드와 관련된 다양한 메소드를 정의하고 있다.
 * 생성자는 5개로 오버로딩 되어 있고 아래 생성자가 호출되도록 정의되어 있다.
 *
 * public MultipartRequest(HttpServletRequest request,
 *     String saveDirectory,
 *     int maxPostSize,
 *     String encoding,
 *     FileRenamePolicy policy) throws IOException {...}
 *
 * 이 생성자를 살펴보면 request, saveDirectory, maxPostSize는 필수사항으로
 * 이 매개변수가 null이거나 0보다 작다면 생성자 안에서 예외를 발생시킨다.
 *
 * request : MultipartRequest에 연결할 사용자의 요청 정보가 담긴 객체
 * saveDirectory : 업로드 된 파일을 저장할 서버의 디렉터리 지정
 * maxPostSize : 업로드 파일의 최대 크기 지정
 * encoding : 파일의 인코딩 방식 지정, 파일 이름이 한글일 경우 필히 utf-8 지정
 * policy : 사용자가 업로드 한 파일을 저장할 서버의 디렉터리에 현재 업로드 되는
 *     파일과 이름이 중복된 파일이 존재할 경우 현재 업로드 되는 파일의
 *     이름을 어떻게 변경할지에 대한 정책을 지정하는 매개변수 이다.
 *     일반적으로 new DefaultFileRenamePolicy()를 사용하며
 *     이 클래스는 abc.jpg 파일을 업로드 할때 이미 같은 이름의 파일이
 *     존재하면 자동으로 abc1.jpg와 같이 파일을 변경해 준다.
 *
 * 아래와 같이 MultipartRequest 객체를 생성하면 saveDirectory에 지정한
 * 서버의 디렉터리로 파일이 바로 업로드 된다.
 */
MultipartRequest multi = new MultipartRequest(request, realPath,
    maxFileSize, encoding, new DefaultFileRenamePolicy());

/* 3. MultipartRequest 객체를 이용해 클라이언트로부터 요청된 데이터를 처리
 *
 * 파일 업로드 처리를 위해서는 모든 요청에 대한 처리를 MultipartRequest 객체를
 * 이용해 접근해야 한다. 위에서 MultipartRequest 객체를 생성할 때 요청에 대한
 * 정보를 담고 있는 request를 생성자의 매개변수로 지정해 MultipartRequest를
 * 통해 사용자의 요청 정보에 접근할 수 있다.

```

```

*
* MultipartRequest 클래스에 정의된 주요 메소드는 아래와 같다.
* getParameter(name) : name에 지정한 파라미터 값을 반환
* getParameterNames() : 폼에서 전송된 모든 파라미터 이름을
*                               Enumeration 타입으로 반환
* getParameterValues(name) : name에 지정한 파라미터 값을 String 배열로 반환
* getFile(fileName) : 업로드 된 파일 중에서 fileName에 지정한 파라미터
*                               이름을 가진 파일의 정보를 File 객체로 반환
* getFileNames() : 폼에서 전송된 모든 파일의 이름을 Enumeration 타입으로 반환
* getFileSystemName(name) : name에 지정한 파라미터 이름을 가진
*                               파일의 이름을 반환
* getOriginalFileName() : 사용자가 업로드 한 파일의 원본 이름을 반환
* getContentType() : 사용자가 업로드 한 파일의 콘텐츠 타입을 반환
**/

/* 사용자가 폼에 입력한 데이터 처리
* MultipartRequest 객체를 통해 파라미터를 읽어 변수에 저장한다.
**/
sNo = multi.getParameter("no");
pass = multi.getParameter("pass");
pageNum = multi.getParameter("pageNum");
type = multi.getParameter("type");
keyword = multi.getParameter("keyword");

/* 글 번호와 페이지 번호가 파라미터로 전달되지 않는 요청은 정상적인 접근이
* 아니므로 자바스크립트를 사용해 경고 창을 띄우고 브라우저에 저장된
* 이전 페이지로 돌려보낸다. Controller로 viewPage 정보를 반환해야
* 하지만 이 경우 viewPage 정보가 없으므로 PrintWriter 객체를
* 이용해 클라이언트로 응답할 자바스크립트 코드를 출력하고 null을
* 반환하면 Controller에서는 viewPage 정보가 null이 아닐 경우만
* 처리하게 되므로 자바스크립트가 브라우저로 전송되어 경고 창이 뜨게 된다.
**/
if(sNo == null || sNo.equals("") || pass == null || pass.equals("")
    || pageNum == null || pageNum.equals("")) {

    /* 스트림에 직접 쓰기위해 응답 객체로 부터 스트림을 구한다.
    * 응답 객체의 스트림을 구하기 전제 ContentType이 설정되어야 한다.
    * 그렇지 않으면 한글과 같은 데이터는 깨져서 출력된다.
    **/
    response.setContentType("text/html; charset=utf-8");
    PrintWriter out = response.getWriter();
    out.println("<script>");
    out.println("alert('정상적인 접근이 아닙니다.');");
    out.println("history.back();");
    out.println("</script>");

```

```

        return null;
    }

    no = Integer.parseInt(sNo);

    // 게시 글의 비밀번호를 체크해 맞지 않으면 이전으로 돌려보낸다.
    boolean isPassCheck = dao.isPassCheck(no, pass);
    if(! isPassCheck) {
        System.out.println("비밀번호 맞지 않음");

        response.setContentType("text/html; charset=utf-8");
        PrintWriter out = response.getWriter();
        StringBuilder sb = new StringBuilder();
        sb.append("<script>");
        sb.append(" alert('비밀번호가 맞지 않습니다.');

```



```

if(board.getFile1() == null) {
    System.out.println("파일이 업로드 되지 않았음");
}

// 요청이 multipart/form-data 아닌 경우
} else {

    request.setCharacterEncoding("utf-8");
    sNo = request.getParameter("no");
    pass = request.getParameter("pass");
    pageNum = request.getParameter("pageNum");
    type = request.getParameter("type");
    keyword = request.getParameter("keyword");

    /* 글 번호와 페이지 번호가 파라미터로 전달되지 않는 요청은 정상적인 접근이
     * 아니므로 자바스크립트를 사용해 경고 창을 띄우고 브라우저에 저장된
     * 이전 페이지로 돌려보낸다. Controller로 viewPage 정보를 반환해야
     * 하지만 이 경우 viewPage 정보가 없으므로 PrintWriter 객체를
     * 이용해 클라이언트로 응답할 자바스크립트 코드를 출력하고 null을
     * 반환하면 Controller에서는 viewPage 정보가 null이 아닐 경우만
     * 처리하게 되므로 자바스크립트가 브라우저로 전송되어 경고 창이 뜨게 된다.
     */
    if(sNo == null || sNo.equals("") || pass == null || pass.equals("")
        || pageNum == null || pageNum.equals("")) {

        /* 스트림에 직접 쓰기위해 응답 객체로 부터 스트림을 구한다.
         * 응답 객체의 스트림을 구하기 전해 ContentType이 설정되어야 한다.
         * 그렇지 않으면 한글과 같은 데이터는 깨져서 출력된다.
         */
        response.setContentType("text/html; charset=utf-8");
        PrintWriter out = response.getWriter();
        out.println("<script>");
        out.println("alert('정상적인 접근이 아닙니다.');

```

```

        response.setContentType("text/html; charset=utf-8");
        PrintWriter out = response.getWriter();
        StringBuilder sb = new StringBuilder();
        sb.append("<script>");
        sb.append(" alert('비밀번호가 맞지 않습니다.');"");
        sb.append(" history.back();"");
        sb.append("</script>");
        out.println(sb.toString());
        return null;
    }

    title = request.getParameter("title");
    writer = request.getParameter("writer");
    content = request.getParameter("content");

    /* 하나의 게시 글 정보를 저장하는 자바빈 객체를 생성하고 파라미터로
     * 넘겨받은 요청 데이터를 Board 객체에 저장한다.
     */
    board = new Board();
    board.setNo(no);
    board.setTitle(title);
    board.setWriter(writer);
    board.setPass(pass);
    board.setContent(content);
}

/* 수정 요청한 게시 글의 내용을 저장하고 있는 Board 인스턴스를
 * 인수로 BoardDAO 클래스의 updateBoard()를 호출해
 * 사용자가 수정한 게시 글을 DB에 업데이트 한다.
 */
dao.updateBoard(board);

/* 요청 파라미터에서 type이나 keyword가 비어 있으면 일반
 * 게시 글 리스트에서 넘어온 요청으로 간주하여 false 값을 갖게 한다.
 * 이 정보는 게시 글 리스트와 검색 리스트로 구분해 돌려보내기 위해 필요하다.
 */
boolean searchOption = (type == null || type.equals(""))
    || keyword == null || keyword.equals("")) ? false : true;

/* 리다이렉트 할 때 게시 글 리스트의 페이지 번호를 파라미터로 넘겨 사용자가
 * 게시 글 수정을 요청한 페이지와 동일한 페이지로 리다이렉트 시킨다.
 */
String url = "boardList.mvc?pageNum=" + pageNum;

/* 검색 리스트 상태에서 게시 글 상세보기로 들어와 게시 글을 수정하는 것이라면

```

```

* 검색 옵션에 해당하는 검색한 결과에 대한 게시 글 리스트 페이지로 Redirect
* 시켜야 하므로 type과 keyword를 Redirect 주소에 추가한다.
* Redirect 기법은 요청한 결과가 이동했다고 브라우저에게 이동할 주소를 응답하는
* 것으로 브라우저는 주소 표시줄에 주소를 입력해 요청하게 되므로 GET 방식 요청이다.
**/
if(searchOption) {

    /* 리다이렉트 할 때 파라미터에 한글이 포함되어 있으면 한글로 된 파라미터 값은
    * 공백문자로 변경되어 리다이렉트 되기 때문에 한글 데이터는 깨지게 된다.
    * 이런 경우에는 java.net 패키지의 URLEncoder 클래스를 이용해 아래와
    * 같이 수동으로 URL 인코딩을 하면 이 문제를 해결할 수 있다.
    */
    keyword = URLEncoder.encode(keyword, "utf-8");
    url += "&type=" + type + "&keyword=" + keyword;
}
System.out.println("keyword : " + keyword);
System.out.println("url : " + url);

/* 최종적으로 Redirect 정보와 View 페이지 정보를 문자열로 반환하면 된다.
*
* 게시 글 수정하기 요청을 처리하고 Redirect 시키지 않으면 사용자가 브라우저를
* 새로 고침 하거나 재요청할 때 마다 이미 DB에서 수정된 게시 글을 계속 수정하려는
* 동작으로 인해서 문제가 발생할 수 있다. 이런 경우에는 Redirect 기법을 이용해 DB에
* 추가, 수정, 삭제가 아닌 조회하는 곳으로 이동하도록 하면 문제를 해결 할 수 있다.
*
* 현재 요청을 처리한 후에 Redirect 하려면 뷰 페이지를 지정하는 문자열 맨 앞에
* "r:" 또는 "redirect:"를 접두어로 붙여서 반환하고 Redirect가 아니라 Forward
* 하려면 뷰 페이지의 경로만 지정하여 문자열로 반환하면 Controller에서 판단하여
* Redirect 또는 Forward로 연결된다.
*
* 게시 글 수정 폼으로 부터 넘어온 게시 글을 DB에서 수정한 후 게시 글 리스트
* 페이지로 이동시키기 위해 View 페이지 정보를 반환할 때 맨 앞에 "r:" 접두어를
* 붙여서 게시 글 리스트 보기 요청을 처리하는 URL를 지정하여 Controller로 넘기면
* Controller는 넘겨받은 View 페이지 정보를 분석하여 Redirect 시키게 된다.
*
* Redirect는 클라이언트 요청에 대한 결과 페이지가 다른 곳으로 이동되었다고
* 브라우저에게 알려주고 그 이동된 주소로 다시 요청하라고 브라우저에게 URL을
* 보내서 브라우저가 그 URL로 다시 응답하도록 처리하는 것으로 아래와 같이
* View 페이지 정보의 맨 앞에 "r:" 또는 "redirect:"를 접두어로 붙여서 반환하면
* Controller에서 View 페이지 정보를 분석해 Redirect 시키고 이 응답을 받은
* 브라우저는 게시 글 리스트를 보여주는 페이지를 다시 요청하게 된다.
**/
return "r:" + url;
}
}

```

1-1-10) 게시 글 삭제하기

◆ 게시 글 삭제 요청을 처리할 모델 클래스

- com.jspstudy.bbs.service.DeleteService

// 게시 글 삭제 요청을 받아 DB에서 게시 글을 삭제하는 모델 클래스

```
public class DeleteService {
    public String requestProcess(
        HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        String sNo = request.getParameter("no");
        String pass = request.getParameter("pass");
        String pageNum = request.getParameter("pageNum");
        String type = request.getParameter("type");
        String keyword = request.getParameter("keyword");

        /* 글 번호와 페이지 번호가 파라미터로 전달되지 않는 요청은 정상적인 접근이
        * 아니므로 자바스크립트를 사용해 경고 창을 띄우고 브라우저에 저장된
        * 이전 페이지로 돌려보낸다. Controller로 viewPage 정보를 반환해야
        * 하지만 이 경우 viewPage 정보가 없으므로 PrintWriter 객체를
        * 이용해 클라이언트로 응답할 자바스크립트 코드를 출력하고 null을
        * 반환하면 Controller에서는 viewPage 정보가 null이 아닐 경우만
        * 처리하게 되므로 자바스크립트가 브라우저로 전송되어 경고 창이 뜨게 된다.
        */
        if(sNo == null || sNo.equals("") || pass == null || pass.equals("")
            || pageNum == null || pageNum.equals("")) {

            /* 스트림에 직접 쓰기위해 응답 객체로 부터 스트림을 구한다.
            * 응답 객체의 스트림을 구하기 전제 ContentType이 설정되어야 한다.
            * 그렇지 않으면 한글과 같은 데이터는 깨져서 출력된다.
            */
            response.setContentType("text/html; charset=utf-8");
            PrintWriter out = response.getWriter();
            out.println("<script>");
            out.println(" alert('정상적인 접근이 아닙니다.');
```

```

if(! isPassCheck) {
    /* 스트림에 직접 쓰기위해 응답 객체로 부터 스트림을 구한다.
     * 응답 객체의 스트림을 구하기 전해 ContentType이 설정되어야 한다.
     * 그렇지 않으면 한글과 같은 데이터는 깨져서 출력된다.
     */
    response.setContentType("text/html; charset=utf-8");
    PrintWriter out = response.getWriter();
    out.println("<script>");
    out.println(" alert('비밀번호가 맞지 않습니다.');");
    out.println(" history.back();");
    out.println("</script>");
    return null;
}

// BoardDao 객체를 이용해 게시 글을 삭제한다.
dao.deleteBoard(no);

/* 요청 파라미터에서 type이나 keyword가 비어 있으면 일반
 * 게시 글 리스트에서 넘어온 요청으로 간주하여 false 값을 갖게 한다.
 * 이 정보는 게시 글 리스트와 검색 리스트로 구분해 돌려보내기 위해 필요하다.
 */
boolean searchOption = (type == null || type.equals(""))
    || keyword == null || keyword.equals("")) ? false : true;

/* 리다이렉트 할 때 게시 글 리스트의 페이지 번호를 파라미터로 넘겨 사용자가
 * 게시 글 수정을 요청한 페이지와 동일한 페이지로 리다이렉트 시킨다.
 */
String url = "boardList.mvc?pageNum=" + pageNum;

/* 검색 리스트 상태에서 게시 글 상세보기로 들어와 게시 글을 삭제하는 것이라면
 * 검색 옵션에 해당하는 검색한 결과에 대한 게시 글 리스트 페이지로 Redirect
 * 시켜야 하므로 type과 keyword를 Redirect 주소에 추가한다.
 * Redirect 기법은 요청한 결과가 이동했다고 브라우저에게 이동할 주소를 응답하는
 * 것으로 브라우저는 주소 표시줄에 주소를 입력해 요청하게 되므로 GET 방식 요청이다.
 */
if(searchOption) {

    /* 리다이렉트 할 때 파라미터에 한글이 포함되어 있으면 한글로 된 파라미터 값은
     * 공백문자로 변경되어 리다이렉트 되기 때문에 한글 데이터는 깨지게 된다.
     * 이런 경우에는 java.net 패키지의 URLEncoder 클래스를 이용해 아래와
     * 같이 수동으로 URL 인코딩을 하면 이 문제를 해결할 수 있다.
     */
    keyword = URLEncoder.encode(keyword, "utf-8");
    url += "&type=" + type + "&keyword=" + keyword;
}

```

```

/* 최종적으로 Redirect 정보와 View 페이지 정보를 문자열로 반환하면 된다.
*
* 게시 글 삭제하기 요청을 처리하고 Redirect 시키지 않으면 사용자가 브라우저를
* 새로 고침 하거나 재요청할 때 마다 이미 DB에서 삭제된 게시 글을 계속 삭제하려는
* 동작으로 인해서 문제가 발생할 수 있다. 이런 경우에는 Redirect 기법을 이용해 DB에
* 추가, 수정, 삭제가 아닌 조회하는 곳으로 이동하도록 하면 문제를 해결 할 수 있다.
*
* 현재 요청을 처리한 후에 Redirect 하려면 뷰 페이지를 지정하는 문자열 맨 앞에
* "r:" 또는 "redirect:"를 접두어로 붙여서 반환하고 Redirect가 아니라 Forward
* 하려면 뷰 페이지의 경로만 지정하여 문자열로 반환하면 Controller에서 판단하여
* Redirect 또는 Forward로 연결된다.
*
* 게시 글 삭제하기 요청을 처리한 후에 게시 글 리스트 페이지로 이동시키기 위해
* View 페이지 정보를 반환할 때 맨 앞에 "r:" 접두어를 붙여서 게시 글 리스트 보기
* 요청을 처리하는 URL를 지정하여 Controller로 넘기면 Controller는 넘겨 받은
* View 페이지 정보를 분석하여 Redirect 시키게 된다.
*
* Redirect는 클라이언트 요청에 대한 결과 페이지가 다른 곳으로 이동되었다고
* 브라우저에게 알려주고 그 이동된 주소로 다시 요청하라고 브라우저에게 URL을
* 보내서 브라우저가 그 URL로 다시 응답하도록 처리하는 것으로 아래와 같이
* View 페이지 정보의 맨 앞에 "r:" 또는 "redirect:"를 접두어로 붙여서 반환하면
* Controller에서 View 페이지 정보를 분석해 Redirect 시키고 이 응답을 받은
* 브라우저는 게시 글 리스트를 보여주는 페이지를 다시 요청하게 된다.
**/
return "r:" + url;
}
}

```


그림 2-1을 살펴보면 header, footer 부분은 매 페이지마다 공통으로 구현해야 하는 부분이고 가운데 Content 부분은 사용자 요청에 따라 동적으로 변화되는 부분이다.

참고로 그림 2-1은 header, footer, Content를 구성하는 3개의 JSP 페이지와 이들을 하나의 페이지로 동작할 수 있도록 묶어 주는 웹 템플릿을 합쳐 전체 4개의 JSP 파일로 구성한 것이다.

그림 2-2는 웹 템플릿을 구성하고 사용자 요청에 따라서 동적으로 변화되는 웹 페이지의 이미지이다. 그림 2-2를 살펴보면 header, footer는 사용자 요청에 따라서 거의 변화가 없는 정적인 부분이고 파란색 줄로 감싼 부분은 사용자 요청에 따라서 동적으로 변화되는 부분이다. 이렇게 웹 페이지에서 각 페이지마다 공통적으로 표현되어야 할 영역과 동적으로 변화는 영역이 존재하는데 이들을 웹 템플릿에 포함시킬 때는 정적인 영역은 주로 include 지시자 사용해서 컴파일 타임에 웹 템플릿에 하나의 파일로 합쳐지도록 하고 동적인 영역은 주로<jsp:include> 표준 액션을 사용해 실행 타임에 웹 템플릿에 동적으로 포함되도록 구성할 수 있다.

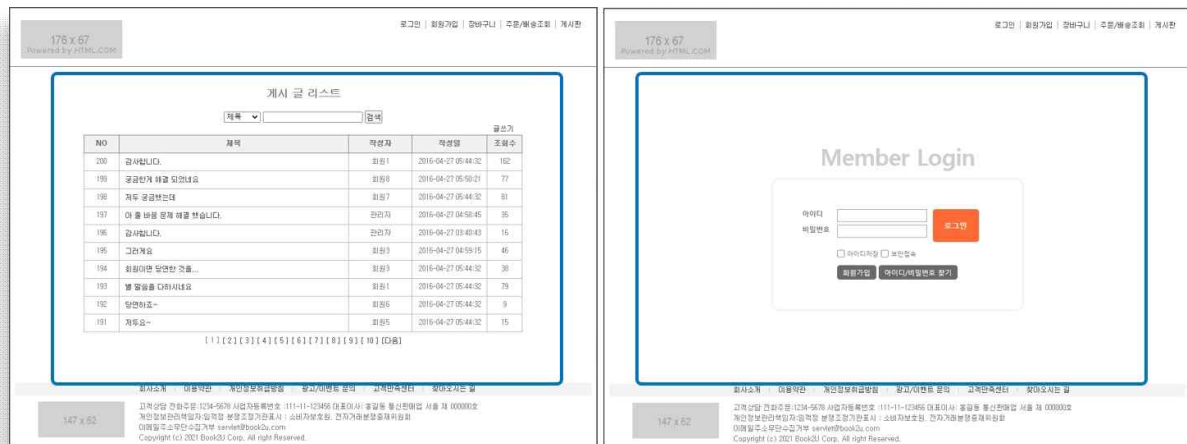


그림 2-2 사용자 요청에 따라 동적으로 변화는 Content 영역

웹 템플릿을 구현할 때는 아래 그림 2-3과 같이 include 지시자와 <jsp:include> 표준 액션을 사용해 구성한다. 아래는 웹 템플릿인 index.jsp에서 각각의 모듈로 분리된 jsp 페이지들을 하나의 페이지로 동작할 수 있도록 묶어 주는 웹 템플릿의 <body> 부분 코드이다.

그림 2-3 웹 템플릿 코드에서 동적으로 변화되는 부분을 살펴보면 EL의 param 내장객체가 사용되는 코드를 볼 수 있다. 요청에 따라서 동적으로 변화되는 콘텐츠 영역은 요청을 처리한 후 그 결과를 웹 템플릿(index.jsp)에 동적으로 포함시키기 위해 아래와 같이 “body” 라는 파라미터를 사용해 View 페이지를 지정할 수 있다. 아래 방법이 절대적인 것은 아니고 모델1에서는 각각의 페이지가 위치한 폴더를 어떻게 구성하느냐에 따라 달라질 수 있고 모델2 기반의 MVC 패턴에서는 컨트롤러에서 요청을 처리한 결과를 뷰 페이지에 어떻게 매핑 하나에 따라서 동적인 요소를 웹 템플릿에 포함시킬 수 있는 여러 가지 방법을 생각할 수 있다.

참고로 header, footer, Content 페이지에서 사용되는 CSS, JavaScript 파일의 참조는 각 페이지마다 할 필요 없이 전체 페이지를 하나로 묶어주는 웹 템플릿 페이지인 index.jsp 한 곳에 코드를 작성하면 된다.


```

<body>
  <div id="wrap">
    <%@ include file="pages/header.jsp" %>
    <c:if test="${ not empty param.body }">
      <jsp:include page="${ param.body }" />
    </c:if>
    <%@ include file="pages/footer.jsp" %>
  </div>
</body>

```

그림 2-3 웹 템플릿 코드

▶ 모델1에서 웹 템플릿 구현 예

index.jsp?body=board/boardList.jsp

index.jsp?body=board/writeForm.jsp

▶ 모델2 기반 MVC 패턴에서 웹 템플릿 구현 예

MVC 패턴에서는 프론트컨트롤러에서 모든 요청을 받고 모델 클래스를 이용해 요청을 처리한 후에 포워딩 방식으로 뷰로 이동해 최종 결과를 만들기 때문에 아래와 같이 WEB-INF 폴더에 웹 템플릿을 만들고 최종적으로 클라이언트에 응답되는 뷰 페이지를 만들 수 있다.

아래는 클라이언트로 응답되는 최종 결과를 만들기 위해서 서버 내부에서 사용되는 파일의 위치로 웹브라우저의 주소 표시줄에는 서블릿 매핑에 따라서 다르게 표시되기 때문에 웹브라우저 주소 표시줄에 JSP 페이지의 정보가 표시되는 모델1에 비해 보안에도 유리하다.

/WEB-INF/index.jsp?body=board/boardList.jsp

/WEB-INF/index.jsp?body=board/writeForm.jsp

우리는 모델2 기반의 MVC 패턴 웹 애플리케이션을 구현하고 있으므로 모든 요청을 프론트컨트롤러에서 받고 요청을 처리한 결과를 뷰로 전달(Forward)하여 최종적인 뷰를 만들기 때문에 컨트롤러에서 Forward 될 때 웹 템플릿인 index.jsp가 항상 호출될 수 있도록 구현해야 한다. 또한 index.jsp 페이지는 include 지시자, <jsp:include> 표준액션 태그 등을 사용해 서로 분리되어있는 각각의 JSP 페이지를 하나로 묶어서 하나의 웹 페이지로 동작할 수 있도록 구성해야 한다.

이번 예제는 앞에서 구현한 완성 프로젝트인 **JSPStudyMvcBBS01 프로젝트에 웹 템플릿을 적용하고 로그인/로그아웃과 에러 페이지 처리 기능을 추가하는 것이므로** 기존의 코드와 동일한 부분은 전체 코드를 교안에 추가하지 않고 설명만 추가할 것이다.

먼저 웹 템플릿을 적용하기 위해서 필요한 실습용 파일은 완성 프로젝트인 JSPStudyMvcBBS02 프로젝트의 “webapp/WEB-INF/Template/” 디렉터리를 찾아보면 볼 수 있을 것이다. 이 디렉터리의

파일을 아래의 표 2-1을 참고해서 웹 템플릿 실습용 프로젝트에 복사하자.

Template/webapp	실습 프로젝트
/css 폴더의 파일	/webapp/css 폴더에 복사
/images 폴더의 파일	/webapp/images 폴더에 복사
/js 폴더의 파일	/webapp/js 폴더에 복사
/WEB-INF/dao/MemberDao.java	com.jspstudy.bbs.dao 패키지에 복사
/WEB-INF/errors 폴더의 파일	/WEB-INF/errors 폴더에 복사
/WEB-INF/member 폴더의 파일	/WEB-INF/member 폴더에 복사
/WEB-INF/pages 폴더의 파일	/WEB-INF/pages 폴더에 복사
/WEB-INF/index.jsp	/WEB-INF/ 에 복사

표 2-1 웹 템플릿 실습파일

▶ 모델을 동일한 방식으로 실행하기 위한 슈퍼 인터페이스

- com.jspstudy.bbs.service.CommandProcess

// 모든 모델 클래스가 구현하는 슈퍼 인터페이스

```
public interface CommandProcess {
    public abstract String requestProcess(
        HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException;
}
```

▶ 컨트롤러 구현

앞에서 if문을 사용해 명령을 분석하여 요청을 처리하는 컨트롤러를 구현해 보았다. 이런 컨트롤러는 명령이 추가될 때마다 명령을 처리하기 위해서 if ~ else if 문이 추가되는데 이런 경우에 컨트롤러의 소스 코드가 수정되기 때문에 매번 다시 컴파일을 해야 하는 문제가 발생하게 된다. 이런 문제를 해결하기 위해서 properties 설정 파일과 Map을 활용해 컨트롤러를 구현할 수 있지만 초보자가 프로그램의 흐름을 이해하는데 있어서 if문 보다 어렵다고 느끼는 것 같아 MVC 패턴의 실행 흐름을 보다 쉽게 이해할 수 있도록 계속해서 if문을 사용해 명령을 처리하는 방식으로 컨트롤러를 구현할 것이며 Map을 활용해 컨트롤러를 구현한 예제는 별도의 소스 코드로 제공할 것이니 참고하기 바란다.

앞에서 if 문을 사용해 컨트롤러를 구현할 때 각각의 모델 클래스가 모두 다른 클래스 타입으로 정의되어 있기 때문에 컨트롤러 클래스에서 사용하는 참조 변수의 타입이 모두 다른 타입으로 정의되어 사용되었다. 이렇게 되면 모델 클래스의 추가나 변경으로 인해서 컨트롤러의 수정이 또 발생하게 되는데 이는 유지보수 차원에서도 바람직하지 못하다. 그래서 이런 문제를 해소하기 위해서 요청을 처리하는 슈퍼 인터페이스를 정의하고 실제 요청을 처리하는 모든 모델 클래스가 이 슈퍼 인터페이스를 구현해서 요청을 처리할 수 있도록 할 것이다. 이렇게 되면 컨트롤러에서 사용하는 모델 클래스의 객체를 참조할 때 모든 모델 클래스들의 부모타입 인 슈퍼 인터페이스 타입 하나로 참조할 수 있기 때문에 모든 모델 클래스를 하나의 타입으로 다룰 수 있고 오버라이딩 기법을 통해 다형성을

구현할 수 있게 된다.

앞의 JSPStudyMvcBBS01 프로젝트의 컨트롤러와 코드가 거의 비슷하지만 @WebServlet 애노테이션을 이용해 서블릿을 등록하는 부분을 이번 예제에서는 삭제하고 web.xml에 서블릿을 등록할 것이다. 또한 서블릿 초기화 메서드 안에서 애플리케이션 초기화 파라미터를 읽어서 File 객체를 생성하는 부분과 모든 모델 클래스가 상속한 슈퍼 인터페이스 하나의 타입으로 모델 클래스를 참조하는 부분이 앞의 예제와 다르다. **변경되거나 수정되는 부분은 빨간색 볼드체로 표시하였다.**

- com.jspstudy.bbs.controller.BoardController

// @WebServlet 애노테이션을 삭제하고 클래스 이름을 BoardController로 바꿀 것

```
public class BoardController extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /* 뷰 페이지 정보 중에서 앞부분과 뒷부분에서 중복되는 데이터를
     * 최소화하기 위해서 사용하는 접두어와 접미어를 상수로 설정
     *
     * 이번 프로젝트는 뷰 페이지를 구성할 때 중복되는 코드를 최소화하기 위해서
     * 웹 템플릿을 사용하는 프로젝트로 여러 JSP 페이지를 모듈화 하여 분리해 놓고
     * 실행될 때 웹 템플릿을 통해서 하나의 JSP 페이지로 동작하도록 구현되어 있다.
     * 그러므로 포워딩을 해야 할 경우에는 웹 템플릿 역할을 하는 index.jsp에
     * 동적으로 포함시킬 콘텐츠 페이지를 body라는 파라미터를 통해서 전달하여
     * 웹 템플릿 페이지인 index.jsp에서 body라는 파라미터 값을 읽어서 동적으로
     * 포함되는 콘텐츠 페이지를 include 하도록 상대 참조 방식으로 지정하면 된다.
     */
    private final String PREFIX = "/WEB-INF/index.jsp?body=";
    private final String SUFFIX = ".jsp";

    /* 서블릿 초기화 메서드
     * 이 서블릿 클래스의 인스턴스가 생성되고 아래 초기화 메서드가 딱 한 번 호출 된다.
     */
    public void init() throws ServletException {

        /* 애플리케이션 초기화 파라미터로 읽어온 업로드 파일이 저장될 폴더의
         * 로컬 경로를 구하여 그 경로와 파일명으로 File 객체를 생성한다.
         */
        ServletContext sc = getServletContext();
        String uploadDir = sc.getInitParameter("uploadDir");
        String realPath = sc.getRealPath(uploadDir);
        File parentFile = new File(realPath);

        /* 파일 객체에 지정한 위치에 디렉토리가 존재하지 않거나
         * 파일 객체가 디렉토리가 아니라면 디렉토리를 생성한다.
         */
        if(! (parentFile.exists() && parentFile.isDirectory())) {
            parentFile.mkdir();
        }
    }
}
```

```

}

/* 업로드 폴더 정보를 ServletContext 객체의 속성에 저장한다.
 * ServletContext 객체는 웹 애플리케이션 당 1개가 생성되며 웹 애플리케이션이
 * 구동되는데 필요한 정보를 담고 있는 객체로 JSP 페이지에서는 application
 * 내장객체로 접근할 수 있다. 아래와 같이 ServletContext 객체의 속성에
 * 저장되면 이 웹 애플리케이션의 모든 컴포넌트에서 이 정보를 사용할 수 있다.
 */
sc.setAttribute("uploadDir", uploadDir);
sc.setAttribute("parentFile", parentFile);
System.out.println("init - " + parentFile);
}

// get 방식의 요청을 처리하는 메소드
protected void doGet(
    HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    doProcess(request, response);
}

// post 방식의 요청을 처리하는 메소드
protected void doPost(
    HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    request.setCharacterEncoding("utf-8");
    doProcess(request, response);
}

/* doGet(), doPost()에서 호출하는 메소드
 * 즉 get방식과 post방식 요청을 모두 처리하는 메소드 이다.
 * 컨트롤러는 이 메소드 안에서 브라우저의 요청에 대한 처리를 요청 URL을 분석해
 * 요청을 처리할 모델 클래스를 결정하고 해당 모델 클래스의 객체를 사용해(위임)
 * 클라이언트의 요청을 처리한 후 그 결과를 뷰로 전달해 결과 화면을 만들게 된다.
 * 뷰로 전달된 데이터는 html 형식의 문서에 출력하여 브라우저에게 응답한다.
 */
public void doProcess(
    HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    /* 컨트롤러에 어떤 요청이 들어왔는지 파악하기 위해 Request 객체로 부터
     * 웹 어플리케이션의 컨텍스트 루트를 포함한 요청 페이지의 URI 정보와
     * ContextPath를 얻어와 String 클래스의 substring()을 이용해
     * 웹 어플리케이션 루트부터 요청한 페이지의 정보를 추출 한다.
     */
    /* 요청 정보를 담고 있는 Request 객체로 부터 요청 URI를 구한다.

```

```

    * /JSPStudyMvcBBS02/boardList.mvc
    **/
String requestURI = request.getRequestURI();

/* 요청 정보를 담고 있는 Request 객체로 부터 ContextPath를 구한다.
    * /JSPStudyMvcBBS02
    **/
String contextPath = request.getContextPath();
System.out.println("uri : " + requestURI + ", ctxPath : " + contextPath);

/* 요청 URI에서 ContextPath를 제외한 요청 명령을 추출 한다.
    * /boardList.mvc
    **/
String command = requestURI.substring(contextPath.length());
System.out.println("command : " + command);

// 모든 모델 클래스가 상속받는 슈퍼 인터페이스
CommandProcess service = null;

/* 뷰 페이지 정보를 저장할 변수
    **/
String viewPage = null;

/* 요청 URI에서 추출한 명령을 비교해 요청을 처리할 모델 클래스를
    * 결정하여 객체를 생성하고 메서드를 호출해 요청을 처리한다.
    **/
if(command.equals("/boardList.mvc")
    || command.equals("/*.mvc")
    || command.equals("/index.mvc")) {

    /* 게시 글 리스트 보기가 요청된 경우의 처리
        * 한 페이지에 출력 할 게시 글 리스트를 DB로 부터 읽어오는
        * BoardListService 클래스의 인스턴스를 생성한 후 Request와
        * Response 객체를 매개변수로 requestProcess()를
        * 호출하여 게시 글 리스트 보기에 대한 요청을 처리 한다.
        **/
    service = new BoardListService();
    viewPage = service.requestProcess(request, response);

} else if(command.equals("/boardDetail.mvc")) {

    /* 게시 글 내용보기가 요청된 경우의 처리
        * 게시 글 하나의 내용을 DB로 부터 읽어오는 BoardDetailService
        * 클래스의 인스턴스를 생성한 후 Request와 Response 객체를
        * 매개변수로 requestProcess()를 호출하여 게시 글 내용보기에

```

```

    * 대한 요청을 처리 한다.
    **/
    service = new BoardDetailService();
    viewPage = service.requestProcess(request, response);

} else if (command.equals("/writeForm.mvc")) {

    /* 게시 글쓰기 폼을 요청한 경우의 처리
    * 게시 글쓰기 폼 요청을 처리하는 WriteFormService 클래스의
    * 인스턴스를 생성한 후 Request와 Response 객체를 매개변수로
    * requestProcess()를 호출하여 게시 글쓰기 폼 요청을 처리한다.
    **/
    service = new WriteFormService();
    viewPage = service.requestProcess(request, response);

} else if (command.equals("/writeProcess.mvc")) {

    /* 게시 글쓰기 폼에서 등록하기 버튼이 클릭된 경우의 처리
    * 사용자가 작성한 게시 글을 DB에 저장하는 BoardWriteService
    * 클래스의 인스턴스를 생성한 후 Request와 Response 객체를
    * 매개변수로 requestProcess()를 호출하여 새로운 게시 글을
    * DB에 저장 한다.
    **/
    service = new BoardWriteService();
    viewPage = service.requestProcess(request, response);

} else if (command.equals("/updateForm.mvc")) {

    /* 게시 글 내용보기에서 수정하기 버튼이 클릭된 경우의 처리
    * 게시 글 수정 폼 요청을 처리하는 UpdateFormService 클래스의
    * 인스턴스를 생성한 후 Request와 Response 객체를 매개변수로
    * requestProcess()를 호출하여 게시 글 수정 폼 요청을 처리한다.
    **/
    service = new UpdateFormService();
    viewPage = service.requestProcess(request, response);

} else if (command.equals("/updateProcess.mvc")) {

    /* 게시 글 수정 폼에서 수정하기 버튼이 클릭된 경우의 처리
    * 게시 글 수정 요청을 처리하는 UpdateService 클래스의
    * 인스턴스를 생성한 후 Request와 Response 객체를 매개변수로
    * requestProcess()를 호출하여 게시 글을 DB에서 수정한다.
    **/
    service = new UpdateService();
    viewPage = service.requestProcess(request, response);

```

```

} else if(command.equals("/deleteProcess.mvc")) {

    /* 게시 글 내용보기에서 수정하기 버튼이 클릭된 경우의 처리
     * 게시 글 수정 폼 요청을 처리하는 UpdateFormService 클래스의
     * 인스턴스를 생성한 후 Request와 Response 객체를 매개변수로
     * requestProcess()를 호출하여 게시 글 수정 폼 요청을 처리한다.
     */
    service = new DeleteService();
    viewPage = service.requestProcess(request, response);

}

/* Redirect 정보와 View 페이지의 경로 정보를 저장하는 viewPage가
 * null이 아니면 Redirect 여부를 판단하여 Redirect라면 Response 객체의
 * sendRedirect()를 이용해 Redirect 시키고 Redirect가 아니라면
 * RequestDispatcher를 이용해 View 페이지로 포워딩 시킨다.
 */
if(viewPage != null) {

    /* 모델 클래스가 반환한 viewPage에 "redirect" 또는 "r" 접두어가
     * 존재하면 아래의 viewPage.split(":")[0] 코드에서 "redirect" 또는
     * "r" 문자열이 반환되고 그렇지 않으면 Forward 할 뷰 페이지의 경로가
     * 반환되므로 다음과 같이 Redirect와 Forward를 구분하여 처리할 수 있다.
     */
    String view = viewPage.split(":")[0];
    System.out.println("view : " + view);

    if(view.equals("r") || view.equals("redirect")) {
        response.sendRedirect(viewPage.split(":")[1]);

    } else {

        /* PREFIX는 view 정보 중 앞에서 중복되는 부분을 없애기 위해 사용
         * SUFFIX는 view 정보 중 뒤에서 중복되는 부분을 없애기 위해 사용
         */
        RequestDispatcher rd =
            request.getRequestDispatcher(PREFIX + view + SUFFIX);
        rd.forward(request, response);
    }
}
}
}

```

▶ web.xml 배포서술자

앞에서 @WebServlet 애노테이션을 사용해 서블릿을 등록하고 서블릿 초기화 파라미터를 설정해 컨트롤러를 구현했지만 이번 예제는 배포서술자(Deployment Descriptor)인 web.xml에 서블릿을 등록하였다. 또한 파일 업로드 폴더를 애플리케이션 초기화 파라미터로 설정해 서블릿 초기화 메서드인 init() 메서드 안에서 읽어 들여 웹 애플리케이션에서 참조할 수 있도록 구성할 것이다. 그리고 웹 애플리케이션이 요청을 처리하는 과정에서 에러가 발생하면 처리할 에러 페이지도 web.xml에 지정할 것이다. **아래에서 추가되거나 수정되는 부분은 빨간색 볼드체로 표시하였다.**

- /WEB-INF/web.xml 에 추가

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  id="WebApp_ID"
  version="3.0">

  <display-name>JSPStudyMvcBBS02</display-name>

  <!--
    아래의 리소스가 META-INF 디렉터리의 context.xml 파일에 정의되어
    있다면 서블릿 3.0부터는 반드시 정의해야 하는 것은 아니다. 다만 웹 어플리케이션을
    위해 JNDI를 사용하는 리소스에 대해 web.xml에 정의하는 것을 권장하고 있다.
    http://kenu.github.io/tomcat70/docs/jndi-resources-howto.html 참고
  -->
  <resource-ref>
    <description>dbcp 정의</description>
    <res-ref-name>jdbc/bbsDBPool</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>

  <!-- 애플리케이션 초기화 파라미터 - 파일을 업로드할 폴더 -->
  <context-param>
    <param-name>uploadDir</param-name>
    <param-value>upload</param-value>
  </context-param>

  <!-- 프론트 컨트롤러 등록 -->
  <servlet>
    <servlet-name>boardController</servlet-name>
    <servlet-class>com.jspstudy.bbs.controller.BoardController</servlet-class>
  </servlet>
```



```

<!-- 프론트 컨트롤러 맵핑 -->
<servlet-mapping>
  <servlet-name>boardController</servlet-name>
  <url-pattern>*.mvc</url-pattern>
</servlet-mapping>

<!-- 에러 페이지 설정 -->
<error-page>
  <error-code>500</error-code>
  <location>/WEB-INF/index.jsp?body=errors/errorpage.jsp</location>
</error-page>
<error-page>
  <error-code>404</error-code>
  <location>/WEB-INF/index.jsp?body=errors/errorpage.jsp</location>
</error-page>
</web-app>

```

▶ 웹 애플리케이션에서 에러가 발생하면 표시할 에러 페이지

- /WEB-INF/errors/errorpage.jsp

```

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

```

```

<%-- page 지시자를 사용해 이 페이지가 에러를 처리하는 페이지임을 지정한다. --%>

```

```

<%@ page isErrorPage="true" %>

```

```

<%-- 현재 페이지가 정상적으로 처리되었다는 응답 상태 코드 설정 --%>

```

```

<% response.setStatus(200); %>

```

```

<div class="row my-5" id="global-content">

```

```

  <div class="col">

```

```

    <div id="errorpage" class="my-5">

```

```

```

```

      <map name="errorpage">

```

```

        <area shape="rect" coords="336, 136, 436, 159"

```

```

          href="index.mvc" alt="홈으로 이동" />

```

```

      </map>

```

```

    </div>

```

```

  </div>

```

```

</div>

```

▶ 웹 템플릿 페이지

웹 템플릿은 여러 개의 JSP 페이지를 하나의 웹 페이지로 동작할 수 있도록 묶어주는 역할을 하는 페이지로 include 지시자와 <jsp:include> 표준액션 태그를 사용해 필요한 페이지를 index.jsp 페이지에 포함되도록 작성하면 된다.

- webapp/WEB-INF/index.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html lang="ko">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>JSP MVC 게시판</title>
    <link href="bootstrap/bootstrap.min.css" rel="stylesheet" >
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.9.1/font/bootstrap-icons.css"
        rel="stylesheet"
    >
    <link rel="stylesheet" type="text/css" href="css/global.css" />
    <link rel="stylesheet" type="text/css" href="css/member.css" />
    <style>
    </style>
    <script src="js/jquery-3.3.1.min.js"></script>
    <script src="js/formcheck.js"></script>
    <script src="js/member.js"></script>
</head>
<body>
    <div class="container">
        <%@ include file="pages/header.jsp" %>
        <jsp:include page="{ param.body }" />
        <%@ include file="pages/footer.jsp" %>
    </div>
    <script src="bootstrap/bootstrap.bundle.min.js"></script>
</body>
</html>
```

아래의 header.jsp, footer.jsp 파일과 각 요청에 대한 뷰 페이지는 이제 웹 템플릿 페이지인 index.jsp 파일의 <div class="container"> 태그의 자식으로 include 되도록 작성할 것이므로 html 태그와 body 태그 등은 기술할 필요가 없다. 또한 header.jsp 페이지에서 회원 로그인과 로그아웃에 대한 링크 처리를 해야 하지만 지금은 웹 템플릿을 먼저 구현하고 뒤이어서 회원 로그인과 로그아웃을 처리하는 부분에 대해 설명할 것이므로 일단은 다음과 같이 작성하자.

- webapp/WEB-INF/pages/header.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!-- header -->
<div class="row border-bottom border-primary">
    <div class="col-4">
```

```

        <p></p>
    </div>
    <div class="col-8">
        <div class="row">
            <div class="col">
                <ul class="nav justify-content-end">
                    <li class="nav-item">
                        <a class="nav-link" href="#">로그인-폼</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="#">로그인-모달</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="boardList.mvc">게시 글 리스트</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="#">회원가입</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="#">주문/배송조회</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="#">고객센터</a>
                    </li>
                </ul>
            </div>
        </div>
    </div>
    <div class="row">
        <div class="col text-end">로그인시 인사말 출력</div>
    </div>
</div>
</div>

```

- webapp/WEB-INF/pages/footer.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!-- footer -->
<div class="row border-top border-primary my-5" id="global-footer">
    <div class="col text-center py-3">
        <p>고객상담 전화주문:1234-5678 사업자등록번호 :111-11-123456
        대표이사: 홍길동 통신판매업 서울 제 000000호<br/>
        개인정보관리책임자:임격정 분쟁조정기관표시 : 소비자보호원, 전자거래분쟁중재위원회<br/>
        Copyright (c) 2023 JSP2U Corp. All right Reserved.
    </p>
    </div>

```

```
</div>
</div>
```

- webapp/css/global.css

```
@CHARSET "UTF-8";
/* 에러페이지 */
#errorpage {
    width: 460px;
    margin: 150px auto;
    padding: 100px 0;
}
#errorpage img { border: none; }
```

▶ 게시 글 리스트, 상세보기, 쓰기, 수정, 삭제 기능

이번 예제는 앞의 프로젝트인 **JSPStudyMvcBBS01 프로젝트**와 게시판 기능의 소스는 거의 동일하지만 게시 글 리스트, 상세보기, 쓰기 폼, 수정 폼의 JSP 페이지는 웹 템플릿 파일인 index.jsp 파일에 동적으로 포함(include)되기 때문에 각 jsp 페이지에서 **header.jsp, footer.jsp**와 중복되는 부분은 삭제하면 된다. 이 외에 게시 글 기능에 대한 **DBManager, DAO, Service** 클래스와 자바스크립트 파일인 **formcheck.js** 파일은 앞의 프로젝트와 동일하기 때문에 교안에는 생략되었다.

이번 프로젝트에서 모델 클래스들을 동일한 타입의 변수로 실행하기 위해서 모든 서비스 클래스가 구현해야 되는 **CommandProcess** 슈퍼 인터페이스가 추가되었다. 그러므로 모든 **Service** 클래스가 **CommandProcess**를 구현하도록 다음과 같이 **implements** 키워드를 이용해 상속해야 한다.

```
public class BoardListService implements CommandProcess {
    public String requestProcess(
        HttpServletRequest request, HttpServletResponse response)
        throws ServletException {

        // 메소드의 소스 코드 내용은 앞의 프로젝트와 동일함

    }
}
```

2-2) 회원 로그인/로그아웃 구현하기

웹 템플릿과 에러 페이지가 잘 적용되었다면 이제는 회원 로그인과 로그아웃 기능을 구현해 보자. 이번에 회원 로그인을 구현할 때 사용할 로그인 폼은 2가지 유형을 사용해 로그인을 처리하는 방법에 대해서 알아볼 것이다. 로그인 폼 2가지 유형 중 하나는 별도의 로그인 폼 페이지를 표시하여 사용자로부터 회원 아이디와 비밀번호를 입력 받을 수 있도록 구현할 것이고 또 다른 하나는 로그인 요청이 발생한 현재 페이지에서 팝업 형식으로 로그인 폼 모달 창을 표시하여 회원 아이디와 비밀번호를 입력 받을 수 있도록 구현할 것이다.

이번에 구현하는 회원 로그인 기능에서 로그인 폼을 2가지를 사용한다고 해서 로그인을 처리하는 로직을 2가지로 작성해야 하는 것은 아니다. 단지 로그인 폼만 2가지 형식으로 만들뿐 로그인을 처리하는 로직은 하나만 구현하면 된다. 다시 말해 Controll, Service, Dao 클래스에서 로그인 기능을 처리하는 메서드는 하나만 작성하면 된다는 말이다.

먼저 JSPStudyMvcBBS02 프로젝트의 webapp/WEB-INF/sql/member.sql 파일을 이용해 DB에 테이블을 만들고 회원 정보를 입력한 후에 다음 코드를 참고해서 회원 관련 DB 작업을 전담할 MemberDao 클래스를 작성하자.

▶ DAO(DataAccessObject) 클래스

- com.jspstudy.bbs.dao.MemberDao

// 회원 테이블에 접근하여 요청을 처리하는 DAO(Data Access Object) 클래스

```
public class MemberDao {
```

```
    // 데이터베이스 작업에 필요한 객체 타입으로 변수를 선언
```

```
    // Connection 객체는 DB에 연결해 작업을 수행할 수 있도록 지원하는 객체
```

```
    private Connection conn;
```

```
    // Statement, PreparedStatement 객체는 DB에 쿼리를 발행하는 객체
```

```
    private PreparedStatement pstmt;
```

```
    // ResultSet 객체는 DB에 SELECT 쿼리를 발행한 결과를 저장하는 객체
```

```
    private ResultSet rs;
```

```
    // 회원 로그인을 처리하는 메소드
```

```
    public int checkMember(String id, String pass) {
```

```
        String loginSql = "SELECT pass FROM member WHERE id = ?";
```

```
        // -1 아이디 없음, 0 비밀번호 틀림, 1 로그인 성공
```

```
        // 1, 2, 3
```

```
        int result = -1;
```

```
        String password = "";
```

```
        try {
```

```
            // DBCP로 부터 Connection을 대여한다.
```

```
            conn = DBManager.getConnection();
```

```
            pstmt = conn.prepareStatement(loginSql);
```

```
            // loginSql 쿼리의 플레이스홀더(?)에 대응하는 데이터를 설정한다.
```

```
            pstmt.setString(1, id);
```

```
            // DB에 쿼리를 전송하여 결과를 ResultSet으로 받는다.
```

```
            rs = pstmt.executeQuery();
```

```

/* ResultSet에 데이터가 존재하지 않으면 가입된 회원이 아니므로 -1을 반환
 * 회원 id는 Primary Key로 중복되지 않기 때문에 회원 테이블에서 SELECT한
 * 결과가 단일 행으로 반환 되므로 if문을 사용해 rs.next()를 호출했다.
 */
if(rs.next()) {
    // ResultSet에 데이터가 존재하면 ID에 대한 비밀번호를 읽어 온다.
    password = rs.getString("pass");
} else {
    return result;
}

/* 로그인 요청시 입력한 비밀번호와 회원 테이블에서 SELECT한 결과로
 * 읽어온 비밀번호가 일치하면 1을 반환 하고 일치하지 않으면 0을 반환 한다.
 */
if(password.equals(pass)) {
    result = 1;
} else {
    result = 0;
}
} catch(Exception e) {
    e.printStackTrace();
} finally {
    DBManager.close(conn, pstmt, rs);
}
return result;
}
}

```

▶ 컨트롤러에 회원 로그인/로그아웃 요청 처리 추가

BoardController의 doProcess() 메서드 안에 요청 명령에 따라 모델 클래스를 호출해 요청을 처리하는 if~else if 문 맨 아래에 다음 코드를 참고해서 회원 로그인과 로그아웃을 처리하는 코드를 추가하자.

- com.jspstudy.bbs.controller.BoardController에 추가

// @WebServlet 애노테이션을 삭제하고 클래스 이름을 BoardController로 바꿀 것

```

public class BoardController extends HttpServlet {

```

... 앞부분 코드 생략 - 요청 처리 if ~ else if 문에 아래 코드 추가 ...

```

} else if(command.equals("/loginForm.mvc")) {

    /* 상단 메뉴에서 로그인 메뉴가 클릭된 경우의 처리
     * 회원 로그인 폼 요청을 처리하는 LoginFormService 클래스의
     * 인스턴스를 생성한 후 Request와 Response 객체를 매개변수로

```

```

        * requestProcess()를 호출하여 회원 로그인 요청을 처리한다.
        **/
        service = new LoginFormService();
        viewPage = service.requestProcess(request, response);

    } else if(command.equals("/login.mvc")) {

        /* 로그인 폼에서 로그인 버튼이 클릭된 경우의 처리
        * 회원 로그인 요청을 처리하는 LoginService 클래스의
        * 인스턴스를 생성한 후 Request와 Response 객체를 매개변수로
        * requestProcess()를 호출하여 회원 로그인 요청을 처리한다.
        **/
        service = new LoginService();
        viewPage = service.requestProcess(request, response);

    } else if(command.equals("/logout.mvc")) {

        /* 로그아웃이 클릭된 경우의 처리
        * 회원 로그아웃 요청을 처리하는 LogoutService 클래스의
        * 인스턴스를 생성한 후 Request와 Response 객체를 매개변수로
        * requestProcess()를 호출하여 회원 로그아웃 요청을 처리한다.
        **/
        service = new LogoutService();
        viewPage = service.requestProcess(request, response);

    }
}

```

... 뒷부분 코드 생략 ...

}

▶ 회원 로그인 폼 요청을 처리할 모델 클래스

앞에서 언급했던 것처럼 모든 모델 클래스는 컨트롤러에서 하나의 타입으로 다룰 수 있도록 CommandProcess 슈퍼 인터페이스를 구현하도록 작성해야 하므로 회원 로그인과 로그아웃 관련 모델 클래스도 CommandProcess 인터페이스를 구현하도록 작성해야 한다.

- com.jspstudy.bbs.service.LoginFormService

// 로그인 폼 요청을 처리하는 모델 클래스

```

public class LoginFormService implements CommandProcess {

    public String requestProcess(
        HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

```

```

/* 최종적으로 Redirect 정보와 View 페이지 정보를 문자열로 반환하면 된다.
*
* 회원 로그인 폼 요청을 처리하는 모델 클래스는 회원 로그인 폼만 보여주면
* 되므로 화면에 출력할 모델 데이터가 필요없다. 그러므로 회원 로그인 폼에
* 대한 View 정보만 반환하면 된다.
*
* 요청에 대한 결과(모델)를 출력할 View 페이지와 View 페이지를 호출하는 방식을
* 아래와 같이 문자열로 지정하면 된다. 현재 요청을 처리한 후에 Redirect 하려면
* 뷰 페이지를 지정하는 문자열 맨 앞에 "r:" 또는 "redirect:"를 접두어로 붙여서
* 반환하고 Redirect가 아니라 Forward 하려면 뷰 페이지의 경로만 지정하여
* 문자열로 반환하면 Controller에서 판단하여 Redirect 또는 Forward로 연결된다.
* 또한 Forward 할 때 뷰 페이지의 정보 중에서 앞부분과 뒷부분에서 중복되는
* 정보를 줄이기 위해서 Controller에서 PREFIX와 SUFFIX를 지정해 사용하기
* 때문에 매번 중복되는 부분을 제외하고 뷰 페이지의 정보를 지정하면 된다.
**/
return "member/loginForm";
}
}

```

▶ 회원 로그인 폼

앞에서 언급했던 것처럼 이번에 회원 로그인을 구현할 때 사용할 로그인 폼은 별도의 로그인 폼 페이지를 표시하는 방식과 현재 페이지에서 팝업 형식으로 로그인 폼 모달 창을 표시하는 2가지 유형을 사용할 것이므로 별도의 로그인 폼 페이지는 따로 작성하면 되지만 팝업 형식으로 로그인 폼 모달 창을 표시하기 위해서는 전체 뷰 페이지에 적용될 수 있는 곳에 로그인 폼 모달 창을 작성해야 하므로 index.jsp 파일에 로그인 폼 모달 창을 작성해야 한다. 앞의 예제에서 **추가되거나 수정되는 부분은 붉은색 볼드체로 표시하였다.**

* 메인 템플릿 페이지에 로그인 모달 창 코드 추가

- webapp/WEB-INF/index.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html lang="ko">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>JSP MVC 게시판</title>
    <link href="bootstrap/bootstrap.min.css" rel="stylesheet" >
    <link href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.9.1/font/bootstrap-icons.css"
    rel="stylesheet"
    <link rel="stylesheet" type="text/css" href="css/global.css" />
    <link rel="stylesheet" type="text/css" href="css/member.css" />
    <style>

```



```

</style>
<script src="js/jquery-3.3.1.min.js"></script>
<script src="js/formcheck.js"></script>
<script src="js/member.js"></script>
</head>
<body>
<div class="container">
  <%@ include file="pages/header.jsp" %>
  <jsp:include page="{ param.body }" />
  <%@ include file="pages/footer.jsp" %>
</div>
<script src="bootstrap/bootstrap.bundle.min.js"></script>

<!-- 로그인 모달 -->
<div class="modal fade" id="loginModal" tabindex="-1" aria-labelledby="loginModalLabel"
aria-hidden="true"
  data-bs-backdrop="static" data-bs-keyboard="false">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header bg-primary bg-gradient text-white">
        <h1 class="modal-title fs-5 fw-bold" id="modalLabel">회원 로그인</h1>
        <button type="button" class="btn-close" data-bs-dismiss="modal"
aria-label="Close"></button>
      </div>
      <form action="login.mvc" method="post">
        <div class="modal-body">
          <div class="mb-3">
            <label for="userId" class="col-form-label fw-bold">아이디 : </label>
            <input type="text" class="form-control" id="userId" name="id">
          </div>
          <div class="mb-3">
            <label for="pass" class="col-form-label fw-bold">비밀번호 : </label>
            <input type="password" class="form-control" id="pass" name="pass">
          </div>
        </div>
        <div class="modal-footer">
          <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">취소
</button>
          <button type="submit" class="btn btn-primary">로그인</button>
        </div>
      </form>
    </div>
  </div>
</div>
</div>

```

```
</body>
</html>
```

▶ 템플릿 헤더 페이지

웹 페이지에서 위쪽의 header.jsp 페이지는 로그인 상태일 때와 로그아웃 상태일 때 상단의 메뉴가 변경되기 로그인과 로그아웃에 대한 처리가 필요하다. 아래 header.jsp 페이지에서 로그인과 로그아웃 처리가 필요한 부분을 빨간색 볼드체로 표시하였다.

- webapp/WEB-INF/pages/header.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!-- header -->
<div class="row border-bottom border-primary" id="global-header">
    <div class="col-4">
        <p></p>
    </div>
    <div class="col-8">
        <div class="row mt-1">
            <div class="col">
                <ul class="nav justify-content-end">
                    <li class="nav-item">
                        <a class="nav-link"
                            href='${ sessionScope.isLogin ? "logout.mvc" : "loginForm.mvc" }'>
                            ${ sessionScope.isLogin ? "로그아웃" : "로그인-폼" }
                        </a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link "
                            ${ not sessionScope.isLogin ? "data-bs-toggle='modal'
data-bs-target='#loginModal'" : ""}
                            href='${ sessionScope.isLogin ? "logout.mvc" : "#" }'>
                            ${ sessionScope.isLogin ? "로그아웃" : "로그인-모달" }
                        </a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="boardList.mvc">게시 글 리스트</a>
                    </li>
                    <li class="nav-item">
                        <c:if test='${ not sessionScope.isLogin }' >
                            <a class="nav-link" href="#">회원가입</a>
                        </c:if>
                        <c:if test='${ sessionScope.isLogin }' >
                            <a class="nav-link" href="#">정보수정</a>
                        </c:if>
                    </li>
                </ul>
            </div>
        </div>
    </div>
</div>
```

```

        </c:if>
    </li>
    <li class="nav-item">
        <a class="nav-link" href="#">주문/배송조회</a>
    </li>
    <li class="nav-item">
        <a class="nav-link" href="#">고객센터</a>
    </li>
</ul>
</div>
</div>
<div class="row">
    <div class="col text-end">&nbsp;</div>
</div>
<div class="row">
    <div class="col text-end pe-5 text-primary">
        <c:if test="${ sessionScope.isLogin }" >
            <div>안녕하세요 ${ sessionScope.id }님</div>
        </c:if>
    </div>
</div>
</div>
</div>
</div>

```

▶ 회원 로그인 폼페이지

- webapp/WEB-INF/member/loginForm.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<div class="row my-5" id="global-content">
    <div class="col">
        <form class="my-5" id="loginForm" action="login.mvc" method="post">
            <h2 class="fw-bold">Member Login</h2>
            <fieldset>
                <legend>Member Loin</legend>
                <div id="login">
                    <p>
                        <label for="userId" class="labelStyle">아이디</label>
                        <input type="text" id="userId" name="id" />
                    </p>
                    <p>
                        <label for="userPass" class="labelStyle">비밀번호</label>
                        <input type="password" id="userPass" name="pass"/>
                    </p>
                </div>
            </fieldset>
        </form>
    </div>
</div>

```

```

        <input type="submit" value="로그인" id="btnLogin" />
    <p id="btn1">
        <input type="checkbox" id="saveId" value="savedIdYes" />
        <label for="saveId">아이디저장</label>
        <input type="checkbox" id="secure" value="secureYes" />
        <label for="secure">보안접속</label>
    </p>
    <p id="btn2">
        <input type="button" value="회원가입" id="btnJoin" />
        <input type="button" value="아이디/비밀번호 찾기" id="btnSearch" />
    </p>
</fieldset>
</form>
</div>
</div>

```

- webapp/css/member.css

```

@CHARSET "UTF-8";
/* 로그인 폼 */
#loginForm h2, #loginForm p,
#loginForm input, #loginForm label {
    margin: 0px;
    padding: 0px;
    font-family: "맑은 고딕",돋움;
    font-size: 12px;
}
#loginForm h2 {
    font-size: 40px;
    letter-spacing: -1px;
    color: #C8C8C8;
    text-align: center;
}
#loginForm legend {
    display: none;
}
#loginForm fieldset {
    width: 430px;
    margin: 10px auto;
    border: 5px solid #efefef;
    padding: 50px;
    border-radius: 15px;
}
#loginForm #login {
    float: left;

```

```

}
#loginForm label.labelStyle {
    width: 60px;
    display: block;
    float: left;
    font-weight: bold;
}
#loginForm #userId, #loginForm #userPass {
    width: 150px;
    border: 1px solid #999;
    margin-bottom: 5px;
    padding: 2px;
}
#loginForm #btnLogin {
    display: block;
    background-color: #FF6633;
    border-radius: 5px;
    border-style: none;
    color: #fff;
    width: 80px;
    height: 57px;
    position: relative;
    float: left;
    left: 10px;
    font-size: 13px;
    font-weight: bold;
    cursor: pointer;
}
#loginForm #btn1 {
    clear: both;
    margin-left: 60px;
    padding: 10px 0px;
}
#loginForm #btn1 label {
    font-size: 11px;
    vertical-align: middle;
}
#loginForm #btn1 input{
    height: 20px;
}
#loginForm #btn2 {
    margin-left: 60px;
}
#loginForm #btn2 input {
    background-color: #666;

```

```
border-style: none;
border-radius: 5px;
color: #fff;
height: 25px;
padding: 5px 10px;
}
```

- webapp/js/member.js

// DOM이 준비되면 실행될 콜백 함수

```
$(function() {
    $("#loginForm").submit(function() {
        var id = $("#userId").val();
        var pass = $("#userPass").val();

        if(id.length <= 0) {
            alert("아이디가 입력되지 않았습니다.\n아이디를 입력해주세요");
            $("#userId").focus();
            return false;
        }
        if(pass.length <= 0) {
            alert("비밀번호가 입력되지 않았습니다.\n비밀번호를 입력해주세요");
            $("#userPass").focus();
            return false;
        }
    });
});
```

▶ 회원 로그인 요청을 처리할 모델 클래스

- com.jspstudy.bbs.service.LoginService

// 회원 로그인 폼에서 들어오는 로그인 요청을 처리하는 모델 클래스

```
public class LoginService implements CommandProcess {

    @Override
    public String requestProcess(
        HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        String id = request.getParameter("id");
        String pass = request.getParameter("pass");

        /* 회원 로그인을 처리하기 위해 MemberDAO 객체를 얻어
         * 회원 테이블의 회원 정보와 비교하여 가입된 회원이 아니면 -1을
         * 로그인 성공 이면 1을 비밀번호가 맞지 않으면 0을 리턴 받는다.
```

```

    **/
MemberDao dao = new MemberDao();
int checkLogin = dao.checkMember(id, pass);

/* 존재하지 않는 아이디이거나 비밀번호가 틀리면 자바스크립트를 응답 데이터로
 * 보내서 경고 창을 띄우고 브라우저에 저장된 이전 페이지로 돌려보낸다.
 * Controller로 viewPage 정보를 반환해야 하지만 이 경우 viewPage 정보가
 * 없으므로 PrintWriter 객체를 이용해 클라이언트로 응답할 자바스크립트 코드를
 * 출력하고 null을 반환하면 Controller에서는 viewPage 정보가 null이 아닐 경우만
 * 처리하게 되므로 자바스크립트가 브라우저로 전송되어 경고 창이 뜨게 된다.
 */
if(checkLogin == -1) { // 아이디가 존재하지 않으면
    /* 스트림에 직접 쓰기위해 응답 객체로 부터 스트림을 구한다.
     * 응답 객체의 스트림을 구하기 전제 ContentType이 설정되어야 한다.
     * 그렇지 않으면 한글과 같은 데이터는 깨져서 출력된다.
     */
    response.setContentType("text/html; charset=utf-8");
    PrintWriter out = response.getWriter();
    out.println("<script>");
    out.println("alert('" + id + "는 가입되지 않은 아이디 입니다.');"");
    out.println("window.history.back();"");
    out.println("</script>");
    return null;

} else if(checkLogin == 0) { // 비밀번호가 틀리면
    /* 스트림에 직접 쓰기위해 응답 객체로 부터 스트림을 구한다.
     * 응답 객체의 스트림을 구하기 전제 ContentType이 설정되어야 한다.
     * 그렇지 않으면 한글과 같은 데이터는 깨져서 출력된다.
     */
    response.setContentType("text/html; charset=utf-8");
    PrintWriter out = response.getWriter();
    out.println("<script>");
    out.println("alert('비밀번호가 맞지 않습니다.');"");
    out.println("window.history.back();"");
    out.println("</script>");
    return null;

} else if(checkLogin == 1) { // 로그인 성공이면

    /* request 객체로 부터 HttpSession 객체를 구해
     * 세션 영역의 속성에 id와 로그인 상태 정보를 저장 한다.
     */
    HttpSession session = request.getSession();
    session.setAttribute("id", id);

```

```

        session.setAttribute("isLogin", true);
    }

    /* 최종적으로 Redirect 정보와 View 페이지 정보를 문자열로 반환하면 된다.
    *
    * 현재 요청을 처리한 후에 Redirect 하려면 뷰 페이지를 지정하는 문자열 맨 앞에
    * "r:" 또는 "redirect:"를 접두어로 붙여서 반환하고 Redirect가 아니라 Forward
    * 하려면 뷰 페이지의 경로만 지정하여 문자열로 반환하면 Controller에서 판단하여
    * Redirect 또는 Forward로 연결된다.
    *
    * 로그인 요청을 처리하고 Redirect 기법을 사용해 게시 글 리스트 페이지로
    * 이동시키기 위해서 View 페이지 정보를 반환할 때 맨 앞에 "r:" 접두어를 붙여서
    * 게시 글 리스트 보기 요청을 처리하는 URL을 지정하여 Controller로 넘기면
    * Controller는 넘겨받은 View 페이지 정보를 분석하여 Redirect 시키게 된다.
    *
    * Redirect는 클라이언트 요청에 대한 결과 페이지가 다른 곳으로 이동되었다고
    * 브라우저에게 알려주고 그 이동된 주소로 다시 요청하라고 브라우저에게 URL을
    * 보내서 브라우저가 그 URL로 다시 응답하도록 처리하는 것으로 아래와 같이
    * View 페이지 정보의 맨 앞에 "r:" 또는 "redirect:"를 접두어로 붙여서 반환하면
    * Controller에서 View 페이지 정보를 분석해 Redirect 시키고 이 응답을 받은
    * 브라우저는 게시 글 리스트를 보여주는 페이지를 다시 요청하게 된다.
    *
    * 지금과 같이 리다이렉트를 해야 할 경우 웹브라우저가 다시 요청할 주소만 응답하고
    * 웹브라우저에서는 이 주소로 재요청하는 동작을 하므로 웹 템플릿 페이지인
    * index.jsp를 기준으로 뷰 페이지를 지정하면 안 된다. 왜냐하면 리다이렉트는
    * 뷰 페이지를 거쳐서 클라이언트로 응답되는 것이 아니라 현재 클라이언트가 요청한
    * 주소가 다른 곳으로 이동되었다고 알려주기 위해 웹브라우저가 이동할 주소만
    * 응답하고 웹 브라우저는 서버로 부터 응답 받은 주소로 다시 요청하는 동작을 하기
    * 때문에 뷰 페이지의 정보가 아닌 웹 브라우저가 이동할 주소를 지정해야 한다.
    */
    return "r:boardList.mvc";
}
}

```

▶ 회원 로그아웃 요청을 처리할 모델 클래스

- com.jspstudy.bbs.service.LogoutService

// 회원 로그아웃 요청을 처리하는 모델 클래스

```
public class LogoutService implements CommandProcess {
```

```
@Override
```

```
public String requestProcess(
```

```
HttpServletRequest request, HttpServletResponse response)
```

```
throws ServletException, IOException {
```



```

/* request 객체로 부터 HttpSession 객체를 구해 현재 세션을 삭제한다.
 * 현재 세션이 삭제되기 때문에 세션 영역에 저장된 모든 데이터가 삭제된다.
 */
HttpSession session = request.getSession();
session.invalidate();

/*
// 아래와 같이 세션에 저장된 개별 속성만 삭제 할 수 있다.
session.removeAttribute("isLogin");
session.removeAttribute("id");
*/

/* 최종적으로 Redirect 정보와 View 페이지 정보를 문자열로 반환하면 된다.
 *
 * 현재 요청을 처리한 후에 Redirect 하려면 뷰 페이지를 지정하는 문자열 맨 앞에
 * "r:" 또는 "redirect:"를 접두어로 붙여서 반환하고 Redirect가 아니라 Forward
 * 하려면 뷰 페이지의 경로만 지정하여 문자열로 반환하면 Controller에서 판단하여
 * Redirect 또는 Forward로 연결된다.
 *
 * 로그아웃 요청을 처리하고 Redirect 기법을 사용해 게시 글 리스트 페이지로
 * 이동시키기 위해서 View 페이지 정보를 반환할 때 맨 앞에 "r:" 접두어를 붙여서
 * 게시 글 리스트 보기 요청을 처리하는 URL을 지정하여 Controller로 넘기면
 *   * Controller는 넘겨받은 View 페이지 정보를 분석하여 Redirect 시키게 된다.
 *
 * Redirect는 클라이언트 요청에 대한 결과 페이지가 다른 곳으로 이동되었다고
 * 브라우저에게 알려주고 그 이동된 주소로 다시 요청하라고 브라우저에게 URL을
 * 보내서 브라우저가 그 URL로 다시 응답하도록 처리하는 것으로 아래와 같이
 * View 페이지 정보의 맨 앞에 "r:" 또는 "redirect:"를 접두어로 붙여서 반환하면
 * Controller에서 View 페이지 정보를 분석해 Redirect 시키고 이 응답을 받은
 * 브라우저는 게시 글 리스트를 보여주는 페이지를 다시 요청하게 된다.
 *
 * 지금과 같이 리다이렉트를 해야 할 경우 웹브라우저가 다시 요청할 주소만 응답하고
 * 웹브라우저에서는 이 주소로 재요청하는 동작을 하므로 웹 템플릿 페이지인
 * index.jsp를 기준으로 뷰 페이지를 지정하면 안 된다. 왜냐하면 리다이렉트는
 * 뷰 페이지를 거쳐서 클라이언트로 응답되는 것이 아니라 현재 클라이언트가 요청한
 * 주소가 다른 곳으로 이동되었다고 알려주기 위해 웹브라우저가 이동할 주소만
 * 응답하고 웹 브라우저는 서버로 부터 응답 받은 주소로 다시 요청하는 동작을 하기
 * 때문에 뷰 페이지의 정보가 아닌 웹 브라우저가 이동할 주소를 지정해야 한다.
 */
return "r:boardList.mvc";
}
}

```

3) 회원제 게시판

- 실습용 프로젝트 : JSPClassMvcBBS03
- 완성 프로젝트 : JSPStudyMvcBBS03

먼저 회원제 게시판을 구현하기 위해서 필요한 실습용 파일은 완성 프로젝트인 JSPStudyMvcBBS03 프로젝트의 “webapp/WEB-INF/Template/” 디렉터리를 찾아보면 볼 수 있을 것이다. 이 디렉터리의 파일을 아래의 표를 참고해서 회원제 게시판 실습용 프로젝트에 복사하자.

Template/webapp	실습 프로젝트
/css 폴더의 파일	/webapp/css 폴더에 복사
/js 폴더의 파일	/webapp/js 폴더에 복사
/WEB-INF/member 폴더의 파일	/WEB-INF/member 폴더에 복사
/WEB-INF/vo 폴더의 파일	com.jspstudy.bbs.vo 패키지에 복사

회원제 게시판 실습파일

3-1) 회원 가입 구현하기

앞에서 구현한 웹 애플리케이션은 회원가입 없이 로그인/로그아웃 기능만 제공하고 있다. 실무에서는 회원가입 없이 로그인/로그아웃 기능만 제공하는 웹 애플리케이션은 없을 것이다 그래서 이번에는 회원가입을 구현하고 로그인을 하지 않으면 게시 글 리스트만 볼 수 있고 검색을 하거나 게시 글 상세보기 같은 기능은 이용할 수 없도록 구현할 것이다.

회원가입 시에 주소를 선택하고 우편번호를 입력하는 기능은 다음(daum.net)에서 제공하는 우편번호 API를 사용할 것이며 추가되거나 수정되는 부분은 전체 코드를 나열하지 않고 필요한 부분만 코드로 표시할 것이므로 전체 코드를 참고하려면 별도로 제공되는 프로젝트 소스를 참고하기 바란다.

먼저 JSPStudyMvcBBS03 프로젝트의 webapp/WEB-INF/sql/member.sql 파일을 이용해 DB에 테이블을 만들고 회원 정보를 입력한 후에 다음 코드를 참고해서 회원 한 명의 정보를 저장하는 VO(Value Object) 클래스인 Member 클래스와 회원 관련 DB 작업을 담당하는 MemberDao 클래스에 회원 가입 및 회원 정보 수정에 대한 메서드를 구현해 보자.

▶ 회원 한 명의 정보를 저장하는 Beans 클래스

- com.jspstudy.bbs.vo.Member

/* 한 명의 회원 정보를 저장하는 클래스(VO, Beans, DTO)

- * VO 객체에 저장될 데이터는 테이블에서 읽어오기 때문에 각각의 변수는
 - * 테이블에서 컬럼이 가지는 데이터 형식과 같거나 자동 형 변환이 가능해야 한다.
- */

```
public class Member {
```

```
    private String name, id, pass, email, mobile;
```

```
    private String phone, zipcode, address1, address2;
```

```

private boolean emailGet;
private Timestamp regDate;

public Member() { }

public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public String getId() {
    return id;
}
public void setId(String id) {
    this.id = id;
}
public String getPass() {
    return pass;
}
public void setPass(String pass) {
    this.pass = pass;
}
public String getEmail() {
    return email;
}
public void setEmail(String email) {
    this.email = email;
}
public String getMobile() {
    return mobile;
}
public void setMobile(String mobile) {
    this.mobile = mobile;
}
public String getPhone() {
    return phone;
}
public void setPhone(String phone) {
    this.phone = phone;
}
public String getZipcode() {
    return zipcode;
}
public void setZipcode(String zipcode) {

```

```

        this.zipcode = zipcode;
    }
    public String getAddress1() {
        return address1;
    }
    public void setAddress1(String address1) {
        this.address1 = address1;
    }
    public String getAddress2() {
        return address2;
    }
    public void setAddress2(String address2) {
        this.address2 = address2;
    }
    public boolean isEmailGet() {
        return emailGet;
    }
    public void setEmailGet(boolean emailGet) {
        this.emailGet = emailGet;
    }
    public Timestamp getRegDate() {
        return regDate;
    }
    public void setRegDate(Timestamp regDate) {
        this.regDate = regDate;
    }
}

```

▶ MemberDao 클래스에 메서드 추가

기존의 MemberDao 클래스에 회원 가입과 중복 아이디를 체크하는 메서드를 추가한다.

- com.jspstudy.bbs.dao.MemberDao

// 회원 테이블에 접근하여 요청을 처리하는 DAO(Data Access Object) 클래스

```
public class MemberDao {
```

... 앞부분 코드 생략 ...

// 회원 가입을 처리하는 메소드

```
public void joinMember(Member member) {
```

```
    String joinSql =
```

```
        "INSERT INTO member VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, SYSDATE)";
```

```
    try {
```

```

// DBManager을 이용해 DBCP로 부터 Connection을 대여한다.
conn = DBManager.getConnection();

pstmt = conn.prepareStatement(joinSql);

// joinSql 쿼리의 플레이스홀더(?)에 대응하는 데이터를 설정한다.
pstmt.setString(1, member.getId());
pstmt.setString(2, member.getName());
pstmt.setString(3, member.getPass());
pstmt.setString(4, member.getEmail());
pstmt.setString(5, member.getMobile());
pstmt.setString(6, member.getZipcode());
pstmt.setString(7, member.getAddress1());
pstmt.setString(8, member.getAddress2());
pstmt.setString(9, member.getPhone());
pstmt.setBoolean(10, member.isEmailGet());

// DB에 쿼리를 전송하여 회원 가입을 완료한다.
pstmt.executeUpdate();

} catch(Exception e) {
    e.printStackTrace();

} finally {

    // DBManager를 이용해 DB 작업에 사용된 자원을 해제 한다.
    DBManager.close(conn, pstmt);
}
}

// 중복 회원을 체크하는 메소드
public boolean overlapIdCheck(String id) {

    String overlapSql = "SELECT id FROM member Where id = ?";
    boolean result = false;

    try{
        // DBManager을 이용해 DBCP로 부터 Connection을 대여한다.
        conn = DBManager.getConnection();

        pstmt = conn.prepareStatement(overlapSql);

        // overlapSql 쿼리의 플레이스홀더(?)에 대응하는 데이터를 설정한다.
        pstmt.setString(1, id);

```

```

// DB에 쿼리를 전송하여 결과를 ResultSet으로 받는다.
rs = pstmt.executeQuery();

/* 회원 가입 폼에서 입력된 id를 회원 테이블에서 SELECT 하여 ResultSet에
 * 데이터가 존재하면 이미 가입된 회원 아이디이므로 true를 반환 한다.
 */
if(rs.next()) {
    result = true;
}
System.out.println("overlapIdCheck(String id)");

} catch(Exception e) {
    e.printStackTrace();

} finally{

    // DBManager를 이용해 DB 작업에 사용된 자원을 해제 한다.
    DBManager.close(conn, pstmt, rs);
}

return result;
}
}

```

▶ 컨트롤러에 회원가입관련 요청 처리 추가

BoardController 클래스의 doProcess() 메서드 안에 요청 명령에 따라 모델 클래스를 호출해 요청을 처리하는 if ~ else if 문 맨 아래에 다음 코드를 참고해서 회원 가입, 중복 아이디 체크 요청을 처리하는 else if 문과 모델 클래스에서 반환하는 뷰 페이지 정보인 viewPage의 값에 따라서 Redirect, 웹 템플릿을 적용하지 않는 Forward, 웹 템플릿을 적용하는 Forward를 처리하는 코드를 추가하자.

- com.jspstudy.bbs.controller.BoardController

```

// 게시판 및 회원관련 요청을 처리하는 프론트 컨트롤러
public class BoardController extends HttpServlet {

```

... 앞부분 코드 생략 ...

아래 부분에서 사용되는 Service 클래스를 CommandProcess 인터페이스를 상속하여 클래스만 먼저 만들어 놓고 교안의 순서대로 하나씩 세부적으로 구현할 것

```

} else if(command.equals("/joinForm.mvc")) {

    /* 회원 가입이 클릭된 경우의 처리

```

```

        * 회원 가입 폼 요청을 처리하는 JoinFormService 클래스의
        * 인스턴스를 생성한 후 Request와 Response 객체를 매개변수로
        * requestProcess()를 호출하여 회원 로그아웃 요청을 처리한다.
        */
        service = new JoinFormService();
        viewPage = service.requestProcess(request, response);

    } else if(command.equals("/overlapIdCheck.mvc")) {

        /* 회원 가입 폼에서 회원 아이디 중복확인 버튼이 클릭된 경우의 처리
        * 회원 아이디 중복확인 요청을 처리하는 OverlapIdCheckService 클래스의
        * 인스턴스를 생성한 후 Request와 Response 객체를 매개변수로
        * requestProcess()를 호출하여 회원 로그아웃 요청을 처리한다.
        */
        service = new OverlapIdCheckService();
        viewPage = service.requestProcess(request, response);

    } else if(command.equals("/joinResult.mvc")) {

        /* 회원 정보 입력 확인 화면에서 가입완료 버튼이 클릭된 경우의 처리
        * 회원 가입 요청을 처리하는 JoinResultService 클래스의
        * 인스턴스를 생성한 후 Request와 Response 객체를 매개변수로
        * requestProcess()를 호출하여 회원 가입 요청을 처리한다.
        */
        service = new JoinResultService();
        viewPage = service.requestProcess(request, response);
    }

    /* Redirect 정보와 View 페이지의 경로 정보를 저장하는 viewPage가
    * null이 아니면 Redirect 여부를 판단하여 Redirect라면 Response 객체의
    * sendRedirect()를 이용해 Redirect 시키고 Redirect가 아니라면
    * RequestDispatcher를 이용해 View 페이지로 포워딩 시킨다.
    */
    if(viewPage != null) {

        /* 모델 클래스가 반환한 viewPage에 "redirect" 또는 "r" 접두어가 존재하면
        * 아래의 viewPage.split(":")[0] 코드에서 "redirect" 또는 "r" 문자열이 반환되어
        * 아래 if 문에서 Redirect 되고 "forward" 또는 "f" 접두어가 존재하면
        * 아래 if 문에서 웹 템플릿을 적용하지 않은 뷰 페이지로 Forward 되고
        * 그렇지 않으면 웹 템플릿을 적용한 뷰 페이지로 Forward 된다.
        */
        String view = viewPage.split(":")[0];
        System.out.println("view : " + view);

        if(view.equals("r") || view.equals("redirect")) {

```

```

        // redirect인 경우 지정한 URL로 Redirect 시킨다.
        response.sendRedirect(viewPage.split(":")[1]);

    } else if(view.equals("f") || view.equals("forward")) {

        /* 웹 템플릿을 적용하지 않는 Forward인 경우
         * PREFIX와 SUFFIX를 적용하지 않고 지정한 view로 Forward 한다.
         */
        RequestDispatcher rd =
            request.getRequestDispatcher(viewPage.split(":")[1]);
        rd.forward(request, response);

    } else {

        /* 웹 템플릿을 적용하는 경우 PREFIX와 SUFFIX를 적용해 Forward 한다.
         * PREFIX는 view 정보 중 앞에서 중복되는 부분을 없애기 위해 사용
         * SUFFIX는 view 정보 중 뒤에서 중복되는 부분을 없애기 위해 사용
         */
        RequestDispatcher rd =
            request.getRequestDispatcher(PREFIX + view + SUFFIX);
        rd.forward(request, response);
    }
}
}
}

```

▶ 웹 템플릿 파일

이번 프로젝트에는 회원 가입과 회원 정보 수정하기가 추가되므로 header.jsp 파일의 <header> 태그 안에서 해당 하는 메뉴로 링크가 추가되어야 한다. **다음에서 수정되거나 추가되는 부분은 빨간색 볼드체로 표시하였다.**

- WEB-INF/pages/header.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!-- header -->
<div class="row border-bottom border-primary" id="global-header">
    <div class="col-4">
        <p></p>
    </div>
    <div class="col-8">
        <div class="row mt-1">
            <div class="col">

```



```

<ul class="nav justify-content-end">
  <li class="nav-item">
    <a class="nav-link"
      href='{ sessionScope.isLogin ? "logout.mvc" : "loginForm.mvc" }'>
      ${ sessionScope.isLogin ? "로그아웃" : "로그인-폼" }
    </a>
  </li>
  <li class="nav-item">
    <a class="nav-link "
      ${      not      sessionScope.isLogin      ?      "data-bs-toggle='modal'
data-bs-target='#loginModal'" : ""}
      href='{ sessionScope.isLogin ? "logout.mvc" : "#" }'>
      ${ sessionScope.isLogin ? "로그아웃" : "로그인-모달" }
    </a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="boardList.mvc">게시 글 리스트</a>
  </li>
  <li class="nav-item">
    <c:if test="${ not sessionScope.isLogin }" >
      <a class="nav-link" href="joinForm.mvc">회원가입</a>
    </c:if>
    <c:if test="${ sessionScope.isLogin }" >
      <a class="nav-link" href="memberUpdateForm.mvc">정보수정</a>
    </c:if>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">주문/배송조회</a>
  </li>
  <li class="nav-item">
    <a class="nav-link" href="#">고객센터</a>
  </li>
</ul>
</div>
</div>
<div class="row">
  <div class="col text-end">&nbsp;</div>
</div>
<div class="row">
  <div class="col text-end pe-5 text-primary">
    <c:if test="${ sessionScope.isLogin }" >
      <div>안녕하세요 ${ sessionScope.id }님</div>
    </c:if>
  </div>
</div>
</div>

```

</div>
</div>

▶ 회원가입 구현

회원가입과 관련된 기능은 이번 프로젝트에서 새롭게 추가되는 기능이기에 때문에 다음에서 설명하는 코드를 참고해서 클래스를 새롭게 만들고 코드를 참고해 우리 프로젝트에 추가하면 된다.

◆ 회원가입 폼 요청을 처리할 모델 클래스

- com.jspstudy.bbs.service.JoinFormService

// 회원가입 폼 요청을 처리하는 모델 클래스

```
public class JoinFormService implements CommandProcess {
```

```
    public String requestProcess(
```

```
        HttpServletRequest request, HttpServletResponse response)
```

```
        throws ServletException, IOException {
```

```
    /* 최종적으로 Redirect 정보와 View 페이지 정보를 문자열로 반환하면 된다.
```

```
    *
```

```
    * 회원 가입 폼 요청을 처리하는 모델 클래스는 회원 정보를 입력하는 폼만
```

```
    * 보여주면 되므로 화면에 출력할 모델 데이터가 필요없다. 그러므로 회원 정보를
```

```
    * 입력하는 폼에 대한 View 정보만 반환하면 된다.
```

```
    *
```

```
    * 요청에 대한 결과(모델)를 출력할 View 페이지와 View 페이지를 호출하는 방식을
```

```
    * 아래와 같이 문자열로 지정하면 된다. 현재 요청을 처리한 후에 Redirect 하려면
```

```
    * 뷰 페이지를 지정하는 문자열 맨 앞에 "r:" 또는 "redirect:"를 접두어로 붙여서
```

```
    * 반환하고 Redirect가 아니라 Forward 하려면 뷰 페이지의 경로만 지정하여
```

```
    * 문자열로 반환하면 Controller에서 판단하여 Redirect 또는 Forward로 연결된다.
```

```
    * 또한 Forward 할 때 뷰 페이지의 정보 중에서 앞부분과 뒷부분에서 중복되는
```

```
    * 정보를 줄이기 위해서 Controller에서 PREFIX와 SUFFIX를 지정해 사용하기
```

```
    * 때문에 매번 중복되는 부분을 제외하고 뷰 페이지의 정보를 지정하면 된다.
```

```
    *
```

```
    * 웹 템플릿을 적용하여 뷰를 만드는 경우 Controller에서 PREFIX에 웹 템플릿의
```

```
    * 위치가 지정되어 있으므로 PREFIX와 SUFFIX를 제외하고 뷰의 정보를 지정하면
```

```
    * 되지만 만약 웹 템플릿을 적용하지 않고 별도로 뷰를 만드는 경우에는 Forward
```

```
    * 할 때 PREFIX가 적용되지 않도록 Controller에 알려주기 위해서 아래 주석으로
```

```
    * 처리한 return 문과 같이 뷰 페이지 정보를 지정하는 문자열의 맨 앞에 "f:" 또는
```

```
    * "forward:"를 접두어로 붙여서 반환하면 된다.
```

```
    **/
```

```
    // return "f:/WEB-INF/member/overlapidCheck.jsp"
```

```
    return "member/memberJoinForm";
```

```
    }
```

```
}
```

◆ 회원가입 폼 요청 결과를 표시할 뷰 페이지

- webapp/WEB-INF/member/memberJoinForm.jsp

```
<%-- 회원가입 폼 요청 처리 결과를 출력할 View 페이지 --%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!--
    새로운 5자리 우편번호로 회원 주소를 입력 받기 위해 daum.net에서
    제공하는 우편번호 찾기 API를 사용하였다.
    참고 사이트 : http://postcode.map.daum.net/guide
-->
<script src=
"https://t1.daumcdn.net/mapjsapi/bundle/postcode/prod/postcode.v2.js"></script>
<div class="row my-5" id="global-content">
    <div class="col">
        <div class="row my-3 text-center">
            <div class="col">
                <h2 class="fs-3 fw-bold">회원 정보 입력</h2>
            </div>
        </div>
        <form action="joinResult.mvc" name="joinForm" method="post" id="joinForm">
            <%--
                회원 아이디 중복 검사를 했는지의 정보를 hidden 필드로 저장
            --%>
            <input type="hidden" name="isIdCheck" id="isIdCheck" value="false"/>
            <div class="row mt-5 mb-3">
                <div class="col-8 offset-2">
                    <label for="name" class="form-label">* 이 름 : </label>
                    <input type="text" class="form-control" name="name" id="name">
                </div>
            </div>
            <div class="row my-3">
                <div class="col-8 offset-2">
                    <label for="id" class="form-label">* 아이디 : </label>
                    <div class="row">
                        <div class="col-6">
                            <input type="text" class="form-control" name="id" id="userId">
                        </div>
                        <div class="col-4">
                            <input type="button" class="btn btn-warning" id="btnOverlapId" value="중복확
인">
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
```

```

<div class="col-8 offset-2">
  <label for="pass1" class="form-label">* 비밀번호 : </label>
  <input type="password" class="form-control" name="pass1" id="pass1">
</div>
</div>
<div class="row my-3">
  <div class="col-8 offset-2">
    <label for="pass2" class="form-label">* 비밀번호 확인 : </label>
    <input type="password" class="form-control" name="pass2" id="pass2">
  </div>
</div>
<div class="row my-3">
  <div class="col-8 offset-2">
    <label for="zipcode" class="form-label">* 우편번호 : </label>
    <div class="row">
      <div class="col-4">
        <input type="password" class="form-control" name="zipcode" id="zipcode"
maxlength="5" readonly>
      </div>
      <div class="col-4">
        <input type="button" class="btn btn-warning" id="btnZipcode" value="우편번호
찾기">
      </div>
    </div>
  </div>
</div>
<div class="row my-3">
  <div class="col-8 offset-2">
    <label for="address1" class="form-label">* 자택주소 : </label>
    <input type="text" class="form-control" name="address1" id="address1"
readonly>
  </div>
</div>
<div class="row my-3">
  <div class="col-8 offset-2">
    <label for="address2" class="form-label">상세주소 : </label>
    <input type="text" class="form-control" name="address2" id="address2">
  </div>
</div>
<div class="row my-3">
  <div class="col-8 offset-2">
    <label for="emailId" class="form-label">* 이메일 : </label>
    <div class="row">
      <div class="col-md-4">
        <input type="text" class="form-control" name="emailId" id="emailId">

```

```

</div> @
<div class="col-md-4">
  <input type="text" class="form-control" name="emailDomain"
id="emailDomain">
</div>
<div class="col-md-3">
  <select class="form-select" name="selectDomain" id="selectDomain">
    <option>직접입력</option>
    <option>네이버</option>
    <option>다음</option>
    <option>한메일</option>
    <option>구글</option>
  </select>
</div>
</div>
</div>
<div class="row my-3">
  <div class="col-8 offset-2">
    <label for="mobile2" class="form-label">* 휴 대 폰 : </label>
    <div class="row">
      <div class="col-md-3">
        <select class="form-select" name="mobile1" id="mobile1">
          <option>010</option>
          <option>011</option>
          <option>016</option>
          <option>017</option>
          <option>018</option>
          <option>019</option>
        </select>
      </div>-
      <div class="col-md-4">
        <input type="text" class="form-control" name="mobile2" id="mobile2"
maxlength="4">
      </div>-
      <div class="col-md-4">
        <input type="text" class="form-control" name="mobile3" id="mobile3"
maxlength="4">
      </div>
    </div>
  </div>
</div>
<div class="row my-3">
  <div class="col-8 offset-2">
    <label class="form-label">메일 수신여부 : </label>

```

```

<div class="row">
  <div class="col-md-3">
    <div class="form-check">
      <input type="radio" class="form-check-input" name="emailGet" id="emailOk"
value="true">
      <label class="form-check-label" for="emailOk">수신함</label>
    </div>
  </div>
  <div class="col-md-3">
    <div class="form-check">
      <input type="radio" class="form-check-input" name="emailGet" id="emailNo"
value="false">
      <label class="form-check-label" for="emailNo">수신않함</label>
    </div>
  </div>
</div>
<div class="row my-3">
  <div class="col-8 offset-2">
    <label for="phone2" class="form-label">자택전화 : </label>
    <div class="row">
      <div class="col-md-3">
        <select class="form-select" name="phone1" id="phone1">
          <option>02</option>
          <option>031</option>
          <option>032</option>
          <option>033</option>
          <option>041</option>
          <option>042</option>
          <option>043</option>
          <option>044</option>
          <option>051</option>
          <option>052</option>
          <option>053</option>
          <option>054</option>
          <option>055</option>
          <option>061</option>
          <option>062</option>
          <option>063</option>
          <option>064</option>
          <option>010</option>
          <option>011</option>
          <option>016</option>
          <option>017</option>
        </select>

```

```

        <option>018</option>
        <option>019</option>
    </select>
</div> -
<div class="col-md-4">
    <input type="text" class="form-control" name="phone2" id="phone2"
maxlength="4">
</div> -
<div class="col-md-4">
    <input type="text" class="form-control" name="phone3" id="phone3"
maxlength="4">
</div>
</div>
</div>
</div>
<div class="row mb-3 mt-5">
    <div class="col-8 offset-2">
        <input type="submit" value="가입하기" class="btn btn-primary">
    </div>
</div>
</form>
</div>
</div>

```

◆ 회원가입 폼에서 아이디 중복검사 요청을 처리할 모델 클래스

- com.jspstudy.bbs.service.OverlapIdCheckService

// 회원 가입시 아이디 중복검사 요청을 처리하는 모델 클래스

```

public class OverlapIdCheckService implements CommandProcess {

    public String requestProcess(
        HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        String id = request.getParameter("id");

        /* 회원 아이디 중복 확인을 처리하기 위해 MemberDAO 객체를 얻어
        * 회원 테이블에서 입력한 id의 회원이 존재하는지 조회 한다.
        * overlapIdCheck()는 중복된 id면 true, 중복된 id가 아니면 false가 반환 됨
        */
        MemberDao dao = new MemberDao();
        boolean overlap = dao.overlapIdCheck(id);

        // Request 영역의 속성에 입력된 id와 회원 아이디 중복 여부를 저장 한다.
        request.setAttribute("id", id);
    }
}

```

```

request.setAttribute("overlap", overlap);

/* 최종적으로 Redirect 정보와 View 페이지 정보를 문자열로 반환하면 된다.
 *
 * 게시 글 리스트 요청에 대한 결과(모델)를 request 영역의 속성에 저장하고
 * 요청에 대한 결과(모델)를 출력할 View 페이지와 View 페이지를 호출하는 방식을
 * 아래와 같이 문자열로 지정하면 된다. 현재 요청을 처리한 후에 Redirect 하려면
 * 뷰 페이지를 지정하는 문자열 맨 앞에 "r:" 또는 "redirect:"를 접두어로 붙여서
 * 반환하고 Redirect가 아니라 Forward 하려면 뷰 페이지의 경로만 지정하여
 * 문자열로 반환하면 Controller에서 판단하여 Redirect 또는 Forward로 연결된다.
 * 또한 Forward 할 때 뷰 페이지의 정보 중에서 앞부분과 뒷부분에서 중복되는
 * 정보를 줄이기 위해서 Controller에서 PREFIX와 SUFFIX를 지정해 사용하기
 * 때문에 매번 중복되는 부분을 제외하고 뷰 페이지의 정보를 지정하면 된다.
 *
 * 웹 템플릿을 적용하여 뷰를 만드는 경우 Controller에서 PREFIX에 웹 템플릿의
 * 위치가 지정되어 있으므로 PREFIX와 SUFFIX를 제외하고 뷰의 정보를 지정하면
 * 되지만 만약 웹 템플릿을 적용하지 않고 별도로 뷰를 만드는 경우에는 Forward
 * 할 때 PREFIX가 적용되지 않도록 Controller에 알려주기 위해서 아래 주석으로
 * 처리한 return 문과 같이 뷰 페이지 정보를 지정하는 문자열의 맨 앞에 "f:" 또는
 * "forward:"를 접두어로 붙여서 반환하면 된다.
 */
return "f:/WEB-INF/member/overlapIdCheck.jsp";
}
}

```

◆ 회원가입 폼에서 아이디 중복검사 요청 결과를 표시할 뷰 페이지

- webapp/WEB-INF/member/overlapIdCheck.jsp

```

<%--
회원 가입시 아이디 중복검사 요청에 대한 처리 결과를 출력할 View 페이지
이 페이지는 새창으로 실행되고 중복 아이디 체크를 할 수 있는 폼을 제공한다.
--%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<link href="bootstrap/bootstrap.min.css" rel="stylesheet" >
<script src="js/jquery-3.3.1.min.js"></script>
<script src="js/member.js"></script>
<title>중복 아이디 체크</title>
</head>
<body>

```



```

<div class="row my-5" id="global-content">
  <c:choose>
    <c:when test="{ overlap }" >
      <div class="col">
        <div class="row text-center">
          <div class="col">
            <h2 class="fs-3 fw-bold">사용할 수 없는 아이디</h2>
          </div>
        </div>
        <div class="row my-3 text-center">
          <div class="col">
            입력하신 ${id}는 이미 사용 중인 아이디 입니다.
          </div>
        </div>
        <div class="row my-3">
          <div class="col text-center">
            다른 아이디를 선택해 주세요
          </div>
        </div>
        <form action="overlapIdCheck.mvc" name="idCheckForm"
          method="post" id="idCheckForm" class="row mt-5">
          <div class="col-10 offset-1">
            <div class="input-group">
              <span class="input-group-text">* 아이디 : </span>
              <input type="text" class="form-control" name="id" id="checkId">
              <input type="submit" class="btn btn-primary" value="중복확인">
            </div>
          </div>
        </form>
      </div>
    </c:when>
    <c:otherwise>
      <div class="col">
        <div class="row text-center">
          <div class="col">
            <h2 class="fs-3 fw-bold">사용할 수 있는 아이디</h2>
          </div>
        </div>
        <div class="row my-3 text-center">
          <div class="col">
            입력하신 ${id}는 사용할 수 있는 아이디 입니다.
          </div>
        </div>
        <div class="row mt-5">
          <div class="col text-center">

```

```

        <input type="button" value="${ id }을(를) 아이디로 사용하기"
            id="btnIdCheckClose" data-id-value="${ id }" class="btn btn-primary"/>
    </div>
</div>
</div>
</c:otherwise>
</c:choose>
</div>
</body>
</html>

```

◆ 회원가입 폼에서 사용되는 JavaScript

- webapp/js/member.js 에 아래 코드를 추가

// DOM이 준비되면 실행될 콜백 함수

```
$(function() {
```

... 앞부분 코드 생략 ...

```

/* 회원 가입 폼, 회원정보 수정 폼에서 폼 컨트롤에서 키보드 입력을
 * 체크해 유효한 값을 입력 받을 수 있도록 keyup 이벤트를 처리 했다.
 */

```

```

$("#userId").on("keyup", function() {
    // 아래와 같이 정규표현식을 이용해 영문 대소문자, 숫자만 입력되었는지 체크할 수 있다.
    var regExp = /^[A-Za-z0-9]/gi;
    if(regExp.test($(this).val())) {
        alert("영문 대소문자, 숫자만 입력할 수 있습니다.");
        $(this).val($(this).val().replace(regExp, ""));
    }
});

```

```

/* 바로 위에서 키보드의 키가 눌러질 때 사용자가 아이디 입력란에 영문 대소문자
 * 또는 숫자만 입력하였는지를 체크하는 이벤트 처리 코드를 작성하였다. 이런 이벤트
 * 처리는 아이디 입력 체크에만 사용할 수 있는 것이 아니라 비밀번호 입력이나 이메일
 * 입력에서도 유효한 데이터가 입력되었는지 체크해야 된다. 이렇게 여러 문서 객체에
 * 동일한 이벤트 처리를 해야 하므로 각각 이벤트 처리를 구현하게 되면 동일한 코드가
 * 계속해서 중복되는 현상이 발생하게 된다. 이럴 경우에는 동일한 코드를 하나의
 * 함수로 만들어 놓고 여러 곳에서 사용하면 코드의 중복을 막고 코드의 재사용성을
 * 높일 수 있다. 아래는 동일한 코드에 대한 별도의 함수를 이 자바스크립트 파일의
 * 아래 부분에 작성해 놓고 이 함수를 이벤트 핸들러를 등록해 사용하는 방식이다.
 */

```

```

$("#pass1").on("keyup", inputCharReplace);
$("#pass2").on("keyup", inputCharReplace);
$("#emailId").on("keyup", inputCharReplace);
$("#emailDomain").on("keyup", inputEmailDomainReplace);

```

```

/* 회원 가입 폼에서 아이디 중복확인 버튼이 클릭되면
 * 아이디 중복을 확인할 수 있는 새 창을 띄워주는 함수
 */
$("#btnOverlapId").on("click", function() {
    var id = $("#userId").val();
    url="overlapIdCheck.mvc?id=" + id;

    if(id.length == 0) {
        alert("아이디를 입력해주세요");
        return false;
    }

    if(id.length < 5) {
        alert("아이디는 5자 이상 입력해주세요.");
        return false;
    }

    window.open(url, "idCheck", "toolbar=no, scrollbars=no, resizable=no, "
        + "status=no, memubar=no, width=500, height=400");
});

/* 새 창으로 띄운 아이디 찾기 폼에서
 * 아이디 중복확인 버튼이 클릭되면 유효성 검사를 하는 함수
 */
$("#idCheckForm").on("submit", function() {
    var id = $("#checkId").val();

    if(id.length == 0) {
        alert("아이디를 입력해주세요");
        return false;
    }

    if(id.length < 5) {
        alert("아이디는 5자 이상 입력해주세요.");
        return false;
    }
});

/* 새 창으로 띄운 아이디 중복확인 창에서 "아이디 사용 버튼"이 클릭되면
 * 새 창을 닫고 입력된 아이디를 부모창의 회원가입 폼에 입력해 주는 함수
 */
$("#btnIdCheckClose").on("click", function() {
    var id = $(this).attr("data-id-value");
    opener.document.joinForm.id.value = id;
});

```

```

    opener.document.joinForm.isIdCheck.value = true;
    window.close();
});

/* 회원 가입 폼과 회원정보 수정 폼에서 "우편번호 검색" 버튼의 클릭 이벤트 처리
 * findZipcode() 함수는 다음 우편번호 API를 사용해 우편번호를 검색하는 함수로
 * 두 페이지에서 사용되어 중복된 코드가 발생하므로 아래에 별도의 함수로 정의하였다.
 */
$("#btnZipcode").click(findZipcode);

// 이메일 입력 셀렉트 박스에서 선택된 도메인을 설정하는 함수
$("#selectDomain").on("change", function() {
    var str = $(this).val();

    if(str == "직접입력") {
        $("#emailDomain").val("");
        $("#emailDomain").prop("readonly", false);
    } else if(str == "네이버"){
        $("#emailDomain").val("naver.com");
        $("#emailDomain").prop("readonly", true);

    } else if(str == "다음") {
        $("#emailDomain").val("daum.net");
        $("#emailDomain").prop("readonly", true);

    } else if(str == "한메일"){
        $("#emailDomain").val("hanmail.net");
        $("#emailDomain").prop("readonly", true);

    } else if(str == "구글") {
        $("#emailDomain").val("gmail.com");
        $("#emailDomain").prop("readonly", true);
    }
});

// 회원 가입 폼이 서브밋 될 때 이벤트 처리 - 폼 유효성 검사
$("#joinForm").on("submit", function() {

    /* 회원 가입 폼과 회원 정보 수정 폼 유효성 검사를 하는 기능도 중복 되는
     * 코드가 많으므로 joinFormCheck()라는 별도의 함수를 만들어서 사용하였다.
     * joinFormChcek() 함수에서 폼 유효성 검사를 통과하지 못하면
     * false가 반환되기 때문에 그대로 반환하면 폼이 서브밋 되지 않는다.
     */
    return joinFormCheck(true);
});

```

```

/* 회원 아이디, 비밀번호, 비밀번호 확인, 이메일 아이디 폼 컨트롤에
 * 사용자가 입력한 값이 영문 대소문자, 숫자 만 입력되도록 수정하는 함수
 */
function inputCharReplace() {
    // 아래와 같이 정규표현식을 이용해 영문 대소문자, 숫자만 입력되었는지 체크할 수 있다.
    var regExp = /^[A-Za-z0-9]/gi;
    if(regExp.test($(this).val())) {
        alert("영문 대소문자, 숫자만 입력할 수 있습니다.");
        $(this).val($(this).val().replace(regExp, ""));
    }
}

/* 이 메일 도메인 입력 폼 컨트롤에 사용자가 입력한 값이
 * 영문 대소문자, 숫자, 점(.)만 입력되도록 수정하는 함수
 */
function inputEmailDomainReplace() {
    var regExp = /^[a-z0-9\.]$/gi;
    if(regExp.test($(this).val())) {
        alert("이메일 도메인은 영문 소문자, 숫자, 점(.)만 입력할 수 있습니다.");
        $(this).val($(this).val().replace(regExp, ""));
    }
}

/* 회원 가입 폼과 회원정보 수정 폼의 유효성 검사를 하는 함수
 * 두 페이지에서 처리하는 코드가 중복되어 하나의 함수로 정의하였다.
 */
function joinFormCheck(isJoinForm) {
    var name = $("#name").val();
    var id = $("#userId").val();
    var pass1 = $("#pass1").val();
    var pass2 = $("#pass2").val();
    var zipcode = $("#zipcode").val();
    var address1 = $("#address1").val();
    var emailId = $("#emailId").val();
    var emailDomain = $("#emailDomain").val();
    var mobile2 = $("#mobile2").val();
    var mobile3 = $("#mobile2").val();
    var isIdCheck = $("#isIdCheck").val();

    if(name.length == 0) {
        alert("이름이 입력되지 않았습니다.\n이름을 입력해주세요");
        return false;
    }
    if(id.length == 0) {

```

```

    alert("아이디가 입력되지 않았습니다.\n아이디를 입력해주세요");
    return false;
}
if(isJoinForm && isIdCheck == 'false') {
    alert("아이디 중복 체크를 하지 않았습니다.\n아이디 중복 체크를 해주세요");
    return false;
}
if(pass1.length == 0) {
    alert("비밀번호가 입력되지 않았습니다.\n비밀번호를 입력해주세요");
    return false;
}

if(pass2.length == 0) {
    alert("비밀번호 확인이 입력되지 않았습니다.\n비밀번호 확인을 입력해주세요");
    return false;
}
if(pass1 != pass2) {
    alert("비밀번호와 비밀번호 확인이 일치하지 않습니다.");
    return false;
}
if(zipcode.length == 0) {
    alert("우편번호가 입력되지 않았습니다.\n우편번호를 입력해주세요");
    return false;
}
if(address1.length == 0) {
    alert("주소가 입력되지 않았습니다.\n주소를 입력해주세요");
    return false;
}
if(emailId.length == 0) {
    alert("이메일 아이디가 입력되지 않았습니다.\n이메일 아이디를 입력해주세요");
    return false;
}
if(emailDomain.length == 0) {
    alert("이메일 도메인이 입력되지 않았습니다.\n이메일 도메인을 입력해주세요");
    return false;
}
if(mobile2.length == 0 || mobile3.length == 0) {
    alert("휴대폰 번호가 입력되지 않았습니다.\n휴대폰 번호를 입력해주세요");
    return false;
}
}

/* 우편번호 찾기 - daum 우편번호 찾기 API 이용
* 회원 가입 폼과 회원정보 수정 폼에서 "우편번호 검색" 버튼이 클릭되면 호출되는 함수
*

```

- * 새로운 5자리 우편번호로 회원 주소를 입력 받기 위해 daum.net에서
- * 제공하는 우편번호 찾기 API를 사용하였다.
- * 참고 사이트 : <http://postcode.map.daum.net/guide>

```

**/
function findZipcode() {
    new daum.Postcode({
        oncomplete: function(data) {
            // 우편번호 검색 결과 항목을 클릭했을때 실행할 코드를 여기에 작성한다.
            // 각 주소의 노출 규칙에 따라 주소를 조합한다.
            // 내려오는 변수가 값이 없는 경우엔 공백('')값을 가지므로, 이를 참고하여 분기 한다.

            var addr = ''; // 주소 변수
            var extraAddr = ''; // 참고 항목 변수

            //사용자가 선택한 주소 타입과 상관없이 모두 도로명 주소로 처리
            addr = data.roadAddress;

            // 법정동명이 있을 경우 추가한다. (법정리는 제외)
            // 법정동의 경우 마지막 문자가 "동/로/가"로 끝난다.
            if(data.bname !== '' && /[동|로|가]$/g.test(data.bname)){
                extraAddr += data.bname;
            }
            // 건물명이 있고, 공동주택일 경우 추가한다.
            if(data.buildingName !== '' && data.apartment === 'Y'){
                extraAddr += (extraAddr !== '' ?
                    ', ' + data.buildingName : data.buildingName);
            }

            // 표시할 참고 항목이 있을 경우, 괄호까지 추가한 최종 문자열을 만든다.
            if(extraAddr !== ''){
                extraAddr = ' (' + extraAddr + ')';
            }

            // 조합된 참고 항목을 상세주소에 추가한다.
            addr += extraAddr;

            // 우편번호와 주소 정보를 해당 입력상자에 출력한다.
            $("#zipcode").val(data.zonecode);
            $("#address1").val(addr);

            // 커서를 상세주소 입력상자로 이동한다.
            $("#address2").focus();
        }
    }).open();
}

```

```
}
```

◆ 회원가입 완료 요청을 처리할 모델 클래스

- com.jspstudy.bbs.service.JoinResultService

```
/* 회원가입 폼에서 사용자가 입력한 데이터를 받아서
 * 회원정보를 DB에 저장하여 회원가입 요청을 처리하는 모델 클래스
 * 회원 가입이 완료되면 로그인 처리도 같이 한다.
 */
```

```
public class JoinResultService implements CommandProcess {

    public String requestProcess(
        HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        // 회원 가입 폼으로 부터 전달된 파라미터를 읽어 변수에 저장 한다.
        request.setCharacterEncoding("utf-8");
        String id = request.getParameter("id");
        String name = request.getParameter("name");
        String pass = request.getParameter("pass1");
        String emailId = request.getParameter("emailId");
        String emailDomain = request.getParameter("emailDomain");
        String mobile1 = request.getParameter("mobile1");
        String mobile2 = request.getParameter("mobile2");
        String mobile3 = request.getParameter("mobile3");
        String zipcode = request.getParameter("zipcode");
        String address1 = request.getParameter("address1");
        String address2 = request.getParameter("address2");
        String phone1 = request.getParameter("phone1");
        String phone2 = request.getParameter("phone2");
        String phone3 = request.getParameter("phone3");
        String emailGet = request.getParameter("emailGet");

        /* MemberBean 인스턴스를 생성하여
         * 회원 가입 폼으로 부터 넘어온 데이터를 저장 한다.
         */
        Member member = new Member();
        member.setId(id);
        member.setName(name);
        member.setPass(pass);
        member.setEmail(emailId + "@" + emailDomain);
        member.setMobile(mobile1 + "-" + mobile2 + "-" + mobile3);
        member.setZipcode(zipcode);
        member.setAddress1(address1);
        member.setAddress2(address2);
    }
}
```



```

if(phone2.equals("") || phone3.equals("")) {
    member.setPhone("");
} else {
    member.setPhone(phone1 + "-" + phone2 + "-" + phone3);
}

```

```

member.setEmailGet(Boolean.valueOf(emailGet));

```

```

/* 회원 가입을 처리하기 위해 MemberDAO 객체를 얻어
 * 회원 테이블에 새로운 회원 정보를 추가 한다.
 */

```

```

MemberDao dao = new MemberDao();
dao.joinMember(member);

```

```

/* 최종적으로 Redirect 정보와 View 페이지 정보를 문자열로 반환하면 된다.

```

```

*
* 회원가입 요청을 처리하고 Redirect 시키지 않으면 사용자가 브라우저를
* 새로 고침 하거나 재요청할 때 마다 이미 가입된 회원정보를 계속 추가하려는
* 동작으로 인해서 중복된 데이터가 저장되거나 또 다른 문제가 발생할 수 있다.
* 이런 경우에는 Redirect 기법을 이용해 DB에 추가, 수정, 삭제하는 동작이 아닌
* 조회하는 곳으로 이동하도록 하면 문제를 해결 할 수 있다.
*
* 현재 요청을 처리한 후에 Redirect 하려면 뷰 페이지를 지정하는 문자열 맨 앞에
* "r:" 또는 "redirect:"를 접두어로 붙여서 반환하고 Redirect가 아니라 Forward
* 하려면 뷰 페이지의 경로만 지정하여 문자열로 반환하면 Controller에서 판단하여
* Redirect 또는 Forward로 연결된다.
*
* 회원가입 폼으로 부터 넘어온 신규 회원정보를 DB에 저장한 후 게시 글 리스트
* 페이지로 이동시키기 위해 View 페이지 정보를 반환할 때 맨 앞에 "r:" 접두어를
* 붙여서 게시 글 리스트 보기 요청을 처리하는 URL를 지정하여 Controller로 넘기면
* Controller는 넘겨받은 View 페이지 정보를 분석하여 Redirect 시키게 된다.
*
* Redirect는 클라이언트 요청에 대한 결과 페이지가 다른 곳으로 이동되었다고
* 브라우저에게 알려주고 그 이동된 주소로 다시 요청하라고 브라우저에게 URL을
* 보내서 브라우저가 그 URL로 다시 응답하도록 처리하는 것으로 아래와 같이
* View 페이지 정보의 맨 앞에 "r:" 또는 "redirect:"를 접두어로 붙여서 반환하면
* Controller에서 View 페이지 정보를 분석해 Redirect 시키고 이 응답을 받은
* 브라우저는 게시 글 리스트를 보여주는 페이지를 다시 요청하게 된다.
*
* 지금과 같이 리다이렉트를 해야 할 경우 웹브라우저가 다시 요청할 주소만 응답하고
* 웹브라우저에서는 이 주소로 재요청하는 동작을 하므로 웹 템플릿 페이지인
* index.jsp를 기준으로 뷰 페이지를 지정하면 안 된다. 왜냐하면 리다이렉트는

```

- * 뷰 페이지를 거쳐서 클라이언트로 응답되는 것이 아니라 현재 클라이언트가 요청한
- * 주소가 다른 곳으로 이동되었다고 알려주기 위해 웹브라우저가 이동할 주소만
- * 응답하고 웹 브라우저는 서버로 부터 응답 받은 주소로 다시 요청하는 동작을 하기
- * 때문에 뷰 페이지의 정보가 아닌 웹 브라우저가 이동할 주소를 지정해야 한다.
- *
- * 회원가입 완료 후 로그인을 위해서 로그인 폼 페이지로 리다이렉트 시킨다.

```

**/
return "r:loginForm.mvc";
}
}

```

3-2) 회원정보 수정하기

앞에서 회원가입을 구현하는 방법에 대해 알아보았다.

이번에는 회원가입 후에 회원정보를 수정하는 기능을 구현하는 방법에 대해 알아볼 것이다. 또한 회원 정보를 수정하는 폼에서 간단히 Ajax 통신을 통해 회원의 비밀번호를 확인 하는 방법에 대해서도 알아 볼 것이다. 그래서 BoardController 외에 Ajax 요청을 전담해 처리하는 AjaxController를 추가로 구현할 것이다.

▶ MemberDao 클래스에 회원정보 수정관련 메서드 추가

MemberDao 클래스에 회원정보 수정 폼에 출력할 회원정보를 읽어오는 메서드와 회원이 수정한 정보를 DB에서 수정하는 다음 메서드를 추가한다.

- com.jspstudy.bbs.dao.MemberDao

// 회원 테이블에 접근하여 요청을 처리하는 DAO(Data Access Object) 클래스

```
public class MemberDao {
```

... 앞부분 코드 생략 ...

// 회원정보 수정 폼에 출력할 회원의 정보를 반환하는 메서드

```
public Member getMember(String id) {
```

```
String selectMember = "SELECT * FROM member WHERE id = ?";
```

```
Member member = null;
```

```
try {
```

```
// DBManager을 이용해 DBCP로 부터 Connection을 대여한다.
```

```
conn = DBManager.getConnection();
```

```
pstmt = conn.prepareStatement(selectMember);
```

```
// loginSql 쿼리의 플레이스홀더(?)에 대응하는 데이터를 설정한다.
```

```
pstmt.setString(1, id);
```

```

// DB에 쿼리를 전송하여 결과를 ResultSet으로 받는다.
rs = pstmt.executeQuery();

/* 회원 id는 Primary Key로 중복되지 않기 때문에 회원 테이블에서 SELECT한
 * 결과가 단일 행으로 반환 되므로 if문을 사용해 rs.next()를 호출했다.
 * id에 해당하는 회원이 없으면 null이 반환된다.
 */
if(rs.next()) {
    member = new Member();
    member.setId(rs.getString("id"));
    member.setName(rs.getString("name"));
    member.setPass(rs.getString("pass"));
    member.setEmail(rs.getString("email"));
    member.setMobile(rs.getString("mobile"));
    member.setZipcode(rs.getString("zipcode"));
    member.setAddress1(rs.getString("address1"));
    member.setAddress2(rs.getString("address2"));
    member.setPhone(rs.getString("phone"));
    member.setEmailGet(rs.getBoolean("email_get"));
    member.setRegDate(rs.getTimestamp("reg_date"));
}
} catch(Exception e) {
    e.printStackTrace();

} finally {
    // DBManager를 이용해 DB 작업에 사용된 자원을 해제 한다.
    DBManager.close(conn, pstmt, rs);
}
return member;
}

```

```

// 회원 정보 수정을 처리하는 메소드
public void updateMember(Member member) {

    String joinSql = "UPDATE member SET name=?, pass=?, email=?, mobile=?"
        + " zipcode=?, address1=?, address2=?, phone=?, email_get=?"
        + " reg_date=SYSDATE WHERE id=?";

    try {
        // DBManager을 이용해 DBCP로 부터 Connection을 대여한다.
        conn = DBManager.getConnection();

        pstmt = conn.prepareStatement(joinSql);
    }
}

```

```

// joinSql 쿼리의 플레이스홀더(?)에 대응하는 데이터를 설정한다.
pstmt.setString(1, member.getName());
pstmt.setString(2, member.getPass());
pstmt.setString(3, member.getEmail());
pstmt.setString(4, member.getMobile());
pstmt.setString(5, member.getZipcode());
pstmt.setString(6, member.getAddress1());
pstmt.setString(7, member.getAddress2());
pstmt.setString(8, member.getPhone());
pstmt.setBoolean(9, member.isEmailGet());
pstmt.setString(10, member.getId());

// DB에 쿼리를 전송하여 회원 가입을 완료한다.
pstmt.executeUpdate();

} catch(Exception e) {
    e.printStackTrace();

} finally {

    // DBManager를 이용해 DB 작업에 사용된 자원을 해제 한다.
    DBManager.close(conn, pstmt);
}
}
}

```

▶ BoardController 클래스에 회원정보 수정관련 요청 처리 추가

이번 예제는 회원정보 수정 폼에서 회원의 기존 비밀번호를 Ajax를 이용해 확인하는 기능을 같이 구현하는데 그래서 BoardController에 회원정보 수정 요청을 처리하는 코드외에 Ajax 요청을 전달해 처리할 AjaxController 클래스가 추가되었다.

- com.jspstudy.bbs.controller.BoardController

```

// 게시판 및 회원관련 요청을 처리하는 프런트 컨트롤러
public class BoardController extends HttpServlet {

```

... 앞부분 코드 생략 ...

아래 부분에서 사용되는 Service 클래스를 CommandProcess 인터페이스를 상속하여 클래스만 먼저 만들어 놓고 교안의 순서대로 하나씩 세부적으로 구현하면 편리함

```

} else if(command.equals("/memberUpdateForm.mvc")) {

    /* 로그아웃이 클릭된 경우의 처리
    * 회원 로그아웃 요청을 처리하는 MemberUpdateFormService 클래스의
    * 인스턴스를 생성한 후 Request와 Response 객체를 매개변수로

```

```

        * requestProcess()를 호출하여 회원 로그아웃 요청을 처리한다.
        */
        service = new MemberUpdateFormService();
        viewPage = service.requestProcess(request, response);

    } else if(command.equals("/memberUpdateResult.mvc")) {

        /* 로그아웃이 클릭된 경우의 처리
        * 회원 로그아웃 요청을 처리하는 MemberUpdateResultService 클래스의
        * 인스턴스를 생성한 후 Request와 Response 객체를 매개변수로
        * requestProcess()를 호출하여 회원 로그아웃 요청을 처리한다.
        */
        service = new MemberUpdateResultService();
        viewPage = service.requestProcess(request, response);

    }

```

... 뒷부분 코드 생략 ...

```

}

```

▶ 회원정보 수정 폼 요청을 처리할 모델 클래스

- com.jspstudy.bbs.service.MemberUpdateFormService

// 회원정보 수정 폼 요청을 처리하는 모델 클래스

```

public class MemberUpdateFormService implements CommandProcess {

    public String requestProcess(
        HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        /* Request 객체로 부터 세션 객체를 얻어 회원 로그인 또는
        * 회원 가입시 세션 영역의 속성에 저장된 회원의 id를 읽어온다.
        */
        HttpSession session = request.getSession();
        String id = (String) session.getAttribute("id");

        /* 회원 정보를 테이블로 부터 읽어오기 위해 MemberDao 객체를 얻어
        * 회원 테이블에서 id에 해당하는 회원 정보를 읽어온다.
        */
        MemberDao dao = new MemberDao();
        Member member = dao.getMember(id);

        // Request 영역의 속성에 테이블로 부터 읽어온 회원 정보를 저장 한다.

```

```
session.setAttribute("member", member);
```

```
/* 최종적으로 Redirect 정보와 View 페이지 정보를 문자열로 반환하면 된다.
```

```
*
```

```
* 회원정보 수정 폼 요청에 대한 결과(모델)를 request 영역의 속성에 저장하고
```

```
* 요청에 대한 결과(모델)를 출력할 View 페이지와 View 페이지를 호출하는 방식을
```

```
* 아래와 같이 문자열로 지정하면 된다. 현재 요청을 처리한 후에 Redirect 하려면
```

```
* 뷰 페이지를 지정하는 문자열 맨 앞에 "r:" 또는 "redirect:"를 접두어로 붙여서
```

```
* 반환하고 Redirect가 아니라 Forward 하려면 뷰 페이지의 경로만 지정하여
```

```
* 문자열로 반환하면 Controller에서 판단하여 Redirect 또는 Forward로 연결된다.
```

```
* 또한 Forward 할 때 뷰 페이지의 정보 중에서 앞부분과 뒷부분에서 중복되는
```

```
* 정보를 줄이기 위해서 Controller에서 PREFIX와 SUFFIX를 지정해 사용하기
```

```
* 때문에 매번 중복되는 부분을 제외하고 뷰 페이지의 정보를 지정하면 된다.
```

```
*
```

```
* 웹 템플릿을 적용하여 뷰를 만드는 경우 Controller에서 PREFIX에 웹 템플릿의
```

```
* 위치가 지정되어 있으므로 PREFIX와 SUFFIX를 제외하고 뷰의 정보를 지정하면
```

```
* 되지만 만약 웹 템플릿을 적용하지 않고 별도로 뷰를 만드는 경우에는 Forward
```

```
* 할 때 PREFIX가 적용되지 않도록 Controller에 알려주기 위해서 아래 주석으로
```

```
* 처리한 return 문과 같이 뷰 페이지 정보를 지정하는 문자열의 맨 앞에 "f:" 또는
```

```
* "forward:"를 접두어로 붙여서 반환하면 된다.
```

```
**/
```

```
// return "f:/WEB-INF/member/overlapidCheck.jsp";
```

```
return "member/memberUpdateForm";
```

```
}
```

```
}
```

▶ 회원정보 수정 폼 요청 결과를 표시할 뷰 페이지

- webapp/WEB-INF/member/memberUpdateForm.jsp

```
<!-- 회원가입 폼 요청 처리 결과를 출력할 View 페이지 -->
```

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

```
pageEncoding="UTF-8"%>
```

```
<!--
```

새로운 5자리 우편번호로 회원 주소를 입력 받기 위해 daum.net에서

제공하는 우편번호 찾기 API를 사용하였다.

참고 사이트 : <http://postcode.map.daum.net/guide>

```
-->
```

```
<script src=
```

```
"https://t1.daumcdn.net/mapjsapi/bundle/postcode/prod/postcode.v2.js"></script>
```

```
<div class="row my-5" id="global-content">
```

```
<div class="col">
```

```
<div class="row my-3 text-center">
```

```
<div class="col">
```

```
<h2 class="fs-3 fw-bold">회원 정보 수정</h2>
```

```
</div>
```

```

</div>
<form action="memberUpdateResult.mvc" name="memberUpdateForm"
method="post" id="memberUpdateForm">
<div class="row mt-5 mb-3">
<div class="col-8 offset-2">
<label for="name" class="form-label">* 이 름 : </label>
<input type="text" class="form-control" name="name" id="name"
value="{sessionScope.member.name}" readonly>
</div>
</div>
<div class="row my-3">
<div class="col-8 offset-2">
<label for="id" class="form-label">* 아이디 : </label>
<div class="row">
<div class="col">
<input type="text" class="form-control" name="id" id="userId"
value="{sessionScope.member.id}" readonly>
</div>
</div>
</div>
<div class="row my-3">
<div class="col-8 offset-2">
<label for="pass1" class="form-label">* 기존 비밀번호 : </label>
<div class="row">
<div class="col-6">
<input type="password" class="form-control" name="oldPass" id="oldPass">
</div>
<div class="col-4">
<input type="button" class="btn btn-warning" id="btnPassCheck" value="비밀
번호 확인">
</div>
</div>
</div>
<div class="row my-3">
<div class="col-8 offset-2">
<label for="pass1" class="form-label">* 새 비밀번호 : </label>
<input type="password" class="form-control" name="pass1" id="pass1">
</div>
</div>
<div class="row my-3">
<div class="col-8 offset-2">
<label for="pass2" class="form-label">* 새 비밀번호 확인 : </label>
<input type="password" class="form-control" name="pass2" id="pass2">

```

```

        </div>
    </div>
    <div class="row my-3">
        <div class="col-8 offset-2">
            <label for="zipcode" class="form-label">* 우편번호 : </label>
            <div class="row">
                <div class="col-4">
                    <input type="password" class="form-control" name="zipcode" id="zipcode"
                        maxlength="5" readonly value="${sessionScope.member.zipcode}">
                </div>
                <div class="col-4">
                    <input type="button" class="btn btn-warning" id="btnZipcode" value="우편번호
찾기">
                </div>
            </div>
        </div>
    </div>
    <div class="row my-3">
        <div class="col-8 offset-2">
            <label for="address1" class="form-label">* 자택주소 : </label>
            <input type="text" class="form-control" name="address1" id="address1"
                readonly value="${sessionScope.member.address1}">
        </div>
    </div>
    <div class="row my-3">
        <div class="col-8 offset-2">
            <label for="address2" class="form-label">상세주소 : </label>
            <input type="text" class="form-control" name="address2" id="address2"
                value="${sessionScope.member.address2}">
        </div>
    </div>
    <div class="row my-3">
        <div class="col-8 offset-2">
            <label for="emailId" class="form-label">* 이 메 일 : </label>
            <div class="row">
                <div class="col-md-4">
                    <input type="text" class="form-control" name="emailId" id="emailId"
                        value="${sessionScope.member.email.split('@')[0]}">
                </div> @
                <div class="col-md-4">
                    <input
                        type="text"
                        class="form-control"
                        name="emailDomain"
                        id="emailDomain"
                        value="${sessionScope.member.email.split('@')[1]}">
                </div>
            </div>
        </div>
    </div>

```



```

        <select class="form-select" name="selectDomain" id="selectDomain">
            <option>직접입력</option>
            <option>네이버</option>
            <option>다음</option>
            <option>한메일</option>
            <option>구글</option>
        </select>
    </div>
</div>
</div>
</div>
<div class="row my-3">
    <div class="col-8 offset-2">
        <label for="mobile2" class="form-label">* 휴 대 폰 : </label>
        <div class="row">
            <div class="col-md-3">
                <select class="form-select" name="mobile1" id="mobile1">
                    <option ${ member.mobile.split('-')[0] == 010 ? "selected" : "" }>
                        010</option>
                    <option ${ member.mobile.split('-')[0] == 011 ? "selected" : "" }>
                        011</option>
                    <option ${ member.mobile.split('-')[0] == 016 ? "selected" : "" }>
                        016</option>
                    <option ${ member.mobile.split('-')[0] == 017 ? "selected" : "" }>
                        017</option>
                    <option ${ member.mobile.split('-')[0] == 018 ? "selected" : "" }>
                        018</option>
                    <option ${ member.mobile.split('-')[0] == 019 ? "selected" : "" }>
                        019</option>
                </select>
            </div>-
            <div class="col-md-4">
                <input type="text" class="form-control" name="mobile2" id="mobile2"
maxlength="4"
                value="${ sessionScope.member.mobile.split('-')[1] }">
            </div>-
            <div class="col-md-4">
                <input type="text" class="form-control" name="mobile3" id="mobile3"
maxlength="4"
                value="${ sessionScope.member.mobile.split('-')[2] }">
            </div>
        </div>
    </div>
</div>
<div class="row my-3">

```

```

<div class="col-8 offset-2">
  <label class="form-label">메일 수신여부 : </label>
  <div class="row">
    <div class="col-md-3">
      <div class="form-check">
        <input type="radio" class="form-check-input" name="emailGet" id="emailOk"
          value="true" ${ member.emailGet ? "checked" : "" }>
        <label class="form-check-label" for="emailOk">수신함</label>
      </div>
    </div>
    <div class="col-md-3">
      <div class="form-check">
        <input type="radio" class="form-check-input" name="emailGet" id="emailNo"
          value="false" ${ member.emailGet ? "" : "checked" }>
        <label class="form-check-label" for="emailNo">수신않함</label>
      </div>
    </div>
  </div>
</div>
<div class="row my-3">
  <div class="col-8 offset-2">
    <label for="phone2" class="form-label">자택전화 : </label>
    <div class="row">
      <div class="col-md-3">
        <select class="form-select" name="phone1" id="phone1">
          <option ${ member.phone.split('-')[0] == 02 ? "selected" : "" }>
            02</option>
          <option ${ member.phone.split('-')[0] == 031 ? "selected" : "" }>
            031</option>
          <option ${ member.phone.split('-')[0] == 032 ? "selected" : "" }>
            032</option>
          <option ${ member.phone.split('-')[0] == 033 ? "selected" : "" }>
            033</option>
          <option ${ member.phone.split('-')[0] == 041 ? "selected" : "" }>
            041</option>
          <option ${ member.phone.split('-')[0] == 042 ? "selected" : "" }>
            042</option>
          <option ${ member.phone.split('-')[0] == 043 ? "selected" : "" }>
            043</option>
          <option ${ member.phone.split('-')[0] == 044 ? "selected" : "" }>
            044</option>
          <option ${ member.phone.split('-')[0] == 051 ? "selected" : "" }>
            051</option>
          <option ${ member.phone.split('-')[0] == 052 ? "selected" : "" }>

```

```

052</option>
<option ${ member.phone.split('-')[0] == 053 ? "selected" : "" }>
053</option>
<option ${ member.phone.split('-')[0] == 054 ? "selected" : "" }>
054</option>
<option ${ member.phone.split('-')[0] == 055 ? "selected" : "" }>
055</option>
<option ${ member.phone.split('-')[0] == 061 ? "selected" : "" }>
061</option>
<option ${ member.phone.split('-')[0] == 062 ? "selected" : "" }>
062</option>
<option ${ member.phone.split('-')[0] == 063 ? "selected" : "" }>
063</option>
<option ${ member.phone.split('-')[0] == 064 ? "selected" : "" }>
064</option>
<option ${ member.phone.split('-')[0] == 010 ? "selected" : "" }>
010</option>
<option ${ member.phone.split('-')[0] == 011 ? "selected" : "" }>
011</option>
<option ${ member.phone.split('-')[0] == 016 ? "selected" : "" }>
016</option>
<option ${ member.phone.split('-')[0] == 017 ? "selected" : "" }>
017</option>
<option ${ member.phone.split('-')[0] == 018 ? "selected" : "" }>
018</option>
<option ${ member.phone.split('-')[0] == 019 ? "selected" : "" }>
019</option>
</select>
</div> -
<div class="col-md-4">
<input type="text" class="form-control" name="phone2" id="phone2"
maxlength="4" value="${ member.phone.split('-')[1] }">
</div> -
<div class="col-md-4">
<input type="text" class="form-control" name="phone3" id="phone3"
maxlength="4" value="${ member.phone.split('-')[2] }">
</div>
</div>
</div>
<div class="row mb-3 mt-5">
<div class="col-8 offset-2">
<input type="submit" value="수정하기" class="btn btn-primary">
</div>
</div>

```

```

    </form>
  </div>
</div>

```

▶ 회원정보 수정 폼에서 사용되는 JavaScript

회원정보 수정 폼에서 비밀번호가 맞는지 Ajax 통신을 이용해 처리하는 자바스크립트 코드와 회원정보 수정 폼이 서버로 보낼 때 유효성 검사 등을 처리하는 자바 스크립트 코드를 다음과 같이 추가한다.

- webapp/js/member.js 에 아래 코드를 추가

// DOM이 준비되면 실행될 콜백 함수

```
$(function() {
```

... 앞부분 코드 생략 ...

```

/* 회원정보 수정 폼에서 "비밀번호 확인" 버튼이 클릭될 때 이벤트 처리
 * 회원정보 수정 폼에서 기존 비밀번호가 맞는지 Ajax 통신을 통해 확인한다.
 */
$("#btnPassCheck").click(function() {
    var oldId = $("#userId").val();
    var oldPass = $("#oldPass").val();

    if($.trim(oldPass).length == 0) {
        alert("기존 비밀번호가 입력되지 않았습니다.\n기존 비밀번호를 입력해주세요");
        return false;
    }

    var data = "id=" + oldId + "&pass="+oldPass;
    console.log("data : " + data);

    $.ajax({
        url: "passCheck.ajax",
        type: "get",
        data: data,
        dataType: "json",
        success: function(resData) {
            console.log(resData);
            if(resData.result == -1) {
                alert("존재하지 않는 아이디 입니다.");

            } else if(resData.result == 0) {
                alert("비밀번호가 다릅니다.\n비밀번호를 다시 확인해주세요");
                $("#oldPass").val("").focus();

            } else if(resData.result == 1){

```

```

        alert("비밀번호가 확인되었습니다.\n비밀번호를 수정해주세요");
        $("#btnPassCheck").prop("disabled", true);
        $("#oldPass").prop("readonly", true);
        $("#pass1").focus();
    }
},
"error": function() {
    console.log("error");
}
});
});

// 회원정보 수정 폼에서 수정하기 버튼이 클릭되면 유효성 검사를 하는 함수
$("#memberUpdateForm").on("submit", function() {

    /* 회원정보 수정 폼에서 "비밀번호 확인" 버튼이 disabled 상태가 아니면
    * 기존 비밀번호를 확인하지 않았기 때문에 확인하라는 메시지를 띄운다.
    */
    if(! $("#btnPassCheck").prop("disabled")) {
        alert("기존 비밀번호를 확인해야 비밀번호를 수정할 수 있습니다.\n"
            + "기존 비밀번호를 입력하고 비밀번호 확인 버튼을 클릭해 주세요");
        return false;
    }

    /* joinFormChcek() 함수에서 폼 유효성 검사를 통과하지 못하면
    * false가 반환되기 때문에 그대로 반환하면 폼이 서브밋 되지 않는다.
    */
    return joinFormCheck(false);
});

```

... 뒷부분 코드 생략 ...

▶ Ajax 요청을 처리하는 컨트롤러

게시판과 회원관련 요청을 처리하는 BoardController 클래스는 배포서술자인 web.xml에 서블릿을 등록했지만 Ajax 요청을 전담해 처리할 AjaxController은 @WebServlet 애노테이션을 이용해 서블릿을 등록하였다.

- com.jspstudy.bbs.ajax.AjaxController

```

// ajax 요청을 처리하는 Controller 클래스
@WebServlet(name="ajaxController", urlPatterns="*.ajax")
public class AjaxController extends HttpServlet {

    /* doGet(), doPost()에서 호출하는 메소드
    * 즉 Ajax로 들어오는 GET 방식과 POST방식 모두를 처리할 메서드 이다.
    */

```

- * 컨트롤러는 이 메소드 안에서 브라우저의 요청에 대한 처리를 요청 URL을 분석해
- * 요청을 처리할 모델 클래스를 결정하고 해당 모델 클래스의 객체를 사용해(위임)
- * 클라이언트의 요청을 처리한 후 그 결과를 뷰로 전달해 결과 화면을 만들게 된다.
- * 뷰로 전달된 데이터는 html 형식의 문서에 출력하여 브라우저에게 응답한다.

*/

```
public void doAjax(
    HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    /* 모든 모델 클래스의 슈퍼 인터페이스인 CommandProcess 타입의
     * 변수를 선언 하고 null로 초기화 한다.
     */
    AjaxProcess ajaxAction = null;

    /* 요청 정보를 담고 있는 Request 객체로 부터 요청 URI를 구한다.
     * /JSPStudyMvcBBS03/passCheck.ajax
     */
    String command = request.getRequestURI();

    /* 요청 정보를 담고 있는 Request 객체로 부터 ContextPath를 구하고
     * /JSPStudyMvcBBS03
     *
     * 요청 URI에서 ContextPath를 제외한 요청 명령을 추출 한다.
     * /passCheck.ajax
     */
    command = command.substring(request.getContextPath().length());
    System.out.println("command : " + command);

    /* 요청 URI에서 추출한 명령을 비교해 요청을 처리할 모델 클래스를
     * 결정하여 객체를 생성하고 메서드를 호출해 요청을 처리한다.
     */
    if(command.equals("/passCheck.ajax")) {

        /* 회원정보 수정시 비밀번호 체크 요청이 들어오는 경우 처리
         * 회원 id에 해당하는 pass가 맞는지를 체크해 주는
         * PassCheckAction 클래스의 인스턴스를 생성한 후 Request와
         * Response 객체를 매개변수로 ajaxProcess()를
         * 호출하여 회원 비밀번호가 맞는지에 대한 요청을 처리 한다.
         */
        ajaxAction = new PassCheckAction();
        ajaxAction.ajaxProcess(request, response);

    }
}
```

```

@Override
public void doGet(
    HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    doAjax(request, response);
}

@Override
public void doPost(
    HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    doAjax(request, response);
}
}

```

▶ Ajax 요청을 처리하는 모델 클래스들이 상속받을 슈퍼 인터페이스

- com.jspstudy.bbs.ajax.AjaxProcess

// ajax 요청을 처리하는 모든 모델 클래스들이 상속 받는 슈퍼 인터페이스

```

public interface AjaxProcess {

    public void ajaxProcess(
        HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException;
}

```

▶ 회원 비밀번호 확인 Ajax를 처리하는 모델 클래스

- com.jspstudy.bbs.ajax.PassCheckAction

// 회원정보 수정 폼에서 회원의 비밀번호 확인을 Ajax로 처리하는 모델 클래스

```

public class PassCheckAction implements AjaxProcess {

    @Override
    public void ajaxProcess(
        HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        String id = request.getParameter("id");
        String pass = request.getParameter("pass");

        //response.setContentType("text/html; charset=utf-8");
        response.setContentType("application/json; charset=utf-8");
        PrintWriter out = response.getWriter();
    }
}

```

```

// 응답 데이터에 사용할 StringBuilder 객체 생성
StringBuilder sb = new StringBuilder();

if(id == null || pass == null) {

    sb.append("<script>");
    sb.append(" alert('정상적인 접근이 아닙니다.');"");
    sb.append("</script>");

    /* Response 객체에 연결된 출력 스트림에
     * 자바스크립트 문자열을 출력하고 메서드를 종료한다.
     */
    out.println(sb.toString());
    return;
}

/* id에 해당하는 pass가 맞는지를 처리하기 위해 MemberDao 객체를
 * 구해 회원 테이블의 회원 정보와 비교하여 가입된 회원이 아니면 -1을
 * 로그인 성공 이면 1을 비밀번호가 맞지 않으면 0을 반환 받는다.
 */
MemberDao dao = new MemberDao();
int result = dao.checkMember(id, pass);

/* StringBuilder를 이용해 응답 데이터를 json 타입으로 작성한다.
 * 클라이언트 쪽에서 json 형식의 데이터를 받아 자바스크립트 객체로
 * 다룰수 있도록 StringBuilder의 toString() 메서드를 이용해
 * 객체를 직렬화 해서 json 형식의 문자열로 응답하고 있다.
 *
 * 클라이언트(jQuery)에서 JSON 데이터로 파싱하기 위해서는 JSON 형식의
 * 문자열 데이터를 만들 때 아래와 같이 작성되어야 제대로 해석 할 수 있다.
 *
 * {"속성 이름": "속성의 값"} 와 같이 안쪽의 속성 이름과 속성의
 * 값이 반드시 쌍 따옴표("")로 감싸서 작성해야 제대로 해석할 수 있다.
 */
sb.append("{ \"result\" : \"\" + result + \"\" }");
out.println(sb.toString());
}
}

```

▶ 회원정보 수정 완료 요청을 처리할 모델 클래스

- com.jspstudy.bbs.service.MemberUpdateResultService

// 회원정보 수정 완료 요청을 처리하는 모델 클래스

```
public class MemberUpdateResultService implements CommandProcess {
```



```

public String requestProcess(
    HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // 회원정보 수정 폼으로 부터 전달된 파라미터를 읽어 변수에 저장 한다.
    request.setCharacterEncoding("utf-8");
    String id = request.getParameter("id");
    String name = request.getParameter("name");
    String pass = request.getParameter("pass1");
    String emailId = request.getParameter("emailId");
    String emailDomain = request.getParameter("emailDomain");
    String mobile1 = request.getParameter("mobile1");
    String mobile2 = request.getParameter("mobile2");
    String mobile3 = request.getParameter("mobile3");
    String zipcode = request.getParameter("zipcode");
    String address1 = request.getParameter("address1");
    String address2 = request.getParameter("address2");
    String phone1 = request.getParameter("phone1");
    String phone2 = request.getParameter("phone2");
    String phone3 = request.getParameter("phone3");
    String emailGet = request.getParameter("emailGet");

    /* MemberBean 인스턴스를 생성하여
     * 회원 가입 폼으로 부터 넘어온 데이터를 저장 한다.
     */
    Member member = new Member();
    member.setId(id);
    member.setName(name);
    member.setPass(pass);
    member.setEmail(emailId + "@" + emailDomain);
    member.setMobile(mobile1 + "-" + mobile2 + "-" + mobile3);
    member.setZipcode(zipcode);
    member.setAddress1(address1);
    member.setAddress2(address2);

    if(phone2.equals("") || phone3.equals("")) {
        member.setPhone("");
    } else {
        member.setPhone(phone1 + "-" + phone2 + "-" + phone3);
    }

    member.setEmailGet(Boolean.valueOf(emailGet));
}

```

```

/* 회원 가입을 처리하기 위해 MemberDAO 객체를 얻어
 * 회원 테이블에 수정된 회원 정보를 적용한다.
 */
MemberDao dao = new MemberDao();
dao.updateMember(member);

/* 최종적으로 Redirect 정보와 View 페이지 정보를 문자열로 반환하면 된다.
 *
 * 회원정보 수정 요청을 처리하고 Redirect 시키지 않으면 사용자가 브라우저를
 * 새로 고침 하거나 재요청할 때 마다 이미 수정된 회원정보를 계속 수정하려는
 * 동작으로 인해서 중복된 데이터가 저장되거나 또 다른 문제가 발생할 수 있다.
 * 이런 경우에는 Redirect 기법을 이용해 DB에 추가, 수정, 삭제하는 동작이 아닌
 * 조회하는 곳으로 이동하도록 하면 문제를 해결 할 수 있다.
 *
 * 현재 요청을 처리한 후에 Redirect 하려면 뷰 페이지를 지정하는 문자열 맨 앞에
 * "r:" 또는 "redirect:"를 접두어로 붙여서 반환하고 Redirect가 아니라 Forward
 * 하려면 뷰 페이지의 경로만 지정하여 문자열로 반환하면 Controller에서 판단하여
 * Redirect 또는 Forward로 연결된다.
 *
 * 회원정보 수정 폼으로 부터 넘어온 회원정보를 DB에서 수정한 후 게시 글 리스트
 * 페이지로 이동시키기 위해 View 페이지 정보를 반환할 때 맨 앞에 "r:" 접두어를
 * 붙여서 게시 글 리스트 보기 요청을 처리하는 URL를 지정하여 Controller로 넘기면
 * Controller는 넘겨받은 View 페이지 정보를 분석하여 Redirect 시키게 된다.
 *
 * Redirect는 클라이언트 요청에 대한 결과 페이지가 다른 곳으로 이동되었다고
 * 브라우저에게 알려주고 그 이동된 주소로 다시 요청하라고 브라우저에게 URL을
 * 보내서 브라우저가 그 URL로 다시 응답하도록 처리하는 것으로 아래와 같이
 * View 페이지 정보의 맨 앞에 "r:" 또는 "redirect:"를 접두어로 붙여서 반환하면
 * Controller에서 View 페이지 정보를 분석해 Redirect 시키고 이 응답을 받은
 * 브라우저는 게시 글 리스트를 보여주는 페이지를 다시 요청하게 된다.
 *
 * 지금과 같이 리다이렉트를 해야 할 경우 웹브라우저가 다시 요청할 주소만 응답하고
 * 웹브라우저에서는 이 주소로 재요청하는 동작을 하므로 웹 템플릿 페이지인
 * index.jsp를 기준으로 뷰 페이지를 지정하면 안 된다. 왜냐하면 리다이렉트는
 * 뷰 페이지를 거쳐서 클라이언트로 응답되는 것이 아니라 현재 클라이언트가 요청한
 * 주소가 다른 곳으로 이동되었다고 알려주기 위해 웹브라우저가 이동할 주소만
 * 응답하고 웹 브라우저는 서버로 부터 응답 받은 주소로 다시 요청하는 동작을 하기
 * 때문에 뷰 페이지의 정보가 아닌 웹 브라우저가 이동할 주소를 지정해야 한다.
 */
return "r:boardList.mvc";
}
}

```

3-3) 회원제 게시판 적용

이번에는 게시 글 리스트를 제외한 모든 요청에 대해서 회원으로 가입한 멤버만 접근할 수 있도록 할 것이다. 이렇게 로그인한 회원만 접근할 수 있도록 하려면 회원 로그인이 되어있는지를 체크하는 코드가 필요하다. 이 코드에서 로그인 상태면 요청한 페이지를 화면에 표시하고 로그인 상태가 아니면 “회원 전용 서비스입니다.” 라는 메시지를 띄우고 회원 로그인 폼으로 이동시킬 것이다.

로그인한 회원만 특정 서비스에 접근할 수 있도록 하려면 로그인 상태인지를 확인하는 아래와 같은 코드가 필요하며 이 코드는 게시 글 리스트 보기, 로그인 폼 보기, 회원가입 폼 보기 등의 로그인이 필요하지 않은 요청을 처리하는 서비스 클래스를 제외한 로그인 서비스가 필요한 모든 서비스 클래스의 requestProcess() 메서드 맨 위에 다음과 같은 코드를 추가해야 한다. 완성된 소스에는 게시 글 리스트에서 클릭해서 접근할 수 있는 BoardListService와 WriteFormService 클래스에만 아래와 같이 로그인을 체크하는 코드를 추가했으니 나머지는 필요한 서비스 클래스에 아래 코드를 추가하여 테스트 해 보자.

```
public String requestProcess(
    HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    /* 회원 전용 서비스를 위해서 현재 접속(Session)이 로그인 상태인지
     * 확인하기 위해서 request 객체로 부터 HttpSession 객체를 구한다.
     * HttpSession 객체를 구하면 사용자가 접속한 현재 세션 정보에 접근할 수 있다.
     * 게시 글 리스트 요청을 처리하는 BoardListService 클래스를 제외하고 모든
     * 모델 클래스에 아래와 같이 코드를 추가하면 된다.
     */
    HttpSession session = request.getSession();

    /* 세션 영역에 isLogin이 존재하지 않으면 NullPointerException이
     * 발생하기 때문에 먼저 null인지를 체크하고 형 변환 했다.
     */
    boolean isLogin = session.getAttribute("isLogin") != null ?
        (Boolean) session.getAttribute("isLogin") : false;

    // 로그인 상태가 아니면 알림 창을 띄우고 회원 로그인 폼으로 보낸다.
    if(! isLogin) {
        /* 스트림에 직접 쓰기위해 응답 객체로 부터 스트림을 구한다.
         * 응답 객체의 스트림을 구하기 전제 ContentType이 설정되어야 한다.
         * 그렇지 않으면 한글과 같은 데이터는 깨져서 출력된다.
         */
        response.setContentType("text/html; charset=utf-8");
        PrintWriter out = response.getWriter();

        out.println("<script>");
        out.println(" alert('회원 전용 서비스입니다.\\n회원 로그인을 해주세요');");
        out.println(" location.href='loginForm.mvc';");
        out.println("</script>");
```

```
/* viewPage 정보가 null 이면 컨트롤러에서 뷰를 거치지 않고
 * 그대로 응답되기 때문에 자바스크립트 구문이 클라이언트로 응답된다.
 **/
return null;
}

... 중략 ...
각 서비스 클래스의 requestProcess() 메서드 안에 기존 코드 위쪽에 작성
}
```

4) 게시 글 상세 페이지 추천/댓글 및 댓글 Ajax 구현

- 실습용 프로젝트 : JSPClassMvcBBS04
- 완성 프로젝트 : JSPStudyMvcBBS04

4-1) 추천/댓글 Ajax 구현

이번 예제는 게시 글 상세보기에서 맘에 드는 게시 글을 추천하거나 자료실 게시판에서 좋은 자료를 올려서 고맙다는 의사를 업로더에게 표시할 수 있는 댓글 기능을 추가해 보자.

사용자가 게시 글 상세보기에서 추천이나 댓글을 클릭했을 때 게시 글 상세보기 페이지 전체를 새롭게 갱신하는 것보다 추천이나 댓글이 표시되는 부분만 갱신하는 것이 네트워크 트래픽도 줄이고 페이지 이동 없이 서비스 할 수 있다는 장점이 있다. 그러므로 이번에 구현할 추천/댓글 기능과 바로 뒤이어서 구현하게 될 댓글 기능은 비동기통신 방식인 Ajax(Asynchronous Javascript And XML) 기술을 이용해 구현할 것이다.

Ajax는 비동기식 자바스크립트와 XML(Asynchronous Javascript And XML)의 약자로 HTML만으로 어려운 작업을 자바스크립트를 사용해 구현하고 사용자와 웹페이지가 상호 작용을 할 수 있도록 도와주는 기술이다. Ajax는 화면에 보이는 전체 웹페이지를 다시 로딩 할 필요 없이 화면에서 필요한 부분만 다시 갱신하면 되므로 가볍고 속도 또한 빠르다.

참고로 이 예제는 자바 객체의 직렬화를 위해서 Gson 라이브러리를 사용하므로 gson-2.10.1.jar 라이브러리가 클래스 패스에 있어야 한다. 아래를 참고해 gson 라이브러리를 다운로드를 받아 /WEB-INF/lib 폴더에 추가하도록 하자.

◆ gson 라이브러리 다운로드 및 참고 사이트

- gson 다운로드 : <https://github.com/google/gson>
- 참고사이트 :
<https://galid1.tistory.com/501>
<https://hianna.tistory.com/629>

▶ 게시 글 상세보기에 댓글 출력하기

추천/댓글 기능을 Ajax로 구현하기 위해서 앞에서 구현한 게시 글 상세보기 화면을 수정할 것이다. 이 게시 글 상세보기 화면에서 추천/댓글 기능과 댓글 기능을 제공해야 되므로 앞에서 작성한 게시 글 상세보기 화면에 추천/댓글 버튼을 추가하고 댓글 리스트를 출력하는 기능을 먼저 구현한 후에 Ajax를 이용한 추천/댓글 기능을 구현할 것이다. 게시 글 상세보기 화면을 구성할 때 게시 글의 추천/댓글 수도 출력해야 하므로 앞에서 사용하던 게시 글 정보를 저장하는 테이블인 jspbbbs 테이블에 추천, 댓글 수를 저장할 수 있는 컬럼이 추가되었기 때문에 Board 클래스에 추천/댓글을 저장할 수 있는 인스턴스 변수가 추가되었다. 또한 하나의 게시 글에 여러 개의 댓글이 존재할 수 있으므로 게시 글의 댓글 정보를 저장할 테이블인 reply 테이블이 추가되었기 때문에 하나의 댓글 정보를 저장할 수 있는 VO(Value Object) 클래스인 Reply 클래스도 추가되었다.

먼저 JSPStudyMvcBBS04 프로젝트의 webapp/WEB-INF/sql/JSPStudyBBS.sql 파일의 주석을 참고해서 수정된 jspbbbs 테이블과 이번에 새롭게 추가되는 reply 테이블을 생성하고 게시 글 정보와 댓글 정보를 추가한 후에 다음 코드를 참고해서 하나의 게시 글 정보를 저장하는 Board 클래스와 하나의 댓글 정보를 저장하는 Reply 클래스를 작성하자.

◆ 하나의 게시 글 정보를 저장하는 VO(Value Object) 클래스

하나의 게시 글 정보를 저장하는 VO(Value Object) 클래스인 Board 클래스에 추천 정보와 땡큐 정보를 저장할 인스턴스 변수를 추가하고 getter와 setter도 같이 추가한다. Board 클래스는 앞에서 사용한 Board 클래스에 추천, 땡큐 정보를 저장할 인스턴스 변수가 되므로 코드는 거의 비슷하면 추가되는 부분에 대해서는 빨간색 볼드체로 표시하였다.

- com.jspstudy.bbs.vo.Board

```
/* 하나의 게시 글 정보를 저장하는 클래스(VO, Beans, DTO)
 * VO 객체에 저장될 데이터는 테이블에서 읽어오기 때문에 각각의 변수는
 * 테이블에서 컬럼이 가지는 데이터 형식과 같거나 자동 형 변환이 가능해야 한다.
 */
public class Board {

    private int no;
    private String title;
    private String content;
    private String writer;
    private Timestamp regDate;
    private int readCount;
    private String pass;
    private String file1;
    private int recommend;
    private int thank;

    public Board() { }
    public Board(int no, String title, String writer, String content,
        Timestamp regDate, int readCount, String pass, String file1) {
        this.no = no;
        this.title = title;
        this.writer = writer;
        this.content = content;
        this.regDate = regDate;
        this.readCount = readCount;
        this.pass = pass;
        this.file1 = file1;
    }

    public int getNo() {
        return no;
    }
    public void setNo(int no) {
        this.no = no;
    }
}
```

```

public String getTitle() {
    return title;
}
public void setTitle(String title) {
    this.title = title;
}
public String getWriter() {
    return writer;
}
public void setWriter(String writer) {
    this.writer = writer;
}
public String getContent() {
    return content;
}
public void setContent(String content) {
    this.content = content;
}
public Timestamp getRegDate() {
    return regDate;
}
public void setRegDate(Timestamp regDate) {
    this.regDate = regDate;
}
public int getReadCount() {
    return readCount;
}
public void setReadCount(int readCount) {
    this.readCount = readCount;
}
public String getPass() {
    return pass;
}
public void setPass(String pass) {
    this.pass = pass;
}
public String getFile1() {
    return file1;
}
public void setFile1(String file1) {
    this.file1 = file1;
}
public int getRecommend() {
    return recommend;
}

```

```

public void setRecommend(int recommend) {
    this.recommend = recommend;
}
public int getThank() {
    return thank;
}
public void setThank(int thank) {
    this.thank = thank;
}
}

```

◆ 하나의 댓글 정보를 저장하는 VO(Value Object) 클래스

하나의 댓글 정보를 저장하는 VO(Value Object) 클래스는 이번에 새로 추가되는 것이므로 아래 클래스를 프로젝트에 추가하면 된다.

- com.jspstudy.bbs.vo.Reply

```

/* 하나의 댓글 정보를 저장하는 클래스(VO, Beans, DTO)
 * VO 객체에 저장될 데이터는 테이블에서 읽어오기 때문에 각각의 변수는
 * 테이블에서 컬럼이 가지는 데이터 형식과 같거나 자동 형 변환이 가능해야 한다.
 */
public class Reply {
    private int no;
    private int bbsNo;
    private String replyContent;
    private String replyWriter;
    private Timestamp regDate;

    public Reply() { }
    public Reply(int bbsNo, String replyContent, String replyWriter) {
        this.bbsNo = bbsNo;
        this.replyContent = replyContent;
        this.replyWriter = replyWriter;
    }
    public Reply(int no, int bbsNo, String replyContent,
        String replyWriter, Timestamp regDate) {

        this.no = no;
        this.bbsNo = bbsNo;
        this.replyContent = replyContent;
        this.replyWriter = replyWriter;
        this.regDate = regDate;
    }
    public int getNo() {
        return no;
    }
}

```



```

}
public void setNo(int no) {
    this.no = no;
}
public int getBbsNo() {
    return bbsNo;
}
public void setBbsNo(int bbsNo) {
    this.bbsNo = bbsNo;
}
public String getReplyContent() {
    return replyContent;
}
public void setReplyContent(String replyContent) {
    this.replyContent = replyContent;
}
public String getReplyWriter() {
    return replyWriter;
}
public void setReplyWriter(String replyWriter) {
    this.replyWriter = replyWriter;
}
public Timestamp getRegDate() {
    return regDate;
}
public void setRegDate(Timestamp regDate) {
    this.regDate = regDate;
}
}

```

◆ BoardDao 클래스의 getBoard() 메서드 수정

Board 테이블에 추천, 땡큐 수를 저장하는 2개의 컬럼이 추가되어 VO 클래스의 속성이 2개 추가되었으므로 게시 글 상세보기 요청처리에서 사용하는 getBoard() 메서드에서도 이 두 개의 정보를 저장하는 코드가 추가되어야 한다. 이미 앞에서 구현한 메서드에 일부 코드만 추가하면 되는 것이므로 아래 코드에서 빨간색 볼드체로 표시한 두 줄의 코드만 추가하면 된다.

- com.jspstudy.bbs.dao.BoardDao

```

public Board getBoard(int no, boolean state) {
    String boardSql = "SELECT * FROM jspbbs WHERE no=?";
    String countSql = "UPDATE jspbbs set read_count = read_count + 1 "
        + "WHERE no = ?";
    Board board = null;

    try{

```

```

// 1. DBManager을 이용해 DBCP로 부터 Connection을 대여한다.
conn = DBManager.getConnection();

// 활성화된 Connection에 트랜잭션을 시작한다.
DBManager.setAutoCommit(conn, false);

// 게시 글 조회 요청일 때 state는 true로 게시 글 조회 수를 1증가 시킨다.
if(state) {

    /* 2. DBMS에 SQL 쿼리를 발생하기 위해 활성화된
     * Connection 객체로 부터 PreparedStatement 객체를 얻는다.
     */
    pstmt = conn.prepareStatement(countSql);

    /* 3. PreparedStatement 객체의 Placeholder(?)에 대응하는
     * 값을 순서에 맞게 지정하고 있다.
     */
    pstmt.setInt(1, no);

    /* 4. 데이터베이스에 UPDATE 쿼리를 발행해 조회수를 1증가 시킨다.
     *
     * executeUpdate()는 DBMS에 INSERT, UPDATE, DELETE 쿼리를
     * 발행하는 메소드로 추가, 수정, 삭제된 레코드의 개수를 반환 한다.
     */
    pstmt.executeUpdate();
}

/* 2. DBMS에 SQL 쿼리를 발생하기 위해 활성화된
 * Connection 객체로 부터 PreparedStatement 객체를 얻는다.
 */
pstmt = conn.prepareStatement(boardSql);

/* 3. PreparedStatement 객체의 Placeholder(?)에 대응하는
 * 값을 순서에 맞게 지정하고 있다.
 */
pstmt.setInt(1, no);

/* 4. PreparedStatement를 이용해 SELECT 쿼리를 발행한다.
 *
 * executeQuery()는 실제 DBMS에 SELECT 쿼리를 발행하는 메소드로
 * DB에서 검색된 데이터를 가상의 테이블 형태인 ResultSet 객체로 반환 한다.
 */
rs = pstmt.executeQuery();

```

```

/* 5. 쿼리 실행 결과를 바탕으로 요청한 게시 글 정보를 구한다.
 *
 * ResultSet 객체는 DB로 부터 읽어온 데이터에 접근하기 위해 테이블의
 * 행을 가리키는 cursor를 제공한다. 맨 처음 ResultSet 객체를 반환
 * 받으면 cursor는 첫 번째 행 바로 이전을 가리키고 있다.
 *
 * 테이블의 PRIMARY KEY인 no에 해당하는 게시 글을 SELECT 해서
 * ResultSet에는 게시 글 하나의 정보만 존재하기 때문에 if 문을 사용했다.
 */
if(rs.next()) {
    board = new Board();
    board.setNo(rs.getInt("no"));
    board.setTitle(rs.getString("title"));
    board.setContent(rs.getString("content"));
    board.setWriter(rs.getString("writer"));
    board.setRegDate(rs.getTimestamp("reg_date"));
    board.setReadCount(rs.getInt("read_count"));
    board.setPass(rs.getString("pass"));
    board.setFile1(rs.getString("file1"));
    board.setRecommend(rs.getInt("recommend"));
    board.setThank(rs.getInt("thank"));
}

// 모든 작업이 완료되면 커밋하여 트랜잭션을 종료한다.
DBManager.commit(conn);

} catch(Exception e) {
    // DB 작업이 하나라도 에러가 발생하면 롤백하고 트랜잭션을 종료한다.
    DBManager.rollback(conn);

    System.out.println("BoardDao - getBoard(no, state)");
    e.printStackTrace();
} finally {
    // 6. DBManager를 이용해 Connection을 DBCP에 반납한다.
    DBManager.close(conn, pstmt, rs);
}

// 요청한 하나의 게시 글을 반환 한다.
return board;
}

```

◆ BoardDao 클래스에 댓글 리스트를 DB로부터 읽어와 반환하는 메서드 추가

게시 글 상세보기의 아래쪽 부분에 출력할 댓글 리스트를 DB에서 읽어와 반환하는 getReplyList() 메서드를 BoardDao 클래스에 새로 추가하면 된다.

- com.jspstudy.bbs.dao.BoardDao

// 특정 게시 글에 해당하는 댓글 리스트를 반환하는 메소드

```
public ArrayList<Reply> getReplyList(int bbsNo) {

    String replyListSql = "SELECT * FROM reply WHERE bbs_no = ?"
        + " ORDER BY no DESC";

    Reply reply = null;
    ArrayList<Reply> replyList = null;

    try {
        // DBManager을 이용해 DBCP로 부터 Connection을 대여한다.
        conn = DBManager.getConnection();

        // Comment 테이블에 저장된 댓글 번호중 제일 큰 댓글 번호를 읽어온다.
        pstmt = conn.prepareStatement(replyListSql);
        pstmt.setInt(1, bbsNo);
        rs = pstmt.executeQuery();

        replyList = new ArrayList<Reply>();

        while(rs.next()) {

            reply = new Reply();
            reply.setNo(rs.getInt("no"));
            reply.setBbsNo(rs.getInt("bbs_no"));
            reply.setReplyContent(rs.getString("reply_content"));
            reply.setReplyWriter(rs.getString("reply_writer"));
            reply.setRegDate(rs.getTimestamp("reg_date"));
            replyList.add(reply);
        }
    } catch(Exception e) {
        System.out.println("BoardDao - replyList(no)");
        e.printStackTrace();
    } finally {
        // DBManager를 이용해 DB 작업에 사용된 자원을 해제 한다.
        DBManager.close(conn, pstmt, rs);
    }
    return replyList;
}
```

◆ 게시 글 상세보기 요청을 처리할 모델 클래스에 댓글 리스트 처리 추가

BoarDetailService 클래스도 이미 앞에서 구현한 코드에 댓글 리스트를 출력하기 위한 코드를 추가하는 것이므로 아래 코드에서 빨간색 볼드체로 표시한 두 줄의 코드만 추가하면 현재 게시 글에 해

당하는 댓글 리스트를 DB에서 읽어와 모델에 추가하면 된다.

- com.jspstudy.bbs.service.BoardDetailService

// 게시 글 상세보기 요청을 처리하는 서비스 클래스

```
public class BoardDetailService implements CommandProcess {

    public String requestProcess(
        HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        /* 회원 전용 서비스를 위해서 현재 접속(Session)이 로그인 상태인지
        * 확인하기 위해서 request 객체로 부터 HttpSession 객체를 구한다.
        * HttpSession 객체를 구하면 사용자가 접속한 현재 세션 정보에 접근할 수 있다.
        * 게시 글 리스트 요청을 처리하는 BoardListService 클래스를 제외하고 모든
        * 모델 클래스에 아래와 같이 코드를 추가하면 된다.
        */
        HttpSession session = request.getSession();

        /* 세션 영역에 isLogin이 존재하지 않으면 NullPointerException이
        * 발생하기 때문에 먼저 null인지를 체크하고 형 변환 했다.
        */
        boolean isLogin = session.getAttribute("isLogin") != null ?
            (Boolean) session.getAttribute("isLogin") : false;

        // 로그인 상태가 아니면 알림 창을 띄우고 이전으로 돌려보낸다.
        if(! isLogin) {
            /* 스트림에 직접 쓰기위해 응답 객체로 부터 스트림을 구한다.
            * 응답 객체의 스트림을 구하기 전제 ContentType이 설정되어야 한다.
            * 그렇지 않으면 한글과 같은 데이터는 깨져서 출력된다.
            */
            response.setContentType("text/html; charset=utf-8");
            PrintWriter out = response.getWriter();

            out.println("<script>");
            out.println(" alert('회원 전용 서비스입니다.\n회원 로그인을 해 주세요');");
            out.println(" history.back();");
            out.println("</script>");

            /* viewPage 정보가 null 이면 컨트롤러에서 뷰를 거치지 않고
            * 그대로 응답되기 때문에 자바스크립트 구문이 클라이언트로 응답된다.
            */
            return null;
        }

        //요청 파라미터로 넘어 온 게시 글 번호와 페이지 번호를 읽어온다.
```

```

String no = request.getParameter("no");
String pageNum = request.getParameter("pageNum");
String type = request.getParameter("type");
String keyword = request.getParameter("keyword");

// no와 pageNum이 비어 있으면 비정상 요청임
if(no == null || no.equals("") || pageNum == null || pageNum.equals("")) {

    /* 스트림에 직접 쓰기위해 응답 객체로 부터 스트림을 구한다.
     * 응답 객체의 스트림을 구하기 전제 ContentType이 설정되어야 한다.
     * 그렇지 않으면 한글과 같은 데이터는 깨져서 출력된다.
     */
    response.setContentType("text/html; charset=utf-8");
    PrintWriter out = response.getWriter();

    out.println("<script>");
    out.println(" alert('정상적인 접근이 아닙니다.');

```

```

* 게시 글 하나의 내용, 현재 페이지 번호, 검색 여부, 댓글 리스트를 속성에 저장 했다.
**/
request.setAttribute("board", board);
request.setAttribute("replyList", replyList);
request.setAttribute("pageNum", pageNum);
request.setAttribute("searchOption", searchOption);

// 검색 요청이면 type과 keyword를 request 영역에 저장한다.
if(searchOption) {
    request.setAttribute("type", type);
    request.setAttribute("keyword", keyword);
}

/* 최종적으로 Redirect 정보와 View 페이지 정보를 문자열로 반환하면 된다.
*
* 게시 글 상세보기 요청에 대한 결과(모델)를 request 영역의 속성에 저장하고
* 요청에 대한 결과(모델)를 출력할 View 페이지와 View 페이지를 호출하는 방식을
* 아래와 같이 문자열로 지정하면 된다. 현재 요청을 처리한 후에 Redirect 하려면
* 뷰 페이지를 지정하는 문자열 맨 앞에 "r:" 또는 "redirect:"를 접두어로 붙여서
* 반환하고 Redirect가 아니라 Forward 하려면 뷰 페이지의 경로만 지정하여
* 문자열로 반환하면 Controller에서 판단하여 Redirect 또는 Forward로 연결된다.
* 또한 Forward 할 때 뷰 페이지의 정보 중에서 앞부분과 뒷부분에서 중복되는
* 정보를 줄이기 위해서 Controller에서 PREFIX와 SUFFIX를 지정해 사용하기
* 때문에 매번 중복되는 부분을 제외하고 뷰 페이지의 정보를 지정하면 된다.
*
* 웹 템플릿을 적용하여 뷰를 만드는 경우 Controller에서 PREFIX에 웹 템플릿의
* 위치가 지정되어 있으므로 PREFIX와 SUFFIX를 제외하고 뷰의 정보를 지정하면
* 되지만 만약 웹 템플릿을 적용하지 않고 별도로 뷰를 만드는 경우에는 Forward
* 할 때 PREFIX가 적용되지 않도록 Controller에 알려주기 위해서 아래 주석으로
* 처리한 return 문과 같이 뷰 페이지 정보를 지정하는 문자열의 맨 앞에 "f:" 또는
* "forward:"를 접두어로 붙여서 반환하면 된다.
**/
// return "f:/WEB-INF/member/overlapidCheck.jsp";
return "board/boardDetail";
}
}

```

◆ 게시 글 상세보기 요청을 처리할 컨트롤러

게시 글 상세보기 요청은 앞에서 구현한 BoardController 클래스를 수정 없이 그대로 사용하면 된다.

◆ 게시 글 상세보기 결과를 출력할 뷰 페이지

아래 코드에서 빨간색 볼드체로 표시한 부분이 추가되거나 수정되는 부분이다.

- WEB-INF/board/boardDetail.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<script src="js/reply.js"></script>
<!-- content -->
<div class="row my-5" id="global-content">
    <div class="col">
        <form name="checkForm" id="checkForm">
            <input type="hidden" name="no" id="no" value="${ board.no }"/>
            <input type="hidden" name="pass" id="rPass" />
            <input type="hidden" name="pageNum" value="${ pageNum }" />

            <%--
                검색 리스트에서 들어온 요청일 경우 다시 keyword에 해당하는
                검색 리스트로 돌려보내기 위해서 아래의 파라미터가 필요하다.
            --%>
            <c:if test="${ searchOption }">
                <input type="hidden" name="type" value="${ type }" />
                <input type="hidden" name="keyword" value="${ keyword }" />
            </c:if>
        </form>
        <div class="row text-center">
            <div class="col">
                <h2 class="fs-3 fw-bold">게시 글 상세보기</h2>
            </div>
        </div>

        <!-- 게시 글 상세보기 영역 -->
        <div class="row my-3">
            <div class="col">
                <table class="table table-bordered" >
                    <tbody>
                        <tr>
                            <th class="table-secondary">제 목</th>
                            <td colspan="3">${ board.title }</td>
                        </tr>
                        <tr>
                            <th>글쓴이</th>
                            <td>${ board.writer }</td>
                            <th>작성일</th>
                            <td><fmt:formatDate value="${ board.regDate }"
                                pattern="yyyy-MM-dd HH:mm:ss" /></td>
                        </tr>
                    </tbody>
                </table>
            </div>
        </div>
    </div>
</div>
```



```

</c:if>
<!--
    검색 리스트에서 온 요청이면 검색 리스트의 동일한 페이지로 돌려보낸다.
--%>
<c:if test="${ searchOption }">
    &nbsp;&nbsp;&nbsp;<input class="btn btn-primary" type="button" value="목록보기"
        onclick="location.href='boardList.mvc?pageNum=${pageNum}&type=${
type
}&keyword=${ keyword }'"/>
</c:if>
</div>
</div>

```

<!--

아래부터 추천/땡큐와 댓글 기능을 처리하는 부분으로

기존의 게시 글 상세보기 뷰 페이지의 아래쪽에 다음의 코드를 추가하면 된다.

-->

<!-- 추천/땡큐 영역 -->

<div class="row my-5">

<div class="col border p-3">

<div id="recommend" class="text-end">

 추천

({ board.recommend })

 |

 땡큐

({ board.thank })

 |

<i class="bi bi-file-earmark-text-fill" style="color: cornflowerblue;"></i> 댓글쓰

기

</div>

</div>

</div>

<!-- 댓글 헤더 영역 -->

<div class="row" id="replyTitle">

<div class="col p-2 text-center bg-dark text-white">

<h3 class="fs-4">이 글에 대한 댓글 리스트</h3>

</div>

</div>

<!-- 댓글 리스트 영역 -->

<!-- 댓글이 존재하는 경우 -->

```

<c:if test="${ not empty replyList }" >
<div class="row mb-3">
  <div class="col" id="replyList">
    <c:forEach var="reply" items="${ replyList }" >
      <div class="replyRow row border border-top-0">
        <div class="col">
          <div class="row bg-light p-2">
            <div class="col-4">
              <span>${ reply.replyWriter }</span>
            </div>
            <div class="col-8 text-end">
              <span class="me-3">
                <fmt:formatDate value="${ reply.regDate}" pattern="yyyy-MM-dd
HH:mm:ss" />
              </span>
              <button class="modifyReply btn btn-outline-success btn-sm" data-no="${
reply.no }">
                <i class="bi bi-journal-text">수정</i>
              </button>
              <button class="deleteReply btn btn-outline-warning btn-sm" data-no="${
reply.no }">
                <i class="bi bi-trash">삭제</i>
              </button>
              <button class="btn btn-outline-danger btn-sm" onclick="reportReply('${
reply.no }')">
                <i class="bi bi-telephone-outbound">신고</i>
              </button>
            </div>
          </div>
        </div>
        <div class="row">
          <div class="col p-3">
            <pre>${ reply.replyContent }</pre>
          </div>
        </div>
      </div>
    </c:forEach>
  </div>
</div>
</c:if>

<!-- 댓글이 존재하지 않는 경우 -->
<c:if test="${ empty replyList }" >
<div class="row mb-5" id="replyList">
  <div class="col text-center border p-5">

```

```

        <div>이 게시 글에 대한 댓글이 존재하지 않습니다.</div>
    </div>
</div>
</c:if>

<!-- 댓글 쓰기 폼 -->
<div class="row my-3 d-none" id="replyForm">
    <div class="col">
        <form name="replyWriteForm" id="replyWriteForm">
            <input type="hidden" name="bbsNo" value="{ board.no }"/>
            <input type="hidden" name="replyWriter" value="{ sessionScope.id }" />
            <div class="row bg-light my-3 p-3 border">
                <div class="col">
                    <div class="row">
                        <div class="col text-center">
                            <span>악의적인 댓글은 예고 없이 삭제될 수 있으며 글쓰기 제한과 아이디 삭
제 처리됩니다.</span>
                        </div>
                    </div>
                    <div class="row my-3">
                        <div class="col-md-10">
                            <textarea name="replyContent" id="replyContent" class="form-control"
rows="4"></textarea>
                        </div>
                        <div class="col-md">
                            <input type="submit" value="댓글쓰기" class="btn btn-primary h-100
w-100" id="replyWriteButton">
                        </div>
                    </div>
                </div>
            </div>
        </form>
    </div>
</div><!-- end replyForm -->

</div>
</div>

```

▶ 추천과 땡큐 기능 구현

앞에서 회원정보를 수정할 때 페이지 갱신 없이 회원의 기존 비밀번호를 Ajax를 통해 확인하는 예제를 다뤘 보았다. 이 때 사용했던 AjaxController와 요청을 동일한 방식으로 처리하기 위해 사용했던 Ajax를 처리하는 모든 모델 클래스의 슈퍼 인터페이스인 AjaxProcess는 그대로 사용할 것이다. 추천과 땡큐 요청 처리를 위해서 AjaxController 클래스의 if~else if 부분이 추가될 것이다.

◆ BoardDao 클래스에 추천과 땡큐 관련 메서드 추가

BoardDao 추천과 땡큐 수를 추가하고 다시 읽어와 HashMap<String, Integer>으로 반환하는 다음 메서드를 추가한다.

- com.jspstudy.bbs.dao.BoardDao

```
/* 게시 글 상세 보기에서 추천과 땡큐 요청을 처리하는 메소드
 * 추천/땡큐의 횟수를 jspbbs 테이블에서 1 증가 시킨다.
 */
public HashMap<String, Integer> getRecommend(int no, String strThank) {

    HashMap<String, Integer> map = null;

    String addRecommendSql = "UPDATE jspbbs set"
        + " recommend=recommend + 1 WHERE no=?";
    String addThankSql = "UPDATE jspbbs set thank=thank + 1 WHERE no=?";
    String selectResultSql = "SELECT recommend, thank FROM jspbbs WHERE no=?";

    try {

        // DBManager을 이용해 DBCP로 부터 Connection을 대여한다.
        conn = DBManager.getConnection();

        // 현재 Connection에 트랜잭션을 시작한다.
        DBManager.setAutoCommit(conn, false);

        // 추천 요청이면
        if(strThank.equals("commend")) {
            /* jspbbs 테이블에서 해당하는 게시 글의 추천 수를 1증가 시키는
             * 쿼리를 캐싱하여 PreparedStatement 객체를 얻는다.
             */
            pstmt = conn.prepareStatement(addRecommendSql);

            // 땡큐 요청이면
        } else if(strThank.equals("thank")){
            /* jspbbs 테이블에서 해당하는 게시 글의 땡큐 수를 1증가 시키는
             * 쿼리를 캐싱하여 PreparedStatement 객체를 얻는다.
             */
            pstmt = conn.prepareStatement(addThankSql);
        }

        // jspbbs 테이블에서 해당하는 게시 글의 땡큐 또는 추천 수를 1증가 시킨다.
        pstmt.setInt(1, no);
        pstmt.executeUpdate();
    }
```

```

/* jspbbs 테이블에서 해당하는 게시 글의 땡큐와 추천 수를 읽어오는
 * 쿼리를 캐싱하여 PreparedStatement 객체를 얻는다.
 */
pstmt = conn.prepareStatement(selectResultSql);
pstmt.setInt(1, no);
rs = pstmt.executeQuery();

// 질의해온 추천 수와 땡큐를 HashMap에 저장한다.
if(rs.next()) {
    map = new HashMap<String, Integer>();
    map.put("recommend", rs.getInt(1));
    map.put("thank", rs.getInt(2));
}

// 모든 DB 작업이 완료되면 commit하고 트랜잭션을 종료한다.
DBManager.commit(conn);
} catch(Exception e) {
    // DB 작업이 실패하면 rollback 하고 트랜잭션을 종료한다.
    DBManager.rollback(conn);
    System.out.println("BoardDao - getRecommend(no, isThank)");
    e.printStackTrace();
} finally {
    // DBManager를 이용해 DB 작업에 사용된 자원을 해제 한다.
    DBManager.close(conn, pstmt, rs);
}
return map;
}

```

◆ Ajax 요청을 처리하는 모든 모델 클래스의 슈퍼 인터페이스

Ajax 요청을 처리하는 모든 모델 클래스가 상속 받는 슈퍼 인터페이스는 앞에서 사용했던 인터페이스와 동일한 인터페이스이다.

- com.jspstudy.bbs.ajax.AjaxProcess

```

// ajax 요청을 처리하는 모든 모델 클래스의 슈퍼 인터페이스
public interface AjaxProcess {

    public void ajaxProcess(
        HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException;
}

```

◆ 추천과 땡큐 Ajax 요청을 처리할 모델 클래스

Ajax 요청을 처리하는 모든 모델 클래스는 앞에서 회원가입에서 기존 비밀번호가 맞는지 체크하는 기능을 구현할 때와 마찬가지로 AjaxProcess 인터페이스를 상속 받아 클래스를 작성하면 된다. 참고로 Ajax 요청을 처리하는 모든 모델 클래스 이름에는 Action이라는 단어가 포함되어 있다.

- com.jspstudy.bbs.ajax.RecommendAction

// 추천과 땡큐 요청을 처리하는 모델 클래스

```
public class RecommendAction implements AjaxProcess {

    public void ajaxProcess(
        HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        String recommend = request.getParameter("recommend");
        String no = request.getParameter("no");

        HashMap<String, Integer> map = null;
        BoardDao dao = new BoardDao();
        map = dao.getRecommend(Integer.parseInt(no), recommend);

        // Gson 생성자를 이용해 Gson 객체를 생성한다.
        Gson gson = new Gson();

        /* dao에서 받은 Map 객체를 json 형식으로 직렬화 한다.
         * dao에서 Map에 데이터를 저장할 때 아래와 같이 저장 하였다.
         *
         * map.put("recommend", rs.getInt(1));
         * map.put("thank", rs.getInt(2));
         *
         * 이 데이터를 json 형식으로 직렬화 하면 아래와 같다.
         *
         * {recommend: 10, thank: 5}
         */
        String result = gson.toJson(map);

        System.out.println("RecommendAction - result : " + result);

        /* 응답 데이터를 스트림을 통해 클라이언트에게 전송하기 위해
         * 응답 객체(response)로 스트림을 연결한다.
         *
         * AjaxController는 Ajax 요청만 받는 컨트롤러로 이 컨트롤러에서
         * 현재 클래스의 ajaxProcess() 메서드를 호출하면 요청을 처리하고
         * 컨트롤러로 돌아갈 때 응답 객체에 연결된 스트림을 통해 result에
         * 저장된 데이터가 웹 브라우저의 XMLHttpRequest 객체로 전달된다.
         */
    }
}
```

```

        response.setContentType("text/html; charset=utf-8");
        PrintWriter out = response.getWriter();
        out.println(result);
    }
}

```

◆ Ajax 요청을 처리하는 컨트롤러

추천과 땡큐 기능은 Ajax를 이용해 처리하므로 BoardController는 사용되지 않는다. 앞에서 구현한 AjaxController 클래스의 doAjax() 메서드 안에 Ajax 요청에 따라서 모델 클래스를 호출해 요청을 처리하는 if~else if 문 맨 아래에 다음 코드를 참고해서 추천과 땡큐 요청을 처리하는 코드를 작성하자.

- com.jspstudy.bbs.ajax.AjaxController

```

// ajax 요청을 처리하는 Controller 클래스
@WebServlet(name="ajaxController", urlPatterns="*.ajax")
public class AjaxController extends HttpServlet {

    ... 앞부분 코드 생략...

    ... doAjax() 메서드 안의 요청 처리 if ~ else if 문에 아래 코드 추가 ...

    } else if(command.equals("/recommend.ajax")) {

        /* 추천과 땡큐 요청이 들어오는 경우 처리
         * 게시 글 번호에 해당하는 추천과 땡큐 요청을 처리하는
         * RecommendAction 클래스의 인스턴스를 생성한 후 Request와
         * Response 객체를 매개변수로 ajaxProcess()를
         * 호출하여 추천과 땡큐에 대한 요청을 처리 한다.
         */
        ajaxAction = new RecommendAction();
        ajaxAction.ajaxProcess(request, response);
    }

    ... 뒷부분 코드 생략...

}

```

◆ 추천과 땡큐관련 자바스크립트 작성

추천과 땡큐를 비롯해 댓글 쓰기, 수정하기, 삭제하기 기능을 구현할 댓글관련 자바스크립트 파일인 reply.js 파일을 만들어 아래 코드를 작성하자.

- webapp/js/reply.js

```

// DOM(Document Object Model)이 준비 되었다면
$(document).ready(function() {

```



```

// 추천/땡큐 Ajax
$("#btnCommend").click(function() {

    var com = $(this).attr("id");
    console.log("com : " + com);

    $.ajax({
        url: "recommend.ajax",

        // type을 지정하지 않으면 get 방식 요청이다.
        type: "post",

        // 파라미터로 보낼 데이터를 객체 리터럴로 지정하고 있다.
        data : { recommend: com, no : $("#no").val()},

        /* RecommendAction 클래스에서 Gson 라이브러리를 이용해
        * 응답 데이터를 json 형식으로 출력했기 때문에 dataType을 json
        * 으로 지정해야 한다. 응답 데이터를 json 형식으로 받았기 때문에
        * Ajax 통신이 성공하면 실행될 함수의 첫 번째 인수로 지정한
        * data는 자바스크립트 객체이므로 닷(.) 연산자를 이용해 접근할 수 있다.
        */
        dataType: "json",
        success: function(data) {
            /* 추천/땡큐가 반영된 것을 사용자에게 알리고
            * 응답으로 받은 갱신된 추천하기 데이터를 화면에 표시한다.
            */
            var msg = com == 'commend' ? "추천이" : "땡큐가";
            alert(msg + " 반영 되었습니다.");
            $("#commend > .recommend").text(" (" + data.recommend + ")");
            $("#thank > .recommend").text(" (" + data.thank + ")");
        },
        error: function(xhr, status, error) {
            alert("error : " + xhr.statusText + ", " + status + ", " + error);
        }
    });
});

/* 아래는 신고하기 버튼을 임시로 연결한 함수 */
function reportReply(elemId) {
    var result = confirm("이 댓글을 신고하시겠습니까?");
    if(result == true) {
        alert("report - " + result);
    }
}

```

4-2) 댓글 기능 Ajax 구현

▶ 댓글 쓰기 Ajax 구현

앞에서 게시 글에 대한 추천과 땡큐 기능을 Ajax로 구현해 보았다. 이번에는 Ajax를 이용해 게시 글에 댓글을 달고 수정하고 삭제하는 기능을 추가해 보자.

앞에서 게시 글 상세 보기 페이지에 댓글 리스트를 출력하는 기능을 구현할 때 댓글 테이블인 reply 테이블을 생성하였으므로 그 테이블을 사용해 댓글 쓰기, 수정하기, 삭제하기 기능을 구현하는 방법에 대해 알아 볼 것이다.

◆ BoardDao에 댓글을 DB에 등록하는 메소드 추가

- com.jspstudy.bbs.dao.BoardDao

```
/* 특정 게시 글에 대한 댓글 쓰기 요청시 호출되는 메소드
 * 댓글 쓰기 요청이 들어오면 ReplyWriteAction 클래스에서
 * 호출되어 사용자가 입력한 댓글을 DB에 추가하는 메소드
 */
public void addReply(Reply reply) {

    String replyInsertSql = "INSERT INTO reply"
        + " VALUES(reply_seq.NEXTVAL, ?, ?, ?, SYSDATE)";

    try {
        // DBManager을 이용해 DBCP로 부터 Connection을 대여한다.
        conn = DBManager.getConnection();
        pstmt = conn.prepareStatement(replyInsertSql);

        // insertSql 쿼리의 플레이스홀더(?)에 대응하는 데이터를 설정한다.
        pstmt.setInt(1, reply.getBbsNo());
        pstmt.setString(2, reply.getReplyContent());
        pstmt.setString(3, reply.getReplyWriter());

        // DB에 쿼리를 전송하여 댓글 추가 작업을 완료한다.
        pstmt.executeUpdate();
    } catch (Exception e) {
        System.out.println("BoardDao - addReply(reply)");
        e.printStackTrace();
    } finally {
        // DBManager를 이용해 DB 작업에 사용된 자원을 해제 한다.
        DBManager.close(conn, pstmt, rs);
    }
}
```

◆ 댓글 쓰기 Ajax 요청을 처리할 모델 클래스

- com.jspstudy.bbs.ajax.ReplyWriteAction

// 댓글 쓰기 요청을 처리할 모델 클래스

```
public class ReplyWriteAction implements AjaxProcess {

    @Override
    public void ajaxProcess(
        HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        // 댓글 쓰기 요청에서 클라이언트가 보내온 데이터를 request 객체를 부터 받는다.
        String bbsNo = request.getParameter("bbsNo");
        String replyContent = request.getParameter("replyContent");
        String replyWriter = request.getParameter("replyWriter");

        Reply reply = new Reply(
            Integer.parseInt(bbsNo), replyContent, replyWriter);

        // BoardDao 객체를 구해 댓글을 DB에 추가한다.
        BoardDao dao = new BoardDao();
        dao.addReply(reply);

        /* 새롭게 추가된 댓글을 포함해 새로운 댓글 리스트를 화면에 출력하기 위해
         * BoardDao로부터 댓글 리스트를 ArrayList<Reply>로 받는다.
         */
        ArrayList<Reply> replyList = dao.getReplyList(Integer.parseInt(bbsNo));

        // Gson 생성자를 이용해 Gson 객체를 생성한다.
        Gson gson = new Gson();

        /* dao에서 받은 ArrayList 객체를 json 형식으로 직렬화 한다.
         * dao에서 ArrayList에 Reply를 저장해 반환했기 때문에 이 데이터를
         * json 형식으로 직렬화 하면 자바스크립트의 객체 배열 형태로 저장된다.
         *
         * [{no: 1, bbsNo: 123, reply: '안녕하세요', writer: 'admin'
         *     regDate: '2016-01-12:12:53:11'},
         * {no: 2, bbsNo: 71, reply: '고맙습니다.', writer: 'midas'
         *     regDate: '2016-02-01:19:27:51'}, ... ]
         */
        String result = gson.toJson(replyList);
        System.out.println("ReplyWriteAction - result : " + result);

        /* 응답 데이터를 스트림을 통해 클라이언트에게 전송하기 위해
         * 응답 객체(response)로 스트림을 연결한다.
         */
    }
}
```

```

* AjaxController는 Ajax 요청만 받는 컨트롤러로 이 컨트롤러에서
* 현재 클래스의 ajaxProcess() 메서드를 호출하면 요청을 처리하고
* 컨트롤러로 돌아갈 때 응답 객체에 연결된 스트림을 통해 result에
* 저장된 데이터가 웹 브라우저의 XMLHttpRequest 객체로 전달된다.
**/
//response.setContentType("text/html; charset=utf-8");
response.setContentType("application/json; charset=utf-8");
PrintWriter out = response.getWriter();
out.println(result);
}
}

```

◆ 댓글 쓰기 Ajax 요청을 처리하는 코드를 AjaxController에 추가

AjaxController 클래스의 doAjax() 메서드 안에 Ajax 요청에 따라서 모델 클래스를 호출해 요청을 처리하는 if~else if 문 맨 아래에 댓글 쓰기 Ajax 요청을 처리하는 다음 코드를 추가하자.

- com.jspstudy.bbs.ajax.AjaxController

```

// ajax 요청을 처리하는 Controller 클래스
@WebServlet(name="ajaxController", urlPatterns="*.ajax")
public class AjaxController extends HttpServlet {

    ... 앞부분 코드 생략...

    ... doAjax() 메서드 안의 요청 처리 if ~ else if 문에 아래 코드 추가 ...

    } else if(command.equals("/replyWrite.ajax")) {

        /* 댓글 쓰기 요청이 들어오는 경우 처리
        * 댓글 쓰기 요청을 처리하는 ReplyWriteAction 클래스의
        * 인스턴스를 생성한 후 Request와 Response 객체를 매개변수로
        * ajaxProcess()를 호출하여 댓글 쓰기에 대한 요청을 처리 한다.
        **/
        ajaxAction = new ReplyWriteAction();
        ajaxAction.ajaxProcess(request, response);
    }

    ... 뒷부분 코드 생략...

}

```

◆ 댓글 쓰기관련 자바스크립트

댓글 쓰기관련 자바스크립트 코드를 `$(function(){ })` 안에 추가하자.

- webapp/js/reply.js

```

// 댓글 쓰기가 클릭되었을 때 이벤트 처리
$("#replyWrite").on("click", function() {

    // 화면에 보이는 상태인지 체크
    console.log($("#replyForm").css("display"));
    console.log($("#replyForm").is(":visible"));

    // 댓글 쓰기 폼이 화면에 보이는 상태이면
    if($("#replyForm").is(":visible")) {

        /* 댓글 쓰기 폼이 현재 보이는 상태이고 댓글 쓰기 위치가 아닌
        * 댓글 수정에 있으면 댓글 쓰기 폼을 슬라이드 업 하고 댓글 쓰기
        * 위치로 이동시켜 0.3초 후에 슬라이드 다운을 한다.
        */
        var $prev = $("#replyTitle").prev();
        if(! $prev.is("#replyForm")) {
            $("#replyForm").slideUp(300);
        }
        setTimeout(function(){
            $("#replyForm").insertBefore("#replyTitle").slideDown(300);
        }, 300);

    } else { // 댓글 쓰기 폼이 보이지 않는 상태이면
        $("#replyForm").removeClass("d-none")
            .css("display", "none").insertBefore("#replyTitle").slideDown(300);
    }

    /* 댓글 쓰기 폼과 댓글 수정 폼을 같이 사용하기 때문에 아래와 같이 id를
    * 동적으로 댓글 쓰기 폼으로 변경하고 댓글 수정 버튼이 클릭될 때 추가한
    * data-no라는 속성을 삭제 한다.
    */
    $("#replyForm").find("form")
        .attr("id", "replyWriteForm").removeAttr("data-no");
    $("#replyContent").val("");
    $("#replyWriteButton").val("댓글쓰기");

});

/* 댓글 쓰기 폼이 submit 될 때
* 최초 한 번은 완성된 html 문서가 화면에 출력되지만 댓글 쓰기를 한 번
* 이상하게 되면 ajax 통신을 통해 받은 데이터를 이전 화면과 동일하게
* 출력하기 위해 그때 그때 동적으로 요소를 생성하기 때문에 delegate 방식의
* 이벤트 처리가 필요하다. 댓글 쓰기, 수정 또는 삭제하기를 한 후에
* 결과 데이터를 화면에 출력하면서 자바스크립트를 이용해 html 요소를

```

- * 동적으로 생성하기 때문에 이벤트 처리가 제대로 동작하지 않을 수 있다.
- * 이럴 때는 아래와 같이 delegate 방식의 이벤트 처리가 필요하다.

```
**/
```

```
$(document).on("submit", "#replyWriteForm", function(e) {
```

```
    if($("#replyContent").val().length < 5) {
        alert("댓글은 5자 이상 입력해야 합니다.");
        return false;
    }
}
```

```
var params = $(this).serialize();
console.log(params);
```

```
$.ajax({
    "url": "replyWrite.ajax",
    "data": params,
    "type": "post",
    "dataType": "json",
    "success": function(resData) {
        console.log(resData);
    }
});
```

```
// 반복문을 통해서 - html 형식으로 작성
```

```
$("#replyList").empty();
```

```
$.each(resData, function(i, v) {
```

```
    // v.regData == 1672300816000
```

```
    var date = new Date(v.regDate);
```

```
    var strDate = date.getFullYear() + "-" + ((date.getMonth() + 1 < 10)
        ? "0" + (date.getMonth() + 1) : (date.getMonth() + 1)) + "-"
        + (date.getDate() < 10 ? "0" + date.getDate() : date.getDate()) + " "
        + (date.getHours() < 10 ? "0" + date.getHours() : date.getHours()) + ":"
        + (date.getMinutes() < 10 ? "0" + date.getMinutes() : date.getMinutes()) + ":"
        + (date.getSeconds() < 10 ? "0" + date.getSeconds() : date.getSeconds());
```

```
    var result =
```

```
        '<div class="row border border-top-0 replyRow">'
        + '<div class="col">'
        + ' <div class="row bg-light p-2">'
        + ' <div class="col-4">'
        + ' <span>' + v.replyWriter + '</span>'
        + ' </div>'
        + ' <div class="col-8 text-end">'
        + ' <span class="me-3">' + strDate + '</span>'
        + ' <button class="modifyReply btn btn-outline-success btn-sm'
        + ' data-no="' + v.no + '">'
```

```

+ '      <i class="bi bi-journal-text">수정</i>'
+ '      </button>'
+ '      <button class="deleteReply btn btn-outline-warning btn-sm"
data-no="' + v.no + '">'
+ '      <i class="bi bi-trash">삭제</i>'
+ '      </button>'
+ '      <button class="btn btn-outline-danger btn-sm"
onclick="reportReply(\' + v.no + '\">'
+ '      <i class="bi bi-telephone-outbound">신고</i>'
+ '      </button>'
+ '    </div>'
+ '  </div>'
+ ' <div class="row">'
+ '   <div class="col p-3">'
+ '     <pre>' + v.replyContent + '</pre>'
+ '   </div>'
+ ' </div>'
+ '</div>'
+ '</div>'

$("#replyList").append(result);
$("#replyList").removeClass("text-center");
$("#replyList").removeClass("p-5");

}); // end $.each()

// 댓글 쓰기가 완료되면 폼을 숨긴다.
$("#replyForm").slideUp(300)
.add("#replyContent").val("");
},
"error": function(xhr, status) {
  console.log("error : " + status);
}
});

// 폼의 전송을 취소
return false;
});

```

▶ 댓글 수정하기 Ajax 구현

댓글의 수정이나 삭제도 게시 글 수정이나 삭제와 마찬가지로 비밀번호를 확인하는 로직이 필요하지
만 댓글 테이블을 생성할 때 비밀번호 컬럼을 만들지 않았기 때문에 이 예제에서는 그 부분은 따로
구현하지 않겠다. 필요하다면 현재 테이블을 수정하지 않고 댓글 수정이나 삭제에서 관리자 권한을
체크하거나 작성자를 체크해서 댓글을 작성한 작성자만 수정/삭제가 가능하도록 구현할 수도 있을

것이다.

◆ BoardDao에 댓글을 DB에서 수정하는 메소드 추가

- com.jspstudy.bbs.dao.BoardDao

```
/* 특정 게시 글에 대한 댓글 수정 요청시 호출되는 메소드
 * 댓글 수정 요청이 들어오면 ReplyUpdateAction 클래스에서
 * 호출되어 사용자가 입력한 댓글을 DB에 추가하는 메소드
 */
public void updateReply(Reply reply) {

    String replyUpdateSql = "UPDATE reply SET reply_content=?"
        + " reg_date=SYSDATE WHERE no=?";

    try {
        // DBManager을 이용해 DBCP로 부터 Connection을 대여한다.
        conn = DBManager.getConnection();
        pstmt = conn.prepareStatement(replyUpdateSql);

        // insertSql 쿼리의 플레이스홀더(?)에 대응하는 데이터를 설정한다.
        pstmt.setString(1, reply.getReplyContent());
        pstmt.setInt(2, reply.getNo());

        // DB에 쿼리를 전송하여 댓글 추가 작업을 완료한다.
        pstmt.executeUpdate();
    } catch (Exception e) {
        System.out.println("BoardDao - updateReply(reply)");
        e.printStackTrace();
    } finally {
        // DBManager를 이용해 DB 작업에 사용된 자원을 해제 한다.
        DBManager.close(conn, pstmt, rs);
    }
}
```

◆ 댓글 수정 Ajax 요청을 처리할 모델 클래스

- com.jspstudy.bbs.ajax.ReplyUpdateAction

// 댓글 수정하기 요청을 처리할 모델 클래스

```
public class ReplyUpdateAction implements AjaxProcess {

    @Override
    public void ajaxProcess(
        HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
```



```

// 댓글 수정 요청에서 클라이언트가 보내온 데이터를 request 객체를 부터 받는다.
String bbsNo = request.getParameter("bbsNo");
String replyNo = request.getParameter("replyNo");
String replyContent = request.getParameter("replyContent");

Reply reply = new Reply(Integer.parseInt(replyNo),
    Integer.parseInt(bbsNo), replyContent, null, null);

// BoardDao 객체를 구해 댓글을 DB에서 수정한다.
BoardDao dao = new BoardDao();
dao.updateReply(reply);

/* 수정된 댓글을 포함해 새로운 댓글 리스트를 화면에 출력하기 위해
 * BoardDao로부터 댓글 리스트를 ArrayList<Reply>로 받는다.
 */
ArrayList<Reply> replyList = dao.getReplyList(Integer.parseInt(bbsNo));

// Gson 생성자를 이용해 Gson 객체를 생성한다.
Gson gson = new Gson();

/* dao에서 받은 ArrayList 객체를 json 형식으로 직렬화 한다.
 * dao에서 ArrayList에 Reply를 저장해 반환했기 때문에 이 데이터를
 * json 형식으로 직렬화 하면 자바스크립트의 객체 배열 형태로 저장된다.
 *
 * [{no: 1, bbsNo: 123, reply: '안녕하세요', writer: 'admin'
 *   regDate: '2016-01-12:12:53:11'},
 * {no: 2, bbsNo: 71, reply: '고맙습니다.', writer: 'midas'
 *   regDate: '2016-02-01:19:27:51'}, ... ]
 */
String result = gson.toJson(replyList);
System.out.println("ReplyUpdateAction - result : " + result);

/* 응답 데이터를 스트림을 통해 클라이언트에게 전송하기 위해
 * 응답 객체(response)로 스트림을 연결한다.
 *
 * AjaxController는 Ajax 요청만 받는 컨트롤러로 이 컨트롤러에서
 * 현재 클래스의 ajaxProcess() 메서드를 호출하면 요청을 처리하고
 * 컨트롤러로 돌아갈 때 응답 객체에 연결된 스트림을 통해 result에
 * 저장된 데이터가 웹 브라우저의 XMLHttpRequest 객체로 전달된다.
 */
//response.setContentType("text/html; charset=utf-8");
response.setContentType("application/json; charset=utf-8");
PrintWriter out = response.getWriter();
out.println(result);
}

```

```
}
```

◆ 댓글 수정하기 Ajax 요청을 처리하는 코드를 AjaxController에 추가

AjaxController 클래스의 doAjax() 메서드 안에 Ajax 요청에 따라서 모델 클래스를 호출해 요청을 처리하는 if~else if 문 맨 아래에 댓글 수정하기 Ajax 요청을 처리하는 다음 코드를 추가하자.

- com.jspstudy.bbs.ajax.AjaxController

```
// ajax 요청을 처리하는 Controller 클래스
```

```
@WebServlet(name="ajaxController", urlPatterns="*.ajax")
```

```
public class AjaxController extends HttpServlet {
```

```
... 앞부분 코드 생략...
```

```
... doAjax() 메서드 안의 요청 처리 if ~ else if 문에 아래 코드 추가 ...
```

```
    } else if(command.equals("/replyUpdate.ajax")) {
```

```
        /* 댓글 수정하기 요청이 들어오는 경우 처리
```

```
        * 댓글 수정하기 요청을 처리하는 ReplyUpdateAction 클래스의
```

```
        * 인스턴스를 생성한 후 Request와 Response 객체를 매개변수로
```

```
        * ajaxProcess()를 호출하여 댓글 쓰기에 대한 요청을 처리 한다.
```

```
        **/
```

```
        ajaxAction = new ReplyUpdateAction();
```

```
        ajaxAction.ajaxProcess(request, response);
```

```
    }
```

```
... 뒷부분 코드 생략...
```

```
}
```

◆ 댓글 수정관련 자바스크립트

댓글 수정관련 자바스크립트 코드를 `$(function(){ });` 안에 추가하자.

- webapp/js/reply.js

```
/* 댓글 수정 버튼이 클릭되면
```

```
    * 댓글을 수정한 후에 동적으로 요소를 생성하기 때문에 delegate 방식으로 이벤트를
```

```
    * 등록해야 한다. 만약 $(".modifyReply").click(function() {}); 형식으로 이벤트를
```

```
    * 등록했다면 새로운 댓글을 등록하거나, 수정 또는 삭제한 후에는 클릭 이벤트가
```

```
    * 제대로 동작되지 않을 수 있기 때문에 delegate 방식의 이벤트 처리가 필요하다.
```

```
    **/
```

```
$(document).on("click", ".modifyReply", function() {
```

```
    // 화면에 보이는 상태인지 체크
```

```

console.log($("#replyForm").css("display"));
console.log($("#replyForm").is(":visible"));

// 수정 버튼이 클릭된 최상의 부모를 구한다.
console.log($(this).parents(".replyRow"));
var $replyRow = $(this).parents(".replyRow");

// 댓글 쓰기 폼이 화면에 보이는 상태이면
if($("#replyForm").is(":visible")) {

    /* 댓글 쓰기 폼이 현재 보이는 상태이고 현재 클릭된 수정 버튼의
    * 부모 요소의 다음 요소가 아니라면 댓글 쓰기 폼을 슬라이드 업 하고
    * 댓글 수정 위치로 이동시켜 0.3초 후에 슬라이드 다운을 한다.
    */
    var $next = $replyRow.next();
    if(! $next.is("#replyForm")) {
        $("#replyForm").slideUp(300);
    }
    setTimeout(function(){
        $("#replyForm").insertAfter($replyRow).slideDown(300);
    }, 300);

} else { // 댓글 쓰기 폼이 화면에 보이지 않는 상태이면
    $("#replyForm").removeClass("d-none")
        .css("display", "none").insertAfter($replyRow).slideDown(300);
}

/* 댓글 쓰기 폼과 댓글 수정 폼을 같이 사용하기 때문에 아래와 같이 id를
* 동적으로 댓글 쓰기 폼으로 변경하고 댓글 수정 버튼이 클릭될 때 추가한
* data-no라는 속성을 삭제 한다.
*/
$("#replyForm").find("form")
    .attr({id: "replyUpdateForm", "data-no": $(this).attr("data-no") });
$("#replyWriteButton").val("댓글수정");

// 현재 클릭된 수정 버튼이 있는 댓글을 읽어와 수정 폼의 댓글 입력란에 출력한다.
var reply = $(this).parent().parent().next().find("pre").text();
$("#replyContent").val($.trim(reply));

});

// 댓글 수정 폼이 submit 될 때
$(document).on("submit", "#replyUpdateForm", function() {
//$("#replyUpdateForm").on("submit", function() {

```

```

if($("#replyContent").val().length <= 5) {
    alert("댓글은 5자 이상 입력해야 합니다.");
    // Ajax 요청을 취소한다.
    return false;
}

/* 아래에서 $("#replyTable").empty(); 가 호출되면
 * 댓글 쓰기 폼도 같이 문서에서 삭제되기 때문에 백업을 받아야 한다.
 */
$replyForm = $("#replyForm").slideUp(300);

/* replyNo는 최초 폼이 출력될 때 설정되지 않았다.
 * 댓글 쓰기 폼과 댓글 수정 폼을 하나로 처리하기 때문에 댓글 번호는
 * 동적으로 구하여 요청 파라미터에 추가해 줘야 한다. 댓글 수정시
 * 댓글 번호를 서버로 전송해야 댓글 번호에 해당하는 댓글을 수정할 수 있다.
 */
var params = $(this).serialize() + "&replyNo=" + $(this).attr("data-no");
console.log(params);

$.ajax({
    url: "replyUpdate.ajax",
    type: "post",
    data: params,
    dataType: "json",
    success: function(resData, status, xhr) {

        console.log(resData);

        // 반복문을 통해서 - html 형식으로 작성
        $("#replyList").empty();
        $.each(resData, function(i, v) {

            // v.regData == 1672300816000
            var date = new Date(v.regDate);
            var strDate = date.getFullYear() + "-" + ((date.getMonth() + 1 < 10)
                ? "0" + (date.getMonth() + 1) : (date.getMonth() + 1)) + "-"
                + (date.getDate() < 10 ? "0" + date.getDate() : date.getDate()) + " "
                + (date.getHours() < 10 ? "0" + date.getHours() : date.getHours()) + ":"
                + (date.getMinutes() < 10 ? "0" + date.getMinutes() : date.getMinutes()) + ":"
                + (date.getSeconds() < 10 ? "0" + date.getSeconds() : date.getSeconds());

            var result =
                '<div class="row border border-top-0 replyRow">'
                + '<div class="col">'

```

```

+ ' <div class="row bg-light p-2">'
+ '   <div class="col-4">'
+ '     <span>' + v.replyWriter + '</span>'
+ '   </div>'
+ '   <div class="col-8 text-end">'
+ '     <span class="me-3">' + strDate + '</span>'
+ '     <button class="modifyReply btn btn-outline-success btn-sm"
data-no="' + v.no + '">'
+ '       <i class="bi bi-journal-text">수정</i>'
+ '     </button>'
+ '     <button class="deleteReply btn btn-outline-warning btn-sm"
data-no="' + v.no + '">'
+ '       <i class="bi bi-trash">삭제</i>'
+ '     </button>'
+ '     <button class="btn btn-outline-danger btn-sm"
onclick="reportReply(\' + v.no + '\">'
+ '       <i class="bi bi-telephone-outbound">신고</i>'
+ '     </button>'
+ '   </div>'
+ ' </div>'
+ ' <div class="row">'
+ '   <div class="col p-3">'
+ '     <pre>' + v.replyContent + '</pre>'
+ '   </div>'
+ ' </div>'
+ ' </div>'
+ '</div>'

```

```

$("#replyList").append(result);

```

```

}); // end $.each()

```

```

// 댓글 쓰기가 완료되면 폼을 숨긴다.

```

```

$("#replyContent").val("");

```

```

$replyForm.css("display", "none");

```

```

$("#global-content > div.col").append($replyForm);

```

```

},

```

```

error: function(xhr, status, error) {
  alert("ajax 실패 : " + status + " - " + xhr.status);
}

```

```

});

```

```

// 폼이 submit 되는 것을 취소한다.

```

```

    return false;
});

```

▶ 댓글 삭제 Ajax 구현

앞에서도 언급했듯이 댓글의 삭제는 삭제 권한을 먼저 체크하고 구현해야 하지만 이 예제에서는 그 부분을 제외하고 Ajax를 통해 댓글을 삭제하는 부분을 알아볼 것이다.

◆ BoardDao에 DB에서 댓글을 삭제하는 메소드 추가

- com.jspstudy.bbs.dao.BoardDao

```

/* 특정 게시 글에 대한 댓글 삭제 요청시 호출되는 메소드
 * 댓글 삭제 요청이 들어오면 ReplyDeleteAction 클래스에서
 * 호출되어 사용자가 입력한 댓글을 DB에 추가하는 메소드
 */
public void deleteReply(Reply reply) {

    String replyDeleteSql = "DELETE FROM reply WHERE no = ?";

    try {
        // DBManager을 이용해 DBCP로 부터 Connection을 대여한다.
        conn = DBManager.getConnection();
        pstmt = conn.prepareStatement(replyDeleteSql);

        // insertSql 쿼리의 플레이스홀더(?)에 대응하는 데이터를 설정한다.
        pstmt.setInt(1, reply.getNo());

        // DB에 쿼리를 전송하여 댓글 추가 작업을 완료한다.
        pstmt.executeUpdate();
    } catch (Exception e) {
        System.out.println("BoardDao - deleteReply(reply)");
        e.printStackTrace();
    } finally {
        // DBManager를 이용해 DB 작업에 사용된 자원을 해제 한다.
        DBManager.close(conn, pstmt);
    }
}

```

◆ 댓글 삭제 Ajax 요청을 처리할 모델 클래스

- com.jspstudy.bbs.ajax.ReplyDeleteAction

// 댓글 삭제하기 요청을 처리할 모델 클래스

```

public class ReplyDeleteAction implements AjaxProcess {

```

```

    @Override

```

```

public void ajaxProcess(
    HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // 댓글 수정 요청에서 클라이언트가 보내온 데이터를 request 객체를 부터 받는다.
    String bbsNo = request.getParameter("bbsNo");
    String replyNo = request.getParameter("replyNo");

    Reply reply = new Reply(Integer.parseInt(replyNo),
        Integer.parseInt(bbsNo), null, null, null);

    // BoardDao 객체를 구해 댓글을 DB에 추가한다.
    BoardDao dao = new BoardDao();
    dao.deleteReply(reply);

    /* 새롭게 추가된 댓글을 포함해 새로운 댓글 리스트를 화면에 출력하기 위해
     * BoardDao로부터 댓글 리스트를 ArrayList<Reply>로 받는다.
     */
    ArrayList<Reply> replyList = dao.getReplyList(Integer.parseInt(bbsNo));

    // Gson 생성자를 이용해 Gson 객체를 생성한다.
    Gson gson = new Gson();

    /* dao에서 받은 ArrayList 객체를 json 형식으로 직렬화 한다.
     * dao에서 ArrayList에 Reply를 저장해 반환했기 때문에 이 데이터를
     * json 형식으로 직렬화 하면 자바스크립트의 객체 배열 형태로 저장된다.
     *
     * [{no: 1, bbsNo: 123, reply: '안녕하세요', writer: 'admin'
     *     regDate: '2016-01-12:12:53:11'},
     * {no: 2, bbsNo: 71, reply: '고맙습니다.', writer: 'midas'
     *     regDate: '2016-02-01:19:27:51'}, ... ]
     */
    String result = gson.toJson(replyList);
    System.out.println("ReplyDeleteAction - result : " + result);

    /* 응답 데이터를 스트림을 통해 클라이언트에게 전송하기 위해
     * 응답 객체(response)로 스트림을 연결한다.
     *
     * AjaxController는 Ajax 요청만 받는 컨트롤러로 이 컨트롤러에서
     * 현재 클래스의 ajaxProcess() 메서드를 호출하면 요청을 처리하고
     * 컨트롤러로 돌아갈 때 응답 객체에 연결된 스트림을 통해 result에
     * 저장된 데이터가 웹 브라우저의 XMLHttpRequest 객체로 전달된다.
     */
    response.setContentType("text/html; charset=utf-8");
    PrintWriter out = response.getWriter();

```

```

        out.println(result);
    }
}

```

◆ 댓글 삭제하기 Ajax 요청을 처리하는 코드를 AjaxController에 추가

AjaxController 클래스의 doAjax() 메서드 안에 Ajax 요청에 따라서 모델 클래스를 호출해 요청을 처리하는 if~else if 문 맨 아래에 댓글 삭제하기 Ajax 요청을 처리하는 다음 코드를 추가하자.

- com.jspstudy.bbs.ajax.AjaxController

```

// ajax 요청을 처리하는 Controller 클래스
@WebServlet(name="ajaxController", urlPatterns="*.ajax")
public class AjaxController extends HttpServlet {

    ... 앞부분 코드 생략...
    ... doAjax() 메서드 안의 요청 처리 if ~ else if 문에 아래 코드 추가 ...

    } else if(command.equals("/replyDelete.ajax")) {

        /* 댓글 수정하기 요청이 들어오는 경우 처리
         * 댓글 수정하기 요청을 처리하는 ReplyDeleteAction 클래스의
         * 인스턴스를 생성한 후 Request와 Response 객체를 매개변수로
         * ajaxProcess()를 호출하여 댓글 쓰기에 대한 요청을 처리 한다.
         */
        ajaxAction = new ReplyDeleteAction();
        ajaxAction.ajaxProcess(request, response);
    }

    ... 뒷부분 코드 생략...

}

```

◆ 댓글 삭제관련 자바스크립트

댓글 삭제관련 자바스크립트 코드를 `$(function(){ });` 안에 추가하자.

- webapp/js/reply.js

```

/* 댓글 삭제 버튼이 클릭되면
 * 댓글을 삭제한 후에 동적으로 요소를 생성하기 때문에 delegate 방식으로 이벤트를
 * 처리를 해야 한다. 만약 $(".deleteReply").click(function() {}); 와 같이 이벤트를
 * 등록했다면 댓글을 삭제한 후에는 클릭 이벤트가 제대로 동작되지 않을 수 있기 때문에
 * 아래 코드와 같이 delegate 방식의 이벤트 처리가 필요하다.
 */
$(document).on("click", ".deleteReply", function() {

```



```

var no = $(this).attr("data-no");
var writer = $(this).parent().prev().find("span").text();
var bbsNo = $("#replyForm input[name=bbsNo]").val();
var params = "replyNo=" + no + "&bbsNo=" + bbsNo;
console.log(params);

/* 아래에서 $("#replyTable").empty(); 가 호출되면
 * 댓글 쓰기 폼이 문서에서 삭제될 수 있으므로 백업을 받아야 한다.
 */
$replyForm = $("#replyForm").slideUp(300);

var result = confirm(writer + "님이 작성한 " + no + "번 댓글을 삭제하시겠습니까?");

if(result) {
    $.ajax({
        url: "replyDelete.ajax",
        type: "post",
        data: params,
        dataType: "json",
        success: function(resData, status, xhr) {

            console.log(resData);

            // 반복문을 통해서 - html 형식으로 작성
            $("#replyList").empty();

            $.each(resData, function(i, v) {

                // v.regData == 1672300816000
                var date = new Date(v.regDate);
                var strDate = date.getFullYear() + "-" + ((date.getMonth() + 1 < 10)
                    ? "0" + (date.getMonth() + 1) : (date.getMonth() + 1)) + "-"
                    + (date.getDate() < 10 ? "0" + date.getDate() : date.getDate()) + " "
                    + (date.getHours() < 10 ? "0" + date.getHours() : date.getHours()) + ":"
                    + (date.getMinutes() < 10 ? "0" + date.getMinutes() : date.getMinutes()) +
                    ":"
                    + (date.getSeconds() < 10 ? "0" + date.getSeconds() : date.getSeconds());

                var result =
                    '<div class="row border border-top-0 replyRow">'
                    + '<div class="col">'
                    + ' <div class="row bg-light p-2">'
                    + ' <div class="col-4">'
                    + ' <span>' + v.replyWriter + '</span>'
                    + ' </div>'

```

```

+ '    <div class="col-8 text-end">'
+ '        <span class="me-3">' + strDate + "</span>"
+ '        <button class="modifyReply btn btn-outline-success btn-sm"
data-no="' + v.no + '">'
+ '            <i class="bi bi-journal-text">수정</i>'
+ '        </button>'
+ '        <button class="deleteReply btn btn-outline-warning btn-sm"
data-no="' + v.no + '">'
+ '            <i class="bi bi-trash">삭제</i>'
+ '        </button>'
+ '        <button class="btn btn-outline-danger btn-sm"
onclick="reportReply(\'' + v.no + '\')>'
+ '            <i class="bi bi-telephone-outbound">신고</i>'
+ '        </button>'
+ '    </div>'
+ '</div>'
+ ' <div class="row">'
+ '     <div class="col p-3">'
+ '         <pre>' + v.replyContent + '</pre>'
+ '     </div>'
+ ' </div>'
+ '</div>'
+ '</div>'

$("#replyList").append(result);

}); // end $.each()

// 댓글 쓰기가 완료되면 폼을 숨긴다.
$("#replyContent").val("");
$replyForm.css("display", "none");
$("#global-content > div.col").append($replyForm);

},
error: function(xhr, status, error) {
    alert("ajax 실패 : " + status + " - " + xhr.status);
}
});
}
// 앵커 태그에 의해 페이지가 이동되는 것을 취소한다.
return false;
});

```

4-3) 파일 다운로드 구현

앞에서 게시 글 상세보기를 구현할 때 boardDetail.jsp 페이지에서 파일 다운로드를 링크 방식으로 구현하였다. 이 방식은 파일이 위치한 경로가 노출되어 보안상 좋지 않아 파일이 손실될 위험도 가지고 있다. 그래서 파일 다운로드 기능을 제공하는 FileDownloadService 클래스를 작성하고 이 클래스에서 스트림을 통해 응답 데이터에 파일을 전송하는 기능을 구현할 것이다.

◆ 파일 다운로드 요청을 처리하는 모델 클래스

- com.jspstudy.bbs.dao.FileDownloadService

// 게시 글 상세 보기에서 들어오는 파일을 다운로드 요청을 처리하는 모델 클래스

```
public class FileDownloadService implements CommandProcess {

    @Override
    public String requestProcess(
        HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        /* 서블릿 초기화 메서드에서 업로드 파일이 저장될 폴더 정보를 서블릿 초기화
         * 파라미터로 부터 읽어와 그 로컬 경로를 ServletContext 객체에 저장하였다.
         *
         * ServletContext 객체는 웹 애플리케이션 당 1개가 생성되며 웹 애플리케이션이
         * 구동되는데 필요한 정보를 담고 있는 객체로 JSP 페이지에서는 application
         * 내장객체로 접근할 수 있다. 아래와 같이 ServletContext 객체의 속성에
         * 저장되면 이 웹 애플리케이션의 모든 컴포넌트에서 이 정보를 사용할 수 있다.
         *
         * request 객체의 getServletContext() 메서드를 이용해 ServletContext
         * 객체를 구하고 그 속성에 저장된 uploadDir 정보를 읽고 있다.
         *
         * 이 예제는 경로를 포함해서 파일 이름을 저장했기 때문에 아래가 필요 없을 수 있지만
         * 파일명만 저장하고 필요한 경로를 추가로 지정하는 방식을 사용할 경우 아래와 같이
         * 기준이 되는 폴더를 ServletContext 객체의 속성에 저장하고 웹 애플리케이션의
         * 여러 컴포넌트에서 이를 가져다 사용할 수 있도록 구현할 수 있다.
         */
        ServletContext ctx = request.getServletContext();
        String downDir = (String) ctx.getAttribute("uploadDir");

        /* 보안적인 부분을 생각한다면 게시 글의 no 정도만 요청 파라미터로 받아 그 no에
         * 해당하는 파일 정보를 DB로 부터 읽어와 처리하는 방식으로 처리해야 하나, 여기서는
         * 파일 이름을 받아서 바로 다운로드 처리하는 방식을 사용했다.
         */
        String fileName = request.getParameter("fileName");

        // 파일의 위치를 현재 시스템을 기준으로 실제 경로를 구하여 File 객체를 생성한다.
        String downPath = ctx.getRealPath(downDir + "/" + fileName);
        File file = new File(downPath);
```

```

System.out.println("downPath : " + downPath);
System.out.println("filePath : " + file.getPath() + ", " + file.getName());

// ServletContext 객체를 통해 다운로드할 파일 유형을 구한다.
String mimeType = ctx.getMimeType(downPath);
System.out.println("mimeType : " + mimeType);

// 알려지지 않은 파일 유형이 이라면
if(mimeType == null) {
    /* 파일 유형이 알려지지 않은 파일일 경우 일련된 8bit 스트림 형식을 지정한다.
     * 이 방식은 알려지지 않은 파일 유형의 대한 읽기 형식을 지정할 때 사용한다.
     */
    mimeType = "application/octet-stream";
}

// 응답 객체에 파일 유형에 대한 콘텐츠 타입을 지정한다.
response.setContentType(mimeType);

/* 파일 이름이 한글이 포함되어 있으면 깨질수 있기 때문에 인코딩 처리를 해 준다.
 * 파일명에 공백이 존재하면 아래 인코딩 시에 공백은 "+" 기호로 바뀌게 된다.
 */
String fileNameEncoding = URLEncoder.encode(file.getName(), "UTF-8");

/* 파일명에 공백이 있으면 위에서 "+"로 처리되기 때문에
 * "+" 기호를 다시 공백으로 변경해 준다.
 */
fileName = fileNameEncoding.replaceAll("\\+", "%20");

/* 기타 추가적인 정보를 응답 객체의 헤더에 추가한다.
 * 브라우저가 이 헤더를 읽어 다운로드로 연결되는 화면을 출력해 준다.
 */
response.setHeader("Content-Disposition",
    "attachment; filename=" + fileName);

// 업로드된 파일을 읽어올 InputStream을 개설한다.
FileInputStream in = new FileInputStream(file);

// 응답 객체를 통해 파일을 보내기 위해 OutputStream을 개설한다.
ServletOutputStream downStream = response.getOutputStream();

// 버퍼로 사용할 byte 배열 생성 - 1MB 씩 한 번에 읽을 예정
byte[] b = new byte[1 * 1024 * 1024];
int readByte = 0;

```

```

while((readByte = in.read(b, 0, b.length)) != -1) {
    downStream.write(b, 0, readByte);
}

// 스트림에 남은 모든 데이터를 전송하고 스트림을 닫는다.
downStream.flush();
downStream.close();
in.close();

/* viewPage 정보가 null 이면 컨트롤러에서 뷰를 거치지 않고 그대로 응답되기
 * 때문에 위에서 응답 객체를 통해 연결한 스트림으로 파일이 전송되어 브라우저로
 * 응답되고 브라우저는 이를 다운로드로 연결해 사용자에게 보여준다.
 */
return null;
}
}

```

◆ BoardController에 파일 다운로드 요청을 처리하는 코드를 추가

BoardController 클래스의 doProcess() 메서드 안에 요청 명령에 따라 모델 클래스를 호출해 요청을 처리하는 if~else if 문 맨 아래에 파일 다운로드 요청을 처리하는 다음 코드를 추가하자.

- com.jspstudy.bbs.controller.BoardController

// 게시판 및 회원관련 요청을 처리하는 프런트 컨트롤러

```
public class BoardController extends HttpServlet {
```

... 앞부분 코드 생략 ...

```
} else if(command.equals("/fileDownload.mvc")) {
```

```

/* 게시 글 상세보기에서 파일 다운로드가 클릭된 경우의 처리
 * 파일 다운로드 요청을 처리하는 FileDownloadService 클래스의
 * 인스턴스를 생성한 후 Request와 Response 객체를 매개변수로
 * requestProcess()를 호출하여 파일 다운로드 요청을 처리한다.
 */

```

```
service = new FileDownloadService();
```

```
viewPage = service.requestProcess(request, response);
```

```
}
```

... 뒷부분 코드 생략 ...

```
}
```

◆ 게시 글 상세보기 뷰 페이지에 파일 다운로드 링크 추가

게시 글 상세보기 페이지에서 링크 방식과 파일 다운로드 기능을 제공하는 모델 클래스를 이용한 파일 다운로드 방식 모듈을 지정할 것이다.

아래 코드에서 빨간색 볼드체로 표시한 부분이 파일 다운로드 링크를 지정한 부분이다.

- WEB-INF/board/boardDetail.jsp

... **중략** ...

```
<tr>
  <th>파일</th>
  <td colspan="3">
    <c:if test="${ empty board.file1 }">
      첨부파일 없음
    </c:if>
    <c:if test="${ not empty board.file1 }">
      <a href="fileDownload.mvc?fileName=${ board.file1 }">파일 다운로드</a>
    </c:if>
  </td>
</tr>
```

... **중략** ...