

# 17<sup>장</sup>

## 모델2 방식으로 효율적으로 개발하기

- 
- 17.1 웹 애플리케이션 모델
  - 17.2 MVC 디자인 패턴
  - 17.3 MVC를 이용한 회원 관리
  - 17.4 모델2로 답변형 게시판 구현하기

# 17.1

## 웹 애플리케이션 모델

7

보통 웹 애플리케이션을 개발할 때 화면은 디자이너가 맡아서 구현하고, 데이터베이스 연동 같은 비즈니스 로직은 프로그래머가 맡아서 구현합니다. 즉, 각자 맡은 기능을 좀 더 분업화해서 개발을 하는 것이죠.

일반적으로 어떤 일을 맡아 진행하게 되면 일단은 기준에 주로 사용했던 방법이나 방식을 따르게 마련입니다. 웹 애플리케이션을 개발할 때도 마찬가지입니다. 일일이 처음부터 새로 개발하는 것 이 아니라 기준에 웹 애플리케이션 개발 방법이나 방식을 따릅니다.

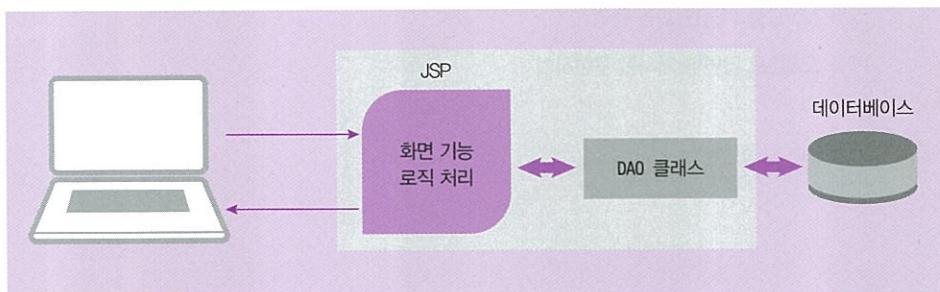
따라서 지금의 웹 애플리케이션 개발은 일반적으로 많이 사용하는 표준화 소스 구조를 만들어 개발을 진행합니다. 이러한 표준화된 소스 구조를 **웹 애플리케이션 모델**이라고 합니다. 웹 애플리케이션 모델의 종류에는 모델1과 모델2 방식이 있습니다.

### 17.1.1 모델1 방식

지금까지 JSP 실습 예제를 구현한 방식이 바로 모델1 방식입니다. 이는 데이터베이스 연동 같은 비즈니스 로직 작업과 그 작업 결과를 나타내주는 작업을 동일한 JSP에서 수행합니다. 즉, 모든 클라이언트의 요청과 비즈니스 로직 처리를 JSP가 담당하는 구조입니다.

그림 17-1은 모델1 방식으로 웹 애플리케이션이 동작하는 과정을 나타낸 것입니다.

▼ 그림 17-1 모델1로 구현한 애플리케이션 동작 방식



모델1 방식은 기능 구현이 쉽고 편리하다라는 장점이 있는 반면에 요즘처럼 웹 사이트 화면 기능이 복잡해지면 화면 기능과 비즈니스 로직 기능이 섞이면서 유지보수에 문제가 생깁니다.

예를 들어 의류 쇼핑몰을 모델1 방식으로 구현해 운영하고 있다고 합시다. 계절이 가을에서 겨울로 바뀌면 화면에 나타낼 의류 상품의 이미지도 바꿔줘야 합니다.

▼ 그림 17-2 의류 쇼핑몰의 경우 계절에 따라 화면에 나타내는 이미지도 바꿔줘야 함



디자이너가 이 작업을 하려면 JSP에 개발자가 관계되는 비즈니스 로직 기능도 알아야 하므로 작업하기가 쉽지 않을 뿐 아니라 비즈니스 로직과 화면 기능이 섞여 코드 재사용성도 떨어집니다. 이렇듯 모델1 방식으로 웹 애플리케이션을 구현할 경우 조금만 기능이 복잡해져도 유지보수가 어렵다는 단점이 있습니다. 모델1 방식의 이러한 단점을 보완한 것이 바로 모델2 방식입니다.

## 17.1.2 모델2 방식

모델2 방식의 핵심은 웹 애플리케이션의 각 기능(클라이언트의 요청 처리, 응답 처리, 비즈니스, 로직 처리)을 분리해서 구현하자는 것입니다. 객체 지향 프로그래밍에서 각각의 기능을 모듈화해서 개발하는 것과 같은 원리죠.

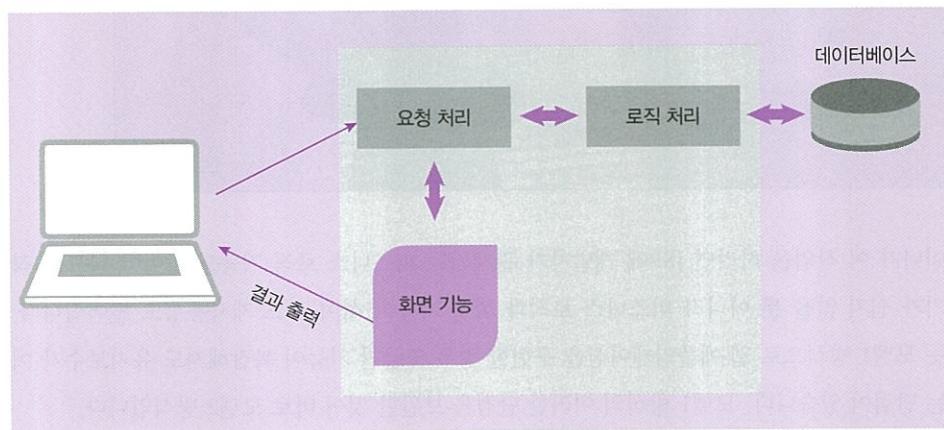
웹 프로그램 개발 시 개발자가 전체 기능을 몰라도 각 기능이 모듈화되어 있으므로 자신이 맡은 부분만 개발하면 됩니다. 각 부분을 조립만 하면 나중에 전체를 사용할 수 있어 개발 효율성도 높습니다. 물론 모델2 방식으로 개발하기 위해서는 필요한 기술이나 개념을 숙지해야 하는 번거로움은 있지만 초급자라면 우선 자신이 맡은 부분만 개발하면 되므로 훨씬 효율적인 개발 방식이라고 할 수 있습니다. 그리고 개발 후 서비스를 제공할 때도 유지보수가 편할 뿐만 아니라 개발한 모듈들은 비슷한 프로그램을 만들 때 사용할 수 있어 코드 재사용성도 높습니다. 현재 모든 웹 프로그램은 모델2 방식으로 개발한다고 보면 됩니다.

모델2 방식의 특징은 다음과 같습니다.

- 각 기능이 서로 분리되어 있어 개발 및 유지보수가 쉽습니다.
- 각 기능(모듈)의 재사용성이 높습니다.
- 디자이너와 개발자의 작업을 분업화해서 쉽게 개발할 수 있습니다.
- 모델2 방식과 관련된 기능이나 개념의 학습이 필요합니다.

그림 17-3은 모델2 방식으로 동작하는 웹 사이트를 나타낸 것입니다.

▼ 그림 17-3 모델2 동작 방식



## 17.2

# MVC 디자인 패턴

앞서 살펴본 모델2 구조에는 여러 가지 개념들이 사용되는데 그중 가장 자주 사용되는 개념이 MVC입니다. MVC란 Model–View–Controller(모델–뷰–컨트롤러)의 약자로, 일반 PC 프로그램 개발에 사용되는 디자인 패턴을 웹 애플리케이션에 도입한 것입니다. 즉, 웹 애플리케이션을 화면 부분, 요청 처리 부분, 로직 처리 부분으로 나누어 개발하는 방법이죠.

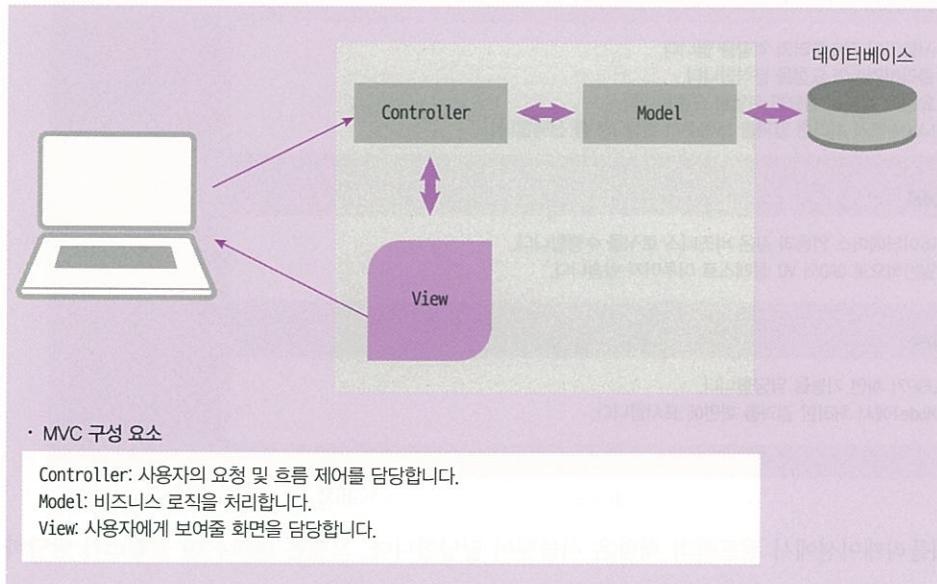
원래 모델2 방식의 구조가 MVC를 포함하는 개념이지만 MVC가 모델2 방식의 뼈대를 이루므로 모델2 방식으로 구현한다는 말은 곧 MVC로 구현한다는 것과 같은 의미로 보면 됩니다.

MVC의 특징은 다음과 같습니다.

- 각 기능이 분리되어 있어 개발 및 유지보수가 편리합니다.
- 각 기능의 재사용성이 높아집니다.
- 디자이너와 개발자의 작업을 분업화해서 쉽게 개발할 수 있습니다.

그림 17-4는 MVC로 이루어진 웹 애플리케이션의 동작 과정을 나타낸 것입니다.

▼ 그림 17-4 MVC 구조와 구성 요소

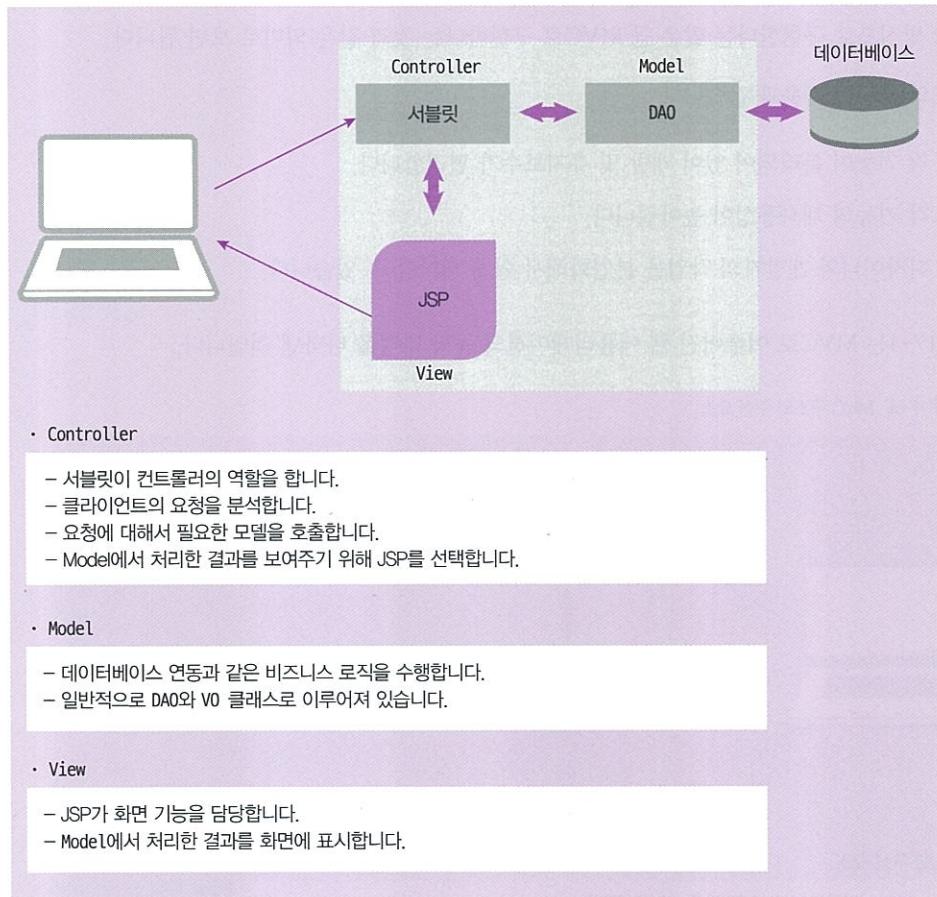


컨트롤러는 사용자로부터 요청을 받아 어떤 비즈니스 로직을 처리해야 할지 제어합니다. 모델은 데이터베이스 연동 같은 비즈니스 로직을 처리하고, 뷰는 모델에서 처리한 결과를 화면에 구현하여 클라이언트로 전송합니다.

### 17.2.1 MVC 구성 요소와 기능

그럼 MVC로 구현한 웹 애플리케이션은 어떤 구조인지 조금 더 면밀히 살펴보겠습니다.

▼ 그림 17-5 MVC로 구현된 웹 애플리케이션과 각 구성 요소의 기능



웹 애플리케이션에서 컨트롤러 역할은 서블릿이 담당합니다. 모델은 DAO나 VO 클래스가 담당하고, 뷰 역할은 JSP가 담당합니다.

# 17.3

## MVC를 이용한 회원 관리

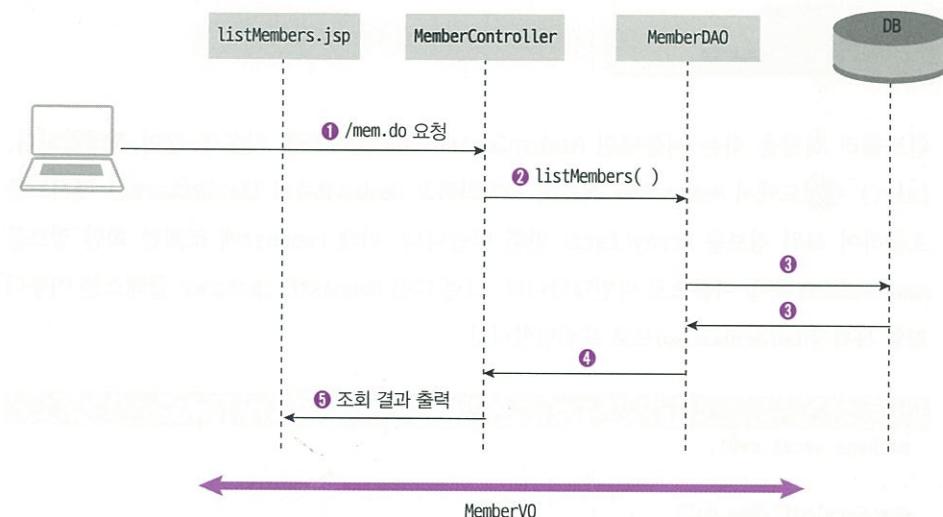
우리는 앞에서 JSP를 이용해 모델1 방식으로 회원 관리 기능을 구현한 적이 있습니다(14장 참고). 기억하나요? 모델1 방식이었기 때문에 모든 기능을 JSP에서 처리하였습니다.

이번에는 MVC 방식으로 브라우저의 요청은 서블릿이 맡고, 비즈니스 처리는 모델이 맡고, 화면은 JSP가 맡는 방식으로 회원 관리 기능을 다시 구현해 보겠습니다. 결국 같은 프로그램을 개발하는 것이지만 방법을 달리함으로써 개발 원리를 쉽게 익히는 것이 학습 목표입니다.

### 17.3.1 회원 정보 조회 기능 구현

다음은 MVC로 구현한 회원 정보 조회 기능을 실행하는 과정입니다.

▼ 그림 17-6 회원 조회 기능 흐름도



- ① 브라우저에서 /mem.do로 요청합니다.
- ② 서블릿 MemberController가 요청을 받아 MemberDAO의 listMembers() 메서드를 호출합니다.
- ③ MemberDAO의 listMembers() 메서드에서 SQL문으로 회원 정보를 조회한 후 회원 정보를 MemberVO에 설정하여 반환합니다.

- ④ 다시 MemberController에서는 조회한 회원 정보를 회원 목록창(listMembers.jsp)으로 포워딩합니다.
- ⑤ 회원 목록창(listMembers.jsp)에서 포워딩한 회원 정보를 목록으로 출력합니다.

그럼 지금부터 회원 정보 조회 기능을 실제로 구현해 보겠습니다.

1. 새 프로젝트 pro17에 sec01.ex01 패키지를 만든 후 MemberController, MemberDAO, MemberVO 클래스를 추가합니다. 그리고 test01 폴더를 만들고 listMembers.jsp를 추가합니다.

▼ 그림 17-7 실습 파일 위치



2. 컨트롤러 역할을 하는 서블릿인 MemberController 클래스를 다음과 같이 작성합니다.

init() 메서드에서 MemberDAO 객체를 초기화하고 MemberDAO의 listMembers() 메서드를 호출하여 회원 정보를 ArrayList로 반환 받습니다. 이때 request에 조회한 회원 정보를 membersList 속성 이름으로 바인딩합니다. 그런 다음 RequestDispatcher 클래스를 이용해 회원 목록창(listMembers.jsp)으로 포워딩합니다.

**코드 17-1 pro17/src/sec01/ex01/MemberController.java**

```
package sec01.ex01;
...
@WebServlet("/mem.do")
public class MemberController extends HttpServlet
{
    private static final long serialVersionUID = 1L;
    MemberDAO memberDAO;

    public void init() throws ServletException
    {
        memberDAO = new MemberDAO(); ----- MemberDAO를 생성합니다.
    }
}
```

```

    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
     *      response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        doHandle(request, response);
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
     *      response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        doHandle(request, response);
    }

    private void doHandle(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        request.setCharacterEncoding("utf-8");
        response.setContentType("text/html;charset=utf-8");
        List<MemberVO> membersList = memberDAO.listMembers();
        request.setAttribute("membersList", membersList);           요청에 대해 회원 정보를 조회합니다.
                                                                조회한 회원 정보를 request에
                                                                바인딩합니다.

        RequestDispatcher dispatch = request.getRequestDispatcher("/test01/listMembers.jsp");
        dispatch.forward(request, response);
    }
}                                         컨트롤러에서 표시하고자 하는 JSP로 포워딩합니다.

```

3. MemberDAO 클래스를 다음과 같이 작성합니다. listMembers() 메서드 호출 시 SQL문을 이용해 회원 정보를 조회한 후 결과를 ArrayList로 반환합니다.

코드 17-2 pro17/src/sec01/ex01/MemberDAO.java

```

package sec01.ex01;
...
public class MemberDAO
{
    private DataSource dataFactory;
    private Connection conn;

```

```

private PreparedStatement pstmt;

public MemberDAO()
{
    try
    {
        Context ctx = new InitialContext();
        Context envContext = (Context) ctx.lookup("java:/comp/env");
        dataFactory = (DataSource) envContext.lookup("jdbc/oracle");
    } catch (Exception e)
    {
        e.printStackTrace();
    }
}

public List<MemberVO> listMembers()
{
    List<MemberVO> membersList = new ArrayList<MemberVO>();
    try
    {
        conn = dataFactory.getConnection();           SQL문을 작성합니다.
        String query = "select * from t_member order by joinDate desc";
        System.out.println(query);
        pstmt = conn.prepareStatement(query);          PrepareStatement 객체를 생성하면서
        ResultSet rs = pstmt.executeQuery();          SQL문을 인자로 전달합니다.

        while (rs.next())
        {
            String id = rs.getString("id");
            String pwd = rs.getString("pwd");
            String name = rs.getString("name");
            String email = rs.getString("email");
            Date joinDate = rs.getDate("joinDate");      조회한 회원 정보를 레코드별로
                                                        MemberVO 객체의 속성에 저장
                                                        합니다.
            MemberVO memberVO = new MemberVO(id, pwd, name, email, joinDate);
            membersList.add(memberVO);                  membersList에 MemberVO 객체들을
                                                        차례대로 저장합니다.
        }
        rs.close();
        pstmt.close();
        conn.close();
    } catch (SQLException e)
    {
        e.printStackTrace();
    }
    return membersList;
}

```

```

public void addMember(MemberVO m)
{
    try
    {
        conn = dataFactory.getConnection();
        String id = m.getId();
        String pwd = m.getPwd();
        String name = m.getName();
        String email = m.getEmail();
        String query = "INSERT INTO t_member(id, pwd, name, email)" + " VALUES(?, ?, ?, ?)";
        System.out.println(query);
        pstmt = conn.prepareStatement(query); ────────── PreparedStatement 객체를 생성하면서
        pstmt.setString(1, id);                      SQL문을 인자로 전달합니다.
        pstmt.setString(2, pwd);
        pstmt.setString(3, name);
        pstmt.setString(4, email);
        pstmt.executeUpdate(); ────────── SQL문을 실행합니다.
        pstmt.close();
        conn.close();
    } catch (SQLException e)
    {
        e.printStackTrace();
    }
}

```

4. MemberVO 클래스를 다음과 같이 작성합니다. 인자 네 개를 갖는 생성자와 인자 다섯 개를 갖는 생성자를 만듭니다.

코드 17-3 pro17/src/sec01/ex01/MemberVO.java

```

package sec01.ex01;

import java.sql.Date;

public class MemberVO
{
    ...
    public MemberVO()
    {
        System.out.println("MemberVO 생성자 호출");
    }

    public MemberVO(String id, String pwd, String name, String email)

```

```

{
    this.id = id;
    this.pwd = pwd;
    this.name = name;
    this.email = email;
}

public MemberVO(String id, String pwd, String name, String email, Date joinDate)
{
    this.id = id;
    this.pwd = pwd;
    this.name = name;
    this.email = email;
    this.joinDate = joinDate;
}

// 각 속성에 대한 getter/setter
// ...

```

---

5. listMembers.jsp를 다음과 같이 작성하여 바인딩된 회원 정보를 차례대로 표시합니다.

**코드 17-4 pro17/WebContent/test01/listMembers.jsp**

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
import=" java.util.* , sec01.ex01.*"
pageEncoding="UTF-8"
isELIgnored="false"
%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
    request.setCharacterEncoding("UTF-8");
%>
<html>
<head>
    <meta charset="UTF-8">
    <title>회원 정보 출력창</title>
    <style>
        .cls1 {
            font-size: 40px;
            text-align: center;
        }
        .cls2 {
            font-size: 20px;
            text-align: center;
        }
    </style>

```

```

        }
    </style>
</head>
<body>
<p class="cls1">회원정보</p>
<table align="center" border="1">
    <tr align="center" bgcolor="lightgreen">
        <td width="7%"><b>아이디</b></td>
        <td width="7%"><b>비밀번호</b></td>
        <td width="7%"><b>이름</b></td>
        <td width="7%"><b>이메일</b></td>
        <td width="7%"><b>가입일</b></td>
    </tr>

    <c:choose>
        <c:when test="${ empty membersList }">
            <tr>
                <td colspan=5>
                    <b>등록된 회원이 없습니다.</b>
                </td>
            </tr>
        </c:when>
        <c:when test="${!empty membersList }">
            <c:forEach var="mem" items="${membersList }">
                <tr align="center">
                    <td>${mem.id }</td>
                    <td>${mem.pwd }</td>
                    <td>${mem.name}</td>
                    <td>${mem.email }</td>
                    <td>${mem.joinDate}</td>
                </tr>
            </c:forEach>
        </c:when>
    </c:choose>
</table>
<a href="#">
    <p class="cls2">회원 가입하기</p>
</a>
</body>
</html>

```

6. <http://localhost:8090/pro17/mem.do>로 요청하여 실행 결과를 확인합니다.

▼ 그림 17-8 실행 결과

회원정보				
아이디	비밀번호	이름	이메일	가입일
ki	1234	기성용	ki@test.com	2018-09-13
kim	1212	김우신	kim@jweb.com	2018-09-04
lee	1212	이순신	lee@test.com	2018-09-04
hong	1212	홍길동	hong@gmail.com	2018-09-04

회원 가입하기

지금까지 회원 정보 조회 기능을 구현해 보았습니다. 하지만 회원 조회만으로는 큰 의미가 없습니다. 지금부터는 새 회원을 추가하고 수정, 삭제할 수 있는 기능까지 MVC 방식으로 차례대로 구현해 보겠습니다.

### 17.3.2 회원 정보 추가 기능 구현

이번에는 컨트롤러에서 회원 정보 조회뿐만 아니라 회원 정보 등록까지 구현해 보겠습니다. 앞에서 보다 브라우저로부터 전달되는 요청 사항이 많아졌기 때문에 우선은 컨트롤러가 브라우저로부터 어떤 요청을 받았는지 알아내야 합니다. 그런 다음 그 요청에 대해 해당하는 모델을 선택하여 작업을 요청해야 하는데, 이 역할을 하는 방법을 커맨드(command) 패턴이라고 합니다.

커맨드 패턴이란 한마디로 브라우저가 URL 패턴을 이용해 컨트롤러에게 수행 작업을 요청하는 방법입니다. 컨트롤러는 `HttpServletRequest`의 `getPathInfo()` 메서드를 이용해 URL 패턴에서 요청명을 받아와 작업을 수행합니다.

URL을 이용해 컨트롤러에 요청하는 형식은 다음과 같습니다. 보통 두 단계로 요청이 이루어집니다.

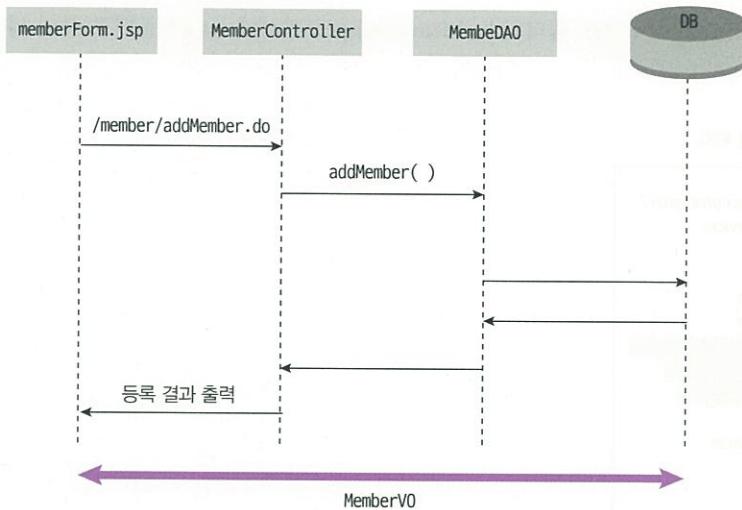
- <http://localhost:8090/pro17/member/listMembers.do>

① /member : 첫 번째 단계의 요청은 회원 기능을 의미합니다.

② /listMembers.do : 두 번째 단계의 요청은 회원 기능 중 회원 조회 기능을 의미합니다.

그림 17-9는 커맨드 패턴을 이용해 회원 가입 기능을 구현하기에 앞서 회원 가입 과정을 그림으로 나타낸 것입니다.

▼ 그림 17-9 회원 정보 등록 기능 흐름도



- ① 회원 가입창에서 회원 정보를 입력하고 URL 패턴을 /member/addMember.do로 서버에 요청합니다.
- ② MemberController에서 getPathInfo() 메서드를 이용해 요청명인 /addMember.do를 받아옵니다.
- ③ 요청명에 대해 MemberDAO의 addMember() 메서드를 호출합니다.
- ④ addMember() 메서드에서 SQL문으로 테이블에 회원 정보를 추가합니다.

1. sec02.ex01 패키지를 만들고 MemberDAO와 MemberVO 클래스는 sec01.ex01 패키지의 것을 복사해 붙여 넣습니다. 그리고 test01 폴더의 listMembers.jsp도 복사해 test02 폴더로 붙여 넣습니다.

▼ 그림 17-10 실습 파일 위치



2. 컨트롤러 역할을 하는 MemberController 클래스를 다음과 같이 작성합니다. 이 컨트롤러에서는 getPathInfo() 메서드를 이용해 두 단계로 이루어진 요청을 가져옵니다. action 값에 따라 if문을 분기해서 요청한 작업을 수행하는데 action 값이 null이거나 /listMembers.do인 경우에 회원 조회 기능을 수행합니다. 만약 action 값이 /memberForm.do면 회원 가입 창을 나타내고 action 값이 /addMember.do면 전송된 회원 정보들을 테이블에 추가합니다.

코드 17-5 pro17/src/sec02/ex01/MemeberController.java

```
package sec02.ex01;  
...  
@WebServlet("/member/*") 브라우저에서 요청 시 두 단계로 요청이 이루어집니다.  
public class MemberController extends HttpServlet  
{  
    private static final long serialVersionUID = 1L;  
    MemberDAO memberDAO;  
  
    public void init() throws ServletException  
    {  
        memberDAO = new MemberDAO();  
    }  
  
    /**  
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
```

```

        *      response)
    */
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
    doHandle(request, response);
}

/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
 *      response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
    doHandle(request, response);
}

private void doHandle(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
    String nextPage = null;
    request.setCharacterEncoding("utf-8");
    response.setContentType("text/html; charset=utf-8");
    String action = request.getPathInfo(); ────────── URL에서 요청명을 가져옵니다.
    System.out.println("action:" + action); ────────── 최초 요청이거나 action 값이 /memberList.do
                                                면 회원 목록을 출력합니다.

    if (action == null || action.equals("/listMembers.do"))
    {
        List<MemberVO> membersList = memberDAO.listMembers();
        request.setAttribute("membersList", membersList);
        nextPage = "/test02/listMembers.jsp"; ────────── test02 폴더의 listMember.jsp로 포워딩합니다.
    } else if (action.equals("/addMember.do")) ────────── action 값이 /addMember.do면 전송된 회원
                                                정보를 가져와서 테이블에 추가합니다.
    {
        String id = request.getParameter("id");
        String pwd = request.getParameter("pwd");
        String name = request.getParameter("name");
        String email = request.getParameter("email");
        MemberVO memberVO = new MemberVO(id, pwd, name, email);
        memberDAO.addMember(memberVO);
        nextPage = "/member/listMembers.do"; ────────── 회원 등록 후 다시 회원 목록을 출력합니다.
    } else if (action.equals("/memberForm.do")) ────────── action 값이 /memberForm.do면 회원
    {                                              가입창을 화면에 출력합니다.
        nextPage = "/test02/memberForm.jsp"; ────────── test02 폴더의 memberForm.jsp로 포워딩합니다.
    }
}

```

```

} else
{
    List<MemberVO> membersList = memberDAO.listMembers();
    request.setAttribute("membersList", membersList);
    nextPage = "/test02/listMembers.jsp";
}
RequestDispatcher dispatch = request.getRequestDispatcher(nextPage);
dispatch.forward(request, response);
}
}

```

그 외 다른 action 값은 회원 목록을 출력합니다.

nextPage에 지정한 요청명으로 다시 서블릿에 요청합니다.

3. 다음과 같이 listMember.jsp에 회원 가입창으로 이동하는 <a> 태그를 추가합니다.

코드 17-6 pro17/WebContent/test02/listMembers.jsp

```
<a href="${contextPath}/member/memberForm.do"><p class="cls2">회원 가입하기</p></a>
```

회원가입하기 클릭 시 서블릿에 /member/memberForm.do로 요청합니다.

4. 회원 가입창에서 회원 정보를 입력하고 action 속성에서 /member/addMember.do로 요청하도록 memberForm.jsp를 작성합니다.

코드 17-7 pro17/WebContent/test02/MemberForm.jsp

```

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

<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="contextPath" value="${pageContext.request.contextPath}" />
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>회원 가입창</title>
<body>
<form method="post" action="${contextPath}/member/addMember.do">
<h1 style="text-align:center">회원 가입창</h1>
<table align="center">
<tr>
<td width="200">
<p align="right">아이디</p>
</td>

```

```
<td width="400"><input type="text" name="id"></td>
</tr>
<tr>
    <td width="200">
        <p align="right">비밀번호
    </td>
    <td width="400"><input type="password" name="pwd"></td>
</tr>
<tr>
    <td width="200">
        <p align="right">이름
    </td>
    <td width="400">
        <p><input type="text" name="name">
    </td>
</tr>
<tr>
    <td width="200">
        <p align="right">이메일
    </td>
    <td width="400">
        <p><input type="text" name="email">
    </td>
</tr>
<tr>
    <td width="200">
        <p>&nbsp;</p>
    </td>
    <td width="400">
        <input type="submit" value="가입하기">
        <input type="reset" value="다시입력">
    </td>
</tr>
</table>
</form>
</body>
</html>
```

5. <http://localhost:8090/pro17/member/listMembers.do>로 요청하여 회원 목록창이 나타나면 하단에 있는 회원 가입하기를 클릭합니다.

▼ 그림 17-11 회원 가입하기 클릭

아이디	비밀번호	이름	이메일	가입일
ki	1234	기성용	ki@test.com	2018-09-13
kim	1212	김유신	kim@jweb.com	2018-09-04
lee	1212	이순신	lee@test.com	2018-09-04
hong	1212	홍길동	hong@gmail.com	2018-09-04

**회원 가입하기**

6. 회원 가입창이 나타나면 다음과 같이 새 회원 차두리의 정보를 입력하고 **가입하기**를 클릭합니다.

▼ 그림 17-12 회원 가입창에서 회원 정보 입력 후 가입하기 클릭

회원 가입창

아이디

비밀번호

이름

이메일

**가입하기**

7. 6번 과정에서 등록한 새 회원(차두리)이 추가된 회원 목록창이 다시 나타납니다.

▼ 그림 17-13 새 회원 추가해 목록 표시

아이디	비밀번호	이름	이메일	가입일
cha2	1212	차두리	cha2@test.com	2018-11-22
ki	1234	기성용	ki@test.com	2018-09-13
kim	1212	김유신	kim@jweb.com	2018-09-04
lee	1212	이순신	lee@test.com	2018-09-04
hong	1212	홍길동	hong@gmail.com	2018-09-04

**회원 가입하기**

### 17.3.3 회원 정보 수정 및 삭제 기능 구현

이번에는 회원 정보를 수정하고 삭제하는 기능을 구현해 보겠습니다. 회원 정보를 수정하는 과정은 다음과 같습니다.

- ❶ 회원 정보 수정창에서 회원 정보를 수정하고 수정하기를 클릭해 /member/modMember.do로 컨트롤러에 요청합니다.
- ❷ 컨트롤러는 전송된 회원 수정 정보를 가져온 후 테이블에서 회원 정보를 수정합니다.
- ❸ 수정을 마친 후 컨트롤러는 다시 회원 목록창을 보여줍니다.

삭제하는 과정도 크게 다르지 않습니다.

- ❶ 회원 목록창에서 삭제를 클릭해 요청명 /member/delMember.do와 회원 ID를 컨트롤러로 전달합니다.
- ❷ 컨트롤러는 request의 getPathInfo() 메서드를 이용해 요청명을 가져옵니다.
- ❸ 회원 ID를 SQL문으로 전달해 테이블에서 회원 정보를 삭제합니다.

1. sec02.ex02 패키지를 만들고 앞에서 사용한 자바 실습 파일들을 붙여 넣습니다. 그리고 test03 폴더를 만들고 JSP 파일들을 붙여 넣습니다.

#### ▼ 그림 17-14 실습 파일 위치



2. 브라우저에서 컨트롤러에 요청하면 request의 getPathInfo() 메서드를 이용해 수정 요청명인 /modMemberForm.do와 /modMember.do를 가져온 후 분기하여 작업을 수행하도록 MemberController 클래스를 다음과 같이 작성합니다.

Tip ☆

소스 코드의 길이가 긴 관계로 주요 부분만 발췌해서 나타내었습니다. 전체 코드는 제공하는 예제 파일을 참고하기 바랍니다.

코드 17-8 pro17/src/sec02/ex02/MemberController.java

```
package sec02.ex02;  
...  
@WebServlet("/member/*")  
public class MemberController extends HttpServlet  
{  
    private static final long serialVersionUID = 1L;  
    MemberDAO memberDAO;  
  
    public void init() throws ServletException  
    {  
        memberDAO = new MemberDAO();  
    }  
  
    ...  
  
    private void doHandle(HttpServletRequest request, HttpServletResponse response)  
    throws ServletException, IOException  
    {  
        String nextPage = null;  
        request.setCharacterEncoding("utf-8");  
        response.setContentType("text/html;charset=utf-8");  
        String action = request.getPathInfo();  
        System.out.println("action:" + action);  
  
        if (action == null || action.equals("/listMembers.do"))  
        {  
            ...  
        } else if (action.equals("/modMemberForm.do"))  
        {  
            String id = request.getParameter("id");  
            MemberVO memInfo = memberDAO.findMember(id);  
            request.setAttribute("memInfo", memInfo);  
            nextPage = "/test03/modMemberForm.jsp";  
        } else if (action.equals("/modMember.do"))  
        {  
            ...  
        }  
    }  
}
```

회원 수정창 요청 시 ID로 회원 정보를 조회한 후 수정창으로 포워딩합니다.

회원 정보 수정창을 요청하면서 전송된 ID를 이용해 수정 전 회원 정보를 조회합니다.

request에 바인딩하여 회원 정보 수정창에 수정하기 전 회원 정보를 전달합니다.

테이블의 회원 정보를 수정합니다.

```

String id = request.getParameter("id");
String pwd = request.getParameter("pwd");
String name = request.getParameter("name");
String email = request.getParameter("email");
MemberVO memberVO = new MemberVO(id, pwd, name, email);
memberDAO.modMember(memberVO);
request.setAttribute("msg", "modified");
nextPage = "/member/listMembers.do";
} else if (action.equals("/delMember.do"))
{
    String id = request.getParameter("id");
    memberDAO.delMember(id);
    request.setAttribute("msg", "deleted");
    nextPage = "/member/listMembers.do";
} else
{
    List<MemberVO> membersList = memberDAO.listMembers();
    request.setAttribute("membersList", membersList);
    nextPage = "/test03/listMembers.jsp";
}
RequestDispatcher dispatch = request.getRequestDispatcher(nextPage);
dispatch.forward(request, response);
}
}

```

회원 정보 수정창에서 전송된 수정 회원 정보를 가져온 후 MemberVO 객체 속성에 설정 합니다.

회원 목록창으로 수정 작업 완료 메시지를 전달합니다.

회원 ID를 SQL문으로 전달해 테이블의 회원 정보를 삭제합니다.

삭제할 회원 ID를 받아옵니다.

회원 목록창으로 삭제 작업 완료 메시지를 전달합니다.

3. MemberDAO 클래스를 다음과 같이 작성합니다. 회원 ID를 이용해 회원 정보를 조회하고, 수정 회원 정보를 갱신하고, 회원 ID로 회원 정보를 삭제하는 메서드를 추가합니다.

코드 17-9 pro17/src/sec02/ex02/MemberDAO.java

```

package sec02.ex02;
...
public class MemberDAO
{
    ...
    public MemberVO findMember(String _id)
    {
        MemberVO memInfo = null;
        try
        {
            conn = dataFactory.getConnection();
            String query = "select * from t_member where id=?"; ← 전달된 ID로 회원 정보
            pstmt = conn.prepareStatement(query); ← 를 조회합니다.
            pstmt.setString(1, _id);

```

```
System.out.println(query);
ResultSet rs = pstmt.executeQuery();
rs.next();
String id = rs.getString("id");
String pwd = rs.getString("pwd");
String name = rs.getString("name");
String email = rs.getString("email");
Date joinDate = rs.getDate("joinDate");
memInfo = new MemberVO(id, pwd, name, email, joinDate);
pstmt.close();
conn.close();
} catch (Exception e)
{
    e.printStackTrace();
}
return memInfo;
}

public void modMember(MemberVO memberVO)
{
    String id = memberVO.getId();
    String pwd = memberVO.getPwd();
    String name = memberVO.getName();
    String email = memberVO.getEmail();
    try
    {
        conn = dataFactory.getConnection();
        String query = "update t_member set pwd=?,name=?,email=? where id=?";
        System.out.println(query);
        pstmt = conn.prepareStatement(query);
        pstmt.setString(1, pwd);
        pstmt.setString(2, name);
        pstmt.setString(3, email);
        pstmt.setString(4, id);
        pstmt.executeUpdate(); ← 전달된 수정 회원 정보를 update문을
                               이용해 수정합니다.
        pstmt.close();
        conn.close();
    } catch (Exception e)
    {
        e.printStackTrace();
    }
}

public void delMember(String id)
{
```

```

try
{
    conn = dataFactory.getConnection();
    String query = "delete from t_member where id=?";
    System.out.println(query);
    pstmt = conn.prepareStatement(query);
    pstmt.setString(1, id);
    pstmt.executeUpdate(); ← SQL 문을 실행합니다.
} catch (Exception e)
{
    e.printStackTrace();
}
}

}

```

4. listMembers.jsp를 다음과 같이 작성합니다. 회원 추가, 수정, 삭제 작업 후에 다시 회원 목록 창을 요청할 경우 컨트롤러에서 가져온 msg 값에 따라 작업 결과를 alert 창에 출력합니다.

코드 17-10 pro17/WebContent/test03/listMembers.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"
isELIgnored="false"
%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="contextPath" value="${pageContext.request.contextPath}" />
<%
request.setCharacterEncoding("UTF-8");
%>
<html>

<head>
<c:choose>
<c:when test='${msg=="addMember" }'>
<script>
window.onload = function () {
    alert("회원을 등록했습니다.");
}
</script>
</c:when>
<c:when test='${msg=="modified" }'>
<script>

```

회원 추가, 수정, 삭제 작업 후 컨트롤러에서 넘긴 msg 값에 따라 작업 결과를 alert 창에 출력합니다.

```

        window.onload = function () {
            alert("회원 정보를 수정했습니다.");
        }
    </script>
</c:when>
<c:when test='${msg=="deleted"}'>
    <script>
        window.onload = function () {
            alert("회원 정보를 삭제했습니다.");
        }
    </script>
</c:when>
</c:choose>

```

```

<meta charset="UTF-8">
<title>회원 정보 출력창</title>
<style>
    .cls1 {
        font-size: 40px;
        text-align: center;
    }

    .cls2 {
        font-size: 20px;
        text-align: center;
    }
</style>

</head>

<body>
    <p class="cls1">회원정보</p>
    <table align="center" border="1">
        <tr align="center" bgcolor="lightgreen">
            <td width="7%"><b>아이디</b></td>
            <td width="7%"><b>비밀번호</b></td>
            <td width="7%"><b>이름</b></td>
            <td width="7%"><b>이메일</b></td>
            <td width="7%"><b>가입일</b></td>
            <td width="7%"><b>수정</b></td>
            <td width="7%"><b>삭제</b></td>
        </tr>
</c:choose>

```

```

<c:when test="${ empty membersList }">
    <tr>
        <td colspan=5>
            <b>등록된 회원이 없습니다.</b>
        </td>
    </tr>
</c:when>
<c:when test="${!empty membersList }">
    <c:forEach var="mem" items="${membersList }">
        <tr align="center">
            <td>${mem.id }</td>
            <td>${mem.pwd }</td>
            <td>${mem.name}</td>
            <td>${mem.email }</td>
            <td>${mem.joinDate}</td>
            <td><a href="${contextPath}/member/modMemberForm.do?id=${mem.id }">수정</a></td>
            <td><a href="${contextPath}/member/delMember.do?id=${mem.id }">삭제</a></td>
        </tr>
    </c:forEach>
</c:when>
</c:choose>
</table>
<a href="${contextPath}/member/memberForm.do">
    <p class="cls2">회원 가입하기</p>
</a>
</body>

</html>

```

5. modMemberForm.jsp를 다음과 같이 작성합니다. 회원 정보 수정창에서 수정된 회원 정보를 입력하고 수정하기를 클릭하면 action 속성에 설정한 요청명 /member/modMember.do와 회원 ID를 전달해 수정하도록 구현합니다.

코드 17-11 pro17/WebContent/test03/modMemberForm.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"
    isELIgnored="false" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="contextPath" value="${pageContext.request.contextPath}" />
<%
    request.setCharacterEncoding("UTF-8");

```

```

.>

<head>
    <meta charset="UTF-8">
    <title>회원 정보 수정창</title>
    <style>
        .cls1 {
            font-size:40px;
            text-align:center;
        }
    </style>
</head>

<body>
    <h1 class="cls1">회원 정보 수정창</h1> 수정하기 클릭 시 컨트롤러에 /member/modMember.do로 요청합니다.
    <form method="post" action="${contextPath}/member/modMember.do?id=${memInfo.id}">
        <table align="center">
            <tr>
                <td width="200">
                    <p align="right">아이디
                </td>
                <td width="400"><input type="text" name="id" value="${memInfo.id}" disabled></td>
            </tr>
            <tr>
                <td width="200">
                    <p align="right">비밀번호
                </td>
                <td width="400"><input type="password" name="pwd" value="${memInfo.pwd}">
            </td>
            </tr>
            <tr>
                <td width="200">
                    <p align="right">이름
                </td>
                <td width="400"><input type="text" name="name" value="${memInfo.name}"></td>
            </tr>
            <tr>
                <td width="200">
                    <p align="right">이메일
                </td>
                <td width="400"><input type="text" name="email" value="${memInfo.email}"></td>
            </tr>
            <tr>
                <td width="200">
                    <p align="right">가입일
                </td>

```

조회한 회원 정보를 텍스트 박스에 표시합니다.

조회한 회원 정보를 텍스트 박스에 표시합니다.

```

</td>
<td width="400"><input type="text" name="joinDate"
value="${memInfo.joinDate }" disabled></td>
</tr>
<tr align="center">
<td colspan="2" width="400"><input type="submit" value="수정하기">
<input type="reset" value="다시입력"> </td>
</tr>
</table>
</form>
</html>

```

6. 회원 목록창에서 차두리의 회원 정보를 수정해 볼까요? 수정을 클릭하여 <http://localhost:8090/pro17/member/modMemberForm.do>로 요청합니다.

▼ 그림 17-15 회원 목록창에서 수정 클릭



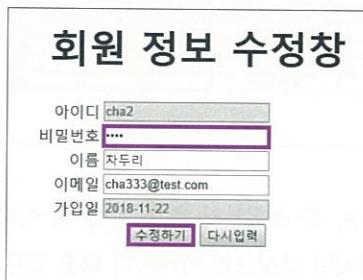
The screenshot shows a table of member information with 7 columns: 아이디 (ID), 비밀번호 (Password), 이름 (Name), 이메일 (Email), 가입일 (Join Date), 수정 (Modify), and 삭제 (Delete). The '수정' column for the user '차두리' contains a purple-bordered button labeled '수정'. The table rows are as follows:

아이디	비밀번호	이름	이메일	가입일	수정	삭제
cha2	1212	차두리	cha2@test.com	2018-11-22	<span style="border: 2px solid #800080;">수정</span>	삭제
ki	1234	기성동	ki@test.com	2018-09-13	<span style="border: 2px solid #800080;">수정</span>	삭제
kim	1212	김유신	kim@jweb.com	2018-09-04	<span style="border: 2px solid #800080;">수정</span>	삭제
lee	1212	이순신	lee@test.com	2018-09-04	<span style="border: 2px solid #800080;">수정</span>	삭제
hong	1212	홍길동	hong@gmail.com	2018-09-04	<span style="border: 2px solid #800080;">수정</span>	삭제

[회원 가입하기](#)

7. 회원 정보 수정창이 나타납니다. 차두리의 회원 정보(비밀번호, 이메일 주소)를 다음과 같이 수정하고 수정하기를 클릭한 후 <http://localhost:8090/pro17/member/modMember.do>로 요청합니다.

▼ 그림 17-16 회원 정보 수정 후 수정하기 클릭



The screenshot shows a '회원 정보 수정창' (Member Information Modification Form) with the following fields:

- 아이디: cha2
- 비밀번호: ....
- 이름: 차두리
- 이메일: cha333@test.com
- 가입일: 2018-11-22
- Buttons: 수정하기 (Modify) and 다시입력 (Re-enter)

8. ‘수정 완료’ 메시지가 나타나고 다음과 같이 수정된 회원 목록을 표시합니다. 차두리의 회원 정보가 제대로 수정되었나요?

▼ 그림 17-17 수정된 회원 목록 표시

회원정보						
아이디	비밀번호	이름	이메일	가입일	수정	삭제
cha2	4321	차두리	cha333@test.com	2018-11-22	수정	삭제
ki	1234	기성용	ki@test.com	2018-09-13	수정	삭제
kim	1212	김유신	kim@jweb.com	2018-09-04	수정	삭제
lee	1212	이순신	lee@test.com	2018-09-04	수정	삭제
hong	1212	홍길동	hong@gmail.com	2018-09-04	수정	삭제

[회원 가입하기](#)

9. 이번에는 차두리의 회원 정보를 삭제해 보겠습니다. 다음과 같이 삭제를 클릭합니다.

▼ 그림 17-18 회원 목록창에서 차두리의 삭제 클릭

회원정보						
아이디	비밀번호	이름	이메일	가입일	수정	삭제
cha2	3333	차두리	cha333@test.com	2018-11-12	수정	삭제
ki	1234	기성용	ki@test.com	2018-09-13	수정	삭제
kim	1212	김유신	kim@jweb.com	2018-09-04	수정	삭제
lee	1212	이순신	lee@test.com	2018-09-04	수정	삭제
hong	1212	홍길동	hong@gmail.com	2018-09-04	수정	삭제

[회원 가입하기](#)

10. 그러면 ‘삭제 완료’ 메시지가 나타난 후 다시 회원 목록을 표시합니다. 목록에서 차두리가 사라진 것을 볼 수 있습니다.

▼ 그림 17-19 삭제 후 회원 목록 다시 표시

회원정보						
아이디	비밀번호	이름	이메일	가입일	수정	삭제
ki	1234	기성용	ki@test.com	2018-09-13	수정	삭제
kim	1212	김유신	kim@jweb.com	2018-09-04	수정	삭제
lee	1212	이순신	lee@test.com	2018-09-04	수정	삭제
hong	1212	홍길동	hong@gmail.com	2018-09-04	수정	삭제

[회원 가입하기](#)

지금까지 커맨드 패턴을 이용해 기본적인 회원 관리 기능, 즉 조회부터 추가, 수정 및 삭제까지 MVC로 구현해 봤습니다. 다음 절에서는 JSP 학습의 마지막 과정으로 이제까지 배운 모든 서블릿과 JSP 기능을 활용하여 모델2 기반의 답변형 게시판을 구현해 보겠습니다.

## 17.4

## 모델2로 답변형 게시판 구현하기

게시판 기능은 모든 웹 페이지의 기본 기능을 포함하기 때문에 게시판을 만들 수 있다면 모든 웹 페이지를 쉽게 만들 수 있습니다.

이번에 우리가 실습을 통해 구현할 답변형 게시판의 글 목록은 그림 17-20과 같은 형태입니다. 부모 글이 목록에 나열되면 각 부모 글에 대해 답변 글(자식 글)이 계층 구조로 나열되어 표시됩니다. 그리고 답변 글에 대한 답변 글은 또 다시 계층 구조로 표시됩니다. 즉, 답변 글에 또 답변 글을 올릴 수 있는 기능을 하는 게시판입니다.

▼ 그림 17-20 답변형 게시판 실제 구현 화면

글번호	작성자	제목	작성일
1	hong	상품평	2018. 9. 18.
2	hong	상품평가	2018. 9. 18.
3	hong	상품 주문이 늦어요.	2018. 9. 18.
4	lee	[답변] 죄송합니다.	2018. 9. 18.
5	hong	상품 판입니다.	2018. 9. 18.
6	hong	최길동글입니다.	2018. 9. 18.
7	kim	김우신입니다.	2018. 9. 18.
8	lee	[답변] 이용 후기입니다.	2018. 9. 18.
9	hong	안녕하세요	2018. 9. 18.
10	lee	[답변] 상품후기입니다..	2018. 9. 18.

1 2 3 4 5 6 7 8 9 10 next

글쓰기

표 17-1은 답변형 게시판 글을 저장하는 테이블 컬럼입니다. 게시판의 글을 작성하려면 회원이 로그인 상태여야 합니다. 즉, 각 글에는 작성자 ID가 저장됩니다. 따라서 게시판 테이블의 ID 컬럼은 회원 테이블의 ID 컬럼에 대해 외래키를 속성으로 가집니다.

▼ 표 17-1 답변형 게시판 테이블(t\_board) 구조

no	컬럼 이름	속성	자료형	크기	유일키 여부	NULL 여부	키	기본값
1	articleNO	글 번호	number	10	Y	N	기본키	
2	parentNO	부모 글 번호	number	10	N	N		0
3	title	글 제목	varchar2	100	N	N		
4	content	글 내용	varchar2	4000	N	N		

no	컬럼 이름	속성	자료형	크기	유일키 여부	NULL 여부	키	기본값
5	imageFileName	이미지 파일 이름	varchar2	100	N			
6	writeDate	작성일	date		N	N		sysdate
7	id	작성자 ID	varchar2	20	N	N	외래키	

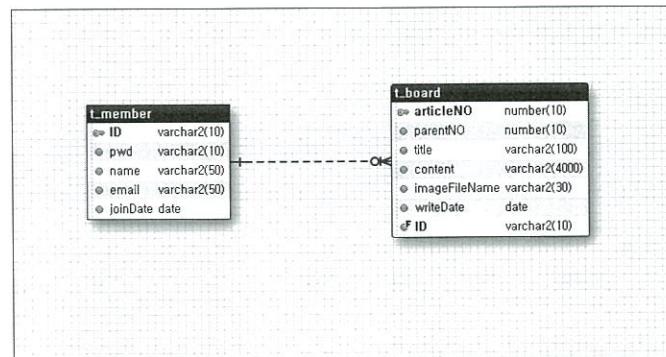
또한 다음은 테이블 컬럼의 주요 속성들입니다. 여기서 부모 글 번호는 답변을 단 글 번호를 의미합니다. 자신이 최초 글이면 부모 글 번호는 0입니다. ID는 작성자의 ID를 의미하고, ID 컬럼은 항상 t\_member 테이블의 ID 컬럼을 참조합니다.

▼ 표 17-2 테이블의 주요 속성

속성	컬럼	설명
글 번호	articleNO	글이 추가될 때마다 1씩 증가되면서 고유 값을 부여합니다.
부모 글 번호	parentNO	답변을 단 부모 글의 번호를 나타냅니다. 부모 글 번호가 0이면 자신이 부모 글입니다.
첨부 파일 이름	imageFileName	글 작성 시 첨부한 이미지 파일 이름입니다.
작성자 ID	id	글을 작성한 작성자의 ID입니다.

그림 17-21을 보면 게시판 테이블이 회원 테이블에 대해서 참조 관계가 있음을 알 수 있습니다.

▼ 그림 17-21 게시판 테이블의 참조 관계



우리가 알고 있는 일반적인 게시판처럼 부모 글에 대한 답변 글을 계층 구조로 나타내려면 어떻게 해야 할까요?

가장 쉬운 방법은 각 글을 테이블에 추가할 때 해당 글의 부모 글 번호(parentNO)를 같이 등록한 후 글 조회 시 부모 글 번호에 대해 계층형 SQL문을 수행하여 계층 구조로 표시하는 것입니다.

먼저 SQL Developer를 이용해 다음과 같이 테이블을 생성하고 테스트 글을 테이블에 추가합니다.

#### 코드 17-12 게시판 기능 테이블 생성 및 데이터 추가하기

```

DROP TABLE t_Board CASCADE CONSTRAINTS;

--게시판 테이블을 생성합니다.
create table t_Board(
    articleNO number( 10 ) primary key,
    parentNO number(10) default 0,
    title varchar2( 500 ) not null,
    content varchar2( 4000 ),
    imageFileName varchar2(100),
    writedate date default sysdate not null ,
    id varchar2(10),
    CONSTRAINT FK_ID FOREIGN KEY(id) REFERENCES t_member(id)
);

-- 테이블에 테스트 글을 추가합니다.
insert into t_board(articleNO, parentNO, title, content, imageFileName, writedate, id)
values(1, 0, '테스트글입니다.', '테스트글입니다.', null, sysdate, 'hong');

insert into t_board(articleNO, parentNO, title, content, imageFileName, writedate, id)
values(2, 0,'안녕하세요', '상품 후기입니다.', null,sysdate, 'hong' );

insert into t_board(articleNO, parentNO, title, content, imageFileName, writedate, id)
values(3, 2,'답변입니다.', '상품 후기에 대한 답변입니다.', null, sysdate, 'hong' );

insert into t_board(articleNO, parentNO, title, content, imageFileName, writedate, id)
values(5, 3,'답변입니다.', '상품 좋습니다.', null,sysdate, 'lee' );

insert into t_board(articleNO, parentNO, title, content, imageFileName, writedate, id)
values(4, 0, '김유신입니다.', '김유신 테스트글입니다.', null, sysdate, 'kim');

insert into t_board(articleNO, parentNO, title, content, imageFileName, writedate, id)
values(6, 2, '상품 후기입니다..', '이순신씨의 상품 사용 후기를 올립니다!!', null, sysdate, 'lee' );

commit; -- 추가 후 반드시 커밋을 해줍니다.

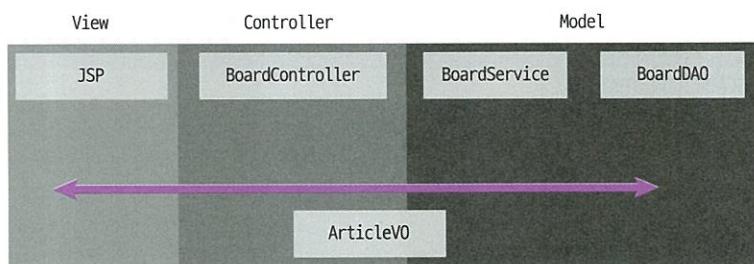
select * from t_board;

```

ID 컬럼을 회원 테이블의 ID 컬럼에 대해 외래키로 지정합니다.

본격적으로 코드를 작성하기에 앞서 답변형 게시판의 기능을 담당하는 클래스와 JSP의 MVC 구조를 살펴보겠습니다(그림 17-22).

▼ 그림 17-22 답변형 게시판 MVC 구조



뷰와 컨트롤러는 그대로 JSP와 서블릿이 기능을 수행하지만 모델은 기존의 DAO 클래스 외에 `BoardService` 클래스가 추가된 것을 볼 수 있습니다.

지금까지 MVC로 기능을 구현할 때 모델 기능은 DAO 클래스가 수행했습니다. 하지만 실제로 개발을 할 때는 Service 클래스를 거쳐서 DAO 클래스의 기능을 수행하도록 구현합니다. 그렇다면 모델에서 Service 클래스를 두는 이유는 무엇일까요?

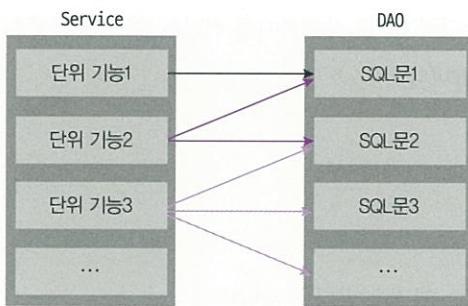
DAO는 데이터베이스에 접근하는 기능을 수행하고 Service는 실제 프로그램을 업무에 적용하는 사용자 입장에서 업무 단위, 즉 트랜잭션(Transaction)으로 작업을 수행합니다. 여기서 업무 단위란 ‘단위 기능’이라고도 하며, 사용자 입장에서 하나의 논리적인 기능을 의미합니다.

웹 애플리케이션에서 일반적으로 묶어서 처리하는 단위 기능에는 다음과 같은 것들이 있습니다.

- 게시판 글 조회 시 해당 글을 조회하는 기능과 조회 수를 갱신하는 기능
- 쇼핑몰에서 상품 주문 시 주문 상품을 테이블에 등록 후 주문자의 포인트를 갱신하는 기능
- 은행에서 송금 시 송금자의 잔고를 갱신하는 기능과 수신자의 잔고를 갱신하는 기능

그림 17-23처럼 단위 기능1은 자신의 기능을 수행할 때 DAO와 연동해 한 개의 SQL문으로 기능을 수행하지만 단위 기능2나 단위 기능3은 여러 SQL문을 묶어서 하나의 단위 기능을 수행합니다.

▼ 그림 17-23 일반적인 모델 구조



실제 개발을 할 때에도 **Service** 클래스의 메서드를 이용해 큰 기능을 단위 기능으로 나눈 후 **Service** 클래스의 각 메서드는 자신의 기능을 더 세부적인 기능을 하는 **DAO**의 SQL문들을 조합해서 구현합니다. 이렇게 하는 이유는 유지보수나 시스템의 확장성 면에서 훨씬 유리하기 때문입니다.

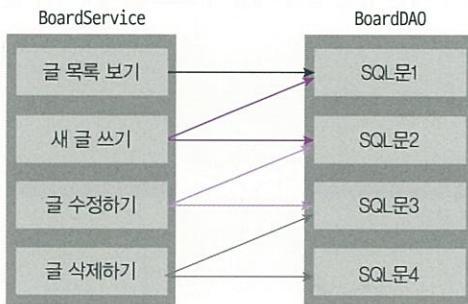
예를 들어 가장 흔한 게시판 기능은 크게 다음과 같이 나눌 수 있습니다.

- 새 글 쓰기
- 글 보기
- 글 수정하기
- 글 삭제하기

즉, 각 글과 관련해 세부 기능을 수행하는 SQL문들을 **DAO**에서 구현하고, **Service** 클래스의 단위 기능 메서드에서 **DAO**에 만들어 놓은 SQL문들을 조합해서 단위 기능을 구현하는 것입니다.

그림 17-24는 실제 답변형 게시판의 모델 구조를 나타낸 것입니다.

▼ 그림 17-24 답변형 게시판 기능의 모델 구조

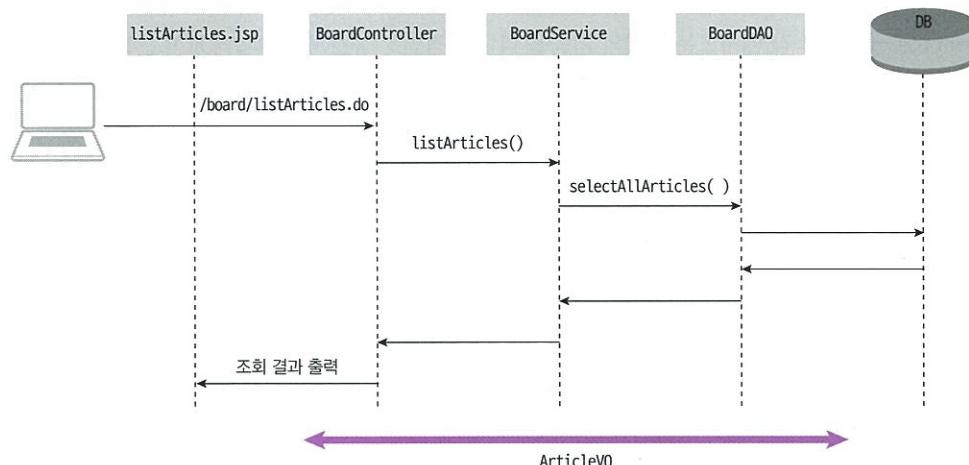


보통 게시판의 Service 클래스 기능은 BoardDAO의 각 메서드의 SQL문 조합으로 기능을 구현합니다. 그럼 지금부터 글 목록 보기, 새 글쓰기, 글 수정하기, 글 삭제하기를 하나씩 차례대로 실습해 보겠습니다. 먼저 글 목록을 보는 기능부터 시작합니다.

### 17.4.1 게시판 글 목록 보기 구현

그림 17-25는 게시판의 글 목록 보기 기능을 구현하는 과정을 나타낸 것입니다.

▼ 그림 17-25 글 목록 보기 흐름도



브라우저에서 `/board/listArticles.do`로 요청하면 Controller가 전달받아 Service와 DAO를 거쳐 글 정보를 조회한 후 `listArticles.jsp`로 전달하여 화면에 글 목록을 보여줍니다. 여기서 문제는 글 목록을 그냥 나열만 하는 것이 아니라 부모 글에 대한 답변 글을 계층 구조로 보여주어야 한다는 것입니다. 이렇게 하려면 어떻게 해야 할까요?

오라클에서 제공하는 계층형 SQL문 기능을 이용하면 이를 구현할 수 있습니다.

코드 17-13 글 목록 조회 계층형 SQL문

```
SELECT LEVEL, _____ 오라클에서 제공하는 가상 컬럼으로 글의  
      articleNO,      깊이를 나타냅니다(부모 글은 1입니다).  
      parentNO,  
      LPAD(' ', 4*(LEVEL-1)) || title  title,  
      content,  
      writeDate,  
      id
```

```

FROM t_board
START WITH parentNO=0 ①
CONNECT BY PRIOR articleNO=parentNO ②
ORDER SIBLINGS BY articleNO DESC; ③

```

SQL문의 각 문법을 조금 들여다볼까요?

- ① 계층형 구조에서 최상위 계층의 로우(row)를 식별하는 조건을 명시합니다. parentNO가 0, 즉 부모 글부터 시작해 계층형 구조를 만든다는 의미입니다.
- ② 계층 구조가 어떤 식으로 연결되는지를 기술하는 부분입니다. parentNO에 부모 글 번호가 있으므로 이를 표현하려면 CONNECT BY PRIOR articleNO = parentNO로 기술해야 합니다.
- ③ 계층 구조로 조회된 정보를 다시 articleNO를 이용해 내림차순으로 정렬하여 최종 출력합니다.

코드 17-13과 같은 계층형 SQL문을 SQL Developer에서 실행하면 그림 17-26처럼 자식 글이 부모 글 아래에 출력됩니다.

▼ 그림 17-26 계층형 SQL문 실행 결과

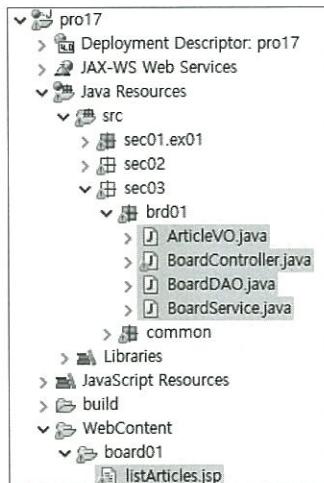
LEVEL	ARTICLENO	PARENTNO	TITLE	CONTENT	WRITEDATE	ID
1	1	4	0 김유진입니다.	부모 글에 대한 자식 글이 계층형	18/09/18	kim
2	1	2	0 안녕하세요	으로 표시되어 있습니다.	18/09/18	hong
3	2	6	2 상품후기입니다..	입니다.	18/09/18	lee
4	2	3	2 답변입니다.	상품	18/09/18	hong
5	3	5	3 답변입니다.	좋습니다.	18/09/18	lee
6	1	1	0 테스트글입니다.	테스트글입니다.	18/09/18	hong

**Note** 계층형 SQL문에 관한 자세한 설명은 오리를 관련 교재나 인터넷 검색을 통해 참고하기 바랍니다.

그럼 실제로 클래스와 JSP를 사용해 이를 구현해 보겠습니다.

- sec03.brd01 패키지를 새로 만들고 관련된 클래스를 추가합니다. 또한 board01 폴더를 만들고 listArticles.jsp를 추가합니다.

#### ▼ 그림 17-27 실습 파일 위치



2. BoardController 클래스를 다음과 같이 작성합니다. 이 클래스는 /board/listArticles.do로 요청 시 화면에 글 목록을 출력하는 역할을 합니다. getPathInfo() 메서드를 이용해 action 값을 가져오고 action 값이 null이거나 /listArticles.do일 경우 BoardService 클래스의 listArticles() 메서드를 호출해 전체 글을 조회합니다. 그리고 조회한 글을 articlesList 속성으로 바인딩하고 글 목록창(listArticles.jsp)으로 포워딩합니다.

코드 17-14 pro17/src/sec03/brd01/BoardController.java

```

    throws ServletException, IOException
    {
        doHandle(request, response);
    }

protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
    doHandle(request, response);
}

private void doHandle(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
    String nextPage = "";
    request.setCharacterEncoding("utf-8");
    response.setContentType("text/html; charset=utf-8");
    String action = request.getPathInfo(); ────────── 요청명을 가져옵니다.
    System.out.println("action:" + action);
    try
    {
        List<ArticleVO> articlesList = new ArrayList<ArticleVO>();
        if (action == null)
        {
            articlesList = boardService.listArticles();
            request.setAttribute("articlesList", articlesList);
            nextPage = "/board01/listArticles.jsp";
        } else if (action.equals("/listArticles.do")) ────────── action 값이 /listArticles.do이면
        {                                            전체 글을 조회합니다.
            articlesList = boardService.listArticles(); ────────── 전체 글을 조회합니다.
            request.setAttribute("articlesList", articlesList);
            nextPage = "/board01/ listArticles.jsp"; ────────── 조회된 글 목록을 articlesList로 바인딩한
        }                                            후 listArticles.jsp로 포워딩합니다.
        RequestDispatcher dispatch = request.getRequestDispatcher(nextPage);
        dispatch.forward(request, response);
    } catch (Exception e)
    {
        e.printStackTrace();
    }
}

```

3. BoardService 클래스를 다음과 같이 작성합니다. BoardDAO 객체를 생성한 후 selectAllArticle() 메서드를 호출해 전체 글을 가져옵니다.

코드 17-15 pro17/src/sec03/brd01/BoardService.java

```
package sec03.brd01;  
...  
public class BoardService  
{  
    BoardDAO boardDAO;  
  
    public BoardService()  
    {  
        boardDAO = new BoardDAO(); ← 생성자 호출 시 BoardDAO 객체를 생성합니다.  
    }  
  
    public List<ArticleVO> listArticles()  
    {  
        List<ArticleVO> articlesList = boardDAO.selectAllArticles();  
        return articlesList;  
    }  
}
```

**Note** ≡ BoardDAO 클래스의 메서드 이름은 보통 각 메서드들이 실행하는 SQL문에 의해 결정됩니다. 예를 들어 selectAllArticles() 메서드는 전체 글 정보를 조회하는 SQL문을 실행하므로 메서드 이름에 selectAll이 들어갑니다.

4. BoardDAO 클래스를 다음과 같이 작성합니다. BoardService 클래스에서 BoardDAO의 selectAllArticles() 메서드를 호출하면 계층형 SQL문을 이용해 계층형 구조로 전체 글을 조회한 후 반환합니다.

코드 17-16 pro17/src/sec03/brd01/BoardDAO.java

```
package sec03.brd01;  
...  
public class BoardDAO  
{  
    private DataSource dataFactory;  
    Connection conn;  
    PreparedStatement pstmt;
```

```

public BoardDAO()
{
    try
    {
        Context ctx = new InitialContext();
        Context envContext = (Context) ctx.lookup("java:/comp/env");
        dataFactory = (DataSource) envContext.lookup("jdbc/oracle");
    } catch (Exception e)
    {
        e.printStackTrace();
    }
}

public List<ArticleVO> selectAllArticles()
{
    List<ArticleVO> articlesList = new ArrayList();
    try
    {
        conn = dataFactory.getConnection();
        String query = "SELECT LEVEL, articleNO, parentNO,
                       title, content, id, writeDate"
                      + " from t_board"
                      + " START WITH parentNO=0"
                      + " CONNECT BY PRIOR articleNO=parentNO"
                      + " ORDER SIBLINGS BY articleNO DESC";
    }

    System.out.println(query);
    pstmt = conn.prepareStatement(query);
    ResultSet rs = pstmt.executeQuery();
    while (rs.next())
    {
        int level = rs.getInt("level");          ← 각 글의 깊이(계층)를 level 속성에 저장합니다.
        int articleNO = rs.getInt("articleNO");   ← 글 번호는 숫자형이므로 getInt()로 값을
        int parentNO = rs.getInt("parentNO");     ← 가져옵니다.
        String title = rs.getString("title");
        String content = rs.getString("content");
        String id = rs.getString("id");
        Date writeDate = rs.getDate("writeDate");
        ArticleVO article = new ArticleVO();
        article.setLevel(level);
        article.setArticleNO(articleNO);
        article.setParentNO(parentNO);
        article.setTitle(title);
        article.setContent(content);
        article.setId(id);
    }
}

```

오리클의 계층형 SQL문을 실행합니다.

글 정보를 ArticleVO 객체의 속성에 설정합니다.

```
        article.setWriteDate(writeDate);
        articlesList.add(article);
    }
    rs.close();
    pstmt.close();
    conn.close();
} catch (Exception e)
{
    e.printStackTrace();
}
return articlesList;
}
}
```

---

5. ArticleVO 클래스를 다음과 같이 작성합니다. 조회한 글을 저장하는 ArticleVO 클래스에 글의 깊이를 저장하는 level 속성을 추가합니다.

코드 17-17 pro17/src/sec03/brd01/ArticleVO.java

```
package sec03.brd01;
...
public class ArticleVO
{
    private int level;
    private int articleNO;
    private int parentNO;
    private String title;
    private String content;
    private String imageName;
    private String id;
    private Date writeDate;

    public ArticleVO()
    {
    }

    public ArticleVO(int level, int articleNO, int parentNO, String title, String content,
                     String imageName, String id) {
        this.level = level;
        this.articleNO = articleNO;
        this.parentNO = parentNO;
        this.title = title;
        this.content = content;
        this.imageName = imageName;
```

```

        this.id = id;
    }
    // 각 속성에 대한 getter/setter
    ...
    public void setImageFileName(String imageName) { // 파일 이름에 특수 문자가 있을
        경우 인코딩
    ...
}

```

6. 이제 JSP에 글 목록을 표시해 보겠습니다. listArticles.jsp를 다음과 같이 작성합니다. 첫 번째 <forEach> 태그를 이용해 articlesList 속성으로 포워딩된 글 목록을 차례로 전달받아 표시합니다. <forEach> 태그 반복 시 각 글의 level 값이 1보다 크면 답글이므로 다시 내부 <forEach> 태그를 이용해 1부터 level 값까지 반복하면서 공백을 만들고(들여쓰기) 답글을 표시합니다. 이때 level 값이 1보다 크지 않으면 부모 글이므로 공백 없이 표시합니다.

**코드 17-18 pro17/WebContent/board01/listArticles.jsp**

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"
    isELIgnored="false" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:set var="contextPath" value="${pageContext.request.contextPath}" />
<%
    request.setCharacterEncoding("UTF-8");
%>
<!DOCTYPE html>
<html>
<head>
<style>
    .cls1 {
        text-decoration: none;
    }
    .cls2 {
        text-align: center;
        font-size: 30px;
    }
</style>
<meta charset="UTF-8">
<title>글목록창</title>
</head>
<body>
    <table align="center" border="1" width="80%">

```

```

<tr height="10" align="center" bgcolor="lightgreen">
    <td>글번호</td>
    <td>작성자</td>
    <td>제목</td>
    <td>작성일</td>
</tr>
<c:choose>
    <c:when test="${empty articlesList }">
        <tr height="10">
            <td colspan="4">
                <p align="center">
                    <b><span style="font-size:9pt;">등록된 글이 없습니다.</span></b>
                </p>
            </td>
        </tr>
    </c:when>
    <c:when test="${!empty articlesList }">
        <c:forEach var="article" items="${articlesList }" varStatus="articleNum">
            <tr align="center">
                <td width="5%">${articleNum.count}</td>
                <td width="10%">${article.id }</td>
                <td align='left' width="35%">
                    <span style="padding-right:30px"></span>
                    <c:choose>
                        <c:when test='${article.level > 1 }'>
                            <c:forEach begin="1" end="${article.level }" step="1">
                                <span style="padding-left:20px"></span>
                            </c:forEach>
                            <span style="font-size:12px;">[답변]</span>
                            <a class='cls1' href="${contextPath}/board/viewArticle.do?
                                articleNO=${article.articleNO}">${article.title}</a>
                        </c:when>
                        <c:otherwise>
                            <a class='cls1' href="${contextPath}/board/viewArticle.do?
                                articleNO=${article.articleNO}">${article.title}
                            </a>
                        </c:otherwise>
                    </c:choose>
                </td>
                <td width="10%">
                    <fmt:formatDate value="${article.writeDate}" />
                </td>
            </tr>
        </c:forEach>
    </c:when>

```

articlesList로 포워딩된 글 목록을 <forEach> 태그를 이용해 표시합니다.

(forEach) 태그의 varStatus의 count 속성을 이용해 글 번호를 1부터 자동으로 표시합니다.

왼쪽으로 30px만큼 여백을 준 후 글 제목들을 표시합니다.

부모 글 기준으로 왼쪽 여백을 level 값만큼 채워 담 글을 부모 글에 대해 들어쓰기합니다.

level 값이 1보다 큰 경우는 자식 글이므로 level 값만큼 부모 글 밑에 공백으로 들어쓰기하여 자식 글임을 표시합니다.

공백 다음에 자식 글을 표시합니다.

```

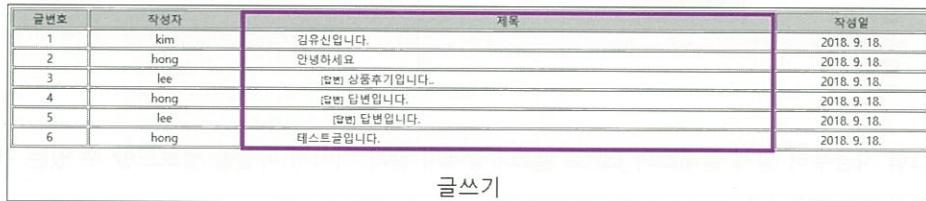
</c:choose>
</table>
<a class="cls1" href="#">  

<p class="cls2">글쓰기</p>
</a>
</body>
</html>

```

7. <http://localhost:8090/pro17/board/listArticles.do>로 요청하여 글 목록이 우리가 원하는 대로 출력되는지 확인합니다. 자식 글은 앞에 level 값만큼 들여쓰기가 되고 [답변]이라는 텍스트 다음에 표시됩니다.

▼ 그림 17-28 실행 결과



글번호	작성자	제목	작성일
1	kim	김유신입니다.	2018. 9. 18.
2	hong	안녕하세요	2018. 9. 18.
3	lee	[답변] 상품후기입니다.	2018. 9. 18.
4	hong	[답변] 답변입니다.	2018. 9. 18.
5	lee	[답변] 답변입니다.	2018. 9. 18.
6	hong	테스트글입니다.	2018. 9. 18.

글쓰기

게시판 글 목록을 계층형으로 구현하는 데 성공했습니다. 이제 세부적인 기능들을 하나씩 구현해 보겠습니다.

## 17.4.2 게시판 글쓰기 구현

게시판의 글쓰기 기능을 구현하는 과정은 다음과 같습니다.

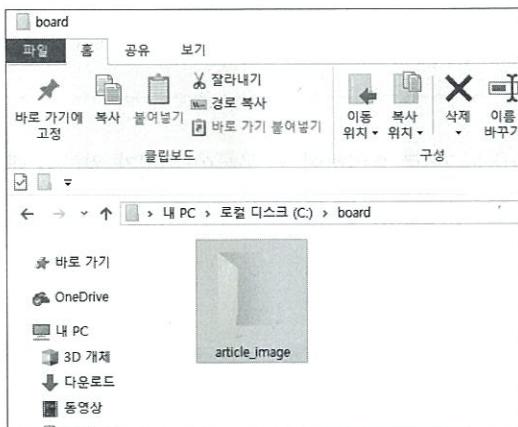
- ① 글 목록창(listArticles.jsp)에서 글쓰기창을 요청합니다.
- ② 글쓰기창에서 글을 입력하고 컨트롤러에 /board/addArticle.do로 글쓰기를 요청합니다.
- ③ 컨트롤러에서 Service 클래스로 글쓰기창에서 입력한 글 정보를 전달해 테이블에 글을 추가합니다.
- ④ 새 글을 추가하고 컨트롤러에서 다시 /board/listArticles.do로 요청하여 전체 글을 표시합니다.



글 추가는 한 개의 글을 추가하므로 /addArticle.do이고, 글 목록에 출력하는 경우는 여러 개의 글을 조회하므로 /listArticles.do이며 s가 붙습니다.

클래스와 JSP를 구현하기 전에 프로젝트의 WebContentWeb 폴더에 파일 업로드와 관련된 라이브러리를 미리 복사해 붙여 넣습니다. 그리고 파일 저장소인 C:\board\article\_image 폴더를 만듭니다.

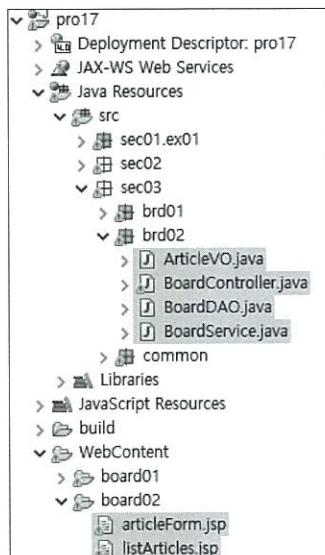
▼ 그림 17-29 파일 저장소 위치 생성



그럼 지금부터 실제 클래스와 JSP로 글쓰기창에서 글과 이미지 파일을 업로드할 수 있는 기능을 구현해 보겠습니다.

1. sec03.brd02 패키지를 만들고 클래스 파일들을 추가합니다. 그리고 board02 폴더와 관련된 JSP 파일들을 추가합니다.

▼ 그림 17-30 실습 파일 위치



2. 컨트롤러를 담당하는 BoardController 클래스를 다음과 같이 작성합니다. action 값이 /articleForm.do면 글쓰기창을 브라우저에 표시하고, action 값이 /addArticle.do면 다음 과정으로 새 글을 추가합니다. upload() 메서드를 호출해 글쓰기창에서 전송된 글 관련 정보를 Map에 key/value 쌍으로 저장합니다.

파일을 첨부한 경우 먼저 파일 이름을 Map에 저장한 후 첨부한 파일을 저장소에 업로드합니다. upload() 메서드를 호출한 후에는 반환한 Map에서 새 글 정보를 가져옵니다. 그런 다음 Service 클래스의 addArticle() 메서드 인자로 새 글 정보를 전달하면 새 글이 등록됩니다.

코드 17-19 pro17/src/sec03/brd02/BoardController.java

```
package sec03.brd02;
...
@WebServlet("/board/*")
public class BoardController extends HttpServlet
{
    private static String ARTICLE_IMAGE_REPO = "C:\\\\board\\\\article_image"; ← 글에 첨부한 이미지 저장 위치를 상수로 선언합니다.

    BoardService boardService;
    ArticleVO articleVO;

    public void init(ServletConfig config) throws ServletException
    {
        boardService = new BoardService();
        articleVO = new ArticleVO();
    }
    ...
    private void doHandle(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
        ...
        } else if (action.equals("/articleForm.do"))
        {
            nextPage = "/board02/articleForm.jsp"; ← /articleForm.do로 요청 시 글쓰기창이 나타납니다.
        } else if (action.equals("/addArticle.do"))
        {
            ...
            Map<String, String> articleMap = upload(request, response); ← 파일 업로드 기능을 사용하기 위해 upload()로 요청을 전달합니다.
            String title = articleMap.get("title");
            String content = articleMap.get("content"); ← articleMap에 저장된 글 정보를 다시 가져옵니다.
            String imageFileName = articleMap.get("imageFileName");

            articleVO.setParentNO(0); ← 새 글의 부모 글 번호를 0으로 설정합니다.
            articleVO.setId("hong"); ← 새 글 작성자 ID를 hong으로 설정합니다.
            articleVO.setTitle(title);
            articleVO.setContent(content);
        }
    }
}
```

```

        articleVO.setImageFileName(imageFileName);
        boardService.addArticle(articleVO);
        nextPage = "/board/listArticles.do";
    }

    RequestDispatcher dispatch = request.getRequestDispatcher(nextPage);
    dispatch.forward(request, response);
} catch (Exception e)
{
    e.printStackTrace();
}
}

private Map<String, String> upload(HttpServletRequest request,
                                    HttpServletResponse response)
throws ServletException, IOException
{
    Map<String, String> articleMap = new HashMap<String, String>();
    String encoding = "utf-8";
    File currentDirPath = new File(ARTICLE_IMAGE_REPO); ────────── 글 이미지 저장 폴더에 대해
    DiskFileItemFactory factory = new DiskFileItemFactory(); 파일 객체를 생성합니다.
    factory.setRepository(currentDirPath);
    factory.setSizeThreshold(1024 * 1024);
    ServletFileUpload upload = new ServletFileUpload(factory);
    try
    {
        List items = upload.parseRequest(request);
        for (int i = 0; i < items.size(); i++)
        {
            FileItem fileItem = (FileItem) items.get(i);
            if (fileItem.isFormField())
            {
                System.out.println(fileItem.getFieldName()
                    + "=" + fileItem.getString(encoding));
                articleMap.put(fileItem.getFieldName(), fileItem.getString(encoding));
            } else ────────── 파일 업로드로 같이 전송된 새 글 관련 매개변수를 Map에 (key,value)로 저장한 후 반환하고, 새 글과 관련된 title, content를 Map에 저장합니다.
            {
                System.out.println("파라미터이름:" + fileItem.getFieldName());
                System.out.println("파일이름:" + fileItem.getName());
                System.out.println("파일크기:" + fileItem.getSize() + "bytes");
            }
        }
    }
}

```

```

if (fileItem.getSize() > 0)
{
    int idx = fileItem.getName().lastIndexOf("\\");
    if (idx == -1)
    {
        idx = fileItem.getName().lastIndexOf("/");
    }

    String fileName = fileItem.getName().substring(idx + 1);
    articleMap.put(fileItem.getFieldName(), fileName);
    File uploadFile = new File(currentDirPath + "\\\" + fileName);
    fileItem.write(uploadFile); └─ 업로드된 파일의 파일 이름을 Map에
                                ("imageFileName", "업로드파일이름")로
                                저장합니다.

} // end if
} // end for
} catch (Exception e)
{
    e.printStackTrace();
}
return articleMap;
}

}

```

업로드한 파일이 존재하는 경우 업로드한 파일의  
파일 이름으로 저장소에 업로드합니다.

3. BoardDAO 클래스의 insertNewArticle() 메서드를 호출하면서 글 정보를 인자로 전달합니다.

코드 17-20 pro17/src/sec03/brd02/BoardService.java

```

package sec03.brd02;
...
public class BoardService {
    BoardDAO boardDAO;
    ...
    public void addArticle(ArticleVO article) {
        boardDAO.insertNewArticle(article);
    }
    ...
}

```

4. `insertNewArticle()` 메서드의 SQL문을 실행하기 전에 `getNewArticleNO()` 메서드를 호출해 새 글에 대한 글 번호를 먼저 가져옵니다.

코드 17-21 pro17/src/sec03/brd02/BoardDAO.java

```
package sec03.brd02;  
...  
public class BoardDAO  
{  
    private DataSource dataFactory;  
    Connection conn;  
    PreparedStatement pstmt;  
  
    ...  
    private int getNewArticleNO()  
    {  
        try  
        {  
            conn = dataFactory.getConnection(); └── 기본 글 번호 중 가장 큰 번호를 조회합니다.  
            String query = "SELECT max(articleNO) from t_board ";  
            System.out.println(query);  
            pstmt = conn.prepareStatement(query);  
            ResultSet rs = pstmt.executeQuery();  
            if (rs.next()) └── 가장 큰 번호에 1을 더한 번호를 반환합니다.  
                return (rs.getInt(1) + 1);  
            rs.close();  
            pstmt.close();  
            conn.close();  
        } catch (Exception e)  
        {  
            e.printStackTrace();  
        }  
        return 0;  
    }  
  
    public void insertNewArticle(ArticleVO article)  
    {  
        try  
        {  
            conn = dataFactory.getConnection();  
            int articleNO = getNewArticleNO(); └── 새 글을 추가하기 전에 새 글에 대한  
            int parentNO = article.getParentNO(); └── 글 번호를 가져옵니다.  
            String title = article.getTitle();  
            String content = article.getContent();  
            String id = article.getId();
```

```
String imageFileName = article.getImageFileName();
String query = "INSERT INTO t_board (articleNO, parentNO, title,
                                         content, imageFileName, id)"
               + " VALUES (?, ?, ?, ?, ?, ?);"
System.out.println(query);
stmt = conn.prepareStatement(query);
stmt.setInt(1, articleNO);
stmt.setInt(2, parentNO);
stmt.setString(3, title);
stmt.setString(4, content);
stmt.setString(5, imageFileName);
stmt.setString(6, id);
stmt.executeUpdate();
stmt.close();
conn.close();
} catch (Exception e)
{
    e.printStackTrace();
}
}
```

5. 다음은 글쓰기와 관련된 JSP 페이지를 작성할 차례입니다. listArticles.jsp를 다음과 같이 작성합니다.

코드 17-22 pro17\WebContent\board02\listArticles.jsp

```
<a class="cls1" href="${ contextPath }/board/articleForm.do">  
    <p class="cls2">글쓰기</p>  
</a>
```

6. articleForm.jsp를 다음과 같이 작성합니다. 쇼핑몰 게시판에 글을 쓸 때는 보통 사진을 첨부하는 경우가 많죠? 이처럼 글쓰기 작업을 할 때 첨부 파일도 같이 업로드할 수 있도록 반드시 <form> 태그의 enctype 속성을 multipart/form-data로 설정합니다.

코드 17-23 pro17\WebContent\board02\articleForm.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
       pageEncoding="UTF-8"
       isELIgnored="false" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
```

```

request.setCharacterEncoding("UTF-8");
%>
<c:set var="contextPath" value="${pageContext.request.contextPath}" />

<head>
    <meta charset="UTF-8">
    <title>글쓰기창</title>
    <script src="http://code.jquery.com/jquery-latest.min.js"></script>
    <script type="text/javascript">
        function readURL(input) {
            if (input.files && input.files[0]) {
                var reader = new FileReader();
                reader.onload = function (e) {
                    $('#preview').attr('src', e.target.result);
                }
                reader.readAsDataURL(input.files[0]);
            }
        }

        function backToList(obj) {
            obj.action = "${contextPath}/board/listArticles.do";
            obj.submit();
        }
    </script>
    <title>새 글 쓰기창</title>
</head>

<body>
    <h1 style = "text-align:center">새 글 쓰기</h1>
    <form name = "articleForm" method = "post" action = "${contextPath}/board/addArticle.do" enctype = "multipart/form-data">
        <table border = "0" align = "center">
            <tr>
                <td align = "right">글제목: </td>
                <td colspan = "2"><input type = "text" size = "67" maxlength="500" name="title" /></td>
            </tr>
            <tr>
                <td align = "right" valign = "top"><br>글내용: </td>
                <td colspan = 2><textarea name = "content" rows = "10" cols = "65" maxlength = "4000"></textarea> </td>
            </tr>
            <tr>
                <td align = "right">이미지파일 첨부: </td>
                <td> <input type = "file" name = "imageFileName" onchange = "readURL(this);"/></td>
            </tr>
        </table>
    </form>
</body>

```

제이쿼리 이용해 이미지 파일 첨부 시 미리 보기 기능을 구현합니다.

action 값을 /addArticle.do로 해서 '새 글 등록'을 요청합니다.

파일 업로드 기능을 위한 것입니다.

```

<td><img id = "preview" src = "#" width = 200 height = 200 /></td>
</tr>
<tr>
    <td align = "right"> </td>
    <td colspan = "2">
        <input type = "submit" value = "글쓰기" />
        <input type = button value = "목록보기" onClick = "backToList(this.form)" />
    </td>
</tr>
</table>
</form>
</body>

</html>

```

7. /board/listArticle.do로 글 목록창을 요청한 후 다시 글쓰기를 클릭해 board/articleForm.do로 글쓰기창을 요청합니다. 그리고 글쓰기창에서 “상품 후기입니다.”라는 새 글을 작성하고 글쓰기를 클릭해 /board/addArticle.do로 요청합니다.

▼ 그림 17-31 새 글을 입력한 후 글쓰기 클릭

### 새글 쓰기

글제목: 상품 후기입니다.

상품 질이 좋습니다.  
더 만들어 주세요.

글내용:



이미지파일 첨부:  duke.png

## 8. 글 목록에 새 글이 추가된 것을 볼 수 있습니다.

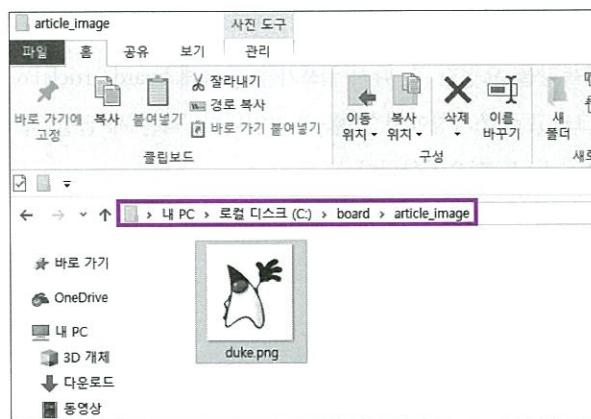
▼ 그림 17-32 새 글 추가 후 나타난 목록창

글번호	작성자	제목	작성일
1	hong	상품 후기입니다.	2018. 9. 18.
2	kim	김유신입니다.	2018. 9. 18.
3	hong	안녕하세요	2018. 9. 18.
4	lee	[답변] 상품후기입니다..	2018. 9. 18.
5	hong	[답변] 답변입니다.	2018. 9. 18.
6	lee	[답변] 답변입니다.	2018. 9. 18.
7	hong	테스트글입니다.	2018. 9. 18.

글쓰기

## 9. 글을 작성할 때 첨부한 파일은 로컬 PC의 다음 경로에 업로드됩니다.

▼ 그림 17-33 새 글 추가 시 로컬 PC에 파일 업로드



그런데 이 글쓰기 기능에는 한 가지 문제가 있습니다. 발견하였나요?

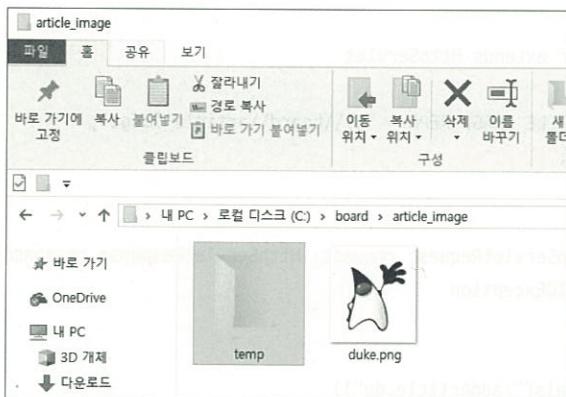
바로 새 글에 첨부한 파일들이 같은 폴더에 저장된다는 것입니다. 얼핏 보기엔 큰 문제가 아닌 것 같지만 이렇게 저장하면 다른 사용자가 첨부한 파일과 이름이 같아 구별하기가 어렵습니다.

따라서 이번에는 업로드한 파일이 각각의 글 번호를 이름으로 하는 폴더를 생성하고, 저장까지 할 수 있도록 구현해 보겠습니다. 과정은 다음과 같습니다.

- ① 글쓰기창에서 새 글 전송 시 컨트롤러의 `upload()` 메서드를 호출해 새 글 정보를 Map으로 반환 받고 첨부한 파일은 임시로 temp 폴더에 업로드합니다.
- ② 컨트롤러는 Service 클래스의 `addNewArticle()` 메서드를 호출하면서 새 글 정보를 인자로 전달해 테이블에 추가한 후 새 글 번호를 반환 받습니다.
- ③ 컨트롤러에서 반환 받은 새 글 번호를 이용해 파일 저장소에 새 글 번호로 폴더를 생성하고 temp 폴더의 파일을 새 글 번호 폴더로 이동합니다.

1. 로컬 PC의 파일 저장소에 temp 폴더를 생성합니다.

▼ 그림 17-34 파일 저장소에 temp 폴더 생성



2. sec03, brd03 패키지를 만들고 다음과 같이 클래스 파일을 추가합니다.

▼ 그림 17-35 실습 파일 위치



3. BoardController 클래스를 다음과 같이 작성합니다. upload() 메서드를 호출해첨부한 파일을 temp 폴더에 업로드한 후 새 글 정보를 Map으로 가져옵니다. 그리고 새 글을 테이블에 추가한 후 반환 받은 새 글 번호로 폴더를 생성하고 temp 폴더의 이미지를 새 글 폴더로 이동합니다.

코드 17-24 pro17/src/sec03/brd03/BoardController.java

```
package sec03.brd03;  
...  
@WebServlet("/board/*")  
public class BoardController extends HttpServlet  
{  
    private static String ARTICLE_IMAGE_REPO = "C:\\board\\article_image";  
    BoardService boardService;  
    ArticleVO articleVO;  
    ...  
    private void doHandle(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException  
    {  
        ...  
        } else if (action.equals("/addArticle.do"))  
        {  
            int articleNO = 0;  
            Map<String, String> articleMap = upload(request, response);  
            String title = articleMap.get("title");  
            String content = articleMap.get("content");  
            String imageFileName = articleMap.get("imageFileName");  
            articleVO.setParentNO(0);  
            articleVO.setId("hong");  
            articleVO.setTitle(title);  
            articleVO.setContent(content);  
            articleVO.setImageFileName(imageFileName);  
            articleNO = boardService.addArticle(articleVO);  
            ..... 파일에 새 글을 추가한 후 새 글에  
            ..... 대한 글 번호를 가져옵니다.  
            ..... 파일을 첨부한 경우에만 수행합니다.  
            if (imageFileName != null && imageFileName.length() != 0)  
            {  
                File srcFile = new File(ARTICLE_IMAGE_REPO + "\\temp" + "\\  
                    + imageFileName);  
                File destDir = new File(ARTICLE_IMAGE_REPO + "\\\" + articleNO);  
                destDir.mkdirs();  
                FileUtils.moveFileToDirectory(srcFile, destDir, true);  
            }  
            ..... temp 폴더의 파일을 글 번호로 풀더로 이동시킵니다.  
            PrintWriter pw = response.getWriter();  
            pw.print("<script>" + " alert('새글을 추가했습니다.');" +  
                + " location.href='"  
                + request.getContextPath()  
                + "/board/listArticles.do';" + "</script>");  
            ..... 새 글 등록 메시지를 나타낸 후 자바스크립트  
            ..... location 객체의 href 속성을 이용해 글 목록을  
            ..... 요청합니다.  
        }  
    }  
}
```

```

        RequestDispatcher dispatch = request.getRequestDispatcher(nextPage);
        dispatch.forward(request, response);
    } catch (Exception e)
    {
        e.printStackTrace();
    }
}

private Map<String, String> upload(HttpServletRequest request,
                                    HttpServletResponse response)
throws ServletException, IOException
{
    ...
    String fileName = fileItem.getName().substring(idx + 1);
    File uploadFile = new File(currentDirPath + "\\temp\\" + fileName);
    fileItem.write(uploadFile); └── 첨부한 파일을 먼저 temp 폴더에 업로드합니다.
} // end if
} // end if
} // end for
} catch (Exception e)
{
    e.printStackTrace();
}
return articleMap;
}
}

```

4. BoardService 클래스를 다음과 같이 작성합니다. addArticle() 메서드의 반환 타입을 int로 변경합니다. 그리고 BoardDAO의 insertNewArticle() 메서드를 호출해 새 글 번호를 받아서 반환합니다.

코드 17-25 pro17/src/sec03/brd03/BoardService.java

```

package sec03.brd03;
...
public class BoardService {
    ...
    public int addArticle(ArticleVO article) { └── 새 글 번호를 컨트롤러로 반환합니다.
        return boardDAO.insertNewArticle(article);
    }
    ...
}

```

5. BoardDAO 클래스에서는 insertNewArticle() 메서드를 호출해 SQL문을 실행한 후 새 글 번호를 반환합니다.

코드 17-26 pro17/src/sec03/brd03/BoardDAO.java

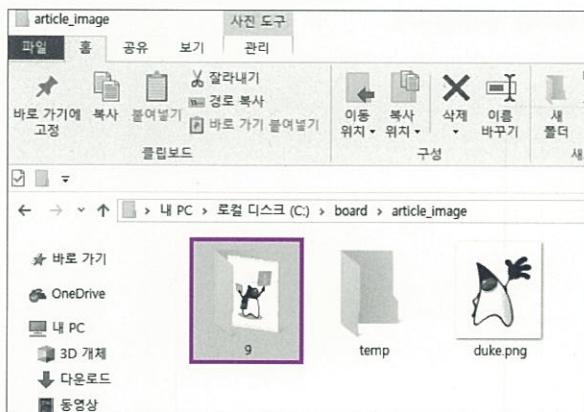
```
package sec03.brd03;  
...  
public int insertNewArticle(ArticleVO article)  
{  
    int articleNO = getNewArticleNO();  
    try  
    {  
        conn = dataFactory.getConnection();  
        int parentNO = article.getParentNO();  
        String title = article.getTitle();  
        String content = article.getContent();  
        String id = article.getId();  
        String imageFileName = article.getImageFileName();  
        String query = "INSERT INTO t_board (articleNO, parentNO, title, content,  
                                         imageFileName, id)"  
                     + " VALUES (?, ?, ?, ?, ?, ?)";  
  
        System.out.println(query);  
        pstmt = conn.prepareStatement(query);  
        pstmt.setInt(1, articleNO);  
        pstmt.setInt(2, parentNO);  
        pstmt.setString(3, title);  
        pstmt.setString(4, content);  
        pstmt.setString(5, imageFileName);  
        pstmt.setString(6, id);  
        pstmt.executeUpdate();  
        pstmt.close();  
        conn.close();  
    } catch (Exception e)  
    {  
        e.printStackTrace();  
    }  
  
    return articleNO; }  
}
```

insert문을 이용해 글 정보를 추가합니다.

SQL문으로 새 글을 추가하고  
새 글 번호를 반환합니다.

6. 그림 17-36은 새 글 쓰기를 실행한 후 생성된 파일 저장소를 나타낸 것입니다.

▼ 그림 17-36 글 번호로 파일 저장 폴더 생성



임시로 저장하는 temp 폴더 외에도 각각의 글 번호에 대한 폴더가 만들어진 것을 볼 수 있습니다.  
이러면 보다 관리하기가 편하겠죠?

### 17.4.3 글 상세 기능 구현

글 목록에서 글 제목을 클릭했을 때 글의 상세 내용을 보여주는 기능을 구현해 보겠습니다.

다음은 글 상세 기능을 구현하는 과정입니다.

- ① 글 목록창에서 글 제목을 클릭해 컨트롤러에 /board/viewArticle.do?articleNO=글번호로 요청합니다.
- ② 컨트롤러는 전송된 글 번호로 글 정보를 조회하여 글 상세창(viewArticle.jsp)으로 포워딩합니다.
- ③ 글 상세창(viewArticle.jsp)에 글 정보와 이미지 파일이 표시됩니다.

1. 글 상세 기능에 관련된 자바 코드와 JSP 파일을 다음과 같이 추가합니다. 글 상세 기능을 구현하는 데 필요한 첨부 이미지를 표시하기 위해 sec03.common 패키지를 만든 후 FileDownloadController 클래스를 생성합니다.

▼ 그림 17-37 실습 파일 위치



2. FileDownloadController 클래스를 다음과 같이 작성합니다. viewArticle.jsp에서 전송한 글 번호와 이미지 파일 이름으로 파일 경로를 만든 후 해당 파일을 내려 받습니다.

코드 17-27 pro17/src/sec03/common/FileDownloadController.java

```
package sec03.common;  
...  
@WebServlet("/download.do")  
...  
private void doHandle(HttpServletRequest request, HttpServletResponse response)  
throws ServletException, IOException  
{  
    request.setCharacterEncoding("utf-8");  
    response.setContentType("text/html; charset=utf-8");  
    String imageFileName = (String) request.getParameter("imageFileName");  
    String articleNO = request.getParameter("articleNO");  
    System.out.println("imageFileName=" + imageFileName);  
    OutputStream out = response.getOutputStream();  
}
```

이미지 파일 이름과 글 번호를 가져옵니다.

```

String path = ARTICLE_IMAGE_REPO + "\\\" + articleNO + "\\\" + imageName;
File imageFile = new File(path); └─ 글 번호에 대한 파일 경로를 설정합니다.

response.setHeader("Cache-Control", "no-cache");
response.addHeader("Content-disposition", "attachment;fileName=" + imageName);
FileInputStream in = new FileInputStream(imageFile);
byte[] buffer = new byte[1024 * 8]; └─ 이미지 파일을 내려 받는 데 필요한
while (true) response에 헤더 정보를 설정합니다.
{
    int count = in.read(buffer); └─ 버퍼를 이용해 한 번에 8Kb씩 전송합니다.
    if (count == -1)
        break;
    out.write(buffer, 0, count);
}
in.close();
out.close();
}

}

```

3. BoardController 클래스를 다음과 같이 작성합니다. /viewArticle.do로 요청하여 글 번호를 받아옵니다. 그리고 그 번호에 해당하는 글 정보를 가져와 article 속성으로 바인딩한 후 viewArticle.jsp로 포워딩합니다.

코드 17-28 pro17/src/sec03/brd04/BoardController.java

```

package sec03.brd04;
...
@WebServlet("/board/*") └─ 실습 파일의 주석을 해제하고 사용하세요.
...
private void doHandle(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException{
    String nextPage = "";
    request.setCharacterEncoding("utf-8");
    response.setContentType("text/html; charset=utf-8");
    ...
else if(action.equals("/viewArticle.do")) { └─ 글 상세창을 요청할 경우 articleNO
    String articleNO = request.getParameter("articleNO"); 값을 가져옵니다.
    articleVO=boardService.viewArticle(Integer.parseInt(articleNO));
    request.setAttribute("article",articleVO);
    nextPage = "/board03/viewArticle.jsp"; └─ articleNO에 대한 글 정보를 조회하
}                                고 article 속성으로 바인딩합니다.
RequestDispatcher dis= request.getRequestDispatcher(nextPage);
dis.forward(request, response
...

```

4. 컨트롤러에서 전달받은 글 번호로 다시 selectArticle() 메서드를 호출합니다.

코드 17-29 pro17/src/sec03/brd04/BoardService.java

```
...
public ArticleVO viewArticle(int articleNO)
{
    ArticleVO article = null;
    article = boardDAO.selectArticle(articleNO);
    return article;
}
```

5. 전달받은 글 번호를 이용해 글 정보를 조회합니다.

코드 17-30 pro17/src/sec03/brd04/BoardDAO.java

```
...
public ArticleVO selectArticle(int articleNO)
{
    ArticleVO article = new ArticleVO();
    try
    {
        conn = dataFactory.getConnection(); └───────── 전달받은 글 번호를 이용해 글 정보를 조회합니다.
        String query = "select articleNO,parentNO,title,content, imageFileName,id,writeDate"
                      + " from t_board"
                      + " where articleNO=?";
        System.out.println(query);
        pstmt = conn.prepareStatement(query);
        pstmt.setInt(1, articleNO);
        ResultSet rs = pstmt.executeQuery();
        rs.next();
        int _articleNO = rs.getInt("articleNO");
        int parentNO = rs.getInt("parentNO");
        String title = rs.getString("title");
        String content = rs.getString("content");
        String imageFileName = rs.getString("imageFileName");
        String id = rs.getString("id");
        Date writeDate = rs.getDate("writeDate");

        article.setArticleNO(_articleNO);
        article.setParentNO(parentNO);
        article.setTitle(title);
        article.setContent(content);
        article.setImageFileName(imageFileName);
        article.setId(id);
        article.setWriteDate(writeDate);
    }
}
```

```

        rs.close();
        pstmt.close();
        conn.close();
    } catch (Exception e)
    {
        e.printStackTrace();
    }
    return article;
}

```

6. viewArticle.jsp를 다음과 같이 작성합니다. 컨트롤러에서 바인딩한 글 정보 속성을 이용해 표시합니다. 이미지 파일이 존재하는 경우는 글 번호와 이미지 파일 이름을 FileDownloadController로 전송한 후 <img> 태그에 다운로드하여 표시합니다.

코드 17-31 pro17/WebContent/board03/viewArticle.jsp

```

<td width="20%" align="center" bgcolor="#FF9933">글제목</td>
<td>
    <input type="text" value="${article.title}" name="title" id="i_title" disabled
/>
</td>
</tr>
<tr>
    <td width="20%" align="center" bgcolor="#FF9933">글내용</td>
    <td>
        <textarea rows="20" cols="60" name="content"
            id="i_content" disabled />${article.content}</textarea>
    </td>
    </tr>
    <c:if test="${not empty article.imageFileName && article.imageFileName!='null' }">
        <tr>
            <td width="20%" align="center" bgcolor="#FF9933" rowspan="2">이미지</td>
            <td>
                <input type="hidden" name="originalFileName" value="${article.imageFileName}" />
                <br>
            </td>
        </tr>
        <tr>
            <td>

```

imageFileName 값이 있으면 이미지를 표시합니다.

<hidden> 태그에 원래 이미지 파일 이름을 저장합니다.

FileDownloadController 서블릿에 이미지 파일 이름과  
글 번호를 전송해 이미지를 <img> 태그에 표시합니다.

7. /board/listArticles.do로 글 목록창을 요청하여 글 제목(상품 후기입니다.)을 클릭하세요.

그러면 /board/viewArticle.do?articleNO=7로 요청합니다.

▼ 그림 17-38 글 번호로 글 상세 화면 요청

글번호	작성자	제목	작성일
1	hong	상품 후기입니다.	2018. 9. 18.
2	kim	김유신입니다.	2018. 9. 18.
3	hong	안녕하세요	2018. 9. 18.
4	lee	[답변] 상품후기입니다..	2018. 9. 18.
5	hong	[답변] 답변입니다.	2018. 9. 18.
6	lee	[답변] 답변입니다.	2018. 9. 18.
7	hong	테스트글입니다.	2018. 9. 18.

글쓰기

8. 전달받은 글 번호로 글 정보를 조회한 후 글 상세 화면에 내용을 표시합니다.

▼ 그림 17-39 글 상세 화면에 이미지 표시

글번호 작성자 아이디 제목	7 hong 상품 후기입니다.
내용	상품 질이 좋습니다. 더 만들어 주세요 .
이미지	
파일 선택 등록일자	선택된 파일 없음 2018. 9. 18.
<input type="button" value="수정하기"/> <input type="button" value="삭제하기"/> <input type="button" value="리스트로 돌아가기"/> <input type="button" value="답글쓰기"/>	

## 17.4.4 글 수정 기능 구현

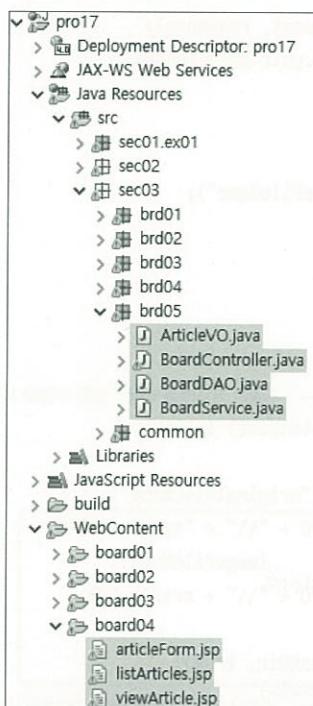
이번에는 기존에 작성한 글을 수정하는 기능을 구현해 보겠습니다.

글 수정 기능을 구현하는 과정은 다음과 같습니다.

- ❶ 글 상세창(viewArticle.jsp)에서 수정하기를 클릭해 글 정보를 표시하는 입력창들을 활성화합니다.
- ❷ 글 정보와 이미지를 수정한 후 수정반영하기를 클릭해 컨트롤러에 /board/modArticle.do로 요청합니다.
- ❸ 컨트롤러는 요청에 대해 upload() 메서드를 이용하여 수정된 데이터를 Map에 저장하고 반환합니다.
- ❹ 컨트롤러는 수정된 데이터를 테이블에 반영한 후 temp 폴더에 업로드된 수정 이미지를 글 번호 폴더로 이동합니다.
- ❺ 마지막으로 글 번호 폴더에 있던 원래 이미지 파일을 삭제합니다.

1. sec03.brd05 패키지를 만들고 글 수정 기능과 관련된 클래스를 다음과 같이 추가합니다.

▼ 그림 17-40 실습 파일 위치



2. BoardController 클래스를 다음과 같이 작성합니다. 컨트롤러에서 수정을 요청하면 upload() 메서드를 이용해 수정 데이터를 Map으로 가져옵니다. Map의 데이터를 다시 ArticleVO 객체의 속성에 저장한 후 SQL문으로 전달하여 수정 데이터를 반영합니다. 마지막으로 temp 폴더에 업로드된 수정 이미지를 다시 글 번호 폴더로 이동하고 글 번호 폴더의 원래 이미지를 삭제합니다.

코드 17-32 pro17/src/sec03/brd05/BoardController.java

```
package sec03.brd05;  
...  
@WebServlet("/board/*") ─────────── 실습 파일의 주석을 해제합니다.  
public class BoardController extends HttpServlet  
{  
    private static String ARTICLE_IMAGE_REPO = "C:\\\\board\\\\article_image";  
    BoardService boardService;  
    ArticleVO articleVO;  
    ...  
    private void doHandle(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException{  
        ...  
        } else if (action.equals("/modArticle.do"))  
        {  
            Map<String, String> articleMap = upload(request, response);  
            int articleNO = Integer.parseInt(articleMap.get("articleNO"));  
            articleVO.setArticleNO(articleNO);  
            String title = articleMap.get("title");  
            String content = articleMap.get("content");  
            String imageFileName = articleMap.get("imageFileName");  
            articleVO.setParentNO(0);  
            articleVO.setId("hong");  
            articleVO.setTitle(title);  
            articleVO.setContent(content);  
            articleVO.setImageFileName(imageFileName);  
            boardService.modArticle(articleVO); ─────────── 전송된 글 정보를 이용해 글을 수정합니다.  
            if (imageFileName != null && imageFileName.length() != 0)  
            {  
                String originalFileName = articleMap.get("originalFileName");  
                File srcFile = new File(ARTICLE_IMAGE_REPO + "\\\" + "temp" + "\\\" +  
                    imageFileName);  
                File destDir = new File(ARTICLE_IMAGE_REPO + "\\\" + articleNO);  
                destDir.mkdirs();  
                FileUtils.moveFileToDirectory(srcFile, destDir, true);  
            }  
        }  
    } ─────────── 수정된 이미지 파일을 폴더로 이동합니다.
```

```

        File oldFile = new File(ARTICLE_IMAGE_REPO + "\\\" + articleNO + "\\"
                                + originalFileName);
        oldFile.delete();
    }
    PrintWriter pw = response.getWriter();
    pw.print("<script>" + " alert('글을 수정했습니다.');" + " location.href='"
            + request.getContextPath()
            + "/board/viewArticle.do?articleNo="
            + articleNO + ";" + "</script>");
    return;
}

```

↑ 전송된 originalFileName을 이용해  
기존의 파일을 삭제합니다.

↑ 글 수정 후 location 객체의 href 속성을  
이용해 글 상세 화면을 나타냅니다.

---

```

RequestDispatcher dispatch = request.getRequestDispatcher(nextPage);
dispatch.forward(request, response);
} catch (Exception e)
{
    e.printStackTrace();
}
}

```

3. BoardService 클래스를 다음과 같이 작성합니다. 컨트롤러에서 modArticle() 메서드를 호출하면 다시 BoardDAO의 updateArticle() 메서드를 호출하면서 수정 데이터를 전달합니다.

코드 17-33 pro17/src/sec03/brd05/BoardService.java

```

...
public void modArticle(ArticleVO article) {
    boardDAO.updateArticle(article);
}

```

4. BoardDAO 클래스를 다음과 같이 작성합니다. 전달된 수정 데이터에 대해 이미지 파일을 수정하는 경우와 이미지 파일을 수정하지 않는 경우를 구분해 동적으로 SQL문을 생성하여 수정 데이터를 반영합니다.

코드 17-34 pro17/src/sec03/brd05/BoardDAO.java

```

...
public void updateArticle(ArticleVO article)
{
    int articleNO = article.getArticleNO();
    String title = article.getTitle();
    String content = article.getContent();
    String imageFileName = article.getImageFileName();
    try
    {
        ...
    }
}

```

```

{
    conn = dataFactory.getConnection();
    String query = "update t_board set title=?,content=?";
    if (imageFileName != null && imageFileName.length() != 0)
    {
        query += ",imageFileName=?";
    } else {
        query += " where articleNO = ?"; ← 수정된 이미지 파일이 있을 때만
    }
    System.out.println(query);
    pstmt = conn.prepareStatement(query);
    pstmt.setString(1, title);
    pstmt.setString(2, content);
    if (imageFileName != null && imageFileName.length() != 0)
    {
        pstmt.setString(3, imageFileName);
        pstmt.setInt(4, articleNO);
    } else
    {
        pstmt.setInt(3, articleNO);
    }
    pstmt.executeUpdate(); ← 이미지 파일을 수정하는 경우와 그렇지
    pstmt.close(); 않은 경우를 구분해서 설정합니다.
    conn.close();
} catch (Exception e)
{
    e.printStackTrace();
}
}

```

5. viewArticle.jsp를 다음과 같이 작성합니다. 수정하기를 클릭해 fn\_enable() 함수를 호출하여 비활성화된 텍스트 박스를 수정할 수 있도록 활성화시킵니다. 또한 글 정보와 이미지를 수정한 후 수정반영하기를 클릭하면 fn\_modify\_article() 함수를 호출하여 컨트롤러로 수정 데이터를 전송합니다.

#### 코드 17-35 pro17/WebContent/board04/viewArticle.jsp

```

<script type="text/javascript">
...
function fn_enable(obj) ← 텍스트 박스의 id로 접근해 disabled 속성을
{
    document.getElementById("i_title").disabled = false; ← false로 설정합니다.
    document.getElementById("i_content").disabled = false;
    document.getElementById("i_imageFileName").disabled = false;
    document.getElementById("tr_btn_modify").style.display = "block";
}

```

```

        document.getElementById("tr_btn").style.display = "none";
    }
}

function fn_modify_article(obj) {
    obj.action = "${contextPath}/board/modArticle.do";
    obj.submit();
}

</script>
</head>

<body>
<form name="frmArticle" method="post" action="${contextPath}"
      enctype="multipart/form-data">
<table border="0" align="center">
    <tr>
        <td width="150" align="center" bgcolor="#FF9933">
            글번호
        </td>
        <td>
            <input type="text" value="${article.articleNO }" disabled />
            <input type="hidden" name="articleNO" value="${article.articleNO}" />
        </td>
    </tr>
    <tr>
        <td width="150" align="center" bgcolor="#FF9933">
            작성자 아이디
        </td>
        <td>
            <input type=text value="${article.id }" name="writer" disabled />
        </td>
    </tr>
    <tr>
        <td width="150" align="center" bgcolor="#FF9933">
            제목
        </td>
        <td>
            <input type=text value="${article.title }" name="title" id="i_title"
                   disabled />
        </td>
    </tr>
    <tr>
        <td width="150" align="center" bgcolor="#FF9933">
            내용
        </td>
    </tr>
</table>
</form>

```

수정반영하기 클릭 시 컨트롤러에  
수정 데이터를 전송합니다.

글 수정 시 글 번호를 컨트롤러로 전송하기 위해 미리  
<hidden> 태그를 이용해 글 번호를 저장합니다.

```

<td>
    <textarea rows="20" cols="60" name="content" id="i_content" disabled />
    ${article.content }</textarea>
</td>
</tr>

<c:if test="${not empty article.imageFileName && article.imageFileName!='null' }">
    <tr>
        <td width="150" align="center" bgcolor="#FF9933" rowspan="2">
            이미지
        </td>
        <td>
            <input type="hidden" name="originalFileName"
                value="${article.imageFileName }" />
            <br>
        </td>
    </tr>
    <tr>
        <td>
            <input type="file" name="imageFileName " id="i_imageFileName"
                disabled onchange="readURL(this);"/>
        </td>
    </tr>
</c:if>
<tr>
    <td width=20% align=center bgcolor="#FF9933>
        등록일자
    </td>
    <td>
        <input type=text value=<fmt:formatDate value=" ${article.writeDate}" />
            disabled />
    </td>
</tr>
<tr id="tr_btn_modify">
    <td colspan="2" align="center">
        <input type=button value="수정반영하기" onClick="fn_modify_article(frmArticle)">
        <input type=button value="취소" onClick="backToList(frmArticle)">
    </td>
</tr>
...
<tr id="tr_btn">
    <td colspan=2 align=center>
        <input type=button value="수정하기" onClick="fn_enable(this.form)">

```

```

<input type=button value="삭제하기"
       onClick="fn_remove_article('${contextPath}/board/removeArticle.do'
                           ,${article.articleNO})">
<input type=button value="리스트로 돌아가기"
       onClick="backToList(this.form)">
<input type=button value="답글쓰기"
       onClick="fn_reply_form('${contextPath}/board/replyForm.do'
                           ,${article.articleNO})">
</td>
</tr>
...

```

6. 글 상세창에서 수정하기를 클릭해 글 정보가 표시된 텍스트 박스를 활성화시킵니다.

▼ 그림 17-41 수정하기 클릭

글번호 작성자 아이디 제목	7 hong 상품 후기입니다.
내용	상품 질이 좋습니다. 더 만들어 주세요.
이미지	
등록일자	파일선택 선택된 파일 없음 2018. 9. 18.
<input type="button" value="수정하기"/> <input type="button" value="삭제하기"/> <input type="button" value="리스트로 돌아가기"/> <input type="button" value="답글쓰기"/>	

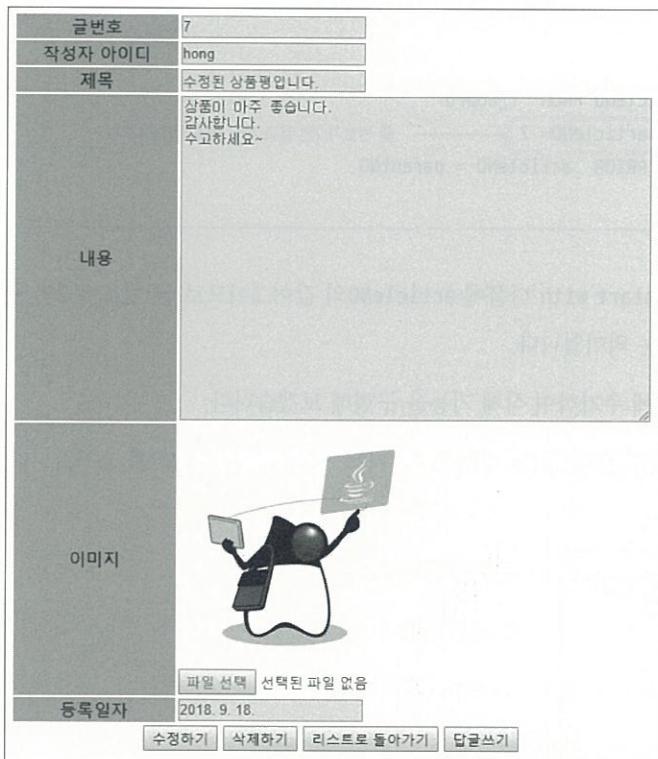
7. 글 정보와 이미지 파일을 수정한 후 수정반영하기를 클릭해 /board/modArticle.do로 요청합니다.

▼ 그림 17-42 글 정보와 이미지 수정 후 수정반영하기 클릭

글번호	7
작성자 아이디	hong
제목	수정된 상품평입니다.
내용	상품이 아주 좋습니다. 감사합니다. 수고하세요~
이미지	
파일 선택	duke2.jpg
등록일자	2018. 9. 18.
<b>수정반영하기</b>	<b>취소</b>

8. 글 수정 후 수정된 내용으로 글 상세 화면에 표시합니다.

▼ 그림 17-43 수정 반영 후 글 상세 화면에 표시



#### 17.4.5 글 삭제 기능 구현

이제 게시판의 글을 삭제하는 과정을 구현할 차례입니다. 글을 삭제할 때는 테이블의 글뿐만 아니라 그 글의 자식 글과 이미지 파일도 함께 삭제해야 합니다.

글 삭제 과정은 다음과 같습니다.

- ① 글 상세창(viewArticle.jsp)에서 삭제하기를 클릭하면 /board/removeArticle.do로 요청합니다.
- ② 컨트롤러에서는 글 상세창에서 전달받은 글 번호에 대한 글과 이에 관련된 자식 글들을 삭제합니다.
- ③ 삭제된 글에 대한 이미지 파일 저장 폴더도 삭제합니다.

코드 17-36은 오라클의 계층형 SQL문을 이용해 부모 글에 대한 자식 글을 삭제하는 SQL문입니다.

#### 코드 17-36 자식 글이 있는 부모 글 삭제 SQL문

```
DELETE FROM t_board
WHERE articleNO in (
    SELECT articleNO FROM t_board
    START WITH articleNO= 2 ────────── 글 번호가 2인 글과 자식 글을 삭제합니다.
    CONNECT BY PRIOR articleNO = parentNO
);
```

다시 말해 delete문에서는 start with 다음에 articleNO의 값이 2이므로 글 번호가 2인 글과 그 자식 글들을 모두 삭제하라는 의미입니다.

그럼 이를 BoardDAO 클래스에 추가하여 삭제 기능을 구현해 보겠습니다.

- sec04.brd06 패키지를 만들고 다음과 같이 삭제 기능 자바 클래스와 JSP를 추가합니다.

#### ▼ 그림 17-44 실습 파일 위치



2. BoardController 클래스를 다음과 같이 작성합니다. 브라우저에서 삭제를 요청하면 글 번호를 메서드로 전달해 글 번호에 대한 글과 그 자식 글을 삭제하기 전에 먼저 삭제할 글 번호와 자식 글 번호를 목록으로 가져옵니다. 그리고 글을 삭제한 후 글 번호로 이루어진 이미지 저장 폴더까지 모두 삭제합니다.

코드 17-37 pro17/src/sec03/brd06/BoardController.java

```

package sec03.brd06;
...
@WebServlet("/board/*")
private void doHandle(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException{
    ...
} else if (action.equals("/removeArticle.do")){
{
    int articleNO = Integer.parseInt(request.getParameter("articleNO"));
    List<Integer> articleNOList = boardService.removeArticle(articleNO);
    for (int _articleNO : articleNOList)
    {
        File imgDir = new File(ARTICLE_IMAGE_REPO + "\\\" + _articleNO);
        if (imgDir.exists())
        {
            FileUtils.deleteDirectory(imgDir);
        }
    }
}
PrintWriter pw = response.getWriter();
pw.print("<script>" + " alert('글을 삭제했습니다.');" + " location.href='"
+ request.getContextPath() + "/board/listArticles.do';"
+ "</script>");
return;
}

RequestDispatcher dispatch = request.getRequestDispatcher(nextPage);
dispatch.forward(request, response);
...

```

articleNO 값에 대한 글을 삭제한 후 삭제된 부모 글과  
자식 글의 articleNO 목록을 가져옵니다.

삭제된 글들의 이미지 저장 폴더들을 삭제합니다.

3. BoardService 클래스를 다음과 같이 작성합니다. 컨트롤러에서 removeArticle() 메서드 호출 시 매개변수 articleNO로 글 번호를 전달받아 BoardDAO의 selectRemovedArticles()를 먼저 호출해 글 번호에 대한 글과 그 자식 글의 글 번호를 articleNOList에 저장합니다. 그런 다음 deleteArticle () 메서드를 호출해 글 번호에 대한 글과 자식 글을 삭제하고 글 번호를 반환합니다.

**코드 17-38** pro17/src/sec03/brd06/BoardService.java

```

...
public List<Integer> removeArticle(int articleNO) {
    List<Integer> articleNOList = boardDAO.selectRemovedArticles(articleNO);
    boardDAO.deleteArticle(articleNO); └─ 글을 삭제하기 전 글 번호들을 ArrayList 객체에 저장합니다.
    return articleNOList; ─────────── 삭제한 글 번호 목록을 컨트롤러로 반환합니다.

```

4. BoardDAO 클래스를 다음과 같이 작성합니다. selectRemovedArticles() 메서드는 삭제할 글에 대한 글 번호를 가져옵니다. deleteArticle() 메서드는 전달된 articleNO에 대한 글을 삭제합니다.

**코드 17-39** pro17/src/sec03/brd06/BoardDAO.java

```

...
public void deleteArticle(int articleNO)
{
    try
    {
        conn = dataFactory.getConnection();
        String query = "DELETE FROM t_board ";
        query += " WHERE articleNO in (";
        query += "  SELECT articleNO FROM t_board ";
        query += "  START WITH articleNO = ?";
        query += "  CONNECT BY PRIOR articleNO = parentNO )";
        System.out.println(query);
        pstmt = conn.prepareStatement(query);
        pstmt.setInt(1, articleNO);
        pstmt.executeUpdate();
        pstmt.close();
        conn.close();
    } catch (Exception e)
    {
        e.printStackTrace();
    }
}

public List<Integer> selectRemovedArticles(int articleNO)
{
    List<Integer> articleNOList = new ArrayList<Integer>();
    try
    {
        conn = dataFactory.getConnection();
        String query = "SELECT articleNO FROM t_board ";
        query += " START WITH articleNO = ?";
        query += " CONNECT BY PRIOR articleNO = parentNO";

```

오라클의 계층형 SQL문을 이용해 삭제 글과 관련된 자식 글까지 모두 삭제합니다.

삭제한 글들의 articleNO를 조회합니다.

```

System.out.println(query);
pst = conn.prepareStatement(query);
pst.setInt(1, articleNO);
ResultSet rs = pst.executeQuery();
while (rs.next())
{
    articleNO = rs.getInt("articleNO");
    articleNOList.add(articleNO);
}
pst.close();
conn.close();
} catch (Exception e)
{
    e.printStackTrace();
}
return articleNOList;
}

```

5. viewArticle.jsp에서 삭제하기를 클릭하면 fn\_remove\_article() 자바스크립트 함수를 호출해 글 번호인 articleNO를 컨트롤러로 전송하도록 구현합니다.

코드 17-40 pro17/WebContent/board05/viewArticle.jsp

```

...
function fn_remove_article(url,articleNO) {
    var form = document.createElement("form");           ← 자바스크립트를 이용해 동적으로
    form.setAttribute("method", "post");                 ← <form> 태그를 생성합니다.
    form.setAttribute("action", url);
    var articleNOInput = document.createElement("input"); ← 자바스크립트를 이용해 동적으로
    articleNOInput.setAttribute("type","hidden");          ← <input> 태그를 생성한 후 name과
    articleNOInput.setAttribute("name","articleNO");       ← value를 articleNO와 컨트롤러로
    articleNOInput.setAttribute("value", articleNO);      ← 글 번호로 설정합니다.
    form.appendChild(articleNOInput);                    ← 동적으로 생성된 <input> 태그를 동적으로
    document.body.appendChild(form);                     ← 생성한 <form> 태그에 append합니다.
    form.submit();                                     ← <form> 태그를 <body> 태그에 추가
...                                                 ← (append)한 후 서버에 요청합니다.
</script>
...
<tr >
<td colspan=2 align=center>
    <input type=button value="수정하기" onClick="fn_enable(this.form)">
    <input type=button value="삭제하기" onClick="fn_remove_article('${contextPath}/board/removeArticle.do',
        ${article.articleNO})">

```

삭제하기 클릭 시 fn\_remove\_article() 자바스크립트 함수를 호출하면서 articleNO를 전달합니다.

```

<input type=button value="리스트로 돌아가기" onClick="backToList(this.form)">
</td>
</tr>

```

6. 글 상세창에서 삭제하기를 클릭해 /board/removeArticle.do?articleNO=8로 요청합니다.

▼ 그림 17-45 삭제하기 클릭

글번호	8
작성자 아이디	hong
제목	수정된 상품평입니다.
내용	상품이 아주 좋습니다. 감사합니다. 수고하세요~
이미지	
파일 선택	선택된 파일 없음
등록일자	2018. 9. 17.
<input type="button" value="수정하기"/> <input type="button" value="삭제하기"/> <input type="button" value="리스트로 돌아가기"/> <input type="button" value="답글쓰기"/>	

7. 글을 삭제한 후 다시 글 목록을 표시합니다.

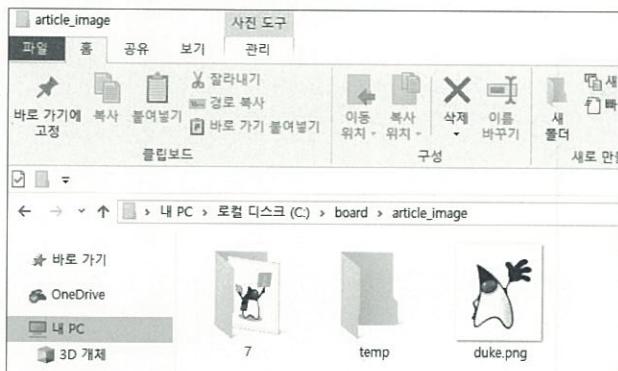
▼ 그림 17-46 삭제 후 다시 글 목록 표시

글번호	작성자	제목	작성일
1	kim	김유신입니다.	2018. 9. 18.
2	hong	안녕하세요	2018. 9. 18.
3	lee	[답변] 상품후기입니다..	2018. 9. 18.
4	hong	[답변] 답변입니다.	2018. 9. 18.
5	lee	[답변] 답변입니다.	2018. 9. 18.
6	hong	테스트글입니다.	2018. 9. 18.

글쓰기

8. 로컬 PC를 확인해 보면 8번 글에 해당하는 저장소 폴더가 삭제된 것을 확인할 수 있습니다.

▼ 그림 17-47 삭제된 글에 해당하는 이미지 저장 폴더 삭제



## 17.4.6 답글 쓰기 기능 구현

지금까지 게시판에서 글 목록을 보고, 새 글을 쓰고, 글 상세를 보고, 수정 및 삭제하는 기능까지 구현해 보았습니다. 엘추 게시판의 기능을 모두 갖춘 것 같습니다. 하지만 아직 빠진 것이 있습니다. 보통 쇼핑몰 게시판을 보면 댓글, 즉 답글을 쓸 수 있는 기능이 있습니다.

▼ 그림 17-48 흔히 볼 수 있는 쇼핑몰 게시판의 답글 쓰기 기능

4	윤 [기타] 배송연체되나요 [답변한로]	2018/09/11	1
3	↪ RE:윤 [기타] 배송연체되나요 [답변한로]	스타***	2018/09/11
2	윤 [소재] 소재가 [답변한로]	유윤*	2018/08/30
1	↪ RE:윤 [소재] 소재가 [답변한로]	스타***	2018/08/31

따라서 이번에는 게시판의 답글 쓰는 기능을 구현해 보겠습니다. 다음은 그 과정입니다.

- ① 글 상세창(viewArticle.jsp)에서 **답글쓰기**를 클릭하면 요청명을 /board/replyForm.do로 하여 부모 글 번호(parentNO)를 컨트롤러로 전송합니다.
- ② 답글 쓰기창(replyForm.jsp)에서 답변 글을 작성한 후 요청명을 /board/addReply.do로 하여 컨트롤러로 요청합니다.
- ③ 컨트롤러에서는 전송된 답글 정보를 게시판 테이블에 부모 글 번호와 함께 추가합니다.

1. sec03.brd07 패키지를 만들고 관련된 클래스 파일을 복사해 붙여 넣은 후 board06 폴더를 만들고 JSP 파일을 추가합니다.

▼ 그림 17-49 실습 파일 위치



2. 답글도 새 글이므로 답글 기능도 새 글 쓰기 기능과 비슷합니다. 다른 점은 답글창 요청(/replyForm.do) 시 미리 부모 글 번호를 parentNO 속성으로 세션(session)에 저장해 놓고, 답글을 작성한 후 등록을 요청(/addReply.do)하면 세션에서 parentNO를 가져와 테이블에 추가한다는 점입니다.

코드 17-41 pro17/src/sec03/brd07/BoardController.java

```
...
private void doHandle(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
    String nextPage = "";
    request.setCharacterEncoding("utf-8");
    response.setContentType("text/html; charset=utf-8");
```

```

HttpSession session; ... 딥글에 대한 부모 글 번호를 저장하기 위해 세션을 사용합니다.

... else if (action.equals("/replyForm.do"))
{
    int parentNO = Integer.parseInt(request.getParameter("parentNO"));
    session = request.getSession();
    session.setAttribute("parentNO", parentNO);
    nextPage = "/board06/replyForm.jsp";
}

... else if (action.equals("/addReply.do"))
{
    session = request.getSession();
    int parentNO = (Integer) session.getAttribute("parentNO");
    session.removeAttribute("parentNO");
    Map<String, String> articleMap = upload(request, response);
    String title = articleMap.get("title");
    String content = articleMap.get("content");
    String imageFileName = articleMap.get("imageFileName");
    articleVO.setParentNO(parentNO); 딥글의 부모 글 번호를 설정합니다.
    articleVO.setId("lee"); 딥글 작성자 ID를 lee로 설정합니다.
    articleVO.setTitle(title);
    articleVO.setContent(content);
    articleVO.setImageFileName(imageFileName);
    int articleNO = boardService.addReply(articleVO); 딥글을 테이블에 추가합니다.

    if (imageFileName != null && imageFileName.length() != 0)
    {
        File srcFile = new
            File(ARTICLE_IMAGE_REPO + "\\" + "temp" + "\\" + imageFileName);
        File destDir = new File(ARTICLE_IMAGE_REPO + "\\" + articleNO);
        destDir.mkdirs();
        FileUtils.moveFileToDirectory(srcFile, destDir, true);
    }

    PrintWriter pw = response.getWriter(); 딥글에 첨부한 이미지를 temp 폴더에서 딥글 번호 폴더로 이동합니다.
    pw.print("<script>" + " alert('답글을 추가했습니다.');" +
           + " location.href='"
           + request.getContextPath()
           + "/board/viewArticle.do?articleNO="
           + articleNO + ";" + "</script>");
}
return;
}

RequestDispatcher dispatch = request.getRequestDispatcher(nextPage);
dispatch.forward(request, response);
...

```

3. 답글 쓰기는 새 글 쓰기와 동일하게 BoardDAO의 insertNewArticle() 메서드를 이용합니다.

BoardService 클래스를 다음과 같이 수정합니다.

코드 17-42 pro17/src/sec03/brd07/BoardService.java

```
public int addReply(ArticleVO article) {  
    return boardDAO insertNewArticle(article);  
}
```

새 글 추가 시 사용한 insertNewArticle()  
메서드를 이용해 답글을 추가합니다.

4. 글 상세창(viewArticle.jsp)에서 답글쓰기를 클릭하면 fn\_reply\_form() 함수를 호출하면서 글 번호와 요청명을 함께 전달합니다. 그리고 다시 <form> 태그와 <input> 태그를 이용해 글 번호를 parentNO 속성으로 컨트롤러에 전달합니다.

코드 17-43 pro17/WebContent/board06/viewArticle.jsp

```
<c:set var="contextPath" value="${pageContext.request.contextPath}" />  
<script>  
    function fn_reply_form(url, parentNO) {  
        var form = document.createElement("form");  
        form.setAttribute("method", "post");  
        form.setAttribute("action", url);  
        var parentNOInput = document.createElement("input");  
        parentNOInput.setAttribute("type", "hidden");  
        parentNOInput.setAttribute("name", "parentNO");  
        parentNOInput.setAttribute("value", parentNO);  
        form.appendChild(parentNOInput);  
        document.body.appendChild(form);  
        form.submit();  
    }  
...  
</script>  
...  
<input type=button value="답글쓰기"  
onClick="fn_reply_form('${contextPath}/board/replyForm.do', ${article.articleNO})">  
...
```

답글쓰기 클릭 시 fn\_reply\_form() 함수를 호출하면서  
요청명과 글 번호를 전달합니다.

5. 답글을 입력한 후 컨트롤러에 /board/addReply.do로 요청하도록 replyForm.jsp를 다음과 같이 작성합니다.

코드 17-44 pro17/WebContent/board06/replyForm.jsp

```
<body>  
    <h1 style="text-align:center">답글쓰기</h1>  
    <form name="frmReply" method="post"
```

```

action="${contextPath}/board/addReply.do" enctype="multipart/form-data">
<table align="center">
  <tr>
    <td align="right"> 글쓴이:&nbsp; </td>
    <td><input type="text" size="5" value="lee" disabled /> </td>
  </tr>
  <tr>
    <td align="right">글제목:&nbsp; </td>
    <td><input type="text" size="67" maxlength="100" name="title" /></td>
  </tr>
  <tr>
    <td align="right" valign="top"><br>글내용:&nbsp;
    <td><textarea name="content" rows="10" cols="65" maxlength="4000">
      </textarea> </td>
  </tr>
  <tr>
    <td align="right">이미지파일 첨부: </td>
    <td> <input type="file" name="imageFileName" onchange="readURL(this);"/>
      </td>
    <td></td>
  </tr>
  <tr>
    <td align="right"> </td>
    <td>
      <input type="submit" value="답글반영하기" />
      <input type="button" value="취소" onClick="backToList(this.form)" />
    </td>
  </tr>
</table>
</form>
</body>

```

6. /board/listArticle.do로 요청하여 게시글 중에 부모 글을 클릭합니다.

▼ 그림 17-50 부모 글 클릭

글번호	작성자	제목	작성일
1	hong	상품평입니다.	2018. 9. 18.
2	kim	김유선입니다.	2018. 9. 18.
3	lee	[답변] 이용 후기입니다.	2018. 9. 18.
4	hong	안녕하세요	2018. 9. 18.
5	lee	[답변] 상품후기입니다..	2018. 9. 18.
6	hong	[답변] 답변입니다.	2018. 9. 18.
7	lee	[답변] 답변입니다.	2018. 9. 18.
8	hong	테스트글입니다.	2018. 9. 18.

글쓰기

7. 글 상세창에서 답글쓰기를 클릭합니다.

▼ 그림 17-51 답글쓰기 클릭

글번호	g
작성자 아이디	hong
제목	상품평입니다.
내용	질이 좋습니다. 수고하세요
이미지	
등록일자	2018. 9. 18.
<input type="button" value="파일 선택"/> 선택된 파일 없음	
<input type="button" value="수정하기"/> <input type="button" value="삭제하기"/> <input type="button" value="리스트로 돌아가기"/> <input style="outline: 2px solid #800080; border-radius: 5px; padding: 2px 10px;" type="button" value="답글쓰기"/>	

8. “상품평입니다.”에 대한 답글(“이용 후기입니다.”)을 작성한 후 답글반영하기를 클릭합니다.

▼ 그림 17-52 답글 작성 후 답글반영하기 클릭

**답글쓰기**

글쓴이:	lee
글제목:	이용 후기입니다.
글내용:	사용하기 편합니다. 감사합니다~
이미지파일 첨부:	<input type="button" value="파일 선택"/> duke2.jpg





지금은 개발 과정을 보여주기 위한 것이므로 부모 글과 댓글의 내용은 큰 상관이 없습니다.

### 9. 내가 작성한 답글의 내용을 한 번 더 표시합니다.

▼ 그림 17-53 답글 결과 표시

글번호	10
작성자 아이디	lee
제목	이용 후기입니다.
내용	사용하기 편합니다. 감사합니다~
이미지	
등록일자	파일 선택 선택된 파일 없음 2018. 11. 22.
<input type="button" value="수정하기"/> <input type="button" value="삭제하기"/> <input type="button" value="리스트로 돌아가기"/> <input type="button" value="답글쓰기"/>	

### 10. 글 목록을 보면 답글이 추가된 것을 볼 수 있습니다.

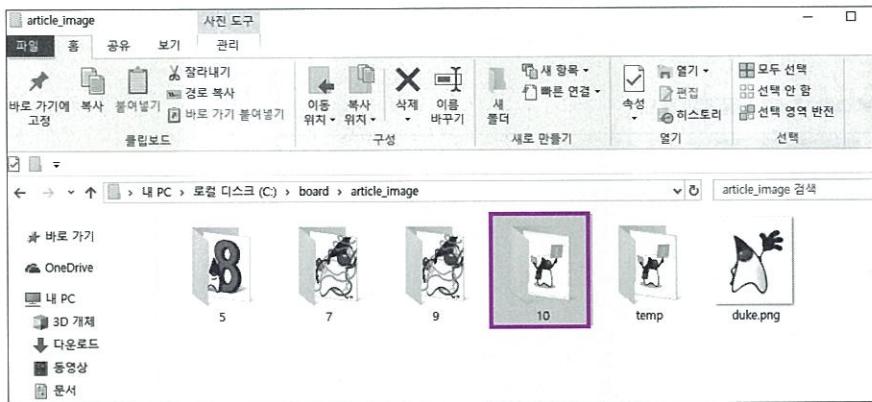
▼ 그림 17-54 부모 글에 답글 표시

글번호	작성자	제목	작성일
1	hong	상품평입니다.	2018. 9. 18.
2	lee	[답변] 이용 후기입니다.	2018. 11. 22.
3	kim	김유신입니다.	2018. 9. 18.
4	lee	[답변] 이용 후기입니다.	2018. 9. 18.
5	hong	안녕하세요	2018. 9. 18.
6	lee	[답변] 상품후기입니다..	2018. 9. 18.
7	hong	[답변] 답변입니다.	2018. 9. 18.
8	lee	[답변] 답변입니다.	2018. 9. 18.
9	hong	테스트글입니다.	2018. 9. 18.

글쓰기

11. 또한 답글에 추가한 이미지도 폴더에 저장됩니다.

▼ 그림 17-55 답글에 첨부한 이미지가 로컬 PC의 저장소에 저장



## 17.4.7 게시판 페이징 기능 구현

게시판 만들기 프로젝트의 마지막 단계입니다.

어떤 게시판이든 목록의 글이 많아지면 한 페이지에 모든 글이 표시되는 것이 아니라 다음과 같이 [1], [2], [3] .. 이렇게 페이지별로 표시됩니다. 이렇게 하는 것이 보기에도 더 좋고 사용자가 이용하기에도 편리하기 때문입니다.

▼ 그림 17-56 게시판의 페이징 기능

번호	제목	작성자	작성일	조회수	답글수
61559	2018 시나공 컴퓨터 활용능력 2급 실기(액셀 2010 사용자용) 최신기출 이해일서비스	서진희	2018.09.28	5	27/1
61558	[2013] 정보처리산업기사 실기 2017 정보처리산업기사 실기 관련입니다..도서 선택에 2017년판이 없어서 2013으로 문의합니다.	조현숙	2018.09.27	0	12/1
61557	시나공 토익 실전 모의고사 시즌 2 문장 질문	이순혁	2018.09.26	1	16/1
61556	모두의 알고리즘 with 파이썬 하노이의 탑 문제 관련	김재영	2018.09.25	75	16/2
61555	모두의 딥러닝 설치에러 외	김재영	2018.09.23	26	15/1
61554	에프터이펙트 CC 2018 무작정 따라하기 360 VR (part 4, part 5)	전수진	2018.09.23	524	12/1
61553	시나공 혼자서 끝내는 토익 950 실전 모의고사 Test3회 109번 질문	이예주	2018.09.23	118	17/1

이번에는 게시판의 페이징 기능을 구현해 보겠습니다. 먼저 글 목록에 페이징 기능이 어떻게 구현 되는지 그 원리부터 살펴봅시다.

그림 17-57은 게시판에 페이징 기능을 적용한 후 글 목록을 표시한 것입니다.

◆ 그림 17-57 게시판에 페이징 기능 적용

글번호	작성자	제목	작성일
1	hong	상품평	2018. 9. 18.
2	hong	상품평가	2018. 9. 18.
3	hong	상품 주문이 늦어요.	2018. 9. 18.
4	lee	[모든] 죄송합니다.	2018. 9. 18.
5	hong	상품평입니다.	2018. 9. 18.
6	hong	최길동글입니다.	2018. 9. 18.
7	kim	김유선입니다.	2018. 9. 18.
8	lee	[모든] 이용 후기입니다.	2018. 9. 18.
9	hong	안녕하세요	2018. 9. 18.
10	lee	[모든] 상품후기입니다..	2018. 9. 18.

1 2 3 4 5 6 7 8 9 10 next

글쓰기

여기서 하단에 보이는 숫자는 페이지 번호입니다. 한 페이지마다 10개의 글이 표시되고, 이 페이지 10개가 모여 한 개의 섹션(section)이 됩니다. 첫 번째 섹션은 첫 번째 페이지부터 열 번째 페이지까지입니다.

두 번째 섹션은 열한 번째 페이지부터 스무 번째 페이지까지입니다. 따라서 사용자가 글 목록 페이지에서 [2]를 클릭하면 브라우저는 서버에 section 값으로는 1을, pageNum 값으로는 2를 전송하는 것이죠. 그리고 글 목록에는 두 번째 페이지에 해당하는 글인 11에서 20번째 글을 테이블에서 조회한 후 표시합니다.

코드 17-45는 페이징 기능을 추가한 글 목록 조회 SQL문입니다.

코드 17-45 section과 pageNum으로 글 목록 조회하는 SQL문

```
SELECT * FROM (
    SELECT ROWNUM as recNum,
        LVL,
        articleNO,
        parentNO,
        title,
        content,
        id,
        writedate
    FROM (
        SELECT LEVEL as LVL,
            articleNO,
```

계층형으로 조회된 레코드의 ROWNUM(recNum)이 표시되도록 조회합니다.

계층형 SQL문으로 글을 계층별로 조회합니다.

```

        parentNO,
        title,
        content,
        id,
        writedate
    FROM t_board
    START WITH parentNO=0
    CONNECT BY PRIOR articleNO=parentNO
    ORDER SIBLINGS BY articleNO DESC
)
) where
recNum between(section-1)*100+(pageNum-1)*10+1 and (section-1)*100+pageNum*10;
--recNum between 1 and 10; section 값이 1이고 pageNum 값이 1인 경우입니다.

```

section과 pageNum 값으로 조건식의 recNum 범위를 정한 후 조회된 글 중 해당하는 값이 있는 경우 최종적으로 조회합니다.

페이지 기능을 구현하기 위해 서브 쿼리문과 오라클에서 제공하는 가상 컬럼인 ROWNUM을 이용합니다. ROWNUM은 select문으로 조회된 레코드 목록에 대해 오라클 자체에서 순서를 부여하여 레코드 번호를 순서대로 할당해 줍니다.

이 서브쿼리문의 실행 순서는 다음과 같습니다.

- ① 기존 계층형 구조로 글 목록을 일단 조회합니다.
- ② 그 결과에 대해 다시 ROWNUM(recNum)이 표시되도록 서브 쿼리문을 이용해 다시 한번 조회합니다.
- ③ ROWNUM이 표시된 두 번째 결과에서 section과 pageNum으로 계산된 where절의 between 연산자 사이의 값에 해당하는 ROWNUM이 있는 레코드들만 최종적으로 조회합니다.

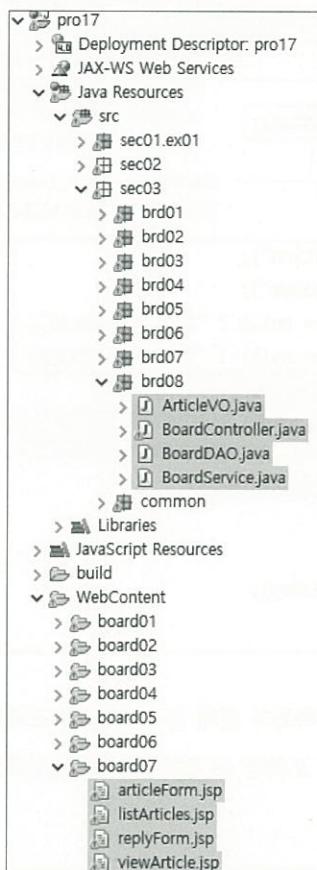
다음은 SQL Developer에서 section 값이 1이고 pageNum이 1인 경우, 즉 첫 번째 페이지에 표시되는 10개의 글을 조회한 결과입니다.

#### ▼ 그림 17-58 section 값이 1이고 pageNum이 1인 경우 조회된 글

RENUM	LVL	ARTICLENO	PARENTNO	TITLE	CONTENT	ID	WRITEDATE
1	1	13	0	상품평	잘쓰고 있습니다. 배송이 약간 늦습니다.	hong	18/09/18
2	2	1	12	0 상품평가	상품이 좋습니다. 감사합니다.	hong	18/09/18
3	3	1	10	0 상품 주문이 늦어요.	상품이 아직 도착하지 않았어요. 처리해 주세요.	hong	18/09/18
4	4	2	11	10 죄송합니다.	죄송합니다. 확인하고 빨리 처리해 드릴께요.	lee	18/09/18
5	5	1	9	0 상품평입니다.	질이 좋습니다. 수고하세요	hong	18/09/18
6	6	1	8	0 최근동글입니다.	배송이 빨라요.	hong	18/09/18
7	7	1	4	0 김유신입니다.	김유신 테스트글입니다.	kim	18/09/18
8	8	2	7	4 이용 후기입니다.	사용하기 편합니다. 감사합니다~~	lee	18/09/18
9	9	1	2	0 안녕하세요	상품 후기 입니다.	hong	18/09/18
10	10	2	6	2 상품후기입니다..	이순신씨의 상품 사용 후기를 올립니다!!	lee	18/09/18

1. 본격적인 실습을 위해 페이지 기능을 구현한 자바 클래스와 JSP를 다음과 같이 추가합니다.

▼ 그림 17-59 실습 파일 위치



2. `BoardController` 클래스를 다음과 같이 작성합니다. `/listArticle.do`로 최초 요청 시 `section`과 `pageNum`의 기본값을 1로 초기화합니다. 컨트롤러에서는 전달된 `section`과 `pageNum`을 `HashMap`에 저장한 후 DAO로 전달합니다.

코드 17-46 pro17/src/sec03/brd08/BoardController.java

```

...
if (action == null)
{
    String _section = request.getParameter("section");
    String _pageNum = request.getParameter("pageNum");
    int section = Integer.parseInt(((section == null) ? "1" : _section));
    int pageNum = Integer.parseInt(((pageNum == null) ? "1" : _pageNum));
}

```

최초 요청 시 또는 `/listArticle.do`로 요청 시  
section 값과 pageNum 값을 구합니다.

최초 요청 시 section 값과  
pageNum 값이 없으면 각  
각 1로 초기화합니다

```

Map<String, Integer> pagingMap = new HashMap<String, Integer>();
pagingMap.put("section", section); ← section 값과 pageNum 값을 HashMap에 저장한 후
pagingMap.put("pageNum", pageNum); ← 메서드로 넘깁니다. ← section 값과 pageNum 값은
Map articlesMap = boardService.listArticles(pagingMap); ← 해당 섹션과 페이지에 해당
articlesMap.put("section", section); ← 되는 글 목록을 조회합니다.
articlesMap.put("pageNum", pageNum); ← 브라우저에서 전송된 section과
request.setAttribute("articlesMap", articlesMap); ← pageNum 값을 articlesMap에 저장
nextPage = "/board07/listArticles.jsp"; ← 한 후 listArticles.jsp로 넘깁니다.
} else if (action.equals("/listArticles.do"))
{
    String _section = request.getParameter("section");
    String _pageNum = request.getParameter("pageNum");
    int section = Integer.parseInt((_section == null) ? "1" : _section));
    int pageNum = Integer.parseInt((_pageNum == null) ? "1" : _pageNum));
    Map<String, Integer> pagingMap = new HashMap<String, Integer>(); ← 조회된 글 목록을 articlesMap으로 바인
    pagingMap.put("section", section); ← 딩하여 listArticles.jsp로 넘깁니다.
    pagingMap.put("pageNum", pageNum); ← 경우 section 값과 pageNum 값을 가져옵니다.
    Map articlesMap = boardService.listArticles(pagingMap);
    articlesMap.put("section", section);
    articlesMap.put("pageNum", pageNum);
    request.setAttribute("articlesMap", articlesMap);
    nextPage = "/board07/listArticles.jsp";

```

3. BoardService 클래스에서는 페이징 기능에 필요한 글 목록과 전체 글 수를 각각 조회할 수 있도록 다음과 같이 구현합니다. HashMap을 생성한 후 조회한 두 정보를 각각 속성으로 저장합니다.

**Tip** JSP로 넘겨줘야 할 정보가 많을 경우에는 각 request에 바인딩해서 넘겨도 되지만 HashMap을 사용해 같은 종류의 정보를 묶어서 넘기면 편리합니다.

#### 코드 17-47 pro17/src/sec03/brd08/BoardService.java

```

...
public Map listArticles(Map pagingMap)
{
    Map articlesMap = new HashMap(); ← 전달된 pagingMap을 사용해 글 목록을 조회합니다.
    List<ArticleVO> articlesList = boardDAO.selectAllArticles(pagingMap);
    int totArticles = boardDAO.selectTotArticles(); ← 테이블에 존재하는 전체 글 수를 조회합니다.
    articlesMap.put("articlesList", articlesList); ← 조회된 글 목록을 ArrayList에 저장한 후 다시
    articlesMap.put("totArticles", totArticles); ← articlesMap에 저장합니다.
    return articlesMap; ← 전체 글 수를 articlesMap에 저장합니다.
}

```

```

public List<ArticleVO> listArticles()
{
    List<ArticleVO> articlesList = boardDAO.selectAllArticles();
    return articlesList;
}
...

```

4. BoardDAO 클래스를 다음과 같이 작성합니다. 전달받은 section과 pageNum 값을 이용해 SQL문으로 조회합니다.

코드 17-48 pro17/src/sec03/brd08/BoardDAO.java

```

public List<String, Integer> selectAllArticles(Map<String, Integer> pagingMap)
{
    List<ArticleVO> articlesList = new ArrayList<ArticleVO>();
    int section = (Integer) pagingMap.get("section");           ← 전송된 section과 pageNum
    int pageNum = (Integer) pagingMap.get("pageNum");           ← 값을 가져옵니다.
    try
    {
        conn = dataFactory.getConnection();
        String query = "SELECT * FROM ( " + "select ROWNUM as recNum,"
            + "LVL," + "articleNO,"
            + "parentNO," + "title,"
            + "id," + "writeDate"
            + " from (select LEVEL as LVL, "
            + "articleNO," + "parentNO," + "title," + "id,"
            + "writeDate" + " from t_board"
            + " START WITH parentNO=0" + " CONNECT BY PRIOR articleNO = parentNO"
            + " ORDER SIBLINGS BY articleNO DESC)" + " ) "
            + " where recNum between(?-1)*100+(-1)*10+1 and (?-1)*100+?*10";
        ↑
        System.out.println(query);
        pstmt = conn.prepareStatement(query);
        pstmt.setInt(1, section);
        pstmt.setInt(2, pageNum);
        pstmt.setInt(3, section);
        pstmt.setInt(4, pageNum);
        ResultSet rs = pstmt.executeQuery();
        while (rs.next())
        {
            int level = rs.getInt("lvl");
            int articleNO = rs.getInt("articleNO");
            int parentNO = rs.getInt("parentNO");
            String title = rs.getString("title");

```

```
String id = rs.getString("id");
Date writeDate = rs.getDate("writeDate");
ArticleVO article = new ArticleVO();
article.setLevel(level);
article.setArticleNO(articleNO);
article.setParentNO(parentNO);
article.setTitle(title);
article.setId(id);
article.setWriteDate(writeDate);
articlesList.add(article);
} // end while
rs.close();
 pstmt.close();
conn.close();
} catch (Exception e)
{
    e.printStackTrace();
}
return articlesList;
}

...
public int selectTotArticles()
{
try
{
conn = dataFactory.getConnection();
String query = "select count(articleNO) from t_board ";
System.out.println(query);
pstmt = conn.prepareStatement(query);
ResultSet rs = pstmt.executeQuery();
if (rs.next())
    return (rs.getInt(1));
rs.close();
pstmt.close();
conn.close();
} catch (Exception e)
{
    e.printStackTrace();
}
return 0;
}
}
```

5. 이제 화면을 구현하는 JSP 페이지인 listArticles.jsp를 다음과 같이 작성합니다. 전체 글 수 (totArticles)가 100개를 넘는 경우, 100인 경우, 100개를 넘지 않는 경우로 나누어 페이지 번호를 표시하도록 구현합니다.

전체 글 수가 100개가 넘지 않으면 전체 글 수를 10으로 나눈 몫에 1을 더한 값이 페이지 번호로 표시됩니다. 예를 들어 전체 글 수가 13개이면 10으로 나누었을 때의 몫인 1에 1을 더해 2가 페이지 번호로 표시됩니다.

만약 전체 글 수가 100개일 때는 정확히 10개의 페이지가 표시되며, 100개를 넘을 때는 다음 section으로 이동할 수 있도록 마지막 페이지 번호 옆에 next를 표시합니다.

코드 17-49 pro17\WebContent\board07\listArticles.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"
isELIgnored="false" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
    request.setCharacterEncoding("UTF-8");
%>
<c:set var="contextPath" value="${pageContext.request.contextPath}" />
<c:set var="articlesList" value="${articlesMap.articlesList}" />
<c:set var="totArticles" value="${articlesMap.totArticles}" />
<c:set var="section" value="${articlesMap.section}" />
<c:set var="pageNum" value="${articlesMap.pageNum}" />
<style>
.no-underline {text-decoration:none;}
.sel-page{text-decoration:none;color:red;}
.cls1 {text-decoration:none;}
.cls2{text-align:center; font-size:30px;}
</style>
...
<c:when test="${ !empty articlesList }">
<c:forEach var="article" items="${articlesList }" varStatus="articleNum">
<tr align=center>
<td width="5%">${articleNum.count}</td>
<td>${article.writer }</td>
<td> ${article.passwd} </td>
...
<div class="txt_center">
<c:if test="${ totArticles != null }">
<c:choose>
<c:when test="${ totArticles >100 }">

```

```

<c:forEach var="page" begin="1" end="10" step="1">
    <c:if test="${section > 1 & page==1 }">
        <a class="no-underline" href=
            "${contextPath}/board/listArticles.do?section=
            ${section-1}&pageNum=${(section-1)*10
            +1 }">&nbsp;
    pre </a>
</c:if>           └── 섹션 값 2부터는 앞 섹션으로 이동할 수 있는 pre를 표시합니다.
    <a class="no-underline" href="${contextPath}/board/listArticles.do?section=
        ${section}&pageNum=${page}">${(section-1)*10
        +page } </a>
    <c:if test="${page == 10 }">
        <a class="no-underline" href="${contextPath}/board/listArticles.do?section=
            ${section+1}&pageNum=${section*10+1}">&nbsp;
        next</a>
    </c:if>
</c:forEach>      └── 페이지 번호 10 오른쪽에는 다음 섹션으로 이동할 수 있는 next를 표시합니다.
</c:when>          └── 전체 글 수가 100개일 때는 첫 번째 섹션의
    <c:when test="${totArticles == 100 }">  10개 페이지만 표시하면 됩니다.
        <c:forEach var="page" begin="1" end="10" step="1">
            <a class="no-underline" href="#">${page} </a>
        </c:forEach>
    </c:when>          └── 전체 글 수가 100개보다 적을
                        때 페이지를 표시합니다.
    <c:when test="${totArticles< 100 }">          └── 글 수가 100개가 되지 않으므로 표시되는 페이지는
        <c:forEach var="page" begin="1" end="${ totArticles/10 +1}" step="1">  10개가 되지 않고, 전체 글 수를 10으로 나누어 구
            <c:choose>                                              한 뒷에 1을 더한 페이지까지 표시됩니다.
                <c:when test="${page==pageNum }">
                    <a class="sel-page" href="${contextPath}/board/listArticles.do?section=
                        ${section}&pageNum=${page}">${page
                    } </a>          └── 페이지 번호와 컨트롤러에서 넘어온 pageNum이 같은 경우 페이지 번호를
                </c:when>                                              빨간색으로 표시하여 현재 사용자가 보고 있는 페이지임을 알립니다.
            <c:otherwise>
                <a class="no-underline" href="${contextPath}/board/listArticles.do?section=
                    ${section}&pageNum=${page}">${page
                } </a>
            </c:otherwise>          └── 페이지 번호를 클릭하면 section 값과 pageNum 값을
        </c:choose>                                              컨트롤러로 전송합니다.
    </c:forEach>
</c:when>
<c:choose>
<c:when>
<c:otherwise>
</c:choose>
</div>
...

```

6. <http://localhost:8090/pro17/board/listArticles.do>로 요청하여 실행 결과를 확인합니다.

그림 17-60은 전체 글 수를 13으로 하여 요청한 결과입니다(전체 글 수가 적으면 먼저 글쓰기 기능을 이용해 원하는 개수만큼 글을 추가합니다).

▼ 그림 17-60 글 목록창 요청 결과

글번호	작성자	제목	작성일
1	hong	상품평	2018. 9. 18.
2	hong	상품평가	2018. 9. 18.
3	hong	상품 주문이 늦어요.	2018. 9. 18.
4	lee	[답변] 칙송합니다.	2018. 9. 18.
5	hong	상품평입니다.	2018. 9. 18.
6	hong	최길동글입니다.	2018. 9. 18.
7	kim	김유산입니다.	2018. 9. 18.
8	lee	[답변] 이용 후기입니다.	2018. 9. 18.
9	hong	안녕하세요	2018. 9. 18.
10	lee	[답변] 상품후기입니다..	2018. 9. 18.

전체 글 수에 대해 총 두  
페이지가 표시됩니다.

1
2

[글쓰기](#)

7. 여기서 두 번째 페이지인 [2]를 클릭하면 <http://localhost:8090/pro17/board/listArticles.do?section=1&pageNum=2>로 요청합니다.

▼ 그림 17-61 두 번째 페이지 요청 결과

글번호	작성자	제목	작성일
1	hong	[답변] 답변입니다.	2018. 9. 18.
2	lee	[답변] 답변입니다.	2018. 9. 18.
3	hong	테스트글입니다.	2018. 9. 18.

1
2

[글쓰기](#)

8. 글 수를 좀 더 늘려볼까요? BoardService 클래스에서 totArticles를 170으로 설정합니다.

▼ 그림 17-62 전체 글 수를 170으로 설정

```

14°   public Map listArticles(Map pagingMap) {
15     Map articlesMap = new HashMap();
16     List<ArticleVO> articlesList = boardDAO.selectAllArticles(pagingMap);
17     int totArticles = boardDAO.selectTotArticles();
18     articlesMap.put("articlesList", articlesList);
19     //articlesMap.put("totArticles", totArticles);
20     articlesMap.put("totArticles", 170);
21   }
22 }
```

9. 브라우저에서 글 목록창을 출력합니다. 전체 글 수가 100개를 넘으므로 next가 표시된 것을 볼 수 있습니다.

▼ 그림 17-63 전체 글 수가 100개가 넘는 경우 next 표시

글번호	작성자	제목	작성일
1	hong	상품평	2018. 9. 18.
2	hong	상품평가	2018. 9. 18.
3	hong	상품 주문이 늦어요.	2018. 9. 18.
4	lee	(답변) 죄송합니다.	2018. 9. 18.
5	hong	상품 평입니다.	2018. 9. 18.
6	hong	최저가글입니다.	2018. 9. 18.
7	kim	김유신입니다.	2018. 9. 18.
8	lee	(답변) 이용 후기입니다.	2018. 9. 18.
9	hong	안녕하세요	2018. 9. 18.
10	lee	(답변) 상품후기입니다..	

1 2 3 4 5 6 7 8 9 10 next

전체 글 수가 100개를 넘으므로 next가 표시됩니다.

글쓰기

지금까지 답변형 게시판을 구현해 봤습니다. 지면의 한계로 소스 코드 중 중요한 부분 위주로 살펴봤는데, 예제 파일로 제공하는 실제 소스 코드를 보면서 직접 구현해 보기 바랍니다.

한걸음 더 나아가 전체 글 수가 100개를 넘을 경우 사용자가 위치한 페이지 번호가 빨간색으로 표시되도록 구현할 수도 있습니다. CSS 속성을 사용하면 현재 클릭해서 보고 있는 페이지 번호를 빨간색으로 표시하여 구분할 수 있겠죠?

이처럼 아직 구현하지 않은 여러 가지 세부 기능들은 여러분이 직접 구현해 보기 바랍니다. 게시판 기능은 모든 웹 프로그래밍의 기본이므로 게시판 기능만 구현할 수 있다면 다른 기능도 쉽게 구현할 수 있을 것입니다.