

# JSP & Servlet

(JavaServerPage & Servlet)

## 2. 자바 웹 개발 환경구축

### 2.1 자바 웹 개발에 필요한 프로그램 다운로드 및 설치

#### ▶ JDK 1.8.xx(Java SE Development Kit 8) 다운로드

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>에서 JDK 1.8.xx(Java Se Development Kit 8uxx) 버전을 다운로드 후 기본 폴더에 설치(이미 Java 수업시간에 설치했음)

#### ▶ 서블릿 컨테이너 : Tomcat 9.0.xx 다운로드

<http://tomcat.apache.org/> 사이트에 접속하여 좌측의 Downlond 메뉴에서 Tomcat 9.0 클릭 -> Tomcat 9 Software Downloads 페이지가 나타나는데 이 페이지에서 아래로 스크롤하여 Binary Distributions 부분의 Core 아래쪽에 있는 “64-bit Windows zip (pgp, sha512)” 링크를 통해 톰캣 서버 파일(apache-tomcat-9.x.xx-windows-x64.zip)을 다운로드 한 후 C:\DeveloperTools\ 폴더에 복사하고 “여기에 압축풀기”로 압축을 풀어 설치하면 된다. 만약 리눅스 또는 맥을 사용한다면 “tar.gz (pgp, sha512)” 링크를 통해 톰캣 서버 파일(apache-tomcat-9.x.xx.tar.gz)을 다운로드 받아 설치하면 된다.

#### ▶ Eclipse IDE for Java EE Developers

<https://www.eclipse.org/downloads/packages/> 에 접속한 후 아래로 스크롤하여 “Eclipse IDE for Enterprise Java and Web Developers”의 오른쪽에 위치한 “Windows x86\_64” 링크를 클릭 해 이동한 페이지에서 “Download” 버튼을 클릭하면 이클립스 프로그램을 zip 파일로 압축한 버전 (예: eclipse-jee-20xx-xx-R-win32-x86\_64.zip)을 다운로드 받을 수 있다. 다운로드 받은 이클립스 프로그램을 C:\DeveloperTools\ 폴더에 복사한 후 “여기에 압축풀기”로 압축을 풀어 설치하고 “eclipse\_jee”로 폴더 명을 변경하자. 만약 윈도우즈 OS가 아닌 리눅스를 사용한다면 Linux x86\_64를 맥을 사용한다면 macOS x86\_64를 다운로드 받아 설치하면 된다.

### 2.2 환경변수 설정

#### ▶ JDK 설치 후 환경변수 설정(필수 사항)

JAVA\_HOME 환경 변수를 Tomcat 실행시 참조하게 된다.

윈도우 탐색기에서 내 PC에 마우스 우 클릭 -> 속성 메뉴를 선택하면 제어판의 시스템 화면이 나타나는데 이 화면에서 “고급 시스템 설정” 메뉴를 클릭해 시스템 속성 화면을 띄운다. 그리고 시스템 속성 화면 아래쪽의 “환경변수” 버튼을 클릭하여 환경 변수 설정 화면을 띄우고 이 화면에서 아래쪽 시스템 변수(S) 부분의 “새로 만들기(W)...” 버튼을 클릭해 시스템 변수를 아래와 같이 설정 한다. 이미 Java 수업시간에 환경 변수 설정을 완료함

- 변수 이름 : JAVA\_HOME
- 변수 값 : C:\Program Files\Java\jdk1.8.0\_271

위에서 지정한 변수 값은 Java SE 8u271 버전을 설치하고 환경 변수를 설정한 예로 변수 값 지정은 실제 설치한 JDK 버전에 따라서 폴더 명을 적절히 지정하고 확인 버튼 클릭하면 된다.

환경변수를 새로 추가했으면 이 변수를 이용해 JDK의 경로를 PATH에 등록해야 시스템이 어떤 위치에서든 JDK를 인식할 수 있게 된다. “시스템 변수(S)”의 변수 리스트 중에서 PATH를 선택하고 “편집” 버튼을 클릭하여 맨 뒷부분으로 이동한다. 이 때 다른 부분이 변경되지 않도록 주의하여 PATH의 맨 뒷부분으로 이동하고 맨 뒤에 세미콜론(:)이 없으면 먼저 세미콜론(:)을 입력한 후 “%JAVA\_HOME%\bin;”을 입력하고 확인 버튼을 클릭하여 설정을 완료한다.

여기까지 정상적으로 환경설정이 완료되었다면 이클립스의 Windows - Preferences - Server - Runtime Environments 메뉴에서 시스템에 설치한 톰캣 서버를 등록한다.

이미 컴퓨터에 OracleXE가 설치되어 실행 중 이라면 오라클이 8080포트를 기본 httpport로 사용하기 때문에 8080 포트를 기본으로 사용하여 실행하는 톰캣 서버와 포트 충돌이 발생하여 톰캣이 정상적으로 실행되지 못하고 에러가 발생하게 된다.

이 문제를 해결하기 위해 sqlplus를 실행하여 sys 아이디로 로그인 한 후 아래 명령으로 오라클이 사용하는 httpport를 8080에서 8090으로 변경 해주면 톰캣 서버를 정상적으로 실행할 수 있을 것이다.

### ▶ sys 아이디로 오라클 접속하기

명령 프롬프트에서 sqlplus sys as sysdba 명령으로 오라클에 접속하거나 SQL Developer를 사용해 sys 계정으로 접속하여 다음 내용을 참고해 오라클이 사용하는 http 포트 변경을 진행한다.

### ▶ 오라클 포트 조회

```
select dbms_xdb.gethttpport from dual
select dbms_xdb.getftpport from dual
```

### ▶ 오라클 포트 변경

```
exec dbms_xdb.sethttpport(8090);
exec dbms_xdb.setftpport(2100);
```

위와 같은 방법으로 OracleXE의 httpport와 ftpport를 변경하고 톰캣 서버를 다시 실행하면 에러 없이 정상적으로 잘 실행된다.

## 2.3 이클립스에서 jsp 페이지 만들기

- JSPStudyCh02/src/main/webapp/today.jsp

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

```

<%@ page import="java.util.*" %>
<%
    /* 날짜 데이터를 다루기 위해서 Calendar 객체를 생성
     * 아래는 시스템의 현재 날짜와 시간 정보를 가진 Calendar 객체가 생성된다.
     */
    Calendar today = Calendar.getInstance();
%>
<!DOCTYPE html>
<html>
<head>
<title>오늘의 날짜</title>
</head>
<body>
    오늘은 <%= today.get(Calendar.YEAR) %>년
        <%= today.get(Calendar.MONTH) + 1 %>월
        <%= today.get(Calendar.DAY_OF_MONTH) %>일 입니다.<br/>
</body>
</html>

```

## 2.4 이클립스에서 서블릿 만들기

- com.jspstudy.ch02.servlet

@WebServlet("/today")

**public class** TodayServlet **extends** HttpServlet {

@Override

**public void** doGet(HttpServletRequest request,  
 HttpServletResponse response) **throws** IOException, ServletException {

```

    /* 날짜 데이터를 다루기 위해서 Calendar 객체를 생성
     * 아래는 시스템의 현재 날짜와 시간 정보를 가진 Calendar 객체가 생성된다.
     */

```

Calendar today = Calendar.getInstance();

```

    /* 웹 브라우저에 응답할 문서의 형식과 문자 셋 지정
     * 문서의 형식과 문자 셋을 잘못지정하면 한글과 같은 문자는 깨져서 표시된다.
     * 응답 객체의 문서 형식과 문자 셋은 응답 객체의 스트림에 접근하기 전에 설정해야
     * 제대로 동작한다. 만약 응답 객체의 스트림에 접근한 후에 응답 객체의 문서 형식과
     * 문자 셋을 지정하면 영문, 숫자 등은 제대로 표시되지만 한글은 깨져서 표시된다.
     */

```

response.setContentType("text/html; charset=utf-8");

// 웹 브라우저로 응답 결과를 보내기 위해 응답 객체를 통해 스트림객체를 구한다.

PrintWriter out = response.getWriter();

```

/* 다음과 같이 응답 객체에 연결된 스트림의 출력 메소드를 이용해 HTML 문서 형식에
 * 맞게 내용을 작성하면 이 데이터가 브라우저로 전달되어 브라우저 화면에 표시된다.
 */
out.println("<html>");
out.println(" <head><title>오늘의 날짜</title></head>");
out.println(" <body>");
out.print("    오늘은 " + today.get(Calendar.YEAR) + "년 ");
out.print("    (today.get(Calendar.MONTH) + 1) + "월 ");
out.print("    today.get(Calendar.DAY_OF_MONTH) + "일 입니다.");
out.println("</body>");
out.println("</html>");

// 작업이 끝나면 스트림을 닫는다.
out.close();
}
}

```

#### [연습문제 2-1] 1부터 100까지 짝수의 합을 출력하는 JSP와 서블릿 만들기

이클립스에서 JSPStudyCh02Exam이라는 DynamicWebProject를 생성하고 1 ~ 100까지 짝수의 합을 구하여 다음과 같이 출력하는 JSP페이지와 서블릿을 작성해 테스트 해 보자.

1부터 100까지 짝수의 합 : 2550

### 3. JSP의 동작원리와 구성요소

#### 3.1 JSP 페이지의 동작원리

클라이언트가 웹 브라우저를 통해 웹 서버에 요청을 전달하면 일반적인 HTML 문서와 같은 정적인 문서는 웹 서버에 의해 그대로 웹 브라우저로 전송된다. 하지만 JSP 페이지는 그림 3-1과 같이 웹 서버의 요청을 받은 서블릿 컨테이너(Servlet Container, 웹 컨테이너라고도 함)에서 실행되어 그 결과만 HTML 문서 형태로 변환된 후 웹 서버를 통해 브라우저에게 전송된다. 요즘과 같이 복잡적이고 다양한 요청을 처리해야 하는 웹 애플리케이션에서는 단순히 HTML 코드로 구성된 페이지는 찾아보기 힘들고 HTML 태그와 프로그래밍 요소가 하나로 구성된 JSP와 같은 페이지가 대부분이다. 이렇게 정적요소를 포함한 여러 가지 프로그래밍 요소로 작성되는 JSP 페이지는 클라이언트의 요청이 웹 서버로 전달되면 웹 서버는 다시 클라이언트의 요청을 서블릿 컨테이너로 전달하여 서블릿 컨테이너에게 처리를 요청하게 된다. 웹 서버로부터 요청을 받은 서블릿 컨테이너는 해당 JSP 페이지를 서블릿 클래스(자바 클래스의 일종)의 소스 파일로 변환하고 변환된 Java 소스 파일을 컴파일한 후 서블릿 객체를 생성한다. 서블릿 컨테이너에 의해 생성된 서블릿 객체는 서블릿 컨테이너에 적재되어 클라이언트의 요청을 처리하고 그 결과를 HTML 문서로 변환하여 서블릿 컨테이너를 통해 웹 서버에 전달하게 된다. 그리고 웹 서버는 서블릿 컨테이너로부터 전달받은 HTML 문서를 클라이언트의 요청에 대한 응답으로 브라우저로 전송한다. 이후 동일한 요청에 대한 처리는 JSP 페이지가 변경되기 전까지 별도의 소스 파일 변환이나 컴파일 작업 없이 서블릿 컨테이너에 적재된 서블릿이 처리하게 된다.

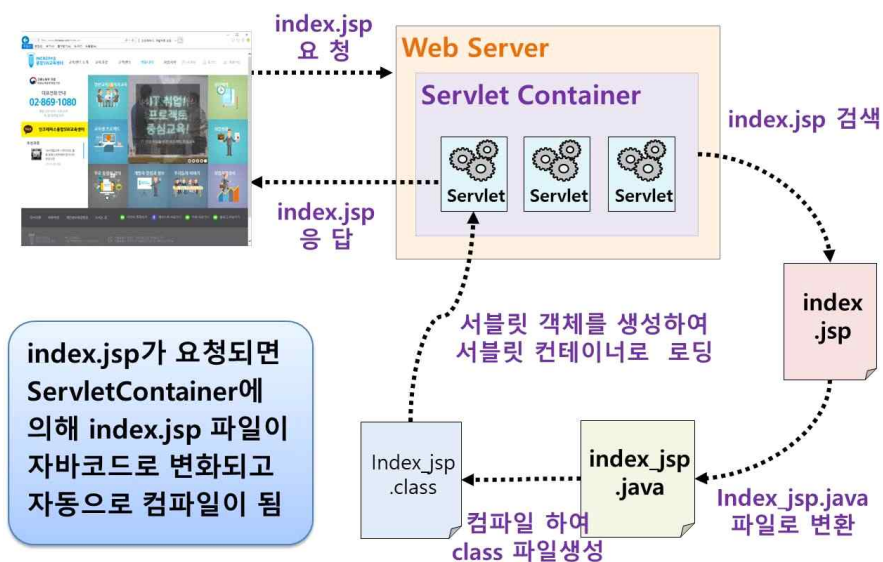


그림 3-1 JSP의 동작원리

#### 3.2 JSP의 스크립팅 요소

클라이언트의 다양한 요청을 처리하는 JSP 페이지는 HTML요소와 이미지 등과 같은 정적 데이터를 포함하여 동적 웹 페이지를 구성할 수 있는 다양한 자바 웹 기술을 사용할 수 있다. JSP는 클라이언트의 요청을 처리한 결과를 동적으로 HTML 문서 형식으로 만들기 위해서 스크립팅 요소(Scripting Elements)를 제공하고 있다. 스크립팅 요소는 JSP 페이지를 구성하는 요소의 하나로 자바 코드를

기술할 수 있는 스크립틀릿(Scriptlet)과 결과를 출력하기 위한 표현식(Expression) 그리고 변수나 메서드를 선언할 수 있는 선언부(Declaration)로 구성된다.

## 1) JSP 페이지의 서블릿 클래스 변환

다음 예제를 톰캣 서버와 이클립스에서 한 번 이상 실행하여 그림 3-2와 같이 JSP 파일이 서블릿 클래스 파일로 변경된 것을 확인하고 서블릿 소스 코드로 변환된 java 파일을 열어 JSP 페이지가 서블릿 소스 코드로 어떻게 변환되었는지 확인해 보자.

- JSPStudyCh03/src/main/webapp/scriptletToServlet.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%--
    여기는 JSP의 주석
    JSP 페이지에 작성된 HTML 태그와 텍스트는 서블릿 클래스 안에서
    현재 위치에 HTML 태그와 텍스트를 출력하는 자바 소스 코드로 변환된다.
--%>
<!DOCTYPE html>
<html>
<head><title>JSP 페이지의 서블릿 클래스 변환</title></head>
<body>
    <%
        /* 스크립틀릿에서의 주석은 자바 주석을 그대로 사용하면 된다.
        *
        * JSP 페이지의 스크립틀릿에 작성된 자바 명령문은
        * 서블릿 클래스 안에서 자바 명령문으로 변환된다.
        **/
        int sum = 0;
        for(int i = 0; i <= 100; i += 2) {
            sum += i;
        }
    %>

    <%--
        JSP 페이지에서 표현식에 사용된 변수나 자바 코드는
        서블릿 클래스 안에서 결과를 출력하는 자바 코드로 변환된다.
    --%>
    1 ~ 100 짝수의 합 : <%= sum %>
</body>
</html>
```

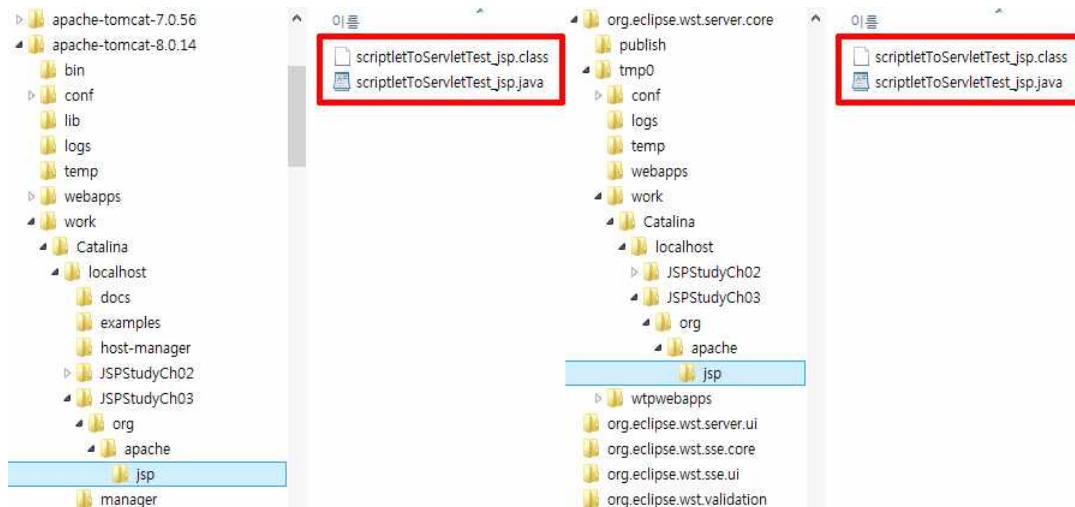


그림 3-2 톰캣 서버와 이클립스에서 JSP 페이지가 서블릿 클래스로 변환된 파일

## 2) 스크립팅 요소 사용하기

JSP 페이지의 스크립틀릿은 서블릿 클래스로 변환되면 모두 `_jspService()` 메서드 안에 자바 코드로 작성되므로 앞 쪽의 스크립틀릿에서 선언된 변수는 다음에 오는 스크립틀릿에서 사용할 수 있다.

### - JSPStudyCh03/src/main/webapp/scriptingElement.jsp

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

여기는 JSP의 주석

선언부에는 클래스 멤버와 인스턴스 멤버를 선언할 수 있다.

JSP 페이지의 선언부에 선언된 변수와 메서드는 서블릿으로 변환된 클래스의 멤버가 되므로 접근지정자, `final`, `static` 등의 예약어를 사용할 수 있다.

한 가지 주의할 점은 서블릿으로 변환된 클래스의 인스턴스는 멀티 스레드로 동작하기 때문에 선언부에 인스턴스 변수를 선언하게 되면 동기화에 문제가 발생할 수 있다. 그러므로 선언부에 인스턴스 변수를 선언하는 것은 피하고 변수가 필요할 경우 `static` 상수로 선언해 사용하는 것이 바람직하다.

jsp의 선언부는 요즘에는 거의 사용하지 않는 기법이다.

```
--%>
<%!
    private int add(int a, int b) {
        return a + b;
    }
%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>스크립팅 요소(Scripting Element) 사용하기</title>
```



```

</head>
<body>
  <!--
    이 부분은 JSP의 주석문 이다.
    JSP 페이지의 스크립틀릿(Scriptlet)에 기술된 코드는 JSP가 서블릿 클래스로
    변환될 때 _jspService() 메서드 안에 기술되므로 위쪽의 스크립틀릿에서 선언된
    변수를 아래쪽의 여러 스크립틀릿과 표현식에서 사용할 수 있다.
  -->
  <%
    int sum = 0;
    for(int i = 1; i <= 100; i++) {
      sum += i;
    }
  %>

  <!-- 표현식 안의 변수나 자바 코드는 결과를 출력하는 서블릿 코드로 변환된다. -->
  1 ~ 100 합 : <%= sum %><br/>

  <%
    for(int i = 1; i <= 10; i++) {
  %>
  <!-- 위의 스크립틀릿의 for 문에서 정의한 i를 출력 -->
  <%= i %>번<br/>
  <%
    }

    // 위쪽의 스크립틀릿에서 선언한 sum에 0을 대입한다.
    sum = 0;
  %>
  sum : <%= sum %><br/>

  <!-- 위쪽의 선언부에 선언한 메서드를 호출하여 덧셈 결과를 출력 -->
  10 + 20 = <%= add(10, 20) %>
</body>
</html>

```

### 3.3 지시자(Directive)

JSP 페이지에서 필요한 설정 정보를 지정할 수 있도록 제공하는 것이 지시자이다.

인코딩이나 콘텐츠 타입과 같이 JSP 페이지에서 필요한 설정 정보나 클래스의 임포트 등에 사용할 수 있는 page 지시자, 표준 태그 라이브러리를 사용할 수 있는 taglib 지시자, 정적으로 다른 JSP 페이지를 포함 시킬 수 있는 include 지시자 등이 있다.

## 1) page 지시자와 JSP 주석

- JSPStudyCh03/src/main/webapp/jspPageDirective.jsp

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

```
<!--
```

여기는 JSP에서 지원하는 주석이다.

page 지시자는 현재 JSP 페이지에서 필요한 설정 정보나 클래스의 임포트 등에 사용할 수 있다. 위의 page 지시자에서 contentType은 브라우저로 전송할 응답 결과에 대한 문서의 형식과 문자 셋을 지정하는 속성으로 브라우저는 이 속성에 지정된 문서형식과 문자 셋을 기준으로 문서를 해석하여 화면에 출력하게 된다.

문서 형식을 MIME(Multipurpose Internet Mail Extensions) 타입이라고 하며 이메일의 내용을 설명하기 위해 정의되었으나 오늘날 메일 뿐만 아니라 HTTP 등의 프로토콜을 이용해 전송되는 문서의 내용을 설명하기 위해 사용하고 있다. MIME 타입을 생략하게 되면 기본 값은 "text/html"이며 charset을 생략하면 기본 값은 톰캣의 기본 문자 셋인 "iso-8859-1"을 사용하게 된다.

pageEncoding은 이 문서의 인코딩 방식을 지정하는 속성으로 서블릿 컨테이너가 JSP 페이지를 분석할 때 JSP 페이지가 어떤 문자 셋으로 작성되었는지 검사하고 그 문자 셋을 이용해 JSP 페이지를 읽어 해석하게 된다.

서블릿 컨테이너는 pageEncoding 속성을 먼저 검색하고 contentType 속성의 charset의 값을 검색하게 되는데 만약 pageEncoding 속성을 지정하지 않고 contentType 속성의 charset에 문자 셋을 잘못 지정하게 되면 한글과 같은 유니코드 문자가 깨지는 현상이 나타나게 된다.

```
--%>
```

```
<%
```

```
int sum = 0;
```

```
/* 1부터 100까지 반복문 - 자바의 여러줄 주석
```

```
* 스크립틀릿(Scriptlet)에서 자바의 한 줄 주석과 여러 줄 주석을 사용할 수 있다.
```

```
**/
```

```
for(int i = 1; i <= 100; i++) {
```

```
    // sum 변수에 i의 값을 누적하여 더해 1~100까지 합을 구한다.
```

```
    sum += i;
```

```
}
```

```
%>
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<!-- 여기부터 HTML Head - HTML 주석 -->
```

```
<head>
```

```
<title>JSP의 pageEncoding 속성</title>
```

```
</head>
```

```
<!-- HTML 본문 시작 -->
```

```
<body>
```

```
<!--
```

JSP 주석은 스크립팅요소 안에서는 사용할 수 없고 HTML 태그 사이에서 사용할 수 있다. 위의 스크립틀릿에서 계산한 결과를 출력

```
--%>
1 ~ 100까지의 합 : <%= sum %><br/>
</body>
</html>
```

## 2) include 지시자(Directive)

- JSPStudyCh03/src/main/webapp/jspIncludeDirective.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="java.util.Date, java.util.Calendar" %>
<%
    Calendar ca = Calendar.getInstance();
    Date now = ca.getTime();
%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>다른 JSP 페이지 포함하기</title>
</head>
<body>
    <h2>◆ 오늘의 메뉴 ◆</h2>
    - 명태조림 10,000원<br/>
    - 버섯 매운탕 10,000원<br/>
    - 불고기 정식 11,000원<br/><br/>

    <%--
        include 지시자를 사용해 컴파일 타임에 다른 JSP 페이지의 내용을
        현재 JSP 페이지에 포함할 수 있다. 이렇게 include 지시자를 사용하면
        JSP 페이지가 톰캣 서버에 의해서 최초로 실행될 때 현재 JSP 페이지
        (부모 페이지라고 함)에 아래에 include 지시자로 지정한 JSP 페이지
        (자식 페이지라고 함)가 하나로 합쳐져서 Servlet 클래스로 변환되고
        컴파일 되어 하나의 서블릿으로 동작하게 된다.
    --%>
    <%@ include file="includeMenu.jsp" %>

    <%-- java.lang 패키지는 기본 패키지로 import 없이 사용 가능하다. --%>
    <b><%= String.format("%1$TY년 %1$Tm월 %1$Td일", now) %></b>
</body>
</html>
```

- JSPStudyCh03/src/main/webapp/includeMenu.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<h3>♣ 저녁 특선 메뉴</h3>
- 참치회 정식 30,000원<br/><br/>
```

### 3.4 표현 언어(Expression Language)

표현 언어는 표현식을 대체하기 위한 취지로 만들어졌으며 자바 개발자 보다 자바 언어에 대해 지식이 부족한 웹 개발자 들이 보다 쉽게 JSP 페이지를 작성할 수 있도록 고안된 언어이다. JSP 페이지의 스크립팅 요소를 사용하면 자바 언어를 사용해 코드를 작성할 수 있으므로 자바의 장점을 그대로 JSP 페이지에서 사용할 수 있다는 장점이 있지만 자바 문법에 대한 이해가 없이는 스크립팅 요소를 사용해 JSP 페이지를 작성할 수는 없다.

표현 언어는 JSP의 pageContext, request, session, application 내장객체의 영역에 저장된 속성에 접근할 수 있으며 수치연산자, 관계연산자, 논리연산자를 제공하고 Servlet 규격 3.0부터 자바 클래스의 메서드를 호출할 수 있다. 또한 표현 언어만의 내장객체를 제공하고 있다.

표현언어는 jstl 1.0 규약에 처음 소개된 스크립트 언어로 JSP 2.0부터 정식으로 포함되었으며 JSP 2.2의 표현 언어 버전은 2.2이다. 그리고 MVC 패턴에서 View 역할을 담당하는 JSP 페이지를 스크립팅요소를 사용하는 것 보다 간결하게 작성할 수 있다는 장점을 가지고 있다.

#### ▶ 표현식과 EL을 이용해 학생정보 출력하기

- com.jspstudy.ch03.vo

// 학생 한 명의 정보를 저장하는 VO(Value Object) 클래스

```
public class Student {

    private String name;
    private int age;
    private String gender;

    public Student() { }

    public Student(String name, int age, String gender) {
        this.name = name;
        this.age = age;
        this.gender = gender;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

```

public int getAge() {
    return age;
}
public void setAge(int age) {
    this.age = age;
}
public String getGender() {
    return gender;
}
public void setGender(String gender) {
    this.gender = gender;
}
}

```

#### - JSPStudyCh03/src/main/webapp/studentInfo.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!-- Student 클래스를 사용하기 위해 page 지시자를 이용해 import 하고 있다. --%>
<%@ page import="com.jspstudy.ch03.Student" %>
<%
    // Student 객체를 생성하고 각 프로퍼티를 설정하고 있다.
    Student student1 = new Student("이순신", 23, "남성");
    Student student2 = new Student();
    student2.setName("홍길동");
    student2.setAge(25);
    student2.setGender("남성");

    // request 내장객체에 속성 이름을 지정하여 Student 객체를 저장한다.
    request.setAttribute("student", student1);

    /* 요청을 처리한 결과를 다른 JSP에 출력하기 위해 pageContext 내장객체의
     * forward() 메서드를 이용해 studentInfoResult.jsp로 포워딩 하여 프로그램의
     * 제어 흐름을 이동시킨다. 일반적으로 서버 프로그래밍에서 요청을 처리하는
     * 코드와 요청을 처리한 결과를 출력하는 코드를 분리하여 작성하는데 이럴 경우
     * 아래와 같이 포워딩 기법을 사용해 프로그램 흐름을 다른 JSP로 이동시킨다.
     */
    pageContext.forward("studentInfoResult.jsp");
%>

```

#### - JSPStudyCh03/src/main/webapp/studentInfoResult.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!-- Student 클래스를 사용하기 위해 page 지시자를 사용해 import 하고 있다. --%>
<%@ page import="com.jspstudy.ch03.Student"%>

```

```
<%
/* JSP 페이지로 부터 포워딩되어 넘어온 request 영역의 속성에 저장된 데이터를 읽는다.
 * Object 타입으로 넘어오기 때문에 다운 캐스팅이 필요하다.
 * 아래에서 request.getAttribute("student")를 호출했을 때 request 내장객체의
 * 속성에 student 라는 이름을 가진 객체가 존재하지 않으면 null을 반환한다.
 */
Student s = (Student) request.getAttribute("student");
```

```
%>
<!DOCTYPE html>
<html>
<head>
<title>표현식과 EL을 이용해 학생정보 출력하기</title>
</head>
<body>
<!--
```

위쪽의 스크립틀릿에서 request 내장객체의 속성 값을 읽어서 변수에 저장하고 그 변수의 값을 아래와 같이 표현식을 이용해 출력할 수 있다.

studentInfo.jsp 페이지를 실행해서 현재 페이지로 forward 되어야 아래에 데이터가 제대로 출력된다. 만약 studentInfo.jsp 페이지를 실행하지 않고 현재 페이지만 실행하게 되면 위쪽의 스크립틀릿에서 request 영역의 속성을 읽어 올 때 null 값을 받으므로 NullPointerException이 발생한다. 그러므로 studentInfo.jsp를 실행하여 그 페이지에서 Student 클래스의 인스턴스가 생성되고 request 내장객체의 속성에 student 라는 이름으로 객체가 저장되어 이 페이지로 포워딩 될 수 있도록 studentInfo.jsp를 먼저 실행해야 한다.

```
-->
<h2>학생정보 출력하기 - 표현식(Expression)</h2>
이 름 : <%= s.getName() %><br/>
나 이 : <%= s.getAge() %><br/>
성 별 : <%= s.getGender() %><br/><br/>
```

```
<!--
표현언어(EL)를 사용해 내장객체의 속성 명을 지정해 값을 읽어 올 수 있다.
스크립팅요소를 사용하는 것에 비해 간단히 내장객체 영역에 저장된 속성의 값을
읽을 수 있다. 내장객체의 속성에 객체가 저장되어 있으면 내장객체의 속성명과
dot 연자자(.)를 사용해 객체의 프로퍼티 값을 읽어 올 수 있다.
표현언어(EL)로 객체의 프로퍼티를 읽기 위해서는 그 객체의 클래스에 읽어 올
프로퍼티에 대한 getter 메서드가 반드시 존재해야 한다.
그렇지 않으면 JasperException이 발생한다.
```

이 JSP 페이지만 실행하면 request 내장객체의 속성에서 읽어올 것이 없으므로 null 값이 되지만 표현 언어(EL)를 사용하면 NullPointerException은 발생하지 않고 화면에 아무것도 출력되지 않는다.

아래는 이전 studentInfo.jsp에서 request 영역의 속성에 student 라는 이름으로 Student 클래스의 인스턴스를 저장하였으므로 스크립틀릿을 사용하지 않아도 표현언어(EL)을 이용해 request 영역의 속성에 바로 접근할 수 있다. 이렇게 표현언어는 pageContext, request, session, application 내장 객체의 속성에 차례로 접근해 지정한 이름의 속성 이름이 있는지 탐색하여 데이터가 존재하면 그 데이터를 읽어 출력하고 데이터가 없으면 아무것도 출력되지 않는다.

```
-->
<h2>학생정보 출력하기 - 표현 언어(Expression Language)</h2>
이 름 : ${ student.name }<br/>
나 이 : ${ student.age }<br/>
성 별 : ${ student.gender }<br/>
</body>
</html>
```

### 3.5 표준 태그 라이브러리(JSTL)와 커스텀 태그(Custom Tag)

커스텀 태그 중에 많이 사용되는 태그들을 모아 JSTL(JSP Standard Tag Library) 규약을 만들어 제공하는 것이 표준 태그 라이브러리로 XML 태그 형태로 되어 있으며 특정 동작을 담당하는 다양한 태그를 제공한다. 표준 태그 라이브러리는 XML 태그 형태로 되어 있어 스크립틀릿과 반복문을 사용해 JSP 페이지를 작성할 때 보다 간편하게 코드를 구성할 수 있고 가독성이 뛰어난 JSP 페이지를 작성할 수 있다. 표준 태그 라이브러리는 코어, XML 지원, 국제화와 포매팅 지원, 데이터베이스 지원, 함수지원 등의 라이브러리로 구성되어 있다.

커스텀 태그는 JSP가 기본적으로 제공하는 표준액션 태그와는 다르게 별도로 라이브러리를 클래스패스에 참조해야 사용할 수 있다.

다음 예제를 실행하기 전에 <https://tomcat.apache.org/download-taglibs.cgi> 사이트에서 jstl 라이브러리를 다운로드 받아 **실습프로젝트/webapp/WEB-INF/lib** 폴더에 복사해서 JSTL 라이브러리가 참조되어야 InternalServer 에러 없이 정상적으로 실행될 수 있다.

위의 사이트에서 taglibs-standard-impl-1.2.5.jar 파일과 taglibs-standard-spec-1.2.5.jar 파일만 다운받아 **실습프로젝트/webapp/WEB-INF/lib** 폴더에 복사하면 된다.

#### ▶ JSTL과 EL를 이용해 로또 당첨 번호 리스트 출력

- com.jspstudy.ch03.vo

// 로또 당첨 번호 1등과 2등의 번호를 저장하는 VO(Value Object) 클래스

```
public class LottoNum {
```

```
    // 회차 정보, 로또 당첨 번호 1등 6개 번호, 2등 보너스 번호
```

```
    private String times;
```

```
    private int num1;
```

```
    private int num2;
```

```
    private int num3;
```

```
    private int num4;
```

```
    private int num5;
```

```
    private int num6;
```

```

private int bonusNum;

public LottoNum(String times, int num1, int num2, int num3,
    int num4, int num5, int num6, int bonusNum) {
    this.times = times;
    this.num1 = num1;
    this.num2 = num2;
    this.num3 = num3;
    this.num4 = num4;
    this.num5 = num5;
    this.num6 = num6;
    this.bonusNum = bonusNum;
}

public String getTimes() {
    return times;
}

public void setTimes(String times) {
    this.times = times;
}

public int getNum1() {
    return num1;
}

public void setNum1(int num1) {
    this.num1 = num1;
}

public int getNum2() {
    return num2;
}

public void setNum2(int num2) {
    this.num2 = num2;
}

public int getNum3() {
    return num3;
}

public void setNum3(int num3) {
    this.num3 = num3;
}

public int getNum4() {
    return num4;
}

public void setNum4(int num4) {
    this.num4 = num4;
}

public int getNum5() {

```



```

        return num5;
    }
    public void setNum5(int num5) {
        this.num5 = num5;
    }
    public int getNum6() {
        return num6;
    }
    public void setNum6(int num6) {
        this.num6 = num6;
    }
    public int getBonusNum() {
        return bonusNum;
    }
    public void setBonusNum(int bonusNum) {
        this.bonusNum = bonusNum;
    }
}

```

- JSPStudyCh03/src/main/webapp/lottoNum.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="com.jspstudy.ch03.vo.*, java.util.*" %>
<%
    // 회 차별 로또 당첨 번호를 저장할 ArrayList 객체 생성
    ArrayList<LottoNum> lottoList = new ArrayList<LottoNum>();

    // 각 회 차별 로또 당첨 번호를 저장할 LottoNum 객체를 생성하고 ArrayList 에 저장
    LottoNum lotto = new LottoNum("968회", 2, 5, 12, 14, 24, 39, 33);
    lottoList.add(lotto);

    lotto = new LottoNum("969회", 3, 9, 10, 29, 40, 45, 7);
    lottoList.add(lotto);

    lotto = new LottoNum("970회", 9, 11, 16, 21, 28, 36, 5);
    lottoList.add(lotto);

    lotto = new LottoNum("971회", 2, 6, 17, 18, 21, 26, 7);
    lottoList.add(lotto);

    lotto = new LottoNum("972회", 3, 6, 17, 23, 37, 39, 26);
    lottoList.add(lotto);

    // Request 영역에 속성 이름을 지정하고 위에서 추출한 로또번호 리스트를 저장

```

```

request.setAttribute("lottoList", lottoList);

/* 요청을 처리한 결과를 다른 JSP에 출력하기 위해 pageContext 내장객체의
 * forward() 메서드를 이용해 lottoNumResult.jsp로 포워딩 하여 프로그램의
 * 제어 흐름을 이동시킨다. 일반적으로 서버 프로그래밍에서 요청을 처리하는
 * 코드와 요청을 처리한 결과를 출력하는 코드를 분리하여 작성하는데 이럴 경우
 * 아래와 같이 포워딩 기법을 사용해 프로그램 흐름을 다른 JSP로 이동시킨다.
 */
pageContext.forward("lottoNumResult.jsp");
%>

```

#### - JSPStudyCh03/src/main/webapp/lottoNumResult.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%--
표준태그라이브러리(JSTL)의 코어 라이브러리를 사용하기 위해 taglib 지시자를 사용해
접두어와 URI 식별자를 지정하고 있다. JSTL 라이브러리는 JSP 페이지에서 공통으로
사용하는 코드의 집합으로 코어, 포매팅, 데이터베이스, XML처리, 함수지원 등의 기능을
제공한다. JSTL 라이브러리를 사용하기 위해서는 필요한 라이브러리를 구분하여 지정해야
하는데 바로 URI 식별자가 이 라이브러리를 구분하는 역할을 한다. 접두어는 JSP
페이지에서 라이브러리를 간편히 사용할 수 있도록 하기 위해 사용하는 접두어 일 뿐이다.
taglib 지시자를 사용해 아래와 같이 지정하게 되면 코어 라이브러리의 URI 식별자가
접두어 c에 연결되고 JSP 페이지에서는 접두어 c를 사용하기만 하면 된다.
코어 라이브러리는 말 그대로 핵심적인 기능을 제공하는 라이브러리로 프로그래밍에서
필요한 변수 선언, 조건문, 반복문 등의 기능을 간편하게 사용할 수 있도록 지원하는
라이브러리 이다.
--%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>EL과 JSTL을 이용해 로또 당첨 번호 리스트 출력</title>
<style>
    ul {
        list-style-type: none;
    }
    ul > li {
        height: 30px;
    }
    li > span {
        color: blue;
    }
</style>

```

```

</head>
<body>
  <h2>로또 당첨 번호 리스트</h2>
  <%--
    표준 태그 라이브러리와 표현 언어를 이용해 로또 당첨 번호가
    존재하면 반복문을 이용해 로또 당첨 번호 리스트를 출력한다.
  --%>
  <c:if test="${not empty lottoList}">
    <ul>
      <%--
        <c:forEach> 태그는 자바의 for문과 forEach문 두 가지 기능을 제공한다.
        자바의 for문과 같이 초기값부터 순차적으로 반복하기 위해 아래와 같은
        속성을 지정하여 사용한다. var 속성에는 forEach문 안에서 사용할 변수
        이름을 지정하고 begin 속성은 var 속성에 지정한 변수의 시작 값을 지정하며
        end 속성에는 변수의 끝 값을 지정한다.(i <= end 가 적용된다.)
        step 속성은 생략할 수 있고 생략하게 되면 기본 값은 1이 된다.

        <c:forEach var="i" begin="0" end="10" step="1" >

        아래에 사용된 코드는 forEach문의 기능을 하는 코드로 items 속성에 배열
        변수의 이름을 지정하고 var 속성에 forEach문 안에서 배열의 각 항목을
        저장하는 변수의 이름을 지정한다. items 속성에 지정할 수 있는 변수는
        스크립틀릿에서 선언한 변수를 표현식<%= member %>으로 지정할 수
        있고 아래와 같이 속성에 저장된 속성 이름을 EL 식을 이용해 지정할 수도 있다.
        items 속성에 지정할 수 있는 데이터는 배열, Collection 객체, Iterator 객체,
        Enumeration 객체, Map 객체와 콤마(,)로 구분된 문자열 등을 지정할 수 있다.

        이전 페이지인 lottoNum.jsp에서 request.setAttribute("lottoList", lottoList);를
        이용해 하나의 요청 범위 안에서 유효한 request 객체의 속성에 "lottoList"라는
        이름으로 저장했으므로 아래와 같이 EL을 이용해 items 속성에 값으로 지정하면
        lottoList의 길이만큼 forEach 문이 반복되어 로또 당첨 번호 리스트가 출력된다.
      --%>
      <c:forEach var="lottoNum" items="${ lottoList }" >

        <%--
          표현언어(EL)로 객체의 프로퍼티를 읽기 위해서는 아래와 같이 속성
          이름에 dot(.) 연산자를 사용해 객체의 프로퍼티 이름을 지정하면 읽어
          올 수 있는데 이때 해당 객체는 반드시 getter 메서드를 가지고 있어야
          한다. 그렇지 않으면 JasperException이 발생한다.
        --%>
        <li>${ lottoNum.times }차 - ${ lottoNum.num1 }, ${ lottoNum.num2},
          ${ lottoNum.num3}, ${ lottoNum.num4}, ${ lottoNum.num5},
          ${ lottoNum.num6} + <span>보너스번호</span>
          ${ lottoN
      </ul>

```

```

</c:if>

<%--
로또 당첨 번호가 존재하지 않으면 존재하지 않는다는 메시지를 출력한다.
--%>
<c:if test="${ empty lottoList }">
로또번호가 존재하지 않습니다.
</c:if>
</body>
</html>

```

### [연습문제 3-1] 학생 리스트 출력하기

먼저 연습문제를 작성할 JSPStudyCh03Exam 이라는 프로젝트를 생성하고 앞의 “표현식과 EL을 이용해 학생정보 출력하기” 예제에서 사용한 com.jspstudy.ch03.vo.Student 클래스를 연습문제 프로젝트로 복사해서 이 클래스를 이용해 5명에 해당하는 Student 객체를 생성하고 ArrayList에 담아서 다음과 같이 스크립틀릿과 표현식을 이용한 방법과 JSTL과 EL을 이용한 방법으로 학생 리스트를 출력하는 프로그램을 작성하시오

#### 스크립틀릿과 표현식을 이용한 학생 리스트 출력

- 홍길동(23) - 남성
- 어머니(21) - 여성
- 왕호감(22) - 남성
- 왕빛나(23) - 여성
- 이나래(25) - 여성

#### JSTL과 EL을 이용한 학생 리스트 출력

- 홍길동(23) - 남성
- 어머니(21) - 여성
- 왕호감(22) - 남성
- 왕빛나(23) - 여성
- 이나래(25) - 여성

## 4. 서블릿(Servlet)

### 4.1. 서블릿

서블릿은 JSP와 함께 자바 웹 프로그래밍의 핵심으로 기준이 되는 기술이다. 우리가 서블릿 기술을 활용해 웹 애플리케이션을 구현하기 위해서는 HttpServlet 클래스를 상속한 서블릿 클래스를 작성해야 한다. 이 서블릿 클래스는 컴파일 된 후 톰캣(Tomcat)과 같은 서블릿 컨테이너(웹 컨테이너, WAS - Web Application Server라고도 함) 안에서 서블릿 객체로 생성되어 실행된다. 서블릿 컨테이너는 서블릿 클래스를 서블릿 객체로 만든 다음 그 객체를 초기화하여 웹 서비스 가능 상태로 만들어 요청을 처리하게 된다.

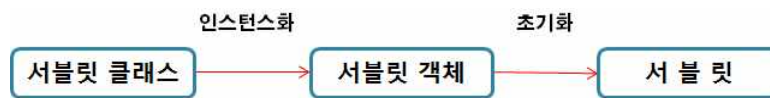


그림 4-1 서블릿

서블릿 컨테이너는 여러 사용자의 동시 요청을 처리하기 위해 하나의 서블릿 클래스를 가지고 멀티스레딩 방식으로 서블릿을 운영한다. JSP 페이지와 서블릿 클래스는 서블릿 컨테이너 안에서 모두 서블릿으로 동작되며 상호 연계되어 사용된다. 또한 MVC(Model View Controller) 패턴에서 JSP는 뷰 페이지를 담당하고 서블릿은 컨트롤러 역할을 담당한다.

### 4.2. HTTP와 서블릿의 동작원리

#### 4.2.1 HTTP(Hyper Text Transfer Protocol)

우리가 웹 브라우저를 통해 인터넷 서핑을 할 때 기본적으로 사용하는 프로토콜이 바로 HTTP 이다. 이 HTTP는 웹 브라우저를 통해서 웹 서버와 통신하여 HTML 문서를 볼 수 있게 해 주는 프로토콜로 비 연결(Connectionless)과 무 상태(Stateless) 속성을 가지고 있다. 비 연결이란 클라이언트의 요청에 응답한 후 바로 연결을 끊어 다음 페이지로 연결이 유지되지 않는 것을 의미하고 무 상태란 서비스를 요청한 사용자가 누구인지 서버가 모르는 상태를 의미한다. 다시 말해 클라이언트의 요청이 들어오면 서버는 그 요청을 처리하고 그 결과를 웹 브라우저에 전송한 후 클라이언트와의 연결을 바로 끊어 버리기 때문에 사용자가 다음 페이지로 이동 할 때 이전 페이지에서 사용한 정보는 다음 페이지에서 기억하지 못하게 되며 사용자 정보 또한 지속적으로 유지할 수 없게 된다. 이런 이유로 HTTP를 비 연결지향형 프로토콜이라고 부른다.

웹 브라우저는 HTTP에 맞게 웹 서버에 요청(Request)을 보내고 웹 서버는 그 요청에 따른 응답(Response)을 HTTP에 맞게 웹 브라우저로 보내준다. 이 때 사용되는 HTTP 메소드(요청방식)에는 Get, Post, Head, Put, Delete, Trace, Options 등 여러 가지가 있지만 일반적으로 Get과 Post 메소드 외에는 자주 사용되지 않기 때문에 이 두 가지 메소드에 대해서만 알아볼 것이다.

#### ▶ Get 메소드

HTTP header에 정보를 실어 보내며 클라이언트의 요청시 별도로 메소드를 지정하지 않으면 기본적으로 적용 되는 메소드이다. Get 메소드는 클라이언트가 서버에 요청시 필요 한 데이터를 URL 뒤에 붙여 전송하므로 전송되는 데이터가 사용자의 눈에 노출되어 보안성 이 좋지 않으며 요청 데이터의 크기도 2000자(브라우저 마다 상이함) 정도 가능하다.

## ▶ Post 메소드

HTTP body에 정보를 실어 보내므로 데이터 크기의 제한이 없으며 사용자의 눈에 보이지 않기 때문에 Get 방식의 요청보다 보안성 좋다.

## 4.2.2 서블릿의 동작원리

javax.servlet 패키지는 프로토콜에 독립적인 서블릿 클래스를 만들기 위해 필요한 클래스를 제공하며 javax.servlet.http 패키지는 HTTP의 고유한 기능(GET, POST)을 제공하는 서블릿 클래스를 만들기 위해 필요한 클래스들을 제공한다.

모든 서블릿 클래스는 javax.servlet.Servlet 인터페이스를 구현해야 한다. 이 Servlet 인터페이스를 구현한 서블릿 클래스가 javax.servlet.GenericServlet 클래스이고 GenericServlet을 상속받아 구현한 클래스가 javax.servlet.http.HttpServlet 클래스이다. 프로토콜에 독립적인 서블릿 클래스를 작성 하려면 GenericServlet 클래스를 상속받아야 하고 HTTP의 고유한 기능을 사용하기 위한 서블릿 클래스를 작성하려면 HttpServlet 클래스를 상속받아 구현하면 된다.

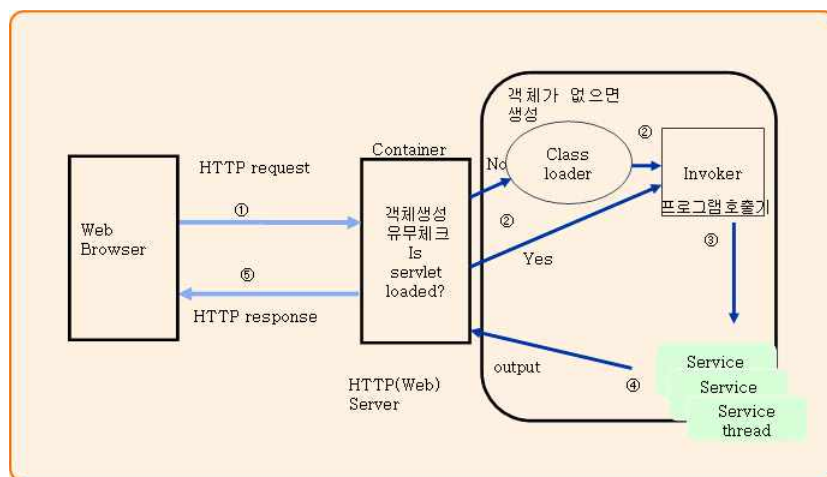


그림 4-2 HTTP 서블릿의 요청과 응답 처리과정

서블릿 클래스는 웹 브라우저의 호출에 의해서 실행되기 때문에 main() 메소드가 없으며 그 대신 HttpServlet 클래스에 구현된 service() 메소드가 서블릿 컨테이너(톰캣)에 의해 호출된다. 이 service() 메소드는 특별한 경우가 아니면 자식 클래스에서 재정의(Override) 하지 않고 대신 GET 방식의 요청을 처리하기 위해 doGet() 메소드와 POST 방식의 요청을 처리하기 위해 doPost() 메소드를 자식 클래스에서 재정의 하면 된다. 이 두 메소드는 javax.servlet.http 패키지의 HttpServletRequest 객체와 HttpServletResponse 객체를 파라미터로 넘겨받아 HTTP 요청과 응답에 대한 처리를 구현한다.

그림 4-2는 HTTP 서블릿의 요청과 응답 과정을 이해를 돕기 위해 그림으로 나타낸 것이다.

## 1) 일반적인 HTML 파일 요청시 처리과정(정적 파일)

클라이언트의 요청이 웹 서버에 전달되면 해당 HTML 문서가 웹 서버에 존재하는지 검사한 후 요청한 HTML 문서가 존재하면 웹 서버가 직접 그 HTML 문서를 클라이언트에게 응답 데이터로 전송한다. HTML 문서가 존재하지 않으면 문서가 존재하지 않는다는 메시지가 출력된다.

## 2) 서블릿 요청시 처리과정(동적 파일)

클라이언트로부터 서블릿에 대한 요청이 들어오면 웹 서버는 클라이언트의 요청 정보를 묶어서 WAS 또는 서블릿 컨테이너(Servlet Container, 이하 서블릿 컨테이너)에게 처리를 의뢰한다. 서블릿 컨테이너는 웹 서버로부터 전달 받은 요청 정보를 분석한 후 요청에 맞는 서블릿을 검색한다. 서블릿이 컨테이너에 존재하면 서블릿을 실행하여 클라이언트의 요청을 처리하고 그 결과를 HTML 문서로 만들어 그림 4-3과 같이 웹 서버를 통해 클라이언트에게 응답하게 된다.

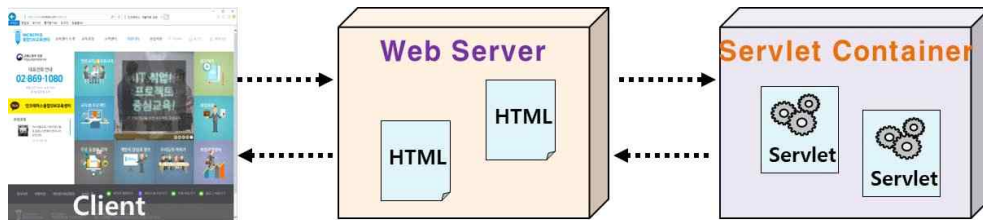


그림 4-3 서블릿 요청시 처리과정

### ▶ 요청 및 응답 객체 생성

서블릿 컨테이너는 웹 서버로부터 넘겨받은 요청 정보를 바탕으로 `HttpServletRequest` 객체와 `HttpServletResponse` 객체를 생성한다.

```
HttpServletRequest request = new HttpServletRequest();  
HttpServletResponse response = new HttpServletResponse();
```

### ▶ 서블릿 객체 생성

서블릿 컨테이너는 서블릿 클래스가 이미 서블릿 컨테이너에 로딩 되어 있다면 바로 해당 서블릿 클래스의 객체를 생성하고 로딩 되어 있지 않다면 해당 서블릿 클래스 파일(.class)을 로딩한 후 서블릿 객체를 생성한다.

```
UserServlet servlet = new UserServlet();
```

## ▶ 서블릿 실행

서블릿 컨테이너는 서블릿 객체를 생성한 후 그 서블릿 객체에 정의되어 있는 doGet(), doPost() 메소드를 호출하여 사용자 요청에 대한 응답을 처리하게 된다.

```
servlet.doGet(request, response);  
servlet.doPost(request, response);
```

일반적으로 서블릿 클래스는 HttpServlet 클래스를 상속받아 구현하게 되는데 서블릿 컨테이너는 사용자 요청에 대한 처리를 위해서 개발자가 구현한 서블릿 클래스의 doGet() 메소드와 doPost() 메소드를 직접 호출하는 것이 아니라 개발자가 구현한 서블릿 클래스의 슈퍼 클래스인 HttpServlet 클래스의 service() 메소드를 통해 doGet() 메소드와 doPost() 메소드를 간접적으로 호출하게 된다. 이 HttpServlet 클래스는 GenericServlet 클래스를 상속받아 HTTP 프로토콜에 적합하게 구현된 서블릿 클래스이다.

사용자 요청에 대한 웹 서버와 서블릿 컨테이너의 실제 응답 처리과정은 그림 4-4와 같다.

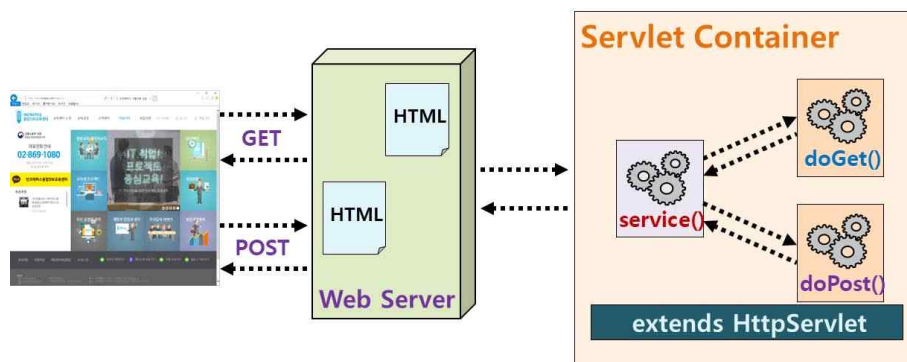


그림 4-4 사용자 요청에 대한 처리과정



## 4.3 서블릿 클래스 작성

### 4.3.1 서블릿 클래스 작성하기

개발자가 작성하는 서블릿 클래스의 상속 관계를 보면 아래 그림 4-5와 같으며 HTTP 서비스를 위한 서블릿 클래스를 작성할 때는 그림 4-6과 같이 `javax.servlet.http.HttpServlet` 클래스를 상속하도록 만들어야 하며 `public` 클래스로 선언해야 한다. 또한 서블릿 클래스 안에서 GET방식 요청을 처리하기 위해서 `doGet()` 메소드를 재정의 하고 POST 방식 요청을 처리하기 위해서는 `doPost()` 메소드를 재정의해야 한다.

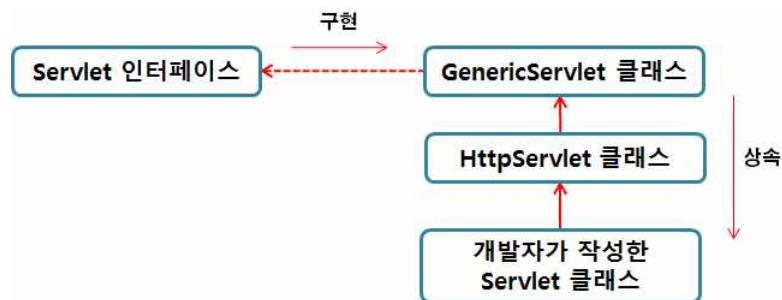


그림 4-5 서블릿 클래스의 상속 관계도

그림 4-6의 `doGet()` 메소드와 같이 첫 번째 파라미터로 `javax.servlet.http.HttpServletRequest` 타입을 받고 두 번째 파라미터로 `javax.servlet.http.HttpServletResponse` 타입을 받을 수 있도록 정의하고 `javax.servlet.ServletException`과 `java.io.IOException`을 선언해야 한다. 이들 Exception 처리가 필요치 않으면 생략할 수 있지만 다른 Exception은 선언할 수 없다.

사용자 요청에 대한 여러 가지 정보는 `doGet()` 메소드 또는 `doPost()` 메소드의 첫 번째 파라미터인 `HttpServletRequest` 객체를 통해서 구할 수 있으며 사용자 요청에 대한 응답 처리는 두 번째 파라미터인 `HttpServletResponse` 객체를 통해서 브라우저로 응답할 데이터를 작성하면 된다.

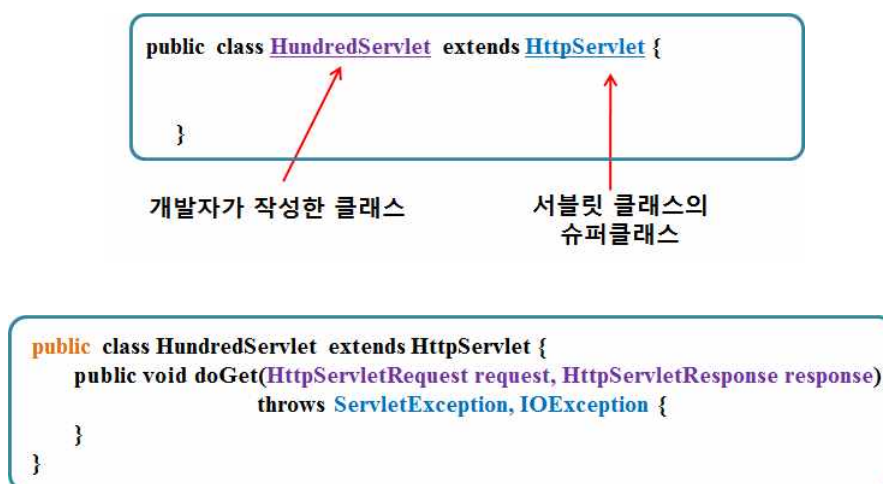


그림 4-6 서블릿 클래스 작성

- com.jspstudy.ch04.servletbasic.NowServlet

```
public class NowServlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {

        /* 웹 브라우저에 출력될 응답문서의 형식과 문자 셋을 지정
         * ContentType 설정은 Response 객체로 부터 PrintWriter
         * 객체를 얻기 전에 지정해야 한글이 깨지지 않고 제대로 동작한다.
         */
        response.setContentType("text/html; charset=UTF-8");

        // 응답 처리를 위해 Response 객체로부터 스트림 객체를 얻음
        PrintWriter out = response.getWriter();
        Calendar cal = Calendar.getInstance();

        /* HTML 문서 형식에 맞게 웹 브라우저로 전송할 데이터를 작성한다.
         * HTML 문서 형식에 맞게 html 태그부터 제대로 작성해야 하지만 out.println()으로
         * 출력되는 내용은 브라우저에 출력되는 내용이 되므로 기본 태그들은 생략했다.
         */
        out.println("<h2>서블릿 작성하기</h2>");
        out.println("Hi Servlet");

        // 작업이 끝나면 스트림을 닫는다.
        out.close();
    }
}
```

### 4.3.2 서블릿 클래스 등록하기

서블릿 클래스가 실행되기 위해서는 JSP 페이지와 달리 등록 과정이 필요하다.

서블릿 클래스는 웹 애플리케이션의 배포 서술자(deployment descriptor) 파일에 등록해야 한다.

배포 서술자 파일이란 웹 애플리케이션 디렉터리 아래 WEB-INF 디렉터리에 위치하는 web.xml 파일을 말하며 이 web.xml 파일은 하나의 웹 애플리케이션에 하나만 만들 수 있다.

웹 서버가 웹 브라우저로부터 URL을 요청 받았을 때 서블릿 클래스를 찾아 호출하기 위해 필요한 정보를 그림 4-7과 같이 배포 서술자인 web.xml에서 <servlet></servlet> 태그와 <servlet-mapping></servlet-mapping> 태그 안에 기술해야 한다. <servlet> 태그는 서블릿으로 사용할 클래스를 지정하는 것으로 서블릿 클래스가 위치한 완전한 클래스 이름(Fully Qualified Name, 패키지 지를 포함한 클래스 이름)을 <servlet-class> 태그에 기술한다. 또한 <servlet-name> 태그에 서블릿의 이름도 지정할 수 있다. 이 이름은 서블릿의 URL 매핑을 설정하는 <servlet-mapping> 태그에서 <servlet> 태그를 참조할 수 있도록 <servlet-name> 태그에 참조할 서블릿 클래스의 이름(별칭)을 지정하는데 사용된다. 그림 4-7은 웹 애플리케이션의 배포 서술자(Deployment Descriptor)인 web.xml에 서블릿을 등록하는 예이다.

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  id="WebApp_ID" version="3.0">
  <servlet>
    <servlet-name>now-servlet</servlet-name>
    <servlet-class>com.jspstudy.ch04.servletbasic.NowServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>now-servlet</servlet-name>
    <url-pattern>/now.do</url-pattern>
  </servlet-mapping>
</web-app>

```

그림 4-7 웹 애플리케이션의 배포 서술자(web.xml)

아래 그림 4-8은 이클립스(좌측)와 톰캣 서버(우측)에서의 web.xml 파일의 위치이다. 또한 서블릿 3.0부터는 web.xml 파일 외에 애노테이션(Annotation)을 이용한 URL 맵핑 기술을 지원 한다. 애노테이션을 이용한 서블릿 등록과 URL 매핑에 대해서는 다음 예제에서 다루어 보자.

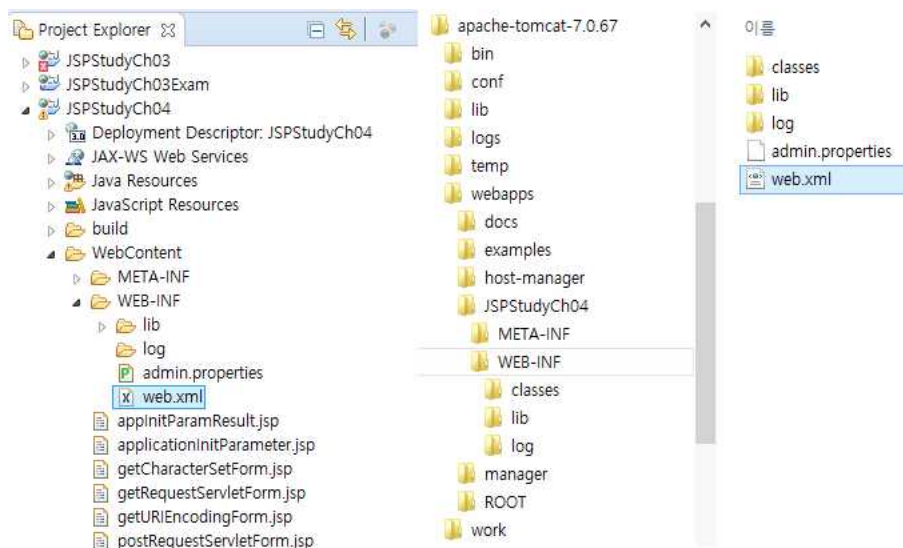


그림 4-8 이클립스와 톰캣 서버의 web.xml

## ▶ 애노테이션을 이용한 서블릿 등록하기

애노테이션은 자바 5.0부터 지원하는 기술로 서블릿 3.0부터 web.xml에 설정해야 하는 내용들을 web.xml에 설정하지 않고 소스 코드에서 애노테이션을 사용해 설정할 수 있도록 지원하고 있다.

앞에서 작성한 NowServlet 클래스를 복사하여 AnnotationNowServlet으로 이름을 지정하고 다음과 같이 클래스 선언부 위에 @WebServlet 애노테이션을 이용해 서블릿 매핑을 설정하고 설정한 서블릿 매핑에 맞게 실행되는지 테스트 해보자.

#### - com.jspstudy.ch04.servletbasic.AnnotationNowServlet

```
/* 서블릿 3.0부터는 web.xml 파일 외에 애노테이션(Annotation) URL 매핑 기술을
 * 지원 한다. 애노테이션은 자바 5.0부터 지원하는 기술로 서블릿 3.0부터 web.xml에
 * 설정해야 하는 내용들을 web.xml에 설정하지 않고 소스 코드에서 애노테이션을 사용해
 * 설정할 수 있다.
 * 서블릿 클래스에 @WebServlet("/now") 또는 @WebServlet(urlPatterns="/now")과
 * 같이 애노테이션을 지정하면 "http://URL/웹애플리케이션이름/now"로 접속되는
 * 요청을 AnnotationNowServlet이 실행될 수 있도록 매핑해 준다.
 */
@WebServlet("/now")
public class AnnotationNowServlet extends HttpServlet {

    // GET 방식의 요청을 처리하기 위해서 doGet() 메서드를 오버라이딩 한다.
    @Override
    public void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {

        /* 웹 브라우저에 출력될 응답문서의 형식과 문자 셋을 지정
         * ContentType 설정은 Response 객체로 부터 PrintWriter
         * 객체를 얻기 전에 지정해야 한글이 깨지지 않고 제대로 동작한다.
         */
        response.setContentType("text/html; charset=UTF-8");

        // 요청에 대한 처리 결과를 웹 브라우저에 출력할 스트림객체 얻기
        PrintWriter out = response.getWriter();
        Calendar cal = Calendar.getInstance();

        /* HTML 문서 형식에 맞게 웹 브라우저로 전송할 데이터를 작성한다.
         * HTML 문서 형식에 맞게 html 태그부터 제대로 작성해야 하지만 out.println()으로
         * 출력되는 내용은 브라우저에 출력되는 내용이 되므로 기본 태그들은 생략했다.
         */
        out.println("<h2>서블릿 애노테이션 등록하기</h2>");
        out.println("Hello Servlet");

        // 작업이 끝나면 스트림을 닫는다.
        out.close();
    }
}
```

이 예제는 웹 애플리케이션 이름이 "JSPStudyCh04"이므로 로컬 네트워크에서 접속한다면 브라우저

주소창에 “http://localhost:8080/JSPStudyCh04/now”를 입력하면 실행된다.

위의 예제는 클라이언트의 요청을 서블릿에서 처리하고 응답 데이터를 저장할 수 있는 Response 객체를 통해서 클라이언트로 출력해 주는 스트림 객체인 PrintWriter 객체를 구해서 서블릿 클래스에서 직접 웹 브라우저로 전송할 데이터를 HTML 문서 형식에 맞게 작성하였다. 응답 결과가 단순하기 때문에 서블릿 클래스에서 HTML 문서 형식으로 작성하는 것이 그렇게 복잡하지 않다고 생각할 수 있겠지만 여러 가지 문서 객체를 포함하는 복잡한 HTML 문서를 서블릿 클래스에서 직접 작성하는 것은 JSP 페이지에서 작성하는 것에 비해서 매우 비효율적이라 할 수 있다. 그러므로 서블릿 클래스에서 요청을 처리하고 그 결과인 응답 데이터를 JSP 페이지로 보내서 웹 브라우저 화면에 보여 지는 부분을 JSP에서 처리하면 서블릿 클래스는 위에서 보다 훨씬 간결해 질 것이다.

#### **[연습문제 4-1] 이름과 나이, 주소를 출력하는 서블릿 작성하기**

이클립스에서 JSPStudyCh04Exam이라는 DynamicWebProject를 생성하고 자신의 이름과 나이, 주소를 출력하는 서블릿을 작성하고 테스트 해 보자.

## 4.4 Annotation을 이용한 URL 패턴 매핑

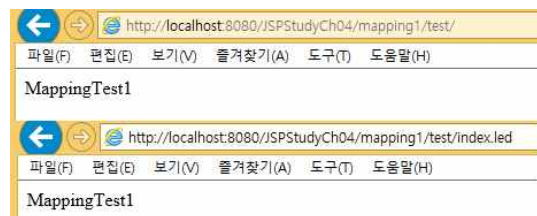
아래 4개의 서블릿 클래스를 가지고 Annotation 방식의 URL 패턴 매핑에 대해서 알아보자.

### - com.jspstudy.ch04.urlmapping.MappingTest1

```
/* JSPStudyCh04/mapping1/test/ 아래로 들어오는
 * 모든 요청을 이 서블릿이 처리하겠다고 선언하는 애노테이션
 * */
@WebServlet(urlPatterns = "/mapping1/test/*")
public class MappingTest1 extends HttpServlet {

    protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html; charset=utf-8");

        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println(" <head><title>MappingTest1</title></head>");
        out.println(" <body>");
        out.println("    MappingTest1");
        out.println(" </body>");
        out.println("</html>");
        out.close();
    }
}
```



### - com.jspstudy.ch04.urlmapping.MappingTest2

```
/* JSPStudyCh04/mapping2/ 아래로 들어오는
 * 모든 요청을 이 서블릿이 처리하겠다고 선언하는 애노테이션
 * */
@WebServlet("/mapping2/*")
public class MappingTest2 extends HttpServlet {

    protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html; charset=utf-8");

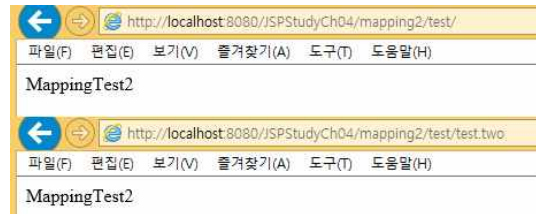
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println(" <head><title>MappingTest2</title></head>");
        out.println(" <body>");
        out.println("    MappingTest2");
        out.println(" </body>");
    }
}
```

```

        out.println("/html>");

        out.close();
    }
}

```



#### - com.jspstudy.ch04.urlmapping.MappingTest3

```

/* JSPStudyCh04/mapping과 JSPStudyCh04/map으로
 * 들어오는 요청을 이 서블릿이 처리하겠다고 선언하는 애노테이션
 * 문자열 배열 형태로 여러 개의 url-pattern을 지정할 수 있다.
 * */
@WebServlet(urlPatterns = { "/mapping", "/map" })
public class MappingTest3 extends HttpServlet {

```

```

    protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html; charset=utf-8");

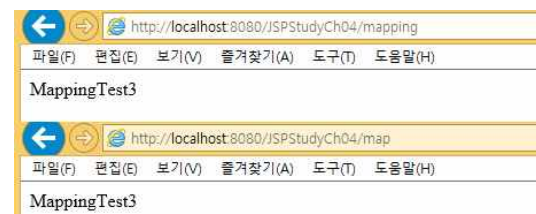
```

```

        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println(" <head><title>MappingTest3</title></head>");
        out.println(" <body>");
        out.println("    MappingTest3");
        out.println(" </body>");
        out.println("/html>");

        out.close();
    }
}

```



#### - com.jspstudy.ch04.urlmapping.MappingTest4

```

/* ContextRoot(JSPStudyCh04/) 아래의 *.led, *.do로
 * 들어오는 모든 요청을 이 서블릿이 처리하겠다고 선언하는 애노테이션
 * 문자열 배열 형태로 여러 개의 url-pattern을 지정할 수 있다.
 * */
@WebServlet({ "*.led", "*.do" })

```

```

public class MappingTest4
    extends HttpServlet {
    protected void doGet(
        HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {

```

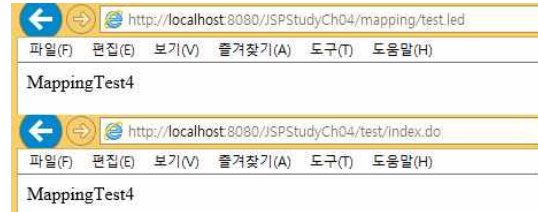
```

response.setContentType("text/html; charset=utf-8");

PrintWriter out = response.getWriter();
out.println("<html>");
out.println(" <head><title>MappingTest4</title></head>");
out.println(" <body>");
out.println("   MappingTest4");
out.println(" </body>");
out.println("</html>");

out.close();
}
}

```



위의 4개 서블릿 클래스에 애노테이션 방식의 URL 패턴 매핑을 설정하였다.  
서블릿을 실행 시킨 후 브라우저 주소창에 위의 그림들과 같이 URL을 바꿔가며 테스트 해보자



## 4.5 요청방식(GET, POST)에 따른 처리

앞에서도 언급했듯이 HttpServlet 클래스를 상속 받아 사용자 정의 서블릿 클래스를 작성 할 때 GET 방식과 POST 방식의 요청을 처리하기 위해 각각 doGet() 메소드와 doPost() 메소드를 재정의 해야 한다. 또한 JSP로 웹 프로그래밍을 하다보면 한글이 깨지는 현상을 빈번히 보게 되는데 이번 절에서 두 가지 요청 방식에 따른 한글 처리 방법도 같이 알아보도록 하겠다.

### 4.5.1 GET방식 요청 처리

GET방식의 요청은 HTTP header를 통해 보내는 요청 URL의 뒷부분에 요청 데이터를 추가하여 보내고 데이터의 크기도 2000자(브라우저 마다 상이함) 정도로 제한된다고 앞에서 언급 하였다. 또한 데이터를 URL뒤에 붙여 보내므로 웹 브라우저의 주소 표시줄에 데이터가 노출되어 보안성도 떨어진다.

아래는 클라이언트가 서버로 GET 방식 요청을 보낼 때 웹 브라우저의 주소 표시줄에 나타나는 요청 URL 이다. 여기서 “?” 뒤에 이어지는 데이터를 쿼리 스트링이라 하며 이 데이터가 서버로 요청을 보낼 때 클라이언트가 보내는 요청 데이터이다. sm=tab의 형태로 데이터가 보내지는데 sm은 데이터의 이름이며 tab은 이름에 대한 데이터의 값이다. 이 한 쌍의 데이터를 파라미터라고 하는데 sm 부분이 파라미터 이름이고 tab 부분이 파라미터 값이 된다. 전송되는 데이터가 여러 개일 경우 아래와 같이 각각의 파라미터를 “&”로 구분하여 전송하게 된다.

<http://search.naver.com/search.naver?sm=tab&where=nexearch&ie=utf8>

- JSPStudyCh04/src/main/webapp/getRequestServletForm.jsp

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

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

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

```
<title>GET 방식 요청처리</title>
```

```
</head>
```

```
<body>
```

```
<!--
```

form 태그의 method 속성을 생략했기 때문에 default인 get 방식이 적용된다.

브라우저는 현재 문서의 문자 셋을 기준으로 폼 데이터를 인코딩하여 서버로 전송한다.

```
-->
```

```
<form name="f1" action="getRequest" >
```

```
    첫 번째 숫자 : <input type="number" name="num1" min="1" /><br/>
```

```
    두 번째 숫자 : <input type="number" name="num2" min="1" /><br/>
```

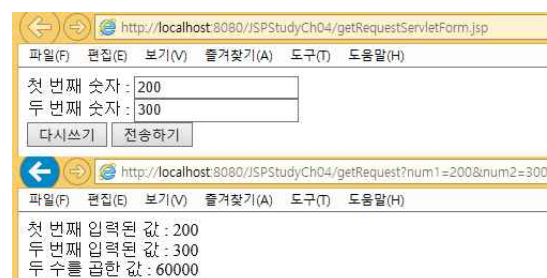
```
    <input type="reset" value="다시쓰기" />
```

```
    <input type="submit" value="전송하기" />
```

```
</form>
```

```
</body>
```

```
</html>
```



- com.jspstudy.ch04.requestmethod.GetRequestServlet

// GET 방식 요청 처리하기

@WebServlet("/getRequest")

**public class** GetRequestServlet **extends** HttpServlet {

// GET 방식의 요청을 처리하기 위해서 doGet() 메서드를 오버라이딩 한다.

@Override

**public void** doGet(HttpServletRequest request,

    HttpServletResponse response)

**throws** ServletException, IOException {

**int** firstNum = 0;

**int** secondNum = 0;

    /\* 웹 브라우저가 GET 방식으로 요청한 파라미터 읽기

    \*

    \* GET 방식의 요청은 웹 브라우저가 서버로 요청을 보낼 때 HTML 문서의 폼에

    \* 입력된 데이터를 요청 URL의 뒷부분에 추가하여 서버로 요청을 보낸다.

    \* 클라이언트가 서버로 요청할 때 서버로 보내는 데이터를 요청 파라미터라고 한다.

    \*

    \* 요청에 대한 정보를 저장하고 있는 HttpServletRequest 객체의

    \* getParameter() 메서드를 이용해 아래와 같이 읽어올 요청 파라미터의

    \* 이름을 지정하면 클라이언트로부터 전송된 요청 파라미터를 읽을 수 있다.

    \*\*/

    String num1 = request.getParameter("num1");

    String num2 = request.getParameter("num2");

    // 웹 브라우저에 출력될 응답문서의 형식과 문자 셋을 지정

    response.setContentType("text/html; charset=utf-8");

    PrintWriter out = response.getWriter();

    // 숫자 형태의 문자열이 아니면 NumberFormatException 발생

    firstNum = Integer.parseInt(num1);

    secondNum = Integer.parseInt(num2);

    // 웹 브라우저에 응답할 내용 작성

    out.println("<h2>GET 방식 요청 처리</h2>");

    out.println("첫 번째 입력 값 : " + firstNum + "<br/>");

    out.println("두 번째 입력 값 : " + secondNum + "<br/>");

    out.println("두 수를 곱한 값 : " + firstNum \* secondNum + "<br/>");

    // 작업이 완료되면 스트림을 닫는다.

    out.close();

}

}

## 4.5.2 POST방식 요청 처리

POST방식의 요청은 HTTP body에 데이터를 실어 보내는 방식으로 데이터 크기의 제한이 없고 사용자의 눈에 보이지 않아 GET방식 보다 보안성이 뛰어나다. GET방식에서는 데이터를 URL 뒤에 추가 하여 전송하였으나 POST 방식은 HTTP 요청 본문에 데이터를 포함하여 전송한다. 즉 HTTP 요청 본문에 첨부파일 형태로 데이터를 추가하여 전송하기 때문에 데이터 크기에 제한이 없다.

### - JSPStudyCh04/src/main/webapp/postRequestServletForm.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>POST 방식 요청처리</title>
</head>
<body>
<!--
    form 태그의 method 속성을 post로 지정 하였다. default는 get 이다.
    브라우저는 현재 문서의 문자 셋을 기준으로 폼 데이터를 인코딩하여 서버로 전송한다.
-->
<form name="f1" action="postRequest" method="post">
    이름 : <input type="text" name="name" /><br/>
    주소 : <input type="text" name="address" /><br/>
    <input type="submit" value="주소입력" />
</form>
<!--
    a 태그를 이용한 요청은 GET 방식이므로 요청 데이터를
    서버로 보낼 때는 요청 URL 뒤에 데이터를 추가해 서버로 보낸다.
-->
<a href="postRequest?name=홍길동&address=서울 관악구 봉천동">링크로 요청하기</a>
</body>
</html>
```

### - com.jspstudy.ch04.requestmethod.PostRequestServlet

```
// POST 방식과 GET 방식 요청 모두 처리하기
```

```
public class PostRequestServlet extends HttpServlet {
```

```
// POST 방식 요청을 처리하기 위해서 doPost() 메서드를 오버라이딩 한다.
```

```
@Override
```

```
public void doPost(HttpServletRequest request,
```

```

    HttpServletResponse response)
        throws ServletException, IOException {

    /* POST 방식 요청인 경우 setCharacterEncoding("UTF-8")을 호출해
    * 웹 브라우저의 인코딩 방식과 동일한 문자 셋인 UTF-8을 지정해야
    * 한글 데이터와 같은 유니코드 문자를 깨지지 않게 처리할 수 있다.
    *
    * getParameter() 메서드가 호출되기 전에 setCharacterEncoding("UTF-8")
    * 메서드를 먼저 호출해 request 영역의 문자 셋을 먼저 처리해야 한다.
    */
    request.setCharacterEncoding("utf-8");

    /* 웹 브라우저가 POST 방식으로 요청한 파라미터 읽기
    *
    * POST 방식의 요청은 웹 브라우저가 서버로 요청을 보낼 때 HTML 문서의 폼에
    * 입력된 데이터를 HTTP 요청 본문(Body)에 추가하여 서버로 요청을 보낸다.
    * 클라이언트가 서버로 요청할 때 서버로 보내는 데이터를 요청 파라미터라고 한다.
    *
    * POST 방식의 요청도 요청에 대한 정보를 저장하고 있는 HttpServletRequest
    * 객체의 getParameter() 메서드를 이용해 아래와 같이 읽어올 요청 파라미터의
    * 이름을 지정하면 클라이언트로부터 전송된 요청 파라미터를 읽을 수 있다.
    */
    String name = request.getParameter("name");
    String address = request.getParameter("address");

    // 웹 브라우저에 출력될 응답문서의 형식과 문자 셋을 지정
    response.setContentType("text/html; charset=utf-8");
    PrintWriter out = response.getWriter();

    out.println("<h2>POST 방식 요청 처리</h2>");
    out.println("이름 : " + name + "<br>");
    out.println("주소 : " + address + "<br>");

    // 작업이 끝나면 스트림을 닫는다.
    out.close();
}

```



```

// GET 방식의 요청 처리하기 위해서 doGet() 메서드를 오버라이딩 한다.
@Override
public void doGet(HttpServletRequest request,

```

```

        HttpServletResponse response)
            throws ServletException, IOException {

    /* 웹 브라우저가 GET 방식으로 요청한 파라미터 읽기
    *
    * GET 방식의 요청은 웹 브라우저가 서버로 요청을 보낼 때 HTML 문서의 폼에
    * 입력된 데이터를 요청 URL의 뒷부분에 추가하여 서버로 요청을 보낸다.
    * 클라이언트가 서버로 요청할 때 서버로 보내는 데이터를 요청 파라미터라고 한다.
    *
    * 요청에 대한 정보를 저장하고 있는 HttpServletRequest 객체의
    * getParameter() 메서드를 이용해 아래와 같이 읽어올 요청 파라미터의
    * 이름을 지정하면 클라이언트로부터 전송된 요청 파라미터를 읽을 수 있다.
    */
    String name = request.getParameter("name");
    String address = request.getParameter("address");

    // 웹 브라우저에 출력될 응답문서의 형식과 문자 셋을 지정
    response.setContentType("text/html; charset=utf-8");
    PrintWriter out = response.getWriter();

    out.println("<h2>GET 방식 요청 처리</h2>");
    out.println("이름 : " + name + "<br>");
    out.println("주소 : " + address + "<br>");

    // 작업이 끝나면 스트림을 닫는다.
    out.close();
}
}

```

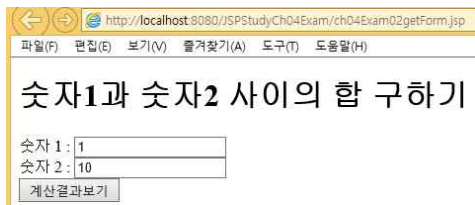


위의 예제를 살펴보면 서블릿 클래스에서 요청을 처리하고 화면에 출력하는 View 페이지도 서블릿 클래스가 담당하다 보니 간단한 예제이지만 서블릿에서 뷰를 직접 만드는 부분이 조금은 복잡해 보인다. 서블릿에서는 요청을 받아 처리하고 그 결과를 JSP로 이동해서 화면에 보여 지는 부분인 View 페이지는 JSP가 담당하도록 구현하는 방법에 대해 알아보고 위의 예제를 다시 작성해 보자.

#### [연습문제 4-2] GET과 POST 방식 모두를 처리하는 서블릿 만들기

[연습문제 4-1]에서 생성한 프로젝트에 서블릿 클래스 파일을 만들고 이 서블릿 클래스에서 아래 그림과 같이 GET방식과 POST방식 요청 모두를 처리하는 서블릿 클래스를 작성해 테스트 해

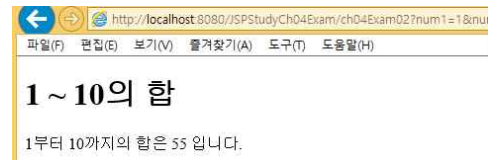
보자.



숫자1과 숫자2 사이의 합 구하기

숫자 1 : 1  
숫자 2 : 10

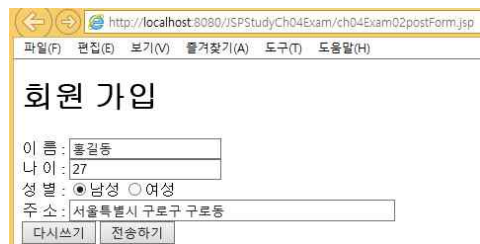
계산결과보기



1 ~ 10의 합

1부터 10까지의 합은 55 입니다.

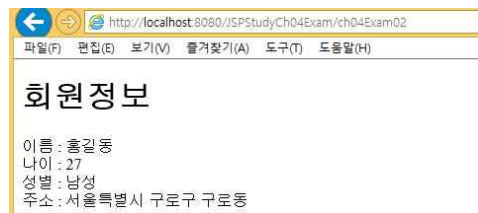
## GET 방식 처리



회원 가입

이름 : 홍길동  
나이 : 27  
성별 : ☒ 남성 ☐ 여성  
주소 : 서울특별시 구로구 구로동

다시쓰기 전송하기



회원정보

이름 : 홍길동  
나이 : 27  
성별 : 남성  
주소 : 서울특별시 구로구 구로동

## POST 방식처리

## 4.6 폼 데이터 처리하기

HTML의 FORM 태그 안에서 한 줄로 간단히 텍스트를 입력할 수 있는 한 줄 텍스트 입력 상자와 여러 줄의 텍스트를 입력할 수 있는 여러 줄 텍스트 입력 상자(textarea)가 있으며 이외에도 라디오 버튼, 체크 박스, 목록 상자(Select Box)등 다양한 형식으로 폼 컨트롤을 구성해 사용자로 부터 데이터를 입력받을 수 있도록 구성할 수 있다.

한 줄 텍스트 입력 상자(비밀번호 입력 상자 포함)와 여러 줄 텍스트 입력 상자에 데이터를 입력하지 않고 폼을 전송하게 되면 요청 파라미터에 공백 문자열이 매핑 되어 서버로 전달된다. 하지만 라디오 버튼과 체크 박스와 같은 선택적인 폼 컨트롤은 사용자가 선택하지 않으면 해당 요청 파라미터는 아예 서버로 전달되지 않기 때문에 `getParameter()` 메서드를 이용해 해당 요청 파라미터를 읽으면 null 값을 반환 받는다.

### 4.6.1 라디오 버튼

라디오 버튼은 사용자가 관련 있는 여러 가지 항목 중에서 오로지 하나만 선택하도록 해야 할 경우 사용할 수 있는 폼 컨트롤이다. 라디오 버튼으로 관련 있는 여러 항목을 표현할 때 `name` 속성의 값을 동일하게 지정해야 관련 있는 항목 중에서 오로지 하나만 선택할 수 있도록 구현할 수 있다.

```
<input type="radio" name="gender"
      value="male"/>
<input type="radio" name="gender"
      value="female"/>
```

```
request.getParameter("gender");
value 또는 null 값을 얻음
```

### 4.6.2 체크박스

체크박스는 사용자가 관련 있는 여러 가지 항목 중에서 한 개 이상을 선택하도록 해야 할 경우 사용할 수 있는 폼 컨트롤이다. 체크박스로 관련 있는 여러 가지 항목을 표현할 때 `name` 속성의 값은 동일하게 지정하거나 각각 다르게 지정할 수 있다. 각각 다르게 지정하게 되면 `getParameter()` 메서드를 사용해 파라미터를 문자열로 읽을 수 있고 동일하게 지정할 경우 `getParameterValues()` 메서드를 사용해 파라미터를 문자열 배열로 한 번에 읽을 수도 있다.

`getParameterValues()` 메서드를 사용해 파라미터를 읽을 경우 체크박스의 `value` 속성의 값을 지정하는 것이 좋다. 왜냐하면 라디오 버튼이나 체크박스에 `value` 속성의 값을 지정하지 않으면 서버에서 요청 파라미터를 받을 때 해당하는 라디오 버튼이나 체크박스가 선택되지 않았다면 null 값을 받고 선택되었다면 문자열 `on` 값을 받기 때문이다.

```
<input type="checkbox" name="nMail"/>
<input type="checkbox" name="aMail"/>
<input type="checkbox" name="iMail"/>
```

```
request.getParameter("nMail");
request.getParameter("aMail");
request.getParameter("iMail");
on 또는 null 값을 얻음
```

<pre> &lt;input type="checkbox" name="nMail"       value="notice"/&gt; &lt;input type="checkbox" name="aMail"       value="advert"/&gt; &lt;input type="checkbox" name="iMail"       value="info"/&gt; </pre>	<pre> request.getParameter("nMail"); request.getParameter("aMail"); request.getParameter("iMail"); request.getParameterValues("nMail"); <b>value 또는 null 값을 얻음</b> </pre>
---	---

### 4.6.3 목록 상자 처리하기

목록 상자는 콤보박스 또는 셀렉트 박스라고도 하며 관련 있는 여러 가지 항목 중에서 오로지 하나만 선택할 수 있도록 하거나 여러 가지를 선택할 수 있도록 할 때 사용할 수 있는 폼 컨트롤이다. 목록 상자는 아래와 같이 <select> 태그 안에 <option> 태그를 사용해 목록 상자의 항목을 추가할 수 있다. <select> 태그에 size 속성을 이용해 화면에 보여 지는 항목의 개수를 지정할 수 있고 multiple 속성을 지정해 사용자가 여러 개의 항목을 선택할 수 있도록 할 수도 있다. multiple 속성은 속성의 값이 없기 때문에 html 방식인 multiple만 기술할 수도 있고 xml 방식인 multiple="multiple"로 지정할 수도 있다.

<pre> &lt;select name="job"&gt;   &lt;option&gt;회사원&lt;/option&gt;   &lt;option&gt;학생&lt;/option&gt;   &lt;option&gt;기타&lt;/option&gt; &lt;/select&gt; </pre>	<pre> request.getParameter("job"); <b>선택된 항목의 텍스트 값을 얻음</b> </pre>
<pre> &lt;select name="job"&gt;   &lt;option value="worker"&gt;회사원&lt;/option&gt;   &lt;option value="student"&gt;학생&lt;/option&gt;   &lt;option value="etc"&gt;기타&lt;/option&gt; &lt;/select&gt; </pre>	<pre> request.getParameter("job"); <b>선택된 항목의 value 값을 얻음</b> </pre>
<pre> &lt;select name="job" multiple size="4"&gt;   &lt;option&gt;회사원&lt;/option&gt;   &lt;option&gt;학생&lt;/option&gt;   &lt;option&gt;기타&lt;/option&gt; &lt;/select&gt; </pre>	<pre> request.getParameterValues("job"); <b>선택된 항목의 텍스트 또는 value 값을 String 배열로 얻음</b> </pre>

<option> 태그에 value 속성을 지정하지 않으면 <option> 태그로 감싼 문자열이 서버로 요청을 보낼 때 파라미터의 값으로 사용되며 명시적으로 value 속성에 값을 지정하면 value 속성에 지정한 값이 파라미터의 값으로 사용된다.



## ▶ 회원가입 폼으로부터 요청된 데이터 처리하기

- JSPStudyCh04/src/main/webapp/formData01.jsp

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

<!DOCTYPE html>

<html>
<head>
<meta charset="UTF-8">
<title>회원 기본 정보 입력 폼</title>
</head>
<body>
```

← → http://localhost:8080/SPStudyCh04/jspFormData01.jsp

파일(F) 편집(E) 보기(V) 즐겨찾기(A) 도구(T) 도움말(H)

## 회원 기본 정보 입력

이름:

아이디:

비밀번호:

성별: ☒ 남 ☐ 여

메일수신: ☒ 공지메일 받음 ☐ 광고메일 받음 ☒ 정보메일 받음

직업:

## 회원 기본 정보 입력

<!-- form 태그의 method 속성을 지정하지 않으면 GET 방식 요청이 된다. -->

```
<form name="fMember1" action="formData01">
```

[illegible]

<input type="text" name="name" /></p>

$\langle p \rangle_0 | \text{ } \& \text{nbsp; } \& \text{nbsp; } 0 | \text{ } \& \text{nbsp; } \& \text{nbsp; } \square :$

```
<input type="text" name="id" /></p>
```

<p>비밀번호 : <input type="password" name="pass" /></p>

<!-- 라디오 버튼은 name 속성의 값을 동일하게 지정해야 그룹으로 묶인다. -->

[illegible]☐남    ☐여

<!--

체크박스는 name 속성의 값을 각각 지정할 수도 있고 동일하게 지정할 수도 있다.

→

<p>메일수신 :

```
<input type="checkbox" name="nMail"/>
```

공지메일 받음

```
<input type="checkbox" name="aMail"/>
```

광고메일 받음

```
<input type="checkbox" name="iMail"/>
```

정보메일 받음

<!--

<option> 태그에 value 속성을 지정하지 않으면

<option> 태그로 감싸 문자열이 value가 된다.

-->

[illegible]

```
<select name="job">
```

<option>회사원</option>

<option>학생</option>

<option>주부</option>

<option>기타</option>

```

        </select></p>
<!--
    submit 버튼이 클릭되면 form 태그의 action 속성에서
    지정한 URL로 무조건 폼 데이터를 전송한다.
-->
<p><input type="reset" value="다시쓰기" />
<input type="submit" value="가입하기" /></p>
</form>
</body>
</html>

```

#### - com.jspstudy.ch04.formdata.FormData01

// 회원등록 폼 데이터를 처리하는 서블릿

@WebServlet("/formData01")

**public class** FormData01 **extends** HttpServlet {

@Override

**public void** doGet(HttpServletRequest request,  
 HttpServletResponse response)  
**throws** ServletException, IOException {

// 파라미터로 전송된 데이터를 저장할 변수 선언

String name, id, pass, gender, nMail, aMail, iMail, job;

/\* GET 방식으로 전달되는 파라미터의 값을 읽어온다.

- \* 폼 컨트롤 중에서 한 줄 텍스트 입력 상자, 비밀번호 입력 상자,
  - \* 여러 줄 입력 상자에 입력된 값이 없으면 요청 파라미터는 존재하지만
  - \* 데이터가 입력되지 않은 상태이므로 공백 문자열("")을 받게 된다.
  - \* 하지만 라디오 버튼과 체크 박스는 사용자가 선택하지 않으면 해당
  - \* 요청 파라미터는 아예 서버로 전달되지 않기 때문에 null 값을 받게 된다.
- \*/

name = request.getParameter("name");

id = request.getParameter("id");

pass = request.getParameter("pass");

gender = request.getParameter("gender");

nMail = request.getParameter("nMail");

aMail = request.getParameter("aMail");

iMail = request.getParameter("iMail");

job = request.getParameter("job");

// view 페이지에 출력할 데이터를 request의 속성에 담고 있다.

request.setAttribute("name", name);

request.setAttribute("id", id);

request.setAttribute("pass", pass);

```

request.setAttribute("gender", gender);
request.setAttribute("nMail", receiveMail(nMail));
request.setAttribute("aMail", receiveMail(aMail));
request.setAttribute("iMail", receiveMail(iMail));
request.setAttribute("job", job);

/* view 페이지로 제어를 넘겨 요청에 대한 결과를 출력하기 위해
 * HttpServletRequest 객체로 부터 RequestDispatcher 객체를
 * 구하여 view/formDataView01.jsp로 포워딩을 하고 있다.
 */
RequestDispatcher rd =
    request.getRequestDispatcher("view/formDataView01.jsp");
rd.forward(request, response);

/* 모델 데이터를 View 페이지로 전달해 View 페이지에서 HTML 문서 형식으로
 * 응답 데이터를 만들기 때문에 아래와 같이 번거롭게 뷰를 직접 만들 필요는 없다.
 */
/*
// 웹 브라우저에 출력될 응답문서의 형식과 문자 셋을 지정
response.setContentType("text/html; charset=UTF-8");
PrintWriter out = response.getWriter();

out.println("<html>");
out.println("<head>");
out.println(" <title>회원 기본 정보</title>");
out.println("</head>");
out.println("<body>");
out.println(" <h2>회원 기본 정보</h2>");
out.printf("    이 름 : %s<br/>", (name.equals("")) ? "null" : name);
out.printf("    아이디 : %s<br/>", id);
out.printf("    비밀번호 : %s<br/>", pass);
out.printf("    성 별 : %s<br/>", gender);
out.printf("    공지메일 : %s<br/>", receiveMail(nMail));
out.printf("    광고메일 : %s<br/>", receiveMail(aMail));
out.printf("    정보메일 : %s<br/>", receiveMail(iMail));
out.printf("    직 업 : %s<br/>", job);
out.println("</body>");
out.println("<html>");

// 작업이 끝나면 스트림을 닫는다.
out.close();*/
}

// 체크박스의 선택 여부에 따라 문자열을 지정하는 메소드
private String receiveMail(String mail) {

```

```

        if(mail == null) {
            return "받지 않음";
        } else {
            return "받음";
        }
    }
}

```

- JSPStudyCh04/src/main/webapp/view/formDataView01.jsp

```

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

```

```

<!DOCTYPE html>

```

```

<html>

```

```

<head>

```

```

<meta charset="UTF-8">

```

```

<title>회원 기본 정보</title>

```

```

</head>

```

```

<body>

```

```

    <h2>회원 기본 정보</h2>

```

```

    <!-- 서블릿 3.0부터 EL에서 자바 클래스의 메서드를 사용할 수 있다. -->

```

```

    이 름 : ${ name.equals("") ? "null" : name }<br/>

```

```

    아이디 : ${ id }<br/>

```

```

    비밀번호 : ${ pass }<br/>

```

```

    성 별 : ${ gender }<br/>

```

```

    공지메일 : ${ nMail }<br/>

```

```

    광고메일 : ${ aMail }<br/>

```

```

    정보메일 : ${ iMail }<br/>

```

```

    직 업 : ${ job }

```

```

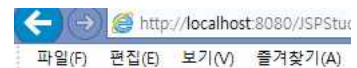
</body>

```

```

</html>

```



## 회원 기본 정보

이 름 : 홍길동  
 아이디 : midas  
 비밀번호 : 1111  
 성 별 : male  
 공지메일 : 받음  
 광고메일 : 받지 않음  
 정보메일 : 받음  
 직 업 : 회사원

### [연습문제 4-3] 학생등록 폼으로부터 요청된 데이터 처리하기

앞에서 우리는 회원등록 폼으로부터 요청된 데이터를 처리하는 방법에 대해서 알아보았다.

이 예제를 참고해 아래 그림과 같이 학생등록 폼으로부터 요청된 데이터를 처리하여 화면에 출력하는 애플리케이션을 작성해 보자.

[연습문제 4-2]와 같은 프로젝트에 예제를 작성하고 파일 정보는 아래와 같다.

- 학생 등록 폼페이지 : webapp/jspFormData02.jsp
- 학생 등록 요청을 처리하는 서블릿 : com.jspstudy.ch04.exam.FormData02
- 결과를 출력하는 View 페이지 : webapp/view/formDataView02.jsp

http://localhost:8080/JSPStudyCh04/jspFormData02.jsp

파일(F) 편집(E) 보기(V) 즐겨찾기(A) 도구(T) 도움말(H)

### 학생 등록 정보

학 생 명 :

성 별 : ☒ 남 ☐ 여

연 락 처 :

희망 취업 분야 :

☒ SI 업체 ☒ SM 업체 ☒ 솔루션 업체

관심분야 :

http://localhost:8080/JSPStudyCh04/formDataHandling02

파일(F) 편집(E) 보기(V) 즐겨찾기(A) 도구(T) 도움말(H)

### 학생 등록 정보

이 름 : 홍길동  
 성 별 : male  
 연락처 : 01034528541  
 희망 취업 분야 : si, sm, solution  
 관심분야 : Spring, jQuery

## 4.7 서블릿 라이프 사이클

우리는 앞에서 서블릿을 작성하는 방법과 서블릿이 동작하는 원리에 대해서 알아보았다.

클라이언트의 요청이 들어오면 그 요청이 첫 요청일 경우 톰캣 서버는 그 요청을 처리하는 서블릿 클래스를 검색하여 메모리에 읽어 들이고 그 클래스의 인스턴스를 생성해 서블릿 컨테이너에 담아두고 클라이언트의 요청을 처리한다. 이렇게 서블릿 컨테이너에 담겨 클라이언트의 요청을 처리할 수 있는 서블릿 클래스의 객체를 서블릿이라고 한다. 이후에는 서블릿 클래스가 변경되기 전까지 동일한 요청에 대해서 새롭게 서블릿 클래스의 인스턴스를 생성하지 않고 서블릿 컨테이너에 담아둔 서블릿을 이용해 클라이언트의 요청을 처리한다. 하지만 톰캣 서버가 새롭게 시작되거나 서블릿 클래스가 변경되면 톰캣 서버는 필요한 서블릿 클래스를 검색하여 메모리에 읽어 들이고 그 클래스의 인스턴스를 생성해 서블릿 컨테이너에 담아두고 클라이언트에 요청을 처리하는 과정을 다시 반복한다.

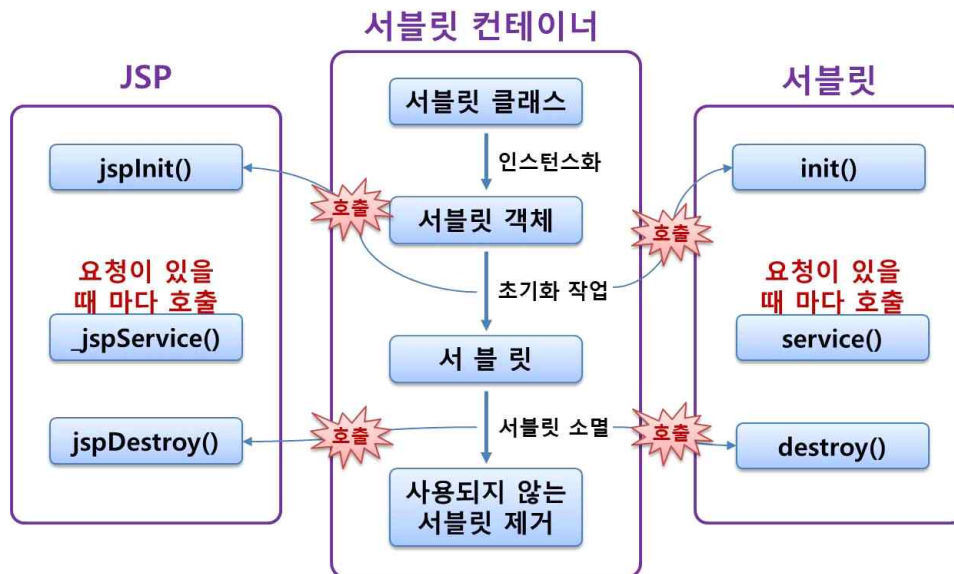


그림 4-10 서블릿과 JSP의 라이프 사이클

톰캣 서버는 어떤 요청이 첫 요청으로 들어올 때 서블릿 클래스의 인스턴스를 생성하고 최초 한 번만 그 클래스의 `init()` 메서드를 호출해 초기화 작업을 진행한다. 그리고 `service()` 메서드를 호출해 GET 방식 요청일 경우 `doGet()` 메서드가 POST 방식의 요청일 경우에 `doPost()` 메서드를 통해 응답을 처리한다. 그리고 첫 요청 이후에 동일한 요청이 들어올 경우 톰캣 서버는 서블릿 클래스의 인스턴스를 다시 생성하는 것이 아니라 서블릿 컨테이너에 등록한 서블릿으로 요청을 처리하기 때문에 요청이 들어올 때 마다 `service()` 메서드를 호출하고 그 안에서 `doGet()` 메서드와 `doPost()` 메서드가 요청을 처리하도록 하고 있다. 이렇게 서비스 하다가 더 이상 서블릿을 사용하지 않거나 톰캣을 재시작 할 경우에 마지막으로 한 번 서블릿의 `destroy()` 메서드를 호출해 소멸화 작업을 진행한다.

### ▶ 서블릿 라이프 사이클 - `init()` 메서드와 `destroy()` 메서드

- JSPStudyCh04/src/main/webapp/servletLifeCycle.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <h2>서블릿 라이프 사이클</h2>
    <form action="servletLifeCycle" method="post">
        <input type="submit" value="POST 요청하기" />
    </form><br/>
    <a href="servletLifeCycle">GET 요청하기</a>
</body>
</html>

```

#### - com.jspstudy.ch04.lifecycle.ServletLifeCycle

```

// 서블릿의 라이프 사이클 - init() 메서드와 destroy() 메서드
/* Servlet 3.0 부터 애노테이션을 사용해 아래와 같이 서블릿을 등록할 수 있고
 * 서블릿 초기화 파라미터와 서블릿 매핑을 지정할 수 있다.
 * urlPatterns에 처리할 패턴이 여러 개일 경우 배열 {}로 지정할 수 있다.
 * 서블릿 초기화 파라미터가 여러 개일 경우에도 initParams에 배열 {}을 지정해
 * @WebInitParam()을 여러 개 지정할 수 있다.
 *
 * web.xml에 servlet-mapping이 설정되어 있다면 애노테이션과 web.xml에
 * 지정한 서블릿 이름이 동일하기 때문에 web.xml에 지정한 url-pattern이 우선한다.
 * 현재 서블릿 클래스의 애노테이션으로 지정한 urlPatterns는 적용되지 않는다.
 * 하지만 서블릿 초기화 파라미터는 파라미터 이름이 서로 다르기 때문에 사용할 수 있다.
 */
@WebServlet(name="servletLifeCycle",
    //urlPatterns={"*.life"},
    initParams=@WebInitParam(
        name="ANNOTATION_PARAM", value="애노테이션 파라미터"))
public class ServletLifeCycle extends HttpServlet {

    /* 서블릿 컨테이너에서 서블릿 객체가 생성되고 초기화된 다음에 init() 메소드가 호출 된다.
     * 이 메소드는 서블릿 객체가 초기화될 때 톰캣에 의해서 딱 한 번 호출되는 메소드 이다.
     * 서블릿 객체가 만들어지고 최초 한 번 실행해야할 작업이 있다면 이 메서드를 이용한다.
     */
    public void init() throws ServletException {
        System.out.println("init() 메소드 호출됨");

        /* web.xml에 서블릿을 등록할 때 servlet 태그 하위에 init-param 태그

```

```

    * 안에 param-name에 지정한 "READ_FILE"이라는 파라미터 이름을 찾아서
    * param-value 태그에 지정한 파라미터 값을 읽어온다.
    **/
String readFile = getInitParameter("READ_FILE");
String testParam = getInitParameter("TEST_PARAM");
String annotationParam = getInitParameter("ANNOTATION_PARAM");

System.out.println("readFile : " + readFile);
System.out.println("testParam : " + testParam);
System.out.println("annotationParam : " + annotationParam);
System.out.println("servletName : " + getServletName());
}

// GET 방식 요청이 들어올 때 마다 실행되는 메서드
@Override
public void doGet(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    System.out.println("*** doGet() 메서드 실행됨 ***");
}

// POST 방식 요청이 들어올 때 마다 실행되는 메서드
@Override
public void doPost(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    System.out.println("### doPost() 메서드 실행됨 ###");
}

/* 서블릿 객체가 소멸되기 직전에 톰캣은 딱 한 번 destroy() 메서드를 호출한다.
 * 서블릿이 소멸될 때 마지막으로 해야 할 작업이 있다면 이 메서드를 이용한다.
 * 주로 열려있는 스트림을 닫거나, 자원을 해제하는 코드 등을 기술한다.
 *
 * 톰캣이 종료되거나 기존 클래스가 수정되면 톰캣은 자신이 관리하고 있는 서블릿을
 * 소멸시키며 서블릿에 구현되어 있는 destroy() 메서드를 마지막으로 호출한다.
 **/
public void destroy() {
    System.out.println("destory() 메소드 호출됨");
}
}

```

- WEB-INF/web.xml에 아래와 같이 서블릿 등록과 서블릿 초기화 파라미터 설정

```

<servlet>
    <servlet-name>servletLifeCycle</servlet-name>

```



```
<servlet-class>com.jspstudy.ch04.lifecycle.ServletLifeCycle</servlet-class>
<init-param>
  <param-name>READ_FILE</param-name>
  <param-value>/WEB-INF/set/setting.txt</param-value>
</init-param>
<!-- 서블릿 파라미터가 여러 개 필요할 경우 init-param을 여러 개 사용할 수 있다. -->
<init-param>
  <param-name>TEST_PARAM</param-name>
  <param-value>테스트 파라미터</param-value>
</init-param>
</servlet>
<servlet-mapping>
  <servlet-name>servletLifeCycle</servlet-name>
  <url-pattern>/servletLifeCycle</url-pattern>
</servlet-mapping>
```

## 5. JSP 내장객체

JSP 내장객체란 JSP 페이지에서 명시적으로 객체를 생성하지 않고도 바로 사용할 수 있는 객체를 말한다. 자바는 객체를 참조하는 참조변수를 선언하고 그 참조변수에 객체를 생성해 참조 값을 할당해야 비로소 참조 변수를 통해서 그 객체에 접근할 수 있는데 JSP에서 사용하는 내장객체처럼 명시적으로 객체를 생성하지 않고 바로 사용할 수 있는 이유는 JSP 페이지가 톰캣에 의해서 서블릿 클래스로 변환될 때 그 변환되는 서블릿 클래스의 `_jspService()` 메서드 안에서 JSP에서 사용하는 내장객체가 자동으로 생성되도록 코드가 만들어지기 때문이다.

### 5.1 request 내장객체

클라이언트에 대한 요청 정보를 저장하고 관리하는 객체가 바로 request 내장객체 이다. request 객체는 `HttpServletRequest` 타입으로 클라이언트에서 서버로 전송되는 요청 정보에 접근할 수 있는 다양한 메서드를 제공하고 있다.

#### ▶ request 내장객체를 이용해 클라이언트와 서버정보 출력하기

- JSPStudyCh05/src/main/webapp/request/requestForm.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>GET 방식과 POST 방식 요청 정보 보기</title>
</head>
<body>
    <%-- 링크는 GET 방식 요청 --%>
    <a href="requestInformation.jsp?num1=100&num2=200">요청정보보기</a><br/><br/>
    <%-- POST 방식 요청 --%>
    <form action="requestInformation.jsp" method="post">
        이름 : <input type="text" name="name" /><br/>
        비밀번호 : <input type="text" name="pass" /><br/>
        <input type="submit" value="전송" />
    </form>
</body>
</html>
```

- JSPStudyCh05/src/main/webapp/request/requestInformaion.jsp

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

```

<meta charset="UTF-8">
<title>request 내장객체</title>
</head>
<body>
  <%--
    클라이언트의 요청 방식이 GET인지 POST인지 문자열로 반환 한다.
  --%>
  <h1><%= request.getMethod() %> 방식 요청 정보</h1>
  <h2>서버 정보</h2>
  <ul>
    <li>서버 이름 : <%= request.getServerName() %></li>
    <li>서버 포트 : <%= request.getServerPort() %></li>
  </ul>

  <h2>요청 정보</h2>
  <ul>
    <li>요청방식 : <%= request.getMethod() %></li>
    <li>프로토콜 : <%= request.getProtocol() %></li>

    <%-- 클라이언트가 요청한 전체 경로를 반환 한다. --%>
    <li>요청URL: <%= request.getRequestURL() %></li>

    <%-- 서버 이름과 포트를 제외한 ContextPath를 포함한 경로를 반환 한다. --%>
    <li>요청URI : <%= request.getRequestURI() %></li>

    <%-- 웹 어플리케이션의 이름을 ContextPath라 한다. --%>
    <li>컨텍스트경로 : <%= request.getContextPath() %></li>
    <li>컨텐츠타입 : <%= request.getContentType() %></li>

    <%--
      요청 본문에 추가되어 서버로 전송된 요청 데이터의 크기를 반환
      파라미터 이름과 파라미터 값을 포함한 전체 데이터의 크기를 의미한다.

      예) name=홍길동&pass=1234
          name=%ED%99%8D%EA%B8%B8%EB%8F%99&pass=1234

      위의 예는 POST 방식으로 요청된 데이터로 데이터가 요청 본문에 추가되어
      서버로 전송되며 콘텐츠의 크기는 42byte 이므로 42를 반환 한다.
      GET 방식의 요청에서는 요청 본문에 데이터가 추가되는 것이 아니라
      URL 끝에 추가되어 파라미터가 전송되므로 -1을 반환 한다.
    --%>
    <li>컨텐츠 길이 : <%= request.getContentLength() %></li>
  </ul>
  <h2>클라이언트 정보</h2>
  <ol>

```

```
<li>클라이언트 이름 : <%= request.getRemoteHost() %></li>
```

```
<!--
```

이클립스의 서버 실행 환경이 IPv6 형태로 설정되어 있어 현재 사용되는 IPv4 형태의 IP 정보를 확인하려면 아래와 같이 설정하고 실행해야 한다.

프로젝트 마우스 오른쪽 -> RunAs -> Run Configurations를 선택하면 Run Configurations 설정 창이 화면에 나타난다.

좌측의 리스트에서 Apache Tomcat ► 를 클릭하여 확장하면 등록되어 있는 Tomcat v7.0 Server 또는 Tomcat v8.0 Server를 선택하면 우측에 나타나는 여러 탭들 중에서 Arguments 탭을 선택한 후 VM arguments 설정 정보 입력란에 기존에 설정된 값들은 변경하지 말고 맨 아래 부분에 아래와 같이 IPv4를 설정하는 명령 한 줄을 추가한다.

```
-Djava.net.preferIPv4Stack="true"
```

```
//-->
```

```
<li>클라이언트 주소 : <%= request.getRemoteAddr() %></li>
```

```
<li>클라이언트 포트 : <%= request.getRemotePort() %></li>
```

```
<li>클라이언트 유저 : <%= request.getRemoteUser() %></li>
```

```
<li>웹 브라우저와 클라이언트의 시스템 정보 보기 :<br/>
```

```
<%= request.getHeader("User-Agent") %></li>
```

```
<li>브라우저 지원 FileType : <%= request.getHeader("Accept") %></li>
```

```
<li>바로이전 URL : <%= request.getHeader("referer") %></li>
```

```
</ol>
```

```
</body>
```

```
</html>
```

## 5.2 response 내장객체

클라이언트의 요청에 대한 응답 처리를 위해 필요한 객체가 바로 response 내장객체이다. response 객체는 요청에 대한 처리 결과를 클라이언트로 보낼 때 사용하는 객체이며 응답 데이터의 헤더 설정이나 리다이렉트 기능을 제공하는 메서드를 제공하고 있다.

### ► response 내장객체의 sendRedirect()를 이용한 로그인 처리

- JSPStudyCh05/src/main/webapp/response/sendRedirectLoginForm.jsp

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

```
pageEncoding="UTF-8"%>
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<title>로그인 화면</title>
```

```
</head>
```

```
<body>
```



```

<form
  action="sendRedirectLoginProcess.jsp"
  method="post">
  아이디: <input type="text" name="id" /><br/>
  비밀번호 : <input type="password" name="pass" /><br/>
  <input type="submit" value="로그인" />
</form>
</body>
</html>

```

#### - JSPStudyCh05/src/main/webapp/response/sendRedirectLoginProcess.jsp

```

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

```

```

<%

```

```

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

```

```

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

```

```

/* 아이디와 비밀번호가 맞으면 response 내장객체의 sendRedirect() 메서드를 이용해
 * 로그인 성공 페이지로 리다이렉트(Redirect) 시킨다. sendRedirect() 메서드는
 * 요청한 자원이 다른 곳으로 이동되었다고 브라우저에게 응답하면서 이동할 URL을
 * 알려주고 그 쪽으로 다시 요청하라고 응답하는 메서드이다. Redirect 기법은 웹
 * 브라우저를 새로 고침(F5) 했을 때 동일한 코드가 재실행되면 문제가 될 수 있는
 * 경우에 클라이언트의 요청을 모두 처리한 후 특정 URL로 이동시키기 위해 주로
 * 사용한다. 예를 들어 게시 글쓰기에 대한 요청을 처리한 후에 Redirect 시키지
 * 않는다면 브라우저의 주소 표시줄에 게시 글쓰기에 대한 URL이 그대로 남아 있기
 * 때문에 사용자가 브라우저를 새로 고침(F5) 하게 되면 바로 이전에 실행된 게시
 * 글쓰기 작업이 반복 실행되어 중복된 데이터가 저장되는 문제가 발행할 수 있다.
 * 이를 방지하기 위해서 게시 글쓰기가 완료되면 게시 글 리스트로 이동시키기
 * 위해서 response 내장객체의 sendRedirect() 메서드를 사용해 게시 글
 * 리스트의 URL을 웹 브라우저에게 응답하고 웹 브라우저는 응답 받은 URL로
 * 다시 요청하도록 하는 것이다. 왜 게시 글 리스트로 리다이렉트 시켜야 하는가?
 * 게시 글 리스트는 DB에서 데이터를 조회하는 쿼리인 SELECT 쿼리를 사용하기
 * 때문에 이 쿼리가 실행되어도 데이터가 중복되어 저장되거나, 동일한 데이터를
 * 반복적으로 수정 또는 삭제하려는 문제가 발생되지 않기 때문이다. 이렇게 게시
 * 글쓰기와 같이 DB에서 데이터의 입력, 수정, 삭제 작업과 연동되는 경우에
 * 사용자의 새로 고침(F5) 등으로 문제가 발생할 수 있기 때문에 Redirect를
 * 사용한다. 이외에 다른 사이트로 이동시킬 때에도 Redirect 기법을 사용 한다.
 */

```

```

if(id.equals("admin") && pass.equals("1234")) {
    response.sendRedirect("sendRedirectLoginOk.jsp?id=" + id);

```

```

} else {

```

```

%>

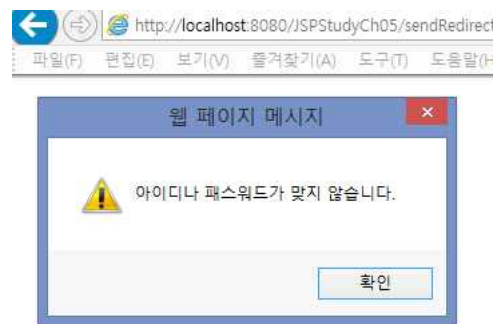
```

```

<%--
아이디 또는 패스워드가 틀리면 로그인 폼페이지로 돌려보내기
위해서 자바스크립트 코드만 응답으로 보내서 자바스크립트가
실행되도록 하여 로그인 폼페이지로 이동시킨다.
--%>
<script type="text/javascript">
    alert("아이디 또는 패스워드가 맞지 않습니다.");
    // 브라우저 주소 창의 주소를 변경해 특정 사이트로 이동 시킨다.
    location.href = "sendRedirectLoginForm.jsp";

    /* 현재 요청된 페이지에서 바로 이전에 있었던 페이지로 이동한다.
    * 현재 페이지의 바로 이전 페이지인 로그인 폼페이지도 이동한다.
    **/
    //history.back();
</script>
<%
}
%>

```



- JSPStudyCh05/src/main/webapp/response/sendRedirectLoginOk.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<title>로그인 성공</title>
</head>
<body>
    <h2> 안녕하세요 <%= request.getParameter("id") %> 님!</h2>
</body>
</html>

```



## 5.3 out 내장객체

out 내장객체는 서버에서 클라이언트로 연결된 출력 스트림으로 클라이언트로 응답할 데이터를 출력하는 역할을 하는 내장객체이다. 서블릿 클래스에서는 PrintWriter 타입의 출력 스트림을 사용해 클라이언트로 응답할 데이터를 만들었지만 JSP 페이지에서는 JSPWriter 타입으로 out 내장객체가 선언되어 있다. PrintWriter 클래스나 JSPWriter 클래스는 Writer 클래스를 상속한 출력 스트림 이다.

### ▶ out 내장객체를 이용해 응답 데이터 출력하기

- JSPStudyCh05/src/main/webapp/out/outObj.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
    /* JSP 페이지가 자바 소스로 변경된 파일을 열어서 _jspService() 메소드를
    * 살펴보면 JspWriter 타입으로 선언된 out 내장객체를 볼 수 있을 것이다.
    * out 객체는 스트림을 통해서 브라우저로 출력을 전달하는 내장객체이다.
    **/
    out.println("<h2>out 내장객체</h2>");
    out.println("<div>div 요소 안의 내용</div>");
%>
<!--
    아래와 같이 JSP 페이지에서 정적인 텍스트 데이터를 기술하면
    이 JSP 페이지가 서블릿 클래스로 변환될 때 _jspService()
    메서드 안에서 out 내장객체를 이용해 출력하는 코드로 변환된다.
-->
<div>여기도 out 내장객체를 이용해 출력된다.</div>
</body>
</html>
```

## 5.4 application 내장객체

하나의 웹 애플리케이션의 정보를 저장하고 관리하기 위해 사용하는 객체가 바로 application 내장객체이다. 이 application 내장객체는 ServletContext 클래스의 객체로써 애플리케이션에서 JSP 페이지(웹 컴포넌트)들 간의 데이터 공유를 위해 사용되며 주로 서버와 웹 애플리케이션의 정보에 접근할 수 있는 메서드를 제공하고 있다. 또한 application 내장객체도 데이터를 저장할 수 있는 속성을 제공하는 객체이며 application 내장객체에 저장된 데이터는 하나의 웹 애플리케이션에서 모든 JSP 페이지(웹 컴포넌트)에서 공통적으로 사용할 수 있다.

### ▶ application 내장객체의 메서드 사용하기

- JSPStudyCh05/src/main/webapp/application/applicationMethods.jsp

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

```

<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <%--
        applicaton 내장객체는 하나의 웹 애플리케이션 정보를 저장하고 관리하기 위해
        사용하는 객체로 ServletContext 타입이다. application 내장객체는 하나의
        웹 애플리케이션 안에서 유효하며 웹 애플리케이션을 사용하는 모든 사용자와 관련해
        파일을 업로드 하는 서버내의 폴더 정보나 웹 애플리케이션의 설정정보 등의 필요한
        정보를 웹 컴포넌트(JSP, Servlet) 간의 공유하기 위해 사용되며 주로 서버와
        웹 애플리케이션 정보에 접근할 수 있는 메서드를 제공하고 있다.

        아래는 application 내장 객체를 이용해 서버 정보와 서블릿 정보를 출력하고 있다.
    --%>
    <h3>웹 서버 정보</h3>
    <ul>
        <li>웹 서버 종류 : <%= application.getServerInfo() %></li>
        <li>서블릿 버전 : <%= application.getMajorVersion() %>.
            <%= application.getMinorVersion() %></li>
    </ul>
    <h3>웹 애플리케이션 정보</h3>
    <ul>
        <li>웹 애플리케이션 컨텍스트 패스 : <%= application.getContextPath() %></li>
        <li>웹 애플리케이션 파일의 절대경로 :
            <%= application.getRealPath("applicationMethods.jsp") %></li>
        <li>웹 애플리케이션 이름 : <%= application.getServletContextName() %></li>
    </ul>
</body>
</html>

```

## 5.5 데이터를 저장할 수 있는 4가지 내장객체

JSP 내장객체 중에서 데이터를 저장할 수 있도록 속성을 제공하는 내장객체는 `pageContext`, `request`, `session`, `application`과 같이 4가지 내장객체가 있으며 이들 내장객체의 데이터 저장 영역은 각 객체에 접근할 수 있는 유효범위라 할 수 있다.

`pageContext` 내장객체의 영역은 한 번의 요청을 처리하는 같은 JSP 페이지 안에서 유효하고 `request` 내장객체는 한 번의 요청을 처리하는 과정 안에서 유효하다. `session`은 하나의 브라우저 접속(한 명의 사용자 접속을 의미하며 이를 세션이라고 한다.) 안에서 유효한데 주로 한 사용자(같은 세션)와 관련된 정보(로그인 정보, 장바구니 등등)를 여러 JSP 페이지(웹 컴포넌트)가 공유하기 위해서 사용한다. 그리고 `application`은 하나의 웹 애플리케이션 안에서 유효하며 웹 애플리케이션을 사용하는 모든 사용자와 관련해서 파일을 업로드 하는 서버내의 폴더 정보나 웹 애플리케이션의 설정 정보 등을 공유하기 위해서 사용한다.

이번 예제는 JSP 페이지에서 데이터를 저장할 수 있는 내장객체인 `pageContext`, `request`, `session`, `application`의 속성에 데이터를 저장하고 사용하는 방법과 이 4가지 내장객체의 속성(데이



터 저장 영역)의 유효범위에 대해서 알아볼 것이다.

## ▶ 내장객체의 속성과 유효범위

- JSPStudyCh05/src/main/webapp/attribute/attrScopeForm.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<title>내장객체 속성의 유효범위</title>
</head>
<body>
    <h2>아이디와 이름 입력</h2>
    <form action="attrScopeProcess.jsp" method="post">
        아이디: <input type="text" name="id" /><br/>
        이름 : <input type="text" name="name" /><br/>
        <input type="submit" value="등록하기" />
    </form>
</body>
</html>
```

- JSPStudyCh05/src/main/webapp/attribute/attrScopeProcess.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%
    /* POST 방식 요청인 경우 setCharacterEncoding("UTF-8")을 호출해
    * 웹 브라우저의 인코딩 방식과 동일한 문자 셋인 UTF-8을 지정해야
    * 한글 데이터와 같은 유니코드 문자를 깨지지 않게 처리할 수 있다.
    *
    * getParameter() 메서드가 호출되기 전에 setCharacterEncoding("UTF-8")
    * 메서드를 먼저 호출해 request 영역의 문자 셋을 먼저 처리해야 한다.
    */
    request.setCharacterEncoding("utf-8");

    // request 내장객체를 이용해 요청 파라미터를 읽어온다.
    String id = request.getParameter("id");
    String name = request.getParameter("name");

    /* 속성을 제공하는 4가지 내장객체에 id를 저장한다.
    * 속성을 제공하는 내장객체에 저장할 수 있는 데이터는 기본형과 참조형 모두 가능하다.
    * 아래와 같이 setAttribute() 메서드를 이용해 속성에 데이터를 저장할 수 있고
    * getAttribute() 메서드와 EL을 이용해 속성의 데이터를 읽어 올 수 있다.
    */
```

```

/* pageContext 내장객체의 속성은 한 번의 요청을 처리하는 같은 JSP 페이지 내에서
 * 데이터를 공유하기 위해서 사용되며 주로 같은 JSP 내에서 스크립트릿과
 * EL(Expression Language) 간의 데이터를 교환할 때 사용된다.
 **/
pageContext.setAttribute("id", id);

/* request 내장객체의 속성은 한 번의 요청을 처리하는 서블릿과 JSP 페이지 간에
 * 데이터를 공유하기 위해 사용되며 RequestDispatcher나 pageContext 객체의
 * forward() 메소드를 이용해 요청 제어를 다른 페이지로 넘길 때 사용한다.
 * forward 되는 JSP 페이지의 스크립트릿이나 표현식에서 getAttribute() 메소드로
 * 속성의 데이터를 읽어 올 수 있으며 EL을 이용해도 속성의 데이터를 읽어 올 수 있다.
 **/
request.setAttribute("id", id);

/* session은 하나의 브라우저 접속(한 명의 사용자 접속을 의미하며 이를 세션이라고 함)
 * 안에서 유효한데 주로 한 사용자(같은 세션)와 관련된 정보(로그인 정보, 장바구니 등등)를
 * 여러 JSP 페이지(웹 컴포넌트)가 공유하기 위해서 사용한다.
 **/
session.setAttribute("id", id);

/* applicaton 내장객체는 하나의 웹 애플리케이션 정보를 저장하고 관리하기 위해
 * 사용하는 객체로 ServletContext 타입이다. application 내장객체는 하나의
 * 웹 애플리케이션 안에서 유효하며 웹 애플리케이션을 사용하는 모든 사용자와 관련해
 * 파일을 업로드 하는 서버내의 폴더 정보나 웹 애플리케이션의 설정정보 등의 필요한
 * 정보를 웹 컴포넌트(JSP, Servlet) 간의 공유하기 위해 사용되며 주로 서버와
 * 웹 애플리케이션 정보에 접근할 수 있는 메서드를 제공하고 있다.
 **/
application.setAttribute("id", id);

/* pageContext 내장객체를 이용해 다른 JSP 페이지로 포워딩 한다.
 **/
// pageContext.forward("attrScopeResult.jsp");
%>
<!DOCTYPE html>
<html>
<head>
<meta charset= "UTF-8">
<title>attrScopeProcess.jsp</title>
</head>
<body>
<h2>attrScopeProcess.jsp</h2>
<%--
    EL를 사용해 속성 이름을 지정하면 pageContext, request, session,
    application 4개의 영역에 저장된 속성을 작은 범위에서 큰 범위 순으로

```

검색하여 지정한 이름의 속성에 대한 값을 얻어 올 수 있다. 속성이 존재하지 않아도 NullPointerException은 발생하지 않고 아무것도 출력되지 않는다.

EL을 사용하면 속성에 데이터를 저장할 수 있는 JSP의 내장객체인 pageContext, request, session, application 영역을 순서대로 검색하여 첫 번째 만나는 속성에 해당하는 데이터를 읽어온다. EL 자체에도 JSP의 각 내장객체에 대응하는 EL의 내장객체를 아래와 같이 지원하고 있다.

JSP 내장객체	->	EL 내장객체
pageContext	->	pageScope
request	->	requestScope
session	->	sessionScope
application	->	applicationScope

아래와 같이 EL 안에서 데이터를 읽어 올 영역(scope)을 지정하면 검색하지 않고 해당하는 영역에서 데이터를 읽어 온다. 해당하는 영역에 지정한 데이터가 존재하지 않으면 아무것도 출력되지 않는다.

```
--%>
page : ${ pageScope.id }<br/>
request : ${ requestScope.id }<br/>
session : ${ sessionScope.id }<br/>
application : ${ applicationScope.id }<br/><br/>
attrScopeResult.jsp
</body>
</html>
```

#### - JSPStudyCh05/src/main/webapp/attribute/attrScopeResult.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>attrScopeResult.jsp</title>
</head>
<body>
<body>
    <h2>attrScopeResult.jsp</h2>
<%--
```

EL를 사용해 속성 이름을 지정하면 pageContext, request, session, application 4개의 영역에 저장된 속성을 작은 범위에서 큰 범위 순으로 검색하여 지정한 이름의 속성에 대한 값을 얻어 올 수 있다. 속성이 존재하지 않아도 NullPointerException은 발생하지 않고 아무것도 출력되지 않는다.

EL을 사용하면 속성에 데이터를 저장할 수 있는 JSP의 내장객체인 pageContext, request, session, application 영역을 순서대로 검색하여 첫 번째 만나는 속성에 해당하는 데이터를 읽어온다. EL 자체에도 JSP의 각 내장객체에 대응하는 EL의 내장객체를 아래와 같이 지원하고 있다.

JSP 내장객체	->	EL 내장객체
pageContext	->	pageScope
request	->	requestScope
session	->	sessionScope
application	->	applicationScope

아래와 같이 EL 안에서 데이터를 읽어 올 영역(scope)을 지정하면 검색하지 않고 해당하는 영역에서 데이터를 읽어 온다. 해당하는 영역에 지정한 데이터가 존재하지 않으면 아무것도 출력되지 않는다.

--%>

<%--

attrScopeProcess.jsp에서 pageContext 내장객체의 속성에 저장한 데이터는 유효범위를 벗어나기 때문에 아래에서 아무것도 출력되지 않는다.

--%>

page : \${ pageScope.id }<br/>

<%--

attrScopeProcess.jsp에서 포워딩하여 이 페이지로 제어가 이동해 왔다면 request 내장객체의 속성에 저장한 데이터는 유효하기 때문에 id가 출력된다. 하지만 attrScopeProcess.jsp 페이지에서 링크를 클릭해서 이 페이지로 넘어왔다면 id는 출력되지 않는다. 이유는 attrScopeProcess.jsp 페이지가 브라우저 화면에 출력될 때 이미 서버에서 응답이 완료되었기 때문에 그 때 서버에서 사용한 request 내장객체는 사라져서 존재하지 않으므로 유효범위를 벗어나게 된다. 결과가 브라우저 화면에 출력된 후 링크를 클릭하여 서버로 요청하면 새로운 요청이 발생하기 때문에 이전의 request 내장객체는 더 이상 유효하지 않게 된다.

--%>

request : \${ requestScope.id }<br/>

<%--

session과 application 내장객체는 여러 번의 요청에서도 유효하다. 사용한 브라우저 화면이 모두 닫히면 세션이 종료되어 현재 session 내장객체의 유효범위를 벗어나게 되고 현재 웹 애플리케이션이 다시 시작되면 이전의 application 내장객체는 소멸되고 다시 생성되기 때문에 현재의 application 내장객체의 유효범위를 벗어나게 된다.

--%>

session : \${ sessionScope.id }<br/>

application : \${ applicationScope.id }<br/><br/>

<a href="#">새로고침</a>

```
</body>
</html>
```

### [연습문제 5-1] 선호도 테스트

이클립스에서 JSPStudyCh05Exam이라는 DynamicWebProject를 생성하고 아래 그림과 같이 선호도 테스트 폼 양식을 만들고 전송하기를 클릭하면 JSP 페이지에서 요청된 데이터를 받아 forward 기법을 이용해 요청 처리와 출력(View)을 분리해서 처리하는 프로그램을 구현하시오.

아래를 참고해 각각의 기능에 따른 파일명을 지정하여 프로그램을 작성하시오.

- 선호도를 입력 받는 폼페이지 : webapp/ch0501ExamForm.jsp
- 선호도 등록 요청을 처리하는 페이지 : webapp/ch0501ExamProcess.jsp
- 선호도 입력 결과를 출력하는 페이지 : webapp/ch0501ExamResult.jsp

#### 선호도 테스트

이름 :

좋아하는 색 : ☒ 빨강색 ☐ 초록색 ☐ 파랑색

좋아하는 음식 :

좋아하는 동물(모두 고르세요):  
☒ 햄스터 ☒ 고양이 ☐ 호랑이 ☐ 사자 ☐ 개

취미(모두 고르세요):

#### 선호도 테스트 결과

##### 홍길동님의 선호도 테스트 결과

홍길동님은 빨강색을 좋아하고, 짜장면을 좋아하며,  
좋아하는 동물은 햄스터, 고양이이고,  
게임, 여행, 독서의 취미를 가지고 계십니다.

## 6. JSP 데이터베이스 프로그래밍

JDBC(Java Database Connectivity)는 자바 응용프로그램과 RDBMS(Relational Database Manager System) 간의 표준적인 연결 방법을 제공하기 위해 정의해 놓은 자바를 위한 데이터베이스 접속 인터페이스이다. 다시 말해 JDBC는 자바 응용프로그램에서 SQL 명령을 사용해 RDBMS(Relational Database Manager System)에 CRUD(Create, Read, Update, Delete) 작업을 위한 표준적인 프로그래밍 방식을 제공하는 라이브러리이다. 이 JDBC 라이브러리는 java.sql 패키지에 정의되어 있으며 RDBMS에 접근 할 수 있는 방법을 제공하는 단일 API(Application Programming Interface)이다.

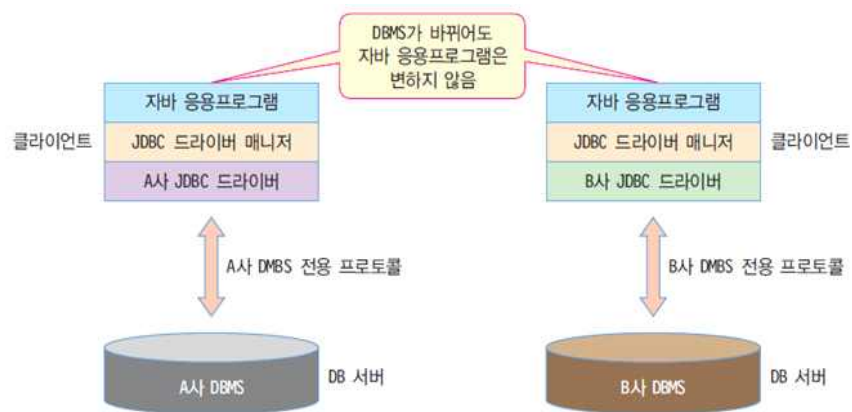


그림 6-1 JDBC의 구조

### 6.1 자바 응용프로그램에서 JDBC 프로그램을 작성하는 순서

1 단계 : `import java.sql.*;`

2 단계 : 각 DBMS에 맞는 JDBC 드라이버를 로드 한다.

JDBC를 이용해 데이터베이스를 연동하는 웹 애플리케이션을 작성하기 위해서는 DBMS에 맞는 접속 드라이버가 있어야 한다. JDBC 접속 드라이버는 자바 응용프로그램과 DBMS간의 연결을 지원하기 위해 각각의 DBMS 제작사들이 자신의 DBMS에 맞게 JDBC 인터페이스를 구현한 클래스들의 묶음이다. 우리는 이들이 제공하는 JDBC 드라이버를 이용해 각각의 DBMS와 연동하는 웹 애플리케이션을 표준화된 방법으로 편리하게 작성할 수 있다.

오라클 접속 드라이버는 “<http://www.oracle.com/us/downloads/index.html>”에서 다운로드 받을 수 있으며 Oracle Expression 11g가 설치된 아래의 폴더에서도 구할 수 있다.

C:\oraclexe\app\oracle\product\11.2.0\server\jdbc\lib\ojdbc6.jar

MySQL 접속 드라이버는 “<http://dev.mysql.com/downloads/connector/j/>”에서 다운로드 받을 수 있다.

JDBC 드라이버를 다운로드 받아 웹 애플리케이션 프로젝트의 “webapp/WEB-INF/lib” 폴더에 추가하면 된다.

JDBC 드라이버를 로드 하는 방법은 어떤 메소드를 사용하느냐에 따라서 몇 가지가 있을 수 있으나 우리는 클래스의 정보를 다루기 위해 자바에서 제공하는 Class 클래스의 forName() 메소드를 이용해 JDBC 드라이버를 동적으로 로드하여 사용할 것이다. 이 메소드의 인수로 접속 할 DBMS에 맞는 Driver 클래스 이름을 문자열로 지정해야 한다. 드라이버 클래스의 이름을 문자열로 지정할 때는 Driver 클래스가 소속된 패키지를 포함하는 완전한 클래스 이름(Fully Qualified Name)을 기재해야 한다.

자바 응용프로그램이 실행되고 forName() 메소드가 호출되면 Driver 클래스를 동적으로 로드하고 드라이버 클래스의 인스턴스를 생성하여 DriverManager에 등록하게 된다. 다시 말해 Driver 클래스의 객체가 실행 시간에 메모리에 생성 되는 것이다.

DBMS에 따라 JDBC 드라이버의 클래스 이름이 다르기 때문에 해당 DBMS의 JDBC 문서를 참조하여 적절한 드라이버 클래스 이름을 지정해야 한다. 만일 지정한 Driver 클래스가 존재하지 않는다면 ClassNotFoundException이 발생하므로 적절한 예외처리 또한 필요하다.

아래는 Oracle과 MySQL의 JDBC 드라이버를 로드하여 객체를 생성하고 DriverManager에 등록하는 코드이다.

```
Oracle DBMS : Class.forName("oracle.jdbc.driver.OracleDriver");
```

```
MySQL DBMS : Class.forName("com.mysql.jdbc.Driver")
```

### 3 단계 : Connection 객체를 생성한다.

DriverManager 클래스는 자바 응용프로그램을 JDBC 드라이버에 연결시켜 주는 역할을 하는 클래스이다. DBMS 드라이버가 메모리에 생성되면 실제 DBMS가 설치된 접속 URL을 DriverManager 클래스의 getConnection() 메소드의 인수로 지정하여 호출 한다.

getConnection() 메소드가 호출되면 DriverManager에 등록된 여러 DBMS의 Driver 중에서 2단계에서 지정한 Driver를 식별하고 그와 관련된 DBMS와 연결된 Connection 객체를 반환 한다. getConnection() 메서드를 사용해 DBMS와 접속을 시도할 때 접속 URL의 정보나 사용자ID, 비밀번호가 일치하지 않으면 SQLException이 발생하므로 적절한 예외 처리가 필요하다.

아래는 Oracle과 MySQL의 접속 URL과 사용자 ID, 비밀번호를 지정하는 예이다. 아래 예와 같이 접속 URL은 “jdbc:” 이후에 URL에 포함되는 형식이 DBMS 마다 다르기 때문에 해당 DBMS의 JDBC 문서를 참조하여 적절한 URL을 지정해야 한다.

```
Oracle DBMS : DriverManager.getConnection(
```

```
    “jdbc:oracle:thin:@서버ip:1521:SID”, “사용자ID”, “비밀번호”);
```

```
MySQL DBMS : DriverManager.getConnection(
```

```
    “jdbc:mysql://서버ip:3306/DB명”, “사용자ID”, “비밀번호”);
```

### 4 단계 : Statement 또는 PreparedStatement 객체를 생성한다.

Statement 객체를 생성하고 DBMS에 요청할 SQL 쿼리를 생성하는 단계이다.

Statement 객체와 PreparedStatement 객체는 DBMS에 SQL 쿼리를 발행할 수 있도록 도와주는 객체로 DBMS에 연결된 Connection 객체로부터 아래와 같이 얻을 수 있다.  
4단계부터 특정 DBMS에 구애받지 않고 동일한 코드로 DB 작업을 할 수 있다.

- ① `Statement stmt = connection.createStatement();`
- ② `PreparedStatement stmt = connection.prepareStatement("SQL 쿼리문");`

## ▶ Statement 객체

DBMS와 연결된 Connection 객체로부터 Statement 객체를 얻어올 때는 위의 코드 ①에서와 같이 SQL 쿼리를 지정하지 않고 실제 SQL 쿼리를 발행하는 단계인 `executeQuery()` 메소드나 `executeUpdate()` 메소드를 호출 할 때 메소드의 인수로 SQL 쿼리를 지정한다.

Statement 객체를 생성하고 한번 `executeQuery()` 메소드나 `executeUpdate()` 메소드를 호출 했다면 더 이상 이 객체를 이용해 `executeQuery()` 메소드나 `executeUpdate()` 메소드를 호출 할 수 없기 때문에 1회성의 단순한 질의 문에 주로 사용한다.

Statement 객체를 이용해 `executeQuery()` 메소드나 `executeUpdate()` 메소드를 다시 호출하기 위해서는 Statement 객체를 다시 생성해야 호출할 수 있다.

아래 코드는 Statement 객체를 사용하는 예이다.

```
String sql = "SELECT * FROM jspbbs";
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(sql);
```

## ▶ PreparedStatement 객체

DBMS와 연결된 Connection 객체로부터 PreparedStatement 객체를 얻어올 때는 위의 코드 ②에서와 같이 SQL 쿼리를 메서드의 인수로 지정해야 한다. 이는 PreparedStatement 객체가 여러 번 사용되는 SQL 문에서 특정 데이터만 바뀌가면서 동일한 SQL 쿼리를 여러 번 실행하기 위해 SQL 쿼리를 캐싱(저장)하여 사용하기 때문이다.

예를 들어 테이블에 여러 행의 데이터를 추가 할 경우 테이블의 구조는 같으나 각 셀마다 데이터가 다르므로 질의 문을 미리 작성하여 캐싱(저장)해 놓고 실제 입력되는 데이터만 바뀌가면서 데이터를 추가할 수 있다. 즉 SQL 문의 기본 구조는 같지만 입력되는 데이터나 조건식에 사용되는 데이터가 수시로 변경될 경우 주로 사용한다.

PreparedStatement 객체는 SQL 쿼리에 위치 표시자(이하 Placeholder)인 물음표(?)를 사용하여 SQL 쿼리 문을 캐싱한 후 `setXxx()` 메소드를 사용해 입력되는 데이터나 조건식에 사용되는 데이터를 Placeholder(?)가 지정된 위치에 설정할 수 있도록 지원하고 있다. Placeholder(?)에 다양한 데이터를 지정할 수 있도록 다양한 데이터 타입에 대응하는 `setXxx()` 메소드가 오버로딩 되어 있다.

여기서 주의할 것은 PreparedStatement 객체의 `setXxx()` 메소드에 지정하는 파라미터의 index는 자바 배열에서 사용되는 index 개념이 아니라 첫 번째 위치의 Placeholder(?), 두 번째 위치의 Placeholder(?)를 가리키는 위치 개념으로 시작 번호가 0이 아닌 1부터 시작된다는 것이다. 아래 코드는 PreparedStatement 객체를 사용하는 예이다.



```

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

PreparedStatement pstmt = conn.prepareStatement(sql);

String title= board.getTitle();
String writer = board.getWriter();
String content = board.getContent();
String pass = board.getPass();

/* 첫 번째 Placeholder(?), 두 번째 Placeholder(?)를 의미하므로 파라미터의
 * index는 위치의 개념으로 1부터 시작한다. PreparedStatement 클래스는
 * 다양한 데이터 타입에 대응하는 setter 메소드를 제공하고 있다.
 */
pstmt.setString(1, title);
pstmt.setString(2, writer);
pstmt.setString(3, content);
pstmt.setString(4, pass);

int result = pstmt.executeUpdate();

```

## 5 단계 : 쿼리로 질의한 결과를 ResultSet 객체로 얻어 데이터를 추출한다.

SELECT 문의 경우는 데이터베이스에 질의하고 그 결과로 ResultSet 객체를 반환 받는다.

이 ResultSet 객체는 SELECT 질의에 대한 결과 집합으로 테이블 구조를 가지는 데이터 이다.

ResultSet 객체는 테이블의 행에 대한 참조를 조작하기 위해 커서(Cursor)를 가지고 있으며 이 커서를 이용해 첫 번째 행이나 마지막 행 또는 바로 이전행과 같이 각 행을 이동할 수도 있고 순차적으로 이동하며 데이터를 읽어 올 수도 있다.

SELECT 쿼리가 실행되고 데이터를 읽어오면 커서의 첫 번째 위치는 실제 데이터 행의 위치보다 작은 -1이 된다. 즉 맨 처음 ResultSet 객체를 얻어오면 커서는 첫 번째 행 바로 이전을 가리키고 있는 형태가 된다.

아래 코드는 ResultSet 객체의 메소드를 사용해 순차적으로 데이터를 읽어오는 예이다.

```

while(rs.next()) {

    // 현재 행의 첫 번째 컬럼, 두 번째 컬럼을 의미하므로 index는 1부터 시작한다.
    int no = rs.getInt(1);

    /* ResultSet에는 다양한 데이터 타입에 대응하는 getter 메소드를 제공한다.
     * SELECT 문장에서 지정한 컬럼의 index 또는 컬럼명으로 테이블의 필드
     * 값을 가져올 수 있도록 다양한 데이터 타입에 대응하는 getXxx() 메소드가
     * 오버로딩 되어 있다. 컬럼의 index 또한 첫 번째 컬럼, 두 번째 컬럼을
     * 가리키는 위치 개념으로 1부터 시작 한다.
    */
}

```

```

    **/
    String name = rs.getString("writer");
    String id = rs.getString(3);
}

```

6 단계 : DB 작업이 끝나면 작업한 객체를 생성한 역순으로 닫아 준다.

```

// ResultSet -> Statement(PreparedStatement) -> Connection순으로 닫는다.
try {
    if(rs != null) rs.close();
    if(pstmt != null) pstmt.close();
    if(conn != null) conn.close();
} catch(SQLException e) { e.printStackTrace(); }

```

## 6.2 데이터베이스에 접속해 게시 글 리스트 읽어오기

데이터베이스의 테이블에 저장된 데이터를 읽어와 저장하기 위해 사용하는 객체를 VO(Value Object)라고 부른다. 또는 이런 객체를 도메인 객체 혹은 DTO(Data Transfer Object)라고도 부르는데 앞으로 VO라는 용어로 통일해 부를 것이다. 이 VO 객체는 테이블로부터 읽어온 한 행의 데이터를 객체로 저장하고 관리하기 위해서 사용하는 객체이다. 일반적으로 DB 작업을 하다보면 테이블로부터 읽어온 데이터를 저장하고 그 데이터를 다른 곳으로 전달해야 하는 경우가 많은데 이럴 경우 VO 객체를 사용하면 편리하다.

### ▶ VO(Value Object) 클래스

먼저 com.jspstudy.ch06.vo 패키지를 만들고 jspbbs 테이블의 한 행의 데이터를 저장하는 다음과 같은 Board 클래스를 작성하자.

#### - com.jspstudy.ch06.vo

```

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

```

```

private String file1;

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

public int getNo() {
    return no;
}
public void setNo(int no) {
    this.no = no;
}
public String getTitle() {
    return title;
}
public void setTitle(String title) {
    this.title = title;
}
public String getWriter() {
    return writer;
}
public void setWriter(String writer) {
    this.writer = writer;
}
public String getContent() {
    return content;
}
public void setContent(String content) {
    this.content = content;
}
public Timestamp getRegDate() {
    return regDate;
}
public void setRegDate(Timestamp regDate) {
    this.regDate = regDate;
}
}

```

```

public int getReadCount() {
    return readCount;
}
public void setReadCount(int readCount) {
    this.readCount = readCount;
}
public String getPass() {
    return pass;
}
public void setPass(String pass) {
    this.pass = pass;
}
public String getFile1() {
    return file;
}
public void setFile1(String file1) {
    this.file = file1;
}
}

```

## ▶ JSP페이지에서 DB의 게시 글 리스트 읽어오기

- webapp/dbConnection/jspDBConnection.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="java.sql.*, java.util.*, com.jspstudy.ch06.vo.*" %>
<%--
    JSP 표준 태그 라이브러리(JSTL)를 사용하기 위한 taglib 지시자
    http://jakarta.apache.org, http://tomcat.apache.org 에서
    다운로드 하여 WEB-INF/lib 폴더에 추가해야 표준 태그라이브러리를 사용할 수 있다.
--%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
    // 오라클 사용자 정보와 접속 드라이버 이름 및 접속 URL 등을 변수에 저장
    String user = "hr";
    String pass = "hr";
    String driver = "oracle.jdbc.driver.OracleDriver";
    String url = "jdbc:oracle:thin:@localhost:1521:xe";
    String title = "게시 글 리스트";
    String select = "SELECT * FROM jsbbbs";

    /* 1. 접속하려는 DBMS의 드라이버를 로딩 한다.
       * Oracle 드라이버를 다운로드 하여 WEB-INF/lib 폴더에 추가해야 한다.
       **/
    Class.forName(driver);

```

```

// 2. 데이터베이스 연결 - DB 커넥션을 생성한다.
Connection conn = DriverManager.getConnection(url, user, pass);

/* 3. DBMS에 SQL 쿼리를 발생하기 위해 활성화된
 * Connection 객체로 부터 Statement 객체를 얻는다.
 */
Statement stmt = conn.createStatement();

// 4. 쿼리를 실행하여 SELECT한 결과를 ResultSet 객체로 받는다.
ResultSet rs = stmt.executeQuery(select);

// 여러 개의 게시 글을 ArrayList에 저장하기 위한 객체 생성
ArrayList<Board> boardList = new ArrayList<Board>();

/* 5. 쿼리 실행 결과를 바탕으로 while문을 이용해 데이터를 출력한다.
 * ResultSet의 next() 메서드는 다음 행의 데이터가 존재하면 true를 반환하므로
 * while 문을 이용해 첫 번째 행부터 마지막 행까지 이동하면서 데이터를 추출할 수 있다.
 *
 * 반복문 안에서 ResultSet 객체의 getXXX() 메서드를 이용해 데이터를
 * 읽어올 수 있다. ResultSet 객체는 자바의 모든 타입에 대응되는 getXXX()
 * 메서드를 제공하고 있으며 아래와 같이 컬럼의 위치(첫 번째, 두 번째...) 값을
 * 정수로 지정하거나 컬럼 이름을 지정해 데이터를 읽어 올 수 있다.
 */
while(rs.next()) {
    Board b = new Board();
    b.setNo(rs.getInt(1));
    b.setTitle(rs.getString("title"));
    b.setWriter(rs.getString(3));
    b.setRegDate(rs.getTimestamp(5));
    b.setReadCount(rs.getInt("read_count"));

    boardList.add(b);
}

/* 6. 사용한 ResultSet과 Statement 객체를 종료 한다.
 * 객체를 생성한 역순으로 닫으면 된다.
 */
if(rs != null) rs.close();
if(stmt != null) rs.close();

// 7. DBMS 연결을 종료한다.
if(conn != null) rs.close();
%>
<%--

```

이 페이지에서는 JSTL의 코어 라이브러리에 속한 <c:set> 태그를 사용하여 pageContext, request, session, application 4개의 영역에 속성으로 데이터를 저장하고 EL 식을 이용해 출력하는 기법을 소개하고 있다.

JSTL의 코어 라이브러리는 말 그대로 JSTL의 가장 핵심적인 기능을 제공하는 라이브러리로 프로그래밍 언어에서 일반적으로 제공하고 있는 변수 선언, 조건문, 반복문에 해당하는 태그를 지원한다. 또한 익셉션, URL 저장, 데이터 출력과 관련된 태그와 다른 JSP 페이지 호출(import, redirect)과 관련된 태그를 지원한다.

아래는 <c:set> 태그를 이용해 pageContext 영역의 속성에 title 이라는 속성 이름으로 위에서 title 변수에 저장한 문자열 데이터를 저장하고 있다. 아래와 같이 scope에 데이터를 저장할 수 있는 속성을 제공하는 내장 객체를 의미하는 page, request, session, application을 지정할 수 있다. scope가 생략되면 기본 값은 page로 설정되어 pageContext 객체에 저장된다.

```
--%>
<c:set var="title" value="%= title %>" scope="page"/>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JSP페이지에서 데이터베이스에 접속해 데이터 읽어오기</title>
<style type="text/css">
    table{
        border: 1px solid blue;
        border-collapse: collapse;
    }
    td {
        border: 1px dotted blue;
        padding: 5px 10px;
    }
</style>
</head>
<body>
    <!-- <c:set> 태그를 사용해 pageContext 속성에 저장한 데이터 출력 --%>
    <h1>${ title }</h1>
    <table>
        <tr>
            <td>no</td>
            <td>제목</td>
            <td>작성자</td>
            <td>작성일</td>
            <td>조회수</td>
        </tr>
        <!-- items에 스크립틀릿에서 정의한 boardList를 표현식으로 지정할 수 있다. -->
        <c:forEach var="board" items="%= boardList %>">
```

```

<tr>
  <td>${ board.no }</td>
  <td>${ board.title }</td>
  <td>${ board.writer }</td>
  <td>${ board.regDate }</td>
  <td>${ board.readCount }</td>
</tr>
</c:forEach>
</table>
</body>
</html>

```

## 6.3 DAO(Data Access Object) 사용하기

앞의 예제에서 DB에 접근해 게시 글 리스트 정보를 읽어와 웹 페이지에 출력하는 방법에 대해서 알아보았다. 앞에서와 같이 JSP에서 DB에 접근하기 위해서 데이터베이스 접속 드라이버를 동적으로 로딩하여 데이터베이스에 접속하는 코드를 각 JSP 페이지 마다 기술해야 한다면 여러 JSP에서 동일한 코드가 중복되는 문제가 발생하게 된다. 코드의 중복을 최소화하기 위해서는 데이터베이스에 접속해 작업하는 코드를 JSP 페이지에서 분리해 별도의 클래스로 만들면(모듈화 하면) 중복 문제를 해결 할 수 있다.

실무에서는 이렇게 코드의 중복을 없애고 작업의 효율을 높이기 위해서 데이터베이스 작업을 전담하는 코드를 분리해 별도의 클래스를 만들어 사용하는데 이런 클래스를 데이터에 직접 접근하는 객체라는 의미로 DAO(Data Access Object) 객체라고 부른다.

이번 예제에서 데이터베이스 작업을 전담하는 BoardDao 클래스와 하나의 게시 글 정보를 저장하는 VO(Value Object) 클래스인 Board 클래스를 이용해 게시 글 리스트를 출력하는 방법에 대해서 알아 볼 것이다.

### ▶ DAO(Data Access Object) 클래스

com.jspstudy.ch06.dao 패키지를 만들고 다음과 같이 BoardDao01 클래스를 작성하자.

#### - com.jspstudy.ch06.dao.BoardDao01

// JDBC를 활용한 DAO(Data Access Object) 클래스

```
public class BoardDao01 {
```

```
// 오라클 접속에 필요한 정보를 static 상수로 정의
```

```
private static final String DRIVER = "oracle.jdbc.driver.OracleDriver";
```

```
private static final String URL = "jdbc:oracle:thin:@localhost:1521:xe";
```

```
private static final String USER = "hr";
```

```
private static final String PASS = "hr";
```

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

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

```
Connection conn;
```

```

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

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

// 기본 생성자가 호출될 때 DB 접속 드라이버를 로딩 하도록 구현하였다.
public BoardDao01() {

    try {
        /* 1. DBMS의 접속 드라이버를 로딩 한다.
         * Oracle 드라이버를 다운로드 하여 WEB-INF/lib 폴더에 추가해야 한다.
         *
         * 아래와 같이 forName() 메서드에 로딩할 접속 드라이버의 클래스 이름을
         * 지정하면 지정한 JDBC 드라이버를 읽어 DriverManager에 등록한다.
         */
        Class.forName(DRIVER);

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

// DB에 등록된 전체 게시 글을 읽어와 ArrayList로 반환하는 메서드
public ArrayList<Board> boardList() {

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

    try {
        /* 2. 데이터베이스 연결 - DB 커넥션을 생성한다.
         * getConnection() 메서드는 인수로 지정한 접속 정보를 바탕으로
         * DriverManager에 등록된 접속 드라이버를 이용해 데이터베이스에
         * 접속하고 데이터베이스와 연결된 Connection 객체를 반환한다.
         */
        conn = DriverManager.getConnection(URL, USER, PASS);

        /* 3. DBMS에 SQL 쿼리를 발생하기 위해 활성화된
         * Connection 객체로 부터 PreparedStatement 객체를 얻는다.
         *
         * PreparedStatement는 SQL 명령을 캐싱하여(저장하여) 반복적으로 사용하기
         * 때문에 preparedStatement()를 호출할 때 SQL 쿼리 문을 지정해야 한다.
         */
        pstmt = conn.prepareStatement(sqlBoardList);
    }
}

```



```

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

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

/* 5. 쿼리 실행 결과를 바탕으로 while문 안에서 하나의 게시 글 정보를 저장할
 * Board 객체를 생성하고 이 객체에 하나의 게시 글 정보를 저장하고
 * Board 객체를 다시 ArrayList에 저장한다.
 *
 * ResultSet 객체는 DB로 부터 읽어온 데이터에 접근하기 위해 테이블의
 * 행을 가리키는 cursor를 제공한다. 맨 처음 ResultSet 객체를 반환
 * 받으면 cursor는 첫 번째 행 바로 이전을 가리키고 있다. ResultSet의
 * cursor가 맨 마지막 행에 도달하면 while문을 빠져 나온다.
 *
 * ResultSet에는 자바의 다양한 데이터 타입에 대응하는 getter 메소드를
 * 지원하고 있으며 SELECT 문장에서 지정한 컬럼의 index 또는
 * 컬럼명으로 테이블의 필드 값을 가져올 수 있도록 getXxx() 메소드가
 * 오버로딩 되어 있어 index와 컬럼명 둘 다 사용이 가능하다.
 * 여기에 지정하는 index는 배열에서 사용되는 index의 개념이 아니라
 * 첫 번째 컬럼, 두 번째 컬럼과 같이 위치의 개념으로 1부터 시작된다.
 */
while(rs.next()) {

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

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

```

```

        b.setFile1(rs.getString("file1"));

        boardList.add(b);
    }
} catch(SQLException e) {
    System.out.println("BoardDao - boardList() - SQLException");
    e.printStackTrace();

} finally {
    try {
        // 6. 사용한 ResultSet과 PreparedStatement를 종료한다.
        if(rs != null) rs.close();
        if(pstmt != null) pstmt.close();

        // 7. Connection 객체를 닫아 DBMS 연결을 종료한다.
        if(conn != null) conn.close();
    } catch(SQLException e) {}
}

// 8. 데이터베이스로 부터 읽어온 게시 글 리스트를 반환한다.
return boardList;

} // end boardList();
}

```

## ▶ BoardDao를 사용해 게시 글 리스트 요청을 처리하는 JSP

- webapp/board/boardList01.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="com.jspstudy.ch06.dao.*, com.jspstudy.ch06.vo.*" %>
<%@ page import="java.util.*" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
    // BoardDao 객체를 생성하고 데이터베이스에서 게시 글 리스트를 읽어온다.
    BoardDao01 dao = new BoardDao01();
    ArrayList<Board> bList = dao.boardList();
%>
<%--

```

아래에서 EL로 접근하기 위해서 <c:set> 태그를 이용해 DB에서 읽어온 게시 글 리스트를 pageContext 영역의 속성으로 저장하는 코드이다.

<c:set> 태그를 이용해 pageContext 영역의 속성에 bList 라는 속성 이름으로 위의 스크립틀릿에서 사용한 변수인 bList를 표현식으로 지정하고 있다. 아래와 같이 scope에 데이터를 저장할 수 있는 속성을 제공하는 내장 객체를

의미하는 page, request, session, application을 지정할 수 있다.

scope가 생략되면 기본 값은 page로 설정되어 pageContext 객체에 저장된다.

--%>

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

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

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

```
<title>게시 글 리스트</title>
```

```
<style type="text/css">
```

```
table{
```

```
border: 1px solid blue;
```

```
border-collapse: collapse;
```

```
}
```

```
td {
```

```
border: 1px dotted blue;
```

```
padding: 5px 10px;
```

```
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<h2>게시 글 리스트</h2>
```

```
<table>
```

```
<tr>
```

```
<td>no</td>
```

```
<td>제목</td>
```

```
<td>작성자</td>
```

```
<td>작성일</td>
```

```
<td>조회수</td>
```

```
</tr>
```

```
<%--
```

```
<c:if> 태그의 test 속성에 조건식을 기술하며 이 조건식이 참일 때
```

```
<c:if> ~ </c:if> 사이의 코드가 실행된다.
```

게시 글 리스트가 존재하면 게시 글 수만큼 반복하면서 게시 글을 출력

--%>

```
<c:if test="${ not empty bList }">
```

```
<%--
```

<c:forEach> 태그는 자바의 for문과 forEach문 두 가지 기능을 제공한다.

자바의 for문과 같이 초깃값부터 순차적으로 반복하기 위해 아래와 같은

속성을 지정하여 사용한다. var 속성에는 forEach문 안에서 사용할 변수

이름을 지정하고 begin 속성은 var 속성에 지정한 변수의 시작 값을 지정하며

end 속성에는 변수의 끝 값을 지정한다.(i <= end 가 적용된다.)

step 속성은 생략할 수 있고 생략하게 되면 기본 값은 1이 된다.

```
<c:forEach var="i" begin="0" end="10" step="1" >
```

아래에 사용된 코드는 forEach문의 기능을 하는 코드로 items 속성에 배열 변수의 이름을 지정하고 var 속성에 forEach문 안에서 배열의 각 항목을 저장하는 변수의 이름을 지정한다. items 속성에 지정할 수 있는 변수는 스크립트릿에서 선언한 변수를 표현식`<%= member %>`으로 지정할 수 있고 아래와 같이 속성에 저장된 속성 이름을 EL 식을 이용해 지정할 수도 있다. items 속성에 지정할 수 있는 데이터는 배열, Collection 객체, Iterator 객체, Enumeration 객체, Map 객체와 콤마(,)로 구분된 문자열 등을 지정할 수 있다.

```
--%>
<c:forEach var="b" items="${bList}">
<tr>
<td>${ b.no }</td>
<td>${ b.title }</td>
<td>${ b.writer }</td>
<td>${ b.regDate }</td>
<td>${ b.readCount }</td>
</tr>
</c:forEach>
</c:if>
<%-- 게시 글 리스트가 존재하지 않으면 --%>
<c:if test="${ empty bList }">
<tr>
<td colspan="5">게시 글이 존재하지 않습니다.</td>
</tr>
</c:if>
</table>
</body>
</html>
```

## 6.4 데이터베이스 커넥션 풀(Database Connection Pool)

데이터베이스 커넥션 풀(Database Connection Pool, DBCP)이란 데이터베이스와 작업하기 위해 필요할 때 마다 데이터베이스와 연결하는 것이 아니라 미리 일정한 수의 커넥션을 생성해 데이터베이스 커넥션 풀에 저장해 놓고 필요한 시점에 애플리케이션에서 임대해 데이터베이스 작업을 한 후 커넥션 풀에 다시 커넥션을 반납하는 일련의 데이터베이스 작업의 관리 체계를 의미한다.

데이터베이스에 동시에 접속할 수 있는 커넥션은 한정되어 있고 웹 서버에서 동시 다발적으로 발생하는 수많은 데이터베이스 작업 요청이 들어올 때 마다 커넥션을 생성하는 것은 현실적으로 불가능하다. 그래서 데이터베이스에 접속하기 위한 일정한 수의 커넥션을 미리 생성해 놓고 필요할 때 마다 대여하여 사용한 후 반납함으로써 원격지에 있는 데이터베이스에 연결하기 위한 사용자 인증 등에 소요되는 시간을 절약할 수 있으며 커넥션의 재사용성을 높이기 위해 커넥션 풀이 널리 사용되고 있다.

커넥션 풀을 사용하는 여러 가지 방법 있을 수 있으나 우리는 최근에 많이 사용되고 있는 JNDI(Java Naming and Directory Interface) 방식의 DBCP(Database Connection Pool) 방식을

사용하는 방법에 대해 알아볼 것이다.

먼저 커넥션 풀에 사용되는 라이브러리 파일을 아래 URL(Apache Commons 프로젝트)에서 dbcp와 pool 라이브러리를 다운받아 프로젝트의 WEB-INF/lib 폴더에 추가하자.

다운로드 받을 필요한 라이브러리 파일은 프로젝트의 webapp/WEB-INF/lib 폴더의 “commons-pool2\_참고.txt” 파일을 참고하여 아래 사이트에서 다운로드를 받으면 된다.

다운로드 사이트 : <http://commons.apache.org>

### 6.4.1 JNDI를 이용한 커넥션 풀 사용하기

JNDI(Java Naming and Directory Interface)란 자바에서 네이밍 서비스를 이용할 수 있도록 제공하는 인터페이스로 논리적인(가상의) 이름을 디렉토리 서비스의 파일 또는 자바 객체, 서버 등과 연결해주는 서비스를 말하며 DNS가 도메인 이름을 IP주소로 변환해 주는 서비스와 비슷한 개념으로 데이터 및 객체를 발견하고 참고하기 위한 자바 API이다. 대표적인 디렉토리 서비스에는 LDAP(Light weight Directory Access Protocol), Active Directory 서비스가 있다.

JNDI는 대규모 애플리케이션 개발의 분산처리 환경에서 여러 가지 형태의 자원을 효율적으로 관리하고 참조할 수 있는 장점이 있으며 J2EE 플랫폼의 일부이다. 또한 JNDI 인터페이스는 javax.naming 패키지에 존재하며 모든 리소스는 기본 네임스페이스인 java:comp/env에 리소스 이름을 추가하는 형식으로 지정한다.

JNDI를 이용한 데이터베이스 커넥션 풀은 커넥션 풀에서 사용하는 DataSource 객체를 네이밍 서비스를 이용해 컨테이너로부터 제공 받는다.

JNDI를 이용한 커넥션 풀의 설정은 톰캣의 context.xml에 설정하는 방법과 애플리케이션에서 별도의 context.xml를 작성해 설정하는 방법이 있다.

먼저 톰캣의 context.xml에 설정하는 방법에 대해 간단히 알아보고 앞으로 우리가 계속해서 사용할 META-INF/context.xml 파일에 커넥션 풀을 설정하는 방법에 대해 알아보자

#### 1) 톰캣 서버에 context.xml 파일에 커넥션 풀 설정하기

톰캣 서버의 context.xml 또는 server.xml 파일에 DBCP 정보를 설정하고 애플리케이션에서 JNDI를 이용하기 위해 web.xml 파일에 JNDI 객체를 참조하는 리소스 참조를 명시적으로 기술해야 한다. 우리는 이클립스를 사용하므로 이클립스에 Server로 등록된 톰캣 서버의 context.xml에 아래와 같이 DBCP 정보를 리소스로 설정하면 된다. 또한 server.xml 파일에서 <Context> 요소 안에 <Resource /> 부분을 추가하면 된다.

<Context> <- 톰캣 서버의 context.xml 파일의 루트 태그

```
<Resource name="jdbc/bbsDBPool"
    auth="Container"
    type="javax.sql.DataSource"
    driverClassName="oracle.jdbc.driver.OracleDriver"
    url="jdbc:oracle:thin:@localhost:1521:xe"
    username="hr"
    password="hr"
    factory="org.apache.tomcat.dbcp.dbcp.BasicDataSourceFactory"/>
```

```

        maxActive="10"
        maxIdle="5" />
</Context>
<!--
    톰캣에 기본 내장되어 있는 DBCP를 사용할 경우 DBCP Factory 클래스 지정
    이클립스에 설치된 톰캣 서버의 context.xml 또는 server.xml에 Context를
    설정하는 경우 factory 클래스는 아래와 같이 지정하면 된다.
-->
<!-- factory="org.apache.tomcat.dbcp.dbcp.BasicDataSourceFactory" -->

```

## 2) 애플리케이션의 META-INF/context.xml 파일에 커넥션 풀 설정하기

웹 애플리케이션 마다 별도의 context.xml 파일을 만들어 DBCP 정보를 설정하는 방식으로 이 파일을 이클립스에서 작성한 프로젝트의 webapp/META-INF 폴더에 저장한다. 애플리케이션에서 JNDI를 이용하기 위해 web.xml에 JNDI 객체를 참조하는 리소스 참조를 명시적으로 기술해야 하지만 이 리소스 참조는 서블릿 3.0부터 필수가 아닌 선택사항으로 바뀌었다.

톰캣 서버에 커넥션 풀을 설정하는 방식은 톰캣 서버가 관리하는 모든 애플리케이션에서 사용할 수 있는 JNDI 방식의 Connection Pool을 설정하는 것이라면 각각의 애플리케이션에서 커넥션 풀을 설정하는 방식은 각 애플리케이션 마다 다른 JNDI 방식의 Connection Pool을 사용하는 것이라 할 수 있다. 별도의 context.xml 파일에 아래와 같이 DBCP 정보를 설정하면 된다.

```

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

```

## 3) web.xml 파일에서 리소스 참조 설정하기

톰캣 서버의 context.xml 파일이나 별도의 context.xml 파일을 작성해 DBCP관련 정보를 설정한 후 web.xml 파일에 JNDI 연결에 필요한 리소스 참조를 <web-app>루트 태그 아래에 다음과 같이 설정한다. <description>과 <res-type>은 생략할 수 있다.

**web.xml에서 DBCP의 리소스 참조 설정은 Servlet 3.0 부터는 필수가 아닌 선택 사항이 되었다.**

```

<resource-ref>
    <description>DBCP Connection 정의</description>
    <res-ref-name>jdbc/bbsDBPool</res-ref-name>

```

```

    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>

```

#### 4) 소스 코드에서 context.xml 파일에 설정된 DBCP 정보를 읽어 커넥션 얻기

```

Context initContext = new InitialContext();
Context envCtx = (Context) initContext.lookup("java:/comp/env");
DataSource ds = (DataSource) envCtx.lookup("jdbc/bbsDBPool");
Connection conn = ds.getConnection();

```

위의 코드를 간단히 설명하면 InitialContext 객체를 생성해서 "java:/comp/env" 네임스페이스에 기술된 이름을 찾아 Context 객체를 구하고 이 Context 객체를 이용해 "jdbc/bbsDBPool" 이라는 이름을 가진 DBCP에 접근할 수 있는 DataSource 객체를 통해 데이터베이스 커넥션을 대여 받는다. 위의 코드를 아래와 같이 한 번에 작성해도 DataSource 객체를 얻을 수도 있다.

```
DataSource ds = (DataSource) init.lookup("java:comp/env/jdbc/bbsDBPool");
```

위에서 "java:/comp/env"는 JNDI 방식에서 이미 정해진 기본 네임스페이스고 "jdbc/bbsDBPool"은 기본 네임스페이스 안에서 사용하는 DBCP 이름으로 프로그래머가 임의로 지정할 수 있다.

## 6.5 DBCP를 활용한 게시판 구현하기

웹 애플리케이션에서 효율적으로 DB에 접근하기 위해서 JNDI 방식의 DBCP를 사용하는 방법에 대해 알아보았다. 이제 본격적으로 JNDI 방식의 DBCP를 활용해서 BoardDao 클래스 작성하고 게시글 리스트, 게시글 상세보기, 게시글쓰기, 수정하기, 삭제하기 기능을 제공하는 웹 애플리케이션을 구현해 보자.

### ▶ 라이브러리 참조하기

- webapp/WEB-INF/lib에 아래 라이브러리 추가할 것  
commons-dbc2-2.8.0.jar, commons-pool2-2.10.0.jar,  
commons-logging-1.2.jar, ojdbc6.jar,  
taglibs-standard-impl-1.2.5.jar, taglibs-standard-spec-1.2.5.jar

### ▶ DBCP Resource 설정

- webapp/META-INF/context.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<Context>
    <Resource name="jdbc/bbsDBPool"
        auth="Container"

```

```

    type="javax.sql.DataSource"
    driverClassName="oracle.jdbc.driver.OracleDriver"
    url="jdbc:oracle:thin:@localhost:1521:xe"
    username="hr"
    password="hr"
    factory="org.apache.commons.dbcp2.BasicDataSourceFactory"
    maxActive="10"
    maxIdle="5" />
</Context>
<!--
    톰캣에 기본 내장되어 있는 DBCP를 사용할 경우 DBCP Factory 클래스 지정
    이클립스에 설치된 톰캣 서버의 context.xml 또는 server.xml에 Context를
    설정하는 경우 factory 클래스는 아래와 같이 지정하면 된다.
-->
<!-- factory="org.apache.tomcat.dbcp.dbcp.BasicDataSourceFactory" -->

```

### 6.5.1 게시 글 리스트 보기

#### ▶ DAO(Data Access Object) 클래스

기존의 com.jspstudy.ch06.dao 패키지에 DBCP를 사용해 DB와 연동하는 BoardDao 클래스를 새로 만들고 아래 코드를 참고해 게시 글 리스트를 읽어오는 메서드를 작성하자.

#### - com.jspstudy.ch06.dao.BoardDao

// JNDI 방식의 DBCP를 활용한 DAO(Data Access Object) 클래스

```
public class BoardDao {
```

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

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

```
    private Connection conn;
```

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

```
    private PreparedStatement pstmt;
```

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

```
    private ResultSet rs;
```

```
    /* DataSource 객체는 데이터 원본과 연결할 수 있도록 지원하는 객체
```

```
    * JNDI 방식으로 DBCP를 찾아 DBCP에서 Connection 객체를 대여하는 객체
```

```
    */
```

```
    private static DataSource ds;
```

```
    /* 기본 생성자가 호출될 때 마다 ConnectionPool에 접근해 Connection 객체를
```

```
    * 얻어 올 수 있는 DataSource 객체를 생성한다.
```

```
    */
```

```
    public BoardDao() {
```



```

try {
    /* 1. 자바 네이밍 서비스를 사용하기 위해
     * javax.naming 패키지의 InitialContext 객체를 생성한다.
     */
    Context initContext = new InitialContext();

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

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

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

}

// DB에 등록된 전체 게시 글을 읽어와 ArrayList로 반환하는 메서드
public ArrayList<Board> boardList() {

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

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

```

```

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

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

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

/* 7. 쿼리 실행 결과를 바탕으로 while문 안에서 하나의 게시 글을 저장할
 * Board 객체를 생성하고 이 객체에 하나의 게시 글 정보를 저장하고
 * Board 객체를 ArrayList에 저장한다.
 *
 * ResultSet 객체는 DB로 부터 읽어온 데이터에 접근하기 위해 테이블의
 * 행을 가리키는 cursor를 제공한다. 맨 처음 ResultSet 객체를 반환
 * 받으면 cursor는 첫 번째 행 바로 이전을 가리키고 있다. ResultSet의
 * cursor가 맨 마지막 행에 도달하면 while문을 빠져 나온다.
 *
 * ResultSet에는 다양한 데이터 타입에 대응하는 getter 메소드를
 * 지원하고 있으며 SELECT 문장에서 지정한 컬럼의 index 또는
 * 컬럼명으로 테이블의 필드 값을 가져올 수 있도록 getXxx() 메소드가
 * 오버로딩 되어 있어 index와 컬럼명 둘 다 사용이 가능하다.
 * 여기에 지정하는 index는 배열에서 사용되는 index의 개념이 아니라
 * 첫 번째 컬럼, 두 번째 컬럼과 같이 위치의 개념으로 1부터 시작된다.
 */
while(rs.next()) {

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

    /* ResultSet 객체의 getXXX() 메서드에 컬럼 위치에 대한 index 값을
     * 1부터 지정할 수도 있고 컬럼 이름을 지정해 데이터를 읽어 올 수 있다.
     */

```

```

        b.setNo(rs.getInt("no"));
        b.setTitle(rs.getString("title"));
        b.setWriter(rs.getString("writer"));
        b.setContent(rs.getString("content"));
        b.setRegDate(rs.getTimestamp("reg_date"));
        b.setReadCount(rs.getInt("read_count"));
        b.setPass(rs.getString("pass"));
        b.setFile1(rs.getString("file1"));

        boardList.add(b);
    }
} catch(SQLException e) {
    System.out.println("BoardDao - boardList() - SQLException");
    e.printStackTrace();

} finally {
    try {
        // 8. 사용한 ResultSet과 PreparedStatement를 종료한다.
        if(rs != null) rs.close();
        if(pstmt != null) pstmt.close();

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

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

} // end boardList();

```

## ▶ 게시 글 리스트 보기 요청을 처리하는 JSP

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

뷰를 구현하는 코드는 앞에서 BoardDao를 이용해 게시 글 리스트를 출력하는 예제와 동일하기 때문에 앞에서 작성한 boardList01.jsp를 복사하고 코드를 약간 수정해서 DB와 잘 연동되는지 테스트한 후에 Bootstrap을 적용해 뷰를 구현해 볼 것이다.

- webapp/board/boardList.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="com.jspstudy.ch06.dao.*, com.jspstudy.ch06.vo.*" %>
<%@ page import="java.util.*" %>
<!--
    JSP 표준 태그 라이브러리(JSTL)를 사용하기 위한 taglib 지시자
    http://jakarta.apache.org, http://tomcat.apache.org 에서
    다운로드 하여 WEB-INF/lib 폴더에 추가해야 표준 태그를 사용할 수 있다.
    이 페이지에서는 JSTL의 코어 라이브러리에 속한 <c:set> 태그를 사용하여
    pageContext, request, session, application 4개의 영역에
    속성으로 데이터를 저장하고 EL 식을 이용해 출력하는 기법을 소개하고 있다.
    JSTL의 코어 라이브러리는 말 그대로 JSTL의 가장 핵심적인 기능을 제공하는
    라이브러리로 프로그래밍 언어에서 일반적으로 제공하고 있는 변수 선언, 조건문,
    반복문에 해당하는 태그를 지원한다. 또한 익셉션, URL 저장, 데이터 출력과 관련된
    태그와 다른 JSP 페이지 호출(import, redirect)과 관련된 태그를 지원한다.
--%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<%
    // BoardDao 객체를 생성하고 게시 글 리스트를 읽어온다.
    BoardDao dao = new BoardDao();
    ArrayList<Board> bList = dao.boardList();
%>
<!--
    JSP 페이지에서 사용할 변수를 선언하고 초기 값을 설정하고 있다.
    <c:set> 태그로 변수를 선언할 때는 변수의 타입은 지정하지 않으며 var 속성에
    변수의 이름을 지정하고 value 속성에 변수의 초기 값을 필히 지정해야 한다.
    scope 속성에는 page, request, session, application 중 하나를
    지정할 수 있으며 생략 가능하다. 생략하게 되면 기본 값은 page로 지정된다.
    scope 속성을 지정하는 것에서 알 수 있듯이 여기에 선언한 변수는 EL식 안에서
    사용할 수 있고 스크립팅 요소에서는 사용할 수 없다. 즉 스크립팅 요소에서 선언한
    자바 코드의 변수가 되는 것이 아니라 위의 4개 영역에 속성으로 저장하는 방식인
    setAttribute()가 호출되어 scope에 지정한 영역에 속성으로 저장된다.
--%>
<!-- 아래에서 EL로 접근하기 위해서 pageContext 영역의 속성으로 저장 -->
<c:set var="bList" value="<%= bList %>" />
<!DOCTYPE html>
<html lang="ko">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title>게시 글 리스트</title>
        <link href="../bootstrap/bootstrap.min.css" rel="stylesheet" >
        <script src="../js/jquery-3.3.1.min.js"></script>
        <script src="../js/formcheck.js"></script>

```

```

</head>
<body>
  <div class="container">
    <!-- header -->
    <div class="row">
      <div class="col">
        여기는 메뉴 부분 다음에 추가함
      </div>
    </div>
    <!-- content -->
    <div class="row my-5 text-center">
      <div class="col">
        <h2 class="fs-3 fw-bold">게시 글 리스트</h2>
      </div>
    </div>
    <form name="searchForm" id="searchForm" action="#"
      class="row justify-content-center my-3">
      <div class="col-auto">
        <select name="type" class="form-select">
          <option value="title">제목</option>
          <option value="writer">작성자</option>
          <option value="content">내용</option>
        </select>
      </div>
      <div class="col-4">
        <input type="text" name="keyword" class="form-control"/>
      </div>
      <div class="col-auto">
        <input type="submit" value="검 색" class="btn btn-primary"/>
      </div>
    </form>
    <div class="row">
      <div class="col text-end">
        <a href="writeForm.jsp" class="btn btn-outline-success">글쓰기</a>
      </div>
    </div>
    <div class="row my-3">
      <div class="col">
        <table class="table table-hover">
          <thead>
            <tr class="table-dark">
              <th>NO</th>
              <th>제목</th>
              <th>작성자</th>
              <th>작성일</th>
            </tr>
          </thead>
        </table>
      </div>
    </div>
  </div>

```

```

        <th>조회수</th>
    </tr>
</thead>
<tbody class="text-secondary">
<!--
    표현언어(EL)를 사용해 내장객체의 속성에 저장할 때 사용한 속성 이름을 지정하면 내
    장객체의 속성에 저장된 데이터를 읽어 올 수 있다. 스크립팅요소를 사용하는 것에 비
    해 더 간단히 내장객체의 속성에 저장된 값을 읽을 수 있다.
    표현언어(EL)를 사용해 속성 이름을 지정하면 pageContext, request, session,
    application 4개의 영역에 저장된 속성을 작은 범위에서 큰 범위 순으로 검색하여
    지정한 이름의 속성에 대한 값을 얻어 올 수 있다. 지정한 속성 이름이 존재하지 않아
    도 NullPointerException은 발생하지 않고 다만 아무것도 출력되지 않는다.
    내장객체의 속성에 객체가 저장되어 있으면 내장객체의 속성 명과 dot 연자자(.)를
    사용해 객체의 프로퍼티(인스턴스 변수) 값을 읽어 올 수 있다. 표현언어(EL)로 객체의
    프로퍼티를 읽기 위해서는 그 객체의 클래스에 읽어 올 프로퍼티에 대한 getter
    메서드가 반드시 존재해야 한다. 그렇지 않으면 JasperException이 발생한다.
-->
<!--
    게시 글이 있는 경우 게시 글 수만큼 반복하면서 게시 글을 출력
-->
<c:if test="${ not empty bList }">
    <c:forEach var="b" items="{bList}">
        <tr>
            <td>${ b.no }</td>
            <!--
                반복문 안에서 한 행의 게시 글을 출력하면서
                게시 글 상세보기로 넘어갈 수 있도록 링크를 설정
            -->
            <td><a href="boardDetail.jsp?no=${b.no}" class="text-decoration-none
link-secondary">${ b.title }</a></td>
            <td>${ b.writer }</td>
            <td>${ b.regDate }</td>
            <td>${ b.readCount }</td>
        </tr>
    </c:forEach>
</c:if>
<!-- 게시 글이 없는 경우 -->
<c:if test="${ empty bList }">
    <tr>
        <td colspan="5" class="text-center">게시 글이 존재하지 않습니다.</td>
    </tr>
</c:if>
</tbody>
</table>
</div>

```

```

    </div>
<!-- footer -->
<div class="row">
    <div class="col">
        여기는 푸터 부분 다음에 추가함
    </div>
</div>
</div>
<script src="../bootstrap/bootstrap.bundle.min.js"></script>
</body>
</html>

```

## 6.5.2 게시 글 상세보기

### ▶ DAO(Data Access Object) 클래스에 메서드 추가

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

#### - com.jspstudy.ch06.dao.BoardDao

```

/* 게시 글 상세보기 요청 시 호출되는 메서드
 * no에 해당하는 게시 글 을 DB에서 읽어와 Board 객체로 반환하는 메서드
 */
public Board getBoard(int no) {

    String sqlBoard = "SELECT * FROM jspbbbs WHERE no=?";
    Board board = null;

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

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

```

```

    * Placeholder(?)와 같이 위치의 개념으로 시작 번호가 1부터 시작된다.
    **/
pstmt = conn.prepareStatement(sqlBoard);

/* 6. PreparedStatement 객체에 SELECT 쿼리의 Placeholder(?)와
 * 데이터를 맵핑 한다.
 */
pstmt.setInt(1, no);

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

/* 8. Board 객체를 생성하고 ResultSet으로 부터 데이터를 읽어
 * Board 객체의 각 프로퍼티에 값을 설정한다.
 * no가 Primary Key 이므로 no에 해당하는 게시 글을 DB에서 읽으면
 * 없거나 또는 하나의 게시 글 정보를 얻을 수 있으므로 if문을 사용했다.
 *
 * ResultSet 객체는 DB로 부터 읽어온 데이터에 접근하기 위해 테이블의
 * 행을 가리키는 cursor를 제공한다. 맨 처음 ResultSet 객체를 반환
 * 받으면 cursor는 첫 번째 행 바로 이전을 가리키고 있다. ResultSet의
 * cursor가 맨 마지막 행에 도달하면 while 문을 빠져 나온다.
 * 여기서는 결과가 많아야 한 행이므로 한 번 실행되고 if문을 빠져 나온다.
 *
 * ResultSet에는 다양한 데이터 타입에 대응하는 getter 메소드를
 * 지원하고 있으며 SELECT 문장에서 지정한 컬럼의 index 또는
 * 컬럼명으로 테이블의 필드 값을 가져올 수 있도록 getXxx() 메소드가
 * 오버로딩 되어 있어 index와 컬럼명 둘 다 사용이 가능하다.
 * 여기에 지정하는 index는 배열에서 사용되는 index의 개념이 아니라
 * 첫 번째 컬럼, 두 번째 컬럼과 같이 위치의 개념으로 1부터 시작된다.
 */
if(rs.next()) {
    board = new Board();

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

```



```

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

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

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

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

} // end getBoard(int no);

```

## ▶ 게시 글 상세보기 요청을 처리하는 JSP

- webapp/board/boardDetail.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="com.jspstudy.ch06.vo.*, com.jspstudy.ch06.dao.*" %>
<%--
    JSP 표준 태그 라이브러리(JSTL)를 사용하기 위한 taglib 지시자
    http://jakarta.apache.org, http://tomcat.apache.org 에서
    다운로드 하여 WEB-INF/lib 폴더에 추가해야 표준 태그를 사용할 수 있다.
    이 페이지에서는 JSTL의 코어 라이브러리에 속한 <c:set> 태그를 사용하여
    pageContext, request, session, application 4개의 영역에
    속성으로 데이터를 저장하고 EL 식을 이용해 출력하는 기법을 소개하고 있다.
    JSTL의 코어 라이브러리는 말 그대로 JSTL의 가장 핵심적인 기능을 제공하는
    라이브러리로 프로그래밍 언어에서 일반적으로 제공하고 있는 변수 선언, 조건문,
    반복문에 해당하는 태그를 지원한다. 또한 익셉션, URL 저장, 데이터 출력과 관련된
    태그와 다른 JSP 페이지 호출(import, redirect)과 관련된 태그를 지원한다.
--%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%
    /* 테이블에서 하나의 게시 글 정보를 읽어오기 위해서 boardList.jsp에서
    * 게시 글 상세보기 요청을 하면서 테이블에서 게시 글 하나를 유일하게 구분할 수 있는
    * 게시 글 번호를 요청 파라미터로 보냈기 때문에 이 게시 글 번호를 요청 파라미터로
    * 부터 읽어 BoardDao를 통해서 no에 해당하는 게시 글 정보를 읽을 수 있다.

```

```

*
* 아래에서 no라는 요청 파라미터가 없다면 NumberFormatException 발생
**/
String no = request.getParameter("no");

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

```

```

아래는 pageContext.setAttribute("board", board)와 동일한 코드이다.
--%>
<c:set var="board" value="<%= board %>" scope="page" />
<!DOCTYPE html>
<html lang="ko">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>게시 글 상세보기</title>
    <link href="../bootstrap/bootstrap.min.css" rel="stylesheet" >
    <script src="../js/jquery-3.3.1.min.js"></script>
    <script src="../js/formcheck.js"></script>
  </head>
  <body>
    <div class="container">
      <!-- header -->
      <div class="row">
        <div class="col">
          여기는 메뉴 부분 다음에 추가함
        </div>
      </div>
      <!-- content -->
      <div class="row my-5" id="global-content">
        <div class="col">
          <div class="row text-center">

```

[illegible]

```

        </tbody>
    </table>
</div>
</div>
<div class="row my-3">
    <div class="col text-center">
        <input class="btn btn-warning" type="button" id="detailUpdate" value="수정하기"
"/>
        &nbsp;&nbsp;&nbsp;<input class="btn btn-danger" type="button" id="detailDelete"
value="삭제하기" />
        &nbsp;&nbsp;&nbsp;<input class="btn btn-primary" type="button" value="목록보기"
onclick="location.href='boardList.jsp'"/>
    </div>
</div>
</div>
<!-- footer -->
<div class="row">
    <div class="col">
        여기는 푸터 부분 다음에 추가함
    </div>
</div>
</div>
<script src="../../bootstrap/bootstrap.bundle.min.js"></script>
</body>
</html>

```

## ▶ 여러 페이지에서 중복되는 header와 footer 페이지

앞에서 게시 글 리스트와 게시 글 상세보기를 작성할 때 화면 상단의 메뉴가 표시되는 header 부분과 화면 하단의 회사 정보 등이 표시되는 footer 부분은 여러 웹 페이지에서 공통적으로 표현되는 부분으로 여러 페이지에 걸쳐서 코드가 중복되는데 이렇게 중복되는 내용을 별도의 JSP 페이지로 작성하고(이런 것을 “모듈화”라고 함) 별도로 작성한 JSP 페이지의 내용이 필요한 곳에서 include 지시자를 사용해 그 JSP 페이지를 포함시켜서 화면을 구성할 수 있는데 이런 기법을 사용하면 중복 코드를 최소화 할 수 있다.

### - webapp/pages/header.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!-- header -->
<div class="row border-bottom border-primary">
    <div class="col-4">
        <p></p>

```

```

</div>
<div class="col-8">
  <div class="row">
    <div class="col">
      <ul class="nav justify-content-end">
        <li class="nav-item">
          <a class="nav-link" href="#">로그인-폼</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">로그인-모달</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="boardList.jsp">게시 글 리스트</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">회원가입</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">주문/배송조회</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">고객센터</a>
        </li>
      </ul>
    </div>
  </div>
  <div class="row">
    <div class="col text-end">로그인시 인사말 출력</div>
  </div>
</div>
</div>

```

#### - webapp/pages/footer.jsp

```

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

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

```

```
</div>
</div>
```

### 6.5.3 게시 글쓰기

#### ▶ 게시 글쓰기 폼 요청을 처리하는 JSP

- webapp/board/writeForm.jsp

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

```

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

```

## ▶ 게시 글쓰기 폼 유효성 검사 스크립트

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

### - webapp/js/formcheck.js

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

```

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

```

```

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

```

## ▶ DAO(Data Access Object) 클래스에 메서드 추가

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

### - com.jspstudy.ch06.dao.BoardDao

```

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

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

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

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

```



```

    * Placeholder(?)와 같이 위치의 개념으로 시작 번호가 1부터 시작된다.
    **/
    pstmt = conn.prepareStatement(sqlInsert);

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

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

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

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

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

} // end insertBoard(Board board);

```

## ▶ 게시 글 등록하기 요청을 처리하는 JSP

게시 글쓰기 폼으로부터 들어오는 게시 글 등록 요청을 처리하는 JSP 페이지

- webapp/board/writeProcess.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="com.jspstudy.ch06.vo.*, com.jspstudy.ch06.dao.*" %>
<%

```

```

//POST 요청 방식의 문자셋 처리
request.setCharacterEncoding("utf-8");

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

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

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

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

```

```

    **/
    response.sendRedirect("boardList.jsp");
%>

```

#### 6.5.4 게시 글 수정하기

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

#### ▶ 게시 글 수정 폼 요청에 사용할 숨김 form 추가

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

##### - webapp/board/boardDetail.jsp에 아래 코드를 추가

... 중략 ...

```

<!-- content -->
<div class="row my-5" id="global-content">
  <div class="col">
    <%--

```

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

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

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

```

--%>
<form name="checkForm" id="checkForm">
  <input type="hidden" name="no" id="no" value="{ board.no }"/>

```

```

        <input type="hidden" name="pass" id="rPass" />
    </form>
    <div class="row text-center">
        <div class="col">
            <h2 class="fs-3 fw-bold">게시 글 상세보기</h2>
        </div>
    </div>

```

... 중략 ...

## ▶ 게시 글 상세 페이지에서 게시 글 수정 폼 요청 스크립트

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

### - webapp/js/formcheck.js

```

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

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

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

```

## ▶ DAO(Data Access Object) 클래스에 메서드 추가

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

### - com.jspstudy.bbs.dao.BoardDao

```

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

```

```

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

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

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

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

    /* 8. no가 Primary Key 이므로 no에 해당하는 게시 글을 DB에서 읽으면
     * 없거나 또는 하나의 게시 글 정보를 얻을 수 있으므로 if문을 사용했다.
     *
     * ResultSet 객체는 DB로 부터 읽어온 데이터에 접근하기 위해 테이블의
     * 행을 가리키는 cursor를 제공한다. 맨 처음 ResultSet 객체를 반환
     * 받으면 cursor는 첫 번째 행 바로 이전을 가리키고 있다. ResultSet의
     * cursor가 맨 마지막 행에 도달하면 while문을 빠져 나온다.
     * 여기서의 결과가 많아야 한 행이므로 한 번 실행되고 if문을 빠져 나온다.
     *
     * ResultSet에는 다양한 데이터 타입에 대응하는 getter 메소드를
     * 지원하고 있으며 SELECT 문장에서 지정한 컬럼의 index 또는
     * 컬럼명으로 테이블의 필드 값을 가져올 수 있도록 getXxx() 메소드가
     * 오버로딩 되어 있어 index와 컬럼명 둘 다 사용이 가능하다.
     * 여기에 지정하는 index는 배열에서 사용되는 index의 개념이 아니라

```

```

        * 첫 번째 컬럼, 두 번째 컬럼과 같이 위치의 개념으로 1부터 시작된다.
        **/
        if(rs.next()) {
            isPass = rs.getString(1).equals(pass);
        }
    } catch(SQLException e) {
        System.out.println("BoardDao - isPassCheck() : SQLException");
        e.printStackTrace();
    } finally {
        try {
            // 9. 사용한 ResultSet과 PreparedStatement 객체를 닫는다.
            if(rs != null) rs.close();
            if(pstmt != null) pstmt.close();

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

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

} // end isPassCheck();

```

## ▶ 게시 글 수정 폼 요청을 처리하는 JSP

- webapp/board/updateForm.jsp

```

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

    /* BoardDao 객체를 생성하고 DB에서 게시 글 번호와 사용자가 입력한 게시 글
    * 비밀번호가 맞는지를 체크하여 맞으면 게시 글 번호에 해당하는 게시 글을 읽어온다.

```

```

    **/
    BoardDao dao = new BoardDao();
    int no = Integer.parseInt(sNo);
    boolean isPassCheck = dao.isPassCheck(no, pass);

    if(isPassCheck) {

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

```

```

        placeholder="작성자를 입력해 주세요" value="<%= board.getWriter() %>">
    </div>
    <div class="col-4 ">
        <label for="pass" class="form-label">비밀번호</label>
        <input type="password" class="form-control" name="pass"
            id="pass">
    </div>
    <div class="col-8 offset-md-2">
        <label for="title" class="form-label">제 목</label>
        <input type="text" class="form-control" name="title"
            id="title" value="<%= board.getTitle() %>">
    </div>
    <div class="col-8 offset-md-2">
        <label for="content" class="form-label">내 용</label>
        <textarea class="form-control" name="content" id="content"
            rows="10"><%= board.getContent() %></textarea>
    </div>
    <div class="col-8 offset-md-2 text-center mt-5">
        <input type="submit" value="수정하기" class="btn btn-primary"/>
        &nbsp;&nbsp;&nbsp;<input type="button" value="목록보기"
            onclick="location.href='boardList.jsp'" class="btn btn-primary"/>
    </div>
</form>
</div>
</div>
<%@ include file="../pages/footer.jsp" %>
</div>
<script src="../bootstrap/bootstrap.bundle.min.js"></script>
</body>
<%
/* 게시 글 번호에 해당하는 비밀번호가 틀리면 비밀번호가 틀리다. 라고 알려주고
 * 브라우저의 history 객체를 이용해 바로 이전에 있었던 주소로 돌려보낸다.
 **/
} else {
%>
    <script>
        alert("비밀번호가 다릅니다.");
        history.back();
    </script>
<%=}%>
</html>

```

## ▶ 게시 글 수정 폼 유효성 검사 스크립트

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



- webapp/js/formcheck.js

// 게시 글 수정 폼 유효성 검사

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

▶ DAO(Data Access Object) 클래스에 메서드 추가

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

- com.jspstudy.bbs.dao.BoardDao

/\* 게시 글 수정 요청시 호출되는 메서드

\* 게시 글 수정 폼에서 수정하기 버튼이 클릭되면 게시 글을 DB에 수정하는 메서드  
\*\*/

```
public void updateBoard(Board board) {
```

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

```
    try {
```

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

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

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

```

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

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

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

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

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

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

```

```

    } catch(SQLException se) {}
}
} // end updateBoard(Board board);

```

## ▶ 게시 글 수정하기 요청을 처리하는 JSP

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

### - webapp/board/updateProcess.jsp

```

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

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

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

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

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

        /* 문자열을 보다 효율적으로 다루기 위해서 StringBuilder 객체를 이용해
        * 응답 데이터를 작성하고 있다. 아래에서는 비밀번호가 틀리면 사용자에게
        * 경고 창을 띄우기 위해서 자바스크립트의 alert() 함수를 실행하는 코드를
        * 응답 데이터로 작성하고 있다.
        */
        StringBuilder sb = new StringBuilder();

```

```

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

```

sb.append(" history.back();");
sb.append("</script>");

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

// 비밀번호가 맞으면 사용자가 게시 글 수정 폼에 입력한 데이터를 읽어온다.
title = request.getParameter("title");
writer = request.getParameter("writer");
content = request.getParameter("content");

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

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

/* 게시 글 수정이 완료된 후 response 내장객체의 sendRedirect() 메서드를
 * 이용해 게시 글 리스트로 Redirect 시킨다. response 내장객체의 sendRedirect()
 * 메서드는 요청한 자원이 다른 곳으로 이동되었다고 응답하면서 URL을 알려주고
 * 그 쪽으로 다시 요청하라고 응답하는 메소드이다. 브라우저가 요청한 콘텐츠가
 * 이동했으니 그 쪽으로 다시 요청하라고 응답 데이터로 웹 주소를 알려주면
 * 브라우저는 그 웹 주소로 다시 요청하게 되는데 이를 리다이렉션이라고 한다.
 *
 * Redirect 기법은 웹 브라우저를 새로 고침(F5) 했을 때 동일한 코드가 다시
 * 실행되면 문제가 될 수 있는 경우 클라이언트의 요청을 처리한 후 특정 URL로
 * 이동시키기 위해 사용하는 기법이다. 예를 들어 게시 글 수정하기 요청을 처리한
 * 후 Redirect 시키지 않으면 게시 글 수정 후에 사용자가 새로 고침(F5) 동작을
 * 하면 바로 이전에 수정한 게시 글 내용과 동일한 내용을 다시 DB에 수정하는 작업을
 * 하게 되는데 이렇게 되면 계속해서 같은 데이터를 수정하려고 하는 문제가 발생한다.
 * 이를 방지하기 위해서 게시 글 수정이 완료되면 게시 글 리스트(select 문은 반복

```

- \* 사용해도 중복된 데이터가 발생하지 않음)로 이동시키기 위해서 response
- \* 내장객체의 sendRedirect() 메소드를 사용해 게시 글 리스트의 URL을
- \* 웹 브라우저에게 응답하고 웹 브라우저는 응답 받은 URL로 다시 요청하도록 하는
- \* 것이다. 이렇게 게시 글 수정과 같이 DB 입력 작업이 연동되는 경우 사용자의
- \* 새로 고침(F5) 동작에 의해 동일한 요청이 다시 발생하여 DB에서 이미 수정된
- \* 게시 글을 수정하거나 SQLException을 발생 시킬 수 있어 Redirect 기법을
- \* 사용한다. 이외에 다른 사이트로 이동시킬 때 Redirect 기법을 사용 한다.

\*/

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

%>

### 6.5.5 게시 글 삭제하기

#### ▶ DAO(Data Access Object) 클래스에 메서드 추가

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

#### - com.jspstudy.bbs.dao.BoardDao

```
/* 게시 글 삭제 요청 시 호출되는 메서드
```

```
* no에 해당 하는 게시 글을 DB에서 삭제하는 메서드
```

```
*/
```

```
public void deleteBoard(int no) {
```

```
String sqlDelete = "DELETE FROM jspbbs WHERE no=?";
```

```
try {
```

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

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

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

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

```
*
```

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

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

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

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

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

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

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

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

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

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

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

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

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

```
*/
```

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

```

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

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

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

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

```

## ▶ 게시 글 상세 페이지에서 게시 글 삭제 요청 스크립트

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

### - webapp/js/formcheck.js

```

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

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

```

```

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

```

## ▶ 게시 글 삭제하기 요청을 처리하는 JSP

### - webapp/board/deleteProcess.jsp

```

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

    /* 게시 글을 삭제하기 위해서 boardDetail.jsp에서 게시 글 삭제 요청을
    * 하면서 테이블에서 게시 글 하나를 유일하게 구분할 수 있는 게시 글 번호를
    * 요청 파라미터로 보냈기 때문에 이 게시 글 번호를 요청 파라미터로부터 읽어
    * BoardDao를 통해서 게시 글 번호에 해당하는 게시 글을 DB에서 삭제할 수 있다.
    *
    * 아래에서 no라는 요청 파라미터가 없다면 NumberFormatException 발생
    */
    String sNo = request.getParameter("no");
    String pass = request.getParameter("pass");
    int no = Integer.parseInt(sNo);

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

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

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

```

```

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

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

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

```



## 6.6 파일업로드를 이용한 자료실 게시판 구현

이번 예제에서는 앞에서 구현한 게시판에 파일을 업로드 할 수 있는 기능을 추가해 보자.

게시판에 파일 업로드 기능이 추가되면 게시판을 통해 파일을 공유할 수 있는 기능을 제공할 수 있으므로 단순 글쓰기 기능만 있는 게시판에 비해 활용도를 높일 수 있다.

파일 업로드 기능은 사용자가 브라우저에서 선택한 파일을 서버에서 받아 서버의 특정 디렉터리나 DB에 저장하여 관리하는 것으로 서버에서 파일을 받아 저장하는 별도의 프로그램이 필요하다.

다시 말해 사용자가 선택한 파일을 브라우저가 서버로 전송하면 이 파일을 받아서 특정 디렉터리에 저장하는 별도의 라이브러리가 필요하다.

우리는 파일 업로드 기능을 직접 구현하지 않고 오픈소스로 제공되는 라이브러리를 사용할 것이다. 이 라이브러리가 제공하는 클래스를 이용해 게시 글쓰기에서 사용자가 업로드 한 파일을 특정 폴더나 데이터베이스에 저장하는 작업만 프로그래밍 하면 된다.

파일 업로드 라이브러리에는 apache 그룹의 commons FileUpload와 cos 라이브러리가 많이 사용된다. 교안에서는 게시 글쓰기에서 cos 라이브러리를 이용해 파일을 업로드 하는 방법에 대해 알아볼 것이다. commons FileUpload를 이용해 파일을 업로드 하는 방법에 대해서는 수업시간에 제공한 프로젝트 소스를 참고하기 바란다.

### 6.6.1 COS 라이브러리를 이용해 파일 업로드 구현하기

이번에는 COS 라이브러리를 이용해 파일을 업로드 하는 방법에 대해서 알아보자.

먼저 <http://www.servlets.com/cos>에 접속해 cos-26Dec2008.zip 파일을 다운로드 받아 압축을 풀고 cos.jar 파일을 웹 프로젝트의 WEB-INF/lib 폴더에 복사 한 후 아래 JSP 페이지를 작성해 COS 라이브러리를 이용해 파일을 업로드 하는 방법에 대해 알아보자.

참고로 이번에 제공되는 완성 소스는 이전의 DBCP 예제와 구분하기 위해서 webapp/upload 폴더를 만들고 그 안에 파일 업로드와 관련된 코드를 작성하였다.

#### ▶ webapp/WEB-INF/web.xml에 애플리케이션 초기화 파라미터 추가

```
<!--
    파일을 업로드할 폴더를 어플리케이션 초기화 파라미터로 설정
    아래와 같이 웹 어플리케이션의 여러 모듈에서 사용할 정보를 어플리케이션 초기화 파라미터로
    설정해 놓으면 이 프로젝트 안에 있는 여러 모듈에서 이 정보를 읽어서 사용할 수 있다.
-->
<context-param>
    <param-name>uploadDir</param-name>
    <param-value>/upload</param-value>
</context-param>
```

#### ▶ DAO(Data Access Object) 객체

게시 글 정보를 저장하는 BoardDao의 코드는 앞의 예제와 동일하기 때문에 교안에는 생략되었다. 업로드 된 파일 자체를 DB에 저장하는 것은 DB가 급격하게 커질 수 있으므로 파일은 서버의 특정 폴더에 저장하고 파일이 저장된 경로와 이름만 문자열로 DB에 저장하는 기법을 사용할 것이다. 또한 파일 업로드 구현은 앞에서 구현한 내용에서 파일 업로드와 관련된 내용이 추가되므로 앞에서의 코드와 거의 비슷하다. 그래서 새롭게 추가되거나 수정되는 부분은 빨간색 볼드체로 표현할 것이다.

## ▶ 게시 글쓰기 파일 업로드 하기(cos)

이번 예제는 기존에 구현한 게시판에 파일 업로드 기능을 추가하는 것이므로 기존의 게시 글쓰기에  
서 사용자가 파일을 선택할 수 있도록 파일 선택 상자를 추가해야 하지만 앞에서 구현할 당시에 이  
미 파일선택 상자는 추가된 상태이다.

이번 실습을 위해서 우리의 프로젝트에 webapp/fileUpload 폴더를 추가하고 webapp/board 폴더  
에서 boardList.jsp, boardDetail.jsp, writeForm.jsp, writeProcess.jsp 파일을 복사하여 가져온  
후에 writeForm.jsp 파일은 writeFormCos.jsp로 파일명을 변경하고 writeProcess.jsp 파일은  
writeProcessCos.jsp로 파일명을 변경하자.

## ▶ 게시 글 리스트 출력하는 JSP 페이지

게시 글 리스트 페이지도 앞에서 작성한 리스트 페이지와 거의 동일하고 게시 글쓰기 폼페이지 링  
크 부분만 아래와 같이 수정하면 된다.

- webapp/fileUpload/boardList.jsp

```
... 중략 ...

<div class="col-auto">
  <input type="submit" value="검 색" class="btn btn-primary"/>
</div>
</form>
<div class="row">
  <div class="col text-end">
    <a href="writeFormCos.jsp" class="btn btn-outline-success">글쓰기(cos)</a>
    <a href="writeFormCommons.jsp" class="btn btn-outline-success">글쓰기
(common)</a>
  </div>
</div>
<div class="row my-3">
  <div class="col">
    <table class="table table-hover">

... 중략...
```

## ▶ 게시 글쓰기 폼 요청을 처리하는 JSP 페이지

게시 글쓰기 폼 페이지도 앞에서 작성한 페이지와 거의 동일하며 파일 업로드를 위해서 <form> 태  
그 부분의 enctype과 파일 선택 상자가 추가되어 빨간색 볼드체로 표현하였다.

- webapp/fileUpload/writeFormCos.jsp

... 중략 ...

<!--

form은 사용자로 부터 데이터를 입력받기 위한 폼 컨트롤들로 구성된다. 사용자가 입력한 일반적인 데이터 즉 이름, 직업 등의 데이터는 문자열로 이루어진 데이터로 사용자가 submit 버튼을 클릭하게 되면 브라우저는 이 데이터를 서버로 보내기 전에 페이지의 문자 셋을 기준으로 인코딩을 한 후 서버로 전송하게 된다. 문자열로 이루어진 데이터를 브라우저가 인코딩 하는 방식은 application/x-www-form-urlencoded 방식으로 form의 기본 enctype 이다. 그래서 enctype을 생략할 수 있지만 파일과 같은 데이터를 전송하기 위해서는 위의 인코딩 방식으로는 서버로 전송할 수 없다. 파일은 문자열 형태의 데이터가 아닌 바이너리 형태의 데이터로 파일을 업로드 하기 위해서는 form 태그의 전송방식을 method 속성에 post를 지정하고 인코딩 타입(enctype)을 multipart/form-data로 지정해야 한다.

-->

```
<form name="writeForm" action="writeProcessCos.jsp" id="writeForm"
class="row g-3 border-primary" method="post" enctype="multipart/form-data">
  <div class="col-4 offset-md-2">
    <label for="writer" class="form-label">글쓴이</label>
    <input type="text" class="form-control" name="writer" id="writer"
      placeholder="작성자를 입력해 주세요">
  </div>
  <div class="col-4 ">
    <label for="pass" class="form-label">비밀번호</label>
    <input type="password" class="form-control" name="pass" id="pass" >
  </div>
  <div class="col-8 offset-md-2">
    <label for="title" class="form-label">제 목</label>
    <input type="text" class="form-control" name="title" id="title" >
  </div>
  <div class="col-8 offset-md-2">
    <label for="content" class="form-label">내 용</label>
    <textarea class="form-control" name="content" id="content"
rows="10"></textarea>
  </div>
  <div class="col-8 offset-md-2">
    <label for="file1" class="form-label">파 일</label>
    <input type="file" class="form-control" name="file1" id="file1" >
  </div>
  <div class="col-8 offset-md-2 text-center mt-5">
    <input type="submit" value="등록하기" class="btn btn-primary"/>
    &nbsp;&nbsp;&nbsp;<input type="button" value="목록보기"
      onclick="location.href='boardList.jsp'" class="btn btn-primary"/>
  </div>
</form>
```

```

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

```

## ▶ 게시 글쓰기 폼에서 요청한 파일 업로드를 처리하는 JSP 페이지

클라이언트가 서버로 전송한 파일을 받아서 파일 업로드 작업을 구현하는 페이지로 새롭게 추가되거나 변경된 부분이 많기 때문에 별도로 표현하지 않았으므로 처음부터 코드를 보면서 학습하면 된다.

### - webapp/fileUpload/writeProcessCos.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@page import="java.util.*, java.io.*"%>
<%@ page import="com.jspstudy.ch06.vo.*, com.jspstudy.ch06.dao.*" %>

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

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

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

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

    /* 파일 객체에 지정한 위치에 디렉토리가 존재하지 않거나
     * 파일 객체가 디렉토리가 아니라면 디렉토리를 생성한다.
     */
    if(! (parentFile.exists() && parentFile.isDirectory())) {
        parentFile.mkdir();
    }
}

```

```

        System.out.println("init - " + parentFile);
    }
%>
<%
/* cos 라이브러리를 이용한 파일 업로드 구현하기
 *
 * 1. MultipartRequest의 생성자 매개변수에 지정할 데이터를 설정
 *
 * 파일이 저장될 폴더의 로컬 경로를 구한다.
 */
String realPath = application.getRealPath(uploadDir);

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

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

/* 2. 파일 업로드를 처리할 MultipartRequest 객체 생성
 *
 * WEB-INF/lib/cos.jar 파일을 살펴보면 MultipartRequest 클래스는
 * com.oreilly.servlet 패키지에 위치하며 파일 업로드를 직접적으로 처리하는
 * 역할을 담당하는 클래스로 파일 업로드와 관련된 다양한 메소드를 정의하고 있다.
 * 생성자는 5개로 오버로딩 되어 있고 아래 생성자가 호출되도록 정의되어 있다.
 *
 * public MultipartRequest(HttpServletRequest request,
 *     String saveDirectory,
 *     int maxPostSize,
 *     String encoding,
 *     FileRenamePolicy policy) throws IOException {...}
 *
 * 이 생성자를 살펴보면 request, saveDirectory, maxPostSize는 필수사항으로
 * 이 매개변수가 null이거나 0보다 작다면 생성자 안에서 예외를 발생시킨다.
 *
 * request : MultipartRequest에 연결할 사용자의 요청 정보가 담긴 객체
 * saveDirectory : 업로드 된 파일을 저장할 서버의 디렉토리 지정
 * maxPostSize : 업로드 파일의 최대 크기 지정
 * encoding : 파일의 인코딩 방식 지정, 파일 이름이 한글일 경우 필히 utf-8 지정
 * policy : 사용자가 업로드 한 파일을 저장할 서버의 디렉토리에 현재 업로드 되는
 *     파일과 이름이 중복된 파일이 존재할 경우 현재 업로드 되는 파일의
 *     이름을 어떻게 변경할지에 대한 정책을 지정하는 매개변수 이다.
 *     일반적으로 new DefaultFileRenamePolicy()를 사용하며
 *     이 클래스는 abc.jpg 파일을 업로드 할때 이미 같은 이름의 파일이
 *     존재하면 자동으로 abc1.jpg와 같이 파일을 변경해 준다.
 *
 */

```

```

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

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

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

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

/* 사용자가 업로드한 파일 데이터 처리

```

```

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

// 업로드된 파일이 존재하면 파일명이 입력되고 존재하지 않으면 null이 입력된다.
board.setFile1(fileName);

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

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

/* 게시 글쓰기가 완료된 후 response 내장객체의 sendRedirect() 메서드를
* 이용해 게시 글 리스트로 Redirect 시킨다. response 내장객체의 sendRedirect()
* 메서드는 요청한 자원이 다른 곳으로 이동되었다고 응답하면서 URL을 알려주고
* 그 쪽으로 다시 요청하라고 응답하는 메소드이다. 요청한 콘텐츠가 다른 곳으로
* 이동했으니 그 쪽으로 다시 요청하라고 응답 데이터로 이동할 주소를 알려주면
* 브라우저는 그 웹 주소로 다시 요청하게 되는데 이를 리다이렉션이라고 한다.
*
* Redirect 기법은 웹 브라우저를 새로 고침(F5) 했을 때 동일한 코드가 다시
* 실행되면 문제가 될 수 있는 경우에 클라이언트의 요청을 모두 처리한 후 특정
* URL로 이동시키기 위해 주로 사용한다. 예를 들어 게시 글쓰기에 대한 요청을
* 처리한 후에 Redirect 시키지 않는다면 브라우저의 주소 표시줄에 게시
* 글쓰기에 대한 URL이 그대로 남아 있기 때문에 사용자가 브라우저를 새로
* 고침(F5) 하게 되면 바로 이전에 실행된 게시 글쓰기 작업이 반복 실행되어
* 중복된 데이터가 DB에 저장되는 문제가 발생할 수 있다.
* 이를 방지하기 위해서 게시 글쓰기가 완료되면 게시 글 리스트로 이동시키기
* 위해서 response 내장객체의 sendRedirect() 메서드를 사용해 게시 글
* 리스트의 URL을 웹 브라우저에게 응답하고 웹 브라우저는 응답 받은 URL로
* 다시 요청하도록 하는 것이다. 왜 게시 글 리스트로 리다이렉트 시켜야 하는가?
* 게시 글 리스트는 DB에서 데이터를 조회하는 쿼리인 SELECT 쿼리를 사용하기
* 때문에 이 쿼리가 실행되어도 데이터가 중복되어 저장되거나, 동일한 데이터를
* 반복적으로 수정 또는 삭제하려는 문제가 발생되지 않기 때문이다. 이렇게 게시
* 글쓰기와 같이 DB에서 데이터의 입력, 수정, 삭제 작업과 연동되는 경우에
* 사용자의 새로 고침(F5) 등으로 문제가 발생할 수 있기 때문에 Redirect를
* 사용한다. 이외에 다른 사이트로 이동시킬 때에도 Redirect 기법을 사용 한다.
**/
response.sendRedirect("boardList.jsp");
%>

```

## [연습문제 6-1] JNDI 방식의 DBCP를 이용해 상품관리 애플리케이션 구현하기

프로젝트를 새로 생성해서 JNDI 방식의 DBCP를 이용해 데이터베이스에 상품을 등록하고 관리하는 아래 그림과 같은 웹 애플리케이션을 구현해 보자. 이 애플리케이션은 아래 그림과 같이 상품 리스트보기, 상품 등록하기, 상품 수정하기, 상세정보 보기, 상품 삭제하기 기능을 제공해야 한다. 또한 데이터베이스에 접속하여 DB 작업을 전담하는 DAO(DataAccessObject) 클래스를 작성하여 각 JSP 페이지에서 사용하고 상품 정보를 DB에서 읽어 올 때는 상품 하나의 정보를 저장하는 VO(Value Object) 객체를 사용하여 구현해 보자.

### 상품 리스트

상품명	가격	상품코드	제조사
<a href="#">트롬세탁기22</a>	120000	LG-TROM001	엘지전자
<a href="#">통돌이세탁기</a>	569000	LG-TONG2001	엘지전자
<a href="#">꼬말스세탁기</a>	498000	LG-KKO1003	엘지전자
<a href="#">클라쎄세탁기</a>	567000	DWG-1234	동부대우전자
<a href="#">도서명</a>	12356	kkk2001	기타
<a href="#">라면</a>	3500	KM5200	기타

[상품 등록하기](#)

### 상품 상세 정보

통돌이세탁기



제조사 : 엘지전자

판매가 : 569000

상품코드 : LG-TONG2001

12Kg, 8가지코스, 다이아몬드글라스도어, 표준/울/섬세/급속/이불/조용조용/속속/내마음세

척, 통세척

[상품 등록하기](#) [상품 수정하기](#) [상품 삭제하기](#) [상품 리스트 보기](#)

### 상품 상세 정보

통돌이세탁기



웹 페이지 메시지


통돌이세탁기 상품을 삭제 하시겠습니까?

확인

취소

12Kg, 8가지코스, 다이아몬드글라스도어, 표준/울/섬세/급속/이불/조용조용/속속/내마음세

척, 통세척

[상품 등록하기](#) [상품 수정하기](#) [상품 삭제하기](#) [상품 리스트 보기](#)



## 상품 등록하기

상 품 명 :

판 매 가 :

상품코드 :

제 조 사 :  ▼

상품 상세설명 :

상품 이미지 :

## 상품 수정하기

상 품 명 :

판 매 가 :

상품코드 :

제 조 사 :  ▼

상품 상세설명 :

12Kg, 8가지코스, 다이아몬드글라스도  
 머, 표준/울/섬세/급속/미물/조용조용/속  
 쏙/내마음세척, 통세척

상품 이미지 :

## 7. JSP 에러처리

### 7.1 에러 페이지 지정과 에러 페이지 작성하기

- webapp/errorPage/pageDirectiveErrorPage.jsp

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

<%--
    이 JSP 페이지에서 에러가 발생하면 page 지시자의 errorPage 속성에
    지정된 JSP 페이지가 호출되어 그 페이지에서 에러를 처리할 수 있다.
--%>
<%@ page errorPage="pageDirectivesErrorPage.jsp" %>
<%
    // NullPointerException
    String str = null;
    System.out.println(str.equals("error"));

    // ArithmeticException
    // int num = 10 / 0;

    // ArrayIndexOutOfBoundsException
    // int[] nums = { 1, 2, 3, 4, 5 };
    // int num = nums[5];
%>
```

위의 예제를 실행하면 NullPointerException이 발생하여 page 지시자의 errorPage 속성에 지정한 아래의 pageDirectivesErrorPage.jsp가 호출된다.

- webapp/errorPage/pageDirectivesErrorPage.jsp

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

<%--
    page 지시자를 사용해 이 페이지가 에러를 처리하는 페이지임을 지정한다.
--%>
<%@ page isErrorPage="true" %>

<%--
    현재 페이지가 정상적으로
    처리되었다는 응답 상태 코드로
    200을 지정한다.
--%>
<% response.setStatus(200); %>
```

← → ↺ ⓘ localhost:8080/JSPStudyCh08/errorPage/pageDirectiveErrorPage.jsp

#### 죄송합니다.

고객님의 요청을 처리하는 과정에서 아래와 같은 예외가 발생 하였습니다.

에러 메시지 : null

에러 타입 : java.lang.NullPointerException

```

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>요청 처리중 에러 발생</title>
</head>
<body>
  <p>죄송합니다.<br/>
  고객님의 요청을 처리하는 과정에서 아래와 같은 예외가 발생 하였습니다.</p>
<!--
  page 지시자의 isErrorPage의 속성이 true로 설정되어 있어야
  exception 내장 객체를 사용할 수 있다.
  isErrorPage 속성을 true로 지정하지 않으면 기본 값은 false 이다.
--%>
  에러 메시지 : <%= exception.getMessage() %><br/>
  에러 타입 : <%= exception.getClass().getName() %><br/>
</body>
</html>

```

## 7.2 에러 코드별 에러 페이지 지정하기

### - webapp/errorCode/internalServerErrorProcess.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!--
  ArithmeticException을 발생 시켜 InternalServerError 유도하고 있다.
--%>
<%= 10 / 0 %>

```

### - webapp/errorCode/internalServerError.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!--
  현재 페이지가 정상적으로 처리되었다는 응답 상태 코드를 설정하고 있다.
  정상처리에 대한 HTTP 응답 코드는 200으로 아래와 같이 상수로 지정하면 된다.
  HttpServletResponse.SC_OK는 상수로 200으로 설정되어 있다.
--%>
<% response.setStatus(HttpServletResponse.SC_OK); %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>InternalServerError 처리</title>

```

```

<style>
  p { font-size: 12px; }
</style>
</head>
<body>
  <h2>서버 오류</h2>
  <div>
    <p>서버에서 요청을 처리하는 중 내부적인 에러가 발생하여,<br/>
    고객님의 요청을 정상적으로 처리할 수 없습니다.</p>
    <p><a href="#">고객센터 문의하기</a></p>
  </div>
</body>
</html>

```

## 서버 오류

서버에서 요청을 처리하는 중 내부적인 에러가 발생하여, 고객님의 요청을 정상적으로 처리할 수 없습니다.

[고객센터 문의하기](#)

### - webapp/errorCode/pageNotFoundError.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!--
  현재 페이지가 정상적으로 처리되었다는 응답 상태 코드를 설정하고 있다.
  정상처리에 대한 HTTP 응답 코드는 200으로 아래와 같이 상수로 지정하면 된다.
  HttpServletResponse.SC_OK는 상수로 200으로 설정되어 있다.
-->
<% response.setStatus(HttpServletResponse.SC_OK); %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>PageNotFound 에러 처리</title>
<style>
  p { font-size: 12px; }
</style>
</head>
<body>
  <h2>페이지 없음</h2>
  <div>
    <p>요청하신 페이지를 찾을 수 없습니다.</p>
    <p>방문하려는 페이지의 주소가 잘못 입력되었거나,<br/>
    페이지가 변경되거나 삭제되어 요청하신 페이지를 찾을 수 없습니다.</p>
    <p><a href="#">고객센터 문의하기</a></p>
  </div>
</body>
</html>

```

## 페이지 없음

요청하신 페이지를 찾을 수 없습니다.

방문하려는 페이지의 주소가 잘못 입력되었거나, 페이지가 변경되거나 삭제되어 요청하신 페이지를 찾을 수 없습니다.

[고객센터 문의하기](#)

- webapp/WEB-INF/web.xml에 아래 코드를 추가한다.

```
<error-page>
  <error-code>404</error-code>
  <location>/errorCode/pageNotFoundError.jsp</location>
</error-page>
<error-page>
  <error-code>500</error-code>
  <location>/errorCode/internalServerError.jsp</location>
</error-page>
```

위의 예제에서 요청 페이지가 존재하지 않거나 ArithmeticException이 발생하면 출력될 페이지를 지정하여 서버 내부에서는 에러가 발생하였지만 사용자에게는 의도한 대로 에러 코드에 맞는 페이지가 출력되었다. 그러므로 에러를 처리하는 JSP 페이지에서 아래와 같이 정상적인 처리라고 웹 브라우저에게 통보하는 응답 상태 코드를 설정해 주는 것이 적절한 처리라고 할 수 있다.

```
response.setStatus(HttpServletResponse.SC_OK);
```

## 7.3 예외 타입별 에러 페이지 지정하기

- webapp/exceptionType/exceptionTypeProcess.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@page import="com.jspstudy.ch07.Product"%>
<%
    // 정수를 0으로 나누어 ArithmeticException을 발생시킨다.
    int num = 1 / (int) 0.1f;

    // 객체를 생성하지 않고 사용하여 NullPointerException을 발생시킨다.
    //Product product = null;
    //out.println(product.getName());

    /* 숫자로 변환 불가능한 문자를 숫자 변환을 시도하여
     * NumberFormatException을 발생시킨다.
     */
    //String str = "13.0f";
    //int num = Integer.parseInt(str);

    /* 배열의 index 범위를 초과하는 index 접근으로
     * ArrayIndexOutOfBoundsException을 발생시킨다.
     */
    /* String[] names = { "홍길동", "이순신", "강감찬", "임꺽정", "장산박" };
    for(int i = 0; i <= names.length; i++) {
        out.println(names[i] + "<br/>");
```

```

    } */
%>

```

#### - webapp/exceptionType/arithmeticError.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%--
    exception 내장 변수를 사용하기 위해 isErrorPage 속성을 true로 설정하고 있다.
--%>
<%@ page isErrorPage="true" %>
<%--
    현재 페이지가 정상적으로 처리되었다는 응답 상태 코드를 설정하고 있다.
    정상처리에 대한 HTTP 응답 코드는 200으로 아래와 같이 상수로 지정하면 된다.
    HttpServletResponse.SC_OK는 상수로 200으로 설정되어 있다.
--%>
<% response.setStatus(HttpServletResponse.SC_OK); %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>ArithmeticException 처리</title>
<style>
    p { font-size: 12px; }
</style>
</head>
<body>
    <h2>서버 오류</h2>
    <div>
        <p>요청을 처리하는 중 아래와 같은 에러가 발생하였습니다.</p>
        <%--
            exception 내장 변수를 이용해 발생한 Exception의 이름을 출력하고 있다.
        --%>
        <p>에러 타입 : <%= exception.getClass().getName() %></p>
        <p><a href="#">고객센터 문의하기</a></p>
    </div>
</body>
</html>

```

### 서버 오류

요청을 처리하는 중 아래와 같은 에러가 발생하였습니다.

에러 타입 : java.lang.ArithmeticException

[고객센터 문의하기](#)

#### - webapp/exceptionType/nullPointerException.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%--
    exception 내장 변수를 사용하기 위해 isErrorPage 속성을 true로 설정하고 있다.

```

```
--%>
<%@ page isErrorPage="true" %>
<%--
    현재 페이지가 정상적으로 처리되었다는 응답 상태 코드를 설정하고 있다.
    정상처리에 대한 HTTP 응답 코드는 200으로 아래와 같이 상수로 지정하면 된다.
    HttpServletResponse.SC_OK는 상수로 200으로 설정되어 있다.
--%>
<% response.setStatus(HttpServletResponse.SC_OK); %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>NullPointerException 처리</title>
<style>
    p { font-size: 12px; }
</style>
</head>
<body>
    <h2>서버 오류</h2>
    <div>
        <p>요청을 처리하는 중 아래와 같은 에러가 발생하였습니다.</p>
        <%--
            exception 내장 변수를 이용해 발생한 Exception의 이름을 출력하고 있다.
        --%>
        <p>에러 타입 : <%= exception.getClass().getName() %></p>
        <p><a href="#">고객센터 문의하기</a></p>
    </div>
</body>
</html>
```

## 서버 오류

요청을 처리하는 중 아래와 같은 에러가 발생하였습니다.

에러 타입 : java.lang.NullPointerException

[고객센터 문의하기](#)

### - webapp/exceptionType/numberFormatError.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%--
    exception 내장 변수를 사용하기 위해 isErrorPage 속성을 true로 설정하고 있다.
--%>
<%@ page isErrorPage="true" %>
<%--
    현재 페이지가 정상적으로 처리되었다는 응답 상태 코드를 설정하고 있다.
    정상처리에 대한 HTTP 응답 코드는 200으로 아래와 같이 상수로 지정하면 된다.
    HttpServletResponse.SC_OK는 상수로 200으로 설정되어 있다.
--%>
<% response.setStatus(HttpServletResponse.SC_OK); %>
<!DOCTYPE html>
```

```

<html>
<head>
<meta charset="UTF-8">
<title>NumberFormatException 처리</title>
<style>
  p { font-size: 12px; }
</style>
</head>
<body>
  <h2>서버 오류</h2>
  <div>
    <p>요청을 처리하는 중 아래와 같은 에러가 발생하였습니다.</p>
    <%--
      exception 내장 변수를 이용해 발생한 Exception의 이름을 출력하고 있다.
    --%>
    <p>에러 타입 : <%= exception.getClass().getName() %></p>
    <p><a href="#">고객센터 문의하기</a></p>
  </div>
</body>
</html>

```

## 서버 오류

요청을 처리하는 중 아래와 같은 에러가 발생하였습니다.

에러 타입 : java.lang.NumberFormatException

[고객센터 문의하기](#)

### - webapp/exceptionType/indexOutOfBoundsError.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"%>
<%--
  exception 내장 변수를 사용하기 위해 isErrorPage 속성을 true로 설정하고 있다.
--%>
<%@ page isErrorPage="true" %>
<%--
  현재 페이지가 정상적으로 처리되었다는 응답 상태 코드를 설정하고 있다.
  정상처리에 대한 HTTP 응답 코드는 200으로 아래와 같이 상수로 지정하면 된다.
  HttpServletResponse.SC_OK는 상수로 200으로 설정되어 있다.
--%>
<% response.setStatus(HttpServletResponse.SC_OK); %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>IndexOutOfBoundsException 처리</title>
<style>
  p { font-size: 12px; }
</style>
</head>
<body>

```

## 서버 오류

요청을 처리하는 중 아래와 같은 에러가 발생하였습니다.

에러 타입 : java.lang.ArrayIndexOutOfBoundsException

[고객센터 문의하기](#)



```

<h2>서버 오류</h2>
<div>
  <p>요청을 처리하는 중 아래와 같은 에러가 발생하였습니다.</p>
  <%--
    exception 내장 변수를 이용해 발생한 Exception의 이름을 출력하고 있다.
  --%>
  <p>에러 타입 : <%= exception.getClass().getName() %></p>
  <p><a href="#">고객센터 문의하기</a></p>
</div>
</body>
</html>

```

- webapp/WEB-INF/web.xml에 아래 코드를 추가

```

<error-page>
  <exception-type>java.lang.ArithmeticException</exception-type>
  <location>/exceptionType/arithmeticError.jsp</location>
</error-page>
<error-page>
  <exception-type>java.lang.NullPointerException</exception-type>
  <location>/exceptionType/nullPointerException.jsp</location>
</error-page>
<error-page>
  <exception-type>java.lang.NumberFormatException</exception-type>
  <location>/exceptionType/numberFormatError.jsp</location>
</error-page>
<error-page>
  <exception-type>java.lang.ArrayIndexOutOfBoundsException</exception-type>
  <location>/exceptionType/indexOutOfBoundsError.jsp</location>
</error-page>

```

위의 exceptionTypeProcess.jsp 파일의 주석을 풀어가며 Exception 별로 테스트 해보자. 앞에서 알아본 여러 가지 에러 처리 방식에서 실행 우선순위는 page 지시자의 errorPage 속성에 지정한 에러 처리 페이지가 1순위로 적용되며 web.xml에서 exception-type에 지정한 Exception 타입과 동일한 에러가 발생 할 경우 2순위로 적용되고 web.xml에서 error-code에 지정한 페이지가 3순위로 적용된다. 마지막으로 에러 페이지가 지정되어 있지 않을 경우 요청을 처리하는 과정에서 에러가 발생하게 되면 톰캣의 에러 페이지가 보여 지게 된다.

## 8. 쿠키와 세션

### 8.1 쿠키(Cookie)

#### ▶ 쿠키(Cookie) 생성하기

- webapp/cookie/cookieCreation.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%
    // 쿠키 이름을 "id"로 쿠키 값을 "midas"로 지정해 Cookie 객체를 생성하고 있다.
    Cookie cookie = new Cookie("id", "midas");

    /* 쿠키의 유효기간을 3분으로 설정하고 있다.
     * 쿠키의 유효기간을 지정하지 않으면 브라우저가 실행되는 동안 유효하다.
     */
    cookie.setMaxAge(60 * 3);

    /* response 객체의 addCookie() 메서드를 이용해 쿠키를 응답 객체에 추가한다.
     * 아래와 같이 응답 데이터에 쿠키를 추가하면 웹 브라우저로 쿠키가 전달되고
     * 웹 브라우저는 이 쿠키를 받아서 사용자 컴퓨터의 메모리나 하드 디스크에 저장한다.
     * 쿠키는 하나의 도메인 당 50개, 쿠키 1개당 4KB 총 3000개를 저장할 수 있다.
     */
    response.addCookie(cookie);
    response.addCookie(new Cookie("name", "John"));
%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>쿠키 생성하기</title>
</head>
<body>
    쿠키 이름 : <%= cookie.getName() %><br/>
    쿠키 값 : <%= cookie.getValue() %><br/>
    쿠키 유효기간 : <%= cookie.getMaxAge() %>초<br/>
</body>
</html>
```

#### ▶ 쿠키를 변경하고 삭제하기 - 위의 “쿠키(Cookie) 생성하기” 실행 후 테스트

- webapp/cookie/cookieChangeRemove.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@page import="java.net.*"%>
<%
```

```

// request 내장 객체로 부터 모든 쿠키를 배열로 얻어온다.
Cookie[] cookies = request.getCookies();

// 쿠키가 존재하면 반복문 안에서 쿠키에 접근한다.
if(cookies != null) {

    for(Cookie c : cookies) {

        String name = c.getName();

        // id라는 쿠키 이름이 존재하면 쿠키의 값을 변경한다.
        if(name.equals("id")) {

            /* 기존에 존재하는 쿠키 이름으로 새로운 쿠키 값을 지정하여 쿠키를
             * 생성하고 쿠키의 유효기간을 5분으로 설정한 후 response 내장 객체에
             * 새로 생성한 쿠키를 추가한다. 만약 이미 기존에 존재하는 쿠키 이름으로
             * 쿠키를 추가하게 되면 같은 이름의 쿠키가 추가로 만들어 진다.
             */
            Cookie cookie = new Cookie(name, "cookie");
            cookie.setMaxAge(60*5);

            /* 브라우저는 웹 서버로 요청을 보낼 때 기본적으로 그 웹 서버(도메인)에
             * 속하는 모든 쿠키를 함께 보내는데 아래와 같이 setPath()를 통해서 새로
             * 생성한 쿠키에 경로를 지정하면 이 경로로 요청할 때만 쿠키를 전송한다.
             * 다시 말해 새로 생성한 쿠키에 아래와 같이 경로를 지정하면 웹 브라우저는
             * http://localhost:8080/JSPStudyCh08/cookie/ 로 요청을 할 때만
             * id라는 이름을 가진 쿠키를 웹 서버로 같이 전송한다.
             */
            cookie.setPath("/JSPStudyCh08/cookie/");
            response.addCookie(cookie);

            // name이라는 쿠키 이름이 존재하면 쿠키를 삭제한다.
            } else if(name.equals("name")) {

                /* 쿠키의 유효기간을 설정하는 setMaxAge() 메서드를 호출하면서
                 * 시간을 0으로 지정하면 쿠키의 유효기간이 0초가 되므로 이런 쿠키를
                 * 받은 웹 브라우저는 같은 이름의 쿠키를 사용자 컴퓨터에서 삭제한다.
                 */
                c.setMaxAge(0);
                response.addCookie(c);
            }
        }
    }
}

%>
<!DOCTYPE html>

```

```

<html>
<head>
<meta charset="UTF-8">
<title>쿠키 변경하고 삭제하기</title>
</head>
<body>
쿠키 변경과 삭제가 완료됨<br/>
<a href="cookieView.jsp">/JSPStudyCh08/cookie/로 쿠키 확인하기</a><br/><br/>
<a href=" ../cookieinfo/cookieView.jsp" >/JSPStudyCh08/cookieInfo/로 쿠키 확인하기</a>
</body>
</html>

```

#### - webapp/cookie/cookieView.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="java.net.*" %>
<!--

```

이 JSP 페이지를 webapp/cookie/ 폴더에 작성하고  
이 파일을 webapp/cookieinfo/에 복사 한 후 테스트 할 것

cookieChangeRemove.jsp에서 쿠키 이름이 id인 쿠키를 변경할 때  
cookie.setPath("/JSPStudyCh08/cookie/");를 지정했기 때문에  
/JSPStudyCh08/cookie/로 들어오는 요청은 브라우저가 쿠키를 같이  
전송하지만 /JSPStudyCh08/cookieinfo/로 들어오는 요청은 경로가  
다르기 때문에 브라우저가 쿠키를 같이 전송하지 않는다. 또한 name인  
쿠키는 c.setMaxAge(0);로 설정하였기 때문에 브라우저가 응답을  
받으면 해당 쿠키를 삭제하기 때문에 다음 요청부터 쿠키를 전송하지 않는다.

```
-->
```

```
<%
```

```
// request 내장 객체로 부터 모든 쿠키를 배열로 얻어온다.
```

```
Cookie[] cookies = request.getCookies();
```

```
// 쿠키가 존재하지 않으면 쿠키를 생성한다.
```

```
if(cookies != null) {
```

```
    for(Cookie c : cookies) {
```

```
        out.println(c.getName() + ", " + c.getValue() + "<br/>");
```

```
    }
```

```
}
```

```
%>
```

## 8.2 세션(Session)

### ▶ 세션 정보 출력하기

- webapp/session/sessionInformation.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="java.util.*, java.text.*" %>
<%
    // 세션 유효시간을 60초로 설정
    session.setMaxInactiveInterval(60);
    Calendar ca = Calendar.getInstance();
    SimpleDateFormat formatter = new SimpleDateFormat("yyyy년 MM월 dd일 HH:mm:ss");
%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>세션 정보 출력하기</title>
</head>
<body>
    <!-- 맨 처음 한 번은 새로운 세션이므로 true, 새로 고침 하면 false 이다. -->
    새로운 세션 여부 : <%= session.isNew() %><br/>

    <!-- 세션 ID는 서버에서 유일한 세션 ID 값을 갖는다. -->
    세션 ID : <%= session.getId() %><br/>
<%
    ca.setTimeInMillis(session.getCreationTime());
%>
    세션 생성 시간 : <%= String.format(
        "%TY년 %Tm월 %Td일 %TT", ca, ca, ca, ca) %><br/>
<%
    ca.setTimeInMillis(session.getLastAccessedTime());
%>
    <!-- 브라우저를 새로 고침 할 때 마다 마지막 접근 시간은 바뀐다. -->
    마지막 접근 시간 : <%= formatter.format(ca.getTime()) %><br/>
    세션 유효 시간 : <%= session.getMaxInactiveInterval() %><br/>
</body>
</html>
```

새로운 세션 여부 : true  
세션 ID : C22801D5B335C6269220C907595FC6B1  
세션 생성 시간 : 2021년 08월 01일 03:51:37  
마지막 접근 시간 : 2021년 08월 01일 03:51:37  
세션 유효 시간 : 60

### ▶ 세션 속성을 이용한 회원가입 구현하기

- webapp/session/sessionAttributeFrom01.jsp

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

[illegible]

- webapp/session/sessionAttributeFrom02.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@page import="com.jspstudy.ch08.Member"%>
<%
    request.setCharacterEncoding("utf-8");
```

```
String name = request.getParameter("name");
String id = request.getParameter("id");
String pass = request.getParameter("pass");
String phone1 = request.getParameter("phone1");
String phone2 = request.getParameter("phone2");
String phone3 = request.getParameter("phone3");
String phone = phone1 + "-" + phone2 + "-" + phone3;
```







```

// 수정된 Member 객체를 세션 영역의 속성에 저장
session.setAttribute("member", member);
%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
<style type="text/css">
    table {
        border: 1px solid blue;
        border-collapse: collapse;
    }
    td {
        border: 1px solid blue;
        height: 30px;
    }
    .title {
        width: 100px;
        padding-left: 5px;
    }
    .content {
        width: 250px;
        padding-left: 5px;
    }
</style>
</head>
<body>
    <table>
        <tr><td colspan="2" style="text-align: center;
            height: 30px; line-height: 30px">
            <h3>회원 등록 완료</h3>
            아래와 같이 회원 등록이 완료 되었습니다.<br/></td></tr>
        <tr><td class="title">이 름</td>
            <td class="content">${ sessionScope.member.name }</td></tr>
        <tr><td class="title">아이디</td>
            <td class="content">${ sessionScope.member.id }</td></tr>
        <tr><td class="title">비밀번호</td>
            <td class="content">${ sessionScope.member.pass }</td></tr>
        <tr><td class="title">전화번호</td>
            <td class="content">${ sessionScope.member.phone }</td></tr>
        <tr><td class="title">성 별</td>
            <td class="content">
                ${ sessionScope.member.gender == null
                ? "입력않됨" : sessionScope.member.gender }
            </td></tr>
    </table>

```

회원 등록 완료	
아래와 같이 회원 등록이 완료 되었습니다.	
이 름	홍길동
아이디	midas
비밀번호	1234
전화번호	010-2345-6789
성 별	남자
나 이	23
관심분야	Android
메일수신	공지메일 받음
	공지메일 받지 않음
	공지메일 받지 않음
직 업	취업준비생

```

        </td></tr>
<tr><td class="title">나 이</td>
    <td class="content">${ sessionScope.member.age }</td></tr>
<tr><td class="title">관심분야</td>
    <td class="content">${ sessionScope.member.interest }</td></tr>
<tr><td rowspan="3" class="title">메일수신</td>
    <td class="content">
        ${sessionScope.member.noticeMail ? "공지메일 받음" : "공지메일 받지 않음"}
    </td></tr>
<tr><td class="content">
        ${ sessionScope.member.addMail ? "공지메일 받음" : "공지메일 받지 않음" }
    </td></tr>
<tr><td class="content">
        ${ sessionScope.member.infoMail ? "공지메일 받음" : "공지메일 받지 않음" }
    </td></tr>
<tr><td class="title">직 업</td>
    <td class="content">${ sessionScope.member.job }</td></tr>
</table>
</body>
</html>

```

## ▶ 세션을 이용한 사용자 로그인 유지하기

### - webapp/pages/header.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!-- header -->
<div class="row border-bottom border-primary">
    <div class="col-4">
        <p></p>
    </div>
    <div class="col-8">
        <div class="row mt1">
            <div class="col">
                <ul class="nav justify-content-end">
                    <li class="nav-item">
                        <a class="nav-link" href='${ sessionScope.isLogin ?
                            "logout.jsp" : "loginForm.jsp" }'>
                            ${ sessionScope.isLogin ? "로그아웃" : "로그인" }</a>
                    </li>
                    <li class="nav-item">
                        <c:if test='${ not sessionScope.isLogin }' >
                            <a class="nav-link" href="#">회원가입</a>
                        </c:if>
                    </li>
                </ul>
            </div>
        </div>
    </div>
</div>

```

```

        <c:if test="${ sessionScope.isLogin }" >
            <a class="nav-link" href="#">정보수정</a>
        </c:if>
    </li>
    <li class="nav-item">
        <a class="nav-link" href="main.jsp">게시 글 리스트</a>
    </li>
    <li class="nav-item">
        <a class="nav-link" href="#">주문/배송조회</a>
    </li>
    <li class="nav-item">
        <a class="nav-link" href="#">고객센터</a>
    </li>
</ul>
</div>
</div>
<div class="row">
    <div class="col text-end">&nbsp;</div>
</div>
<div class="row">
    <div class="col pe-5 text-end text-primary">
        <c:if test="${ sessionScope.isLogin }">
            <span>안녕하세요 ${ sessionScope.id }님!</span>
        </c:if>
    </div>
</div>
</div>
</div>

```

#### - webapp/pages/footer.jsp

```

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

```

Copyright (c) 2023 JSP2U Corp. All right Reserved.

```

    </p>
</div>
</div>

```

- webapp/session/main.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html lang="ko">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title>세션을 이용한 사용자 로그인 유지하기</title>
        <link href="../bootstrap/bootstrap.min.css" rel="stylesheet" >
    </head>
    <body>
        <div class="container">
            <!-- header -->
            <%@ include file="../pages/header.jsp" %>
            <!-- content -->
            <div class="row my-5 text-center" id="global-content">
                <div class="col">
                    <div class="row">
                        <div class="col">
                            <h2 class="fs-3 fw-bold">게시 글 리스트</h2>
                        </div>
                    </div>
                    <form name="searchForm" id="searchForm" action="#"
                        class="row justify-content-center my-3">
                        <div class="col-auto">
                            <select name="type" class="form-select">
                                <option value="title">제목</option>
                                <option value="writer">작성자</option>
                                <option value="content">내용</option>
                            </select>
                        </div>
                        <div class="col-4">
                            <input type="text" name="keyword" class="form-control"/>
                        </div>
                        <div class="col-auto">
                            <input type="submit" value="검 색" class="btn btn-primary"/>
                        </div>
                    </form>
                    <div class="row">
                        <div class="col text-end">
                            <a href="#" class="btn btn-outline-success">글쓰기</a>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </body>
</html>
```

```

    </div>
</div>
<div class="row my-3">
  <div class="col">
    <table class="table table-hover">
      <thead>
        <tr class="table-dark">
          <th>NO</th>
          <th>제목</th>
          <th>작성자</th>
          <th>작성일</th>
          <th>조회수</th>
        </tr>
      </thead>
      <tbody class="text-secondary">
        <tr>
          <td>200</td>
          <td><a href="#" class="text-decoration-none link-secondary">감사합니
다.</a></td>
          <td>회원1</td>
          <td>2023-04-27 05:44:32</td>
          <td>162</td>
        </tr>
        <tr>
          <td>199</td>
          <td><a href="#" class="text-decoration-none link-secondary">궁금한게 해
결 되었네요</a></td>
          <td>회원8</td>
          <td>2023-04-27 05:50:21</td>
          <td>77</td>
        </tr>
        <tr>
          <td>198</td>
          <td><a href="#" class="text-decoration-none link-secondary">저두 궁금했
는데</a></td>
          <td>회원7</td>
          <td>2023-04-27 05:44:32</td>
          <td>81</td>
        </tr>
        <tr>
          <td>197</td>
          <td><a href="#" class="text-decoration-none link-secondary">아 줄 바꿈
문제 해결 했습니다.</a></td>
          <td>관리자</td>
          <td>2023-04-27 04:58:45</td>

```

	35
--	----

	196	<a href="#">감사합니</a>
--	-----	----------------------

다.
  

	관리자	2023-04-27 03:40:43	16
--	-----	---------------------	----

	195	<a href="#">그러게요</a>
--	-----	----------------------

	회원3	2023-04-27 04:59:15	46
--	-----	---------------------	----

	194	<a href="#">회원이면 당</a>
--	-----	------------------------

연한 것을...
  

	회원3	2023-04-27 05:44:32	38
--	-----	---------------------	----

	13	<a href="#">별 말씀을 다</a>
--	----	-------------------------

하시네요
  

	회원1	2023-04-27 05:44:32	79
--	-----	---------------------	----

	192	<a href="#">당연하</a>
--	-----	---------------------

쥬~
  

	회원6	2023-04-27 05:44:32	9
--	-----	---------------------	---

	191	<a href="#">저두</a>
--	-----	--------------------

```

요~</a></td>
        <td>회원5</td>
        <td>2023-04-27 05:44:32</td>
        <td>15</td>
    </tr>
</tbody>
</table>
</div>
</div>
<!-- 페이지 네이션 -->
<div class="row">
    <div class="col">
        <nav aria-label="Page navigation">
            <ul class="pagination justify-content-center">
                <li class="page-item active" aria-current="page">
                    <span class="page-link">1</span>
                </li>
                <li class="page-item"><a class="page-link" href="#">2</a></li>
                <li class="page-item"><a class="page-link" href="#">3</a></li>
                <li class="page-item"><a class="page-link" href="#">4</a></li>
                <li class="page-item"><a class="page-link" href="#">5</a></li>
                <li class="page-item"><a class="page-link" href="#">6</a></li>
                <li class="page-item"><a class="page-link" href="#">7</a></li>
                <li class="page-item"><a class="page-link" href="#">8</a></li>
                <li class="page-item"><a class="page-link" href="#">9</a></li>
                <li class="page-item"><a class="page-link" href="#">10</a></li>
                <li class="page-item">
                    <a class="page-link" href="#">Next</a>
                </li>
            </ul>
        </nav>
    </div>
</div>
</div>
</div>
</div>
<!-- footer -->
<%@ include file="../pages/footer.jsp" %>
</div>
<script src="../bootstrap/bootstrap.bundle.min.js"></script>
</body>
</html>

```

200 x 100

로그인 회원가입 게시 글 리스트 주문/배송조회 고객센터

게시 글 리스트

제목

검색

글쓰기

NO	제목	작성자	작성일	조회수
200	감사합니다.	회원1	2023-04-27 05:44:32	162
199	궁금한게 해결 되었네요	회원8	2023-04-27 05:50:21	77
198	저두 궁금했는데	회원7	2023-04-27 05:44:32	81
197	아 줄 바꿈 문제 해결 했습니다.	관리자	2023-04-27 04:58:45	35
196	감사합니다.	관리자	2023-04-27 03:40:43	16
195	그렇게요	회원3	2023-04-27 04:59:15	46
194	회원이면 당연한 것을...	회원3	2023-04-27 05:44:32	38
13	별 말씀을 다하시네요	회원1	2023-04-27 05:44:32	79
192	당연하죠~	회원6	2023-04-27 05:44:32	9
191	저두요~	회원5	2023-04-27 05:44:32	15

1 2 3 4 5 6 7 8 9 10 Next

고객센터 전화주문:1234-5678 사업자등록번호 :111-11-123456 대표이사: 홍길동 통신판매업 서울 제 000000호  
 개인정보관리책임자:임직원 분쟁조정기관표시 : 소비자보호원, 전자거래분쟁중재위원회  
 Copyright (c) 2023 JSP2U Corp. All right Reserved.

## - webapp/session/loginForm.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html lang="ko">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>세션을 이용한 사용자 로그인 유지하기</title>
<link href="../bootstrap/bootstrap.min.css" rel="stylesheet" >
<link href="../css/member.css" rel="stylesheet" >
</head>
<body>
<div class="container">
<!-- header -->
<%@ include file="../pages/header.jsp" %>
<!-- content -->
<div class="row my-5" id="global-content">
<div class="col">
<form class="my-5" id="loginForm" action="loginProcess.jsp" method="post">

```



```

<h2 class="fw-bold">Member Login</h2>
<fieldset>
  <legend>Member Loin</legend>
  <div id="login">
    <p>
      <label for="userId" class="labelStyle">아이디</label>
      <input type="text" id="userId" name="id" />
    </p>
    <p>
      <label for="userPass" class="labelStyle">비밀번호</label>
      <input type="password" id="userPass" name="pass"/>
    </p>
  </div>
  <input type="submit" value="로그인" id="btnLogin" />
  <p id="btn1">
    <input type="checkbox" id="saveId" value="savedIdYes" />
    <label for="saveId">아이디저장</label>
    <input type="checkbox" id="secure" value="secureYes" />
    <label for="secure">보안접속</label>
  </p>
  <p id="btn2">
    <input type="button" value="회원가입" id="btnJoin" />
    <input type="button" value="아이디/비밀번호 찾기" id="btnSearch" />
  </p>
</fieldset>
</form>
</div>
</div>
<!-- footer -->
<%@ include file="../pages/footer.jsp" %>
</div>
<script src="../bootstrap/bootstrap.bundle.min.js"></script>
</body>
</html>

```

#### - webapp/css/member.css

```

@CHARSET "UTF-8";
/* 로그인 폼 */
#loginForm h2, #loginForm p,
#loginForm input, #loginForm label {
  margin: 0px;
  padding: 0px;
  font-family: "맑은 고딕",돋움;
  font-size: 12px;
}

```

```

}
#loginForm h2 {
    font-size: 40px;
    letter-spacing: -1px;
    color: #C8C8C8;
    text-align: center;
}
#loginForm legend {
    display: none;
}
#loginForm fieldset {
    width: 430px;
    margin: 10px auto;
    border: 5px solid #efefef;
    padding: 50px;
    border-radius: 15px;
}
#loginForm #login {
    float: left;
}
#loginForm label.labelStyle {
    width: 60px;
    display: block;
    float: left;
    font-weight: bold;
}
#loginForm #userId, #loginForm #userPass {
    width: 150px;
    border: 1px solid #999;
    margin-bottom: 5px;
    padding: 2px;
}
#loginForm #btnLogin {
    display: block;
    background-color: #FF6633;
    border-radius: 5px;
    border-style: none;
    color: #fff;
    width: 80px;
    height: 57px;
    position: relative;
    float: left;
    left: 10px;
    font-size: 13px;
    font-weight: bold;
}

```

```

    cursor: pointer;
}
#loginForm #btn1 {
    clear: both;
    margin-left: 60px;
    padding: 10px 0px;
}
#loginForm #btn1 label {
    font-size: 11px;
    vertical-align: middle;
}
#loginForm #btn1 input{
    height: 20px;
}
#loginForm #btn2 {
    margin-left: 60px;
}
#loginForm #btn2 input {
    background-color: #666;
    border-style: none;
    border-radius: 5px;
    color: #fff;
    height: 25px;
    padding: 5px 10px;
}

```

200 x 100

로그인 회원가입 게시 글 리스트 주문/배송조회 고객센터

## Member Login

아이디

로그인

비밀번호

☐ 아이디저장 ☐ 보안검색

회원가입

아이디/비밀번호 찾기

고객상담 전화주문:1234-5678 사업자등록번호 :111-11-123456 대표이사: 홍길동 통신판매업 서울 제 000000호

개인정보관리책임자:임격정 분쟁조정기관표시 : 소비자보호원, 전자거래분쟁중재위원회

Copyright (c) 2023 JSP2U Corp. All right Reserved.

- webapp/session/loginProcess.jsp

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

<%--

<c:set /> 태그는 scope에 지정한 내장객체에 속성을 생성하고 속성 값을 지정하는 데 사용하는 태그로 이 태그의 scope 속성에 지정할 수 있는 값은 속성에 데이터를 저장할 수 있는 page, request, session, application이 있다. scope 속성을 지정하지 않으면 기본적으로 page가 적용된다.

아래는 application 내장객체에 ADMIN\_ID라는 속성 이름으로 admin 이라는 값을 저장하게 된다. 즉 application.setAttribute("ADMIN\_ID", "admin");과 같은 역할을 수행한다.

--%>

```
<c:set var="ADMIN_ID" value="admin" scope="application" />
<c:set var="ADMIN_PASS" value="1234" scope="application" />
```

<%--

EL의 param 내장 객체를 이용해 로그인 폼에서 파라미터로 넘어온 id와 pass를 읽어 위에서 <c:set /> 태그로 application 영역에 속성으로 저장한 데이터와 비교해 맞으면 isLogin 변수에 true를 설정해 session 영역에 저장한다. 또한 로그인에 성공한 사용자 아이디도 session 영역에 저장했다.

EL을 사용하면 속성에 데이터를 저장할 수 있는 JSP의 내장객체인 pageContext, request, session, application 영역을 순서대로 검색하여 첫 번째 만나는 속성에 해당하는 데이터를 읽어온다. EL 자체에도 JSP의 각 내장객체에 대응하는 EL의 내장객체를 아래와 같이 지원하고 있다.

JSP 내장객체	->	EL 내장객체
pageContext	->	pageScope
request	->	requestScope
session	->	sessionScope
application	->	applicationScope

EL의 내장객체를 이용해 아래와 같이 해당 영역에서 데이터를 바로 읽어 올 수 있다.

--%>

```
<c:if test="${ param.id == applicationScope.ADMIN_ID
    && param.pass == applicationScope.ADMIN_PASS }" >
    <c:set var="isLogin" value="true" scope="session" />
    <c:set var="id" value="${ param.id }" scope="session" />
<!--
```

jstl이 제공하는 태그를 이용해 리다이렉트 하고 있다.  
response.sendRedirect()를 호출한 결과와 동일하다.

```

-->
<c:redirect url="main.jsp" />
</c:if>
<c:if test="${ not (param.id == applicationScope.ADMIN_ID
&& param.pass == applicationScope.ADMIN_PASS) }" >
<script>
    alert("아이디 또는 비밀번호가 맞지 않습니다.");
    history.back();
</script>
</c:if>

```

#### - webapp/session/logout.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
    /* invalidate() 메서드는 현재 세션을 종료하는 메서드로 이 메서드가 호출되면
    * 기존의 세션을 종료하고 새로운 세션이 시작되기 때문에 이전 세션 영역에
    * 저장된 모든 데이터가 삭제된다.
    **/
    session.invalidate();

    /*
    // 아래와 같이 세션에 저장된 개별 속성만 삭제 할 수 있다.
    session.removeAttribute("isLogin");
    session.removeAttribute("id");
    */
%>
<%--
    아래와 같이 jstl을 사용해서 세션에 저장된 개별 속성만 삭제할 수 있다.
    <c:remove var="isLogin" scope="session" />
    <c:remove var="id" scope="session" />
    --%>
<!--
    jstl이 제공하는 태그를 이용해 리다이렉트 하고 있다.
    response.sendRedirect()를 호출한 결과와 동일하다.
-->
<c:redirect url="main.jsp" />

```

## 게시 글 리스트

제목 ▼

검색

글쓰기

NO	제목	작성자	작성일	조회수
200	감사합니다.	회원1	2023-04-27 05:44:32	162
199	궁금한게 해결 되었네요	회원8	2023-04-27 05:50:21	77
198	저두 궁금했는데	회원7	2023-04-27 05:44:32	81
197	아 줄 바꿈 문제 해결 했습니다.	관리자	2023-04-27 04:58:45	35
196	감사합니다.	관리자	2023-04-27 03:40:43	16
195	그러게요	회원3	2023-04-27 04:59:15	46
194	회원이면 당연한 것을...	회원3	2023-04-27 05:44:32	38
13	별 말씀을 다하시네요	회원1	2023-04-27 05:44:32	79
192	당연하죠~	회원6	2023-04-27 05:44:32	9
191	저두요~	회원5	2023-04-27 05:44:32	15

[1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [Next](#)

## 9. JSP 표준 액션태그

JSP 표준 액션태그(액션태그라고도 함)는 JSP 페이지를 구성하는 기본요소 중의 하나로 자바 소스코드를 대체하는 역할을 하는 태그이다. JSP 페이지에서 표준액션 태그를 사용하면 코드의 가독성을 높일 수 있고 같은 기능을 스크립트릿을 사용해 자바코드로 기술하는 것 보다 코드의 양을 많이 줄일 수 있다. 또한 JSTL(Java Standard Tag Library)은 커스텀 태그 중에서 많이 사용하는 것을 모아서 사용하는 태그로 별도의 라이브러리를 프로젝트에 추가해야 되지만 표준 액션태그는 JSP 페이지의 기본적인 구성요소이므로 별도의 라이브러리 추가 없이 바로 사용이 가능하다.

### 9.1 <jsp:forward>를 이용해 포워딩하기

- webapp/forward/jspForward.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <h1>회원 가입</h1>
    <form action="jspForwardProcess.jsp">
        이 름 : <input type="text" name="name" /><br/>
        나 이 : <input type="text" name="age" /><br/>
        성 별 : <input type="radio" name="gender" value="남성" />남성
               <input type="radio" name="gender" value="여성" />여성<br/>
        <input type="reset" value="다시쓰기" />
        <input type="submit" value="전송하기" />
    </form>
</body>
</html>
```

#### 회원 가입

이 름 :	<input type="text" value="홍길동"/>
나 이 :	<input type="text" value="27"/>
성 별 :	<input checked="" type="radio"/> 남성 <input type="radio"/> 여성
<input type="button" value="다시쓰기"/> <input type="button" value="전송하기"/>	

- webapp/forward/jspForwardProcess.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%
    // <jsp:param> 태그에 대한 문자 셋 처리
    request.setCharacterEncoding("utf-8");

    // 폼에서 전송된 파라미터 읽기
    String name = request.getParameter("name");
    String age = request.getParameter("age");
```

```
String gender = request.getParameter("gender");
%>
<%--
```

<jsp:forward> 표준 액션 태그는 요청 처리를 다른 JSP 페이지로 넘길 때 사용한다. 이때 이동하는 페이지로 파라미터를 보내려면 <jsp:param> 태그를 사용해 파라미터를 전달할 수 있다. <jsp:param> 태그는 독립적으로 사용할 수 없고 <jsp:forward> 태그의 자식 태그로 사용할 수 있다. 또한 <jsp:param> 태그로 전달되는 파라미터는 request 내장객체의 setCharacterEncoding() 메소드에 지정한 문자 셋을 사용해 파라미터를 인코딩하므로 파라미터에 한글이 포함되는 경우 <jsp:param> 태그를 사용하기 이전에 setCharacterEncoding() 메소드를 사용해 적절한 문자 셋을 지정해야 한글 데이터가 깨지지 않는다.

forward 기법은 요청 처리와 화면 구현을 분리해서 처리하기 위해 서버에서 요청을 처리하는 과정에서 요청 처리가 완료되면 제어를 화면 구현하는 쪽으로 이동해 요청을 처리한 결과 데이터를 출력하여 최종적으로 응답하기 위해 사용하는 기법이다.

forward로 이동하는 페이지에 아래와 같이 파라미터로 데이터를 보내는 것 보다 request.setAttribute() 메서드를 이용해 request 영역의 속성에 데이터를 저장하여 View 페이지에서 사용할 수 있도록 하는 것이 EL을 사용해 데이터에 바로 접근할 수 있기 때문에 더 간결하고 효율적이다.

```
--%>
<jsp:forward page="jspForwardResult.jsp">
  <jsp:param name="name" value="<%= name %>" />
  <jsp:param name="age" value="<%= age %>" />
  <jsp:param name="gender" value="<%= gender %>" />
</jsp:forward>
```

#### - webapp/forward/jspForwardResult.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
  <h1>회원정보</h1>
  <%-- EL의 param 내장객체를 이용해 요청 파라미터를 읽을 수 있다. --%>
  이름 : ${ param.name }<br/>
  나이 : ${ param.age }<br/>
  성별 : ${ param.gender }<br/>
</body>
</html>
```

### 회원정보

이름 : 홍길동  
나이 : 27  
성별 : 남성



## 9.2 <jsp:include>를 이용해 동적으로다른 JSP 페이지 포함하기

- webapp/include/jspIncludeParam.jsp

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

```
<%
```

```
// <jsp:param> 태그에 대한 문자 셋 처리
```

```
request.setCharacterEncoding("utf-8");
```

```
String book = "백견불여일타 JSP&Servlet";
```

```
String image = "images/jsp&servlet.jpg";
```

```
int price = 27000;
```

```
%>
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

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

```
<title>Insert title here</title>
```

```
</head>
```

```
<body style="font-size: 12px;">
```

```
<h2>도서 정보 보기</h2>
```

```
<%--
```

<jsp:include> 표준 액션 태그를 사용해 다른 JSP 페이지를 실행 타임에 include 하고 <jsp:param> 표준 액션 태그를 사용해 포함될 JSP 페이지에 파라미터를 전달할 수 있다. <jsp:param> 태그는 독립적으로 사용할 수 없고 <jsp:include> 태그의 자식 태그로 사용할 수 있다. 또한 <jsp:param> 태그로 전달되는 파라미터는 request 내장객체의 setCharacterEncoding() 메소드에 지정한 문자 셋을 사용해 파라미터를 인코딩하므로 파라미터에 한글이 포함되는 경우 <jsp:param> 태그를 사용하기 이전에 setCharacterEncoding() 메소드를 사용해 적절한 문자 셋을 지정해야 한글 데이터가 깨지지 않는다.

include 기법은 다른 페이지를 현재 페이지에 포함 시키는 기능으로 여러 페이지를 하나의 페이지 처럼 묶어서 동작시킬 때 유용하게 사용할 수 있다. 일반적인 웹 페이지는 헤더, 푸터, 컨텐츠 부분으로 나누어지는데 헤더와 푸터는 거의 대부분 내용이 동일하기 때문 매 페이지의 상단과 하단이 중복된다. 이럴 때 각각 헤더, 푸터, 컨텐츠를 분리해서 따로 작성하고 실제로 웹 페이지가 실행될 때 하나로 합쳐져서 실행되도록 구현하면 최종적으로 하나의 완성된 HTML 문서만 클라이언트에 응답되기 때문에 여러 페이지에 중복될 수 있는 코드의 중복을 최소화 시킬 수 있는 보다 효율적인 웹 페이지 제작 기법이라 할 수 있다.

```
--%>
```

```
<jsp:include page="jspIncludeParamSub.jsp" >
```

```
<jsp:param name="book" value="<%= book %>" />
```

```
<jsp:param name="image" value="<%= image %>" />
```

```

    <jsp:param name="price" value="<%= price %>" />
</jsp:include>
</body>
</html>

```

#### - webapp/include/jspIncludeParamSub.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!-- 다른 JSP 페이지에 포함되므로 전체 HTML 태그는 필요 없다. --%>
<table>
    <tr>
        <td rowspan="3">
            </td>
        <td>&nbsp;</td>
        <td><h3>${ param.book }</h3></td>
    </tr>
    <tr>
        <td>&nbsp;</td>
        <td>정가 ${ param.price }원</td>
    </tr>
    <tr>
        <td>&nbsp;</td>
        <td>저자 : 성운정 | 출판일 : 2014년 07월 28일</td>
    </tr>
</table>

```

### 9.3 include 지시자를 이용해 JSP 페이지 포함하기

이번 예제는 앞에서 작성한 “<jsp:include> 액션태그를 이용해 다른 JSP 페이지 포함하기” 예제를 include 지시자를 이용해 구현한 예제이다. 이렇게 include 지시자를 이용해 다른 JSP 페이지를 포함하면 jspIncludeDirective.jsp 페이지가 서블릿 클래스로 변환될 때 jspIncludeDirectiveSub.jsp를 컴파일 타임에 포함하게 되므로 자바 소스 코드가 하나만 생성된다.

예제를 실행하여 테스트 해 본 후에 아래의 jspIncludeDirective.jsp 파일이 서블릿 컨테이너에 의해서 서블릿 클래스 코드로 변환된 jspIncludeDirective\_jsp.java 파일의 자바소스 코드를 확인해 보면 jspIncludeDirectiveSub.jsp 파일의 내용이 포함되어 있는 것을 확인 할 수 있다.

<jsp:include> 액션태그를 사용한 동적 페이지 포함은 request 내장객체의 속성을 사용하거나 액션태그의 파라미터를 사용해 포함될 페이지에 데이터를 전송하였으나 include 지시자를 이용한 JSP 페이지의 포함은 포함될 페이지에서 부모 페이지의 변수를 그대로 사용하였다. 포함되는 JSP에서 부모 페이지의 스크립틀릿에서 사용된 변수를 그대로 사용할 수 있는 이유는 서브 페이지가 실행 타임에 포함되는 것이 아니라 부모 페이지가 서블릿 클래스 코드로 변환 될 때 서브 페이지의 코드가 부모 페이지에 포함되어 변환되기 때문에 부모 페이지의 변수를 그대로 사용할 수 있다.

- webapp/include/jspIncludeDirective.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%
    request.setCharacterEncoding("utf-8");

    String book = "백견불여일타 JSP&Servlet";
    String image = "images/jsp&servlet.jpg";
    int price = 27000;
%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>include 지시자 사용하기</title>
</head>
<body style="font-size: 12px;">
    <h2>도서 정보 보기</h2>
    <%--
        include 지시자를 사용하여 컴파일 타임에 이 위치에 다른 JSP 페이지를 포함 할 수 있다.
    --%>
    <%@ include file="jspIncludeDirectiveSub.jsp" %>
</body>
</html>
```

- webapp/include/jspIncludeDirectiveSub.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%--
    include 지시자를 사용하면 이 JSP 페이지를 include 하는 부모 JSP 페이지가
    컴파일 될 때 자식 페이지인 현재 JSP 페이지가 포함되어 같이 컴파일되므로
    부모 JSP 페이지의 스크립틀릿에 선언된 변수를 별도의 선언이나 값 전달 없이
    아래 코드와 같이 자식 JSP 페이지의 표현식에서 그대로 사용할 수 있다.
    아래는 이클립스에서 에러로 표시되지만 문제없이 잘 실행된다.
--%>
<%-- 다른 JSP 페이지에 포함되므로 전체 HTML 태그는 필요 없다. --%>
<table>
    <tr>
        <td rowspan="3">
            </td>
            <td>&nbsp;</td>
            <td><h3><%= book %></h3></td>
        </tr>
    <tr>
```

```

        <td>&nbsp;</td>
        <td>정가 <%= price %>원</td>
    </tr>
</tr>
    <td>&nbsp;</td>
    <td>저자 : 성윤정 | 출판일 : 2014년 07월 28일</td>
</tr>
</table>

```

## 9.4 <jsp:useBean> 액션태그

- com.jspstudy.ch09.vo

// Java Bean 클래스

```

public class Student {

    private String name, phone, gender;
    private int age;

    public Student() { }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPhone() {
        return phone;
    }
    public void setPhone(String phone) {
        this.phone = phone;
    }
    public String getGender() {
        return gender;
    }
    public void setGender(String gender) {
        this.gender = gender;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}

```

- webapp/useBeans/jspBeanForm.jsp

[illegible]

- webapp/useBeans/jspBeanProcess.jsp

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

```

<head>
<meta charset="UTF-8">
<title>학생 등록 완료</title>
<style type="text/css">
    table {
        border: 1px solid blue;
        border-collapse: collapse;
    }
    td { border: 1px solid blue; height: 30px; }
    .title {
        width: 100px;
        padding-left: 5px;
    }
    .content {
        width: 200px;
        padding-left: 5px;
    }
</style>
</head>
<body style="font-size: 0.8em">
    <%--

```



학생 등록 완료	
이름	홍길동
성별	남자
나이	25
연락처	010-1234-5678

<jsp:useBean> 태그를 사용하게 되면 이 JSP 페이지가 서블릿 클래스로 변경될 때 \_jspService() 메소드 안에서 <jsp:useBean> 태그의 class 속성에 지정한 클래스 타입으로 id에 지정한 이름으로 변수가 선언되고 scope에 지정한 영역의 속성에 저장된다. 속성 저장되는 이름은 id에 지정한 이름이 사용된다.

<jsp:useBean id="student" class="com.jspstudy.ch09.vo.Student" >  
태그는 아래와 같은 자바 코드로 변경된다.

```

Student student = null;
if(pageContext.getAttribute("student") == null) {
    student = new Student();
    pageContext.setAttribute("student", student);
}

```

<jsp:setProperty> 태그의 property 속성의 값을 "\*"로 지정하면 폼으로 부터 전송된 요청 파라미터와 동일한 이름을 가진 자바 빈 클래스의 프로퍼티에 요청 파라미터의 값을 자동으로 설정할 수 있다. 만약 같은 이름을 가진 요청 파라미터가 없으면 에러는 발생하지 않고 자바 빈 클래스의 프로퍼티는 기본 값으로 설정된다. 만약 요청 파라미터의 이름과 자바 빈 클래스에 프로퍼티의 이름이 다를 경우 property의 속성에 자바 빈 클래스의 프로퍼티를 지정하고 param 속성에 요청 파라미터 이름을 지정하면 된다. 또한 EL(Expression Language)의 param 내장 객체를 이용하여 파라미터를 읽어 올 수도 있다

Bean 클래스의 속성이 age 이고 요청 파라미터가 age1 이라면 아래와 같은 <jsp:setProperty> 태그를 사용해 Bean 클래스의 age 값을 요청 파라미터로 들어오는 age1으로 설정한다.

```
<jsp:setProperty name="student" property="age" param="age1" />
```

위의 <jsp:setProperty> 태그는 아래와 같은 자바 코드로 변경된다.

```
String age1 = request.getParameter("age1");
student.setAge(age1)
--%>
<jsp:useBean id="student" class="com.jspstudy.ch07.beans.Student" />
<jsp:setProperty name="student" property="*" />
<jsp:setProperty name="student" property="age" param="age1" />
<jsp:setProperty name="student" property="phone"
    value="${ param.phone1 }-${ param.phone2 }-${ param.phone3 }"/>
<table>
<tr>
<td colspan="2" style="text-align: center;
    height: 30px; line-height: 30px">
<h3>학생 등록 완료</h3></td>
</tr>
<tr>
<td class="title">이 름</td>
<td class="content">
<%--
    아래 <jsp:getProperty> 태그는 다음과 같은 자바 코드로 변경된다.
    pageContext.getAttribute("student").getName();
--%>
<jsp:getProperty name="student" property="name" /></td>
</tr>
<tr>
<td class="title">성 별</td>
<td class="content">${ student.gender == null ? "선택않됨" :
    student.gender == "male" ? "남자" : "여자" }</td>
</tr>
<tr>
<td class="title">나 이</td>
<%--
    <jsp:useBean> 태그의 id 속성에 지정한 값은 이 JSP 페이지가
    서블릿 클래스로 변경될 때 _jspService() 메소드 안에서 지역 변수로
    선언되고 이 변수가 사용되기 때문에 같은 _jspService() 메소드 안에
    자바 코드로 변경되는 스크립틀릿이나 표현식에서도 <jsp:useBean>
    태그의 id 속성에 지정한 이름을 변수로 하여 객체에 접근할 수 있다.
```

```
--%>
<td class="content"><%= student.getAge() %></td>
</tr>
<tr>
<td class="title">연락처</td>
<td class="content">
<jsp:getProperty name="student" property="phone" /></td>
</tr>
</table>
</body>
</html>
```