CHAPTER

4

JSP 기초 문법

이번 장에서는 JSP의 스크립트 요소인 주석(Comment), 지시자(Directive), 액션 태그(Action Tag), 선언문(Declaration), 스크립트릿(Scriptlet), 표현식(Expression)에 대한 기본 문법을 예제를 통하여 학습할 것이다.

4.1 주석

JSP에서의 주석(Comment) 처리 방법에는 HTML 주석, JSP 주석, 스크립트릿의 자바 주석이 있다. 자바에서의 주석 처리 방법은 3장을 참고하기 바란다.

우리는 앞서 home 프로젝트 생성 시 작성한 index.jsp 페이지에서 주석 처리를 사용했었다.

HTML 주석 형식

```
〈!-- HTML 주석 문장 --〉
```

위의 형식으로 HTML 주석을 사용하면 클라이언트에서 소스 보기를 했을 때 소스코드가 출력된다. 즉, 클라이언트에서 읽을 수 있는 주석문이 된다.

JSP 주석 형식

```
〈%-- JSP 주석 문장 --%〉
```

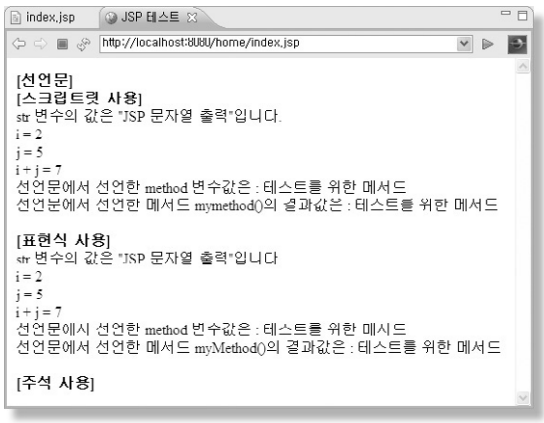
하지만 위의 형식으로 JSP 주석을 사용하면 클라이언트에서 소스 보기를 했을 때 소스코드가 출력되지 않는다. 즉, 클라이언트에서 읽을 수 없는 주석문이 된다.

[/WebContent/index.jsp]: home 프로젝트 생성 시 작성한 소스코드

```
01  <%@ page language="java" contentType="text/html; charset=UTF-8"
02    pageEncoding="UTF-8"%>
03  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
04  "http://www.w3.org/TR/html4/loose.dtd">
05  <html>
06  <head>
07  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
08  <title>JSP 테스트</title>
09  </head>
10  <body>
11
12  <b>[선언문]</b><br>
13  <%!
14  //선언문
15  private String method = "테스트를 위한 메서드";
16  private String myMethod() {
17          return method;
18  }
19  %>
20
```

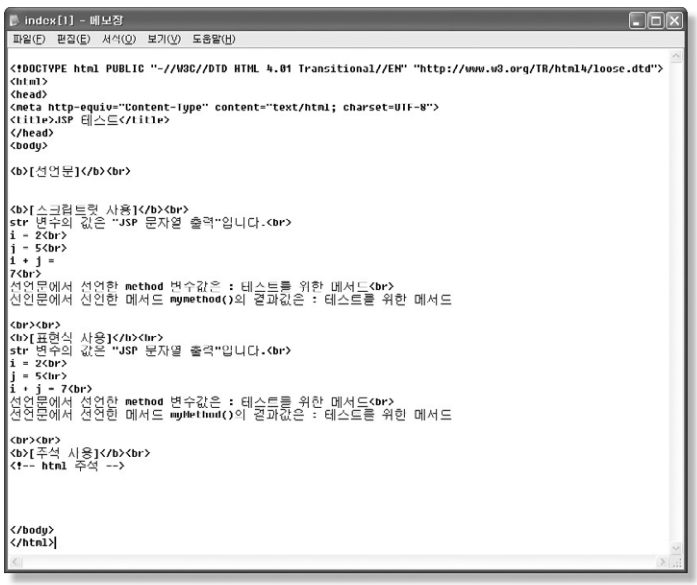
```
21  <b>[스크립트릿 사용]</b><br>
22  <%
23  String str = "JSP 문자열 출력";
24  int i = 2;
25  int j = 5;
26
27  out.println("str 변수의 값은 \"" + str + "\"입니다.<br>");
28  out.println("i = " + i + "<br>");
29  out.println("j = " + j + "<br>");
30  out.println("i + j = ");
31  out.println(i + j + "<br>");
32  out.println("선언문에서 선언한 method 변수 값은 : " + method + "<br>");
33  out.println("선언문에서 선언한 메서드 mymethod()의 결과 값은 : " + myMethod());
34  %>
35  <br><br>
36  <b>[표현식 사용]</b><br>
37  str 변수의 값은 "<%= str %>"입니다.<br>
38  i = <%=i %><br>
39  j = <%=j %><br>
40  i + j = <%=i+j %><br>
41  선언문에서 선언한 method 변수 값은 : <%= method %><br>
42  선언문에서 선언한 메서드 myMethod()의 결과 값은 : <%= myMethod() %>
43
44  <br><br>
45  <b>[주석 사용]</b><br>
46  <!-- html 주석 -->
47  <%-- JSP 주석 <%= str %> --%>
48
49  <%
50  //한 줄 주석
51  /*
52  여러 줄 주석
53  여러 줄 주석
54  */
55  %>
56
57  </body>
58  </html>
```

46번 라인을 보면 HTML 주석을 사용하였으며, 47번 라인에서는 JSP 주석을 사용하였다. 그리고 스크립트릿 내부에서는 자바에서의 주석 처리 방법인 //, /* */ 형식을 사용하였다.



● **그림 4-1** http://localhost:8080/home/index.jsp

웹 브라우저로 접속하여 확인해 보면 위의 그림과 같이 [주석 사용] 다음 부분에는 아무런 문자도 출력되지 않고 있다. 만약 웹 페이지를 다른 컴퓨터에서 보고자 한다면, localhost 대신 해당 서버 IP를 입력해 주면 된다(http://192.168.1.101:8080/home/index.jsp).



● 그림 4-2 http://localhost:8080/home/index.jsp 소스 보기

웹 페이지를 소스 보기로 보면 위의 그림과 같이 HTML 주석을 사용한 주석만 볼 수 있으며, JSP 주석과 스크립트릿에서 주석 처리한 부분은 소스 보기로도 출력되지 않음을 확인할 수 있다.

4.2 지시자

지시자(Directive)는 JSP 파일의 속성을 기술하는 곳이며, 클라이언트의 요청(request)에 의해 JSP 웹 페이지가 호출될 때 응답(response)에 필요한 정보를 JSP 컨테이너(톰캣)에게 전달하는 역할을 한다. 즉, JSP 컨테이너에게 "지시자가 지시하는 내용으로 처리하시오"라고 지시하는 것이다.

지시자의 종류에는 크게 다음 3가지 종류가 있으며, 상단 태그 안에서 '@' (at)으로 시작한다.

● 표 4-1

① page 지시자 JSP의 속성 정의	② include 지시자 JSP에 다른 파일 삽입	③ taglib 지시자 JSTL, 태그 라이브러리 사용
〈%@ page … %〉	〈%@ include … %〉	〈%@ taglib … %〉

4.2.1 page 지시자

page 지시자는 JSP 컨테이너가 처리할 때 필요한 각종 속성들을 정의하는 지시자이다.

형식

```
<%@ page 속성1="속성 값1" 속성2="속성 값2" … %>
```

● 표 3-5

속성	값	기본 값	예제
info	텍스트	없음	info="Copyleft 2010 by www.lug.or.kr"
language	스크립트 언어	"java"	language="java"
contentType	MIME 타입, 문자 집합	contentType="text/html"; charset=ISO-8859-1"	contentType="text/html"; charset=UTF-8"
extends	클래스명	없음	extends="kr.or.lug.myjsp"
import	외부 클래스, 패키지명	없음	import="java.util.Vector" import="java.sql.*,java.net.*"
session	boolean값	"true"	session="true"
buffer	buffer값 또는 "none"	"8kb"	buffer="12kb" buffer="false"
autoFlush	boolean값	"true"	autoFlush="false"
isThreadSafe	boolean값	"true"	isThreadSafe="true"
errorPage	로컬 URL	없음	errorPage="/error/error.jsp"
isErrorPage	boolean값	"false"	isErrorPage="false"
pageEncoding	페이지 캐릭터 인코딩값	"ISO-8859-1"	pageEncoding="UTF-8"

참고

JSP 한글 사용

예전에는 EUC-KR 캐릭터셋을 사용하였지만, 최근에는 UTF-8 캐릭터셋 사용을 권장한다.

4.2.1.1 info 속성

info 속성은 페이지를 설명해 주는 문자열이며, 속성 값의 내용이나 길이 제한이 없으며 설정하지 않아도 무관하다.

```
<%@ page info="www.lug.or.kr" %>
```

4.2.1.2 language 속성

language 속성은 JSP 스크립트 요소에서 사용할 언어를 지정하는 속성이다. 이 속성을 지정하지 않으면 기본 값으로 java가 지정된다.

```
<%@ page language="java" %>
```

4.2.1.3 contentType 속성

contentType 속성은 JSP 페이지의 내용을 어떤 형태로 출력할 것인지 MIME 형식으로 웹 브라우저에게 알려주는 속성이다. 지정할 속성 값으로는 text/html, text/plain, text/xml, text/gif 등이 있으며, 기본 값은 text/html의 MIME 형식이다.

```
<%@ page contentType="text/html" %>
```

그리고 JSP 페이지에서 사용할 캐릭터셋 문자형식을 지정할 수 있다. charset의 기본 값은 "ISO-8859-1"이지만 한글을 사용하기 위해 이 책에서는 UTF-8(utf-8)을 사용한다.

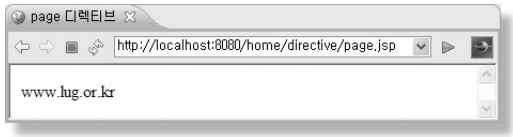
```
<%@ page contentType="text/html"; charset="UTF-8" %>
```

지금까지의 지시자를 사용한 다음 예제에서 **this.getServletInfo()** 메서드를 사용하여 현재 페이지의 info값을 호출해 보았다.

[/WebContent/type/casting_auto.jsp]

```
01 <%@ page info="www.lug.or.kr" language="java" contentType="text/html; charset=UTF-8"
02 pageEncoding="UTF-8"%>
03 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
04 "http://www.w3.org/TR/html4/loose.dtd">
05 <html>
06 <head>
07 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

```
08  <title>page 디렉티브</title>
09  </head>
10  <body>
11  <%=this.getServletInfo() %>
12  </body>
13  </html>
```



● **그림 4-3** http://localhost:8080/home/directive/page.jsp

4.2.1.4 extends 속성

JSP 페이지는 JSP 컨테이너에 의해 Servlet으로 변환된 후, 처리 결과를 웹 서버에 전송하여 클라이언트에게 보여주게 된다. 이때 JSP 페이지가 Servlet 소스로 변환되는 시점에서 자신이 상속받을 클래스를 지정할 때 extends 속성을 사용한다. 하지만 JSP 컨테이너가 알아서 적절한 클래스들을 상속시켜 변환해 주기 때문에 사용할 일은 거의 없다.

```
<%@ page extends="kr.or.lug.Directive" %>
<%-- kr.or.lug.Directive 클래스를 상속하겠다는 의미 --%>
```

4.2.1.5 import 속성

import 속성은 JAVA의 import 키워드와 동일한 기능이며, JSP 페이지 내에서 package 이름을 지정하지 않아도 패키지 내의 메서드 등을 사용할 수 있도록 하는 것이다. 그리고 import 속성은 page 지시자 중 유일하게 중복 사용이 가능한 속성이다.

```
<%@ page import="java.util.*, java.text.*" import="java.io.*" %>
<%-- 여러 개의 패키지를 쉼표(,)로 구분해서 사용할 수 있다. --%>
<%-- import 속성은 중복으로 사용할 수 있다. --%>
```

4.2.1.6 session 속성

세션은 웹 브라우저와 웹 서버가 지속적으로 상대를 인식하기 위해 필요한 정보를 임시로 저장해 두는 방법이며, 쇼핑몰 등에서 로그인 또는 장바구니 등을 구현할 때 사용한다.

session 속성은 JSP 페이지가 HttpSession을 사용할지 여부를 지정하는 속성이며, true와 false 값을 가질 수 있는데, true일 경우에는 현재의 페이지가 세션을 유지하고 동일한 세션이 존재하지 않을 경우에는 새로운 세션을 생성하여 서로 연결하게 된다. false일 경우에는 세션에 연결되지 않

는다. 이 속성의 기본 값은 true이기 때문에 세션을 사용하지 않을 경우에만 false로 설정하면 된다.

```
<%@ page session="false" %>
```

4.2.1.7 buffer 속성

JSP 페이지 내용을 출력하려면 out 객체를 사용해야 하는데, 이때 사용할 버퍼의 크기를 설정하는 속성이다. 기본 값은 8kb이며, none으로 설정하면 출력 버퍼를 사용하지 않고 JSP 페이지를 즉시 웹 브라우저로 전송하겠다는 의미이다.

```
<%@ page buffer="16kb" %>
<%@ page buffer="none" %>
```

단, buffer 속성을 none으로 설정하면 다음의 기능을 사용할 수 없다.

① <jsp:forward …>기능을 사용할 수 없다.
② 즉시 전송되기 때문에 출력한 내용을 취소할 수 없다.

버퍼란 데이터를 좀 더 효율적이고 안전하게 전송하기 위한 기술로서 데이터를 일정 용량만큼 미리 확보한 다음, 확보한 데이터를 한 번 전송하고 또 다시 확보, 전송을 반복하는 방법이다. 버퍼를 사용하면 전송 도중 문제가 발생하더라도 어느 정도 안정적으로 데이터를 전송할 수 있다.

일반적으로 인터넷 방송을 시청할 때 미디어플레이어에서 버퍼링하는 것도 이와 같은 방식이다. 이처럼 서버에서 웹 브라우저에 데이터를 전송할 때 지정한 버퍼의 크기만큼 모았다가 한 번에 전송하는 것이다.

4.2.1.8 autoFlush 속성

flush란 버퍼가 모두 찼을 때(full) 버퍼에 저장된 데이터를 실제 전송할 곳, 즉 클라이언트에게 전송하고 버퍼를 비우는(empty) 것이다. autoFlush 속성은 이와 같이 버퍼가 모두 찼을 때 자동으로 비울 것(전송할 것)인지 설정하는 속성이다. 기본 값은 true이며, buffer 속성에 지정된 크기만큼 버퍼를 유지하고 버퍼가 모두 차면 자동으로 클라이언트에게 전송한다. 속성 값을 false로 설정하면 버퍼 오버플로우(OverFlow)로 인한 예외(Exception)가 발생한다. buffer 속성이 none이면 autoFlush 속성은 false를 가질 수 없다. 기본 값은 true이다.

```
<%@ page autoFlush="true" %>
```

[/WebContent/directive/autoflush.jsp]

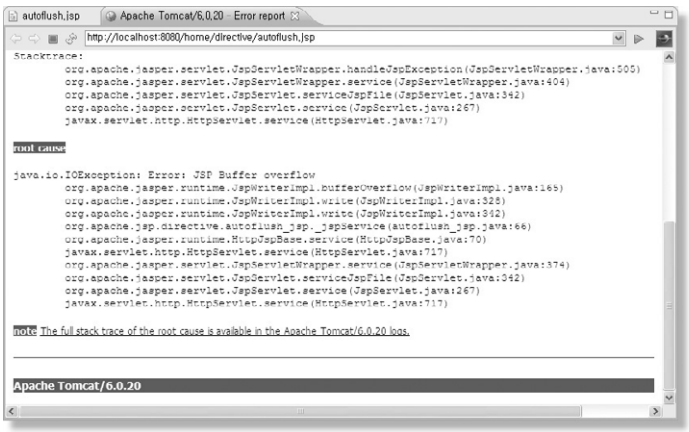
```
01  <%@ page language="java" contentType="text/html; charset=UTF-8"
02    pageEncoding="UTF-8"%>
03  <%@ page buffer="1kb" autoFlush="false" %>
04  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
```

```
05  "http://www.w3.org/TR/html4/loose.dtd">
06  <html>
07  <head>
08  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
09  <title>autoflush 속성 예제</title>
10  </head>
11  <body>
12  <%
13          for (int i = 0; i <= 1000 ; i++) {
14  %>
15  JSP
16  <%
17          }
18  %>
19  </body>
20  </html>
```

예제에서 buffer 크기를 1kb로 설정하고 autoFlush를 false를 설정하였기 때문에 전송할 데이터의 양이 버퍼 크기를 넘어서고 있다. 그래서 다음 그림과 같이 JSP buffer Overflow로 인한 예외가 발생한다.

예제에서 autoFlush를 true로 변경하면 정상적인 결과 값을 출력할 수 있다.

● 그림 4-4 http://localhost:8080/home/directive/autoflush.jsp 페이지 상단



● **그림 4-5** http://localhost:8080/home/directive/autoflush.jsp 페이지 하단

● **그림 4-6** http://localhost:8080/home/directive/autoflush.jsp autoFlush="true"일 때

4.2.1.9 isThreadSafe 속성

서블릿은 접속 요청에 대해 프로세스 단위가 아닌 쓰레드로 처리하게 된다. 즉, 동시 접속요청에 유용하다. 이때 각 쓰레드는 자원(멤버 변수)을 공유하기 때문에 데이터의 안정성을 보장할 수 없게 되는데, isThreadSafe 속성은 이와 같은 쓰레드 처리 상황에서 데이터의 안정성을 보장하는 속성이며, 기본 값은 true이므로 직접 지정할 필요는 없다.

```
<%@ page isThreadSafe="true" %>
```

4.2.1.10 errorPage, isErrorPage 속성

errorPage 속성은 현재 JSP 페이지를 처리하는 도중 에러가 발생할 경우에 호출할 페이지를 지정한다.

```
<%@ page errorPage="error.jsp" %>
```

isErrorPage 속성은 현재 JSP 페이지가 에러 처리를 담당하는 페이지인지 아닌지 지정할 때 사용

하는 속성이다. 기본 값은 false이며, errorPage 속성으로 지정된 JSP 페이지라면 isErrorPage 속성을 true로 지정하면 된다.

```
<%@ page isErrorPage="true" %>
```

[/WebContent/directive/errorpage.jsp]: 예외 에러 발생 페이지

```
01  <%@ page language="java" contentType="text/html; charset=UTF-8"
02    pageEncoding="UTF-8" errorPage="/directive/error.jsp"%>
03  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
04  "http://www.w3.org/TR/html4/loose.dtd">
05  <html>
06  <head>
07  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
08  <title>예외 에러 발생 페이지</title>
09  </head>
10  <body>
11  <%
12          int a = 10;
13          int b = 0;
14  %>
15  a와 b의 사칙 연산<br>
16  a + b = <%=a + b %><br>
17  a - b = <%=a - b %><br>
18  a * b = <%=a * b %><br>
19  a / b = <%=a / b %><!-- 0으로 나누지 못하므로 예외 에러 발생 -->
20  </body>
21  </html>
```

예제 JSP 페이지의 page 디렉티브에서 errorPage 속성을 "/directive/error.jsp" 파일로 지정하였다. 이 경로명은 home 프로젝트의 하위 경로를 의미한다. 즉, 현재 페이지에서 에러가 발생하면 지정한 페이지를 출력하여 에러가 발생했음을 시각적으로 보여주도록 한 것이다. 즉, 현재 소스코드의 19번 라인에서 변수 a의 값을 0으로 나누도록 작성함으로써 인위적으로 ArithmeticException이 발생하도록 하였다.

이제 에러에 대해 출력할 error.jsp 페이지를 작성하자.

[/WebContent/directive/error.jsp]: 예외 에러가 발생하면 보여줄 페이지

```
01  <%@ page language="java" contentType="text/html; charset=UTF-8"
02    pageEncoding="UTF-8" isErrorPage="true"%>
03  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
04  "http://www.w3.org/TR/html4/loose.dtd">
05  <html>
06  <head>
```

```
07  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
08  <title>예외 에러 발생!</title>
09  </head>
10  <body>
11  <h1>Exception Error!</h1>
12  아래와 같은 예외가 발생하였습니다.<br>
13  <%=exception.getMessage() %><br>
14  <%=pageContext.getErrorData().getThrowable().toString() %>
15  </body>
16  </html>
```

에러가 발생하면 처리할 /derective/error.jsp 페이지에서 page 디렉티브의 isErrorPage 속성을 true로 지정하고, body 태그 부분에서 에러 메시지를 출력하기 위해 다음의 두 가지 메서드를 사용할 수 있다. 두 가지 방법 중 pageContext 내장 객체 사용을 권장한다.

```
// exception 객체의 에러 메시지 출력
exception.getMessage()
// pageContext를 사용하여 ErrorData 객체에 담겨 있는 에러 메시지 출력
pageContext.getErrorData().getThrowable().toString()
```

● 그림 4-7 http://localhost:8080/home/directive/errorpage.jsp

4.2.1.11 pageEncoding 속성

pageEncoding 속성은 JSP 페이지에서 사용하는 문자의 인코딩을 지정할 때 사용한다. 기본 값은 ISO-8859-1로 인코딩되지만, 한글을 출력하기 위해 UTF-8로 지정한다.

```
<%@ page pageEncoding="UTF-8" %>
```

pageEncoding 속성 값은 contentType 속성의 charset 설정 값과 같다. 즉, 다음의 두 코드는 동일한 설정이다.

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

4.2.2 include 지시자

웹 페이지를 구성하다 보면 초기 페이지 또는 특정 페이지에 공통적으로 포함될 내용이 필요할 경우가 있다. 이런 경우, 공통적인 내용을 별도의 파일에 저장해 두고 include 지시자를 사용하여 file 속성에 포함할 파일명을 지정하면 된다.

```
<%@ include file="포함할 파일명" %>
```

include 지시자를 사용한 JSP 페이지를 컴파일하면 포함될 JSP 페이지의 소스 내용을 포함하여 컴파일하게 된다. 즉, 2개의 파일이 하나의 파일로 통합되는 것이다.

이와 같이 include 지시자를 사용하면 두 개의 페이지는 하나의 페이지가 되기 때문에 include될 페이지를 코딩할 때 하나의 페이지로 포함될 것을 고려하여 변수를 사용해야 한다. 즉, 변수의 중복 선언 등은 할 수 없다.

이처럼 하나의 페이지로 구현할 수 있는 것을 굳이 2개로 나누어서 사용하는 이유는 여러 페이지에 공통적으로 사용될 페이지가 존재하기 때문이다. 예를 들면, 모든 페이지의 상단에 top.jsp 파일로 메뉴 링크를 작성해 두고 하단에 포함될 bottom.jsp 페이지에 사이트 정보, 관리자 이메일 등을 작성해 둔 다음, 모든 페이지 파일에서 include 지시자를 사용하여 top.jsp와 bottom.jsp 파일을 포함시켜 사용하면 보다 효율적일 것이다.

[/WebContent/directive/include.jsp]

```
01  <%@ page language="java" contentType="text/html; charset=UTF-8"
02    pageEncoding="UTF-8"%>
03  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
04  "http://www.w3.org/TR/html4/loose.dtd">
05  <html>
06  <head>
07  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
08  <title>include 지시자 사용</title>
09  </head>
10  <body>
11  top : <%@ include file="top.jsp" %>
12  <hr>include.jsp의 본문 내용입니다.<hr>
13  bottom : <%@ include file="bottom.jsp" %>
14  </body>
15  </html>
```

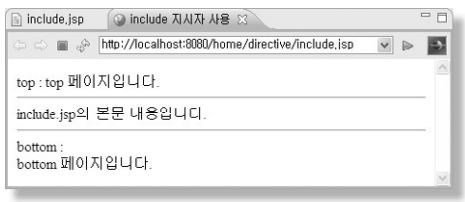
[/WebContent/directive/top.jsp]

```
01  <%@ page language="java" contentType="text/html; charset=UTF-8"
02    pageEncoding="UTF-8"%>
03  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
```

```
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>top 페이지</title>
</head>
<body>
top 페이지입니다.<br>
</body>
</html>
```

[/WebContent/directive/bottom.jsp]

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>bottom 페이지</title>
</head>
<body>
<br>bottom 페이지입니다.
</body>
</html>
```



● **그림 4-8** http://localhost:8080/home/directive/include.jsp

그리고 각종 변수의 선언을 따로 분리하여 하나의 페이지를 작성해 두면, 변수들의 값을 변경해야 할 경우가 발생하더라도 클라이언트에게 보여주는 페이지는 수정하지 않고 변수가 정의된 페이지만 변경하면 되기 때문에 보다 간결한 코드를 작성할 수 있다.

만약 특정 디렉토리 내의 모든 페이지에 동일한 상단, 하단 페이지를 포함해야 한다면 모든 파일에 include 지시자를 추가해야 할 것 같다. 하지만 좀 더 간단한 방법이 없을까?

답은 간단하다. /WebContent/WEB-INF/web.xml 설정 파일에 다음과 같이 입력하면 된다.

```
...
  <jsp-config>
    <jsp-property-group>
        <url-pattern>/directive/*</url-pattern>
        <include-prelude>/directive/top.jsp</include-prelude>
        <include-coda>/directive/bottom.jsp</include-coda>
    </jsp-property-group>
  </jsp-config>
</web-app>
```

위의 web.xml 코드를 해석하면 /directive 디렉토리 내에서 *, 즉 요청받는 모든 파일들에 대해 **<include-prelude>** 태그에 의해 파일의 상단에 **top.jsp** 파일을 삽입하고, **<include-coda>** 태그에 의해 파일의 하단에 **bottom.jsp** 파일을 삽입하라는 의미이다.

위와 같이 web.xml 파일을 변경한 다음, /directive 디렉토리의 page.jsp 파일에 접속해 보면 다음과 같이 top.jsp 파일과 bottom.jsp 파일이 삽입되었음을 확인할 수 있다. 단, web.xml 파일의 설정이 곧바로 적용되지 않기 때문에 page.jsp 파일에서 공백을 하나 추가한 다음에 재저장하면 web.xml 설정이 적용될 것이다.

● **그림 4-9** http://localhost:8080/home/directive/page.jsp

4.2.3 taglib 지시자

taglib 지시자는 사용자 정의형 태그를 JSP 페이지에서 사용하기 위한 지시자이다. 태그 라이브러리는 URI, Prefix 등의 속성으로 구성되는데, URI는 TLD(Tag Library Descriptor) 파일을 지정할 때 사용한다. TLD 파일은 사용자 정의형 태그이며, XML 형식으로 미리 작성되어 있어야 한다. 그리고 TLD 파일에서는 사용자 정의형 태그의 실제적인 구현을 위한 자바 클래스 파일을 지정해야 한다. Prefix는 현재 JSP 페이지에서 사용자 정의형 태그를 사용하기 위한 접두어 역할을 한다. XML 스키마에서의 네임스페이스라고 생각하면 된다.

형식

```
<%@ taglib uri="/META-INF/mytag.tld" prefix="mytag" %>
```

참고

URL과 URI

URL(Uniform Resource Locator)은 **실제의 네트워크 경로**를 가리키며, 네트워크 상의 리소스 접근 시에 사용된다. 정보 자원의 위치(Location)를 나타내는 식별 체계이다.

● **표 4-3**

구분	프로토콜	호스트 주소	포트 번호	디렉토리	파일명
예제	http://	www.lug.or.kr	:8080	/jsp/	index.jsp

URI(Uniform Resource Identifier)는 URL에서의 프로토콜, 호스트 주소, 포트 번호를 제외한 자원의 경로만 기술한다. URI는 모든 웹 상의 정보 자원(문헌, 이미지, 다운로드가 가능한 파일, 서비스, 전자메일)의 이름과 주소를 짧은 문자열로 표현한 집합이다.

URL **http://www.lug.or.kr/jsp/directive/taglib.jsp**

URI **/jsp/directive/taglib.jsp**

[예제] – 다음 예제는 mytag.tld 파일이 필요하기 때문에 실행되지는 않는다.

```
01  <%@ page language="java" contentType="text/html; charset=UTF-8"
02    pageEncoding="UTF-8"%>
03  <%@ taglib uri="/META-INF/mytag.tld" prefix="mytag" %>
04  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
05  "http://www.w3.org/TR/html4/loose.dtd">
06  <html>
07  <head>
08  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
09  <title>taglib : 태그 라이브러리</title>
10  </head>
11  <body>
12  <mytag:GetInfo name="myinfo" />
13  </body>
14  </html>
```

예제의 <mytag:GetInfo name="myinfo" /> 부분은 mytag.tld 파일에 정의된 사용자 정의형 태그 중 GetInfo 태그를 사용하고, name 속성에 myinfo 값을 전달하는 의미이다. 만약 위의 예제를 정상적으로 실행하려면 mytag.tld 파일을 작성하면서 실제 기능을 수행할 클래스 파일을 지정해야 하고, 해당 클래스 파일에서는 GetInfo 태그를 처리하는 부분을 구현하고 name 속성으로 전달받은 myinfo를 사용하여 데이터베이스 연결과 같은 필요한 작업을 수행하여 현재 태그의 위치에 결과를 출력하도록 구성해야 한다.

태그 라이브러리는 복잡하고 반복되는 작업을 간단한 태그 형태로 작성한 것을 말한다. 이와 같은 태그 라이브러리를 사용하면 파일 관리의 일관성을 유지하고 유지보수의 효율성을 높일 수 있다. 하지만 사용자 정의형 태그를 너무 많이 사용하면 호환성이 떨어지고 라이브러리에 대한 별도의 공부가 필요하게 되는 문제점이 있다.

일반적으로 많이 사용하는 사용자 정의형 태그는 **JSTL(JSP Standard Tag Library)**로 배포되고 있다. taglib 지시자에 대한 보다 자세한 내용은 다시 공부하게 될 것이다.

4.3 액션 태그

액션 태그는 JSP 문법의 구성 요소이며, 다음과 같은 기능을 제공한다.

① JSP 페이지 간의 이동 제어
② 자바 애플릿 실행
③ 자바빈즈 기능 사용

형식: XML 형식 사용

```
<jsp:액션 태그 속성들 />
```

액션 태그의 종류는 다음과 같이 6가지 종류가 있다.

include, forward, plug-in, useBean, setProperty, getProperty

4.3.1 include 액션 태그

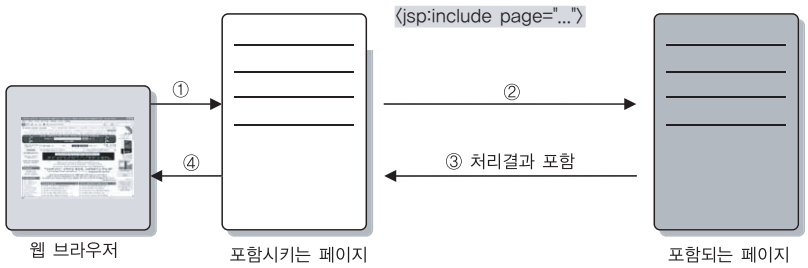
include 액션 태그는 다른 페이지를 현재 페이지에 포함시키는 기능을 수행하며, include 지시자의 기능과 유사하다. 특히 include 지시자는 단순히 소스 내용을 텍스트로 포함하지만, **include 액션 태그는 포함시킬 페이지의 처리 결과를 포함시킨다.** 이때 포함되는 페이지는 HTML, JSP, Servlet 모두 가능하다.

즉, include 액션은 두 개의 파일을 각각 컴파일해서 관리하기 때문에 동적인 페이지에 포함시킬 때 유용하며, include 지시자는 두 개의 파일을 하나의 파일로 컴파일하기 때문에 자주 변경되지 않는 정적인 페이지를 포함시킬 때 사용하는 것이 바람직하다.

flush 속성은 포함될 페이지로 이동할 때 현재 페이지가 지금까지 출력 버퍼에 저장한 결과를 어떻게 처리할 것인지 결정하는데, true로 설정하면 포함할 페이지의 내용을 삽입하기 이전에 현재 페이지가 지금까지 버퍼에 저장한 내용을 출력하게 된다.

형식

```
<jsp:include page="로컬 파일 경로" flush="true" />
```



① 요청(request) → ② 요청(request) → ③ 응답(response) → ④ 응답(response)

● **그림 4-10**

[/WebContent/actiontag/include.jsp]

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>include 액션 태그</title>
</head>
<body>
<h1>include 액션 태그 예제</h1>
<form method="post" action="includetag.jsp">
이름 : <input type="text" name="name"><br>
국가 : <input type="text" name="nation"><br><br>
<input type="submit" value="작성 완료">
</form>
</body>
</html>
```

폼에 이름과 국가를 입력하도록 구성하였다.

[/WebContent/actiontag/includetag.jsp]

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

```
<title>includetag.jsp</title>
</head>
<body>
<%
        request.setCharacterEncoding("utf-8"); // 캐릭터셋 설정
        String name = "JSP 기초"; // 사용되지 않는 변수
%>
<jsp:include page="includetagheader.jsp" flush="true" />
<hr>
include 액션 태그의 body 페이지입니다.
</body>
</html>
```

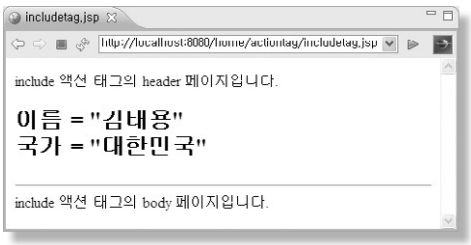
한글을 처리하기 위해 include.jsp에서 전달된 name, nation 변수 값의 캐릭터셋을 utf-8로 설정하였으며, name 변수의 값을 재정의하였다. 하지만 재정의한 name 변수의 값은 사용되지 않으며, 폼에서 전달된 변수의 값이 include 액션 태그로 사용한 파일에 적용된다.

[/WebContent/actiontag/includetagheader.jsp]

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>includetagheader.jsp</title>
</head>
<body>
include 액션 태그의 header 페이지입니다.<p>
<h2>
이름 = "<%=request.getParameter("name") %>"<br>
국가 = "<%=request.getParameter("nation") %>"
</h2>
</body>
</html>
```

include.jsp 페이지에 접속하여 다음 그림과 같이 입력하고나서 작성 완료 버튼을 클릭하면, include.jsp 폼에서 전달받은 name과 nation 변수의 값을 includetagheader.jsp 페이지에서 전달받아 JSP 페이지를 처리하고 그 결과를 출력하게 된다.

● **그림 4-11** http://localhost:8080/home/actiontag/include.jsp



● **그림 4-12** http://localhost:8080/home/actiontag/includetag.jsp

JSP 페이지에서 포함될 페이지에 파라미터를 추가하여 전달할 수 있는데, include 액션 태그 몸체 안에 <jsp:param name="파라미터명" value="파라미터 값" /> 형식을 사용한다.

형식

```
<jsp:include page="포함될 페이지" flush="true">
  <jsp:param name="파라미터명" value="파라미터 값" />
  <jsp:param name="파라미터명" value="파라미터 값" />
</jsp:include>
```

[/WebContent/actiontag/include2.jsp]

```
01  <%@ page language="java" contentType="text/html; charset=UTF-8"
02    pageEncoding="UTF-8"%>
03  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
04  "http://www.w3.org/TR/html4/loose.dtd">
05  <html>
06  <head>
07  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
08  <title>include 액션 태그2</title>
09  </head>
10  <body>
11  <h1>include 액션 태그 예제2</h1>
12  <form method="post" action="includetag2.jsp">
13  웹 프로그래밍 언어 : <input type="text" name="language"><br><br>
```

```
14  <input type="submit" value="작성 완료">
15  </form>
16  </body>
17  </html>
```

폼을 사용하여 language 파라미터를 includetag2.jsp로 전달하는데, 필자는 "JSP" 문자열을 입력하였다.

[/WebContent/actiontag/includetag2.jsp]

```
01  <%@ page language="java" contentType="text/html; charset=UTF-8"
02    pageEncoding="UTF-8"%>
03  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
04  "http://www.w3.org/TR/html4/loose.dtd">
05  <html>
06  <head>
07  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
08  <title>includetag2.jsp</title>
09  </head>
10  <body>
11  <%
12          request.setCharacterEncoding("utf-8");
13  %>
14  <jsp:include page="includetagheader2.jsp">
15          <jsp:param name="language" value="PHP" />
16  </jsp:include>
17  <hr>
18  <b>
19  request parameter : <%=request.getParameter("language") %>
20  </b><br>
21  include 액션 태그의 body 페이지입니다.
22  </body>
23  </html>
```

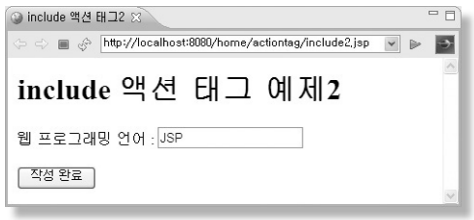
먼저 요청 캐릭터셋을 utf-8로 설정하고 <jsp:include …> 액션 태그를 사용하여 인클루드할 includetagheader2.jsp 파일의 language 파라미터 값으로 "PHP" 문자열을 전달한다. 그리고 19번 라인에서 앞서 요청받은 language 파라미터의 값을 출력하면 JSP가 출력된다.

[/WebContent/actiontag/includetagheader2.jsp]

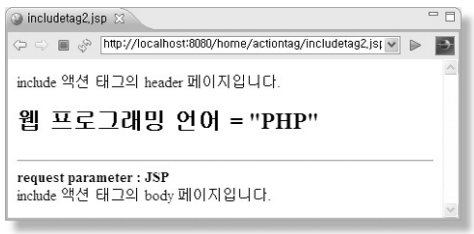
```
01  <%@ page language="java" contentType="text/html; charset=UTF-8"
02    pageEncoding="UTF-8"%>
03  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
```

```
04  "http://www.w3.org/TR/html4/loose.dtd">
05  <html>
06  <head>
07  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
08  <title>includetagheader2.jsp</title>
09  </head>
10  <body>
11  include 액션 태그의 header 페이지입니다.<p>
12  <h2>
13  웹 프로그래밍 언어 = "<%=request.getParameter("language") %>"
14  </h2>
15  </body>
16  </html>
```

includetagheader2.jsp 페이지의 language 파라미터 값은 includetag2.jsp 페이지의 include 액션 태그에서 설정한 파라미터 값인 "PHP" 문자열을 출력하게 된다.



● **그림 4-13** http://localhost:8080/home/actiontag/include2.jsp



● **그림 4-14** http://localhost:8080/home/actiontag/includetag2.jsp

<jsp:include …> 액션 태그와 include 지시자의 비교 정리는 다음 표를 참고하자.

● **표 4-4** 〈jsp:include …〉 액션 태그와 include 지시자 비교 정리

비교 항목	〈jsp:include …〉 액션 태그	include 지시자
처리시간	요청 시간에 처리.	JSP 파일을 자바 소스로 변환 시 처리.
기능	별도 파일로 요청 처리 흐름을 이동. 포함할 파일의 실행 결과를 포함하여 처리한다.	현재 파일에 삽입. 현재 파일에 포함할 파일의 소스코드를 삽입하여 하나의 파일로 구성한 다음, 한 번에 처리한다.
데이터 전달 방법	request 내장 객체나 〈jsp:param〉을 이용한 파라미터 전달.	외부 페이지에 변수를 선언한 후, 변수의 값을 변경 또는 출력.
용도	화면 레이아웃의 일부분을 모듈화할 때 주로 사용.	다수의 JSP 페이지에서 공통으로 사용되는 변수를 지정하는 코드나 저작권과 같은 공통의 문장을 포함할 때 사용.

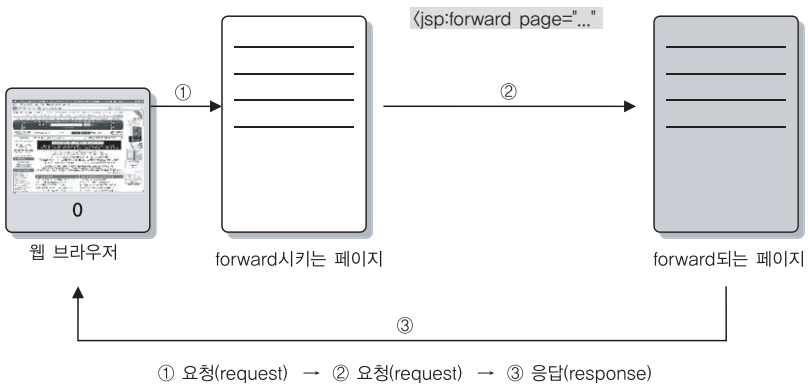
4.3.2 forward 액션 태그

forwad 액션 태그는 현재 페이지로 접속 요청을 하면 다른 페이지로 이동하도록 하는 액션 태그이며, 사용자 클라이언트의 요청 파라미터에 따라 이동할 페이지를 각각 다르게 지정하여 처리할 수 있다.

include 액션 태그는 다른 JSP 페이지로 제어권이 넘어갔다가 다시 원래의 JSP 페이지로 제어권이 넘어오지만, forward 액션 태그는 제어권이 완전히 넘어가는 액션 태그이다.

형식

```
<jsp:forward page="포워딩할 파일명" />
<jsp:forward page="포워딩할 파일명"></jsp:forward>
<jsp:forward page='<%=expression %>' />
```



● **그림 4-15**

forward 액션 태그는 그림 4-15와 같이 ①클라이언트 웹 브라우저의 요청이 있으면 forward시키는 페이지에서 ②forward되는 페이지로 제어권을 넘기고 ③제어권을 넘겨받은 페이지에서 요청을 처리한 결과를 직접 클라이언트 웹 브라우저에게 응답해 준다.

[/WebContent/actiontag/forward.jsp]

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>forward 액션 태그 예제</title>
</head>
<body>
<h1>forward 액션 태그 예제</h1>
<form method="post" action="forwardtag.jsp">
ID : <input type="text" name="id"><br>
PASSWORD : <input type="password" name="pass"><br><br>
<input type="submit" value="보내기">
</form>
</body>
</html>
```

form 태그를 사용하여 입력한 id와 pass 파라미터를 forwardtag.jsp 파일로 전달한다.

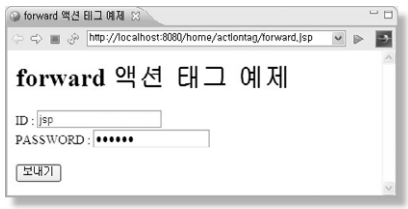
[/WebContent/actiontag/forwardtag.jsp]

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>forwardtag.jsp</title>
</head>
<body>
<%
        request.setCharacterEncoding("utf-8");
%>
forward 액션 태그 사용
<jsp:forward page="forwardtag2.jsp" />
</body>
</html>
```

요청된 캐릭터셋을 utf-8로 설정하고 forward 액션 태그를 사용하여 forwardtag2.jsp 파일로 포워딩한다. 이와 같이 forward 액션 태그를 사용하면 제어권이 완전히 넘어가기 때문에 14번 라인의 문자열이 출력되지 않는다.

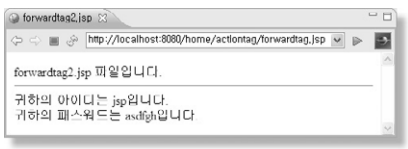
[/WebContent/actiontag/forwardtag2.jsp]

```
01  <%@ page language="java" contentType="text/html; charset=UTF-8"
02    pageEncoding="UTF-8"%>
03  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
04  "http://www.w3.org/TR/html4/loose.dtd">
05  <html>
06  <head>
07  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
08  <title>forwardtag2.jsp</title>
09  </head>
10  <body>
11  forwardtag2.jsp 파일입니다.<hr>
12  귀하의 아이디는 <%=request.getParameter("id") %>입니다.<br>
13  귀하의 패스워드는 <%=request.getParameter("pass") %>입니다.
14  </body>
15  </html>
```



● **그림 4-16** http://localhost:8080/home/actiontag/forward.jsp

폼에서 ID에는 "jsp", PASSWORD에는 "asdfgh" 문자열을 입력하였다. 다음 결과 화면을 보면 URL은 변경되지 않고 출력되는 페이지만 forwardtag2.jsp를 출력하고 있다.



● **그림 4-17** http://localhost:8080/home/actiontag/forwardtag.jsp

그리고 forward 액션 태그에서 추가적인 파라미터 값을 전달하고자 할 경우 <jsp:param name="파라미터명" value="파라미터 값" /> 형식을 사용할 수 있다.

형식

```
<jsp:forward page="포워딩할 페이지">
  <jsp:param name="파라미터명" value="파라미터 값" />
  <jsp:param name="파라미터명" value="파라미터 값" />
</jsp:forward>
```

4.3.3 plugin 액션 태그

<jsp:plugin ...> 액션 태그는 자바 플러그인을 사용하여 자바 애플릿이나 자바빈즈 컴포넌트를 JSP 페이지에서 실행할 때 사용하는 액션 태그이다. JSP 컨테이너는 <jsp:plugin ...> 액션 태그를 웹 브라우저에서 인식할 수 있는 태그로 변환하여 웹 브라우저가 자바 플러그인을 사용하여 자바 애플릿을 실행하도록 만들어준다.

4.3.4 useBean, setProperty, getProperty 액션 태그

useBean 액션 태그는 자바빈즈(JavaBeans)와 통신하기 위해 구현한 액션 태그이며, 액션에서 가장 중요한 부분이다. 기본 문법 구조는 다음과 같다.

useBean, setProperty, getProperty 액션 태그 형식

```
<jsp:useBean id="변수명" class="빈즈 클래스명" />
<jsp:setProperty name="변수명" property="속성명" />
<jsp:getProperty name="변수명" property="속성명" />
```

useBean 액션 태그는 빈즈 클래스를 사용한다는 의미이며, 빈즈에서 값을 가져올 때에는 getProperty, JSP 파일에서 빈즈 객체의 값을 설정할 때에는 setProperty를 사용한다. 여기서 getProperty는 빈즈 클래스의 getter 메서드(getXxx())와 setter 메서드(setXxx())를 내부적으로 호출한다.

useBean 액션 태그는 자바빈즈 부분에서 자세히 공부하도록 하자.

4.4 선언문과 표현식

JSP에서는 선언문<%! %>에서 멤버 변수나 메서드를 선언하고 표현식<%= %>에서 호출할 수 있다.

4.4.1 선언문

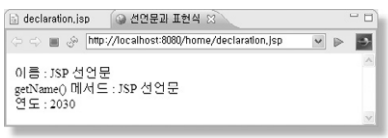
선언문(Declaration)에서는 멤버 변수나 메서드를 선언한다. 이때 선언된 멤버 변수와 메서드는 JSP 페이지가 파싱(parsing)될 때 Servlet의 멤버 변수와 메서드로 변환되며, 출력될 내용은 JSP 파일의 _jspService() 메서드에 포함된다.

[선언문 형식: 스크립트릿과 다르게 선언문의 시작은 "<%!"로 시작한다.]

```
<%!
멤버 변수나 메서드를 선언한다.
%>
```

[/WebContent/declaration.jsp]

```
01  <%@ page language="java" contentType="text/html; charset=UTF-8"
02    pageEncoding="UTF-8"%>
03  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
04  "http://www.w3.org/TR/html4/loose.dtd">
05  <html>
06  <head>
07  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
08  <title>선언문과 표현식</title>
09  </head>
10  <body>
11  <%!
12          String name = "JSP 선언문";
13          int year = 2030;
14          public String getName() {
15                  return name;
16          }
17  %>
18  이름 : <%=name %><br>
19  getName() 메서드 : <%=getName() %><br>
20  연도 : <%=year %>
21  </body>
22  </html>
```



● **그림 4-18** http://localhost:8080/home/declaration.jsp

위 예제에서는 선언문 형식<%! %>을 사용하여 멤버 변수 2개와 getName() 메서드를 선언하였다. 그리고 표현식을 사용하여 name 멤버 변수의 값(<%=name %>)을 출력하고, getName() 메서

드를 호출(<%=getName() %>)한 결과 값을 출력하고, year 멤버 변수를 출력하였다(<%=year %>). 앞서도 언급하였지만 선언문은 사용하지 않도록 하자.

4.4.2 표현식

표현식(Expression)은 앞서 많이 보아왔던 것처럼 <%= %> 형식을 사용하여 데이터나 메서드를 호출하여 그 결과 값을 출력하기 위해 사용한다. 단, 표현식의 마지막에는 세미콜론(;)을 사용하지 않는다.

표현식은 서블릿으로 변환 시 out.println() 메서드로 변환되는데, out.println()에서 사용할 수 있는 산술식이나 + 연산자를 사용한 연산과 문자열의 결합도 가능하다.

형식

```
변수 출력: <%=name %>
메서드 호출: <%=getName() %>
사칙 연산과 문자열 결합: <%="i+100="+(i+100)+"입니다" %>
```

4.5 스크립트릿

스크립트릿(Scriptlet)은 JSP 파일 내에서 자바 코드를 기술하는 부분으로서 가장 많이 사용하는 스크립트 요소이며, JSP 파일 내의 어디에서나 사용할 수 있다. JSP 파일이 서블릿으로 변환되고 요청될 때 _jspService() 메서드 안에 선언되기 때문에 선언문과는 달리 스크립트릿에서 선언되는 변수는 **지역 변수**로 선언된다.

스크립트릿을 사용하면 자바 코드를 자유 자재로 구사할 수 있기 때문에 유용할지 모르지만, 복잡한 프로그램일 경우와 그래픽 디자이너와 함께 협업 작업을 해야 할 팀 단위 작업의 경우에 스크립트릿을 사용하여 코딩하면 추후 유지보수가 어렵게 된다. 이와 같은 문제점을 해결하기 위해 EL(Expression Language) 표현 언어와 JSTL 커스텀 태그 라이브러리, 자바빈즈를 사용한다.

스크립트릿 부분에서는 순수 자바 코드만 사용할 수 있으므로 HTML 출력이 필요한 경우에 out.println() 메서드를 사용하거나 스크립트릿을 닫고 HTML 태그와 표현식을 이용해서 출력하고 다시 스크립트릿을 시작하여 작성한다.

[스크립트릿 형식: 선언문과는 다르게 "<%!"로 시작한다.]

```
<%
변수 선언, if/for/while/switch 등 자바 코드 작성
%>
```

[/WebContent/scriptlet.jsp]

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>스크립트릿 예제</title>
</head>
<body>
<%
          // 스크립트릿
          int number = 1;
          for (int i =0; i < 5; i++) {
                    out.println(number + " : ");
                    out.println(getName() + "<br>");
                    number++;
          }
%>
<%!
          // 선언문
          String name = "홍길동";
          public String getName() {
                    return name;
          }
%>
</body>
</html>
```

위의 예제 소스 파일이 서블릿으로 변환된 scriptlet_jsp.java 파일의 소스코드를 아래에 적어 두었다. 자세히 보면 선언문으로 선언한 멤버 변수와 메서드는 서블릿의 상단에 선언되어 있으므로 전역 변수와 메서드가 되고, 스크립트릿의 자바 코드는 _jspService() 메서드 내에 선언되어 있으므로 지역 변수가 되는 것을 쉽게 확인할 수 있다.

특히 위의 JSP 예제에서는 스크립트릿을 먼저 작성하고 선언문을 나중에 선언하였지만, JSP 파일이 서블릿으로 변환될 때 선언문의 변수와 메서드는 JAVA 소스의 상단에 위치하기 때문에 JSP 파일 내의 어느 위치에서나 스크립트릿을 사용하여 선언문의 변수와 메서드를 사용할 수 있다.

위의 예제의 스크립트릿 부분에서 문자열 출력을 위하여 out.println() 메서드를 사용하였지만, 다음과 같이 HTML 영역에서 표현식을 사용하여 출력할 수도 있다. 이와 같이 표현식을 사용하여 작업하면 레이아웃 등의 디자인 작업에 있어서 편리하다. 그러나 표현식을 남용하면 소스코드의 가독성이 떨어지기 때문에 특별한 경우가 아니라면 JSTL 태그 라이브러리나 자바빈즈를 사용하여 프로그래밍하는 것이 바람직하다.

[스크립트릿의 out.println() 메서드를 표현식으로 변경]

```
<%
	// 스크립트릿
	int number = 1;
	for (int i =0; i < 5; i++) {
			//out.println(number + " : ");
			//out.println(getName() + "<br>");
			//number++;
	//}
%>
<%=number %> : <%=getName() %><br>
<%
			number++;
	}
%>
```

[D:\java\workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\work\Catalina\localhost\home\org\apache\jsp\scriptlet_jsp.java]

```
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;

public final class scriptlet_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {


        // 선언문
        String name = "홍길동";
        public String getName() {
        return name;
        }

  private static final JspFactory _jspxFactory = JspFactory.getDefaultFactory();

  private static java.util.List _jspx_dependants;

  private javax.el.ExpressionFactory _el_expressionfactory;
  private org.apache.AnnotationProcessor _jsp_annotationprocessor;

  public Object getDependants() {
    return _jspx_dependants;
  }

  public void _jspInit() {
```

```
    _el_expressionfactory =
_jspxFactory.getJspApplicationContext(getServletConfig().getServletContext()).getExpressionFactory();
    _jsp_annotationprocessor = (org.apache.AnnotationProcessor)
getServletConfig().getServletContext().getAttribute(org.apache.AnnotationProcessor.class.getName());
  }

  public void _jspDestroy() {
  }

  public void _jspService(HttpServletRequest request, HttpServletResponse response)
        throws java.io.IOException, ServletException {

    PageContext pageContext = null;
    HttpSession session = null;
    ServletContext application = null;
    ServletConfig config = null;
    JspWriter out = null;
    Object page = this;
    JspWriter _jspx_out = null;
    PageContext _jspx_page_context = null;


    try {
      response.setContentType("text/html; charset=UTF-8");
      pageContext = _jspxFactory.getPageContext(this, request, response,
                        null, true, 8192, true);
      _jspx_page_context = pageContext;
      application = pageContext.getServletContext();
      config = pageContext.getServletConfig();
      session = pageContext.getSession();
      out = pageContext.getOut();
      _jspx_out = out;

      out.write("\r\n");
      out.write("<!DOCTYPE html PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\"
\"http://www.w3.org/TR/html4/loose.dtd\">\r\n");
      out.write("<html>\r\n");
      out.write("<head>\r\n");
      out.write("<meta http-equiv=\"Content-Type\" content=\"text/html; charset=UTF-8\">\r\n");
      out.write("<title>스크립트릿 예제</title>\r\n");
      out.write("</head>\r\n");
      out.write("<body>\r\n");

          // 스크립트릿
          int number = 1;
          for (int i =0; i < 5; i++) {
                    out.println(number + " : ");
                    out.println(getName() + "<br>");
```

```
                    number++;
                }

            out.write('\r');
            out.write('\n');
            out.write("\r\n");
            out.write("</body>\r\n");
            out.write("</html>");
        } catch (Throwable t) {
            if (!(t instanceof SkipPageException)){
                out = _jspx_out;
                if (out != null && out.getBufferSize() != 0)
                    try { out.clearBuffer(); } catch (java.io.IOException e) {}
                if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
            }
        } finally {
            _jspxFactory.releasePageContext(_jspx_page_context);
        }
    }
}
```

● **그림 4-19** http://localhost:8080/home/scriptlet.jsp

이번 장에서는 JSP 웹 프로그래밍을 위한 기초 문법을 학습하였다. JSP 페이지 내에서 주석 처리 방법, page, include, taglib 지시자와 include, forward, plugin, useBean, setProperty, getProperty 액션 태그에 대해 알아보았다. 그리고 선언문과 표현식, 스크립트릿에 대해 학습하였다. 다음 장에서는 톰캣과 같은 JSP/서블릿 컨테이너가 기본적으로 제공하는 내장 객체에 대해 알아볼 것이다. 내장 객체란 JSP 내에서 선언하지 않아도 기본적으로 사용할 수 있는 객체를 의미한다.