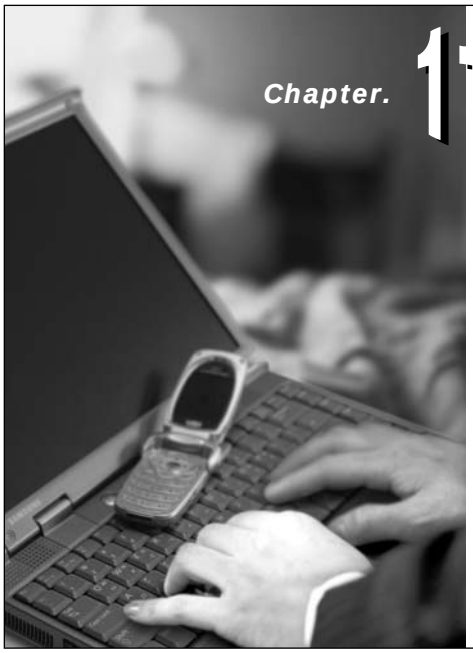


Chapter. 11 요약본

JSP 의 스크립트 기반 구현

- 스크립트 기반 구현의 개요
- 주석문(Comment)
- 지시문(Directive)
- 표현문, 수행문, 선언문

Global Inspiration
경기여성e-러닝센터

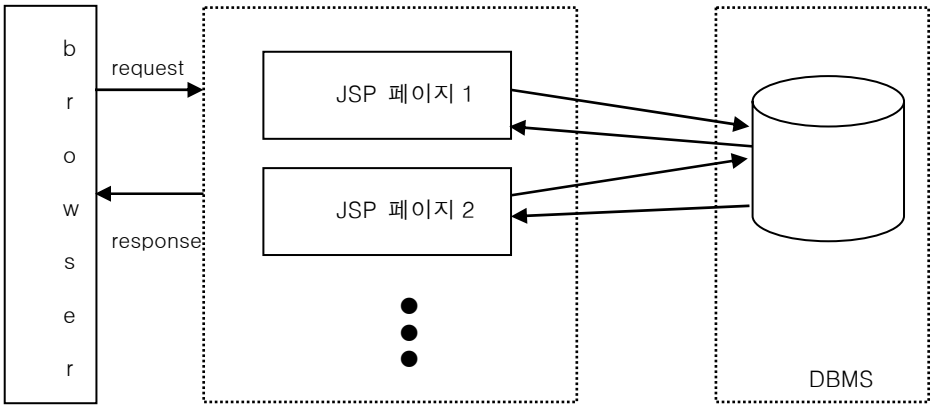
Chapter.11 : JSP 의 스크립트 기반 구현

1. 스크립트 기반 구현의 개요

모든 기능을 JSP 안에 Java 소스 코드를 포함하여 구현하는 방식으로서 프리젠테이션, 컨트롤, 비즈니스 로직 등이 모두 JSP 페이지 안에 구현된다. 즉, JSP 페이지 중심으로 어플리케이션을 구축하는 것으로 JSP 페이지 내에 스크립트 언어를 이용해서 프리젠테이션 영역뿐만 아니라 비즈니스 로직까지 포함하여 구현한다.

모든 어플리케이션 로직과 어떠한 결과를 응답할 것인지에 대한 모든 결정이 JSP 페이지 자체에서 하드 코딩 되는 방식이다.

다음은 페이지 중심 구현이 갖는 아키텍쳐이다.



만약 DBMS 에 연결이 필요한 JSP 페이지가 여러 개 존재한다면 각 JSP 페이지 마다 JDBC 프로그램 소스가 존재하게 된다. 또한 동적인 데이터는 같고 보여주기 위한 HTML 코드만 다르다 하더라도 동적인 데이터를 구하기 위한 소스 코드가 동일하게 있어야 한다.

[스크립트 기반 구현의 장점]

구조적인 관점에서 보면 상당히 단순하다. 변경되는 부분도 적고, 추상화 수준도 낮은 데다가 구성 계층도 적기 때문에 HTML 디자인과 자바 개발에 모두 익숙한 " 개인" 이나 소규모 개발팀이 빠른 시일 안에 동적인 Web 페이지와 JSP 어플리케이션을 만들 때 적합하다.

[스크립트 기반 구현의 단점]

페이지 중심 구현은 거의 초급 수준의 개발 형태를 띄는 경우가 많으며 어느 정도의 복잡도를 지닌 어플리케이션을 만드는 일에 대해서는 문제 개발과 유지 보수 측면에서 문제점을 가지고 있다. 어플리케이션을 이루고 있는 JSP 페이지들이 프리젠테이션과 비즈니스 로직, 컨트롤코드를 모두 가지고 있기 때문에 어플리케이션을 유지 보수하기가 힘들다. HTML 과 JSP 코드가 섞여 있어서 Web 페이지 디자이너와 자바 프로그래머 사이의 역할 구분이 희미해지고, 코드의 일부분을 수정 하였을 때 이와 연관된 다른 부분도 수정해야 하는 일이 많이 발생될 수 있다. 그리고 각 JSP 파일 안에 필요한 수행 코드들이 각각의 JSP 안에 하드코딩 되므로 여러 JSP 에 동일한 행 코드들이 중복되어 정의될 수 있다.

스크립트 기반 구현에서 Java 수행 코드를 포함하는 용도로 사용될 수 있는 대표적인 JSP 태그는 다음과 같다.

- 수행문

<% ..... %>

- 표현문

<%= ..... %>

- 선언문

<%! ..... %>

2. 주석문(Comment)

주석문은 JSP 페이지가 어떤 기능을 하며, 무슨 목적으로 작성되었는지 등의 요청 처리에 관계되지 않은 일반적인 정보를 JSP 페이지 내에 삽입하기 위해 사용된다. JSP 페이지에는 세가지 유형의 주석문을 사용할 수 있다.

다음은 JSP 페이지에서 사용 가능한 주석문의 종류와 구문에 대한 표이다.

종류	구문	주석으로 처리되는 시점
JSP 주석	<%-- --%>	Servlet 코드로 변환될 때
HTML 주석	<!-- -->	브라우저에 의해 응답이 파싱될 때
자바주석	//, /* */	Servlet 소스가 컴파일 될 때

주석문의 종류에 따라서 주석 처리를 하는 시점이 달라지므로 어느 부분에 주석이 보여야 할지를 파악한 후 선택한다.

1) HTML 주석문

HTML 이나 XML 에서 사용되는 표준 주석문 형식을 사용하며 JSP 응답의 일부로서 브라우저로 보내진다. JSP 컨테이너는 JSP 페이지를 작성하기 위한 문법적인 요소가 아닌 경우 일반 템플릿 데이터로 인식하므로 HTML 주석문도 예외가 아니다. JSP 페이지 내에서 HTML 주석문을 사용하면 브라우저에서 응답 정보를 화면에 뿌릴 때 주석으로 인식되어 화면에는 표현되지 않으며 브라우저 상에서 " 소스 보기" 메뉴를 통해서만 볼 수 있다.

이 주석문은 페이지 출력을 구성하는 일부분이기 때문에, 원한다면 이것도 동적 컨텐츠화 시킬 수 있다. 다음 예제에서와 같이 HTML 주석문에 JSP 태그를 사용하여 구현할 수 있고, 이 태그의 수행 결과가 포함된 주석문의 내용이 브라우저에 응답된다.

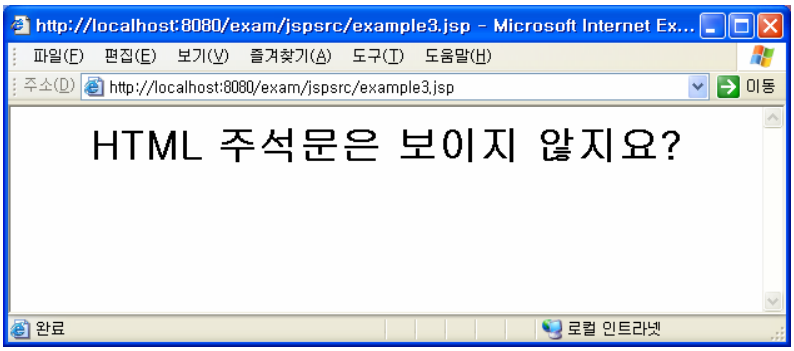
JSP 예제 (example3.jsp)

```
1  <%@ page contentType="text/html; charset=euc-kr" %>
2  <HTML>
3  <BODY>
4  <!-- HTML 주석 예제 -->
5  <!--  <%= application.getServerInfo() %> 에 의해 처리된 JSP 의
6  결과입니다.-->
7  <CENTER>
8  <H2> HTML 주석문은 보이지 않지요? </H2>
9  </CENTER>
10 </BODY>
11 </HTML>
```

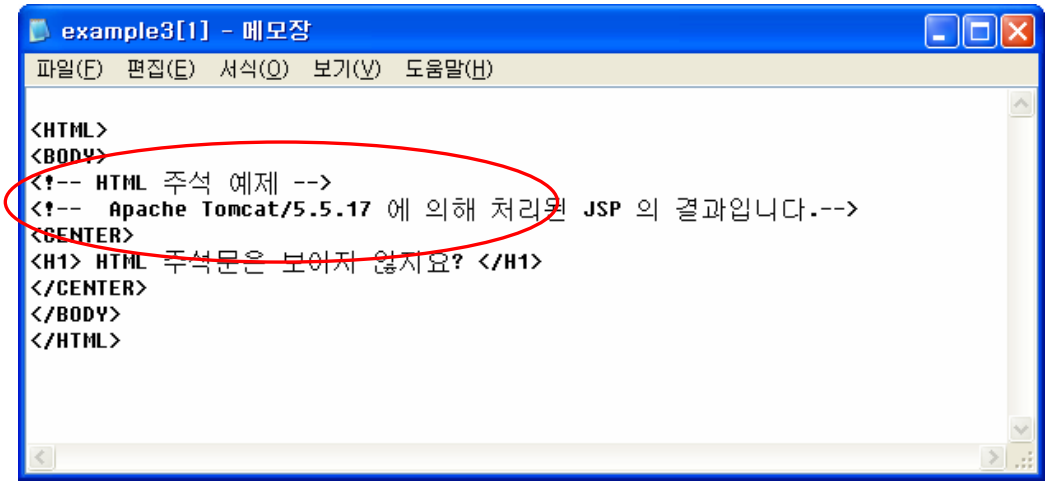
5 행 : HTML 주석문 안에 JSP 의 표현문 태그가 사용된 것을 볼 수 있다.
application 은 JSP 의 내장 객체 변수이다. 내장 객체 변수에 대해서는 13 장에서 자세히 학습한다. application.getServerInfo() 는 JSP 를 수행시키는 서버의 정보를 추출하게 되어 주석문의 내용이 되게 한다.

다음은 example3.jsp 를 브라우저로 요청한 결과 화면이다. HTML 주석문의 내용은 화면에

출력되지 않는다.



다음은 브라우저의 소스 보기 메뉴를 선택하여 출력한 내용이다. HTML 주석문 내용이 보여지고 있으며 application.getServerInfo()의 수행 결과가 포함된 것을 알 수 있다.



2) JSP 주석문

JSP 주석문은 JSP 컨테이너에 의해 주석으로 인식된다. JSP 컨테이너가 JSP 페이지를 구현 Servlet 소스로 변환할 때 JSP 주석문의 내용은 빼고 나머지 부분에 대해서만 자바 소스 코드를 생성한다. JSP 페이지를 요청한 후 실행 에러가 발생하면 페이지의 일부 코드를 제외하고 수행하기 위해 즉, 주로 디버깅을 하는데 사용한다.

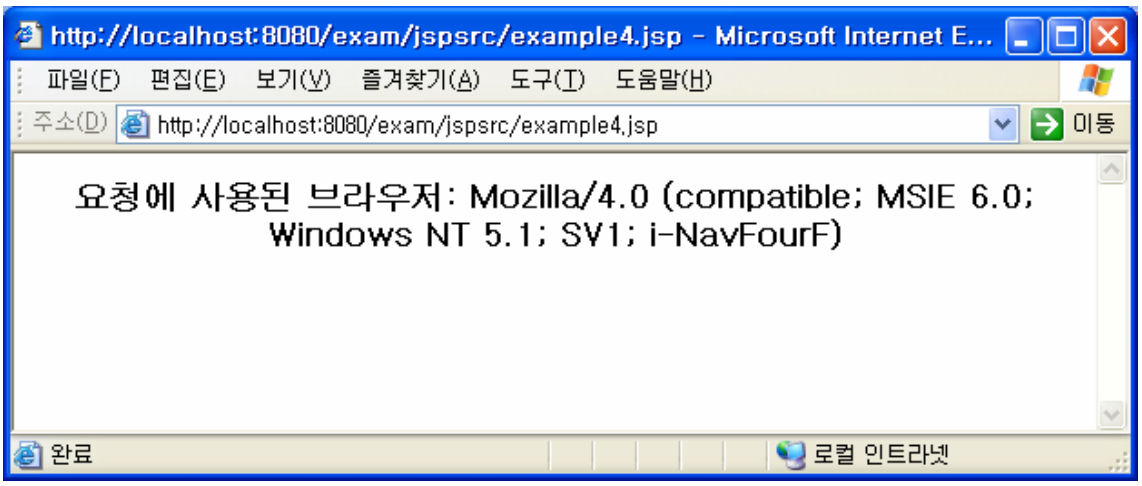
JSP 예제 (example4.jsp)	
1	<%@ page contentType="text/html; charset=euc-kr" %>
2	<HTML>
3	<BODY>
4	<CENTER>

```
5     <%-- <H3>요청 방식: <%= request.getMethod() %></H3> --%>
6     <H3>요청에 사용된 브라우저:
7           <%= request.getHeader("User-Agent") %></H3>
8     </CENTER>
9     </BODY>
10  </HTML>
```

4 행 : JSP 주선문 안에 구현된 내용은 Servlet 소스로 변환되지 않으므로 수행도 되지 않고 클라이언트로 결과가 출력되지도 않는다.

7 행 : 이 행에서 사용된 request 는 JSP 의 내장 객체 변수이다. 내장 객체 변수에 대해서는 13 장에서 자세히 학습한다. request.getHeader("User-Agent") 는 요청을 보내온 브라우저에 대한 정보를 추출할 때 사용한다.

다음은 example4.jsp 를 브라우저로 요청한 결과 화면이다. JSP 주석문의 내용은 화면에 출력되지 않는다.



3) Java 주석문

자바 주석문은 JSP 페이지내에 스크립팅 언어로 자바를 사용하는 경우 사용할 수 있는 주석문이다. 이 주석문은 스크립팅 언어에 따라 달라지는 부분으로 해당 스크립트 언어에서 제공하는 주석 처리 방식을 사용할 수 있는 것이다. 그러므로 이 주석문은 자바의 스크립트 코드가 들어갈 수 있는 스크립트릿과 표현식, 선언문(Declaration)등에서만 사용할 수 있다. 자바에서는 /* 와 */ 그리고 // 을 주석문을 위한 구분자로 사용한다.

JSP 예제 (example5.jsp)

```
1     <%@ page contentType="text/html; charset=EUC-KR" %>
```

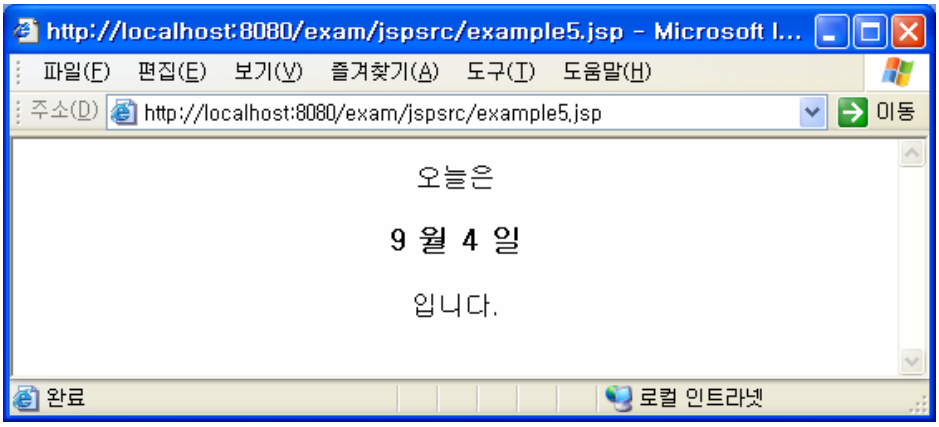
```
2     <HTML>
3     <BODY>
4     <CENTER>
5     오늘은
6     <%
7     /* java.util.Date date = new java.util.Date() ; */
8     // Calendar 객체 생성
9     java.util.Calendar cal = java.util.Calendar.getInstance();
10    %>
11    <H3><%=cal.get(java.util.Calendar.MONTH)+1%> 월
12    <%=cal.get(java.util.Calendar.DATE) %> 일 </H3>
13    입니다.
14    </CENTER>
15    </BODY>
16  </HTML>
```

7 행 : JSP 의 수행문 태그안에 Java 의 단일행 주석문이 사용되었다.

8 행 : JSP 의 수행문 태그안에 Java 의 다중행 주석문이 사용되었다.

이렇게 Java 주석문은 스크립트 태그 안에 사용해야만 하며 주로 Java 소스 코드에 대한 설명 추가한다거나 코드를 주석 처리하고자 할 때 사용한다.

다음은 example3.jsp 를 브라우저로 요청한 결과 화면이다. HTML 주석문의 내용은 화면에 출력되지 않는다.



3. 지시문(Directive)

1) 지시문의 종류

지시문은 JSP 페이지가 Servlet 으로 변환될 때 적용될 수 있는 다양한 속성을 지정하거나 JSP 파일 내에 다른 파일의 내용을 포함하고자 하는 경우 그리고 커스텀 태그 라이브러리를 정의하는 기능에 사용된다. 지시문은 JSP 1.2 까지는 3 개가 제공되었지만 JSP 2.0 부터는 3 개가 더 추가되어 총 6 개가 사용될 수 있다. 종류로는 page 지시문, include 지시문 그리고 taglib 지시문, JSP 2.0 에서 추가된 attribute 지시문, tag 지시문 그리고 variable 지시문이 있다. 그러나 기초 과정에서는 다음 2 개의 지시문만 학습한다.

<%@page {attr = value ..} %> → page 지시문
<%@include {attr = value ..} %> → include 지시문

지시문은 클라이언트로의 어떠한 출력을 처리하는 구문이 아니며 주로 JSP 컨테이너에게 정보를 전달하는 기능을 처리다.

2) page 지시문

JSP 페이지가 Servlet 으로 변환될 때 적용될 수 있는 다양한 속성을 지정 경우 활용된다. page 지시문에 지정되는 속성은 name=value 의 형식으로써 얼마든지 여러 개 올 수 있다. 다음과 같은 속성들이 사용될 수 있으며 import 속성의 경우에 한해서 중복지정이 가능하다.

<%@ page
[language="java"]
[extends="package.class"]
[import="{package.class | package.*}, ..."]
[session="true|false"]
[buffer="none|8kb|sizekb"]
[autoFlush="true|false"]
[isThreadSafe="true|false"]
[info="text"]
[errorPage="relativeURL"]
[contentType="mimeType [; charset=characterSet]" |
"text/html ; charset=ISO-8859-1"]

[isErrorPage="true|false"]
[pageEncoding="characterSet | ISO-8859-1"]
[isELIgnored="true|false"]
%>

[page 지시문이 사용된 예]

<%@ page import="java.util.*, java.lang.*" %>
<%@ page session="false" isErrorPage="true" %>
<%@ page errorPage="error.jsp" %>

[page 지시문의 속성]

위에서 제시되지 않은 속성을 지정한 경우이거나, 잘못된 속성지정은 JSP 소스 코드를 파싱하여 Servlet 소스 코드를 만들어 내는 과정에서 오류를 발생하게 된다.

- language 속성

페이지에서 사용되는 스크립트 언어를 지정할 때 사용된다. 이 항목이 생략되면 JSP 컨테이너는 기본적으로 Java 언어로 간주한다. 대부분의 JSP 컨테이너들은 Java 언어를 기본적으로 지원하는데 JSP 컨테이너 제품에 따라서는 다른 언어를 지원할 수도 있다.

- extends 속성

JSP 컨테이너에 의해 JSP 페이지가 파싱되어 생성되는 Servlet 클래스가 상속할 부모 클래스를 지정한다. 그런데 이 속성은 거의 사용되지 않는다. 일반적으로 JSP 컨테이너가 부모 클래스를 이미 정해져 있는 클래스로 지정하도록 한다. 일반적으로 javax.servlet.jsp.HttpJspPage 또는 javax.servlet.jsp.JspPage 를 상속하여 구현한 클래스를 상속받게 된다.

- import 속성

extends 속성과는 다르게 JSP 페이지에서 자주 사용되는 속성으로서 JSP 페이지에서 사용되는 Java 클래스들에 대한 패키지들의 import 를 지정한다. 사용하는 클래스마다 패키지명을 붙여서 표현하는 것도 가능하지만 여러 개라면 import 속성을 사용하여 패키지 단위로 지정하는 것이 더 간단하다. import 해야 하는 패키지 또는 클래스가 여러 개인 경우에는 컴마(,) 연산자로 구분하여 지정한다. java.lang 패키지, javax.servlet 패키지, javax.servlet.http 패키지 그리고 javax.servlet.jsp 패키지는 자동적으로 import 된다.

- session 속성

주어진 JSP 페이지의 세션(session) 관리 처리 여부를 지정하고자 하는 경우 사용된다. true 또는 false 값을 지정한다. 기본값은 true 이기 때문에 모든 페이지들은 자동적으로 세션을 생성하게 된다. 특별히 세션 처리를 하지 않으려는 경우에는 이 속성의 값을 false 로 지정한다.

- buffer 속성

JSP 페이지의 버퍼 출력을 조정한다. 버퍼 출력를 하지 않고 HTTP 응답을 클라이언트로 바로 보내려면 이 속성의 값을 none 으로 할당해 준다. buffer 속성의 기본값은 8kb 로서 예상되는 버퍼 크기의 최소값으로 설정한다. 일정 크기의 버퍼를 미리 잡아 두는 것은 클라이언트로 부터의 요청이 들어올 때마다 JSP 컨테이너가 출력 버퍼를 생성해야 하는 부담이 없어지고 수행 성능이 높아지게 된다.

- autoFlush 속성

이 속성도 버퍼 출력과 연관되는 속성으로서 출력 버퍼가 Full 이 되었을 때의 동작을 설정한다. 이 속성이 true(기본값)로 설정되면 출력 버퍼가 자동적으로 비워(flush)지며 현재의 출력 버퍼의 내용이 클라이언트로 전송된다. autoFlush 속성을 false 로 설정하면 출력 버퍼가 Full 이 되어도 버퍼를 비우지 않으며 예외를 발생시키게 된다.

- isThreadSafe 속성

JSP 가 Servlet 으로 변환될 때, SingleThreadModel 의 추가 상속 여부를 결정한다. 이 속성의 값이 false 이면 SingleThreadModel 을 추가 상속하게 된다.

- info 속성

페이지 작성자가 페이지의 역할을 설명하는 도큐먼트 문자열을 추가할 수 있다. 일반적으로 작성자에 대한 정보와 버전, 저작권 등에 대한 정보를 정의한다.

- errorPage 속성

JSP 수행하는 도중 오류가 발생하였을 때 대신 처리될 파일을 설정하는 속성이다. 대신 처리될 파일의 상대 URL 을 지정한다.

- contentType 속성

JSP 페이지가 생성하는 응답의 MIME(Multipurpose Internet Mail Extension) 타입을

지정한다. 클라이언트로 응답되는 JSP 파일의 출력 결과가 어떠한 형식의 문서인지를 알려준다. 대부분의 MIME 타입은 " text/html", " text/xml", " text/plain" 등이 쓰인다. contentType 속성은 JSP 페이지에서 응답될 문자 집합에 대한 정보를 지정하는 용도로도 사용된다. MIME 타입 뒤에 세미콜론(;)을 추가하고 charset= 을 추가한 다음 문자 집합 이름을 붙여 주면 된다.

- isErrorPage 속성

해당 페이지가 오류 페이지로 동작할 수 있는 여부를 결정한다. 이 속성의 값이 true 라는 것은 현재 페이지에서 exception 내장객체 변수를 사용할 수 있는 결과가 된다.

- pageEncoding 속성

응답되는 결과의 문자 집합에 대한 정보를 지정하는 용도로 사용된다. 생략했을 때의 디폴트 값은 " ISO-8859-1" 이므로 출력 결과에 한글이 포함되어 있는 경우에는 " EUC-KR" 설정한다. 물론 contentType 속성에 " text/html; charset=EUC-KR" 값을 추가하여 해결할 수도 있다.

- isELIgnored 속성

JSP 에서 EL() 표현문의 사용 여부를 결정한다. true 이면 JSP 파일 내에 구현된 EL 표현문(${...}) 이 컨테이너에 의해 무시된다.

3) include 지시문

JSP 문장들이 번역(translate)될 때 다른 코드 또는 텍스트를 포함시키고자 하는 경우 사용되는 지시문이다. 다음과 같은 구문을 사용하는 지시문이다.

<%@ include file="relativeURL" %>

include 될 파일은 file 이라는 속성에 상대 URL 을 지정하며 이 파일의 내용이 page 지시문 위치에 대체되는 결과가 된다. 상대 URL 이란 동일한 Web Application 에 존재하는 파일에 대한 URL 을 의미하므로 프로토콜, 서버주소, 포트번호 모두 생략하고 대상 파일의 패스만을 지정하는데 제일 앞에 Wab Application 을 알리는 패스(컨텍스트 패스)를 생략하여야 한다. 다음과 같은 URL 을 지정할 수 있다.

<%@ include file="/test.html" %>
<%@ include file="/jspsrc/example1.jsp" %>
<%@ include file="example1.jspf" %>

include 지시문의 사용 회수에는 제한이 없으며 중첩하여 지정하는 것도 가능하다. 즉, 한 JSP 페이지에서 다른 JSP 페이지를 포함하고 포함된 JSP 페이지가 또 다른 JSP 페이지를 포함하는 처리가 가능하다.

다른 파일 안에 include 되는 용도로만 사용되는 파일은 확장자를 .jspf 로 지정하는 것이 좋다. jspf 는 JSP Fragment 의 약어로 다른 JSP 의 일부분이다라는 의미로 해석할 수 있으며 확장자만 보더라도 완전한 JSP 파일이 아니며 include 용도의 파일이다는 것을 알 수 있기 때문이다.

include 지시문으로 포함되는 파일의 내용이 수정되면 JSP 가 요청될 때 다시 변환이 발생된다. 즉 A.jsp 에서 B.jsp 를 include 지시문으로 처리한 경우 A.jsp 를 부라우저로 요청하면 A.jsp 안에 B.jsp 의 내용을 포함하여 Servlet 으로의 변환이 발생되고 수행된다. 이 후 A.jsp 또는 B.jsp 의 내용이 수정되면 A.jsp 를 브라우저로 요청하였을 때 A.jsp 안에 B.jsp 의 내용이 포함되어 Servlet 으로 변환하는 과정이 다시 처리된다.

(5) 지시문 관련 예제들

page 지시문의 일부 속성(import, errorPage, isErrorPage)과 include 지시문 관련 예제들을 점검하고 수행 결과를 확인해 본다.

[예제 1]

page 지시문에서 import 속성을 이용하여 java.util 패키지를 import 처리 하고 있는 예제이다.

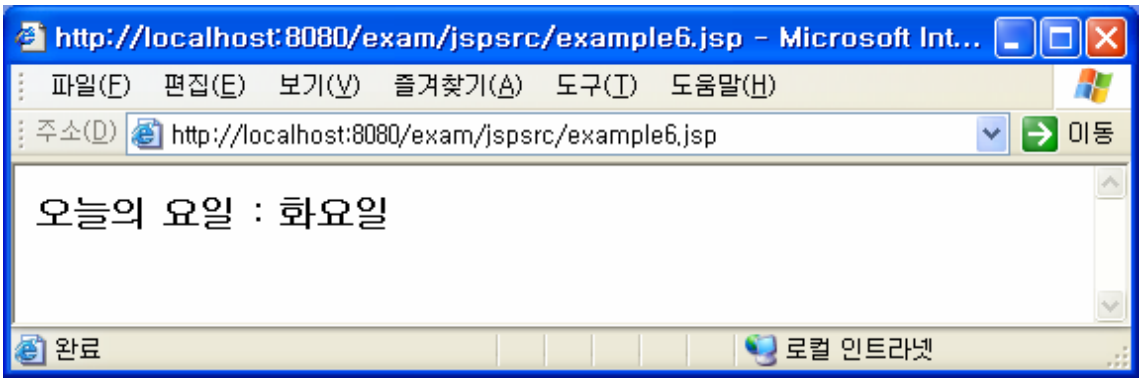
JSP 예제 (example6.jsp)

```
1  <%@ page import="java.util.*" contentType="text/html; charset=EUC-KR" %>
2  <HTML>
3  <BODY>
4  <%
5  String[] days = new String[] {"일요일", "월요일", "화요일", "수요일",
6                                "목요일", "금요일", "토요일"};
7  Calendar calendar = Calendar.getInstance();
8  Date trialTime = new Date();
9  calendar.setTime(trialTime);
10
```

```
11      int day = calendar.get(Calendar.DAY_OF_WEEK);
12      %>
13      <H3>오늘의 요일 : <%= days[day] %></H3>
14      </BODY>
    </HTML>
```

1 행 : page 지시문 태그의 import 속성에 java.util.* 를 지정했으므로 java.util 패키지의 어떠한 API 도 JSP 안에서 사용할 수 있게 된다.

다음은 example6.jsp 를 브라우저로 요청한 결과 화면이다.



[예제 2]

브라우저로부터 2 개의 숫자 값에 대한 Query 문자열을 전달받아 나눗셈을 처리한 결과를 브라우저로 응답하는 예제이다.

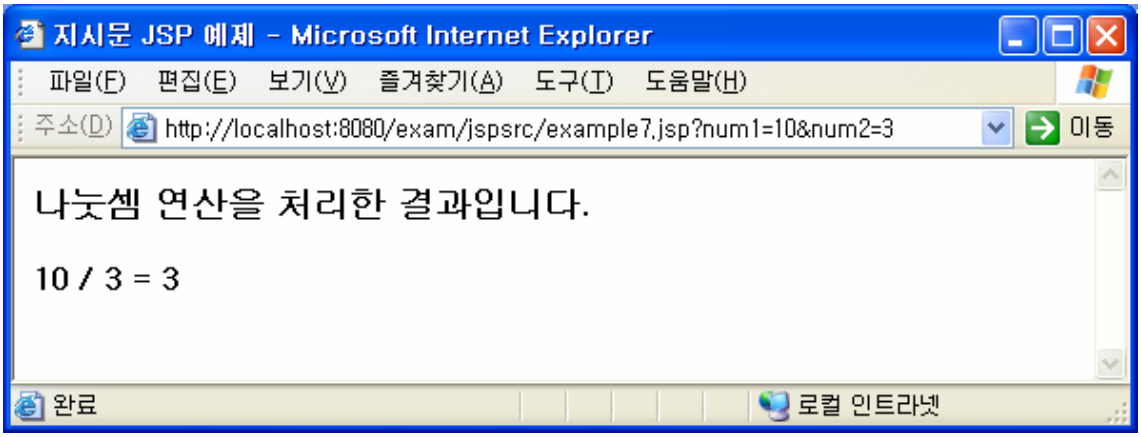
JSP 예제 (example7.jsp)

```
1       <%@ page contentType="text/html; charset=EUC-KR" %>
2       <HTML>
3       <HEAD>
4       <TITLE>지시문 JSP 예제 </TITLE>
5       </HEAD>
6       <BODY>
7       <H3>나눗셈 연산을 처리한 결과입니다.</H3>
8       <%
9       int su1   = Integer.parseInt(request.getParameter("num1"));
10      int su2   = Integer.parseInt(request.getParameter("num2"));
11      %>
12      <H4> <%= su1 %> / <%= su2 %> = <%= su1/su2 %> </h4>
13
```

14	</BODY> </HTML>

9-10 행 : request.getParameter(“ num1”) 은 Query 문자열을 추출하는 기능을 수행한다. request 는 내장 객체 변수로서 HttpServletRequest 객체를 참조하게 된다. Integer.parseInt() 메서드의 기능은 숫자 문자열을 정수형 숫자로 변환하는 기능을 수행한다. Query 문자열로 숫자가 전달될 때에는 숫자 문자열로 전달되므로 사용하기 전에 이렇게 숫자로 변환하는 작업을 처리해야 한다.

다음은 example7.jsp 가 정상적으로 수행된 결과 화면이다. 200 을 20 으로 나눈 결과가 출력되었다.



다음은 example7.jsp 을 브라우저로 요청했을 때 오류가 발생한 결과 화면이다. Query 문자열을 전달하지 않고 요청하면 Integer.parseInt() 메서드 호출시 null 을 가지고 수행하게 되므로 NumberFormatException 이 발생하게 되어 다음과 같은 오류 화면이 출력된다. 이 화면은 WAS 에 의해 출력되는 내용이다.

[예제 3]

page 지시문에서 errorPage 속성을 이용하여 실행 시 오류가 발생되면 error.jsp 의 수행결과를 대신 응답하고 있는 예제이다. error.jsp 에서는 exception 내장 객체 변수를 사용하기 위하여 isErrorPage 속성을 true 로 설정하고 있다.

JSP 예제 (example8.jsp)

```
1  <%@ page contentType=" text/html; charset=EUC-KR"
2  errorPage=" error.jsp" %>
3  <HTML>
4  <HEAD>
5  <TITLE>지시문 JSP 예제 </TITLE>
6  </HEAD>
7  <BODY>
8  <H3>나눗셈 연산을 처리한 결과입니다.</H3>
9  <%
10 int su1  = Integer.parseInt(request.getParameter(" num1" ));
11 int su2  = Integer.parseInt(request.getParameter(" num2" ));
12 %>
13 <H4> <%= su1 %> / <%= su2 %> = <%= su1/su2 %> </h4>
14 </BODY>
</HTML>
```

1 행 : page 지시문 태그의 errorPage 속성에 " error.jsp" 를 지정하여 실행 도중에 실행 오류가 발생하면 error.jsp 의 수행 결과를 대신 응답하도록 하고 있다.
이 때 errorPage 속성에 사용 가능한 자원의 종류로는 제한이 없다.

JSP 예제 (error.jsp)

```
1  <%@ page contentType="text/html; charset=EUC-KR"
2  isErrorPage="true" %>
3  <HTML>
4  <HEAD><TITLE> 오류 발생 </TITLE></HEAD>
5  <BODY>
6  <H3><%= request.getAttribute("javax.servlet.error.request_uri") %>에서
7  오류가 발생하였습니다.</H3>
8  발생된 오류의 종류 :
9  <%= exception.getClass().getName() %><BR>
10 발생 원인 :
11 <%= exception.getMessage() %><BR>
12 </BODY>
</HTML>
```

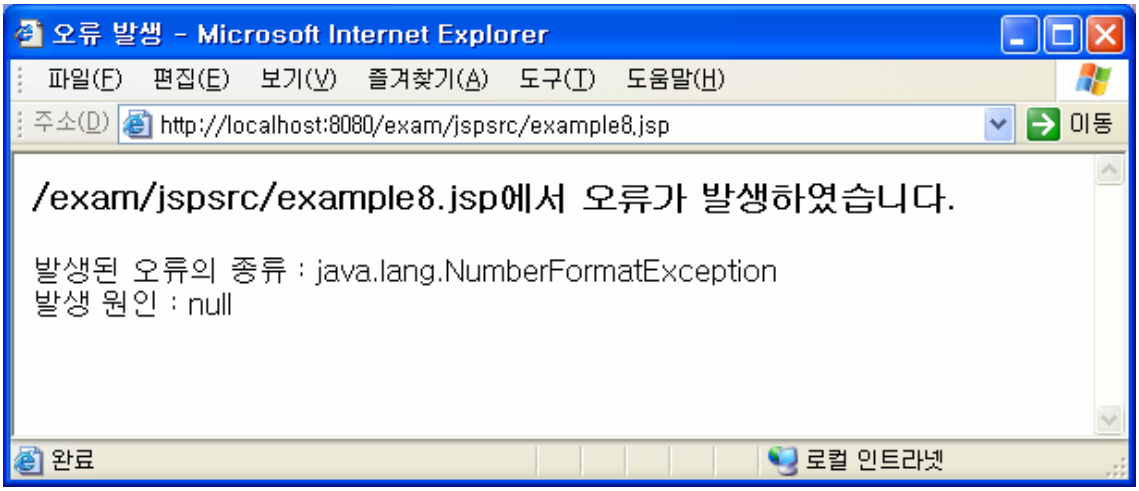
1 행 : page 지시문 태그의 isErrorPage 속성에 true 를 지정하여 exception 내장 객체 변수를 사용할 수 있도록 속성을 설정하고 있다. exception 내장 객체 변수는 실행 오류발생시 생성되는 Exception 객체를 참조하게 되는 변수이다.

5 행 : request.getAttribute("javax.servlet.error.request_uri") 호출은 오류가 발생된 대상에 대한 URI 정보를 추출하는 메서드 호출이다.

8 행 : exception.getClass().getName()은 발생된 오류의 종류를 추출하는 메서드 호출이다.

10 행 : exception.getMessage()는 오류가 발생될 때 초기화된 오류 메시지를 추출하는 메서드 호출이다.

다음은 example8.jsp 을 브라우저로 요청했을 때 오류가 발생한 결과 화면이다. Query 문자열을 전달하지 않고 요청하면 Integer.parseInt() 메서드 호출시 null 을 가지고 수행하게 되므로 NumberFormatException 이 발생하게 되어 다음과 같은 오류 화면이 출력된다. 이 화면은 error.jsp 의 수행 결과가 대신 응답된 것이다.



[예제 4]

include 지시문을 처리하고 있는 예제이다. copyright.jspf 의 내용을 포함하여 Servlet 으로 변환하고 처리된 결과를 클라이언트로 응답한다.

JSP 예제 (example9.jsp)	
1	<%@ page contentType="text/html; charset=euc-kr" %>
2	<HTML>
3	<BODY>
4	<CENTER>

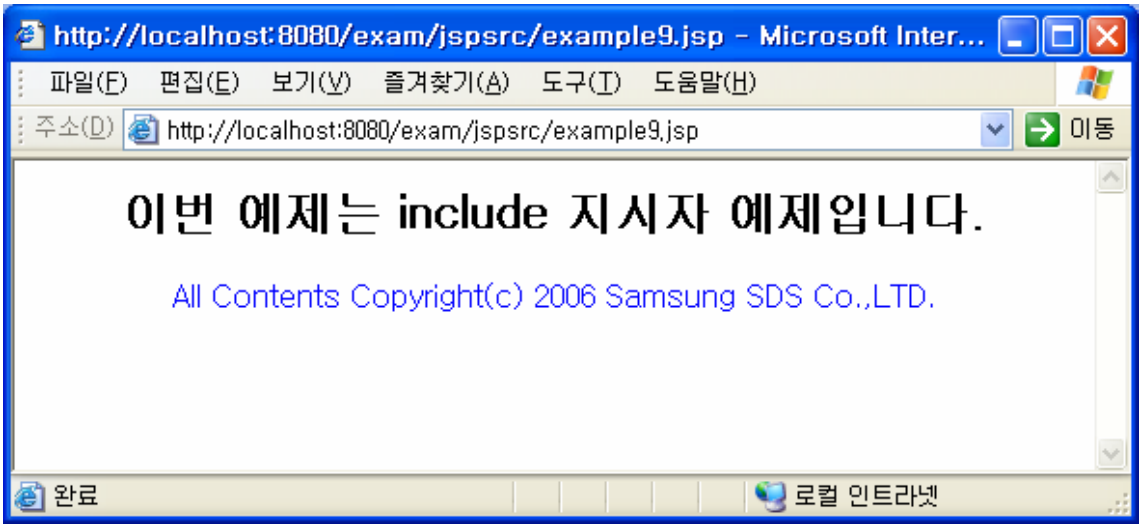
```
5     <H2> 이번 예제는 include 지시자 예제입니다.  </H2>
6     <%@ include file="copyright.jspf" %>
7     </CENTER>
8     </BODY>
9  </HTML>
```

6 행 : include 지시자 태그를 사용하여 이 위치에 copyright.jspf 의 내용을 포함 하여 수행하도록 구현하고 있다.

include 용 JSP (copyright.jspf)

```
1     <FONT color="blue">
2     All Contents Copyright(c) 2006 Samsung SDS Co.,LTD.
3  </FONT>
```

다음은 example9.jsp 가 수행된 결과 화면이다. copyright.jspf 의 내용이 포함되어(파란색 부분) 처리된 결과가 브라우저로 출력된 것을 볼 수 있다. example8.jsp 가 변환된 example8_jsp.java 소스의 내용도 점검하여 cipyright.jspf 의 내용이 포함되어 변환된 것을 확인해 본다.



4. 표현문, 수행문, 선언문

1) 표현문(Expression)과 EL 표현문

JSP 표현문은 <%= .... %>로 구현되는 스크립트 기반 태그로서 주어진 연산식의 연산 결과를 정의된 위치에 표현하도록 하는 JSP 태그이다. EL 표현문은 JSP 2.0 부터 정식으로 추가된 JSP 표현 언어(EL)를 사용하는 표현문으로서 JSP 에서 자주 활용되는 API 의 처리 결과를 쉽게 표현하고 JavaBeans 컴포넌트에 저장된 데이터를 쉽게 접근할 수 있도록 지원하는 태그이다. 여기에서는 표현문만 학습하고 EL 표현문은 심화 과정에서 학습한다.

이미 학습한 예제들 중에서도 표현문을 활용하고 있는 예제도 있어서 그렇게 생소하지는 않겠지만 먼저 표현문의 사용 형식부터 점검해 보면 다음과 같다.

<%= expression %>

표현문에서 중요한 것은 표현문 안에는 반드시 식(expression)이 정의되어야 한다는 것이다. 하나의 완전한 수행 문장이 아니고 산술식, 호출식, 조건식 등 식이 와야 한다. 이유는 표현문안에 구현되는 내용은 Servlet 으로 변환될 때 out.print() 메서드의 매개변수가 되기 때문이다.

메서드를 호출할 때 지정 가능한 매개변수로는 변수, 상수, 연산식, 호출식등이 올 수 있는것과 같이 표현문 안에는 반드시 매개변수로 가능한 요소만을 지정할 수 있다.

다음은 표현문이 사용된 예이다.

<%= new java.util.Date().toString() %>
<%= 100 + 200 %>
<%= (hours < 12) ? “AM” : “PM” %>
<%= sum/su %>

2) 수행문(Scriplet)

수행문 태그는 JSP 가 요청될 때마다 수행되는 Java 코드를 추가하고자 할 때 사용되는 코드의 블록이다. <% 와 %> 로 구성되며 이 안에는 멤버(변수 또는 메서드) 정의와 관련된 내용을 제외하고 어떠한 수행 코드든(예를 들어 제어문, 오류 처리 구문등) 올 수 있다. 수행문 영역에 정의되어 있는 코드들은 클라이언트로 부터 요청될 때마다 수행된다. 다음은

수행문의 구현 형식이다.

<% code fragment %>

JSP 에 구현되는 Java 수행 코드 중에서 멤버 변수 선언이나 메서드 정의는 선언문을 사용하고 연산의 결과를 해당 위치에 표현하고자 하는 경우에는 표현문을 사용한다, 그외의 Java 수행 코드들은 모두 수행문을 사용하여 처리할 수 있다. 다음은 수행문을 사용하여 현재 시간이 오전인지 오후인지에 따라서 " Good Morning" 또는 " Good Afternoon"을 출력하는 부분 예제이다.

```
<% if (Calendar.getInstance().get(Calendar.AM_PM) == Calendar.AM) { %>
        Good Morinng
<% } else { %>
        Good Afternoon
<% } %>
```

선언문과 표현문은 용도와 사용될 수 있는 문장의 종류가 결정되어 있는 반면 수행문(scriptlet)은 어떠한 종류의 코드든 구문적으로 유효한 수행 코드를 정의할 수 있는 일반적인 용도의 스크립트 기반의 태그이다.

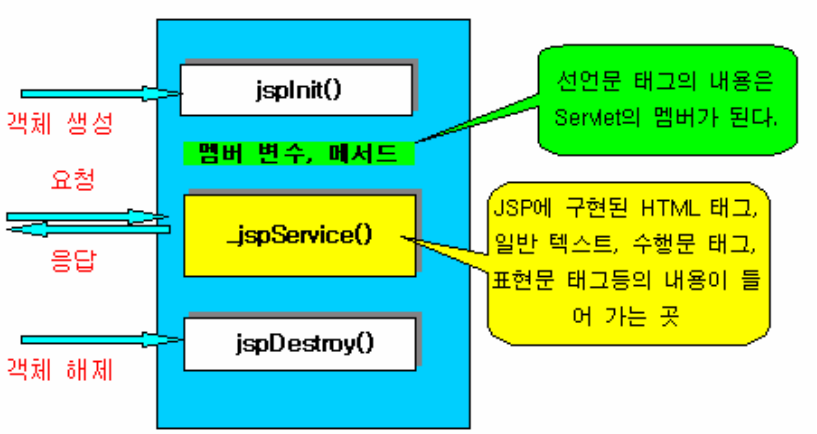
수행문에도 변수를 선언할 수 있는데 수행문에 선언된 변수는 _jspService()의 지역 변수가 된다.

3) 선언문(Declaration)

선언문 태그는 JSP 의 멤버 변수를 선언하거나 메서드를 정의하고자 하는 경우에 사용되는 특별한 목적의 태그로서 선언문의 내용은 Servlet 의 멤버가 된다. 다음은 수행문의 구현 형식이다.

<%! declaration; [declaration;]+ ... %>

대부분의 스크립트 기반의 태그들에 구현된 코드들은 _jspService() 메서드의 수행코드가 되지만 유일하게 선언문의 내용은 Servlet 의 멤버가 되는 결과가 된다. 다음은 이것을 소개하고 있는 그림이다.



다음은 선언문을 사용하여 멤버 변수와 메서드를 정의하는 예이다.

```
<%! private int su = 100; %>
<%! int total = 0; float increment = 3.5; %>
<%! public static int counter = 0 %>
<%! public long fact(long x) {
        if (x == 0) return 1;
        else return x * fact(x-1);
} %>
```

JSP 의 수행 흐름에서 최초에 한 번 호출(초기화)되고 마지막에 한 번 호출(객체 해제)되는 jspInit()과 jspDestroy()를 구현하고자 하는 경우에도 선언문을 사용한다.

```
<%!
        public void jspInit() {
  // 초기화 코드
  }
  public void jspDestroy() {
  // 객체 해제 코드
  }
%>
```

다음은 표현문, 수행문, 선언문에 대하여 예제이다. 각각의 태그가 어떠한 역할을 구현하고

있는지 점검해 본다.

JSP 예제 (example10.jsp)

```
1  <%@ page contentType="text/html; charset=EUC-KR"
   import="java.util.*"%>
2  <HTML>
3  <%!
4  public String getUser(String user){
5    try {
6      if ( user == null)
7        return "손님";
8      else
9        return new String(user.getBytes("8859_1"), "KSC5601");
10   } catch (Exception e) {
11       return user;
12   }
13 }
14 %>
15 <BODY>
16 <%
17 String user = getUser(request.getParameter("name")) ;
18 Calendar calendar = Calendar.getInstance();
19 Date trialTime = new Date();
20 calendar.setTime(trialTime);
21 %>
22 <H3><%= user %> 님!</H3>
23 <H4>오늘은 <%= calendar.get(Calendar.YEAR) %> 에서
24 <%= calendar.get(Calendar.DAY_OF_YEAR) %> 일 되는 날입니다.</H4>
25 <% if (Calendar.getInstance().get(Calendar.AM_PM) == Calendar.AM) { %>
26 <h3>오늘 하루 활기차게 시작하세요!</h3>
27 <% } else { %>
28 <h3>오늘 하루 마무리 잘 하세요!</h3>
29 <% } %>
30 </BODY>
```

	</HTML>

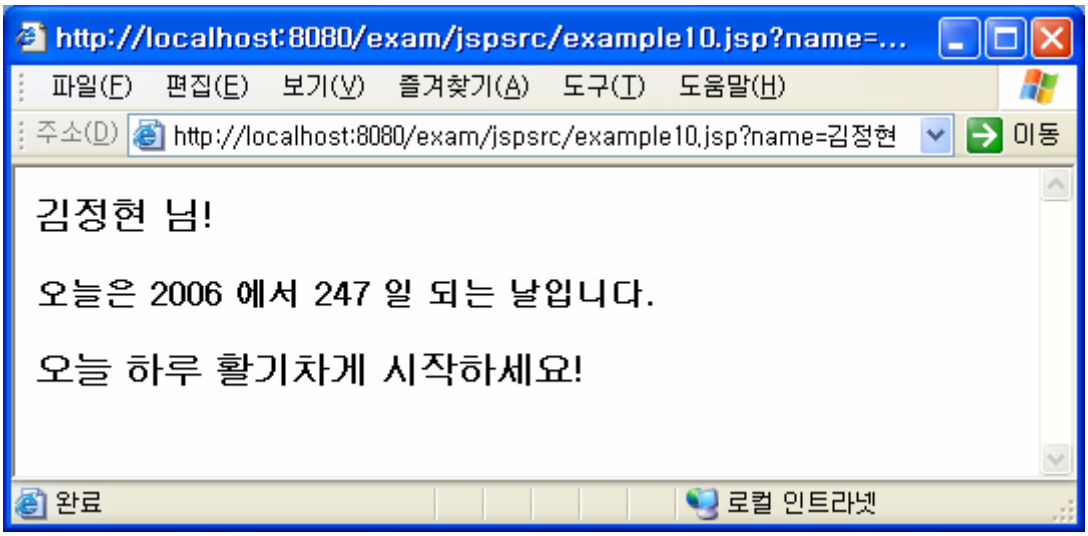
3-14 행 : 선언문 태그를 사용하여 getUser() 라는 메서드를 정의하고 있다. getUser() 는 Query 문자열의 존재 여부에 따라 적절한 user 정보를 설정하는 기능을 수행한다.

16-21 행 : 수행문 태그를 사용하여 브라우저로 부터 전송된 Query 문자열을 추출하는 기능, Calender 객체 생성과 Date 객체 생성 등을 수행하도록 구현하였다.

22-24 행 : 표현문 태그를 사용하여 user 정보, 년도와 날짜 정보등을 브라우저에 출력하는 기능을 구현하였다.

25 행 : 요청 받은 시점을 기분으로 오전인지 오후인지를 구분하여 응답이 달리 처리되도록 제어문을 사용하여 구현하였다.

다음은 Query 문자열로 user=김정현을 지정하여 example10.jsp 를 요청한 결과 화면이다.



다음은 Query 문자열을 생략하고 example10.jsp 를 요청한 결과 화면이다.